Faculdade de Engenharia da Universidade do Porto

Joint Motor Control

Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Robótica Industrial

Professor: António Moreira

Carlos Tiago Feijão Duarte Torres Almeida UP201303918
Sandro Augusto Costa Magalhães UP201304932
Telmo Miguel Silva Costa UP201305393

Maio de 2017

1 Introdução

Pretende-se desenvolver um sistema de controlo PID em cascata em malha fechada para um motor Joint.

Para tal começa-se por se analisar e modelar o sistema, para, iterativamente implementar as diferentes componentes do PID e analisar o seu comportamento, melhorando os parâmteros.

2 Análise do modelo à variação manual da tensão

Por aplicação de diferentes tensões no motor, pode-se de imediato inferir que este está limitado a uma tensão de $\pm 12V$.

3 Controlo proporcional do motor

O controlo proporcional do motor consiste na multiplicação de um ganho pelo erro, resultando, daqui, a tensão que vai ser aplicada ao motor.

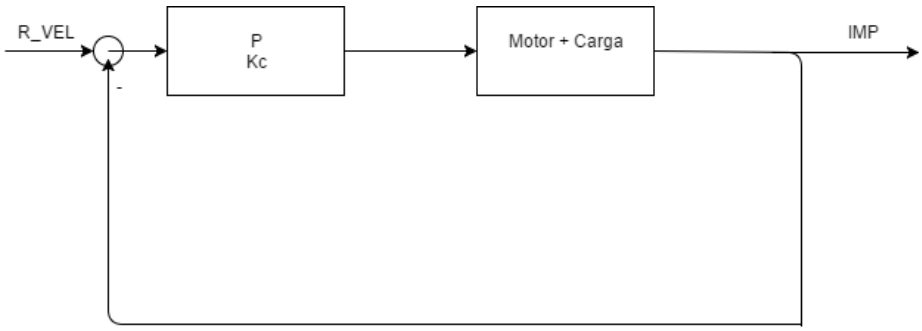

Figura 1: Modelo em malha fechada do controlo proporcional

```
// Global Variables
var
  Vnom: double;

procedure Control;
var Pulses: integer;
    Volts, Gain, Error: double;
begin
  Pulses := GetAxisOdo(0, 0);
  SetRCValue(2,2,format('%d',[Pulses]));
  Reference := GetRCValue (3,2);
  Gain := GetRCValue (4,2);

  Error := Pulses - Reference;

  // Manual Control
  Volts := Error * Gain;
  SetAxisVoltageRef(0, 0, Volts);
end;

procedure Initialize;
begin
  SetMotorActive(0, 0, true);
end;
```

Neste tipo de controlo, quando os ganhos são elevados, quanto maior o erro, maior é a tensão a ser aplicada ao motor. Portanto, quando o motor está muito perto do erro nulo, ele ainda está a sofrer inércia de tensões aplicadas anteriormente. Devido a este efeito, o motor nunca é capaz de recuperar e estabilizar em erro nulo, tornando o seu modo de funcionamento instável e oscilatório; sendo que este efeito aumenta conforme aumenta o ganho do motor.

Por outro lado, se aplicado ganhos pequenos, tipicamente inferiores a 1, o sistema nunca é capaz de atingir a referência.

Podemos assim deduzir que ganhos elevados permitem-nos atingir a referência, mas levam-nos para a instabilidade. Por outro lado, ganhos pequenos são mais estáveis, mas não nos permitem anular o erro. Conclui-se, por conseguinte, que um controlador do tipo proporcional não é suficiente para controlar o motor.

4 Modelo do motor mais carga

Com recurso ao código que simula o comportamento do motor em modo manual, por alteração da tensão do motor, pretende-se calcular o modelo aproximado do sistema a controlar que segue a forma da equação 1.

$$G(s) = \frac{k_p}{\tau s + 1} \tag{1}$$

Começou-se por induzir na entrada uma tensão de 2V que resulta numa velocidade de 8 impulsos/ciclo. De seguida, provoca-se uma tensão de 10V que resulta em 50 impulsos/ciclo. Por fim, por simplificação do sistema, assumiu-se que a recta que une os dois pontos é linear e, com recurso à equação 2, calculou-se o declive da recta, que corresponde ao ganho do sistema, que é $k_p = 5,25$.

$$k_p = \frac{v_{max} - v_{min}}{V_{max} - V_{min}} \tag{2}$$

Numa segunda fase, pretende-se obter a constante de tempo, τ, que modela o sistema motor mais carga em malha aberta.

