11 實例探究

閱讀完本章後，讀者將：

1. 在機電一體化系統的設計和實時實施方面實驗他的知識。
2. 進行從A到Z的機電系統設計。
3. 能夠執行機電一體化系統設計的不同階段。
4. 能夠解決控制問題並建立我們必須實時執行的控制規則。

11.1 介紹

在前面的章節中，我們提出了一些概念，並通過學術示例說明了它們的應用，以向讀者展示這些結果如何應用。更具體地說，我們已經了解瞭如何設計機電一體化系統。介紹了成功完成所需機電系統設計所必須遵循的不同步驟。我們已經介紹了在以下設計中必須使用的方法：

- 機械部分
- 電子電路
- C 語言中用於實時執行的程序

這些工具已應用於一些實際系統，並提供了更多詳細信息以幫助讀者執行自己的設計。

對於控制算法，我們提供的大多數示例都是具有理想模型的學術模型。不幸的是，對於一個實際的系統，我們將擁有的模型只是一個可以在某些特定條件下描述該系統的實現，並且由於某些原因，該模型無法在算法的實時實現過程中按預期方式完美運行。這可能是由於不同的動態動力學可能導致某些頻率的行為改變而引起的。

本章的目的是向讀者展示我們如何實時實現前面幾章中針對實際系統開發的理論結果。我們將逐步進行並顯示所有步驟，以使讀者輕鬆閱讀。我們在本章中考慮的案例研究是在前幾章中討論和設計的案例研究。

11.2 速度 直流電機套件的控制

作為第一個例子，讓我們考慮驅動機械零件的直流電動機的速度控制。該示例的選擇非常重要，因為大多數係統將使用此類直流電動機。我們將考慮的直流電動機由 Maxon 公司製造。該電動機非常重要，因為它帶有齒輪箱（比率 6：1）和編碼器，該編碼器每轉給出一百個脈衝，我們每轉產生 600 個脈衝，我們通過使用正交方法將其變為 24400 個脈衝。每轉。如果在本示例中使用的系統

具有更靈活和更多的優勢，則可以在前面介紹的控制算法的實時實現中使用。

該電動機的數據表給出了所有重要參數，因此容易獲得該執行器的傳遞函數。在此示例中，我們正在考慮的負載是一個帶有刻度的小磁盤，我們希望在速度上控制它，然後在位置上進行控制。這種設置在**圖 11.1** 和 11.2 中示出。我們正在考慮的圓盤的直徑等於 0.06 m、質量等於 0.050 Kg。利用這些數據和直流電動機的數據表之一，我們可以獲得磁盤速度和輸入電壓之間的傳遞函數。

首先讓我們專注於負載的速度控制。在這種情況下，為了建立該系統的傳遞函數（直流電動機執行器及其負載），我們可以使用數據表，磁盤上的信息以及 Boukas[1]中的結果。

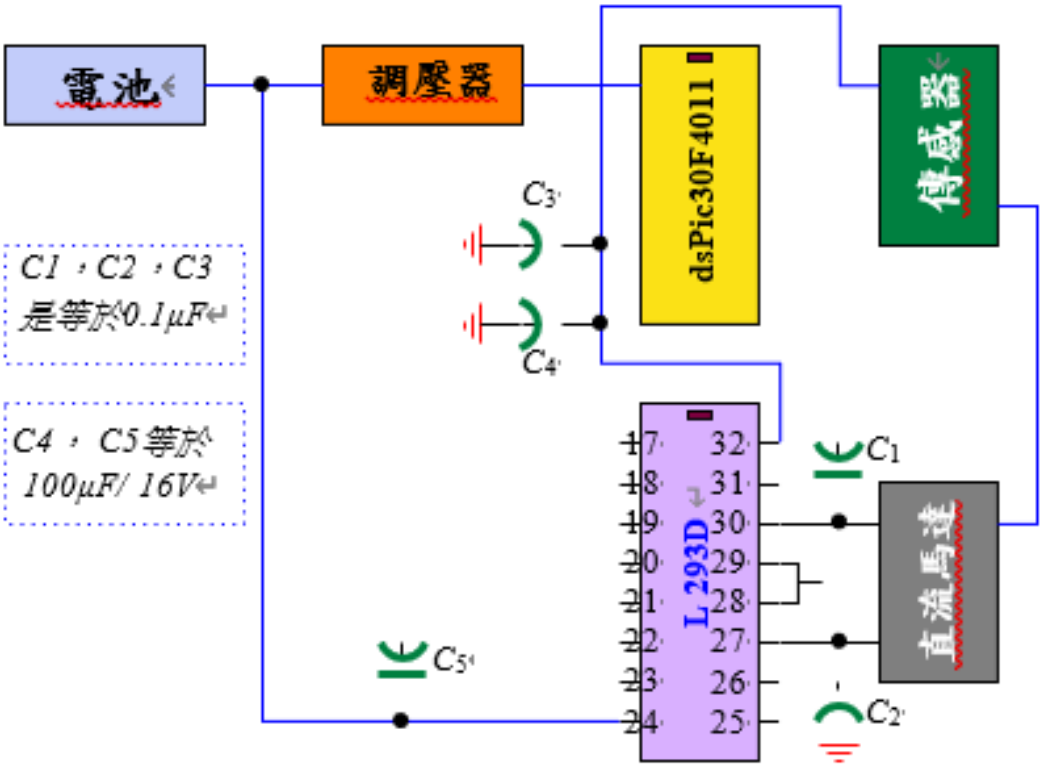

圖 11.1 直流電動機套件的電子電路

或繼續進行識別。使用第一種方法，[1]中第 2 章的結果是:

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

和

$$k = 48.91$$

$$\tau = 63.921 \text{ 毫秒}$$

為了設計控制器，我們首先應該指定我們希望系統具有的性能。首先，我們需要係統穩定，還需要係統速度在過渡狀態下具有良好的性能，在穩態狀態下具有零誤差，以作為階躍參考。對於瞬態，我們希望負載的穩定時間小於或等

於 3τ/5 的 5%，而過衝則小於或等於 5%。

為了完成適當控制器的設計，我們既可以連續進行設計，然後獲得應在軟件部分進行編程的算法，也可以直接在離散時間內進行所有設計。在本示例的其餘部分中，我們將選擇第二種方法。

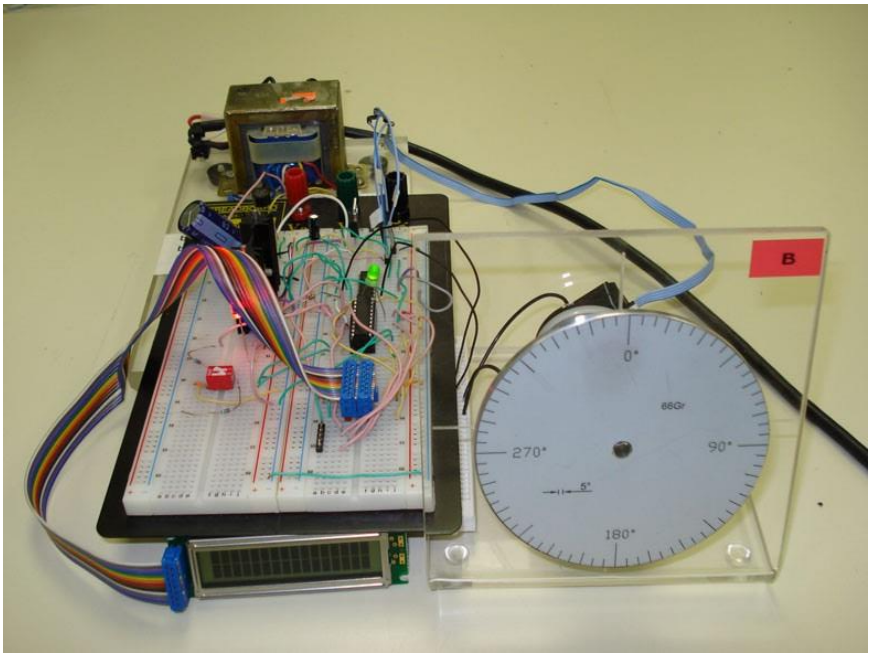

圖 11.2 實時實施設置

根據系統傳遞函數的表達式以及所需的性能，結果表明我們至少需要一個比例積分器（PI）控制器。該控制器的傳遞函數由下式給出：

$$C(s) = K_P + \frac{K_I}{s}$$

其中 KP 和 KI 是要確定的結果，以迫使負載具有我們施加的性能。

使用零序持有者和 Z-transform 表，我們得到：

$$\begin{aligned} G(z) &= \frac{Kz\left(1-e^{-\frac{T}{\tau}}\right)\left(1-z^{-1}\right)}{(z-1)\left(z-e^{-\frac{T}{\tau}}\right)} \\ &= \frac{K\left(1-e^{-\frac{T}{\tau}}\right)}{z-e^{-\frac{T}{\tau}}} \end{aligned}$$

對於控制器，使用梯形離散化，我們得到：

$$\begin{aligned} C(z) &= \frac{U(z)}{E(z)} = K_P + K_I\frac{T}{2}\frac{z+1}{z-1} \\ &= \frac{\left(K_P+\frac{TK_I}{2}\right)z+\left(-K_P+\frac{TK_I}{2}\right)}{z-1} \end{aligned}$$

將分子和分母除以 z 並回到時間，我們得到：

$$u(k) = u(k-1) + \left(K_P + \frac{TK_I}{2}\right)e(k) + \left(-K_P + \frac{TK_I}{2}\right)e(k-1)$$

結合執行器及其負載和控制器之一的傳遞函數，我們得到以下閉環傳遞函數：

$$F(z) = \frac{K\left(1-e^{-\frac{T}{\tau}}\right)\left(K_P + \frac{TK_I}{2}\right)z + K\left(1-e^{-\frac{T}{\tau}}\right)\left(-K_P + \frac{TK_I}{2}\right)}{z^2 + \left(K\left(K_P + \frac{TK_I}{2}\right)\left(1-e^{-\frac{T}{\tau}}\right) - 1 - e^{-\frac{T}{\tau}}\right)z + K\left(1-e^{-\frac{T}{\tau}}\right)\left(-K_P + \frac{TK_I}{2}\right) + e^{-\frac{T}{\tau}}}$$

現在使用期望的性能，很容易得出結論，主導根點是:

$$s_{1,2} = -\zeta\omega_n \pm jw_n\sqrt{1-\zeta^2}$$

其中 ζ 和 ωn 分別代表我們系統控制閉環的阻尼比和固有頻率。

根據控制理論（請參見 Boukas [1]），眾所周知，過衝 d%和 5%的建立時間 ts 由下式給出：

$$d\% = 100e^{\frac{-\pi\zeta}{\sqrt{1-\zeta^2}}}$$
$$t_s = \frac{3}{\zeta\omega_n}$$

