创新创业 project2 说明文档

一、水印生成 generateWatermark

1. 初始化：若传入 message，通过 hash<string> 生成哈希值作为随机种子（保证相同消息生成相同水印）；若未传消息，使用构造函数的随机密钥 key 作为种子（生成随机水印）。

```
Mat generateWatermark(const string& message = "") {
    Mat watermark(watermarkSize, CV_8UC1);

    if (!message.empty()) {
        // 基于消息生成水印
        hash<string> hashFunc;
        size_t hashVal = hashFunc(message);
        rng.seed(hashVal);
    }
    else {
        // 生成随机水印
        rng.seed(key);
    }
```

2. 生成二值水印：

用 uniform_int_distribution 生成 0 或 1，映射为 0 或 255（CV_8UC1 格式）

```
uniform_int_distribution<int> dist(0, 1);
for (int i = 0; i < watermark.rows; ++i) {
    for (int j = 0; j < watermark.cols; ++j) {
        watermark.at<uchar>(i, j) = dist(rng) * 255;
    }
}
```

二、水印嵌入 embedWatermark

1. **图像预处理：**读入原图，转换为 YCrCb 色彩空间（仅修改亮度通道 Y，避免颜色失真）。

```
Mat embedWatermark(const string& imagePath, const Mat& watermark, const string& outputPath = "") {
    Mat image = imread(imagePath);
    if (image.empty()) {
        throw runtime_error("无法读取图像，请检查路径是否正确");
    }

    // 转换为YCrCb颜色空间
    Mat ycrcbImage;
    cvtColor(image, ycrcbImage, COLOR_BGR2YCrCb);
```

```
vector<Mat> channels;
split(ycrcbImage, channels);
Mat yChannel = channels[0].clone();
yChannel.convertTo(yChannel, CV_32FC1);

int h = yChannel.rows;
int w = yChannel.cols;
int wh = watermark.rows;
int ww = watermark.cols;
```

2. **分块处理（8x8 块）：**遍历图像的每个 8x8 块（DCT 标准块大小，平衡计算量和鲁棒性）。

```
for (int i = 0; i < h; i += 8) {
    for (int j = 0; j < w; j += 8) {
        // 确保块大小为8x8
        if (i + 8 > h || j + 8 > w) continue;

        Mat block = yChannel(Rect(j, i, 8, 8));
        Mat dctBlock;
```

3. **DCT 变换：**对块执行 dct()，转换到频率域。

```
dct(block, dctBlock);
```

4. **嵌入水印：**选择 (5,5) 中频系数（低频影响图像主体，高频易被滤波去除，中频兼顾鲁棒和不可见性）；根据水印值（0 或 255），调整系数：dctBlock.at<float>(5,5) += alpha * (1 或 -1) 。

```
int wi = (i / 8) % wh;
int wj = (j / 8) % ww;
uchar wmVal = watermark.at<uchar>(wi, wj);
dctBlock.at<float>(5, 5) += alpha * (wmVal > 0 ? 1 : -1);
```

5. **逆变换与合成：**对块执行 idct() 转回空间域，合并通道后转换为 BGR 格式，保存或返回。

```
        // 逆DCT变换
        idct(dctBlock, block);
    }
}

// 合并通道并转换回BGR
yChannel.convertTo(channels[0], CV_8UC1);
merge(channels, ycrcbImage);
Mat watermarkedImage;
cvtColor(ycrcbImage, watermarkedImage, COLOR_YCrCb2BGR);
```

三、水印提取 extractWatermark

1. **图像预处理：**分别读入原始图和含水印图，转换为 YCrCb 并提取 Y 通道。

```
// 转换为YCrCb颜色空间
Mat originalYCrCb, watermarkedYCrCb;
cvtColor(originalImage, originalYCrCb, COLOR_BGR2YCrCb);
cvtColor(watermarkedImage, watermarkedYCrCb, COLOR_BGR2YCrCb);

// 提取Y通道
vector<Mat> originalChannels, watermarkedChannels;
split(originalYCrCb, originalChannels);
split(watermarkedYCrCb, watermarkedChannels);

Mat originalY = originalChannels[0].clone();
Mat watermarkedY = watermarkedChannels[0].clone();

originalY.convertTo(originalY, CV_32FC1);
watermarkedY.convertTo(watermarkedY, CV_32FC1);

int h = originalY.rows;
int w = originalY.cols;
```

