Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Adquisición de datos

Carpeta de Trabajo

M.C. Maestro: Antonio Cayetano Lozano García

Semestre: Ene – Jun 2022

Hora: LMV M3 Grupo: 001

Matrícula	Nombre	Carrera
1902373	Samuel Ernesto Torres Cosío	IMTC
1901410	Julio de Jesús Moreno Sánchez	IMTC
1894698	Christopher Russ Acuña Rodríguez	IMTC
1895481	Alfonso Emiliano Sandoval Juárez	IMTC
1820718	Cesar Alonso Cantú Espinosa	IMTC
1827011	Emilio González Rojas	IMTC

Lugar: Pedro de Alba SN, Niños Héroes, Ciudad Universitaria, San Nicolás de los Garza, N.L.

Fecha: 27/05/2022

Índice

Índice 2
Diseño: 4
Diseño de Programa 4
Descripción de librerías 5
Estructura lógica del programa 7
Diseño del Hardware: 10
Arduino 10
Acondicionamiento 11
Planta 11
Cálculos: 12
Señales 12
Escalón 13
Rampa 14
Seno 15
Acondicionamiento 16
PCB 17
Impresión de los códigos fuente de los programas 20
Software 20
Código Arduino 20
Código Python 25
(IO-DAQS.py) 25
(WorkingArea.py) 32
Firmware 35
Software de aplicación específica 36
Software de la etapa de comunicación 36
Sección de implementación 36
Sección de experimentación 39
Sección de resultados obtenidos 41
Seccion de analisis de resultado 42
Apéndices 44
Bibliografía: 44

Tabla de figuras

Ilustración 1: Esquema de Proyecto 4
Ilustración 2: Diseño del programa (frontend y backend) 5
Ilustración 3: Diagrama de flujo del programa (parte 1) 7
Ilustración 4: Diagrama de flujo del programa (parte 2) 8
Ilustración 5: Diagrama de flujo del programa (parte 3) 9
Ilustración 6: Placa Arduino UNO R3 10
Ilustración 7: Convertidor Digital Analógico R-2R 11
Ilustración 8: Sistema a experimentar (respuesta transitoria) 12
Ilustración 9: Gráfica señal escalón (Desmos) 14
Ilustración 10: Gráfica señal rampa (Desmos) 15
Ilustración 11: Gráfica señal seno (Desmos) 15
Ilustración 12: DAC 16
Ilustración 13: Ecuaciones del comportamiento del DAC 16
Ilustración 14: Circuito implementado en 2015 (referencia física) 17
Ilustración 15: Implementación del circuito 17
Ilustración 16: Diseño PCB integral ya en físico (primer diseño) 18
Ilustración 17: Circuito en Proteus (segundo diseño- propuesta) 19
Ilustración 18: Layout del PCB (segundo diseño- propuesta) 19
Ilustración 19: Vista 3D circuito en proteus (segundo diseño-propuesta) 20
Ilustración 20: Equipo de trabajo en laboratorio de Mecatrónica (FIME) 36
Ilustración 21: Reunión en laboratorio de DAQ, mediciones con el circuito 37
Ilustración 22: BK PRECISION 1761 DC POWER SUPPLY(Fuente Dual utilizada para la alimentación de nuestro circuito) 37
Ilustración 23: Tektronix TDS 2012 osciloscopio usado para verificar que la señal de salida entregada por el circuito sea el mismo que se muestra en la interfaz. 37
Ilustración 24: BK PRECISION 4017A 10MHz SWEEP/FUNCTION GENERATOR Generador usado para realizar pruebas básicas iniciales con el circuito 38
Ilustración 25: Reunión en laboratorio de DAQ; se puede observar el comportamiento de la señal de rampa(pendiente positiva) 38
Ilustración 26: Prueba señal rampa, reunión en laboratorio de mecatrónica (FIME) 39
Ilustración 27: Diseño PCB 39
Ilustración 28: Primeras pruebas, obtención de datos de GUI 40
Ilustración 29: Prueba detección de errores en datos 40
Ilustración 30: Prueba detección de errores en datos 41
Ilustración 31: Prueba detección de errores en datos 41

Diseño:

Como se viene comentando, el proyecto de adquisición de datos en cuanto a concepto no es algo emergente sino que se basa en el antecedente elaborado en 2015 donde se usa Arduino y LabVIEW (GUI). Entonces el proyecto en cuestión incluye al software pero también al hardware el cual engloba la unidad de control, el sistema a medir y acondicionamientos necesarios; los cuales se pueden representar como un todo a través del siguiente esquema:

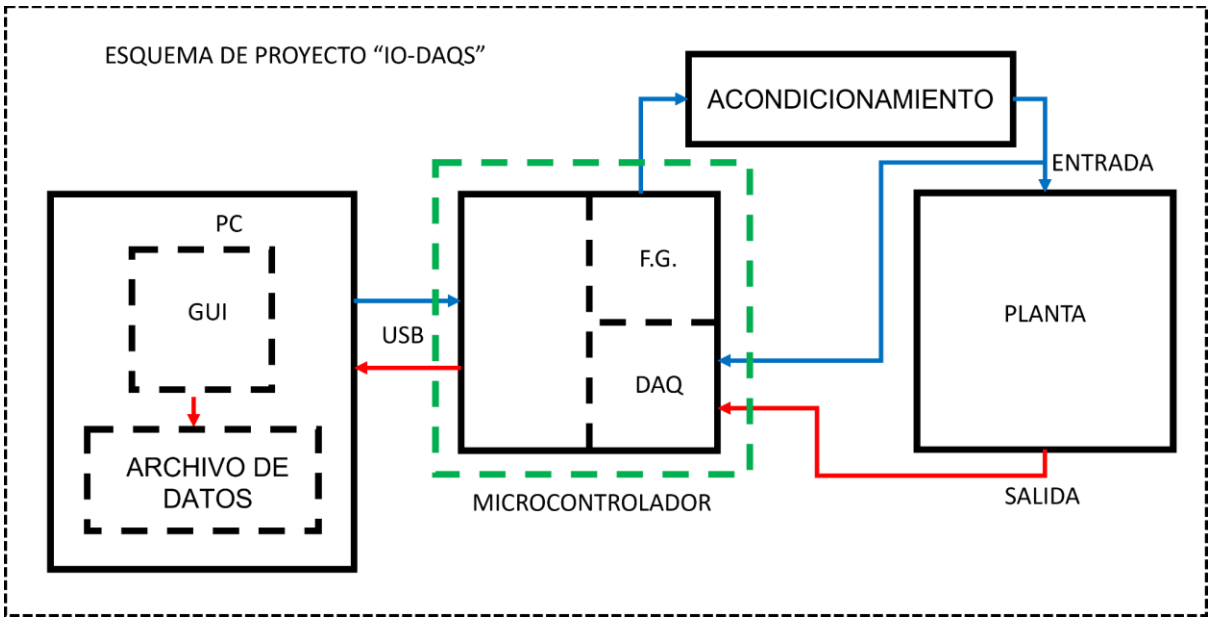

Ilustración 1: Esquema de Proyecto

Así que en términos de diseño los elementos como el microcontrolador, el acondicionamiento y la planta permanecen iguales en términos generales. Pues en el trabajo de referencia el principal problema yacía en el uso de LabVIEW, es decir, la GUI (Graphical User Interface), puesto que no es tan accesible en términos monetarios. Así que para cumplir con los objetivos, la propuesta requiere que ahora la GUI se elabore con Python además que la programación del microcontrolador requiere de ajustes para actuar sincronizadamente con las demás señales (diferentes de la escalón); lo que se traduce en nuevas características.

Diseño de Programa

En cuanto a desarrollo, se viene comentando que son dos estructuras principales, la llevada a cabo en Python y aquella con el IDE de Arduino. Ambas estructuras son cobijadas por el programa y se organizan a modo que conviven y desempeñan en

sinergia, en donde uno requiere del otro para funcionar correctamente. De esta forma el programa como tal se puede ilustrar y desglosar de la siguiente forma:

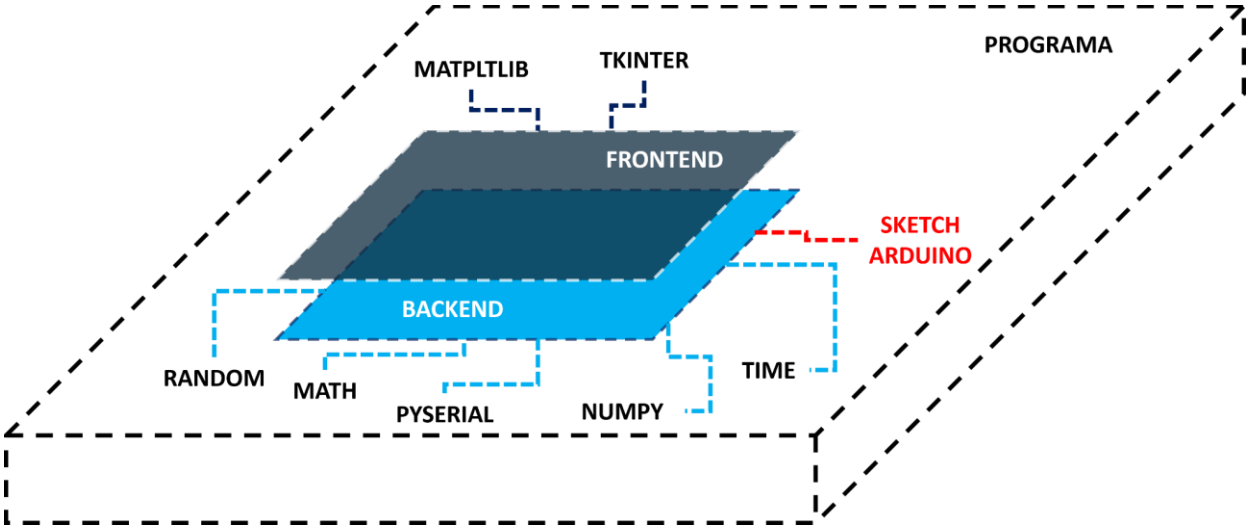

Ilustración 2: Diseño del programa (frontend y backend)

A como se observa en la figura 2, el programa se desglosa en dos “capas”, la primera que es el “***frontend***” y la segunda que es el “***backend***”; cada una se crea o depende de las llamadas “librerías” o “módulos” los cuales contienen funciones específicas que contribuyen al funcionamiento general del programa y como se intuye esta visión del programa no discrimina el lenguaje empleado sino su propósito.

Descripción de librerías

El *frontend* (front por su traducción al español como frente o parte frontal) describe a todo aquello que sea visible y con lo cual el usuario va a interactuar, es decir, la GUI. Entonces para construir esta interfaz se requirieron las librerías “***tkinter***” para los elementos como ventanas, botones, textos, separadores, entre otros; y la librería “***matplotlib***”, la cual se encarga de mostrar datos graficados en un plano cartesiano, junto con sus datos como títulos, nombre de ejes, leyenda entre otros. En conjunto forman la cara del programa.

Pero una GUI es inútil si no existe una lógica entre los distintos componentes para llevar a cabo funciones específicas y realizar una tarea de interés. Es por ello que se requiere el “backend” (back por su traducción a español como espalda o parte trasera) que se refiere a toda relación lógica como condiciones, estructuras de datos, variables, operaciones lógicas y bucles; con los cuales se condiciona el comportamiento que tendrá la GUI como conjunto. Por ejemplo un botón físico de un circuito, por sí mismo no hace nada si no está conectado apropiadamente. Aquí el botón es un elemento “frontend” mientras que el circuito que realiza funciones

dependientes del botón es el “backend”. Este mismo escenario es trasladable al software.

De tal suerte que las librerías encargadas de la lógica de la GUI son por ejemplo “***numpy***”, la cual es una herramienta de cálculo numérico, permite crear arreglos como vectores y matrices, los cuales contienen la información de los datos medidos y que también son utilizados para generar las gráficas y el contenido de archivos con datos exportados. Por otro lado se tiene “***pyserial***”, que en esencia permite la comunicación con el arduino, dando lugar tanto a la lectura como escritura vía puerto serial. “***Random***” sirve para hacer funcionar la señal de ruido, pues aporta con la generación de números pseudoaleatorios para producirla.

Similarmente la librería “***time***” aporta con poder obtener fechas u operar con datos del tipo fecha, su uso en el programa es mínimo y se limita a añadir la fecha y hora de realización de mediciones, una vez se guardan los datos en un archivo. Similarmente ocurre con la librería math, pues permite llevar a cabo operaciones y funciones matemáticas de orden un tanto más complejo que las operaciones aritméticas y booleanas básicas; por ejemplo conversión de números a distintos sistemas numéricos (decimal a binario) y truncamiento de decimales. Cabe mencionar que la conversión de uno a otro tipo de datos es en extremo frecuente, (de entero a float, de float a string etc.)

Del mismo modo el ***sketch de Arduino*** cuyo lenguaje es en sí C++, no sé mezcla o requiere directamente de las demás librerías de Python para operar, pues es cierto que se ocupan datos de inicio para funcionar, pero en sí el código solo se encuentra cargado en el microcontrolador, por lo que a cómo este se energice, el programa empezará a ejecutarse. Se considera backend ya que no aporta a la disposición de los elementos gráficos sino que se encarga de generar señales al igual que medirlas y posteriormente comunicarlas por puerto serial.

En la cuestión de diseño, debido a la gran aportación que se realizó de parte del anterior equipo encargado de comenzar este proyecto no se necesitó como tal un diseño físico o alguna renovación grande de estos diseños propuestos con anterioridad, pero esto no quita el hecho de que si se realizaron cambios mínimos en componentes.

