P4 – CONEXIÓN Y PROGRAMACIÓN DE CODIFICADORES INCREMENTALES

Un codificador (o *encoder*) incremental permite determinar la posición de un mecanismo o eje de forma digital, gracias a las señales de pulsos que genera. En esta práctica se estudiará el funcionamiento de un codificador incremental rotativo de dos canales (A y B), se abordará como se conecta a un microcontrolador, y realizarán programas que gestionen la cuenta de pulsos y calculen la posición y velocidad del eje.

La práctica se desarrollará tanto utilizando componentes reales, incluyendo un Arduino Uno y un codificador incremental industrial, como con la aplicación web Circuits del proyecto Tinkercad de Autodesk (https://www.tinkercad.com), la cual permite simular el funcionamiento de circuitos controlados con un Arduino Uno. Entre los componentes que se pueden añadir a un circuito, Tinkercad – Circuits incluye dos motores con codificadores rotativos incrementales. De este modo, se utilizará tanto con codificador rotativo industrial real por sí solo, como un conjunto motor-codificador-reductor en la simulación.

En los siguientes apartados, primero se explican todos los conceptos básicos que conviene conocer para afrontar el desarrollo de la práctica, y después se plantean los experimentos a realizar. La gran mayoría de los conceptos explicados en este manual ya se han estudiado en las clases de teoría, especialmente en el tema 6, pero también se incluyen en este manual de forma resumida, a modo de repaso. De todas formas, es conveniente repasar el tema 6 de las clases de teoría, que aborda funcionamiento y uso de los codificadores.

Contenido

Conceptos básicos 2
1.1. Codificadores rotativos incrementales 2
1.1.1. Funcionamiento 2
1.1.2. Cálculo de posición 4
1.1.3. Cálculo de velocidad 5
1.2. Control simple de un motor de corriente continua 5
1.3. Codificador incremental E6B2 en el laboratorio 8
1.4. Codificadores incrementales en Tiknercad 10
1.4.1. Opciones 10
1.4.2. Conexiones básicas 11
1.4.3. Conexión con Arduino 12
1.5. Aspectos sobre programación de codificadores incrementales 15
1.5.1. Gestión de interrupciones 15
1.5.2. Acceso directo a los pines de entrada y salidas digitales 18
1.5.3. Medida de tiempo y eventos de temporizador 19
2. Experimentos 20
2.1. Actividades básicas (7 puntos) 20
2.1.1. Características de un codificador de Tinkercad (1,5 puntos) 20
2.1.2. Cuenta de pulsos en modo 1x (2 puntos) 21
2.1.3. Cuenta de pulsos en modo 4x (2 puntos) 22
2.1.4. Cálculo de la posición (1,5 puntos) 22
2.2. Actividades complementarias 23
2.2.1. Pantalla LCD (2 puntos) 23
2.2.2. Cálculo de la velocidad (2 puntos) 23
2.2.3. Posicionar motor (2 puntos) 23
3. Entrega 24

Conceptos básicos

1.1. Codificadores rotativos incrementales

1.1.1. Funcionamiento

La construcción más habitual de un codificador incremental rotativo es la que muestra la Figura 1. Dispone de un optoacoplador cuyo haz de luz interrumpe o se deja pasar según las zonas opacas y las ventanas (o líneas) que tiene un disco unido al eje de rotación a medir. El LED se mantiene activado para que emita luz de forma continua. De esta forma, a media que el eje y el disco giran, el fototransistor cambia entre corte y conducción (saturación), produciendo una señal digital con una sucesión de pulsos que llegan al controlador. Conocido el número de líneas que tiene el disco, el controlador puede saber la posición actual del eje respecto a una posición inicial simplemente contando los pulsos que ha generado el codificador durante el movimiento.

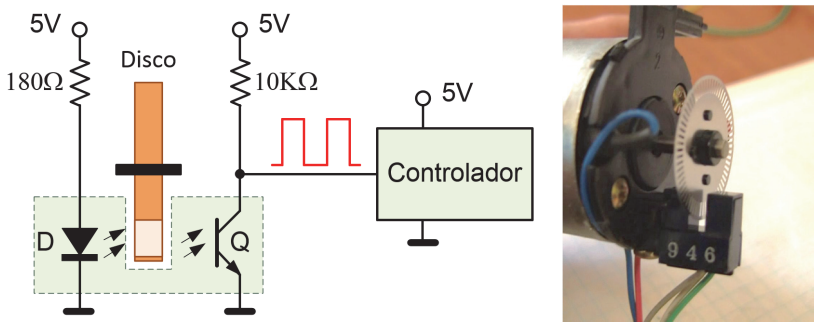


Figura 1. Funcionamiento de un codificador incremental rotativo básico de un canal.

Un codificador se caracteriza por su resolución, que corresponde con el desplazamiento del eje que equivale a cada pulso, esto es, la unidad de desplazamiento mínima que puede detectar. En un codificador rotativo, la resolución (es el valor de grados por pulso (°/pulso) o radianes por pulso (rad/pulso). Pero es más habitual es que el fabricante de un codificador indique el número de pulsos que el codificador genera por cada vuelta o revolución del eje, que es igual al número de líneas (NL) del disco del codificador. La resolución del codificador (R) se puede calcular entonces como:

$$R = 360°/NL\ [°/pulso] = 2\pi/NL\ [rad/pulso]$$

El codificador de la Figura 1 tiene un canal de salida porque solo proporciona una señal de pulsos. Con un codificador como este, el controlador no puede conocer el sentido de giro del eje a partir de la señal de pulsos del codificador, solo el desplazamiento que ha realizado el eje, sea en uno u otro sentido de giro.

En más habitual emplear codificadores incrementales de dos o tres canales de salida, esto es, que proporcionan más de una señal de pulsos. La muestra Figura 2 un ejemplo de codificador de tres canales, los cuales se denominan habitualmente canales A, B e I, aunque algunos fabricantes también llaman Z al canal I. Los canales A y B ofrecen salidas idénticas de pulsos en función de la rotación del disco, pero desfasadas un cuarto de ciclo. El canal I (índice) ofrece un pulso por vuelta, lo que puede ser utilizado por el controlador para determinar cuando el eje está en una posición inicial o de referencia.

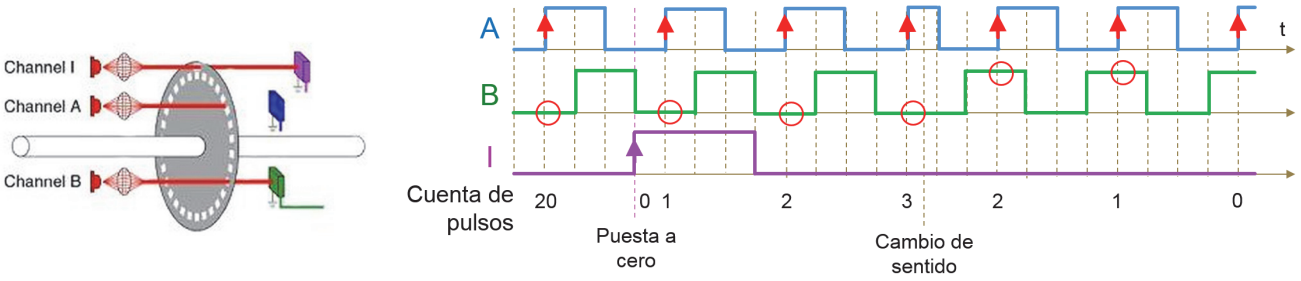


Figura 2. Funcionamiento de un codificador incremental rotativo de tres canales, con cuenta en modo 1X.

El controlador el codificador puede analizar y comparar las señales A y B no solo para llevar una cuenta de los pulsos y saber la posición, sino también para determinar el sentido de giro del eje. Y en función del sentido de giro, la cuenta de pulsos debe aumentar o disminuir. Esto se puede llevar a cabo dentro del programa del controlador de diferentes formas o modos de funcionamiento. Los modos más habitualmente utilizados son los tres siguientes:

- Modo 1x. Se usa el canal A para contar pulsos en cada flanco de subida de la señal (transición de nivel bajo a alto), y el canal B solo para determinar el sentido de giro. Por ejemplo, se puede activar una rutina de interrupción del programa en cada flanco de subida de la señal del canal A. En esa rutina se realiza la cuenta de pulsos de A, como muestra la Figura 2, de forma que la cuenta se actualiza en función del nivel que tenga la señal del canal B. Así, la cuenta aumenta una unidad si B está a nivel bajo, o disminuye una unidad si B está nivel alto. Con este modo, se tiene que el total de pulsos por vuelta o revolución (PPR) es igual al número de líneas del disco (NL), y de ahí el nombre "1x":

$$\mathrm{PPR} = 1 \cdot \mathrm{NL} \quad [\mathrm{ppr}]$$

- Modo 2x. Se cuentan pulsos tanto en los flancos de subida como en los de bajada de la señal del canal A, como muestra la Figura 3. Para ello se puede programar una rutina de interrupción que se active para ambos flancos del canal A, si el controlador lo admite. El sentido de giro se determina en función de si el nivel de la señal del canal B es igual o no al de la señal del canal A después de un flanco de la segunda. En este caso se tiene que:

$$\mathrm{PPR} = 2 \cdot \mathrm{NL} \quad [\mathrm{ppr}]$$

- Modo 4x. Se cuentan todos los flancos de subida y bajada de las señales de los canales A y B, como se representa en la Figura 4. Para llevar a cabo esto, se puede programar dos rutinas de interrupción, una que se activa con los flancos de la señal del canal A y otra con los del canal B. El sentido de giro se determina comparando entre sí los niveles de la señales de los dos canales después de producirse un flanco. Con este modo se tiene que:

$$\mathrm{PPR} = 4 \cdot \mathrm{NL} \quad [\mathrm{ppr}]$$

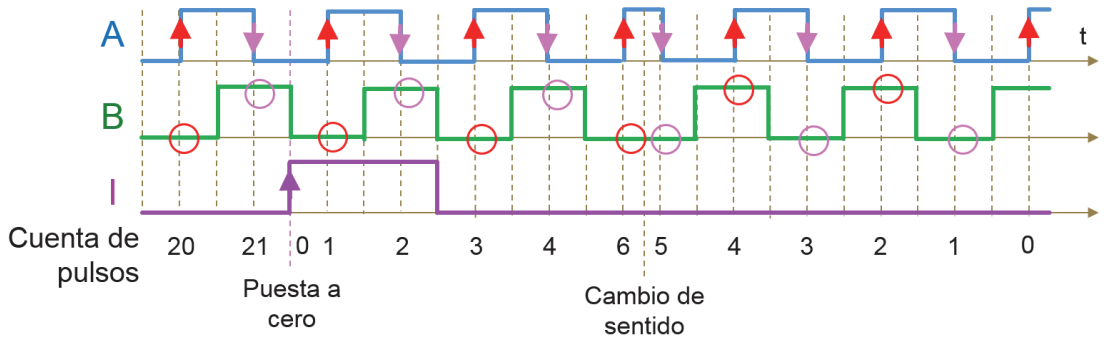


Figura 3. Modo 2x para la cuenta de pulsos.

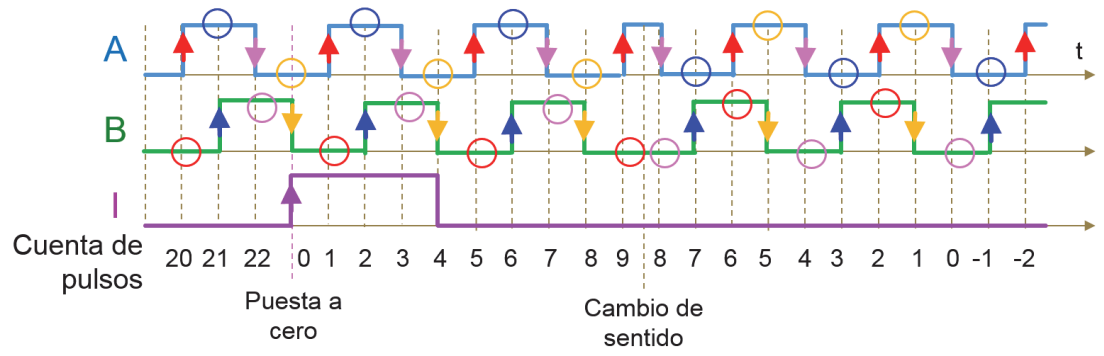


Figura 4. Modo 4x para la cuenta de pulsos.

