分类号：_____ 密级：_____

UDC：_____ 编号：_____

工学硕士学位论文

面向地形勘察使命的AUV群体协同使命控制方法研究

硕士研究生：高晗

指导教师：王宏健 教授

学科、专业：控制理论与控制工程

论文主审人：李 娟 副教授

哈尔滨工程大学

2012年3月

哈尔滨工程大学
学位论文原创性声明

本人郑重声明：本论文的所有工作，是在导师的指导下，由作者本人独立完成的。有关观点、方法、数据和文献的引用已在文中指出，并与参考文献相对应。除文中已注明引用的内容外，本论文不包含任何其他个人或集体已经公开发表的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者（签字）：高略

日期：2012 年 3月9 日

哈尔滨工程大学
学位论文授权使用声明

本人完全了解学校保护知识产权的有关规定，即研究生在校攻读学位期间论文工作的知识产权属于哈尔滨工程大学。哈尔滨工程大学有权保留并向国家有关部门或机构送交论文的复印件。本人允许哈尔滨工程大学将论文的部分或全部内容编入有关数据库进行检索，可采用影印、缩印或扫描等复制手段保存和汇编本学位论文，可以公布论文的全部内容。同时本人保证毕业后结合学位论文研究课题再撰写的论文一律注明作者第一署名单位为哈尔滨工程大学。涉密学位论文待解密后适用本声明。

本论文（☑在授予学位后即可　□在授予学位12个月后　□解密后）由哈尔滨工程大学送交有关部门进行保存、汇编等。

作者（签字）：高略　　导师（签字）：王宏建

日期：2012 年 3月9 日　　2012 年 3月9 日

分类号：_____ 密级：_____

UDC：_____ 编号：_____

工学硕士学位论文

面向地形勘察使命的AUV群体协同使命控制方法研究

硕士研究生：高 晗

指 导 教 师：王宏健 教授

学 位 级 别：工学硕士

学 科、专 业：控制理论与控制工程

所 在 单 位：自动化学院

论文提交日期：2012年1月

论文答辩日期：2012年3月

学位授予单位：哈尔滨工程大学

Classified Index:

U.D.C:

A Dissertation for the Degree of M. Eng

Research on Swarm Cooperation Mission Control Method of AUV Terrain Survey

Candidate: Gao Han

Supervisor: Prof. Wang Hongjian

Academic Degree Applied for: Master of Engineering

Specialty: Control Theory and Control Engineering

Date of Submission: Jan,2012

Date of Oral Examination: Mar,2012

University: Harbin Engineering University

摘　　要

自主水下航行器（AUV）可替代人类深入海洋，完成危险作业，且以其成本低、智能化程度高等优势，已经越来越多地应用于海洋勘探、资源开发以及军事应用等领域。特别是通过多个 AUV 的相互配合，实现群体协同作业，大大提高了作业效率。本文以执行海底地形勘察使命为背景，针对多 AUV 群体协同使命控制技术，开展了多 AUV 群体协同体系结构设计、AUV 使命控制系统建模、使命控制方法与仿真平台构建等方面的研究。

协同体系结构是多 AUV 群体协同使命控制的基础。本文基于经典多机器人系统体系结构分析，设计了一种集中式多 AUV 体系结构，该结构由一个主控 AUV 和若干作业 AUV 组成。设计出了各个单元的功能模块以及模块间的信息流，建立了多 AUV 系统模型。

采用进程描述主控 AUV 和作业 AUV，从逻辑上对群体中的各个 AUV 单元进行划分。设计多 AUV 群体协同地形勘察使命的划分方式，主控 AUV 将勘察使命分解为可供作业 AUV 执行的任务，并将其分配给各个作业 AUV；作业 AUV 进一步将所分配到的任务划分为顺序执行的子任务，并对任务执行过程中出现的事件进行响应。采用线程机制设计了子任务执行和事件响应模块，设计了利用线程描述的子任务与事件处理过程的实现方法

建立了使命控制系统的 Petri 网模型，基于此模型设计了适用于 AUV 群体完成地形勘察的使命控制方法，采用基于优先级的抢占式多线程调度方式，实现了作业 AUV 的子任务能够有序执行，以及对作业 AUV 各项内部和外部事件的快速正确响应。设计了基于仲裁规则的多 AUV 群体协同使命重规划方法，当某一作业 AUV 出现故障无法完成自身任务时，主控 AUV 将根据情况派遣能够正常工作的作业 AUV 完成协同任务。

基于 Linux 系统的 C 语言和 Qt Creator 开发平台构建了多 AUV 群体协同使命控制仿真平台，通过人机交互设置地形勘察使命和多 AUV 初始作业信息，实现了 AUV 群体协同系统及其使命控制方法的仿真验证。结果表明：基于多进程和多线程机制所构建的多 AUV 群体协同体系结构通过子任务调度满足了使命作业的需求，且出现系统事件时系统能够完成自主使命控制和使命重规划，实现了多 AUV 协同作业的目标。

关键词：自主水下航行器；地形勘察；使命控制；多线程；Petri 网

Abstract

The Autonomous Underwater Vehicle (AUV) can finish the dangerous task in the deep-sea instead of human. Because of the low cost and high intelligence, AUVs are more and more used in sea prove, resource exploitation and military affairs. Especially the mulit-AUVs system, can realization swarm cooperation, highly improves the efficiency. In the back of seabed landform reconnaissance, this paper aimed at multi-AUVs swarm coorperation mission control technology, do some researches on multi-AUVs system framework design, AUV mission control system modeling, mission control method and emulator design.

Coorperation framework is the base of multi-AUVs swarm coorperation mission control. On the basis of analysis on the multi-robot system`s framework, this paper presents a centralized multi-AUV framework. This framework is built up by one commander AUV and some warrior AUV. Then designed each AUV`s function module

Use independenct processes describe Commander AUV and every Warrior AUVs, in order to compartmentalize the individual AUV logically in the swarm. Commander AUV devides the mission into tasks which the Warrior AUVs can carry out, then send the tasks to every Warrior AUV; the Warrior AUV farther devides the task into subtask. Every subtask and event management is described by thread.

Build up the Petri Net module of mission control system; use the method of priority-based preemptive multi-thread scheduling. Realize the AUV swarm coorperation mission control through the sequentially implement of subtask and response of event with different priority in Warrior AUVs.

Based on C language in Linux operation system and Qt Creator, we build a multi-AUVs swarm coorperation mission control emulator platform. The mission is initialized through Human-computer interaction. The emulator platform successfully tests the multi-AUVs system`s and validates the mission control method. The result shows that, the multi-AUVs system built on the basis of multi-process and multi-threads can satisfy the require of mission implement through subtask scheduling, when the system event is come out the system could finish self-determination mission control and mission replan, achieve the goal of multi-AUVs coorperation operate.

Key words: Autonomous Underwater Vehicle; landform reconnaissance; mission control; multi-process; multi-threads; Petri net

目　　录

第 1 章 绪论

1.1 研究的目的和意义

海洋资源的开发有着广阔的前景。海洋中的生物、矿产、能源和空间等，都是各国争相开发的对象。由于海洋环境恶劣复杂，人类很难进行实地勘测，因此，无人水下潜器，尤其是 AUV（Automatic Underwater Vehicle——自主水下航行器）为海底勘测提供了高效便捷的手段。目前，世界上很多国家正在对 AUV 进行研究，且很多已经取得了丰硕的成果，推出了适应于诸如海底地形勘察、水文环境勘察和渔业资源勘察等用途的 AUV 产品。

由于单个 AUV 工作的局限性，科学家开始研究多水下机器人系统(Multiple Autonomous Underwater Vehicle System，简称 MAUVS 或多 AUV 系统)以应对更加复杂的海洋环境和更加繁重的勘测任务。多 AUV 系统概念在 20 世纪 80 年代被提出，多 AUV 技术作为海洋智能机器人技术和机器人学发展到一定阶段的产物将会得到广泛应用，特别是多 AUV 系统能够通过相互合作与协调完成复杂的水下作业任务，系统所具有的高度并行性、冗余性等是单个水下机器系统所无法比拟的，例如利用多 AUV 系统对指定的海洋区域进行勘测，可以大大提高工作效率[1]。

在 AUV 的研究中，使命控制是必不可少的一个环节。使命是 AUV 需执行的任务的集合，使命控制的作用是将 AUV 需要执行的任务按照一定规则进行排序，使 AUV 的作业能够有序进行，更重要的是 AUV 要能够对作业过程中遇到的各种事件，如行进路线上出现障碍、AUV 硬件故障等进行正确响应，以保证使命的顺利完成。

1.2 国内外研究现状

Coral 是 1994 年葡萄牙的 ISR 为 Marius AUV 设计的使命控制系统。Coral 具有很多系统部件，例如水声通信系统，避障检测系统，导航系统等。每个系统部件负责控制 AUV 的一些功能，并且具有一个基于信息交互的通信机制。一个使命由一些潜器基本单元（Vehicle Primitive，简称为 VP）进行描述，VP 是被参数化描述的操作。当 VP 被执行时，一个或更多的系统任务处于活跃状态并同步。然而，VP 不能任意执行，在执行前它们要满足一些前提条件并触发一些后续的状态。因此，用来定义使命的 VP 们可以通过 Petri 网链接和表达[2,3,4]。

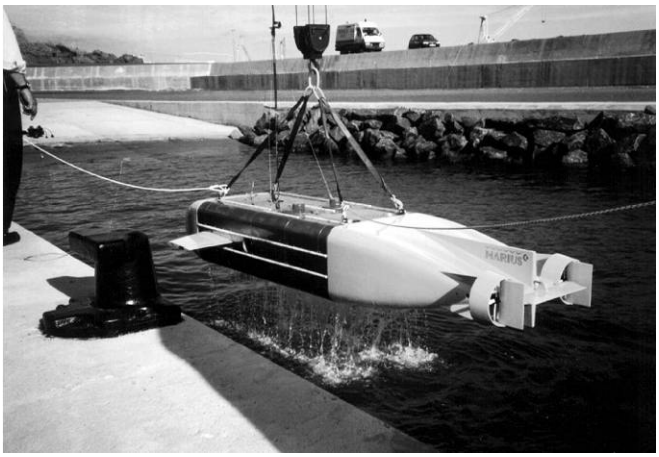


图 1.1　MARIUS AUV

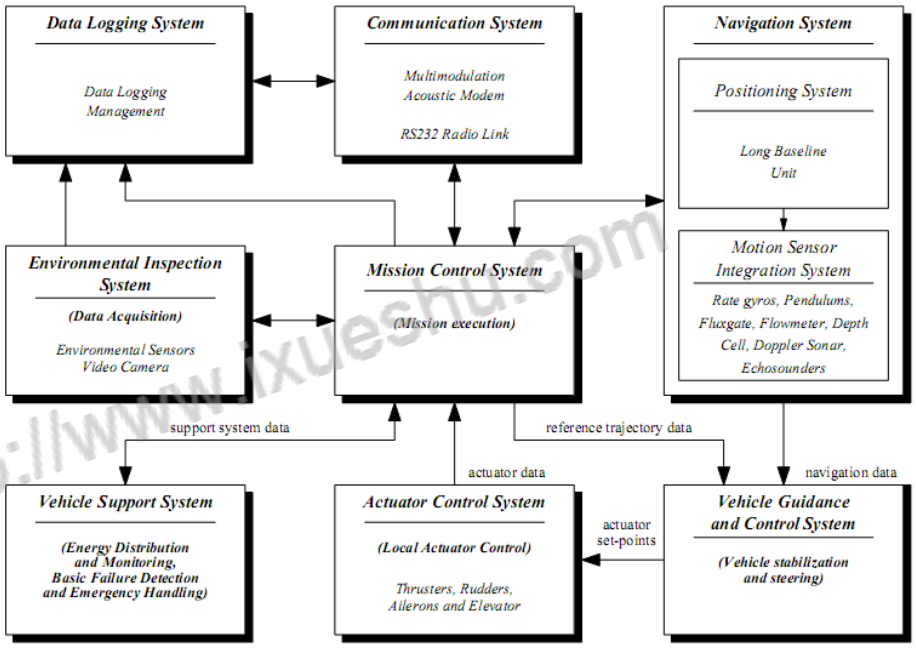


图 1.2　MARIUS AUV 控制结构

Starfish 型 AUV 的使命控制基于 C2 系统（Command and Control System）。它采用了一种协商——反应结构并包含一系列相互作用的智能组件，用以描绘命令的不同等级。一个 C2 系统执行的使命内容包括规划、定位、管理和控制 AUV 上的多种动作。它从传感器接收处理过的数据，向 AUV 的推进器或控制系统输出控制指令来产生期望的行为，从而在保证 AUV 安全的情况下完成使命目标[5,6]。

图 1.3　Starfish AUV

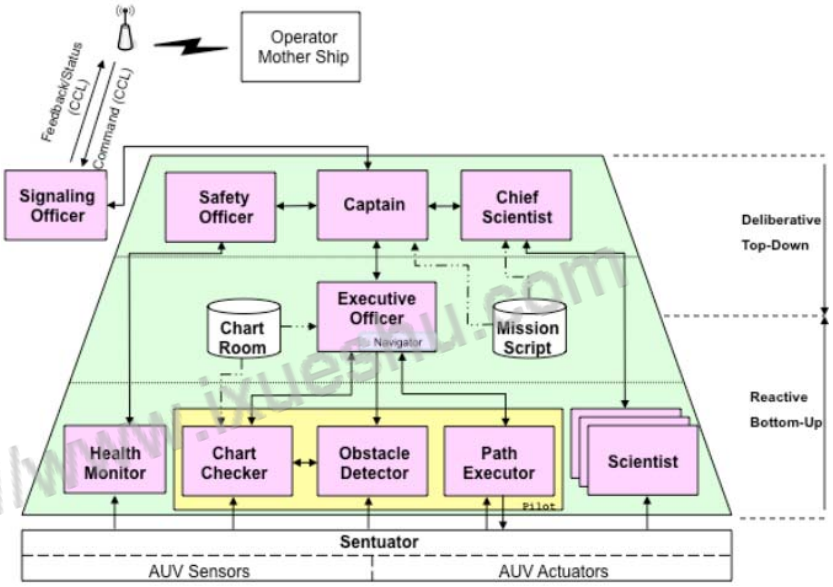


图 1.4　Starfish AUV 使命控制结构

Phoenix 型 AUV 的使命控制基于推理行为模型（Rational Behavior Model），其结构包括三个层次：战略层、执行层和战术层。系统中的运动控制部分被包含在执行层中，负责控制 AUV 的运动和稳定性；执行层对 AUV 的执行机构进行实时控制。离散事件系统通过将 AUV 使命执行的基本行为定义为相应的基本单元，并采用规则语言完成这些基本单元的排序控制与有序链接，这一工作在战略层中完成。战术层通过执行层与战略层相连接，其基本功能是根据从伺服层收集到的数据，来判断基本单元的执行是否完成，并从传感器提取信息[7]。

图 1.5　Phoenix AUV

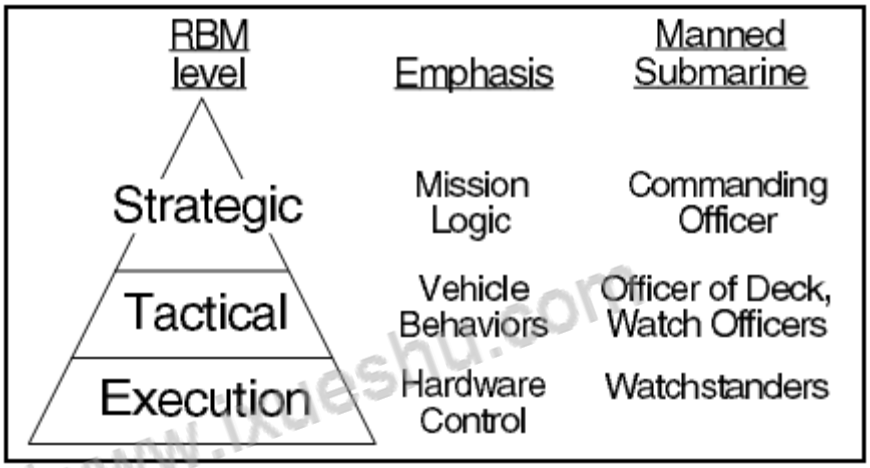


图 1.6　Phoenix AUV 采用的 RBM 结构

Odyssey 型 AUV 的使命控制系统 Helm 是基于 MOOS 结构的。MOOS 包含一系列用于控制机器人（陆上或水下）的库和可执行模块。它具有相互独立的部件和一个黑板，工作时，所有部件将它们收集到的信息写在黑板上，并且从黑板上获得它们工作所需的数据。这些部件可以是传感器，执行机构或高级任务模块，例如领航员模块从多个传感器获得异步的数据并计算出一个最新的位置和速度估计，电子自动记录器模块将黑板上或 Helm 中的所有消息制作成一个报告文档。Helm 从领航员部件收集导航数据，从其他部件收集其他数据。使命目标通过决定执行何种潜器动作来完成。为实现这一目标，Helm 用一系列具有优先级的基本任务描述经过规划的使命。每个基本单元需要一个激活信息才能开始执行并在其结束时发出一个信息。基本单元在达到其目标，或检测到超时，或没有收到与其相关的部件的输入信号时将结束。一个结束信息可以作用于 0 个，1 个或多个基本单元。同样，其他部件的信息也可以激活一个基本单元。当两个或更多基本单元尝试使用同一个资源时，优先级更高者将获得该资源。以上这些规则使得基本单元和接口部件允许用一种图的形式来定义使命[8,9]。

图 1.7　Odyssey AUV

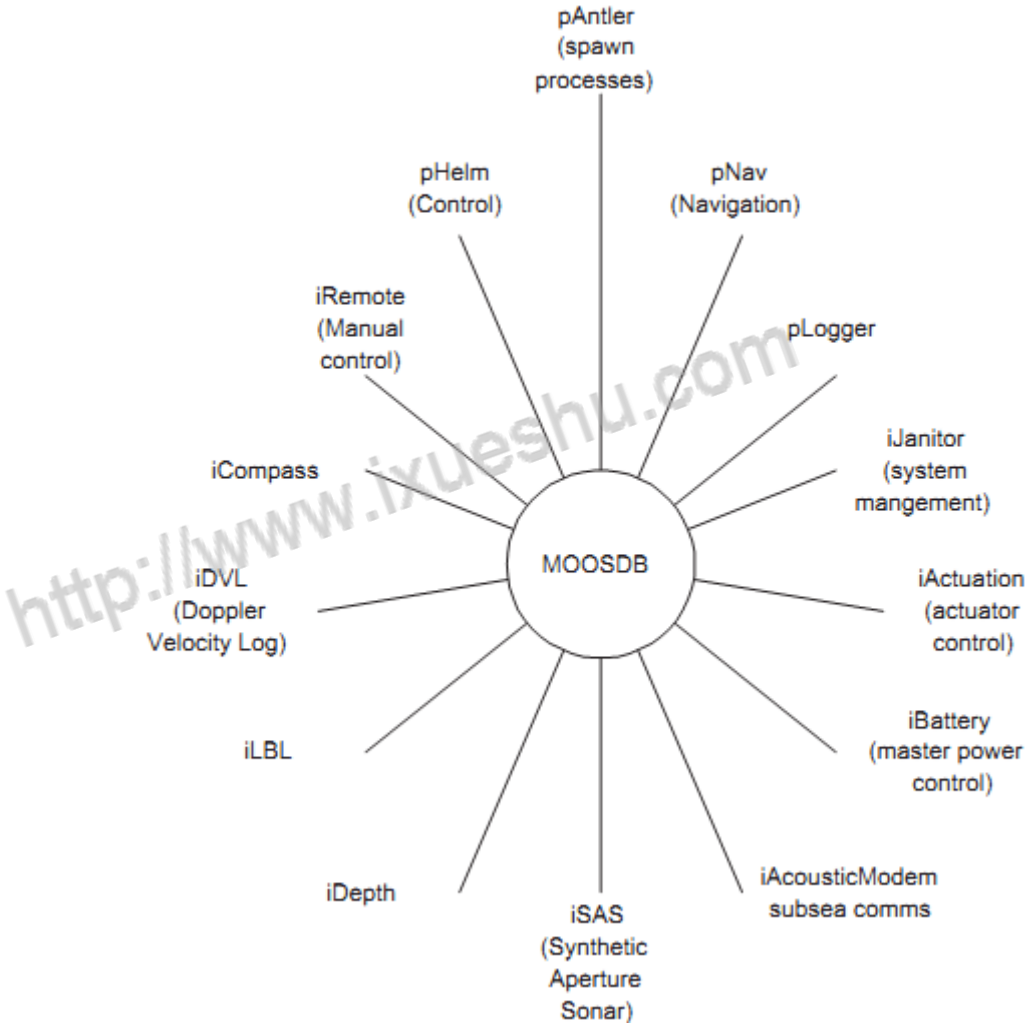


图 1.8　用于 AUV 控制的 MOOS 结构

SAUVIM 型 AUV 的使命控制系统是由夏威夷大学设计的 STDL(The Sauvim Task Description Language，SAUVIM 任务描述语言)。这一使命控制系统有一个记录基本动作的库，可以使用与 C 语言相似的命令语言来执行。库中包含的基本单元具有运动指令、

用于控制特定硬件输入/输出的指令和特定作业指令例如深度或速度控制。这种语言用于实现这些有基本功能的基本单元例如数值操作，状态指令，环路指令，操作指令，变量排序，数学函数等。使用 SAUVIM 任务描述语言，使命规划形式与 C 语言程序相同，且需要预测所有可能出现的情况，包括错误情况。SAVUIM 采用混合的控制方法，包括三个层次：规划、控制和执行。将使命划分为一组子任务。每个子任务都被安排了一系列预处理程序任务单元以保证子任务目标的实现。任务单元是本控制方法的基本组成模块，可以动态地被任务处理程序排序。这样，系统变得可分级，能够在保持对动态环境快速反应的情况下给出可预测的输出[10]。

图 1.9　SAUVIM AUV

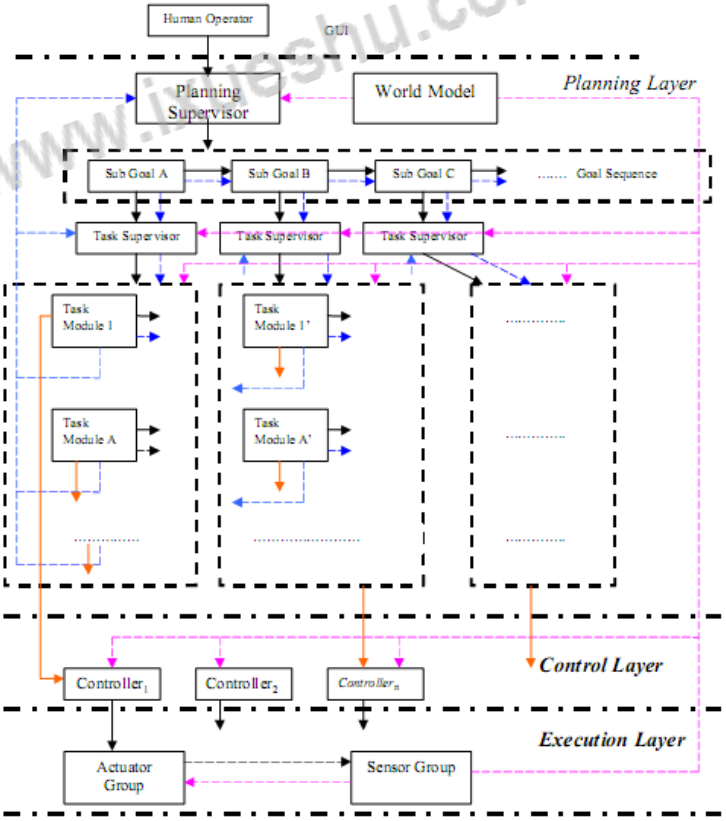


图 1.10　SAUVIM AUV 使命控制结构

ICTINEU 型 AUV 的使命控制基于一种混合协商——反应式控制结构 O^2CA^2。这意味着该结构意图兼顾基于协商和基于行为两种结构的优点并减少两种结构各自的缺点。机器人控制在潜器级执行，机器人导航在任务级执行。这两级构成一个嵌套的控制环路，具有一个内部的潜器控制环和一个外部的导航控制环，这里行为控制系统负责机器人导航。最后使命级（上层）负责定义一系列用来完成使命的依次执行的任务。并定义需要完成每个任务的任务级的配置信息。这样，O^2CA^2 控制结构将因任务级的结构表现出反应式特征，因使命级的结构表现出协商式特征。机器人行为包括在任务级的潜器基本单元中，使命级将用它们来完成每个任务[11]。

