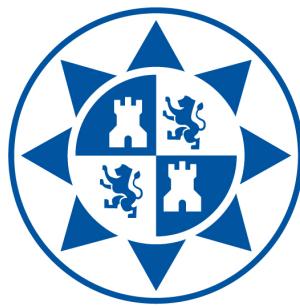




Escuela Técnica
Superior
de Ingeniería de
Telecomunicación



Universidad
Politécnica
de Cartagena

Desarrollo de un teclado para el S.XXI,
accesible y con MicroPython

Autor: Pablo Martínez Bernal

Director: José Alfonso Vera Repullo

Máster Universitario en Ingeniería de Telecomunicación

Curso 2022-23

RESUMEN

En la actualidad es cada vez más la gente que pasa varias horas al día delante de un ordenador, nuestra principal herramienta para controlarlos es el teclado y, sin embargo, su diseño apenas ha avanzado desde su aparición. Este diseño arcaico supone problemas de salud, falta de accesibilidad y reduce la productividad.

Por esto, en el presente documento se estudian los avances y variaciones que ha realizado la comunidad de aficionados en los últimos años y se detalla el diseño y construcción de un teclado más acorde al mundo actual que solventa los problemas recién citados, así como permitiendo una gran capacidad de personalización gracias a su diseño modular y a su diseño(hardware y software) *open source*

ÍNDICE

1. Introducción	3
1.1. Motivación	3
1.2. Objetivo	3
2. Estado del arte	4
2.1. Diseño anticuado	4
2.1.1. Forma del teclado	4
2.1.2. Ubicación de las letras	5
2.2. Algunos avances	6
2.2.1. <i>Split</i>	6
2.2.2. Pulgares	7
2.2.3. Reposamuñecas	7
2.2.4. <i>Tenting</i>	7
2.2.5. Ergonomía	8
3. Implementación del firmware	9
3.1. Instalar MicroPython	9
3.1.1. Preparar el compilador	9
3.1.2. Compilar para Linux (Opcional)	9
3.1.3. Compilar para RP2040	9
4. Referencias	11

1.1. Motivación

Hoy en día, es mucha la gente que se pasa buena parte del día frente a un ordenador, tanto por trabajo como en su tiempo libre. Esto, por supuesto, puede suponer problemas para la salud si no se toman las precauciones necesarias. Por ejemplo, problemas de vista por pasar excesivas horas mirando un monitor, aunque en este frente ya hay una buena cantidad de divulgación e investigación.

Sin embargo, el periférico que más usamos es el teclado y, sin embargo, su *problemático* diseño apenas ha cambiado desde que existen los ordenadores y puede resultar en diversas lesiones y enfermedades en las muñecas.

1.2. Objetivo

El principal fin de este trabajo va a ser el diseño de un teclado que se adapte mejor a la anatomía humana y que, a su vez, incorpore mejoras que lo hagan accesible a gente con diversas discapacidades:

- Teclas con letras grandes y alto contraste
- Un joystick integrado que permita controlar el teclado
- Solenoide para mayor feedback táctil y sonoro

SECCIÓN 2

ESTADO DEL ARTE

2.1. Diseño antiguado



(a) Teclado IBM Model M (1984)



(b) Teclado Logitech MX Mechanical (2022)

Figura 1: Comparativa teclado antiguo y actual



Figura 2: Teclado en Android

Como podemos ver, los teclados no han variado en los últimos 40 años, ni siquiera para adaptarse a las pequeñas pantallas táctiles de los móviles.

2.1.1. Forma del teclado

Quizás nunca nos lo hayamos preguntado puesto que tenemos muy interiorizada la forma de los teclados, pero si lo pensamos un poco, es peculiar la posición relativa de las teclas. Cada fila tiene un pequeño desfase con las demás en vez de estar alineadas. Esto es un legado de sus antecesoras, las máquinas de escribir, donde por limitaciones mecánicas esto tenía que ser así y evitar choques entre las piezas móviles de cada tecla.



