Visión en estéreo.

J.F. Sánchez Castillo, Área Posgrado ITNM.

I. INTRODUCCIÓN.

Este documento muestra las funciones de software utilizadas para realizar la Calibración y Rectificación de dos cámaras en estéreo por medio de la captura de pares de imágenes tomadas en los extremos derecho e izquierdo respectivamente.

Además, se Genera el mapa de profundidad correspondiente a cada par de imágenes a partir de su mapa de disparidad.

Los parámetros generados se basan en el modelo en perspectiva definido por la ecuación:

$$s\ m' = A[R|t]M'$$

Donde:

(X,Y,Z) son las coordenadas de un punto en 3D.
(u,v) son las coordenadas del punto proyectado en pixeles.
A es la matriz de la cámara, o matriz de parámetros intrínsecos.
(cx,cy) es el punto ubicado en el centro de la imagen.
fx,fy son las longitudes focales expresadas en pixeles.

El modelo en perspectiva se muestra a continuación:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

En el caso en que la imagen de la cámara sea escalada, los parámetros son también escalados en la misma relación, la matriz de parámetros intrínsecos no depende de la escena, por eso una vez calculados pueden ser reutilizados siempre y cuando la longitud focal sea fija (Lentes acercamiento). El modelo de perspectiva equivalente para z≠0 es el siguiente:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x/z$$
$$y' = y/z$$
$$u = f_x * x' + c_x$$
$$v = f_y * y' + c_y$$

II. Detección de esquías y rectificación de imágenes

En la etapa

A. Detección de esquinas.

En esta etapa se utiliza la imagen de un tablero de ajedrez y se localizan las esquinas internas, si no se detectan esquinas en la imagen la función regresa un resultado de cero. Un tablero de ajedrez regular tiene una cuadricula de 8x8 y una malla de 7x7 esquinas. La función findChessboardCorners() detecta coordenadas aproximadas para después pasar sus parámetros a la función cornerSubPix() que calcula las coordenadas exactas para después ser dibujadas en la imagen por medio de la función drawChessboardCorners().

B. Matriz de la cámara.

En esta etapa la función initCameraMatrix2D() toma las esquinas detectadas y encuentra una matriz inicial de puntos en 3D correspondientes en 2D, donde (cx,cy) representan un punto en el centro de la imagen, fx,fy son las longitudes focales expresadas en pixeles.

$$\begin{bmatrix} f_x^{(j)} & 0 & c_x^{(j)} \\ 0 & f_y^{(j)} & c_y^{(j)} \\ 0 & 0 & 1 \end{bmatrix}$$

C. Calibración estéreo.

Posteriormente la función stereoCalibrate() inicia el proceso de calibración estéreo que consiste en encontrar la relación de posición y orientación de la segunda cámara respecto a la primera. Dados los parámetros R1,T1 es posible calcular R2,T2 por medio de la fórmula.

$$R_2 = R * R_1 T_2 = R * T_1 + T,$$

Y conociendo la posición y orientación de ambas cámaras se puede calcular R y T.

Luego se calcula la matriz Esencial:

$$E = \begin{bmatrix} 0 & -T_2 & T_1 \\ T_2 & 0 & -T_0 \\ -T_1 & T_0 & 0 \end{bmatrix} * R$$

para: $T = [T_0, T_1, T_2]^T$

La matriz Fundamental se calcula con la ecuación:

$$F = cameraMatrix2^{-T} E cameraMatrix1^{-1}$$

D. Rectificación estéreo.

Esta etapa se calcula la matriz de rectificación para cada cámara que virtualmente hace que ambos planos de imágenes sean el mismo plano, en consecuencia, se hace que las líneas epipolares sean paralelas lo que simplifica el problema de la correspondencia estéreo. El proceso toma los parámetros generados en la etapa de calibración, como resultado genera dos matrices de rectificación y dos matrices de proyección con las nuevas coordenadas. La función utilizada es stereoRectify().

Para estéreo Horizontal ambas cámaras son desplazadas con relación entre ellas, principalmente en el eje x, con un pequeño desplazamiento vertical. En las imágenes rectificadas las líneas epipolares en ambas cámaras son horizontales y tienen las mismas coordenadas.

