Sistemas con Microprocesadores

Práctica 3: Control de motores con Arduino.

Resumen:

Utilizar una placa Arduino para controlar un motor de corriente continua (mediante un puente H) y un servomotor.

Entrega:

Hacer públicos los diseños Tinkercad de los ejercicios 1 y 2.
Entregar un PDF de 2 páginas aproximadamente que contenga:

- Nombre y apellidos.
- Dirección web de los proyectos Tinkercad.
- Captura de pantalla del funcionamiento del motor al 40% de velocidad.
- Captura de pantalla del funcionamiento del servo en la posición de 120º.
- Código fuente pegado como texto de ambos ejercicios.
  (Antes de pegar el código fuente en el documento PDF lo colorearemos, por ejemplo, utilizando la página: https://highlight.hohli.com/?language=cpp)

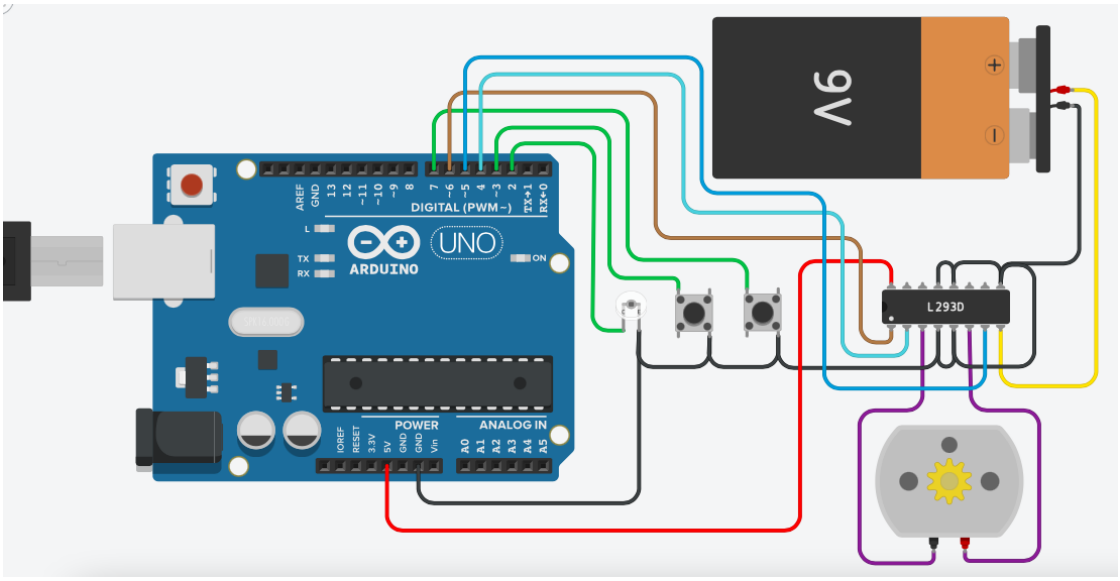
Desarrollo:

Esta práctica está dividida en 2 ejercicios:

1) Realizar un diseño que permita el control de un motor de corriente continua con un puente H mediante 2 botones y un fototransistor.

Cómo funcionarán los botones:
- El fototransistor controla el encendido y apagado del motor. Cuando el fototransistor detecta luz, el motor permanece encendido. En caso contrario el motor permanece apagado.
- El segundo botón cambia el sentido de giro, a cada pulsación cambia de un sentido al otro.
- El tercer botón controla la velocidad. A cada pulsación cambia entre: 20%, 40%, 60%, 80% y 100%.

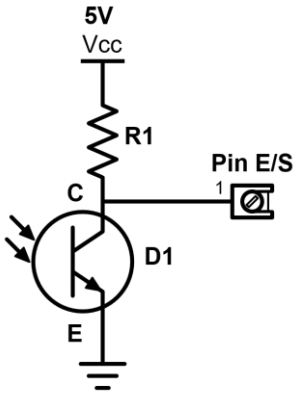
Utilizaremos el circuito integrado L293D como puente H, el cual permite controlar hasta 2 motores de corriente continúa pudiéndolos hacer girar en cualquier sentido, o bien un motor paso a paso bipolar. http://www.ti.com/lit/ds/symlink/l293.pdf



Para controlar la velocidad del motor utilizamos una salida PWM (*pulse width modulation*, "modulación por ancho de pulsos") del Arduino, la cual conectamos a la entrada de habilitación (*enable*) del canal usado del L293 (*1,2EN* en el *datasheet*). Por tanto, utilizamos la función `analogWrite(pin, valor);` donde `pin` es el pin de salida a usar del Arduino y `valor` puede estar entre 0 y 255. Un valor de 255 establece una velocidad del 100%. Debemos tener en cuenta que no todas las patillas del microcontrolador soportan la salida PWM (por hardware). Para nuestra salida PWM utilizaremos el pin 3, 5, 6, 9, 10 u 11.

