

## 9. 상태 변수 방정식을 이용한 제어 시스템 설계

### 9.5 실험 실습: Lab 9

#### 9.5.1 아날로그 다이내믹 시뮬레이터에 대한 상태 변수 피드백 제어기

이 실험에서는 그림 9.33과 같이 2개의 적분 회로가 직렬로 연결된 아날로그 다이내믹 시뮬레이터에 대한 상태 변수 피드백 제어기를 적용해 본다.

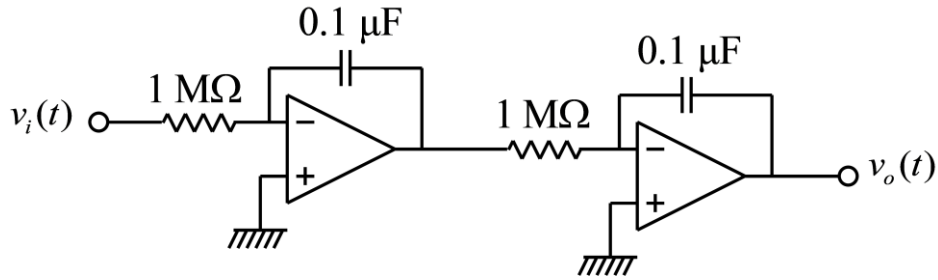


그림 9-33 아날로그 다이내믹 시뮬레이터

각 적분기의 출력을 상태 변수로 정의한 후, 상태 변수 피드백 제어기를 적용하면 그림 9-34와 같다.

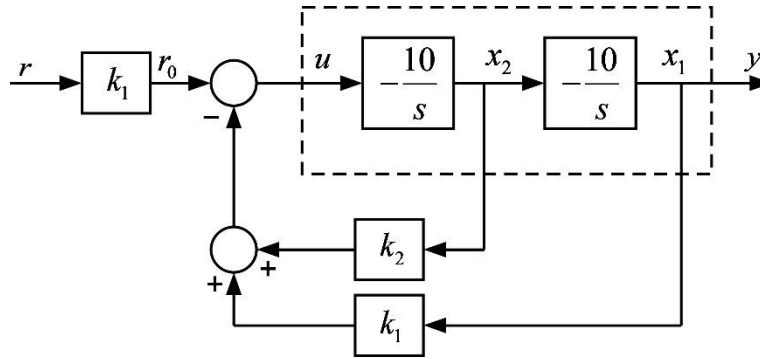


그림 9-34 아날로그 다이내믹 시뮬레이터에 대한 상태 변수 피드백 제어기

$X_1(s)$ ,  $X_2(s)$ ,  $U(s)$ 를 각각  $x_1(t)$ ,  $x_2(t)$ ,  $u(t)$ 의 라플라스 변환이라고 정의하면 다음과 같은 관계식을 얻을 수 있다.

$$\frac{X_1(s)}{X_2(s)} = -\frac{10}{s} \quad (9.149)$$

$$\frac{X_2(s)}{U(s)} = -\frac{10}{s} \quad (9.150)$$

위의 식에서 다음의 상태 변수 방정식을 얻을 수 있다.

$$\dot{x}_1(t) = -10x_2(t) \quad (9.151)$$

$$\dot{x}_2(t) = -10u(t) \quad (9.152)$$

위의 식은 다음과 같은 벡터 형태의 상태 변수 방정식으로 나타낼 수 있다.

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & -10 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ -10 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \quad (9.153)$$

폐루프 시스템의 극점의 위치가  $-10 \pm j20$  이 되는 상태 변수 피드백 제어를 설계한다. 애커만의 공식을 적용하면 상태 변수 피드백 이득 값은 다음과 같다.

$$k_1 = 5, k_2 = -2 \quad (9.154)$$

아래의 MATLAB 코드를 실행하면 상태 변수 피드백 제어 시스템의 시뮬레이션 결과를 얻을 수 있다.

