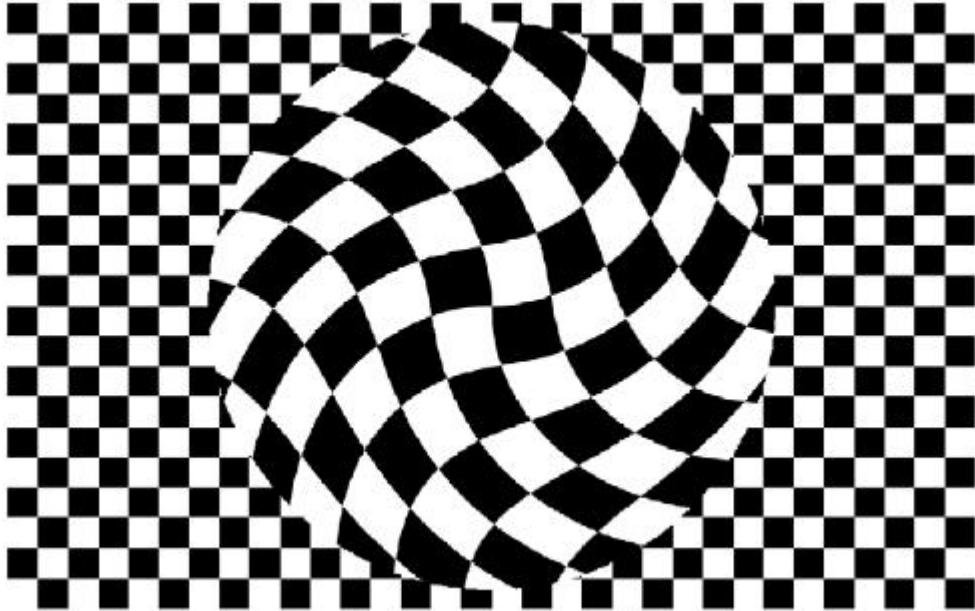


## 实验六 角点检测与圆拟合

2151094 宋正非

### 1. 实验目标

创建角点关键点检测器。使用角点关键点和 RANSAC 从下图中找到一个圆。



准备工作：

```
git clone https://git.tongji.edu.cn/ipmv/examples/lab_5.git
```

### 2. 实验流程

实验主要流程包含以下部分：使用高斯滤波器的导数计算图像梯度；计算图像 A、B、C；使用 A、B 和 C 计算整幅图像的角点度量；对角点度量图像进行阈值处理并寻找局部最大值。

#### 1) 滤波核--Filter kernels

在 filters.cpp 文件中已有 `cv::Mat create1DGaussianKernel(float sigma, int radius = 0)` 函数，其用于生成一维高斯核，需要完成 `cv::Mat create1DDerivedGaussianKernel(float sigma, int radius = 0)` 用来生成一维导数高斯核。高斯导数公式如下所示，

$$\frac{\partial}{\partial x} G_\sigma(x) = -\frac{x}{\sigma^2} G_\sigma(x)$$

```
cv::Mat create1DDerivedGaussianKernel(float sigma, int radius)
{
    if (radius <= 0) radius = static_cast<int>(std::ceil(3.5f * sigma));

    // Todo: Step 1: Use the function above and finish this function.
    cv::Mat kernel;

    kernel = create1DGaussianKernel(sigma, radius);
    const int length = kernel.rows;
```

```
const float factor = -1.0f/(sigma*sigma);

for (int i = 0; i < length; ++i)
{
    const auto x = static_cast<float>(i - radius);
    kernel.at<float>(i) *= x *factor;
}
return kernel;
}
```

