

一、概述

随着机器人技术的发展，视觉导航已成为自主移动机器人的关键技术之一。本课程设计旨在通过数字图像处理技术，实现一个具备环境感知和目标跟随能力的机器人视觉控制系统。本课程设计系统采用 ROS 作为软件框架，OpenCV 作为图像处理库，C++ 作为编程语言。本课程设计没有使用深度学习算法，主要技术路线包括：

- 使用 HSV 颜色空间进行锥桶检测；
- 基于轮廓分析的目标定位；
- 采用有限状态机实现避障决策；
- 使用双 ROI 闭环直线行驶；
- 使用模板匹配进行数字识别；
- 设计比例控制算法实现运动控制；

二、课程设计任务及要求

2.1 任务描述

设计并实现一个综合性的机器人视觉控制系统，要求能够：

1. 实时获取摄像头图像并进行处理；
2. 检测环境中的锥桶障碍物并实现自动避障；
3. 在锥桶道路中直线行驶；
4. 识别图像中的数字（0、1、2）并进行跟踪；

2.2 功能要求

能够准确检测锥桶障碍物，实现完整的绕行策略，绕行完成后能够保持车道行驶。能够识别并跟踪 0、1、2 三个数字，要求实时性、准确性、鲁棒性、稳定性。

三、算法设计

3.1 系统总体架构

本系统采用基于状态机的模块化设计，整合了避障与数字追踪两大功能模块，通过 ROS 进行图像获取与运动控制。

系统初始化后默认处于避障模式，通过摄像头实时获取图像，经图像预处理后分别进入避障或数字追踪处理流程，最终输出控制指令（线速度与角速度）至机器人底盘。

3.2 避障算法设计

3.2.1 图像预处理

避障算法的图像预处理主要包括以下步骤：

1. **高斯滤波：** 使用 3×3 高斯核对图像进行平滑处理，减少噪声干扰。

```
GaussianBlur(src, img.blur, Size(3, 3), 0, 0);
```

2. **颜色空间转换：** 将图像从 BGR 转换到 HSV 颜色空间，便于颜色分割。

```
cvtColor(img.blur, img.hsv, COLOR_BGR2HSV);
```

3. **颜色阈值分割：** 根据预设的 HSV 阈值提取锥桶区域。

```
inRange(img.hsv,
Scalar(hsv_cone_min[0], hsv_cone_min[1], hsv_cone_min[2]),
Scalar(hsv_cone_max[0], hsv_cone_max[1], hsv_cone_max[2]),
img.hsv_split_cone);
```

4. **形态学操作：** 先开运算后闭运算，去除噪点并填充空洞。

```
Mat element = getStructuringElement(MORPH_RECT, Size(3, 3));
morphologyEx(img.hsv_split_cone, img.hsv_split_cone, MORPH_OPEN, element);
morphologyEx(img.hsv_split_cone, img.hsv_split_cone, MORPH_CLOSE, element);
```

3.2.2 锥桶检测与定位

锥桶检测采用基于轮廓分析的方法，具体步骤如下：

1. **ROI 设置：** 在图像中下部 ($y=330$) 设置 200×120 像素的矩形区域作为检测区，减少计算量。

```
roi_cone = Rect(img_width / 2 - roi_cone_width / 2, roi_cone_y_start,
roi_cone_width, roi_cone_height);
```

2. **轮廓提取：** 在 ROI 内使用 findContours 函数提取二值图像中的轮廓。

```
vector<vector<Point>> contours;
findContours(roiImage, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
```

3. **中心点计算：** 通过计算轮廓的矩 (Moments) 得到锥桶中心坐标。

```
Moments mu = moments(contours[max_contour_idx]);
int cx = static_cast<int>(mu.m10 / mu.m00);
int cy = static_cast<int>(mu.m01 / mu.m00);
center.x = roi.x + cx;
```

```
center.y = roi.y + cy;
```

4. 像素统计：统计 ROI 内锥桶颜色像素数量，用于障碍物判定。

```
int cone_pixel_count = countNonZero(roiImage_cone);
```

