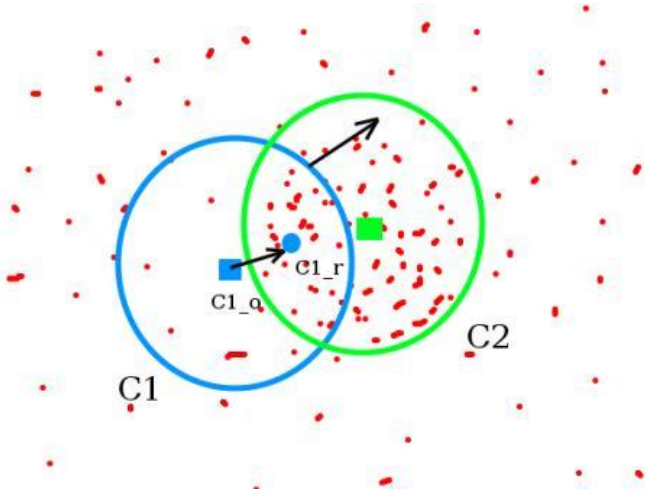
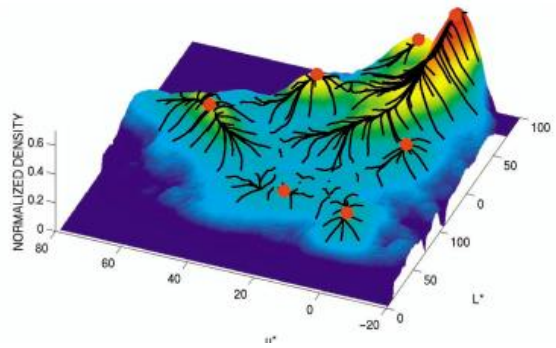
PROCESSAMENTO DE IMAGENS APLICADO A AGROINDUSTRIA

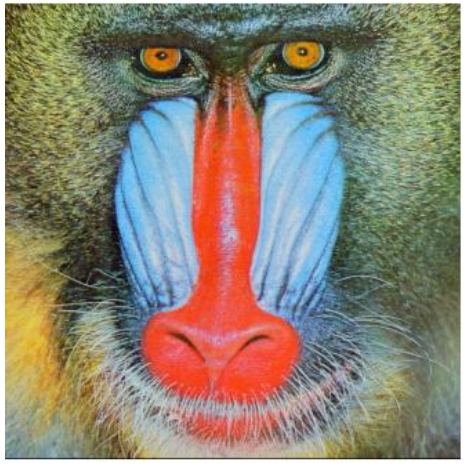
Pedro Luiz de Paula Filho

Mean Shift

- Esta técnica permite encontrar elementos em vídeo e imagens e segui-los.
- A ideia por trás da técnica é:
  - Considerando um série de pontos, dá-se uma janela, que se move até encontrar a densidade máxima de pixels (centroides)







δc = 50 r = 12

δc = 70 r = 12

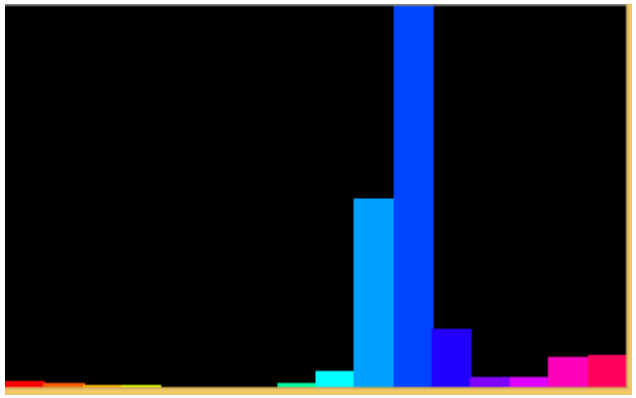
δc = 16 r = 6

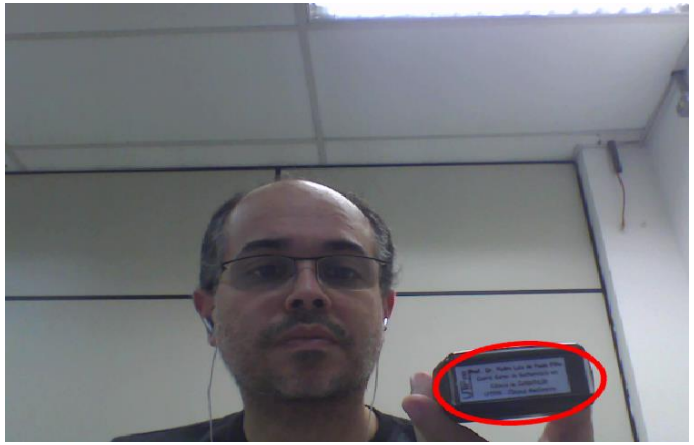
(δc) → desvio padrão de cor - (r) → raio da janela