使用我們的性能和這些表示，我們得出以下結論:

$$\zeta = 0.707$$
$$\omega_n = \frac{5}{\tau\zeta} = 110.6387 rad/s$$

它具有以下主要優勢：

$$s_{1,2} = -78.2216 \pm 78.2452j$$

使用 z = e^{Ts} 且 T = τ /10= 0.0064 的變換，我們得到以下結果 $\mathcal{Z}$ 域中的主要根點：

$$z_{1,2} = 0.5317 \pm 0.2910j$$

對於這些根點，我們具有以下特徵方程式：

$$\Delta_d = (z - 0.5317 - 0.2910j)(z - 0.5317 + 0.2910j)$$
$$= z^2 - 1.0634z + 0.3674$$

現在使用根點放置技術，我們得到：

$$1 + C(z)G(z) = \Delta_d$$

這意味著：

$$K_I = \frac{0.3040}{KT\left(1 - e^{-\frac{T}{\tau}}\right)}$$

$$K_P = \frac{-0.4308 + 2e^{-\frac{T}{\tau}}}{2K\left(1 - e^{-\frac{T}{\tau}}\right)}$$

使用 K，T 和 τ 的值，我們得到增益 KP 和 KI 的以下表達式：

$$K_P = 0.1480$$
$$K_I = 10.1951$$

備註 11.2.1 在這種情況下，必須謹慎，因為我們不在乎傳遞函數零的位置，因此在實現此控制器時可能會有些意外。顯然，我們將獲得的性能（穩定時間和過衝）將取決於零的位置。有關此問題的更多詳細信息，請向讀者介紹 Boukas[1]。

至現在實施該 PI 控制算法，並確保達到理想的性能，我們將使用 Microshipl 的微控制器。這種選擇是由於我們在此類微控制器方面的經驗所致。讀者可以記住，其他製造商的任何其他微控制器都可以做些小改動。在本示例中，我們將使用 Microhip 的單片機 dsPIC30F4011。

我們實現的代碼使用 C 語言編寫。採用這種語言是因為其簡單性。該實現具有以下結構：

```
          //
          // Put here the include
          //

          #include "p30F4011.h" // proc specific header

//
// Define a struct
//
  typedef struct {
        // PI Gains
        float K_P; // Propotional gain
        float K_I; // Integral gain
```

```
      //
      // PI Constants
      //
      float Const1_pid; // KP + T KI/2
      float Const2_pid; // -KP + T KI/2
      float Reference; // speed reference

      //
      // System variables
      //
      float y_k; // y_m[k] -> measured output at time k
      float u_k; // u[k] -> output at time k
      float e_k; // e[k] -> error at time k

      //
      // System past variables
      //
      float u_prec; // u[k-1] -> output at time k-1
      float e_prec; // e[k-1] -> error at time k-1

    }PIStruct;

    PIStruct thePI;

    thePI.Const1= thePI.K_P+T*thePI.K_I/2;
    thePI.Const2=-thePI.K_P+T*thePI.K_I/2;
    thePI.Reference=600;

//
// Functions
//
float ReadSpeed(void);

float ComputeControl(void);

float SendControl(void);

//
```

```
// Interrupt program here using Timer 1 (overflow of counter Timer 1)
//
void __ISR _T1Interrupt(void) // interrupt routine code
    {
        // Interrupt Service Routine code goes here
        float Position_error;

        //
        // Read speed
        //
        thePI.y_m=ReadSpeed();

        thePI.e_k= thePI.Reference-thePI.y_m;

        //
        // Compute the control
        //
        ComputeContrl();

        //
        // Send control
        //
        SendControl();

        IFS0bits.T1IF=0; // Disable the interrupt
    }
int main ( void ) // start of main application code
{
// Application code goes here
int i;

// Initialize the variables Reference and ThePID.y_m (it can be read
from inputs) Reference = 0x8000; // Hexadecimal number
(0b... Binary number) ThePID = 0x8000;

// Initialize the registers
TRISC=0x9fff; // RC13 and RC14 (pins 15 and 16) are configured as
outputs
```

```
IEC0bits.T1IE=1; // Enable the interrupt on Timer 1

// Indefinite loop
while (1)
{
}
return 0
}

        % ReadSpeed function
        int ReadSpeed (void)
        {
        }

% ComputeControl function
int ComputeControl (void)
{
thePI.u_k=thePI.u_prec+thePI.Const1*thePI.e_k+thePI.Const2*thePI.e_pr
ec;
}

% SendControl function
int Send Control (void)
{
sendControl()

//
// Update past data
//
thePI.u_prec=thePI.u_k;
ThePI.e_prec=thePI.e_k;
}
```

從該結構可以看出，首先我們注意到系統將進入循環，並在每次中斷時調用函數：

- ReadSpeed;
- ComputeControl;
- SendControl;

並採取適當的措施。

ReadSpeed 函數在每個採樣時間返回加載速度，該速度將由 ComputeControl 函數使用。SendControl 功能通過 L293D 芯片將適當的電壓發送到執行器。

使用編譯器 HighTec C 獲取十六進制代碼，並使用 PicKit-2 將文件上傳到微控制器的內存中。 有關如何獲取十六進制代碼的更多詳細信息，我們邀請讀者閱讀編譯器 HighTec C 或 Microchip 的編譯器 C30 的手冊。

在這種情況下，國家方法是微不足道的，我們將不發展它。

11.3 直流電機套件的位置控制

讓我們專注於負載位置控制。遵循與上一節中開發的負載速度控制類似的步驟，我們首先需要選擇我們希望系統具有的理想性能。進行以下展示：

- 系統穩定在閉環狀態；
- 建立時間 ts 為 2%等於我們可以擁有的最佳時間
- 超調等於 5%
- 階躍函數作為輸入的穩態等於零

使用性能和傳遞函數，很容易得出結論，比例控制器 KP 足以滿足這些性能。

在此示例中，我們將使用連續時間方法進行控制器的設計。在上一章的基礎上，我們的系統模型如下：

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

其中 K 和 τ 取與速度控制相同的值。

讓傳輸控制器由以下方式給出：

$$C(s) = K_P$$

使用這些表達式，閉環傳遞函數由下式給出：

$$\begin{aligned} F(s) &= \frac{C(s)G(s)}{1 + C(s)G(s)} \\ &= \frac{\frac{KK_P}{\tau}}{s^2 + \frac{1}{\tau}s + \frac{KK_P}{\tau}} \end{aligned}$$

由於系統類型為 1，因此使用比例控制器輸入的階躍函數的誤差等於零。

根據規範，以下複雜的兩根：

$$s_{1,2} = -\zeta w_n \pm j w_n \sqrt{1 - \zeta^2}$$

將完成這項工作，相應的特徵方程式如下：

$$s^2 + 2\zeta w_n s + w_n^2 = 0.$$

與閉環系統之一等效，我們得到：

$$\begin{aligned} 2\zeta w_n &= \frac{1}{\tau} \\ w_n^2 &= \frac{KK_P}{\tau}. \end{aligned}$$

為了確定最佳穩定時間 ts 為 2%，請注意，我們有：

$$t_s = \frac{4}{\zeta w_n}.$$

現在使用事實：

$$\zeta w_n = \frac{1}{2\tau}$$

我們獲得：

$$t_s = 8\tau$$

因此，使用此控制器可以在 2%處獲得的最佳建立時間是系統恆定時間的 8 倍。小於可獲得的任何值。實際上，如果我們在改變 KP 時關注閉環系統的根源，這是微不足道的。這由**圖 11.3** 給出。為了將控制器的增益固定為所需的根點 s1,2 = 7.5 j，我們使用該圖並選擇一個 ζ = 0.707。得出 KP = 0.1471。

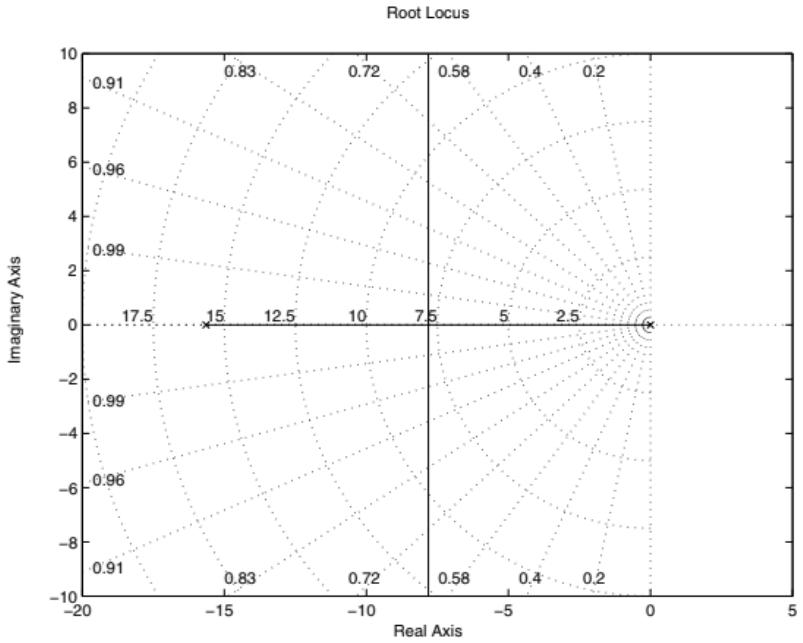

圖 11.3 帶比例控制器的直流電動機的根軌跡

使用該控制器，幅度等於 30 度的階躍函數的時間響應由**圖 11.4** 表示，從中我們可以得出結論，所設計的控制器以 2%的穩定時間等於 0.5115 s 滿足了所有期望的性能。但是，如果我們實現該控制器，則實際情況將與仿真有所不同，因為齒輪箱的齒隙並未包含在所使用的模型中，因此實時結果將有所不同，誤差永遠不會為零。為了克服這個問題，我們可以使用比例和微分控制器，它可以在 2%處提供更好的建立時間。讓該控制器的傳遞函數由下式給出：

$$C(s) = K_P + K_D s$$

其中 KP 和 KD 是要確定的增益。

備註 11.3.1 重要的是要注意，比例和微分控制器的使用將在閉環傳遞中引入零，如果放置得當，則可以縮短建立時間。根據其位置，過沖和建立時間將是受到影響。有關此問題的更多詳細信息，請參閱[1]。

