Wellter Mompean Sozin	188625	Turma U
Guilherme Augusto Amorim Terrell	168899	Turma U

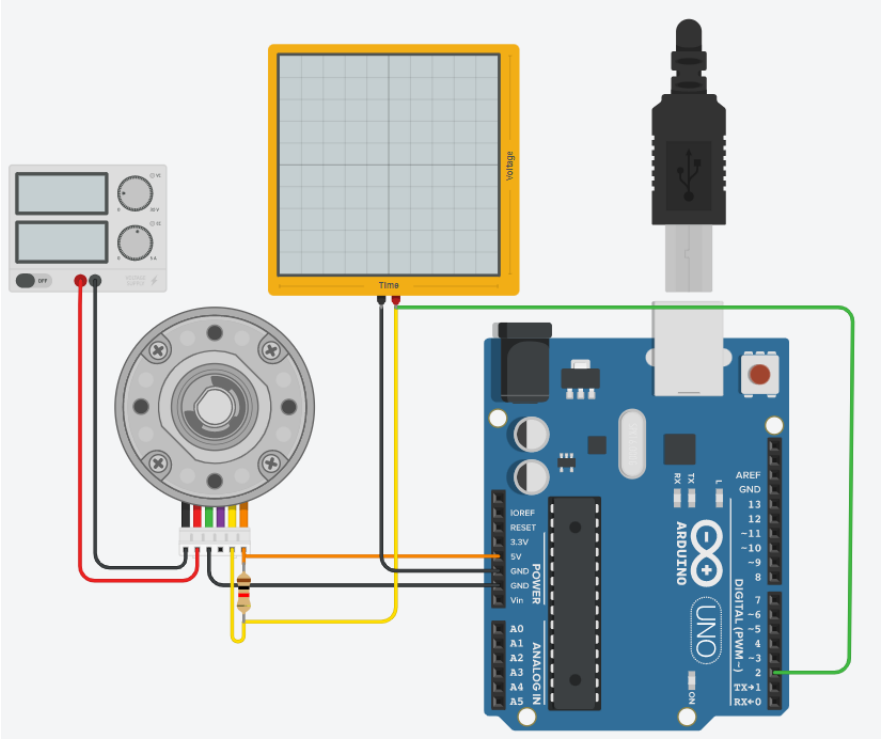
EA075 – Laboratório de Sistemas Embarcados
Prof. Eric Rohmer
Data de entrega: 11/05/2022

Projeto 2 (parte 2)

Função para Estimar RPM e Controlar Giro do Motor

Função para Estimar RPM

Para montar um código de estimação de velocidade em RPM de um motor, nos baseamos no circuito do Tinkercad mostrado na figura que se segue:



Nesse circuito, o encoder do motor é conectado na porta 2 do Arduino UNO. A implementação do código se deu utilizando-se basicamente de três mecanismos. O mecanismo de interrupção por borda de subida do pino 2 para detectar toda vez em que a pá/furo do encoder passe pelo codificador óptico; um mecanismo de base de tempo implementado por um temporizador com interrupções periódicas de 8 ms; e a função propriamente dita de cálculo da estimativa de velocidade do motor em RPM.

Dessa forma, temos o seguinte funcionamento: a partir da interrupção configurada no pino 2 onde é ligado o encoder, realizamos a contagem de quantas vezes o codificador óptico identifica a passagem de um furo/pá do encoder, utilizando-se da base de tempo de 8 ms, a cada 200ms utilizamos essa quantidade para estimar a velocidade utilizando-se da seguinte equação:

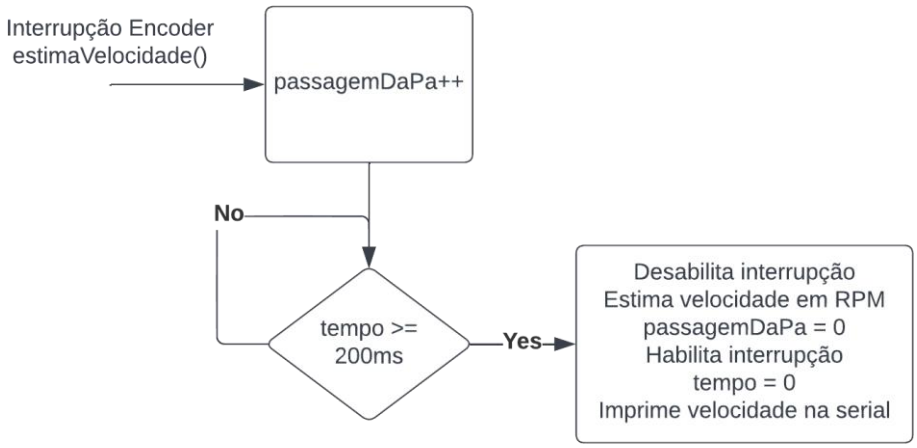
$$VelocidadeEstimadaRPM = \frac{\left(\frac{60 \times 1000}{Nº\ de\ furos\ do\ encoder}\right)}{N \times 8} \times Nº\ de\ passagens\ dos\ furos$$

