软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022, 33(6): 2150–2171 [doi: 10.13328/j.cnki.jos.006575]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

多旋翼飞控推进子系统的 Coq 形式化验证*

石正璞, 崔 敏, 谢果君, 陈 钢

(南京航空航天大学 计算机科学与技术学院, 江苏 南京 211106)

通讯作者: 陈钢, E-mail: gangchensh@qq.com

摘 要: 飞行器需要高可靠的飞行控制系统软件(飞控)来控制其运行. 在传统开发模式下, 先由人工将领域知识描述为自然语言形式的模型, 再根据模型手动编写代码, 然后使用软件测试技术来排除软件错误, 这种模式由于人工易出错、自然语言存在二义性、测试技术的不完备性, 导致难以构建出高可靠的飞控软件. 基于形式验证技术的新型软件开发方法可从多方面提高飞控系统的可靠性. 使用 Coq 定理证明器对全权提出的多旋翼飞控推进子系统进行了完整的形式验证, 生成了一个可用的高可靠函数式软件库. 主要工作有: 首先将领域知识整理为具有层次结构以适合进行形式验证的文档, 分离了基本函数和复合函数, 并提出最简形式函数概念; 再根据该文档进行形式化描述, 定义常量、变量、基本函数、复合函数、最简形式函数和公理等; 其次对各类导出函数的推导正确性建立为引理并予以证明; 再次对多旋翼最长悬停时间等实际问题给出了求解算法; 最后利用 Coq 程序抽取功能生成了 OCaml 语言的函数式软件库. 后续将对飞控更多子系统进行基于形式验证的开发, 并最终建立完整的经形式化验证的高可靠飞控系统.

关键词: 形式化验证; 定理证明; 多旋翼飞行器; 飞行控制系统; 推进系统; Coq; OCaml
中图法分类号: TP311

中文引用格式: 石正璞, 崔敏, 谢果君, 陈钢. 多旋翼飞控推进子系统的 Coq 形式化验证. 软件学报, 2022, 33(6): 2150–2171. http://www.jos.org.cn/1000-9825/6575.htm

英文引用格式: Shi ZP, Cui M, Xie GJ, Chen G. Coq Formalization of Propulsion Subsystem of Flight Control System for Multicopter. Ruan Jian Xue Bao/Journal of Software, 2022, 33(6): 2150–2171 (in Chinese). http://www.jos.org.cn/1000-9825/6575.htm

Coq Formalization of Propulsion Subsystem of Flight Control System for Multicopter

SHI Zheng-Pu, CUI Min, XIE Guo-Jun, CHEN Gang

(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)

Abstract: A highly reliable flight control system (FCS) is a necessary prerequisite for the reliable operation of an aircraft. Under the traditional development approach, the domain knowledge is first modeled by the human in the form of natural language, and then code is written by humans according to the model, and finally, the software defects are eliminated by using testing technology. The approach fails to build reliable FCS, because of human error, natural language ambiguity, and incompleteness of test techniques. A novel development approach based on formal verification technology could improve the reliability of FCS from many aspects. This paper presents a formal design and verification method for multicopter propulsion subsystem based on Coq and generated a usable and highly reliable functional software library. The main work of this study includes: the domain knowledge is organized into a hierarchical document suitable for formal verification, the basic functions, and composite functions are separated, and the concept of the simplest form of function (SFF) is proposed; formalize the system in Coq according to this document, defining constants, variables, basic functions, composite functions, SFF, axioms, etc.; the correctness of the derivation of all kinds of composite functions is expressed as lemmas and be proved; the algorithm for solving practical problems such as the longest hover time of multicopter is given; and a functional software library is

*本文由“定理证明理论与应用”专题特约编辑曹钦翔副教授、詹博华副研究员、赵永望教授推荐.
收稿时间: 2021-09-03; 修改时间: 2021-10-14, 2022-02-07, 2022-02-25; 采用时间: 2022-03-21

generated using OCaml language by COQ program extraction ability. In the future, more subsystems of FCS will be developed based on formal verification, and finally, a complete and highly reliable FCS with formal verification will be established.

Key words: formal verification; theorem proving; multicopter; flight control system; propulsion system; Coq; OCaml

飞行器是广泛存在于航空航天领域的一类能够半自主或全自主运行的设备, 包括航空器、航天器、制导武器、运载火箭等, 各类飞行器都需要飞行控制系统(飞控)控制其运行[1,2]. 飞控的可靠性历来是一个研究热点, 人们迫切需要高可靠的飞控, 但现实的飞控却难以可靠, 并且飞控缺陷常带来巨大的生命及财产损失. 例如, 2018 年 10 月 29 日印尼狮航 737 MAX 客机坠毁致全部人员遇难, 2019 年 3 月 10 日埃塞俄比亚航空 737 MAX 客机坠毁致全部人员遇难, 后经证实该机型存在设计缺陷[3]. 实现高可靠的飞控非常困难. 首先飞控软件规模庞大, 其次飞控系统非常复杂, 其设计依赖于多学科多领域的知识, 以及各种实验和经验数据[1,2,4,5]. 尽管飞控研发投入了大量人力物力, 但仍难以实现完全可靠.

经分析, 传统的飞控开发模式在保障系统可靠性方面存在一定的弊端. 传统的飞控开发模式如图 1 所示, 首先由飞控领域专家对飞控知识进行抽象和建模, 给出自然语言描述的软件开发参考文档, 然后由软件开发人员根据该文档手工编写代码实现软件, 最后由测试人员使用测试技术找出软件中的错误. 这种模式的弊端有多个方面: 第一, 飞控领域知识建模是人工进行的, 包括各种定理使用、公式推导等, 人工推导的失误和不严谨性将会在飞控中留下设计缺陷; 第二, 人工建立的模型是由自然语言描述可能存在歧义, 软件人员可能因理解偏差而编写出不符合原始设计的软件; 第三, 根据模型手工编写代码可能引入错误; 第四, 传统软件测试技术不能证明软件没有缺陷. 总之, 在传统开发模式下, 既不能确保软件模型正确, 也不能确保软件与模型一致, 更不能证明软件没有缺陷[6].

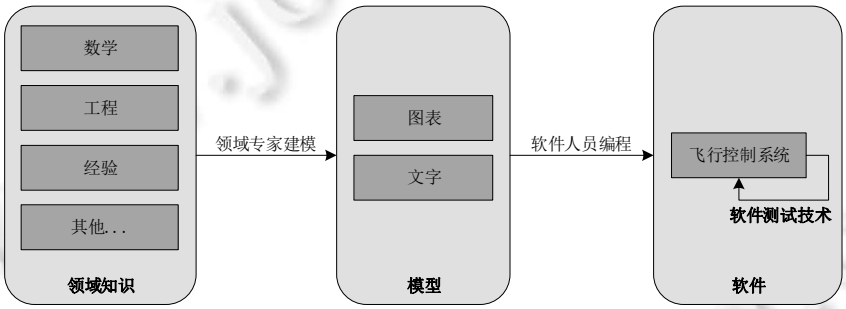


图 1 传统的飞控软件开发模式

形式化方法是基于严格的数学基础, 对计算机软(硬)件系统进行形式规约、开发和验证的技术. 形式化方法在处理软件开发复杂性和提高软件可靠性方面已显示出无可取代的潜力. 交互式定理证明技术是形式化方法中的一个技术分支, 有很强的表达能力, 可以准确表达系统的形式规约并验证复杂的系统性质[6]. 将交互式定理证明技术应用于飞控开发, 把飞控中的计算公式表示为函数, 公式推导过程的正确性表示为引理并予以证明, 将能够消除一大类潜在的错误, 提高飞控设计的可靠性[7,8].

基于定理证明的飞控开发模式, 如图 2 所示. 首先, 领域专家与形式化人员共同建立形式化的模型, 形成层次结构清晰以适合于形式验证的文档和形式化描述脚本, 该模型包括了飞控中的概念、运算、约束等各种飞控知识细节, 并且对人和机器都是友好的. 对人而言, 可以轻松地理解和修改模型; 对机器而言, 模型中的所有内容都是带有类型的数学对象, 适合于机器验证. 然后, 形式化人员完成所有性质的证明以确保形式化模型正确. 其次, 使用代码生成技术自动生成程序, 从而得到可靠的飞控. 该开发模式的优势是: 将系统可靠性问题提前到领域知识的需求分析阶段, 由领域专家和形式验证人员共同建立系统模型可以确保把模糊的飞控知识转换为严谨的形式化描述, 再由机器验证所有给定约束命题的正确性后可以证明系统模型的正确, 然后由代码生成工具来自动生成程序, 也避免了人工编码的错误, 这样开发的程序天生具有可靠性, 满足给定

的形式规约. 该开发模式的缺点是: 初始阶段的开发周期较长. 但由于形式化的知识具有可重用性, 且能减少测试工作量, 长期来看是缩短了开发周期, 降低了开发成本. 在安全关键领域以及业务知识较为稳定的场景下采用这种开发模式是必要且合理的选择.

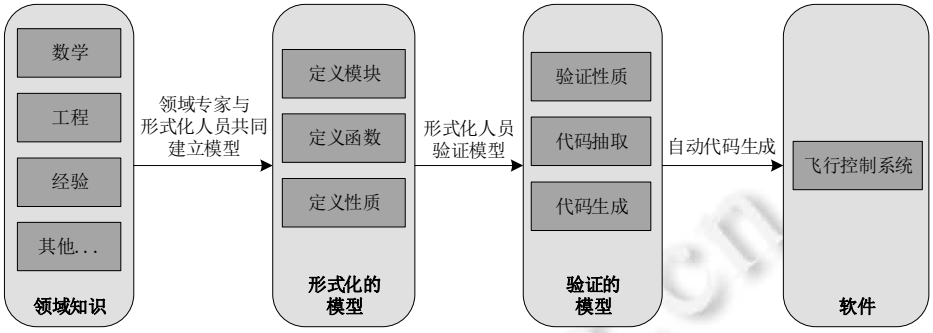


图 2 基于定理证明的飞控开发模式

现在是智能软件时代, 与传统软件相比有两个特殊性, 一个是大量引入连续性数学(微积分等), 另一个是多学科交融(材料、机械、电子和控制等). 这两个新特点对传统软件提出了新的挑战, 从需求描述到软件设计都带来一系列问题, 也就是说, 在算法描述和设计过程中需要使用数学推导, 并且使用大量不熟悉的专业知识. 为适应这方面的需求, 我们进行了工程数学形式化的研究工作. 本文以多旋翼无人机飞控的推进子系统为研究对象, 使用 Coq 定理证明器进行系统建模和形式验证, 并生成软件库. 主要创新之处在于: 将文献[1]中推进子系统的工程理论转换成了结构清晰的形式化文本, 封装了复杂的工程知识, 为形式验证和软件开发提供了统一的接口; 实现了文献[1]中推进子系统的完整形式化并给出了可用的函数式软件库; 提出了推进子系统的变量关系图, 直观地显示了变量间的拓扑关系; 提出了数学上等价的最简形式函数的概念, 它能辅助公式推导并降低计算成本.