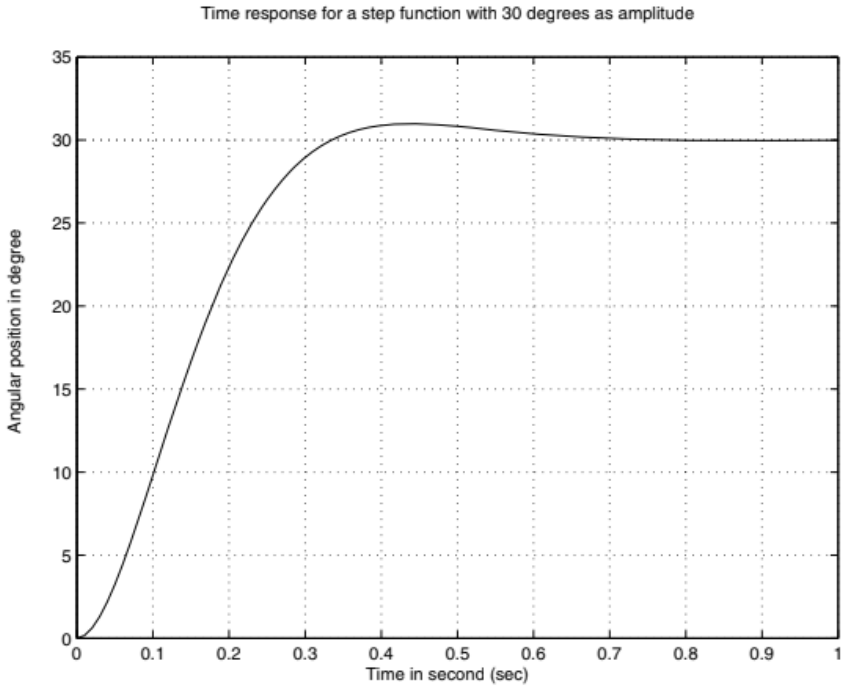

圖 11.4 幅度為 30 度的階躍函數的時間響應

使用該控制器，閉環傳遞函數由下式給出:

$$G(s) = \frac{C(s)G(s)}{1 + C(s)G(s)} = \frac{\frac{KK_D s + KK_P}{\tau}}{s^2 + \frac{1+KK_D}{\tau}s + \frac{KK_P}{\tau}}$$

和以前一樣，控制器的設計使用了兩個複雜的根點。如果將兩個特徵方程式

相等，則得到：

$$2\zeta w_n = \frac{1 + KK_D}{\tau}$$
$$w_n^2 = \frac{KK_P}{\tau}.$$

在這種情況下，我們有兩個未知變量 KP 和 KD 以及兩個唯一確定增益的代數方程。它們的表達式如下：

$$K_P = \frac{\tau w_n^2}{K}$$
$$K_D = \frac{2\tau\zeta w_n}{K}$$

現在使用期望的性能，我們得出與之前類似的結論，即等於等於 30 度幅值階躍函數的輸入的穩態誤差等於零，並且與過衝等於 5%的阻尼比 ζ 等於 0.707。我們可以將其固定為系統時間常數的一部分的穩定時間 ts 在 2%處給出：

$$w_n = \frac{4}{\zeta t_s}.$$

現在，如果將穩定時間固定為 3τ，我們將得到：

$$w_n = 29.4985.$$

使用這些值，我們可以得到以下控制器增益值：

$$K_P = 1.1374$$
$$K_D = 0.0545.$$

它給出了以下複雜的兩根：

$$s_{1,2} = -28.6763 \pm 6.9163j.$$

零為：

$$z = -20.8618.$$

使用該控制器，幅度等於 30 度的輸入的時間響應如**圖 11.5** 所示。從該圖可以看出，過沖和建立時間少於使用比例控制器獲得的過沖和建立時間。

至實施比例或比例和微分控制器，我們需要得到控制律的遞推方程。為此，我們需要使用前面介紹的不同方法來離散化控制器的傳遞函數。讓我們使用梯形方法，該方法包括將 s 替換為 $\frac{2}{T}\frac{z-1}{z+1}$。這給出：

$$C(z) = K_p \text{ for the proportional controller}$$

$$C(z) = K_P + K_D \frac{2}{T}\frac{z-1}{z+1} \text{ for the proportional and derivative controller}$$

如果我們通過控制分別表示 u（k）和 e（k）以及瞬時 kT 處參考與輸出之間的誤差，我們將得到以下表達式：

1. 對於比例

$$u(k) = K_P e(k)$$

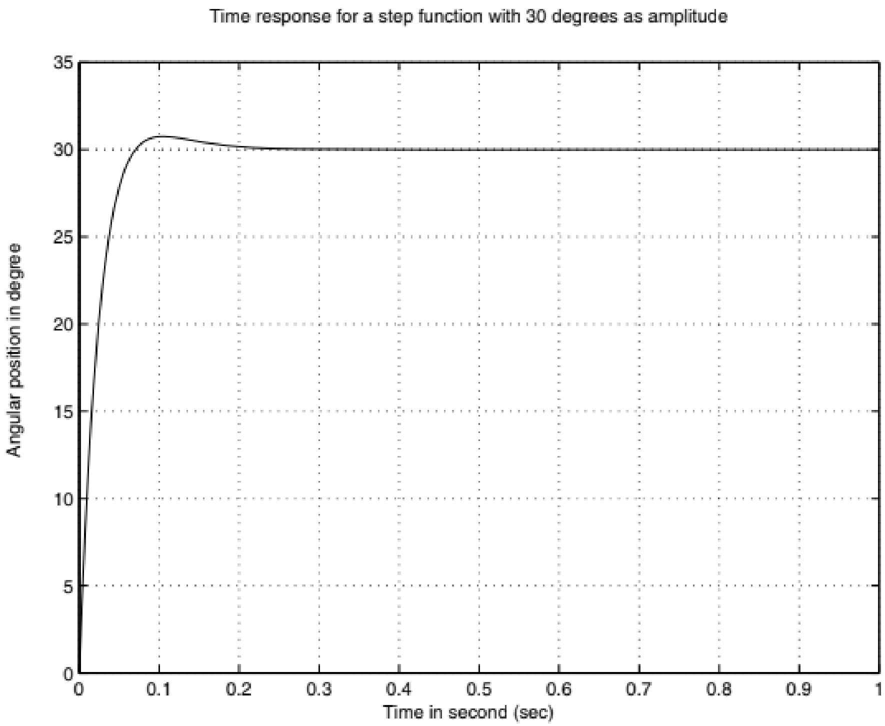

圖 11.5 幅度為 30 度的階躍函數的時間響應

2. 用於比例和微分控制器

$$u(k) = -u(k-1) + \left(K_P + \frac{2K_D}{T}\right)e(k) + \left(K_P - \frac{2K_D}{T}\right)e(k-1)$$

實現是該控制器使用相同的功能，但有一些小的更改。列出相應的功能是：

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main program %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


main
% Data
```

```
% Variables

% While loop
  while (1)
    do
      ReadSpeed;
      ComputeControl;
      SendControl;
    end;
% ReadSpeed function

% ComputeControl function

% SendControl function
```

現在讓我們使用此示例的狀態空間表示，並設計一個狀態反饋控制器，以保證所需的性能。對於這種情況，我們將首先假定完全訪問狀態，其次通過假設僅可以訪問該職位來放寬此假設。像我們之前所做的那樣，我們可以連續進行，也可以不連續進行。

先前我們建立了該系統的狀態空間描述，其描述為：

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{1}{\tau} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{K}{\tau} \end{bmatrix} u(t)$$

where $x(t) \in \mathbb{R}^2$ ($x_1(t) = \theta(t)$ and $x_2(t) = \dot{\theta}(t)$), and $u(t) \in \mathbb{R}$ (the applied voltage).

From the desired performances with a settling time at %2 equal to 3τ, we get the same dominant poles as before, and therefore the same characteristic equation, $\Delta_d(s) = s^2 + 2\zeta w_n s + w_n^2 = 0$ (with $\zeta = 0.707$ and $w_n = \frac{4}{3\zeta\tau}$). Using the controller expression the closed-loop characteristic equations given by:

$$\det(s\mathbb{I} - A + BK) = 0$$

通過均衡這兩個方程式，我們可以得到以下結果：

$$K_1 = 1.1146$$
$$K_2 = 0.0326$$

使用該控制器，幅度等於 30 度的輸入的時間響應如圖 11.6 所示。從該圖可以看出，超調和建立時間就是我們想要的。重要的是要注意穩態狀態下錯誤的存在。如果在循環中添加一個積分動作，則可以消除此錯誤。有關此的更多詳

細信息，請向讀者介紹[1]。

對於第二種情況，由於我們無法訪問負載速度，因此我們可以從位置計算負載速度，也可以使用觀察者來估計系統狀態。如前所述，我們用於觀察者設計的根點應該比控制器設計中使用的根點更快。

擇以下根點（s1,2，為控制器設計中所用實數的四倍）：

$$\begin{aligned} s_{1,2} &= -4\zeta w_n \pm j w_n \sqrt{1-\zeta^2} \\ &= -83.4218 \pm 20.8618 j \end{aligned}$$

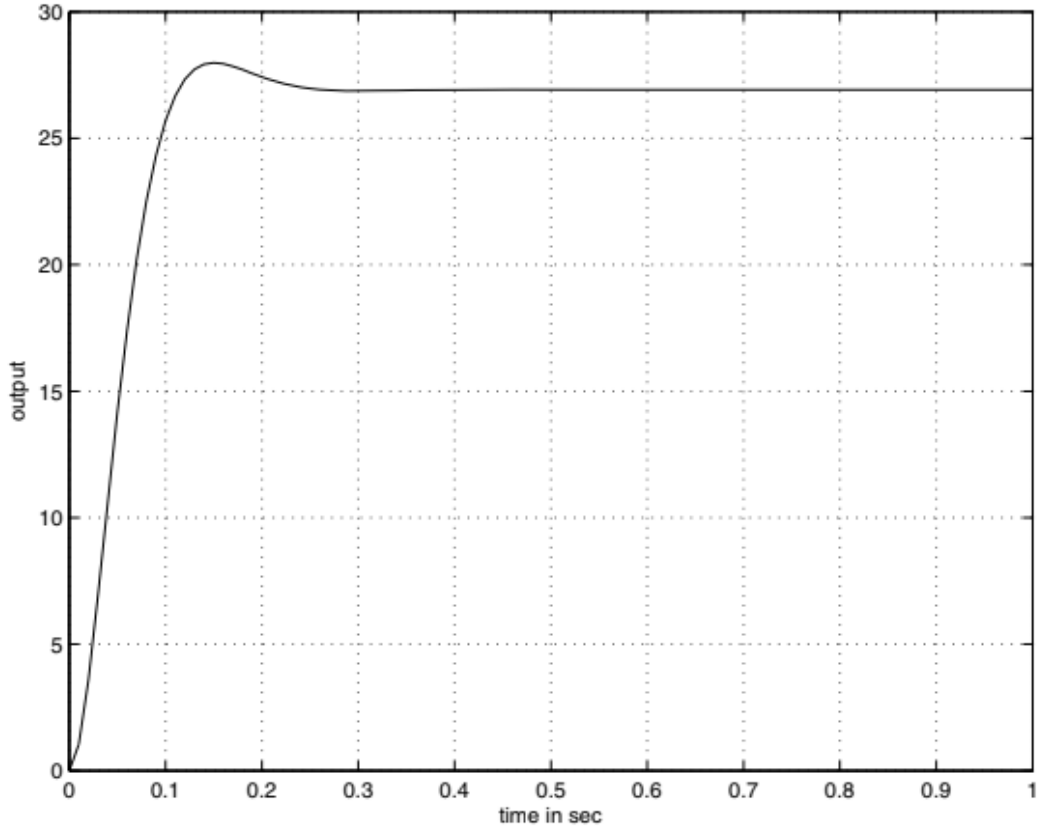

圖 11.6 幅度為 30 度的階躍函數的時間響應

我們為觀察者帶來以下結果：

$$\begin{aligned} L_1 &= 151.2 \\ L_2 &= 5029.4 \end{aligned}$$

控制器的增益與對狀態向量的完全訪問情況相同。

在下面的 Matlab 中，我們同時提供控制器和觀察者的設計，並進行仿真，以顯示系統和觀察者的狀態相對於時間的行為。