Mean Shift

- Encontra um objeto em uma imagem
- Sintaxe: int meanShift(InputArray probImage, Rect& window, TermCriteria criteria)
  - probImage → Histograma do objeto
  - Window → janela inicial de busca
  - Criteria → critério de parada do algoritmo
  - Retorno → numero de interações do CAMSHIFT

Mean Shift x CamShift





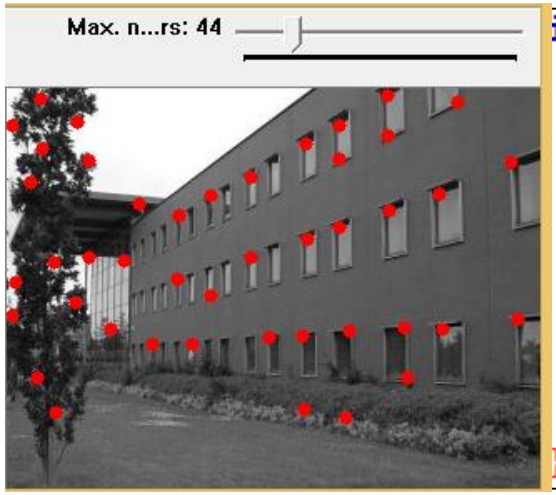
C:\opencv\opencv\sources\samples\cpp\camshiftdemo.cpp

GoodFeaturesToTrack

- Determina cantos “fortes” em uma imagem
- void goodFeaturesToTrack(InputArray image, OutputArray corners, int maxCorners, double qualityLevel, double minDistance, InputArray mask=noArray(), int blockSize=3, bool useHarrisDetector=false, double k=0.04)
  - Image → 8 bits
  - Corners → vector dos cantos
  - maxCorner → retorna o “n” cantos mais fortes
  - qualityLevel → qualidade do canto
  - minDistance → distância mínima entre os cantos
  - Mask → ROI (opcional)

```
Mat image, image_gray;
int max_corners = 20;
```

```
void on_slider(int, void *) {
    if(image_gray.empty()) return;
    max_corners = max(1, max_corners);
    setTrackbarPos("Max no. of corners", "Corners", max_corners);
    float quality = 0.01;
    int min_distance = 10;
    vector<Point2d> corners;
    goodFeaturesToTrack(image_gray, corners, max_corners,
                        quality, min_distance);
    // Draw the corners as little circles
    Mat image_corners = image.clone();
    for(int i = 0; i < corners.size(); i++) {
        circle(image_corners, corners[i], 4, CV_RGB(255, 0, 0), -1);
    }
    imshow("Corners", image_corners);
}
```



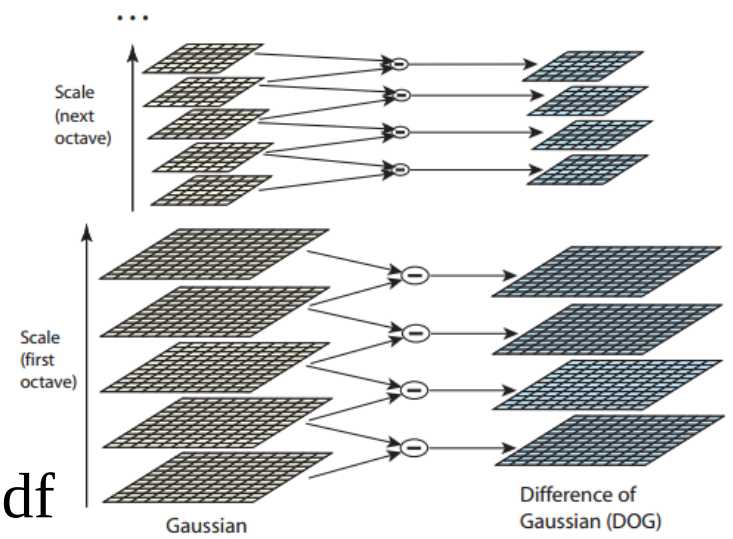

```
int main() {
    image = imread("d:/predio.jpg");
    cvtColor(image, image_gray, CV_RGB2GRAY);
    namedWindow("Corners");
    on_slider(0, 0);
    createTrackbar("Max. no. of corners", "Corners",
                   &max_corners, 250, on_slider);
    while(char(waitKey(1)) != 27) {}
    return 0;
}
```

