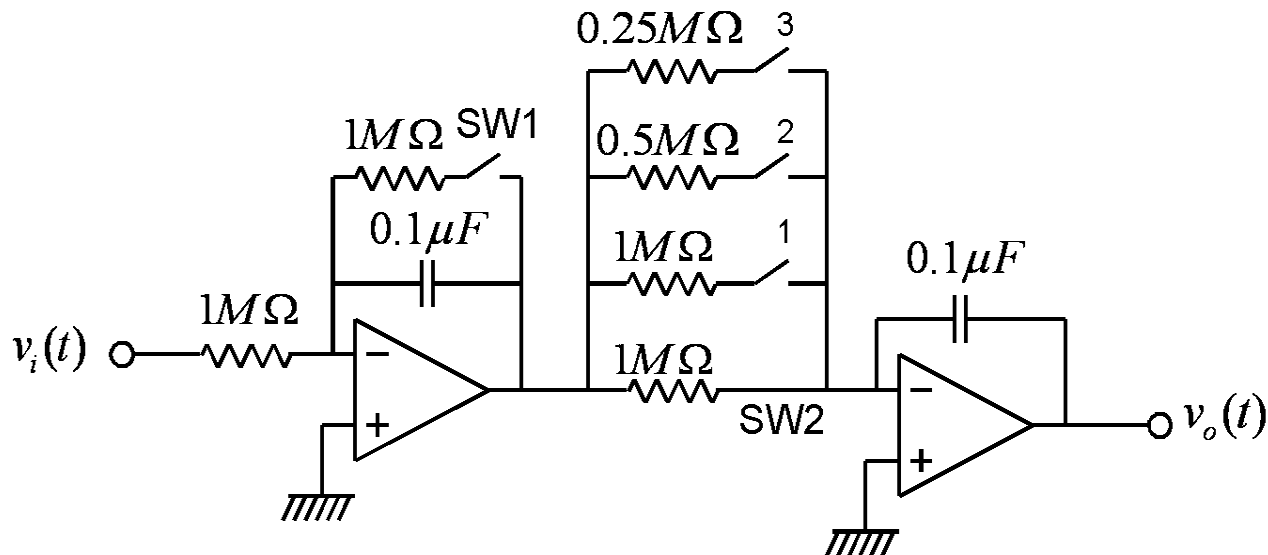


Lab 6

Full-State Feedback Controller



- 이 실험에서는 다음 그림에 주어진 **Dynamic Simulator**에 대해서 **full-state feedback controller**를 적용해 본다.