```
clear all
```

```
%data
tau=0.064
k=48.9
A = [0 1;0 -1/tau];
B = [0 ; k/tau];
C = [1 0];
D = 0;

% controller design
K = acker(A,B,[-3+3*j -3-3*j]);
L = acker(A' ,C' ,[-12+3*j -12-3*j])' ;
% Simualation data
Ts = 0.01;
x0 = [1 ; 1];
z0 = [1.1 ; 0.9];
Tf = 2; %final time

%augmented system
Ah = [A -B*K;
L*C A-B*K-L*C];
Bh = zeros(size(Ah,1),1);
Ch = [C D*K];
Dh = zeros(size(Ch,1),1);
xh0 = [x0 ; z0];
t=0:Ts:Tf;
u = zeros(size(t));
m = ss(Ah,Bh,Ch,Dh);

%simulation
[y,t,x] = lsim(m,u,t,xh0);

%plotting
figure;
plot(t,y);
title(' Output' );
xlabel(' Time in sec' )
ylabel(' Output' )
grid
```

```
figure;
plot(t,x(:,1:size(A,1)));
title(' States of the system' );
xlabel(' Time in sec' )
ylabel(' System states' )
grid

figure;
plot(t,x(:,size(A,1)+1:end));
title(' states of the observer' );
xlabel(' Time in sec' )
ylabel(' Observer states' )
grid
```

圖 11.7 與 11.9 給出了輸出、系統狀態和觀察者狀態的說明。

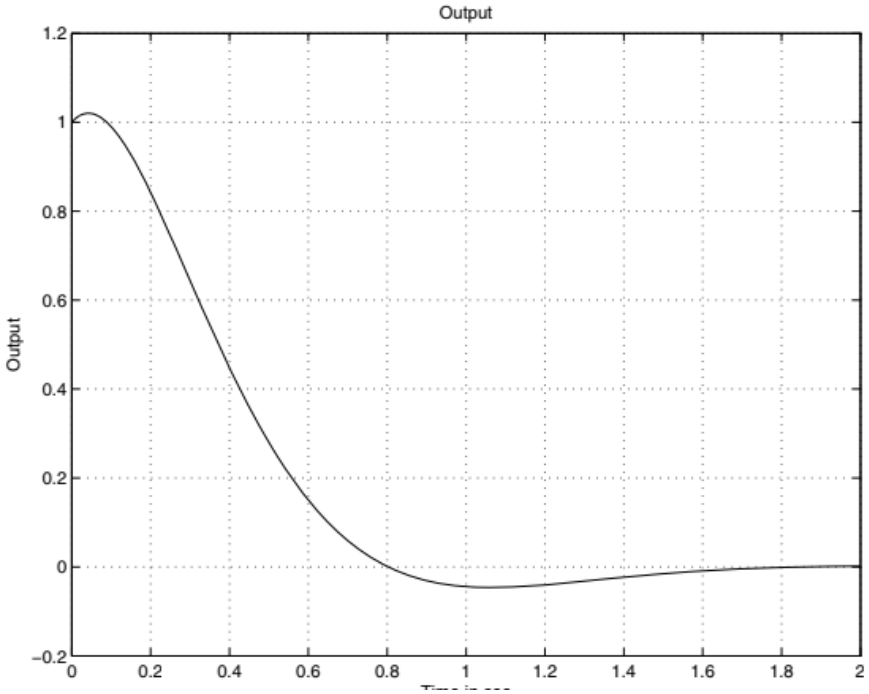

圖 11.7 輸出與時間

我們還可以使用線性二次調節器設計狀態反饋控制器。實際上，如果我們為成本函數選擇以下矩陣：

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$$

$$R = 10$$

備註 11.3.2 通常，對於成本函數的矩陣的選擇沒有魔術規則。但是通常，例如，我們對控件使用較高的值將迫使控件採用較小的值，並可能防止飽和。

使用這些矩陣和 Matlab 函數 lqr，我們得到：

$$K = \begin{bmatrix} 0.3162 & 0.6875 \end{bmatrix}.$$

我們也可以使用魯棒控制部分的結果設計狀態反饋控制器。由於系統沒有不確定性，也沒有外部干擾，所以我們可以為名義動態設計一個狀態反饋控制器。使用系統數據和 Maltlab，我們得到：

$$X = \begin{bmatrix} 1.1358 & -0.3758 \\ -0.3758 & 1.1465 \end{bmatrix}$$

$$Y = \begin{bmatrix} -0.0092 & 0.0228 \end{bmatrix}$$

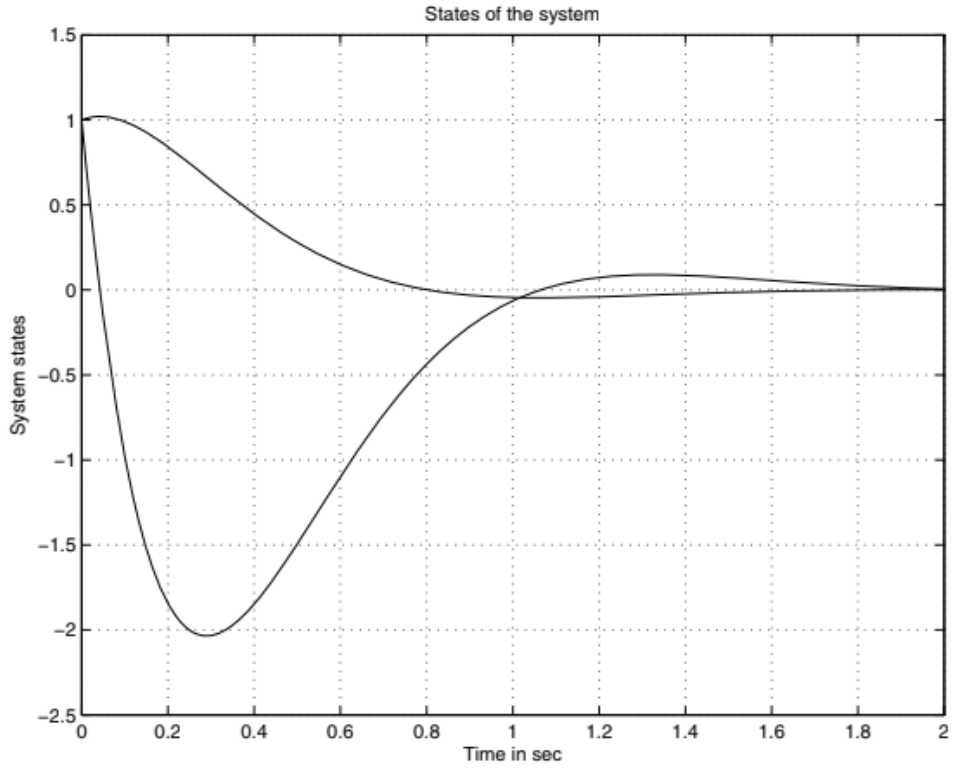

圖 11.8 系統狀態與時間

給出了相應的控制器增益:

$$K = \begin{bmatrix} -0.0017 & 0.0193 \end{bmatrix}.$$

備註 11.3.3 由於我們有直流電動機套件的連續時間模型，因此我們用它來設計控制器增益。在這種情況下，我們解決了以下 LMI：

$$AX + XA^{\top} + BY + Y^{\top}B^{\top} < 0$$

增益 K 由下式得出：$K = YX^{-1}$。

有關連續時間情況的更多詳細信息，請向讀者介紹 Boukas [2]及其中的參考文獻。

11.4 平衡機器人控制

從控制的角度來看，平衡機器人是一個具有挑戰性的系統，因為它是一個不穩定的開環系統。該系統吸引了許多研究人員，為此已經提出了許多設計方案。在這裡，我們將介紹

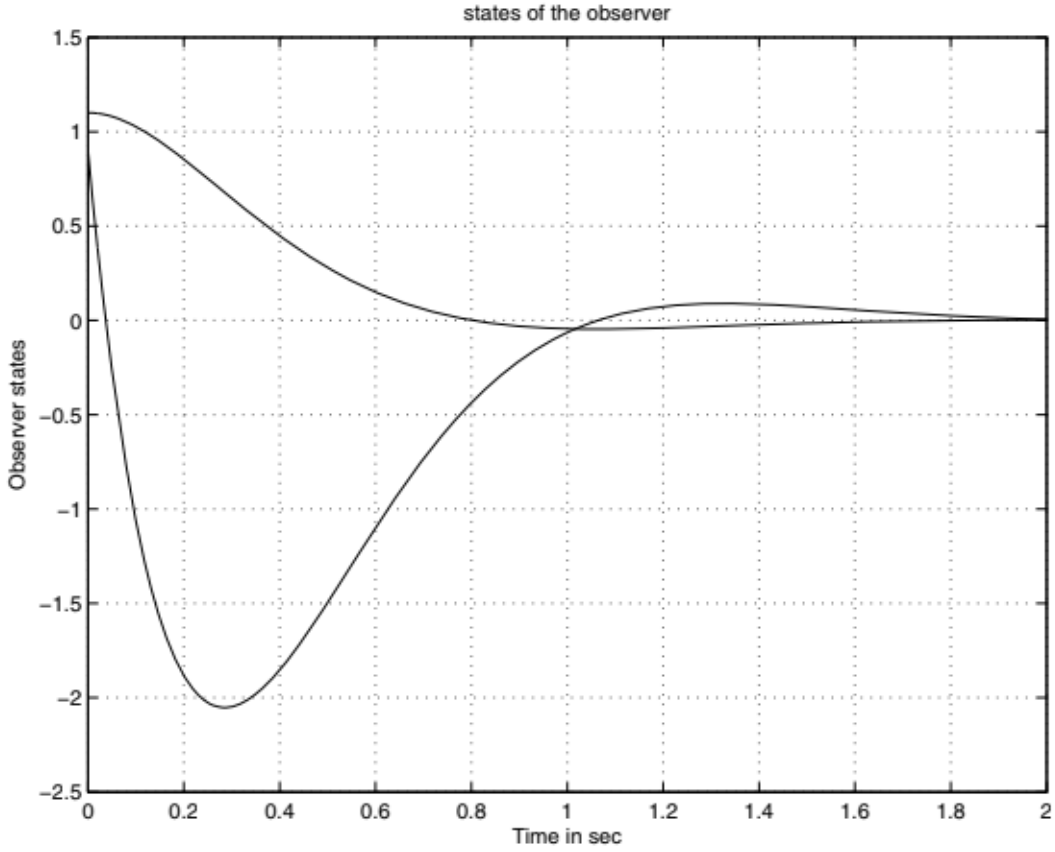

圖 11.9 觀察者狀態與時間

圖 11.10 與 11.11 給出了機器人的概念。它是為研究目的而開發的，目的是使機電一體化的學生能夠實現他們的控制算法並熟悉複雜的系統。該機器人有兩個獨立的輪子，每個輪子都由一個直流電動機通過一個傳動比為 1：6 的齒輪驅動。每個電機都有一個編碼器來測量軸的速度。這兩個電機連接到機器人的主體。其他傳感器（如加速度計和陀螺儀）用於測量傾斜角。引入了適當的濾波器以消除這些措施的噪聲，從而獲得有用的控制信號。

機器人的大腦圍繞 30F4011 系列的 MicrochipdsPIC 構建。在 Microchip 的 C30 使用 PCKit2 生成可執行代碼之後，所有的編程都以 C 語言完成，並插入到 dsPIC 中。

如果我們參考第 4 章，則數學模型如下：

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) \end{cases}$$

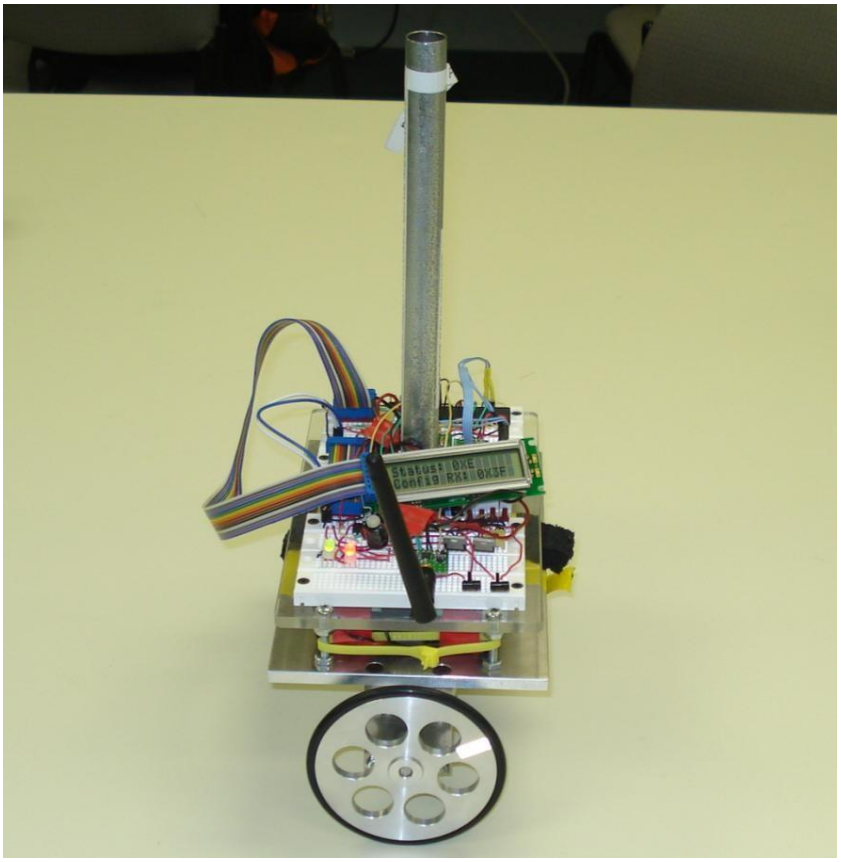

圖 11.10 平衡機器人

而

$$x(t) = \begin{bmatrix} \psi(t) \\ \dot{\psi}(t) \\ x(t) \\ \dot{x}(t) \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 147.2931 & -0.4864 & 0 & -10.6325 \\ 0 & 0 & 0 & 1 \\ 0 & -0.0429 & 0 & -0.9371 \end{bmatrix},$$

$$B = \begin{bmatrix} 0 \\ 1.4687 \\ 0 \\ 0.1295 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

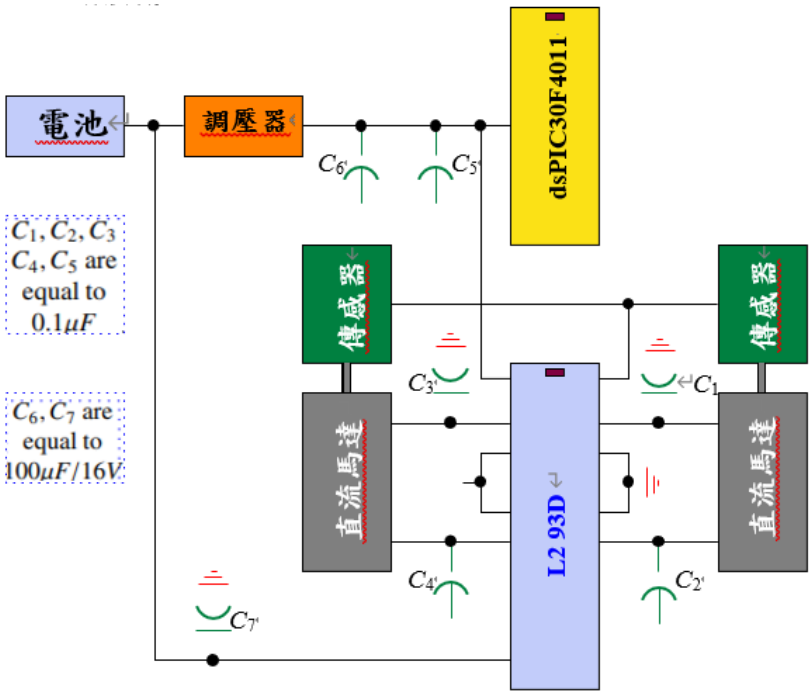

