

# エージェントシステム第5講義

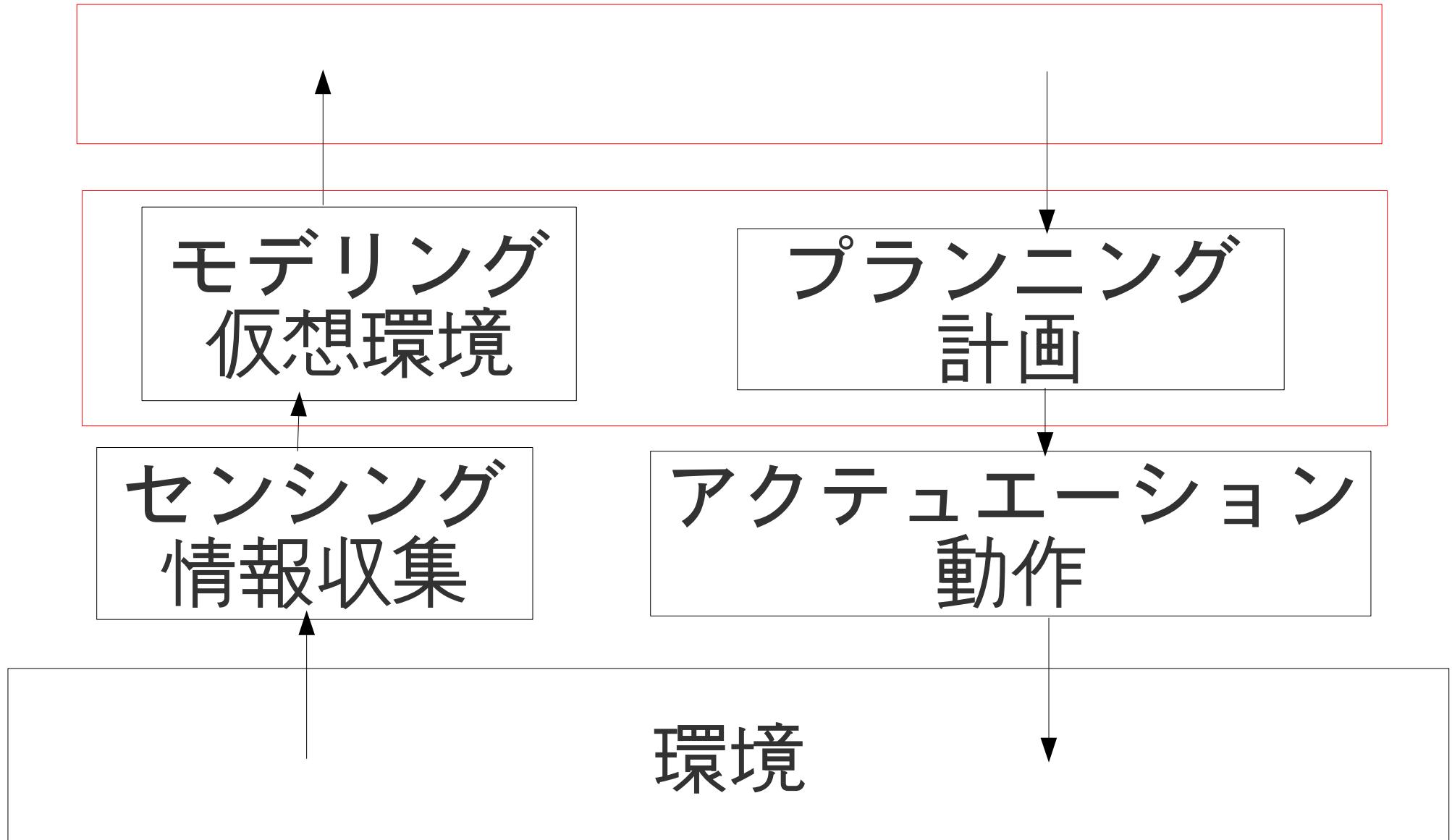
動作計画  
OpenRAVE

2011-05-25  
出杏光 魯仙  
であんこう ろせん

# 内容

- 動作計画基礎
- 知的マニピュレーション
  - 自動化プロセスの概念
  - 動作計画用の知識ベース
  - 動作計画アルゴリズム
- OpenRAVE

# エージェントシステムの基本像



# 計画

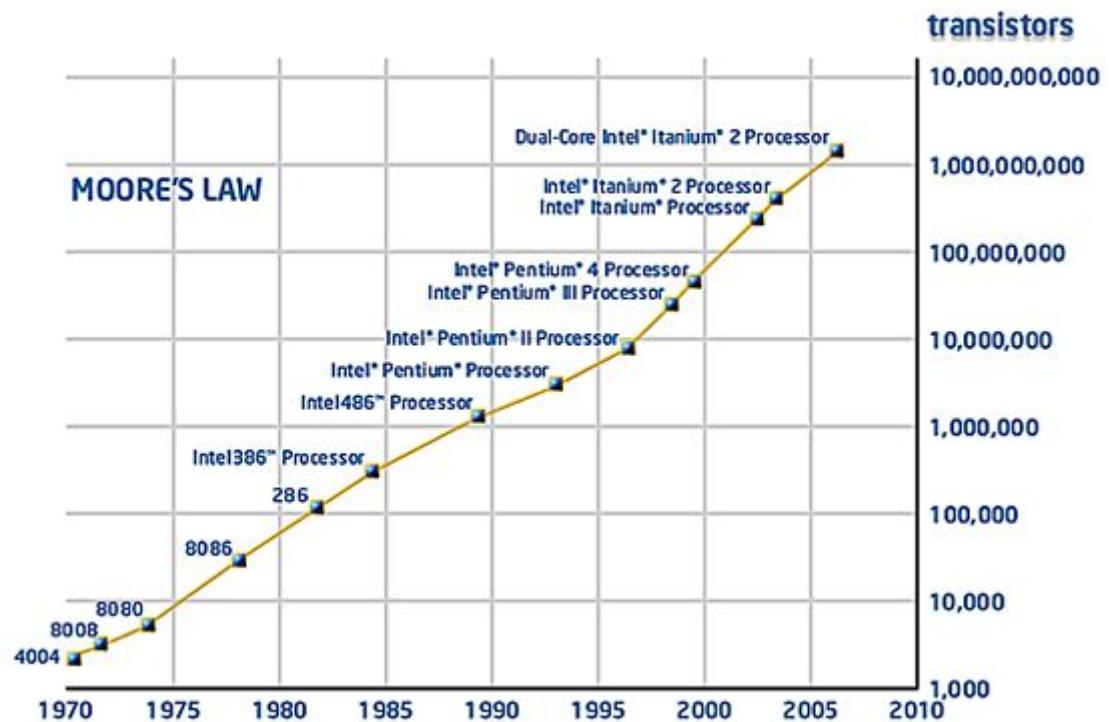


# 探索法の力

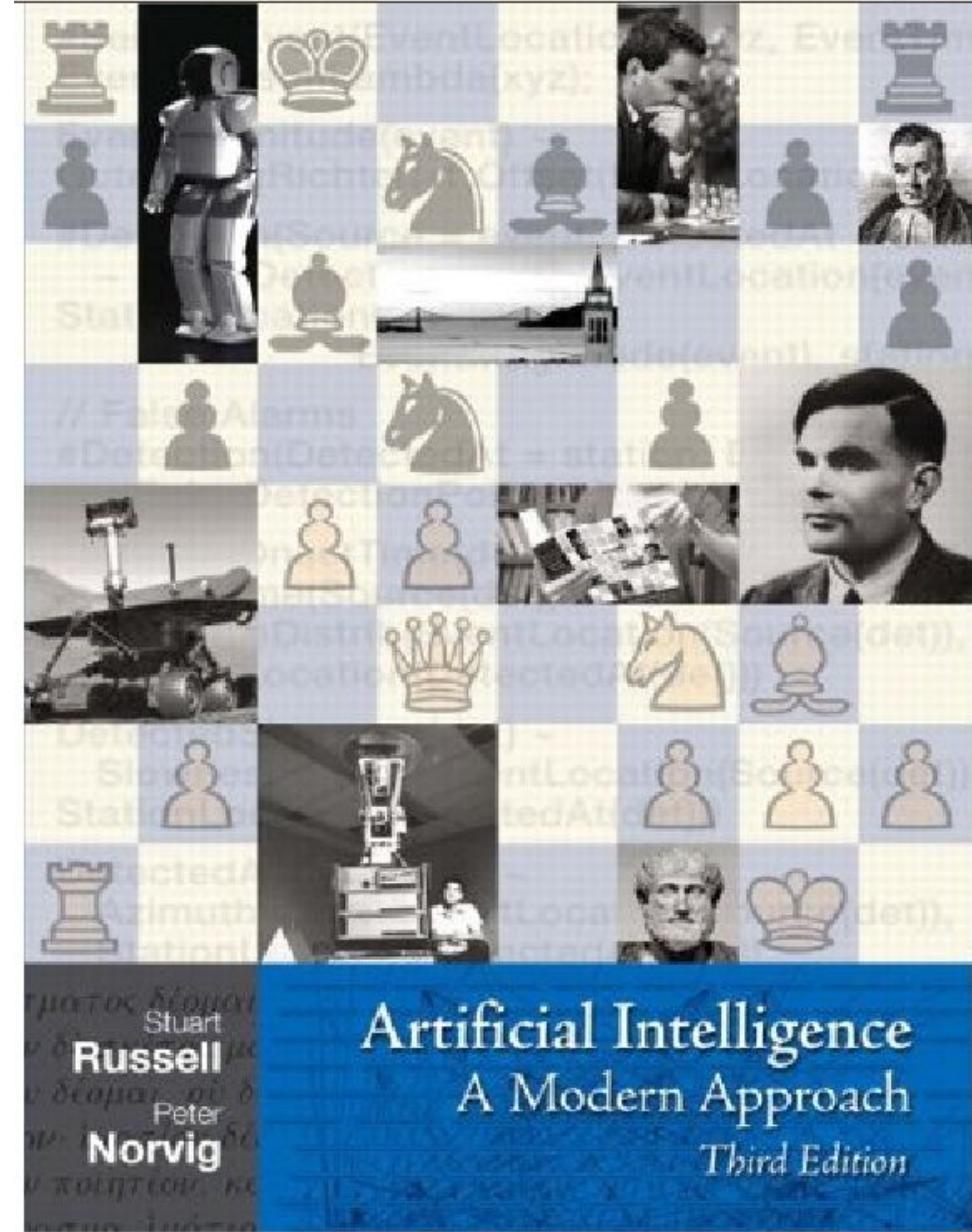
- 人工知能の基本
- 数多くの選択肢の考慮
- 毎年速くなっている



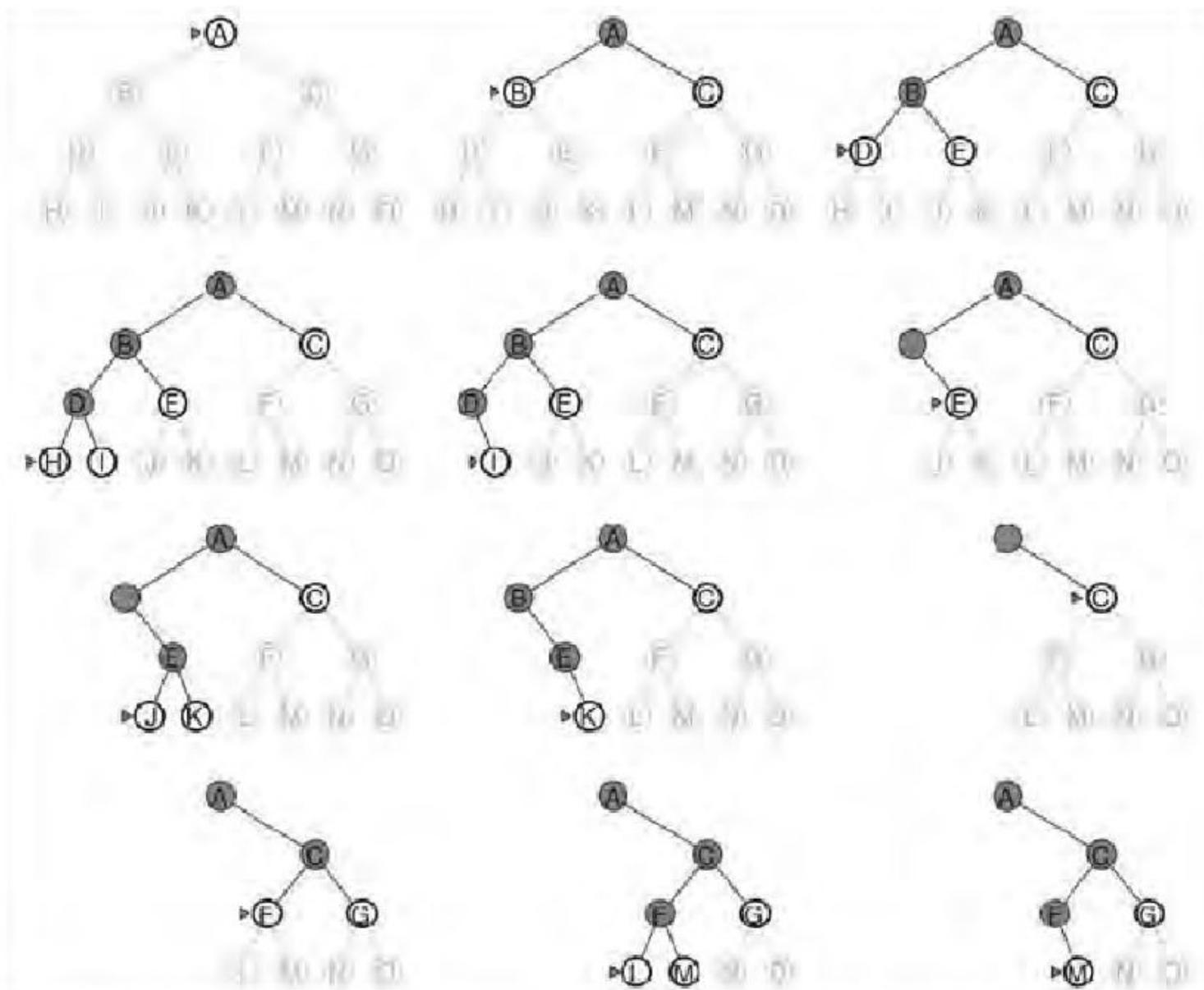
Gordon Moore, Intel Co-Founder



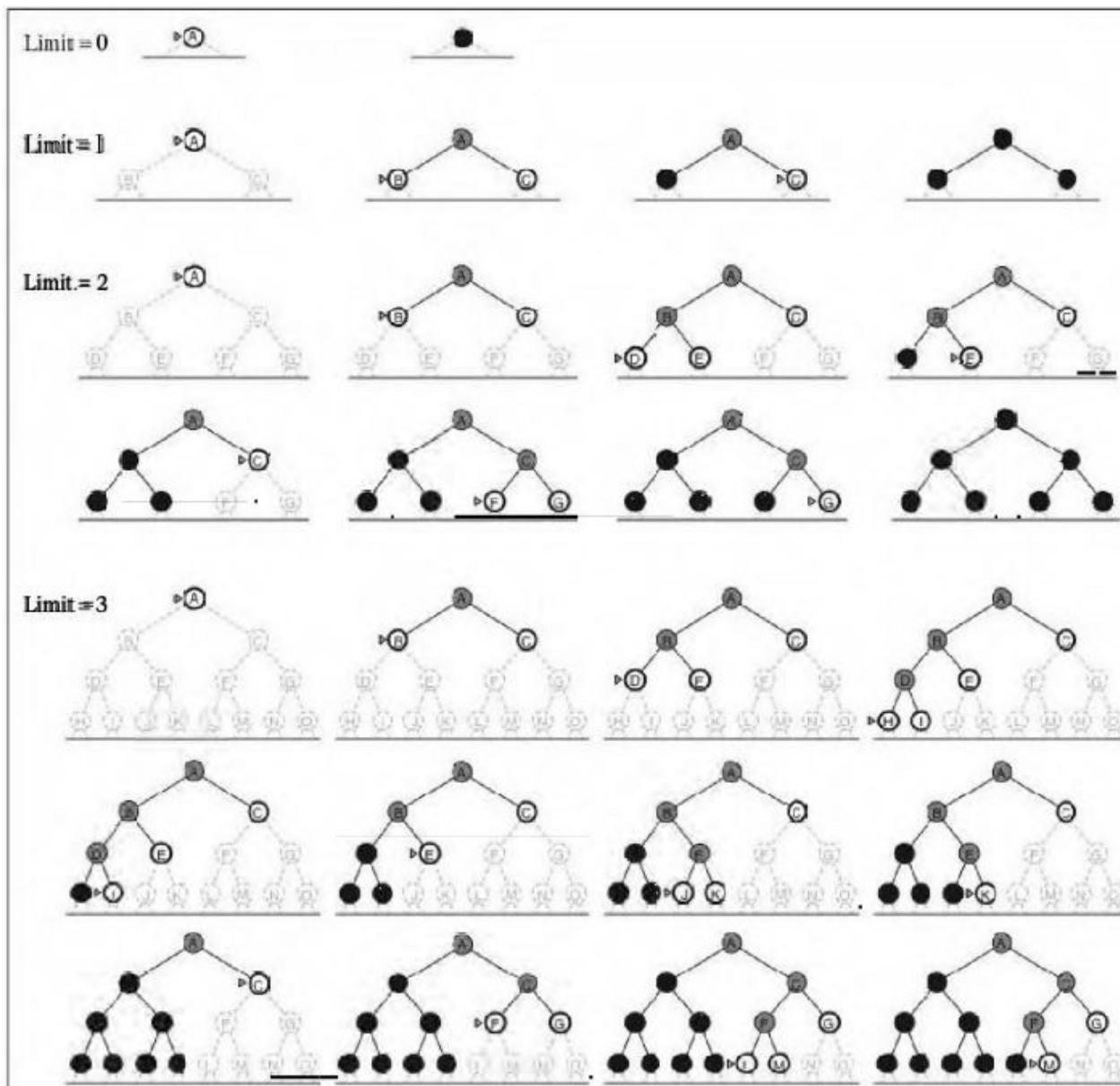
# 一般計画 アルゴリズム



# Depth First Search (DFS)



# Iterative Deepening DFS



# A\*探索：最短の距離

$$f(n) = g(n) + h(n)$$

- $f(n)$  - nを通る最短距離のコスト
- $g(n)$  - 初期値からのnまでのコスト
- $h(n)$  - nから目的地までのコスト

条件 :  $h(x) \leq d(x, y) + h(y)$



# 一般的な前進探索アルゴリズム

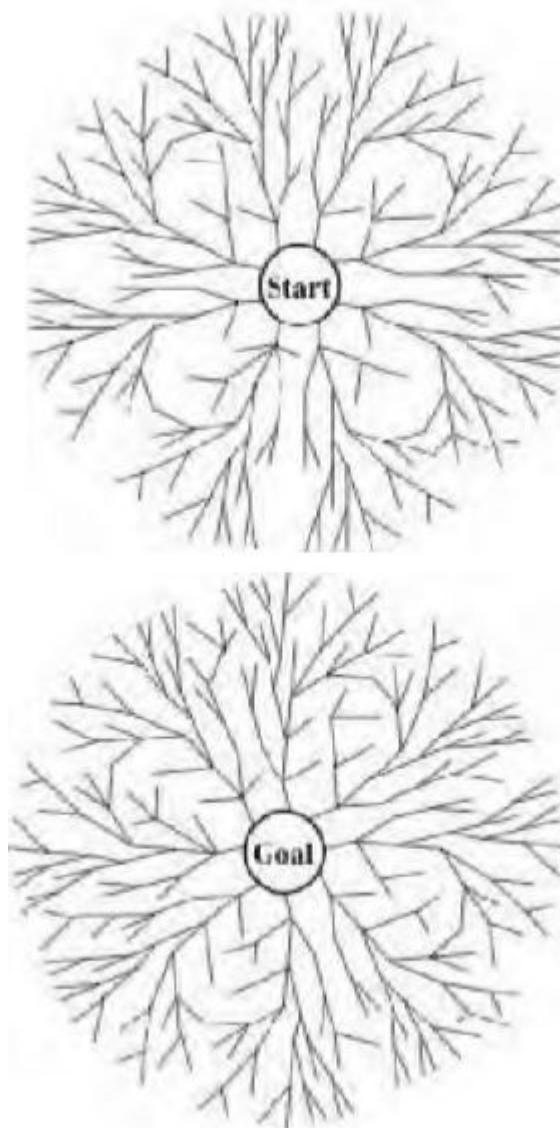
---

FORWARD\_SEARCH

```
1   Q.Insert( $x_I$ ) and mark  $x_I$  as visited
2   while  $Q$  not empty do
3        $x \leftarrow Q.GetFirst()$ 
4       if  $x \in X_G$ 
5           return SUCCESS
6       forall  $u \in U(x)$ 
7            $x' \leftarrow f(x, u)$ 
8           if  $x'$  not visited
9               Mark  $x'$  as visited
10              Q.Insert( $x'$ )
11          else
12              Resolve duplicate  $x'$ 
13      return FAILURE
```

Qが整頓されている

# 両進探索



## BIDIRECTIONAL SEARCH

```
1    $Q_I.Insert(x_I)$  and mark  $x_I$  as visited
2    $Q_G.Insert(x_G)$  and mark  $x_G$  as visited
3   while  $Q_I$  not empty and  $Q_G$  not empty do
4       if  $Q_I$  not empty
5            $x \leftarrow Q_I.GetFirst()$ 
6           if  $x = x_G$  or  $x \in Q_G$ 
7               return SUCCESS
8           forall  $u \in U(x)$ 
9                $x' \leftarrow f(x, u)$ 
10              if  $x'$  not visited
11                  Mark  $x'$  as visited
12                   $Q_I.Insert(x')$ 
13              else
14                  Resolve duplicate  $x'$ 
