Rampa digital para el uso de una computadora

Goyret, Marcos

Universidad Tecnológica Nacional, Departamento de Electrónica

Abstract

El desarrollo de una persona puede verse condicionado por su contexto y por las oportunidades que tenga a lo largo de su vida. El ejercicio de derechos del colectivo de personas con discapacidad muchas veces se ve cercenado por la mirada con la cual se desarrolla la tecnología. Esta publicación aborda el desarrollo de una rampa digital basada en la emulación de las acciones de un mouse para favorecer el acceso y uso de programas que se encuentran en una computadora como también recursos de acceso vía internet. En particular, su diseño basado en clases y bajo licencia de software libre pretende generar un intercambio con la comunidad y retroalimentar el mismo a partir de las experiencias de los usuarios.

Palabras Clave

Rampa digital, emulación, discapacidad

Introducción

Hoy en día, vivimos en una sociedad de la información y comunicación. Las personas planifican y viven sus vidas dando por sentado el acceso a distintas herramientas, como, por ejemplo, la computadora. Pero la realidad es que hay personas a las que el acceso, en este caso, a una computadora, se ve restringido por dificultades físicas. Por lo tanto, lo que se busca es poner al servicio de estas personas el conocimiento técnico adquirido en la universidad, y buscar facilitarles el acceso a una herramienta tan útil como es la computadora. Particularmente, este proyecto busca permitir a una persona controlar el mouse de una computadora, emitiendo únicamente un estímulo con alguna parte de su cuerpo. La manera en la que dicho estimulo es vinculado eléctricamente entre el usuario y la aplicación, es externa a este desarrollo.

Desarrollo

Se desarrolló una aplicación consistente en una ventana principal apreciada en la Ilustración 1, que muestra al usuario las distintas sub-ventanas por las que puede navegar. Es necesario que el usuario este habilitado para visualizar la interfaz gráfica de la aplicación, ya que el uso de la misma está basado en la selección de opciones a partir de un barrido gráfico. Por lo tanto, si el usuario no pudiera ver la interfaz, no sabría en qué momento seleccionar las distintas opciones.
Podrá proceder a las distintas ventanas de control del mouse, o configurar una serie de opciones para adecuar la experiencia de uso como mejor le resulte.

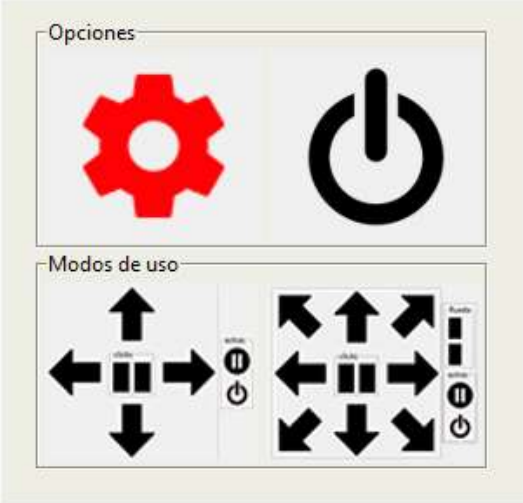

Ilustración 1 – Ventana principal

La aplicación recibe mediante sockets una señal generada por un dispositivo electrónico, el cual capta estímulos generados a partir de la interacción del usuario, y utilizando dicha información logra controlar un mouse, con un único tipo de señal.

La aplicación se realizó en lenguaje Python, creando una interfaz gráfica a través de la biblioteca “tkinter” [1]. A pesar de la gran cantidad de bibliotecas para desarrollo gráfico, se optó por tkinter ya que está incluida por defecto con la instalación de Python, y esto podría facilitar futuros desarrollos en equipo.

La aplicación consta de distintos hilos que se encargan de correr:

- Interfaz gráfica
- Servidor socket
- Controlador del mouse

Trabajando con threads de Python utilizando la biblioteca “threading” [2], el programa corre cada uno de estos módulos en paralelo, relacionados entre sí principalmente por un archivo común de configuración y variables globales.

