Grasping
GraspIt! GPD and PointNetGPD

Lukas Rustler

What we have in our group

Barrett Hand

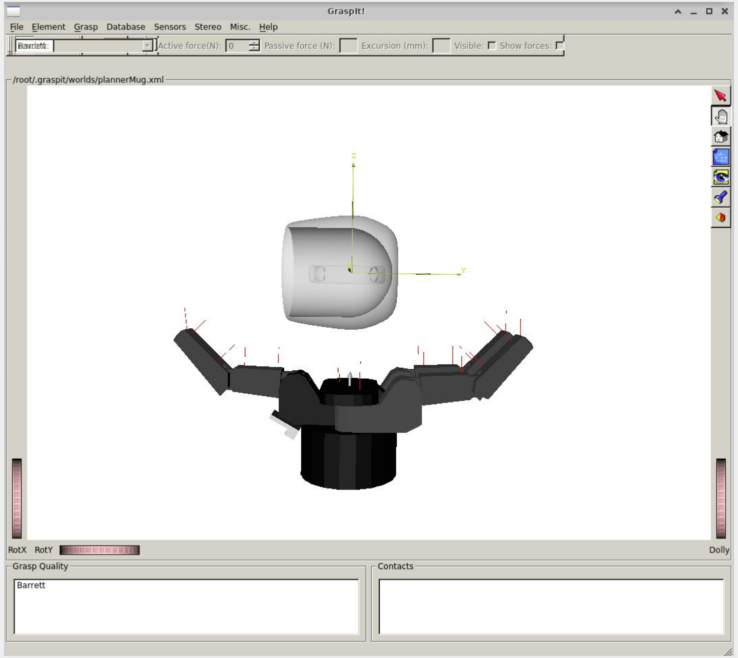
OnRobot RG6

qb SoftHand

Robotiq 2F-80

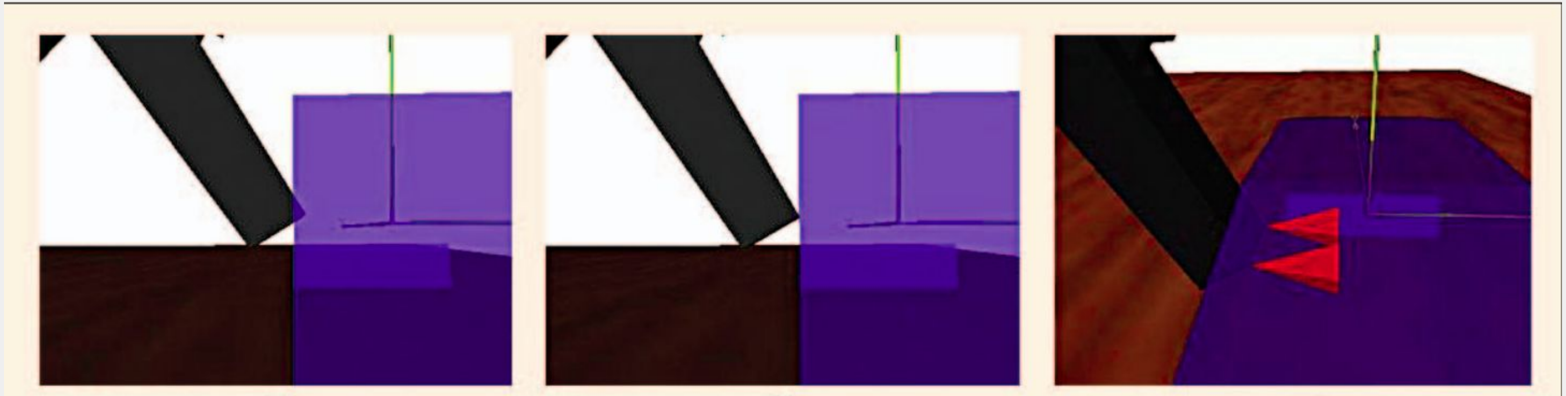
GraspIt! - Overview

- http://graspit-simulator.github.io
  - [Miller, A. T., & Allen, P. K. (2004). Graspit: A versatile simulator for robotic grasping. IEEE Robotics and Automation Magazine.]()
- Used for long time
  - For example as generator of labeled grasps
- Supports different hands or robots
  - Users can define their own
- Support obstacles
  - Importable as meshes
- Support materials
  - Different coefficients of friction
- Dynamic simulation can be enabled
  - Bullet



