Introducción

En días anteriores estuvimos viendo cómo con diferentes sensores podíamos encender leds, tanto progresivamente como de modo digital (on/off). En éste día pretendemos que dichos sensores, junto con algo de programa, nos den una idea del entorno y accionen otros elementos externos o actuadores.

Así vamos a trabajar hoy con sensores de infrarrojos, de ultrasonidos y los mezclaremos con servos de rotación angular y algún que otro elemento.

Los materiales que vamos a utilizar en las prácticas de hoy son los siguientes:

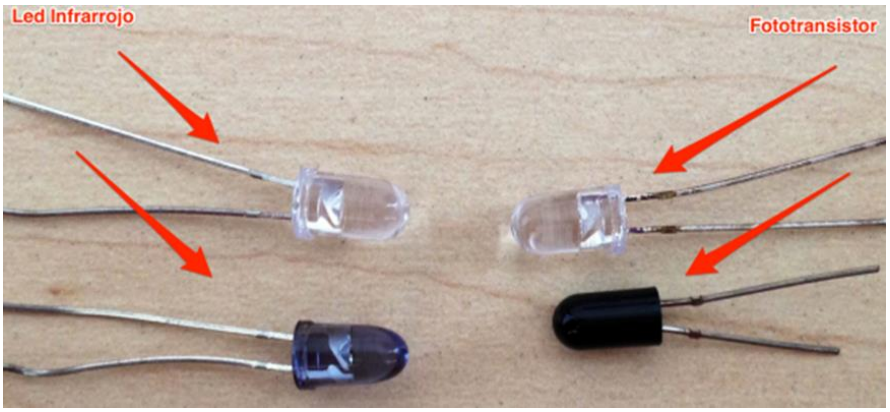
- Placa board
- Placa Arduino uno y shield arduino.
- Sensor de infrarrojos.
- Sensor de ultrasonidos.
- Servomotor.
- Diodos led.
- Resistencias e diversos valores.
- Display
- Decodificador 7447 BCD a 7 segmentos.
- Motor DC
- Transistor NPN
- Diodo
- Condensador 1mF.
- Placa puente en H
- Laca de dos relés

Como funciones de programación utilizaremos, además de las ya conocidas, otras funciones de control o lectura/escritura de datos de los nuevos elementos a utilizar.

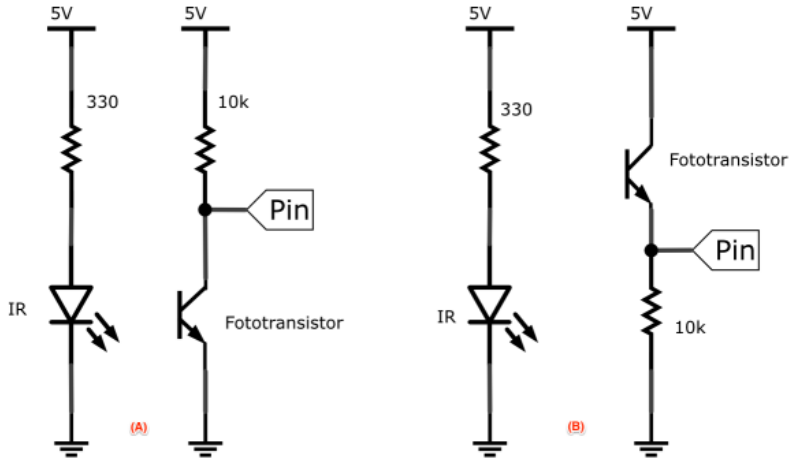
Práctica primera, barrera infrarroja.

Para elaborar una barrera infrarroja necesitaremos como elemento fundamental un sensor infrarrojo, en nuestro caso tendrá la función de barrera, pero se pueden utilizar en una gran variedad de ellas, aproximación, interrupción, etc.

La construcción es básica, por ello, en el mercado existen infinidad de presentaciones, ambos componentes (emisor/receptor) por separado, en un mismo soporte, etc. Siendo la más básica el montaje en serie de uno de estos diodos emisores infrarrojos con una resistencia limitadora y en la parte del receptor, este fototransistor y otra resistencia. La disposición de estos más cercano a la alimentación o a la masa nos determinará la salida de señal a nivel alto o bajo del detector infrarrojo.



