Manual del usuario para el programa de termogravimetrías

Instituto Balseiro - Laboratorio Avanzado - Agustín Bernardo

3 de mayo de 2019

Índice

1. Arquitectura del programa y generalidades

1.1. Uso del programa

Para usar el programa, es necesario inicializarlo a través de Python. En este momento, esto se hace abriendo una ventana de comandos, yendo hasta la carpeta donde se encuentra el archivo *interfazContinua2.py*, y corriendo el comando *python interfazContinua2.py*. Esto inicializa todo el programa, y aparece la interfaz gráfica. Desde aquí se puede rehacer un experimento o programar uno nuevo.

Para programar un experimento hay que ir a la ventana **Experimento**, escribir las cosas necesarias en los campos de texto a rellenar, y tocar el botón **Añadir**. Se añadirá un subexperimento o módulo a la lista. Se pueden encadenar los subexperimentos de forma indefinida. Para ejecutarlo, solo hace falta presionar **EJECUTAR!**. Se pueden observar los gráficos en la pestaña **gráficos**. Por último, algunos parámetros de funcionamiento pueden modificarse en la ventana **mediciones**, especialmente aquellos relacionados con el PID.

Por preguntas acerca de como utilizar el programa, o por cualquier cambio relacionado, comunicate con:

- Mail: bagustin.sfe@gmail.com
- Teléfono: +54 342 155 29 34 35
- Nombre: Agustín Bernardo

1.2. Arquitectura de programa

El programa se encuentra compuesto de cuatro bloques (o hilos) fundamentales:

- El hilo **maestro/GUI** (representado por la clase masterThread y la clase GUI)
- El hilo **PID** (representado por la función loopPID, dentro del masterThread)
- El hilo **Mediciones** (representado por la función chequeaSiEjecuta, dentro de la clase masterThread)

Para correrlo, se llama simplemente a la función mainFunction(), que crea un objeto de tipo *Tk()*, el cual inicializa el cliente gráfico Tkinter, y un objeto *masterThread* que se ocupa de la coordinación de todo el programa.

A continuación se detallan las funciones que pertenecen a cada clase y se explica el funcionamiento de cada una de ellas.

2. subExp

De alguna forma, el corazón de todo esto está acá, en la clase de subexperimentos. Es el bloque fundamental que construye cada experimento a realizarse, se ocupa de comunicarse con los archivos, de pensar que variables setear, y de cuando decirle a la presión que se controle.

2.1. init

El inicializador de cada objeto de tipo subexperimento. Cada vez que creamos ingresamos un subexperimento a la lista de experimentos de *experimentoVariable*, pasamos por esta función. Le pasamos la cola de presiones, *queuePressure*, para que se comunique con el PID. Le cargamos todas las variables que obtuvimos de las variables de estado de la interfaz gráfica (tiempo total, tiempo paso, presión inicial, presión final, los checkbox de las válvulas). Se le inicializa el diccionario de condición de salida. Además, se inicializan vectores de medición que son auxiliares al cálculo de las derivadas. Se da un valor a la cantidad de puntos que se toman en el cálculo de cada derivada. Por último, a partir de qué flujos tenemos, los valores de presión y demás, se setean las válvulas que van a estar abiertas a lo largo del subexperimento.

2.2. actualizaSetPoints

La variable *experimentoVariable* llama a esta función desde el loop de ejecución para actualizar los valores que se van a setear, en función del paso de tiempo. Nada del otro mundo.

2.3. seteaVariables

Esta función se ocupa de enviar los setpoints a los distintos controladores: temperatura y flujos. Además, si se cumple que el subexperimento está en un tiempo de control (en este momento, cada seis segundos), se envía el valor a setear de la presión al control PID que se encuentra alojado en el otro hilo.

2.4. seteaPresion

Esta función coloca un vector de cuatro elementos en la cola de presiones, *queuePressure*. Estos cuatro elementos son la presión actual, la presión seteada, la derivada de la presión y el flujo, los cuales se utilizan en el PID para controlar.