Onde 1000/(N*8) representa a quantidade de vezes por segundo em que a velocidade do motor é verificada/estimada. Nesse caso, estamos utilizando N=25 e, portanto, estimamos a velocidade 5 vezes a cada segundo.

Essa equação utilizada, permite estimarmos a velocidade do motor relacionando a quantidade de passagens dos furos pela quantidade total de furos que o encoder possui. Essa quantidade de passagens dos furos foi observada num período e, portanto, precisamos dividir por esse período e, para sairmos da unidade

de rotações por milissegundos, precisamos usar o fator 60*1000 para chegarmos na unidade de rotações por minuto (RPM).

A seguir é apresentado um fluxograma do funcionamento básico do código implementado.



A quantidade de vezes por segundo em que estimamos a velocidade não deve ser grande a ponto do tempo entre uma estimativa e outra ser menor que a base de tempo de 8 ms do código e não deve ser grande a ponto de não identificar nenhum furo/pá do encoder quando o motor estiver em baixas rotações. Desse modo, para permitir uma boa precisão, estamos estimando a velocidade 5 vezes por segundo usando um N=25.

No seguinte link pode-se encontrar o circuito com o código implementado no Tinkercad:

https://www.tinkercad.com/things/8EXcSPzWIuB?sharecode=326O3zjO1j4PkBtj2yuZIOUBqL1UruqRx9wQbci49kU

- Código completo da estimativa de velocidade

```
// Wellter Mompean Sozin - 188625
// Guilherme Augusto Amorim Terrell - 168899



#define pinEncoder 2


unsigned int pulsosPorVolta = 60; // Numero de pas/furos no encoder
int velocidadeEstimadaRPM = 0;
int counter = 0;
volatile unsigned long passagemDaPa = 0;
unsigned long verificaVelocidade = 25;
```

