graspnetAPI

Release 1.2.10

graspnet

May 17, 2021

CONTENTS:

CHAPTER

ONE

ABOUT GRASPNETAPI

GraspNet is an open project for general object grasping that is continuously enriched. Currently we release GraspNet-1Billion, a large-scale benchmark for general object grasping, as well as other related areas (e.g. 6D pose estimation, unseen object segmentation, etc.). graspnetAPI is a Python API that assists in loading, parsing and visualizing the annotations in GraspNet. Please visit graspnet website for more information on GraspNet, including for the data, paper, and tutorials. The exact format of the annotations is also described on the GraspNet website. In addition to this API, please download both the GraspNet images and annotations in order to run the demo.

1.1 Resources

- Documentations
- PDF_Documentations
- Website
- Code

1.2 License

graspnetAPI is licensed under the none commercial CC4.0 license [see https://graspnet.net/about]

CHAPTER

TWO

INSTALLATION

Note: Only Python 3 on Linux is supported.

2.1 Prerequisites

Python version under 3.6 is not tested.

2.2 Dataset

2.2.1 Download

Download the dataset at https://graspnet.net/datasets.html

2.2.2 Unzip

Unzip the files as shown in https://graspnet.net/datasets.html.

2.2.3 Rectangle Grasp Labels

Rectangle grasp labels are optional if you need labels in this format. You can both generate the labels or download the file.

If you want to generate the labels by yourself, you may refer to *Generating Rectangle Grasp Labels*.

Note: Generating rectangle grasp labels may take a long time.

After generating the labels or unzipping the labels, you need to run copy_rect_labels.py to copy rectangle grasp labels to corresponding folders.

2.2.4 Dexnet Model Cache

Dexnet model cache is optional without which the evaluation will be much slower(about 10x time slower). You can both download the file or generate it by yourself by running gen_pickle_dexmodel.py_(recommended).

2.3 Install API

You may install using pip:

```
pip install graspnetAPI
```

You can also install from source:

```
git clone https://github.com/graspnet/graspnetAPI.git
cd graspnetAPI/
pip install .
```

CHAPTER

THREE

GRASP LABEL FORMAT

3.1 Raw Label Format

The raw label is composed of two parts, i.e. labels for all grasp candidates on each object and collision masks for each scene.

3.1.1 Labels on Objects

The raw label on each object is a list of numpy arrays.

```
>>> import numpy as np
>>> l = np.load('000_labels.npz') # GRASPNET_ROOT/grasp_label/000_labels.npz
>>> l.files
['points', 'offsets', 'collision', 'scores']
>>> l['points'].shape
(3459, 3)
>>> l['offsets'].shape
(3459, 300, 12, 4, 3)
>>> l['collision'].shape
(3459, 300, 12, 4)
>>> l['collision'].dtype
dtype('bool')
>>> l['scores'].shape
(3459, 300, 12, 4)
>>> l['scores'][0][0][0][0]
-1.0
```

- 'points' records the grasp center point coordinates in model frame.
- 'offsets' records the in-plane rotation, depth and width of the gripper respectively in the last dimension.
- 'collision' records the bool mask for if the grasp pose collides with the model.
- 'scores' records the minimum coefficient of friction between the gripper and object to achieve a stable grasp.

Note: In the raw label, the **lower** score the grasp has, the **better** it is. However, -1.0 score means the grasp pose is totally not acceptable.

300, 12, 4 denote view id, in-plane rotation id and depth id respectively. The views are defined here in graspnetAPI/utils/utils.py.

```
        cloud = cloud.reshape([-1, 3])
    return cloud

def generate_views(N, phi=(np.sqrt(5)-1)/2, center=np.zeros(3, dtype=np.float32),␣
↪R=1):
    ''' Author: chenxi-wang
    View sampling on a sphere using Febonacci lattices.

    **Input:**
```

3.1.2 Collision Masks on Each Scene

Collision mask on each scene is a list of numpy arrays.

```
>>> import numpy as np
>>> c = np.load('collision_labels.npz') # GRASPNET_ROOT/collision_label/scene_0000/
↪collision_labels.npz
>>> c.files
['arr_0', 'arr_4', 'arr_5', 'arr_2', 'arr_3', 'arr_7', 'arr_1', 'arr_8', 'arr_6']
>>> c['arr_0'].shape
(487, 300, 12, 4)
>>> c['arr_0'].dtype
dtype('bool')
>>> c['arr_0'][10][20][3]
array([ True,  True,  True,  True])
```

'arr_i' is the collision mask for the *i* th object in the *object_id_list.txt* for each scene whose shape is (num_points, 300, 12, 4). num_points, 300, 12, 4 denote the number of points in the object, view id, in-plane rotation id and depth id respectively.

Users can refer to `graspnetAPI.GraspNet.loadGrasp()` for more details of how to use the labels.

3.2 API Loaded Labels

Dealing with the raw labels are time-consuming and need high familiarity with graspnet. So the API also provides an easy access to the labels.

By calling `graspnetAPI.GraspNet.loadGrasp()`, users can get all the positive grasp labels in a scene with their parameters and scores.

There are totally four kinds of data structures for loaded grasp labels: **Grasp**, **GraspGroup**, **RectGrasp** and **RectGraspGroup**. The internal data format of each class is a numpy array which is more efficient than the Python list. Their definitions are given in grasp.py

3.2.1 Example Labels

Before looking into the details, an example is given below.

Loading a GraspGroup instance.

```
__author__ = 'mhgou'
__version__ = '1.0'

from graspnetAPI import GraspNet, Grasp, GraspGroup
import open3d as o3d
import cv2
import numpy as np

# GraspNetAPI example for loading grasp for a scene.
# change the graspnet_root path

####################################################################
graspnet_root = '/disk1/graspnet' # ROOT PATH FOR GRASPNET
####################################################################

sceneId = 1
annId = 3

# initialize a GraspNet instance
g = GraspNet(graspnet_root, camera='kinect', split='train')

# load grasps of scene 1 with annotation id = 3, camera = kinect and fric_coef_thresh␣
↪= 0.2
_6d_grasp = g.loadGrasp(sceneId = sceneId, annId = annId, format = '6d', camera =
↪'kinect', fric_coef_thresh = 0.2)
print('6d grasp:\n{}'.format(_6d_grasp))

# _6d_grasp is an GraspGroup instance defined in grasp.py
print('_6d_grasp:\n{}'.format(_6d_grasp))
```

Users can access elements by index or slice.

```
# index
grasp = _6d_grasp[0]
print('_6d_grasp[0](grasp):\n{}'.format(grasp))

# slice
print('_6d_grasp[0:2]:\n{}'.format(_6d_grasp[0:2]))
print('_6d_grasp[[0,1]]:\n{}'.format(_6d_grasp[[0,1]]))
```