Estructura lógica del programa

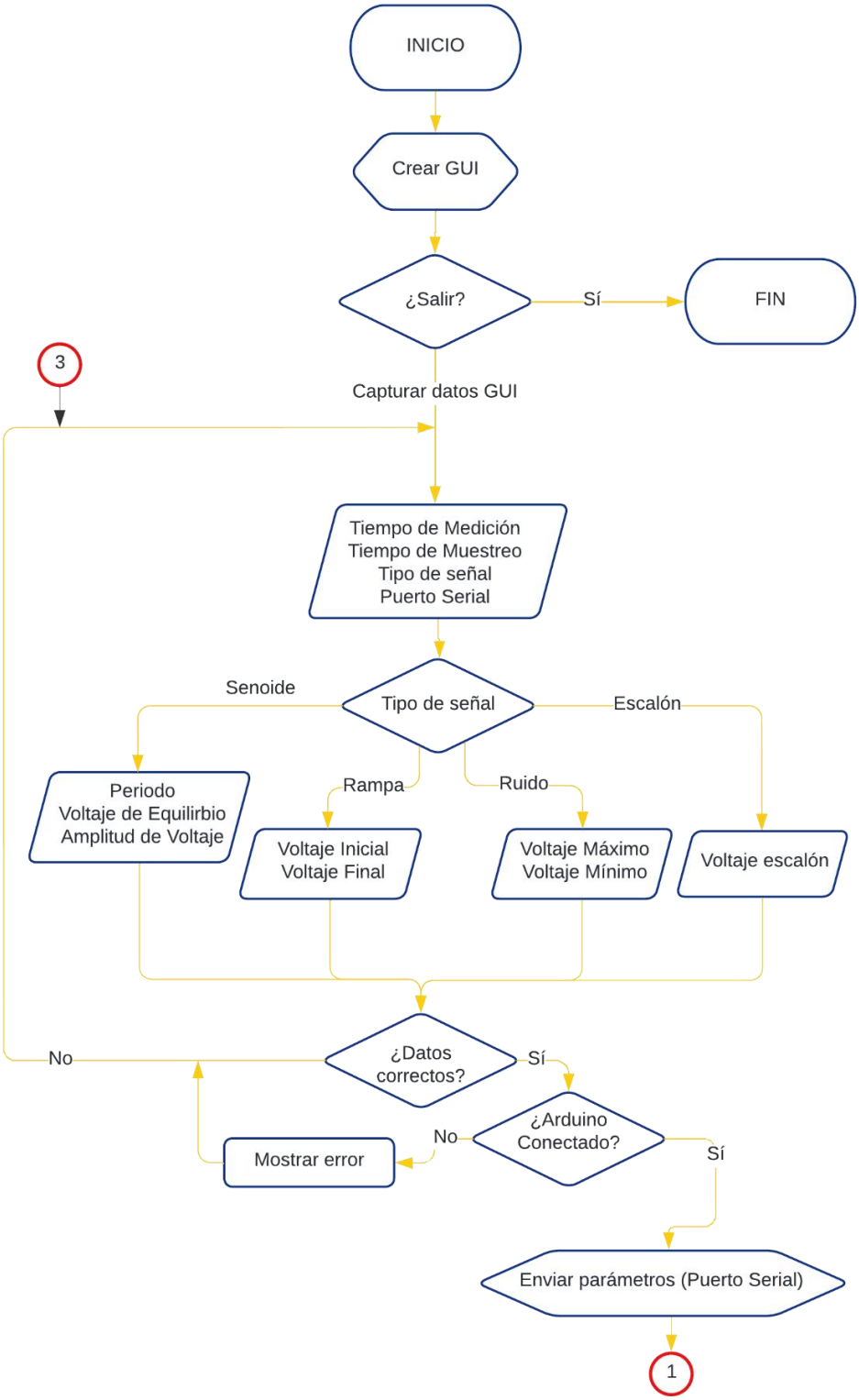

Ilustración 3: Diagrama de flujo del programa (parte 1)

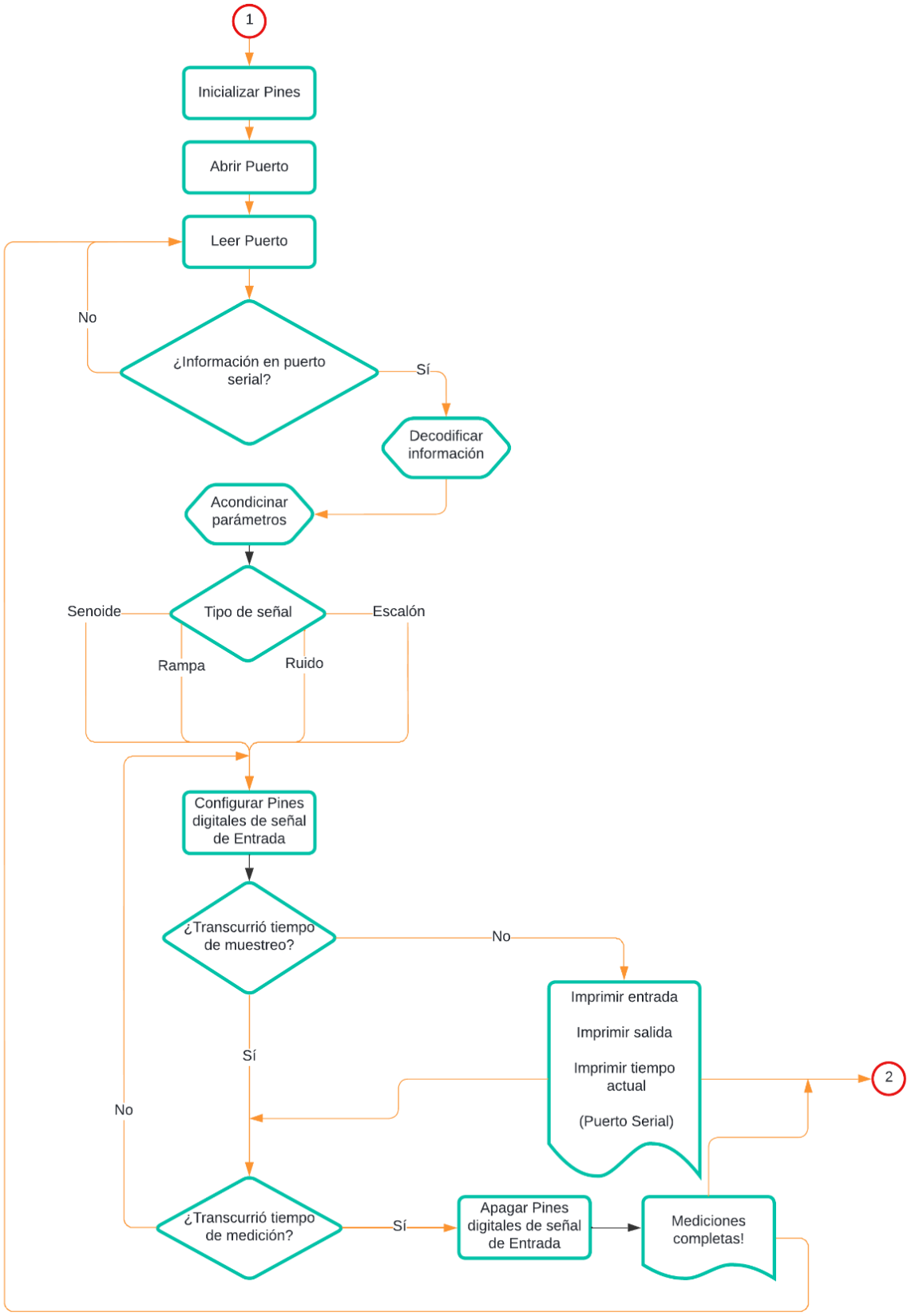

Ilustración 4: Diagrama de flujo del programa (parte 2)

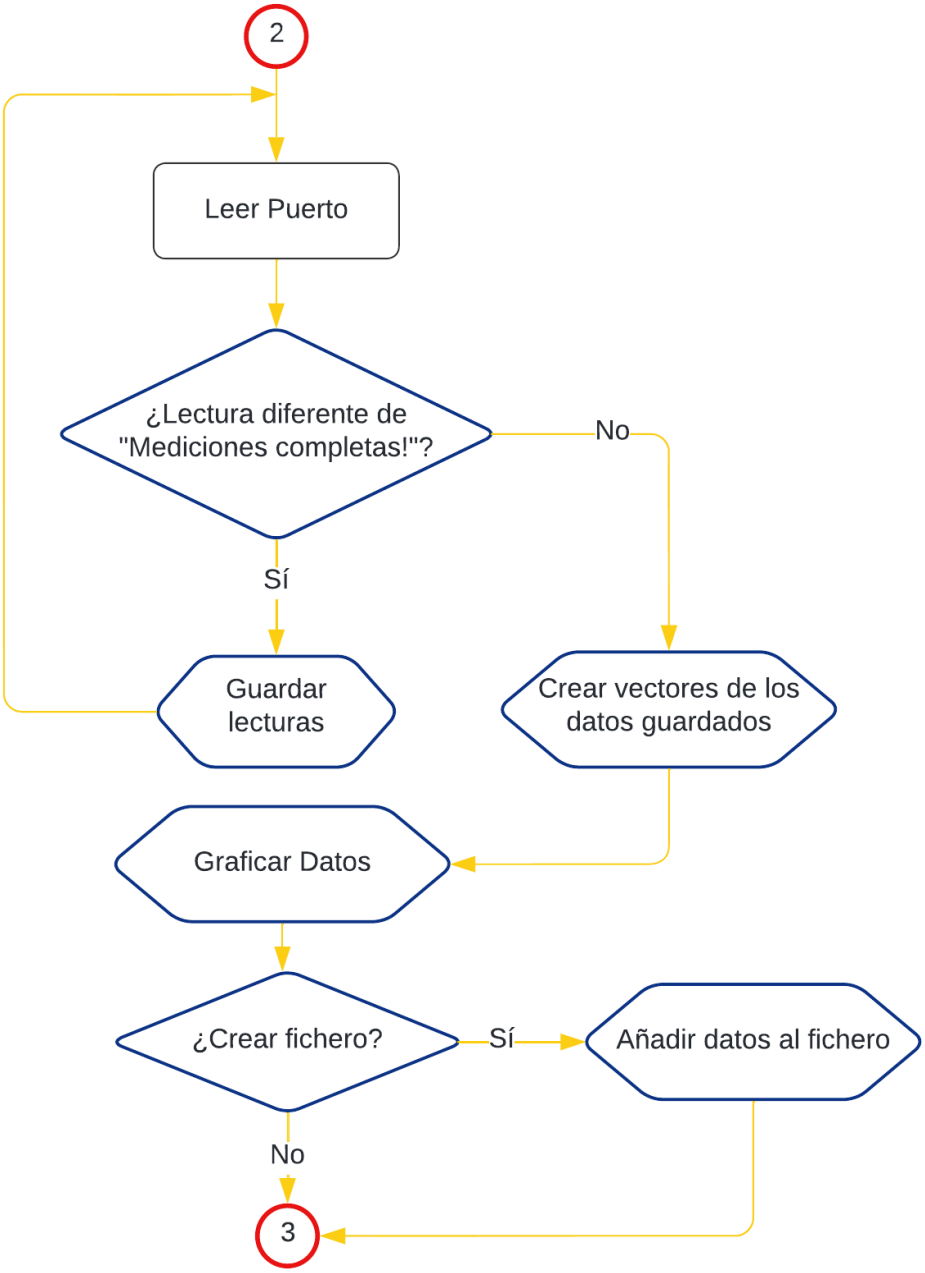

Ilustración 5: Diagrama de flujo del programa (parte 3)

Diseño del Hardware:

Arduino

Ilustración 6: Placa Arduino UNO R3

El arduino nos provee de una gran cantidad de funciones y herramientas, sin embargo las únicas utilizadas fueron dos pines del puerto analógico (A1 y A2) que trabajaron como puntos de medición, así como ocho pines del puerto digital (2 - 9).

Tomando como referencia el esquema del proyecto, primero se usan los pines digitales (2 - 9), los cuales mediante el acondicionamiento de señal, se transfieren a un convertidor digital analógico de 8 bits, generando distintos tipos de señales; esta señal se dirige a la planta y se mide con los dos pines analógicos (A1 y A2), estos pines se usaron en configuración de ADC (Convertidor analógico-digital) después de haber sido multiplexados, esto con el objetivo de medir dos señales, la que entra a la planta (A1) y la que sale de la misma (A2); la resolución implementada fue de 10 bits, esto es posible mediante la composición interna del microcontrolador ATMEGA328P (color amarillo), el cual cuenta con un ADC incorporado junto con un multiplexor de 6 puertos.

