

车载软件开发基础

课后实践3



**2024至2025学年第 1 学期**

|  |  |
| --- | --- |
| 学号 | 姓名 |
| E2021077 | 沈冲 |
| 任课教师 | 刘骥 |
| 成 绩 |  |

|  |  |
| --- | --- |
| 任务书 | |
| 任务内容 | 请针对课后实践2的内容，补充多线激光雷达和底盘模块，使用订阅者模式，实现以下功能。   1. 多线激光雷达作为主题方，发布主题，并接受订阅 2. 底盘模块中增加观察者，向雷达模块进行订阅注册，并接收订阅通知。 3. 当雷达更新障碍物状态时，底盘模块通过订阅者模式接收通知并执行对应操作。 4. 障碍我状态为“前方“，则底盘执行”后退“执行 5. 障碍我状态为“右前方“，则底盘执行”左转“执行 6. 障碍我状态为“左前方“，则底盘执行”右转“执行   注意：  1、底盘执行指令，只需要在屏幕输出类似“左转。。。。”文字即可。  2、障碍物状态可通过命令行方式输入。比如1代表前方障碍，2代表左前方障碍，3代表右前方障碍。 |
| 程序规范 | （1）所有程序代码采用C++编写，使用git进行源代码管理；  （2）类名、变量名、函数名应符合C++的命名规范，并在代码中前后保持一致；  （3）涉及面向对象的程序，例如自定义的类，应符合面向对象的设计原则；  （4）正确使用头文件和源文件，自定义的头文件应符合头文件的编写原则，例如用条件宏定义确保头文件不被多次引用、不在头文件中进行类和函数的实现（模板除外）； |
| 报告要求 | （1）报告至少应该包括程序设计、程序效果展示、总结分析3个部分；  （2）程序设计描述组成程序的模块、类、函数以及他们之间的相互关系，若有算法，可以描述算法流程；  （3）程序效果展示除了程序运行效果截图之外，应该有必要的文字说明；  （4）总结分析可以分析实现的效果与理想情况的差异，分析导致这些差异的原因，切忌不要写成心得体会；  （5）报告应该格式规范、排版整洁、少语病和错误。 |
| 作业提交 | （1）含有git仓库（有.git目录）的完整源代码；  （2）任务报告。 |
| 评分标准 | 按照五级制打分，分为优秀、良好、中等、及格、不及格，各评分项占总成绩的比例为：  （1）任务完成情况占评分的60%；  （2）报告占评分的40%。  评分老师根据各部分的完成情况，直接给出总成绩。 |

1. **程序设计**

本次实践通过两个代码文件实现预期功能，分别是Publisher.cpp和Subscriber.cpp。Publisher.cpp实现多线激光雷达的功能，即发布者功能，该文件接收用户输入数字1、2、3，分别代表前方障碍、左前方障碍、右前方障碍，并将用户的输入数据发送到话题“Lidar”上去；Subscriber.cpp实现底盘的功能，即订阅者功能，底盘从话题“Lidar”上接收发布者发布的数据，并执行相应的操作，如接收的数据为1时，底盘执行后退，接收的数据为2时，底盘执行右转，接收的数据为3时，底盘执行左转。

以下是Publisher.cpp的内容：

1. *// Publisher.cpp*
2. #include <iostream>
3. #include <fstream>
4. #include <string>
5. *// 定义消息类型*
6. struct LaserRadarMessage {
7. std::string obstacleStatus;
8. };
9. *// 定义发布者类*
10. class LaserRadarPublisher {
11. public:
12. LaserRadarPublisher(const std::string& topic) : topic\_(topic) {}
13. void publish(const LaserRadarMessage& message) {
14. std::ofstream outFile(topic\_ + ".txt", std::ios::out | std::ios::trunc);
15. if (outFile.is\_open()) {
16. outFile << message.obstacleStatus << std::endl;
17. outFile.close();
18. } else {
19. std::cerr << "无法打开文件进行写入: " << topic\_ << ".txt" << std::endl;
20. }
21. }
22. void run() {
23. LaserRadarMessage message;
24. while (true) {
25. std::cout << "1代表前方障碍，2代表左前方障碍，3代表右前方障碍。" << std::endl;
26. std::cout << "请输入障碍物状态（1，2，3），或输入'exit'退出：" << std::endl;
27. std::cin >> message.obstacleStatus;
28. if (message.obstacleStatus == "exit") {
29. break;
30. }
31. publish(message);
32. std::cout << std::endl;
33. }
34. }
35. private:
36. std::string topic\_;
37. };
38. int main() {
39. LaserRadarPublisher radarPublisher("Lidar");
40. std::cout << "激光雷达发布者启动..." << std::endl;
41. radarPublisher.run();
42. return 0;
43. }

**主要功能**: 该程序模拟了一个激光雷达（Lidar）模块，用户通过控制台输入障碍物的状态，程序将障碍物状态作为消息发布到一个文件中。这个文件作为消息的存储和传输媒介。

**组成模块**

1. 结构体 LaserRadarMessage

* 功能: 定义了激光雷达消息的数据结构，包含一个字符串成员 obstacleStatus，用于存储障碍物的状态。
* 成员变量:

std::string obstacleStatus: 存储障碍物状态的字符串。

1. 类 LaserRadarPublisher

* 功能: 模拟激光雷达数据发布系统，负责接收用户输入的障碍物状态，并将这些状态发布到一个文件中。
* 成员变量:

std::string topic\_: 存储话题名称，用于标识文件名。

* 成员函数:

构造函数: LaserRadarPublisher(const std::string& topic)

功能: 初始化激光雷达发布者，设置话题名称。

参数: topic - 话题名称。

发布函数: void publish(const LaserRadarMessage& message)

功能: 将消息发布到文件中。

参数: message - 要发布的激光雷达消息。

流程:

打开文件（topic\_.txt），如果文件不存在则创建，std::ios::trunc 模式确保文件内容被清空。

将消息写入文件。

关闭文件。

运行函数: void run()

功能: 运行发布者，不断接收用户输入并发布。

流程:

循环等待用户输入。

输出提示信息。

接收用户输入的障碍物状态。

如果输入为 "exit"，则退出循环。

调用 publish 函数发布消息。

输出空行以分隔多次输入。

1. 主函数 int main()

* 功能: 程序的入口点。
* 流程:

创建 LaserRadarPublisher 对象，初始化话题名称为 "Lidar"。

输出启动信息。

调用 run 函数开始运行发布者。

**相互关系**

LaserRadarMessage 结构体 与 LaserRadarPublisher 类:

LaserRadarPublisher 使用 LaserRadarMessage 结构体来存储和发布障碍物状态。

LaserRadarPublisher 类 与 主函数:

主函数中创建并运行 LaserRadarPublisher 对象，它们之间通过方法调用进行交互。

**算法流程**

以 LaserRadarPublisher 类的 run 函数为例，算法流程如下:

开始

输出提示信息："1代表前方障碍，2代表左前方障碍，3代表右前方障碍。"

输出提示信息："请输入障碍物状态（1，2，3），或输入'exit'退出："

输入: 用户输入障碍物状态。

判断: 如果用户输入 "exit"，则跳到步骤 9。

调用: publish 函数发布用户输入的消息。

输出: 空行。

回到步骤 3 继续循环。

结束

该程序模拟了消息发布者的行为，通过控制台与用户交互，将用户输入的消息发布到文件中，实现了基本的输入和文件写入功能。

以下是Subscriber.cpp的内容：

1. *// Subscriber.cpp*
2. #include <iostream>
3. #include <fstream>
4. #include <string>
5. #include <thread>
6. #include <chrono>
7. *// 定义消息类型*
8. struct LaserRadarMessage {
9. std::string obstacleStatus;
10. };
11. *// 定义订阅者类*
12. class ChassisSubscriber {
13. public:
14. ChassisSubscriber(const std::string& topic) : topic\_(topic) {}
15. void subscribe() {
16. std::cout << "底盘订阅者订阅了话题: " << topic\_ << std::endl;
17. while (true) {
18. std::ifstream inFile(topic\_ + ".txt");
19. if (inFile.is\_open()) {
20. std::string line, lastLine;
21. while (std::getline(inFile, line)) {
22. lastLine = line;
23. }
24. inFile.close();
26. if (!lastLine.empty()) {
27. LaserRadarMessage message{lastLine};
28. onMessageReceived(message);
29. }
30. } else {
31. std::cerr << "无法打开文件进行读取: " << topic\_ << ".txt" << std::endl;
32. }
33. std::this\_thread::sleep\_for(std::chrono::seconds(1)); *// 模拟实时接收消息*
34. }
35. }
36. void onMessageReceived(const LaserRadarMessage& message) {
37. *// 根据雷达消息更新底盘状态*
38. std::cout << "底盘订阅者收到消息: " << message.obstacleStatus << std::endl;
39. if (message.obstacleStatus == "1") {
40. std::cout << "底盘执行后退操作。" << std::endl;
41. } else if (message.obstacleStatus == "2") {
42. std::cout << "底盘执行右转操作。" << std::endl;
43. } else if (message.obstacleStatus == "3") {
44. std::cout << "底盘执行左转操作。" << std::endl;
45. }
46. std::cout << std::endl;
47. }
48. private:
49. std::string topic\_;
50. };
51. int main() {
52. ChassisSubscriber chassisSubscriber("Lidar");
53. std::cout << "底盘订阅者启动..." << std::endl;
54. chassisSubscriber.subscribe();
55. return 0;
56. }

**主要功能**: 该程序模拟了一个底盘订阅系统，它定期检查一个文件来获取激光雷达（Lidar）系统发布的消息，根据消息内容执行相应的操作。

**组成模块**

1. 结构体 LaserRadarMessage

功能: 定义了激光雷达消息的数据结构，包含一个字符串成员 obstacleStatus，用于存储障碍物的状态。

成员变量:

std::string obstacleStatus: 存储障碍物状态的字符串。

1. 类 ChassisSubscriber

功能: 模拟底盘订阅系统，负责定期检查文件中的新消息，并根据消息内容执行操作。

成员变量:

std::string topic\_: 存储话题名称，用于标识文件名。

成员函数:

构造函数: ChassisSubscriber(const std::string& topic)

功能: 初始化底盘订阅者，设置话题名称。

参数: topic - 话题名称。

订阅函数: void subscribe()

功能: 订阅消息，定期检查文件中的新消息。

流程:

循环等待文件 topic\_.txt 的更新。

打开文件。

如果文件打开成功，读取文件中的每一行，只保留最近的一条消息。

关闭文件。

如果最后一行不为空，将其作为消息传递给 onMessageReceived 函数处理。

等待一秒再次检查。

消息处理函数: void onMessageReceived(const LaserRadarMessage& message)

功能: 根据接收到的消息执行相应的底盘操作。

参数: message - 接收到的激光雷达消息。

流程:

输出收到的消息。

根据消息中的障碍物状态执行相应的操作：

"1": 输出底盘执行后退操作的信息。

"2": 输出底盘执行右转操作的信息。

"3": 输出底盘执行左转操作的信息。

输出空行以分隔多次操作。

1. 主函数 int main()

功能: 程序的入口点。

流程:

创建 ChassisSubscriber 对象，初始化话题名称为 "Lidar"。

输出启动信息。

调用 subscribe 函数开始运行订阅者。

**相互关系**

LaserRadarMessage 结构体 与 ChassisSubscriber 类:

ChassisSubscriber 使用 LaserRadarMessage 结构体来接收和处理激光雷达消息。

ChassisSubscriber 类 与 主函数:

主函数中创建并运行 ChassisSubscriber 对象，它们之间通过方法调用进行交互。

**算法流程**

以 ChassisSubscriber 类的 subscribe 函数为例，算法流程如下:

开始

输出: "底盘订阅者订阅了话题: Lidar"

进入循环:

尝试打开文件 Lidar.txt。

判断: 如果文件打开成功：

初始化变量 lastLine 为空字符串。

循环读取: 读取文件的每一行，更新 lastLine 为最新的一行。

关闭文件。

判断: 如果 lastLine 不为空：

创建 LaserRadarMessage 对象，初始化障碍物状态为 lastLine。

调用: onMessageReceived 函数，传递 LaserRadarMessage 对象。

判断: 如果文件打开失败：

输出: 错误信息 "无法打开文件进行读取: Lidar.txt"。

等待: 暂停一秒。

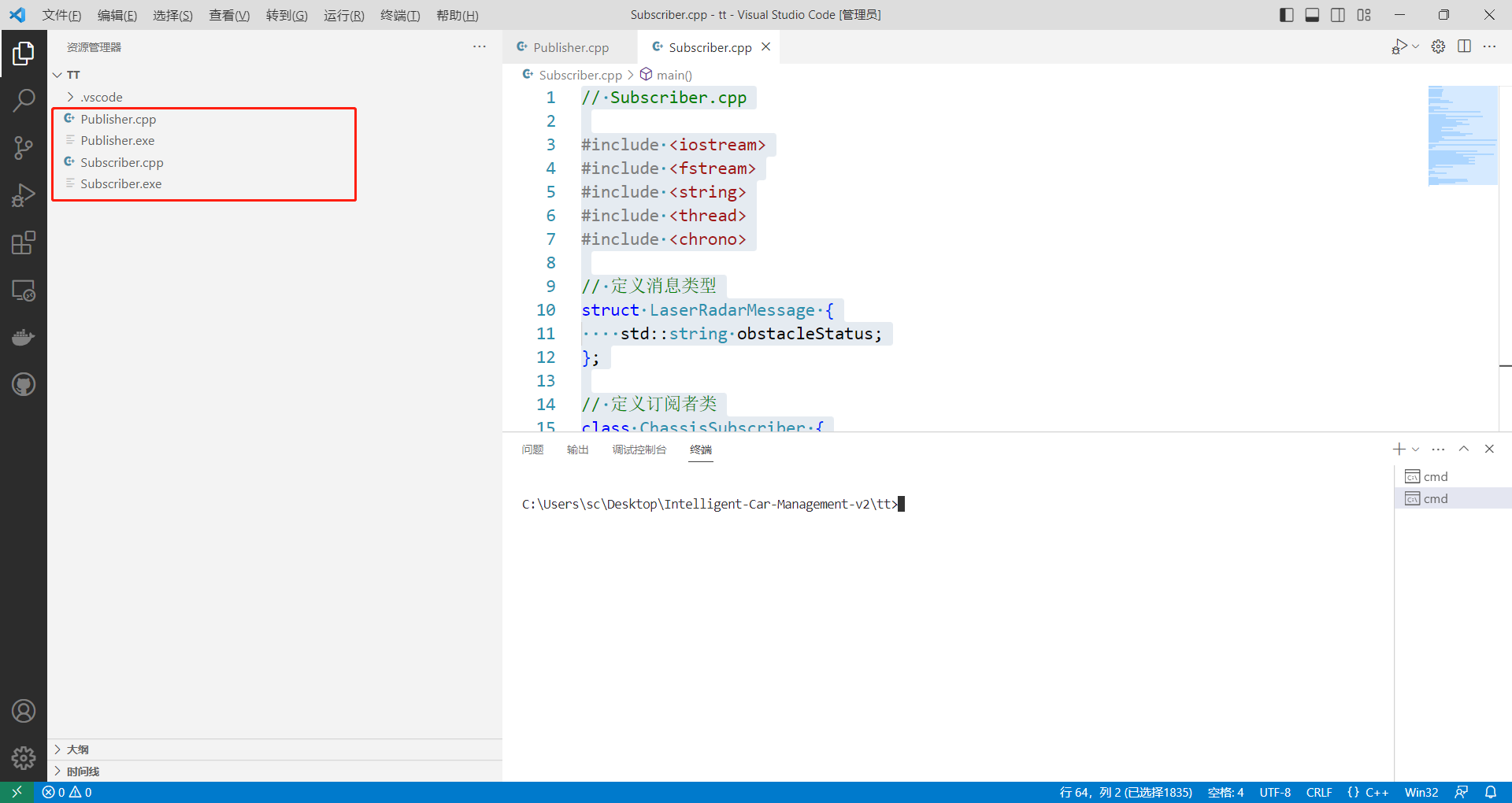
回到步骤 3 继续循环。

结束

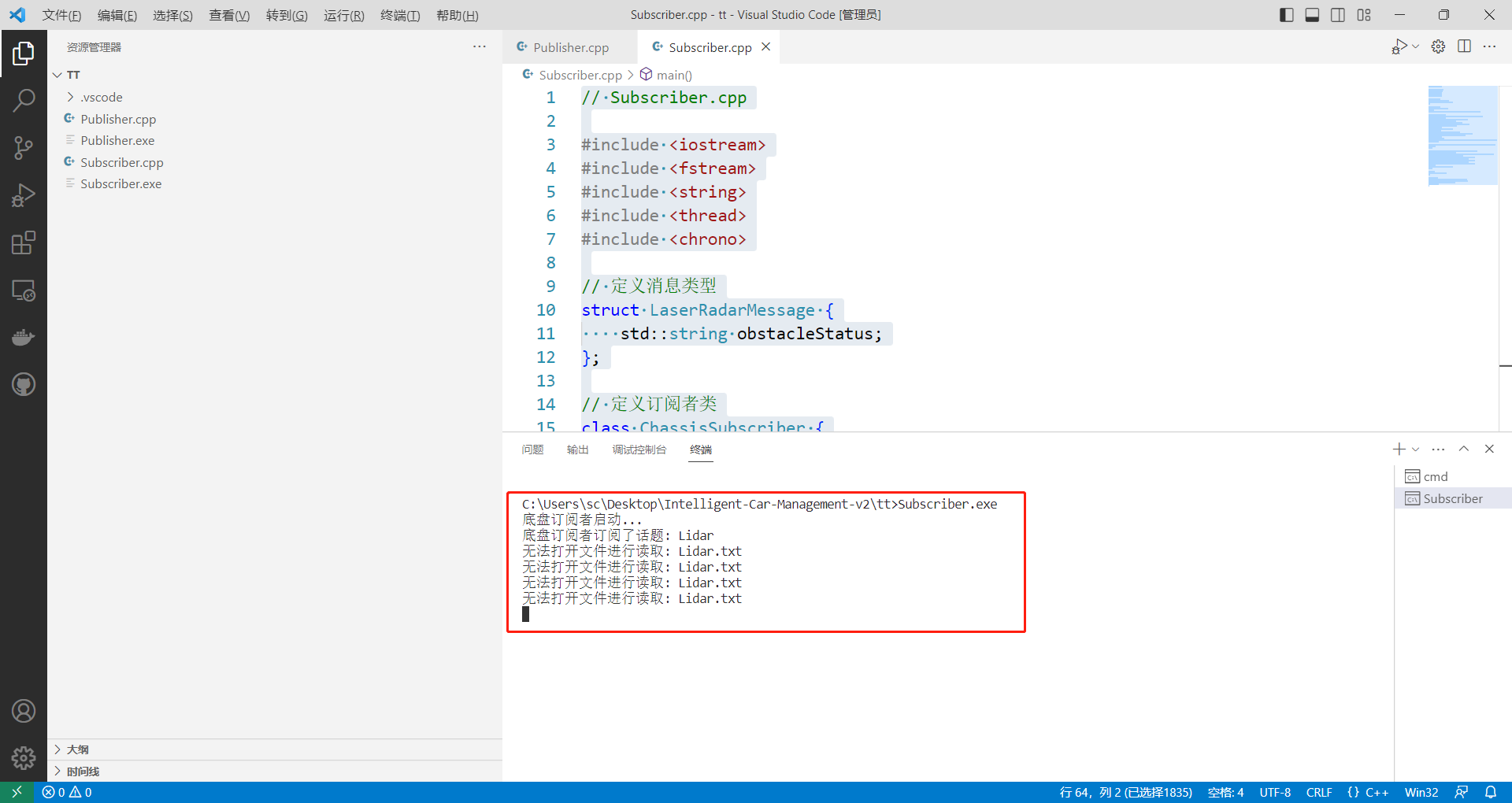
该程序模拟了消息订阅者的行为，通过定期检查文件来接收消息，实现了文件读取和消息处理功能。