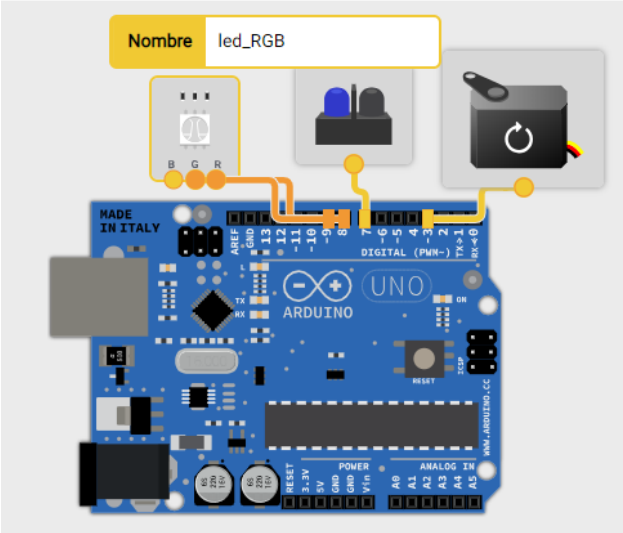
Generalmente, los emisores son diodos leds, con lo cual necesitarán una resistencia limitadora para adecuar las tensiones de trabajo y los detectores son transistores que habrá que polarizar dependiendo si queremos atacar Arduino con una entrada a nivel alto o bajo, sabiendo que estos transistores tienen colector y emisor



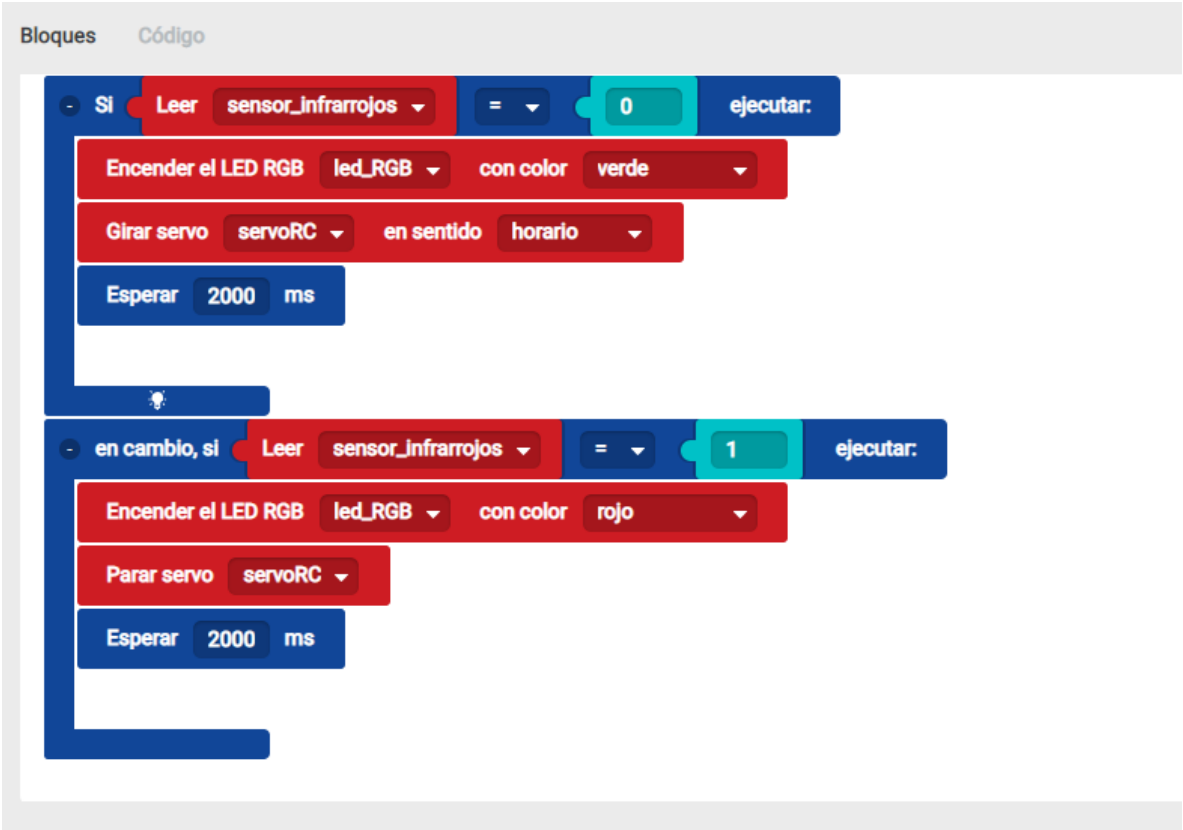
pero la base es la parte sensora de luz, así que el transistor estará en corte cuando no tenga luz infrarroja activándolo y entrará en saturación (conducción) cuando incida sobre él la luz, infrarroja por ende. En el esquema, la primera configuración da un nivel alto estando en reposo y un nivel bajo cuando sea excitado por la luz, al activarse y dejar la unión colector-emisor a tensión cero o casi cero. En la segunda configuración, el receptor está a nivel cero y pone la salida a nivel uno cuando es excitado. Recordad el segundo día cuando vimos los pulsadores, si os fijáis, los esquemas son muy similares si sustituimos el transistor por un pulsador.

