

# CarMap - Documentación

Repositorio: [https://github.com/aigora/twIA\\_2021-carmap\\_chikito](https://github.com/aigora/twIA_2021-carmap_chikito)

## Miembros del grupo

Tag de GitHub en paréntesis

- Echedey Luis Álvarez (@echedey-luis-alvarez)
- Fernando Moreno Santa Cruz (@Fer014)
- Amelie Nader Prieto (@amelie-nader-prieto)
- Luis Fernando Rubio Regojo (@luisrubio27)
- Celia Torrecillas Barba (@Celia-t22)
- Lucas Sánchez Solera (@luc-39)

*Todos (somos) estudiantes de primer año de Electrónica (grupo A109)*

## Índice

- El proyecto
- Descripción del sistema
- Algoritmos
- Funciones y librerías
- Modificaciones del documento
- Enlaces de interés

# El proyecto CarMap

## Propósito del proyecto

Aunque a priori parece un proyecto cutre de estudiantes de primero de carrera (lo es) pretendíamos crear una aproximación bastante aproximada de lo que podría ser un robot que pueda ayudar en eventos catastróficos, en lugares que suponen un riesgo alto para los humanos. Sobra decir que el proyecto se queda lejos de la realidad, porque ni el Arduino UNO, ni nuestro tiempo ni nuestro nivel dan para tanto (por ahora).

Esencialmente no tiene mucha más utilidad que la de moverlo por un sitio e ir registrando su camino. Con este algoritmo, se podría estimar su posición, o moverse por sí mismo siguiendo una línea, tal vez llegar a volver por sí solo.

## Problema a resolver

Concretamos al principio del proyecto que este se trataría de un robot similar a un coche tele-dirigido y cuyo algoritmo le permite que su trayectoria se guarde en un mapa (fichero) y que el robot pueda “recordar” su trayectoria para poder regresar por el mismo camino que hizo. Además, queremos que sea fácil de manejar, para lo que intentaremos hacer algo similar a un mando radio-control.

Posteriormente consideramos que debido a la limitación del hardware y los requerimientos de la tarea, principalmente porque hay que manejar ficheros, lo que mejor se adapta es gestionar comandos y todo lo relativo a propiedades del robot en el PC, enviarlos y que el robot los ejecute.

## Descripción del sistema

Nuestro programa cuenta con tres subsistemas: por un lado haremos un teclado de macros (o HID, *Human Interface Device*) que permita manejar el robot. Por otra parte, un programa en el ordenador que tendrá el algoritmo más profundo en cuanto a manejar datos: entrada por teclado de los movimientos que hacer, exponer en la pantalla un mapa con los movimientos que va realizando el robot y algún otro dato, enviar datos al robot por Bluetooth y gestionar un fichero con la trayectoria. Finalmente, el robot, que deberá moverse en función de los comandos recibidos por Bluetooth.

En nuestro repositorio de GitHub se pueden ver varias carpetas. Estas son las que contienen a los códigos de los distintos subsistemas que hemos definido:

- Para el teclado de macros (HID): Teclado\_Macros/
- Para el programa del PC: Mapeador/
- Para el robot: Robot\_CMC/

Tanto el teclado de macros como el robot funcionan con Arduino. Asimismo, se programará el Mapeador en C con Visual Studio, aunque también tiene código en C++ para usar clases; en el mismo ejemplo de conexión por Bluetooth se utiliza la clase `String`, y en el *Mapeador* porque en la biblioteca `SerialClass.h`; también para la propia conexión se utiliza una clase.

Planteamos los problemas a resolver por distintas partes del proyecto antes mencionadas: el teclado de macros o HID, el programa en el PC (*Mapeador*), y el robot, que hemos querido bautizar como *Chikito* por cuestiones de marketing.

## Descripción detallada e identificación de sensores y actuadores

Este apartado fue escrito inicialmente en la Wiki del proyecto en GitHub después de discutir el tema entre todos en varias reuniones.

### Robot

- El Arduino se conecta al ordenador por Bluetooth
  - Lucas Sánchez (@luc-39) se encarga del código en el Arduino.
    - Para poder conectar el software de Arduino con la placa que se encontrará en el robot es necesario usar un módulo bluetooth HC-05 que se encontrará en la protoboard. Además será necesario hacer uso de `SoftwareSerial.h`.
    - @Celia-t22 se encarga del movimiento de los motores mediante funciones.
  - Se pueden incluir sensores como opto acopladores para monitorizar el giro de las ruedas. Al final esta parte no procede.

Para completar la información relacionada con el robot se utilizará el pdf<sup>[3]</sup> de moodle sobre la comunicación con Arduino y PC mediante bluetooth y la página<sup>[4]</sup> sobre cómo usar Arduino y C++.

## Mapeador

Tras una profunda lluvia de ideas se asignaron las siguientes tareas:

- Añadir función que calcule ángulo entre 2 vectores en vector.h para @Fer014.
- Maquetar una librería con propiedades del robot, de forma genérica, para alterarlas de forma más cómoda. Para @Fer014.
- Empezar con la entrada de comandos desde el teclado, que realizará @luisrubio27 con ayuda de la librería time.h.
  - Se crearán vectores que se añaden a un array.
  - Estos vectores dependen de la tecla; la dimensión del array, del tiempo presionado.
    - Sólo las teclas asociadas al movimiento (posición) del robot responden continuamente al mantenerse presionadas; en este proceso entran los vectores de posición y el tiempo total en el que la tecla se mantiene pulsada. Por tanto, los sensores serán el teclado del PC y el teclado de macros. No habrán actuadores *per se*, pero sí operaciones con los vectores y salida por la consola.

## Teclado de macros

Esta parte tiene que ser sencilla, implementando solo entradas digitales (pulsadores) y cuya actuación simula la pulsación de botones del teclado (conceptualmente, la actuación es una función programada, la cual puede ser pulsar una tecla o varias o hacer cualquier cosa más compleja). Desarrollándose por Amélie (@amelie-nader-prieto).

En resumen los sensores son los pulsadores del teclado y el actuador es la comunicación con el ordenador.

## Diseño del sistema por subsistemas

El contenido de este apartado fue también escrito inicialmente en GitHub tras una reunión del grupo, pero se ha actualizado ligeramente al ponerlo aquí.