Acondicionamiento

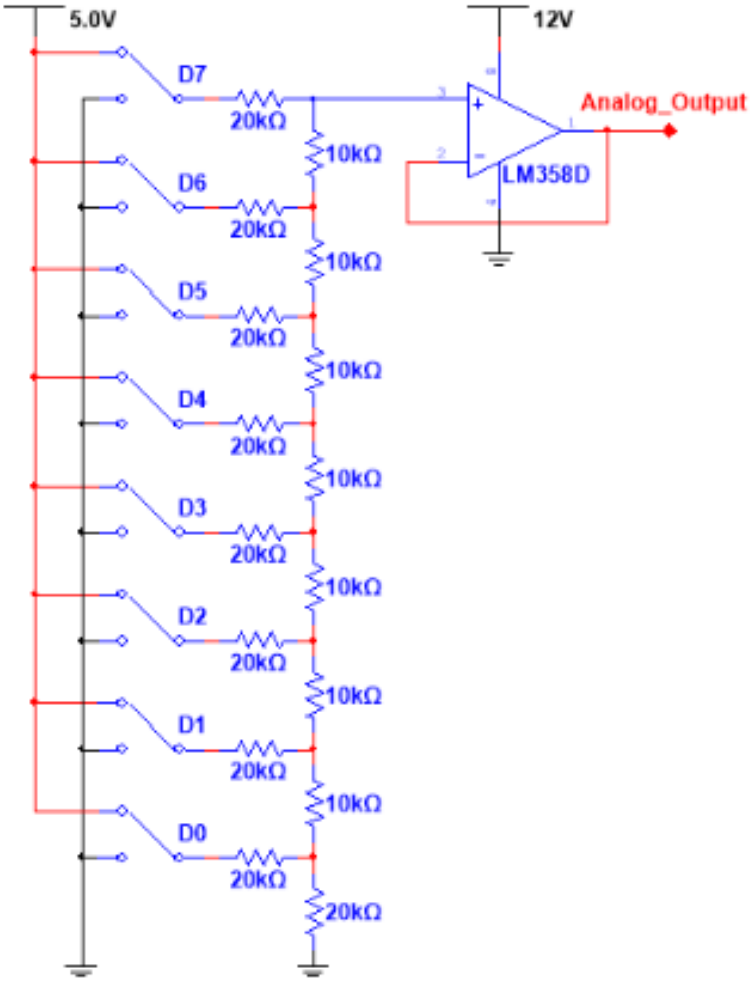

Figura 5.1

Ilustración 7: Convertidor Digital Analógico R-2R

Este acondicionamiento se lleva a cabo mediante un arreglo de resistencias y un amplificador operacional, que en conjunto forman un convertidor digital analógico de 8 bits, el cual es del tipo escalera con resistencia en relación R-2R. El amplificador operacional, que estaba en configuración de seguidor de voltaje, se usó con el objetivo de acoplar impedancias, esto es necesario para que la planta reciba el voltaje que necesita y no menos.

Planta

El de referencia con el cual se llevaron a cabo los análisis es el siguiente:

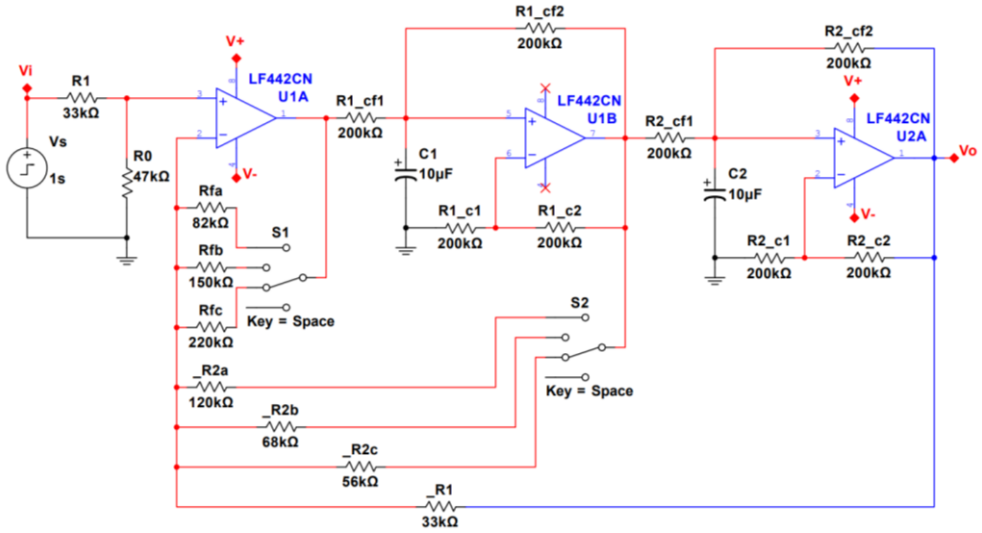

Ilustración 8: Sistema a experimentar (respuesta transitoria)

Este circuito fue el utilizado, el cual mediante la selección apropiada de resistencias, se puede estimar la respuesta transitoria que tendrá el circuito, respuesta asumida para entradas escalón. Como tal y para efectos de esta etapa el desarrollo del circuito en su contexto matemático no se lleva a cabo, pues el autor y diseñador del circuito contempla los cálculos y descripciones más detalladas del funcionamiento en el propio documento.

Cálculos:

Señales

Para poder generar las señales deseadas se requiere una serie de transformaciones previo a alimentar el sistema e incluso previo a ser acondicionadas por el DAC. Simplificando el proceso, los parámetros de la GUI son enviados por puerto serial en forma de una gran cadena de texto que separa los valores por comas; al llegar al arduino, esta cadena se procesa a modo que cada valor se guarda en un arreglo, después se les cambia el tipo de dato ya sea entero, flotante o permanece como cadena de texto.

Algo común a todas las señales es que si bien se tiene un valor de voltaje en rango de 0 a 5 V, se deben transformar a modo que al final se obtenga una palabra en binario que indique cómo se deben configurar los pines digitales. Así que lo primero que se contempla es la cantidad de bits del DAC, que en este caso es 8. Cabe mencionar que la cantidad de bits puede variar y con ella la precisión del voltaje convertido.

A partir de este dato se puede determinar la resolución del DAC la cual se determina de la siguiente manera:

$$Resolución: \frac{rango}{valores\ discretos} = \frac{5+0}{2^8-1} = \frac{5}{255} = 0.01960\ V$$

Este número lo único que sugiere es el cambio mínimo y discreto que puede detectar el DAC. Ya que solo se cuenta con 255 niveles discretos, se puede asumir que el último nivel corresponde al valor máximo del rango, es decir 5 representaría 255 en esa escala, es entonces que se usan razones para determinar la conversión a esta escala; que dicho de otra forma es la ecuación de la recta al asumir una relación lineal:

$$y = mx + b$$

$$\frac{5}{voltaje} = \frac{255}{nivel\ discreto}$$

$$nivel\ discreto = \frac{255}{5} * voltaje$$

El nivel discreto debe ser forzosamente un número de tipo entero por lo que se puede redondear o truncar, por lo general se trunca; después se debe pasar del sistema decimal al sistema binario, en donde dicho número debe ser forzosamente de 8 bits al igual que el DAC, en donde cada pin digital puede presentar un estado lógico de 1 o 0; y que tras el acondicionamiento se traduce en el voltaje analógico que desde un principio se estableció.

Una de los métodos más conocidos para hacer la conversión (decimal a binario) es la de dividir el número decimal entre la base del sistema (2 en este caso) y dividir continuamente el cociente; hasta que el el cociente no pueda ser dividido por un entero y que el resultado también sea entero. Los residuos son 1 o 0, por lo que la conversión a binario es los residuos leídos de forma inversa (del último al primer cociente).

Con esto explicado lo siguiente es adentrarse en cuales son las ecuaciones que rigen el comportamiento de las señales.

Escalón

Esta señal es como un tal un valor constante con condiciones como las siguientes:

$$V_{in}(T_A) = voltaje \quad \{0 \leq voltaje \leq 5\}\ y\ \{0 \leq T_A \leq T_F\}$$

En donde:

V_{in} = Voltaje de Entrada
T_A = $Tiempo$ Actual
T_F = $Tiempo$ Final

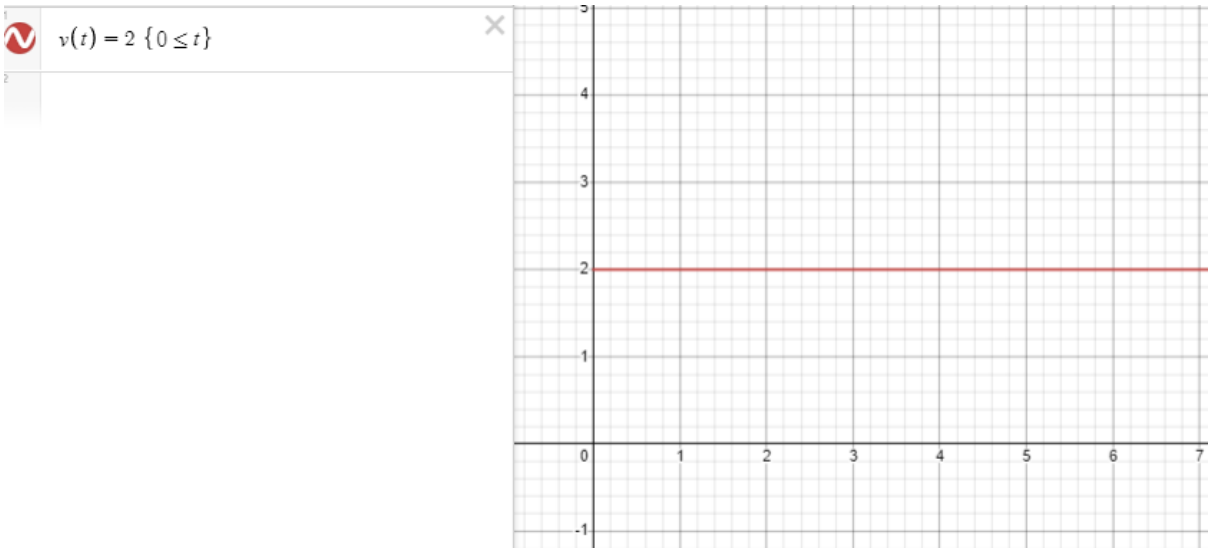

Ilustración 9: Gráfica señal escalón (Desmos)

Rampa

Esta señal en específico se caracteriza ya que cambia de forma uniforme, es decir, lineal, pero se han hecho una serie de modificaciones para que la señal evolucione de esta forma hasta que el tiempo de medición acabe, en este caso se hace referencia a una interpolación lineal entre dos puntos, puntos que en realidad están condicionados por el tiempo de inicio y fin (eje horizontal) y por los voltajes de inicio y fin (eje vertical):

$$V_{in}(T_A) = \frac{V_F - V_I}{T_F - T_I} * (T_A - T_I) + V_I$$

Y las condiciones de tanto de voltajes como tiempos son:

$$T_I \leq T_A \leq T_F \quad 0 \leq V_F \leq 5 \quad 0 \leq V_I \leq 5$$

En donde:

V_{in} = Voltaje de Entrada
T_A = $Tiempo$ Actual
T_I = $Tiempo$ Inicial
T_F = $Tiempo$ Final
V_I = Voltaje Inicial
V_F = Voltaje Final

Ejemplo:

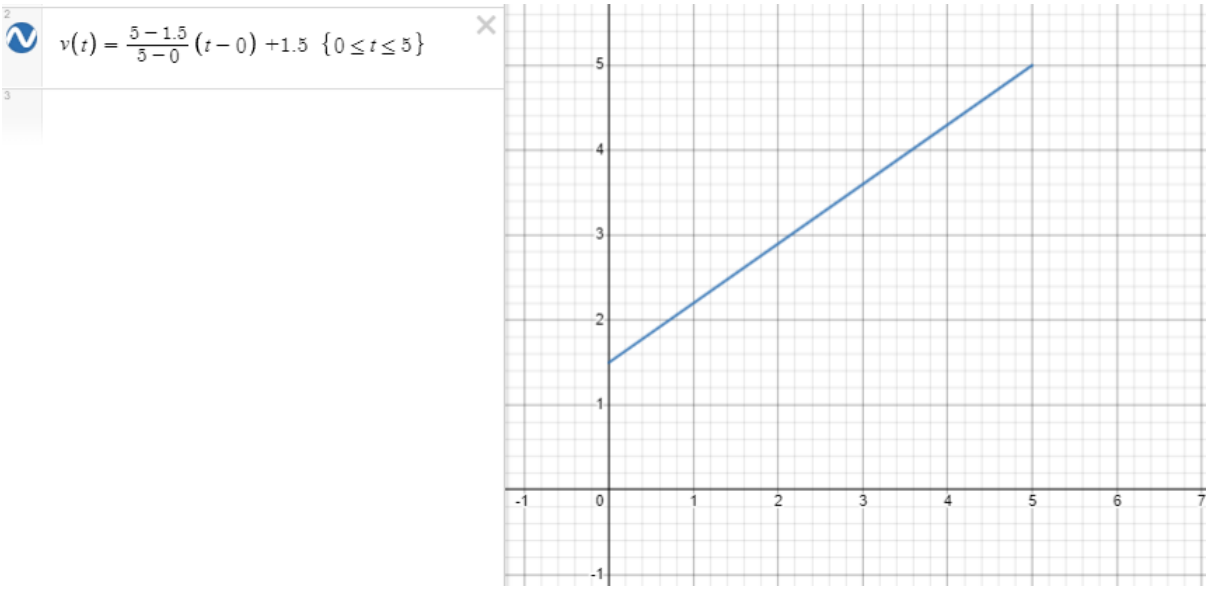

Ilustración 10: Gráfica señal rampa (Desmos)

Seno

Esta señal incluye la función seno y además de incorporar un periodo como argumento, también se requiere de establecer un punto de equilibrio al igual que la amplitud de la misma; la expresión que describe esta señal es la siguiente:

$$V_{in}(T_A) = V_E + A * \sin\left(\frac{2\pi}{T} * T_A\right)$$

Las condiciones que se deben cumplir son:

$$0 \le A \le 5 \qquad 0 \le V_E \le 5 \qquad T_I \le T_A \le T_F$$

Y los significados de cada variable son:

$V_{in} =$ Voltaje de Entrada
$T_A = Tiempo$ Actual
$T = Periodo$
$V_E = Voltaje\ de\ Equilibrio$
$A = Amplitud\ de\ Voltaje$

Finalmente la gráfica representativa de esta señal es:

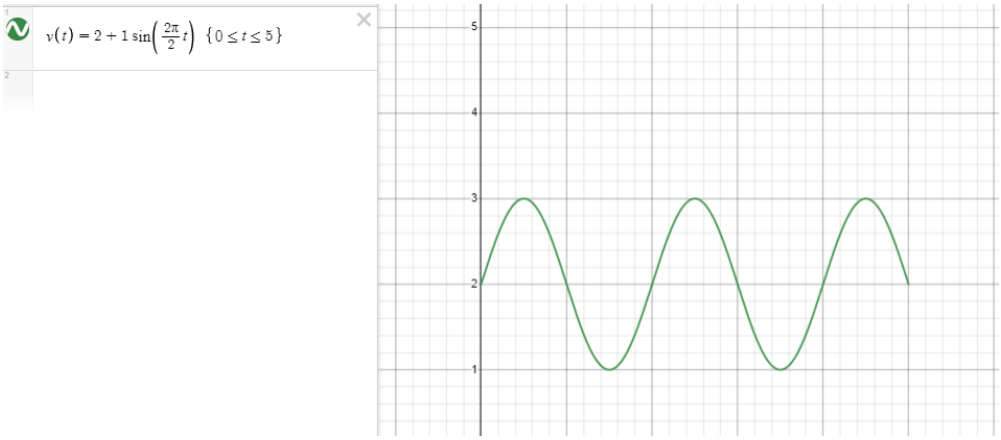

Ilustración 11: Gráfica señal seno (Desmos)

Acondicionamiento

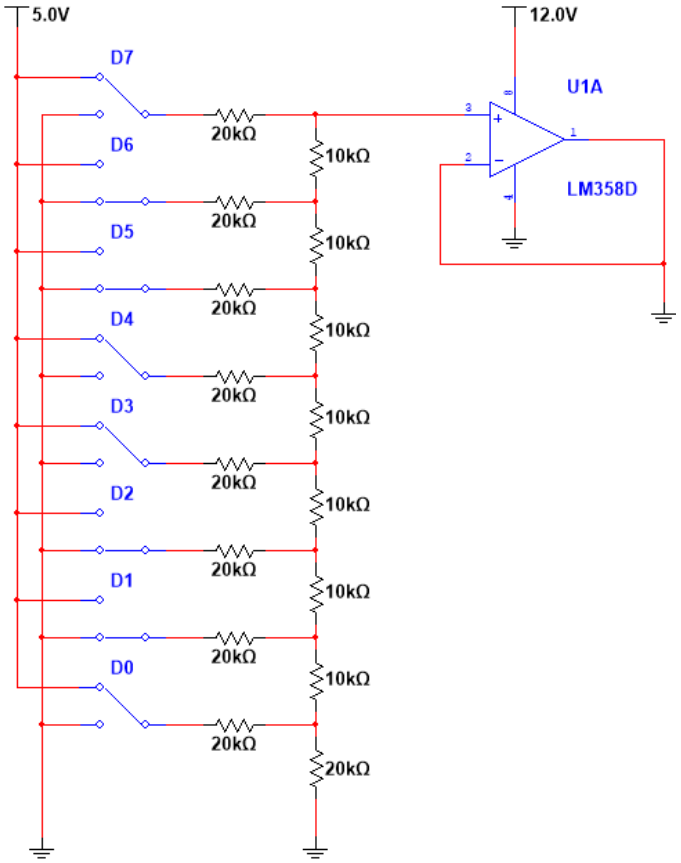

Ilustración 12: DAC

$$V_o = A * V_{ref}(\frac{B_0}{2^8} + \frac{B_1}{2^7} + \frac{B_2}{2^6} + \frac{B_3}{2^5} + \frac{B_4}{2^4} + \frac{B_5}{2^3} + \frac{B_6}{2^2} + \frac{B_7}{2^1})$$

$$V_{ref} = 5.0\,V$$

$$A = 1$$

$$B_0, B_3, B_4, B_7 = 1$$

$$V_o = 1 * 5(\frac{1}{2^8} + \frac{0}{2^7} + \frac{0}{2^6} + \frac{1}{2^5} + \frac{1}{2^4} + \frac{0}{2^3} + \frac{0}{2^2} + \frac{1}{2^1})$$

$$V_o = 1 * 5(\frac{1}{2^8} + \frac{1}{2^5} + \frac{1}{2^4} + \frac{1}{2^1})$$

$$V_o = 2.988\,V$$

Ilustración 13: Ecuaciones del comportamiento del DAC

En las imágenes anteriores, se muestra y comprueba la situación de conversión del valor digital “10011001” al analógico, el cual sería “3”; esto se llevó a cabo mediante una ecuación en específico, colocada en la segunda imágen, donde Vref es el voltaje de salida de los pines digitales, A es la ganancia de la configuración del amplificador operacional, el cual al estar en “seguidor de voltaje” su ganancia es unitaria y las B’s representan el estado de salida, en donde pueden estar bajo o alto, 1 o 0, de los pines

digitales; como resultado obtenemos un valor cercano a “3”. Análisis consultado por (electronica, 2017)

PCB

El uso de PCB nos permite tener una mejor presentación y control del espacio (compactificación). Se realizaron por medio de Proteus y su función para generar PCB; para acelerar el proceso se optó por mandar el archivo a un lugar especializado en imprimir este tipo de tarjetas (JLCPCB).

Para las pruebas principales a lo largo del desarrollo requirieron del circuito prueba y el acondicionamiento; por fortuna se pudo conseguir el circuito implementado en 2015 el cual es el siguiente:

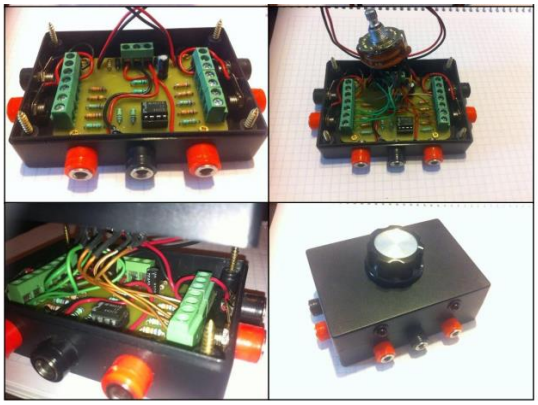

Ilustración 14: Circuito implementado en 2015 (referencia física)

Este primer esfuerzo ya lograba acomodar los componentes tanto del acondicionamiento como de la planta en bloques circuitales conectables. De hecho las pruebas con las que se verificó el funcionamiento óptimo del programa fue con este circuito, aun así el convertidor digital analógico fue implementado en una tablilla de conexiones

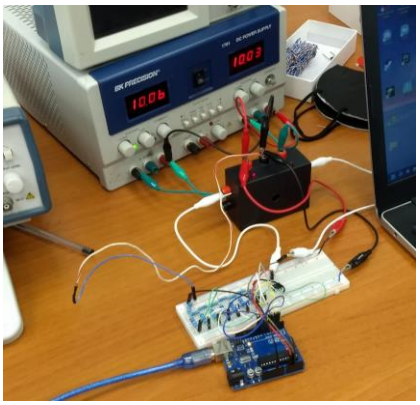

Ilustración 15: Implementación del circuito

Las emisión de señales de características definidas fueron probadas en múltiples ocasiones con la finalidad de garantizar repetibilidad. Entre las múltiples ocasiones, en la imagen se describe una de ellas.

El inconveniente que presenta es que se requieren muchos cables para poder realizar las conexiones necesarias, es por ello que el acondicionamiento se intentó incorporar junto con la planta sobre una misma

placa. Proceso que requirió el programa ya mencionado, para agregar los componentes y conectarlos apropiadamente para luego ser manufacturado por terceros (JLCPCB).

El circuito en cuestión acopla al acondicionamiento y la planta bajo una misma placa, esto a modo que coincida con las dimensiones del Arduino así como los pines de salida y de entrada; en el proceso se evita el uso de cables ya que se puede realizar una conexión directa por el tipo de terminales. El PCB como tal solo requería de “”rutear” la salida del DAC a la planta; a continuación se muestra el pcb sin las resistencias soldadas:

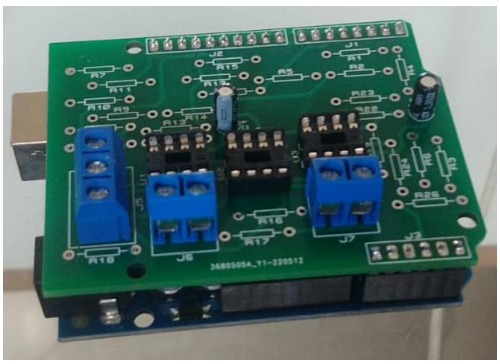

Ilustración 16: Diseño PCB integral ya en físico (primer diseño)

Pero al intentar múltiples pruebas con el PCB, las señales salían muy distorsionadas respecto a los resultados ya comprobados, pues si se instaba alimentar uno de los pines de lectura, el circuito y el programa los leía apropiadamente, más al incorporar el PCB, la distorsión era tal que se asume errores de diseño en la PCB o de conexión interna de componentes.

Debido a este problema se planteó una reorganización de los bloques circuitales al igual que un exhaustiva revisión a las conexiones para garantizar funcionalidad, esfuerzo que por falta de gestión quedó en una propuesta de diseño que no se concretó en un circuito físico. El diagrama contiene cada componente anexado en el desglose financiero al igual que sus nombres de identificación:

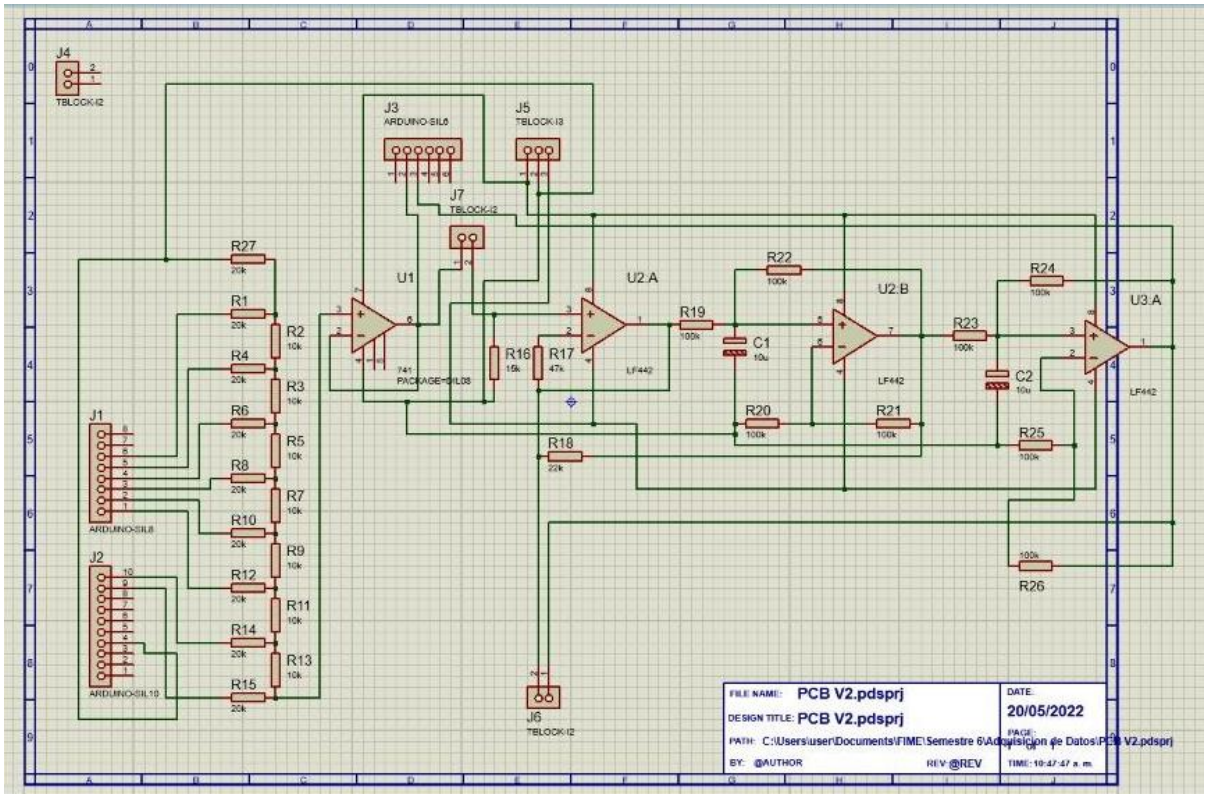

Ilustración 17: Circuito en Proteus (segundo diseño- propuesta)

Y la distribución de los componentes en el layout de Proteus luce tal que así:

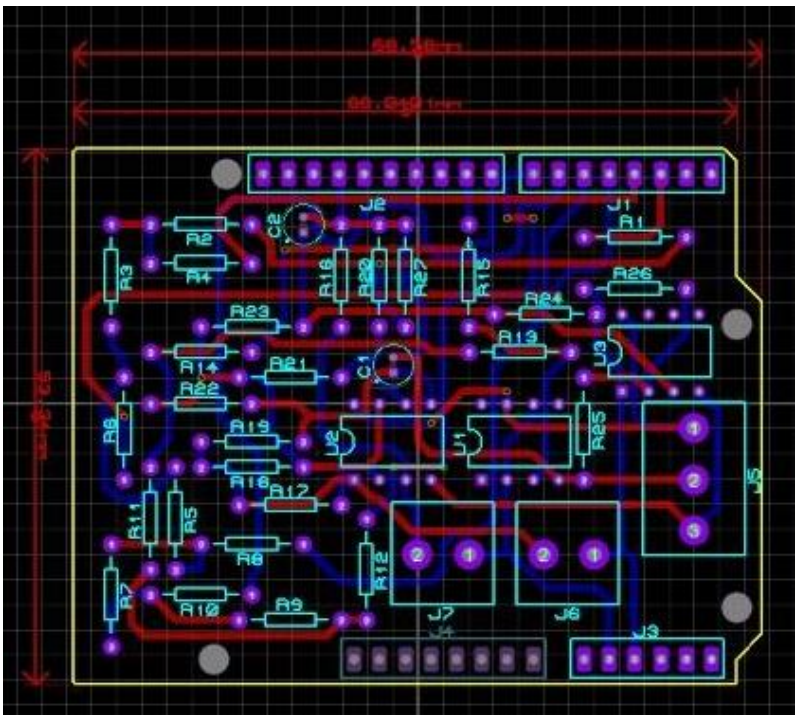

Ilustración 18: Layout del PCB (segundo diseño- propuesta)

