Lab 7 Report - MileStone 7

Yuhan Wang 2233095

Wankang Zhai

ECE4437, DMU, Spring 2024

1. Overview

This lab is more like an individual lab, it is based on lab6 but also contains many new knowledge such as U-turn and R-turn. To put it in a nutshell, when robotic car is passing straight line, it will obey PID control. When encountering a turning point, it will turn right. The turning point is determined by the right distance sensor. What's more, when car detects object front, it will turn back for 180 degree and then begin to move.

Overall, this lab contains PID control, U-turn and R-turn.

2. Detailed Description and Results

2.1 Enable front sensor

To enable front sensor, it will use the same way as the right distance sensor. The only difference is the different pin. The code is shown in Figure1.

```
void ADC1_init(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_6);

    ADCSequenceDisable(ADC0_BASE, 2);
    ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_CH1);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_CH1);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_CH1);
    ADCSequenceStepConfigure(ADC0_BASE, 2, 3, ADC_CTL_CH1|ADC_CTL_IE|ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 2);
    IntMasterEnable();
    IntEnable(INT_ADC0SS2);
    ADCIntEnable(ADC0_BASE, 2);
}
```

Figure 1 The code of configuring the front distance sensor

As the code shows, it will use pin6 of port E. Here I still use ADC0 instead of ADC1 because they are individual with each other and they are just two modules, no matter using which of them, it will show right work.

The most important thing here is I use sequence 2 to sample the physical variables in nature. Notice that there are 4 sequences in total. Sequence 4 is one sampling that is not fitting with this problem. Sequence 0 is 8 sampling that will occupy too much memory room. As a result, choosing sequence 1 seems to be a very good choice. Then I follow the steps that I have done when configuring the right distance sensor.

2.2 The ADC1IntHandler function

Correspondingly, the interrupt handler function is also needed for a new sensor and sequence. I open up the startup_ccs.c file and change the default name to ADC1IntHandler. The code is shown in Figure 2.

```
void ADC1IntHandler(void)
{
    ADCIntClear(ADC0_BASE, 2);
    ADCSequenceDataGet(ADC0_BASE, 2, ADC1Val);

    int j = 0;
    for (j = 0; j<4;j++)
    {
        mean1+=ADC1Val[j];
    }
    mean1 = mean1/4;
    distance1 = -1.858203*pow(10,-9)*pow(mean1, 3) + 1.3214630459*pow(10, -5)*pow(mean1, 2) - 0.0335649164396
    if(distance1 < 6.3)
    {
        u_turn = 1;

    }
    else if (distance1 > 10.68)
    {
        u_turn = 2;
    }
}
```

Figure 2 The code of ADC1IntHandler function

In the code, the calculation method is the same as the right distance sensor function. As the lab instruction said, the front distance should be about 8cm, but actually it does not work very well so that I change the lower limit to 6.3cm. For the upper limit, it is 10.68cm after many times' detections.

2.3 U-turn

U-turn is the code that lets the robotic car turn back. The code is shown in Figure 3.

```
void UTURN(void)
{
    working = 1;


    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_7);
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2|GPIO_PIN_3, GPIO_PIN_3);
    duty1 = 78;
    duty2 = 78;
    width1 = load * duty1*0.01;
    width2 = load * duty2*0.01;
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, width1);
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, width2);
    GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|GREEN_LED|BLUE_LED, RED_LED);
}
```

Figure 3 The code of U-turn

The working is a flag that refers it is now woking with the U-turn. To realize U-turn, the most important thing is to let the two wheels rotate for different directions. As for the duty cycle, just let them equal. In this way, the robotic car can realize the U-turn.

Here is another thing that is needed to be done. When detecting the

front object, the car will first stop and then takes the U-turn. The code is shown in Figure 4.

```
if((u_turn == 1) && (working != 1))
{
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6|GPIO_PIN_7, 0);
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2|GPIO_PIN_3, 0);
    SysCtlDelay(SysCtlClockGet() / (1000 * 3));
    UTURN();
    u_turn = 0;
}
```

Figure 4 The code of stopping the car first

This code is written in the main function under the infinite while loop. The condition is that u_turn is set while working is not equals to 1, which means that it is the first time to execute the U-turn function. Firstly, it will set the pulsewidth of two wheels to 0 and then call the UTURN function.

