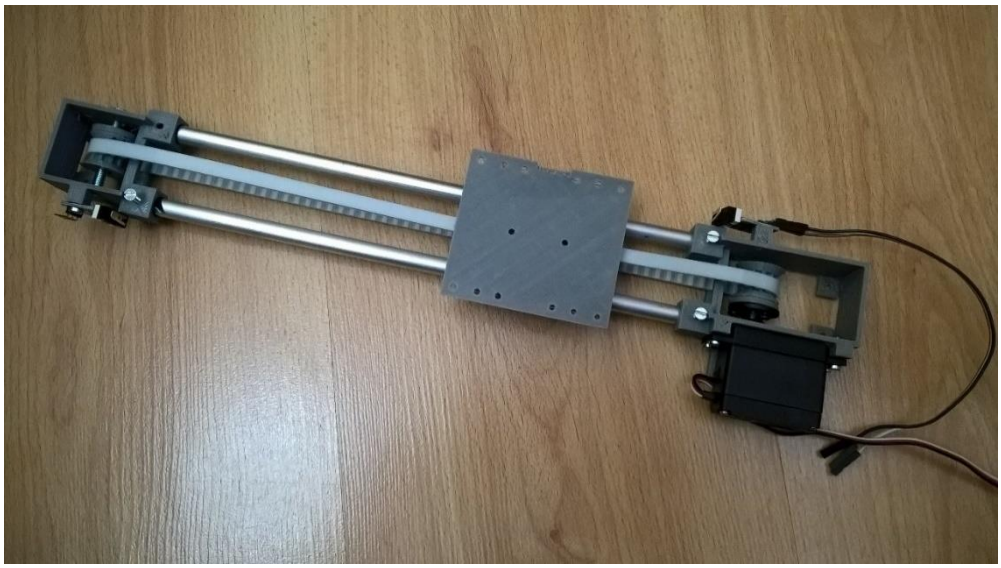
Actuador lineal



Práctica 4 – Teoría (v1.7 septiembre 2022)

Software para robots

Cristian González García

gonzalezcristian@uniovi.es

Basado en el material original de Jordán Pascual Espada

Índice

1. Introducción

Durante la sesión de prácticas programaremos el comportamiento de los actuadores y 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.

Nota muy importante: si se os queda atascado el carro en un lateral durante su funcionamiento por una mala o incorrecta programación, o veis que esto va a ocurrir, y al subir el código queda el motor haciendo fuerza y golpeando y haciendo «saltar/forzar» el carro y los engranajes, desenchufad rápidamente uno de los cables del servomotor para que así podáis subir el código (activando finales de carro o cambiando dirección del motor) sin que se estropee el actuador lineal, el motor, o alguno de los otros componentes. Una vez esté el código subido, enchufáis ese cable y listo. Todo tipo *plug and play*.

