Control System II

Motor Control Lab

106030009 葉蓁

本次實驗專題目的為：實作一個「馬達角度控制」的控制系統，並期望能使用控二所學的“State Space Design”設計補償器，並且轉換成數位控制器，實現到微控制器版上。

實驗器材：

1. 馬達
2. STM32 Nucleo-F446RE 開發板
3. 馬達驅動板
4. 筆電(使用 Mbed 撰寫程式、Serial oscilloscope 觀測實驗結果)
5. 電源供應器

實驗步驟：

分為三個階段：

一、 熟悉 UART 及 STM32 LED 控制

二、 測量轉速及計算轉移函數

三、 完成控制器設計，實現馬達角度控制

以下將依這三個階段各別說明：

一、 熟悉 UART 及 STM32 LED 控制

實驗一其實有個最重要的任務：**焊接馬達驅動版**。這部分如果沒做好，會影響後面實驗的進行。根據助教給的電路圖，我們依序把電子元件焊上去。因為是第一次焊接電路板，雖然有上網查很多教學，但實際操作起來還是沒那麼容易。也許是因為設備不夠高級，常常焊錫會黏在焊槍上無法焊上電路板，或是焊槍溫度太高(60 瓦)焊錫直接揮發掉。經過多次修正後，終於完成焊接。檢測方式：通 9V 電池，檢查三個 LED 燈會不會亮。

完成驅動版焊接後，接續兩個任務：

1.了解 UART 如何使用

比較重要的為以下三個部份的 code：

(1) 宣告 Serial 變數： bt(USBTX, USBRX)

```
1  #include "mbed.h"
2
3  //------------------------------------
4  //----------------- Hardware Declaration ----
5  //------------------------------------
6  Serial bt(USBTX, USBRX);
7
```

(2) 初始化 UART()：

a. 設定 baurate 為 115200

b. Enable uart_rx_itr_read 函式

```
36
37  void init_UART(){
38
39      bt.baud(115200);
40      bt.attach(&uart_rx_itr_read, Serial::RxIrq);
41      printf("UART OK\r\n");
42
43  }
44
```

(3) uart 接收腳位的 Interrupt 函式

a. 如果 bt.readable() (可以接收資料)

b. 宣告一個 char c，接收使用者從電腦按下的鍵盤輸入，這邊我設定如果按下 “u” ，則 Serial 會印下此刻程式執行多少時間，完成 Lab1 第一個驗收。

```
46  void uart_rx_itr_read(){
47      while(bt.readable()) {
48          //type your code here
49          char c = bt.getc();
50          if(c == 'u'){
51              bt.printf("This program runs since %d seconds.\r\n", i);
52           }
53      }
54  }
```

2.用 stm32 控制馬達驅動版上的 LED 亮滅行為。

在第二個驗收，目的為：輸入兩種指令，使驅動版上的 LED 燈以兩種模式交替閃爍：(1)0.5 秒 (2) 2 秒

在程式撰寫部分，使用一個很簡單的 Finite state machine 架構，定義兩個 state: short(以 0.5 秒閃爍), Long(以 2 秒閃爍)

- Default: 預設 state 為 short (初始為 0.5 秒交替閃爍)

```
16    int state = 0;
17    #define short 0
18    #define long 1
19    float blink_half = 0.5; //0.5sec
20    float blink_two = 2;    //2sec.
21
```

接著宣告會用到的變數及函式：

```
1   #include "mbed.h"
2
3   //-----------------------------------------
4   //---------------- Hardware Declaration -----
5   //-----------------------------------------
6   Serial bt(USBTX, USBRX);
7   DigitalOut myled1(A4);
8   DigitalOut myled2(A5);
9   Ticker tim;
10
```

DigitalOut 的 myled1(A4)意思是 stm 對到的 A4 pin 腳位為我們要控制的閃爍的 myled1 燈。myled2(A5)以此類推。

Ticker tim 是用來控制秒數的。

```
26
27    void init_TIMER();
28    void init_UART();
29    void uart_rx_itr_read();
30    void timer_ITR();
31
```

接著進入主程式部分：

值得注意的是，main function 內除了初始化兩個 function，在無窮回圈內是不做事的，意思是任何行為都是用 interrupt 來觸發。

```
34  int main() {
35      init_UART();
36      init_TIMER();
37
38      while(1) {
39      }
40  }
41
```

在 init_UART()中，和驗收一一樣 baurate 設 115200，

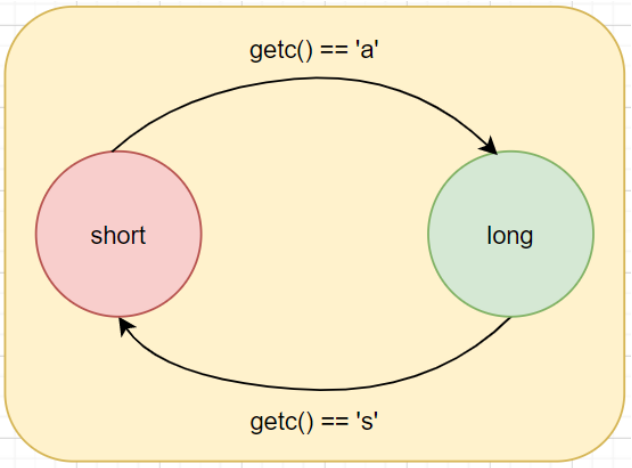
並且 attach 一個 interrupt function 來接收 uart 輸入。

```
43
44  void init_UART(){
45      bt.baud(115200);
46      bt.attach(&uart_rx_itr_read, Serial::RxIrq);
47      printf("UART OK\r\n");
48
49  }
50
```

Uart 輸入分為兩類，

若按下 a 鍵：進入 long state。

若按下 s 鍵：進入 short state。




```
52  void uart_rx_itr_read(){
53
54      while(bt.readable()) {
55      //type your code here
56          char c = bt.getc();
57          if(c == 'a'){
58              state = long;
59              bt.printf("Now switch to long pattern.\r\n");
60           }
61          if(c == 's'){
62              state = short;
63              bt.printf("Now switch to short pattern.\r\n");
64           }
65      }
66  }
67
```