De este modo, la resolución efectiva del codificador (Re) se puede mejorar (hacer más pequeña) respecto a su resolución física (R) si usan los modos 2x o 4x. En general, para un modo Mx (M=1, 2 o 4) se tiene que:

$$\mathrm{Re} = \frac{\mathrm{R}}{\mathrm{M}} = \begin{cases} = \dfrac{360^\circ}{\mathrm{M}\cdot\mathrm{NL}} = \dfrac{360^\circ}{\mathrm{PPR}} \ \left[{}^\circ/_{\mathrm{pulso}}\right] \\ = \dfrac{2\pi}{\mathrm{M}\cdot\mathrm{NL}} = \dfrac{2\pi}{\mathrm{PPR}} \ \left[{}^{\mathrm{rad}}/_{\mathrm{pulso}}\right] \end{cases}$$

1.1.2. Cálculo de posición

Conocidos el número de líneas del codificador (NL), y el modo de funcionamiento (M), calcular la posición angular (α_C) de un codificador incremental correspondiente a una cuenta de pulsos es muy sencillo:

$$\alpha_C = \mathrm{cuenta}\cdot\mathrm{Re} = \begin{cases} = \mathrm{cuenta}\cdot\dfrac{360^\circ}{\mathrm{M}\cdot\mathrm{NL}} = \mathrm{cuenta}\cdot\dfrac{360^\circ}{\mathrm{PPR}} \ [^\circ] \\ = \mathrm{cuenta}\cdot\dfrac{2\pi}{\mathrm{M}\cdot\mathrm{NL}} = \mathrm{cuenta}\cdot\dfrac{2\pi}{\mathrm{PPR}} \ [\mathrm{rad}] \end{cases}$$

Hay que tener en mente que un codificador incremental no da una medida absoluta de posición, sino que proporciona la posición respecto a una posición inicial de referencia, para la cual la cuenta de pulsos es cero. Por eso, en un programa, antes de comenzar a realizar cálculos de la posición angular actual, hay que asegurar que el eje del codificador está en la posición de referencia y que la cuenta de pulsos es cero.

Si el codificador está instalado en el eje de salida de un actuador, después de un reductor de velocidad, como muestra la Figura 5-A, la posición angular que proporciona el codificador será la del eje final ($\alpha_S = \alpha_C$). En este caso, la posición angular del eje del motor (α_M) será proporcional a la del codificador según la relación de reducción, y la resolución en la medida en eje de salida (R_S) es la misma que la del codificador (Re):

$$\alpha_\mathrm{M} = \frac{\mathrm{N_M}}{\mathrm{N_S}}\alpha_\mathrm{S} \ \text{ donde } \alpha_\mathrm{S} = \alpha_\mathrm{C} \qquad\qquad \mathrm{R_S} = \mathrm{Re}$$

En cambio, si el codificador está instalado en el eje del motor, como muestra la Figura 5-B, la posición del codificador es la del eje del motor, y la posición del eje de salida y la resolución en ese eje se calcula en función de la relación de reducción:

$$\alpha_\mathrm{S} = \frac{\mathrm{N_S}}{\mathrm{N_M}}\alpha_\mathrm{M} \ \text{ donde } \alpha_\mathrm{M} = \alpha_\mathrm{C} \qquad\qquad \mathrm{R_S} = \frac{\mathrm{N_S}}{\mathrm{N_M}}\mathrm{Re}$$

En las expresiones anteriores, las relaciones de reducción están expresadas como el número de vueltas que da el eje del motor (N_M) frente al número de vueltas que da el eje de salida (N_S). La relación entre N_M y N_S se determina a partir de la configuración mecánica de los engranajes del reductor y sus dientes.

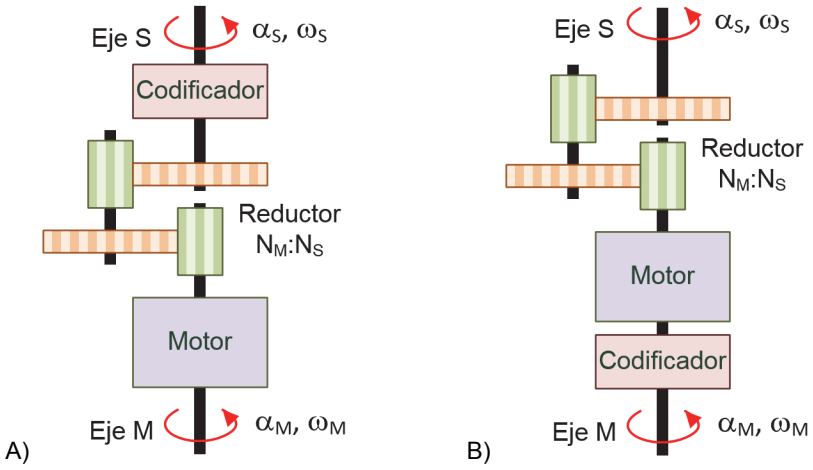


Figura 5. Opciones de instalación de un codificador: A) en el eje de salida, B) en el eje del motor.

1.1.3. Cálculo de velocidad

La velocidad de giro del eje del codificador (ω_C) se puede determinar inicialmente según su definición:

$$\omega_C = \frac{d\alpha_C}{dt} \approx \frac{\Delta\alpha_C}{T_S}$$

A partir de la definición, hay dos formas básicas de implementar en un programa el cálculo de la velocidad:

- Medida de la frecuencia de los pulsos de un canal del codificador. De forma más general, se puede medir la frecuencia de los flancos de las señales de los canales en función del modo de funcionamiento (1x, 2x o 4x). Así, la frecuencia se puede determinar contando el número de pulsos (Δcuenta) en un intervalo de tiempo (T_S). Multiplicando la frecuencia por la resolución efectiva de codificador (ver apartado 1.1.1), se tiene la velocidad. Este método funciona bien a velocidades altas, porque entonces se pude contar muchos pulsos en el intervalo de medida y el error porcentual cometido es menor.

$$\omega_C \approx \frac{\Delta\alpha_C}{T_S} = \frac{\Delta\text{cuenta} \cdot Re}{T_{Sf}} = \frac{\Delta\text{cuenta}}{T_{Sf}} \cdot \frac{360^\circ}{PPR}\left[{}^\circ/_s\right] = \frac{\Delta\text{cuenta}}{T_{Sf}} \cdot \frac{2\pi}{PPR}\left[{}^{rad}/_s\right] = \frac{\Delta\text{cuenta}}{T_{Sf}} \cdot \frac{60}{PPR}[RPM]$$

- Medida del periodo de un pulso del codificador. De forma más general, se puede medir el tiempo entre dos detecciones consecutivas de flancos de las señales de los canales en función del modo de funcionamiento (1x, 2x o 4x). La velocidad se determina como la variación de la posición mínima medible, la resolución efectiva (Re), en relación al tiempo entre los flancos detectados (T_S). Esta opción funciona mejor a velocidades bajas, cuando el tiempo entre flancos es grande y entonces se puede medir éste con menor error porcentual.

$$\omega_C \approx \frac{\Delta\alpha_C}{T_S} = \frac{Re}{T_{Sp}} = \frac{360^\circ}{PPR \cdot T_{Sp}}\left[{}^\circ/_s\right] = \frac{2\pi}{PPR \cdot T_{Sp}}\left[{}^{rad}/_s\right] = \frac{60}{PPR \cdot T_{Sp}}[RPM]$$

Si el codificador está instalado en el eje de salida después de un reductor de velocidad (Figura 5-A) la velocidad angular que proporciona el codificador será la del eje final ($\omega_S = \omega_C$), y la velocidad del eje del motor (ω_M) será proporcional a la del codificador según la relación de reducción:

$$\omega_M = \frac{N_M}{N_S}\omega_S \text{ donde } \omega_S = \omega_C$$

En cambio, si el codificador está instalado en el eje del motor (Figura 5-B), la velocidad del codificador es la del eje del motor, y la velocidad del eje de salida se calcula en función de la relación de reducción:

$$\omega_S = \frac{N_S}{N_M}\omega_M \text{ donde } \omega_M = \omega_C$$

1.2. Control simple de un motor de corriente continua

Los codificadores incrementales disponibles en Tinkercad-circuits forman parte de unos conjuntos que incluyen un motor CC, un reductor y un codificador, y por eso es necesario activar el motor para hacer girar el codificador. El uso de estos conjuntos es algo muy habitual en la práctica, porque simplifica mocho el montaje mecánico. Por estos motivos, conviene conocer los aspectos básicos para poner en marcha un motor de corriente continua (CC) y controlar su velocidad y dirección de giro.

Un motor CC típico se compone de un bobinado en el rotor y un campo magnético fijo en la armadura, además de un conmutador con escobillas que proporciona la corriente al bobinado, como se puede ver en la Figura 6. Este tipo de motor proporciona un par (T) en su eje que es función de la corriente que se le aplica a sus bobinas (i). La corriente que circula por las bobinas depende a su vez de la tensión que se aplica al motor (e), de la resistencia (R) e inductancia (L) del bobinado, y de la fuerza contra-electromotriz que produce el motor (e_b). Todo motor CC es un generador, y produce una fuerza contra-electromotriz en función de la velocidad de giro del eje (ω). Esa velocidad depende de la carga que el motor debe mover y el par que genera el motor, según el modelo dinámico de la carga.

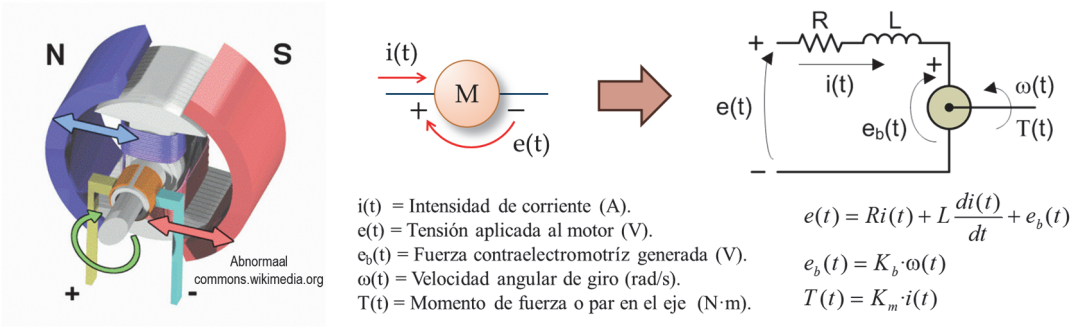


Figura 6. Estructura y modelo de funcionamiento de un motor CC típico.

Analizando el funcionamiento del motor descrito anteriormente, se concluye que, al variar la tensión aplicada al motor, se modifica la corriente de las bobinas, lo que a su vez cambia el par en el eje, y eso altera la velocidad de giro. Además, para invertir el sentido de giro hay que invertir el signo del par en el eje, lo que se consigue cambiando el sentido de la corriente por las bobinas, y para ello hay que invertir la polaridad de la tensión aplicada al motor. De este modo, la velocidad del eje y el sentido de giro dependen del valor y la polaridad de la tensión aplicada al motor.

Se puede ajustar la velocidad de un motor CC conectándolo a una fuente de alimentación ajustable. Y para cambiar el sentido del motor sin tener que desconectar, intercambiar y volver a conectar sus dos polos a la fuente de alimentación, lo más sencillo es utilizar un relé con un conmutador doble como elemento inversor de la polaridad.

Un relé es básicamente un conmutador, o conjunto de conmutadores, activado por un electroimán, como muestra la Figura 7. Aplicando una corriente a la bobina del electroimán (1), éste atrae y desplaza una palanca metálica (2) que cambia el estado de los conmutadores (3). Una gran ventaja del relé es que, con una corriente pequeña en la bobina (normalmente sin importa su sentido), se controlan unos conmutadores que son capaces de conducir una corriente alta, funcionando como amplificador. Además un relé aísla eléctricamente el circuito de la bobina del circuito de los contactos. Como inconveniente, un relé emplea elementos mecánicos y contactos que sufren desgastes, limitando bastante su vida útil.

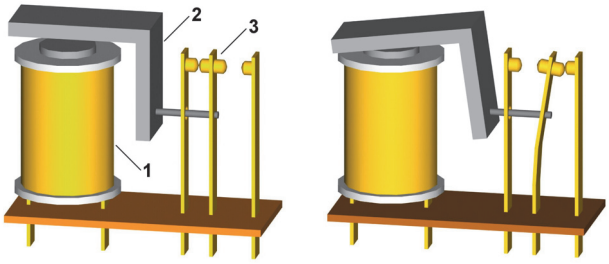


Figura 7. Ejemplo de relé con un solo conmutador: en la imagen izquierda el electroimán está desactivado, mientras que en la imagen derecha está activado.

La Figura 8 muestra un circuito en el que se conecta un motor (M) a una fuente de alimentación ajustable (V_M) para controlar su velocidad, utilizando un relé con conmutador doble (RL2) para invertir la polaridad y cambiar el sentido de giro. El circuito también incluye otro relé de conmutador simple (RL1) para cortar dejar pasar la corriente por el circuito, por lo que actúa a modo de interruptor para poner en marcha o parar el motor.

