LAB 5 - MOTOR & ENCODERS

INTRO / EXERCISE

Some motors come with **encoders** to track their motor shaft angle. Encoders may also be purchased separately. In order to control robots or other machinery, it is necessary to be able to read these encoders and translate their data into angular units like degrees. Here we will learn to read encoders from a motor to determine its shaft angle.

Note in this lab we MAY also have to drive the motor. Motors with high gear ratio ("torquey" motors turn slow with high torque) cannot be turned by hand (called backdriveability). If so, we must command Arduino to turn the motor while simultaneously reading the encoders. Low gear ratio motors or motors with no gear head may possibly be turned by hand. In this case you don't need to command Arduino to move the motor.

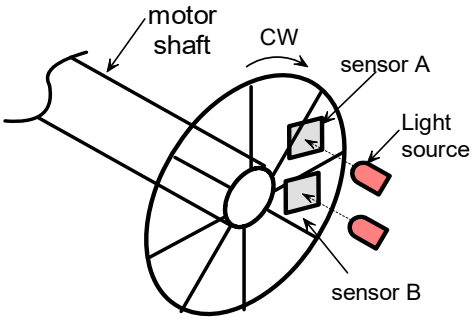
Combining motor control and sensor reading is the essence of robotics!

ITEMS NEEDED

1. Computer with Arduino IDE, Arduino, USB cable
2. DC motor - E-S motor (Mfr #: 22PG 2230 4.75 EN 12V)
   12V, 22 mm planetary gear box, 1900 rpm no load speed; 4.75:1 gear ratio; 4 mm shaft
   16 pulse/revolution encoder, 0.2 kg-cm torque, 1.3 kg-cm stall torque
   2.5 A stall current, 90 mA no load current, 330 mA "rated" current
   Note - this motor is geared for more speed, less torque so it gives use more encoder resolution
3. Motor driver/controller – Cytron MDD10, dual channel (2 motors), 10A, 7-30V ($21)
4. Power supply (12V, bench, power brick, or battery)

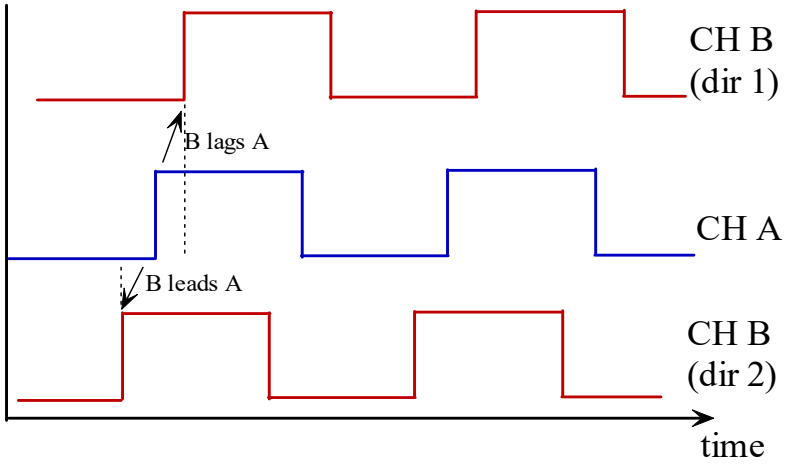
BACKGROUND

How do encoders work? A **code wheel** is mounted to the motor shaft and turns with it. The code wheel has a number of spokes on it (e.g., 64 spokes per revolution). A light or LED is mounted on one side of the code wheel, and a sensor is mounted on the other side. As the motor shaft and code wheel turn, light either passes between the spokes (striking the sensor) or it is blocked by a spoke. Each time a spoke passes, a pulse is produced by the encoder's electronics to produce a square wave. By **counting pulses** (you must program this!), you can infer how much the motor has turned.



