Coding Escornabot

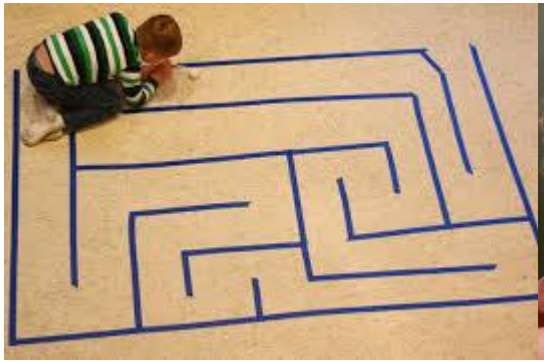
@procastino & @caligari

XIII Xornadas Libres

FIC - 10 outubro 2014

Que é un *Escornabot*?

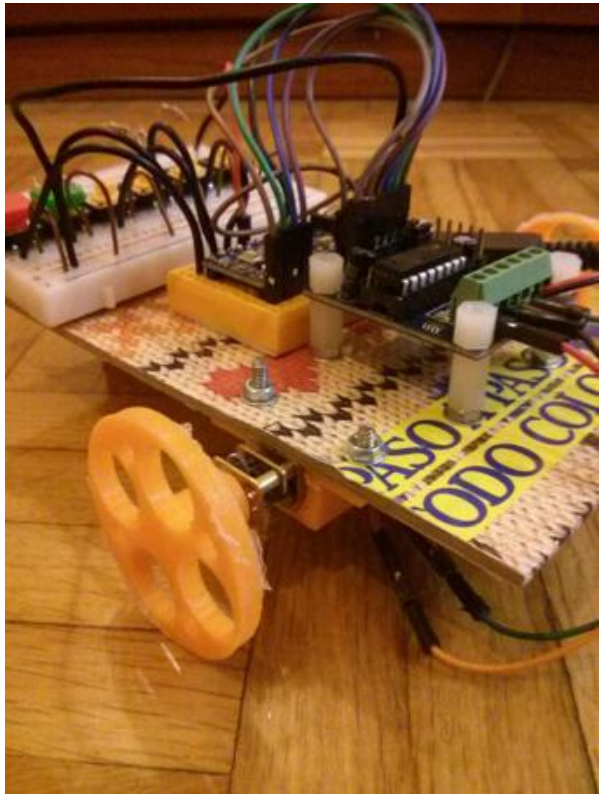
Escornabot e un **xoguete-robot** que se despraza con ordes moi simples de dirección **adiante-atrás** e xiro **esquerda-dereita**.



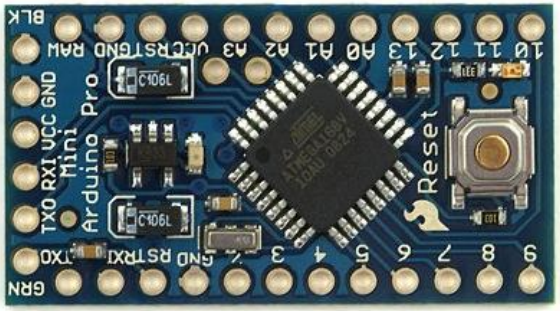
E primeiro foi o hardware

Unha interesante montaxe de *@procastino* nunha "tapa dura de libro".

Bricolabs, 11 de xullo de 2014



Un sketch para o Arduino



Arduino Pro Mini

Arduino Nano

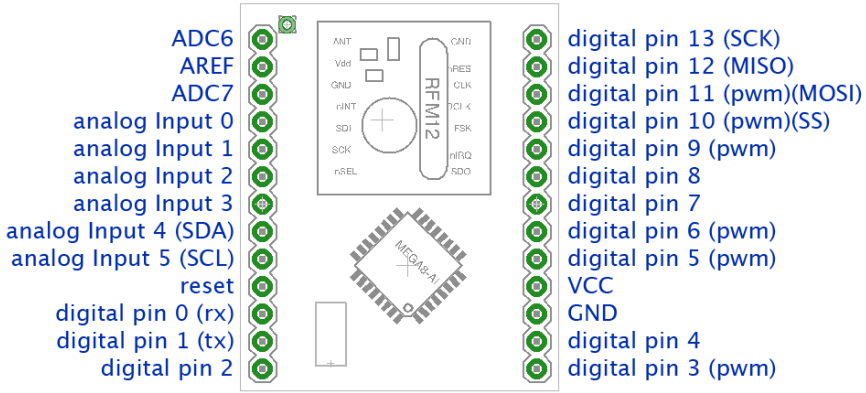
Requisitos de usuario:

