Robótica con Arduino

Ing. Edison Sásig

Encoder de cuadratura para Robótica

¿Qué es un encoder de cuadratura?

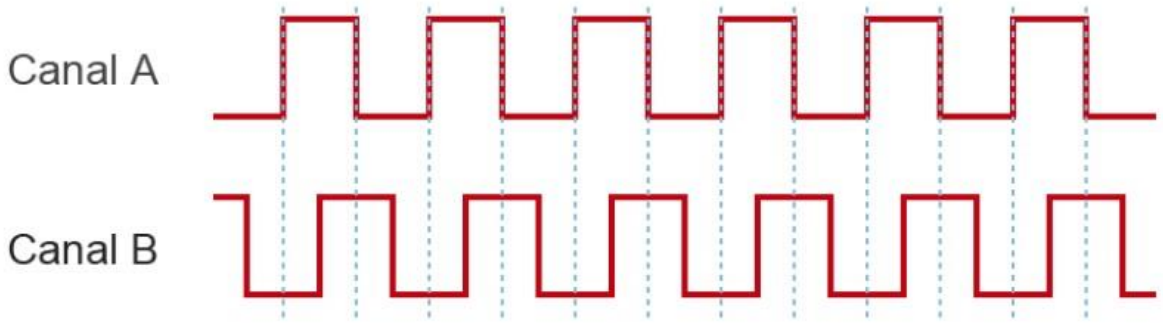
Un encoder de cuadratura es un tipo de encoder rotativo incremental que tiene la capacidad de indicar tanto la posición como la velocidad y la dirección del movimiento del eje del motor.

En este apartado vamos a revisar encoders con sensores magnéticos de efecto Hall. A diferencia de los encoders ópticos, los sensores hall al ser magnéticos pueden trabajar en entornos extremos (temperatura, salpicadura de fluidos, grasas, etc) y polvorientos; no hará falta mantenimiento de limpieza.

Debido a estas características los encoder en cuadratura son muy útiles para las siguientes aplicaciones en robótica:

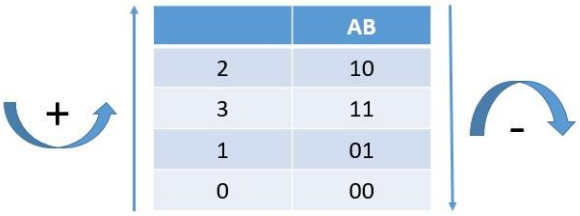
- Robots móviles tipo Uniciclo
- Robots móviles tipo omnidireccional (Mecanum, 3 ruedas)
- Robots móviles tipo Car-Like
- Brazos robóticos
- Robot tipo péndulo
- Robot rueda esférica y mucho más.

Este encoder tiene dos sensores de efecto Hall que generan dos señales de pulsos digitales desfasada en 90º eléctricos o en cuadratura gracias a un disco magnético giratorio, montado en el eje trasero del motor. A estas señales de salida, se les llama comúnmente A y B. Estos dos sensores se van activando y desactivando en una secuencia que nos permite saber la dirección y el número de desplazamientos que han ocurrido en el encoder.



Sentido de giro

Para la determinar el sentido de giro (signo), se toman como referencia el desfasamiento que existe entre las señales AB. El hecho de trabajar con dos canales, nos lleva a tener 4 estados, estos mismos son la combinación de A y B.



	AB
2	10
3	11
1	01
0	00

	Anterior	0	1	2	3
Actual	AB	00	01	10	11
0	00	0	-1	+1	E
1	01	+1	0	E	-1
2	10	-1	E	0	+1
3	11	E	+1	-1	0

Si asumimos la salida A como el bit más significativo y la salida B como el bit menos significativo, en la tabla de verdad podemos saber la dirección de giro, si comparamos el estado actual con el estado anterior.

Además, en la figura también se muestra una matriz de incrementos donde podemos comparar el estado anterior con el actual y saber cómo fue el movimiento. Por ejemplo, si el estado anterior fue 01 y el estado actual del encoder es 11 podemos saber que el motor está girando en sentido anti-horario (+). Esta matriz es muy útil para saber en pocos pasos que ocurre con el encoder. También nos puede servir para detectar errores de nuestra señal o del encoder. Ya que hay 4 combinaciones que no pueden aparecer en nuestra señal, como por ejemplo la combinación (11,00), marcada con una E en la matriz.