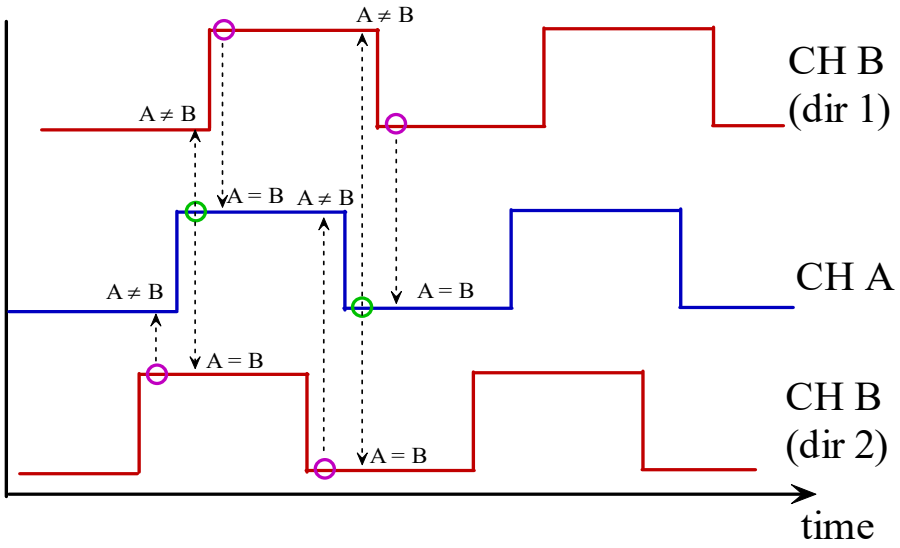
The encoders described above are optical (using light), but they encoders can also use magnets (called "Hall" encoders).

A single channel, however, cannot sense the direction of the motor. One solution is to add a second sensor. Each produces its own signal (channel A and B). If B "leads" A (B's pulse comes a bit later than A's), then the motor is turning one way. If B "leads" A (its pulse comes earlier), then the motor is turning the other way. The code should add pulses in one case, and subtract pulses in the other case. This system is called a **quadrature** encoder because it multiplies the total number of pulses per rotation by four (creating more resolution... better!).



The logic of the programming code for counting pulses can be tricky. First the code must check for changes in A (rising or falling, green circles in figure). Then test if A = = B. Notice that, for direction 1, A ≠ B just after A changes. This is regardless of whether A has risen or fallen. For direction 2, A = B just after A changes (rise or fall).

You must also check for B changing (purple circles in figure). For direction 1, just after B changes, A = B. For direction 2, just after B changes, A ≠ B. You have now established direction 1 and 2. Your code would simply add to the count for one direction, and subtract from the count for the other direction.



DC gearhead motors have a gearhead that usually "gears down" the motor. This means that the OUTPUT SHAFT is turning slower (and with greater torque) than the internal motor shaft (which you may not be able to see). If the DC motor is equipped with an encoder, it is usually connected to the internal motor shaft (the faster turning component). And we usually want to know the angular position of the OUTPUT SHAFT. This configuration gives you more encoder counts (pulses) per output shaft revolution (i.e., greater resolution). This is good. Here is an example. Let's say you have a DC motor with a gear ratio of 10:1. And it also has an encoder with 32 CPR (counts or pulses per revolution). The encoder 32 CPR means 32 pulses per turn of ROTOR to which it is connected. And the rotor turns 10 times for each turn of output shaft. This means you are getting 320 counts (pulses) for every 1 turn of the output shaft (32 CPR times 10 of gear ratio).

CONNECTIONS

Encoders will have 2 inputs: power (usually 5VDC) and ground, which can be drawn from the Arduino board. They will have 2 outputs (channel A and channel B) which are passed to digital input pins on Arduino (make sure these go to pins 2 and 3!). Verify Cytron motor controller jumpers are NOT set on D2 and D3 as we need those for the encoder signals.

