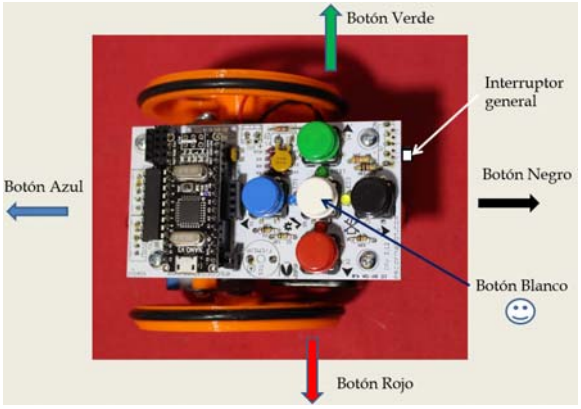
Entender el Programa de Funcionamiento de Escornabot

Introducción

Escornabot puede ejecutar secuencias de movimientos que son programados por el usuario mediante la pulsación de las teclas del robot. También se puede ampliar su control manejándose desde un teléfono móvil o una "tablet" a través de Bluetooth o Wifi.

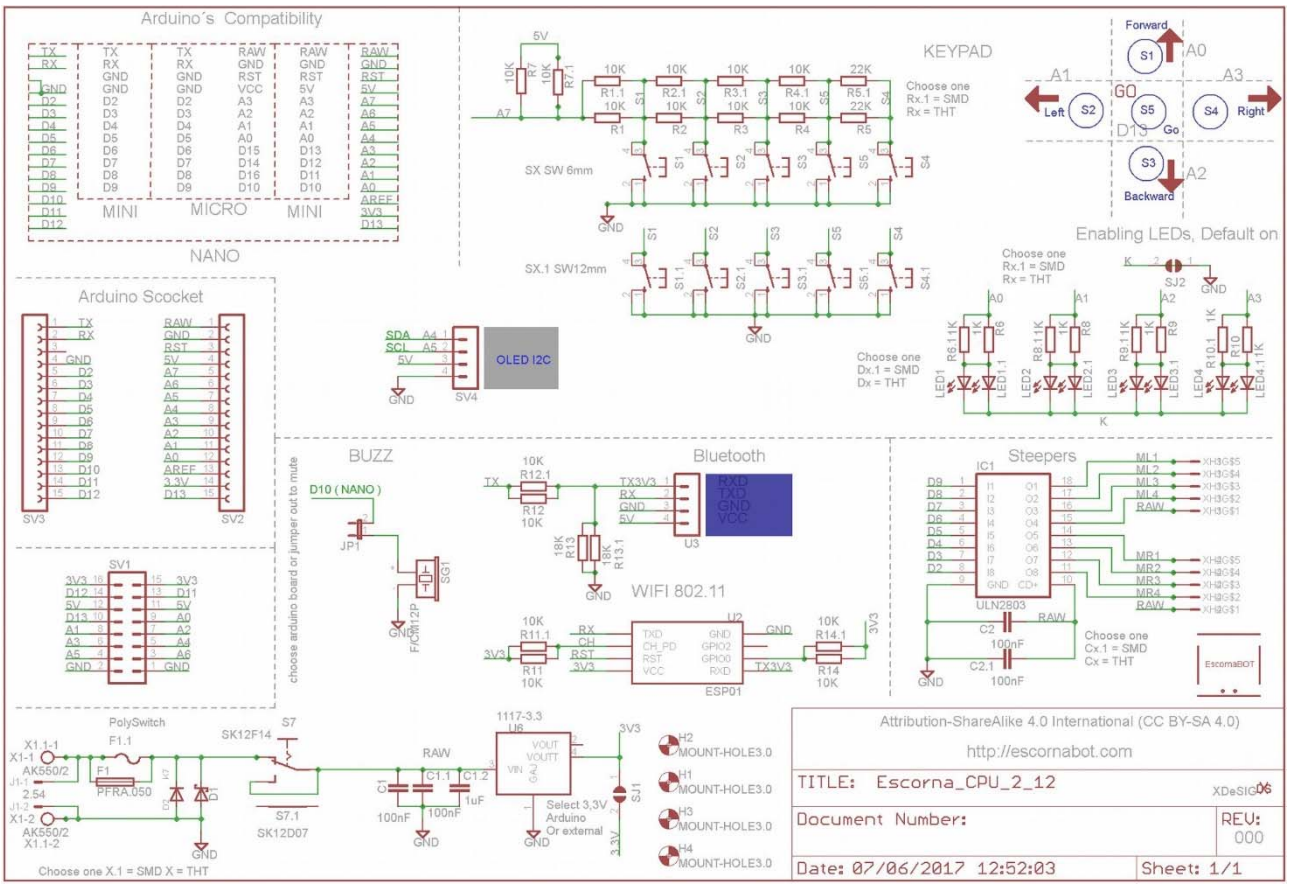
El Escornabot dispone de un interruptor general de puesta en marcha y 4 botones que permite programar los movimientos que queremos que haga, cada pulsación programada hará avanzar 10 cm, permite mover hacia adelante (tecla azul), mover hacia atrás (tecla negra), girar a la izquierda (tecla roja), girar a la derecha (tecla verde) y la tecla del medio de color blanco hará que ejecute la secuencia programada. También la tecla blanca puede parar la secuencia cuando el Escornabot está en marcha.