Además, el circuito de la Figura 8 incluye un fusible (F) para limitar la corriente que circula por el motor y así proteger éste. Como se ve en la Figura 8-A, en el esquema eléctrico los relés se pintan en su estado de reposo, en este caso con RL1 cortando el paso de la corriente. La Figura 8-B representa lo que ocurriría cuando se activa la bobina de RL1: el conmutador de este relé deja pasar la corriente de modo que el motor gire en sentido horario. Si se activase RL2 además de RL1, como se presenta en la Figura 8-C, la corriente también circularía por el motor, pero en sentido contrario al caso anterior, y el motor giraría en sentido antihorario.

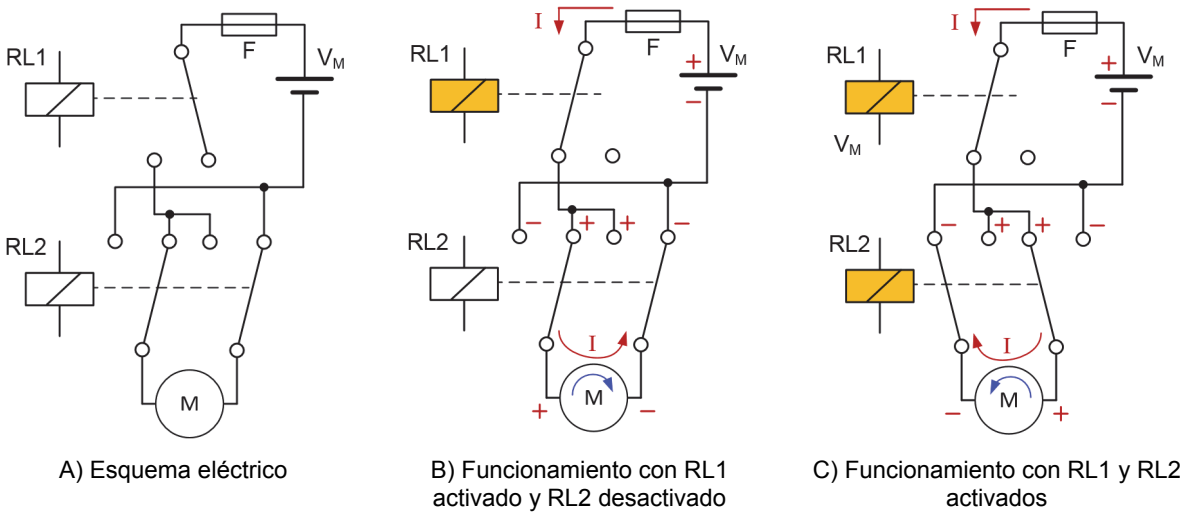


Figura 8. Ejemplo de esquema eléctrico de un circuito para activar un motor mediante relés.

En el simulador Circtuits de Tinkercad hay dos tipos de relés disponibles, equivalentes dos modelos de relés reales. La Figura 9 muestra el conexionado de ambos relés. Las características principales de éstos son:

- Relé LU-5-R: Conmutador de dos posiciones, tensión para activar la bobina de 5V, corriente por la bobina de 40mA, y corriente máxima de los contactos de 3A.
- Relé KS2E-M-DC5. Conmutador doble de dos posiciones, tensión de 5V para activar la bobina, corriente por la bobina de 40mA (para 5V), y corriente máxima de los contactos de 2A.

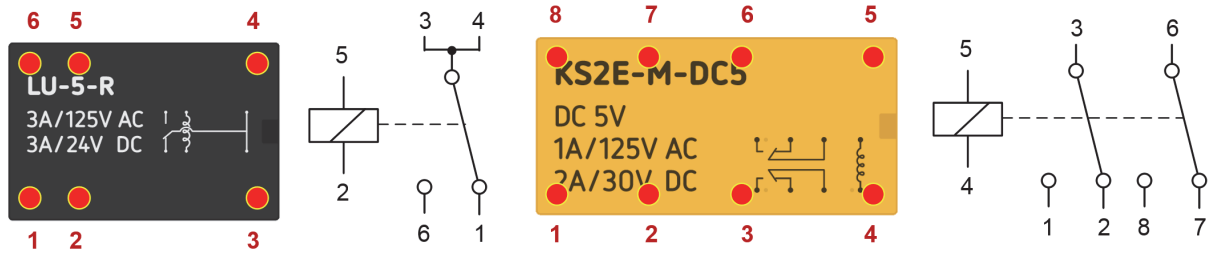


Figura 9. Modelos de relés disponibles en el simulador de circuitos de Tinkercad.

La bobina de un relé se puede activar conectándola directamente a una fuente de alimentación de la tensión adecuada, según las especificaciones del relé. Por ejemplo, una fuente de alimentación de 5V para los relés de la Figura 9. Pero la gran utilidad del relé es poder controlarlo desde la salida digital de un microcontrolador, aunque normalmente no se puede conectar directamente la bobina de un relé a la salida de un microcontrolador. En primer lugar, porque la corriente de una salida digital de un microcontrolador es habitualmente insuficiente para activar la bobina, aunque tenga la tensión adecuada. Y en segundo lugar, porque la bobina de un relé produce picos de corriente muy altos cuando se desactiva, debido a la descarga de la energía que ésta almacena como campo magnético, y esos picos de corriente pueden dañar el microcontrolador.

El método más simple y habitual para conectar un relé a un microcontrolador es hacer uso de un transistor bipolar como muestra el circuito de la Figura 10. En este circuito, se emplea el transistor Q1, de tipo NPN, como un amplificador para controlar el relé RL1, el diodo D1 se encarga de atenuar los picos de corriente que produce la bobina del relé en su descarga, y la resistencia R1 limita la corriente en la base del transistor. Q1 conduce (en saturación) y activa el relé cuando la salida digital está a nivel alto (5V). Si la salida digital está a nivel bajo (0V), Q1 está en corte y la bobina está desactivada.

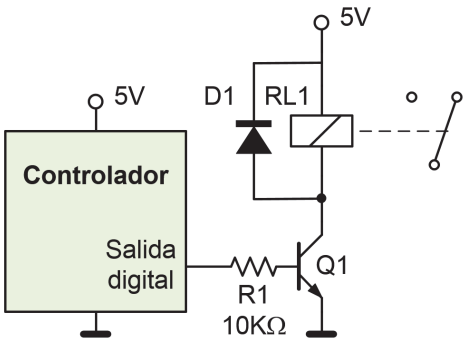


Figura 10. Ejemplo de circuito para controlar un relé mediante una salida digital de un microcontrolador usando un transistor NPN.

Entre los componentes disponibles en el simulador Circuits de Tinkercad, hay un transistor NPN y un diodo genéricos, los cuales pueden usarse para implementar un circuito como el de la Figura 10.

1.3. Codificador incremental E6B2 en el laboratorio

Para esta práctica, cada equipo de alumnos dispondrá de un codificador modelo E6B2-CWZ6C-100 de Omron, como el mostrado en la Figura 11, cuyas características principales son:

- Codificador incremental, con tres canales de salida: A, B y Z (o índice).
- NL = 100 líneas por vuelta.
- Frecuencia máxima de 30KHz, o 300 vueltas por segundo.
- Salidas NPN de colector abierto.
- Tensión de alimentación de 5V a 12V.
- Consumo máximo de corriente de 30mA.

Las características detalladas del codificador se pueden consultar en su *datasheet*, que está disponible en la carpeta de materiales de la asignatura.

Figura 11. Codificador E6B2-CWZ6C -100.

La Figura 12 muestra un esquema de ejemplo del montaje de una conexión directa del codificador a un Arduino Uno, mientras que el esquema eléctrico correspondiente se puede ver en la Figura 13.

En la parte izquierda de la a 12 se puede ver el conexionado del codificador E6B2. Las líneas A y B son las salidas correspondientes a los canales con los pulsos (desfasados un cuarto de ciclo entre A y B), la línea Z es la salida de índice que proporciona un pulso por vuelta y sirve para marcar una posición de referencia, y las líneas 0V y VCC son las entradas de alimentación, negativo y positivo respectivamente. El codificador también tiene una conexión al blindaje del cable (*shield*) marcada como GND. En los cables del codificador, no hay que confundir la línea de alimentación de 0V, que también es la referencia de las tensiones de las señales, con la línea GND. En la práctica, la línea GND se puede dejar conectar a la referencia de 0V.

Cabe destacar que las salidas de los canales A, B y Z del codificador son de tipo NPN de colector abierto., como muestra el esquema de la Figura 13. Por eso, cuando el codificador se conecta a entradas digitales de un microcontrolador, como por ejemplo el de una placa Arduino Uno, se requiere colocar unas resistencias de *pull-up* entre cada salida y la línea de tensión de alimentación que garanticen el nivel alto de las señales de pulsos. Considerando que la tensión de alimentación es de 5V, para las resistencias de *pull-up* conviene usar valores de 1KΩ a 4,7KΩ, antes que las resistencias internas de *pull-up* del microcontrolador, para que la corriente por las líneas de salida del codificador sea de unos pocos miliamperios (1 a 5mA).

En los esquemas de la Figura 12 y la Figura 13 también se han incluido tres LEDs que sirven para monitorizar visualmente los pulsos del codificador, cuando este gira despacio, y que no son imprescindibles.

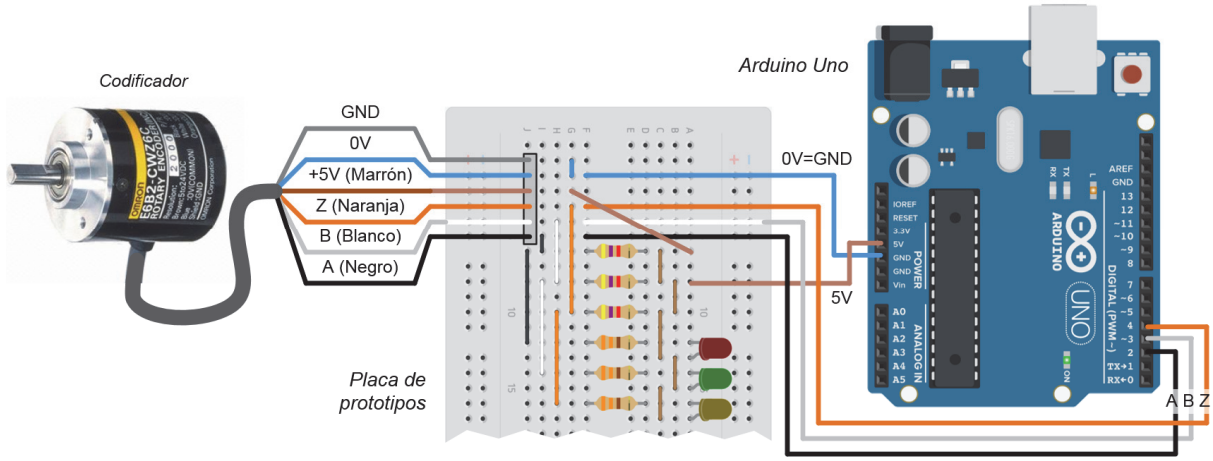


Figura 12. Esquema de montaje de ejemplo de conexión del codificador E6A2 a un Arduino Uno.

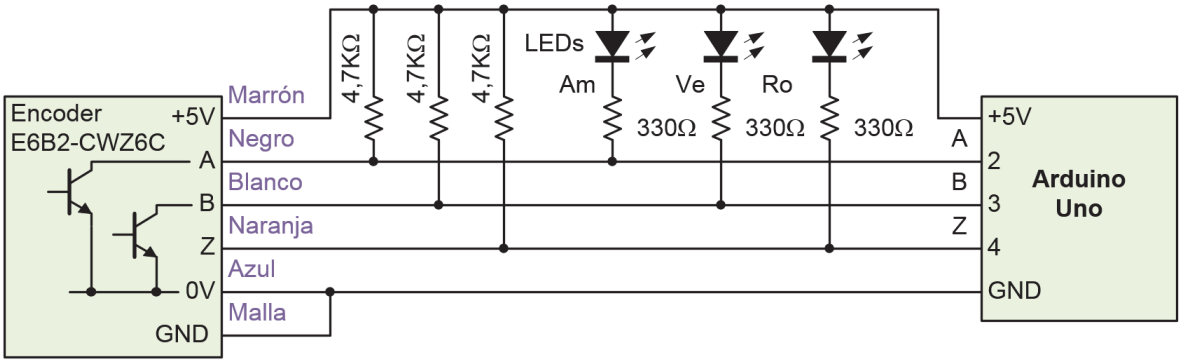


Figura 13. Esquema de eléctrico correspondiente al montaje de la Figura 12.

