浙江大学

本科生毕业设计（论文）

中文题目：海上无人救助艇抛缆射云台仰角估计算法研究

英文题目：**Entry point forecast algorithm for life ring launch system based on unmanned rescue boat's position and altitude estimation**

姓名学号:

指导教师:

年级与专业:

所在学院:

2017 年 5 月 19 日

浙江大学本科生毕业设计（论文）诚信承诺书

1.本人郑重地承诺所呈交的毕业设计（论文），是在指导教师的指导下严格按照学校和学院有关规定完成的。

2.本人在毕业设计（论文）中引用他人的观点和参考资料均加以注释和说明。

3. 本人承诺在毕业设计（论文）选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。

4. 在毕业设计（论文）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

毕业论文（设计）作者签名：

2017 年 5 月 19 日

本科生毕业设计（论文）任务书

一、题目：海上无人救助艇抛缆射云台仰角估计算法研究

二、指导教师对毕业论文（设计）的进度安排及任务要求：

任务要求：

装备有抛射器的抛射云台是无人救助艇的主要救助系统。该毕业设计需融合来自通讯系统（上游环节）的母船指令信息以及各传感器关于海况和落海人员位置信息，利用计算机迅速求解出抛射云台的俯仰角度和回转角度，并将角度估计结果传给抛射云台的姿态控制系统（下游环节）。本次毕业设计要求做到以下几点：

1.查阅资料，确定切实有效的抛射云台仰角估计算法研究方案。

2.分别进行弹道计算模型、缆绳拖曳模型、空气动力模型的建模与仿真。

3.搭建实验平台，采集抛射器在各种俯仰角度下的真实弹道轨迹。

4.编写完整的计算代码，使用辅助计算软件完成大量关键点的计算。完成查表插值结果的修正公式研究，使其满足预期指标。

进度安排：

第一阶段（2016.11.15—2017.1.8）完成项目的总体方案设计；

第二阶段（2017.1.9—2017.3.15）完成绳索拖曳模型的建模与仿真。

第三阶段（2017.3.16—2017.4.1）搭建实验平台完成实地抛射实验。

第四阶段（2017.4.2—2017.4.30）完成考虑无人艇位姿和位姿变化的云台姿态求解模型和考虑风况的云台姿态求解模型建模。

第五阶段（2017.5.1—2017.6.5）撰写论文，准备答辩。

起讫日期 2016 年 11 月 15 日至 2017 年 6 月 5 日

指导教师（签名）职称

三、系或研究所审核意见:

负责人（签名）

年月日

毕业设计（论文）的进度安排

毕业设计（论文） 各阶段工作内容	工作进度周次安排																								导师签名 检查日期
	秋冬学期								春夏学期																
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
1.文献阅读、文献翻译																									
2.完成开题报告、文献综述																									
3.抛射器的选型和其内弹道计算模型建模																									
4.建立缆绳拖曳模型和编写对应计算程序																									
5.搭建实验平台，验证缆绳拖曳模型准确性																									
6.建立云台姿态求解模型																									
7.编写模型计算程序，对算例进行仿真																									
8.建立云台姿态求解模型（考虑风况）																									
9.编写云台姿态求解模型程序（考虑风况）																									
10.完成毕业论文																									

注：①本表由指导教师填写，各专业学科毕业设计负责人审核并签名。②各阶段工作内容应包括：查阅文献、文献综述、外文文献翻译、开题报告、调研、计算机仿真、设计绘图、实验、撰写毕业设计（论文）等，其中 4-9 栏目由指导教师布置、填写。③工作进度周次安排由指导教师在相应周次里画横线表示，（毕设 24 周，周次可以写为 3-4 周、8-9 周等）。④导师应在检查各阶段工作进度完成情况后签名，此计划可根据实际情况作调整

摘要

由于海上风浪时常达到八级以上，在开展救援工作时救援人员和搜救船往往难以靠近救援目标。近年来，无人艇愈加成熟的传感器系统、通讯系统和控制系统使得远程无人救援成为可能。通过装备配套的抛射器和抛射平台，基于无人艇的位姿（船体的纵倾和横倾）、抛射平台位姿、海况（风、浪、流）和抛射器的充气状况，可迅速预测射出的漂浮缆的入海点坐标范围。而如何求取准确的云台姿态并实现入海点始终处于救援目标附近，是本文的主要研究内容。

第 1 章，阐述了论文的研究背景与意义，并对比了国内外各相关领域的研究现状，针对存在的主要问题，提出了本文的研究内容和考核指标。

第 2 章，建立抛射器的缆绳拖曳模型。抛射器是一种广泛应用于海上救援、消防救援等领域的远距离抛绳的器具。在第 2 章中，本文针对其拥有无限自由度的特点，采用微元思想，推导连续性方程并生成拉力矩阵。通过将时间离散化的方式循环求取每个绳索单元的运动状态和受力状态。为保证该模型的准确性，将最终生成的弹道数据与实验进行了对比和误差分析。

第 3 章，建立云台姿态求解模型（未考虑风况）。首先通过多线程编程的方式生成弹道表，并利用救援目标坐标和无人艇坐标等上游信息进行查表求取对应的云台姿态参数（俯仰角度、俯仰转动线速度、回转角度、回转线速度），使其预测入海点始终满足考核指标。本文综合考虑了无人艇航速、运动姿态、运动姿态变化等无人艇系统状态信息，对查表结果进行修正。并给出算例和误差分析。

第 4 章，建立云台姿态求解模型（考虑风况）。在第 3 章所建立的云台姿态求解模型（未考虑风况）的基础上，融合了风速、风向等现场环境信息，对查表结果进行修正，求取对应的云台姿态参数并给出算例和误差分析。

在第 5 章中，对本文的研究内容进行总结，并对下一步的模型改进和研究工作进行展望。

关键词：抛射器、水面无人艇、弹道修正、云台姿态估计

Abstract

The sea state generally reaches more than eight level, so the rescue workers and rescue boats can’t get close to the rescued target. In recent years, the mature sensor system, communication system as well as the control system of unmanned surface vehicle(USV) gives possibilities to the remote unmanned rescues. Through the allocated catapult and cast platform, it is able to rapidly forecast the entry point coordinate range basing on the position and pose of USV (the trim and heel of hull), the cast platform position, sea conditions (wind, wave, current) as well as the air inflation of catapult. The main research content in this dissertation is seeking the accurate pan-tilt-zoom position to realize that the forecast entry point is always close to the rescue target.

In Chapter 1, the dissertation introduced the research background and significance, and overviewed the domestic and foreign present research situation. According to the main problems, the main research contents and the examination index of this dissertation were put forward.

In Chapter 2, the dissertation established the mooring rope towing model of Pneumatic Line thrower. Pneumatic Line thrower is an instrument for distant mooring rope, being widely applied in salvage, fire rescue and many other fields. In the Chapter 2, the dissertation adopted the infinitesimal method to deduce equation of continuity and generate tension matrix, aiming at its infinite degree of freedom. Through time discretization, it circularly gained the state of motion and stress state of each rope unit. In order to guarantee the accuracy of the model, it made the comparative analysis and tolerance analysis of the final generated trajectory data.

In Chapter 3, the dissertation set up the solving model of pan-tilt-zoom position (without considering the influence of the wind). Firstly, it generated the ballistic table by multithreaded programming. Then it utilized rescuing target coordinates, USV coordinates and other upstream information to search the table and find out the corresponding pan-tilt-zoom position parameter (elevation angle, elevation rotational line speed, rotation angle and rotary speed), making the predicted entry point always satisfy with the criteria of assessment. The dissertation comprehensively considered the USV’s speed, movement state, the changes of movement state

and other USV system status information to update computed parameters. Chapter 3 presented the calculating examples and error analysis, which illustrated that the accuracy and rapidity accorded with the assessment index of on-the-spot rescues.

In Chapter 4, the dissertation set up the solving model of pan-tilt-zoom position (considering the influence of the wind). Based on the solving model of pan-tilt-zoom position (without considering the influence of the wind) established in Chapter 3, the dissertation considered the wind speed, wind direction and other on-site condition information to update computed parameters. Chapter 4 presented the calculating examples and error analysis

In Chapter 5, it concluded the research contents of the dissertation and further outlook the next model rectification and research work.

Keywords: Pneumatic Line thrower, Unmanned Surface Vehicle, Trajectory correction, Pan-tilt's attitude estimation.

目录

浙江大学本科生毕业设计（论文）诚信承诺书.....1

本科生毕业设计（论文）任务书.....2

毕业设计（论文）的进度安排.....3

摘要.....4

Abstract.....5

目录.....6

第 1 章绪论.....10

1.1 研究背景与意义.....10

1.2 国内外研究现状.....11

1.2.1 国外研究现状.....11

1.2.2 国内研究现状.....13

1.3 目前存在的主要问题.....15

1.4 论文主要研究内容及结构安排.....16

第 2 章缆绳拖曳模型.....19

2.1 引言.....19

2.2 抛射器的缆绳拖曳模型.....19

2.2.1 炮筒内空气动力模型建立.....19

2.2.2 外弹道运动状态研究的基本假设.....22

2.2.3 抛射体头部的运动状态方程.....22

2.2.4 单绳段单元的运动状态方程.....24

2.2.5 求解连续性方程.....25

2.2.6 生成拉力矩阵方程.....26

2.3 代码实现.....27

2.3.1 编程语言和数据结构.....27

2.3.2 弹道计算代码实现.....29

2.4 实验验证和模型计算分析 30
2.4.1 实验平台搭建 30
2.4.2 实验结果和计算结果对比 31
2.4.3 缆绳拖曳模型的仿真计算分析 33
2.5 本章小结 36
第 3 章云台姿态求解模型（不考虑风况） 37
3.1 引言 37
3.2 利用 ParallelPython 生成弹道表 37
3.3 现场救援计算的坐标系统 39
3.3.1 无人艇坐标系的建立 39
3.3.2 描述无人艇运动姿态和状态的参数 41
3.3.3 无人艇和救援目标的坐标系变换 42
3.4 移动炮台对射程的影响 44
3.4.1 修正理论概述 44
3.4.2 移动炮口对初速度的修正 45
3.5 云台姿态求解模型（不考虑风况） 47
3.5.1 设定瞄准坐标系 48
3.5.2 求解其余未知参数 50
3.6 代码实现俯仰角度的求解（不考虑风况） 51
3.6.1 代码实现最小化代价函数的可行性分析（不考虑风况） 51
3.6.2 代码实现最小化代价函数（不考虑风况） 54
3.7 算例（不考虑风况） 55
3.8 本章小结 57
第 4 章云台姿态求解模型（考虑风况） 58
4.1 引言 58
4.2 对风的建模 58
4.3 对风的修正 59
4.3.1 横风产生的侧偏（相对运动法） 60

4.3.2 纵风对射程的修正 61
4.4 云台姿态求解模型（考虑风况） 62
4.4.1 瞄准坐标系下的风速 62
4.4.2 瞄准坐标系下风的影响 63
4.5 代码实现求取俯仰角度（考虑风况） 64
4.5.1 代码实现最小化代价函数的可行性分析（考虑风况） 64
4.5.2 代码实现最小化代价函数（考虑风况） 67
4.6 算例（考虑风况） 68
4.7 小结 70
第 5 章总结与展望 71
5.1 工作总结 71
5.2 研究展望 72
参考文献 73
附录（程序源代码） 75
致谢 89

第 1 章绪论

1.1 研究背景与意义

近年来，海上气候变化异常，极端自然灾害频发，强台风、强冷空气等频繁袭击我国沿海地区，导致水上重大突发安全事故频发，严重威胁国家和人民群众的生命安全。

恶劣海况下，引缆救助是在无法靠近遇险船舶的情况下保证人命及财产同时获救的最有效方法，约占船舶救助作业的 90%以上，该救援方法在我国海上救助作业中被广泛应用。传统的引缆方式是由救助人员手持抛射器站在船舶甲板上根据目测遇险船舶位置进行的，引缆成功与否很大程度上依赖于救助人员的经验。在恶劣海况下，救助工作人员自身难以保持平衡，无法稳定地把持抛射器进行引缆操作。加之恶劣海况下风力强劲、船体姿态变化快，会严重影响抛射方向、抛射距离、抛射精准度等指标，导致引缆失败率急剧上升。如何在恶劣海况下安全、快捷、有效地为遇险船只引缆成为当今救助打捞界公认的难题之一。针对传统抛射器存在的低准确性，亟待开发适用于恶劣海况的船载抛绳救援系统，提高引缆救助成功率，保障救助人员安全。

图 1.1 传统引缆救援示意图

由于海上风浪时常达到八级以上，在开展救援工作时救援人员和搜救船往往难以靠近遇难人员[1]。近年来无人艇的研究受到越来越多的关注，其愈加成熟的传感器系统、通讯系统和控制系统使得远程无人救援成为可能。通过装备配套的抛射器和抛射平台，基于无人艇的位姿（船体的纵倾和横倾）、抛射平台位姿、海况（风、浪、流）和抛射器的充气状况，可迅速预测射出的漂浮缆的入海点坐标范围。该课题旨在通过求解抛射云台补偿后的位姿，实现预测入海点始终在救援目标附近，减少预测入海点与救援目标

的偏差。

1.2 国内外研究现状

1.2.1 国外研究现状

（1）抛射器设计

在抛射器设计方面，挪威的 PLT（pneumatic Line Thrower）公司生产的抛射器在全球同类产品中属于领先地位。挪威 PLT 经过挪威船级社认证，满足 SOLAS 公约和国际海事组织公约 74/83,在国际市场上占有很大的份额。按射程和使用目的不同，可以分为 R230、R125、M75、M150 和 PWH 五种型号[2]。其用途广泛，可以用来发射缆绳、锚钩、自充气救生圈等。

R125 和 R230 主要用于船舶救援引缆，额定射程分别为 125m 和 230m；

M75 和 M150 主要用于船舶系泊引缆，额定射程分别为 75m 和 150m；

PWH 主要用于攀岩、勘测、自然灾害救援等。工作气压 7MPa，最大后坐力高达 5400N。绳索直径 3.2mm，最小拉力 2000N。弹体采用聚氯乙烯材质，以减小撞击受损。

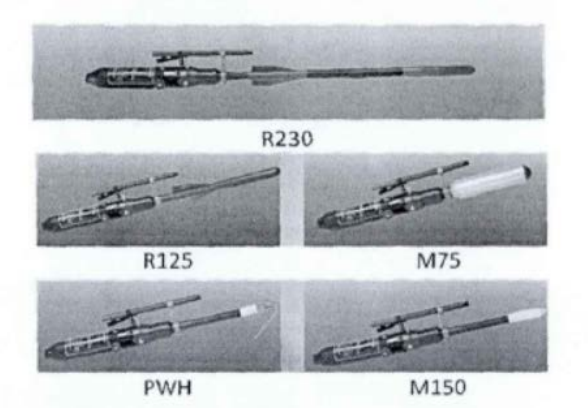


图 1.2 挪威 PLT 各型号

（2）绳索拖曳模型

目前拖缆系统方面较为成熟的为拖缆火箭系统。该技术能够在复杂地理与气象条件下和极端时间内，利用火箭（或某带有初速度的物体）将绳索快速而精确的送达目的地，形成大跨度的海上或空中救援通道，满足应急救援人员要求。因此火箭抛绳技术与装备研制对提高灾害救援的效率，提升当前国家整体救援保障能力具有十分重要的意义。抛绳飞行体系统的飞行动力学理论是该问题研究的难点和关键。 参与飞行的绳索为变

质量柔性系统，它具有无限自由度，动力学特性较复杂，目前，国外对于火箭拖带软绳索飞行的复杂动力学问题的研究较少。

在 Paul williams 等人的论文中[3]，介绍了关于采用微元思想，对绳索进行建模，建立了抛绳火箭飞行动力学模型（如图 1.3），并对其进行动力学特性分析。不考虑绳索的轴向伸长和弯曲，将绳索分为足够小的绳段，质量集中于其中心点[14]，段与段之间通过铰链连接。可分析得飞行体拖绳系统上的主动力由整体重力、空气阻力、末端绳子受力组成。由于柔软织物具有透气性，所以难以准确计算绳索所受气动力，故采用与火箭所受空气阻力相似的方法，得每段绳段所受空气阻力为$F_{dix} = C_i q_{ix} S_i (i = 1,2 \cdots, \mathrm{n})$,其中$C_i$是第 i 个绳段的阻力系数，$S_i$是第 i 个绳段的特征面积。根据 Kane 法，将各个方向上的力用矩阵表示，将所有作用力等效为广义主动力F_{Lj}和广义惯性力F_{Lj}^*，得到$F_L + F_L^* = 0$（Kane 方程），式中有 n+2 个方程和 n+2 个变量，可解算得飞行体和各绳段的运动学参数。

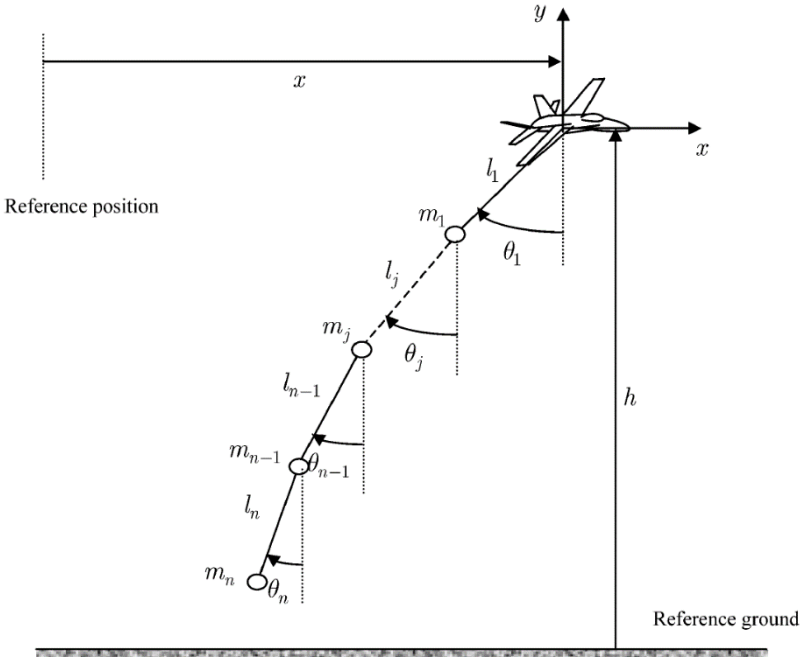


图 1.3 采用微元思想分析缆绳拖曳模型

Nakagawa 和 Obata 使用一种缆绳振动假想模型来分析拖缆空中系统的稳定性[4]。在假设中，缆绳是可活动的但不可伸展。只考虑了两个维度上的运动，数值计算表明拖曳系统在拖曳物体较小的情况将在两个维度上进行震荡（如图 1.4）。他们在处理该模型时是采用线性化的运动方程，其拥有有限个自由度。采用拉格朗日等式的解法，有一定的误差。Etkin 同时也在研究拖曳飞行器系统的稳定性[5]，他在建模的时候将缆绳看作

可伸展的。他的研究表明受到空气阻力的缆绳系统不是不稳定的，而那些提供上升力的物体反而使得整个系统变得不稳定。通过缆绳上关键点的适当配重可减小这种不稳定性。

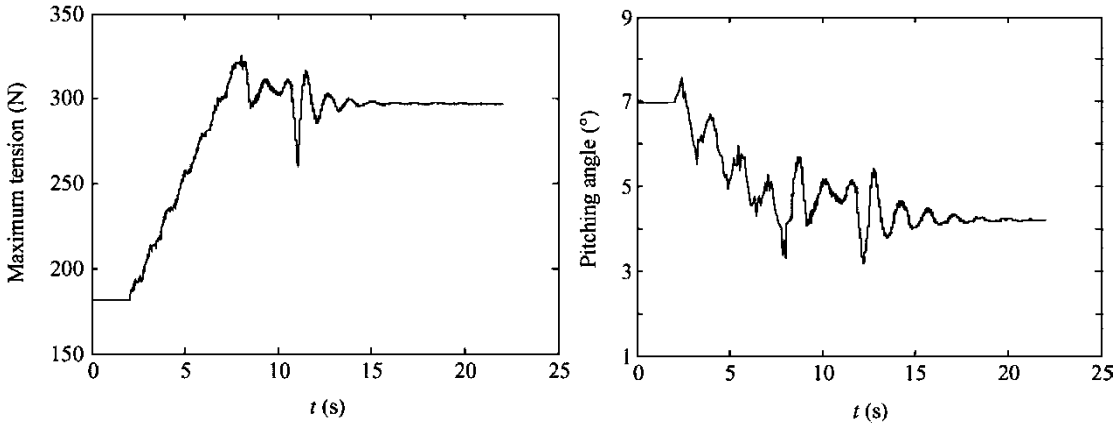


图 1.4 左图：缆绳最大压力随着时间的变化　右图：飞行头部的速度方向随时间变化

1.2.2 国内研究现状

（1）抛射器设计

厦门某公司生产的 SQS-230 远距离救生抛投器是目前国内救助作业中的常见型号，该型抛射器与挪威 PLT 抛射器构造类似，属于弹射式气动抛射器。其额定工作压力为 70bar，弹体抛射初速度较大，最大抛射后坐力 5400 N，水用时最大抛射距离可达 150 m。因为弹体带有尾翼，并且具有流线型弹体外形，具有较好的飞行稳定性和抗风能力，且缆绳内置于弹体内，防止了缆绳打结而减小发射距离[6]。

图 1.5 国产 SQS-230 远距离救生抛投器

目前，大连海事大学在新型抛射器装备研发方面开展了大量研究工作，对抛射器进行全新的结构设计，与宁波星箭航天机械有限公司通力合作，于 2011 年和 2013 年分别成功研发了两代性能优良的新型弹射式气动抛射器，其研发的新型气动抛射器集抛射器、气控箱和抛射架于一体，对控制阀体进行了创新性设计，将机械式手动击发改进为气动击发或电磁控制击发，对弹体进行了减阻优化设计，提出了弹簧和缓冲器两级缓冲的后坐力缓冲方案，大量抛射试验结果表明其设计的抛射器具有优良性能[7]。

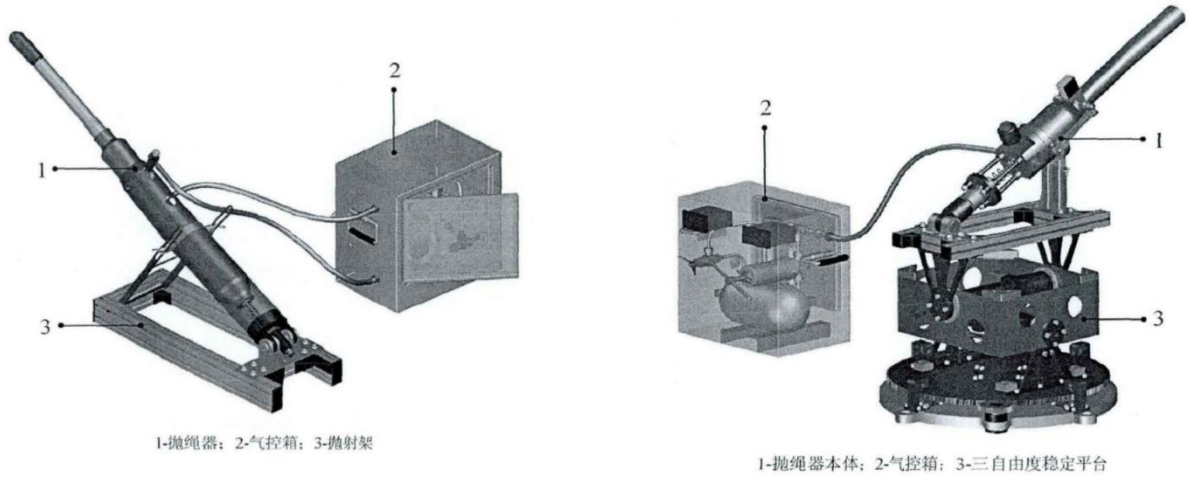


图 1.6 左图为大连海事大学一代抛射器，右图为大连海事大学二代抛射器

（2）内弹道的数值模拟研究

抛射器作为一种特殊的发射装置，其在腔内的发射过程与气体炮内极其相似。两者都是先向气室充气至所要求的初始压力。发射时，启动控制阀，气室内压缩气流流经控制阀推动弹丸向前运动，在炮筒内不断加速抛射体至炮筒口，从而抛射体获得炮口速度，完成内弹道发射过程。

