Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Proyecto Final

Reconstrucción de Escenas 3D Utilizando Stereo Vision y Deep Learning

Alumno: Ariel Núñez Lobos.
Profesor: Javier Ruiz del Solar.
Auxiliar: Patricio Loncomilla.
Ayudante de proyecto: Jhon Pilataxi.
Fecha de entrega: 14 de Diciembre de 2022
Santiago, Chile

Índice de Contenidos

1. Introducción **1**

2. Marco Teórico **2**
2.1. Stereo Vision . . . 2
2.2. Mesh R-CNN . . . 3

3. Código desarrollado **6**
3.1. Calibración de Camara . . . 6
3.2. Stereo Vision . . . 8
3.3. Deep Learning . . . 11

4. Pruebas y Resultados **13**
4.1. Calibración de la cámara . . . 13
4.2. Primeras pruebas con Stereo visión . . . 14
4.3. Conjunto de entrenamiento . . . 16
4.3.1. SGBM . . . 18
4.3.2. BM . . . 20
4.4. Conjunto de prueba . . . 22
4.4.1. SGBM . . . 23
4.4.2. BM . . . 24
4.4.3. Deep Learning . . . 26

5. Análisis **28**

6. Conclusión **29**

Índice de Figuras

1. Ejemplo de geometría epipolar, donde la profundidad de l trazo visto en dos imágenes puede ser estimado a partir de las distancias entre estas . . . 2
2. Ejemplo de translación y rotación en las imágenes, lo cual conforma la matriz esencial . 3
3. Ejemplo del funcionamiento del algoritmo Mesh R-CNN, donde se identifican objetos para su posterior reconstrucción en 3D, rescatado de [1] . . . 4
4. Proceso de detección y creación de Mesh por algoritmo Mesh R-CNN, rescatado de [1] . 5
5. Detección de esquinas al calibrar dentro del conjunto de calibración entregado . . . 13
6. Proyección de cubos en imágenes sin distorsión, para comprobar calibración . . . 14
7. Par de imágenes para prueba inicial . . . 15
8. mapa de disparidad y reconstrucción 3D obtenida . . . 15
9. Sistema de fotografías para el proyecto, mostrando la posición de la cámara respecto al objeto . . . 16
10. Objeto de entrenamiento: ball . . . 17
11. Objeto de entrenamiento: bottle . . . 17
12. Objeto de entrenamiento: cup . . . 17

13. Objeto de entrenamiento: dumbell 17
14. Objeto de entrenamiento: teacup 18
15. Objeto de entrenamiento: trash 18
16. Objeto de entrenamiento: wood 18
17. Mapas resultantes del entrenamiento, en orden alfabetico y cronológico (inicio de entrenamiento a final de izquierda a derecha, arriba a abajo) 20
18. Mapas resultantes del entrenamiento, en orden alfabetico y cronológico (inicio de entrenamiento a final de izquierda a derecha, arriba a abajo) 21
19. Objeto de prueba: cuptrash 22
20. Objeto de prueba: sneaker 22
21. Objeto de prueba: tallgeese 22
22. Objeto de prueba: tallgeeseball 23
23. Mapas resultantes de prueba, en orden alfabetico a partir de SGBM 23
24. Mesh 3D resultantes para el conjunto de prueba a partir de SGBM 24
25. Mapas resultantes de prueba, en orden alfabetico a partir de BM 25
26. Mesh 3D resultantes para el conjunto de prueba a partir de BM 25
27. Mascaras obtenidas para el conjunto de prueba 26
28. Mascaras para objetos conocidos por el algoritmo, sillas 26
29. Mesh resultantes a partir de los objetos detectados por Deep Learning, en orden: copa, silla y detecciones erradas 27

Índice de Tablas

1. Parámetros utilizados en entrenamiento para SGBM 19
2. Parámetros utilizados en entrenamiento para BM 21

1. Introducción

El presente documento tiene como objetivo presentar el proyecto final de semestre para el curso de procesamiento avanzado de imágenes, correspondiente a la reconstrucción de escenas 3D a partir de dos métodos principales: Stereo Vision y Deep Learning. A partir de esto se podrán contrastar las ventajas y desventajas de cada método, al igual que llegar a ejemplos concretos de la aplicación de estas herramientas.

Stereo Vision, proveniente de la intuición dada por la visión humana y cuyo funcionamiento es dado gracias a cálculos de geometría epipolar, crea diferencias de profundidad en una imagen a partir de las diferencias de perspectivas encontradas en dos capturas tomadas con cierta distancia entre ellas, similar nuevamente a la captura recibida por dos ojos. Esto permite identificar que objetos se encuentran frente a otros, lo cual permite generar mapas de disparidad y asimismo generar una reconstrucción en 3D mediante este mapa.

Deep Learning, por otro lado, se basa en el uso de redes neuronales para su aprendizaje y reconstrucción, siendo el método particular usado en este experiencia el dado por Mesh R-CNN, el cual hace uso del algoritmo de segmentación semántica y detección de objetos Detectron2 para identificar la forma de variados objetos dentro de una sola imagen, para después reconstruirlos en una imagen 3D. Nótese que como algoritmo Deep Learning, este requiere un entrenamiento previo para la obtención de una detección correcta, y sus detecciones dependerán de los datos con los cuales se entrene la red. Por otro lado la reconstrucción mediante visión Stereo solo necesita de dos imágenes, más su reconstrucción no resulta tan detallada y depende de afinar parámetros particulares.

2. Marco Teórico

2.1. Stereo Vision

La reconstrucción 3D inicial utilizada en esta experiencia se basa en la generación de mapas de disparidad, donde se identifica la diferencia de profundidad entre distintos sectores de una imagen, conformado a partir de las diferencias geometricas entre dos imágenes tomadas a una distancia particular [4]. El uso de dos imágenes para generar esta reconstrucción se considera Stereo Vision, o visión estéreo, implicando justamente una visión mediante dos canales distintos, lo cual basta paran la reconstrucción de una imagen mediante geometría epipolar.

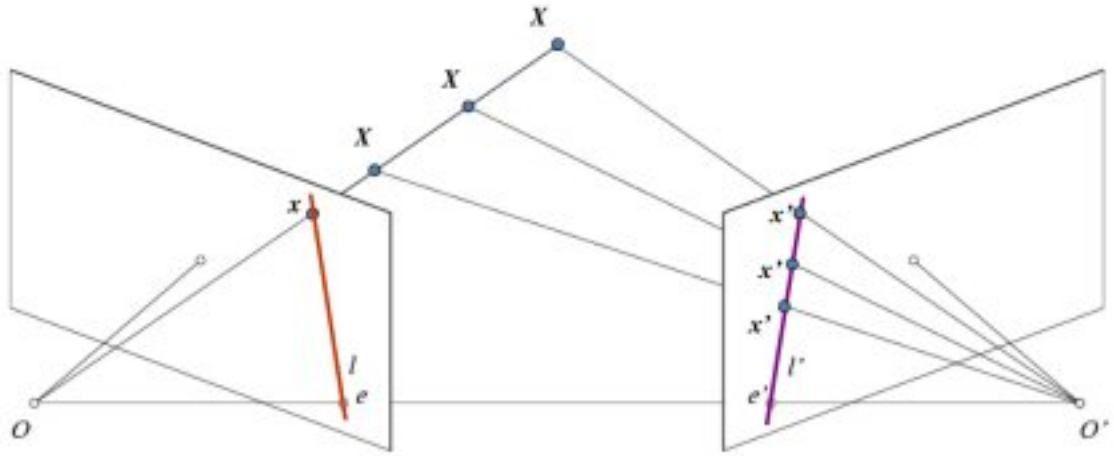

Figura 1: Ejemplo de geometría epipolar, donde la profundidad de l trazo visto en dos imágenes puede ser estimado a partir de las distancias entre estas

Como es visto en la figura anterior, la geometría epipolar permite la identificación de la profundidad de los puntos de interés en una imagen, mas esta depende de la distancia entre las dos imágenes y la posición particular del objeto. Esto presenta una dificultad ya que se deben realizar cálculos a partir de las imágenes particulares para definir translación y rotación de la imagen. Esto puede ser generado a partir de la matriz esencial de estas.

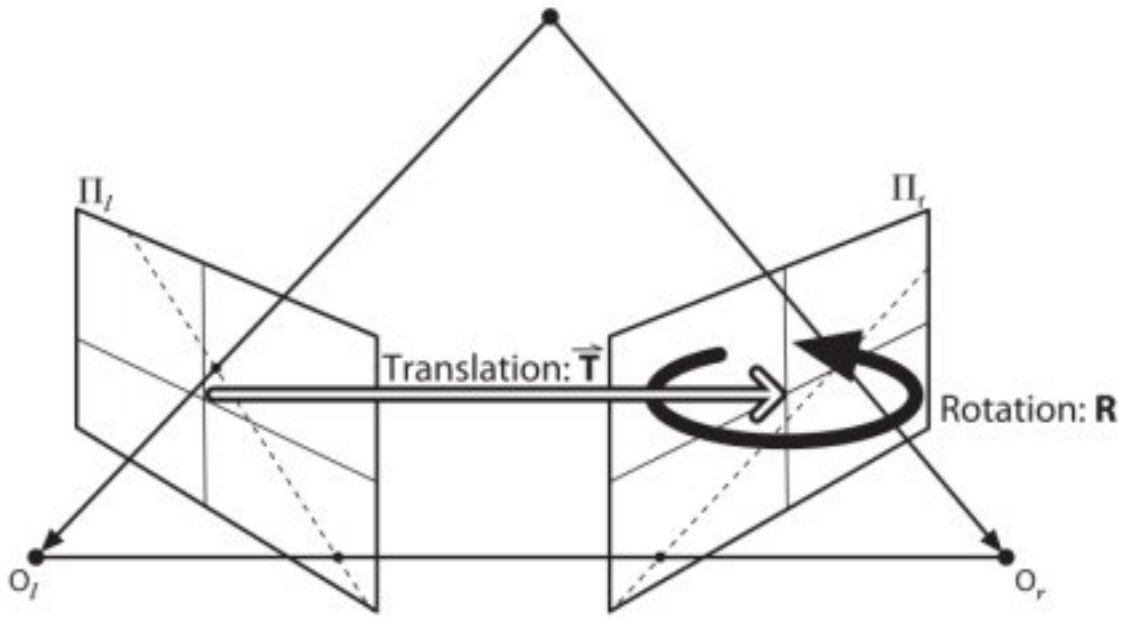

Figura 2: Ejemplo de translación y rotación en las imágenes, lo cual conforma la matriz esencial

Calcular la matriz esencial es conocido, y esta depende de un parámetro conocido como focal length, o largo focal, el cual corresponde a una característica intrínseca de cada cámara usada, y muestra que tan fuerte, o débilmente, el foco de la cámara converge la luz. A partir del calculo de este se puede calcular la matriz que genera la reconstrucción.

Para el caso de visión estéreo, es fundamental por tanto la calibración de la cámara usada para no generar distorsión dado el foco, lo cual provoca curvas innecesarias las fotos, lo cual es generado por algoritmos de la librería cv2, y posteriormente el calculo del focal length de la camara para hacer uso de la matriz esencial.

2.2. Mesh R-CNN

Mesh R-CNN [1] corresponde a un algoritmo de Deep Learning generado para la creación de estructuras 3D a partir de la identificación de objetos dentro de una imagen mediante el algoritmo Detectron2, el cual genera segmentación semántica de una imagen en distintos objetos. A partir de la identificación en 2D dentro de una imagen, el algoritmo esta entrenado para generar una reconstrucción en tres dimensiones.