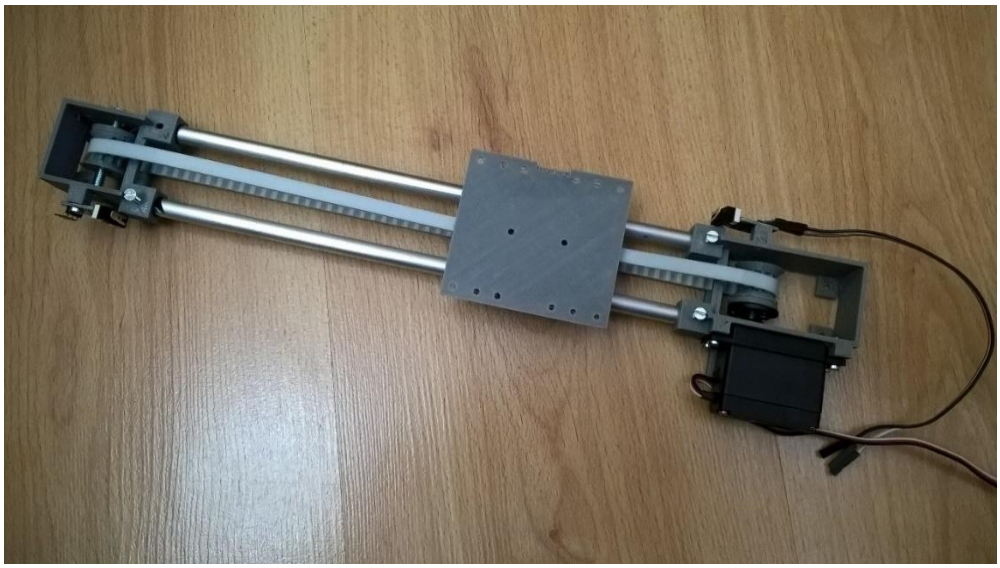


Ilustración 1 Actuador lineal

2. Servomotor de rotación continua

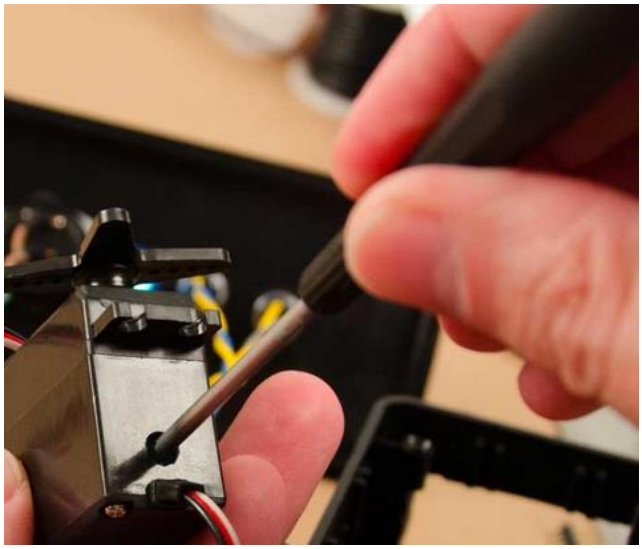
En este ejemplo vamos a realizar una prueba de funcionalidad a un servomotor de rotación continua.

Aunque utiliza la misma librería que los servomotores de rotación no continua, el mecanismo de funcionamiento no es el mismo y las señales que se le envían tienen un significado diferente. En vez de ser un ángulo, se le envía la dirección y velocidad de giro.

Además, este motor necesita un voltaje de entre 4,8V y 7,2V. Luego, necesitaremos enchufar el Arduino con a la red eléctrica, pues el USB no da suficiente potencia. Esto implica que solamente pueda ser usado con el pin de 5V del Arduino y en caso de querer usar una potencia mayor hubiera que alimentarlo externamente (pero compartiendo misma línea de tierra).

Servomotor de rotación continua SM-S4303R: este tipo de servomotor permite realizar movimientos de rotación continua en sentido horario y antihorario 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 y no haga ningún ruido o zumbido. Si está parado y hace ruido significa que está recibiendo potencia y quiere moverse, pero no tiene suficiente potencia para realizar el movimiento. En ocasiones, **la calibración también se puede perder por el uso**. Normalmente, se necesita un destornillador de estrella.

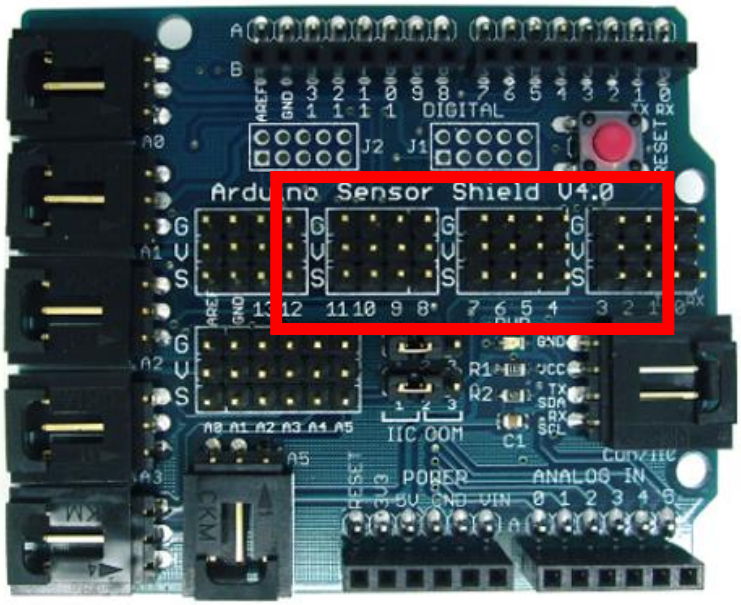


Alimentación Eléctrica

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, mientras que en futuras practicas utilizaremos baterías recargables. Un cargador de 12V también valdría, pero haría que se calentara más la placa y se desgastara primero, además de ser una potencia innecesaria para esta práctica.

