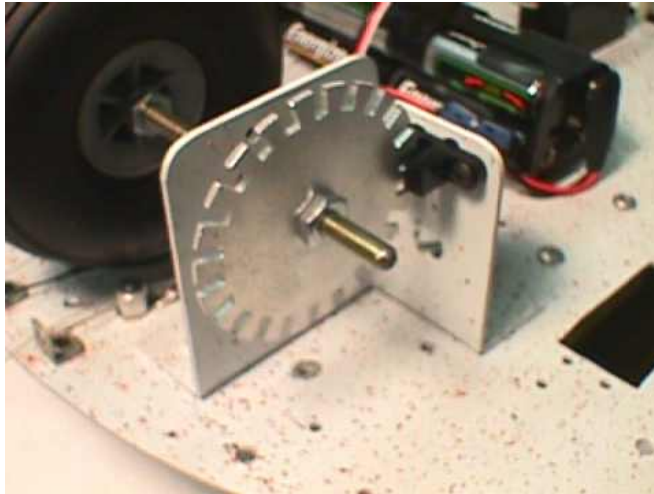


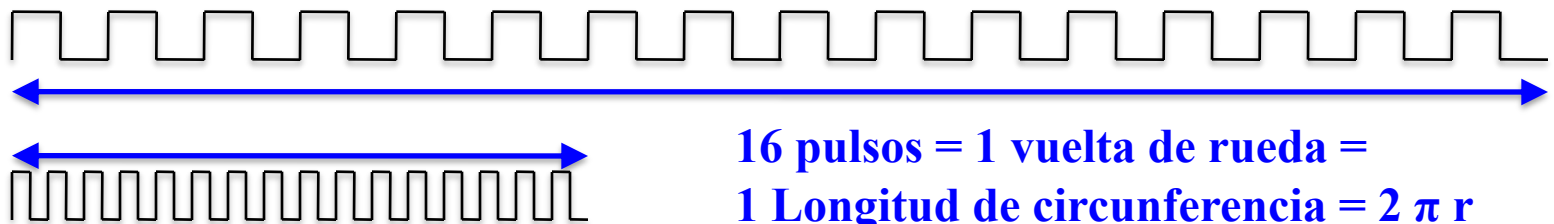
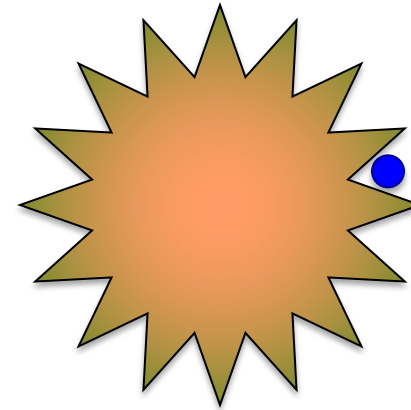