2.4 Right-turn

Right-turn is the code that lets the robotic car turn back. The code is shown in Figure 4.

```
void TURNRIGHT(void)
{

    duty1 = 31;
    duty2 = 95;
    width1 = load * duty1*0.01;
    width2 = load * duty2*0.01;

    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, width1);
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, width2);
    GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|GREEN_LED|BLUE_LED, BLUE_LED);
    GPIOPinWrite(GPIO_PORTF_BASE, RED_LED, 0);
    GPIOPinWrite(GPIO_PORTF_BASE, GREEN_LED, 0);
}
```

Figure 4 The code of turn right

The key point is to tune the duty cycle of the two wheels. The basic rule is that the left wheel's duty cycle is lager than the right wheel's. After detecting for many times, I finally get the value that the right wheel's duty cycle is 31 and the left wheel's duty cycle is 95, under this condition, it can reach a very perfect destination of turning right.

What's more, another important thing is to set the limit of the condition of turning right. There is an necessary limitation that the distance from right should be larger than the value in PID control. The reason is obvious. The condtion is shown in Figure 5.

```
void ADC0IntHandler(void)
{
    ADCIntClear(ADC0_BASE, 1);
    ADCSequenceDataGet(ADC0_BASE, 1, ADC0Val);
    int i = 0;
    for(i = 0;i<4;i++)
    {
         mean+= ADC0Val[i];

    }
    mean = mean / 4;

    distance = -1.858203*pow(10,-9)*pow(mean, 3) + 1.3214630459*pow(10, -5)*pow(mean, 2) - 0.03356491643966*m

    SB = target - distance;
    if (distance > 13.967)
    {
        right = 1;
    }
    else
    {
        right = 2;
    }

}
```

Figure 5 The limit of turning right

2.5 The while loop in main function

Now we have already have the enough functions to realize U-turn and right-turn. The only thing that we need to do now is to use different function based on the setted condition. The code is shown in Figure 6.

```
while(1)
{
    SysCtlDelay(SysCtlClockGet() / (1000 * 3));
    if((u_turn == 1) && (working != 1))
    {
        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6|GPIO_PIN_7, 0);
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2|GPIO_PIN_3, 0);
        SysCtlDelay(SysCtlClockGet() / (1000 * 3));
        UTURN();
        u_turn = 0;
    }
    else if((u_turn == 1) && (working == 1))
    {

        UTURN();
        u_turn = 0;

    }
    else if (u_turn == 2 )
    {
        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_6);
        GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2|GPIO_PIN_3, GPIO_PIN_3);
        duty1 = 50;
        duty2 = 50;
        width1 = load * duty1*0.01;
        width2 = load * duty2*0.01;

        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, width1);
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, width2);
        PID_CONTROL();
        u_turn = 0;
    else if((right == 1) && (u_turn != 1))
    {
        TURNRIGHT();
        right = 0;
    }
    else if((right == 2) &&(u_turn !=1))
    {

        PID_CONTROL();
        right = 0;
    }
    else if ((right == 2)&& (u_turn == 1))
    {
        GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|GREEN_LED|BLUE_LED, RED_LED);

    }
```

