Sistemas con Microprocesadores

Práctica 5

Título: Control de las revoluciones de un motor de corriente continua con codificador.

Objetivo:

Utilizar una placa Arduino para mantener constantes las revoluciones de un motor de corriente continua (C.C.), incluso cuando se producen cambios en el voltaje de alimentación. Para establecer la velocidad del motor usamos un potenciómetro.

Entregar:

Hacer público el diseño Tinkercad.
Entregar un PDF de 2 páginas aproximadamente que contenga:

- Nombre y apellidos
- Dirección web del proyecto Tinkercad
- Captura de pantalla de Tinkercad con el motor a 50 R.P.M. aproximadamente y código fuente.
- Captura de una gráfica de las R.P.M. mostrando: la adaptación a un cambio de consigna de 10 a 60 R.P.M. (al girar el potenciómetro del mínimo al máximo), la adaptación a un cambio de tensión de la fuente de 12 V a 24 V, y la adaptación a un cambio de 24 V a 12 V (como se muestra en la gráfica al final de este guion).
  (Antes de pegar el código fuente en el documento PDF lo colorearemos, por ejemplo, utilizando la página: https://highlight.hohli.com/?language=cpp)

Desarrollo:

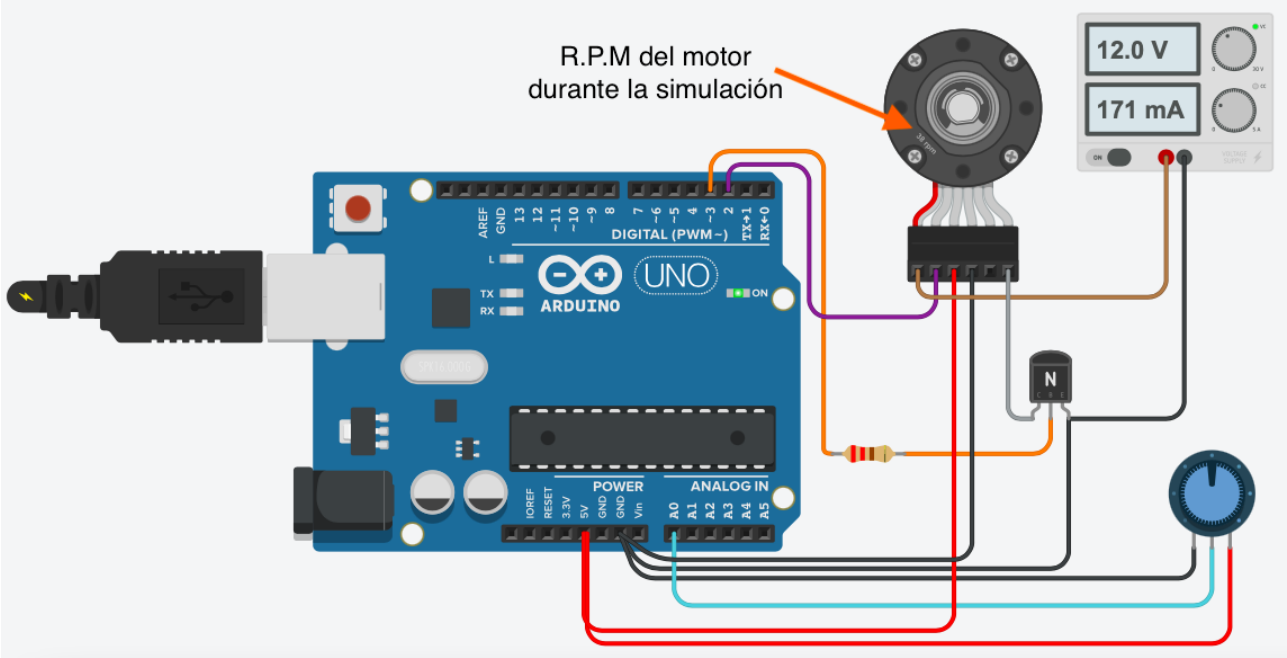
El motor de C.C. gira más rápido cuanta más energía se le suministra, sin embargo, la velocidad exacta a la que girará para una misma energía dependerá de la carga acoplada al eje del motor y otros factores. Para mantener su velocidad constante debemos medir su velocidad continuamente y adaptar la energía que le suministramos para compensar las variaciones en las condiciones de funcionamiento.

Introducimos en Tinkercad el circuito que aparece a continuación, donde una salida P.W.M. (modulación por anchura de pulsos) del microcontrolador controla la energía suministrada a un motor de C.C.

A continuación, en el circuito, pinchando sobre el motor, establecemos el parámetro R.P.M. (revoluciones por minuto) de la configuración del motor a 116, lo que le indica a Tinkercad que el

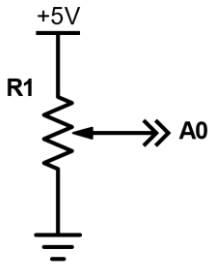
motor debe girar a 116 R.P.M. cuando sea alimentado de forma continua a 12 V. Durante la simulación Tinkercad nos indicará las R.P.M del motor mediante un pequeño letrero cerca del eje.

