Semana 2&3: Visión Estéreo

Práctica 1: Calibración Estéreo

Crea un programa llamado *stereo_calibrate* que lea desde fichero las imágenes stereo de calibración en el siguiente enlace y obtenga los paramétros de calibración estéreo:

Para ello use las funciones:

```
cv::findChessboardCorners, cv::cornerSubPix y cv::stereoCalibrate
```

Como salida el debe crear un fichero .yml .

```
Uso:
./stereo_calibrate dir_with_images out.yml
```

Puede ayudarse de la clase DirReader que lee los arhivos de un directorio:

```
DirReader Dir;
auto files=Dir.read(argv[1],".jpg",DirReader::Params(true));
```

El fichero resultante deberá ser tal y como éste.

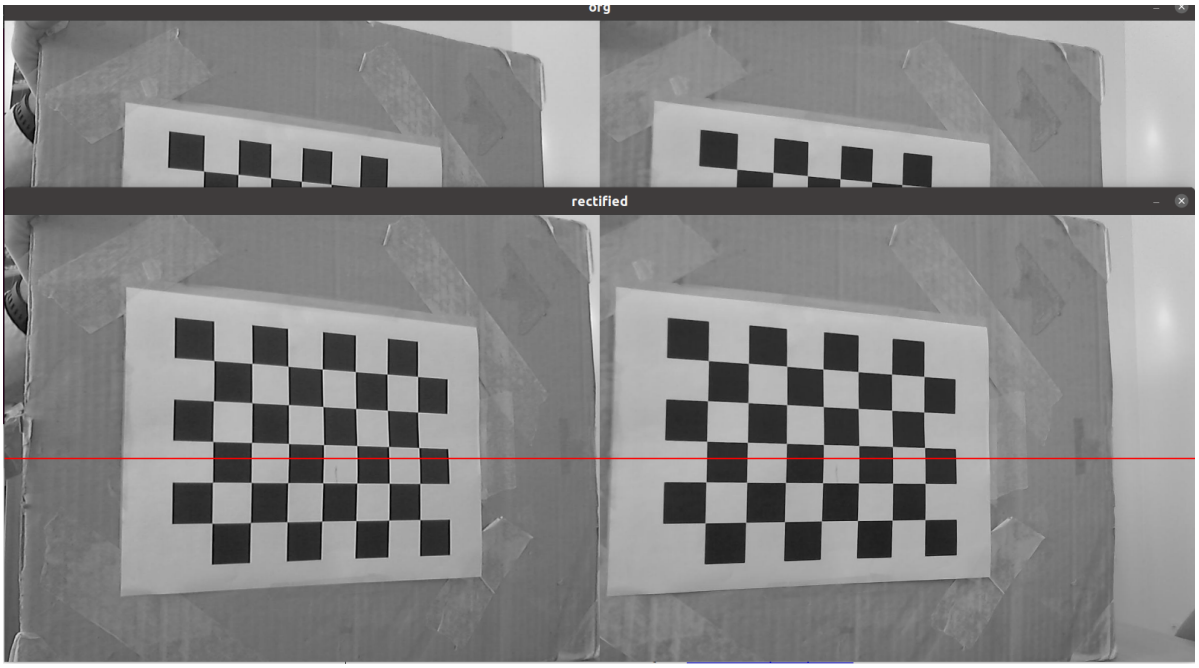
Nota:
cv::Size CheckerBoardSize={7,5};//
double SquareSize=0.02875;//size of each square
Usar el critero de optimización:
cv::TermCriteria(cv::TermCriteria::MAX_ITER + cv::TermCriteria::EPS, 60, 1e-6)

Práctica 2: Comprobación de imágenes rectificadas

Cree el programa *stereo_checkundistorted* que reciba como entrada una imágen estéreo y el correpondiente fichero de calibración estéreo. El programa debe mostrar dos ventanas. En la primera ventana se muestran las imágenes izquierda y derecha orginales de la cárama estereo. Al pasar el ratón por encima, se dibujará de forma dinámica una línea horizontal que cruza ambas imágenes.

