南開大學

高级语言程序设计 课程实验报告

掌间流明项目

学院　人工智能学院

专业　工科试验班（信息类）

姓名　黄子豪

学号　2413989

2025 年 5 月 13 日

目录

图表

1 作业题目

学生自选题目，使用 C++语言完成一个图形化的小程序。

1. 图形化平台不限，可以是 MFC、QT 等任何 C++图形化平台(不限制)。
2. 程序内容主题不限，可以是小游戏、小工具等。

2 开发软件

Visual Studio Code、Qt Creator

3 课题要求

1. 面向对象。
2. 单元测试。
3. 模型部分
4. 验证

4 主要流程

4.1 整体流程

本项目的整体流程旨在通过计算机视觉技术实时识别用户在摄像头前的手掌，并在识别到的手掌位置动态生成和展示粒子视觉特效。主要流程分为以下几个关键步骤：

4.1.1 程序初始化:

- 加载用户界面 (UI)，包括摄像头画面显示区域、控制按钮（如打开/关闭摄像头、开启/关闭粒子特效、拍照、录像等）。
- 初始化摄像头对象，准备从默认摄像头捕获视频流。
- 加载预训练的 YOLO（You Only Look Once）目标检测模型（包括 .cfg 配置文件和 .weights 权重文件）以及对应的类别名称文件 (.names)。
- 初始化粒子系统模块，设置粒子效果的默认参数（如粒子数量、生命周期、颜色生成方式、运动模式等）。
- 设置定时器，用于周期性地读取摄像头帧并进行处理。

4.1.2 摄像头操作与图像捕获:

- 用户通过 UI 按钮控制摄像头的开启与关闭。
- 当摄像头开启时，定时器启动，程序在每个时间间隔从摄像头捕获一帧原始图像。

4.1.3 手掌识别（基于 **YOLO** 模型）:

- 对捕获到的每一帧（或按一定间隔处理的帧），进行预处理以符合 YOLO 模型的输入要求（例如，缩放到 416x416 像素，归一化）。
- 将预处理后的图像输入到已加载的 YOLO 神经网络中进行前向传播，得到模型的原始输出。
- 对模型的原始输出进行后处理：

  解析输出层，提取候选目标框的位置、置信度以及类别 ID。

  应用置信度阈值过滤掉低置信度的候选框。

  应用非极大值抑制 (NMS) 消除冗余和重叠的检测框，得到最终的手掌检测结果。

4.1.4 粒子特效生成与更新:

- 如果粒子特效功能被用户开启，并且当前帧成功检测到手掌：

  获取检测到的手掌框的中心点作为粒子特效的触发中心。

  调用粒子系统的生成函数，在该中心点创建一批新的粒子，并为每个粒子设置初始属性（如位置、颜色、大小、生命周期、运动速度、扩散/旋转参数等）。这些新粒子被添加到活动粒子列表中。
- 在每一帧，调用粒子系统的更新函数：

  遍历所有活动粒子。

  更新每个粒子的生命周期，移除生命周期结束的粒子。

  更新存活粒子的状态（例如，根据预设的运动逻辑更新其位置、大小、颜色、旋转角度等）。

4.1.5 图像渲染与显示:

- 在原始摄像头帧上绘制 YOLO 检测到的手掌识别框和类别标签。
- 将所有更新后的活动粒子绘制到同一图像帧上。
- 对最终合成的图像帧进行镜像翻转。
- 将处理并绘制好特效的 cv::Mat 图像转换为 QImage 格式。
- 将 QImage 显示到 UI 界面的 QLabel 控件上，用户即可看到实时带有特效的画面。

4.1.6 附加功能实现:

- 拍照： 用户点击拍照按钮时，将当前显示的带有特效的 QImage 保存为本地图片文件。
- 录像： 用户点击开始录像按钮时，初始化 cv::VideoWriter；在录像状态下，将每一处理后的带特效的帧写入视频文件；点击停止录像时，释放写入器并保存视频。

4.1.7 程序结束： 关闭摄像头，释放所有资源。

4.2 算法或公式

本项目核心涉及的算法与公式主要体现在以下几个方面：

4.2.1 YOLO (You Only Look Once) 目标检测算法:

- 本项目使用 YOLO 模型进行手掌实时检测。
- 流程概述：

输入图像分割： 图像被划分为 S x S 的网格。

边界框预测： 每个网格单元预测 B 个边界框（Bounding Box）及其置信度分数，以及该网格单元包含目标的概率和目标属于各个类别的条件概率。

置信度计算：

```
Confidence_score = Pr(Object) IOU(truth, pred)
```

类别特定置信度：

```
Class_specific_confidence_score = Pr(Class_i | Object) * Pr(Object) *
IOU(truth, pred) = Pr(Class_i)  IOU(truth, pred)
```

