

车载软件开发基础

课后实践3



**2022至2023学年第 1 学期**

|  |  |
| --- | --- |
| 学号 | 姓名 |
| E2021244 | 戴兴龙 |
| 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**

**1.1 项目源码及代码仓库地址**

代码托管在Github：<https://github.com/daixll/EIE/tree/master/车载软件开发基础/课后实践3-设计模式>

在这里也附一份源代码：



**1.2 组成程序的模块**

程序包含以下三个主要模块：激光雷达（Lidar）、底盘（Chassis）和事件轮询（EventPoll）。它们在控制系统中分别起到了不同的作用，通过彼此协作完成整体功能。以下是对代码中每个模块的详细分析：

激光雷达模块 (Lidar)

功能：负责感知环境。激光雷达通常用于获取周围的距离数据或对象检测信息。

在代码中的作用：Lidar 通过 EventPoll 被持续监控，并在检测到新的数据或事件时，将信息传递给 Chassis。

这里使用 std::make\_shared<Lidar>() 创建了一个 Lidar 对象，采用共享指针 (std::shared\_ptr) 来管理其生命周期，允许多个对象共享该实例。

底盘模块 (Chassis)

功能：作为车辆的底盘，负责接收外部传感器（如激光雷达）的信息并进行处理，比如控制运动或执行其他任务。

在代码中的作用：Chassis 对象通过 deal() 函数处理激光雷达传递来的状态信息。

这里使用 std::make\_unique<Chassis>() 创建了一个 Chassis 对象，采用唯一指针 (std::unique\_ptr)，表明它是独占的，不会被其他对象共享。

事件轮询模块 (EventPoll)

功能：负责事件的注册与轮询。它定期检查或等待激光雷达是否有新的数据或状态，并将这些信息传递给底盘模块。

在代码中的作用：EventPoll 持有激光雷达对象，负责不断监听激光雷达的数据，处理事件，并返回状态信息。

使用 std::make\_unique<EventPoll>(l) 创建了一个 EventPoll 对象，EventPoll 持有一个指向 Lidar 的共享指针，并负责轮询激光雷达的状态。

主循环

功能：主循环负责持续监听事件并将事件交给底盘进行处理。

在代码中的作用：代码中的 while (true) 是一个无限循环，用于不断地调用 EventPoll 的 wait() 函数。wait() 函数会返回一个包含事件或状态的向量 (std::vector)，每个状态依次被传递给 Chassis 进行处理。

代码分析：

e->wait() 等待来自激光雷达的事件，返回一个状态向量 (std::vector)。 for (auto& s : v) 迭代状态向量中的每一个状态，交由 Chassis 的 deal() 方法处理。

**1.3 设计模式**

观察者模式 (Observer Pattern)

定义：观察者模式定义了对象之间的一对多依赖关系，当一个对象状态发生变化时，所有依赖于它的对象都会收到通知并自动更新。

在代码中的体现：激光雷达 (Lidar) 和事件轮询 (EventPoll) 之间的关系可以被看作是一种观察者模式。EventPoll 作为观察者持续监听 Lidar 的状态变化。当 Lidar 产生新数据或发生事件时，EventPoll 会获取到这些信息并传递给其他对象（如 Chassis）。

EventPoll 持有 Lidar 的共享指针，并且通过 wait() 函数获取事件，这种关系符合观察者模式的结构，Lidar 类似于被观察者。

**1.4 关键算法**

事件轮询算法

算法目标：持续监听传感器（激光雷达）的状态变化，获取事件并返回给调用者进行处理。

算法描述：EventPoll::wait() 方法是事件轮询的关键部分。它的任务是不断等待激光雷达（或其他传感器）的事件，并在事件发生时将其状态以向量形式返回。

具体的算法步骤可能如下：

①初始化事件监听：系统进入等待状态，等待传感器（Lidar）产生新的数据或事件。

②事件捕获：一旦有新的事件发生，EventPoll::wait() 将捕获这些事件。

③返回事件集合：将所有捕获到的事件打包为向量，并返回给调用者（主循环）。

**2 程序效果展示**



图 1程序运行效果

如图1所示，程序运行之后，通过激光雷达发布的信息，打印出相应的操作。

**3 总结和分析**

该项目通过多个模块的协作实现了一个典型的传感器-控制系统架构，主要功能是持续监听激光雷达的状态，并根据获取到的状态驱动底盘做出相应的处理。整个系统围绕事件轮询和状态处理两大核心机制，具有较高的模块化设计和灵活性。

核心模块

激光雷达 (Lidar)：

激光雷达作为传感器，负责检测外部环境状态。它是状态变化的源头，并通过事件轮询机制传递信息给其他模块。

事件轮询 (EventPoll)：

事件轮询是系统的关键，它持续监听传感器的状态并等待新事件发生。事件轮询将激光雷达的状态作为事件打包并传递给处理单元，起到了“中介”作用。

底盘 (Chassis)：

底盘是执行者，它根据传感器传递过来的状态信息进行处理。根据不同的状态，底盘可以执行不同的动作，例如停止、转向或前进。

设计模式分析

该项目利用了多种经典的设计模式来提高系统的可扩展性和维护性：

观察者模式用于管理传感器（激光雷达）状态的监听，确保事件发生时能够通知到事件轮询模块。

策略模式通过底盘的 deal() 函数体现，底盘根据不同的状态执行相应的策略操作。

责任链模式在事件轮询和状态处理之间存在，每个事件从传感器到执行者的处理是逐步进行的。

中介者模式则通过 EventPoll 模块实现，负责协调传感器和底盘之间的通信。

关键算法

从算法角度来看，事件轮询和状态处理是整个系统的核心：

事件轮询算法：持续监听激光雷达的状态变化并将事件返回，确保系统能够实时响应外部环境。

状态处理算法：底盘根据不同的传感器状态做出相应的反应，执行不同的行为操作。

主循环算法：系统的主控制逻辑，通过不断调用事件轮询和状态处理模块来保持系统的实时性。

代码优势

模块化设计：代码的设计高度模块化，传感器、事件处理、执行器各自独立运行，降低了模块之间的耦合。

可扩展性：如果需要添加新的传感器或新的状态处理逻辑，可以在不改变原有系统架构的情况下轻松扩展。

实时响应能力：通过持续的事件轮询和状态处理，系统能够及时对外界环境做出反应。

改进空间

异常处理：当前的设计中没有明确处理异常或错误的部分，例如当传感器出现故障时如何响应。可以考虑加入异常处理机制，以提高系统的鲁棒性。

多传感器支持：当前系统设计只处理了一个传感器（激光雷达）的数据，未来可以扩展以支持多个传感器的数据融合。

总结

这段代码通过模块化设计和多种设计模式的应用，实现了一个实时性强、可扩展性高的传感器-执行系统。事件轮询和状态处理是其关键算法，保证了系统能够根据传感器反馈及时做出相应的动作。