Começa-se, assim, por registar o gráfico da transição de velocidade que corresponde à variação da tensão de 2 para 10 Volts, representado na figura 2.

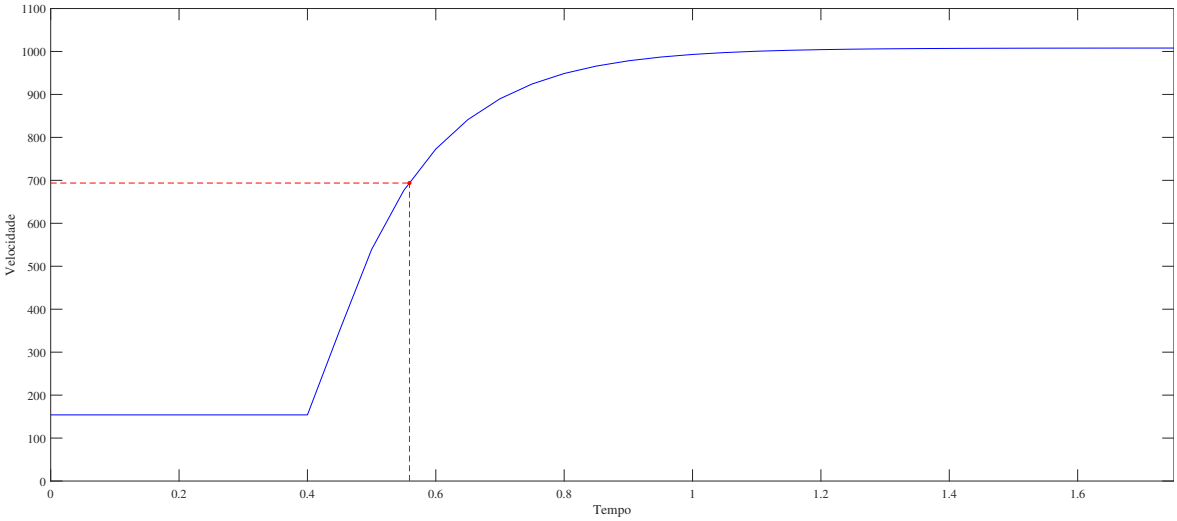

Figura 2: Gráfico da variação de velocidade de 2 para 10V.

Por análise do registo do gráfico, concluímos que há uma transição de velocidade que vai de 154 para $1007,98$, que corresponde a uma diferença de $853,98$. Considerando que a constante de tempo corresponde a $63,2\%$ da elevação, temos que esta corresponde ao intervalo de tempo que vai desde que começou a subida até ao instante em que é $539,72 + 154 = 693,72$.

Aplicando uma interpolação linear, equação 3, concluímos que o instante referido é 559ms. Subtraindo agora o instante inicial da subida, isto é, 400ms, concluímos que $\tau = 159$ms.

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0} \tag{3}$$

Concluímos, assim, que de acordo com o modelo específicado na equação 1, o modelo do nosso motor é

$$G(s) = \frac{5,25}{0,159s + 1} \tag{4}$$

5 Controlo proporcional e integral (PI) do motor

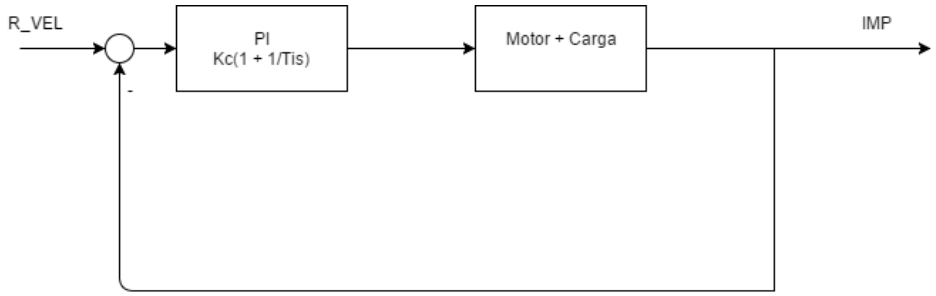

Figura 3: Malha fechada do controlador PD

De forma a calcular os parâmetros do controlador PI, recorreu-se ao método IMC-PID para duas constantes de tempo em malha fechada ($\tau_{CL} = 0,10$ e $\tau_{CL} = 0,15$).