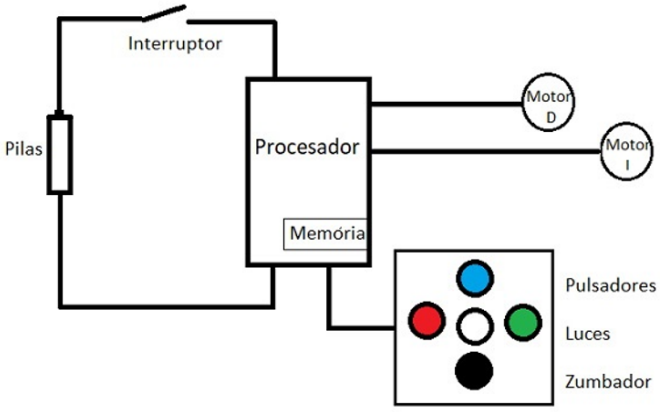


Componentes de Escornabot versión Singularis

- ⊡ 2 motores para mover a cada rueda
- ⊡ 1 procesador para la secuencia las ordenes hacia los motores y LEDs de señalización.
- ⊡ 1 memoria para guardar las órdenes.
- ⊡ 5 teclas para pulsar las órdenes.
- ⊡ 5 luces de indicación.
- ⊡ Zumbador para oir que está haciendo.
- ⊡ 4 pilas para dar energía al Escornabot.
- ⊡ Interruptor para poner en marcha.

