EXTRACCIÓN CARACTERÍSTICAS

Módulo features2D (ej7.cpp)

- Puntos Característicos: SIFT/SURF/ORB
- Descriptores: SIFT/SURF/ORB

https://docs.opencv.org/4.5.5/d5/d51/group__features2d__main.html
https://docs.opencv.org/4.5.5/d3/df6/namespacecv_1_1xfeatures2d.html

https://docs.opencv.org/4.5.5/d9/d97/tutorial_table_of_content_features2d.html

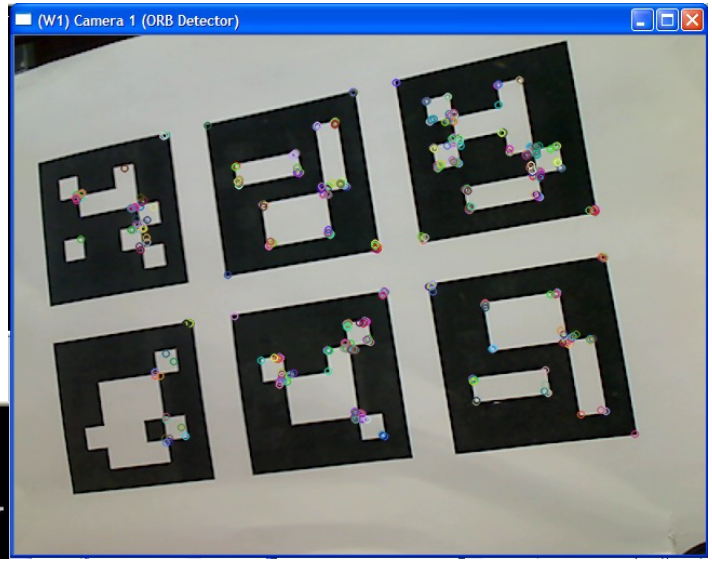
Extracción Características: módulo features2D

- Ejemplo (**ej7.cpp**): extracción de características
- Detectores:
  - Vector de puntos: vector<cv::**KeyPoint**> keypoints;
    - cv::**KeyPoint**::**KeyPoint(**cv::Point2f **_pt**, float **_size**, float **_angle**=-1, float **_response**=0, int **_octave**=0, int **_class_id**=-1)
    - cv::**Point2f**(float **x**, float **y**)
  - **SIFT**, **ORB**, **AKAZE**, **KAZE**, **FAST, MSER**, **GFTT** [, **HARRIS**], **AGAST**, **BRISK**, **SimpleBlob**
  - **SURF**, **STAR, MSD, TBMR, HARRISLP** **(modulo xfeatures2D – opencv-contrib)**

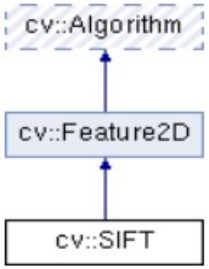
    SURF precisa compilación con *flag* OPENCV_ENABLE_NONFREE

- Descriptores: cv::**Mat** descriptors;
  - Vectores de características (por filas):
  - **SIFT**, **ORB**, **AKAZE(MLDB), KAZE(MSURF), BRISK** **(modulo xfeatures2D)**
  - **SURF**, **FREAK**, **VGG, BRIEF, DAISY LUCID, LATCH, BOOST, BEBLID**

```
Capturing images.
 Hit q/Q to exit.
Detected: 500 points
Descriptor Dimension: 32
[168, 114, 58, 109, 96, 14, 115, 120, 121, 10, 118, 88, 95, 27, 11, 48, 180, 17,
 70, 106, 224, 180, 49, 15, 231, 217, 32, 1, 66, 128, 106, 163]_
```



Extracción Características: módulo features2D



- Ejemplo (**ej7.cpp**): extracción de características
- Clase: **SIFT**: cv::**SIFT**
  - cv::SIFT::**create**(int **nfeatures**=0, int nOctaveLayers=3, double contrastThreshold=0.04, double edgeThreshold=10, double sigma=1.6) → detector (cv::**Ptr**<cv::**SIFT**>)