## 2) 计算图像梯度--Computer the image gradients

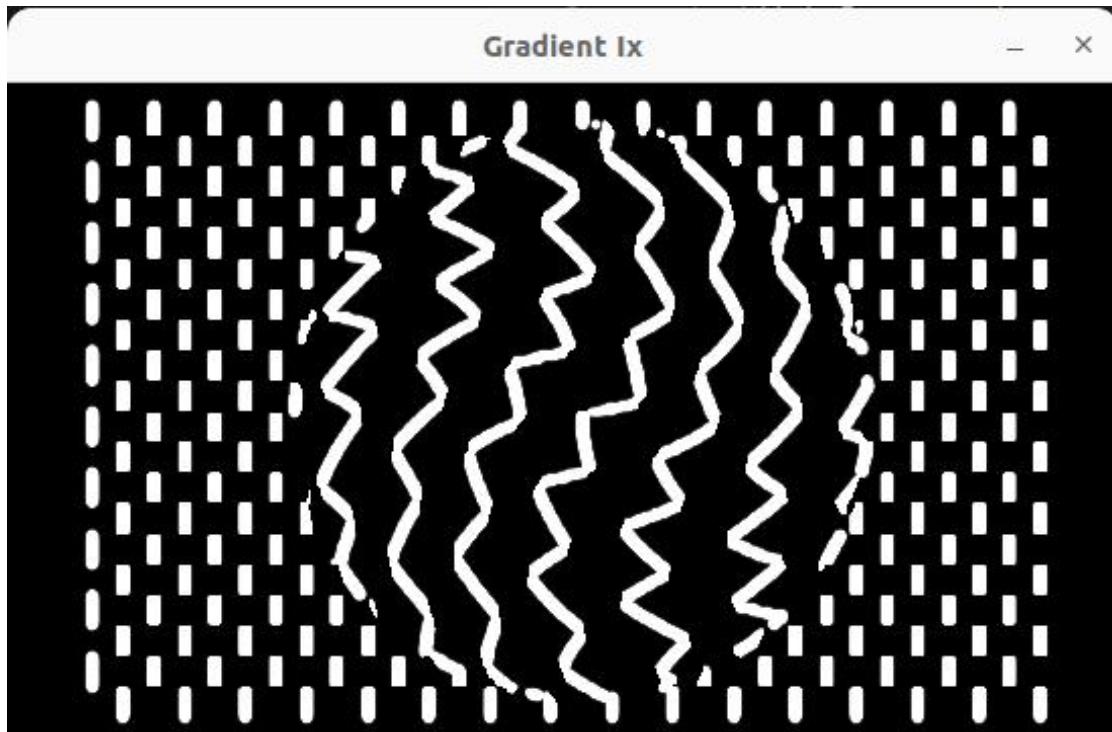
接下来，完善 CornerDetector.cpp。

完善 CornerDetector::detect 函数的实现。使用 g\_kernel\_ 和 dg\_kernel 预估图像梯度 Ix 和 Iy。

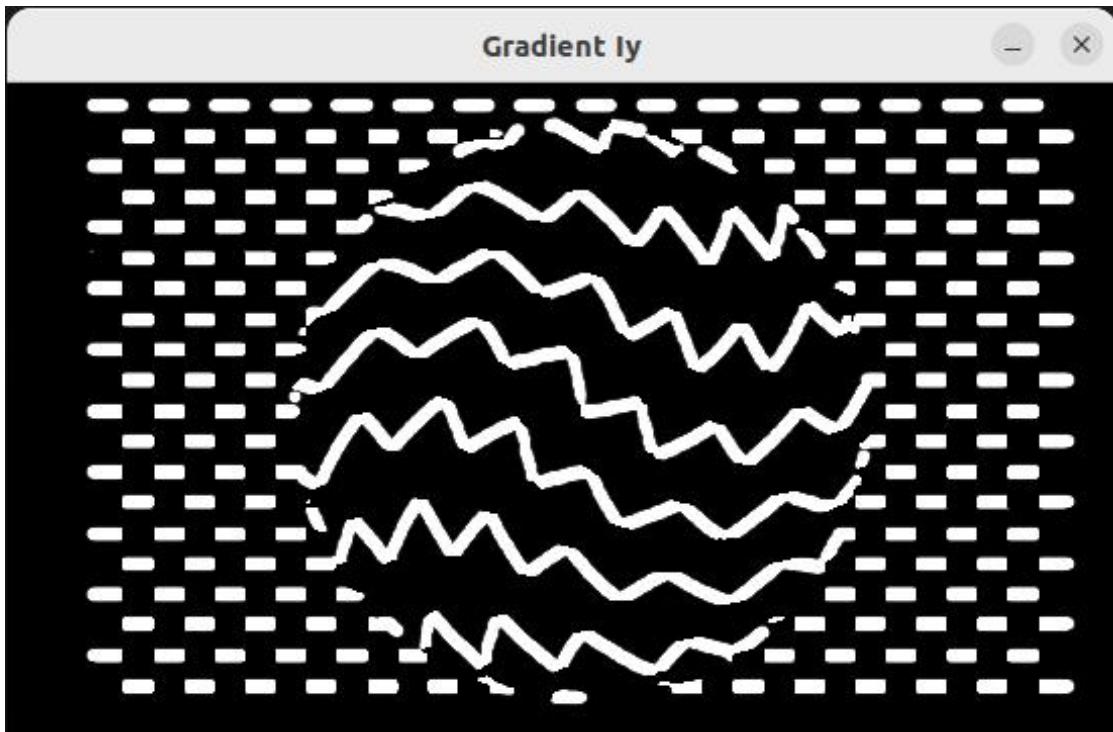
```
// Estimate image gradients Ix and Iy using g_kernel_ and dg_kernel.
// Todo: Step 2: Estimate image gradients Ix and Iy using g_kernel_ and dg_kernel_.

cv::Mat Ix;
cv::Mat Iy;
cv::sepFilter2D(image, Ix, CV_32F, dg_kernel_, g_kernel_);
cv::sepFilter2D(image, Iy, CV_32F, g_kernel_, dg_kernel_);
```