코드 9.1

```
a=[0 -10;0 0]
b=[0;-10]
c=[1 0]
d=0
p=[-10+j*20 -10-j*20]
k=acker(a,b,p)
a1=a-b*k
h=ss(a1,b,c,d)
t = 0:0.001:4; % vector of time samples
ref = (rem(t,4)<=2)*k(1); % square wave values
%ref=ones(size(t))*k(1); % unit step
[ys ts xs]=lsim(h,ref,t,[0;0]);
% plot states
figure(1)
subplot(2,1,1);
plot(ts,xs(:,1),'-',ts,xs(:,2),'-');axis([0 4 -0.5 1.5])
grid on
ylabel('x1,x2');
% plot control
subplot(2,1,2)
plot(ts,ref'-k(1)*xs(:,1)-k(2)*xs(:,2));axis([0 4 -10 10])
grid on
ylabel('u');xlabel('time(sec'))
```

Figure 9.35 shows the MATLAB simulation responses, where the state variables and control signals are shown. 그림 9-35는 MATLAB 시뮬레이션 결과를 보여주며, 그림에서 상태 변수와 제어 신호의 그래프를 볼 수 있다.

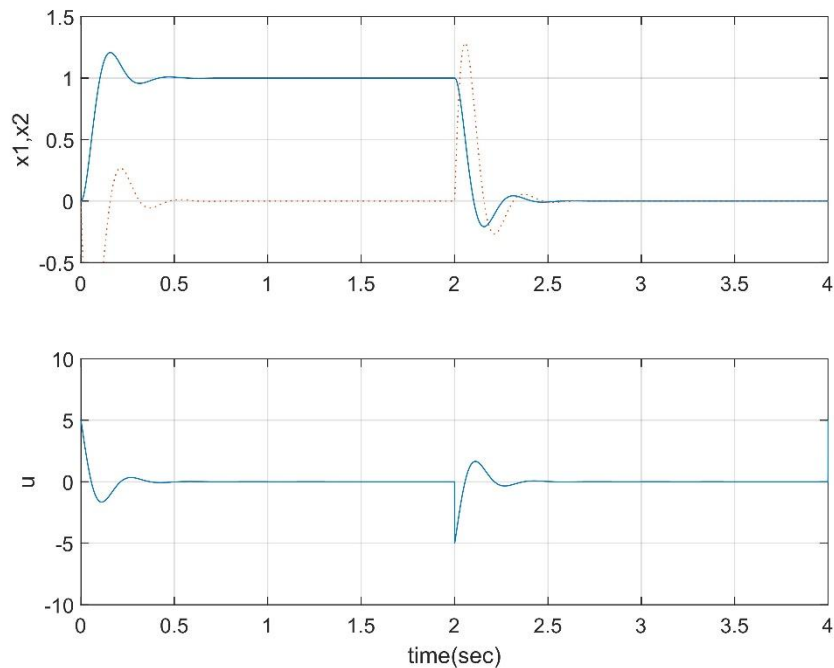


그림 9-35 MATLAB 시뮬레이션 응답

상태 변수 피드백 제어를 아날로그 다이내믹 시뮬레이터에 적용하기 위한 코드는 다음과 같다. 8장의 8.8절의 코드와 비슷하며 코드 8.2, 코드 8.3, 코드 8.4, 코드 8.5를 각각 코드 9.2, 코드 9.3, 코드 9.4, 코드 9.5로 교체한다.

코드 9.2

```
/* 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 */
```

코드 9.3

```
/* USER CODE BEGIN 1 */
    K1=5;K2=-2;
```

```
sampling_frequency=1000;
/* USER CODE END 1 */
```

#### 코드 9.4

```
/* 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(GPIOG, GPIO_PIN_14);
    }
}
/* USER CODE END WHILE */
```

#### 코드 9.5

```
/* 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;
    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;
}
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 */

```

그림 9-36은 상태 변수의 응답과 제어 신호를 보여준다.