- Clase base: cv::**Feature2D**

  Detectar Puntos:
  - cv::Feature2D::**detect**(InputArray **image**, vector< KeyPoint > &**keypoints**, InputArray **mask**=noArray())

    mask: 8-bit binary matrix with non-zero values in the region of interest

  Calcular Descriptores:
  - cv::Feature2D::**compute**(InputArray **image**, std::vector< KeyPoint > &**keypoints**, OutputArray **descriptors**)

    Si no existe el descriptor para un punto se elimina el keypoint.
    También se pueden añadir nuevos (múltiples orientaciones dominantes)

    descriptors: cv::Mat(np,ds)

  - cv::Feature2D::**detectAndCompute**(InputArray **image**, InputArray **mask**, vector< KeyPoint > &**keypoints**, OutputArray **descriptors**, bool **useProvidedKeypoints** = false)
  - cv::Feature2D::**descriptorSize**() → retval (int)
  - cv::Feature2D::**descriptorType**() → retval (int)

Extracción Características: módulo features2D

- Ejemplo (**ej7.cpp**): **SIFT** partiremos del código **ej1.cpp**

```
string FEATURE_TYPE = "sift";                // Default Feature Detector sift / surf / orb
```

```
vector<cv::KeyPoint> keypoints;   // Vector for storing detected points
cv::Mat descriptors;              // Matrix for storing descriptors, each row is a keypoint descriptor

cv::Ptr<cv::FeatureDetector> detector;// Feature Detector handler

//Init FeaturePoint Detector
if (FEATURE_TYPE == "sift")
        detector = cv::SIFT::create(100);          // nfeatures

// Detector information
cout << "Feature Type: "<< FEATURE_TYPE << endl;
cout << "Descriptor size: " << detector->descriptorSize() << endl;
```

```
# processing section (image capture loop)
cv::cvtColor(capture, gray_image, cv::COLOR_BGR2GRAY);

detector->detect(gray_image, keypoints);       // Detect featured points

detector->compute(gray_image, keypoints, descriptors);   // compute featured  descriptors
```

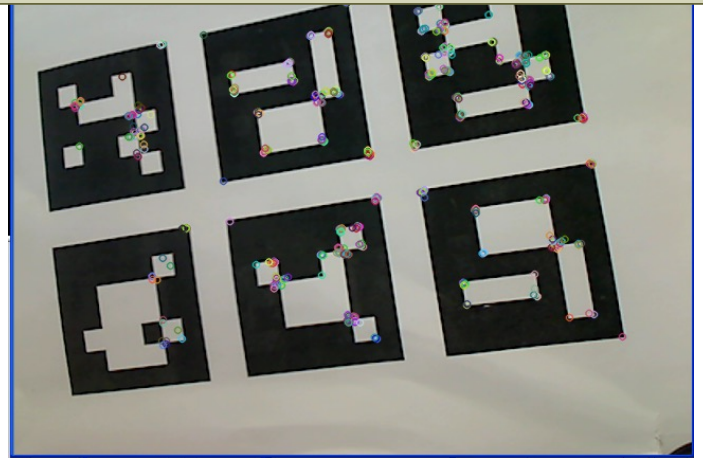
Extracción Características: módulo features2D

- Ejemplo (**ej7.cpp**): Dibujar puntos

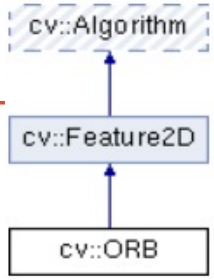
cv::**drawKeypoints**(InputArray **image**, vector<cv::KeyPoint>& **keypoints**, InputOutputArray **outImage**, cv::Scalar &**color**=cv::Scalar::all(-1), cv::DrawMatchesFlags **flags**)