Descritores de pontos chave

- Neste tipo de descritor, não se busca uma “região” da imagem e sim pontos chaves (keypoints)
- Através destes pontos chaves busca saber quantos deles são similares entre imagens diferentes
- Os keypoints podem ser simples cantos, ou usando técnicas mais elaboradas para serem conseguidos

SIFT

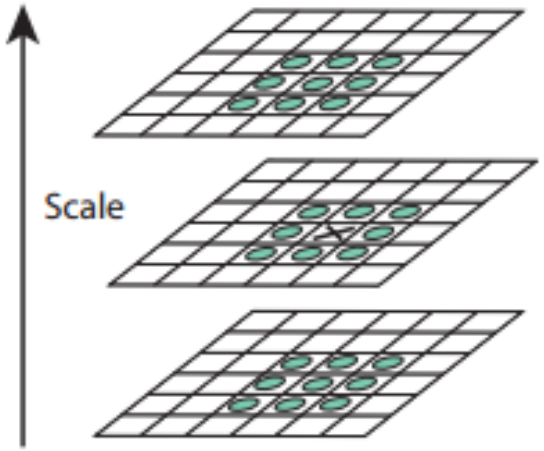
- Usado para detecção de objetos.
- Técnica invariante a escala e rotação.
- Para achar os keypoints usa uma pirâmide gaussiana e as sucessivas imagens da pirâmide são subtraídas uma das outras, gerando uma imagem (DoG - *Difference of Gaussians*)



http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

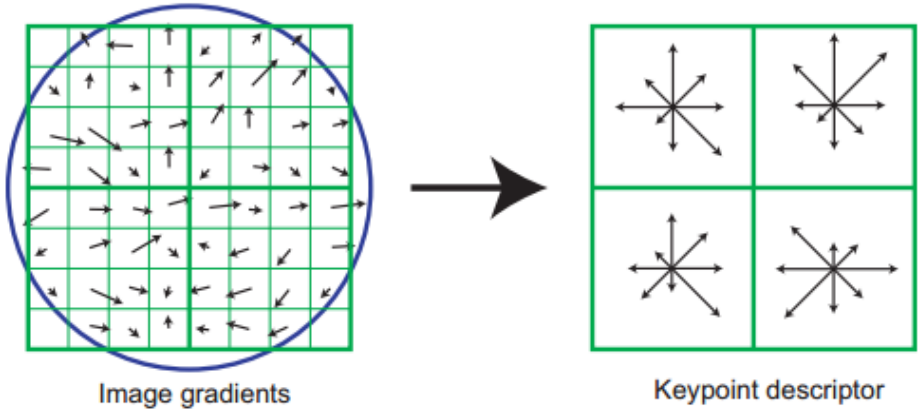
SIFT

- A partir do DoG (*Difference of Gaussians*) são extraídos os mínimos e máximos comparando os 26 vizinhos da pirâmide
- Na sequência calcula-se a orientação e a magnitude destes pontos e calcula-se um histograma com os 36 valores (10 graus) possíveis



SIFT

- Uma região é dividida em 16 blocos e para cada um deles é feito um histograma de 8 posições possíveis referente a orientação, gerando um total de 128 elementos associado a cada keypoint
- Dois pontos são considerados equivalentes se a distancia euclidiana entre os 128 elementos de cada ponto for pequena