Además, deberá mostrar otra ventana similar donde se muestra las imágenes izquierda y derecha despues de la rectificación. Igualmente, el programa deberá pintar una línea horizontal al moverse sobre la ventana.

De esta manera, podremos comprobar visualmente cómo de bien la rectificación ha funcionado, creando dos cámaras paralelas.



Uso:
./stereo_checkundistorted stereo_image.jpg stereocalibrationfile.yml

Para rectificar las imágenes, puede usar el siguiente código:

```
//Structure that contains the Stereo Pair Calbration information.
//This will be calculated using stereo_calibrate
struct StereoParams{
   cv::Mat mtxL,distL,R_L,T_L;
   cv::Mat mtxR,distR,R_R,T_R;
   cv::Mat Rot, Trns, Emat, Fmat;
};

void rectifyStereoImages(const StereoParams &sti,cv::Mat &left,cv::Mat &rigth){
   cv::Mat rect_l, rect_r, proj_mat_l, proj_mat_r, Q;
   cv::Mat Left_Stereo_Map1, Left_Stereo_Map2;
   cv::Mat Right_Stereo_Map1, Right_Stereo_Map2;
   cv::stereoRectify(sti.mtxL,
sti.distL,sti.mtxR,sti.distR,left.size(),sti.Rot,sti.Trns,
                     rect_l,rect_r,proj_mat_l,proj_mat_r,
                     Q,cv::CALIB_ZERO_DISPARITY, 0);
```

```
    cv::initUndistortRectifyMap(sti.mtxL,sti.distL,rect_l,proj_mat_l,
                                left.size(),CV_16SC2,
                                Left_Stereo_Map1,Left_Stereo_Map2);
    cv::initUndistortRectifyMap(sti.mtxR,sti.distR,
                                rect_r,proj_mat_r,
                                left.size(),CV_16SC2,
                                Right_Stereo_Map1,Right_Stereo_Map2);
    cv::Mat AuxImage, Right_nice;
    cv::remap(left, AuxImage, Left_Stereo_Map1, Left_Stereo_Map2,
              cv::INTER_LANCZOS4,cv::BORDER_CONSTANT,0);
    AuxImage.copyTo(left);
    cv::remap(rigth,  AuxImage,  Right_Stereo_Map1,  Right_Stereo_Map2,
              cv::INTER_LANCZOS4,cv::BORDER_CONSTANT,0);
    AuxImage.copyTo(rigth);
}
```

Práctica 3: Mapa Denso de Disparidad Estéreo

Cree el programa *stereo_disparity* que recibe como entrada una imágen estéreo, el fichero de calibración estéreo, y como resultado calcula la disparidad entre las imágenes en cada pixel. Para aquellos puntos con disparidad válida, genera como salida un fichero con formato PCD de la librería PCL que pueda ser visualizado con el programa *pcl_viewer*. Para probar puede utilizar las imágenes en el siguiente enlace.

Uso:
./stereo_disparity stereo_image.jpg calibration.yml out.pcd

Para realizar el proceso, deberá

1) Cargar la imágen estéreo
2) Rectificar las imágenes
3) Utilizar la clase cv::StereoBM para realizar el cáculo de la disparidad.
4) Convierta la disparidad obtenida a valores 32bits flotantes

```
// Converting disparity values to CV_32F from CV_16S
    disp.convertTo(disparity,CV_32F, 1.0);
    disparity=disparity/16.f;
    // Scaling down the disparity values and normalizing them
    cv::Mat disparityNorm = (disparity/16.0f -
(float)minDisparity)/((float)numDisparities);
```