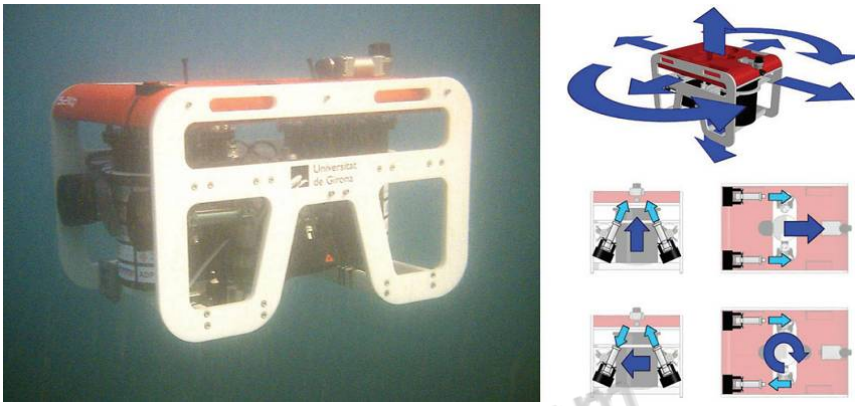


图 1.11　ICTINEU AUV

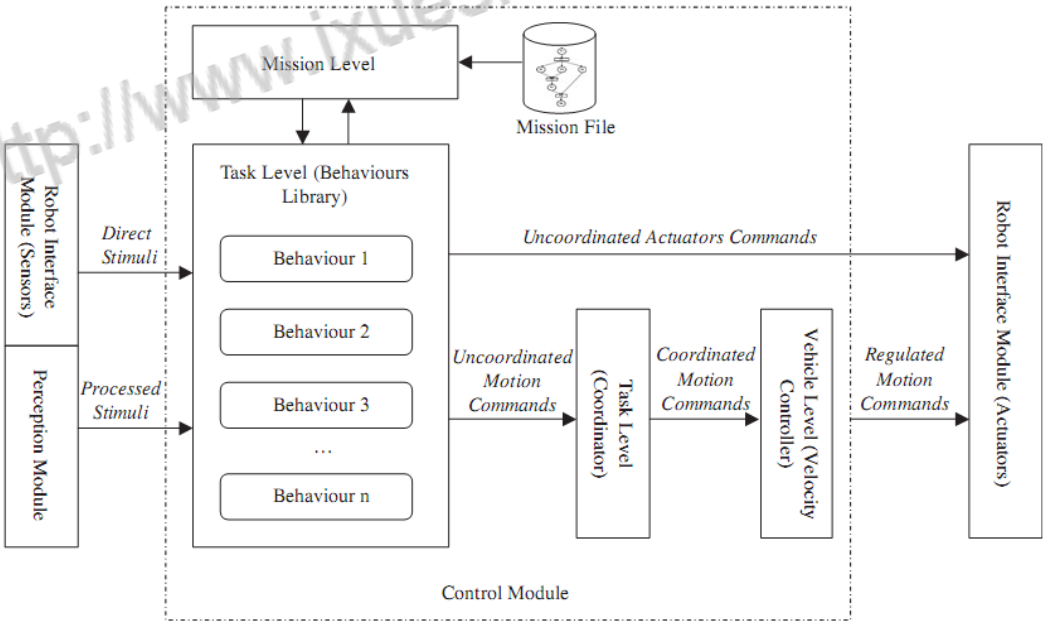


图 1.12　ICTINEU AUV 使命控制结构

在 AUV 使命规划程序的设计中，层次化、模块化等已成为趋势。如在对 Autosub 型 AUV 的设计中，把结构化设计作为了多任务系统设计的关键——把系统分割成分别执行的任务。这样做的优点是可以更有效的利用中央处理器。当一个任务暂停等待事件

发生时，另一个任务可以使用中央处理器资源，且可以简化系统。异步的进程或者以不同频率重复的进程可以以离散的方式更方便地编程[12]。

多 AUV 间的任务分配、通讯、全局和局部路径规划和 AUV 之间以及 AUV 与障碍物之间的避碰等问题比之单 AUV 其复杂程度大大提高。吴小平[13]、崔荣鑫[14]等均提出了基于虚拟参考点的 AUV 的编队方式，即以系统中各个 AUV 所处位置为顶点做出多边形，以多边形的质心为参考点，基于该参考点来计算各个 AUV 的相对与绝对位置，进而对海洋覆盖任务进行规划和重规划。在对作业区域环境部分已知的多 AUV 地形勘察任务中，可以在任务开始之前对任务进行分配，即各个 AUV 有自己的勘察区域，通过一定的算法使未被覆盖的区域尽可能的少。Yeun-Soo Jung[15]在他们的研究中针对多 AUV 海洋覆盖任务，提出了多 AUV 角色变换配合覆盖法：任务开始时不是所有的 AUV 都分配有任务。开始仅有一个 AUV 接受扫描任务并开始探测。一旦它遇到海湾，其它 AUV 将依次接到覆盖任务并进入海湾，如果一个执行覆盖任务的 AUV 完成了覆盖而执行扫描任务的 AUV 尚未找到另一个海湾的入口时，它们将等待。从而实现 AUV 之间的避碰与目标区域的完全覆盖。

局部路径规划方面，田广[16]等提出了一种基于行为的避障和趋向目标方法。根据行为动力学原理，设计了水平方面的趋向目标行为模块和避障行为模块，利用水平面的宏行为实现了水平面行为模块的融合。将虚拟前视声呐按相隔 15°的间距划分为 6 个扇形区域，碍物的信息为二元组，根据 6 个扇区检测到障碍物的情况进行避障。Fodrea, L. R[17]在对 REMUS 避障的研究中指出，REMUS 的局部路径规划是基于任务作业过程中采集的信息。REMUS 用于雷区勘察时，通常采取“剪草机”法来遍历整个区域，水雷形状物体的位置都是根据实时检测数据更新的，根据前视声呐采集的障碍物信息进行避障。

1.3 主要研究内容与研究方法

（1）多 AUV 群体体系结构设计

设计多 AUV 系统的层次结构。体系结构的设计直接关系到系统中每个单元具有的功能，为下一步 AUV 单元的仿真建模提供依据。通过对集中式、分布式、分层式等典型体系结构对比分析，并结合系统需求，选择合适的体系结构构建系统的总体框架。

（2）AUV 群体仿真平台构建

根据所构建的体系结构，设计系统中不同层次和功能的 AUV 单元。各个 AUV 单元使用不同的程序进行描述，从而在物理上将各个 AUV 分开。针对所构建的 AUV 群体体系结构，基于仿真通信实现 AUV 群体内部的协同交互与消息共享，设计有效的群体协同通讯机制，保证系统中各个单元间的信息传递不出现阻塞。

（3）面向地形勘察使命的 AUV 群体协同使命控制

采用仿真的方法设计地形勘察典型案例，研究面向使命规划结果的在线使命控制方法，包括：子任务调度、事件感知与仲裁、行为（重）规划与控制等；研究 AUV 群体协作机制与冲突消解策略，以安全、高效为 AUV 群体协同使命控制的目标。具体方法是：为 AUV 实时规划出执行使命的子任务序列；针对 AUV 群体所感知的不确定事件，基于事件库的辅助决策完成子任务控制过程中的在线事件仲裁，形成安全、合理的行为序列；基于一定的优化策略完成 AUV 群体地形勘察（重）规划与控制，实现 AUV 群体基于任务的协作；

（4）仿真验证

仿真平台程序设计的主要目的是验证所设计方法的正确性，验证无误后方可进行实际应用。因此本程序需要设计友好美观的程序界面，便于使用者对多 AUV 系统进行初始化，系统各个单元在使命执行过程中的状态也应清晰显示，便于根据界面显示信评价使命控制方法是否科学有效；若方法不够完善也可以明确显示出问题所在，便于程序设计者对方法进行改进和补充。

1.4 论文的组织结构

本文共分为五个章节，各个章节主要介绍的内容为：

第一章为绪论，对课题研究的背景、目的和意义进行了介绍。分析了相关领域的国内外研究现状，对本课题的主要研究内容和方法进行了简要说明。给出了论文的组织结构，引导课题研究内容的介绍和展开。

第二章为多 AUV 群体体系结构与功能模块设计。这一章对多机器人系统的几种经典体系结构进行了分析，选择适合地形勘察使命的结构构建系统框架。针对选择的框架设计多 AUV 系统中各个单元的职能和功能，对各个 AUV 单元的结构和功能模块进行了详细的介绍，为多 AUV 群体协同使命控制的实验搭建平台。

第三章为基于多进程和多线程机制的 AUV 群体协同仿真平台构建。首先介绍了地形勘察使命的划分层次与分解方法，设计执行使命的具体步骤。接着介绍了使用不同的进程描述主控 AUV 和作业 AUV，通过进程间通信实现多 AUV 系统中的信息交互的实现方法。最后介绍了使用不同的线程描述 AUV 内部的各个功能模块与使命具体执行过程的方法，并给出了多 AUV 系统中各个单元间的通信协议。

第四章为基于 Petri 网的多 AUV 群体协同使命控制。介绍了多 AUV 系统中各个单元使命控制过程的 Petri 网模型。基于此模型设计了使命控制实现方法，调度各个子任务和事件处理线程，完成使命执行过程。

第五章为多 AUV 群体协同使命控制仿真实现。针对具体地形勘察使命仿真实验，以程序截图结合说明的方式给出了课题各部分研究内容的验证结果，包括系统基本功能

测试、使命控制方法以及使命重规划的正确性验证。

第 2 章 多 AUV 群体体系结构与功能模块设计

2.1 引言

多 AUV 群体体系结构的设计关系到了系统中各个单元所在的层次与功能的划分，是各个 AUV 单元结构设计的前提。AUV 单元的体系结构决定了使命的划分方式、信息流的内容以及使命执行的具体步骤。因此，多 AUV 系统体系结构的设计应遵循以下原则：

（1)清晰的层次关系。层次关系包括群体层次和 AUV 内部功能模块的层次两部分。清晰的群体层次关系便于系统中各个单元的分别设计与最终集成；AUV 内部功能模块的层次明确有利于系统设计的规范化与模块化，便于系统的剪裁与扩充。

（2）合理的功能分配。应合理规划处于系统中不同层次的 AUV 的功能，避免某些 AUV 功能过多过复杂而另一些 AUV 功能较单一，否则不但会影响系统的整体性能、降低执行效率，甚至会由于数据处理量过大而导致部分 AUV 单元瘫痪。

（3）高效的信息传递。系统结构设计应充分考虑各个 AUV 单元之间以及 AUV 单元内部各个功能模块之间传递信息的内容和格式。设计规范的、简洁的信息流对提高系统执行效率、减少故障有着重要意义。

2.2 多 AUV 系统群体体系结构

多 AUV 系统的组织形式可以分为集中式(Centralized)、分散式(Decentralized)和分布式(Distributed)三种[18]。

（1）集中式(Centralized)控制结构系统

如图 2.1 所示，系统由一个主控单元集中控制整个系统，它是一种规划与决策的自上而下式的层次控制结构，其层次的数量和复杂性决定了系统响应所需的时间和行为决策的质量。系统的协调性较好，但实时性、动态性较差，对环境变化响应能力差，集中式系统由一个主控单元和多个与之在结构上分散的、独立的协作作业 AUV 构成。主控单元负责任务的动态分配与资源的动态调度，协调各作业 AUV 间的竞争与合作。该类系统较易实现系统的管理、控制和调度[19]。

（2）分散式(Decentralized)控制结构系统

如图 2.2 所示，系统中每个个体之间都是平等的关系，各 AUV 具有高度智能自治能力，各 AUV 自行处理信息、自行规划与决策、自行执行自己的任务，与其它 AUV 相互通讯来协调各自行为而没有任何集中控制单元。这种结构有较好的灵活性、可扩展性、故障冗余和可靠性，但对通讯要求较高（多边通信），且多边协商效率较低（各有

各的算法，思路不统一），很难或无法保证全局目标的实现。

（3）分布式(Distributed)控制结构系统

其介于上述两者之间，是一种全局上各 AUV 等同的智能分布——分层式结构而局部集中的结构方式。如图 2.3 所示，这种结构方式是分散式的水平交互与集中式的垂直控制相结合的产物，其由彼此独立、完全平等、无逻辑上的主从关系的、能够自律的一组 AUV 构成。各 AUV 按预先规定的协议，根据系统的目标、状态与自身的状态、能力、资源和知识，利用通信网络相互间协商与谈判，确定各自的任务，协调各自的行为活动，实现资源、知识、信息和功能的共享，协作完成共同的任务以达到整体目标。在该类系统中，各 AUV 在结构和功能上彼此独立，都以同样的方式通过网络通信相互发生关系，具有良好的封装性，因此使系统具有很好的容错性、开放性和扩展性。既提高了协调效率，又不影响系统的实时性、动态性、容错性和扩展性[20,21]。但问题是系统复杂性也大大提升，如何保证 AUV 与主控单元之间以及各个 AUV 之间信息的及时传递，如何根据所获得的信息对使命规划进行实时调整，都是需要解决的问题。

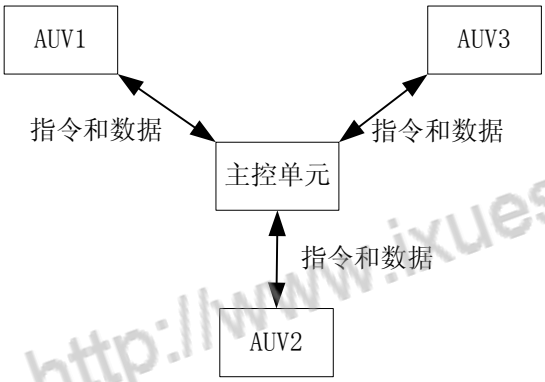


图 2.1　集中式结构

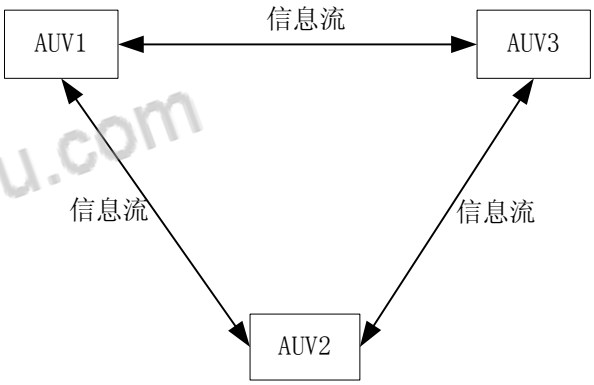


图 2.2　分布式结构

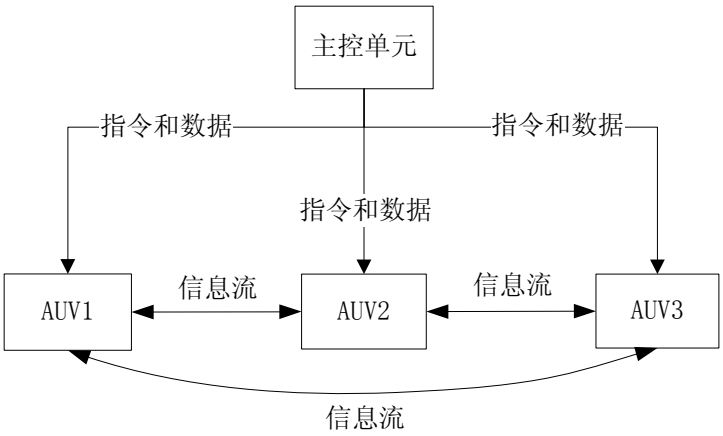


图 2.3　分层式结构

本课题中，若采用分布式结构，使命执行过程中依靠各个 AUV 间的协商和谈判，

完成使命的规划和重规划，并监测其他各个作业 AUV 的状态和使命执行情况，则各个 AUV 单元功能过于复杂，系统设计的难度将大大提高。若采用分层式结构，主控 AUV 仅负责初始化和下达使命信息，作业过程完全由作业 AUV 群体执行和监控，则主控单元的功能过于单一，而作业 AUV 的功能过多；由于作业 AUV 单元的数量较多，则必然造成系统设计难度的提高。故将 AUV 状态监测和使命执行情况监测的任务分配给主控单元更能简化系统结构和复杂度，所以在系统结构的设计上选择集中式结构构建多 AUV 系统。

使命执行过程中，各作业 AUV 以一定时间间隔向主控单元发送当前的速度、坐标和当前时间等信息，主控单元将轮询各个与作业 AUV 通信的端口接收数据，对各个作业 AUV 的状态和使命执行情况进行监测，并在界面上予以显示。若有作业 AUV 发生故障，主控单元对使命进行重规划，并通过一定的仲裁机制选择出执行协同任务的作业 AUV，将重规划信息发送给该作业 AUV 完成协同任务。

2.3 AUV 个体体系结构

2.3.1 主控 AUV

在集中式结构中，通常由地面基站、母船或群体中的某一 AUV 担任主控单元的角色。在本课题中，将主控单元设计为 AUV 的形式，主控 AUV 与作业 AUV 是相互独立的进程[22,23]。主控 AUV 不具备传感器等作业执行部件，仅仅负责多 AUV 系统的使命规划、控制和监测。使命开始前，首先在主控 AUV 单元程序上进行地形勘察使命初始化。这里，使命（mission）是对整体作业的宏观描述，在本课题中，使命描述了待勘察区域以及各个执行作业的 AUV 的起点和回收点等信息。完成初始化后主控 AUV 将使命根据 AUV 的数量进行使命的分解，划分成下达给各个 AUV 的任务，将任务信息发送给各个作业 AUV。任务（task）是使命的子集和具体实现，定义了各个 AUV 初始状态和要实现的目标。使命执行过程中，主控 AUV 定时接收各个作业 AUV 发送来的使命执行信息并显示，在地图上绘制出各个作业 AUV 的路径。若收到作业 AUV 因故障无法完成自身任务的信息，主控 AUV 进行使命重规划，发送协同任务信息，派遣能正常工作的作业 AUV 完成协同任务。主控 AUV 的结构如图 2.4 上半部分所示。

2.3.2 作业 AUV

主控 AUV 将完成初始化的使命信息制作成文本，发送给作业 AUV 单元。作业 AUV 单元中的子任务调度模块负责将接收到的使命信息分解成子任务。子任务（subtask）是使命控制中的最基本单元，只能串行发生，它描述了完成某一动作的具体步骤。完成分解后，子任务被发送给规划控制与仲裁模块，生成子任务序列。子任务序列与感知到的

事件一起参与仲裁，根据其各自的优先级确定执行次序以形成最终动作序列。

事件（event）是在作业 AUV 执行使命的过程中，通过传感器采集环境信息，发送给感知器，由感知器感知到的 AUV 内部和外部的各种突发情况，其特点是无法在使命规划阶段预测到且会对 AUV 的正常作业产生影响，若不能正确做出处理，很可能导致

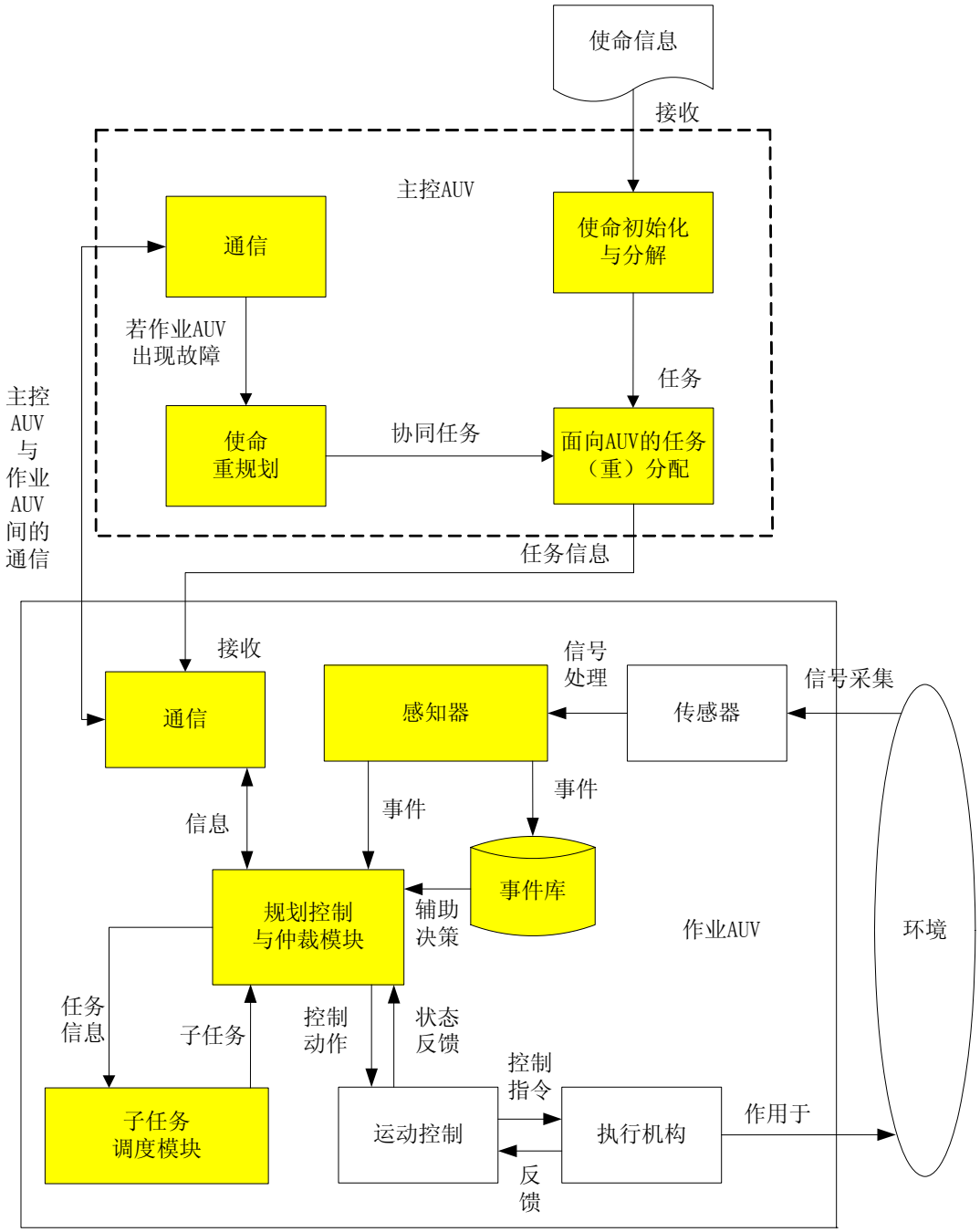


图 2.4　多 AUV 系统体系结构

使命不能正常执行。感知器兼具采集环境数据与监控作业 AUV 自身状态的功能。当子任务与事件并发时，感知器将事件发送给事件库和规划控制与仲裁模块；规划控制与仲裁模块根据事件库提供的辅助决策，结合子任务与事件的优先级顺序做出仲裁，生成动作执行序列。