Resolución

La resolución de un encoder es una medida en pulsos por revolución **(PPR)**, es decir el número de conteo **(cuentas)** que genera el encoder en cada vuelta. En el caso de los motores Chihai Motor CHR-GM25-370 de 140RPM a 12v, el encoder incremental de cuadratura emite 11 cuentas por revolución del eje del motor, que corresponde a 495 cuentas por revolución del eje de salida de la caja de reductora **(Holzer resolution)**.

Gear Motor parameter list	
Rated voltage	DC12.0V
No-load speed	140RPM 0.10A
Max efficiency	Load 1.7kg.cm/110rpm/2.1W/0.33A
Max power	Load 4.3kg.cm/70rpm/3.2W/0.75A
Stall	STALL TORQUE 8.5kg.cm STALL CURRENT 1.4A
Retarder reduction ratio	1 : 45
Holzer resolution	Motor Holzer 11×ratio 45=495 PPR

Respecto a la precisión, tenemos más de una opción.

- Precisión simple, registrando un único flanco (subida o bajada) en un único canal.
- Precisión doble, registrando ambos flancos en un único canal.
- Precisión cuádruple, registrando ambos flancos en ambos canales.

Resolución para una precisión cuádruple

Para aplicaciones de robótica se utiliza la precisión cuádruple, esta precisión es la que permite detectar el sentido de giro del eje del motor. Por lo tanto, la resolución del encoder para una precisión cuádruple **(R)** se calcula de manera diferente al **Holzer resolution**.

$$R = mH \times s \times r$$

Donde:

R: Es la resolución del encoder para una precisión cuádruple.

mH: Es el número de cuentas por revolución del eje del motor **(Motor Holzer)**. En nuestro caso 11.

s: Es el número de estados generado por los canales **AB**. En nuestro caso 4.

r: Es la relación de reducción de la caja reductora **(ratio)**. En nuestro caso 45 cuentas por revolución.

Por lo tanto,

$$R = 11 \times 4 \times 45 = 1980$$

Sentido de giro en Arduino

En base a la tabla anterior se debe crear un código que permita comparar el estado anterior y el estado actual de los pines C1 (canal A) y C2 (canal B) del encoder.

En primer lugar, se debe detectar los flancos de subida y bajada de ambos canales AB. Para esta tarea lo más recomendable es usar interrupciones. Entonces, definimos los pines y las variables que se van a utilizar, configuramos los pines como entradas y activamos las interrupciones para detectar ambos flancos.

```
const int     C1 = 3;                    // Entrada de la señal A del encoder.
const int     C2 = 2;                    // Entrada de la señal B del encoder.

void setup()
{

  pinMode(C1, INPUT);
  pinMode(C2, INPUT);

  attachInterrupt(digitalPinToInterrupt(C1), encoder, CHANGE);
  attachInterrupt(digitalPinToInterrupt(C2), encoder, CHANGE);

}
```

En la función encoder vamos a comparar el estado anterior y actual e incrementar o decrementar un contador según la tabla. Para esta tarea, vamos a leer un puerto completo que dependiendo en donde se encuentre conectado los pines puede variar.

Los chips utilizados en la placa Arduino (ATmega8 y ATmega168) tienen tres puertos:

- B (pines digitales de 8 a 13) **PINB**
- C (pines de entrada analógica) **PINC**
- D (pines digitales de 0 a 7) **PIND**

En este caso estamos utilizando los pines 2 y 3 es decir el puerto D, por lo tanto, para leer el puerto completo usaremos **PIND**.

Adicionalmente, debemos filtrar solo los pines de interés este caso el 2 y 3. Entonces considerando que los puertos de Arduino (ATmega8 y ATmega168) son de 8 bits se puede aplicar una operación **and** entre el valor de **PIND** y **00001100** (**12** DECIMAL).

