Desarrollo de un Prototipo Mecatrónico

Presentación de la Asignatura
(parte de programación y control)
Curso 2023-2024

Desarrollo de un prototipo mecatrónico

- Profesorado (Area de Ingeniería de Sistemas y Automática)
  - Ignacio Alvarez: programación (ialvarez@isa.uniovi.es)

Implementación de Sistemas de Control

- Clases presenciales: Miércoles de 09:00 a 13:00

Primer Curso - Segundo Semestre

Calendario Base *(si es factible)*

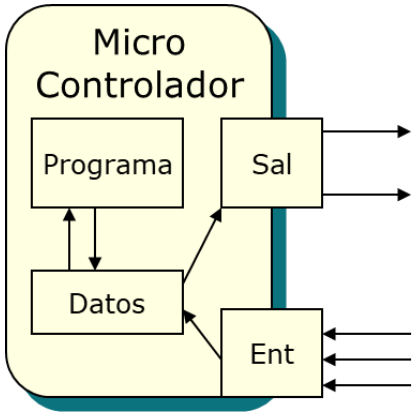
	LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES
09h00-10h00	CTRL	PROT	P-MEC	SINT / ALIM	PROT / CTRL
10h00-11h00	CTRL	PROT	P-MEC	SINT / ALIM	PROT / CTRL
11h00-12h00	ALIM	SINT	P-MEC	ALIM / SINT	CTRL / PROT
12h00-13h00	ALIM	SINT	P-MEC	ALIM / SINT	CTRL / PROT
13h00-14h00					
14h00-15h00			FR		
15h00-16h00	FR / AL		FR / AL		CONF II
16h00-17h00	FR / AL		AL		CONF II
17h00-18h00					CONF II

Las asignaturas de idioma y de 'Conferencias y Seminarios' pueden modificar su horario antes del inicio de las clases.

Objetivo de la asignatura

- Desarrollar un sistema mecatrónico tipo robot, con las características siguientes :
  - Desarrollo de software basado en micro-controlador (Arduino, ESP32), utilizando C++ y librerías
  - Selección de dispositivos (microcontrolador, drivers de motores, etc.)
  - Aplicación de conocimientos adicionales: visión artificial, comunicaciones, etc.

SISTEMAS DE CONTROL SENCILLOS BASADOS EN MICRO-CONTROLADOR



- 1 solo programa gestiona todo el control
- Toda la “inteligencia” en un solo equipo
- Interfaz usuario sencillo
- Comunicaciones básicas
- Opciones limitadas para bb.dd. , interfaz gráfico, comunicaciones red, etc.

MIM EU4M

1er cuatrimestre →

- Se desarrolla toda la electrónica
- Programación en C
- No se usan apenas librerías ni componentes existentes
- Proyecto guiado por los profesores

2º cuatrimestre →

- Se utilizan componentes y librerías de mercado
- Programación en C++
- Proyecto "dirigido" por los alumnos
- Adición de conocimientos "extra" (visión artificial, comu. básica, etc.)

3er cuatrimestre →

- Programación en C++ orientada a eventos (Qt)
- Programación de microprocesador bajo S.O.
- Comunicaciones
- Interfaz gráfico de usuario (GUI)

SISTEMAS DE CONTROL SENCILLOS BASADOS EN MICRO-CONTROLADOR

Microcontroladores:

- Microchip PIC
- Microchip (Atmel) AVR
- ARM
- SMT32
- Espressif ESP32

…

- Integran CPU, memoria, E/S, etc.
- No necesitan casi nada más para funcionar
- No usan Sistema Operativo
- Normalmente son mono-procesador / mono-tarea

Tarjetas y sistemas basados en micro-controlador: (muy baratos, entre 10 y 60€)

- Arduino
- ESP-xxxx
- M5-Stack
- Raspberrry pico
- STMicroelectronic
- Adafruit
- …

SISTEMAS DE CONTROL COMPLEJOS BASADOS EN MICRO-PROCESADOR

Microprocesadores:

- ARM
- Intel Core
- AMD

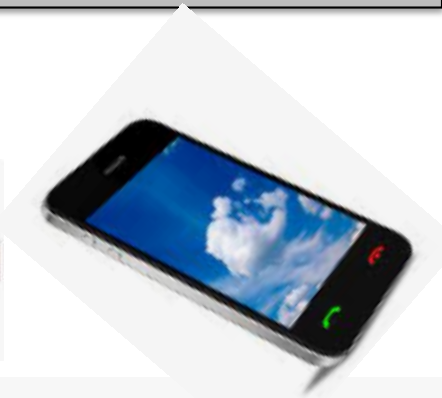
…

- Integran CPU, ALU y memoria caché
- Necesitan muchos chips adicionales y periféricos para funcionar
- Usan Sistema Operativo para la gestión de los programas
- Normalmente son multi-procesador / multi-tarea

Tarjetas y sistemas basados en micro-procesador

- Raspberry Pi
- Odroid
- Beaglebone
- Smart-phone
- PC embebido
- PC portátil
- PC sobremesa
- PC industrial
- PC panel

…



SISTEMAS DE CONTROL SENCILLOS BASADOS EN MICRO-CONTROLADOR

Tarjetas y sistemas basados en micro-controlador:

Electrónica y E/S integrada válida para muchas aplicaciones:
- Entrada/salida digital
- Conversores A/D
- Timers y contadores
- Salidas PWM
- Comunicaciones con periféricos SPI, I2C
- Comunicaciones RS-232, LAN, WiFi
- …

Accesorios electrónicos "complejos":
- Sensores:
  - Temperatura, humedad, presión
  - Acelerómetros
  - Cámaras
  - Micrófonos
  - GPS
- Accionamientos
  - Servomotores
  - Drivers para motores (DC, stepper, BLDC, …)
- Interfaz de usuario
  - Pantallas LCD (texto)
  - Pantallas OLED (texto+gráficos)
  - Tiras de LEDs

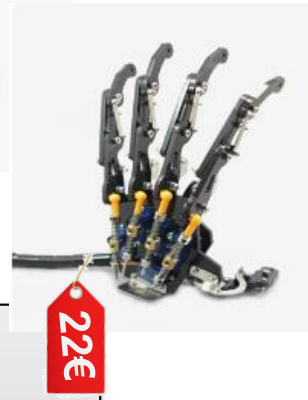
Librerías (software) ya desarrollado:
- Lectura de sensores
- Accionamiento de motores
- Interfaz de usuario
- Comunicaciones
- …

Hardware y software abierto:
- Libre de licencias
- Permite uso comercial

SISTEMAS DE CONTROL SENCILLOS BASADOS EN MICRO-CONTROLADOR

Tarjetas y sistemas basados en micro-controlador:

¡¡¡ SON REALMENTE MUUUUUUUY BARATOS !!!!



Pero

¡¡¡¡ OJO !!!! Muchos han nacido, crecido y proliferado para usos no profesionales (hobbyists)

- Poca calidad, o calidad no contrastada, del hardware
- Especificaciones detalladas difíciles de obtener
- Documentación muy deficiente
- Entorno de desarrollo (IDE) pobre, dificultad para compilar y depurar
- Software de poca calidad, poco probado, y/o muy mal documentado
- Comunidades de aficionados que intercambian código, informaciones, dudas y respuestas

ARDUINO-IDE

Arduino fue la primera plataforma en popularizarse, debido a la sencillez y bajo coste de su hardware y software. Originalmente se creó para la enseñanza.

Arduino IDE es el entorno de programación más usual para Arduino, utiliza como lenguaje de programación C++ (extensión de C para Programación Orientada a Objetos - OOP).

Se han desarrollado numerosas librerías para Arduino, que son sencillas y ampliamente utilizadas, y están basadas en clases C++.

Nuevas plataformas que han surgido tienden a ser compatibles a nivel de IDE y librerías para facilitar la transición de los usuarios.