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.

1. Después de importarla observaremos que se ha incluido un nuevo #include en nuestro programa.
2. Creamos la variable global de tipo Servo.
3. Inicializamos el puerto Serial para imprimir la salida en el método `setup()`, y asociamos el pin 8 al servo. No hace falta especificar que el pin 8 será una salida digital pues lo hará la librería Servo por nosotros. Además, **esta librería, nos permitirá utilizar como PWM cualquier pin digital**.
4. En el método `loop()` indicamos que el servo debe moverse en sentido antihorario y realizamos una pausa en la ejecución del programa durante 2 segundos. Después de esto, el servomotor sigue manteniendo el sentido de giro indicado.
5. A continuación, indicamos que debe detenerse y realizamos una pausa en la ejecución del programa durante 2 segundos. Hay que tener en cuenta que, a pesar de la pausa, el servomotor sigue manteniendo el sentido de giro indicado.
6. Finalmente indicamos que debe girar en sentido horario y realizamos una pausa en la ejecución del programa durante 2 segundos. Después de esto, el servomotor sigue manteniendo el sentido de giro indicado.
7. Código de ejemplo:

```
//1.-
```

```
#include <Servo.h>

//2.-
Servo servo;

void setup(){
  //3.-
  Serial.begin(9600);
  servo.attach(8);
}

void loop(){
  //4.- Posición giro antihorario (0) esperar 2s
  Serial.println("Enviar: giro en sentido 1");
  servo.write(0);
  delay(2000);

  //5.- Posición detener (90) esperar 2s
  Serial.println("Enviar: detener");
  servo.write(90);
  delay(2000);

  //6.- Posición giro horario (180) esperar 2s
  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 cuanto más cercano al 90 y más rápido cuanto más
cercano al 0 y 180
}
```

3. Joystick

Este tipo de joysticks funcionan con un mecanismo de entrada analógica para las direcciones del joystick referentes a los ejes X e Y, y de entrada digital para el botón pulsador que poseen.

En el siguiente ejemplo realizaremos una prueba de funcionalidad.

Joystick analógico: permite obtener la medición analógica de los ejes X e Y de forma analógica. Los valores se obtienen en el rango de valores 0 - 1023. En la posición por defecto el joystick devuelve 512,512.

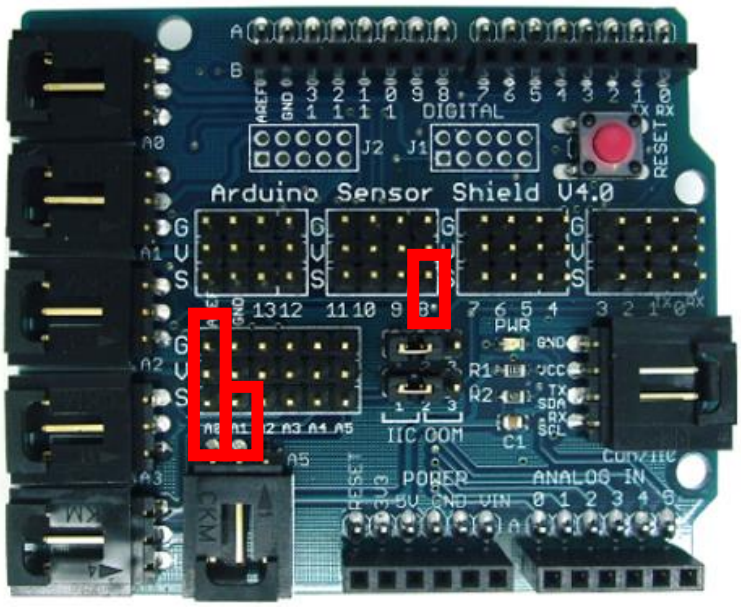
En la siguiente tabla se muestran los valores devueltos (X, Y) según la posición del joystick, teniendo sus pines mirando a nuestra izquierda, y siendo la posición central y sin movimiento (512, 512). Aunque n ose esté tocando, la posición central variará continuamente entre 480-520, dependiendo del joystick:

0, 0	512, 0	1023, 0
0, 512	512, 512	1023, 512
0, 1023	512, 1023	1023, 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 pulsador a: **GND (a GND), +5V (a 5V), VRx (a una entrada analógica) VRy (otra entrada analógica), SW (a un pin digital).**
2. La forma de conexión en la placa de sensores resulta muy poco directa.



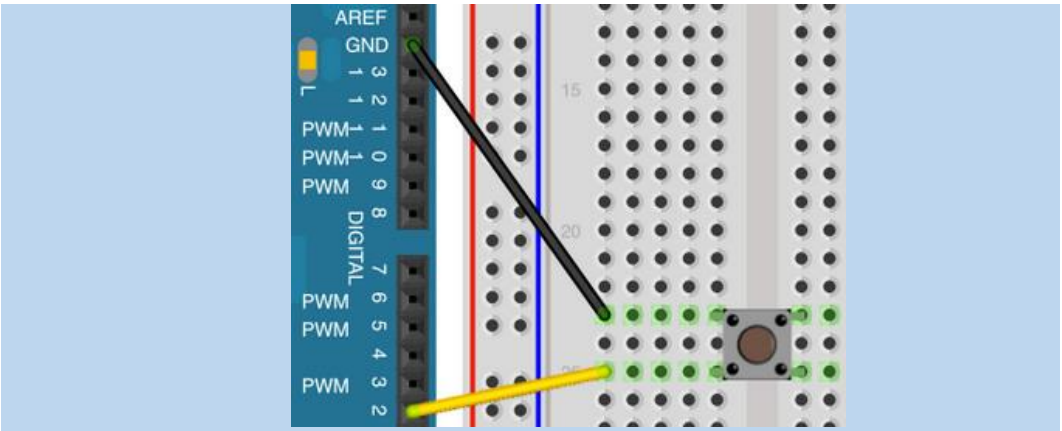
Programación de Arduino:

1. Registramos los identificadores de los pines a los que vamos a conectar el sensor.
2. En el método `setup()` iniciamos el Serial y registramos el pin digital como: **INPUT_PULLUP**, que es un tipo especial de INPUT:
   - https://www.arduino.cc/en/Tutorial/InputPullupSerial
   - https://www.arduino.cc/en/Tutorial/DigitalPins
   - Arduino Uno Rev3: https://store.arduino.cc/arduino-uno-rev3

   Esta función agrega una resistencia interna en Pull-Up de entre 20-50k-ohm, depende de la placa, simplificando la utilización de botones y pulsadores. De esta forma, podemos utilizar directamente el pulsador que incorpora el Joystick. **Revisad la placa que tenéis para comprobar la resistencia, por si acaso.**

```
pinMode(boton_pin, INPUT_PULLUP);
```

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

3. En el método `loop()` leemos las entradas analógicas y digitales y las mostramos por el Serial.
4. Código de ejemplo:

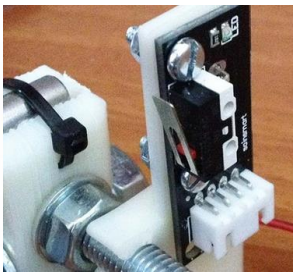
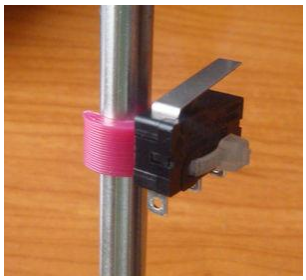
```
    //1.-
    int boton_pin = 8; // Pin digital para el botón
    int X_pin = A0; // Pin analógico para leer eje X
    int Y_pin = A1; // Pin analógico para leer eje Y

    void setup() {
      //2.- Inicializar pin 8 (Entrada) con resistencia
      Serial.begin(9600);
      pinMode(boton_pin, INPUT_PULLUP);
    }

    void loop() {
      //3.-
      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);
    }
```

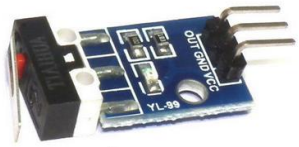
4. Sensor de colisión / choque

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

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





Sensor de colisión: permite alimentaciones entre 3V - 12V.