La conexión de un codificador con salidas NPN directamente al Arduino Uno implica que las señales digitales con los pulsos que llegan al Arduino estén invertidas con respecto a cómo son las señales lógicas que genera en codificador, como muestra el diagrama de la Figura 14. Esta inversión provoca un desplazamiento de medio ciclo en las señales A y B, que en la mayoría de aplicaciones no tiene importancia. Puede ser más importante que los niveles de la señal Z de índice también son invertidos. Si no se desea que las señales queden invertidas, hay que colocar un circuito inversor para cada señal, entre el codificador y el microcontrolador. Para el desarrollo de esta práctica de laboratorio, el desplazamiento de las señales no afectará al funcionamiento de los experimentos y no supone un inconveniente, por lo que las salidas del codificador se conectarán directamente al Arduino Uno, como en el esquema de la Figura 13, sin necesidad de inversores.

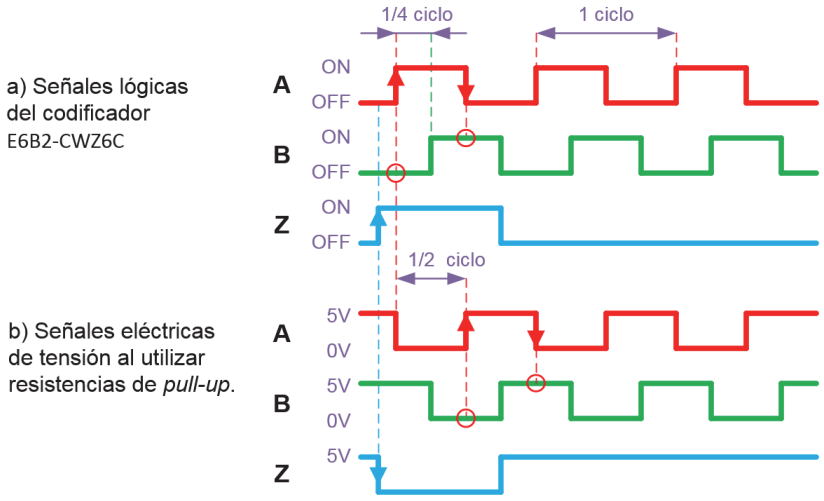


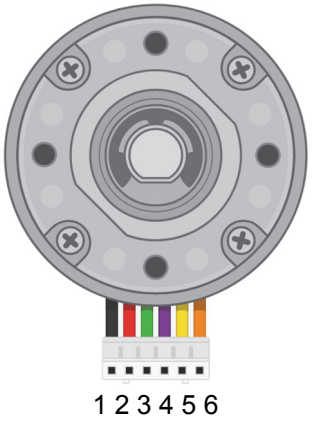
Figura 14. Señales en los canales del codificador E6B2, con salidas NPN y resistencias de pull-up.

1.4. Codificadores incrementales en Tiknercad

1.4.1. Opciones

En la lista de componentes del simulador Circuits de Tinkercad hay dos modelos muy parecidos de conjuntos motor-codificador-reductor, y ambos se llaman "Motor de CD con codificador". El primer modelo es de color plateado, y su aspecto y conexiones son los mostrados en la Figura 15. El segundo modelo es más pequeño que el primero, y de color negro, como se muestra en la Figura 16. Las diferencias entre los dos modelos, aparte de su color y tamaño, son básicamente tres: el orden de las conexiones, las opciones disponibles para las velocidades en el eje de salida, y la resolución del codificador.

Los dos modelos de motor-codificador-reductor de Tinkercad – Circuits tenen codificadores incrementales de dos canales (A y B), y no disponen del canal de índice. Por ello, si se desea una detección de una posición inicial o de referencia en la simulación, ésta debe llevarse a cabo mediante otro sensor independiente al codificador.



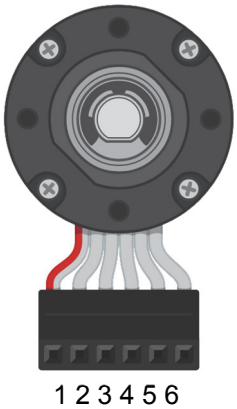
Conexiones:

1. Motor + (negro): Positivo del motor.
2. Motor – (rojo): Negativo del motor.
3. GND (verde): 0V o referencia del codificador.
4. B (violeta): Salida del canal B, en colector abierto.
5. A (amarillo): Salida del canal A, en colector abierto.
6. VCC (naranja): Alimentación de +5V del codificador.

Se puede escoger entre las siguientes RPM del eje de salida para para una tensión de alimentación de 12V para el motor: 12, 16, 23, 32, 45, 60, 84, 118, 165, 313, 437, 612, y 1.621.

Figura 15. Motor-codificador-reductor plateado disponible en el simulador Tinkercad – Circuits.

Para escoger la velocidad de salida del eje del motor con cualquiera de los dos modelos, hay que acceder a las propiedades del componente pulsando con el ratón sobre el mismo, como con cualquier otro componente del simulador Tinkercad – Circuits. Entonces, es posible escoger la velocidad en RPM (Revoluciones Por Minuto) para el eje de salida de entre una lista de posibles valores, como muestra la Figura 17-A. Cada modelo de motor-codificador-reductor dispone de diferentes opciones de velocidad, como se indica en la Figura 15 y en Figura 16. La velocidad escogida es la velocidad que tiene el eje de salida cuando el motor se alimenta a 12V, con lo que indirectamente se está seleccionando la relación de reducción del conjunto. Evidentemente, si el motor se alimenta con una tensión diferente de 12V, la velocidad del motor y del eje de salida cambiará. Pero la relación entre la tensión y la velocidad en el simulador es proporcional, y si, por ejemplo, el motor se alimenta a 6V, la velocidad será la mitad de la seleccionada en las propiedades.



Conexiones:

1. Motor + (rojo): Positivo del motor para giro antihorario.
2. B: Salida del canal B, en colector abierto.
3. VCC: Alimentación de +5V del codificador.
4. GND: 0V o referencia del codificador.
5. A: Salida del canal A, en colector abierto.
6. Motor – : Negativo del motor para giro antihorario.

Se puede escoger entre las siguientes RPM del eje de salida para para una tensión de alimentación de 12V para el motor: 26, 32, 38, 44, 52, 116, 142, 195, 280, 350, 416, 520, 624, 730 y 2.737.

Figura 16. Motor-codificador-reductor negro disponible en el simulador Tinkercad – Circuits.

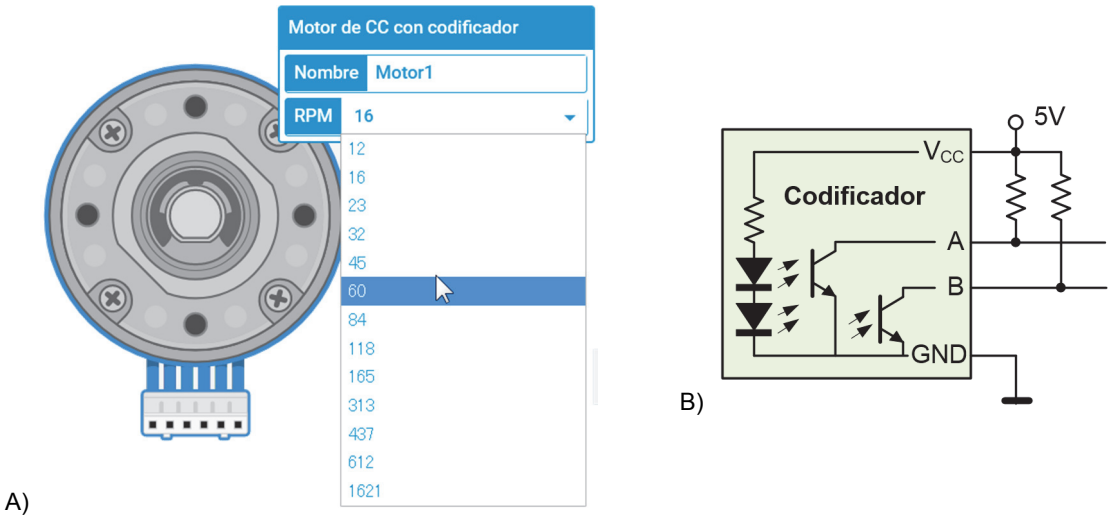


Figura 17. Imagen A: Opciones de la selección de velocidad para el motor-codificador-reductor plateado. Imagen B: esquema de conexión del codificador, con salidas en colector abierto para los canales A y B.

Igual que ocurre con el codificador real E6B2 (ver apartado 1.3), los dos modelos de codificadores disponibles en Tinkercad – Circuits tienen salidas de los canales A y B de tipo NPN de colector abierto, como muestra al esquema de la Figura 17-B. Por eso, en la simulación también hace falta añadir unas resistencias de *pull-up* entre las conexiones de los canales y la línea de alimentación de +5V que garanticen unos niveles de tensión correctos en las señales de pulsos. Se pueden usar resistencias de *pull-up* entre 1KΩ y 4,7KΩ como con el codificador real, en vez de las resistencias *pull-up* internas del Arduino Uno.

1.4.2. Conexiones básicas

La Figura 18 muestra un circuito de ejemplo que tiene las conexiones básicas para verificar el funcionamiento del conjunto motor-codificador-reductor plateado. El motor está conectado directamente a una fuente de alimentación variable, de forma que, ajustando la tensión de salida de ésta, se puede variar la velocidad del motor. Concretamente, se observa que el motor se alimenta a 6V, y que el motor indica una velocidad de 8 RPM en eje de salida, según indica el texto que hay en el motor sobre el tornillo inferior izquierdo. Esto es porque el motor está configurado para una velocidad en el eje de salida de 16 RPM para 12V.

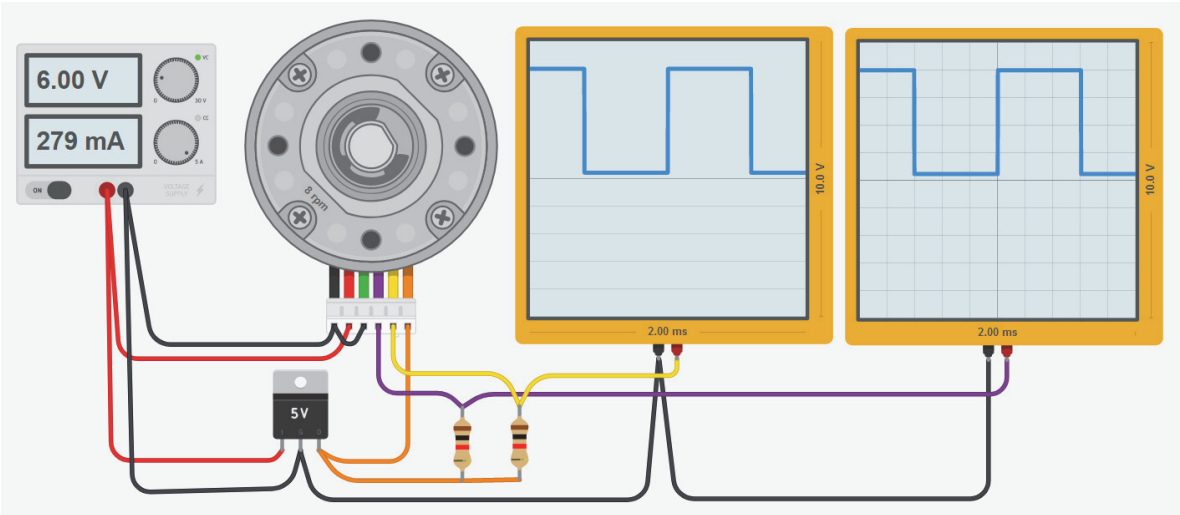


Figura 18. Conexiones de prueba del motor-codificador-reductor plateado.

Además, en el circuito de la Figura 18 se ha incluido un regulador integrado de 5V para obtener la tensión de alimentación para el codificador a partir de la fuente de alimentación variable que alimenta el motor. Como las

salidas del codificador para los canales A y B son de colector abierto, requieren estar conectadas a la alimentación de 5V a través de resistencias de *pull-up* de 1KΩ. Los osciloscopios de la Figura 18 están ajustados para 1V y 200µs por división para los ejes vertical y horizontal respectivamente. De este modo, las señales de pulsos mostradas en los osciloscopios tienen un periodo de 1,2ms y una amplitud de 4V. Se observa también que las señales de pulsos para los canales A y B parecen iguales, sin un desfase aparente de tiempo entre ellas, pero esto es debido a que cada canal se muestra en un osciloscopio independiente, con su propia base de tiempos y detección del cero de la onda, y los osciloscopios no están sincronizados.

La Figura 19 muestra un circuito idéntico al anterior, salvo que en este caso se utiliza el conjunto motor-codificador-reductor negro. En este circuito, el indicador de velocidad que tiene el motor, sobre su tornillo inferior izquierdo, muestra 13 RPM, porque el motor está configurado para una velocidad de salida de 26 RPM para 12V. Las señales en los osciloscopios tienen un periodo de un poco más de 8ms y una amplitud de 4V.