初始化 Timer 部分，其實只有把一個 timer_ITR()函式已每 1000 μs(1ms 秒觸發一次)。

```
70  void init_TIMER()
71  {
72      tim.attach_us(&timer_ITR, 1000);
73  }
74
```

而真正控制 LED 閃爍的函示在 timer_ITR 內：

在此宣告一個 static int aa 當作一個 counter，每次 interrupt 進來時 counter 會+1(多 1 毫秒)。使用 static 目的為：在下一次 interrupt 進來後，會紀錄上次的值，繼續加秒數。

接著用 state 來區分：

1. 若現在是 short state：當 counter 小於 blink_half*1000 的時候，使 A4 亮，A5 暗。因為 blink_half 為我們上面定義的 0.5 秒，所以 0.5*1000(ms) = 0.5 second。又因為初始化 aa 是 0，表示：第 0 到第 0.5 秒中：是讓 A4 亮 A5 暗；第 0.5 到 1 秒，是 A4 暗 A5 亮。記得：1 秒後要把 counter 歸零，判斷式才會正常運作。
2. 若現在式 long state，程式邏輯相同，不同的是 blink_two = 2，所以第 0-2 秒是讓 A4 亮 A5 暗；第 2 到 4 秒，是 A4 暗 A5 亮。大於 4 秒後歸零重來。

```
76  void timer_ITR(){
77      static int aa = 0;
78      aa++;
79      switch(state){
80          case short:
81                  if(aa < blink_half*1000) {
82                      myled1 = 1;     //A4 high
83                      myled2 = 0;     //A5 low
84                      }
85                  else {
86                      if(aa < blink_half*2000){
87                          myled1 = 0; //A4 low
88                          myled2 = 1; //A5 high
89                      }
90                      else aa = 0;
91                  }
92          break;
93          case long:
94                  if(aa < blink_two*1000) {
95                      myled1 = 1;     //A4 high
96                      myled2 = 0;     //A5 low
97                      }
98                  else {
99                      if(aa < blink_two*2000){
100                         myled1 = 0; //A4 low
101                         myled2 = 1; //A5 high
102                     }
103                     else aa = 0;
104                 }
105         break;
106     }
107 }
```

二、測量轉速及計算轉移函數

第二部分的實驗：馬達鑑別。有三個驗收：

- 驗收 1:　僅需馬達會轉
- 驗收 2:　pc.printf 的轉速需要與轉速計相同
- 驗收 3:　透過示波器找到轉移函數

驗收 1：

在驗收 1 部份，其實花了我們最多時間。因為馬達有時候轉，有時候不轉。本來以為是我們程式 PWM 輸入訊號打錯或其他問題，最後經助教協助，才發現應該是「馬達驅動板」沒焊好、假焊的問題。經過組員努力拯救，終於把驅動板補好。**焊接電路板真的是一門學問!**看來只有熟能生巧，沒有別的訣竅。

驗收 2：我們測量轉速的公式如下：

$$\text{velocityA} = \frac{\text{EncoderPositionA} * 60}{\text{dt} * HALL\_RESOLUTION * \text{GEAR\_RATIO}}$$

由於 TIM2 的 CNT(count) 每次進來時會歸零，每次進入這個 ReadVelocity 函式都確保式上次與這次的「相對位移」。

(1) 除以 dt：轉速 = 單位時間的位移格數

(2) 除以 HALL_RESOLUTION：因為 Encoder 轉一圈，會增加 60 格位置。所以除以 60 為了得到 revolutions per second。

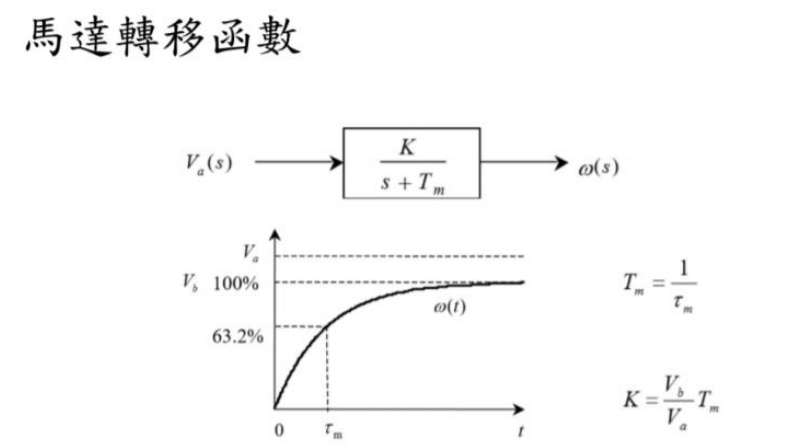
(3) 除以 GEAR_RATIO：因為 encoder 的轉速到馬達「指針轉速」有一個減速比的差距：因此要求馬達指針轉速，要除以他的減速比。

(4) 乘以 60 原因是為了轉成 r.p.m. (revolutions per minute)。

```
void ReadVelocity() {
    short EncoderPositionA;
    short EncoderPositionB;

    EncoderPositionA = TIM2->CNT ;
    EncoderPositionB = TIM3->CNT ;

    // if you want count to zero you can do it
      TIM2->CNT = 0;
//    TIM3->CNT = 0;

    ////////////////////////////////////////////////////// maybe you need to do something here

    //////////////////////////////////////////////////////

    velocityA = EncoderPositionA*60/(dt*HALL_RESOLUTION*GEAR_RATIO); //unit:rpm
    // please refer to PPT. (then you will find out you may need to add some variables)
    velocityB = EncoderPositionB/you_need_to_do;

//    pc.printf("Velocity: %.3f rpm\r\n",velocityA);
}
```

驗收 3: 透過示波器找出轉移函數：

使用轉移函數公式如下：



透過將示波器的資料輸出成文字檔，放入 excel 後計算。

使用公式求出我們馬達的 transfer function：

$$\text{Plant } G(s) = \frac{228.44}{s(s+14.13)}$$

三、完成控制器設計，實現馬達角度控制

在 main 主程式中，若按下 button，則觸發 step_command 一次。另外每 1000000 μ s = 1second 觸發一次 position_control。(即 sample period = 1second)

```
74  int main() {
75      pc.baud(115200);            //default baud rate
76      InitEncoder();              //don't change it
77
78      InitMotor(PWM_FREQUENCY);   // Set pwm period. Don't ch
79
80      mybutton.fall(&step_command);   // if you push blue bot
81
82      main_function.attach_us(&position_control, dt*1000000);
83
84      while(1){}
85  }
```

