东南大学

《智能机器人系统综合设计》

过程报告

(实现与评估阶段)

基于机器人触觉和摇晃动作的液量动态感知

姓　　名： 蔡雨洋　**学　　号：** 08120141　**评定成绩：** _____

专　　业： 自动化　**方　　向：** 机器人

学　　期： 2023/9-2024/1　**报告时间：** 2023 年 11 月 5 日

审阅教师： 钱堃

一、简介（Introduction）

报告人：蔡雨洋

团队成员：

1. **陈志坤：**参与国家级 srtp 且作为主力发表论文；负责机器人手眼标定；系统集成
2. **梅静宜：**有着出色的数学建模、coding 能力；负责采用 RGB-D 相机识别瓶子，并获得抓取姿态
3. **王硕：** 曾担任院学生会主席，有着出色的领导力和组织能力；负责 PCA 合成信号、训练回归器
4. **刘嘉琨：**有着出色的多语论文撰写能力；负责 baxter 运动控制
5. **蔡雨洋：**参与过电设、嵌入式等竞赛，有着一定的软硬件经验；负责改装 Baxter 手指（后期辅助王硕录制数据集和调试回归器）

团队情况： 我们的团队以多元化的专业背景和优异的个体能力为特点，相互间紧密合作，线下交流经验丰富。

报告提交时间：11 月 13 日

课程目标：

基本要求：测试时通过摇晃判断瓶子内的水量；提高抓取成功率和识别正确率。

提高要求：测试时判断未训练过的瓶子内的水量

二、项目描述（Project Description）

整个项目旨在通过结合机器人的触觉和摇晃动作，实现对液体动态感知的高效控制。我们使用实验室中的 Baxter 机器人以及带有锚点的触觉传感器 GelSight，控制机器人对水杯进行摇晃，经过触觉传感器 GelSight 得到触觉锚点形变信号。这不仅为机器人在复杂环境中执行任务提供了更灵活的能力，还为各种应用场景，包括制药、化工、甚至餐饮服务等领域提供了创新性的解决方案。