15              if  $Q_G$  not empty
16                   $x' \leftarrow Q_G.GetFirst()$ 
17                  if  $x' = x_I$  or  $x' \in Q_I$ 
18                      return SUCCESS
19                  forall  $u^{-1} \in U^{-1}(x')$ 
20                       $x \leftarrow f^{-1}(x', u^{-1})$ 
21                      if  $x$  not visited
22                          Mark  $x$  as visited
23                           $Q_G.Insert(x)$ 
24                      else
25                          Resolve duplicate  $x$ 
26      return FAILURE
```

# エージェント用探索アルゴリズム

- 後進探索の重要性
  - 一歩進んで計画どおりに行かなければ再計画の速さ
- D\* Lite (火星探索ロボット利用)
  - $g(s)$  = 目的地から  $s$ へのコスト
  - $v(s)$  =  $s$ からのベストコスト
  - 状態の優先  $k(s) = [k_1(s), k_2(s)]$   
 $= [\min(v(s), g(s)) + h(s_{start}, s), \min(v(s), g(s))]]$

Anthony Stentz, Dave Ferguson, Likachev  
カーネギーメロン大学

## ComputePath()

```
5 while(key( $s_{start}$ ) > min $_{s \in OPEN}(\text{key}(s))$  OR  $v(s_{start}) < g(s_{start})$ )
6   remove  $s$  with the smallest key( $s$ ) from  $OPEN$ ;
7   if ( $v(s) > g(s)$ )
8      $v(s) = g(s)$ ;
9   for each predecessor  $s'$  of  $s$ 
10    if  $s'$  was never visited before
11       $v(s') = g(s') = \infty$ ;  $bp(s') = \text{null}$ ;
12      if ( $g(s') > c(s', s) + v(s)$ )
13         $bp(s') = s$ ;
14         $g(s') = c(s', bp(s')) + v(bp(s'))$ ; UpdateSetMembership( $s'$ );
15    else
16       $v(s) = \infty$ ; UpdateSetMembership( $s$ );
17      for each predecessor  $s'$  of  $s$ 
18        if  $s'$  was never visited before
19           $v(s') = g(s') = \infty$ ;  $bp(s') = \text{null}$ ;
20          if ( $bp(s') = s$ )
21             $bp(s') = \text{argmin}_{s'' \in Succ(s')} c(s', s'') + v(s'')$ ;
22             $g(s') = c(s', bp(s')) + v(bp(s'))$ ; UpdateSetMembership( $s'$ );
```

