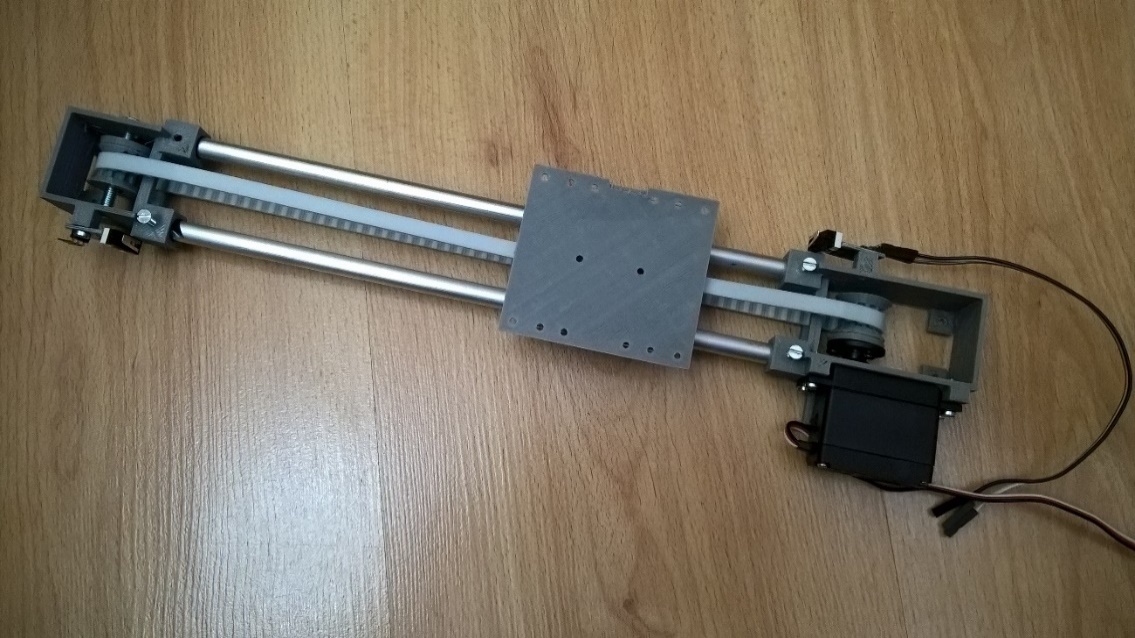
|  |
| --- |
| Software para Robots Jordán Pascual : pascualjordan@uniovi.es |

|  |  |
| --- | --- |
| 4 | Actuador lineal |

# Introducción Durante la sesión de prácticas programaremos el comportamiento de los actuadores sensores de un actuador lineal. Una vez comprendido el funcionamiento de los componentes realizaremos varias actividades relacionadas con la programación de tareas en un sistema manipulador basado en actuadores lineales.

# Actuador lineal

****

## Servomotor de rotación continua

En este ejemplo vamos a realizar un test de funcionalidad a un servomotor de rotación continua.

Aunque utiliza la misma librería el mecanismo de funcionamiento no es el mismo que el de los servos de rotación no continua.

|  |  |
| --- | --- |
|  | **Servomotor 360 ° SM-S4303R:** este tipo de servomotor permite realizar movimientos de rotación continua en sentido horario y anti horario a diferentes velocidades.  La primera vez que se utiliza requiere calibración. Para calibrar un servomotor se le envía la señal de detener motor **servo.write(90)**. Si al ejecutar esta instrucción el servomotor continúa moviéndose debemos girar el tornillo situado en la parte delantera del servomotor hasta que esté completamente detenido.  En ocasiones la calibración también se puede perder por el uso. |

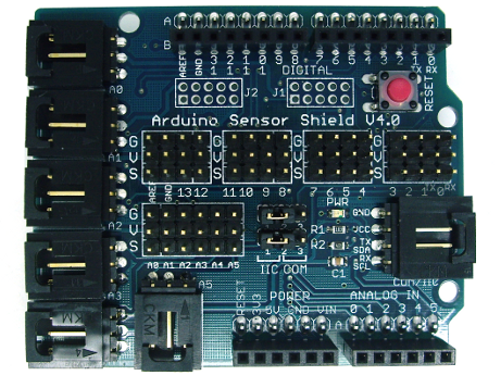
**Alimentación Eléctrica**

La alimentación eléctrica del USB a la placa puede no ser suficiente en muchos casos para alimentar uno o varios servomotores grandes. Cuando la alimentación eléctrica no es suficiente el servomotor comienza a funcionar de una forma “aleatoria”, sin hacer caso realmente a los comandos que le enviamos. Para evitar estos problemas debemos conectar siempre nuestra placa a una fuente de alimentación adicional durante esta práctica utilizaremos el cargador de 9V de Arduino, en futuras practicas utilizaremos baterías recargables.