1. **程序效果展示**

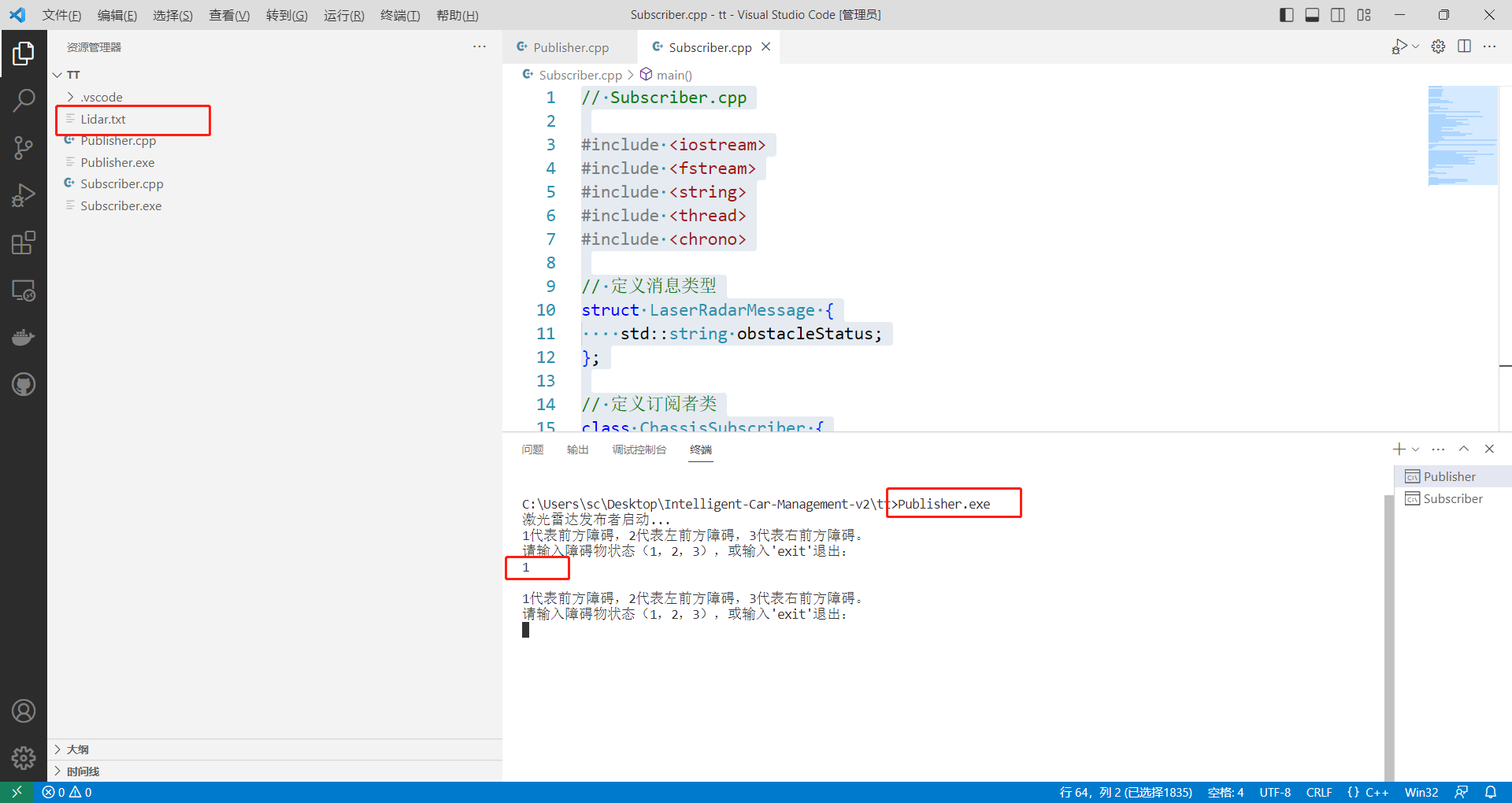
通过g++ -o Publisher Publisher.cpp和g++ -o Subscriber Subscriber.cpp将文件编译成Publisher.exe和Subscriber.exe。然后终端输入Publisher.exe和Subscriber.exe就可以执行程序了。当然为了中文显示不乱码，在执行程序前先在终端执行chcp 65001指令。