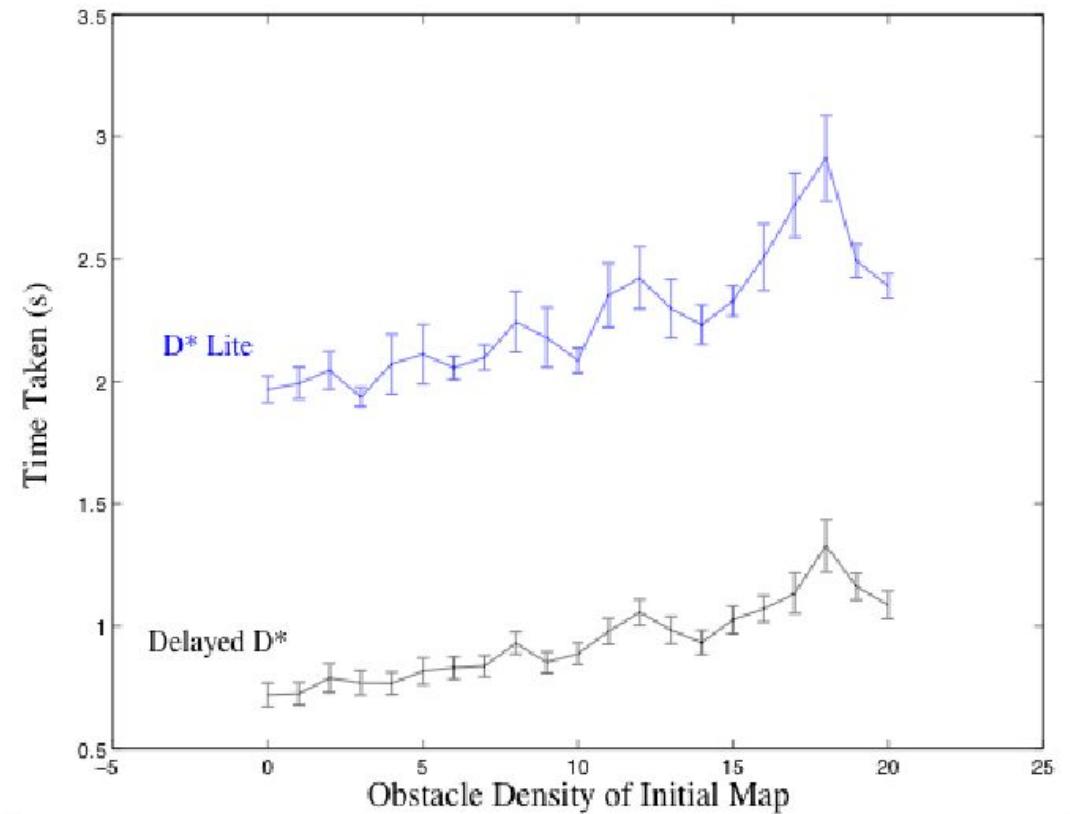
# D\* Lite

### UpdateSetMembership( $s$ )

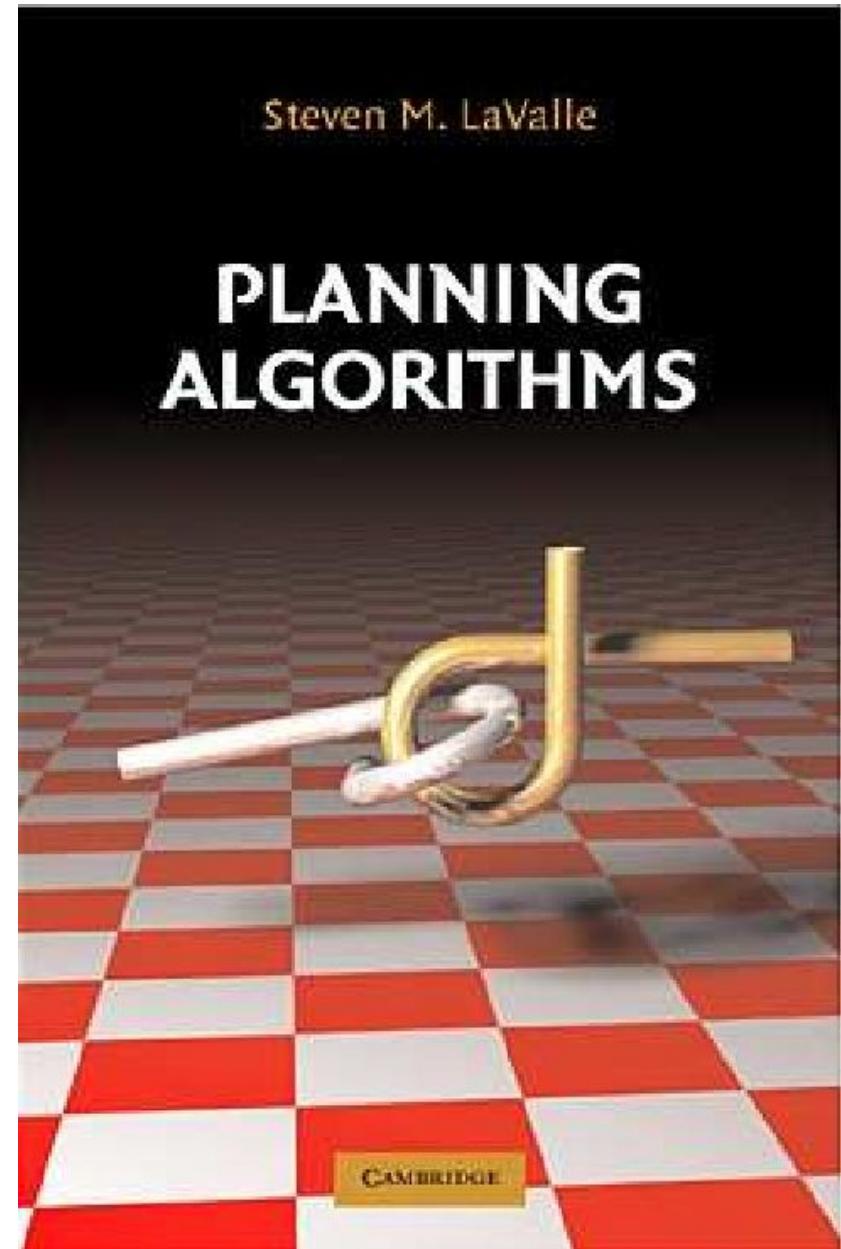
```
1  if ( $v(s) \neq g(s)$ )
2    insert/update  $s$  in  $OPEN$  with key( $s$ );
3  else
4    if ( $s \in OPEN$ ) remove  $s$  from  $OPEN$ ;
```

# Delayed D\*

- 状態を調べる際
  - 経路に影響を与えないものを省く

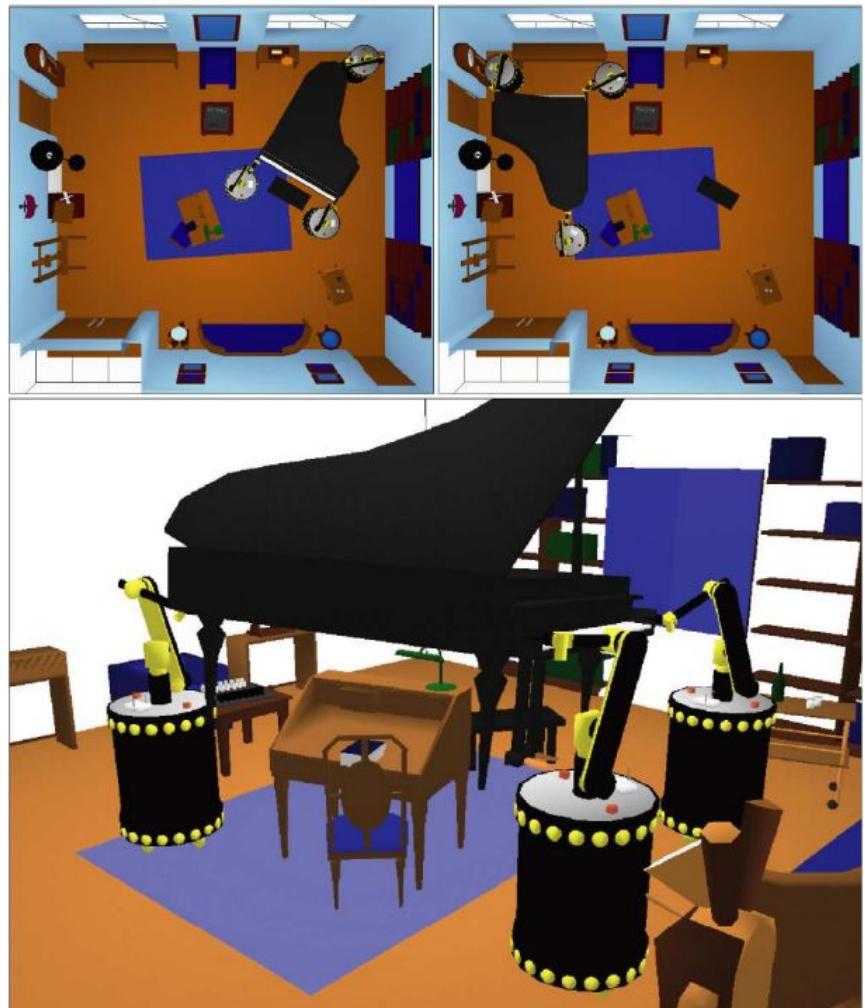


# 動作計画 アルゴリズム



# Configuration Space コンフィグレーションスペース 状態空間

- 3D位置
- 3D姿勢
- 関節の値



Steven LaValle. Planning Algorithms.  
Cambridge University, 2006

# 状態空間 : C-Space

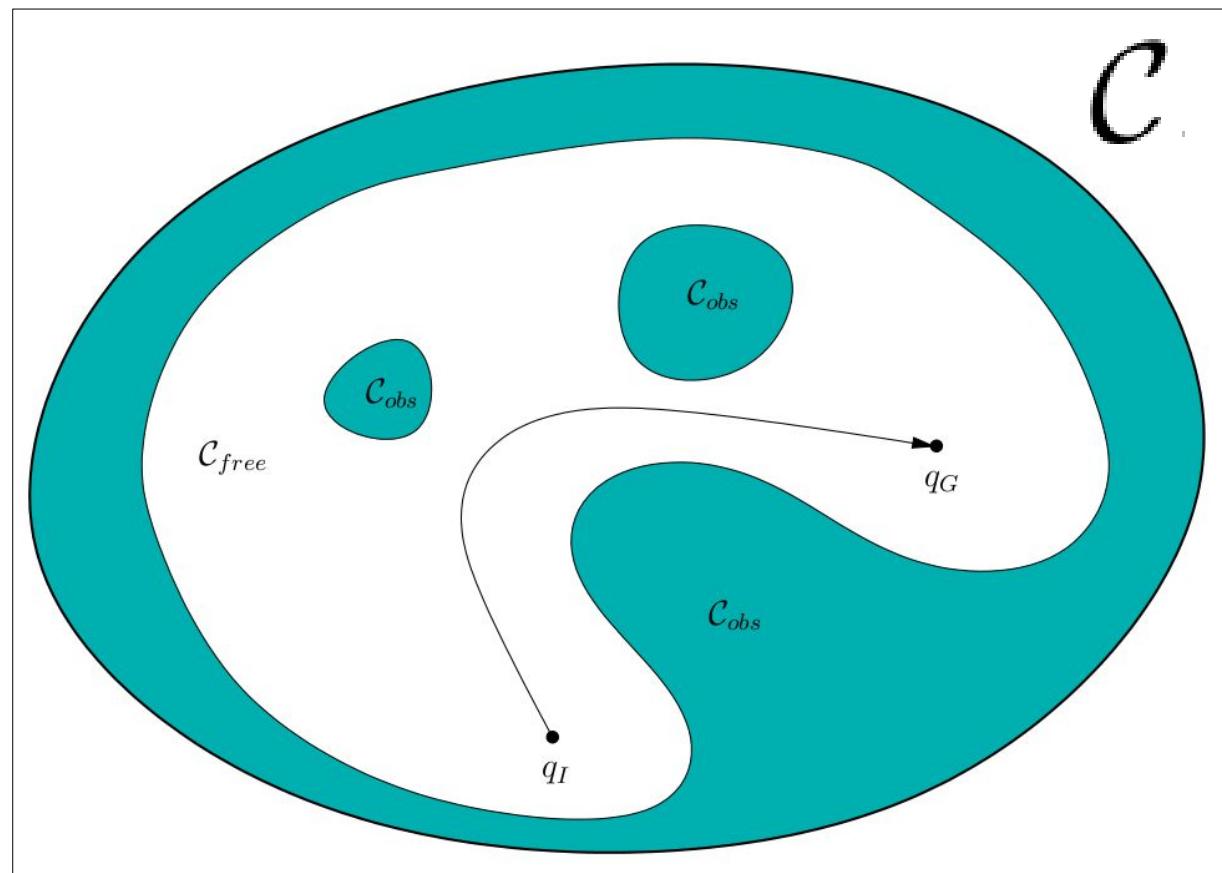
$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}$$

ロボット領域

障害物

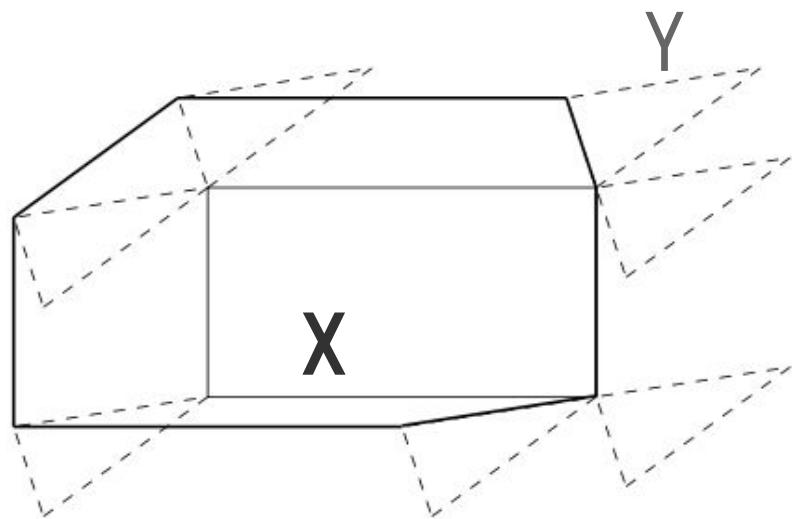
$$\mathcal{C}_{obs} \subseteq \mathcal{C}$$