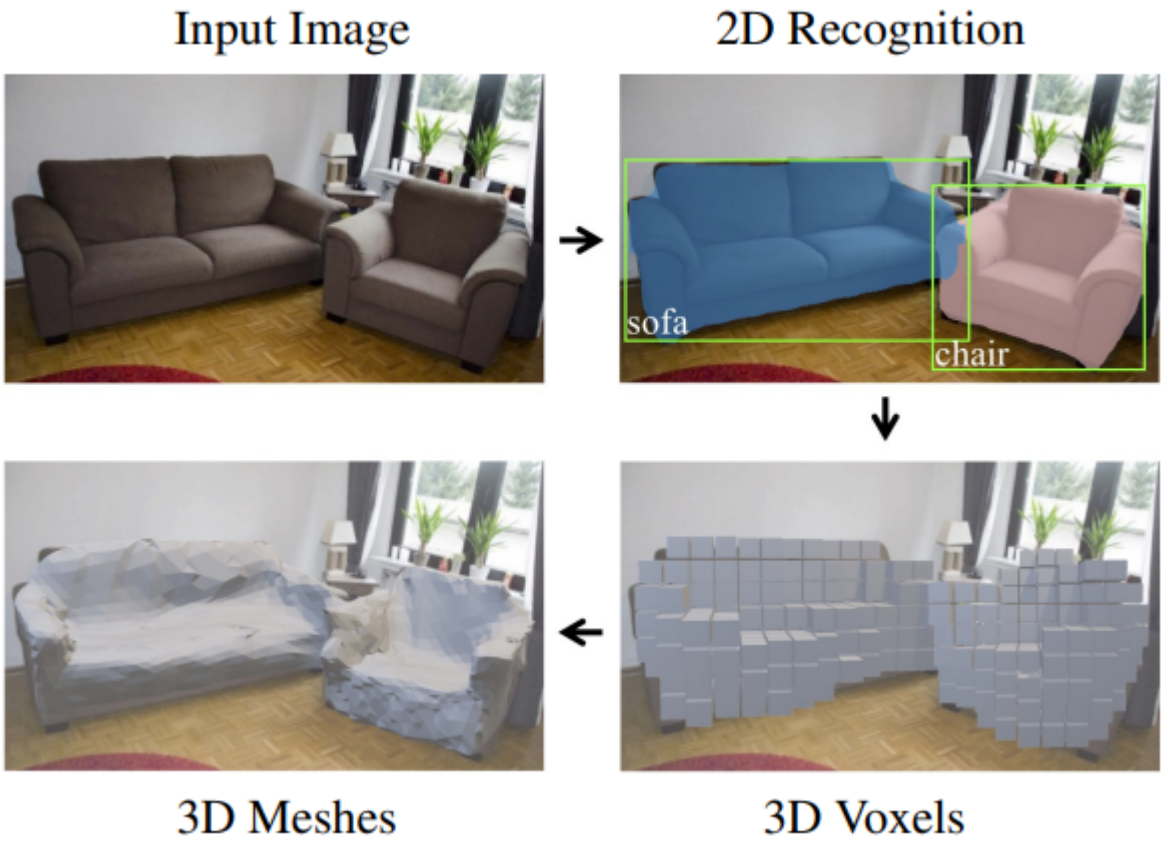

Figura 3: Ejemplo del funcionamiento del algoritmo Mesh R-CNN, donde se identifican objetos para su posterior reconstrucción en 3D, rescatado de [1]

La forma en que se generan los meshes parte en la identificación de un objeto gracias a un algoritmo de segmentación semántico, para posteriormente encerrar la figura encontrada dentro de cubos. Dentro de estos cubos se genera una convolución con la detección 2D para borrar los bordes de esto y afinar de forma más detallada el objeto. Este proceso se repite nuevamente para generar una figura más fina y detallada, y finalmente llegar al objeto deseado. El proceso se describe visualmente a continuación:

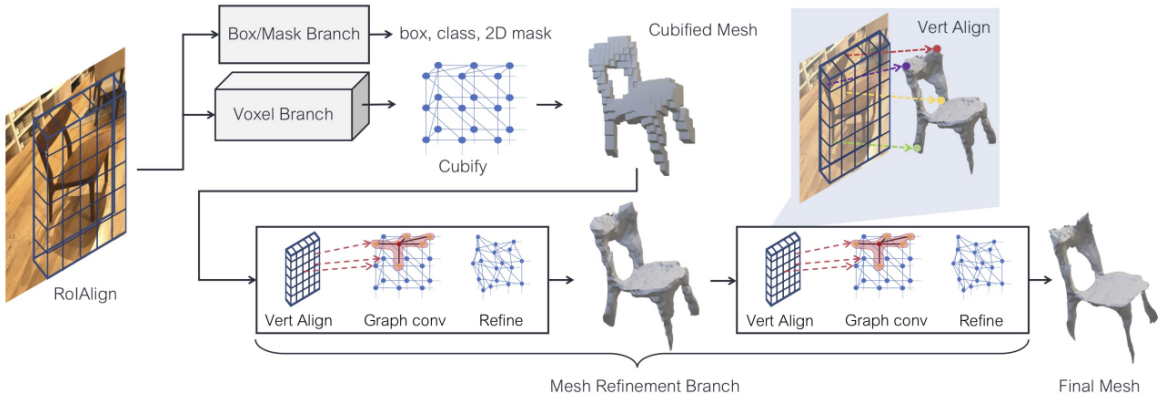

Figura 4: Proceso de detección y creación de Mesh por algoritmo Mesh R-CNN, rescatado de [1]

3. Código desarrollado

El código anexo al trabajo y descrito a continuación corresponde a la implementación de tanto Stereo Vision como Mesh R-CNN dentro de Google Colab, gracias a su capacidad de ejecutarse remotamente independiente de las capacidades computacionales del usuario a ejecutar el código. Sus revisiones correspondientes pueden encontrarse en el repositorio del trabajo.

3.1. Calibración de Camara

La implementación de Stereo Vision se basó principalmente en el código original de calibración de cámaras otorgado por la documentación de cv2 [3], junto al articulo entregado en material docente [4]. El código fue modificado para la implementación local haciendo uso de un patrón de dimensiones (9,6) y una cámara Logitech C270.

El código inicia preparando los parámetros particulares para el patrón a leer, su tamaño y la cantidad de cuadrados en este. Estos se indican y se preparan las listas y arreglos que contendrán la información obtenida de la calibración.

```
square_size = 2.5 #tamaño del cuadrado estimado
img_mask = "./ImgCalibration/*.jpg" #directorio de imagenes de calibración
pattern_size = (9, 6) #cantidad de cuadrados en patrón a analizar
figsize = (20, 20) #tamaño para imagenes

nombre_img = glob.glob(img_mask)
numero_img = len(nombre_img)

pattern_points = np.zeros((np.prod(pattern_size), 3), np.float32) #preparamos para guardar indices
pattern_points[:, :2] = np.indices(pattern_size).T.reshape(-1, 2)
pattern_points *= square_size

obj_points = [] #preparamos para guardar puntos
img_points = []
h, w = cv2.imread(nombre_img[0]).shape[:2]
```

Ya al prepararse, se genera la calibración al iterar dentro de cada imagen en la carpeta que las contiene, cargándolas, convirtiéndolas a escala de grises y cargándolas en la función cv2.findChessboardCorners. Con esto los resultados se agregan a la lista de puntos por imagen y objeto y se presentan los resultados.

```
plt.figure(figsize=figsize)

for i, fn in enumerate(nombre_img):
    print("processing %s... " % fn)

    #transforma imagen cargada a rgb y gray
    imgBGR = cv2.imread(fn)
```

```
    imgRGB = cv2.cvtColor(imgBGR, cv2.COLOR_BGR2RGB)
    imggray = cv2.cvtColor(imgRGB, cv2.COLOR_RGB2GRAY)

    assert w == imggray.shape[1] and h == imggray.shape[0], f"size: {imggray.shape[1]} x {imggray.shape[0]}" #imagenes deben tener el mismo tamaño
    found, corners = cv2.findChessboardCorners(imggray, pattern_size) #usamos funcion para encontrar esquinas, booleano identifica si estas se encontraron o no

    if not found: #Si no encuentra el patrón, imprimir para ver cuantas calibraciones fueron correctas
        print("patrón no encontrado")
        continue

    if i < 12:
        img_w_corners = cv2.drawChessboardCorners(imgRGB, pattern_size, corners, found)
        plt.subplot(4, 3, i + 1)
        plt.imshow(img_w_corners)

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


plt.show() #mostramos 12 patrones encontrados
```

Después de realizarse la calibración y guardarse los parámetros de esta, se obtiene los valores de matriz de calibración y con estos el parámetro focal length, a partir de los parámetros particulares del sensor de la cámara (sensor de tamaño 3.58mm x 2.02mm e imágenes de tamaño 1280x720). Con esto se pueden probar las imágenes sin distorsión y finalmente hacer una prueba de los resultados mediante proyección 3D de cubos a partir de los puntos.

```
#obtenemos focal length

S = cv2.calibrationMatrixValues(camera_matrix, (720,1280), 3.58, 2.02) #valores del sensor de la camara
focal_length = S[2]

# quitar distorsión a partir de la calibración
plt.figure(figsize=figsize)
for i, fn in enumerate(nombre_img): #por cada imagen
    imgBGR = cv2.imread(fn) #leemos imagen y transformamos
    imgRGB = cv2.cvtColor(imgBGR, cv2.COLOR_BGR2RGB)
    dst = cv2.undistort(imgRGB, camera_matrix, dist_coefs) #quita distorsión
    if i < 12:
        plt.subplot(4, 3, i + 1)
        plt.imshow(dst)
plt.show()

#plot de cubos 3d a partir del patrón

objectPoints = (
    3
```

```
    * square_size
    * np.array([[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 0, 0], [0, 0, -1], [0, 1, -1], [1, 1, -1], [1, 0, -1]])
) #guardamos puntos para proyección 3d

plt.figure(figsize=figsize)
for i, fn in enumerate(nombre_img):
  try:
    imgBGR = cv2.imread(fn)
    imgRGB = cv2.cvtColor(imgBGR, cv2.COLOR_BGR2RGB)
    dst = cv2.undistort(imgRGB, camera_matrix, dist_coefs)
    imgpts = cv2.projectPoints(objectPoints, _rvecs[i], _tvecs[i], camera_matrix, dist_coefs)[0] #
      ↪ proyección de puntos
    drawn_image = draw(dst, imgpts)

    if i < 12: #guardamos hasta 12 imagenes
        plt.subplot(4, 3, i + 1)
        plt.imshow(drawn_image)
  except:
    print("Tuplex index out of range")

plt.show()
```

3.2. Stereo Vision

Obtenida la calibración, se toma el algoritmo presentado en la guía dada por el enunciado para la obtención de un mapa de disparidad. Inicialmente se carga la imagen y se genera todo el procesamiento correspondiente.

```
#
    ↪ ============================================================
    ↪
# Stereo 3D reconstruction
#
    ↪ ============================================================
    ↪
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './TestImgs/yoagain1.jpg'
img_path2 = './TestImgs/yoagain2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

```

```
# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)
```

Obtenida la imagen sin distorsión y con downsample, esta se puede integrar dentro de la función para obtener el mapa de disparidad, el cual depende si es para SGBM o BM. A continuación se presentan los códigos respectivos:

```
# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 5
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 3,
 uniquenessRatio = 1,
 speckleWindowSize = 30,
 speckleRange = 20,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()
```

```
gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
```

```


n_block = 4 # 2 en adelante
n_disp = 2


block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16


# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")

disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)


#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()
```

Concluido esto, se puede obtener el mesh 3D para la reconstrucción de escena haciendo uso de la matriz de transformación que hace uso del parámetro focal length. Se quitan los puntos con valor igual a 0, y se utiliza la función que proyecta la imagen a 3D para generar un mapa de puntos para la reconstrucción, llevando a una imagen .ply para verse en un programa como MeshLab.

```
#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
```

```
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'cara1BM.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)
```