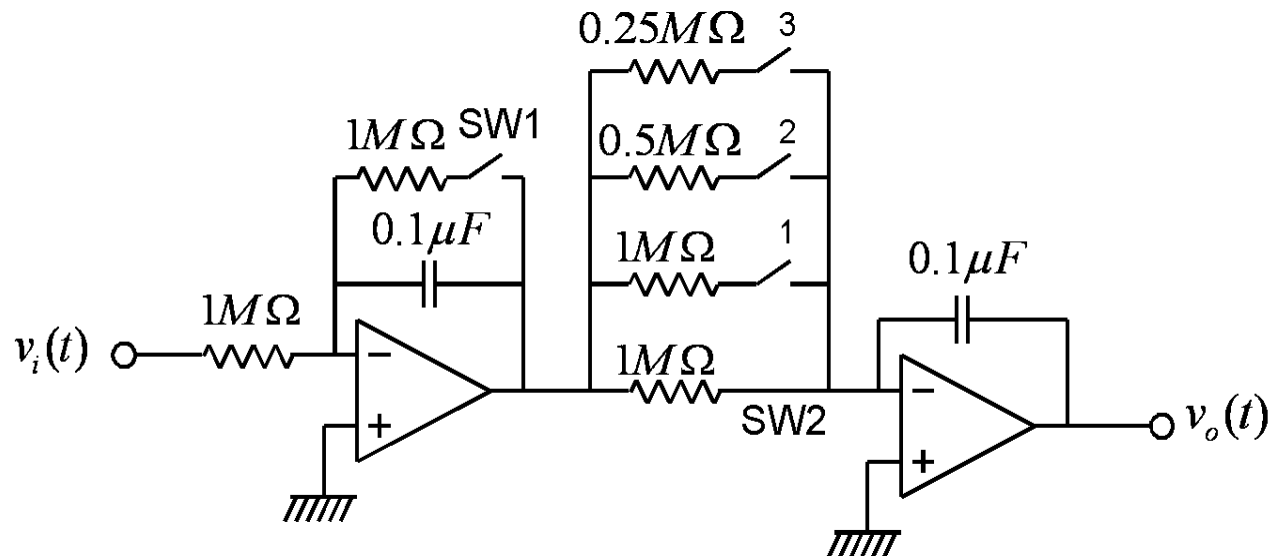
- 실험용 보드에는 시뮬레이터의 동적 특성을 변화시키기 위해서 DIP 스위치가 2개 있으며, 2점 스위치가 SW1, 3점 스위치가 SW2 이다.
- SW2에 의한 시스템 동적 특성의 변화는 다음과 같다.
 - ▶ SW2의 1번 ON: $G(s) = \frac{20}{s(0.1s+1)}$
 - ▶ SW2의 1번과 2번 모두 ON: $G(s) = \frac{40}{s(0.1s+1)}$
 - ▶ SW2의 1번, 2번과 3번 모두 ON: $G(s) = \frac{80}{s(0.1s+1)}$
- 실험 시작 전에 SW1의 1번은 ON 상태로, SW2의 모든 스위치는 OFF 상태로 놓는다. 이 스위치 상태에서 시스템의 전달 함수는 다음과 같다.

$$G(s) = \frac{10}{s(0.1s+1)}$$

- 샘플링 주파수는 1000 Hz로 한다.

Exercise 1

- 먼저 위의 시스템에 대해서 PDControl 프로젝트의 P 제어기($K_p=1.0, K_d=0$)를 실행한다. 이전과 동일하게 약간의 오버슈트가 있지만 제어가 됨을 관찰한다.
- P 제어기가 실행되는 상태에서 SW1의 1번 스위치를 OFF로 변경하고 시스템 응답을 관찰한다. 이와 같이 하면 시스템 출력이 발진하고 제어가 제대로 되지 않음을 알 수 있다.



- 위의 dynamic simulator에서 SW1의 1번을 OFF로 설정하면, dynamic simulator의 전달 함수는 다음과 같다.

$$\left(-\frac{10}{s}\right)\left(-\frac{10}{s}\right) = \frac{100}{s^2}$$

- 이 시스템에 proportional control gain 만을 적용했을 때, 전체 closed-loop system의 전달 함수는 다음 식과 같고, 모든 pole이 허수 축에 존재하여 시스템이 불안정 하게 됨을 알 수 있다.

$$\frac{K_p \frac{100}{s^2}}{1 + K_p \frac{100}{s^2}} = \frac{100K_p}{s^2 + 100K_p}$$

Exercise 2

- 이 실험에서는 dynamic simulator의 SW1의 1번을 OFF로 설정한 시스템에 대해서 full-state feedback controller를 구현하여 실험을 수행한다. Dynamic simulator의 각 OP amp의 출력 전압을 state variable로 설정하면 state equation은 다음과 같다.