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$$

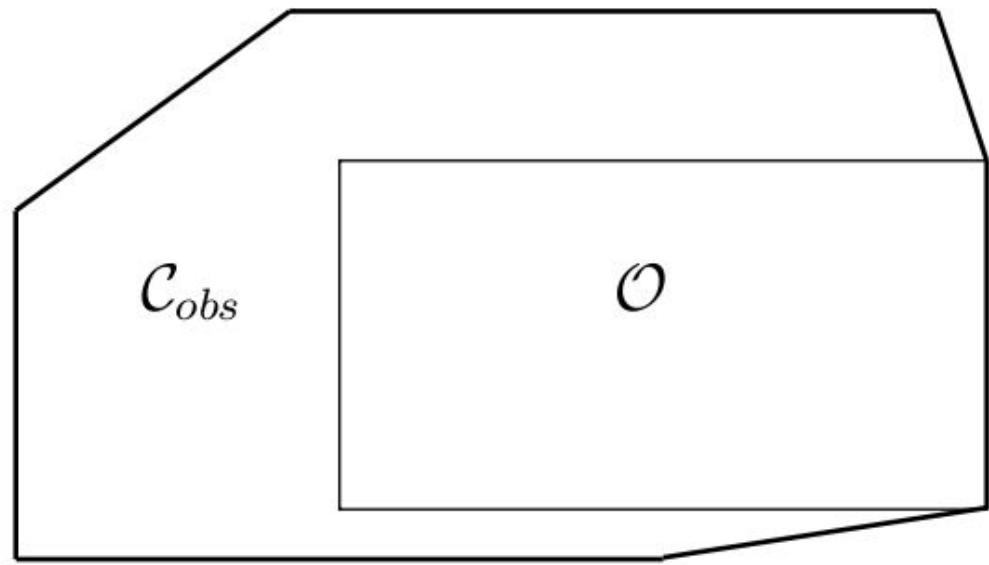


# Minkowski引算

$$X \ominus Y = \{x - y \in \mathbb{R}^n \mid x \in X \text{ and } y \in Y\}$$



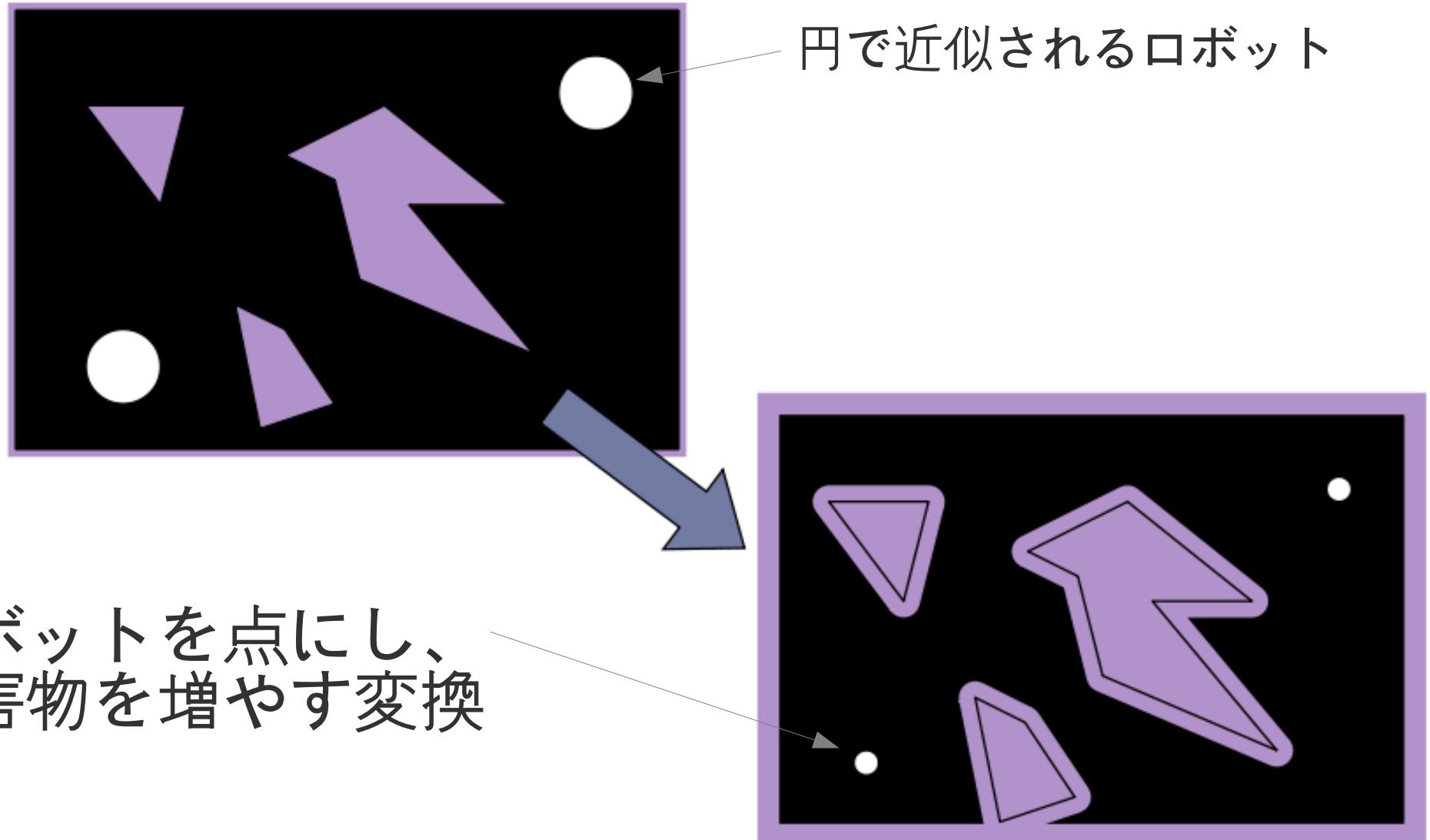
(a)



(b)

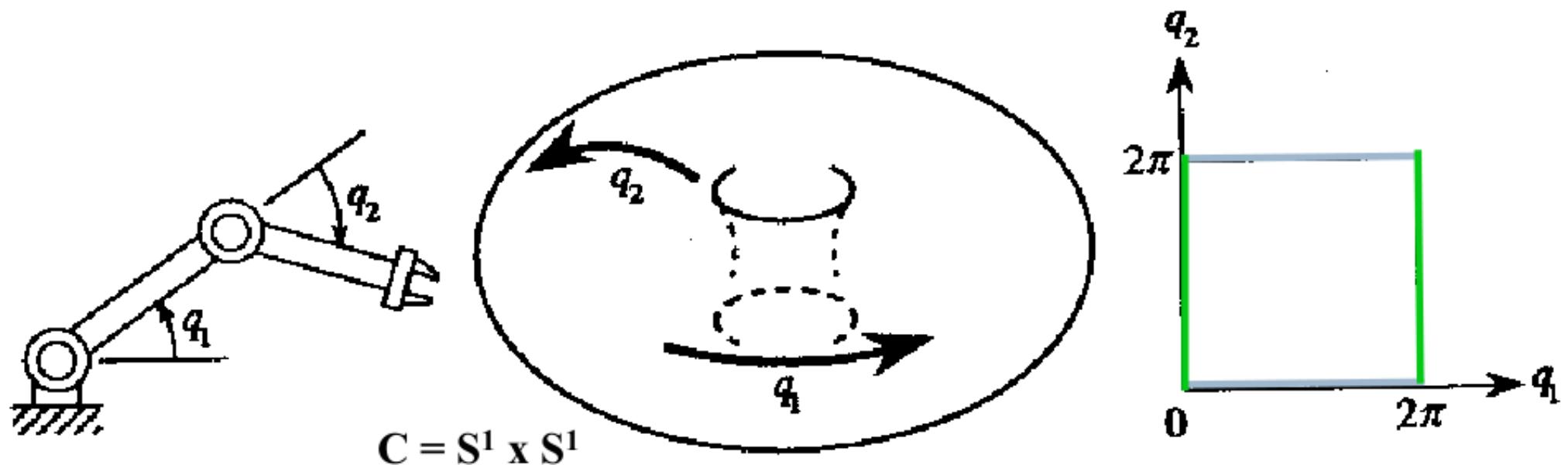
$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\} \longrightarrow \mathcal{C}_{obs} = \mathcal{O} \ominus \mathcal{A}(0)$$