在 step_command()中，用一個 counter 判斷。若按下次數為奇數次，命令為「轉 90 度」。若按下次數為偶數次，命令為「轉 0 度」。

```
87  int counter=0;
88  void step_command() {
89      counter++;
90      if(counter%2 ==0){
91          command = 0;
92      }else{
93          command = 90;
94      }
95      led1 = !led1;
96      button_state = !button_state;
97      // reset = true;   // should reset now.
98      // tracking = 1;
99      ////////////////////////////////////////////////////////
100
101     // you may change your command in here.
102
103     ////////////////////////////////////////////////////////
104 }
105
```

```
107   void position_control() {
108 > #if CONTROLLER == 0···
120   #endif
121
122   #if CONTROLLER == 1
123       ReadPosition();
124       e = positionA - command;
125       u = Controller(e);
126       motor_drive(u, 0);
127
```

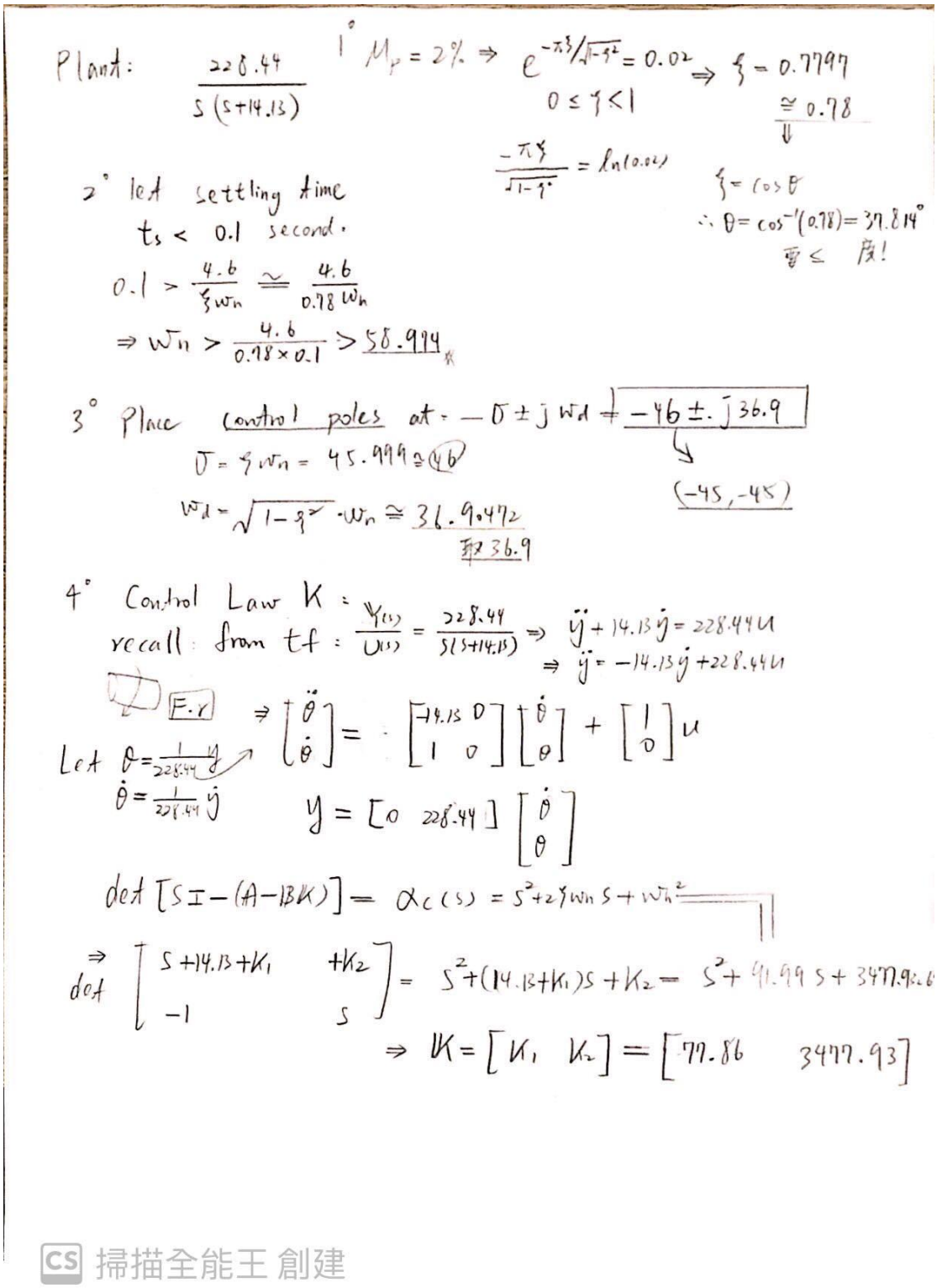
首先因為要進行馬達角度控制，為了取得 Sensor 回授之角度訊號，要撰寫 ReadPosition()，如下：