3.3. Deep Learning

El código de Deep Learning se basa principalmente en una demo de la aplicación de Mesh R-CNN, debido a sus dificultades de implementarse. El código original puede encontrarse en [6]. Inicialmente se cargan las librerías correspondientes, proceso que demora aproximadamente 20 minutos debido a la librería PyTorch3D. Las especificidades de Colab hacen que este proceso sea muy particular, y por lo tanto la documentación normal de Mesh R-CNN resulta dificil de aplicar directamente.

```
# install dependencies: (use cu101 because colab has CUDA 10.1)
%env FORCE_CUDA=1
!pip install -U torch==1.6 torchvision==0.7 -f https://download.pytorch.org/whl/cu101/
    ↪ torch_stable.html
!pip install pyyaml==5.1 pycocotools>=2.0.1
!pip install -U fvcore==0.1.3.post20210317
!pip install -U --force-reinstall iopath==0.1.6
!pip install 'git+https://github.com/facebookresearch/detectron2.
    ↪ git@4b539e41f4b19570bc6b722fa0f516f927cdae02'
!pip install 'git+https://github.com/facebookresearch/pytorch3d.git'


#instalacion de Mesh R-CNN

import torch, torchvision
!git clone https://github.com/facebookresearch/meshrcnn.git
!cd meshrcnn && pip install -e .
```

Una vez se instale Mesh R-CNN, se debe subir la imagen a evaluar con el nombre *test.jpg*. Subida la imagen se agrega a la demostración de Mesh R-CNN con demo.py, cono lo cual se llega a una mascara y mesh de salida en la carpeta output_demo. Lo que genera esta demo es una segmentación de la imagen para tomar objetos conocidos, para posteriormente reconstruirlos en 3D, generando un archivo de mascara segmentadora y modelo 3D.

```
from google.colab import files
uploaded = files.upload()  # importar imagen como test.jpg

!python ./meshrcnn/demo/demo.py --config-file ./meshrcnn/configs/pix3d/meshrcnn_R50_FPN.
```

```
    ↪ yaml \
--input ./test.jpg --output output_demo --onlyhighest MODEL.WEIGHTS meshrcnn://
    ↪ meshrcnn_R50.pth
```

Finalmente, con la mascara resultante esta se puede visualizar y descargar, al igual que la reconstrucción 3D del objeto identificado.

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))

img = plt.imread(filename)
plt.imshow(img)
plt.axis('off')
plt.show()

plt.figure(figsize=(10,10))
img = plt.imread("./output_demo/test/0_mask_" + maskname+'.png')
plt.imshow(img)
plt.axis('off')
plt.show()