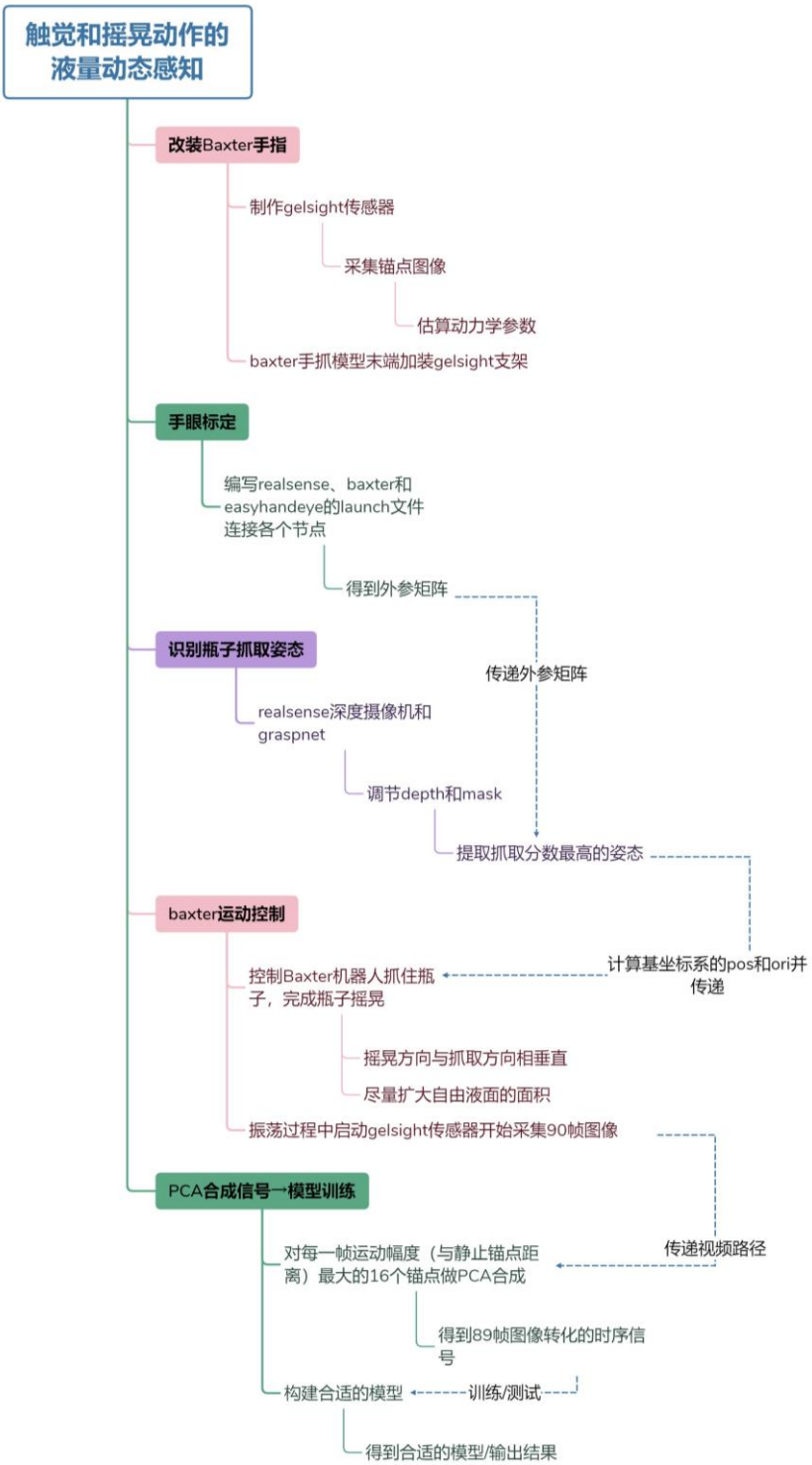

图 1 触觉感知任务流程

图一描述了触觉感知任务的流程，聚焦于项目的实现和评估。具体任务如下：

1. 机械手改装和传感器制作：

通过选择合适的摄像头和凝胶材料，制作 Gelsight（凝胶+摄像头）单元。在原本的 baxter 机械手末端添加 Gelsight（凝胶+摄像头）单元进行集成和调试，确保机器人能够准确获取液体的触觉信息，并实现采集图像程序。这涉及到传感器的硬件连接以及驱动程序的开发。

2. 采用 RGB-D 相机识别瓶子抓取姿态:

使用 realsense 摄像头和训练好的 graspnet 网络，调节 depth（深度）范围和 mask（图像使用范围），尽量将视野仅包含需要摇晃的瓶子，并提取抓取分数最高的姿态，将获取的 pos 和 ori 左乘外参矩阵后发布给运动程序。

3. baxter 运动控制：

控制 Baxter 机器人抓住瓶子，完成瓶子摇晃功能。其中，摇晃方向与抓取方向相垂直，同时尽量扩大自由液面的面积，这保证了凝胶感受的是瓶子所来的切向力。在摇晃过程中，启动 gelsight 传感器开始采集 h264 帧。

4. PCA 合成信号→模型训练：

采集摇晃过程中 89 帧（30fps）的 h264 流数据，对每一帧运动幅度（与静止锚点距离）最大的 16 个锚点做 PCA 合成得到一维向量，最后获取 89 帧的时序信号。利用采集到的数据，我们将训练液体动态感知的机器学习模型，预测液体的状态并测试了多种模型的预测效果，为机器人采取相应的动作提供指导。

5. 手眼标定：

使用 easyhandeye 和 aruco 标，编写 realsense、baxter 和 easyhandeye 的 launch 文件连接各个节点，采集图像完成标定工作。其得到的外参矩阵提供给抓取程序计算基坐标空间位置。

6. 系统集成：

由于我们的系统在 baxter.sh 的子 shell 空间下，我们在 baxter.sh 子 shell 空间下添加执行代码，完成脚本设计；同时，提供各个节点传递数据。在运动结束后，使用模型测试采集数据，输出瓶子水量：全空、半空和全满。

三、过程总结（Progress Summary）

在实现与评估阶段，我作为项目团队的一员，前期主要负责 gelsight 的制作和机械手的改装，后期辅助 PCA 合成和模型训练，并或多或少地参与其余模块的调试工作。

具体工作流程已上传：[newmancai/seu_gelsight (github.com)]

以下是我的个人工作细节：

1. **机械手改装和传感器制作：**

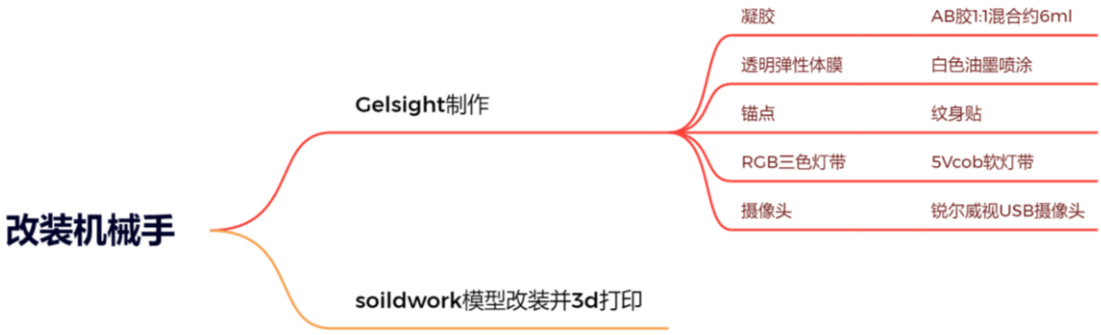

图 2 改装机械手流程

图 2 描述了改装机械手的流程，分为制作 gelsight 和 soildwork 改装模型并 3d 打印两个并行部分，其中凝胶为 1:1AB 胶混合，锚点采用纹身贴，RGB 灯带采用 cob 软灯带，摄像头为锐尔威视 usb 摄像头，可采集 yuyv 和 mjpeg 图像，支持 linux 系统。