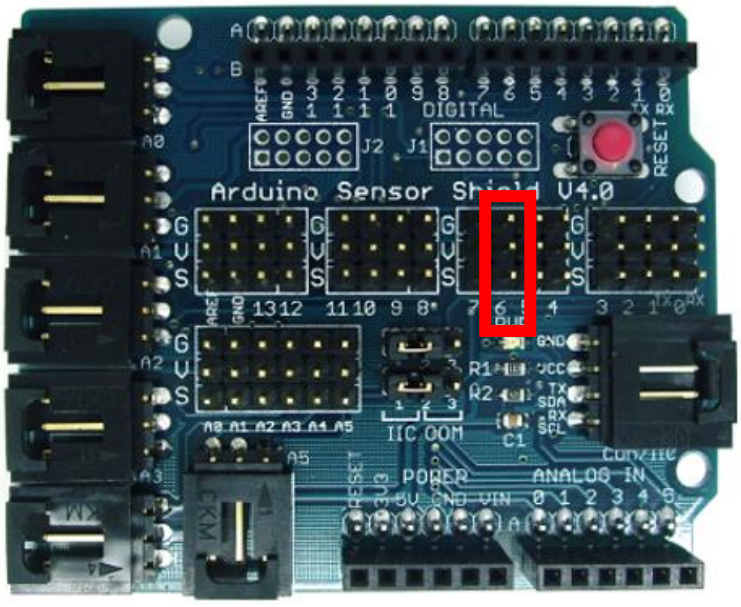
Al detectar una **colisión** se enciende el LED integrado y registra **una señal digital de 0**. En cambio, cuando **no está pulsado**, envía **una señal digital 1**.

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 sensor de colisión 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()` iniciamos el Serial y registramos el pin digital como: **INPUT**
2. En el método `loop()` leemos las entrada digital y la mostramos por el Serial.
3. Código de ejemplo:

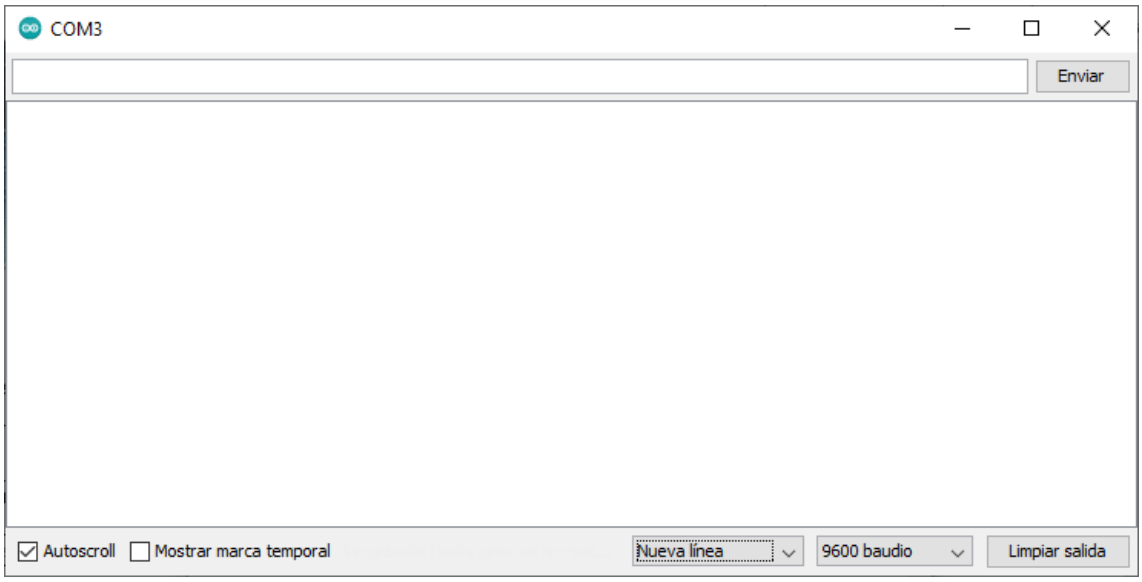
```
void setup() {
  //1.-
  Serial.begin(9600);
  pinMode(6, INPUT);
}

void loop() {
  //2.-
  Serial.println("Sensor Crash: "+String(digitalRead(6)));
}
```