Finalmente de las herramientas con las que cuenta el software de Proteus es la de visualización 3D de la placa diseñada. En esta ocasión la placa resultante es la siguiente:

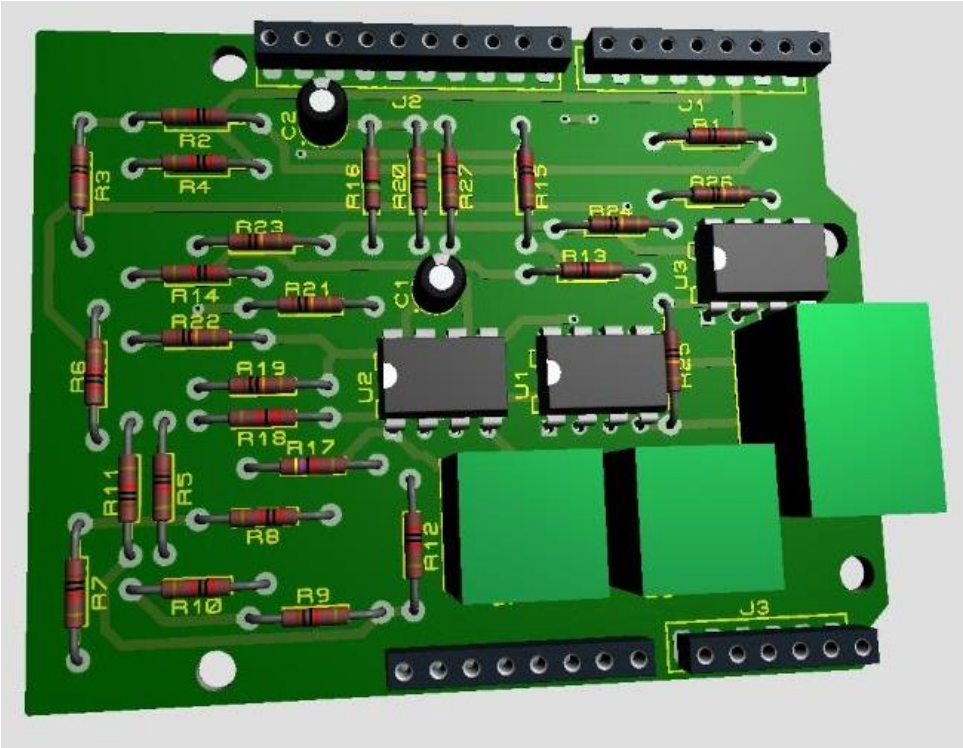

Ilustración 19: Vista 3D circuito en proteus (segundo diseño-propuesta)

Debido a problemas con los tiempos de manufactura y entrega, las pruebas tampoco se concretaron, de modo que el circuito presentado, queda en más que una propuesta de diseño unificador que aun así acarrea el control de espacio y de algún modo aspecto estético. Por lo que los resultados obtenidos quedan atados al circuito expuesto en un principio.

Impresión de los códigos fuente de los programas

Software

Código Arduino

```
#define InputPinMeasurement A1
#define OutputPinMeasurement A2
const int FeedingVoltagePin[]={2,3,4,5,6,7,8,9};
void setup() {
  Serial.begin(115200);
  Serial.println("Puerto encendido");
  pinMode(InputPinMeasurement,INPUT);
  pinMode(OutputPinMeasurement,INPUT);
  for(int j=0;j<8;j++){
    pinMode(FeedingVoltagePin[j],OUTPUT);
  }

}
```

```
void loop() {
  DecodeDataChain();
}

void DecodeDataChain(){
    if(Serial.available()){

    //Serial.println("Serial Port Open");
    #define ArraySize 7
    String parameters[ArraySize],DataChain, element;
    #define SeparatorCharacter ","
    int SubstringInitialPosition, SubstringLastPosition, ArrayIndex;

    DataChain = Serial.readString();
//if something in Serial Port, save it in a variable
    delay(1000);
    SubstringInitialPosition=0;
//first character's position of the first substring
    SubstringLastPosition =
DataChain.indexOf(SeparatorCharacter,SubstringInitialPosition);  //last character's
position of the first substring

    ArrayIndex = 0;
    while (SubstringLastPosition!=-1) {
//while last comma's position is different from the end of DataChain, keep decoding
      element = DataChain.substring(SubstringInitialPosition,
SubstringLastPosition);
      parameters[ArrayIndex]= element;

      SubstringInitialPosition = SubstringLastPosition+1;
      SubstringLastPosition = DataChain.indexOf(SeparatorCharacter,
SubstringInitialPosition);
      ArrayIndex =ArrayIndex+1;
    }

    element = DataChain.substring(SubstringInitialPosition, DataChain.length());
    parameters[ArraySize-1] = element;

        //##### SECTION [Measuring and feeding the circuit]
    unsigned long InitialTime,CurrentTime,MeasurementTime,SamplingTime,TriggerTime;
    MeasurementTime = parameters[0].toFloat()*1000000;
    SamplingTime = parameters[1].toFloat()*1000000;
    int ArraysLenght = (int)(MeasurementTime/SamplingTime);

    long InputVoltage;
    long Dec_val;
    String SignalType = parameters[2];
    int FinalV = parameters[4].toInt();
    int InitialV = parameters[3].toInt();
    unsigned long Period = parameters[5].toFloat()*1000000;
    unsigned long FinalTime = MeasurementTime+SamplingTime;
    long seed = parameters[6].toInt();
    randomSeed(seed);

    //#########ASIGNACIÓN DE TIEMPOS#########
    float m =((float)(FinalV-InitialV)/((float)(FinalTime-InitialTime)));    //
(FinalV-InitialV)/(FinalTime-InitialTime)
```

```
//    Serial.print(SignalType);Serial.print(" ");Serial.println(m);
//    Serial.print(InitialV);Serial.print(" ");Serial.println(FinalV);

    //#########EVALUAR SignalType###################3

    //#######STEP LOOP#####################
    //#######STEP LOOP#####################
    //#######STEP LOOP#####################
    if((SignalType == "step")){
      InputVoltage = InitialV;
      for(int i=7; i>=0; i--){
        bool LogicState = bitRead(InputVoltage, i);
        //Serial.print(LogicState, BIN);  //shows: 00000011
        if(LogicState==1){
          digitalWrite(FeedingVoltagePin[i],HIGH);
        }
        else{
          digitalWrite((FeedingVoltagePin[i]),LOW);
        }
      }
      InitialTime = micros();
      CurrentTime = micros();
      TriggerTime = InitialTime;

      do{
        if((CurrentTime-TriggerTime)>SamplingTime){
//if the sampling time has elapsed, then update TriggerTime and measure
          TriggerTime = CurrentTime;
          //take the measurement
          Serial.println(analogRead(InputPinMeasurement));
          Serial.println(analogRead(OutputPinMeasurement));
          Serial.println(micros());
        }

        CurrentTime = micros();
      }    while((CurrentTime-InitialTime)<=(MeasurementTime+SamplingTime));
    }


    //#######SLOPE LOOP#####################
    //#######SLOPE LOOP#####################
    //#######SLOPE LOOP#####################
    if((SignalType == "slope")){

      InitialTime = micros();
      CurrentTime = micros();
      TriggerTime = InitialTime;

      do{
        InputVoltage= round(m*(CurrentTime-InitialTime)+InitialV);

        for(int i=7; i>=0; i--){
          bool LogicState = bitRead(InputVoltage, i);
          if(LogicState==1){
              digitalWrite(FeedingVoltagePin[i],HIGH);
          }
          else{
            digitalWrite((FeedingVoltagePin[i]),LOW);
```

```
            }
        }
        if((CurrentTime-TriggerTime)>SamplingTime){
//if the sampling time has elapsed, then update TriggerTime and measure
          TriggerTime = CurrentTime;
          //take the measurement
          Serial.println(analogRead(InputPinMeasurement));
          Serial.println(analogRead(OutputPinMeasurement));
          Serial.println(micros());
        }
        CurrentTime = micros();
      }    while((CurrentTime-InitialTime)<=(MeasurementTime+SamplingTime));
    }


    //#######SINE LOOP####################
    //#######SINE LOOP####################
    //#######SINE LOOP####################
    if((SignalType == "sine")){
      InitialTime = micros();
      CurrentTime = micros();
      TriggerTime = InitialTime;

      do{
        InputVoltage = round(FinalV +
float(InitialV*sin((2*PI/(float)Period)*CurrentTime)));

        if (InputVoltage>255){
          InputVoltage = 255;

        }
        if(InputVoltage<0){
          InputVoltage = 0;
        }

        for(int i=7; i>=0; i--){
          bool LogicState = bitRead(InputVoltage, i);

          if(LogicState==1){
            digitalWrite(FeedingVoltagePin[i],HIGH);
            //Serial.print(1);
          }
          else{
            digitalWrite((FeedingVoltagePin[i]),LOW);
            //Serial.print(0);
            }
        }
        //Serial.println("");
        if((CurrentTime-TriggerTime)>SamplingTime){
//if the sampling time has elapsed, then update TriggerTime and measure
          TriggerTime = CurrentTime;
          //take the measurement
          Serial.println(analogRead(InputPinMeasurement));
          Serial.println(analogRead(OutputPinMeasurement));
          Serial.println(micros());
        }
        CurrentTime = micros();
```

```
      }    while((CurrentTime-InitialTime)<=(MeasurementTime+SamplingTime));
//While MeasurementTime hasn't elapsed yet, keep up measuring
    }

    //#######NOISE LOOP####################
    //#######NOISE LOOP####################
    //#######NOISE LOOP####################
    if((SignalType == "noise")){
      InitialTime = micros();
      CurrentTime = micros();
      TriggerTime = InitialTime;

      do{
        InputVoltage = random(InitialV,FinalV+1);
        for(int i=7; i>=0; i--){
          bool LogicState = bitRead(InputVoltage, i);

          if(LogicState==1){
            digitalWrite(FeedingVoltagePin[i],HIGH);
          }
          else{
            digitalWrite((FeedingVoltagePin[i]),LOW);
            }
        }
        if((CurrentTime-TriggerTime)>SamplingTime){
//if the sampling time has elapsed, then update TriggerTime and measure
          TriggerTime = CurrentTime;
          //take the measurement
          Serial.println(analogRead(InputPinMeasurement));
          Serial.println(analogRead(OutputPinMeasurement));
          Serial.println(micros());
        }
        CurrentTime = micros();
      }    while((CurrentTime-InitialTime)<=(MeasurementTime+SamplingTime));
    }

    unsigned long dif = CurrentTime-(InitialTime+SamplingTime);
//Shows the "real" MeasurementTime
    //Serial.println(dif);
    //Serial.println(samples);
//Shows the amount of measurements done

    for (int j=1; j<9; j++){
//Turn off all digital pins
      digitalWrite(FeedingVoltagePin[j],LOW);
    }

    Serial.println("Measurements completed!");
    }

}
```