Segundo este método, para o modelo motor mais carga utilizado, o valor de Ti é igual ao valor da constante de tempo, 159ms, enquanto o produto $K_c \cdot K_p$ é dado por $\frac{\tau}{\tau_{CL}}$. Assim, uma vez que $K_p = 5,25$ e $\tau = 159$ms, infere-se que os valores de K_c são 0,303 e 0,202, para $\tau_{CL} = 0,10$s e $\tau_{CL} = 0,15$s, respetivamente. Obtidos estes valores, é possível simular o sistema com o controlador PI implementado e inferir a respetiva constante de tempo.

Tal como no cálculo do modelo do motor + carga, observou-se a resposta ao degrau do sistema, verificando-se o tempo decorrido entre o início da subida e até ser atingido 63,2% desta. Assim, para $\tau_{CL} = 0,10$ s, verificou-se uma subida de velocidade de 801,8843 sendo que o valor da velocidade quando são atingidos 63,2% da elevação é de 704,6459. Recorrendo ao mesmo método usado para o cálculo do modelo do motor, temos que a constante de tempo do PI em malha fechada é $\tau_{CL} = 0,129$ s.

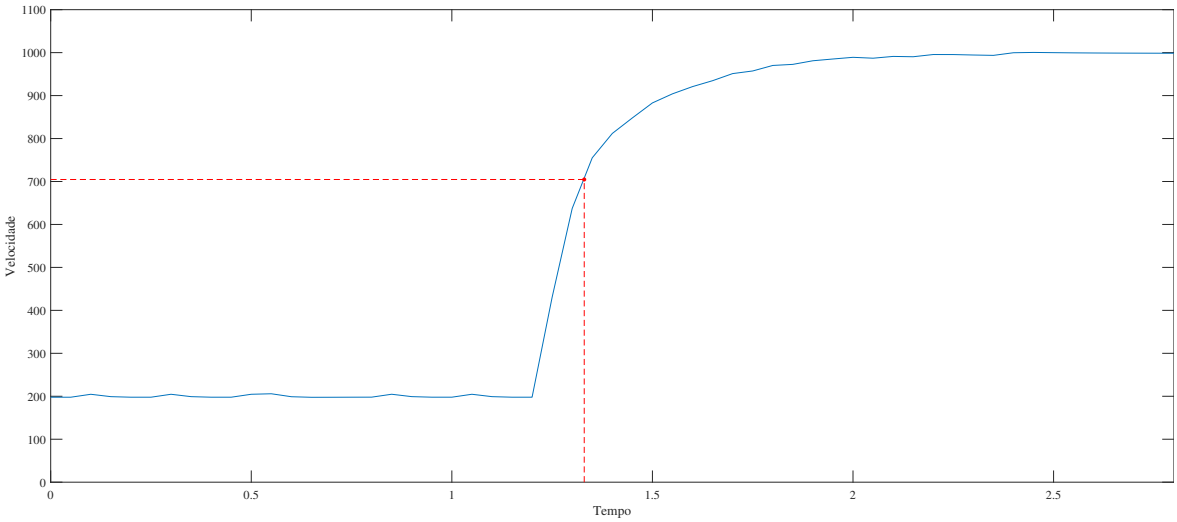

Figura 4: Gráfico da variação de velocidade de uma variação acentuada da referência.

Considerando $\tau_{CL} = 0,15$ s e aplicando o mesmo processo, verifica-se uma elevação de velocidade de 806,4422, atingindo-se 63,2% da elevação a uma velocidade de 705,3972. Temos então que $\tau_{CL} = 0,169$ s.