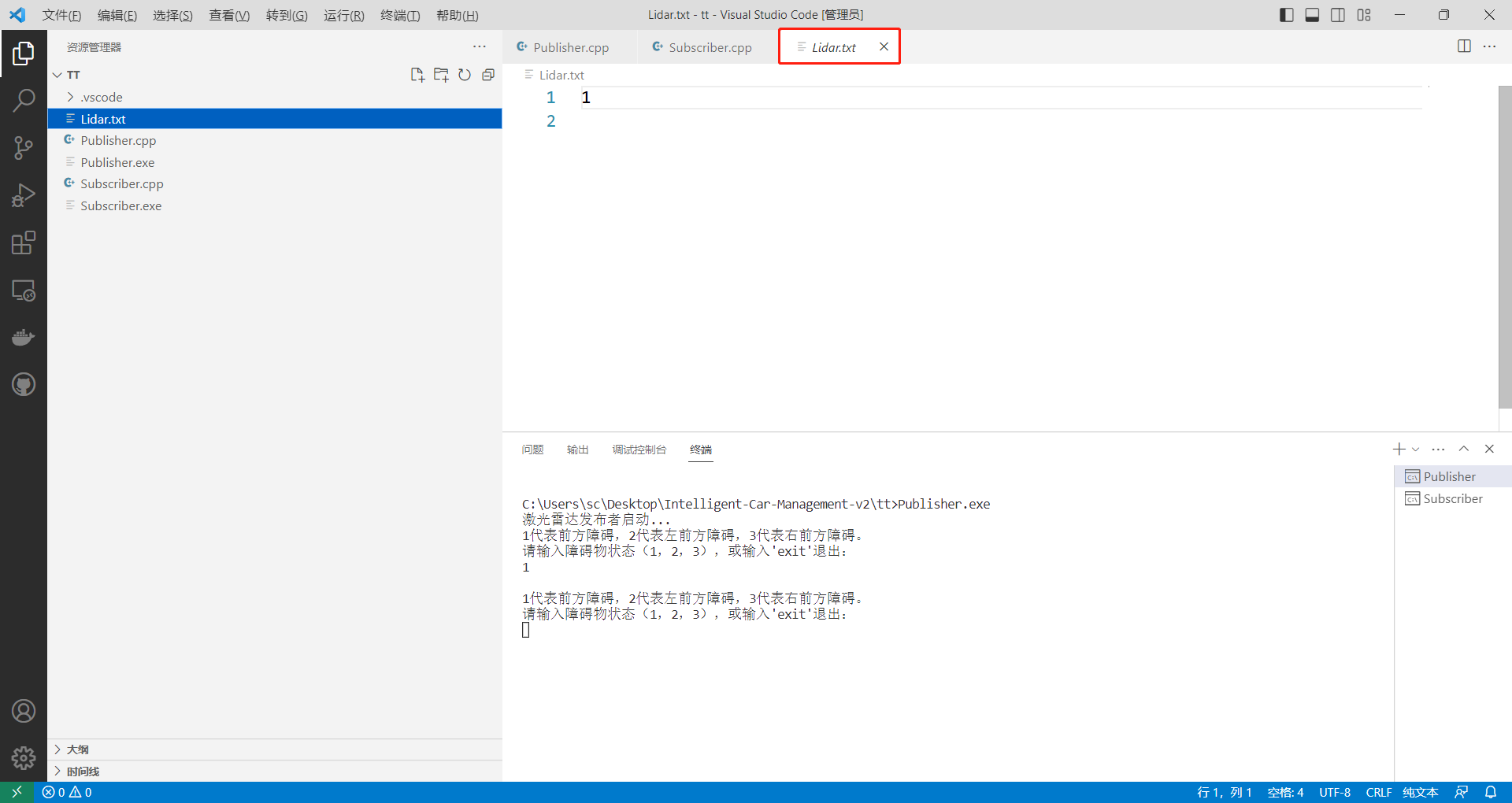


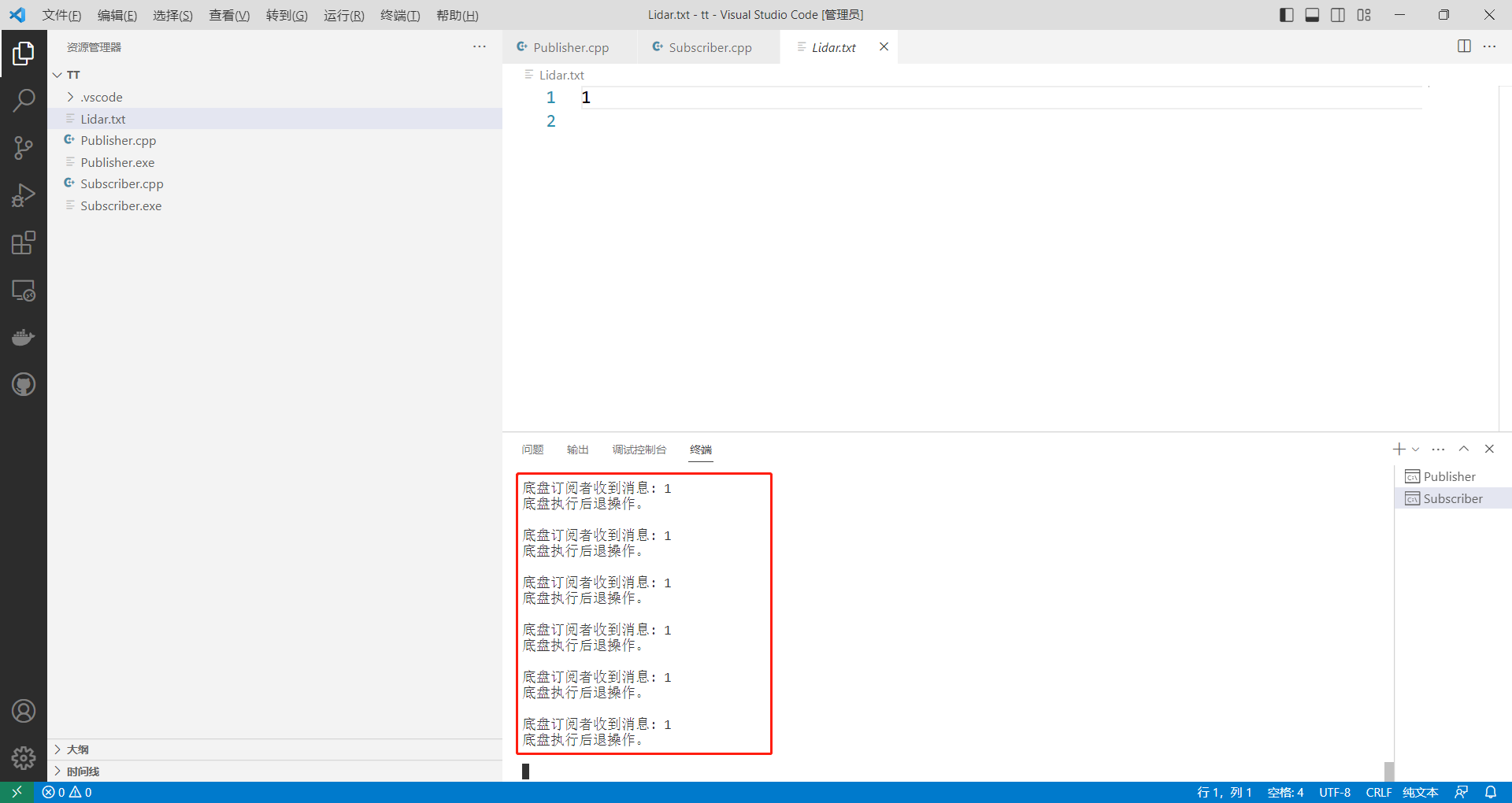
开启两个终端，在第二个终端中执行Subscriber.exe，此时由于没有执行Publisher.exe，所以激光雷达没有发布数据，底盘自然就接收不到数据。底盘模块会提示“无法打开文件进行读取: Lidar.txt”，这是因为激光雷达的数据是会发布到Lidar.txt中存储起来，然后底盘从文件中读取数据，这个时候因为是第一次执行程序且激光雷达模块没有启动，Lidar.txt并不存在，自然底盘就会输出提示信息了。