Medida de Giro y Velocidad

Encoder óptico incremental



Engranaje de 16 dientes



16 pulsos = 1 vuelta de rueda =
1 Longitud de circunferencia = $2 \pi r$

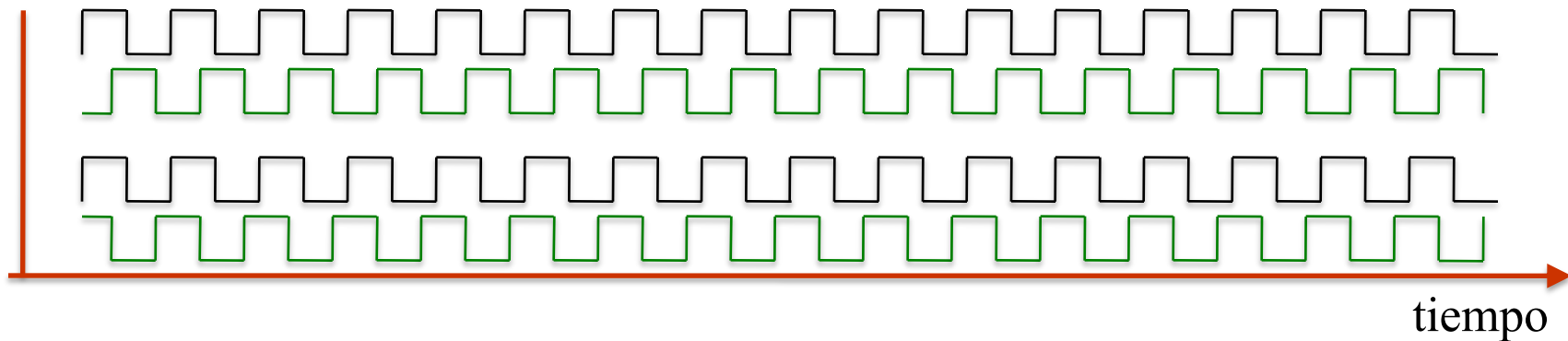
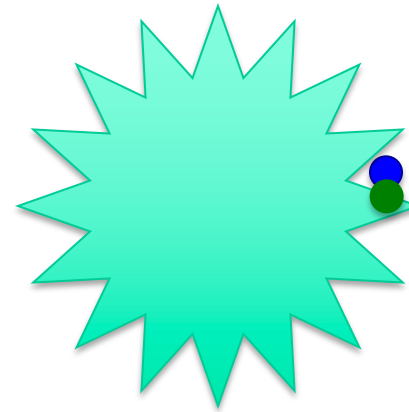
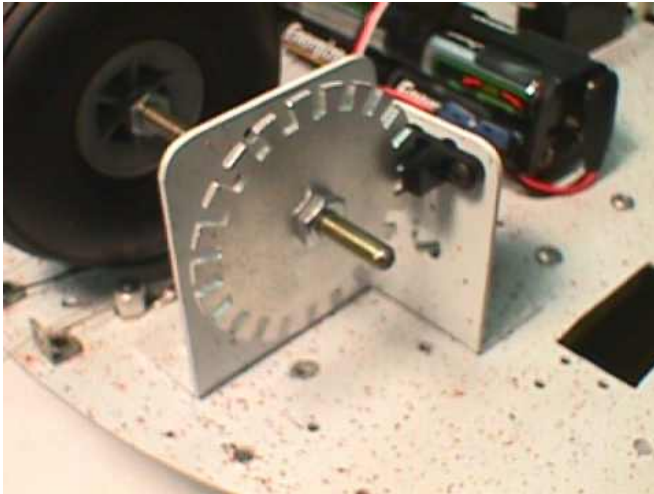
tiempo

$$\text{Velocidad} = \text{pulsos} / \text{tiempo} \left\{ \begin{array}{l} \text{N pulsos} / \text{Tiempo que tardan los N pulsos} \\ \text{N pulsos medidos} / \text{Tiempo fijo} \end{array} \right.$$



Detección de Movimiento

Encoder óptico incremental

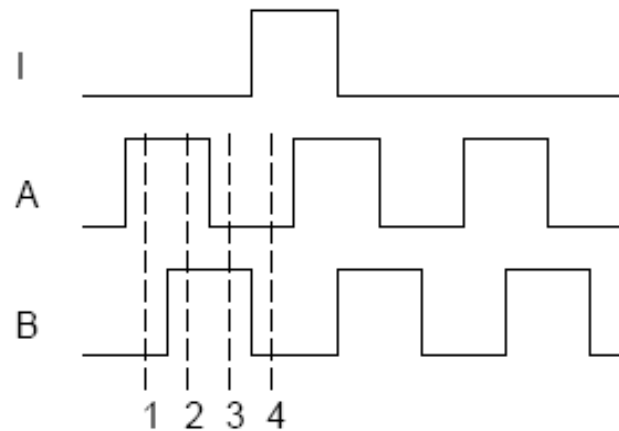
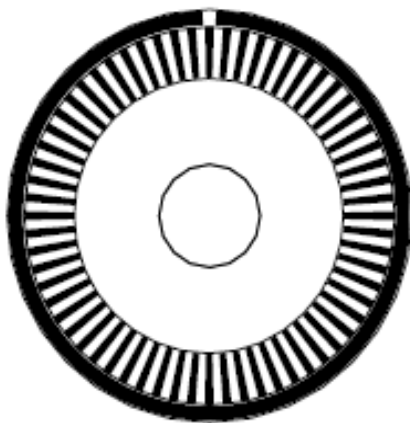