国内在气动发射器弹道的数值模拟方面也已经建立起很多的理论计算方法。陈大年使用准一维不定常的 lagrangian 算法[8]，对许多模拟中出现的问题进行了讨论并解决，从而为气体发射器的设计参数和工作参数提供了数值理论参考；吴应湘运用一维非定常流的特征线理论对气体炮进行了模拟[9]，在模拟中考虑了真实气体中摩擦和传热损失的影响，详细描述了储气室、压缩管和发射管中的气流以及活塞和模型的运动规律；管小荣，徐诚运在研究结构参数和装填条件对气体炮发射性能的影响分别使用了龙格—库达法和二阶 YVD 格式对气体炮中发射部分的内弹道过程进行了模拟；杨均匀等人使用了 Von Neumann Richtmyer 差分格式模拟了气体炮内弹道，其结果十分符合实验测得参数

[10]。上述学者都是通过数值模拟的方法从理论上详细描述了气体炮内弹道，对各种可能影响炮弹速度的因素进行了研究，并对发射过程进行仿真，从而提高了预测炮弹出口速度的精度。

（3）绳索拖曳模型

吴小平等人在计算拖缆火箭弹道时，考虑了飞行物体的形状，列出了飞行体绕质心的摆动方程[11]。并列出了每一绳段的运动方程，都受到了自身重力，空气阻力，前后端点的拉力，由于绳索只能承受拉力，得到补充方程，方程组封闭有解，将矢量方程化为标量形式进行数值解。由于其解唯一，对微分方程组用等步长四阶龙格-库塔方法求解，并用主元高斯消去法求解线性方程组，获得瞬间绳索拉力。不过其中所建立的数学模型基于假定和离散化处理还存在一些不足，给计算结果带来一定误差。

而在王海潮等人的论文中，将参照了文献中[12]的单元切向和法向流体阻力公式，以及一些工程经验公式[3]，将绳索所受的气动阻力分为切向和法向两部分。分别为

$$R_{ni} = -\frac{1}{2}\rho C_m l_i d_i v_{ni}^2 \tag{1.1}$$

$$R_{\tau i} = -\frac{1}{2}\rho C_{\tau i} l_\tau d_\tau v_{\tau i}^2$$

式中：ρ为绳索所处高度的空气密度；C_m和$C_{\tau i}$分别为第 i 个绳段单元法向和切向阻力系数；l 和 d 为长度和直径，v_{ni}和$v_{\tau i}$分别为绳索单元的法向速度和切向速度。该处理方法为工程常用空气阻力处理法[13]。

1.3 目前存在的主要问题

（1）计算程序运行速度慢

对于计算程序，其中著名的有 Van Dyke 混杂理论基础上编制的气动力计算程序（Fortran 语言编写），计算精度较好，但计算程序十分复杂，计算时间较长[14]；对于一些根据实验曲线进行差值计算的气动力计算程序，虽然程序简单，计算速度十分理想，但精度难以保证。而对于抛射体的轨迹预测问题，必须有一个计算速度快、精度也较好的气动力计算程序，否则难以应用于实际救援场合中。

（2）出口速度难以确定

恶劣海况下，船舶摇晃剧烈，抛射器的抛射角度难以保持预定值，抛射精度受影响严重，同时抛射器在抛射弹体时会产生一个巨大发射后坐力，对救援系统的稳定性带来不利的影响。由于减压阀输出压力无法调节，进而导致抛射器射程不能通过调节压力值来调控。

（3）空气阻力大

由于海上风浪巨大，弹体所受的空气阻力不可忽略。风向的变幻莫测使得外弹道的研究更加具有挑战性。侧向风将使得炮弹偏离原本的方向，并可能获得一个巨大的侧向加速度。正向风将不可避免的减缓炮弹的飞行速度，从而使整个抛体运动呈现非对称性的椭圆轨迹。

1.4 论文主要研究内容及结构安排

本文主要分为三部分研究内容。分别对应每一章所建立的模型。具体为：抛射器的缆绳拖曳模型、考虑移动炮口的云台姿态求解模型、同时考虑移动炮口和风的影响的云台姿态求解模型。

（1）抛射器的缆绳拖曳模型

基于抛射器的硬件参数和充气情况，可得到射出的漂浮缆的运动学方程和动力学方程。同时用拖曳缆绳模型和考虑漂浮缆和绳子形状的空气动力模型对上述方程进行校正，得到系列方程组。通过程序代码进行方程求解和数值计算，计算出用各参数表示的入海点坐标值的表达式。由于若是采用代入救援目标位置反解出云台抛射角度和回转角度的计算方法将会导致巨大的计算量，以及将遇到多解的问题。因此为了节省现场计算时间，采用事先建立对应表的方式。利用建好的空气动力模型编制一张俯仰角度与射程的对应表和一张水平坐标与云台回转角度的对应表，以便后期查表加快求解速度。

（2）云台姿态求解模型

在现场救援时采用搭载抛射器的抛缆云台进行自主抛缆作业。装配在无人艇上的云台如图 1.7 所示。需要实时的调整云台的俯仰角度和回转角度，使其预测入海点到救援目标的距离始终满足考核指标。

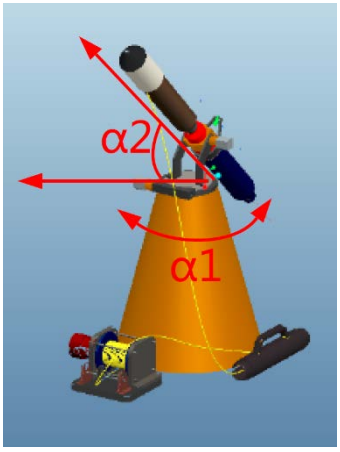


图 1.7 抛缆云台示意图

实际抛射条件不可能与标准条件[15]一致，会造成实际弹道与标准弹道不同。二者之间对应的弹道诸元就有了偏差。由于受到多种干扰因素的影响，例如无人艇的位姿（船体的纵倾和横倾）、抛射平台位姿、海况等变化所带来的影响，弹道出现偏差，造成预测入海点与救援目标之间有误差距离，因此需要在查表插值得到的数据的基础上进行修正。

表 1.1 炮弹抛射标准条件

标准气象条件	标准射表条件	标准地形、地球条件
（1）地面气温 $t_{0N}=15$ ℃，空气湿度 $(\rho_e)_{0N}=8.47$ hPa，虚温 $\tau(y)$ 遵守随高度变化的标准定律； （2）地面气压 $p_{0N}=1000$ hPa，气压随高度的变化遵守标准定律； （3）无风雨	（1）初速等于表定初速； （2）弹重符合图纸规定； （3）装药量符合标准，药温 $t_{0N}=15$ ℃； （4）火炮静止；	（1）弹着点在炮口水平面上； （2）地表为平面； （3）重力加速度 $g=9.8$ m/s^2，方向铅直向下； （4）科氏加速度为零

最终建立的云台姿态求解模型需要达到指标如表 1.2 所示。

表 1.2 云台姿态求解模型的考核指标

考核指标	
抛射距离	10~30m
入海点预测精度	相对误差小于 15%
计算时间	小于 0.1s

（3）结构安排

第一章，介绍了本文的研究背景和各国对于抛射器的研究现状，提出了本文的研究目标和考核指标。

第二章，建立了有多绳索单元参与飞行的缆绳拖曳模型，并搭建实验平台，对实验和仿真结果进行对比分析。

第三章，建立了考虑无人艇的位姿（船体的纵倾和横倾）和位姿变化的云台姿态求解模型，可即时输出云台的俯仰角度、俯仰转动线速度、回转角度、回转线速度，实现抛射器的预测入海点始终满足考核指标。

第四章，考虑风对射程的影响，进行俯仰角度的修正。建立考虑风况的云台姿态求解模型，并进行算例的误差分析。

第五章，总结了全文研究内容，并对文中存在的问题和进一步的研究方向进行了探讨和展望。

第 2 章缆绳拖曳模型

2.1 引言

由于抛射器中抛射体在飞行时受到参与飞行的绳索对其的拉力，因此我们需建立对应的缆绳拖曳模型来研究其射程与抛射条件的关系。这个模型是第 3 章和第 4 章研究内容的基础，所有有关补偿、修正、查表求值的方法都需要以缆绳拖曳模型为基准。本章将会研究抛射器的内弹道模型、外弹道模型，并进行实验以验证所建立模型的准确性，并对仿真计算结果和实验结果进行对比分析。

2.2 抛射器的缆绳拖曳模型

在海上救援作业时，抛缆飞行体系统工作过程是：飞行体与高强度救援缆绳一端连接，绳索被整齐地码在储绳箱内；控制阀打开时，飞行体在气体压力的推动下沿着炮筒向空中飞出，牵引着救援缆绳飞向瞄准点。参与飞行的绳索为变质量柔性系统，它具有无限自由度，动力学特性较复杂。绳索较长。因此我们在建立对应的计算模型时，拉力对抛射体运动的影响不可忽略。

2.2.1 炮筒内空气动力模型建立

（1）工作原理

抛射器作为一种特殊的发射装置，其在腔内的发射过程与气体炮内极其相似。如图 2.1 所示，两者都是先向气室充气至所要求的初始压力P_0，发射时，启动控制阀，气室内压缩气流流经控制阀推动弹丸向前运动，在炮筒内不断加速抛射体至炮筒口，从而抛射体获得炮口速度V_0，完成内弹道发射过程。

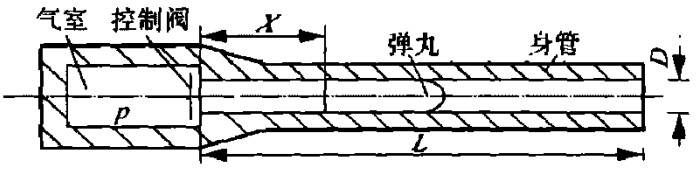


图 2.1　气动缆索抛射器系统内部结构

需要我们注意的是：由于减压阀输出压力无法调节，导致其不能根据实际需要来设定工作气压值，进而在现场救援时抛射器射程不能通过调节压力值来调控。

（2）基本假设

1.在海上现场进行救援时，抛射体在压缩气体作用下向前运动。由于气室容积的改变，空间会存在着压力分布，产生了复杂的气体动力学过程，使得气室内的压力分布规律比较复杂。实际上为了简化模型，施加在抛射体底部的空气压强常用平均压强 P 来表示。因此我们假设抛射体所受的空气压强即为气室平均压强 P。

2.可以在随后 2.4 小节实验中观察得到：在抛射体发射过程中，气室内部与外部系统几乎没有热量的交换，气室此时可看作一种封闭体系，并且由于发射过程时间极短，因此我们可以假设整个内弹道的发射过程是绝热过程，可使用描述绝热系统的气体方程来研究平均压强 P 的变化和压缩气体对抛射体的加速。

3.由于在发射过程中有较多能量损失，且其大小在计算抛射体初速度时不可忽略，而在工程上计算能量损失常用的方法为计算次要功系数φ，其有较为通用的计算式即式子(2.6)，可方便我们进行建模推导抛射体初速度的表达式。因此我们需要引入次要功计算因数φ来考虑摩擦消耗和其他能量的损失。次要功包括抛射体旋转运动所具有的能量、抛射体克服腔体内的摩擦阻力所消耗能量、气体的运动能量、后座部分的后座运动能量。

（3）建立方程

引入次要功计算因数φ，设 m 为抛射体质量，A 为气筒内室的横截面积，P 为弹射后气体瞬间压力，v 为抛射体运动速度。由于抛射体在气室内做变加速运动，需对瞬时变加速度进行积分才可得到抛射出口初速度。根据牛顿第二定律，我们可列出建立抛射体的内弹道运动方程为：

$$\varphi m\frac{dv}{dt}=AP \tag{2.1}$$

在式子（2.1）中，唯一的未知量为 P。根据抛射器内弹道的工作原理，在绝热过程下，我们设P_0为气体初始压力，W_0为气室初始容积，x 为抛射体的弹道运动行程，γ为气体绝热指数。则可得抛射器气室内的充气气体的状态方程为：

$$P_0W_0^r=P(W_0+Ax)^r \tag{2.2}$$

因此 P 可用式子（2.2）化简后来表示。联立式子（2.1）和式子（2.2），并将$v=\frac{dx}{dt}$

代入进行积分计算得到下列等式：

$$\int_0^v v dv = \frac{AP_0W_0^r}{\varphi m}\int_0^L \frac{dx}{(W_0+Ax)^r} \tag{2.3}$$

设V_0为抛射体的出口速度，L 是抛射体的内弹道行程总长度。从式子（2.3）计算积分我们可以得到抛射体的出口初速度表达式为：

$$V_0^2 = \frac{2AP_0W_0^r}{\varphi m}\frac{1}{A(1-r)}[(W_0+AL)^{1-r} - W_0^{1-r}] \tag{2.4}$$

在内弹道研究中，常用符号$Y = \frac{AL}{W_0}$来表示容积膨胀系数。我们可以将容积膨胀系数代入式子（2.4）中，整理得到：

$$V_0^2 = \frac{2P_0W_0}{\varphi m(\gamma-1)}\left[1 - \frac{1}{(1+Y)^{\gamma-1}}\right] \tag{2.5}$$

该等式的等号右边仅剩φ为未知量。根据火炮内弹道学，我们可以得到次要功计算因数的计算公式为[16]：

$$\varphi = K + \frac{m_q}{3m} \tag{2.6}$$

其中 K 是跟做功效率有关的实验因数,一般工程应用时取$K = 1.2$。m 为抛射体质量。m_q为气体质量，需列出其表达式。设 R 为气体常数；T 为绝对温度。M 是气体的摩尔质量。可通过气体动力学得气体质量的表达式为：

$$m_q = \frac{p_0W_0}{RT}M \tag{2.7}$$

结合式子（2.6）和式子（2.7）我们可以得到次要功计算因数φ的表达式为：

$$\varphi = K + \frac{p_0W_0}{3RTm}M \tag{2.8}$$

将式子（2.8）代入式子（2.5），我们得到最终抛射体的出口初速度表达式为：

$$V_0^2 = \frac{6P_0W_0RT}{(3KRTm + p_0W_0M)(\gamma-1)}\left[1 - \frac{1}{(1+Y)^{\gamma-1}}\right] \tag{2.9}$$

（4）计算关键数据点

观察式子（2.10）得，抛射初速度与初始工作压力、气室初始容积、抛射体内弹道行程总长度、气筒内腔的横截面积、抛射体质量、气体介质等有关。在本章 2.4 小节中，实验阶段采用东台市水尚救生设备有限公司生产的气动救生抛射器。工作压力为 10MPa~20MPa，工作气腔容积 0.4L，抛射体质量为 1.5Kg，发射管长 0.4m，发射管管径为 0.03m。对于压缩空气，绝热系数γ=1.4，T=300K，$R = 8.3\mathrm{J/mol\cdot K}$，M=0.028Kg/mol。

由式子（2.10）我们可算出不同工作压力下的抛射初速度分布如图 2.2。

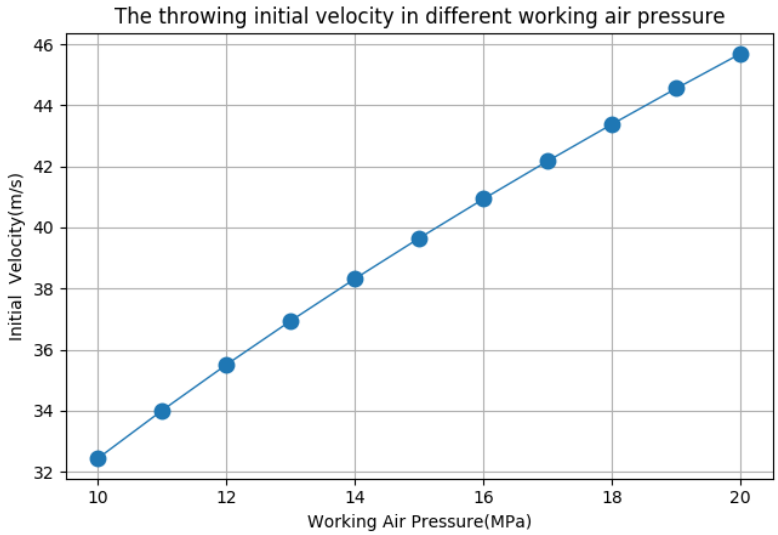


图 2.2 不同工作气压下的抛射初速度

2.2.2 外弹道运动状态研究的基本假设

以抛射体和绳段为研究对象，为研究方便提出以下假设：

（1）飞行系统沿射击面作平面运动，不考虑平面运动外的其他因素。对于平面之外的运动将在第 3 章所建立的瞄准坐标系下详细介绍。

（2）飞行系统中只考虑绳段和头部的空气阻力，而不考虑风的影响。对于风的影响，将在第 4 章中建立云台姿态求解模型（考虑风况）中具体对风进行建模和根据弹道理论进行修正。

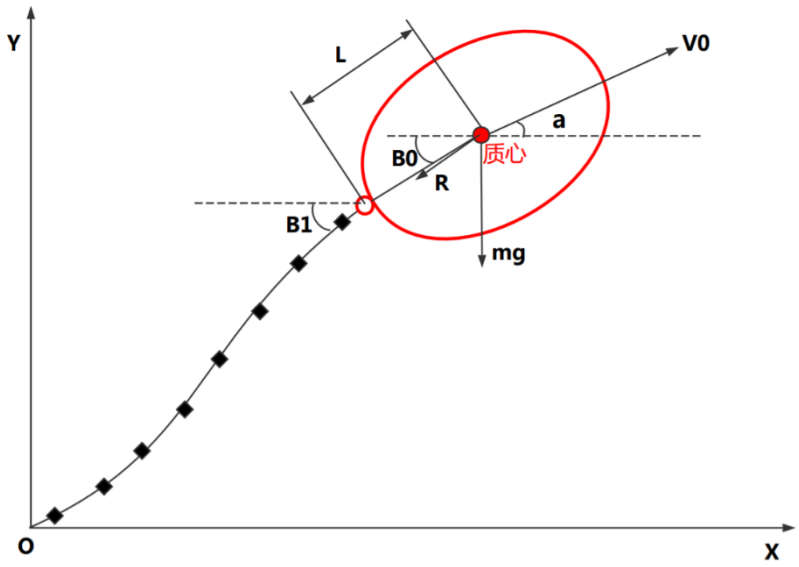
（3）在对各绳段单元进行研究时，为了方便研究，忽略各绳段的起始位置的微小偏差，假设飞行系统中的所有绳段从同一点（坐标原点）拉出。

（4）绳子为刚性绳，不可伸长。因此在建立方程的过程中可将每一个绳段看作是铰链进行分析。

2.2.3 抛射体头部的运动状态方程

以抛射体射出点位置为坐标原点，抛射体射向的水平方向为 X 轴，竖直方向为 Y 轴；参与飞行的绳索均匀离散为 n 个单元，从抛射体的连接端到刚离开地面端进行标

号，依次为 1,2,3，•••，n。每个单元的质量集中在远离抛射体的一端的节点上，并且他们之间的相互作用力和外力作用在这一系列节点上。前 n-1 个单元长度为极小值 l，第 n 个单元为变长度变质量单元。在单元不断被拉出的过程中，当最后一段的长度达到设



定条件时，长度不再变化，并拉起一个新的绳段 n+1，如图 2.3 所示。

图 2.3　缆绳拖曳模型的头部运动状态

其中：V_0是抛射体的质心速度；α表示速度方向与水平方向的夹角；β_0为抛射体轴线与水平方向的夹角；T_1、β_1为绳索第一个单元与抛射之间的作用力以及其方向与水平方向的夹角；L 是抛射体质心到抛射体末端的距离;J 是抛射体的转动惯量；ω_0是抛射体的角速度；s_0是抛射体的弹道弧长；R 是抛射体受到的空气阻力。

我们可在惯性坐标系下建立抛射体头部的运动方程：

$$\frac{dV_0}{dt} = \frac{-T_1\,cos(\beta_1 - \alpha) - mgsin\alpha - R}{m} \tag{2.10}$$

$$\frac{d\alpha}{dt} = \frac{-T_1\,sin(\beta_1 - \alpha) - mgcos}{mV_0} \tag{2.11}$$

$$\frac{d\omega_0}{dt} = \frac{T_1\,sin(\beta_1 - \beta_0) \times L_0}{J} \tag{2.12}$$

$$\frac{d\beta_0}{dt} = \omega_0 \tag{2.13}$$

$$\frac{ds_0}{dt} = V_0 \tag{2.14}$$

$$\frac{dx_0}{dt} = V_0 cos\alpha \tag{2.15}$$

$$\frac{dy_0}{dt} = V_0 sin\alpha \tag{2.16}$$

抛射体末端属于抛射体刚体上的一个点，由抛射体质心的运动方程以及末端与 O 点的几何关系，我们可以利用 Matlab 进行运算，得到末端运动方程：

$$\frac{dV_e}{dt}=\frac{\{-2JT_1\,cos(\alpha_0-\beta_1)+2L_0\omega_0^2m_0Jcos(\alpha_0-\beta_0)-L_0^2T_1m_0[cos(\alpha_0-\beta_1)-cos(\alpha_0-2\beta_0+\beta_1)]-2Jm_0gsin\alpha_0\}}{2m_0J} \tag{2.17}$$

$$\frac{d\alpha_e}{dt}=\frac{\{2JT_1\,sin(\alpha_0-\beta_1)-2L_0\omega_0^2m_0Jsin(\alpha_0-\beta_1)+L_0^2T_1m_0[sin(\alpha_0-\beta_1)-sin(\alpha_0-2\beta_0+\beta_1)]-2Jm_0gcos\alpha_0\}}{2m_0JV_e} \tag{2.18}$$

2.2.4 单缆段单元的运动状态方程

第 i 个绳段单元受到的绳段间的张力表现为前一个单元和后一个单元对它的拉力矢量和，即$\Delta T_i=T_i-T_{i-1}$,图 2.4 即为第 i 个绳段单元的受力情况。

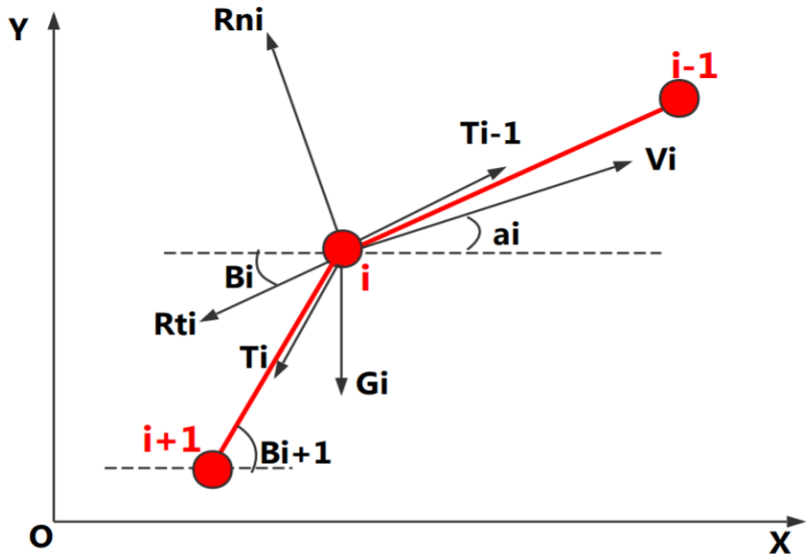


图 2.4 单个绳段单元的受力分析

我们可列出绳段单元的运动方程如下：

$$\frac{dx_i}{dt}=V_icos\alpha_i \tag{2.19}$$

$$\frac{dy_i}{dt}=V_isin\alpha_i \tag{2.20}$$

$$\frac{dV_i}{dt}=\frac{T_{i-1}\,cos(\beta_i-\alpha_i)-T_i\cos(\beta_{i+1}-\alpha_i)-R_{\tau i}\cos(\beta_i-\alpha_i)-R_{ni}\sin(\beta_i-\alpha_i)-m_igsin\alpha_i}{m_i} \tag{2.21}$$

$$\frac{d\alpha_i}{dt}=\frac{T_{i-1}\,sin(\beta_i-\alpha_i)-T_i\sin(\beta_{i+1}-\alpha_i)-R_{\tau i}\sin(\beta_i-\alpha_i)-R_{ni}\cos(\beta_i-\alpha_i)-m_i g cos\alpha_i}{m_i V_i} \tag{2.22}$$

$$\beta_i=arcta\,n\left(\frac{y_{i-1}-y_i}{x_{i-1}-x_i}\right) \tag{2.23}$$

其中：V_i是绳段单元质心的速度；α_i是绳段单元速度方向与水平方向的夹角；β_i是绳段单元轴线与水平方向的夹角；x_i、y_i是绳段单元的质心坐标；m_i是绳段单元的质量；T_i、T_{i-1}分别表示绳段后一个单元和前一个单元对第 i 个绳段单元的拉力。