具体的制作 gelsight 过程可参考 [newmancai/seu_gelsight (github.com)]

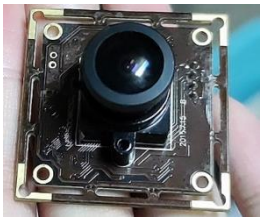

图 3 锐尔摄像头

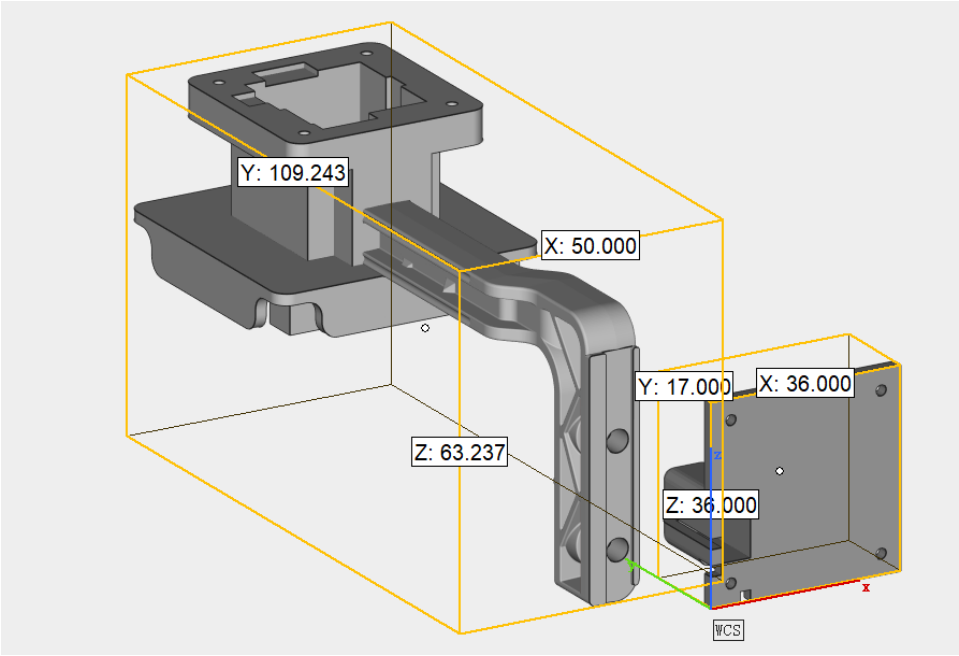

图 4 gelsight 支架模型

图 4 展示了 Gelsight 传感器支架的 soildwork 模型，在支架上装上凝胶、摄像头和灯带即构造了基本的 gelsight 传感器，使用摄像头捕获凝胶形变恢复力和运动。

我们购买了两种摄像头，无畸变摄像头和短焦摄像头，模拟摄像头视场区域的不同。同时，采用 AB 胶制作凝胶底，凝胶靠近模具的一侧不平坦，另一侧不平坦，白色油墨与稀释剂 1:4 混合形成覆盖层喷与平坦层，不平坦层使用纹身贴贴锚点。最后，焊接灯带、usb 接线和摄像头，封装贴入支架，完成了 gelsight 制作。图 6 右为搭载无畸变摄像头 gelsight 的新夹爪。

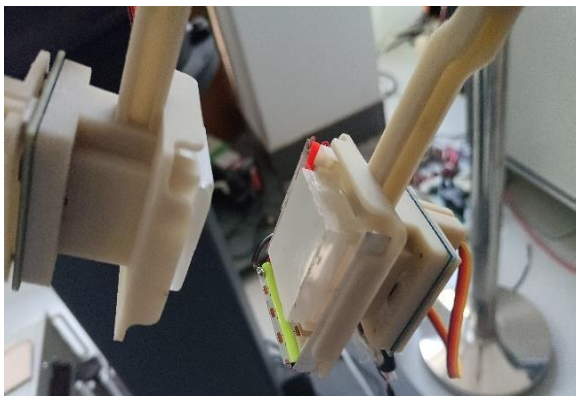

图 5 Gelsight mini（左为系统，右为单独凝胶）图 6 安装在 baxter 上的 gelsight 传感器

Cartridge Replacement	Easily user replaceable cartridge that maintains position. Replacement does not require any tools/hardware.
Gel Thickness	4.25 mm +/- .20 mm
Gel Concentricity to Cartridge	< 0.5 mm
Gel Material & Coating	Lambertian Silicone Gel
Camera Resolution	8MP
Camera Frame Rate	25FPS
Illumination	RGB LEDs
Field of View	18.6(H) x 14.3(V) mm
Material & Coating	Minimize residue on surface
Gel Durability	1000 coin presses