Each element of GraspGroup is a Grasp instance. The properties of Grasp can be accessed via provided methods.

```
# grasp is a Grasp instance defined in grasp.py
# access and set properties
print('grasp.translation={}'.format(grasp.translation))
grasp.translation = np.array([1.0, 2.0, 3.0])
print('After modification, grasp.translation={}'.format(grasp.translation))
print('grasp.rotation_matrix={}'.format(grasp.rotation_matrix))
grasp.rotation_matrix = np.eye(3).reshape((9))
print('After modification, grasp.rotation_matrix={}'.format(grasp.rotation_matrix))
print('grasp.width={}, height:{}, depth:{}, score:{}'.format(grasp.width, grasp.
↪height, grasp.depth, grasp.score))
```

(continues on next page)

(continued from previous page)

```
print('More operation on Grasp and GraspGroup can be seen in the API document')
```

RectGrasp is the class for rectangle grasps. The format is different from Grasp. But the provided APIs are similar.

```
# load rectangle grasps of scene 1 with annotation id = 3, camera = realsense and␣
↪fric_coef_thresh = 0.2
rect_grasp_group = g.loadGrasp(sceneId = sceneId, annId = annId, format = 'rect',␣
↪camera = 'realsense', fric_coef_thresh = 0.2)
print('rectangle grasp group:\n{}'.format(rect_grasp_group))

# rect_grasp is an RectGraspGroup instance defined in grasp.py
print('rect_grasp_group:\n{}'.format(rect_grasp_group))

# index
rect_grasp = rect_grasp_group[0]
print('rect_grasp_group[0](rect_grasp):\n{}'.format(rect_grasp))

# slice
print('rect_grasp_group[0:2]:\n{}'.format(rect_grasp_group[0:2]))
print('rect_grasp_group[[0,1]]:\n{}'.format(rect_grasp_group[[0,1]]))

# properties of rect_grasp
print('rect_grasp.center_point:{}, open_point:{}, height:{}, score:{}'.format(rect_
↪grasp.center_point, rect_grasp.open_point, rect_grasp.height, rect_grasp.score))
```

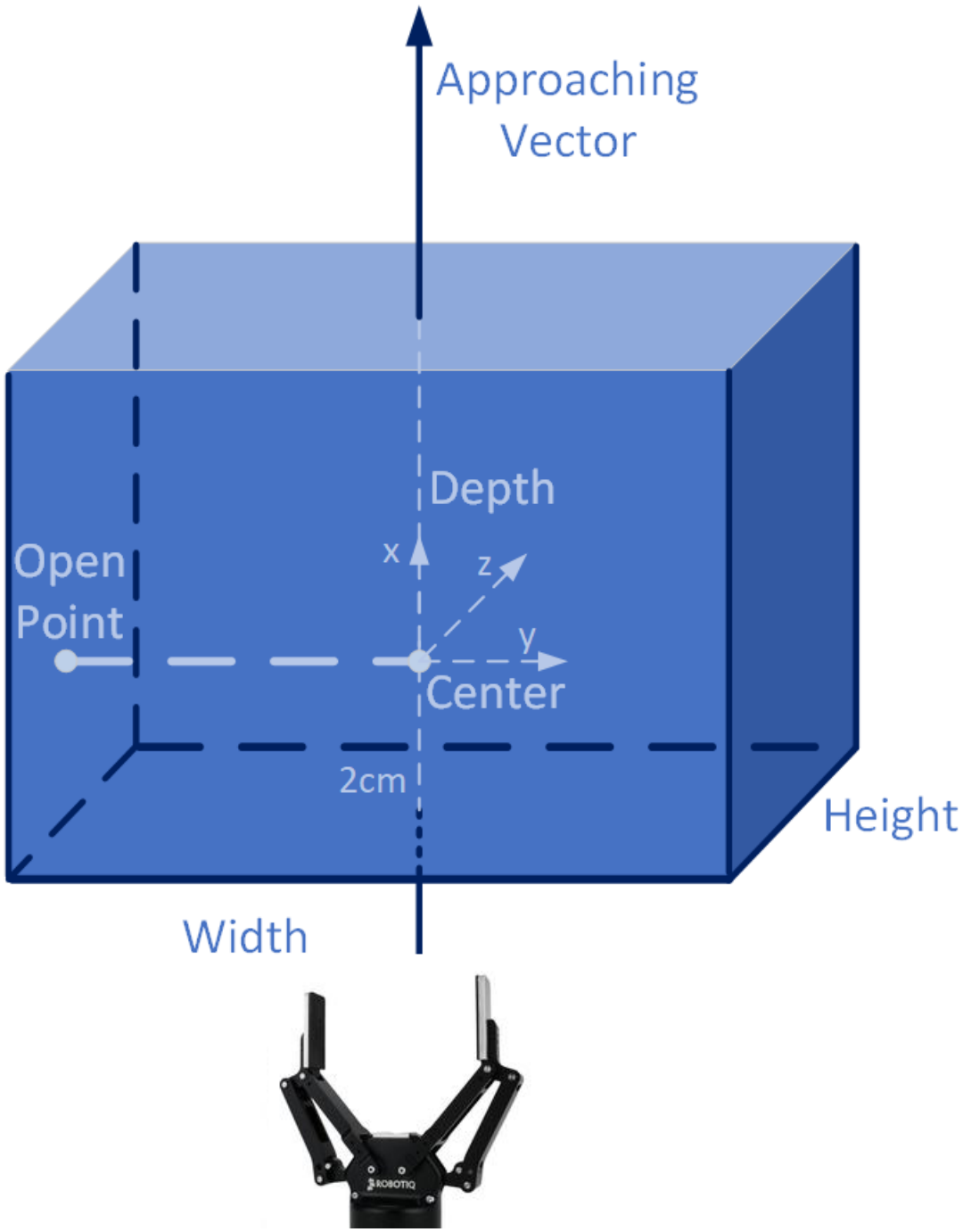
3.2.2 6D Grasp

Actually, 17 float numbers are used to define a general 6d grasp. The width, height, depth, score and attached object id are also part of the definition.