Las matrices de proyección se definen como:

$$P1 = \begin{bmatrix} f & 0 & cx_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$P2 = \begin{bmatrix} f & 0 & cx_2 & T_x * f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Donde Tx= desplazamiento horizontal entre las cámaras y cx1=cx2.

Para estéreo vertical las matrices de proyección son:

$$P1 = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy_1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$P2 = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy_2 & T_y * f \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Donde Ty= desplazamiento vertical entre las cámaras y cx1=cx2

Enseguida la función initUndistortRectifyMap() calcula el mapa de transformación en el espacio de pixeles, es decir que para cada pixel (u,v) en la imagen destino, se calcula las coordenadas correspondientes con la imagen de referencia. Para cada cámara se calcula la matriz de homografía H que realiza una rectificación en el dominio de pixeles como lo indica la siguiente formula.

$$\mathtt{R} = \mathtt{cameraMatrix}^{-1} \cdot \mathtt{H} \cdot \mathtt{cameraMatrix}$$

Una vez obtenidos los mapas de rectificación para cada imagen se envían como parámetro a la función remap() que se encarga de aplicar una transformación geométrica a cada imagen de acuerdo al mapa correspondiente a cada imagen. La siguiente formula describe este proceso:

$$\mathtt{dst}(x,y) = \mathtt{src}(\mathtt{map}_x(x,y), \mathtt{map}_y(x,y))$$

III. CORRESPONDENCIA EN ESTÉREO

A. Correspondencia en estéreo.

La correspondencia en estéreo se encarga de encontrar un par de puntos en una imagen que pueden ser identificados en otra imagen, si se conoce la geometría del sistema en estéreo el mapa de disparidad se puede convertir en un mapa en 3D de la escena.

Para el cálculo del mapa en 3D se deben considerar la profundidad y la disparidad en el sistema, la profundidad (z) de un punto en una escena es inversamente proporcional a la diferencia de distancia de puntos correspondientes en la imagen y los centros de las cámaras. La siguiente formula indica esta relación:

$$disparity = x - x' = \frac{Bf}{Z}$$

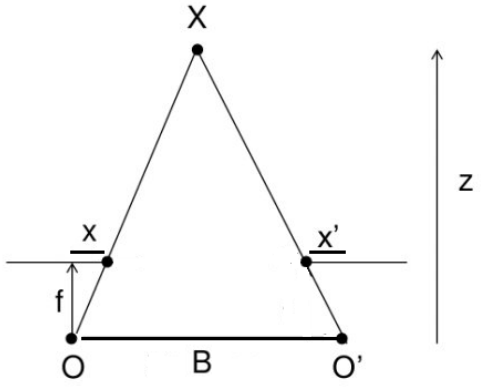

Figura1. Parámetros de un sistema en estéreo. O, O' – Centros de cada cámara, B -Distancia entre centros.

En esta esta etapa del proyecto se utilizan las funciones StereoSGBM_create() para indicar el algoritmo con el que se calculara la correspondencia, y stereo.Compute() para realizar los cálculos, stereo.Compute() recibe como parámetro las imágenes que fueron transformadas geométricamente en la etapa anterior con la función remap() y como resultado entrega el mapa de disparidad.

B. Mapa de profundidad.

El mapa de profundidad se obtiene con la función reprojectImageTo3D(), que se encarga de proyectar el mapa de disparidad de una imagen a un espacio 3D, y para hacer esto utiliza la matriz Q de re proyección obtenida por el proceso de rectificación. El proceso se define con la siguiente ecuación.

$$[X\ Y\ Z\ W]^T = \mathtt{Q} * [\mathtt{x}\ \mathtt{y}\ \mathtt{disparity}(\mathtt{x},\mathtt{y})\ 1]^T$$
$$\mathtt{\_3dImage}(\mathtt{x},\mathtt{y}) = (X/W,\ Y/W,\ Z/W)$$

IV. RESULTADOS.

Se obtuvieron diferentes pares de imágenes rectificadas, donde se muestra las esquinas detectadas en cada imagen patrón, se observa que cada punto detectado se ubica en la misma posición con relación a la segunda imagen. Además se obtuvo el mapa de profundidad correspondiente el cual se puede ver en un archivo con extensión ".ply" en meshlab. El resto de las imágenes resultantes se ubican en la ruta: ./output/.

