

班 级 1802012
学 号 18020100198

西安电子科技大学

本科毕业设计论文



题 目 基于无人机的自主航点飞行与双光热成像

目标监测算法研究

学 院 电子工程学院

专 业 电子信息工程

学生姓名 吕瑞涛

导师姓名 李隐峰

毕业设计（论文）诚信声明书

本人声明：本人所提交的毕业论文《基于无人机的自主航点飞行与双光热成像目标监测算法研究》是本人在指导教师指导下独立研究、写作成果，论文中所引用他人的无论以何种方式发布的文字、研究成果，均在论文中加以说明；有关教师、同学和其他人员对本文本的写作、修订提出过并为我在论文中加以采纳的意见、建议，均已在我的致谢辞中加以说明并深致谢意。

本文和资料若有不实之处，本人承担一切相关责任。

论文作者：_____ (签字) 时间： 年 月 日
指导教师已阅：_____ (签字) 时间： 年 月 日

摘 要

在过去的十年中，四旋翼飞行器的受欢迎程度稳步上升。它们被用于军事、消防和救援巡检等领域，并随着公众的普及，逐渐进入民用级、消费级市场。目前以DJI为主的商业公司已具备完善的软硬件体系，但由于其商业性质较浓厚，缺乏对于开源协议和开源地面站的兼容。与此同时，**Ardupilot**和**PX4**作为飞控开源界最大的两个项目，采用**QGroundControl**作为地面站软件并通过**MAVLink**协议实现空地间的数据传输。因此，市面上迫切需要一种解决方案将开源地面站的相关功能与DJI飞行器MSDK接口进行结合，并通过**MAVLink**协议开放其飞行控制接口。

卡尔曼滤波器(KF)作为目前较为优秀的滤波和最优估计算法，对于检测后的降落坐标点进行最优估计，最大程度上降低了因误识别或扰动等导致的降落坐标误差，提升了飞行器的降落精准性。传统的基于比例、积分、微分(**PID**)的控制器存在控制超调、振荡等因素，控制效果随着飞行器的负载和外界扰动的变化而变化。**模型预测控制(MPC)**做为替代方案，它能够通过最小化参考值和预测输出之间的误差来生成控制系统所需的输入序列，从而根据特定的飞行器的数学模型设计特定的控制模型，从根本上避免了PID的超调、振荡的问题。

为了实现无人机的自主航点飞行巡检以及视觉引导自动降落，并结合开源地面站的功能，本课题利用DJI御2行业进阶版无人机，使用**DJI MSDK**接口，利用无人机端侧遥控器通过VPN以太网数据链路远程组网传输**MAVLink**协议帧信息至远端的**QGC**地面站软件。无人机根据地面站给出的航点信息进行航点飞行，航点飞行结束后视觉识别降落点并使用卡尔曼滤波器得到最优估计后的降落点坐标，进一步基于**MPC**控制引导降落至特定降落目标点。

关键词：**MPC控制** **DJI MSDK** **MAVLink** **目标检测**

摘要

Abstract

Over the past decade, the popularity of quadcopters has steadily increased. They are used in the fields of military, fire and rescue inspection, and gradually enter the civilian and consumer markets with the popularity of the public.

At present, commercial companies based on DJI have a perfect software and hardware system, but due to their strong commercial nature, they lack compatibility with open source protocols and open source ground stations. At the same time, Ardupilot and PX4, as the two largest projects in the open source industry of flight control, use **QGroundControl** as ground station software and achieve data transmission between air and ground through the **MAVLink protocol**. Therefore, there is an urgent need for a solution that combines the relevant functions of open source ground stations with the MSDK interface of DJI aircraft and opens up its flight control interface through the MAVLink protocol.

Kalman filter (KF) as the current excellent filtering and optimal estimation algorithm, the optimal estimation of the detected landing coordinate point, to the greatest extent to reduce the landing coordinate error caused by misidentification or disturbance, etc., improve the landing accuracy of the aircraft. Traditional **scale, integration, differentiation (PID)** controllers have control overshoot, oscillation and other factors, and the control effect changes with the load of the aircraft and the change of external disturbances.

Model Predictive Control (MPC) as an alternative, it can generate the input sequence required by the control system by minimizing the error between the reference value and the prediction output, so as to design a specific control model according to the mathematical model of a specific aircraft, fundamentally avoiding the problem of overshoot and oscillation of PID.

In order to realize the autonomous waypoint flight inspection and visual guidance automatic landing of the UAV, and combined with the functions of the open source ground station, this project uses the DJI Royal 2 Industry Advanced UAV, using **DJI MSDK interface**, using the drone end-side remote control to remotely network through

Abstract

the VPN Ethernet data link to transmit the MAVLink protocol frame information to the remote QGC ground station software. The UAV conducts a waypoint flight according to the waypoint information given by the ground station, visually identifies the landing point after the waypoint flight and uses the Kalman filter to obtain the optimal estimated landing point coordinates, and further guides the landing to a specific landing target point based on MPC control.

Key words: MPC Control DJI MSDK MAVLink Object Detection

目 录

第一章 绪论

1.1 研究背景

无人机（UAV）是无人航空器（**Unmanned Aerial Vehicle**）的简称，是一种不载操作人员、用空气动力产生运载工具升力、能够自主或遥控飞行、能够一次使用或回收并且载有杀伤或非杀伤有效载荷的有动力的航空器。总的来说，分为固定翼无人机、无人直升机和多旋翼无人机三大类。多旋翼无人机体积较小，成本较低，与固定翼无人机相比，具有可定点悬停、可垂直起降等优势，被应用于电力巡线、救灾探测、勘探测绘、人员搜救等用途。

在安防、救援、火情检测等方面，多旋翼飞行器正发挥着强大的作用，目前以**DJI**为主的多旋翼飞行器已经实现了包括自动起飞/航点巡航/视觉检测/自主目标追踪等功能。随着行业化应用的普及，无人机逐渐由以航拍器为主的娱乐化和影视行业发展至工业化检测等应用，并作为空中飞行平台，搭载其他行业应用负载设备，实现特定行业化功能。

在过去的几十年里，红外热成像成为工业和建筑领域的重要话题。这方面的最新发展之一是从空中进行红外热成像检测。红外热像仪与无人机相结合对检测光伏系统等热敏设备尤其有用。对于红外热成像检测领域，由于受制于无人机本身尺寸的限制，很难使用720P及更高分辨率的热成像相机，进一步导致了拍摄的热源图较为模糊，难以辨认局部细节。因此，目前主流的行业无人机普遍采用双光相机方案，即：**变焦可见光相机+红外热成像相机一体化方案**。利用机械或电子变焦的可见光镜头可以细致描绘视角内的影像的局部细节，进而同步查看红外热像相机的影像，两者比对后综合进行检测与分析。

根据市面上已有的成品无人机或无人机开发套件，我们发现，目前以**DJI**为主的行业双光无人机存在不兼容开源地面站、自有协议难以实现通用等问题。

1.2 国内外进展

20世纪以来，随着飞行控制算法技术和飞行控制硬件的发展，无人飞行器技术使传统的地面检测技术得以拓展，进入了空中检测的时代并随着算法和硬件的不断升级而快速发展^[?]。其中，国内有以DJI为主的行业级多旋翼飞行器以及配套的司空地面站平台，也有国外以开源为主的平台（如APM、PIX飞控系统和对应的QGroundControl、MissionPlanner等开源地面站），以及由国内其他公司研发与设计的基于STM32或TI嵌入式芯片为主的简易平台（如无名飞控、匿名飞控以及其他相关平台）。

如下对比介绍了一些知名的配套平台：

1.DJI-商业多旋翼飞行器平台。对于整机飞行系统，DJI采用御/晓/精灵/经纬等系列无人机，搭配DJI GO系列APP实现遥控器端控制飞行以及相关飞行参数获取和控制拍照摄影等。DJI不提供针对于个人用户的电脑端配套地面站、云服务以及相关技术支持。对于企事业单位，DJI提供以司空系统为主的飞行器管理平台。提供基本的多无人机控制功能以及DJI官方提供的云上解决方案，不提供地面站的开发接口，无法进行定制化开发与服务，并且由于司空系统的封闭性，对于云端也同样无法进行开发。不支持MAVLink微型空中飞行器链路通讯协议。

2.Ardupilot/PX4-开源地面站平台。ArduPilot和PX4是两大主流开源无人机系统，前者由DroneCode基金会维护，后者由Auterion公司维护，ArduPilot历史要远长于PX4，因此功能更丰富，有“功能完善、运行稳定”的优势。如图??所示。



图 1.1 Dronecode软硬件平台展示

Ardupilot与PX4在一些关键算法上是相互借鉴的，因此算法先进程度差不多。PX4由于起步晚，历史包袱少，最初就搭建了一个很先进的架构，因此获得了代

码简洁易懂易懂的优势。该开源系统在国内外影响力较大，除多旋翼外同样支持固定翼、垂直起降等机型。支持MAVLink微型空中飞行器链路通讯协议。

3. STM32/TI-嵌入式芯片飞控平台。该系列以匿名、无名飞控为主，最初是由大学生创业团队设计并开发的，具有光流定位、GPS定点等功能。只支持多旋翼系列机型，并且由于其配套地面站程序较为简易，主要适用于教学实验等领域，不适用于商业等对稳定性要求较高的环境中。对于某些特殊的机型，部分兼容MAVLink微型空中飞行器链路通讯协议。

1.3 项目目标

传统的PID位置控制算法由于是基于无模型控制，随着P、I、D三个参数的变换，控制系统存在振荡或响应较慢等现象。因此，我们在PID的基础上，使用基于线性模型(MPC)的预测控制算法。该算法可以有效地克服过程的不确定性、非线性和并联性，并能方便的处理过程被控变量和操纵变量中的各种约束。

因此，本课题利用DJI御2行业进阶版无人机，在自研的APP内通过使用DJI MSDK接口^[?]，将DJI MSDK协议信息转换至MAVLink UDP协议帧信息，并利用无人机端侧遥控器通过以太网数据链路远程组网传输至Windows端侧QGroundControl软件。

由于国内外以开源为主的飞行器大多采用以MAVLink协议为主的通信解决方案，因此该方案可实现将DJI系列飞行器与开源飞行器进行地面站统一化管理与云上部署，显著地提升了飞行器间的兼容性与可靠性。

无人机根据兼容**MAVLink (Micro Air Vehicle Link, 微型空中飞行器链路通讯协议)**协议的QGC地面站给出的GPS航点信息进行航点飞行，飞行任务结束后，检测部分使用基于OpenCV霍夫圆特征检测+自适应阈值，并将检测后的中心点传入至卡尔曼滤波器中，得到的最优估计坐标点作为飞行器的降落点位置参考。控制部分采用MPC (Model Predictive Control) 位置控制，利用上一步估计的坐标点实现自主引导降落。

1.4 本文安排

本文第一章介绍了国内外开源飞控系统以及以DJI为主的商业飞控系统的相关

架构，分析了基于MAVLink协议的通信可行性，并概括说明了本文的研究内容。

本文第二章介绍了项目所需的相关技术和算法，包括OpenCV形态学处理+自适应阈值+霍夫圆检测，以及卡尔曼滤波器在降落点最优估计的应用，以及MPC模型预测控制。

本文第三章分析了APP中集成的项目中关键算法的实现过程。

本文第四章阐述了软硬件在环仿真以及实际试飞效果的测试。

最后一章概述了项目的总体结构，对已完成的工作的分析，以及对于未来工作的展望。

第二章 关键技术和算法

2.1 飞控平台

DJI MSDK拥有一系列API来控制飞机的软件和硬件接口。为开发人员提供了开源生产示例和教程，以便在移动设备上开发更具竞争力的无人机解决方案，广泛应用于能源、测绘、工农业、科教等领域，并提高了MSDK应用开发的体验和效率。

MSDK提供的功能接口有：飞行器参数获取和设置，飞行控制，应用数据处理以及其他相关功能。支持二次开发航点自动飞行，虚拟摇杆飞行，并实时获取RTK、GPS定位信息，以及飞行器状态参数。支持获取实时H.264高码率视频流，并可以利用OpenCV计算机视觉库完成特征检测与分析。

2.2 视觉检测

根据项目需求，无人机检测地面上面的H标记采用基于OpenCV的框架，算法使用自适应阈值+形态学处理+霍夫圆检测，并返回检测的中心点，用于滤波以及后续的位置优化。

2.2.1 形态学处理

形态学处理包括开闭运算、礼帽/顶帽/黑帽、检测边和角点等。这里我们为了获取图像中的主要对象，对一副二值图连续使用闭运算和开运算。

1. 腐蚀和膨胀

腐蚀的基本思想就像土壤侵蚀一样，它腐蚀了前景对象的边界（始终尝试将前景保持在白色）。它的作用为：内核在图像中滑动（如在2D卷积中）。仅当内核下的所有像素均为1时，原始图像中的像素（1或0）才会被视为1，否则将被腐蚀（变为零）。

膨胀与腐蚀正好相反。此处，如果内核下的至少一个像素是“1”，则像素元

素为“1”。所以它增加了图像中的白色区域或前景对象的大小增加。通常，在消除噪音的情况下，侵蚀后会扩张。因为，侵蚀可以消除白噪音，但它也会缩小我们的物体。所以我们扩张它。由于噪点消失了，它们不会再回来了，但是我们的物体面积会增加。它在连接对象的损坏部分时也很有用。

