ELEC25400: Control Systems Lab Report

Jialeng Guo
{el17jg}@leeds.ac.uk

Report Task

Task 1 Answer

LPC1768 Mbed micro-controller		controller K(s)
Compass, magnetometer, servo motor and battery pack		plant G(s)
The direction of compass pointer		output signal Y(s)
PWC signal	***is***	actuation signal U(s)
The value of magnetic North		reference value R(s)
The difference between reference value and measured output signal		control error E(s)
The angle measurement formula provided by compass module		measurement H(s)
The position measured by the angle measurement / the result value		measured output signal Y^(s)

Task 2 Answers

2.5) **i) Code**

```
1.  float updatePI(float pos, float target, float dt)
2.  { // add your controller here
3.      float U_PI = 0.0;
4.      E = target - pos;
5.      U_PI = K * (E);
6.      U_hat = U_PI + U_AW;
7.      U = saturation(U_hat);
8.      return U;
9.  }
```

ii) Controller Parameters

K = 0; Ti = 10000.

2.6) **i) Code**

```
1.  float updatePI(float pos, float target, float dt)
2.  { // add your controller here
3.      float U_PI = 0.0;
4.      E = target - pos;
5.      U_PI = K * (E);
6.      U_hat = U_PI + U_AW;
```

```
7.      U = saturation(U_hat);
8.      return U;
9.  }
```

ii) Controller Parameters

K = 0.1, Ti = 10000;

K = 2, Ti = 10000;

K = 3, Ti = 10000;

K = 3.149, Ti = 10000;

K = 3.15, Ti = 10000;

K = 3.151, Ti = 10000;

K = 3.2, Ti = 10000;

K = 3.25, Ti = 10000;

K = 3.3, Ti = 10000;

K = 3.5, Ti = 10000;

K = 4, Ti = 10000;

K = 5, Ti = 10000;

K = 6, Ti = 10000;

K = 6.6, Ti = 10000;

K = 6.5, Ti = 10000;

K = 7, Ti = 10000.

2.7) **i) Code**

```
1.  float updatePI(float pos, float target, float dt)
2.  { // add your controller here
3.      float U_PI = 0.0;
4.      E = target - pos;
5.      U_PI = K * (E);
6.      U_hat = U_PI + U_AW;
7.      U = saturation(U_hat);
8.      return U;
9.  }
```

ii) Controller Parameters

K = 0.1, Ti = 10000;

K = 2, Ti = 10000;

K = 3, Ti = 10000;

K = 3.149, Ti = 10000;

K = 3.15, Ti = 10000;

K = 3.151, Ti = 10000;

K = 3.2, Ti = 10000;

K = 3.25, Ti = 10000;

K = 3.3, Ti = 10000;

K = 3.5, Ti = 10000;

K = 4, Ti = 10000;

K = 5, Ti = 10000;

K = 6, Ti = 10000;

K = 6.6, Ti = 10000;

K = 6.5, Ti = 10000;

K = 7, Ti = 10000.

iii) Answer

K = 0.1, Ti = 10000 => Stable and 1 LED lighted;

K = 2, Ti = 10000 => Stable and 3 LEDs lighted;

K = 3, Ti = 10000 => Stable and 3 LEDs lighted;

K = 3.149, Ti = 10000 => Stable and 3 LEDs lighted;

K = 3.15, Ti = 10000 => Stable and 3 LEDs lighted;

K = 3.151, Ti = 10000 => Stable and 4 LEDs lighted!

K = 3.2, Ti = 10000 => Stable and 3 LEDs lighted;

K = 3.25, Ti = 10000 => Stable and 3 LEDs lighted;

K = 3.3, Ti = 10000 => Stable, 3 LEDs lighted;

K = 3.5, Ti = 10000 => Stable, 3 LEDs lighted;

K = 4, Ti = 10000 => Stable, 3 LEDs lighted;

K = 5, Ti = 10000 => Stable, 3 LEDs lighted;

K = 6, Ti = 10000 => Stable, 3 LEDs lighted;

K = 6.6, Ti = 10000 => Unstable;

K = 6.5, Ti = 10000 => Unstable;

K = 7, Ti = 10000 => Unstable.

K	Control Accuracy	Convergence Speed
0.1	Low	*Extremely low*
2 ~ 3.15	High	Slow
3.151	Highest	Slowest
3.2 ~ 6	High	Slow
6.6 ~ 7	Low	Fast

2.8) **i) Code**

```
1.  float updatePI(float pos, float target, float dt)
2.  { // add your controller here
3.      float U_PI = 0.0;
4.      E = target - pos;
5.      U_PI = K * (E);
```

```
6.      U_hat = U_PI + U_AW;
7.      U = saturation(U_hat);
8.      return U;
9.  }
```

ii) Controller Parameters

K = 0.1, Ti = 10000;

K = 2, Ti = 10000;

K = 3, Ti = 10000;

K = 3.149, Ti = 10000;

K = 3.15, Ti = 10000;

K = 3.151, Ti = 10000;

K = 3.2, Ti = 10000;

K = 3.25, Ti = 10000;

K = 3.3, Ti = 10000;

K = 3.5, Ti = 10000;

K = 4, Ti = 10000;

K = 5, Ti = 10000;

K = 6, Ti = 10000;

K = 6.6, Ti = 10000;

K = 6.5, Ti = 10000;

K = 7, Ti = 10000.

iii) Answer

P controller is used to stabilize the unstable process. The main usage of a P controller is to decrease the steady state error of the system. From the parameters' data in 2.7), the system was more and more stable when the proportional gain factor K increased. Thus, as the proportional gain factor K increases, the steady state error decreased.