Ix 如下图所示。



Iy 如下图所示。



### 3) 通过计算 A、B、C 隐式计算 M--Compute M implicitly by computing A, B, C

完善 CornerDetector::detect。根据图像梯度  $I_x$  和  $I_y$  计算  $A, B, C$ 。将  $A, B, C$  与高斯窗口滤波器  $win\_kernel\_$  进行卷积。公式如下。

$$\begin{aligned}
 A &= \sum_{x,y} w(x,y) I_x^2 \\
 B &= \sum_{x,y} w(x,y) I_x I_y \\
 C &= \sum_{x,y} w(x,y) I_y^2 \\
 M &= \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \sum_{x,y} w(x,y) \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix}
 \end{aligned}$$

使用 `cv::sepFilter2D(A, A, -1, ?, ?)` 函数实现上述计算。

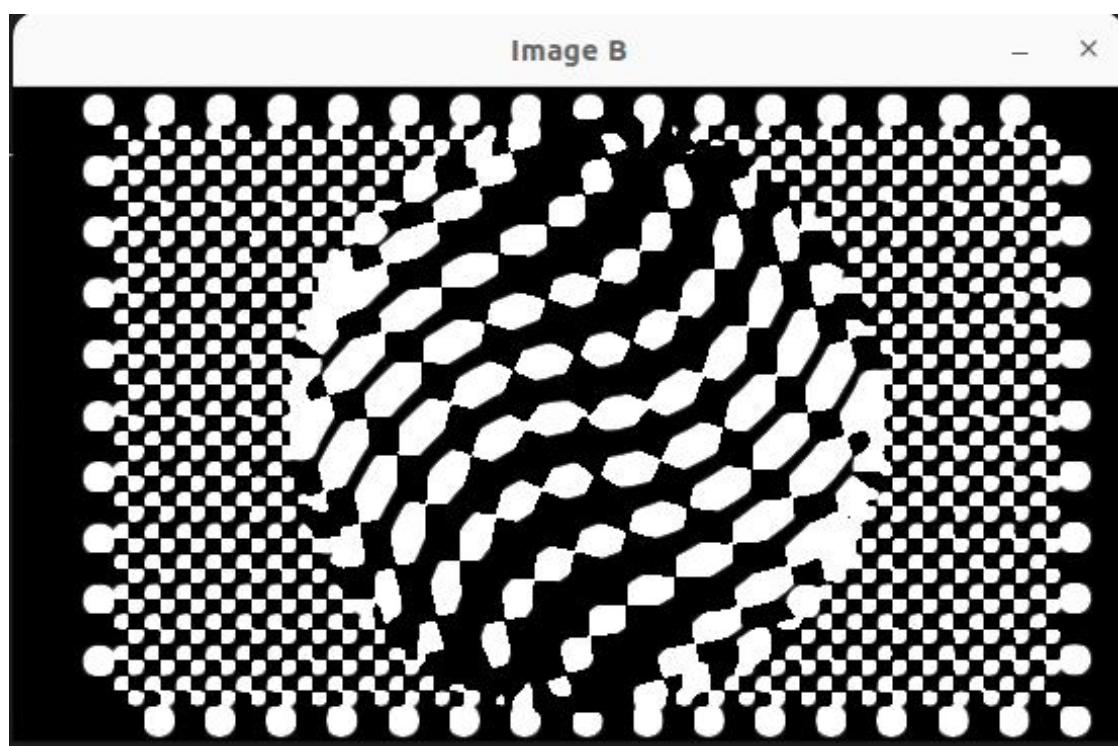
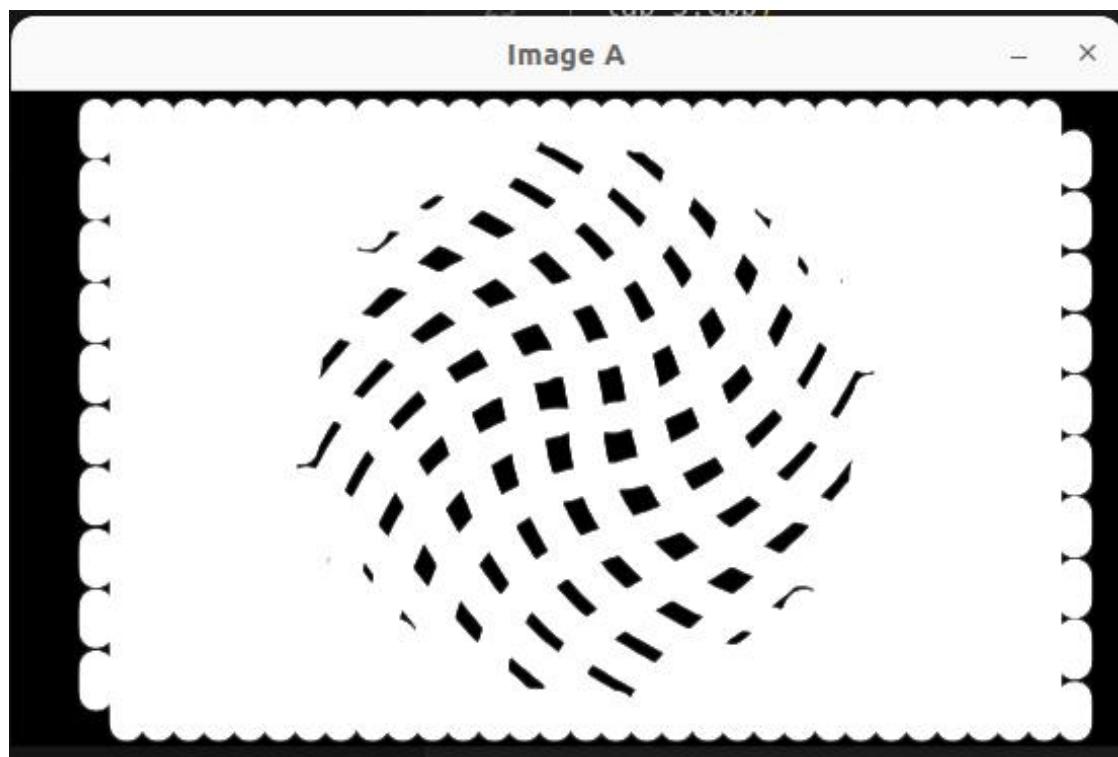
```

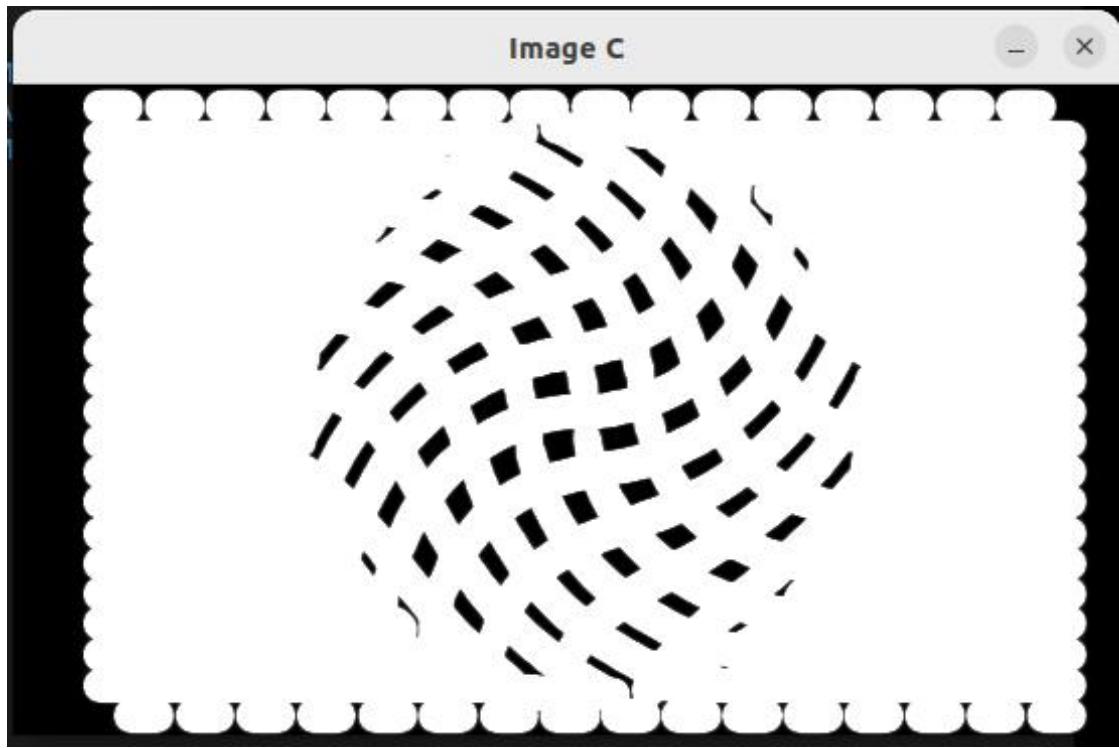
// Compute the elements of M; A, B and C from Ix and Iy.
// Todo: Step 3: Compute the elements of M; A, B and C from Ix and Iy.
cv::Mat A;
cv::Mat B;
cv::Mat C;
A = Ix.mul(Ix);
B = Ix.mul(Iy);
C = Iy.mul(Iy);

// Apply the windowing gaussian win_kernel_ on A, B and C.
// Todo: Step 3: Apply the windowing gaussian.
cv::sepFilter2D(A, A, -1, win_kernel_, win_kernel_);
cv::sepFilter2D(B, B, -1, win_kernel_, win_kernel_);
cv::sepFilter2D(C, C, -1, win_kernel_, win_kernel_);

```

经过处理，得到如下结果。





#### 4) 实现角点指标--Implement the corner metrics

实现如下计算。

CornerDetector::harris\_metric

$$f = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2 = \det(M) - \alpha \text{trace}(M)^2$$

CornerDetector::harmonic\_mean\_metric

$$f = \frac{\lambda_1\lambda_2}{\lambda_1 + \lambda_2} = \frac{\det(M)}{\text{trace}(M)}$$

CornerDetector::min\_eigen\_metric

$$\lambda = \frac{1}{2} \left[ (A + C) \pm \sqrt{4B^2 + (A - C)^2} \right]$$

// Compute corner response.

