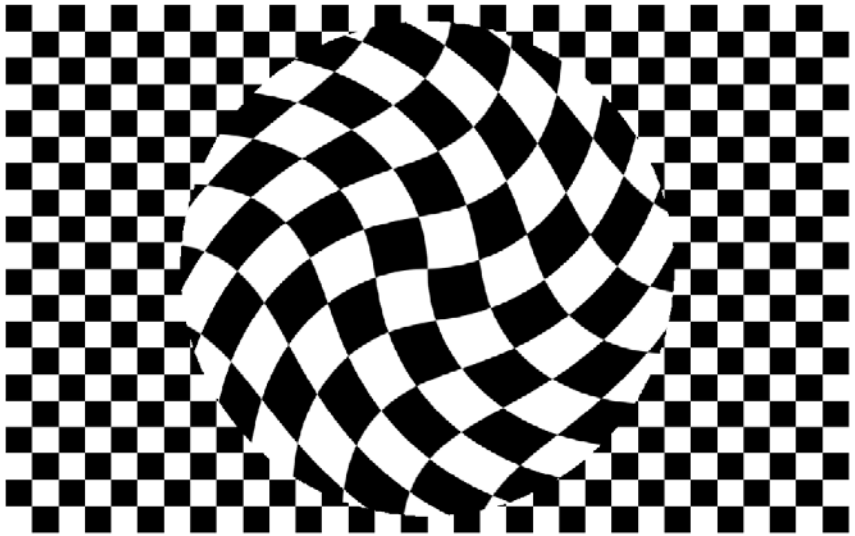
Lab 3 – Circles from corner detection

09.02.2017

Dagens

- Lage hjørnedetektor
- Bruke hjørner og RANSAC til å finne sirkel!



CornerDetector

Hovedelementer

- Beregn bildegradientene med deriverte gaussfiltre
- Beregn bildene A, B, C
  - Ytterproduktet av gradientene
  - Anvend vindu ved å konvolvere med større gaussfilter
- Bruk A, B, og C til å beregne en hjørnemetrikk for hele bildet
  - λ_{min}, Harris, Harmonic Mean, andre?
- Terskle metrikkbildet og finn lokale maksima
  - Morfologiske operasjoner

Sett opp prosjekt

- Last ned halvferdig prosjekt fra semestersiden
- Legg til Release-konfigurasjon i CLion

Steg 1: Filterkjerner

- Ta en titt på denne funksjonen i filters.cpp

```
cv::Mat gaussian_kernel(double sigma, int radius = 0)
```

  - Returnerer 1D gaussfilter

- Lag ferdig

```
cv::Mat derivated_gaussian_kernel(double sigma, int radius = 0)
```

  - Bruk gaussfilteret over.

$$G_x(x) = -\frac{x}{\sigma^2} G(x)$$

Steg 2: Beregn gradientene

`CornerDetector::detect`

- Bruk 1D gaussfiltrene til å beregne gradientbildene I_x og I_y
  - `cv::sepFilter2D(image, Ix, CV_32F, ?, ?);`
  - `cv::sepFilter2D(image, Iy, CV_32F, ?, ?);`

Steg 3: Beregn M

`CornerDetector::detect`

- Beregn A, B og C fra gradientbildene I_x og I_y
- Konvolver A, B, og C med vindusfilteret
  - win_kernel_

Steg 4: Implementer hjørnemetrikkene

- `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`

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

Steg 5: Finn terskel og terskle metrikkbildet

`CornerDetector::detect`

- Finn maksresponsen
  - `cv::minMaxLoc(…)`
- Beregn terskelen
  - `max_val * quality_level_`
- Terskle metrikkbildet
  - `cv::threshold(…)`
  - Alle verdier under terskelen skal settes til 0
  - `cv::THRESH_TOZERO`

Steg 6: Finn lokal maksima

`CornerDetector::detect`

```
cv::Mat local_max;
cv::dilate(response, local_max, cv::Mat{});

cv::Size img_size = image.size();
std::vector<cv::KeyPoint> key_points;
float keypoint_size = static_cast<float>(3.0 * window_sigma_);
for (int y = 1; y < img_size.height - 1; ++y)
{
  for (int x = 1; x < img_size.width - 1; ++x)
  {
    float val = response.at<float>(y, x);
    float local_max_val = local_max.at<float>(y, x);

    if (val != 0 && val == local_max_val)
    {
      cv::Point2f point{static_cast<float>(x), static_cast<float>(y)};
      key_points.push_back(cv::KeyPoint{point, keypoint_size, -1, val});
    }
  }
}
```

Ferdig!

- Prøv forskjellige metrikker og innstillinger
- Hva blir detektert?
- Hva kan vi bruke dette til?
- Hvordan fordeler punktene seg?
  - ANMS, Szliski

CircleEstimator

Steg 7: Legg til i prosjekt

- Endre CMakeLists.txt
  - Legg til følgende filer i add_executable(…)
    - circle_estimator.h
    - circle_estimator.cpp
    - circle.h

Steg 8: Legg til i kode

- Inkluder Circle_estimator.h i main.cpp
- Konstruer en `CircleEstimator estimator`
- Kall `estimator.detectAndVisualizeCircle(…)`

Steg 9: Gjør ferdig RANSAC-løkka

`CircleEstimator::ransacEstimator(…)`

- Hvis `tst_circle` har flere inneliggere enn `best_circle`
  - Oppdater `best_circle`
  - Oppdater `best_num_inliers`
  - Estimer ny `max_iterations`

Ferdig!

- Finn sirkelen i sjakkbrettet!

- Se hva som skjer når du skrur litt på RANSAC-parameterene
  - `CircleEstimator(double p, double distance_threshold)`