上述过程如图??所示：

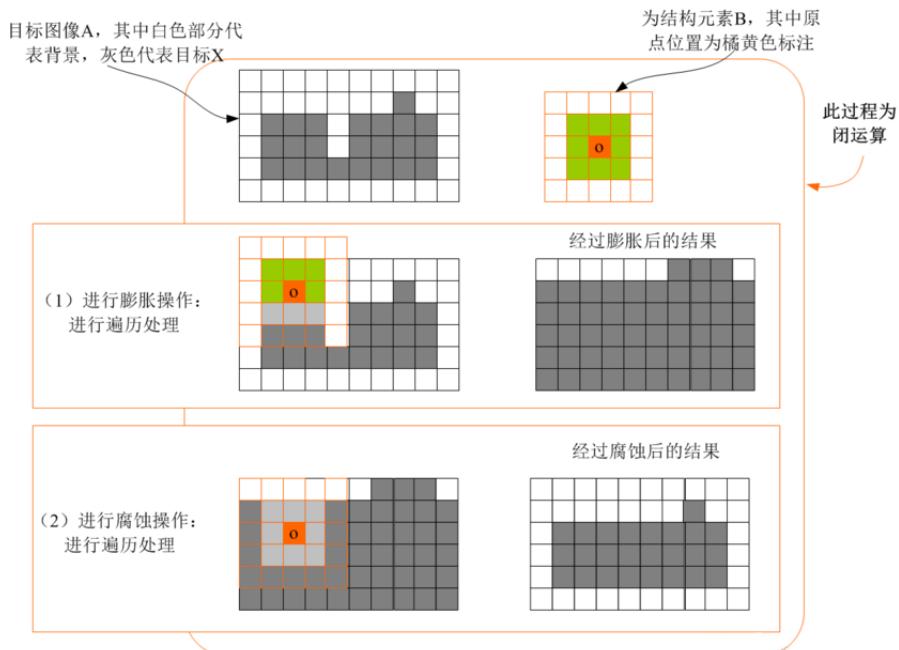


图 2.1 腐蚀和膨胀变换示意

2.开闭运算

开运算和**闭运算**就是将腐蚀和膨胀按照一定的次序进行处理。但这两者并不是可逆的，即先开后闭并不能得到原先的图像。

具体流程为：

开运算：先腐蚀后膨胀，用于移除由图像噪音形成的斑点；**闭运算**：先膨胀后腐蚀，用来连接被误分为许多小块的对象。

OpenCV实现形态学运算的函数为：morphologyEx

该函数实现的功能是一些基于图像形状的简单操作。它通常在二进制映像上执行。它需要两个输入，一个是我们的原始图像，第二个称为结构元素或内核，它决定了操作的性质。两个基本的形态学运算符是腐蚀和膨胀。然后它的变体形式，如打开，关闭，渐变等也开始发挥作用。

2.2.2 霍夫圆检测

1.霍夫变换

霍夫变换(Hough Transform, HT)于1962年由Paul Hough首次提出，后于1972年由Richard Duda和Peter Hart推广使用，经典霍夫变换来检测图像中的直线，是模式识别领域中对二值图像进行直线检测的有效方法^[?]，后来霍夫变换扩展到任意形状物体的识别，多为圆和椭圆，如图??所示。

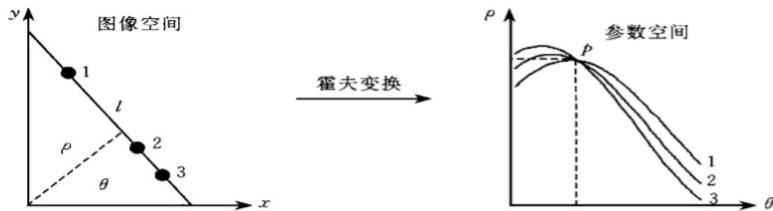


图 2.2 霍夫变换示意

在标准参数化方式下，图像空间中的直线1的表达为：

$$d = x \cos \theta + y \sin \theta, \quad d \geq 0, 0 \leq \theta < \pi \quad (2-1)$$

右边的图意思是在一个点上，不同的 θ 对应不同的 ρ ， θ 是穿过该点的直线的斜率， ρ 是穿过该点的直线与原点的距离。这样穿过每个点作无数条直线，参数 ρ 和 θ 就能绘出一个正弦图像。而如果1,2,3在一条直线上就可以找到一个 θ 使得过1,2,3点的直线的 ρ 很接近。也就是有图上相交的过程。而通常使用的方法是累加法，求得累加次数最多的 (θ, ρ) 从而确定这条直线。

4.霍夫变换用于圆检测

与使用 (r, θ) 来表示一条直线相似，使用 (a, b, r) 来确定一个圆心为 (a, b) 半径为 r 的圆。

某个圆过点 (x_1, y_1) ，则有： $(x_1 - a)^2 + (y_1 - b)^2 = r^2$

那么过点 (x_1, y_1) 的所有圆可以表示为 $(a_1(i), b_1(i), r_1(i))$ ，其中 $r_1 \in (0, \infty)$ ，每个 i 值都对应一个不同的圆， $(a_1(i), b_1(i), r_1(i))$ 表示了无穷多个过点 (x_1, y_1) 的圆。

过点 (x_1, y_1) 的所有圆可以表示为 $(a_1(i), b_1(i), r_1(i))$ ，过点 (x_2, y_2) 的所有圆可以表示为 $(a_2(i), b_2(i), r_2(i))$ ，过点 (x_3, y_3) 的所有圆可以表示为 $(a_3(i), b_3(i), r_3(i))$ ，如果这三个点在同一个圆上，那么存在一个值 (a_0, b_0, r_0) ，使得 $a_0 = a_1(k) = a_2(k) =$

$a_3(k)$ 且 $b_0 = b_1(k) = b_2(k) = b_3(k)$ 且 $r_0 = r_1(k) = r_2(k) = r_3(k)$ ，即这三个点同时在圆 (a_0, b_0, r_0) 上。

图??可以形象的看出。

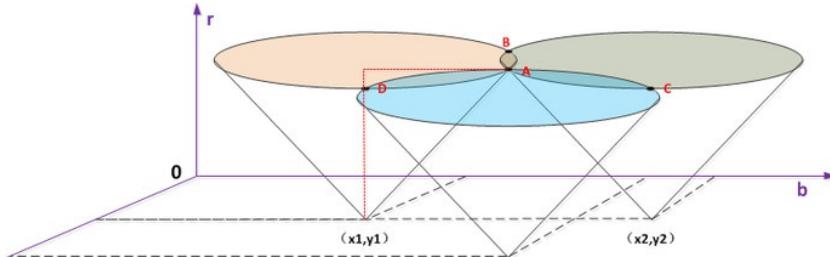


图 2.3 霍夫圆示意

先，分析过点 (x_1, y_1) 的所有圆 $(a_1(i), b_1(i), r_1(i))$ ，当确定 $(r_1(i))$ 时， $(a_1(i), b_1(i))$ 的轨迹是一个以 $(x_1, y_1, r_1(i))$ 为中心半径为 $r_1(i)$ 的圆。那么，所有圆 $(a_1(i), b_1(i), r_1(i))$ 的组成了一个以 $(x_1, y_1, 0)$ 为顶点，锥角为90度的圆锥面。三个圆锥面的交点A即是同时过这三个点的圆。累加次数最多的A点就是一个圆心。

2.3 滤波优化

对于上一步每一次检测到的降落板中心点，由于检测中存在一定的服从高斯分布的噪声，准确检测的降落点和这些噪声点同时作为参考点传入至无人机MPC位置控制中，会一定程度上导致无人机的最终位置控制存在较大的振荡与偏差，进而影响最终的降落精准度。

因此，我们需要一种有效的方法尽可能去除检测的降落点中心的高斯噪声，因此这里采用卡尔曼滤波器去除检测中心点的噪声并实现较为精确的降落点预测。

2.3.1 卡尔曼滤波器

卡尔曼滤波（**Kalman filtering, KF**）是一种利用线性系统状态方程，通过系统输入输出观测数据，对系统状态进行最优估计的算法^[?]。卡尔曼滤波的一个典型实例是从一组有限的，对物体位置的，包含噪声的观察序列中预测出物体的坐标位置及速度。在很多工程应用（雷达、计算机视觉）中都可以找到它的身影。同时，卡尔曼滤波也是控制理论以及控制系统工程中的一个重要话题。

卡尔曼滤波最早可以追溯到**Wiener滤波**，不同的是卡尔曼采用状态空间来描述它的滤波器，卡尔曼滤波器同时具有模糊/平滑与预测功能，特别是后者在视频分析与对象跟踪应用场景中被发扬光大，在离散空间（图像或者视频帧）使用卡尔曼滤波器相对简单。假设我们根据一个处理知道一个变量值如下：

$$x_{k+1} = \Phi x_k + w_k \quad (2-2)$$

其中 x_k 是在k时刻的状态， Φ 是从k到k + 1时刻的变换矩阵， w_k 是 k时刻相关白噪声的协方差矩阵。

对观测值建模如下：

$$z_k = Hx_k + v_k \quad (2-3)$$

其中 z_k 是在k时刻对x的实际测量值，H是状态矩阵与测量矩阵无噪声链接， v_k 是测量错误

这样就得到两个协方差矩阵

$$\begin{aligned} Q &= E [w_k w_k^T] \\ R &= E [v_k v_k^T] \\ P_k &= E [e_k e_k^T] = E [(x_k - \hat{x}_k) (x_k - \hat{x}_k)^T] \end{aligned} \quad (2-4)$$

假设 \hat{x}_k 前一个评估为 \hat{x}'_k ，可以得到

$$\hat{x}_k = \hat{x}'_k + K_k (z_k - H\hat{x}'_k) \quad (2-5)$$

其中 K_k 被称为卡尔曼增益，有了它就可以更新测量模型，从而更新状态空间的下个预测。

以上流程可以用图??表示。

2.4 控制算法

根据项目需求，无人机需要在自主航点飞行完毕后自主降落至指定区域，指定降落区域地面上贴有标记为H的橙红色降落板。自主降落下降阶段我们改进了

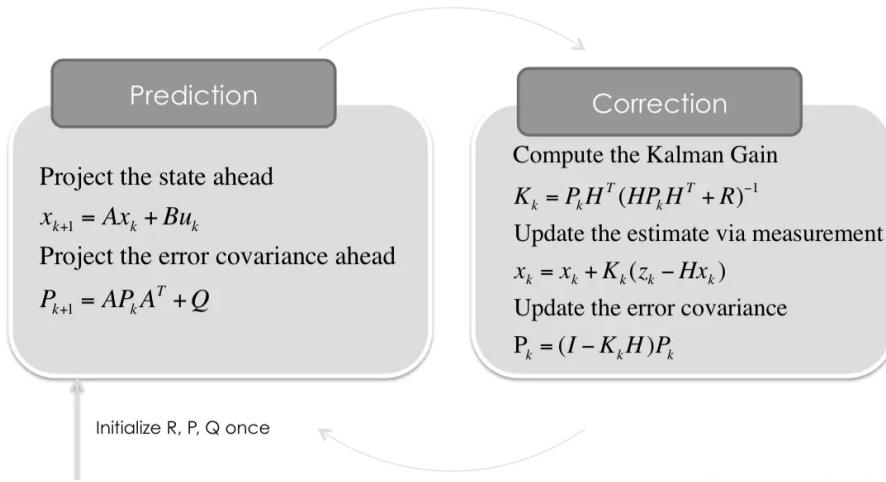


图 2.4 卡尔曼滤波器预测与修正

传统的基于PID的控制方案，通过采用基于MPC模型预测控制的控制方案对控制参数进行自适应优化，从而使得降落控制阶段更为平稳。

2.4.1 PID控制

PID控制作为应用非常广泛的控制算法，小到控制一个元件的温度，大到控制无人机的飞行姿态和飞行速度等等，都可以使用PID控制。**PID(proportion integration differentiation)**其实就是指比例，积分，微分控制，它具有原理简单，易于实现，适用面广，控制参数相互独立，参数的选定比较简单等优点；而且在理论上可以证明，对于过程控制的典型对象——“一阶滞后+纯滞后”与“二阶滞后+纯滞后”的控制对象，PID控制器是一种最优控制^[?]。

图??为常见的PID控制系统结构。

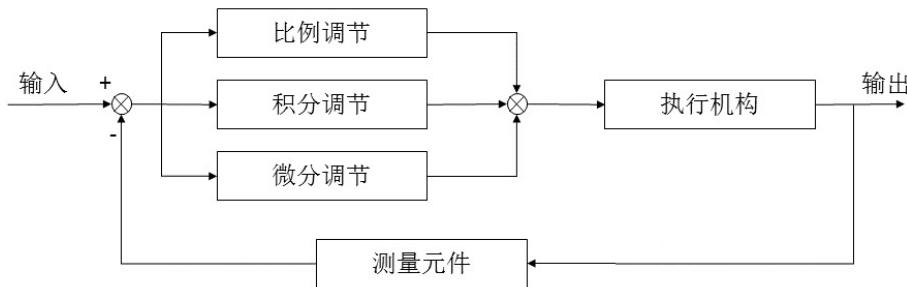


图 2.5 PID控制系统结构图

根据上图所示，PID控制即为：当得到系统的输出后，将输出经过比例，积分，微分3种运算方式，叠加到输入中，从而控制系统的 behavior。

在工业过程中，连续控制系统的理想PID控制规律为：

$$u(t) = K_p \left(e(t) + \frac{1}{T_t} \int_0^t e(t) dt + T_D \frac{de(t)}{dt} \right) \quad (2-6)$$

式中, K_p 为比例增益, 比例度成倒数关系; T_t 为积分时间常数; T_D 为微分时间常数; $u(t)$ 为PID控制器的输出信号; $e(t)$ 为给定值 $r(t)$ 与测量值之差。

针对于传统的PID控制器, 其本身一般不依赖于未来的误差, 这是不对的。如果依赖未来的误差, 也没有考虑车辆的模型, 必然不准确的控制。而对于MPC控制器来说, 延迟的一个重要因素就是执行器的动态特性。例如, 从转向角指令下发到实际到达该转向角度的时间。这个可以通过简单的动态系统进行建模, 并且整合到车辆模型中去。一种方法就是将使用车辆模型, 根据延迟时间推算延迟时间后的车辆状态, 这个车辆状态就是MPC计算的新初始状态。^[?]