规划控制与仲裁模块将控制信号发送给运动控制单元，运动控制单元控制执行机构，并负责向规划控制与仲裁模块反馈执行机构的状态；规划控制与仲裁模块将作业执行信息和作业 AUV 工作状态等信息封装成数据包通过进程间通信的方式发送给主控 AUV[24]。作业 AUV 根据动作执行序列调度系统资源，完成地形勘察任务。本文针对作业 AUV 研究的主要方面为图中的感知器、事件库、子任务调度模块、规划控制与仲裁模块和通信模块的实现。作业 AUV 的结构如图 2.4 下半部分所示。

2.4 功能模块设计

2.4.1 主控 AUV 功能

2.4.1.1 使命初始化与分解

使命开始前，首先通过主控 AUV 设置执行使命的作业 AUV 数量。主控 AUV 具有待勘察海域的地图信息，在地图上初始化勘察使命，其初始化内容为各个作业 AUV 的起点、回收点和勘察区域。完成使命初始化后，分解为发送给各个作业 AUV 的任务，制作成信息文本，待通信端口就绪后发送给各个作业 AUV。

2.4.1.2 通信

通信模块负责向各个作业 AUV 下达勘察任务，将完成初始化的使命信息文本发送给各个作业 AUV。使命开始执行后，各个作业 AUV 将定时向主控 AUV 发送使命执行信息，主控 AUV 将接收到的信息进行处理；若作业 AUV 出现故障，则向主控 AUV 报告故障情况，主控 AUV 根据情况对使命进行重规划。

2.4.1.3 使命的规划与重规划

使命规划体现在地形勘察使命的分工，在使命初始化阶段即完成了各个作业 AUV 任务的分配；使命的重规划实现了作业 AUV 间的协作，当某一作业 AUV 出现故障时，会向主控 AUV 报告故障情况，主控 AUV 会根据其使命的完成程度进行重规划，制定出协同任务，并采取一定仲裁机制，选择能够正常工作的作业 AUV 完成协同任务，确保整个使命能够顺利完成。

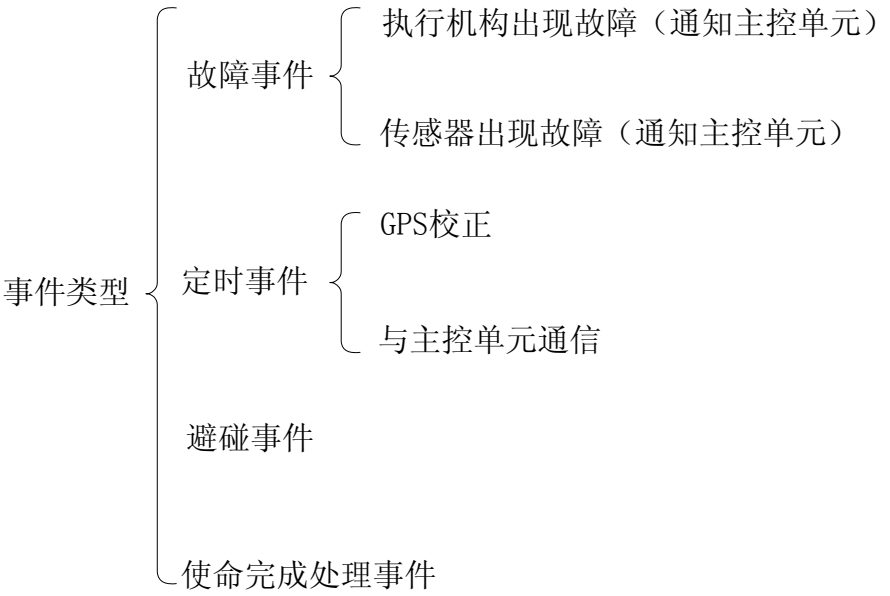
2.4.2 作业 AUV 功能

2.4.2.1 感知器

感知器用于感知作业 AUV 在任务执行过程中出现的各种事件。在 AUV 实体的设计中，感知器负责根据传感器采集到的 AUV 外部和内部环境信息数据，判断 AUV 作业是否遇到阻碍或发生异常，进而感知是否有事件发生。在本例中，事件可分为定时触发事件和不确定事件。定时事件由计时器定时触发，如定时通信事件，在事件发生前系统已做好处理准备且事件的类型已知；不确定事件理论上是由作业 AUV 内外部环境触发，如系统故障或避障等，在实际程序设计中是通过作业 AUV 程序界面上的按键进行触发，需要通过感知器捕捉，并对事件类型进行识别才能进行相应处理。

2.4.2.2 事件库

事件库中包含了各个子任务和已预设的事件及其优先级。在作业 AUV 单元将接收到的任务信息划分为子任务序列后，事件库将各子任务的优先级提供给仲裁器进行参考，用以生成正确的动作执行序列；当有事件发生时，事件库将给出当前事件的优先级，指导规划控制与仲裁模块将事件处理动作插入到动作执行序列的正确位置。根据仿真实验的需要，设计出的事件库中包含的事件如下：



各个子任务与事件的优先级顺序如表 2.1 所示：

表 2.1　子任务与事件优先级顺序

优先级	各子任务和事件名称
高	使命完成
↓	执行机构故障
↓	传感器故障
↓	避障事件
↓	GPS 校正
↓	归航子任务
↓	勘察子任务
↓	航渡子任务
低	协同子任务

2.4.2.3 子任务调度模块

当主控 AUV 将任务信息制作成文本文件并发送给作业 AUV 后，作业 AUV 的子任务调度模块使用数据流的方式打开任务文本，提取出初始化各个子任务的数据，包括 AUV 作业的起点、待勘察的矩形区域四个顶点的坐标和回收区域的坐标。将各个坐标值赋值给各个用于初始化子任务的变量。

坐标信息提取完成后，初始化对应的子任务，即向各个子任务线程发送各段路径的起点和终点坐标，其中航渡任务的起点位置可以直接确定，而航渡任务的终点应为勘察区域四个顶点坐标中与起点距离最近的一个；勘察任务的起点即为航渡任务的终点，而终点位置需要根据作业 AUV 侧扫声纳的范围进行计算，甚至可能出现勘察任务终点的位置不在勘察区域中的情况，此计算方法将在下文进行介绍；勘察区域的终点为归航任务的起点，任务文本中的回收点即为归航任务的终点。

2.4.2.4 规划控制与仲裁模块

规划控制与仲裁模块负责结合子任务信息和感知到的 AUV 事件，生成正确的动作执行序列，动作执行序列是作业 AUV 执行地形勘察使命的具体步骤。任务执行开始时，动作执行序列中仅包含航渡、勘察和归航三项子任务；在任务执行过程中会定地时或突发地插入事件处理过程，各个事件处理过程也有着不同的优先级，因此不同的事件处理过程和子任务之间存在着对当前执行权的竞争关系。当感知到事件产生时，感知器将事件类型发送给规划控制与仲裁模块和事件库，事件库提供新触发事件和当前正在执行的

子任务或事件处理过程的优先级来为仲裁提供依据，进而将事件处理过程插入到动作执行序列中的正确位置；优先级较低的子任务或事件处理过程能被阻塞，让出执行权限使得优先级较高的事件处理过程能够及时响应，保证作业 AUV 的安全运行与勘察任务的顺利完成。

2.4.2.5 通信

通信模块负责以进程间通信的方式，完成各个作业 AUV 与主控 AUV 之间的信息传输。任务执行开始时，主控 AUV 将生成的任务信息文本发送给作业 AUV；任务执行开始后，作业 AUV 单元将定时生成任务执行信息文本，发送给主控 AUV，主控 AUV 据此对作业 AUV 的运行情况进行监控。

2.5 本章小结

本章对经典多机器人系统体系结构进行了分析，设计出了由一个主控 AUV 和多个作业 AUV 组成的集中式多 AUV 群体协同体系结构。根据此体系结构对主控 AUV 和作业 AUV 的功能进行了划分，设计出了各个单元的功能模块以及模块间的信息流。为下面群体协同使命控制的研究提供了实验平台。

第 3 章 基于多进程和多线程机制的 AUV 群体协同仿真平台构建

3.1 引言

基于上一章设计的体系结构，本章介绍如何对地形勘察使命进行分解，以及分解后的具体描述方法。

在多 AUV 系统的实际设计中，多 AUV 系统中的各个单元都是相互独立的个体。因此在仿真程序设计中，也应使用相互独立的进程描述主控 AUV 和各个作业 AUV，以模拟实际中 AUV 个体之间物理上的独立性，且通过各个进程之间的通信模拟群体中 AUV 之间的信息交互，从而更接近实际情况。

另一方面，每个 AUV 个体的内部结构也应采用适合的方式来构建，进而设计子任务和事件处理过程的描述方法，最终实现地形勘察使命执行过程的仿真。从功能上来看，AUV 单元的各个功能模块可采用函数或线程的方式来描述，函数和线程都可以作为子任务与事件处理的描述方式。函数是一个静态实体，是功能模块的划分，是一组指令的有序集合，它本身没有任何运行的含义；线程是一个动态的实体，它有自己的生命周期，反映了一个程序在一定的数据集上运行的全部动态过程。线程还具有并发性和交互性，这也与函数的封闭性不同。线程都是由操作系统所体会的程序运行的基本单元，系统利用该基本单元实现系统对应用的并发性。由于多 AUV 使命控制程序中子任务的和事件处理的并发性和有序性，且均需持续运行一段时间的并具有较复杂的功能，故选择线程描述子任务和事件处理过程。

使用多线程方式构建 AUV 使命控制结构的理由之二是线程间方便的通信机制。对于函数来说，在被调用前就需要将所需的各个参数传递给函数，由于 AUV 勘察作业的动态特性，诸如 AUV 当前坐标等信息都是实时变化的，要通过函数的方式描述 AUV 作业过程，只能通过以一定时间间隔调用相应函数来实现，这样仿真程序便模拟不了 AUV 实际作业的连续过程。线程则不然，由于同一进程下的线程之间共享数据空间，所以一个线程的数据可以直接为其它线程所用，这不仅快捷，而且方便，线程运行所需的各项参数可以被修改，只需保证对参数的修改是原子操作即可保证程序的正常运行。

3.2 地形勘察群体协同使命分解

在实际操作中，作业 AUV 由主控 AUV 进行总体控制和调度。主控 AUV 根据作业需求，进行使命规划；使命在完成规划后形成任务下达给作业 AUV，作业 AUV 将任务又进一步划分为子任务[25]，每个子任务都是执行作业的具体动作。在作业 AUV 执行使

命的过程中，往往需要对各种事件进行处理。事件是未在作业 AUV 所获得的使命信息中列出，而在作业 AUV 执行使命的过程中出现的，需要作业 AUV 进行正确处理以保证使命能够正确完成。通常为了能使作业 AUV 对各种事件应付自如，设计者通常需要在实际执行使命前，对 AUV 可能遇到的各种事件进行预测，编写出尽可能完善的事件库。事件库中包含了事件的有关信息以及作业 AUV 需采取的相应对策。作业 AUV 对事件的处理能力越强，则说明 AUV 的智能程度越高，AUV 就需要涵盖信息更全面的事件库。

综上所述，作业 AUV 在执行使命前，需要将各个子任务进行排序，调度系统资源，形成动作执行序列，并根据自身事件库中的内容，对作业执行过程中出现的各种事件进行响应，产生执行作业的实际行为。如图 3.1 所示，使命完成初始化后在使命规划层完成规划，分解为任务发送给各个作业 AUV。作业 AUV 的使命控制层将任务分解为若干子任务，进行调度。子任务通过使命控制，转化为执行机构的具体行为[26]。

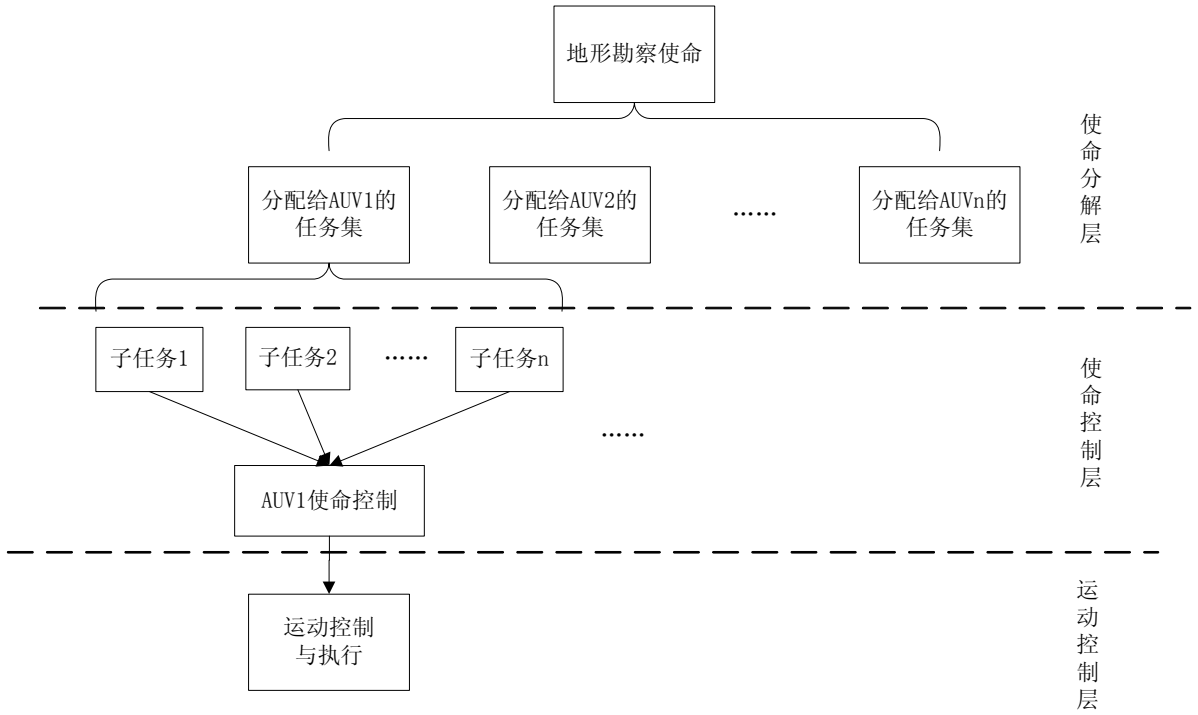


图 3.1　使命与任务间的关系

主控 AUV 根据全局路径规划制定出使命，使命被划分为任务，发送给各个作业 AUV，任务由作业 AUV 进一步被划分为子任务，作业 AUV 控制器根据输入的子任务序列和自身事件库，产生执行使命的具体行为。任务包括航渡子任务、归航子任务、作业子任务和协同子任务，事件仅列举出了避障事件、通信事件和 GPS 校正等几项。子任务的划分与事件的划分如表 3.1 所示。通过事件触发、定时和仲裁等机制来协调系统

资源的调度，产生 AUV 作业的实际行为[27,28]。

作业执行过程描述如下：作业 AUV 从起点开始，首先执行航渡子任务驶向作业区；到达作业区后执行勘察子任务，对目标区域进行遍历扫描；完成勘察子任务后，驶向回收区。使命执行过程中将对航线上出现的障碍物进行避让，定时与主控 AUV 通信，以及根据需要上浮，使用 GPS 校正 AUV 坐标。当全部作业完成后，使命完成事件被触发，使命结束。各子任务与事件的优先级顺序如表 3.1 所示。

表 3.1　任务与事件

子任务	航渡：驶向勘察区域 归航：驶向回收区域 勘察：遍历扫描作业区域 协同：完成其他作业 AUV 因故未完成的使命
事件	使命完成事件 避障事件 故障事件（包括执行机构故障和传感器故障） 与主控 AUV 定时通信 GPS 校正

3.3 主控 AUV 进程的多线程结构设计及其实现

3.3.1 使命规划线程

使命规划线程负责完成使命的初始化，记录通过程序界面设置的各个作业 AUV 的起点、回收点和勘察区域坐标，并对数据格式进行处理和封装，加上使命文本开始标志和结尾标志制作成使命文本文档，通过进程间通讯的方式发送给各个作业 AUV。

3.3.2 重规划线程

当检测到某一作业 AUV 出现故障不能完成自身使命时，主控 AUV 调用重规划线程，根据故障 AUV 的使命完成情况对其使命进行重规划，计算未完成勘察的勘察区域坐标，封装成数据包准备发送给执行协同任务的作业 AUV。

重规划线程的算法流程为：首先根据故障 AUV 的起点位置和勘察子任务完成的百分比计算剩余未完成勘察的区域坐标；接着计算哪个能够正常工作的 AUV 能够最快到达协同任务的勘察区域；最后制作重规划的使命信息文本，发送给该作业 AUV，命令其完成协同任务。

3.3.3 通信线程

负责使命完成初始化后向作业 AUV 发送使命信息，以及定时接收作业 AUV 发送来的文本数据，内容包括当前时间、当前执行子任务、当前坐标、当前子任务执行百分比等。经过分析后根据所获得数据在界面上显示并在地图上绘制出路径。主控 AUV 任务消息发布通信协议为：

（1）任务消息

主控 AUV 向作业 AUV 发送任务文本的通信协议格式如下：

- 起始符\n
- 作业 AUV 起点坐标\n
- 作业 AUV 终点坐标\n
- 勘察区域四个顶点的坐标\n
- 结束符

其中，起始符为#MISSION，结束符为$MISSION。所有坐标的表示方式为先经度后纬度，各个坐标值间用逗号隔开。每个为了解析文本时更加简便，因而以整数方式表示，后六位为小数部分。示例如下：

#MISSION
118612853,38991435
119326019,38357602
118722571,38751606,119095611,38751606,118722571,38546039,119095611,38546039
$MISSION

（2）重规划消息

主控 AUV 向作业 AUV 发送任务文本的通信协议格式如下：

- 起始符\n
- 重规划出的勘察区域四个顶点的坐标\n
- 结束符\n

其中，起始符为#REPLAN，结束符为$REPLAN。坐标表示方法及格式与任务信息相同。示例如下：

#REPLAN
122545775,34709756,123095070,34709756,122545775,34615779,123095070,34615779
$REPLAN

3.3.4 界面显示

界面显示功能在 AUV 实体设计中并不需要，由于本课题研究内容为使命控制方法的仿真，故需设计主控 AUV 程序界面。在主控 AUV 程序界面上可完成地形勘察使命的初始化；使命执行过程中，根据接收到的作业 AUV 任务执行信息在地图上绘制出各个作业 AUV 的路径。

3.4 作业 AUV 进程的多线程结构设计及其实现

3.4.1 功能模块线程

3.4.1.1 计时器线程

计时器线程用于获得当前系统时间，在其他线程需要获得当前时间时向该线程发送当前时间，精确至毫秒。且负责按照设定的时间触发 GPS 校正事件和与主控机通信事件，并定时在作业 AUV 单元程序界面上显示当前的使命执行情况。

3.4.1.2 规划控制与仲裁线程

规划控制与仲裁线程负责在子任务调度模块完成任务分解后，将各个子任务插入到动作执行序列中，并依动作执行序列的顺序依次创建各个子任务线程。在这里，子任务调度模块由于只需在接收到任务信息后执行任务分解工作，不需持续运行，因此用函数的方式描述。规划控制与仲裁线程还负责在任务执行过程中接收感知器感知到的事件，根据优先级将事件处理线程插入到动作执行序列的正确位置。

3.4.1.3 界面线程

界面显示功能在 AUV 实体设计中并不需要，由于本课题研究内容为使命控制方法的仿真，故需设计作业 AUV 程序界面。作业 AUV 的界面用于设置 AUV 模型的基本参数；接收到任务信息后，完成规划后的任务信息会显示在界面上；在任务执行过程中，界面上显示作业 AUV 当前正在执行的子任务或事件处理过程，以及当前坐标，用于与主控 AUV 显示的内容进行对照，以验证通信是否正常。

另一方面，界面上有用于触发 AUV 事件的按钮，因此在本例中界面线程也兼具感知器的功能。

3.4.2 子任务线程

3.4.2.1 航渡线程

航渡线程用于控制 AUV 按照设定的路线，从起点航行至勘察区域。线程启动后，通过

子任务调度模块获得航渡子任务起点和终点的经纬度坐标，接着通过计时器线程获得子任务开始的准确时间。由于 AUV 运动控制不在本课题的研究范围之内，仿真程序中没有 AUV 运动模型，因此实现 AUV 航行的方法为：设 AUV 在无扰动无障碍的环境中按照设定速度匀速航行，这样即可通过设定的速度和耗时计算出 AUV 航行的距离。线程主体是一个循环，以一定时间间隔计算作业 AUV 的当前位置，并判断当前位置是否超过了设定的终点；若超过则循环终止，航渡子任务完成。显而易见，计算的时间间隔越小则检测精度越高。由于一般小型 AUV 的速度普遍在 3m/s 以下，而两个路径点间的距离往往是几百米甚至几公里，所以将计算 AUV 当前时刻位置的时间间隔设置为 1000ms 就完全可以满足需要。每到计算时间，就通过计时器线程获得当前的时间（精确到 ms），

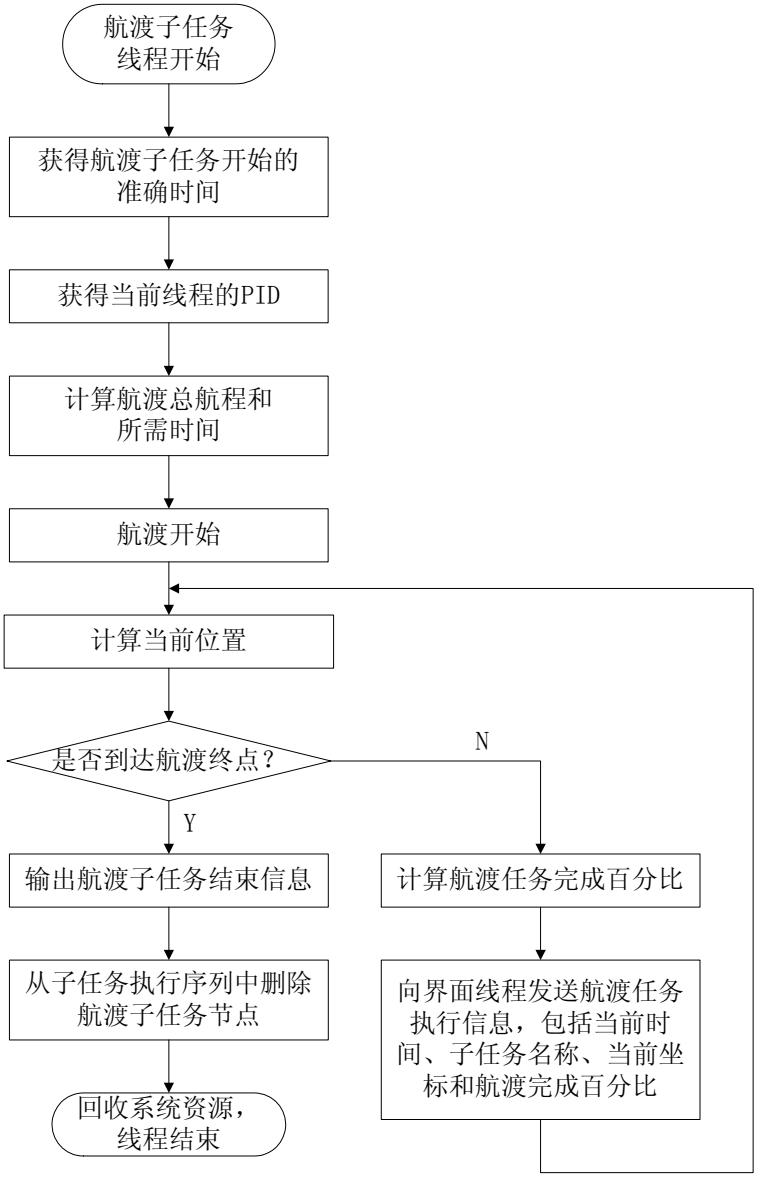


图 3.2　航渡子任务流程图

并输出当前的经纬度坐标。若作业 AUV 当前位置超过终点，则主循环终止，输出航渡子任务完成信息。为减少运行时间，所有线程中作业 AUV 实际速度为设定速度的 1000 倍，侧扫声呐范围为设定范围的 100 倍。航渡子任务线程流程图如图 3.2 所示。