Trabajamos en este paso a partir de las especificaciones hechas en la [tarea anterior](#), de nuevo teniendo en cuenta los requisitos funcionales y no funcionales de hardware y código de las distintas partes del proyecto. Previamente a esta tarea definimos los datos que se manejan en distintas funciones del programa y se profundizó más en asuntos del hardware; en esta tarea nos interesa más trabajar en el software de nuestro proyecto; tanto en la estructura del programa principal ejecutado en el ordenador como en las aplicaciones de las plataformas auxiliares (robot y teclado).

En esta segunda reunión del equipo de desarrollo nos interesamos en aclarar, organizar y establecer los límites en el proceso de desarrollo que llevaremos individualmente sin perder de vista el trabajo en equipo. Necesitamos mantener cohesionado el proyecto y asegurar que las conexiones entre cada parte sean sólidas. A continuación exponemos en qué parte de la línea de la información se centra cada miembro, en función de cada apartado del proyecto:

## HID / Teclado de Macros

En la aplicación de PC para el teclado de macros se realizan las tareas de monitorización de las señales mediante funciones de SerialClass.h.

- Especificaciones y desarrollo por Amélie (@amelie-nader-prieto). Esta parte es la más simple en el código, pero se comunica directamente con el programa principal, y finalmente sirve como un dispositivo para controlar manualmente el robot.
  - Las funciones (macros) aparecen programadas en el código de Arduino, y en la primera versión del proyecto se limitan a pulsar teclas.
  - Para este propósito se puede considerar prescindible un teclado de macros, pero siempre se puede mejorar el código para que se puedan hacer funciones más complicadas del programa.
  - Se utiliza la librería de Arduino Keyboard.h para utilizar el teclado, mediante funciones en las que no se pasan valores.[\[9\]](#)

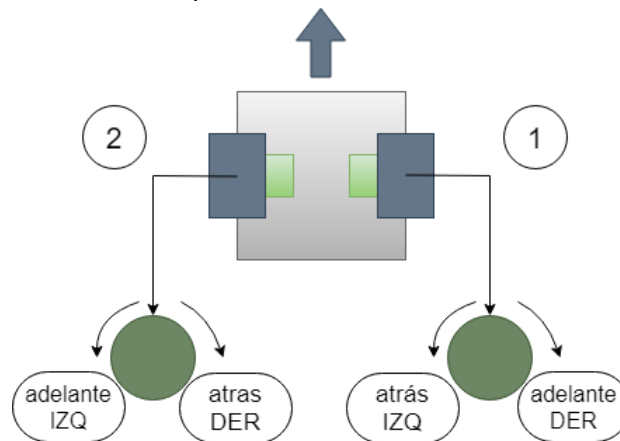
## Mapeador

En esta parte del proyecto establecemos aquellas funciones principales que queremos que se gestionen en el PC. Las partes más importantes son la entrada de datos mediante el teclado de macros, los cálculos relacionados con el Robot (el algoritmo), la escritura de ficheros, y la comunicación (envío de datos) por el puerto serie Bluetooth con el Robot.

- Especificaciones y desarrollo por @luisrubio27
- i. Diseñar el mapeador a través de la función gotoxy para imprimir un mapa que de a entender el recorrido del robot. Que se sitúe centralizado en la pantalla.
- ii. Diseñar el sistema brújula para que introduciendo las letras n, o, s, e se mueve en esa dirección tanto el mapa como el robot.
- iii. Introducir los datos de entrada para conectar el mapa con el algoritmo. Para posterior desarrollo de @Fer014.
- Especificaciones y desarrollo por @Fer014
  - i. Calcular el movimiento de las ruedas del robot, su avance y rotación, a partir de los datos recibidos y de la biblioteca "robot.h".
  - ii. Calcular las coordenadas de la posición del robot, para su posterior uso por @echedey-luis-alvarez.
- Especificaciones y desarrollo por @echedey-luis-alvarez
  - i. Escritura en fichero de los puntos por los que pasa el robot. Los puntos se toman del código a cargo de @Fer014 .
  - ii. Envío de datos por Bluetooth. Se acuerda entre todo el equipo que al robot se le pasará la cantidad de tiempo y el sentido en que debe girar cada rueda. Más concretamente, el formato de la cadena con el que se envía la información se aclarará con @luc-39 .

## Robot

- Especificaciones y desarrollo por @luc-39
  - i. Recibir los datos por Bluetooth desde el ordenador con ayuda de la librería SoftwareSerial.h. Para poder conectar el programa del ordenador con el del Arduino se verá con @echedey-luis-alvarez.
  - ii. Enviar la información recibida a @Celia-t22 y que así pueda operarse la última parte del código.
- Especificaciones y desarrollo por @Celia-t22
  - i. Recibo la información que manda @luc-39.
  - ii. Con esta información, los motores se moverán de forma que permita que *Chikito* se mueva en la dirección y el tiempo deseado.
  - iii. A continuación un esquema del movimiento de los motores:



Este esquema representa los nombre que se le ha dado a cada movimiento en el código. Primero se ha puesto un nombre para cada distinguir cada motor, es decir “1” al motor que va a la derecha de *chikito* y “2” al que va la izquierda. Después, para representar si deseamos que la rueda gire hacia delante o hacia atrás, se ha representado como “IZQ” o como “DER”; estos nombre se han puesto así porque representan el lado que girará la rueda si se mira de frente.

## Entrada y salida de datos

Las entradas y salidas de datos se describen como una sucesión lineal entre las distintas partes del proyecto.

### Teclado de Macros

- Entrada: En el programa se realiza la lectura sucesiva de los pins digitales del controlador (los cuales son INPUT), y se guarda el valor (HIGH o LOW, donde LOW ocurre cuando el botón es pulsado) de todos ellos en un vector.

- Salida: El programa de Arduino no tiene datos de salida pues sus funciones no devuelven ningún valor, sino que pulsan las teclas N,S,E,O, que son las “flechas direccionales” con las que se dirigirá el robot en su movimiento. La salida no es un dato realmente, sino una *acción* que se realiza según los datos de entrada.

Dentro del mapeador están preparadas las bibliotecas y funciones más complejas que se ocupan de relacionar las señales recibidas del teclado con los comandos del robot (biblioteca de operaciones con vectores, biblioteca de propiedades del robot).