Para apagar el motor dejamos la patilla de habilitación del L293 en baja, es decir, escribimos un valor 0 con `analogWrite()`. Y para encenderlo escribimos el valor PWM de velocidad que haya sido establecido por el usuario.

El encendido/apagado lo determina el fototransistor, el cual, en nuestro caso se comportará como un interruptor. El fototransistor disponible en Tinkercad es de tipo NPN, por tanto, lo conectaremos con la siguiente configuración:

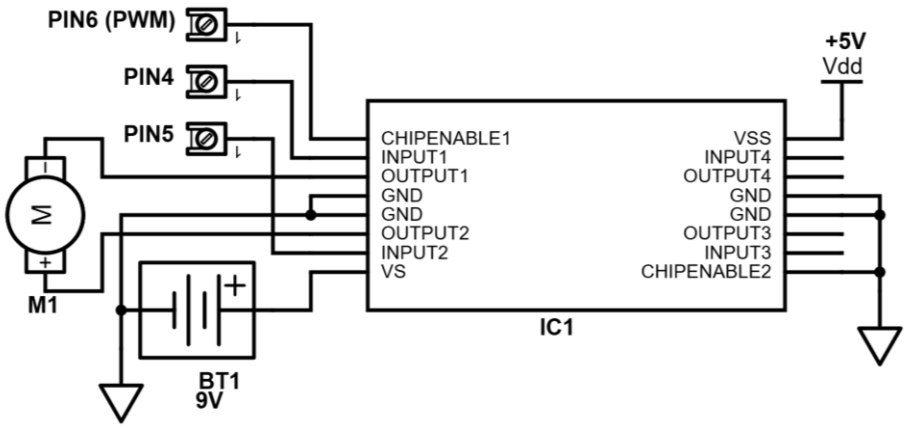


Cuando el fototransistor (D1) recibe luz, conduce la corriente del colector (C) al emisor (E), y se comporta como un interruptor cerrado, conectando la patilla del microcontrolador a masa, por tanto, el microcontrolador leería un 0 en esa patilla. Cuando el fototransistor no recibe luz, no conduce corriente, se comporta como un interruptor abierto, y la resistencia de *pull-up* lleva el potencial de 5 V a la patilla, por tanto, el microcontrolador leería un 1 en esa patilla. Como el microcontrolador que

estamos usando permite activar resistencias de *pull-up* internas, no es necesario incluir la resistencia R1 en nuestro circuito mediante un componente electrónico. El fototransistor de Tinkercad tiene la patilla del emisor a la derecha, por tanto, esta será la que conectemos a masa. Para indicarle a Tinkercad la cantidad de luz que está recibiendo el fototransistor, una vez iniciada la simulación, pinchamos en él y aparecerá una barra de control deslizante.

Para establecer el sentido de giro utilizaremos las 2 entradas del canal 1 del L293 (*1A* y *2A* en el *datasheet*, o *INPUT1* e *INPUT2* en el esquema de abajo), debiendo siempre estar una de ellas en alta (1) y la otra en baja (0) o viceversa para establecer una diferencia de potencial en los bornes del motor (cambiando de una configuración a la otra al pulsar el botón de sentido de giro).

En el siguiente esquemático podemos observar cómo conectar el L293D (IC1 en el esquemático) para este aparatado de la práctica:



El circuito integrado L293D incluye 2 *diodos de libre circulación* por cada salida (*output*) que protegen al puente H de lo picos de tensión que se producen al desconectar la alimentación del motor. Si usásemos el circuito integrado L293, tendríamos que incluir nosotros estos diodos en el circuito.