- ❏ Ler a botoneira e almacenar os movementos
- ❏ Executar os movementos na maquinaria *HBridge*

Límites hardware a ter en conta

	ATMega168	ATMega328	ATmega1280
Flash (2k for boobtloader)	16kB	32kB	128kB
SRAM	1kB	2kB	8kB
EEPROM	512B	1kB	4kB

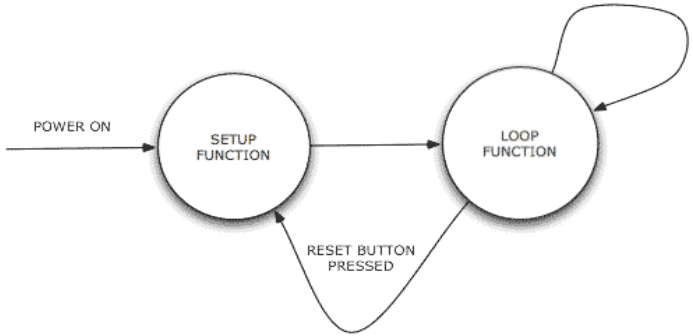
Arduino pins



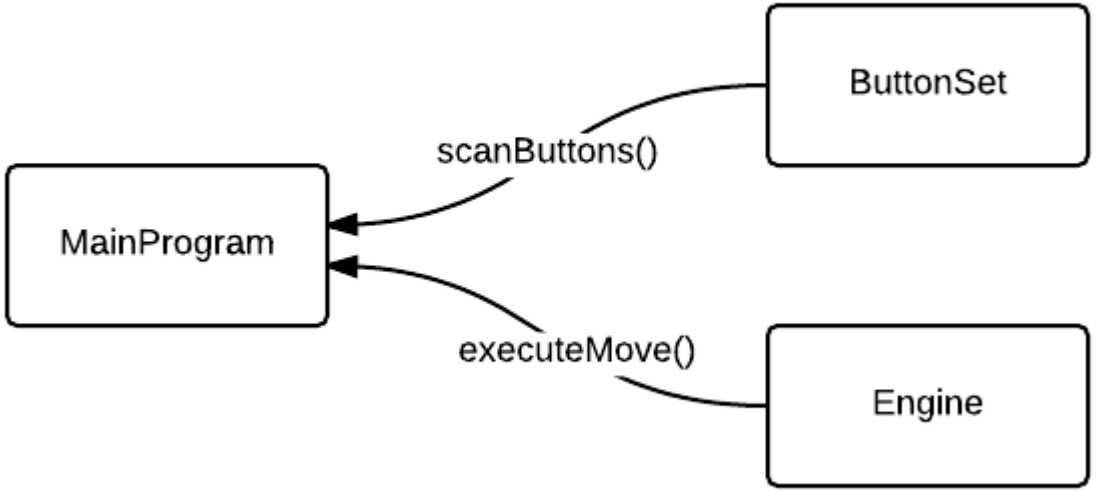
	Duemilanove/ Nano/ Pro/ ProMini	Mega
# of IO	14 + 6 analog (Nano has 14+8)	54 + 16 analog
Serial Pins	0 - RX 1 - TX	0 - RX1 1 - TX1 19 - RX2 18 - TX2 17 - RX3 16 - TX3 15 - RX4 14 - TX4
Ext Interrupts	2 - (Int 0) 3 - (Int 1)	2,3,21,20,19,18 (IRQ0- IRQ5)
PWM pins	5,6 - Timer 0 9,10 - Timer 1 3,11 - Timer 2	0-13
SPI	10 - SS 11 - MOSI 12 - MISO 13 - SCK	53 - SS 51 - MOSI 50 - MISO 52 - SCK
I2C	Analog4 - SDA Analog5 - SCK	20 - SDA 21 - SCL

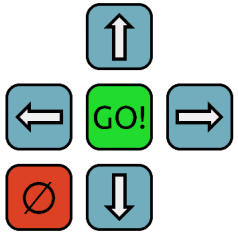
E naceu o primeiro Vacalourabot.ino

- ~ 2 horas de tempo
- Arduino IDE
- Primeiro intento: móvese!!!
- Segundo intento: requisitos OK