// Todo: Step 4: Finish all the corner response functions.

cv::Mat response;

switch (metric\_type)

{

case CornerMetric::harris:

response = harrisMetric(A, B, C); break;

case CornerMetric::harmonic\_mean:

response = harmonicMeanMetric(A, B, C); break;

case CornerMetric::min\_eigen:

response = minEigenMetric(A, B, C); break;

}

cv::Mat CornerDetector::harrisMetric(cv::Mat& A, cv::Mat& B, cv::Mat& C) const

{

// Compute the Harris metric for each pixel.

// Todo: Step 4: Finish all the corner response functions.

```

const float alpha = 0.06f;
cv::Mat det_M = A.mul(C)-B.mul(B);
cv::Mat trc_M = A + C;
return det_M - alpha*(trc_M).mul(trc_M);
// return cv::Mat();
}

cv::Mat CornerDetector::harmonicMeanMetric(cv::Mat& A, cv::Mat& B, cv::Mat& C) const
{
    // Compute the Harmonic mean metric for each pixel.
    // Todo: Step 4: Finish all the corner response functions.
    cv::Mat det_M = A.mul(C)-B.mul(B);
    cv::Mat trc_M = A + C;
    return det_M.mul(1.f/trc_M);
    // return cv::Mat();
}

cv::Mat CornerDetector::minEigenMetric(cv::Mat& A, cv::Mat& B, cv::Mat& C) const
{
    // Compute the Min. Eigen metric for each pixel.
    // Todo: Step 4: Finish all the corner response functions.
    cv::Mat root;
    cv::sqrt(4.f*B.mul(B)+(A-C).mul(A-C), root);
    return 0.5f*((A+C)-root);
    // return cv::Mat();
}

```

## 5) 扩大图像以找到局部最大值--Dilate image to find local maxima

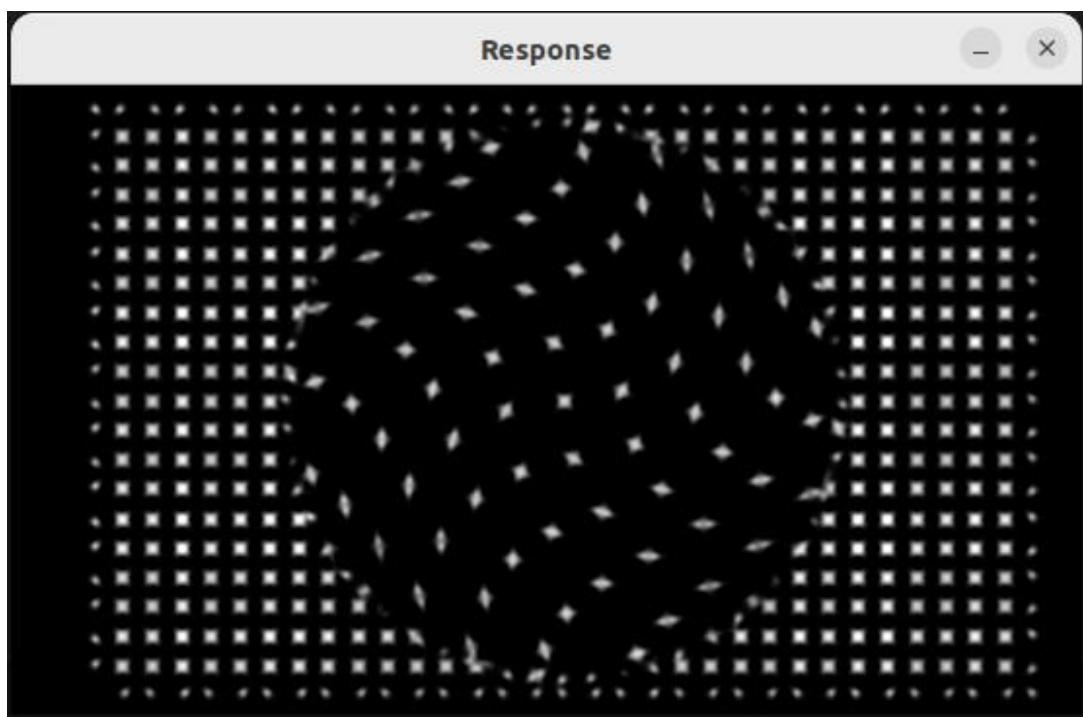
扩大图像，使每个像素等于邻域中的最大值，使用 `cv::dilate(response, local_max, ...)`;

```

// Todo: Step 5: Dilate image to make each pixel equal to the maximum in the neighborhood.
cv::Mat local_max;
cv::dilate(response, local_max, cv::Mat{});