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t)$$

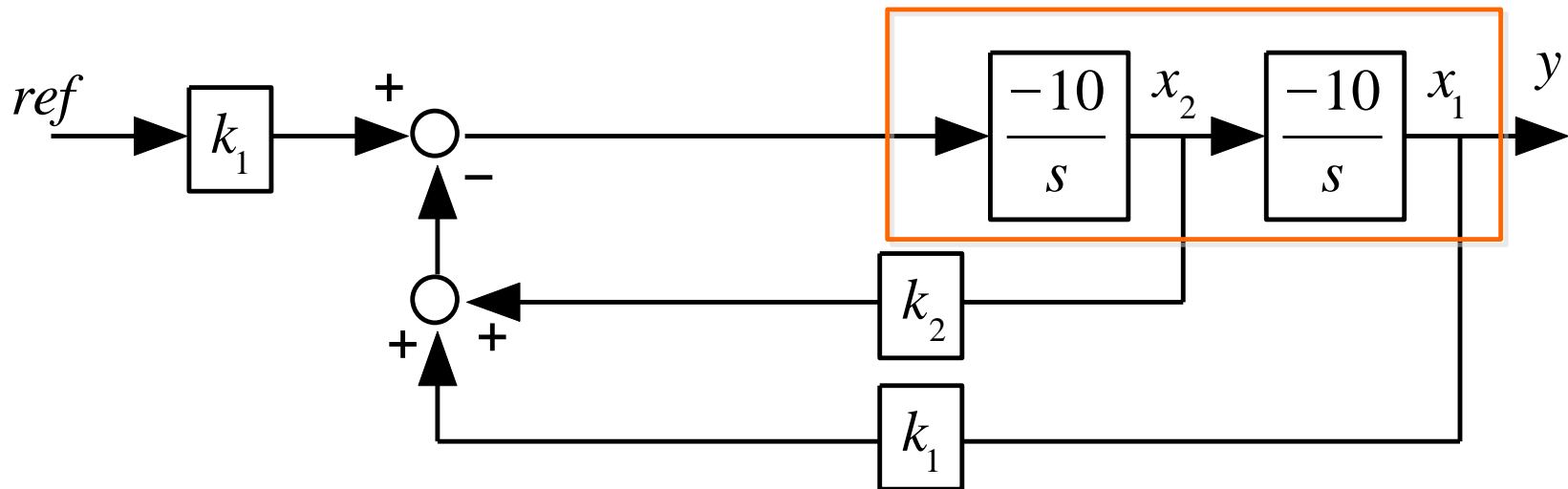
where

$$x(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix}, A = \begin{bmatrix} 0 & -10 \\ 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ -10 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$\dot{x}_1(t) = -10x_2(t)$$

$$\dot{x}_2(t) = -10u(t)$$

$$y(t) = x_1(t)$$



- 이 시스템에 대한 full-state feedback controller를 설계한다. 이 시스템에 대한 full-state feedback controller는 다음 식과 같다.