非极大值抑制 (NMS)： 用于消除对同一目标的多个重叠检测框，保留置信度最高的框。其基本思想是：对于一组重叠度（通常用 IOU 衡量）超过阈值的检测框，只保留置信度最高的那个。

4.2.2 粒子系统运动与状态更新:

1. 径向扩散：

```
particle.current_ring_radius_from_center += particle.expansion_speed;
```

2. 角度旋转：

```
particle.angle_on_ring += particle_rotation_speed;
```

3. 位置计算：

```
particle.position.x = particle.base_center.x +
particle.current_ring_radius_from_center *
std::cos(particle.angle_on_ring);
particle.position.y = particle.base_center.y +
particle.current_ring_radius_from_center *
std::sin(particle.angle_on_ring);
```

4. 大小更新 (根据距离中心线性变化，并在生命周期末期渐隐)：

```
t_dist = (distance_from_center - reference_inner_radius) /
(reference_outer_radius - reference_inner_radius);
radius_based_on_distance = max_ball_dist * (1.0f - t_dist) + min_ball_dist
* t_dist;
```

5. 生命周期进度

```
life_progress = 1.0f - (static_cast<float>(particle.lifetime) /
particle.max_lifetime);
```

6. 生命末期渐隐

```
t_fade_life = (life_progress - fade_start_lifetime_ratio) / (1.0f -
fade_start_lifetime_ratio);
```

7. 生命末期大小

```
particle.current_radius_ball = radius_at_fade_start  (1.0f - t_fade_life)
+ final_fade_out_radius  t_fade_life;
```

8. 生命周期：

```
particle.lifetime--;
```

9. 颜色生成：

随机 RGB：

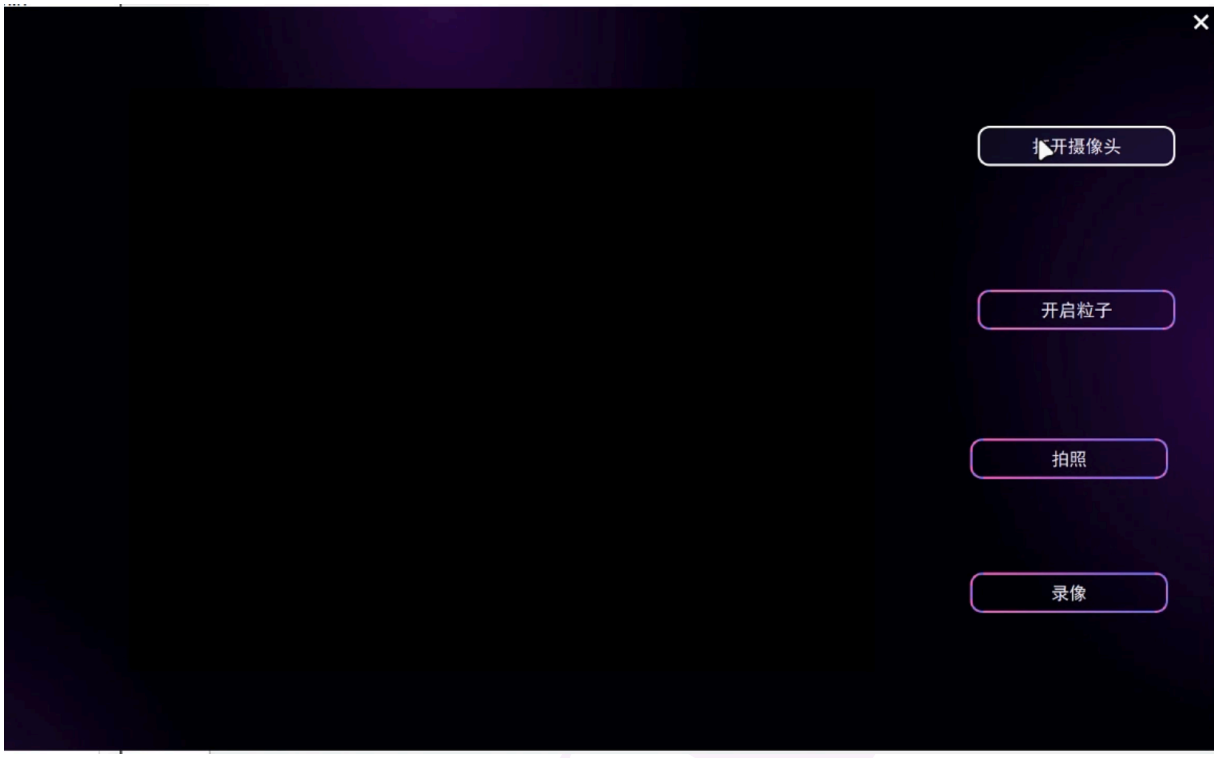
```
R = rand() % 256; G = rand() % 256; B = rand() % 256;
```