from google.colab import files
files.download('/content/semestreprimavera2022/ProyectoImagenes/output_demo/test/0_mesh_'+
    ↪ maskname + '.obj')
```

4. Pruebas y Resultados

En la presenta sección se muestran los resultados objetivos del código planteado previamente.

4.1. Calibración de la cámara

La calibración de la cámara fue hecha a partir de 88 capturas de un patrón de tablero de ajedrez blanco y negro como se indica en la documentación de cv2. Esta logro efectivamente identificar las esquinas de los cuadrados y por tanto, calibrarse apropiadamente para la eliminación de distorsión por el foco de la cámara.

Figura 5: Detección de esquinas al calibrar dentro del conjunto de calibración entregado

Seguido a esto con la proyección en 3D se notaron ciertos errores, mas estos resultaron no afectar

de forma notoria la posterior generación de mapa de disparidad. Esto puede notarse en ciertas figuras resultantes donde los cubos proyectados no se asocian a ninguna esquina particular del patron, y en ciertos casos abandonan a este.

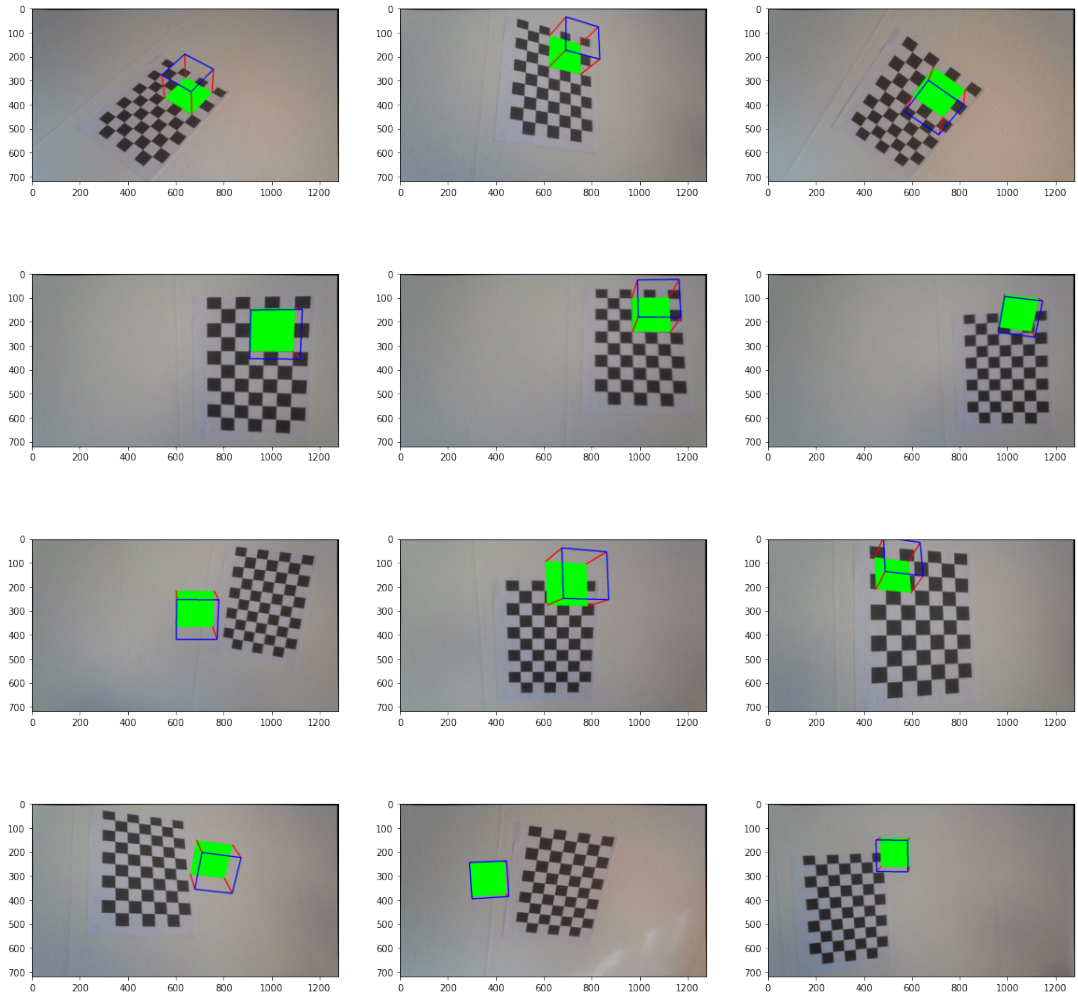

Figura 6: Proyección de cubos en imágenes sin distorsión, para comprobar calibración

4.2. Primeras pruebas con Stereo visión

Una vez fue calibrada la cámara se realizaron pruebas iniciales de visión en estéreo a partir del algoritmo. Para esto se uso, similar a la documentación de ejemplo, un par de imágenes de una persona (el autor) frente a la cámara.

Figura 7: Par de imágenes para prueba inicial

El mapa de disparidad y mesh de puntos 3D otorgados a partir de esto fueron los siguientes:

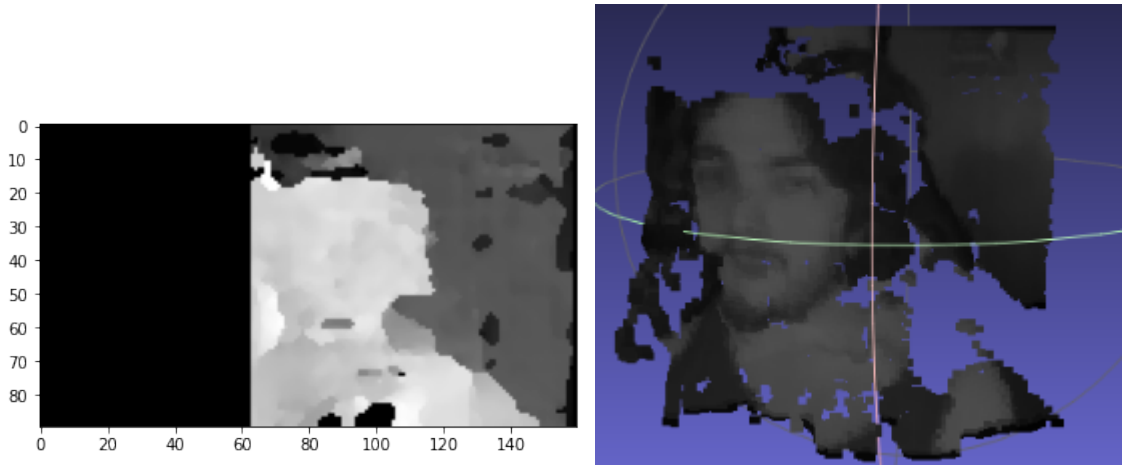

Figura 8: mapa de disparidad y reconstrucción 3D obtenida

La detección de la persona frente al fondo fue exitosa y el mesh 3D reconstruido, si bien presenta

ruido, logra diferencia de forma correcta las profundidades y genera una reconstrucción 3D apropiada a lo esperado.

4.3. Conjunto de entrenamiento

Seguido a las pruebas iniciales, se procede a generar un conjunto de entrenamiento con pares de imágenes siguiendo una metodología particular para replicar resultados. Se tomo la cámara montada en un trípode, apuntando al objeto a una distancia de un metro, tomando dos fotografías a una distancia aproximadamente de 5 centímetros de movimiento horizontal entre fotos. De este forma todas las fotos poseen una geometría reciproca entre casos para la generación del mapa de disparidad con parámetros iguales. La forma en que se capturaron las fotografías puede verse a continuación:

Figura 9: Sistema de fotografías para el proyecto, mostrando la posición de la cámara respecto al objeto

Para el conjunto de entrenamiento se tomaron 7 pares de fotos con diversos objetos de dimensiones similares, los cuales se pueden ver a continuación:

Figura 10: Objeto de entrenamiento: ball

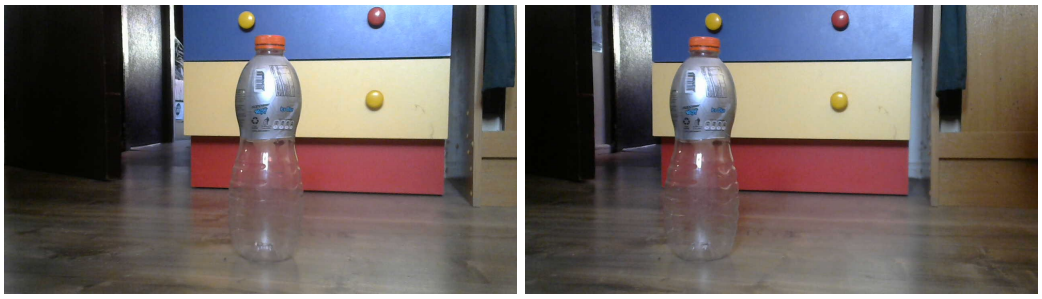

Figura 11: Objeto de entrenamiento: bottle

Figura 12: Objeto de entrenamiento: cup

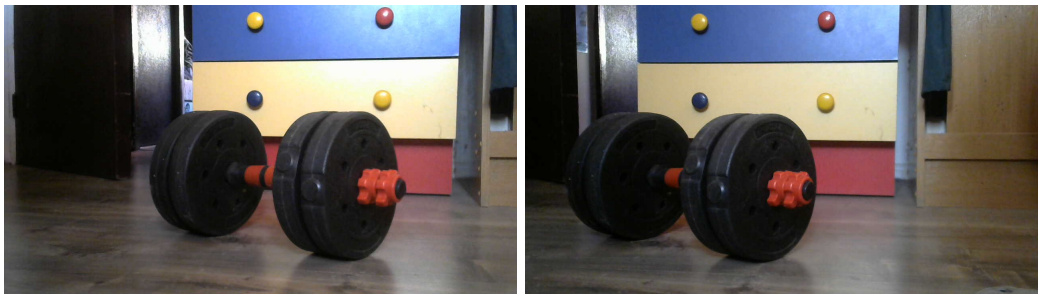

Figura 13: Objeto de entrenamiento: dumbell

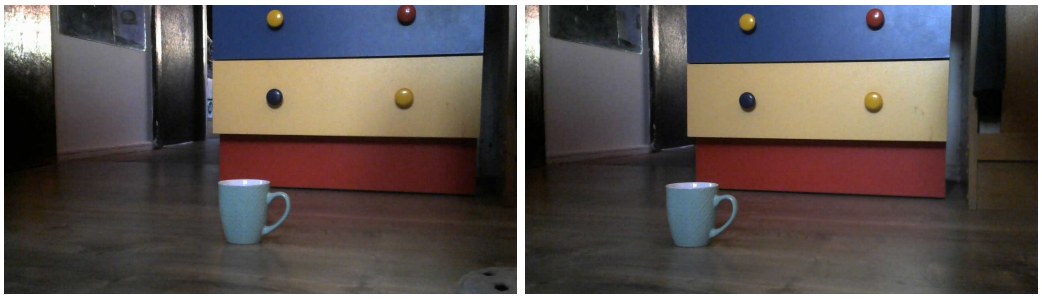

Figura 14: Objeto de entrenamiento: teacup

Figura 15: Objeto de entrenamiento: trash

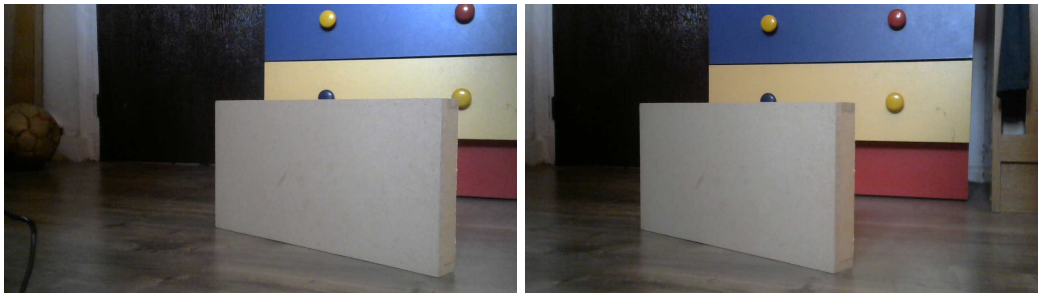

Figura 16: Objeto de entrenamiento: wood

A partir de estas capturas se procede a realizar entrenamiento para SGBM y BM.

4.3.1. SGBM

Se realizó ajuste de parámetros SGBM dentro del conjunto, para llegar a mapas de disparidad apropiados para cada imagen. Los parámetros resultantes fueron los siguientes:

Tabla 1: Parámetros utilizados en entrenamiento para SGBM

Parámetro	Valor Inicial	Valor Final
minDisparity	-1	-1
numDisparities	64	64
blockSize	5	5
uniquenessRatio	10	2
speckleWindowSize	20	20
speckleRange	15	15
disp12MaxDiff	15	15

Los mapas resultantes fueron los siguientes:

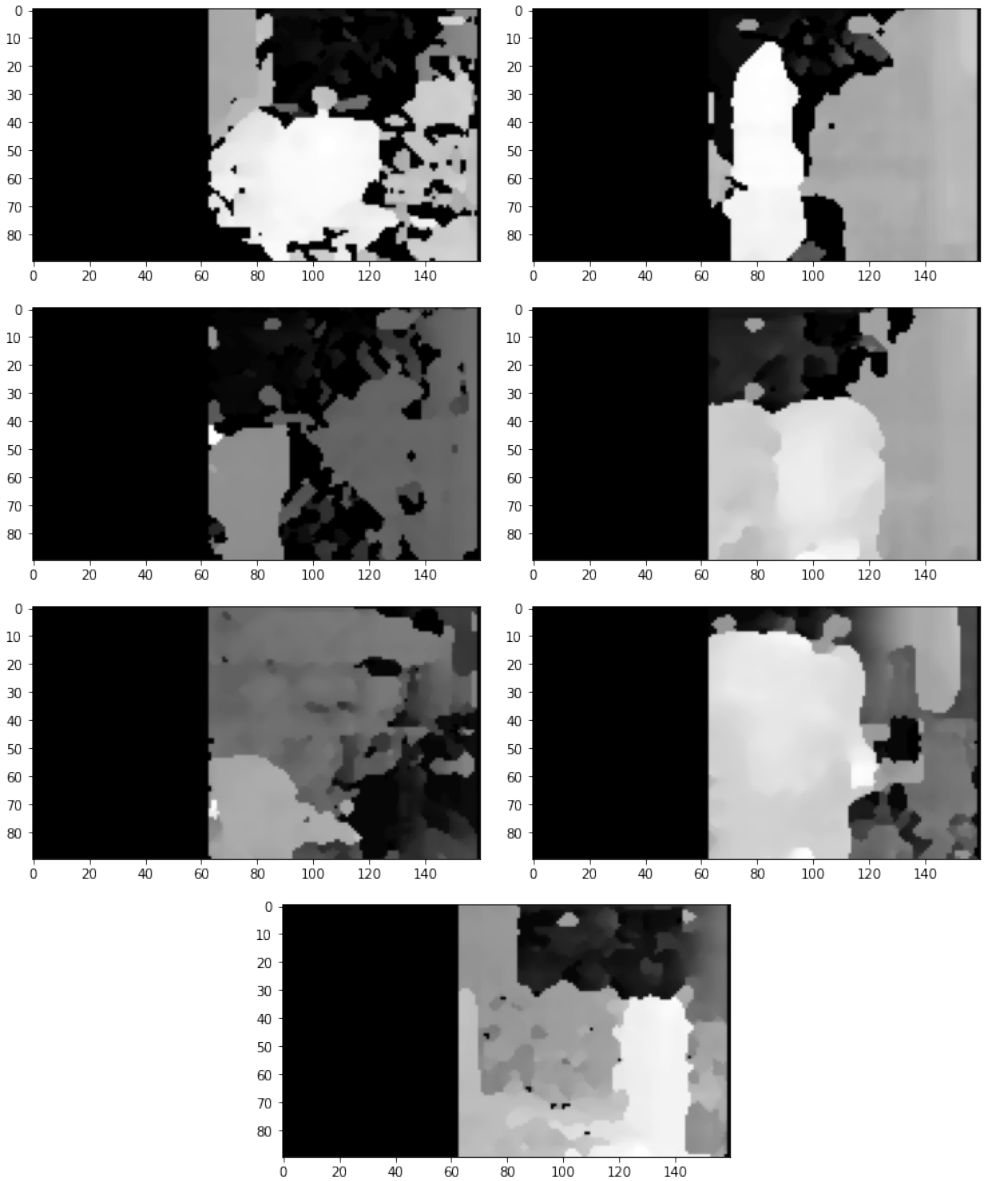

Figura 17: Mapas resultantes del entrenamiento, en orden alfabetico y cronológico (inicio de entrenamiento a final de izquierda a derecha, arriba a abajo)

4.3.2. BM

Se realizó ajuste de parámetros BM dentro del conjunto, para llegar a mapas de disparidad apropiados para cada imagen. Los parámetros resultantes fueron los siguientes:

Tabla 2: Parámetros utilizados en entrenamiento para BM

Parámetro	Valor Inicial	Valor Final
numDisparities	64	64
blockSize	7	11

Los mapas resultantes fueron los siguientes:

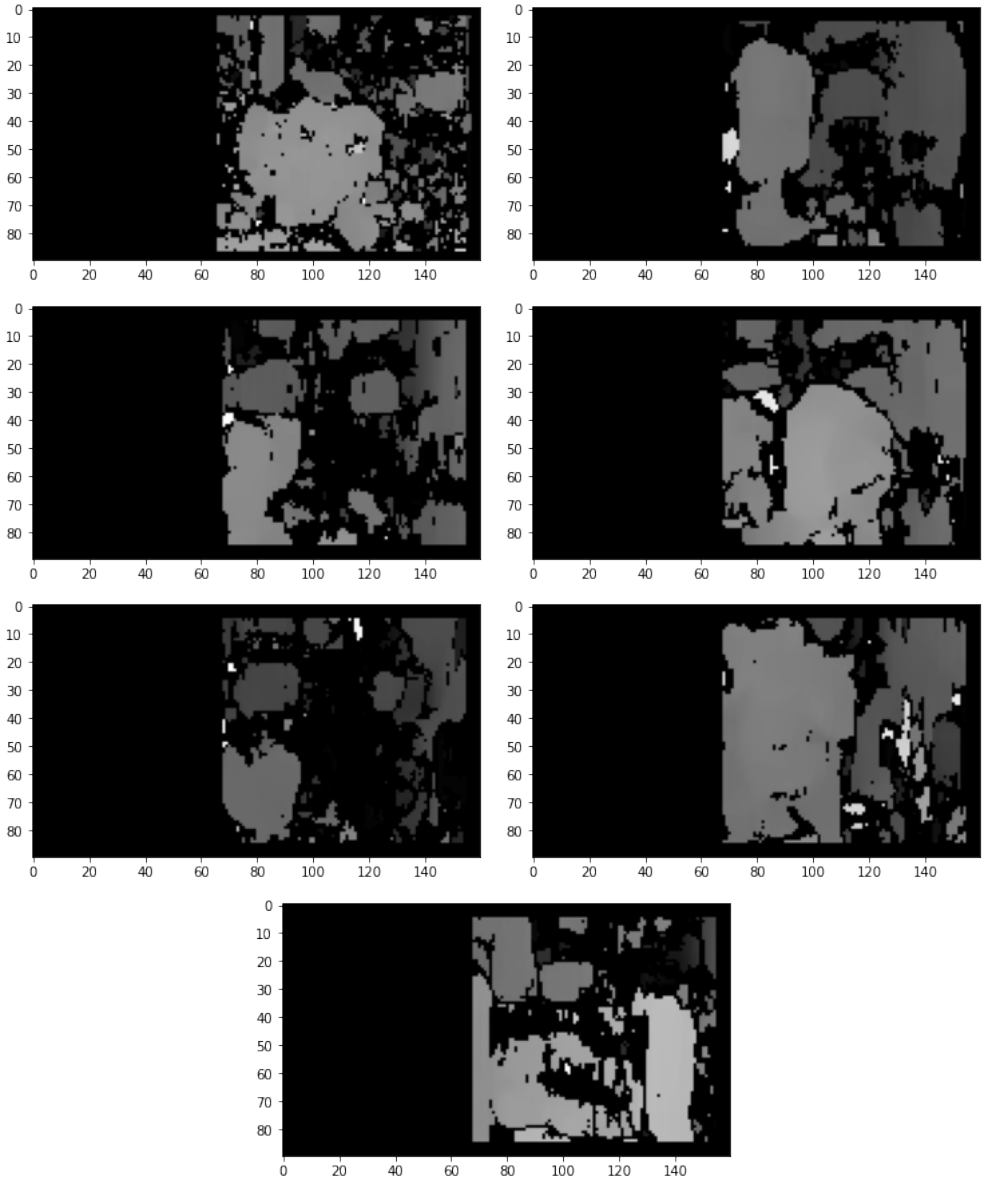

Figura 18: Mapas resultantes del entrenamiento, en orden alfabetico y cronológico (inicio de entrenamiento a final de izquierda a derecha, arriba a abajo)

4.4. Conjunto de prueba

Una vez obtenidos los parámetros a utilizar, se utilizan para generar mapas de disparidad y un mesh 3D de la imagen a reconstruir. El conjunto de prueba a utilizar fue el siguiente:

Figura 19: Objeto de prueba: cuptrash

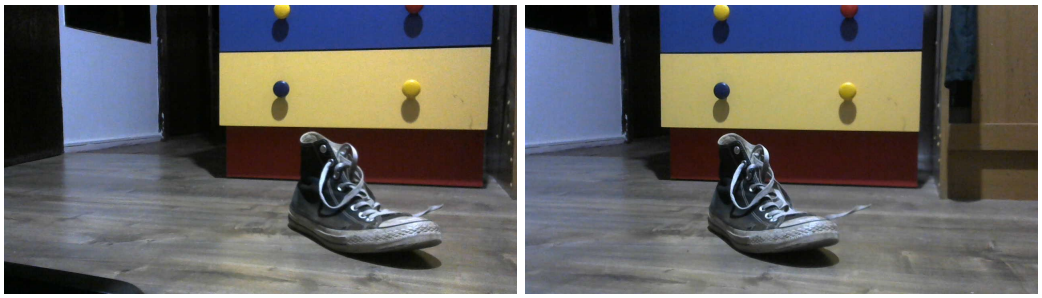

Figura 20: Objeto de prueba: sneaker

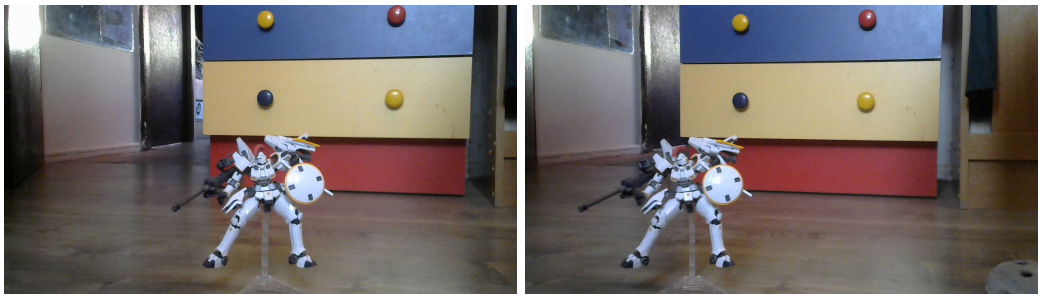

Figura 21: Objeto de prueba: tallgeese

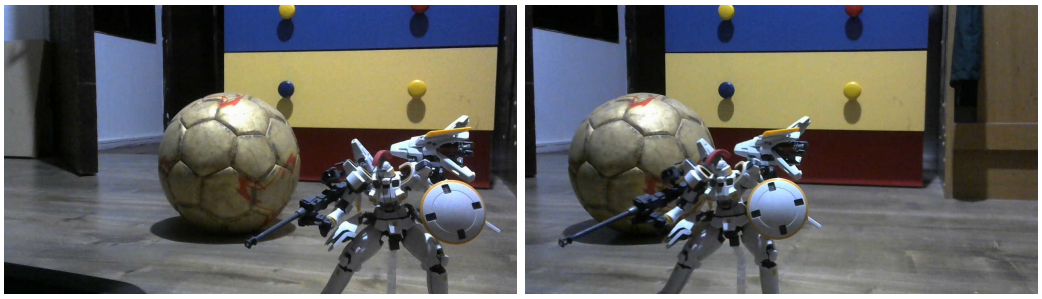

Figura 22: Objeto de prueba: tallgeeseball

La intuición del conjunto de prueba recae en entregar al algoritmo de stereo vision previamente planteado objetos conocidos, pero varios en una sola imagen, al igual que objetos distintos, como es la zapatilla y la figura, la cual es más detallada y por lo tanto puede generar más cambios entre imágenes por su geometría.

4.4.1. SGBM

Los resultados para mapas de disparidad del conjunto de prueba se presentan a continuación:

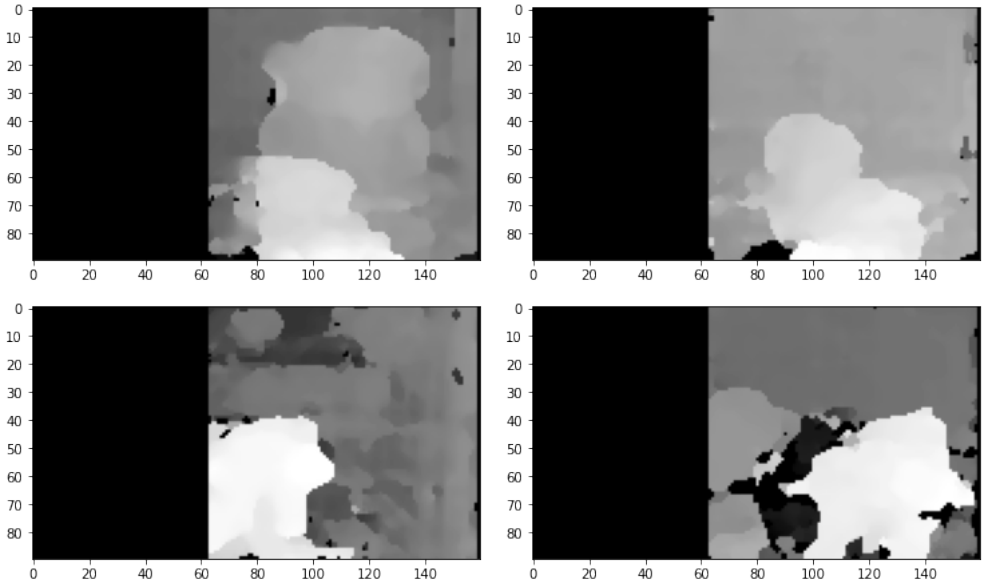

Figura 23: Mapas resultantes de prueba, en orden alfabetico a partir de SGBM

A partir de estos, los mesh 3D resultantes corresponden a los siguientes:

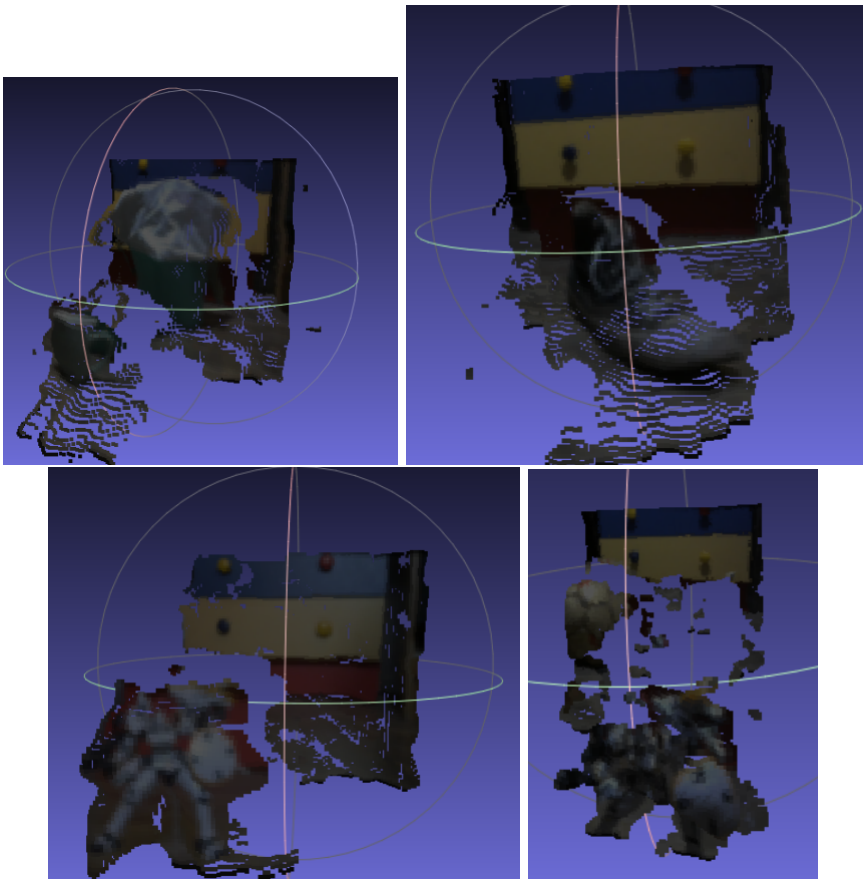

Figura 24: Mesh 3D resultantes para el conjunto de prueba a partir de SGBM

4.4.2. BM

Los resultados para mapas de disparidad del conjunto de prueba se presentan a continuación:

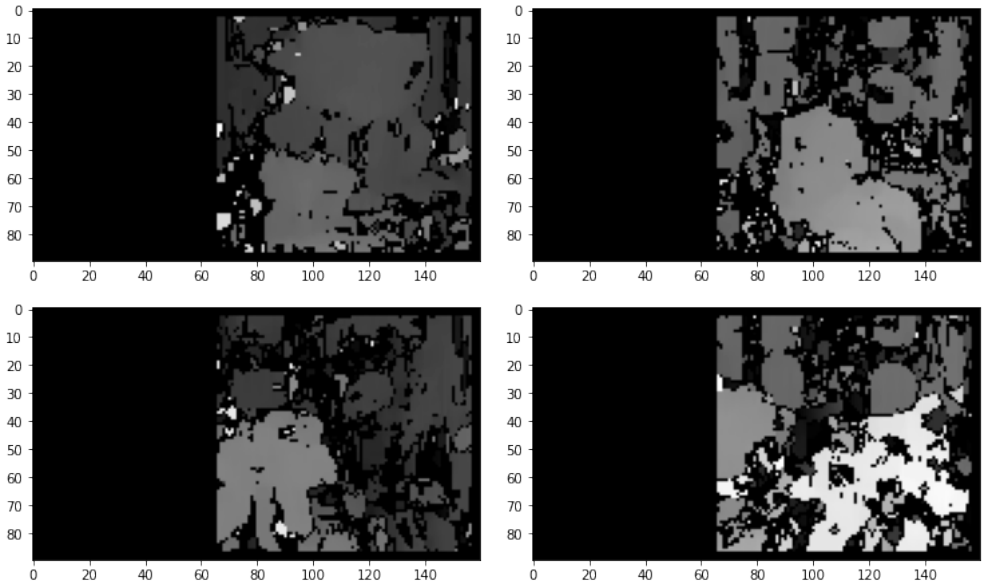

Figura 25: Mapas resultantes de prueba, en orden alfabetico a partir de BM

A partir de estos, los mesh 3D resultantes corresponden a los siguientes:

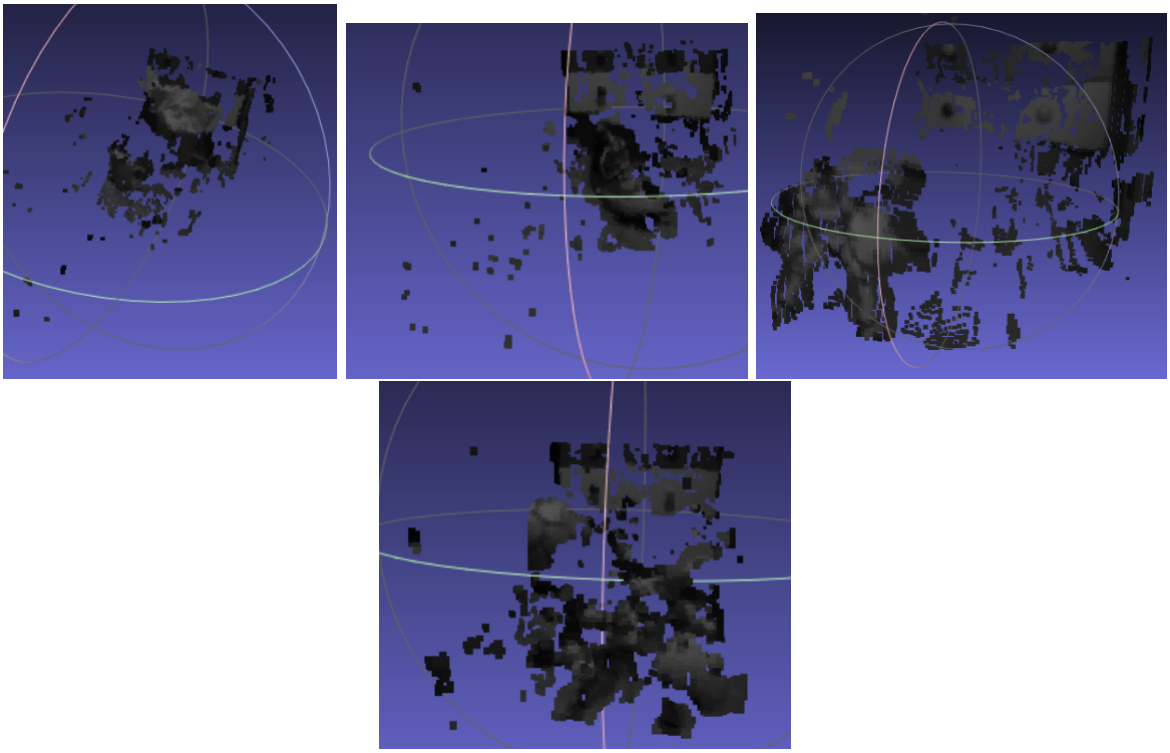

Figura 26: Mesh 3D resultantes para el conjunto de prueba a partir de BM

4.4.3. Deep Learning

Seguido al trabajo mediante Stereo Vision, se hizo prueba del algoritmo de Deep Learning Mesh R-CNN. Se tomaron las imágenes de prueba, además de imágenes extra para la prueba del algoritmo. A continuación se presentan las mascaras generadas por la segmentación de imagen:

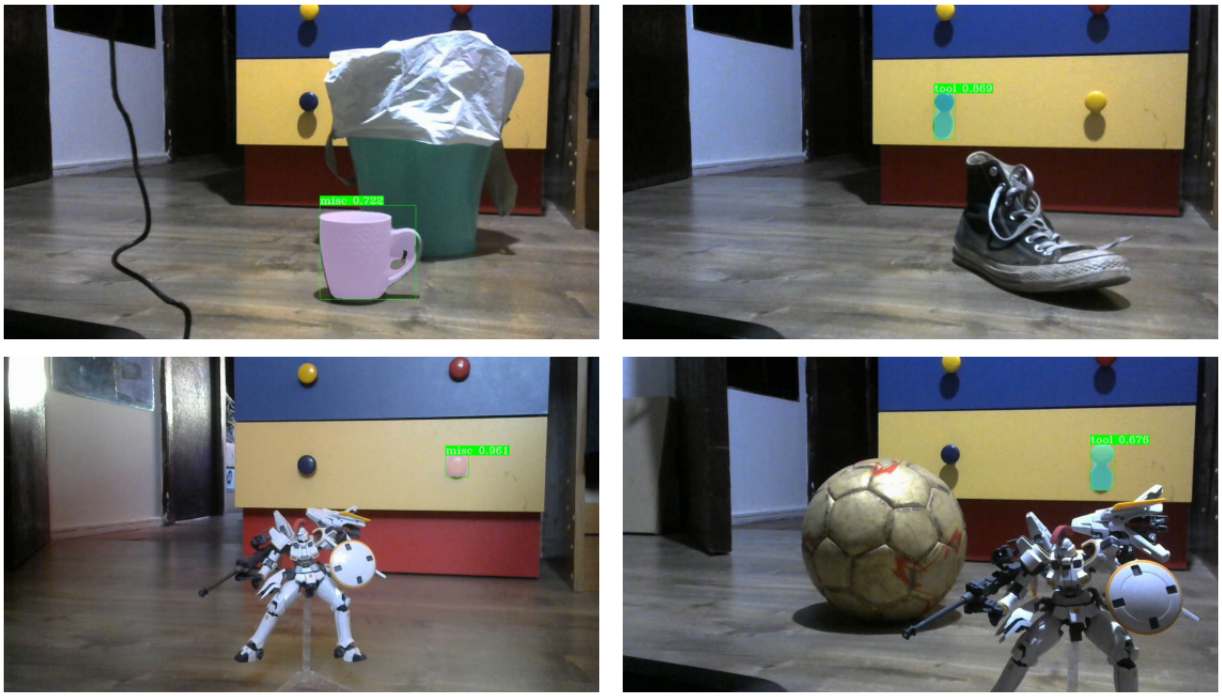

Figura 27: Mascaras obtenidas para el conjunto de prueba

Como puede notarse, ciertos objetos no son detectados por el algoritmo ya que este no se entrenó para ellos en mente. Para ver el comportamiento de la red neuronal y la reconstrucción 3D, se prueba con objetos que propiamente identifique.

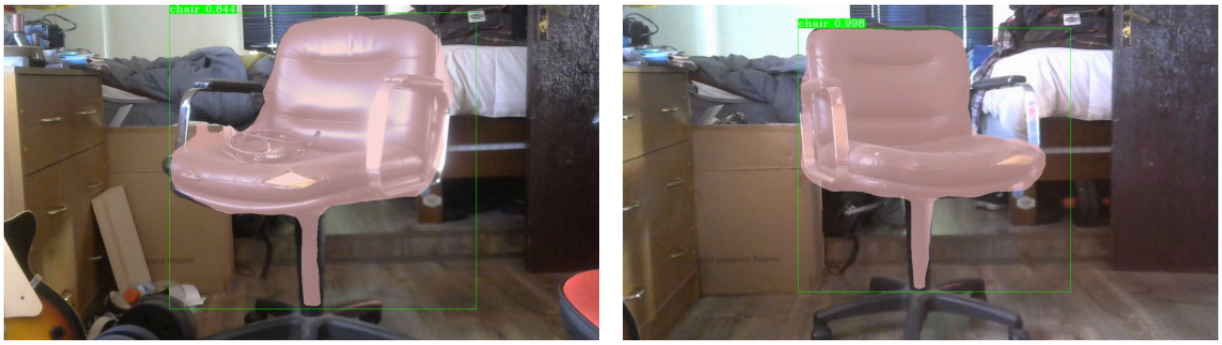

Figura 28: Mascaras para objetos conocidos por el algoritmo, sillas

A partir de esto, se presentan los mesh 3D generados a partir de los objetos detectados:

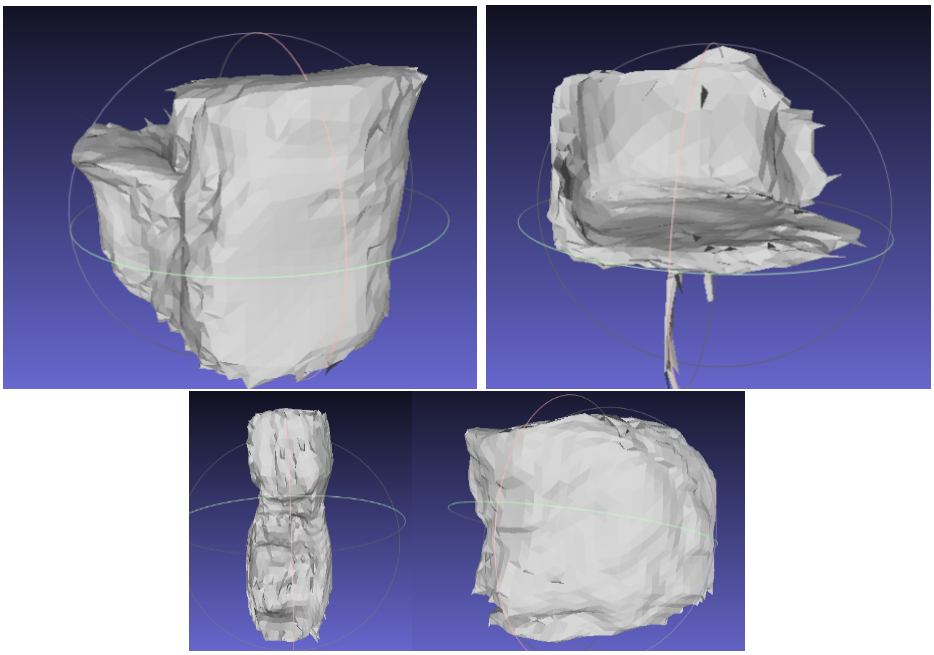

Figura 29: Mesh resultantes a partir de los objetos detectados por Deep Learning, en orden: copa, silla y detecciones erradas

5. Análisis

A partir de los resultados, se puede observar principalmente la exitosa reconstrucción de imágenes 3D tanto para SGBM como BM, ya que ambas lograron diferenciar el objeto principal respecto al fondo de cada imagen. Se identifica al contrastar los resultados de SGBM y BM, que los primeros hacen divisiones más solidas entre fondo y objeto, mas la última tiene mayor cantidad de ruido pero mejor segmentación de los objetos identificados. Esto puede verse, por ejemplo, en los mesh que involucran la figura, ya que SGBM la separa como un bloque completo, mientras que BM detalla elementos de la figura como el espacio entre las piernas o en sus extremidades por general. Sin embargo, estos últimos además presentan mucho ruido en comparación, lo cual genera dificultades.