$$u(t) = -Kx(t)$$

where

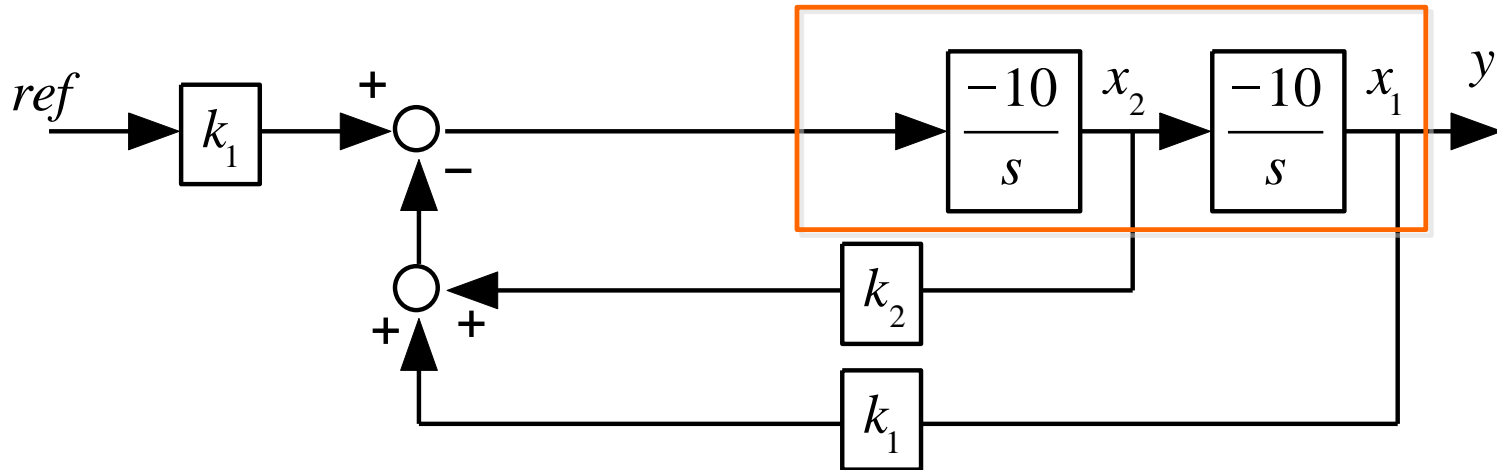
$$K = [k_1 \quad k_2]$$

- 만약 non-zero reference가 입력 되어서 step reference input을 따라 갈 수 있도록 하려 한다면, 제어기는 다음 식과 같다.

$$u(t) = k_1 ref - Kx(t)$$

where

$$K = [k_1 \quad k_2]$$



$$u(t) = k_1 ref - k_1 x_1(t) - k_2 x_2(t)$$

■ MATALB

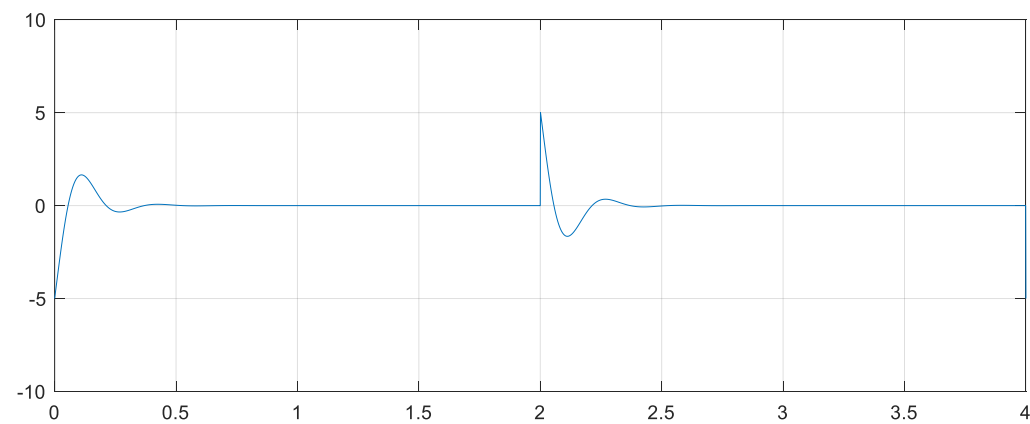
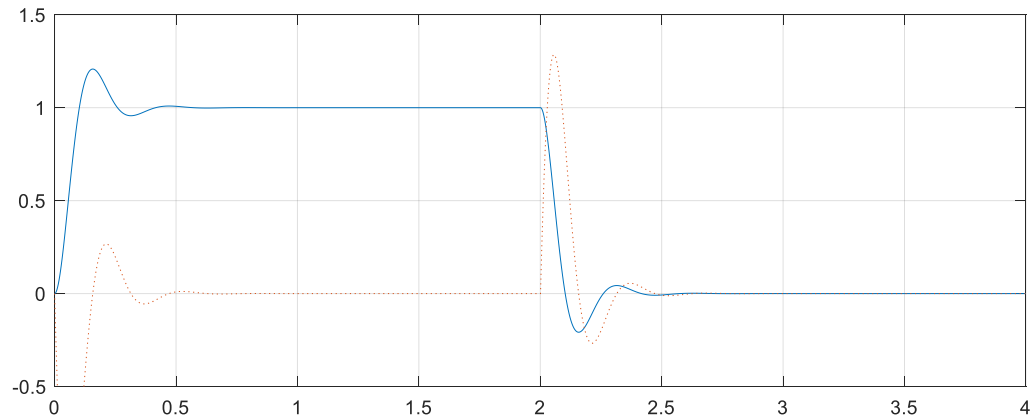
$P = [-10 + j*20 \quad -10 - j*20]$