Note: In the loaded label, the **higher** score the grasp has, the **better** it is which is different from raw labels. Actually, score = 1.1 - raw_score (which is the coefficient of friction)

```
                target_points = target_points[mask1]
                target_points = transform_points(target_points, trans)
                target_points = transform_points(target_points, np.linalg.inv(camera_
↪pose))
```

The detailed defition of each parameter is shown in the figure.




```
class Grasp():
    def __init__(self, *args):
        '''
        **Input:**

        - args can be a numpy array or tuple of the score, width, height, depth,␣
↪rotation_matrix, translation, object_id

        - the format of numpy array is [score, width, height, depth, rotation_
↪matrix(9), translation(3), object_id]
```

(continues on next page)

(continued from previous page)

```
        - the length of the numpy array is 17.
        '''
        if len(args) == 0:
            self.grasp_array = np.array([0, 0.02, 0.02, 0.02, 1, 0, 0, 0, 1 ,0 , 0, 0,
↪ 1, 0, 0, 0, -1], dtype = np.float64)
        elif len(args) == 1:
            if type(args[0]) == np.ndarray:
                self.grasp_array = copy.deepcopy(args[0])
            else:
                raise TypeError('if only one arg is given, it must be np.ndarray.')
        elif len(args) == 7:
            score, width, height, depth, rotation_matrix, translation, object_id =␣
↪args
            self.grasp_array = np.concatenate([np.array((score, width, height,␣
↪depth)),rotation_matrix.reshape(-1), translation, np.array((object_id)).reshape(-
↪1)]).astype(np.float64)
        else:
            raise ValueError('only 1 or 7 arguments are accepted')
```

3.2.3 6D Grasp Group

Usually, there are a lot of grasps in a scene, `GraspGroup` is a class for these grasps. Compared with `Grasp`, `GraspGroup` contains a 2D numpy array, the additional dimension is the index for each grasp.

```
class GraspGroup():
    def __init__(self, *args):
        '''
        **Input:**

        - args can be (1) nothing (2) numpy array of grasp group array (3) str of the␣
↪npy file.
        '''
        if len(args) == 0:
            self.grasp_group_array = np.zeros((0, GRASP_ARRAY_LEN), dtype=np.float64)
        elif len(args) == 1:
            if isinstance(args[0], np.ndarray):
                self.grasp_group_array = args[0]
            elif isinstance(args[0], str):
                self.grasp_group_array = np.load(args[0])
            else:
                raise ValueError('args must be nothing, numpy array or string.')
        else:
            raise ValueError('args must be nothing, numpy array or string.')
```

Common operations on a list such as indexing, slicing and sorting are implemented. Besides, one important function is that users can **dump** a GraspGroup into a numpy file and **load** it in another program by calling `GraspGroup.save_npy()` and `GraspGroup.from_npy()`.

3.2.4 Rectangle Grasp

7 float numbers are used to define a general rectangle grasp, i.e. the center point, the open point, height, score and the attached object id. The detailed defition of each parameter is shown in the figure above and the coordinates for center point and open point are in the pixel frame.

```
class RectGrasp():
    def __init__(self, *args):
        '''
        **Input:**

        - args can be a numpy array or tuple of the center_x, center_y, open_x, open_
↪y, height, score, object_id

        - the format of numpy array is [center_x, center_y, open_x, open_y, height,␣
↪score, object_id]

        - the length of the numpy array is 7.
        '''
        if len(args) == 1:
            if type(args[0]) == np.ndarray:
                self.rect_grasp_array = copy.deepcopy(args[0])
            else:
                raise TypeError('if only one arg is given, it must be np.ndarray.')
        elif len(args) == RECT_GRASP_ARRAY_LEN:
            self.rect_grasp_array = np.array(args).astype(np.float64)
        else:
            raise ValueError('only one or six arguments are accepted')
```

3.2.5 Rectangle Grasp Group

The format for `RectGraspGroup` is similar to that of `RectGrasp` and `GraspGroup`.

```
class RectGraspGroup():
    def __init__(self, *args):
        '''
        **Input:**

        - args can be (1) nothing (2) numpy array of rect_grasp_group_array (3) str␣
↪of the numpy file.
        '''
        if len(args) == 0:
            self.rect_grasp_group_array = np.zeros((0, RECT_GRASP_ARRAY_LEN),␣
↪dtype=np.float64)
        elif len(args) == 1:
            if isinstance(args[0], np.ndarray):
                self.rect_grasp_group_array = args[0]
            elif isinstance(args[0], str):
                self.rect_grasp_group_array = np.load(args[0])
            else:
                raise ValueError('args must be nothing, numpy array or string.')
        else:
            raise ValueError('args must be nothing, numpy array or string.')
```

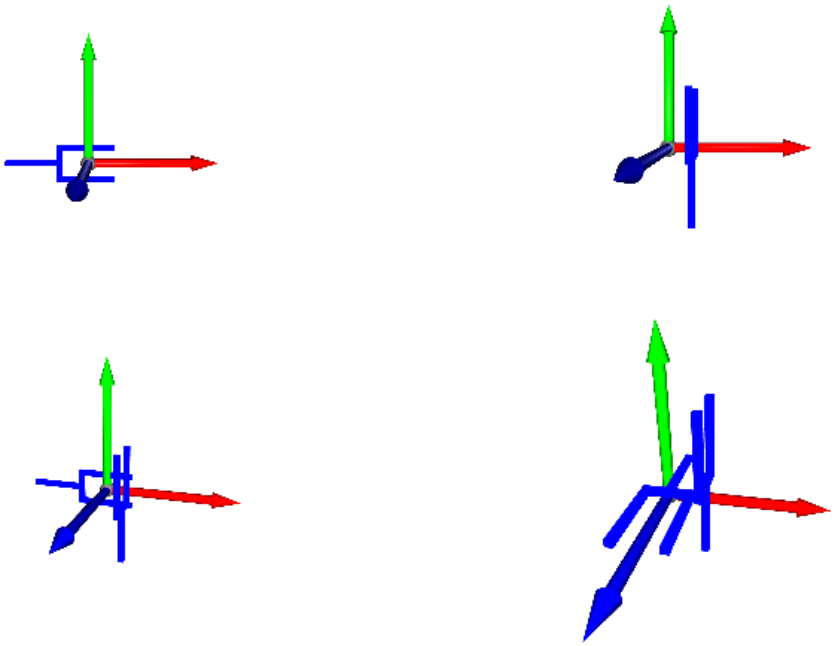
Note: We recommend users to access and modify the labels by provided functions although users can also manipulate

the data directly but it is **Not Recommended**. Please refer to the Python API for more details.