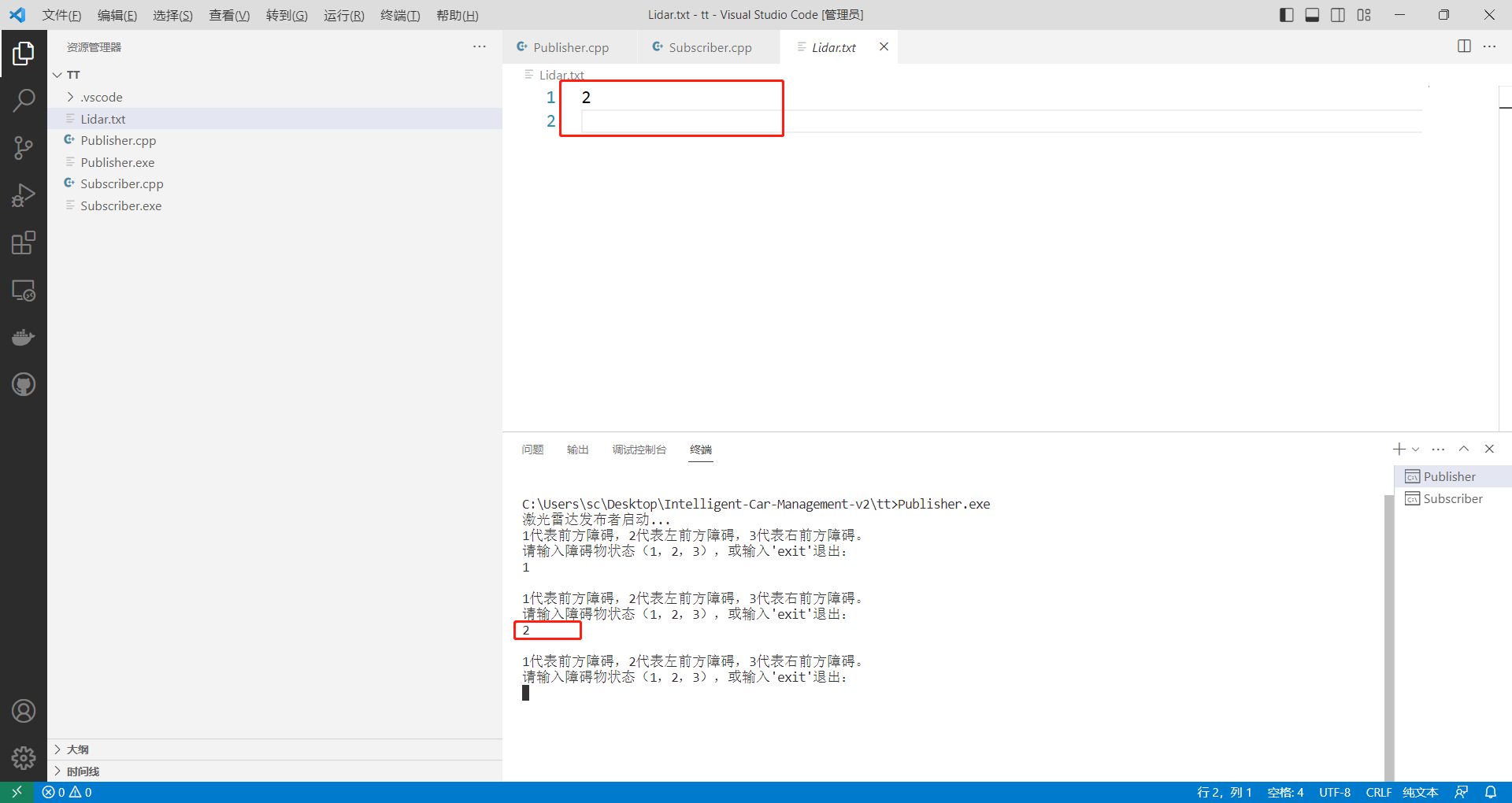
在第一个终端中执行Publisher.exe，然后输入1回车，这个时候会自动生成一个Lidar.txt文件，文件里面存储的是激光雷达的数据。此时再回到终端二，会发现底盘订阅到了激光雷达模块发送的数据，并正确执行了相应的操作。我们在终端一输入的是1，也就是代表前方有障碍物，此时底盘执行后退操作。