However, if the proportional gain factor K was increased continually, more oscillations would be happened. The system was in an unstable state.

Task 3 Answers

3.1) **i) Code**

```
1.  float updatePI(float pos, float target, float dt)
2.  { // add your controller here
3.      float U_PI = 0.0;
```

```
    E = target - pos;
    Ei = Ei + E * dt
    U_PI = K * (E + 1 / Ti * Ei);
    U_hat = U_PI + U_AW;
    U = saturation(U_hat);
    return U;
}
```

ii) Controller Parameters

K = 6.6; Ti = 10000.

3.2) **i) Code**

```
float updatePI(float pos, float target, float dt)
{ // add your controller here
    float U_PI = 0.0;
    E = target - pos;
    U_PI = K * (E);
    U_hat = U_PI + U_AW;
    U = saturation(U_hat);
    return U;
}
```

ii) Controller Parameters

K = 2.64; Ti = 0.728

iii) Answer

Using Ziegler-Nichols method,

Ultimate gain K = 6.6

=> K = 6.6*0.4 = 2.64

Ti = 10000, delta(T) = 10 seconds

=> delta(N) = 11 oscillations

=> Ultimate period T = 0.91 oscillations/sec

=> Ti = 0.91*0.8 = 0.728

Thus, K = 2.64; Ti = 0.728.

3.3) **i) Code**

```
// controller coefficients
float K = 2.64;
float Ti = 0.728; //
float Tr = sqrt(Ti); // anti-windup gain
```

```
float updatePI(float pos, float target, float dt)
{ // add your controller here
    float U_PI = 0.0;
    E = target - pos;
    Ei = Ei + E * dt;
    U_PI = K * (E + 1 / Ti * Ei);
    U_hat = U_PI + U_AW;
    U = saturation(U_hat);
    return U;
}
```

ii) Controller Parameters

K = 2.64; Ti = 0.728.

iii) Answer

Compare: the pointer was more stable than the situation in 2.6) and got a highest control accuracy.

3.4) **i) Code**

```
// controller coefficients
float K = 2.64;
float Ti = 0.728; //
float Tr = sqrt(Ti); // anti-windup gain


float updatePI(float pos, float target, float dt)
{ // add your controller here
    float U_PI = 0.0;
    E = target - pos;
    Ei = Ei + E * dt;
    U_PI = K * (E + 1 / Ti * Ei);
    U_hat = U_PI + U_AW;
    U = saturation(U_hat);
    return U;
}
```

ii) Controller Parameters

K = 2.64; Ti = 0.728.

iii)Answer

K = 2.64

Ti	Control Accuracy	Convergence Speed
0.728	Highest	*Slowest*
0.7	High	Slow
0.65	Medium	Medium
0.4	Low	Fast
0.3	Low	Fast
0.25	Be in unstable state, has no control accuracy and has very fast convergence speed.	

3.5) **i) Code**

```
// controller coefficients
float K = 2.64;
float Ti = 0.728; //
```

```
12. float Tr = sqrt(Ti); // anti-windup gain
```

```
21. float updatePI(float pos, float target, float dt)
22. { // add your controller here
23.     float U_PI = 0.0;
24.     E = target - pos;
25.     Ei = Ei + E * dt;
26.     U_PI = K * (E + 1 / Ti * Ei);
27.     U_hat = U_PI + U_AW;
28.     U = saturation(U_hat);
29.     return U;
30. }
```

ii) Controller Parameters

K = 2.64; Ti = 0.728.

iii) Answer

PI controllers were designed to achieve zero steady-state error.

In the output of the system, the steady-state error exists when the system only has a proportional control. In integral control, the output of the controller is proportional to the integral of the input error signal. The integral term is, in fact, the integral of error in time. As time increases, the integral item increases. Thus, the error is smaller and smaller. The integral term will also increase with time. Therefore, the proportional and integral (PI) controller can make the system not generate the steady-state error after entering the steady state.

Task 4 Answers

4.1) **i) Code**

```
1.  float antiwindup(float U, float U_hat, float dt)
2.  { // add your anti-windup function here
3.      Esat = U - U_hat;
4.      Esati += Esat * dt;
5.      return (1 / Tr * (Esati));
6.  }
```

```
1.  float updatePI(float pos, float target, float dt)
2.  { // add your controller here
3.      float U_PI = 0.0;
4.      E = target - pos;
5.      Ei = Ei + E * dt;
6.      U_PI = K * (E + 1 / Ti * Ei);
7.      U_AW = antiwindup(U, U_hat, dt); // anti-windup
8.      U_hat = U_PI + U_AW;
9.      U = saturation(U_hat);
10.     return U;
11. }
```

ii) Controller Parameters

K = 2.64; Ti = 0.728.

4.2) **i) Code**

```
1.  float antiwindup(float U, float U_hat, float dt)
```

```
{ // add your anti-windup function here
    Esat = U - U_hat;
    Esati += Esat * dt;
    return (1 / Tr * (Esati));
}
```

```
float updatePI(float pos, float target, float dt)
{ // add your controller here
    float U_PI = 0.0;
    E = target - pos;
    Ei = Ei + E * dt;
    U_PI = K * (E + 1 / Ti * Ei);
    U_AW = antiwindup(U, U_hat, dt); // anti-windup
    U_hat = U_PI + U_AW;
    U = saturation(U_hat);
    return U;
}
```

ii) Controller Parameters

K = 2.64; Ti = 0.728.

iii) Answer

The system was faster and got less oscillations before pointing the final accurate North with anti-windup.