0	0	0	0	1	1	0	0
7	6	5	4	3	2	1	0

Entonces en base a tabla debemos hacer la comparación sin olvidar que los valores de la tabla también cambian en base a la conexión de los pines, es decir, si hay un flanco en el canal A (pin 3) y nada en el canal B (pin 2) el valor binario es 10 (2 DECIMAL) se debe cambiar a 00001000 (8 DECIMAL) y así en todas las combinaciones 00, 01 y 11.

0	0	0	0	1	0	0	0
7	6	5	4	3	2	1	0

```
// Encoder precisión cuádruple.
void encoder(void)
{

    ant=act;
    act=PIND & 12;

    if(ant==0  && act== 4)  n++;
    if(ant==4  && act==12)  n++;
    if(ant==8  && act== 0)  n++;
    if(ant==12 && act== 8)  n++;

    if(ant==0 && act==8)  n--;
    if(ant==4 && act==0)  n--;
    if(ant==8 && act==12) n--;
    if(ant==12 && act==4) n--;


}
```

Entonces, si cargamos el siguiente sketch en Arduino al dar una vuelta completa al eje del motor se debe mostrar en el monitor serial el valor de la resolución del encoder para una precisión cuádruple R. Para este caso $R = 1980$.

```
const int    C1 = 3;                  // Entrada de la señal A del encoder.
const int    C2 = 2;                  // Entrada de la señal B del encoder.

volatile int n = 0;
volatile byte ant      = 0;
volatile byte act     = 0;

unsigned long lastTime = 0;  // Tiempo anterior
unsigned long sampleTime = 100;  // Tiempo de muestreo

void setup()
{
  Serial.begin(9600);

  pinMode(C1, INPUT);
  pinMode(C2, INPUT);

  attachInterrupt(digitalPinToInterrupt(C1), encoder, CHANGE);
  attachInterrupt(digitalPinToInterrupt(C2), encoder, CHANGE);

  Serial.println("Numero de conteos");

}

void loop() {
  if (millis() - lastTime >= sampleTime)
  {  // Se actualiza cada sampleTime (milisegundos)
      lastTime = millis();
      Serial.print("Count: ");Serial.println(n);
   }
}
```

```
// Encoder precisión cuádruple.
void encoder(void)
{

    ant=act;
    act=PIND & 12;

    if(ant==0  && act== 4)  n++;
    if(ant==4  && act==12)  n++;
    if(ant==8  && act== 0)  n++;
    if(ant==12 && act== 8)  n++;

    if(ant==0 && act==8)  n--;
    if(ant==4 && act==0)  n--;
    if(ant==8 && act==12) n--;
    if(ant==12 && act==4) n--;


}
```

El código anterior se aplica para un solo motor. Sin embargo, en robótica el número de motores puede ser mucho mayor dependiendo de la aplicación. Entonces vamos a necesitar pines adicionales con interrupciones, y como saben Arduino (ATmega8 y ATmega168) dispone de dos interrupciones externas, por lo cual debemos usar otro Arduino como el MEGA o incluso un ESP32 que tienes muchas características adicionales al tradicional Arduino.

Sin embargo, el Arduino tradicional dispone de interrupciones especiales llamadas PINCHANGE INTERRUPTIONS, las cuales se pueden utilizar en todos los pines incluso en las entradas analógicas, pero son un poco complejas de administrar, sin embargo, gracias la librería PinChangeInterrupt.h podemos utilizar estas interrupciones de la misma manera que una interrupción normal.

En el siguiente ejemplo se ha empleado las interrupciones en el puerto C y en los pines A4 (canal A) y A5 (canal B).

```
#include "PinChangeInterrupt.h"


const int C1 = A5;
const int C2 = A4;
volatile int n = 0;
volatile int ant   = 0;
volatile int act   = 0;

unsigned long lastTime, sampleTime = 100;

void setup() {

  Serial.begin(9600);

  pinMode(C1, INPUT);
  pinMode(C2, INPUT);

  attachPinChangeInterrupt(digitalPinToPinChangeInterrupt(C1), encoder,
CHANGE);
  attachPinChangeInterrupt(digitalPinToPinChangeInterrupt(C2), encoder,
CHANGE);
```

```
  lastTime = millis();
}

void loop() {
  if(millis()-lastTime >= sampleTime)
  {
    lastTime = millis();
    Serial.println(n);
  }
}


void encoder(void)
{

    ant=act;
    act=PINC & 48;


    if(ant==0  && act==16)  n++;
    if(ant==16 && act==48)  n++;
    if(ant==32 && act== 0)  n++;
    if(ant==48 && act==32)  n++;

    if(ant==0  && act==32)  n--;
    if(ant==16 && act== 0)  n--;
    if(ant==32 && act==48)  n--;
    if(ant==48 && act==16)  n--;

}
```