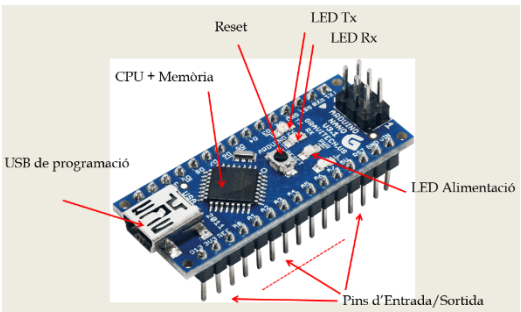
Esquema del Escornabot Singularis



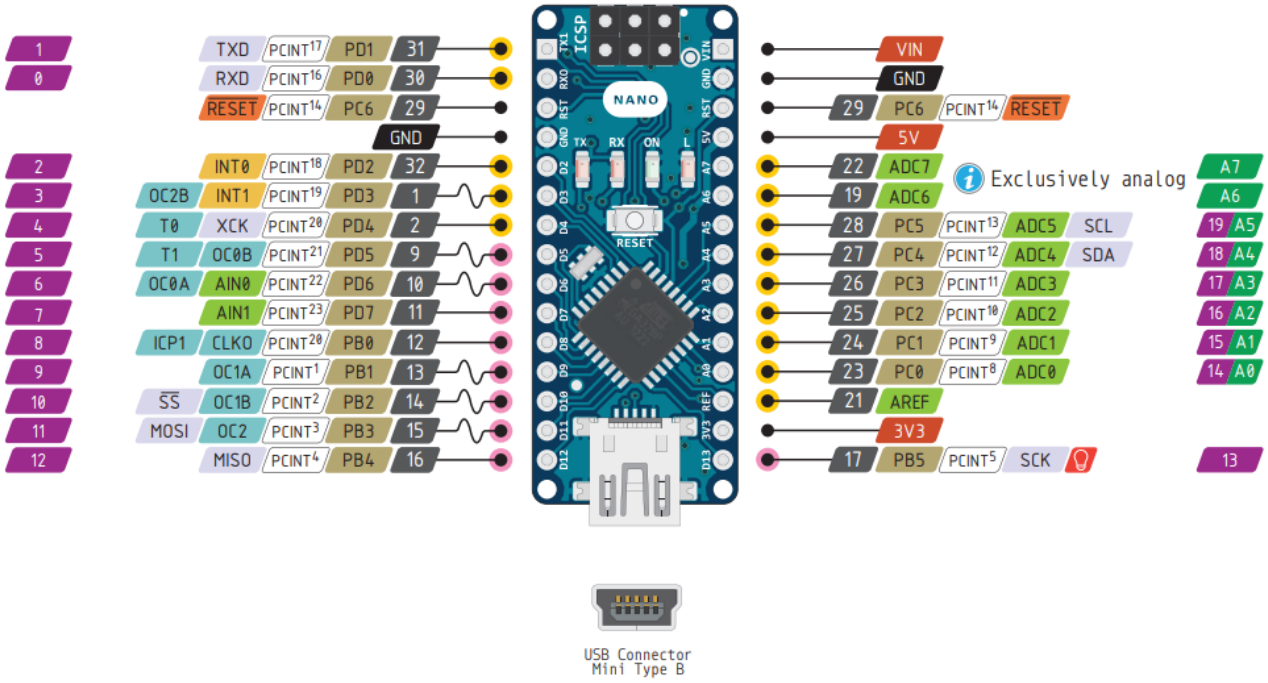


Procesador Arduino

La unidad de control de proceso Arduino es el cerebro del robot, es donde se guardan las instrucciones recibidas desde el teclado y donde se controlan los motores, LEDs y zumbador. También incorpora un conector USB para poder actualizar el firmware del robot.



Esta placa de circuito impreso de Arduino (llamada Arduino Nano) contiene un microcontrolador de 8 bit del fabricante Microchip de la familia AVR (antes Atmel) ATMEGA328P, incorpora en su interior la unidad central de proceso (CPU), una memoria Flash de 32KBytes (donde se alberga el programa y se puede reprogramar), una memoria RAM 2 KBytes (para almacenar valores temporalmente, ya que cuando se quita alimentación se pierden estos datos y una memoria Eeprom 1 KBytes (donde se guardan datos durante la ejecución de un programa y permanecerán aunque se quite alimentación).



Entradas analógicas van al convertidor analógico digital (ADC de 10 bit) y las salidas analógicas pueden ser PWM (para el control de motores).

Entradas/salidas digitales y otras permiten una comunicación serie (I2C, SPI y UART). Un pin de Reset que va al pulsador de la placa.

Una parte imprescindible es un reloj de cuarzo de 16 MHz con un oscilador de hasta 20MHz que equivale a poder ejecutar 20 millones de instrucciones por segundo.

Un apunte más sobre la memoria Flash, es donde se programa un software llamado Bootloader (que ocupa 0,5 KBytes), permite cargar el programa que realicemos vía serie a través del USB.

Esta placa se alimenta con 5V

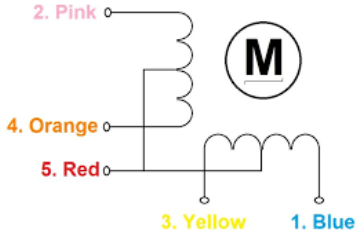
Motor paso a paso

El Escornabot incorpora dos motores paso a paso, quiere decir que se mueven por impulsos, dentro del motor incorporan una reductora, cada 64 pulsos dan una vuelta, que permite una gran precisión. Según la frecuencia de los pulsos proporcionan la velocidad.

Si giran en el mismo sentido el robot avanza o retrocede, si uno gira en un sentido y el otro en sentido contrario, el robot girará 90 grados hacia la derecha o 90 grados hacia la izquierda.

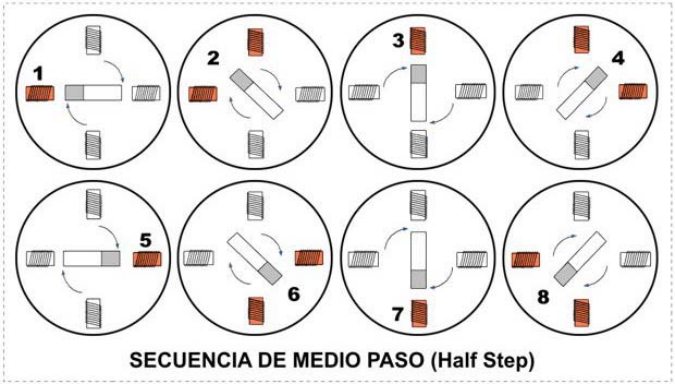
Hay una nueva revisión de firmware, la 1.6, que puede girar a 60 y a 90 grados, esto es para usuarios más grandes, y permite ir en diagonal.

Este tipo de motor paso a paso es de 4 fases Unipolar y se utiliza la conmutación de Medio Paso.



Secuencia de conmutación de Medio Paso

Número y color de pata	→ Dirección de la agujas del reloj (fases 1-2)							
	1	2	3	4	5	6	7	8
(4) Naranja	■	■						■
(3) Amarillo		■	■	■				
(2) Rosa				■	■	■		
(1) Azul						■	■	■



SECUENCIA DE MEDIO PASO (Half Step)