## Mapeador

- El programa recibe los datos digitales del teclado y realiza con ellos las funciones correspondientes.
- Se realizará un bucle switch/case con los comandos correspondientes a las teclas que realizan las funciones más básicas del robot (la tecla de encendido y de cambio de perfiles)
- A partir de las coordenadas se genera un array de los vectores de posición del robot.

## Robot

- Las variables y/o constantes necesarias serán manejadas en la biblioteca de propiedades del robot localizada en el mapeador.
  - En los inicios del proyecto se planteó crear una estructura asociada al robot (template del robot) con únicamente tres variables distintas, pero decidimos posponer la definición de una estructura para otro momento.
- La conexión entre el robot y el ordenador será inalámbrica (Bluetooth).
- El robot funciona con una Arduino UNO Rev3 y la conexión Bluetooth con el módulo Bluetooth HC-05;

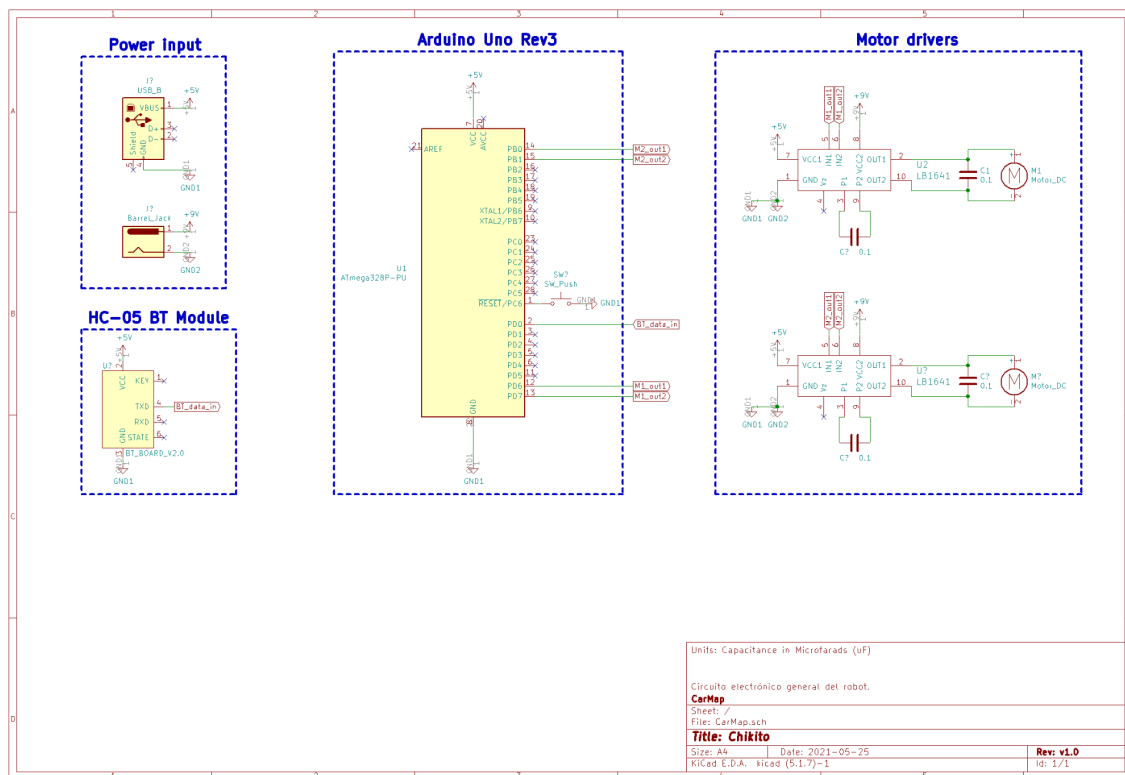
Coincidimos en que las especificaciones en el modo en que los valores obtenidos son almacenados y transmitidos entre las partes del proyecto, empleando ficheros y definiendo arrays numéricos (matrices) y vectores de estructuras se deja para fases más avanzadas; sin embargo tenemos una idea de los datos que es crucial manejar en el programa para que este pueda realizar su objetivo.

- Posición en el espacio expresada en coordenadas cartesianas
  - Posición
  - Rotación: ángulo en radianes o vector dirección
- Velocidad de movimiento
  - Falta concretar las componentes y si habrá aceleración

Ahora bien, la salida se hace por un par de motores. Estos se controlan mediante dos circuitos integrados LB1641<sup>[5]</sup>, unos drivers bidireccionales que se controlan mediante dos pines digitales cada uno.

## Esquema electrónico del robot

A continuación se explica el funcionamiento del sistema electrónico que le da vida al robot. Los recursos diseñados en KiCad se pueden consultar en el repositorio.





2. Los 5V alimentan la parte lógica: Arduino, módulo BT y el sistema lógico de los drivers.
3. La otra fuente alimenta la parte de potencia, es decir, los motores con los drivers de intermediarios.
4. Las señales lógicas que determinan en qué sentido se mueve cada motor son las que salen de los pines 6 a 9, dos por motor: como se puede consultar en el datasheet del LB1641 [\[5\]](#), cuando un pin está encendido y el otro apagado se mueve en un sentido. Si se intercambian estos pines, gira en el contrario. Si ambos están a nivel lógico HIGH o LOW no se mueven. Aunque no se especifica expresamente, se indica que en este último caso estarían en modo "*Braking*", lo que implica que las salidas al motor estarían en corto y por autoinducción debería frenar antes.
5. Finalmente, el módulo Bluetooth se conecta desde su pin TXD (transmisión) al RX del Arduino (recepción), o sea, que el ATmega recibe la información enviada por el HC-05 que recibe desde el PC.

## Algoritmos y código como conjunto

A continuación se exponen cada una de las funciones que tienen lugar código. Para comentar cada librería que hemos creado o partes en específico que requieren más atención desde un punto de vista especialmente técnico vaya al apartado [Funcionamiento de las distintas partes del código](#).

### Teclado de macros - HID

El HID funciona con un código escrito con el Arduino IDE y con un algoritmo no muy complejo.