Resulta de interés ver en detalle, como ejemplo, la zapatilla y la taza con la basura de fondo, ya que sen las reconstrucciones 3D que mejor forman la diferencia de profundidad, siendo en la taza y basura vistas como tres capas (objeto frontal, intermedio y fondo), mientras que en la zapatilla se observa una profundidad variada desde el talón a la punta. Sin embargo, debe notarse el ruido principalmente en bordes, ya que el objeto no se divide de forma uniforme, lo cual puede buscar ajustarse mediante la ventana Speckle, o al tomar imágenes más precisas o con fondos más planos.

Para las mascaras y reconstrucciones 3D resultantes de Deep Learning, se destaca como el algoritmo al no reconocer los objetos utilizados para el conjunto test, no puede generar una reconstrucción idónea y en vez de eso, identifica objetos falsos positivos en el fondo de la imagen. Solo se identifica la copa (como objeto miscelaneo) y una nueva imagen de una silla como elementos encontrados por el algoritmo, lo cual es consecuencia de su entrenamiento principalmente con inmuebles. De cambiar el entrenamiento e incluir ejemplos de pelotas, o calzado, el algoritmo debería ser capaz de reconstruir estos, mas este proceso resultaría sumamente costoso para el alcance de este proyecto. Esto permite identificar una notoria diferencia entre Stereo Vision y Deep Learning: Stereo Vision puede reconstruir cualquier cosa siempre y cuando se pueda sacar capturas de alta fidelidad del objeto deseado, mas estas no son tan detalladas. Por otro lado, Deep Learning puede generar reconstrucciones de alto detalle, mas estas dependen del entrenamiento del algoritmo para poder detectar un objeto particular, lo cual hace las posibilidades más cerradas.

6. Conclusión

Mediante el trabajo realizado, exitosamente se llegó a la reconstrucción de imágenes tomadas por el alumno a un ambiente 3D por un mesh de puntos y algoritmos de deep learning, siendo vital el entendimiento de la importancia de cada uno en distintas circunstancias. Stereo Vision resuelve el problema, mas no de forma detallada y con mucho ruido de haber un fondo complicado. Por otro lado, Deep Learning puede resolver el problema con mucho detalle y un modelo completamente 3D (no solo un grupo de puntos con distinta profundidad), mas este es especifico a lo entrenado del modelo. La diversidad de las herramientas en visión computacional, por tanto, genera la decisión del usuario de encontrar la mejor alternativa para cada uno de estos problemas e implementarla.

Resulta vital, por tanto, mantener una metodología y orden en la creación del sistema que toma capturas para Stereo Vision, ya que se debe encontrar valores y parámetros ideales para la medición particular. Factores como la camara, iluminación, fondo, detalle de objetos, entre varios otros hacen que la elección de parámetros sea de mucha importancia para un reconstrucción fiel, por lo cual su consistencia en la geometría es fundamental.

Con esto, queda pendiente para futuro análisis observar como cambian los resultados de otorgar menor cantidad de ejemplos para la calibración, o tomar fotos similares con distinta iluminación o fondos, para ver como estos afectan a la imagen, su geometría y finalmente, resultados. Por otro lado, queda en el interés del alumno ver otros algoritmos de deep learning que puedan recrear objetos, inclusive los previamente no conocidos en su entrenamiento.

Referencias

[1] Gkioxari, G., Malik, J., & Johnson, J. (2019). Mesh r-cnn. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 9785-9795).

[2] OpenCV (2022). Camera Calibration and 3D Reconstruction. https://docs.opencv.org/4.x/d9/db7/tutorial_py_table_of_contents_calib3d.html

[3] Repositorio GitHub de OpenCV, código sample de reconstrucción, calibrate.py. https://github.com/opencv/opencv/blob/master/samples/python/calibrate.py

[4] Padierna, O. (2019). Tutorial: Stereo 3D reconstruction with openCV using an iPhone camera. Part I. Becoming Human: Artificial Intelligence Magazine. https://becominghuman.ai/stereo-3d-reconstruction-with-opencv-using-an-iphone-camera-part-i-c013907d1ab5

[5] Repositorio GitHub de Mesh R-CNN https://github.com/facebookresearch/meshrcnn

[6] Shashwat, N. Tono, A. Demo Mesh R-CNN en Google Colab. https://colab.research.google.com/drive/1eQLZrNYRZMo9zdnGGccE0hFswGiinO-Z?usp=sharing#scrollTo=QkR0aNt6n0Fh

Anexo: Código completo