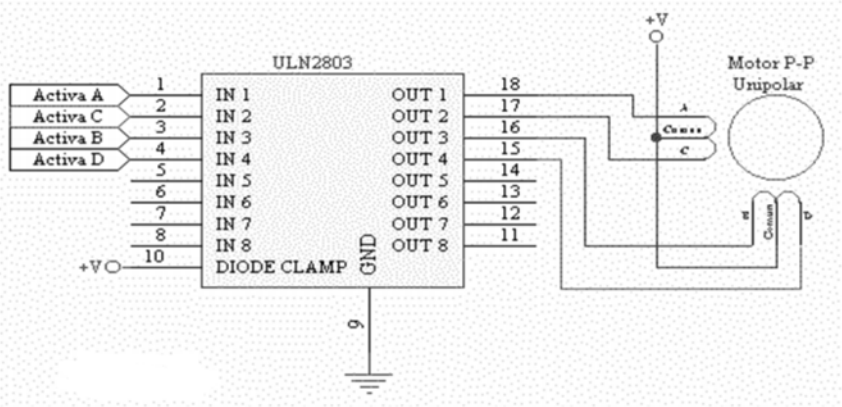
En esta configuración de control llamada de Medio Paso, las dos bobinas que tienen un punto medio formando cuatro bobinas, se puede ver cómo se van alternando los pulsos en la tabla anterior para realizar los movimientos paso a paso. Se necesitan 4 pasos, un giro en ocho ciclos y un giro completo del eje exterior 64 vueltas del roto, por lo que se requieren 2048 pasos para una vuelta.

$$\text{Nº de pasos para una vuelta} = 4 \times 8 \times 64 = 2048$$

Con el siguiente código conseguimos una vuelta completa del motor:

```
const static uint8_t step_pattern[] = {
                B00001, B00011, B00010, B00110, B00100, B01100, B01000, B01001
                                                        } ;
```

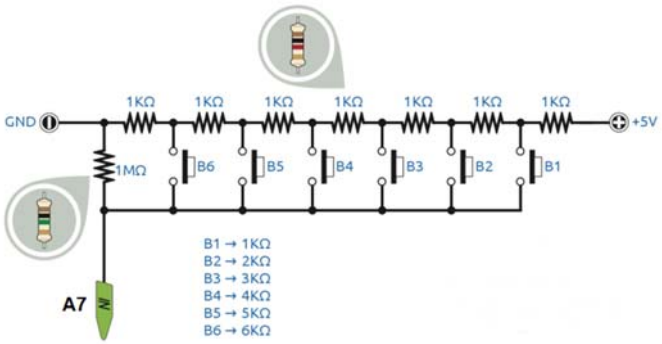
Se puede constatar que el patrón del código corresponde a la tabla de secuencia de conmutación de Medio Paso, cada 1 en la tabla aparece un punto en rojo.



La corriente de cada bobina es relativamente alta y es necesario poner un "driver" para que el microcontrolador no entregue esta corriente si no la entrega cada transistor interno.

Control del teclado

El teclado está minimizado para el uso de un solo pin, utilizando una entrada analógica de la siguiente manera.

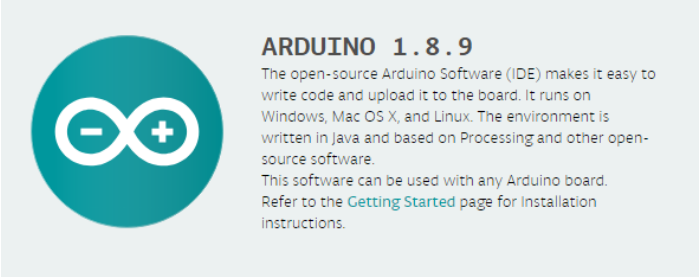


El convertidor analógico/digital del microcontrolador "lee" el valor de voltaje del pin de entrada A7 y "entiende" qué botón se ha pulsado.

¿Cómo se programa?

Escornabot usa un módulo de Arduino, muy popular en el mundo DIY, donde se tiene que grabar el firmware de funcionamiento, que está totalmente disponible y documentado en https://github.com/escornabot