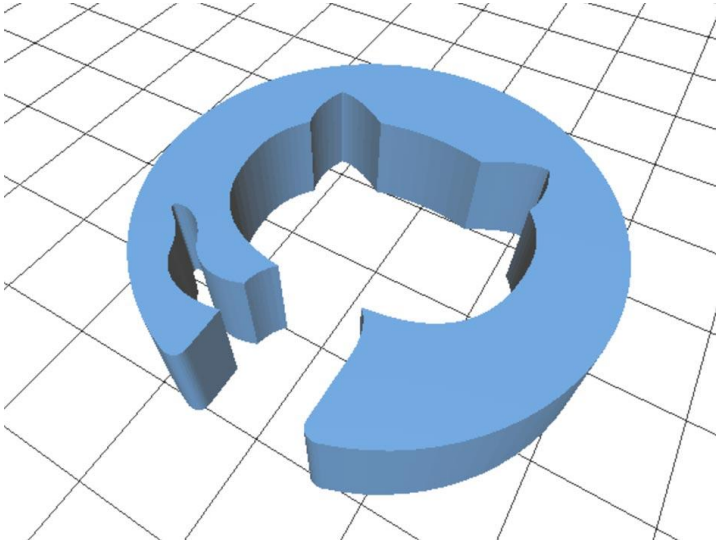


A lóxica principal





Proxecto compartido en GitHub



https://github.com/**brico-labs/Escornabot**

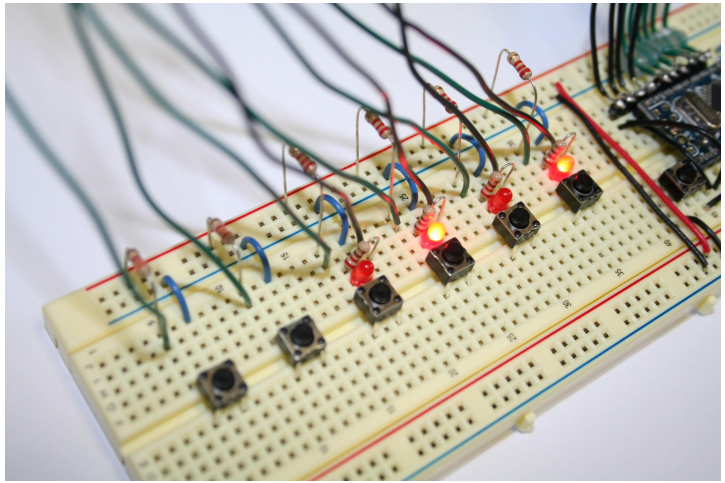
A lectura da botoneira (6 digital input)

```
// configuración dos pins dos botóns
#define PIN_BOTON_ESQUERDA 3
#define PIN_BOTON_DEREITA 5
#define PIN_BOTON_ADIANTE 6
#define PIN_BOTON_ATRAS 7
#define PIN_BOTON_IR 8
#define PIN_BOTON_BORRAR 9
```

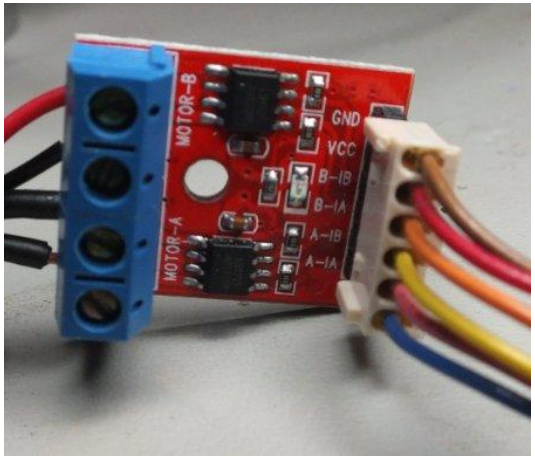
```
// memoria para os estados dos botóns
byte estado_boton_esquerda;
byte estado_boton_dereita;
byte estado_boton_adiante;
byte estado_boton_atras;
byte estado_boton_ir;
byte estado_boton_borrar;
```

```
// configurar botóns
pinMode(PIN_BOTON_ESQUERDA, INPUT);
pinMode(PIN_BOTON_DEREITA, INPUT);
pinMode(PIN_BOTON_ADIANTE, INPUT);
pinMode(PIN_BOTON_ATRAS, INPUT);
pinMode(PIN_BOTON_IR, INPUT);
pinMode(PIN_BOTON_BORRAR, INPUT);
```

```
// leer os botóns
if (digitalRead(PIN_BOTON_ESQUERDA) ^ estado_boton_esquerda)
{
  estado_boton_esquerda = !estado_boton_esquerda;
  if (estado_boton_esquerda == HIGH)
  {
    memorizar(MOVEMENTO_ESQUERDA);
  }
}

if (digitalRead(PIN_BOTON_DEREITA) ^ estado_boton_dereita)
{
```



A activación dos motores H-Bridge




```
// configuración dos pins do driver dos motores
#define PIN_MOTOR_ESQUERDA_A 10
#define PIN_MOTOR_ESQUERDA_B 11
#define PIN_MOTOR_DEREITA_A 12
#define PIN_MOTOR_DEREITA_B 13
```