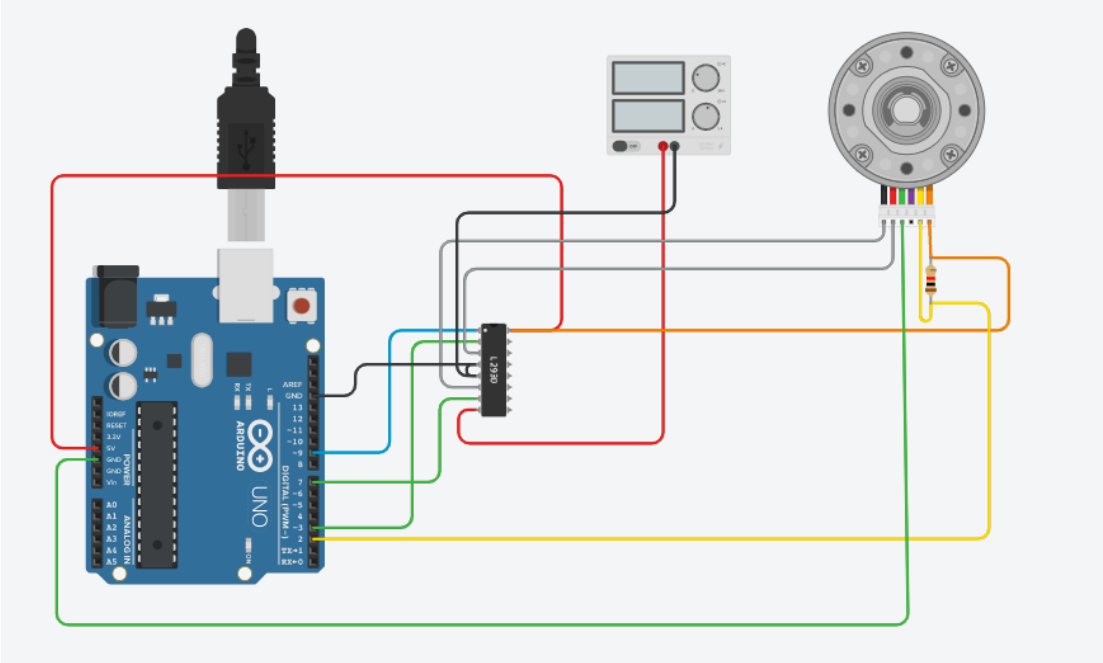
```
// Rotina de servico de interrupcao do temporizador 0
ISR(TIMER0_COMPA_vect){
  counter++;
}

// Rotina de servico de interrupcao do pinEncoder
void estimaVelocidade(){
  passagemDaPa++;
}



void setup(){
  Serial.begin(9600);
  pinMode(pinEncoder, INPUT);
  cli();
  configuracao_Timer0();
  sei();

  // Habilita interrupcao do encoder
  attachInterrupt(digitalPinToInterrupt(pinEncoder), estimaVelocidade, RISING);
}



void loop(){
  _delay_ms(1);

  /* counter = 125 -> 1s */
  /* counter = 25 -> 0.2s */
  /* Durante 0.2s segundo incremento passagemDaPa */
```

```
        /* passageDaPa indica o número de voltas foram dadas em 0.2s */
        /* Multiplicar por 5 para estimar a rotação em 1s */
        if(counter >= verificaVelocidade){
          detachInterrupt(digitalPinToInterrupt(pinEncoder)); // Desabilita interrupcao para calcular a velocidade
          velocidadeEstimadaRPM = (60*1000/pulsosPorVolta)/(verificaVelocidade*8) * passagemDaPa;
          passagemDaPa = 0;
          attachInterrupt(digitalPinToInterrupt(pinEncoder), estimaVelocidade, RISING); // Habilita interrupcoes apos calculo da velocidade
          counter = 0;
          Serial.println(velocidadeEstimadaRPM);
        }
      }


      void configuracao_Timer0(){

////////////////////////////////////////////////////////////////////////////////////////////////////////////
        // Configuracao Temporizador 0 (8 bits) para gerar interrupcoes periodicas a cada 8ms no modo Clear Timer on Compare Match (CTC)
        // Relogio = 16e6 Hz
        // Prescaler = 1024
        // Faixa = 125 (contagem de 0 a OCR0A = 124)
        // Intervalo entre interrupcoes: (Prescaler/Relogio)*Faixa = (64/16e6)*(124+1) = 0.008s

        // TCCR0A – Timer/Counter Control Register A
        // COM0A1 COM0A0 COM0B1 COM0B0 – – WGM01 WGM00
        // 0     0     0     0        1     0
        TCCR0A = 0x02;

        // OCR0A – Output Compare Register A
        OCR0A = 124;
```

