Índice

1 CAPÍTULO 1. MOTIVACIÓN, OBJETIVOS Y FASES

1.1 Introducción

El presente PFC surge de la idea de generar un paquete de prácticas de programación, electrónica, automatización y robótica básica orientadas a la asignatura de Tecnología impartida en diferentes cursos de educación secundaria obligatoria. Aunque en principio estas prácticas se orientan a secundaria, la idea es extenderlas para su uso en los primeros cursos de Ingeniería Industrial y de Telecomunicación.

La razón de utilizar Arduino como herramienta educativa viene dada por su fácil manejo, compatible con una gran capacidad de control electrónico. Arduino reduce considerablemente el tiempo de diseño electrónico así como la programación de su microcontrolador. Estamos ante una plataforma muy intuitiva a la que se le puede sacar mucho provecho. Vemos como poco a poco Arduino se va abriendo camino en robótica, sistemas de comunicaciones, domótica, telemática y un largo etcétera. En un futuro no muy lejano y gracias a sus características podremos apostar en esta herramienta como una oportunidad de negocio, implantándola en ámbitos de automatización y comunicaciones industriales, creación de juguetes, así como la programación y manejo de impresoras 3D. Es por tanto una herramienta de trabajo apropiada para el uso docente, pero con una gran proyección en otros ámbitos.

El modelo de trabajo propuesto es ir generando un conjunto de prácticas de dificultad creciente, empezando por montajes muy sencillos, con programas muy simples, adaptados a los primeros de años de la educación secundaria e ir añadiendo complejidad de forma progresiva. La última práctica es un pequeño proyecto de automatización, que puede adaptarse a las necesidades de cada centro docente. En esta práctica, además de Arduino se utilizan otros recursos, como son AVR Studio, Fritzing y Pololu 3pi.

De esta manera, partiendo desde la instalación del IDE Arduino iremos viajando a través de sencillas prácticas hasta ser capaces de controlar un circuito con un cruce, cuatro semáforos, cuatro barreras y un puente levadizo. Así mismo explicaremos cómo “manejar” Arduino con LabVIEW. Según esto, la presente memoria está organizada de la siguiente manera:

- Introducción a Arduino y estudio de la placa Arduino UNO
- Redacción de prácticas Arduino y diseño con Fritzing
- Arduino y LabVIEW
- Estudio y programación del robot Pololu 3pi

- Aplicación práctica en una maqueta

1.2 Trabajos relacionados

Arduino va abriéndose paso poco a poco en campos relacionados con la electrónica, la programación, robótica… Podemos observar como cada vez son más los programas que incluyen entre sus librerías las necesarias para trabajar con las distintas placas de Arduino o bien la introducción al mercado de nuevo software para el manejo de Arduino. En nuestro proyecto usaremos las librerías que National Instruments pone a nuestra disposición para controlar nuestra placa Arduino y realizaremos los esquemáticos de cada práctica con la ayuda de Fritzing, programa que describimos a continuación:

Fritzing

Ilustración 1 Fritzing

Fritzing es una aplicación con la que podemos realizar los esquemas de proyectos electrónicos. Nosotros utilizaremos la versión 0.7.11. Está pensada principalmente para organizar proyectos con Arduino de forma clara y sencilla. Cada proyecto creado contiene tres vistas principales (protoboard, esquema y PCB) organizadas en pestañas como podemos ver a continuación:

La vista protoboard nos muestra una visión realista de la implementación del proyecto. Realizaremos las conexiones arrastrando los componentes que necesitemos y uniéndolos entre sí hasta completar el circuito, pudiendo añadir notas aclaratorias en cualquier parte del diseño.

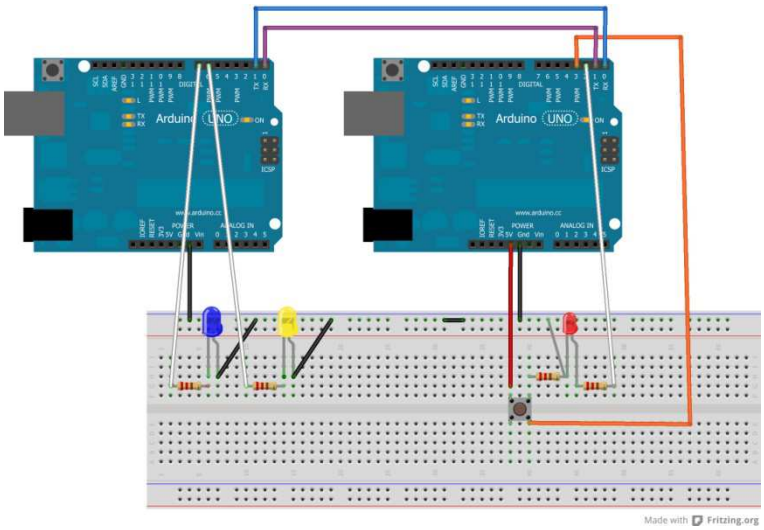


Ilustración 2 Fritzing Vista Protoboard

En la vista PCB (layout) podremos observar la distribución de los componentes en la placa de circuito impreso. El programa nos permite modificar tanto el tamaño como la complejidad de las conexiones en función de nuestras necesidades. Dispone también de una opción de autorruteo para realizar las pistas de cobre (por desgracia esta opción no es muy fiable y se recomienda hacerlo a mano).

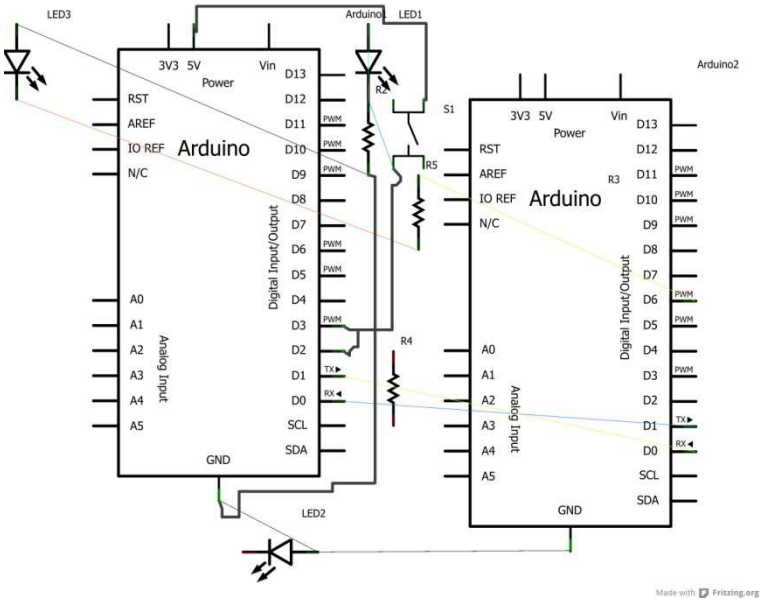


Ilustración 3 Fritzing Vista Layout

La vista esquema nos ofrece de forma abstracta los componentes y conexiones. Gracias a ella podemos comprobar la correcta disposición de las conexiones realizadas en las vistas anteriores.

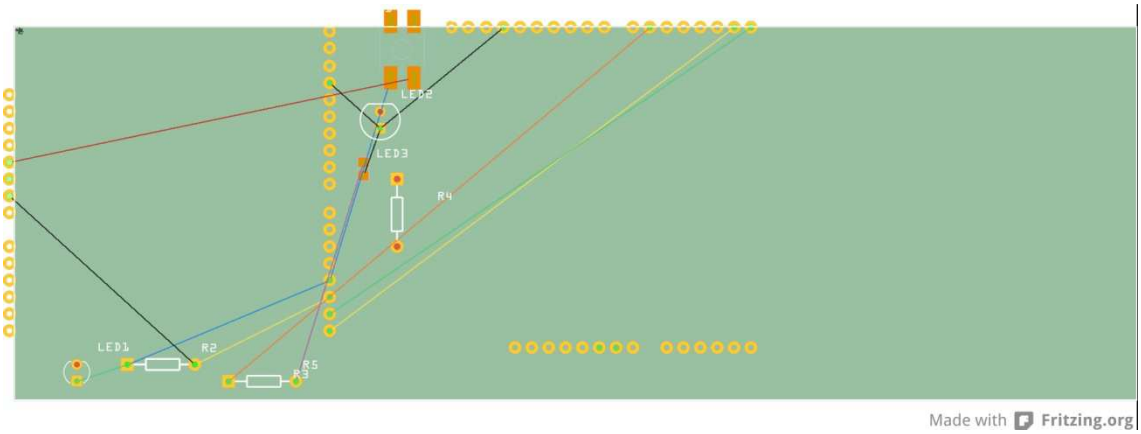


Ilustración 4 Vista Esquema

Fritzing nos permite exportar nuestro proyecto, en cualquiera de sus tres vistas, en diferentes formatos (PDF, PNG, JPG, SVG,…). Tiene licencia GPL para el código y Creative Common para el contenido, es decir, podemos obtenerlo y utilizarlo gratuitamente; además, en su página web www.fritzing.org podremos compartir y discutir proyectos y experiencias.

Otros programas

A continuación vamos a comentar muy de pasada algunos de los programas relacionados con Arduino que xisten en el mercado y nos podrían ayudar en un futuro no muy lejano en la divulgación y enseñanda de esta nueva herramienta.

Minibloq

Minibloq es un entorno de programación gráfica para dispositivos físicos informáticos y robots. Uno de sus principales objetivos es el de llevar la computación física y las plataformas robóticas a la escuela primaria (niños y principiantes. Principales características:

- Fácil e intuitivo
- Generador de código en tiempo real
- Comprobación de errores en tiempo real
- Drag & Drop básico con giro automático
- Interfaz avanzada
- Terminal incorporado
- Todo en uno para comenzar a utilizarlo
- Portable
- Rápido
- Modular y ampiable

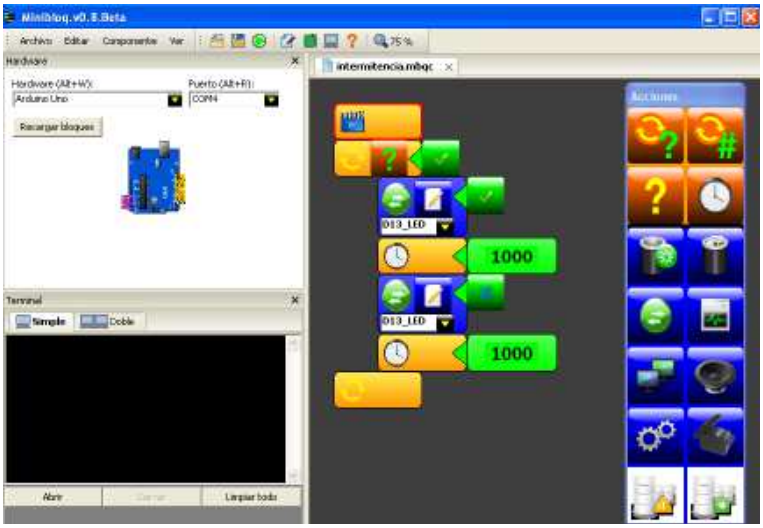


Ilustración 5 Minibloq

Más información en http://blog.miniblog.or/

Ardublock

Se trata de una utilidad gráfica cuya misión es generar código compatible con el entorno IDE Arduino y sus principales características son:

- Herramienta gratuita
- Fácil creación de sketchs para Arduino
- Genera código directamente
- Posee una colección de bloques funcionales básicos que facilitan la comprensión de la programación
- Indicado para aplicaciones educativas en niveles básicos dónde el usuario no necesita tener conocimientos de programación
- Aplicación muy sencilla de instalar
- Se trata de un "plugin" que el IDE Arduino reconoce e instala como Tool

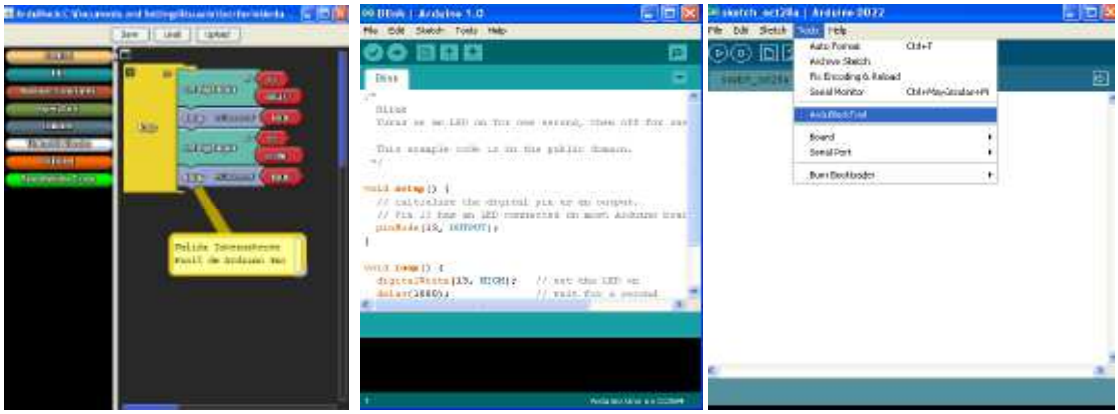


Ilustración 6 Ardublock

Más información en http://blog.ardublock.com/

Amici

Amici ha sido desarrollado por investigadores de DIMEB de la Universidad de Bremen, como parte de la EduWear (proyecto europeo). El software ha sido utilizado en más de 25 talleres formados por niños y jóvenes.

Se suministra junto con una versión completa del IDE Arduino, lo que permite realizar cualquier programa sin tener que cargar ningún otro firmware adicional. Una vez realizado el programa, se genera el código correspondiente y se hace el volcado en la tarjeta Arduino. Es uno de los primeros entornos creados para programar Arduino generando código.

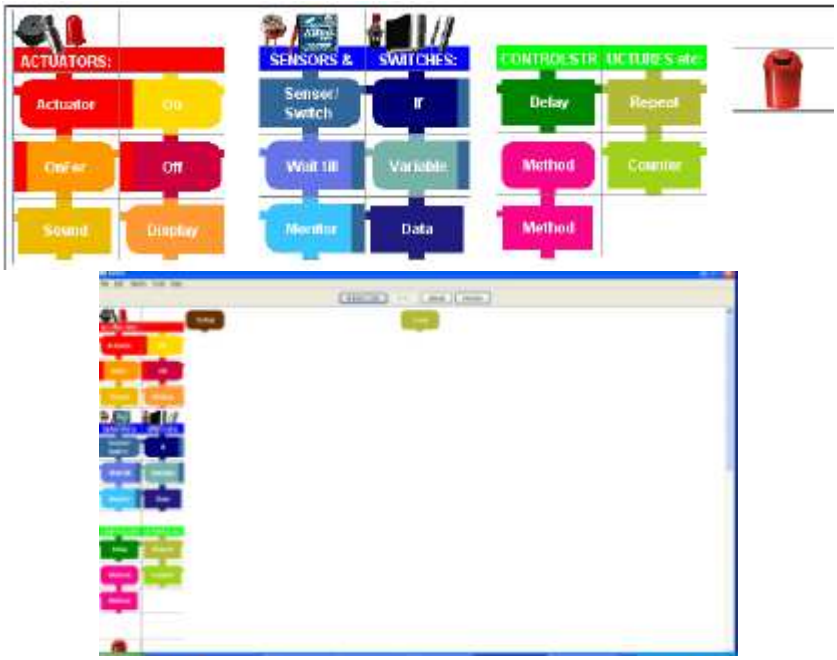


Ilustración 7 Amici

Más información en http://www.dimeb.de/

Modkit

Es un entorno de programación para microcontroladores. Nos permite programar Arduino y hardware compatible con simples bloques gráficos y/o código de texto tradicional. El entorno de bloques gráficos de Modkit está basado en el Scratch, entorno de programación desarrollado por el grupo Lifelong Kindergarten del Media Lab del MIT. Se ejecuta en cualquier navegador web y requiere de un widget de escritorio para comunicarse con la placa de Arduino. Podemos usarlo de forma gratuita o formar parte del Club de Alpha, dónde podemos contribuir con nuestros proyectos y disfrutar de características adicionales antes de que se lancen al público en general.

Ilustración 8 Modkit

Más información en http://www.modk.it/

VirtualBread Boared

Entorno de simulación y desarrollo de aplicaciones integradas que utilizan los microcontroladores. Fácil de usar y capaz de sustituir a una protoboard para experimentar con nuevos diseños.

Nos permite diseñar el prototipo en la protoboard virtual:

1. Realizar la PCB del diseño
2. Importar sketches de Arduino
3. Trabajar con distintas PICs
4. Descargar sobre Arduino la aplicación

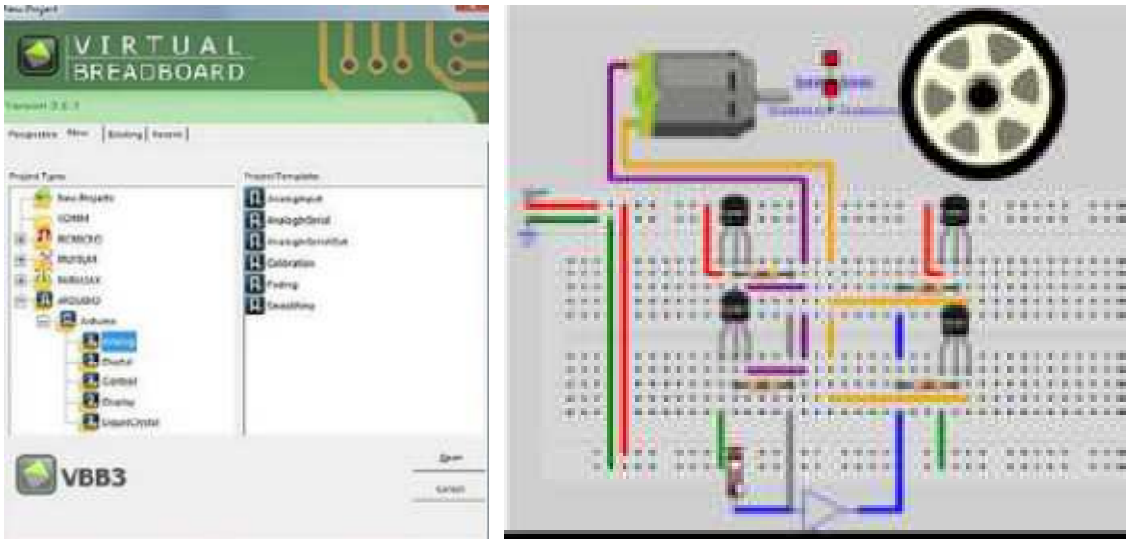


Ilustración 9 VirtualBread Boared

Más información en http://www.virtualbreadboard.com/

Physical Etoys

Es una herramienta de programación visual que une el mundo virtual de los ordenadores con el mundo físico. Con Physical Etoys podemos programar fácilmente objetos del mundo real (por ejemplo robots) para realizar cualquier acción o mover objetos gráficos en la pantalla a través de variables recogidas del mundo físico.
Interfaces con las que se comunica:

- Arduino
- Nintendo Wiimote
- Puerto paralelo
- RoboSapien v2
- Roboquad
- I-Sobot
- Lego Mindstorms Nxt

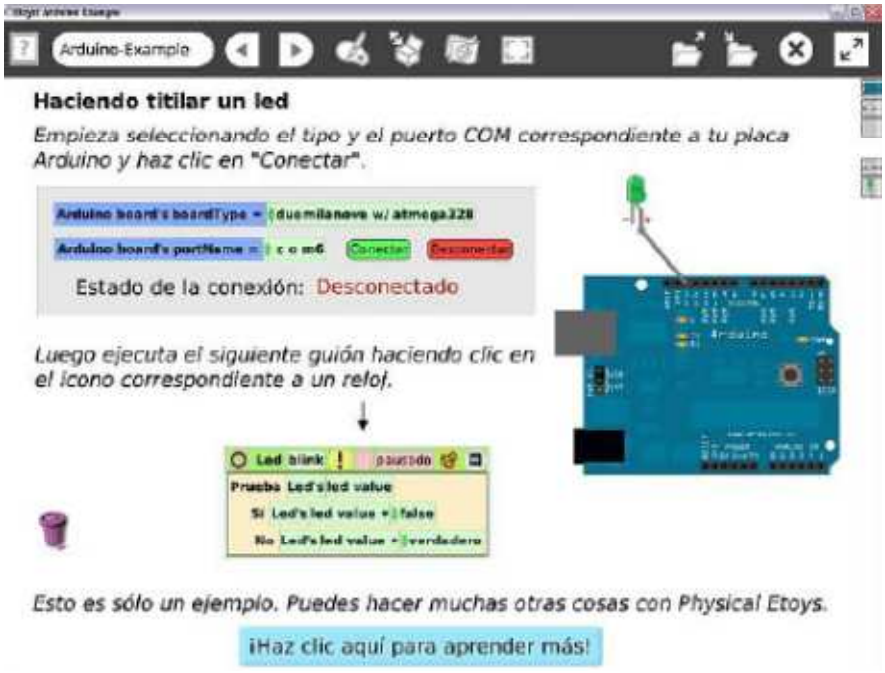


Ilustración 10 Physical Etoys

Más información en http://tecnodata.com.ar/gira/projects/physical-etoys/

S4A (Scratch) + Arduino

Se trata de un proyecto de Citilab y tiene el aval de estar realizada en el entorno Scratch, que es uno de los más conocidos y poderosos en cuanto a programación gráfica se refiere desarrollado en el MIT y escrito en lenguaje Smalltalk.

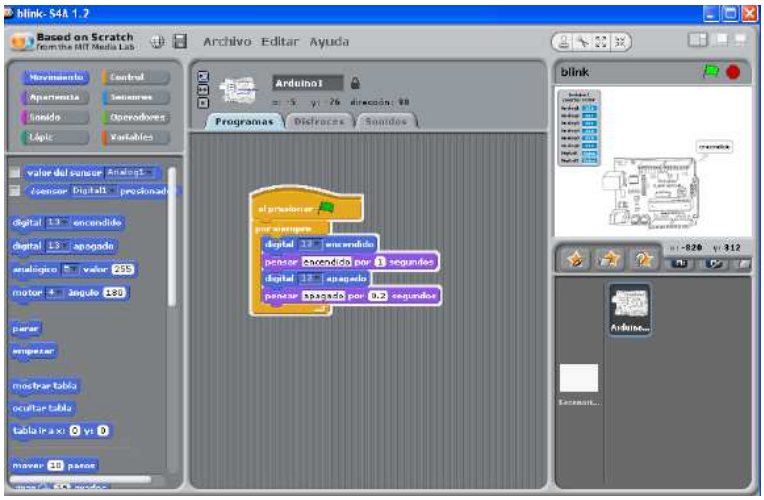


Ilustración 11 S4A (Scratch) + Arduino

Más información en http://seaside.citilab.eu/scratch/arduino

Ardulab

Entorno de trabajo que permite interactuar con una placa Arduino para crear un laboratorio virtual. Podemos realizar una serie de experimentos y actividades orientados principalmente al aprendizaje de sencillos conceptos relacionados con la tecnología (electrónica y robótica). No es un entorno de programación, es un laboratorio virtual que nos permite aprender, probar y conocer como paso previo a la programación de sistemas, con sensores y actuadores, basados en Arduino. Se puede adquirir gratuitamente y tenemos acceso desde el principio a toda su funcionalidad.