Pero…

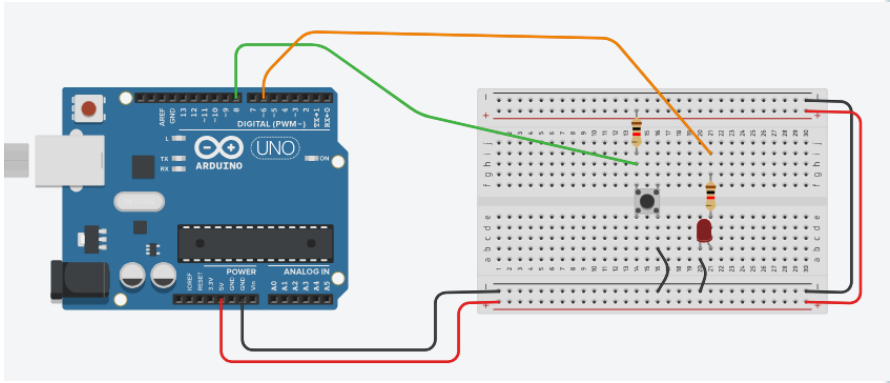
El entorno es de lo más simple y "flojo" del mercado, porque se creó para la sencillez y no para la productividad.
La documentación es escasa y de mala calidad: está pensada para la sencillez, no para la productividad.

ARDUINO

Un primer montaje+programa sencillo para Arduino-Uno.

Objetivo:
- Cada vez que se presiona un pulsador, se disminuye la iluminación de un LED. Si llega a 0, la siguiente pulsación lo devuelve al máximo.

Utilizar el simulador web TinkerCad: https://www.tinkercad.com/circuits



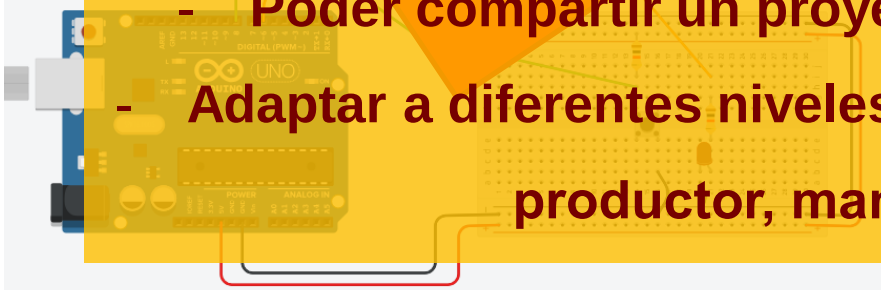
```
// C++ code

bool is_pressed;
int t_on=0;

void setup()  // Se ejecuta una sola vez al inicio
{
  pinMode(8,INPUT);
  pinMode(6,OUTPUT);
  is_pressed=false;
}

void loop() // Se ejecuta repetidamente tras setup()
{
   if (digitalRead(8)==LOW)  {  // Si es LOW, está pulsado
     if (! is_pressed) { // Antes no estaba pulsado: hay un
                         // flanco → pulsación
        is_pressed=true;
        t_on = t_on-1000; // Modificamos tiempo ON
        if (t_on < 0) {
            t_on = 5000; // Si nos pasamos, volvemos al máx
        }
     }
   }
   else
      is_pressed=false; // Si no es LOW, no está pulsado

   if (t_on>0) { // Generar la PWM mediante retardos
      digitalWrite(6,HIGH);
      delayMicroseconds(t_on);
      digitalWrite(6,LOW);
      delayMicroseconds(5000-t_on);
   }
  else
      digitalWrite(6,LOW);

}
```

ARDUINO

Un primer mont… rama sencillo
para Arduino …

Objetivo:
- Cada vez que se presion… pulsador, se disminuye la ilum… de un LED. Si llega a 0, la siguien… pulsación lo devuelve al máximo.

Utilizar el simulador web T…
https://www.tinkercad…



```
// C++ code
int t_on=0;

void setup()  // Se ejecuta u… inicio
  pinMode(8,INPUT);
  pinMode(6,OUTPUT);
  is_pressed=fals…

d loo… epetidamente tras setup()

…=LOW)  {  // Si es LOW, está pulsado
…ed) { // Antes no estaba pulsado: hay un
     // flanco → pulsación
…true;
…000; // Modifiquemos tiempo ON
…  // Si nos pasamos, volvemos al máx
    }
  }
  else
    …ed=false;

  if (t_on>0) { // Generar… …etardos
    digitalWrite(6,HIGH);
    delayMicroseconds(t_on);
    digitalWrite(6,LOW);
    …delayMicroseconds(2000-t_on);
  }
  else
    …delay(2);
}
```