2.4.2 MPC控制

1.MPC控制介绍

传统PID算法的缺点是, 对于系统的状态变化, 控制器的输出不能很好的控制系统的状态变化, 因此, 在这种情况下, PID算法的效率不高。而MPC算法则是一种更好的控制方法, 它能够更好的控制系统的状态变化, 并且能够更好的控制系统的输出。

模型预测控制 (Model predictive control、MPC) 是过程控制中, 在满足特定限制条件时, 控制过程的进阶控制方式, 自1980年代起已用在化学工厂及炼油厂的工业过程中。模型预测控制是以过程的动态模型为基础, 多半是透过系统识别得到的线性经验模型。模型预测控制的特点是每一次针对目前的时间区块内作最佳化, 然后下一个时间再针对时间区块内作最佳化, 这和LQR控制器不同。模型预测控制可以预测未来事件并且进行对应的处理, 而传统的PID控制器没有这样的预测功能。预测控制打破了传统控制中对模型结构的严格要求, 更着重于在系统已获取的信息的基础上根据功能要求按照最方便的途径建立模型。尽管模型预测控制形式多种多样, 但都依赖下述三项基本原理:

1.1 预测模型

预控制是一种基于模型的控制算法, 这一校型称为预测模型^[?]。对于预测控制来讲, 只注重模型的功能, 而不注重模型的形式, 预测模型的功能就是能根据

对象的历史信息和未来输入预其未来输出。从方法的角度讲，只要是具有预测功能的信息集合，不论其有什么样的表现形式，均可作为预测模型。因此，状态方程，传递函数这类传统的模型都可以作为预测模型。

1.2 滚动优化

预测控制的主要特征是在线优化。预测控制这种优化控制算法是通过某一性能指标的最优来确定未来的控制作用的。这一性能指标涉及到系统未来的行为，例如，通常可取对象输出在未来的采样点上跟踪某一期望轨迹的方差最小。但也可取更广泛的形式，要求控制能量为最小而同时保持输出在某一给定范围内等。性能指标中涉及到的系统未来的行为，是根据预测模型由未来的控制策略决定的^[2]。

1.3 反馈校正

预控制算法在进行滚动优化时，优化的基点应与系统实际一致。但作为基础的预测模型，只是对象动态特性的粗略描述，由于实际系统中存在的非线性、时变、模型失配、干扰等因素，基于不变模型的预测不可能和实际情况完全相符，这就需用附加的预测手段补充模型的不足，或者对基础模型进行在线修正^[2]。滚动优化只有建立在反馈校正的基础上，才能体现其优越性。因此，预测控制算法在通过优化确定了一系列未来的控制作用后，为了防止模型失配或环境干扰引起控制对理想状态的偏离，并不是把这些控制作用逐一全部实施，而只是实现本时刻的控制作用^[2]。到下一采样时刻，则首先检测对象的实际输出，并利用这一实时信息对基于模型的预测进行修正，然后再进行新的优化^[2]。

方框图^{??}提供了MPC原理的可视化表示：

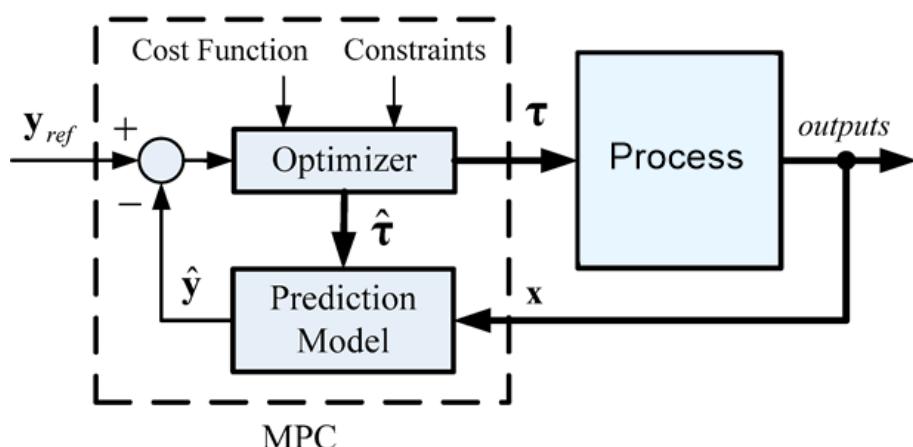


图 2.6 MPC 控制系统结构图

图??说明了MPC中的预测范围:

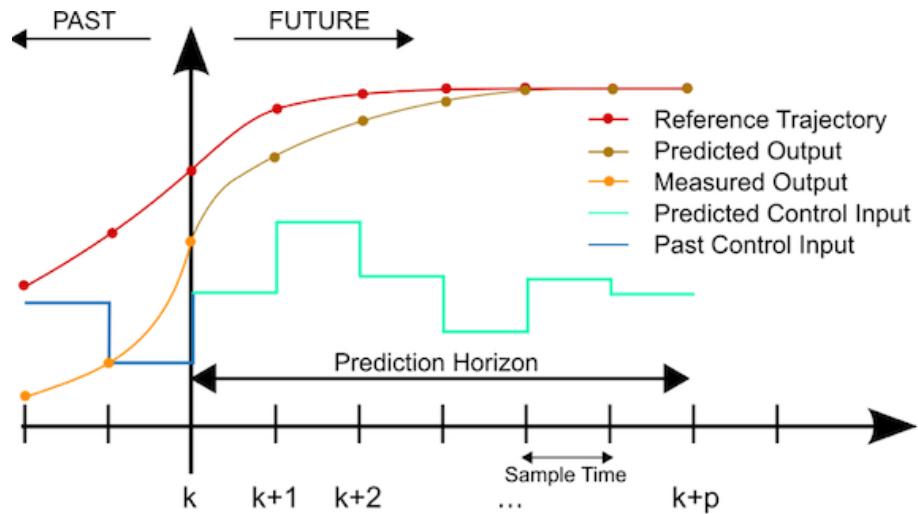


图 2.7 MPC 预测范围

2.MPC控制系统控制器建模

对于以下控制模型:

$$x(k+1) = Ax(k) + Bu(k) \quad (2-7)$$

写成矩阵格式:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0.1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} u(k) \quad (2-8)$$

此处的 x_1 和 x_2 可以理解为误差项

$$X(k) = \begin{bmatrix} x(k | k) \\ x(k+1 | k) \\ \vdots \\ x(k+j | k) \\ \vdots \\ x(k+N | k) \end{bmatrix} \quad (2-9)$$

$$U(k) = \begin{bmatrix} u(k | k) \\ u(k+1 | k) \\ \vdots \\ u(k+i | k) \\ \vdots \\ u(k+N-1 | k) \end{bmatrix} \quad (2-10)$$

对于MPC矩阵：

$$x_k = Mx_k + Cu_k \quad (2-11)$$

$$M = \begin{bmatrix} I_{n \times n} \\ A_{n \times n} \\ A_n^2 \\ \vdots \\ A^N \end{bmatrix} \quad (2-12)$$

$$C = \begin{bmatrix} 0 & 0 & \dots & 0 \\ \vdots & \dots & & \vdots \\ \vdots & 0 & & 0 \\ 0 & 0 & \dots & 0 \\ B_{n \times p} & 0 & \dots & 0 \\ AB_{n \times p} & B & \vdots & 0 \\ \vdots & \vdots & & B \end{bmatrix} \quad (2-13)$$

$$X(k) = \begin{bmatrix} x(k | k) \\ x(k+1 | k) \\ x(k+2 | k) \\ x(k+3 | k) \end{bmatrix} = \begin{bmatrix} x_1(k | k) \\ x_2(k | k) \\ x_1(k+1 | k) \\ x_2(k+1 | k) \\ x_1(k+2 | k) \\ x_2(k+2 | k) \\ x_1(k+3 | k) \\ x_2(k+3 | k) \end{bmatrix} \quad (2-14)$$

$$U(k) = \begin{bmatrix} u(k | k) \\ u(k+1 | k) \\ u(k+2 | k) \end{bmatrix} = \begin{bmatrix} u(k | k) \\ \frac{u(k | k)}{u(k+1 | k)} \\ u(k+2 | k) \end{bmatrix} \quad (2-15)$$

计算出M矩阵和C矩阵，之后构建二次规划模式（cost function）^[?]。

$$J = \sum_{k=0}^{N-1} x(k+i | k)^T Q x(k+i | k) + u(k+i | k)^T R u(k+i | k) + x(k+N | k)^T F x(k+N | k) \quad (2-16)$$

上式中Q、R、F表示对应输入输出的权重系数矩阵，Q矩阵影响x变量的相关性，R矩阵跟输入相关，F矩阵跟终值相关， $x(k+N | k)^T F x(k+N | k)$ 与系统对应的终值状态相关。

简化上面的代价函数（cost function）矩阵得到下面的公式：

$$J = \mathbf{x}(k)^T \mathbf{G} \mathbf{x}(k) + \mathbf{U}(k)^T \mathbf{H} \mathbf{U}(k) + 2\mathbf{x}(k)^T \mathbf{E} \mathbf{U}(k) \quad (2-17)$$

该矩阵只包含了输入项和已知的初始输入项 $x(k)$ 。

$$\begin{aligned} G &= M^T \bar{Q} M \\ E &= C^T \bar{Q} M \\ H &= C^T \bar{Q} C + \bar{R} \end{aligned} \quad (2-18)$$

\bar{Q} 和 \bar{R} 表示上面式子中的Q和R矩阵的增广形式，G、H、E矩阵跟上面的M和C矩

阵相关。

$$\bar{\mathbf{Q}} = \begin{bmatrix} \mathbf{Q} & \cdots \\ \vdots & \mathbf{Q} & \vdots \\ \cdots & \cdots & \mathbf{F} \end{bmatrix} \quad \bar{\mathbf{R}} = \begin{bmatrix} \mathbf{R} & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & R \end{bmatrix} \quad (2-19)$$

2.4.3 MAVLink协议

MAVLink (Micro Air Vehicle Link) 协议最早由苏黎世联邦理工学院计算机视觉与几何实验组的Lorenz Meier于2009年发布，并遵循LGPL开源协议。MAVLink协议是在串口通讯基础上的一种更高层的开源通讯协议，主要应用在微型飞行器(micro aerial vehicle)的通讯上。Mavlink是为小型飞行器和地面站(或者其他飞行器)通讯时常常用到的数据制定一种发送和接收的规则并加入了校验(checksum)功能^[2]。

MAVLink传输时的基本单位是消息帧，图2.8为消息帧的结构。

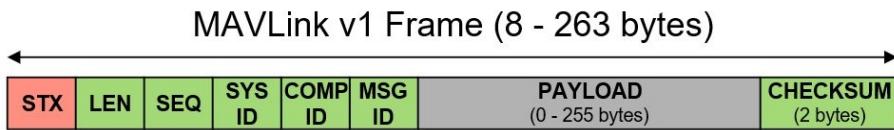


图 2.8 MAVLink Frame

如上图所示，每个消息帧都是上述的结构，除了灰色外，其他的格子都代表了一个字节的数据。

红色的是起始标志位(stx)，在v1.0版本中以“FE”作为起始标志。这个标志位在mavlink消息帧接收端进行消息解码时有用处。

第二个格子代表的是灰色部分(payload，称作有效载荷，要用的数据在有效载荷里面)的字节长度(len)，范围从0到255之间。在mavlink消息帧接收端可以用它和实际收到的有效载荷的长度比较，以验证有效载荷的长度是否正确。

第三个格子代表的是本次消息帧的序号(seq)，每次发完一个消息，这个字节的内容会加1，加到255后会从0重新开始。这个序号用于mavlink消息帧接收端计算消息丢失比例用的，相当于是信号强度。

第四个格子代表了发送本条消息帧的设备的系统编号(sys)，使用PIXHAWK刷PX4固件时默认的系统编号为1，用于mavlink消息帧接收端识别是哪个设备发来的消息。

第五个格子代表了发送本条消息帧的设备的**单元编号 (comp)**，使用PIXHAWK刷PX4固件时默认的单元编号为50，用于mavlink消息帧接收端识别是设备的哪个单元发来的消息（暂时没什么用）。

第六个格子代表了有效载荷中**消息包的编号 (msg)**，注意它和序号是不同的，这个字节很重要，mavlink消息帧接收端要根据这个编号来确定有效载荷里到底放了什么消息包并根据编号选择对应的方式来处理有效载荷里的信息包。

最后两个字节是**16位校验位**，ckb是高八位，cka是低八位。校验码由crc16算法得到，算法将整个消息（从起始位开始到有效载荷结束，还要额外加上个MAVLINK_CRC_节）进行crc16计算，得出一个16位的校验码。之前提到的每种有效载荷里信息包（由消息包编号来表明是哪种消息包）会对应一个MAVLINK_CRC_EXTRA，这个MAVLINK_CRC_EXTRA 是由生成mavlink代码的xml文件生成的，加入这个额外的东西是为了当飞行器和地面站使用不同版本的mavlink协议时，双方计算得到的校验码会不同，这样不同版本间的mavlink协议就不会在一起正常工作，避免了由于不同版本间通讯时带来的重大潜在问题。

MAVLink帧的介绍如图??所示。

Byte Index	Content	Value	Explanation
0	Packet start sign	v1.0: 0xFE (v0.9: 0x55)	Indicates the start of a new packet.
1	Payload length	0 - 255	Indicates length of the following payload.
2	Packet sequence	0 - 255	Each component counts up his send sequence. Allows to detect packet loss
3	System ID	1 - 255	ID of the SENDING system. Allows to differentiate different MAVs on the same network.
4	Component ID	0 - 255	ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
5	Message ID	0 - 255	ID of the message - the id defines what the payload "means" and how it should be correctly decoded.
6 to (n+6)	Data	(0 - 255) bytes	Data of the message, depends on the message id.
(n+7) to (n+8)	Checksum (low byte, high byte)	ITU X.25/SAE AS-4 hash, excluding packet start sign, so bytes 1..(n+6)	Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed from message fields. Protects the packet from decoding a different version of the same packet but with different variables).