圖 11.11 平衡機器人的電子電路

由於系統在開環中不穩定，因此讓我們設計一個狀態反饋控制器，該控制器具有以下性能：

1. 系統閉環穩定；
2. 超調小於或等於 5%;
3. 2%的穩定時間等於 1.5 s;

從規格中我們得到：

$$\zeta = 0.707$$
$$w = \frac{4}{\zeta \times 1.5} = 3.7718 \text{ rad/s}$$

相應的主根對由下式給出：

$$s_{1,2} = -2.6667 \pm 2.6675j.$$

由於矩陣 A 的等級為 4，因此我們需要再放置兩個根點以確定狀態反饋控制器增益 K。讓我們選擇以下主根：

$$s_3 = -13.3335$$
$$s_4 = -13.3335$$

使用功能 acker，我們獲得了以下結果：

$$K = \begin{bmatrix} 339.5604 & 27.5946 & -132.5973 & -76.8450 \end{bmatrix}.$$

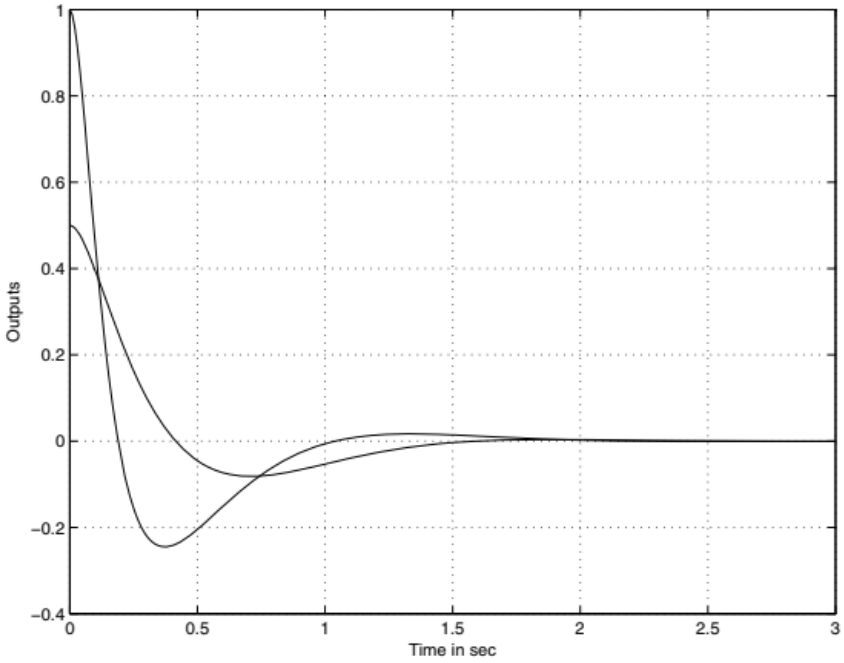

圖 11.12 輸出與時間

該控制器的模擬結果在圖 11.12 和圖 11.13 中示出。系統從初始狀態開始 $x_0^T = [1\ 0\ 0.5\ 0]$ 輸入為零。要是我們嘗試發送輸入參考，我們將在狀態或輸出中出現錯誤。克服這需要添加一個整體動作。如果我們用 $\tilde{x}(t) = \int_0^t (x_r(t) - x(t))\,dt,$ xr（t）為位置參考，然後跟隨[1]，我們得到：

$$\begin{aligned} \dot{\eta}(t) &= \tilde{A}\eta(t) + \tilde{B}u(t) \\ y(t) &= \tilde{C}\eta(t) \end{aligned}$$

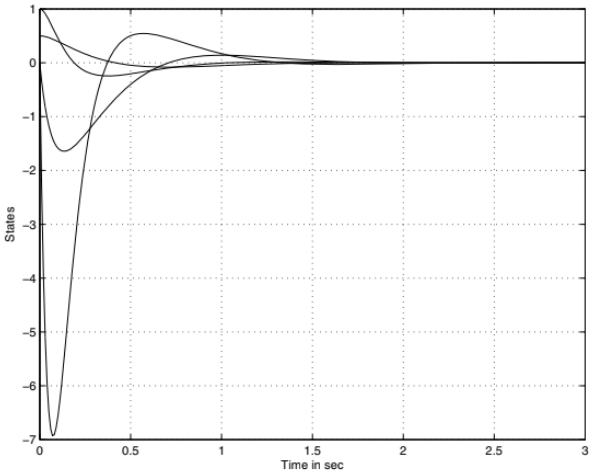

圖 11.13 狀態與時間

而

$$\begin{aligned}\eta(t) &= \begin{bmatrix} x(t) \\ \tilde{x}(t) \end{bmatrix}, \\ \tilde{A} &= \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}, \\ \tilde{B} &= \begin{bmatrix} B \\ 0 \end{bmatrix}, \\ \tilde{C} &= \begin{bmatrix} C & 0 \end{bmatrix}\end{aligned}$$

新的動力學變為五階，我們需要修復三個主導根點以及規範中的主導根點。這些根固定為以下根：

$$\begin{aligned}s_{3,4} &= -13.3335 \pm 2.6675\,j \\ s_5 &= -13.3335\end{aligned}$$

使用功能 acker，我們獲得了以下結果：

$$K = \begin{bmatrix} 339.5604 & 27.5946 & -132.5973 & -76.8450 & 0.1 \end{bmatrix}.$$

我們還可以使用線性二次控制技術設計狀態反饋控制器。實際上，如果我們選擇以下矩陣：

$$\begin{aligned}Q &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix} \\ R &= \begin{bmatrix} 10 \end{bmatrix}\end{aligned}$$

並使用 Matlab 函數 lqr 求解 Ricatti 方程，我們得到：

$$K = \begin{bmatrix} 218.9494 & 17.7264 & -1.0000 & -15.7658 \end{bmatrix}$$

閉環的相應特徵值由下式給出：

$$\begin{aligned}s_1 &= -12.7439 \\ s_2 &= -11.5936 \\ s_3 &= -0.9420 \\ s_4 &= -0.1371\end{aligned}$$

可以類似地獲得該控制器的仿真結果，並省略了細節。

對於該系統，我們還可以使用穩健控制理論設計狀態反饋控制器。可以連續時間或離散時間完成。由於我們的模型是連續的，因此我們將使用 LMI 連續進行設計。求解適當的 LMI（用於矩陣尺寸不同的直流電動機套件的 LMI，我們得

到：

$$X = \begin{bmatrix} 0.0595 & -0.2156 & 0.1053 & 0.0177 \\ -0.2156 & 1.9238 & -0.0705 & 0.7732 \\ 0.1053 & -0.0705 & 1.4496 & -0.3439 \\ 0.0177 & 0.7732 & -0.3439 & 1.5028 \end{bmatrix}$$

$$Y = \begin{bmatrix} -7.2149 & 27.5211 & -13.6001 & 7.4801 \end{bmatrix}$$

為狀態反饋控制器提供以下增益：

$$K = \begin{bmatrix} -174.2563 & -10.5741 & 6.0437 & 13.8536 \end{bmatrix}$$

閉環的相應特徵值由下式給出：

$$s_{1,2} = -6.9770 \pm 6.4079j$$
$$s_{3,4} = -0.6028 \pm 0.9598j$$

對於其他控制器的設計可以很容易地獲得，我們將此作為因為設計是為了編寫 Matlab 程序而為讀者進行的類似於我們在文本中給出的內容。

作為第一個示例，讓我們考慮兩輪機器人的 H∞控制問題。在這種情況下，我們在狀態動態中添加一個項。 該項為 Bw（t），其中 w（t）是具有有限能量的外部干擾。 通過 $\gamma = 0.1$ 求解適當的 LMI，我們得到：

$$X = \begin{bmatrix} 1.1233 & -2.7517 & 0.2742 & 1.2667 \\ -2.7517 & 61.0512 & -1.5973 & 14.4055 \\ 0.2742 & -1.5973 & 2.0232 & -3.4932 \\ 1.2667 & 14.4055 & -3.4932 & 16.4930 \end{bmatrix}$$