在对每一个绳段单元进行受力分析时，需要考虑绳段所受的气动力。在所查找的文献中工程经验公式将绳段所受的气动阻力分为切向和法向两部分，他们分别是：

$$R_{ni}=-\frac{1}{2}\rho C_m l_i d_i v_{ni}^2 \tag{2.24}$$

$$R_{\tau i}=-\frac{1}{2}\rho C_{\tau i} l_\tau d_\tau v_{\tau i}^2 \tag{2.25}$$

式中：ρ为绳段所处高度的空气密度；C_m和$C_{\tau i}$分别为第 i 个绳段单元法向和切向阻力系数；l_i、d_i为长度和直径，v_{ni}和$v_{\tau i}$分别为绳段单元的法向速度和切向速度。该处理方法为工程常用空气阻力处理法。但我们需要大量的实验和经验公式对阻力系数进行校正[17]。

2.2.5 求解连续性方程

我们由基本假设的第四条可知，相邻的绳段单元在飞行过程中的距离应保持不变。以第 i-1 个绳段单元和第 i 个绳段单元为例，该两点间距离不变可表示为：

$$(x_{i-1}-x_i)^2+(y_{i-1}-y_i)^2=d_i \tag{2.26}$$

式中：d_i为第 i-1 个绳段单元和第 i 个绳段单元质点间的距离，在某一个弹道时刻可认为是常量。我们将该式对时间 t 进行二次求导，得到：

$$(x_{i-1}-x_i)\left(\frac{dV_{x,i-1}}{dt}-\frac{dV_{x,i}}{dt}\right)+(y_{i-1}-y_i)\left(\frac{dV_{y,i-1}}{dt}-\frac{dV_{y,i}}{dt}\right)+(V_{x,i-1}-V_{x,i})^2+$$

$$(V_{y,i-1}-V_{y,i})^2=0 \tag{2.27}$$

我们将之前的参数代入，进一步得到完整的连续性方程：

$$d_i cos\beta_i \left[\frac{dV_{i-1}}{dt} cos\alpha_{i-1} - V_i sin\alpha_{i-1} \frac{d\alpha_{i-1}}{dt} - \frac{dV_i}{dt} cos\alpha_i + V_i sin\alpha_i \frac{d\alpha_i}{dt}\right] +$$

$$d_i sin\beta_i \left[\frac{dV_{i-1}}{dt} sin\alpha_{i-1} - V_{i-1} cos\alpha_{i-1} \frac{d\alpha_{i-1}}{dt} - \frac{dV_i}{dt} sin\alpha_i + V_i cos\alpha_i \frac{d\alpha_i}{dt}\right] + (V_{i-1} cos\alpha_{i-1} -$$

$$V_i cos\alpha_i)^2 + (V_{i-1} sin\alpha_{i-1} - V_i sin\alpha_i)^2 = 0 \quad (2.28)$$

我们可以发现，该连续性方程把相邻两个绳段单元质点的运动状态和广义力联系在一起。随后可以将上述运动方程都代入该连续性方程，生成拉力矩阵。

2.2.6 生成拉力矩阵方程

我们将$\frac{dV_e}{dt}$、$\frac{d\alpha_e}{dt}$、$\frac{dV_i}{dt}$、$\frac{d\alpha_i}{dt}$代入连续性方程（未考虑气动力），可将连续性方程中的参数替换为采用运动状态参数来表示。可得到：

$$A_{11}T_1 + A_{12}T_2 = B_1 \quad (2.29)$$

$$A_{11} = -\frac{2m_0J + 2m_1J + m_1m_0L_0^2[1 - \cos(2\beta_0 - 2\beta_1)]}{2m_0m_1J} d_1 \quad (2.30)$$

$$A_{12} = \frac{d_1}{m_i} \cos(\beta_2 - \beta_1) \quad (2.31)$$

$$B_1 = -[V_0^2 + V_1^2 - 2V_0V_1 \cos(\alpha_1 - \alpha_0) + d_iL_0\omega_0^2 \cos(\beta_0 - \beta_1)] \quad (2.32)$$

我们将$\frac{dV_{i-1}}{dt}$、$\frac{d\alpha_{i-1}}{dt}$、$\frac{dV_i}{dt}$、$\frac{d\alpha_i}{dt}$代入连续性方程（未考虑气动力），得到：

$$A_{i,i-1}T_{i-1} + A_{i,i}T_i + A_{i,i+1}T_{i+1} = B_i \quad (2.33)$$

$$A_{i,i-1} = \frac{d_i}{m_i} co\,s(\beta_i - \beta_{i-1}) \quad (2.34)$$

$$A_{i,i} = -\frac{d_i}{m_i} \quad (2.35)$$

$$A_{i,i+1} = \frac{d_i}{m_i} co\,s(\beta_i - \beta_{i+1}) \quad (2.36)$$

$$B_i = \frac{d_i}{m_i}[V_{i-1}^2 + V_i^2 - 2V_{i-1}V_i\, cos(\alpha_{i-1} - \alpha_i)] \quad (2.37)$$

设某时刻有 n 个绳段参与飞行运动，可得到以下矩阵方程：

$$\begin{bmatrix} A_{11} & A_{12} & 0 & \cdots & 0 & 0 & 0 \\ A_{21} & A_{22} & A_{23} & 0 & \cdots & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & \cdots & 0 \\ \cdots & 0 & A_{i,i-1} & A_{i,i} & A_{i,i+1} & 0 & \cdots \\ 0 & \cdots & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & A_{n-1,n-2} & A_{n-1,n-1} & A_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & A_{n,,n-1} & A_{n,n} \end{bmatrix} \times \begin{bmatrix} T_1 \\ T_2 \\ \cdots \\ T_i \\ \cdots \\ T_{n-1} \\ T_n \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \cdots \\ B_i \\ \cdots \\ B_{n-1} \\ B_n \end{bmatrix} \tag{2.38}$$

查资料[18]可得，最后一个绳段（第 n 个绳段）为运动与静止界面的分界端，其所受拉力大小为

$$T_{n+1} = \rho V_n^2 \tag{2.39}$$

式中：ρ为第 n 个绳段单元的线密度。该拉力与第 n 个绳段单元的速度共线，方向相反。代入可求矩阵中的B_n。我们可以通过求解矩阵方程 AT=B，即可计算出各个绳段单元受到的拉力状态。

2.3 代码实现

为实现以上缆绳拖曳模型的计算与仿真，采用 Python 科学计算库来进行编程实现。而实现的目标主要有以下三点：

（1）高准确性：采用保留位数高达 11 位的 ndarray 数据结构进行编程计算；

（2）快速性：采用 C 语言为底层实现的 Numpy 库；

（3）可应用性：可在多种抛射器硬件条件下进行计算；

2.3.1 编程语言和数据结构

（1）编程语言选择

选用 Python 作为编程语言。Python 是一种解释型、面向对象、动态的高级程序设计语言，其有以下优点[19]：

- 可移植性——Python 程序无需修改就可在树莓派/Linux/Win 平台上面运行。
- 解释性——Python 语言写的程序不需要编译成二进制代码。可直接从源代码运行程序。
- 可嵌入性——可把 Python 嵌入其他的 C/C++程序。

（2）数据结构选择

选用 Python 的科学计算库 Numpy 下的 ndarray 作为主要数据结构。NumPy 的全名为 Numeric Python，是一个开源的 Python 科学计算库，它包括：

- 一个强大的 N 维数组对象 ndrray；
- 用于整合 C/C++和 Fortran 代码的工具包；
- 实用的线性代数、傅里叶变换和随机数生成函数
- 底层采用 C 语言实现

NumPy 最重要的一个特点就是其 N 维数组对象（即 ndarray）,该对象是一个快速而灵活的大数据集容器，该对象由两部分组成：实际的数据和描述这些数据的元数据。采用矢量化的线性代数计算与采用多个循环的常规计算相比，计算速度将提高至上百倍。

可将 ndarray 看作一种新的数据类型，如同 list、tuple、dict 一样，但需要注意的是 ndarray 中所有元素的类型必须是相同的，Python 支持的数据类型有整型、浮点型以及复数型，但这些类型不足以满足科学计算的需求，因此 NumPy 中添加了许多其他的数据类型，如 bool、int64、float32、complex64 等。本文采用的数据类型为 float64，有高达 11 位的计算精度。

表 2.1 float64 数据类型

Data Type	Description
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa 双精度浮点型

2.3.2 弹道计算代码实现

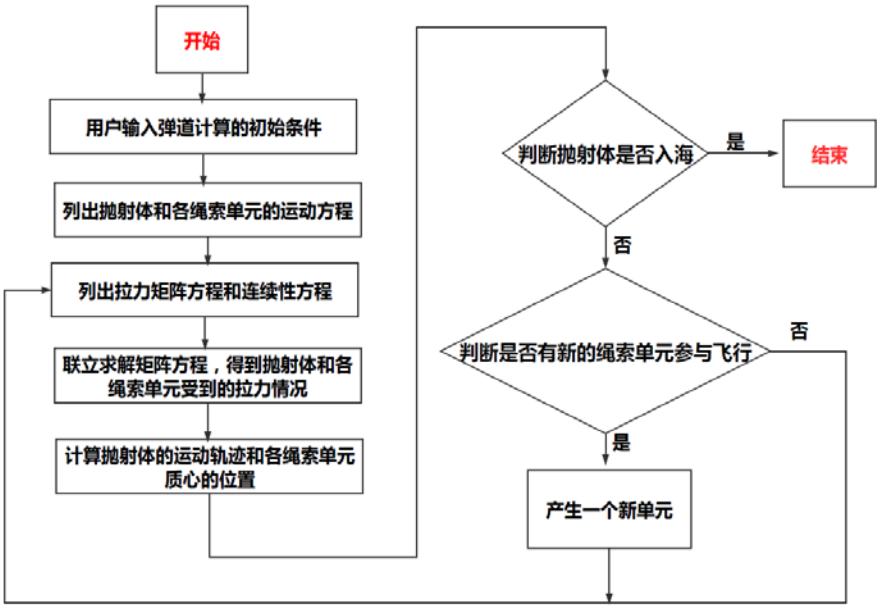


图 2.5 缆绳拖曳模型外弹道计算的代码框架

图 2.5 即为缆绳拖曳模型计算代码框架。用户输入弹道计算的初始条件，包括：抛射初速度$v_0(\mathrm{m/s})$、抛射俯仰角度$\alpha_0(^\circ)$、抛射体的转动惯量 J$(\mathrm{Kg}\cdot m^2)$、抛射体的质量$m_0(\mathrm{Kg})$、抛射体的质心到其末端的距离$L_0(m)$、缆绳的线密度ρ(Kg/m)、重力加速度g(m/s^2)。则程序将开始计算弹道轨迹。

随后程序进入一个 while 循环，退出循环的条件是抛射体的 Y 坐标小于 0，即抛射体入海。而在每次 while 循环的最后，对是否有新的绳段单元参与飞行进行判断，若有，则需要加入新的绳段单元。由于飞行时间大约为 2 秒以上，程序设定每个时间单元为 0.001 秒以保证计算准确度，这说明需要进行两千次以上的 while 循环。取每个绳段长为 0.05m，在飞行的中后阶段有将近 600~800 的绳段参与飞行。每次循环需要计算求解上百阶的矩阵，并且更新每个绳段单元的受力状态和运动状态，使得弹道计算十分复杂。

我们在求解矩阵的时候，采用 Numpy 构建的一个集成了多种数学算法的科学计算 Scipy 库中的 Scipy.linalg.solve(matA,matB) 函数。该函数在许多平台上都可以得到高度优化的版本，并提供非常好的性能，特别适用于大型稀疏矩阵。

表 2.2 solve 函数

Function	Description

scipy.linalg.solve(a,b,sym_pos=False)	Solve the equation ax = b for x.

2.4 实验验证和模型计算分析

为验证模型的正确性，本次毕业设计进行了实验。该小节将采用图表的方式对实验结果和仿真结果进行对比和分析。

2.4.1 实验平台搭建

本次实验采用东台市水尚救生设备有限公司生产的气动救生抛射器，装配有 6L 的充气气瓶（氮气）。在浙江大学紫金港校区内进行实验。

图 2.6 实验器材和实验现场

基本设备信息：

1.压缩空气工作压力：100bar~200bar

2.抛投质量：1.5Kg

3.抛投距离 30~100m

4.缆绳尺寸：Φ4mm×100m

5.测量装备：50m 卷尺、角度测量仪、秒表

6.充气气瓶气压：24Mpa

2.4.2 实验结果和计算结果对比

（1）实验测试

我们的实验测试结果如表 2.3。

表 2.3 抛射器实验测试结果数据记录

俯仰角度/°	测试压力/MPa	飞行时间/s	射程/m
30	10	1.44	21.2
45	10	2.01	29.2
60	10	2.35	36.5
75	10	2.85	45.5
60	10	2.40	35.0
45	10	1.90	30.2
30	10	1.38	21.4

（2）实验测试误差分析

1.由于采用手持式进行抛射，在发射的瞬间有巨大的后坐力，使得发射俯仰角度有误差。

2.第一次试验的时候由于绳子缠绕的原因，使得射程偏小。

3.时间的记录误差也比较大。由于飞行时间短，手动测时间带来的误差不可忽略。

（3）程序仿真计算结果

我们将设备信息输入轨迹计算程序中，得到结果在表 2.4 中。

表 2.4 缆绳拖曳模型的程序仿真计算结果数据记录

俯仰角度/°	发射初速度/m/s	飞行时间/s	射程/m
30	32	1.27	21.99
45	32	1.89	31.78
60	32	2.49	38.70

75	32	3.06	47.28

（4）对比仿真计算结果和实验结果

对比结果如下：

1.飞行距离的仿真计算结果和实验结果对比

图 2.7 为实验的飞行距离与计算的飞行距离对比（横轴为俯仰角度）。

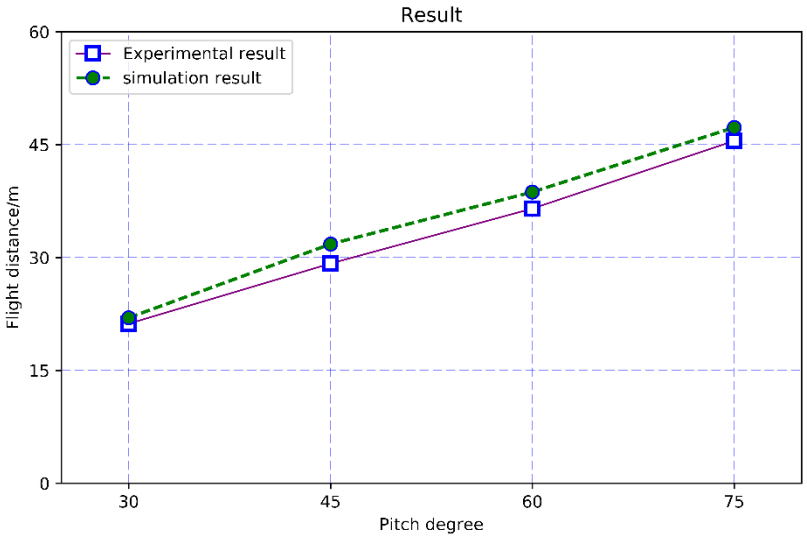


图 2.7　实验和仿真的飞行距离对比

平均相对误差为：5.62%

最大相对误差为：8.83%

2.飞行时间的仿真计算结果和实验结果对比

图 2.8 为实验的飞行时间与仿真计算的飞行时间对比（横轴为俯仰角度）。

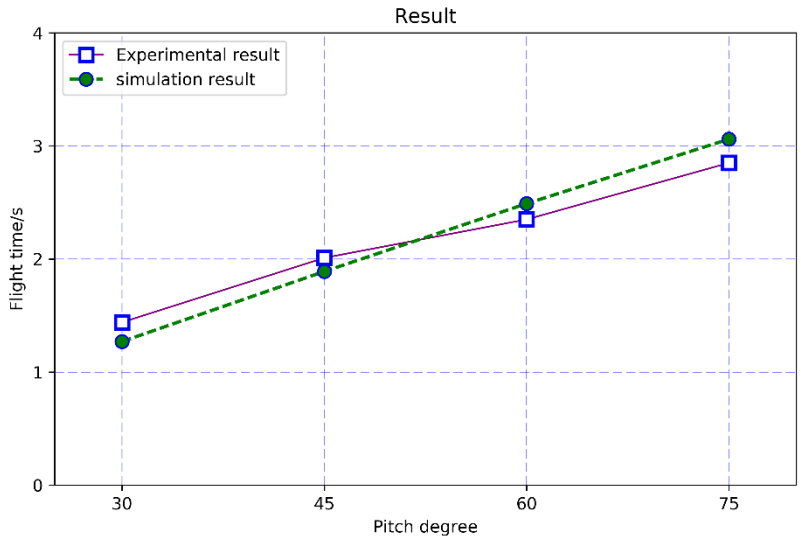


图 2.8　实验和仿真的飞行时间对比

平均相对误差为：7.78%

最大相对误差为：11.80%

通过以上各个图表的对比，我们可知仿真计算结果与实验结果非常接近，在各个俯仰角度下都有不错的准确性，说明建立的抛射器缆绳拖曳模型比较精确。

2.4.3 缆绳拖曳模型的仿真计算分析

该小节将利用图表的方式对所建立的缆绳拖曳模型进行仿真计算结果分析,并分析其是否和实际运动情况相符。

（1）飞行过程中缆绳的拉力变化分析

假设抛射器的硬件信息如 2.4.1 节相同，初速度为 30m/s， 俯仰角度为 45° 。得到抛射体头部所受的拉力变化如图 2.9。可知在发射的一瞬间抛射体头部承受到了巨大的阻碍飞行的缆绳拉力，随后逐渐波动下降，在飞行过程的中后期稳定在 T=200N 左右。

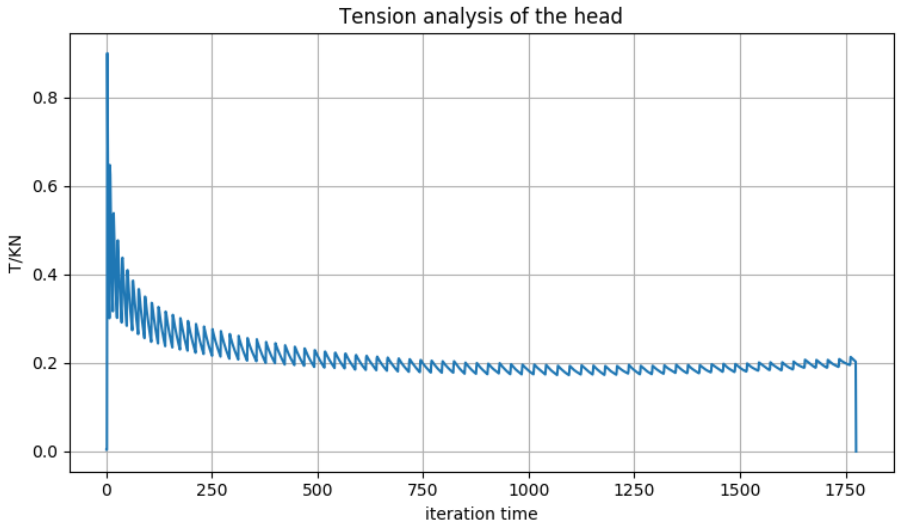


图 2.9　头部所受拉力的变化分析

（2）飞行过程中的角速度和角度变化分析

假设抛射器的硬件信息如 2.4.1 节相同，初速度为 30m/s， 俯仰角度为 30° 。在飞行过程中记录头部所受拉力的角度方向（angle of tension）、头部的角速度（angle velocity）、和头部的速度方向（angle of speed）。可以得到变化曲线如图 2.10。到速度的角度方向从正到负，绳子的拉力角度到了中后期也越来越小，因此角速度的幅值将趋于稳定，末期角速度的变化可以忽略。

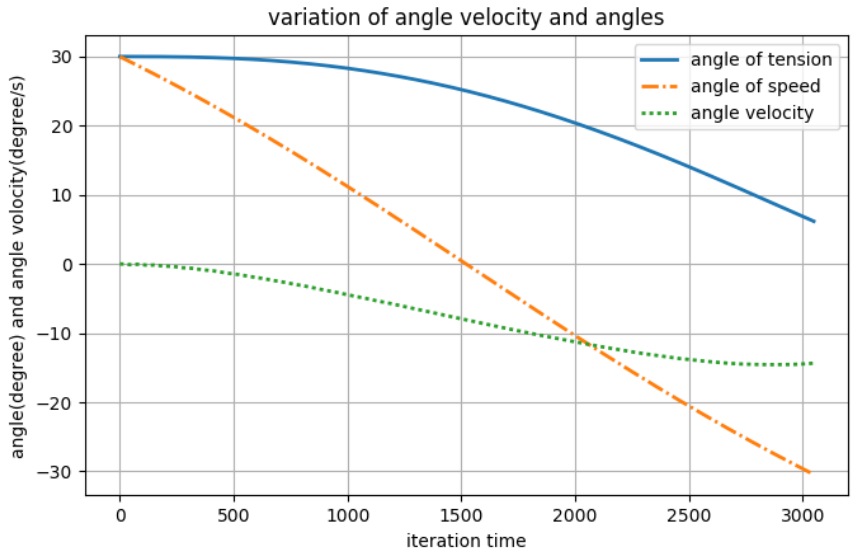


图 2.10　拉力角度、角速度、速度方向的变化

（3）抛射体的运动轨迹变化分析

假设抛射器的硬件信息如 2.4.1 节相同，初速度为 30m/s，轨迹随着俯仰角度的变化而变化，如图 2.11 所示。我们可知随着俯仰角度的增加，射程不断的增大。在海上救援现场时，可根据俯仰角度的调整来达到不同的射程。而关于海况、无人艇姿态变化所产生的初速度的影响，也需要通过俯仰角度的微调进行补偿。具体内容将在第三章、第四章进行阐述。

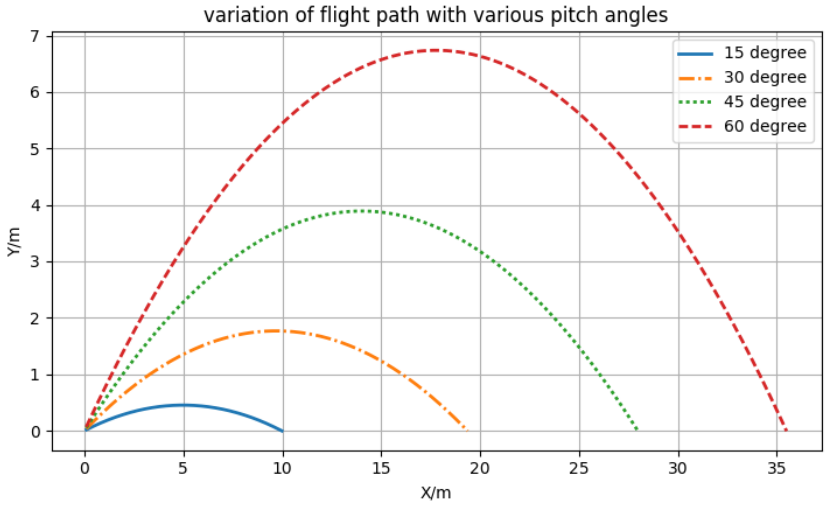


图 2.11　随俯仰角度变化的飞行轨迹分布图

假设抛射器的硬件信息如 2.4.1 节相同，俯仰角度为 30°，轨迹随着抛射初速度的变化而变化，如图 2.12 所示。我们可知随着抛射初速度的增加，射程不断的增大。抛射初速度是由充气状态决定的，因此在海上救援时，抛射初速度不可调。需在无人艇派出之前选择一个适当的抛射初速度进行充气准备，以保证能其射程变化范围能覆盖主要救援目标范围。