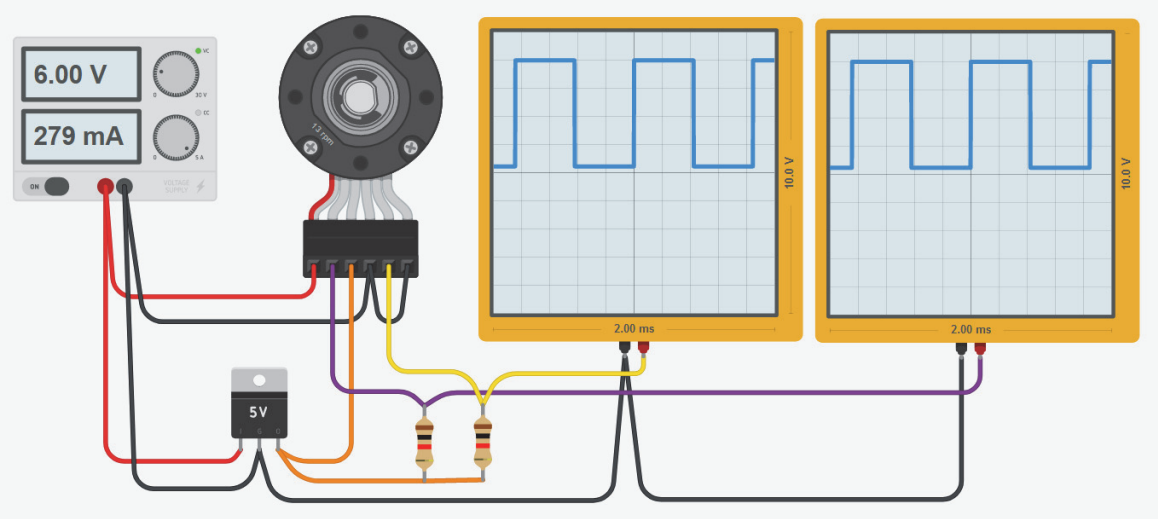


Figura 19. Conexiones de prueba del motor-codificador-reductor negro.

1.4.3. Conexión con Arduino

La conexión entre un Arduino Uno y los canales del codificador incremental de un conjunto motor-codificador-reductor dentro del simulador Tinkercad – Circuits es muy sencilla. Como muestra el esquema eléctrico de la Figura 20. Basta con añadir dos conexiones entre los canales del codificador y dos pines digitales del Arduino configurados como entradas, sin olvidar incluir las dos resistencias de *pull-up* entre cada canal y la alimentación de 5V del codificador, que puede ser la que proporciona el Arduino Uno en su salida de 5V. Si además se desea asociar unas rutinas de interrupción que se activen con los flancos de las señales de pulsos de los canales, necesariamente los pines usados deben ser el 2 y el 3, que corresponden con las dos únicas interrupciones externas INT0 e INT1 de que dispone el Arduino Uno. La programación de estas interrupciones se detalla en el apartado 1.5.1 de este manual.

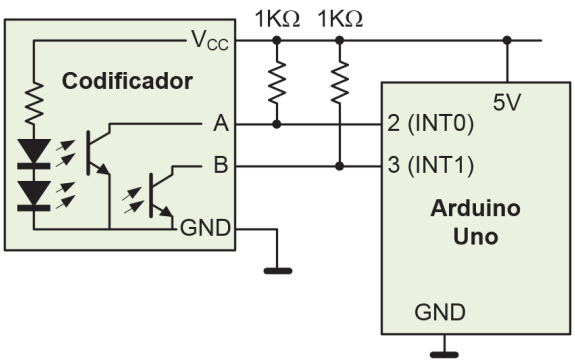


Figura 20. Conexión de un codificador incremental a un Arduino Uno; el codificador tiene salidas de colector abierto para los canales A y B y por eso se requieren dos resistencia de pull-up.

Si, además de conectar el codificador, se desea gobernar el motor para controlar su activación y sentido de giro de una forma cómoda mediante interruptores, o incluso desde el propio Arduino, sin necesidad de parar la simulación para cambiar las conexiones, entonces se requieren relés y otros componentes. Así, la complejidad de montaje está en la conexión del motor, no en la del codificador, y depende del grado de automatización que se quiera para el motor. A continuación se describen dos ejemplos de conexión: en el primero el motor se puede controlar manualmente durante la simulación mediante dos interruptores, y en el segundo el motor se controla desde el programa del Arduino Uno.

Un aspecto muy importante a tener en cuenta con la simulación de motores en Tinkercad – Circuits es que ésta falla al actualizar el estado del motor-codificador-reductor en algunos casos, en los cuales, aunque el motor tienen tensión y aparece indicada la velocidad en el mismo, su eje no se mueve y el codificador no genera pulsos. En concreto, esto ocurre cuando deja de circular corriente por el motor, por ejemplo porque directamente no se aplica tensión al motor al iniciar la simulación, o porque se deja de aplicar tensión al motor durante la simulación. Para evitar esto, en vez de cortar totalmente la corriente del motor simulado cuando se quiere pararlo, hay que hacer circular una corriente mínima por él, que sea insuficiente para que se mueva significativamente. Para ello se puede usar una resistencia de valor elevado, por ejemplo 100KOhm, en paralelo con el interruptor o conmutador que se usa para activar el motor. Un motor real no se movería con esa resistencia en serie porque necesita una corriente alta para arrancar.

La Figura 21 muestra el esquema eléctrico del circuito de ejemplo que permite el control manual del motor. Para ello, primero se incluye el conmutador-interruptor “Activar” conectado en serie entre el polo positivo de la fuente de alimentación del motor y el resto del circuito del motor, de modo que éste interruptor permite parar o poner en marcha el motor se según se abra o se cierre. En paralelo con el interruptor “Activar” hay una resistencia de 100KΩ, que sirve para garantizar la corriente mínima por el motor cuando el interruptor Activar está abierto, de forma que la simulación del motor no falle. Entre el interruptor “Activar” y el motor, se ha colocado un relé de conmutador doble para poder invertir el sentido de giro, según se describe en el apartado 1.2 de este manual. La bobina del relé se puede activar manual mente con el conmutador-interruptor “Sentido”, sobre el que se puede actuar durante la simulación para cambiar el sentido de giro del motor.

En el esquema eléctrico de la Figura 21 también se ha añadido el fusible F, para proteger el motor ante un posible exceso de corriente, y el pulsador “Inicio” que sirve para indicar alguna acción al programa de Arduino.

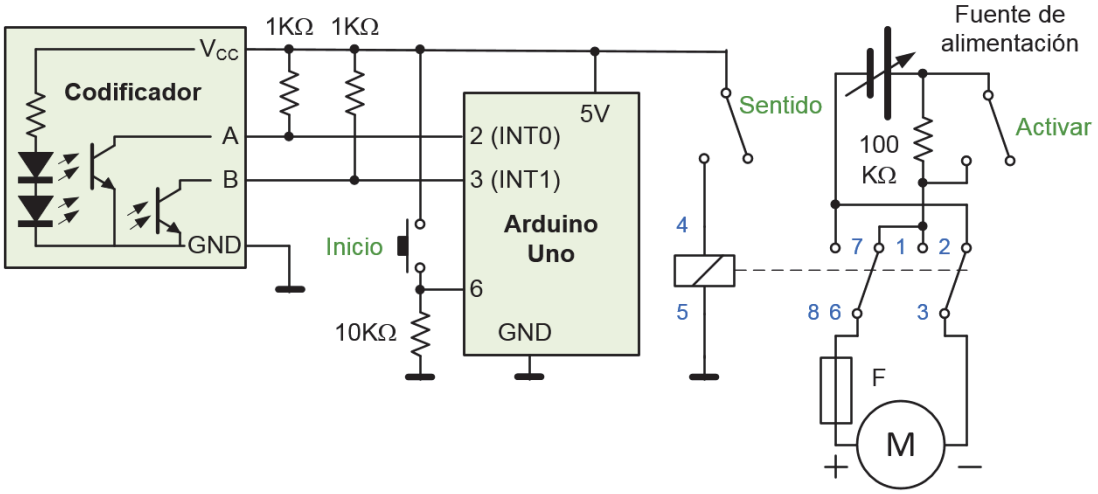


Figura 21. Esquema eléctrico de ejemplo para la conexión del motor-codificador-reductor gris al Arduino Uno, con un control manual del motor mediante dos interruptores: Activar y Sentido.

La Figura 22 muestra un ejemplo de montaje en el simulador Tinkercad – Circuits correspondiente al circuito descrito en el párrafo anterior. Las únicas diferencias entre el esquema eléctrico y el montaje es que, en el segundo, se ha incluido un voltímetro para conocer la tensión aplicada al motor (lo que es muy útil para ver el estado del motor), y en cambio no se ha podido incluir el fusible F porque este tipo de componente no está disponible en el simulador.

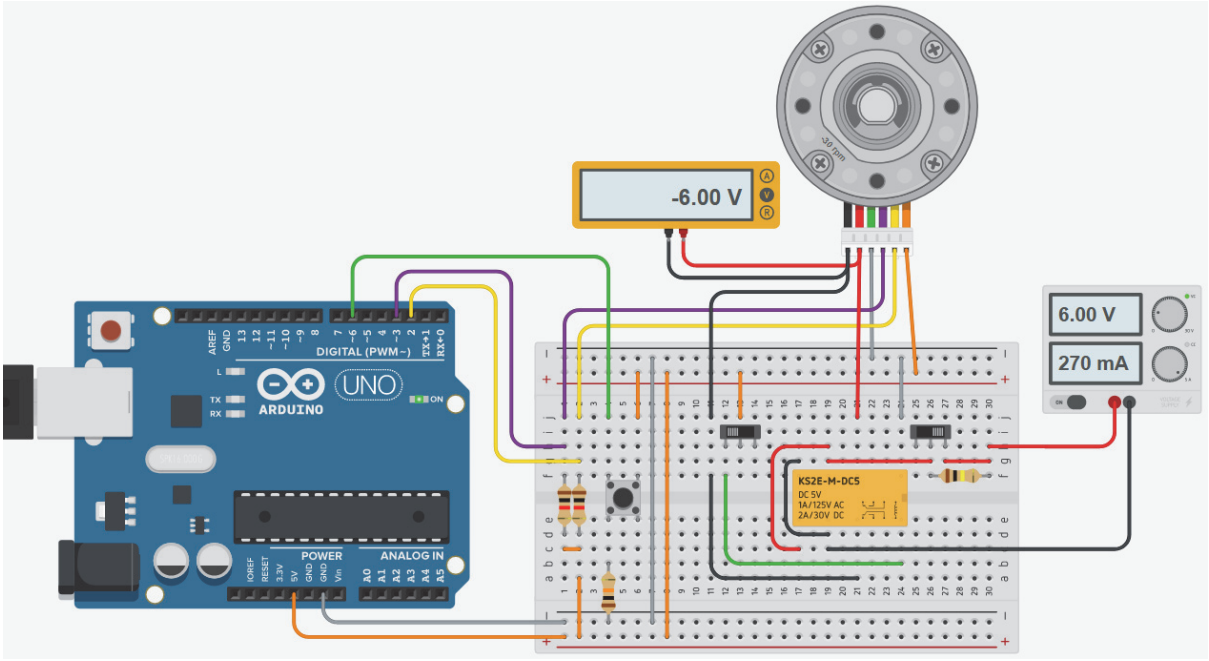


Figura 22. Ejemplo de montaje en Tinkercad – Circuits para el esquema eléctrico de la Figura 21.

Si se desea controlar el motor desde el programa del Arduino Uno, lo más sencillo es utilizar dos relés cuyas bobinas se gobiernen desde dos pines digitales del Arduino configurados como salidas, conforme a las explicaciones del apartado 1.2. La Figura 23 muestra el esquema de un circuito de ejemplo que incluye los dos relés: RL1 para parar o poner en marcha el motor y que está gobernado por el pin 4, y RL2 para cambiar el sentido de giro y que está gobernado por el pin 5. Entre los pines del Arduino y las bobinas de los relés se han incluido unos transistores NPN como amplificadores y unos diodos para atenuar las corrientes generadas por las bobinas, según lo explicado en el del apartado 1.2. Como añadido, se ha colocado también un LED en paralelo a la bobina de cada relé para tener información visual sobre cuando está activado cada relé. En la Figura 24 se puede ver un ejemplo con el montaje correspondiente al esquema eléctrico de la Figura 23.