图 2.9 MAVLink Frame介绍

在MAVLink消息帧中最重要的两个内容：一个是**msgid**，一个是**payload**，前

者是payload中内容的编号，后者则存放了消息。消息有许多种类型，在官网的网页中以蓝色的“#”加数字的方式来表示消息的编号如“#0”。

由于DJI MSDK协议本身不存在与MAVLink的兼容，因此我们针对于开源项目的MAVLink协议部分进行转换，根据实际项目控制的需要进行适配兼容MAVLink部分协议，并未完全兼容MAVLink支持的所有协议。

第三章 APP算法设计

3.1 算法概要

本项目使用大疆御2行业进阶版无人机作为主体，基于DJI MSDK进行二次开发，对DJI MSDK自有的协议转换为兼容QGC地面站的MAVLink协议，并通过无人机端侧带屏遥控器通过以太网数据链路远程组网传输信息至远端的QGC地面站软件，以及可见光和热成像视频流信息。无人机根据地面站给出的航点信息完成航点飞行，任务结束后，云台自动向下旋转90度，并利用OpenCV计算机视觉库完成自适应阈值+形态学处理+霍夫圆检测，并使用MPC控制算法完成自主视觉引导降落至特定降落目标点。

3.2 APP整体框架

本项目基于开源代码**rosettadrone**进行二次开发，在Android源代码的基础上单独添加了OpenCV检测框架以及MPC位置控制器设计部分^[?]。Rosetta Drone是DJI SDK的MAVLink包装器，允许用户使用MAVLink语言的地面控制站控制DJI无人机。从理论上讲，它应该与任何MAVLink GCS一起使用，但到目前为止，我们所有测试都是使用QGroundControl完成的。^[?]

APP集成的功能：

- 1.在QGC中报告遥测数据，如位置、姿态、相对高度、航向和电池剩余电量
- 2.从QGC返回启动的命令
- 3.在QGC中查看无人机视频源或将RTP转发到您选择的IP地址
- 4.创建和飞行航点任务
- 5.通过操纵杆和QGS飞行
- 6.从DroneKit中的 Python 起飞
- 7.使用Mavproxy同时连接QGC和DroneKit
- 8.使用Gstreamer/OpenCV/FFMPEG和DroneKit创建复杂的AI功能

飞行任务流程：

1. 将无人机配套的带屏遥控器打开，然后打开DJI无人机的电源。

2. 启动Rosetta Drone APP。如果应用程序成功与您的无人机通信，右上角的DJI灯将变为绿色。

3. 如果需要在外部设备上使用QGroundControl，单击齿轮图标以访问“设置”，选中“在外部设备上使用GCS”，然后指定IP地址。

4. 启动QGroundControl。应立即建立遥测连接，Rosetta Drone中的GCS指示灯将变为绿色。请注意，如果在与Rosetta无人机相同的设备上使用QGroundControl，则如果QGC在背景中，GCS灯可能不会变为绿色。

5. UDP视频回传步骤：

a. 点击QGC左上角的“Q”图标

b. 在“视频”下，将“视频源”更改为“UDP视频流”。

c. 将UDP端口更改为5600。

6. 为确保飞行安全，建议使用遥控器手动起飞。要从GCS解锁并起飞，需要单击“安全启用”按钮。它将变为绿色，并显示“准备飞行”。然后使用QGroundControl起飞或启动任务功能。

7. 飞行后，确保在接近无人机之前启用安全开关。

3.3 OpenCV视觉检测

使用DJI的codecManager函数获取H.264的RGBA原始视频，并将获取到的每一帧新建一个Mat类型的矩阵，用于后续处理。之后利用cvtColor函数将该帧图像的RGB色彩通道转化为灰度图像，之后进行闭运算并利用findContours函数寻找图像的轮廓，寻找到轮廓后，利用drawContours函数将轮廓绘制出来，最后将图像转化为Mat类型的矩阵，并将其存入一个vector类型的容器中，最后进行霍夫圆检测。

OpenCV实现的是一个比标准霍夫圆变换更为灵活的检测方法——霍夫梯度法，该方法运算量相对于标准霍夫圆变换大大减少。其检测原理是依据圆心一定是在圆上的每个点的模向量上，这些圆上点模向量的交点就是圆心，霍夫梯度法的第一步就是找到这些圆心，这样三维的累加平面就又转化为二维累加平面。第二步是根据所有候选中心的边缘非0像素对其的支持程度来确定半径^[?1]。

具体实现方法如下：

- 1.对于drawContours函数绘制的边缘图像，对其中的每一个非零点，考虑其局部梯度，即用Sobel函数计算x和y方向的Sobel 阶导数得到梯度。
- 2.利用得到的梯度，由斜率指定的直线上的每一个点都在累加器中被累加，这 的斜率是从一个指定的最小值到指定的最大值的距离。
- 3.同时，标记边缘图像中每一个非0像素的位置。
- 4.然后从二维累加器中这些点中选择候选的中心，这些中心都大于给定阈值并且大于其所有近邻。这些候选的中心按照累加值降序排列，以便于最支持像素的中心首先出现。
- 5.接下来对每一个中心，考虑所有的非0像素，并把这些像素按照其与中心的距离排序。从到最大半径的最小距离算起，选择非0像素最支持的 一条半径^[?]。
- 6.如果 一个中心收到边缘图像 0像素最充分的支持，并且到前期被选择的中心有足够的距离，那么它就会被保留下。

3.4 卡尔曼滤波与优化

OpenCV中有两个版本的卡尔曼滤波方法：KalmanFilter(C++)和CvKalman，这里我们使用前者方案。由上一步逐帧检测得到的中心点作为待估计的状态点传入卡尔曼滤波器中，并将最优估计后的结果传出作为MPC控制的输入。

卡尔曼滤波器具体实现方法如下：

- 1.初始化相关参数：根据需要，定义转移矩阵A与测量矩阵H，过程噪声Q与测量噪声R，最小均方误差P，系统初始状态x(0)与初始测量值z(0)
- 2.预测模型：利用KF.predict()函数，返回的是下一时刻的状态值KF.statePost(k+1)
- 3.更新上一步的测量值measurement
- 4.更新卡尔曼增益量KF.correct(measurement)，最终的结果应该是更新后的状态估计statePost

以上流程用图??表示。

3.5 MPC位置控制

1.MPC控制器设计

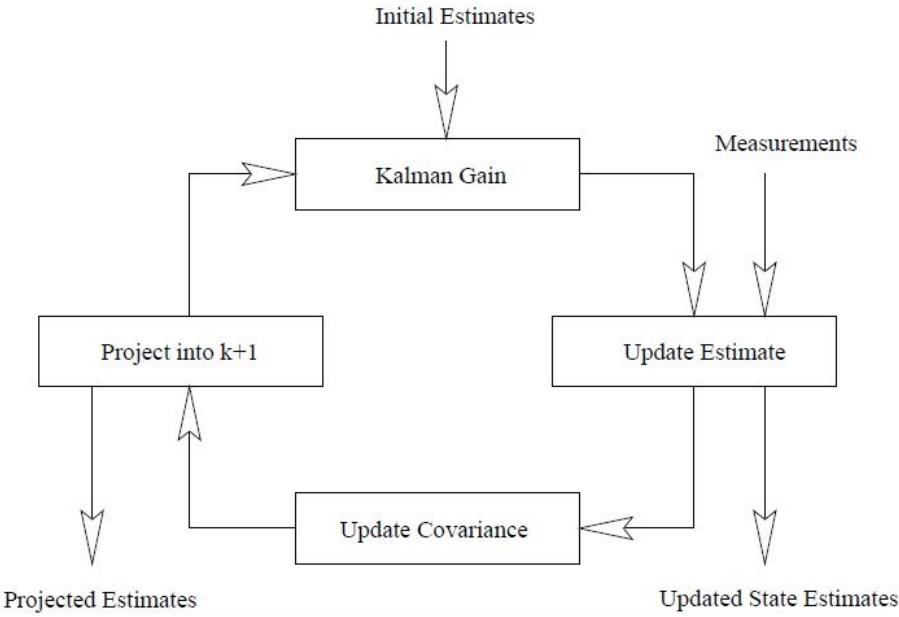


图 3.1 Kalman 滤波器工作流程

我们提出的四轴飞行器模型预测器位置控制如下图??方框中

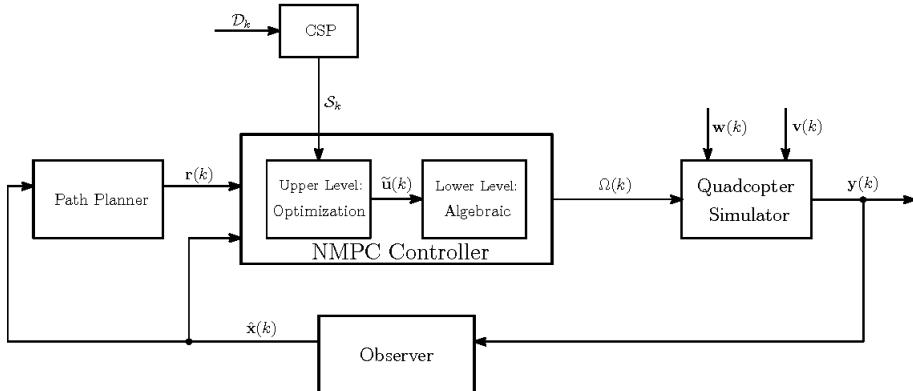


图 3.2 MPC 控制过程图

在MPC控制中，每个优化回路开始前，使用四轴飞行器状态空间模型来增强当前的角速率（这里由于MSDK API设定，认为角速率近似等于对应的虚拟摇杆控制的速率，我们使用虚拟摇杆进行控制）。这样做是为了获得更精确的四轴飞行器状态预测并用于优化器。使用这种方法可以确保在建模过程中做出的模型的不确定性和假设得到补偿^[?]。

速率设定点从姿态控制器发送到优化器块。优化器还接收成本函数和系统约束。希尔德雷斯的二次规划程序被用于优化器块，以最小化受预定义的四轴飞行器约束的成本函数。MPC控制器的输出是控制或输入向量u。在将这个向量发布到DJI虚拟摇杆控制之前，在-1和1之间进行缩放；这种形式的缩放是针对于DJI虚

拟摇杆控制的要求。最终将缩放后的向量发送到混控器中用于最终的四旋翼位置控制^[?]。

2. 飞行器运动学模型

我们区分两个主框架——惯性框架（**ENU Frame**）和身体框架（**Body Frame**），如图??所示。惯性坐标系在这里给出了东-北-上（ENU）坐标，并且身体框架以向前-左-向上（FLU）坐标给出^[?]。选择此约定是因为它是SDK中提供的约定。因此，数据是已经在此参考框架中交付。有关四轴飞行器SDK的更多详细信息，请参阅DJI MSDK相关文档说明。偏航角 ψ 定义为无人机的前向一侧和东面。

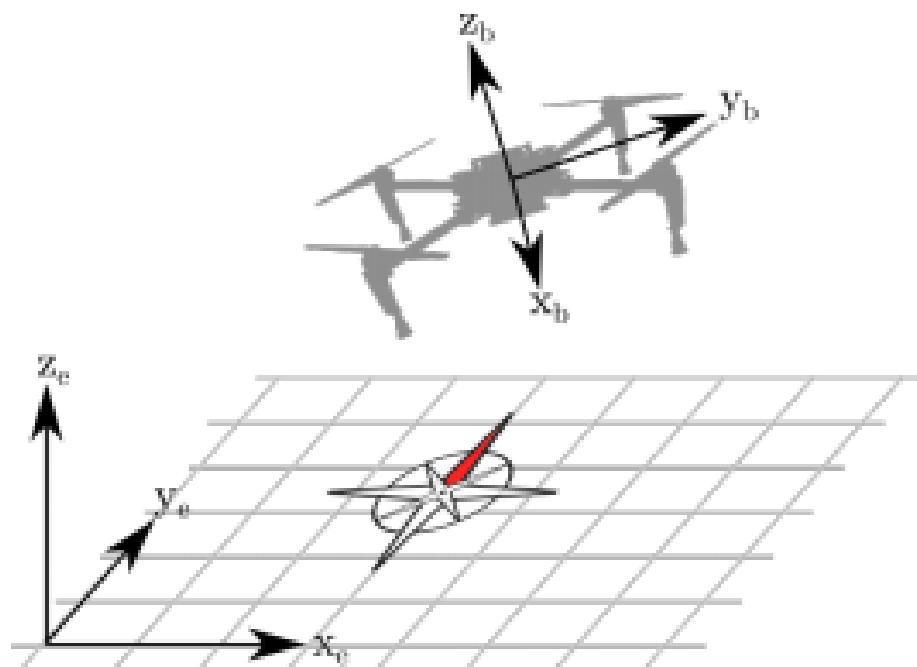


图 3.3 飞行器坐标系

刚体坐标系与惯性坐标系之间的关系可以用旋转矩阵（**Rotation Matrix**）表示

$$\mathbf{R}_{ib} = \begin{bmatrix} \cos \psi \cos \theta & \cos \psi \sin \varphi \sin \theta - \sin \psi \cos \varphi & \cos \psi \cos \varphi \sin \theta + \sin \psi \sin \varphi \\ \sin \psi \cos \theta & \sin \psi \sin \varphi \sin \theta + \cos \psi \cos \varphi & \sin \psi \cos \varphi \sin \theta - \cos \psi \sin \varphi \\ -\sin \theta & \sin \varphi \cos \theta & \cos \varphi \cos \theta \end{bmatrix} \quad (3-1)$$