- **outImage: empty** (DEFAULT flags copia image), Imagen de fondo (DRAW_OVER_OUTIMG)
- **color**: (cv::Scalar::all(-1). random)
- **flags**: cv::DrawMatchesFlags::DEFAULT, (crea *outimage* con *image* como fondo)
  cv::DrawMatchesFlags::DRAW_OVER_OUTIMG,
  cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS, (muestra escala y orientación)
  cv::DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS

```
# Draw keyPoints
cv::Mat dispimage;
cv::drawKeypoints( capture, keypoints, dispimage, cv::Scalar::all(-1),
                                    cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS );

cv::imshow(WINDOW_CAMERA1, dispimage);     # Display the resulting frame
```



Extracción Características: módulo features2D

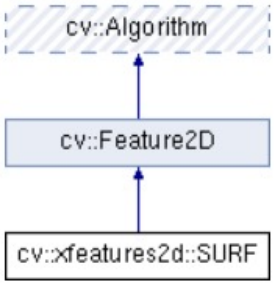


- Ejemplo (**ej7.cpp**): extracción de características
- Clase: **ORB**: cv::**ORB**
  - cv::ORB::**create**(int **nfeatures**=500, …) → detector (cv::**Ptr**<cv::**ORB**>)
- Métodos:
  - cv::ORB::**setMaxFeatures**(int maxFeatures)

```
# Initialization section
else if (FEATURE_TYPE == "orb")
        detector = cv::ORB::create(100);              // nfeatures
```

```
# processing section (image capture loop)
cv::cvtColor(capture, gray_image, cv::COLOR_BGR2GRAY);

detector->detect(gray_image, keypoints);        // Detect featured points

detector->compute(gray_image, keypoints, descriptors);   // compute featured  descriptors
```

Extracción Características: módulo features2D



- Ejemplo (**ej7.cpp**): extracción de características
- Clase: **SURF**: cv::xfeatures2d::**SURF**

  (modulo xfeatures2D – opencv-contrib) (patentado)

  SURF: Precisa compilacion con **OPENCV_ENABLE_NONFREE flag**

  - cv.xfeatures2d.**SURF_create**(double **hessianThreshold**=100, ….)
    → detector (cv::**Ptr**<cv::xfeatures2d::**SURF**>)
- Métodos:
  - cv::xfeatures2d::SURF::**setExtended**(bool extended) Tamaño descriptor → true: 128, **false**: 64
  - cv::xfeatures2d::SURF::**setHessianThreshold**(double hessianThreshold)

```
#include <opencv2/xfeatures2d.hpp>        // opencv-contrib xfeatures2d (SURF)
```

```
# Initialization section
else if (FEATURE_TYPE == "surf") // needs contrib package with OPENCV_ENABLE_NONFREE flag
        detector = cv::xfeatures2d::SURF::create(400);          // hessianThreshold
```

```
# processing section (image capture loop)
cv::cvtColor(capture, gray_image, cv::COLOR_BGR2GRAY);

detector->detect(gray_image, keypoints);        // Detect featured points

detector->compute(gray_image, keypoints, descriptors);   // compute featured  descriptors
```

Extracción Características: módulo features2D

- Resumen Detectores y Descriptores

- Detectores:
  - cv::**SIFT**::**create**()
  - cv::**ORB**::**create**()
  - cv::**AKAZE**::**create**()
  - cv::**KAZE**::**create**()
  - cv::**BRISK**::**create**()
  - cv::**FastFeatureDetector**::**create**()
  - cv::**MSER**::**create**()
  - cv::**GFTTDetector**::**create**()
  - cv::**AgastFeatureDetector**::**create**()
  - cv::**SimpleBlobDetector**::**create**()

- Paquete opencv-contrib:
  - cv::xfeatures2d::**SURF**::**create**()
  - cv::xfeatures2d::**StarDetector**::**create**()
  - cv::xfeatures2d::**MSDDetector**::**create**()
  - cv::xfeatures2d::**TBMR**::**create**()
  - cv::xfeatures2d::**HarrisLaplaceFeatureDetector**::**create**()

- Descriptores:
  - cv::**SIFT**::**create**()
  - cv::**ORB**::**create**()
  - cv::**AKAZE**::**create**()
  - cv::**KAZE**::**create**()
  - cv::**BRISK**::**create**()