# 2Dの状態空間



# 2関節ロボットの状態空間

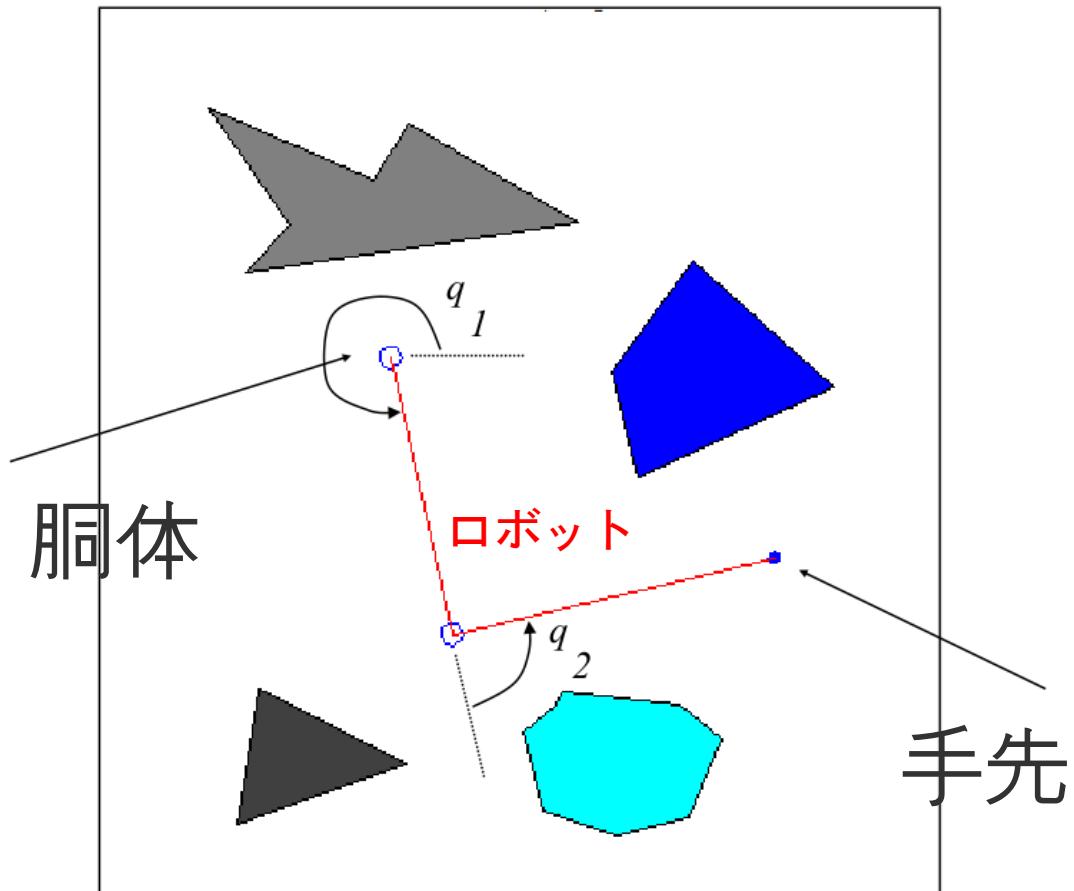
- 二つの関節は円環面 (torus) の表面を描く



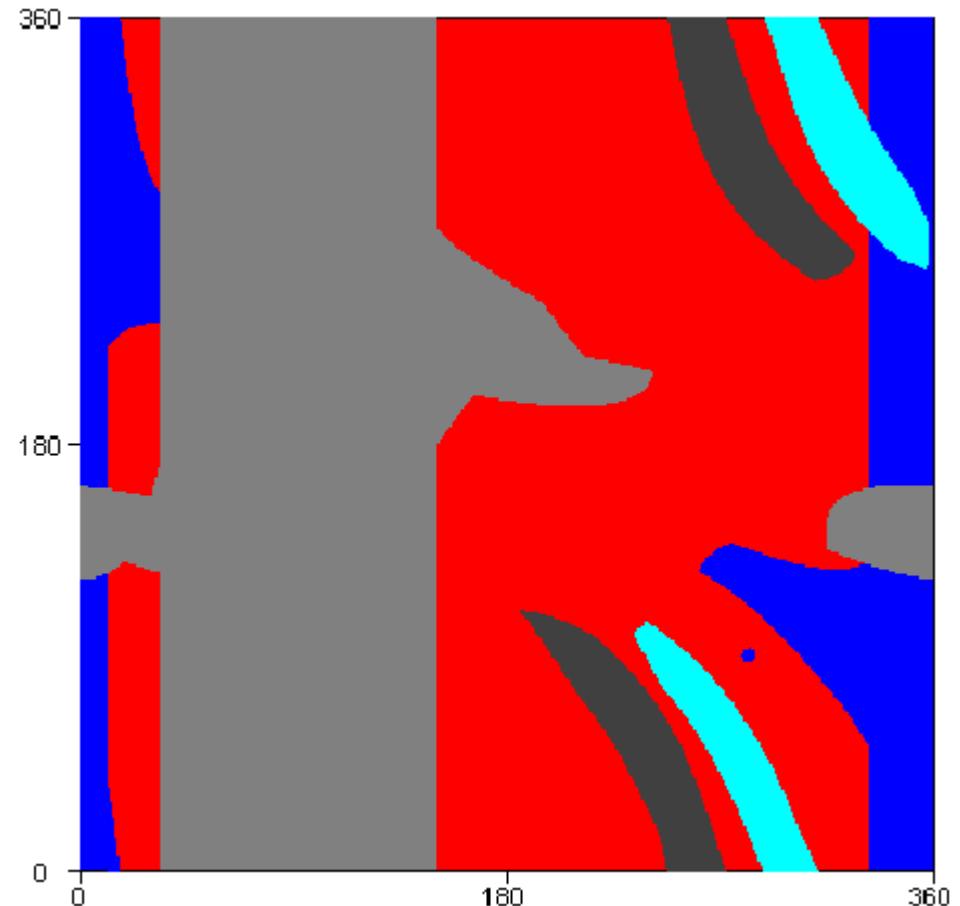
# 2D作業空間、2関節ロボット

<http://ford.ieor.berkeley.edu/cspace/>

## 作業空間

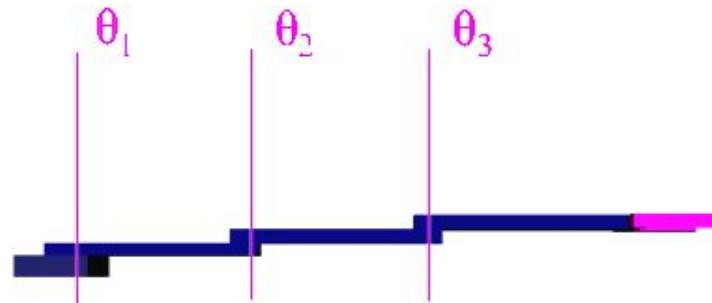


## 状態空間

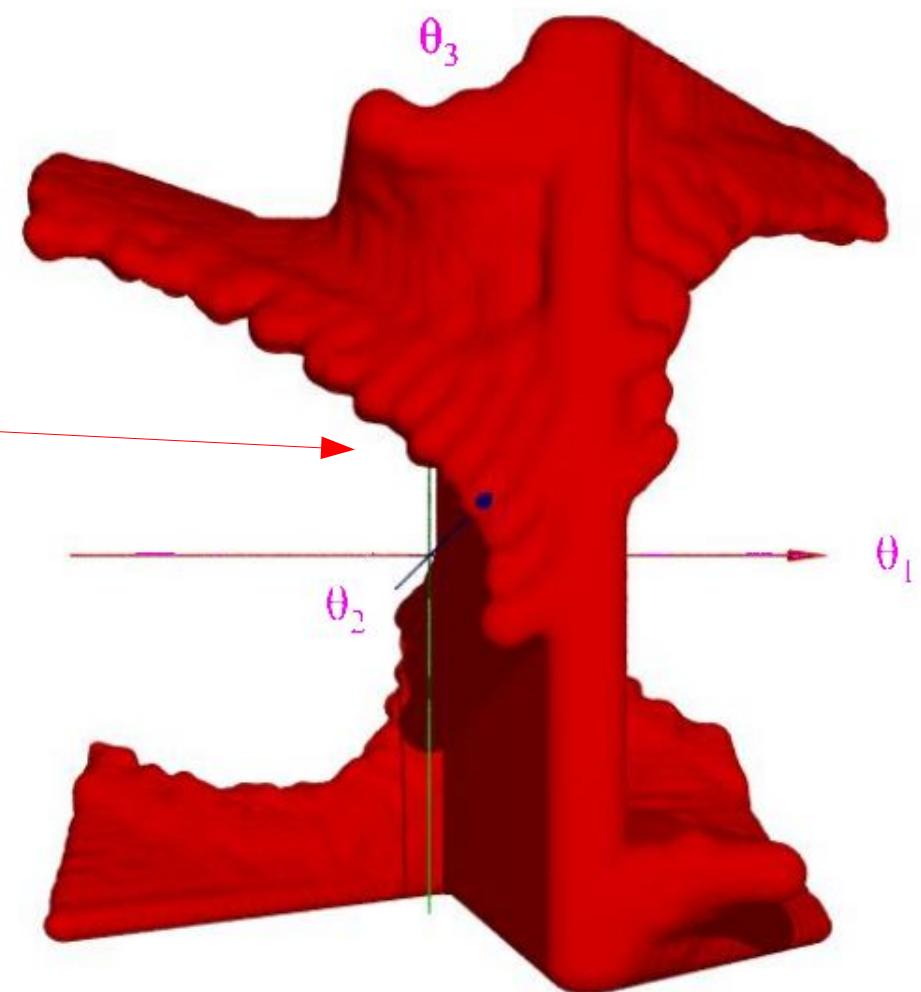


# 2D作業空間、3関節ロボット

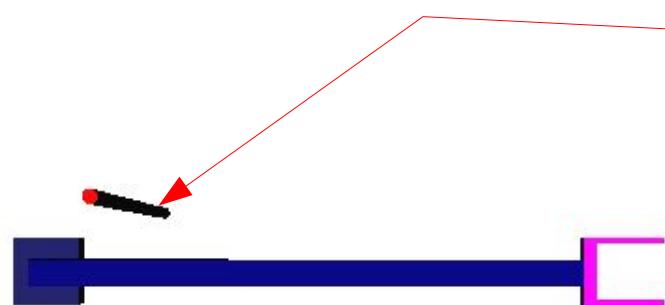
作業空間



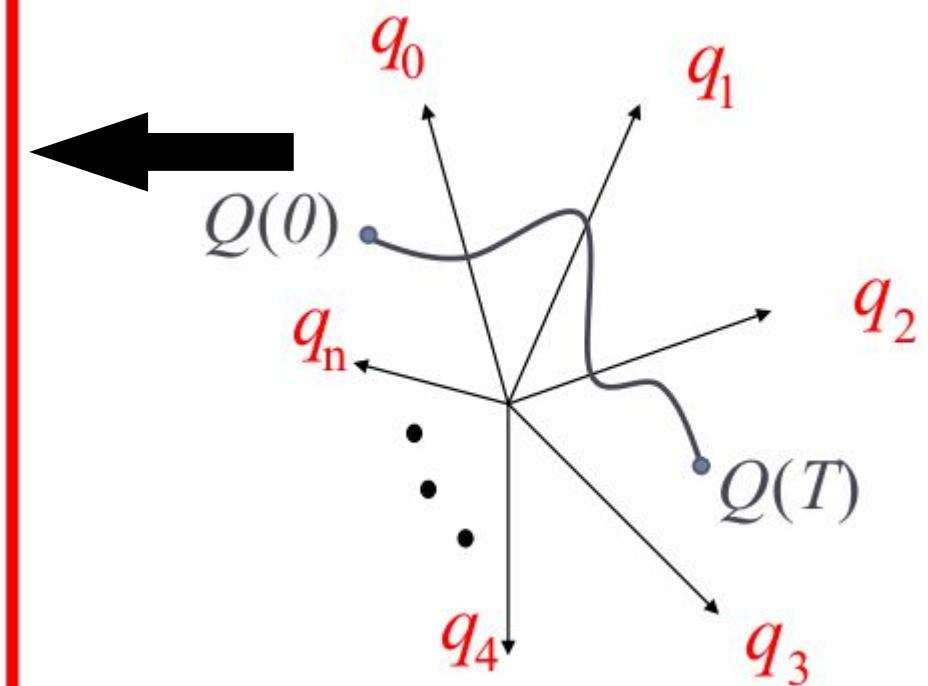
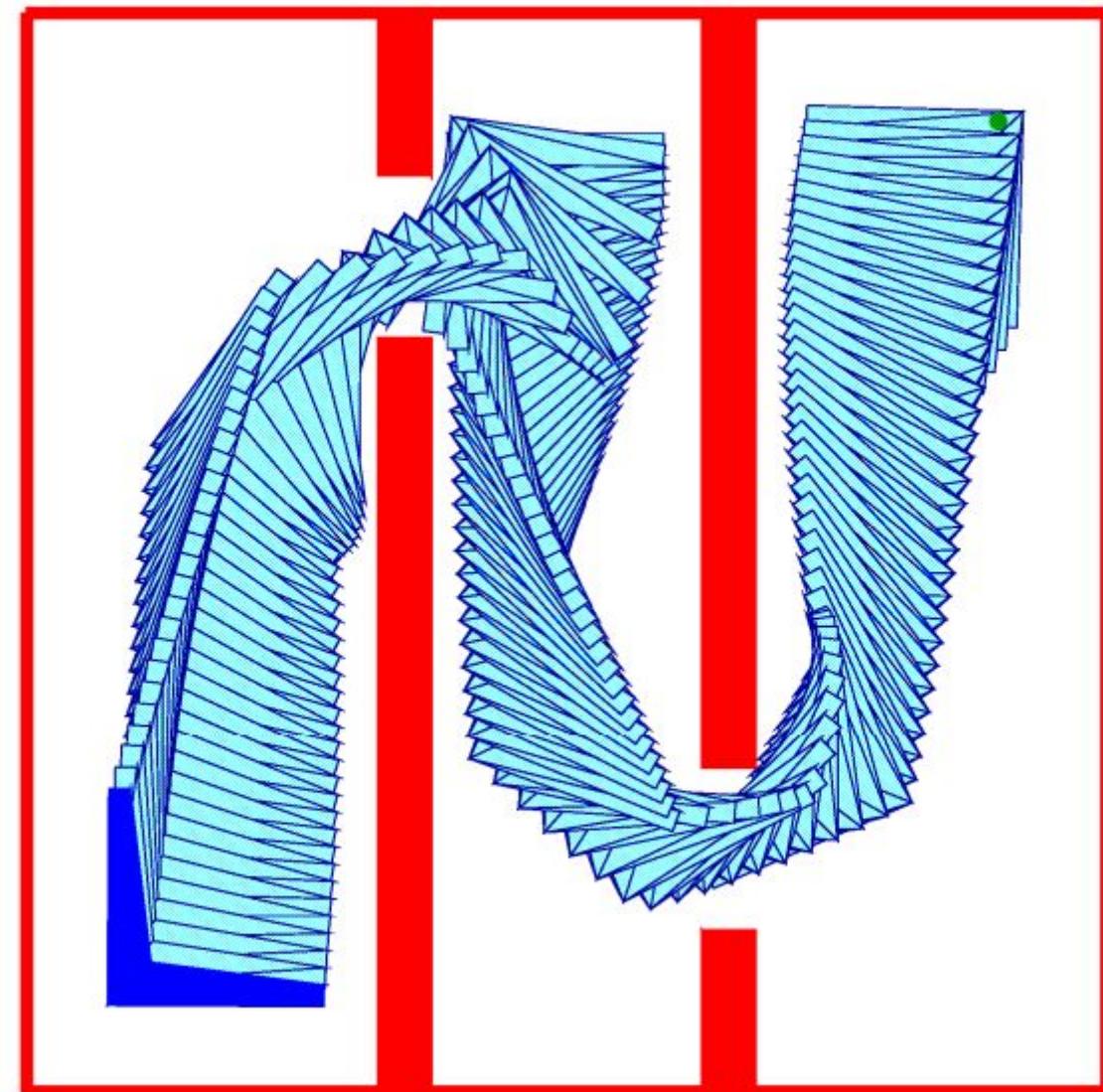
状態空間



障害物

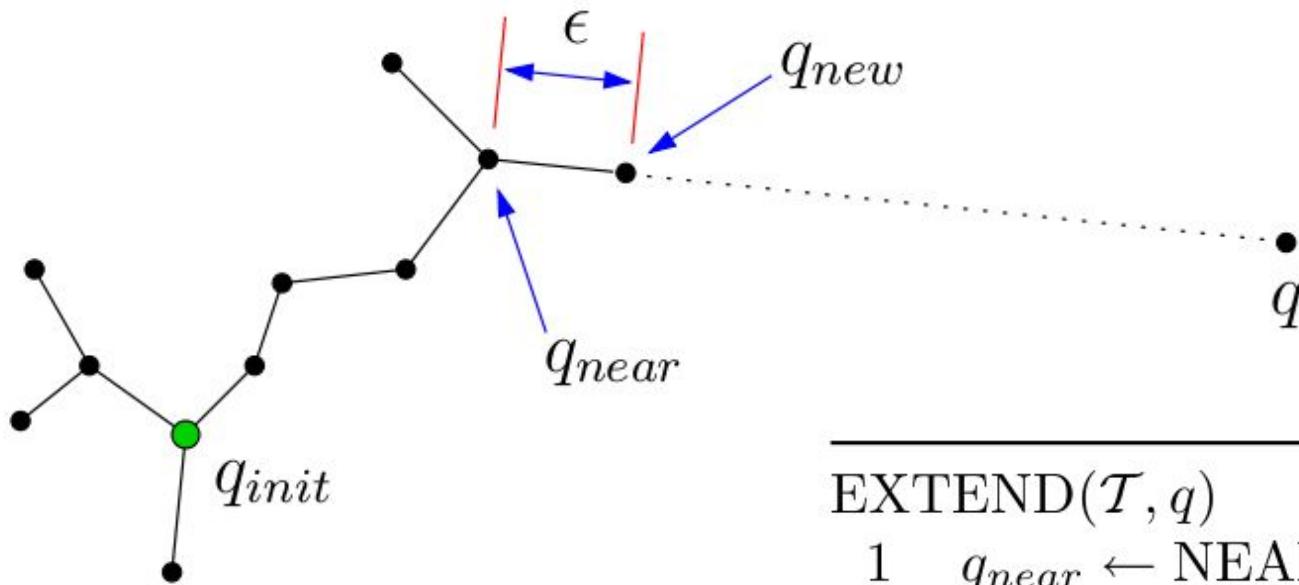


# 動作計画の目的：状態空間の経路



# Rapidly-exploring Random Trees (RRTs)

James Kuffner, Steve Lavalle  
カーネギーメロン大学



BUILD\_RRT( $q_{init}$ )

```
1  $\mathcal{T}.\text{init}(q_{init});$ 
2 for  $k = 1$  to  $K$  do
3    $q_{rand} \leftarrow \text{RANDOM\_CONFIG}();$ 
4   EXTEND( $\mathcal{T}, q_{rand}$ );
5 Return  $\mathcal{T}$ 
```