2. 分块 DCT 变换（定位水印嵌入位置）

（1）遍历所有 8x8 块：按行和列以 8 为步长遍历图像，确保每个块的尺寸为 8x8（边缘不足 8 像素的块跳过，避免尺寸不匹配）：

```
if (i + 8 > h || j + 8 > w) continue;
```

（2）提取对应块并执行 DCT：从原始图像和含水印图像的 Y 通道中，提取当前位置的 8x8 块，并分别执行 DCT 变换（转换到频率域）：

```
Mat originalBlock = originalY(Rect(j, i, 8, 8));
Mat watermarkedBlock = watermarkedY(Rect(j, i, 8, 8));

Mat originalDCT, watermarkedDCT;
dct(originalBlock, originalDCT);
dct(watermarkedBlock, watermarkedDCT);
```

3. 计算系数差异（提取水印信号）

（1）计算 (5,5) 位置系数差：差异值 diff 等于“含水印块的 (5,5) 系数”减去“原始块的 (5,5) 系数”，这个差异直接反映了嵌入的水印信号：

```
double diff = watermarkedDCT.at<float>(5, 5) - originalDCT.at<float>(5, 5);
```

若嵌入时水印位为 1（255），则差异 diff ≈ +alpha；
若嵌入时水印位为 0，则差异 diff ≈ -alpha（嵌入逻辑：dctBlock += alpha * (1 或 -1)）。

（2）映射到水印坐标：每个 8x8 块对应水印矩阵中的一个位置，通过块的索引计算水印坐标（与嵌入时的映射规则完全一致，确保位置对应）：

```
int wi = (i / 8) % watermarkSize.height;
int wj = (j / 8) % watermarkSize.width;
```

（3）二值化判定水印位：根据差异值 diff 的符号（或与阈值比较），判定当前水印位是 1 还是 0：

```
extractedWatermark.at<uchar>(wi, wj) = (diff > threshold) ? 255 : 0;
```

通常阈值设为 0（因为嵌入时 diff 符号明确）；
若图像受噪声干扰，可适当调整阈值（如 0.1*alpha）过滤噪声。

4. 生成提取的水印矩阵

所有块处理完成后，得到一个与原始水印尺寸相同的二值矩阵（CV_8UC1 格式，0 或 255），即提取的水印。

四、鲁棒性测试函数

1.核心图像处理函数

（1）翻转图像 flipImage

实现水平 / 垂直 / 双向翻转，模拟图像编辑中的翻转操作。

```
Mat flipImage(const Mat& image, int flipCode) {
    Mat result;
    flip(image, result, flipCode);
    return result;
}
```

（2）平移图像 translateImage

沿 X/Y 方向平移图像，模拟图像裁剪偏移或错位。

```
Mat translateImage(const Mat& image, int x, int y) {
    Mat result;
    Mat translation = (Mat_<double>(2, 3) << 1, 0, x, 0, 1, y);
    warpAffine(image, result, translation, image.size());
    return result;
}
```

块的绝对位置偏移，但相对位置关系不变。若平移量是 8 的倍数（块大小），对提取影响较小；否则会导致块边缘错位，引入提取误差。

（3）裁剪图像 cropImage

截取图像的中间区域，模拟图像被裁剪（丢失边缘信息）。参数意义：(x1,y1) 为左上角坐标，(x2,y2) 为右下角坐标（需确保在图像范围内）。对水印的影响：直接丢失边缘的 8x8 块，导致对应位置的水印信息永久丢失。裁剪比例越大（如超过 30%），相似度可能骤降，考验水印的冗余设计（如重复嵌入）。

```
Mat cropImage(const Mat& image, int x1, int y1, int x2, int y2) {
    return image(Rect(x1, y1, x2 - x1, y2 - y1)).clone();
}
```

（4）调整对比度 adjustContrast

通过线性变换调整图像亮度 / 对比度，模拟后期调色操作。对水印的影响：Y 通道（亮度）整体缩放，导致 DCT 系数成比例变化。若水印嵌入时的 alpha 较大（信号强），则对比度调整对差异值 diff 的影响较小，提取更稳定。

```
Mat adjustContrast(const Mat& image, double alpha, int beta = 0) {
    Mat result;
    image.convertTo(result, -1, alpha, beta);
    return result;
}
```