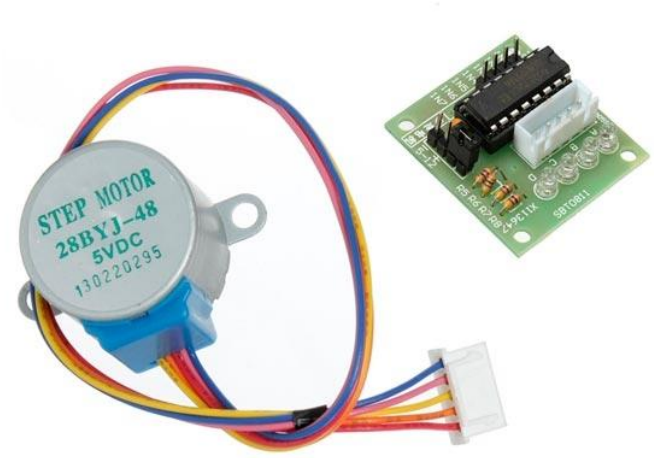
```
// configurar motores
pinMode(PIN_MOTOR_ESQUERDA_A, OUTPUT);
pinMode(PIN_MOTOR_ESQUERDA_B, OUTPUT);
pinMode(PIN_MOTOR_DEREITA_A, OUTPUT);
pinMode(PIN_MOTOR_DEREITA_B, OUTPUT);
```

```
void activarMotor(int motor, SENTIDO direccion)
{
  if (motor == MOTOR_DEREITA)
  {
      // motor dereita
      digitalWrite(PIN_MOTOR_DEREITA_A, direccion ? HIGH : LOW);
      digitalWrite(PIN_MOTOR_DEREITA_B, direccion ? LOW : HIGH);
  }
  else
  {
      // motor esquerda
      digitalWrite(PIN_MOTOR_ESQUERDA_A, direccion ? HIGH : LOW);
      digitalWrite(PIN_MOTOR_ESQUERDA_B, direccion ? LOW : HIGH);
  }
}

void desactivarMotor(int motor)
{
  if (motor == MOTOR_DEREITA)
  {
      // motor dereita
      digitalWrite(PIN_MOTOR_DEREITA_A, LOW);
      digitalWrite(PIN_MOTOR_DEREITA_B, LOW);
  }
  else
  {
      // motor esquerda
      digitalWrite(PIN_MOTOR_ESQUERDA_A, LOW);
      digitalWrite(PIN_MOTOR_ESQUERDA_B, LOW);
  }
}
```

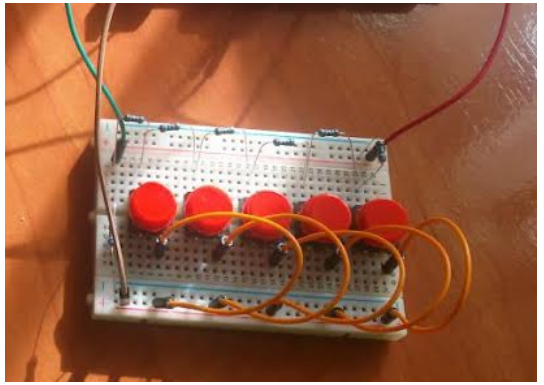
```
void avanzar(byte unidades, SENTIDO direccion)
{
    // activamos os dous motores
    activarMotor(MOTOR_DEREITA, direccion);
    activarMotor(MOTOR_ESQUERDA, direccion);

    delay(unidades * PASO_MILISEGUNDOS);

    // desactivamos os dous motores
    desactivarMotor(MOTOR_DEREITA);
    desactivarMotor(MOTOR_ESQUERDA);
}

void xirar(XIRO xiro, int milisegundos)
{
   if (xiro == XIRO_DEREITA)
   {
      // xiro en sentido reloxo
      activarMotor(MOTOR_ESQUERDA, SENTIDO_ADIANTE);
      activarMotor(MOTOR_DEREITA, SENTIDO_ATRAS);
   }
   else
   {
     // xiro en sentido anti-reloxo
      activarMotor(MOTOR_DEREITA, SENTIDO_ADIANTE);
      activarMotor(MOTOR_ESQUERDA, SENTIDO_ATRAS);
   }

   delay(milisegundos);

   desactivarMotor(MOTOR_ESQUERDA);
   desactivarMotor(MOTOR_DEREITA);
}
```