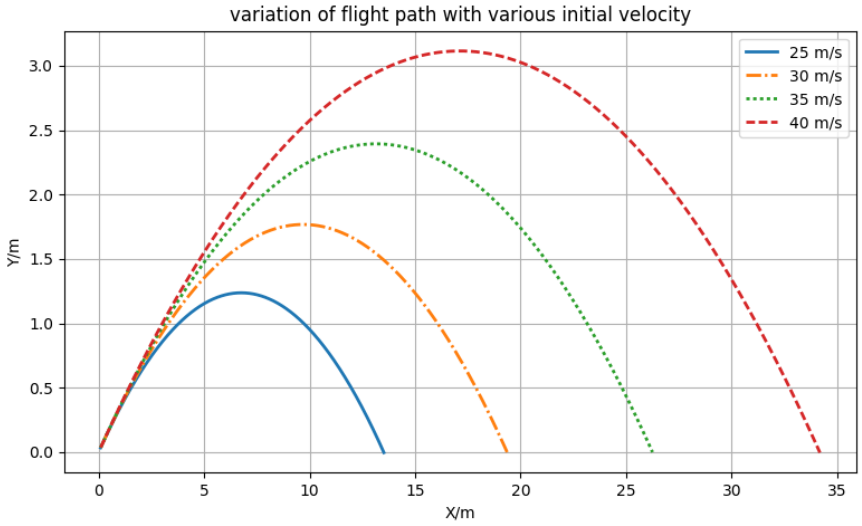


图 2.12 随抛射初速度变化的飞行轨迹图

2.5 本章小结

本章建立了抛射器的缆绳拖曳模型。本章首先对抛射体的运动情况做了基本假设，针对抛索运动中的变质量变结构条件动力学问题，将绳索离散为一系列质量集中在一点的单元，采用集中质量法建立了抛索运动方程，并用程序对其进行仿真数值计算。随后在现有的国产抛射器上进行了实验研究，通过对实验数据和仿真计算数据的对比，我们可得该缆绳拖曳模型在各个俯仰角度下都具有较高的准确度，是一种切实可行的理论模型。

第 3 章云台姿态求解模型（不考虑风况）

3.1 引言

在第 2 章中我们完成了对抛射器的缆绳拖曳模型的建模。在本章中，我们需要利用该模型生成一个弹道表，以供现场救援使用。本章研究目标是：在给定任意无人艇大地系坐标、救援目标大地系坐标、无人艇的速度与航向、无人艇的姿态、无人艇的姿态变化角速度、抛射体的初速度的情况下，需要在现场立刻求解出装配有抛射器的云台的回转角度、回转角速度、俯仰角度、俯仰角速度，以实现预测的抛射体入海点始终位于救援目标周围。

3.2 利用 ParallelPython 生成弹道表

在本小节中，我们采用 Python 的 PP 模块的多线程编程思想快速生成大型的弹道表。PP（ParallelPython）是一个 Python 模块，PP 模块通过两种方式实现了并行。第一种方式是在一个机器上有多 CPU 或多核心时，利用 SMP 架构。第二种方式是通过网络把任务分配到各个机器中，形成云计算模式。本文采用第一种方式，进程间的信息交换通过调用高度抽象函数实现，则编程人员不用担心管道和套接字的底层细节了。通过参数和函数就可以简单地实现交换信息。

用 PP 最大的优势是模块提供了高层的抽象。主要特性如下所示：

- 自动发现进程数量实现负载均衡
- 在运行阶段可以分配处理器
- 运行阶段可以负载均衡

在 PP 模块里有一个 Server 类，可用它来封装和发放本地与远程进程间的任务。在初始化时（__init__）有一些重要参数需注意，如下所示：

ncpus：这个参数允许编程人员设置 worker 进程的数量。若未设置，默认就会查看本机 CPU/核心数量，然后创建对应数量的 worker 进程执行任务。

ppservers：这个参数是一个包含机器名称或 IP 地址的元组，并行 Python 执行服务器（Parallel Python Execution Servers，PPES）。PPES 由网络中具有 ppservers.py 功能的

机器构成，运行并等待任务执行。

Server 类的实例有一个 submit 方法，可向目标机器发放任务。submit 函数签名如表 3.1 所示。

表 3.1 submit 函数签名

Function	Signature
Submit（self）	submit(self, func, args=(), depfuncs=(), modules=(), callback=None, callbackargs=(), group='default',globals=None)

submit 方法主要参数介绍如下：

func：本机 CPU 或远程服务器要执行的函数

args：func 函数的参数

modules：函数执行需要导入的远程代码或 func 函数执行需要导入的进程。例如，如果任务分配函数用了 time 模块，那么参数就要设置为 modules=('time',)

callback：这是后面要用的回调函数。当 func 参数的函数获取进程结果时，回调函数就是对结果进行处理。

图 3.1 即为采用 PP 模块进行编程的核心代码。只需设置好 inputs，程序将迅速调用主机所有 CPU 核进行多线程计算。

```
if len(sys.argv) > 1:
    ncpus = int(sys.argv[1])
    # Creates jobserver with ncpus workers
    job_server = pp.Server(ncpus, ppservers=ppservers)
else:
    # Creates jobserver with automatically detected number of workers
    job_server = pp.Server(ppservers=ppservers)
print "pp  可用工作核心线程数", job_server.get_ncpus(), "workers"
jobs = [(input, job_server.submit(into_function.computing,(input,))) for input in inputs]
for input, job in jobs:
    result = job()
job_server.print_stats()
```

图 3.1　多线程编程实现创建弹道表

我们分别采用单线程编程和多线程编程计算 20 条弹道轨迹，运行结果如图 3.2 所示，若运行主机 CPU 有 4 个线程，则采用 PP 模块可让平均计算时间减少为单线程耗时的四分之一。

```
[1.276, 27.681994563780719]
[1.302, 28.227466871092254]
单线程执行，总耗时 34.7160000801 s
pp 可以用的工作核心线程数 4 workers
<string>:1: SyntaxWarning: import * only allowed at module level
<string>:1: SyntaxWarning: import * only allowed at module level
<string>:1: SyntaxWarning: import * only allowed at module level
<string>:1: SyntaxWarning: import * only allowed at module level
多线程下执行耗时:  8.43099999428 s
Job execution statistics:
 job count | % of all jobs | job time sum | time per job | job server
        20 |        100.00 |      32.1470 |     1.607350 | local
Time elapsed since server creation 8.43099999428
0 active tasks, 4 cores
```

图 3.2 多线程编程与单线程编程耗时对比

3.3 现场救援计算的坐标系统

在该小节中，我们将无人艇大地系坐标、救援目标大地系坐标、无人艇的速度与航向、无人艇的姿态、无人艇的姿态变化角速度都转换成在无人艇的水平附体坐标系下的描述系数，以便进行后续分析。

3.3.1 无人艇坐标系的建立

为研究船舶运动，我们必须建立表达船舶运动的坐标系，描述船舶在海浪中的六自由度运动。必须同时采用以下三种坐标系：空间固定坐标系（惯性坐标系）、附体坐标系（运动坐标系）和水平附体坐标系（平面运动坐标系）。需将用空间固定坐标系描述的无人艇位置与救援目标位置、用附体坐标系表示的无人艇运动状态统一转换为在平面附体坐标系下，以备下一步的计算。

（1）空间固定坐标系$O x_0 y_0 z_0$：

该坐标系的原点 O 取在海面上的某固定点，x_0Oy_0面与静水面重合，Ox_0取在正北方向，Oy_0取在正东方向，Oz_0取指向地心。该坐标系可用来描述船舶的位置和姿态以及救援目标位置。

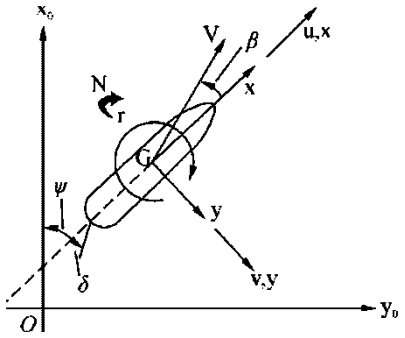


图 3.3 空间固定坐标系

（2）附体坐标系Gxyz:

Gxyz是原点位于船舶重心 G 上的附体坐标系，规定 Gx 轴指向船首，Gy 轴指向右舷，Gz 轴指向龙骨，该坐标系随船舶一起作摇荡运动，用来描述船舶运动。因为运动坐标系固定于船体上，随船体作任意形式的运动，所以它除了静止和在作匀速直线运动的情况外，都不能被认为是惯性系。该坐标系用于描述抛射器的初始抛射状态[20]。

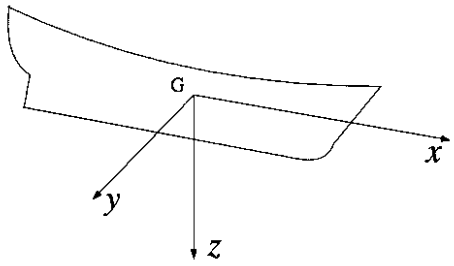


图 3.4 无人艇附体坐标系

（3）水平附体坐标系$Gx'y'z'$:

$Gx'y'z'$是原点位于船舶重心 G 上的水平附体坐标系，规定Gx'轴在水平面内指向船首，Gy'轴在水平面内指向右舷，Gz'轴指向龙骨，坐标面$Gx'y'$也始终与静水面重合，当船舶在波浪中作摇荡运动，该坐标系仅随船做纵荡、横荡、垂荡、首摇运动，而不作横摇和纵摇转动运动。该小节目的即将位于地面坐标系的船舶位置、救援目标位置和位于

附体坐标系的抛射器抛射状态转换为在该坐标系下进行研究。

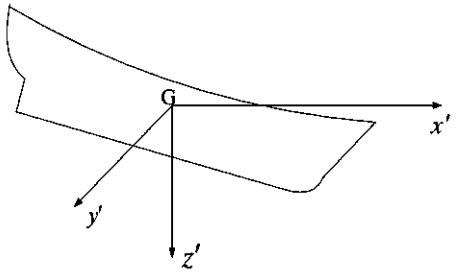


图 3.5 无人艇的水平附体坐标系

3.3.2 描述无人艇运动姿态和状态的参数

（1）描述船舶运动姿态的参数

无人艇在固定系$Ox_0y_0z_0$内的运动可以用它的位置和姿态来描述。其中位置指的是随动系原点 O 的三个空间坐标，分别为x_0、y_0、z_0。无人艇的姿态指的是它的三个欧拉角，它实际上确定了随动系Gxyz与固定系$Ox_0y_0z_0$之间的几何方位关系，分别以ψ、θ、ϕ表示，如图所示，ψ称为艏向角(azimuth)，θ称为纵倾角(pitching angle)，ϕ称为横倾角(rolling angle)[21]。

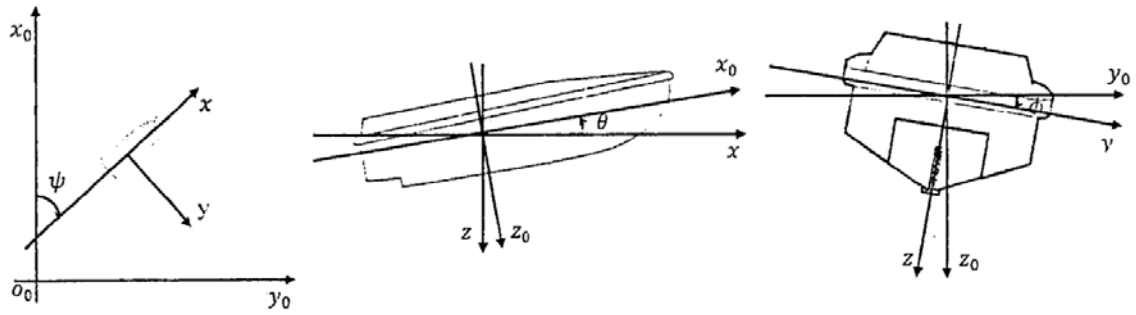


图 3.6 无人艇的欧拉姿态角

（2）描述船舶运动状态的参数

在空间固定坐标系中，船舶运动可用船舶的位置和姿态来描述，其中船舶的位置为附体坐标系原点 G 的空间坐标x_0、y_0、z_0，船舶姿态为船舶转动产生的三个欧拉角ψ、θ、ϕ。

在附体坐标系Gxyz中，规定 x 轴方向的前进速度为 u，y 轴方向横移速度为 v，z 轴方向垂荡速度为 w；绕 x 轴转动的横摇角速度为 p，绕 y 轴转动的纵摇角速度为 q，绕 z 轴转动的首摇角速度为 r。

在水平附体坐标系G$x'y'z'$中，x'轴方向的前进速度为 U，y'轴方向横移速度为 V，z'轴方向垂荡速度为 W；绕x'轴转动的横摇角速度为$\dot{\phi}$，绕y'轴转动的纵摇角速度为$\dot{\theta}$，绕z'轴转动的首摇角速度为$\dot{\psi}$。

3.3.3 无人艇和救援目标的坐标系变换

设在地面坐标系中，救援目标和无人艇此时的已知位置信息分别为（x_1，y_1），（x_2，y_2）

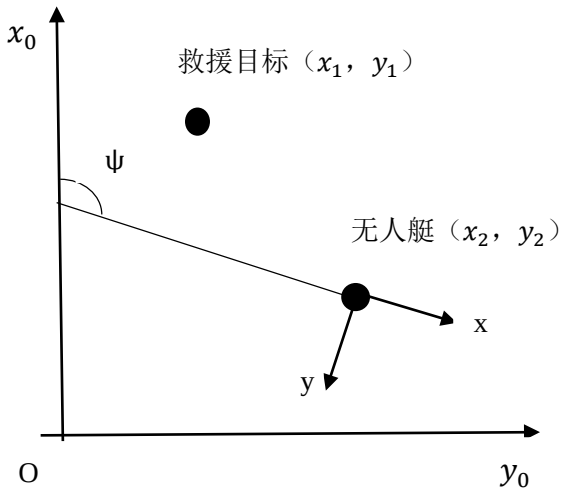


图 3.7 地面坐标系下救援目标与无人艇位置

(1)救援目标位置的坐标系变换

由无人艇指向救援目标的向量为$\vec{d}$（$x_1 - x_2$，$y_1 - y_2$），

ox 轴单位向量为$\vec{X}(cos\psi, sin\psi)$

oy 轴单位向量为$\vec{Y}(cos(\psi + \frac{\pi}{2}), sin(\psi + \frac{\pi}{2}))$=$(-sin\psi, cos\psi)$

我们可以得到救援目标在水平附体坐标系中的坐标为：

$$x'_{target} = \vec{d} \cdot \vec{X} = (x_1 - x_2) \times cos\psi + (y_1 - y_2) \times sin\psi \quad (3.1)$$

$$y'_{target} = \vec{d} \cdot \vec{Y} = (x_1 - x_2) \times (-sin\psi) + (y_1 - y_2) \times cos\psi$$

救援目标水平附体坐标系中的速度为：

$$v'_{target\_x} = -U \tag{3.2}$$
$$v'_{target\_y} = -V$$

而 U、V 等线速度量可由 u、v、w（已知信息）经过变换矩阵计算得到。

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} cos\theta & sin\phi sin\theta & cos\phi sin\theta \\ 0 & cos\phi & -sin\phi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{3.3}$$

则若抛射体的飞行时间为t，则可计算抛射体入海时救援目标在水平附体坐标系下的坐标为：

$$x'_{target\_t} = x'_{target} + v'_{target\_x} \times t \tag{3.4}$$
$$y'_{target\_t} = y'_{target} + v'_{target\_y} \times t$$

(2)抛射体初始状态的坐标系变换

由于无人艇的横摇、纵摇、首摇，抛射体将带有的初始角速度分别为 p、q、r（已知）。查资料可得，水平附体坐标系和附体坐标系之间的角速度运动量的欧拉角变换关系如下：

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & sin\phi tan\theta & cos\psi tan\theta \\ 0 & cos\phi & -sin\phi \\ 0 & sin\psi sec\theta & cos\phi sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3.5}$$

由此可得到抛射体在水平附体坐标系下的初始角速度。

为了后续内容论述方便，将上述表达式中欧拉角矩阵分别标记如下：

$$L_1(\Theta) = \begin{bmatrix} cos\theta & sin\phi sin\theta & cos\phi sin\theta \\ 0 & cos\phi & -sin\phi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{bmatrix} \tag{3.6}$$

$$L_2(\Theta) = \begin{bmatrix} 1 & sin\phi tan\theta & cos\psi tan\theta \\ 0 & cos\phi & -sin\phi \\ 0 & sin\psi sec\theta & cos\phi sec\theta \end{bmatrix} \tag{3.7}$$

（3）计算流程图

综合任意无人艇大地系坐标、救援目标大地系坐标、无人艇的速度与航向、无人艇的姿态、无人艇的姿态变化角速度等给定的输入条件，可通过向量计算和欧拉姿态角的变换矩阵等方式，得到在水平附体坐标系下的输出参数，即救援目标的最终坐标和无人艇的姿态变化角速度。

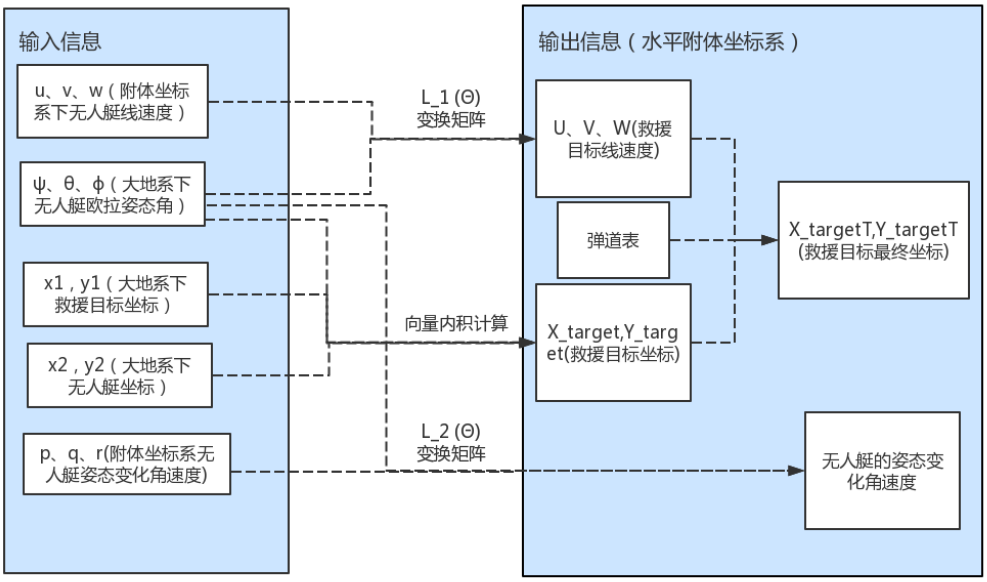


图 3.8 转换为水平附体坐标系的计算流程图

3.4 移动炮台对射程的影响

在抛射体准备对目标区域进行抛射时，由于运载抛射器的发射云台改变了抛射体出炮口时相对地面的初速度大小和方向，必须予以修正。实际上正是利用移动炮台初速度的作用来调整抛射器的入海点，使得抛射体能到达救援目标附近。

3.4.1 修正理论概述

（1）修正理论的实际意义

实际设计条件不可能与标准条件一致，因此造成实际弹道与标准弹道不同，二者对应的弹道诸元就有了偏差[22]。为了进行修正，就必须算出实际条件下弹道诸元与标准条件下弹道诸元之差，这个差值称为偏差。例如，对于距离X_0上的目标，按照弹道表基本诸元查出俯仰角度θ_0，它表示在标准条件下用θ_0进行抛射可以命中X_0处的目标；但在实际条件下用θ_0抛射时射程为 X，二者之间差值 $\Delta X = X - X_0$即为射程偏差。为了在实际条件下命中目标，则必须进行修正，如果 $\Delta X > 0$，这应将X_0减去 ΔX，按$X_1 = X_0 - \Delta X$去查弹道表重新赋予对应的俯仰角度θ_1。于是在实际条件下以俯仰角度θ_1抛射时的射程

即为$\mathrm{X} = X_1 + \Delta\mathrm{X} = X_0$。由此可见修正量和偏差量符号正好相反。

（2）修正的方法

计算修正量的方法有两种：求差法和微分法。

所谓求差法，是要建立考虑各种实际条件的非标准运动微分方程，分别用实际条件和标准条件去结算弹道诸元，然后求二者响应弹道诸元之差，即为修正量。

所谓微分法，就是将弹道诸元（如射程 X）看作各种影响因素的函数，即：

$$X = X(c, v_0, \theta_0, w_x, w_z \cdots) \tag{3.8}$$

我们将弹道诸元的偏差看作是由于这些因素的偏差引起的，并且当这些影响因素与标准值偏差不大时，可以认为他们对射程的影响是彼此独立的，影响的大小与影响因素偏差的大小成比例。于是，这些因素偏差对射程的总影响为各因素影响之和，即：

$$\Delta X = \frac{\partial X}{\partial v_0}\Delta v_0 + \frac{\partial X}{\partial \theta_0}\Delta\theta_0 + \frac{\partial X}{\partial w_x}w_x + \frac{\partial X}{\partial w_z}w_z + \cdots \tag{3.9}$$

式中，$\frac{\partial X}{\partial v_0}$、$\frac{\partial X}{\partial \theta_0}$等成为射程对某一因素的敏感因子或称为修正系数[23]。

在修正系数中，我们将对射程的修正系数$\frac{\partial X}{\partial v_0}$、$\frac{\partial X}{\partial \theta_0}$称为主要修正系数，其他因素偏差对射程的影响均可用主要修正系数表示。这些修正系数可用求差法获得，也可由弹道表用数值微分法求得。

由于微分法公式简单，并且当各影响因素偏离标准值不大时，修正量计算的准确性一般可以保证，故在炮兵实践中经常使用，因此也能应用于抛绳器的抛射。本文采用微分法进行弹道修正。

3.4.2 移动炮口对初速度的修正

在抛射体准备对目标区域进行抛射时，由于运载抛射器的发射云台改变了抛射体出炮口时相对地面的初速度大小和方向， 我们必须予以修正。

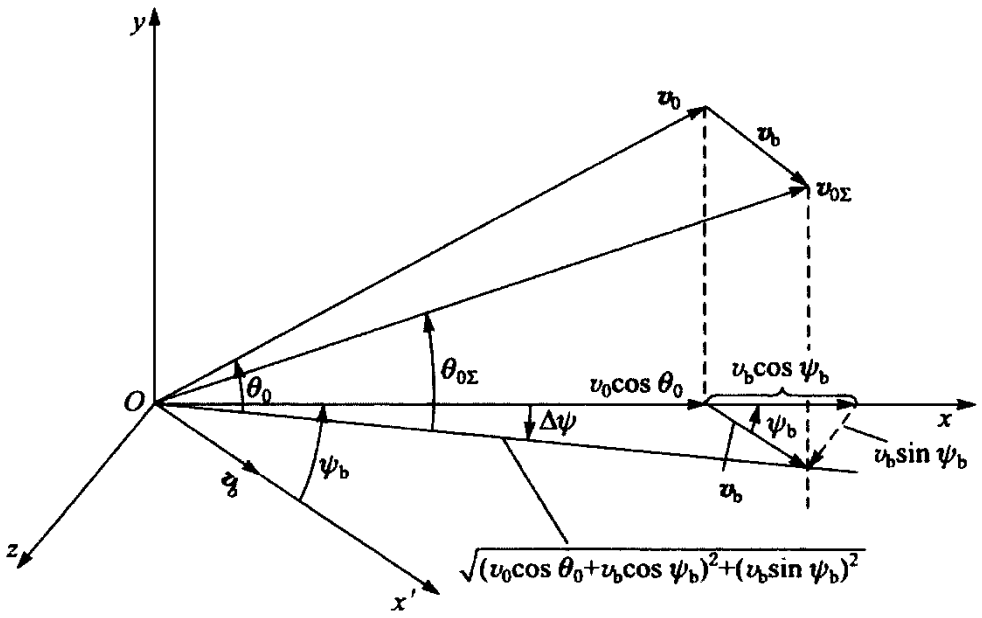


图 3.9 抛射炮口的移动对抛射初速度的影响

如图所示，设抛射器发射瞬时位置在 O 点，以速度v_b向Ox'方向前进，而目标在 Ox 方向上。Ox 与Ox'的夹角为ψ_b，如抛射体沿 Ox 方向以射角θ_0和初速v_0射击，则射出炮口时的合成速度矢量为$v_{0\Sigma}$，射角为$\theta_{0\Sigma}$，而由于抛射运动对射向的修正为 $\Delta\psi$

$$\Delta\psi = \arctan\frac{v_0 sin\psi_b}{v_0 cos\theta_0 + v_b cos\psi_b} \tag{3.10}$$