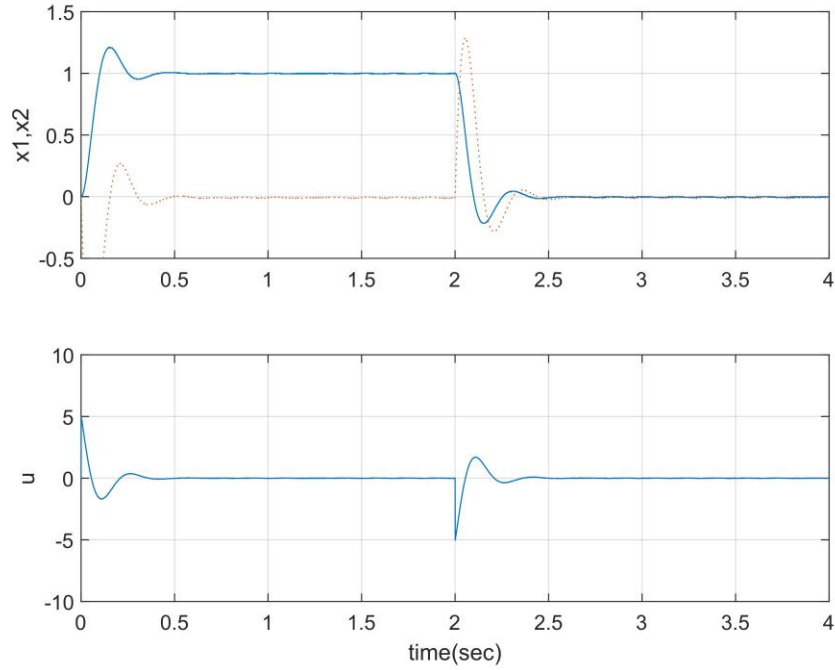


그림 9-36 상태 변수의 응답과 제어 신호

### 실험 연습문제 9.1

페루프 시스템이 좌평면의 2중 실수 극점을 가지도록 상태 변수 피드백 제어기의 이득을 구해서 위의 실험을 반복한다.

### 9.5.2 아날로그 다이내믹 시뮬레이터에 대한 상태 변수 추정기 기반 제어기

9.5.1절의 아날로그 다이내믹 시뮬레이터에 대해서 상태 변수 추정기 기반 제어기를 설계하고 구현해 본다. 출력 신호는 상태 변수  $x_1$  이고 상태 변수  $x_2$  는 측정할 수 없다고 가정한다. 상태 변수 추정기의 극점은  $-50 \pm j100$  으로 선택하면 상태 변수 추정기의 이득은 다음과 같다.

$$l_1 = 100, l_2 = -1250 \quad (9.155)$$

상태 변수 추정기의 식은 다음과 같다.

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + L(Cx(t) - C\hat{x}(t)) \quad (9.156)$$

위의 식에서 변수와 계수는 다음과 같다.

$$\hat{x}(t) = \begin{bmatrix} \hat{x}_1(t) \\ \hat{x}_2(t) \end{bmatrix}, A = \begin{bmatrix} 0 & -10 \\ 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ -10 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}, L = \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \quad (9.157)$$

제어 신호의 식은 다음과 같다.

$$u(t) = k_1 ref - K\hat{x}(t) \quad (9.158)$$

상태 변수 추정기와 제어 신호의 식은 다음 식과 같이 다시 쓸 수 있다.

$$\begin{aligned} \dot{\hat{x}}_1(t) &= -10\hat{x}_2(t) + l_1(y(t) - \hat{x}_1(t)) \\ \dot{\hat{x}}_2(t) &= -10u(t) + l_2(y(t) - \hat{x}_1(t)) = -10(k_1 ref - k_1\hat{x}_1(t) - k_2\hat{x}_2(t)) + l_2(y(t) - \hat{x}_1(t)) \\ u(t) &= k_1 ref - k_1\hat{x}_1(t) - k_2\hat{x}_2(t) \end{aligned} \quad (9.159)$$

Euler의 적분 근사식을 이용하면 위의 식에 대한 디지털 구현식은 다음과 같다. 참고로 이 방법은 앞의 4장 4.7.1절의 2차 시스템 실시간 시뮬레이션에서 사용한 방법과 비슷한 방법이다.

