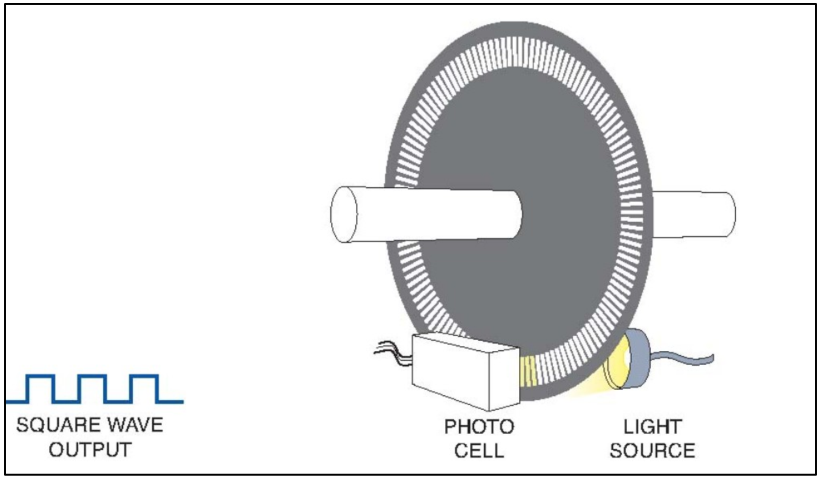
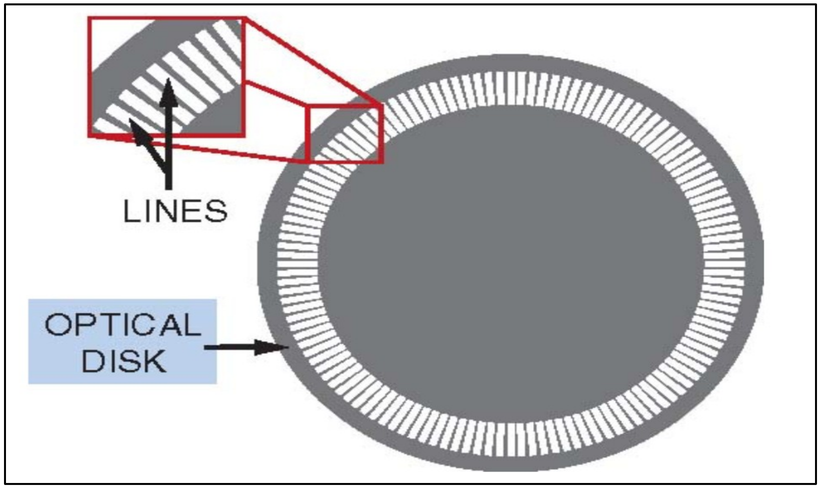
OPTICAL ENCODER – WORKING PRINCIPLE

- An infrared light beam is passed through an encoder disk with multiple openings
- The openings interrupt continuous beam from light source detected by a photo detector
- These events produce transitions from light to dark and is captured by the photo detector
- Encoder's current position and direction are then calculated by counting transitions and state of the previous transitions