Segunda iteración do prototipo

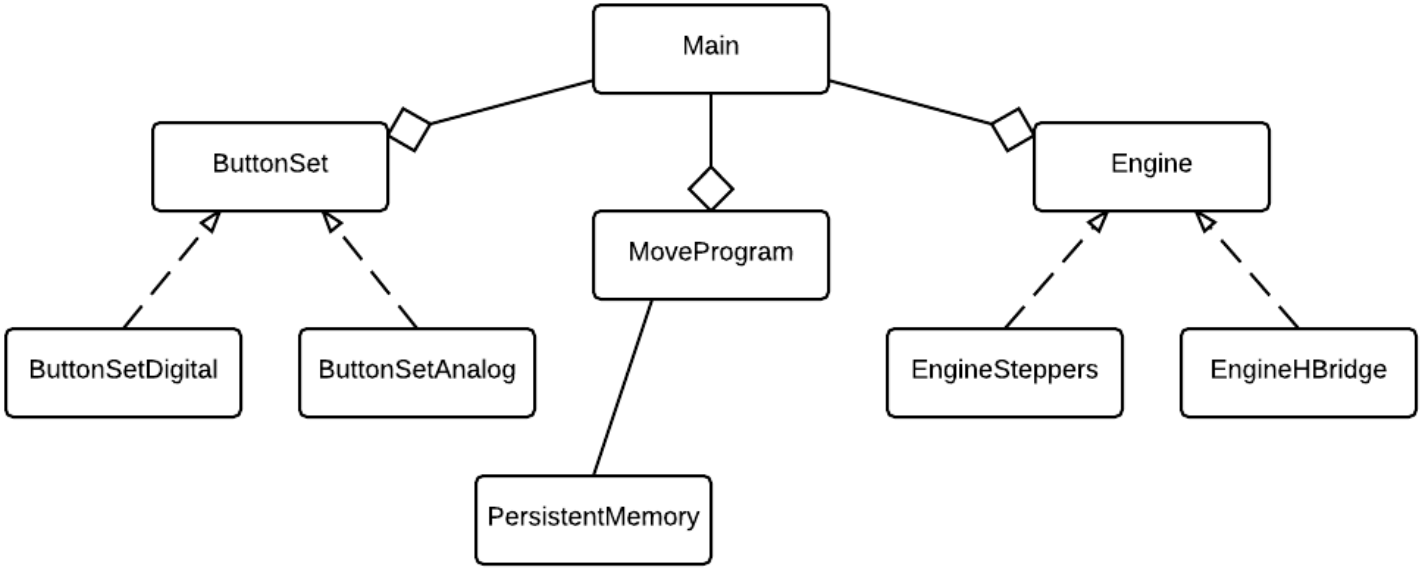
Problema principal: o *Escornabot* non vai recto!



- As 6 entradas dixitais da botoneira → 1 analóxica.



Escalado do desenvolvemento



Refactorización do código

- ButtonSet.h
- ButtonSetAnalog.cpp
- ButtonSetAnalog.h
- ButtonSetDigital.cpp
- ButtonSetDigital.h
- Configuration.h
- Engine.cpp
- Engine.h
- EngineHBridge.cpp
- EngineHBridge.h
- EngineSteppers.cpp
- EngineSteppers.h
- Escornabot.h
- Escornabot.ino
- LICENSE.txt
- MoveProgram.cpp
- MoveProgram.h
- PersistentMemory.cpp
- PersistentMemory.h

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

// engine to use
#define ENGINE_TYPE_STEPPERS
//#define ENGINE_TYPE_HBRIDGE

// button set to use
//#define BUTTONS_DIGITAL
#define BUTTONS_ANALOG

// store configuration and program within internal EEPROM
#define USE_PERSISTENT_MEMORY true