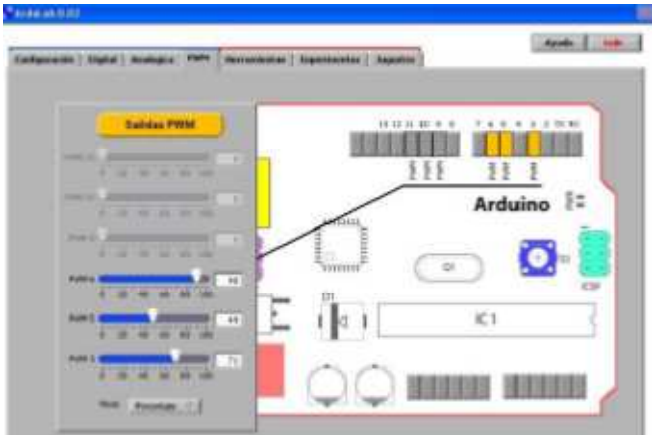


Ilustración 12 Ardulab

Más información en:
http://complubot.educa.madrid.org/proyectos/arduino/ardulab/ardulab_index.php

Rhino + Firefly

Podemos conectar Arduino al poderoso entorno gráfico Rinho a través del conocido plugin Grasshopper, que es un entorno gráfico muy versátil y fácil de utilizar con el que se programa eventos y gobierna imágenes de Rinho. Una de la librerías de Grasshopper es Firefly y está concebida para interactuar con Arduino en el gobierno de entradas y salidas tanto digitales como analógicas.

Ilustración 13 Rhino + Firefly

Más información en http://www.fireflyexperiments.com/download/

MyOpenlab

Entorno orientado a la simulación y modelado de sistemas físicos, electrónicos, robóticos y de control con un amplio campo de aplicaciones didácticas. Sus principales características son:

- Facilidad de uso
- Amplia biblioteca de funciones analógicas y digitales
- Gran biblioteca de objetos gráficos de visualización y/o actuación
- Tratamiento de los tipos de datos y operaciones con éstos.
- Realización de las aplicaciones mediante el uso de bloques de función
- Posibilidad de ampliación de su librería de componentes, editándolos en código JAVA
- Posibilidad de creación de "submodelos de panel" y "submodelos de circuito" encapsulados
- Librerías propias: Elementos Funcionales; Elementos de Decoración, Visualización y Actuación

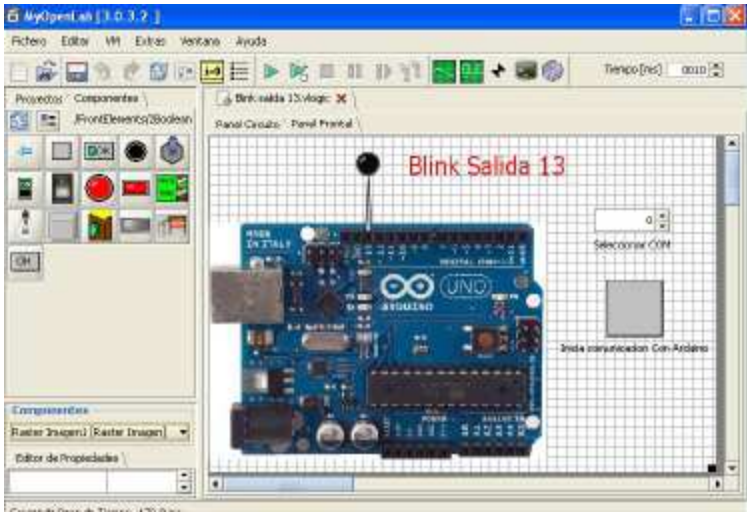


Ilustración 14 MyOpenlab

Más información en http://es.myopenlab.de/

2 CAPÍTULO 2. LA PLACA ARDUINO Y EL ENTORNO DE DESARROLLO

En el presente apartado se van a describir las principales características de Arduino como plataforma; profundizaremos en la placa con la que hemos realizado todos nuestros ejercicios (Arduino UNO); estudiaremos el entorno de programación así como ciertas funciones específicas y veremos al final como conseguir e instalar el IDE de Arduino.

2.1 Introducción

Utilizaremos para el desarrollo del proyecto el hardware y software de Arduino. Como indican sus creadores, Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y relativamente fáciles de usar. Arduino se "vende" como herramienta pensada para artistas, diseñadores, arquitectos y cualquiera que esté interesado en crear objetos o entornos interactivos.

Arduino basa toda su fuerza en los microcontroladores Atmega168, Atmega 328, Atmega 1280, Atmega8 y otros similares; chips sencillos y de bajo coste capaces de desarrollar múltiples diseños. Su hardware no deja de ser por tanto una plataforma controlada por un microcontrolador mediante computación física como otros muchos disponibles hoy en día en el mercado. Pero, ¿qué le hace diferente del resto? Tres pilares son los causantes de su gran aceptación y rápida expansión: **bajo coste** (tarjetas económicas y con soporte para Windows, Linux y Macinthos); incorporación de **librerías y funciones específicas** que facilitan y simplifican la programación y quizá lo que le hace más fuerte, software de **código abierto**.

El lenguaje de programación de Arduino es el "wiring", que está basado en el lenguaje "processing". Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización y que está enfocado a la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. Fue creado por Ben Fry y Casey Reas a partir de reflexiones en el Aesthetics and Computation Group del MIT Media Lab. Processing es desarrollado por artistas y diseñadores, de ahí la anterior definición de Arduino, como una herramienta alternativa al software propietario.

Con objeto de hacer esta memoria lo más autocontenida posible, en el *Anexo I* se incluye la descripción de las placas y módulos de extensión de Arduino más comunes. Las principales características de la placa Arduino utilizada en la realización de este proyecto se resumen a continuación.

2.2 La placa Arduino UNO

Vamos a analizar a fondo la placa elegida para nuestro proyecto: Arduino UNO.

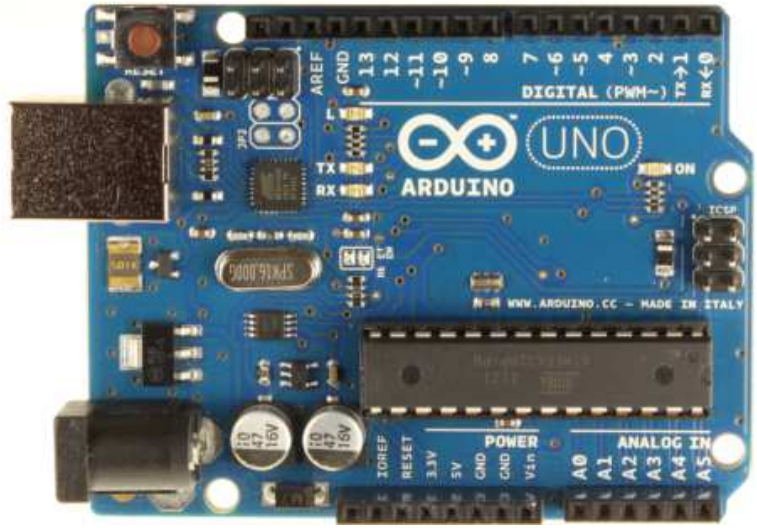


Ilustración 15 Placa Arduino UNO

Su tamaño es de 74x53 mm. La programamos mediante una conexión USB que también usaremos para alimentarla (5V). Existe la posibilidad de usar la alimentación externa, que ha de ser de 9V. Posee 14 pines de E/S digital (6 de las cuales pueden ser usadas como PWM) y 6 pines analógicos. Nos da la oportunidad de alimentar nuestro circuito con dos voltajes distintos, 5V o bien 3,3V.

Vamos a explicar a continuación cada parte con más detalle.

Comenzamos con la más importante, el microprocesador ATmega328, que posee una memoria flash de 32 KB (512 bytes son usados por el bootloader), RAM de 2KB y 1KB de memoria EEPROM. El voltaje de operación, como ya hemos dicho, es de 5V y la frecuencia de trabajo del reloj es de 16 MHz. Destacamos también la preinstalación del bootloader.

Dispone de un reset (botón rojo) que suministra un valor LOW que reinicia el microcontrolador. A su lado, encontramos el conector ICSP (In Circuit Serial Programming), que es el sistema utilizado en los dispositivos PIC para programarlos sin ser necesario la retirada del chip del circuito del que formase parte.

Vayamos ahora conociendo las capacidades de cada pin: los pines 3, 5, 6, 9, 10 y 11 son pines provistos de 8 bits de salida PWM (modulación por ancho de pulsos). Estos pines nos permiten obtener información del exterior y que la placa actúe en función de dicha información (sensores, motores, servos,..). Los pines 0 (Rx) y 1 (Tx)

son los encargados de enviar y recibir datos serie TTL. La función de los pines 2 y 3 es la de manejar interrupciones (Arduino UNO sólo es capaz de manejar dos interrupciones por tanto). Encontramos que los pines 10, 11, 12 y 13 sirven de apoyo a la comunicación SPI con la biblioteca SPI. El bus SPI (Serial Peripheral Interface) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. Es un estándar para el control de cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj. Tenemos 3 pines de tierra marcados como GND (0V). La alimentación al circuito puede ser de 5V o 3,3V en su respectivo pin. Podemos aplicar un voltaje de entrada a la placa mediante el pin Vin cuando ésta sea alimentada por una fuente externa y conocer el valor exacto del voltaje aplicado a la placa. Respecto a entradas analógicas, Arduino UNO dispone de 6 distribuidas en los pines A0, A1, A2, A3, A4 y A5. Cada una de ellas proporciona una resolución de 10 bits (1024 valores). Por defecto se mide en estos pines la tierra a 5V, aunque podemos cambiar la cota superior de este rango mediante el pin AREF, que se encarga de la tensión de referencia para las entradas analógicas.

El puerto USB nos permite una comunicación serie con el ordenador mediante el estándar de los controladores USB COM, sin necesidad de controlador externo. La placa nos avisa con un parpadeo de los leds Rx y Tx que la comunicación se está llevando a cabo.

El conector plug hembra de 2.1 mm lo podemos usar para alimentar a la placa externamente, evitando así el uso del USB (si el sketch ya está cargado en la placa, no necesitamos el ordenador para que la placa funcione, basta alimentarla). *En el Anexo II se incluye el datasheet de la placa para mayor información.*

2.3 Entorno de programación y funciones específicas

2.3.1 Entorno de Programación

Para programar nuestra tarjeta Arduino UNO, acudimos a su página web (http://www.arduino.cc/) y descargamos el software de nuestro sistema operativo. Este software es bastante sencillo e intuitivo de usar (en el capítulo siguiente explicaremos cómo descargar, instalar y comenzar a utilizarlo). Es de licencia con distribución y uso gratuito (open-hardware). En esta misma página podemos acceder a un foro en que seremos ayudados por la gran comunidad de usuarios de Arduino.

El entorno de desarrollo de Arduino lo constituye un editor de texto, donde plasmaremos el código; una consola de texto; un área de mensajes y la típica barra de herramientas con sus menús.

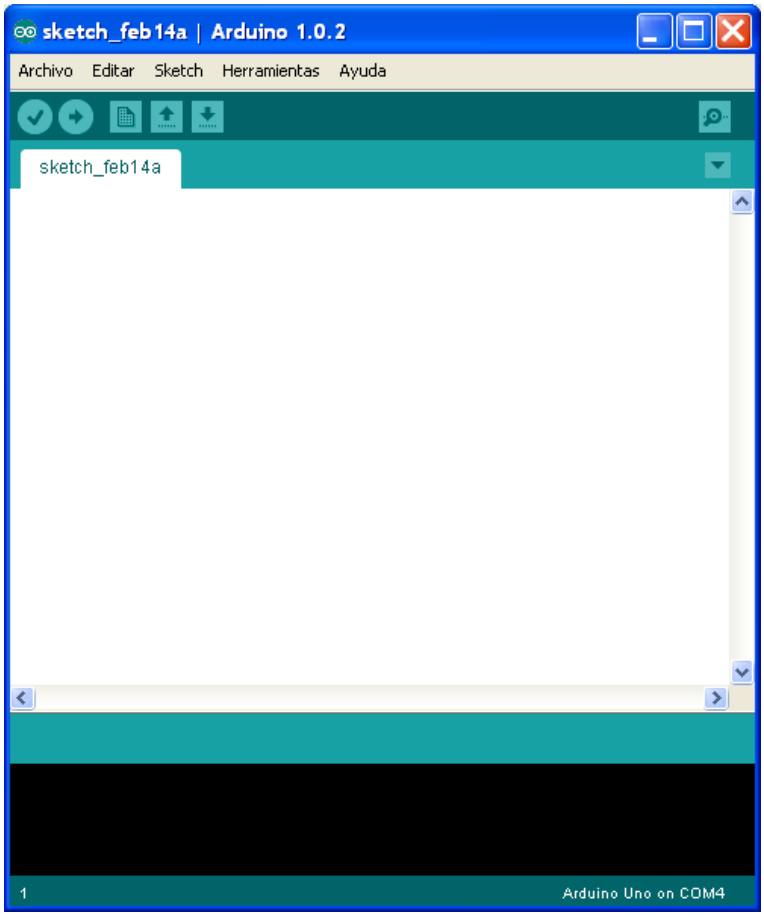


Ilustración 16 IDE Arduino

El código lo escribiremos en un “sketch” (programa) en el editor de texto. El área de mensajes (zona inferior negra) nos mostrará el estado de carga de los programas y los posibles errores. La barra de herramientas es bastante intuitiva: el primer botón verifica el código, el segundo carga el programa en la tarjeta, el tercero nos abre un

nuevo “sketch”, el cuarto nos da la opción de abrir un programa que se encuentre en el pc, el quinto es para simplemente guardar y la lupa de la derecha inicia la monitorización serie.

Dentro de los cinco menús horizontales podremos encontrar diversos submenús que son los típicos de cualquier programa estándar: edición, trato de archivos, configuración de la tarjeta, selección del puerto com, carga de ejemplos… A lo largo de este documento iremos explicando más a fondo cada uno de ellos.

2.3.2 Estructura y funciones específicas

Estructura de un programa

Arduino se programa en C++, admitiendo la gran mayoría de librerías usadas en C++ y todas sus estructuras básicas. Todo sketch tiene siempre la misma estructura:

```
DECLARACIÓN DE VARIABLES;

void setup() {…..}

void loop() {……}
```

- **SETUP()** → La función `setup()` se establece en cuanto se inicia un sketch. Se usa para iniciar las variables declaradas anteriormente, asignar pines, cargar librerías, etc. Esta función sólo se ejecuta una vez desde que se conecta la placa al pc o se reinicia.
- **LOOP()** → Una vez inicializados y preparados todos los valores y funciones necesarias, esta función, como su nombre indica, se ejecuta sucesivamente (permitiendo al sketch cambiar y responder) hasta la desconexión de la placa. Gracias a esta función controlamos activamente la placa.

Una vez visto la estructura típica de cada programa, las funciones específicas que posteriormente emplearemos en la realización de las prácticas y la maqueta final se resumen en las siguientes tablas:

Funciones específicas

NOMBRE	DESCRIPCIÓN	SINTAXIS	PARÁMETROS	DEVOLUCIÓN
PIN MODE()	Configuramos el pin especificado como entrada o salida	`pinMode(pin, modo)`	pin: número de pin que usaremos modo: INPUT (entrada), OUTPUT (salida)	Nada
DIGITALWRITE()	Escribimos un valor eléctrico (alto o bajo) en el pin seleccionado	`digitalWrite(pin, valor)`	pin: el número de pin que usaremos valor: HIGH ó LOW	Nada
DIGITALREAD()	Leemos el valor en el pin especificado (HIGH ó LOW)	`digitalRead(pin)`	pin: el número de pin que queremos leer (int)	HIGH ó LOW
DELAY()	Pausamos el programa por un determinado espacio de tiempo	`delay(tiempo_en_mseg)`	tiempo en milisegundos	Nada

Tabla 1 Control de entradas y salidas digitales

NOMBRE	DESCRIPCIÓN	SINTAXIS	PARÁMETROS	DEVOLUCIÓN
ANALOGREAD()	Leemos el valor de tensión en el pin analógico. Los valores oscilarán entre 0 y 1023 (8 bits) con una resolución de 0,0049 Voltios.	`analogRead(pin)`	pin: el número de pin que queremos leer (int)	int(0 a 1023)
ANALOGWRITE()	Escribimos un valor analógico (PWM) en el pin deseado. Con esta función podemos controlar la luminosidad de un LED o la velocidad de giro de un motor. Tras la llamada a la función, el pin asignado genera una señal cuadrada estable hasta que se la vuelva a llamar	`analogWrite(pin, valor)`	pin: el número de pin PWM en el que generaremos la señal cuadrada (pulso) valor: ciclo de trabajo (int de 0 [siempre apagado] a 255 [siempre encendido]	Nada
ATTACHINTERRUPT()	Mediante esta función invocamos a otra (saltamos de una función a otra)	`attachInterrupt( interrupción, función, modo)`	interrupción: el número de la interrupción (int que puede ser 0 ó 1) función: función a la que invocamos cuando se hace efectiva la interrupción. Esta función no debe tener parámetros ni devolver nada modo: nos define el momento de disparo de la interrupción. Existen cuatro valores válidos: LOW (cuando el pin se encuentre en este estado); CHANGE (cuando el pin cambie de estado); RISING (cuando el pin pase de LOW a HIGH) y FALLING (cuando el pin pase de HIGH a LOW)	
DETACHINTERRUPT (INTERRUPT)	Apaga la interrupción pasada como argumento (0 ó 1)			
NOINTERRUPTS()	Desactiva todas las interrupciones			
INTERRUPTS()	Activa las interrupciones			

Tabla 2 Control de entradas y salidas analógicas

NOMBRE	DESCRIPCIÓN	SINTAXIS	PARÁMETROS	DEVOLUCIÓN
BEGIN()	Mediante esta función establecemos la velocidad de datos en bits por segundo (baudios) para la transmisión de datos en serie. Para nuestro caso, la velocidad óptima será de 9600 bps	`Serial.begin(velocidad)`	velocidad: velocidad en bits por segundo (long)	Nada
END()	Desactiva la comunicación con la placa, dejando libres los pines Tx y Rx	`Serial.end()`	Ninguno	Nada
AVAILABLE()	Nos informa del número de bytes disponibles para ser leídos por el puerto serie (tanto datos recibidos como disponibles en el búfer de recepción)	`Serial.available()`	Ninguno	Número de bytes disponibles para su lectura (máximo 128 bytes)
READ()	Recoge los datos del puerto serie	`Serial.read()`	Ninguno	El primer byte disponible recibido por el puerto serie (devolverá -1 en caso de que no haya ningún dato)
FLUSH()	Nos vacía el búfer de entrada de datos al puerto serie	`Serial.flush()`	Ninguno	Nada
PRINT()	Imprime los datos del puerto serie como texto ASCII	`Serial.print(valor)` `Serial.print(valor, formato)`	valor: el valor que queremos imprimir formato: especifica el número de la base (para int) o el número de posiciones decimales (para float)	Nada
PRINTLN()	Imprime los datos del puerto serie como texto ASCII añadiendo un retorno de carro y un avance de línea	`Serial.println(valor)` `Serial.println(valor, formato)`	valor: el valor que queremos imprimir formato: especifica el número de la base (para int) o el número de posiciones decimales (para float)	Nada
WRITE()	Escribe datos binarios en el puerto serie	`Serial.write(valor)`	valor: puede ser un valor a enviar como un solo byte, un “string” o un “array”	Nada

Tabla 3 Funciones relativas a la comunicación entre placa y usuario.

2.4 Descarga e instalación

Comenzaremos este manual explicando cómo y dónde descargar el IDE de Arduino. Accedemos a http://www.arduino.cc/ y pinchamos la pestaña download.

