燕山大學 YANSHAN UNIVERSITY

本科生毕业设计（论文）

论文题目　基于视觉的自动追踪控制系统设计

作者姓名　韩云海

专　　业　机械设计制造及其自动化

指导教师　史小华

2019 年

燕山大学本科生毕业设计（论文）

基于视觉的自动追踪控制系统设计

学　　院：机械工程学院
专　　业：机械设计制造及其自动化
姓　　名：韩云海
学　　号：150101010279
指 导 教 师：史小华
答 辩 日 期：2019 年 6 月

燕山大学毕业设计(论文)任务书

学院：　　机械工程学院　　系级教学单位：　　机械电子工程系

学号		学生姓名	韩云海	专 业 班 级	机电 15
题目	题目名称	基于视觉的自动追踪控制系统设计			
	题目性质	1.理工类：工程设计 （ ）；工程技术实验研究型（ √ ）； 理论研究型（ ）；计算机软件型（ ）；综合型（ ）。 2.文管类（ ）；3.外语类（ ）；4.艺术类（ ）。			
	题目类型	1.毕业设计（ √ ）　　2.论文（ ）			
	题目来源	科研课题（ ）　生产实际（ √ ）自选题目（ ）			
主要内容	课题的目标是设计一套基于视觉的机器人自动识别、自动追踪系统： 1、搭建机器人运动底盘和三维云台； 2、图像处理模块的应用； 3、图像处理算法实验比较和分析。 4、性能验证和展示。				
基本要求	1.结构设计要正确、合理，具有可行性； 2.要考虑经济性； 3.考虑操作性； 4.按毕业设计要求整理设计资料；				
参考资料	图像处理类书籍、论文 电机选型手册 RM 俱乐部样机				
周 次	秋 19—开学	1 — 3 周	4 — 9 周	10 —13 周	14 — 15 周
应完成的内容	完成资料调查，研究现状和进展、技术等文献综述，准备好开题报告。	系统结构方案设计。零部件分析和计算和选择。电气系统元件选择	零部件设计，撰写毕业论文	零部件修改，论文修改	
指导教师：史小华 职称： 讲师　　2019 年 2 月 26 日					

摘 要

本文阐述了如何搭建基于视觉的自动追踪控制系统，其研究成果已实际运行在燕山大学竞技机器人俱乐部的机器人上。传统的追踪控制系统需要操作人员的远程控制，且由于图像延迟的存在，导致追踪准确度不高。本文通过在 Ubuntu 系统上运行 OpenCV 开源视觉库，通过机器人炮管上搭载的工业相机，实时读取当前视野图像，并对图像进行分析，判断此图像中是否有敌方机器人的装甲板，若存在装甲板，则根据装甲板上的实际尺寸大小和识别在摄像头成像仪上的像素位置的对应关系，通过 PnP 算法，求解出敌方机器人装甲板在我方机器人云台坐标系里的三维坐标，从而得到当前时刻下云台 pitch,yaw 的角度信息，最终控制云台转动到相应角度并实现射击。除此之外，本系统考虑到了弹道下坠的问题，根据弹道下坠的物理模型和目标的三维空间信息，对炮管的角度信息进行修正，以获得较高的射击命中率。本系统主要由 C++语言编写而成，包括视觉算法代码和串口通信代码。除此之外，由于此系统是运行在 Ubuntu 操作系统上，因此还有 shell 语言编写的脚本程序，包括 C++程序的编译链接程序代码和开机自启动程序代码。经过实际运行测试，本系统下的程序的运行帧率可以达到 130 帧，可以保证实际使用的需要。

关键词：机器人视觉系统；自动追踪；电动射击系统；嵌入式控制系统；弹道补偿

Abstract

This paper demonstrates how to build an automatic tracking system based on vision methods. The system has successfully been used on robots made by YSU-Eagle. In the past, the operators needed to control the rotational movement of robots' gimbal units by themselves, which led to poor performance of firing due to large delay of image transmission. In this paper, I build a system which can free operators from this hard task. The system runs on Ubuntu operating system and OpenCV has been installed on it. With the industrial camera fixed on the gun, the system could read image in real time and use image analysis method to find armors. Then, if there is any armor in this image, it uses PnP algorithm to calculate the three-dimensional coordinates in the camera coordinate system and then turns them into the pitch&yaw angles. This algorithm needs to know the true size of the armor and the pixel coordinates of each corners of the armor. Besides, the system has taken bullet's falling into consideration. Based on the model of bullet's falling, the system adds some compensation value to the pitch&yaw angles. This system is mainly compiled by C++ programming language, including vision algorithm codes and serial communication codes. In addition, because it runs on Ubuntu operating system, there are some scripts written in shell programming language to automatically manage this system, including the compiling links of C++ programming language and start-up program. During real operating test, the frame rate can reach 130 FPS and that satisfies the actual needs.

Keywords: visual system for robots; automatic tracking system; electric shooting system; embedded control system; compensation for bullets' trajectory

目 录

第 1 章 绪　论

1.1 课题背景及研究的目的和意义

机电一体化概念最早出现在 1971 年日本杂志《机械设计》的副刊上，其是一种将械技术、电工电子技术、微电子技术、信息技术、传感器技术、接口技术、信号变换技术等多种技术进行有机地结合，并综合应用到实际中去的综合技术[1]。凭借着对机电一体化技术的深入研究和不断尝试，日本在上世纪 70 年代紧跟世界发展潮流，不断提高在各种工业产品的市场占有率，如汽车，家电等。而我国在当时才逐步开始在机电一体化方面进行研究和应用，起步晚，因此在全球经济市场的竞争中，我国面临着严峻的形势。

另一方面，步入 21 世纪以后，互联网产业成为了最受瞩目的产业。人工智能(AI)和计算机视觉(CV)以及自然语言处理(NLP)等许多新兴技术让人们看到了无限的可能性。除此之外，机器人领域[2]，作为一个最具代表性的交叉领域，也吸引住了许多人的目光。2015 在北京举办的世界机器人大会上，习近平主席对机器人领域的发展寄予厚望，他表示，中国将机器人和智能制造纳入了国家科技创新的优先重点领域，中国将积极追赶智能产业蓬勃兴起的潮流，布局中国自身经济转型的同时，为人类文明发展贡献中国智慧和中国力量[3]。

综合来看，中国虽然在 20 世纪末的机电一体化潮流中稍逊一筹，但是在当下的互联网浪潮中紧跟不舍。因此，作为机电专业的学生，我们应当将计算机理论知识与我们自身的专业知识结合起来，真正做到“软硬结合”，既设计出满足使用并且易于维修的机电系统，又设计出代码严谨容错率高的计算机软件系统，争取成为未来能独当一面，规划整个系统的总工程师！

因此，我结合了我在 RM 俱乐部的经历，选择了这个选题。此题目既包括了机器人底盘及云台的设计，云台电机的 yaw 和 pitch 轴稳定性和快速性的调试，也包括了图像处理模块视觉系统的设计，还包括了下位机控制系统与上位机视觉系统之间的数据通信，这些内容能有效地提高我对于机电计系统的理解，为以后的学习打下基础。

1.2 国内外研究现状

近年来，国内外对基于视觉的自动追踪问题的研究正处于高潮，许多重要的国际期刊以及许多重要的国际会议发表了大量的有关视觉跟踪方面的论文。基于视觉的自动追踪能够引起广泛讨论是由于它能够应用于民用[4]和军事的许多领域。例如，在视频监控领域，最常见的是对于停车场、民宅、学校、公共场合等地的监控[5-7]，以防止偷盗，破坏行为的发生。1997 年，美国国防部高级研究项目署设立了以卡内基梅隆大学为首，麻省理工学院等高校参与的视觉监控项目 VSAM[8](Visual surveillance and monitoring),该系统可应用于民用场景及战场的实时监控。随后，卡内基梅隆大学又建立了一个校园监控系统[9]，以保障校园安全。除此之外，在交通系统中，视觉跟踪技术的研究也有非常广阔的应用前景，包括行人行为判定，车辆异常行为检测，智能车辆等很多方面。Coifman 等人[10]建立了一个基于视频图像处理系统的交通监控系统，此系统可以用来监控交通流量以及对不同车型进行统计。这项研究在国内外非常普遍，主要研究解决的问题是如何准确实现对车辆的分割和跟踪。道路上车辆异常行为检测在交通事故的预防和处理方面也有重要意义。Tai 等人[11]设计了一个可以用于交通事故检测的视频监视系统，能够自动检测视野中的运动车辆并预测其运动轨迹。Haag 和 Nagel[12]专门对机动车辆的跟踪问题进行了研究。Pai 等人[13]针对十字路口的行人检测和跟踪进行了研究以保证驾驶员可以在十字路口安全驾驶。Masound 和 Papanikolopoulos[14]通过对道路行人的跟踪并计数，为道路交通管理提供了信息。智能车辆是目前计算机视觉研究中的一个热点，其最终目的是实现自动驾驶车辆。

相比国外，我国在视频监控领域方面的研究起步稍晚，但是在政府各部门的大力支持下，也在这方面取得了很大的进展。20 世纪 90 年代初，一些高校和交通研究机构引荐国外先进理论和控制思想，开始了城市交通车辆系统技术研究，并在一些交通系统比较发达的地方设定测试点进行试验，取得了一定成果。中科院自动化研究所同雷丁大学展开合作，双方建立了共同的软件平台，并发起了国际视觉监控学术研讨会。另外，一些公司也在视觉监控技术上取得一定进展，在这里就不一一叙述了。

1.3 研究内容及拟解决的主要问题

研究的基本内容包括：

（1）参与设计搭建具有两自由度的云台，并安装在麦轮驱动的机器人底盘上。

（2）对安装好的云台进行控制算法调试，以保证其满足位置响应和速度响应的要求。

（3）通过机器人上搭载的微型计算机和安装好的摄像头，运用计算机视觉的算法，识别出目标物品，并根据测距算法得到位置信息，调整云台转角，使达到实时追踪的目的。

拟解决的主要问题：

（1）云台的机械结构需要满足一定的要求，以便于达到预定的控制效果。

（2）机械结构尽可能设计美观，且牢固，同时减少加工件的使用，以减少成本。

（3）控制算法应尽量简单，再保证控制精度的前提下，提高控制器的运算频率。

（4）设计视觉识别测距算法，并对其做适当优化，使帧率满足实际使用的要求。

（5）对接视觉和电控系统，将视觉系统解算出来的角度传给电控系统，调整云台的转角。

1.4 研究步骤及方法

研究步骤主要包括：机械结构的理论设计和实物搭建；电控系统的设计和调试；视觉系统的设计；调试和不同算法性能比较；视觉电控系统的对接。

研究方法包括：运用理论知识指导云台机械机构的设计，包括运用理论力学知识分析云台的机械性能，运用机械设计和机械制造方面的知识设计云台的结构；通过相应的现象及经验，对于控制系统进行整定，使达到满意的控制精度；通过学习的 C++编程知识和计算机视觉算法，设计出鲁棒的视觉检测系统，并将我们解算得到的角度信息传递给下层的单片机控制系统，以此来控制云台的运动，达到实时追踪的效果。

第 2 章 方案对比分析

此系统可分为三个独立的子系统，分别是机械系统，电控系统和视觉系统。因此我对三个系统单独分析方案。

2.1 机械系统

机械系统的主要目的是设计出二自由度的云台，每个自由度都由一个单独的动力源进行驱动，考虑到结构的紧凑型和控制的可行性，因此选择使用最广泛的电机驱动方式。考虑到直流供电系统，我们可以有以下三种电机驱动方式：直流电机；直流无刷电机；步进电机。

查阅资料可知，直流无刷电机具有响应迅速，起动转矩大的特点，因此云台设计中采用直流无刷电机作为驱动源。此外，由于采用的电动射击方式，因此采用拨盘和摩擦轮组成的电动射击系统，此发射系统模型与传统的火药爆炸产生推力为主的发射有较大区别。但是，本毕业设计中不是以机械设计为主，且本人大部分精力也不是放在机械设计上，而是在视觉硬件系统的搭建和软件代码的撰写上，所以机械系统部分不着重介绍。

2.2 电控系统

电控系统的主要目的是对两个直流无刷电机进行驱动，并通过控制算法使其具备快速响应不超调的特点。根据我以往比赛经验，有三种控制方案可供选择：

（1）PID 控制算法

PID 控制器（比例-积分-微分控制器）是一个在工业控制应用中常见的反馈回路部件，由比例单元 P、积分单元 I 和微分单元 D 组成。PID 控制的基础是比例控制；积分控制可消除稳态误差，但可能增加超调；微分控制可加快大惯性系统响应速度以及减弱超调趋势。

（2）ADRC 控制算法(自抗扰控制技术)

传统 PID 有限制，即调节速度和超调一定同时存在，想要得到较好的控制效果，用现代控制理论解决，要知道精确的系统模型。但是对具体模型的建立有些复杂繁

琐，要么是通过理论对系统进行建模，通过动力学模型分析系统内的响应特性，要么通过系统辨识的方法，通过多项式的传递函数去拟合系统的响应特性。然后通过系统的响应特性，设计反馈网络进行控制，并可以通过设计校正网络，提高系统的抗扰动特性以适应外界或内部的扰动情况。与此相反的是，一套整定好的 PID 参数可能只在当前的物理参数下控制效果良好，当物理条件改变后，比如重量变重，就无法继续使用了，即不抗扰动。我在网上看到了 ADRC 这种综合了 PID 和现代模型的优势的控制算法。其既保留了 PID 控制器的特点(黑箱属性)又具备现代模型控制的抗绕特点。

（3）现代控制理论

建立系统的物理模型控制，具有良好的抗扰动性，然而缺点是只有在物理模型准确的情况下获得好的控制效果，若实际机器人的物理属性与理论模型相差较大，则控制效果不好。

综合比较各个控制算法实现的难易性以及此课题的需求，我认为最常见 PID 控制器即可达到要求，因此就选择 PID 控制算法作为控制器设计方案。

2.3 视觉系统

视觉主要的主要目的是识别目标物然后得到目标空间位置，最后将此位置换算到转角改变量传输到控制系统。这里可以细分为三个环节，识别，测距，传输。

目标识别算法有很多，如最常见的 RCNN，SSD，YOLO 等，由于需要识别的物体的特征十分独特，因此采用基于特征的识别方法。待识别的物体如图 2-1 所示。

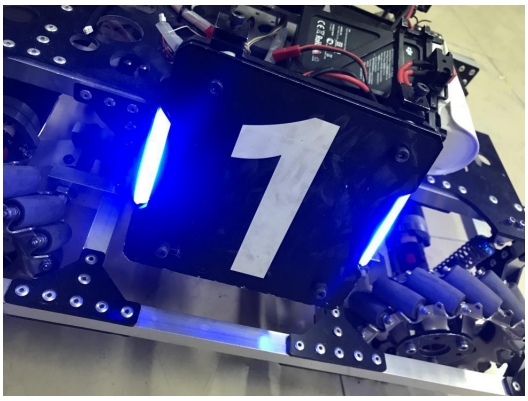


图 2-1 待识别的物体(红蓝灯条装甲板)

测距部分，常用的测距算法主要包括单目测距[15]，双目测距[16]，多目混合测距。

测距算法的选择，主要是时间和成本的考虑。双目测距的结果一般来说会更精准些，但是它要对两幅图像进行图像校正，极线匹配，比较耗时。单目测距无需这些工作，但是由于单目测距对深度维度的信息缺失，因此需要知道待测物体实际的物理尺寸，才能通过特定的数学算法求得实际的物理尺寸。多目测距本质上是结合了单目和双目测距，通过一定的决策法则对单双目进行融合，亦或是运用双方的结果根据模型产生一个新的更精确的结果。视觉测距准备先理论学习单目和双目测距，编程实现并比较各自运行时间开销。若单目视觉的准确度和帧率都能达到理想的要求，则采取单目视觉。

传输部分，此部分没有理论分析的需要，只需要完成这一目的即可。为了方便起见，我决定通过 Ubuntu 系统自带的串口驱动(可自行设置波特率，数据位，奇偶校验位等)和 CH340 模块实现有线的串口传输数据。

2.4 技术经济性分析

与上述分析相同，我把整个系统同样分为三个子系统分析。

从技术经济性角度分析，各个系统的组成包括硬件部分和软件部分。硬件部分主要包括需要购买的材料和零部件，软件部分主要包括设计所使用的各类软件和编程工具。首先列出各个系统所需要的硬件设备：

（1）机械系统

机械设计上应该尽量避免加工件的使用，以减少系统的成本。电机和电调选用 RM 俱乐部内物资，无需额外采购。除去这些需要设计的较大零部件，机械系统还包括很多标准零件，如螺钉螺母，垫片，铜柱等。

（2）电控系统

主控板选择 DJI 开发的主控板，可以直接接 24V 直流电源。电源也使用 DJI 开发的电源模块。正如上文所述，控制算法我决定采取 PID 控制器[17]，其参数整定过程比较熟悉，可以胜任此次课题的要求。我使用的传感器包括陀螺仪，光电开关，微动开关，稳压模块，CAN 通信模块和各类电子制作中常使用的设备如焊烙铁，热风枪。

（3）视觉系统

由于微机系统的价格昂贵，我选择使用 RM 俱乐部内部购买的英特尔微型计算

机 NUC 系列。摄像头的选取上需要综合考虑摄像头感光元件能提供的最大帧率，摄像头的自身畸变，摄像头的价格，摄像头的驱动方式，摄像头的曝光是否可调等。通过对比多家摄像头提供商提供的摄像头参数资料，最终选择了由深圳昊天科技有限公司开发的工业相机系列，其分辨率为 1280×1024，且读取帧率可以达到 240 帧。除此之外，相机的曝光可以自由修改，并且其提供了许多底层的图像处理函数接口，便于我们二次开发使用。除此之外，视觉系统还需要诸如显示器，分线器，转接线等常见设备。

其次是所使用的软件设备：

机械系统：SolidWorks 软件设计云台机械机构，并根据实际使用的材料设置不同零件的质量属性，最终得到云台装配体的质心位置。

电控系统：Keil5 软件编写 C 语言的控制程序，使用 jlink 仿真器和 jscope 仿真软件，根据陀螺仪反馈的角度数据曲线和电机电流曲线，在线整定 PID 参数值，使得电机达到最优的控制效果。

视觉系统：英特尔微型计算机 Mini PCNUC7i7DNHE，并在其上运行 Ubuntu 操作系统。软件编程上，需要熟悉 Linux 操作命令，熟悉 C++编程语言，熟悉 OpenCV 开源的计算机视觉库，熟悉 Linux 下 C++语言的编译工具 cmake 的使用，熟悉编写 Linux 的脚本程序。

下面的三个章节中我会介绍毕业各个系统的设计思路。由于我主要负责的是视觉系统的搭建，因此机械系统和电控系统我只会稍加阐述，具体内容包括系统的设计思路和最终成果。机械系统和电控系统是燕山大学 RM 俱乐部所有人员的共同努力的成果，而我则参与过其中部分的研发过程。

2.5 本章小结

本章将总系统分为了机械系统、电控系统和视觉系统，并对其进行了方案的分析及选取。确定了基本的机械系统、选取了 PID 控制算法作为控制器设计方案，视觉系统选取基于特征的识别方法，对单目视觉和双目视觉进行了比较确定了选取的标准，采用 Ubuntu 系统自带的串口驱动和 CH340 模块实现有线的串口传输数据。从技术经济性角度出发选取了机械系统、电控系统和视觉系统的硬件及软件。

第 3 章 机械系统设计

机械系统设计主要包括云台结构的设计，具体零部件的采购以及最终的组装。

3.1 云台结构设计

云台结构设计的主要目的是方便电控 PID 算法加快云台响应速度且减少超调量以达到良好的控制效果。因此在机械结构的设计阶段，所有成员得到共识，即云台的总质量应该越轻越好，这样可以减少系统的转动惯量，获得较快的响应速度。除以之外，因为 PID 算法适应于线性系统，因此我们需要尽量保证系统的质心位于 pitch 和 yaw 的转轴轴线上，这样可以保证云台在任何转角的时候由于重力 G 和 $\cos(\omega t)$ 的乘积产生的非线性项保持为 0。综上所述，我们综合考虑了机械设计的要求，保证能够便捷得安装和拆卸，各个零部件的质量分布，以减轻质量以及调整质心位置以及电线布线的要求以保证电线和信号线不会在运动过程中有脱落的风险，云台的机械结构如图 3-1 所示。

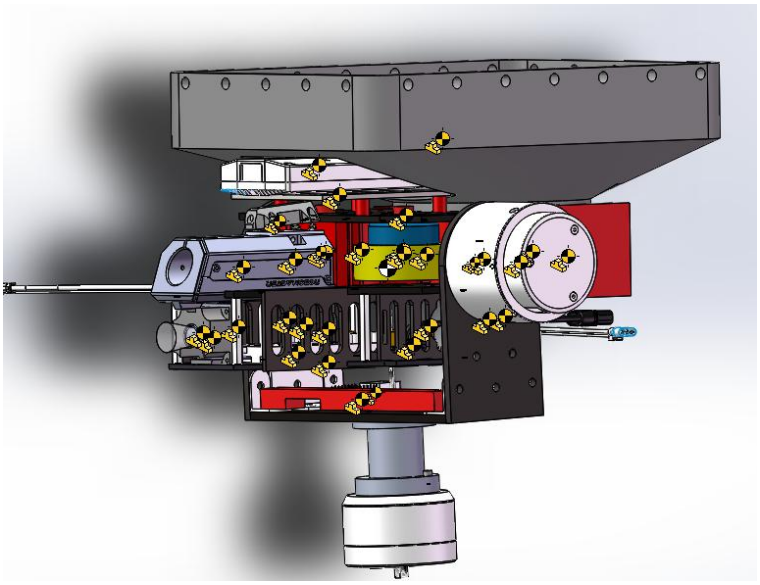
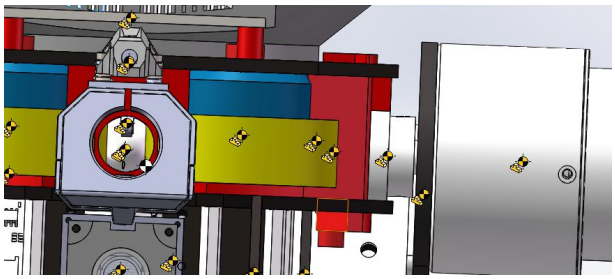
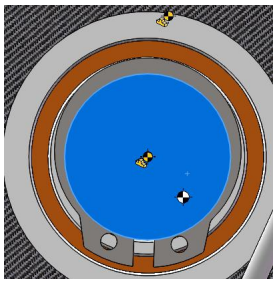


图 3-1 云台的机械结构

如图 3-2 所示，图中的黄色标记代表的是各个主要零件的质心位置，省略了全部的螺栓和部分不重要的 3D 打印件。图 3-2 中的白色标记代表的是整体的质心位置，其由所有黄色标记件的质心位置和质量通过 SolidWordks 的质量属性自动求得。



a) 分图 1

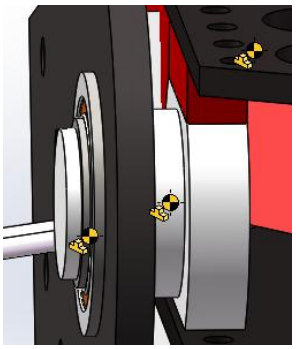


b) 分图 2

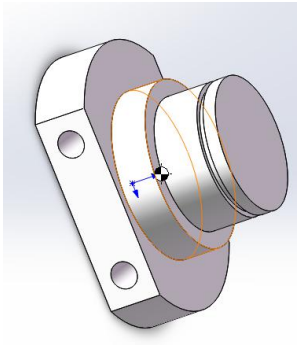
图 3-2 云台质心的分布

如图 3-3 中所示，我们可以看出，质心已经很接近 pitch 和 yaw 轴的轴线上，但是由于结构的问题以及具体材料质量的原因，其质心无法完全正好位于轴线上。从图 3-2 中可以看出，由于云台只有右侧有 RM6623 电机，因此质心肯定会比较偏右侧。

另一方面，我们出于减少成本的目的，主要使用碳纤维板，3D 打印件作为主体的零件材料，只有在需要承受较大载荷以及需要较高的定位精度的地方使用机械加工件，包括”工”型加工件(需要承受整个云台的重量)和 pitch 轴电机与云台的连接件(需要很高的定位精度)。



a) 分图 1



b) 分图 2

图 3-3 pitch 轴电机和云台的连接件

如图 3-4 所示为“工”型加工件，中间空心的目的是安放导电滑环。我们使用导电滑环的目的是当云台 360°旋转的时候，电线可以通过导电滑环到达云台，而不会缠绕在云台上。关于导电滑环的内容我放在电控部分加以阐述。加工件右侧的上下螺纹通孔的目的是将加工件连接到下方电机和上方云台。

如图 3-5 所示，是另外一种需要使用加工件的地方，因为这里需要较高的定位精度，所以采用的是机械加工件。

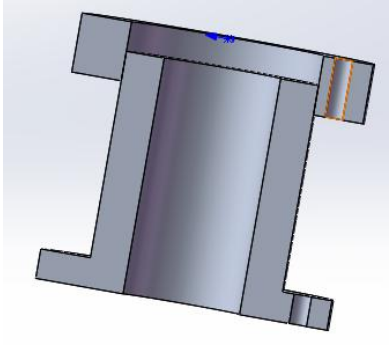


图 3-4 “工”型加工件

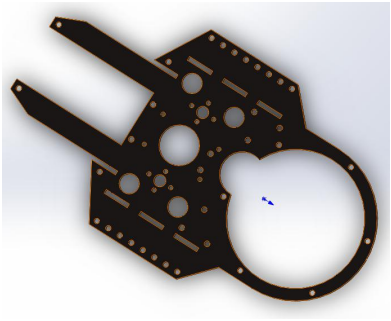


图 3-5 一块碳纤维板

如图 3-5 所示，这是其中一块碳纤维板，我们在考虑碳纤维板的结构的时候，同时考虑了安装要求，电线布线要求和减重要求。小的圆孔是用来连接螺钉(安装零件或者固定传感器)，大中型的圆孔一方面起到减重的目的，一方面可以让扎带从其中穿过，将电线固定在板上，电线有单片机供电线，电调信号线和供电线，直流无刷电机的供电线和信号线，陀螺仪的信号线，无线蓝牙信号线，图传信号线和 RM 比赛裁判系统的信号线。

3.2 零部件购买说明

在这里列出云台上全部的物品，包括电控视觉系统部件以及机械零部件。

机械零件包括：3D 打印件，如图 3-6 所示的红色零件，其中炮管是用光敏树脂打印的，其余部分是由 PLA 材料打印的，包括弹仓，拨弹盘等，花费 500 左右；碳纤维板，图一中的黑色零件，构成了云台的外部框架，花费 400 左右；加工件，一共 3 个，上文中已经列举过了，花费 400 左右；其余各类标准件，如铜柱，螺钉螺母，扎带，角件，胶轮等。

电控视觉系统包括：微型计算机 MINIPC，花费 4200；工业相机和 6mm 焦距镜

头，花费 2480；DJI 主控板，花费 400 左右；电源转接板(将电池的电源线路分线到主控板和电调)，花费 100 左右；电调和电机，包括 yaw 轴电机 6620，pitch 轴电机 6623，两个摩擦轮电机 3510，拨盘电机 3510，花费 1600 左右；陀螺仪，花费 150 左右；RM 比赛的图传和裁判系统，其余部分包括天线接收器，蓝牙模块，各类接口，电线等，花费 200 左右。

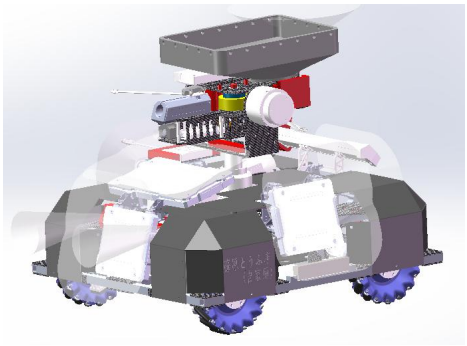
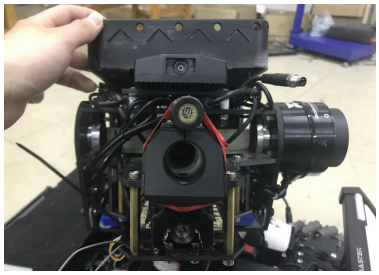


图 3-6 机器人整体三维模型

3.3 机器人云台实物

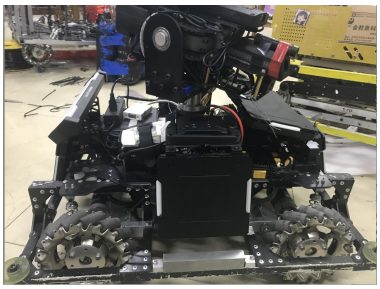
如图 3-7 所示，为机器人云台实物的几个分图



a) 分图 1

b) 分图 2

c) 分图 3



d) 分图 4

图 3-7 机器人实物图

3.4 本章小结

本章主要对机械系统的云台结构进行了设计，包括“工”形加工件、碳纤维板及其他机加工件等，并对系统的机械结构部分、电控部分及视觉控制部分的零部件的购买情况进行了简单的介绍，最后展示了机器人云台的实物图片。

第 4 章 电控系统设计

电控系统主要包括云台 pitch，yaw 轴的 PID 控制，摩擦轮和拨盘电机 PID 控制和云台电路布线。

4.1 云台 PID 控制

云台的控制主要使用的是双环 PID 进行反馈控制。对于云台 yaw 轴电机，内环速度环使用的反馈信号是固定在云台上的陀螺仪的角速度，外环位置环使用的反馈信号是 yaw 轴电机 6620 绝对式编码器的数值。通过调节内外环 PID 参数，可以使 yaw 轴得到良好的控制效果，具体表现为，能以较快的响应速度达到设定的 yaw 轴位置，且运动过程中没有抖动，电机脱力（由于 PID 设置死区的原因）等现象。对于 pitch 轴同理，内环使用的反馈信号是板载陀螺仪反馈的角速度，外环使用的是 pitch 轴电机 6623 绝对式编码器的数值。

如图 4-1 所示是调试的效果图，使用的是 JLink 工具和 JScope 软件在线调试，能通过数据曲线判断当前参数的控制效果，在 Keil5 中通过 debug 模式在线调整参数再比较曲线。

如图 4-2 所示，图中红线表示的是速度环 PID 计算后的输出，紫线表示的是位置环的实际值。当紫线趋于稳定的时候，即位置保持不变的时候，速度环的输出很小接近于 0。当紫线出现下降趋势的时候，即位置开始改变的时候，速度环的本质作用是保持运动平稳，增加系统的阻尼，因此当云台在外力的作用下产生速度的时候，速度环的输出就会增加，以抵消这个外界产生的速度。图 4-2 的曲线结果也与此推断一致，在图 4-1 中，黄线是实际速度，蓝线是只有比例项赋值的 PID 输出。我们可以看到当速度产生后，PID 速度环的输出是逆速度方向的。通过上图的曲线，我们一方面可以通过曲线比较参数性能，另一方面可以通过底下变量状态栏的具体数据判断程序执行是否正确。