Así aparece en numerosos (la mayoría) de tutoriales de Internet

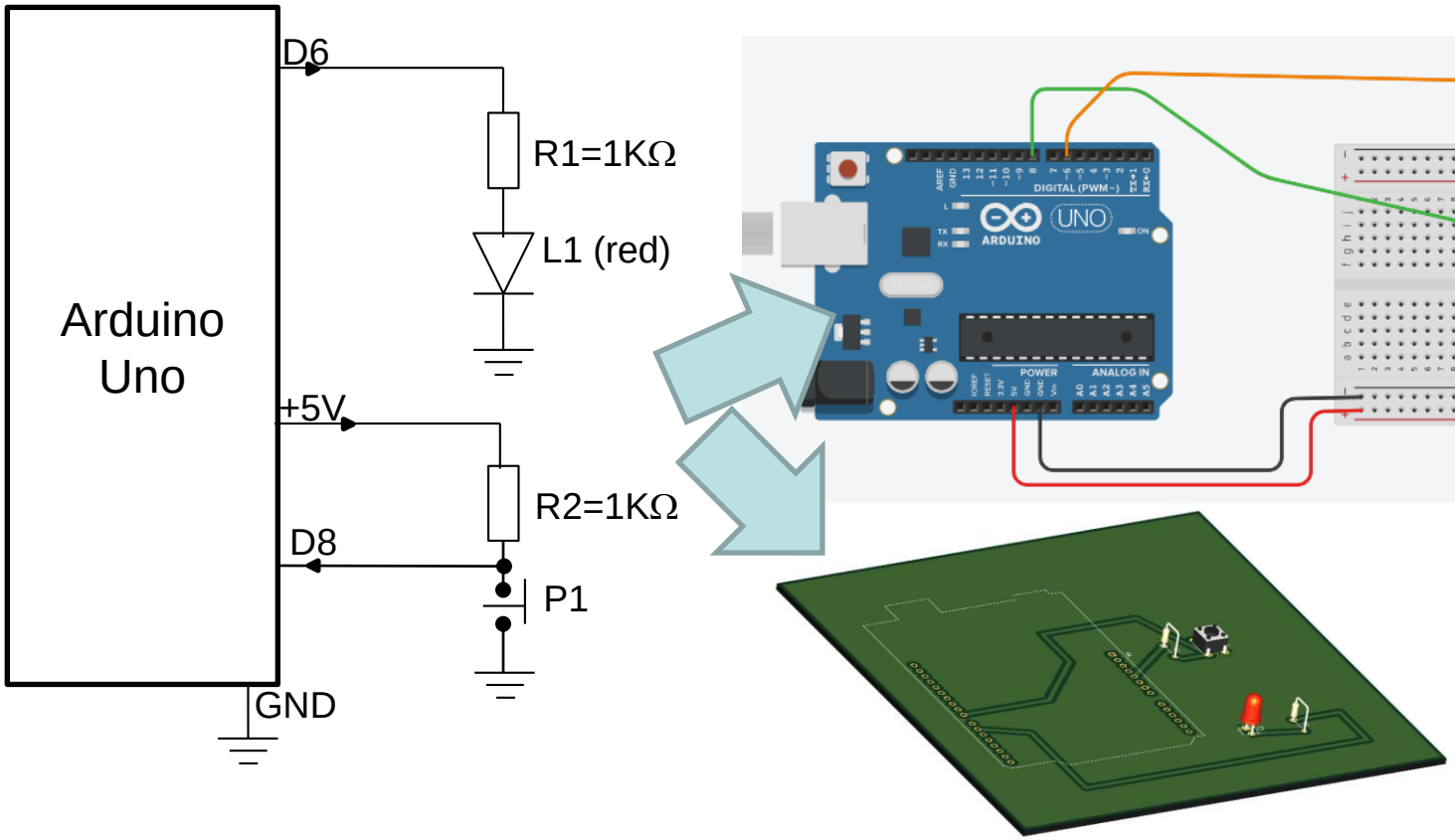
Así trabajan o se les presenta a los aficionados (hobbyists)

Nosotros somos ingenieros, necesitamos:

- **Más calidad**
- **Mejor documentación**
- **Asegurar que va a funcionar según lo especificado**
- **Soluciones reutilizables para múltiples proyectos**
- **Poder cambiar decisiones sin que afecten a todo el proyecto**
- **Poder compartir un proyecto entre varios desarrolladores**
- **Adaptar a diferentes niveles de acceso: desarrollador, usuario, productor, mantenedor, reparador, …**

ARDUINO: A diferencia de los aficionados…

(1) ¡¡¡ Hay que hacer un esquemático del circuito !!! Después se cableará en una tarjeta prototipo, se ensamblará en una PCB, etc., pero en el esquemático es donde se diseña, se puede discutir sobre la solución, se especifican los componentes, se aprecian los problemas, etc.