Price	不超过¥499
Gel Durablity	5000 coin presses
Camera Resolution	720P(1MP)
Illumination	RGB 灯带
Gel Material & Coating	1:1 混合 AB 胶
Camera Frame Rate	30fps

我们的传感器与市面上的 Gelsight 相比，使用的价格更低，更具有性价比，但 Field of View、Resolution 和 Material 具有较大差距。

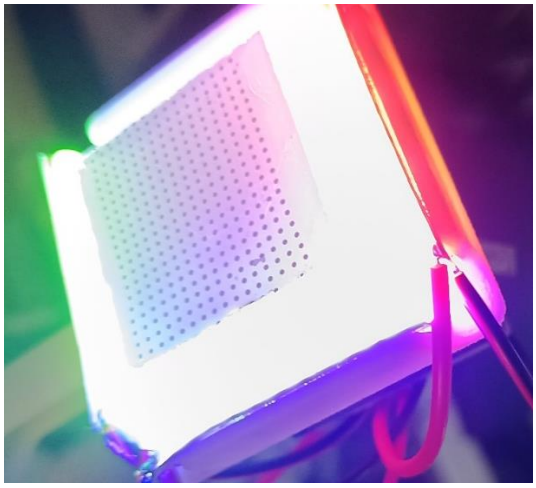

图 7 连接 usb 后，可以看到锚点

图 7 为连接后的传感器，可以清晰地看到锚点图像。

2. **PCA 合成信号→模型训练：**

AB 胶属于韧性胶黏剂，其制作的 Gel 的 Durablity 较高，但剪切模量大于柔性胶黏剂，且本次使用的凝胶厚度约 8mm，剪切强度（与厚度正相关）较高。

压缩模量较高

拉伸强度 kg/mm2 ≥22，

较难有径向形变

剪 切 强 度 高

其剪切强度为 80±2℃×2h,LY12CZ

≥18Mpa，可以感受振荡过程

该凝胶较硬，回弹速度快，因此无法捕获阻尼振荡过程，所以我们选择在振荡过程采集凝胶形变数据集。

结合模型训练的需求，需要 3s 的图像集。锐尔威视的摄像头有 yuyv 模式和 mjpeg 模式，1280*720 只有 30fps。我采用 ffmpeg 调用摄像头录制和 gpac 包装 h264 流的 MP4，共 90 帧，即 3s。图 8 展示了摄像头采集的锚点图像。