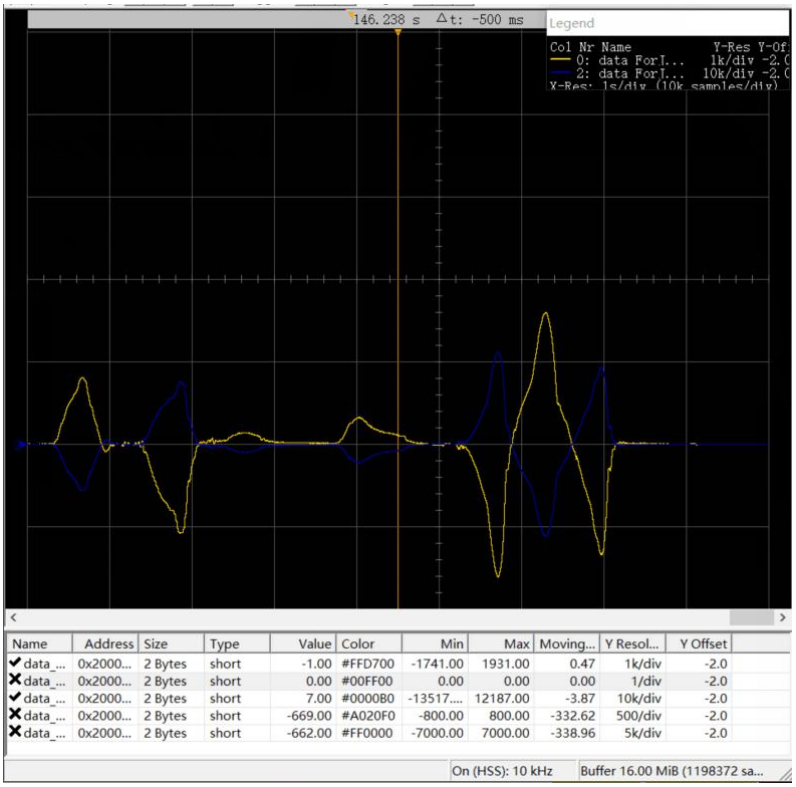


图 4-1 调试效果图

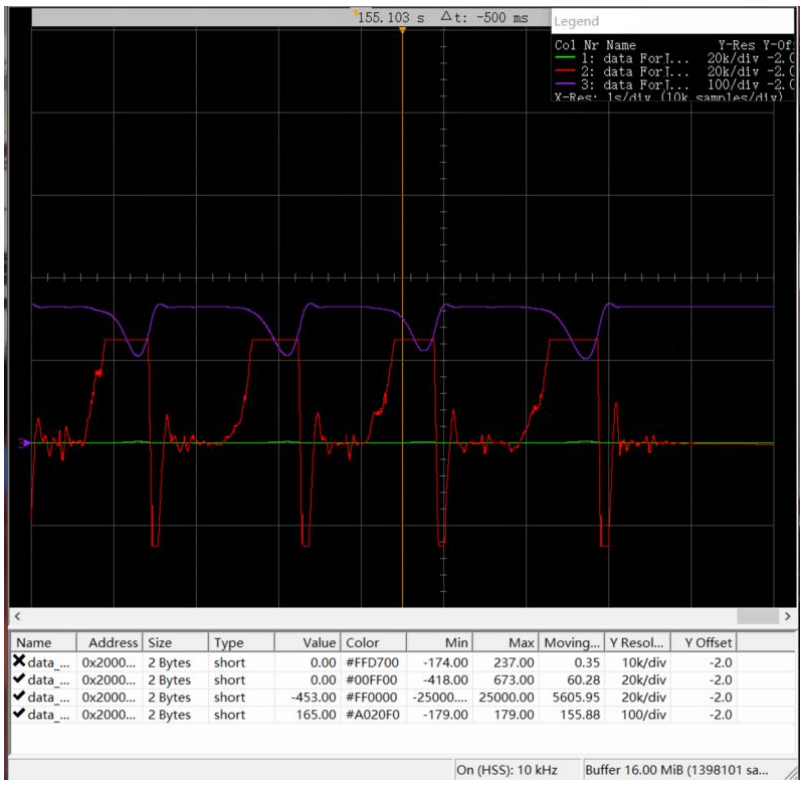


图 4-2 JScope 软件调试效果图

4.2 发射机构 PID 控制

摩擦轮和拨盘电机的控制与发射要求有关，摩擦轮转速与射出子弹的初速度有关，拨盘电机的转速与单位时间内发射子弹的数量有关。我们对摩擦轮进行闭环控制，减少其转速波动，稳定在我们设定的转速值上。同理，我们对拨盘电机的转速进行闭环控制，以保证射频稳定，不会超过比赛规定的机器人热量上限。

云台电机和摩擦轮拨盘电机的控制都是在 Keil5 软件上通过 C 语言编写程序。

4.3 云台电路布线

云台电路布线主要考虑的难点有两个，其一是由于云台 360°运动的要求，电线不能直接从底盘引出到云台上，必须经过导电滑环；其二是电线在云台上本身的布置要求，即电线不应在任何情况下有松动脱落的风险。

如图 4-4 所示是一个 24 路 2A 的导线滑环。导电滑环参数 24 路指的是一共从滑环内部穿过了 24 跟导线，2A 是每根导线上可以承受的最大电流。这些线路的分配情况为：电源分配了 12 路，正负极各 6 路。由于云台上共有 5 个电机，需要保证较大电流才能满足同时驱动的要求，因此需要分配 12 路导线承受较大的峰值电流。5 路分配给了工业相机。

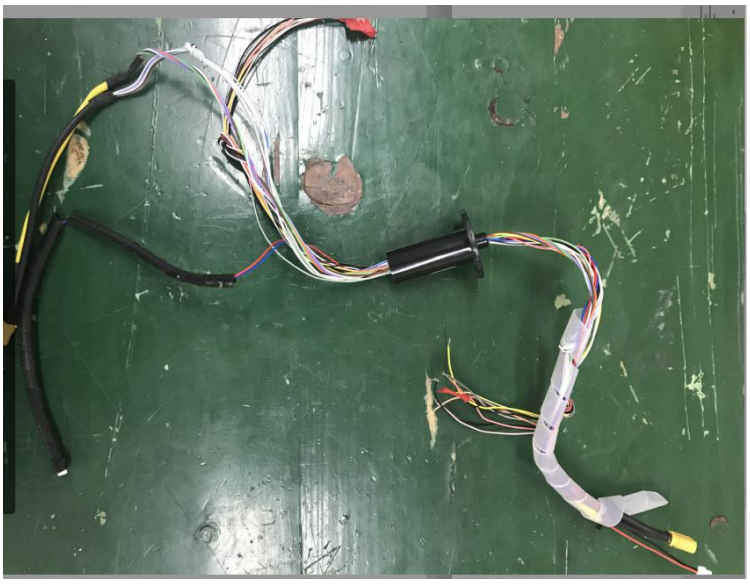


图 4-4 导电滑环

图 4-5 摄像头

如图 4-5 所示，相机是通过 USB 线传输信号，因此一开始我以为只需要 4 路信号线就可以将云台上的摄像头引入到底盘上的微型计算机 MINI PC 上，PC 设备的资源界面如图 4-6 所示。然而，在实际操作中发现，如果只接入摄像头的这四根线，PC 会无法检测到摄像头设备。

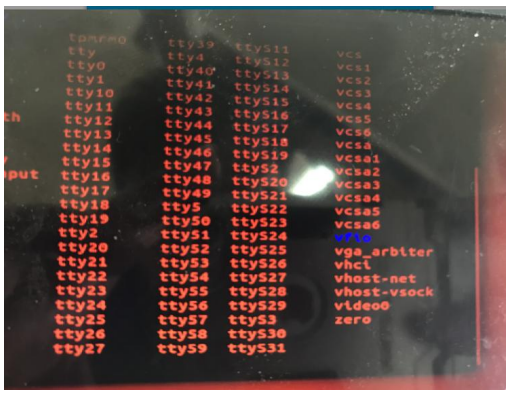


图 4-6 PC 设备资源界面

查阅资料后得知，许多 USB 信号线其实是有 5 根线的，如图 4-7 所示，第 5 根线是一层金属箔层，围绕在内部四根线外面，其目的是减少信号干扰。

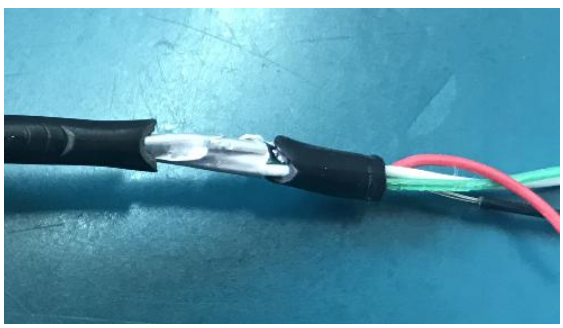


图 4-7 USB 数据线

我们首先采取的是双绞线的方式将四根线缠绕在一起。双绞线是工程上常用的布线方式，能有效减少信号之间的串扰。此时，设备资源上出现了这个摄像头，但是当我们实际运行程序，读取摄像头的数据的时候，发现数据干扰严重，画面呈现“雪花状”，且伴随摄像头传过来的错误提示“数据干扰严重”，运行一段时间过后，程序就报错终止运行了。最后我们在导电滑环中加上了第 5 根导线，连接的是 USB 两侧的金属箔层。这时，我们在运行程序的时候，画面清晰且没有错误提示。除此之外，市面上也有专门针对 USB 信号线的导电滑环，如图 4-8 所示。

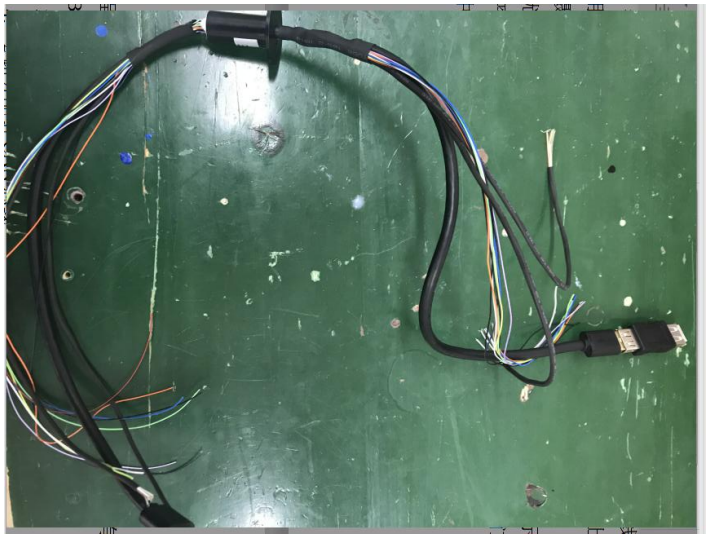


图 4-8 USB 线导电滑环

4 路分配裁判系统：RM 比赛过程中，位于云台上的主控板需要读取机器人底盘上的裁判系统的数据。

2 路分配给 CAN 总线：机器人底盘的 4 个麦轮电机需要接收云台上主控板的 CAN 控制信号并反馈自身转速信息，除此之外，主控板还需要通过串口接收底盘微机传来的视觉数据。而由于导电滑环的线路有限，因此需要将这些数据整合一起。故我们在底盘上又额外增加了一块单片机，如图 4-9 所示，其通过串口接收从微机传来的视觉数据，并且外接一块 CAN 总线模块，负责将视觉数据通过 CAN 数据的方式发送给主控板同时与主控板双向传输与底盘电机有关的控制信号。

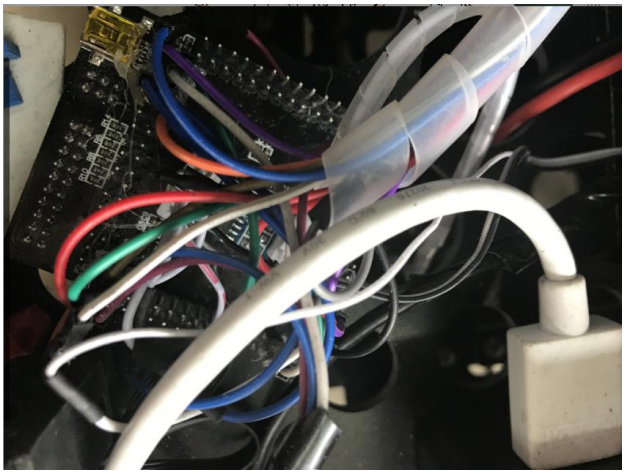


图 4-9 底盘单片机

1 路空余：云台自身的布线要求的宗旨是保证导线连接处稳固，杜绝脱落的可能性并且保持导线紧凑以保证在云台的运动过程中不会与导线干涉。通过机械部分碳板上预留的豁口以及螺孔，我们通过扎带螺钉等工具将导线固定在云台上。对于多根方向一致的导线，我们用绕线管将导线缠绕在一起。

图 4-10 布线图

图 4-11 云台布线

4.4 本章小结

本章主要对电控系统的 PID 控制进行了介绍，如云台的 PID 控制及调试效果图、曲线图，还对发射机构的 PID 控制方法进行了介绍，最后对云台控制系统的相机、导电滑环、单片机、单片机和 USB 之间的线路布置进行了介绍及实物的展示。

第 5 章 视觉系统设计

视觉系统是我主要完成的部分，我将详细介绍此系统的设计以及使用的视觉算法的数学原理和 OpenCV 实现方法。

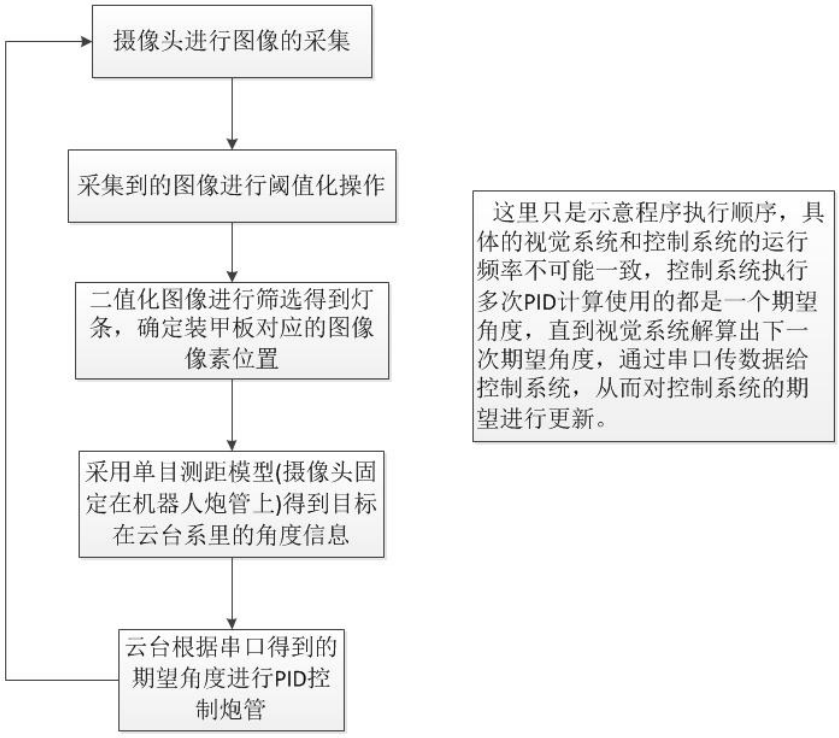


图 5-1 视觉系统流程图

由图 5-1 可以看出，视觉系统主要包括图像获取，图像处理，单目测距和串口通信四个部分。其中，图像获取和串口通信部分使用的都是现成的借口函数，因此不加以具体阐述。

5.1 图像处理

从图 2-1 中可以看出，待识别的物体具有非常明显的颜色特征，即蓝红灯条，因此对于获取的 RGB 图像，我首先进行了颜色通道的分离，得到各个颜色通道的图像。对于获得的图像，如果敌方机器人是蓝色灯条，我则处理蓝色通道图像；若是红色灯条，我则处理红色通道图像。对于这个通道的图像，设定颜色阈值，对所有像素

点与此阈值对比，可以获得一个二值化黑白图像。这个黑白图像是对原始图像的颜色提取后的结果，保存了包括灯条在内的颜色信息（黑色像素组成的矩形）。下一步我要做的就是在此图像上根据灯条的几何信息进行筛选。

灯条的几何属性包括：面积，长宽比，平行度，高度差等。通过设置多个阈值条件，我对黑白图像上的所有黑色像素矩形进行筛选，最终挑选出符合条件的两个矩形，认定他们就是待找的两个灯条。

除此之外，装甲板中间还有非常明显的数字特征。通过数字识别算法，我就可以不光定位一块装甲板，还可以确定是哪辆车的装甲板。然而，通过神经网络组成的数字分类器的计算较慢，即使是 Lenet 这种小型的网络，对于 28×28 像素尺寸大小的数字图像仍需要大约 100ms 的运行时间，对于我们实时性要求高的追踪场合而言，不太适合。因此，我没有考虑数字图像代表的数值信息，而是考虑其颜色信息，作为装甲板灯条筛选的最终条件，即两块灯条中间有较多的白色像素块。

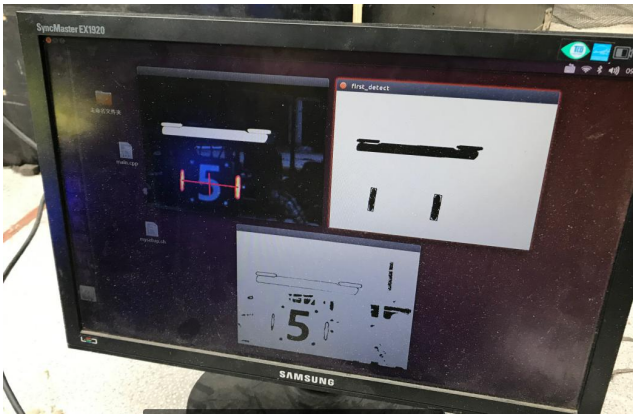


图 5-2 图像处理效果图

从图 5-2 可以看出，装甲板灯条可以很好地筛选出来，实现瞄准功能。我将实现上述功能的 C++代码附在论文最后，可供感兴趣的人士参考。当我获得装甲板的像素信息以后，就可以使用下一节介绍的相机成像模型和 solvePnP 算法实现 2D 像素坐标计算 3D 物理坐标这一测距目标，继而获得角度信息，最终控制炮管实现实时追踪。

5.2 图像测距

由于图像测距算法牵涉较多的相机数学模型知识，因此我将从相机的成像原理开始介绍，逐步过渡到最后的测距算法。

视觉方面的研究起源于对光线的捕捉。光线可能来自于不同的光源(比如电灯或者太阳光)。光线从光源发出，直到碰到物品为止，都在空间中传播。当光线碰到物品后，大部分光线被吸收，同时没有被物品吸收的光线就是我们看见的物品的颜色。这些未被吸收的光线进入我们人的视网膜或者相机的成像仪中，带给人类一个缤纷斑斓的世界。光线从物品开始，直到最终被人眼或者相机捕获的过程对于当下的计算机视觉有着及其重要的意义。

小孔成像模型是一个简单但是非常有用的相机模型。这个模型假想了一个虚拟的墙壁且这个墙壁的中间有一个很小的孔。这个孔只使能刚好穿过小孔的光线传过墙而阻挡其余大部分光线。在本文介绍的方法中，我先阐述一下小孔成像的基本几何计算。然而，在现实世界中，一个小孔并不是一个很好的呈像方法，因为在较短的曝光时间下，并没有足够多的光线能从小孔中穿过形成一个清晰的图像。这就是为什么我们人眼和相机采用视网膜或者镜片聚集光线。这种方法的缺点是不仅仅模型本身比小孔成像复杂，还会因为镜片本身的制造问题引入畸变。

首先，我介绍了一种去除镜片带来的图像畸变的相机标定方法。图像标定的操作对于测距算法非常重要，因为现实世界中的物品在二维成像仪上是由像素位置表示的，任何测距算法都是通过此像素位置进行计算。这意味着这些像素本身精度能极大程度上决定测距算法的精度，只有当物品特征点的像素位置能最大地符合此特征点的三维位置的时候，测距算法才能发挥最大的性能。所以，对于相机平面的像素单位和物理世界的物理单位之间的关系研究是任何三维视图重建工作的重要一环。

图像标定需要表述下相机的成像几何模型和镜片的畸变模型。当得到这个两个模型的数学表达以后，我就可以引入相机的内参数矩阵概念。

我同样会引入单应性变换的概念，它是一个数学上的表示方法，有助于描述相机成像几何模型和镜片畸变以及与之相应的图像纠正。

5.2.1 相机成像模型

我首先探讨最简单的小孔成像模型。如图 5-3 所示的模型，光线只能从小孔进入，投射到后面的成像平面上。因此，成像平面上的投影总是在其焦距位置上（或者说只有焦距位置上的投影是清晰的），同时，投影相对于物品本身大小的比例关系也

仅仅由一个物理量表达，即焦距。在这个理想的小孔成像模型中，从小孔所在的墙壁到成像平安的距离就是焦距的距离，就如下图所示，f表示的是相机的焦距，Z是相机到目标物品的距离，X是目标物品自身的高度，x是目标物品在成像平面上的高度。在这个图中，我可以很简单地得到这个关系式

$$-x/f = X/Z \tag{5-1}$$

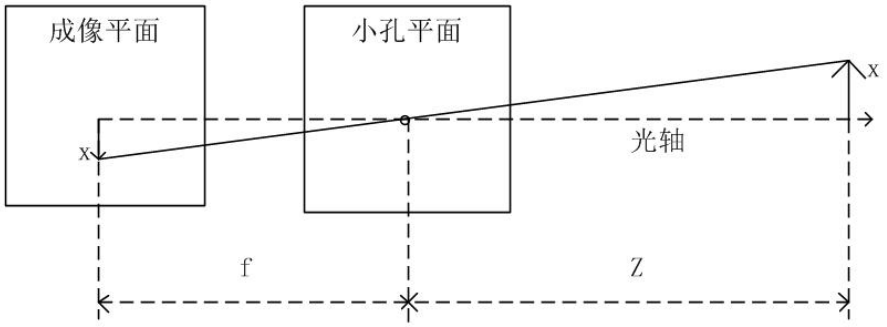


图 5-3 小孔成像模型

我现在对这个小孔成像模型进行一些修改，再保留其物理性质的同时，使得数学表达更为简单。在图 5-4 中，我交换了小孔平面和成像平面的位置。这两种方式主要的区别就是交换过后，物品和成像平面出现在同一侧。原先的小孔在这里被当做成了投影中心。这里由光轴和成像平面所形成的连接点被称为主点(principal point)。在现在这个全新的图 5-4 中的成像平面上，目标物品的图像投影尺寸和图 5-3 中的原先的成像平面上的尺寸完全一致。这里的图像都是由光线和成像平面相交产生的，因此在图 5-3 和图 5-4 中的几何关系都是一致的，且在图 5-4 中，这个几何关系可以更加直观地看出。同时，原先(5-1)式中的负号也可以去掉。

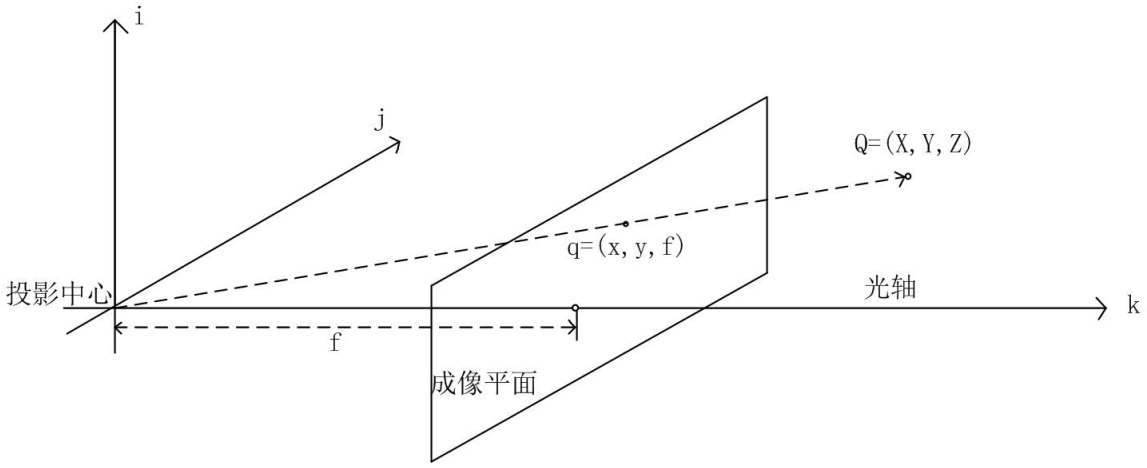


图 5-4 新小孔成像模型

如图5-4所示，一个特征点Q通过小孔成像模型透射到成像平面上得到投影点q，这里的成像平面仅仅是把图5-4中的平面人为地移动到了另一侧。

大部分人可能会认为主点就是相机成像仪的中心，然而这就意味着相机成像仪的安装精度需要达到微米级别的精度。事实上，成像仪的中心往往不在光轴上。所以我们引入了两个新的变量，c_x和c_y，他们表示可能存在的微小偏移量。加上这两个变量后的模型表达式为（假设Q是目标的一个特征点，其坐标是（X，Y，Z），他投影到成像仪上后得到的像素坐标为（x，y），其中

$$x = f_x \frac{X}{Z} + C_x, y = f_y \frac{Y}{Z} + C_y \tag{5-2}$$

注意到这里我引入了两个不同的焦距变量，原因是对于大部分低造价的成像仪而言，每个像素的形状是长方形而不是正方形。焦距f_x，事实上是镜片的物理焦距长度和成像仪每个单独元素的尺寸s_x乘积（这应该是很合理的，因为s_x的单位是像素每毫米，f的单位是毫米，这就代表着f_x的单位是像素）。同理可得f_y和s_y。值得注意的是，s_x和s_y是无法通过任何物理手段直接测量的，同样的，物理焦距长度也是无法测得的。在不拆开相机直接测量其镜片的前提下，我仅仅能通过相机标定算法得到相机的f_x和f_y。

（1）基本的投影几何

投影变换指的是将一系列世界坐标系里的左边点$Q(X,Y,Z)$投影到相机的成像平面上，得到其各自的像素坐标(x,y)。当我们需要处理这样的问题的时候，齐次坐标表示是一个非常好的表达方法。当我们使用齐次坐标表示的时候，一个在n维的投影空间里的坐标点通常可以用$n+1$维来表示(比如xyz可以用$xyzw$表示)。这种表示引入了一种额外的约束，即当两个点的坐标对应项都成比例的时候，这两个点表示的是此投影空间中的同一个点。在本文中，成像平面是投影空间，其有两个维度，所以我会把在成像平面里的点用一个由3个变量组成的向量表示，即$\boldsymbol{Q}=(\boldsymbol{q}_1,\boldsymbol{q}_2,\boldsymbol{q}_3)$。正如上面所说，当两个点的坐标值成比例的时候，他们表示的是同一个点，我们可以通过将前两项除以第三项得到实际成像平面上的像素坐标。这样的表示方法可以让我们将上述提到的4个定义相机的参数(f_x,f_y,c_x,c_y)重新排列到一个3×3的矩阵里，这个矩阵就是相机的内参数矩阵。最后可得，将世界坐标系里的坐标往相机成像平面里投影的方程如下

$$\boldsymbol{q} = \boldsymbol{MQ} \tag{5-3}$$

其中，$\boldsymbol{q}=\begin{bmatrix} x \\ y \\ w \end{bmatrix}, \boldsymbol{M}=\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{Q}=\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$。

我们不难发现，$w=Z$，同时，点 q 用的是齐次坐标表示的，我可以将 $\boldsymbol{q}$ 的前两项除以 $w(Z)$，得到我们在式(5-2)中得到的像素坐标。注意，这里是没有负号的，表明成像平面和目标物体在小孔的同一侧。

在这种理想的小孔成像模型中，我知道了如何将世界坐标中的 3D 坐标投影到相机的成像平面里。然而，需要了解的是，只有很少的光线能从小孔中穿过，因此，事实上基于此的相机成像速度非常慢，因为它需要很多的时间来捕捉足够多的光线以呈现一幅明亮的画面。对于想要快速成像的相机而言，我必须确保它能在很短的时间里，在较大的区域里捕捉到足够的光线，除此之外，它还需要使这些光线发生弯折，从而在成像平面上交汇得到投影点。为了实现上述要求，大部分相机使用的是镜片。镜片可以在一个点上聚集大量的光线从而实现快速成像的目的，随之而来的代价就是引入畸变。

（2）镜片畸变

理论上设计一个不引入任何畸变的镜片是可行的。然而实际上，没有完美的镜片。这个的主要原因是制造误差是无法避免的。从制造上来说，制造球状的镜片比制造一个数学上理想的抛物状的镜片简单许多。同样地，将镜片和成像仪完全对准也是非常难的，这也会引入镜片畸变。下面介绍两种主要的镜片畸变和它们的数学模型。径向畸变是由于镜片的形状引入的，切向畸变是由相机的装配误差引入的。

我首先从径向畸变开始阐述。真实相机的镜片通常会很显著地使成像仪边缘部分的图像发生畸变。最为显著的现象就是“桶形”和“鱼眼”效应。图 5-5 展示了为什么会有径向畸变。对于某些镜片而言，远离、镜片中心的光线会弯折的程度比靠近中心的光线要大。对一种典型的便宜镜片而言，从远离镜片中心的地方捕捉图像容易带来畸变。“桶形”畸变在便宜的网络监控摄像头中非常常见，但是在高级的摄像机中，畸变较少。这主要是由于高级的相机在减少径向畸变上的要求上，对像素坐标进行了修正，从而得到一个理想的投影效果。

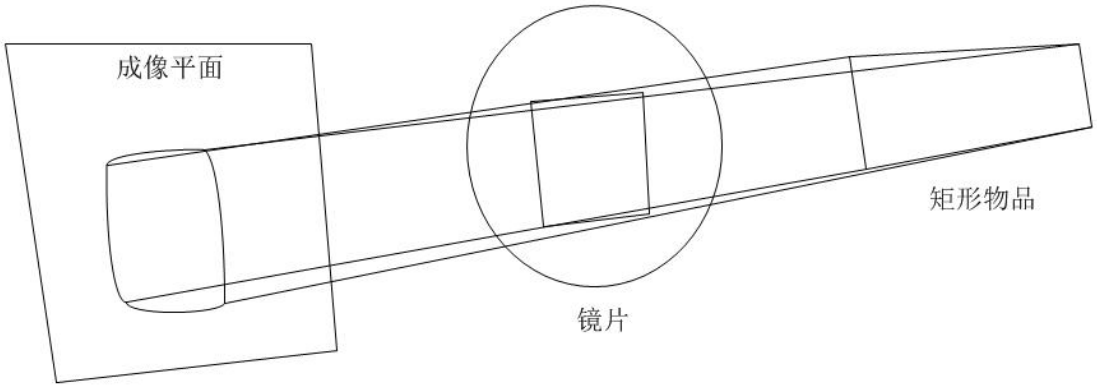


图 5-5 径向畸变示意图

对于径向畸变而言，成像平面中心(光点)处的畸变为 0，且从中心开始，往两边的畸变逐渐变大。事实上，畸变效果不是很大，因此可以通过在 r=0 处的只含前几项的泰勒展开式来表示径向畸变的效果。对于便宜的网络监控相机，我一般会用前两项泰勒展开式，其中，第一项系数被叫做 k_1，第二项系数被叫做 k_2。对于畸变较大的相机，如鱼眼相机，我额外使用第三项，其系数被叫做 k_3。因此，考虑径向畸变以后成像平面上的像素位置可以用如下公式表示

$$x_{\text{corrected}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{5-4}$$

$$y_{\text{corrected}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{5-5}$$

式(4)和式(5)中的(x,y)是成像平面上的初始像素的位置，$(x_{\text{corrected}},y_{\text{corrected}})$是去除径向畸变以后得到的新的像素位置。

第二种影响较大的畸变是切向畸变。这种畸变是由于制造误差引起的。这里的制造误差主要来自于成像平面的 CMOS 芯片所在平面和镜片平面的非平行度。这种制造误差在便宜的相机中比较常见，如图 5-6 所示。

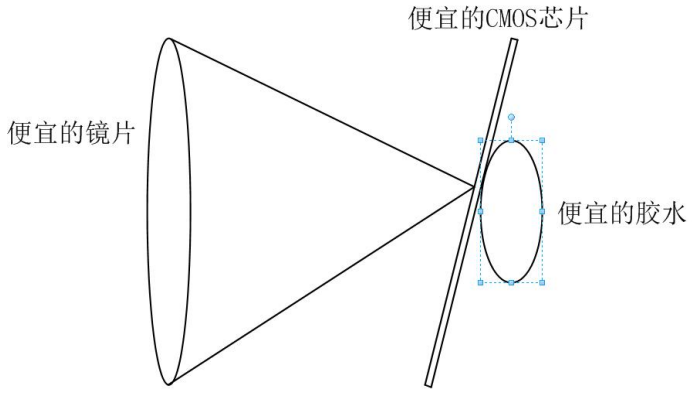


图 5-6 便宜的相机

切向畸变通常由额外的两个参数 p_1 和 p_2 表示，如下面公式所示

$$x_{\text{corrected}} = x + [2p_1xy + p_2(r^2 + 2x^2)] \tag{5-5}$$

$$y_{\text{corrected}} = y + [2p_2xy + p_1(r^2 + 2y^2)] \tag{5-6}$$

由此可知，在标定环节总共有 5 个额外的变量需要计算。因为这 5 个变量在大多数使用相机的场合都会用到，他们经常被放在一起使用。这样就产生了一个 5×1 的矩阵，包含这 5 个元素 k_1，k_2，k_3，p_1，p_2。