Recuerda que para leer todo el puerto C se debe usar **PINC** y también se debe realizar la conversión en decimal de los estados de los pines A4 y A5.

Posición en grados

Para medir la posición relativa del eje del motor, vamos a utilizar la siguiente fórmula.

$$P = \frac{n \times 360}{R}$$

Donde:

P: Es la posición relativa en grados.

n: Es el número de cuentas generadas.

R: Es la resolución del encoder para una precisión cuádruple. En nuestro caso **1980** cuentas por revolución.

Posición en grados en Arduino

Una vez verificado el código anterior es muy fácil obtener la posición, velocidad y sentido de giro del eje del motor, simplemente hay que aplicar las fórmulas.

```
const int    C1 = 3;                  // Entrada de la señal A del encoder.
const int    C2 = 2;                  // Entrada de la señal B del encoder.


volatile int n = 0;
volatile byte ant      = 0;
volatile byte act     = 0;

double P = 0.0;
unsigned long lastTime = 0;  // Tiempo anterior
unsigned long sampleTime = 100;  // Tiempo de muestreo

unsigned int R = 1980; // Resolucion del encoder R = mH*s*r

void setup()
{
  Serial.begin(9600)

  pinMode(C1, INPUT);
  pinMode(C2, INPUT);

  attachInterrupt(digitalPinToInterrupt(C1), encoder, CHANGE);
  attachInterrupt(digitalPinToInterrupt(C2), encoder, CHANGE);

  Serial.println("Posicion en grados");

}

void loop() {

   if (millis() - lastTime >= sampleTime)
   {  // Se actualiza cada sampleTime (milisegundos)
      lastTime = millis();
      P = (n*360.0)/R;
      Serial.print("Posicion: ");
```

```
      Serial.println(P);
   }
}


// Encoder precisión cuádruple.
void encoder(void)
{

    ant=act;
    act=PIND & 12;

    if(ant==0  && act== 4)  n++;
    if(ant==4  && act==12)  n++;
    if(ant==8  && act== 0)  n++;
    if(ant==12 && act== 8)  n++;

    if(ant==0 && act==8)  n--;
    if(ant==4 && act==0)  n--;
    if(ant==8 && act==12) n--;
    if(ant==12 && act==4) n--;


}
```

Velocidad angular en RPM

Para medir las revoluciones por minuto, vamos a utilizar la siguiente fórmula.

$$N = \frac{n \times 60}{t \times R}$$

Donde:

N : Es la velocidad de rotación en **Revoluciones Por Minuto (RPM).**

n : Es el número de conteo generado en determinado tiempo t

t : Es el tiempo de generación de los pulsos en segundos **(s)**.

R : Es la resolución del encoder para una precisión cuádruple. En nuestro caso **1980** cuentas por revolución.

Velocidad angular en RPM en Arduino

En este caso debemos obtener el número de cuentas en un determinado tiempo de muestreo y aplicar la fórmula. Recuerde que el tiempo t es en segundos, sin embargo, en microcontroladores se trabaja en el orden de los microsegundos o milisegundos.

Para trabajar en milisegundos se debe hacer la siguiente conversión 1s = 1000ms y aplicarlo a la fórmula de la siguiente manera.

$$N = \frac{n \times 60 \times 1000}{t \times R}$$

```
if (millis() - lastTime >= sampleTime)
   {// Se actualiza cada tiempo de muestreo
     N =(n*60.0*1000.0)/((millis()-lastTime)*R);// Velocidad en RPM
     lastTime = millis(); // Almacenamos el tiempo actual.
     n = 0;  // Inicializamos los pulsos.
   }
```

Recuerde que cada tiempo de muestreo debemos resetear el valor del contador para obtener una nueva lectura en el siguiente instante de muestreo.