- Paquete opencv-contrib:
  - cv::xfeatures2d::**SURF**::**create**()
  - cv::xfeatures2d::**BEBLID**::**create**()
  - cv::xfeatures2d::**BoostDesc**::**create**()
  - cv::xfeatures2d::**BriefDescriptorExtractor**::**create**()
  - cv::xfeatures2d::**DAISY**::**create**()
  - cv::xfeatures2d::**FREAK**::**create**()
  - cv::xfeatures2d::**LATCH**::**create**()
  - cv::xfeatures2d::**LUCID**::**create**()
  - cv::xfeatures2d::**VGG**::**create**()

CORRESPONDENCIA CARACTERÍSTICAS

Módulo features2D (ej7b.cpp)

DescriptorMatcher

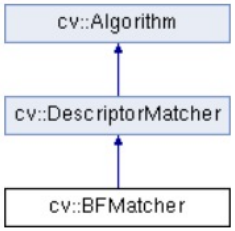
https://docs.opencv.org/4.5.5/d8/d9b/group__features2d__match.html

https://docs.opencv.org/4.5.5/d5/d6f/tutorial_feature_flann_matcher.html

Correspondencia Características: módulo features2D

- Ejemplo (**ej7b.cpp**): extracción de características /matching
- Matchers: cv::**BFMatcher** (Brute Force) / cv::**FlannBasedMatcher**

  - cv::BFMatcher::**create(** int **normType**=cv::NORM_L2, bool **crossCheck**=false)
    → cv::**Ptr**<cv::**BFMatcher**> object

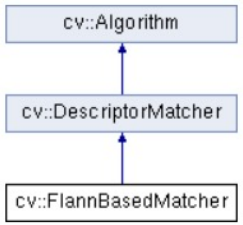


normType: parámetro medida de distancia
- cv::**NORM_L1**, cv::**NORM_L2** -> SIFT, SURF
- cv::**NORM_HAMMING**, cv::**NORM_HAMMING2** -> ORB, BRIEF, BRISK

crossCheck : (bool) correspondencia valida si coinciden en los dos sentidos

  - cv::FlannBasedMatcher::**create**( ) → cv::**Ptr**<cv::**FlannBasedMatcher**> object

  - **Precisa que el descriptor sea de tipo CV_32F (ORB es entero)**



Para Descriptores binaros es preciso configurar los parámetros de la clase con el constructor por defecto:
cv::FlannBasedMatcher::FlannBasedMatcher(cv::Ptr<cv::flan::indexParams> **indexParams**,
cv::Ptr<cv::flan::SearchParams> **searchParams**)

  - cv::FlannBasedMatcher(new cv::flann::LshIndexParams(20, 10, 2)) → flannMatcherObject

  - cv::**makePtr**<cv::FlannBasedMatcher>(flannMatcherObject) or use **new**

Correspondencia Características: módulo features2D

- Ejemplo (**e7b.py**): extracción de características /matching

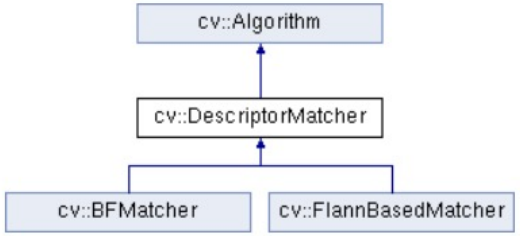
```
string MATCHER_TYPE = "flann";                    // Default Feature Matcher bf / flann

cv::Mat capture, dispimage, image_ref;            // Images
bool new_reference = true;                        // Take new reference image

vector<cv::KeyPoint> keypoints, keypoints_ref;    // Vector for storing detected points
cv::Mat descriptors, descriptors_ref;             // Matrix for storing descriptors,
vector<cv::DMatch> matches;                       // Vector for storing matches

cv::Ptr<cv::Feature2D> detector;
cv::Ptr<cv::DescriptorMatcher> matcher;
```

```
// Init Feature Matcher
if (MATCHER_TYPE == "flann" )
{
        if (FEATURE_TYPE == "orb")   // Binary descriptor->NORM_HAMMING
                matcher = new cv::FlannBasedMatcher(new cv::flann::LshIndexParams(20, 10, 2));
        else    matcher = cv::FlannBasedMatcher::create();
}
else    // Brute Force matcher
{
        if (FEATURE_TYPE == "orb")   // Binary descriptor->NORM_HAMMING
                matcher = cv::BFMatcher::create(cv::NORM_HAMMING, true);
        else
                matcher = cv::BFMatcher::create(cv::NORM_L2, true);      // Scalar descriptor->NORM_L2
}
```