3.3 Grasp and GraspGroup Transformation

Users can transform a Grasp or GraspGroup giving a 4x4 matrix.

```
# transform grasp
g = Grasp() # simple Grasp
frame = o3d.geometry.TriangleMesh.create_coordinate_frame(0.1)

# Grasp before transformation
o3d.visualization.draw_geometries([g.to_open3d_geometry(), frame])
g.translation = np.array((0,0,0.01))

# setup a transformation matrix
T = np.eye(4)
T[:3,3] = np.array((0.01, 0.02, 0.03))
T[:3,:3] = np.array([[0,0,1.0],[1,0,0],[0,1,0]])
g.transform(T)

# Grasp after transformation
o3d.visualization.draw_geometries([g.to_open3d_geometry(), frame])

g1 = Grasp()
gg = GraspGroup()
gg.add(g)
gg.add(g1)

# GraspGroup before transformation
o3d.visualization.draw_geometries([*gg.to_open3d_geometry_list(), frame])

gg.transform(T)

# GraspGroup after transformation
o3d.visualization.draw_geometries([*gg.to_open3d_geometry_list(), frame])
```



CHAPTER

FOUR

EXAMPLES

4.1 Check Dataset Files

You can check if there is any missing file in the dataset by the following code.

```
__author__ = 'mhgou'
__version__ = '1.0'

from graspnetAPI import GraspNet

# GraspNetAPI example for checking the data completeness.
# change the graspnet_root path

if __name__ == '__main__':

    ################################################################
    graspnet_root = '/home/gmh/graspnet'  ### ROOT PATH FOR GRASPNET ###
    ################################################################

    g = GraspNet(graspnet_root, 'kinect', 'all')
    if g.checkDataCompleteness():
        print('Check for kinect passed')


    g = GraspNet(graspnet_root, 'realsense', 'all')
    if g.checkDataCompleteness():
        print('Check for realsense passed')
```

4.2 Generating Rectangle Grasp Labels

You can generate the rectangle grasp labels by yourself.

Import necessary libs:

```
# GraspNetAPI example for generating rectangle grasp from 6d grasp.
# change the graspnet_root path and NUM_PROCESS

from graspnetAPI import GraspNet
from graspnetAPI.graspnet import TOTAL_SCENE_NUM
import os
import numpy as np
from tqdm import tqdm
```

Setup how many processes to use in generating the labels.

```
########################################################################
NUM_PROCESS = 24 # change NUM_PROCESS to the number of cores to use. #
########################################################################
```

The function to generate labels.

```
def generate_scene_rectangle_grasp(sceneId, dump_folder, camera):
    g = GraspNet(graspnet_root, camera=camera, split='all')
    objIds = g.getObjIds(sceneIds = sceneId)
    grasp_labels = g.loadGraspLabels(objIds)
    collision_labels = g.loadCollisionLabels(sceneIds = sceneId)
    scene_dir = os.path.join(dump_folder,'scene_%04d' % sceneId)
    if not os.path.exists(scene_dir):
        os.mkdir(scene_dir)
    camera_dir = os.path.join(scene_dir, camera)
    if not os.path.exists(camera_dir):
        os.mkdir(camera_dir)
    for annId in tqdm(range(256), 'Scene:{}, Camera:{}'.format(sceneId, camera)):
        _6d_grasp = g.loadGrasp(sceneId = sceneId, annId = annId, format = '6d',␣
↪camera = camera, grasp_labels = grasp_labels, collision_labels = collision_labels,␣
↪fric_coef_thresh = 1.0)
        rect_grasp_group = _6d_grasp.to_rect_grasp_group(camera)
        rect_grasp_group.save_npy(os.path.join(camera_dir, '%04d.npy' % annId))
```

Run the function for each scene and camera.

```
if __name__ == '__main__':
    ########################################################################
    graspnet_root = '/home/minghao/graspnet' # ROOT PATH FOR GRASPNET ##
    ########################################################################

    dump_folder = 'rect_labels'
    if not os.path.exists(dump_folder):
        os.mkdir(dump_folder)

    if NUM_PROCESS > 1:
        from multiprocessing import Pool
        pool = Pool(24)
        for camera in ['realsense', 'kinect']:
            for sceneId in range(120):
                pool.apply_async(func = generate_scene_rectangle_grasp, args =␣
↪(sceneId, dump_folder, camera))
        pool.close()
        pool.join()

    else:
        generate_scene_rectangle_grasp(sceneId, dump_folder, camera)
```

4.3 Loading Grasp Labels

Both *6d* and *rect* format labels can be loaded.

First, import relative libs.

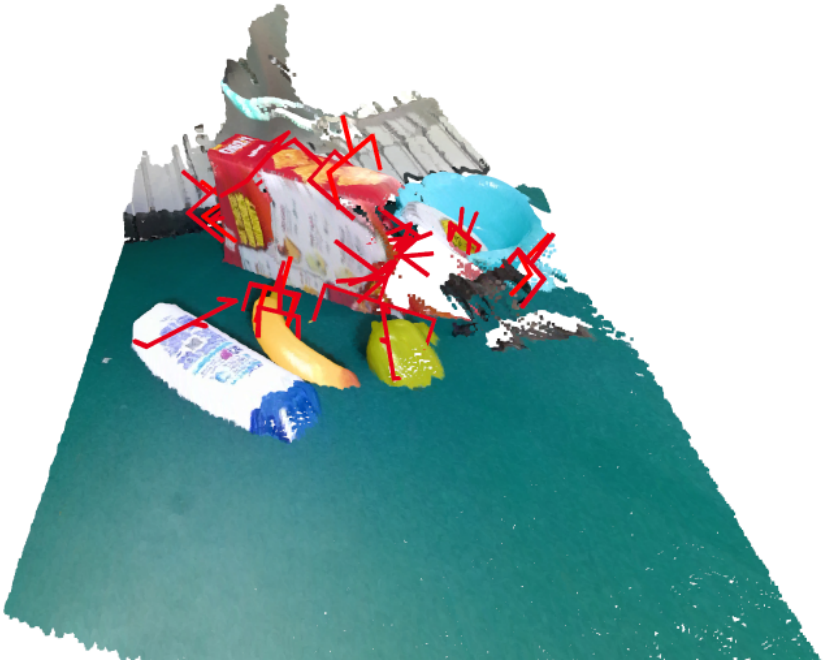
```
from graspnetAPI import GraspNet
import open3d as o3d
import cv2

```

Then, get a GraspNet instance and setup parameters.

```
############################################################################
graspnet_root = '/home/gmh/graspnet' # ROOT PATH FOR GRASPNET
############################################################################

sceneId = 1
annId = 3

# initialize a GraspNet instance
g = GraspNet(graspnet_root, camera='kinect', split='train')
```

Load GraspLabel in *6d* format and visulize the result.

```
# load grasps of scene 1 with annotation id = 3, camera = kinect and fric_coef_thresh␣
↪= 0.2
_6d_grasp = g.loadGrasp(sceneId = sceneId, annId = annId, format = '6d', camera =
↪'kinect', fric_coef_thresh = 0.2)
print('6d grasp:\n{}'.format(_6d_grasp))

# visualize the grasps using open3d
geometries = []
geometries.append(g.loadScenePointCloud(sceneId = sceneId, annId = annId, camera =
↪'kinect'))
geometries += _6d_grasp.random_sample(numGrasp = 20).to_open3d_geometry_list()
o3d.visualization.draw_geometries(geometries)
```



Load GraspLabel in *rect* format and visulize the result.

```
# load rectangle grasps of scene 1 with annotation id = 3, camera = realsense and␣
↪fric_coef_thresh = 0.2
rect_grasp = g.loadGrasp(sceneId = sceneId, annId = annId, format = 'rect', camera =
↪'realsense', fric_coef_thresh = 0.2)
print('rectangle grasp:\n{}'.format(rect_grasp))

# visualize the rectanglegrasps using opencv
bgr = g.loadBGR(sceneId = sceneId, annId = annId, camera = 'realsense')
img = rect_grasp.to_opencv_image(bgr, numGrasp = 20)
cv2.imshow('rectangle grasps', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4.4 Visualization of Dataset

Get a GraspNet instance.