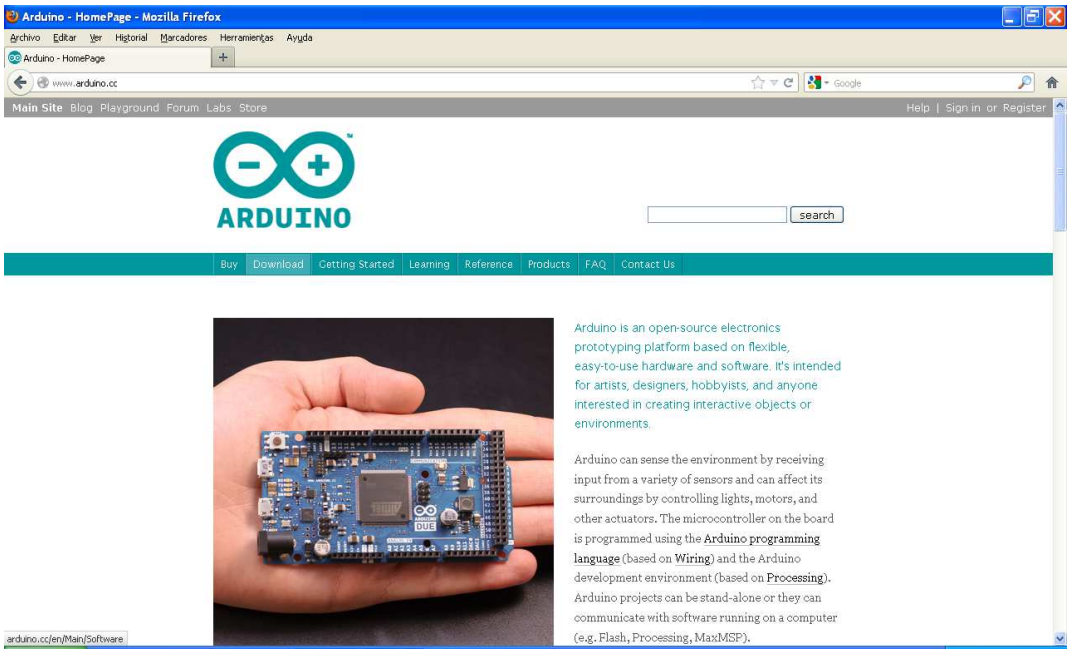


Ilustración 17 Descarga Arduino

A continuación, clicamos la versión para nuestro sistema operativo (Windows en nuestro caso):

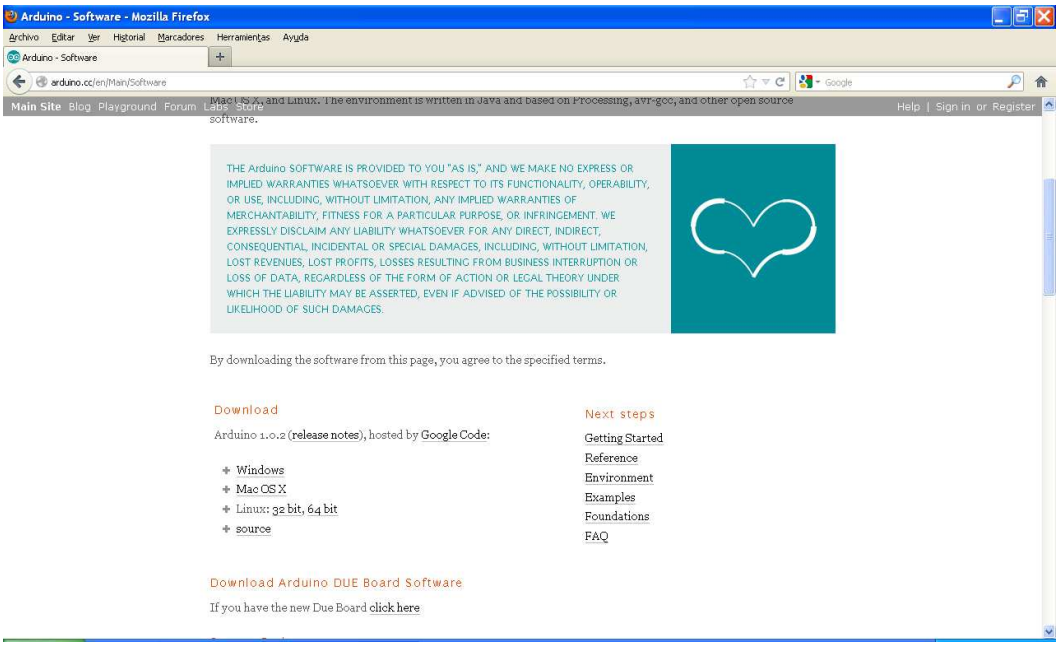


Ilustración 18 Arduino Windows

Descargamos el archivo (arduino-1.0.3-windows.zip de 91.2 Mb). Una vez descargado, descomprimimos el archivo en el directorio raíz: C:\arduino-1.0.3. Dentro de la carpeta creada, buscamos el icono 

y abrimos el programa. Es hora de conectar nuestra placa y esperar que el pc la reconozca. Para corroborar que todo ha ido bien, debemos observar en la placa como el LED de PWR queda encendido. Dependiendo de la versión de Windows, éste será capaz de reconocer a la placa automáticamente o no. En el caso de Win7 y Vista, este paso es automático: descargamos los drivers de la página oficial y los instalamos; para el caso de XP, deberemos instalarlos manualmente desde la ubicación donde los hayamos guardado (C:\.../drivers/FTDI USB Drivers)

Es hora de comenzar a usar y disfrutar de nuestra Arduino UNO. Ejecutamos el programa y abrimos el ejemplo Blink:

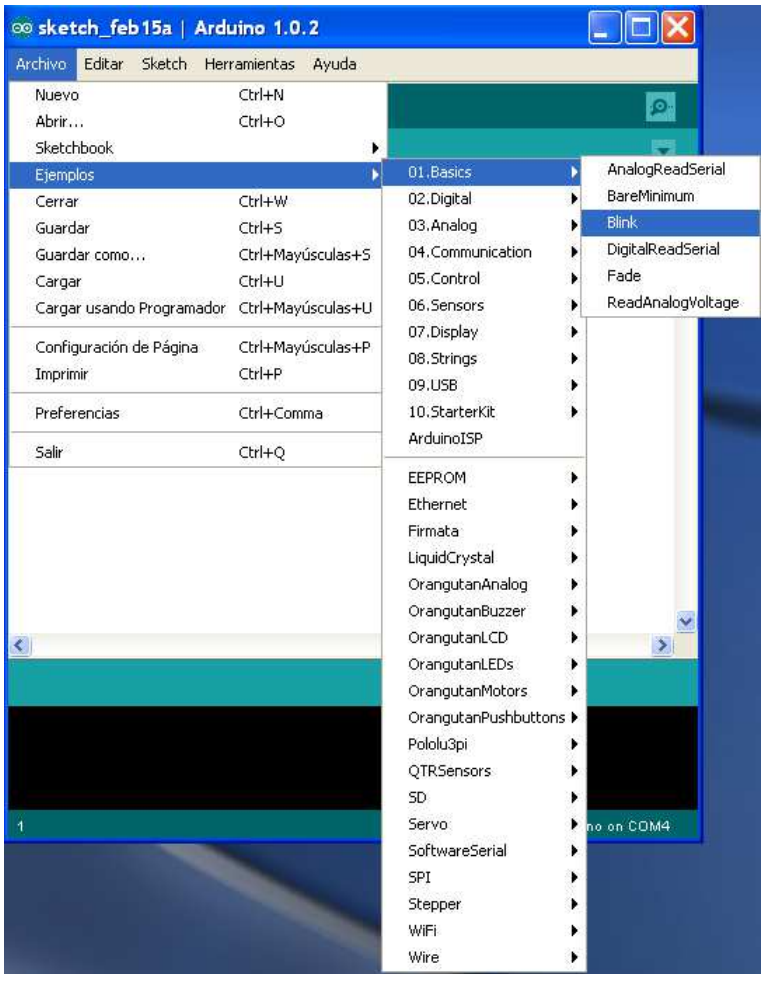


Ilustración 19 Ejemplo Blink

Debemos decirle ahora al programa que placa es la que le hemos conectado, ya que como vimos, existen distintos tipos.

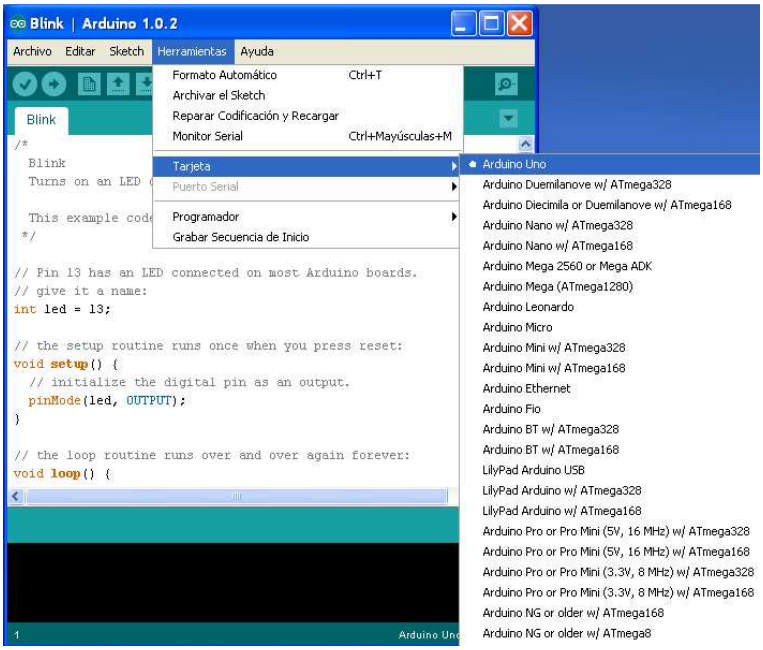


Ilustración 20 Elección placa Arduino UNO

Ahora le decimos en que puerto serie la tenemos conectada. Como trabajamos con USB, Windows simula un puerto seria virtual, el cual suele ser el COM3.

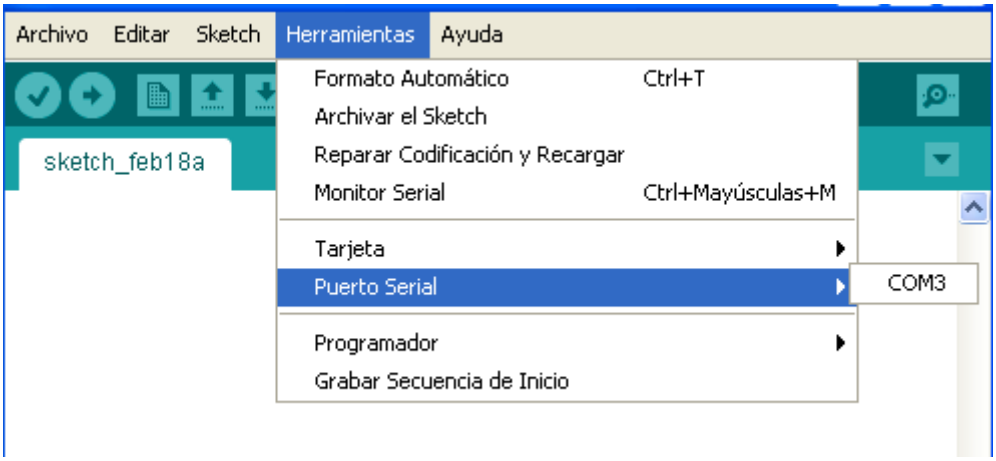


Ilustración 21 Elección puerto serie

Por último, cargamos el sketch presionando el botón Cargar: 

Si todo ha ido bien, podremos ver en el área de mensajes “Done uploading” y como el LED incorporado en la placa comienza a parpadear (color naranja).

Ya tenemos nuestra placa lista para jugar con ella e ir cargando los diferentes programas que a continuación vamos a exponer como ejercicios de prácticas.

3 CAPÍTULO 3. ARDUINO DESDE CERO. PRÁCTICAS

3.1 Prácticas con Arduino

3.1.1 Práctica 1. Control secuencial de un LED

Material necesario: Arduino UNO, 1 resistencia de 220 Ohmios y 1 LED.

Circuito:

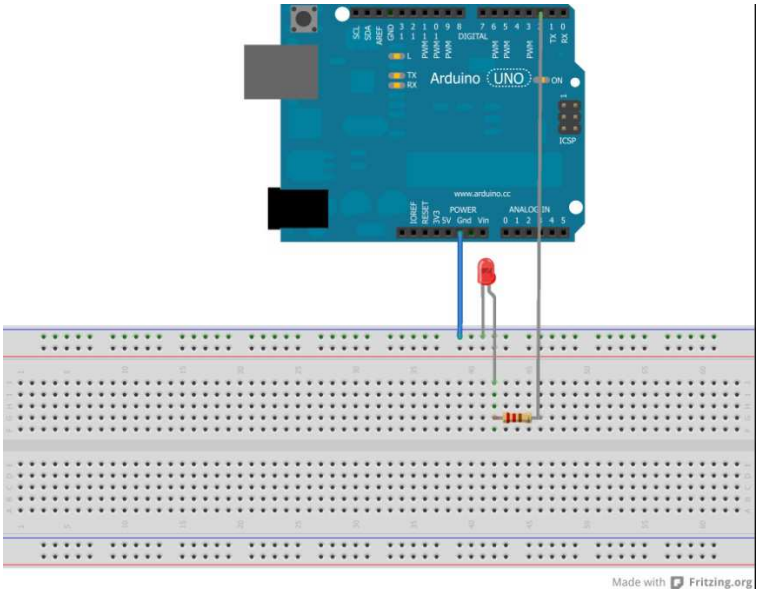


Ilustración 22 Esquemático Práctica 1

Nota: La pata más larga del LED va conectada a la resistencia (parte positiva).

Programa:

```
void setup(){
  pinMode(2, OUTPUT); //Pin 2 en modo salida
}
void loop(){
  digitalWrite(2, HIGH); //Pin 2 en alta (5V)Encendido
  delay(2000); //Esperamos 2 segundos con el LED encendido
  digitalWrite(2, LOW); //Pin 2 en baja (0V)Apagado
  delay(2000); //Esperamos 2 segundos con el LED apagado
}
```

Con este sencillo programa, controlamos el tiempo de encendido y apagado de un LED durante los ciclos de tiempo que le asignemos. Esta práctica es extensible al control de hasta 14 LEDs creando secuencias de encendido y apagado independientes.

Edad recomendada: 3ºESO

Práctica 2. Encendido de un LED (con botón)

Material necesario: Arduino UNO, 2 resistencias de 220 Ohmios, 1 LED y 1 pulsador.

Circuito:

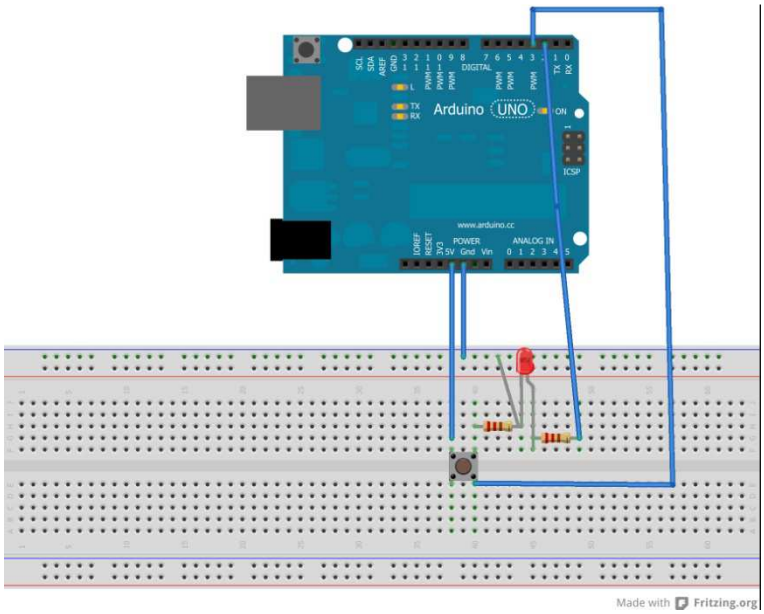


Ilustración 23 Esquemático Práctica 2

Programa:

```
int boton = 0;

void setup(){
pinMode(2, OUTPUT); //Pin 2 en modo salida
pinMode(3, INPUT); //Pin 3 en modo entrada
}
void loop(){
boton = digitalRead(3); //Leemos el estado del pulsador
if(boton == HIGH){
digitalWrite(2, HIGH); //Si está en alta, enciende el LED
}
else{
digitalWrite(2, LOW); //Si está en baja, apaga el LED
}}
```

Con este programa somos capaces de controlar el encendido y apagado del LED mediante un pulsador. Queda como ejercicio la incorporación de otro LED que funcione de forma opuesta al presionar el pulsador.

Práctica 3. Arduino nos avisa si pulsan un botón

Material necesario: Arduino UNO, 2 resistencias de 220 Ohmios, 1 LED y 1 pulsador.

Circuito:

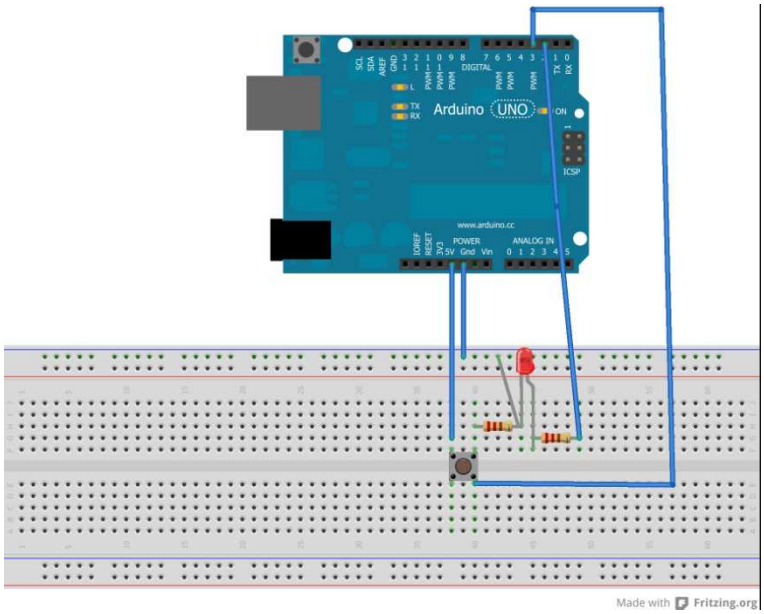


Ilustración 24 Esquemático Práctica 3

Programa:

```
int boton = 0;

void setup(){
  pinMode(2, OUTPUT); //Pin 2 en modo salida
  pinMode(3, INPUT); //Pin 3 en modo entrada
  Serial.begin(9600); //Inicia el puerto serie para comunicarnos
}
void loop(){
  boton = digitalRead(3); //Leemos el estado del pulsador
  Serial.println(boton); //Sacamos por el puerto serie el valor leido
  if(boton == HIGH){
    digitalWrite(2, HIGH); //Si está en alta, enciende el LED
  }
  else{
    digitalWrite(2, LOW); //Si está en baja, apaga el LED
  }
}
```

En este sketch la placa nos avisa por consola si alguien ha pulsado el botón. Podemos ver el cambio de los valores transmitidos abriendo en Menú/Herramientas/Monitor Serial. Mostramos una imagen de la consola:

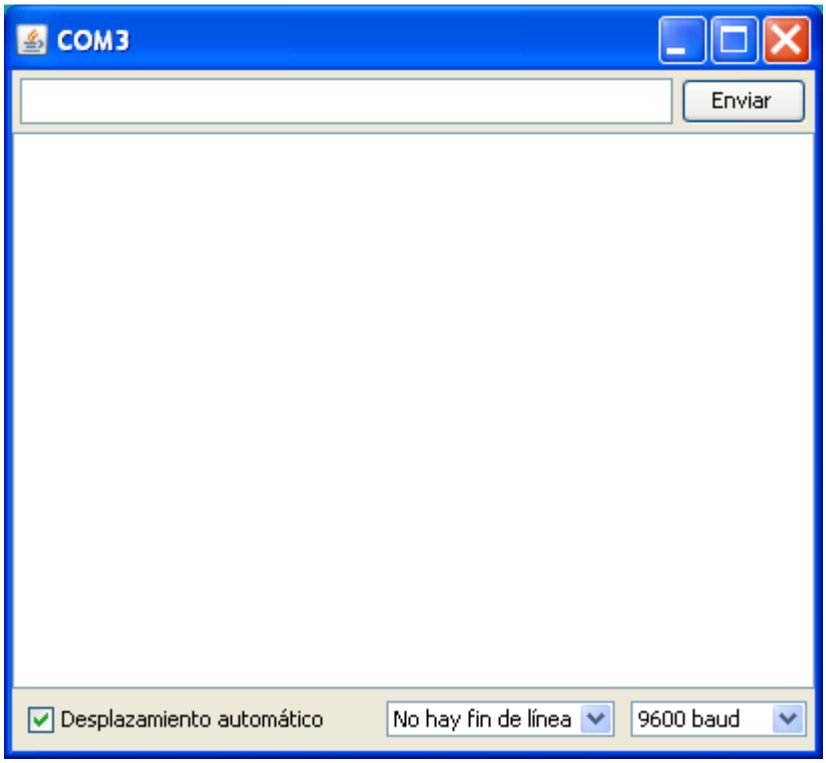


Ilustración 25 Consola IDE Arduino

En la siguiente práctica mejoraremos la comunicación de Arduino hacia nosotros mostrando mensajes más entendibles.

Edad recomendada: 3ºESO

Práctica 4. Arduino nos avisa en castellano si pulsan un botón

Material necesario: Arduino UNO, 2 resistencias de 220 Ohmios, 1 LED y 1 pulsador.

Circuito:

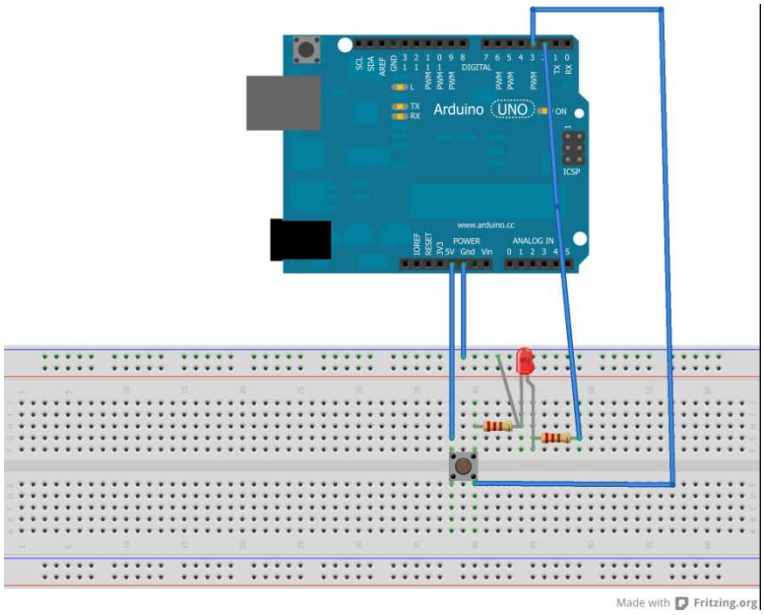


Ilustración 26 Esquemático Práctica 4

Programa:

```
int boton = 0;
int contador = 0;
void setup(){
  pinMode(2, OUTPUT); //Pin 2 en modo salida
  pinMode(3, INPUT); //Pin 3 en modo entrada
  Serial.begin(9600);} //Inicia el puerto serie para comunicarnos
void loop(){
  boton = digitalRead(3); //Leemos el estado del pulsador
  if(boton == HIGH){
    digitalWrite(2, HIGH); //Si está en alta, enciende el LED
    contador = contador + 1;
    Serial.print("Alguien ha pulsado el botón unas ");
    Serial.print(contador);
    Serial.println("veces"); //Muestra por consola
  }
  else{
    digitalWrite(2, LOW); //Si está en baja, apaga el LED
  }}
```

Ahora ya parece que vamos entendiendo a Arduino y él a nosotros. Hemos introducido un contador que vamos incrementando conforme encendemos el LED mediante el botón y Arduino es capaz de mostrarnos el número de veces que ha sido pulsado gracias a la función Serial.print

Práctica 5. Arduino llamando a Arduino

Material necesario: 2 placas Arduino UNO, 3 LEDs, 4 resistencias de 220 Ohmios y 1 pulsador.