Correspondencia Características: módulo features2D

- Ejemplo (**ej7b.cpp**): extracción de características /matching
- Matchers: cv::**BFMatcher** (Brute Force) / cv::**FlannBasedMatcher**

- Componentes clase cv::**DMatch**:
  - **queryIdx*:* índice vector de descriptores 1 (cámara)**
  - **trainIdx*:* índice vector de descriptores 2 (referencia)**
  - **distance*:* distancia calculada entre vectores de descripción**



- Métodos clase cv::**DescriptorMatcher**:
  - void cv::DescriptorMatcher::**match**(cv::Mat& **queryDescriptors**, cv::Mat& **trainDescriptors**, vector<cv::DMatch>& **matches**, cv::Mat& **mask**=cv::Mat())

    matches (vector<cv.Dmatch>)

  - cv::DescriptorMatcher.**knnMatch**(cv::Mat& **queryDescriptors**, cv::Mat& **trainDescriptors**, vector<vector<cv::Dmatch>>& **matches,** int **k,** cv::Mat& **mask**, bool **compactResult**)

    matches (vector< vector<cv::Dmatch> >) (vector of k-vector of cv::DMatch)

    busca k vecinos más cercanos.

  - cv::DescriptorMatcher.**radiusMatch**(cv::Mat& **queryDescriptors**, cv::Mat& **trainDescriptors**, vector<vector<cv::Dmatch>>& **matches,** float **r,** cv::Mat& **mask**, bool **compactResult**)

    busca vecinos a distancia **r**. Devuelve (vector of k-vector of cv::DMatch)

Correspondencia Características: módulo features2D

- Ejemplo (**ej7b.cpp**): extracción de características /matching
- Bucle principal:

```
detector->detect(gray_image, keypoints);  // Detect featured points
detector->compute(gray_image, keypoints, descriptors);     // compute featured  descriptors

 if (new_reference)         // Store reference on first frame or Reset
 {
    keypoints_ref = keypoints;
    descriptors_ref = descriptors.clone();            // Clone cv::Mat (avoid reference copy)
    image_ref = capture.clone();                      // Clone cv::Mat (avoid reference copy)
    new_reference = false;
 }

 // Match descriptors.
 matcher->match(descriptors, descriptors_ref, matches);   // vector of DMatch

// Sort matches by distance (std::sort() only works with integers so we need a workaround)
vector<int> idx(matches.size());                  // shorted indexes
std::iota(idx.begin(), idx.end(), 0);      // init as correlative indexes
std::sort(idx.begin(), idx.end(),
            [&matches](int i1, int i2) {return matches[i1].distance < matches[i2].distance; });
```