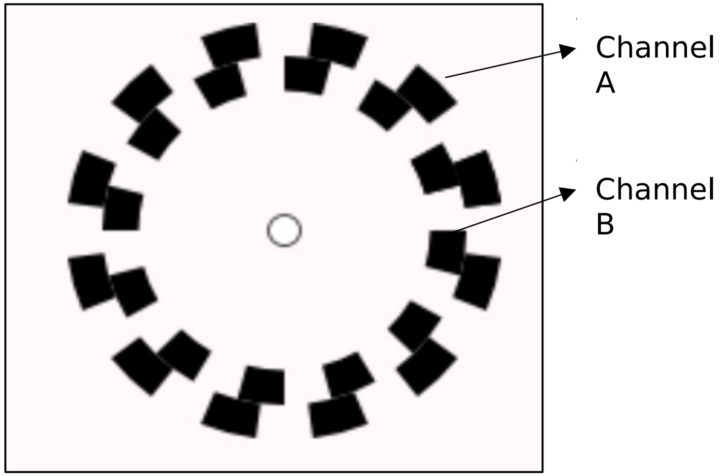


Ref: http://blog.nidec-avtron.com/encoders/how-optical-encoders-work

QUADRATURE MODE

- A two track encoder with channels A and B can be in four possible states
- Rotation direction is determined by the order of the state transitions
- Encoder's position is derived by counting each state transitions
- A common optical encoders (in AGV4000) has 600 openings on each channel
- One openings on each channel (total of 4 combinations) can produce 4 states. So, maximum count for each encoder rotation is 600x4 = 2400, e.g. encoder maximum count per rotation = 2400 steps

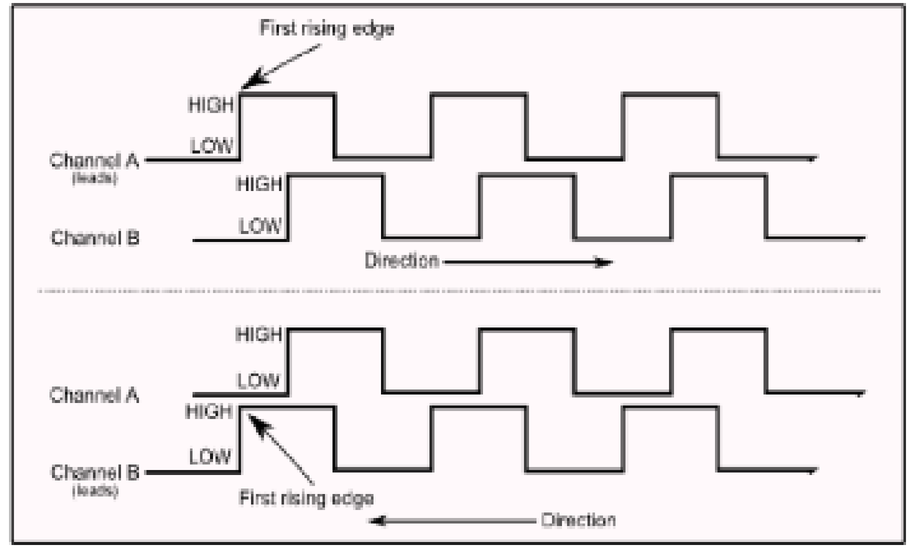
Counter-clockwise rotation		
Phase	**Channel A**	**Channel B**
1	0	0
2	0	1
3	1	1
4	1	0
Clockwise rotation		
Phase	**Channel A**	**Channel B**
1	1	0
2	1	1
3	0	1
4	0	0



Ref: http://www.robotoid.com/appnotes/circuits-quad-encoding.html

Quad Mode Characteristics

- The two channels are out of phase by 90 degrees
- The order in which the signal changes from low to high indicates direction of rotation
- Pins 2 and 3 of ARNOD1 and ARNOD2 are connected to channel A and channel B of encoders on each wheel of AGV4000
- The changes in signal can be accurately captured by attaching interrupts to pins 2 and 3 of the MCU
- So, every time there is an interrupt from the pins 2 and 3, a counter is incremented or decremented depending on the direction of rotation, as soon as the total number of pulses reach 2400, then one revolution is reached.