本文第 1 节是相关工作. 第 2 节是多旋翼推进子系统的介绍. 第 3 节是 Coq 定理证明器的介绍. 第 4 节是推进子系统的形式化验证, 其中, 第 4.1 节是构建适合形式验证的文档; 第 4.2 节是常量的定义; 第 4.3 节是变量的定义; 第 4.4 节是函数的定义; 第 4.5 节是引理的定义; 第 4.6 节是引理证明; 第 4.7 节是典型问题求解示例; 第 4.8 节是抽取 OCaml 代码示例. 第 5 节是总结.

1 相关工作

本研究工作受美国 DARPA (Defense Advanced Research Projects Agency, 国防高级研究计划局)的 HACMS (high assurance cyber military systems, 高可信军事网络系统)项目所启发[9]. 该项目在无人机开发中大量使用了形式化方法技术, 目标是确保无人机软件的高安全性, 消除能让黑客侵入计算机系统的软件缺陷. 该项目的目标不仅是提供安全性, 而且是提供可验证的安全性. 该项目在系统需求分析、系统架构设计、领域专用语言等层面都使用了形式化技术, 最终开发出了被誉为“地球上最安全的无人机”. 本文受 HACMS 项目启发, 尝试对多旋翼无人机的系统需求和系统架构进行类似的研究工作, 专注于建立形式化的飞控模型. 本文与 HACMS 项目相比, 更重视数学算法推导的正确性研究. 飞控中数学推导非常复杂, 我们认为在算法层进行形式化验证是有必要的. 比如, 对于 737 MAX 事故, 这不仅仅是软件实现的问题, 更重要的是算法设计出现了问题.

全权等人的专著《多旋翼飞行器设计与控制》[1,2]全面而系统地介绍了多旋翼飞行器系统. 史东杰等人的电动多旋翼飞行器的实用性能评估方法[10], 提出了多旋翼系统的实用性能估算方法和具体的测试方法. 以上文献包含了飞控推进子系统的一种典型模型, 但它们是用自然语言描述的传统工程类文献, 其工程知识的逻辑关系是否有矛盾, 其数学表述和推理是否有错误等难以判断. 本文参考了这些文献中的工程技术细节, 并

使用定理证明器来构建飞控推进子系统的形式化模型, 力图消除潜在的推导错误. 本文将工程知识的基础部分和衍生部分分离, 基础部分仍然需要人工验证, 但工作量已经大大减少, 衍生部分依靠定理证明可以保证其可靠性.

用于 Coq 实数分析的 Coquelicot 库[11], 包含大量实数理论; 用于 Coq 数学基础研究的 MathComp 库[12], 包含大量现代数学基础理论. 这些库对于控制系统的形式化研究是重要的支撑. Sa'ed Abed 等人使用 HOL 定理证明器进行无人机形式化分析[13], 形式化地分析了无人机的连续动力学方程. 其主要贡献包括: (1) 用 HOL Light 定理证明器进行复数矩阵的形式化; (2) 扩展了拉普拉斯变换的形式化, 为分析多输入多输出系统提供了支持; (3) 用 HOL Light 对无人机的连续动力学方程进行形式化分析, 包括坐标系统中旋转矩阵、动力学方程的变换等. 以上文献从不同方面研究数学理论和工程应用的形式化, 但只关注抽象的数学性质或工程问题的某一方面, 并未对某一工程系统进行完整的形式化. 本文更关注如何从系统需求开始形式化地构建一个完整的系统, 相比而言系统性更强. 本文已将文献[1]中推进子系统所有的基本概念、运算、性质完整地进行了形式化(注: 有些基础设施仍需要更深入的建模和分析, 本阶段暂未深究), 并提供了典型应用场景如何复用基础设施进行算法设计的样例, 可以解决推进系统内部的大量问题.

Vernaelen 等人的基于 IDP (imperative declarative programming)的自主无人机检查任务行为形式化验证[14], 使用了有界模型检查(BMC)和不变式检查(IC), 对无人机的飞行高度、电池能量等物理量以及几个安全性质进行了建模和验证, 属于模型检查技术. 但模型检查技术存在固有的状态爆炸问题, 难以验证复杂的系统性质. 本文使用定理证明技术对飞控推进子系统中数学推导正确性进行了验证, 能够对复杂性质进行验证.

Pollien 等人使用 Frama-C 进行无人机自驾仪数学库的验证[15]. 这项研究工作是从源代码的角度利用 Frama-C 框架对自驾仪开源项目 Paparazzi 的数学库进行形式验证, 主要关注飞控程序运行时错误的排查, 以及一些特定函数数学性质的验证(比如四元数与旋转矩阵互相转换函数的互逆性). 飞控是由复杂工程理论构建而成, 仅通过代码难以反向重建飞控设计中的复杂性质以及验证飞控程序设计是否符合飞控理论. 所以该项研究具有一定的局限性, 不能对飞控设计中的各类性质进行全面验证. 相比而言, 本文方法可以更加全面地验证飞控的复杂性质.

马振威等人的基于记录的矩阵形式化[16], 为在 Coq 中进行矩阵的形式化提供了相对简洁的实现方案, 同时保有程序抽取能力. 麻莹莹等人的基于 Coq 的分块矩阵运算的形式化[17], 进一步丰富了基于记录的矩阵实现方法, 并首次实现了分块矩阵的形式化. 由于矩阵是广泛应用于控制系统中的一种基本数学工具, 这些研究为飞控形式化打下了一定的基础. 汪一飞等人的 Laplace 变换在 Coq 中的形式化及其在飞控系统验证中的应用[18,19], 在 Coq 中形式化了 Laplace 变换的定义和主要性质, 并对飞行控制系统、机器人控制系统中的一些传递函数的推导进行了形式化的描述和验证. 由于 Laplace 变换是用于求解线性微分方程的一个关键数学工具, 也是飞控算法在内的许多控制系统算法设计的基础工具, 因此该研究为飞控系统运动学方程的形式验证提供了一定的基础. 马振威等人在 Coq 中进行坐标转换推导和验证[20], 将飞控中常用的 5 种坐标系的相互转换进行了正确性验证. 坐标转换正是在飞行控制等研究领域的重要组成部分, 该研究为飞控的姿态表示形式验证工作提供了一定的基础. 以上这些研究工作, 都是从不同方面来逐步构建形式化的工程数学库, 本文是这些研究工作的延续和积累.

2 多旋翼的推进子系统

多旋翼飞行器推进子系统(propulsion sub-system, PSS)是指以电池为动力并由电调驱动电机带动螺旋桨旋转产生动力以使飞行器飞行的装置, 其配置决定了飞行器最终的性能指标. 推进子系统的典型配置包括螺旋桨、无刷直流电机、电子调速器和电池, 典型的性能指标包括最长悬停时间、最大负载、最大俯仰角、最远飞行距离以及系统效率等. 推进子系统各部件选型的不同会对多旋翼的整体性能产生非常复杂的影响[1].

在多旋翼推进子系统的设计中要解决两类问题. 第 1 类是根据设计参数求解性能指标, 称为正向求解问题. 第 2 类是根据性能指标求解设计参数, 称为逆向求解问题. 根据本文给出的变量关系图(如图 3 所示), 以

及各类函数定义, 可以求解所有第 1 类问题. 第 2 类问题本身比较复杂, 但使用这些函数也能降低求解难度. 因此, 研究推进子系统数学模型, 本质上就是建立飞控各部件之间的函数关系并找出这些函数的适用条件.

本文所使用的推进子系统模型来自于该领域认可度较高的文献(见文献[1,2,10]), 且所选参考文献内容较为清晰和实用, 便于形式化建模. 其中, 文献[1]和文献[2]是分别用英文和中文撰写的内容基本相同的两本教材. 而文献[10]是对文献[1]和文献[2]有关推进子系统建模的单独讨论. 本文模型通用性较好, 适用性和实用性都较强. 首先, 该模型可用于市面上常见的电调电池式多旋翼飞行器, 它包含的参数是多旋翼设计中所需的基本且必要的参数, 通用性较好. 其次, 该模型对机身布局、机身尺寸和旋翼个数没有限制, 适用性较强. 再次, 该模型对一些细节作了合理的简化, 实用性较强.

3 Coq 定理证明器

Coq 是基于归纳构造演算理论而实现的交互式定理证明器, 可用于构造程序和机器可验证的证明. Coq 可以进行命题演算、时序逻辑、谓词逻辑和高阶逻辑的证明, 甚至可以自定义逻辑, 然后在这种新定义的逻辑系统中进行逻辑推导. 在 Coq 中不但可以对逻辑公式进行推导, 还可以构造算法, 并进行算法正确性验证.

Coq 能够基于 Curry-Howard 同构原理进行程序抽取. 也就是说, 证明过程等同于程序开发过程. 使用证明技术构造出满足类型需求的函数, 即相当于设计出了满足输入输出的程序. Coq 支持将某些数据结构以及作用在这些数据结构上的函数转变成 OCaml 或 Haskell 等函数式程序, 这一称为程序抽取的能力使得 Coq 更适合于进行软件的正确性验证, 以及可证正确的软件开发[9,21].

4 推进子系统的形式化验证

将 Coq 定理证明技术应用于推进子系统, 对该系统进行形式化设计与验证, 有助于设计出高可靠的模型和软件. 推进子系统模型的主要任务是在各个变量之间建立联系. 将飞行器的设计参数定义为 Coq 中的常量, 其余参数作为变量, 将变量之间的计算定义为各种函数. 分离出由工程理论或模型直接得到的基本函数, 而基本函数可以直接推导出反函数, 基本函数多步复合可以推导出更实用的复合函数, 各类函数还可以推导出性能更好的最简形式函数(见第 4.4.4 节)等, 利用这些函数可以有效地应对各种需求. 除基本函数来自于参考文献以外, 其余函数都是根据基本函数手动演算而来. 利用 Coq 定理证明将数学推导表述为引理并予以证明, 能够保证这些推导过程的正确以及前提条件不被遗漏.

在新的开发方法下, 飞控领域专家只需关注少量基本概念和基本函数的正确性; 而大量复杂问题的求解及正确性可以由形式验证技术来加以保证; 软件编写也不再是由手工进行, 而是依赖于代码抽取或代码生成技术. 本文目前只实现了代码抽取, 而更复杂的代码生成功能尚未实现. 需要指出的是, 本文假定了来自参考文献的基本函数的正确性, 其详细验证需要后续继续加以研究. 这类向底层继续验证的工作可用本文方法进行类似的处理, 只是切换到了另一些问题域, 同理向上层继续验证也可使用本文方法.

对各类函数进行形式验证同时也面临着诸多挑战: 如何提取出各类函数, 这需要理解文献中的工程原理, 把数学知识从飞控专业知识中分离出来, 成为可以进行形式验证的对象; 衍生函数是使用手工推导方式对多个函数复合而来, 其表达式本身难以书写正确, 当形式验证证明不出时难以确认是由于函数定义错误还是证明本身较为困难; Coq 中实数理论的实现较为复杂, 为了处理不等式、多项式、根式、三角函数等数学对象, 需要引用较多零散的引理, 难以高度地自动化实现. 但是, 随着飞控基础函数库的逐步建立, 这些问题都将一一得到解决.