1. El controlador se conecta con un cable al puerto COM3 del ordenador y es necesario conectarlo para que funcione el código.
2. En la función `void loop()` se realiza la lectura sucesiva de los pins digitales en los que están conectados los pulsadores, y guarda sus valores (HIGH o LOW) en un vector.
3. Si el programa detecta un LOW en la posición `n` del vector, llama a una función de parámetro `n` que se encarga de pulsar la tecla que corresponda mediante un switch. Esta función sirve para emular pulsaciones en el teclado del ordenador.

Y sobra ya decir que está incluida `Keyboard.h`.

### Definiciones

```
#define N 6  
#define S 7  
#define E 8  
#define O 9
```

Se definen los pins digitales que están conectados a cada pulsador, y que en la ejecución del programa son leídos sucesivamente al estar en un vector.

Además de estos, se podrían añadir otros botones asignados a cualquier otro pin del controlador (del 2 al 13), y programar funciones para ellos.

```
const int arrowpad[] = {N, S, E, O};
```

Aunque es técnicamente una variable global, en Arduino no suele ser tan problemático porque se suelen manejar funciones que no modifican o devuelven valores numéricos. Además en este caso, tener definidas las constantes facilita la adición de nuevas teclas y funciones ya que la dimensión del vector `arrowpad[]` está declarado como `const int`, por lo cual se podría cambiar el vector sin tener que actualizar otras funciones o bucles.

### Funciones

Son muy simples. Intenté hacer este programa corto, especialmente hacer la función principal (`void loop()`) lo más corta posible.

En `setup()` se inicializa el Serial, los pines del vector, y el teclado.

En `loop()` se realiza esta lectura sucesiva del vector con los botones, leyendo uno por uno los índices del vector, y si uno de ellos está pulsado, lo vuelve a leer y llama a, digamos la función del teclado;

key(i+1); donde el parámetro es una variable utilizada en el bucle donde se hacían estas lecturas; for(i=0;i<dim;i++){...} por lo cual i es el índice del vector en cada momento.

Esta es la función:

```
void key(int i){  
    switch(i){  
        case 1:north();break;  
        case 2:south();break;  
        case 3:east();break;  
        case 4:west();break;  
        default:break;  
    }  
}
```

Y esas funciones llamadas en cada case son los comandos de Keyboard.h que pulsar las teclas en el ordenador; Keyboard.press(tecla); Keyboard.release(tecla);

Los nombres de las teclas son valores definidos en Keyboard.h.

## Mapeador

Este programa utiliza una gran cantidad de librerías, ya que implementa muchas funcionalidades distintas entre sí; estas son:

Bibliotecas genéricas

1. **stdio.h**
  - a. Entrada básica de datos por consola (printf y scanf)
2. **stdlib.h**
  - a. Asignar memoria de forma dinámica
3. **string.h**
  - a. Permite realizar algunas operaciones con cadenas, se usa para crear los comandos del Robot.
4. **wchar.h**
  - a. Para utilizar caracteres “amplios”, i.e., que ocupan más de un byte. Esto se usa porque la biblioteca **SerialClass.\*** nos daba un problema en el tipo de puntero al puerto utilizado que se pasaba al SO.
5. **iso646.h**
  - a. Posiblemente la librería que más aporte al programa: resulta más legible and, or y not que “&&”, “| |” y “!” respectivamente. Sobre todo si vienes de Python.
6. **math.h**
  - a. Introduce las funciones trigonométricas y de redondeo que se usan para el mapa y el cómputo de coordenadas.
7. **stdbool.h**
  - a. Define los tipos bool si estás en C. En C++ ya vienen definidos. Esto es un poco por tontuna; es para la parte de crear los comandos, que tenga un poco el sentido binario que tiene, ya que se desplaza o rota, hacia delante o hacia atrás, hacia la izquierda o hacia la derecha.

8. **conio.h**

- a. Introduce la función `_getch()` que permite tomar un carácter del búfer de entrada nada más se pulsa (como `getchar()`), pero esta última daba problemas y no funcionaba ubicar el búfer al final con `fseek(stdin, 0, SEEK_END)`).

9. **windows.h**

- a. Para ubicar la posición del cursor en la consola (`gotoxy()`).

Bibliotecas locales:

10. **vector.h**

- a. Define la estructura `vector2D` y múltiples operaciones. Al final no se usa ni la mitad, pero aporta flexibilidad por si se quiere añadir alguna funcionalidad o refactorizar la parte del mapeador.

11. **SerialClass/SerialClass.h**

- a. Añade la clase `Serial` (un tipo de dato de C++) para conectar por Bluetooth más fácilmente. Recuperada de <https://github.com/Gmatarrubia/LibreriasTutoriales>, modificada localmente.

12. **waypoints\_filehandler.h**

- a. Funciones que permiten inicializar, escribir y leer vectores del archivo de la trayectoria con el formato explicado en el apartado [2.4.1. Gestión del fichero en memoria](#).

En primer lugar se inicializa una ristra de variables que se utilizan durante la ejecución.

En segundo lugar, se le pide al usuario que indique por qué puerto COM se debe conectar al Arduino. Se intenta conectar, y si falla imprime el mensaje de error y se detiene la ejecución. Si es un éxito, se crea el archivo que va a guardar los vectores en binario y se inicializa. Si ocurriese algún error en esta parte, se detendría.