Detección de Movimiento

Encoder óptico incremental con salida en cuadratura

- Tiene dos canales de salida de pulsos.
- Al girar el motor se obtienen pulsos por cada uno de los dos canales
- El desfase entre los canales da idea del sentido de giro
- Se puede aumentar la resolución contando los estados por los que van pasando las dos señales.



State	Ch A	Ch B
S ₁	High	Low
S ₂	High	High
S ₃	Low	High
S ₄	Low	Low

Parámetro: número de pulsos por vuelta

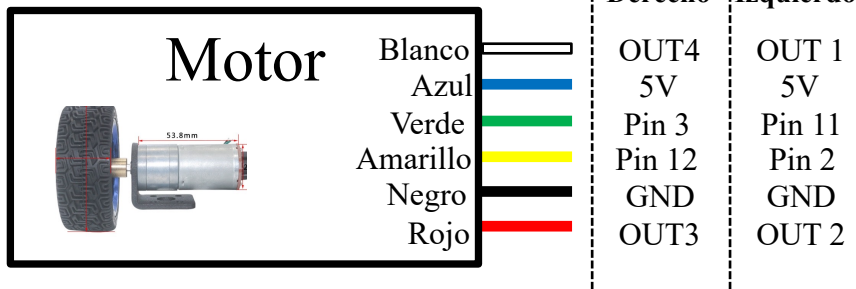


Detección de movimiento

Motor del KIT y conexión del encoder (11 pulsos por vuelta)

- 11 pulsos y reductora de 35:1 $\rightarrow 11 \times 35 = 385$ pulsos/vuelta de rueda
- Rueda de 68mm de diámetro
- Cable rojo: alimentación positiva del motor (+)
- Cable blanco: alimentación negativa del motor (-)
- Cable azul: alimentación positiva del codificador (+) (3,3-5V)
- Cable negro: alimentación negativa del codificador (-) (0V) GND.
- Cable amarillo: canal A del encoder
- Cable verde: canal B del encoder

Conexión a Arduino

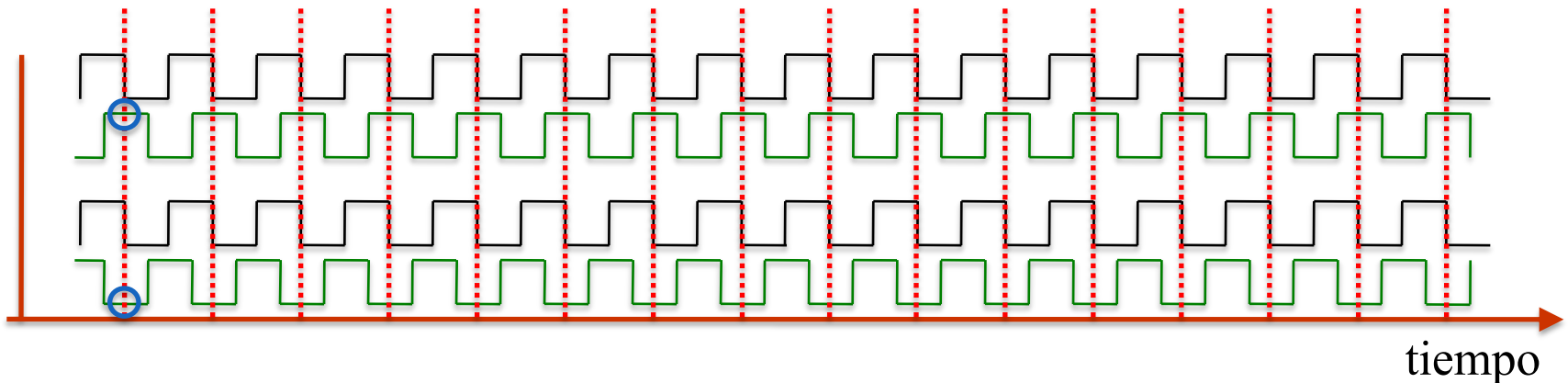




Detección de movimiento

¿Cómo detectamos los pulsos del encoder?