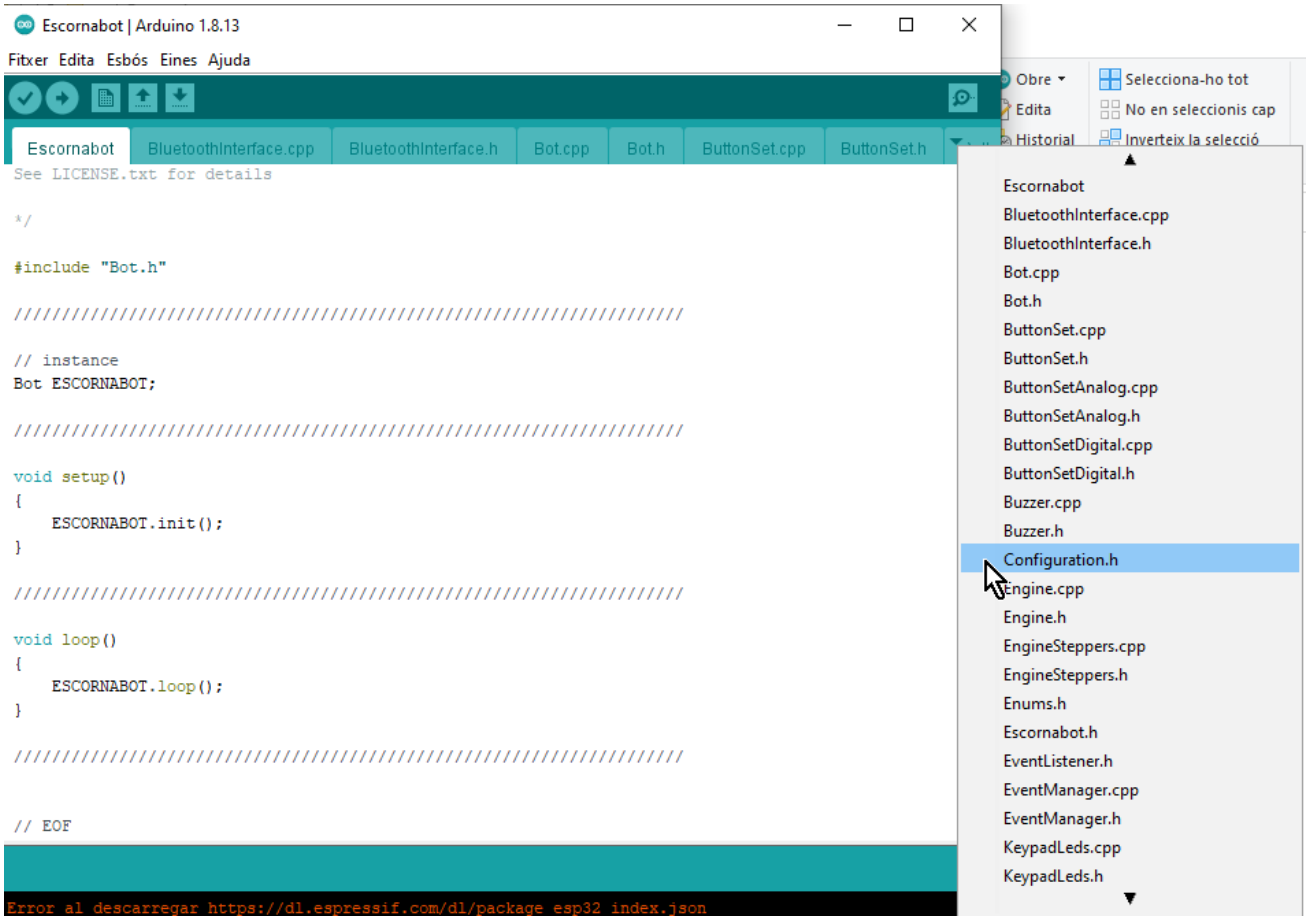
Se usa el entorno de desarrollo de Arduino y se puede hacer como taller de Informática el desarrollo completo o parcial del funcionamiento, evidentemente adecuando a la edad correspondiente.



Entender el programa de funcionamiento

El programa de este Escornabot ha sido realizado por @caligari y seguidamente se va a describir la parte principal del funcionamiento.

Al abrir el firmware 1.4.3 o 1.6.2 Escornabot.ino con el entorno de Arduino, aparecen una serie de pestañas donde se muestran cada parte control.



La estructura del lenguaje de programación es parecida al C++, el programa se puede ejecutar en dos partes:

- void setup() es la configuración y se encuentra siempre al comienzo del programa, se configuran los pinMode, se inicializa la comunicación serie, etc y se ejecutará una sola vez después de cada encendido o después de un Reset con el pulsador de la placa.
- void loop() es la ejecución del código, como lectura de entradas y activación de salidas.

El archivo Configuration.h es dónde se aloja la parte más importante para entender el funcionamiento, la Configuración.

Cuando hay la línea empieza con dos // o más quiere decir que lo que viene a continuación es sólo comentarios que ha puesto el programador para entender su programa. El programa original está en color verde y los comentarios adicionales para detallar en color rojo.

Si no se tiene conocimientos de programación en C++ o C para Arduino, se puede ver una documentación muy bien detallada http://manueldelgadocrespo.blogspot.com/p/se-utiliza-para-la-comunicacion-entre_3.html

```
//////////////////////////////////////////////////////////////////
///// general configuration
//////////////////////////////////////////////////////////////////

// engine to use   Define el motor a usar, este caso motor paso a paso
#define ENGINE_TYPE_STEPPERS
```

#define en C es un componente útil que permite al programador para dar un nombre a un valor constante antes de compilar el programa. Las constantes definidas en Arduino no ocupan ningún espacio de memoria de programa en el chip. El compilador reemplaza las referencias a estas constantes con el valor definido en tiempo de compilación. En Arduino **define** tiene la misma sintaxis que **define** en C:

No hay punto y coma después de la instrucción #define. Si se incluye uno, el compilador genera errores. Tampoco puede haber un signo (=) a un número valor (5) o (true), por ejemplo. #

```
// button set to use (analog input, digital input)  Configura el tipo de entrada que proviene de pulsadores Analógicos o Digitales
#define BUTTONS_ANALOG   Define el tipo de entrada que proviene de pulsadores Analógicos
//#define BUTTONS_DIGITAL  Define el tipo de entrada que proviene de pulsadores Digitales (no está activado)

// milliseconds after a button is considered as pressed  Define cuantos milisegundos, como mínimo, tiene que estar pulsado un botón, para evitar pulsos parásitos inesperados en esta línea de pulsadores.
#define BUTTON_MIN_PRESSED 30     // En este caso 30 ms

// milliseconds after a button is considered as long pressed  Define cuantos milisegundos, cómo mínimo, tiene que estar un pulsador un botón, para considerar que se trata de una pulsación “larga”, para configurar giros de 60 grados.
#define BUTTON_LONG_PRESSED 1000   // En este caso 1000 ms que determina pulsación larga

// put to false to add movements to the program after its execution
#define PROGRAM_RESET_ALWAYS true   // Hace un Reset de los movimientos al final de ejecutarse un ciclo programado.

// store configuration and program within internal EEPROM
#define USE_PERSISTENT_MEMORY false // Serviría para almacenar en la Eeprom los movimientos, pero no se utiliza esta funcionalidad

// memory capacity for program movements   Define la cantidad de movimientos que se pueden almacenar en la memoria RAM
#define MOVE_LIMIT 100   // En este caso se podrán almacenar hasta 100 movimientos, se puede ampliar el número de movimientos programados, ocupando más memoria RAM, que se comparte con la que usa el propio programa, pero típicamente es suficiente. El microcontrolador ATMega328P de la placa Arduino dispone de 2K bytes de RAM y no sobra mucha, pero se podría, si es necesario, configurar para almacenar más movimientos, pero no se ha probado el límite).
```

// milliseconds for the "pause" movement Define cuantos milisegundos tendrá que estar pulsado el botón para programar una pausa
#define PAUSE_MOVE_MILLIS 1000 // En este caso 1000 ms, que determina una Pausa

// milliseconds delay before starting to move Define cuantos milisegundos esperará el robot para empezar a moverse
#define DELAY_BEFORE_GO 500 // En este caso se empezará a mover después de 500 ms

// milliseconds to pause after every movement Define cuantos milisegundos tendrá que estar pulsado el botón para programar una pausa
#define AFTER_MOVEMENT_PAUSE 0 // En este caso se empezará a mover después 0 ms después de una Pausa

// point of view set when Vacalourabot is started **Escornabot** (antes *Vacalourabot*)
#define POV_INITIAL POV_ESCORNABOT // Punto de vista (Point Of View) de la ejecución de los movimientos, es decir desde el usuario o desde el robot, pero no está implementado

// bluetooth serial Define el puerto serie que proviene del módulo Bluetooth
#define USE_BLUETOOTH true // Define el uso de modulo Bluetooth
#define BLUETOOTH_BAUDS 9600 // Define la velocidad de transmisión del módulo Bluetooth

// buzzer Define el Zumbador
#define USE_BUZZER true // Define el uso de modulo Bluetooth
#define BUZZER_PIN 10 // Define el pin usado por el Zumbador (en este caso D10)
#define PROGRAM_FINISHED_RTTL RTTL_FIDO // Buzzer,h // Define la musiquilla al **final de la ejecución**
#define TONE_FREQ_UP 2637 // Define la frecuencia de 2637 (que corresponde a un Mi octava 7 musical) para señalizar que va **hacia adelante**
#define TONE_FREQ_RIGHT 4434 // Define la frecuencia de 4434 (que corresponde a un Do# octava 8 musical) para señalizar que **gira a la derecha**
#define TONE_FREQ_DOWN 3520 // Define la frecuencia de 3520 (que corresponde a un La octava 7 musical) para señalizar que va **hacia atrás**
#define TONE_FREQ_LEFT 2217 // Define la frecuencia de 2217 (que corresponde a un Do# octava 7) para señalizar que **gira a la izquierda**

Control del Zumbador

Una salida del microcontrolador PWM genera frecuencias que se pueden reproducir em un zumbador

En esta web se pueden encontrar cada nota musical con su correspondiente frecuencia y octava correspondiente
https://juegosrobotica.es/musica-con-arduino/#

Control del LED

// simple led Define el LED
#define USE_SIMPLE_LED false // Define el uso del LED
#define SIMPLE_LED_PIN 13 // Define el pin 13 para uso del LED

// keypad leds Define los LED de los pulsadores
#define USE_KEYPAD_LEDS true

#define KEYPAD_LED_PIN_UP A0 // Define el pin A0 para el LED para señalizar que va **hacia adelante**
#define KEYPAD_LED_PIN_RIGHT A3 // Define el pin A3 para uso del LED para señalizar que **gira a la derecha**
#define KEYPAD_LED_PIN_DOWN A2 // Define el pin A2 para uso del LED para señalizar que va **hacia atrás**
#define KEYPAD_LED_PIN_LEFT A1 // Define el pin A1 para uso del LED para señalizar que **gira a la izquierda**
#define KEYPAD_LED_PIN_GO 13 // Define el pin 13 para el LED para señalizar que se **pone en marcha**