$$\begin{aligned} \hat{x}_1(t) &= \hat{x}_1(t - \Delta t) + \Delta t [-10\hat{x}_2(t - \Delta t) + l_1(y(t - \Delta t) - \hat{x}_1(t - \Delta t))] \\ \hat{x}_2(t) &= \hat{x}_2(t - \Delta t) + \Delta t [-10(k_1 ref - k_1\hat{x}_1(t - \Delta t) - k_2\hat{x}_2(t - \Delta t)) + l_2(y(t - \Delta t) - \hat{x}_1(t - \Delta t))] \\ u(t) &= k_1 ref - k_1\hat{x}_1(t) - k_2\hat{x}_2(t) \end{aligned} \quad (9.160)$$

아래는 위의 제어기를 적용한 시스템의 MATLAB 시뮬레이션 코드이다.

코드 9.6

```
a=[0 -10;0 0]
b=[0;-10]
c=[1 0]
d=0
p=[-10+20*j -10-20*j]
pe=5*p
k=acker(a,b,p)
l=acker(a',c',pe)
a1=[a -b*k;l'*c a-b*k-l'*c];
b1=[b;b];
c1=[c 0 0];
d1=0;
h=ss(a1,b1,c1,d1);
t = 0:0.001:4; % vector of time samples
ref = (rem(t,4)<=2)*k(1); % square wave values
%ref=ones(size(t))*k(1); % unit step
[ys ts xs]=lsim(h,ref,t,[0;0;0;0]);
% plot states
```

```

subplot(2,1,1);
plot(ts,xs(:,1),'-',ts,xs(:,2),':');axis([0 4 -0.5 1.5])
ylabel('x1,x2');xlabel('time(sec)');
grid on
% plot state estimates
subplot(2,1,2)
plot(ts,xs(:,3),'-',ts,xs(:,4),':');axis([0 4 -0.5 1.5])
ylabel('x1hat,x2hat');xlabel('time(sec)');
grid on

```

그림 9-37 은 위의 MATLAB 시뮬레이션 결과를 보여준다. 아래의 그림에서 첫째 그래프는 상태 변수를 두번째 그래프는 상태 변수 추정기의 변수를 보여준다.

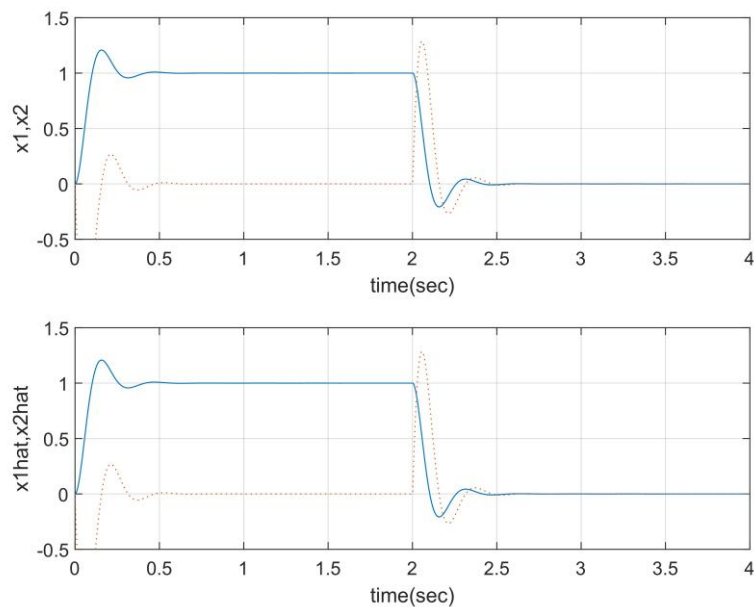


그림 9-37 MATLAB 시뮬레이션 응답

실습용 보드를 위한 STMCubeIDE 설정은 9.5.1 절과 같으며, 코드 9.2, 코드 9.3, 코드 9.4, 코드 9.5 를 아래의 코드 9.7, 코드 9.8, 코드 9.9, 코드 9.10 으로 교체한다.