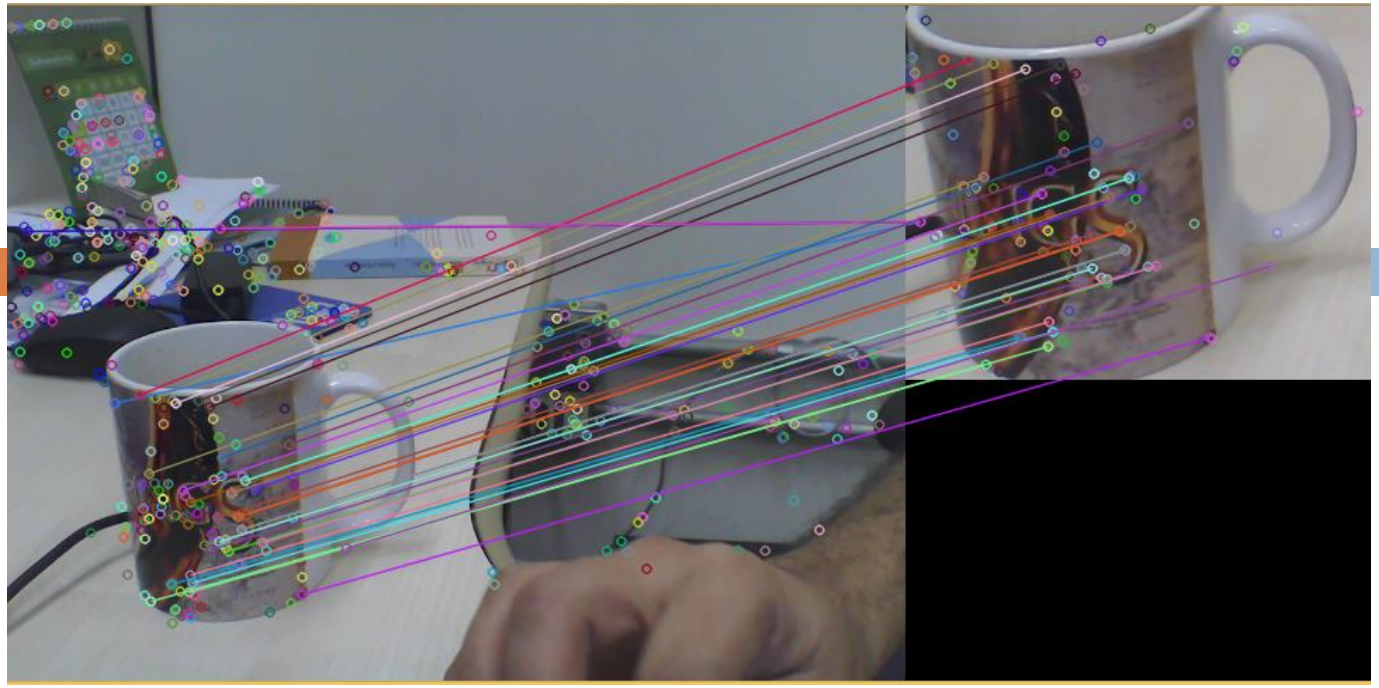
FeatureDetector - OpenCv

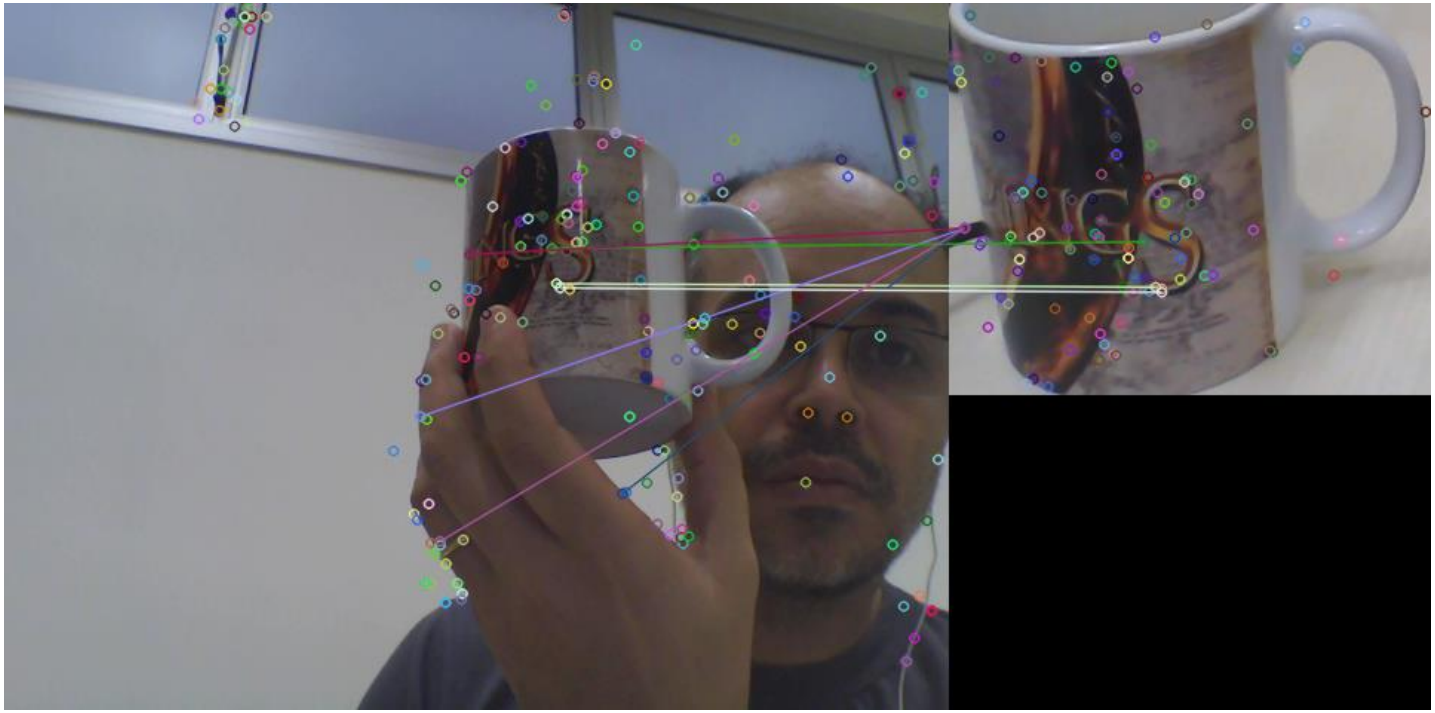
- FeatureDetector é uma classe base para detecção de keypoints em várias técnicas (SIFT, SURF, ORB, ...)
- O método detect() retorna um vector de KeyPoint
- SiftFeatureDetector (herda de FeatureDetector) extrai keypoints de uma imagem em tons de cinza.

```
Mat train = imread("d:/caneca.png"), train_g;
cvtColor(train, train_g, CV_BGR2GRAY);
//detect SIFT keypoints and extract descriptors in the t
vector<KeyPoint> train_kp;
Mat train_desc;
SiftFeatureDetector featureDetector;
featureDetector.detect(train_g, train_kp);
SiftDescriptorExtractor featureExtractor;
featureExtractor.compute(train_g, train_kp, train_desc);

// Brute Force based descriptor matcher object
   BFMatcher matcher;
FLANN based descriptor matcher object
FlannBasedMatcher matcher;
vector<Mat> train_desc_collection(1, train_desc);
matcher.add(train_desc_collection);
matcher.train();
```

```
VideoCapture cap(0);
unsigned int frame_count = 0;
while(char(waitKey(1)) != 'q') {
    double t0 = getTickCount();
    Mat test, test_g;
    cap >> test;
    cvtColor(test, test_g, CV_BGR2GRAY);
    //detect SIFT keypoints and extract descriptors in the
    vector<KeyPoint> test_kp;
    Mat test_desc;
    featureDetector.detect(test_g, test_kp);
    featureExtractor.compute(test_g, test_kp, test_desc);
    // match train and test descriptors, getting 2 nearest
    vector<vector<DMatch> > matches;
    matcher.knnMatch(test_desc, matches, 2);
    vector<DMatch> good_matches;
    for(int i = 0; i < matches.size(); i++) {
        if(matches[i][0].distance < 0.6 * matches[i][1].distance)
            good_matches.push_back(matches[i][0]);
    }
    Mat img_show;
    drawMatches(test, test_kp, train, train_kp, good_matches, img_show);
    imshow("Matches", img_show);
    cout << "Frame rate = " << getTickFrequency() / (getTickCount() - t0)
}
return 0;
}
```