//
///// Steppers engine setup // Configura los motores paso a paso
//

#ifdef ENGINE_TYPE_STEPPERS

#ifdef permite compilar una sección de un programa solo si la macro que se especifica como parámetro ha sido definida, sin importar cuál sea su valor.
// stepper pin setup (digital outputs) Configuración de los pines de control de los motores paso a paso
#define STEPPERS_MOTOR_RIGHT_IN1 5
#define STEPPERS_MOTOR_RIGHT_IN2 4
#define STEPPERS_MOTOR_RIGHT_IN3 3
#define STEPPERS_MOTOR_RIGHT_IN4 2
#define STEPPERS_MOTOR_LEFT_IN1 9
#define STEPPERS_MOTOR_LEFT_IN2 8
#define STEPPERS_MOTOR_LEFT_IN3 7
#define STEPPERS_MOTOR_LEFT_IN4 6

// stepper pin setup (digital outputs) Configuración de los pines de control de los motores paso a paso con giro inverso
#define STEPPERS_MOTOR_RIGHT_IN1 2
#define STEPPERS_MOTOR_RIGHT_IN2 3
#define STEPPERS_MOTOR_RIGHT_IN3 4
#define STEPPERS_MOTOR_RIGHT_IN4 5
#define STEPPERS_MOTOR_LEFT_IN1 6
#define STEPPERS_MOTOR_LEFT_IN2 7
#define STEPPERS_MOTOR_LEFT_IN3 8
#define STEPPERS_MOTOR_LEFT_IN4 9

// step calibration Configuración de la calibración de los pasos
#define STEPPERS_STEPS_PER_SECOND 1000 // Define el número de pasos por segundo, se puede modificar hasta 2300 pasos, pero depende del motor y también de la alimentación de la batería
#define STEPPERS_LINE_STEPS 1738 // Define el número de pasos para cada paso (10 cm), se puede crear un tapete de juego con dimensiones diferentes, 1cm será de 174. A partir de aquí podemos cambiar el valor para que avance lo que queramos.
#define STEPPERS_TURN_STEPS 1024 // Define el número de pasos para cada giro (90 grados), para girar a 45 grados se puede modificar a 512 pasos, En caso de decimales redondear el valor.
Un juego; practicar las horas del reloj variando los ángulos de giro.

#endif

Estas directivas permiten incluir o descartar parte del código de un programa si se cumple una determinada condición.
#ifdef permite compilar una sección de un programa solo si la macro que se especifica como parámetro ha sido definida, sin importar cuál sea su valor.
Por ejemplo, digamos que tienes una aplicación y te gustaría que cuando estés desarrollando, siempre inicie con un usuario determinado.

En este caso las estas directivas te pueden ayudar:

```
#define PRUEBA

#ifdef PRUEBA
  // CODIGO PARA CARGAR USUARIO POR DEFECTO

#endif
```

El código dentro de la directiva solo será compilado si PRUEBA está definida. Así cuando vayas a publica tu aplicación, solo tendieras que eliminar la definición de PRUEBA y el programa seguirá su flujo normal.

Control de los Pulsadores

Como se ha mostrado al principio los pulsadores están enlazados con resistencias a un solo pin, en el archivo Configuration.h

Por lo que esta parte de configuración Digital no está activada

```
//////////////////////////////////////////////////////////////////
///// Button set digital
//////////////////////////////////////////////////////////////////

#ifdef BUTTONS_DIGITAL

// keypad pin setup (digital or analog inputs) (use 255 if key doesn't exist)
#define BS_DIGITAL_UP A0
#define BS_DIGITAL_RIGHT A1
#define BS_DIGITAL_DOWN A2
#define BS_DIGITAL_LEFT A3
#define BS_DIGITAL_GO A4
#define BS_DIGITAL_RESET 255

#endif // BUTTONS_DIGITAL
```