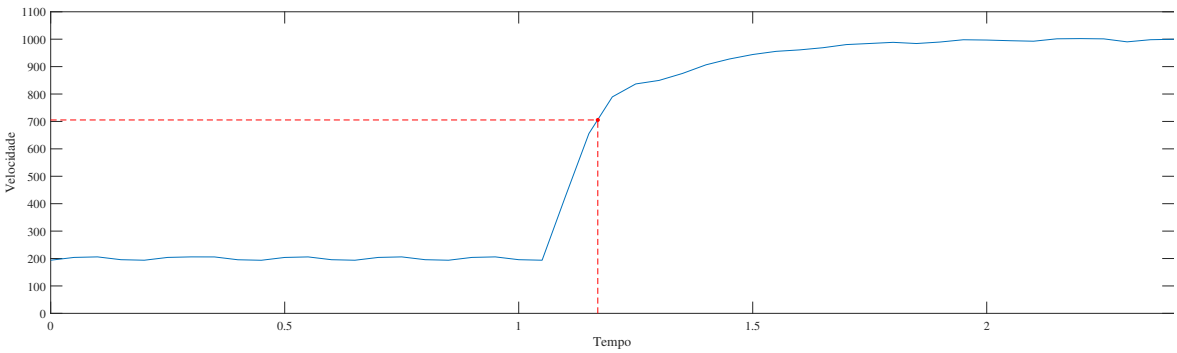

Figura 5: Gráfico da variação de velocidade de uma variação acentuada da referência.

Um dos critérios heurísticos tidos em conta para a selecção da constante de tempo em malha fechada é que $\tau_{CL} \geq 2 \times \tau_{motor}$. No entanto, devemo-nos afastar um pouco deste limite, devido às aproximações que fazemos ao processo, portanto, conclui-se que a melhor constante de tempo a utilizar é de $\tau_{CL} = 15$ms.

Uma alternativa para se poder utilizar $\tau_{CL} = 10$ms era reduzir o tempo de amostragem. No entanto, esta torna-se inviável devido ao modo de funcionamento dos *encoders*, que não são capazes de amostrar mais rápido.

```
// Global Variables
var
  Integral: double;
  Ti,Tc, Kc: double;

procedure Control;
var Pulses: integer;
    Volts, Reference, Error: double;

begin
  // Leitura
  Pulses := GetAxisOdo(0, 0);
  Reference := GetRCValue (3,2);

    // Processo
  Error := Reference - Pulses;
  Integral := Integral + Error*Tc;
  Volts := Kc *(Error + (1/Ti)*Integral);

  // Anti-Windup
  if (Volts > 12) then begin
    Volts := 12;
    Integral := Integral - Error*Tc;
  end;
  if (Volts < -12) then begin
    Volts := -12;
    Integral := Integral - Error*Tc;
  end;

  // Escrita
  SetAxisVoltageRef(0, 0, Volts);
  SetRCValue(2,2,format('%d',[Pulses]));
  SetRCValue (1,2,format('%g',[Volts]));
  SetRCValue (5,2, format('%g',[Error]))
end;

procedure Initialize;
begin
  SetMotorActive(0, 0, true);

  Integral := 0;
  Ti := 0.159;
  Tc := 0.05
  Kc := 0.202;
end;
```

6 Controlo proporcional e derivativo (PD) do motor

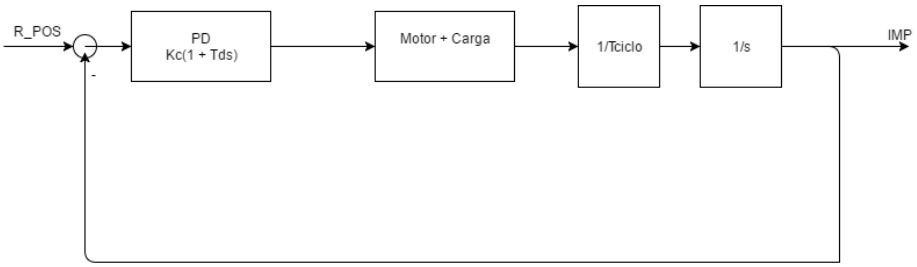

Figura 6: Malha fechada do controlador PD

Do modelo apresentado anteriormente facilmente se retira:

$$\frac{IMP(s)}{R_{POS}(s)} = \frac{G(s)}{1 + G(s) \cdot H(s)} \tag{5}$$

Dado que, neste caso, $H(s) = 1$ e $G(s) = \dfrac{K_c \cdot (1 + T_d \cdot s) \cdot K_p}{(\tau s + 1) \cdot T_c iclo \cdot s}$, tem-se:

$$\frac{G(s)}{(1 + G(s) \cdot H(s)} = \frac{K_c \cdot (1 + T_d \cdot s) \cdot K_p}{(\tau s + 1) \cdot T_{ciclo} \cdot s + K_c(1 + T_d \cdot s)K_p} = \frac{N(s)}{D(s)} \tag{6}$$