5) Para aquellos puntos con disparidad > 10, triangule usado las ecuaciones básica del par estéreo: Z= B*f /d; X= (x-cx) /Z ; Y= (y-cy) /Z ;
6) Guarde los puntos a formato pcd. Puede utilizar la siguiente función para ello:

```
void writeToPCD(std::string path,std::vector<cv::Point3f> points){
   std::ofstream file(path,std::ios::binary);
```

```
    if(!file)throw std::runtime_error("Could not ope  file:"+path);
    file<<"# .PCD v.7 - Point Cloud Data file format"<<std::endl<<"VERSION
.7"<<std::endl;
    file<<"FIELDS x y z "<<std::endl<<"SIZE 4 4 4 "<<std::endl;
    file<<"TYPE F F F "<<std::endl<<"COUNT 1 1 1 "<<std::endl;
    file<<"WIDTH "<<points.size()<<std::endl<<"HEIGHT 1"<<std::endl;
    file<<"VIEWPOINT 0 0 0 1 0 0 0"<<std::endl;
    file<<"POINTS "<<points.size()<< std::endl;
    file<<"DATA binary"<<std::endl;
    file.write((char*)&points[0],sizeof(cv::Point3f)*points.size() );
}
```

Práctica 4: Reconstrucción 3D dispersa con par estéreo

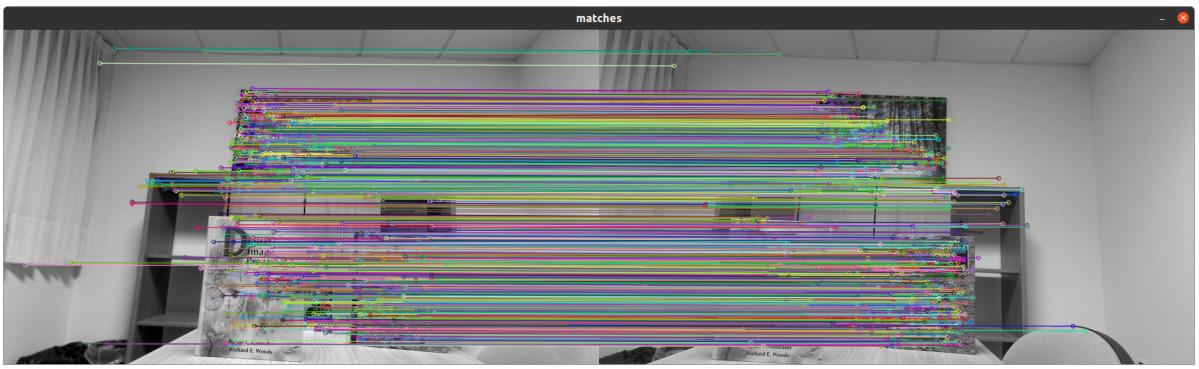
El programa anterior utilizaba block-matching para encontrar emparejamientos entre imágenes. En este ejercicio vamos a buscar los emparejamientos usando keypoints usando el descriptor AKAZE.

El programa debe tener el nombre *stereo_sparse* y debe usarse como.
Uso:
./stereo_sparse *stereo_image.jpg calibration.yml out.pcd*
El programa deberá hacer lo siguiente.

1) Cargar la imágen estéreo
2) Rectificar las imágenes
3) Busca keypoints en ambas imágenes usando AKAZE y el descriptor matcher BruteForce-Hamming

```
auto Detector=cv::AKAZE::create(cv::AKAZE::DESCRIPTOR_MLDB,  0,  3, 1e-4f );
Detector ->detectAndCompute(left, cv::Mat(), keypoints_query, descriptors_query);
Detector ->detectAndCompute(rigth, cv::Mat(), keypoints_train,
descriptors_train);
auto matcher = cv::DescriptorMatcher::create("BruteForce-Hamming");
matcher->match(descriptors_query, descriptors_train, matches, cv::Mat());
```

4) Filtra los matches y deja aquellos que estan en lineas horizontales.
5) Dibuja los matches obtenidos antes y despues del filtrado
6) Triangula los matches y los guarda en el fichero de salida con el formato PCD.


matches