La interfaz realiza un barrido de las distintas opciones seleccionables, en función de la ventana que esté abierta, y para seleccionar una opción el usuario debe enviar una señal en el momento en que la opción deseada esté marcada.

La función de la aplicación es que el usuario, emitiendo un único tipo de señal, es decir, realizando un único tipo de estímulo, pueda controlar un mouse de computadora sin asistencia de otros.

Una vez que todos los threads están ejecutados, la aplicación consume más recursos. Por lo tanto, para cargar todas las imágenes necesarias para la interfaz gráfica, el hilo principal carga todas las imágenes en distintas variables presentes en el archivo de configuración. De esta manera, una vez ejecutados los hilos, cuando se requiere mostrar una imagen no hace falta abrirla, si no únicamente mostrar una imagen precargada en una variable.

Luego de tener las imágenes cargadas, ejecuta los tres threads en paralelo antes de ejecutar la interfaz gráfica que bloquea la ejecución del main, ya que la función de la biblioteca “tkinter” que muestra la interfaz, es una función bloqueante. A continuación, detallamos los distintos threads que son ejecutados:

- **Thread retroalimentación auditiva**

Un thread del programa ejecuta un reproductor de audio que saluda en forma audible y brinda la bienvenida. Además, podrían agregarse retroalimentaciones durante el uso de la aplicación si así se deseara.

Se utilizaron dos bibliotecas:

1) “gtts” [3]
   a partir de la función “gTTS” genera un archivo de texto. Este texto está escrito en el código, y será lo que se reproducirá.
2) “playsound” [4]
   a partir de la función “playsound” reproduce en forma audible un archive de texto que se pasa como parámetro. Este archive es el generado anteriormente con “gTTS”

- **Thread socket**

El hardware generador de la señal se encarga de enviar una trama por sockets, a un puerto determinado. Luego, la aplicación tiene uno de los hilos encargado de escuchar constantemente dicho puerto, y recibir todos los paquetes de datos. El puerto a utilizar es configurable a partir del archivo de configuración.

La trama enviada consiste en 3 partes:

3) Inicio de trama: ‘#$’
4) Señal
5) Fin de trama: ‘$#’

Los inicio y fin de trama son para filtrar la posible basura que ingrese al socket de la aplicación. Es una verificación sencilla, ya que no es una comunicación muy compleja, por lo que se considera que no habrá mucho riesgo de ruido.

En cuanto a la señal, en principio habría un único tipo de señal cuando el usuario realiza un estímulo, por lo que sería un dato fijo. En caso de que el hardware permita enviar más de una señal, si el usuario está capacitado para realizar estímulos diferentes, podrían enviarse distintas señales, con distintos códigos cada una, e interpretar estas señales en la aplicación para realizar las acciones deseadas.

En base a las señales que entran, el thread se encarga de configurar una variable booleana

que indica el flanco ascendente y descendente de la señal enviada por el usuario. En la Ilustración 2 se puede apreciar gráficamente el camino que recorre una trama de datos recibida mediante el socket.
El trabajo de sockets se realizó con la biblioteca "socket" [5].