至于对射程的影响，将有初速和射角的改变来决定。由图我们可得到：

$$v_{0\Sigma} = \sqrt{(v_0 cos\theta_0 + v_b cos\psi_b)^2 + (v_0 sin\theta_0)^2 + (v_0 sin\psi_b)^2} \tag{3.11}$$

$$v_{0\Sigma} = \sqrt{v_0^2 + 2v_0 v_b cos\theta_0 cos\psi_b + v_b^2}$$

从根号中提出v_0并利用二项式定理展开在略去二次以上各项，得合速度$v_{0\Sigma}$以及初速增量 Δv_0

$$v_{0\Sigma} = v_0\left[1 + \frac{v_b cos\theta_0 cos\psi_b}{v_0}\right] \tag{3.12}$$

$$\Delta v_0 = v_b cos\theta_0 cos\psi_b \tag{3.13}$$

接下来求射角改变量。由于

$$\tan\theta_{0\Sigma} = \frac{v_0 sin\theta_0}{\sqrt{(v_0 cos\theta_0 + v_b cos\psi_b)^2 + (v_0 sin\psi_b)^2}} \approx tan\theta_0(1 - \frac{v_b cos\psi_b}{v_0 cos\theta_0})$$

利用二项式定理展开并略去二次和二次以上各项，再由

$$\Delta \tan\theta_0 = (tan\theta_0)' \times \Delta\theta_0 = \frac{\theta_0}{cos^2\theta_0} \tag{3.14}$$

$$\Delta\theta_0 = \frac{-v_b cos\psi_b sin\theta_0}{v_0} \tag{3.15}$$

再利用微分修正公式和敏感因子$\frac{\partial X}{\partial v_0}$、$\frac{\partial X}{\partial \theta_0}$，我们就可得到由于云台运动产生的射程修正量公式。在修正时，只需将云台运动速度向 Ox，Oy，Oz 三轴分解，再分别加到初速v_0的三个分量上去计算抛射体运动时的弹道即可。

3.5 云台姿态求解模型（不考虑风况）

在该小节中将阐述云台姿态求解模型（不考虑风况）的计算过程。我们先取值确定了回转角度并设定瞄准坐标系，将所有的运算都分解到射击面和垂直射击面的瞄准坐标系OY_a轴上。在射击面上，通过计算由于炮口移动影响下的射程增量，调整得到一个补偿后的俯仰角度和适当的俯仰转动线速度。随后查弹道表得到飞行时间，在OY_a轴方向上计算得到回转线速度。

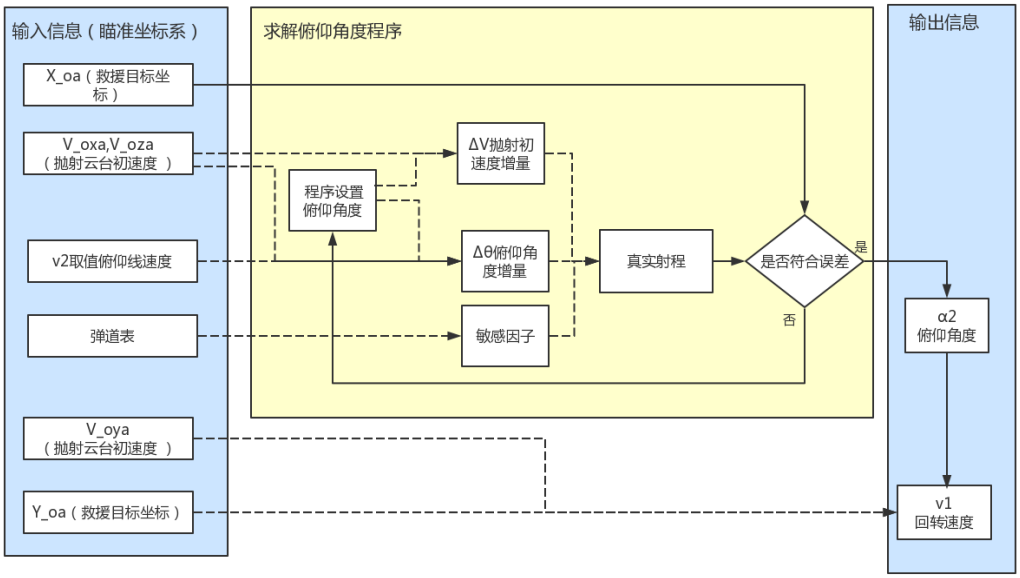


图 3.10 云台姿态求解模型（不考虑风况）的核心计算流程图

从而我们可以得到俯仰角度、俯仰线速度、回转角度、回转线速度四个所需的输出量，以便下游的云台控制环节进行角度调整。

3.5.1 设定瞄准坐标系

上游环节将无人艇大地系坐标、救援目标大地系坐标、无人艇的速度与航向、无人艇的姿态、无人艇的姿态变化角速度都传递过来，利用 3.2 节所推导的公式，将这些变量都转换为在水平附体坐标系下，得到x'_{target}。y'_{target}、U、V、$\dot{\phi}$、$\dot{\theta}$。由于$\dot{\psi}$比较小，与其他初速度相比可忽略。

（1）设定瞄准坐标系

我们可得到水平附体坐标系下救援目标的位置为x'_{target}。y'_{target}，根据平均飞行时间 t 秒计算，得到救援目标 t 秒后的坐标为x'_{target_t}，y'_{target_t}。当$y'_{target_t} > 0$时，救援目标落在水平附体坐标系的第四象限，设定此时方位角为负值-a（a为方位角的绝对值）。当$y'_{target_t} < 0$时，救援目标落在水平附体坐标系的第三象限，设定此时方位角为正值+a（a为方位角的绝对值）。由于变量太多，取回转角度α_1为方位角的 λ（0< λ <1）倍，该 λ 值可调整云台回转线速度，初始可设为 λ $=\frac{1}{2}$即：

$$\alpha_1 = -\lambda a 或 \alpha_1 = +\lambda a \tag{3.16}$$

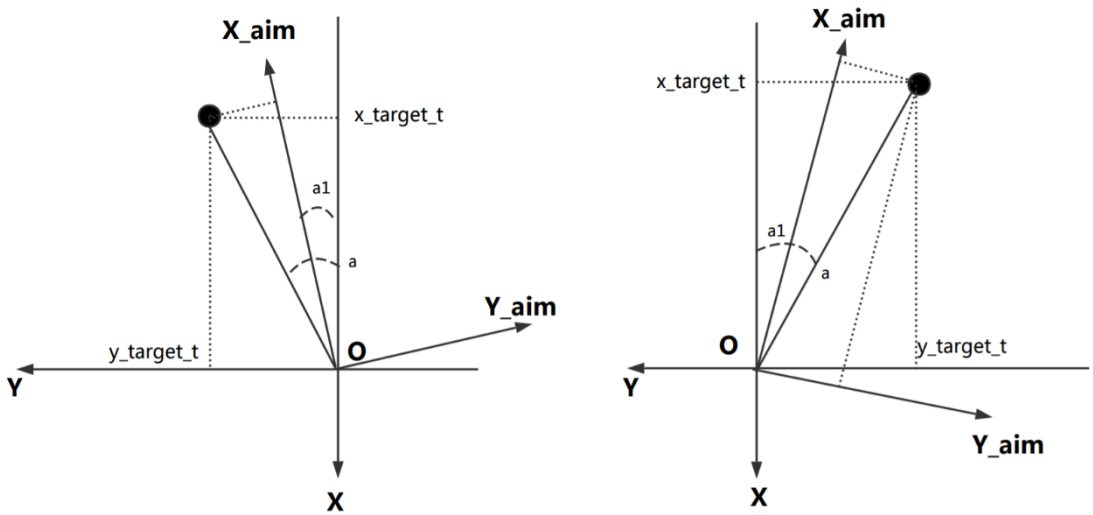


图 3.11 瞄准坐标系

我们建立瞄准坐标系$OX_{aim}Y_{aim}Z_{aim}$，其坐标原点 O 为抛射云台所在位置，OX_{aim}方向为沿α_1方向并与水平面平行(即将水平附体坐标系逆时针旋转角度$(\pi - \alpha_1)$)，$X_{aim}OY_{aim}$平面和水平面平行。OZ_{aim}方向与指向地心方向相反。可计算出救援目标在

瞄准坐标系中的坐标为（$x_a, y_a, 0$）,可得到以下如图所示，左图为方位角为负值时的救援目标位置，右图为方位角为正值时的救援目标位置。

（2）求解瞄准坐标系下的抛射体初速度（考虑无人艇姿态和姿态变化）

我们假设抛射器在水平附体坐标系下的坐标为（L_1, L_2），则该时刻抛射器受到的在水平附体坐标系下沿各轴方向的初速度为：

$$v_{0z} = L_2 \times \dot{\phi} \times \cos\phi + L_1 \times \dot{\theta} \times cos\theta \tag{3.17}$$

$$v_{0y} = L_2 \times \dot{\phi} \times \sin\phi$$

$$v_{0x} = L_1 \times \dot{\theta} \times sin\theta$$

我们需要将初速度转换到瞄准坐标系下，转换关系为：

$$\begin{bmatrix} v_{oxa} \\ v_{oya} \\ v_{oza} \end{bmatrix} = \begin{bmatrix} \cos(\pi - \alpha_1) & -\sin(\pi - \alpha_1) & 0 \\ sin(\pi - \alpha_1) & cos(\pi - \alpha_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{0x} \\ v_{0y} \\ v_{0z} \end{bmatrix} \tag{3.18}$$

设：

$$L_3(\Theta) = \begin{bmatrix} cos(\pi - \alpha_1) & -sin(\pi - \alpha_1) & 0 \\ sin(\pi - \alpha_1) & cos(\pi - \alpha_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

则救援目标的位置为：

$$\begin{bmatrix} x_{oa} \\ y_{oa} \\ z_{oa} \end{bmatrix} = L_3(\Theta) \begin{bmatrix} x'_{target\_t} \\ y'_{target\_t} \\ 0 \end{bmatrix} \tag{3.20}$$

我们整理以上计算步骤，得流程图如下：

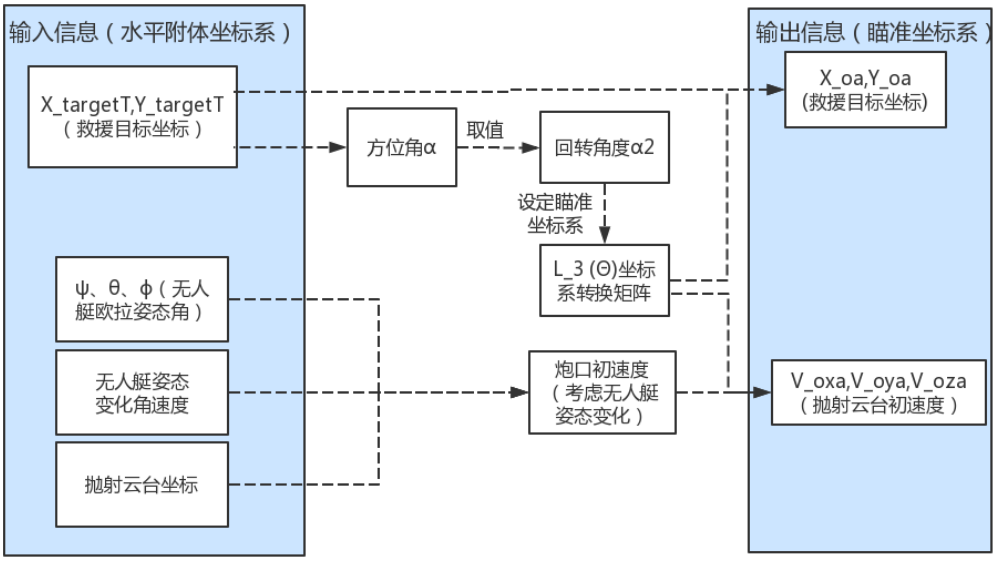


图 3.12 转换为瞄准坐标系的计算流程图

现取方位角为正值为例阐述下一步研究。

3.5.2 求解其余未知参数

（1）求解射程增量（考虑炮口移动速度）

理想状态下，我们需要抛射体在瞄准坐标系中沿着射击面（即$X_{aim}OZ_{aim}$面）飞行x_{oa}距离，需要沿着OY_{aim}方向飞行y_{oa}距离。

由于变量太多，设俯仰转动线速度为固定值v_2（方向从下向上），俯仰角度为α_2。我们根据式子(3.13)，在射击面（即$X_{aim}OZ_{aim}$面）上由于炮口的初速度的影响导致的初速增量为：

$$\Delta v_0 = v_{oxa}\cos(\alpha_2) + v_{oza}\sin(\alpha_2) \tag{3.21}$$

根据式子(3.15)，在射击面上由于炮口的初速度导致的射角增量为：

$$\Delta \alpha_2 = \Delta \alpha_2 = \frac{-v_{oxa}sin\alpha_2 + v_{oza}cos\alpha_2 + v_2}{v_0} \tag{3.22}$$

则此时的敏感因子为：$\frac{\partial X}{\partial v_0}|_{v_0=v_0}$、$\frac{\partial X}{\partial \alpha_2}|_{\alpha_2=\alpha_2}$，我们可通过查弹道表求得该偏微分值。

根据式子(3.9)，我们可得到在炮口初速度的影响下，抛射器的射程增量为：

$$\Delta X = \frac{\partial X}{\partial v_0}\Delta v_0 + \frac{\partial X}{\partial \alpha_2}\Delta \alpha_2 \tag{3.23}$$

代入式子(3.21)和式子(3.22)，得到：

$$\Delta X = \frac{\partial X}{\partial v_0}(v_{oxa}\cos\alpha_2 + v_{oza}\sin\alpha_2) + \frac{\partial X}{\partial \alpha_2}\left(\frac{-v_{oxa}sin\alpha_2 + v_{oza}cos\alpha_2 + v_2}{v_0}\right) \tag{3.24}$$

我们需要调整俯仰角度α_2，使得其可以尽量补偿一定的射程，以实现消除ΔX增量的目标。这部分计算采用代码实现，在 3.6 节中将详细阐述。

（2）求解水平转动线速度

利用代码得到补偿后的俯仰角度，即可通过弹道表查询其飞行时间t。则在已知回转角度α_1、炮口初速度v_{oya}的情况下进行求解水平转动线速度如下

$$v_1 = \frac{y_a}{t} - v_{oya} \tag{3.25}$$

3.6 代码实现俯仰角度的求解（不考虑风况）

在 3.5 小节中，我们建立了云台姿态求解模型，其中求补偿后的俯仰角度的过程将在该小节中详细阐述。该过程是整个云台姿态求解模型的难点和关键实现方法。我们将先阐述代码实现求解补偿后的俯仰角度的可行性分析。

3.6.1 代码实现最小化代价函数的可行性分析（不考虑风况）

我们需要编写代码求取最佳俯仰角度α_2，以实现最小化以下代价函数（cost function）：

$$J(\alpha_2) = |X_{distance}(\alpha_2) + \Delta X(\Delta v_0, \Delta \alpha_2) - x_{oa}| \qquad \alpha_2 \in (0{,}70^\circ) \tag{3.26}$$

其中$X_{distance}(\alpha_2)$为当俯仰角度为α_2时查弹道表表得到的射程，ΔX为考虑移动炮口、无人艇姿态变化的射程增量，x_{oa}为沿射击面的救援目标距离。

该代价函数仅在$J(\alpha_2)$函数值为 0 时不可导，其他定义域下处处可导。

定义以下函数用于观察$J(\alpha_2)$的单调性。

$$J_{temp}(\alpha_2) = X_{distance}(\alpha_2) + \Delta X(\Delta v_0, \Delta \alpha_2) - x_{oa} \quad \alpha_2 \in (0{,}70^\circ) \tag{3.27}$$

我们统一输入的俯仰角度单位为角度制，则求导可得：

$$J'_{temp}(\alpha_2)=\frac{\partial X}{\partial \alpha_2}+\frac{\partial X}{\partial v_0}\times\left(\frac{\pi\sqrt{v_{oxa}^2+v_{oza}^2}}{180}\cos\left(\alpha_2\times\frac{\pi}{180}+\vartheta_1\right)\right)+\frac{\partial}{\partial \alpha_2}\left(\frac{\partial X}{\partial \alpha_2}\right)$$
$$\times\left(\frac{-v_{oxa}\sin(\frac{\pi}{180}\alpha_2)+v_{oza}cos(\frac{\pi}{180}\alpha_2)+v_2}{v_0}\right)+\left(\frac{\partial X}{\partial \alpha_2}\right)$$
$$\times\left(\frac{\pi\sqrt{v_{oxa}^2+v_{oza}^2}}{180}\cos\left(\alpha_2\times\frac{\pi}{180}+\vartheta_2\right)}{v_0}\right) \tag{3.28}$$

其中 $\vartheta_1=\arctan\left(\frac{v_{oxa}}{v_{oza}}\right)$、$\vartheta_2=\arctan(\frac{v_{oza}}{-v_{oxa}})$

我们先观察(3.28)式子的第一项。由弹道表v_0观察$\frac{\partial X}{\partial \alpha_2}$值的分布图 3.13，观察可得敏感因子$\frac{\partial X}{\partial \alpha_2}$分布在 0.4~0.7 之间。

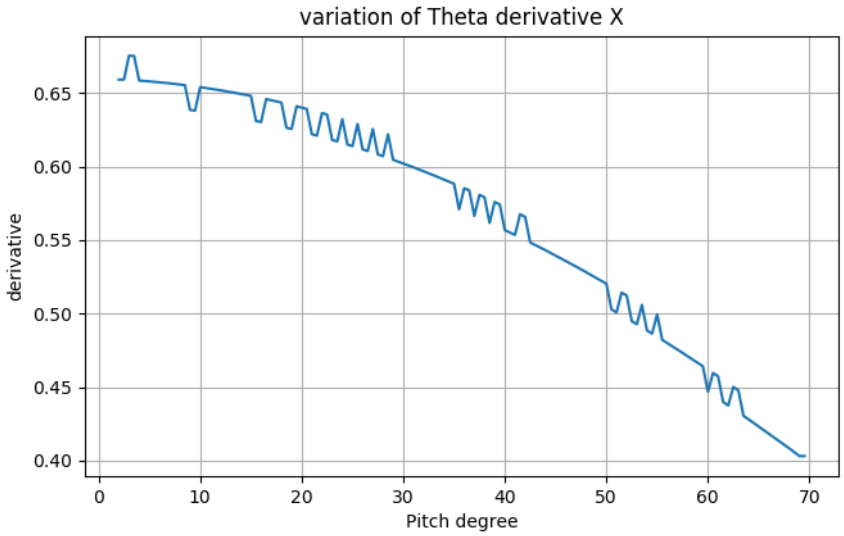


图 3.13 敏感因子$\frac{\partial X}{\partial \alpha_2}$函数值分布图

我们继续观察式子(3.28)的第 2~4 个相加项。进行抛射时，v_{oza}和v_{oxa}的值大多在-2~2m/s 内。观察得该 3 个相加项的分母都较大，则(3.28)式子的第 2~4 个相加项的绝对值与第 1 个相加项相比较小。综合得式子(3.27)的导数恒大于零，为单调递增函数。

我们可作图进行验证。令$v_{oza}=-2m/s, v_{oxa}=2m/s$代入,得到以下$J_{temp}(\alpha_2)$导数值分布图 3.14 和函数值分布图 3.15。我们可以发现在定义域内$J_{temp}(\alpha_2)$的第一项的函数值和导数值都远大于第二项。由于弹道表的敏感因子$\frac{\partial X}{\partial \alpha_2}$较大，则$J_{temp}(\alpha_2)$的单调性

主要由$\frac{\partial X}{\partial \alpha_2}$的值决定。故可证明$J_{temp}(\alpha_2)$的导数恒大于 0，为单调递增函数。

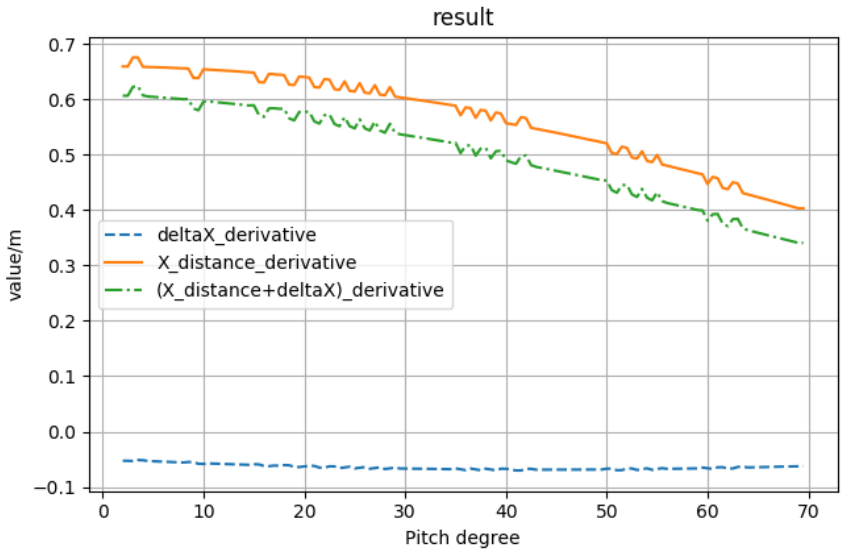


图 3.14　$J_{temp}(\theta_0)$导数值分布图

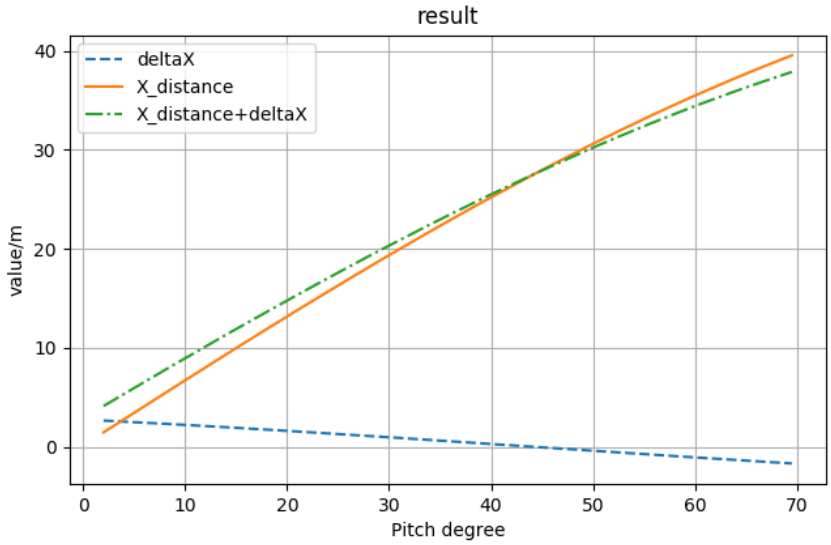


图 3.15$J_{temp}(\theta_0)$函数值分布图

设$x_{oa}=25m$，我们可绘图得到代价函数$J(\alpha_2)$的函数值分布图 3.16。其图形类似于凸函数，可设置优化步长 δ（步长 δ 需为随循环次数变化的单调非递增函数），即可迅速找到可使$J(\alpha_2)$接近最小化的α_2。