Tengo que avisar que esta configuración necesita una cierta proximidad entre emisor y receptor, además de un cierto nivel de intensidad lumínica por parte del emisor. Considerando que el conjunto está un tanto apartado de la luz solar que podría activarlo. Si se cumplen estor parámetros, el detector podría conectarse en las entradas digitales de Arduino, sin embargo, cualquier imprevisto podría hacer que el sensor funcione de forma analógica, en este caso, y para evitar posibles problemas, podríamos conectarlo a una entrada analógica y programarlo de modo que si esta entrada tiene un nivel superior(o inferior, según se necesite) a un determinado valor de referencia, la tome como nivel alto o nivel bajo y obre en consecuencia.

El ejercicio que vamos a realizar consistirá en la programación de una barrera con un led emisor, que estará encendido continuamente, un receptor y en este caso un servo de movimiento continuo. El programa consistirá en que mientras no haya nada que interfiera en la barrera, el servo estará girando, cuando se corte la barrera y mientras esté interrumpida, el servo estará parado. (barrera de la cinta de la caja de un supermercado). En mi caso he añadido un diodo led RGB de modo que, cuando esté funcionando el motor se encienda el led en color verde y cuando sd para el motor se ponga rojo.



Un ejemplo de conexiones en la placa arduino podría ser el de la figura de la derecha. El código en bitbloq es el siguiente:



Como otras veces, el código para el IDE, sería:

```
/***  Included libraries  ***/
#include <BitbloqRGB.h>
#include <Servo.h>

/***  Global variables and function definition  ***/
ZumRGB led_RGB(8, 9, null);
const int sensor_infrarrojos = 7;
```

```
Servo servoRC;

/***  Setup  ***/
void setup() {
      pinMode(sensor_infrarrojos, INPUT);
      servoRC.attach(3);

}

/***  Loop  ***/
void loop() {
      if (digitalRead(sensor_infrarrojos) == 0) {
            led_RGB.setRGBcolor(0, 255, 0);
            servoRC.write(180);
            delay(2000);
      } else if (digitalRead(sensor_infrarrojos) == 1) {
            led_RGB.setRGBcolor(255, 0, 0);
            servoRC.write(90);
            delay(2000);
      }
}
```

Práctica segunda, elemento de barrido por ultrasonidos

Esta práctica va a consistir en un sensor ultrasónico montado sobre un servo que irá girando sucesivamente haciendo cuatro paradas en cada recorrido.
Cuando el servomotor gira, va haciendo un barrido de la estancia con el sensor a distancia, así podemos determinar con una relativa exactitud la distancia existente desde el sensor a los objetos más próximos de la estancia. Tenemos también cuatro leds que se van a corresponder con los cuatro puntos de medida, donde el sensor va a hacer lecturas y su brillo dependerá de la distancia al objeto en esa lectura.

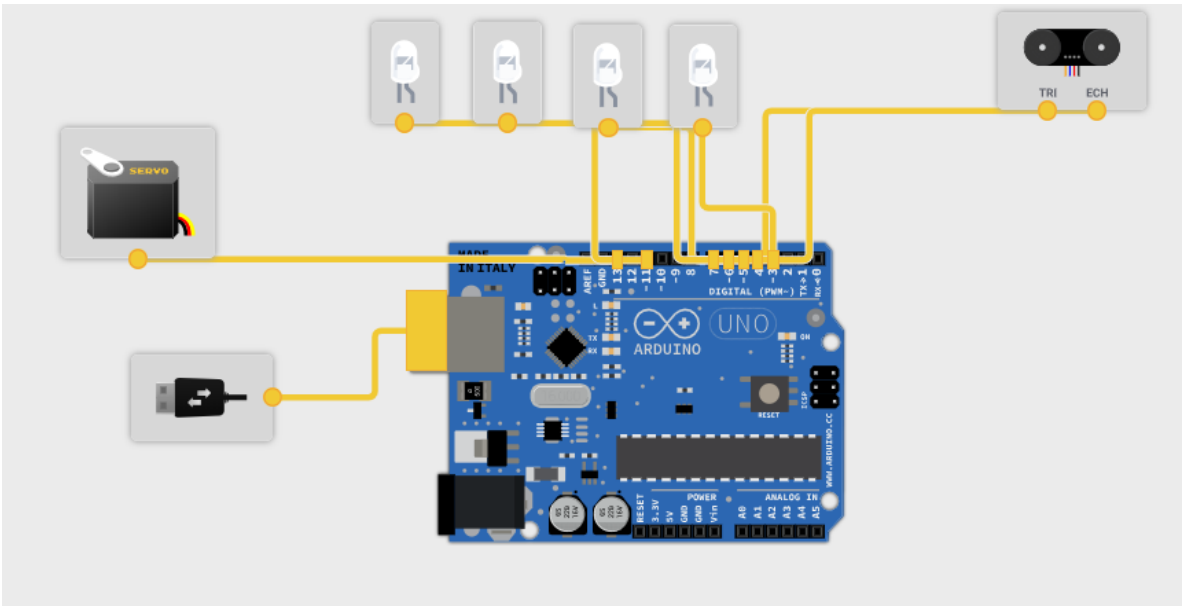
También podemos monitorizar dicha señal y enviarla por el puerto usb a la pantalla del ordenador.