Código Python

(IO-DAQS.py)

```
#Porject Name: IO-DAQS (Input Output Data Acquisition System)
juliomoreno7217@gmail.com
#Creation date: 2022 / march / 3 / 16:55 h

from tkinter import *
from tkinter import ttk
from tkinter import Tk
from tkinter import messagebox
from WorkingArea import Tab2Widgets
import time
import os
import serial
import math
from tkinter.filedialog import asksaveasfile
import numpy
from datetime import datetime

import matplotlib
matplotlib.use("TkAgg")         #backend
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)
from matplotlib.figure import Figure
import random



class IO_DAQS(Tab2Widgets):
    def __init__(self, Window):
        super().__init__()
        self.MainWindow = Window
        self.MainWindow_Width="1050"
        self.MainWindow_Height="700"

self.MainWindow.geometry('{}x{}'.format(self.MainWindow_Width,self.MainWindow_Heigh
t))
        self.Title="Interfaz Python-Arduino"

        self.CreateWidgets()

        #Graph settings:
        self.GraphFigure = Figure(figsize=(5,4.5),dpi=100,edgecolor='black')
        self.Graph = self.GraphFigure.add_subplot(111)

            #canvas._tkcanvas.destroy()
        self.canvas = FigureCanvasTkAgg(self.GraphFigure,
self.MatplotlibGraph_LFrame)
        self.canvas.get_tk_widget().pack(side=TOP, fill = BOTH, expand=False)
        toolbar = NavigationToolbar2Tk(self.canvas, self.MatplotlibGraph_LFrame)
        toolbar.update()
        self.canvas._tkcanvas.pack(side=TOP, fill = BOTH, expand=False)
        self.SetGraphProperties()
```

```
    def CreateWidgets(self):
        self.CreateNotebook()
        self.AddTabs_Notebook()

    def CreateNotebook(self):
        self.notebook = ttk.Notebook(self.MainWindow,
            width=self.MainWindow_Width,
            height=self.MainWindow_Height)
        self.notebook.grid(row=0,column=0, padx=10, ipadx=20)

    def AddTabs_Notebook(self):
        self.FillTabs(self.notebook)
        self.notebook.add(self.FrameTab1, text= "Apartado 1")
        self.notebook.add(self.FrameTab2, text= "Apartado 2")
        self.notebook.add(self.FrameTab3, text= "Apartado 3")

    def FillTabs(self,ParentName):
            self.FillTab1(ParentName)
            self.CreateWidgets_Tab2(ParentName)
            self.FillTab3(ParentName)



    #To do: Filltab1 and FilTab3 muest be gone!
    def FillTab1(self,ParentName):
        self.FrameTab1 = ttk.Frame(ParentName, width='1200', height='600')
        self.FrameTab1.grid(row=1,column=0)

        self.Title1 = Label(self.FrameTab1, text = 'Menú Principal',padx=50,
pady=20,font=('Arial Rounded MT Bold', 20))
        self.Title1.grid(row=0, column=0)

        self.Intro = Label(self.FrameTab1, text = """
        Esta inerfaz tiene como proposito seleccionar la configuración en la que
una placa Arduino
       pueda llevar a cabo la adquisición de datos, actuando tanto como sistema de
medición y
        generador de funciones. Para empezar con la configuración dirigase al
siguiente apartado.
        """,anchor='e', font=('Arial Rounded MT Bold', 10))
        self.Intro.grid(row=1, column=0)

    def FillTab3(self,ParentName):
        self.FrameTab3 = ttk.Frame(ParentName, width='1200', height='600')
        self.FrameTab3.grid(row=1,column=0, columnspan=4)

        self.Title3 = Label(self.FrameTab3, text = 'Acerca de
Nosotros',padx=50,pady=20, font=('Arial Rounded MT Bold', 20))
        self.Title3.grid(row=0, column=0)

        self.AboutUs = Label(self.FrameTab3, text = """
        Porgrama desarrollado por: Julio de Jesús Moreno Sánchez

        Correo: juliomoreno7217@gmail.com

        Link del repositorio: https://github.com/IODAQS-Experts/IODAQS-
Repository.git
```

```
        Fecha de elaboración: marzo 23 - mayo 20, 2022

        """,anchor='e', font=('Arial Rounded MT Bold', 10))
        self.AboutUs.grid(row=1, column=0)


##INTERACTION##ZONE####INTERACTION##ZONE####INTERACTION##ZONE####INTERACTION##ZONE#
#

##INTERACTION##ZONE####INTERACTION##ZONE####INTERACTION##ZONE####INTERACTION##ZONE#
#
    def RunMeasurements(self):
        self.SendDataToArduino(self.EvaluateDataType())
        self.ReadDataFromArduino()
        self.Create_ReadMeasurementsArrays()
        self.PlotData()

    def EvaluateDataType(self):
        try:
            ##Data##Conditioning##Section####Data##Conditioning##Section##
            ##Data##Conditioning##Section####Data##Conditioning##Section##

            #Sampling-time-prefix adecuation
            prefix={'s':1,'ks':1000,'ms':.001,'us':.000001}

            for key in prefix:
                if self.SamplingPrefix.get()==key:

self.SamplingTime=str(round(float(self.SampligCoefficient.get())*prefix.get(key),6)
)

                if self.PeriodPrefix.get()==key:

self.Period=str(round(float(self.PeriodCoefficient.get())*prefix.get(key),6))

            #Time quantities (in seconds) must be greater than 0!
            if float(self.MeasurementTime.get())>0 and
float(self.SamplingTime)>=.0018 and
float(self.MeasurementTime.get())>float(self.SamplingTime) and
float(self.Period)>0:

                #Converting times to microseconds:
                MeasurementTime = float(self.MeasurementTime.get())
                SamplingTime = float(self.SamplingTime)
                Period = float(self.Period)

                #InputVoltage convertion to a 0-255 value (for ditial pins)
                self.MaxVoltage = 5
                InputFinalVoltage_decimal =
math.trunc((255/self.MaxVoltage)*self.FinalVoltage.get())


                InputInitialVoltage_decimal =
math.trunc((255/self.MaxVoltage)*self.InitialVoltage.get())


                if self.SignalType.get()=='step':
                    InputFinalVoltage_decimal=0
```

```
                seed = random.randint(-99999999,99999999)

                if self.SignalType.get()=='noise' and
InputInitialVoltage_decimal>=InputFinalVoltage_decimal:
                    self.ShowErrorMessage( "Error!","!El voltaje minimo debe ser
menor que el voltaje máximo. Por seguridad los valores se han invertido para
coincidir!")
                    InputInitialVoltage_decimal,
InputFinalVoltage_decimal=InputFinalVoltage_decimal,InputInitialVoltage_decimal

                return
str(MeasurementTime),str(SamplingTime),self.SignalType.get(),str(InputInitialVoltag
e_decimal),str(InputFinalVoltage_decimal),str(Period),str(seed)

        except:
            pass

    def SendDataToArduino(self,parameters=()):
        try:
            self.DataChain = ",".join(parameters)   #makes a single string
separated by the symbol inside ""

            self.EvaluateConexion()

            self.arduino.readline().decode(encoding='ascii', errors='strict')
            self.arduino.write(self.DataChain.encode("ascii", errors='strict'))


        except:
            self.ShowErrorMessage("Error de datos" , "Datos (tiempos o voltajes)
inválidos*")
            self.measurementString.set("0")
            self.samplingString.set("0")

    def EvaluateConexion(self):
        try:
            self.arduino = serial.Serial()    #Open serial Port
            self.arduino.port = self.SerialPort.get()
            self.arduino.baudrate = 115200
            self.arduino.open()
        except:
           self.ShowErrorMessage("Fallo en Conexión","¡Falló la Conexión con
Arduino!")

    def ShowErrorMessage(self, message,title):
        #The must appear a window showing the error, and the inputs must be set to
default
        messagebox.showerror(message, title)

    def ReadDataFromArduino(self):
        try:
            reading = "Measurements started!"
            self.readings = []

            while reading != "Measurements completed!\r\n":
                reading = self.arduino.readline().decode(encoding='ascii',
errors='strict')
```

```
                self.readings.append(reading)

            messagebox.showinfo(message="Lectura Exitosa!", title="¡Datos leídos
exitosamente!")
        except:
            self.ShowErrorMessage( "Fallo en Conexión","¡Falló la Conexión con
Arduino!")

    def StopMeasurements(self):
        "Serial Port Closed!!"
        self.arduino.close()
        pass

    def SaveMeasurements(self):
        if len( self.InputVoltageArray) != 0:
            files = [('All Files', '*.*'),
                ('Comma Separated Values Document', '*.cvs'),
                ('Excel Document','.xls'),
                ('Text Document', '*.txt')]
            file = asksaveasfile(filetypes = files, defaultextension = files)
            if file:
                if self.SignalType.get()=="step":
                    file.write("""Date: {0}
        Measuring time (s): {1}
        Sampling time (s): {2}
        Signal Type: {3}
        Input voltage (V): {4}

""".format(datetime.now(),self.MeasurementTime.get(),self.SamplingTime,self.SignalT
ype.get(),self.InitialVoltage.get()))

                elif self.SignalType.get()=="slope":
                    file.write("""Date: {0}
        Measuring time (s): {1}
        Sampling time (s): {2}
        Signal Type: {3}
        Initial voltage (V): {4}
        Final voltage (V): {5}

""".format(datetime.now(),self.MeasurementTime.get(),self.SamplingTime,self.SignalT
ype.get(),self.InitialVoltage.get(),self.FinalVoltage.get()))

                elif self.SignalType.get()=="sine":
                    file.write("""Date: {0}
        Measuring time (s): {1}
        Sampling time (s): {2}
        Signal Type: {3}
        Amplitude voltage (V): {4}
        Equilibrium voltage (V): {5}
        Period (s): {6}

""".format(datetime.now(),self.MeasurementTime.get(),self.SamplingTime,self.SignalT
ype.get(),self.InitialVoltage.get(),self.FinalVoltage.get(),self.Period))

                elif self.SignalType.get()=="noise":
                    file.write("""Date: {0}
        Measuring time (s): {1}
        Sampling time (s): {2}
```

```
        Signal Type: {3}
        Minimum voltage (V): {4}
        Maximum voltage (V): {5}

""".format(datetime.now(),self.MeasurementTime.get(),self.SamplingTime,self.SignalT
ype.get(),self.InitialVoltage.get(),self.FinalVoltage.get()))


                file.write("\n[ Time(s), Input(V), Output(V) ]: \n\n")
                numpy.savetxt(file,self.MeasurementsArray,delimiter=',')

                messagebox.showinfo(message="Guardado Exitoso!", title="¡Datos
guardados exitosamente!")
                file.close()

    def Create_ReadMeasurementsArrays(self):
        try:
            #Organizing info in lists:
            self.InputVoltageList = []
            self.OutputVoltageList = []
            self.TimeList =[]

            self.MeasurementsList=[]
            for index in range(2,(len(self.readings)-1),3):
                #The last data is always time so the index is referenced to each
datatime indexes' position (because of how readline() works)
                #time in seconds
                self.TimeList.append(float(self.readings[index][:-2])/1000000)
                #Voltages conversion*

self.OutputVoltageList.append(self.MaxVoltage*(float(self.readings[index-1][:-
2])/1023))

self.InputVoltageList.append(self.MaxVoltage*(float(self.readings[index-2][:-
2])/1023))

                self.MeasurementsList.append([float(self.readings[index][:-
2])/1000000,

self.MaxVoltage*(float(self.readings[index-2][:-2])/1023),

self.MaxVoltage*(float(self.readings[index-1][:-2])/1023)])
            # Arrays are transformed into arrays:
            self.InputVoltageArray = numpy.array(self.InputVoltageList)
            self.OutputVoltageArray = numpy.array(self.OutputVoltageList)
            self.TimeArray = numpy.array(self.TimeList)
            self.MeasurementsArray = numpy.array(self.MeasurementsList)

        except:
            pass

    def PlotData(self):
        try:
            if len( self.InputVoltageArray) != 0:

                self.Graph.clear()
                self.Graph.plot(self.TimeArray,self.InputVoltageArray,
color='#A5F4FA',linestyle='solid', marker='o',markersize='4', label="V_in")
```

```
                self.Graph.plot(self.TimeArray,self.OutputVoltageArray,
color='#E3FA98',linestyle='solid', marker='x',markersize='4', label="V_out")
                self.canvas.draw_idle()
                self.SetGraphProperties()
        except:
            pass

    def SetGraphProperties(self):
        font = {'family': 'Arial Rounded MT Bold',
        'color':  'white',
        'weight': 'normal',
        'size': 12,
        }
        self.Graph.set_title('Comparación Voltaje de Entrada vs Salida', fontdict =
font)
        self.Graph.grid(color='#667B84')
        self.Graph.set_xlabel("Tiempo (s)",fontdict = font)
        self.Graph.set_ylabel("Voltaje (V)",fontdict = font)
        self.Graph.legend(["V_in","V_out"])
        self.Graph.grid(color='#667B84')
        self.GraphFigure.patch.set_facecolor('#465162')
        self.Graph.xaxis.label.set_color('white')        #setting up X-axis label
color
        self.Graph.yaxis.label.set_color('white')          #setting up Y-axis label
color
        self.Graph.set_facecolor('#465162')           #Grah Background color
        self.Graph.tick_params(axis='x', colors='#92C2E2',labelsize=10)    #setting
up X-axis tick color to red
        self.Graph.tick_params(axis='y', colors='#92C2E2',labelsize=10)  #setting
up Y-axis tick color to black
        pass

    def SetSliderLabel(self):
        if self.SignalType.get()=='step':
            self.InitialVoltage['label']= "Voltaje Escalón: "
            self.FinalVoltage.grid_forget()

            self.PeriodCoefficient.grid_forget()
            self.PeriodLabel.grid_forget()
            self.PeriodPrefix.grid_forget()

        elif self.SignalType.get()=='slope':
            self.FinalVoltage.grid(row=0, column=1)
            self.InitialVoltage['label'] = "Voltaje Inicial: "
            self.FinalVoltage['label'] = "Voltaje Final: "

            self.PeriodCoefficient.grid_forget()
            self.PeriodLabel.grid_forget()
            self.PeriodPrefix.grid_forget()

        elif self.SignalType.get()=='sine':
            self.FinalVoltage.grid(row=0, column=1)
            self.InitialVoltage['label'] = "Amplitud de Voltaje: "
            self.FinalVoltage['label'] = "Voltaje de Equilibrio: "

            self.PeriodCoefficient.grid(row=3, column=1, ipadx=0,ipady=0,)
            self.PeriodLabel.grid(row=3, column=0, ipadx=1, ipady=5, pady=5)
            self.PeriodPrefix.grid(row=3, column=2)
```

```
        elif self.SignalType.get()=='noise':
            self.FinalVoltage.grid(row=0, column=1)
            self.InitialVoltage['label'] = "Voltaje Mínimo: "
            self.FinalVoltage['label'] = "Voltaje Máximo: "

            self.PeriodCoefficient.grid_forget()
            self.PeriodLabel.grid_forget()
            self.PeriodPrefix.grid_forget()
        pass

if __name__ == '__main__':
    Window = Tk()
    application = IO_DAQS(Window)
    Window.mainloop()
```

