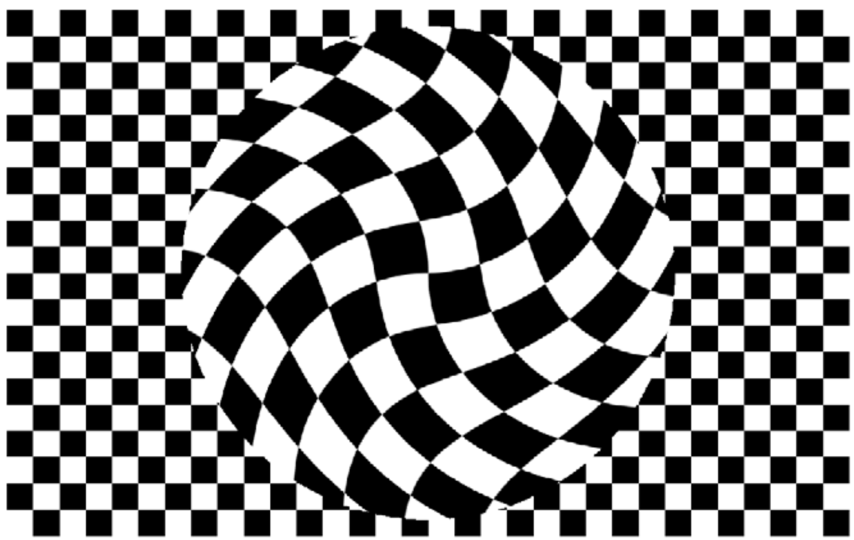
Lab 3 – Circles from corner detection

Topics

- Create your own corner keypoint detector
- Use corner keypoints and RANSAC to find a circle!



CornerDetector

Main steps

- Compute the image gradients using derivative of Gaussian filters

- Compute the images A, B, C
  - Elementwise products of the gradients
  - Apply windowing by convolving these with a (bigger) Gaussian

- Use A, B, and C to compute a corner metric for the entire image
  - λ_{min}, Harris, Harmonic Mean, other?

- Threshold the corner metric image and find local maxima
  - Morphological operations
  - Logical operations

Step 1: Filter kernels

- Take a look at this function in filters.cpp:

```
cv::Mat create1DGaussianKernel(float sigma, int radius = 0)
```

  - Returns a 1D filter

- Finish this function:

```
cv::Mat create1DDerivatedGaussianKernel(float sigma, int radius = 0)
```

  - Use the Gaussian filter from above!
  - Derivative of Gaussian:

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

Step 2: Compute the image gradients

- We are now ready to finish `CornerDetector`

- Go to `CornerDetector::detect`

- Use the 1D filters to compute the gradient images I_x and I_y
  - `g_kernel_`
  - `dg_kernel_`

  - `cv::sepFilter2D(image, Ix, CV_32F, ?, ?);`
  - `cv::sepFilter2D(image, Iy, CV_32F, ?, ?);`

Step 3: Compute *M* implicitly by computing *A*, *B*, *C*

`CornerDetector::detect`

- Compute *A*, *B* and *C* from the image gradients I_x and I_y

- Convolve *A*, *B*, and *C* with the Gaussian windowing filter
  - `win_kernel_`
  - `cv::sepFilter2D(A, A, -1, ?, ?)`

$$A = \sum_{x,y} w(x,y) I_x^2$$

$$B = \sum_{x,y} w(x,y) I_x I_y$$

$$C = \sum_{x,y} w(x,y) I_y^2$$

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

$$= \sum_{x,y} w(x,y) \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix}$$

Step 4: Implement the corner metrics

- `CornerDetector::harris_metric`

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

- `CornerDetector::harmonic_mean_metric`

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

- `CornerDetector::min_eigen_metric`

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

Step 5: Dilate image to find local maxima

```
CornerDetector::detect
```

- Dilate the image to make each pixel equal to the maximum in the neighborhood

```
cv::dilate(response, local_max, …);
```

Step 6: Compute the metric threshold

`CornerDetector::detect`

- Find the maximum response
  - `cv::minMaxLoc(…)`

- Compute the threshold
  - `max_val * quality_level_`

Step 7: Extract local maxima above threshold

```
CornerDetector::detect
```

- Find local maxima and threshold maximum response

```
cv::Mat is_strong_and_local_max;
// = response > threshold and response == local_max
```

- Extract each detected point

```
cv::findNonZero(is_strong_and_local_max, max_locations);
```

`CornerDetector` is finished!

- Try different metrics and parameters
- What is detected?
- Evaluate:
  - Repeatability
  - Distinctiveness
  - Efficiency
  - Locality
- How is the distribution of the points?
  - ANMS, Szliski

CircleEstimator

Step 8: Finish the RANSAC loop

- Go to `CircleEstimator::ransacEstimator`

- Remove the break and perform the correct test

```
…
Eigen::Index tst_num_inliers = is_inlier.count();

    // Check if this estimate gave a better result.
    // Todo: Step 8: Remove break and perform the correct test!
    break; // Remove!
    if (false)
    {
      // Update circle with largest inlier set.
      best_circle = tst_circle;
      best_num_inliers = tst_num_inliers;
      best_is_inlier = is_inlier;

      // Update max iterations.
      double inlier_ratio = static_cast<double>(best_num_inliers) / static_cast<double>(pts.cols());
      max_iterations = static_cast<int>(std::log(1.0 - p_) /
                                 std::log(1.0 - inlier_ratio*inlier_ratio*inlier_ratio));
    }
```

Lab 3 is finished!

- Find the circle on the chessboard
- Play around with the RANSAC parameters
  - `CircleEstimator(double p = 0.99, float distance_threshold = 5.0f)`
- Read through the estimator
  and try to relate the code to the example in the lecture