3.4.2.2 勘察子任务线程

勘察子任务线程主要功能为以 S 行路线遍历个待勘察区域，所以需要根据勘察区域的大小和 AUV 的侧扫声纳扫描范围计算出 AUV 需要在哪些位置转向。这些转向的位置构成了一段段路径，AUV 沿着这一系列路径航行即可实现勘察区域的遍历。勘察路径的示意图如下所示：

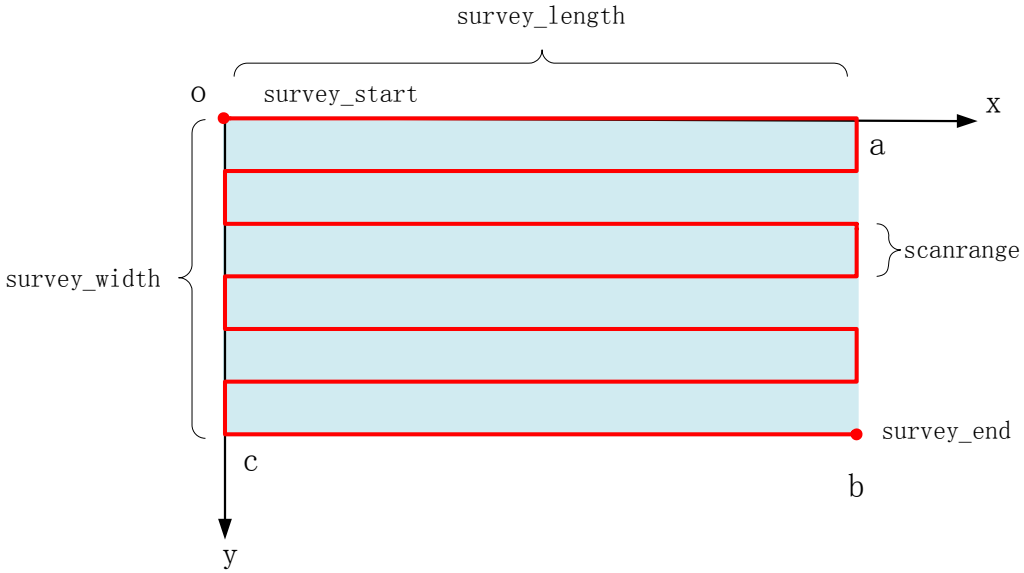


图 3.3　勘察路径示意图

图中灰色区域 oabc 为矩形待勘察区域，粗实线为 AUV 航行轨迹，以起点（survey_start）为原点建立坐标系，oa 为勘察区域长度（survey_length），oc 为勘察区域宽度（survey_width）。AUV 以 S 形遍历整个区域。为了获得较高的遍历效率，减少转向次数，则应沿矩形较长一边（即 x 轴）航行，到达区域边界后转向 90°，沿 y 轴方向航行一段距离（图中 scanrange）后转向 90°，沿 x 轴负方向航行，该段距离（scanrange）即为 AUV 侧扫声纳的探测范围。如此反复即可完成整个区域的遍历。

根据上述思想，可得出以下公式：

$$\text{float}\quad k = (\text{survey\_width} + \text{scanrange}/2\ ) / \text{scanrange};$$

$$\text{yaw\_times} = (\text{int})k;$$

其中 yaw_times 为转弯次数。由于侧扫声纳可以扫描 AUV 两侧的区域，通过计算可以得知，通常 k 值都是具有小数部分的，即使 k 值有小数部分，以 k 的整数部分值作

为转弯次数进行遍历依然可以遍历整个勘察区域，故转弯次数应为 k 值取整数。在某些情况下可能出现 yaw_times * scanrange > survey_width 的情况，故应根据转弯次数计算出勘察子任务的实际终点（即归航子任务的起始点）。

则生成的路径点个数为：

路径点个数 = (转弯次数+ 1) * 2

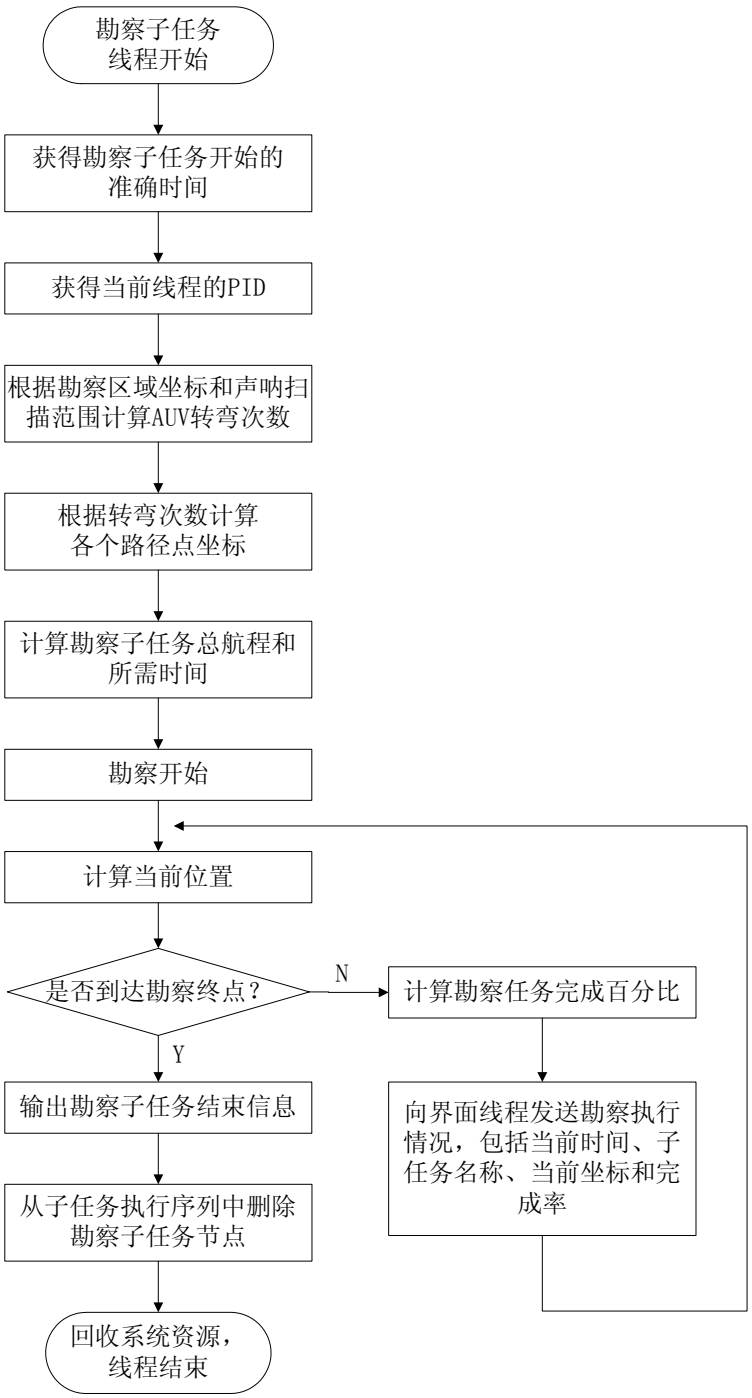


图 3.4　勘察子任务流程图

Linux 环境下的 C 语言编译器的一个特点是支持使用变量初始化数组的大小，从而可以实现以计算出的路径点个数初始化一个二维数组用于存储各个路径点的坐标，之后线程主体以循环的形式判断 AUV 在各段路径的位置，若驶过一个路径点，则向下一个路径点航行，直至完成勘察区域遍历。再利用之前航渡线程部分介绍的方法实现 AUV 运动，航行过程中同样以 1000ms 间隔进行位置计算。若 AUV 当前位置超过勘察终点，则主循环终止，输出航渡子任务完成信息。勘察子任务线程流程图如图 3.4 所示。

3.4.2.3 归航子任务线程

归航子任务线程与航渡子任务线程内容基本一致，所不同的仅仅是起点位置需通过计算勘察子任务结束的位置得出。

3.4.2.4 协同子任务线程

当某个作业 AUV 完成自己的任务，到达回收点后，会向主控 AUV 报告自身状态，并等待重规划信息。在此之前若有其他作业 AUV 出现故障不能完成任务，主控 AUV 会收到消息并规划出协同子任务，等待某个正常工作的作业 AUV 完成任务后，向其发送协同任务信息。作业 AUV 则创建协同线程。协同线程完成协同子任务信息的分析，

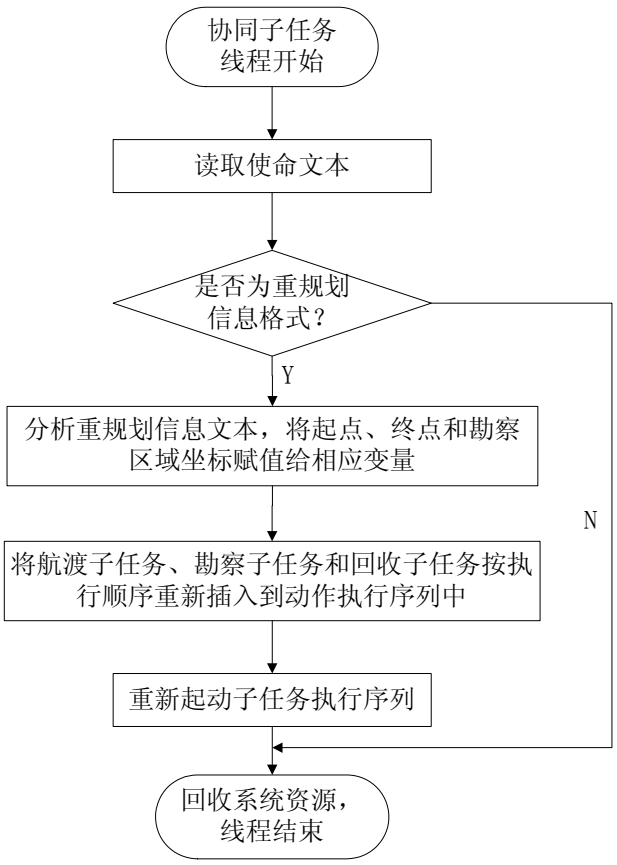


图 3.5　协同子任务线程流程图

并将航渡、勘察和归航线程顺序插入到执行序列中。这里航渡子任务的起点为作业 AUV 的回收点，回收点依然为原回收点。作业 AUV 将继续执行其他作业 AUV 因故障未能完成的任务。协同子任务线程流程图如图 3.5 所示。

3.4.3 事件线程

3.4.3.1 与主控机通信事件线程

由计时器线程每隔一段时间创建，通过进程间通信的方式，向主控机发送当前时间、作业 AUV 当前正在执行的子任务、坐标以及当前子任务完成百分比。作业 AUV 向主控 AUV 发送任务执行信息文本的通信协议格式如下：

- 时间，格式为：hh:mm:ss:msmsms\t
- 当前子任务\t
- 当前坐标\t
- 当前子任务完成百分比

消息中时间精确至毫秒；当前正在执行的子任务类型有 Loiter、Survey 和 Recover 三种。示例如下：

11:09:30:022 Recover (122.039343, 38.982931) 100.00%

当任务结束后，作业 AUV 会对自身的任务执行情况进行总结，并将自己的总航程和总耗时发送给主控 AUV。示例如下：

Mission Complete!! Total distance: 423954.93m. Total cost time: 169.002s.

3.4.3.2 GPS 校正事件线程

用于测试是否能正确向队列中插入事件并阻塞前一个线程，由定时器线程每隔一段时间创建。使 GPS 校正线程阻塞正在执行的子任务线程，并在屏幕上输出文字以示 GPS 校正正在进行，隔一定秒数后线程结束，原先被阻塞的线程将继续运行。

3.4.3.3 使命结束报告线程

当作业 AUV 完成任务，且不需执行协同任务的情况下，就会创建使命结束报告线程。当作业 AUV 成功完成任务时，使命结束报告线程会计算该 AUV 航行的总路程和完成任务的总耗时，并发送给主控 AUV。所有子任务线程都会记录 AUV 行驶的航程，使命结束报告线程将各个子任务的航程求和计算出整个任务的总航程。在作业 AUV 程序开始执行任务时，会通过计时器线程获得当前的系统时间并记录，使命结束报告线程以同样方式获得当前系统时间，根据使命开始执行的时间计算出完成任务的总耗时。若作业 AUV 因传感器故障未能正常完成任务，该线程创建后会直接向主控 AUV 报告任

务失败及失败原因。最后回收系统资源，完成作业 AUV 单元进程的正常退出。若作业 AUV 因机械故障未能完成任务，则此线程不会被创建，从而使作业 AUV 进程不能正常退出以表示其出现故障。使命结束报告线程流程图如图 3.6 所示。

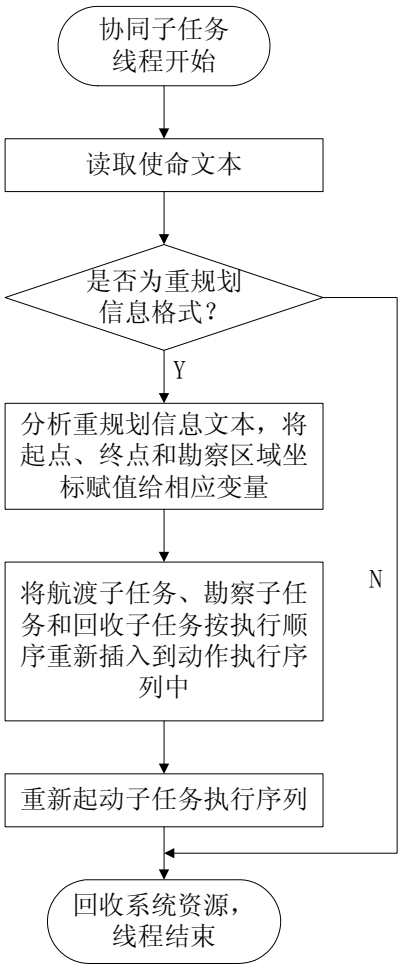


图 3.6　使命结束报告线程流程图

3.4.3.4 执行机构故障事件处理线程

该线程通过作业 AUV 界面上的按钮触发，用于模拟 AUV 因推进器故障不能继续执行任务且无法行动的情况。当此线程被触发时，向主控 AUV 报告该 AUV 执行机构故障，以及出现故障时的坐标，当前正在执行的子任务线程和事件线程都被强行退出，任务将不能继续执行，作业 AUV 进程也不能正常退出，以此表示其故障状态。

3.4.3.5 传感器故障事件处理线程

该线程通过作业 AUV 界面上的按钮触发，用于模拟 AUV 声呐出现故障不能继续执行勘察子任务的情况。当此线程被触发时，向主控 AUV 报告该作业 AUV 声呐系统

出现故障，以及出现故障时的坐标。由于勘察任务不能继续执行，本线程会将动作执行序列中的所有子任务删除，作业 AUV 将直接驶向回收点等待回收，并输出任务执行失败信息。

3.4.3.6 避障事件处理线程

该线程通过作业 AUV 界面上的按钮触发，模拟 AUV 的避障动作，用于测试使命控制方法能否阻塞当前正在执行的子任务或优先级较低的事件处理线程，转而执行避障线程。该线程用于测试高先级的事件线程能否阻塞低优先级的事件线程。

3.5 本章小结

本章根据上一章确定的总体集中，个体分层式的体系结构，设计了基于多线程的主控 AUV 和作业 AUV 仿真构建方法，并且将作业的执行过程，即各个子任务和事件处理过程也使用线程加以描述，为下面多 AUV 群体协同使命控制算法的设计打下了基础。

第 4 章 基于 Petri 网的多 AUV 群体协同使命控制研究

4.1 引言

在仿真程序实际设计中，使用不同的进程描述主控 AUV 和各个作业 AUV，多 AUV 系统中的各个单元通过进程间通信的方式进行交互。使命的初始化和执行过程为：起动仿真程序，首先通过主控 AUV 对勘察使命进行设置，内容包括：作业 AUV 数量、各作业 AUV 初始位置、勘察区域及回收点等信息；接着主控 AUV 通过进程间向各作业 AUV 发送任务信息，AUV 开始执行任务，在主控 AUV 界面上显示出当前时间、各作业 AUV 当前坐标等信息，并在地图上绘制出各作业 AUV 运行的路径；加入故障模拟功能，可在主控程序中使某一作业 AUV 出现故障从而不能完成任务，此时主控程序对使命进行重规划，派遣其他状态正常的作业 AUV 前去执行未完成的勘察任务。

作业 AUV 任务执行过程描述如下：作业 AUV 从起点开始，首先执行航渡子任务，驶向作业区；到达作业区后执行勘察子任务，进行地形勘察；完成勘察子任务后，执行归航子任务，驶向回收区。整个作业过程中将定时与主控 AUV 通信，报告作业 AUV 当前的任务执行情况，并定时上浮进行 GPS 位置校正。作业执行过程中，对于出现的各类事件也应具有相应的处理方法：当作业 AUV 执行机构或传感器出现故障，将向主控 AUV 报告故障信息；主控 AUV 能获知某作业 AUV 的故障状态，从而使主控 AUV 进行使命重规划，这种情况下将触发协助事件：主控 AUV 根据故障的作业 AUV 的任务执行情况计算出未完成勘查的区域，并选择出执行协同任务的作业 AUV，若有多个作业 AUV 有能力进行协同，则通过一定的仲裁机制进行选择，设计价值函数对各个 AUV 的当前使命完成情况以及到达协同任务区域的距离等进行评估，选择出能够最快到达协同任务勘察区域的 AUV，完成协同任务的规划。将协同任务信息发送给目标 AUV 后，该 AUV 会在完成自身的任务后执行协同任务，并最终回到自己的回收点，此时全部使命执行完毕[29,30,31]。

4.2 使命控制系统的 Petri 网建模

4.2.1 Petri 网原理与方法

多 AUV 使命控制系统主要负责上层的系统管理、任务调度，是一个复杂的离散事件系统，具有并行、异步和冲突竞争等特点。离散事件系统(DES)是一类由事件驱动系统进程的动态系统，事件发生的时间具有不确定性。因此需要使用一种有效的建模方法来对每个作业 AUV 的子任务调度与整个系统的使命控制过程进行描述[32]。

离散事件系统概念是由哈佛大学何毓琦教授于 20 世纪 80 年代提出的[33]。研究离散

事件系统的方法很多，Wonham 和 Ramadge 提出的离散事件监控理论(RW 理论)是应用得比较广泛的一种，这种理论最早是以自动机和形式语言为模型提出的。有限状态自动机（FSM "Finite State Machine" 或者 FSA "Finite State Automaton"）是为研究有限内存的计算过程和某些语言类而抽象出的一种计算模，它拥有有限数量的状态，每个状态可以迁移到零个或多个状态，输入字串决定执行哪个状态的迁移。有限状态自动机是自动机理论的研究对象，可以表示为一个有向图。自动机模型在计算复杂性方面有很大的缺陷，为了改进 RW 理论，目前普遍采用 Petri 网作为模型研究离散事件系统的动态特性[34]。

Petri 网于 20 世纪 60 年代由卡尔·A·佩特里发明，适合于描述异步的、并发的计算机系统模型。Petri 网是对离散并行系统的数学表示，既有严格的数学表述方式，也有直观的图形表达方式，既有丰富的系统描述手段和系统行为分析技术，又为计算机科学提供坚实的概念基础。Petri 网可将相关性和独立性用网络形式来表示，适合于描述离散事件系统模型，可较好地描述并发系统的结构，并能对系统的状态性质(有界性，安全性，公平性，可达性，可逆性，活性，不变量等)进行分析。主要应用于系统建模、仿真，广泛应用于自动控制系统和计算机科学。Petri 网作为离散事件系统建模技术具有以下优点：Petri 网允许以描述系统事件的方式建模，事件间的过渡能够以规则的集合来描述；并且它们可以并发的描述事件系统。Petri 网是分散的和模块化的复合系统的便利工具。使用 Petri 网，单独的系统部件可以简单的定义并且它们之间相互作用的等级也可清晰地显示，这样简化了增加的建模任务。Petri 网既是图形工具又是数学工具，具有图形直观、能描述冲突和真并发，且能以状态分布表示等优点，被认为是迄今研究离散事件系统的最有力的工具[35,36,37]。

Petri 网利用网络图形描述对象之间的输入输出关系，能很好地反映系统的静、动态特性。其动态特性通过托肯(token)的移动和传播进行控制，一般地，规则的前提和结论子句与 Petri 网的库所对应，一条诊断规则与一个转移对应；一旦规则的前提条件得到满足(输入库所有一个托肯)，当规则被激活(转移点火)时，可确认该规则的结论成立(托肯置于输出库所)[38]。

4.2.2 主控 AUV 使命控制 Petri 网建模

主控 AUV 的使命控制过程 Petri 网如图 4.1 所示，其中库所表示主控 AUV 的状态，变迁表示状态的转换。各个库所和变迁的含义如表 4.1 和 4.2 所示。

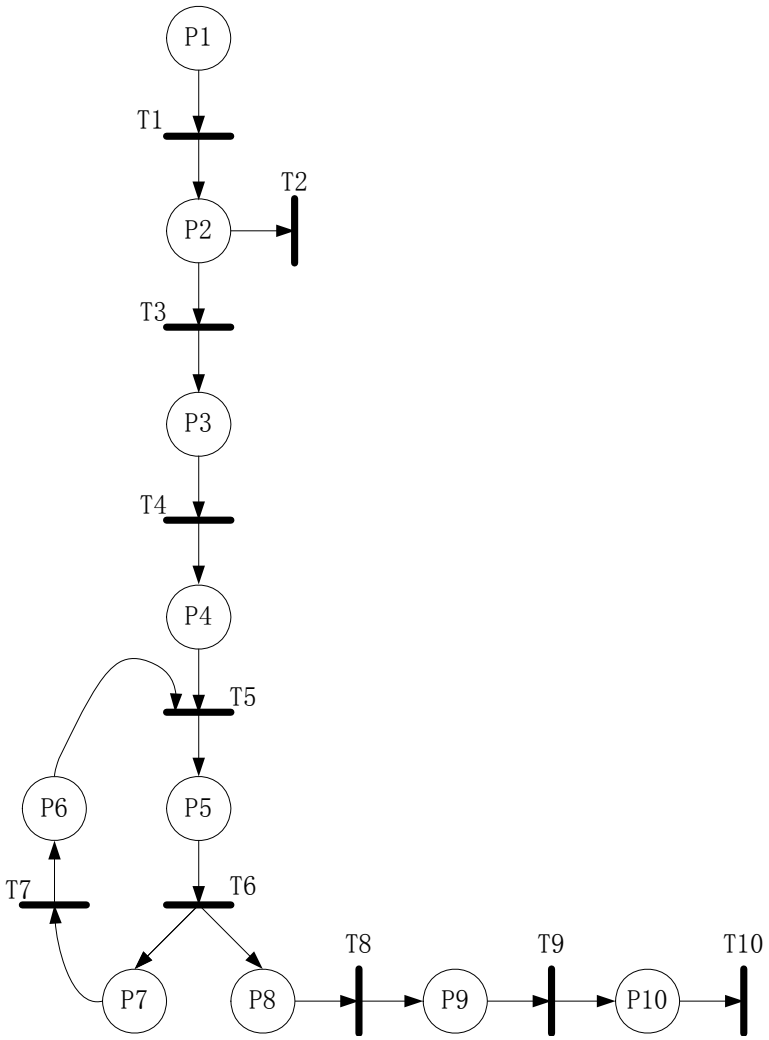


图 4.1　主控 AUV 的任务执行过程 Petri 网建模

表 4.1　主控 AUV 的任务执行过程 Petri 网库所定义

库所	含义	库所	含义
P1	主控 AUV 启动	P6	作业 AUV 任务执行信息显示完毕
P2	主控 AUV 就绪	P7	未收到作业 AUV 执行机构故障信息
P3	任务信息就绪	P8	作业 AUV 故障信息
P4	任务信息发送完毕	P9	重规划线程就绪
P5	作业 AUV 任务执行信息	P10	协同任务信息就绪