```
# -*- coding: utf-8 -*-
"""Proyecto Imagenes Ariel

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1Nh9KCovFoq-ShYTJowaYikJh8mnbyAd9
"""

# Commented out IPython magic to ensure Python compatibility.
#importe de imagenes desde repositorio en github
import sys
if 'google.colab' in sys.modules:
    import subprocess
    subprocess.call("pip install -U opencv-python".split())

import os
import re
from os.path import exists

# %matplotlib inline

project_name = "Proyecto_imagenes"
if not exists(project_name):
  # clone and install
  !git clone -q --recursive https://github.com/Marmotato/semestreprimavera2022

```

```python

import sys

import cv2
import numpy as np
import glob
import matplotlib.pyplot as plt
import math


# Este paquete solo se debe usar si se usa colaboratory
from google.colab.patches import cv2_imshow

"""# 0. Funciones auxiliares"""

# Dibujar imagen con  cubos para mostrar forma 3D
def draw(img, imgpts): #funcion original de cv2
  imgpts = np.int32(imgpts).reshape(-1, 2)
  # draw ground floor in green
  img = cv2.drawContours(img, [imgpts[:4]], -1, (0, 255, 0), -1)
  # draw pillars in blue color
  for i, j in zip(range(4), range(4, 8)):
    img = cv2.line(img, tuple(imgpts[i]), tuple(imgpts[j]), (255), 3)
  # draw top layer in red color
  img = cv2.drawContours(img, [imgpts[4:]], -1, (0, 0, 255), 3)
  return img

# Downsample
def downsample_image(image, reduce_factor):
   for i in range(0,reduce_factor):
      #Imagen a color o grayscale
      if len(image.shape) > 2:
         row,col = image.shape[:2]
      else:
         row,col = image.shape
      image = cv2.pyrDown(image, dstsize= (col//2, row // 2))
   return image

# Crea salida ply para mesh
def create_output(vertices, colors, filename):
   colors = colors.reshape(-1,3)
   vertices = np.hstack([vertices.reshape(-1,3),colors])

   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
    '''
  with open(filename, 'w') as f:
    f.write(ply_header %dict(vert_num=len(vertices)))
    np.savetxt(f,vertices,'%f %f %f %d %d %d')

"""# 1. Calibración de camara

Camara usada: Logitech C270
"""

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/semestreprimavera2022/ProyectoImagenes/

square_size = 2.5 #tamaño del cuadrado estimado
img_mask = "./ImgCalibration/*.jpg" #directorio de imagenes de calibración
pattern_size = (9, 6) #cantidad de cuadrados en patrón a analizar
figsize = (20, 20) #tamaño para imagenes

nombre_img = glob.glob(img_mask)
numero_img = len(nombre_img)

pattern_points = np.zeros((np.prod(pattern_size), 3), np.float32) #preparamos para guardar indices
pattern_points[:, :2] = np.indices(pattern_size).T.reshape(-1, 2)
pattern_points *= square_size

obj_points = [] #preparamos para guardar puntos
img_points = []
h, w = cv2.imread(nombre_img[0]).shape[:2]

plt.figure(figsize=figsize)

for i, fn in enumerate(nombre_img):
   print("processing %s... " % fn)

   #transforma imagen cargada a rgb y gray
   imgBGR = cv2.imread(fn)
   imgRGB = cv2.cvtColor(imgBGR, cv2.COLOR_BGR2RGB)
   imggray = cv2.cvtColor(imgRGB, cv2.COLOR_RGB2GRAY)

   assert w == imggray.shape[1] and h == imggray.shape[0], f"size: {imggray.shape[1]} x {imggray.shape[0]}" #imagenes deben tener el mismo tamaño
   found, corners = cv2.findChessboardCorners(imggray, pattern_size) #usamos funcion para encontrar esquinas, booleano identifica si estas se encontraron o no

   if not found: #Si no encuentra el patrón, imprimir para ver cuantas calibraciones fueron correctas
      print("patrón no encontrado")
      continue

   if i < 12:
```

```
        img_w_corners = cv2.drawChessboardCorners(imgRGB, pattern_size, corners, found)
        plt.subplot(4, 3, i + 1)
        plt.imshow(img_w_corners)

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


plt.show() #mostramos 12 patrones encontrados

# calculate camera distortion
rms, camera_matrix, dist_coefs, _rvecs, _tvecs = cv2.calibrateCamera(obj_points, img_points, (w, h), None, None)

ret, K, dist, rvecs, tvecs = rms, camera_matrix, dist_coefs, _rvecs, _tvecs

#Save parameters into numpy file
np.save("./camera_params/ret", ret)
np.save("./camera_params/K", K)
np.save("./camera_params/dist", dist)
np.save("./camera_params/rvecs", rvecs)
np.save("./camera_params/tvecs", tvecs)

K, dist #coeficientes utilizados para la calibración

#obtenemos focal length

S = cv2.calibrationMatrixValues(camera_matrix, (720,1280), 3.58, 2.02) #valores del sensor de la camara
focal_length = S[2]

# quitar distorsión a partir de la calibración
plt.figure(figsize=figsize)
for i, fn in enumerate(nombre_img): #por cada imagen
    imgBGR = cv2.imread(fn) #leemos imagen y transformamos
    imgRGB = cv2.cvtColor(imgBGR, cv2.COLOR_BGR2RGB)
    dst = cv2.undistort(imgRGB, camera_matrix, dist_coefs) #quita distorsión
    if i < 12:
        plt.subplot(4, 3, i + 1)
        plt.imshow(dst)
plt.show()

#plot de cubos 3d a partir del patrón

objectPoints = (
    3
    * square_size
    * np.array([[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 0, 0], [0, 0, -1], [0, 1, -1], [1, 1, -1], [1, 0, -1]])
) #guardamos puntos para proyección 3d

plt.figure(figsize=figsize)
```

```
for i, fn in enumerate(nombre_img):
  try:
    imgBGR = cv2.imread(fn)
    imgRGB = cv2.cvtColor(imgBGR, cv2.COLOR_BGR2RGB)
    dst = cv2.undistort(imgRGB, camera_matrix, dist_coefs)
    imgpts = cv2.projectPoints(objectPoints, _rvecs[i], _tvecs[i], camera_matrix, dist_coefs)[0] #
      ↪ proyección de puntos
    drawn_image = draw(dst, imgpts)

    if i < 12: #guardamos hasta 12 imagenes
        plt.subplot(4, 3, i + 1)
        plt.imshow(drawn_image)
  except:
    print("Tuplex index out of range")

plt.show()

"""# 2. Reconstrucción Stereo 3D: Pruebas iniciales

## 2.1 Cara
"""

#
    ↪ ==========================================================================
    ↪
# Stereo 3D reconstruction
#
    ↪ ==========================================================================
    ↪
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './TestImgs/yoagain1.jpg'
img_path2 = './TestImgs/yoagain2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

```

```
"""### 2.1.1 Semi-Global Matching"""

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 5
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 3,
 uniquenessRatio = 1,
 speckleWindowSize = 30,
 speckleRange = 20,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on
    ↪ computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
```

```
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'cara1SGBM.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

"""### 2.1.2 Block Matching"""

gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso


n_block = 4 # 2 en adelante
n_disp = 2


block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16


# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")

disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)


#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
```

```
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'cara1BM.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

"""## 2.2 Guitarra"""

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './TestImgs/guitar1.jpg'
img_path2 = './TestImgs/guitar2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
```

```
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

"""### 2.2.1 Semi-Global Matching"""

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 5
min_disp = -1
max_disp = 31 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 5,
 speckleWindowSize = 5,
 speckleRange = 5,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
```

```
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'guitar1SGBM.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

"""### 2.2.2 Block Matching"""

gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso


n_block = 5 # 2 en adelante
n_disp = 2


block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16


# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")

disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)


#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
```

```
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'guitar1BM.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

"""## 2.3 Silla"""

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './TestImgs/chairclear1.jpg'
```

```
img_path2 = './TestImgs/chairclear2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

"""### 2.3.1 Semi-Global Matching"""

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 7
min_disp = -1
max_disp = 15 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 15,
 speckleWindowSize = 5,
 speckleRange = 5,
 disp12MaxDiff = 5,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
```

```
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'chair1SGBM.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

"""### 2.3.2 Block Matching"""

gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso


n_block = 6 # 2 en adelante
n_disp = 1


block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16


# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
```

```
                        blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")

disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)


#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'chair1BM.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

"""# 3. Train & Test: SGBM

## 3.1 Train
"""

#
    ==========================================================
```

```
↪
# Stereo 3D reconstruction
#
↪ ==========================================================
↪
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/ball1.jpg'
img_path2 = './train/ball2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 6
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 10,
 speckleWindowSize = 20,
 speckleRange = 15,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

```

```
#
    ↪ ==========================================================
    ↪
# Stereo 3D reconstruction
#
    ↪ ==========================================================
    ↪
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/bottle1.jpg'
img_path2 = './train/bottle2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 7
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 10,
 speckleWindowSize = 20,
 speckleRange = 15,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
```

```
plt.show()

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/cup1.jpg'
img_path2 = './train/cup2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 7
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 10,
 speckleWindowSize = 20,
 speckleRange = 15,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

```

```
#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/dumbell1.jpg'
img_path2 = './train/dumbell2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 7
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 2,
 speckleWindowSize = 20,
 speckleRange = 15,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
```

```
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

# ==========================================================
# Stereo 3D reconstruction
# ==========================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/teacup1.jpg'
img_path2 = './train/teacup2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 7
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 2,
 speckleWindowSize = 20,
 speckleRange = 15,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
```

```
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/trash1.jpg'
img_path2 = './train/trash2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 5
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 2,
 speckleWindowSize = 20,
 speckleRange = 15,
 disp12MaxDiff = 15,
```