5.2.2 相机标定

到这里，我已经基本阐释了相机的内参数矩阵和畸变参数矩阵，这两个矩阵共同构成了相机的标定的数学模型。接下来就是具体阐述如何进行标定。在大部分的标定场合下，标定算法都是通过一个具有很多特征像素点的已知尺寸的物品(如棋盘)和其在相机成像平面上对应的像素之间的关系进行计算的。我将此物品在相机的可见视野里进行小范围的移动或者转动，得到不同的图像，接着计算每个图像在相机系里的旋转平移矩阵。当得到这些矩阵以后，就可以通过我后面会介绍的方法计算得到相机的内参数矩阵。

为了得到相机的内参数矩阵，我需要数次移动旋转物体，所以我首先介绍旋转平移矩阵。

（1）旋转平移矩阵

对于每一个相机拍摄到的包含目标物体的图像，我总可以通过旋转平移矩阵描述该物体在相机坐标系里的位姿信息(pose)，如图 5-7 所示。

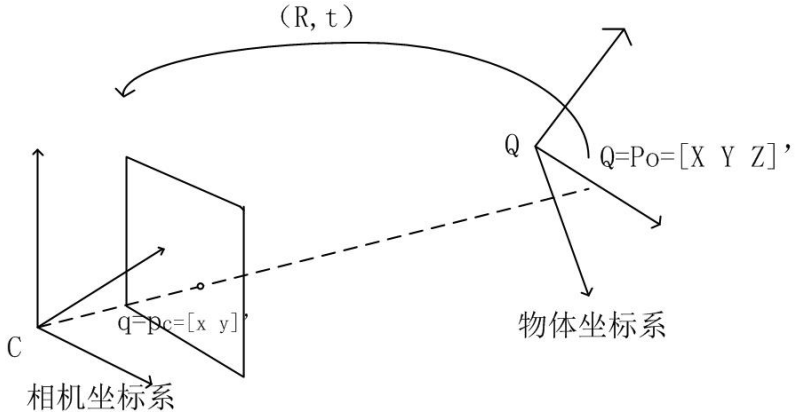


图 5-7 旋转平移矩阵

总的来说，任意维度的旋转可以用原系中的坐标点和一个维度相同的方阵的矩阵乘积来表示。因此，旋转就相当于引入此坐标点在另外一个坐标系中的坐标表示。我将原始坐标系旋转θ度等效于逆时针绕着坐标系原点旋转坐标系中的所有坐标θ度。图 5-8 表示了在一个二维平面内的坐标系旋转。三维空间里的旋转可以被分解成三个二维平面内的旋转，每个旋转中只有两个坐标轴发生旋转变化，而旋转轴保持不变。如果我按照顺序绕 x,y,z 旋转轴旋转，其旋转角度依次是ψ，φ和θ。最终得到的结果就是一个由 3 个分项乘积得到的旋转矩阵 $\boldsymbol{R}$。下面列出的就是这三个分项

$$\boldsymbol{R}_x(\psi)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\psi & \sin\psi\\ 0 & -\sin\psi & \cos\psi\end{bmatrix}$$

$$\boldsymbol{R}_y(\phi)=\begin{bmatrix}\cos\phi & 0 & -\sin\phi\\ 0 & 1 & 0\\ \sin\phi & 0 & \cos\phi\end{bmatrix}$$

$$\boldsymbol{R}_z(\theta)=\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$

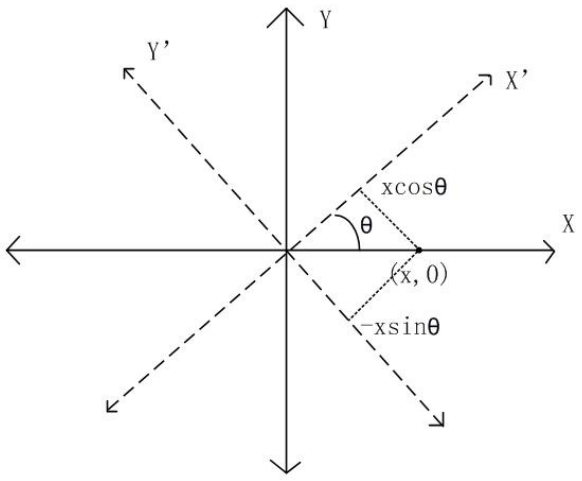


图 5-8 二维平面旋转示意图

观察上面三个分量表达式，每个分量表达式矩阵都是标准正交矩阵，即矩阵自身和矩阵的转置的乘积为单位矩阵。同样地，由线性代数原理可知，多个正交矩阵的乘积仍然是正交矩阵，故此处旋转矩阵 $\boldsymbol{R}$ 的逆等于矩阵 $\boldsymbol{R}$ 的转置。

平移向量是我用来描述两个坐标系之间的距离关系，其中原点固定在第二个坐

标系的原点上。换句话说，平移向量就是第一个坐标系的原点在第二个坐标系里的偏移量。因此，如果我要表示物体上坐标系在相机坐标系中的位置关系，我只需要找到合适的平移向量 $\boldsymbol{T} = origin_{\text{object}} - origin_{\text{camera}}$。最后，我将旋转矩阵和平移向量合在一起考虑(假定在图 5-8 所示的情况下)，可以得到一个在物体系内的点 $\boldsymbol{P}_{\text{o}}$ 在相机坐标系里的投影点 $\boldsymbol{P}_{\text{c}}$,其关系如下

$$\boldsymbol{P}_{\text{c}} = \boldsymbol{R}(\boldsymbol{P}_{\text{o}} - \boldsymbol{T}) \tag{5-7}$$

将我前面得到的相机内参数矩阵(f_x，f_y，c_x，c_y)和式(5-7)结合在一起就可以得到基本的系统模型表达式。当我可以求解出这个表达式后，我就可以求得相机的内参数矩阵和各个图像位置的旋转平移矩阵。

我刚才介绍了三维空间里的旋转表达 $\boldsymbol{R}$ 可以由三个分量的乘积构成，每个分量中包含一个旋转角度。同样地，三维空间里的平移向量也可以由 3 个分量来表示（x，y，z）。因此，我总共有 6 个变量需要求解。除此之外，相机的内参数矩阵中有 4 个变量，故每个图像中总共有 10 个变量需要求解（注意，对于每幅图像而言，内参数矩阵中的 4 个变量是一样的）。在这里，我使用一个平面物体，且其的每一幅图像都可以提供出 8 个约束方程。因为每一幅图像都有各自的旋转平移 6 个变量，因此每一幅图像还有额外的两个约束可以用来求解内参数矩阵的 4 个变量。由此可见，我最少需要两幅不同的图像求解所有未知的变量。

在后续的介绍中，我会更详细地介绍具体的约束形式和计算细节。但是首先，我先具体讨论下标定用的平面物体。由于我使用的 OpenCV 开源计算机视觉库的标准函数使用的是一个 7×6 的棋盘作为标定对象，我将遵循这个标准，以后的内容阐述都是以棋盘标定为基础的。

（2）标定棋盘

从原理上来说，任何合适的特征明显的物体都可以用来作为标定对象。出于计算的方便性，这里选取的是平面物体（相比较 3 维物体，平面物体可以减少一个维度的信息且我很难测量三维物体的特征点在自身坐标系里的坐标），例如棋盘，圆点网格等，如图 5-9 和图 5-10 所示。OpenCV 故选取了棋盘作为标准的标定模型。同时，出于计算的准确性，虽然最少两幅图片就可以计算出全部的变量，但考虑到在每幅图像中像素位置的跳变性（由于一个像素对应的物理尺寸非常小，在物体不动的情况下，像素仍然会出现微幅跳变的可能）。通常，研究人员倾向于采取多幅

图像，然后采用非线性优化的方法得到最优解，比如最小二乘法。下面我将介绍使用棋盘作为标定对象的 OpenCV 标定函数。

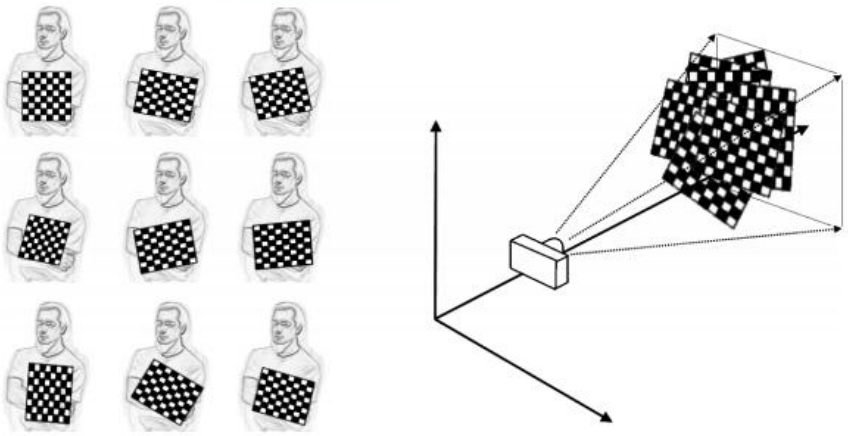


图 5-9 棋盘标定（左侧是改变棋盘获得多个图像的过程）

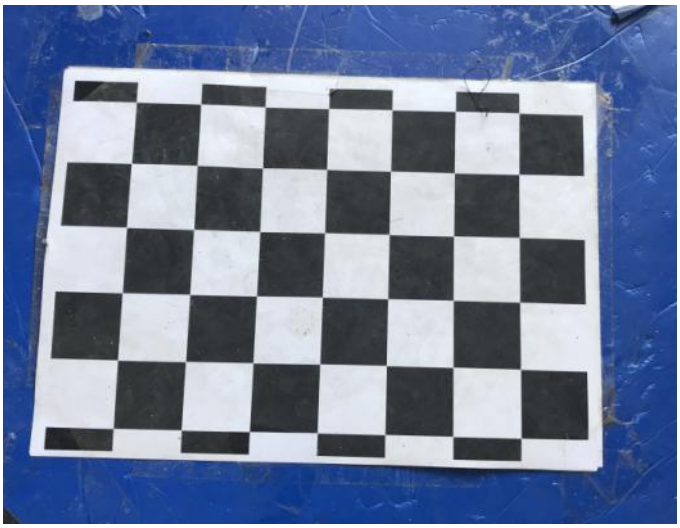


图 5-10 本人使用的棋盘

棋盘是由黑白交替出现的方格组成，这样就确保了在图像获取的过程不会出现误匹配的过程，因为每一方格出现的位置都是事先定好的。

（3）寻找棋盘角点

OpenCV 中定义了寻找棋盘角点的函数 cv::findChessboardCornors()。我们可以使用这个函数获得棋盘角点在成像仪上的像素位置。下面我列出了这个函数原型：

bool cv::findChessboardCornors(//如果检测到角点，则返回 true

cv::InputArray image, //输入的带有棋盘的图像，灰度图像或者彩色图像

```
    cv::size            patternSize ,          //每一行和每一列的角点数构成的尺寸
矩阵
    cv::OutputArray     corners,               //输出的标注好角点的图像
    Int                 flags      =       cv::CALIB_CB_ADAPTIVE_THRESH
                                   |       cv::CALIB_CB_NORMALIZE_IMAGE
);
```

这个函数将一个包含棋盘的单独图像作为第一个参数输入。这个图像必须是一个 8 位的图像。第二个参数表示的是棋盘尺寸大小，即棋盘的每一行每一列有多少个角点，以图 8 中的棋盘为例，此棋盘每一行有 7 个角点，每一列有 6 个角点，因此第二个参数为 cv::Size(7,6)。第三个参数是将原始图像上棋盘角点位置记录后输出，其中记录了每一个角点的具体像素位置。最后第四个参数是一个标志位参数，表示的是是否采用一些额外的过滤算法以找到正确匹配的棋盘角点。我可以使用几下几种的标志位：

1) cv::CALIB_CB_ADAPTIVE_THRESH

cv::findChessboardCorners()函数原先根据输入图像的平均亮度值进行阈值化操作，但是如果这个标志位被设置了，那么函数将采用自适应阈值算法。

2) cv::CALIB_CB_NORMALIZE_IMAGE

如果这个标志位被设置了，这个图像数据在阈值化之前会先进行标准化，其标准化函数使用的是 cv::equalizeHist()。

3) cv:::CALIB_CB_FILTER_QUADS

一旦图像阈值化操作结束以后，算法就会定位棋盘上特征点的像素位置。然而定位是会引入额外误差的，这是因为棋盘上每一个黑色矩形的边被默认是直的，然而在实际上，由于径向畸变和切向畸变的作用，这只是一个真实模型的一个近似。如果这个标志位被设置了，就会产生许多额外的约束条件，以确保错误的数据能够被剔除掉。

4) cv::CALIB_CV_FAST_CHECK

如果这个标志位被设置了，那么就会有一个快速对图像进行扫描的过程。此扫描的目的是判断当前图像内是否有任意角点存在。如果没有角点存在，则这幅图像就被完全略过。当你能完全确定图像中一定存在棋盘的时候，这个标志位是没有意

义的。但是在另一方面，如果角点不存在或者不确定是否有角点存在，那么这个前置扫描过程可以帮助节省很多时间。

如果检测到这个与 size 设定的个数一致的角点数，这个函数的返回值将会被设置为 true，反之则为 false。

（4）棋盘角点的亚像素

cv::findChessboardCorners()只能得到角点的大致像素位置。所以，在函数内部，cv::cornerSubPix()被默认调用了，通过亚像素级别的定位，可以获得更准确的结果。如果还想获得更高的精度，则在 cv::findChessboardCorners()的输出图像上手动再运行一遍 cv::cornerSubPix()函数，现在的迭代终止条件也应该更苛刻一些。

（5）棋盘角点的绘制

当在调试的过程中，如果能将定位得到的角点在最初的图像上显示出来的话，则会很有帮助。因为，这样方便我确定检测到的角点是否匹配预先设置的角点。而这一步操作，OpenCV 也内置了方便调用的函数 cv::drawChessboardCorners()。如果相机图像中包含的角点个数与设置的个数不符，则这个函数会把每一个角点用红色圆圈标记出来；如果个数恰好相等，则这些角点会用不同颜色的圆圈标记出来（每一行的角点用相同的颜色标记）且这些圆圈会被线条连起来。下面我列出函数的原型：

```
void cv::drawChessboardCornors(                    //没有返回值
    cv::InputOutputArray        image,             //输入和输出图像，即原地修改，颜色图像
    cv::size                    patternSize ,      //每一行和每一列的角点数构成的尺寸矩阵
    cv::InputArray              corners,           //indChessboardCorners()中得到的角点
    bool                        patternWasFound
);
```

cv::drawChessboardCorners()函数的第一个参数是输入和输出的图像。由于每个角点会用带有颜色的圆圈标记，所有这个图像必须是三通道的 8 位图像。大多数情况下，这个图像是输入给 cv::findChessboardCorners()函数的图像（但是得确保这个图

像是三通道的 8 位图像）。后面的两个参数，patternSize 和 corners 和 cv::findChessboardCorners()函数中的一致。最后的参数 patternWasFound 返回的值表示整个棋盘都成功地找到与否，这个参数可以传入 cv::findChessboardCorners()函数的返回值。

接下来，我就会介绍当找到棋盘匹配点以后，这些数据能如何通过投影变换求得相机的内参数矩阵和旋转平移矩阵。其中，投影变换所需的参数用一个 3×3 的齐次矩阵表示。

（6）齐次矩阵

在计算机视觉研究领域中，人们通常定义平面坐标齐次变换表示从一个平面投影到另一个坐标系。因此，从一个二维的平面投影到相机的成像平面可以用平面坐标齐次变换表示。这样的话，投影过程的数学模型可以用简单的矩阵相乘表示，这里投影点在自身坐标系下的坐标用 $\boldsymbol{Q}$ 表示，其在成像平面上的像素点用 $\boldsymbol{q}$ 表示

$$\boldsymbol{Q} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \text{和} \quad \boldsymbol{q} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

可以用下式表示投影的过程

$$\boldsymbol{q} = s\boldsymbol{H}\boldsymbol{Q} \tag{5-8}$$

注意在这里，我引入了变量 s，其是一个任意的尺度变量。它的作用仅仅是为了说明单应性矩阵是与尺度变量有关的。这个变量也可以乘到矩阵 $\boldsymbol{H}$ 里，不过传统做法是把它提取出来，这里我按照传统的做法。

通过一些基本的几何知识和矩阵分析理论，我们可以求解式(5-8)，得到旋转平移矩阵。这里最重要的一点是，矩阵 $\boldsymbol{H}$ 是由两部分组成的，它既包含坐标系之间的旋转平移关系又包括由内参数矩阵构成的坐标系和成像平面的映射关系。

旋转平移关系顾名思义是由旋转矩阵和平移向量构成的。因为这里我使用的是齐次坐标，我们可以把这个变量放在一个新的矩阵里如下所示

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix}$$

然后，用 $\boldsymbol{M}$ 表示相机的内参数矩阵。此时我已经知道如何用相机内参数矩阵和相机成像平面的像素点位置建立关系，故坐标 $\boldsymbol{Q}$ 可以可以直接转化到最终的像素点

位置，如下式表示

$$\boldsymbol{q} = s\boldsymbol{MWQ} \tag{5-9}$$

其中，$\boldsymbol{M}$ 的定义为

$$\boldsymbol{M} = \begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix}$$

由于我假设第二个坐标系是在平面内，因此此坐标系可以默认 z=0。我在这里进行了这个操作，其好处是显而易见的：如果我们把旋转矩阵 $\boldsymbol{R}$ 看作是 3 列，其中每一列是一个 3×1 的向量，我只需要任意两个向量就可以求得第三个向量

$$\begin{bmatrix} \mathrm{x} \\ y \\ 1 \end{bmatrix} = s\boldsymbol{M}\begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 & \boldsymbol{t} \end{bmatrix}\begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = s\boldsymbol{M}\begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t} \end{bmatrix}\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{5-10}$$

接着，此时的齐次坐标矩阵 $\boldsymbol{H}$ 就可以写成 $\boldsymbol{H} = s\boldsymbol{M}\begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t} \end{bmatrix}$。这里的 $\boldsymbol{H}$ 矩阵是一个 3×3 的矩阵。

OpenCV 中棋盘标定使用的这个方法。它使用棋盘在不同角度下的多张图片来计算每张图片的旋转平移矩阵以及所有图片共享的相机内参数矩阵。正如我前面所说，对于每张图片，旋转和平移各自可以由三个变量表示。相对的，每张图片可以提供 8 个约束方程，即平面坐标系上的矩形可以投影到成像平面上得到一个四边形，此四边形有四个像素点(x,y)，每个 x 或者 y 都是一个约束方程。一个图片引入 6 个额外的变量以及 8 个约束方程，因此在可以得到任意多的图像的前提下，无论内参数矩阵里面包括了多少未知的变量，我们都是可以一一求解得到的。

单应性矩阵 $\boldsymbol{H}$ 表达了物体坐标系上物体坐标点和其在相机成像平面上的像素点之间的关系，如

$$\boldsymbol{p}_{\mathrm{dst}} \equiv \begin{bmatrix} x_{\mathrm{dst}} \\ y_{\mathrm{dst}} \\ 1 \end{bmatrix} = \boldsymbol{H}\boldsymbol{p}_{\mathrm{src}} \tag{5-11}$$

值得一提的是，我无需已知相机的内参数矩阵或者图像的旋转平移矩阵，就可以计算式(5-11)中的单应性矩阵 $\boldsymbol{H}$。其唯一的条件就是有足够多的（至少四对）匹配点对。事实上，OpenCV 就是通过复数多张图片来计算单应性矩阵的。OpenCV 同样

提供了封装好的函数以供计算单应性矩阵 cv::findHomography()。这个函数的输入参数就是匹配点对，返回计算好的单应性矩阵。我们需要知道的是每张图片至少需要提供四个坐标点，但是如果我使用更多的坐标点且它们都有正确的匹配关系，可以有效减少图像噪声或者误匹配带来的误差。

```
cv::Mat cv::findHomography(                          //返回单应性矩阵 H
    cv::InputArray    srcPoints,                     //输入的点集(2D)
    cv::InputArray    dstPoints ,                    //输入的点集(3D)
    cv::int           methods          =0,           //0，CV::RANSAC，CV::LMEDS 等
    Double            ransacReprojThreshold = 3,     //最大重投影误差
    cv::OutputArray mask = cv::noArray()             //掩膜，只是用掩膜矩阵非 0 部分
    );
```

输入参数 srcPoints 和 dstPoints 分别包含了物体特征点在其自身平面和相机成像平面中的坐标。这些坐标都应该是二维的坐标，因为我这里用的是棋盘标定，去除了棋盘平面的高度信息 z。

输入参数 methods 决定了程序使用哪种算法计算单应性矩阵。如果左侧赋予的值是预设值 0，那么程序就是考虑所有的输入点集，最终得到的结果是使这些点集重投影误差最小的单应性矩阵 $\boldsymbol{H}$。重投影误差是所有物体自身坐标系内的坐标经过变换投影到成像平面后的像素与相机检测到的像素的欧氏距离的平方和。

最方便的快速计算的算法无法解决数据异常点(outliers)的存在，即如果存在误差很大的误匹配，此算法会考虑到这个误匹配带来的影响（此误差的梯度极大），改变已有的接近正确的解去适应这个误匹配导致对于所有正确的匹配而言，总的误差和反而增加了。在相机标定的过程中，这种异常值是很容易产生的，因此得到的结果往往会很偏离正确的解。OpenCV 因此提供了三种比较鲁棒的求解算法，这些算法虽然会耗更多的计算资源，不过可以取得更好的结果。

第一个是 cv::RANSAC，当这个标志位被设置以后，程序将会使用 RANSAC 算法（random sampling with consensus）。程序会随机挑选这些点集中的子集，并且针对每个子集计算了单应性矩阵。接着，程序会保留与这些子集计算出来的单应性矩阵较为匹配的匹配坐标点而抛弃误差较大的那些。这个算法会计算很多组随机子集，并且最终使用包含正确数据最大的子集的计算结果。这个算法在实际中能非常有效

地去除异常数据从而找到正确的解。

第二个是 cv::LMEDS，但这个标志位被设置后，程序会使用 LMEDS 算法（least median of squares）。正如这个算法的名称所示，其本质原理是最小化中位数误差，而不是像预设的算法一样减少平均数误差。

这个算法的优点是它无需任何额外的信息或者参数，缺点是它只有在异常值比较少的场合下才能取得较好的结果。与之相反的是，RANSAC 算法无论异常值的数量多少都可以起到作用返回比较好的结果。但是 RANSAC 算法需要我们输入可以被认为是有效匹配点的最大重投影误差（单位是像素）。这个输入变量就是 ransacReprojThreshold。如果使用别的算法，这个变量可以忽略。

最后一个是 cv::PROSAC，当这个标志位被设置后，程序会使用 RHO 算法，其原理是带有权重的 RANSAC 算法，可以在有很多异常值的场合下较快运行。

最后一个输入变量，掩膜，只有在上述三种之一的标志位被设置的时候才会 使用。如果这个掩膜被设置成功，cv::findHomography()函数只会对不在掩膜上的点集进行单应性矩阵的计算。

函数的返回值是一个 3×3 的矩阵。当我们默认为 H_{33}=1 的时候，单应性矩阵 $\boldsymbol{H}$ 中只有 8 个自由变量。如果对矩阵 $\boldsymbol{H}$ 进行尺度上的变化的话，我们可以得到第九个可变的变量，但是通常来说，更倾向于将尺度变量单独提取出来用 s 表示，然后把整个 $\boldsymbol{H}$ 矩阵乘上这个尺度变量。

（7）相机标定

在介绍了前述这么多内容以后，我们终于来到相机标定的部分了。在这个环节，我将具体阐述如何求得相机的内参数矩阵和畸变矩阵以及介绍 OpenCV 提供的计算函数 cv::calibrateCamera()。我使用这个函数纠正图像中相机畸变引起的误差，对于所有像素点都进行纠正的操作，我就可以获得一幅没有畸变的图像。我首先说明下需要多少张包含棋盘的图像来完成此操作，然后我们将介绍这个算法背后的数学原理。

多少个棋盘角点对应多少变量呢？首先，让我回顾下目前存在的未知变量，即我希望通过标定求解出的变量。在上述部分里，我定义了与相机内参数矩阵有关的 4 个变量（f_x，f_y，c_x，c_y）和与畸变有关的 5 个变量（可以更多），即径向畸变的 k_1，k_2，k_3 和切向畸变的 p_1，p_2。内参数矩阵是将特征点从物体特征系上线性投影变换到

图像的成像平面里。我将内参数矩阵好外参数矩阵（旋转平移矩阵）合在一起就可以把任意的三维空间中的点投影到成像平面上，得到像素坐标。

畸变是在二维的成像平面上产生的，它描述了原本的每个像素点坐标(x_{screen}，y_{screen})和在畸变作用下得到新的像素点坐标之间的关系。从原理上来看，三个匹配的坐标点总共能提供 6 个约束条件，而我们一共只有五个未知变量，因此一张包含棋盘的图像就足够求解出这五个变量。

然而，由于内参数矩阵和外参数矩阵之间互相影响，一张图片是不足以求解的。我们需要记住，每个外参数矩阵包括三个旋转变量和三个平移变量，因此每张图像总共有 6 个外参数矩阵变量需要求解。总的来说，每张图像都有总共十个内外矩阵变量需要求解。

假设一共有 K 张图像，每张图像上包括 N 个角点。那么我现在需要确定 K 和 N 需要满足的关系以求解出全部的未知变量。

总共 K 张图片能提供 $2\times N\times K$ 个约束（每个角点都有 x，y 坐标）。

暂时忽略畸变系数，我们现在总共有四个内参数矩阵变量（对于所有的图像都是一样的）和 6 个外参数变量（每张图片都不一样）。

求解这个方程组需要满足的条件是：$2\times N\times K>=6\times K+4$，变换后得到，$(N-3)\times K>=2$。

如果仅仅从这个数学关系来看，如果 $N=5$，那么 $K=1$ 就能满足条件，即我只需要一张图片。但是这里需要注意，对于使用棋盘的相机标定而言，K 是必须大于一的。这个原因也很明显，通过式(5-10)可以看出，棋盘标定下每张图片的单应性矩阵最多只有 8 个变量。这 8 个变量可以通过四个匹配点对求解出来。从几何上解释，这是因为平面投影只需要四个坐标点就可以完全投影这个平面。四个点能够在四个方向上延伸一个正方形，使其变成一个任意的四边形。所以无论一张图片上我们可以得到多少个角点，4 个角点就能包含所有的图像信息了。对于每一张棋盘图像而言，只有四个角点对于我求解方程有一定的帮助，因此我可以把原先的条件表达式改写成：$(4-3)\times K>1$，这就意味着 K 必须大于 1。同时，这也意味着，两张包含 3×3 棋盘（只考虑内部的角点，因此有 $2\times 2=4$ 个角点）信息的图像是求解这个方程组的最小要求。当考虑到噪声和数值计算的稳定性要求，我总会提供超过两张图片，且每张图片也总会包括更多的角点（在本文中是 7×6 的棋盘），然后使用优化方法求

得最优的未知变量数值。在实际操作中，为了得到准确的结果，我至少会使用 10 张图片，其中每张图片都会包括 7×8 甚至更大的棋盘。

理论求解的最小要求和工程上实际操作的情况巨大差异是由于内参数矩阵对于误差的高度敏感性，及其微小的像素变化都会对内参数变量的计算结果产生较大的影响。

（8）标定背后的数学原理

这一节会具体介绍标定的计算过程。事实上，有很多的方法可以求解这些未知变量。正如前面所说，OpenCV 采用的是棋盘标定的方法，其内部求解焦距长度和光轴偏移量采用的是 zhang 的方法[18]，求解畸变参数采用的是 Brown 的方法[19]。

首先，当我求解内外矩阵参数的时候，先假设相机不受畸变影响。这样，对于采集到的每一幅图像，我可以求得其单应性矩阵（回顾一下，这个矩阵是将坐标从物体系里投影到相机的成像平面系里）。我将这个矩阵改写成列向量的形式，$\boldsymbol{H}=\begin{bmatrix}\boldsymbol{h}_1 & \boldsymbol{h}_2 & \boldsymbol{h}_3\end{bmatrix}$，这里的每一个 $\boldsymbol{h}$ 都是一个 3×1 的列向量。接着，我把这个 $\boldsymbol{H}$ 矩阵重新表达成相机内参数矩阵和旋转矩阵中前两项以及平移向量构成的矩阵的乘积

$$\boldsymbol{H}=\begin{bmatrix}\boldsymbol{h}_1 & \boldsymbol{h}_2 & \boldsymbol{h}_3\end{bmatrix}=s\boldsymbol{M}\begin{bmatrix}\boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t}\end{bmatrix}$$

将矩阵的等式拆分成向量的等式，我可以得到

$$\boldsymbol{h}_1=s\boldsymbol{M}\boldsymbol{r}_1 \text{或者} \boldsymbol{r}_1=\lambda\boldsymbol{M}^{-1}\boldsymbol{h}_1$$

$$\boldsymbol{h}_2=s\boldsymbol{M}\boldsymbol{r}_2 \text{或者} \boldsymbol{r}_2=\lambda\boldsymbol{M}^{-1}\boldsymbol{h}_2$$

$$\boldsymbol{h}_3=s\boldsymbol{M}\boldsymbol{t} \text{或者} \boldsymbol{t}=\lambda\boldsymbol{M}^{-1}\boldsymbol{h}_3$$

这里的 $\lambda=\frac{1}{\mathrm{s}}$。

旋转矩阵中的任何一列都是与其他列是正交的。这里既然尺度变量 s 已经提取出来了，$\boldsymbol{r}_1$ 和 $\boldsymbol{r}_2$ 我可以看做是标准正交的，其会带来两个计算结果：旋转向量的内积为 0；两个向量的模长相等。因此，可以得到

$$\boldsymbol{r}_1^{\mathrm{T}}\boldsymbol{r}_2^{\mathrm{T}}=0 \tag{5-12}$$

对于任意的向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 而言，向量乘积的转置总是等于向量转置的乘积，只是乘积的先后顺序需要发生改变，即 $(\boldsymbol{ab})^{\mathrm{T}}=\boldsymbol{b}^{\mathrm{T}}\boldsymbol{a}^{\mathrm{T}}$，这里我将上述式子的中两个向量用 $\boldsymbol{h}$ 向量和 $\boldsymbol{M}$ 矩阵表示，得到第一个约束条件

$$\boldsymbol{h}_1^{\mathrm{T}}\boldsymbol{M}^{-\mathrm{T}}\boldsymbol{M}^{-1}\boldsymbol{h}_2=0 \tag{5-13}$$

这里 $\boldsymbol{M}^{-\mathrm{T}}$ 是 $\left(\boldsymbol{M}^{-1}\right)^{\mathrm{T}}$ 的缩略形式。我同样知道向量的模长计算公式

$$\left\|\boldsymbol{r}_1\right\|=\left\|\boldsymbol{r}_2\right\| \text{ 即 } \boldsymbol{r}_1^{\mathrm{T}}\boldsymbol{r}_1=\boldsymbol{r}_2^{\mathrm{T}}\boldsymbol{r}_2$$