Circuito:

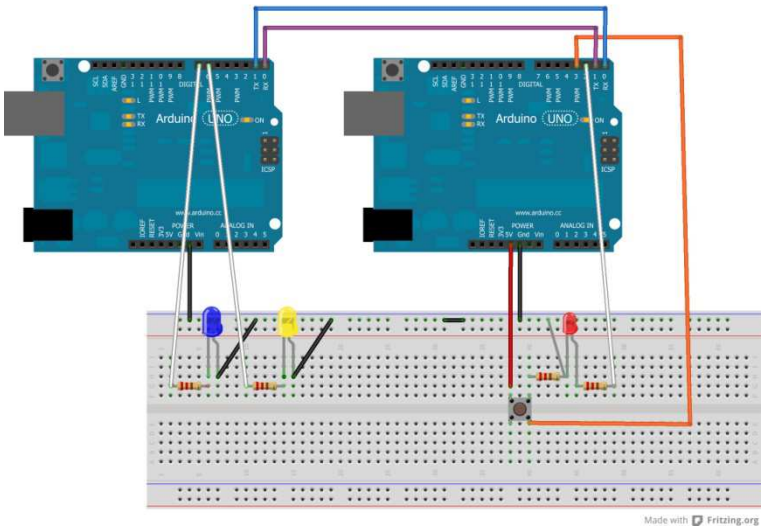


Ilustración 27 Esquemático Práctica 5

Programas:

```
int boton = 0;    //Arduino Maestro
int contador = 0;

void setup(){
  pinMode(2, OUTPUT); //Pin 2 en modo salida
  pinMode(3, INPUT); //Pin 3 en modo entrada
  Serial.begin(9600); //Inicia el puerto serie para comunicarnos
}
void loop(){
  boton = digitalRead(3); //Leemos el estado del pulsador
  if(boton == HIGH){
    digitalWrite(2, HIGH); //Si está en alta, enciende el LED
    contador = contador + 1;
    Serial.print(contador);
    }
  else{
    digitalWrite(2, LOW); //Si está en baja, apaga el LED
  }
}
```

```
int antes = 0;  //Arduino Esclavo
int ahora = 0;

void setup(){
  pinMode(6, OUTPUT); //Pin 6 en modo salida
  pinMode(7, OUTPUT); //Pin 7 en modo entrada
  Serial.begin(9600); //Inicia el puerto serie para comunicarnos
}
void loop(){
  if(Serial.available() > 0){ //Miramos si hemos recibido algo
    ahora = Serial.read();
    if(ahora > antes){ //Si el número recibido es mayor al que
      antes = ahora;   //teníamos, encendemos los LEDs
      digitalWrite(6, HIGH);
      delay(500);
      digitalWrite(6, LOW);
      digitalWrite(7, HIGH);
      delay(500);
      digitalWrite(7, LOW);
    }
  }
}
```

Esta práctica parece que se complica pero vamos a ver que realmente no es para tanto. Queremos que Arduino 1 le diga a Arduino 2 lo que sucede con una de sus entradas y éste reaccione encendiendo dos LEDs. El primer programa es casi idéntico al de la práctica anterior y con él sólo pretendemos enviar a Arduino 2 el número de veces que ha sido pulsado el botón y éste encienda dos LEDs si el número enviado es mayor al anterior recibido.

Edad recomendada: 3ºESO

Práctica 6. Monitorización con Arduino de entradas analógicas

Material necesario: Arduino UNO y 1 potenciómetro.

Circuito:

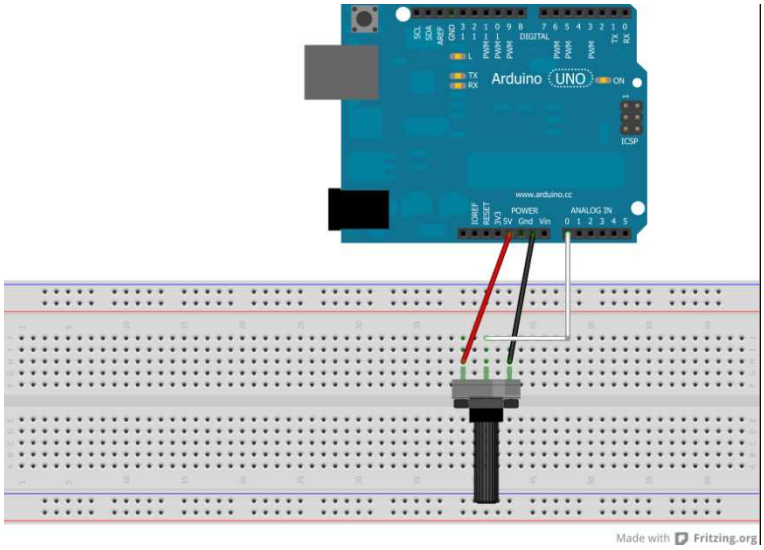


Ilustración 28 Esquemático Práctica 6

Programa:

```
int valor_potenciometro = 0;
void setup(){
  Serial.begin(9600);
 }
void loop(){
  valor = analogRead(0);
  Serial.println(valor_potenciometro); //Leemos el valor del
                                       //potenciómetro
  delay(500); //Damos tiempo a la comunicación
}
```

Arduino nos informa de la diferencia de potencial en cada momento entre las patillas de un potenciómetro conectado al pin analógico 5. A pesar de su sencillez, nos ayudará a desarrollar programas más complejos. Como vimos en las especificaciones de la placa, Arduino UNO trabaja en el rango de 0 a 1023 (8bits), por lo que los valores que obtengamos estarán dentro de dicho rango.

Edad recomendada: 3ºESO

Práctica 7. Entradas y salidas analógicas

Material necesario: Arduino UNO, 1 potenciómetro, 1 LED y 1 resistencia de 220 Ohmios.

Circuito:

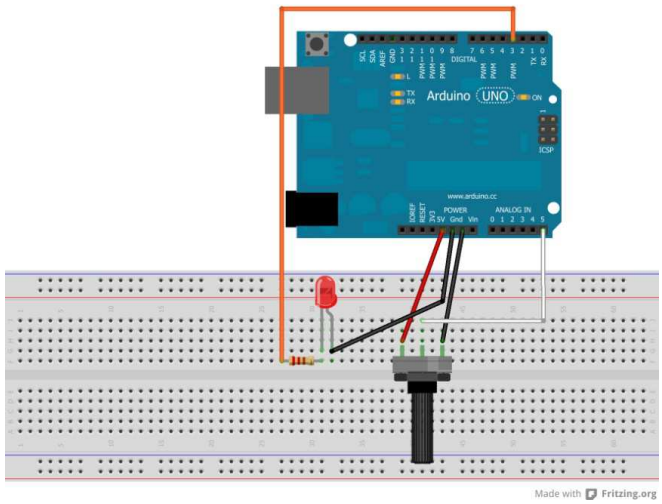


Ilustración 29 Esquemático Práctica 7

Programa:

```
int valor_potenciometro = 0;
void setup(){
  Serial.begin(9600);
 }
void loop(){
  valor = analogRead(5); //Leemos el valor del potenciómetro
  Serial.println(valor_potenciometro); //Sacamos el valor del
                                                    //potenciómetro
  valor_potenciometro = map(valor_potenciometro, 0, 1023, 0, 255);
  analogWrite(3, valor_potenciometro);
}
```

Apoyándonos en la práctica anterior, realizamos la que nos ocupa. Parece que ya vamos viendo la parte práctica de este invento. Esta práctica simula el encendido de una bombilla mediante un regulador. Como vemos, mediante el potenciómetro, vamos dando o quitando luminosidad al LED. Usamos uno de los pines para señales PWM para, una vez leído el valor del potenciómetro y su posterior interpolación, encender el LED con cierta intensidad (valores de 0 a 255).

Edad recomendada: 3ºESO

Práctica 8. Fotorresistencia (LDR)

Material necesario: Arduino UNO, 1 LDR, 1 LED y 2 resistencias (220 y 10k Ohmios)

Circuito:

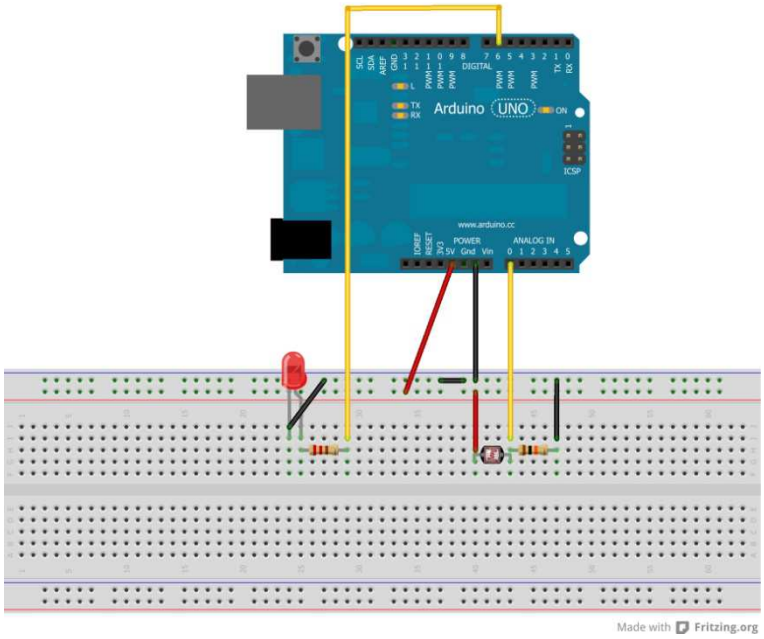


Ilustración 30 Esquemático Práctica 8

Programa:

```
int LDR = 0; // Asignamos el pin 0 a la fotorresistencia
int LED = 6; // Pin 6 para el control del LED
int lectura_LDR; // Donde iremos guardando los valores de la
fotorresistencia
int LDR_mapeado; // Valores interpolados para ir encendiendo el LED
void setup() {
Serial.begin(9600);
pinMode(LED,OUTPUT);
// Aunque no es necesario declarar los pines ANALOGOS como INPUTS, lo
haremos
pinMode(LDR,INPUT);
}
void loop(){
lectura_LDR = analogRead(LDR); // leemos el valor del LDR
LDR_mapeado = constrain(lectura_LDR,0,255);// Interpolamos al rango [0-255]
analogWrite(LED,LDR_mapeado); // Le pasamos al LED el valor mapeado
Serial.print("LDR mapeado = ");// Sacamos por consola el valor que se ha
asignado al LED
Serial.println(LDR_mapeado);
}
```

Una resistencia LDR es quizá el sensor más barato que podemos encontrar en el mercado. Actúa como un potenciómetro, pero en vez de cambiar su valor nosotros, lo hace en función de la luz que recibe. Esos cambios de voltaje los podemos leer con Arduino y especificar distintas acciones para cada rango.

En esta práctica hemos comprobado lo fácil que es controlar una fuente luminosa (LED en este caso) en función de la sensibilidad de la LDR. Con este programa podemos fabricar barreras de luz como las que se utilizan en cualquier aparcamiento o dispositivos de detección de objetos.

Edad recomendada: Bachillerato

Práctica 9. Servomotor

Material necesario: Arduino UNO, 1 potenciómetro y 1 servomotor

Circuito:

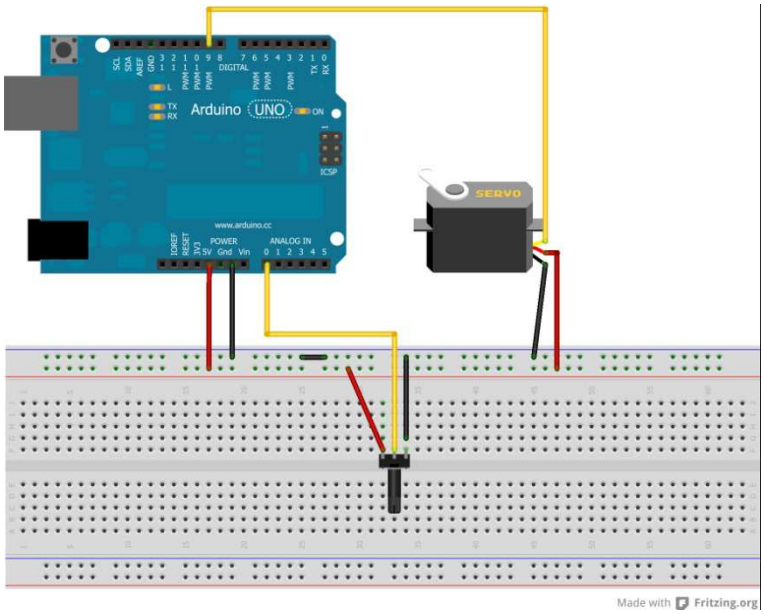


Ilustración 31 Esquemático Práctica 9

Programa:

```
#include <Servo.h> // Cargamos la librería servo para poder manejar
                   // nuestro servo
Servo miServo; // Creamos un objeto servo
int potenciometro = 0; //Asignamos el pin 0 al potenciómetro
int valor; //Variable donde guardaremos los datos del potenciómetro
void setup() {
  miServo.attach(9); //Asignamos el pin 9 a nuestro servo
}
void loop() {
  valor = analogRead(potenciometro); //Obtenemos el valor
                                     //marcado por el potenciómetro
  valor = map(valor, 0, 1023, 0, 179);// Interpolamos a [0-179]
  miServo.write(valor); //Le decimos al servo la posición que debe tomar
  delay(15); // Pausamos 15 milisegundos para refrescar el servo
}
```

Un servomotor es un motor de corriente continua dentro de una carcasa que dispone de cierta lógica de control, la que nos permite decirle la posición en la que

queremos que se sitúe (normalmente el rango de posición es de 0 a 180º, aunque existen servomotores de giro completo) y la velocidad de giro. Además de la lógica de control, dentro de la carcasa hay también una caja reductora que nos brinda mayor fuerza y menor velocidad que un motor normal de continua. Dispone de tres cables para su conexionado, alimentación (rojo), conexión a tierra (negro) y control (blanco).

Esta práctica nos introduce en el control de servos, nos presenta algunas funciones de la librería servo y deja entrever las numerosas utilidades que nos ofrecen los servomotores.

Edad recomendada: Bachillerato

Práctica 10. El girasol. LDR y Servomotor

Material necesario: Arduino UNO, 2 LDRs, 1 servomotor y 2 resistencias de 220 Ohmios.

Circuito:

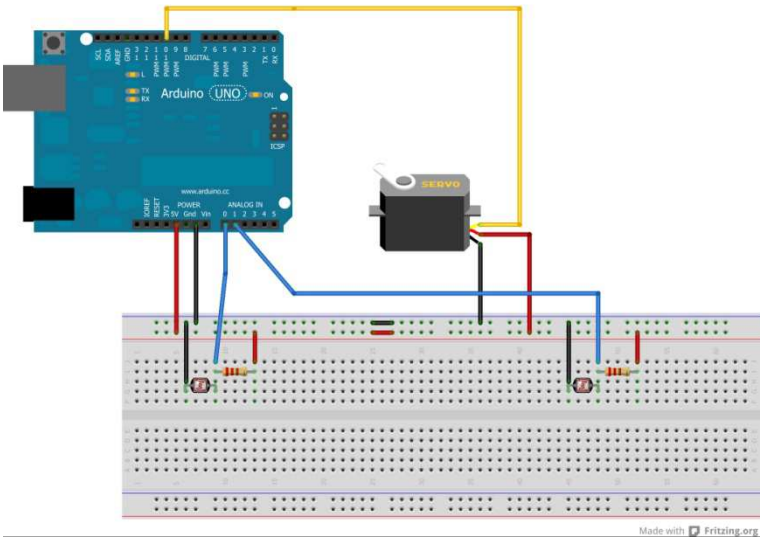


Ilustración 32 Esquemático Práctica 10

El objetivo de esta práctica es el de orientar un servomotor hacia el lugar con mayor luminosidad dado por las dos fotorresistencias. Esta pequeña aplicación, con pequeñas variantes, es muy utilizada en domótica y gestión de placas solares.

Programa:

```
#include <Servo.h> //Cargamos la librería servo
int valorLDR_derecha = 0; //Variables donde guardaremos valores
int valorLDR_izquierda = 0; //obtenidos por las fotorresistencias
int valor_Servo = 0; //Aquí guardaremos la posición del servo

Servo miServo; //Creamos nuestro servo
void setup(){
  Serial.begin(9600);
  miServo.attach(10); //Lo ponemos en el pin 10 (los servos se
                      //controlan con señales PWM)
}
void loop(){
  valorLDR_izquierda = analogRead(1); //Leemos el valor de la LDR
  Serial.print("Sensor Izquierdo: "); //de la izquierda y lo
  Serial.print(valorLDR_izquierda);   //sacamos por pantalla
  Serial.print("   ");
  valorLDR_derecha = analogRead(0); //Igual para el de la derecha
  Serial.print("Sensor Derecho: ");
  Serial.print(valorLDR_derecha);
  Serial.print("   ");
  //Queremos que el servo se oriente hacia el "sensor" donde haya
  //más luz. El servo comienza orientado 0º (hacia la derecha)
  //Si hay más luz en el sensor derecho, se queda como estaba
  if(valorLDR_izquierda < valorLDR_derecha){
    valor_Servo = valor_Servo -10;
    if (valor_Servo < 0){ //El servo no admite valores negativos
      valor_Servo = 0;    //así que los pasamos como 0
    }
  }
  //Si hay más luz en el sensor izquierdo, cambiamos la posición
  //a 179º
  else{
    valor_Servo = valor_Servo +10;
    if(valor_Servo > 179){ //Limitamos el valor del servo a 179
      valor_Servo = 179;   //ya que su rango es de 0º a 180º
    }
  }
  miServo.write(valor_Servo); //Escribimos en el servo el valor
                              //para que este se posicione
}
```

Edad recomendada: Bachillerato

Práctica 11. Motor de continua controlado con chip LD293

Material necesario: Arduino UNO, 1 pulsador, 1 chip LD293, 1 resistencia de 2,2 kilo Ohmios, 1 LED y 1 motor DC.

Circuito:

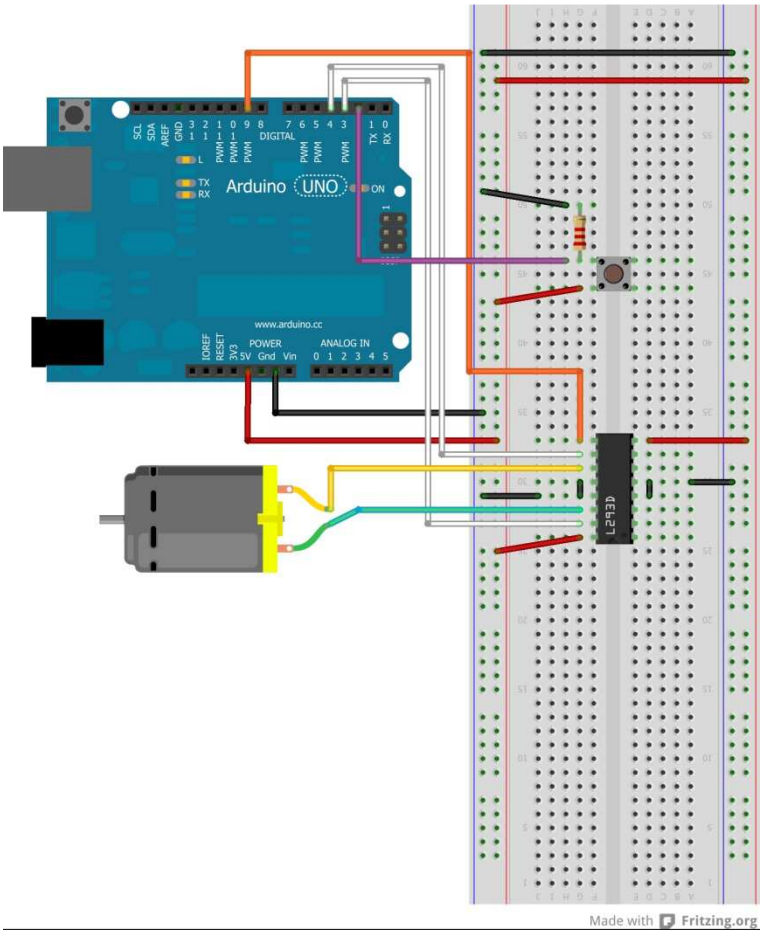


Ilustración 33 Esquemático Práctica 11

El chip LD293 es un driver diseñado para proporcionar corriente a mecanismos impulsores bidireccionales de hasta 1 Amperio con voltajes entre 4,5 y 36 Voltios con una capacidad máxima de disipación de potencia de 5 Wattios. Su principal características es la alimentación propia independiente de la alimentación de los canales, lo que da estabilidad al circuito en el que se monte.

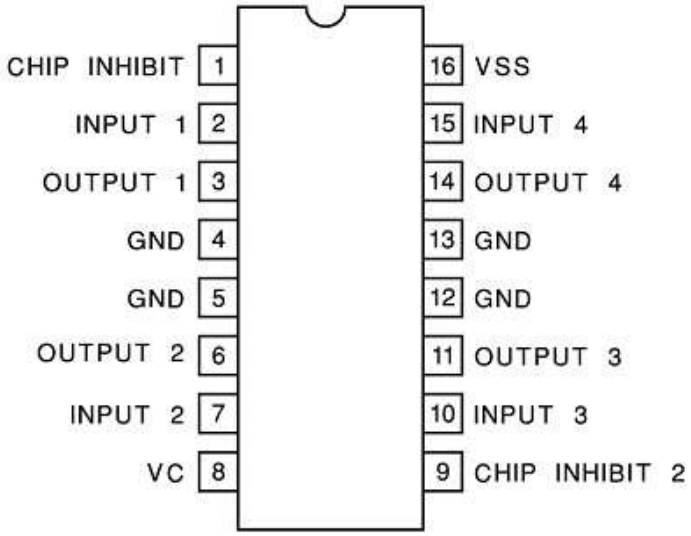


Ilustración 34 Esquema Chip L293D

Con este chip podemos conectar dos motores DC, uno en cada parte del chip.

En esta práctica controlaremos con la ayuda del chip presentado anteriormente el sentido de giro del motor con la ayuda de un pulsador.