No debemos olvidar configurar todos los pines de entrada y salida que utilicemos mediante la función `pinMode()`, e inicializar los pines de salida usados a un valor conocido en la función `setup()`. Como estamos utilizando un microcontrolador de 8 bits es conveniente que se utilicen variables de tipo entero de 8 bits (o más pequeñas) siempre que sea posible. En esta práctica los tipos elementales que necesitamos utilizar son solamente `uint8_t` y `bool`.

2) Realizar un diseño para controlar la posición de un servomotor mediante un mando a distancia por infrarrojos.

Funcionamiento de los botones del mando:
- Una pulsación del botón “avanzar” realizará un avance de 30 grados (hasta 180 grados como máx.).
- Una pulsación del botón “retroceder” realizará un retroceso de 30 grados (0 grados como mín.).
- Inicialmente el servomotor parte de la posición 90 grados.
- Una pulsación del botón “0” devolverá el servomotor a la posición inicial (90 grados).
- No debemos especificar posiciones inferiores a 0 grados o superiores a 180 grados.

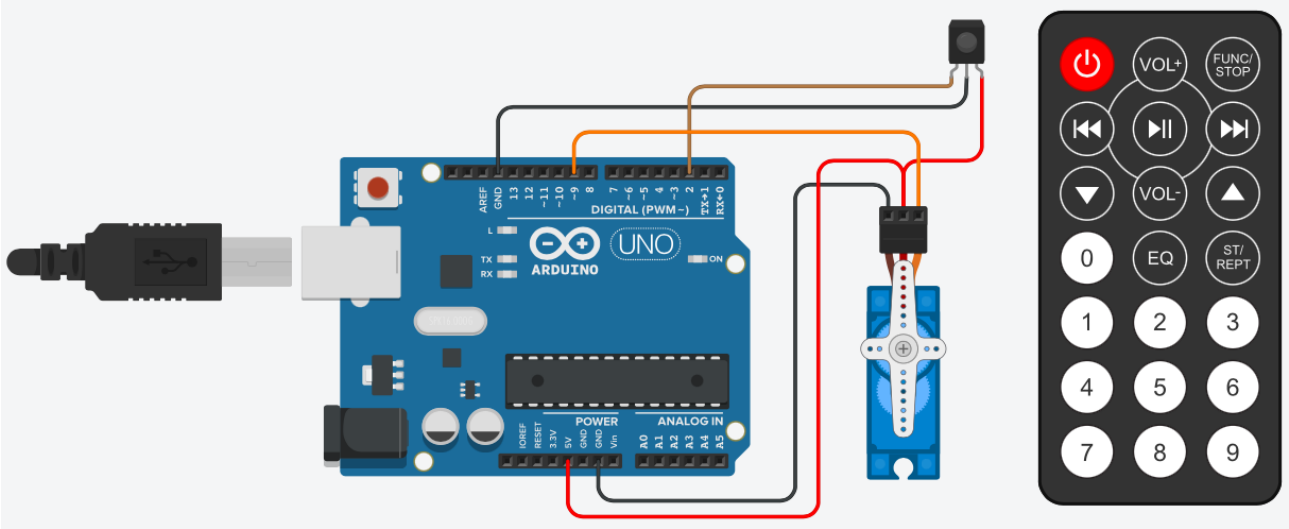
Control del servomotor:

Podemos utilizar la biblioteca `Servo` de Arduino para generar las señales que controlarán nuestro servomotor. Para ello incluiremos en nuestro código fuente el fichero de cabecera: `#include <Servo.h>`

Declaramos un objeto `Servo` como variable global. Por ejemplo: `Servo Miservo;`

En la función `setup()`, indicamos el pin asociado al control del servo: `Miservo.attach(PIN_SERVO);`