Simplificando o denominador, tem-se:

$$D(s) = s^2 + \left(\frac{1}{\tau} + \frac{K_c K_p T_d}{T_{ciclo} \cdot \tau}\right) \cdot s + \frac{K_p K_c}{T_{ciclo} \cdot \tau} \tag{7}$$

Por outro lado, sabe-se que :

$$\left(s - \frac{B_1}{T_{est}}\right)\left(s - \frac{\overline{B_1}}{T_{est}}\right) = s^2 + b_1 s + b_0 \tag{8}$$

E como para o protótipo de Bessel de segunda ordem temos $B_1 = -4,0530 + \mathrm{j}2,34$ e settling time (T_{est}) ajustado para 0,9s, é possível inferir os correspondentes valores dos coeficientes $b_1 = -9,00667$ e $b_0 = 27,040011$.

De seguida, igualando os coeficientes das equações anteriores e resolvendo obtém-se $K_c = 0,0409$ e $T_d = 0,1005$.

```
// Global Variables
var
  Integral_out, Error_ant: double;
  Ti,Tc, Kc, Td: double;

procedure Control;
var Pulses: integer;
    Volts, Reference, Error, Derivative, Pulses_second: double;

begin
  // Leitura
  Pulses := GetAxisOdo(0, 0);
  Reference := GetRCValue (3,2);

    // Controlador
  Pulses_second := Pulses / Tc;
  Integral_out := Integral_out + Pulses_second*Tc;
  Error := Reference - Integral_out;
  Derivative := (Error-Error_ant)/Tc;
  Volts := Kc *(Error + Td*Derivative);

  Error_ant := Error;

    // Anti-Windup
  if (Volts > 12) then begin
    Volts := 12;
  end;
```

```
28    if (Volts < -12) then begin
29      Volts := -12;
30    end;
31
32    // Escrita
33    SetAxisVoltageRef(0, 0, Volts);
34
35    SetRCValue(2,2,format('%d',[Pulses]));
36    SetRCValue (1,2,format('%g',[Volts]));
37    SetRCValue (5,2, format('%g',[Error]));
38    SetRCValue (7,2, format('%g',[Integral_out]));
39  end;
40
41  procedure Initialize;
42  begin
43    SetMotorActive(0, 0, true);
44
45    Integral_out := 0;
46    Error_ant := 0;
47
48    Ti := 0.159;
49    Tc := 0.05;
50    Td := 0.1005;
51    Kc := 0.0409;
52  end;
```

7 Controlo proporcional, integral e derivativo (PID) em cascata do motor

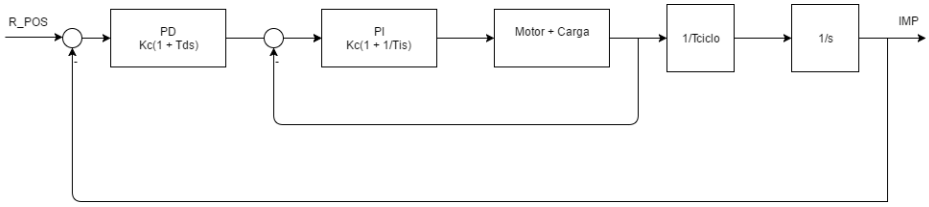

Figura 7: Malha fechada do controlador PD

Nesta fase do trabalho procurou-se integrar as soluções de controlo de velocidade e posição num único controlador em cascata. De forma a obter os parâmetros do controlador proporcional-derivativo basta aplicar o método descrito no ponto anterior, recorrendo às expressões 6 e 7, tendo em consideração que utilizamos a constante de tempo de malha fechada do controlador PI aplicado, isto é, $\tau_{CL} = 0,169$ s.