Ref: http://www.robotoid.com/appnotes/circuits-quad-

```
//       bitB             _____          ______
//              Pin2 ____|     |_____|      |_____ Pin2
// negative <---    _____        _____        __     --> positive
//              Pin3 __|     |_____|     |_____|   Pin3
//       bitA

               //    new    new    old    old
               //    pin2   pin3   pin2   pin3   Result
               //    ----   ----   ----   ----   ------
               //    0      0      0      0      no movement
               //    0      0      0      1      +1
               //    0      0      1      0      -1
               //    0      0      1      1      +2  (assume pin2 edges only)
               //    0      1      0      0      -1
               //    0      1      0      1      no movement
               //    0      1      1      0      -2  (assume pin2 edges only)
               //    0      1      1      1      +1
               //    1      0      0      0      +1
               //    1      0      0      1      -2  (assume pin2 edges only)
               //    1      0      1      0      no movement
               //    1      0      1      1      -1
               //    1      1      0      0      +2  (assume pin2 edges only)
               //    1      1      0      1      -1
               //    1      1      1      0      +1
               //    1      1      1      1      no movement
```

Waveform Analysis Example

Example:
1. 2 bits for outer and inner opening states;
2. Plus history of outer and inner openings;
3. So the total number of transition is 4 bits, hence 16 states.

Previous state:

S0p	0 0
S1p	0 1
S2p	1 0
S3p	1 1

Current state:

S0c(S4)	0 0
S1c(S5)	0 1
S2c(S6)	1 0
S3c(S7)	1 1

```
//                          _______         _______
//             Pin2 _____|       |_______|       |_____ Pin2
// negative <---       ______         ______         __      --> positive
//             Pin3 __|       |______|       |______|   Pin3

                //   new    new    old    old
                //   pin2   pin3   pin2   pin3   Result
                //   ----   ----   ----   ----   ------
                //   0      0      0      0      no movement
                //   0      0      0      1      +1
                //   0      0      1      0      -1
                //   0      0      1      1      +2  (assume pin2 edges only)
                //   0      1      0      0      -1
                //   0      1      0      1      no movement
                //   0      1      1      0      -2  (assume pin2 edges only)
                //   0      1      1      1      +1
                //   1      0      0      0      +1
                //   1      0      0      1      -2  (assume pin2 edges only)
                //   1      0      1      0      no movement
                //   1      0      1      1      -1
                //   1      1      0      0      +2  (assume pin2 edges only)
                //   1      1      0      1      -1
                //   1      1      1      0      +1
                //   1      1      1      1      no movement
```

Ref: http://www.robotoid.com/appnotes/circuits-quad-encoding.html

FSM Approach

```
//                        ______        ______
//             Pin2 _____|      |______|      |_____ Pin2
// negative <---      ______        ______        __      --> positive
//             Pin3 __|      |______|      |______|   Pin3
```

Previous state:

S0p	0 0
S1p	0 1
S2p	1 0
S3p	1 1

Current state:

S0c(S4)	0 0
S1c(S5)	0 1
S2c(S6)	1 0
S3c(S7)	1 1

Current state:

S0c(S4)	0 0
S1c(S5)	0 1
S2c(S6)	1 0
S3c(S7)	1 1

		f_(+1)	f_(+2)	f_(-1)	f_(-2)	f_0
S0c(S4)	S0p					1
S0c(S4)	S1p					
S0c(S4)	S2p					
S0c(S4)	S3p					
S1c(S5)	S0p					
S1c(S5)	S1p					1
S1c(S5)	S2p					
S1c(S5)	S3p					
S2c(S6)	S0p					
S2c(S6)	S1p					
S2c(S6)	S2p					1
S2c(S6)	S3p					
S3c(S6)	S0p					
S3c(S6)	S1p					
S3c(S6)	S2p					
S3c(S6)	S3p					1

Inference Engine

1 Table 1. Attribute Table

	Ceiling Lights (class w1)	Window Lights (class w2)
Shape	Rectangles x1	Rectangles x1
	Ellipses x2	Ellipses x2
	Circles x3	Circles x3
Location	Anywhere x4 smaller part image x5	Anywhere x4 smaller part image x5
Color	white x6	white x6
Repeated Pattern	maybe x7	maybe x7