Funcionamiento de un sensor IR

La luz IR es una parte del espectro electromagnético no visible por el ojo humano. Se puede implementar un sistema de este estilo para simular un sistema de visión nocturna. El sensor IR hace brillar un led de luz IR y a través de un sistema de circuitos complejos permite calcular el ángulo en que esa luz emitida regresa a un fotosensor que hay al lado del diodo led IR. A través de una serie de tensiones que crean las lecturas del fotosensor IR se calcula la distancia y se convierte la señal de voltaje analógica para poder leerla en el microprocesador. Así, aún a oscuras, el sensor permite hacer esas medidas al usar una longitud de onda diferente a la que percibe el ojo humano.

Modo de funcionamiento del lector

El sensor tiene dos conexiones, echo y return, que ocuparán dos patillas de arduino, además conectaremos cuatro diodos leds a las patillas PWM para que varíen su brillo (en este caso usaremos los pines 3, 5, 6 y 11. Los pines 9 y 10 también son PWM pero no sirven para crear señales con analogWrite cuando se usa la biblioteca del servo. Por usar el mismo temporizador Hardware que el que se usa para los conectores PWM en esos dos pines.



El último paso a seguir sería la programación del sistema. El sistema funciona del siguiente modo, primero rota a una posición determinada, mide la distancia en ese ángulo, la convierte en un valor que pueda usarse para el led, cambiar el brillo del led, se mueve a la siguiente posición y así sucesivamente.

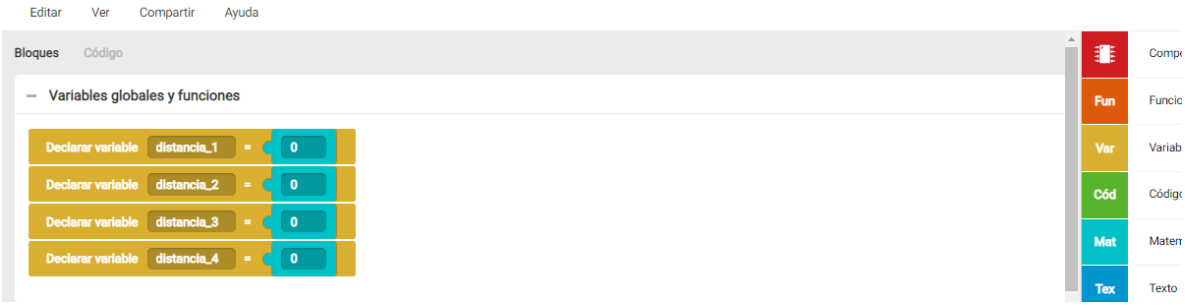
Tenemos que tener en cuenta, a la hora de hacer el programa, el punto donde se va a situar el lector de distancias dado que si se instala en un ángulo, el barrido del lector es de 90º y si se localiza en otro punto de la habitación el barrido será de 180º. Así, si vamos a hacer 4 lecturas en cada barrido, en el primer caso el movimiento del servo será de 22'5º para cada lectura y en el caso de 180º se hará una lectura cada 45º.

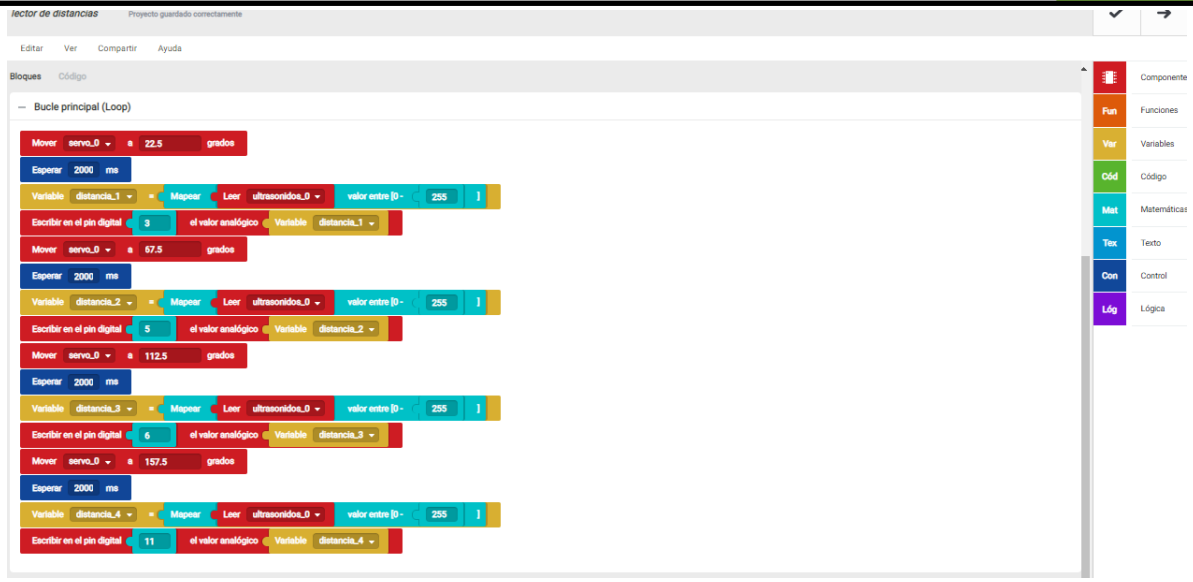
El ciclo del programa será pues:

- Giro del servo hasta el punto inicial de los movimientos (a).
- Esperar 2 segundos.
- Hacer lectura de la distancia.
- Convertir la lectura en una tensión entre 0 y 5V.
- Dar esa tensión a la salida del led correspondiente.
- Girar el servo al siguiente ángulo de lectura (b).
- ….

  Repetiríamos de igual modo en los cuatro ángulos de lectura.

Primero declararíamos las variables medidas





Si lo queremos hacer en el IDE, el código sería el que sigue.

```
/***  Included libraries  ***/
#include <BitbloqUS.h>
#include <Servo.h>
#include <BitbloqSoftwareSerial.h>

/***  Global variables and function definition  ***/
US ultrasonidos_0(7, 4);
Servo servo_0;
bqSoftwareSerial puerto_serie(0, 1, 9600);
const int led = 3;
const int led_2 = 5;
const int led_3 = 6;
const int led_4 = 11;

float distancia_1 = 0;
float distancia_2 = 0;
float distancia_3 = 0;
float distancia_4 = 0;

/***  Setup  ***/
void setup() {
      servo_0.attach(13);
      puerto_serie.begin(9600);
      pinMode(led, OUTPUT);
      pinMode(led_2, OUTPUT);
      pinMode(led_3, OUTPUT);
      pinMode(led_4, OUTPUT);

}

/***  Loop  ***/
void loop() {
      servo_0.write(22.5);
      delay(2000);
      distancia_1 = map(ultrasonidos_0.read(), 0, 1023, 0, 255);
      analogWrite(3, distancia_1);
      servo_0.write(67.5);
```

```
    delay(2000);
    distancia_2 = map(ultrasonidos_0.read(), 0, 1023, 0, 255);
    analogWrite(5, distancia_2);
    servo_0.write(112.5);
    delay(2000);
    distancia_3 = map(ultrasonidos_0.read(), 0, 1023, 0, 255);
    analogWrite(6, distancia_3);
    servo_0.write(157.5);
    delay(2000);
    distancia_4 = map(ultrasonidos_0.read(), 0, 1023, 0, 255);
    analogWrite(11, distancia_4);
}
```

Otra variable que se podría poner es cada una de las lecturas, a través del puerto USB. Así, además de variar las tensiones en los diodos, podríamos ver dicha distancia a través del monitor del programa.

CONTROLAR MOTORES C.C.

Los motores C.C. los vamos a encontrar en infinidad de aparatos, suelen funcionar con pequeñas pilas y según se ajuste su tensión girarán de una forma más o menos rápida. Igualmente, si le invertimos la polaridad le invertiremos el sentido de giro, y con ello la forma de funcionar del elemento que le hayamos conectado a la salida.

Como descripción escueta, decir que son motores que tienen escobillas para transferir la corriente a una bobina (eje) que girará dentro del campo magnético creado por unos imanes estacionarios. Se les conoce por ello como motores de escobillas. Cuando manejemos estos motores con Arduino tendremos que tener cuidado por varios motivos:

a) La alimentación de Arduino no sea capaz de mover estos motores, por lo que habrá que montarlos sobre las placas shield y alimentarlos exteriormente, teniendo en cuenta la corriente que absorben al girar.
b) Si se giran estando parados, pueden funcionar como generadores de tensión, y con ello poner tensión distinta de 5V a una entrada/salida Arduino, con el problema de romper la placa.

Para ello tendremos que utilizar una serie de circuitos con transistores para controlar el encendido o apagado de estos motores desde Arduino, así como el control de la velocidad del mismo utilizando las salidas PWM de las placas. Un circuito como el de la figura es muy práctico para esta aplicación.

Los componentes del circuito tienen una función determinada:

- Q1 Alimenta con 9V el motor según esté en estado de corte-saturación (Encendido-apagado), según le llega corriente a la base varía la intensidad de colector. Cuando está en corte, el circuito está abierto y no pasa corriente que haga funcionar el motor. Los 5V que genera Arduino son suficientes para hacerlo funcionar. En su estado de saturación, el transistor, con una pequeña corriente en la base permite pasar la suficiente corriente entre el colector y emisor que hace funcionar el motor.