$K = \text{acker}(A, B, P)$

$K = \text{place}(A, B, P)$

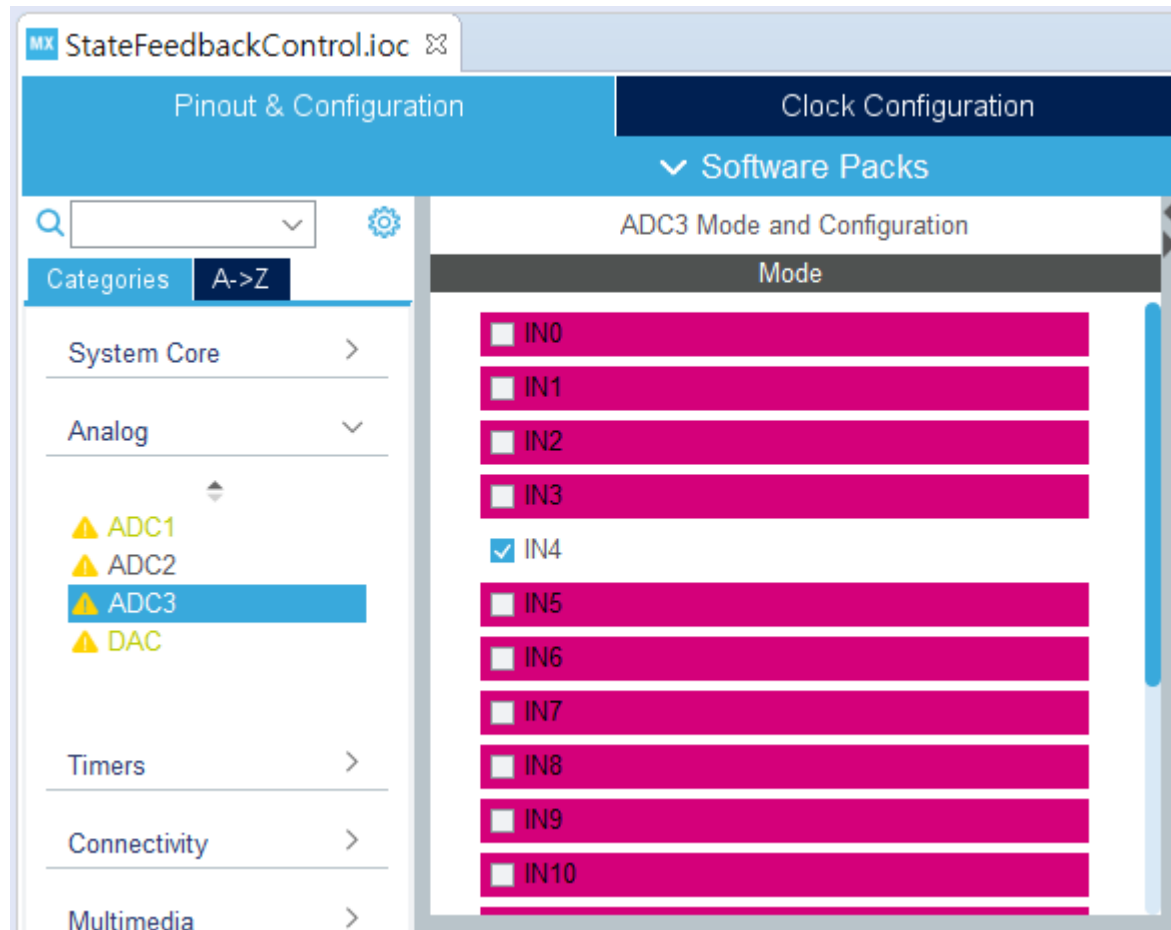
주의: P가 double pole인 경우는 place 함수는 사용할 수 없음.