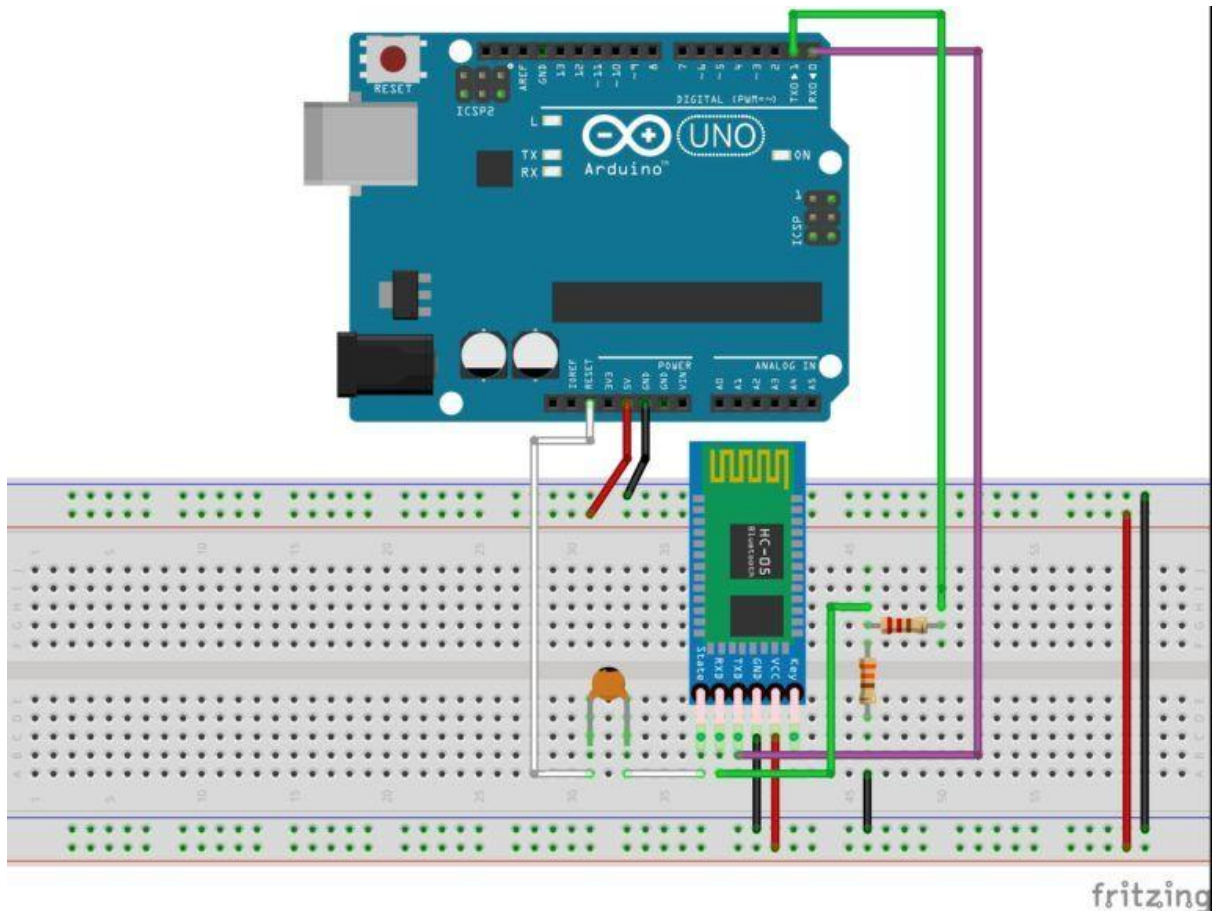
Si todo ha funcionado correctamente, imprime el estado del programa y los controles, para que el usuario sepa cómo proceder. Tras una pulsación que se haga en el teclado, se limpia lo escrito por pantalla, se inicializa el mapa y la posición del robot.

Una vez en el bucle, se espera a una pulsación del usuario, y si está definida (n, s, e, o) se crea un comando que se envía por Bluetooth. En caso contrario no se hace nada. Si se envía el comando, se cambia la posición en el mapa del robot y se vuelven a mostrar la posición, la distancia recorrida y la distancia desde el origen.

Por último, cada vez que se envía un comando se añade el vector al fichero y se espera el tiempo necesario a que espere de

## Robot

El esquema de la protoboard con el módulo bluetooth HC-05 sería el siguiente:



Anotación: Es necesario que la resistencia esté ya que del pin TX salen 5V y el módulo bluetooth HC-05 admite un voltaje más bajo, unos 3,3V.

Para poder configurar el módulo bluetooth se utilizan los comandos 'AT':

- 'AT+NAME'. Se define el nombre del dispositivo.
- 'AT+PSWD'. Se define la contraseña (PIN) del módulo bluetooth para cuando nos queramos conectar con el bluetooth.
- 'AT+UART'. Sirve para los parámetros de la comunicación, como la velocidad.
- 'AT+ROLE'. Se utiliza para indicar el rol del módulo: si es un 0 su rol es el de esclavo (recibe los datos del maestro) y si es un 1 su rol es el de maestro (envía los datos al esclavo).
- 'AT+ORGL'. Restaura los valores de fábrica.
- 'AT+RESET'. Vuelve al modo usuario. Sirve para salir del modo de configuración.


Estos comandos se pueden usar de dos formas:

- 'COMANDO?'. Lectura del valor, no lo modifica.
- 'COMANDO='. Asigna un valor al comando.

Para poder usar estos comandos es necesario primero que el módulo esté en modo configuración. Para ello hay que presionar durante los cinco primeros segundos de encendido el botón que tiene el módulo bluetooth hasta que los leds estén parpadeando de forma lenta, si parpadean rápido quiere decir que se encuentra en modo usuario.

Una vez subido el programa del bluetooth a la placa Arduino es necesario abrir el monitor serial y verificar la configuración de este: habilitar autoscroll, habilitar Ambos NL & CR en los ajustes de línea y ajustar la velocidad a 9600 baudios. Una vez hecho esto se puede empezar a configurar el módulo bluetooth con los comando 'AT' escribiéndolos en el monitor serial. Una vez finalizado el proceso de configuración se introduce el comando 'AT+RESET' para salir al modo usuario y poder emparejarlo mediante bluetooth.

Programa básico del módulo bluetooth:



```
Programa_bluetooth_base Arduino 1.8.13
Archivo Editar Programa Herramientas Ayuda

Programa_bluetooth_base $
#include <SoftwareSerial.h>

SoftwareSerial miBT(10,11);

void setup() {    // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("Listo");
  miBT.begin(38400);
}

void loop() {    // put your main code here, to run repeatedly:
  if (miBT.available())    // lee BT y envía a Arduino
    Serial.write(miBT.read());

  if (Serial.available())
    miBT.write(Serial.read());    // lee Arduino y envía a BT
}
```

# Funcionamiento de las distintas partes del código

Mapeador: procesamiento pulsaciones, cálculo de coordenadas y mapa

## Objeto

En este apartado se detalla el funcionamiento del Mapeador. Este término engloba al conjunto de funciones encargadas de recibir las pulsaciones del teclado de macros, calcular las coordenadas del robot y el tiempo que se debe mover, y dibujar en la consola un mapa que muestre la trayectoria del robot.