Figura 2. Par de imágenes rectificadas.

Figura 3. Par de imágenes rectificadas (2).

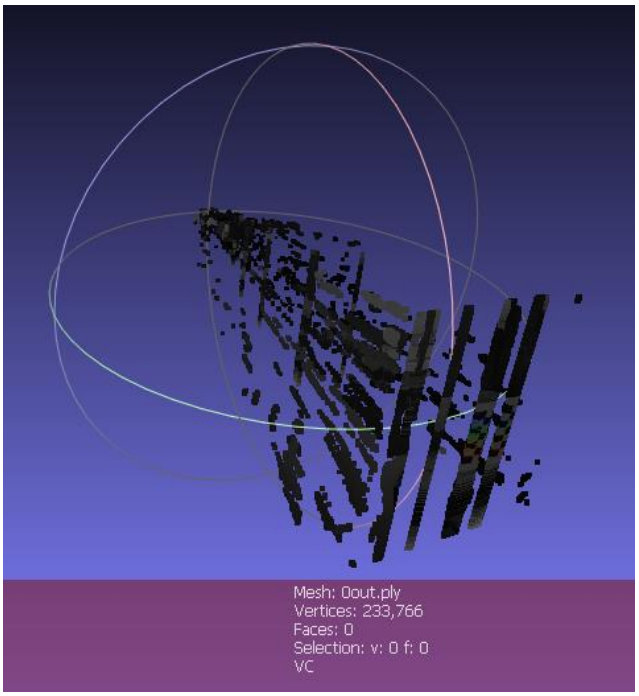

Figura 3. Mapa de profundidad.

V. CONCLUSIONES.

En este proyecto se realizaron las tareas de calibración, rectificación y correspondencias de imágenes en estéreo. Partiendo del modelo de la cámara en perspectiva se obtuvieron las matrices de rotación, traslación de cada cámara (parámetros extrínsecos).

También se obtuvieron las matrices Esencial y Fundamental provenientes del proceso de calibración. De igual manera se obtuvieron los mapas de rectificación que se aplicaron para realizar la transformación geométrica de cada imagen.

Finalmente se utilizaron las matrices de disparidad y re proyección para obtener el Mapa de profundidad.