- Necesitamos contar pulsos para medir la distancia recorrida
- Viendo la distancia recorrida cada cierto tiempo calculamos la velocidad
- Podemos saber el sentido de giro si vemos el valor de un canal cuando hay un flanco de bajada (por ejemplo) en otro canal.
- Si fuéramos capaces de enterarnos cuándo hay un flanco de bajada en un canal, el estado del otro canal nos proporciona el sentido de giro.
- **Necesitamos que Arduino se entere cuando llega un flanco de bajada.**





Interrupciones externas en Arduino

Concepto de interrupción

- Una función de interrupción o ISR (Interruption Service Routines) es una **función que se ejecuta cuando se produce un evento hardware** determinado interrumpiendo la ejecución del programa principal y retornando a él al finalizar.
- Un microcontrolador suele tener muchos **eventos hardware** que pueden provocar interrupciones que se pueden **habilitar y deshabilitar de forma independiente**.
 - Flancos en algunos pines, un temporizador llega al final de la cuenta, se recibe un dato nuevo por el puerto serie, ...
- Cuando se produce una interrupción en Arduino **se deshabilitan el resto de interrupciones** (salvo que se fuerce su habilitación). Al finalizar se ejecuta otra interrupción que estuviera en espera (si así fuera).



Interrupciones externas en Arduino

Concepto de interrupción

- La función de interrupción tiene que ser una función que **no recibe nada y no devuelva nada** (ya que nadie la llama). void IRQ (void)
- Las funciones de interrupción **cuanto más cortas mejor** para evitar que se pierdan eventos de otras interrupciones.
- Las variables que se utilicen en una interrupción **deben declararse como volatile** para que el compilador tenga en cuenta que en cualquier momento puede cambiar su valor



Interrupciones externas en Arduino

Interrupciones externas INTx

- Se puede configurar por nivel bajo, flanco de subida, bajada o ambos
- **attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)**
 - El primer parámetro es el **número de interrupción** que se obtiene automáticamente con la función `digitalPinToInterrupt()`
 - El segundo parámetro es el **nombre de la función** a la que se debe llamar cuando se produce la interrupción.
 - El tercer parámetro indica el **evento que genera la interrupción** externa
RISING, ocurre en el flanco de subida de LOW a HIGH.
FALLING, ocurre en el flanco de bajada de HIGH a LOW.
CHANGE, ocurre cuando el pin cambia de estado (rising + falling).
LOW, se ejecuta continuamente mientras está en estado LOW.

Modelo	INT0	INT1	INT2	INT3	INT4	INT5
UNO	2	3				
Mega	2	3	21	20	19	18



Interrupciones externas en Arduino

Interrupciones externas INTx

- Otras funciones relacionadas con las de interrupciones
 - **noInterrupts() o asm("CLI;")** Desactiva la ejecución de interrupciones
 - **interrupts() o asm("SEI;")** Habilita las interrupciones definidas con `attachInterrupt()`.
 - **detachInterrupt(num Interrupt)**, Anula la interrupción indicada.
- Pasos a seguir para configurar las interrupciones externas
 - **Configurar el pin** de entrada de interrupción como entrada con `INPUT_PULLUP` para evitar que se produzcan interrupciones cuando no está conectado.
 - **Definir una función de atención a la interrupción** a la que llamar cuando se produzca el evento asociado.
 - **Configurar la interrupción** indicando el tipo de evento que la provocará, asociándola a un pin y a una función de interrupción



Interrupciones externas en Arduino

Interrupciones externas INT0 e INT1 (Pin 2 y Pin 3)