---

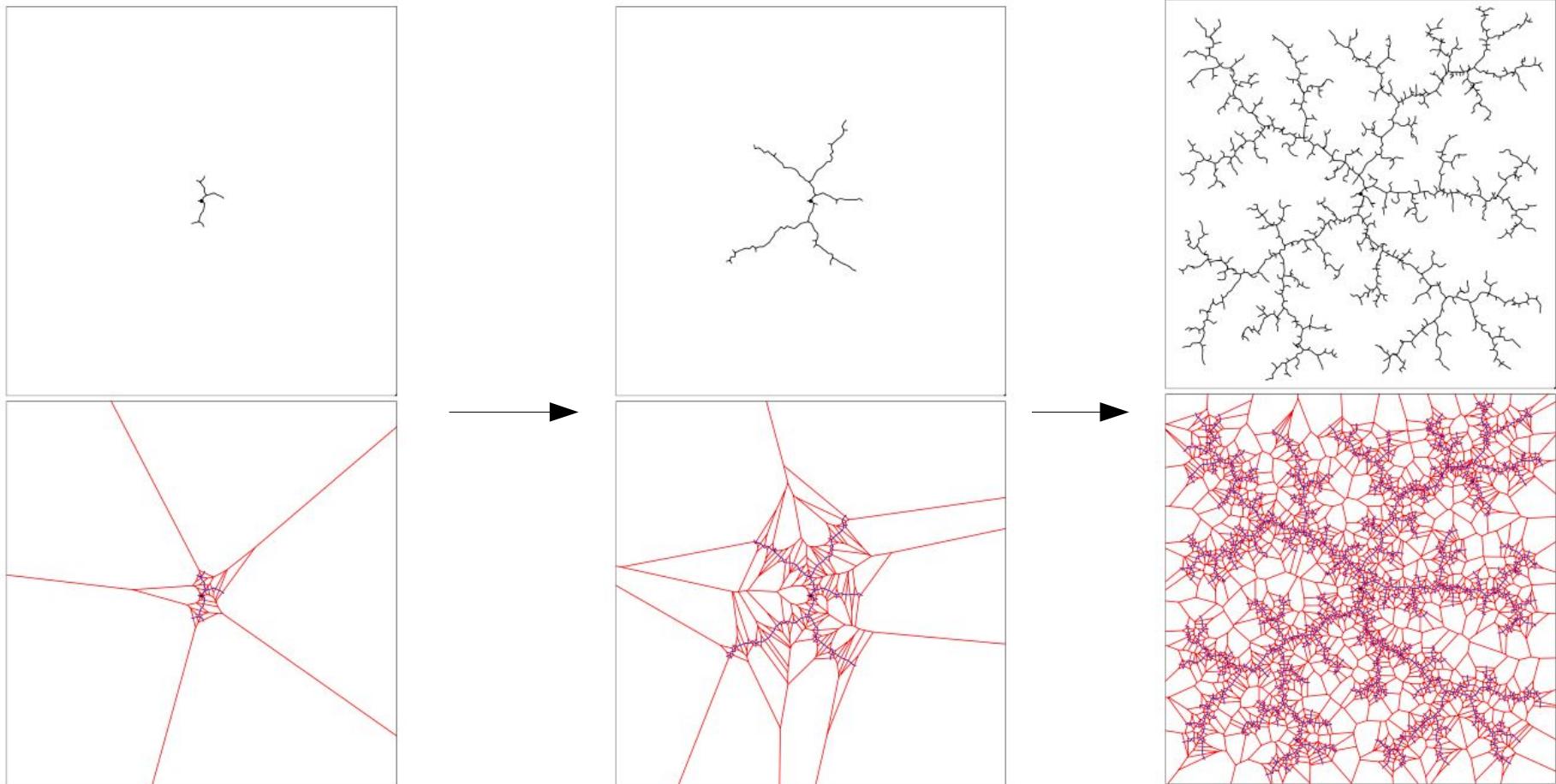
EXTEND( $\mathcal{T}, q$ )

```
1  $q_{near} \leftarrow \text{NEAREST\_NEIGHBOR}(q, \mathcal{T});$ 
2 if NEW_CONFIG( $q, q_{near}, q_{new}$ ) then
3    $\mathcal{T}.\text{add\_vertex}(q_{new});$ 
4    $\mathcal{T}.\text{add\_edge}(q_{near}, q_{new});$ 
5   if  $q_{new} = q$  then
6     Return Reached;
7   else
8     Return Advanced;
9 Return Trapped;
```

---

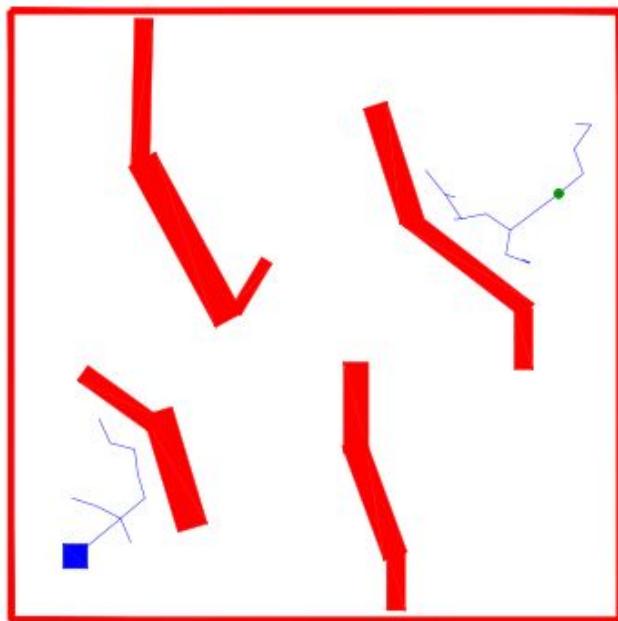
# RRTsの探索傾向

- 不探索空間への生きがち



Voronoi Region

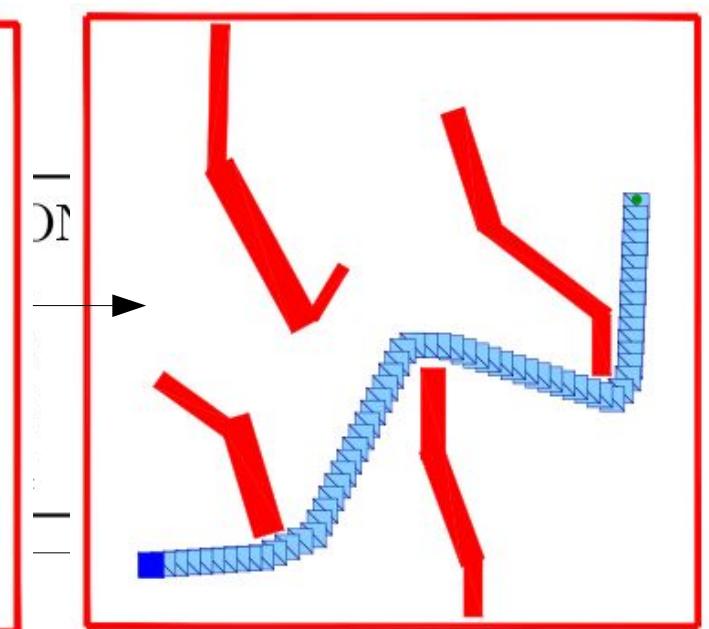
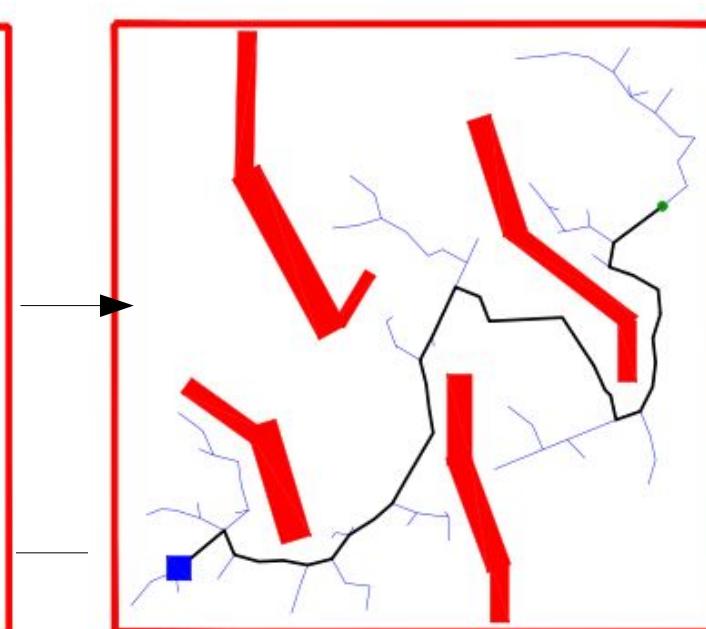
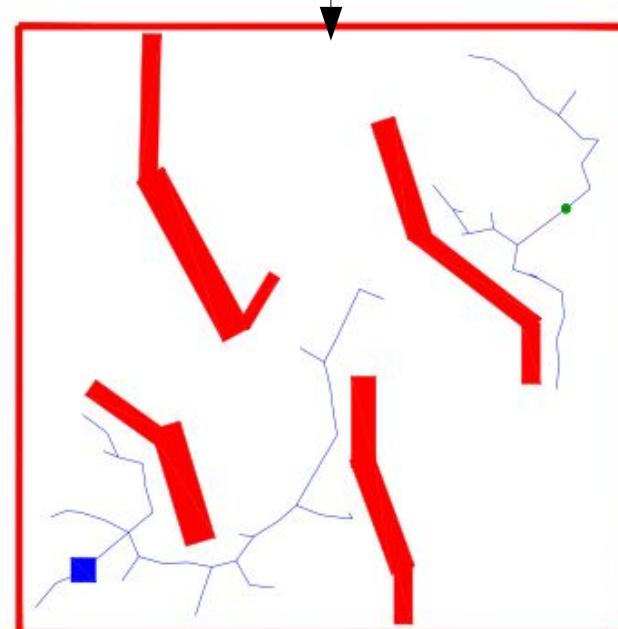
# RiRRT<sub>c</sub> (両進探索)



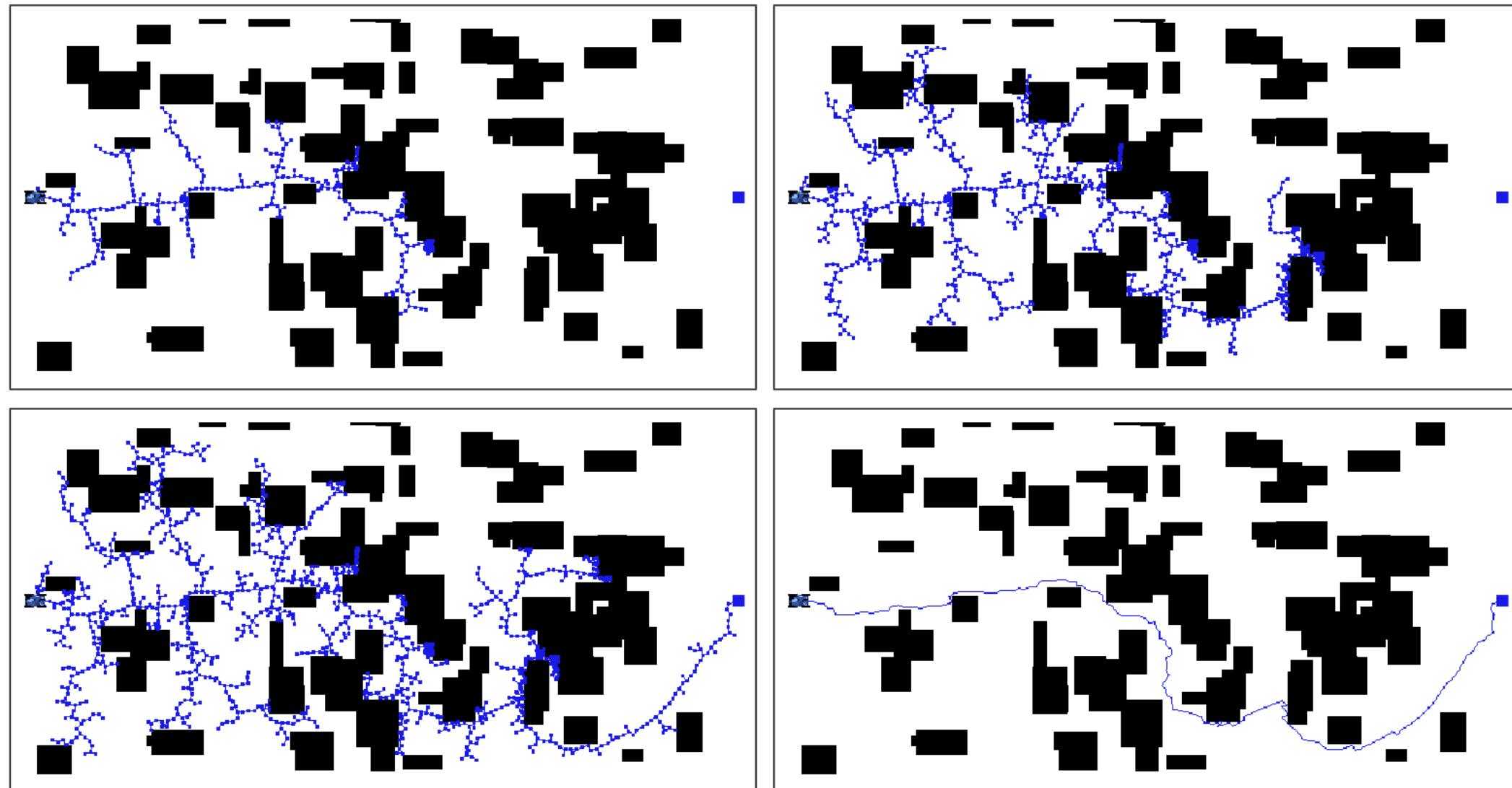
---

```
RRT_CONNECT_PLANNER( $q_{init}, q_{goal}$ )
1  $\mathcal{T}_a.init(q_{init}); \mathcal{T}_b.init(q_{goal});$ 
2 for  $k = 1$  to  $K$  do
3    $q_{rand} \leftarrow \text{RANDOM\_CONFIG}();$ 
4   if not ( $\text{EXTEND}(\mathcal{T}_a, q_{rand}) = \text{Trapped}$ ) then
5     if ( $\text{CONNECT}(\mathcal{T}_b, q_{new}) = \text{Reached}$ ) then
6       Return PATH( $\mathcal{T}_a, \mathcal{T}_b$ );
7     SWAP( $\mathcal{T}_a, \mathcal{T}_b$ );
8 Return Failure
```