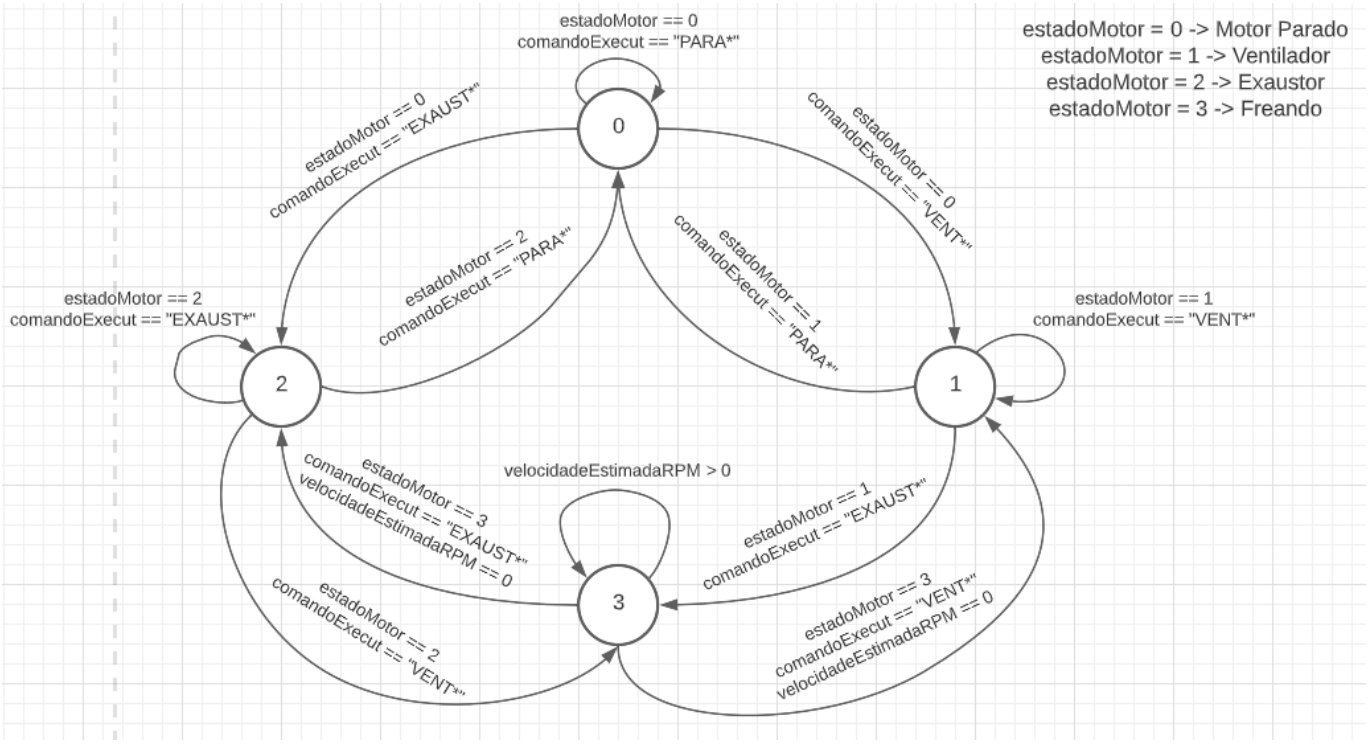
```
// TIMSK0 – Timer/Counter Interrupt Mask Register
// – – – – – OCIE0B OCIE0A TOIE0
// – – – – – 0     1     0
TIMSK0 = 0x02;

// TCCR0B – Timer/Counter Control Register B
// FOC0A FOC0B – – WGM02 CS02 CS01 CS0
// 0    0       0    1    0    1
TCCR0B = 0x05;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
}
```

Função para Controlar Giro do Motor

Foram implementadas 3 funções que juntas controlam o sentido de rotação e a velocidade de giro do motor cc: maqEstadosMotor(), rotacaoMotor() e convertPercentPWM(). Abaixo temos o circuito implementado no TinkerCad, que também pode ser acessado no link: https://www.tinkercad.com/things/cjZmahCeehg. Para fins de simulação o estado inicial é estadoMotor = 1 e velocidade de rotação começa em 100 (devido a um bug no tinkercad, se o motor parar em algum momento, o mesmo não gira mais durante a simulação).



A função maqEstadosMotor() descreve através da implementação de uma máquina de estados o comportamento que deve ser executado pelo motor CC. Essa função trabalha com duas variáveis: estadoMotor, que indica o estado atual do motor CC na máquina de estados, e a variável comadoExecut, que recebida via comunicação serial. Abaixo temos o diagrama de estados e o pseudocódigo que representam essa função. Note que não existe caminho direto de 1 para 2 ou de 2 para 1 (ou seja, o motor obrigatoriamente deve ter velocidade de rotação igual a zero antes de inverter o sentido de rotação).




```
void maqEstadosMotor(){
 Se comandoExecut == "PARA*"{
  /*Motor parado*/
  estadoMotor = 0;
 }
 Se estadoMotor == 0 E comando == "VENT*"{
  estadoMotor = 1;
  limpar string comando;
 }
 Se estadoMotor == 0 E comando == "EXAUST*"{
  estadoMotor = 2;
  limpar string comando;
 }
```

```
 Se estadoMotor == 1 E comando == "VENT*"{
  estadoMotor = 1;
  limpar string comando;
 }
 Se estadoMotor == 2 E comando == "EXAUST*"{
  estadoMotor = 2;
  limpar string comando;
 }
 Se estadoMotor == 1 E comando == "EXAUST*"{
  /*Freando o motor*/
  estadoMotor = 3;
  /*Mudar sentido de rotação apenas apos parar de girar*/
  Se velocidadeEstimadaRPM == 0{
   estadoMotor = 2;
  }
  Se velocidadeEstimadaRPM > 0{
   estadoMotor = 1;
  }
 }
 Se estadoMotor == 2  E comando == "VENT*"{
  /*Freando o motor*/
  estadoMotor = 3;
  /*Mudar sentido de rotação apenas apos parar de girar*/
  Se velocidadeEstimadaRPM == 0 {
   estadoMotor = 1;
  }
  Se velocidadeEstimadaRPM > 0 {
   estadoMotor = 2;
  }
 }
}
```