5. Arduino

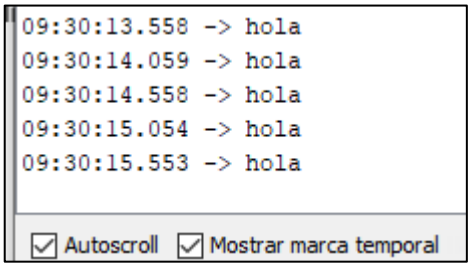
Serial monitor (consola)

Desde el monitor de serie del Arduino IDE se pueden enviar datos desde el PC al Arduino utilizando el puerto serie (USB).

La consola del Arduino IDE tiene varias opciones en su barra inferior:



- **Autoscroll**: para que la visualización de la consola vaya bajando según salgan nuevos datos.
- **Mostrar marca temporal**: para mostrar la hora en la que se imprime por consola un elemento determinado.



- **Nueva línea**: sirve para ajustar como se indicará el paso a una nueva línea al darle al botón enviar. **Esto puede dar problemas** cuando se **introduce texto usando la consola**, pues dependiendo de la opción, añadirá o no un ajuste de línea determinado, lo que puede ocasionar problemas el recibir esta información adicional, como se verá en el ejemplo de lectura de datos. Lo mismo, cuando utilicemos nuestro programa en un ordenador con **Windows, MacOS o Linux** y le estemos enviando texto por el serial. Por eso, dependiendo de la opción elegida, si comparamos String, deberemos añadir al final de esto la opción correcta. Hay 4 opciones:
    - **Sin ajuste de línea**.
    - **Nueva línea**: normalmente, por defecto. \n
    - **Retorno de carro**: \r
    - **Ambos NL & CR**: \r\n

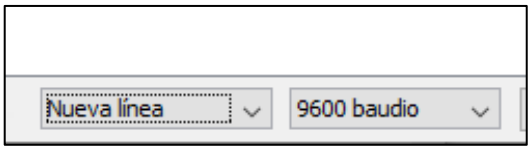


Ilustración 2 En la izquierda el ajuste de línea y en la derecha la selección de baudios

- **Baudios**: es la velocidad seleccionado para la señal de comunicación entre el Arduino y el puerto serie (USB). Se suele utilizar 9600 y tiene que corresponderse con los baudios escritos en el método *begin* del objeto *Serial*, en el código fuente, si no, no mostrará nada la consola.
- **Limpiar salida**: borra todos los datos que han sido impresos en la consola.

Lectura de datos por el puerto serie (Consola)

Métodos importantes:

- **Serial.available()**: devuelve el número de bytes disponibles para leer, en caso de que los haya. Tiene un buffer de 64 bytes/256 bits.
    - https://www.arduino.cc/reference/en/language/functions/communication/serial/available/
    - >0: así, si hay algo en el buffer (habrá más de 0 bytes), imprime todo de 1 en 1.
    - **Nota**: si en la consola tenemos seleccionado «Sin ajuste de línea», esto funcionará perfecto. Pero si seleccionamos nueva línea, veremos que se salta una pregunta e imprime en blanco. Esto es debido a que al introducir algo con nueva línea la consola añade un «\n» que entra en la siguiente pregunta. Lo mismo sucede con las otras dos opciones, en las que, incluso, nos deja enviar sin escribir nada debido a este motivo, algo que no ocurre con la opción «Sin ajuste de línea».
- **Serial.read()**: lee los datos de uno en uno del puerto serie. Devuelve -1 si no hay datos.
    - https://www.arduino.cc/reference/en/language/functions/communication/serial/read/
    - ==0: si no hay nada en el buffer, esperamos por una entrada. Es parecido a un *delay* pero con la capacidad de recibir datos de entrada. Y después continuamos la ejecución del programa.

```
    char consoleData;

    void setup(){
       Serial.begin(9600); // Iniciamos la consola en 9600
baudios, hay que seleccionar lo mismo en la consola
}

    void loop(){
      // N° de datos disponibles para leer, si son más de 0,
entramos en el bucle
      while(Serial.available() > 0){
          consoleData = Serial.read(); // Lee de uno en uno
los carácteres
          Serial.println(consoleData);
      }
    }
```