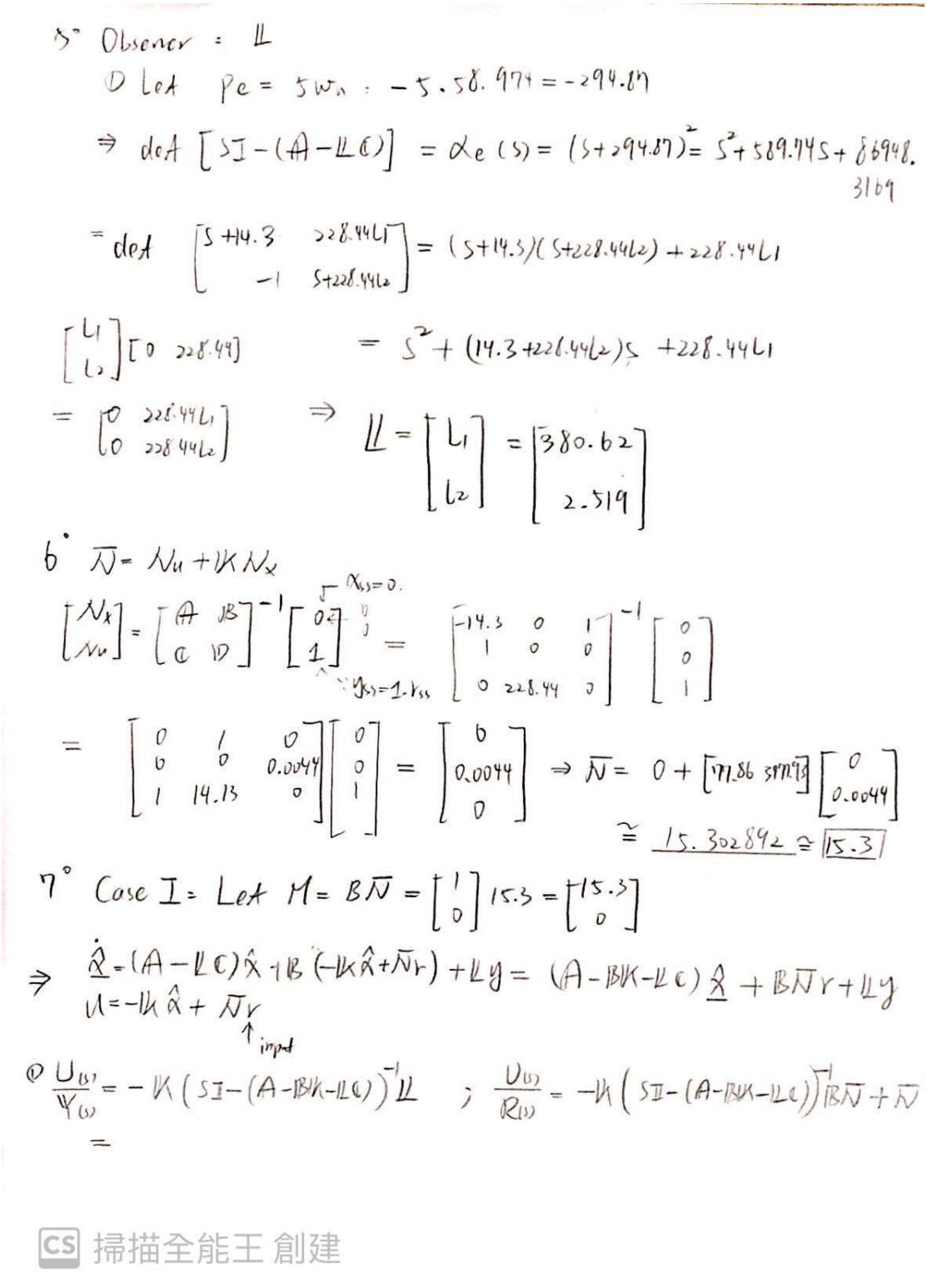
```
199  void ReadPosition() {
200      short EncoderPositionA;
201      short EncoderPositionB;
202
203      EncoderPositionA = TIM2->CNT;
204      EncoderPositionB = TIM3->CNT;
205
206      // if you want count to zero you can do it
207  //    if(reset== true){
208  //        reset = false; // already reset.
209  //        TIM2->CNT = 0;
210  //        TIM3->CNT = 0;
211  //    }
212
213      // Be careful if you always don't set the register to zero, it might overflow.
214
215      positionA = EncoderPositionA*360/(HALL_RESOLUTION*GEAR_RATIO);  //output
216      positionB = EncoderPositionB / you_need_to_do;
217
218
219  //    pc.printf("%d\r\n",positionA);
220  }
```

只需將 velocityA 的 dt 部分與 60 部分拿掉，因為需要讀取每個當下的位移 counter 讀數。要注意的是 Encoder 讀數是「相對」而非絕對的。

補償器設計思路：

我們最初使用 state space 設計一組 Compensator，設計思路如下：





轉成差分方程後，不知為何馬達不會動。因此我們先使用 P control 嘗試控制成效。只用 P control 就已經可將穩態誤差降到最小。

(理論上因為系統 system type 是 Type 1，追 step command 應該無穩

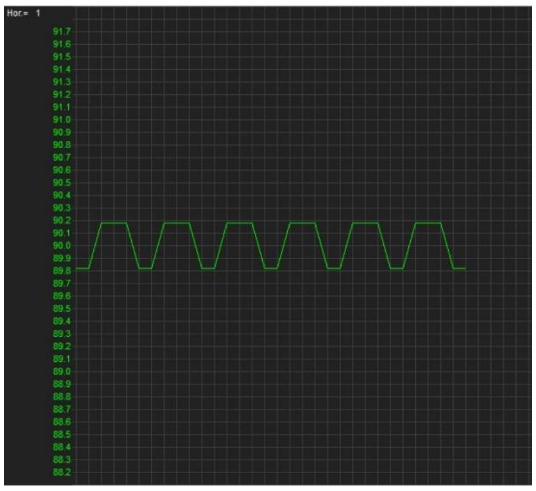
態誤差，但因為有摩擦力的影響，如上解釋，會有些微誤差。)

注意：在 Lab2 有要我們找出馬達的**最低可動電壓**，目的即為為克服摩擦力造成的非線性影響。在誤差很小時，透過控制算出的驅動信號 u，若小於「馬達可動最低電壓」e.x. 1 伏特，馬達即使有收到此信號，也無法動，因為**摩擦力會抵消這個信號造成的驅動力**。因次我們需要手動限制邊界條件：