GraspIt! - How it works

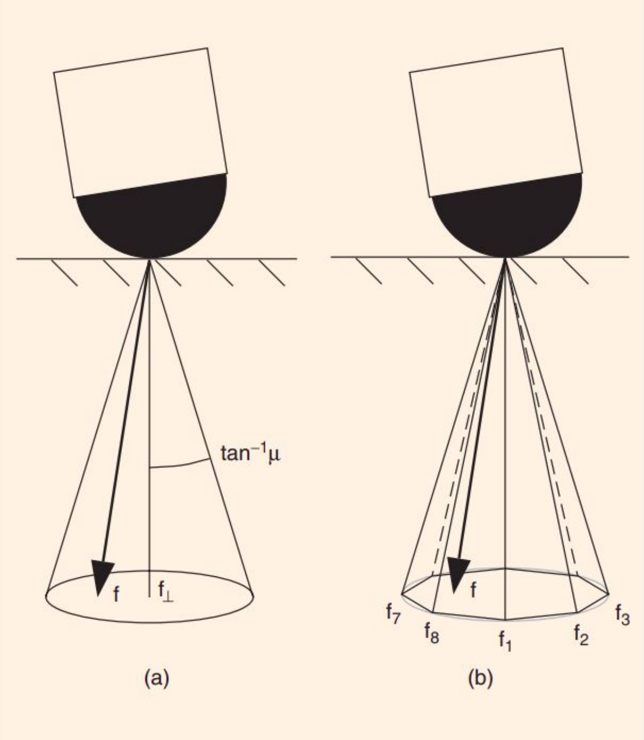
- Contact between object and gripper is detected (a)
    - Using collision detection based on trees of bounding boxes
- Joint angle which caused the collision is found and the movement is reverted before collision (b)
- Geometry of the contact is found and friction cones are created (c)



(a) (b) (c)

GraspIt! - Friction cones

- Coulomb friction model
  - Force applicable at the contact is in the friction cone
- Friction cone (a)
  - Apex in the contact point
  - Axis along the normal force $\boldsymbol{f}_\perp$
  - Half angle $tan^{-1}\mu$
    - μ is the friction coefficient
- During grasp analysis, the cone is approximated with an *m* side pyramid (b)
  - **f** is convex combination of *m* vectors



GraspIt! - Grasp Wrech Space

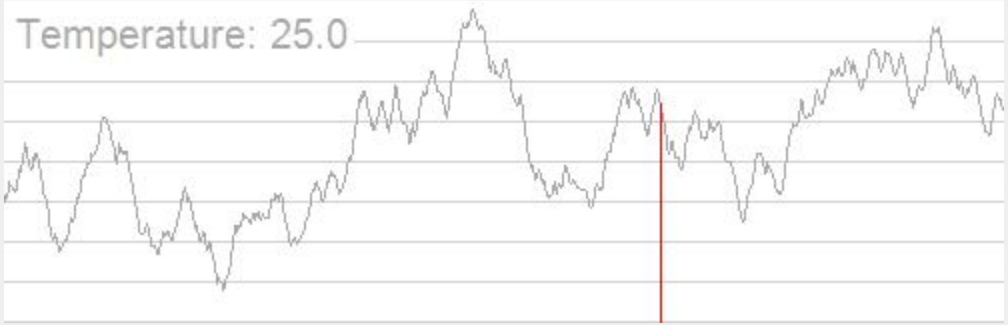
- Wrenches $$\boldsymbol{w}_{i,j} = \begin{bmatrix} \boldsymbol{f}_{i,j} \\ \lambda(\boldsymbol{d}_i \times \boldsymbol{f}_{i,j}) \end{bmatrix}$$
  - $\boldsymbol{f}_{i,j}$ one of *m* forces from the cone at contact point *i*
  - $\boldsymbol{d}_i$ vector from the torque origin
  - λ force to torque multiplicator
- GWS - space of wrenches applicable to the object given limit on normal force
  - Computed as convex hull of wrenches
- $\boldsymbol{W}_{L1} = ConvexHull\left(\bigcup_{i=1}^{n}(\boldsymbol{w}_{i,j}, \ldots, \boldsymbol{w}_{i,m})\right)$
  - Used in GraspIt!
- $\boldsymbol{W}_{L\infty} = ConvexHull\left(\bigoplus_{i=1}^{n}(\boldsymbol{w}_{i,j}, \ldots, \boldsymbol{w}_{i,m})\right)$
  - Minkowski sum
- For 3D object the GWS is 6D -> three coordinates need to be fixed for visualization