El motor de C.C. incorpora un codificador (*encoder*) rotatorio que gira solidario al eje del motor. Este codificador genera una serie de pulsos en cada revolución, los cuales utilizaremos para, desde el programa de Arduino, conocer la velocidad del motor y generar la correspondiente señal P.W.M. para controlar dicha velocidad mediante un sencillo lazo de control.
Esta señal P.W.M. activará/desactivará un transistor bipolar NPN, el cual a su vez conectará/desconectará la alimentación del motor de forma intermitente guiado por los pulsos P.W.M. dosificando la energía suministrada al motor. Limitamos la corriente que suministra el microcontrolador a la base del transistor mediante una resistencia en serie de 220 Ω para no dañar el *driver* del microcontrolador correspondiente a esta patilla ni el transistor.



Para medir las revoluciones utilizaremos el canal B del codificador del motor. No necesitamos utilizar los 2 canales del codificador (A y B) porque no necesitamos determinar el sentido de giro del motor (en nuestro caso siempre girará en el mismo sentido).
Conectamos la salida del canal B a una de las patillas del microcontrolador que pueden generar interrupciones, por ejemplo, el pin 2 de la placa. Y configuraremos esta patilla para que genere interrupciones en el flanco positivo de los pulsos (ver código fuente a continuación). La salida del codificador es de colector abierto, de forma que debe tener una resistencia de *pull-up* para que la señal regrese a alta (+5 V) cada vez que el codificador deje de ponerla en baja.

Para indicar la velocidad deseada del motor (consigna) usamos un potenciómetro configurado como divisor de tensión. Es decir, su patilla de la izquierda la conectamos a masa, la de la derecha a alimentación (+5 V), y así la patilla central tendrá una tensión entre 0 y 5 V, dependiendo del ángulo que le demos al eje del potenciómetro. Para medir esta tensión usamos una entrada del microcontrolador con capacidad de hacer lecturas de una señal analógica, por ejemplo, el pin A0.



En la función **setup** configuramos la patilla del microcontrolador conectada al potenciómetro y la patilla de interrupciones como entradas. En la patilla de las interrupciones además tenemos que indicarle al microcontrolador que active la resistencia de *pull-up* interna (en caso de que no le coloquemos una resistencia de *pull-up* externa a esta patilla). También configuramos la patilla de P.W.M. como salida, la cual debemos inicializar a un valor de 0. Aparte, configuramos la velocidad del puerto serie, y, por último, asociamos la función `rutina_int_codif` a la interrupción:

```
attachInterrupt(digitalPinToInterrupt(PIN_INT_CODIF),
rutina_int_codif, RISING);
```

Definimos macros para los pines que utilicemos. Por ejemplo:

```
#define PIN_INT_CODIF 2
#define PIN_POTEN A0
```

Y definimos la función que se encargará de atender a la interrupción y declaramos sus variables globales:

```
#define MOTOR_REVOL_MIN 84
#define CODIF_PULSOS_VUELTA 1656

uint8_t Rpm=0;
unsigned long Tiempo_int_prev;

void rutina_int_codif()
{
  unsigned long tiempo_int_actual = micros();
  unsigned long periodo = tiempo_int_actual - Tiempo_int_prev;

  Rpm = MOTOR_REVOL_MIN * 1000000 / periodo / CODIF_PULSOS_VUELTA;
  Tiempo_int_prev = tiempo_int_actual;
}
```

Esta función medirá el tiempo transcurrido en microsegundos entre dos pulsos consecutivos generados por el codificador del motor, o más concretamente entre los flancos positivos de estos pulsos. Este es, por tanto, el periodo de tiempo con el que se están produciendo pulsos. Si dividimos 1000000 µs/s entre este periodo, obtenemos los pulsos producidos por segundo. El codificador

debería genera 1656 pulsos por cada revolución del motor, pero por particularidades de la implementación de Tinkercad, el número de pulsos generados varía según la velocidad a la que es configurado el motor en la interfaz gráfica. Por tanto, dividimos entre 1656 pulsos/revolución (`CODIF_PULSOS_VUELTA`) y multiplicamos por la constante 84 (`CODIF_REVOL_MIN`) en vez de 60 s/min para compensar la implementación de Tintekcad y obtener las revoluciones por minuto. Finalmente almacenamos este valor en una variable global para poder consultarlo desde fuera de esta función. La variable `Tiempo_int_prev` debe ser inicializada en la función **setup**.