（5）添加高斯噪声 addNoise

向图像添加高斯噪声，模拟传输 / 压缩中的信号干扰。参数意义：mean 噪声均值（通常为 0），var 噪声方差（越大噪声越强）。对水印的影响：在频率域引入随机干扰，可能掩盖水印信号（diff 值被噪声淹没）。DCT 中频系数对中等噪声有一定抵抗力，但强噪声会显著降低相似度。

```
Mat addNoise(const Mat& image, double mean = 0, double var = 0.001) {
    Mat result, imageFloat;
    image.convertTo(imageFloat, CV_32FC3);
    imageFloat /= 255.0;

    Mat noise(image.size(), CV_32FC3);
    randn(noise, mean, sqrt(var));

    result = imageFloat + noise;
    result = min(result, 1.0);
    result = max(result, 0.0);
    result *= 255.0;
    result.convertTo(result, CV_8UC3);

    return result;
```

（6）旋转图像 rotateImage

绕中心旋转图像，模拟视角变化（如照片旋转）。对水印的影响：块发生形变（不再是严格 8x8 矩形），且坐标映射关系被破坏。旋转角度越大，块的形变越严重，水印提取难度越大（尤其是非 90 度倍数的旋转）。

```
Mat rotateImage(const Mat& image, double angle) {
    Mat result;
    Point2f center(image.cols / 2.0f, image.rows / 2.0f);
    Mat rotation = getRotationMatrix2D(center, angle, 1.0);
    warpAffine(image, result, rotation, image.size());
    return result;
```

2.主程序 main 函数流程

（1）初始化水印系统

Size(32,32)：水印尺寸（32x32 像素，共 1024 位信息）；

0.08：嵌入强度 alpha（值越大水印越鲁棒，但图像失真越明显）。

```
WatermarkSystem watermarkSystem(Size(32, 32), 0.08);
```

（2）生成水印

传入字符串作为种子，确保相同字符串生成相同水印（可用于版权标识）；

水印为 CV_8UC1 格式（单通道），像素值为 0 或 255（二值水印）。

```
Mat watermark = watermarkSystem.generateWatermark("MySecretWatermark123");
```

（3）嵌入水印并保存

读取原图，嵌入水印后返回含水印图像，并保存到 watermarked_image.jpg。

```
string originalImagePath = "original_image.jpg"; // 替换为你的图像路径
Mat watermarkedImage = watermarkSystem.embedWatermark(originalImagePath, watermark, "watermarked_image.jpg");
cout << "水印嵌入成功！" << endl;
```

（4）提取水印并计算基准相似度

从“未受攻击”的含水印图像中提取水印，计算与原始水印的相似度（理想情况应接近 100%）。

```
Mat extractedWatermark = watermarkSystem.extractWatermark(originalImagePath, watermarkedImage);
double similarity = watermarkSystem.calculateSimilarity(watermark, extractedWatermark);
cout << "原始图像水印提取相似度: " << fixed << setprecision(2) << similarity << "%" << endl;
```

（5）可视化水印对比

创建窗口显示“原始水印”和“提取的水印”，直观对比效果：

```
namedWindow("水印对比", WINDOW_NORMAL);
Mat watermarkDisplay(180, 340, CV_8UC3, Scalar(255, 255, 255));
putText(watermarkDisplay, "原始水印", Point(10, 15), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 0), 1);
Mat resizedOrig, resizedExt;
resize(watermark, resizedOrig, Size(150, 150));
cvtColor(resizedOrig, resizedOrig, COLOR_GRAY2BGR);
resizedOrig.copyTo(watermarkDisplay(Rect(10, 20, 150, 150)));

putText(watermarkDisplay, format("提取的水印 (相似度: %.2f%%)", similarity),
    Point(180, 15), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0, 0, 0), 1);
resize(extractedWatermark, resizedExt, Size(150, 150));
cvtColor(resizedExt, resizedExt, COLOR_GRAY2BGR);
resizedExt.copyTo(watermarkDisplay(Rect(180, 20, 150, 150)));

imshow("水印对比", watermarkDisplay);
waitKey(0);
```

（6）执行鲁棒性测试并输出结果

对所有测试用例计算相似度，打印结果（如“旋转 (30 度): 78.32%”），评估算法抗攻击能力。

```
    cout << "\n开始鲁棒性测试..." << endl;
    map<string, double> testResults = watermarkSystem.performRobustnessTests(
        originalImagePath, watermarkedImage, watermark);

    // 打印所有测试结果
    cout << "\n鲁棒性测试结果:" << endl;
    for (auto& pair : testResults) {
        if (pair.second == -1) {
            cout << pair.first << ": 失败" << endl;
        }
        else {
            cout << pair.first << ": " << fixed << setprecision(2) << pair.second << "%" << endl;
        }
    }


tch (const exception& e) {
    cerr << "发生错误: " << e.what() << endl;
    return 1;
```