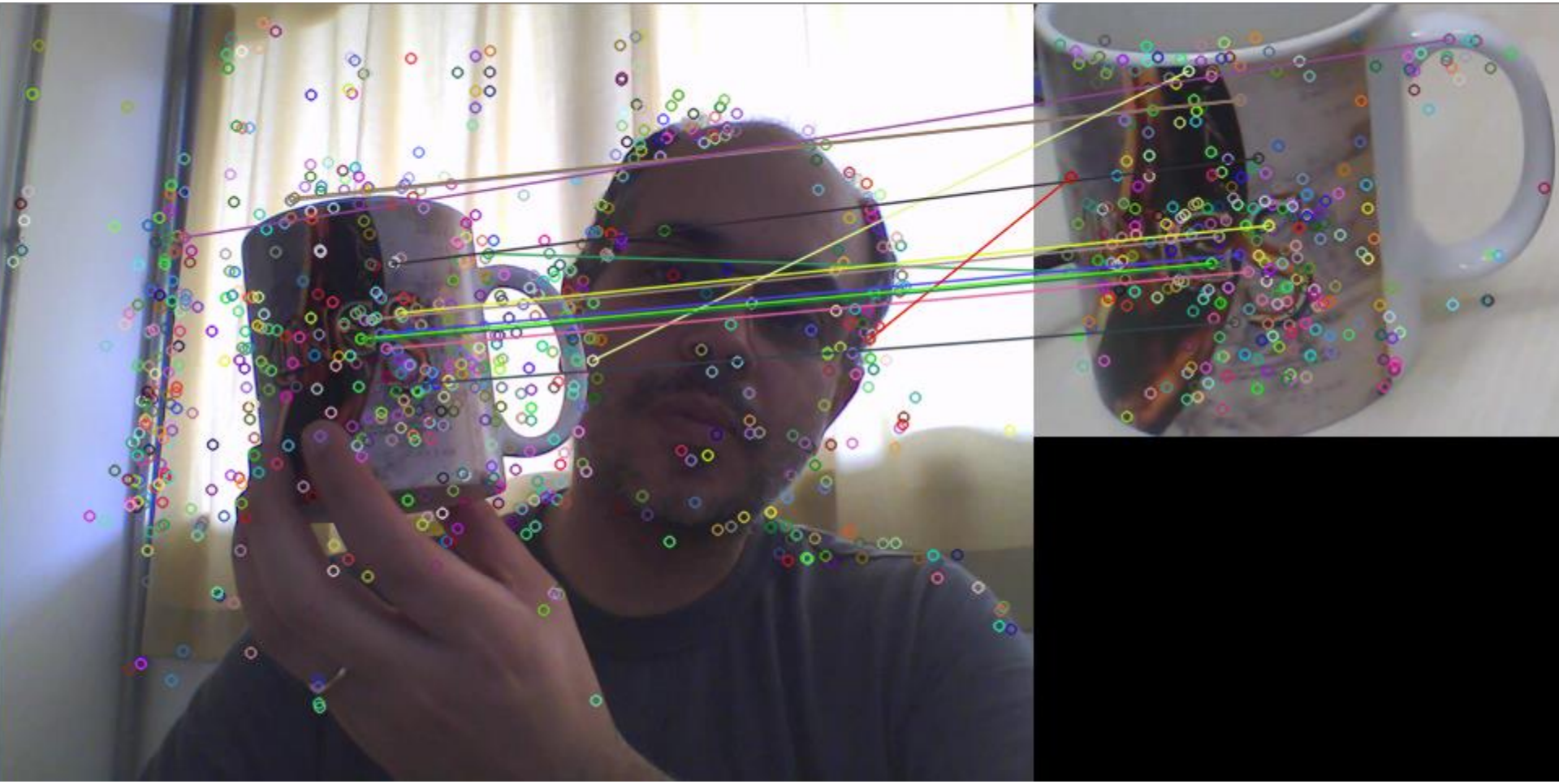
SURF


```
Mat train = imread("d:/caneca.png"), train_g;
cvtColor(train, train_g, CV_BGR2GRAY);
letect SIFT keypoints and extract descriptors in the trair
vector<KeyPoint> train_kp;
Mat train_desc;

SurfFeatureDetector featureDetector(100);
featureDetector.detect(train_g, train_kp);
SurfDescriptorExtractor featureExtractor;
featureExtractor.compute(train_g, train_kp, train_desc);

// FLANN based descriptor matcher object
FlannBasedMatcher matcher;
vector<Mat> train_desc_collection(1, train_desc);
matcher.add(train_desc_collection);
matcher.train();
```

http://www.vision.ee.ethz.ch/~surf/eccv06.pdf



ORB -Oriented FAST and Rotated BRIEF


```
Mat train = imread("d:/caneca.png"), train_g;
cvtColor(train, train_g, CV_BGR2GRAY);

//detect SIFT keypoints and extract descriptors in the train image
vector<KeyPoint> train_kp;
Mat train_desc;

OrbFeatureDetector featureDetector;
featureDetector.detect(train_g, train_kp);
OrbDescriptorExtractor featureExtractor;
featureExtractor.compute(train_g, train_kp, train_desc);

cout << "Descriptor depth " << train_desc.depth() << endl;
// FLANN based descriptor matcher object
flann::Index flannIndex(train_desc, flann::LshIndexParams(12, 20, 2),
                        cvflann::FLANN_DIST_HAMMING);
```

https://www.willowgarage.com/sites/default/files/orb_final.pdf