旋转编码器

旋转编码器可通过旋转可以计数正方向和反方向转动过程中输出脉冲的次数，旋转计数不像电位计，这种转动计数是没有限制的。配合旋转编码器上的按键，可以复位到初始状态，即从 0 开始计数。

工作原理： 增量编码器是一种将旋转位移转换为一连串数字脉冲信号的旋转式传感器。这些脉冲用来控制角位移。在 Eltra 编码器中角位移的转换采用了光电扫描原理。读数系统以由交替的透光窗口和不透光窗口构成的径向分度盘（码盘）的旋转为依据，同时被一个红外光源垂直照射，光把码盘的图像投射到接收器表面上。接收器覆盖着一层衍射光栅，它具有和码盘相同的窗口宽度。接收器的工作是感受光盘转动所产生的变化，然后将光变化转换成相应的电变化。再使低电平信号上升到较高电平，并产生没有任何干扰的方形脉冲，这就必须用电子电路来处理。读数系统通常采用差分方式，即将两个波形一样但相位差为 180°的不同信号进行比较，以便提高输出信号的质量和稳定性。读数是再两个信号的差别基础上形成的，从而消除了干扰。

增量编码器

顺时针运动	逆时针运动
A B 1 1 0 1 0 0 1 0	A B 1 1 1 0 0 0 0 1

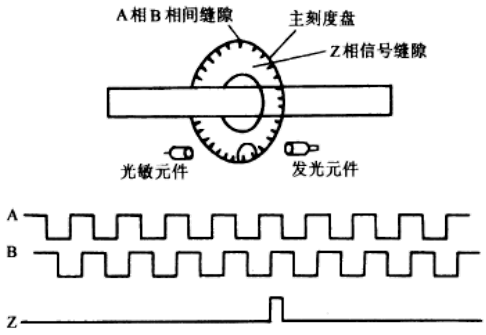
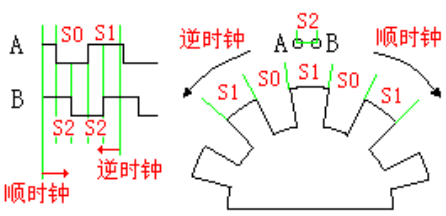
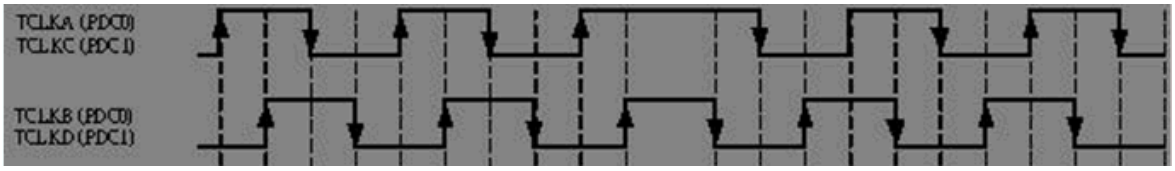


图 3 光电编码器工作原理图及输出波形

增量编码器给出两相方波，它们的相位差 90°，通常称为 A 通道和 B 通道。其中一个通道给出与转速相关的信息，与此同时，通过两个通道信号进行顺序对比，得到旋转方向的信息。还有一个特殊信号称为 Z 或零通道，该通道给出编码器的绝对零位，此信号是一个方波与 A 通道方波的中心线重合。



增量型编码器精度取决于机械和电气两种因素，这些因素有：光栅分度误差、光盘偏心、轴承偏心、电子读数装置引入的误差以及光学部分的不精确性。确定编码器精度的测量单位是电气上的度数，编码器精度决定了编码器产生的脉冲分度。以下用 360°电气度数来表示机械轴的转动，而轴的转动必须是一个完整的周期。要知道多少机械角度相当于电气上的 360 度，可以用下列公式来计算： 电气 360 =机械 360°/n°脉冲/转