Esta parte si que es la que funciona

```
//////////////////////////////////////////////////////////////////
///// Button set analog
//////////////////////////////////////////////////////////////////

#ifdef BUTTONS_ANALOG

#define BS_ANALOG_WIRES 2   // WIRE 2 ?
//#define BS_ANALOG_WIRES 3    // no se usa
```

```
// keypad pin setup (analog input)
#define BS_ANALOG_PIN A7    // Configura el pin A7 como entrada de los Pulsadores

// input values for each key pressed (0 if key doesn't exist)
#define BS_ANALOG_VALUE_UP 512    // Valores para Pulsador Adelante
#define BS_ANALOG_VALUE_RIGHT 860  // Valores para Pulsador Derecha
#define BS_ANALOG_VALUE_DOWN 769  // Valores para Pulsador Atras
#define BS_ANALOG_VALUE_LEFT 683  // Valores para Pulsador Izquierda
#define BS_ANALOG_VALUE_GO 810  // Valores para Pulsador  GO
#define BS_ANALOG_VALUE_RESET 0  // Valores para RESET ?

#endif // BUTTONS_ANALOG


//////////////////////////////////////////////////////////////////////
///// Button set Bluetooth   //Configuración del Bluetooth
//////////////////////////////////////////////////////////////////////

#ifdef USE_BLUETOOTH

// Arduino serial port (default is Serial, use Serial1 with Arduino Micro)
//#define BLUETOOTH_SERIAL    Serial    si quitamos las barras  // ya deja preparado para recibir datos a
través del modulo Bluetooth
//#define BLUETOOTH_SERIAL    Serial1
//#define BLUETOOTH_SERIAL    Serial2
//#define BLUETOOTH_SERIAL    Serial3

#endif // USE_BLUETOOTH
```

Modo de juego a 90 o 60 grados

El firmware está preparado para trabajar a 90 grados y a 60 grados con pulsación larga. La selección del tipo de juego viene definida en el archivo Enums.h

```
// game modes
enum
{
   GAME_MODE_GRID_90 = 0, // squared grid (classical mode)
   GAME_MODE_GRID_60 = 1, // triangled grid
};
typedef uint8_t GAME_MODE;
```

Programación de los movimientos de "Saludo"

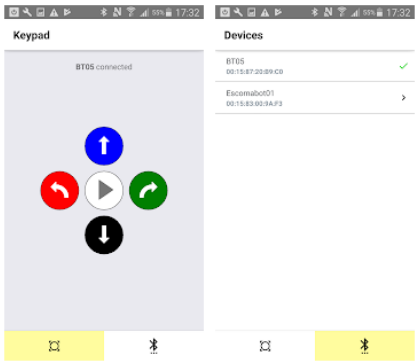
Los movimientos de "saludo" inicial al dar alimentación al Escornabot, están programados en la memoria Eeprom del microcontrolador, dónde quedan almacenadas, aunque se quite la alimentación y en el firmware está en el archivo MoveList.h

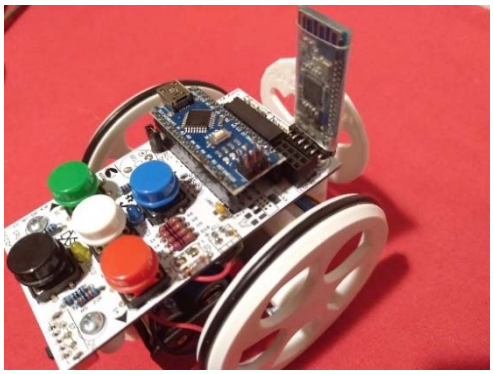
```
const static MOVE PROGRAM_ESCORNA_GREETING[] = {
   MOVE_FORWARD,    // Mueve hacia adelante
   MOVE_LEFT,       // Mueve hacia la izquierda
   MOVE_RIGHT,      // Mueve hacia la derecha
   MOVE_RIGHT,      // Mueve hacia la derecha
   MOVE_LEFT,       // Mueve hacia la izquierda
   MOVE_BACKWARD,   // Mueve hacia atrás
   MOVE_NONE
};
```

Si se quieren programar otros movimientos, solo hay que modificar, quitar o añadir las líneas que se muestran en el archivo MoveList.h . El número de movimientos como se puede ver son 6, pero la memoria Eeprom permite una cantidad elevada de movimientos, tiene 1 KBytes

Escornabot sigue creciendo

Además, Escornabot está preparado para añadir un módulo de Bluetooth tipo HM-10 con su App disponible para Android y IOs.





También está preparado para un módulo Wifi, aunque no es recomendable típicamente por el alto consumo con respecto al módulo Bluetooh, pero es posible.

El control remoto no es recomendable para los usuarios de menor edad, ya que es mejor trabajar encima de una mesa o en el suelo. En cambio, el uso de control remoto permite realizar otro tipo de prácticas como el diseño de un control desde un smartphone con AppInventor.

EngineSteppers.cpp

Es dónde hace girar en un sentido u otro los motores

```
void EngineSteppers::init()
{
   pinMode(_config->motor_left_in1, OUTPUT);
   pinMode(_config->motor_left_in2, OUTPUT);
   pinMode(_config->motor_left_in3, OUTPUT);
   pinMode(_config->motor_left_in4, OUTPUT);
   pinMode(_config->motor_right_in1, OUTPUT);
   pinMode(_config->motor_right_in2, OUTPUT);
   pinMode(_config->motor_right_in3, OUTPUT);
   pinMode(_config->motor_right_in4, OUTPUT);

   // set coils in row // justo al alimentar hace este pequeño movimiento para estabilitzar la posición de las bobines de los motores
   _movement_steps_r = 8;
   _movement_steps_l = 8;

   EVENTS->add(this);
}
```