GraspIt! - Metrics

- Task wrench space
  - Space of wrenches which needs to be applied to carry out the given task
    - 6D ball when we assume that disturbances can come from any direction
- 1) Epsilon-quality
  - Radius of the biggest 6D ball in the torque origin which can fit into unit GWS
  - The closer to 1, the better quality
- 2) Volume of $\boldsymbol{W}_{L1}$
  - The bigger, the better

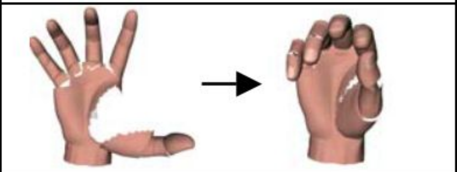
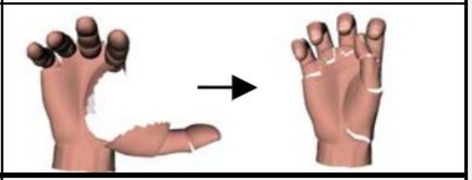
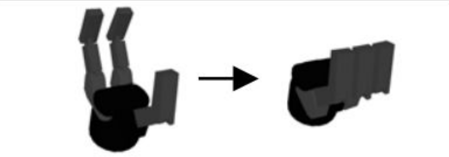
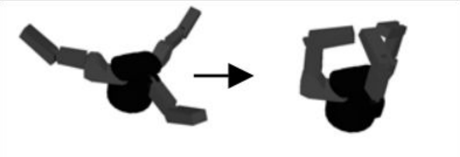
GraspIt! - Simulated Annealing

- Used to find global extrema
- Randomly computes a neighbor of current states and probabilistically decides if to change state or not
- Use parameter “Temperature T”
    - Decreases in time
    - If T = 0, it is basic hill climbing algorithm
- Used in GraspIt! to sample possible grasps



GraspIt! - Eigengrasps

- Ciocarlie *et al.* ,2007. Dimensionality reduction for hand-independent dexterous robotic grasping. IEEE International Conference on Intelligent Robots and Systems.
- Reduction of DOF of hands
  - Based on results from robotics and neuroscience
    - Majority of grasps lacks individual finger movements
- For example, human hand needs only 2 eigengrasps

Human	20	Thumb rotation Thumb flexion MCP flexion Index abduction		Thumb flexion MCP extension PIP flexion	
Barrett	4	Spread angle opening		Finger flexion	

GraspIt! - Interface

- ROS interface https://github.com/graspit-simulator/graspit_interface
  - Publishes topics and services based on GraspIt! API
- Python client https://github.com/graspit-simulator/graspit_commander
  - Access the services with Python
  - Minimal knowledge of ROS needed
    - Only datatypes - Point, Quaternion, etc.

```
In [ ]: from graspit_commander import GraspitCommander
```

```
In [ ]: GraspitCommander.clearWorld()
        GraspitCommander.importRobot("BarrettBH8_280")
        GraspitCommander.importGraspableBody("my_object.ply")
        plan = GraspitCommander.planGrasps(max_steps=70000)
```