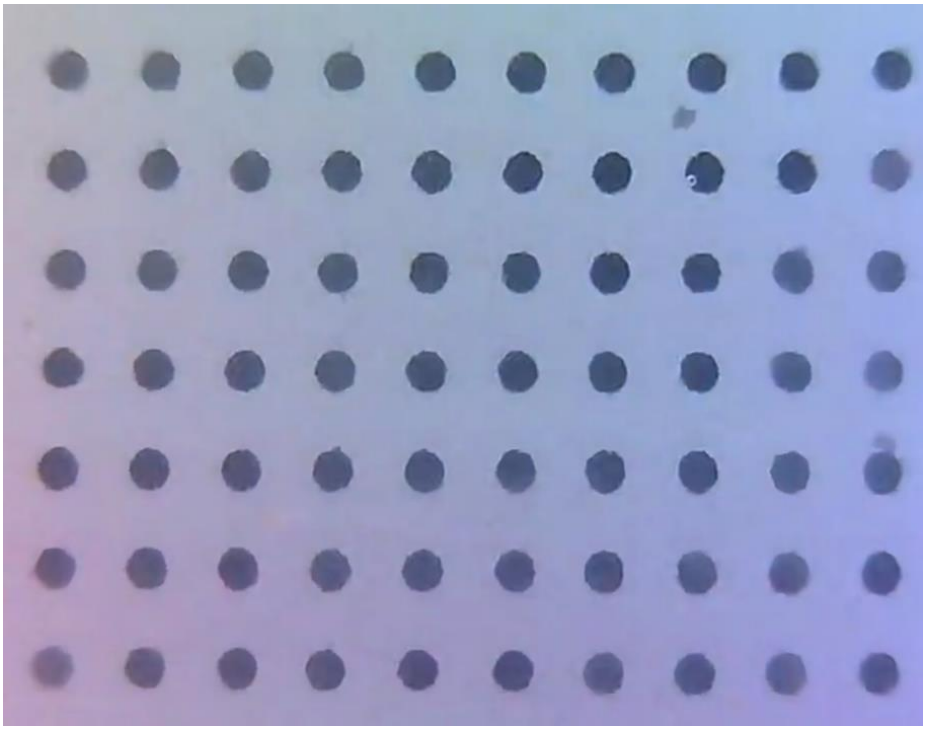

图 8 部分锚点图像（即凝胶表面人为贴上的排列规则的点，用于计算凝胶形变）

之后录制摇晃过程中的视频数据集，编写了单独 baxter 摇晃程序后，我们将瓶子预放入夹爪，采集视频图像，每一次更换夹爪夹取瓶子的角度，一个水位录制 10 份，分为全空、半满和全满。在此基础上，我们对比了各种瓶子，图 9 左是测试使用的各种瓶子，由于 AB 胶(白色油墨)制作的凝胶的杨氏模量和剪切模量较高,所以凝胶本身的形变是较为困难的，我们需要较为柔软、且有弹性的瓶子传递振荡。在 5 个瓶子中，农夫山泉的形变是最为规律、明显的。

图 9 左：测试瓶子（未加包装封纸） 右：实验瓶子

3. **其余工作**

协助完成了 baxter 运动控制中的摇晃过程编写，解决在 movit 规划下无法单独旋转关节的问题；协助手眼标定和抓取工作，解决 yaml 中的参数异常和外参矩阵的计算问题；与团队一起参与了系统集成与优化的工作，并确保传感器能与其他系统协同工作。

为了满足需求，即无法通过肉眼看到瓶子的水量，我在原有农夫山泉的瓶子外贴上白纸，如图 5 右所示。

4. **优化设计**

针对与灯带额外受力脱落重焊带来的人力资源消耗，我认为可以设计简易的电路板，类似与下图，实现 usb 转接，提供 usb 输入接口、usb 输出端口（摄像头）、5V 和 gnd（灯带）。

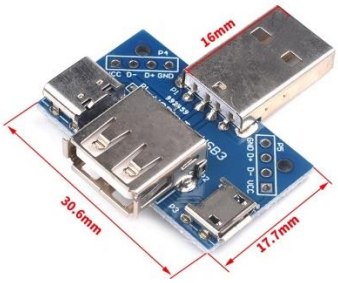

图 10 usb 转接电路板

再购置 3 个 cob 灯带免焊接口，先将三个灯带连接在一起，再连出一个接口，直接连入转接板，完成组装，这样就可以减少多余线的冗余连接，美观的同时能有效避免线的脱落。

图 11 免焊接口连接下的 3 线

四、遇到的问题（**Problems Encountered**）

问题梗概	问题详细描述	解决方案
传感器制作	凝胶脱模质量差	重新设计了新的模具，是的其深度符合对应需求高度，3d 打印精磨制作。
视频丢帧	采集视频无法正常作为数据输入的问题	luvcview 采集 mjpeg 或者 ffmpeg 处理 h264 编码 MP4 包装的数据都会导致缺帧。因此最后采用了 ffmpeg 采集 yuyv422 数据转换为 yuv420 再用 gpac 处理数据封装成 video
锚点提取	锚点无法正常识别	首先调节了识别锚点的色彩空间代码，其次原本我们采用的锚点识别算法采用的是预设矩形坐标定位初始锚点，之后对每一个锚点做卡尔曼跟踪，这意味着对于初始图像的平直有着较高的需求。我们对锚点空间做了限制。后期我们更换了算法，采用光流法，彻底解决了该问题。
运动控制	在启动 baxter 运动后，无法单独旋转关节	将问题定位到了 mov_home 中的 group.go()，在 group 启用 go 后无法单独使用关节调整函数，所以我们将目标位置组在关节空间下的改变转换为 pos 和 ori，由 movit 单独规划运动。
系统集成	工作空间需要启动 baxter.sh,开辟了子 shell 空间，无法正常传参	我们跟踪子 shell 空间，在子 shell 空间内完成节点的信息传递。

图 12 深度不适配的旧模具

图 13 新模具

问题的影响：

这些问题的出现在一定程度上影响了任务的进度和效率。凝胶制作延缓了 gelsight 制作进度，而锚点提取问题则是增加了数据集录制难度。值得庆幸的是，通过及时的问题识别和有效的解决方案，我们成功地克服了这些障碍，并确保项目进展顺利。这些问题的解决也为后续阶段的工作提供了宝贵经验教训，使团队更加具备应对类似挑战的能力。

五、需求改变（**Changes in Requirements**）

在项目的实施过程中，我们经历了一些需求的调整和改变，主要涉及到以下几个方面：

1. **传感器规格：**
   - **原需求：**规定了特定型号的摄像头。
   - **变更：** 采用多种摄像头。（适应图像区域过小的问题）
2. **模型训练：**
   - **原需求：**规定了全满和全空。
   - **变更：** 变更为全满、半满和全空。
3. **摇晃方向：**
   - **原需求：**液体摇晃方向与抓取方向一致。（径向力）
   - **变更：** 液体摇晃方向与抓取方向相垂直。（切向力）