VI. CÓDIGO FUENTE

```
#!/usr/bin/env python


from __future__ import print_function
import numpy as np
import cv2
import os
import sys
import getopt
from glob import glob
from common import splitfn

################################################################################################
####
################################################################################################
####

ply_header = '''ply
format ascii 1.0
element vertex %(vert_num)d
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
end_header
'''

def write_ply(fn, verts, colors):
    verts = verts.reshape(-1, 3)
    colors = colors.reshape(-1, 3)
    verts = np.hstack([verts, colors])
    with open(fn, 'wb') as f:
        f.write((ply_header % dict(vert_num=len(verts))).encode('utf-8'))
        np.savetxt(f, verts, fmt='%f %f %f %d %d %d ')

######Detección puntos cámara
1#####################################################################
################################################################################################
####

img_mask = './left*.ppm'  # default
img_names = glob(img_mask)

debug_dir = './output/'
if not os.path.isdir(debug_dir):
    os.mkdir(debug_dir)
square_size = float(1.0)

pattern_size = (9, 6)
pattern_points = np.zeros((np.prod(pattern_size), 3), np.float32)
pattern_points[:, :2] = np.indices(pattern_size).T.reshape(-1, 2)
pattern_points *= square_size
```

```
obj_points = []
img_points = []
h, w = 0, 0
img_names_undistort = []
for fn in img_names:
    print('processing %s... ' % fn, end='')
    img = cv2.imread(fn, 0)
    if img is None:
        print("Failed to load", fn)
        continue

    h, w = img.shape[:2]
    found, corners = cv2.findChessboardCorners(img, pattern_size)

    if found:
        term = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_COUNT, 30, 0.1)
        cv2.cornerSubPix(img, corners, (5, 5), (-1, -1), term)
        img=cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
        cv2.drawChessboardCorners(img, pattern_size, corners, found)
        path, name, ext = splitfn(fn)
        outfile = debug_dir + 'points' + name + '.jpg'
        cv2.imwrite(outfile, img)
        if found:
            img_names_undistort.append(outfile)



    img_points.append(corners.reshape(-1, 2))
    obj_points.append(pattern_points)

#####Detección puntos cámara
2#########################################################################
#############################################################################################
####

img_mask2 = './right*.ppm'  # default
img_names2 = glob(img_mask2)
debug_dir2 = './output/'
if not os.path.isdir(debug_dir2):
    os.mkdir(debug_dir2)
square_size2 = float(1.0)

pattern_size2 = (9, 6)
pattern_points2 = np.zeros((np.prod(pattern_size2), 3), np.float32)
pattern_points2[:, :2] = np.indices(pattern_size2).T.reshape(-1, 2)
pattern_points2 *= square_size2

obj_points2 = []
img_points2 = []
h2, w2 = 0, 0
img_names_undistort2 = []
for fn2 in img_names2:
    print('processing %s... ' % fn2, end='')
    img2 = cv2.imread(fn2, 0)
    if img2 is None:
```

```
        print("Failed to load", fn2)
        continue

    h2, w2 = img.shape[:2]
    found2, corners2 = cv2.findChessboardCorners(img2, pattern_size2)
    if found2:
        term2 = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_COUNT, 30, 0.1)
        cv2.cornerSubPix(img2, corners2, (5, 5), (-1, -1), term2)
        img2=cv2.cvtColor(img2, cv2.COLOR_GRAY2BGR)
        cv2.drawChessboardCorners(img2, pattern_size, corners2, found)
        path, name, ext = splitfn(fn2)
        outfile = debug_dir + 'points' + name + '.jpg'
        cv2.imwrite(outfile, img2)
        if found:
            img_names_undistort2.append(outfile)

    img_points2.append(corners2.reshape(-1, 2))
    obj_points2.append(pattern_points2)



########################################################################################################
####
########################################################################################################
####

#Se obtiene la matriz de cada cámara (parámetros intrínsecos)

camera_matrix1=cv2.initCameraMatrix2D(obj_points,img_points,(w,h))
camera_matrix2=cv2.initCameraMatrix2D(obj_points2,img_points2,(w2,h2))
term_crit = (cv2.TERM_CRITERIA_MAX_ITER + cv2.TERM_CRITERIA_EPS, 100, 1e-5)


#Se obtienen Parámetros extrínsecos R y T de cada cámara
flags = 0
#flags |= cv2.CALIB_FIX_INTRINSIC
#flags |= cv2.CALIB_USE_INTRINSIC_GUESS
#flags |= cv2.CALIB_FIX_PRINCIPAL_POINT
#flags |= cv2.CALIB_FIX_FOCAL_LENGTH
flags |= cv2.CALIB_FIX_ASPECT_RATIO
flags |= cv2.CALIB_ZERO_TANGENT_DIST
flags |= cv2.CALIB_SAME_FOCAL_LENGTH
flags |= cv2.CALIB_RATIONAL_MODEL
flags |= cv2.CALIB_FIX_K3
flags |= cv2.CALIB_FIX_K4
flags |= cv2.CALIB_FIX_K5

dist_coeffs1=0
dist_coeffs2=0
(rms,camera_matrix1,dist_coeffs1,camera_matrix2,dist_coeffs2,R,T,E,F)=cv2.stereoCalibrate(obj_points,img
_points,img_points2,camera_matrix1,dist_coeffs1,camera_matrix2,dist_coeffs2,(w,h),criteria=term_crit,flags
=flags)


#Se obtiene Mapas de rectificación R1,R2,P1,P2 y matriz de re proyección Q.
```

```
R1, R2, P1, P2, Q, roi1,
roi2=cv2.stereoRectify(camera_matrix1,dist_coeffs1,camera_matrix2,dist_coeffs2,(w,h),R,T,alpha=0)


map11,map12=cv2.initUndistortRectifyMap(camera_matrix1,dist_coeffs1,R1,P1,(w,h),cv2.CV_16SC2)
map21,map22=cv2.initUndistortRectifyMap(camera_matrix2,dist_coeffs2,R2,P2,(w,h),cv2.CV_16SC2)


##########################Se aplica transformación geométrica a cada imagen###################
##########################Tomando los mapas de rectificación###################################


print('loading images...')


num=0
while num <= (len(obj_points)):

   if num == len(obj_points):
      imgL=cv2.imread('imagen_izq.ppm')
      imgR=cv2.imread('imagen_der.ppm')
   else:
      imgL=cv2.imread(img_names_undistort[num])
      imgR=cv2.imread(img_names_undistort2[num])

   rima1=cv2.remap(imgL,map11,map12,cv2.INTER_LINEAR)
   rima2=cv2.remap(imgR,map21,map22,cv2.INTER_LINEAR)
   concat = np.concatenate((rima1, rima2), axis=1)
   height,width=concat.shape[:2]
   count = 0
   while count < width:

      concat=cv2.line(concat,(0,count),(width,count),(0,255,0),1,8)
      count +=16

   concat2=cv2.resize(concat,(int(width*.7),int(height*.7)))
   cv2.imwrite(debug_dir2 + str(num) +'concat.jpg', concat)

   # Se indica el algoritmo a utilizar para calcular correspondencia estereo
   #SGBM
   window_size = 3
   min_disp = 16
   num_disp = 112-min_disp
   stereo = cv2.StereoSGBM_create(minDisparity = min_disp,
      numDisparities = num_disp,
      blockSize = 16,
      P1 = 8*3*window_size**2,
      P2 = 32*3*window_size**2,
      disp12MaxDiff = 1,
      uniquenessRatio = 10,
      speckleWindowSize = 100,
      speckleRange = 32
   )

   #Se hace el cálculo de correspondencia en estéreo y mapa de
   #disparidad
```

```
    print('computing disparity...')
    disp = stereo.compute(imgL, imgR).astype(np.float32) / 16.0

    print('generating 3d point cloud...',)
    h3, w3 = imgL.shape[:2]
    f = 0.8*w3

    #Se reproyecta mapa disparidad a 3D y se obtiene mapa de
    #profundidad.
    points = cv2.reprojectImageTo3D(disp, Q)
    colors = cv2.cvtColor(imgL, cv2.COLOR_BGR2RGB)
    mask = disp > disp.min()
    out_points = points[mask]
    out_colors = colors[mask]
    out_fn = 'out.ply'

    write_ply(debug_dir + (str(num) + 'out.ply'), out_points, out_colors)
    print('%s saved' % 'out.ply')

    num+=1
    cv2.destroyAllWindows()
#################################################################################################
#################################################################################################
```