其中 φ, θ 和 ψ 分别代表滚转角，俯仰角和偏航角。

3. 飞行器动力学模型

作用在四轴飞行器上的力和力矩是来自四个转子叶片的重力、阻力和推力，以及来自风等的外部干扰的结果。总推力是在飞行器框架中的负z方向上定向的，并且是来自四个单独电机的推力的总和^[?]。在惯性系中，总力可以表示为

$$\begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} = - \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} + \mathbf{R}_{ib} \begin{pmatrix} 0 \\ 0 \\ \sum_{k=1}^4 T_k \end{pmatrix} + \mathbf{F}_{drag} + \mathbf{F}_{ext}, \quad (3-2)$$

其中g是重力加速度， \mathbf{F}_{drag} 是来自阻力的力，而 \mathbf{F}_{ext} 包含了所有的外力，例如由于风。假设阻力与各自方向上的速度成正比。最低水平的控制输入是提供给四个电机的单独电源。为了简单和安全，有一个内置的姿态控制系统，用于计算单个电机的推力指令^[?]。现在，控制输入是命令滚动角 φ_{cmd} 、俯仰角 θ_{cmd} 、偏航率 $\dot{\varphi}_{cmd}$ 和垂直速度 ω_{cmd} 。假设垂直速度 ω 对垂直速度命令的名义响应为一阶系统的形式，表示为拉普拉斯传递函数

$$W(s) = \frac{k_w}{\tau_w s + 1} W_{cmd}(s) \quad (3-3)$$

其中， $\omega(t)$ 为惯性系中的垂直速度。这意味着t时刻的总推力可以表示为

$$T(t) = \frac{g + \dot{w}(t)}{\cos \theta(t) \cos \varphi(t)} = \frac{g + 1/\tau_w (k_w w_{cmd}(t) - w(t))}{\cos \theta(t) \cos \varphi(t)} \quad (3-4)$$

姿态动力学近似为二阶系统。对于滚动和俯仰，从命令到姿态的传递函数以拉普拉斯传递函数的形式表示为

$$\Phi(s) = \frac{k_\phi \omega_\phi^2}{s^2 + 2\xi_\phi \omega_\phi s + \omega_\phi^2} \Phi_{cmd}(s) \quad (3-5)$$

$$\Theta(s) = \frac{k_\theta \omega_\theta^2}{s^2 + 2\xi_\theta \omega_\theta s + \omega_\theta^2} \Theta_{cmd}(s) \quad (3-6)$$

各自地对于偏航运动，速率控制被建模为

$$\Psi(s) = \frac{1}{s} \frac{k_\psi}{\tau_\psi s + 1} \dot{\Psi}_{cmd}(s) \quad (3-7)$$

现在，这些动态模型可以总结为

$$\begin{aligned}
\dot{\mathbf{p}}(t) &= \mathbf{v}(t) \\
\dot{\mathbf{v}}(t) &= -\mathbf{g} + \mathbf{R}_{ib}(t)\mathbf{T}(t) - \mathbf{D}\mathbf{v}(t) + \frac{1}{m}\mathbf{F}_{ext}(t) \\
\dot{\boldsymbol{\theta}}(t) &= \boldsymbol{\omega}(t) \\
\dot{\boldsymbol{\omega}}(t) &= f_{\boldsymbol{\omega}}(\boldsymbol{\omega}(t), \boldsymbol{\theta}(t), \boldsymbol{\theta}_{cmd}(t))
\end{aligned} \tag{3-8}$$

其中 p 表示位置, v 表示速度, a 表示加速度, θ 表示姿态, ω 表示角加速度。函数 $f_{\boldsymbol{\omega}}$ 由上述方程给出。可以看出, 该模型仅在速度导数上是非线性的, 其中输入项 $\mathbf{R}_{ib}\mathbf{T}$ 由以下向量给出

$$\mathbf{R}_{ib}\mathbf{T} = (g + \dot{w}) \begin{bmatrix} \frac{\cos \psi \cos \varphi \sin \theta + \sin \psi \sin \varphi}{\cos \theta \cos \varphi} \\ \frac{\sin \psi \cos \varphi \sin \theta - \cos \psi \sin \varphi}{\cos \theta \cos \varphi} \\ 1 \end{bmatrix} = (g + \dot{w}) \begin{bmatrix} \cos \psi \tan \theta + \frac{\sin \psi \tan \varphi}{\cos \theta} \\ \sin \psi \tan \theta - \frac{\cos \psi \tan \varphi}{\cos \theta} \\ 1 \end{bmatrix} \tag{3-9}$$

对于飞行器的每一步控制时间段内, 假设角度 ψ 和 θ 很小, 这可以近似为

$$\mathbf{R}_{ib}\mathbf{T} \approx (g + \dot{w}) \begin{bmatrix} \cos \psi \cdot \theta + \sin \psi \cdot \varphi \\ \sin \psi \cdot \theta - \cos \psi \cdot \varphi \\ 1 \end{bmatrix} \tag{3-10}$$

系统方程在状态空间形式上表示如下

$$\frac{d}{dt} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{3-11}$$

$$\begin{aligned}
\frac{d}{dt} \begin{bmatrix} u \\ v \\ w \end{bmatrix} &= - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} - \begin{bmatrix} k_{dx} & 0 & 0 \\ 0 & k_{dy} & 0 \\ 0 & 0 & k_{dz} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \frac{1}{m} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} \\
&\quad + \begin{bmatrix} \cos \psi \tan \theta + \frac{\sin \psi \tan \varphi}{\cos \theta} \\ \sin \psi \tan \theta - \frac{\cos \psi \tan \varphi}{\cos \theta} \\ 1 \end{bmatrix} (g + \dot{w})
\end{aligned} \tag{3-12}$$

$$\frac{d}{dt} \begin{bmatrix} \varphi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \quad (3-13)$$

$$\frac{d}{dt} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = - \begin{bmatrix} \omega_\phi^2 & 0 & 0 \\ 0 & \omega_\theta^2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} - \begin{bmatrix} 2\omega_\phi\xi_\phi & 0 & 0 \\ 0 & 2\omega_\theta\xi_\theta & 0 \\ 0 & 0 & 1/\tau_\psi \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} + \begin{bmatrix} k_\phi\omega_\phi^2 & 0 & 0 \\ 0 & k_\theta\omega_\theta^2 & 0 \\ 0 & 0 & k_\psi/\tau_\psi \end{bmatrix} \begin{bmatrix} \varphi_{cmd} \\ \theta_{cmd} \\ \psi_{cmd} \end{bmatrix}. \quad (3-14)$$

4. 降落点约束

在着陆过程中需要满足的相对位置有一些限制。第一个约束与接近的方向有关。因为我们不希望飞行器离降落板太远，而且由于降落板附近的地形存在一定的不确定性，我们制定了一个空间约束，迫使飞行器从大约上方接近降落板，这种非凸约束可以表示为：

$$\begin{aligned} \mathbf{A} \begin{bmatrix} \Delta x(t) \\ \Delta y(t) \\ h(t) \end{bmatrix} &\geq \mathbf{k} - M \mathbf{f}_b(b(t)) \\ \mathbf{B} b(t) &\leq \mathbf{1} \end{aligned} \quad (3-15)$$

矩阵 $\mathbf{A} \in \mathbb{R}^{k \times 3}$ 和向量 $\mathbf{k} \in \mathbb{R}^k$ 一起描述区域的形状的平面不等式的形式，和 $\mathbf{f}_b(b) \in \{0, 1\}$ 是一个二进制向量值函数表明约束是活跃在一个给定的时间。此外， M 被选择为一个足够大的常数。

根据实际降落地形需要，这里的降落控制区域被选择为一个有六个边的多面体。对于边坡度的任意定义表示为

$$\begin{bmatrix} -h_s & 0 & (d_{x_1} - d_l) \\ h_s & 0 & (-d_{x_2} - d_l) \\ 0 & -h_s & (d_{y_1} - d_l) \\ 0 & h_s & (-d_{y_2} - d_l) \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \Delta x(t) \\ \Delta y(t) \\ h(t) \end{bmatrix} \geq \begin{bmatrix} d_l h_s \\ -d_l h_s \\ d_l h_s \\ -d_l h_s \\ h_s \\ 0 \end{bmatrix} \quad (3-16)$$

除了这个空间约束外，我们还对着陆时的垂直速度进行了约束，表示为

$$\begin{aligned} w(t) &\geq w_{\min} \\ w(t) &\geq k_1 h(t) + w_{\min, \text{land}} \end{aligned} \quad (3-17)$$

其中 $w_{\min, \text{land}} < 0$ 为降落的极限速度， $k_1 < 0$ 为一个常量。

5. 飞行器相关测试参数

我们采用的DJI御2行业进阶版飞行器相关参数如表??所示

Quadcopter Parameters	
Mass, m	0.909 kg
Moment arm, d	0.203 m
Thrust coefficient, b	3.0587×10^{-7} N/rpm ²
Drag coefficient, k	5.4270×10^{-9} Nm/rpm ²
Moment of inertia about x-axis, I_{xx}	0.0208kgm ²
Moment of inertia about y-axis, I_{yy}	0.0194kgm ²
Moment of inertia about z-axis, I_{zz}	0.0252kgm ²

表 3.1 DJI御2行业进阶版飞行器相关参数

3.6 MAVLink协议QGC地面站通信过程

APP在连接上地面站后会主动向地面站发送心跳包（HEARTBEATS），飞行器姿态，系统状态，遥控器信号等组成的数据流。各个数据都会以一定的频率发送，心跳包发送频率为1Hz，姿态信息频率为7-8Hz，采用UDP VPN组网连接QGC时的姿态数据发送频率在7-8Hz左右。QGC地面站会在刚连接上遥控器时发送命令，请求飞控传回所有参数，飞控根据自己的情况判断是否接受地面站的

请求，并根据不同的命令执行相应的操作，针对于一些特殊命令需要飞控回复地面站确认信号。之后地面站根据用户的操作会发送相应的maylink消息给飞控，比如设置航点，解锁起飞等。整个通信过程采用半双工方式（在同一时刻只能选择发送或者选择接受数据，不能同时收发数据）^[?]。

以上UDP通信的内容可以通过Wireshark软件进行抓包分析，由于数据包为原始信息，难以直接辨认其数据帧的内容，因此我们使用MAVLink Lua脚本（mavgen）对数据帧的内容进行解析与类别对应。解析后的数据帧内容如图??所示

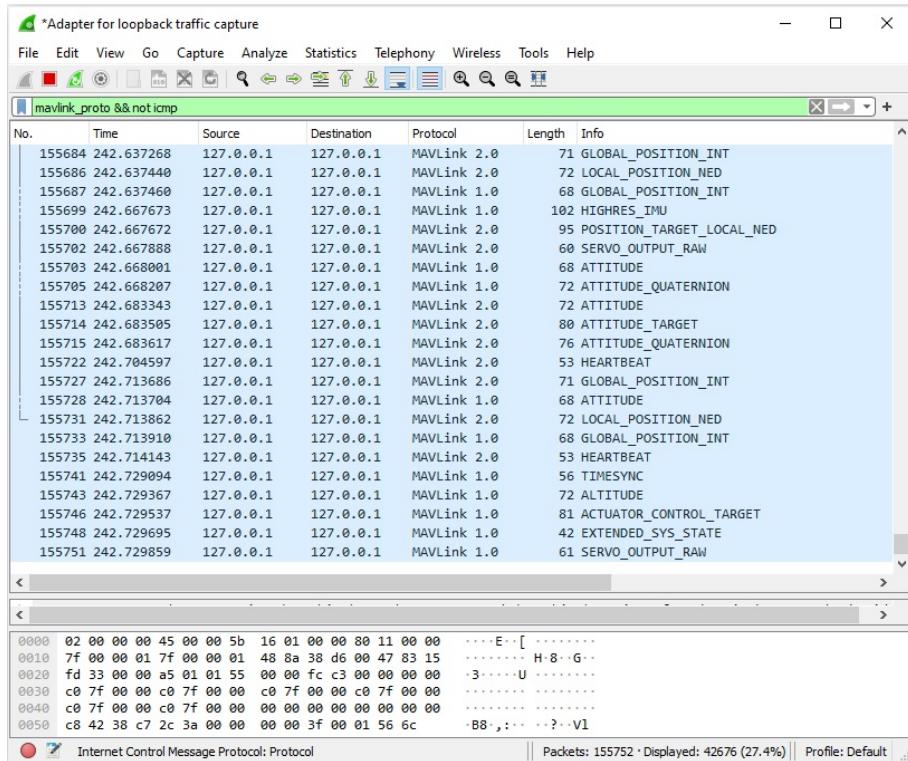


图 3.4 Wireshark MAVLink数据抓包解析

MAVLink支持多种飞行模式，比如manual（手动）模式、auto（自动）模式、althold（定高）模式、posctl（位置）模式、guided（外部引导控制）等。这里我们仅使用guided模式作为地面站控制模式，manual模式作为摇杆控制的手动飞行模式，如图??所示

这里由于我们使用QGC地面站外部控制航点飞行，因此我们选用**Guided**模式作为地面站控制模式。

All flight modes are explained in detail below.