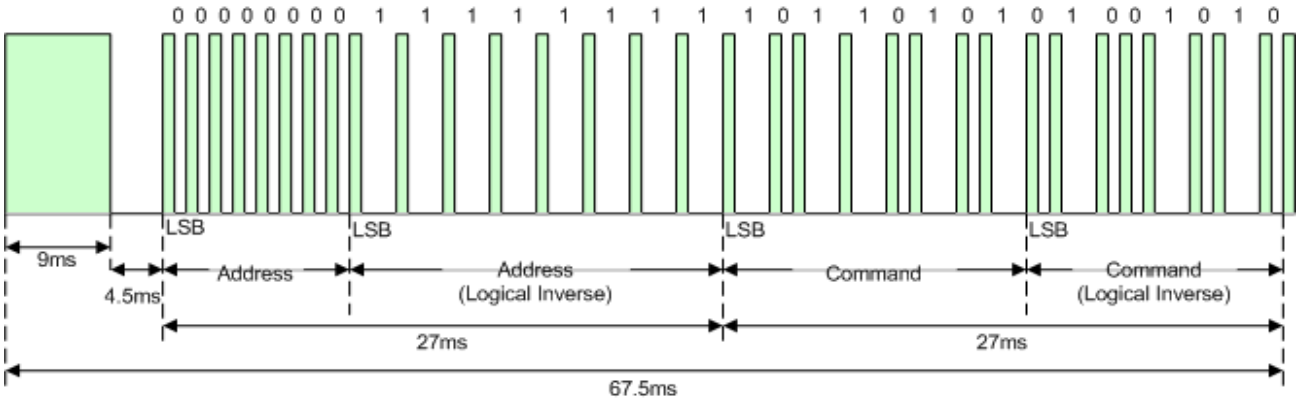
Y cuando queramos realizar un cambio del ángulo en el que está posicionado el servo utilizaremos: `Miservo.write(angulo);`



Lectura del mando a distancia por infrarrojos:

En nuestro diseño incluimos el receptor de infrarrojos que incluye Tinkercad y el mando a distancia, como se muestra en la figura anterior. Estos utilizan el protocolo NEC IR, el cual es utilizado por diversos mandos a distancia.

- La codificación en bits que usa este protocolo para codificar las pulsaciones de botones es:
  - Inicio: Pulso de 9 ms seguido de un espacio 4,5 ms
  - 0 lógico: pulso de 562,5 µs seguido de un espacio de 562,5 µs, con un tiempo de transmisión total de 1,125 ms.
  - 1 lógico: pulso de 562,5 µs seguido de un espacio 1,6875 ms, con un tiempo total de transmisión de 2,25 ms.
- Trama completa (32 bits): Dirección + Dirección complementada + Orden + Orden complementada.



La dirección identifica el equipo que se está controlando y el comando indica qué botón se presionó. En la versión original del protocolo existen unos bits de dirección y de comando complementados que añaden información redundante (como se muestra en la figura anterior). Estos se pueden utilizar para compararlos con el complemento de sus versiones no complementadas, y así verificar que coinciden y no ha habido errores en la transmisión. No obstante, algunos mandos a distancia envían direcciones de 16 bits y prescinden de los 8 bits de dirección complementados (como el mando de Tinkercad), y otros incluso envían comandos de 16 bits.
Este protocolo también utiliza un código para indicar repeticiones de botones (cuando un botón se deja pulsado).
Para procesar las señales que el receptor de infrarrojos transmite a la placa Arduino y obtener los bits de la trama utilizamos el objeto `IrReceiver` como se muestra en el siguiente código fuente.
En la figura anterior los bits se representan con el orden en el que son transmitidos a lo largo del tiempo, en concreto, los bits menos significativos primero. Mientras que si imprimimos los datos devueltos por `IrReceiver.decodedIRData.decodedRawData`, los veremos totalmente al revés, con lo que los bits menos significativos se muestran a la derecha.

```
#include <IRremote.h>
#define PIN_SENSOR_IR 2 // Sensor conectado al pin 2

void setup()
{
  Serial.begin(9600);
  // Inicializa el receptor
  IrReceiver.begin(PIN_SENSOR_IR, ENABLE_LED_FEEDBACK);
}

void loop()
{
  if (IrReceiver.decode()) // ¿Hay trama?
  {
    uint32_t trama_ir;
    uint8_t comando_ir;
    trama_ir = IrReceiver.decodedIRData.decodedRawData;
    Serial.println(trama_ir, HEX);

    // Extraemos de la trama el comando sin complementar
    comando_ir = (trama_ir>>16) & 0xFF;
    if(comando_ir == 0x06)
      Serial.println("Avanza");
    else if (comando_ir == 0x04)
      Serial.println("Retrocede");
  }
  IrReceiver.resume();
}
```

Modificaremos el código fuente anterior del programa para controlar el servo como se indica al inicio del enunciado del ejercicio.