- Programa que conmuta el LED con cada flanco de bajada recibido por el pin de interrupción 2,

InterrupcionExterna

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(interruptPin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(interruptPin),
                    blink, FALLING);
}

void loop() {
    digitalWrite(ledPin, state);
}

void blink() {
    state = !state;
}
```



Interrupciones externas en Arduino

Interrupción asociada al cambio del nivel en pines PCINTn

- En principio el hardware permite generar una **interrupción con cualquiera de los pines de entrada.**
- **No hay funciones específicas de Arduino para configurarlo.**
- Es necesario habilitar las interrupciones de cambio del puerto 0, 1 o 2 en el registro PCICR
- Los flags de interrupción están en PCIFR
- Las máscaras locales de los pines de cada puerto en PCMSK0 (PCINT0 a PCIN7), PCMSK1 (PCINT8 a PCIN14) y PCMSK2 (PCINT16 a PCINT23)
- Hay que asociar una función de interrupción a la interrupción

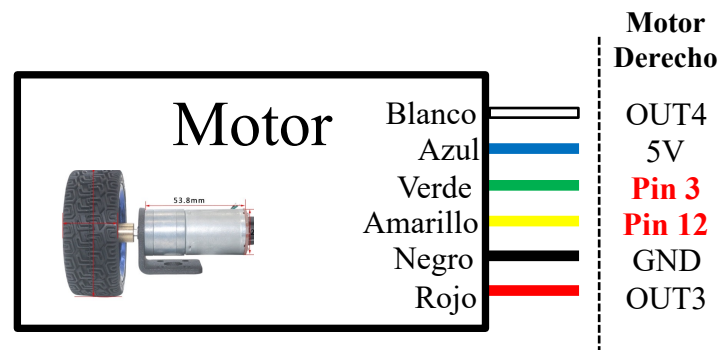
Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x0002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2



Detección de movimiento

Lectura del número de pasos de avance de un encoder

- Se conecta uno de los dos canales del encoder a una de las entradas de interrupción. El otro a un pin digital normal.
- Se configura la interrupción externa para que, con un flanco de bajada del canal conectado al pin de interrupción, incremente o decemente una variable que cuenta los pulsos dependiendo del valor del otro canal.
- La variable que cuenta los pulsos puede llamarse “_pulsosD” y debe ser de un tipo de datos que no se desborde fácilmente, por ejemplo “long”
- La función a la que llama la interrupción puede ser “encoderD()”





Detección de movimiento

Lectura del número de pasos de avance de encoder

- Este programa envía los pulsos medidos por el puerto serie
- Pruébalo en el simulador y en el robot

Encoder_1

```
const byte CanalA_D_Pin = 3;
const byte CanalB_D_Pin = 12;
volatile long _pulsosD = 0;
long pulsosD = 0;
void setup() {
    pinMode(CanalA_D_Pin, INPUT_PULLUP);
    pinMode(CanalB_D_Pin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(CanalA_D_Pin),
                    encoderD, FALLING);

    Serial.begin(9600);
}
void loop() {
    noInterrupts();
    pulsosD = _pulsosD;
    interrupts();