코드 9.7

```

/* USER CODE BEGIN PV */
float x1hat,x1hat_old,x2hat,x2hat_old;
float delt;
float K1,K2,L1,L2,control;
volatile int32_t
x1,x1_old,x2,ref,interrupt_counter,sampling_frequency,data_counter;
int16_t data[4000],data2[4000],data3[4000],data4[4000];
volatile uint8_t data_flag=0,data_done=0;
/* USER CODE END PV */

```



#### 코드 9.8

```
/* USER CODE BEGIN 1 */
K1=5;K2=-2;
L1=100;L2=-1250;
sampling_frequency=1000;
delt=1/(float)(sampling_frequency);
/* USER CODE END 1 */
```

#### 코드 9.9

```
/* USER CODE BEGIN WHILE */
while (1)
{
    if(data_done == 1) {
        for (int i=0; i < sampling_frequency*4 ;i++){

printf("%d %d %d %d %d\r\n",i,data[i],data2[i],data3[i],data4[i]);
        }
        data_flag=0;
        data_done=0;
        HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_14);
    }
}
/* USER CODE END WHILE */
```

#### 코드 9.10

```
/* 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;
    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;
}
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)x1hat;
        data4[data_counter]=(int16_t)x2hat;
        data_counter++;
    }
    else {
        data_done=1;
    }
}
x1hat=x1hat_old+delt*(-10.0*x2hat_old+L1*(x1_old-x1hat_old));
x2hat=x2hat_old+delt*(-10.0*(K1*ref-K1*x1hat_old-
K2*x2hat_old)+L2*(x1_old-x1hat_old));
control = K1*(float)(ref-x1hat)-K2*(float)x2hat;
x1_old=x1;
x1hat_old=x1hat;
x2hat_old=x2hat;
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 */

```

그림 9-38 은 아날로그 다이내믹 시뮬레이터의 상태 변수와 상태 변수 추정기의 변수를 보여준다.

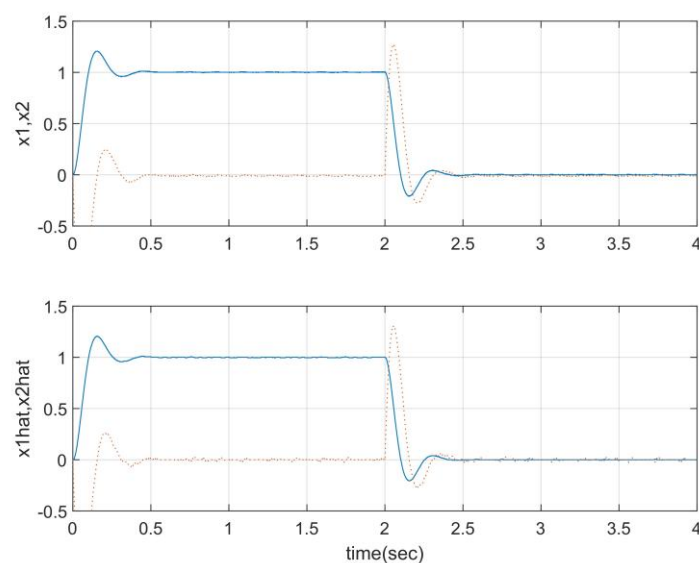


그림 9-38 아날로그 다이내믹 시뮬레이터의 상태 변수와 상태 변수 추정기의 변수

## 실험 연습문제 9.2

위의 상태 변수 추정기 기반 제어를 실험을 반복하되, 상태 변수 피드백을 적용한 페루프 시스템의 극점이 실수 2 중근이 되도록 제어 이득 값을 결정한다. 상태 변수 추정기의 이득은 위와 동일하게 정한다.