// memory capacity for program movements
#define MOVE_LIMIT 100

// point of view set when Vacalourabot is started
#define POV_INITIAL     POV_VACALOURA
```

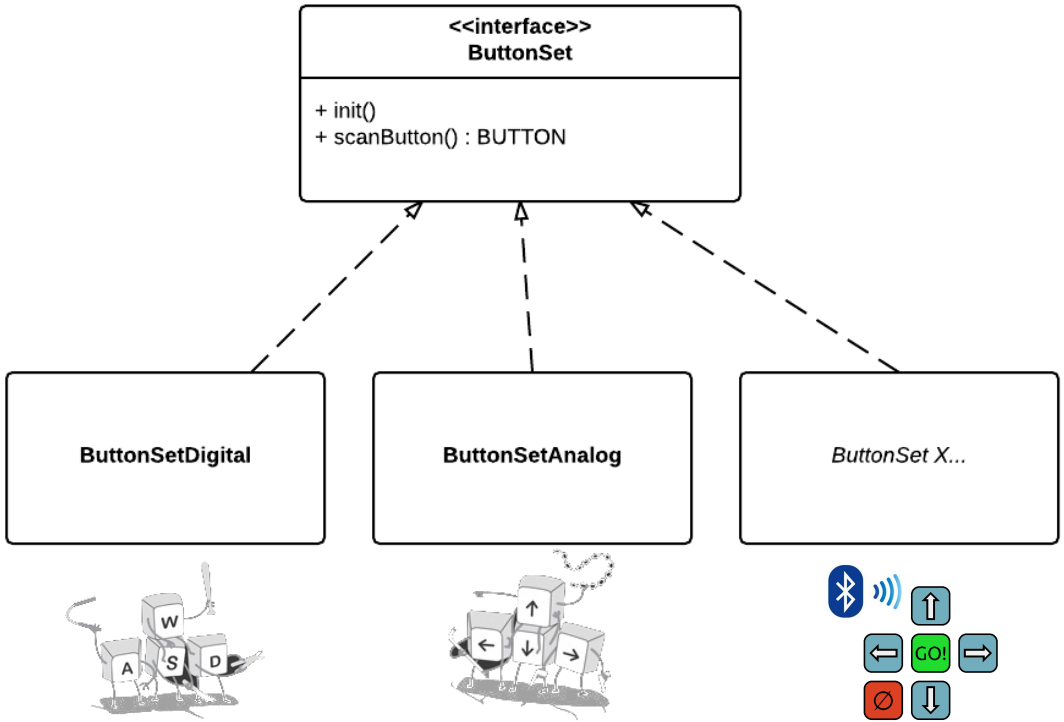
Compilación condicional

```
#if defined(BUTTONS_DIGITAL)

    #include "ButtonSetDigital.h"
    static const ButtonSetDigital::Config BS_CONFIG = {
        pin_button_up: BS_DIGITAL_UP,
        pin_button_right: BS_DIGITAL_RIGHT,
        pin_button_down: BS_DIGITAL_DOWN,
        pin_button_left: BS_DIGITAL_LEFT,
        pin_button_go: BS_DIGITAL_GO,
        pin_button_reset: BS_DIGITAL_RESET,
    };
    static ButtonSetDigital BUTTONS_INSTANCE (&BS_CONFIG);

#elif defined(BUTTONS_ANALOG)

    #include "ButtonSetAnalog.h"
    static const ButtonSetAnalog::Config BS_CONFIG = {
        pin_button_set: BS_ANALOG_PIN,
        value_button_up: BS_ANALOG_VALUE_UP,
        value_button_right: BS_ANALOG_VALUE_RIGHT,
        value_button_down: BS_ANALOG_VALUE_DOWN,
        value_button_left: BS_ANALOG_VALUE_LEFT,
        value_button_go: BS_ANALOG_VALUE_GO,
        value_button_reset: BS_ANALOG_VALUE_RESET,
    };
    static ButtonSetAnalog BUTTONS_INSTANCE (&BS_CONFIG);

#endif // Button set
```

```
#ifdef BUTTONS_ANALOG

// Button set pin setup (analog input)
#define BS_ANALOG_PIN A0

// input values for each key pressed
#define BS_ANALOG_VALUE_UP 471
#define BS_ANALOG_VALUE_RIGHT 299
#define BS_ANALOG_VALUE_DOWN 211
#define BS_ANALOG_VALUE_LEFT 118
#define BS_ANALOG_VALUE_GO 158
#define BS_ANALOG_VALUE_RESET 82