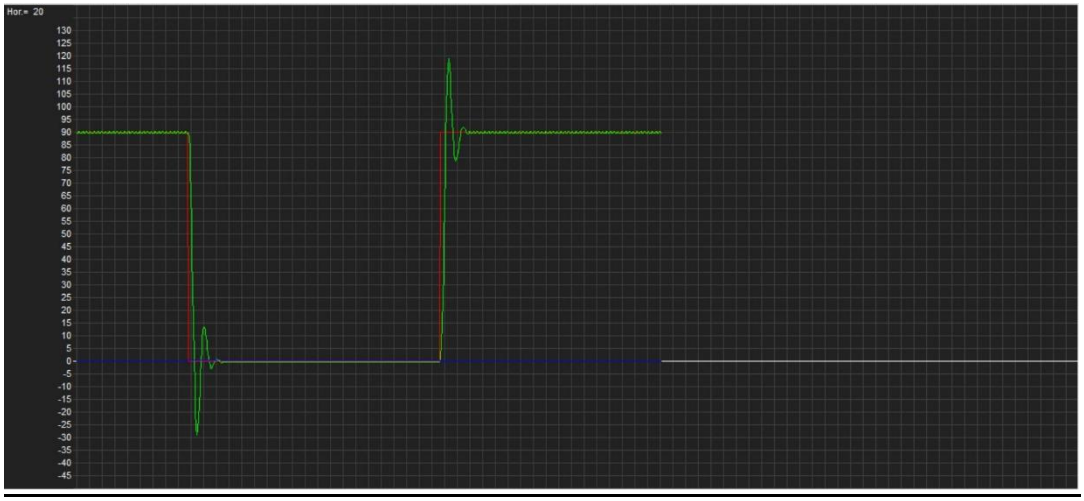
```
223   float Controller(float e) {
224       // if you want, you can add parameters by yourself
225        u = 0.35*e;
226
227       if(u<=2 && u >0) u = 2;
228       else if(u<0 && u>-2) u = -2;
229
230       return u;
231
232   }
```

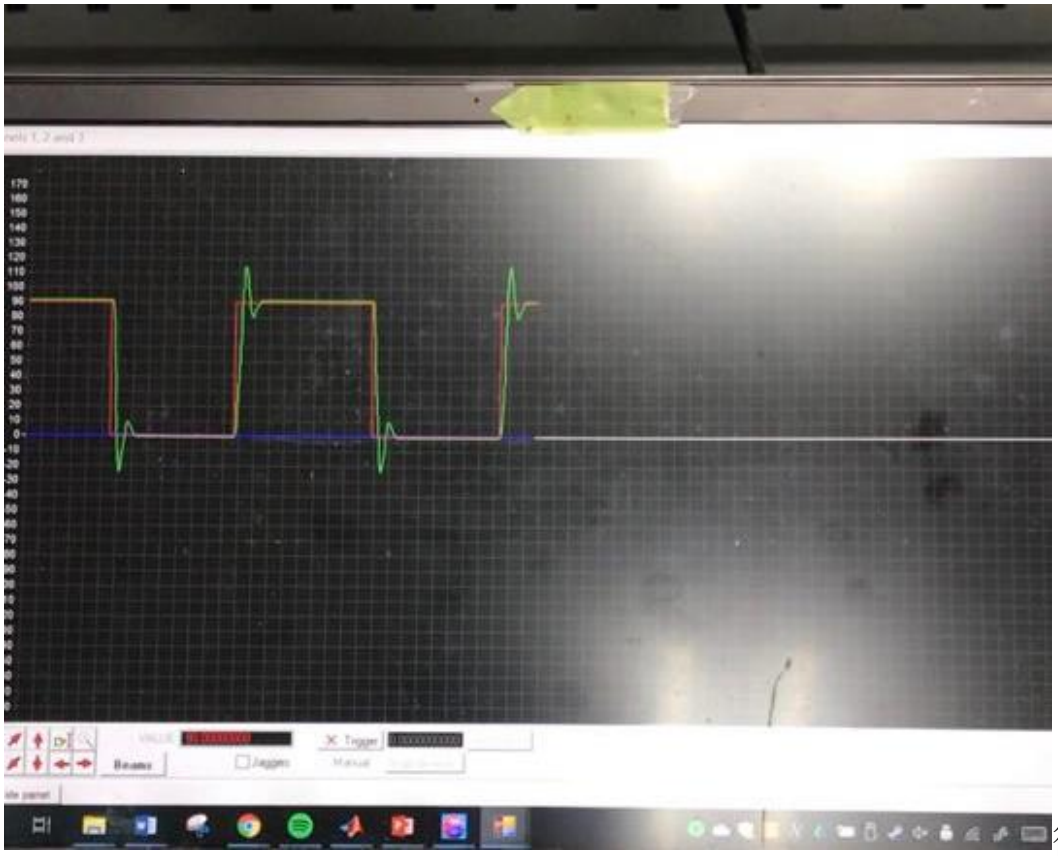
觀測波型圖可發現未有下方震盪行為(因為我們手動設定邊界條件)

解析度為：0.2 度左右(89.8-90.2)



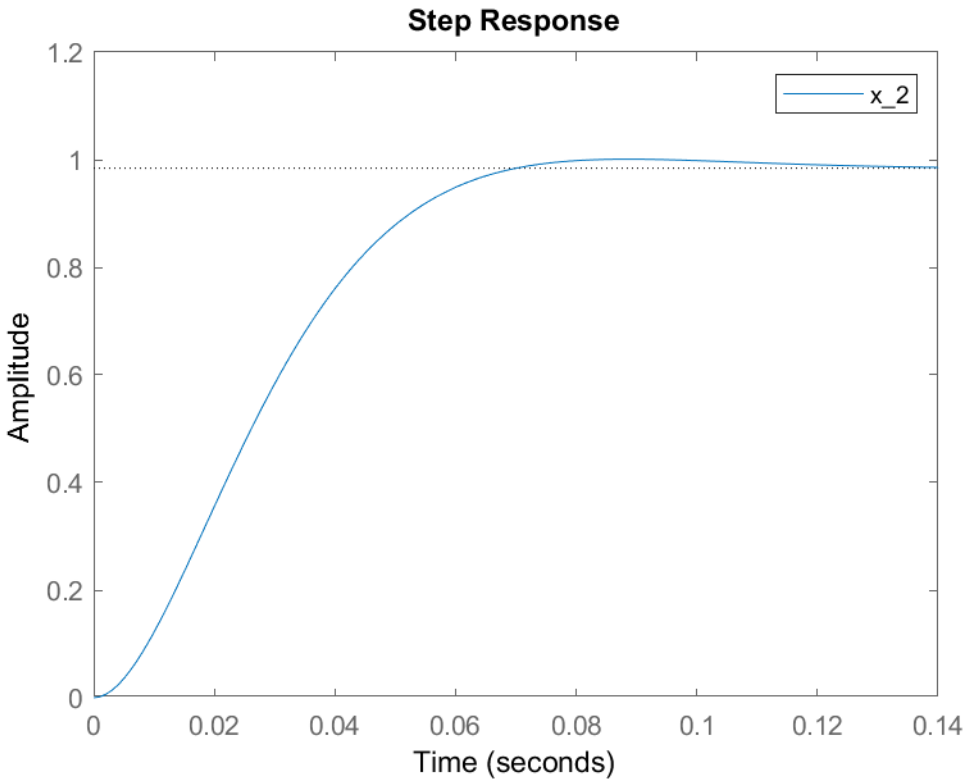
Overall response(紅色：command, 綠色：response)