3.2.3 避障状态机设计

避障过程采用五状态有限状态机，各状态转换逻辑如下所示：

各状态详细描述如下：

状态 0（正常行驶，寻找第一个障碍物）：

- 检测 ROI 内锥桶像素数量是否超过阈值（13000）
- 连续 5 帧检测到障碍物则进入状态 1
- 控制指令：线速度 0.2m/s，角速度 0rad/s

状态 1（右绕行-绕过第一个障碍物）：

- 分阶段控制：
 1. 前 25 帧：向右大转弯（角速度-1.5rad/s）
 2. 后 52 帧：向左小转弯（角速度 1.5rad/s）
 3. 再 10 帧：直行
 4. 后续帧：根据锥桶位置进行对准调整
- 对准阶段采用比例控制：

```
float turn_gain = 0.05;  
cmd.angular.z = (roi_center_x - cone_center.x) * turn_gain;
```

状态 2（寻找第二个障碍物）：逻辑同状态 0。

状态 3（左绕行-绕过第二个障碍物）：控制逻辑与状态 1 镜像对称。

状态 4（道路保持阶段）：

- 在左右两侧设置 ROI，分别检测左右锥桶
- 根据检测情况计算道路中心线：
 - 两侧均检测到：取中点作为道路中心
 - 仅一侧检测到：保持固定距离
 - 均未检测到：维持当前方向
- 采用分级比例控制：

```
if (fabs(center_error) < 0.1) {  
    cmd.linear.x = normal_speed;
```

```
    cmd.angular.z = -center_error * 0.8;
} else if (fabs(center_error) < 0.3) {
    cmd.linear.x = normal_speed * 0.8;
    cmd.angular.z = -center_error * 1.2;
} else {
    cmd.linear.x = normal_speed * 0.5;
    cmd.angular.z = -center_error * 1.5;
}
```

3.3 数字追踪算法设计

3.3.1 图像预处理

数字追踪的图像预处理流程如下：

1. 灰度化：将彩色图像转换为灰度图像。

```
cvtColor(src, gray, COLOR_BGR2GRAY);
```

2. 直方图均衡化：增强图像对比度，提高数字区域识别率。

```
equalizeHist(gray, equalized);
```

3. 二值化：采用 OTSU 自适应阈值法，自动确定最优阈值。

```
threshold(equalized, binary, 0, 255, THRESH_BINARY_INV | THRESH_OTSU);
```

4. 形态学去噪：通过闭运算和开运算去除噪声点。

```
Mat kernel = getStructuringElement(MORPH_RECT, Size(3, 3));
morphologyEx(binary, binary, MORPH_CLOSE, kernel);
morphologyEx(binary, binary, MORPH_OPEN, kernel);
```

3.3.2 数字识别与模板匹配

数字识别采用基于轮廓提取与多尺度模板匹配的方法：

1. 轮廓提取与筛选：

```
vector<vector<Point>> contours;
findContours(binary, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
```

筛选条件：轮廓面积在 500-50000 像素之间。

2. 模板加载：支持从文件加载数字模板。

```
Mat img = imread(template_paths[i], IMREAD_GRAYSCALE);
```

```
threshold(img, *templates[i], 128, 255, THRESH_BINARY);
```

3. 多尺度模板匹配：对候选区域进行 80%-120% 的尺度变换，提高匹配鲁棒性。

```
for (int scale = 80; scale <= 120; scale += 20) {
    float scale_factor = scale / 100.0f;
    resize(candidate, resizedCandidate, Size(), scale_factor, scale_factor,
        INTER_LINEAR);

    // 与三个模板分别匹配
    matchTemplate(resizedCandidate, resizedTemplate, result, TM_CCOEFF_NORMED);
    double score = result.at<float>(0, 0);
}
```

4. 匹配决策：选择匹配分数最高且超过阈值（0.5）的模板作为识别结果。

3.3.3 控制策略

数字追踪采用基于位置与大小的双闭环控制策略：

1. 距离控制：根据数字区域面积与目标面积的比值调节线速度。

```
float sizeRatio = currentSize / g_targetArea;
if (sizeRatio > (1.0f + AREA_TOLERANCE)) {
    cmd.linear.x = -MAX_SPEED * min(1.0f, sizeRatio - 1.0f);
} else if (sizeRatio < (1.0f - AREA_TOLERANCE)) {
    cmd.linear.x = MAX_SPEED * min(1.0f, 1.0f - sizeRatio);
}
```