$$Y = \begin{bmatrix} -143.3219 & 316.5973 & -60.7376 & -33.8808 \end{bmatrix}$$

為狀態反饋控制器提供以下增益：

$$K = \begin{bmatrix} -193.7547 & -9.4711 & 39.7088 & 29.5092 \end{bmatrix}$$

閉環的相應特徵值由下式給出：

$$s_{1,2} = -3.8372 \pm 8.9414j$$
$$s_{3,4} = -1.9189 \pm 2.0781j$$

對於其他控制器的設計，可以很容易地獲得，並且我們將此作為讀者的練習，因為設計被引入來編寫類似於我們在本文中給出的 Matlab 程序。

11.5 磁懸浮系統

在本節中，我們將介紹前面介紹的磁懸浮系統。由我們的機電實驗室開發的機電系統由兩部分組成：一個固定的部分，代表線圈並產生電磁力，另一個是我們希望通過作用於電磁場產生的電磁力而放置在某個位置的鐵磁物體。線圈。該系統的目的是通過輸入電壓調節電磁鐵中的電流來控制移動物體的垂直位置。使用霍爾效應傳感器測量物體位置。dsPIC30F4011 周圍的電子電路通過 L298（集成電路）為線圈供電，電流與致動器的指令電壓成正比。由於磁力只具有吸引力，互導放大器會轉為否定命令。該系統如**圖 11.14** 所示。

該系統的數學模型由以下方程式給出:

$$m\ddot{l}(t) = mg - F_1 - F_2$$

其中 m 是運動物體的質量，l（t）R+是從電磁體測量的距離，F1 和 F2 分別是當電流為 i（t）時線圈產生的力以及電磁體與電磁體之間的電磁力。永久磁鐵放置在移動物體的頭部。

這些力的表達方式如下：

$$F_1 = K_1 \frac{i^2(t)}{l^2(t)}$$

$$F_2 = K_2 \frac{1}{l^2(t)}$$

我們可以線性化這個非線性模型，得到以下信息（請參閱第 1 章）:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases}$$

而

$$x(t) = \begin{bmatrix} x_1(t)(position) \\ x_2(t)(velocity) \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 1 \\ \frac{2\left[sign(u_2)k_c u_e^2 + k_p R^2\right]}{mR^2 x_e^3} & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ \frac{-2sign(u)k_c u_e}{mR^2 x_e^2} \end{bmatrix}$$

$$C = \begin{bmatrix} \frac{-3C_p}{0.032x_e^4} & 0 \end{bmatrix}$$

$$D = \frac{C_b}{0.032R}$$

該系統的數據由表 11.1 給出。使用此數據，矩陣由下式給出：

$$A = \begin{bmatrix} 0 & 1 \\ 2490.8 & 0 \end{bmatrix},$$
$$B = \begin{bmatrix} 0 \\ -1.2711 \end{bmatrix},$$
$$C = \begin{bmatrix} 473.5711 & 0 \end{bmatrix},$$
$$D = \begin{bmatrix} -0.0833 \end{bmatrix},$$

重要的是要注意，系統在開環中不穩定，因為它具有積極的一面。可以通過計算的特徵值來檢查矩陣 A。

讓我們設計一個狀態反饋控制器來保證以下性能：

1. 系統穩定閉環
2. 超調量小於或等於 0.2%
3. %2 的建立時間等於 0.05 秒

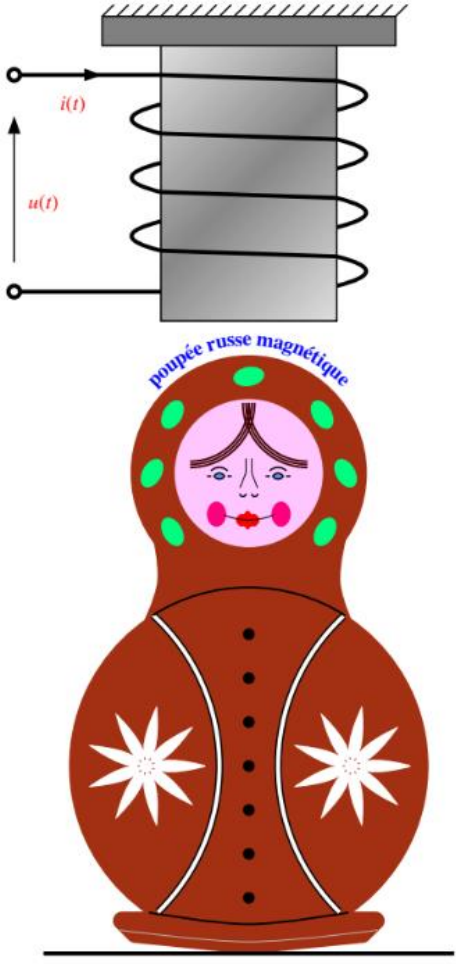

圖 11.14 磁懸浮系統

表 11.1 磁懸浮系統的數據

Variable	value
R	62.7 Ω
L	60 mH
m (object mass)	7.64 g
k_c	$5.9218\ 10^{-4}$
k_p	$4.0477\ 10^{-6}$
C_b	−0.1671
C_p	$-1.9446\ 10^{-8}$
diameter of the permanent magnet	9 mm

由於過衝小於或等於 0.2%，因此ζ= 0.9 的 2%的時間建立時間由下式給出：

$$t_s = \frac{4}{\zeta w_n}$$

wn 是自然脈衝。如果將穩定時間固定為 0.05 秒，我們將得到：

$$w_n = \frac{4}{0.05 \times 0.9} = 88.8889$$

設計的主要根點如下：

$$s_{1,2} = -\zeta w_n \pm j w_n \sqrt{1-\zeta^2} = -80.000 \pm 38.75j$$

使用這對根，我們得到：

$$K_1 = -175.6$$
$$K_2 = -125.9$$

使用該控制器，從給定的初始條件開始的時間響應如圖 11.15 所示。從該圖可以看出，過沖和建立時間就是我們想要的。

對於第二種情況，由於我們無法訪問負載速度，因此我們可以從位置計算負載速度，也可以使用觀察者來估計系統狀態。如前所述，我們用於觀察者設計的根點應該比控制器設計中使用的根點更快。

選擇以下根點（s1,2，為控制器設計中所用實數的四倍）：

$$s_{1,2} = -4\zeta w_n \pm j w_n \sqrt{1-\zeta^2}$$
$$= -320.0 \pm 38.75j$$

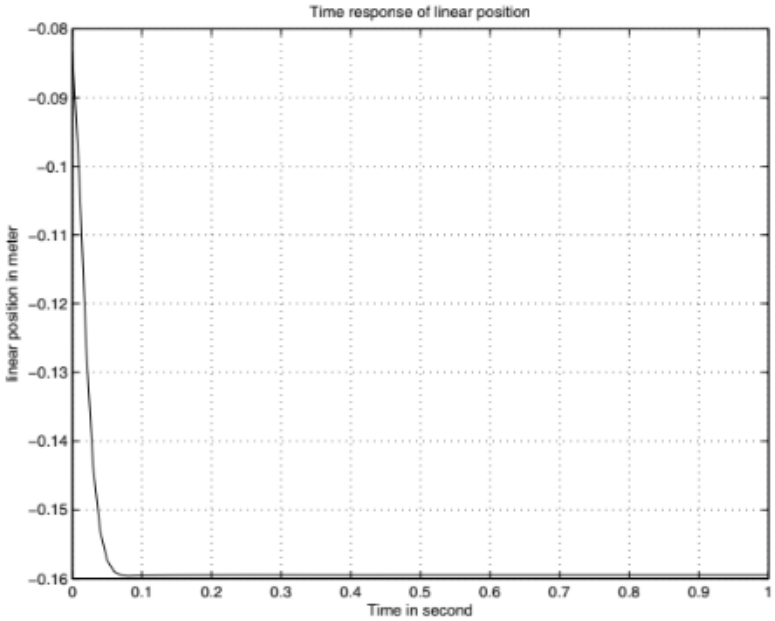

圖 11.15 運動物體的時間響應

我們為觀察者帶來以下結果：

$$L_1 = 1.351$$
$$L_2 = 224.5927$$

我們可以像我們對直流電動機套件和雙輪機器人所做的那樣試驗所有其他控制器，但是我們更喜歡讓這一部分作為練習供讀者練習工具。注意，我們邀請他/她在連續時間和離散時間情況下進行設計並比較結果。該系統的尺寸可以做到這一點。

下面給出了用於對該系統的狀態反饋控制進行編程的示例：