### 9.5.3 자기 부상 장치에 대한 상태 변수 추정기 기반 제어기

앞의 7장의 7.7.2절에서는 자기 부상 장치에 대한 진상 제어를 설계하고 구현하는 실험을 진행했다. 이 절에서는 자기 부상 장치에 대해서 상태 변수 추정기 기반 제어를 설계하고 구현한다. 자기 부상 장치는 비선형 특성을 가지고 있지만, 7.7.2절에서 사용한 선형화된 모델을 사용해서 제어를 설계한다. 아래는 선형화된 상태 변수 방정식이다. 편의상 7.7.2절의 상태 변수 앞의  $\Delta$ 는 생략한다.

$$\begin{aligned}\dot{x}_1(t) &= x_2(t) \\ \dot{x}_2(t) &= \frac{2g}{Y_0} x_1(t) - \frac{0.3\gamma g}{I_0} u(t)\end{aligned}\quad (9.1)$$

위의 시스템에 대한 상태 변수 추정기의 식은 다음과 같다.

$$\begin{aligned}\dot{\hat{x}}_1(t) &= \hat{x}_2(t) + l_1 (x_1(t) - \hat{x}_1(t)) \\ \dot{\hat{x}}_2(t) &= \frac{2g}{Y_0} \hat{x}_1(t) - \frac{0.3\gamma g}{I_0} u(t) + l_2 (x_1(t) - \hat{x}_1(t))\end{aligned}\quad (9.2)$$

상태 변수 추정기 기반 제어기의 식은 다음과 같다.

$$u(t) = k_1 ref - k_1 \hat{x}_1(t) - k_2 \hat{x}_2(t) \quad (9.3)$$

Euler의 방법 중 포워드 사각형 규칙을 사용해서 위의 상태 변수 추정기의 식을 아래와 같은 디지털 식으로 변환한다.

$$\begin{aligned}\hat{x}_1(t) &= \hat{x}_1(t - \Delta t) + \Delta t (\hat{x}_2(t - \Delta t) + l_1 (x_1(t - \Delta t) - \hat{x}_1(t - \Delta t))) \\ \hat{x}_2(t) &= \hat{x}_2(t - \Delta t) + \Delta t \left( \frac{2g}{Y_0} \hat{x}_1(t - \Delta t) - \frac{0.3\gamma g}{I_0} u(t - \Delta t) + l_2 (x_1(t - \Delta t) - \hat{x}_1(t - \Delta t)) \right)\end{aligned}\quad (9.4)$$

먼저 상태 변수 피드백의 극점은  $-15 \pm j40$ 으로, 상태 변수 추정기의 극점은  $-60 \pm j160$ 으로 정한다. 상태 변수 피드백 이득과 상태 변수 추정기의 이득은 다음과 같다.

$$k_1 = -2.23, k_2 = -0.025 \quad (9.5)$$

$$l_1 = 120, l_2 = 30052 \quad (9.6)$$

실습용 보드를 위한 STMCubeIDE 설정은 9.5.2 결과 같으며, 코드 9.7, 코드 9.8, 코드 9.9, 코드 9.10 을 아래의 코드 9.11, 코드 9.12, 코드 9.13, 코드 9.14 로 교체한다.

#### 코드 9.11

```
/* USER CODE BEGIN PV */
long interrupt_counter;
long ref,data_counter;
short data[8001];
char data_flag,data_done;
long sampling_frequency;
int y,oldy;
float delt,control;
float k1,k2;
float Gamma=333.3;float g=9.8;float I0=0.817;float X0=0.023;
float x1hat,x1hat_old,x2hat,x2hat_old;
float l1,l2;
/* USER CODE END PV */
```

#### 코드 9.12

```
/* USER CODE BEGIN 1 */
sampling_frequency= 1000;
delt=1/(float)sampling_frequency;
k1=-2.23;k2=-0.025;
l1=120;l2=30052;
interrupt_counter=0;
data_counter=0;
data_flag=0;
data_done=0;
/* USER CODE END 1 */
```

#### 코드 9.13

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

#### 코드 9.14

```
/* 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;
        }
    }
    y = 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=-102;
    }
    if (interrupt_counter >= sampling_frequency*2) {
        ref=-205;
    }
    if (data_flag==2) {
        if (data_counter<=sampling_frequency*4) {
            data[data_counter++]=(int16_t)y;
        }
        else {
            data_done=1;
        }
    }
    x1hat=x1hat_old+delt*(x2hat_old+l1*(oldy-x1hat_old));
    x2hat=x2hat_old+delt*((2*g/X0)*x1hat_old-
(0.3*Gamma*g/I0)*control+l2*(oldy-x1hat_old));
    control=k1*(ref-x1hat)-k2*(x2hat);
    x1hat_old=x1hat;
    x2hat_old=x2hat;
    oldy=y;
    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 */

```

그림 9-39 는 상태 변수 추정기 기반 제어기가 적용된 계단 응답을 보여준다. 제어기의 극점을 복소수로 선택을 했으므로 응답은 부족 감쇠 응답을 보여준다.