这里，我将 $\boldsymbol{r}_1$ 和 $\boldsymbol{r}_2$ 用表达式替换，得到第二个约束条件

$$\boldsymbol{h}_1^{\mathrm{T}}\boldsymbol{M}^{-\mathrm{T}}\boldsymbol{M}^{-1}\boldsymbol{h}_1=\boldsymbol{h}_2^{\mathrm{T}}\boldsymbol{M}^{-\mathrm{T}}\boldsymbol{M}^{-1}\boldsymbol{h}_2 \tag{5-14}$$

为了简化表达式，我定义 $\boldsymbol{B}=\boldsymbol{M}^{-\mathrm{T}}*\boldsymbol{M}^{-1}$。我将 $\boldsymbol{B}$ 展开，得到下式

$$\boldsymbol{B}=\boldsymbol{M}^{-\mathrm{T}}\boldsymbol{M}^{-1}\equiv\begin{bmatrix}B_{11} & B_{12} & B_{13}\\ B_{21} & B_{22} & B_{23}\\ B_{31} & B_{32} & B_{33}\end{bmatrix}$$

注意矩阵 $\boldsymbol{M}$ 的形式我是已知的（$\boldsymbol{M}$ 是相机的内参数矩阵），因此我也可以将 $\boldsymbol{B}$ 矩阵用内参数矩阵中的未知变量表示，因此 $\boldsymbol{B}$ 矩阵有解析表示式，即

$$\boldsymbol{B}=\begin{bmatrix}\dfrac{1}{f_x^2} & 0 & -\dfrac{c_x}{f_x^2}\\ 0 & \dfrac{1}{f_y^2} & -\dfrac{c_y}{f_y^2}\\ -\dfrac{c_x}{f_x^2} & -\dfrac{c_y}{f_y^2} & \left(\dfrac{c_x}{f_x^2}+\dfrac{c_y}{f_y^2}+1\right)\end{bmatrix}$$

我可以往两个约束条件都代入 $\boldsymbol{B}$，这样我可以得到一致的形式：$\boldsymbol{h}_i^{\mathrm{T}}\boldsymbol{B}\boldsymbol{h}_j$，又因为 $\boldsymbol{B}$ 是一个对称矩阵，我可以把这个乘积表达式改写成向量内积的形式。我将 $\boldsymbol{B}$ 中必要的元素重新转移到一个 6×1 的向量 $\boldsymbol{b}$ 中，我们可以得到

$$\boldsymbol{h}_i^{\mathrm{T}}\boldsymbol{B}\boldsymbol{h}_j=\boldsymbol{v}_{ij}^{\mathrm{T}}\boldsymbol{b}=$$
$$\begin{bmatrix}h_{i,1}h_{j,1} & (h_{i,1}h_{j,2}+h_{i,2}h_{j,1}) & h_{i,2}h_{j,2} & (h_{i,3}h_{j,1}+h_{i,1}h_{j,3}) & (h_{i,3}h_{j,2}+h_{i,2}h_{j,3}) & h_{i,3}h_{j,3}\end{bmatrix}\times \tag{5-15}$$
$$\begin{bmatrix}B_{11}\\ B_{12}\\ B_{22}\\ B_{13}\\ B_{23}\\ B_{33}\end{bmatrix}$$

我将 $\boldsymbol{v}_{i,j}^{\mathrm{T}}$ 代入到上面的两个约束中去，这样可以得到改写后的式子

$$\begin{bmatrix}\boldsymbol{v}_{12}^{\mathrm{T}}\\ (\boldsymbol{v}_{11}-\boldsymbol{v}_{22})^{\mathrm{T}}\end{bmatrix}\boldsymbol{b}=0 \tag{5-16}$$

如果我使用 K 张有棋盘的图片，那么我们可以把这 K 张图片提供的 $2\times K$ 个约束条件用一个矩阵乘积表示，即

$$\boldsymbol{V}\boldsymbol{b}=0 \tag{5-17}$$

这里的 $\boldsymbol{V}$ 是一个 $2\times K\times 6$ 大小的矩阵，如果这里的 K 大于等于 2，则这个矩阵乘积表达式是可以求出 $\boldsymbol{b}=\begin{bmatrix}B_{11} & B_{12} & B_{22} & B_{13} & B_{23} & B_{33}\end{bmatrix}$ 的解析解的。继而，我可以直接根据 $\boldsymbol{b}$ 向量的解得出相机的内参数矩阵。

$$f_x=\sqrt{\lambda \big/ B_{11}}$$

$$f_y=\sqrt{\frac{\lambda B_{11}}{B_{11}B_{22}-B_{12}^2}}$$

$$c_x=\frac{B_{13}f_x^2}{\lambda}$$

$$c_{\mathrm{y}}=\frac{B_{12}B_{13}-B_{11}B_{23}}{B_{11}B_{22}-B_{12}^2}$$

在这个四个表达式中，λ 是一个定义的中间变量，其具体的数学含义如下

$$\lambda=B_{33}-\frac{(\boldsymbol{B}_{13}^2+c_y(\boldsymbol{B}_{12}\boldsymbol{B}_{13}-\boldsymbol{B}_{11}-\boldsymbol{B}_{23}))}{\boldsymbol{B}_{11}}$$

这里的每张图片的相机的外参数（旋转平移矩阵）可以根据单应性变换的表达式中的几列相乘得到。

$$\boldsymbol{r}_1=\lambda\boldsymbol{M}^{-1}\boldsymbol{h}_1$$

$$\boldsymbol{r}_2=\lambda\boldsymbol{M}^{-1}\boldsymbol{h}_2$$

$$\boldsymbol{r}_3=\boldsymbol{r}_1\boldsymbol{r}_2$$

同理可得，$\boldsymbol{t}=\lambda\boldsymbol{M}^{-1}\boldsymbol{h}_3$。

这里，需要注意的一个技巧是，前面提出的尺度变量 s（$\lambda=1/s$）是根据 $\boldsymbol{r}_1,\boldsymbol{r}_2,\boldsymbol{r}_3$ 组成的矩阵的标准正交性得到的（即这里的 λ 的值是将每一列向量的模长标准化为 1 而确定的），即 $\lambda=1\big/\left\|\boldsymbol{M}^{-1}\boldsymbol{h}_1\right\|$ 或者 $\lambda=1\big/\left\|M^{-1}\boldsymbol{h}_2\right\|$。注意，这里需要额外说明下，在大多数的实际场合下，当我输入实际的数据以后，将得到的 $\boldsymbol{r}_1,\boldsymbol{r}_2,\boldsymbol{r}_3$ 向量组合在一个矩阵中后，这个矩阵并不是严格意义上的标准正交矩阵，即 $\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R}=\boldsymbol{R}\boldsymbol{R}^{\mathrm{T}}=\boldsymbol{I}_3$ 的关系不会完全成立，而是较为符合，存在一定误差。

为了应对这个问题，我常会对 $\boldsymbol{R}$ 矩阵进行奇异值分解（SVD）。SVD 是将一个矩阵拆分成两个标准正交矩阵的乘积 $\boldsymbol{U}$ 和 $\boldsymbol{V}$，其中间还有一个矩阵 $\boldsymbol{D}$，对角线上是

尺度值。这让我们可以将矩阵 $\boldsymbol{R}$ 分解成 $\boldsymbol{R}=\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathrm{T}}$，由于矩阵 R 本身是标准正交，因此矩阵 $\boldsymbol{D}$ 必须是单位矩阵 $\boldsymbol{I}_3$，故 $\boldsymbol{R}=\boldsymbol{U}\boldsymbol{I}_3\boldsymbol{V}^{\mathrm{T}}$。我可以“强制”我们计算出来的矩阵 $\boldsymbol{R}$ 变成标准正交矩阵，通过 $\boldsymbol{R}$ 矩阵的奇异值分解得到 $\boldsymbol{U}$ 和 $\boldsymbol{V}$ 矩阵，再将这两个矩阵和单位矩阵乘在一起得到新的标准正交矩阵，也就是新的旋转矩阵 $\boldsymbol{R}$。

到现在为止，我还没有对镜片畸变进行处理。我使用计算出来的内参数矩阵 $\boldsymbol{M}$（这里我们假设的是畸变参数都为 0，没有畸变影响作为初始解，然后优化求解这个方程组）。

我在成像平面求得的像素坐标点由于畸变作用的影响其实是不正确的。这里我们用(x_p,y_p)表示小孔成像模型下的像素坐标，(x_d,y_d)表示考虑了畸变作用影响以后的像素坐标，然后有这个关系式（小孔成像的模型）

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix}=\begin{bmatrix} f_x\dfrac{X_W}{Z_W}+c_x \\ f_y\dfrac{Y_W}{Z_W}+c_y \end{bmatrix} \tag{5-18}$$

考虑了畸变作用以后的得到的像素坐标表达式

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix}=(1+k_1r^2+k_2r^4+k_3r^6)\begin{bmatrix} x_d \\ y_d \end{bmatrix}+\begin{bmatrix} 2p_1x_dy_d+p_2(r^2+2x_d^2) \\ p_1(r^2+2y_d^2)+2p_2x_dy_d \end{bmatrix} \tag{5-19}$$

这里的求解思路与不考虑畸变作用的时候相同，即我们至少需要与未知变量相同的约束条件，这里我多了五个额外的畸变变量，因此至少还需要五个约束条件。由于一张图像的单应性矩阵 $\boldsymbol{H}$ 只能提供两个约束条件，因此，此时需要更多的图像来获得这额外的约束条件。这些工作 OpenCV 都提供了一个封装好的函数 cv::calibrateCamera()帮助我们完成！

（9）标定函数

当我得到数张图像上的角点坐标后，我可以使用 cv::caliberateCamera()函数来完成标定工作。一般地，经过图像标定以后，我可以得到相机的内参数矩阵和畸变参数矩阵，以及各个输入图像的旋转平移矩阵。这里的前二项构成了相机的内部参数，后二项表明了物体在相机系里的位置信息。畸变系数（k_1,k_2,k_3,p_1,p_2 和任意高阶的 k）是我先前介绍的径向和切向畸变方程里，在我想要去除畸变作用的时候会用这些参数（如相机畸变比较严重的情况下，实际形状比较笔直的物体会变弯曲）。相机的内参数矩阵是最重要的计算结果，它能将三维坐标系里的物体投影到相机的成像平

面上。我同样可以用相机的内参数矩阵做逆向求解，即将成像平面上的物体，但是往往只能得到三维坐标下的一条线（缺少深度信息），不过就像上面所言，如果我们知道物体角点的实际位置，我是可以求解得到其唯一的三维信息的，这个算法就是大名鼎鼎的 PnP 算法。

下面介绍这个函数接口：

```
Double cv::caliberateCamera(
        cv::InputArrayOfArrays      objectPoints,      //K 个向量（每个向量 N 个坐标点，三维）
        cv::InputArrayOfArrays      imagePoints,       //K 个向量（每个向量 N 个坐标点，图像）
        cv::Size                    imageSize,         //输入图像的像素尺寸
        cv::InputArrayOfArrays      cameraMatrix       //得到的相机内参数矩阵
        cv::InputArrayOfArrays      distCoeffs,        //畸变参数系数向量
        cv::OutputArrayOfArrays     rvecs,             //由 K 个旋转向量组成的旋转向量
        cv::OutputArrayOfArrays     tvecs,             //由 K 个平移向量组成的向量
        Int                         flags      =   0,  //算法选择标志位
        cv::TermCriteria            criteria   =   cv::TermCriteria(
        cv::TermCriteira::COUN|
        cv::TermCriteria::EPS,30,DBL_EPSILON)   //30 对应的是迭代次数
        //DBL_EPSILON 是重投影误差
        );
```

当我使用这个函数的时候，我需要输入很多的参数。不过，大部分参数的物理意义在前面的论述过程中其实已经表明清楚了。下面，我详细介绍下函数中出现的每个参数:

第一个参数是 objectPoints。它是一个由多个向量总成的向量，每个向量记录了三维坐标系里的角点的坐标，在我们的标定例子中，就是棋盘上每个角点在棋盘坐标系里的坐标，注意只有二个方向上的坐标，z 设置为 0。

第二个参数是 imagePoints。它同第一个参数类似，只不过是对应的角点在图像

平面里的二维坐标。

第三个参数是 imageSize。它表明的是图像的尺寸大小。

第四和第五个参数是相机的内参数矩阵和畸变参数矩阵。它们也是这个函数求得的计算结果。内参数矩阵是一个 3×3 的矩阵，而后面的畸变参数矩阵可能包含 4，5 或者 8 个元素，这取决于使用者的设定。如果畸变参数矩阵只返回 4 个元素，那么这四个元素是 k_1，k_2，p_1，p_2。

畸变参数矩阵在使用鱼眼镜头的时候比较有用，因为此时的畸变比较严重。当使用者设置了 cv::CALIB_RATIONAL_MODEL 后，才会返回 8 个元素，这一般只用在非常特殊的镜头下。这里需要记住的是，所需要的图片的数量会随着待求解的变量个数而急剧增加。

第 6 和第 7 个参数是向量组成的向量，分别对应旋转和平移信息。此处的旋转信息是按照罗德里格斯向量形式存储的，即用一个包含 3 个变量的旋转向量表示旋转。这里的向量个数与图片张数一致。

在大多数情况下，我会有多于变量个数的约束方程。因此，这时候会需要引入优化算法。有时候如果希望一次性计算出所有的最优解，则会产生不准确的解或者解不收敛,特别是初始的猜测解与实际存在的真实解相差较大。所以，逐步找到最好的初始猜测解是一种比较好的求解策略。因此，我在实际求解的过程中，总会让几个变量保持不变，然后在此基础上，求解出剩余变量的优化解，接着在下一次优化过程中，使上一次保持不变的发生改变，然后上一次发生改变的保持不变。最终，当我认为所有的参数值都较为准确的情况下，我可以一次性求解出前面所有相机图像的位姿信息。

每次调用此函数的时候都可以通过 flags 标志位选择合适的控制策略。下述的几种控制策略可以通过布尔条件判断组合起来使用。

1) cv::CALIB_USE_INTRINSIC_GUESS

通常来说，相机的内参数矩阵是由标定函数直接计算得到的，这个计算过程中除了图像平面以后是不需要任何额外信息的。对于光轴补偿 c_x 和 c_y 而言，它们的初始值默认设置成图像尺寸的一半，即默认光轴对应图像中心。如果这个标志位被设置了，那么就表示相机内参数矩阵在输入的时候存储的值用来当做优化计算的初始解。

2) cv::CALIB_FIX_PRINCIPAL_POINT

这个标志位可以与前一个标志位组合使用。当这个标志位被设置了，如果前一个标志位没有被设置，那么光轴就被固定在图像的中心了；如果前一个标志位被设置了，那么光轴就被固定在提供的初始解的位置处。

3) cv::CALIB_FIX_ASPECT_TATIO

如果这个标志位被设置了，那么优化算法只会对 f_x 和 f_y 一起进行求解，同时保持这两个变量的比例保持不变。

4) cv::CALIB_FIX_FOCAL_LENGTH

如果这个标志位被设置了，那么 f_x 和 f_y 的值在优化计算中保持为输入的相机内参数矩阵中的猜测解。

5) cv::CALIB_FIX_K1,cv::CALIB_FIX_K2,...,cv::CALIB_FIX_K6

这几个标志位表明在计算过程中是否需要把这些畸变系数设定为定值。

6) cv::CALIB_ZERO_TANGENT_DIST

这个标志位对于标定高端相机来说起到较大作用，因为高端相机由于具备一定的制造精度，因此制造误差引起的切向畸变可以当做不存在。此外，对于在 0 附近的变量进行优化会引入带有噪声的错误数据，影响算法的稳定性。

7) cv::CALIB_RATIONAL_MODEL

如果设置了这个标志位，那么函数会计算 k_4，k_5，k_6 这三个畸变参数。如果不设置的话，即使输入的畸变向量中包含 8 个元素，函数也只会计算前 3 个 k。

最后的一个变量表示的是这个函数的迭代终止准则。与绝大多数优化算法相同，这里的终止准则可以是迭代次数也可以重投影误差的数值大小。重投影误差指的是三维坐标投影到成像平面上得到的像素坐标与通过原始图像检测到的像素坐标之间的平方差值。

到这里位置，我已经详细介绍了相机的基本数学模型和单目视觉使用的基本模型以及详细阐述了相机标定的方法和背后的数学原理。下面我将介绍如何通过内参数矩阵来计算相机的外参数矩阵，也就是待求的旋转平移向量。

5.2.3 单目测距

在很多情况下，我们已经有了相机的内参数矩阵，而只需要求解目标物品在图

像坐标系里的旋转平移向量。这种情况明显与相机标定模型不同，但是却在实际应用中也有很大的价值。特别是在自动追踪的领域，如果我已知待追踪的物品的实际尺寸，比如说敌方机器人的装甲板形状尺寸，我需要知道也只是此装甲板在我们图像坐标系里的三维信息（因为摄像头固定在云台上，当我们知道装甲板在图像坐标系中的信息以后，也就可以通过坐标系移动的方法计算出在云台系里的位置，从而驱动云台电机实现自动追踪）。通常来说，这个可以叫做 PnP 问题。OpenCV 针对这个也提供了封装好的函数如下：

```
bool cv::solvePnP(
    cv::InputArray        objectPoints,                    //物体坐标(物体坐标系)
    cv::InputArray        imagePoints,                     //像素坐标(成像平面)
    cv::InputArray        cameraMatrix,                    //3*3 的相机内参数矩阵
    cv::InputArray        disCoeffs,                       //畸变系数组成的向量
    cv::OutputArray       rvec,                            //得到的旋转向量
    cv::OutputArra        tvec,                            //得到的平移向量
    bool                  useExtrinsicGuess = false,       //是否设置初始估计解
    int                   flags             = cv::INTERTIVE
);
```

solvePnP 函数里面的参数的物理意义和前面相机标定中的意义大部分是完全一致的，但是还有两点不同。首先这个函数输入的物体坐标和像素坐标是在单一图像中的（注意，变量的类型是 InputArray 而不是 InputArrayOfArrays）；其次，这里的相机内参数矩阵和畸变参数矩阵是作为输入参数而不是计算出来的值。这里的旋转变量都是罗德里格斯变换下的，即只需要 3 个变量就可以表示三维空间里的旋转（旋转向量的方向表示的是旋转轴，模长表示旋转的角度）。这个旋转向量可以通过逆变换得到 3*3 的旋转矩阵，此矩阵是我们比较熟悉的旋转表达形式。平移变量是相机坐标系和物体坐标系之间的三维偏移量。

下一个变量 useExtrinsicGuess 如果被设置为 true 的话，就表明输入的旋转平移向量作为计算的初始解。程序的预设值是 false。

最后一个 flag 变量可以被设置成三个标志位中的一个，分别对应一种计算方法。这三个标志位是 CV::ITERATIVE,CV::P3P 和 CV::EPNP。如果使用的是

CV::ITERATIVE，程序会使用 LM(Levenberg-Marquardt)优化算法减少重投影误差。如果使用的是 CV::P3P，程序使用的是 Gao[20]等人提出的算法。在这种情况下，我们只应该输入 4 个物体坐标点和 4 个匹配的图像坐标点，只有当计算结果准确的时候才会返回计算结果。最后，如果使用的是 CV::EPNP，程序会使用 Moreno-Noguer07[21]提出的算法。后面两个方法不会迭代求解，因此会比第一种方法快不少。

cv::solvePnPRansac 函数计算旋转平移矩阵

cv::solvePnP 算法的一个主要缺陷是它对于异常值很敏感。在相机标定的情况下，这并不是一个问题，因为棋盘本身的角点是比较易于找到的，且具有非常明显的顺序关系，因此匹配关系比较可靠。不同的是，在定位的情况下，真实世界中的角点提取是有一定难度的，且误匹配是普遍存在的情况，因此如果在这种充满异常值的情况下直接使用 PnP 算法，会带来非常严重的计算错误。而这种情况下，RANSAC 算法能有效地排除异常值(outliers)，保留正确值(inliers)。下面是基于 RANSAC 算法的 PnP 改进算法的函数接口：

```
bool cv::solvePnPRansac(
    cv::InputArray      objectPoints,                   //物体坐标(物体坐标系)
    cv::InputArray      imagePoints,                    //像素坐标(成像平面)
    cv::InputArray      cameraMatrix,                   //3*3 的相机内参数矩阵
    cv::InputArray      distCoeffs,                     //畸变系数组成的向量
    cv::OutputArray     rvec,                           //得到的旋转向量
    cv::OutputArray     tvec,                           //得到的平移向量
    bool                useExtrinsicGuess   = false     //是否设置初始解
    int                 IterationCount      = 100       //RANSAC 的迭代次数
    float               reprojectionError   = 8.0,      //RANSAC 算法的最大容错
值
    int                 minInliersCount     = 100,      //RANSAC 算法的终止个
数
    cv::OutputArray   inliers               = cv::noArray(), //输出的正确数据
    int                 flags               = cv::ITERATIVE //同 cv::solvePnP()一样
    );
```

solvePnP 算法中包含的输入输出参数在 solvePnPRansac 算法也都存在。除此之外，solvePnPRansac 算法还有一些新的参数，这些参数与其中的 Ransac 部分有关。其中，iterationsCount 参数与 Ransac 最大迭代次数有关，reprojectionError 与可将坐标数据归为正常值的最大允许重投影误差。minInliersCount 这个参数的名字其实与其功能有点不太匹配，它的功能是如果在某次样本采样的过程中，Ransac 算法找到的一组样本数据中被计算后归为正确值的个数超过 minInliersCount，则此次采样过程立马结束，然后将此 minInliersCount（个数，例如 100）个数据加入到此次采样对应的正确数据集中，计算并记录此集合的误差总和，然后直接进行下一次采样。这个参数若设置得当能对算法性能起到很大帮助，若设置得不好（比如过低或者过高）可能会造成负面效果。最后的 inliers 参数是一个输出参数，会标记所有被认为正确样本集的坐标数据。

到这里为止，所有在实际中使用的单目测距的理论部分就介绍完毕了，剩下的部分就是在实际机器人上编写代码的时候，需要不停地根据物理环境因素修改各个视觉检测算法中的阈值,就如我在上一节图像处理中所说。

5.3 弹道补偿模型

通过上述章节中阐述的图像获取，图像处理和单目测距算法三个部分，我可以得到目标装甲板在云台系里的三维坐标。然而，由于子弹离开炮管是一个自由落体运动而不是直线运动，其在达到装甲板之前会有一段下坠。因此，根据弹道飞行轨迹对子弹的射击角度加以补偿是获得较高命中准确率的必要保证。

5.3.1 子弹弹道模型

我假设子弹从离开摩擦轮到最终击打到装甲板的这段过程中，唯一受到的外力就是重力，即忽略空气阻力和子弹在炮管中的任何碰撞。因此，子弹弹道模型可以如图 5-11 所示。

首先，我解释下图 5-11 中所表示的弹道模型。正如前文所说，我使用的是单目测距模型，且摄像头是固结在炮管上的，因此，这里的坐标系的 z 轴方向是从炮管延伸出来的，x 轴和 y 轴表示其余两个正交方向。

为了简化模型，我假设装甲板的中心在 x 轴方向上的偏移量为 0,，故此时计算

出来的 yaw 轴角度也是 0（yaw 轴角度与 pitch 轴的补偿角度没有关系）。从图 5-10 中，可以看出子弹在离开炮管以后经历了自由落体的运动，所以我不能直接让炮管转到θ处，而是需要加上弹道补偿角度θ'。接下来，我需要做的就是根据子弹下坠的高度与炮管 pitch 角度θ之间的关系求解出射击需要的弹道补偿角度θ'。我假设子弹离开炮管的初速度是恒定的，为 v，单位是 m/s。

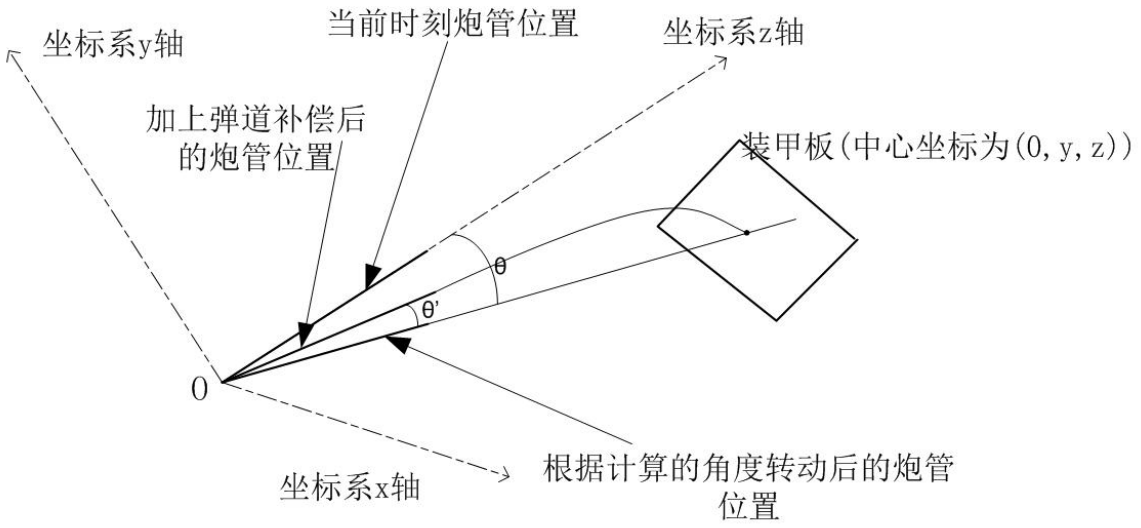


图 5-11 子弹弹道示意图

首先，在当前时刻的炮管位置处，我通过单目测距算法获得了装甲板中心坐标(0,y,z)，因此可以求得不考虑子弹下坠的炮管转动角度θ，即

$$\theta = \arctan(\frac{y}{z}) \tag{5-20}$$

注意这里的装甲板中心在初始坐标系的下方，因此 y 应该是一个负数，计算得到的θ也是一个负数。从图中可以看出，新的坐标系是通过顺时针旋转θ后得到的，故θ为负是正确的。

5.3.2 弹道补偿模型

我在新的炮管位置θ处，考虑由于重力引起的子弹下坠。我需要知道此时装甲板中心在新的炮管坐标系中的 z 轴距离。从前面介绍的旋转平移矩阵内容可知，两个原点重合的坐标系之间的转换只需要乘上一个旋转矩阵即可，故

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{5-21}$$

此时，我可以得到新炮管坐标系下的 z 轴距离 z'=-sin(θ)y+cos(θ)z。接下来，我根据得到的 z'计算子弹弹道补偿角度θ'。这里我又做了一个假设，即忽略炮管与地面水平系之间的夹角，认为重力的方向是垂直于炮管而不是永远保持竖直向下。这个假设在本文所描述的情景下是很有意义的，原因如下：

在实际测试中，炮管多是处于较小仰角的状态。此时重力的实际方向（竖直向下）与我假设的方法（垂直于炮管）之间的夹角较小，不会产生很大误差。

如果重力方向竖直向下，我在计算子弹弹道补偿的时候，需要知道炮管方向和地平面之间的夹角，然而对于视觉系统而言，炮管对地角度并不是能直接获取到的信息。因此，若想要得到此信息，需要电控系统通过串口往视觉系统反馈电机编码器的绝对值。出于系统的稳定性和通信数据的正确性上而言，我希望视觉系统只向电控系统发送数据而不接收数据，故我做出如此假设。

基于这个假设，我可以快速列出与子弹弹道补偿角度θ'有关的方程

$$\begin{cases} v\cos(\theta')t = z' \\ v\sin(\theta') - 0.5gt = 0 \end{cases} \tag{5-22}$$

求解上式，就可以得到子弹补偿角度θ'。然后，我就可以对单目测距算法求得的炮管角度进行修正，即炮管转动角度θ加上子弹补偿角度θ'。

5.4 本章小结

本章将视觉系统分为了图像获取，图像处理，单目测距和串口通信四个部分。其中，通过对要识别的目标进行分析确定了图像处理的方法并获得了较好的结果，对图像测距的相机成像模型的原理进行了介绍，并对相机标定及单目测距进行了原理分析以及算法的设计，最后通过对子弹弹道模型的分析设计了弹道补偿的模型及算法。

结　论

本论文阐述了如何在机电计一体化系统中进行设计的方法。在复杂的集成系统中，我们不能仅仅从单一系统的角度出发，而是需要在设计某一系统的时候，考虑到其对其他系统可能会产生的影响，如设计机械系统时，我们可以预料到摄像头安装位置会对图像的采集产生影响，从而影响测距质量；除此之外，摄像头的安装姿态也会对坐标系转换产生影响，因此最理想的安装位置就是与炮管平行安装，减少不必要的坐标转换。因此，在设计复杂集成的系统时候，很大程度上需要一个对多个系统都有较深认识的设计人员，能从宏观上布局整个系统的设计，将其拆分成数个子系统，并对每个子系统设置设计要求，交由各自设计人员开展具体设计。本文的主要亮点是基于单目视觉算法的测距和追踪系统设计。单目视觉由于缺少深度信息，在绝大多数的情况下不能实现测距的任务，然而在某些特定的场合下，PnP 算法可以通过 3D2D 匹配点对求解出位姿信息。对于求解出来的位姿信息，我将其传输到下位控制系统，从而驱动云台实现实时追踪。

然而如今的系统虽然实现自动追踪瞄准的功能，但是在运动目标的射击准确上仍有欠缺。其原因主要是弹丸从离开炮管到命中装甲板需要一定的时间，而运动目标在这段时间里已经移动了一些距离。由于装甲板尺寸比较小，因此弹丸有很大可能偏离装甲板一些距离，导致命中率降低。因此，今后的工作应该集中在如何根据视觉系统得到的转角信息和云台电机的转速信息，对运动目标进行预测判断，将此补偿值加到电机的转动角度命令上，以提高对运动目标的击打命中率。

参考文献

[1] 张建民. 机电一体化系统设计(第四版)[M]. 北京: 高等教育出版社, 2014: 121-136.

[2] 郭润梅. 机电一体化技术在机器人领域中的应用研究[J]. 世界有色金属, 2018(24): 121-124.

[3] 王一鸣. 建设有国际竞争力的现代产业体系[N]. 学习时报, 2019-03-04(002).

[4] 丁蕾. 智慧城市——西安智能公交站牌视觉应用研究[J]. 电脑知识与技术, 2017, 13(22): 172-173.

[5] 张泽鹏. 基于大数据技术的交通视频监控分析[J]. 科技传播, 2019, 11(04): 177-178.

[6] 张帆. 车载雷达检测图像自动识别追踪[J]. 物探化探计算技术, 2018, 40(04): 487-494.

[7] 蔡英凤, 王海, 张为公. 基于视频的城市快速路交通流检测及车辆行为监控[J]. 东南大学学报, 2011, 27(02): 164-168.

[8] Collins R, Lipton A, Kanade T, et al. A System for Video Surveillance and Monitoring[R]. VSAM Final Report. 2003:23-25..

[9] Collins R, Lipton A, Fujiyoshi H, et al. Algorithms for Cooperative Multisensor Surveillance[J]. Proceedings of the IEEE, 2001, 89(10): 1456-1477.

[10] Coifman B, Beymer D, Mclauchlan P, et al. A Real-time Computer Vision System for Vehicle Tracking and Traffic Surveillance[J]. 1'ansportation Research Part C, 1998, 6(4): 271-288.

[11] Tai J, Tsang S, Lin C, et al. Real-time Image Tracking for Automatic Traffic Monitoring and Enforcement Application[J]. Image and Vision Computing,2004, 22(6): 485-501.

[12] Haag M, Nagel H. Tracking of Complex Driving Maneuvers in Traffic Image Sequences[J]. Image and Vision Computing, 1998, 16(8): 517-527.

[13] Pai C, Tyan H, Liang Y, et al. Pedestrian Detection and Tracking at Crossroads[J]. Pattern Recognition, 2004, 37(5): 1025-1034.

[14] Masoud O, Papanikolopoulos N P. A novel Method for Tracking and Counting Pedestrians in Real-time Using a Single Camera[J]. IEEE Transactions on Vehicular Technology, 2001, 50(5): 1267-1278.

[15] 肖大伟, 翟军勇. 轮式移动机器人单目视觉的目标测距方法[J]. 计算机工程, 2017, 43(04): 287-291.