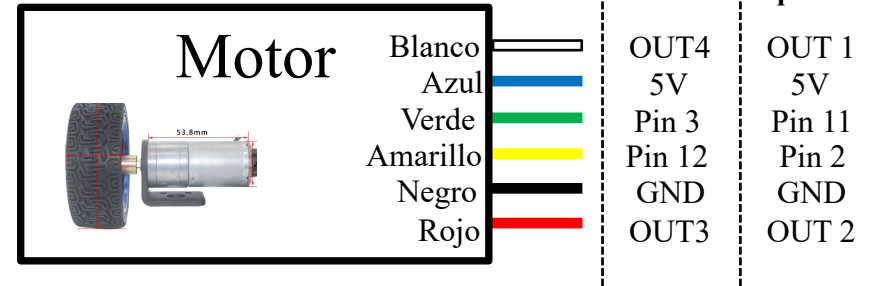
    Serial.println(pulsosD);
}

void encoderD() {
    if (digitalRead(CanalB_D_Pin)==1)
        _pulsosD++;
    else
        _pulsosD--;
}
```



Detección de movimiento

Propuesta de actividades



– **Actividad 1** *Encoder_2*

- Conecta los dos encoders a Arduino, modifica el ejemplo para que lea pulsos de los dos encoder y los envíe por el puerto serie. No es necesario mover los motores, se puede mover el robot manualmente.

– **Actividad 2** *Encoder_3*

- Modifica el programa anterior para que envíe por el puerto serie los pulsos de los encoder una vez por segundo. No utilices la función delay() sino millis().

– **Actividad 3** *Encoder_4*

- Modifica el programa anterior para que envíe por el puerto serie los pulsos de los encoder y los pulsos que van avanzado cada segundo.



Detección de movimiento

Solución de la actividad 1

Encoder_2

– Configuración de los pines y variables

```
const byte CanaA_D_Pin = 3;
const byte CanaB_D_Pin = 12;
const byte CanaA_I_Pin = 11;
const byte CanaB_I_Pin = 2;

volatile long _pulsosD = 0;
volatile long _pulsosI = 0;
long pulsosD = 0;
long pulsosI = 0;

void setup() {
  pinMode(CanaA_D_Pin, INPUT_PULLUP);
  pinMode(CanaB_D_Pin, INPUT_PULLUP);
  pinMode(CanaA_I_Pin, INPUT_PULLUP);
  pinMode(CanaB_I_Pin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(CanaA_D_Pin),
                  encoderD, FALLING);
  attachInterrupt(digitalPinToInterrupt(CanaB_I_Pin),
                  encoderI, FALLING);
  Serial.begin(9600);
}
```



Detección de movimiento

Solución de la actividad 1

Encoder_2

- Funciones de interrupción y programa principal

```
void encoderD() {  
    if (digitalRead(CanalB_D_Pin)==1)  
        _pulsosD++;  
    else  
        _pulsosD--;  
}
```

```
void encoderI() {  
    if (digitalRead(CanalA_I_Pin)==1)  
        _pulsosI++;  
    else  
        _pulsosI--;  
}
```

```
void loop() {  
    noInterrupts();  
    pulsosD = _pulsosD;  
    pulsosI = _pulsosI;  
    interrupts();  
  
    Serial.print(pulsosD);  
    Serial.print(" ");  
    Serial.println(pulsosD);  
}
```



Detección de movimiento

Solución de la actividad 2

- Para medir el tiempo no utilizamos delay() sino la función millis(). De esta manera el programa no está bloqueado en la función delay

Encoder_3

```
unsigned long millisActual = 0;
unsigned long millisAnterior = 0;

void loop() {
    noInterrupts();
    pulsosD = _pulsosD;
    pulsosI = _pulsosI;
    interrupts();

    millisActual = millis();
    if ((millisActual - millisAnterior) >= 1000) {
        millisAnterior = millisActual;
        Serial.print(pulsosD);
        Serial.print(" ");
        Serial.println(pulsosI);
    }
}
```



Detección de movimiento

Solución de la actividad 3

- Se almacena el valor medido para utilizarlo la siguiente vez

```
long pulsosAnteriorD;  
long pulsosAnteriorI;  
void loop() {  
    noInterrupts();  
    pulsosD = _pulsosD;  
    pulsosI = _pulsosI;  
    interrupts();  
  
    millisActual = millis();  
    if ((millisActual - millisAnterior) >= 1000) {  
        millisAnterior = millisActual;  
        Serial.print(pulsosD);  
        Serial.print(" ");  
        Serial.print(pulsosI);  
        Serial.print(" ");  
        Serial.print(pulsosD - pulsosAnteriorD);  
        Serial.print(" ");  
        Serial.println(pulsosI - pulsosAnteriorI);  
        pulsosAnteriorD = pulsosD;  
        pulsosAnteriorI = pulsosI;  
    }  
}
```

Encoder_4