```

响应图像如下所示。



## 6) 计算指标阈值--Compute the metric threshold

使用 cv::minMaxLoc(...)查找最大响应， 使用 max\_val \* quality\_level\_计算阈值。

```
// Todo: Step 6: Compute the threshold.  
// Compute the threshold by using quality_level_ on the maximum response.  
double max_val(0.0);  
cv::minMaxLoc(response, nullptr, &max_val);  
const float threshold = static_cast<float>(max_val) * quality_level_;
```

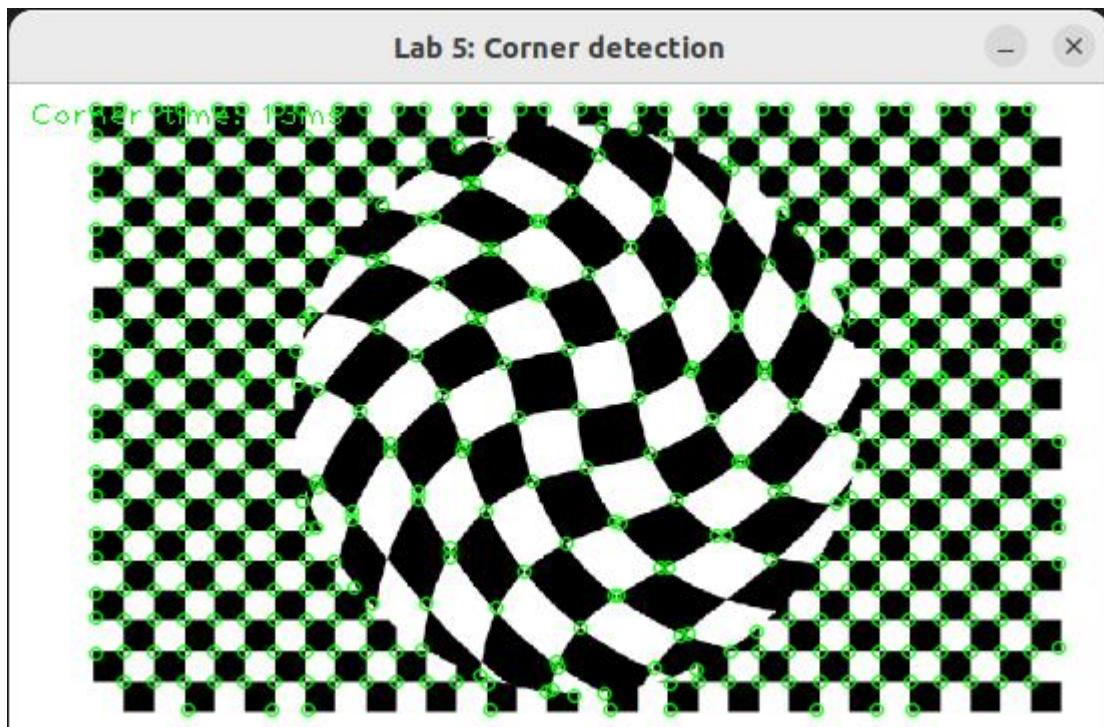
## 7) 提取阈值以上的局部最大值--Extract local maxima above threshold

根据已计算的最大响应值和阈值，找到图像中的局部极大值，并对最大响应值进行阈值处理。找到局部极大值后，提取每个检测到的角点的位置信息。

```
// Todo: Step 7: Extract local maxima above threshold.  
cv::Mat is_strong_and_local_max = (response > threshold) & (response == local_max); // =  
response > threshold and response == local_max  
std::vector<cv::Point> max_locations;  
cv::findNonZero(is_strong_and_local_max, max_locations);  
local_max 图像如下。
```



## 8) 角点检测器完成--Corner Detector Done



### 3. 补充实验

#### 1) Adaptive Non-Maximal Suppression

在常规的角度检测中（如 Harris），我们通常根据响应值选择强角点。但是这些角点可能会非常密集地分布在图像的某些区域，导致空间分布不均匀。ANMS 的目标是在所有响应值都大于某一阈值的角点中，选择一组响应强且分布均匀的点。

在上述代码的基础上增加 ANMS 部分来得到点的分布。CornerDetector::detect 中加入如下代码。

```
// Apply ANMS to get well-distributed corners.  
key_points = applyANMS(key_points, max_features );  
  
if (do_visualize_ )  
{  
    if (!Ix.empty()) { cv::imshow("Gradient Ix", Ix); };  
    if (!Iy.empty()) { cv::imshow("Gradient Iy", Iy); };  
    if (!A.empty()) { cv::imshow("Image A", A); };  
    if (!B.empty()) { cv::imshow("Image B", B); };  
    if (!C.empty()) { cv::imshow("Image C", C); };  
    if (!response.empty()) { cv::imshow("Response", response / (0.9 * max_val)); };  
    if (!is_strong_and_local_max.empty()) { cv::imshow("Local max",  
is_strong_and_local_max); };  
  
    cv::Mat vis;  
    cv::cvtColor(image, vis, cv::COLOR_GRAY2BGR);  
    for (const auto& kp : key_points)  
        cv::circle(vis, kp.pt, 2, cv::Scalar(0, 255, 0), -1);  
    cv::imshow("ANMS keypoints", vis);  
}  
  
return key_points;
```