Correspondencia Características: módulo features2D

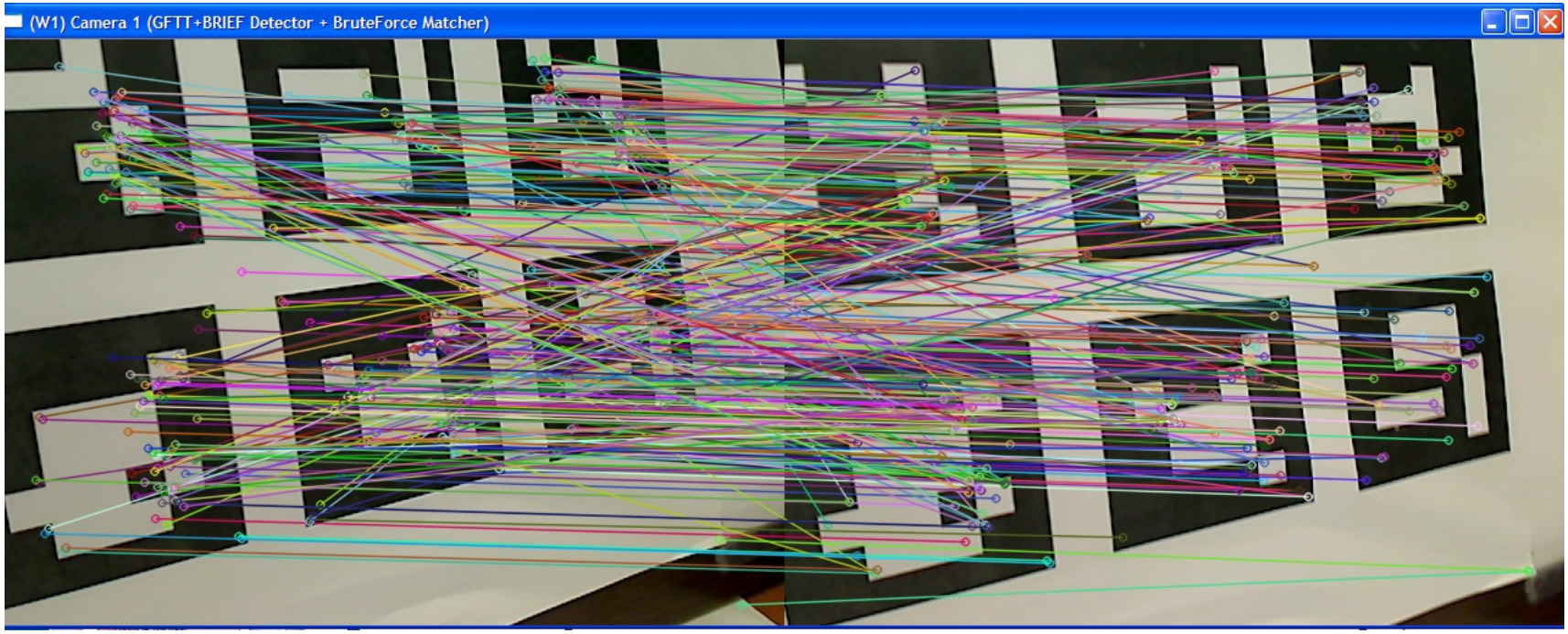
- Ejemplo (**ej7b.cpp**): extracción de características /matching
- Visualización Correspondencias:
- cv::**drawMatches**(img1, **keypoints1**, img2, **keypoints2**, **matches1to2**, outImg, matchColor, singlePointColor, matchesMask, flags)
- cv::**drawMatchesKnn**(img1, **keypoints1**, img2, **keypoints2**, **matches1to2**, outImg, matchColor, singlePointColor, matchesMask, flags)
  - **flags**: cv::DrawMatchesFlags::DEFAULT, (crea *outimage* con *image* como fondo)
    cv::DrawMatchesFlags::DRAW_OVER_OUTIMG,
    cv::DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS

```
// Extracts the 20 best matches (minimun distance)
vector<cv::DMatch> filtered_matches;
for (int i = 0; i < idx.size() && i < 20; i++)
        filtered_matches.push_back(matches[ idx[i] ]);

// Draw best  20 matches.
cv.drawMatches(capture, keypoints, image_ref, keypoints_ref,   filtered_matches,
                        dispimage, cv::Scalar::all(-1), cv::Scalar::all(-1), vector<char>(),
                                        cv::DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

cv.imshow(WINDOW_CAMERA1, dispimage);     # Display matches
…….
// check keystroke to exit (image window must be on focus)
else if (key == 'r' || key == 'R' or key == ' ')
       new_reference = true;     // Reset Reference image
```

Correspondencia Características: módulo features2D

- Ejemplo (**ej7b.cpp**): extracción de características /matching
- Mostrar correspondencias:

```
// shows matches on console
cout << "Left points: " << keypoints.size() << " - Right points: " << keypoints_ref.size() << endl;
cout << "Matched points: " << matches.size() << endl;
for (int i = 0; i < min(10, (int)matches.size()); i++)
{
        cout << "-- Match [" << i << "] Keypoint Cam: " << matches[i].queryIdx;
        cout << " -- Keypoint Ref: " << matches[i].trainIdx << " -- dist: " << matches[i].distance << endl;
}
```