Programa:

```
int botonPin = 2; // Ponemos el botón al pin 2
int motorPin1 = 3; // Pata 1 del motor al pin 3
int motorPin2 = 4; // Pata 2 del motor al pin 4
int speedPin = 9; // Canal de habilitación del chip al pin 9
void setup() {
pinMode(botonPin, INPUT);
pinMode(motorPin1 , OUTPUT);
pinMode(motorPin2 , OUTPUT);
pinMode(speedPin, OUTPUT);
digitalWrite(speedPin, HIGH); // Motor encendido desde el comienzo
}
void loop() {
if (digitalRead(botonPin) == HIGH) {
// Si pulsamos el botón, el motor cambia el sentido de giro
// mientras tengamos pulsado el botón
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, HIGH);
}
else {
digitalWrite(motorPin1, HIGH);
digitalWrite(motorPin2, LOW);}}
```

Práctica 12. Control de velocidad y giro de motor de continua

Material necesario: Arduino UNO, 1 chip LD293, 1 motor DC y 2 potenciómetros.

Circuito:

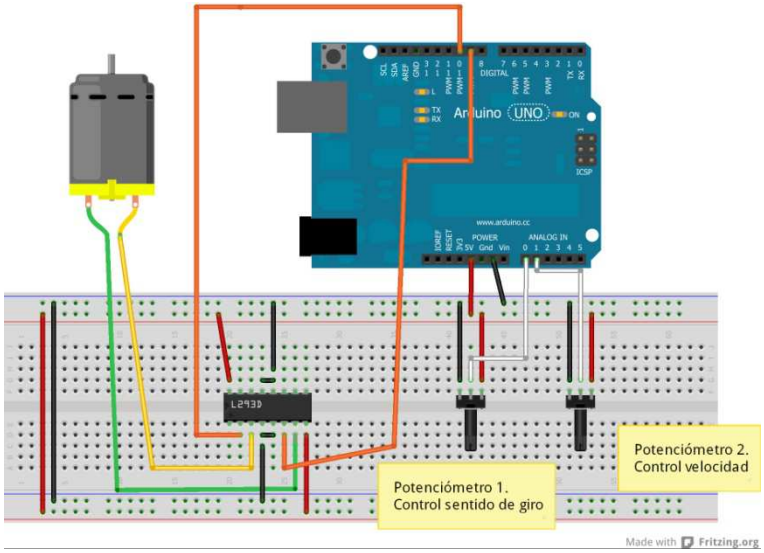


Ilustración 35 Esquemático Práctica 12

En este sketch hemos incorporado dos métodos auxiliares con sus respectivas variables locales para apoyarnos en el control del sentido de giro. Destacamos también la declaración e inicialización de los pines del motor en forma de array.

Programa:

```
int valorPot1; // Leemos el valor del potenciómetro 1
int valorPot2; // Leemos el valor del potenciómetro 2
int valor_interpolado; // Necesitamos interpolar el valor del
//segundo potenciómetro, encargado de dar la velocidad
int motor[ ] = {9, 10}; // Array de asignación de pines al motor
int Potenciometro1 = 0;
int Potenciometro2 = 1;
void setup() { Serial.begin(9600);
pinMode(Potenciometro1, INPUT);
pinMode(Potenciometro2, INPUT);
// Variable de apoyo para poner los pines del motor como salidas
int i; for(i = 0; i < 2; i++){
pinMode(motor[i], OUTPUT);}}
void loop() {
valorPot1 = analogRead(Potenciometro1);
valorPot2 = analogRead(Potenciometro2); // interpolamos el valor del segundo
//potenciómetro al rango admitido por el motor (0-255)
valor_interpolado = map(valorPot2,0,1023,0,255);
// Para valores inferiores a la mitad (512) del primer
//potenciómetro, el motor gira en sentido antihorario
if(valorPot1 <= 512){
Serial.print("Gira a Izquierdas");
// LLamamos al método de Retroceso del motor y le pasamos como
//argumento el valor interpolado del segundo potenciómetro
motorRetro(valor_interpolado);}
// Para valores superiores a la mitad (512) del primer potenciómetro, el
//motor gira en sentido horario
if(valorPot1 > 512){ Serial.print("Gira Derechas");
// LLamamos al método de Avance del motor y le pasamos como
//argumento el valor interpolado del segundo potenciómetro
motorAvance(valor_interpolado);}
Serial.print("/t"); Serial.print("Revoluciones Por Minuto: ");
Serial.println(valor_interpolado);
delay(50);}
// Método Avance.Una pata a cero y la otra con el valor  interpolado del
//segundo potenciómetro
void motorAvance(int Revoluciones1){ analogWrite(motor[0], Revoluciones1);
analogWrite(motor[1], 0);}
// Método Retroceso. Ahora ponemos la pata que estaba a cero
// con el valor interpolado(cambiamos el sentido de giro) y la otra a cero.
void motorRetro(int Revoluciones2){ analogWrite(motor[0], 0);
analogWrite(motor[1], Revoluciones2);}
```

Práctica 13. Semáforo simple

Material necesario: Arduino UNO, 3 LEDs (rojo, verde y amarillo), 3 resistencias de 220 Ohmios, 1 pulsador y 1 resistencia de 2,2 kilo Ohmios.

Circuito:

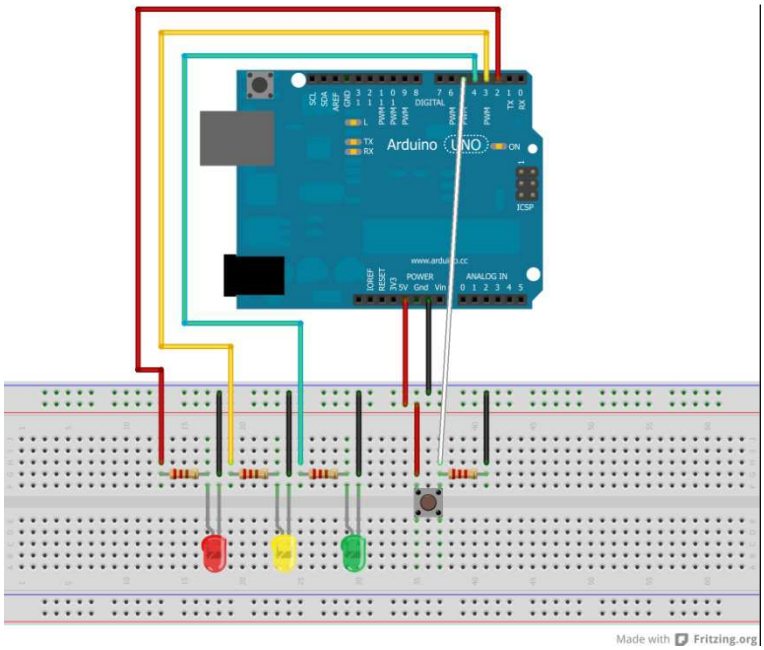


Ilustración 36 Esquemático Práctica 13

En esta práctica entramos en el mundo del control semafórico. Este programa controla manualmente el funcionamiento de un semáforo. Mientras tengamos pulsado el botón, el semáforo irá cambiando su estado y quedará fijo si lo soltamos. Con la función "delay" marcamos el tiempo de encendido de cada LED, algo que como veremos en posteriores ejercicios, nos facilitará la sincronización cuando introduzcamos más semáforos.

Programa:

```
int PinRojo = 2;
int PinAmarillo = 3;
int PinVerde = 4;
int boton = 5;
int estado_actual = 0; //Estado en el que se encuentra el semáforo
void setup(){
pinMode(PinRojo, OUTPUT);
pinMode(PinAmarillo, OUTPUT);
pinMode(PinVerde, OUTPUT);
pinMode(boton, INPUT);
}
void loop()
{
if (digitalRead(boton))
{
if (estado_actual == 0)//Si está en reposo
{
Luces(HIGH, LOW, LOW);//Ponemos el semáforo en rojo
estado_actual = 1;
}
else if (estado_actual == 1)
{
Luces(HIGH, HIGH, LOW);
estado_actual = 2;//Ponemos el semáforo en rojo y amarillo
}
else if (estado_actual == 2)
{
Luces(LOW, LOW, HIGH);//Ponemos el semáforo en verde
estado_actual = 3;
}
else if (estado_actual == 3)
{
Luces(LOW, HIGH, LOW);//Ponemos el semáforo en amarillo
estado_actual = 0;
}
delay(1000); //Esperamos 1 segundo
}}
void Luces(int Rojo, int Amarillo, int Verde)
{
digitalWrite(PinRojo, Rojo);
digitalWrite(PinAmarillo, Amarillo);
digitalWrite(PinVerde, Verde);}
```

Práctica 14. Semáforo controlado por Rotary Encoder

Material necesario: Arduino UNO, 3 LEDs (rojo, verde y amarillo), 3 resistencias de 220 Ohmios y 3 resistencias de 100 kilo Ohmios.

Circuito:

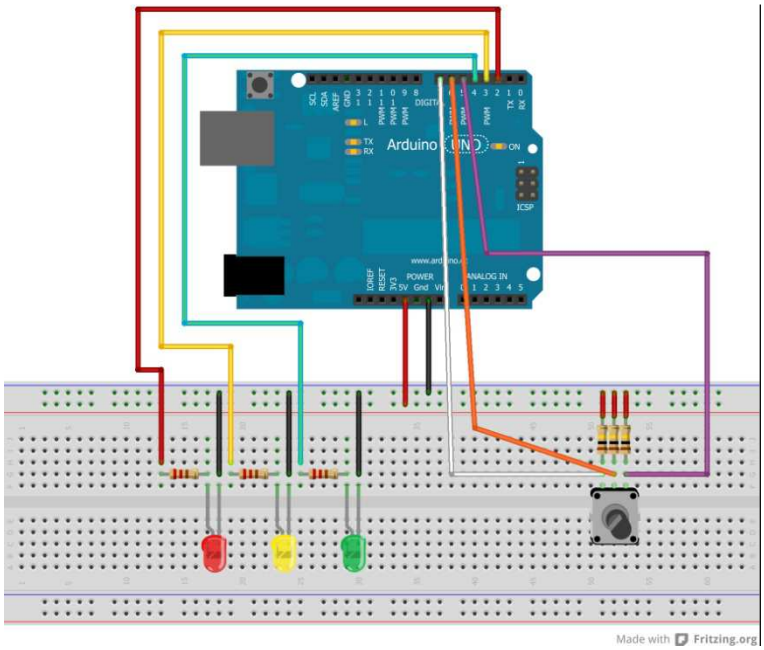


Ilustración 37 Esquemático Práctica 14

Un "rotary encoder" no es más que un potenciómetro electromecánico que convierte la posición angular del potenciómetro en un valor digital o en un pulso. Vulgarmente lo podríamos definir como un potenciómetro sin fin con el cual obtenemos mayor precisión. Aprovecharemos cada posición para ir cambiando los tiempos de encendido de cada LED, lo que nos aproxima al modelo real de funcionamiento de un semáforo. Nosotros utilizaremos uno de tres patillas como el que se muestra en la figura:

Ilustración 38 Rotary Encoder

Programa:

```
int PinRojo = 2;
int PinAmarillo = 3;
int PinVerde = 4;
int PinEncoderA = 6;
int PinEncoderB = 7;
int boton = 5;
int estado_actual = 0;
int PeriodoLargo = 5000; // Tiempo en verde o en rojo
int PeriodoCorto = 700; // Tiempo en amarillo
int CuentaUsada = PeriodoCorto;
int cuenta = 0;
void setup()
{
pinMode(PinEncoderA, INPUT);
pinMode(PinEncoderB, INPUT);
pinMode(boton, INPUT);
pinMode(PinRojo, OUTPUT);
pinMode(PinAmarillo, OUTPUT);
pinMode(PinVerde, OUTPUT);
}
void loop()
{
cuenta++;
if (digitalRead(boton))
{
Luces(HIGH, HIGH, HIGH);
}
else
{
int cambio = getEncoderTurn();
int nuevoPeriodo = PeriodoLargo + (cambio * 1000);
if (nuevoPeriodo >= 1000 && nuevoPeriodo <= 10000)
{
PeriodoLargo = nuevoPeriodo;
}
if (cuenta > CuentaUsada)
{
setEstado();
cuenta = 0;
}}
delay(1);}
```

```
int getEncoderTurn(){                  // Devolverá -1, 0, o +1
static int oldA = LOW;
static int oldB = LOW;
int result = 0;
int newA = digitalRead(PinEncoderA);
int newB = digitalRead(PinEncoderB);
if (newA != oldA || newB != oldB){     // Si hay algún cambio
if (oldA == LOW && newA == HIGH)
{
result = -(oldB * 2 - 1);
}}
oldA = newA;
oldB = newB;
return result;
}
int setEstado()
{
if (estado_actual == 0)
{
Luces(HIGH, LOW, LOW);
CuentaUsada = PeriodoLargo;
estado_actual = 1;
}
else if (estado_actual == 1)
{
Luces(HIGH, HIGH, LOW);
CuentaUsada = PeriodoCorto;
estado_actual = 2;
}
else if (estado_actual == 2)
{
Luces(LOW, LOW, HIGH);
CuentaUsada = PeriodoLargo;
estado_actual = 3;}
else if (estado_actual == 3)
{
Luces(LOW, HIGH, LOW);
CuentaUsada = PeriodoCorto;
estado_actual = 0;
}}
void Luces(int Rojo, int Amarillo, int Verde){
digitalWrite(PinRojo, Rojo);
digitalWrite(PinAmarillo, Amarillo);
digitalWrite(PinVerde, Verde);}
```

4 CAPÍTULO 4. ARDUINO Y LABVIEW

4.1 Instalación de Librerías

A continuación vamos a explicar como instalar la librería LIFA que nos permitirá programar nuestra tarjeta Arduino.

Una vez instalado LabView, veamos los pasos que tenemos que seguir para completar la instalación de dicha librería:

1. Instalar los controladores VISA de National Instruments para nuestro sistema operativo. http://joule.ni.com/nidu/cds/view/p/id/2251/lang/es
2. Descargar e instalar el LabRunTime de NI con Engine Z011 para nuestra versión de LabView 10.
3. Seguidamente instalamos el paquete JKI VI (VIPM) gratuito. http://www.jki.net/vipm
4. Instalamos la interfaz de LabView para Arduino como explica el siguiente enlace: http://digital.ni.com/public.nsf/allkb/A20FBBD3682066908625788600 4D5F4D?OpenDocument
5. Conectamos la placa Arduino al PC como se describe a continuación: http://digital.ni.com/public.nsf/allkb/0F9DADF9055B086D86257841005D1773?OpenDocument
6. Cargamos la interfaz de LabView para Arduino. http://digital.ni.com/public.nsf/allkb/8C07747189606D148625789C005C2DD6?OpenDocument
7. Ya podemos usar el IDE Arduino para implementar el software en la placa.

Una vez tenemos instalada la librería, debemos comunicar LabView con el IDE Arduino. Tenemos que cargar el siguiente fichero (…*\National Instruments\LabVIEW 2010\vi.lib\LabVIEW Interface for Arduino\Firmware\LVIFA_Base)* en la misma carpeta donde tengamos el IDE Arduino. Ejecutamos Arduino y procedemos de la siguiente manera:

1. Abrimos el IDE Arduino y abrimos desde menú el fichero LVIFA_Base.pde

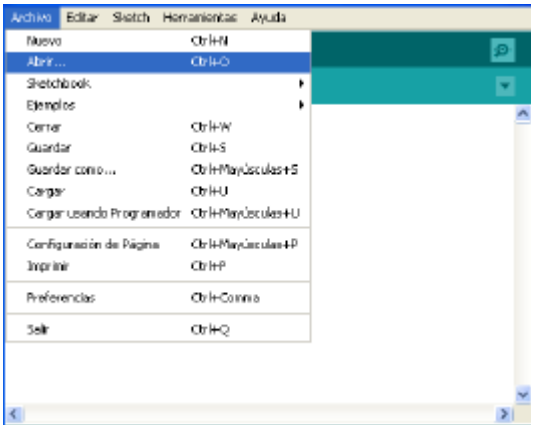


Ilustración 39 Intalación LIFA

2. Cargamos el fichero y seleccionamos nuestra tarjeta (Arduino UNO).

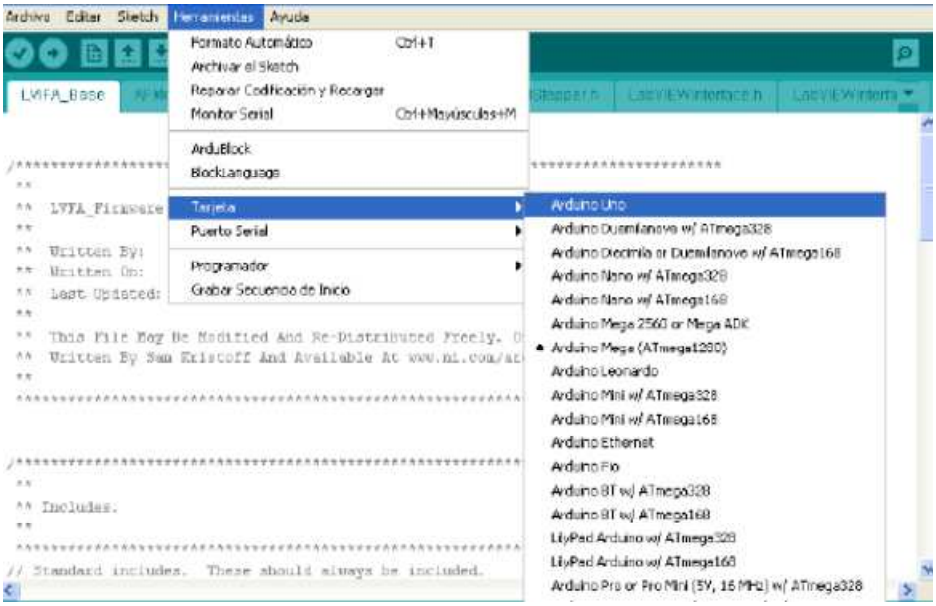


Ilustración 40 Selección tarjeta Arduino UNO para LIFA

3. Seleccionamos el puerto COM.

Ilustración 41 Selección puerto serie para LIFA

4. Cargamos el sketch para que el programa se cargue en la tarjeta y haga que esté preparada para comunicarse con LabView.

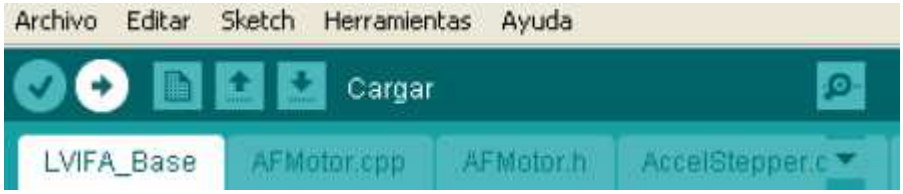


Ilustración 42 Carga sketch LIFA

Una vez realizados todos los pasos anteriormente descritos, ya estamos en disposición de abrir LabView y realizar cualquier proyecto que queramos implementar en nuestra placa Arduino UNO.

4.2 Prácticas con LabView

Práctica 1. Control secuencial de un LED

Al igual que vimos en el capítulo anterior, queremos controlar el encendido y apagado de un LED, pero esta vez lo controlaremos con LabView. Necesitaremos el mismo material y el montaje eléctrico será el mismo.

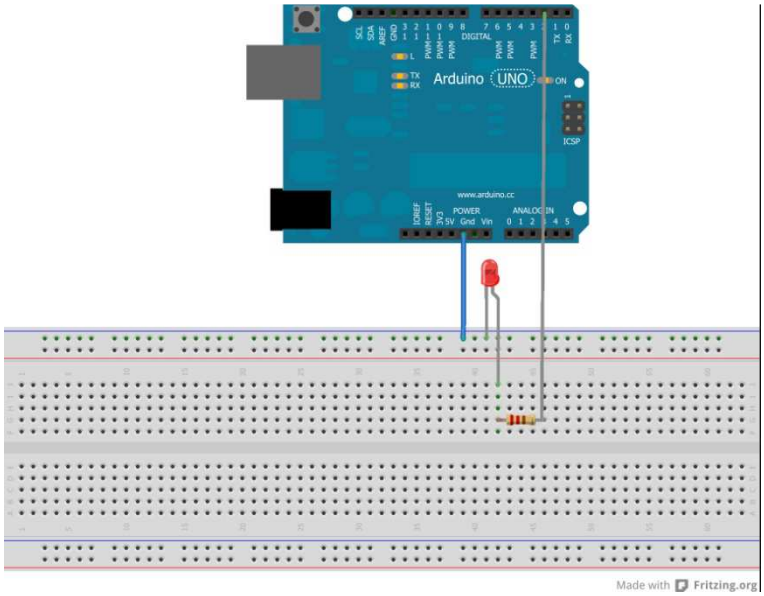


Ilustración 43 Esquemático Práctica 1 LabView

Al ser esta la primera práctica, nos detendremos en la configuración de Arduino que será la misma para cada proyecto. En nuestro proyecto en blanco, añadimos el bloque "init" con la siguiente configuración:

- Puerto de comunicación→ Nuestro caso el 14
- Velocidad de transmisión→ 115200 bps
- Tipo de tarjeta Arduino→ Arduino UNO
- Número de bits de paquetes de comunicación→15
- Tipo de puerto de comunicación→USB/Serial

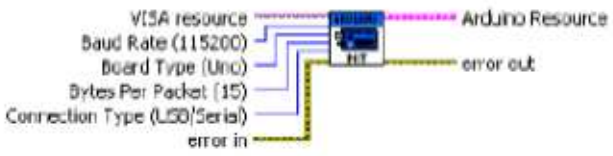


Ilustración 44 Bloque Init

Tras la creación del “init”, introducimos la estructura “while loop” que será la encargada de realizar las mismas funciones que la función “void loop” mencionada en la estructura de Arduino. Esta estructura se ejecutará hasta que cerremos el puerto de conexión. Es en esta estructura donde se introducirán todos los módulos de control de la tarjeta.

NOTA: Los pines 0 y 1 los utiliza LabView para comunicarse con la tarjeta Arduino.