增加 CornerDetector::applyANMS 函数。

```
// --- ANMS Implementation ---

std::vector<cv::KeyPoint> CornerDetector::applyANMS(const std::vector<cv::KeyPoint>&
keypoints, int max_pts) const
{
    if (keypoints.size() <= max_pts) return keypoints;

    std::vector<cv::KeyPoint> sorted = keypoints;
    std::sort(sorted.begin(), sorted.end(),
        [] (const cv::KeyPoint& a, const cv::KeyPoint& b) {
            return a.response > b.response;
        });

    std::vector<float> radii(sorted.size(), std::numeric_limits<float>::max());

    for (size_t i = 0; i < sorted.size(); ++i)
    {
        for (size_t j = 0; j < i; ++j)
        {
            if (sorted[j].response > sorted[i].response)
            {
                float dx = sorted[i].pt.x - sorted[j].pt.x;
                float dy = sorted[i].pt.y - sorted[j].pt.y;
                float dist_sq = dx * dx + dy * dy;
                if (dist_sq < radii[i]) radii[i] = dist_sq;
            }
        }
    }

    std::vector<std::pair<float, int>> radius_idx;
    for (size_t i = 0; i < radii.size(); ++i)
        radius_idx.emplace_back(radii[i], static_cast<int>(i));

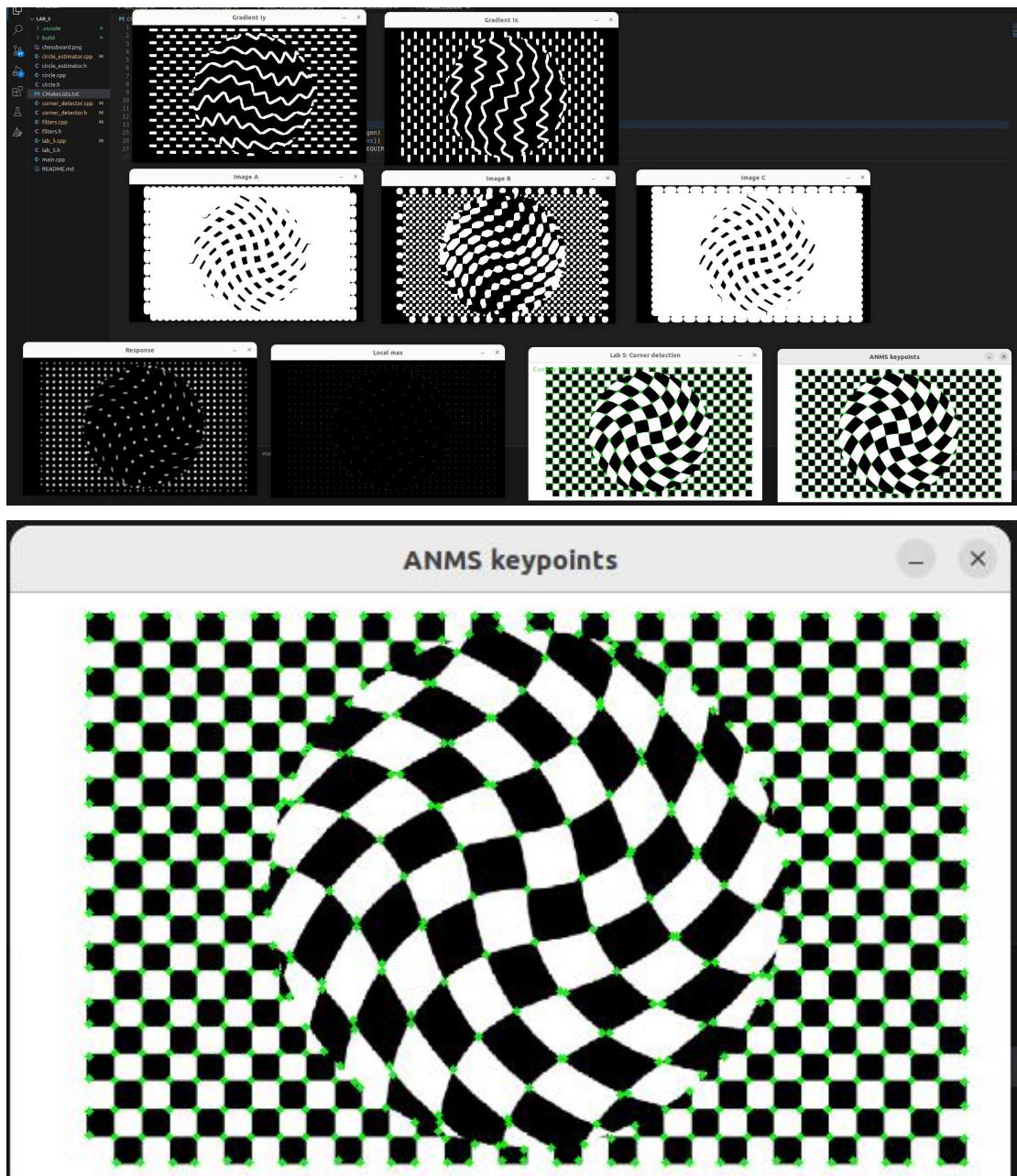
    std::sort(radius_idx.rbegin(), radius_idx.rend()); // descending

    std::vector<cv::KeyPoint> anms_keypoints;
    for (int i = 0; i < std::min(max_pts, static_cast<int>(radius_idx.size())); ++i)
        anms_keypoints.push_back(sorted[radius_idx[i].second]);

    return anms_keypoints;
```