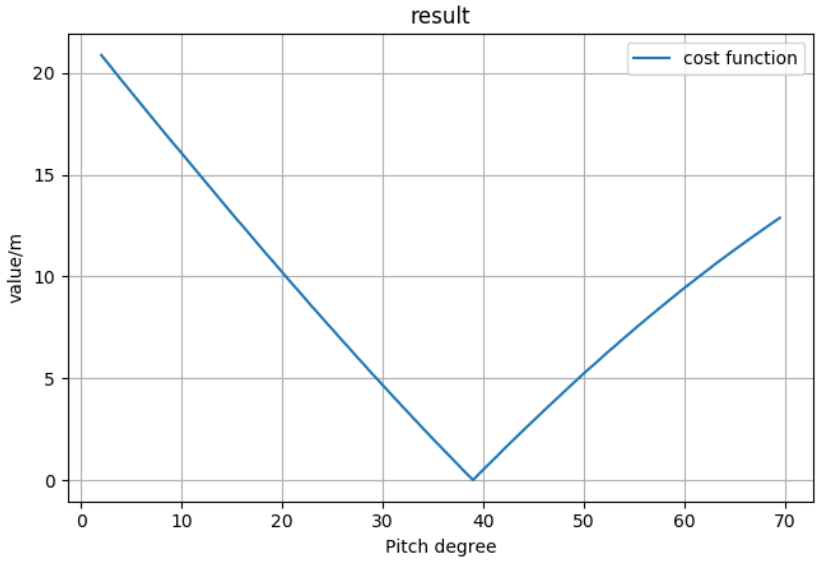


图 3.16 代价函数J(α_2)函数值分布图

3.6.2 代码实现最小化代价函数（不考虑风况）

```
def compute_pitch_degree(V_updated_aim，X_target_position_aim_t):
    alpha_temp=search_data_by_distance(X_target_position_aim_t)
    while(True):
        X_distance = search_data_by_input(alpha_temp)[3]
        delta_V0=compute_delta_V0(V_updated_aim,alpha_temp)
        delta_Theta = compute_delta_Theta(V_updated_aim,alpha_temp,V0)
        V0_derivative_X = compute_V0_derivative_X(V0)
        Theta_derivative_X = compute_Theta_derivative_X(alpha_temp)
        X_real=X_distance+V0_derivative_X*delta_V0+Theta_derivative_X*delta_Th
eta
        if(abs(X_real-X_target_position_aim_t)<0.1):
            pitch_degree = alpha_temp
            return pitch_degree # the pitch degree After the compensation
        if (X_real < X_target_position_aim_t):alpha_temp = alpha_temp + delta
        else:alpha_temp = alpha_temp - delta
```

图 3.17 求解俯仰角度的核心代码

我们编写一个compute_pitch_degree 函数，以求得最佳角度，其核心代码如图3.17。该函数有两个输入。一个输入是式子(3.18)中计算得到的瞄准坐标系下的抛射器初速度，另一个输入是式子(3.20)的x_{oa}。输出为补偿后的俯仰角度。

该函数先是通过x_{oa}进行查询弹道表，得到最初未补偿的俯仰角度α_{temp}。接下来进入一个 while 循环中。在循环中，先查询弹道表求解当前α_{temp}对应的抛射距离$\mathrm{X_{distance}}$。随后根据式子(3.21)和式子(3.22)计算当前状态下的Δv_0、$\Delta \alpha_2$，以及计算当前对应的敏感因子$\frac{\partial X}{\partial v_0}|_{v_0=v_0}$、$\frac{\partial X}{\partial \alpha_2}|_{\alpha_2=\alpha_{temp}}$。根据式子(3.24)得到射程增量$\Delta \mathrm{X}$。于是可计算该俯仰角度下的真实射程为：

$$X_{real}(\theta_0) = X_{distance}(\theta_0) + \Delta X(\Delta v_0, \Delta \alpha_2) \tag{3.29}$$

随后程序进入判断环节，若是满足$|X_{real} - x_{oa}| < \varepsilon$(ε 为一固定小量)，则将当前$\alpha_{temp}$作为最佳俯仰角度，停止 while 循环，输出该最佳俯仰角度。

若不满足该停止条件，则进入俯仰角度的调整，若是真实射程大于x_{oa}，则$\alpha_{temp} \coloneqq \alpha_{temp} - \delta$，若是真实射程小于$x_{oa}$，则$\alpha_{temp} \coloneqq \alpha_{temp} - \delta$。其中$\delta$是一个以迭代次数为参数的单调非递增函数值。

3.7 算例（不考虑风况）

我们输入初始条件，设此时无人艇的状态为：$\mathrm{u} = 3\mathrm{m/s}, \mathrm{v} = 2\mathrm{m/s}, \mathrm{w} = 1\mathrm{m/s}, \psi = 115°, \theta = 20°, \phi = -18°, \mathrm{p} = 0.02\mathrm{rad/s}, \mathrm{q} = 0.6\ \mathrm{rad/s}, \mathrm{r} = 0.7\mathrm{rad/s}$。目标位置和无人艇在空间固定坐标系下的坐标分别为(40,20)，(20,40)。抛射云台在水平附体坐标系中的坐标为(2,1)，设定δ为 0.5°，ε为 0.2m。

得到计算结果如表 3.2 所示。

表 3.2　算例计算结果

参数名称	参数值
水平附体坐标系下救援目标坐标	[-26.5785,-9.6738]
经过平均分组时间 t 秒后坐标	[-32.3454, -14.0214]

方位角	23.4362°
取回转角度（取 $\lambda=\frac{5}{6}$）	19.5301°
取俯仰线速度	0.05m/s
瞄准坐标系下救援目标坐标	[35.1719, 2.4015]
瞄准坐标系下炮口初速度	[-0.4872,0.2367,1.2937](m/s)
未补偿的俯仰角度	59.5°
随着俯仰角度的调整，代价函数值	1.33224512045 1.07904919517 0.839672446054 0.583995917831 0.342238896791 0.0840528952186
补偿后俯仰角度	57.0°
回转线速度	0.8421m/s
真实飞行时间	2.226s
假设平均飞行时间	1.966s
程序运算耗时	<0.002 s

程序的输出为：俯仰角度$(57.0^\circ-\theta)=37.0^\circ$，俯仰线速度 0.050m/s（由下到上），回转角度 19.5°，回转线速度 0.8421m/s

程序中可计算的误差为真实飞行时间和假设平均飞行时间的不同而导致的救援目标位置估计误差。可进行定量分析，误差分析如表 3.3。

表 3.3 算例误差分析

参数名称	参数值
真实救援目标位置估计（水平附体坐标系）	[-33.1073, -14.5957]
真实救援目标位置（瞄准坐标系）	[36.0819, 2.6881]
抛射体入海点估计（瞄准坐标系）	[35.2559, 2.4015]
绝对误差/m	0.87427
相对误差	3.091%

我们可知相对误差满足第 1 章所设定的误差允许条件，则该云台姿态求解模型是一个解算迅速、准确度高的模型。

还有其他的误差需要通过现场分析，例如：

（1）现场救援目标的快速移动引起的误差

（2）现场云台调整时间过长引起的误差

（3）现场风的影响引起的误差（将在第 4 章进行分析）

（4）现场云台俯仰角度变化引起的误差

3.8 本章小结

本章建立了搭载抛射器的云台姿态求解模型（未考虑风况）。本章首先通过上一章的缆绳拖曳模型建立了对应的弹道表。随后在给定任意无人艇大地系坐标、救援目标大地系坐标、无人艇的速度与航向、无人艇的姿态、无人艇的姿态变化角速度、抛射体的初速度的情况下，建立云台姿态求解模型，可实现在现场立刻求解出救援云台的回转角度、回转角速度、俯仰角度、俯仰角速度。通过算例仿真，我们可得到该模型有着耗时少、准确度高的特点。

第 4 章云台姿态求解模型（考虑风况）

4.1 引言

在现场恶劣的海上救援过程中，巨大的风力带来的影响是不可忽略的。在本章中，本文将利用第 3 章所建立的云台姿态求解模型，并考虑风的影响所带来的抛射偏差，建立更加复杂的云台姿态求解模型。通过俯仰角度的补偿，减小估计入海点与救援目标的距离。

4.2 对风的建模

空气的水平移动称为风，风有风速和风向，通常将风的来向称为方向，从正北方顺时针转至风的来向的角度为风向方位角。风向和风速大小主要与大气压力场的分布、地球的自转以及空气的粘性有关，也与气压梯度的大小和方向、科氏惯性力、空气粘性系数等有关。通过状态方程，气压与气温及密度有关，而气温又随季节、高度、下垫面特性（如海洋）而变化，加上密度随高度变化，这使得风随地点、高度、季节、昼夜的变化很大，难以用一个统一的公式来描述风的分布。

图 4.1 为摩擦层中风速随时间迅速变化的示意图。

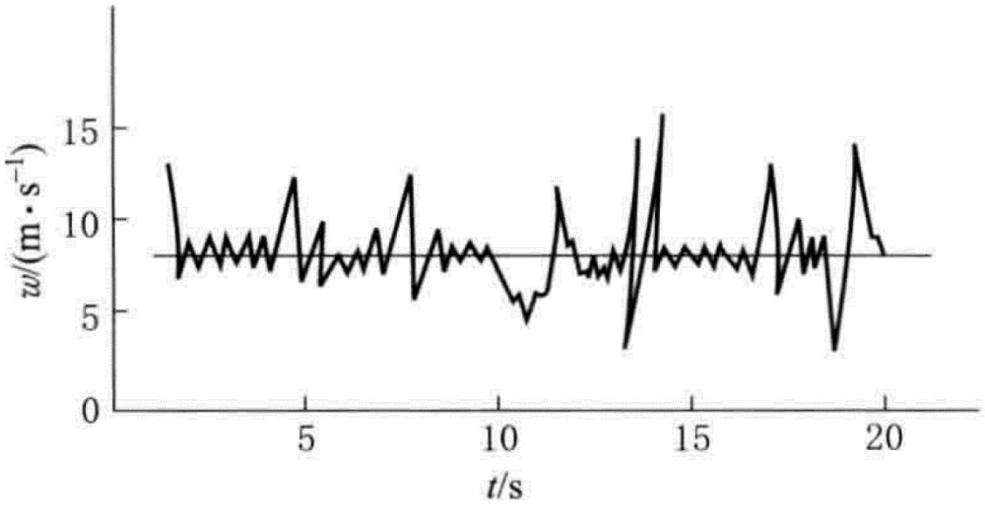


图 4.1 摩擦层中风速随时间变化示意图

为了研究方便，通常将变化着的风在一段时间内平均，我们将此平均值称为平均风，

记为w，而瞬时风与此平均值之差称为阵风。摩擦层中的风对抛射体、炮弹的飞行影响很大。在外弹道学中，把近地面层中的风称为低空风，而将近地面层以上的风称为高空风，沿弹道的平均风将使各抛射体的弹道相对于无风弹道产生相同的偏差，如射程偏差、方向偏差。

习惯上将风力大小用风力等级来描述，如 0~13 级风分别称为无风、软风、轻风、微风、和风、清劲风、强风、疾风、大风、烈风、狂风、暴风、飓风。他们都代表一定风速范围，如 6 级强风，风速为 10.8~13.8m/s，平均为 12.3m/s，变化范围为$\pm 1.5\mathrm{m/s}$。风力等级 K 与其代表的平均风速w可用以下近似公式计算[24]：

$$w = K + 0.17K^2, K = 0,1\cdots,17 \tag{4.1}$$

风速沿时间坐标或空间坐标的变化过程都是随机过程，在一定时间或空间范围内，还可视为平稳随机过程。可用谱密度或相关函数加以描述。抛射体在飞行过程中所受的风速变化过程也是一个随机过程，用随机过程理论来研究风的影响目前正在进行中。由于风对抛射体飞行中的运动特性影响较大，必须进行实际测量才能较好地考虑风影响。但就大范围而言，风的变化也表现出一定的规律，如沿海地区白天风从海洋吹向陆地，晚上风从陆地吹向海洋（常称海陆风）等。

4.3 对风的修正

实际风速、风向沿高度是变化的。为便于修正，我们常使用一个不变的平均风来代替随高度变化的实际风，使其对射程和侧偏产生的效果相同，这个平均风称为弹道风。其中纵风W_x主要影响射程，横风W_z主要影响方向。

4.3.1 横风产生的侧偏（相对运动法）

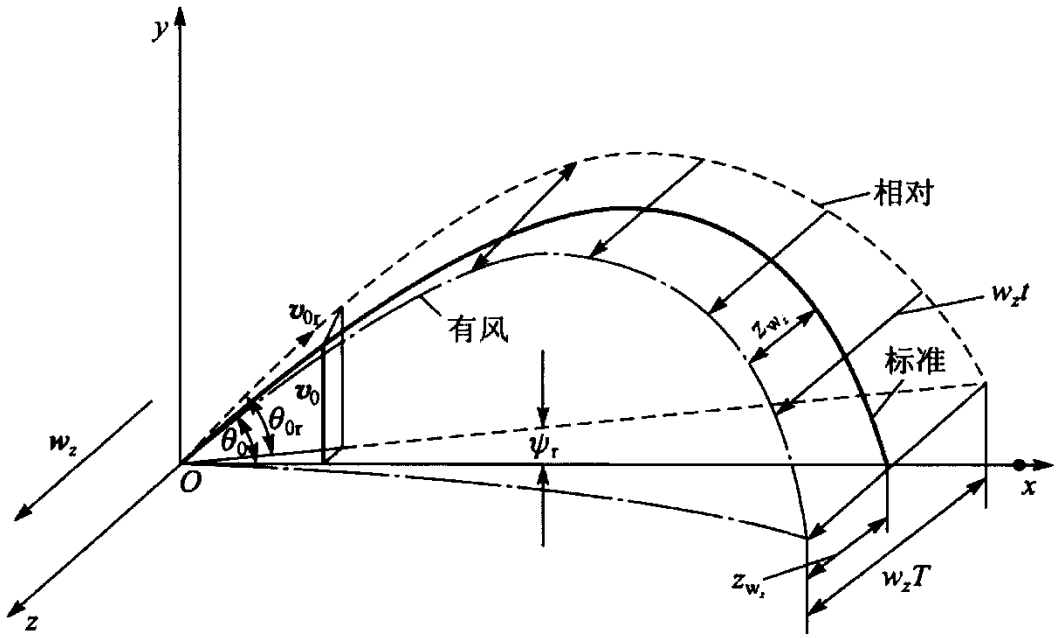


图 4.2 横风产生的侧偏示意图

如图 4.2 所示，无风时的弹道为黑色曲线，有横风W_z时，作一动坐标系随同横风一起运动，则在动坐标系里如同无风一样，但在动坐标系里的观察者看来，初速度以及整个射击面向左偏了一个角度ψ_r。

$$\psi_r = -\frac{W_z}{V_0 cos\theta_0} \tag{4.2}$$

那么我们可以得到在任一时刻 t、水平距离 x 的抛射体在相对运动中的侧偏为

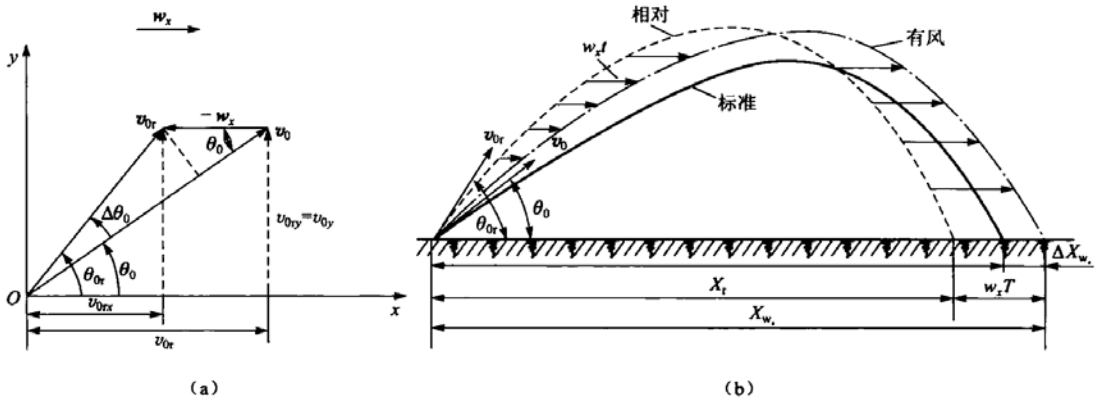
$$z_r = x\psi_r = -\frac{W_z x}{V_0 cos\theta_0} \tag{4.3}$$

但由于动坐标系向右平移产生了牵连运动，在时刻 t 牵连位移$z_e = W_z t$。根据运动合成关系得：

$$\mathrm{z} = z_r + z_e = W_z\left(\mathrm{t} - \frac{x}{V_0 cos\theta_0}\right) \tag{4.4}$$

实际上，在动坐标系里和在静坐标系里的观察者将同时观察到弹丸在同一时刻着地，故在相对坐标系与绝对坐标系里，飞行时间是相同的。在实际中，由于空气弹道水平分速度不断减小，故平均水平分速度$v_x < V_0 cos\theta_0$，因此可知抛射体由横风引起的侧偏总是顺风向的。

4.3.2 纵风对射程的修正



4.3 纵风对射程的影响

在有纵风w_x时，作一动坐标系随风平移，则在动坐标系里无风，但相对初速矢量为

$$v_{0r} = v_0 - w_x \tag{4.5}$$

我们由图 4.3 可以看出，相对初速v_{0r}和相对射角θ_{0r}以及速度和射角的改变量的大小分别为：

$$v_{0r} = \sqrt{v_{0rx}^2 + v_{0ry}^2} = \sqrt{(v_0 cos\theta_0 - w_x)^2 + (v_0 sin\theta_0)^2}$$

$$v_{0r} = v_0 \sqrt{1 - 2\frac{w_x}{v_0} cos\theta_0 + (\frac{w_x}{v_0})^2} \tag{4.6}$$

$$\Delta v_0 = v_{0r} - v_0 \approx -w_x \cos\theta_0 \tag{4.7}$$

$$\theta_{0r} = \arctan\left(\frac{v_0 sin\theta_0}{v_0 cos\theta_0 - w_x}\right) = \arctan\left[\frac{1}{1 - \frac{w_x}{v_0 cos\theta_0}} tan\theta_0\right] \tag{4.8}$$

$$\Delta \theta_0 = \theta_{0r} - \theta_0 \approx \frac{w_x sin\theta_0}{v_0} \tag{4.9}$$

根据弹道表可求得无风射程 X，再计算得由动坐标系平移产生的牵连射程$w_x T$，并考虑求得在动坐标系里的相对射程X_r等，既得有纵风时抛射体相对于地球的射程为$X_{w_x} = X_r + w_x T$。

$$\Delta X_{w_x} = (X_r + w_x T) - X = (X_r - X) + w_x T \tag{4.10}$$

而X_r与 X 不同是因为θ_{0r}、v_{0r}不同于θ_0、v_0，于是我们得纵风对射程的修正公式

$$\Delta X_{w_x} = w_x\left(T - \frac{\partial X}{\partial v_0}cos\theta_0 + \frac{\partial X}{\partial \theta_0}\frac{sin\theta_0}{v_0}\right) \tag{4.11}$$

4.4 云台姿态求解模型（考虑风况）

本文只分析当风力等级小于等于 5 级时的情况，即$K \leq 5$。与 3.5 节阐述的计算步骤相同，不同的是需要考虑风的影响。先取值确定了回转角度并设定瞄准坐标系，将所有的运算都分解到射击面和垂直射击面的瞄准坐标系OY_a轴上。在射击面上，通过计算由于炮口移动影响下和纵风影响下的射程增量，调整得到一个补偿后的俯仰角度和适当的俯仰转动线速度。随后查弹道表得到飞行时间，在OY_a轴方向上考虑横风的影响下的射程增量，计算得到回转线速度。其中回转角度和俯仰转动线速度的求法与 3.5 节相同，不同的是俯仰角度和回转线速度的求解过程。图 4.4 为考虑风况下的云台姿态求解模型的计算流程图。

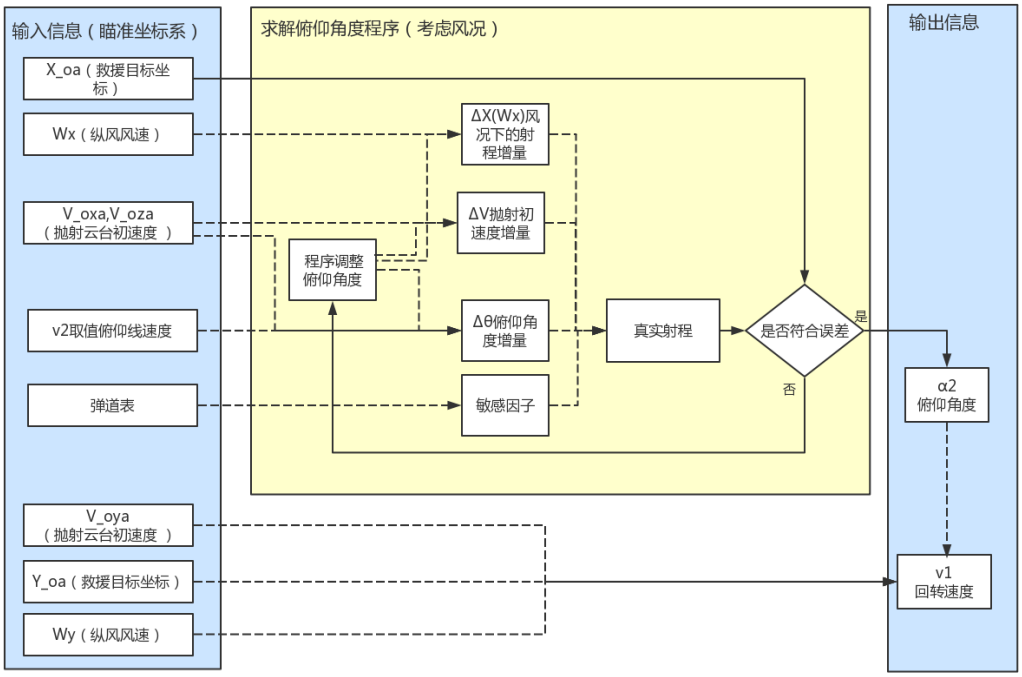


图 4.4 云台姿态求解模型的计算流程图（考虑风况）

4.4.1 瞄准坐标系下的风速

已知风的风向方位角θ_w，以及风力等级 K。可通过式子(4.1)计算得到风速大小为w。

风速在空间固定坐标系下的向量坐标为

$$\vec{w} = (-w\cos\theta_w, -w\cos\theta_w) \tag{4.12}$$

无人艇在空间固定坐标系下的速度向量坐标表示

$$V_s^T = \begin{bmatrix} cos\psi & -sin\psi \\ sin\psi & cos\psi \end{bmatrix} (U,V)^T \tag{4.13}$$

即：

$$\overrightarrow{V_s} = (Ucos\psi - Vsin\psi, Usin\psi + Vcos\psi) \tag{4.14}$$

则在空间固定坐标系下相对风速的向量坐标为：

$$\overrightarrow{w_r} = \vec{w} - \overrightarrow{V_s} \tag{4.15}$$

在水平附体坐标系下，相对风速沿各轴的分量为

$$w_{hx} = \overrightarrow{w_r} \cdot \vec{X} \tag{4.16}$$

$$w_{hy} = \overrightarrow{w_r} \cdot \vec{Y}$$

则我们可以得到在瞄准坐标系下，相对风速沿各轴的分量为(不考虑上升风)：

$$\begin{bmatrix} w_{ax} \\ w_{ay} \\ w_{az} \end{bmatrix} = \begin{bmatrix} cos(\pi - \alpha_1) & -sin(\pi - \alpha_1) & 0 \\ sin(\pi - \alpha_1) & cos(\pi - \alpha_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{0x} \\ v_{0y} \\ 0 \end{bmatrix} \tag{4.17}$$

其中$\mathrm{w_{ax}}$即为沿着射击面的纵风，而$\mathrm{w_{ay}}$即为垂直射击面的横风。

4.4.2 瞄准坐标系下风的影响

(1) 瞄准坐标系下纵风的影响

在瞄准坐标系的射击面（即$\mathrm{X_a}OZ_a$面）上，由式子(4.11)得，纵风造成的射程增量为：

$$\Delta X_{w_x} = w_x \left(T - \frac{\partial X}{\partial v_0} cos\alpha_2 + \frac{\partial X}{\partial \alpha_2} \frac{sin\alpha_2}{v_0} \right) \tag{4.18}$$

则在某俯仰角度α_2下，抛射体的真实射程为：

$$x_{oaf} = X_{distance}(\alpha_2) + \Delta \mathrm{X}(\Delta v_0, \Delta\alpha_2) + \Delta X_{w_x}(\mathrm{w_{ax}}) \tag{4.19}$$

其中$X_{distance}(\alpha_2)$为当俯仰角度为α_2时查弹道表得到的射程，$\Delta X(\Delta v_0, \Delta\alpha_2)$为考虑移动炮口、无人艇姿态变化的射程增量，$\Delta X_{w_x}(\mathrm{w_{ax}})$为纵风引起的射程增量。我们的目标即为寻找一个俯仰角度，能够实现尽量减小x_{oaf}和x_{oa}的差值。对于补偿后的俯仰角度