Figura 3: Detalle de las letras en una máquina de escribir

Este posicionamiento relativo de las teclas supone un problema a la hora de escribir, ya que la forma óptima de hacerlo sería la siguiente:



Figura 4: “Mapa” de mecanografía

Sin embargo, para escribir así, las muñecas terminan en posiciones un poco forzadas, y los dedos hacen movimientos incómodos. Para arreglar esto surgieron los teclados ortolineales, donde todas las filas están alineadas y los dedos se mueven en una línea recta. Estos teclados suelen tener todas las teclas del mismo tamaño, optimizando así la cantidad de teclas que podemos tener ocupando el mismo espacio (donde antes había una barra espaciadora pueden entrar varias teclas). Como se puede ver en la siguiente imagen, normalmente también prescinden del teclado numérico para reducir el tamaño, este modelo se conoce como “75 %” ya que tiene 75 teclas mientras que los teclados comunes (“100 %”) tienen 104/105 teclas. Otras variantes comunes son “40 %”, “60 %”, “65 %”



Figura 5: Teclado ortolineal *RGB75*

2.1.2. Ubicación de las letras

Otro legado que nos dejaron las máquinas de escribir es la distribución QWERTY, que probablemente sea la única distribución que hemos visto a lo largo de nuestra vida. El problema con esta disposición es que, si bien distribuye las letras de forma que se usan las dos manos por igual, se diseñó en la década de 1860, por lo que uno de sus objetivos era el de reducir los atascos en las máquinas de escribir separando las teclas más usadas de la parte central.

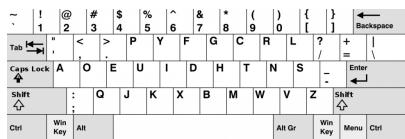
En contra, ahora que gracias a la electrónica no tenemos estas limitaciones, se han diseñado distribuciones que minimizan la distancia media que se debe recorrer al escribir, por lo que una vez acostumbrados a ellas



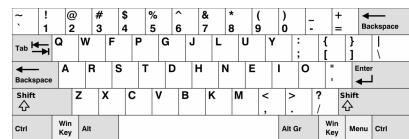
se puede escribir más rápido y reduciendo la fatiga en los dedos. Las dos más extendidas son DVORAK y COLEMAK.



Figura 6: Distribución QWERTY



(a) Dvorak



(b) Colemak

Figura 7: Distribuciones alternativas

2.2. Algunos avances

No todo iba a ser quedarse en el pasado, gracias a la apasionada y extensa comunidad de aficionados a los teclados mecánicos, en los últimos años han aparecido multitud de diseños que incorporan diferentes cambios respecto al paradigma actual, algunos de estos conceptos son:

2.2.1. Split

Como hemos comentado ya, uno de los problemas más comunes en gente que usa mucho los teclados es la aparición de dolencias en las muñecas, a fin de que estas se posicen de una forma más natural y cómoda, se opta por partir el teclado en dos mitades.



Figura 8: Teclado *Quefrency*



2.2.2. Pulgares

Muchos teclados dotan de una mayor utilidad a los pulgares, que normalmente solo utilizamos para la barra espaciadora, añadiendo unas cuantas teclas en lo que comúnmente se conoce como *thumb cluster*.



Figura 9: Teclado *Ergodox*

2.2.3. Reposamuñecas

Otros diseñadores aprovechan la oportunidad para integrar este accesorio de una forma más eficaz que la típica “rampa” de plástico que estamos acostumbrados a ver.



Figura 10: Teclado *Moonlander*

2.2.4. *Tenting*

Esta conocida técnica solo se puede aplicar en teclados *split* y consiste en levantar la parte central del teclado, de forma que la muñeca está en una posición más natural en vez de estar paralela al plano que genera la mesa. Lo mejor de esta mejora es que se puede añadir a cualquier teclado que no la incorpore en su diseño añadiendo algún objeto para levantarla.