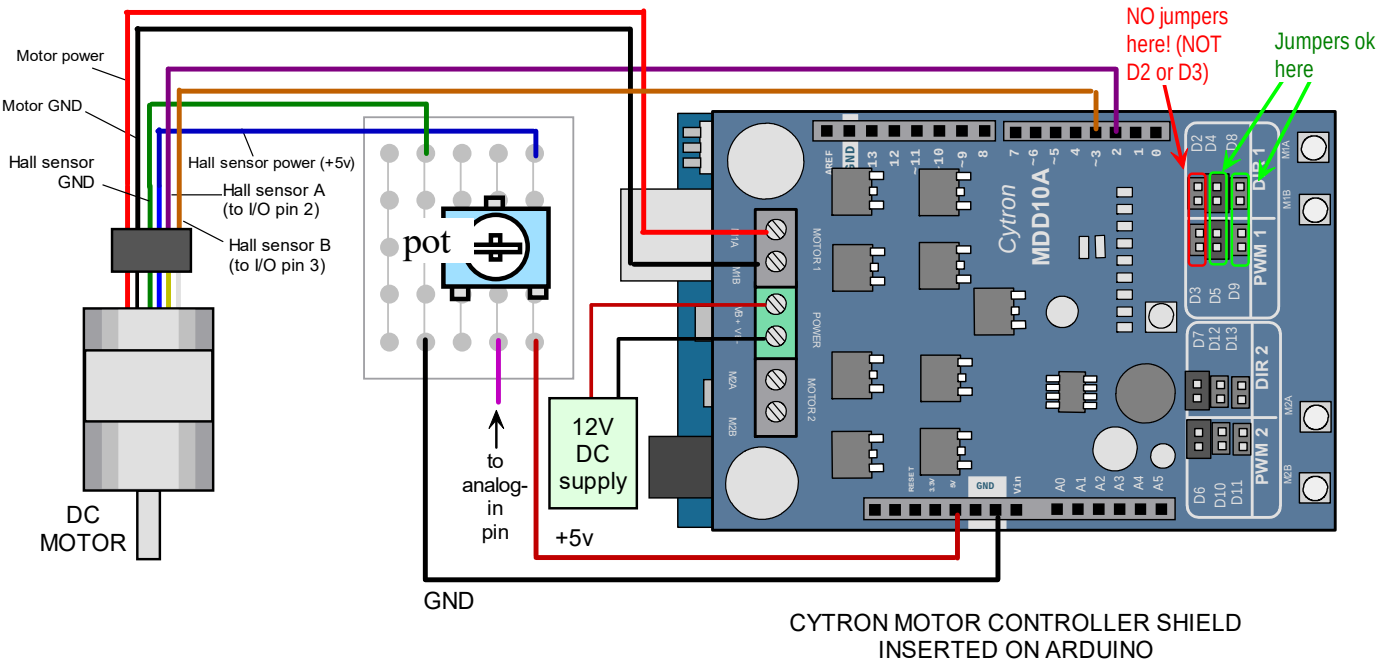
To test the encoder code, you'll need to turn the motor – either by hand or by programming Arduino to command it to turn (then you need the motor driver/controller). Hand-turning the motor may be possible with lower gear ratios. High-gear ratio motors are often not "back-driveable" (ie - can't be turned by hand). And forcing them could damage the motor shaft or gears.

Note the CYTRON MOTOR CONTROLLER SHIELD works using jumpers to select the I/O pins for direction (DIR) and motor signal (PWM). The advantage is that you have fewer wires and a neater circuit. The drawback is that you cannot select any I/O pin you want. You are stuck with the options provided by the board. Example - for motor 1, the PWM pin choices are 3, 5, or 9.

Note the connections for the DC wire harness

RED - (+) voltage to power the motor ((+) V output of motor controller)
BLACK - GND for motor power (GND output of motor controller)
BLUE - sensor (+) power (5V from Arduino)
GREEN - sensor ground (connect to Arduino GND)