## Definiciones

El programa utiliza una serie de constantes definidas por el usuario:

`#define WHEEL_RADIUS`: Radio de las ruedas del robot (m).

`#define ROBOT_WIDTH`: Distancia existente entre las dos ruedas del robot (m).

`#define WHEEL_TURN`: Número de giros que las ruedas del robot realizan hacia delante o hacia atrás, tras cada pulsación.

`#define ROT_ANGLE`: ángulo que el robot rota a derecha o izquierda (rad), tras cada pulsación.

`#define ANGULAR_VELOCITY`: Velocidad angular a la que rotan las ruedas del robot (rad/s).

## Funciones

1. `int ProcessPulsation(pulsation* p_pulsation, int inputChar)`
  - a. Recibe la información transmitida por el teclado de macros (HID), detecta el tipo de pulsación que se ha producido (n, s, e, o), y contabiliza el número total de pulsaciones de cada tipo.

Argumentos:

- i. `p_pulsation`: puntero a estructura que almacena el tipo de pulsación y el número total de estas.
- ii. `inputChar`: recibe un carácter del teclado a través de la función `_getch`.

2. `void Calc_Coordinate_Time(pulsation* p_pulsation, coordinate* p_coord, tiempos* p_time)`
- Calcula las coordenadas (en metros) a las que se desplaza el robot tras cada pulsación.
  - Calcula el ángulo de rotación (en radianes) del robot respecto de la posición inicial (mirando en la dirección positiva del eje OY) tras cada pulsación. El conocimiento de este ángulo es necesario para poder calcular las coordenadas de manera efectiva, puesto que el robot debe ser capaz de desplazarse en todas las direcciones. La rotación hacia la derecha se expresa con un ángulo positivo, mientras que la rotación hacia la izquierda se expresa con un ángulo negativo.
  - Calcula el tiempo (en segundos) que el robot debe realizar cada uno de sus movimientos básicos (avanzar, retroceder, rotar hacia la derecha y rotar hacia la izquierda) tras cada pulsación.
  - Calcula la distancia recorrida por el robot, la distancia del robot respecto al origen de coordenadas, e imprime en pantalla las coordenadas del robot y las dos distancias anteriores.

Argumentos:

- `p_pulsation`: puntero a estructura que almacena el tipo de pulsación y el número total de estas.
- `p_coord`: puntero a estructura que almacena las coordenadas del robot, el ángulo de rotación del robot, y la distancia recorrida por este, tras cada pulsación.
- `p_time`: puntero a estructura que almacena el tiempo que el robot debe avanzar, retroceder, rotar hacia la derecha y rotar hacia la izquierda, tras cada pulsación.

El cálculo de los anteriores parámetros (coordenadas y tiempos) es posible gracias a la utilización de fórmulas matemáticas y físicas (movimiento circular uniforme). En concreto:

I)  $S = 2 \cdot \pi \cdot R_{wheel} \cdot Wheel_{turn}$  (Longitud de la circunferencia de la rueda).

II)  $S = R_{wheel} \cdot \omega \cdot t$  (Para calcular el avance y retroceso del robot).

III)  $S = \alpha \cdot (Robot_{width} / 2)$

Sustituyendo I) en II) y despejando  $t$  tenemos:

IV)  $t = (2 \cdot \pi \cdot Wheel_{turn}) / (\omega)$  (Para calcular el tiempo que el robot avanza y retrocede).

Igualando II) y III) y despejando  $t$  tenemos:



$$V) t_r = (\alpha \cdot Robot_{width}) / (2 \cdot \omega \cdot R_{wheel})$$
 (Para calcular el tiempo que el robot rota a derecha o izquierda, tras cada pulsación).

Donde:

- $S$ : Distancia recorrida por el robot al avanzar o retroceder (m), tras cada pulsación.
- $t$ : Tiempo que el robot avanza o retrocede (s), tras cada pulsación.
- $t_r$ : Tiempo que el robot rota a derecha o izquierda (s), tras cada pulsación.
- $\omega$ : Velocidad angular a la que rotan las ruedas del robot (rad/s). Definida anteriormente como `#define ANGULAR_VELOCITY`.
- $\alpha$ : Ángulo que el robot rota a derecha o izquierda (rad), tras cada pulsación. Definido anteriormente como `#define ROT_ANGLE`.
- $R_{wheel}$ : Radio de las ruedas del robot (m). Definido anteriormente como `#define WHEEL_RADIUS`.
- $Wheel_{turn}$ : Número de giros que las ruedas del robot realizan hacia delante o hacia atrás, tras cada pulsación. Definido anteriormente como `#define WHEEL_TURN`.
- $Robot_{width}$ : Distancia existente entre las dos ruedas del robot (m). Definido anteriormente como `#define ROBOT_WIDTH`.

3. `void DrawMap(coordinate* p_coord)`

- Imprime en pantalla el carácter “#”, que representa la posición del robot, sobre unos ejes cartesianos dibujados previamente. Para mostrar la posición del robot en el mapa, sabiendo sus coordenadas, se hace uso de la función `gotoxy`. Los puntos del mapa por los que se ha desplazado el robot tras cada pulsación quedan reflejados y guardados en el mapa, lo cual nos permite conocer su trayectoria.
- Imprime en pantalla el carácter “+”, siempre junto al carácter “#” del robot, y colocado respecto a este de manera que señale la dirección en la que está orientado el robot.

Argumentos:

- `p_coord`: puntero a estructura que almacena las coordenadas del robot, el ángulo de rotación del robot, y la distancia recorrida por este, tras cada pulsación.

Para que el movimiento del robot en el mapa sea uniforme, y no dependa del radio de las ruedas ni de  $\pi$ , se aplica un factor de corrección en ambos ejes OX y OY.

4. `void gotoxy(short x, short y)`

- a. Permite situar el cursor a un punto determinado de la pantalla de manera que cualquier función de impresión que se ejecute a continuación aparecerá a partir de ese punto. Es necesaria para imprimir en pantalla los ejes y la posición del robot.

Argumentos:

- i. `short x`: Componente X del punto buscado.
- ii. `short y`: Componente Y del punto buscado.