So decision function 3

$$\boxed{f(X) = x1\ x5\ x6 + x2\ x6 + x3\ x6 + x5\ x6 \quad \ldots (1)}$$

C/c++ implementation of the inference engine (switching function) 4

Table 2. Identification Table 2

	x1 rect	x2 elli	x3 cir	x4 loc	x5 sml	x6 wht	x7 rep	f(X)
x1 x5 x6	1	D	D	D	1	1	D	1
x2 x6	D	1	D	D	D	1	D	1
x3 x6	D	D	1	D	D	1	D	1
x5 x6	D	D	D	D	1	1	D	1

Define primary implicant, removal of any of its column will result in the mis-identification of f(X)

No: C/C++ Inference Engine

```
#include<stdio.h>
int And(int a, int b);
int Or(int a, int b);
int Not(int a);
void main()
{
 ///where main body of code will go
}
int And(int a, int b)
{
 int output;
 if(a==0 && b==0)
  output=0;
  if(a==1 && b==0)
  output=0;
 if(a==0 && b==1)
  output=0;
 if(a==1 && b==1)
  output=1;
 return (output);
}
```

Simplify it 1. as boolean; 2. logically as &&

```
int Or(int a, int b)
{
 int output;
 if(a==0 && b==0)
  output=0;
  if(a==1 && b==0)
  output=1;
 if(a==0 && b==1)
  output=1;
 if(a==1 && b==1)
  output=1;
 return (output);
}
int Not(int a)
{
int output;
 if(a==0 )
  output=1;
 if(a==1 )
  output=0;
  return (output);
}
```

In fact C/C++ support all the boolean logic operators, so build inference engine should be straight forward

Simplify it 1. as boolean;

```
int And(int a, int b)
{
    return a && b;
}
```

```
return Not(And(a, b));
```

Build NAND, NOR, XOR etc

Harry Li, Ph.D

C/C++ Bitwise Operators

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

```
// C Program to demonstrate the working of logical operators
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;
    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) equals to %d \n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) equals to %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) equals to %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) equals to %d \n", result);

    result = !(a != b);
    printf("!(a == b) equals to %d \n", result);

    result = !(a == b);
    printf("!(a == b) equals to %d \n", result);
    return 0;
}
```

C/C++ Inference Engine

```c
//----Inference Engine to find reflection spots---//
//----April 7, 2018, by HL, version 0x0.1; -------//
 #include <stdio.h>
#include <stdbool.h>
#define dimension 100
bool    x[dimension], f_identification;
int     item;

int main()
{
    printf("Inference Engine to identify reflections \n");
    printf("x1 rectangle? 1 for Y or 0 for N \n");
    scanf("%i",&item);
    if (item == 1) x[1] = true;
    if (item == 0) x[1] = false;

    printf("x2 ellips? 1 for Y or 0 for N \n");
    scanf("%i",&item);
    if (item == 1) x[2] = true;
    if (item == 0) x[2] = false;

    printf("x3 circle? 1 for Y or 0 for N \n");
    scanf("%i",&item);
    if (item == 1) x[3] = true;
    if (item == 0) x[3] = false;

    printf("x4 location? 1 for Y or 0 for N \n");
    scanf("%i",&item);
    if (item == 1) x[4] = true;
    if (item == 0) x[4] = false;

printf("x5 small size? 1 for Y or 0 for N \n");
    scanf("%i",&item);
    if (item == 1) x[5] = true;
    if (item == 0) x[5] = false;

    printf("x6 white color? 1 for Y or 0 for N \n");
    scanf("%i",&item);
    if (item == 1) x[6] = true;
    if (item == 0) x[6] = false;

    printf("x7 repetative? 1 for Y or 0 for N \n");
    scanf("%i",&item);
    if (item == 1) x[7] = true;
    if (item == 0) x[7] = false;

    f_identification = (x[1] && x[5] && x[6])
                   || (x[2] && x[6])
                   || (x[3] && x[6])
                   || (x[5] && x[6]);

    if (f_identification){
    printf("The object is reflection\n");}
    else {
    printf("The object is not reflection\n");}

    return 0;
}
```

Optic Encoder