[16] 康新晨, 杨卫, 邓立齐. 小型可移动平台双目定位方法研究[J]. 电视技术, 2018, 42(07): 29-33.

[17] 宋晓伟, 樊战亭, 田锐.直流电动机 PID 转速控制系统设计[J]. 科技创新与应用, 2018(13): 53-54.

[18] Zhang, Z. A flexible New Technique for Camera Calibration[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2000 (22): 1330–1334.

[19] Brown. Close-range Camera Calibration[J]. Photogrammetric Engineering.1971 (37): 316-320.

[20] Gao, Xiao S, Hou X R, et al. Complete Solution Classification for The Perspective Three Point Problem[J]. IEEE Transactions Pattern Analysis and Machine Intelligence. 2003 (25): 930–943.

[21] Moreno-Noguer F, Lepetit, V, Fua P. Accurate Non-Iterative O(n) Solution to the PnP Problem[C]. IEEE 11th International Conference on Computer Vision. 2007: 1-8.

致　谢

衷心感谢导师史小华老师对本人的精心指导。他的言传身教将使我终生受益。

感谢 RM 俱乐部全体人员的热情帮助和支持！

感谢在学习过程中，与我一起讨论过问题的许多参加 RM 比赛的外校同学！

附录 1（程序代码）

附录部分主要包括了相机标定后得到的相机内参数数据，相机驱动的 C++程序代码，视觉系统的 C++程序代码，makefile 链接文件和 Linux 开机自启动的脚本文件。

intrinsics.xml(相机内参数数据)

```
<?xml version="1.0"?>
<opencv_storage><image_width>640</image_width>
<image_height>480</image_height>
+<camera_matrix type_id="opencv-matrix">
-<distortion_coefficients type_id="opencv-matrix">
<rows>1</rows>
<cols>5</cols>
<dt>d</dt>
<data>-1.5865658095660354e-01 1.6731059482252764e+00 0. 0.
-4.0518047753260284e+01</data>
</distortion_coefficients>
</opencv_storage>
```

RM_capture.h(相机读取函数头文件)

```
//
// Created by hanyunhai on 9/21/18.
//
#ifndef OPENCV3_RMCAPTURE_H
#define OPENCV3_RMCAPTURE_H
#pragma once
#include "opencv2/core.hpp"

class RMVideoCapture {
public:
    RMVideoCapture(const char * device, int size_buffer = 1);    //摄像头的 id
    ~RMVideoCapture();
    bool startStream();
    bool closeStream();
    bool setBlueBalance(int val);
    bool setExposureTime(bool auto_exp, int t);
    //功能：一个视频有很多属性，比如：帧率、总帧数、尺寸、格式等，VideoCapture
的 get 方法可以获取这些属性。
    bool setVideoFormat(int width, int height, bool mjpg = 1);
    bool changeVideoFormat(int width, int height, bool mjpg = 1);
    bool getVideoSize(int & width, int & height);

    bool setVideoFPS(int fps);          //设置帧率      ?
```

```
        bool setBufferSize(int bsize);     //设置文件流的缓冲区大小
        void restartCapture();                  //重新开始
        int getFrameCount(){
            return cur_frame;                   //得到总帧数
        }
        void info();
        RMVideoCapture& operator >> (cv::Mat & image);     //读取视频文件或者捕获数据从解码和返回刚刚捕获的帧
    public:
        int camnum;     //相机编号
    private:
        void cvtRaw2Mat(const void * data, cv::Mat & image);
        bool refreshVideoFormat();
        bool initMMap();
        int xioctl(int fd, int request, void *arg);

    private:
        struct MapBuffer {
            void * ptr;
            unsigned int size;
        };
        unsigned int capture_width;
        unsigned int capture_height;
        unsigned int format;
        int fd;
        unsigned int buffer_size;
        unsigned int buffer_idx;
        unsigned int cur_frame;
        MapBuffer * mb;
        const char * video_path;
    };
    #endif //OPENCV3_RMCAPTURE_H
```

RM_capture.cpp(相机读取函数库文件)

```
#include "RMCapture.h"
#include "linux/videodev2.h"
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>

RMVideoCapture::RMVideoCapture(const char * device, int size_buffer) :
```

```
video_path(device) {
        fd = open(device, O_RDWR);
        buffer_size = size_buffer;
        buffer_idx = 0;
        cur_frame = 0;
        capture_width = 0;
        capture_height = 0;
        mb = new MapBuffer[buffer_size];
    }

    void RMVideoCapture::restartCapture(){
        close(fd);
        fd = open(video_path, O_RDWR);
        buffer_idx = 0;
        cur_frame = 0;
    }

    RMVideoCapture::~RMVideoCapture(){
        close(fd);
        delete [] mb;
    }

    void RMVideoCapture::cvtRaw2Mat(const void * data, cv::Mat & image){
        if (format == V4L2_PIX_FMT_MJPEG){
            cv::Mat src(capture_height, capture_width, CV_8UC3, (void*) data);
            image = cv::imdecode(src, 1);       //解码
        }
        else if(format == V4L2_PIX_FMT_YUYV){
            cv::Mat yuyv(capture_height, capture_width, CV_8UC2, (void*) data);
            cv::cvtColor(yuyv, image, CV_YUV2BGR_YUYV);
        }
    }

    RMVideoCapture & RMVideoCapture::operator >> (cv::Mat & image) {
    //    std::cout << "current buffr idx: " << buffr_idx << std::endl;
        struct v4l2_buffer bufferinfo = {0};
        bufferinfo.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        bufferinfo.memory = V4L2_MEMORY_MMAP;
        bufferinfo.index = buffer_idx;
        if(ioctl(fd, VIDIOC_DQBUF, &bufferinfo) < 0){
            perror("VIDIOC_DQBUF Error");
            exit(1);
        }

        //std::cout << "raw data size: " << bufferinfo.bytesused << std::endl;
        cvtRaw2Mat(mb[buffer_idx].ptr, image);
```

```
        //memset(&bufferinfo, 0, sizeof(bufferinfo));
        bufferinfo.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        bufferinfo.memory = V4L2_MEMORY_MMAP;
        bufferinfo.index = buffer_idx;

        // Queue the next one.
        if(ioctl(fd, VIDIOC_QBUF, &bufferinfo) < 0){
                perror("VIDIOC_DQBUF Error");
                exit(1);
        }
        ++buffer_idx;
        buffer_idx = buffer_idx >= buffer_size ? buffer_idx - buffer_size : buffer_idx;
        ++cur_frame;
        return *this;
}

bool RMVideoCapture::initMMap(){
        struct v4l2_requestbuffers bufrequest = {0};
        bufrequest.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        bufrequest.memory = V4L2_MEMORY_MMAP;
        bufrequest.count = buffer_size;

        if(ioctl(fd, VIDIOC_REQBUFS, &bufrequest) < 0){
                perror("VIDIOC_REQBUFS");
                return false;
        }

        for(int i = 0; i < buffer_size; ++i){
                struct v4l2_buffer bufferinfo = {0};
                bufferinfo.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                bufferinfo.memory = V4L2_MEMORY_MMAP;
                bufferinfo.index = i; /* Queueing buffer index 0. */

                // Put the buffer in the incoming queue.
                if(ioctl(fd, VIDIOC_QUERYBUF, &bufferinfo) < 0){
                        perror("VIDIOC_QUERYBUF");
                        return false;
                }

                mb[i].ptr = mmap(
                                NULL,
                                bufferinfo.length,
                                PROT_READ | PROT_WRITE,
                                MAP_SHARED,
                                fd,
                                bufferinfo.m.offset);
                mb[i].size = bufferinfo.length;
```

```
            if(mb[i].ptr == MAP_FAILED){
                perror("MAP_FAILED");
                return false;
            }
            memset(mb[i].ptr, 0, bufferinfo.length);

            // Put the buffer in the incoming queue.
            memset(&bufferinfo, 0, sizeof(bufferinfo));
            bufferinfo.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
            bufferinfo.memory = V4L2_MEMORY_MMAP;
            bufferinfo.index = i;
            if(ioctl(fd, VIDIOC_QBUF, &bufferinfo) < 0){
                perror("VIDIOC_QBUF");
                return false;
            }
        }
        return true;
}

bool RMVideoCapture::startStream(){
        cur_frame = 0;
        refreshVideoFormat();
        if(initMMap() == false)
            return false;

        __u32 type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if(ioctl(fd, VIDIOC_STREAMON, &type) < 0){
            perror("VIDIOC_STREAMON");
            return false;
        }
        return true;
}

bool RMVideoCapture::closeStream(){
        cur_frame = 0;
        buffer_idx = 0;
        __u32 type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if(ioctl(fd, VIDIOC_STREAMOFF, &type) < 0){
            perror("VIDIOC_STREAMOFF");
            return false;
        }
        for(int i = 0; i < buffer_size; ++i){
            munmap(mb[i].ptr, mb[i].size);
        }
        return true;
}
```

```
bool RMVideoCapture::setExposureTime(bool auto_exp, int t){
	if (auto_exp){
		struct v4l2_control control_s;
		control_s.id = V4L2_CID_EXPOSURE_AUTO;
		control_s.value = V4L2_EXPOSURE_AUTO;
		if( xioctl(fd, VIDIOC_S_CTRL, &control_s) < 0){
			printf("Set Auto Exposure error\n");
			return false;
		}
	}
	else {
		struct v4l2_control control_s;
		control_s.id = V4L2_CID_EXPOSURE_AUTO;
		control_s.value = V4L2_EXPOSURE_MANUAL;
		//if( xioctl(fd, VIDIOC_S_CTRL, &control_s) < 0){
		//	printf("Close MANUAL Exposure error\n");
		//	return false;
		//}

		control_s.id = V4L2_CID_EXPOSURE_ABSOLUTE;
		control_s.value = t;
		//if( xioctl(fd, VIDIOC_S_CTRL, &control_s) < 0){
		//	printf("Set Exposure Time error\n");
		//	return false;
		//}
	}
	return true;
}

bool RMVideoCapture::setBlueBalance(int val){
	struct v4l2_control ctrl;
	ctrl.id = V4L2_CID_AUTO_WHITE_BALANCE;
	ctrl.value = V4L2_WHITE_BALANCE_MANUAL;
	int ret = xioctl(fd,VIDIOC_S_CTRL,&ctrl);
	if(ret < 0){
		printf("Error on read blue balance!\r\n");
		return false;
	}
	ctrl.id = V4L2_CID_GAMMA;
	ctrl.value = val;
	ret = xioctl(fd,VIDIOC_S_CTRL,&ctrl);
	if(ret < 0){
		printf("Error on set blue balance!\r\n");
		return false;
	}
	printf("blue balance val:%d\r\n",ctrl.value);
```

```
}

bool RMVideoCapture::changeVideoFormat(int width, int height, bool mjpg){
    closeStream();
    restartCapture();
    setVideoFormat(width, height, mjpg);
    startStream();
    return true;
}

bool RMVideoCapture::setVideoFormat(int width, int height, bool mjpg){
    if (capture_width == width && capture_height == height)
        return true;
    capture_width = width;
    capture_height = height;
    cur_frame = 0;
    struct v4l2_format fmt = {0};
    fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    fmt.fmt.pix.width = width;
    fmt.fmt.pix.height = height;
    if (mjpg == true)
        fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
    else
        fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
    fmt.fmt.pix.field = V4L2_FIELD_ANY;

    if (-1 == xioctl(fd, VIDIOC_S_FMT, &fmt)){
        printf("Setting Pixel Format\n");
        return false;
    }
    return true;
}

bool RMVideoCapture::refreshVideoFormat(){
    struct v4l2_format fmt = {0};
    fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    if (-1 == xioctl(fd, VIDIOC_G_FMT, &fmt)) {
        perror("Querying Pixel Format\n");
        return false;
    }
    capture_width = fmt.fmt.pix.width;
    capture_height = fmt.fmt.pix.height;
    format = fmt.fmt.pix.pixelformat;
    return true;
}
```

```
bool RMVideoCapture::getVideoSize(int & width, int & height){
    if (capture_width == 0 || capture_height == 0){
        if (refreshVideoFormat() == false)
            return false;
    }
    width = capture_width;
    height = capture_height;
    return true;
}

bool RMVideoCapture::setVideoFPS(int fps){
    struct v4l2_streamparm stream_param = {0};
    stream_param.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    stream_param.parm.capture.timeperframe.denominator = fps;
    stream_param.parm.capture.timeperframe.numerator = 1;

    if (-1 == xioctl(fd, VIDIOC_S_PARM, &stream_param)){
        printf("Setting Frame Rate\n");
        return false;
    }
    return true;
}

bool RMVideoCapture::setBufferSize(int bsize){
    if (buffer_size != bsize){
        buffer_size = bsize;
        delete [] mb;
        mb = new MapBuffer[buffer_size];
    }
}

void RMVideoCapture::info(){
    struct v4l2_capability caps = {};
    if (-1 == xioctl(fd, VIDIOC_QUERYCAP, &caps)) {
        perror("Querying Capabilities\n");
        return;
    }

    printf( "Driver Caps:\n"
                "    Driver: \"%s\"\n"
                "    Card: \"%s\"\n"
                "    Bus: \"%s\"\n"
                "    Version: %d.%d\n"
                "    Capabilities: %08x\n",
                caps.driver,
                caps.card,
                caps.bus_info,
```

```
                (caps.version>>16)&&0xff,
                (caps.version>>24)&&0xff,
                caps.capabilities);
        camnum = caps.bus_info[17] -48;

        struct v4l2_cropcap cropcap = {0};
        cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if (-1 == xioctl (fd, VIDIOC_CROPCAP, &cropcap))    {
            perror("Querying Cropping Capabilities\n");
            return;
        }

        printf( "Camera Cropping:\n"
                "    Bounds: %dx%d+%d+%d\n"
                "    Default: %dx%d+%d+%d\n"
                "    Aspect: %d/%d\n",
                cropcap.bounds.width,    cropcap.bounds.height,    cropcap.bounds.left,
cropcap.bounds.top,
                cropcap.defrect.width,    cropcap.defrect.height,    cropcap.defrect.left,
cropcap.defrect.top,
                cropcap.pixelaspect.numerator, cropcap.pixelaspect.denominator);

        struct v4l2_fmtdesc fmtdesc = {0};
        fmtdesc.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        char fourcc[5] = {0};
        char c, e;
        printf("    FMT : CE Desc\n--------------------\n");
        while (0 == xioctl(fd, VIDIOC_ENUM_FMT, &fmtdesc)) {
            strncpy(fourcc, (char *)&fmtdesc.pixelformat, 4);
            c = fmtdesc.flags & 1? 'C' : ' ';
            e = fmtdesc.flags & 2? 'E' : ' ';
            printf("    %s: %c%c %s\n", fourcc, c, e, fmtdesc.description);
            fmtdesc.index++;
        }

        struct v4l2_format fmt = {0};
        fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if (-1 == xioctl(fd, VIDIOC_G_FMT, &fmt)) {
            perror("Querying Pixel Format\n");
            return;
        }
        strncpy(fourcc, (char *)&fmt.fmt.pix.pixelformat, 4);
        printf( "Selected Camera Mode:\n"
                "    Width: %d\n"
                "    Height: %d\n"
                "    PixFmt: %s\n"
                "    Field: %d\n",
```

```
                fmt.fmt.pix.width,
                fmt.fmt.pix.height,
                fourcc,
                fmt.fmt.pix.field);

        struct v4l2_streamparm streamparm = {0};
        streamparm.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if (-1 == ioctl(fd, VIDIOC_G_PARM, &streamparm)) {
            perror("Querying Frame Rate\n");
            return;
        }
        printf( "Frame Rate:    %f\n====================\n",
                (float)streamparm.parm.capture.timeperframe.denominator /
                (float)streamparm.parm.capture.timeperframe.numerator);
}

int RMVideoCapture::xioctl(int fd, int request, void *arg){
        int r;
        do r = ioctl (fd, request, arg);
        while (-1 == r);
        return r;
}
```

main.cpp(视觉主要程序)

```
#include "CameraApi.h" //相机 SDK 的 API 头文件
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdio.h>
#include <stdlib.h>
#include<iostream>
#include<opencv2/opencv.hpp>
#include <time.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <string.h>
#include<math.h>
#include <vector>
#include <algorithm>
#include <numeric>
#include "RMCapture.h"     //这里是既包括工业相机的自瞄图像读取又包括底盘大符双目相机读取，两种不同的读取方式
#define pi 3.141592654
using namespace std;
using namespace cv;
```

```
unsigned char                * g_pRgbBuffer;        //处理后数据缓存区
//********************
Mat diff[3];
Mat armor;
Mat thres_one,thres_two,thres_three,thres_four;
RotatedRect rect;
vector<RotatedRect> first_rect;
vector<Rect> boundingrect;
vector<vector<Point> > beacons_vertex;
vector<int> motion_prediction;
vector<int> motion_yaw;
int motion_ten_frames=0;
int pitch_comp_last=-1;
double yaw_last=-1.;
double pitch_last=-1.;
int error_times=0;
//the main paremeters       and right now the exposure time is 6(sunshine    150 30)
int beacon_threshold=150;
int number_threshold=50;
int pixel_x_diff=600;
int pixel_number=150;
int x_sum=-1;
int yaw_int_last=-1;
float x_last=-1;
//单目测距需要
vector<Point3f> objP;
vector<Point3f> objP1;
vector<Point2f> object_cornors;
Mat r2 = Mat(1, 3, CV_64FC1);
Mat r3 = Mat(1, 3, CV_64FC1);
//**********
vector<vector<int> > scores_group;
static bool RotateRectSort(const RotatedRect &a1, const RotatedRect &a2) {
    return a1.center.x < a2.center.x;
}
static bool contourheight(const Point& n1, const Point& n2) {
    return n1.y<n2.y;
}
static bool contourwidth(const Point& n1,const Point& n2){
    return n1.x<n2.x;
}
static bool contourvectex(const Point& n1,const Point& n2)
{
    if(n1.y!=n2.y)
        return n1.y<n2.y;
    else
        return n1.x<=n2.x;
```

```
    }
    static bool dis_beacon_center(const vector<int> &P1,const vector<int> &P2){
        return P1[1]<P2[1];
    }
    int get_pixel(Mat& img, Point pt) {
        int width = img.cols; //图片宽度
        int height = img.rows; //图片宽度 t;//图片高度
        uchar* ptr = (uchar*) img.data + pt.y * width; //获得灰度值数据指针  .data 返回首地址
        int intensity = ptr[pt.x];
        return intensity;
    }
    void clear()
    {
        first_rect.clear();
    }
    void first_delect()
    {
        clear();
        vector<vector<Point> > contours;
        vector<Vec4i> hierarchy;
        findContours(thres_one, contours, RETR_TREE, CHAIN_APPROX_SIMPLE);
        vector <vector<Point> >::iterator iter = contours.begin();
        for (; iter != contours.end();)
        {
            double g_dConArea = contourArea(*iter);
            if (g_dConArea>10000 || g_dConArea<50)

            {
                iter = contours.erase(iter);
            }
            else
            {
                ++iter;
            }
        }
        for (size_t i = 0; i < contours.size(); i++)
        {
            rect = minAreaRect(contours[i]); //  用最小矩形包起来每个轮廓便于后续处理
            Point2f P[4];
            rect.points(P);
            //  得到外包矩形的四个顶点坐标
            //这些顶点应该是白色区域，即像素值为 255
            int p_value_0=get_pixel(thres_one,Point2f(abs(P[0].x),abs(P[0].y)));
            int p_value_1=get_pixel(thres_one,Point2f(abs(P[1].x),abs(P[1].y)));
```

```
                int p_value_2=get_pixel(thres_one,Point2f(abs(P[2].x),abs(P[2].y)));
                int p_value_3=get_pixel(thres_one,Point2f(abs(P[3].x),abs(P[3].y)));
                int p_sum_value=p_value_1+p_value_2+p_value_3+p_value_0;
                if(p_sum_value>=4*255)
                {
                    vector<Point> beacon_contour_height = contours[i];
                    vector<Point> beacon_contour_width = contours[i];
                    vector<Point> beacon_contour_vertex=contours[i];

sort(beacon_contour_width.begin(),beacon_contour_width.end(),contourwidth);

sort(beacon_contour_height.begin(),beacon_contour_height.end(),contourheight);

sort(beacon_contour_vertex.begin(),beacon_contour_vertex.end(),contourvectex);
                    vector<Point> beacon_vertex;
                    int width,height;

width=beacon_contour_width[beacon_contour_width.size()-1].x-beacon_contour_width[0]
.x;

height=beacon_contour_height[beacon_contour_height.size()-1].y-beacon_contour_height
[0].y;
                    if(height>1.5*width) //灯条的长宽比例
                    {
                        beacon_vertex.push_back(beacon_contour_vertex[0]);

beacon_vertex.push_back(beacon_contour_vertex[beacon_contour_vertex.size()-1]);
                        Point2f P[4];
                        rect.points(P);
                        for (int j = 0; j <= 3; j++)
                            line(thres_two, P[j], P[(j + 1) % 4], Scalar(0, 255, 0), 2);
                        first_rect.push_back(rect);     //旋转矩形
                        beacons_vertex.push_back(beacon_vertex);
                    }
                    beacon_contour_height.clear();
                    beacon_contour_width.clear();
                    beacon_contour_vertex.clear();
                    beacon_vertex.clear();
                }
            }
            imshow("first_detect",thres_two);
            if(first_rect.size()!=0)
                sort(first_rect.begin(), first_rect.end(), RotateRectSort);
        }
        void get_Image(Mat& src1, Mat& dst, int nThre)
        {
            for (int nI = 0; nI<src1.rows; nI++)
```

```
        {
            uchar* pchar1 = src1.ptr<uchar>(nI);
            uchar* pchar3 = dst.ptr<uchar>(nI);
            for (int nJ = 0; nJ <src1.cols; nJ++)
            {
                if (pchar1[nJ]> nThre)
                {
                    pchar3[nJ] = 0;
                }
                else
                {
                    pchar3[nJ] = 255;
                }
            }
        }
    }
    void get_Image1(Mat& src1, Mat& dst, int nThre,int mThre)
    {
        for (int nI = 0; nI<src1.rows; nI++)
        {
            uchar* pchar1 = src1.ptr<uchar>(nI);
            uchar* pchar3 = dst.ptr<uchar>(nI);
            for (int nJ = 0; nJ <src1.cols; nJ++)
            {
                if (pchar1[nJ]> nThre&&pchar1[nJ]<mThre)
                {
                    pchar3[nJ] = 0;
                }
                else
                {
                    pchar3[nJ] = 255;
                }
            }
        }
    }
    //**************
    int main(int argc, char* argv[])
    {
        //按照开发手册对采集到的图像进行二次开发 1
        int                 iCameraCounts = 1;
        int                 iStatus=-1;
        tSdkCameraDevInfo   tCameraEnumList;
        int                 hCamera;
        tSdkCameraCapbility tCapability;      //设备描述信息
        tSdkFrameHead       sFrameInfo;
        BYTE*               pbyBuffer;
```

```
        IplImage *iplImage = NULL;                  //OpenCV 的 C 语言接口，用于存储图像
        int                     channel=3;
        BOOL                    AeState;
        int                     ExposureTime=atoi(argv[1]);   //传入的第一个参数是设置的曝光时间
        CameraSdkInit(1);

        //枚举设备，并建立设备列表
        iStatus = CameraEnumerateDevice(&tCameraEnumList,&iCameraCounts);
            printf("state = %d\n", iStatus);

            printf("count = %d\n", iCameraCounts);
        //没有连接设备
        if(iCameraCounts==0){
            return -1;
        }

        //相机初始化。初始化成功后，才能调用任何其他相机相关的操作接口
        iStatus = CameraInit(&tCameraEnumList,-1,-1,&hCamera);

        //初始化失败
            printf("state = %d\n", iStatus);
        if(iStatus!=CAMERA_STATUS_SUCCESS){
            return -1;
        }

        //获得相机的特性描述结构体。该结构体中包含了相机可设置的各种参数的范围信息。决定了相关函数的参数
        CameraGetCapability(hCamera,&tCapability);

        //
        g_pRgbBuffer = (unsigned char*)malloc(tCapability.sResolutionRange.iHeightMax*tCapability.sResolutionRange.iWidthMax*3);
        //g_readBuf = (unsigned char*)malloc(tCapability.sResolutionRange.iHeightMax*tCapability.sResolutionRange.iWidthMax*3);

        /*让 SDK 进入工作模式，开始接收来自相机发送的图像
        数据。如果当前相机是触发模式，则需要接收到
        触发帧以后才会更新图像。    */
        CameraPlay(hCamera);

        /*其他的相机参数设置     二次开发！
```

例如 CameraSetExposureTime CameraGetExposureTime 设置/读取曝光时间

CameraSetImageResolution CameraGetImageResolution 设置/读取分辨率

CameraSetGamma、CameraSetConrast、CameraSetGain 等设置图像伽马、对比度、RGB 数字增益等等。

更多的参数的设置方法，，清参考 MindVision_Demo。本例程只是为了演示如何将 SDK 中获取的图像，转成 OpenCV 的图像格式,以便调用 OpenCV 的图像处理函数进行后续开发

*/

```
if(tCapability.sIspCapacity.bMonoSensor){
    channel=1;
    CameraSetIspOutFormat(hCamera,CAMERA_MEDIA_TYPE_MONO8);
}else{
    channel=3;
    CameraSetIspOutFormat(hCamera,CAMERA_MEDIA_TYPE_BGR8);
}
objP.push_back(Point3f(3,40,0));
objP.push_back(Point3f(3,92,0));
objP.push_back(Point3f(130,40,0));
objP.push_back(Point3f(130,92,0));
objP1.push_back(Point3f(3,40,0));
objP1.push_back(Point3f(3,90,0));
objP1.push_back(Point3f(230,40,0));
objP1.push_back(Point3f(230,90,0));
//可以在这里进行二次开发
FileStorage fs("intrinsics.xml", FileStorage::READ);
fs.open("intrinsics.xml", cv::FileStorage::READ);
cout << "\nimage width: " << static_cast<int>(fs["image_width"]);
cout << "\nimage height: " << static_cast<int>(fs["image_height"]);
cv::Mat intrinsic_matrix_loaded, distortion_coeffs_loaded;
fs["camera_matrix"] >> intrinsic_matrix_loaded;
fs["distortion_coefficients"] >> distortion_coeffs_loaded;
cout << "\nintrinsic matrix:" << intrinsic_matrix_loaded<<endl;
//原版的读取普通相机的程序
int exp = 1000;
const char* camera_r("/dev/video0");
const char* camera_l("/dev/video1");
int fd_r=open(camera_r,O_RDWR); //通过 fd 是否为-1 判断是否打开摄像头文件 video0
int fd_l=open(camera_l,O_RDWR);
if(fd_r!=-1)
    cout<<"open the right camera!"<<endl;
RMVideoCapture cap_r(camera_r,1); //这里定义的 cap 类里面的 fd 变量记录的
```

```
是打开文件是否成功
        cap_r.setVideoFormat(640,480,1);
        //cap_r.setVideoFPS(30);
        cap_r.setExposureTime(0,exp);
        cap_r.startStream();
        if(fd_l!=-1)
            cout<<"open the left camera!"<<endl;
        RMVideoCapture cap_l(camera_l,1); //这里定义的 cap 类里面的 fd 变量记录的
是打开文件是否成功
        cap_l.setVideoFormat(640,480,1);
        //cap_l.setVideoFPS(30);
        cap_l.setExposureTime(0,exp);
        cap_l.startStream();
        //原版的读取普通相机的程序
        //工业相机
        CameraSetAeState(hCamera,FALSE);//设置为手动曝光模式
        CameraSetExposureTime(hCamera,ExposureTime); //曝光时间，单位为微妙
        int fd,wr_num=0;
        struct termios options;
        //speed_t baud_rate_i,baud_rate_o;
        fd=open("/dev/ttyUSB0",
O_RDWR|O_NONBLOCK|O_NOCTTY|O_NDELAY);//打开串口
        //串口在 linux 系统中被看做是一个文件
        //O_RDWR 读写模式
        //O_NOCTTY 如果路径名指向终端设备，不要把这个设备用作控制终端()
        //对于串口的打开操作，必须使用 O_NOCTTY 参数，它表示打开的是一个
终端设备，程序不会成为该端口的控制终端
        // 如果不使用此标志，任务的一个输入(比如键盘终止信号等)都会影响进
程。
        //O_NONBLOCK 如果路径名指向 FIFO/块文件/字符文件，则把文件的打开
和后继 I/O 设置为非阻塞模式（nonblocking mode）
        //非阻塞模式，如果 Receive 或者 Send 不足够快，造成内存的 Copy 很多，
从而降低效率。
        //O_NDELAY 表示不关心 DCD 信号所处的状态（端口的另一端是否激活或
者停止）。
        //非阻塞模式下，read 停留在代码出直至读到数据，没读到直接返回 0
        if(fd==-1) {
            printf("can not open the USB0!\n");
            fd=open("/dev/ttyUSB1",
O_RDWR|O_NONBLOCK|O_NOCTTY|O_NDELAY);//打开串口
            if(fd==-1)
                printf("can not open the USB1!\n");
            else
                printf("open USB1 ok!\n");
```

```
        }
        else
            printf("open USB0 ok!\n");

        //判断是否是终端设备
        if(isatty(STDIN_FILENO) == 0)
            printf("不是终端设备\n");
        else
            printf("是终端设备\n");
        //
    //    if( fcntl(fd, F_SETFL, 0) <0 ) //改为阻塞模式
    //        printf("fcntl failed\n");
    //    else
    //        printf("fcntl=%d\n", fcntl(fd, F_SETFL, 0));
        if(fcntl(fd,F_SETFL,FNDELAY) < 0)//非阻塞，覆盖前面 open 串口的属性
            printf("fcntl failed\n");
        else
            printf("fcntl=%d\n",fcntl(fd,F_SETFL,FNDELAY));
        tcgetattr(fd, &options);
        //串口设置
        options.c_cflag |= (CLOCAL | CREAD);
        options.c_cflag &= ~PARENB;//设置无奇偶校验位，N
        options.c_cflag &= ~CSTOPB; //设置停止位 1
        options.c_cflag &= ~CSIZE;
        options.c_cflag |= CS8; //设置数据位
    //    //如果不是开发终端之类的，只是串口传输数据，而不需要串口来处理，那么使用原始模式(Raw Mode)方式来通讯，设置方式如下
        //options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); /*Input*/
        //options.c_oflag &= ~OPOST; /*Output*/
        options.c_cc[VTIME]=0;//阻塞模式的设置
        options.c_cc[VMIN]=1;
        //options.c_cc[VMIN]=0; 可以尝试一下，跟等待的字节数有关
        //设置波特率
        //之前的波特率是 4098 之后仍然是，就是函数写的问题了，不过我设置的是 115200
        // 在 COM2 端口也是设置 115200 可以实现通信，说明设置是对了，不过显示仍存在问题
        //改变波特率后，获得的值仍然是 4098，但是应该是设置成功了
        cfsetospeed(&options, B115200);
        //tcgetattr(fd, &newstate);

        //激活新配置
        tcsetattr(fd, TCSANOW, &options);
        //非阻塞模式下的 read 和 write:
```

```
        //当缓冲区内没有数据的时候，read 函数立马返回，不会在此处阻塞
        char buff[1024];
        int nread;
        int key;
        while(1)
        {
            nread = read(fd, buff, 1024);//每次循环开始检测是否接受到串口的大符击打请求
            //cout<<"read:"<<nread<<endl;
            if(nread>0)    //测试是否接受到了来自下位机的请求击打大符的请求，进入此处一次，标志位反转一次。
            {    //中间放打大符的代码
                Mat windmill_r,windmill_l;
    //                cap_r >> windmill_r;
    //                cap_l >> windmill_l;
            }
            else {
                double Time = (double) cvGetTickCount();
                if (CameraGetImageBuffer(hCamera, &sFrameInfo, &pbyBuffer, 1000) == CAMERA_STATUS_SUCCESS) {
                    CameraImageProcess(hCamera, pbyBuffer, g_pRgbBuffer, &sFrameInfo);
                    if (iplImage) {
                        cvReleaseImageHeader(&iplImage);
                    }
                    iplImage = cvCreateImageHeader(cvSize(sFrameInfo.iWidth, sFrameInfo.iHeight), IPL_DEPTH_8U, channel);
                    cvSetData(iplImage, g_pRgbBuffer, sFrameInfo.iWidth * channel);//此处只是设置指针，无图像块数据拷贝，不需担心转换效率
                    //以下方式将 Ipl 图像(c 下的 opencv)变成 Mat 图像(c++下的 opencv)
                    Mat Iimag;//这里只是进 行指针转换，将 IplImage 转换成 Mat 类型
                    Iimag = cvarrToMat(iplImage);
                    Mat armor = Iimag(Rect(320, 272, 640, 480));
                    //imshow("origin",Iimag);
                    key = waitKey(1);
                    int total_number = 0;
                    //************
                    //imshow("The original image", armor);
                    thres_one.create(armor.rows, armor.cols, CV_8UC1);
                    thres_four.create(armor.rows, armor.cols, CV_8UC1);
                    split(armor, diff);
                    //imshow("B",diff[0]);
                    //imshow("G",diff[1]);
```

```
                    //imshow("R",diff[2]);
                    get_Image(diff[2], thres_one,beacon_threshold);  //调参数，传入的是 green 通道的图像，第 3 个参数是阈值    中间的数字的阈值大概是 10-20，因此可以通过这个判断是不是同一个装甲板上的灯条
                    //imshow("阈值化", thres_one);
                    get_Image1(diff[1], thres_four, number_threshold,
                                beacon_threshold); //这个参数跟后面判断是否有数字有关系，此阈值应该等于后面的阈值 Pixel>n
                    imshow("4",thres_four);
                    thres_two = thres_one.clone();
                    first_delect();
                    //这里得到的 beacons_vertex.size()=first_rect.size()
                    //这里的 beacons_vertex 的大小是符合长款比的灯条个数
                    if (first_rect.size() != 0) {
                        for (int i = 0; i < first_rect.size(); i++) {
                            Point n1(beacons_vertex[i][0]); //某轮廓最上层点
                            Point n2(beacons_vertex[i][1]); //某轮廓最下层点
                            if (abs(n1.x - n2.x) >= 25) //调整   同一个轮廓的宽度不应该超过 25
                                continue;
                            if (!first_rect.empty()) {
                                float area1 = first_rect[i].size.area();
                                for (int j = i + 1; j < first_rect.size(); j++) {
                                    Point n3(beacons_vertex[j][0]); //另一轮廓最上层点
                                    Point n4(beacons_vertex[j][1]); //另一轮廓最下层点
                                    if (abs(n3.x - n4.x) >= 25) //调整
                                        continue;
                                    float area2 = first_rect[j].size.area();//
                                    //设计新的平行方法
                                    //手动找到轮廓的最高最低点
                                    int upper_point_y_diff = n1.y - n3.y;
                                    int bottom_point_y_diff = n2.y - n4.y;
                                    int upper_point_x_diff = abs(n1.x - n3.x);
                                    int bottom_point_x_diff = abs(n2.x - n4.x);
                                    if (area1 <= area2) {
                                        float temp = area1;
                                        area1 = area2;
                                        area2 = temp;
                                    }
                                    if (area2 <= 2.5 * area1 &&
                                        ((abs(upper_point_y_diff  <=  100)  &&
abs(bottom_point_y_diff) <= 100))
```

```
                                        && upper_point_x_diff >= 20 &&
upper_point_x_diff <= pixel_x_diff &&
                                        bottom_point_x_diff >= 20 &&
                                        bottom_point_x_diff <= pixel_x_diff)
{  //调整
                                        int number_or_not = 0;
                                        if (n3.x > n1.x) {
                                            int dif = n3.x - n1.x;
                                            if (dif % 2 == 0)
                                                dif = dif;
                                            else
                                                dif += 1;
                                            int dif_2 = dif / 2;
                                            for (int i = n1.x + dif_2 - 10; i !=
n1.x + dif_2 + 10; i++) {
                                                for(int
j=(n1.y+n3.y)/2;j!=(n2.y+n4.y)/2;j++)
                                                {
                                                    int
pixel=get_pixel(thres_four,Point(i,j));
                                                    if(pixel==0)
                                                        number_or_not++;
                                                }
                                            }
                                        } else {
                                            int dif = n1.x - n3.x;
                                            if (dif % 2 == 0)
                                                dif = dif;
                                            else
                                                dif += 1;
                                            int dif_2 = dif / 2;
                                            for (int i = n3.x + dif_2 - 10; i !=
n3.x + dif_2 + 10; i++) {
                                                for(int
j=(n1.y+n3.y)/2;j!=(n2.y+n4.y)/2;j++)
                                                {
                                                    int
pixel=get_pixel(thres_four,Point(i,j));
                                                    if(pixel==0)
                                                        number_or_not++;
                                                }
                                            }
                                        }

//cout<<"numbers:"<<number_or_not<<endl;
                                        if (number_or_not <= pixel_number) //-1
```

