Cover Page

Lab 3 Report

Implementation of **CTI** commands application

ECE4437, DMU

Spring 2024

Wankang Zhai

2232923

Contents

1. Overview

In this experiment, we implemented the command line interpreter technique to enter simple instructions in the Bluetooth terminal and then call specific functions. We understood the CLI and completed the following tasks:

1. Use function pointers and structs to establish the basic struct of CLI
2. Enter 3-letter commands then look up the actions in the struct, and then take actions
3. Create my own commands to realize ‘FWD’, ‘RRR’, ‘LLL’, ‘TTT’, so that it will send different sentence via UART0 to TIVAC terminal.
4. I integrated the 3-letter commands and functions into the struct and used a For loop to search. Finish finding and executing the command.

We successfully implemented the CLI function by completing a lookup table and applied it to the TIVAC development board to complete the LAB3 experiment. I learned:

1. How to combine character and function pointer through struct, and use function pointer to find the corresponding function by inputting character.
2. How to traverse function pointers and find the corresponding function to execute
3. The true meaning of CTI and its applications.

2. Description of Tasks and Results

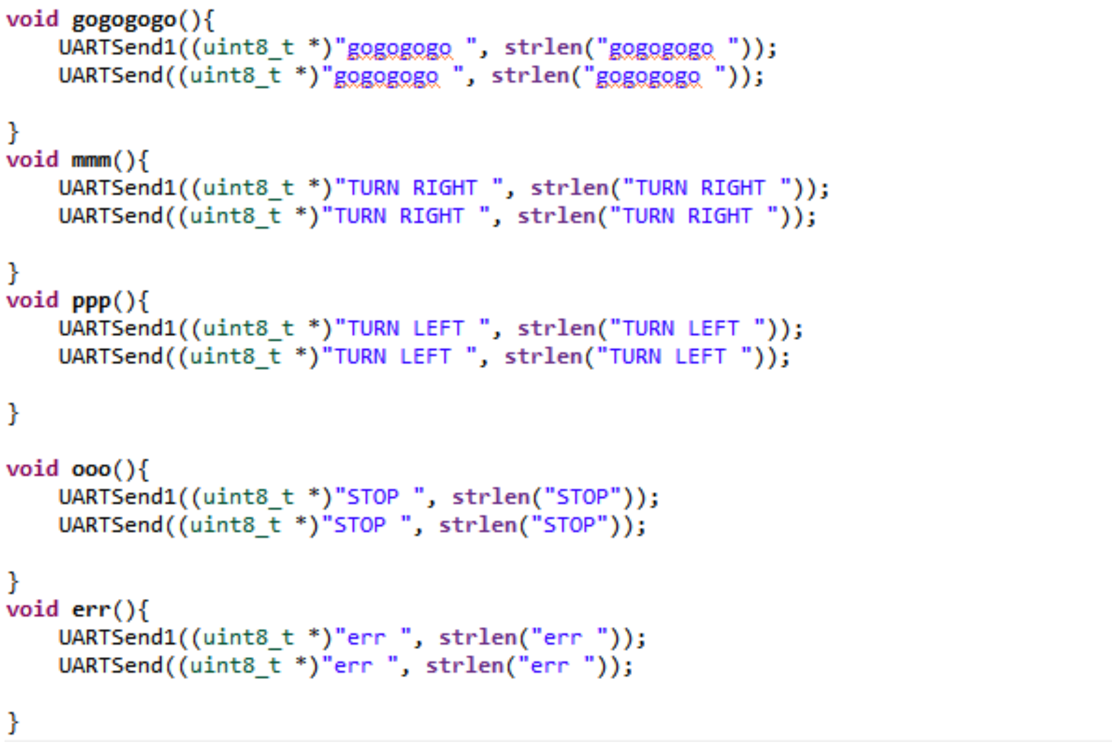
Task1：Set the corresponding function to be called through the function pointer of the struct.

In Lab3, I set up a total of 4 functions, corresponding to: forward, turn right, turn left, stop. The specific four letters commands and function functions of the function are shown in Table 1.