ARDUINO: A diferencia de los aficionados…

(2) ¡¡¡ Hay que desarrollar y probar componentes por separado !!!

Ejemplo: el código para detectar cualquier pulsador será siempre el mismo, sólo cambia el pin al que está conectado : desarrollamos **una clase** y **probamos con objeto(s)** de esa clase:

Test.ino

```
// C++ code

#include "MyPushButton.h"

MyPushButton _btn;

void setup()
{
   pinMode(8,OUTPUT);
   _btn.begin(6,MyPushButton::INVERSE_LOGIC);
}

void loop()
{
   _btn.Check();
   int counter=_btn.GetEdge(MyPushButton::RISING_EDGE);
   if (counter > 0)  {
       digitalWrite(8,counter % 2);
   }
}
```

MyPushButton.h

```
#ifndef _INC_MYPUSHBUTTON_H
#define _INC_MYPUSHBUTTON_H

class MyPushButton
{
public:
 // Variables, functions and types for external use
  typedef enum {INVERSE_LOGIC,
                DIRECT_LOGIC} logicTypeEnum;
  typedef enum {RISING_EDGE,
                FALLING_EDGE} edgeTypeEnum;

  MyPushButton();
  bool SetPin(int i_pinNumber,logicTypeEnum i_logic);
  void Check();
  bool IsPressed();
  int GetEdge(edgeTypeEnum type);

private:
// variables and functions for internal use only ...
};

#endif // _INC_MYPUSHBUTTON_H
```

ARDUINO: A diferencia de los aficionados…

(2) ¡¡¡ Hay que desarrollar y probar componentes por separado !!!

MyPushButton.h

```
#ifndef _INC_MYPUSHBUTTON_H
#define _INC_MYPUSHBUTTON_H

class MyPushButton
{
public:
  typedef enum {INVERSE_LOGIC,
                DIRECT_LOGIC} logicTypeEnum;
  typedef enum {RISING_EDGE,
                FALLING_EDGE} edgeTypeEnum;

  MyPushButton();
  bool begin(int i_pinNumber,logicTypeEnum i_logic);
  void Check();
  bool IsPressed();
  int GetEdge(edgeTypeEnum type);

private:
  bool state[2];
  int counter[2]={0,0}; // Rising and falling
  int pinNumber;
  logicTypeEnum logic;
  bool GetCurrentState();
};

#endif // _INC_MYPUSHBUTTON_H
```

MyPushButton.cpp

```
#include "MyPushButton.h"
#include "Arduino.h"

MyPushButton::MyPushButton() // Basic initialization
{
   counter[0]=counter[1]=0;
   state[0]=state[1]=false;
   pinNumber=-1;
   logic=DIRECT_LOGIC;
}

bool MyPushButton::begin(i_pinNumber,
                         logicTypeEnum i_logic)
// Full initialization
{
   pinNumber=pin;
   logic=i_logic;
   state[0]=state[1]=GetCurrentState();
   counter[0]=counter[1]=0;
}

void MyPushButton::Check() // Call every loop
{
   state[1]=state[0];
   state[0]=GetCurrentState();
   if (state[0] && !state[1])
      counter[RISING_EDGE]++;
   else if (state[0] && !state[1])
      counter[FALLING_EDGE]++;
}
```

ARDUINO: A diferencia de los aficionados…

(3) Antes de desarrollar, buscar si alguien ha hecho algo similar **y confiable**

Búsqueda en Internet: Arduino button class

https://forum.arduino.cc/t/buttons-an-object-oriented-approach/279724/2

https://pololu.github.io/pushbutton-arduino/class_pushbutton.html

https://github.com/evert-arias/EasyButton

https://arduinogetstarted.com/tutorials/arduino-button-library

https://github.com/JChristensen/JC_Button

PREGUNTA:

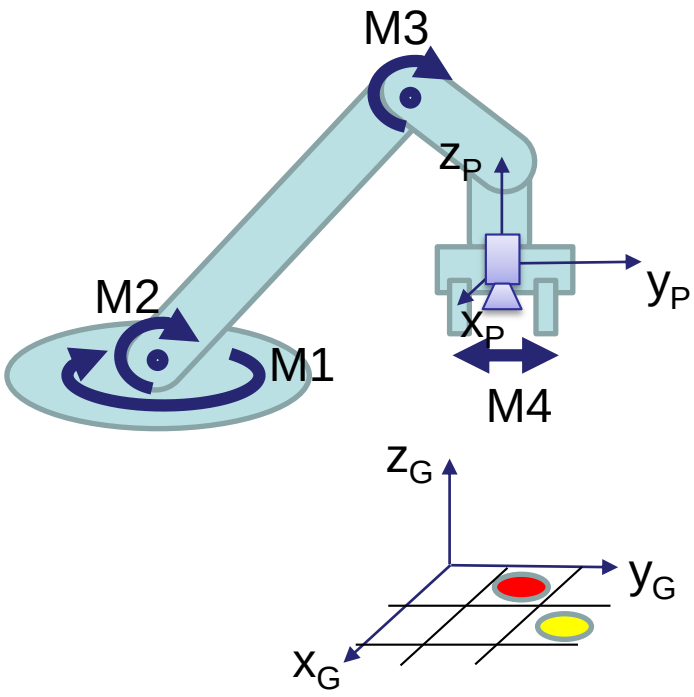
¿Cuál usar? ¿Me sirve? ¿Es robusta? ¿Parece fiable? ¿Está bien documentada?

RESPUESTA:

Difícil sin experiencia. Las "buenas" suelen aparecer en más entradas, pero no siempre. Las más usadas suelen estar mejor documentadas, y aparecen en más preguntas de usuarios (stackoverflow, stackexchange).

ARDUINO: A diferencia de los aficionados…

(4) ¡¡¡ Hay que **desacoplar** en el diseño y desarrollo, utilizando "componentes" genéricos independientes de la tecnología subyacente!!!



- M1 a M4 son motores. Pueden ser diferentes por el diseño electro-mecánico (DC, stepper, servo, AC, BLDC, sin/con controladora, con/sin sensor, diferentes potencias, …), pero **de cara al programa no debería haber diferencias sustanciales**. Quien programa los movimientos querrá hacer algo como:

```
m1.MoveAbs(deg_m1,speed_m1);
m2.MoveAbs(deg_m2,speed_m2);
…
```

¡¡¡ independientemente del tipo de motor !!!

- La cámara sirve para posicionar de forma relativa los elementos reconocibles en $[x_G , y_G , z_G]$ respecto al sistema de coordenadas de la pinza $[x_P , y_P , z_P]$. Quien programa los movimientos quiere hacer algo como:

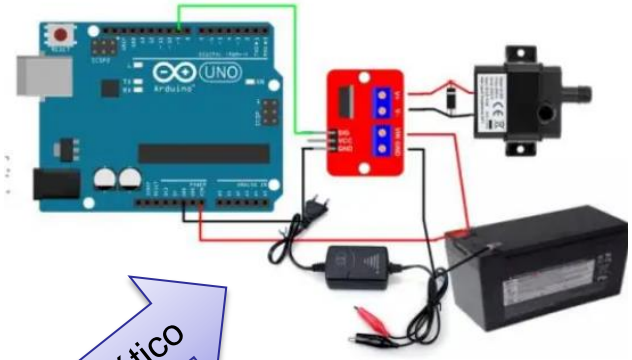
```
pos_rel=camera.DetectaPieza(COLOR_ROJO);
pos_abs=pinza2glob.ConvertirCoordenadas(pos_rel);
robot.CinematicaInversa(pos_rel,&deg_m1,&deg_m2,...);
m1.MoveAbs(deg_m1,speed_m1);
...
```

¡¡¡ independientemente del tipo de cámara, cómo se detectan las piezas, etc. !!!

ARDUINO: A diferencia de los aficionados…

(5) ¡¡¡ Evitar soluciones "de andar por casa" !!!