Una vez tenemos configurado nuestro “while-loop”, veamos que bloques tenemos que añadir y que pin usaremos como salida. Para el ejercicio que nos ocupa conectaremos con la placa mediante el pin 8 mediante la función de escritura “Digital Write Pin”, quedando el diagrama de bloques en LabView de la siguiente forma:

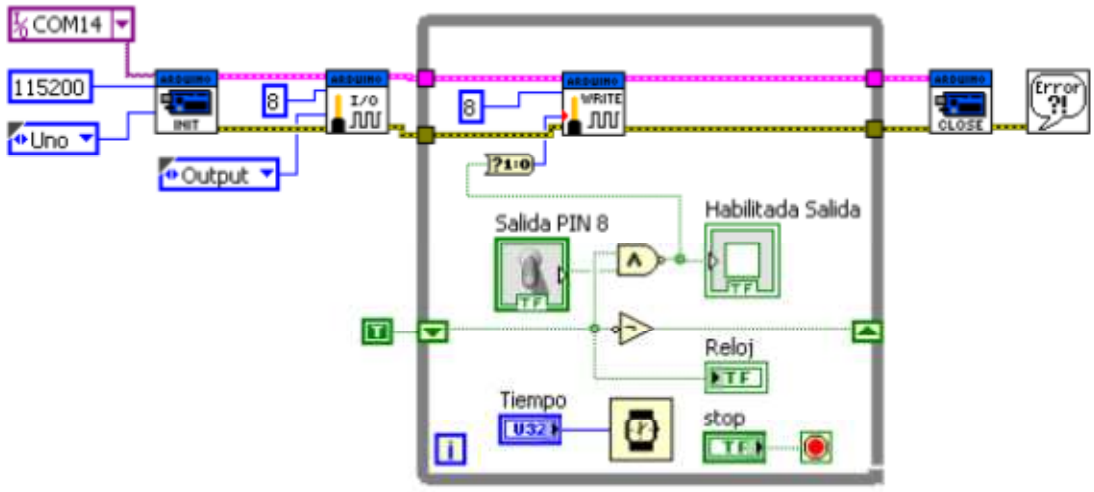


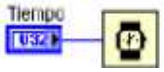
Ilustración 45 Diagrama de bloques Práctica 1

La secuencia de ejecución es:

1. Inicializamos y configuramos la conexión con Arduino con la velocidad por defecto de 115200 bps.
2. Configuramos el pin 8 como salida (OUTPUT)
3. Escribimos en el pin 8 la señal del reloj
4. Cerramos conexión con Arduino
5. Control de errores

Veamos a continuación el panel que hace de interfaz entre Arduino, LabView y nosotros:

Ilustración 46 Panel Práctica 1

Para controlar el tiempo de encendido y apagado del LED es necesario la implementación de un reloj. Ésta se lleva a cabo en el propio "while loop": añadimos un "shift register" con lo que conseguiremos que se ejecute cada cierto tiempo marcado por el usuario 

. Colocamos una "puerta AND" para poder habilitar mediante un interruptor (salida del pin 8) el tránsito de la señal de control del reloj al bloque "Digital Write Pin". También introducimos dos LEDs informativos, uno que informa si está habilitada la señal de reloj y otro para comprobar que se ha mandado la orden de escritura al pin 8.

Práctica 2. Control de un semáforo simple

En esta práctica simularemos el funcionamiento de un semáforo de la forma más sencilla posible, asignando nosotros un tiempo fijo para cada estado (rojo, ámbar y verde). El panel es bastante sencillo, mostrando los tres LEDs y un botón de parada:

Ilustración 47 Panel Práctica 2

El montaje eléctrico visto con Fritzing es el siguiente:

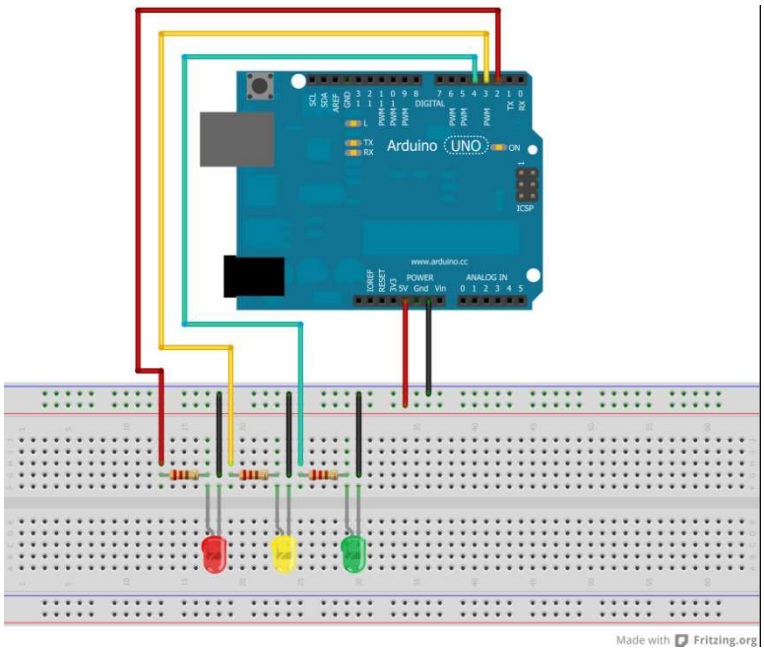


Ilustración 48 Esquemático Práctica 2 LabView

La asiganción de pines y tiempo a cada LED es la siguiente:
Rojo→Pin 8→Tiempo Activo→1000ms=1segundo
Verde→Pin 10→Tiempo Activo→1000ms=1segundo
Amarillo→Pin 9→Tiempo Activo→700ms=0.7segundos

El proceso de ejecución es el siguiente:

1. Inicializamos y configuramos la conexión con Arduino mediante el “init”
2. Configuramos los pines 8,9 y 10 como salidas mediante el bloque “Set Digital Pin Mode” de la librería Arduino
3. Generamos las tres señales (rojo, verde y ámbar)
4. Asignamos cada estado a cada caso de la estructura “Case Structure”
5. Escribimos el valor de cada señal (rojo, verde y ámbar) en sus pines correspondientes. Las salidas del secuencializador deben ser TRUE/FALSE para que se puedan escribir en el “Digital Write Pin”
6. Cerramos el puerto mediante el bloque “Close”
7. Tratamos los errores con “Simple Error”

El diagrama de bloques queda de la siguiente manera:

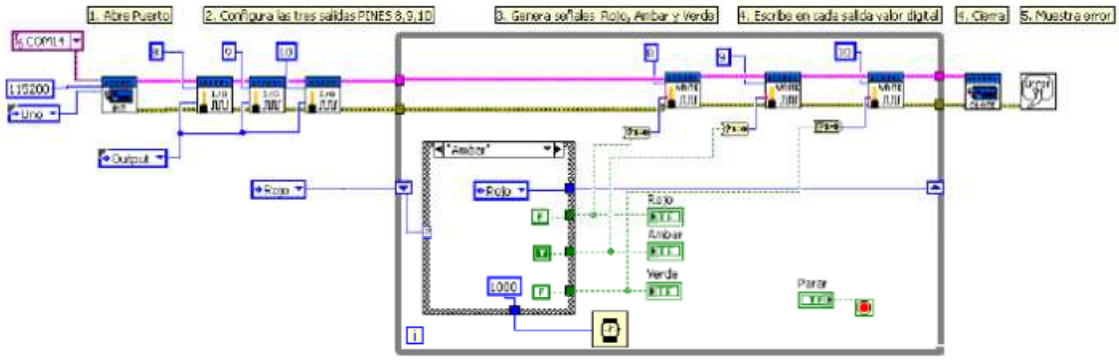


Ilustración 49 Diagrama de bloques Práctica 2

Comentamos a continuación cada uno de los tres casos creados para cada estado:

Estado Rojo: La secuencia será Rojo TRUE, ámbar FALSE y verde FALSE con tiempo 1000ms. El estado siguiente debe ser verde.

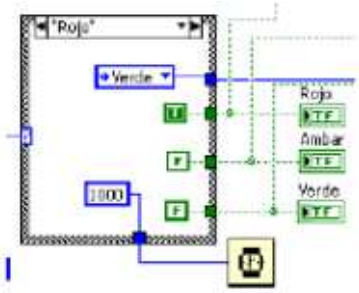


Ilustración 50 Estado rojo

Estado Ámbar: La secuencia será Rojo FALSE, ámbar TRUE y verde FALSE con tiempo 700 ms. El estado siguiente debe ser rojo.

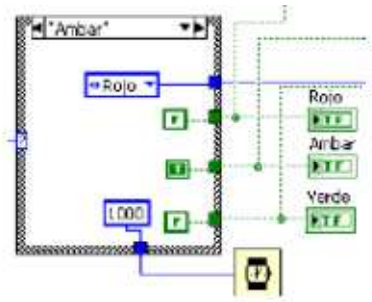


Ilustración 51 Estado Ámbar

Estado verde: La secuencia será Rojo FALSE, ámbar TRUE y verde FALSE con tiempo 700 ms. El estado siguiente debe ser ámbar.

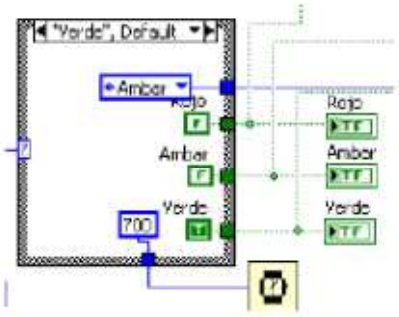


Ilustración 52 Estado Verde

Práctica 3. Control de un semáforo ajustable

En esta práctica vamos a mejorar el control semafórico introduciendo en ella la posibilidad de ajustar mediante el panel el tiempo de cada estado (rojo, verde y amarillo). El panel queda:

Ilustración 53 Panel Práctica 3

Tenemos de nuevo tres LEDs y un botón de parada; añadimos tres objetos de entrada de valor para ajustar los tiempos de cada estado.
El montaje eléctrico es el mismo que hemos visto en la práctica 2, por lo que omitimos su figura y explicación.
El diagrama de bloques general es el siguiente:

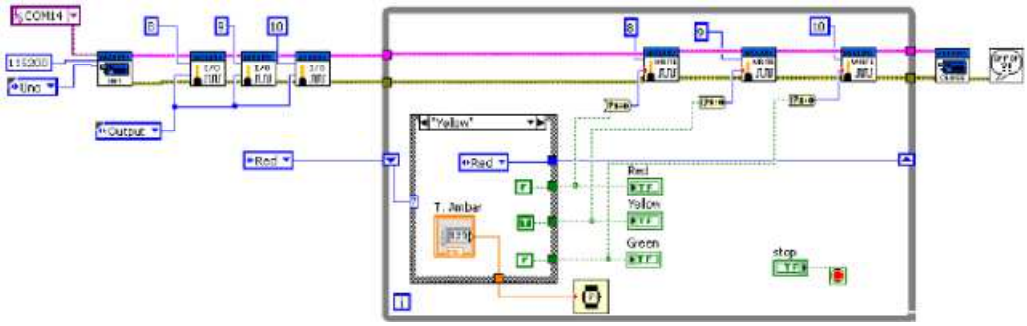


Ilustración 54 Diagrama de bloques Práctica 3

El añadido de lectura del valor de tiempo para cada estado se realiza dentro de cada estructura “Case”, como podemos observar en la siguiente figura:

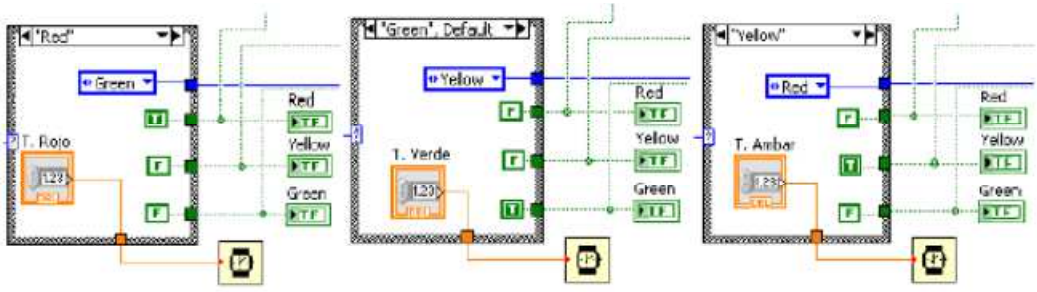


Ilustración 55 Estructura Case Práctica 3

Práctica 4. Control de dos servos

En esta práctica explicaremos como controlar la velocidad y sentido de giro de dos servomotores así como el ángulo girado.
El panel de control queda como se indica a continuación:

Ilustración 56 Panel Práctica 4

Podemos seleccionar si queremos controlar uno o dos servos, la velocidad y el ángulo de desplazamiento.
En esta práctica introduciremos cuatro nuevos tipos de bloques de la librería Arduino:

1. "Set Numbers of Servos" → Indicamos cuantos servos vamos a manejar. Mediante "Create Control" en el menú del nuevo bloque le indicamos que genere un número de tipo entero (dos en nuestro caso)

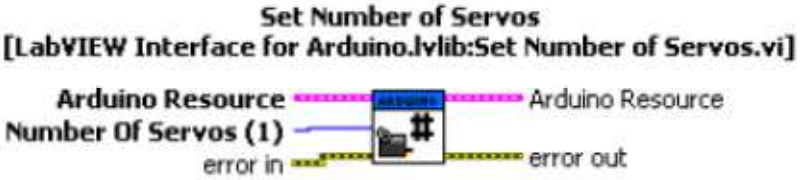


Ilustración 57 Bloque Set Number of Servos

2. "Configure Servo" → Con este bloque nombramos al servo y le asignamos un pin para su control. En nuestro caso necesitaremos dos bloques

Ilustración 58 Bloque Configure Servo

3. "Servo Write Angle" → Simplemente escribimos en el servo el ángulo que ha de girar en grados

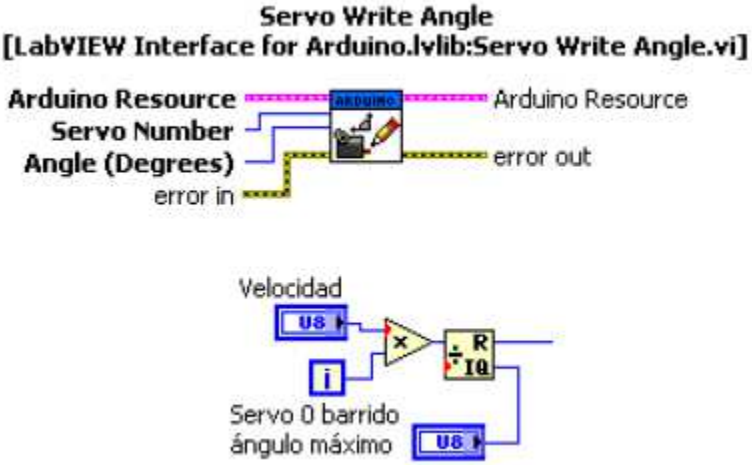


Ilustración 59 Bloque Servo Write Angle

4. "Servo Read Angle" → Nos indica la posición del servo y la sacamos con un instrumento analógico ("Servo 1" en nuestro caso)

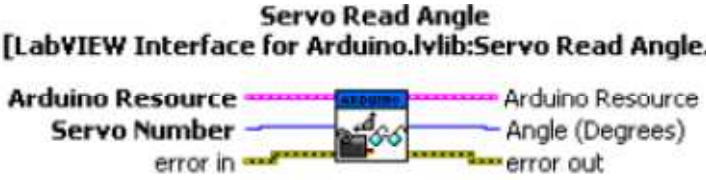


Ilustración 60 Bloque Servo Read Angle

El diagrama de bloques:

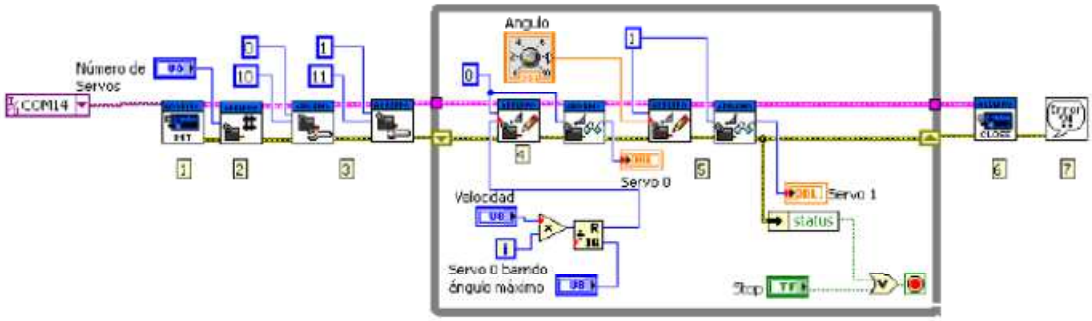


Ilustración 61 Diagrama de bloques Práctica 4

El proceso de ejecución es el siguiente:

1. Inicializamos y configuramos la conexión con Arduino mediante el “init”
2. Establecemos el número de servos a manejar mediante “Set Number of Servos”
3. Configuramos los dos servos asignando pines digitales para su control (pin 10 y pin 11)
4. Escribimos un ángulo de 0 basado en la repetición del bucle. Servo 0 nos barre desde los 0 grados hasta el ángulo que le asignemos y la repetición. Este ángulo también se lee desde el servo y se muestra en el panel
5. Ajustamos a mano el servo 2
6. Cerramos la conexión con Arduino
7. Tratamos los errores con “Simple Error”

El montaje eléctrico:

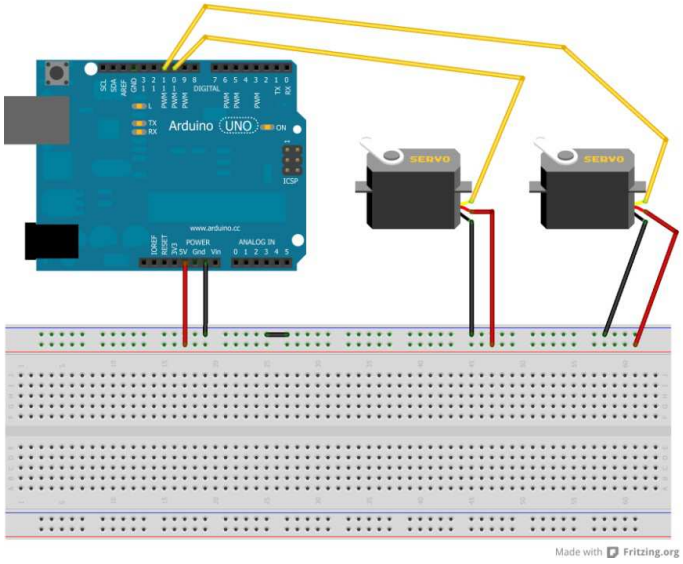


Ilustración 62 Esquemático Práctica 4 LabView

Práctica 5. Control de velocidad y sentido de giro de motor de continua

Con esta práctica vamos a controlar la velocidad y el sentido de giro de un motor de corriente continua, ayudados por el integrado L293D que ya comentamos en la práctica 11 de Arduino. Las conexiones entre tarjeta, motor e integrado son las mismas que en la práctica anteriormente referenciada, por lo que nos centraremos en el diagrama de bloques de LabView.
El panel lo simplificamos a un único mando y un botón de parada:

Ilustración 63 Panel Práctica 5

Veamos primero el diagrama de bloques y comentemos después el proceso de ejecución.

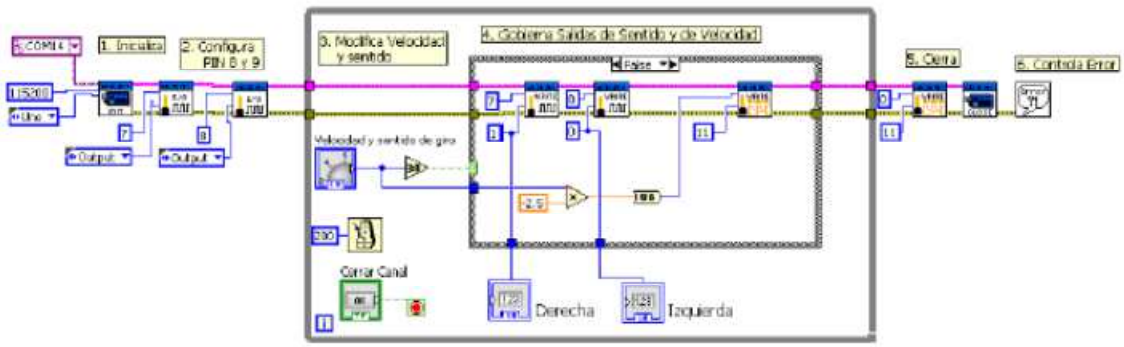


Ilustración 64 Diagrama de bloques Práctica 5

Proceso de ejecución:

1. Inicializamos y configuramos la conexión con Arduino mediante el "init"
2. Configuramos el pin 8 como salida para sentido DERECHA y el pin 9 para sentido IZQUIERDA
3. Dentro del bucle colocamos una "Case Structure" encargada de controlar el motor que ejecutaremos en intervalos de 200 ms. Tendremos dos casos, TRUE y FALSE. Si es TRUE el giro es hacia la izquierda, por lo que sacaremos los valores correspondientes a los pines 7 y 8 de Arduino y la velocidad quedará marcada por el valor del panel. Si es FALSE, girará a derechas e invertiremos los valores de los pines de salida 7 y 8. Veamos con una tabla los valores para cada caso:

GIRO	PIN ARDUINO	VALOR	PIN L293D INPUT 0	PIN L293D INPUT 0
Derechas	Pin 7 Pin 8	1 0	1	0

Tabla 4. Valores Pines Motor L293D caso FALSE

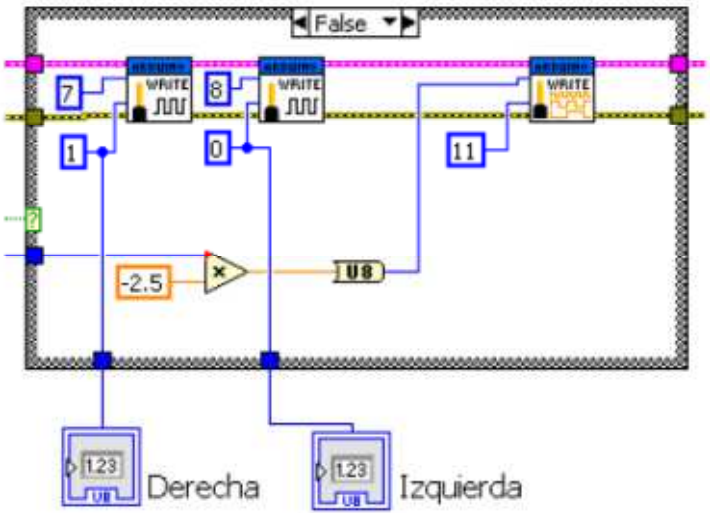


Ilustración 65 Case giro motor a derechas

GIRO	PIN ARDUINO	VALOR	PIN L293D INPUT 0	PIN L293D INPUT 0
Izquierdas	Pin 7 Pin 8	0 1	0	1

Tabla 5. Valores Pines Motor L293D caso TRUE

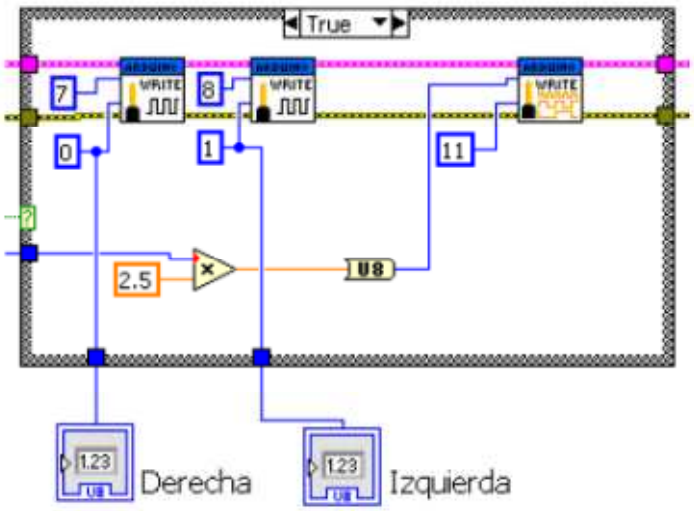


Ilustración 66 Case giro motor a izquierdas

4. Colocamos los indicadores de giro para saber por pantalla hacia donde gira el motor.
5. Detenemos el motor enviando un cero al pin 11 (salida PWM) y cerramos la conexión con Arduino
6. Tratamos los errores con “Simple Error”

El cambio en la “Case Structure” se lleva a cabo mediante un operador del tipo “Greater or Equal To 0”

Ilustración 67 Operador Greater or Equal to 0

Si recordamos, la velocidad de estos motores se escala desde -255 a 255, que es el valor máximo admitido por un pin PWM, por lo que interpolamos nuestra escala de -100 a 100 y multiplicamos por 2.5.