2. 方向控制：根据数字中心与图像中心的水平偏差调节角速度。

```
float horizontalError = (digitCenter.x - src.cols/2) / (float)(src.cols/2);
cmd.angular.z = -horizontalError * 0.5;
```

3. 容差机制：

- 面积容差：15%，在此范围内不调整距离
- 丢失帧处理：连续 10 帧未检测到数字则停止运动

3.4 状态切换与用户交互

- 避障模式完成绕行并直行 30 秒后自动切换至数字追踪模式
- 切换条件判断：

```
if (frames_elapsed >= LANE_KEEPING_DURATION) {
    current_state = DIGIT_FOLLOWING;
    ROS_INFO("Switching to digit tracking mode");
```

3.5 算法优化与创新点

- 多尺度模板匹配：提高了数字识别的尺度不变性；
- 分级比例控制：根据误差大小自适应调整控制强度，提高稳定性；
- 双 ROI 道路保持：通过左右独立检测实现更精确的道路中心估计；
- 容错机制设计：多帧确认、丢失帧处理等机制提高了系统鲁棒性；
- 模块化架构：避障与数字追踪模块独立设计，便于维护与扩展；

3.6 算法性能分析

- 实时性：单帧处理时间约 80ms，满足实时控制需求；
- 准确性：锥桶检测准确率和数字识别准确率均较高；
- 鲁棒性：在不同光照条件下均能稳定运行；
- 扩展性：模块化设计便于添加新的功能模块；