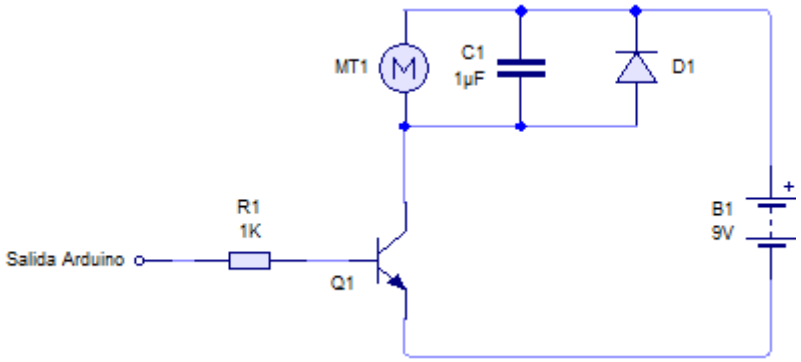
  Por otro lado, si utilizamos una salida PWM en lugar de la digital normal, podremos modificar la velocidad del motor si modificamos la frecuencia de encendido/apagado del transistor. Así, al

mantener el motor el impulso, el ciclo de trabajo de la PWM es el encargado de variar la velocidad de giro del motor.

- R1 es una resistencia para limitar la corriente de Arduino a Q1. En circuitos industriales se suele sustituir por un optoacoplador que aísla completamente la salida de la parte de potencia.

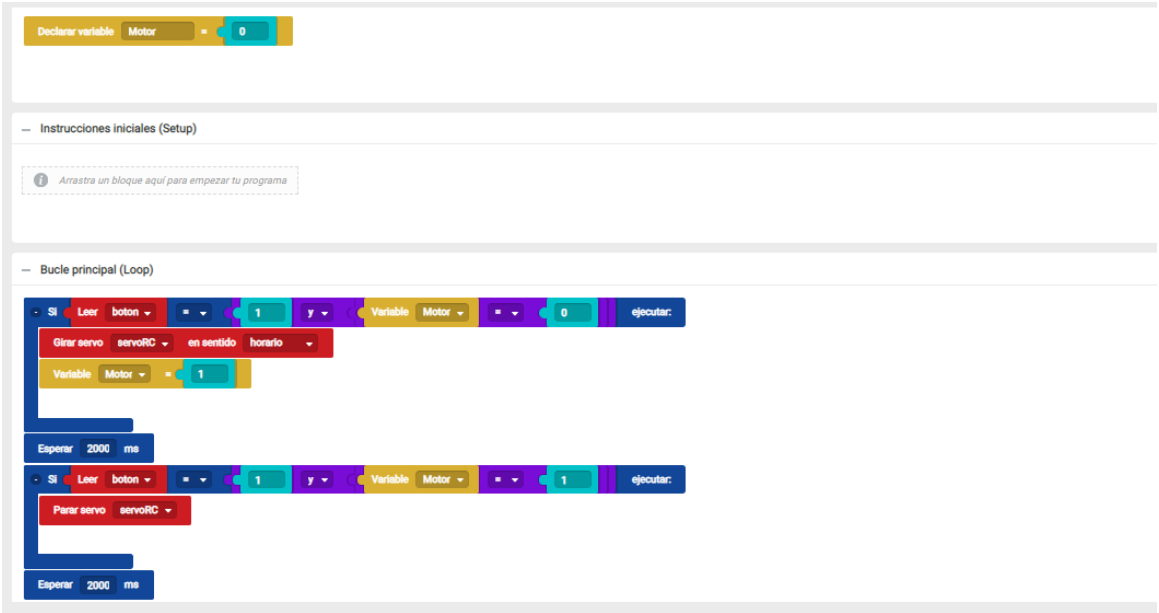
- C1 es un filtro para anular el ruido del motor.

- D1 evita que si se mueve el motor, haga de generador de tensión y produzca una tensión que destruya la salida de Arduino. Es muy importante tener en cuenta los problemas que causan los



motores de C.C. al tener inductancias para su funcionamiento. A medida que el motor va girando, aumenta la energía y se queda almacenada en las bobinas del motor. Si se elimina la alimentación en un momento determinado, esa energía se disipa a través de las escobillas en forma de pico inverso de tensión, y eso podría provocar daños en la alimentación e incluso en el elemento de control. Por eso se ponen estos diodos de protección, si se genera una corriente, ésta pasará por el diodo inverso, que absorberá la corriente generada.