Velocidad angular en radianes por segundo (rad/s)

Con las revoluciones por minuto **RPM** se puede encontrar la velocidad angular en **rad/s**. Una revolución equivale a 2π radianes y 1 minuto equivale a **60** segundos. Por la tanto, la velocidad angular se puede calcular de la siguiente manera.

$$w = \frac{2\pi \times N}{60}$$

Donde:

w: Es la velocidad de rotación en radianes por segundo **(rad/s)**.

N : Es la velocidad de rotación en Revoluciones Por Minuto **(RPM)**.

Otra forma de calcular la velocidad angular en función del número de cuentas es de la siguiente manera.

$$w = \frac{2\pi \times n}{t \times R}$$

Donde:

w: Es la velocidad de rotación en radianes por segundo **(rad/s)**.

n : Es el número de conteo generado en determinado tiempo t

t : Es el tiempo de generación de los pulsos en segundos **(s)**.

R : Es la resolución del encoder para una precisión cuádruple. En nuestro caso **1980** cuentas por revolución.

Velocidad angular en rad/s en Arduino

Al igual que en el cálculo de la velocidad en RPM, en el cálculo de la velocidad angular en rad/s se debe aplicar la fórmula con el respectivo cambio del tiempo en ms.

$$w = \frac{2\pi \times 1000 \times n}{t \times R}$$

Donde $\pi = 3.1416$

```
if (millis() - lastTime >= sampleTime)
   {// Se actualiza cada tiempo de muestreo
      w =(2*pi*1000.0*n)/((millis()-lastTime)*R);// Velocidad angular
      lastTime = millis(); // Almacenamos el tiempo actual.
      n = 0;  // Inicializamos los pulsos.
   }
```

Sin embargo, para optimizar el código vamos a considerar $\frac{2\pi \times 1000}{R}$ como una constante.

Si $\pi = 3.1416$, $R = 1980$, entonces, la constante es $constValue = 3.1733$

```
if (millis() - lastTime >= sampleTime)
   {// Se actualiza cada tiempo de muestreo
      w =(constValue*n)/(millis()-lastTime); // Velocidad angular rad/s
      lastTime = millis(); // Almacenamos el tiempo actual.
      n = 0;  // Inicializamos los pulsos.
   }
```

Velocidad lineal en metros por segundo (m/s)

Para calcular la velocidad lineal en metros por segundo solamente se tiene que multiplicar la velocidad angular por el radio de la llanta conectada al eje de salida de la caja reductora.

$$u = r_{wheel} \times w$$

Donde:

u : Es la velocidad lineal en metros por segundo **m/s**.

w: Es la velocidad de rotación en radianes por segundo **(rad/s).**

r_{wheel} : Es el radio de la rueda en metros **(m)**.

Otra manera de calcular la velocidad lineal en función del número de cuentas y el diámetro de la llanta es la siguiente.

$$u = \frac{d_{wheel} \times \pi \times n}{t \times R}$$

Donde:

u : Es la velocidad lineal en metros por segundo **m/s**.

d_{wheel} : Es el diámetro de la rueda en metros **(m)**.

n : Es el número de conteo generado en determinado tiempo t

t : Es el tiempo de generación de los pulsos en segundos **(s)**.

R : Es la resolución del encoder para una precisión cuádruple. En nuestro caso **1980** cuentas por revolución.

Velocidad lineal en Arduino

De la misma manera vamos a optimizar el código considerando $\frac{d_{wheel} \times \pi \times 1000}{R}$ como constante.

Si $d_{wheel} = 0.067$, $R = 1980$, entonces, la constante es $constValue = 0.1063$

```
if (millis() - lastTime >= sampleTime)
   {// Se actualiza cada tiempo de muestreo
      u =(constValue*n)/(millis()-lastTime); // Velocidad lineal m/s
      lastTime = millis(); // Almacenamos el tiempo actual.
      n = 0;  // Inicializamos lo``s pulsos.
    }
```