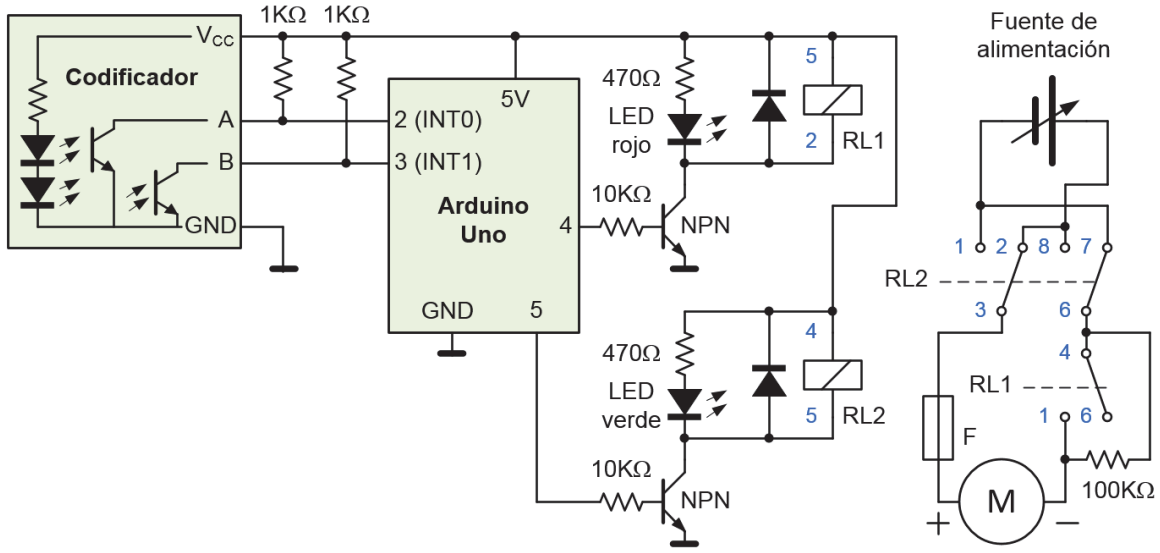


Figura 23. Esquema eléctrico de ejemplo para la conexión del motor-codificador-reductor gris al Arduino Uno, con un control automático del motor desde el programa del Arduino.

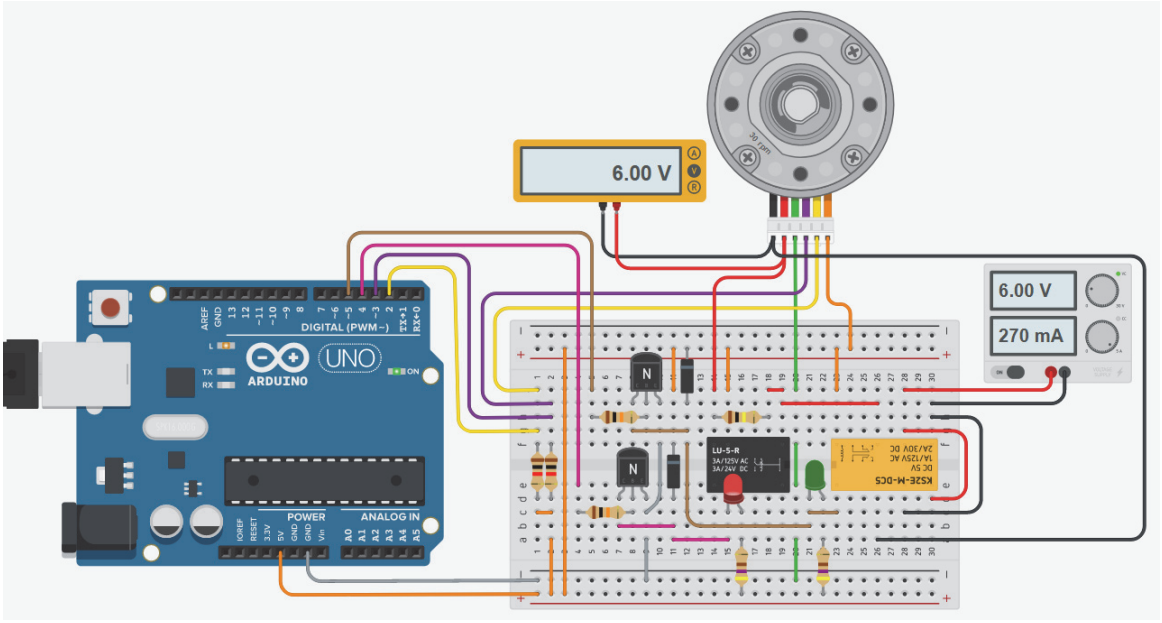


Figura 24. Ejemplo de montaje en Tinkercad – Circuits para el esquema eléctrico de la Figura 23.

1.5. Aspectos sobre programación de codificadores incrementales

En los puntos de este apartado se resumen los principales aspectos sobre la programación de interrupciones, acceso a los puertos de entrada y salida, y temporizaciones, que son importantes tener en cuenta cuando se gestiona un codificador incremental. Aunque las explicaciones se centran en la plataforma Arduino, en general son también portables a otros microcontroladores sustituyendo los pines, funciones, y tipos de datos por los correspondientes.

1.5.1. Gestión de interrupciones

La gestión de interrupciones suele considerarse una característica avanzada de la programación de procesadores, aunque con el entorno de Arduino es relativamente fácil configurar las interrupciones para algunos eventos. En concreto, las interrupciones más fáciles de utilizar con Arduino son las externas, que se asocian a un pines digitales que dispone de esa opción y están configurados como entradas. De esa forma, los cambios en los niveles de tensión de un pin con interrupción dispara inmediatamente una rutina de servicio de interrupción (ISR) específica, interrumpiendo el flujo normal del programa. Básicamente, una ISR es una función más del programa, pero definida específicamente como ISR.

La siguiente lista describe los pasos que conviene seguir para poner en marcha las interrupciones externas con una placa Arduino de una forma robusta y fiable.

1. Decidir que pin se desea utilizar para captar eventos mediante una interrupción. Con el Arduino Uno, se dispone de dos pines con opción de interrupción: la interrupción INT0 asociada al pin 2 y la interrupción INT1 asociada al pin 3. Para cada uno de estas interrupciones, se puede escoger que se disparen cuando la entrada cambia de nivel bajo a alto, de alto a bajo, o en ambos casos.
2. Definir qué operaciones va a realizar la función ISR de la interrupción. Una función ISR debe ser muy corta y realizar operaciones básicas, ya que interrumpe el flujo normal del programa y puede afectar a determinadas operaciones, sobre todo las que dependen del tiempo o de otras interrupciones. Hay que tener en mente que microcontrolador sencillo como el del Arduino Uno deshabilita las interrupciones mientras está atendiendo ya una y no gestiona prioridades de interrupciones, por lo que, durante la ejecución de la función ISR de una interrupción, no se pueden atender las demás (aunque sí se detectan y quedan pendientes). Aunque es posible activar las interrupciones dentro de una función ISR, esto puede provocar problemas.

No es recomendable incluir bucles de muchas iteraciones dentro de una interrupción. Y no se deben usar funciones de comunicaciones (como las que usan el objeto Serial), y mucho menos funciones de pausa del programa como "delay()", no solo por el retraso que pueden provocar, sino porque esas funciones dependen a su vez de otras interrupciones. Sí que se puede utilizar "delayMicroseconds()" ya que no ejecuta interrupciones, pero no es recomendable alargar la duración de la interrupción más de lo imprescindible. Otras funciones de tiempo como "millis()" o "micros()" devuelven la cuenta actual, pero dejan de actualizarse.

En muchas ocasiones basta con que la función ISR simplemente active un indicador booleano, cambie el valor de una variable, o realice unas operaciones básicas.

3. Declarar la función ISR. En Arduino una función ISR se declara como cualquier otra función, pero no debe tener parámetros y debe ser necesariamente de tipo "void", es decir, tampoco debe devolver ningún resultado. Por ejemplo "`void INT0_handler() {…}`".

4. Una función ISR se comunica con el resto del programa mediante variables globales, ya que no puede tener parámetros. Las variables globales usadas por funciones ISR se deben declarar de una forma especial, usando la sentencia "volatile" antes del tipo, como por ejemplo "`volatile long count`;". Esta sentencia fuerza a que el compilador almacene el valor de variable siempre en la misma posición de memoria, y no en copias en registros temporales, como se suele hacer para mejorar el rendimiento. Esto es necesario porque en cualquier momento se puede activar una función ISR que requiere acceder a la variable y su contenido debe estar actualizado. De igual modo, el programa principal que hace uso de la variable debe también disponer del valor actualizado.

5. Asociar la función ISR a la interrupción. Esto se hace habitualmente en la función "setup()". Pero también es posible asociar y desasociar una función ISR en cualquier otra parte del programa, lo que puede ser útil para activar o desactivar así solo una interrupción concreta, o para cambiar la función ISR que atiende una interrupción en función del programa. Incluso se puede asociar o desasociar una función ISR dentro de la misma u otra función ISR. Para asociar o desasociar una función ISR se dispone de estas funciones:

   - **attachInterrupt(interrupt, función_ISR, mode).** Asocia la función con nombre "función_ISR" a la interrupción cuyo número es "interrupt". El número de interrupción puede ser 0 (INT0) o 1 (INT1) para el Arduino 1, estando esos números de interrupción asociados a los pines digitales 2 y 3 respectivamente. El parámetro "mode" indica en qué situación se debe ejecutar la función ISR, y las opciones más habituales son "RISING" para detectar el cambio de nivel bajo a alto, "FALLING" para el cambio de alto a bajo, y "CHANGE" para cualquiera de los dos cambios anteriores.
   - **digitalPinToInterrupt(intPin)**. Esta función devuelve el número de interrupción que corresponde con un el pin digital dado por "intPin". Es aconsejable usarla junto con la función anterior de la siguiente forma: "`attachInterrupt(digitalPinToInterrupt(intPin), ISR_external, FALLING)`". De este modo, no es necesario conocer el número de interrupción, solo el número de pin.
   - **detachInterrupt(interrupt)**. Desasocia la función ISR para el número de interrupción indicado. De esta manera, la interrupción deja de atenderse.

6. Activar las interrupciones. Por defecto, las interrupciones están activadas al comenzar la ejecución de un programa de Arduino, por lo que no hace falta activarlas explícitamente.

7. Otras partes del programa fuera de la interrupción pueden acceder a las variables globales de tipo "volatile". Se pueden usar estas variables para intercambiar datos entre la función ISR y otras partes del programa durante la ejecución del mismo. También se puede leer una variable "volatile" para comprobar si se ha disparado la interrupción, y realizar así operaciones complejas que no se pueden incluir en la función ISR.

8. Localizar las posibles *secciones críticas* del código. Cuando las variables globales "volatile" no son de un tipo entero básico que encaja en un registro del procesador, o cuando se necesita un grupo de variables

para intercambiar información entre el programa principal y la función ISR, las operaciones de lectura o escritura de esos grupos de valores en el programa principal se denominan *secciones críticas*.

Por ejemplo, en un Arduino Uno, cuyo procesador es de 8 bits, los tipos enteros básicos son "byte", "bool", "char", "int8_t" y "uint8_t". Por eso la definición de tipos de variable de mayor tamaño, como por ejemplo "int", "long", "float", etc. implica que estas ocupen varias posiciones de memoria o varios registros, y además el acceso a las mismas necesita de un conjunto de operaciones de lectura o escritura. Y ese conjunto de operaciones puede ser interrumpido como cualquier otra parte del programa si las interrupciones están habilitadas.

Por eso, es necesario desactivar la interrupción asociada a una función ISR durante una *sección crítica* que accede a variables de gran tamaño usadas en la misma función ISR, incluso aunque las variables se hayan definido como "volatile", para asegurar que esas variables están correctamente actualizadas cuando se usan. Si no se desactivase la interrupción, podría ocurrir que la función ISR alterase una parte de una variable, o del grupo de variables, mientras que la sección crítica está accediendo a otra parte. Así, el acceso a las variables "volatile" fuera de la función ISR se debe programar como un "bloque atómico" no interrumpible.

Para definir un bloque atómico en un programa se puede desactivar temporalmente la interrupción implicada según lo explicado en el punto 5, aunque es más sencillo desactivar y activar todas las interrupciones si el bloque atómico es corto. Lo segundo se puede hacer en Arduino mediante las siguientes funciones:

- **noInterrupts()**. Desactiva todas las interrupciones. Hay que tener en cuenta que esto implica que ciertas funciones del entorno Arduino dejan de funcionar, como por ejemplo las comunicaciones o las cuentas de tiempo. Por eso hay que volver a activar las interrupciones lo antes posible.
- **interrupts()**. Habilita todas las interrupciones.

Una alternativa más clara que el uso de las funciones anteriores es emplear la macro estándar de ATOMIC_BLOCK. Pero, aunque esta macro si está soportada en el compilador del IDE de Arduino, no lo está en el compilador que usa el simulador Tinkercad.