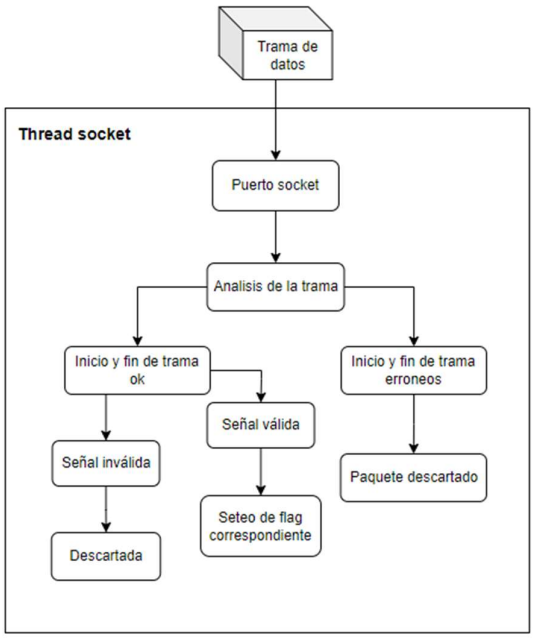

Ilustración 2 – Diagrama de flujo del socket

Podemos ver que, al recibir una trama, si los inicio y fin de trama no son correctos, o si la señal tiene un código no conocido, directamente se descarta la trama, y el thread socket no comunica nada a los otros threads. En cambio, si llega una señal correcta, se establecen los flags correspondientes.
El thread socket ejecuta una función miembro de una clase llamada "Server", y dicha función es la que queda ejecutando un loop que escucha el puerto. Luego, las otras funciones miembro de la clase se engargan de distintas tareas:

close_server()
Cierra el socket en caso de finalizar la aplicación.

check()
Se encarga de verificar las tramas recibidas por el puerto. Verifica el inicio y fin de trama, y guarda el dato de la señal en una variable.

- **Thread mouse**

Este último hilo se encarga de estar constantemente chequeando la variable booleana que indica los flancos, y si se recibe una señal, chequea cual opción de la interfaz gráfica está pintada en ese momento determinado del barrido. Si la acción coincide con abrir o cerrar una ventana, la aplicación muestra/esconde las ventanas correspondientes. Si el usuario esta dentro de la ventana de configuración y la acción es una opción de configuración, entonces se modifican los datos pertinentes, que serán variables presentes en el archivo de configuración. Si se está en la ventana de uso del mouse, en base a la acción deseada, la biblioteca "pyautogui" [6] realiza las acciones del mouse.
El archivo de configuración contiene variables globales que se modifican durante la ejecución del programa, y algunas constantes definidas como dimensiones de la interfaz gráfica, velocidades del movimiento del mouse y barrido, etc. Estas modificaciones puede realizarlas el usuario mediante la ventana de configuración apreciada en la Ilustración 5.
El tread mouse también ejecuta una función miembro de la clase "Mouse", que consistirá en el loop que analiza el barrido. Luego, hay funciones miembro de dicha clase que se encargan de realizar las distintas acciones:

actualizar_velovidad
Modifica el valor de la velocidad del puntero si el usuario realiza esta configuración

mv_mouse
Ejecuta el movimiento del mouse mientras que el usuario envía el estímulo. La función utilizada es pyautogui.move()

sc_mouse
Realiza el "scrolling" o deslizamiento mientras que el usuario este seleccionando la opción de rueda. La función utilizada es pyautogui.scroll()

Al ejecutar la aplicación, el usuario puede cerrarla, ejecutar la ventana de configuración, o seleccionar el modo de uso.

Modos de uso (seleccionable directamente desde la ventana principal)
Entre las opciones de uso de la aplicación se encuentra el modo simple (Ilustración 3) y el avanzado (Ilustración 4). El primero de ellos permite una cantidad limitada de opciones para direccionar el mouse, y el segundo modo adiciona movimientos diagonales, y la posibilidad de presionar los botones de rueda. Todo esto es ejecutado por el thread mouse, utilizando la biblioteca previamente mencionada.