4.1 构建适合形式验证的文档

飞控系统的领域知识分散在各种教科书、讲义、手册中, 并且大都是以自然语言描述, 并不适合直接进行形式验证. 我们需要设计一种中间媒介来处理飞控领域知识的抽象模型, 既允许领域专业人员查看和更新设计, 又适合形式验证人员无歧义地使用这些模型. 形式验证的第一项也是最重要的一项工作就是对飞控领

域知识进行整理, 分析概念出现的先后顺序, 并逐步构建出具有层次结构的以适合形式验证的文档. 图3所示为领域知识模型.

具有层次结构是指将飞控系统中的知识转化为一种“金字塔”式自底向上的知识库: 最底层是基本对象, 包括基本概念、基本定理、基本公式、基本模型等; 往上是利用基本对象构建的飞控领域的基本函数、基本性质; 再往上是飞控中需要解决实际问题的复杂函数、复杂性质; 最顶层是构建出的飞控系统. 适合于形式验证是指该文档在内容编排顺序、表现形式、完整性方面有一定的要求. 在内容编排顺序上, 根据内容的依赖次序进行编排, 比如先介绍常量、变量, 然后是基本函数、复合函数等, 以降低理解难度. 在表现形式上, 多使用表格、公式、函数等简洁的形式. 在内容完整性上要求所论述内容是一个完整的系统, 使得形式验证人员无需查阅其他文献即可完成验证.

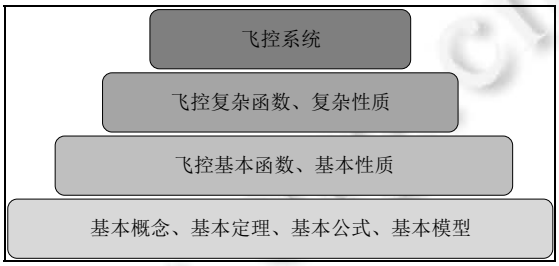


图 3 适合于形式验证的领域知识模型

构建适合形式验证的文档具体完成的工作有: 理解并整理领域知识; 将基本概念建立为 Coq 常量和 Coq 变量, 基本运算建立为基本函数, 数学推导而得的运算建立为复合函数; 将推导过程的正确性建立为命题; 设计了变量关系图, 用于帮助证明; 提出了最简形式函数概念并制作了最简形式函数汇总表, 用于优化运算; 给出了多个典型求解问题的算法设计.

文献[1]中给出的工程理论大都是用文字来表述的, 难以看清推进子系统中变量的依赖关系. 本文已将这些理论重新组织, 把推进子系统设计中变量及其函数关系绘制在一张图中, 得到了变量依赖关系的拓扑图, 简称变量关系图. 去掉常量参数, 则整个推进子系统被简化为仅含有14个变量的变量关系图, 如图4所示. 图中的圆圈表示变量, 连线表示变量之间的函数关系. 图中虚线表示的复合函数仅给出了几个示例, 并不是全部. 但不难发现, 任何两个变量之间都可以给出一个函数关系, 只是有些函数的形式比较复杂.

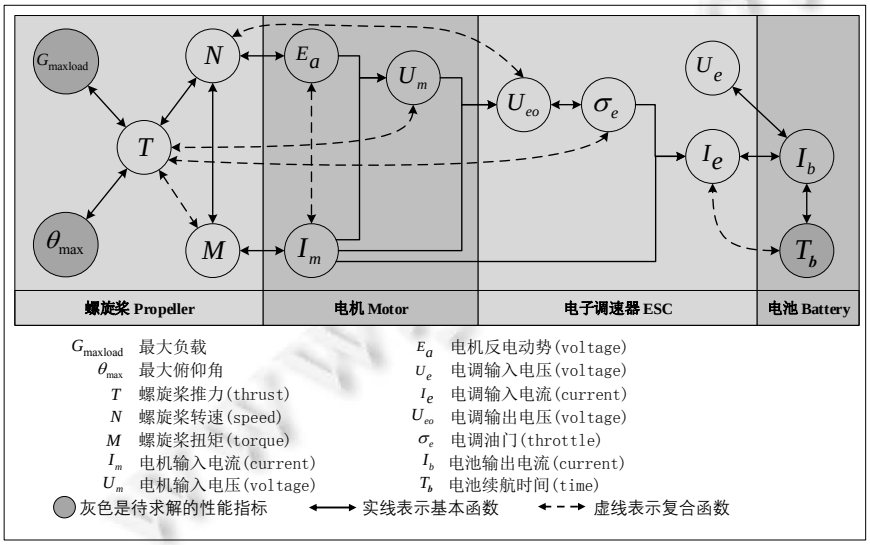


图 4 推进子系统的变量关系图

4.2 常量的定义

常量是指飞行器设计参数的一种数学抽象, 包括环境、螺旋桨、电机、电子调速器和电池的设计参数, 见表 1. 本文根据文献中的作法对空气动力学部分作了一些简化, 将飞行高度和环境温度设置为常量. 因为一般情况下, 小型多旋翼飞行高度变化不大, 由相关大气理论可知温度变化不大, 将高度和温度作为常量输入不影响验证结论.

表 1 推进子系统中的常量

分组	数学符号	Coq 变量名	名称	单位
环境	T_0	T_0	标准温度	K
	p_0	p_0	标准大气压	Pa
	ρ_0	rho_0	标准大气密度	kg/m^3
	G	G	总重量	N
	n_r	n_r	螺旋桨个数	piece
	I_{other}	I_other	其他电流	A
	h	h	飞行海拔高度	m
	T_t	T_t	环境温度	°C
	p	p	大气压力	Pa
	ρ	rho	大气密度	kg/m^3
螺旋桨	D_p	D_p	螺旋桨直径	m
	H_p	H_p	螺旋桨螺距	m
	B_p	B_p	螺旋桨叶片数	piece
	G_p	G_p	螺旋桨重量	N
	C_{fd}	C_fd	零升阻力系数	none
	C_T	C_T	拉力系数	none
	C_M	C_M	扭矩系数	none
	C_d	C_d	叶片截面翼型的阻力系数	none
电机	K_{V0}	K_V0	标称空载电机 KV 值	r/min/V
	U_{m0}	U_m0	标称空载电机电压	V
	I_{m0}	I_m0	标称空载电机电流	A
	$I_{m\mathrm{Max}}$	I_mMax	电机最大输入电流	A
	R_m	R_m	电机内阻	Ω
	G_m	G_m	电机重量	N
	K_E	K_E	反电动势常数	none
	K_T	K_T	转矩常数	none
电子调速器	$I_{e\mathrm{Max}}$	I_eMax	电调最大电流	A
	R_e	R_e	电调内阻	Ω
	G_e	G_e	电调重量	N
电池	R_b	R_b	电池内阻	Ω
	U_b	U_b	电池满电电压	V
	C_b	C_b	电池容量	mAh
	$C_{\min}$	C_min	电池最小保留容量	mAh
	K_b	K_b	最大放电倍率	C
	G_b	G_b	电池重量	N

在 Coq 系统中, 使用 Parameter 关键字来声明一个常量, 在实际使用时再指定其具体值. 虽然 Coq 系统部分支持使用 Latex 风格的标识符, 以及任意 Unicode 符号作为标识符, 如表 1 中数学符号一列的内容可以在 Coq 中直接输入. 但考虑到需要与其他语言保持一致, 我们还是使用传统的 ASCII 字符作为 Coq 变量名, 比如 alpha 表示α, T_t 表示 T_t 等. 示例 Coq 代码见表 2.

控制系统中的大多数物理量需要用实数来建模, 使用 Coq 标准库 Coq.Reals 中类型 R 来表示实数. 在标准库中有大量与实数有关的运算和性质, 可以方便地用于工程应用中[22]. 对于更复杂的实数理论, 可以参考 Coquelicot 和 Mathmatical Component 这两个数学库[11,12]. 计算机能处理的浮点数实际上是实数的一个子集,

本文暂不处理因计算舍入而引起的精度问题, 本文重点关注模型和算法的正确性.

表 2 推进子系统中的常量声明脚本

```
1.  Require Export Reals.
2.  Open Scope R.
3.  (** Global *)
4.  Parameters T_0 p_0 rho_0 h T_t n_r G I_other: R.
5.  (** Propeller *)
6.  Parameters D_p H_p B_p G_p PP_A PP_epsilon PP_lambda PP_zeta PP_e PP_K0 PP_alpha0 C_fd: R.
7.  (** Motor *)
8.  Parameters K_V0 I_mMax U_m0 I_m0 R_m G_m: R.
```

有一些变量是根据已知常量给出的一个算术表达式, 把这一类变量也当作是常量. 例如, 公式(1)用于计算空气压强[1−3]. 其中, p_0 是标准大气压强, h 是飞行海拔高度, T_0 是标准温度, T_t 是实际温度, −0.006 5 是温度随海拔变化的比例系数(表示在对流层 0−11 km 中, 海拔每上升 1 000 m, 温度下降 6.5 ℃), 5.256 1 是在对流层内温度与压强变化率的一个系数(根据气体状态方程建模并求解而来).

$$p = p_0\left(1-0.0065\frac{h}{T_0+T_t}\right)^{5.2561} \tag{1}$$

示例 Coq 代码见表 3.

表 3 空气压强计算公式的相关脚本

```
1   (* Some identifiers for special real numbers *)
2   Parameter value_0_0065: R. (* 0.0065 *)
3   (* Some axioms for special real numbers *)
4   Axiom gt0_value_0_0065: 0<value_0_0065.
5   (* Give a special value during extraction *)
6   Extract Constant value_0_0065 =>"0.0065".
7   (* Definition of Air pressure *)
8   Definition p: =p_0 * (Rpower (R1-value_0_0065 * (h/(T_0+T_t))) value_5_2561).
```

其中, Rpower 是实数库中的幂指函数, R1 是实数 1, value_0_0065 是一个标识符, 表示一个值为 0.006 5 的 R 类型的数, gt0_value_0_0065 是 0.006 5 为正数的这个性质被表述为公理. 使用标识符而不是小数样式的浮点数有几个原因: 一是 Coq 8.10 之前的版本不支持直接输入带小数样式的浮点数, 为了保持与旧版本兼容; 二是因这些常数多次使用, 需要防止输入错误; 三是防止在证明时被计算指令展开为复杂表达式而加大证明难度. 本质原因是实数的构造相当复杂, Coq 中 0.006 5 是由 6 500 和 10 000 这两个整数构造出有理数后经过类型提升得到的实数, 而整数(Z 类型)又是使用正整数(positive 类型)来归纳构造. 在 Coq 中, 6 500 被表示为 Zpos (xO (xO (xI (xO (xO (xI (xI (xO (xI (xO (xO (xI xH))))))))))).