#endif // BUTTONS_ANALOG
```

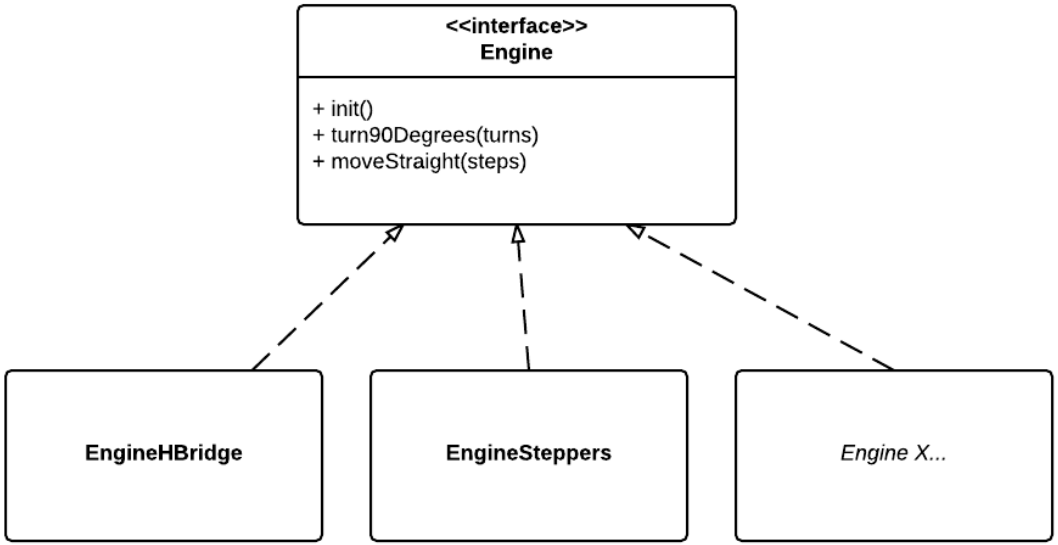
```
// EEPROM as persistent memory
#if USE_PERSISTENT_MEMORY
    #include "PersistentMemory.h"
#endif
```

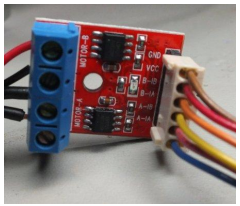
Interface para a botoneira

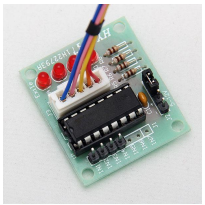



```
class ButtonSet
{
public:

        enum
        {
                BUTTON_NONE,
                BUTTON_UP,
                BUTTON_RIGHT,
                BUTTON_DOWN,
                BUTTON_LEFT,
                BUTTON_GO,
                BUTTON_RESET,
        };
        typedef uint8_t BUTTON;

        /**
         * Does the hardware initialization.
         */
        virtual void init() = 0;

        /**
         * Scans the button input to test if anyone is pressed.
         * @return The button being pressed.
         */
        virtual BUTTON scanButtons() = 0;

};
```

Interface para a maquinaria

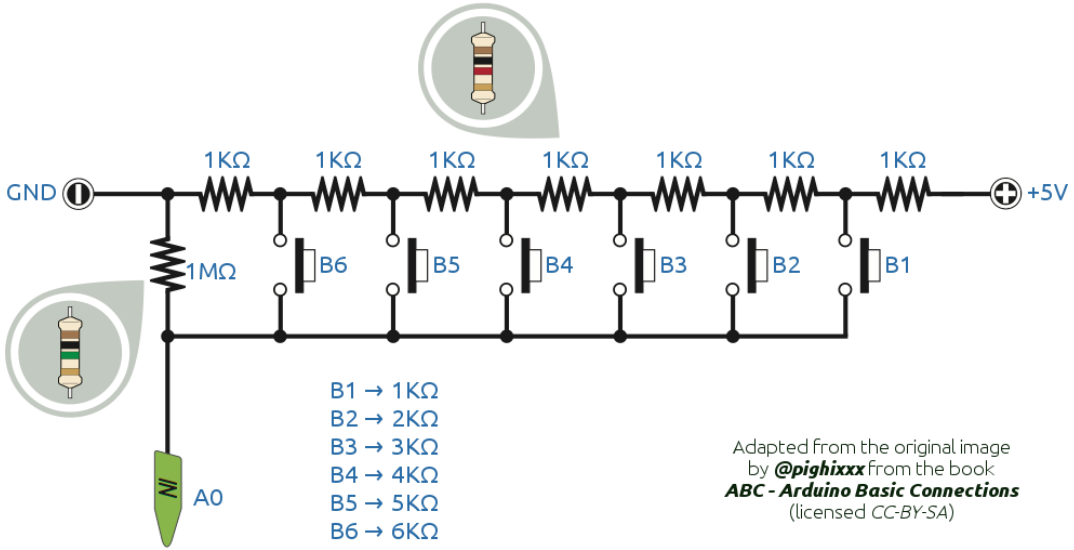






```
class Engine
{
public:

        /**
         * Does the hardware initialization.
         */
        virtual void init() = 0;

        /**
         * Turns left or right in 90 degrees angles (from Vacaloura's POV).
         * @param times Amount of right angles to turn. Positive is clockwise,
         *     negative is counter-clockwise.
         */
        virtual void turn90Degrees(int8_t times) = 0;

