COMP462
Project 2: Object Grasping

In this project, you will program to evaluate the quality of grasp on an object, and then optimize the grasp to improve this quality.

The object mesh model is provided and loaded by `trimesh` library in the program, with which you can easily query some essential properties of the mesh model, for example, the number of faces/vertices, the coordinates of vertices and center-of-mass, the normals of faces, face adjacencies, etc. Please refer to the library documentation (https://trimsh.org/trimesh.html) and examples (https://trimsh.org/examples.quick_start.html) for a quick start.

A grasp on an object is represented by the indices of the faces (i.e., triangles) where the contact is. For example, if a grasp is defined by `grasp=[2, 7, 36, 94]`, this means the object is contacted by this grasp at its 2-nd, 7-th, 36-th, and 94-th faces. The faces of any object provided in this project are all triangles. Therefore, you can use the centroids of the contacted triangular faces as the coordinates of the contact points of the grasp. To calculate the centroids given the indices of triangles, you can use the following function in `utils.py`:

```
def get_centroid_of_triangles(mesh, tr_ids):
    """
    Calculate the centroids of the triangles.
    args:   mesh: mesh: The object mesh model.
                  Type: trimesh.base.Trimesh
        tr_ids: The indices of the triangles on the mesh model.
                  Type: list of int
    returns: cen: The centroids of the triangles.
                  Type: numpy.ndarray of shape (len(tr_ids), 3)
    """
```

Task 1: Grasp Quality Evaluation

First, provided with an object mesh model and some random grasp, you are required to calculate the L_1 quality score of this grasp. Major steps for this task are detailed below:

1. First, please calculate to obtain the primitive wrenches for a grasp given the friction coefficient of the object surface. Specifically, you need to implement the following function in `alg.py`:

```
def primitive_wrenches(mesh, grasp, mu=0.2, n_faces=8):
    """
    Find the primitive wrenches for each contact of a grasp.
    args:  mesh: The object mesh model.
                 Type: trimesh.base.Trimesh
          grasp: The indices of the mesh triangles being contacted.
                 Type: list of int
             mu: The friction coefficient of the mesh surface.
                 (default: 0.2)
        n_faces: The number of faces of the friction polyhedral cone.
                 Type: int (default: 8)
    returns:  W: The primitive wrenches.
                 Type: numpy.ndarray of shape (len(grasp) * n_faces, 6)
    """
```

Given the friction coefficient `mu` of the object surface, you should first compute the polyhedral friction cone for each contact, which is represented by `n_faces` number of primitive forces.

Then for each primitive force $\boldsymbol{f}$, compute the corresponding torque $\boldsymbol{\tau}$ generated by this force to obtain the primitive wrench. In the end, the returned value `W` of the function should be a `numpy.ndarray`, each row of which is one primitive wrench $[\boldsymbol{f}^\top, \boldsymbol{\tau}^\top]$.

For example, given a grasp `grasp = [2, 7, 36, 94]` consisting of four contacts, your returned `W` should be of shape `(4 * n_faces, 6)`. Every `n_faces` rows of `W` are the primitive wrenches for one contact of the grasp.

2. Next, you need to implement the following function in `alg.py` to evaluate the L_1 quality of a given grasp, based on the primitive wrenches you implemented in the previous step.

```
def eval_Q(mesh, grasp, mu=0.2, n_faces=8, lmbd=0.5):
    """
    Evaluate the L1 quality of a grasp.
    args:   mesh: The object mesh model.
                  Type: trimesh.base.Trimesh
           grasp: The indices of the mesh triangles being contacted.
                  Type: list of int
              mu: The friction coefficient of the mesh surface.
                  (default: 0.2)
         n_faces: The number of faces of the friction polyhedral cone.
                  Type: int (default: 8)
            lmbd: The scale of torque magnitude.
                  (default: 0.5)
    returns:   Q: The L1 quality score of the given grasp.
    """
```

As introduced in the lecture, the $L1$ quality equals to the distance from the origin to the nearest hyperplane of the convex hull of primitive wrenches. You can import and use the `scipy.spatial` package for convex hull-related calculations.

The distance of the wrench space can be defined in different ways by scaling the torque magnitude with a parameter λ (0.5 by default in the program):

$$\|\boldsymbol{w}\| = \sqrt{\|\boldsymbol{f}\|^2 + \lambda\|\boldsymbol{\tau}\|^2}$$

Please try with different values of λ and report what you observed.