```
 #include <p30fxxxx.h>
#include <stdio.h>
#include <stdlib.h>
#include <adc10.h>
#include <math.h>
#include <uart.h>

//
// Configuration
//
// Interne frequency (30 MIPS) instructions/sec
// Number of samples: 7,37*16/4 = 29480000
```

```
_FOSC(CSW_FSCM_OFF & FRC_PLL16);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_OFF & MCLR_DIS);
_FGS(CODE_PROT_OFF);
_FICD( ICS_NONE );
__C30_UART=2;

//
// Variables

    #define Freq_pic 29480000 // PIC Frequency

    #define a11 7.8510061454215840e-001
    #define a12 2.2727760074413661e-004
    #define a21 5.0829248960838420e+001
    #define a22 1.0000643300984893e+000
    #define b11 3.7771272752681438e-005
    #define b12 4.5392069137012870e-004
    #define b21 8.7496385285734044e-003
    #define b22 1.0852723324638587e-001

    #define Ts 2.2727272727272727e-004
    #define u_max 1.1789999999999999e+001
    #define ref_tension 5.0000000000000000e+000
    #define ref_pic 1.0240000000000000e+003
    #define duty_cycle_ref 5.8481764206955049e+001
    #define x_ref 7.8768775539549939e-003
    #define u_ref 2.0000000000000000e+000
    #define y_ref 8.5691877396730676e-001
    #define K0 5.2128678707944724e+004
    #define K1 3.9336557697049994e+002

        double y[2] = {0.0, 0.0};
        double u[2] = {0.0, 0.0};
        double y_tilde[2] = {0.0, 0.0};
        double tension_tilde[2] = {0.0, 0.0};
        double tension = 0.0;
        double duty_cycle_tilde = 0.0;
```

```
    double lim_Sup = 0.0;
    double lim_Inf = 0.0;
    double position_tilde[2] = {0.0, 0.0};
    double vitesse_tilde[2] = {0.0, 0.0};
    double integrale_tilde[2] = {0.0, 0.0};
    double duty_cycle = 0.0;
    double temps_total = 0.0;
    double n = 6553500.0/65536.0;
    int compteur = 0;
    int compteur_freq = 0;
    int uart_flag = 1;
    unsigned long Val_reg = 0;

//
// Functions
//

void init(void){

INTCON1bits.NSTDIS=0; // Activation of the level of interruption
TRISE = 0; // Configuration of PORTE as output
TRISD = 0; //Configuration of PORTD as output
PORTEbits.RE8 = 1;
PORTEbits.RE2 = 0;
PORTDbits.RD0 = 1;
ADPCFG= 0xFFFF; // Configuration of the pins of PORTB as digital
I/O
}

void init_ADC (void){

SetChanADC10(ADC_CHX_POS_SAMPLEA_AN3AN4AN5 &
ADC_CHX_NEG_SAMPLEA_NVREF);

ConfigIntADC10(ADC_INT_DISABLE);

OpenADC10(ADC_MODULE_ON & ADC_IDLE_CONTINUE & ADC_FORMAT_INTG
  & ADC_CLK_AUTO & ADC_AUTO_SAMPLING_ON &
```

```
  ADC_SAMPLE_SIMULTANEOUS,
  ADC_VREF_AVDD_AVSS & ADC_SCAN_OFF & ADC_CONVERT_CH_0ABC
  & ADC_SAMPLES_PER_INT_1 & ADC_ALT_BUF_OFF & ADC_ALT_INPUT_OFF,
  ADC_SAMPLE_TIME_1 & ADC_CONV_CLK_SYSTEM & ADC_CONV_CLK_32Tcy,
  ENABLE_AN4_ANA & ENABLE_AN5_ANA, SCAN_NONE);
}

void init_Timer1 (void){

INTCON1bits.NSTDIS=0; // Activation of mode 16 bits of the Timer1
T1CONbits.TON = 1; // Autorisation du Timer1
T1CONbits.TGATE = 0; // Dsactivation du mode Timer Gate
T1CONbits.TSIDL=1; // Synchronisation of Timer1 sur le Idle mode
T1CONbits.TCKPS = 0; // Choice of the Prescaler 1:1
                          (1=1:8, 2=1:64)
T1CONbits.TCS=0; // Selection of the interne clock (0=FOSC/4)
IFS0bits.T1IF = 0; // Put to zero the overflow bit for the
                          interrupt of Timer
IEC0bits.T1IE = 1; // Activation of the interruption of Timer1
PR1 = 6699; // Sampling frequency at 4400 Hz environ
IPC0bits.T1IP = 5; // Priority 5 for the interruption of the
                          Timer1

}

/* ROUTINE D' INITIALISATION DE L' UART */

void init_UART (void){

ConfigIntUART2(UART_RX_INT_DIS & UART_RX_INT_PR0
& UART_TX_INT_DIS & UART_TX_INT_PR0);

    // Configuration of the register

U2MODEbits.UARTEN = 1; // UART pins controlled by UART
U2MODEbits.USIDL = 0; // UART communication continue in
                        Idle Mode
```

```
U2MODEbits.WAKE = 1; // Wake up enable in sleep Mode
U2MODEbits.LPBACK = 0; // Loopback mode disabled
U2MODEbits.ABAUD = 0; // Autobaud process disabled
U2MODEbits.PDSEL = 0; // 8-bit data, no parity
U2MODEbits.STSEL = 0; // 1 stop-bit.

// Configuration du registre U2STA

U2STAbits.UTXISEL = 0; // Transmission Interrupt Mode
                          Selection bit
U2STAbits.UTXBRK = 0; // UxTX pin operates normally
U2STAbits.UTXEN = 1; // Transmit enable
U2STAbits.URXISEL = 1; // Interrupt occurs when one charater
                          is received
U2STAbits.ADDEN = 0; // Address detect disabled

U2BRG = 31; // Value for 57600 bps baudrate
}
//
// Initialization of the complementary mode PWM
//
void init_PWM (void){

Val_reg = 1023; // Frquence de 30000 Hz environ
lim_Sup = (u_max*(2*Val_reg + 1)/(2*Val_reg + 2)) - u_ref;
lim_Inf = -u_max - u_ref;

PTCONbits.PTEN = 1; // Activation of the time base
PTCONbits.PTSIDL = 1; // Configuration in Idle Mode
PTCONbits.PTCKPS = 0; // Selection de 4TCY ( Prescale: 00 = 1:1;
                          01= 1:4; 10 = 1:16; 11 = 1:64)
PTCONbits.PTMOD = 0; // Selection of the free running mode

PTMRbits.PTDIR = 0; // Increment of the time base
PTMRbits.PTMR = Val_reg; // Register value of the Time base

PTPER = Val_reg; // Value of the signal period
```

```
PWMCON1bits.PMOD1 = 0; // Selection the mode PWM complementary
PWMCON1bits.PEN1H = 1; // Activation of the pins in mode PWM
PWMCON1bits.PEN1L = 1; // Activation of the pins in mode PWM

DTCON1bits.DTAPS = 0; // Time base unit is 1TCY
DTCON1bits.DTA = 0; // Value of the DT for the unity A
PDC1 = 0; // zero of the dutycycle

}

void __attribute__((interrupt, auto_psv)) _T1Interrupt (void){

if (IFS0bits.T1IF){

PORTEbits.RE2 = !PORTEbits.RE2;
PDC1 = (2.0 * (Val_reg + 1) * duty_cycle)/100.0; // Calcul de la
          valeur du registre PDC1
y[0] = (ReadADC10(2)*ref_tension)/ref_pic; // Signal of the
                                              sensor in Volts
y_tilde[0] = y[0] - y_ref;

position_tilde[1] = position_tilde[0];
vitesse_tilde[1] = vitesse_tilde[0];
integrale_tilde[1] = integrale_tilde[0];
y_tilde[1] = y_tilde[0];
tension_tilde[1] = tension_tilde[0];

position_tilde[0] = (a11*position_tilde[1]+a12*vitesse_tilde[1]
                    +b11*tension_tilde[1]+b12*y_tilde[1]);
vitesse_tilde[0] = (-a21*position_tilde[1]+a22*vitesse_tilde[1]
                    +b21*tension_tilde[1]+b22*y_tilde[1]);
tension_tilde[0] = (K0*position_tilde[0]) +
                  (K1*vitesse_tilde[0]);// + N*ref;

if(tension_tilde[0]>lim_Sup){tension_tilde[0] = lim_Sup;}
// Saturation of the tension tilde
if(tension_tilde[0]<lim_Inf){tension_tilde[0] = lim_Inf;}
```

```
tension = u_ref + tension_tilde[0];
duty_cycle_tilde = tension_tilde[0]*(50.0/u_max);
duty_cycle = duty_cycle_ref + duty_cycle_tilde; // Computation
                of the duty cycle in percentage

temps_total += Ts;

compteur_freq = 0;

compteur++;
if(compteur == 10){ // Print data every 1 ms
compteur = 0.0;
uart_flag = 1;
}

IFS0bits.T1IF = 0; // put to zero of the overflow bit
}
}

/* PROGRAMME PRINCIPAL */

int main (void){

    init();
    init_PWM();
    init_ADC();
    init_UART();
    init_Timer1();
    while(1){
        init();
        init_PWM();
        init_ADC();
        init_UART();
        init_Timer1();
        while(1){
            if (uart_flag){
            printf("%lf %lf %lf %lf %lf %lf %lf\n\r",
            temps_total, tension, u_ref, y[0], y_ref,
```