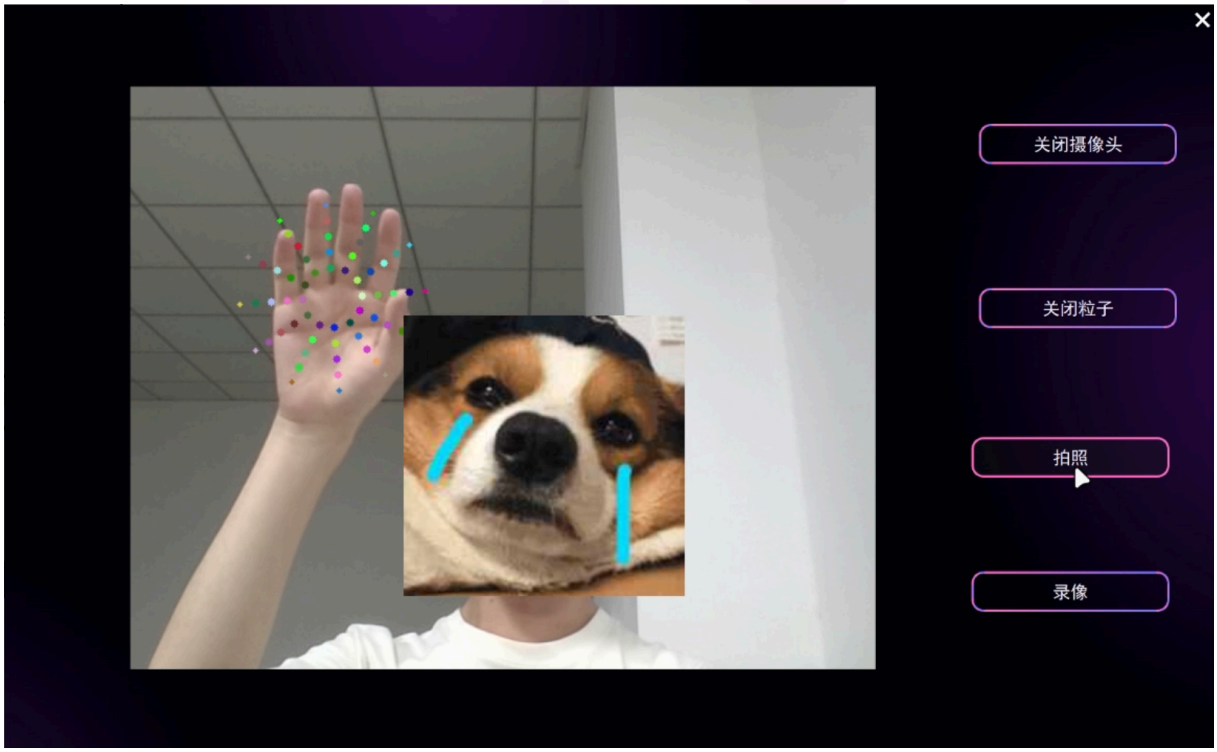
10. 图像镜像翻转：

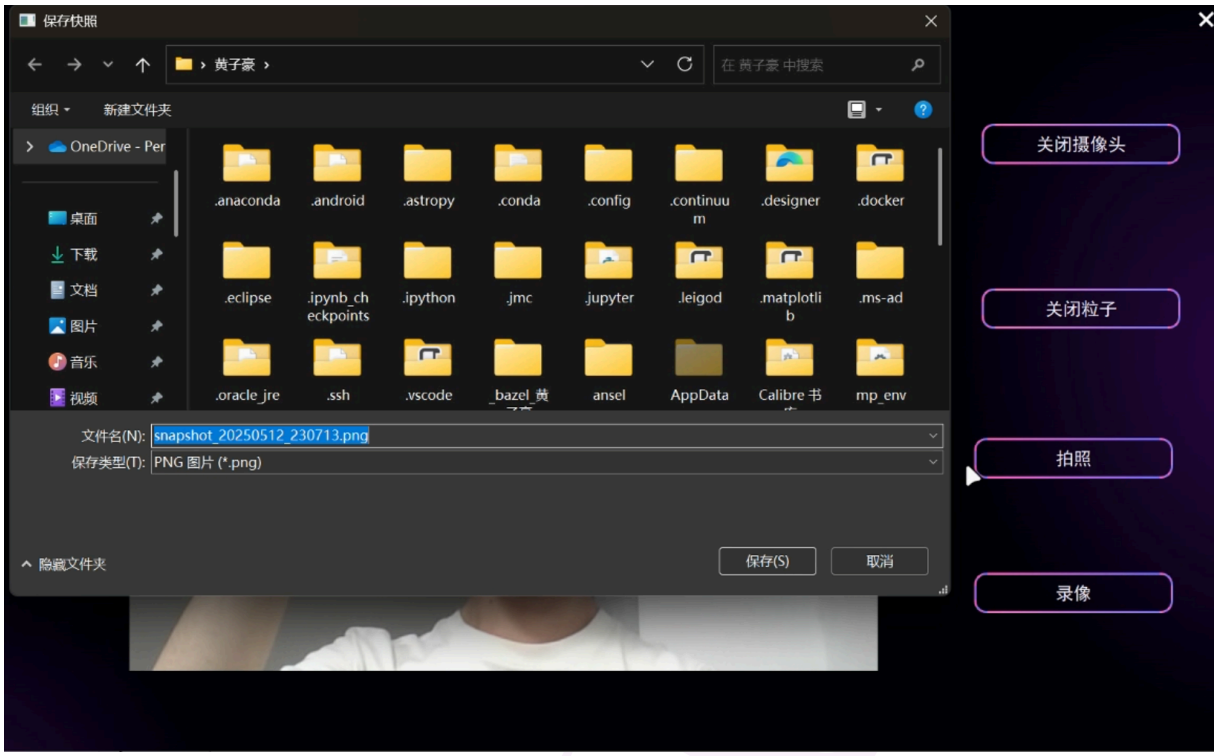
```
cv::flip(source_image, destination_image, 1);
```

11. 定时器： 使用 QTimer 以固定的时间间隔 30 毫秒触发 readfarme() 函数，实现视频流的逐帧处理。

5 测试


开摄像头
开启粒子
拍照
录像


关闭摄像头
关闭粒子
拍照
录像



6 收获

通过本次项目实践，我获得了以下几方面的收获和提升：

1. 掌握了使用 OpenCV 进行实时视频捕获与处理的基本流程： 包括打开摄像头、逐帧读取图像、图像转换 (cv::Mat 到 QImage) 以及在 Qt 界面中显示。
2. 深入理解并实践了 YOLO 目标检测模型的应用： 学会了如何加载预训练的 YOLO 模型，对输入图像进行预处理，执行前向传播，以及对模型输出进行后处理（置信度过滤、NMS）以获得最终的检测结果。
3. 掌握了粒子系统的基本设计与实现方法： 学习了如何创建和管理大量独立运动的粒子对象，包括粒子的生成、状态更新（位置、大小、颜色、生命周期）、运动逻辑（扩散、旋转）以及在图像上进行绘制。