5. `void clearScreen(int spaces)`

- a. Limpia cantidad de caracteres en pantalla, sustituyéndolos por espacios en blanco.
- b. Argumentos
  - i. `spaces`: Cantidad de caracteres que se desea limpiar

## 2.4.1. Gestión del fichero en memoria

### Objeto

En este apartado se detalla el funcionamiento del tratamiento del archivo que almacena la trayectoria en forma de puntos. Más concretamente, se tratará de explicar el funcionamiento del código declarado en `waypoints_filehandler.h` y definido en `waypoints_filehandler.cpp`. A pesar de estar en formato `.cpp`, el código está escrito en C. Es para evitar un incómodo error que ocurre cuando se utilizan archivos `.cpp` y `.c` simultáneamente en VS: LNK2019.

### Dependencias

- Biblioteca locales:
  - `vector.h`: los puntos se declaran como vectores bidimensionales, i.e. estructuras `vector2D`.
- Bibliotecas genéricas:
  - `stdio.h`: manipulación básica de los ficheros.
  - `stdlib.h`: asignación dinámica de memoria.
  - `time.h`: creación de la estampa de tiempo para hacer los archivos únicos
  - `string.h`: copia y obtención de largo de cadenas. Funciones `strcpy_s()` y `strlen()`, para crear el nombre de archivo.

- `inttype.h`: creación de variables de tipos de ancho fijo.

## Archivo

El archivo que se utilizará será en binario, es decir, los datos se guardan tal y como las variables están en la RAM (en nuestro caso dos tipos, un `uint32_t` y una estructura `vector2D`). Esto facilita la búsqueda de un dato en concreto, especialmente porque no me aclaro con los finales de línea y los `scanf_s()`.

En primer lugar se guarda una variable con la cantidad de puntos que se han escrito en él. Esto permite facilitar la lectura (o, incluso, la asignación de memoria dinámica a un array de puntos). Esta es la variable `uint32_t`. Esta primera variable está definida en todo el programa como `uint32_t` (estándar C11), lo que implica que el tamaño que ocupa tanto en memoria como en el fichero es de 32 bits o 4 bytes independientemente de la arquitectura del sistema / OS que se utilice. Aplicando el mismo razonamiento para el tamaño de las estructuras `vector`, 2 tipos `double` deberían ocupar 128 bits (16 bytes): según la IEEE 754 ocupan 64 bits u 8 bytes cada uno. Esto puede cambiar entre sistemas en casos muy reducidos<sup>[4]</sup>, fuera del estándar.

El resto de información en el fichero son chunks de estructuras que representan cada punto, tantos como se especifica en la primera variable. La dimensión de cada chunk de estos debería ser consistente entre distintos PCs, ya que los tipo `double` siempre deben de medir lo mismo (8 bytes, por lo que la estructura `vector2D` mide 16 bytes).

## Ejecución

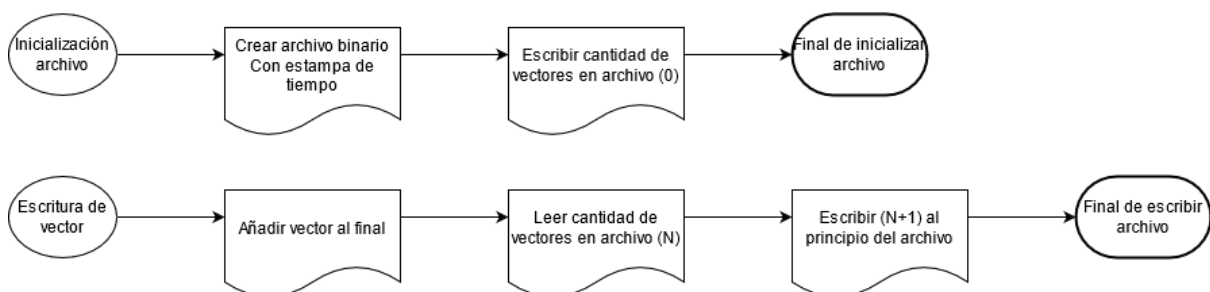


Diagrama de ejecución de las funciones que inicializa el archivo y escribe un vector en él

Lo primero es inicializar el archivo con `waypts_bcreate_file()`. Este lo crea en binario con el nombre especificado o uno generado en función de la fecha y hora, minuto y segundo. Posteriormente añade la primera variable, el número de vectores que contiene en ese momento (0). Se genera automáticamente cuando la cadena está vacía o se usa la macro `MODE_TIMESTAMP` (equivalente a `"\0"`) en el argumento del nombre. Además, el modo es de lectura y escritura (que si no existe se crea) en binario. Esto último es importante para que la función de añadir puntos luego funcione correctamente.

Cada vez que se quiere añadir un punto de la trayectoria, se hace una llamada a la función `waypts_bappend_vect()`. Esta coge el vector que se le ha pasado y lo concatena al final del archivo. Si pudo, ahora va al principio a leer la cantidad de vectores que el archivo dice haber, lo incrementa en 1, vuelve al principio y lo escribe de nuevo. Así queda

actualizado el archivo finalmente. Si en algún momento no puede realizar alguna operación se devuelve un error. El archivo, si se hubo modificado, modificado quedó.

Por último, para hacer la lectura de un vector, se usa `waypts_bread_vect()`; los argumentos que necesita son el archivo de puntos, el array o puntero a vector al que se va a escribir los datos leídos, el tamaño de este búfer, desde qué posición se desea leer, y qué cantidad de puntos se desea obtener. Lo primero que hace es colocar la posición del búfer en un offset equivalente al tamaño de la primera variable más la posición del vector a leer por el tamaño de cada `vector2D`.