VII. BIBLIOGRAFÍA.

[1] OPENCV, Camera Calibration and 3D reconstruction, www.opencv.org, https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html, [online], Ultimo acceso: 6 mayo 2020.

[2] OPENCV, Camera Calibration and 3D reconstruction,www.opencv.org, https://docs.opencv.org/3.4.10/d9/d0c/group__calib3d.html, [online], Ultimo acceso: 6 mayo 2020.

[3] OPENCV, StereoSGBM Class Reference, www.opencv.org, https://docs.opencv.org/3.4/d2/d85/classcv_1_1StereoSGBM.html, [online], Ultimo acceso: 6 mayo 2020.

[4] OPENCV, Depth Map from Stereo Images, www.opencv.org, https://docs.opencv.org/master/dd/d53/tutorial_py_depthmap.html, [online], Ultimo acceso: 6 mayo 2020.

[5] OPENCV, Depth Map from Stereo Images, www.opencv.org, https://docs.opencv.org/master/dd/d53/tutorial_py_depthmap.html, [online], Ultimo acceso: 6 mayo 2020.

[6] OPENCV, cv::stereo::StereoMatcher Class Reference, www.opencv.org, https://docs.opencv.org/3.4/d9/d07/classcv_1_1stereo_1_1StereoMatcher.html, [online], Ultimo acceso: 6 mayo 2020.