A função rotacaoMotor() é responsável por controlar o sentido de giro do motor CC ou por acionar o modo de freio do mesmos (a depender do valor da variavel estadoMotor). Abaixo temos o pseudoCódigo dessa função.

```
void rotacaoMotor(){
  Se (estadoMotor == 0){
    /*motor desligado*/
    escrever nível lógico zero no pino 1 da ponte H;
    escrever nível lógico zero no pino 2 da ponte H;
    escrever 0 no pino PWM
  }
  Se (estadoMotor == 1){
    /*Motor como ventilador*/
    escrever nível lógico baixo no pino 1 da ponte H;
    escrever nível lógico alto no pino 2 da ponte H;
    escrever velocidade desejada no pino PWM
  }
  Se (estadoMotor == 2){
    /*Motor como exaustor*/
    escrever nível lógico alto no pino 1 da ponte H;
    escrever nível lógico baixo no pino 2 da ponte H;
    escrever velocidade desejada no pino PWM
  }
  Se (estadoMotor == 3){
    /*Frear motor*/
    escrever nível lógico alto no pino 1 da ponte H;
    escrever nível lógico alto no pino 2 da ponte H;
    escrever 0 no pino PWM
  }
}
//Fim
```

A função convertPercentPWM() é responsável por ler o comando “VEL XXX*”, extrair o valor da porcentagem e converte-lo em um inteiro dentro da faixa aceitável do nível PWM (0~254). Abaixo temos o pseudocódigo dessa função.

```
/*Função que converte o valor contido na string comando*/
/*em um valor dentro da escala PWM (0, 254)*/
/*valorMaximoPWM = 254*/
void convertPercentPWM(){
  Se comando recebido == "VEL XXX"{
    int velocidadePorcentagem = XXX convertido para inteiro
    velocidadeMotorPWM = (velocidadePorcentagem*valorMaximoPWM)/100;
  }
}
```