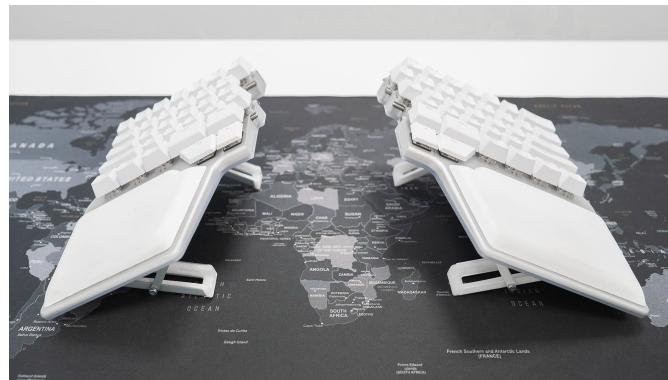


Figura 11: Teclado *Dymga Raise*

2.2.5. Ergonomía

El mayor ejemplo de esta filosofía de diseño es el *Dactyl Manuform*, un teclado que debido a su particular forma ni siquiera puede funcionar con una PCB (placa de circuito impreso) y tiene que soldarse a mano la unión entre todos sus componentes. El beneficio de su diseño es que tiene en cuenta la forma de las manos, por lo que las teclas se encuentran posicionadas acorde al movimiento de los dedos. Además, hay usuarios que optan por modificar el diseño e integrarle una *trackball* para poder controlar el cursor sin tener que mover la mano entre el teclado y el ratón.



Figura 12: Teclado *Dactyl Manuform* con trackball

SECCIÓN 3

IMPLEMENTACIÓN DEL FIRMWARE

3.1. Instalar MicroPython

Python es un lenguaje interpretado, por lo que existen distintas implementaciones del mismo, por nombrar algunas:

- Cython (C)
 - IronPython (.NET)
 - Jython (Java)
 - PyPy (Python)

Para este proyecto voy a utilizar MicroPython, que también está escrito en C pero su objetivo es el de minimizar el uso de recursos (mayormente RAM) para que sea viable emplearlo en microcontroladores.

3.1.1. Preparar el compilador

Tras clonar el repositorio de MicroPython (`git clone https://github.com/miropython/micropython`), hacemos `make -C mpy-cross` para compilar el compilador cruzado de MicroPython que nos permitirá compilar nuestro código para ser ejecutado en diferentes arquitecturas

3.1.2. Compilar para Linux (Opcional)

Si queremos usar MicroPython en nuestro ordenador para hacer pruebas, en vez de CPython(que es la versión más común), usaremos el compilador que acabamos de construir para compilar el código fuente del intérprete y usarlo en nuestra máquina `cd ports/unix && make submodules && make`
Y ahora podemos ejecutarlo con `cd build standard`

```
$ ./micropython
MicroPython 13dceaa4e on 2022-08-24; linux [GCC 12.2.0] version
Use Ctrl-D to exit. Ctrl-F for paste mode.
```

3.1.3. Compilar para RP2040

Primero instalamos un compilador necesario para la arquitectura del procesador, en mi caso (Arch Linux), el comando es `sudo pacman -S arm-none-eabi-gcc`.

Ahora, añadimos TinyUSB a la configuración del compilador, para que añada dicha librería siguiendo la información de noobee [1].

```
</> ports/rp2/CMakeLists.txt
    | set(MICROPY_SOURCE_PORT
(+)|     modusb_hid.c
    |     fatfs_port.c
```



```
        .  
        | set(MICROPY_SOURCE_QSTR  
(+) | ${PROJECT_SOURCE_DIR}/modusb_hid.c  
        | ${MICROPY_SOURCE_PY}
```

Y añadimos/editamos los archivos necesarios (ver commits en su repositorio de GitHub). Por último, de forma análoga al paso anterior, hacemos `cd ../../rp2 && make submodules && make`

SECCIÓN 4

REFERENCIAS

1. noobee, *Add TinyUSB to RP2040*, <https://github.com/noobee/micropython/tree/usb-hid>.