A continuación se muestra un ejemplo sencillo de uso de interrupciones que tiene en cuenta todos los aspectos de la lista de pasos anterior. El programa realiza la cuenta de los pulsos que se reciben en el pin digital 2 configurado como entrada con interrupción externa. La cuenta se guarda en una variable "volatile" de tipo "unsigned long", de 32 bits, cuya lectura en el programa principal se define como un bloque atómico. A la vez, en el programa principal se activa y desactiva el LED del pin 13 en función de si la cuenta actual tiene un número par o impar.

```
const byte intPin = 2;                // Pin for external interruption (2 or 3 for Uno)
const byte ledPin = 13;               // LED

volatile unsigned long count = 0L;    // To count interruption events
unsigned long temp_count;             // To read the count of pulses

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(intPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(intPin), ISR_external, RAISING);
}

void loop() {
   noInterrupts();       // Begin of atomic block: disable interruptions
   temp_count = count;   // Read the pulse count: long is not a basic integer
   interrupts();         // End of atomic block: enable interruptions
```

```
    // Turn on the LED only when the count is an even number (remaining is 0)
    digitalWrite(ledPin,(temp_count % 2 > 0)? LOW:HIGH);
}
void ISR_external() {    // ISR function
  count++;               // The code for the ISR function must be simple
}
```

La gestión de otros tipos de interrupciones, como las asociadas a los temporizadores/contadores hardware, comunicaciones, ADC, Watcdog, etc. es más compleja que la gestión de las interrupciones externas explicada aquí, y para esas interrupciones lo más adecuado es recurrir a bibliotecas que facilitan su uso.

Si se desea una información más detallada sobre el uso y funcionamiento de las interrupciones de Arduino, o ejemplos más avanzados de programación de interrupciones, las siguientes fuentes pueden ser muy útiles:

- Qué son y cómo usar interrupciones en Arduino, Luis Llamas, 2016:
  https://www.luisllamas.es/que-son-y-como-usar-interrupciones-en-arduino/
- Referencia de la web de Arduino/Genuino para la función “attachInterrupt()”:
  https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/
- Variable Scope & Qualifier: Volatile (Arduino Reference)
  https://www.arduino.cc/reference/es/language/variables/variable-scope-qualifiers/volatile/
- Guía de Nick Gammon sobre las interrupciones del Arduino Uno:
  http://gammon.com.au/interrupts
- Reading Rotary Encoders (Arduino)
  http://playground.arduino.cc/Main/RotaryEncoders
- Atomically and Non-Atomically Executed Code Blocks
  https://www.nongnu.org/avr-libc/user-manual/group__util__atomic.html

1.5.2. Acceso directo a los pines de entrada y salidas digitales

Las funciones “digitalWrite()” y “digitalRead()” de Arduino están desarrolladas para que funcionen en todas las placas oficiales de Arduino, y en muchas otras placas compatibles con Arduino. De este modo, un programa que usa estas funciones para acceder a los pines digitales es portable entre diferentes placas Arduino. Pero para conseguir eso, estas funciones incluyen muchas líneas de código, y aunque ese código está optimizado, una simple operación de escritura o lectura sobre un pin digital implica la ejecución de muchas instrucciones.

Por ejemplo, la función “digitalRead()” tarda sobre 5µs en ejecutarse en un Arduino Uno con el microcontrolador ATmega328 a 16MHz. Este puede ser un tiempo muy grande para algunas aplicaciones concretas, como la lectura de pulsos de un codificador que gira a gran velocidad. En cambio, con un acceso directo al puerto de entradas y salidas digitales en el mismo microcontrolador, la lectura de un bit se puede hacer directamente con dos instrucciones de ensamblador que en total duran solo 125ns.

Así, se puede mejorar mucho el tiempo de respuesta y la memoria de programa utilizada usando operaciones para acceso directo a los pines de entrada y salida, en vez de usar las funciones estándar de Arduino. El problema es que el código necesario para un acceso directo es específico de cada microcontrolador y no sirve para otros microcontroladores soportados en la plataforma Arduino. De esta forma hay que escoger entre eficiencia o portabilidad, según las necesidades de la aplicación.

Para acceder de forma directa a los puertos de entradas y salidas digitales de Arduino, hay que trabajar con operaciones de bit sobre los registros de los puertos. Por ejemplo, en un Arduino Uno con microcontrolador Atmega328p, los pines 0 a 7 se conectan a los bits 0 a 7 del puerto D (PORTD), el cual se puede gestionar con estos registros:

- **PORTD** (lectura y escritura): permite escribir en los bits del puerto D como salidas, así como leer los últimos valores escritos en los bits de las salidas.

- **PIND** (lectura): permite leer los bits del puerto D configurados como entradas.
- **DDRD** (lectura/escritura): define si un bit del puerto D es de entrada (0) o salida (1).

A continuación se muestran algunos ejemplos sencillos de acceso al puerto D de un Arduino Uno:

```
// Pone a 1 los bits 4 a 7 del registro DDRD (deja 0 a 3 sin tocar)
// Así los pines 4 a 7 quedan como salidas, los pines 0=TD y 1=RX no se tocan
// La B que antecede al valor "1111000" indica que éste está expresado en binario
DDRD |= B11110000;  // Es lo mismo que DDRD = DDRD | B11111100;

// Pone a 0 los bits 2 y 3 del registro DDRD (deja el resto sin tocar)
// Así los pines 2 y 3 quedan como entradas, los demás pines no se tocan
DDRD &= B111100111;  // Es lo mismo que DDRD = DDRD & B11111100;

// Pone a 0 los bits 4 a 7 del puerto D (deja 0 a 3 sin tocar)
// Así los pines 4 a 7 de salida pasan a nivel bajo todos a la vez
PORTD &= B00001111;  // Es lo mismo que PORTD = PORTD & B00000011;

// Pone a 1 los bits 4 a 7 del puerto D (deja 0 a 3 sin tocar)
// Así los pines 4 a 7 de salida pasan a nivel alto todos a la vez
PORTD |= B11110000;  // Es lo mismo que PORTD = PORTD | B00000011;

// Comprueba si los bits 2 o 3 del puerto D son 1
// Así detecta si alguno de los pines 2 y 3 de entrada está a nivel alto
if (PIND & B00001100)
  entradas = true;
// Comprueba si el bit 2 del puerto D es 1, para detectar si el pin 2
// de entrada está a nivel alto. En ese caso, pone a 1 el bit 4 del puerto D,
// para poner a nivel alto el pin 4 configurado como salida
if (PIND & B00000100) {
  PORTD |= B00010000;
}
```

Para saber más sobre el acceso directo a las entradas y salidas digitales se pueden consultar estas referencias:

- Port Register, Arduino.
  https://www.arduino.cc/en/Reference/PortManipulation
- El Registro PORT (puerto) de Arduono, Losé Villalaz, PanamaHitek, 2014.
  http://panamahitek.com/registro-port-puerto/
- Faster IO on the Arduino, SK Pang Electroncis, 2011.
  http://skpang.co.uk/blog/archives/323

1.5.3. Medida de tiempo y eventos de temporizador

En un programa de Arduino conviene usar las funciones "millis()" o "micros()" para ejecutar operaciones después de un tiempo, frente al uso de las funciones "delay()" y "delayMicroseconds()" que bloquean la ejecución del programa durante el tiempo de espera indicado.

- **millis**(). Devuelve el número de milisegundos desde que la placa Arduino comenzó a ejecutar el programa actual como un valor "unsigned long". El valor devuelto puede sobrepasar el límite del tipo "unsigned long" y volver a contar desde cero aproximadamente cada 50 días.
- **micros**(). Devuelve el número de microsegundos desde que la placa Arduino comenzó a ejecutar el programa actual como un valor "unsigned long". El valor devuelto puede sobrepasar el límite del tipo "unsigned long" y volver a contar desde cero aproximadamente cada 70 minutos. En un Arduino Uno, la resolución en el valor devuelto es de 4µs.

Hay que tener en cuenta que las cuentas de tiempo de "millis()" o "micros()" no se actualizan dentro de una función ISR. Aun así, estas funciones pueden ser útiles dentro de una función ISR para calcular el tiempo aproximado entre llamadas a esa función.

Uno de los puntos fuertes de los microcontroladores, como los utilizados en las placas de la plataforma Arduino, es que incluyen módulos de temporizadores y contadores hardware, que permiten una cuenta de tiempo muy precisa, la generación de eventos (interrupciones o salidas PWM) de forma periódica, y el contaje de eventos (interrupciones de entradas digitales) sin pérdida de cuentas. Sin embargo, el uso de estos temporizadores suele ser complejo, pues implica trabajar con múltiples registros de configuración, que además suelen ser muy particulares en cada modelo de microcontrolador. Además, el lenguaje C/C++ de Arduino no incluye funciones para gestionar los temporizadores de forma general. Por eso, la mejor opción para sacar provecho de los temporizadores para aplicaciones concretas es recurrir a bibliotecas que encapsulan su funcionamiento y simplifican mucho su utilización. Esta es la forma común de trabajar con el IDE de Arduino y otros entornos de desarrollo para las placas de Arduino. Pero el compilador de Tinkercad – Circuits no admite usar bibliotecas que no sean las básicas que incorpora, y, en el momento de escribir este manual, no incluía ninguna biblioteca para usar los temporizadores hardware. De todas formas, para el desarrollo de las actividades de esta práctica no se requiere el uso de temporizadores.

2. Experimentos

2.1. Actividades básicas (7 puntos)

Se proponen tres actividades básicas, que hay completar para que se evalué la práctica, y que suman hasta 7 puntos. En cada actividad hay que resolver varias tareas (a, b, c, etc.).

2.1.1. Características de un codificador de Tinkercad (1,5 puntos)

a) Crear un circuito en el simulador Circuits de Tinkercad con el motor-codificador-reductor plateado grande, de forma que el motor se alimente con una fuente de alimentación ajustable, el codificador se alimente con 5V, y las señales de los canales A y B se vean en osciloscopios. Capturar una imagen del montaje para añadirla al informe.

b) Configurar el circuito con las tensiones de la fuente de alimentación del motor y las velocidades del eje de salida indicadas en la siguiente tabla. Completar las casillas de la tabla con las medidas correspondientes del periodo de las señales de pulsos de los canales A y B, y la velocidad que se indica en el motor:

Velocidad configurada	Tensión del motor = 12V			Tensión del motor = 6V		
	Periodo A	Periodo B	Velocidad**	Periodo A	Periodo B	Velocidad**
16 RPM						
32 RPM						
60 RPM						
118 RPM						
Periodo medio*						

* Tiempo medio calculado para el periodo A o B a partir de los tiempos obtenidos para las velocidades de 16 RPM a 118 RPM. ** Velocidad a la que gira el eje de salida del conjunto motor-reductor-codificador.

c) En base a los datos obtenidos en la tabla anterior, contestar estas preguntas: ¿Las señales de pulsos de los canales A y B tienen el mismo periodo para la misma configuración? ¿Qué diferencia de fase hay entre las dos señales? ¿El codificador está en el eje del motor, o en el eje de salida tras el reductor?

d) A partir de los datos de la tabla anterior, calcular el número de pulsos que corresponden a una vuelta completa del eje de salida del reductor para las velocidades y tensiones dados en la siguiente tabla:

Velocidad configurada	Tensión del motor = 12V	Tensión del motor = 6V
32 RPM		
60 RPM		
118 RPM		

e) Analizar si se puede determinar la relación de reducción N_M/N_S del conjunto motor-codificador-reductor y la velocidad a la que gira el eje del motor, dados unos valores concretos de tensión de alimentación del motor y de configuración de la velocidad de salida para 12V.

2.1.2. Cuenta de pulsos en modo 1x (2 puntos)

Con el codificador E6B2-CWZC6 y un Arduino Uno reales:

f) Crear un circuito en el laboratorio que conecte el codificador E6B2-CWZC6 a un Arduino Uno, usando una placa de prototipos y otros componentes necesarios, teniendo en cuenta que:

- El codificador se debe alimentar con los 5V que proporciona el Arduino.
- Los canales A, B y Z del codificador se deben conectar, respectivamente, a los pines 2, 3 y 4 del Arduino Uno.