```
                position_tilde[0], x_ref);
                uart_flag = 0;
                }
    }
}
```

11.6 結論

本章涵蓋了一些案例研究，這些案例是在我們位於蒙特利爾的E'cole PolytechniquedeMontre´al 的機電一體化實驗室中開發的。我們詳細介紹了機電系統設計的所有步驟。重點放在每個提出的系統的控制算法的設計。

11.7 問題

1.讓我們考慮一個具有以下數據的動態離散時間系統：

$$
\begin{aligned}
A &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -2 & -3 \end{bmatrix}, \\
B &= \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \\
C &= \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}, \\
D &= \begin{bmatrix} 0 \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
D_A &= \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix}, \\
E_A &= \begin{bmatrix} 0.1 & -0.2 & -0.1 \end{bmatrix}, \\
D_B &= \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix}, \\
E_B &= \begin{bmatrix} 0.1 & -0.2 \end{bmatrix}, \\
D_C &= \begin{bmatrix} 0.1 \end{bmatrix}, \\
E_C &= \begin{bmatrix} 0.1 & 0.2 & -0.1 \end{bmatrix},
\end{aligned}
$$

(a)　為標稱系統設計以下控制器：

i.　狀態反饋控制器

ii.　靜態輸出反饋控制器

iii.　動態輸出反饋控制器

(b)　為不確定係統設計以下控制器：

i.　狀態反饋控制器

ii.　靜態輸出反饋控制器

iii.　動態輸出反饋控制器

(c)　如果我們在狀態動力學中有一個額外的術語會增加外部干擾：$x(k+1) = [A + \Delta A]\,x(k) + [B + \Delta B]\,u(k) + B_w w(k).$。為標稱和不確定係統設計控制器（狀態反饋、靜態輸出、動態輸出反饋），以確保 H∞性能。

(d)　設計狀態反饋，靜態輸出反饋和動態反饋控制器，以確保所有容許的不確定性的保證成本。

2.在這個問題中，我們邀請您進行小船的設計，您可以使用操縱桿控制小船，使其例如在小紫膠中移動。

(a)　給出原理圖設計（電池、電動機等）

(b)　建立數學模型

(c)　修正您想要的規格，並設計合適的控制器以達到這種性能

3.在這個問題上，我們邀請您進行小型飛機的設計，您可以使用操縱桿控制小型飛機的飛行。

(a)　給出原理圖設計（電池、電動機等）

(b)　建立數學模型

(c)　修正您想要的規格，並設計合適的控制器以達到這種性能

4.在這個問題上，我們要求設計真空吸塵器。此設備應為自動設備，並避免其環境中的障礙物。設計便宜的，可以通過發射器、接收器和操縱桿進行無線通信的控制器也很重要。

(a)　給出原理圖設計（電子電路、電機等）

(b)　建立數學模型

(c)　修正您想要的規格，並設計合適的控制器以達到這種性能

5.在這個問題上，我們邀請您繼續進行機電系統的設計，該系統控制乒乓球的小球的位置。可以使用壓縮空氣來定位球。

(a)　給出原理圖設計（電子電路、電機等）

(b)　建立數學模型

(c)　修正您想要的規格，並設計合適的控制器以達到這種性能

6.在此問題中，我們希望設計一種單腿機器人，該機器人可以在保持垂直位置的情況下使用一個輪子移動。提供此類機電一體化系統的設計。

(a)　給出原理圖設計（電子電路、電機等）

(b)　建立數學模型

(c) 修正您想要的規格，並設計合適的控制器以達到這種性能

7.太陽能是可以使用的替代能源。在這個問題上，我們要求您設計一個太陽能係統，以使太陽能電池板產生的能量最大化。

(a) 給出原理圖設計（電子電路、電機等）

(b) 建立數學模型

(c) 修正您想要的規格，並設計合適的控制器以達到這種性能

8.在這個問題中，我們要求設計一種可通過操縱桿通過發射器和接收器控制在水上進行密封的胡佛。

(a) 給出原理圖設計（電子電路、電機等）

(b) 建立數學模型

(c) 修正您想要的規格，並設計合適的控制器以達到這種性能

附錄 A C 語言教程

本附錄的目的是複習 C 語言，以刷新讀者的記憶，並幫助他開始編寫程序，而無需閱讀有關該主題的大量書籍。我們的目的不是替代這些書籍，我們強烈鼓勵不熟悉該主題的讀者閱讀 Kernighan 和 Ritchie 的書籍。

首先，我們邀請讀者下載 C 編譯器和文本編輯器。為了體驗我們將提供的 C 語言中的不同程序，讀者必須鍵入程序，保存並編譯。有關如何執行此操作的更多詳細信息，請參閱讀者閱讀二手 C 編譯器的手冊。

開始我們的教程，讓我們考慮我們的第一個簡單程序。該程序旨在編寫**歡迎使用機電一體化課程**。該程序的清單由以下幾行給出：

```
#include < stdio.h>
void main()
      {
        printf("\nWelcome to mechatronics course\n");
      }
```

要查看此簡單程序的輸出，我們需要一個編輯器和一個 C 編譯器。

每個 C 程序均由變量和函數組成，並且必須具有“主”函數。除了保留字以外，必須先聲明所有變量和函數，然後才能使用它們。變量可以採用以下類型之一：

- 整數
- 真實
- 字符
- 其他

這些功能指定了程序必須執行的任務以及為其設計的任務。“主要”功能建立了代碼的整體邏輯。

```
#include < stdio.h>
```

包括 CI / O 庫的規範。按照慣例，.h 文件是**頭文件**，其中包含程序功能所必需的變量和函數的定義，無論它可以是用戶編寫的代碼部分，還是標準 C 庫的一部分。

指令

```
#include
```

告訴 C 編譯器在代碼中該點插入指定文件的內容。

記法

```
<...>
```

指示 C 編譯器在某些**標準**系統目錄中查找文件。

以 **main** 開頭的 void 表示 main 是 **void** 類型的，也就是說，它沒有與之關聯的類型，這意味著從另一種意義上說，它不能在執行期間返回結果。

；表示語句的結尾。 如同功能定義一樣，語句塊放在花括號{···}中。 所有 C 語句均以自由格式定義，即沒有指定的佈局或列分配。 空格（製表符或空格）絕不重要，除非在引號內作為字符串的一部分。

語句 printf 行在 **stdout**（與運行代碼的 X 終端窗口相對應的輸出流）上打印消息**歡迎使用機電一體化課程**。

```
\n
```

打印**換行**字符，將光標移至下一行。根據構造，函數 printf 絕不會單獨插入此字符，而將其交給程序員。

標準 C 語言具有一些保留的關鍵字，程序員不能使用它們來命名變量或其他目的，並且他必須使用它們，因為建議這樣做，否則在編譯過程中會出現錯誤。這些關鍵字在**表 A.1** 中列出。

Table A.1 List of C language keywords

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Table A.2 Number representations

Base	Representation	Chiffres permis	Example
Decimal (10)		0123456789	5
Binary (2)	0b...	01	0b10101010
Octal (8)	0...	01234567	05
Hexadecimal (16)	0x...	0123456789ABCDEF	0x5A

Table A.4 Decimal representations

Type	Taille (bits)	Emin	Emax	Nmin	Nmax
float	32	-128	+127	2^{-126}	2^{128}
double	32	-128	+127	2^{-126}	2^{128}
long double	64	-1022	+1023	2^{-1022}	2^{1024}

讓我們看一下常量和變量。必須先定義常量，然後才能使用它。我們用來定義這些常數的結構是：

```
#define name value
```

名稱一詞是我們給常量的名稱，值是該常量所取的值。以下示例提供了一些常量：

```
#define acceleration 9.81
#define pi 3.14
#define mot "welcome to mechatronics course''
#define False 0
```

通常，當我們處理數據時，我們使用不同的數字表示形式。表 A.2 提供了在對微控制器進行編程時通常使用的基礎。

對於變量，我們還必須在聲明它們之前。使用以下語法：

```
type <name>;
```

其中類型是以下之一：

- int（用於整數變量）
- short（短整數）
- long（對於長整數）
- float（用於單精度實數（浮點）變量）
- double（用於雙精度實數（浮點）變量）
- char（用於字符變量（單字節））

表 A.3 和 A.4 給出了每種類型的值。值得一提的是，編譯器會檢查程序中使用的所有變量的類型是否一致，以防止出錯。以下示例顯示了一些變量：

```
int i,j,k;
float x,y;
unsigned char var1;
unsigned char var2[10] = "welcome";
```

一旦定義了這些變量和常量，我們可以做什麼？答案是我們可以做很多事情，例如：

- 算術運算（作用於一個或多個變量）
- 邏輯運算（作用於一個或多個變量）

- 關係運算（作用於一個或多個變量）
- 其他

表 A.5 和 A.6 給出關於我們可以執行的不同操作的想法常數或變量。

Table A.5 Arithmetic operations

Symbol	Meaning	example
+	addition	$u[k] = u_1[k] + u_2[k];$
-	substrate	$u[k] = u_1[k] - u_2[k];$
/	division	$u[k] = u_1[k]/u_2[k];$ ($u_2[k]$ must be different from zero)
*	multiplication	$u[k] = K * x[k];$
%	modulo	$u[k] = u_1[k]\%u_2[k];$

Table A.6 Logic operations

Symbol	Meaning	example
&	ET	$u[k] = u_1[k]\&u_2[k];$
|	OR	$u[k] = u_1[k] \mid u_2[k];$
∧	XOR	$u[k] = u_1[k] \wedge u_2[k];$
~	inversion	$u[k] = \sim x[k];$
≪	shift left	$u[k] = u_1[k] \ll u_2[k];$
≪	shift right	$u[k] = u_1[k] \gg u_2[k];$

通常，我們需要在屏幕或 LCD 上打印數據。為此，可以使用打印功能。可以指示此函數正確打印整數，浮點數和字符串。通用語法是

```
printf( "format", variables );
```

其中**格式**指定轉換規格，變量是要打印的數量的列表。

有用的格式有：

```
%.nd integer (optional n = number of columns; if 0, pad with zeroes)
%m.nf float or double (optional m = number of columns,
n = number of decimal places)
%ns string (optional n = number of columns)
%c character
\n \t to introduce new line or tab
\g ring the bell ( ‘ ‘beep’ ’ ) on the terminal
```

在某些程序中，最好使用循環的概念來執行需要對數據流或內存區域進行重複操作的計算。在標準 C 中有幾種循環方法，以下是最常用的循環方法：

```
// while loop
while (expression)
{
```

```
block of statements to be executed
}
// for loop
for (expression_1; expression_2; expression_3)
{
block of statements to be executed
}
```

在某些情況下，我們需要根據某些條件的實現或取決於給定變量的值來採取措施。在這種情況下，可以使用單詞 if 或 if else 以及開關。使用以下結構：

```
if (conditional_1)
{
block of statements to be executed when conditional_1 is true
}
else if (conditional_2)
{
block of statements to be executed when conditional_2 is true
}
else
{
block of statements to be executed otherwise
}
```

表 A.7 和 A.8 顯示了我們可能在表達式中使用的最常用的條件運算符。

Table A.7 Logic operations

Symbol	Meaning	example
&&	and	*x&&y*
||	or	*x||y*
!	not	*x!y*

Table A.8 Logic operations

Symbol	Meaning	example
<	smaller than	$x < y$
<=	smaller than or equal to	$x <= y$
==	equal to	$x == y$
! =	not equal to	$x! = y$
>=	greater than or equal to	$x >= y$
>	greater than	$x > y$

```
switch (expression)
{
case const_expression_1:
{
block of statements to be executed
break;

}
case const_expression_2:
{
block of statements to be executed
break;

}
default:
{
block of statements
}
}
```

C 語言允許程序員使用指針直接訪問內存位置。為了說明其工作原理，讓我們考慮定義如下的變量 Xposition：

```
float Xposition;
Xposition = 1.5;
```

例如，如果我們要獲取變量 Xposition 的地址，則可以使用以下內容：

```
float* pXposition;
float Xposition;

Xposition = 1.5;
pXposition = &Xposition;
```

此代碼中，我們定義了一個指向 float，Xposition 類型的對象的指針，並將其設置為等於變量 Xposition 的地址。

要獲得指針所引用的存儲位置的內容，可以使用*運算符（這稱為對指針進行解引用）。因此，* pXposition 表示 Xposition 的值。

任何類型的數組在 C 中都起著重要的作用。數組很簡單，下面給出了這一點：

```
type varname[dim];
```

其中 dim 是我們要賦予數組 varname 的維。

在標準 C 中，數組從位置 0 開始，並且其所有元素都佔據內存中的相鄰位置。因此，如果矩陣是一個數組，則* matrix 是同一件事作為矩陣[0]，而*（matrix + 1）與 matrix [1]相同，依此類推。