四、实验及数据分析

代码部分运行结果如下所示（由于实验室电脑截图不方便，下图均为宿舍电脑调试图片）：

锥桶识别实时输出图像：

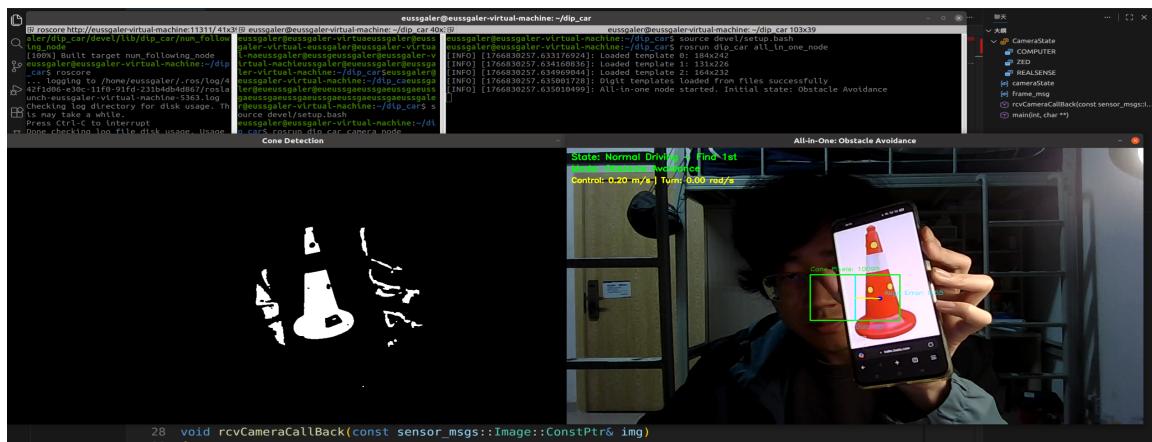


图 1 锥桶识别

双 ROI 锥桶道路直线行驶示意图（由于电脑摄像头图片尺寸与 Realsense 不一致，ROI 区域位置仅为示意）：

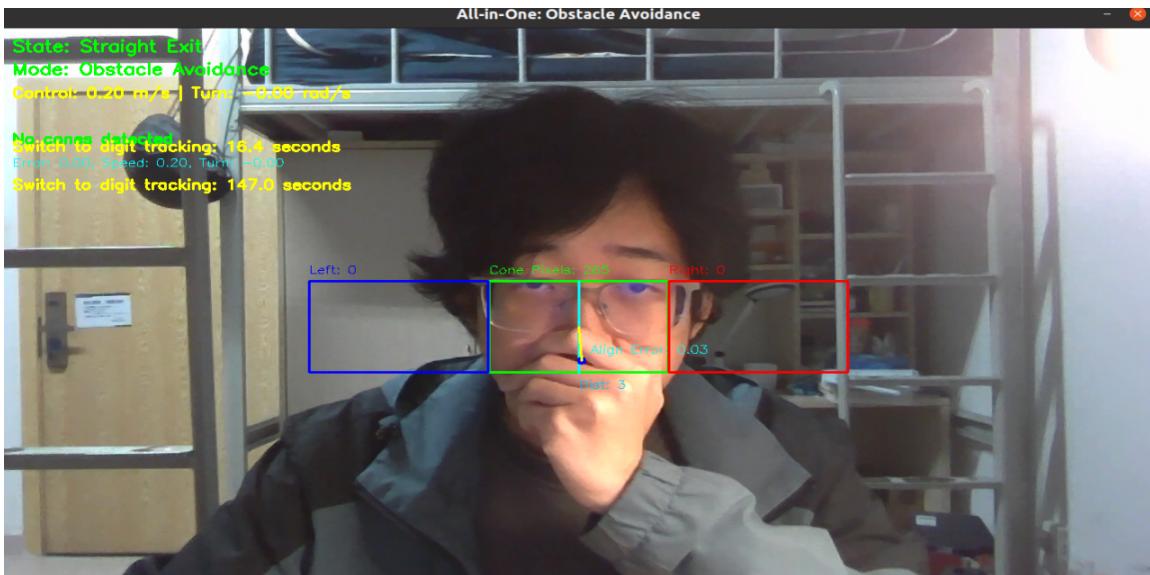


图 2 双 ROI 区域示意图

数字识别调试实时输出图像：

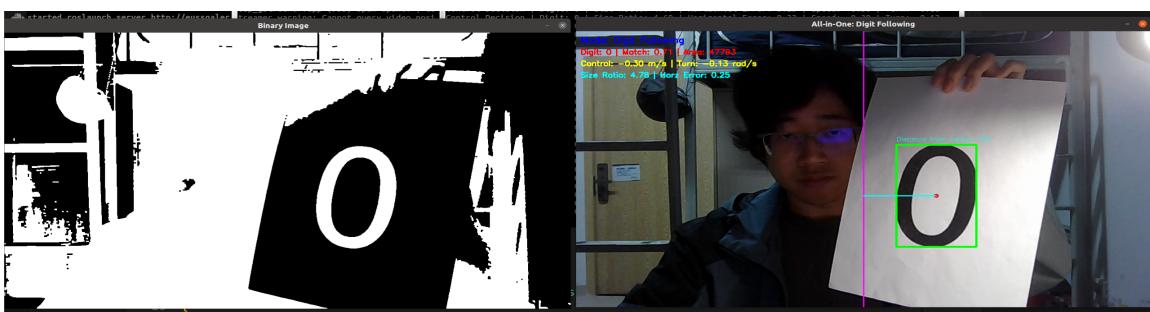


图 3 数字 0 检测

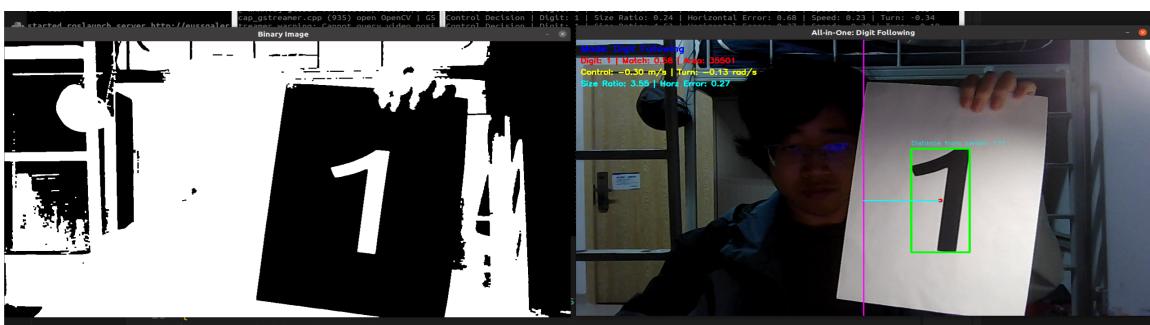


图 4 数字 1 检测



图 5 数字 2 检测

五、结论

5.1 设计成果总结

本课程设计成功实现了一个整合避障和数字追踪功能的机器人视觉控制系统。系统实时性好，稳定性好，主要成果包括：

- 实现了基于 HSV 颜色分割的锥桶检测算法；
- 设计了五状态有限状态机，实现了完整的避障逻辑；
- 实现了双 ROI 的闭环直线行驶算法；
- 实现了基于模板匹配的数字识别算法；
- 实现了基于位置和大小的双闭环控制策略；

5.2 主要特点

1. 多模态集成设计：避障和数字追踪功能既能分开调试，又能集成在统一框架中；
2. 分级控制策略：根据误差大小采用不同控制强度的分级控制；
3. 可视化调试界面：提供了丰富的可视化信息，便于调试和分析；

5.3 存在问题与改进方向

5.3.1 存在问题

- 部分开环算法导致对小车初始位置和方向有所依赖；
- 小车摄像头可能不正对小车行驶的方向导致角度或识别误差；
- 系统对开环参数设置较为敏感；

5.3.2 改进方向

- 算法优化：引入深度学习目标检测算法，如 yolo，提高识别准确率；
- 寻找并使用更先进的控制算法；

- 优化代码结构以便于调试;