这些需求的调整主要基于实际工作中的技术和经验积累，以及对系统性能的不断优化。与指导老师的密切沟通确保了这些变更的合理性和可接受性，并且我们在每一次变更后都进行了相应的文档更新，以保持团队的一致理解。这些变更对项目的整体目标并没有造成实质性的影响，反而提高了系统的适应性和性能。

六、总体评价（**Overall Assessment**）

整个项目的实施过程充满了挑战，但也取得了显著的进展。团队成员的紧密协作和灵活应对问题的能力是项目成功的关键。以下是对整个项目过程的总体评价：

1. **团队协作：** 团队协作非常出色，每个成员都充分发挥了自己的专业优势。团队成员之间的沟通畅通，共同面对挑战，保持了高效的工作氛围。
2. **问题解决：**遇到的问题多种多样，包括硬件兼容性、数据标注一致性等方面的挑战。团队成功地应对了这些问题，通过及时的沟通和协作找到了解决方案。
3. **需求管理：** 随着项目的进行，发现了一些需求的调整和改变。团队对需求的变更作出了及时的反应，并通过合理的变更管理确保了项目的方向不偏离初衷。
4. **技术实现：** 在实现与评估阶段，团队成功集成了传感器、graspnet 并完成了机器学习模型的训练，并对于手眼标定的结果精益求精，进行了系统优化。

设计并完成该项目较之于传统的大作业而言，题目更加的新颖，有着清晰的任务指标和需求，但整个过程的细节实现要结合相应论文和经验来完成，可以说是具有相当的挑战性的。

本次项目实现了基于 baxter 机器人的触觉(主动)感知未知水量的问题，我们构建了 batxer 抓取瓶子并摇晃的运动过程，通过分析摇晃过程中的凝胶形变来感知水量，并对整个过程提出了两个指标，抓取成功率和识别成功率。

（1）抓取过程

该过程由识别、抓取、运动三部分组成。

我们最终选择的抓取对象为农夫山泉，其宽度和夹爪的最大直径差仅为 0.8cm，这对我们的抓取提出了巨大的挑战。

受光照影响，识别率大约为 80%。受限于 baxter 的重复定位精度和手眼标定精度，想要实现精准的抓取是较为困难的，在识别正确后成功率约为 50%。由于没有存在闭环的矫正，我们没有给出在实际到达瓶子附近时的偏差修正，是一个可行的改进方向。

在现阶段 baxter 的运动控制中，我们放弃了 movit 而选择了直线规划，因为往往较为简单但路径在 movit 规划下依然非常扭曲，同时有碰撞的风险，一方面是我们在地图中没有构建类似桌子类的障碍物，另一方面是 baxter 机器人自身的底座碰撞体积存在底层问题，后续可以更改底层逻辑更高地完成运动控制的设计工作。

（2）识别过程

该过程需要在摇晃构成中采集凝胶并分析数据。

如今 Gelsight 的机械手采用 usb4 线分接的方法，耗费人力资源的同时增加了线和灯的额外受力，增加了掉线、掉灯的风险。后续可以将摄像头区域侧边镂空，设计 pcb 电路，增加购置灯带免焊连接头，先将三灯连接，再配入 5V 电源。凝胶后续可以采用柔性粘合剂制作，同时减少其厚度，可以使得采集图像变化更大，增加识别率。

识别采用了 PCA+SVM 多种 mode，只有一种 mode 下的识别率超过了 70%。一方面是数据集过少，后续可以在数据集的数量上下文章。但就分类而言，由于其运动图像特征（幅度）与水量并非正比关系，因此如果要增加水量的分类，需要重新设计运动过程或者适配更高级的凝胶来提取更详细的特征。另一方面是识别采用了 89 维的时序信号本身 SVM，其效果一般，后续可以提取频率和最高赋值等参数构建不超过 5 维的特征向量进行训练。

总体而言，项目团队在各个阶段都展现出高度的专业素养和团队协作能力。虽然面临了一些挑战，但通过团队成员的共同努力，成功地克服了这些困难。整个项目的实施为团队成员提供了宝贵的实战经验，同时也为未来类似项目的开展提供了有益的教训。