同时，需要添加到 corner\_detector.h 的接口。包含 `std::vector<cv::KeyPoint>`  
`applyANMS(const std::vector<cv::KeyPoint>& keypoints, int max_pts) const;` 以及 `int max_features_`；

得到如下结果。



## 2) 处理相机获取的实时图片

在 void lab\_5() 函数开头添加如下代码来打开相机。

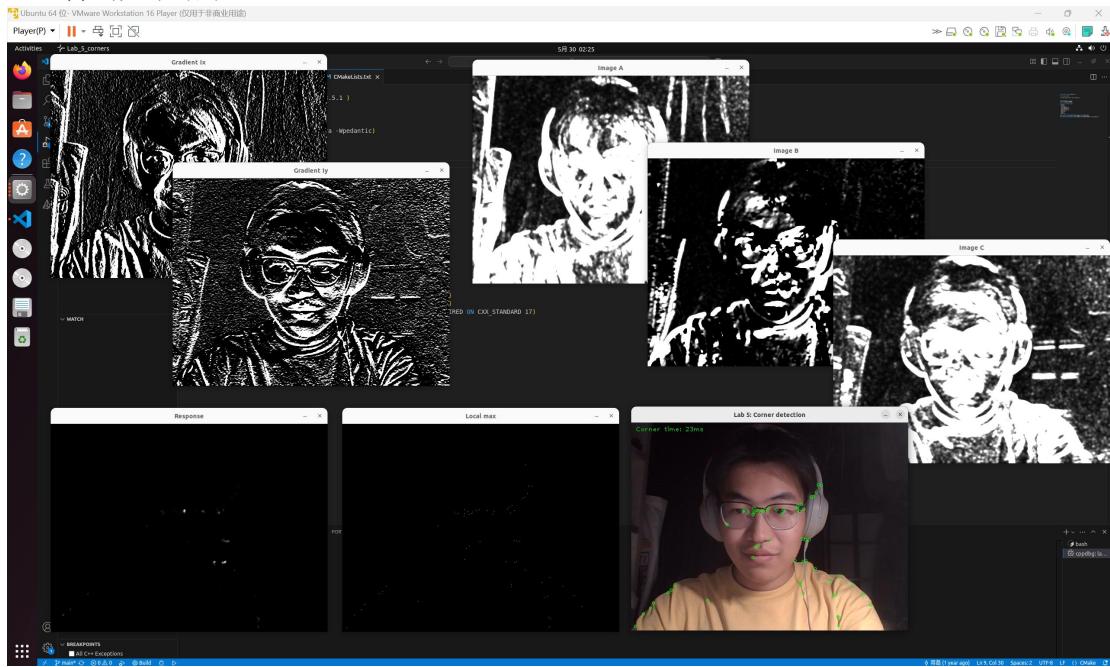
```
// Open video stream from camera.
const int camera_id = 0; // Should be 0 or 1 on the lab PCs.
cv::VideoCapture cap(camera_id);
if (!cap.isOpened())
{
    throw std::runtime_error("Could not open camera");
}
```

在循环中找到 imread(), 把从图像读取改为从相机读取。

```
// Read a frame from the camera.
cv::Mat frame;
cap >> frame;
```

```
if (frame.empty())
{ break; }
```

得到如下结果。



### 3) 圆拟合

前往 CircleEstimator::ransacEstimator，移除 break，实现正确的测试。

代码修改如下。

```
// Check if this estimate gave a better result.  
// Todo: Step 8: Remove break and perform the correct test!  
// break; // Remove!  
// Check if this estimate gave a better result.  
if (tst_num_inliers > best_num_inliers)  
{  
    // Update circle with largest inlier set.  
    best_circle = tst_circle;  
    best_num_inliers = tst_num_inliers;  
    best_is_inlier = is_inlier;  
  
    // Update max iterations based on the inlier ratio.  
    double inlier_ratio = static_cast<double>(best_num_inliers) / static_cast<double>(pts.cols());  
    if (inlier_ratio > 0.0)  
    {  
        max_iterations = static_cast<int>(std::log(1.0 - p_) / std::log(1.0 - std::pow(inlier_ratio, 3)));  
    }  
}
```

得到结果如下。