YELLOW - encoder output A
WHITE - encoder output B



5.1 ENCODER WITH POLLING

The code below should count the pulses from a quadrature encoder mounted to a DC motor. This program uses **polling**, where the processor is directed to look for an event each time through the loop ( ) function. When an event occurs (a change in an input pin) we assess the direction of movement and update the encoder count. This is then translated into motor angle knowing encoder CPR (counts per revolution) and motor gear ratio.

```
// encoder1_polling.ino -------------------------------------
// read quadrature encoder count using polling
// (no code for commanding motor)
// es-motor, 1900 rpm, 4.75:1 gear ratio, 16 cpr encoder (or 64 cpr with quadrature)
// loop speed = 7500 <-- measured this before

const int pinA = 2;
const int pinB = 3;

boolean encA = 0;
boolean encB = 0;
boolean encAprev = 0;
boolean encBprev = 0;

long count = 0;
unsigned long tlast = 0;
long loops = 0;

float motorAngle = 0.0;
int encoderCPR = 16*4;        // cpr with quadrature
float gearRatio = 4.75;

void setup()
```

```
{
  pinMode(pinA, INPUT);
  pinMode(pinB, INPUT);
  Serial.begin(9600);
}

void loop()
{
  unsigned long tnow = millis();
  loops++;
  encoderCount();
  //motorAngle = 360 * count/(encoderCPR * gearRatio); //convert to degrees

  if (tnow - tlast >= 1000)
  {
    Serial.println(count);
    //Serial.println(loops);
    tlast = tnow;
    loops = 0;
  }
  encAprev = encA;        //update
  encBprev = encB;
}

void encoderCount() // ------------------------------
{
  encA = digitalRead(pinA);
  encB = digitalRead(pinB);

  if (encA != encAprev)         // A changes (up or down)
  {
    if (encA != encB)           //dir 1
    { count++; }
    else                        //dir 2
    { count--; }
  }

  if (encB != encBprev)         // B changes (up or down)
  {
    if (encA == encB)           //dir 1
    { count++; }
    else                        //dir 2
    { count--; }
  }
}

// end ----------------------------------------------------
```

BEHAVIOR

Polling may not work well especially if it is not fast enough. The loop function must run fast enough to capture the A and B pulsing events. If loop() does not run fast enough, or if A and B pulse too quickly (e.g., motor turning fast), then the pulse event will be missed and the encoder count will be wrong. You can test the loop speed by commenting out to serial print "count", and uncomment to serial print "loops". The rate of serial print matters. Printing to the serial monitor really slows down the loop function and can cause your encoder program to work poorly. When serial printing at 1 Hz, the loop function was shown to run at about 7500 Hz.

5.2 ENCODER WITH INTERRUPTS

Polling is easier to program, but it wastes CPU computations checking on an event that may or may not happen at that moment. Polling could work okay if the loop function is cycling fast enough though. But if it runs too slow (e.g., too much code to process), then it may not function properly. A more efficient approach is to use **interrupts**. With interrupts a pin can "interrupt" the CPU when an event occurs; allowing the CPU to stop what it is doing and immediately do something about the event.

Let's say you are waiting for a letter from the postman. With polling, the postman doesn't ring your doorbell. Instead you get up every few minutes to go to the door to see if your letter arrived. It wastes your time and energy. There's a good chance you'll miss him. In contrast, interrupts alert you to the event. In the interrupt case, the mailman rings your doorbell, you stop what you are doing, and go to the door to get your letter. You don't waste your time and you will not miss the event. Interrupts are a bit trickier to program though.