```
##################################################################
graspnet_root = '/home/gmh/graspnet' # ROOT PATH FOR GRASPNET
##################################################################

from graspnetAPI import GraspNet

# initialize a GraspNet instance
g = GraspNet(graspnet_root, camera='kinect', split='train')
```

Show grasp labels on a object.

```
# show object grasps
g.showObjGrasp(objIds = 0, show=True)
```

Show 6D poses of objects in a scene.

```
# show 6d poses
g.show6DPose(sceneIds = 0, show = True)
```

Show Rectangle grasp labels in a scene.

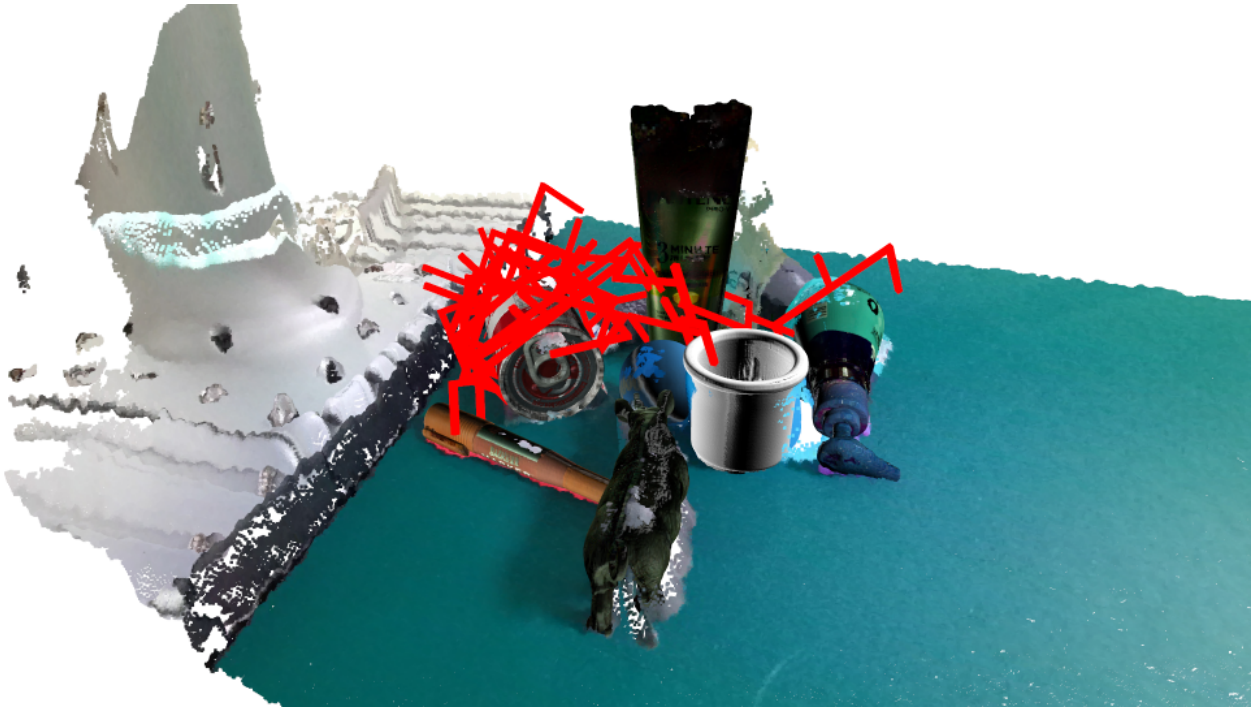
```
# show scene rectangle grasps
g.showSceneGrasp(sceneId = 0, camera = 'realsense', annId = 0, format = 'rect',␣
↪numGrasp = 20)
```

(continues on next page)

(continued from previous page)

Show 6D grasp labels in a scene.

```
# show scene 6d grasps(You may need to wait several minutes)
g.showSceneGrasp(sceneId = 4, camera = 'kinect', annId = 2, format = '6d')
```



4.5 Apply NMS on Grasps

Get a GraspNet instance.

```
# GraspNetAPI example for grasp nms.
# change the graspnet_root path

####################################################################
graspnet_root = '/home/gmh/graspnet' # ROOT PATH FOR GRASPNET
####################################################################

sceneId = 1
annId = 3

from graspnetAPI import GraspNet
import open3d as o3d
import cv2

# initialize a GraspNet instance
g = GraspNet(graspnet_root, camera='kinect', split='train')
```

Loading and visualizing grasp lables before NMS.

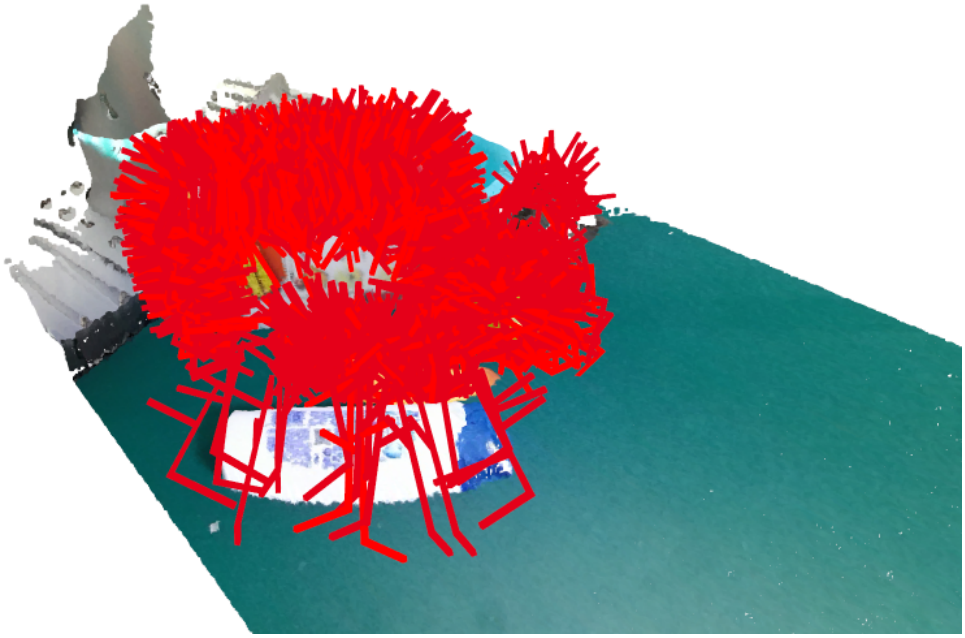
```
# load grasps of scene 1 with annotation id = 3, camera = kinect and fric_coef_thresh␣
↪= 0.2
_6d_grasp = g.loadGrasp(sceneId = sceneId, annId = annId, format = '6d', camera =
↪'kinect', fric_coef_thresh = 0.2)
print('6d grasp:\n{}'.format(_6d_grasp))