表 4.2　主控 AUV 的任务执行过程 Petri 网变迁定义

变迁	含义	变迁	含义
T1	多 AUV 系统初始化	T6	信息处理
T2	起动作业 AUV	T7	显示各个作业AUV的任务执行信息并在地图上绘制出路径
T3	使命初始化	T8	起动重规划线程
T4	向作业 AUV 发送任务信息	T9	根据故障 AUV 的勘察完成率进行重规划
T5	定时接收各个作业 AUV 的任务执行信息	T10	向执行协同任务的作业AUV发送协同任务信息

主控 AUV 程序启动后进入状态 P1，首先设置群体中作业 AUV 的数量，据此初始化系统以及主控 AUV 程序界面，进入主控 AUV 程序界面后到达状态 P2。主控 AUV 就绪后就可以执行变迁 T2 通过主控 AUV 启动各个作业 AUV。接着执行变迁 T3 进行使命初始化，初始化各个作业 AUV 的起点、勘查区域和回收点。完成使命初始化执行变迁 T4 将任务信息发送给各个作业 AUV。多 AUV 系统就绪后使命开始执行，此时主控 AUV 的主要任务是监控各个作业 AUV 的状态并予以显示，即图 4.1 中 P5、P6、P7 这一循环；作业 AUV 每隔一定时间会与主控 AUV 通信报告自身的任务执行情况，包括当前子任务、当前坐标和子任务完成百分比，主控 AUV 将这些信息进行处理和显示，在地图上绘制出各个作业 AUV 的路径。当主控 AUV 接收到 AUV 的故障信息时，将对使命进行重规划，根据故障 AUV 的任务完成情况规划出协同任务，并从状态正常的作业 AUV 中选择出适合执行协同任务者，向其发送协同任务信息，命令其在自身任务执行完毕后执行协同任务。

4.2.3 作业 AUV 使命控制 Petri 网建模

作业 AUV 的使命控制过程 Petri 网如图 4.2 所示，其中库所表示作业 AUV 的状态，变迁表示状态的转换。各个库所和变迁的含义如表 4.3 和 4.4 所示。

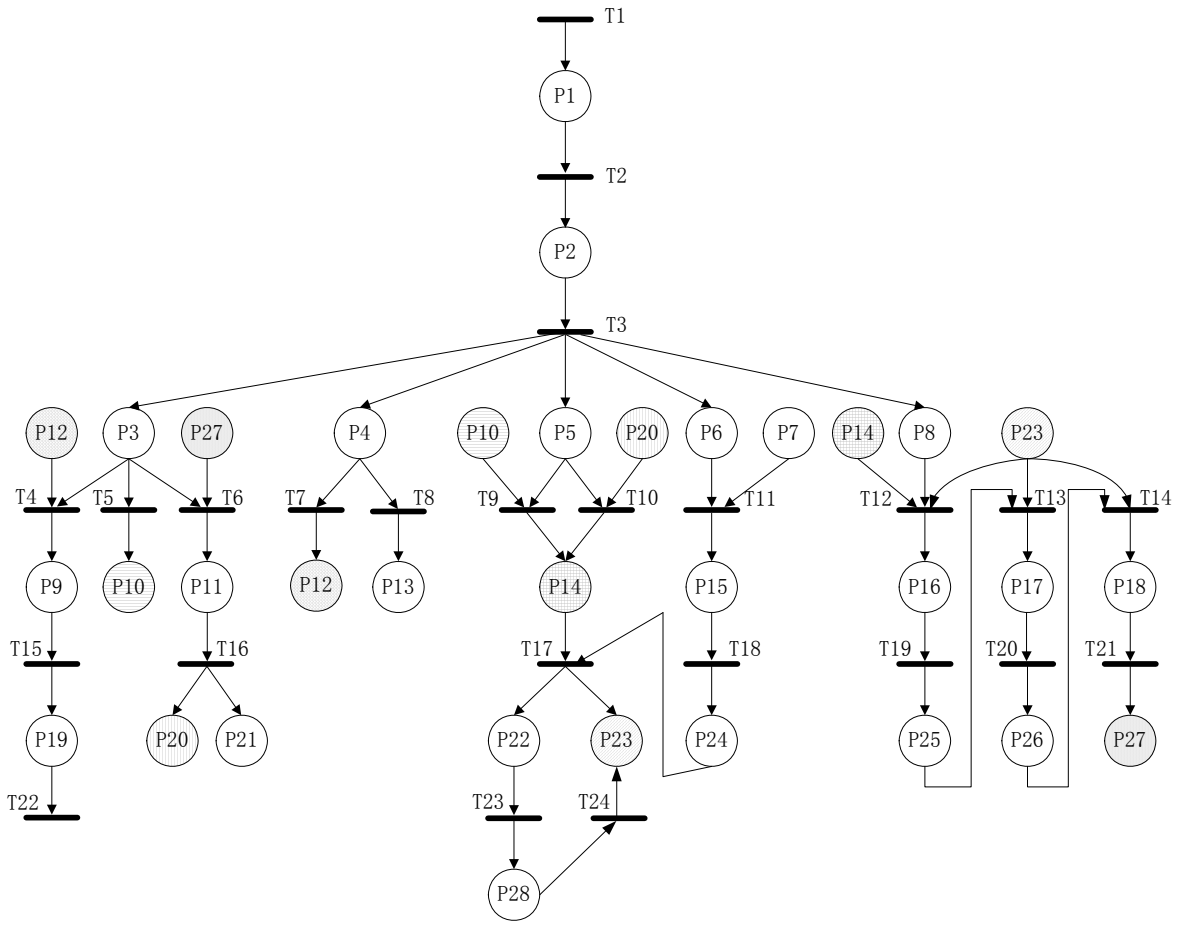


图 4.2　作业 AUV 的任务执行过程 Petri 网建模

表 4.3　作业 AUV 的任务执行过程 Petri 网库所定义

库所	含义	库所	含义
P1	作业 AUV 启动	P15	获得事件类型
P2	作业 AUV 就绪	P16	执行航渡子任务
P3	通信线程就绪	P17	执行勘察子任务
P4	计时器线程就绪	P18	执行规划子任务
P5	规划控制与仲裁线程就绪	P19	任务执行信息
P6	感知器线程就绪	P20	协同任务信息
P7	产生事件	P21	结束，向主控 AUV 报告使命耗时和总路程长度

P8	子任务调度线程就绪	P22	事件响应线程获得执行权
P9	定时通信就绪	P23	子任务线程获得执行权
P10	接收到的任务信息	P24	事件处理线程优先级
P11	任务结束信息	P25	航渡子任务结束
P12	定时通信信号	P26	勘察子任务结束
P13	触发 GPS 校正事件	P27	任务完成
P14	子任务队列	P28	事件处理结束

表 4.4　作业 AUV 的任务执行过程 Petri 网变迁定义

变迁	含义	变迁	含义
T1	主控 AUV 起动作业 AUV	T13	启动勘察子任务线程
T2	系统初始化	T14	启动归航线程
T3	各个线程启动	T15	获得 AUV 任务执行情况
T4	定时通信启动	T16	等待重规划信息
T5	任务执行前与主控 AUV 通信	T17	仲裁
T6	任务完成后与主控 AUV 通信	T18	查询事件库
T7	到达 GPS 校正时间	T19	到达航渡终点
T8	到达定时通信时间	T20	到达勘察区终点
T9	任务规划	T21	到达回收点
T10	协同任务规划	T22	发送给主控 AUV
T11	事件识别	T23	启动事件处理线程
T12	启动航渡线程	T24	将执行权限还给子任务

作业 AUV 通过主控 AUV 的启动动作（变迁 T1）完成启动，进入状态 P1，经过变迁 T2 完成初始化，对 AUV 的速度和传感器参数进行设置，AUV 进入就绪状态 P2，经过变迁 T3 启动 AUV 的各个线程。P3 表示通信线程就绪，通信线程有三种通信状态，

通过不同的状态触发，由 3 个变迁表示：T4 表示触发由库所 P12 表示的定时通信时间到状态触发的定时通信事件，定时通信会获取当前的任务执行状况，发送给主控 AUV；T5 表示任务开始执行前第一次与主控 AUV 进行通信，获得任务信息；T6 表示使命结束时，由库所 P27——任务结束状态触发的与主控 AUV 通信，此次通信要报告主控 AUV 任务完成，并执行变迁 T16 准备接收协同任务信息；若未接到协同任务，则进入库所 P20 向主控 AUV 报告执行任务的总耗时和总航程，作业 AUV 进程退出；若接收到协同任务信息进入库所 P21，P21 将触发协同任务的执行。P4 表示计时器线程，该线程记录系统时间，并按照设定的时间间隔触发 GPS 校正事件（T7）和定时通信事件（T8）。

P5 表示规划控制与仲裁线程就绪，当接收到规划信息（P10）时，将触发变迁 T9 进行任务规划；当接收规划信息（P20）时，通过变迁 T10 进行协同任务规划；上述两种情况都将制定出子任务序列，进入库所 P14，结合当前出现事件的优先级进行仲裁，若事件处理优先级高，则事件处理线程获得执行权（P22）并予以执行（T23）；若无事件产生或事件优先级低于当前正在执行的子任务的优先级，则对子任务赋予执行的权限（P23）。事件处理线程结束后（P28），将把执行权限还给子任务（T24）。

P6 表示感知器线程就绪，当产生事件时（P7），首先进行事件类型识别（T11），获得事件的类型（P15），接着查询事件的优先级（T18），将查询到的事件优先级提供给规划控制与仲裁线程进行仲裁，以决定当前应执行哪个线程。

P8 表示子任务调度线程就绪。结合规划控制与仲裁线程提供的子任务执行序列（P14），以及仲裁后获得的子任务执行权限（P23），依次执行航渡、勘察和规划子任务，执行完毕后产生任务结束状态 P27，任务结束状态将触发通信线程的与主控 AUV 通信动作。

4.3 AUV 使命控制的多线程调度

4.3.1 线程的创建与结束方式

在一个程序中的多个执行路线就叫做线程（thread）。更准确的定义是：线程是一个进程内部的一个控制序列。线程又称轻量级进程，是比进程更小的可以独立运行的基本单位。线程的和进程的区别主要在于线程不用有资源，即一个进程的所有线程共享这个进程的资源。

在 Linux 环境下的 C 语言中，线程的创建使用如下函数[39]：

```
int  pthread_create (pthread_t  *thread,  pthread_attr_t  *attr,
                     void *(*start_routine)(void *),  (void *)arg);
```

第一个参数为指向线程标识符的指针，第二个参数用来设置线程属性，第三个参数是线程要启动执行的函数，最后一个参数是传递给函数的参数[40]。

函数 pthread_join 用来等待一个线程的结束。函数原型为：

int pthread_join (pthread_t th, void **thread_return);

第一个参数为被等待的线程标识符，第二个参数为一个用户定义的指针，它可以用来存储被等待线程的返回值[41]。这个函数是一个线程阻塞的函数，调用它的函数将一直等待到被等待的线程结束为止，当函数返回时，被等待线程的资源被收回[42]。一个线程的结束有两种途径，一种是函数结束了，调用它的线程也就结束了；另一种方式是通过函数 pthread_exit 来实现[43,44]。它的函数原型为：

void pthread_exit (void *retval);

唯一的参数是函数的返回代码，只要 pthread_join 中的第二个参数 thread_return 不是 NULL，这个值将被传递给 thread_return。

要实现在线程执行过程中插入新的线程，就必须要在原线程结束前，即创建线程的函数获得返回值之前，也就是 pthread_join()调用前创建新的线程。然而事件线程的特点就是创建时间的不确定，如何保证在任务线程返回之前创建是一个难题。

Linux 环境下的多线程编程中，线程属性结构如下：

```
typedef   struct
{
    int                 detachstate;      //线程的分离状态
    int                 schedpolicy;      //线程调度策略
    struct sched_param  schedparam;       //线程的调度参数
    int                 inheritsched;     //线程的继承性
    int                 scope;            //线程的作用域
    size_t              guardsize;        //线程栈末尾的警戒缓冲区大小
    int                 stackaddr_set;    //指定创建的线程将要使用的堆栈基址
    void *              stackaddr;        //线程栈的位置
    size_t              stacksize;        //线程栈的大小
}pthread_attr_t;
```

通过对线程属性结构的研究可以发现，线程的分离状态决定一个线程以什么样的方式来终止自己[45,46]。采用线程的默认属性，即为非分离状态时，原有的线程等待创建的线程结束。只有当 pthread_join()函数返回时，创建的线程才算终止，才能释放自己占用的系统资源。而分离线程没有被其他的线程所等待，运行结束了，线程也就终止了，马上释放系统资源。设置线程分离状态的函数为：

pthread_attr_setdetachstate (pthread_attr_t *attr, int detachstate)

其中第二个参数可选为 PTHREAD_CREATE_DETACHED（分离线程）和

PTHREAD_CREATE_JOINABLE（非分离线程）。线程结束不需函数等待的情况下，线程的插入就容易了很多。本程序使用的是基于优先级的抢占式调度法[47]。这种调度方法为每个任务指定不同的优先级。没有阻塞或结束的最高优先级任务将一直运行下去；当更高优先级的任务由就绪态进入运行时，程序立即保存当前任务的上下文，切换到更高优先级的任务[48]。据此，通过软件方法可实现事件的实时响应。具体程序段如下：

```
void Dispatcher(int priority)
{
    void  *speed  =  &e_auv_speed;
    int  res;
    pthread_t  pthread;
    pthread_attr_t  attr;
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr,  PTHREAD_CREATE_DETACHED);

    switch(priority)
    {
        case  LOITER:
        {
            res  =  pthread_create(&thread,  &attr,  Loiter,  speed);
            if(res  !=  0)
            {
                perror("Loiter  Thread  creation  failed!\n");
                exit(EXIT_FAILURE);
            }
            pthread_attr_destroy(&attr);
            break;
        }
    ……
    }
}
```

函数 Dispatcher 的功能是调用执行序列中节点内存储的优先级所对应的线程，pthread_t 型变量用于声明线程 ID，pthread_attr_t 型变量用于初始化记录线程属性的结构体，pthread_attr_init()函数用于初始化一个线程对象的属性。在线程创建前，首先要

对线程的属性 pthread_attr_t attr 进行初始化，在这里是将线程设置为分离线程；接着用设置好的线程属性参数 attr 传递给函数 pthread_create()创建线程，其中 Loiter 为指向线程函数的指针，speed 是 AUV 的航速，为传递给线程的参数。线程创建完成后，即调用 pthread_attr_destroy()函数对线程属性去初始化，避免对创建新线程造成影响。

4.3.2 基于链表的子任务线程队列的创建

在许多类型的程序的设计中，数据结构的选择是一个基本的设计考虑因素。许多系统的构造经验表明，系统实现的困难程度和系统构造的质量都严重的依赖于是否选择了最优的数据结构。选择了数据结构，算法也随之确定，系统构造的关键因素是数据而不是算法。

计算机解决一个具体问题时，大致需要经过以下几个步骤[49]：首先要从具体问题中抽象出一个适当的数学模型，然后设计一个解此数学模型的算法（Algorithm），最后编写出程序、进行测试、调整直至得到最终解。寻求数学模型的实质是分析问题，从中提取操作的对象，并找出这些操作对象之间含有的关系，然后用数学的语言加以描述。计算机算法与数据的结构密切相关，算法无不依附于具体的数据结构，数据结构直接关系到算法的选择和效率[50]。运算是由计算机来完成，这就要设计相应的插入、删除和修改的算法。也就是说，数据结构还需要给出每种结构类型所定义的各种运算的算法。

数组(Array)：在程序设计中，为了处理方便， 把具有相同类型的若干变量按有序的形式组织起来，这些按序排列的同类数据元素的集合称为数组。在C语言中，数组属于构造数据类型。一个数组可以分解为多个数组元素，这些数组元素可以是基本数据类型或是构造类型。因此按数组元素的类型不同，数组又可分为数值数组、字符数组、指针数组、结构数组等各种类别。数组在查找时的时间和空间复杂度很低，但由于初始化时需制定长度且长度不可变，插入和删除数据都很不灵活[51]。

栈(Stack)：是只能在某一端插入和删除的特殊线性表。它按照后进先出的原则存储数据，先进入的数据被压入栈底，最后的数据在栈顶，需要读数据的时候从栈顶开始弹出数据（最后一个数据被第一个读出来）。

队列(Queue)：一种特殊的线性表，它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作。进行插入操作的端称为队尾，进行删除操作的端称为队头。队列中没有元素时，称为空队列。

链表(Linked List)：是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。链表由一系列结点（链表中每一个元素称为结点）组成，结点可以在运行时动态生成。每个结点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个结点地址的指针域。

线程的调度需按照一定的顺序进行，因此要以一定的数据结构来存储线程。采取数组的方法，插入和删除元素都很不便，增加信息量只能增加数组的维度，调用复杂，且数组的长度需固定，不够灵活，为保证队列长度能够满足需要，则必须将数组的长度设置为一个较大的值，难免会造成存储空间的浪费。栈和队列只允许在表的一段进行插入和删除动作，不能适应使命控制中子任务和事件根据优先级插入到动作执行序列的合适位置的需求。链表的优势在于可以自由地增加和删除节点，且本例中所使用的链表长度较短，查找时的时间复杂度和空间复杂度均很低。由于在使命执行过程中是按照动作序列顺序执行各个子任务和事件处理动作，故选择单向链表构建动作执行序列。

链表中每个节点的形式为：

```
typedef  struct  actionarray
{
    int  num;
    struct  actionarray  *next;
}ActionArray;
```

其中 num 存储优先级信息，*next 存储下一个节点的地址。初始化链表时，头指针(head)的下一个节点指向尾节点(end)，尾节点中 num 的值是定义为使命完成标志的一个宏——FINISH。之后按优先级次序将子任务线程加入链表。当前要创建和执行的线程为与 head 节点后的第一个节点(head->next)内优先级(head->next->num)对应的线程。

与动作序列初始化相关的函数有：

```
void  InitAction(ActionArray  *head,  int  tasknum)
{
     ActionArray  *ins;      //插入项的指针
     ins  =  (ActionArray  *)malloc(sizeof(ActionArray));
     ins->num  =  tasknum;
     ins->next  =  head->next;
     head->next  =  ins;
}
```

函数 InitAction 的功能是初始化动作序列，将最基本动作加入序列，动作的插入顺序与 InitAction 的调用顺序相反。入口参数 ActionArray *head 为链表头指针, int tasknum 为待插入线程的优先级。

```
void  InsertAction(ActionArray *head, int tasknum)
{
        ActionArray  *ins,  *p;      //插入项的指针
```

```
    int    x;
    ins = (ActionArray *)malloc(sizeof(ActionArray));
    p = head->next;
    ins->num = tasknum;

    while(p)
    {
        if(p->num == 0)                    //若链表为空时的插入方法
        {
            ins->next = p->next;
            p->next = ins;
            x = p->num;
            p->num = ins->num;
            ins->num = x;
            break;
        }
        else if(tasknum < p->num)      //优先级小于当前节点的优先级则前插
        {
            ins->next = p->next;
            p->next = ins;
            x = p->num;
            p->num = ins->num;
            ins->num = x;
            break;
        }
        else
            p = p->next;
    }
}
```

函数 InsertAction 的功能是根据优先级顺序向动作序列中插入线程，用于在任务执行过程中向动作执行序列的正确位置插入事件处理线程。入口参数 ActionArray *head 为链表头指针, int tasknum 为待插入线程的优先级。

```
void    DeleteAction(ActionArray *head)
```

```
{
        ActionArray    *del;            //待删除项的指针
        del = head->next;
        head->next = del->next;
        free(del);
}
```

函数 DeleteAction 的功能是删除头指针后的第一个节点（即当前正在执行的线程节点）。入口参数 ActionArray *head 为链表头指针。

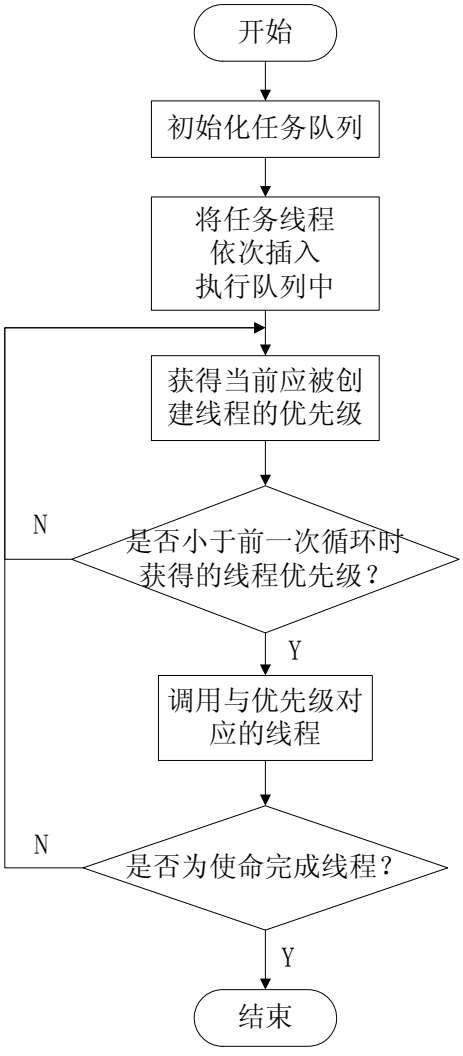


图 4.3　主线程流程图

当一个线程主体运行完成后会在线程结束前，即 pthread_exit()调用前将链表中存储自身信息的节点删除，将 head->next 指向自己后面的节点。这样，下一个要被创建的进程将被创建，或者原先被阻塞的线程由于 priority 又恢复为原值而得以从 while 循环判断

处继续运行，从而实现被阻塞的线程恢复执行。该线程阻塞和恢复的方法实现简便，且在不用保留被阻塞线程的断点信息的情况下保证了数据不会出现丢失。

当链表中的子任务和事件全部执行完之后，则

head->next->num = end->num = FINISH

程序检测到使命完成标志后，触发使命完成（Finish）事件，使命控制程序结束。综上，程序主循环的流程图如图 4.3 所示。

4.3.3 子任务与事件的优先级排序

子任务和事件的执行是根据其优先级进行排序的，优先级信息被存储在事件库中，各子任务、事件代号及优先级的设置情况为：

表 4.5　各线程关系及优先级

可并行线程	主要线程（子任务）		冲突线程（事件处理）	
计时器（Timer）	归航（Recover）	5	使命完成（Finish）	0
定时通信（Com）	勘察（Survey）	6	执行机构故障（Breakdown）	1
定时显示（ShowMsg）	航渡（Loiter）	7	传感器故障（SonarBreak）	2
	协同（Cooper）	8	避障（ObAvoid）	3
			GPS 校正（GPS adjust）	4

线程后的数字代表各个线程的优先级，数值越小则优先级越高。从上表可以看出，主要线程即子任务线程，子任务分为航渡、勘察、归航和协同四项，此四项构成了整个使命控制程序的主体。冲突线程即由事件触发的且需改变当前使命执行过程的线程，执行机构故障和传感器故障将终止整个使命的执行，并触发相应的事件处理动作；避障和 GPS 校正等是需阻塞当前正在执行的子任务进行的事件；使命完成事件优先级最高，用以终止作业 AUV 的整个使命执行过程。而可并行线程，定时通信事件和定时显示事件无需改变作业 AUV 执行机构的当前动作，是可以与正在进行的任何子任务或事件处理并行的线程。

4.3.4 子任务与事件线程的调度

4.3.4.1 常用的线程调度方法

在为多道程序并行运动所设计的系统中，可以有多个进程需要运行，但是单处理器的硬件系统在每一时刻只能让一个进程占用处理器。因此系统需要对进程进行调度。常用的进程调度算法有先来先服务、优先级、时间片轮转及多级反馈调度等算法[52]。