4.3 变量的定义

变量是除常量以外所有出现在系统中的物理量. 在 Coq 系统中, 变量的值并不是直接给定, 而是在具体问题中随着函数调用给出其当前上下文中的值. 比如, 螺旋桨转速 N, 在不同的问题中有不同的求解结果. 这些变量在第 4.7 节的具体问题中会用到. 表 4 给出了所有的变量.

表 4 推进子系统中的变量

分组	数学符号	Coq 变量名	名称	单位
整机	G_{maxload}	G_maxload	最大载荷	kg
	$\theta_{\max}$	theta_max	最大俯仰角	m/s
	T_b	T_b	最长悬停时间	min
	η	eta	系统效率	none

表 4 推进子系统中的变量(续)

分组	数学符号	Coq 变量名	名称	单位
螺旋桨	T	T	螺旋桨推力	N
	M	M	螺旋桨扭矩	N.m
	N	N	螺旋桨转速	r/min
电池	I_b	I_b	电池输出电流	A
电机	E_a	E_a	反电动势	V
	U_m	U_m	等效电机输入电压	V
	I_m	I_m	等效电机输入电流	A
电子调速器	I_e	I_e	电调输入电流	A
	U_e	U_e	电调输入电压	V
	U_{eo}	U_eo	等效电调输出电压	V
	σ_e	sigma_e	电调油门	none

4.4 函数的定义

4.4.1 基本函数的定义

根据物理原理、模型或经验给出的变量之间求解公式作为基本函数. 这些函数来源于参考文献[1−3].

例如, 由螺旋桨转速 N 来计算螺旋桨推力 T 的公式(2). 其中, ρ、C_T、C_P 都是常量, 可参考表 1.

$$get\_T\_by\_N(N)=\rho C_T D_P^4\left(\frac{N}{60}\right)^2 \tag{2}$$

对应的 Coq 代码是一个单参数的函数, 见表 5.

表 5 由螺旋桨转速计算螺旋桨推力的函数

```
1  (* Calculate propeller thrust T(unit: N) by propeller speed N(unit: r/min) *)
2  Definition get_T_by_N N: =rho * C_T * (D_p ^4) * ((N/60) ^2).
```

该定义的正确性本文未深入研究, 原因见第 4 节开始处的讨论. 这里只是确保 Coq 定义与参考文献中的定义一致, 依赖于手工检查. 基本函数约 14 个, 部分基本函数见表 6.

表 6 PSS 中部分基本函数

函数名称	函数功能
get_T_by_N	根据螺旋桨转速计算螺旋桨推力
get_M_by_N	根据螺旋桨转速计算螺旋桨扭矩
get_E_a_by_N	根据螺旋桨转速计算电机反电动势
get_M_by_I_m	根据电机等效输入电流计算螺旋桨扭矩
get_U_m_by_E_a_and_I_m	根据电机等效输入电流和电机反电动势计算电机等效输入电压
get_U_eo_by_U_m_and_I_m	根据电机等效输入电压和电机等效输入电流计算电调等效输出电压

4.4.2 基本函数的反函数定义

基本函数的反函数是指对基本函数的逆运算, 简称基本反函数. 利用反函数可以方便地处理正向或反向求解问题. 这些反函数并不是直接来源于参考文献[1,2,10], 而是手工推导而来. 需要指出的是, 手工推导虽然大多数时是正确的, 但是推导中产生的一些约束条件可能会被遗漏. 比如平方根运算要求输入值不小于 0, 这一要求可能因遗漏而成为一个隐藏的软件缺陷. 而定理证明可以确保捕获到所有的前提条件.

例如, 由螺旋桨推力 T 计算螺旋桨转速 N 的公式(3), 可以由上一小节的公式(2)反推而来.

$$get\_N\_by\_T(T)=60\sqrt{\frac{T}{\rho C_T D_p^4}} \tag{3}$$

对应的 Coq 代码见表 7.

表 7 由螺旋桨推力计算螺旋桨转速的函数

```
1  (* Calculate propeller speed N(unit: r/min) by propeller thrust T(unit: N) *)
2  Definition get_N_by_T T: =60 * sqrt (T/(rho * C_T * (D_p ^4))).
```

传统上我们会直接使用该函数而不加以证明, 其实该函数的正确性需要被证明. 在第 4.5 节的证明中我们会看到, 该函数需要多个前提条件才能成立. 这也反映出定理证明器中采用严格的数学形式所带来的好处. 基本反函数约 10 个, 部分基本反函数见表 8.

表 8 PSS 中部分基本反函数

函数名称	函数功能
get_N_by_T	根据螺旋桨推力计算螺旋桨转速
get_N_by_M	根据螺旋桨扭矩计算螺旋桨转速
get_N_by_E_a	根据电机反电动势计算螺旋桨转速
get_I_m_by_M	根据螺旋桨扭矩计算电机等效输入电流
get_sigma_e_by_U_eo	根据电调等效输出电压计算电调油门指令

4.4.3 复合函数的定义

复合函数是指由基本函数和基本反函数经过多次复合而得到的函数. 复合函数将原本需要多次调用基本函数的步骤封装了起来, 减少了中间变量个数, 降低了人为调用可能出错的风险. 其核心思想是封装业务逻辑, 对飞控知识进行一层层的抽象. 这些函数表达式的形式和推导过程往往比较复杂, 很容易产生错误, 使用 Coq 定理证明可以确保这类推导过程的正确性.

例如, 由电调输出电压 U_{eo} 计算螺旋桨转速 N 的计算公式(4)是由多个公式复合而成, 有两种可能的结果.

$$get\_N\_by\_U_{eo}(U_{eo})=\frac{1800K_T}{(R_m+R_e)C_M\rho D_p^5}\left(-K_E\pm\sqrt{K_E^2-\frac{(R_m+R_e)C_M\rho D_p^5}{900K_T}\left((R_m+R_e)I_{m0}-U_{eo}\right)}\right) \tag{4}$$

对应的 Coq 代码, 见表 9.

表 9 由电机输入电压计算螺旋桨转速的函数

```
1   (* Calculate propeller speed N(unit: r/min) by the motor output voltage U_eo(unit: V) *)
2   Definition get_N_by_U_eo1 U_eo: =(1800 * K_T)/((R_m+R_e) * C_M * rho * (D_p^5)) * (-K_E+sqrt((K_E^2)
3   -((R_m+R_e) * C_M * rho * (D_p^5))/(900 * K_T) * ((R_m+R_e) * I_m0-U_eo))).
4   Definition get_N_by_U_eo2 U_eo: =(1800 * K_T)/((R_m+R_e) * C_M * rho * (D_p^5)) * (-K_E-sqrt((K_E^2)
5  - ((R_m + R_e) * C_M * rho * (D_p^5))/(900 * K_T) * ((R_m+R_e) * I_m0-U_eo))).
```

公式(4)的形式已经比较复杂, 如果是手工进行这样的推导, 不但工作量大, 而且容易出错. 复合函数约 16 个, 部分复合函数见表 10.

表 10 PSS 中部分复合函数

函数名称	函数功能
get_T_by_M	根据螺旋桨扭矩计算螺旋桨推力
get_M_by_T	根据螺旋桨推力计算螺旋桨扭矩
get_I_m_by_T	根据螺旋桨推力计算电机等效输入电流
get_E_a_by_T	根据螺旋桨推力计算电机反电动势
get_U_m_by_T	根据螺旋桨推力计算电机等效输入电压
get_U_m_by_N_and_M	根据螺旋桨转速和螺旋桨扭矩计算电机等效输入电压
get_N_by_U_eo1	根据电调等效输出电压计算螺旋桨转速(情形 1)
get_N_by_U_eo2	根据电调等效输出电压计算螺旋桨转速(情形 2)

4.4.4 最简形式函数的定义与分析

工程应用中的公式(或称函数)一般形式比较复杂, 表现为含有多个常量、变量和各类运算的复杂形式, 复杂的形式无论从理解和分析的角度, 还是从计算机运行效率的角度而言都不是最优的. 为分析功能相同但形

式不同的函数间的计算成本差异, 我们引入了一个概念, 最简形式函数(simplest form function, SFF). 在确定了常量值后将公式中剩余的变量当作自变量, 把函数变形得到对自变量而言在数学形式上最简的函数称为最简形式函数. 按照自变量数目逐步给出 SFF 的数学形式.

当只有 1 个自变量时:

$$f(x)=a_n x^n+a_{n-1}x^{n-1}+\ldots+a_2x^2+a_1x+a_0=\sum_{i=0}^{n}a_i x^i \tag{5}$$

当有 2 个自变量时:

$$f(x_1,x_2)=a_0+a_1x_1+a_2x_2+a_3x_1^2+a_4x_2^2+a_5x_1x_2+\ldots=\sum_{i=0}^{n_1\times n_2}\sum_{j_1=0}^{n_1}\sum_{j_2=0}^{n_2}a_i x_1^{j_1}x_2^{j_2} \tag{6}$$

一般地:

$$f(x_1,x_2,\ldots,x_k)=\sum_{i=0}^{n_1\times n_2\times\ldots\times n_k}\left(\sum_{j_1=0}^{n_1}\ldots\sum_{j_k=0}^{n_k}\left(a_i\prod_{s=1}^{k}x_s^{j_s}\right)\right) \tag{7}$$

SFF 带来的一些便利: 从数学角度来看, 函数右侧的表达式按照自变量幂次进行同类项合并, 使得自变量与因变量的关系清晰而又简洁; 从计算机角度来看, 将原始函数转换为 SFF 后, 所有常数项系数表达式(上述公式中的 a_i)可以在编译时期被计算为常数, 使得整个函数在运行时的计算成本减小, 包括计算次数减少、中间变量数目减少. 下面举例说明 SFF 这一概念, 并分析 SFF 相比普通函数所带来的计算成本上的优化.

例如, 由螺旋桨转速 N 来计算电机输入电压 U_m, 这里给出两组计算公式:

$$get\_U_m\_by\_N(N)=K_EN+R_m\left(\frac{C_M\rho\left(\frac{N}{60}\right)^2D_P^5}{K_T}+I_{m0}\right) \tag{8}$$

$$get\_U_m\_by\_N\_sff(N)=\frac{R_mC_m\rho D_p^5}{3600K_T}N^2+K_EN+R_mI_{m0} \tag{9}$$

公式(8)是由各类已知函数直接推导而来的复合函数, 这里称为原始函数. 公式(9)是对公式(8)的优化, 是一个以 N 为自变量、以 U_m 为因变量的标准的一元二次方程, 各个加法项已经按照自变量 N 的不同幂次进行了同类项合并, 因此成为最简形式函数. 在公式(9)的右侧除 N 外其余参数都是常数, 在编译期间即可确定. 分析这两个函数的计算成本, 可发现 SFF 版本的函数在空间和时间上都是最优的, 如图 5 所示. 原始函数与最简形式函数的等价性在第 4.6 节中给出证明.