# visualize the grasps using open3d
geometries = []
geometries.append(g.loadScenePointCloud(sceneId = sceneId, annId = annId, camera =
↪'kinect'))
geometries += _6d_grasp.random_sample(numGrasp = 1000).to_open3d_geometry_list()
o3d.visualization.draw_geometries(geometries)
```

```
6d grasp:
----------
Grasp Group, Number=90332:
Grasp: score:0.9000000357627869, width:0.11247877031564713, height:0.
↪019999999552965164, depth:0.029999999329447746, translation:[-0.09166837 -0.
↪16910084  0.39480919]
rotation:
[[-0.81045675 -0.57493848  0.11227506]
 [ 0.49874267 -0.77775514 -0.38256073]
 [ 0.30727136 -0.25405255  0.91708326]]
object id:66
Grasp: score:0.9000000357627869, width:0.10030215978622437, height:0.
↪019999999552965164, depth:0.019999999552965164, translation:[-0.09166837 -0.
↪16910084  0.39480919]
rotation:
[[-0.73440629 -0.67870212  0.0033038 ]
 [ 0.64608938 -0.70059127 -0.3028869 ]
 [ 0.20788456 -0.22030747  0.95302087]]
object id:66
Grasp: score:0.9000000357627869, width:0.08487851172685623, height:0.
↪019999999552965164, depth:0.019999999552965164, translation:[-0.10412319 -0.
↪13797761  0.38312319]
```

(continues on next page)

(continued from previous page)

```
rotation:
[[ 0.03316294  0.78667939 -0.61647028]
[-0.47164679  0.55612743  0.68430364]
[ 0.88116372  0.26806271  0.38947764]]
object id:66
......
Grasp: score:0.9000000357627869, width:0.11909123510122299, height:0.
↪019999999552965164, depth:0.019999999552965164, translation:[-0.05140382  0.
↪11790846  0.48782501]
rotation:
[[-0.71453273  0.63476181 -0.2941435 ]
[-0.07400083  0.3495101   0.93400562]
[ 0.69567728  0.68914449 -0.20276351]]
object id:14
Grasp: score:0.9000000357627869, width:0.10943549126386642, height:0.
↪019999999552965164, depth:0.019999999552965164, translation:[-0.05140382  0.
↪11790846  0.48782501]
rotation:
[[ 0.08162415  0.4604325  -0.88393396]
[-0.52200603  0.77526748  0.3556262 ]
[ 0.84902728  0.4323912   0.30362913]]
object id:14
Grasp: score:0.9000000357627869, width:0.11654743552207947, height:0.
↪019999999552965164, depth:0.009999999776482582, translation:[-0.05140382  0.
↪11790846  0.48782501]
rotation:
[[-0.18380146  0.39686993 -0.89928377]
[-0.61254776  0.66926688  0.42055583]
[ 0.76876676  0.62815309  0.12008961]]
object id:14
------------
```



Apply nms to GraspGroup and visualizing the result.

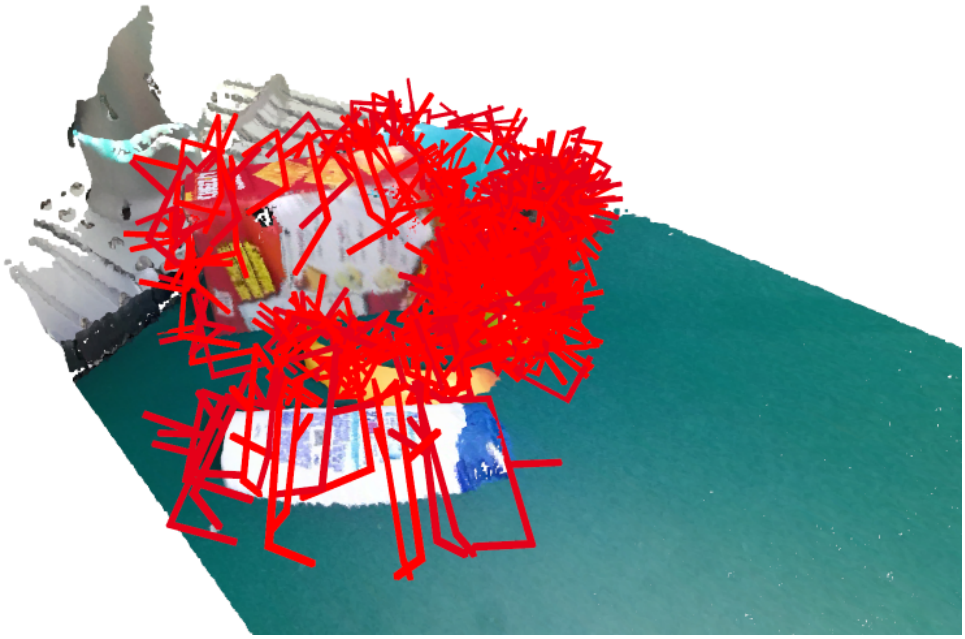
```
nms_grasp = _6d_grasp.nms(translation_thresh = 0.1, rotation_thresh = 30 / 180.0 * 3.
↪1416)
print('grasp after nms:\n{}'.format(nms_grasp))

# visualize the grasps using open3d
geometries = []
geometries.append(g.loadScenePointCloud(sceneId = sceneId, annId = annId, camera =
↪'kinect'))
geometries += nms_grasp.to_open3d_geometry_list()
o3d.visualization.draw_geometries(geometries)
```