（1）先来先服务调度算法

这种调度算法是按照进程进入就绪队列的先后次序选择可以占用处理器的进程。当有进程就绪时，把该进程排入就绪队列的末尾，而进程调度综述把处理器分配给就绪队列中的第一个进程。一旦一个进程占有了处理器，它就一直运行下去，直到因等待某事件或进程完成了工作才让出处理器资源。由于要求运行事件很长的长作业执行时间非常长，而在先来先服务的调度方式下，它会一致占有处理器直到其运行结束，因此先来先服务的调度方式非常不利于短作业。

（2）优先级调度算法

对每个进程确定一个优先级，进程调度每一次切换进程都是让具有最高优先级的进程使用处理器；如果进程具有相同的优先级，则对这些有相同优先级的进程再按先来先服务的次序分配处理器。

一般的，系统在进程创建时为其确定一个优先级，进程的优先级可以是固定的，也可以随进程的执行过程变化。进程的优先级一直保持不变的调度方式称为静态优先级调度；在运行过程中根据情况改变的调度方式叫做动态优先级。

同时，优先级调度算法分为“非抢占式”和“抢占式”两种。其中在非抢占式的系统中，进程一旦开始运行就不会被打断，直到其结束才会调入新的进程。而可抢占式调度正好相反。可抢占式的调度方式对系统的利用率更高，但是也加大了系统切换进程的开销。

（3）时间片轮转调度算法

这是并行系统中最常用的调度算法。系统规定一个“时间片”的值。调度算法让就绪进程按就绪的先后次序排成队列，每次总是选择就绪队列中的第一个进程占用处理器，但规定只能使用的一个“时间片”。如果一个时间片用完，哪怕进程尚未结束也必须让出处理器而被重新排到就绪队列的末尾，等待再次运行，当再次轮到运行时，重新开始使用一个新的时间片。这样，就绪队列中的进程就依次轮流地占用处理器运行。这样对所有的进程来说都是公平的。

（4）多级反馈调度算法

在优先级调度算法和时间片轮转调度算法的基础上加以改进而衍生出的调度算法。在这种调度算法下，当前运行进程每运行一个时间片就减少一定的优先级，直到等待队列中出现比当前运行进程优先级更高的进程；每个进程创建时都有一个初始的优先级；每个就绪进程多等待一个时间片就增加一定的优先级。这样可以提高系统吞吐量、缩短进程的响应时间，同时对所有进程而言也是公平的。

4.3.4.2 基于优先级的抢占式调度与实现

在 AUV 的使命控制中，要实现子任务的顺序执行，并正确响应事件，需要设计子

任务和事件的排序方式，实现按照优先级调度子任务和事件处理动作，且高优先级的事件可以阻塞当前正在执行的子任务或低优先级的事件处理动作而抢先执行；高优先级的事件结束后，被阻塞的任务或事件处理动作可以从断点处继续执行且不会出现数据的丢失等异常情况。故应使用基于优先级的抢占式线程调度方法。

基于优先级的抢占式线程的调度方法示意图如图 4.4 所示：

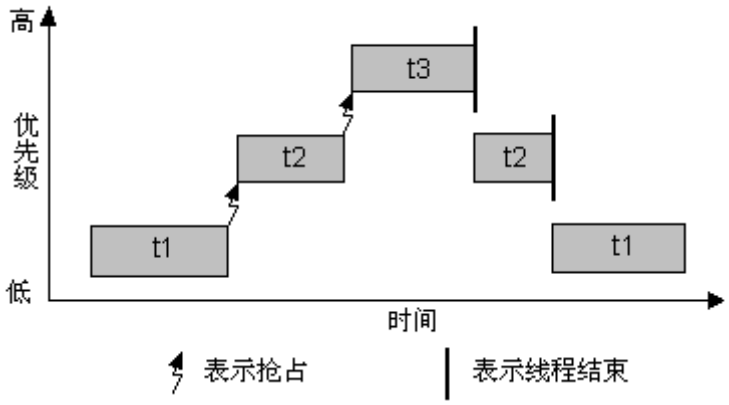


图 4.4　基于优先级的抢占式调度

图中 t1，t2，t3 表示三个线程，优先级顺序 t3>t2>t1。在 t1 运行时，当优先级更高的线程 t2 就绪，当前正在运行的线程 t1 将转化为阻塞态，同时运行 t2，t2 抢占了系统资源。t3 以同样的方式阻塞 t2。当优先级最高的线程 t3 结束后，先前被阻塞的线程 t2 恢复运行；t2 运行结束后，t1 恢复运行。

具体实现方法为：在程序的中，设置一个全局整型变量 priority 记录当前需调用的线程的优先级；在程序主循环中声明一个局部整型变量 pre 记录当前正在运行的线程的优先级。当二者不同时（通常是 priority ＜ pre，因为程序会按照优先级将事件插入到链表中），主程序会调用新优先级对应的线程。每个子任务和事件线程的主体都是一个循环，在循环的开始处都有一个判断，例如，在航渡线程（Loiter）中有：

```
while (priority != LOITER);
```

这里 LOITER 是一个宏，定义了航渡线程的优先级。当正在执行航渡线程时，若有事件相应线程插入，且其优先级高于航渡，则当前应执行线程的优先级不再是 LOITER，则航渡线程就会被阻塞，从而实现了高优先级的线程被创建并阻塞低优先级的线程。虽然使用这种方法并没有将低优先级线程的系统资源回收，尚不是真正意义上的阻塞线程，但是以目前计算机的配置，处于 while 循环中线程所占用的系统资源是微不足道的，从实现的角度来讲该方法可行并且简便，因为其省去了保存被阻塞线程数据以及线程重新运行时数据恢复的复杂步骤。每个线程在即将结束、释放所占用系统资源前，都会将当前应执行线程的优先级设置为下一个将要执行的线程，故事件响应线程结束后，当前

应执行线程的优先级又恢复为 LOITER，航渡线程不再停留在 while 死循环中，航渡线程得以继续执行。

4.4 基于仲裁规则的多 AUV 群体协同使命重规划

AUV 群体协同的使命控制的目的在于，当使命执行过程中有 AUV 出现故障无法完成自己的任务时，可以根据当时状况派遣其他作业 AUV 完成未被完成的那一部分任务。当作业 AUV 出现故障时，将立刻向主控 AUV 报告。主控 AUV 将对受损 AUV 的任务执行情况进行分析，根据其勘察区域已完成的扫描范围制定出重规划使命。接着，主控 AUV 根据其他未受损的作业 AUV 的状况，通过仲裁决定派遣哪个作业 AUV 完成协同任务。作业 AUV 在完成自己的任务后，到达回收点，与主控 AUV 通信报告任务完成状态，并等待是否有协同任务信息；若收到协同任务信息，作业 AUV 将从自身的回收点出发执行协同任务；协同任务完成后再次回到回收点，程序运行结束。

程序设计初期采用的仲裁规则仅仅考虑作业 AUV 的回收点距离协同任务的勘察区域的距离，因而会出现回收点距离协同任务勘察区域近的作业 AUV 尚有较多任务未完成，而回收点距离协同任务勘察区域远的作业 AUV 任务已经完成却没有被分配到协同任务。因此，结合了距离因素和任务完成情况，制定了新的仲裁规则：首先查看正常工作的 AUV 使命完成程度，结合回收点距离重规划的勘察区域的距离，计算出 AUV 完成剩余任务的时间，以及从其回收点到达重规划的勘查区域的耗时，从而实现一种比较优化的规划结果。

4.5 本章小结

本章用多线程调度的方法实现了 AUV 的使命控制。从理论上证明了 AUV 群体能够以顺序执行各项子任务，并根据优先级的顺序响应各种事件，从而顺利完成地形勘察作业。控制方法的实际效果将在下一章的仿真实验中给出。

第 5 章 多 AUV 群体协同使命控制仿真验证

5.1 引言

在前面几章研究内容的基础上，设计出了 AUV 群体协同使命控制仿真平台程序，用以验证使命控制方法的正确性和可靠性。该程序由一个主控 AUV 进程和若干作业 AUV 进程构成，其中作业 AUV 的数量可在程序中进行设置，最多支持 3 个。仿真程序的核心算法基于 Linux 下的 C 语言编写，界面使用 Qt Creator 设计。

5.2 多 AUV 群体协同使命控制仿真程序开发

5.2.1 主控 AUV 运行仿真实现

在多 AUV 使命控制程序中，主控 AUV 的名称为“AUV Commander”。程序启动后会出现对话框，用于设置执行使命的作业 AUV 数量，最多支持 3 个，默认值为 3，且主控 AUV 界面的控件布局会根据作业 AUV 的数量变化。点击确认后显示出程序主界面。

AUV 主控 AUV 界面如图 5.1 所示。作业 AUV 使命初始化和作业 AUV 使命执行信息显示部分采用了 TabPage 的布局方式，目的是为了节省界面空间和便于功能的剪裁和扩展，若未来需要扩展系统规模，增加作业 AUV 数量，只需以相同的方式多创建出若干 TabPage 即可满足需要。TabPage 的数量根据作业 AUV 数量设置对话框中输入的 AUV 数量生成。AUV 使命初始化部分分别初始化各个作业 AUV 的使命，AUV 使命信息显示每隔一定时间显示从各个作业 AUV 接收到的使命执行信息。地图部分即 AUV 工作空间，该图片来自 GoogleEarth 卫星地图，实际位置为我国山东半岛附近海域。使用鼠标指针在地图上移动，坐标显示区可显示当前鼠标指针在地图上的坐标。此坐标值通过坐标变换，将屏幕坐标系转换为实际地理坐标，以小数形式表示。使命执行相关功能键用于启动作业 AUV 和打开主控 AUV 与作业 AUV 的通信端口。

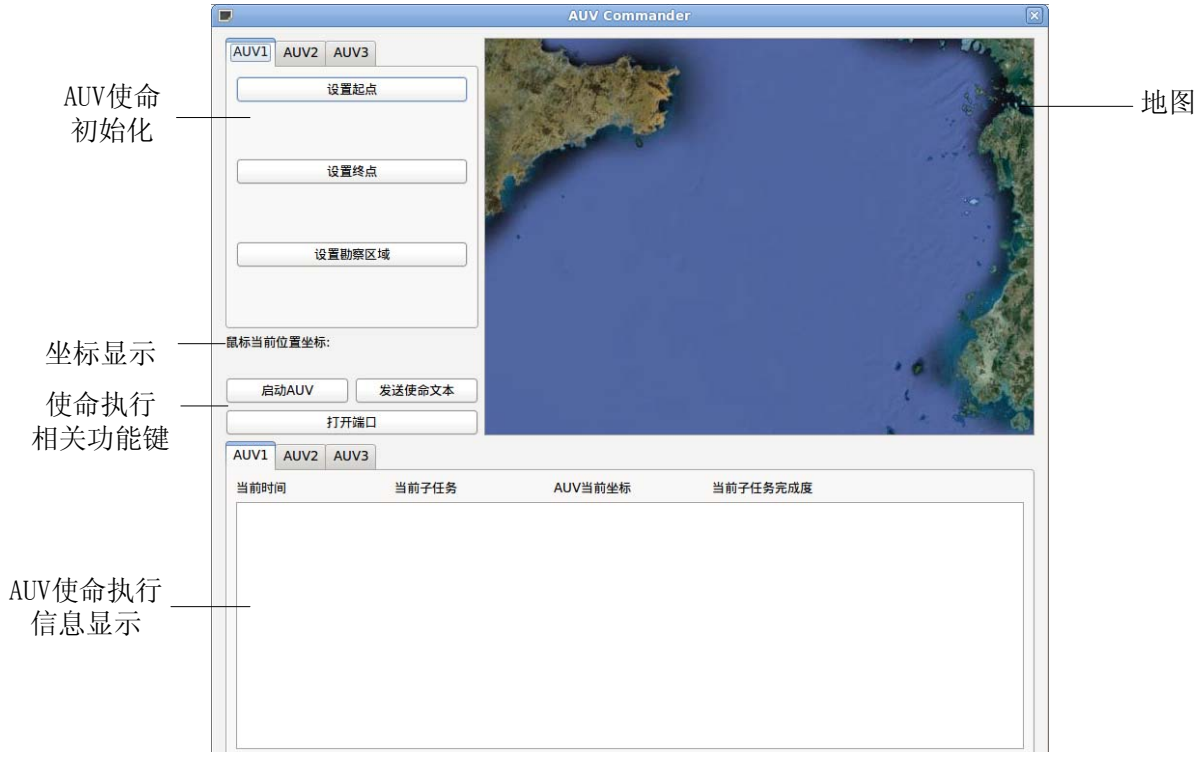


图 5.1　主控 AUV 界面

（1）AUV 使命初始化

在各个 AUV 的 TabPage 下点击“设置起点”按钮后，在图上单击鼠标，即将该点设置为作业 AUV 起点，以一圆形标记表示，具体坐标显示在按钮下；点击“设置终点”按钮后，在图上单击鼠标，即将该点设置为作业 AUV 回收点，以一正方形标记表示，具体坐标显示在按钮下；点击“设置勘察区域”按钮，在地图上拖动形成的矩形区域即为作业 AUV 勘察区域，勘察区域四个顶点的坐标显示在按钮下。各个作业 AUV 的使命信息在地图上以不同颜色的图形表示。实际效果如图 5.2 所示。

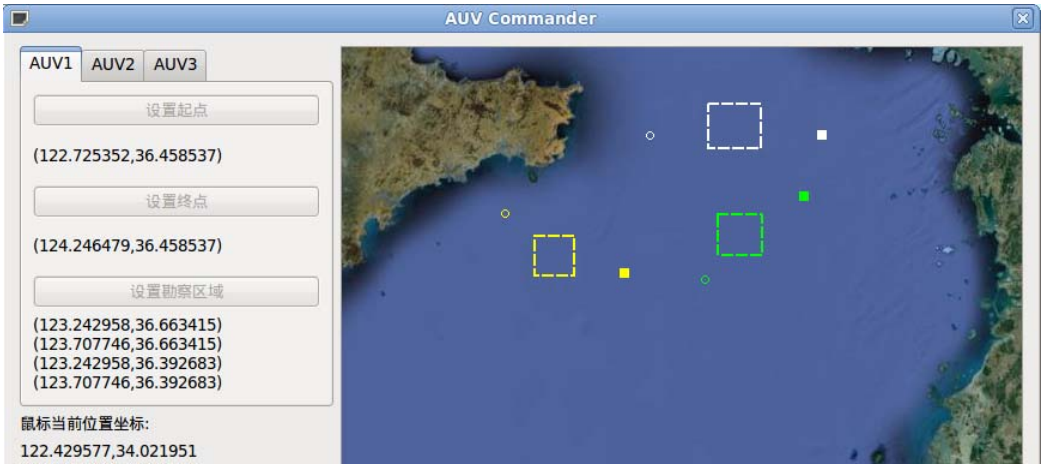


图 5.2　AUV 使命初始化效果图

（2）使命开始

使命开始执行前，点击“启动 AUV”按钮，主控 AUV 进程将按照设置的数量启动若干作业 AUV 进程，每个作业 AUV 进程以编号进行区分。如图 5.3 所示。

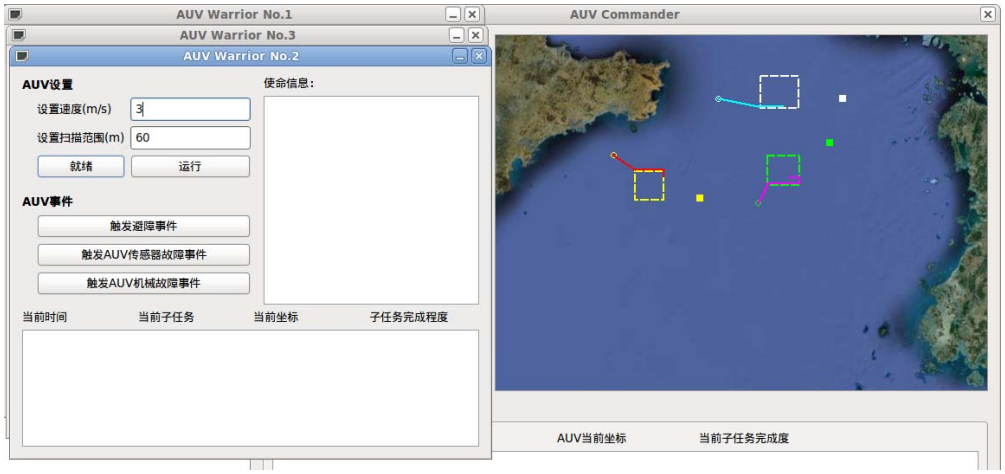


图 5.3　主控 AUV 启动 AUV 进程

点击“发送使命文本”后，主控 AUV 将各个作业 AUV 的使命执行信息制作成 txt 格式文本，通过进程间通信的方式发送给各个作业 AUV 程序。点击“打开端口”后，主控 AUV 即可接收来自各个作业 AUV 的使命执行信息。在各个作业 AUV 界面上完成基本参数设置后，点击“运行”按钮，作业 AUV 即开始执行使命，并定时与主控 AUV 通信，报告当前使命执行情况。主控 AUV 将作业 AUV 使命执行信息进行显示，并在地图上绘制出各个作业 AUV 的路径。如图 5.4 所示。

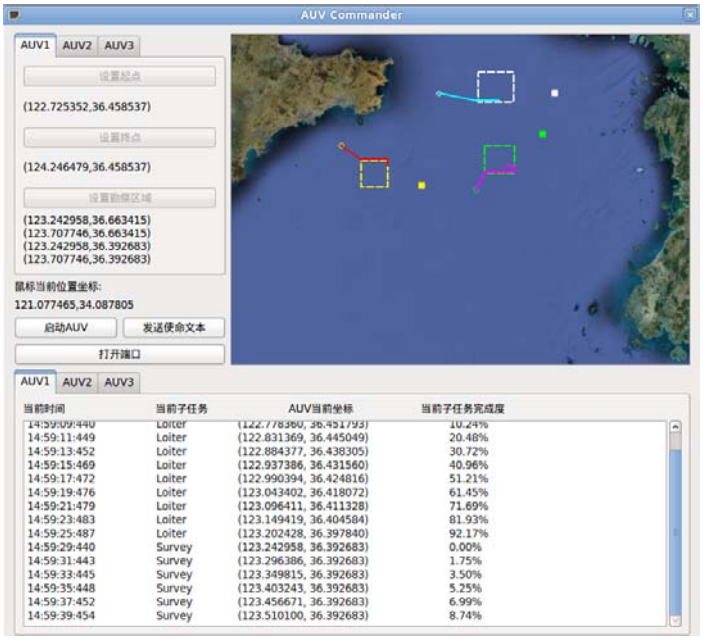


图 5.4　使命开始执行后的主控 AUV 界面

5.2.2 作业 AUV 个体运行仿真实现

在多 AUV 使命控制程序中，作业 AUV 进程的名称为“AUV Warrior”。作业 AUV 程序界面由 AUV 设置部分、事件触发按钮、规划和使命信息显示部分和使命执行信息显示部分组成。作业 AUV 程序界面如图 5.5 所示，每个作业 AUV 以编号进行区分，如下图所示的作业 AUV 编号为 No.1。

图 5.5　作业 AUV 单元程序界面

（1）作业 AUV 设置

AUV 设置用于设置作业 AUV 的移动速度和侧扫声呐扫描范围。本课题中不考虑运动模型，作业 AUV 始终以设定速度匀速运行。一般的小型 AUV 航速通常在 5 节/小时，故将 AUV 速度的默认值为 3m/s；侧扫声呐的扫描范围默认值为 60m。点击“就绪”完成作业 AUV 基本参数设置，点击“运行”后 AUV 开始执行任务。使命执行时的界面如图 5.6 所示。

（2）任务信息显示

如图 5.6 所示，点击“运行”按钮后，使命信息文本框中即出现经过作业 AUV 处理过的使命信息文本，计算出航渡子任务、勘察子任务和归航子任务的起点和终点坐标。若使命完成后接收到重规划信息，也将以上述方式处理并予以显示，如图 5.6、5.7 所示。

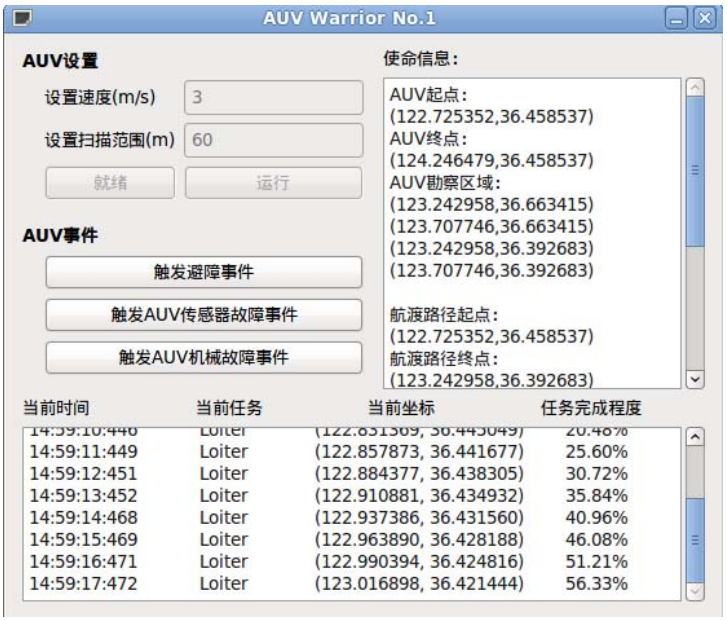


图 5.6　任务执行时的界面

图 5.7　重规划任务执行时的界面

（3）事件触发

AUV 事件通过 3 个按钮进行触发，包括避障事件、传感器故障事件和机械故障事件，其设置目的是为了检验高优先级线程能否正确插入到动作执行序列中的正确位置执行并阻塞低优先级线程。

其中，“触发避障事件”按钮用于测试线程调度算法能否响应事件处理线程并阻塞子任务线程。避障事件响应线程内容为显示作业 AUV 当前正在执行避障动作，并指示线程运行的剩余时间，如图 5.8 所示，在当前的使命环境中，勘察子任务（Survey）在每个检测周期中大约完成 2.55%，可以看出避障事件线程结束后，勘察子任务（Survey）能够从断点处继续执行，子任务完成程度的递增速度没有发生变化，对子任务的正常执行未造成任何影响。

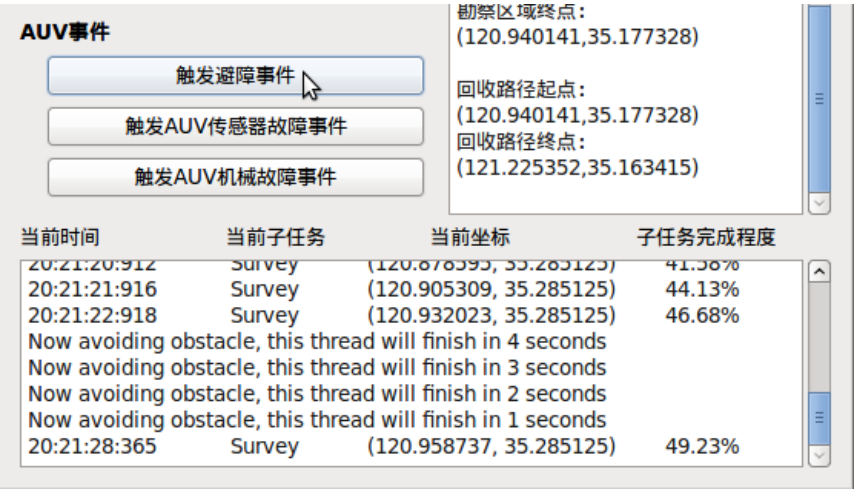


图 5.8　对避障事件的响应

“触发 AUV 传感器故障事件”按钮用于模拟作业 AUV 侧扫声呐出现故障时的情况。此时作业 AUV 无法继续完成地形勘察使命，然而执行机构并未受损，因此将向主控 AUV 报告自身故障状况，并自主进行使命重规划。使命重规划的内容为删除动作执行队列中的所有子任务，将传感器故障处理线程插入到动作序列中并执行，直接驶向回收地点。如图 5.9 所示，当按下“触发 AUV 传感器故障事件”按钮后，作业 AUV 当前正在执行的勘察任务被取消，转而执行“SonarBreak”线程，驶向回收点等待回收。