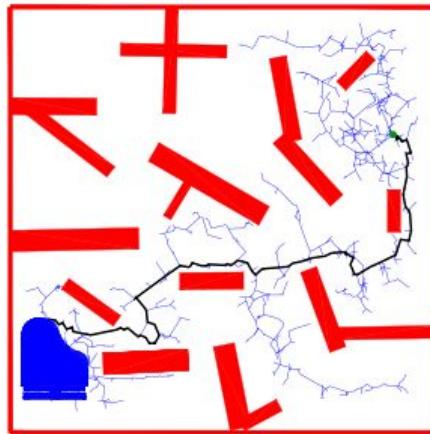
---



# RRT探索例



# 状態空間の柔軟性

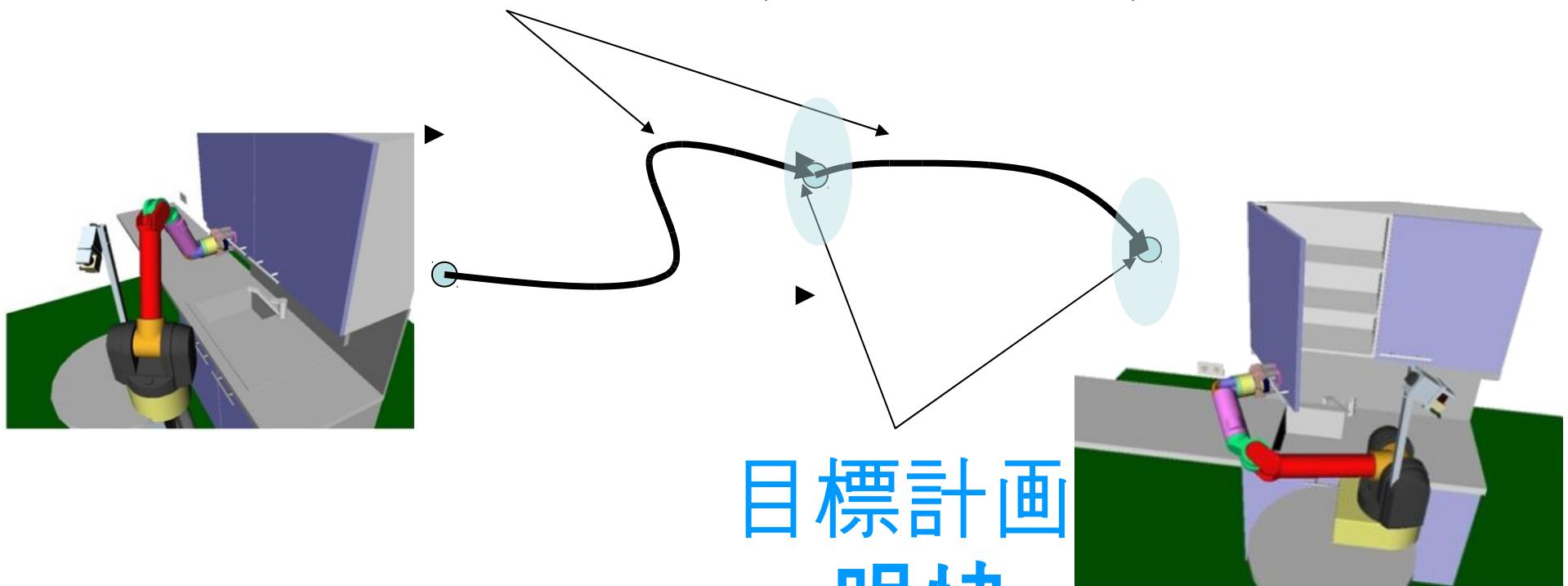


# マニピュレーション 動作計画

OpenRAVE

# 計画の二段階

動作経路 (BiRRT, Delayed D\*, Anytime RRTs)