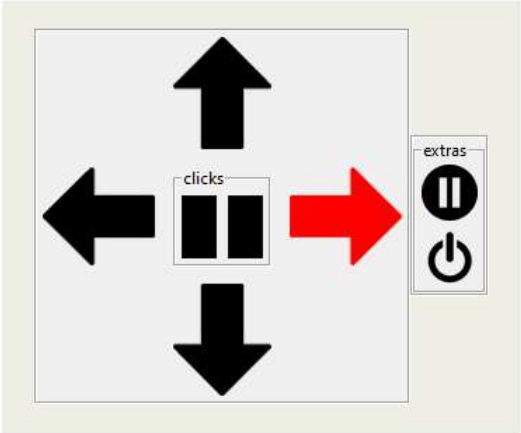

Ilustración 3 – Modo de uso simple

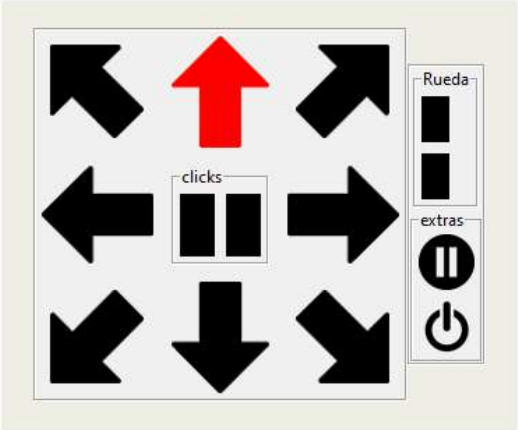

Ilustración 4 – Ventana de uso avanzado

Al momento de elegir el modo de uso a partir de la ventana principal, se genera una ventana secundaria, la ventana de control del mouse, y en base a la opción elegida hay una serie de "if" en el código que se encargan de cargar o no las imágenes de las flechas diagonales, y los botones de rueda del mouse.
Si en lugar de seleccionar un modo de uso, el usuario ingresa en la ventana de configuración, se encontrará con las siguientes opciones que pueden verse en la ilustración 5.

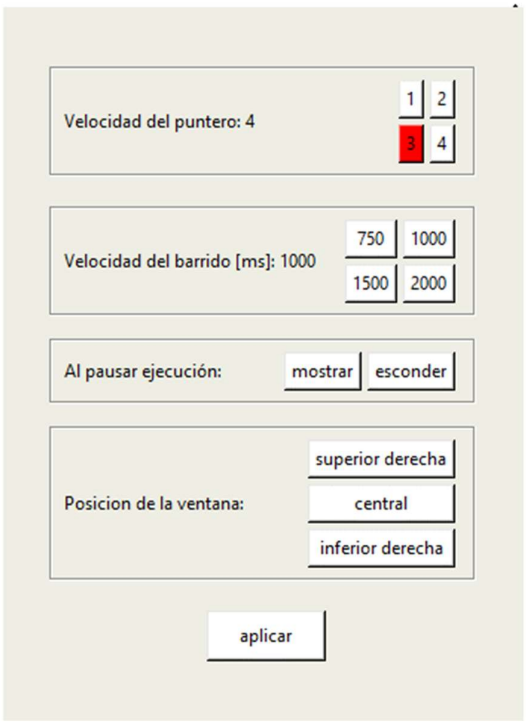

Ilustración 5 – Ventana de configuración

- **Velocidad del puntero y de barrido**

Es posible configurar las velocidades a las que se realizan el barrido de opciones, y el movimiento del puntero.

- **Pausa**

Mientras se usa la aplicación, el usuario podría desear pausar el barrido. Es posible simplemente pausar la ejecución de la aplicación si el barrido molestara visualmente, e inclusive esta la opción de esconder la aplicación mientras la misma se encuentre pausada. Esto sirve, por ejemplo, si se está reproduciendo un video y la ventana obstruye la visita. Para reanudar la ejecución, el usuario simplemente envía el estímulo único, en cualquier momento, y la ejecución será continuada.
Para elegir si la aplicación debe esconderse o no, simplemente se selecciona la opción deseada en la ventana de configuración.

- **Posición de la ventana**

La ventana principal y la de configuración, siempre son ejecutadas en el centro de la pantalla, pero existe la opción de seleccionar tres distintas posiciones para la ventana de control del mouse. Esto se debe a que, al usar la computadora, la ventana de control en el centro de la pantalla podría resultar incomoda.

Todas estas configuraciones trabajan mediante flags globales presentes en el archivo de configuración.