5 CAPÍTULO 5. ROBOT POLOLU 3π

5.1 Introducción

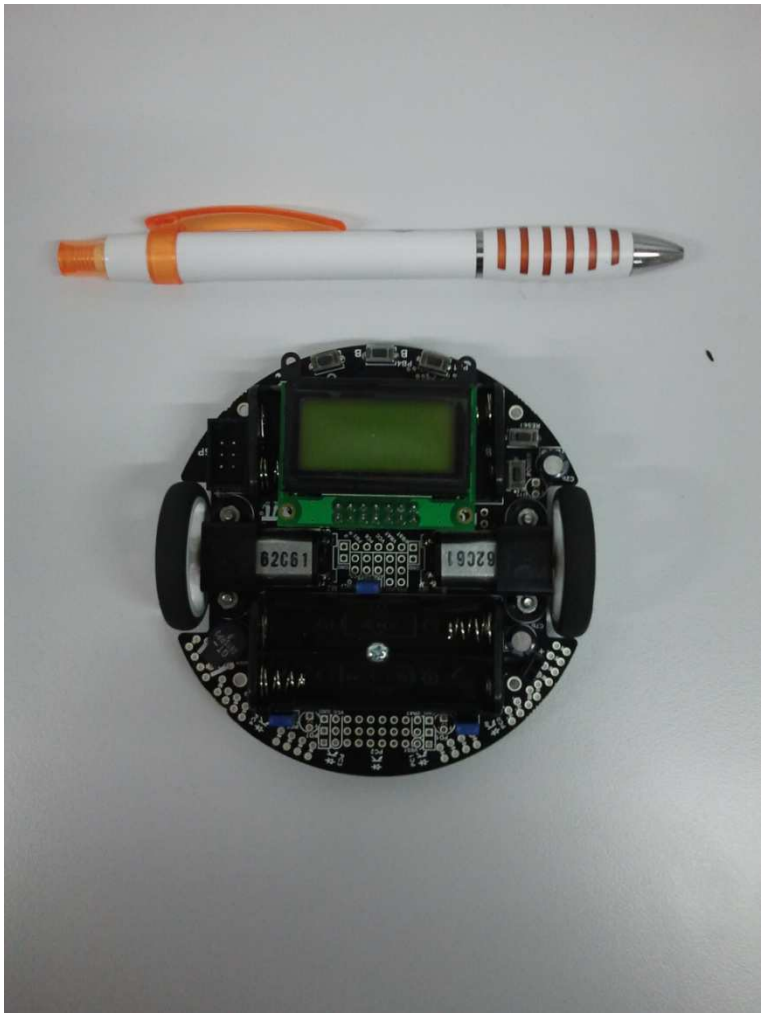


Ilustración 68 Robot Pololu 3pi

El robot Pololu 3π debe su nombre a su tamaño, ya que su diámetro es en centímetros el número pi multiplicado por tres. Se trata de un robot autónomo con motores duales, cinco sensores QTR de reflectancia, una pantalla LCD, un zumbador, cinco botones de control y se alimenta con 4 pilas AAA. El cerebro de este robot es un microcontrolador ATMega328 programable. El que disponga de este microcontrolador es una de las principales causas por la que elegimos este robot, ya que es totalmente compatible con Arduino.

Es un gran robot para principiantes, como es nuestro caso, y con un grandísimo abanico de opciones conforme vayamos descubriéndolo. Está diseñado principalmente como un seguidor de línea, otro motivo para nuestra elección, y resolución de laberintos.

Para su programación necesitamos un programador AVR ISP externo como el USB AVR Programmer.

Para más información, se puede consultar el *Anexo II "Pololu 3pi Robot User's Guide"*.

5.2 Programar nuestro 3π

El robot trae precargado un programa de demostración; pero nosotros hemos adquirido dos robots de este tipo para que simulen dos vehículos en una maqueta. Para ello, debemos programarlos como seguidores de línea.

También hemos adquirió un programador AVR ISP con el que pasaremos nuestro código al robot.

Necesitamos cierto software para llevar acabo nuestra tarea de programar como seguidor de línea nuestro Pololu 3pi:

- WinAVR, entorno de desarrollo para los microcontroladores de la familia AVR
- AVR Studio, paquete de desarrollo integrado de la casa Atmel con IDE que trabaja sin problemas con el compilador WinAVR's. AVR Studio incluye el software AVR ISP que nos permite cargar nuestros programas en el robot 3pi.

Navegando por la red hemos encontrado un archivo ejecutable que dispone de todo lo necesario para programar nuestro Pololu 3pi. El archivo pesa 164 megas y se adjunta en el DVD de la memoria.

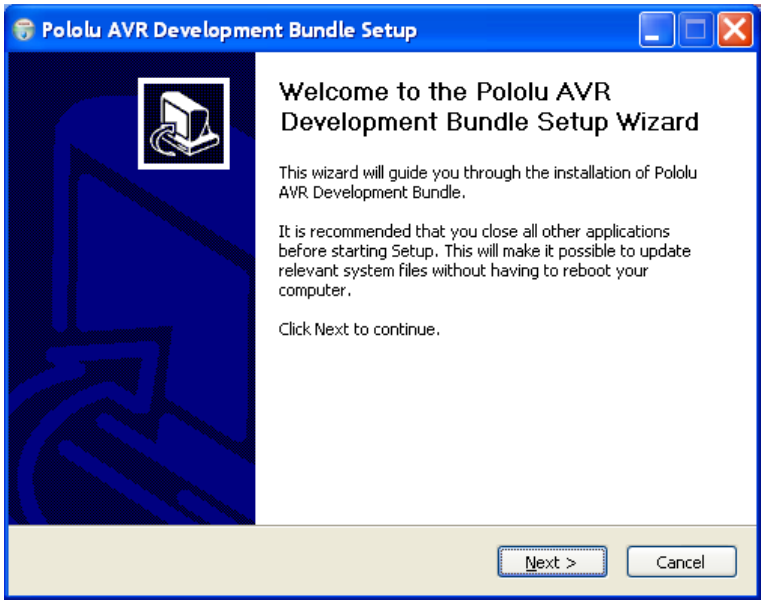


Ilustración 69 Instalación Pololu AVR

Iniciamos el ejecutable y seleccionamos las aplicaciones y librerías que queramos instalar:

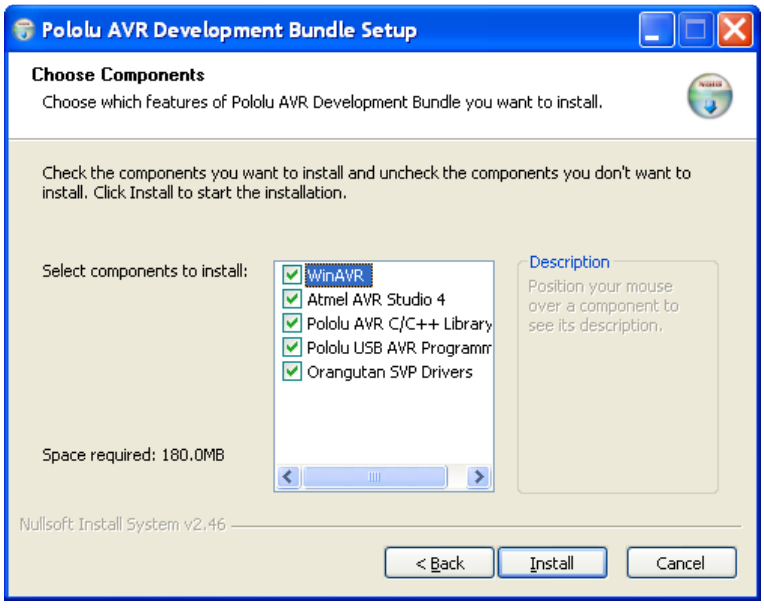


Ilustración 70 Autoejecutable AVR

Las instalamos todas, aunque podríamos prescindir de Orangutan SVP Drivers. Una vez instaladas, ya estamos listos para comenzar a programar el robot.

Abrimos el programa AVR Studio (nuestra versión es la 4):

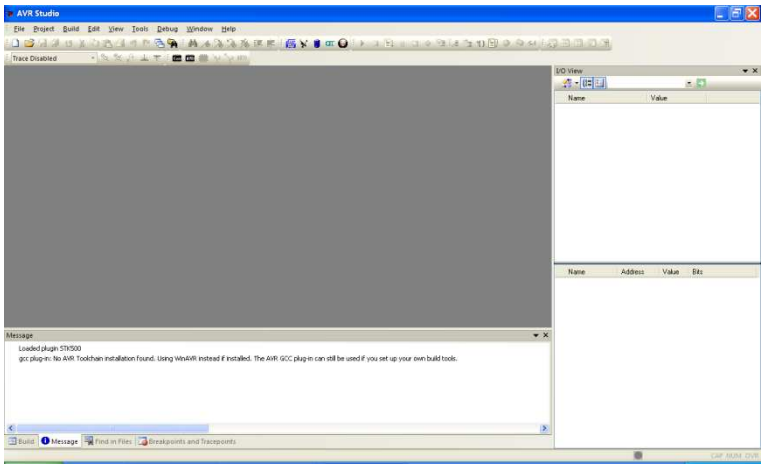


Ilustración 71 AVR Studio v4

Como hemos instalado las librerías de Pololu AVR C/C++, disponemos de numerosos ejemplos que podemos cargar en nuestro robot. Queremos que nuestro 3pi actúe como seguidor de línea, por lo que le vamos a cargar el programa ejemplo line-follower que se encuentra en C:\Libpololu-avr\Examples\ATMega328P\3pi-linefollower.

Una vez abierto, debemos seleccionar nuestro programador:

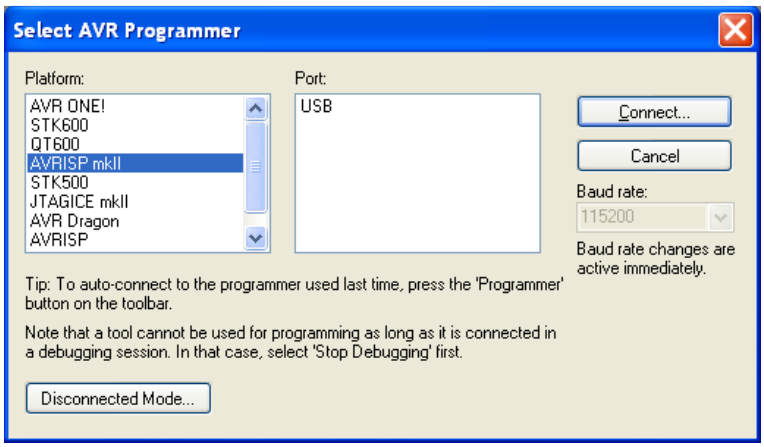


Ilustración 72 Selección Programador AVR-ISP

Conectamos el programador con el robot mediante el cable USB y procedemos a la carga del programa ejemplo.

Ilustración 73 Programador AVRISP

Con la carga de este programa, ya disponemos de un robot sigue-líneas que nos hará las veces de automóvil en nuestra maqueta final, aunque como se explicará en el siguiente capítulo, tendremos que modificar y adaptar el código para satisfacer nuestras necesidades.

Se adjuntan el *Anexo IV "Pololu USB AVR Programmer" y el Anexo V "AVR Programming Quick Start"*.

Aclaración: Este robot puede ser programado también mediante Arduino. Debido a una incompatibilidad del programador con la versión actual de Arduino (error de comunicación con el puerto serie virtual) tuvimos que buscar una solución que fue la expuesta anteriormente.

6 CAPÍTULO 6. MAQUETA

6.1 Introducción

El objetivo de crear esta maqueta es divulgar en alumnos de Secundaria, Bachillerato, Módulos Profesionales y universitarios las posibilidades que Arduino nos brinda. Hemos visto mediante 14 prácticas lo fácil e intuitivo que es trabajar con esta plataforma y queremos poner en práctica los conocimientos adquiridos. Para ello nos apoyaremos en los distintos programas explicados para ensamblarlos todos en un programa final que sea capaz de controlar nuestra maqueta.

Nuestra maqueta pretende simular el paso por un puente levadizo con su correspondiente control semafórico. Consta de 4 barreras con sus correspondientes semáforos, el puente levadizo y dos robots Pololu 3pi que harán de vehículos. Todo ello integrado en un tablero para su posible transporte y exposición.

Ilustración 74 Maqueta

6.2 Componentes principales

- **Puente levadizo**
  En la siguiente figura podemos observar el puente levadizo adquirido en Opitec.

Ilustración 75 Puente Levadizo

Obviamente no viene montado, lo que lo hace más divertido.

Ilustración 76 Paquete de Puente Levadizo

Este puente consta de un motor de continua que eleva y desciende el puente. Gracias a un interruptor, controlamos su subida o bajada y dos finales de carrera hacen que se pare en la posición deseada. Va alimentado con dos pilas de 1,5 Voltios. Dispone de un semáforo a cada lado que estarán en rojo cuando el puente esté en movimiento o elevado y en verde cuando los robots puedan pasar por él.

El tiempo de construcción del mismo fue de 12 horas y a continuación podemos ver un detalle de la caja de engranajes reductores donde se aloja el motor y el puente terminado:

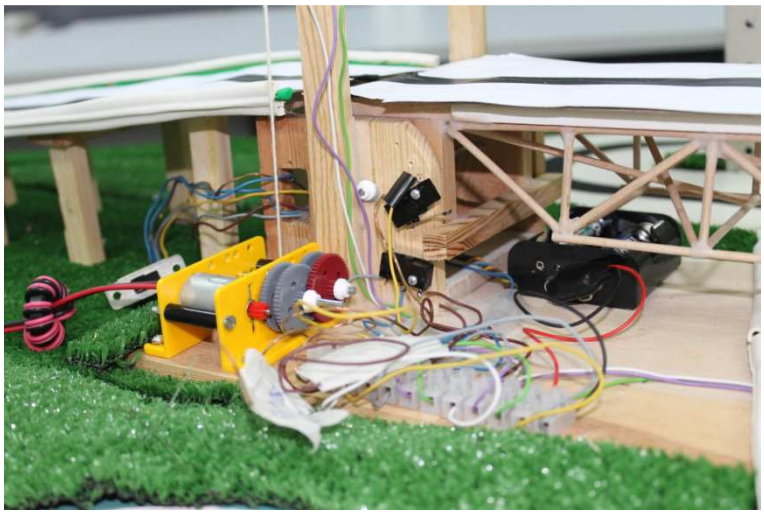


Ilustración 77 Caja Motor y Engranajes

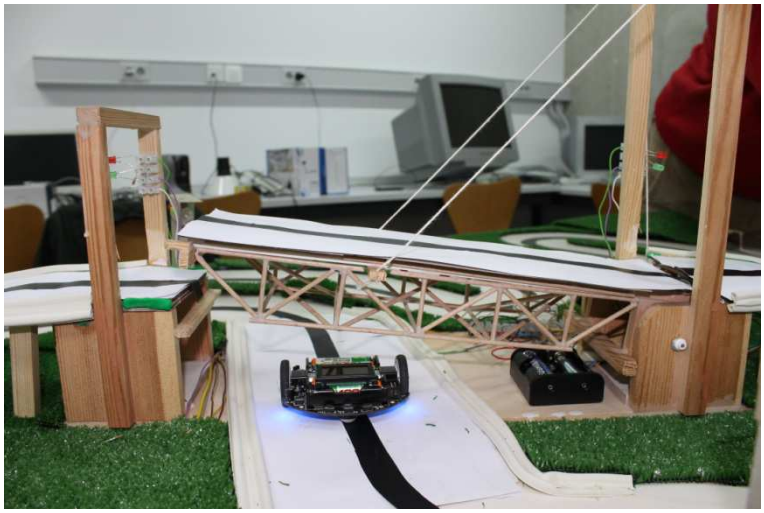


Ilustración 78 Pololu en maqueta bajo puente

Nota 1: Dirección Web:http://es.opitec.com/opitec-web/articleNumber/105456/kshp/p/2
Nota 2: Debido al tamaño de nuestros robots, tuvimos que modificar el paso del puente, ensanchándolo hasta obtener 12 centímetros.

Se adjunta también el *Anexo VI "Manual de montaje puente levadizo"* con todas las instrucciones de montaje y conexiones eléctricas.

- **Tablero**

Conseguimos en recuperación de la UPCT un tablero verde ideal para nuestra maqueta que tuvimos que cortar para que fuera posible su posterior transporte. Sus medidas finales son 100x180 centímetros. Pintamos de blanco la "carretera" por donde pasarán nuestros robots para mejorar el rendimiento de los cinco sensores QTR a la hora de seguir la línea negra (cinta aislante).

Ilustración 79 Tablero recién pintado

Cubrimos todos los huecos con césped artificial para mejorar el aspecto.

Ilustración 80 Maqueta II

- **Rampas de acceso al puente**

Estos dos elementos han sido con diferencia los que más problemas nos han ocasionado. Primero se optó por utilizar arcilla para crear una estructura y sobre ella colocar “eva” pero la estructura se agrietó y tuvimos que pensar otra solución.

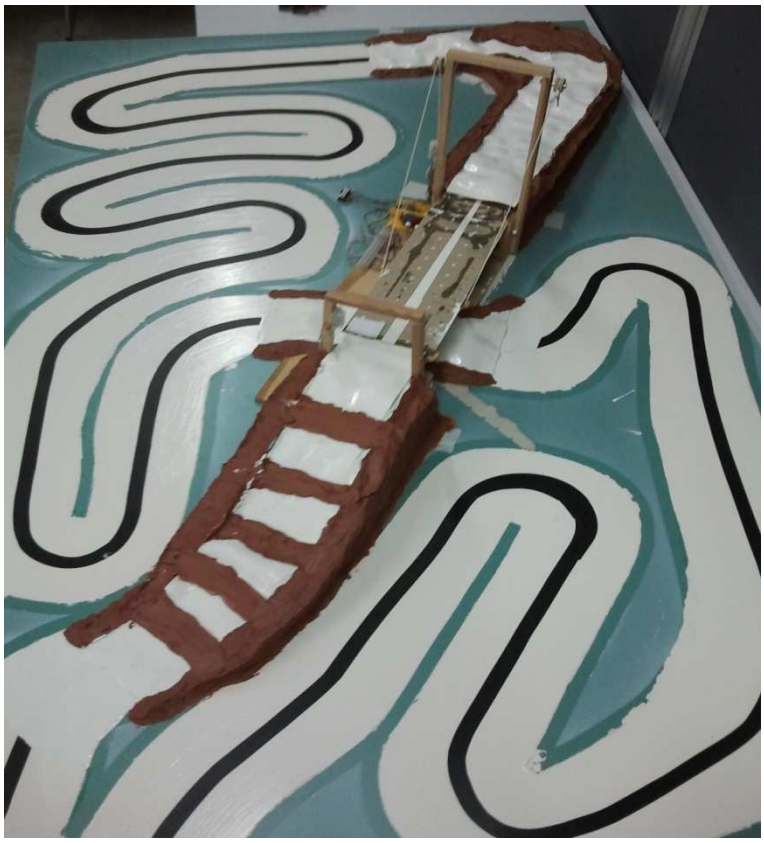


Ilustración 81 Rampa Acceso Arcilla

Finalmente optamos por construir pórticos escalados a una cierta distancia que nos sirvieran como estructura y unirlos con varillas para ganar estabilidad.

Ilustración 82 Pórticos Rampas Acceso

- **Semáforos**

Los cuatro semáforos están construidos con piezas TechCard, de cartón pretroqueladas, recortadas y premarcadas que son muy fáciles de manejar. Vienen a ser como las estructuras Meccano pero más económicas.