GPD - Overview

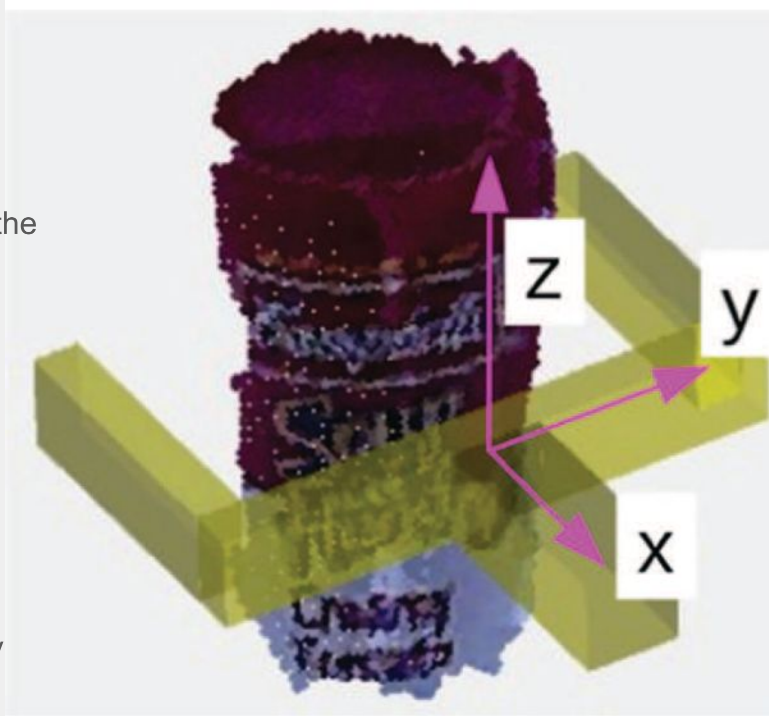
- https://github.com/atenpas/gpd
  - ten Pas *et al.*, 2017. Grasp Pose Detection in Point Clouds. International Journal of Robotics Research.
- Based on point clouds
  - even one-view
- Machine learning
- No physical properties needed
  - Materials, etc.
- Faster than GraspIt!
- Work in cluttered environment
- Assumes only two-finger grippers

GPD - Point Clouds

- Point clouds from RGB-D cameras
  - One view is sufficient
  - Basic pre-processing is needed
    - Denoising, downsampling, outliers removal
- Only information in Region of Interest (ROI) is considered
  - Segmented object,
  - or only given region in point cloud, *e.g.*, workspace