After completing this part, you can run the following command in your terminal to test your program: `python main.py --task 1`. This will visualize the mesh model and the given grasp as examplified in Figure 3, and print your results in the terminal.

Optinally, you can try with different mesh models `bunny`, `cow`, and `duck`. To do this, you can just add `--mesh the_model_you_choose` to the end of your command.

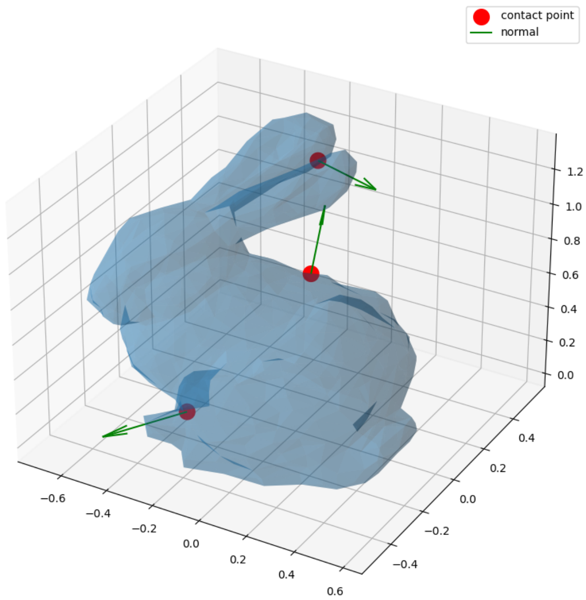

Figure 1: The visualization of an example grasp on the bunny. The red dots are the contact points of this grasp and the green arrows are the normals at the contact.

Task 2: Sample a Stable Grasp

For this part, you need to randomly sample a stable grasp on a provided mesh model, that is, the quality of this grasp is above a threshold. You can call your functions implemented in Task 1. Specifically, you need to implement the following function in `grasp.py`:

```
def sample_stable_grasp(mesh, thresh=0.0):
    """
    Sample a stable grasp such that its L1 quality is larger than a threshold.
    args:      mesh: The object mesh model.
                     Type: trimesh.base.Trimesh
             thresh: The threshold for stable grasp.
                     (default: 0.0)
    returns: grasp: The stable grasp represented by indicies of triangles.
                     Type: list of int
                 Q: The L1 quality score of the sampled grasp,
                     expected to be larger than thresh.
    """
```

After completing this part, you can invoke the following command to test your program, which visualizes the mesh model and your sampled grasp and prints your results in the terminal: `python main.py --task 2`

Task 3: Grasp Optimization

Given a mesh model and an initial grasp on it, you need to optimize the grasp and record the trajectory of your optimization. The trajectory should be a list of intermediate grasps during your optimization process. Below are the major steps needed for this task:

1. First you need to implement the following function in `alg.py` to find the η-order neighbors of any face indexed by `tr_id` on the mesh model. For this, you might be interested in accessing the `face_neighborhood` property of the mesh model to retrieve the pairs of faces that share a vertex.

```
def find_neighbors(mesh, tr_id, eta=1):
    """
    Find the eta-order neighbor faces (triangles) of tr_id on the mesh model.
    args:       mesh: The object mesh model.
                      Type: trimesh.base.Trimesh
                r_id: The index of the query face (triangle).
                      Type: int
                 eta: The maximum order of the neighbor faces:
                      Type: int
    returns: nbr_ids: The list of the indices of the neighbor faces.
                      Type: list of int
    """
```

Hint: You might be interested in accessing the `face_neighborhood` property of the mesh model to retrieve the pairs of faces that share a vertex.

2. Implement the following function in `alg.py` to find the optimal neighbor grasp of an input grasp.

```
def local_optimal(mesh, grasp):
    """
    Find the optimal neighbor grasp of the given grasp.
    args:     mesh: The object mesh model.
                    Type: trimesh.base.Trimesh
             grasp: The indices of the mesh triangles being contacted.
                    Type: list of int
    returns: G_opt: The optimal neighbor grasp with the highest quality.
                    Type: list of int
             Q_max: The L1 quality score of G_opt.
    """
```

Two grasps are considered neighbors if their corresponding contacts are neighbored faces (triangles) on the mesh model. For this step, you might want to use your `find_neighbors()` function implemented in the previous step to find neighbor faces for each contact of the grasp. And a neighbor grasp of the input grasp is just one of the combinations of the neighbor faces, .