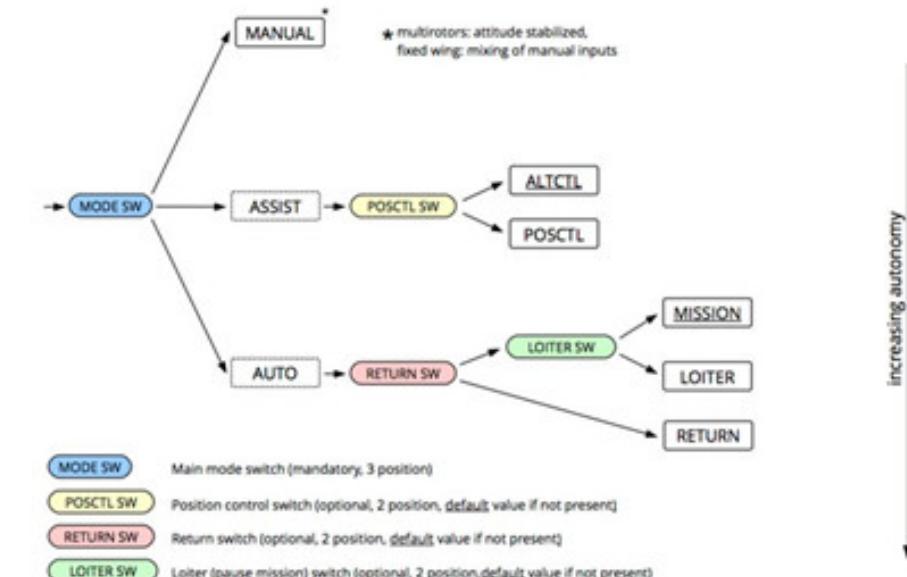


图 3.5 MAVLink 飞行模式图

第四章 APP算法实现与测试

我们使用蒲公英4G VPN室外工业路由器广播5.8G WiFi信号，并WiFi连接带屏遥控器端，并在地面站端侧安装蒲公英客户端，并设置路由器端与地面站软件端连接至同一虚拟网络中，使得地面站端可以与遥控器端侧通过虚拟局域网通信。在此基础上，我们使用QGroundControl开源地面站软件，并在APP中配置MAVLink UDP目的地址IP，端口14550，地面站QGC端配置MAVLink UDP端口同样为14550，如图??所示。

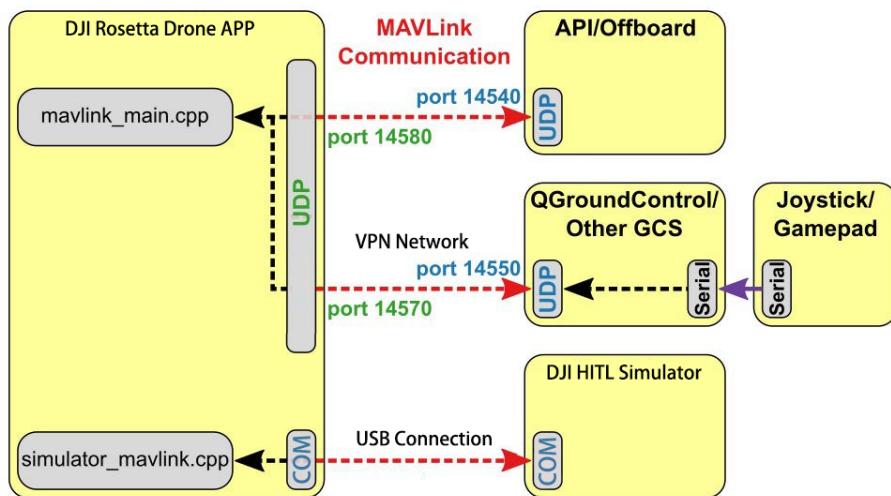


图 4.1 QGC地面站网络连接至APP

连接配置成功后可以在地面端看到对应的电量显示以及信号强度显示。

QGC地面站MAVLink回传显示??

由于无人机室外飞行存在较大的不确定性，为避免炸机造成损失，我们使用了DJI提供的DJI Assistant 2 for Mavic软件实现了硬件在环仿真（**HITAL Simulation**）。与此同时，我们在带屏遥控器的APP中集成了基于虚拟摇杆输入控制的DJI Simulator，并实现了APP内的软件在环仿真（**SITL Simulation**）。

根据实际的多次项目试飞测试，我们进行了以下的优化处理：

使用DJI飞行器传回的原始H.264视频数据进行检测处理，每帧图像耗时100ms 150ms，实时性较低，因此我们一定程度上降低了原始的H.264视频图像分辨率。

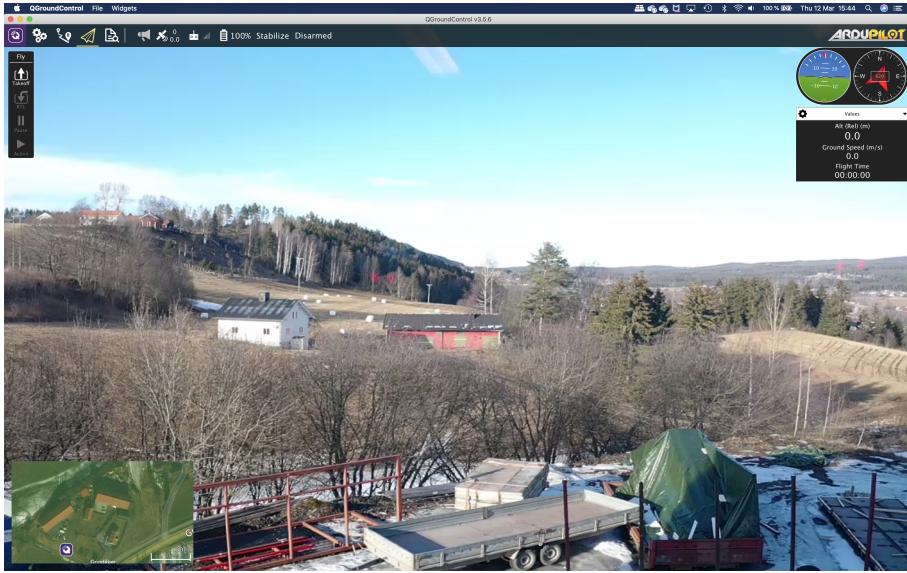


图 4.2 QGC主界面

优化了无人机的控制轨迹模型，定义了更为合适的代价函数(cost function)用于将预测的输出驱动到所期望的参考值。该方法可以在样本时间k时，根据预测范围的大小生成输出预测。

4.1 OpenCV视觉检测效果

原始1080P H.264编码格式的视频逐帧处理所需时间在100ms 150ms之间，帧率约为6.7fps 10fps，难以满足实时检测的要求，而且由于该帧率下对应的控制频率在5Hz 10Hz之间，相较于期望控制频率20Hz 30Hz较低，使用MPC算法进行xy位置控制存在较大的肉眼可见的控制滞后现象，因此我们将分辨率由1920x1080等比缩小并降低至568x320，此时的检测帧率可提升至25fps 35fps，对应的控制频率可提升至20Hz 30Hz，该控制频率下无人机的xy位置控制振荡现象明显减少，而且控制过程更为柔和。

根据二值化的阈值参数不同，以及调整霍夫圆检测中的圆心与圆心间的距离，图像中的圆的大小设置，最终将检测效果进行了一些相关优化，最终的降落板识别效果如图??所示。

4.2 MPC模型预测控制效果

我们使用MATLAB对模型进行理想化仿真，并在APP上集成了MPC模型控制



图 4.3 霍夫圆检测效果

n_u	n_y	\mathbf{p}, \mathbf{q}	\mathbf{r}	disturbance
3	6	$\sin t + \frac{\cos(3t)}{2}$	$\sin t + \frac{\cos(2t)}{2} + \frac{\sin(3t)}{3}$	$(\text{rand}(3, 101) * 2 - 1) * 0.5$

表 4.1 MATLAB 模拟需要初始化的参数

器。正弦函数为用作控制器要跟踪的引用。它的性能是通过将模型输出（或响应）与参考一起绘制，通过改变控制和预测范围来观察。以上的权重对控制输入进行了调整，以适应这些变化。此外，适用的矩阵和向量取决于控制和预测的大小范围，被修改为兼容的矩阵和矢量计算。

MPC角速率控制器跟踪的正弦参考是下面的傅里叶级数

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \sin t + \frac{\cos(3t)}{2} \\ \sin t + \frac{\cos(3t)}{2} \\ \sin t + \frac{\cos(2t)}{2} + \frac{\sin(3t)}{3} \end{bmatrix} \quad (4-1)$$

变量 $\dot{\phi}$ 、 $\dot{\theta}$ 和 $\dot{\psi}$ 是滚动、俯仰和偏航的角速率轴分别以弧度/秒为单位测量， t 是模拟时间步长，其数值边界在0和20之间；增量为0.2。

表??显示了MATLAB模拟需要初始化的参数，其中 $\begin{bmatrix} p & q & r \end{bmatrix}^T = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$ ，
 $n_u = 3, n_y = 6$

为了验证MPC模型预测控制的效果，我们通过最小化过程通过封装**Hildreth**的二次规划函数和状态空间模型参数来减缓，在向混控器模块输出输入（或扭矩）命令之前至少运行两次。运行三次循环导致四轴飞行器在起飞后偏离航线，随后出现控制不稳定。然而，一个带有两次迭代的循环导致了通过每个任务路径点的

相对平稳的飞行，并且任务以一个安全着陆结束。

上一步使用MATLAB完成了控制器设计与仿真测试，在此基础上使用DJI Simulation完成软件在环仿真与硬件在环仿真测试，并实现实际飞行测试，步骤如图??所示。

实际测试中，我们不同类型的扰动进行了扰动补偿测试，主要表现为作用于四轴飞行器上的阵风以及由于降落点检测误差等导致的极少数错误的降落点坐标。^[2] DJI Simulator导出的结果显示，在大约距离地面10m时，飞行器会暂时减缓下降速度，以使其更好地对齐降落板，以便垂直速度更好地匹配。然后，当无人机继续下降时，它以一个安全的相对垂直速度着陆，使用估计的上升运动作为飞行高度的参考。该仿真结果说明了该控制器在静态干扰下的工作原理。

图??和??为MPC控制的滚转速率(roll rate)和俯仰速率(pitch rate)曲线，绘制的飞行数据中使用的控制视野和预测视野分别为2和4。

4.3 降落点精准度评价

未使用卡尔曼滤波器的降落点检测，无人机降落落点精度为：-15cm~15cm。

通过算法的优化与改进，以及添加了卡尔曼滤波器的最优估计，可以使无人机最终降落精度达到-5cm~5cm。

图??展示了使用卡尔曼滤波器进行降落点最优估计的检测效果。(绿色的圆点表示预测的位置，其中该圆点的圆心表示预测的中心坐标)