图 5 中, 顶部两张表是原始函数与最简形式函数的计算过程分析; 中间是两种不同计算过程的树形结构图, 树根是因变量, 阴影方形叶子是自变量 N, 其余叶子是常量, 树的高度反映了计算成本, 阴影圆形节点个数反映了临时变量个数; 底部一张汇总表显示了 SFF 带来的计算优化和存储优化效果. 从中可以看出, 原始函数的乘法次数、总计算次数、中间变量个数都多于 SFF 函数. 这些数据说明 SFF 对计算成本有显著降低, 时间和空间两方面都有优势. 在实时性要求较高的飞控场景, 如果这一转换被广泛应用于各类函数的设计中, 当这些函数被高频调用时将为系统节约宝贵的时间和空间资源. 但是, 变换的正确性是这些函数可用的必要前提, 而使用 Coq 定理证明可以确保这些变换是正确的.

对应的 Coq 代码见表 11. 此处是函数式风格的程序, 计算过程不明显, 若在命令式程序中, 可以将这里的 a、b、c 这 3 个中间变量使用宏定义来预先算出.

详细的函数列表由于数量较多, 此处不再罗列. 所有这类函数都是以“_sff”结尾, 表示最简形式函数.

本文还在 SFF 的基础之上, 提出了推进子系统的最简形式函数汇总表. 如图 6 中表格所示, 左边行首是自变量, 上边列首是因变量, 在行首和列首以外的单元格中存储了自变量到因变量的函数关系式, 关系式中的 a、b、c 表示某个常数系数. 例如, 左上角第 4 行第 2 列单元格中的 $ax+b$ 表示了自变量 T 到因变量 G_{maxload} 的函数关系为 $y=ax+b$, 亦即 $G_{\text{maxload}}=aT+b$. 该函数的物理含义是, 多旋翼的最大负载 G_{maxload} 等于 n_r 个螺旋

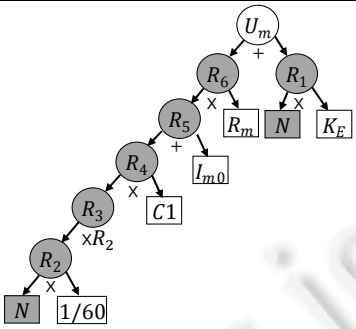
桨乘以每个桨产生的推力 T 得到总推力再减去整机的重量 G. 从任意一个变量出发, 都可以求解出任意其他变量, 只是有些求解公式成了高次多项式或含有三角运算等以致可能有多个解而未列出, 但根据其他约束条件可以得到确定的解.

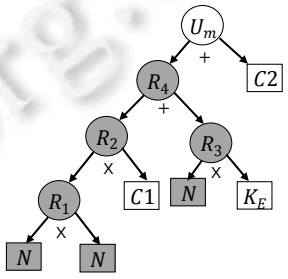
原始函数的计算代价分析

步骤	输入	输出	常数	计算类型
1	N	$R1 = K_E * N$	K_E	乘法
2	N	$R2 = N * 1/60$	$1/60$	乘法
3	$R2$	$R3 = R2 * R2$	--	乘法
4	$R3$	$R4 = R3 * C1$	$C1 = \frac{C_m \rho D_p^5}{3600 K_T}$	乘法
5	$R4$	$R5 = R4 + I_{m0}$	I_{m0}	加法
6	$R5$	$R6 = R5 * R_m$	R_m	乘法
7	$R1, R6$	$U_m = R1 + R6$	--	加法

最简形式函数(SFF)的计算代价分析

步骤	输入	输出	常数	计算类型
1	N	$R1 = N * N$	--	乘法
2	$R1$	$R2 = C1 * R1$	$C1 = \frac{R_m C_m \rho D_p^5}{3600 K_T}$	乘法
3	N	$R3 = K_E * N$	K_E	乘法
4	$R2, R3$	$R4 = R2 + R3$	R_m	加法
5	$R4$	$U_m = R4 + C2$	$C2 = R_m * I_{m0}$	加法





各项指标	原始函数=A	最简函数形式=B	$\frac{\text{已优化的数量}}{\text{原始数量}} = \frac{A-B}{A}$
总乘法次数	5	3	**40%**
总加法次数	2	2	**0%**
总计算次数	7	5	**28%**
总中间变量个数	6 ($R1,\ldots,R6$)	4 ($R1,\ldots,R4$)	**33%**

图 5 原始函数与最简形式函数的计算成本比较

表 11 由螺旋桨转速计算电机输入电压的函数

```
1   (* Calculate motor output voltage U_m(unit: V) by propeller speed N(unit: r/min). (Original Function) *)
2   Definition get_U_m_by_N N: =K_E * N+R_m * ((C_M * rho * ((N/60)^2) * (D_p^5))/K_T+I_m0).
3   (* Calculate motor output voltage U_m(unit: V) by propeller speed N(unit: r/min).(Simplest Form of Function) *)
4   Definition get_U_m_by_N_sff N: =
5     let a: =R_m * C_M * rho * (D_p^5)/(3600 * K_T) in
6     let b: =K_E in
7     let c: =R_m * I_m0 in
8     let x: =N in
9       a * x^2+b * x+c.
```

最简形式函数汇总表有几个用途.

(1) 可直观地确定任意两个变量之间的函数关系, 尤其是可用于构造新的函数关系. 例如, 若需要在该表的空单元格处增加函数以确定两个变量之间的关系, 只需利用已知的结果通过多步复合推导出一个确定的函数形式, 然后再去处理常数项即可.

(2) 可用于辅助发现某些物理规律. 由于最简形式函数屏蔽了常数项细节仅保留自变量与因变量之间的

数学关系, 并且若自变量和因变量之间原本不存在物理联系但又能推导出一个简洁的数学关系, 则有可能在这些物理量之间发现新的物理规律.

最简形式函数的推导目前是手工完成的, 其正确性难以保证, 但使用定理证明技术可以确保这些推导是正确的, 见第 4.5 节和第 4.6 节.

x \ y	$G_{maxload}$	θ_{max}	T	N	M	E_a	I_m	U_m	U_{eo}	σ_e	I_e	I_b	T_b	U_e
$G_{maxload}$	1		$ax+b$	$a\sqrt{x+b}$	$ax+b$	$a\sqrt{x+b}$	$ax+b$							
θ_{max}		1	$\frac{a}{\cos x}$											
T	$ax+b$	$a\cos\left(\frac{a}{x}\right)$	1	$a\sqrt{x}$	ax	$a\sqrt{x}$	$ax+b$	$\frac{ax^2+}{b\sqrt{x}+c}$						
N	ax^2+b		ax^2	1	ax^2	ax	ax^2+b	$\frac{ax^2+}{bx+c}$	$\frac{ax^2+}{bx+c}$	$\frac{ax^2+}{bx+c}$	$\frac{ax^4+}{bx^3+}$ $\frac{cx^2+}{dx+e}$			
M	$ax+b$		ax	$a\sqrt{x}$	1	$a\sqrt{x}$	$ax+b$							
E_a	ax^2+b		ax^2	ax	ax^2	1	ax^2+b							
I_m	$ax+b$		$ax+b$	$a\sqrt{x+b}$	$ax+b$	$a\sqrt{x+b}$	1							
U_m								1						
U_{eo}									1	ax				
σ_e									ax	1				
I_e											1	$ax+b$	$\frac{b}{x+a}$	$ax+b$
I_b											$ax+b$	1	$\frac{a}{x}$	$ax+b$
T_b											$\frac{a}{x}+b$	$\frac{a}{x}$	1	$\frac{a}{x}+b$
U_e											$ax+b$	$ax+b$	$\frac{b}{x+a}$	1

图 6 推进子系统的最简形式函数汇总表

4.5 引理的定义

工程问题与纯数学问题的形式化流程类似, 但有所不同. 纯数学问题的形式化, 一般是定义数学结构、定义该结构上的运算、定义运算需要满足的性质、然后证明这些运算满足期望的性质, 这类问题的处理流程相对比较固定. 工程问题的形式化, 首先需要将工程知识“编译”为形式化描述的模型, 这些内容可以看作是一种“数学结构”; 然后将工程问题中的控制流程封装为函数定义, 可看作是“数学运算”; 下一步是将这些运算需要满足的性质作为引理予以证明, 这里的性质是指数学推导的正确性.

第 4.2 节–第 4.4 节介绍了常量、变量和基本函数和各种衍生函数. 利用这些函数和进一步推导可以在任意两个变量之间建立函数关系. 这些函数的正确性需要设计一批引理来保证. 在假定基本函数正确的前提下, 设计相关引理来证明推导过程正确, 从而说明这些衍生函数在假设条件下正确.

首先, 验证反函数的正确性. 由于已经假定了基本函数的正确性, 因此只要验证反函数与基本函数互逆即可. 然后, 验证复合函数的正确性, 通过证明复合函数的直接计算结果与多步调用简单函数的结果相等来验证. 其次, 最简形式函数的正确性验证, 只需验证两种计算方式的计算结果相同即可.

对基本反函数的正确性验证, 设计名为 verify_trans_XX_and_YY 的一对引理, 其中, XX 和 YY 分别表示两个变量. 例如, 螺旋桨转速 N 和螺旋桨推力 T 之间存在一对反函数, 设计两个引理见表 12.

表 12 基本反函数的正确性验证引理

```
1    (* get_N_by_T is the left inverse of get_T_by_N *)
2    Lemma verify_trans_N_and_T N: 0<=N-> get_N_by_T(get_T_by_N N)=N.
3    (* get_T_by_N is the left inverse of get_N_by_T *)
4    Lemma verify_trans_T_and_N T: 0<=T-> get_T_by_N(get_N_by_T T)=T.
```

第 1 个引理表示: 由螺旋桨转速 N 计算出螺旋桨推力 T, 再由 T 计算出新的螺旋桨转速 N', 此时 N 和 N' 须相等. 第 2 个引理表示: 由螺旋桨推力 T 计算出螺旋桨转速 N, 再由 N 计算出新的螺旋桨推力 T', 此时 T 和 T'须相等.

对复合函数的正确性验证, 设计名为 verify_XXX 的引理, 其中, XXX 表示扩展函数的名称. 例如: 验证由螺旋桨推力 T 求解电机输入电压 U_m 的公式, 代码见表 13.

表 13 复合函数的正确性验证引理

```
1    (* Correctness of function of calculate motor input voltage U_m by propeller torque *)
2    Lemma verify_get_U_m_by_T T: get_U_m_by_T T=get_U_m_by_E_a_and_I_m(get_E_a_by_T T) (get_I_m_by_T T).
```

其中, T 表示螺旋桨推力, I_m 表示等效电机输入电流, U_m 表示等效电机输入电压, E_a 表示电机反电动势, get_U_m_by_E_a_and_I_m 表示根据 E_a 和 I_m 来计算 U_m, get_E_a_by_T 表示根据 T 来计算 E_a, get_I_m_by_T 表示根据 T 来计算 I_m. 该引理表示: 直接由 T 计算 U_m, 与带有中间过程的 3 步计算, 两种计算结果应该相同.