“触发 AUV 机械故障事件”按钮用于模拟作业 AUV 执行机构出现故障时的情况。此时作业 AUV 无法航行，既不能继续完成使命也无法驶向回收区域，只能显示出故障信息，并向主控 AUV 报告故障状况和当前坐标。如图 5.10 所示。

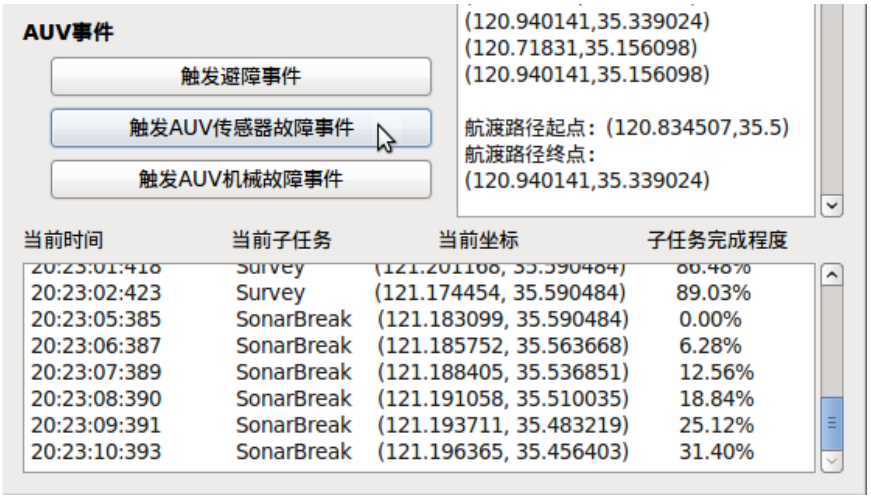


图 5.9　对声呐故障事件的响应

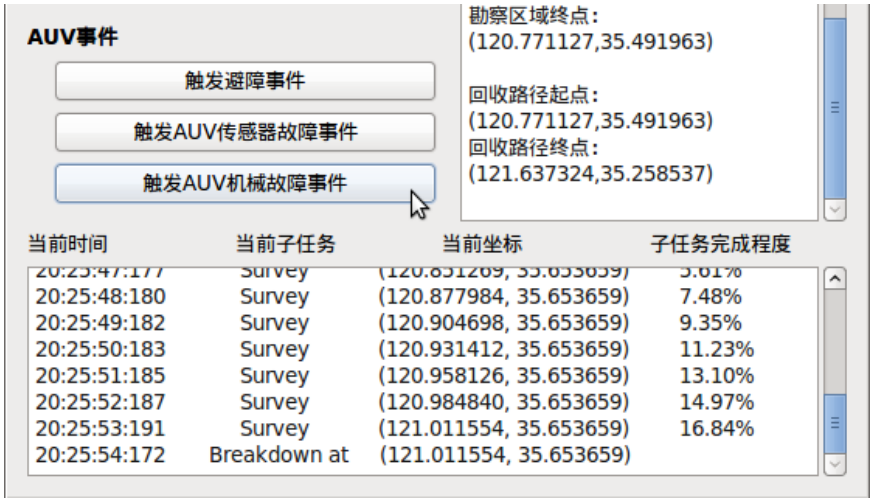


图 5.10　对执行机构故障事件的响应

5.3 仿真实验及其结果分析

5.3.1 AUV 群体协同地形勘察使命仿真实验

（1）实验目的

通过多 AUV 群体执行协同地形勘察使命，验证仿真平台软件的功能是否满足需求，运行情况是否稳定。

（2）仿真实验

运行 AUV 使命控制程序，设置 AUV 数量为 3 个，在主控 AUV 界面上完成使命初始化，三个 AUV 的起点、勘察区域和终点分别用不同的颜色加以区分，其位置及坐标如图 5.11 所示。其中 S 代表起点(Start)，E 代表终点(End)，Z 代表勘察区域(Zone)。不同 AUV 的任务信息用编号加以区分。

完成使命初始化后，点击主控 AUV 的“起动 AUV”按钮，起动作业 AUV 进程 No.1、No.2 和 No.3；点击发送使命文本，主控 AUV 将地形勘察使命规划为各个作业 AUV 的任务信息，制作成文本通过进程间通信的方式发送给各个作业 AUV 进程。本例中，各个作业 AUV 接收到的任务信息文本为：

AUV1 任务信息文本:

#MISSION

120802817,35104878

122091549,35060976

121183099,35258537,121658451,35258537,121183099,35009756,121658451,35009756

$MISSION

AUV2 任务信息文本:

#MISSION

124056338,35792683

123940141,35046341

123739437,35536585,124235915,35536585,123739437,35287805,124235915,35287805

$MISSION

AUV2 任务信息文本:

#MISSION

122841549,34175610

122746479,35075610

122545775,34709756,123095070,34709756,122545775,34468293,123095070,34468293

$MISSION

图 5.11　AUV 使命信息初始化

最后点击“打开端口”按钮，主控 AUV 进程便可以接收来自各个作业 AUV 单元的使命执行信息。自此主控 AUV 上的所有操作已完成。

如图 5.12 所示，首先对各个作业 AUV 的航速和传感器扫描范围两项基本参数进行设置，接着点击“就绪”按钮完成初始化，并将接收到的任务信息进行分析和规划，规划后的使命信息如下：

表 5.1　规划后的 AUV 任务信息

编号 / 项目	AUV1	AUV2	AUV3
起点	(120.802817,35.104878)	(124.056338,35.792683)	(122.841549,34.17561)
终点	(122.091549,35.060976)	(123.940141,35.046341)	(122.746479,35.07561)
勘察区域	(121.183099,35.258537) (121.658451,35.258537) (121.183099,35.009756) (121.658451,35.009756)	(123.739437,35.536585) (124.235915,35.536585) (123.739437,35.287805) (124.235915,35.287805)	(123.09507,34.468293) (122.545775,34.6838877) (123.09507,34.468293) (122.545775,34.6838877)
航渡路径起点	(120.802817,35.104878)	(124.056338,35.792683)	(122.841549,34.17561)
航渡路径终点	(121.183099,35.009756)	(124.235915,35.536585)	(123.09507,34.468293)
勘察区域起点	(121.183099,35.009756)	(124.235915,35.536585)	(123.09507, 34.468293)
勘察区域终点	(121.183099,35.2792494)	(124.235915,35.2670916)	(122.545775,34.6838877)
归航路径起点	(121.183099,35.2792494)	(124.235915,35.2670916)	(122.545775,34.6838877)
归航路径终点	(122.091549,35.060976)	(123.940141,35.046341)	(122.746479,35.07561)

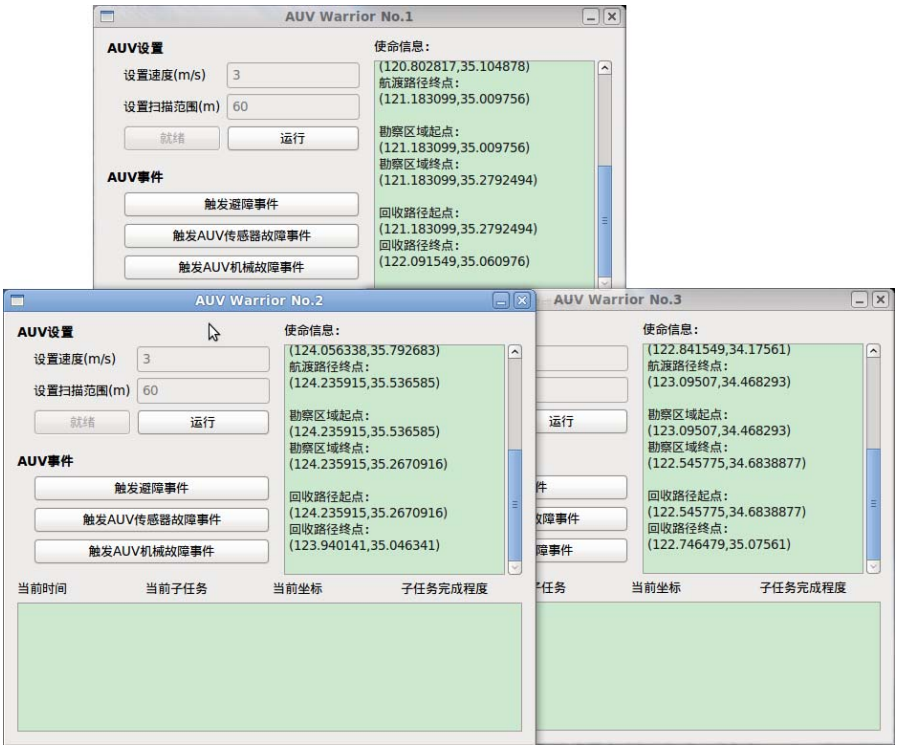


图 5.12　完成初始化的各个作业 AUV 单元

点击各个作业 AUV 的“运行”按钮后，使命开始执行，各个作业 AUV 显示当前任务的执行情况，并每隔 2s 与主控 AUV 进行一次通信，报告当前的使命执行情况，主控 AUV 在地图上绘制出各个作业 AUV 的路径，如图 5.13 所示。

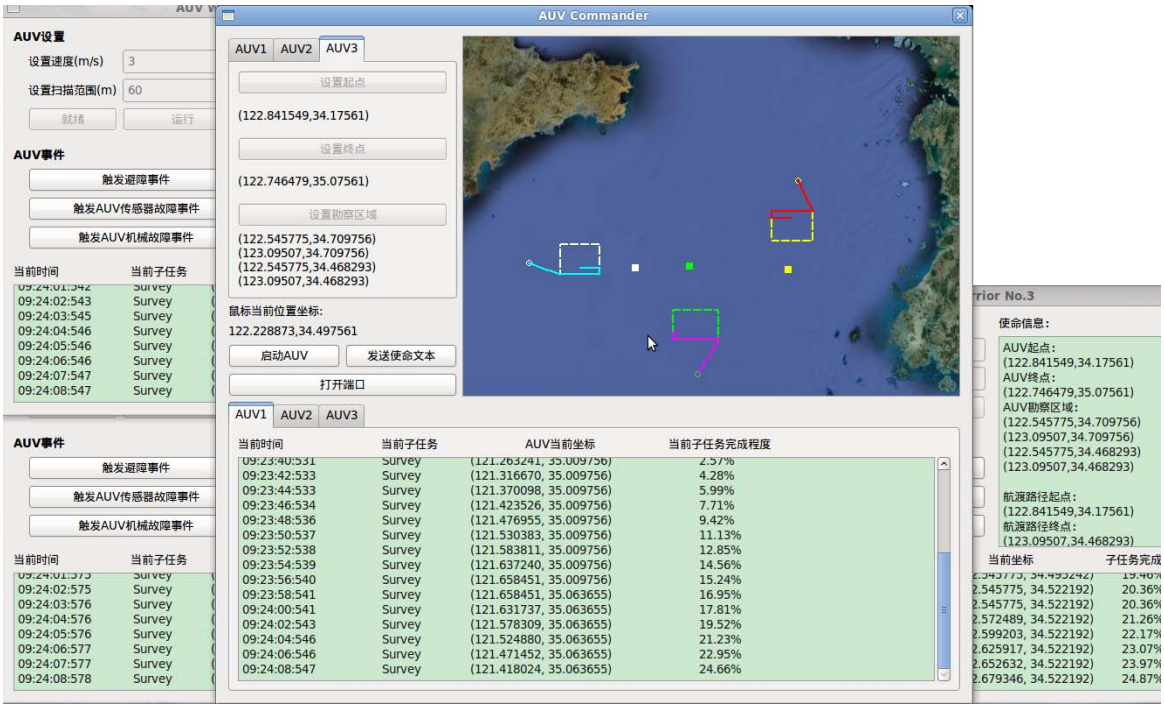


图 5.13　AUV 使命控制程序运行状态

（3）结果分析

通过上述实验可以验证，“AUV 群体协同使命控制仿真平台”程序能够完成设置 AUV 数量、初始化使命、起动作业 AUV 进程、主控 AUV 和作业 AUV 间通信以及正确执行使命并进行相应显示等功能，为验证使命控制方法正确性的实验提供了平台。

5.3.2 任务执行过程中的事件响应

（1）实验目的

验证作业 AUV 进程能否快速且正确地响应任务执行过程中出现的事件，阻塞当前正在执行的子任务线程并调用对应的事件处理线程；以及能否能根据事件的优先级确定事件处理线程的执行顺序。

（2）仿真实验

在作业 AUV 执行任务的过程中，打开 Linux 的终端，键入指令：ps –ef 查找作业 AUV 程序——AUV_Warrior 进程的 PID（Process ID——进程编号），以其中一个作业 AUV 进程为例，获得其 PID 值为 4395；接着使用指令：ps -Lf 4395 即可查看该进程的相关信息，如图 5.14 所示：

```
@ubuntu:~$ ps -Lf 4395
     PID  PPID   LWP  C NLWP STIME TTY      STAT   TIME
    4395  2321  4395  0    1 10:47 ?        S      0:00
```

图 5.14 获取进程 PID 及进程信息

在这些信息中，需要关注的几项为：LWP（轻量级进程编号）、NLWP（该进程中轻量级进程的数量）和 STAT（进程的状态）。这里需要指出的是，Linux 操作系统下的多线程是通过轻量级进程（LWP）来实现的[53]，轻量级进程是 Linux 系统调度的最小单元。此时，LWP 值为 4395，NLWP 值为 1，表示该 AUV_Warrior 进程的线程数量为 1，说明只有界面显示线程正在工作。STAT 值为 S，表示界面显示线程正在等待事件的发生。

当程序开始运行后，再次使用指令：ps -Lf 4395，如图 5.15 所示。此时的进程信息为：

```
@ubuntu:~$ ps -Lf 4395
     PID  PPID   LWP  C NLWP STIME TTY      STAT   TIME
    4395  2321  4395  0    2 10:47 ?        Sl     0:00
    4395  2321  4479  0    2 10:50 ?        Sl     0:00
```

图 5.15 程序运行后的进程信息

进程中出现了一个编号为 4479 的新线程，NLWP 值也变为 2，表明航渡线程（Loiter）被创建。此时两个线程的 STAT 都变为 Sl，其中 l 表示进程是多线程的。

图 5.16　事件线程插入后程序界面显示内容

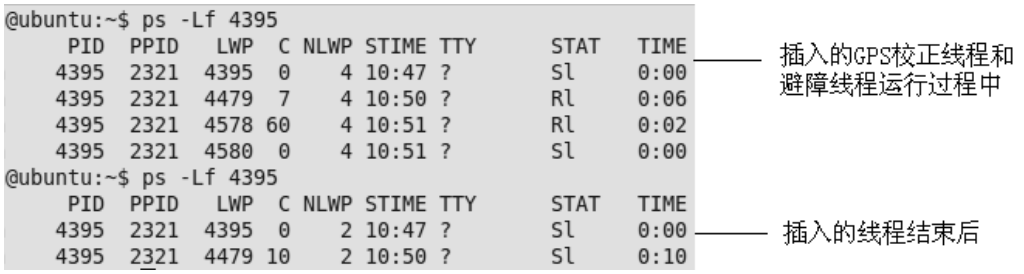


图 5.17　事件线程插入与结束后

在测试中，等待 GPS 校正事件定时触发，接着在 GPS 校正线程结束前，又通过按键触发避障事件。通过图 5.16 和图 5.17 可以看出，AUV_Warrior 进程中又先后出现了两个编号为 4578 和 4580 的新线程，可以看出优先级较高的 GPS 校正线程将 Loiter 线程阻塞；Loiter 线程由于响应了 GPS 校正线程创建这一事件，所以其 STAT 值由 Sl 变为 Rl，R 表示该线程处于正在执行或即将运行状态。接着当避障事件触发后，因其优先级高于 GPS 校正事件，此时 GPS 校正线程被避障线程阻塞，STAT 值也变为 Rl。NLWP 的值为 4，表示当前有 4 个线程正在运行。

AUV_Warrior 界面的显示结果表明，当避障线程结束后，GPS 校正线程继续运行；GPS 校正线程结束后，Loiter 线程能够从刚才的断点继续运行，此时的 AUV_Warrior 进程又恢复了最初只有界面和 Loiter 这 2 个线程运行的状态，两个线程的 STAT 值也恢复为 Sl。本例中，航渡线程、GPS 校正线程和避障线程分别对应了图 4.4 中的 t1、t2 和 t3。

（3）结果分析

根据作业 AUV 进程界面的显示内容以及 Linux 终端上显示的信息可以看出，本课题中使用的基于优先级的抢占式调度方法实现了预定功能。作业 AUV 在任务执行过程中，能够对感知到的事件进行快速而正确的响应，并且事件处理线程结束后，子任务线程可以从断点处继续执行，没有出现数据丢失以及线程不能正常启动、结束的错误，验证了算法的有效性和可靠性。

5.3.3 AUV 群体协同的仿真实验

（1）实验目的

当使命执行过程中有 AUV 出现故障无法完成自己的任务时，可以根据当时状况派遣其他作业 AUV 完成未被完成的那一部分任务。本例中，通过作业 AUV 进程上的事件触发按钮来模拟作业 AUV 出现故障不能继续执行任务的情况；当作业 AUV 出现故障时，将立刻向主控 AUV 报告。主控 AUV 将对受损 AUV 的任务执行情况进行分析，根据其勘察区域已完成的扫描范围制定出重规划使命。接着，主控 AUV 根据其他未受损的作业 AUV 的状况，通过仲裁决定派遣哪艘作业 AUV 完成协同任务。

（2）仿真实验：

以 AUV3 出现故障为例，点击 AUV Warrior No.3 进程的“触发传感器故障事件”按钮，AUV3 将终止执行任务，起动传感器故障处理线程，直接驶向回收点，如图 5.20 所示。到达回收点后将向主控 AUV 报告任务失败信息，如图 5.21 所示。

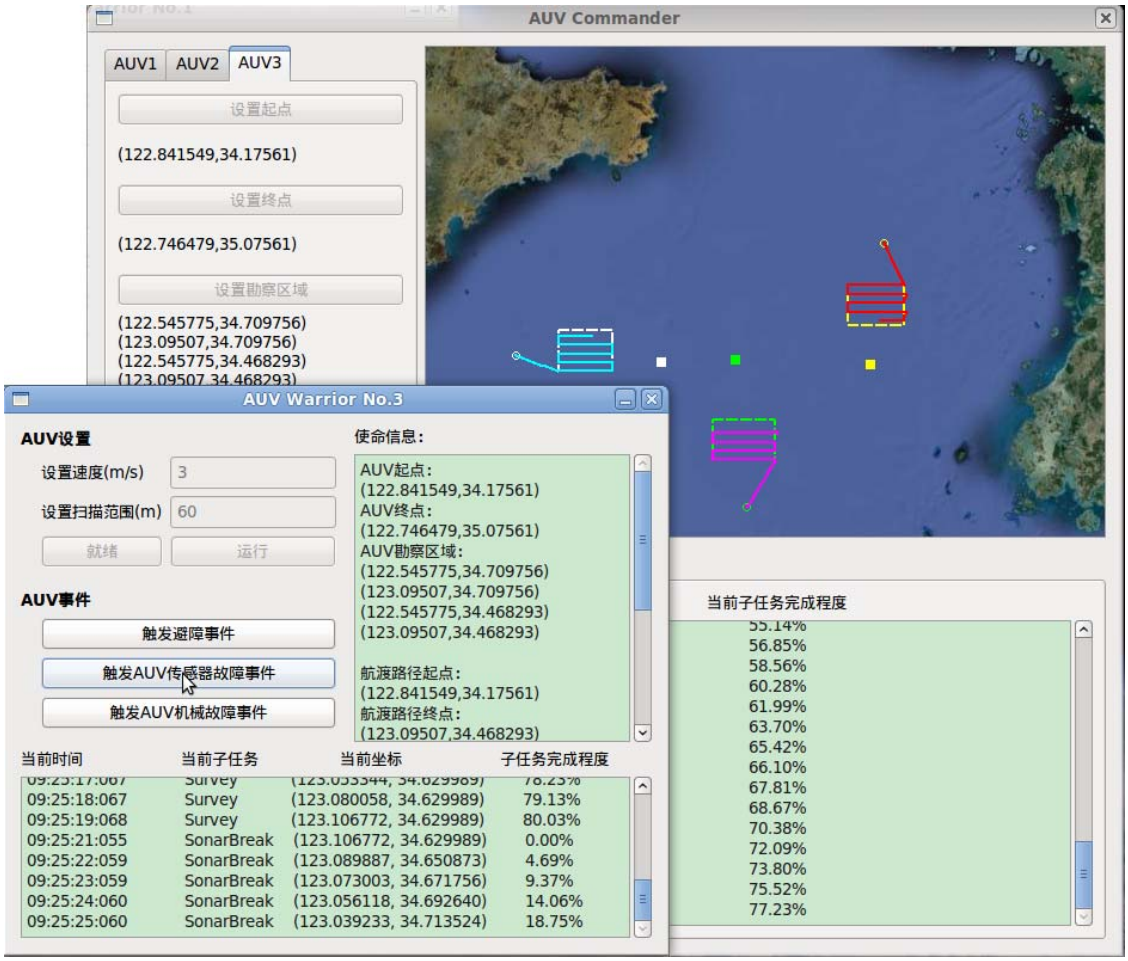


图 5.18　触发 AUV3 的传感器故障

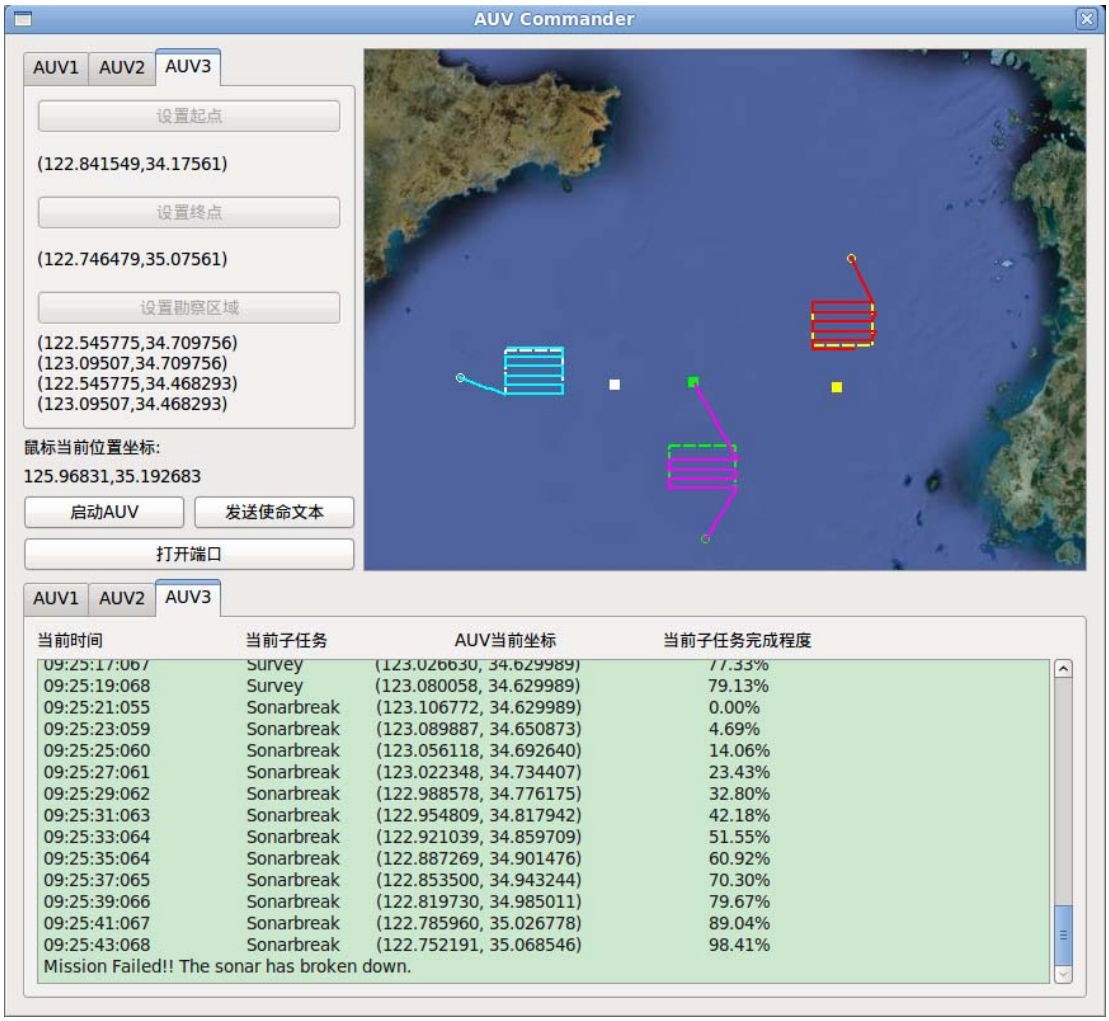


图 5.19　AUV3 到达回收点后向主控 AUV 发送的任务失败信息

当主控 AUV 接收到 AUV3 出现故障的信息后，根据其勘察任务的完成情况，计算出协同任务的勘察区域坐标；计算出勘察区域坐标后，计算正常工作的两艘 AUV 的回收点距离新的勘察区域的距离，选择距离较近的 AUV 执行协同任务，并根据执行协同任务的 AUV 的回收点坐标确定协同任务的起点和终点坐标；协同任务规划完成后，将立刻发送给执行协同任务的 AUV。如图 5.19 所示，AUV3 未能完成勘察任务，直接驶向回收点。AUV1 的回收点距离协同任务的勘察区域较近，所以被选为执行协同任务的 AUV；AUV1 执行完自身任务后，对接收到的协同任务使命进行分析，规划出自身的重规划使命。AUV1 接收到的使命文本内容为：