- Código completo controle de giro do motor

```
// Wellter Mompean Sozin - 188625
// Guilherme Augusto Amorim Terrell - 168899

#define pinEntrada1PonteH 3
#define pinEntrada2PonteH 7
#define pinPWM 9
#define pinEncoder 2
#define valorMaximoPWM 254

/*Para fins de simução a velocidade começa em 100 e o estadoMotor começa em 1*/
int velocidadeMotorPWM = 100;
int estadoMotor = 1; /*0 -> motor parado; 1 -> ventilador; 2 -> exaustor; 3 -> freio*/
String comandoExecut = "";
/*Função que converte o valor contido na string comando*/
/*em um valor dentro da escala PWM (0, 254)*/
void convertPercentPWM(){
  if(comando.substring(0,4) == "VEL "){
    int velocidadePorcentagem = comando.substring(4,7).toInt();
```

```
   velocidadeMotorPWM = (velocidadePorcentagem*valorMaximoPWM)/100;
  }
 }

/*Função que define o funcionamento do motor CC*/
/*comando = VENT* -> motor opera como ventilador (girar sentido horário)*/
/*comando = EXAUST* -> motor opera como exaustor (girar sentido anti-horário)*/
/*comando = PARA* -> motor parado*/

void maqEstadosMotor(){
  if(comandoExecut == "PARA*"){
    /*Motor parado*/
    estadoMotor = 0;
  }
  if((estadoMotor == 0) && (comandoExecut == "VENT*")){
    /*motor que estava parado deve operar como ventilador*/
    estadoMotor = 1;
    comandoExecut = "";
  }
  if((estadoMotor == 0) && (comandoExecut == "EXAUST*")){
    /*motor que estava parado deve operar como exaustor*/
    estadoMotor = 2;
    comandoExecut = "";
  }
  if((estadoMotor == 1) && (comandoExecut == "VENT*")){
    /*continuar operando como ventilador*/
    estadoMotor = 1;
    comandoExecut = "";
  }
  if((estadoMotor == 2) && (comandoExecut == "EXAUST*")){
    /*contnuar operando como exaustor*/
```

```
  estadoMotor = 2;
  comandoExecut = "";
 }
 if((estadoMotor == 1 || estadoMotor == 3) && (comandoExecut == "EXAUST*")){
  /*motor que estava operando como ventilador deve operar como exaustor*/
  /*Freando o motor*/
  estadoMotor = 3;
  /*Mudar sentido de rotação apenas apos parar de girar*/
  if(velocidadeEstimadaRPM == 0){
   estadoMotor = 2;
  }
  /*Caso não tenha parado deve continuar freando*/
  if(velocidadeEstimadaRPM > 0){
   estadoMotor = 3;
  }
 }
 if((estadoMotor == 2 || estadoMotor == 3) && (comandoExecut == "VENT*")){
  /*motor que estava operando como exaustor deve operar como ventilador*/
  /*Freando o motor*/
  estadoMotor = 3;
  /*Mudar sentido de rotação apenas apos parar de girar*/
  if(velocidadeEstimadaRPM == 0){
   estadoMotor = 1;
  }
  /*Caso não tenha parado deve continuar freando*/
  if(velocidadeEstimadaRPM > 0){
   estadoMotor = 3;
  }
 }
}
```

```
/*Função que habilita o sentido de rotação de acordo com o estado do motor*/
void rotacaoMotor(){
 if(estadoMotor == 0){
  /*motor desligado*/
  digitalWrite(pinEntrada1PonteH, LOW);
  digitalWrite(pinEntrada2PonteH, LOW);
  velocidadeMotorPWM = 0;/*zerar velocidade*/
  analogWrite(pinPWM, velocidadeMotorPWM);
 }
 if(estadoMotor == 1){
  /*Motor como ventilador*/
  digitalWrite(pinEntrada1PonteH, LOW);
  digitalWrite(pinEntrada2PonteH, HIGH);
  /*Deve receber parâmetro de velocidade do comando VEL XXX**/
  analogWrite(pinPWM, velocidadeMotorPWM);
 }
 if(estadoMotor == 2){
  /*Motor como exaustor*/
  digitalWrite(pinEntrada1PonteH, HIGH);
  digitalWrite(pinEntrada2PonteH, LOW);
  /*Deve receber parâmetro de velocidade do comando VEL XXX**/
  analogWrite(pinPWM, velocidadeMotorPWM);
 }
 if(estadoMotor == 3){
  /*Frear motor*/
  digitalWrite(pinEntrada1PonteH, HIGH);
  digitalWrite(pinEntrada2PonteH, HIGH);
  analogWrite(pinPWM, 0);
 }
}
```

```
void setup()
{
  pinMode(pinPWM, OUTPUT);
  pinMode(pinEncoder, INPUT);
  pinMode(pinEntrada1PonteH, OUTPUT);
  pinMode(pinEntrada2PonteH, OUTPUT);
  //digitalWrite(pinEntrada1PonteH, LOW);
  //digitalWrite(pinEntrada2PonteH, LOW);
  Serial.begin(9600);
  cli();
  configuracao_Timer0();
  sei();
  /*Habilita interrupcao do encoder*/
  attachInterrupt(digitalPinToInterrupt(pinEncoder), estimaVelocidade, RISING);
}

void loop()
{
  _delay_ms(1);
  LerComando();
  DecodComando();
  maqEstadosMotor();
  rotacaoMotor();
  convertPercentPWM();

  // Habilita a interrupcao do encoder em um tempo menor de 200ms
  if(counter < verificaVelocidade){
    attachInterrupt(digitalPinToInterrupt(pinEncoder), estimaVelocidade, RISING);
  }
  /* counter = 125 -> 1s */
  /* counter = 25 -> 0.2s */
```

```
        /* Durante 0.2s segundo incremento passagemDaPa */
        /* passageDaPa indica o número de voltas foram dadas em 0.2s */
        /* Multiplicar por 5 para estimar a rotação em 1s */
        if(counter >= verificaVelocidade){
          detachInterrupt(digitalPinToInterrupt(pinEncoder)); // Desabilita interrupcao para calcular a velocidade
          velocidadeEstimadaRPM = (60*1000/pulsosPorVolta)/(verificaVelocidade*8) * passagemDaPa;
          passagemDaPa = 0;
          attachInterrupt(digitalPinToInterrupt(pinEncoder), estimaVelocidade, RISING); // Habilita interrupcoes apos calculo da velocidade
          counter = 0;
          Serial.println(velocidadeEstimadaRPM);
        }
        Serial.println(estadoMotor);
        Serial.println(velocidadeEstimadaRPM);
      }
```