对于最简形式函数的正确性验证, 与复合函数的正确性验证类似, 这里不再赘述.

4.6 引理的证明

对于在第 4.5 节中设计的引理, 其证明难点主要是实数等式和不等式有关的证明, 以及如何提高自动化证明程度. 在 Coq 中, 实数类型 R 是以公理化方式实现, 其内部实现比较复杂[9,21,23], 于是与 R 类型有关的证明都会比较复杂和繁琐.

例如, 螺旋桨转速 N 和螺旋桨推力 T 之间的函数互逆性验证引理, 一种原始的繁琐证明过程见表 14. 这条引理的直观含义是: 任意一个大于 0 的螺旋桨转速 N, 先经过 get_T_by_N 函数运算得到螺旋桨推力 T, 再经过 get_N_by_T 函数运算得到新的螺旋桨转速 N', 两个转速应该相同. 其中, Lemma 表示开始定义引理; Proof 表示证明开始; Qed 表示证明完成; lra 是标准库提供的简单实数不等式证明策略, 其余的 Rmult_assoc、Rinv_r_simpl_m 等是实数库中自带的引理.

表 14 手动证明的一个引理

```
1    Lemma verify_trans_N_and_T' N: 0<=N-> get_N_by_T(get_T_by_N N)=N.
2    Proof.
3      intros H1.unfold get_N_by_T, get_T_by_N.
4      remember(rho * C_T * D_p^4) as a.unfold Rdiv.rewrite Rinv_r_simpl_m.
5     -rewrite sqrt_pow2.
6      +rewrite<-Rmult_assoc.rewrite Rinv_r_simpl_m; trivial.
7      +apply Rle_mult_inv_pos.trivial.lra.
8      -rewrite Heqa.apply Rmult_integral_contrapositive.split.
9       +apply Rmult_integral_contrapositive.split.
10          * apply Rgt_not_eq.apply Rlt_gt; apply gt0_rho.
11          * apply Rgt_not_eq.apply Rlt_gt; apply gt0_C_T.
12        + apply pow_nonzero; apply Rgt_not_eq; apply Rlt_gt; apply gt0_D_p.
13   Qed.
```

实数库有丰富的引理可以使用, 包含了常用的数学运算及其性质, 但是手工进行变换非常繁琐. 由于实数及其算术运算在数学上构成域, 利用 Coq 中的 Field 策略可以执行交换律、结合律、消除单位元、化简等各种变换. 另外, Field 策略的使用也有一些额外的要求, 比如等式中所含变元必须成对出现, 一些自定义记号需

要手动展开等. 此外, 利用 autounfold、autorewrite 策略可以进一步提高证明的自动化程度. 利用 Ltac 语言可以将这些技术组合在一起构造出符合需求的自定义策略. 如表 15 所示的代码, 演示了如何扩展 auto 数据库以及自定义的策略 simple_equation, 利用该策略可将 50 个引理中的 36 个简化为一步完成证明.

表 15 创建自定义策略的相关脚本

```
(* Update database of autounfold tactics *)
Global Hint Unfold get_E_a_by_N get_E_a_by_T: fcs.
(* Update database of auto tactics *)
Global Hint Resolve gt0_G gt0_n_r gt0_D_p gt0_C_T: flyctrl.
(* Update database of autorewrite tactics *)
Hint Rewrite pow2_sqrt cos_arccos_rev: flyctrl.
(* A custom tactic for
Ltac simple_equation: =
  intros; repeat autounfold with flyctrl;
  auto with flyctrl; unfold Rdiv; autorewrite with flyctrl;
  try field; try lra; auto with flyctrl;
  try apply Rmult_le_pos; auto with flyctrl;
  try apply Rmult_integral_contrapositive; auto with flyctrl;
  repeat try split; auto with flyctrl.
```

对于本小节一开始给出的手动证明的引理, 其简化后的证明见表 16.

表 16 使用自定义策略简化先前的证明

```
(* Another proof script using custom tactic *)
Lemma verify_trans_N_and_T N: 0<=N->get_N_by_T(get_T_by_N N)=N.
Proof.simple_equation.Qed.
```

飞控系统中大多数研究对象是实数有关的变量和表达式, 在包含除式或开根式运算时需要判定表达式非零或非负, 这类判定处理非常频繁, 且手工证明具有一定的相似性. 这类问题可以使用 Hint Resolve 或 Hint Rewrite 命令将适当的引理加入自动化数据库中, 以进一步提升证明的自动化程度. 有必要指出的是, 需要谨慎地加入引理, 以避免产生循环效应. 因为自动化策略复杂度增加以后其执行逻辑会非常难以跟踪, 极有可能使整个策略陷入死循环而无法终止. 表 17 列举了本文所用到的实数库中的部分引理, 这些引理被用于自定义证明策略 simple_equation 中, 它们不会产生循环效应, 罗列在此仅供参考.

表 17 自定义策略中用到的部分实数引理

引理名称	引理类型
Rlt_0_1	0<1
Rmult_le_pos	forall r1 r2: R, 0<=r1->0<=r2->0<=r1 * r2
Rmult_lt_0_compat	forall r1 r2: R, 0<r1->0<r2->0<r1 * r2
pow_nonzero	forall (x: R) (n: nat), x<>0->x^n<>0
Rinv_neq_0_compat	forall r: R, r<>0->/r<>0
Rinv_r_simpl_m	forall r1 r2: R, r1<>0->r1 * r2 */r1=r2
Rsqr_sqrt	forall x: R, 0<=x->(sqrt x)²=x
Rsqr_pow2	forall x: R, x²=x^2
sqrt_pow2	forall x: R, 0<=x->sqrt (x^2)=x

4.7 典型问题求解示例

在构造了各类函数以后, 便具有了在任意变量之间相互求解的能力. 对于具体的性能求解问题, 其实只是一个确定初始条件, 并给出求解序列的过程, 变得非常简单.

4.7.1 最长悬停时间求解

这是一个正向求解的例子[1−3].

问题: 在给定参数并假定没有外部干扰的情况下, 估算飞行器最长悬停时间.

解答: 在悬停状态下, 推进子系统产生的推力与飞行器本身重量相等. 计算过程如图 7 所示, 图中区块 T 表示起点, T_b 表示终点, 加粗的连线表示数据流.

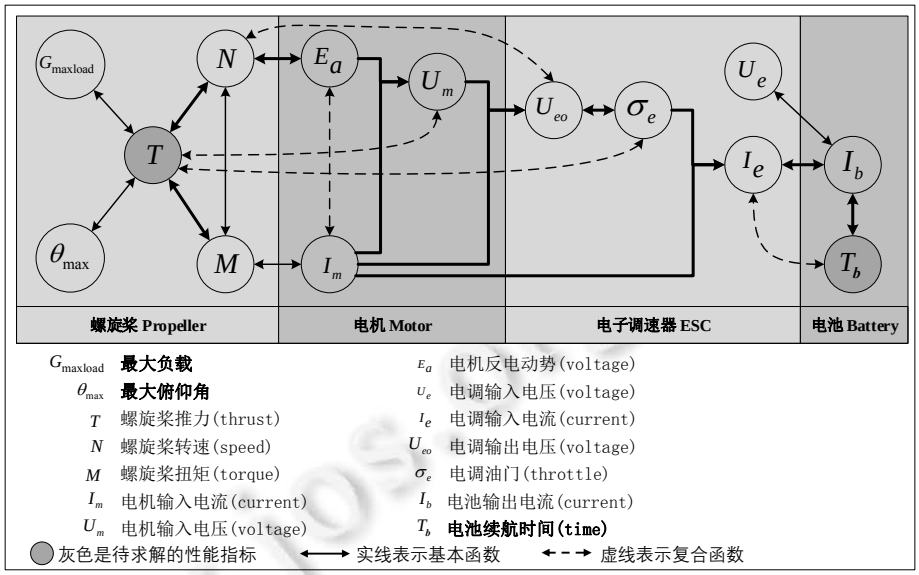


图 7 最长悬停时间求解数据流图

计算步骤如下.

(1) 计算单个螺旋桨需要产生的推力, 等于总质量除以螺旋桨个数.

$$T = G / n_r \tag{10}$$

(2) 计算螺旋桨转速

$$N = get\_N\_by\_T(T) \tag{11}$$

(3) 计算螺旋桨扭矩

$$M = get\_M\_by\_T(T) \tag{12}$$

(4) 计算电机反电动势

$$E_a = get\_E\_a\_by\_N(N) \tag{13}$$

(5) 计算电机输入电流

$$I_m = get\_I\_m\_by\_M(M) \tag{14}$$

(6) 计算电机输入电压

$$U_m = get\_U\_m\_by\_E\_a\_and\_I\_m(E\_a, I\_m) \tag{15}$$

(7) 计算电调输出电压

$$U_{eo} = get\_U\_eo\_by\_U\_m\_and\_I\_m(U\_m, I\_m) \tag{16}$$

(8) 计算电调油门

$$sigma_e = get\_sigma\_e\_by\_U\_eo(U\_eo) \tag{17}$$

(9) 计算电调输入电流

$$I_e = get\_I\_e\_by\_I\_m\_and\_sigma\_e(I\_m, sigma\_e) \tag{18}$$

(10) 计算电池输出电流

$$I_b = get\_I\_b\_by\_I\_e(I\_e) \tag{19}$$

(11) 计算悬停时间

$$T_b = get\_T\_b\_by\_I\_b(I\_b) \tag{20}$$

4.7.2 最大负载和最大俯仰角求解

这是一个逆向求解的例子[1−3].

问题: 假定没有外部干扰, 求解多旋翼最大负载重量和最大俯仰角.

解答: 只需要对电调油门保留一定的安全裕度即可. 设定电调油门为 80%, 可以逐步反向求解出关键变量——螺旋桨推力 T, 然后可以求得最大负载和最大俯仰角. 此时的数据流图如图 8 所示. 图中区块 σ_e 表示起点, G_{maxload} 和 $\theta_{\max}$ 表示终点, 加粗的连线表示数据流.

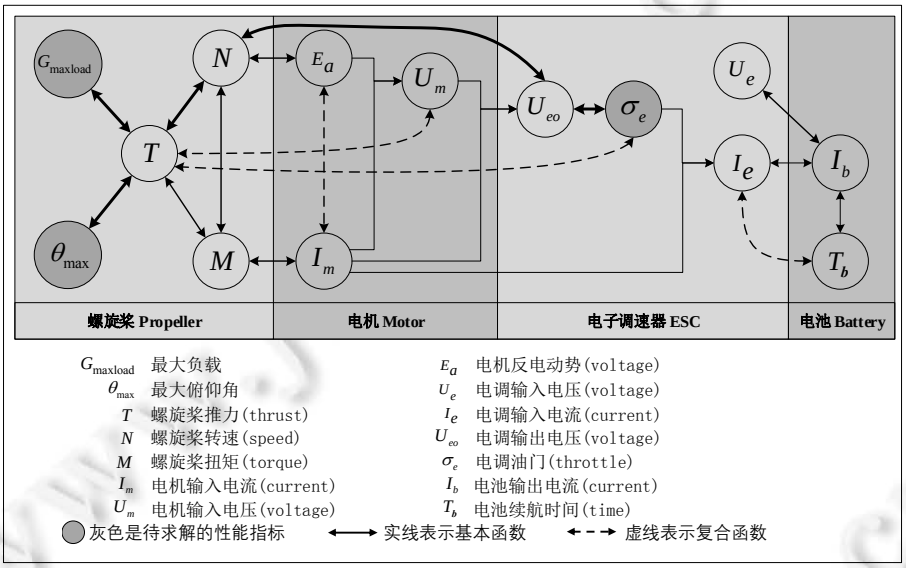


图8 最大负载与最大俯仰角求解数据流图

计算步骤如下.