En la Ilustración 6 se puede apreciar gráficamente la interacción entre los distintos threads y el archivo de configuración mediante el diagrama de clases de la aplicación. Vemos la inicialización del main antes de que arranquen los threads, para mayor velocidad. Luego, los tres threads mencionados previamente:

- **Retroalimentación auditiva** (no Ilustración en el diagrama, ya que sumplemente ejecuta una función y termina)
- **Socket**
- **Mouse**

Luego de iniciar dichos threads, ejecuta la interfaz gráfica mediante la función bloqueante hasta que muera dicha interfaz.

Por un lado, el thread socket queda escuchando al puerto correspondiente, y trabajando como fue mencionado previamente. Al recibir una señal, modifica un flag del archivo de configuración indicando que hubo una entrada.

Por otro lado, cada línea roja representa una acción ante un cambio del flag de entrada.

El thread de la GUI queda mostrando y ocultando las distintas ventanas según corresponda, barriendo las distintas opciones, consultando y modificando el archivo de configuración según se modifiquen las opciones de uso como velocidad del puntero o del barrido.

Por último, el thread Mouse queda controlando el flag de entrada, y el estado del barrido de la GUI, para ejecutar funciones de control del mouse según corresponda.

De esta manera, queda el thread principal, es decir el main, y los threads secundarios del socket y el mouse, ejecutados en forma paralela, sincronizados mediante el archivo de configuración, y los métodos de las clases.

El código del trabajo realizado se puede acceder a partir del enlace adjunto en la sección de referencias, correspondiente a un "repositorio de GitHub" [7].

Se opto por trabajar mediante "entornos virtuales" [8], para tener mayor organización en cuanto a las versiones usadas de las distintas bibliotecas.

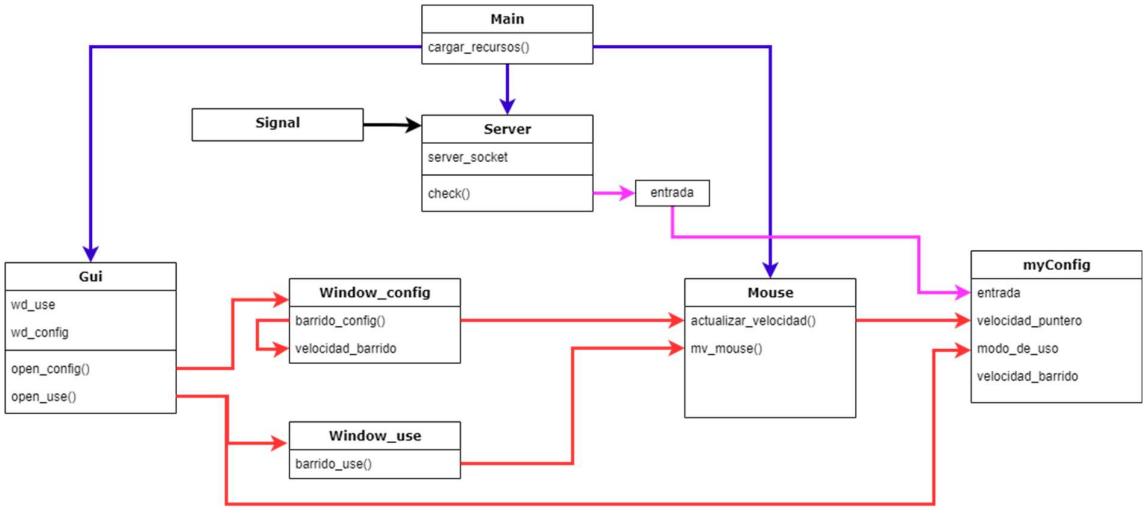

Ilustración 6 - Diagrama de clases

En la sección de referencias se puede encontrar un enlace a un "video demostrativo" [9] de la aplicación en uso.

Software emulador

Para el proceso de desarrollo, se generó una aplicación secundaria que simula el dispositivo electrónico, enviando por socket una señal que recibe la aplicación principal. De esta manera, todo el equipo puede probar el software sin necesidad de tener disponible el hardware.

Este emulador corre en paralelo a la rampa digital, y consiste en un loop que realiza la técnica de sondeo, consultando si se presionó una tecla determinada del teclado de la computadora. Para esto se utiliza la biblioteca "keyboard" [10]. En cuanto a la tecla usada, para el testeo debe presionarse la tecla "f7", que no obstruye con el uso de la computadora.