3-letter commands	Function return sentence
FWD	gogogo
RRR	TURN RIGHT
LLL	TURN LEFT
TTT	STOP
WRONG COMMANDS	err

Table 1

We implement these functions as shown in Figure 1, and send different characters to the TivaC terminal and Bluetooth terminal through the UARTSend and UAETSend1 functions respectively.



```
void gogogogo(){
    UARTSend1((uint8_t *)"gogogogo ", strlen("gogogogo "));
    UARTSend((uint8_t *)"gogogogo ", strlen("gogogogo "));

}
void mmm(){
    UARTSend1((uint8_t *)"TURN RIGHT ", strlen("TURN RIGHT "));
    UARTSend((uint8_t *)"TURN RIGHT ", strlen("TURN RIGHT "));

}
void ppp(){
    UARTSend1((uint8_t *)"TURN LEFT ", strlen("TURN LEFT "));
    UARTSend((uint8_t *)"TURN LEFT ", strlen("TURN LEFT "));

}

void ooo(){
    UARTSend1((uint8_t *)"STOP ", strlen("STOP"));
    UARTSend((uint8_t *)"STOP ", strlen("STOP"));

}
void err(){
    UARTSend1((uint8_t *)"err ", strlen("err "));
    UARTSend((uint8_t *)"err ", strlen("err "));

}
```

Figure 1

Task 2： Set up the struct and complete the struct declaration.

In this experiment, we defined the struct according to the content in Week3. The struct contains two members. The first member is cmdd, a character array containing four characters. The second member is a function pointer that takes no parameters and returns no value. We name this struct lookup. Then declare a lookup-shaped array named commands. Execute different functions according to different commands. The specific syntax and implementation steps are shown in Figure 2.

```
    typedef struct{
    char cmdd[4];
    void(*function)(void);
}lookup;

lookup commands[] = {
        {"FWD",gogogogo},
        {"RRR",mmm},
        {"LLL",ppp},
        {"TTT",ooo},
};
```

Figure2

Task 3： Redesign the counter, use the counter to store the character array, and set the input completion flag.

In Lab3, we need to check whether our input reaches three. If it does not reach three input characters, the flag is 0. We do not need to detect characters and declarations in the struct. So we defined a new conter in UARTHandler. Figure 3 shows how we define counter and how to store the characters corresponding to the flag bits of counter into the character array.

```
            if(i%3==0){
                    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
                    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3,
        GPIO_PIN_3);
                    i =0;
                    flag = 1;
                }
                }else{
                    flag = 0;
                }
                }
            }
```

Figure 3

We set the flag to 1 and i to 0 in the last condition, after the green light turns on. If the condition that the remainder after division by 3 is 0 is not met, the flag is set to 0. Therefore, we only need to check the value of flag to determine whether there are three instructions.

Task 4: Loop according to the flag

In Task4, we judge the previously set flag and complete the loop in the struct. I design the program according to the following program logic.

```
  while(1)
    {
        for (x= 0; x< sizeof(commands)/sizeof(commands[0]);
++x){
            if (flag ==1){
                ans = strncmp(cmd, commands[x].cmdd,3);
                if(ans==0){
                    commands[x].function();
                    flag = 0;}

                if(ans==1){
                    err();
                    flag = 0;}//

            }
        }
    }
}
```

Figure 4

First, we define an int type variable x and let x traverse the length of the commands character array to find the definition of each set of characters and functions in this struct. Then we use commands[x] to index each element in the struct. The characters can be indexed through the commands[x].cmdd command and compared with the character array we previously stored in the UARTHandler. We use the strncmp command to compare whether the first three characters are the same. The cmd characters and commands.cmd characters we defined are both 4 bytes. Therefore, the first three characters store the three letters starting from the green light.

Through comparison, we have to complete the following tasks: If the flags are equal (which means we have entered three characters), then if any character array in my struct is the same as the character array I entered, then judgment 1 is performed. Through commands [x].function(); The function calls the function corresponding to the character array in the struct. If it is different from the character array I entered, an error will be reported and the error function will be called. Through two judgments, we can complete the judgment of all situations under the premise of flag=1 and ensure the completeness of the program.

3. Overview

In this experiment, we mastered function pointers proficiently. More importantly, we completed the application of CLI through function pointers and structs, and successfully implemented the execution of CLI commands on the TivaC terminal. We define a struct containing a character array and a function pointer, so that we can input a character array, check whether it is the same as the declared character array, and then call the corresponding function to complete the CLI settings.

We also realize the practicality of CLI. CLI is the Command Line Interface. Users can interact and control the program by inputting simple text. It is very convenient and lays a broad foundation for our future robot control.