**Construcción del circuito:**

1. Conectamos el servomotor a: **GND (cable negro), 5V (cable Rojo) y un pin digital (cable blanco).**
2. La forma de conexión más sencilla es utilizar una de las columnas de la placa de sensores:



**Programación de Arduino:**

En primer lugar importamos la librería que Arduino nos ofrece para controlar el servomotor.

Después de importarla observaremos que se ha incluido un nuevo #include en nuestro programa.

#include <Servo.h>

1. Creamos la variable global de tipo Servo.

Servo servo1**;**

1. Inicializamos el puerto para registrar la salida en el método setup(). asociamos el pin 8 al servo (no hace falta especificar que el pin 8 será una salida digital).

Serial**.**begin**(**9600**);**

servo1**.**attach**(**8**);**

1. En el método loop(). Indicamos que el servo debe moverse en sentido anti horario y realizamos una pausa en la ejecución del programa durante 5 segundos (El servomotor sigue manteniendo el sentido de giro indicado).

Serial**.**println**(**"Enviar: giro en sentido 1"**);**

servo**.**write**(**0**);**

delay**(**2000**);**

1. A continuación indicamos que debe detenerse y realizamos una pausa en la ejecución del programa durante 5 segundos (El servomotor sigue manteniendo el sentido de giro indicado).

Serial**.**println**(**"Enviar: detener"**);**

servo**.**write**(**90**);**

delay**(**2000**);**

1. Finalmente indicamos que debe girar en sentido horario y realizamos una pausa en la ejecución del programa durante 5 segundos (El servomotor sigue manteniendo el sentido de giro indicado).

Serial**.**println**(**"Enviar: giro en sentido 2"**);**

servo**.**write**(**180**);**

delay**(**2000**);**

|  |
| --- |
| #include <Servo.h>  Servo servo**;**  void setup**()**  **{**  Serial**.**begin**(**9600**);**  servo**.**attach**(**8**);**  **}**  void loop**()**  **{**  // Posición giro anti horario (0) esperar 5 s  Serial**.**println**(**"Enviar: giro en sentido 1"**);**  servo**.**write**(**0**);**  delay**(**2000**);**    // Posición detener (90) esperar 5 s  Serial**.**println**(**"Enviar: detener"**);**  servo**.**write**(**90**);**  delay**(**2000**);**    // Posición giro horario (180) esperar 5 s  Serial**.**println**(**"Enviar: giro en sentido 2"**);**  servo**.**write**(**180**);**  delay**(**2000**);**  // Otros valores entre (0-84) y (93-180) hacen que gire más despacio    **}** |

## Joystick

Este tipo de Joysticks funcionan como mecanismo de entrada analógico (direcciones del joystick) y digital (botón pulsador).

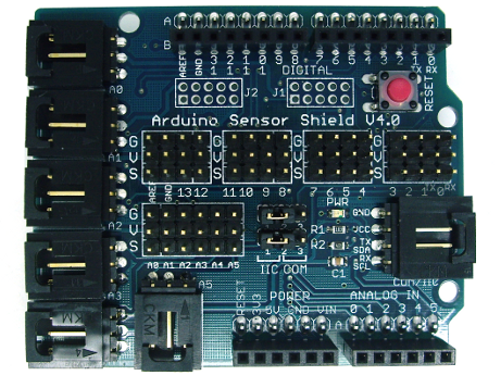
Permite registrar diferentes posiciones en el eje Y e X y pulsaciones sobre el botón central.

En el siguiente ejemplo realizaremos una prueba de funcionalidad.

|  |  |
| --- | --- |
|  | **Joystick analógico:** permite obtener la medición analógica de los ejes XY de forma analógica. Los valores se obtienen en el rango 0 -1023.  Registra la pulsación del botón central mediante una entrada digital, 1 cuando el botón no está pulsado y 0 cuando lo está. |

**Construcción del circuito :**

1. Conectamos el servomotor a: **GND (a GND), +5V (a 5V) , VRx (a una entrada analógica) VRy (A una entrada analógica), SW (a un pin digital).**
2. La forma de conexión en la placa de sensores resulta muy poco directa.