的求解将在 4.5 小节中详细阐述。

（2）瞄准坐标系下横风的影响

随后研究在瞄准坐标系$\mathrm{OY_a}$轴上抛射体的运动。横风大小为W_y。

在动坐标系里的观察者看来，初速度以及整个射击面向左偏了一个角度ψ_{ra}。

$$\psi_{ra} = -\frac{W_y - \mathrm{v_1} - v_{oya}}{V_0 cos\alpha_2} \tag{4.20}$$

那么，水平距离 x 的抛射体在相对运动中的侧偏为

$$y_{ra} = x\psi_{ra} = -\frac{(W_y - v_1 - v_{oya})x}{V_0 cos\alpha_2} \tag{4.21}$$

但由于动坐标系向右平移产生了牵连运动，在时刻 T 牵连位移$y_{ea} = W_y T$。我们可以根据运动合成关系得：

$$\mathrm{y_a} = y_{ra} + y_{ea} = W_y T - \frac{(W_y - v_1 - v_{oya})x_{oaf}}{V_0 cos\alpha_2} \tag{4.22}$$

我们的目标即为寻找一个回转线速度v_1，能够实现尽量减小$\mathrm{y_a}$和$\mathrm{y_{oa}}$的差值。由于除了v_1外所有参数值都可以计算或查表得到，故可列出v_1计算式：

$$v_1 = W_y - \frac{(W_y T - y_{oa})V_0 cos\alpha_2}{x_{oaf}} - v_{oya} \tag{4.23}$$

4.5 代码实现求取俯仰角度（考虑风况）

在 4.4 小节中，我们建立了考虑风况的云台姿态求解模型，其中求补偿后的俯仰角度的过程将在该小节中详细阐述。我们将先阐述代码实现求解补偿后的俯仰角度的可行性分析，随后对核心代码的逻辑进行分析。

4.5.1 代码实现最小化代价函数的可行性分析（考虑风况）

我们需要编写代码求取最佳俯仰角度α_2，以最小化以下代价函数（cost function）：

$$\mathrm{J_w}(\alpha_2) = \left|X_{distance}(\alpha_2) + \triangle \mathrm{X}(\Delta v_0, \Delta\alpha_2) + \triangle X_{w_x}(\mathrm{w_{ax}}) - x_{oa}\right| \tag{4.24}$$

其中$X_{distance}(\alpha_2)$为当俯仰角度为α_2时查弹道表得到的射程，$\Delta X(\Delta v_0, \Delta\alpha_2)$为考虑

移动炮口、无人艇姿态变化的射程增量，$\Delta X_{w_x}(\mathrm{w_{ax}})$为纵风引起的射程增量， x_{oa}为沿射击面的救援目标距离。该代价函数仅在$\mathrm{J_w}(\alpha_2)$函数值为 0 时不可导，其他$\alpha_2 \in (0,70^\circ)$处处可导。

设在不考虑风的影响下的射程为$\mathrm{X_{ww}}$(Without Wind)：

$$X_{ww}(\alpha_2) = X_{distance}(\alpha_2) + \Delta \mathrm{X}(\Delta v_0, \Delta\alpha_2) \quad \alpha_2 \in (0,70^\circ) \tag{4.25}$$

我们将式子(4.19)代入式子(4.18)则有：

$$J_w(\alpha_2) = \left|X_{ww}(\alpha_2) + \Delta X_{w_x}(w_{ax}) - x_{oa}\right| \quad \alpha_2 \in (0,70^\circ) \tag{4.26}$$

设函数$\mathrm{J_{wtemp}}(\alpha_2)$来研究代价函数$\mathrm{J_w}(\alpha_2)$的单调性。

$$J_{wtemp}(\alpha_2) = X_{ww}(\alpha_2) + \Delta X_{w_x}(\mathrm{w_{ax}}) - x_{oa} \quad \alpha_2 \in (0,70^\circ) \tag{4.27}$$

统一输入的俯仰角度单位为角度制,其导数为：

$$J'_{wtemp}(\alpha_2) = X'_{ww}(\alpha_2) + w_{ax} \times \left(\frac{\partial T}{\partial \alpha_2} + \frac{\partial X}{\partial v_0}\frac{\pi}{180} sin\left(\frac{\pi}{180}\alpha_2\right) + \frac{\partial}{\partial \alpha_2}\left(\frac{\partial X}{\partial \alpha_2}\right)\right.$$
$$\left. \times \frac{sin\left(\frac{\pi}{180}\alpha_2\right)}{v_0} + \frac{\partial X}{\partial \alpha_2} \times \frac{\pi}{180}\frac{cos\left(\frac{\pi}{180}\alpha_2\right)}{v_0}\right) \tag{4.28}$$

我们先观察$\mathrm{J'_{wtemp}}(\alpha_2)$的第一项。实则$\mathrm{X'_{ww}}(\alpha_2) = J'_{temp}(\alpha_2)$，$\mathrm{X_{ww}}(\alpha_2) = \mathrm{J_{temp}}(\alpha_2)$。在第 3 章的 3.6.1 小节中，我们证明了$\mathrm{J_{temp}}(\alpha_2)$为单调递增函数,则。根据图 3.14 得$\mathrm{X_{ww}}$的导数值约分布在 0.3~0.7 的范围内。

随后观察$\mathrm{J'_{wtemp}}(\alpha_2)$的第二项$\Delta X'_{w_x}(\mathrm{w_{ax}})$。在五级风力等级以下时，$\mathrm{w_{ax}}$的计算值在-7~7m/s 范围内。然后进行观察$\mathrm{w_{ax}}$后的大括号内的相加项。作图得$\frac{\partial T}{\partial \alpha_2}$函数值分布图如图 4.5。其值为 0.036~0.041。其余的相加项中由于分母较大，因此与敏感因子$\frac{\partial T}{\partial \alpha_2}$相比，绝对值较小可以忽略。则我们得到第二项$\Delta X'_{w_x}(\mathrm{w_{ax}})$的函数值范围为$-0.3$~$0.3$。

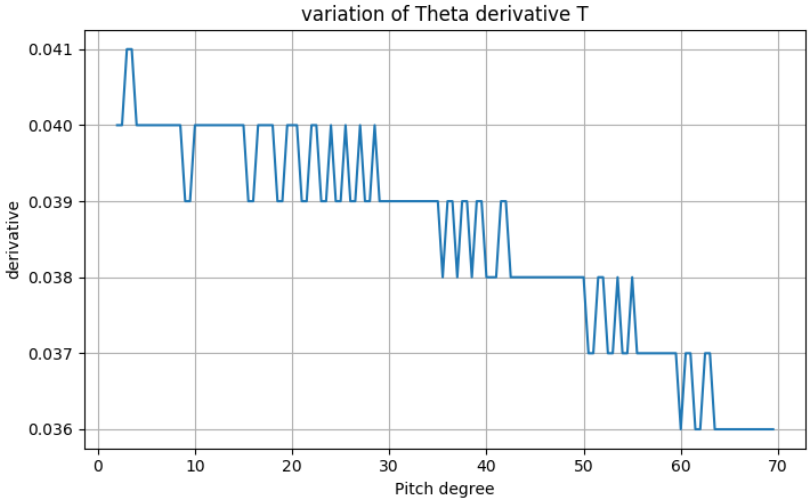


图 4.5 敏感因子$\frac{\partial T}{\partial \alpha_2}$函数值分布图图

我们可以作图进行验证。设纵风w_{ax} =7m/s，，$J_{wtemp}(\alpha_2)$导数和函数值的图像如图 4.6 和图 4.7。可知当风力小于等于 5 级时，风力和移动炮口所带来的微分量与敏感因子相比较小，则$J_{wtemp}(\alpha_2)$仍是以α_2为参数的单调递增函数。

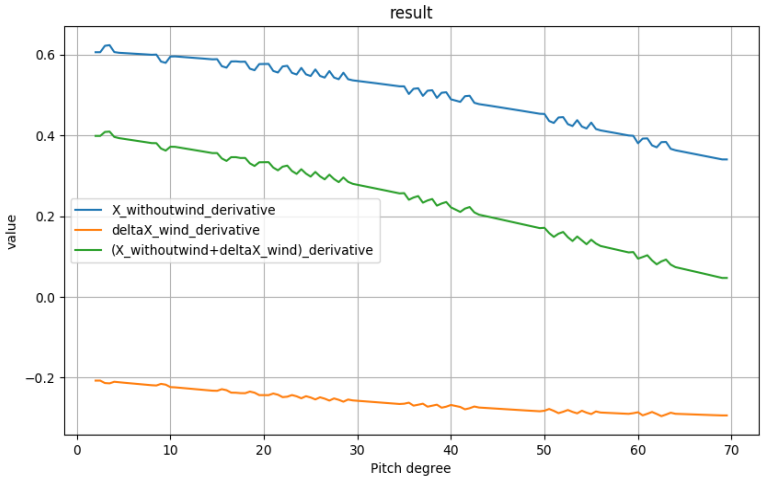


图 4.6 $J_{wtemp}(\alpha_2)$导数值分布图

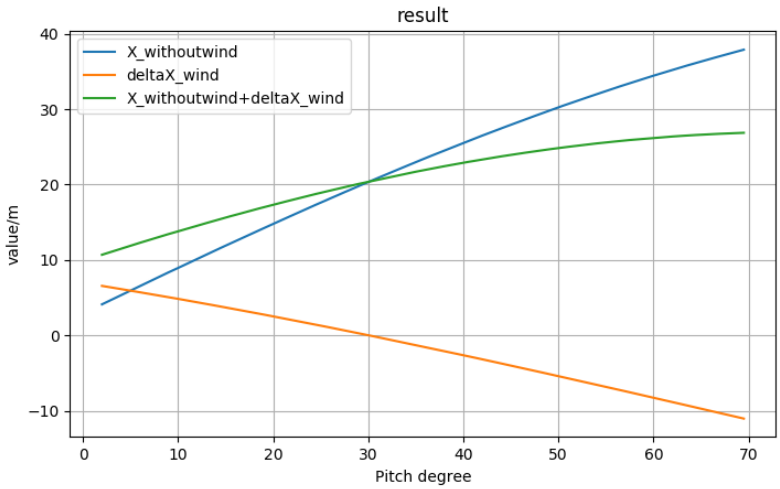


图 4.7　$\mathbf{J_{wtemp}(\alpha_2)}$函数值分布图

设$x_{oa} = 25m$，我们可绘图 4.8 得到代价函数$J_w(\alpha_2)$的函数值分布图 3.16。可知在最小值的两侧皆为单调函数，可设置优化步长 δ（步长 δ 需为随循环次数变化的单调非递增函数），即可迅速找到可使$J_w(\alpha_2)$接近最小化的α_2。

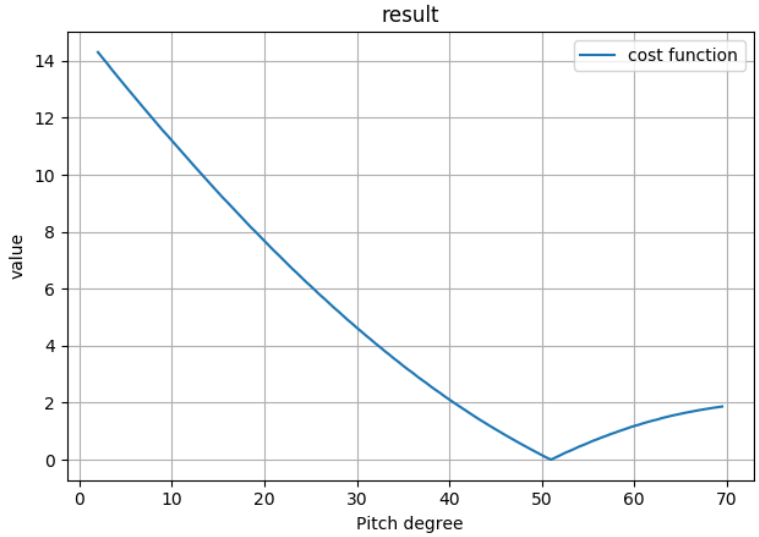


图 4.8 代价函数$J_w(\alpha_2)$函数值分布图

4.5.2 代码实现最小化代价函数（考虑风况）

我们编写一个 compute_pitch_degree_updated 函数，以求取补偿后的俯仰角度。核心代码如图 4.9。

```
    def compute_pitch_degree_updated(X_target_position_aim_t,V_wind):
  distance_without_wind = X_target_position_aim_t
  while(True):
    pitch_degree_temp=compute_pitch_degree(V_updated_aim,
distance_without_wind)
      deltaX_wind=compute_deltaX_wind(pitch_degree_temp,V_wind)
      X_real=distance_without_wind + deltaX_wind
      error = X_real - X_target_position_aim_t
      if(abs(error)<0.2):return pitch_degree_temp
      if(error>0):distance_without_wind -=delta
      else:distance_without_wind +=delta
```

图 4.9 compute_pitch_degree_updated 函数的核心代码

该函数有两个输入。一个输入是在瞄准坐标系下的相对风速，另一个输入是式子(3.20)的x_{oa}。输出为补偿后俯仰角度。

该函数先是将x_{oa}作为X_{ww}(distance_without_wind)的初值，随后X_{ww}进入 while 循环。利用第 3 章的云台姿态求解模型，求出在当前给定距离X_{ww}下未考虑风力影响的对应俯仰角度α_{temp}。随后进行计算当前风力下的射程增量ΔX，得到当前俯仰角度下的真实飞行距离，若真实飞行距离大于x_{oa},则$\alpha_{temp} \coloneqq \alpha_{temp} - \delta$，若真实飞行距离小于$x_{oa}$，则$\alpha_{temp} \coloneqq \alpha_{temp} - \delta$。其中$\delta$是一个以迭代次数为参数的单调非递增函数值。

4.6 算例（考虑风况）

设此时救援目标周围海域的风力等级 5 级，其风向方位角为-80°，其余输入条件与 3.7 小节相同。输入程序中，我们得到计算结果如表 4.1

表 4.1　算例计算结果

参数名称	参数值
大地系下风速	9.25 m/s
大地系下相对风速	[2.07657879　4.89419098]
水平附体坐标系下相对风速	[3.55804328 -3.95039401]
瞄准坐标系下相对风速	[-2.0327, 4.9125]
回转角度（取 $\lambda=\frac{5}{6}$）	19.5301°
取俯仰转动线速度	0.05m/s
未补偿的俯仰角度	57.0°
随着 while 循环，代价函数值变化	3.0018985006 2.61903476842 2.23662329722 1.85375568745 1.47307609425 1.08897589109 0.70909895481 0.38468059422 0.00278583233
补偿后的俯仰角度	65.5°
真实飞行时间	2.537s
分组平均时间	1.96625s
回转线速度	1.3536m/s
耗时	<0.013 s

程序的输出为：俯仰角度$(65.5^\circ-\theta)=45.5^\circ$，俯仰线速度 0.050m/s（由下到上），回转角度 19.5°，回转线速度 1.3536m/s。

程序中可计算的误差为真实飞行时间和假设平均飞行时间的不同而导致的救援目标位置估计误差。可进行定量分析，误差分析如表 4.2。

表 4.2 算例误差分析结果

参数名称	参数值
真实救援目标位置估计（水平附体坐标系）	[-34.0194，-15.2834]
真实救援目标位置（瞄准坐标系）	[37.1714, 3.0312]
抛射体入海点估计（瞄准坐标系）	[35.1719，2.4015]
绝对误差	2.0964m
相对误差	7.412%

还有许多其他因素可能引起误差，需要现场试验分析：

（1）风速的变化，包括上升风、下降风引起的射程误差

（2）恶劣海况下救援目标移动速度过快导致的误差

（3）风速的影响下导致飞行时间的微小变化

4.7 小结

本章建立了考虑风况的云台姿态求解模型。本章在第 3 章的基础上，考虑了横风和纵风对抛射体的射程和方向的影响，通过对俯仰角度和回转线速度的修正，使该云台姿态求解模型在 5 级风力等级以下可在现场迅速求解出抛射云台的回转角度、回转线速度、俯仰角度、俯仰转动线速度。

第 5 章总结与展望

5.1 工作总结

本文在综合分析了多单元飞行模型和弹道修正的国内外研究的基础上，提出了一种基于救生抛射器的缆绳拖曳模型，并利用该模型形成弹道表。在查表的基础上，综合考虑无人艇姿态、无人艇姿态变化、救援目标位置、风向等信息，对查表结果进行修正，建立了云台姿态求解模型。工作总结如下：

（1）建立抛射器的缆绳拖曳模型：对带有无限自由度的软性缆绳采用微元思想进行分析，建立多单元参与飞行的缆绳拖曳模型。针对抛射器的硬件参数，综合考虑每个绳索单元受到的拉力和空气阻力，推导连续性方程，并生成拉力矩阵，利用运算速度优秀的 Numpy 库计算在某俯仰角度、抛射初速度下的飞行时间和抛射轨迹。

（2）多线程编程生成弹道表：利用 ParallelPython 模块进行多线程编程，并利用矢量化的 Ndarray 数据类型加快计算速度，将上述的缆绳拖曳模型生成对应的弹道表，以供后续的查表修正。

（3）建立不考虑风况的云台姿态求解模型：根据弹道修正理论，我们充分考虑无人艇姿态（欧拉角）、和姿态变化角速度、航速、无人艇和救援目标位置等具体现场信息，研究移动炮口对射程和射向的影响，对查表结果进行修正。为了计算得到最佳云台姿态以实现抛射入海点始终处于救援目标附近，在第 3 章中建立代价函数，并利用函数图像和函数表达式的推导证明了迭代法求取最佳姿态的可行性。随后给出算例，利用迭代程序求取补偿后的俯仰角度、俯仰转动线速度、回转角度、回转线速度，并进行了误差分析。

（4）建立考虑风况的云台姿态求解模型：针对现场救援环境的恶劣海况，根据弹道修正理论对现场救援时风对弹道的影响进行考虑，对云台姿态进行修正。在第 3 章的基础上建立更为复杂的云台姿态求解模型，并建立相应的代价函数以证明迭代法求取云台姿态的可行性。同样给出算例和对应的误差分析。

5.2 研究展望

本文所建立的缆绳拖曳模型和云台姿态求解模型已能满足第 1 章所提出的考核指标。为了进一步满足恶劣海况下的救援需求，以下方面有待于进一步的研究：

（1）风力等级大于 6 级以上时的云台姿态求解：在现实的救援环境中，常常会遇到大于风力等级 6 级以上的情况，此时传统的弹道修正理论已经不再适用，需利用风洞模拟实验和其他弹道修正模型来求取补偿后的俯仰角度。

（2）考虑对于飞行时间的修正：在受到移动炮口和风的影响下的弹道飞行时间由于敏感因子存在而有微小改变。本文中的模型对于飞行时间的初始化估计采用按射击面射程进行分组求取平均飞行时间的方式，该方法存在一定误差，可通过修正公式的逆向推导进行分析并优化减小。

（3）考虑浪涌对无人艇的影响：在现场救援中，若能够迅速在现场建立相应浪涌中的无人艇运动模型，对无人艇的欧拉姿态角进行估计，则可更快的求取云台姿态角度。相关研究领域已有较为成熟的无人艇运动的干扰力数学模型和无人艇操纵运动数学模型。若能将这些数学模型与云台姿态求解模型一同应用到海上救援作业中，可大大提高救援成功率。

参考文献

[1] 滕达. 海洋灾害应急管理机制研究[D].广东海洋大学，2015.

[2] The stylebook of Pneumatic Line Throwers. https://restech.no/

[3] Williams, Paul,Sgarioto, Daniel,Trivailo, Pavel. Optimal control of an aircraft-towed flexible cable system. Journal of Guidance Control and Dynamics. 2006

[4] Nakagawa, N., and Obata, A., “Longitudinal Stability Analysis of Aerial-Towed Systems,” Journal of Aircraft, Vol. 29, No. 6, 1992:978-985

[5] Etkin, B. “Stability of a Towed Body,” Journal of Aircraft, Vol. 35, No. 2, 1998: 197–205.

[6] 王海涛,刘浪波,熊伟. 气动抛绳器的改进设计[J]. 液压与气动，2013，(04):110-112.

[7] 刘浪波. 新型抛绳器设计及试验研究[D].大连海事大学，2013.

[8] 陈大年.二级轻气炮内弹道的数值模拟与性能分析.爆破与冲击，1999(1): 37-42

[9] 吴应湘，郑之初，P.Kupschus.二级轻气炮发射性能的数值模拟.中国科学，1995(4): 374-384

[10] 吴静，方茂林，蓝强，鲜海峰.一级气体炮内弹道数值模拟.爆轰波与冲击波，2004(2):67-71

[11] 吴小平,郑友祥,丘光申.拖缆火箭弹道计算研究[J]. 弹道学报，1995(01)

[12] Ablow C M,Schechter S.Numerical simulation of undersea cable dynamics. Ocean Engineering. 1983

[13] 王海潮,陆鸣,余静. 抛索火箭平面运动弹道模型研究[J]. 兵器装备工程学报，2016(04).

[14] Frank G, Moore. Body alone aerodynamic of guided and unguided projectiles at Subsonic, Transonic and Supersonic math numbers. AD-754098. 1972.

[15] 浦发. 外弹道学[M]. 国防工业出版社，1980

[16] 王金贵. 气体炮原理及技术. 国防工业出版社，2001.

[17] 王中原. 关于旋转弹丸气动力外弹道计算研究[J]. 南京理工大学学报(自然科学版) ，1985，(S1):277-288.

[18] 顾文彬, 陆鸣, 刘建青, 董勤星, 陈江海, 王振雄. 基于 Kane 方法的火箭抛绳系统飞行动力学模型[J]. 弹道学报，2015，(02):19-23.

[19] Alex Martelli,Anna Ravenscroft,David Ascher. Python Cookbook. 2005

[20] 高双. 高速无人艇的建模与控制仿真[D].哈尔滨工程大学，2007.

[21] 杨炜帆. 水面无人艇的建模与运动特性仿真[D].大连海事大学，2013.

[22] 黄克明,陈群斋. 在线解算外弹道问题探讨[J]. 舰船科学技术，1997，(04):47-50.

[23] 刘文,张为华,周珞晶. 用于自动瞄准的火炮射角快速算法[J]. 火炮发射与控制学报，2003，(03):41-43.

[24] 黄西密. 无人艇建模及操纵运动仿真的研究[D].大连海事大学，2015.