附录

采集数据集代码

```
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "libavdevice/avdevice.h"
#include "libavformat/avformat.h"
#include "libavutil/avutil.h"
#define V_WIDTH 1280
#define V_HEIGHT 720
#define FRAMES 90

void create_time_command(char command[]){
    time_t tt = time(0);
    char s[32];
    strftime(s, sizeof(s), "%02m%02d_%02H%02M%02S", localtime(&tt));
    sprintf(command,"MP4Box -add video.h264 -fps 30 ./result/out%s.mp4",s);
}

//@brief
// return
static AVFormatContext *open_dev() {
    int ret = 0;
    char errors[1024] = {
        0,
    };

    // ctx
    AVFormatContext *fmt_ctx = NULL;
    AVDictionary *options = NULL;

    //摄像头的设备文件
    char *devicename = "/dev/video6";

    // register video device
    avdevice_register_all();

    // get format
    AVInputFormat *iformat = av_find_input_format("video4linux2");

    av_dict_set(&options, "video_size", "1280x720", 0);
    av_dict_set(&options, "framerate", "30", 0);
    av_dict_set(&options, "pixel_format", "yuyv422", 0);
```

```
42.
43.     // open device
44.     ret = avformat_open_input(&fmt_ctx, devicename, iformat, &options);
45.     if (ret < 0) {
46.         av_strerror(ret, errors, 1024);
47.         fprintf(stderr, "Failed to open video device, [%d]%s\n", ret, errors);
48.         return NULL;
49.     }
50.
51.     return fmt_ctx;
52. }
53.
54. static void open_encoder(int width, int height, AVCodecContext **enc_ctx) {
55.     int ret = 0;
56.     AVCodec *codec = NULL;
57.     //新版本无需调用
58.     //avcodec_register_all();
59.
60.     codec = avcodec_find_encoder_by_name("libx264");
61.     if (!codec) {
62.         printf("Codec libx264 not found\n");
63.         exit(1);
64.     }
65.
66.     *enc_ctx = avcodec_alloc_context3(codec);
67.     if (!enc_ctx) {
68.         printf("Could not allocate video codec context!\n");
69.         exit(1);
70.     }
71.
72.     // SPS/PPS
73.     (*enc_ctx)->profile = FF_PROFILE_H264_HIGH_444;
74.     (*enc_ctx)->level = 50; //表示 LEVEL 是 5.0
75.
76.     //设置分辨率
77.     (*enc_ctx)->width = width;   // 640
78.     (*enc_ctx)->height = height; // 480
79.
80.     // GOP
81.     (*enc_ctx)->gop_size = 250;
82.     (*enc_ctx)->keyint_min = 25; // option
83.
84.     //设置 B 帧数据
85.     (*enc_ctx)->max_b_frames = 3; // option
```

```
86.     (*enc_ctx)->has_b_frames = 1; // option
87.
88.     //参考帧的数量
89.     (*enc_ctx)->refs = 3; // option
90.
91.     //设置输入 YUV 格式
92.     (*enc_ctx)->pix_fmt = AV_PIX_FMT_YUV420P;
93.
94.     //设置码率
95.     (*enc_ctx)->bit_rate = 600000; // 600kbps
96.
97.     //设置帧率
98.     (*enc_ctx)->time_base = (AVRational){1, 30}; //帧与帧之间的间隔是 time_base
99.     (*enc_ctx)->framerate = (AVRational){30, 1}; //帧率，每秒 30 帧
100.
101.    ret = avcodec_open2((*enc_ctx), codec, NULL);
102.    if (ret < 0) {
103.        printf("Could not open codec: %s!\n", av_err2str(ret));
104.        exit(1);
105.    }
106. }
107.
108. static AVFrame *create_frame(int width, int height) {
109.    int ret = 0;
110.    AVFrame *frame = NULL;
111.
112.    frame = av_frame_alloc();
113.    if (!frame) {
114.        printf("Error, No Memory!\n");
115.        goto __ERROR;
116.    }
117.
118.    //设置参数
119.    frame->width = width;
120.    frame->height = height;
121.    frame->format = AV_PIX_FMT_YUV420P;
122.
123.    // alloc inner memory
124.    ret = av_frame_get_buffer(frame, 32); //按 32 位对齐
125.    if (ret < 0) {
126.        printf("Error, Failed to alloc buffer for frame!\n");
127.        goto __ERROR;
128.    }
129.
```

```
130.     return frame;
131. 
132. __ERROR:
133.     if (frame) {
134.         av_frame_free(&frame);
135.     }
136. 
137.     return NULL;
138. }
139. 
140. static void encode(AVCodecContext *enc_ctx, AVFrame *frame, AVPacket *newpkt,
141.                    FILE *outfile) {
142.     int ret = 0;
143.     if (frame) {
144.         //printf("send frame to encoder, pts=%ld", frame->pts);
145.     }
146.     //送原始数据给编码器进行编码
147.     ret = avcodec_send_frame(enc_ctx, frame);
148.     if (ret < 0) {
149.         printf("Error, Failed to send a frame for enconding!\n");
150.         exit(1);
151.     }
152. 
153.     //从编码器获取编码好的数据
154.     while (ret >= 0) {
155.         ret = avcodec_receive_packet(enc_ctx, newpkt);
156. 