Aunque podríamos incluir un controlador PID completo (controlador Proporcional, Integral y Derivativo) para ajustar con eficacia la velocidad del motor a la consigna, en esta práctica nos limitaremos a implementar un controlador I, ya que con la parte integral es suficiente, aunque la adaptación a los cambios no sea especialmente rápida. Podemos utilizar una constante $K_i = 10 / 1000$ y un lazo de control que es suficiente con actualizarlo cada 0,1 segundos aproximadamente para obtener una respuesta suave del controlador. Es decir, introducimos un retardo tras cada iteración del bucle de control. El cálculo del valor (tiempo en alta) de la señal P.W.M. quedaría así:

```
uint8_t computa_i()
{
  const int ki = 10;

  long pwm_inicial;
  uint8_t pwm_final;

  long err, compon_ki;

  static unsigned long tiempo_i_prev = 0; // Con static se
conserva el valor de la variable de una llamada de la fn a otra
  unsigned long tiempo_i_act = millis();
  unsigned long tiempo_transcurr_i = tiempo_i_act - tiempo_i_prev;

  err = (long)Consigna_rpm - (long)Rpm;
  Err_acum += err * (long)tiempo_transcurr_i;
  compon_ki = ki*Err_acum/1000;
  pwm_inicial = compon_ki;
  if(pwm_inicial > 255)
    pwm_final = 255;
  else if(pwm_inicial < 1)
    pwm_final = 1; // No detenemos completamente el motor nunca
para no dejar de generar interrupciones
  else
    pwm_final = pwm_inicial;

  tiempo_i_prev = tiempo_i_act;

  return pwm_final;
}
```

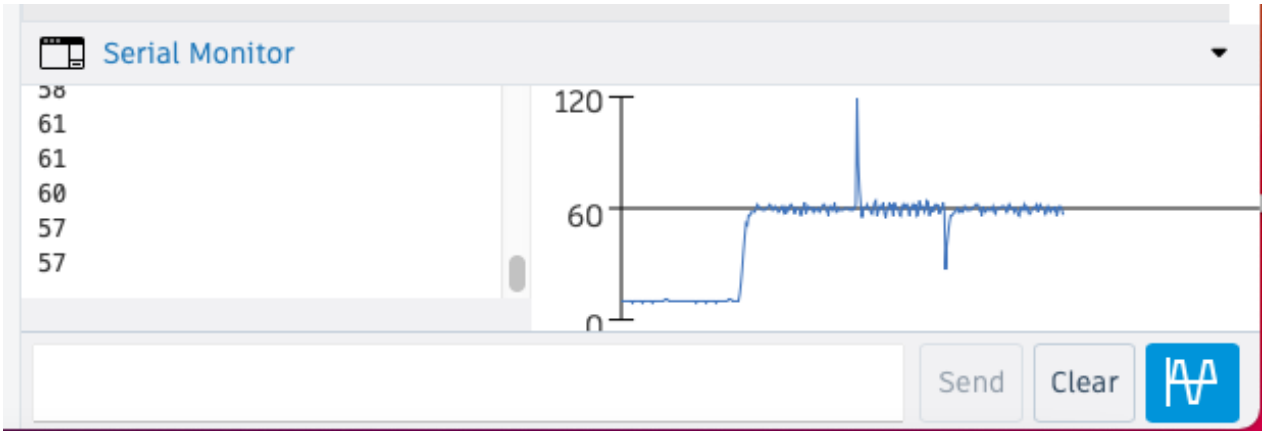
Esta función va acumulando las diferencias entre el valor de revoluciones del motor y la consigna de revoluciones que le hemos dado, multiplicadas por el tiempo transcurrido. Es decir, va integrando el error a lo largo del tiempo. Cuanto mayor es el error integrado mayor será el valor de P.W.M. calculado para contrarrestar el error. Esta función hace uso de la variable global `Err_acum`, la cual debe ser declarada e inicializada. Prestamos atención al tipo que debe tener esta variable a partir de los tipos de las variables que se utilizan para su cómputo (¿valores positivos y negativos? ¿tamaño?). También se consultan las variables globales `Consigna_rpm` y `Rpm`.

Finalmente añadimos código para que se lea continuamente el valor del potenciómetro a través de una entrada analógica:

```
uint16_t volt_poten = analogRead(PIN_POTEN);
```

`analogRead()` devuelve valores entre 0 y 1023 (en nuestro caso será 0 cuando el potenciómetro está al mínimo y 1023 cuando está al máximo). A partir de `volt_poten` debemos calcular `Consigna_rpm`, de forma que cuando `volt_poten` valga 0 `Consigna_rpm` valga 10 R.P.M., y cuando `volt_poten` valga 1023 `Consigna_rpm` valga 60 R.P.M. Para esto podemos usar la función `map()` de Arduino o implementar una expresión matemática que haga el cálculo.

Si enviamos el valor de las R.P.M. por el puerto serie, podemos indicarle al monitor serie de Tinkercad que genere una gráfica en tiempo real de las revoluciones del motor. Para esto pulsamos el botón "Activar/desactiva gráfico" que hay debajo del Monitor serie y la derecha. En la siguiente gráfica podemos observar cómo las revoluciones del motor se ajustan ante un cambio de consigna y también ante 2 cambios bruscos de la tensión de la fuente de alimentación.



Las operaciones matemáticas en el código se han realizado de forma que se evita obtener valores en las variables que sean fracciones de 1, evitando así el uso de variables de coma flotante, las cuales son costosas para el microcontrolador. Los nombres de las variables globales empiezan con mayúscula para distinguirlas claramente de las variables locales (en minúscula) ya que debemos ser especialmente cuidadosos al manipular variables globales para obtener un programa que funcione correctamente.