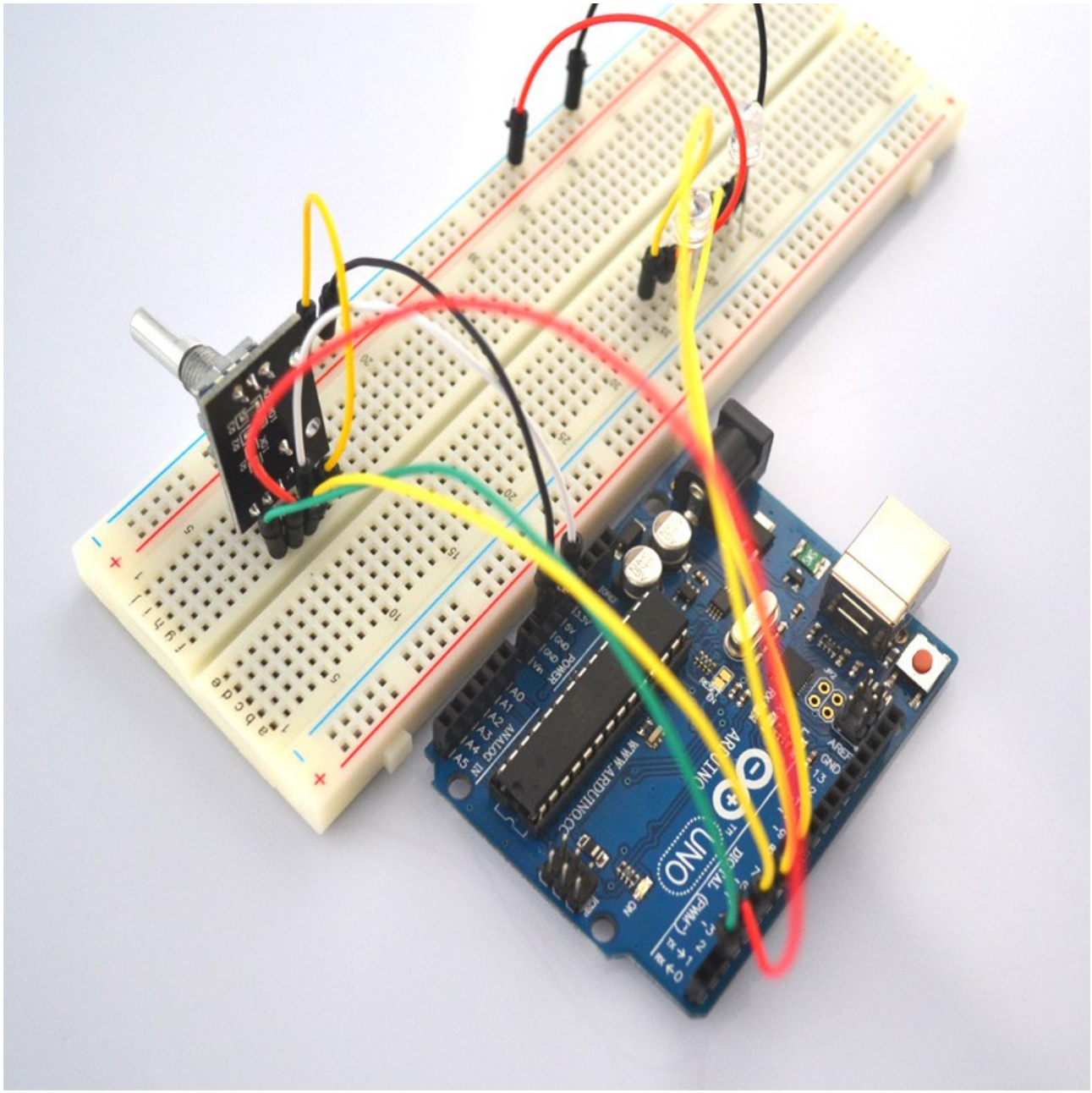


图：A、B 换向时信号

编码器分度误差是以电气角度为单位的两个连续脉冲波的最大偏移来表示。误差存在于任何编码器中，这是由前述各因素引起的。Eltra 编码器的最大误差为±25 电气角度（在已声明的任何条件下），相当于额定值偏移±7%，至于相位差 90°（电气上）的两个通道的最大偏差为±35 电气度数相当于额定值偏移±10%左右。

UVW 信号增量型编码器

除了上述传统的编码器外，还有一些是与其它的电气输出信号集成在一起的增量型编码器。与 UVW 信号集成的增量型编码器就是实例，它通常应用于交流伺服电机的反馈。这些磁极信号一般出现在交流伺服电机中，UVW 信号一般是通过模拟磁性原件的功能而设计的。在 Eltra 编码器中，这些 UVW 信号是用光学方法产生，并以三个方波的形式出现，它们彼此偏移 120°。为了便于电机启动，控制电动机用的启动器需要这些正确的信号。这些 UVW 磁极脉冲可在机械轴旋转中重复许多次，因为它们直接取决于所连接的电机磁极数，并且用于 4、6 或更多极电机的 UVW 信号。



****

ARDUINO 测试代码:

```
int redPin = 2;
int yellowPin = 3;
int greenPin = 4;
int aPin = 6;
int bPin = 7;
int buttonPin = 5;

int state = 0;
int longPeriod = 5000; // Time at green or red
int shortPeriod = 700; // Time period when changing
int targetCount = shortPeriod;
int count = 0;
```

```
void setup()
{
  pinMode(aPin, INPUT);
  pinMode(bPin, INPUT);
  pinMode(buttonPin, INPUT);
  pinMode(redPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
}
void loop()
{
  count++;
  if (digitalRead(buttonPin))
  {
    setLights(HIGH, HIGH, HIGH);
  }
  else
  {
    int change = getEncoderTurn();
    int newPeriod = longPeriod + (change * 1000);
    if (newPeriod >= 1000 && newPeriod <= 10000)
    {
      longPeriod = newPeriod;
    }
    if (count > targetCount)
    {
      setState();
      count = 0;
    }
  }
  delay(1);
}

int getEncoderTurn()
{
  // return -1, 0, or +1
  static int oldA = LOW;
  static int oldB = LOW;
  int result = 0;
  int newA = digitalRead(aPin);
  int newB = digitalRead(bPin);
  if (newA != oldA || newB != oldB)
  {
    // something has changed
    if (oldA == LOW && newA == HIGH)
```

```
    {
      result = -(oldB * 2 - 1);
    }
  }
  oldA = newA;
  oldB = newB;
  return result;
}

int setState()
  {
    if (state == 0)
    {
      setLights(HIGH, LOW, LOW);
      targetCount = longPeriod;
      state = 1;
    }
    else if (state == 1)
    {
      setLights(HIGH, HIGH, LOW);
      targetCount = shortPeriod;
      state = 2;
    }
    else if (state == 2)
    {
      setLights(LOW, LOW, HIGH);
      targetCount = longPeriod;
      state = 3;
    }
    else if (state == 3)
    {
      setLights(LOW, HIGH, LOW);
      targetCount = shortPeriod;
      state = 0;
    }
  }

void setLights(int red, int yellow, int green)
{
  digitalWrite(redPin, red);
  digitalWrite(yellowPin, yellow);
  digitalWrite(greenPin, green);
}
```