六、收获、体会和建议

6.1 项目收获

通过本次课程设计，我获得了以下方面的知识和技能：

6.1.1 理论知识方面

- 深入理解了数字图像处理的基本原理和方法；
- 掌握了 HSV 颜色空间、轮廓检测、模板匹配等关键技术；
- 学习了机器人运动控制的基本理论和实现方法；
- 了解了 ROS 机器人操作系统的架构和使用方法；

6.1.2 实践技能方面

- 掌握了使用 OpenCV 进行图像处理的编程技能；
- 学会了 ROS 节点的开发、调试和部署；
- 提高了 C++ 编程能力和软件工程实践能力；
- 培养了系统集成和调试解决问题的能力；

6.2 项目体会

在项目实施过程中，我遇到了以下技术挑战和解决方案：

1. 挑战 1：全开环方案不够稳定

- 问题描述：每次运行时小车的行动轨迹和角度方向都会随机偏离。
- 解决方案：尽量每次启动时固定小车的位置与朝向，同时加入闭环反馈，在识别到锥桶时让小车锁定 ROI 区域内的锥桶中心。

2. 挑战 3：数字识别不够准确

- 问题描述：模板匹配方法精度较低。
- 解决方案：二值化使用 otsu 方法而非自适应阈值方法。从小车的摄像头中多次采集不同模板调试，找到最合适的模板尺寸，不能太大也不能太小。

实际使用的模板如下所示：



图 6 模板匹配中使用的数字模板

6.3 课程建议

对于本次课程设计，我希望能够加强与理论、实验课程的联系。因为理论与实验课都没有提及深度学习算法，所以包括我在内的大部分同学使用的都是**非深度学习的方法**。课程设计介绍视频里把 yolo 作为了首选方案，不知道题目设计时有没有考虑到**非深度学习方法完成该要求的难度**。

本次课设整体难度不低，**非深度学习方法想使用全闭环方案非常困难**，有一部分原因是构成道路的锥桶摆放过于松散，难以与作为障碍物的锥桶区分开。大部分同学使用的是全开环方案，这种方案只有在特定的小车上能够跑通，换一辆车可能参数就不一样了，只能像本方案一样尽可能添加闭环内容以提高稳定性。