An encoder attached to a rotating member of a robot drive system can be used to measure the distance traveled by the robot.

GEAR has in stock several motors equipped with AndyMark 2 channel Hall Effect encoders (http://www.andymark.com/product-p/am-2816a.htm). The motor speed is listed at 5700 RPM at 12 volts DC (http://www.andymark.com/product-p/am-3109.htm). The encoder delivers seven pulses of +5 volts DC to each channel with one rotation of the motor. Therefore, each channel delivers 39,900 pulses per minute or 665 pulses per second when motor is rotating at 5700 RPM.

The Hall sensors for the channels are positioned such that the signals are 90 degrees out of phase, the so-called quadrature design. This allows for the determination of rotation direction as well as rotation speed. However, the code provided in this document is simple code, which only determines the number of motor rotations and does not distinguish between directions.

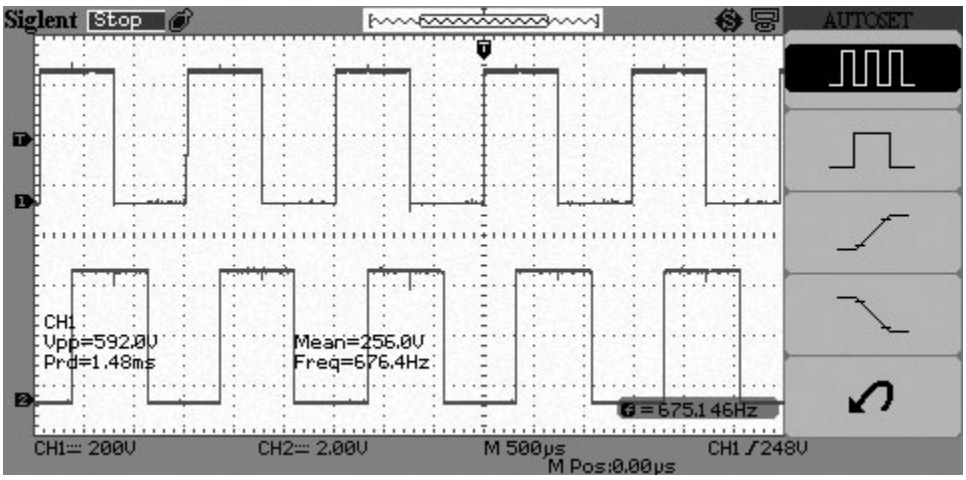

Figure 1 Oscilloscope traces of both channels of encoder, motor running at 5798 RPM

When a two-channel oscilloscope is connected to the output of the AndyMark encoder, the wave forms are graphed (Figure 1). We can see that the signals are typical digital signals (alternating low and high, 0 volts and 5 volts). The time the signal is low is equal to the time the signal is high. Furthermore, the two signals are separated by approximately 90 degrees of phase. That is, the same point in the cycle of one channel lags behind the other channel by 1/4th of a cycle. The frequency of the pulses is 676.4 Hz, which means there are 676.4 pulses per second. Since there are seven pulses per channel for each revolution of the motor, the motor is turning at 676.4/7 = 96.63 revolutions per second or 5798 revolutions per minute (RPM).

For our simple code we will be reading the signal from only one channel (it does not matter which channel). It is necessary to appreciate that at full motor speed, the encoder delivers close to 700 pulses per second. That is many pulses to count and depending on the complexity of your complete robot code, it may be difficult to prevent missing the counting of some of the pulses. By keeping the code simple, hopefully nearly all of the pulses will be counted.

The choice of functions to use in the code may affect the accuracy of counting the encoder pulses. If the functions you are using take too long to complete, then some pulses of the encoder will not be counted.

You may wish to take a look at these two references:
https://jeelabs.org/2010/01/06/pin-io-performance/

http://www.billporter.info/2010/08/18/ready-set-oscillate-the-fastest-way-to-change-arduino-pins/

Here we learn from the tests done that using the function **digitalRead()** may take 50 to 100 times longer to complete than using a direct port addressing method. Thus, if we use a function like **digitalRead()** multiple times in a code block to process the encoder signal, and when the number of pulses per second result in a time span between pulses that is shorter than the time needed to run the code block, there will be missed pulses in the counting.

For this code the only function used to read the signal input is a function named **attachInterrupt()**. The advantage of this function is that it can be configured to run when the Arduino detects a change in the signal (either a rising or falling voltage). You should read the Arduino code reference for this function, which is available here:
https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

The function takes three arguments. The first argument specifies the interrupt number. Following the recommendation of the reference, we use **digitalPinToInterrupt()** function for the first argument like this **digitalPinToInterrupt(2)**, which specifies we are using digital pin 2 to read the encoder channel. The second argument of the **attachInterrupt()** function specifies the function that will run when the interrupt happens. In this code that function is named **readEncoder()**. The third argument of **attachInterrupt()** is the mode. Here we have five options: LOW, CHANGE, RISING, FALLING, HIGH. Our code uses RISING. Therefore, when the voltage starts to rise on the input pin, the interrupt is activated. We could have used FALLING instead and the code would still work properly. If we used CHANGING, then the code would result in twice the counts since it would fire the interrupt when there was a rise and when there was a fall in voltage. For our needs we want to keep the count number relatively low, so only need to make one count per cycle. Using LOW or HIGH would not work for our code because we would get multiple counts for each cycle of the signal.

We have a simple code here. When the signal starts to rise, the interrupt is activated, which results in the running of the function named **readEncoder().** This function adds a value of one to the variable named **encoderCount**. This is exactly what we want. For each pulse on the encoder channel, we add a value of one to the count.

For testing purposes this code uses **Serial.println()** in the **void loop()** to print to the serial monitor the current value contained in the variable **encoderCount**. Notice that this happens after 10 seconds of time have elapsed. In other words, the **encoderCount** variable will contain the number of pulses during 10 seconds of running. Then the loop resets **encoderCount** to zero to start counting pulses for the next 10 seconds.

Of course during the running of the robot, you probably won't want to print values to the serial monitor (unless you are going to run along with the robot!). You can use the value stored in **encoderCount** to determine how far the robot has traveled.

```
/*  Arduino Code for AndyMark Hall Effect 2 channel encoder am-2816a

This simple code reads only one channel of the encoder. Attach one of
the channel wires of encoder to Arduino digital pin number 2.  Code
does not identify direction of encoder rotation and will not give a
proper count of encoder if direction of rotation changes.

Code by Jeffrey La Favre, October 12, 2018

*/

volatile unsigned int encoderCount = 0;

void setup() {

 pinMode(2, INPUT);

 attachInterrupt(digitalPinToInterrupt(2), readEncoder, RISING);

 Serial.begin (9600);

}

void loop() {

 delay(10000);                              //print out every 10 seconds

 Serial.println (encoderCount, DEC);        //print encoder count

 encoderCount = 0;                          //reset encoder count

}

void readEncoder() {

 encoderCount = encoderCount + 1;           // add one to encoder count

 }
```