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

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

```
    String text;
    int age;

    void setup() {
      Serial.begin(9600); // Iniciamos la consola en 9600
baudios, hay que seleccionar lo mismo en la consola
    }

    void loop() {

      Serial.println("Introduzca el nombre: ");
      while (Serial.available() == 0) {} // Esperamos por un
String hasta que sea introducido por el usuario

      text = Serial.readString();
      Serial.println("Su nombre es "+ text);

      Serial.println("Introduzca su edad: ");
      while (Serial.available() == 0) {} // Esperamos por un
String hasta que sea introducido por el usuario

      age = Serial.parseInt(); // Convertimos la entrada en
entero
      Serial.println("Tienes "+ String(age)); // Convertimos
el entero en un String

    }
```

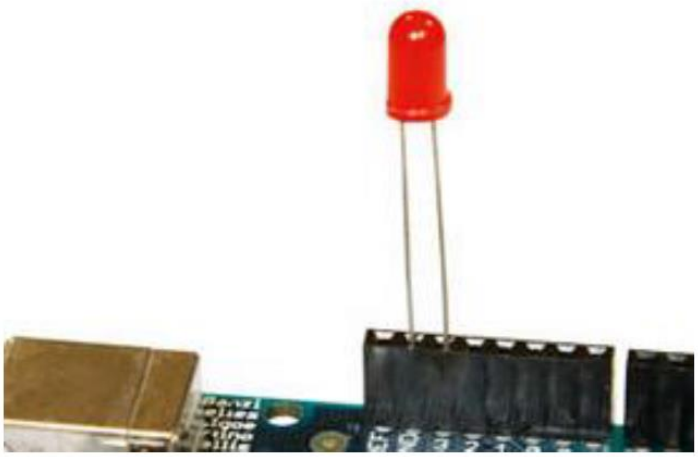
Existen otras funciones que nos pueden ayudar a simplificar la lectura de conjuntos de bytes como **Serial.readBytes()**.

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» encenderá o apagará el LED conectado al pin 13.

Nota: la consola deberá estar en la opción: «**sin ajuste de línea**». Si tenemos seleccionado «**nueva línea**» el String a comparar deberá llevar al final «**\n**».



```
String command;
int pinLED = 13;

void setup() {
  Serial.begin(9600);
  pinMode(pinLED, OUTPUT);
}

void loop() {
  Serial.println("Gestionamos el LED");
  while (Serial.available() == 0) {}  // Esperamos por la
entrada...

  command = Serial.readString();

// Si tenemos nueva línea sería "encender\n"
  if(command.equals("encender")) {
    Serial.println("Encendido");
    digitalWrite(pinLED, HIGH);

  } else if (command.equals("apagar")) {
    Serial.println("Apagado");
    digitalWrite(pinLED, LOW);
  }
}
```

Split en Arduino

Hay 2 maneras de dividir una cadena en Arduino:

1. **A mano**: ir leyendo carácter por carácter e ir detectando los tokens de separación. Para esto, se puede leer la línea con la función readString y después ir iterando carácter a carácter por la línea leída hasta encontrar el token deseado. Hay algunas funciones útiles:
    a. readStringUntil(): lee hasta un determinado carácter que se le pasa por parámetro. Después se puede continuar leyendo todo, o hasta otro carácter determinado.
        i. https://www.arduino.cc/reference/en/language/functions/communication/serial/readstringuntil/
2. **Funciones de C**: se puede usar algunas de las siguientes funciones de C.
    a. strtok: divide una cadena en subcadenas usando el separador que se le pasa como parámetro.
        i. http://www.cplusplus.com/reference/cstring/strtok/
    b. strchr: busca un carácter dentro de una cadena y retorna la posición usando un entero.
        i. http://www.cplusplus.com/reference/cstring/strchr/