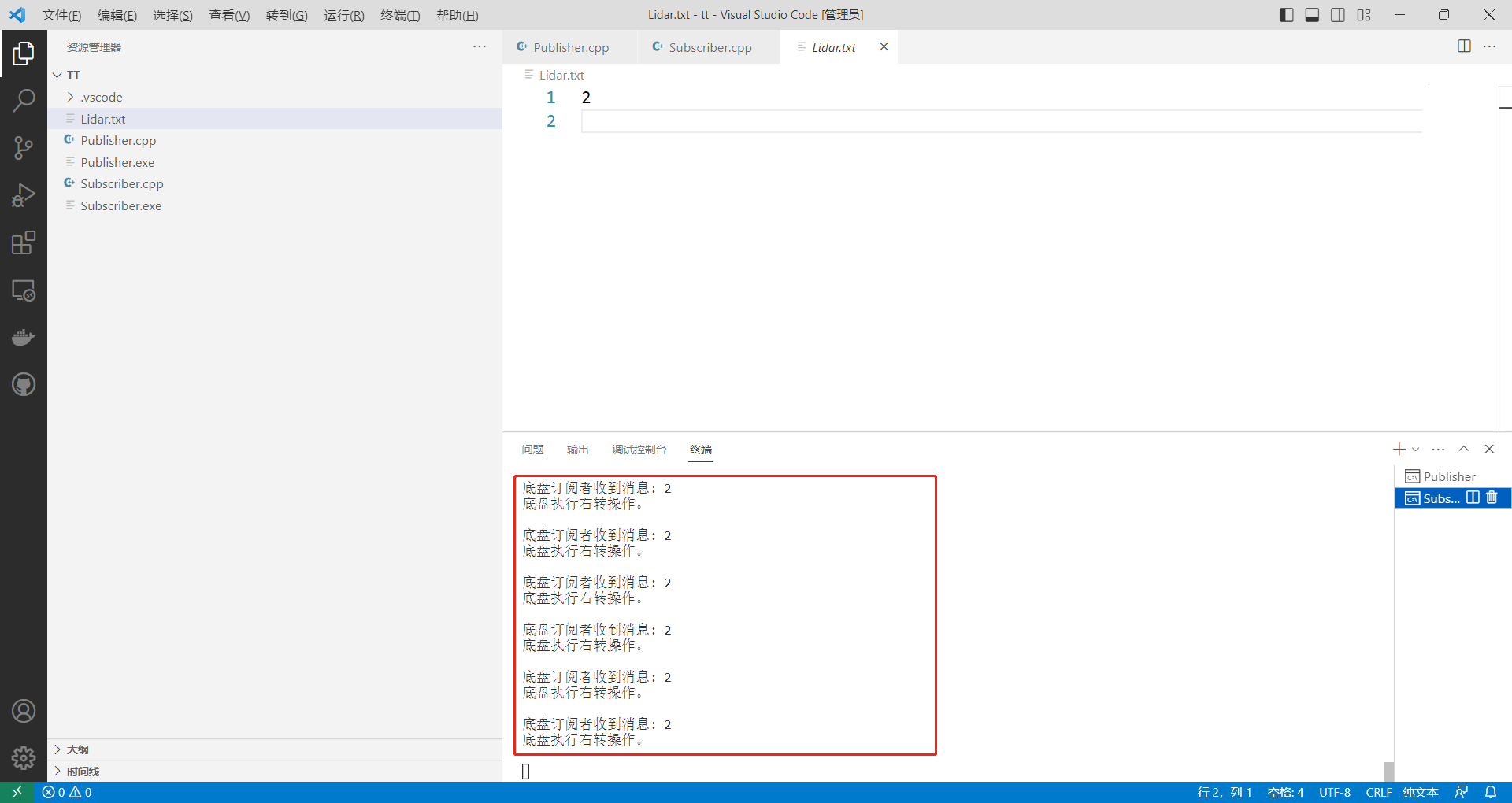




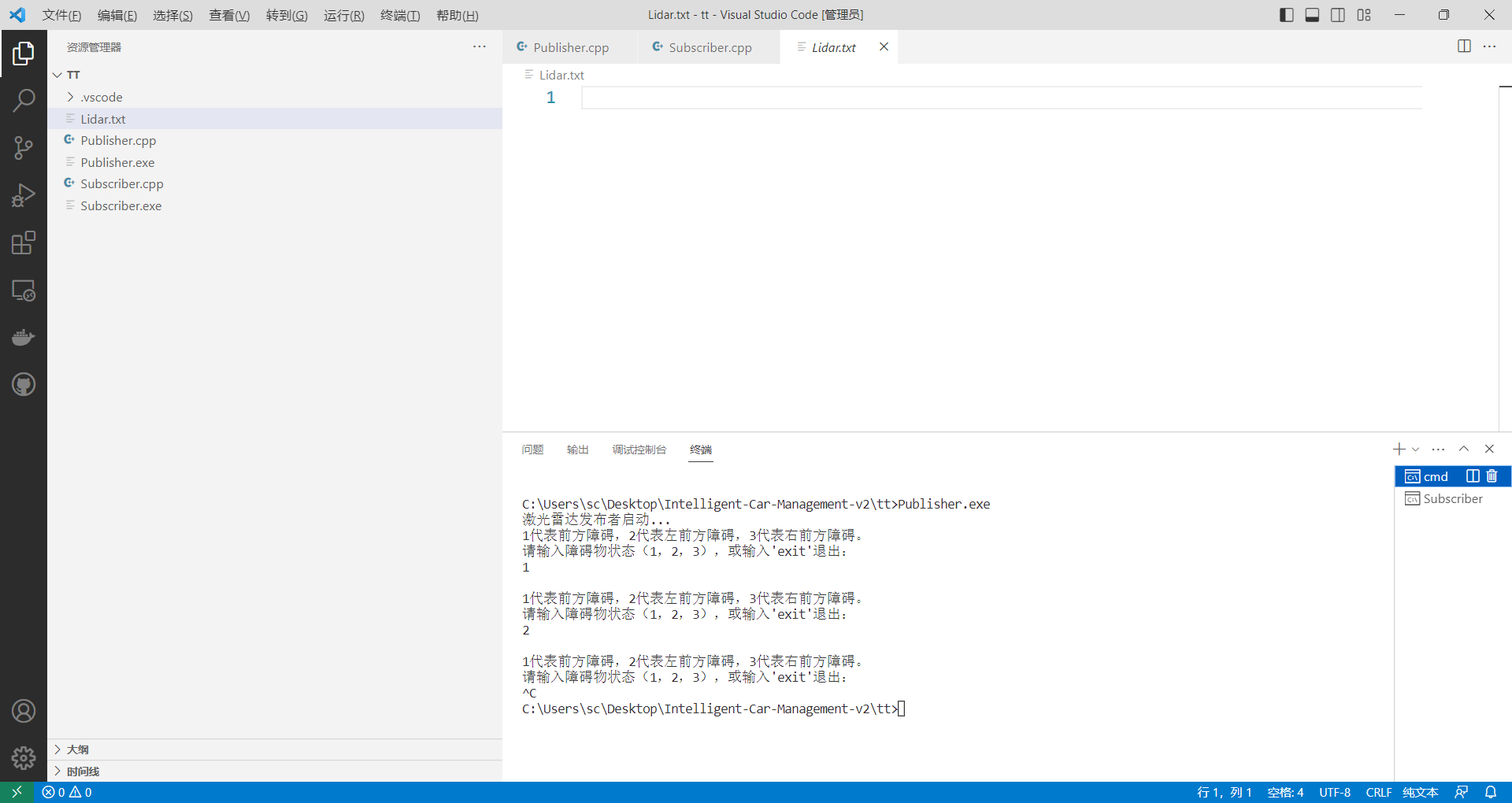


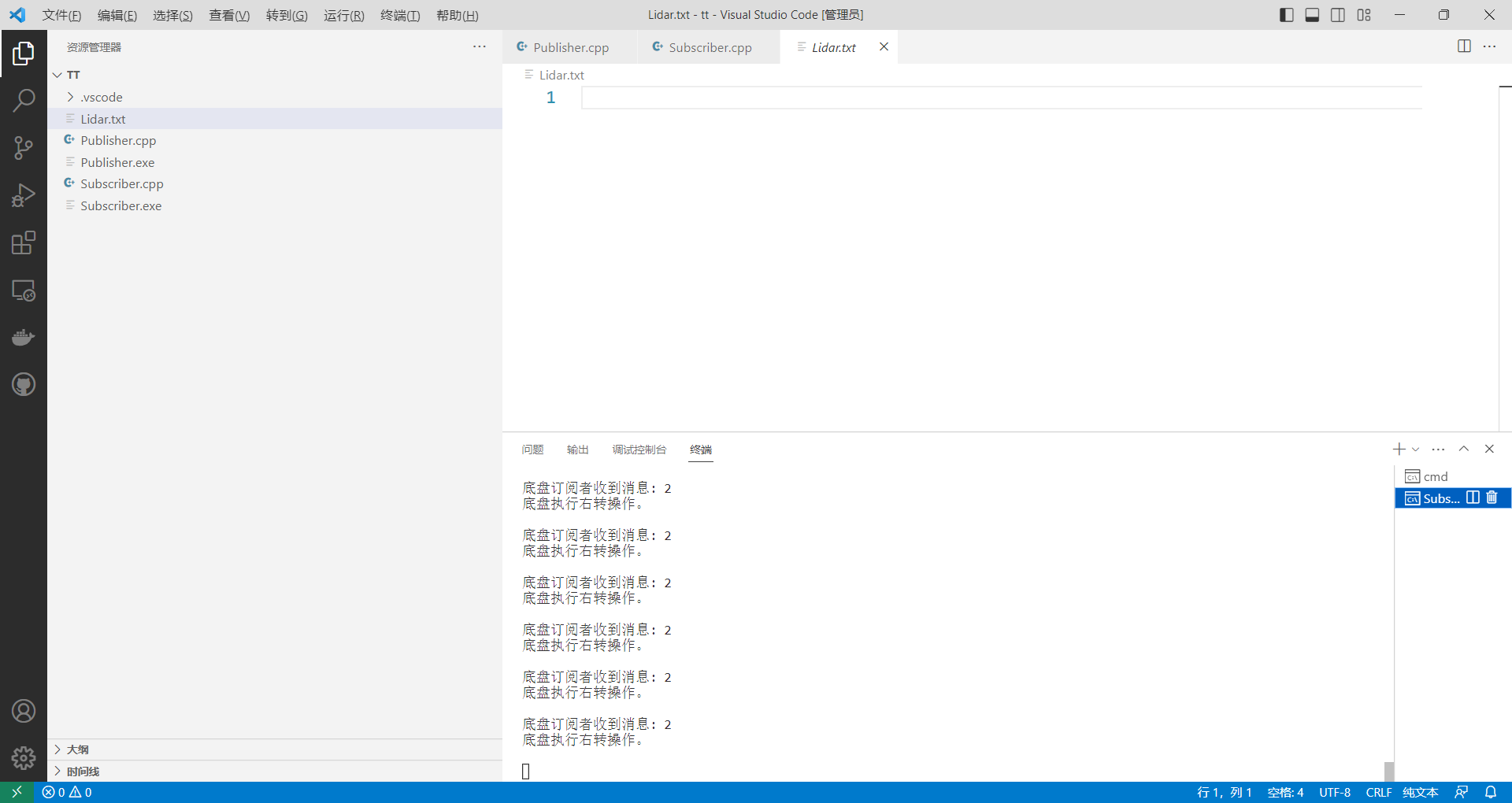
当我们在终端一中再次输入2时，会发现Lidar.txt文件内容实时更新成了2，同时终端二也实时接收到了激光雷达的数据，执行了相应操作。激光雷达发送的数据为2，也就是代表左前方有障碍物，底盘执行右转操作。



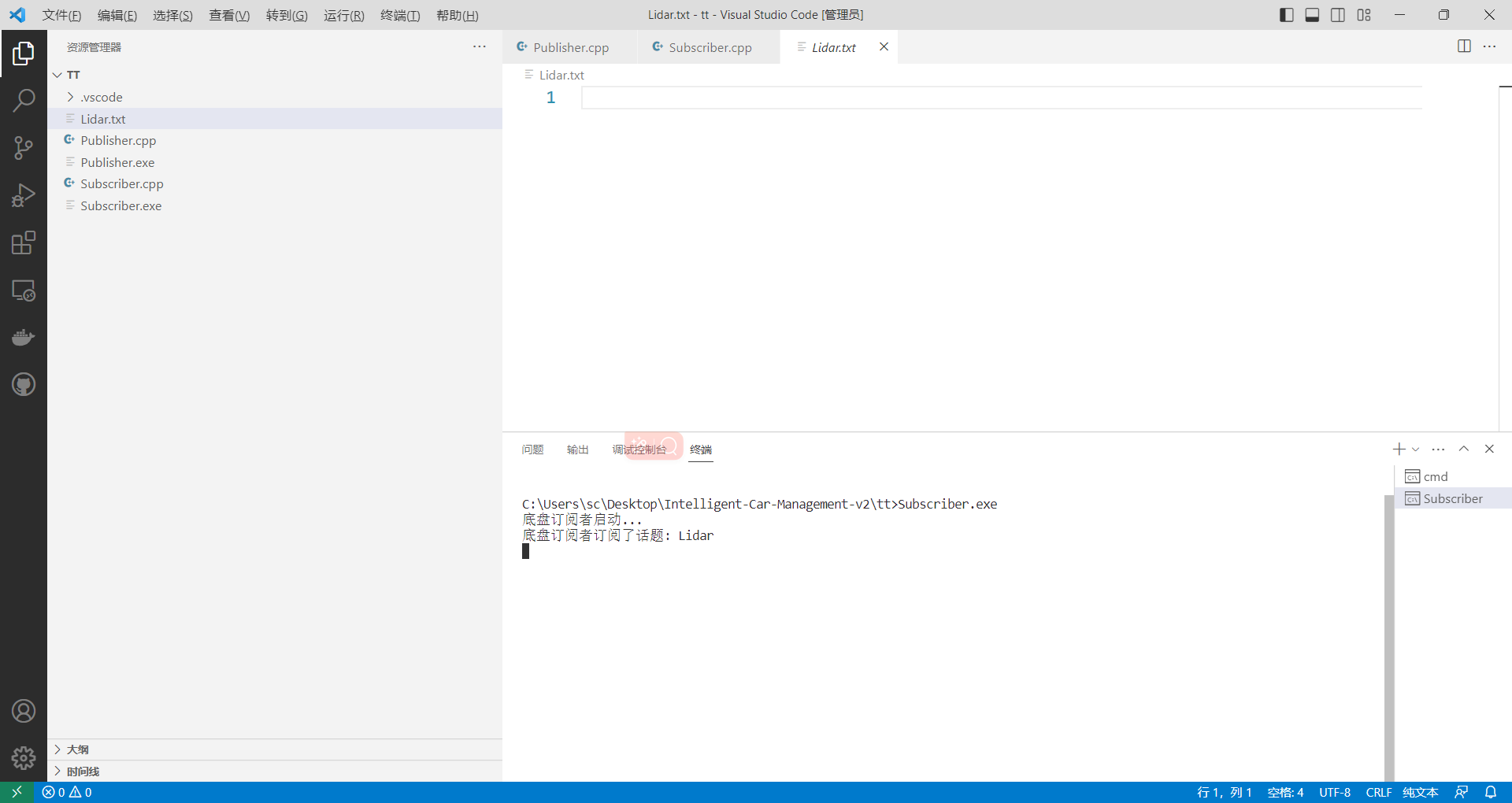


当我们终止掉终端一的程序时，也就是终止激光雷达模块，此时终端二也就是底盘模块不再有数据刷新了。这也是符合要求的，因为当激光雷达模块终止后，没有数据传输了，那么底盘模块就接收不到数据，自然就不能有数据刷新。





如果我们首先执行Subscriber.exe，当Lidar.txt文件不存在时，终端会提示找不到这样一个文件，当Lidar.txt文件存在时，终端提示底盘模块正常启动，但是终端不会有数据刷新，因为激光雷达模块没有启动。



1. **总结分析**

程序的实际效果和理想效果完全符合，完成了预期功能。两个程序可以独立运行，Publisher.exe和Subscriber.exe的执行顺序也没有要求，符合发布者-订阅者思想。