将落点坐标以xy坐标系形式绘制如图??，其中红色的点表示检测到的数据，预测的点是将检测到的数据进行卡尔曼滤波最优估计后的数据：

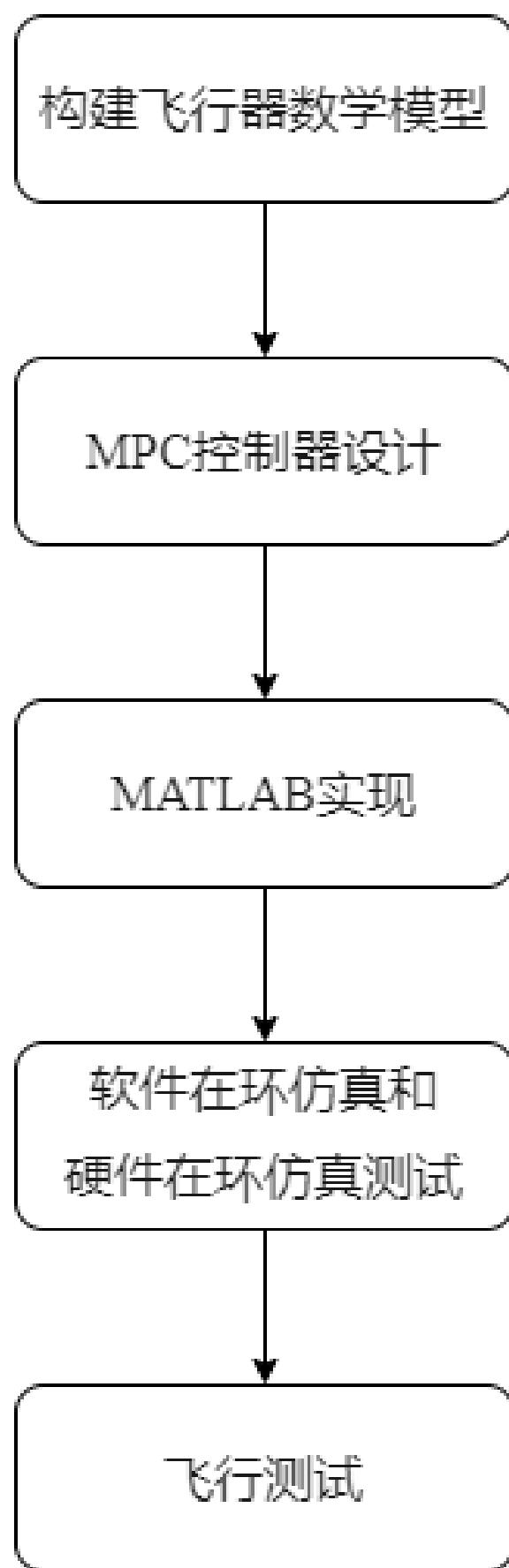


图 4.4 MPC 控制步骤

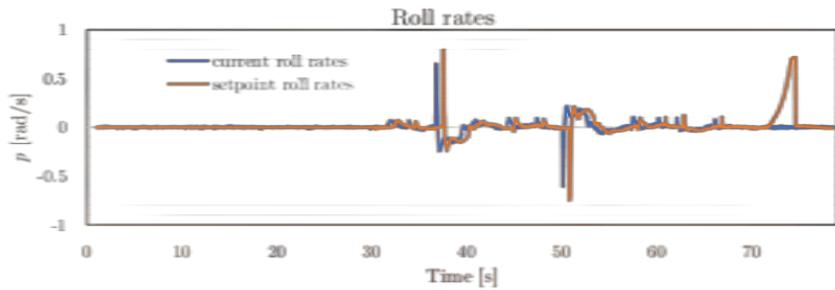


图 4.5 MPC 滚转速率曲线

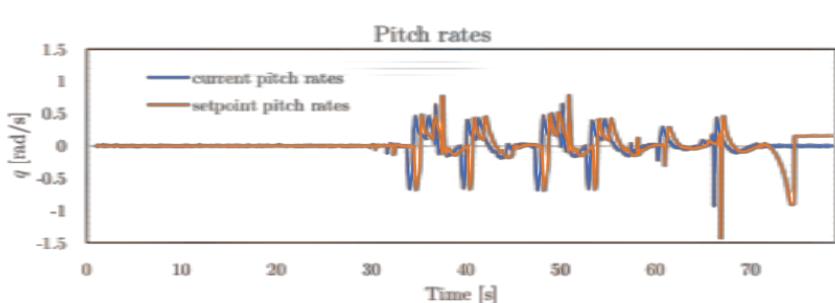


图 4.6 MPC 俯仰速率曲线

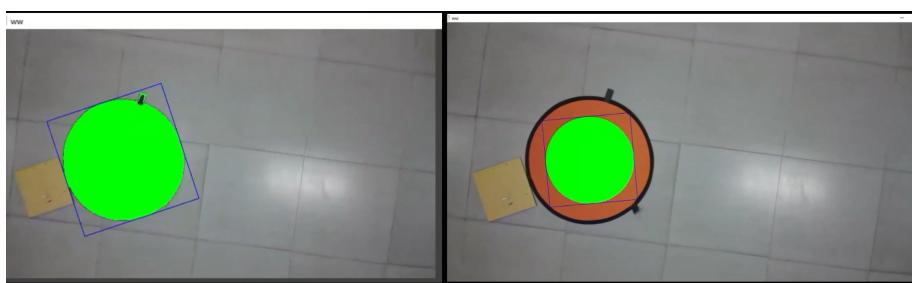


图 4.7 卡尔曼滤波器最优估计

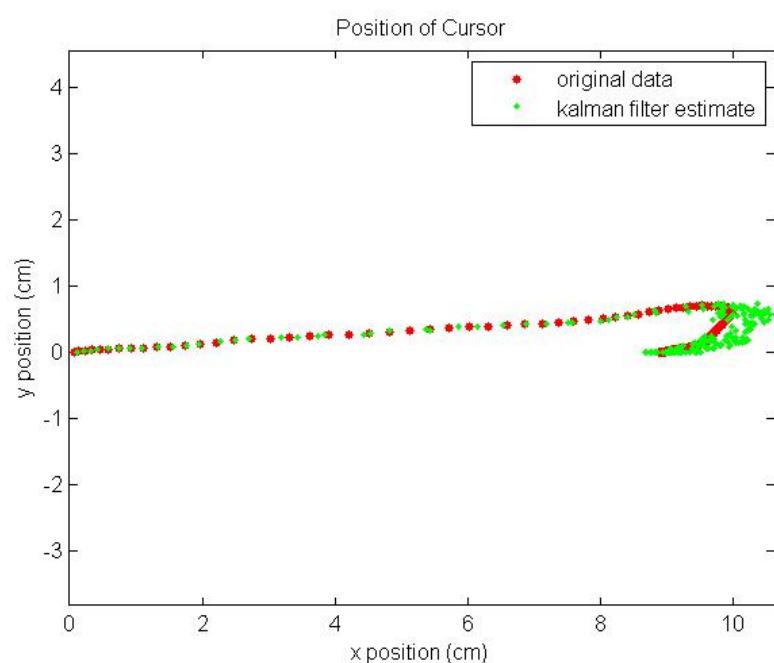


图 4.8 落点坐标对比

第五章 总结与展望

5.1 结论

DJI MSDK为开发者提供了便携的API接口，同时基于DJI行业级别的飞行器，提供了更为稳定的飞行平台。相对于开源的项目，基于DJI行业化飞行平台可以更便捷地进行项目的开展与相关实验。QGC地面站作为支持MAVLink的开源地面站，基于QT打造，支持二次开发与用户自定义接口，同时由于其强大的Dronecode社区的支持，相关资料和API开发手册也较为完整。

由于自主航点飞行的部分基于DJI MSDK API实现，但是DJI系列的飞行器本身不自带降落点视觉检测与自主降落控制，因此我们把项目的重点放在对于降落板的视觉识别与最优估计以及基于MPC模型预测控制的降落控制上。

由于我们的降落板特征较为明显，且受到带屏遥控器本身的硬件算力限制，相对于深度学习目标检测，传统的OpenCV相关检测算法显著地提升了检测效率，在合适的参数范围下具有较好的效果。卡尔曼滤波器最优化估计作为状态空间模型中无法直接获取的数据，基于自身观测和预测的数据进行最优估计，对于含有高斯噪声的线性系统效果尤为优秀。

测试中发现，MPC控制对于飞行器的角速率变化较为敏感，针对于飞行器的不同负载状态，角度控制和角速度控制存在较大的差异。所得结果表明，MPC角速率控制器能够获得对设定点角速率的密切参考跟踪，这些结果证明了在DJI飞行器上实现MPC模型预测位置控制的可能性^[?]。

MAVLink作为兼容开源界多种飞行器构型的轻小型化飞行器通讯协议，具有较多的功能和帧格式，但是由于DJI飞行器接口数据类型的限制等，无法将大部分MAVLink多旋翼的相关参数传回至QGC地面站。因此，我们根据实际需要有选择性地传输相关飞行参数，并针对于DJI MSDK接口数据进行了相关类型和协议的转换。

5.2 未来工作

在上一章中提出了基于DJI飞行器与QGC地面站的通信可行性，以及基于OpenCV传统算法+卡尔曼滤波器对于降落点最优估计，MPC模型预测位置控制的应用。考虑到未来的进一步的行业化应用，我们将会使用DJI经纬M300/M30T等可拓展的行业化飞行器进行进一步的开发与适配。同时，针对目前的MAVLink的协议内容和数据传输稳定性等问题，我们也会进一步开发与不断完善。

目前我们的项目重点在APP的开发上面，未来我们将基于现有的成果进行相关拓展，在**DJI Onboard SDK**板载端基于ROS开发相关的协议转换以及飞行控制、视觉识别等内容。另外，针对目前DJI带屏遥控器的性能相关问题，未来我们也将通过集成高性能CPU/GPU/SoC等嵌入式开发板至飞行器平台上，提高机载端运算效率。

致 谢

大学本科的四年学习即将结束，感谢导师对于项目的指导，感谢实验室的成员的共同参与加快项目的发展。

从大一到大四，从简易的电动直升机到匿名/无名STM32电赛简易四旋翼飞行器再到基于pixhawk的多旋翼飞行器，从大疆NAZA飞控到A3 Pro商业级飞控，从自研二轴倾转飞行器到V22鱼鹰型无人机，从开源VTOL 4+1控制到VTOL倾转矢量控制无人机。我的每一步发展都是基于之前的相关经验和教训，一步步地将飞控算法和导航算法等做到比上一次更好。与此同时，我也在尝试将传统的PID算法进行不断地改良，优化控制模型，采用ADRC自抗扰控制等提高飞行器的控制鲁棒性。

感谢开源界两大教科书式项目ArduPilot和PX4，感谢世界上对于开源飞控维护的社区与团队，使得我们能从中学到很多相关的知识，了解飞控参数调试和算法优化的一些经验。同时，作为ArduPilot官方论坛的成员，我也积极参与论坛内的相关技术讨论，进一步了解当前开源飞控系统的发展趋势、硬件性能、以及相关新功能适配等。

飞控与导航和视觉识别领域作为无人机的小脑、大脑和眼睛，是无人机自主飞行不可缺少的部分，可能因为一个参数错误就会导致飞控系统出现故障，进而导致无人机的姿态不可控甚至炸机。因此，对于无人机领域的开发者来说，更要求的是一种严谨的态度，需要严密的数学计算与仿真以及上机测试从而确保系统不会出故障。

因此，我们在试飞无人机与测试代码之前，也充分考虑到了可能出现的情况，并加以防范，先是飞控软件硬件在环仿真，然后是室内护网飞行，最后是室外实际测试。

附录 A 照片展示

A.1 实物展示

以下是我们作品实物展示照片：



图 A.1 作品实物展示照片

A.2 效果测试

效果测试照片（右下角小窗口内的是识别的降落板）：



图 A.2 作品实物展示照片

A.3 APP软件截图

APP启动、带屏遥控器设备识别界面：

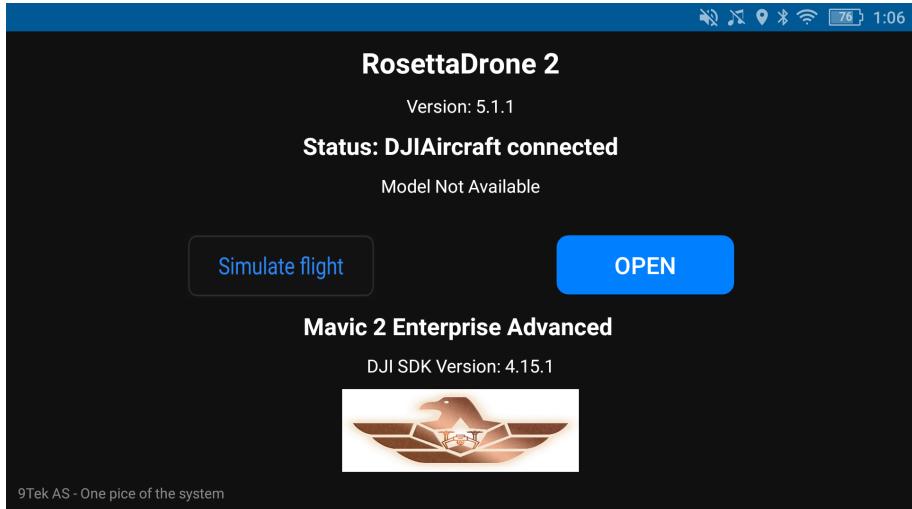


图 A.3 APP启动界面

带屏遥控器APP主界面：



图 A.4 APP主界面

带屏遥控器端QGC本机回环测试：

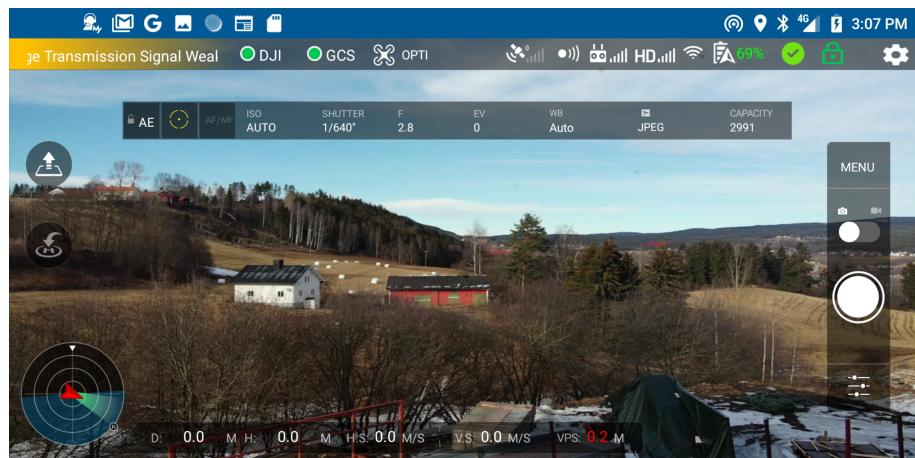


图 A.5 带屏遥控器端QGC

A.4 效果测试

第一次测试航点飞行路径：



图 A.6 第一次测试航点

第二次测试航点飞行路径：



图 A.7 第二次测试航点

A.5 控制效果

视觉引导降落过程中，使用MPC控制算法，Pitch和Roll的角度和角速度曲线：



图 A.8 Roll 角度曲线

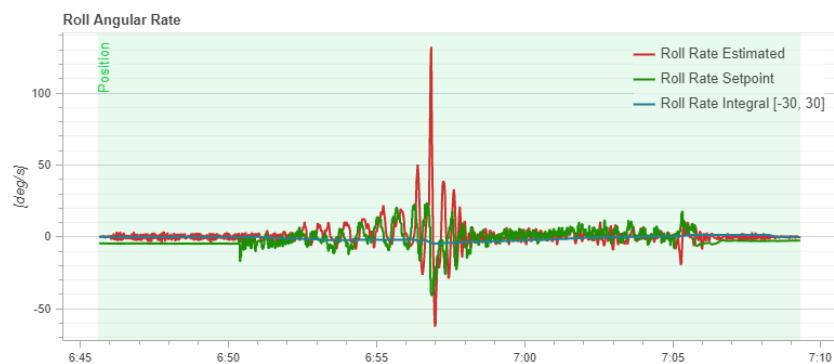


图 A.9 Roll 角速度曲线

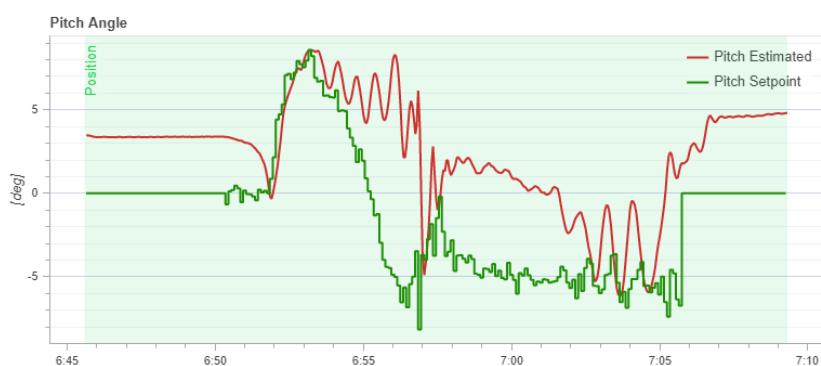


图 A.10 Pitch 角度曲线



图 A.11 Pitch 角速度曲线

附录 B 数据

B.1 代码

代码 B.1 OpenCV目标检测核心代码

```
1  /**
2   * Program: Hello world;<br>
3   * Time: 2015/12/04.
4   * @author Stick Cui
5   */
6  public class JavaTest{
7      /**
8       * The main methord.
9       * @param args The parameter when the methord is called .
10      */
11     public static void main(String[] args) {
12         System.out.println("Hello world!");
13     }
14 }
```

