**Explicando como o Encoder está codificado:**

**Não** uso interrupção **nem uma biblioteca específica** para encoder. **Simplesmente leio os pinos do encoder.**

O primeiro passo é informar ao Arduino que os pinos do Encoder são somente de ENTRADA (INPUT)   (Função setup() )

pinMode(ENCODER\_PIN\_A, INPUT);

pinMode(ENCODER\_PIN\_B, INPUT);

Depois declaro as variáveis de controle do encoder (incluindo o tempo mínimo para a processar a próxima ação)

long elapsedTimeEncoder = millis();

// Encoder variable control

unsigned char encoder\_pin\_a;

unsigned char encoder\_prev = 0;

unsigned char encoder\_pin\_b;

O tipo **unsigned char** é na realidade um número inteiro, não sinalizado de 8 bits. Ele admite valores entre 0 e 255.  Não uso as variáveis do encoder para tratar caracteres. São números que, no caso do encoder, terão valores 0 (nível lógico baixo) ou diferente de 0 (nível lógico alto). Poderia ser um int. A questão aqui é de economia de memória e de cíclo de máquina.

a variável **encoder\_prev** armazena a última ação do encoder. Dessa forma, na ação seguinte do encoder eu terei o valor da ação anterior para poder fazer comparação;

Na função  loop(), o processo inicia checando se já ocorreu mais de 5 milisegundos após a última ação do encoder.  Caso não tenha ocorrido mais de 5 milisegundos, não processa a ação do encoder (não leio os pinos). Isso é útil para eliminar efeitos indesejáveis causados pelo mal contato das placas metálicas do encoder (ver Debounce);

if ((millis() - elapsedTimeEncoder) > 5)

{

encoder\_pin\_a = digitalRead(ENCODER\_PIN\_A);

encoder\_pin\_b = digitalRead(ENCODER\_PIN\_B);

if ((!encoder\_pin\_a) && (encoder\_prev)) // has ENCODER\_PIN\_A gone from high to low?

{ // if so, check ENCODER\_PIN\_B. It is high then clockwise (1) else counter-clockwise (-1)

changeFreq(((encoder\_pin\_b) ? 1 : -1));

}

encoder\_prev = encoder\_pin\_a;

elapsedTimeEncoder = millis(); // keep elapsedTimeEncoder updated

}

caso o tempo decorrido entre a última ação do encoder e o tempo atual for maior que 5 milissegundos, eu leio os dois pinos em que o encoder está conectado;

se o PINO A (encoder\_pin\_a) estiver em nível lógico baixo (0) e o encoder\_prev (ação anterior deste pino) estiver em nível lógico alto (1), então é porque houve mudança no status do encoder (houve giro);

Lemvrando que   !**encoder\_pin\_a** poderá ter os seguintes resultados: se encoder\_pin\_a for 0, o resultado será 1 (verdadeiro); se  encoder\_pin\_a for 1, o resultado será 0 (falso);

Se a expressão **if ((!encoder\_pin\_a) && (encoder\_prev))** for verdadeira (houve giro no encoder). Agora vamos determinar qual o sentido (horário ou anti-horário). Quem vai dizer isso é estado do **encoder\_pin\_b**.

Se **encoder\_pin\_b** estiver em nível lógico alto (1), então é sentido horário. Caso contrário, nível lógico baixo (0), é anti-horário.

A linguagem C permite uma atribuição condicional na forma apresentada no exemplo a seguir:  X = ( A > B)? 1:-1); Neste exemplo, X recebe 1 se A for maior que B; caso contrário, recebe -1;

Observe a expressão dentro da chamada da função changeFreq():

((encoder\_pin\_b) ? 1 : -1)

A função changeFreq() recebe dois possíveis valores como parâmero: 1 para incrementar (adicionar o step) a frequência (sentido horário do encoder); e -1 para decrementar (sentido anti-horário).

se **encoder\_pin\_b** for 1 (ou qualquer valor diferente de 0), então changeFreq() será chamada com o valor 1 (equivalente à chamada changeFreq(1)). Caso contrário, **encoder\_pin\_b** for igual a 0 changeFreq() será chamada com o valor -1 (equivalente à chamada changeFreq(-1)).

Note que dentro do bloco atendida pela condição do encoder **[if ((millis() - elapsedTimeEncoder) > 5)**]  , eu armazeno a ação corrente na variável **encoder\_prev**. E também atualizo o tempo em elapsedTimeEncoder.

encoder\_prev = encoder\_pin\_a;

elapsedTimeEncoder = millis(); // keep elapsedTimeEncoder updated

com isso, após 5 milissegundos, poderei ler os pinos do encoder novamente e verificar se ouve uma ação de giro.