是用在夜晚环境下，识别不到 4 的时候，下午的时候参数改为 150

```
                                        continue;
                                    //cout << number_or_not << endl;
                                    total_number++; //对应的是匹配灯条的
Index,方便后续处理->记录每组匹配的信息，后续直接处理
                                    int x_ = (n1.x + n2.x + n3.x + n4.x) / 4;
                                    int y_ = (n1.y + n2.y + n3.y + n4.y) / 4;
                                    //cout << "x:" << x_ << endl;
                                    //cout << "y:" << y_ << endl;
                                    int x_diff=abs(x_-320);
                                    int y_diff=abs(y_-300);
                                    int score = x_diff + y_diff;//记录每组的
中心位置，当自瞄跟上的时候，装甲板中心区域应该保持在视野的中间
                                    //即中间区域的坐标值距离(640,512)的
欧式距离应该最小。
                                    //cout<<"score: "<<score<<endl;
                                    vector<int> index_score;
                                    index_score.push_back(total_number);
                                    index_score.push_back(score);
                                    index_score.push_back(n1.x);
                                    index_score.push_back(n1.y);
                                    index_score.push_back(n2.x);
                                    index_score.push_back(n2.y);
                                    index_score.push_back(n3.x);
                                    index_score.push_back(n3.y);
                                    index_score.push_back(n4.x);
                                    index_score.push_back(n4.y);
                                    scores_group.push_back(index_score);
                                }
                            }
                        }
                    }
                // cout << "size:" << scores_group.size() << endl;
                    if (scores_group.size() >= 1) {
                        sort(scores_group.begin(),          scores_group.end(),
dis_beacon_center);
                        int selected_index = 0;
                        //这里应该对所有可能的灯条匹配队列根据中心位置
进行排序,按照中心位置排列，[0]就应该是最靠近中心的灯条。
                        //但是有可能出现把不在一块的装甲板的两块灯条识
别在一起，此时，这两个灯条之间还包裹着应该是正确的灯条匹配
                        //这个正确的灯条匹配应该也是在很靠近中心的位
置，筛选出来
                        if(scores_group.size() >= 2)
                        {
                            for(int i=0;i!=scores_group.size()-1;i++)
```

```
                                {
if(scores_group[selected_index+i+1][1]-scores_group[selected_index][1]<=150)
                                    {
                                        vector<int> max = scores_group[selected_index];
                                        vector<int> max_last = scores_group[selected_index+i+1];
                                        float max_distance = abs(max[2] - max[6]);  //最外面最大的那层装甲板
//cout<<"max_distance:"<<max_distance<<endl;
                                        float max_last_distance = abs(max_last[2] - max_last[6]); //里面的装甲板
//cout<<"max_last_distance:"<<max_last_distance<<endl;
                                        float ratio = max_last_distance/max_distance;  //长度比值，
                                        //cout<<"ratio:"<<ratio<<endl;
                                        if(ratio<=0.9&&ratio>=0.15) //找寻内部的装甲板
                                        {
                                            selected_index+=i;
                                            selected_index++;
                                                break;
                                        }
                                    }
                                }
                            }
                            //cout<<"selected_index:"<<selected_index<<endl;
                            int center1x = (scores_group[selected_index][2] + scores_group[selected_index][4]) / 2;
                            int center2x = (scores_group[selected_index][6] + scores_group[selected_index][8]) / 2;
                            int center1y = (scores_group[selected_index][3] + scores_group[selected_index][5]) / 2;
                            int center2y = (scores_group[selected_index][7] + scores_group[selected_index][9]) / 2;
if(scores_group[selected_index][2]<scores_group[selected_index][6])
                            {
                                object_cornors.push_back(Point2f((float) scores_group[selected_index][2],
                                                                 (float) scores_group[selected_index][3]));
                                object_cornors.push_back(Point2f((float)
```

```
scores_group[selected_index][4],
                                                                    (float)
scores_group[selected_index][5]));
                        object_cornors.push_back(Point2f((float)
scores_group[selected_index][6],
                                                                    (float)
scores_group[selected_index][7]));
                        object_cornors.push_back(Point2f((float)
scores_group[selected_index][8],
                                                                    (float)
scores_group[selected_index][9]));
                    }
                    else
                    {
                        object_cornors.push_back(Point2f((float)
scores_group[selected_index][6],
                                                                    (float)
scores_group[selected_index][7]));
                        object_cornors.push_back(Point2f((float)
scores_group[selected_index][8],
                                                                    (float)
scores_group[selected_index][9]));
                        object_cornors.push_back(Point2f((float)
scores_group[selected_index][2],
                                                                    (float)
scores_group[selected_index][3]));
                        object_cornors.push_back(Point2f((float)
scores_group[selected_index][4],
                                                                    (float)
scores_group[selected_index][5]));
                    }
                    cout<<object_cornors[0]<<endl;
                    cout<<object_cornors[1]<<endl;
                    cout<<object_cornors[2]<<endl;
                    cout<<object_cornors[3]<<endl;
                    float
beacon_height=abs(object_cornors[0].y-object_cornors[1].y);
                    float
beacon_width=abs(object_cornors[1].x-object_cornors[2].x);
                    float beacon_ratio=beacon_width/beacon_height;
                    cout<<"beacon_height"<<beacon_height<<endl;
                    cout<<"beacon_width:"<<beacon_width<<endl;
                    cout<<"beacon_ratio: "<<beacon_ratio<<endl;
                    //这里应该使用的是 CV_ITERATIVE
                    //内部使用的是 LM 优化方法，有较好的鲁棒性
                    if(beacon_ratio<=2.9)
                        solvePnP(objP,                    object_cornors,
```

```
intrinsic_matrix_loaded, distortion_coeffs_loaded, r3, r2, false,
                                        CV_ITERATIVE);
                                else
                                    solvePnP(objP1,                    object_cornors,
intrinsic_matrix_loaded, distortion_coeffs_loaded, r3, r2, false,
                                        CV_ITERATIVE);
                                //cout << "r2:"<<r2 << endl;
                                Mat r1(3, 3, CV_64FC1);
                                Rodrigues(r3, r1);
                                float r0_0 = r1.at<double>(0, 0);
                                float r0_1 = r1.at<double>(0, 1);
                                float r0_2 = r1.at<double>(0, 2);
                                float r1_0 = r1.at<double>(1, 0);
                                float r1_1 = r1.at<double>(1, 1);
                                float r1_2 = r1.at<double>(1, 2);
                                float r2_0 = r1.at<double>(2, 0);
                                float r2_1 = r1.at<double>(2, 1);
                                float r2_2 = r1.at<double>(2, 2);
                                Point3f target(67, 62.5, 0);
                                Point3f target_new((target.x*r0_0 + target.y*r0_1 +
target.z*r0_2), (target.x*r1_0 + target.y*r1_1 + target.z*r1_2), (target.x*r2_0 +
target.y*r2_1 + target.z*r2_2));
                                target_new.x += r2.at<double>(0, 0);
                                target_new.x = -target_new.x;
                                target_new.y += r2.at<double>(1, 0);
                                target_new.z =target_new.z + r2.at<double>(2, 0);
                                 cout<<"x:"<<target_new.x<<endl;
                                //根据实际数据进行线性拟合
                                //cout << target_new << endl;
                                line(armor, Point(center1x, center1y), Point(center2x,
center2y), Scalar(0, 0, 255), 4, LINE_AA);
                                line(armor, Point((center1x + center2x) / 2, center1y +
40),
                                        Point((center1x + center2x) / 2, center1y - 40),
Scalar(0, 0, 255), 4, LINE_AA);
                                //假设 yaw 轴是圆周旋转的，pitch 轴是上下俯仰的
                                target_new.y=-target_new.y;
                                //cout<<"y:"<<target_new.y<<endl;
                                float y_true=target_new.y*0.987-target_new.z*0.156;
                                float z_true=target_new.y*0.156+target_new.z*0.987;
                                target_new.y=y_true;
                                target_new.z=z_true;
                                //y 现在的方向是摄像头上方为正
                                target_new.z+=132.6;    //安装尺寸有关
                                //cout<<"z:"<<target_new.z<<endl;
                                target_new.y-=66.3;      //安装尺寸有关
```

```
                    if(target_new.z<=1000)
                    {
                        pixel_x_diff=600;
                        pixel_number=150;
                    }
                    else if(target_new.z<=2000)
                    {
                        pixel_x_diff=450;
                        pixel_number=130;
                    }
                    else if(target_new.z<=3000)
                    {
                        pixel_x_diff=300;
                        pixel_number=110;
                    }
                    else
                    {
                        pixel_x_diff=250;
                        pixel_number=90;
                    }
                    //cout<<"pixel:"<<pixel_x_diff<<endl;
                    //cout<<"pixel_number:"<<pixel_number<<endl;
                    unsigned char buf[9]={0x25,0x25,0x25,0x26,0x25,0x26,0x00,0x7e,0x7d}; //向串口发送的数组 应该传给下位机的
                    //unsigned char buf1[8]={0x88,0xA1,0x04,0x25,0x26,0x25,0x26,0x00}; //向串口发送的数组 应该传给下位机的
                    //这里的 8 位，除去最后的两帧，中间的 6 位表示 pitch 和 yaw 的数据和正负
                    if(target_new.x<=0)   //[0-2]是关于 yaw 的数据
                        buf[0]=0;
                    else
                        buf[0]=1;
                    if(target_new.y<=0)   //[3-5]是关于 pitch 的数据
                        buf[3]=0;
                    else
                        buf[3]=1;
                    double yaw=atan2(abs(target_new.x),abs(target_new.z));
                    yaw=yaw*180/pi;
                    yaw*=10;
                    double pitch=atan2(abs(target_new.y),abs(target_new.z));
                    pitch=pitch*180/pi;
                    int z_diff;
```

```
if(target_new.z>=500)
        z_diff=target_new.z-500;
else
        z_diff=0;
int pitch_comp;
if(z_diff<=1500)
        pitch_comp=z_diff*14/500;
else if(z_diff<=3000)
        pitch_comp=1200*14/500+(z_diff-1200)*8/500;
else
        pitch_comp=3000*10/500+(z_diff-3000)*7/500;
if(pitch_comp_last!=-1)
{

pitch_comp=0.1*pitch_comp+0.9*pitch_comp_last;
}
pitch_comp_last=pitch_comp;
// cout<<"pitch_comp "<<pitch_comp<<endl;

pitch*=10;
if(buf[3]==1)
        pitch+=pitch_comp;
else
{
        if(pitch>pitch_comp)
                pitch-=pitch_comp;
        else
        {
                pitch=pitch_comp-pitch;
                buf[3]=1;
        }
}
if(yaw_last==-1&&pitch_last==-1)
{
        yaw_last=yaw;
        pitch_last=pitch;
}
else
{

if(abs(yaw_last-yaw)>=10||abs(pitch_last-pitch)>=10)
        {
                //cout<<"error"<<endl;
                error_times++;
                if(error_times>=3)
                {
                        yaw_last=yaw;
```

```
                    pitch_last=pitch;
                    error_times=0;
                    yaw_int_last=yaw;
                    x_last=-1;
                }
                else
                {
                    scores_group.clear();
                    object_cornors.clear();
                    beacons_vertex.clear();

CameraReleaseImageBuffer(hCamera,pbyBuffer);
                    continue;
                }
            }
            else
            {
                yaw_last=yaw;
                pitch_last=pitch;
            }
        }
        if(x_sum==-1)
        {
            x_sum+=target_new.x;
            x_sum+=1;
        }
        else
            x_sum+=target_new.x;
        //cout<<"x_sum:"<<x_sum<<endl;
        int yaw_int= static_cast<int>(yaw);
        int pitch_int= static_cast<int>(pitch);
        if(buf[3]==0)
            cout<<"pitch: -"<<pitch_int<<endl;
        if(buf[3]==1)
            cout<<"pitch: +"<<pitch_int<<endl;
        int yaw1= static_cast<int>(yaw_int/100);
        int yaw2= static_cast<int>(yaw_int%100);
        if(buf[0]==0)
            cout<<"yaw: -"<<yaw_int<<endl;
        if(buf[0]==1)
            cout<<"yaw: +"<<yaw_int<<endl;
        int pitch1= static_cast<int>(pitch_int/100);
        int pitch2= static_cast<int>(pitch_int%100);
        char yaw1_c(yaw1);
        char yaw2_c(yaw2);
        char pitch1_c(pitch1);
        char pitch2_c(pitch2);
```

```
                                buf[1]=yaw1_c;
                                buf[2]=yaw2_c;
                                buf[4]=pitch1_c;
                                buf[5]=pitch2_c;
                                buf[6]=0;
                                for(int i=0;i<6;i++)
                                    buf[6]+=buf[i];
                                //写入串口
                                wr_num=write(fd,buf,sizeof(buf));
                                if(wr_num>0)
                                    printf("write success!\n");
                                 else
                                    printf("write fail!\n");

                            }
                            imshow("初始图", armor);
                            scores_group.clear();
                            object_cornors.clear();
                            beacons_vertex.clear();
                        }
                        CameraReleaseImageBuffer(hCamera, pbyBuffer);
                        // 在成功调用 CameraGetImageBuffer 后，必须调用 CameraReleaseImageBuffer 来释放获得的 buffer。
                        //否则再次调用 CameraGetImageBuffer 时，程序将被挂起一直阻塞，直到其他线程中调用 CameraReleaseImageBuffer 来释放了 buffer
                        Time = (double) cvGetTickCount() - Time;
                        cout<<Time /(cvGetTickFrequency()*1000)<<"ms"<<endl;
                        if (key == 'q') break;
                        beacons_vertex.clear();
                        //************
                    }
                }
            }
            CameraUnInit(hCamera);
            //注意，现反初始化后再 free
            free(g_pRgbBuffer);
            close(fd);
            cap_r.closeStream();
            cap_l.closeStream();
            return 0;
        }
```

makefile 文件

```
CXX ?= g++     #makefile 中符号的意义：
#     = 是最基本的赋值
#     := 是覆盖之前的值
```

```
#    ?= 是如果没有被赋值过就赋予等号后面的值
#    += 是添加等号后面的值
#这里的两个反引号``是执行了 linux 下的这个命令 pkg-config，这个命令是查找已安装的库的使用接口，用于 C++编译和程序的链接，通俗来说，就是头文件 include 和链接库 lib 的位置
#所有用 opencv 的其他程序，在编译时，只需要写“pkg-config opencv –-libs(库) –-cflags(头文件)”,而不需要自己去找 opencv 的头文件在哪里，要链接的库在哪里！省时省力！
#如果你写了一个库，不管是静态的还是动态的，要提供给第三方使用，那除了给人家库/头文件，最好也写一个 pc 文件，这样别人使用就方便很多，不用自己再手动写依赖了你哪些库，只需要敲一个”pkg-config [YOUR_LIB] –-libs –-cflags”。
CFLAGS =`pkg-config --cflags opencv`      #这里查找的是 include 文件的位置
LIBS = `pkg-config --libs opencv`         #这里查看的是 lib 库的位置
INCLUDE= -I../../include
all: main
main:
    $(CXX) $(CFLAGS) -o main main.cpp RMCapture.cpp $(LIBS) $(INCLUDE) -lMVSDK -O2  #-O1 -O2 -O3 这三个是编译优化选项，可以选择不同的代码优化方式
#优化后的代码运行速度会有较大提升。
clean:
    rm -f *.o
    rm -f main
```

Linux 开机自启动脚本文件

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          land.sh
# Required-start:    $local_fs $remote_fs $network $syslog
# Required-Stop:     $local_fs $remote_fs $network $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: starts the svnd.sh daemon
# Description:       starts svnd.sh using start-stop-daemon
### END INIT INFO