## Macros

Macro	Valor
<code>MODE_TIMESTAMP</code>	<code>"\0"</code>

## Funciones

1. `int waypts_bcreate_file(FILE** fp, const size_t sz, const char filename[], char* assigned)`
  - a. Crea e inicializa el archivo de puntos por los que pasa el robot. Devuelve cero si consigue hacer el archivo [devuelve lo mismo que `fopen_s()` o `fwrite()` al inicializarlo]. Argumentos:
    - i. `fp`: puntero a puntero de archivo (`FILE*`) (se modifica),
    - ii. `sz`: tamaño máximo del nombre archivo
    - iii. `filename`: nombre archivo o protocolo a utilizar (`MODE_TIMESTAMP` ó `"\0"` para usar la estampa de tiempo)
    - iv. `assigned`: puntero a nombre asignado al archivo finalmente, tamaño mayor o igual al especificado en `sz`
  - b. Devuelve los errores (0 indica éxito):
    - i. 100: no puede asignarse memoria dinámica
    - ii. 150: el nombre especificado no es adecuado (salida por pantalla del error)
    - iii. Ó el retornado por `fopen_s` o `feof`. Puede coincidir con alguno de los anteriores
2. `int waypts_bget_nvects(FILE* fp, uint32_t* dest)`
  - a. Toma la cantidad de puntos en un archivo con el formato aquí descrito y lo escribe en `dest`. Argumentos:
    - i. `fp`: puntero al archivo
    - ii. `dest`: puntero a variable a la que se asigna la cantidad de puntos
  - b. Devuelve 0 si funciona correctamente.
3. `int waypts_bappend_vect(FILE* fp, vector2D* src)`
  - a. Añade 1 vector de dos componentes (`src`) al archivo apuntado por `fp`, en binario Argumentos:
    - i. `fp`: puntero a archivo que se modifica

- ii. `src`: vector que se concatena al archivo
  - b. Devuelve 0 si funciona correctamente.
- 4. `int waypts_bread_vect(FILE* fp, vector2D* dest, size_t bufferSize, uint32_t start, uint32_t N)`
  - a. Lee N-vectores desde el punto en la posición `start`
    - ii. `fp`: puntero a archivo desde el que se lee
    - iii. `dest`: puntero a array de vectores en el que se guardan los valores
    - iv. `bufferSize`: tamaño del array (puede ser `sizeof(vector2D)`)
    - v. `start`: elemento desde el que se empieza a leer (el primer elemento tiene la posición 0)
    - vi. `N`: cantidad de vectores a leer
  - b. Devuelve 0 si funciona correctamente. En caso de error sale con un código mayor de 210.

## Versiones del documento: Fecha e historial de modificaciones en el documento.

El documento fue creado el 7 de Mayo de 2021 para coincidir con la primera entrega del proyecto, aunque gran parte de la información detallada aquí proviene de documentos que ya existían en la Wiki del proyecto, creados durante el último mes mayoritariamente.

A lo largo de las tres semanas siguientes al 7 de mayo fuimos editando el documento para entregarlo a tiempo el día 28 junto con el proyecto terminado y un vídeo explicando todo.

Queremos poner toda la información acerca del funcionamiento del proyecto a nivel de código y sistemas, y dejar la wiki y el readme para las especificaciones más prácticas y de uso una vez esté enteramente funcional.

Historial de las modificaciones:

- **7 de mayo:** Creación del documento
- **15 de mayo:** Actualización del primer apartado (problemas a resolver)
- **16 de mayo:** Reunión de grupo y actualización entre todos
- **20 de mayo:** Nueva actualización (teniendo en cuenta sugerencias y comentarios anteriores)
- **23 de mayo:** Nueva actualización, reubicando párrafos y cambiando títulos
- **25 de mayo:** Nueva actualización; añadir [Esquema electrónico del robot](#) y actualizado [Algoritmos y código como conjunto](#)
- **27 de mayo:** Añadir partes faltantes y último repaso.

## Enlaces

1. Wiki; Descripción detallada de sensores y actuadores  
[https://github.com/aigora/twIA\\_2021-carmap\\_chikito/wiki/1.-Descripci%C3%B3n-detallada-e-identificaci%C3%B3n-de-sensores-y-actuadores](https://github.com/aigora/twIA_2021-carmap_chikito/wiki/1.-Descripci%C3%B3n-detallada-e-identificaci%C3%B3n-de-sensores-y-actuadores)
2. Wiki: Diseño del sistema  
[https://github.com/aigora/twIA\\_2021-carmap\\_chikito/wiki/2.-Dise%C3%B1o-del-Sistema](https://github.com/aigora/twIA_2021-carmap_chikito/wiki/2.-Dise%C3%B1o-del-Sistema)
3. PDF en moodle  
[https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/9266173/mod\\_resource/content/1/Comunicaci%C3%B3n\\_Bluetooth\\_Arduino\\_PC.pdf](https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/9266173/mod_resource/content/1/Comunicaci%C3%B3n_Bluetooth_Arduino_PC.pdf)
4. Página consultada sobre Arduino y C++  
<https://geekytheory.com/como-usar-arduino-y-cplusplus>
5. LB1641 Datasheet  
<https://www.alldatasheet.com/datasheet-pdf/pdf/439406/SANYO/LB1641.html>

6. Github - Gmatarrubia/LibreriasTutoriales  
<https://github.com/Gmatarrubia/LibreriasTutoriales>
7. Precisión doble (Wikipedia)  
[https://en.wikipedia.org/wiki/Double-precision\\_floating-point\\_format](https://en.wikipedia.org/wiki/Double-precision_floating-point_format)
8. KiCad\_system en Robot\_CMC  
[https://github.com/aigora/twIA\\_2021-carmap\\_chikito/tree/master/Robot\\_CMC/KiCad\\_system](https://github.com/aigora/twIA_2021-carmap_chikito/tree/master/Robot_CMC/KiCad_system)
9. Arduino: Keyboard.h reference  
<https://www.arduino.cc/reference/en/language/functions/usb/keyboard/>

## Otros sitios consultados durante el desarrollo

- Repositorio de un exalumno que dio una charla en el CREA sobre un teclado de macros: [https://bitbucket.org/AlexAlc/macro\\_keyboard/src/master/](https://bitbucket.org/AlexAlc/macro_keyboard/src/master/)
- Más teclados de macros:
  - <https://youtu.be/adVdUyBJf6E>
  - <https://youtu.be/72a85tWOJYY>
- Quitar el debouncing en pulsadores:
  - <https://www.eejournal.com/article/ultimate-guide-to-switch-debounce-part-4/>
- Pinout del Arduino UNO, para realizar el modelo en KiCad.
  - <https://www.circuito.io/blog/arduino-uno-pinout/>
- Recursos publicados en Moodle:
  - Paradigma modular
  - Diseño del sistema
  - Comunicación Bluetooth Classic: Arduino - Visual C++ 2019
  - Gotoxy en Dev-C++