Cuando se hagan las conexiones habrá que tener cuidado de la polarización del transistor, el motor y el diodo. La fuente que alimenta al motor se le conectará solamente al colector del transistor y se evitará unir con la alimentación de Arduino, si acaso, el polo negativo sí puede ser común, pero solamente el negativo. Así el programa se quedaría como la figura:

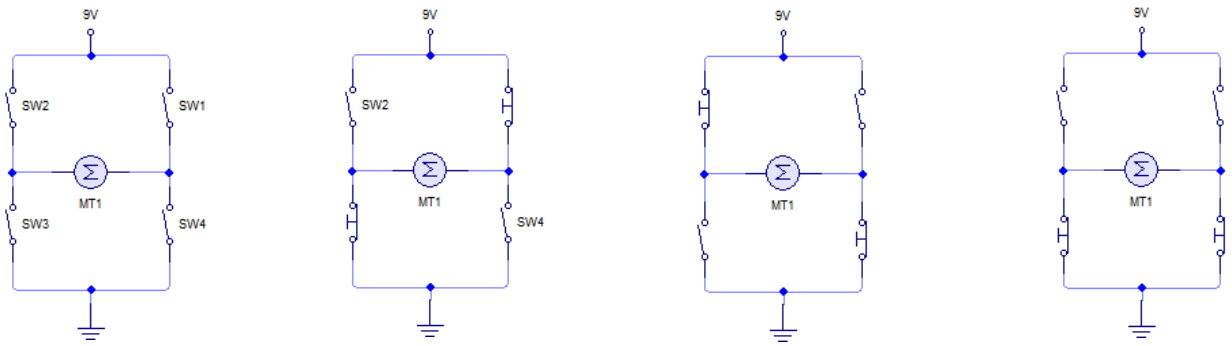


El programa es el encendido de un motor, solamente encendido, otras variables las veremos más adelante. Vamos a hacer un programa que cuando se accione un pulsador se active el motor. Utilizaremos una salida normal (no PWM). Al hacerlo en bitbloq no tenemos motor DC, que vamos a poner como un servo-motor de movimiento continuo. Si hacemos el esquema anterior la salida arduino bien puede ser, al ser solamente encendido-apagado, una salida led normal.

El código sería:

```
/***  Included libraries  ***/
#include <Servo.h>

/***  Global variables and function definition  ***/
const int boton = 6;
Servo servoRC;

float Motor = 0;

/***  Setup  ***/
void setup() {
    pinMode(boton, INPUT);
    servoRC.attach(8);

}

/***  Loop  ***/
void loop() {
    if (digitalRead(boton) == (1 && (Motor == 0))) {
        servoRC.write(180);
        Motor = 1;
    }
    delay(2000);
    if (digitalRead(boton) == (1 && (Motor == 1))) {
        servoRC.write(90);
    }
    delay(2000);
}
```

El sentido de giro del motor se puede controlar de varios modos. Esta variación de sentido permite a Arduino poner motores en robots o elementos que puedan permutar el sentido de giro de los motores que se conecten. Para ello se puede conectar un sencillo puente H; el funcionamiento de un puente H se resume como un motor puesto en medio de cuatro transistores conectados precisamente en forma de H.

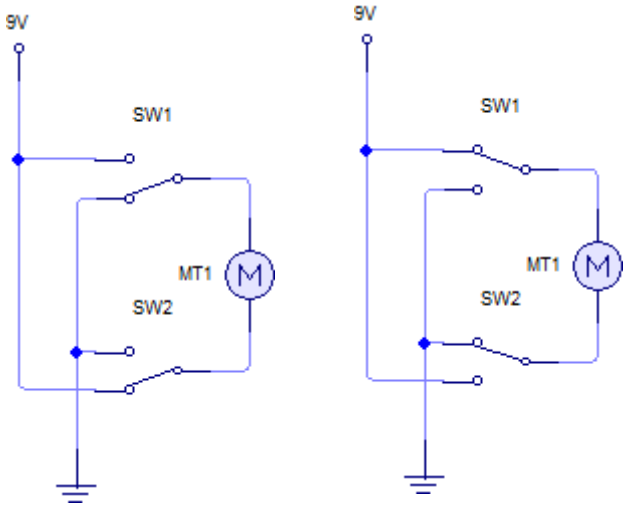


Puente abierto Avance Retroceso Frenado

El puente tiene cuatro formas de funcionamiento, puente abierto, avance, retroceso y frenado. En el primero de los casos, puente abierto, el motor no funcionará. El puente en sentido avance el motor marchará en un sentido por tener dos de los transistores cruzados cerrados, el puente en sentido retroceso hace girar el motor en sentido contrario al anterior (marcha atrás) y en caso de frenar el motor, el puente abrirá la alimentación y conectará los dos polos a través de los transistores a masa, esta conexión hace que la posible generación de corriente en las bobinas del motor se deriva a masa a través de estos transistores. Hay varios tipos de estos puentes, por ejemplo el SN754410.