#任务脚本
#进入要执行脚本目录
cd /home/hanyunhai/Desktop/linuxSDK_V2.1.0.2/demo/OpenCv
#取得 root 权限，'123456'为密码，不用加引号，'ls'无实际作用
echo 84408356a|sudo -S ls
#执行脚本./bin/mywork，sudo -S 需要加上
sudo -S ./main 500
#任务脚本
```

附录 2（开题报告）

燕 山 大 学

本科毕业设计（论文）开题报告

课题名称：基于视觉的自动追踪控制系统设计

学　　院：机械工程学院

年级专业：2015 级机电三班

学生姓名：韩云海

指导教师：史小华

填写日期：03/09/2019

综述本课题国内外研究动态，说明选题的依据和意义

1.1 选题的依据和意义

机电一体化概念最早出现在 1971 年日本杂志《机械设计》的副刊上，其是一种将械技术、电工电子技术、微电子技术、信息技术、传感器技术、接口技术、信号变换技术等多种技术进行有机地结合，并综合应用到实际中去的综合技术[1]。凭借着对机电一体化技术的深入研究和不断尝试，日本在上世纪 70 年代紧跟世界发展潮流，不断提高在各种工业产品的市场占有率，如汽车，家电等。而我国在当时才逐步开始在机电一体化方面进行研究和应用，起步晚，因此在面对全球经济市场的竞争中，我国在这个领域面临严峻的形式。

另一方面，步入 21 世纪以后，互联网产业成为了最受瞩目的产业。人工智能(AI)和计算机视觉(CV)以及自然语言处理(NLP)等许多新兴技术让人们看到了无限的可能性。除此之外，机器人领域[2]，作为一个最具代表性的交叉领域，也吸引住了许多人的目光。2015 在北京举办的世界机器人大会上，习近平主席对机器人领域的发展寄予厚望，他表示，中国将机器人和智能制造纳入了国家科技创新的优先重点领域，中国将积极追赶智能产业蓬勃兴起的潮流，布局中国自身经济转型的同时，为人类文明发展贡献中国智慧和中国力量[3]。

综合来看，中国虽然在 20 世纪末的机电一体化潮流中稍逊一筹，但是在当下的互联网浪潮中紧跟不舍。因此，作为机电专业的学生，我们应当将计算机理论知识与我们自身的专业知识结合起来，真正做到“软硬结合”，既设计出满足使用并且易于维修的机电系统，又设计出代码严谨容错率高的计算机软件系统，争取成为未来能独当一面，规划整个系统的总工程师！

因此，我结合了我在 RM 俱乐部的经历，选择了这个选题。此题目既包括了机器人底盘及云台的设计，云台电机的 yaw 和 pitch 轴稳定性和快速性的调试，也包括了图像处理模块视觉系统的设计，还包括了下位机控制系统与上位机视觉系统之间的数据通信，因此能有效地提高我对于机电计系统的理解，为以后的学习打下基础。

1.2 国内外研究现状

近年来，国内外对基于视觉的自动追踪问题的研究正处于高潮，许多重要的国

际期刊以及许多重要的国际会议发表了大量的有关视觉跟踪方面的论文。基于视觉的自动追踪能够引起广泛讨论是由于它能够应用于民用[4]和军事的许多领域。例如，在视频监控领域，最常见的是对于停车场、民宅、学校、公共场合等地的监控[5][6][7]，以防止偷盗，破坏行为的发生。1997 年，美国国防部高级研究项目署设立了以卡内基梅隆大学为首，麻省理工学院等高校参与的视觉监控项目 VSAM[8](Visual surveillance and monitoring),该系统可应用于民用场景及战场的实时监控。随后，卡内基梅隆大学又建立了一个校园监控系统[9]，以保障校园安全。除此之外，在交通系统中，视觉跟踪技术的研究也有非常广阔的应用前景，包括行人行为判定，车辆异常行为检测，智能车辆等很多方面。Coifman 等人[10]建立了一个基于视频图像处理系统的交通监控系统，此系统可以用来监控交通流量以及对不同车型进行统计。这项研究在国内外非常普遍，主要研究解决的问题是如何准确实现对车辆的分割和跟踪。道路上车辆异常行为检测在交通事故的预防和处理方面也有重要意义。Tai 等人[11]设计了一个可以用于交通事故检测的视频监视系统，能够自动检测视野中的运动车辆并预测其运动轨迹。Haag 和 Nagel[12]专门对机动车辆的跟踪问题进行了研究。Pai 等人[13]针对十字路口的行人检测和跟踪进行了研究以保证驾驶员可以在十字路口安全驾驶。Masound 和 Papanikolopoulos[14]通过对道路行人的跟踪并计数，为道路交通管理提供了信息。智能车辆是目前计算机视觉研究中的一个热点，其最终目的是实现自动驾驶车辆。

相比国外，我国在视频监控领域方面的研究起步稍晚，但是在政府各部门的大力支持下，也在这方面取得了很大的进展。20 世纪 90 年代初，一些高校和交通研究机构引荐国外先进理论和控制思想，开始了城市交通车辆系统技术研究，并在一些交通系统比较发达的地方设定测试点进行试验，取得了一定成果。中科院自动化研究所同雷丁大学展开合作，双方建立了共同的软件平台，并发起了国际视觉监控学术研讨会。另外，一些公司也在视觉监控技术上取得一定进展。

一、研究的基本内容，拟解决的主要问题

研究的基本内容包括：

1.搭建具有 pitch,yaw 轴两自由度的云台，并安装在麦轮驱动的机器人底盘上。

2.对安装好的云台进行控制算法调试，以保证其满足位置响应和速度响应的要

求。

3.通过机器人上搭载的微型计算机和安装好的摄像头，运用计算机视觉的算法，识别出目标物品，并根据测距算法得到位置信息，调整云台转角，使达到实时追踪的目的。

拟解决的主要问题：

1.云台的机械结构需要满足一定的要求，以便于达到预定的控制效果。

2.机械结构尽可能设计美观，且牢固，同时减少加工件的使用，以减少成本。

3.控制算法应尽量简单，再保证控制精度的前提下，提高控制器的运算频率。

4.设计视觉识别测距算法，并对算法做适当优化，使帧率满足实际使用的要求。

5.对接视觉和电控系统，将视觉系统解算出来的角度传给电控系统，调整云台的转角。

三、研究步骤、方法及措施

研究步骤主要包括:机械结构的理论设计和实物搭建；电控系统的设计和调试；视觉系统的设计，调试和不同算法性能比较；视觉电控系统的对接。

方法及措施包括：运用理论知识指导云台机械机构的设计，包括运用理论力学知识分析云台的机械性能，运用机械设计和机械制造方面的知识设计云台的结构。通过相应的现象及经验，对于控制系统进行整定，使达到满意的控制精度。通过学习的 C++编程知识和计算机视觉算法，设计出鲁棒的视觉检测系统，并将我们解算得到的角度信息传递给下层的单片机控制系统，以此来控制云台的运动，达到实时追踪的效果。

四、研究工作进度

4.1 研究方案及对比分析

此系统可分为三个独立的子系统，分别是机械系统，电控系统和视觉系统。因此我对三个系统单独分析方案。

机械系统：

机械系统的主要目的是设计出二自由度的云台，每个自由度都由一个单独的动力源进行驱动，考虑到结构的紧凑型和控制的可行性，因此选择使用最广泛的电机驱动方式。考虑到直流供电系统，我们可以有以下三种电机驱动方式：

1. 直流电机

2. 直流无刷电机

3. 步进电机

查阅资料可知，直流无刷电机具有响应迅速，起动转矩大的特点，因此我选择直流无刷电机作为驱动源。

图 4-1 是一种可行的云台结构，可作为我设计的参考。

图 4-1 参考云台机构示意图

然而此结构有着明显的缺陷，即许多零件需要独自加工，因此成本较高，我在后续的优化中需要尽可能地减少加工件的使用，可选择使用碳板或者亚克力板作为替代或者在一些不太受力的地方使用 3D 打印件。

电控系统：

电控系统的主要目的是对两个直流无刷电机进行驱动，并通过控制算法使其具备快速响应不超调的特点。根据我以往比赛经验，有三种控制方案可供选择：

1. PID 控制算法

PID 控制器（比例-积分-微分控制器）是一个在工业控制应用中常见的反馈回路部件，由比例单元 P、积分单元 I 和微分单元 D 组成。PID 控制的基础是比例控制；积分控制可消除稳态误差，但可能增加超调；微分控制可加快大惯性系统响应速度以及减弱超调趋势。

2. ADRC 控制算法(自抗扰控制技术)

传统 PID 有限制，即调节速度和超调一定同时存在，想要得到较好的控制效果，用现代控制理论解决，要知道精确的系统模型。但是对具体模型的

建立有些复杂繁琐，要么是通过理论对系统进行建模，通过动力学模型分析系统内的响应特性，要么通过系统辨识的方法，通过多项式的传递函数去拟合系统的响应特性。然后通过系统的响应特性，设计反馈网络进行控制，并可以通过设计校正网络，提高系统的抗扰动特性以适应外界或内部的扰动情况。与此相反的是，一套整定好的 PID 参数可能只在当前的物理参数下控制效果良好，当物理条件改变后，比如重量变重，就无法继续使用了，即不抗扰动。我在网上看到了 ADRC 这种综合了 PID 和现代模型的优势的控制算法。其既保留了 PID 控制器的特点(黑箱属性)又具备现代模型控制的抗绕特点。

3. 现代控制理论

建立系统的物理模型控制，具有良好的抗扰动性。

综合比较各个控制算法实现的难易性以及此课题的需求，我认为最常见的 PID 控制器即可达到要求，因此就选择 PID 控制算法作为控制器设计方案。

视觉系统：

视觉主要的主要目的是识别目标物然后得到目标空间位置，最后将此位置换算到转角改变量传输到控制系统。这里可以细分为三个环节，识别，测距，传输。

识别部分，目标识别算法有很多，如最常见的 RCNN，SSD，YOLO 等，如图 4-2 所示，由于我需要识别的物体的特征十分独特，因此我采用基于特征的识别方法。

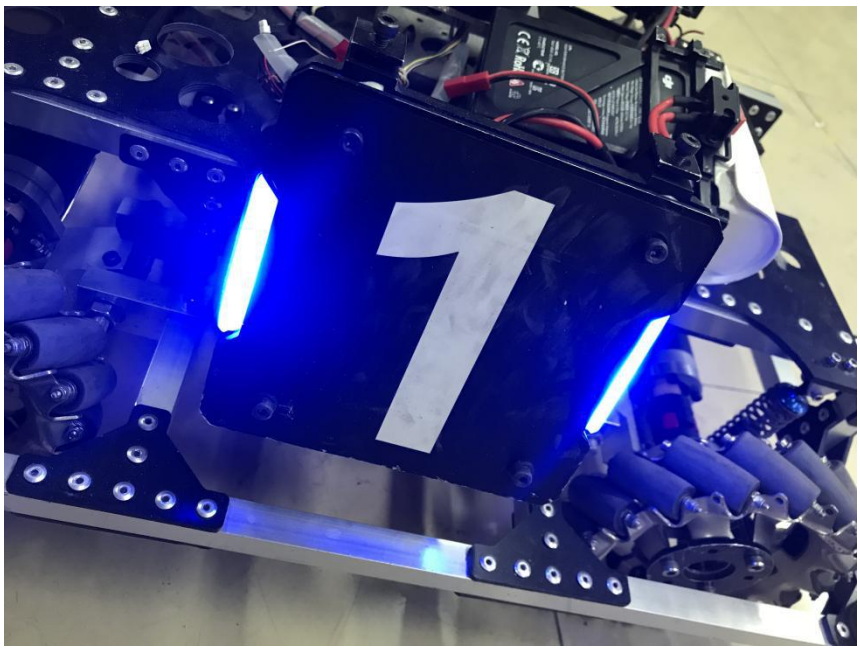


图 4-2 待识别的物体

测距部分, 常用的测距算法主要包括单目测距[15]，双目测距[16]，多目混合测距。测距算法的选择，主要是时间和成本的考虑。双目测距的结果一般来说会更精准些，但是它要对两幅图像进行图像校正，极线匹配，比较耗时。单目测距无需这些工作，但是由于单目测距对深度维度的信息缺失，因此需要知道待测物体实际的物理尺寸，才能知道通过三角形相似方法的原理求得实际的物理尺寸。多目测距本质上是结合了单目和双目测距，通过一定的决策法则对单双目进行融合，亦或是运用双方的结果根据模型产生一个新的更精确的结果。视觉测距准备先理论学习单目和双目测距，编程实现并比较各自运行时间开销。若单目视觉的准确度和帧率都能达到理想的要求，则采取单目视觉。

传输部分，此部分没有理论分析的需要，只需要完成这一目的即可。为了方便起见，我决定采用有线的串口传输数据。

4.2 技术经济性分析

与上述分析相同，我把整个系统同样分为三个子系统分析。

机械系统：

机械设计上应该尽量避免加工件的使用，以减少系统的成本。电机和电调选用 RM 俱乐部内物资，无需额外采购。

电控系统：

主控板选择 DJI 开发的主控板，可以直接接 24V 直流电源。电源也使用 DJI 开发的电源模块。正如上文所述，控制算法我决定采取 PID 控制器，其参数整定过程比较熟悉，可以胜任此次课题的要求。

视觉系统：

由于微机系统的价格昂贵，我选择使用 RM 俱乐部内部购买的英特尔微型计算机 NUC 系列。摄像头的选取上需要综合考虑摄像头感光元件能提供的最大帧率，摄像头的自身畸变，摄像头的价格，摄像头的驱动方式，摄像头的曝光是否可调等。通过在网络上找多家摄像头提供商做对比，我把摄像头价位大致定在 100~200 元内，帧率>60FPS，像素尺寸大小为 640×480。

4.3 研究工作进度

第一阶段中的内容如下：

查阅相关资料和阅读参考文献，确定课题方向，完成开题报告，准备节点考核。

第 1~3 周：查找资料并分析，确定各组成模块的设计要求。

机械部分：

这里机械部分需要实现的目标是保证二自由度的云台运动平稳可靠。

根据理论力学和机械原理的知识，确定云台设计过程需要注意的性能要求。

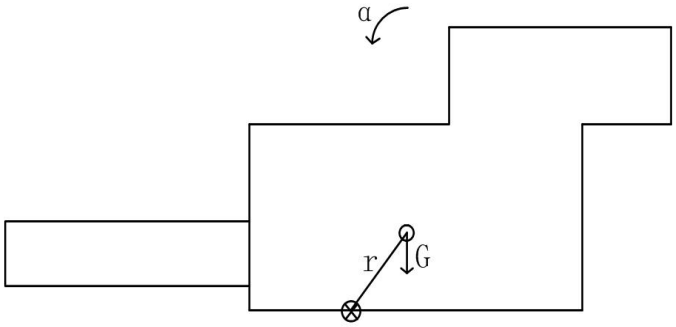


图 4-3 云台 pitch 轴的机械结构示意图

图 4-3 中，G 表示云台 pitch 轴部分的重心位置，G 代表重力，r 代表重心距离转轴的偏移距离, α表示云台转动的方向。根据图 1 所示，我们可以得到如下的方程：

$$\alpha = M\text{-}Gr\cos(\omega t) \tag{4-1}$$

式中，M 是电机的驱动力矩, ω 是 pitch 轴的转动角速度，t 是时间。

观察这个方程，我们可以发现这个是一个二阶的非线性方程，因此会对后面的线性控制系统(PID 控制器)产生影响。因此我们应该尽量使云台 pitch 轴部分的重心与转轴重合，即减小 r 到 0，以便提高后续的控制算法的效果。

云台的 yaw 轴部分同理，也应该尽可能的保证重心位于转轴上。

电控部分：

电控部分的要求是使云台具有较快的响应速度以及较好的平稳性，即在运动过程中不会出现抖动剧烈的现象。因此，根据我本科前三年的比赛学习经历，我决定通过电机的编码器和陀螺仪进行反馈，采取双环闭环 PID 控制器[17]。硬件上采用 DJI 的核心主控板，电机选择 6020 和 6623 直流无刷电机

视觉部分：

视觉部分的核心是物体的检测跟踪和测距技术，根据我们待检测的物品的颜色特征，我认为可以采取基于特征的识别方法，通过装甲板的物理特征对摄像头采集到的图像进行检索筛选，最终确定装甲板的位置。当确定装甲板板在摄像头平面里的位置后，根测距算法，检测到当前视野中的装甲板在机器人 自身坐标系里的三维位置。根据此三维位置，我们可以继而求得云台的转角，从而实现实时追踪的效果。

图 4-4 是视觉系统的流程图。

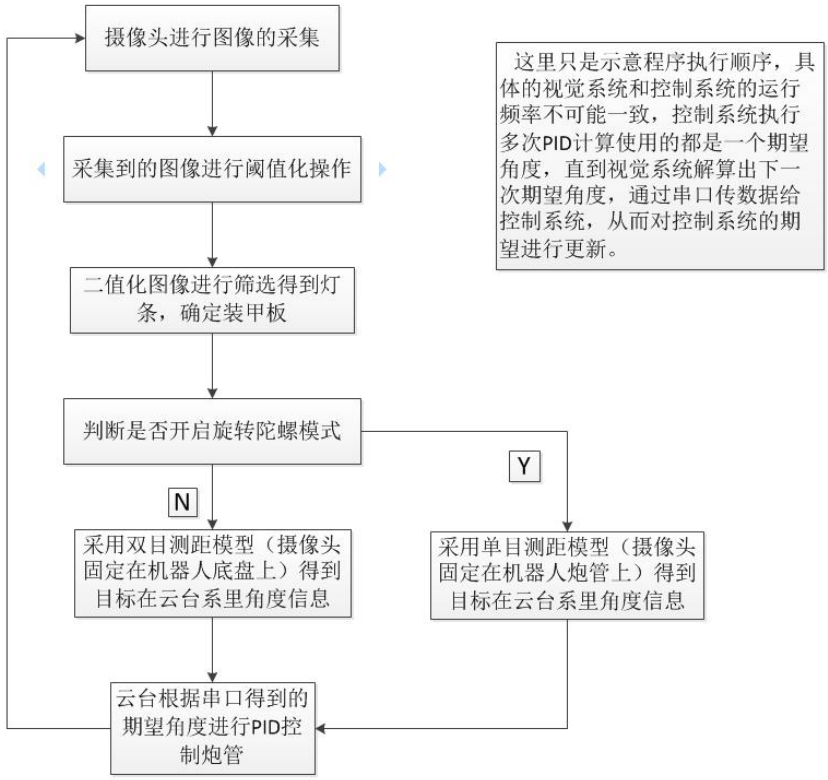


图 4-4 视觉系统流程图

机械系统，电控系统，视觉系统设计所需软件：

机械系统：

SolidWorks 软件设计云台机械机构，并根据实际使用的材料设置不同零件的质量属性，最终得到云台装配体的质心位置。

电控系统：

Keil5 软件编写 C 语言的控制程序，使用 jlink 仿真器和 jscope 仿真软件，根据陀螺仪反馈的角度数据曲线和电机电流曲线，在线整定 PID 参数值，使得电机达到最优的控制效果。

视觉系统：

英特尔微型计算机 Mini PCNUC7i7DNHE，并在其上运行 Ubuntu 操作系统。软件编程上，需要熟悉 Linux 操作命令，熟悉 C++编程语言，熟悉 OpenCV 开源的计算机视觉库，熟悉 Linux 下 C++语言的编译工具 cmake 的使用，熟悉编写 Linux 的脚本程序。

除此之外，第一阶段还需制作汇报 PPT，补充和整理材料，为第一阶段考核

做准备。

第二阶段的内容如下：

设计云台的机械机构，并检验重心位置。当机械系统搭建完成后，开始电控系统的调试。同时，学习 C++编程技巧和 OpenCV 函数使用手册，在 Ubuntu 系统上构建编译环境，最终完成程序的编写工作。

第 4 周：设计云台机械结构。

第 5 周：学习 C++编程知识。

第 6 周：学习 OpenCV 函数手册及算法原理。

第 7 周：搭建 Linux 开发环境，并熟悉其操作方式。

第 8 周：调试搭建好的机械云台。

第 9 周：开始着手编写视觉检测程序。

第 10 周：整理资料，制作汇报 PPT，为第二阶段考核做准备。

第三阶段的内容如下：

对视觉程序进行调试，尝试不同的特征筛选条件，并比较性能，设计出一个最好的检测方案。根据检测到的目标在图像的位置，距离解算出其在摄像头坐标系里的位置。

第 11 周：继续改进编写视觉检测程序。

第 12 周：编写测距程序。

第 13 周：对接视觉电控系统并翻译外文文献。

第 14~16 周：撰写论文，整理论文。

第四阶段的内容如下：

完成后续工作.

第 17 周：制作汇报 PPT。

第 18 周：准备终期汇报。

五、主要参考文献

[1]张建民. 机电一体化系统设计(第四版). 北京：高等教育出版社.

[2]郭润梅.机电一体化技术在机器人领域中的应用研究[J/OL].世界有色金属,2018(24):121+124[2019-03-12].

[3]王一鸣. 建设有国际竞争力的现代产业体系[N]. 学习时报,2019-03-04(002).

[4]丁蕾.智慧城市——西安智能公交站牌视觉应用研究[J].电脑知识与技术,2017,13(22):172-173+182.

[5]张泽鹏.基于大数据技术的交通视频监控分析[J].科技传播,2019,11(04):177-178.

[6]张帆.车载雷达检测图像自动识别追踪[J].物探化探计算技术,2018,40(04):487-494.

[7]蔡英凤,王海,张为公.基于视频的城市快速路交通流检测及车辆行为监控(英文)[J].Journal of Southeast University(English Edition),2011,27(02):164-168.

[8]Collins R，Lipton A，Kanade T，Fujiyoshi H，Duggins D，Tsin Y，Tolliver D，Enomoto N，Hasegawa O，Burr P，Wixson L．A System for Video Surveillance and Monitoring. VSAM final report. Carnegie Mellon University: Technical Report CMU.RI.TR 一 00.12.2000

[9]Collins R， Lipton A， Fujiyoshi H， Kanade T． Algorithms for Cooperative Multisensor Surveillance．Proceedings of the IEEE，2001，89(10)：1456-1477

[10]Coifman B，Beymer D，Mclauchlan P，Malik J．A Real-time Computer Vision System for Vehicle Tracking and Traffic Surveillance．1’ansportation Research Part C，1998，6(4)：271-288

[11]Tai J，Tsang S，Lin C，Song K．Real-time Image Tracking for Automatic Traffic Monitoring and Enforcement Application．Image and Vision Computing,2004，22(6)：485—501

[12]Haag M，Nagel H．Tracking of Complex Driving Maneuvers in Traffic Image Sequences．Image and Vision Computing，1998，16(8)：517-527

[13]Pai C，Tyan H，Liang Y，Liao H M，Chen S．Pedestrian Detection and Tracking at Crossroads．Pattern Recognition，2004，37(5)：1025-1034

[14]Masoud O，Papanikolopoulos N P．A Novel Method for Tracking and Counting Pedestrians in Real-time Using a Single Camera． IEEE Transactions on Vehicular Technology，2001，50(5)：1267-1278

[15]肖大伟,翟军勇.轮式移动机器人单目视觉的目标测距方法[J].计算机工程,2017,43(04):287-291.

[16]康新晨,杨卫,邓立齐.小型可移动平台双目定位方法研究[J].电视技术,2018,42(07):29-33.

[17]宋晓伟,樊战亭,田锐.直流电动机 PID 转速控制系统设计[J].科技创新与应用,2018(13):53-54.

六、指导教师意见

指导教师签字：

年　月　日

七、系级教学单位审核意见

开题考核分数：_____

考核组长签字：

年　月　日

附录 3（中期报告）

燕 山 大 学

本科毕业设计（论文）中期报告

课题名称：基于视觉的自动追踪控制系统设计

学　　院：机械工程学院

年级专业：2015 级机电 3 班

学生姓名：韩云海

指导教师：史小华

填写日期：04/22/2019

任务书中本阶段工作目标与任务要求

任务书中的主要内容如下：

课题的目标是设计一套基于视觉的机器人自动识别、自动追踪系统：

5、搭建机器人运动底盘和三维云台；

6、图像处理模块的应用；

7、图像处理算法实验比较和分析。

8、性能验证和展示。

任务书中的基本要求包括：

1.结构设计要正确、合理，具有可行性；

2.要考虑经济性；

3.考虑操作性；

4.按毕业设计要求整理设计资料。

在开题报告中设计的第二阶段的内容如下：

总体内容：

设计云台的机械机构，并检验重心位置。当机械系统搭建完成后，开始电控系统的调试。同时，学习 C++编程技巧和 OpenCV 函数使用手册，在 Ubuntu 系统上构建编译环境，最终完成程序的编写工作。

分周内容：

第 4 周：设计云台机械结构。

第 5 周：学习 C++编程知识。

第 6 周：学习 OpenCV 函数手册及算法原理。

第 7 周：搭建 Linux 开发环境，并熟悉其操作方式。

第 8 周：调试搭建好的机械云台。

第 9 周：开始着手编写视觉检测程序。

第 10 周：整理资料，制作汇报 PPT，为第二阶段考核做准备。

一、目前已完成任务情况

我将整体系统分为机械，电控，视觉三部分，因此我将从三部分分别阐述目前已经完成的情况。

（一）机械部分：机械部分主要包括云台结构的设计，具体零部件的采购以及最

终的组装工作。

云台结构设计的主要目的是方便电控PID算法加快云台响应速度且减少超调量以达到良好的控制效果。因此在机械结构的设计阶段，我和俱乐部中的机械组的成员达成一致认识，即云台的总质量应该越轻越好，这样可以减少系统的转动惯量，获得较快的响应速度。除以之外，因为 PID 算法适应于线性系统，因此我们需要尽量保证系统的质心位于 pitch 和 yaw 的转轴轴线上，这样可以保证云台在任何转角的时候由于重力 G 和 cos(ωt)的乘积产生的非线性项保持为 0。综上所述，我们综合考虑了机械设计的要求，保证能够便捷得安装和拆卸，各个零部件的质量分布，以减轻质量以及调整质心位置以及电线布线的要求以保证电线和信号线不会在运动过程中有脱落的风险。

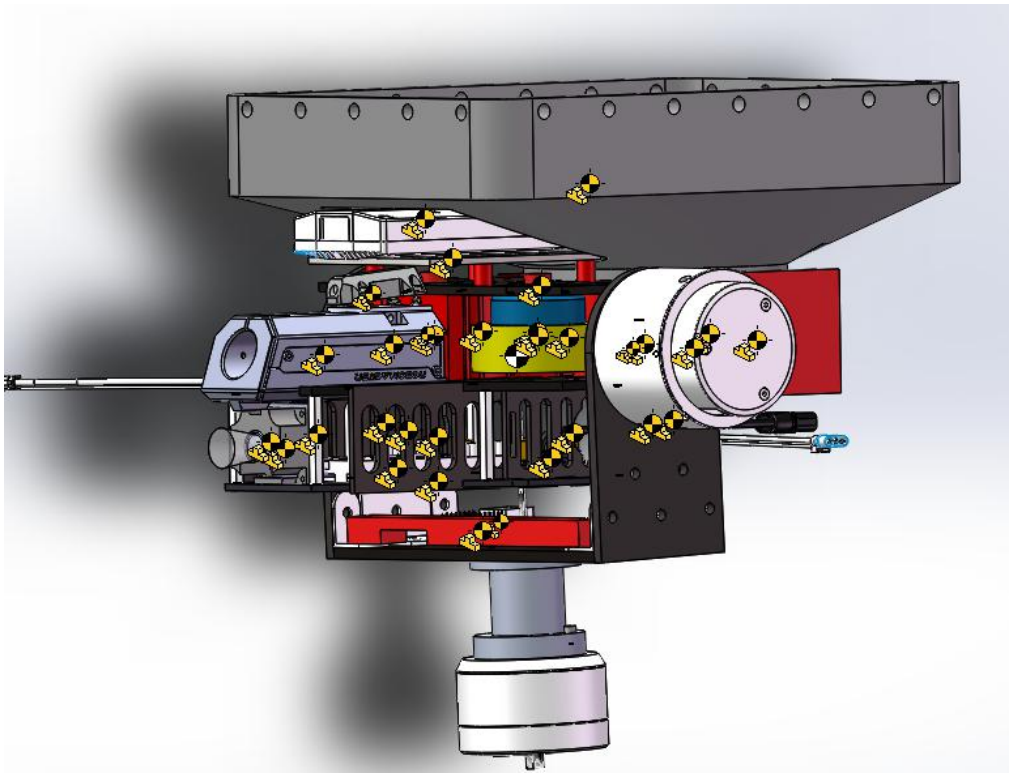


图 1-1 云台的机械结构

图 1-1 中的黄色标记代表的是各个主要零件的质心位置，省略了全部的螺栓和部分不重要的 3D 打印件。图 1-1 中的白色标记代表的是整体的质心位置，其由所有黄色标记件的质心位置和质量通过 SolidWorks 的质量属性自动求得。

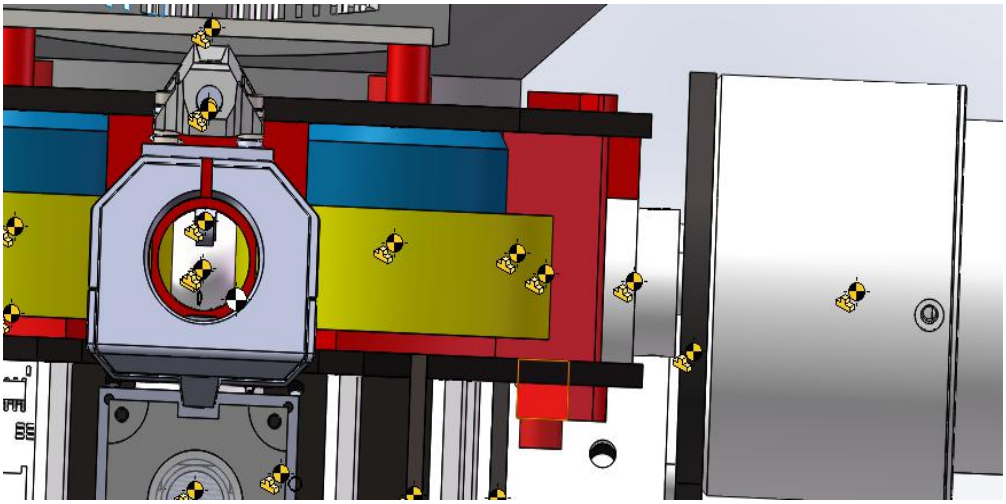


图 1-2 云台质心的分布

从图 1-2 我们可以看出，质心已经很接近 pitch 和 yaw 轴的轴线上，但是由于结构的问题以及具体材料质量的原因，其质心无法完全正好位于轴线上。从图 1-1 可以看出，由于云台只有右侧有 RM6623 电机，因此质心肯定会比较偏右侧。

另一方面，我们出于减少成本的目的，主要使用碳纤维板，3D 打印件作为主体的零件材料，只有在需要承受较大载荷以及需要较高的定位精度的地方使用机械加工件，包括图 1-1 中最下面的“工”型加工件(需要承受整个云台的重量)和 pitch 轴电机与云台的连接件(需要很高的定位精度)。

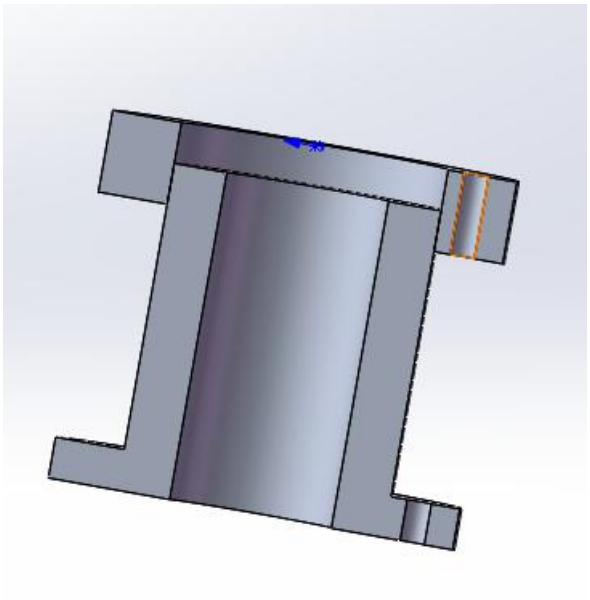


图 1-3“工”型加工件

图 1-3 中表示的是“工”型加工件，中间空心的目的是安放导电滑环。我们

使用导电滑环的目的是当云台 360°旋转的时候，电线可以通过导电滑环到达云台，而不会缠绕在云台上。关于导电滑环的内容我放在电控部分加以阐述。加工件右侧的上下螺纹通孔的目的是将加工件连接到下方电机和上方云台。

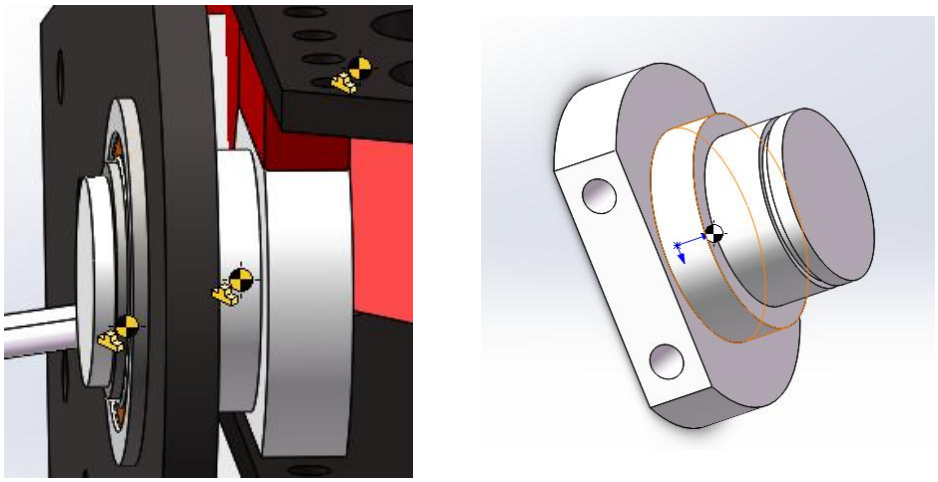


图 1-4 pitch 轴电机和云台的连接件

图 1-4 中表示的是另外一种需要使用加工件的地方，因为这里需要较高的定位精度，所以采用的是机械加工件。

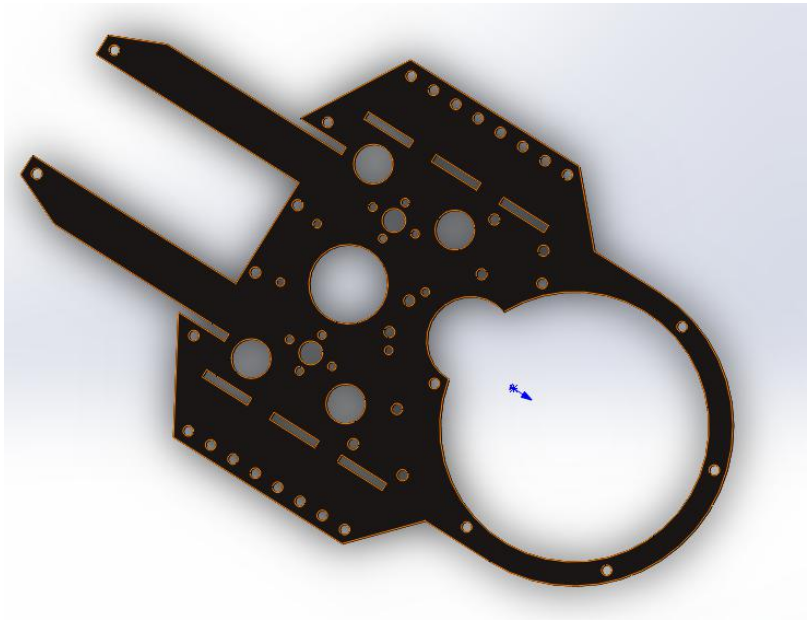


图 1-5 一块碳纤维板

图 1-5 表示的是其中一块碳纤维板，我们在考虑碳纤维板的结构的时候，同时考虑了安装要求，电线布线要求和减重要求。如图 1-5 所示，小的圆孔是用来连接螺钉(安装零件或者固定传感器)，大中型的圆孔一方面起到减重的目的，一方面可以让扎带从其中穿过，将电线固定在板上，电线有单片机

供电线，电调信号线和供电线，直流无刷电机的供电线和信号线，陀螺仪的信号线，无线蓝牙信号线，图传信号线和 RM 比赛裁判系统的信号线。

最后，我列出云台上全部的物品，包括电控视觉系统部件以及机械零部件。

机械零件包括：

1. 3D 打印件，图一中的红色零件，其中炮管是用光敏树脂打印的，其余部分是由 PLA 材料打印的，包括弹仓，拨弹盘等。 花费：500 左右
2. 碳纤维板，图一中的黑色零件，构成了云台的外部框架。 花费：400 左右
3. 加工件，一共 3 个，上文中已经列举过了。 花费：400 左右
4. 其余各类标准件，如铜柱，螺钉螺母，扎带，角件，胶轮等。

电控视觉系统包括：

1. 摄像头， 花费：195.
2. DJI 主控板， 花费：400 左右
3. 电源转接板(将电池的电源线路分线到主控板和电调) 花费：100 左右
4. 电调和电机，包括 yaw 轴电机 6620，pitch 轴电机 6623，两个摩擦轮电机 3510，拨盘电机 3510 花费：1600 左右
5. 陀螺仪， 花费：150 左右
6. RM 比赛的图传和裁判系统
7. 其余部分，包括天线接收器，蓝牙模块，各类接口，电线等， 花费：200 元左右

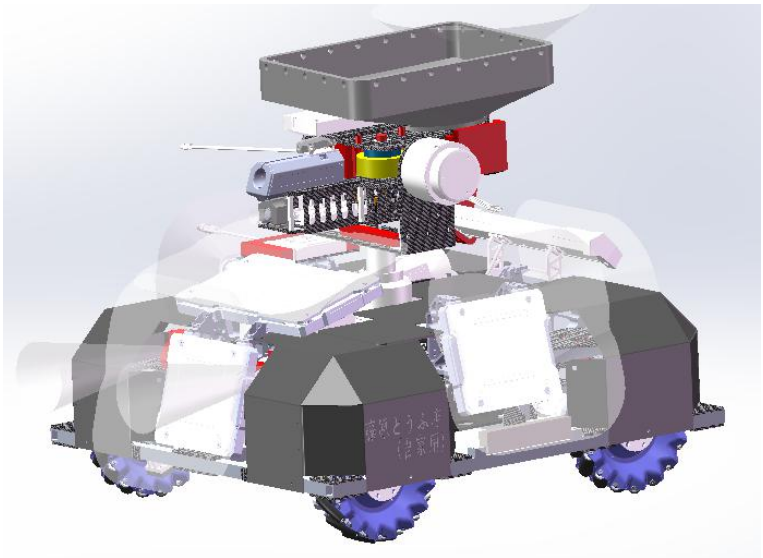
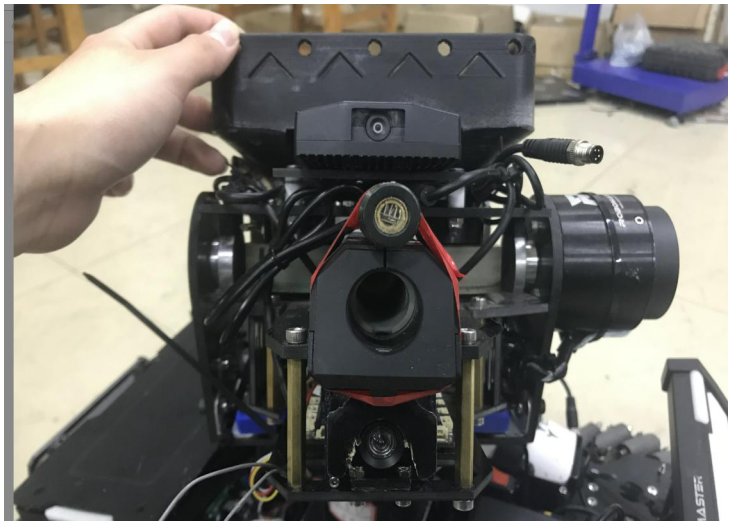


图 1-6 机器人整体三维模型




ROBOMASTER


ROBOMASTER

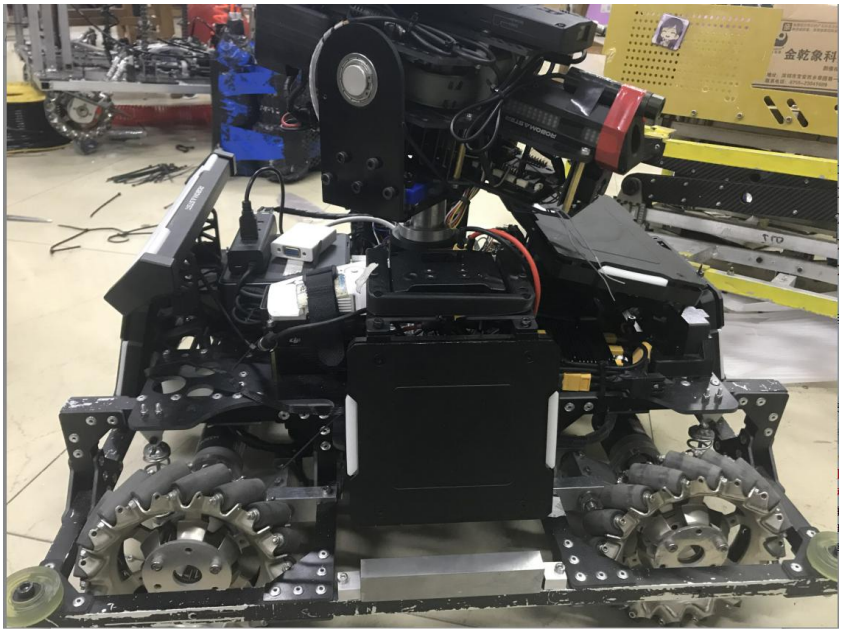
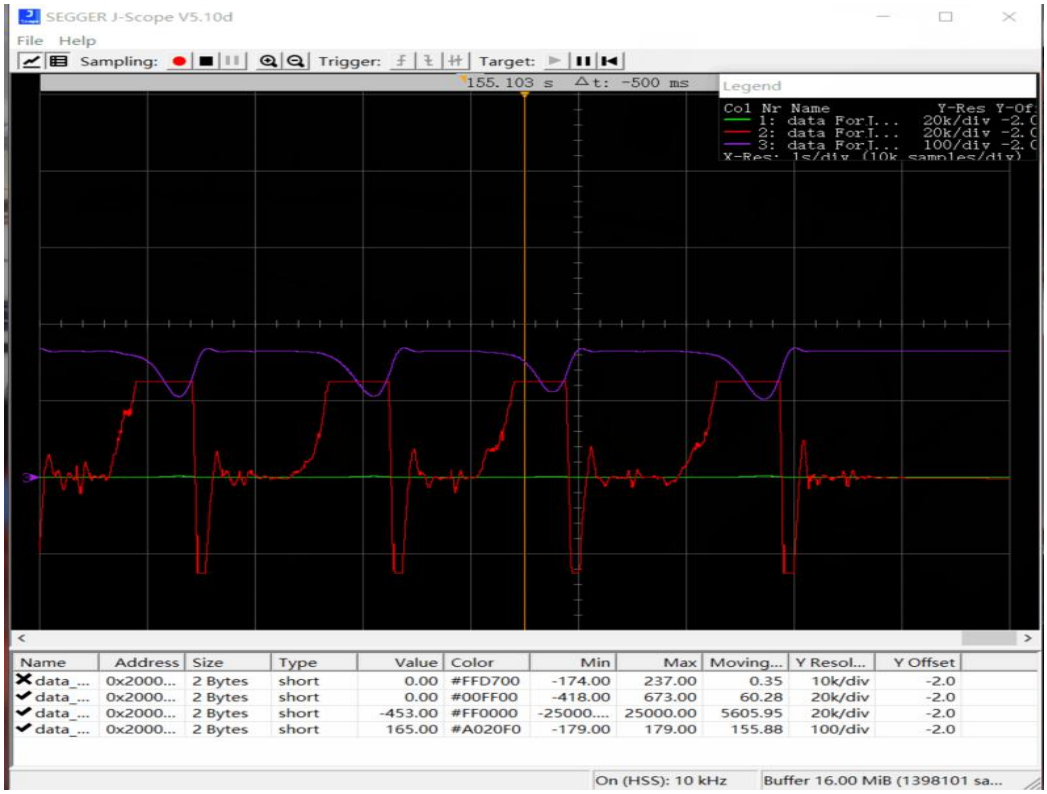


图 1-7 机器人实物图

（二）电控部分：电控系统主要包括云台 pitch,yaw 轴的 PID 控制，摩擦轮和拨盘电机 PID 控制和云台电路布线。

a.云台的控制主要使用的是双环 PID 进行反馈控制。对于云台 yaw 轴电机，内环速度环使用的反馈信号是固定在云台上的陀螺仪的角速度，外环位置环使用的反馈信号是 yaw 轴电机 6620 绝对式编码器的数值。通过调节内外环 PID 参数，可以使 yaw 轴得到良好的控制效果，具体表现为，能以较快的响应速度达到设定的 yaw 轴位置，且运动过程中没有抖动，电机脱力(由于 PID 设置死区的原因)等现象。对于 pitch 轴同理，内环使用的反馈信号是板载陀螺仪反馈的角速度，外环使用的是 pitch 轴电机 6623 绝对式编码器的数值。

图 1-8 是调试的结果图，使用的是 JLink 工具和 JScope 软件在线调试，能通过数据曲线判断当前参数的控制效果，在 Keil5 中通过 debug 模式在线调整参数再比较曲线。



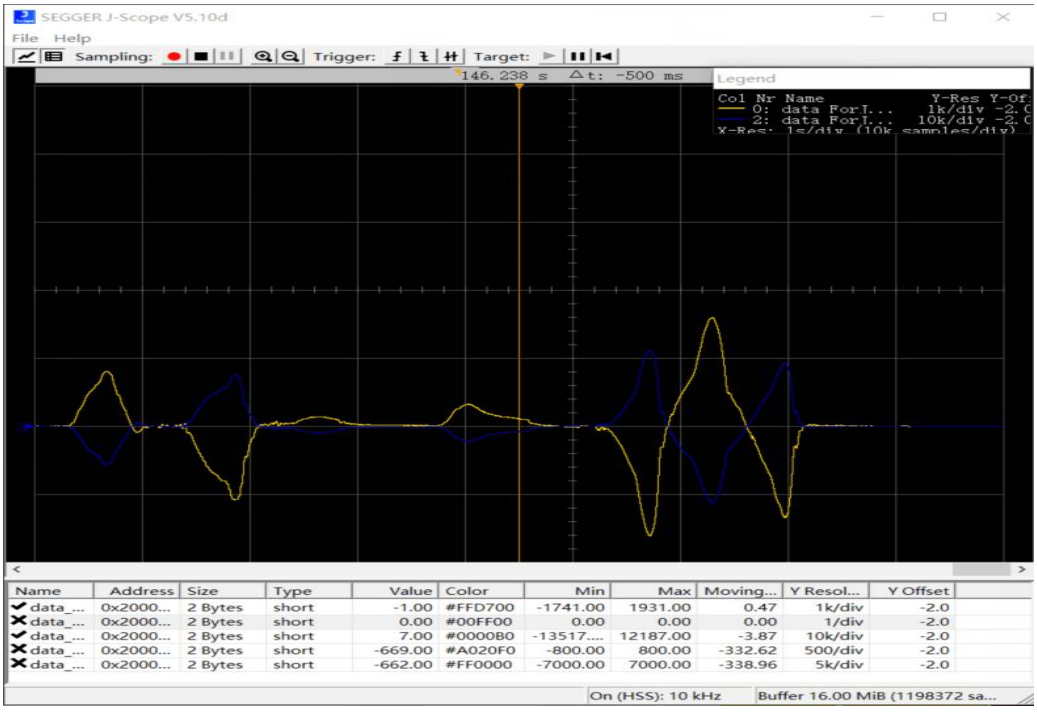


图 1-8 JScope 软件调试效果图

如图 1-8 中所示，图 1-8a 中红线表示的是速度环 PID 计算后的输出，紫线表示的是位置环的实际值。当紫线趋于稳定的时候，即位置保持不变的时候，速度环的输出很小接近于 0。当紫线出现下降趋势的时候，即位置开始改变的时候，速度环的本质作用是保持运动平稳，增加系统的阻尼，因此当云台在外力的作用下产生速度的时候，速度环的输出就会增加，以抵消这个外界产生的速度。图 1-8b 的曲线结果也与此推断一致，在图 1-8b 中，黄线是实际速度，蓝线是只有比例项赋值的 PID 输出。我们可以看到当速度产生后，PID 速度环的输出是逆速度方向的。通过上图的曲线，我们一方面可以通过曲线比较参数性能，另一方面可以通过底下变量状态栏的具体数据判断程序执行是否正确。

摩擦轮和拨盘电机的控制与发射要求有关，摩擦轮转速与射出子弹的初速度有关，拨盘电机的转速与单位时间内发射子弹的数量有关。我们对摩擦轮进行闭环控制，减少其转速波动，稳定在我们设定的转速值上。同理，我们对拨盘电机的转速进行闭环控制，以保证射频稳定，不会超过机器人热量上限。

总结：云台电机和摩擦轮拨盘电机的控制都是在 Keil5 软件上通过 C 语言编写程序。

b.云台电路布线主要考虑的难点有两个，其一是由于云台 360°运动的要求，电线不能直接从底盘引出到云台上，必须经过导电滑环；其二是电线在云台上本身的布置要求，即电线不应在任何情况下有松动脱落的风险。

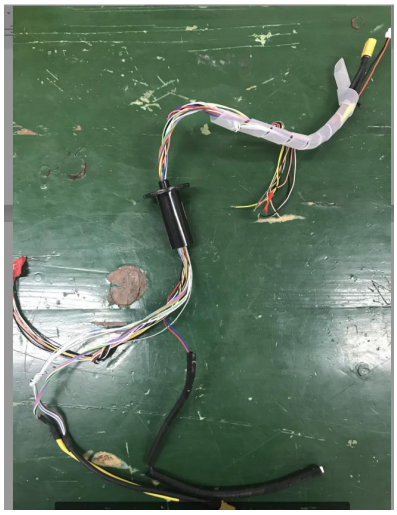


图 1-9 导电滑环

图 1-9 所示是一个 24 路 2A 的导线滑环。导电滑环参数 24 路指的是一共从滑环内部穿过了 24 跟导线，2A 是每根导线上可以承受的最大电流。这些线路的分配情况如下：

1.电源分配了 12 路，正负极各 6 路

由于云台上共有 5 个电机，需要大电流才能满足同时驱动的要求，因此需要分配 12 路导线承受较大的峰值电流。

2.5 路分配给了摄像头

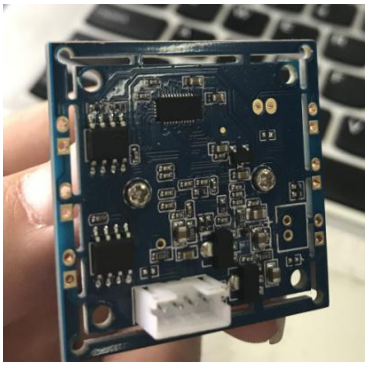


图 1-10 摄像头

由图 1-10 所示，摄像头 PCB 背部的的引脚一共有 4 个，因此一开始我以为只需要4路信号线就可以将云台上的摄像头引入到底盘上的微型计算机MINI PC上。然而，在实际操作中发现，如果只接入摄像头的这四根线，PC 会无法检测到摄像头设备。

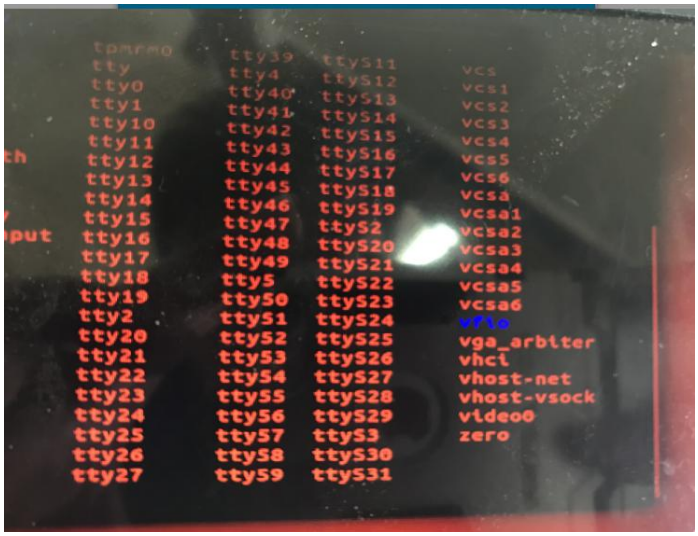


图 1-11 PC 设备资源界面

查阅资料后得知，许多 USB 信号线其实是有 5 根线的，第 5 根线是一层金属箔层，围绕在内部四根线外面，其目的是减少信号干扰。

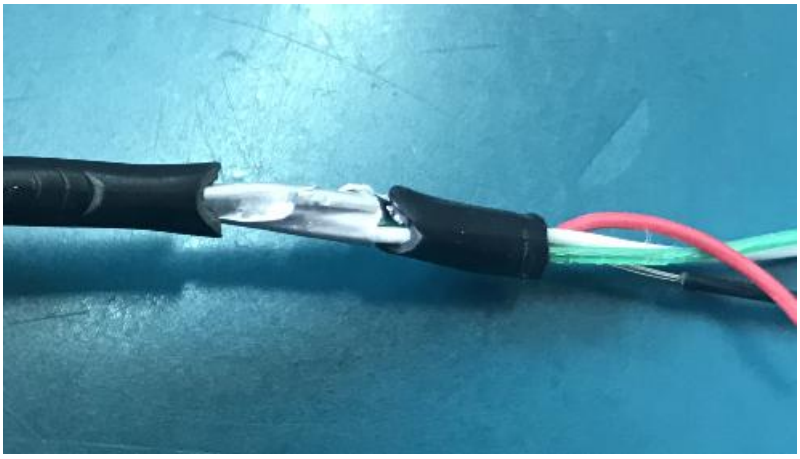


图 1-12 USB 数据线

我们首先采取的是双绞线的方式将四根线缠绕在一起。双绞线是工程上常用的布线方式，能有效减少信号之间的串扰。此时，设备资源上出现了这个摄像头，但是当我们实际运行程序，读取摄像头的数据的时候，发现数据干扰严重，画面呈现“雪花状”，且伴随摄像头传过来的错误提示“数据干扰严重”，运行一段时间过后，程序就报错终止运行了。最后我们在导电滑环中加上了第 5 根导线，连接的是 USB 两侧的金属箔层。这时，我们在运行程序的时候，画面清晰且没有错误提示。除此之外，市面上也有专门针对 USB 信号线的导电滑环。

图 1-13 USB 线导电滑环

3.4 路分配裁判系统

RM 比赛程中，位于云台上的主控板需要读取机器人底盘上的裁判系统的数据。

4.2 路分配给 CAN 总线

机器人底盘的 4 个麦轮电机需要接收云台上主控板的 CAN 控制信号并反馈自身转速信息，除此之外，主控板还需要通过串口接收底盘微机传来的视觉数据。而由于导电滑环的线路有限，因此需要将这些数据整合一起。故我们在底盘上又额外增加了一块单片机，其通过串口接收从微机传来的视觉数据，并且外接一块 CAN 总线模块，负责将视觉数据通过 CAN 数据的方式发送给主控板同时与主控板双向传输与底盘电机有关的控制信号。

5.1 路空余

除了导电滑环以外，云台自身的布线要求的宗旨是保证导线连接处稳固，杜绝脱落的可能性并且保持导线紧凑以保证在云台的运动过程中不会与导线干涉。通过机械部分碳板上预留的豁口以及螺孔，我们通过扎带螺钉等工具将导线固定在云台上。对于多根方向一致的导线，我们用绕线管将导线缠绕在一起。

图 1-14 云台布线

（三）视觉部分：视觉部分主要包括通过以下步骤：

1. 摄像头进行图像的采集
2. 对采集到的图像进行阈值化操作
3. 二值化图像进行筛选得到灯条，确定装甲板灯条对应的图像像素位置
4. 采用单目测距算法(摄像头固定在机器人炮管上)得到目标在云台系里的角度信息
5. 云台根据串口得到的期望角度进行 PID 计算控制炮管

这里的步骤是指视觉算法每一帧中内的步骤，具体视觉系统的帧率和控制系统的控制频率不可能一致，控制系统的控制频率稳定在 100HZ，而视觉系统的帧率稳定在 50HZ，因此粗略来说，控制系统执行两次 PID 计算使用的都是都是一个期望角度，直到视觉系统解算出下一次期望角度，通过串口传数据给控制系统，从而对控制系统的期望进行更新。

视觉系统的硬件和开发平台包括：