#REPLAN

122545775,34709756,123095070,34709756,122545775,34615779,123095070,34615779

$REPLAN

经过 AUV1 分解和规划后的协同任务信息为：

表 5.2　处理后的 AUV1 协同任务信息

起点	(122.091549,35.060976)
终点	(122.091549,35.060976)
勘察区域	(122.545775,34.709756) (123.09507,34.709756) (122.545775,34.615779) (123.09507,34.615779)
航渡路径起点	(122.091549,35.060976)
航渡路径终点	(122.545775,34.709756)
勘察区域起点	(122.545775,34.709756)
勘察区域终点	(123.09507,34.6019587)
归航路径起点	(123.09507,34.6019587)
归航路径终点	(122.091549,35.060976)

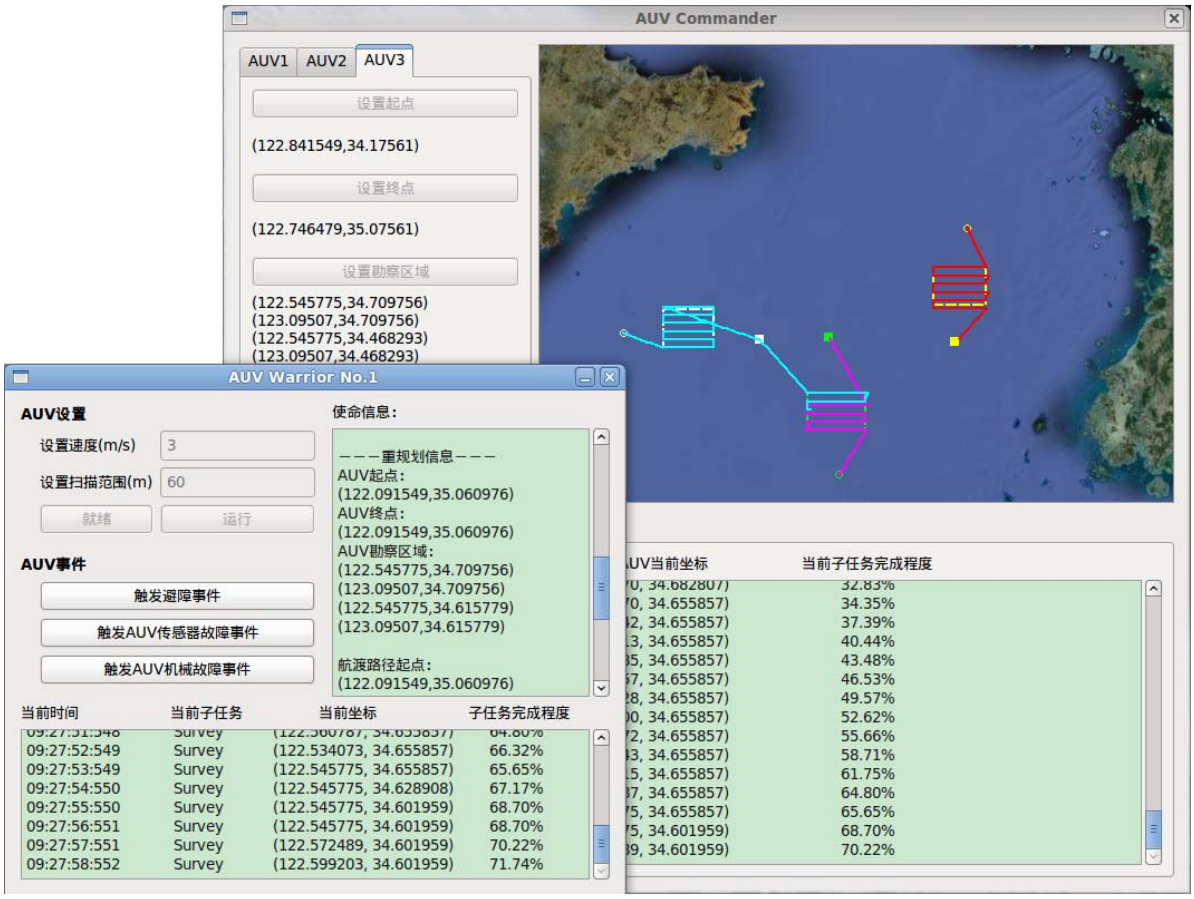


图 5.20　AUV1 被选择执行协同任务

（3）实验结果分析：

实验结果如图 5.21 所示。AUV3 出现故障后，停止执行勘察作业，驶向其回收点。AUV1 在完成了自身任务后，执行了协同任务，完成了 AUV3 未完成的勘察任务，回到其回收点；AUV1 进程正常结束，并给出使命完成信息，证明多 AUV 群体协同使命控制算法正确可行。

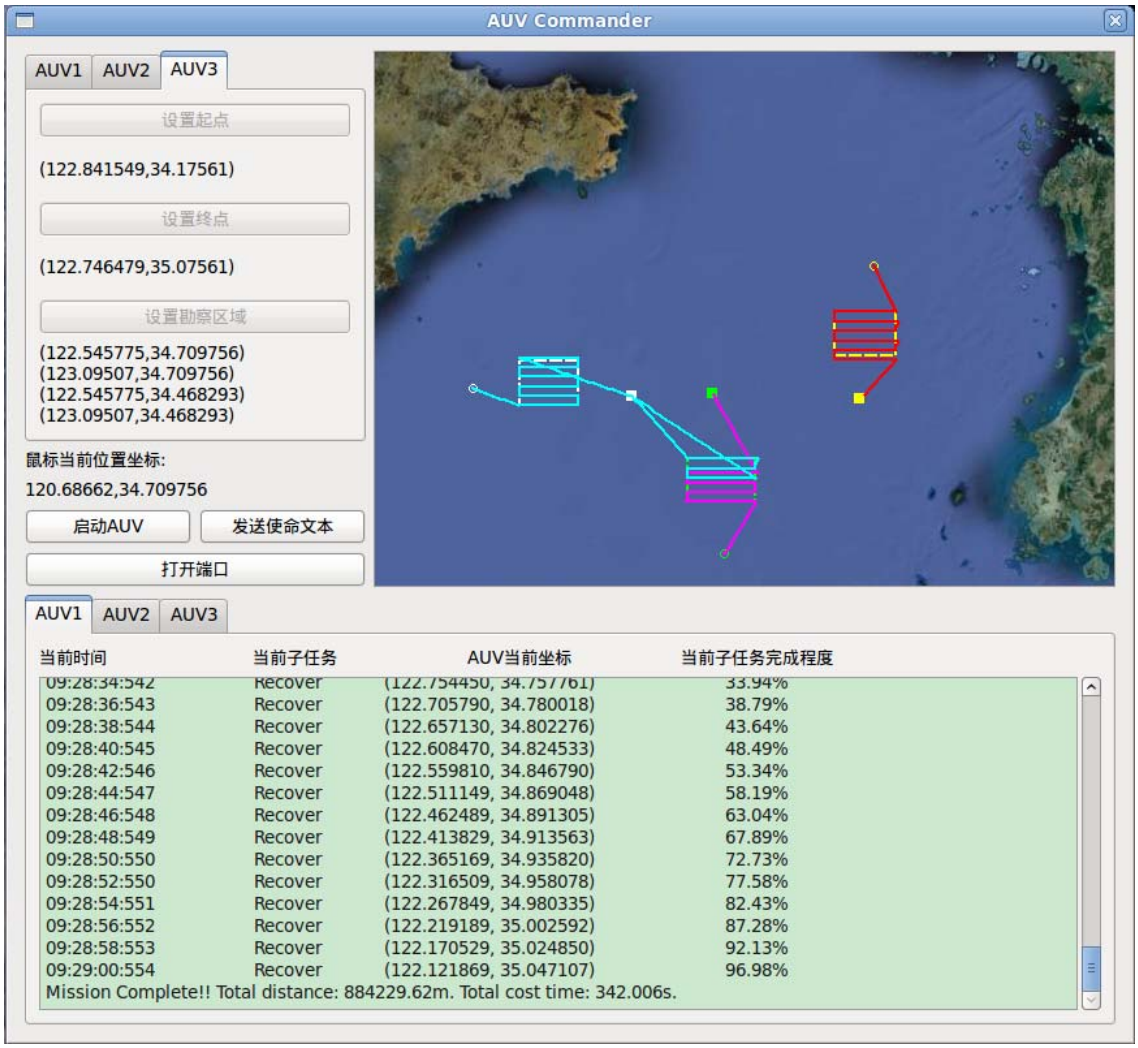


图 5.21　AUV1 完成协同任务

5.4 本章小结

本章对 AUV 群体协同使命控制仿真平台程序进行了介绍，通过实验的方式验证了程序各项功能均满足实验要求。在仿真平台上成功执行了多 AUV 群体协同地形勘察使命，验证了故障情况下的群体协同使命控制方法是否有效。最终，实验结果证明了该方法的正确性和可行性。

结　论

本文的主要研究内容为多基于地形勘察使命的 AUV 群体协同使命控制。对使命的划分、多 AUV 系统的层次结构、AUV 个体的结构以及群体协同使命控制方法进行了较为深入细致的研究。本文的成果主要包括以下几点：

（1）对典型的多机器人系统结构进行了分析，设计了集中式多 AUV 群体体系结构。设计了主控 AUV 和作业 AUV 的功能模块划分与结构，建立了多 AUV 系统模型。

（2）设计多 AUV 群体协同地形勘察使命的划分方式，将宏观上的使命划分为分配给各个作业 AUV 的任务，作业 AUV 将各自的任务划分为顺序执行的子任务，并对任务执行过程中出现的事件进行响应；设计了利用线程描述的子任务与事件处理过程的实现方法。

（3）利用 Petri 网建立了 AUV 使命控制过程模型，基于此模型设计了适用于 AUV 使命控制的基于优先级的抢占式调度方法，实现了子任务的顺序执行与事件的快速正确响应。

（4）完成了“AUV 群体协同使命控制仿真平台”程序的编写，通过程序的实际运行效果验证了多 AUV 群体协同使命控制方法的正确性和可靠性。

目前针对 AUV 使命控制，尤其是多 AUV 使命控制的研究尚不多见，希望本文的研究成果能为该领域的研究提供一些参考。由于时间仓促和个人能力有限，本课题的研究尚不很完善。作者认为多 AUV 群体协同使命控制的研究可以在以下方面加以完善：

（1）在仿真平台中加入 AUV 运动模型、执行机构（推进器与舵）模型，通过各模型的仿真解算获得 AUV 的运动状态，从而能够更真实地模拟 AUV 地形勘察使命控制过程中典型故障及事件处理的有效性和可行性。

（2）加入环境干扰和障碍模型，有海流扰动和障碍物情况下的使命控制过程将更为复杂，因此，加入上述模型可以更有效地验证使命控制方法的正确性，有助于该方法的实际应用。

（3）通过多台计算机设备模拟水下通信环境，即采用多个计算机构建多 AUV 群体，每台计算机上运行一个 AUV 个体的进程，模拟水声通信声纳实际通信性能，将更真实地模拟 AUV 群体的实际作业过程，同样有助于实际应用。

攻读硕士学位期间发表的论文和取得的科研成果

[1] 王宏健，高晗，吕洪莉．基于线程调度的 AUV 使命控制实现方法研究．自动化学院学术年会论文集．已录用

[2] 可伸缩叶轮式爬楼机器人．第十一届“挑战杯”全国大学生课外学术科技作品竞赛二等奖．国家级．2009 年 11 月．第五作者

参考文献

[1] 朱亦峰. 多 AUV 协同作业中的互定位方法研究. 哈尔滨工程大学硕士学位论文. 2009:6 页

[2] P. Oliveira, A. Pascoal, V. Silva, C. Silvestre. Mission Control of the MARIUS AUV: System Design, Implementation, and Sea Trials. International Journal of Systems Science. 1998, Vol. 29, No. 10: 1065-1080P

[3] P. Oloveira, A. Pascoal, V. Silva, and C. Silvestre, “Design, develop-ment, and testing at sea of the mission control system for the marius autonomous underwater vehicle”. in Oceans MTS/IEEE, 1996.

[4] Narc´s Palomeras PereRidao, MarcCarreras, CarlosSilvestre.Towards a Mission Control Language for AUVs , The International Federation of Automatic Control. 2008: 15028-15033P

[5] Chew J.L., Koay T.B., Tan Y.T., Eng Y.H., Gao R., Chitre M., Chandhavarkar N. STARFISH: An Open-Architecture AUV and its Applications. 2011

[6] Koay T.B., Tan Y.T.,Eng Y.H., Gao R.,Chitre M., Chew J.L., Chandhavarkar N., Khan R.R., TaherT and Koh J. STARFISH–A Small Team of Autonomous Robotic Fish. NISCAIR-CSIR, Apr. 2011,

[7] Don Brutzman .et.al, NPS Phoenix AUV Software Integration and In-Water Testing. IEEE,1996

[8] M.CARRERAS, N.PALOMERAS, P.RIDAO and D.RIBAS. Design of a mission control system for an AUV. International Journal of Control. 2007, Vol.80,No.7: 993-1007P

[9] Paul Michael Newman. MOOS - Mission Orientated Op erating Suite. Oxford University, March. 2006

[10] P.Ridao, J. Yuh, J. Batlle, K. Sugihara. On AUV Control Architecture. IEEE,2000

[11] Pere Ridao, Marc Carreras. Mission Control System for an AUV, University of Girona. March. 2007.

[12] Stephen McPhail, Development of a Simple Navigation System for the Autosub Autonomous Underwater Vehicle. In Proc. Engineering in Harmony with Ocean. 1993: 504--509P

[13] 吴小平，冯正平, 多 AUV 覆盖控制研究. 中国造船, 2009.6, Vol.50 No.2: 118-127 页

[14] 崔荣鑫等. 基于虚拟参考点的 AUV 编队控制. 火力与指挥控制. 2008.10, Vol.33

No.10：53-57 页

[15] Yeun-Soo Jung, Kong-Woo Lee and Beom-Hee Lee, Advances in Sea Coverage Methods Using Autonomous Underwater Vehicles (AUVs). Recent Advances in Multi-Robot Systems. 2008.3: 69-100P

[16] 田广 等. 自治式水下机器人避障行为机制研究. 传感器与微系统. 2009, Vol 28 No.12：59-63 页

[17] Fodrea, L. R.. Obstacle Avoidance Control for the REMUS Autonomous Underwater Vehicle. MSME Thesis, Naval Postgraduate School. Dec. 2001

[18] 谭民，王硕，曹志强. 多机器人系统.北京:清华大学出版社. 2005：21-36 页

[19] Zarm K A，Schmidt G. A Decentralized Approach for the Conflict free Motion of Multiple Mobile Robots.IEEE IROS96, 1996: 1667-1675P

[20] 王俊松 等. 基于 Multi-Agent 的多机器人控制技术. 天津职业技术师范学院学报. 2004 年 6 月，第 14 卷,第 2 期：3-7 页

[21] 苏治宝, 陆际联. 多移动机器人队形控制的研究方法.机器人. 2003 年 1 月,第 25 卷，第 1 期：88-91 页

[22] 薛红香.水下机器人任务规划方法研究.哈尔滨工程大学硕士论文，2009.01: 8 页

[23] 王宏健 等. 自主式水下潜器虚拟仿真系统研究. 系统仿真学报. 2004，16(5)：927-930 页.

[24] P. Hagen, AUV/UUV mission planning and real time control with the hugin operator system. IEEE OCEANS 2001, Nov 2001, Vol. 1.

[25] Stephen McPhail, Development of a Simple Navigation System for the Autosub Autonomous Underwater Vehicle. In Proc. Engineering in Harmony with Ocean. 1993: 504—509P.

[26] 高剑等，一种自主水下航行器分布式控制系统. 兵工学报. 2009.8, Vol.30 No.8:1139-1142 页

[27] 常宗虎，边信黔，严浙平，汪玉. AUV 实时任务协调方法研究.应用科技. 2004.12，31(12)：46-48 页

[28] 李炳森. 基于行为的模块化 AUV 决策控制系统及其仿真实现. 中国海洋大学硕士论文. 2009.6:10-12 页

[29] Carl E. Nehme, Jacob W. Crandall, M. L. Cummings. An Operator Function Taxonomy for Unmanned Aerial Vehicle Missions. 12th International Command and Control Research and Technology Symposium. Newport, RI, June, 2007

[30] Patron, P. Miguelanez, E. Petillot, Y.R. Lane, D.M. Salvi, J. Adaptive mission plan

diagnosis and repair for fault recovery in autonomous underwater vehivcles. OCEANS 2008: 1 – 9P

[31]Claude Barrouil et.al. An integrated navigation system for a long range AUV. OCEANS '98 Conference Proceedings, vol.1: 331 – 335P

[32]李凤英，古天龙，徐周波. Petri 网的符号 ZBDD 可达树分析技术. 计算机学报. 2009,32(12)，2009: 2420-2428 页

[33]常宗虎，施小成，刘光军. AUV 使命控制过程 Petri 网建模研究. 第二十三届中国控制会议论文集.2004: 1204-1209P

[34]Ho Y C. Performance evaluation and perturbati analysis of discrete event dynamic system. IEEE Transactions on Automatic Control.1987, 32(7)：563-572P

[35]邹海,边信黔,熊华胜. AUV 控制系统规划层使命与任务协调方法研究. 机器人. Nov 2006, Vol.28 No.6: 651-655 页

[36]阮灵等. 基于 Petri 网的工作流建模研究. 软件导刊. 2010, Vol.9,No.10：158-161 页

[37]陈黎静. 基于 Petri 网的异步并发系统的建模及在 Linux 进程队列中的应用. 长沙理工大学. 2004:17-18 页

[38]乐晓波，陈黎静. Petri 网应用综述. 长沙交通学院学报. June 2004,V01.20 No.2: 51-55P

[39]Matthew.N.，Stones,R.著, Linux 程序设计(第 4 版).陈健，宋健建 译. 北京：人民邮电出版社. 2010.6: 408-432 页

[40]肖文等. 多线程编程之——Linux 篇.中文信息：程序春秋. 2003 年 9 期:65-71 页

[41]许俊　基于嵌入式 Linux 的网络化电梯远程监测系统. 浙江大学. 2003: 42-43 页

[42]张海光. 浅谈 Linux 操作系统下的多线程编程. 华南金融电脑. 2006 年 3 期:98-101 页

[43]张卿等. Linux 线程属性的研究及其在 WAP 网关中的应用. 计算机应用研究. 2002 年 1 期:148-151 页

[44]张鹤高，基于 GPSGSM 双模移动定位技术的研究与实现. 贵州大学硕士研究生学位论文. 2006:26-28 页

[45]姚继锋，Linux 下的多线程编程. http://www.china-pub.com ,2001.08.

[46]杨奕，贺皓，张俊伟. Linux 的多线程编程的高效开发经验. http://www.ibm.com/developerworks/cn/linux/l-cn-mthreadps/，2009.4.

[47]党纪红，李东明，袁赣南.VxWorks 实时内核调度的研究分析.应用科技，30(2)，2003.2.：34-36 页

[48]张超. VxWorks 操作系统在雷达信号处理中的应用. 西安电子科技大学研究生学位

论文, 2007:35-39 页
[49]魏连秋. 数学算法对计算机编程优化的分析与研究. 科技创新导报. 2010 年 30 期:3-4 页
[50]谢晋. 试谈离散数学在计算机学科中的重要性. 黄石理工学院学报. 2006 年 1 期:90-93 页
[51]武丽娟. 论《C 语言程序设计》中的数组. 福建电脑. 2010 年 7 期：204-205 页
[52]杨沙洲.Linux 线程实现机制分析. http://www.ibm.com/developerworks/cn/linux/kernel/l-thread/ index.html. 2003.5
[53]刘晓娇. Unix 系统中进程调度算法的分析与评价. 科技信息. 2010 年 17 期:39-40 页

致　　谢

本课题从立题、设计到顺利完成，前后历时共一年有余。在这一年多的时间里，我的导师王宏建教授对我在工作中遇到的各种问题的耐心解答，悉心指导，一次又一次的为我指明了前进的方向，每次探讨后都使我感觉犹如醍醐灌顶，毕设工作也随之顺利进行。我从本科大四毕业设计期间进入实验室，到现在已经有三年多的时间，导师在学习上、工作上和生活上都给予了我无微不至的帮助和关心。在学术上，她敏锐的洞察力、渊博的学识、深厚的理论基础及严谨求实的治学态度都充分表现出了她儒雅的学者风范，令我由衷的敬佩；在工作上，她高尚的敬业精神、高瞻远瞩的视野以及对事业孜孜不倦的追求，深深地影响了我；在生活中，她丰富的阅历、高尚的品格和平易近人的态度给我留下了深刻的印象。在此，再次向恩师致以最衷心的感谢和最崇高的敬意！

回顾硕士研究生学习期间，409 教研室的老师和同学们给予了我很多关怀和鼓励。在此要感谢曾经向我传授过知识的边信黔教授、夏国清教授、严浙平教授和王元慧老师。感谢为我的毕业设计提供过无私帮助的陈子印师兄。感谢与我互相鼓励、并肩作战的同学们，是你们让我的学习生活更加丰富多彩。

特别需要感谢的是养育我多年、教育我成人的父母，他们给予了我享用不尽的支持和关怀，是我最坚实的后盾，最温暖的避风港。每次当我遇到困难时，他们都会鼓励我坚持，我也会不辜负他们的期望，昂首阔步迈向美好的前方！

荏苒之间，我的在校生涯已近终点。在哈尔滨工程大学学习的七年中，给我留下了无数美好的回忆。七年来，学校对我的教育为我将来的人生和职业生涯奠定了坚实的基础，无论是学术上还是人格上，都使我完成了一次又一次的蜕变与升华。感谢母校对我多年来的培育，今天我以母校而骄傲，明天必将使母校因我而自豪！