2.5. mideVariables

Esta función llama a las funciones de la librería *funcionesBalanza* para medir la temperatura, las presiones alta y baja, y los flujos. Además, si estoy en la balanza, mide la masa. Por último, añade los valores medidos a los vectores auxiliares utilizados para calcular las derivadas.

2.6. imprimeArchivo

Esta función escribe al archivo *file* que se pasa como argumento todos los valores de variables instantáneas medidas y seteadas (temperaturas, flujos, masa, caudales, etc).

2.7. condicion

Esta función se ocupa de calcular la condición de salida. En función del diccionario de condiciones se fija cuales son las que nos permiten terminar el experimento. Funciona como *and*, es decir, con que se cumpla una **el subexperimento termina**. Está conformada por una serie de ifs. El diccionario tiene la siguiente forma: ["t": (1), "T": (0, 0), "P": (1, 120), ...]. Eso quiere decir, por ejemplo, que se puede salir por tiempo máximo, no se puede salir por temperatura, se puede salir si la presión - presión final es menor que 120, etc.

Devuelve 1 si se cumple o 0 si no.

2.8. mideDerivada

Esta función utiliza la librería de *numpy* para ajustar una recta con los últimos *puntosDerivada* datos, y nos permite calcular la derivada de la presión, temperatura o masa respecto al tiempo. Para ello crea un *nparray* o vector de numpy con el tiempo y con la variable que se busca derivar. Luego, se calcula un vector de coeficientes con la función *np.polyfit*. Por último, la pendiente (o el primer coeficiente) es el valor de la derivada calculada. En el caso de que no tengamos suficientes puntos, se devuelve un valor de derivada absurdo (elegí 10000) para evitar problemas.

2.9. vectorVariables

La función *vectorVariables* construye un diccionario donde se vinculan las variables con el valor medido. Esto es utilizado dentro del loop de *ejecuta* dentro de *experimentoVariable* para graficar con la función *graficaGraficosVentanaGraficos*. Podría ser una ensalada peor.

2.10. seteaValvulas

Esta función se llama apenas se inicializa el subexperimento. Se ocupa de decidir el vector de válvulas que van a quedar abiertas:

- si hay flujo n-ésimo (o está el checkbox activo) abre las dos válvulas del controlador n-ésimo
- si se activa el checkbox de las válvulas Gas-Horno, Gas-Venteo u Horno-Venteo, se abre la válvula correspondiente
- si la presión inicial o final tienen los valores adecuados, se activa la válvula de la válvula de aguja (¡qué trabalengüas!)
- otros casos, subexperimentos especiales como el que se usa para setear el cero o para una purga, quedan planteados acá para posteriori

2.11. overrideaValvulas

Esta función overridea lo anterior, le enchufamos un vector arbitrario y rogamos que funcione. Queda como una función a la que puede tener acceso un superusuario...

2.12. cambiaValvulas

Esta función se ocupa de activar o cerrar las válvulas que son necesarias. Para ello lee cuales están activas actualmente y mira las diferencias.

2.13. mideDerivadaInicial

Esta función calcula de la misma forma que *mideDerivada* la derivada de la función, pero en vez de tomar los últimos *puntosDerivada* puntos, toma los primeros. Sirvió para la calibración de la válvula de aguja. Está *deprecated*, pero en un futuro puede ser útil.

2.14. tiempoControl

Esta función se utiliza para saber si es momento de controlar la presión o no. Por ahora, simplemente devuelve 1 si el tiempo del subexperimento es divisible por 6, y 0 sino.

3. Exp

Los objetos de clase *Exp* representan a un experimento completo. Este se conforma por bloques o módulos, los *subExp* o subexperimentos. Esta clase se ocupa de ejecutar los experimentos. La clase experimento, además, lleva el tiempo total y tiene la posibilidad de ejecutar subexperimentos especiales, como el que fue utilizado para el control de la presión y demás.