**Programación de Arduino:**

1. Registramos los identificadores de los pines a los que vamos a conectar el sensor.

#define boton\_pin 8 // pin digital para el botón

#define X\_pin A0 // pin analógico para leer eje X

#define Y\_pin A1 // pin analógico para leer eje Y

1. En el método setup(). registramos el pin digital como : **INPUT\_PULLUP**, es un tipo especial de INPUT

[https://www.arduino.cc/en/Tutorial/InputPullupSerial](https://www.arduino.cc/en/Tutorial/InputPullupSerial%20)

Esta función agrega una resistencia interna en pull-up de 50k-ohm, simplifica la utilización de botones y pulsadores, de esta forma podemos utilizar directamente el pulsador que incorpora el Joystick.

pinMode**(**boton\_pin**,** INPUT\_PULLUP**);**

Serial**.**begin**(**9600**);**

|  |
| --- |
| Anteriormente colocábamos una resistencia Pull Up física en el circuito para poder leer un pulsador, con INPUT\_PULLUP la resistencia física ya no es necesaria.    [https://www.arduino.cc/en/Tutorial/InputPullupSerial](https://www.arduino.cc/en/Tutorial/InputPullupSerial%20) |

1. En el método loop() leemos las entradas analógicas y digitales y las mostramos por el Serial.

Serial**.**println**(**"Boton pulsado: "**+**String**(**digitalRead**(**boton\_pin**)));**

int valorX **=** analogRead**(**X\_pin**);**

Serial**.**println**(**"X: "**+**String**(**valorX**));**

int valorY **=** analogRead**(**Y\_pin**);**

Serial**.**println**(**"Y: "**+**String**(**valorY**));**

delay**(**100**);**

|  |
| --- |
| #define boton\_pin 8 // pin digital para el botón  #define X\_pin A0 // pin analógico para leer eje X  #define Y\_pin A1 // pin analógico para leer eje Y  void setup**()** **{**  // inicializar pin 2 (Entrada)  pinMode**(**boton\_pin**,** INPUT\_PULLUP**);**  Serial**.**begin**(**9600**);**    **}**    void loop**()** **{**  Serial**.**println**(**"Boton pulsado: "**+**String**(**digitalRead**(**boton\_pin**)));**  int valorX **=** analogRead**(**X\_pin**);**  Serial**.**println**(**"X: "**+**String**(**valorX**));**  int valorY **=** analogRead**(**Y\_pin**);**  Serial**.**println**(**"Y: "**+**String**(**valorY**));**  delay**(**100**);**  **}** |

## Sensor de colisión / choque

El sensor de colisión funciona de la misma forma que un botón, al detectar una colisión envía una señal. Estos sensores se utilizan para detectar colisiones entre componentes del propio sistema o con elementos exteriores.

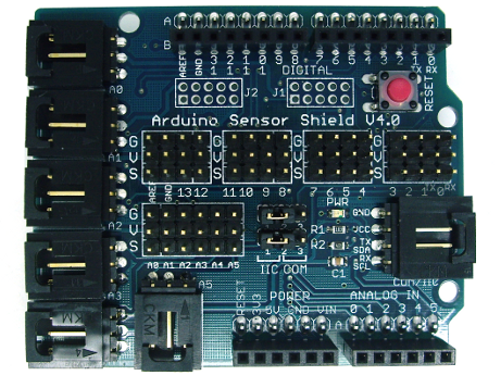
Ejemplo: muchas impresoras 3D utilizan estos sensores para detectar los "topes" de su recorrido, muchos robots móviles los utilizan para detectar obstáculos.

|  |  |  |
| --- | --- | --- |
|  | |  |
| http://www.iearobotics.com/wiki/images/thumb/7/70/Endstop-clamp-01.JPG/300px-Endstop-clamp-01.JPG |  | |

|  |  |
| --- | --- |
|  | **Sensor de colisión:** permite alimentaciones entre 3V - 12V.  Al detectar una colisión se enciende el led integrado y registra una señal digital de 0 (De lo contrario cuando no está pulsado envía una señal digital 1).  Suele ser fijado a las superficies utilizando un tornillo.  La superficie metálica que detecta las colisiones puede ser ampliada con otras carcasas o componentes adicionales. |

**Construcción del circuito :**

1. Conectamos el servomotor a: **GND (a GND), VCC (a 5V) , OUT (a un pin digital).**
2. Conexión en la placa de sensores:



**Programación de Arduino:**

1. En el método setup(). registramos el pin digital como : **INPUT**

Serial**.**begin**(**9600**);**

pinMode**(**6**,** INPUT**);**

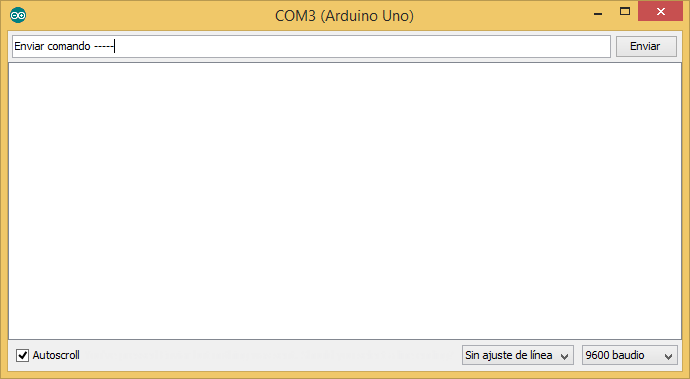
1. En el método loop() leemos las entrada digital y la mostramos por el Serial.

Serial**.**println**(**"Sensor Crash: "**+**String**(**digitalRead**(**6**)));**

|  |
| --- |
| void setup**()** **{**  Serial**.**begin**(**9600**);**  pinMode**(**6**,** INPUT**);**  **}**  void loop**()** **{**  Serial**.**println**(**"Sensor Crash: "**+**String**(**digitalRead**(**6**)));**  **}** |

## Envío de datos por el puerto COM

Desde el propio monitor de serie del IDE de podemos enviar desde el PC al Arduino utilizando el puerto COM.



Para leer los datos enviados desde Arduino debemos utilizar la función, **Serial.available()** para comprobar si hay datos disponibles para la lectura (y cuantos hay) , y **Serial.Read()** para leer cada uno de esos datos.

**Serial. available()** <https://www.arduino.cc/en/Serial/Available>

**Serial.read()** <https://www.arduino.cc/en/Serial/Read%20>

|  |
| --- |
| char datoLeido**;**  void setup**(){**  Serial**.**begin**(**9600**);**  Serial**.**println**(**"Setup()"**);**  **}**  void loop**()**  **{**  **while(**Serial**.**available**()** **>** 0**)** // Nº de datos disponibles para leer.  **{**  datoLeido **=** Serial**.**read**();** // Leer  Serial**.**println**(**datoLeido**);**  **}**  **}** |

En el ejemplo anterior los caracteres se procesan uno a uno, si queremos procesar más de un carácter de forma simultánea por ejemplo "11", "start" podemos utilizar un array de chars[] o utilizar las funciones **Serial.readString() , Serial.parseInt(), Serial.parseFloat()**, etc.

Ejemplo función **Serial.readString() y Serial.parseInt().**

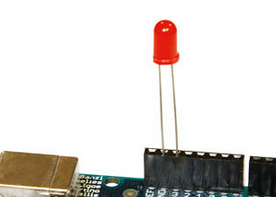
|  |
| --- |
| String cadena**;**  int entero**;**    void setup**()** **{**  Serial**.**begin**(**9600**);**  **}**    void loop**()** **{**    Serial**.**println**(**"Nombre: "**);**  **while** **(**Serial**.**available**()** **==** 0**)** **{** **}** // Esperar mientras no haya nada    cadena **=** Serial**.**readString**();**  Serial**.**println**(**"Es "**+** cadena**);**    Serial**.**println**(**"Edad: "**);**  **while** **(**Serial**.**available**()** **==** 0**)** **{}**  entero **=** Serial**.**parseInt**();** // pasar a entero  Serial**.**println**(**"Es "**+** String**(**entero**));**  **}** |

Existen otras funciones que nos pueden ayudar a simplificar la lectura de conjuntos de bytes como **Serial.readBytes().**

**Serial.readBytes()** <https://www.arduino.cc/en/Serial/ReadBytes>

### Ejemplo de prueba

Crearemos un programa que se comunique con el PC y al recibir la cadena "encender" / "apagar" hará lo propio con el led conectado al Pin 13 y GND (directamente sobre la placa).



|  |
| --- |
| String comando**;**  void setup**()** **{**  Serial**.**begin**(**9600**);**  // Pin 13 como salida  pinMode**(**13**,** OUTPUT**);**  **}**    void loop**()** **{**  **while** **(**Serial**.**available**()** **==** 0**)** **{** **}** // Esperar ...    comando **=** Serial**.**readString**();**  **if(**comando**.**equals**(**"encender"**))** **{**  Serial**.**println**(**"encendido"**);**  digitalWrite**(**13**,** HIGH**);**  **}** **else** **if** **(**comando**.**equals**(**"apagar"**))** **{**  Serial**.**println**(**"apagado"**);**  digitalWrite**(**13**,** LOW**);**  **}**  **}** |