157.         //如果编码器数据不足时会返回  EAGAIN,或者到数据尾时会返回 AVERROR_EOF
158.         if (ret == AVERROR(EAGAIN) || ret == AVERROR_EOF) {
159.             return;
160.         } else if (ret < 0) {
161.             printf("Error, Failed to encode!\n");
162.             exit(1);
163.         }
164. 
165.         fwrite(newpkt->data, 1, newpkt->size, outfile);
166.         fflush(outfile);
167.         av_packet_unref(newpkt);
168.     }
169. }
170. 
171. void yuyv422ToYuv420p(AVFrame *frame, AVPacket *pkt) {
172.     int i = 0;
173.     int yuv422_length = V_WIDTH * V_HEIGHT * 2;
```

```
174.     int y_index = 0;
175.     // copy all y
176.     for (i = 0; i < yuv422_length; i += 2) {
177.         frame->data[0][y_index] = pkt->data[i];
178.         y_index++;
179.     }
180.
181.     // copy u and v
182.     int line_start = 0;
183.     int is_u = 1;
184.     int u_index = 0;
185.     int v_index = 0;
186.     // copy u, v per line. skip a line once
187.     for (i = 0; i < V_HEIGHT; i += 2) {
188.         // line i offset
189.         line_start = i * V_WIDTH * 2;
190.         for (int j = line_start + 1; j < line_start + V_WIDTH * 2; j += 4) {
191.             frame->data[1][u_index] = pkt->data[j];
192.             u_index++;
193.             frame->data[2][v_index] = pkt->data[j + 2];
194.             v_index++;
195.         }
196.     }
197. }
198.
199. void rec_video() {
200.     int ret = 0;
201.     int base = 0;
202.     int count = 0;
203.     int frame_index = 0;
204.     int64_t calc_duration = 0;
205.     // pakcet
206.     AVPacket pkt;
207.     AVFormatContext *fmt_ctx = NULL;
208.     AVCodecContext *enc_ctx = NULL;
209.
210.     // set log level
211.     av_log_set_level(AV_LOG_DEBUG);
212.
213.     // create file
214.     char *yuvout = "./video.yuv";
215.     char *out = "./video.h264";
216.
217.     FILE *yuvoutfile = fopen(yuvout, "wb+");
```

```
218.     FILE *outfile = fopen(out, "wb+");
219.
220.     //打开设备
221.     fmt_ctx = open_dev();
222.
223.     //打开编码器
224.     open_encoder(V_WIDTH, V_HEIGHT, &enc_ctx);
225.
226.     calc_duration = (double)AV_TIME_BASE / av_q2d((*enc_ctx).framerate);
227.
228.     //创建 AVFrame
229.     AVFrame *frame = create_frame(V_WIDTH, V_HEIGHT);
230.
231.     //创建编码后输出的 Packet
232.     AVPacket *newpkt = av_packet_alloc();
233.     if (!newpkt) {
234.         printf("Error, Failed to alloc avpacket!\n");
235.         goto __ERROR;
236.     }
237.
238.     // read data from device
239.     while (((ret = av_read_frame(fmt_ctx, &pkt)) == 0) && (count++ < FRAMES)) {
240.         int i = 0;
241.         //av_log(NULL, AV_LOG_INFO, "packet size is %d(%p)\n", pkt.size,
242.         //       pkt.data);
243.
244.         // YUYVYUYVYUYVYUYV   YUYV422
245.         // YYYYYYYYUUVV YUV420
246.         yuyv422ToYuv420p(frame, &pkt);
247.
248.         fwrite(frame->data[0], 1, 307200, yuvoutfile);
249.         fwrite(frame->data[1], 1, 307200 / 4, yuvoutfile);
250.         fwrite(frame->data[2], 1, 307200 / 4, yuvoutfile);
251.
252.         frame->pts = base++;
253.
254.
255.         if(newpkt->pts == AV_NOPTS_VALUE) {
256.           //Write PTS
257.           AVRational time_base1 =  (*enc_ctx).time_base;
258.           //Duration between 2 frames (us)
259.           //Parameters
260.           newpkt->pts = (double)(frame_index*calc_duration) /
(double)(av_q2d(time_base1)*AV_TIME_BASE);
```

```
261.            newpkt->dts = newpkt->pts;
262.            newpkt->duration = (double)calc_duration /
(double)(av_q2d(time_base1)*AV_TIME_BASE);
263.            frame_index++;
264.        }
265.        encode(enc_ctx, frame, newpkt, outfile);
266.        //
267.        av_packet_unref(&pkt); // release pkt
268.    }
269.
270.    encode(enc_ctx, NULL, newpkt, outfile);
271.
272. __ERROR:
273.    if (yuvoutfile) {
274.        // close file
275.        fclose(yuvoutfile);
276.    }
277.
278.    // close device and release ctx
279.    if (fmt_ctx) {
280.        avformat_close_input(&fmt_ctx);
281.    }
282.
283.    av_log(NULL, AV_LOG_DEBUG, "finish!\n");
284.    return;
285. }
286.
287.
288. int main(int argc, char *argv[]) {
289.    rec_video();
290.    char command[128];
291.    create_time_command(command);
292.    system(command);
293.    return 0;
294. }
```