3.1. init

La función *init* se utiliza cuando se crea un objeto de tipo *Exp*, tal como *experimentoVariable*, el utilizado en la interfaz gráfica. Esta función inicializa una lista de subexperimentos vacíos (donde la función *añadeSubExp* de la GUI incluye los subexperimentos), el tiempo (en 0), el flag de corrida de experimento, el nombre de archivo donde se graban los datos, y todos los flag relacionados con correr los experimentos especiales. Además, se inicializa la cola con la que se envía la información a las GUI para actualizar los gráficos.

3.2. anadeSubExp

Función que se ocupa de añadir un subexperimento a la lista. Se utiliza desde la función homónima en la GUI.

3.3. ejecuta

Esta función es la que se ocupa de llevar a cabo el experimento. Intenta setear un nombre de archivo, lo abre, y para cada subexperimento en la lista:

- cambia las válvulas a la posición correcta, preseteada en el subexperimento
- mide el tiempo inicial
- inicializa un loop que solo se termina cuando se cumpla la condición de salida del subexperimento o se llame a la función *saltaSubExp*

  se empieza a medir el tiempo

  se miden las variables analógicas (con la función *mideVariables* del subexp)

  se setean las variables analógicas (con la función *seteaVariables* del subexp)

  imprime los valores medidos al archivo

  actualiza los valores a setear en la próxima iteración

  pone en la cola los valores que se midieron y el tiempo, para enviar a la interfaz gráfica

  termina de medir el tiempo

  espera el tiempo que tenía que esperar, menos el tiempo que pasó midiendo
- una vez que termina el loop, se suma al tiempo total del experimento el tiempo transcurrido en el subexperimento

3.4. ejecutaPresion

De forma muy similar a la función anterior corre un experimento especial que sirve para caracterizar la válvula de presión. Como esto se utilizó y se dejó de lado, no vale la pena describirlo en profundidad. Dejo el código en el archivo por si algún día se necesita sacar algo de allí.

3.5. ejecutaInicializador/definedNdp/buscaPosicionValvula/...

De la misma forma que antes, el inicializador es el experimento que busca determinar la posición inicial de la válvula, con la finalidad de vincular el tiempo característico de evolución de las exponenciales con la presión de equilibrio y la mar en coche. Queda *deprecated*. El resto de las funciones cae en la misma consideración.

4. GUI

Toda la interfaz gráfica está programada en Tkinter, que es una libreria que viene de forma estándar con python y anda bastante bien para casi todos los usos. Cada objeto dentro de la aplicación, cada botón, gráfico, texto, checkbox o lo que sea es un *widget*. Estos widget se posicionan de forma matricial a través de la función *grid*. Además, pueden estar vinculados a una variable (lo cual se utiliza a veces cuando necesitamos conocer el estado de un checkbox), o a una función (cuando presionamos un botón y queremos que la ejecuta), y tienen propiedades que van desde texto a elementos dentro del mismo widget (en el listado de subexperimentos).

4.1. init

La función *init* se utiliza cuando se inicializa un objeto de tipo GUI. Esta inicializa el experimento, a través de la variable *experimentoVariable*, asigna las colas de medición y presión, *queue* y *queuePressure* a la interfaz gráfica, el hilo o núcleo de Tkinter a la raíz de la interfaz, crea una *notebook* llamada nb donde se agregan las pestañas de historial, mediciones, experimentos y gráficos (*logs, med, exp, grp*), inicializa las variables de estado dadas por *inicializaVariables*, arma las pestañas mencionadas anteriormente y configura algunos aspectos gráficos (el padding en x y en y de los objetos.)

4.2. armaVentanaMediciones

La ventana de mediciones o panel de control tiene muchos *widgets* (botones, entrada de texto, variables, labels) que mostrar y todos se inicializan en esta función. Las funciones que empiezan con *ttk.* son aquellas que agregan widgets a las ventanas. No hay mucho más que decir al respecto.