硬件平台:Intel Mini PCNUC7i7DNHE,威刚 128G 的硬盘 sx6000,三星的 8G 内存条和 USB 读取的摄像头。

开发平台:Ubuntu 操作系统以及 cmake 编译工具，C++编程软件和 OpenCV 开源计算机视觉库。

图 1-15 硬件平台(不包括摄像头)

视觉部分所涉及到的内容比较多，且大部分甚至全部都是我在课外时间自学学到的如图 1-16 中所示，包括 C++编程技巧，OpenCV 使用技巧，Linux 编程技巧，cmake 编译方法。

图 1-16 学习图片

视觉部分的核心是物体的检测跟踪和测距技术，根据我们待检测的物品的颜色特征，我认为可以采取基于特征的识别方法，通过装甲板的物理特征对摄像头采集到的图像进行检索筛选，最终确定装甲板的位置。当确定装甲板板在摄像头平面里的位置后，通过测距算法，检测到当前视野中的装甲板在机器人云台自身坐标系里的三维位置。根据此三维位置，我们可以继而求得云台的转角，从而实现实时追踪的效果。

具体的单目测距算法的原理由于篇幅原因我不在这里详细说明。单目测距可以类比为人只睁开一只眼睛的情况。此时由于三角形相似的问题，单目测距无法直接得到物体的深度信息(当人只睁开一只眼睛的时候，很难判断出物体距离自己的实际位置)。只有当单目测距算法使用到复数个匹配的特征点以及各个特征点在自身系下的坐标，我们才可以通过公式计算出那个坐标系距离当前摄像头坐标系的 6 维位姿信息(三个转角三个距离)。

目前的视觉算法在装甲板静止或者缓慢移动的时候可以保证接近于 95%的射击命中率，但是当装甲板运动较快的时候，由于摄像头的帧率限制，以及子弹的飞行时间造成的滞后，射击命中率较低，子弹多半射中在装甲板的两侧。这也是下一个阶段待解决的主要问题。

二、存在的问题和拟解决方法

机械系统和电控系统已经成熟，不存在问题了，现在的主要问题有两点：

1. 视觉系统在装甲板运动过程中的弹道滞后问题。这个问题需要通过运动预测模型加以补偿。

2. 微机在上电以后自动运行程序无需人为通过显示屏，鼠标和键盘进行操作。这个问题需要通过编写 Linux 上电自启动的 shell 脚本解决。

指导教师签字：

年　月　日

四、系级教学单位审核意见：

中期考核分数：_____

考核组长签字：

年　月　日

附录 4（外文文献翻译）

Vehicle Pose and Shape Estimation through Multiple Monocular Vision

Abstract

In this paper, we present a method to estimatea vehicle's pose and shape from off-board multi-view images. These images are taken from monocular cameras with small overlaps. We utilize state-of-the-art Convolutional Neural Networks (CNNs) to extract vehicle's semantic keypoints and introduce a Cross Projection Optimization (CPO) method to estimate the 3D pose. During the iterative CPO process, an adaptive shape adjustment method named Hierarchical Wireframe Constraint (HWC) is implemented to estimate the shape. Our approach is evaluated under both simulated and real-world scenes for performance verification. It's shown that our algorithm outperforms other existing monocular and stereo methods for vehicles'pose and shape estimation. This approach provides a new and robust solution for off-board visual vehicle localization and tracking, which can be applied to massive surveillance camera networks for intelligent transportation.

INTRODUCTION

Most recently, road scene understanding is well studied for improving the perception ability of intelligent transportation. Meanwhile, 3D pose estimation for objects becomes a hot research topic, owing to its significance to the field of computer vision and robotics. These factors inspire us to focus on pose and shape estimation of vehicles to improve the perception ability of intelligent transportation.

Despite sensors like LiDAR, depth camera and stereo camera have been used for a

long time, their application scopes are constrained due to high cost and other limitations. Therefore, to estimate vehicle's pose information, more and more works concentrate on monocular visual estimation methods. These methods have potential to be applied to the massive surveillance camera network in real world.

In fact, mobile robots usually conduct on-board methods called simultaneous localization and mapping (SLAM) [8]. In contrast, off-board visual methods can also be considered for 3D pose estimation tasks. Considering that off-board methods have the advantage of possessing a better Field-of View (FoV), and that many of the latest deep-learning based technics are developed based on off-board vision, there is a huge potential to use them for vehicle 3D pose estimation.

For recent vision-based 3D pose estimation research, deep learning tools are widely used. Keypoints of objects are defined and detected to aid with the pose estimation [20]. Those CNN methods provide a new way to solve this kind of problem. But in most cases, single image is processed for tackling estimation task and this suffers from several drawbacks. As a remedy, it can be greatly improved if multiple images from different views can be used together in the scene of traffic monitoring.

Based on above background, we propose an approach using multiple off-board cameras (two at least) with small overlaps to obtain 3D pose and shape of vehicles. An example of our approach is shown in Fig. 1. In comparison to methods with bounding box annotation [4], our approach utilizes a wireframe model to describe vehicle's 3D pose and shape information.

The whole algorithm is divided into two stages as shown in Fig. 2. First, multiple images taken from cameras of different views (two images for simplification) are fed into a coarse-to-fine CNN that is trained for vehicle semantic keypoints detection specifically. After CNN processing, two sets of keypoints are obtained as outputs. Second, in optimization stage, CPO method projects a general 3D vehicle model onto each image with camera intrinsic and extrinsic parameters, and iteratively minimizes projection errors. It's worth mentioning that no prior knowledge about the target vehicle is required in this approach. Vehicle's shape estimation starts from a general wireframe model and is

adjusted with HWC method.

II. RELATED WORKS

A. Convolutional Neural Network

The past few decades have witnessed the development of neural networks, especially in the field of complex feature detection. [2], [3] are state-of-the-art works of object detection. Ren et al. [2] provided a real-time region proposal network to detect multiple objects on the 2D image. Josephet al. [3] proposed a detection method which can achieve the classification of 9,000 objects. All these detection methods help localization algorithm to focus mainly on their target.

For most detection and estimation tasks, CNN is used to recognize complex and high-level features, which cannot be sufficiently tackled by conventional vision methods. Lately, there are already some end-to-end methods for pose estimation. [10], [4] utilized video or multi-channel information obtained from on-board device to localize vehicles. [11] and [9] directly trained a CNN with a single image and 3D

landmarks, but complex and large 3D datasets require much time and human resource for annotating.

In the recent study, a stacked hourglass framework [1] is proposed to detect semantic keypoints on the bodies of human beings. Owing to its coarse-to-fine architecture, features can be detected on multiple resolutions, leading to high accuracy results. Compared to traditional random forests method like [14], CNN outputs more accurate keypoints. Our approach follows this direction and takes advantages of these works.

B. Single Image Vehicle Localization

As for robotics and intelligent transportation, vehicle localization is always at the cutting edge [21]. Both on-board and off-board methods have representative works.

Most of the off-board works combine a single image with complex vehicle models. Zhu et al. [27] used constrained discriminative parts and pre-defined wireframe models to estimate the 3D pose. [17] proposed a similarity measure method with off-board camera,

and a top-down perception approach is proposed in [28]. Works mentioned above all require complex accurate car models, which is usually un┐available. Besides, their feature detecting methods are not robust enough for arbitrary scene and viewpoint.

Some latest works [6], [7] combined semantic keypoints with simple models. They achieve large advantages in pose estimation task with the help of CNN. These approaches outperform most existing methods. However, methods withsingle image have a problem with translation error and model scale. Even if [6], [7] have shape adjustment in their process, this problem remains unsolved. According to the principle of 3D projection, depth of object depends on the scale of the model. Whether accurate pose is provided or not, the

translation error increases when disproportionate model is used. Fig. 3. shows a simple demonstrative experiment. We choose a wireframe which is smaller than groundtruth, and the left two columns show the results with only one camera. Disproportionate model generates right orientation but wrong translation and shape. As a contrast, the last column shows that two projects are both right.

Besides the defect mentioned above, the robustness of the keypoints detection is poor when vehicle occultation (by trees, walls, or other vehicles) occurs. To some extent, multiple camera approach can help with these limits.

C. Stereoscopic and Multiple Camera System

The most similar algorithm with ours is stereoscopic algorithm. Stereo cameras usually consist two parallel cam┐eras with small baseline or large overlap. Suppose using

stereoscopic algorithm in our scenario. After getting two sets of keypoints of two images, we can calculate the 3D position of every keypoints with intrinsic and extrinsic parameters of camera. Then, we can connect those keypoints into a wireframe. But stereoscopic algorithm requires high precision, which means errors of keypoint from CNN lead to a large shift in 3D space. Still further, the shape of the vehicle will be asymmetrical. More results will be displayed and discussed at the experiment part.

Most multi-camera systems are applied to the field of human pose estimation, such as [12], [13], [14]. Pavlakos etal. [14] utilized random forest to classify each pixel in each image, [12] recovered a volumetric prediction from multiple images. The standard

principle of human pose estimation is using 3D pictorial structure, which has been proved to be effective. Our method is inspired by these approaches to combine image features with a deformable model.

To the best of our knowledge, there are few works of vehicle localization using multiple cameras. Chen etal. [4] used bird view LiDAR data and front image to localize surrounding vehicles for the purpose of automatic driving, but only rough 3D bounding box results are given in this approach. [15] and [16] improved SLAM algorithm by integrating multi-view cameras and shape information. They outperformed some traditional SLAM methods. Calibrated stereo camera with small baseline can be used to reconstruct depth of the scene like [18] and [5], but the narrow overlap┐ping region is not wide enough for accurate vehicle local┐ization. Although some defects still exist, works mentioned above prove the superiority of multi-view approaches and guide us to explore more possibilities in this direction.

D. Our Contribution

Considering all the pros and cons of existing single-image and multi-image methods, we propose a new framework for vehicle 3D localization. Our contributions are as follow:

• We take advantage of hourglass architecture CNN to extract 2D feature from 2D annotation dataset. Training data and training process are much easier to implement.

• Our methods can be applied to small overlap condition, which is more consistent with traffic monitoring camera system.

• Our multi-camera method has better performance over mono-camera method in aspects of precision and ro┐bustness of sheltered environments.

• Inspired by deformable methods, we propose HWC to adaptively estimate vehicle's shape.

基于多个单目视觉系统的汽车姿态和形状估计

摘　要

这篇论文论述了一种通过多个不同视角的图像估计汽车姿态和形状的方法。这些图像是从多个有重叠视角的单目相机得到的。本文使用了最新的卷积神经网咯（CNN）提取汽车的特征角点并且引入了交叉投影优化方法(CPO)估计 3D 位姿。在 CPO 迭代过程中，本文运用了一种自适应的形状调整方法，层次网格约束(HWC)估计汽车的形状。通过在模拟数据和现实场景的应用评估下，本文提出的方法的有效性得到了证实。结论表明本文提出的算法比其他任何已经存在单目和双目算法估计汽车姿态和形状的效果好。本文对基于固定摄像头的汽车定位和追踪问题提出了一种新的鲁棒的方法，此方法可以运用于大规模的监控摄像头网络，以实现智能交通的解决方案。

导　言

最近，道路场景感知领域上的进步提高了智能交通系统的感知能力。同时，由于目标 3D 位姿估计对于计算机视觉和机器人领域的重要性，其逐渐成为了一个火热的研究领域。这些因素促使我们致力于汽车的位姿和形状估计，以提高智能交通系统的感知能力。

尽管许多传感器，如雷达，深度相机，双目相机已经在这一领域使用了很长一段时间，但是由于其高额花销和其他的一些限制，他们的应用范围仍然很受限制。因此，越来越多的研究人员开始把精力放在如何通过单目视觉系统估计汽车的位姿信息。这些成果日后有可能被应用于现实世界中的大规模监控线机网络。

事实上，移动机器人通常使用基于自身上安装的设备的方法来估计位姿，此方法叫做同步定位和建图(SLAM)。相对的，基于非自身搭载的设备的视觉方法同样可以用于 3D 位姿估计。考虑到此种方式下有较大视场角的优势以及当前许多最前沿的深度学习的方法使用的是此种设备得到的图像，本文认为此种方法在 3D 姿态估计有巨大的潜力。

深度学习的方法在当下的基于视觉的 3D 姿态估计中使用广泛。定义和识别目标的特征点的方法有助于位姿估计。这些 CNN 算法提供了一些新的方法来解决此类

问题。但是在最多的应用场合下，处理单个图像得到的追踪结果有许多的缺点。作为一种改进措施，同时使用交通监控下多视角拍摄到图片可以极大的改善结果。

基于上述提到的北京，本文提出了一种使用多视角且有重叠视角的相机的方法(最少两个)以获得汽车的 3D 姿态和形状。图 1 表明了一个例子。与边框回归标注算法对比，本人提出的方法使用了网格模型来描述汽车的 3D 姿态和形状信息。

整个算法可以被分成两个步骤，如图 2 中所示。首先，多张来自不同视角相机的图片(简便起见假设两张)被喂入 一个特别经过特征点提取训练的粗糙-平滑 的 CNN 网络。CNN 后，得到两组特征点作为网络输出，进入下一个步骤。其次，在优化步骤中，CPO 通过相机的内外参数矩阵将一个一般的 3D 汽车模型投影到各自图像系中，并且迭代减小重投影误差。这里额外说明的是，在此方法中无需任何有关目标汽车的先验知识。通过一般的网格模型可以初步得到汽车的形状估计，然后使用 HWC 方法调整估计值。

相关工作

A. CNN 神经网络

过去的几十年见证了神经网络的快速发展，特别是其在复杂特征提取领域里的巨大作用。[2][3]是现在最前沿的目标检测算法.Ren 在[2]中提出了在 2D 图像平面上的一种实时的区域性多目标检测的网络。Joseph 在[3]中提出了一种可以区分 9000 种不同物品的检测算法。所有的这些方法可以提高定位算法在识别其目标上的准确度。

对于大多数的检测和估计任务来说，CNN 被用来识别复杂的特征，这些特征不能有效地被传统的视觉算法处理。后来，研究人员发现了一些端对端的姿态评估算法。[10],[4]使用了录像或者是从车载设备获得的多通道的图像信息来定位车辆位置。[11]和[9]通过单通道图像和 3D 路标点直接训练了 CNN 网络，但是大量的复杂 3D 数据库需要大量的计算时间和标注时间。

最近的研究中，有人关于提取人类身体的特征点提出了堆叠沙漏网络。由于网络自身的 粗糙-平滑的运算结构特点，可以设置不同分辨率得到特征，从而得到很精确的结果。与传统的随机森林方法对比，CNN 可以得到更精确的特征点。本文的工作将基于这些现有的理论研究从而有效利用其优点。

B. 单个图像的汽车定位

对于机器人和智能交通系统来说，车辆定位总是处在总前沿的位置。车载相机和固定相机的方法都有各自代表性的成果问世。

大多数的固定相机的方法融合了单个图像信息和复杂的车辆模型。[17]提出了使用固定相机的相似性检测方法，以及[28]提出了一种自上而下的感知算法。上述提到的算法都需要复杂且准确的车辆模型，而这些模型在实际中通常是很难获得的。不仅如此，上述特征提取算法不能保证在任意的场景和角度都能取得良好的结果。

一些最新的成果[6],[7]结合了特征点和简单的模型。他们使用了 CNN 在姿态估计方面上取得了很好的效果。这些算法相比较现金其余算法而言，有更大的优势。然而，基于单个图像的算法总是有关于图像尺度和平移误差的问题。即使是[6],[7]中提出的方法仍然在处理过程中有形状的调整，说明这个问题还是没有解决。根据单目 3D 投影的原理，目标的深度信息取决于系统的尺度。无论得到的位姿数据是否准确，当模型不合比例的时候总会出现平移误差。图 3 是一个简单的示例。本文中使用了一个比地面小一些的网格模型。图中的左侧两列是在单目算法下得到的投影结果。从图中可以看到，在不合比例的模型下虽然能产生正确的投影朝向但是位置和形状上有些错误。与之对比的是，最后的一列表示的两个投影都是正确的。

除了上述提到的缺点，当车辆被树，墙壁或者其他车辆遮挡的时候，车辆特征点提取算法的鲁棒性比较差。在一定程度上，多视角图像处理算法可以克服这些缺点。

C. 立体视觉和多视角相机系统

立体视觉算法是和本文提出的算法最相似的算法。双目相机系统通常由两个平行放置的有较小的基距或者较大重叠的相机组成。假设在现在的实际环境下使用双目相机，当得到两组图片各自的特征点后，我们可以通过相机的内外参数矩阵得到每个特征点的 3D 位姿。接着我们可以把这些特征点在一个网格图中连接起来。但是立体视觉算法需要很高的准确度，这就意味着 CNN 算法得到的特征点若是有错误则会导致计算出的 3D 位姿数据有很大的偏差。更进一步地说，这可能会导致估计出来的车辆的形状不再对称。本文会在实验数据部分阐述展示和讨论更多结果。

绝大多数的多视角相机系统在人体姿态估计领域得到了广泛的应用，例如[12],[13],[14]。Pavlakos 在[14]中在每幅图中使用随机森林算法对每个像素点实现了分类。人体姿态估计的标准做法是使用 3D 图片的结构特点，此做法已经被证实是十分有效的。本文提出的方法正是基于上述提到的方法，对一个变形的模型进行了特征的提取合并。

在本文作者的观点里，基于多相机的车辆定位研究还没有很多人开展过。Chen 在[4]中使用了鸟瞰雷达相机和前置相机来定位附近的车辆以实现自动驾驶

的目的，然而此方法只呈现了比较粗糙的 3D 边框回归结果。[15]和[16]通过融合

多视角相机的图像和形状信息改进了 SLAM 算法，使其比传统的 SLAM 算法有更高的性能。标定后的双目相机可以通过较小的基距重建[18]和[5]中的场景的深度信息，但是狭窄的重叠区域并不能实现准确的大范围车辆定位。尽管当下还存在不少缺陷，上述提到的工作证明了多视角图像算法的优越性并且可以帮助后面的研究者在这个方向有得到更多的发现。

D. 本文的贡献

本人讨论了所有现在使用的单目和多目算法的优缺点，继而提出了一种新的用于车辆 3D 定位的算法框架。本文的贡献可以归纳为如下 4 点：

1.本文利用了“沙漏”结构(hourglass archetecture)的 CNN 神经网络从标注好的 2D 数据集中提取 2D 特征。此方法的训练数据方便获得且训练过程较简单。

2. 本文提出的算法可以适用于多相机有小重叠的情况下，且这种情况在交通监控系统中十分常见。

3.本文提出的多相机算法比单相机算法在精度和庇护环境下的鲁棒性上都有更好的效果。

4.本文受到变形算法的启发，提出了 HWC 方法自适应判断车辆的形状。