Deste modo, obtêm-se os valores de K_c e T_d, com $K_c = 0,04325$ e $T_d = 0,11425$.

```
1   // Global Variables
2   var
3     Integral_out, Integral_PI, Error_ant: double;
4     Ti,Tc, Kc, Td, Kc_PI: double;
5
6   procedure Control;
7   var Pulses: integer;
8       Volts, Reference, Error, Error_PI, Derivative, Pulses_second: double;
9
10  begin
11    // Leitura
12    Pulses := GetAxisOdo(0, 0);
13    Reference := GetRCValue (3,2);
14
15      // Controlador PD
16    Pulses_second := Pulses / Tc;
17    Integral_out := Integral_out + Pulses_second*Tc;
18    Error := Reference - Integral_out;
19    Derivative := (Error-Error_ant)/Tc;
20    Volts := Kc *(Error + Td*Derivative);
```

```
21
22    Error_ant := Error;
23
24      // Controlador PI
25    Error_PI := Volts - Pulses;
26    Integral_PI := Integral_PI + Error_PI*Tc;
27    Volts := Kc_PI *(Error_PI + (1/Ti)*Integral_PI);
28
29      // Anti-Windup
30    if (Volts > 12) then begin
31      Volts := 12;
32    end;
33    if (Volts < -12) then begin
34      Volts := -12;
35    end;
36
37
38    // Escrita
39    SetAxisVoltageRef(0, 0, Volts);
40
41    SetRCValue(2,2,format('%d',[Pulses]));
42    SetRCValue (1,2,format('%g',[Volts]));
43    SetRCValue (5,2, format('%g',[Error]));
44    SetRCValue (7,2, format('%g',[Integral_out]));
45  end;
46
47  procedure Initialize;
48  begin
49    SetMotorActive(0, 0, true);
50
51    Integral_out := 0;
52    Error_ant := 0;
53    Integral_PI := 0;
54
55    Ti := 0.159;
56    Tc := 0.05;
57    Td := 0.11425;
58    Kc := 0.04352;
59    Kc_PI := 0.202;
60  end;
```

8 Implementação do código num sistema real

8.1 Código controlador PI

```
1     /* Inicio do controlador */
2
3     Erro =  RefVel - vel_odo1;
4     Ierro = Ierro + Erro*Tc;
5     voltage_M1 = Kc * (Erro + (1/Ti)*Ierro);
6
7     // Anti-Windup
8     if (voltage_M1 > 12){
9       voltage_M1 = 12;
10      Ierro = Ierro - Erro*Tc;
11    }
12    if (voltage_M1 < -12){
13      voltage_M1 = -12;
14      Ierro = Ierro - Erro*Tc;
15    }
16
17    // Controller code - available variables:
18    // vel_odo1 - impulses per cicle
19    // encoder1_pos - position counter
20
21    voltage_M1 = voltage_M1*(1023/12);
22
23    if (voltage_M1 > 1023){
24      voltage_M1 = 1023;
25    }
```

```
    if (voltage_M1 < -1023){
      voltage_M1 = -1023;
    }

    set_M1_voltage(voltage_M1);
```

8.2 Código controlador PD

```
    /* Inicio do controlador */

    Iout = Iout+vel_odo1;
    Erro = RefPos-Iout;
    derivada = (Erro - Erro_ant)/Tc;
    voltage_M1 = Kc * (Erro + derivada*Td);

    Erro_ant = Erro;

    // Controller code - available variables:
    // vel_odo1 - impulses per cicle
    // encoder1_pos - position counter

    voltage_M1 = voltage_M1*(1023/12);

    if (voltage_M1 > 1023){
      voltage_M1 = 1023;
    }
    if (voltage_M1 < -1023){
      voltage_M1 = -1023;
    }

    set_M1_voltage(voltage_M1);
```

8.3 Código controlador PID

```
   /* Inicio do controlador */

    Iout = Iout+vel_odo1;
    Erro = RefPos-Iout;
    derivada = (Erro - Erro_ant)/Tc;
    RefVel = Kc * (Erro + derivada*Td);

    Erro_ant = Erro;

    Erro =  RefVel - vel_odo1;
    Ierro = Ierro + Erro*Tc;
    voltage_M1 = Kc_PI * (Erro + (1/Ti)*Ierro);

    // Anti-Windup
    if (voltage_M1 > 12){
      voltage_M1 = 12;
      Ierro = Ierro - Erro*Tc;
    }
    if (voltage_M1 < -12){
      voltage_M1 = -12;
      Ierro = Ierro - Erro*Tc;
    }

    // Controller code - available variables:
    // vel_odo1 - impulses per cicle
    // encoder1_pos - position counter

    voltage_M1 = voltage_M1*(1023/12);

     if (voltage_M1 > 1023){
      voltage_M1 = 1023;
    }
    if (voltage_M1 < -1023){
      voltage_M1 = -1023;
    }
```

```
36
37      set_M1_voltage(voltage_M1);
38
39    }
```