CORRESPONDENCIA CARACTERÍSTICAS

Módulo features2D. (ej7c.cpp)

- Correspondencia entre dos cámaras
- knnMatch() obtener k correspondencias
- Radio test: compara las dos mejores

Correspondencia Características: módulo features2D

- Ejemplo (**e7c.py**): correspondencia entre dos cámaras

```
int CAMERA_ID[2] = {0, 1};    //  camera ids

cv::VideoCapture camera[2];     // Camera handlers
cv::Mat capture[2];        // Mat headers for both cameras
cv::Mat gray_image[2];
cv::Mat dispimage;                // Mat for drawing matches

vector< cv::KeyPoint> keypoints[2];// Vector for storing detected points
cv::Mat descriptors[2];  // Matrix for storing descriptors, each row is a keypoint descriptor
vector< cv::DMatch> matches;              // Vector for storing matches
vector<cv::DMatch> goodMatches;
```

```
// Configure cameras
for(int i=0; i<2; i++)
{
        camera[i].open(CAMERA_ID[i]); //  open  camera
        if (!camera[i].isOpened())
        { cout << "you need to connect the camera (" << i+1 << "), sorry.\n";  getchar();  return -1;  }
        //set capture properties
        camera[i].set(CV_CAP_PROP_FRAME_WIDTH, camSize.width);
        camera[i].set(CV_CAP_PROP_FRAME_HEIGHT, camSize.height);
}
```

```
// free camera resources
for (int i = 0; i < 2; i++)
                if (camera[i].isOpened())        camera[i].release();
```

Correspondencia Características: módulo features2D

- Ejemplo (**e7c.py**): correspondencia de dos cámaras
- Procesamiento para cada cámara:

```
while (true)
{
        // read image from each camera
        for(int i=0; i<2; i++)
            camera[i].read(capture[i]); // read camera frame

        if(capture[0].empty() || capture[1].empty() )
              continue;       // capture has failed, continue

       for(int i=0; i<2; i++)
       {
            cv::cvtColor( capture[i], gray_image[i], cv::COLOR_BGR2GRAY );  // color to gray

           detector->detect(gray_image[i],  keypoints[i]);          // Detects featured points
            detector->compute(gray_image[i],  keypoints[i], descriptors[i]);  // featured  descriptors
       }

        // Find matches between camera and reference
     ................
}
```

Correspondencia Características: módulo features2D

- Ejemplo (**e7c.py**): correspondencia de dos cámaras
- Filtrado de correspondencias:

```
// knnMatch descriptors. vector<vector<cv::DMatch>>
matcher->knnMatch(descriptors[0], descriptors[1], matches, 2)

// Apply ratio test to the two best matches
goodMatches.clear();
for (int i = 0; i < matches.size(); i++)
{
    cv::DMatch m1, m2;
    m1 = matches[i][0]; m2 = matches[i][1];    //  vector with two elements k == 2
    if (m1.distance / m2.distance < 0.75)
        goodMatches.push_back(m1);
}

// Sort matches by distance (std::sort() only works with integers so we need a workaround)
vector<int> idx(goodMatches.size());            // shorted indexes
std::iota(idx.begin(), idx.end(), 0);        // init as correlative indexes
std::sort(idx.begin(), idx.end(),
        [&goodMatches](int i1, int i2) {return goodMatches[i1].distance < goodMatches[i2].distance; });
```

Correspondencia Características: módulo features2D

- Ejemplo (**e7c.py**): correspondencia de dos cámaras
- Visualización correspondencias:

```
// Draw first good matches.
cv::drawMatches(captures[0], keypoints[0], captures[1], keypoints[1], goodMatches,  dispImage,
                cv::Scalar::all(-1), cv::Scalar::all(-1), vector<char>(),
                cv::DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS)
```