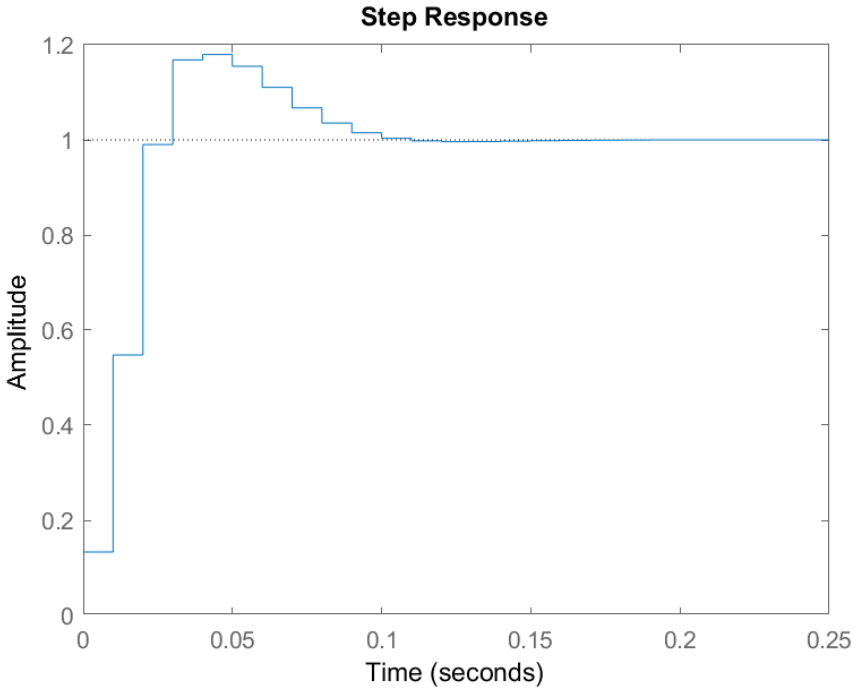
後來嘗試 PD control，從 matlab 上 trial and error 後響應效果如下：

PD controller :



將此 PD control 用 Tustin 近似後結果：

$$\frac{\mathrm{U(s)}}{\mathrm{E(s)}} = \frac{4.317z - 3.159}{z + 1}$$



轉成差分方程式： $\mathrm{u}[k] = 4.317e[k] - 3.159e[k-1] - u[k-1]$

寫入 MCU 後，馬達響應結果不甚理想。

我們最後決定再試一次 State Space Design 的 Compensator。

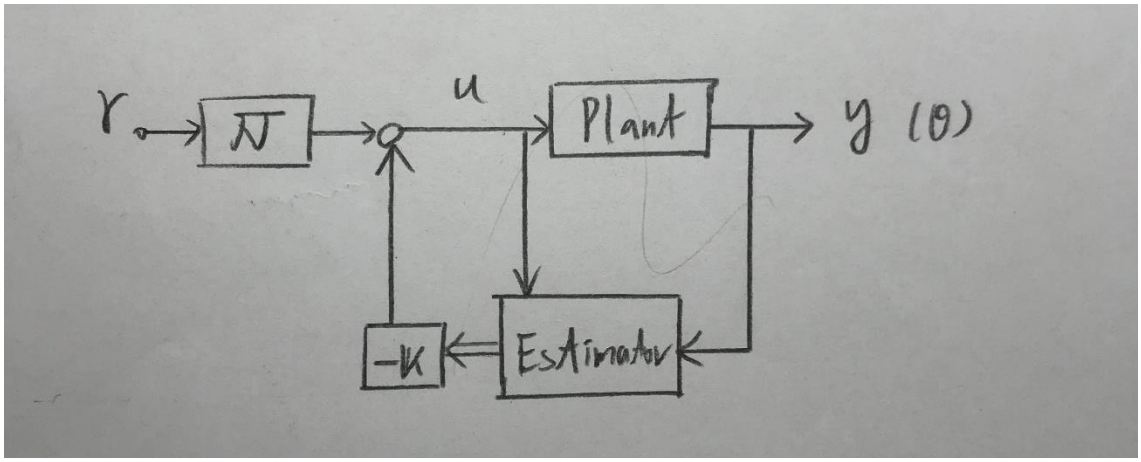
pc = [-44, -45]
pe = [-300, -310]
Nbar = 8.665

我們嘗試三種方法：

1. 引用 reference input 的 case 1 (Autonomous estimator)

Case1 讓擺置的零點和估測器集點消除，因此估測器的 input 為：回授的 y(角度)和輸入受控場的 u(電壓)。

方塊圖：



Compensator Transfer function：

$$\frac{\mathrm{U(s)}}{\mathrm{u(s)}} = \frac{-0.2414\,z\text{^}2\ -\ 0.3654\,z\ -\ 0.124}{z\text{^}2\ +\ 0.4157\,z\ +\ 0.04314}$$

$$\frac{\mathrm{U(s)}}{\mathrm{y(s)}} = \frac{-28.95\,\mathrm{z}\text{^}2\ -\ 6.322\,\mathrm{z}\ +\ 22.63}{z\text{^}2\ +\ 0.4157\,z\ +\ 0.04314}$$