4. 提升了 Qt GUI 编程能力： 熟悉了通过按钮与用户交互、更新 UI 元素（如按钮文本、QLabel 显示的图像）、使用 QTimer 实现周期性任务等 Qt 常用功能。
5. 增强了 C++编程和调试技能： 在实现过程中，通过不断地调试（例如使用 qDebug）、解决编译错误和逻辑错误，对 C++的语法、面向对象编程、以及问题排查能力有了更深的理解。
6. 学习了代码模块化和重构的重要性： 体验了将特定功能（如粒子系统）封装到独立的类中，可以提高代码的可读性、可维护性和复用性。
7. 体验了计算机视觉与图形特效结合的乐趣： 将严肃的目标检测技术与有趣的视觉特效相结合，直观地看到了代码产生的动态视觉成果，极大地激发了学习兴趣。

8. 对性能优化有了初步认识： 在实现过程中遇到了卡顿问题，促使我去思考和学习如何分析性能瓶颈（如 YOLO 推理耗时、粒子数量过多），并了解了一些基本的优化方向（如降低处理频率、减少计算量）。
9. 培养了耐心和解决问题的能力： 从环境配置、编译错误到逻辑调试，整个过程充满了挑战，但也锻炼了面对问题、分析问题、最终解决问题的能力和毅力。