Abajo de todo se incluyen entradas para los parámetros del PID. Modifique bajo su propio riesgo.

4.3. armaVentanaExperimentos

La ventanada de experimentos tiene un montón de *widgets* a inicializar. Básicamente, incluye:

- Un widget de notebook donde se alojan los gráficos para ver como va a ser el experimento
- Widgets de entrada de texto, labels, y checkboxes para añadir subexperimentos
- Un widget de tipo *treeview* para podeer mirar los subexperimentos ya programados
- Un widget de *progressbar* que nos permite saber por donde va el experimento

4.4. armaVentanaGraficos

La función *armaVentanaGráficos* tiene varios widgets, que se ocupan de alojar los gráficos y los valores de las variables para las mediciones.

4.5. armaVentanaLogs

Esta función crea un widget de tipo *treeview* que aloja toda la información de mediciones pasadas, las cuales se guardan en un archivo llamado *experimentoLogs* y está en la carpeta de raíz del programa. Si uno limpia este archivo, asimismo se limpia el Log de experimentos. Por como está programado, hay que ver que esto funcione bien: a veces queda una línea vacía al principio (y en principio se puede arreglar o hacer más robusto).

4.6. inicializaVariables

La función *inicializaVariables* se ocupa de, valga la redundancia, inicializar todas las variables de estado que se utilizan en la interfaz gráfica. Estas tienen la particularidad de poder vincularse a entradas de texto, botones, y todo tipo de widgets. Más aún, para setear o leer el valor de estas variables se utilizan las funciones *get* y *set*. Hay de todos los tipos: vectores, variables *double*, *int*, *bool*. Aquellas que tienen un *value* definido tienen un valor inicializado.

4.7. actualizaValores

La función actualizaValores nos permite medir los valores de presión y temperatura en la ventana de experimentos. Para ello se utilizan las funciones de la librería *funcionesBalanza*, llamados *midePresionAlta*, *midePresionBaja*, *mideTemperaturaMuestra*, etc.

4.8. actualizaValoresMasa

Nota: esta es una mala elección de nombre de función. Esta función, al igual que la anterior, me permite medir los valores, esta vez de los caudales másicos.

4.9. botonVerGraficas

Es una función asociada al botón de la ventana de historial de experimentos, que abre la carpeta donde se alojan las gráficas obtenidas.

4.10. botonRepetirExperimento

Es una función que se encarga de cargar todos los subexperimentos que se utilizaron en un experimento dado. Para ello utiliza el archivo de configuración.

4.11. botonTemperaturaHorno

Una función muy sencilla que busca setear la temperatura del horno. Ver función *actualizaValores*.

4.12. botonTemperaturaBano

Idem anterior, pero para la temperatura del baño.

4.13. botonMFCn

Las tres funciones a continuación setean el caudal para el controlador de flujo enésimo. Están vinculadas con botones en la ventana de mediciones.

4.14. botonCheckboxFlujon

Son funciones que habilitan el uso de los distintos gases en la ventana de experimentos. Ocurre que para utilizar un gas, es necesario que tenga las válvulas del controlador de flujo abiertas, pero su recíproca no es cierta: a veces queremos mantener las válvulas de un gas abierto con la finalidad de poder establecer un flujo a posteriori en el programa.

4.15. botonCheckBoxCondicionX

Activa la condición X como posible salida para el subexperimento. Entre ellas, el tiempo, la presión final, temperatura, estabilidad de la presión o temperatura, masa, etc.

4.16. abrir/cerrarPC

Son funciones vinculadas a abrir o cerrar la válvula de control de la presión.

4.17. botonMasa

Es una función que nos permite leer el valor de masa que nos entrega la balanza.

4.18. limpiaSubExp

Función que se ocupa de borrar todos los subexperimentos que se encuentran en *experimentoVariable*. Borra los elementos del árbol y actualiza los gráficos.