(1) 给定油门指令

$$sigma_e = 0.8 \tag{21}$$

(2) 计算电调输出电压

$$U_{eo} = get\_U\_eo\_by\_sigma\_e(sigma\_e) \tag{22}$$

(3) 计算电机转速(也就是螺旋桨转速)

$$N = get\_N\_by\_U\_eo(U\_eo) \tag{23}$$

(4) 计算螺旋桨推力

$$T = get\_T\_by\_N(N) \tag{24}$$

(5) 计算最大负载

$$G_{\text{maxload}} = get\_G\_\text{maxload}\_by\_T(T) \tag{25}$$

(6) 计算最大俯仰角

$$\theta_{\max} = get\_\theta\_\max\_by\_T(T) \tag{26}$$

4.8 抽取OCaml代码

使用 Coq 编写的函数通常是可以用函数式程序设计语言实现的函数. 把 Coq 中的函数自动映射到另一种程序设计语言中, 便得到了程序, 这种机制称为代码抽取. 由于抽取出的程序满足在形式开发中描述的规范说明, 所以利用 Coq 系统可以生成确证的软件.

首先要做的是对归纳定义的数据类型或公理化数据类型进行类型映射. 例如, 将 R 类型映射到 OCaml 中的“float”类型, 并将 R 中的算术运算映射为对应的“float”类型上的算术运算. 部分抽取指令见表 18.

表 18 从 Coq 抽取 OCaml 代码的指令

```
1   Require Import Extraction.
2   Extract Constant R0=>"0.0".
3   Extract Constant R1=>"1.0".
4   Extract Constant PI=>"Float.pi".
5   Extract Constant Rplus=>"(+.)".
6   Extract Constant Ropp=>"fun a->(0.0-.a)".
7   Extract Constant Rpower=>"Float.pow".
8   Extract Constant sin=>"Float.sin".
9   Extract Constant value_0_0065=>"0.0065".
10  Extraction "Basic.ml"
11    p rho C_T C_d C_M K_E K_T
12    get_E_a_by_N get_E_a_by_T get_G_maxload_by_T get_I_m_by_M get_I_m_by_T get_I_b_by_I_e
13    get_I_b_by_T_b get_I_b_by_U_e get_I_e_by_I_b get_I_e_by_E_a_and_I_m.
```

抽取出的部分 OCaml 代码见表 19.

表 19 抽取出的部分 OCaml 代码

```
1   (** val get_T_by_N: coq_R->coq_R **)
2   let get_T_by_N n=
3     coq_Rmult(coq_Rmult rho c_T)
4       (pow0 d_p(Int.succ(Int.succ(Int.succ 0)))))) (pow0 (rdiv n(iZR(Zpos(XO(XO(XI(XI(XI XH))))))))
    (Int. succ(Int.succ 0)))
5   (** val get_M_by_N: coq_R->coq_R **)
6   let get_M_by_N n=
7     coq_Rmult(coq_Rmult(coq_Rmult c_M rho)
8       (pow0(rdiv n(iZR(Zpos(XO(XO(XI(XI(XI XH))))))) (Int.succ(Int.succ 0))))
9       (pow0 d_p(Int.succ(Int.succ(Int.succ(Int.succ 0))))))
10  (** val get_E_a_by_N: coq_R->coq_R **)
11  let get_E_a_by_N n=coq_Rmult k_E n
12  (** val get_M_by_I_m: coq_R->coq_R **)
13  let get_M_by_I_m i_m=coq_Rmult k_T(rminus i_m i_m0)
```

可以看到, 抽取出的函数定义与 Coq 中的定义是一致的. 其中有一部分函数可读性较差, 比如(Int.succ (Int.succ 0)) 实际上表示数字 2, 而(Zpos(XO(XO(XI(XI(XI XH))))))实际上是 60. 产生复杂代码的原因是, 在 Coq 中自然数和整数等数据类型都是归纳定义, 在 OCaml 中也被翻译成了归纳定义的数据类型, 而代码抽取时保留了归纳定义数据类型的构造方式, 利用构造子的反复使用来表示一个具体数据. 虽然 OCaml 标准库支持的数据类型相较于 Coq 标准库要丰富很多, 包含整数、浮点数、向量、矩阵等各种类型, 但基于 Coq 验证时需要从更底层来构建各种数据类型及其运算和性质. 如何将 Coq 中的自定义数据结构更友好地映射到 OCaml 中的基本数据类型, 这是一个值得研究的课题. 一种办法是在 Coq 中使用公理化方式来抽象表示这些基本数据类型, 但需要非常谨慎地分析引入的每一条公理是否会产生矛盾. 需要指出的是, 目前这些冗长的

代码不会影响到程序的正确性.

我们测试了最长悬停时间求解算法在 OCaml 中的运行效果. 其执行结果见表 20.

表 20 最长悬停时间问题的 OCaml 运行结果

1	val t: float=3.675
2	val n: float=5223.79614376495465
3	val i_m: float=6.53286739462531
4	val u_m: float=6.49961283521551803
5	val u_eo: float=6.55187577437252067
6	val sigma_e: float=0.545989647864376648
7	val i_e: float=3.56687796833614046
8	val u_e: float=11.8473248812665553
9	val i_b: float=15.2675118733445618
10	val t_b: float=15.7196537321195251

上述符号和数据对应到具体物理量的解释, 见表 21.

表 21 最长悬停时间问题的 OCaml 结果解释

1	螺旋桨推力 T=3.675 N
2	螺旋桨转速 N=5223 r/min
3	电机等效输入电流 I_m=6.53 A
4	电机等效输入电压 U_m=6.49 V
5	电调等效输出电压 U_eo=6.55 V
6	电调输入电流 I_e=3.57 A
7	电调输入电压 U_e=11.58 V
8	电调油门 $sigma\_e$=0.54=54%
9	电池输出电流 I_b=15.27 A
10	电池持续时间 T_b=15.71 min

这些结果与文献[1]中的数据吻合. 也就是说, 使用由 Coq 定理证明器抽取出的 OCaml 程序, 将原文献给出的已知条件作为 OCaml 程序输入, OCaml 程序计算出的结果与该文献中给出的结果是一致的. 我们已经测试了原文献给出的所有 4 组案例的数据. 从而说明了这里的求解算法能够完成相关功能, 更重要的是, 这些算法是由经验证的 Coq 模型自动生成的代码, 其可靠性远远超过人工编写的程序.

5 总 结

飞控系统的高可靠性是飞行器安全可靠运行的必要前提, 而飞控系统软件的正确性是基本前提. 飞控中大量的模型、公式、算法基本都可以归结为数学知识. 传统上这些知识是由人工推演来保证其正确性, 而随着系统复杂性的增加使得人工推演结果的可靠性越来越低. 将所有这些领域知识进行形式化描述, 建立数学模型并给出推导过程的证明, 将确保生成可靠的数学模型. 同时形式化的数学模型还为自动生成软件代码提供了良好的基础, 从而有助于构建出可靠的软件系统.

本文对文献[1]中多旋翼飞控的推进子系统形式验证后, 可说明在基础理论(以定义或公理的方式给出的来自于参考文献的内容)正确的前提下本文给出的函数库正确. 本文的验证结论对推进子系统设计具有辅助作用, 对其正确性具有支撑作用. 一方面, 飞控推进子系统设计人员只需关注少量基础理论的正确性, 而大量推理过程已经完成, 故而工作量大为减少; 另一方面, 传统上由自然语言描述可能带有歧义的工程知识以及人工易出错的推理过程可能带来潜在的设计缺陷, 而本文的验证工作可消除这类隐藏缺陷.

本文在形式验证工作中也发现或解决了一些问题. 第一, 飞控中的变量是有具体含义且带有单位的物理

量, 不同于数学中抽象的数, 使用不当可能带来软件缺陷. 物理量的单位复杂、多样(参考 SI 系统), 而数学推导过程中一般会省略单位, 在使用具体物理定律时若单位不匹配需进行额外的单位转换, 这类转换若遗漏或出错则可能导致设计缺陷. 使用形式验证可以建立完整的带单位演算系统, 从而消除这类潜在的设计缺陷, 解决这一问题是我们正在进行的另一个研究课题. 第二, 已发现文献中的一些错误, 比如文献[1]中公式(5.12)的正负号错误问题, 已与原作者联系并确认. 此类错误若不进行详细推导很难发现, 而使用定理证明, 则能够发现并纠正这类错误. 文献[1]中原本编号为(5.12)的公式, 即本文中公式(27), 用矩形圈出的两项有错误. 该公式表示从机体坐标系到地球固连坐标系的转换矩阵 R_b^e 在俯仰角 θ 取两个极值时的情形.

$$\text{当}\theta=\pm\frac{\pi}{2}\text{时, } R_b^e=\begin{bmatrix}0 & -\sin(\psi\mp\phi) & \boxed{\cos(\psi\mp\phi)}\\ 0 & \cos(\psi\mp\phi) & \boxed{\sin(\psi\mp\phi)}\\ \mp1 & 0 & 0\end{bmatrix} \tag{27}$$

纠正后, 正确的公式为公式(28)和公式(29).

$$\text{当}\theta=\frac{\pi}{2}\text{时, } R_b^e=\begin{bmatrix}0 & -\sin(\psi-\phi) & \boxed{\cos(\psi-\phi)}\\ 0 & \cos(\psi-\phi) & \boxed{\sin(\psi-\phi)}\\ -1 & 0 & 0\end{bmatrix} \tag{28}$$

$$\text{当}\theta=-\frac{\pi}{2}\text{时, } R_b^e=\begin{bmatrix}0 & -\sin(\psi+\phi) & \boxed{-\cos(\psi+\phi)}\\ 0 & \cos(\psi+\phi) & \boxed{-\sin(\psi+\phi)}\\ 1 & 0 & 0\end{bmatrix} \tag{29}$$

本文模型所用的简化主要有: 飞行高度和环境温度使用常量, 螺旋桨气动系数使用常量, 机载其他设备电流使用常量, 电池放电模型简化. 它们均来自文献[1], 对验证结论没有影响, 对实际系统而言会有一定的精度误差. 如下几方面值得讨论. 第一, 关于飞行高度和环境温度使用常量, 由于在低空飞行时飞行高度和环境温度变化不大, 且这两个参数的变化对其余参数的影响很小, 不影响验证结论. 第二, 螺旋桨气动参数选用的是适用性较广的平均参数, 不影响验证结论, 而实际的螺旋桨因不同型号和工艺有所不同, 可用实际值代入来求解. 第三, 电池放电模型简化, 这一简化可能给实际系统带来一定的精度问题. 由于电池放电过程中电量释放是非线性的, 而且电池电压也会逐步下降, 但这里做了电量线性化下降和电压保持不变的假设, 所以这一假设会使得此处求解结果与实际系统运行结果有一定的误差.