Ejemplo: activación de una bomba de agua con Arduino, solución "de andar por casa" que se encuentra en Internet:



Sin esquemático

```
1  const int pin = 9;  //Declarar pin D9
2
3  void setup()
4  {
5    pinMode(pin, OUTPUT);  //Define pin 9 como salida
6  }
7
8  void loop()
9  {
10   digitalWrite(pin, HIGH);   // Poner el pin en HIGH (activar)
11   delay(600000);               //Espera 10 min
12   digitalWrite(pin, LOW);    //Apaga la bomba
13   delay(2000);               // Esperará 2 segundos y comenzará ciclo
14 }
```

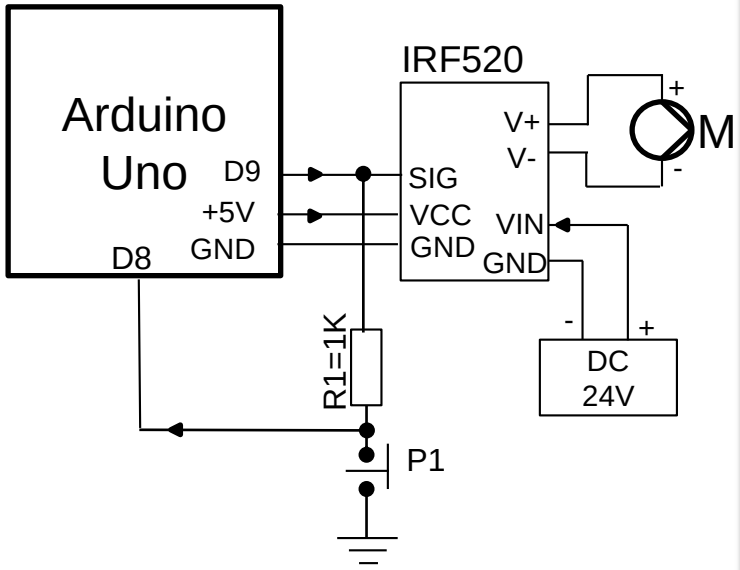
delay() no es lo correcto

Si usamos delay() tenemos la solución "fácil y rápida", pero estaremos 10 minutos sin respuesta del equipo: ¡¡¡ Cualquier otra interacción que queramos añadir no funcionará !!!!

ARDUINO: A diferencia de los aficionados…

(5) ¡¡¡ Evitar soluciones "de andar por casa" !!!

Ejemplo: activación de una bomba de agua con Arduino, solución "de ingeniero":




```
#include "MyWaterPump.h"
#include "MyPushButton.h"

#define PIN_OUT_WATERPUMP           9
#define PIN_IN_PULS                 8
#define TIME_OFF_ms                 (2*1000)
#define TIME_ON_ms                  (60*1000-TIME_ON_MS)

MyWaterPump _waterPump;
MyPushButton _p1;

void setup()
{
   _p1.begin(PIN_IN_PULS,MyPushButton::INVERSE_LOGIC);
   _waterPump.begin(PIN_OUT_WATER_PUMP, TIME_OFF_ms, TIME_ON_ms);
}

void loop()
{
   if (_p1.Check() )
   {
      if (_p1.IsToggledOn() )
         _waterPump.start();
      if (_p1.IsToggledOff() )
         _waterPump.stop();
   }
   _waterPump.Run1Step();
}
```

ARDUINO: A diferencia de los aficionados…

(6) Seguir normativas

Comprobar las normativas de aplicación, específicamente:

- Nombres y símbolos de elementos en planos
- Normativa de seguridad si aplicable
- Marcado CE
- REBT si aplicable
- etc.

Ejemplo planos eléctricos/electrónicos:
http://www.uco.es/electrotecnia-etsiam/simbologia/Normalizacion_simbologia_electrica.pdf

ARDUINO: A diferencia de los aficionados…

(7) Diseñar y documentar correctamente

El desarrollador del prototipo no es el único involucrado en el ciclo de vida del proyecto. Hay múltiples agentes:

- Desarrollador líder
- Diseñadores de algunos componentes
- Usuario final
- Fabricante
- Montador
- Mantenedor/reparador
- Actualizaciones

¡ Cada agente requiere un acceso diferente, tanto al sistema como a la documentación !

Ejemplos:
- Prever apagado rápido (seta emergencia) para usuario final
- Prever modo manual para mantenimiento/reparación
- Manuales/documentos de diseño, usuario, producción, mantenimiento, etc.