Sin embargo podemos hacer otra variación, sobre todos si usamos una salida PWM para variar también la velocidad del motor, como vamos a ver ahora. Si utilizásemos una salida digital libre, conectando a ella sendos relés cuyos contactos sean Común, NO y NC podemos tener dos combinaciones, según el esquema siguiente tales que nos sirven para invertir el sentido de giro.

SW1 serían los contactos del relé 1, cuyo común iría al motor y las salidas NO y NC irían a positivo y negativo, los contactos de SW2 irán igual pero cruzados, así mientras la salida NC de SW1 está a alimentación, la misma salida de SW2 estará conectada a masa y viceversa, de este modo, con una salida podemos invertir el sentido de giro, sobre todo si con otra salida controlamos la velocidad. Para la inversión de sentido bastará con poner un pulsador que nos active los relés.

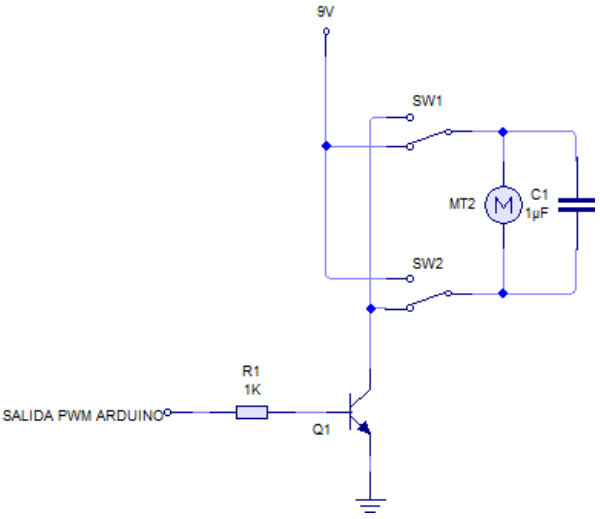


Si al esquema anterior le sumamos una salida PWM podemos controlas análogamente la velocidad del motor.

En el ejemplo que vamos a poner vamos a ir variando progresivamente la salida PWM de arduino para controlar un motor CC, es un bucle que va incrementando el valor de la tensión a la salida desde 0 hasta los 255 pasos que tienen dichas salidas. Espera 2 segundos y posteriormente empieza a decrementar la tensión hasta llegar nuevamente a cero voltios. Repitiendo el ciclo. Si a esto se le añade un pulsador que active o no los dos relés, según el esquema eléctrico que se muestra, conseguiremos una variación de velocidad conjuntamente con una inversión de giro.

Dos puntos a destacar del presente diseño:

- No hay diodo de protección al ser un circuito con inversión de polaridad, si lo mantuviésemos provocaríamos un cortocircuito al polarizar en una de las formas.
- Los relés los controla la misma salida, con lo que podría ser necesario poner un elemento para controlar la potencia que consuman y no quemar la salida de la placa (recuerdo que cada patilla tiene una corriente de salida de



40mA). También a destacar que las conexiones están cruzadas Alimentación iría a la patilla NO de SW1 y a la vez a la NC de SW2, por el contrario, a masa conectaríamos la patilla NC de SW1 y también la NO de SW2. Los dos polos del motor irían a las patillas comunes de ambos relés.

El programa de control de la salida PWM sería, añadiéndole el pulsador de inversión de giro:

El programa IDE, para el que se quiera complicar un poco sería el que sigue:

```
/***  Included libraries  ***/
#include <Servo.h>

/***  Global variables and function definition  ***/
const  int  boton  =  8;
const  int  led  =  12;
```

```
Servo servoRC;

float Sentido_giro = 0;
float Salida_motor = 0;
float contador = 0;

/***  Setup  ***/
void setup() {
      pinMode(boton, INPUT);
      pinMode(led, OUTPUT);
      servoRC.attach(3);

}

/***  Loop  ***/
void loop() {
      if (Sentido_giro == (0 && (digitalRead(boton) == 1))) {
            digitalWrite(led, HIGH);
            Sentido_giro = 1;
            delay(2000);
      }
      if (Sentido_giro == (1 && (digitalRead(boton) == 1))) {
            digitalWrite(led, HIGH);
            Sentido_giro = 0;
            delay(2000);
      }
      for (contador = 0; contador <= 255; contador += 1) {
            analogWrite(3, Salida_motor);
            delay(10);
      }
      delay(2000);
      for (contador = 255; contador >= 0; contador -= 1) {
            analogWrite(3, Salida_motor);
            delay(10);
      }
      delay(2000);
}
```

Quiero destacar en este caso que el programa btbloq no tiene entre sus componentes el motor C.C., lo he tenido que sustituir por un servo de movimiento continuo. Tendríamos que cambiar en la tercera línea del programa, donde pone:

```
Servo servoRC;
```

Habría que poner: Cons int Motor =3;

Igualmente, en la tercera línea del Void Setup, donde pone:

```
ServoRC.attach(3);
```

Sustituirlo por: pinMode (Motor, Output);