Tanto para la aplicación principal como para el emulador, se utilizaron comandos de "pyinstaller" [11] para poder generar archivos únicos ejecutables. De esta manera, el usuario no debe ejecutar comandos por consola para utilizar la aplicación.

Son aplicaciones portables, es decir no requieren instalación, alcanza con tener las carpetas con los archivos necesarios, y un acceso directo al ".exe" correspondiente.

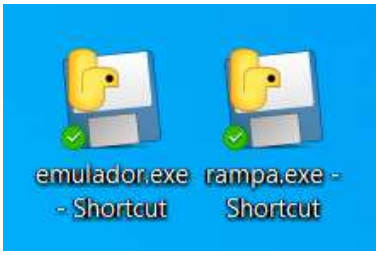

Ilustración 7 – iconos ejecutables de las aplicaciones

Conclusión y Trabajos Futuros

Para una persona que no tiene la capacidad física de poder sostener un mouse, desplazarlo y utilizar los botones, esta aplicación brinda una gran alternativa que le permite valerse de la computadora.

Vale aclarar que un usuario sin necesidad de utilizar esta rampa digital, al probar la aplicación podría sentir que no es un método lo suficientemente cómodo como para usarlo cotidianamente. Esto se debe a que en lugar de utilizar la computadora en la forma fluida a la que está acostumbrado, la velocidad de uso es mucho menor. Pero para un usuario que no tiene posibilidad de usar la computadora de la forma tradicional, esta podría ser una de las pocas herramientas que se lo permitan. La rampa está pensada únicamente para estos casos, y no pretende de ninguna manera reemplazar el uso tradicional en los casos en que el usuario estuviera habilitado para el mismo.

Hay principalmente dos aspectos que podrían mejorarse en la aplicación en futuros desarrollos:

- Mayor fluidez del movimiento del puntero
- Una GUI gráfica más moderna y amigable

Además, seria de mucha utilidad agregar un módulo que permita al usuario utilizar un teclado en pantalla controlado por la misma aplicación, para ampliar sus posibilidades frente a la computadora.

Con respecto al trabajo de desarrollo y testeo, en un futuro sería de gran ayuda desarrollar un emulador por hardware, para así no precisar del teclado de la computadora para poder probar la aplicación. Además, sería una emulación más cercana al uso real de la aplicación.

Agradecimientos

Agradezco al departamento de ingeniería electrónica de la Universidad Tecnológica Nacional, Facultad Regional Buenos Aires, por haberme introducido en la programación, y ahora darme la oportunidad de participar de este proyecto.

Especialmente al director del proyecto Ing. Nahuel González por el acompañamiento a lo largo del desarrollo.

Referencias

1) **Tkinter**
   https://docs.python.org/es/3/library/tkinter.html
2) **Threading**
   https://docs.python.org/3/library/threading.html
3) **Gtts**
   https://gtts.readthedocs.io/en/latest/
4) **Playsound**
   https://pypi.org/project/playsound/
5) **Socket**
   https://docs.python.org/3/library/socket.html
6) **Pyautogui**
   https://pyautogui.readthedocs.io/en/latest/mouse.html
7) **Repositorio de GitHub**
   https://github.com/mgoyret/rampa_digital
8) **Entornos virtuales**
   https://docs.python.org/3/library/venv.html
9) **Video demostrativo**
   https://1drv.ms/v/s!AkAzRL87Lf9GjeoTuv8b_0xzN8_X4A?e=tZYpkY
10) **Keyboard**
    https://github.com/boppreh/keyboard
11) **Pyinstaller**
    https://pyinstaller.org/en/stable/

Datos de Contacto

Marcos Goyret
Universidad Tecnológica Nacional, Facultad Regional Buenos Aires
Av. Medrano 951, Almagro, C.A.B.A
mgoyret@frba.utn.edu.ar