4.19. saltaSubExp

Función vinculada al botón *saltar* de la ventana de experimentos. Permite saltear al siguiente subexperimento, dando la condición de salida 1.

4.20. modificaSubExp

Permite hacer modificaciones sobre un subexperimento dado. Para ello, lee todas las variables del subexperimento a modificar, y las sobreescribe con las que están dadas por los widgets de entrada de texto en la ventana de experimentos. Por último, actualiza el widget de árbol y los gráficos.

4.21. añadeSubExp

Añade un subexperimento dentro de la variable *experimentoVariable*, a la lista o vector de subexperimentos en su interior. Se tratará con mayor énfasis qué es el subexperimento en la sección correspondiente.

4.22. remueveSubExp

Función que se ocupa de quitar el subexperimento seleccionado en el árbol. Elimina el mismo de la lista de subexperimentos de *experimentoVariable*, lo quita del árbol y modifica los gráficos.

4.23. ejecutaExp

Función que setea el flag de ejecución en *experimentoVariable* a 1. Esto es leído luego por el hilo de *chequeaSiEjecuta*, lo que luego llama a la función *ejecutar* dentro del experimento. Así, se pone en marcha el proceso de medición. Este se detalla en la clase *Exp*.

4.24. ejecutaExpPresion

Igual que el anterior pero para el caso del experimento especial de caracterización de la válvula de aguja. Posiblemente, esté inutilizado en la versión final.

4.25. ejecutaDefinePosicion/BuscaPosicion

Idem anterior - este se utilizó para la medición del tiempo propio del sistema.

4.26. procesoLlegando

Esta función es la que se ocupa de obtener los datos del experimento y almacenarlo en las variables de estado. Esta función se corre desde el hilo maestro. Lo que hace es mirar cada una determinada cantidad de tiempo si hay algo en la cola *queue*. En caso de que si, saca lo que hay adentro, actualiza las variables de mediciones y actualiza los gráficos de la ventana de gráficos

4.27. drawfigure

Esta función se ocupa de dibujar las figuras. Esta copypasteada de StackOverflow - sencillamente le paso donde va a estar la imagen, la figura que quiero meter ahí, y me devuelve una "foto"que se mantiene en el tiempo.

4.28. guardaGraficosyArchivos

Esta función es la que se encarga de llevar el historial al día. Lo que hace es guardar en una carpeta (por autor, luego por día/hora, y por muestra) los archivos relacionados a la medición, y modificar el *experimentoLog*, para que quede registrado. Por último, setea todos los vectores que se usan para graficar a vectores vacíos, y frena la barra de progreso.

4.29. actualizaGraficosVentanaExperimento

Esta función utiliza la librería *matplotlib* para graficar las variables a medir en la ventana de experimentos. Para ello, genera vectores vacíos y los rellena con los valores que se esperan tener o setear en los subexperimentos. Luego, se generan figuras con las vectores correspondientes a las variables y se nombran los ejes. Por último, la función *drawfigure* se ocupa de colocarlas en el lugar que les corresponde.

4.30. actualizaGraficosVentanaGraficos

Es una función muy similar a la anterior, pero utiliza no solo los valores esperados para las variables sino que también los medidos. Utiliza vectores que contienen el historial de variables que se fueron midiendo, y cada vez que la función se llama, se agregan los actuales. Luego, repite el mismo proceso de armar la figura y dibujarla en el lugar correspondiente con *drawfigure*.

4.31. todoEnCero

Es una función que se llama cuando termina un experimento o se cierra el programa. Asegura enviar todas las válvulas al estado inactivo, y enviar todos los setpoints analógicos a cero.

4.32. condicionSalida

A la hora de agregar un subexperimento, las condiciones de salida se establecen a través de un diccionario. Esta función escribe el diccionario, vinculando cada posible condición de salida con un valor booleano y un valor double. El booleano nos dice si esta condición nos permite salir del subexperimento, y el valor double está relacionado al valor que debe tomar para hacerlo.