```
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/wood1.jpg'
img_path2 = './train/wood2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 5
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 2,
 speckleWindowSize = 20,
```

```
 speckleRange = 15,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

"""## 3.2 Test"""

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './test/cuptrash1.jpg'
img_path2 = './test/cuptrash2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 5
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
```

```
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 2,
 speckleWindowSize = 20,
 speckleRange = 15,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'cuptrashsgbm.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

#
    ==========================================================
```

```
↪
# Stereo 3D reconstruction
#
↪ ==========================================================
↪
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './test/sneaker1.jpg'
img_path2 = './test/sneaker2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 5
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 2,
 speckleWindowSize = 20,
 speckleRange = 15,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

```

```
#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'sneakersgbm.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './test/tallgeese1.jpg'
img_path2 = './test/tallgeese2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
```

```
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 5
min_disp = -1
max_disp = 63 #min_disp * 9
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 2,
 speckleWindowSize = 20,
 speckleRange = 15,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
```

```
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'tallgeesesgbm.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

# ==========================================================
# Stereo 3D reconstruction
# ==========================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './test/tallgeeseball1.jpg'
img_path2 = './test/tallgeeseball2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
win_size = 5
min_disp = -1
max_disp = 63 #min_disp * 9
```

```
num_disp = max_disp - min_disp # Needs to be divisible by 16
# Creacion de objeto block matching.
stereo = cv2.StereoSGBM_create(minDisparity= min_disp,
 numDisparities = num_disp,
 blockSize = 5,
 uniquenessRatio = 2,
 speckleWindowSize = 20,
 speckleRange = 15,
 disp12MaxDiff = 15,
 P1 = 8*3*win_size**2,#8*3*win_size**2,
 P2 =32*3*win_size**2) #32*3*win_size**2)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)

#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on
    ↪ computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'tallgeeseballsgbm.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)
```

```

"""# 4. Train & Test: BM

## 4.1 Train
"""

# ==========================================================
# Stereo 3D reconstruction
# ==========================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/ball1.jpg'
img_path2 = './train/ball2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)


# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso


```

```
n_block = 3 # 2 en adelante
n_disp = 4


block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16


# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")

disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)


#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

# ============================================================
# Stereo 3D reconstruction
# ============================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/bottle1.jpg'
img_path2 = './train/bottle2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)
gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)
```

```
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
n_block = 5 # 2 en adelante
n_disp = 4

block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16

# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/cup1.jpg'
img_path2 = './train/cup2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
```

```
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)
gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
n_block = 5 # 2 en adelante
n_disp = 4

block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16

# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/dumbell1.jpg'
img_path2 = './train/dumbell2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
```

```
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)
gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
n_block = 5 # 2 en adelante
n_disp = 4

block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16

# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
```

```
img_path1 = './train/teacup1.jpg'
img_path2 = './train/teacup2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)
gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
n_block = 5 # 2 en adelante
n_disp = 4

block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16

# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
```

```
↪
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/trash1.jpg'
img_path2 = './train/trash2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)
gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
n_block = 5 # 2 en adelante
n_disp = 4

block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16

# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

```

```
#
    ↪ ==========================================================
    ↪
# Stereo 3D reconstruction
#
    ↪ ==========================================================
    ↪
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './train/wood1.jpg'
img_path2 = './train/wood2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)
gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
n_block = 5 # 2 en adelante
n_disp = 4

block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16

# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
```

```
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

"""## 4.2 Test"""

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './test/cuptrash1.jpg'
img_path2 = './test/cuptrash2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
n_block = 3 # 2 en adelante
```

```
n_disp = 4

block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16

# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'cuptrashbm.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

# ==========================================================================
```

```
# Stereo 3D reconstruction
# ==========================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './test/sneaker1.jpg'
img_path2 = './test/sneaker2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
n_block = 3 # 2 en adelante
n_disp = 4

block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16

# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
```

```
#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'sneakerbm.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './test/tallgeese1.jpg'
img_path2 = './test/tallgeese2.jpg'
```

```
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
n_block = 3 # 2 en adelante
n_disp = 4

block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16

# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
```

```
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'tallgeesebm.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

# ==========================================================================
# Stereo 3D reconstruction
# ==========================================================================
# Carga de parametros
ret = np.load('./camera_params/ret.npy')
K = np.load('./camera_params/K.npy')
dist = np.load('./camera_params/dist.npy')
# Path de par de imagenes a probar
img_path1 = './test/tallgeeseball1.jpg'
img_path2 = './test/tallgeeseball2.jpg'
# Lectura de imagenes
img_1 = cv2.imread(img_path1)
img_2 = cv2.imread(img_path2)
# Alto y ancho de imagenes
h,w = img_2.shape[:2]

# Obtención de la mejor matriz para la camara a partir de los parámetros
new_camera_matrix, roi = cv2.getOptimalNewCameraMatrix(K,dist,(w,h),1,(w,h))
# Remover distorsión de camara
img_1_undistorted = cv2.undistort(img_1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(img_2, K, dist, None, new_camera_matrix)
```

```
# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

gray1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)

# Remover distorsión de camara
img_1_undistorted = cv2.undistort(gray1, K, dist, None, new_camera_matrix)
img_2_undistorted = cv2.undistort(gray2, K, dist, None, new_camera_matrix)

# Downsampling de imagenes
img_1_downsampled = downsample_image(img_1_undistorted,3)
img_2_downsampled = downsample_image(img_2_undistorted,3)

# Creando mapa de disparidad
# Parametros de disparidad, ajustados para cada caso
n_block = 3 # 2 en adelante
n_disp = 4

block = 1+(2*n_block) # must be odd, be within 5..255
num_disp = 16*n_disp # Needs to be divisible by 16

# Creacion de objeto block matching.
stereo = cv2.StereoBM_create(numDisparities=num_disp,
                             blockSize=block)
#Compute disparity map
print ("\nComputing the disparity  map...")
disparity_map = stereo.compute(img_1_downsampled, img_2_downsampled)
#Show disparity map before generating 3D cloud to verify that point cloud will be usable.
plt.imshow(disparity_map,'gray')
plt.show()

#Generate  point cloud.
print ("\nGenerating the 3D map...")
#Get new downsampled width and height
h,w = img_2_downsampled.shape[:2]
#Load focal length.
#Perspective transformation matrix
#This transformation matrix is from the openCV documentation, didn't seem to work for me.
Q = np.float32([[1,0,0,-w/2.0],
    [0,-1,0,h/2.0],
    [0,0,0,-focal_length],
    [0,0,1,0]])
#This transformation matrix is derived from Prof. Didier Stricker's power point presentation on computer vision.
#Link : https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws14-15/3DCV_lec01_camera.pdf
Q2 = np.float32([[1,0,0,0],
    [0,-1,0,0],
    [0,0,focal_length*0.05,0], #Focal length multiplication obtained experimentally.
    [0,0,0,1]])
```

```
#Reproject points into 3D
points_3D = cv2.reprojectImageTo3D(disparity_map, Q2)
#Get color points
colors = cv2.cvtColor(img_1_downsampled, cv2.COLOR_BGR2RGB)
#Get rid of points with value 0 (i.e no depth)
mask_map = disparity_map > disparity_map.min()
#Mask colors and points.
output_points = points_3D[mask_map]
output_colors = colors[mask_map]
#Define name for output file
output_file = 'tallgeeseballbm.ply'
#Generate point cloud
print ("\n Creating the output file... \n")
create_output(output_points, output_colors, output_file)

"""# 5. Deep Learning: MeshRCNN """

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/semestreprimavera2022/ProyectoImagenes/

!git clone -q --recursive https://github.com/facebookresearch/detectron2

"""## Detectron 2"""

!python -m pip install pyyaml==5.1
import sys, os, distutils.core

dist = distutils.core.run_setup("./detectron2/setup.py")
!python -m pip install {' '.join([f"'{x}'" for x in dist.install_requires])}
sys.path.insert(0, os.path.abspath('./detectron2'))

# Properly install detectron2. (Please do not install twice in both ways)
# !python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'

import torch, detectron2
!nvcc --version
TORCH_VERSION = ".".join(torch.__version__.split(".")[:2])
CUDA_VERSION = torch.__version__.split("+")[-1]
print("torch: ", TORCH_VERSION, "; cuda: ", CUDA_VERSION)
print("detectron2:", detectron2.__version__)

# Some basic setup:
# Setup detectron2 logger
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import some common libraries
import numpy as np
import os, json, cv2, random
from google.colab.patches import cv2_imshow
```

```
# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog

!wget http://images.cocodataset.org/val2017/000000439715.jpg -q -O input.jpg
im = cv2.imread("./input.jpg")
cv2_imshow(im)

cfg = get_cfg()
# add project-specific config (e.g., TensorMask) here if you're not running a model in detectron2's core library
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5  # set threshold for this model
# Find a model from detectron2's model zoo. You can use the https://dl.fbaipublicfiles... url as well
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
predictor = DefaultPredictor(cfg)
outputs = predictor(im)

# We can use 'Visualizer' to draw the predictions on the image.
v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])

im = cv2.imread("/content/semestreprimavera2022/ProyectoImagenes/TestImgs/chair2.jpg")
cv2_imshow(im)

cfg = get_cfg()
# add project-specific config (e.g., TensorMask) here if you're not running a model in detectron2's core library
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5  # set threshold for this model
# Find a model from detectron2's model zoo. You can use the https://dl.fbaipublicfiles... url as well
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
predictor = DefaultPredictor(cfg)
outputs = predictor(im)

# We can use 'Visualizer' to draw the predictions on the image.
v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/semestreprimavera2022/ProyectoImagenes/
```

```
"""## Instalacion completa"""

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/semestreprimavera2022/ProyectoImagenes/

# Commented out IPython magic to ensure Python compatibility.
# install dependencies: (use cu101 because colab has CUDA 10.1)
# %env FORCE_CUDA=1
!pip install -U torch==1.6 torchvision==0.7 -f https://download.pytorch.org/whl/cu101/
    ↪ torch_stable.html
!pip install pyyaml==5.1 pycocotools>=2.0.1
!pip install -U fvcore==0.1.3.post20210317
!pip install -U --force-reinstall iopath==0.1.6
!pip install 'git+https://github.com/facebookresearch/detectron2.
    ↪ git@4b539e41f4b19570bc6b722fa0f516f927cdae02'
!pip install 'git+https://github.com/facebookresearch/pytorch3d.git'

import torch, torchvision
!git clone https://github.com/facebookresearch/meshrcnn.git
!cd meshrcnn && pip install -e .

# pix3d data
!./meshrcnn/datasets/pix3d/download_pix3d.sh

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/semestreprimavera2022/ProyectoImagenes

"""## Demo : Obtención de mascara"""

from google.colab import files
uploaded = files.upload()  # importar imagen como test.jpg

!python ./meshrcnn/demo/demo.py --config-file ./meshrcnn/configs/pix3d/meshrcnn_R50_FPN.
    ↪ yaml \
--input ./test.jpg --output output_demo --onlyhighest MODEL.WEIGHTS meshrcnn://
    ↪ meshrcnn_R50.pth

"""## Visualización"""

filename = 'test.jpg' #nombre como test.jpg o del archivo subido
maskname = 'chair_0.994' # Cambiar nombre al de la mascara en 'output_demo/test/0
    ↪ _mask_sofa_1.000.png'

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))

img = plt.imread(filename)
plt.imshow(img)
```

```
plt.axis('off')
plt.show()

plt.figure(figsize=(10,10))
img = plt.imread("./output_demo/test/0_mask_" + maskname+'.png')
plt.imshow(img)
plt.axis('off')
plt.show()

"""## Descarga del Mesh resultante"""

from google.colab import files
files.download('/content/semestreprimavera2022/ProyectoImagenes/output_demo/test/0_mesh_'+
    ↪ maskname + '.obj')
```