        /**
         * Moves forward or backward direction (from Vacaloura's POV).
         * @param units Amount of units to move. Positive is forwards, negative
         *     is backwards.
         */
        virtual void moveStraight(int8_t units) = 0;
```

A botoneira analóxica (1 analog input)



Adapted from the original image by ***@pighixxx*** from the book ***ABC - Arduino Basic Connections*** (licensed *CC-BY-SA*)

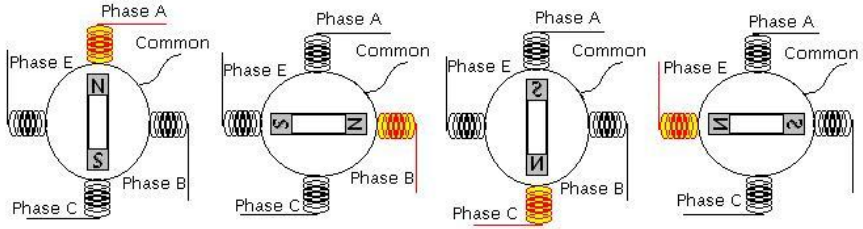
```
ButtonSet::BUTTON ButtonSetAnalog::scanButtons()
{
    delay(200);
    int16_t value = analogRead(_config->pin_button_set);
    int16_t diff, minor_diff;

    minor_diff = value;
    BUTTON button = BUTTON_NONE;

    diff = abs(value - _config->value_button_up);
    if (diff < minor_diff)
    {
        minor_diff = diff;
        button = BUTTON_UP;
    }

    diff = abs(value - _config->value_button_right);
    if (diff < minor_diff)
    {
```

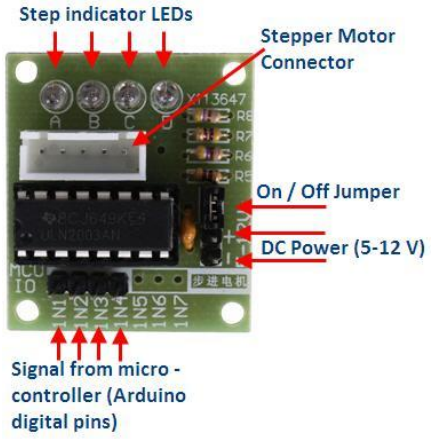
...

```
        // return button only when it changes
        if (button != _last_button)
        {
            _last_button = button;
            return button;
        }

        return BUTTON_NONE;
}
```

A maquinaria EngineSteppers

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






```
void EngineSteppers::_motorStepRight(uint8_t pattern)
{
    digitalWrite(_config->motor_right_in1, bitRead(pattern, 0));
    digitalWrite(_config->motor_right_in2, bitRead(pattern, 1));
    digitalWrite(_config->motor_right_in3, bitRead(pattern, 2));
    digitalWrite(_config->motor_right_in4, bitRead(pattern, 3));
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

void EngineSteppers::_motorStepLeft(uint8_t pattern)
{
    digitalWrite(_config->motor_left_in1, bitRead(pattern, 0));
    digitalWrite(_config->motor_left_in2, bitRead(pattern, 1));
    digitalWrite(_config->motor_left_in3, bitRead(pattern, 2));
    digitalWrite(_config->motor_left_in4, bitRead(pattern, 3));
}
```

```
void EngineSteppers::_motorsOn(int16_t steps_left, int16_t steps_right)
{
    int8_t delta_left = (steps_left > 0 ? 1 : -1);
    int8_t delta_right = (steps_right > 0 ? 1 : -1);

    bool end = false;
    while (!end)
    {
        end = true;

        if (steps_left != 0)
        {
            end = false;
            _motorStepLeft(step_pattern[_pattern_index_left]);
            _pattern_index_left += delta_left + 8;
            _pattern_index_left %= 8;
            steps_left -= delta_left;
        }

        if (steps_right != 0)
        {
            end = false;
            _motorStepRight(step_pattern[_pattern_index_right]);
            _pattern_index_right += delta_right + 8;
            _pattern_index_right %= 8;
            steps_right -= delta_right;
        }

        delayMicroseconds(1000000 / _config->steps_per_second);
    }

    _motorStepLeft(0);
    _motorStepRight(0);
}
```

O futuro do Escornabot.ino

Inmediato:

- Obxeto ***BluetoothInterface***
- Nova interface ***StatusIndicator*** + obxeto global ***StatusIndicatorManager***

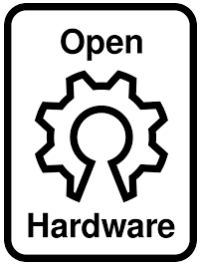
Brainstorming:

- Emisión de sons e leds de lucería
- Detección de liñas (diodos led)
- *Escornabot* bailarín e interacción entre *Escornabots*

Máis alá da implementación

O **bot** programable **libre** e **aberto**.

Un **recurso para aprender** e compartir coñecemento.



Grazas! Cuestións?

Ou marchamos a outra música...