```
grasp after nms:
----------
Grasp Group, Number=358:
Grasp: score:1.0, width:0.11948642134666443, height:0.019999999552965164, depth:0.
↪03999999910593033, translation:[-0.00363996  0.03692623  0.3311775 ]
rotation:
[[ 0.32641056 -0.8457799   0.42203382]
[-0.68102902 -0.52005678 -0.51550031]
[ 0.65548128 -0.11915252 -0.74575269]]
object id:0
Grasp: score:1.0, width:0.12185929715633392, height:0.019999999552965164, depth:0.
↪009999999776482582, translation:[-0.03486454  0.08384828  0.35117128]
rotation:
[[-0.00487804 -0.8475557   0.53068405]
[-0.27290785 -0.50941664 -0.81609803]
[ 0.96202785 -0.14880882 -0.22881967]]
object id:0
Grasp: score:1.0, width:0.04842342436313629, height:0.019999999552965164, depth:0.
↪019999999552965164, translation:[0.10816982 0.10254505 0.50272578]
rotation:
[[-0.98109186 -0.01696888 -0.19279723]
[-0.1817532   0.42313483  0.88765001]
[ 0.06651681  0.90590769 -0.41821831]]
object id:20
......
Grasp: score:0.9000000357627869, width:0.006192661356180906, height:0.
↪019999999552965164, depth:0.009999999776482582, translation:[0.0122985  0.29616502
↪0.53319722]
rotation:
[[-0.26423979  0.39734706  0.87880182]
[-0.95826042 -0.00504095 -0.28585231]
[-0.10915259 -0.91765451  0.38209397]]
object id:46
Grasp: score:0.9000000357627869, width:0.024634981527924538, height:0.
↪019999999552965164, depth:0.009999999776482582, translation:[0.11430283 0.18761221
↪0.51991153]
rotation:
[[-0.17379239 -0.96953499  0.17262182]
[-0.9434278   0.11365268 -0.31149188]
[ 0.28238329 -0.2169912  -0.93443805]]
object id:70
Grasp: score:0.9000000357627869, width:0.03459500893950462, height:0.
↪019999999552965164, depth:0.009999999776482582, translation:[0.02079188 0.11184558
↪0.50796509]
rotation:
[[ 0.38108557 -0.27480939  0.88275337]
```

(continues on next page)

(continued from previous page)

```
[-0.92043257 -0.20266907  0.33425891]
[ 0.08704928 -0.93989623 -0.33017775]]
object id:20
----------
```



4.6 Convert Labels between rectangle format and 6d format

Get a GraspNet instance.

```
from graspnetAPI import GraspNet
import cv2
import open3d as o3d

# GraspNetAPI example for checking the data completeness.
# change the graspnet_root path

camera = 'kinect'
sceneId = 5
annId = 3

################################################################
graspnet_root = '/home/gmh/graspnet' # ROOT PATH FOR GRASPNET
################################################################

g = GraspNet(graspnet_root, camera = camera, split = 'all')

bgr = g.loadBGR(sceneId = sceneId, camera = camera, annId = annId)
depth = g.loadDepth(sceneId = sceneId, camera = camera, annId = annId)
```

4.6.1 Convert rectangle format to 6d format

First, load rectangle labels from dataset.

```
# Rect to 6d
rect_grasp_group = g.loadGrasp(sceneId = sceneId, camera = camera, annId = annId,␣
↪fric_coef_thresh = 0.2, format = 'rect')
```

Convert a single RectGrasp to Grasp.

Note: This conversion may fail due to invalid depth information.

```
# RectGrasp to Grasp
rect_grasp = rect_grasp_group.random_sample(1)[0]
img = rect_grasp.to_opencv_image(bgr)

cv2.imshow('rect grasp', img)
cv2.waitKey(0)
cv2.destroyAllWindows()

grasp = rect_grasp.to_grasp(camera, depth)
if grasp is not None:
    geometry = []
    geometry.append(g.loadScenePointCloud(sceneId, camera, annId))
    geometry.append(grasp.to_open3d_geometry())
    o3d.visualization.draw_geometries(geometry)
else:
    print('No result because the depth is invalid, please try again!')
```

Before Conversion:

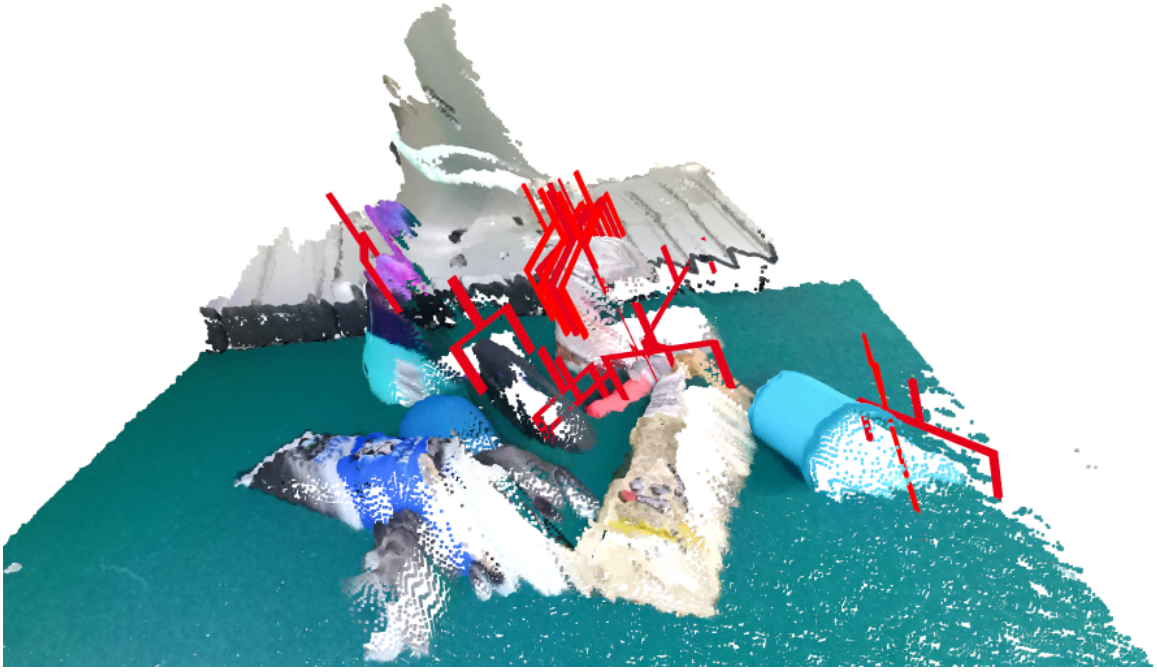
After Conversion:

Convert RectGraspGroup to GraspGroup.