(WorkingArea.py)

```
        #Creation date: 2022 / march / 3 / 22:48 h

from tkinter import *
from tkinter import ttk
from tkinter import Tk
import time
import os
from tkinter.tix import ComboBox

class Tab2Widgets:
    def __init__(self,):
        pass

    def CreateWidgets_Tab2(self,ParentFrame):
        self.CreateFrames(ParentFrame)
        self.CreateLabelFrames(self.FrameTab2)
        self.CreateLabels(self.FrameTab2)
        self.CreateSpinBox(self.MeasurementTime_LFrame)
        self.CreateComboboxes()
        self.CreateSliders()
        self.CreateRadiobuttons()
        self.CreateButtons()
        return self.FrameTab2

    def CreateButtons(self):
        #-------Buttons in Controls Label Frame ---------#
        self.RunButton= Button(self.Controls_LFrame, command=self.RunMeasurements,
text="Ejecutar", anchor='c', font=('Arial Rounded MT Bold', 9))
        self.RunButton.grid(row=0, column=0, padx=25)
```

```
        self.StopButton= Button(self.Controls_LFrame,
command=self.StopMeasurements, text="Detener", anchor='c', font=('Arial Rounded MT
Bold', 9))
        self.StopButton.grid(row=0, column=1,padx=8 )

        self.SaveButton= Button(self.Controls_LFrame,
command=self.SaveMeasurements, text="Guardar", anchor='c', font=('Arial Rounded MT
Bold', 9))
        self.SaveButton.grid(row=0, column=2,padx=8 )

    def CreateComboboxes(self):
        #-------Comboboxes in SamplingTime Label Frame----------#
        self.SamplingPrefix = ttk.Combobox(self.SamplingTime_LFrame,width=3,
value=['ks','s','ms','us'])
        self.SamplingPrefix.set('s')
        self.SamplingPrefix.grid(row=0, column=2)

        #-------Comboboxes in Signal Label Frame----------------#
        self.PeriodPrefix = ttk.Combobox(self.Signal_LFrame,width=3,
value=['s','ms','us'])
        self.PeriodPrefix.set('ms')

        #-------Comboboxes in Controls Label Frame--------------#
        self.SerialPort = ttk.Combobox(self.Controls_LFrame,width=6,
value=['com1','com2','com3','com4','com5','com6'])
        self.SerialPort.set('com3')
        self.SerialPort.grid(row=1, column=1, padx=10,columnspan=2)

    def CreateFrames(self,ParentName):
        self.FrameTab2 = ttk.Frame(ParentName, width='1200', height='600')
        self.FrameTab2.grid(row=1,column=0)

        self.ConfigurationsFrame = ttk.Frame(self.FrameTab2, width='1200',
height='600')
        self.ConfigurationsFrame.grid(row=1,column=0)


    def CreateLabels(self,ParentFrame):
        #-------Labels in FrameTab2-------------------------#
        self.Title=Label(ParentFrame,text = 'Área de Trabajo', padx=50, pady=20,
anchor='e', font=('Arial Rounded MT Bold', 20), relief="solid")
        self.Title.grid(row=0, column=0, columnspan=1)

        #-------Labels in MeasurementTime Label Frame-------#
        self.EndTimeLabel=Label(self.MeasurementTime_LFrame, text = 'Duración:
(s)', anchor='c', font=('Arial Rounded MT Bold', 9))
        self.EndTimeLabel.grid(row=0, column=0, ipadx=1, ipady=5, pady=5)

        #-------Labels in SamplingTime Label Frame----------#

        self.SamplingTime_Label=Label(self.SamplingTime_LFrame, text = 'Tiempo de
Muestreo:  ', anchor='c', font=('Arial Rounded MT Bold', 9))
        self.SamplingTime_Label.grid(row=0, column=0, ipadx=1, ipady=5, pady=5)

        #-------Labels in Signal Label Frame----------------#
        self.SignalTypeLabel=Label(self.Signal_LFrame, text = 'Tipo de Señal:  ',
anchor='w', font=('Arial Rounded MT Bold', 9))
```

```
        self.SignalTypeLabel.grid(row=0, column=0, ipadx=1, ipady=5, pady=5,
columnspan=2)

        self.PeriodLabel=Label(self.Signal_LFrame, text = 'Periodo:  ', anchor='w',
font=('Arial Rounded MT Bold', 9))

        #-------Labels in Controls Label Frame-----------------#
        self.PortLabel=Label(self.Controls_LFrame, text = 'Puerto serial:  ',
anchor='w', font=('Arial Rounded MT Bold', 9))
        self.PortLabel.grid(row=1, column=0, ipadx=1, ipady=5, pady=5,
columnspan=2)

    def CreateLabelFrames(self,ParentFrame):
        self.MeasurementTime_LFrame = LabelFrame(self.ConfigurationsFrame,
width=300, height= 200, text="Tiempo de Medición",
        font=('Arial Rounded MT Bold', 10), labelanchor= "n", relief="solid")
        self.MeasurementTime_LFrame.grid(row=0, column=0, padx = 40, pady=15,
ipadx=20,ipady=1)

        self.SamplingTime_LFrame = LabelFrame(self.ConfigurationsFrame, width=300,
height= 200, text="Razón de Muestreo",
        font=('Arial Rounded MT Bold', 10), labelanchor= "n", relief="solid")
        self.SamplingTime_LFrame.grid(row=1, column=0, padx = 40, pady=15,
ipadx=20,ipady=5)

        self.Signal_LFrame = LabelFrame(self.ConfigurationsFrame, width=300,
height= 200, text="Careacterísticas de Señal",
        font=('Arial Rounded MT Bold', 10), labelanchor= "n", relief="solid")
        self.Signal_LFrame.grid(row=2, column=0, padx = 40, pady=15,
ipadx=20,ipady=5)

        self.Controls_LFrame = LabelFrame(self.ConfigurationsFrame, width=300,
height= 200, text="Controles de Operación",
        font=('Arial Rounded MT Bold', 10), labelanchor= "n", relief="solid")
        self.Controls_LFrame.grid(row=3, column=0, padx = 40, pady=15,
ipadx=20,ipady=5)

        self.MatplotlibGraph_LFrame = LabelFrame(ParentFrame, width=400, height=
400, text="Datos Leídos: ",
        font=('Arial Rounded MT Bold', 10), labelanchor= "n", relief="solid")
        self.MatplotlibGraph_LFrame.grid(row=1, column=1, padx = 40, pady=15,
ipadx=20,ipady=5)


        #-------Frames in Signal Label Frame----------#
        self.SlidersFrame = ttk.Frame(self.Signal_LFrame, width='240', height='20')
        self.SlidersFrame.grid(row=4,column=0, columnspan=3)

    def CreateRadiobuttons(self):
        #-------Radiobuttons in Signal Label Frame----------#
        self.SignalType = StringVar()
        self.SignalType.set("step")

        self.StepInput =
Radiobutton(self.Signal_LFrame,command=self.SetSliderLabel, text='Escalon',
value='step', variable=self.SignalType)
        self.StepInput.grid(row=1, column=0 )
```

```
        self.SlopeInput =
Radiobutton(self.Signal_LFrame,command=self.SetSliderLabel, text='Rampa',
value='slope', variable=self.SignalType)
        self.SlopeInput.grid(row=1, column=1)

        self.NoiseInput =
Radiobutton(self.Signal_LFrame,command=self.SetSliderLabel, text='Ruido',
value='noise', variable=self.SignalType)
        self.NoiseInput.grid(row=2, column=0)

        self.SineInput =
Radiobutton(self.Signal_LFrame,command=self.SetSliderLabel, text='Seno',
value='sine', variable=self.SignalType)
        self.SineInput.grid(row=2, column=1)

    def CreateSliders(self):

        self.RightSliderCaption = 'Voltaje Escalón'
        self.FinalVoltage = Scale(self.SlidersFrame, length=130, from_=0, to=5,
orient='horizontal', resolution=.01, label=self.RightSliderCaption)


        self.LeftSliderCaption = 'Voltaje Escalón'
        self.InitialVoltage = Scale(self.SlidersFrame,length=130, from_=0, to=5,
orient='horizontal', resolution=.01, label=self.LeftSliderCaption)
        self.InitialVoltage.grid(row=0, column=0)

    def CreateSpinBox(self,ParentFrame):
        #-------SpinBoxes in MeasurementTime Label Frame-------#
        self.measurementString = StringVar()
        self.measurementString.set("1")
        self.MeasurementTime = Spinbox(ParentFrame,width=7, from_=0.0, to=1800.0,
textvariable= self.measurementString)
        self.MeasurementTime.grid(row=0, column=1, ipadx=0,ipady=0, padx=8)

        #-------SpinBoxes in SamplingTime Label Frame----------#
        self.samplingString = StringVar()
        self.samplingString.set("1")
        self.SampligCoefficient = Spinbox(self.SamplingTime_LFrame, wrap=True,
width=5,  from_=0.0, to=1800.0, textvariable=self.samplingString)
        self.SampligCoefficient.grid(row=0, column=1, ipadx=0,ipady=0, padx=8)

        #-------SpinBoxes in Signal Label Frame----------#
        self.periodString = StringVar()
        self.periodString.set("1")
        self.PeriodCoefficient = Spinbox(self.Signal_LFrame, wrap=True, width=5,
from_=0.0, to=1800.0, textvariable=self.periodString)
```

Firmware

El firmware de nuestro proyecto consta de básicamente 3 secciones generales, la primera es completamente de software, esta es la interfaz desarrollada en Python; la segunda es la tarjeta Arduino Uno (junto con el acondicionamiento), que debido a que conecta el lenguaje Python por medio de su programación con la tarjeta física de

Arduino, se puede considerar que es el puente entre la tercera sección la cual es nuestro circuito implementado (sistema).

Software de aplicación específica

Como tal el programa encargado de llevar el mayor desempeño a la hora de adquirir datos, es la GUI de Python ya que en ella se lleva a cabo la configuración de parámetros de inicio al igual que su escritura en la placa, también lleva a cabo la lectura del puerto, el guardado y acondicionamiento de los datos para después mostrarlos visualmente en gráficas y opcionalmente exportarlos en ficheros de distinta extensión.

Software de la etapa de comunicación

Como tal la comunicación es a través de una vía física, pero los programas que intervienen para poder llevarla a cabo es en sí la librería de python "*pyserial*" que permite la lectura y escritura de información a la placa arduino. Ya que esta última es la que recolecta la información del sistema al igual que proporciona las señales para evaluar la respuesta del mismo.

Sección de implementación

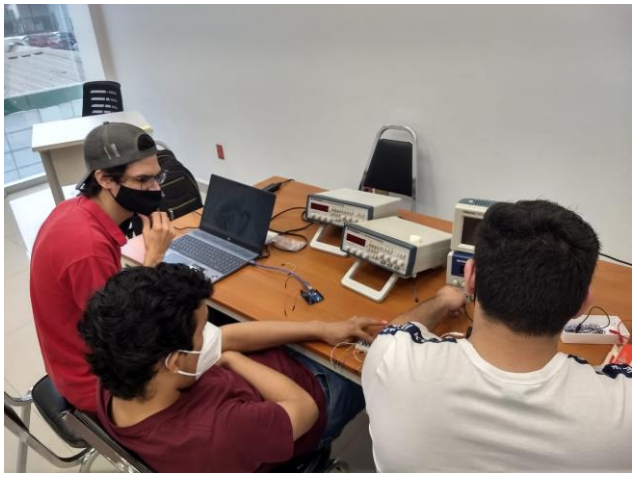

Ilustración 20: Equipo de trabajo en laboratorio de Mecatrónica (FIME)

Reunión en laboratorio de DAQ, se realizan las pruebas de los 3 tipos de señales para comprobar funcionamiento.

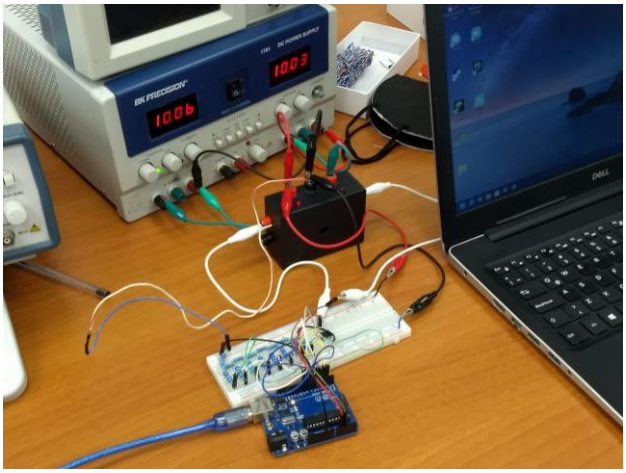

Ilustración 21: Reunión en laboratorio de DAQ, mediciones con el circuito

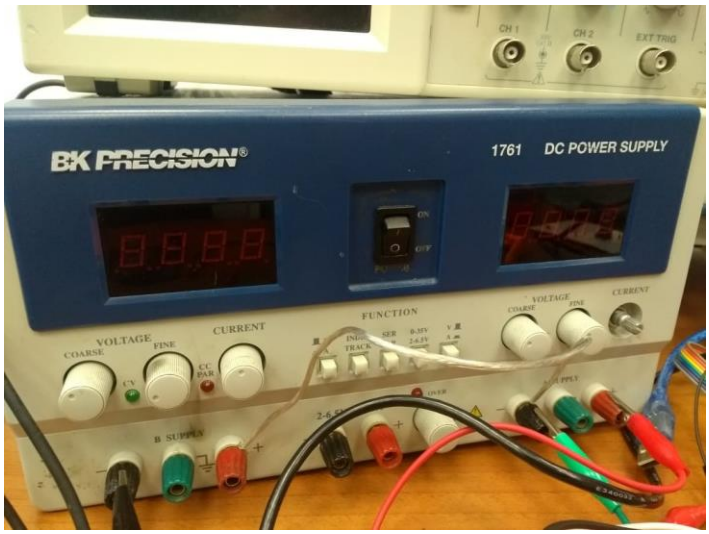

Ilustración 22: BK PRECISION 1761 DC POWER SUPPLY(Fuente Dual utilizada para la alimentación de nuestro circuito)

Ilustración 23: Tektronix TDS 2012 osciloscopio usado para verificar que la señal de salida entregada por el circuito sea el mismo que se muestra en la interfaz.