Pole location: $-10 \pm j20, k_1 = 5, k_2 = -2$



- 두개의 서로 다른 pole location에 대해서 full-state feedback controller 를 설계하고, 실험 결과와 MATLAB 시뮬레이션을 비교한다.
- 두 개의 서로 다른 pole location은 서로 다른 특성을 가지도록 정한다. 예를 들면 속도가 빠른 경우와 속도가 느린 경우, 또는 오버슈트가 있는 경우와 오버슈트가 없는 경우 등으로 두 가지 서로 다른 응답 특성을 가지도록 원하는 pole location을 정한다.
- 주의할 점은 제어 신호의 범위가 $-10V \sim +10V$ 의 범위를 초과하지 않도록 한다.
- Reference input 을 2 Volt로 할 경우 제어 출력이 $-10V \sim +10V$ 의 범위를 넘을 가능성이 많으므로, reference input은 1볼트로 한다

ADC3



Code(1)

```
/* USER CODE BEGIN Includes */
#include "stdio.h"
/* USER CODE END Includes */

/* USER CODE BEGIN PV */
float K1,K2,control;
volatile int32_t x1,x2,ref,interrupt_counter,sampling_frequency,data_counter;
int16_t data[4000],data2[4000],data3[4000];
volatile uint8_t data_flag=0,data_done=0;
/* USER CODE END PV */

/* USER CODE BEGIN 0 */
#ifdef __GNUC__
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */
PUTCHAR_PROTOTYPE
{
    HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 0xFFFF);
    return ch;
}
/* USER CODE END 0 */
```

Code(2)

```
/* USER CODE BEGIN 1 */
    K1=5.0;K2=-2.0;
    sampling_frequency=1000;
/* USER CODE END 1

/* USER CODE BEGIN WHILE */
while (1)
{
    if(data_done == 1) {
        for (int i=0; i < sampling_frequency*4 ;i++){
            printf("%d %d %d %d\r\n",i,data[i],data2[i],data3[i]);
        }
        data_flag=0;
        data_done=0;
        HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
    }
/* USER CODE END WHILE */
```

Code(3)

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);
    data_flag=1;
}
/* USER CODE END 4 */

/* USER CODE BEGIN Callback 0 */
    int32_t da_value,ad_value,sum;
    if (htim->Instance == TIM10) {
        sum=0;
        for (int i=0; i<20 ; i++) {
            HAL_ADC_Start(&hadc1);
            if (HAL_ADC_PollForConversion(&hadc1, 10000) == HAL_OK) {
                ad_value = HAL_ADC_GetValue(&hadc1);
                sum += ad_value;
            }
        }
        x1 = sum/20 - 2048;
```


Code(4)

```
sum=0;
for (int i=0; i<20 ; i++) {
    HAL_ADC_Start(&hadc3);
    if (HAL_ADC_PollForConversion(&hadc3, 10000) == HAL_OK) {
        ad_value = HAL_ADC_GetValue(&hadc3);
        sum += ad_value;
    }
}
x2 = sum/20 - 2048;
interrupt_counter++;
if (interrupt_counter >= sampling_frequency*4) {
    interrupt_counter=0;
    if (data_flag==1) {
        data_counter=0;
        data_flag=2;
    }
    ref=205;
}
if (interrupt_counter >= sampling_frequency*2) {
    ref=0;
}
```

Code(5)

```
if (data_flag==2) {
    if (data_counter<sampling_frequency*4) {
        data[data_counter]=(int16_t)x1;
        data2[data_counter]=(int16_t)x2;
        data3[data_counter]=(int16_t)control;
        data_counter++;
    }
    else {
        data_done=1;
    }
}
control = K1*(float)(ref-x1)-K2*(float)x2;
    if (control > 2047) control = 2047;
    if (control < -2048) control = -2048;
    da_value = control + 2048;
    HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R,
(uint32_t)(da_value));
}
/* USER CODE END Callback 0 */
```