Dibujar el esquema eléctrico del circuito (de forma similar a los esquemas eléctricos de ejemplo mostrados en este manual) y capturar una imagen del montaje, para añadir las dos imágenes al informe. Para dibujar el esquema eléctrico se puede usar la aplicación que se desee.

g) Implementar un programa para Arduino Uno que lleve la cuenta de pulsos con estas consideraciones:

- Se debe usar el modo 1x mediante una interrupción para el canal A del codificador.
- Hay que controlar que la cuenta pueda crecer o disminuir según el sentido de giro detectado a partir del codificador.
- Cada vez que se detecte un flanco de bajada en el canal I o Z, la cuenta se debe poner a 0.
- Cada 500ms hay que enviar el valor actual de la cuenta de pulsos por el puerto serie utilizando una cadena de texto con el siguiente formato, de forma que se pueda ver el estado de la cuenta en la ventana del monitor serie del IDE de Arduino:

  tiempo, sentido, cuenta [CRLF]

  Donde “tiempo” es el instante de tiempo respecto al inicio del programa, en milisegundos; “sentido” es un carácter “H” o “A” en función de si el codificador gira en sentido Horario o Antihorario; “cuenta” es un número entero (positivo o negativo) con la cuenta de pulsos, y “CRLF” son los caracteres de retorno de carro y nueva línea, que son enviados automáticamente con la función “Serial.prinln()”.

- No se permite utilizar bibliotecas de otros autores que faciliten la gestión de codificadores.

h) Determinar experimentalmente los siguientes parámetro del codificador E6B2-CWZC6 utilizando el programa anterior:

- El número de pulsos que el programa cuenta para una vuelta del eje (PPR) en modo 1x, moviendo el eje manualmente. Para obtener una medida mejor, se puede girar el eje del codificador a mano varias vueltas completas (por ejemplo 3), y determinar el número de pulsos medio en una vuelta dividiendo la cuenta total de pulsos entre el número de vueltas.
- El número de líneas (NL) que tiene el disco del codificador a partir del valor de PPR.
- La resolución efectiva del codificador (Re) en grados por pulso, a partir del PPR obtenido antes.

Con el motor-codificador-reductor plateado grande y un Arduino Uno simulados en Tinkercad – Circuits:

i) Diseñar e implementar en el simulador Tinkercad – Circuits el circuito necesario para conectar el conjunto motor-codificador-reductor a un Arduino Uno, teniendo en cuenta que:

- Los canales A y B del codificador se deben conectar a los pines 2 y 3 respectivamente.

- Hay que configurar una velocidad de salida para el motor-codificador-reductor de 60rpm, y una tensión de 6V en la fuente de alimentación para el motor.
- El control del motor se puede hacer de una de estas formas, según se desee: manualmente con interruptores o desde el programa de Arduino. Si el control se hace desde el programa, hay que usar el pin 5 para activación del motor, y el 6 para cambiar el sentido de giro.
- Hay que conectar al pin 4 del Arduino Uno un sensor que simule la detección de la posición inicial del eje, aunque haya que activarlo manualmente. Justificar la elección del sensor escogido. Cada vez que se detecte una activación del sensor, la cuenta se debe poner a 0.

Dibujar el esquema eléctrico del circuito (de forma similar a los esquemas eléctricos de ejemplo mostrados en este manual) y capturar una imagen del montaje, para añadir las dos imágenes al informe. Para dibujar el esquema eléctrico se puede usar la aplicación que se desee.

j) Usar el programa desarrollado en la tarea g) para Arduino Uno simulado en Tinkercad – Circuits, con las adaptaciones que se consideren necesarias (si las hay).

k) Determinar los siguientes parámetros del motor-codificador-reductor en Tinkercad – Circuits:

- El número de pulsos en una vuelta del eje de salida (PPR) del conjunto motor-reductor-codificador en modo 1x. Para obtener una medida mejor, hay obtener la cuenta total de pulsos cuando el eje de salida completa 3 vueltas, y dividir esa cuenta entre 3.
- La resolución efectiva en el eje de salida (Re) en grados por pulso, a partir del PPR obtenido antes.

2.1.3. Cuenta de pulsos en modo 4x (2 puntos)

Con el codificador E6B2-CWZC6 y un Arduino Uno reales:

l) Usando el mismo circuito real montado en la tarea f) de la actividad anterior, modificar el programa desarrollado para la tarea g) para que lleve a cabo la cuenta de pulsos en modo 4x.

m) Usando el nuevo programa, determinar el número de pulsos para una vuelta (PPR) del eje del codificador y la resolución efectiva (Re) del eje del codificador en grados por pulso que se tienen en modo 4x, trabajando forma similar a como se plantea en la tarea h).

Con el motor-codificador-reductor plateado grande y un Arduino Uno simulados en Tinkercad – Circuits:

n) Usando el mismo circuito de Tinkercad – Circuits montado en la tarea i) de la actividad anterior, adaptar el programa desarrollado en la tarea l) para que funcione en el Arduino Uno simulado.

o) Usando el nuevo programa, determinar el número de pulsos para una vuelta del eje de salida (PPR) del conjunto motor-codificador-reductor, y la resolución efectiva (Re) del eje de salida en grados por pulso que se tienen en modo 4x, trabajando forma similar a como se plantea en la tarea k).

2.1.4. Cálculo de la posición (1,5 puntos)

Con el motor-codificador-reductor plateado grande y un Arduino Uno simulados en Tinkercad – Circuits:

p) Actualizar el programa de la tarea o) para que calcule la posición angular del eje de salida, teniendo en cuenta que:

- La posición angular debe ser valor entre 0° y 359°, obtenido a partir de la cuenta de pulsos, siendo 0° grados la posición marcada por el sensor de posición inicial. Hay que prestar especial atención al paso entre el ángulo máximo de 359° y 0°, de forma que el valor 360° se muestre como 0°.
- Modificar el programa para que la cadena de texto que se envía cada 500ms incluya también la posición angular, como un valor entero de grados entre 0 y 359, con el siguiente formato:

tiempo, sentido, cuenta, posición-angular [CRLF]

2.2. Actividades complementarias

Se plantean a continuación varias actividades que se pueden desarrollar para completar la nota de la práctica hasta llegar a un máximo de 10. No es necesario desarrollar todas tres actividades, y se puede escoger cuales de ellas se implementan. Cada una de estas actividades se puede realizar para el codificador y Arduino Uno reales, o para el montaje con el conjunto motor-codificador-reductor simulado en Tinkercad – Circuits.

2.2.1. Pantalla LCD (2 puntos)

Usar una pantalla LCD para mostrar el estado de la cuenta de pulsos y la posición angular del eje del codificador E6B2-CWZC6, o del eje de salida del conjunto motor-codificador-reductor de Tinkercad – Circuits.

q) Mostrar la cuenta de pulsos actual y el sentido de giro ("H" para giro horario, "A" para antihorario) en la línea superior de una pantalla LCD, con una cadencia de actualización de 1s. Hay que usar estos pines para la pantalla LCD:

- Pin 7: E (Enable).
- Pines 8 a 11: líneas de datos D4 a D7 (D4→ 8, D5→ 9, D6→ 10, D7→ 11).
- Pin 12: RS (Register Select).

r) En la línea inferior se debe mostrar un indicador estilo "barra de progreso" que marque gráficamente la posición de 0° a 359° grados. Para ello hay que escalar la posición en grados a una barra (pintada con bloques solidos o guiones por ejemplo) en la posición el cursor de 0 a 31 de la segunda fila.

s) Se puede mejorar la representación gráfico de la posición como una barra de progreso de mayor resolución en la pantalla LCD de la siguiente forma: en vez de pintar de 0 a 32 caracteres completos, se puede diseñar 6 caracteres nuevos que muestren una barra de 0 a 5 pixeles horizontales, y usar esos caracteres para pintar tramos intermedios de la barra de progreso. De este modo, la barra tendrá un total de 5x32=160 puntos o pixeles. Las siguientes páginas pueden ser de ayuda para aprovechar bien las características de la pantalla LCD.

- Cómo usar un LCD de 16×2 caracteres con Arduino (Descubriendo Arduino):
  https://descubrearduino.com/como-usar-un-lcd/
- Generador de caracteres especiales para LCD alfanuméricas (Aladuino):
  https://www.aladuino.com.mx/blog/generador-de-caracteres-especiales-para-lcd-alfanumericas/

2.2.2. Cálculo de la velocidad (2 puntos)

Incluir, en el programa que utiliza el modo 4X, el cálculo de la velocidad angular del eje de salida a partir de la medida de pulsos.

t) Seleccionar e implementar uno de estos método para el cálculo de la velocidad:

- Medida de periodo.
- Medida de frecuencia.
- Los dos métodos anteriores, y una selección automática del método usado según la velocidad.

u) Modificar el programa para que la cadena de texto que se envía cada 500ms incluya también la velocidad angular, como un valor entero (positivo o negativo) expresado en RPM (vueltas por minuto), con el siguiente formato:

tiempo, sentido, cuenta, posición-angular, velocidad-angular [CRLF]

2.2.3. Posicionar motor (2 puntos)

Se plantea que el programa de Arduino pueda controlar el movimiento del conjunto motor-codificador-reductor de Tinkercad – Circuits, de forma que el eje de salida se mueva desde la posición actual a una posición deseada.

v) Si no se ha hecho antes en la tarea i), adaptar el circuito del simulador Tinkercad – Circuits para poder controlar la activación del motor y su sentido de giro desde el programa del Arduino Uno.

w) Modificar el programa del Arduino Uno para que pueda recibir un número desde el monitor serie en el ordenador, que indicará una posición deseada para el eje de salida como un valor de 0° a 359° (sin decimales). En función de la posición deseada y la posición actual se puede determinar el error en posición que existe, y el sentido de giro en que hay que hay que mover el motor.

x) Hacer que el programa actúe sobre las salidas que controlan los relés de activación del motor y selección del sentido de giro, para que el motor gire hasta que la posición del eje de salida sea la posición deseada, esto es, hasta que el error de posición sea inferior a 5 grados.

3. Entrega

Para entregar la práctica, hay que incluir en un archivo comprimido tipo ZIP los siguientes archivos:

- Informe en formato PDF donde se incluyan las respuestas para las tareas las distintas actividades. En las tareas en las que se pide completar una tabla, se realizan preguntas, se pide calcular valores, o se piden imágenes de esquemas o montajes, hay que contestar con la respuesta solicitada. Al comienzo del informe se deben indicar claramente los miembros del equipo y el grupo de prácticas al que están asignados. A continuación se resumen las tareas que deben estar incluidas en el informe:

Tarea a)	Circuito de prueba del motor-codificador-reductor de Tinkercad – Circuits.
Tarea b)	Tabla con tiempos y velocidades del motor-codificador-reductor.
Tarea c)	Respuesta a las preguntas sobre del motor-codificador-reductor
Tarea d)	Tabla con el número de pulsos número de pulsos del motor-codificador-reductor.
Tarea e)	Analizar posibilidad de calcular la relación N_M/N_S del motor-codificador-reductor.
Tarea f)	Esquema eléctrico y foto del montaje del circuito real con el codificador E6B2-CWZC6.
Tarea h)	Parámetros del codificador E6B2-CWZC6 en modo 1x: PPR, NL y Re.
Tarea i)	Esquema eléctrico y captura del montaje en Tinkercad – Circuits.
Tarea k)	Parámetros del motor-codificador-reductor en modo 1x: PPR y Re.
Tarea m)	Parámetros del codificador E6B2-CWZC6 en modo 4x: PPR, NL y Re.
Tarea o)	Parámetros del motor-codificador-reductor en modo 4x: PPR y Re.
Tarea v)	Circuito de Tinkercad – Circuits que permite controlar la activación del motor y su sentido de giro desde el programa de Arduino.

- Código fuente de diferentes programas de Arduino propuestos en las actividades, cada uno en una carpeta con el nombre de la tarea. El código debe compilar sin errores en el Arduino Uno real, o en Tinkercad – Circuits, según corresponda, para que sea evaluado. Además, el código debe estar correctamente documentado con comentarios que expliquen brevemente las tareas que lleva a cabo. Es muy importante que el programa use los mismos pines que se indican en el enunciado de las tareas, y este aspecto también tendrá en cuenta en la evaluación. A continuación se resume la lista de programas planteados en las actividades:

Tarea g)	Cuenta de pulsos en modo 1x con el codificador E6B2-CWZC6 y Arduino Uno reales.
Tarea j)	Cuenta de pulsos en modo 1x con el motor-codificador-reductor de Tinkercad.
Tarea m)	Cuenta de pulsos en modo 4x con el codificador E6B2-CWZC6 y Arduino Uno reales.
Tarea o)	Cuenta de pulsos en modo 4x con el motor-codificador-reductor de Tinkercad.
Tarea p)	Cuenta de pulsos en modo 4x y cálculo de posición angular con el motor-codificador-reductor de Tinkercad – Circuits.
Tareas q) r) s)	Cuenta de pulsos en modo 4x, cálculo de posición angular, y visualización de los datos en la pantalla LCD.
Tareas t) u)	Cuenta de pulsos en modo 4x, cálculo de posición angular, y cálculo de la velocidad.
Tarea v)	Control de la posición del motor-codificador-reductor de Tinkercad – Circuits.