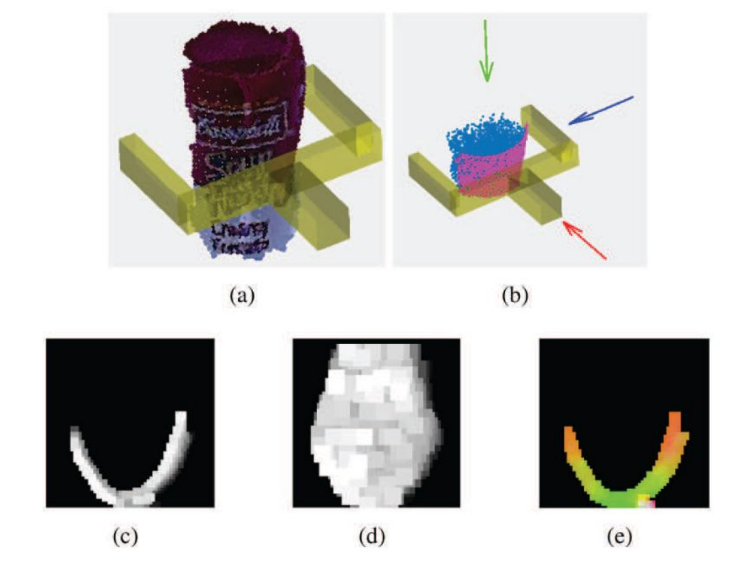
GPD - Grasps sampling

- Candidates sampled uniformly randomly over the point cloud
- Two conditions:
  - The body of the hand is not in collision with the point cloud
  - The closing region of the hand contains at least one point from the point cloud
- For each candidate, reference frame **F** of the hand is computed
- Grid search in grid $G = Y \times Z$ is performed. *Y* and *Z* contains values along *y* and *z* axis of **F**
  - Corresponding rotation and translation for each grid point are applied to the hand
- Rotated hand is pushed along negative *x* axis until contact with point cloud occurs
  - Last point before contact is added to set of possible grasp if any point from the point cloud is in the closing region of the hand



GPD - Grasp Classification

- Four-layer CNN
  - Binary classification - grasp/no grasp
- Trained from 300 thousand (sampled from 1.5 million) labeled grasp for 55 objects
- Points in closing region (b) are voxelized (MxMxM voxels)
- Input to CNN are heightmaps (c, d) of voxels projected to planes orthogonal to axes of the hand (b) and surface normals (e)

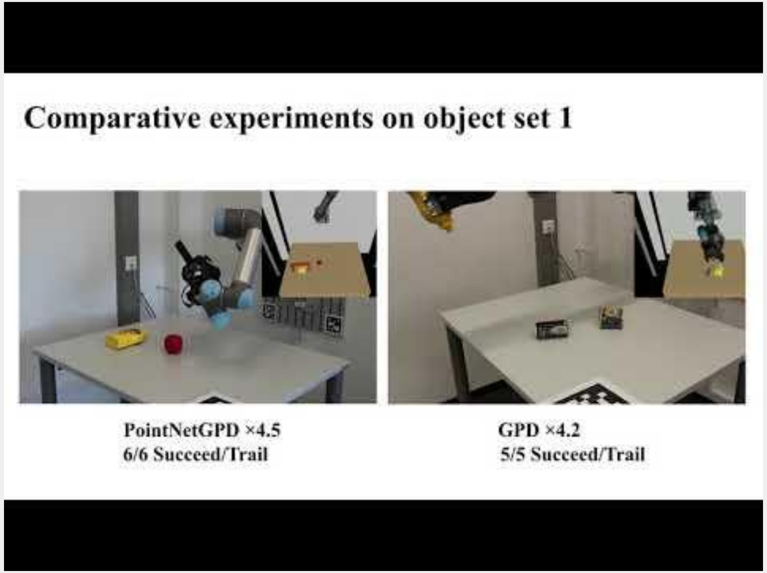


GPD - Usage

- Each model contains config file
  - We will use model trained with Eigen
  - User can set ROI, grid, set visualizations
- Individual functions can be called directly
  - Written in C++
  - Our case
- Or ROS Interface can be used

Others - PointNetGPD

- https://github.com/lianghongzhuo/PointNetGPD
    - Liang *et al.*, 2018. PointNetGPD: Detecting Grasp Configurations from Point Sets, IEEE International Conference on Robotics and Automation.
- The same grasp sampling as GPD
- Less parameters in CNN than GPD -> less prone to overfitting
- No hand-crafted features needed for training
- Works with more sparse point clouds
- Provides dataset with 350k real point clouds
- Grasp with probability, not only binary



Others - Dex-Net

- https://github.com/BerkeleyAutomation/dex-net
  - Mahler *et al.*, 2017. Dex-Net 2.0: Deep learning to plan Robust grasps with synthetic point clouds and analytic grasp metrics. Robotics: Science and Systems.
- Provides 3D datasets with evaluated grasps
  - 10 000 3D objects
- Provides Python package for manipulation with objects, grasps, *etc.*
  - Usable for testing new algorithms
- Trained Grasp-Quality CNN
  - Trained on 6.7 million point clouds