3. Finish the following function in `alg.py` to iteratively find the optimal neighbor grasp until the grasp quality cannot be improved within its neighbors. At each iteration, remember to trace the locally optimized grasp as you need to return the whole trajectory in the end.

```
def optimize_grasp(mesh, grasp):
    """
    Optimize the given grasp and return the trajectory.
    args:    mesh: The object mesh model.
                   Type: trimesh.base.Trimesh
            grasp: The indices of the mesh triangles being contacted.
                   Type: list of int
    returns: traj: The trajectory of the grasp optimization.
                   Type: list of grasp (each grasp is a list of int)
    """
```

After completing this part, you can run the following command in your terminal to test your program, which visualizes the mesh model and your trajectory: `python main.py --task 3`

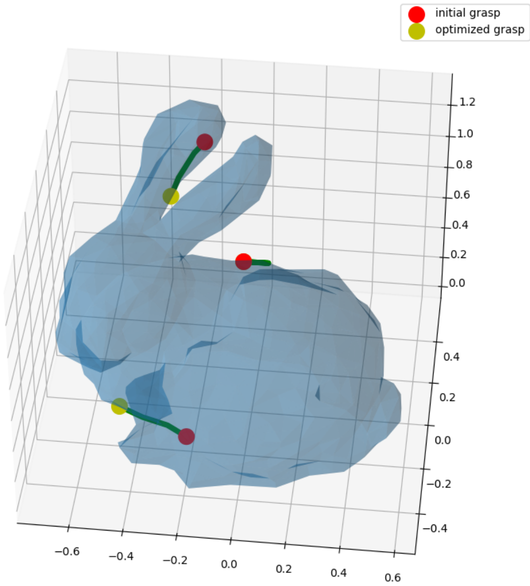

Figure 2: An example grasp optimization on the bunny. The red dots and yellow dots are the initial and optimized grasp respectively, and the green lines are the grasp trajectory during optimization.

(Optional) Task 4: Grasp Optimization with Reachability

Given a mesh model you need to first sample a grasp under the reachability constraint, and then optimize this grasp while complying with the reachability constraint. The reachability is used to constrain the contact points of a grasp not too far away from each other. Formally, this constraint is satisfied when the average of the distances from contact points to their centroid are less than a reachability measure r:

$$\frac{1}{m}\sum_{i=1}^{m}\|p_i - \psi\| < r, \quad \text{where } p1, \cdots, p_m \text{ are contact points and } \psi = \frac{1}{m}\sum_{i=1}^{m} p_i$$

Specifically, you need to implement the following function located in `grasp.py`. Feel free to implement more functions if needed. Please test with different values of r and report what you have found.

```
def optimize_reachable_grasp(mesh, r=0.5):
    """
    Sample a reachable grasp and optimize it.
    args:    mesh: The object mesh model.
                   Type: trimesh.base.Trimesh
                r: The reachability measure. (default: 0.5)
    returns: traj: The trajectory of the grasp optimization.
                   Type: list of grasp (each grasp is a list of int)
    """
```

Hint: After you sample a reachable grasp, you can use the same algorithm you implemented in Task 3 to optimize the grasp while only accepting the the reachable neighbors at each iteration.

After completing this part, you can run the following command in your terminal to test your program, which visualizes the mesh model and your trajectory, as examplified in Figure ??: `python main.py --task 4`

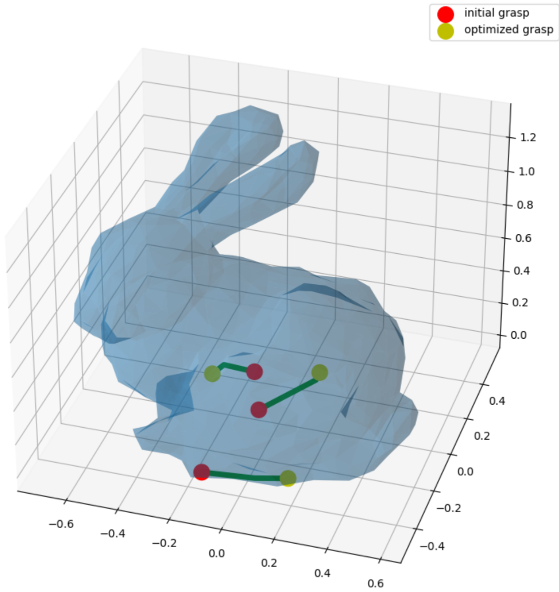

Figure 3: An example grasp optimization with reachability measure of $r = 1.0$.