本文将 Coq 定理证明技术应用到飞控的设计中, 对文献[1]中多旋翼无人机的推进子系统进行了完整的形式化验证. 主要贡献有:

(1) 对文献[1]中多旋翼无人机推进子系统的领域知识进行了整理和封装, 建立了具有层级结构以适用于形式验证的文档, 区分了常量、变量, 分离了基本函数、复合函数, 给出了变量关系图, 提出了计算成本更低的最简形式函数概念, 给出了最简形式函数汇总表, 给出了约 77 个函数定义.

(2) 在 Coq 系统中对上述内容进行了形式化描述, 建立了相应的函数、引理、证明策略等, 并对基本反函数、复合函数和最简形式函数的推导正确性进行了证明.

(3) 对两个典型的求解问题给出了算法示例, 抽取了 OCaml 程序, 并执行后得到了与参考文献中相同的结果. 其余场景可类比这两个算法示例使用已验证函数进行快速设计.

(4) 最终形成了一个可用的高可靠函数式软件库, 并将领域知识文档和各种软件代码开源. 本文主要工作定量说明如下: Tex 文件约 1 000 行, 用于生成飞控知识文档, 以及在线文档, 地址为 http://fem-nuaa.cn/FML4FCS/PropulsionSystem/; Coq 脚本文件约 1 300 行(原始版本约 3 000 行, 因使用自定义策略而大幅简化了证明), 见代码库 https://gitee.com/fem-nuaa/coq_flyctrl; 从 Coq 脚本抽取的 OCaml 代码约 5 000 行.

(5) 本文的研究方法为相似的控制系统形式化验证提供了一个研究样例. 同类系统可参照本文流程进行: 领域知识的形式化文档建立; 概念、函数和性质的形式化定义; 性质的证明; 利用已证明的正确函数进行实际问题的算法设计; 生成可用的计算机程序.

另外, 这项研究工作也面临一些挑战, 需要进一步加以研究. 分析如下.

(1) 飞控领域知识的形式化文档构建过程比较困难. 既需要熟悉自动控制领域的飞控专业知识, 又需要熟悉计算机领域的形式化技术. 而这两项技能需要跨学科的研究, 并且都需要投入大量精力.

(2) 飞控系统所需要的很多基础数学理论在目前的 Coq 系统中不够完善. 比如矩阵、复数、四元数、微积分等这些内容在 Coq 中尚不完善, 需要开发者自行实现所欠缺的功能, 或者是从第三方库进行移植. 要构建出飞控系统所需的所有基础理论库, 本身就是一项复杂的工作, 需要投入大量精力.

(3) 目前 OCaml 程序直接用于控制系统较为困难. OCaml 程序是函数式编程语言, 它是一种带有自动内存分配和垃圾回收功能的高级语言, 其运行过程占用的内存资源和处理器时间可能难以精确量化, 并且目前还没有在嵌入式环境下运行 OCaml 程序的方法. 一种思路是将来从 Coq 或 OCaml 直接生成实用性更强的 C 程序, 另一种思路是借鉴 HACMS 项目中的 Ivory 语言而设计在 Coq 和 OCaml 环境下的领域专用语言. 但值得说明的是, 虽然以往 OCaml 语言在工业控制中应用较少, 但这种状况正在发生改变. 因为人们认识到函数式语言可以为控制系统提供更好的安全性, 美国 DARPA 的 HACMS 项目就是一个有力的见证. 由此可见, 未来函数式语言将越来越多地被工业控制行业所接受.

总之, 使用 Coq 定理证明技术来设计飞控, 能够建立严谨的飞控数学模型, 有利于将来的代码生成, 可以从根本上提高系统的可靠性. 未来, 还将研究命令式程序生成技术以便能够生成实用性更强的 C 语言程序, 以及对更多的飞控模块进行基于定理证明的设计和验证, 从而建立一个完整的高可靠飞控系统.

致谢 感谢范永乾、麻莹莹、邓昊以及讨论班同学对本课题提出的宝贵意见和给予的各种支持. 感谢评审人对本文的内容完整性、专业用语严谨性和相关工作对比分析等多方面给予的宝贵建议.

References:

[1] Quan Q. Introduction to Multicopter Design and Control. Springer, 2017. [doi: 10.1017/aer.2018.133]

[2] Quan Q. Introduction to Multicopter Design and Control. Beijing: Publishing House of Electronics Industry, 2018 (in Chinese).

[3] Phillip J, Harris R. The Boeing 737 MAX Saga: Lessons for software organizations. Software Quality Professional, 2019, 21(3): 4–12.

[4] Ducard GJJ. Fault-tolerant flight control and guidance systems for a small unmanned aerial vehicle. ETH Zurich, 2007. [doi: 10.3929/ETHZ-A-005582839]

[5] Xu M. The Basis of Air and Gas Dynamics. 2nd ed., Northwestern Polytechnical University Press Co. Ltd., 2015 (in Chinese).

[6] Wang J, Zhan NJ, Feng XY, Liu ZM. Overview of formal methods. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 33−61 (in Chinese with English abstract). http://www.jos.org.cn/10000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[7] Chen G, Shi ZP. Formalized engineering mathematics. Communications of the CCF, 2017, 13(10) (in Chinese with English abstract).

[8] Chen G Formalized mathematics and proof engineering. Communications of the CCF, 2016, 12(9) (in Chinese with English abstract).

[9] Fisher K, Launchbury J, Richards R. The HACMS program: USIng formal methods to eliminate exploitable bugs. Philosophical Trans. of the Royal Society A, 2017, 375: 20150401. [doi: 10.1098/rsta.2015.0401]

[10] Shi D, Dai X. A practical performance evaluation method for electric multicopters. IEEE/ASME Trans. on Mechatronics, 2017, 22(3): 1337–1348. [doi: 10.1109/TMECH.2017.2675913]

[11] Boldo S, Lelay C, Melquiond G. Coquelicot: A user-friendly library of real analysis for Coq. Mathematics Computer Science, 2015, 9(1): 41–62. [doi: 10.1007/s11786-014-0181-1]

[12] Mathematical Components. https://math-comp.github.io [doi: 10.5281/zenodo.4457887]

[13] Abed S, Rashid A, Hasan O. Formal analysis of unmanned aerial vehicles using higher-order-logic theorem proving. Journal of Aerospace Information Systems, 2020, 17(9): 481–495. [doi: 10.2514/1.I010730]

[14] Vermaelen H, Dinh T, Holvoet T. Formal verification of autonomous UAV behavior for inspection tasks using the knowledge base system IDP. In: Demazeau Y, Holvoet T, Corchado JM, Costantini S, eds., Advances in Practical Applications of Agents, Multi-Agent Systems, and Trustworthiness. The PAAMS Collection, Vol.12092. Cham: Springer International Publishing, 2020. 315–326. [doi: 10.1007/978-3-030-49778-1_25]

[15] Pollien B, Garion C, Hattenberger G, Roux P, Thirioux X. Verifying the Mathematical Library of an UAV Autopilot with Frama-C. In: Proc. of the Int'l Conf. on Formal Methods for Industrial Critical Systems. 2021. 167–173. [doi: 10.1007/978-3-030-85248-1_10]

[16] Ma ZW, Chen G. Matrix formalization based on Coq record. Computer Science, 2019, 46(7): 139–145 (in Chinese with English abstract). [doi: 10.11896/j.issn.1002-137X.2019.07.022]

[17] Ma YY, Ma ZW, Chen G. Formalization of operations of block matrix based on Coq. Ruan Jian Xue Bao/Journal of Software, 2021, 32(6): 1882–1909 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6255.htm [doi: 10.13328/j.cnki.jos.006255]

[18] Wang YF, Chen G. Formalization of Laplace transform in Coq. In: Proc. of the 2017 Int'l Conf. on Dependable Systems and Their Applications (DSA). 2017. 13–21. [doi: 10.1109/DSA.2017.12]

[19] Wang YF, Chen G. A formal proof of two properties of Laplace transform. In: Proc. of the 2018 IEEE Int'l Conf. of Safety Produce Informatization (IICSPI). Chongqing, 2018. 883–887. [doi: 10.1109/IICSPI.2018.8690475]

[20] Ma ZW, Chen G. Formal derivation and verification of coordinate transformations in theorem prover Coq. In: Proc. of the 2017 Int'l Conf. on Dependable Systems and Their Applications (DSA). 2017. 127–136. [doi: 10.1109/DSA.2017.29]

[21] Bertot Y, Casteran P. Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions. Springer, 2004.

[22] Coq Development Team. The Coq Reference Manual 8.13.2.

[23] Pierce BC, *et al*. Software Foundations Volume 1. https://softwarefoundations.cis.upenn.edu/

附中文参考文献:

[2] 全权. 多旋翼飞行器设计与控制. 北京: 电子工业出版社, 2018.

[5] 徐敏. 空气动力学基础. 第 2 版, 西安: 西北工业大学出版社, 2015.

[6] 王戟, 詹乃军, 冯新宇, 刘志明. 形式化方法概貌. 软件学报, 2019, 30(1): 33–61. http://www.jos.org.cn/10000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[7] 陈钢, 施智平. 形式化工程数学. 中国计算机学会通讯, 2017, 13(10).

[8] 陈钢. 形式化数学和证明工程. 中国计算机学会通讯, 2016, 12(9).

[16] 马振威, 陈钢. 基于 Coq 记录的矩阵形式化方法. 计算机科学, 2019, 46(7): 139–145.

[17] 麻莹莹, 马振威, 陈钢. 基于 Coq 的分块矩阵运算的形式化. 软件学报, 2021, 32(6): 1882–1909. http://www.jos.org.cn/1000-9825/6255.htm [doi: 10.13328/j.cnki.jos.006255]

石正璞(1986－), 男, 博士生, CCF 学生会员, 主要研究领域为形式化工程数学, Coq 定理证明, 飞行控制系统, 硬件设计, 嵌入式系统.

谢果君(1992－), 男, 博士生, CCF 学生会员, 主要研究领域为形式化工程数学, 控制系统形式化, 定理证明.

崔敏(1986－), 女, 博士生, 主要研究领域为形式化工程数学, 飞行控制基础理论研究, 定理证明, 云计算.

陈钢(1958－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为形式化工程数学, COQ 定理证明, 函数式语言, 类型系统, 形式化方法, 控制系统.