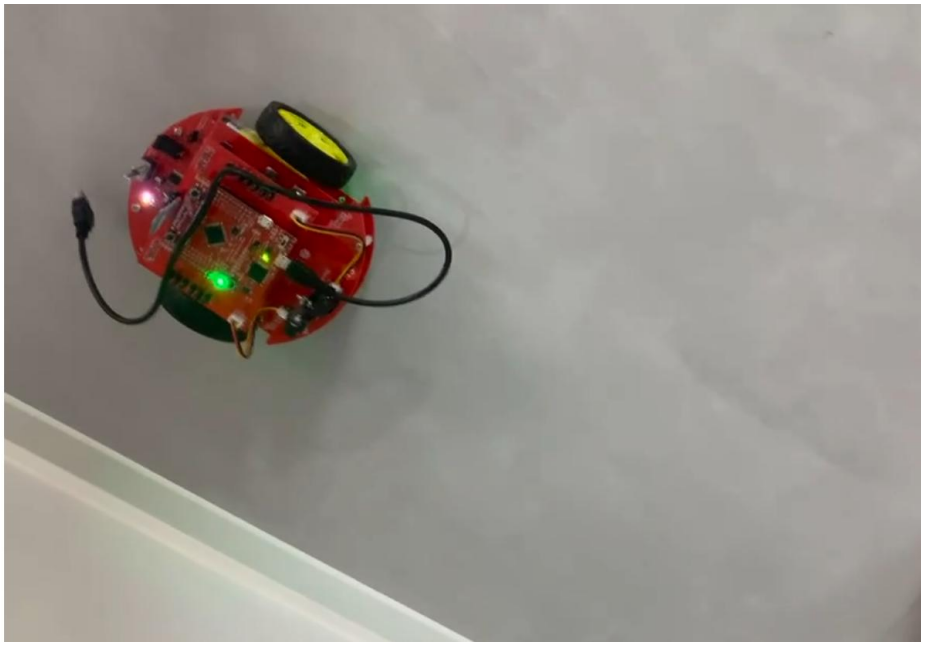
Figure 6 The code in the infinite loop

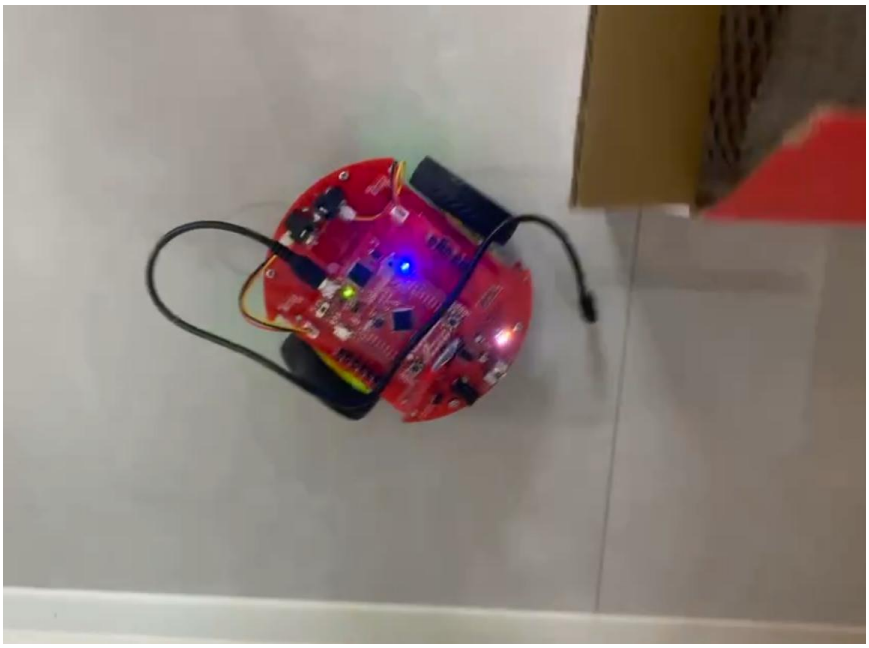
The code is above. There are many different if statements. I will explain them one by one. The first condition is u_turn equals to 1 and working is not 1. In this condition, it will first stop the car and then call the UTURN function. For the second condition, it is used in the case that u_turn equals to 1 and working also equals to 1. This means that it is within the U-turning process.

For the third case, when u_turn equals to 2, which means that it will not turning back. The first thing to do is to let the duty cycle of the two wheels equal to avoid them rotate for more than 180 degree. Then it will take into the PID control. Another case is that when right equals to 1, then it will call the turning right and finally it will reset the turning right's flag to 0.

When the robotic car needs to go on the straight line, that is, under the condition that right equals to 2 and u_turn not equals to 1. Then, PID control will play an important role.

2.6 The real performance





3. Conclusion

After finishing this lab, I get to know how to control a robotic car in a relatively much more intelligent way. Now, my robotic car can move really based on what I have set.

Looking back at what I have done from Lab1 to Lab7, it is a really pleasant journey. Now in Lab7, I really learn how to control the car and let it going freedomly.