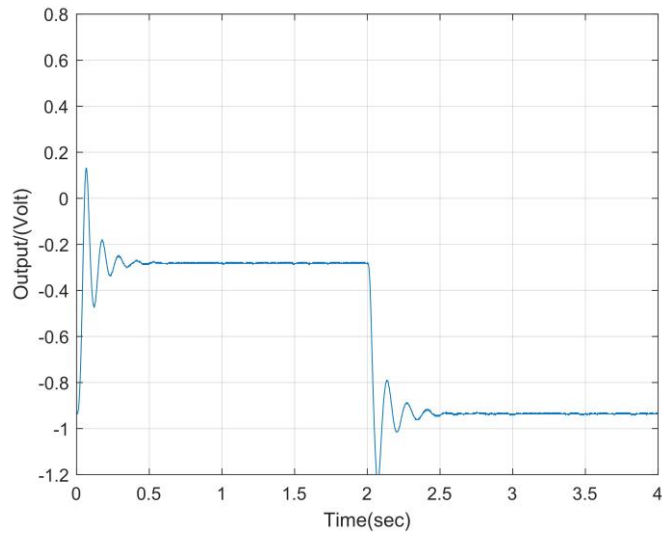


그림 9-39 복소수 극점에 대한 계단 응답

비교를 위해서 상태 변수 피드백 극점을  $-30$ 의 이중 극점으로 선택을 한다. 상태 변수 추정기의 극점은 위와 동일하게 정한다. 상태 변수 피드백 이득과 상태 변수 추정기의 이득은 다음과 같다.

$$k_1 = -1.46, k_2 = -0.05 \quad (9.7)$$

$$l_1 = 120, l_2 = 30052 \quad (9.8)$$

그림 9-40은 실수 극점에 대한 계단 응답을 보여준다. 실수 극점이므로 오버슈트 거의 없는 것을 볼 수 있다. 한편 그림 9-40의 정상 상태의 값이 그림 9-39의 정상 상태의 값보다 큰 것을 볼 수 있다. 이 제어기에서는 중력에 의한 정상 상태 오차가 발생하며 정상 상태 오차의 크기는 제어기 이득  $k_1$ 과 관련이 있다. 그림 9-40의 경우에는 그림 9-39의 경우보다  $k_1$ 의 값이 작으므로 중력에 의해서 쇠구슬은 정상 상태에서 더 아래쪽에 있게된다. 출력 값이 전자석과 쇠구슬 사이의 거리이므로 전자석과 쇠구슬 사이의 거리가 멀어지면 출력 값은 더 크게 나타난다.

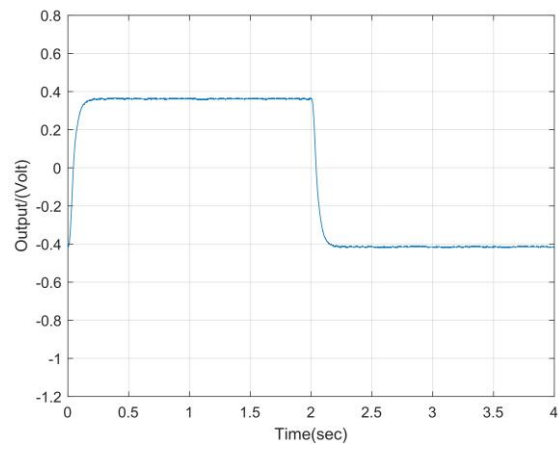


그림 9-40 실수 극점에 대한 계단 응답