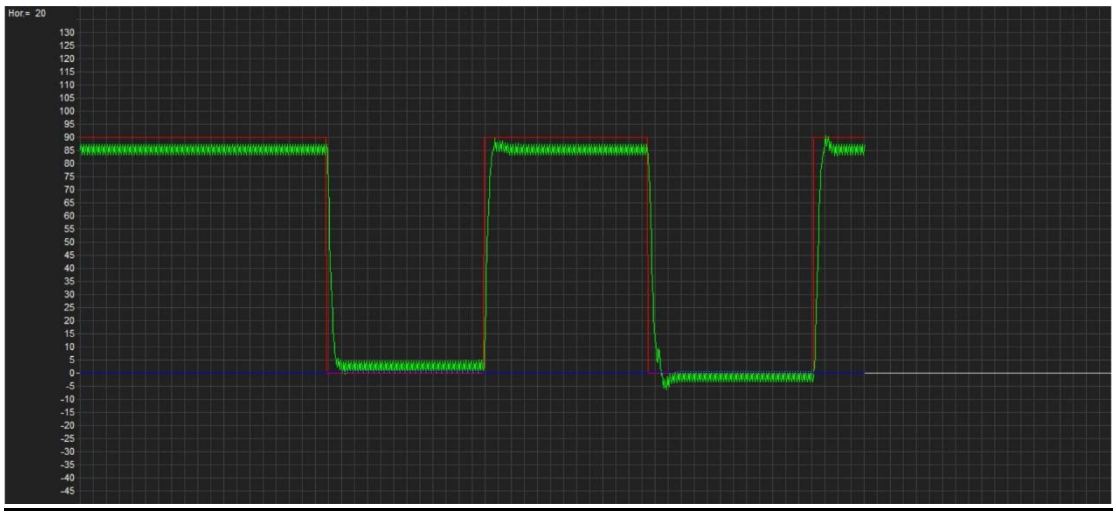
Compensator 差分方程式：

$$\mathrm{u1}[k] = -0.2414u[k] - 0.3654u[k-1] - 0.124u[k-2] - 0.4157u1[k-1] - 0.04314u1[k-2]$$

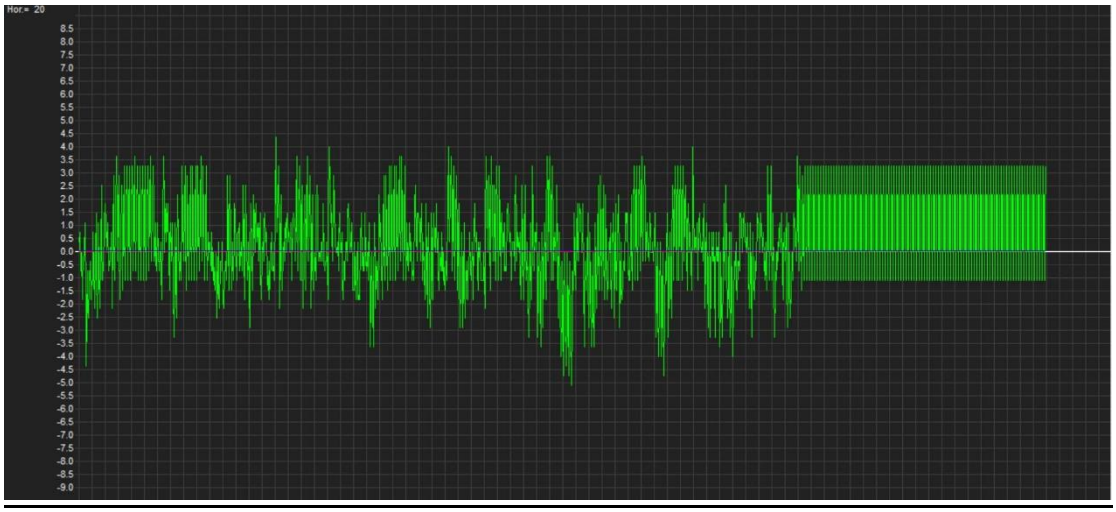
$$\mathrm{u2}[k] = -28.95y[k] - 6.322y[k-1] + 22.63y[k-2] - 0.4157u2[k-1] - 0.04314u[k-2]$$

$$\mathrm{u}[k] = u1[k] + u2[k]$$

Overall response (紅色：command, 綠色：response)



Resolution：震盪稍微大一些。

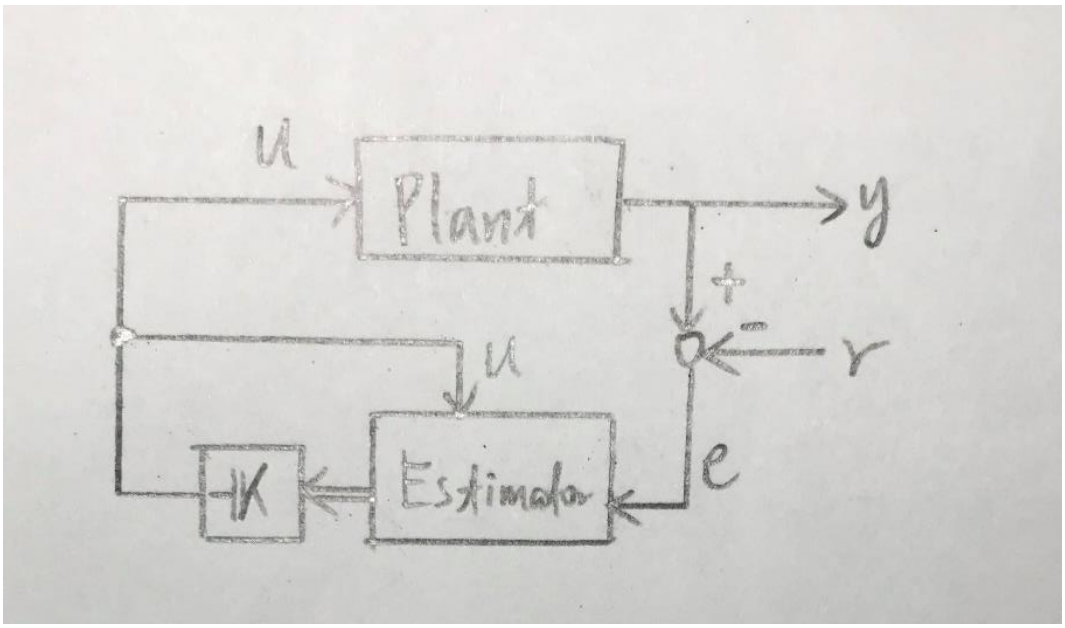


2. 引用 reference input 的 case 2 (tracking error e)