Ilustración 24: BK PRECISION 4017A 10MHz SWEEP/FUNCTION GENERATOR Generador usado para realizar pruebas básicas iniciales con el circuito

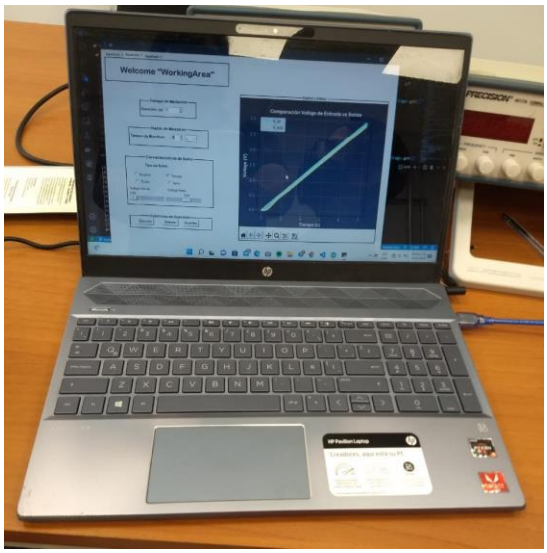

Ilustración 25: Reunión en laboratorio de DAQ; se puede observar el comportamiento de la señal de rampa(pendiente positiva)

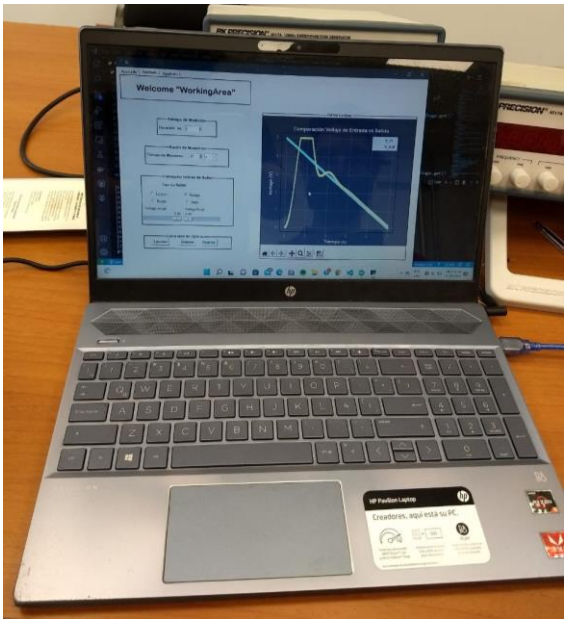

Ilustración 26: Prueba señal rampa, reunión en laboratorio de mecatrónica (FIME)

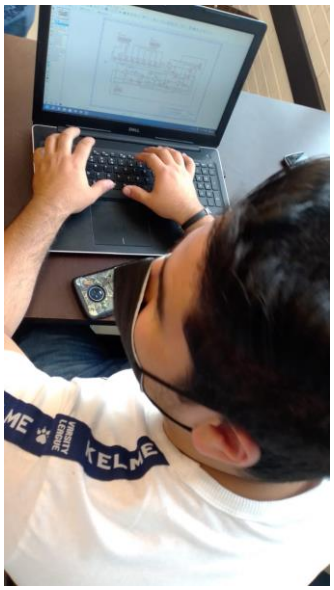

Ilustración 27: Diseño PCB

Sección de experimentación

A Continuación se muestran una serie de imágenes en las que se hicieron las primeras pruebas de validación y obtención de datos de la GUI; se muestran los datos de la señal (en esta ocasión solo de la señal escalón):

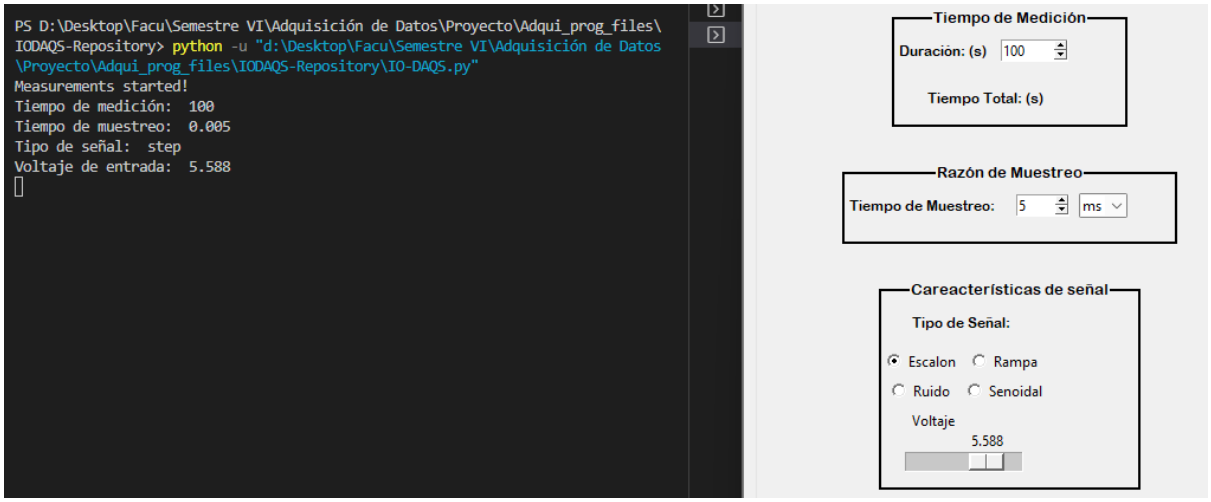

Ilustración 28: Primeras pruebas, obtención de datos de GUI

También se evaluó que si los datos se ingresan erróneamente como tiempos negativos o datos no numéricos, que notifique dicho error: (en la página siguiente se muestran imágenes alusivas a lo mencionado anteriormente).

Ilustración 29: Prueba detección de errores en datos

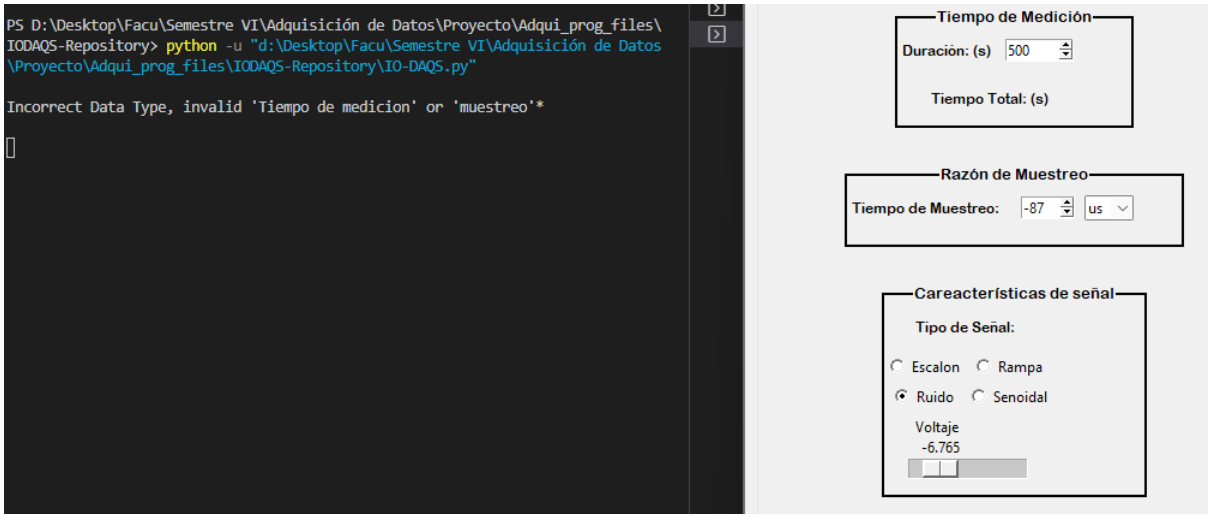

Ilustración 30: Prueba detección de errores en datos

Ilustración 31: Prueba detección de errores en datos

Sección de resultados obtenidos

Después de realizar las pruebas necesarias y de considerar como finalizado el desarrollo obtenido, los resultados obtenidos son aún más grandes de lo que en realidad se pensaba que se lograría, logrando incorporar en el código y en la interfaz la posibilidad de tomar 4 tipos de señales y el poder variar los parámetros logrando crear un ambiente en el que se puede experimentar de distintas formas.

Se logró generar una interfaz amigable para todo tipo de usuarios, que incluso en la opinión de los miembros del equipo es posible ser usada sin la necesidad de tener alguna introducción previa a está, claro que queda implícito que el usuario necesita tener conocimientos básicos para lograr diferenciar las señales y los parámetros que estas necesitan para funcionar.

La interfaz también cuenta gracias a librerías ya existentes en el software de Python la posibilidad de guardar las señales entregadas junto con sus respectivos tiempos, esto lográndose a las opciones que se encuentran en la parte inferior de la sección de gráfica. El fichero generado también cuenta con datos de las configuraciones de la prueba y los datos se encuentran en tercias de valores separados por comas para que puedan ser leídos fácilmente por otros programas

Seccion de analisis de resultado

Otro aspecto técnico muy importante y que es necesario resaltar, es que el tiempo de muestreo mínimo al que funciona el Arduino según datos del fabricante, es de 100 microsegundos. y el alcanzado en la práctica es de 1800 microsegundos. Es decir un 1800 % superior al ideal.

La razón del resultado es tal debido al modo de programación, pues ya que la comunicación es vía puerto serial, la información requiere un determinado tiempo para enviarse, y ese tiempo además está en función del tamaño de la palabra envíada (cantidad de caracteres y los bits de cada carácter) y del baudaje utilizado. Así que si se intenta aproximar este tiempo se tiene que:

$$T.M. = Palabra * [caracter] * \left[\frac{bits}{caracter}\right] * baudaje \left[\frac{bits}{s}\right]$$

En donde T.M. se refiere al tiempo de muestreo, la palabra indica la cantidad de caracteres enviados por puerto serial, y de ahí las unidades que la describen (caracteres). Se debe contemplar la cantidad de bits por carácter al igual que la velocidad de envío de bits por segundo, es decir el baudaje. Con esto se puede plantear una buena aproximación.

Atendiendo al tamaño de la “palabra”, se debe considerar que los datos enviados por puerto son 3 por muestra: voltaje de entrada, voltaje de salida y tiempo de medición. En el caso de los voltajes los valores enviados van del 0 al 1023 ya que dependen de los bits del ADC, por lo que aquí ya se cuenta una extensión máxima de 4 caracteres.

Por otro lado el tiempo se mide en microsegundos, y los tiempos más comunes usados en todas las pruebas van de 1 a 30 segundos, por lo que su equivalencia en microsegundos es de 30,000,000 us; que en caso máximo se tiene un tamaño de palabra de 8 caracteres.

Estos valores se envían por el puerto serial a modo de cadena de texto en donde además se agregan saltos de línea; por lo que a cada palabra se le deben sumar 2 bytes, ya que automáticamente se agrega "\r\n", en donde r es el retorno y \n el salto de línea (1 byte = 8 bits). Ahora bien también se agrega un último término nulo que ocupa un byte más. Este término indica el fin de la cadena de texto. Finalmente se considera que cada carácter está constituido por 8 bits; de estos datos se puede determinar aproximadamente el tiempo que toma enviar cada uno por puerto serial e incluso un valor total de la muestra que involucra a los 3 datos. Así que se procede:

Tiempos de impresión:
Voltaje de entrada:

$$T.I._{min} = \frac{(4+2+1) * [caracter] * \left[\frac{8\ bits}{caracter}\right]}{115200\left[\frac{bits}{s}\right]} \approx 4.86 * 10^{-4}\ s \approx 486\ \mu s$$

Voltaje de salida:

$$T.I._{min} = \frac{(4+2+1) * [caracter] * \left[\frac{8\ bits}{caracter}\right]}{115200\left[\frac{bits}{s}\right]} \approx 4.86 * 10^{-4}\ s \approx 486\ \mu s$$

Tiempo de medición:

$$T.I._{min} = \frac{(8+2+1) * [caracter] * \left[\frac{8\ bits}{1\ caracter}\right]}{115200\left[\frac{bits}{s}\right]} \approx 7.63 * 10^{-4}\ s \approx 763\ \mu s$$

La suma de estos tiempos indica el tiempo mínimo de muestreo el cual es el siguiente:

$$T.M._{min} = (486\ \mu s) * 2 + 763\ \mu s = 1736\ \mu s \cong 1800\ \mu s$$

Y como se aprecia, es muy próximo al obtenido experimentalmente. Además esto también sugiere que a tiempos mayores de medición probablemente el tiempo de muestreo tienda a ser mayor ya que el tamaño de la palabra/valor aumenta; y de forma inversa si se trabajan tiempos menores, el tiempo de muestreo será menor. A fin de cuentas se vería afectado el desempeño de las lecturas.

Cabe mencionar que fue la lógica de programación la que produjo estos resultados, es decir, enviar los datos a cómo se leían; pero una alternativa implica usar arreglos (vectores), en los que se van guardando las lecturas, a modo que la impresión o envío de datos ocurra una vez acabe el tiempo de medición. Esta opción reduce el tiempo de muestreo significativamente ya que las lecturas están en ejecución libre, pero la restricción reside en la memoria del microcontrolador, que al ser limitada, impide tener tiempos de medición prolongados.

De esta forma el método usado garantiza tener un tiempo de muestreo algo superior pero el tiempo de medición se puede prolongar considerablemente ya que ahora esos datos pasan a guardarse en el ordenador

Apéndices

Colocar el link del github**:** **https://github.com/IODAQS-Experts/IODAQS-Repository.git**

Bibliografía:

electronica, W. (12 de enero de 2017). *Conversor digital analogico R-2R*. Recuperado el 25 de mayo de 2022, de wilaebaelectronica.blogspot.com: https://wilaebaelectronica.blogspot.com/2017/01/conversor-digital-analogico-r-2r.html