OpenCV Kalman滤波器实现

```
1 private void kalmanInit() {
2
3     kf = new KalmanFilter(2, 2, 0, CV_32F);
4
5     Log.d(TAG, "CHECKING KalmanFilter Matrix: " +
6           "\n transitionMatrix: " + kf.get_transitionMatrix().size() +
```

```
7      "\n measurementMatrix: " + kf.get_measurementMatrix().size() +
8      "\n statePre: " + kf.get_statePre().size() +
9      "\n processionNoiseCov: " + kf.get_processNoiseCov().size() +
10     "\n measurementNoiseCov: " + kf.get_measurementNoiseCov().size() +
11     "\n errorCovPost: " + kf.get_errorCovPost().size());
12
13     kalmanMatrixInit2();
14 }
15
16 private void kalmanMatrixInit4() {
17 //     CHECKING KalmanFilter Matrix:
18 //     transitionMatrix : 4x4
19 //     measurementMatrix: 4x2
20 //     statePre : 1x4
21 //     processionNoiseCov: 4x4
22 //     measurementNoiseCov: 2x2
23 //     errorCovPost: 4x4
24     transitionMatrix = Mat.eye(4, 4, CV_32F);
25     transitionMatrix.put(4, 4, new float[]{1, 0, 1, 0,
26                           0, 1, 0, 1,
27                           0, 0, 1, 0,
28                           0, 0, 0, 1});
29     kf.set_transitionMatrix(transitionMatrix);
30
31     measurementMatrix = Mat.eye(2, 4, CV_32F);
32     kf.set_measurementMatrix(measurementMatrix);
33
34     statePre = new Mat(4,1, CV_32F);
35     statePre.put(0, 0, 0.5f);
36     statePre.put( 1,0, 0.5f);
```

```
37     statePre.put( 2, 0,0);
38     statePre.put( 3,0, 0);
39     kf.set_statePre(statePre);
40
41     processNoiseCov = Mat.eye(4, 4, CV_32F);
42     processNoiseCov = processNoiseCov.mul(processNoiseCov, 1e-1);
43     kf.set_processNoiseCov(processNoiseCov);
44
45     measurementNoiseCov = Mat.eye(2, 2, CV_32F);
46     measurementNoiseCov = measurementNoiseCov.mul(measurementNoiseCov, 1e-1);
47     kf.set_measurementNoiseCov(measurementNoiseCov);
48
49     errorCovPost = Mat.eye(4, 4, CV_32F);
50     errorCovPost = errorCovPost.mul(errorCovPost, 0.1);
51     kf.set_errorCovPost(errorCovPost);
52 }
53
54 private void kalmanMatrixInit2() {
55     //      CHECKING KalmanFilter Matrix:
56     //      transitionMatrix : 2x2
57     //      measurementMatrix: 2x2
58     //      statePre : 1x2
59     //      processNoiseCov: 2x2
60     //      measurementNoiseCov: 2x2
61     //      errorCovPost: 2x2
62     transitionMatrix = Mat.ones(2, 2, CV_32F);
63     kf.set_transitionMatrix(transitionMatrix);
64
65     measurementMatrix = Mat.eye(2, 2, CV_32F);
66     kf.set_measurementMatrix(measurementMatrix);
```

```
67
68     statePre = new Mat(2,1, CV_32F);
69     statePre.put(0, 0, 0.5f);
70     statePre.put( 1,0, 0.5f);
71     kf.set_statePre(statePre);
72
73     processNoiseCov = Mat.eye(2, 2, CV_32F);
74     processNoiseCov = processNoiseCov.mul(processNoiseCov, 1e-1);
75     kf.set_processNoiseCov(processNoiseCov);
76
77     measurementNoiseCov = Mat.eye(2, 2, CV_32F);
78     measurementNoiseCov = measurementNoiseCov.mul(measurementNoiseCov, 1e-1);
79     kf.set_measurementNoiseCov(measurementNoiseCov);
80
81     errorCovPost = Mat.eye(2, 2, CV_32F);
82     errorCovPost = errorCovPost.mul(errorCovPost, 0.1);
83     kf.set_errorCovPost(errorCovPost);
84 }
85
86 private void kalmanWork() {
87     //kalman
88
89     //update
90     kf.get_measurementMatrix().put(0, 0, targetPoint.x*videoWidth);
91     kf.get_measurementMatrix().put(1, 0, targetPoint.y*videoWidth);
92
93     Mat measure = new Mat(2, 1, CV_32F);
94     measure.put(0, 0, targetPoint.x*videoWidth);
95     measure.put(1, 0, targetPoint.y*videoHeight);
96 }
```

```
97     // correct
98     Mat test = kf.correct(measure);
99     Log.d(TAG, "correct: " + test.size());
100
101    // predict
102    Mat prediction = kf.predict();
103    if(prediction != null) {
104        Log.d(TAG, "prediction size: " + prediction.size());
105        // Log.d(TAG, "prediction1 : " + prediction .get(0, 0)[0]);
106        // Log.d(TAG, "prediction2 : " + prediction .get(0, 0)[1]);
107        // Log.d(TAG, "prediction3 : " + prediction .get(1, 0)[0]);
108    } else {
109        Log.d(TAG, "predictionIsNull");
110    }
111    if(prediction.get(0,0) != null && prediction.get(1,0) != null) {
112        PointF predictPt = new PointF((float) prediction.get(0, 0)[0]
113                                         targetPoint = predictPt;
114                                         targetPoint.x = targetPoint.x / videoWidth;
115                                         targetPoint.y = targetPoint.y / videoHeight;
116
117        Log.d(TAG, "prediction point: " + predictPt);
118    }
119 }
120
121 public boolean isTargetInVision() {
122     targetPoint = getTargetPoint(testingActivity.yuv);
123     if(targetPoint != null) {
124         return true;
125     }
126     return false;
```

127 }

原版PID控制核心代码

```
1 //-----Flight Control-----
2 @Subscribe(threadMode = ThreadMode.ASYNC)
3 public void flightControl(TargetPointResultEvent targetPointResultEvent)
4     // receive the target detection result
5     targetPoint = targetPointResultEvent.targetPoint;
6     setFlightControlData(targetPoint);
7 }
8
9
10 float errorXI = 0;
11 float errorYI = 0;
12 float altitudeI = 0;
13 float errorXPre = 0;
14 float errorYPre = 0;
15 float altitudePre = 0;
16 public void setFlightControlData(PointF targetPoint) {
17
18     float errorXCur = targetPoint.x - 0.5f;
19     float errorYCur = targetPoint.y - 0.5f;
20     errorXI += errorXCur;
21     errorYI += errorYCur;
22
23     //-----pitch-----
24     // specific
25     float pitchP = -0.0001f;
26     float pitchI = 0f;
27     float pitchD = -0.01f;
```

```
28     // pitch angle: (0.00001, 0.001)
29     float pitchAngle = pitchP*errorXCur+pitchI*errorXI+pitchD*(errorX
30
31     //--------------------------------roll-----
32     // specific
33     float rollP = 0.1f;
34     float rollI = 0.001f;
35     float rollD = 0.1f;
36     // roll angle range in (-0.4, 0.4)
37     float rollAngle = rollP*errorXCur+rollI*errorXI+rollD*(errorX
38
39     //--------------------------------vertical-----
40     // specific
41     float verticalThrottleP = -0.1f;
42     float verticalThrottleI = -0.0001f;
43     float verticalThrottleD = 0.1f;
44     float altitudeCur = ((Aircraft)RDApplication.getProductInstan
45             .getState().getAircraftLocation().getAltitude();
46     altitudeI += altitudeCur;
47     // roll angle range in (-5, 5)
48 //     float verticalThrottle = verticalThrottleP *altitudeCur+ verticalThrottleI *altitu
49     float verticalThrottle = 0;
50
51     flightControlData = new FlightControlData(pitchAngle, rollAng
52
53     errorXPre = errorXCur;
54     errorYPre = errorYCur;
55     altitudePre = altitudeCur;
56 }
57
```

```
58  public void setFlightControlData2() {  
59      //for test  
60  
61      //build the center point  
62      float latitudeCur = (float) ((Aircraft) RDApplication.getProductInstan  
63          .getState().getAircraftLocation().getLatitude();  
64  
65      float longitudeCur = (float) ((Aircraft) RDApplication.getProductInsta  
66          .getState().getAircraftLocation().getLongitude();  
67  
68      float altitudeCur = ((Aircraft) RDApplication.getProductInstance()).ge  
69          .getState().getAircraftLocation().getAltitude();  
70  
71      Point3D flightPoint = new Point3D(latitudeCur, longitudeCur, altitudeC  
72  
73      float errorXCur = flightPoint.x - target.x; //0.1  
74      float errorYCur = flightPoint.y - target.y;  
75      errorXI += errorXCur;  
76      errorYI += errorYCur;  
77  
78      //-----pitch-----  
79      // specific  
80      float pitchP = -0.01f;  
81      float pitchI = 0.0000001f;  
82      float pitchD = 0.001f;  
83      // pitch angle: (0.00001, 0.001)  
84      float pitchAngle = pitchP*errorXCur+pitchI*errorXI+pitchD*(errorXCur-  
85  
86      //-----roll-----  
87      // specific
```

```

88     float rollP = 0.4f;
89     float rollI = 0.0000001f;
90     float rollD = 1f;
91     // roll angle range in (-0.4, 0.4)
92     float rollAngle = rollP*errorXCur+rollI*errorXI+rollD*(errorX
93
94     //-----vertical-----
95     // specific
96     float verticalThrottleP = -0.1f;
97     float verticalThrottleI = 0;
98     float verticalThrottleD = -0.1f;
99
100    altitudeI += altitudeCur;
101    // roll angle range in (-5, 5)
102    float verticalThrottle = verticalThrottleP*altitudeCur+vertical
103
104    flightControlData = new FlightControlData(pitchAngle, rollAng
105
106    Log.d(TAG, "Flight control data: " + flightControlData.getPitch(
107    Log.d(TAG, "Flight body data: " + flightPoint.x+" "+flightPoint.
108
109    errorXPre = errorXCur;
110    errorYPre = errorYCur;
111    altitudePre = altitudeCur;
112 }

```

MPC控制MATLAB仿真代码

```

1 function [M,C,Q_bar,R_bar,G,E,H,U_k] = MPC_Zero_Ref(A,B,N,x_k,Q,R,F
2 n = size(A,1);           % 是An*矩阵, 得到nn
3 p = size(B,2);           % 是Bn*矩阵, 得到pp

```

```

4    % 初始化矩阵, 矩阵是 $MM(N+1)*n*$ 的n
5    % 矩阵上面是一个 $Mn*$ 的矩阵, 这一步先把矩阵下半部分初始化为 $nI0$ 
6    M = [eye(n); zeros(N*n, n)];
7    % 初始化矩阵, 这一步令它有 $C(N+1)*N*$ 个 $P0$ 
8    C = zeros((N+1)*n, N*p);
9    tmp = eye(n);           % 定义一个 $n*$ 的矩阵 $nI$ 
10   for i = 1:N            % 循环次N
11       rows = i*n+(1:n); % 定义当前行数, 从i*开始, 共行nn
12       C(rows, :) = [tmp*B, C(rows-n, 1:end-p)]; % 将矩阵填满C
13       tmp = A*tmp;        % 每一次将左乘一次tmp
14       M(rows, :) = tmp; % 将矩阵填满M
15   end
16   % 定义 $Q_{bar}$ 
17   S_q = size(Q, 1);      % 输出的维度Q
18   S_r = size(R, 1);      % 输出的维度R
19   Q_bar = zeros((N+1)*S_q, (N+1)*S_r); % 初始化为全矩阵 $Q_{bar}0$ 
20   for i = 0:N            % 循环N次+1
21       % 将写到对角线上 $QQ_{bar}$ 
22       Q_bar(i*S_q+1:(i+1)*S_q, i*S_q+1:(i+1)*S_q) = Q;
23   end
24   Q_bar(N*S_q+1:(N+1)*S_q, N*S_q+1:(N+1)*S_q) = F;
25   % 定义 $R_{bar}$ 
26   R_bar = zeros(N*S_r, N*S_r);
27   for i = 0:N-1
28       R_bar(i*S_r+1:(i+1)*S_r, i*S_r+1:(i+1)*S_r) = R;
29   end
30   % 求解、、 $GEH$ 
31   G = M' * Q_bar * M;
32   E = C' * Q_bar * M;
33   H = C' * Q_bar * C + R;

```

```
34      % 最优化  
35      f = (x_k'*E')';           % 定义矩阵f  
36      U_k = quadprog(H, f); % 求解最优化U_k  
37  end
```