附录（程序源代码）

```
#-*- coding: UTF-8 -*-
from math import *
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sys, time

def search_data_by_distance(distance1):
    for i in range(len(good_data)):
        if (good_data[i,3]-distance1) < 0.4 and good_data[i,3]>distance1:
            return good_data[i]
    print "cannot find good distance by distance"
    sys.exit()
def search_data_by_time(time_temp):
    for i in range(len(good_data)):
        if abs(good_data[i,2]-time_temp)<0.2:
            return good_data[i]
    print "cannot find good data by time"
    sys.exit()

def search_data_by_input(degree):
    for i in range(len(good_data)):
        if (good_data[i,1]==degree):
            return good_data[i]
    print "cannot find good data by degree"
    sys.exit()

def to_pi(degree):
    return degree*pi/180.0
def to_degree(angle_in_pi):
    return angle_in_pi*180.0/pi

def compute_target_position_horizon(target_position,ship_position):
    d_vector=target_position - ship_position
    d_vector_length = sqrt(sum(d_vector**2))
```

```
    ox_vector=np.array([cos(inputdata[3]),sin(inputdata[3])])
    oy_vector=np.array([-sin(inputdata[3]),cos(inputdata[3])])
    x_target_horizon=sum(d_vector*ox_vector)
    y_target_horizon=sum(d_vector*oy_vector)
    tareget_position_horizon=np.array([x_target_horizon,y_target_horizon])
    target_position_angle_horizon = atan(y_target_horizon/(x_target_horizon))
    target_position_angle_horizon=to_degree(target_position_angle_horizon)
    print "target_position_horizon:",tareget_position_horizon
    print "target_position_angle_horizon:",target_position_angle_horizon
    return tareget_position_horizon,target_position_angle_horizon,d_vector_length

def generate_uvw_to_UVW_mat(inputdata):
    uvw_to_UVW=np.zeros((3,3))
    uvw_to_UVW[0][0]=cos(inputdata[4])
    uvw_to_UVW[0][1]=sin(inputdata[5])*sin(inputdata[4])
    uvw_to_UVW[0][2]=cos(inputdata[5])*sin(inputdata[4])
    uvw_to_UVW[1][0]=0
    uvw_to_UVW[1][1]=cos(inputdata[5])
    uvw_to_UVW[1][2]=-sin(inputdata[5])
    uvw_to_UVW[2][0]=-sin(inputdata[4])
    uvw_to_UVW[2][1]=sin(inputdata[5])*cos(inputdata[4])
    uvw_to_UVW[2][2]=cos(inputdata[5])*cos(inputdata[4])
    uvw_to_UVW_mat=np.mat(uvw_to_UVW)
    return uvw_to_UVW_mat

def generate_UVW(inputdata,uvw_to_UVW_mat):
    uvw=np.array([[inputdata[0]],[inputdata[1]],[inputdata[2]]])
    uvw=np.mat(uvw)
    UVW=uvw_to_UVW_mat*uvw
    UVW=UVW.tolist()
    return UVW
def generate_deltaT(divided=8):
    time_min = search_data_by_input(1)[2]
    time_max =search_data_by_input(74)[2]
    delta = (time_max-time_min)/divided
    result = []
    for i in range(divided):
        tempresult = time_min+delta/2+i*delta
        result.append(tempresult)
```

```
    return result
def generate_deltaD(divided=8):
    distance_min = search_data_by_input(1)[3]
    distance_max =search_data_by_input(74)[3]
    delta = (distance_max-distance_min)/divided
    result = []
    for i in range(divided):
        tempresult = distance_min+delta/2+i*delta
        result.append(tempresult)
    return result
def find_index(delta_distance,d_vector_length):
    index = 0
    for i in range(len(delta_distance)):
        if(abs(d_vector_length-delta_distance[i])<abs(d_vector_length-
delta_distance[index])):
            index=i
    return index
def compute_average_deltaT(d_vector_length):
    deltaT = generate_deltaT(8)
    delta_distance = generate_deltaD(8)
    divided_index = find_index(delta_distance, d_vector_length)
    return deltaT[divided_index]

def compute_target_position_horizon_t(UVW,t,target_position_horizon):
    V_target_horizon = np.array([-UVW[0][0], -UVW[1][0]])
    target_position_horizon_t = target_position_horizon + V_target_horizon * t
    return target_position_horizon_t

def generate_angle_speed_transform_mat(inputdata):
    angle_speed_transform=np.zeros((3,3))
    angle_speed_transform[0][0]=1
    angle_speed_transform[0][1]=sin(inputdata[5])*tan(inputdata[4])
    angle_speed_transform[0][2]=cos(inputdata[5])*tan(inputdata[4])
    angle_speed_transform[1][0]=0
    angle_speed_transform[1][1]=cos(inputdata[5])
    angle_speed_transform[1][2]=-sin(inputdata[5])
    angle_speed_transform[2][0]=0
    angle_speed_transform[2][1]=sin(inputdata[5])/cos(inputdata[4])
    angle_speed_transform[2][2]=cos(inputdata[5])/cos(inputdata[4])
```

```
    angle_speed_transform_mat=np.mat(angle_speed_transform)
    return angle_speed_transform_mat

def compute_angle_speed_horizon(inputdata):
    angle_speed_transform_mat = generate_angle_speed_transform_mat(inputdata)
    pqr = np.mat([[inputdata[6]], [inputdata[7]], [inputdata[8]]])
    angle_speed_horizon = (angle_speed_transform_mat *
pqr).transpose().tolist()[0]
    # print angle_speed_horizon
    return  angle_speed_horizon

def set_V_by_angle_speed_horizon(V_original_speed,inputdata,angle_speed_horizon):
    V_up = 1 * angle_speed_horizon[0] * cos(inputdata[5])
    V_right = 1 * angle_speed_horizon[0] * sin(inputdata[5])
    if (inputdata[5] < 0):
        V_up = -V_up
    V_original_speed[1] += V_right
    V_original_speed[2] += V_up

    V_up = 2 * angle_speed_horizon[1] * cos(inputdata[4])
    V_x = 2 * angle_speed_horizon[1] * sin(inputdata[4])
    if (inputdata[4] < 0):
        V_up = -V_up
    V_original_speed[0] += V_x
    V_original_speed[2] += V_up
    return V_original_speed

def transform_to_aim(angle_to_negative_x,dataxy):
    #angle_to_negative_x = -angle_to_negative_x
    angle_temp = pi-angle_to_negative_x
    transform_to_aim_mat=np.mat([[cos(angle_temp),-
sin(angle_temp)],[sin(angle_temp),cos(angle_temp)]])
    dataxy_array=np.array([[dataxy[0]],[dataxy[1]]])
    dataxy_mat = np.mat(dataxy_array)
    new_dataxy_mat = transform_to_aim_mat*dataxy_mat
    new_dataxy = new_dataxy_mat.tolist()
    new_dataxy = [new_dataxy[0][0],new_dataxy[1][0]]
    return new_dataxy
def compute_V_updated_aim(inputdata,angle1,angle_speed_horizon):
```

```
    V_original_speed = [0, 0, 0]
    V_updated = set_V_by_angle_speed_horizon(V_original_speed, inputdata,
angle_speed_horizon)
    V_updated_aim = transform_to_aim(angle1, V_updated[:2])
    V_updated_aim.append(V_updated[2])
    return V_updated_aim  # have taken angle_speed into consideration


def compute_delta_V0(V_updated_aim,alpha_temp):
    return
(V_updated_aim[0]*cos(to_pi(alpha_temp))+V_updated_aim[2]*sin(to_pi(alpha_temp)))
def compute_delta_Theta(V_updated_aim,alpha_temp,V0=30):
    temp = -
V_updated_aim[0]*sin(to_pi(alpha_temp))+V_updated_aim[2]*cos(to_pi(alpha_temp))
    return (temp*1.0+0.05)/V0   #v2 =0.05
def compute_V0_derivative_X(V0):
    return 1.4  # for calculating
def compute_Theta_derivative_X(Theta):
    number1 = search_data_by_input(Theta-0.5)[3]
    number2 = search_data_by_input(Theta+0.5)[3]
    result  = number2-number1
    return result

def compute_pitch_degree(V_updated_aim,X_target_position_aim_t):
    alpha_temp = search_data_by_distance(X_target_position_aim_t)[1]
    #print "original degree：ê?",alpha_temp
    counttime=0
    while(True):
        flight_distance = search_data_by_input(alpha_temp)[3]
        delta_V0 = compute_delta_V0(V_updated_aim,alpha_temp)
        delta_Theta=compute_delta_Theta(V_updated_aim,alpha_temp,V0=30)
        V0_derivative_X = compute_V0_derivative_X(V0=30)
        Theta_derivative_X = compute_Theta_derivative_X(alpha_temp)
        X_real = flight_distance +
V0_derivative_X*delta_V0+Theta_derivative_X*delta_Theta
        if(counttime>15 or abs(X_real-X_target_position_aim_t)<0.2):
            pitch_degree = alpha_temp
            error_Xoa=X_real-X_target_position_aim_t
            #print pitch_degree
```

```
            #print "final error: ",abs(X_real-X_target_position_aim_t)
            return pitch_degree
        if (X_real < X_target_position_aim_t):
            alpha_temp = alpha_temp + 0.5
        else:
            alpha_temp = alpha_temp - 0.5
            # print alpha_temp
        counttime += 1
        #print "temp error:",X_real-X_target_position_aim_t
          # the wanted pitch degree
def compute_real_flight_location(error_Xoa,real_target_position_aim_t):
    result = np.array(real_target_position_aim_t).copy()
    result[0] +=error_Xoa
    return result
def compute_error(real_flight_location,target_position_aim_t,d_vector_length):
    X_error=real_flight_location[0]-target_position_aim_t[0]
    Y_error=real_flight_location[1]-target_position_aim_t[1]
    absolute_error=sqrt(X_error*X_error+Y_error*Y_error)
    related_error=absolute_error/d_vector_length
    print target_position_aim_t
    print real_flight_location
    print "X_error",target_position_aim_t[0]-real_flight_location[0]
    print "Y_error",target_position_aim_t[1]-real_flight_location[1]

    return absolute_error,related_error
def compute_deltaX_wind(pitch_degree_temp,V_wind_zong):
    tempT=search_data_by_input(pitch_degree_temp)[2]
    V0_derivative_X = compute_V0_derivative_X(V0=30)
    Theta_derivative_X = compute_Theta_derivative_X(pitch_degree_temp)
    deltaX_wind = V_wind_zong*(tempT-
V0_derivative_X*cos(to_pi(pitch_degree_temp))+Theta_derivative_X*sin(to_pi(pitch_d
egree_temp))/30)
    return deltaX_wind

def compute_pitch_degree_updated(X_target_position_aim_t, V_wind,V_updated_aim):

    distance_without_wind = X_target_position_aim_t
    V_wind_zong=V_wind[0]
```

```
original_pitch_degree=compute_pitch_degree(V_updated_aim,distance_without_wind)

    print "original pitch angle",original_pitch_degree
    while(True):

pitch_degree_temp=compute_pitch_degree(V_updated_aim,distance_without_wind)
        deltaX_wind=compute_deltaX_wind(pitch_degree_temp,V_wind_zong)
        X_real=distance_without_wind+deltaX_wind
        error = X_real-X_target_position_aim_t
        print "temp error:",abs(error)
        if(abs(error)<0.2):
            print "pitch_degree_update:",pitch_degree_temp
            print "final error:",error
            return pitch_degree_temp
        if(error>0):
            distance_without_wind -=0.5
        else:
            distance_without_wind +=0.5

def
compute_V_horizon_aim(V_wind,pitch_degree,target_position_aim_t,V_updated_aim,flig
ht_time):

    v1=V_wind[1]-(V_wind[1]*flight_time-
target_position_aim_t[1])*30*cos(to_pi(pitch_degree))/target_position_aim_t[0]
    return v1

if __name__== '__main__':
    good_data = pd.read_excel("test7.xlsx").values
    start_time = time.time()
    inputdata = [3, 2, 1, (115 * pi / 180.0), (20 * pi / 180.0), (-18 * pi /
180.0), 0.02, 0.6,
                 0.7]  # u,v,w,azimuth,pitching angle,rolling angle,azimuth
speed,pitching speed,rolling speed)

    target_position =np.array([40,20])   #x1,y1
    ship_position = np.array([20,40])   #x2,y2
    target_position_horizon, target_position_angle_horizon_pi,d_vector_length=
compute_target_position_horizon(target_position,ship_position)
```

```
    uvw_to_UVW_mat = generate_uvw_to_UVW_mat(inputdata)
    UVW = generate_UVW(inputdata,uvw_to_UVW_mat)

    average_deltaT = compute_average_deltaT(d_vector_length)

    target_position_horizon_t =
compute_target_position_horizon_t(UVW,average_deltaT,target_position_horizon)
    print "target_position_horizon_t:",target_position_horizon_t

target_position_angle_horizon_t=to_degree(atan(target_position_horizon_t[1]/target
_position_horizon_t[0]))
    print "target_position_angle_horizon_t:",target_position_angle_horizon_t

    angle1_pi = to_pi(target_position_angle_horizon_t*(5.0/6))
    angle2_pi = target_position_angle_horizon_t-angle1_pi
    print "angle1",to_degree(angle1_pi)
    angle_speed_horizon = compute_angle_speed_horizon(inputdata)
    target_position_aim_t=transform_to_aim(angle1_pi,target_position_horizon_t)
    print "target_position_aim_t",target_position_aim_t
    X_target_position_aim_t = target_position_aim_t[0]

    # have taken angle_speed into consideration
    V_updated_aim = compute_V_updated_aim(inputdata,angle1_pi,angle_speed_horizon)
    print "V_updated_aim:",V_updated_aim

    K=4
    w_abs=K+0.17*K*K
    w_angle=80
    V_wind_original=[-cos(to_pi(w_angle))*w_abs,sin(to_pi(w_angle))*w_abs]
    print "V_wind_original:",V_wind_original
    UV=np.array([UVW[0][0], UVW[1][0]])
    V_s=[UV[0]*cos(inputdata[3])-
UV[1]*sin(inputdata[3]),UV[0]*sin(inputdata[3])+UV[1]*cos(inputdata[3])]

    w_r=np.array(V_wind_original)-np.array(V_s)
    print "wr:",w_r
    ox_vector=np.array([cos(inputdata[3]),sin(inputdata[3])])
    oy_vector=np.array([-sin(inputdata[3]),cos(inputdata[3])])
```

```
    w_hx=np.dot(w_r,ox_vector)
    w_hy=np.dot(w_r,oy_vector)
    w_h=np.array([w_hx,w_hy])
    print "w_h",w_h
    w_a=transform_to_aim(angle1_pi, w_h)
    print "w_a:",w_a

    #to compute pitch degree and pitch line speed
    V_wind=w_a
    pitch_degree= compute_pitch_degree_updated(X_target_position_aim_t,
V_wind,V_updated_aim)
    flight_time = search_data_by_input(pitch_degree)[2]
    V_horizon_aim =
compute_V_horizon_aim(V_wind,pitch_degree,target_position_aim_t,V_updated_aim,flig
ht_time)
    print "V_updated_aim:",V_updated_aim
    print "V_horizon_aim:",V_horizon_aim
    print "real time",flight_time
    print "averate_deltaT",average_deltaT
    real_target_position_horizon_t = compute_target_position_horizon_t(UVW,
flight_time, target_position_horizon)
    print
"real_target_position_horizon_t(people) :",real_target_position_horizon_t

real_target_position_aim_t=transform_to_aim(angle1_pi,real_target_position_horizon
_t)
    print "real_target_position_aim_t(peole):",real_target_position_aim_t
    real_flight_location = target_position_aim_t
    print "real_flight_location:",real_flight_location

absolute_error,related_error=compute_error(real_flight_location,real_target_positi
on_aim_t,d_vector_length)
    print "absolute_error",absolute_error
    print "related_error",related_error
    print 'time ', time.time() - start_time, 's'
def computing(input):  # input:[V0,Alpha0]
    import scipy.linalg
    from math import *
    import numpy as np
```

```
import matplotlib.pyplot as plt

# old_settings = np.seterr(all='ignore')
DetT = 0.001  # the time unit
V0 = input[0]
alpha0 = (input[1] * pi / 180.0)
maxtimes = 50000
maxnodes = 10000

J = 5.12 * 0.001  # moment of inertia of the head section
M0 = 1.5  # the mass of the head section
g = 9.8  # the acceleration of gravity
di = 0.05
L0 = 0.05  # the distance from the center to the tail of the head section
Mi = di * 4.56 /1000
W0 = 0
intensity_of_line = 4.56 / 1000  # kg/s
countTime = 0

Beta = np.zeros(maxnodes,dtype=np.float64)
Beta[0] =Beta[1]=Beta[2]=Beta[3] = alpha0
Alpha = np.zeros(maxnodes,dtype=np.float64)
Alpha[0] = Alpha[1] = Alpha[2] =alpha0
V = np.zeros(maxnodes,dtype=np.float64)
V[0] = V[1] =V[2] =V0

X = np.zeros(maxnodes,dtype=np.float64)
X[0] = di * cos(Beta[2]) + di * cos(Beta[1]) + L0 * cos(Beta[0])
X[1] = di * cos(Beta[2])
Y = np.zeros(maxnodes,dtype=np.float64)
Y[0] = di * sin(Beta[2]) + di * sin(Beta[1]) + L0 * sin(Beta[0])
Y[1] =  di * sin(Beta[2])

A = np.zeros((2, 2), dtype=np.float64)  # come with positionAX,positionAY
B = np.zeros((2, 1), dtype=np.float64)  # come with positionB
#  T = np.mat(np.zeros((2, 1)))    # come with positionT
positionT = 0

record_x = np.zeros(maxtimes)
```

```
    record_y = np.zeros(maxtimes)
    record_t = np.zeros(maxtimes)
    record_T = np.zeros(maxtimes)
    N = 2  # the number of nodes.
    while 1:
        if Y[0] < 0:
            break
        A[0][0] = -(2 * M0 * J + 2 * Mi * J + M0 * Mi * L0 * L0 * (1 - cos(2 *
Beta[0] - 2 * Beta[1]))) * di / (
        2 * M0 * Mi * J)  # A11
        A[0][1] = di * cos(Beta[2] - Beta[1]) / Mi  # A12
        B[0][0] = -(
        V[0] * V[0] + V[1] * V[1] - 2 * V[0] * V[1] * cos(Alpha[1] - Alpha[0]) +
di * L0 * W0 * W0 * cos(
            Beta[0] - Beta[1]))  # B1
        for i in range(2, N + 1, 1):
            positionT = i - 1  # ==1
            A[positionT][positionT - 1] = di * cos(Beta[i] - Beta[i - 1]) / Mi  #
A(i,i-1)
            A[positionT][positionT] = -2 * Mi * di / (Mi * Mi)  # A(i,i)
            B[i - 1][0] = di * (
            V[i - 1] * V[i - 1] + V[i] * V[i] - 2 * V[i - 1] * V[i] * cos(Alpha[i -
1] - Alpha[i])) / Mi  # Bi
            if i != N:
                A[positionT][positionT + 1] = di * cos(Beta[i] - Beta[i + 1]) / Mi
# A(i,i+1)
            if i == N:
                B[i - 1][0] -= intensity_of_line * V[i] * V[i] * di * cos(
                    Beta[i] - Beta[i + 1]) / Mi  # Bn = Bi - A(n, n+1) * Tn
attention Tn

        B[B>1e-16] *=-1
        tempT = scipy.linalg.solve(A, B)
        T = np.zeros(N+1, dtype=np.float64)
        for i in range(0, N, 1):
            T[i]=tempT[i][0]
        T[N]=intensity_of_line * V[N] * V[N] # attention Tn
        T[np.isnan(T)] = 1e-10
        T[T < 0] = 1e-10
```

```
        T[T>1e-16] *=0.001
        T1 = T[0]
        T2 = T[1]

        V[0] += DetT * (-T1 * cos(Beta[1] - Alpha[0]) - M0 * g * sin(Alpha[0])) /
M0
        Alpha[0] += DetT * (-T1 * sin(Beta[1] - Alpha[0]) - M0 * g *
cos(Alpha[0])) / (M0 * V[0])
        W0 += DetT * (T1 *sin(Beta[1] - Beta[0]) * L0) / J
        Beta[0] += DetT * W0
        X[0] += DetT * V[0] * cos(Alpha[0])
        Y[0] += DetT * V[0] * sin(Alpha[0])

        record_x[countTime]=X[0]
        record_y[countTime]=Y[0]
        record_T[countTime]=T[0]
        record_t[countTime] = countTime
        countTime += 1

        for i in range(1, N + 1, 1):
            positionT = i - 1
            V[i] += DetT * (T[positionT] * cos(Beta[i] - Alpha[i]) - T[positionT +
1] * cos(
                Beta[i + 1] - Alpha[i]) - Mi * g * sin(Alpha[i])) / Mi
            # V[i] = round(V[i], 10)
            Alpha[i] += DetT * (T[positionT] * sin(Beta[i] - Alpha[i]) -
T[positionT + 1] * sin(
                Beta[i + 1] - Alpha[i]) - Mi * g * cos(Alpha[i])) / (Mi * (V[i] +
1e-8))
            #Alpha[i] = round(Alpha[i], 10)
            X[i] += DetT * V[i] * cos(Alpha[i])
            Y[i] += DetT * V[i] *sin(Alpha[i])
            Beta[i] = atan((Y[i - 1] - Y[i]) / (X[i - 1] - X[i]))

        #Beta[Beta<0] = 1e-10

        for i in range(1,1,1):
            X[i] = X[i-1] -di*cos(Beta[i])
            Y[i] = Y[i-1] -di*sin(Beta[i])
```

```
        if (X[N]*X[N]+Y[N]*Y[N]) >= (di*di):
            tempA1 = np.zeros(N)
            tempA2 = np.column_stack((A, tempA1))
            tempA3 = np.zeros(N + 1)
            A = np.row_stack((tempA2, tempA3))
            tempB1 = np.zeros(1)
            B = np.row_stack((B, tempB1))

            N += 1
    #plt.plot(subplot='True')
    #import_xy = np.load('test_first11.npz')
    #plt.plot(record_x[0:countTime], record_y[0:countTime], import_xy['a'],
import_xy['b'])
    #record_t[countTime]=record_t[countTime-1]
    #plt.plot(record_t[0:countTime+1], record_T[0:countTime+1])
    #plt.scatter(X[0:countTime],Y[0:countTime])
    #plt.grid(True)
    #plt.show()
    return [countTime*DetT,X[0]]

inputs = []
if False:
    V0 =np.arange(57,65,0.1)
    Alpha = np.arange(15,25,0.1)
    for element1 in V0:
        for element2 in Alpha:
            inputs.append([element1,element2])

if True:  #V0 still Alpha change
    V0 = 40
    Alpha = np.arange(15, 25, 0.5)
    for element2 in Alpha:
        inputs.append([V0, element2])

if False:
    V0 = np.arange(60,65,0.5)
    Alpha = 20
    for element1 in V0:
```

```
        inputs.append([element1, Alpha])

output_1 = []
# tuple of all parallel python servers to connect with
ppservers = ()
#ppservers = ("10.0.0.1",)
start_time = time.time()

for input in inputs:
    print into_function.computing(input)
for input in inputs:
    print into_function.computing(input)
print 'time ', time.time() - start_time, 's'

if len(sys.argv) > 1:
    ncpus = int(sys.argv[1])
    # Creates jobserver with ncpus workers
    job_server = pp.Server(ncpus, ppservers=ppservers)
else:
    # Creates jobserver with automatically detected number of workers
    job_server = pp.Server(ppservers=ppservers)
print "pp ", job_server.get_ncpus(), "workers"
start_time = time.time()
jobs = [(input, job_server.submit(into_function.computing,(input,))) for input in
inputs]
for input, job in jobs:
    tempresult = job()
    #print "answer is" ,tempresult
    output_1.append([input[0],input[1],tempresult[0],tempresult[1]])
print "time consumed: ", time.time() - start_time, "s"
job_server.print_stats()
```

致谢

岁月如梭，这四年的本科生活让我学会了很多东西，使我受益匪浅。回首这大半年来的撰写本科毕业论文的时间，从文献阅读到试验平台搭建、撰写论文、修改论文的过程中，我得到了许多老师和同学们的指导和关怀，谨向这些年来给予我关心的良师益友们致以最真诚的谢意！

我要感谢我的导师顾临怡教授。顾老师为人谦和，育人细心。在论文的选题、查阅资料和论文撰写过程中，都得到了顾老师的悉心指导。每当我有所疑问或对课题感到迷茫时，顾老师都会不厌其烦的认真指点我。在我本科学习生涯中，顾老师严谨的教学风格将影响和激励我一生，他对我的关心和教诲我将永远铭记。

我要感谢我的朋友同学以及家人，本论文的完成离不开他们的关心与帮助。感谢实验室中所有的师兄师姐们和同学金健安、鲁晨阳，在我的论文撰写过程中，他们给予我很多的帮助和支持。感谢我的父母，他们的关心和鼓励让我能够专心地致力于本课题的研究学习。

即将告别我的本科学习，但我相信，本科毕业将会是我人生的另一个起点。感谢我的母校浙江大学，让我有了更加广阔的舞台继续追逐自己的目标和理想，我将带着这份美好的回忆，走入社会继续努力奋斗。

毕业设计（论文）考核

一、指导教师对毕业设计（论文）的评语：

指导教师(签名)

年月日

二、答辩小组对毕业设计（论文）的答辩评语及总评成绩：

成绩比例	文献综述（10%）	开题报告（20%）	外文翻译（10%）	毕业设计（论文）质量及答（60%）	总评成绩
分值					

答辩小组负责人（签名）

年月日

浙江大学本科生毕业设计（论文）存档资料检查表

（此表请装订在毕业设计（论文）末页，由指导教师在毕业设计（论文）答辩结束后填写）

检查内容		检查结果（打√、X）
毕业设计	1．装订（封面白色铜版纸，A4）	
	2．毕业设计（论文）诚信承诺书(签名)	
	3. 任务书（题目、要求、工作进度表）	
	4．中、英文摘要	
	5．目录（要标注页码）	
	6．正文	
	7．附件 1：图纸、程序	
	8．附件 2：毕业设计期间取得的成果	
	9. 参考文献	
	10.毕业设计指导教师评语、答辩评语和成绩	
开题报告	11．装订（文献综述、开题报告、外文翻译、外文原稿）	
	12．文献综述、开题报告的成绩、评语	
其它	13．毕业设计（论文）专家评阅意见（2 份）	
	14．毕业设计（论文）答辩记录表（1 份）	

注：“毕业设计（论文）专家评阅意见”、“毕业设计（论文）答辩记录表”检查后装入“毕业设计档案袋”，直接归档，不装订。

检查人（签名）：检查日期：