第二個 case 使 M = -L, Nbar = 0：得到較好的響應，但是震盪太大。

經討論後決定將 matlab 中使用 Tustin 近似的 sampling period T 調大 (從 0.001 變為 0.1)。響應結果如下：(誤差收斂到 0.2 個解析度)

方塊圖：



Compensator Transfer function：

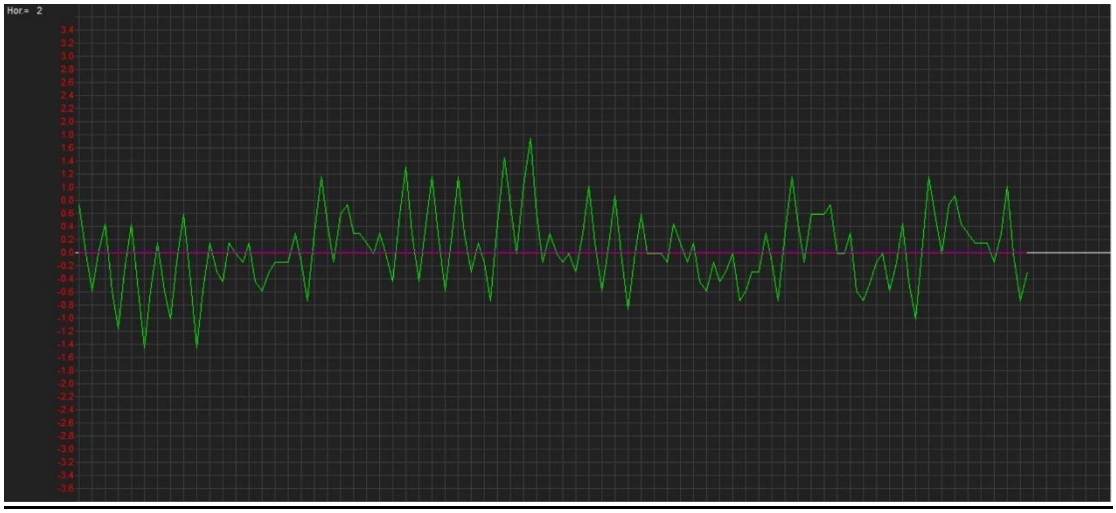
$$\frac{\mathrm{U(s)}}{\mathrm{e(s)}} = \frac{-23.32\,\mathrm{z}^2 - 5.093\,\mathrm{z} + 18.23}{\mathrm{z}^2 + 0.6292\,\mathrm{z} + 0.1346}$$

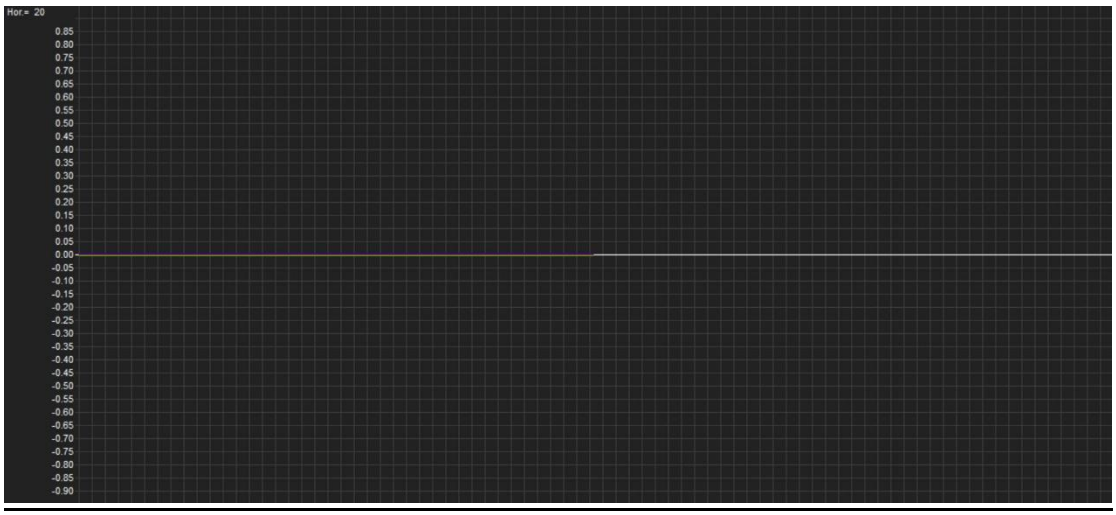
Compensator 差分方程式：

$$\mathrm{u}[k] = -23.32e[k] - 5.093e[k-1] - 18.23e[k-2] - 0.6292u[k-1] - 0.1346u[k-2]$$

Overall response (紅色：command, 綠色：response)

resolution





3. 增加積分控制：

增加積分控制後強健性有變好，但是不曉得為甚麼響應速度非常慢，也許是 Integral term 的 gain 不夠大。

因為照片不好看就沒有放上來了。

四、實驗心得

首先謝謝老師與助教們用心設計這三次的實驗，控制系統課程的實驗每次都讓我收穫滿滿。再來是感謝我的組員，皓庭與永鈞，他們都比我認真好多，組員一起討論系統設計與遭遇困難時的解決辦法，真的使這門課變得更有趣。最感動是最後有利用本學期課程核心概念：State space 狀態空間來設計補償器，雖然步驟稍微繁瑣了一些，但概念上和控一的 P、PD 相比，感覺又高級了許多。尤其是對於輸入 r 可以做前饋項(Nbar)，加上積分控制，使整個系統更加強健。

這次實驗透過 MCU 板撰寫程式，也理解到數位控制與類比控制的不同。雖然更方便也更快速，但有些小細節須注意：例如取樣時間的大小。另外也從實驗更加了解現實世界的「非線性」：例如馬達最低可動電壓不是無限小，要抵抗最大靜摩擦力的影響。再次謝謝老師、助教、組員、以及堅持下來的自己! 期待數位控制能學到更進階的控制理論與實務。