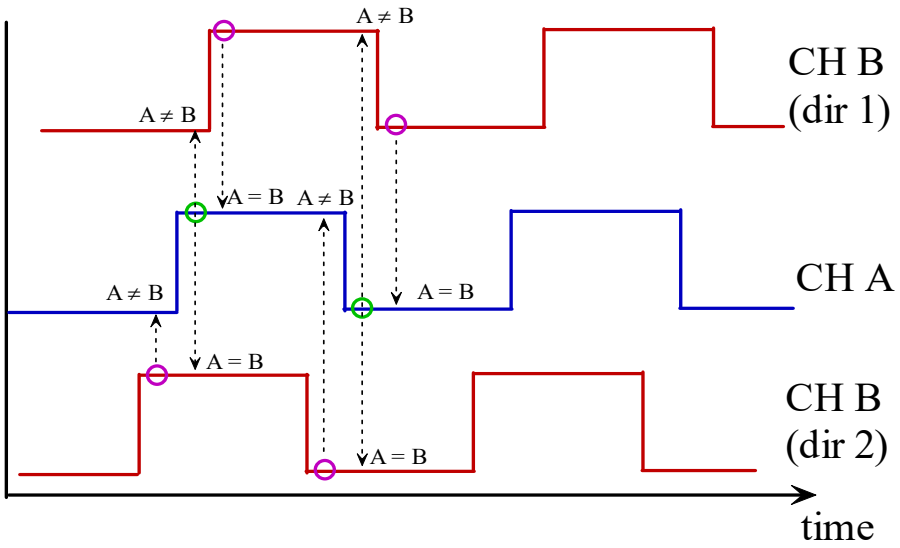
Interrupts with Arduino work when an event occurs at one of the digital pins. The hardware interrupts the CPU and has it immediately execute a special function called the interrupt service routine (ISR). The ISR should be a short and fast bit of code so the processor does not miss another important computation.

There are 3 types of interrupts: change, rising, and falling. ISR's cannot return values or take parameters. The delay () and millis() don't work inside ISR's. The delayMicroseconds ( ) WILL work in ISR's. Any variables changed in the ISR must be declared with VOLATILE modifier. On UNO **only pins 2 and 3** have this so-called "external interrupt" capability. With external interrupts the ISR knows which pin generated the interrupt, and therefore does not need to read the pin. There are also "pin change interrupts" (PCI), which allows almost any pin on Arduino to interrupt the CPU. But with PCI, the ISR must perform more software logic steps to determine which explicit pin in the port caused the interrupt. This is harder to program.

Study the "encoderChanged" functions. See if you can figure out the logic.

```
// encoder2_interrupts.ino ------------------------------------
// Obtain quadrature encoder count using interrupts
// (no code for commanding motor)
// es-motor, 1900 rpm, 4.75:1 gear ratio, 16 cpr encoder (64 cpr with quadrature)

int pinA = 2;       // for uno, only pins 2 & 3 can do interrupts
int pinB = 3;

int encoderCPR = 16*4;  // with quadrature
float gearRatio = 4.75;
volatile long count = 0;
unsigned long tlast;
float outputShaftAngle = 0.0;

void setup()
{
  pinMode(pinA, INPUT_PULLUP);
  pinMode(pinB, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(pinA), encoderAchanged, CHANGE);
  attachInterrupt(digitalPinToInterrupt(pinB), encoderBchanged, CHANGE);
  Serial.begin(9600);
}

void loop()
{
```

```
  outputShaftAngle = 360*count/(encoderCPR*gearRatio); //convert to degrees
  if (millis() - tlast >= 1000)       // update SM periodically
  {
    Serial.println(outputShaftAngle);
    tlast = millis();
  }
}

void encoderAchanged() // -----------------------------------------------
{
  if (digitalRead(pinA) != digitalRead(pinB)) //dir 1
  {
    count++;
  }
  else
  {
    count--;
  }
}

void encoderBchanged() // -------------
{
  if (digitalRead(pinA) == digitalRead(pinB))
  {
    count++;              // dir 1
  }
  else
  {
    count--;
  }
}

//end ---------------------------------
```



SUBMISSION

Submissions are in general a demonstration to instructor.
Instructor will let you know if a video upload is an option.
For video uploads, the file names should be same as indicated on provided code
A merged video is acceptable, but video must be annotated with the exercises being done.

NOTE

Prior motors used are shown below. The Pololu motors used traditional gearheads, but the back-lash was significant. Thus we stopped using these and went with E-S motors which use a planetary gear head (and hopefully less backlash)

1. Pololu 4752: 12V, 30:1 gear, 37Dx68L, 64 CPR encoder; no load: 330 rpm, 150 mA;
   stall: 190 oz in/ 14 kg-cm; 5.5A; 6 mm D shaft, $40
2. Pololu 4846: 12V?, 75:1 gear, 48 CPR encoder...