```
# RectGraspGroup to GraspGroup
sample_rect_grasp_group = rect_grasp_group.random_sample(20)
img = sample_rect_grasp_group.to_opencv_image(bgr)
cv2.imshow('rect grasp', img)
cv2.waitKey(0)
cv2.destroyAllWindows()

grasp_group = sample_rect_grasp_group.to_grasp_group(camera, depth)
if grasp_group is not None:
    geometry = []
    geometry.append(g.loadScenePointCloud(sceneId, camera, annId))
    geometry += grasp_group.to_open3d_geometry_list()
    o3d.visualization.draw_geometries(geometry)
```

Before Conversion:

After Conversion:

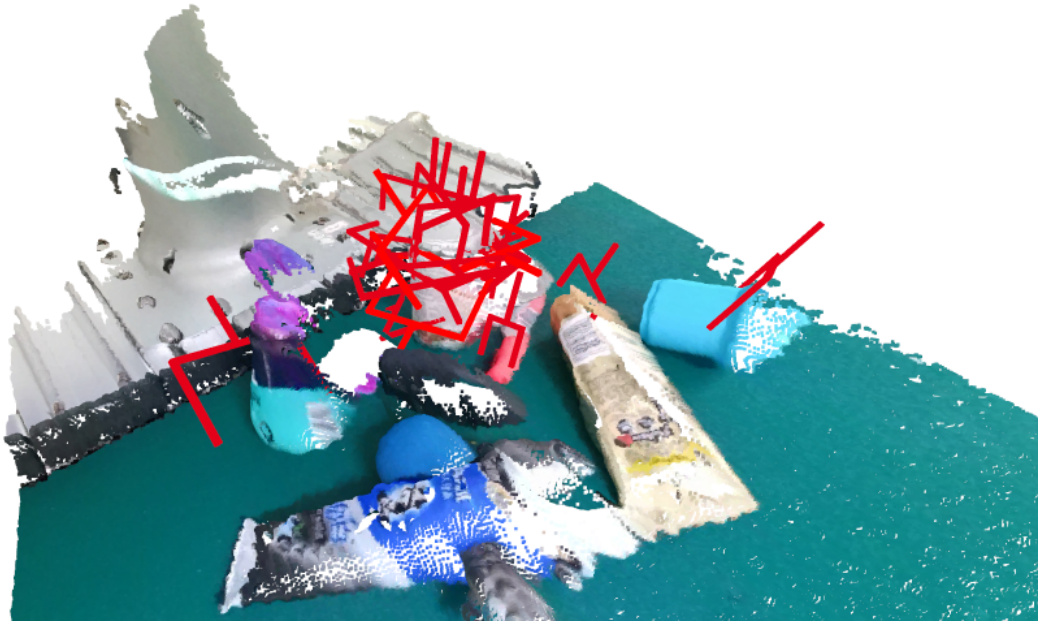


4.6.2 Convert 6d format to rectangle format

Note: Grasp to RectGrasp conversion is not applicable as only very few 6d grasp can be converted to rectangle grasp.

```
# 6d to Rect
_6d_grasp_group = g.loadGrasp(sceneId = sceneId, camera = camera, annId = annId, fric_
↪coef_thresh = 0.2, format = '6d')

# Grasp to RectGrasp conversion is not applicable as only very few 6d grasp can be␣
↪converted to rectangle grasp.

# GraspGroup to RectGraspGroup
sample_6d_grasp_group = _6d_grasp_group.random_sample(20)
geometry = []
geometry.append(g.loadScenePointCloud(sceneId, camera, annId))
geometry += sample_6d_grasp_group.to_open3d_geometry_list()
o3d.visualization.draw_geometries(geometry)

rect_grasp_group = _6d_grasp_group.to_rect_grasp_group(camera)
img = rect_grasp_group.to_opencv_image(bgr)

cv2.imshow('rect grasps', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Before Conversion:



After Conversion:

4.7 Evaluation

4.7.1 Data Preparation

The first step of evaluation is to prepare your own data. You need to run your code and generate a *GraspGroup* for each image in each scene. Then call the *save_npy* function of *GraspGroup* to dump the results.

The file structure of dump folder should be as follows:

```
|-- dump_folder
    |-- scene_0100
    |   |-- kinect
    |   |   |
    |   |   --- 0000.npy to 0255.npy
    |   |
    |   --- realsense
    |       |
    |       --- 0000.npy to 0255.npy
    |
    |-- scene_0101
    |
    ...
    |
    --- scene_0189
```

You can only generate dump for one camera, there will be no error for doing that.

4.7.2 Evaluation API

Get GraspNetEval instances.

```
# GraspNetAPI example for evaluate grasps for a scene.
# change the graspnet_root path
import numpy as np
from graspnetAPI import GraspNetEval

##################################################################
graspnet_root = '/home/gmh/graspnet' # ROOT PATH FOR GRASPNET
dump_folder = '/home/gmh/git/rgbd_graspnet/dump_affordance_iounan/' # ROOT PATH FOR␣
↪DUMP
##################################################################

sceneId = 121
camera = 'kinect'
ge_k = GraspNetEval(root = graspnet_root, camera = 'kinect', split = 'test')
ge_r = GraspNetEval(root = graspnet_root, camera = 'realsense', split = 'test')
```

Evaluate A Single Scene

```
# eval a single scene
print('Evaluating scene:{}, camera:{}'.format(sceneId, camera))
acc = ge_k.eval_scene(scene_id = sceneId, dump_folder = dump_folder)
np_acc = np.array(acc)
print('mean accuracy:{}'.format(np.mean(np_acc)))
```

Evaluate All Scenes

```
# # eval all data for kinect
# print('Evaluating kinect')
# res, ap = ge_k.eval_all(dump_folder, proc = 24)
```

Evaluate 'Seen' Split

```
# # eval 'seen' split for realsense
# print('Evaluating realsense')
# res, ap = ge_r.eval_seen(dump_folder, proc = 24)
```

CHAPTER

FIVE

PYTHON API

5.1 graspnetAPI package

5.1.1 Subpackages

graspnetAPI.utils package

Subpackages

graspnetAPI.utils.dexnet package

Subpackages

graspnetAPI.utils.dexnet.grasping package

Subpackages

graspnetAPI.utils.dexnet.grasping.meshpy package

Submodules

graspnetAPI.utils.dexnet.grasping.meshpy.mesh module

graspnetAPI.utils.dexnet.grasping.meshpy.obj_file module

graspnetAPI.utils.dexnet.grasping.meshpy.sdf module

graspnetAPI.utils.dexnet.grasping.meshpy.sdf_file module

graspnetAPI.utils.dexnet.grasping.meshpy.stable_pose module

Module contents

Submodules

graspnetAPI.utils.dexnet.grasping.contacts module

graspnetAPI.utils.dexnet.grasping.grasp module

graspnetAPI.utils.dexnet.grasping.grasp_quality_config module

graspnetAPI.utils.dexnet.grasping.grasp_quality_function module

graspnetAPI.utils.dexnet.grasping.graspable_object module

graspnetAPI.utils.dexnet.grasping.quality module

Module contents

Submodules

graspnetAPI.utils.dexnet.abstractstatic module

graspnetAPI.utils.dexnet.constants module

Module contents

Submodules

graspnetAPI.utils.config module

graspnetAPI.utils.eval_utils module

graspnetAPI.utils.pose module

graspnetAPI.utils.rotation module

graspnetAPI.utils.trans3d module

graspnetAPI.utils.utils module

graspnetAPI.utils.vis module

graspnetAPI.utils.xmlhandler module

Module contents

5.1.2 Submodules

5.1.3 graspnetAPI.grasp module

5.1.4 graspnetAPI.graspnet module

5.1.5 graspnetAPI.graspnet_eval module

5.1.6 Module contents

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search