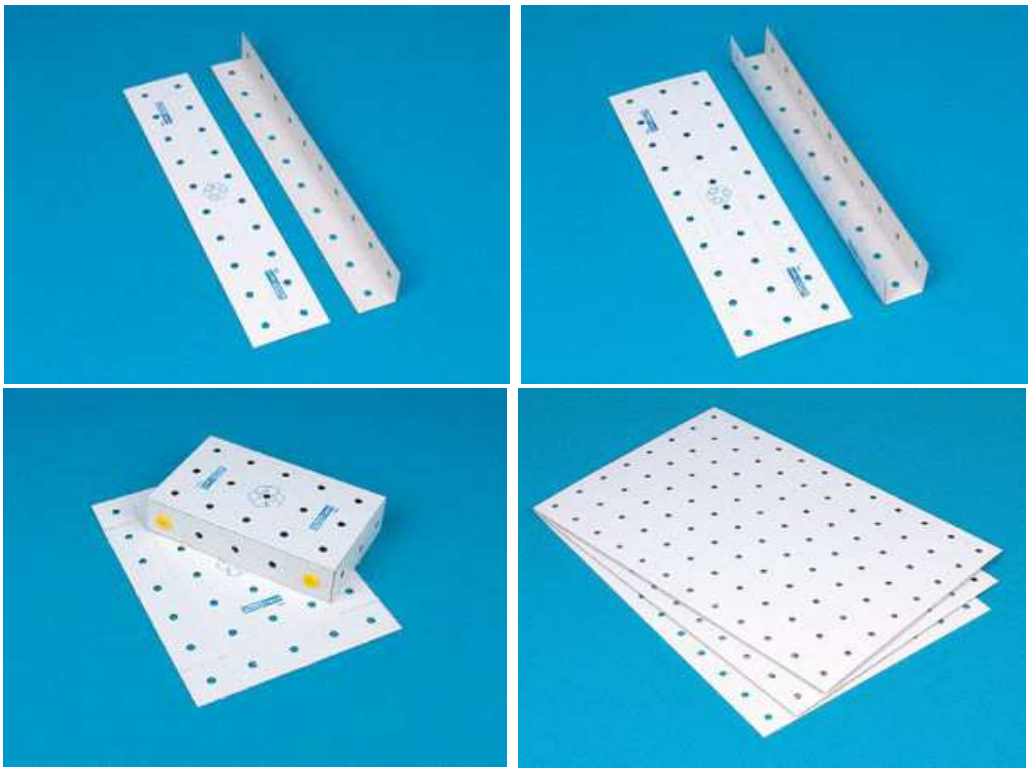


Ilustración 83 TechCards

En tres de sus agujeros colocamos los tres leds (rojo, verde y amarillo) sacando las conexiones de cada uno. De cada semáforo saldrán cuatro conexiones, las tres señales de control de cada led y la masa de todos.

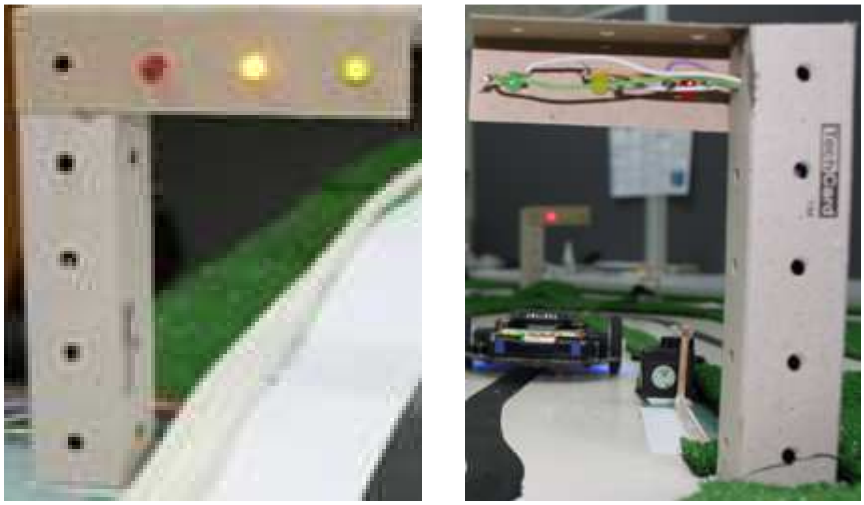


Ilustración 84 Semáforos

En el *Anexo VII "TechCard"* se explica su uso y encontraremos diversas plantillas de simpáticos proyectos.

- **Barreras**

Nuestras cuatro barreras están modeladas por un servomotor al que le pegamos una cartulina que será la encargada de "tapar" la línea y hacer que el robot pare hasta que de nuevo le "enseñe" la línea.

Ilustración 85 Barreras I

En las barreras de paso por debajo del puente, pondremos dos tiras blancas antes de las barreras para evitar que el robot quede girado a la hora de llegar a la zona de parada; ya que al estar en línea recta, cuando ve la barrera, tendía a girar hacia el servo.

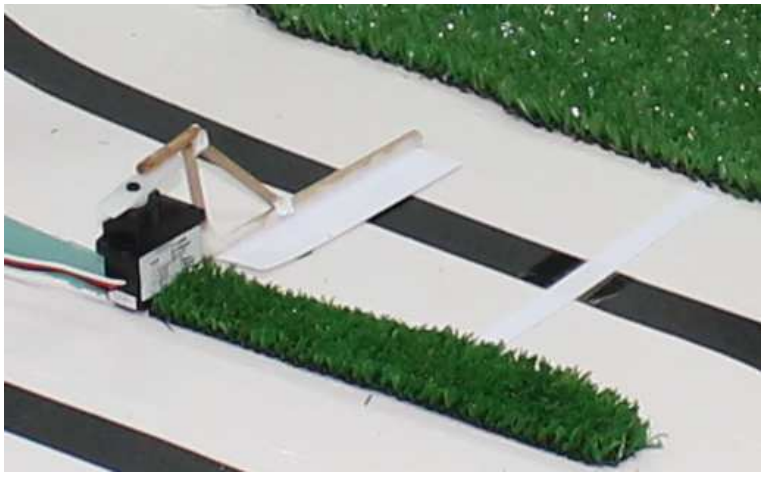


Ilustración 86 Detalle Barrera

6.3 Programas Arduino

6.3.1 Semáforos programados por tiempo

Tomando como base el código de la práctica 13, simplemente añadimos una pausa entre el cambio de encendido de luz para controlar el tiempo en el que queramos que el semáforo esté en verde, en transición (amarillo) o en rojo.

```
int pinVerde =7;
int pinAmarillo =8;
int pinRojo =12;
void setup()
{
pinMode(pinVerde, OUTPUT);
pinMode(pinAmarillo, OUTPUT);
pinMode(pinRojo, OUTPUT);
Serial.begin(9600);
}
void loop()
{
verd();
amar();
rojo();
}
void verd()
{
digitalWrite(pinVerde, HIGH);
delay(10000);
digitalWrite(pinVerde, LOW);
}
void amar()
{
digitalWrite(pinAmarillo, HIGH);
delay(1000);
digitalWrite(pinAmarillo, LOW);
}
void rojo()
{
digitalWrite(pinRojo, HIGH);
delay(10000);
digitalWrite(pinRojo, LOW);
}
```

Como podemos observar en el código, damos 10 segundos de semáforo en verde, un segundo en amarillo para el cambio a rojo que durará otros 10 segundos.

El circuito montado físicamente queda así:

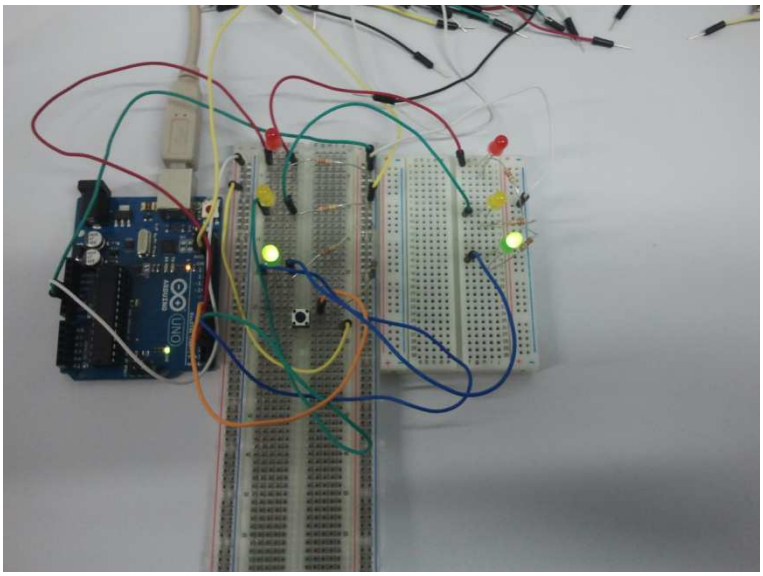


Ilustración 87 Montaje Prueba Semáforo

Realizaremos un montaje en paralelo e invirtiendo verde y rojo para controlar los cuatro semáforos con tres pines para optimizar recursos e introducir menos corrientes parásitas.

6.3.2 Motor programado por tiempo con cambio de sentido

De nuevo nos apoyamos en una práctica anterior; esta vez en la práctica 11, para programar el motor de continua que elevará y descenderá el puente.

```
int botonPin = 2;
int motorPin1 = 3;
int motorPin2 = 4;
int speedPin = 9;
void setup() {
pinMode(botonPin, INPUT);
pinMode(motorPin1 , OUTPUT);
pinMode(motorPin2 , OUTPUT);
pinMode(speedPin, OUTPUT);
digitalWrite(speedPin, HIGH); // Motor on desde principio
}
void loop() {
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, HIGH);
delay(10000);
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, LOW);
delay(15000);
digitalWrite(motorPin1, HIGH);
digitalWrite(motorPin2, LOW);
delay(8000);
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, LOW);
delay(12000);
}
```

Analizando muy por encima el código, veremos cómo hemos programado un ciclo completo de subida y bajada del puente que se repetirá mientras la placa de Arduino esté conectada. En cuanto alimentemos el circuito, el puente se estará elevando durante 10 segundos; una vez transcurridos se mantendrá abierto durante 15 segundos; comenzará a descender durante 8 segundos (hay que tener en cuenta que al motor le cuesta menos bajar que subirlo) y permanecerá cerrado 12 segundos para que se pueda circular a través suyo.

El montaje en la protoboard se muestra a continuación:

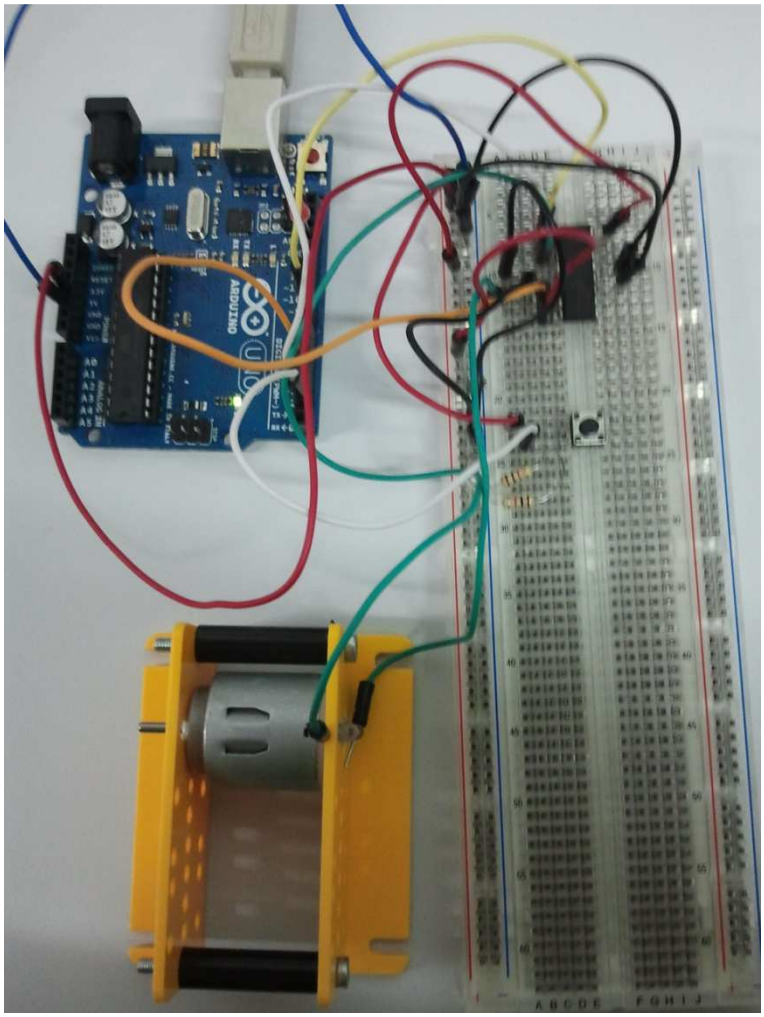


Ilustración 88 Montaje Motor Puente

6.3.3 Servomotores (barreras) programadas por tiempo

En este caso manejaremos dos a dos las barreras para que cuando unas se abran para dejar paso, las opuestas se cierren. El método de control es el mismo que en los programas anteriores: abrimos unas y cerramos otras, esperamos 8 segundos, cerramos y abrimos y volvemos a esperar otros 8 segundos. De nuevo este proceso se repetirá mientras el circuito esté alimentado.

```
#include <Servo.h>
Servo miServoPuente;
Servo miServoPaso;
int valor;
void setup() {
miServoPuente.attach(6);
miServoPuente.write(0);
miServoPaso.attach(5);
miServoPaso.write(90);
}
void loop() {
miServoPuente.write(90);
miServoPaso.write(0);
delay(8000);
miServoPuente.write(0);
miServoPaso.write(90);
delay(8000);
}
```

6.3.4 Programa final

Si sincronizamos y juntamos los tres programas anteriores y asignamos correctamente los pines a cada elemento, obtendremos el programa final que será el encargado del control de nuestra maqueta.

```
#include <Servo.h>
Servo miServoPuente;
Servo miServoPaso;
int motorPin1 = 3;
int motorPin2 = 4;
int speedPin = 9;
int pinVerde =7;
int pinAmarillo =8;
int pinRojo =12;
void setup() {
pinMode(motorPin1 , OUTPUT);
pinMode(motorPin2 , OUTPUT);
pinMode(speedPin, OUTPUT);
digitalWrite(speedPin, HIGH);
pinMode(pinVerde, OUTPUT);
pinMode(pinAmarillo, OUTPUT);
pinMode(pinRojo, OUTPUT);
Serial.begin(9600);
miServoPuente.attach(6);
miServoPuente.write(0);
miServoPaso.attach(5);
miServoPaso.write(90);
}
void loop() {
//ABRIENDO PUENTE
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, HIGH);
miServoPuente.write(90);
miServoPaso.write(0);
//Verde al paso X DEBAJO
digitalWrite(pinRojo, LOW);
digitalWrite(pinVerde, HIGH);
digitalWrite(pinAmarillo, HIGH);
delay(5000);
```

```
//PUENTE ABIERTO
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, LOW);
miServoPuente.write(90);
miServoPaso.write(0);
digitalWrite(pinRojo, LOW);
digitalWrite(pinAmarillo, LOW);
digitalWrite(pinVerde, HIGH);
delay(18000);
//CERRANDO PUENTE
digitalWrite(motorPin1, HIGH);
digitalWrite(motorPin2, LOW);
miServoPuente.write(0);
miServoPaso.write(90);
//Rojo AL PASO X DEBAJO
digitalWrite(pinRojo, HIGH);
digitalWrite(pinVerde, LOW);
digitalWrite(pinAmarillo, HIGH);
delay(4500);
//Puente Cerrado
digitalWrite(motorPin1, LOW);
digitalWrite(motorPin2, LOW);
miServoPuente.write(0);
miServoPaso.write(90);
digitalWrite(pinRojo, HIGH);
digitalWrite(pinVerde, LOW);
digitalWrite(pinAmarillo, LOW);
delay(18000);
}
```

6.4 Programa Pololu 3pi como sigue-línea

```
// Incluimos la librería pololu3pi
#include <pololu/3pi.h>
//Incluimos los archivos que nos permitirán guardar los datos en el ATmega
#include <avr/pgmspace.h>
//Mensaje de bienvenida que mostrará el robo en su pantallita
const char welcome_line1[] PROGMEM = " Riky";
const char welcome_line2[] PROGMEM = "Robot";
const char demo_name_line1[] PROGMEM = "Sigue-Linea";
const char demo_name_line2[] PROGMEM = "P1e";
// Reproducimos un par de tonos  almacenados
const char welcome[] PROGMEM = ">g32>>c32";
const char go[] PROGMEM = "L16 cdegreg4";
//Barras gráficas
const char levels[] PROGMEM = {
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b00000,
        0b11111,
        0b11111,
        0b11111,
        0b11111,
        0b11111,
        0b11111,
        0b11111
};
//Cargamos las barras en el LCD
void load_custom_characters()
{
        lcd_load_custom_character(levels+0,0); // una barra
        lcd_load_custom_character(levels+1,1); // dos barras
        lcd_load_custom_character(levels+2,2); // etc...
        lcd_load_custom_character(levels+3,3);
        lcd_load_custom_character(levels+4,4);
        lcd_load_custom_character(levels+5,5);
        lcd_load_custom_character(levels+6,6);
        clear(); // Limpiamos el LCD
}
```

```
//Muestra la lectura de los cinco sensores mediante barras gráficas
void display_readings(const unsigned int *calibrated_values)
{
        unsigned char i;
        for(i=0;i<5;i++) {
                const char display_characters[10] = {' ',0,0,1,2,3,4,5,6,255};
                char c = display_characters[calibrated_values[i]/101];
                print_character(c);
        }
}
// Iniciamos el robot con el mensaje de bienvenida, calibramos y hacemos sonar la melodía void
initialize()
{
        unsigned int counter; // cuenta atrás simple
        unsigned int sensors[5]; // guardamos los valores de los sensores en un array
        // This must be called at the beginning of 3pi code, to set up the
        // sensors.  We use a value of 2000 for the timeout, which
        // corresponds to
//Set up de los sensores  ->2000*0.4 us = 0.8 ms en nuestro procesador de 20 MHz
        pololu_3pi_init(2000);
        load_custom_characters(); // Cargamos
        // Suena la melodía de bienvenida y muestra un mensaje
        print_from_program_space(welcome_line1);
        lcd_goto_xy(0,1);
        print_from_program_space(welcome_line2);
        play_from_program_space(welcome);
        delay_ms(1000);
        clear();
        print_from_program_space(demo_name_line1);
        lcd_goto_xy(0,1);
        print_from_program_space(demo_name_line2);
        delay_ms(1000);
        // Muestra el nivel de la bacteria y espera que pulsemos el botón B
        while(!button_is_pressed(BUTTON_B))
        {
                int bat = read_battery_millivolts();
                clear();
                print_long(bat);
                print("mV");
                lcd_goto_xy(0,1);
                print("PULSE B");
                delay_ms(100);
        }
```

```
// Esperamos siempre a que se pulse el botón B para que el robot empiece a moverse
        wait_for_button_release(BUTTON_B);
        delay_ms(1000);
        //Se autocalibran los sensores girando sobre su propio eje a derechas e izquierdas
        for(counter=0;counter<80;counter++)
        {
                if(counter < 20 || counter >= 60)
                        set_motors(40,-40);
                else
                        set_motors(-40,40);
                // Grabamos la posición de máximo y mínimo percibida por los sensores
                //Encendemos los emisores de infrarrojos
                calibrate_line_sensors(IR_EMITTERS_ON);
                delay_ms(20);
        }
        set_motors(0,0);
        // Muestra los valores de calibración en barra gráfica
        while(!button_is_pressed(BUTTON_B))
        {
                // Se leen los valores tomados en la posición
                unsigned int position = read_line(sensors,IR_EMITTERS_ON);
//Mostramos la posición medida, que va desde 0 (el sensor más a la izquierda está
//encima de la línea) hasta 4000 (el sensor más a la derecha está sobre la línea)
                clear();
                print_long(position);
                lcd_goto_xy(0,1);
                display_readings(sensors);
                delay_ms(100);
        }
        wait_for_button_release(BUTTON_B);
        clear();
        print("Vámonos!");
        //Suena una musiquita, muestra el mensaje y arranca el robot
        play_from_program_space(go);
        while(is_playing());
}
```

```
// Función main
int main()
{
	unsigned int sensors[5]; // Array donde guardaremos los valores de los sensores
	// Inicializamos el robot
	initialize();
	// Función "main loop" que siempre se ejecuta
	while(1)
	{
		// Obtiene la posición de la línea (todos los sensores)
		unsigned int position = read_line(sensors,IR_EMITTERS_ON);
		if(position < 1000)
		{
			// Está lejos la parte derecha de la línea -> Gira a la izquierda
			// Ponemos el motor a derecho a 100 y el izquierdo a 0 para girar
			//El valor máximo de velocidad del motor es 255
			set_motors(0,100); //0,100
// Simplemente por diversión, indica mediante los Leds azules que dirección está tomando
			left_led(1);
			right_led(0);
		}
		else if(position < 3000)
		{
	// Si está  centrado en la línea, corre más y recto y enciende los dos Leds azules
			set_motors(70,70);
			left_led(1);
			right_led(1);
		}
//Hacemos que pare cuando encuentre la barrera (cartulina)
//Jugando con los valores tomados por los sensores llegamos a la conclusión de que estos
//son los óptimos para su óptimo control
//Paramos el robot y apagamos los Leds
		else if(position > 3700)
		{
			set_motors(0,0);
			left_led(0);
			right_led(0);
		}
		//hasta aquí el control de parada
		else
		{
		//Si estamos lejos de la parte izquierda de la línea, gira a la derecha
			set_motors(100,0); //100,0
			left_led(0);
			right_led(1);
		}}
```

6.5 Líneas futuras de la maqueta

Una vez que disponemos de la maqueta, quedan abiertas innumerables mejoras aplicables: incorporación de un elevador, uso de sensores para sincronización inteligente (barreras ópticas), control telemático mediante el módulo XBee, introducción de cámaras IP, etc…

Comentamos en el capítulo 5 la compatibilidad de Pololu 3pi con Arduino (puede programarse directamente a través de su placa) y como ya habíamos realizado el código, lo exponemos en el *Anexo VIII "Código Pololu 3pi sigue-línea Arduino"* para su estudio y desarrollo.

7 Bibliografía

http://www.Arduino.cc (Entorno Arduino)

http://www.freeduino.org/ (Entorno Arduino)

http://www.ladyada.net/learn/lcd/charlcd.html (Componentes y robótica en general)

http://www.hispavila.com/3ds/atmega/pulsadores.html (Componentes y robótica en general)

http://www.Pololu.com (Componentes y robótica en general)

http://www.adafruit.com/forums/viewtopic.php?f=8&t=14423 (Foro entorno Arduino)

http://www.arduinobot.pbworks.com/w/page/10175779/Motores-DC (Componentes Electrónicos)

http://www.wikipedia.org/ (Enciclopedia electrónica)

http://www.atmel.info/dyn/resources/prod_documents/8271S.pdf

http://es.wikipedia.org/wiki/AVR

http://www.arduino.cc/es/

http://es.wikipedia.org/wiki/Arduino

http://www.multiplo.org/duinos/wiki/index.php?title=Main_Page

http://blog.bricogeek.com/noticias/arduino/duinos---sistema---operativo---multitarea---para---arduino/

http://www.henningkarlsen.com/electronics/a_1_ds1302.php