5. Clase *masterThread*

5.1. init

La función *init* es aquella que inicializa la clase. Cada vez que se crea un objeto de tipo masterThread, se imprime en la consola una frase de llegada (*"holu"*), se inicializan dos *queues* que permiten la comunicación entre los distintos hilos, se setea en 1 el flag de que el programa está corriendo (necesario para la función chequeoPeriodico), y se inicializan los hilos de la interfaz gráfica, la medición y el loop del PID.

5.2. chequeoPeriódico

Cada 200 ms esta función se fija si el hilo de medición envió la información medida a la interfaz gráfica, a través de la función *procesoLlegando*, que será cubierta en la sección de la GUI. Luego, se fija si el programa sigue corriendo. Caso contrario, lo cierra y mata el proceso.

5.3. kill

La función *kill*, que se ocupa de matar el proceso, es aquella que se llama al presionar el botón **salir** en la interfaz gráfica. Pone todo en cero antes de salir, lo cual es crítico para la seguridad del equipo.

5.4. chequeaSiEjecuta

Esta función nos permite llamar a los distintos bloques de ejecución del programa. Está dentro de un loop que se repite solo cada 0,1 segundos. Básicamente, cuando queremos correr el experimento que tenemos formado en la variable *experimentoVariable* dentro de la interfaz gráfica, levantamos un flag que el hilo de mediciones lee: luego, llama dentro de este hilo a la función *ejecuta* de la clase experimento, que nos permite correr el experimento, poner todo en cero y guardar la información. Además, baja el flag de la ejecución del experimento, para no correrlo dos veces.

La misma función permite correr experimentos especiales, tales como el que se utilizó para la caracterización de la válvula, o aquellos donde se medía el τ con el truco del pulso.

5.5. loopPresionPID

Esta función se ocupa de procesar el setpoint de la presión. Como tenemos controladores externos para todo menos para esto, esto virtualiza un controlador de tipo PID.

Lo que tenemos es un loop que se repite cada 0.5 segundos. Si hay algún elemento dentro de la cola *queuePressure*, que se ocupa de comunicar los elementos de otros hilos a este, abre ese elemento y se fija cual es su longitud. Si tiene 2 elementos, entonces es para el control manual de la presión (contiene [abrir/cerrar, tiempo]). Si tiene 4 elementos, entonces es para el control automático de la presión: (contiene [presionsetpoint, presionmedida, dp/dt, caudal]).

En el control manual simplemente abrimos o cerramos el tiempo que nos dicta el vector que llegó a través de la *queuePressure*. En el control automático, nos fijamos que esté activo el flag de control (que finalmente estará siempre activo), y nos fijamos que no estemos en el momento en el que estamos empezando a medir y no tengamos la derivada calculada (derivada ¡100). Entonces, calculamos el *deltaApertura*, con la fórmula de:

$$\Delta_{ap} = P(p - p_{sp}) + D\left(\frac{dp}{dt}\right)$$

Acá nos fijamos varias cosas. Si es mayor (menor) que un valor de tiempo de apertura máximo, t_{max}, puede que sea porque estamos muy lejos, y abrir y cerrar un montón es mala idea. Entonces, lo que hacemos es compararlo con un valor de derivada máxima (que es proporcional al flujo) o mínima (que depende de la presión que tengamos y la presión afuera): si no estamos en la condición de máxima velocidad de cambio de presión, entonces abrimos o cerramos el tiempo máximo, y si estamos en la condición de máxima velocidad, entonces no hacemos nada.

Luego, con el Δ_{ap} calculado, procedemos a abrir o cerrar la cantidad que corresponda. Si el Δ_{ap} está debajo de un valor umbral, ni movemos la válvula para no sobreactuar.

Los valores que pueden modificiarse desde el panel de control son P, I, D, t_{max}, dp/dt_{max} y Δ_{ap}^{min}.

Este proceso se repite cada 6 segundos, determinado desde la clase *subExp*. Podría cambiarse ese tiempo, en principio.