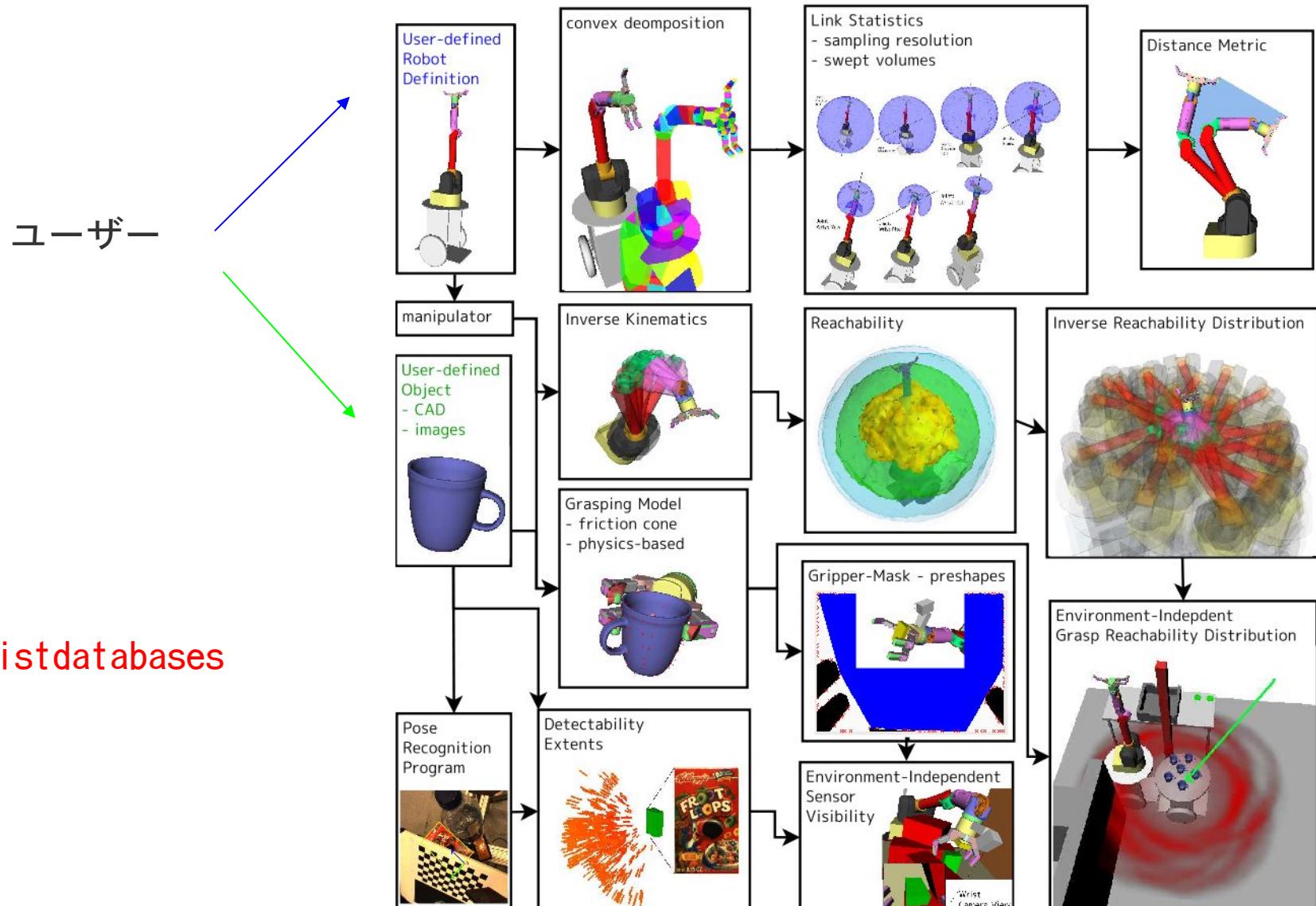


目標計画  
- 明快  
- 暗黙

# 動作と作業目標をつなぐ 知識ベース

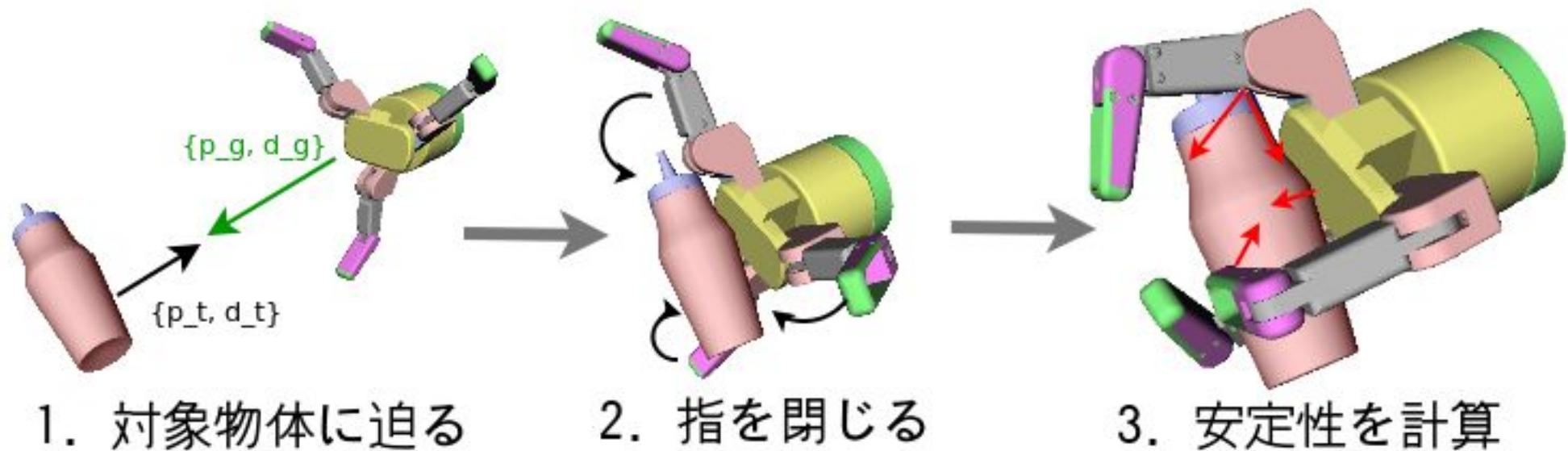
ユーザー

openrave.py --listdatabases



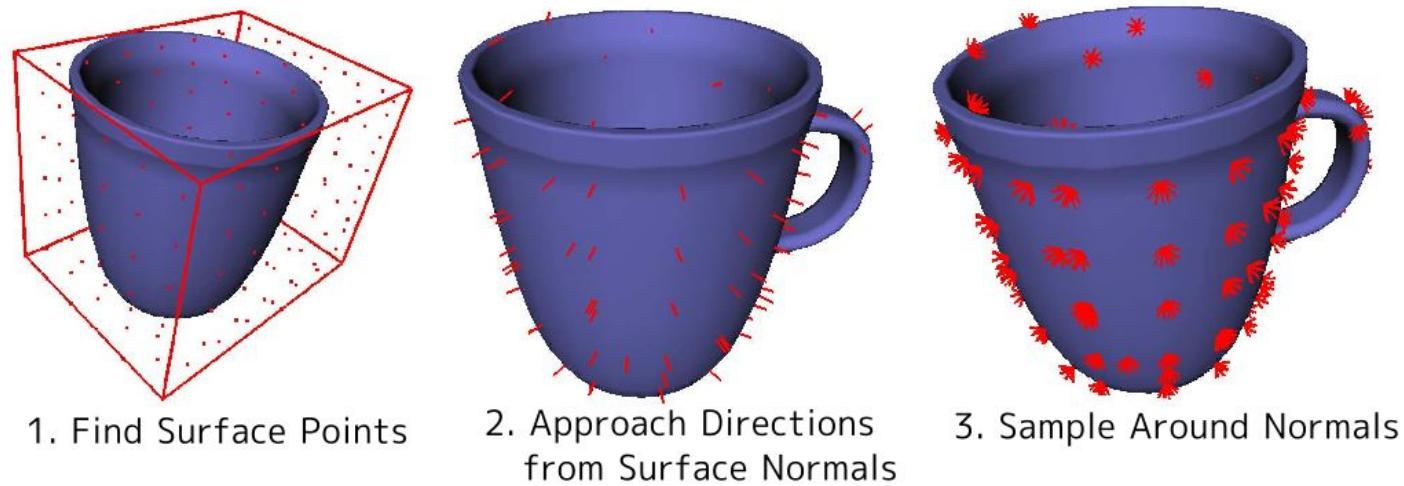
# 知識ベース：把持

- ・ 戰略
- ・ 把持空間のパラメータ化
- ・ 把持安定性測定法
- ・ OpenRAVE：
  - ・ 把持セットの作成

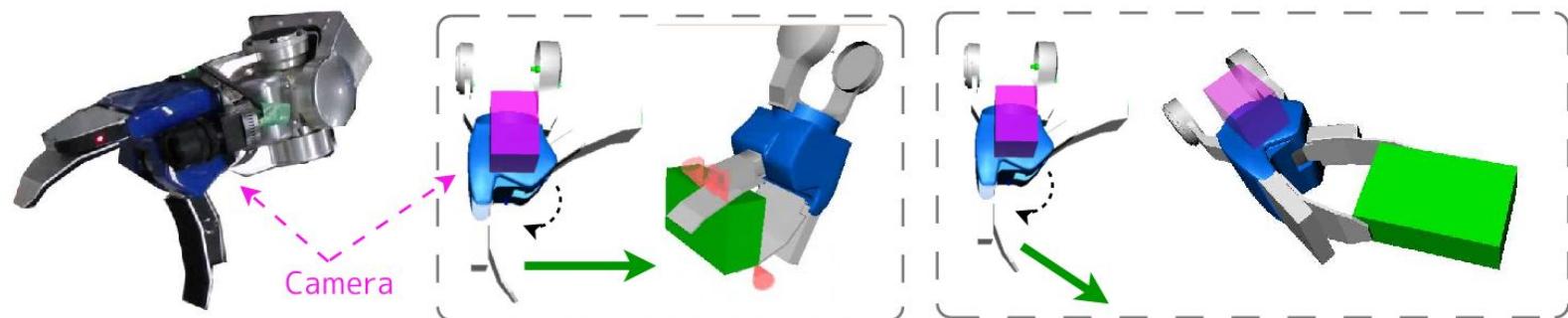


# 把持空間のパラメタ化：8自由度

物体表面上  
サンプリング  
(4DOF)

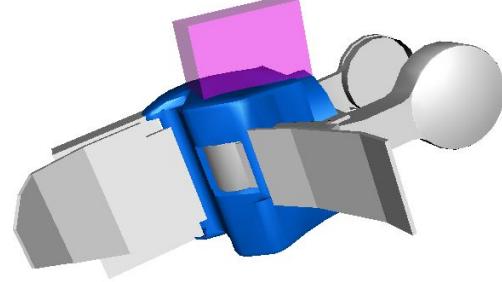


ロボット  
表面上  
(4DOF)



Stand-off (1DOF)

# 把持セット



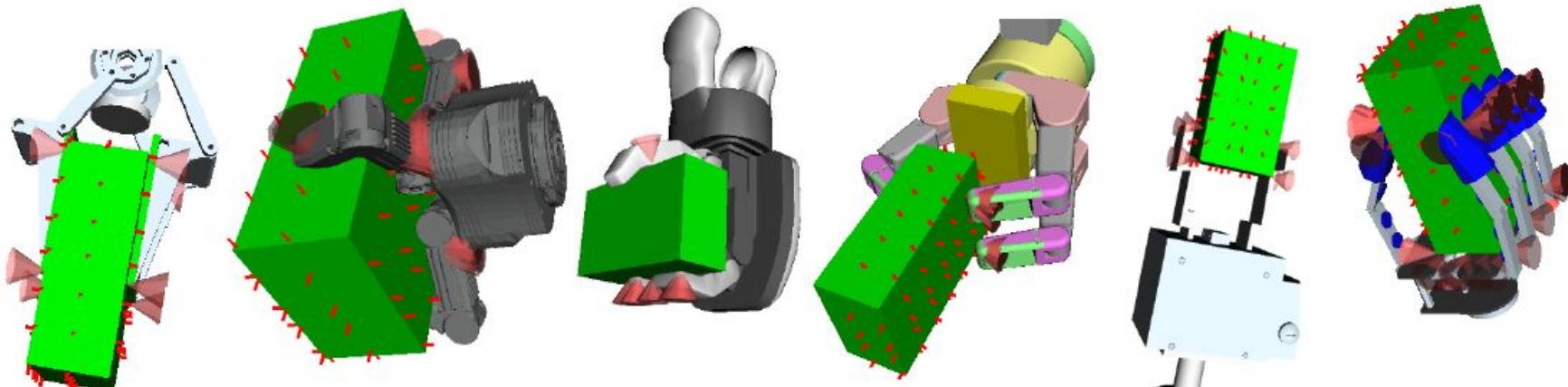
`openrave.py --database grasping`

把持セットの表示

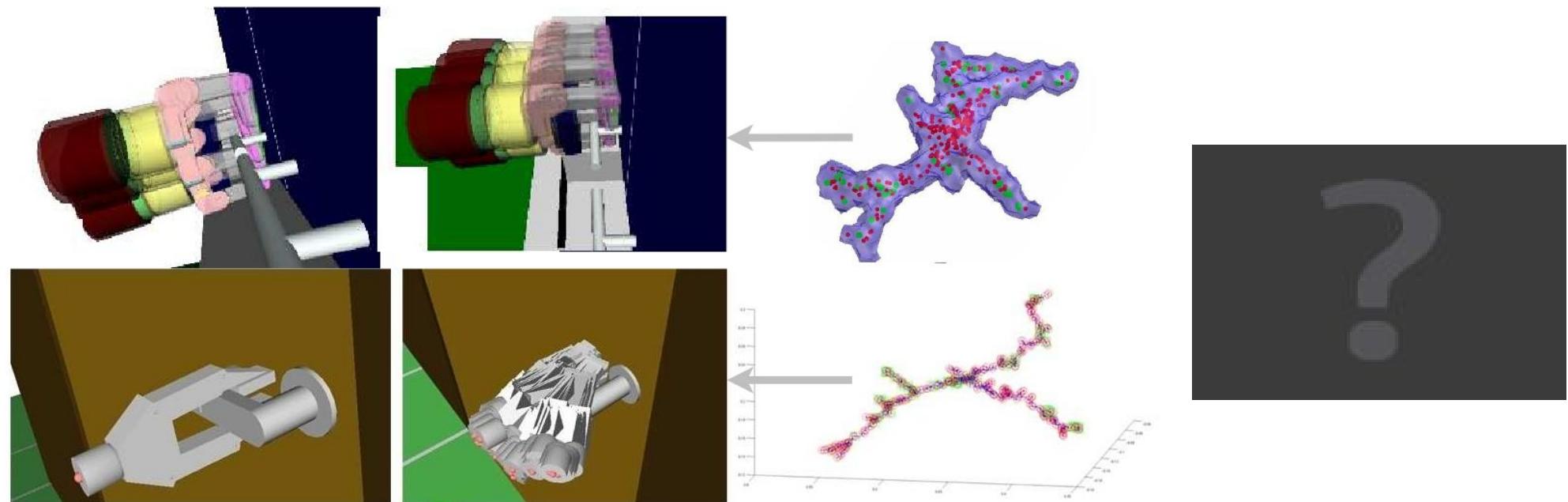
`openrave.py --database grasping --show`

ノイズを加える

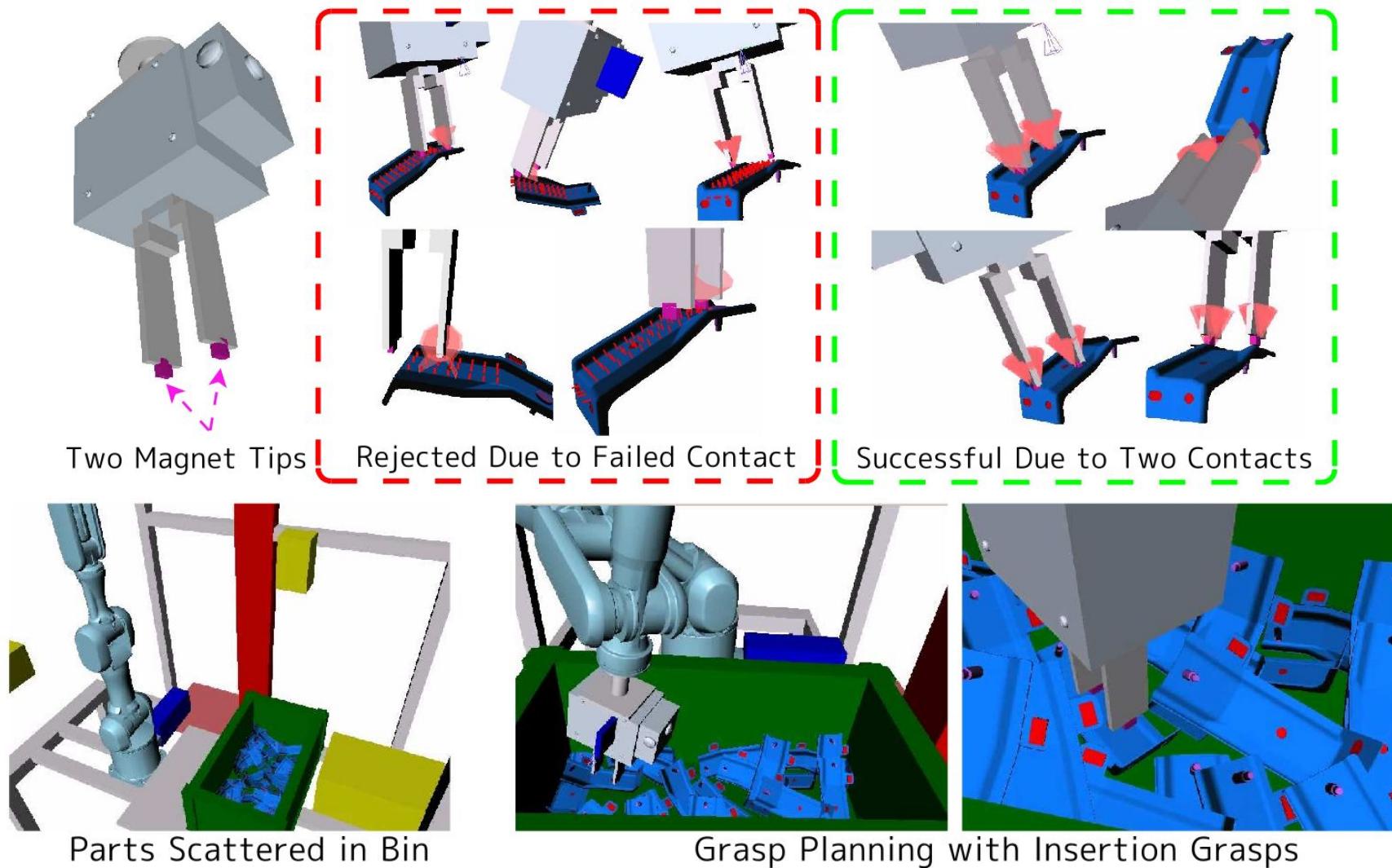
`openrave.py --database grasping --graspingnoise=0.01`



# 把持を空間で表現する



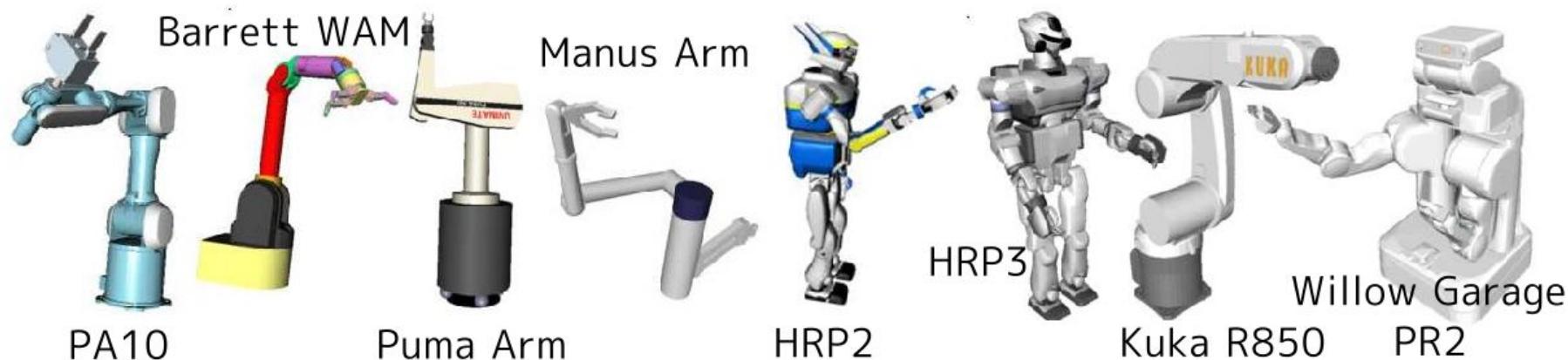
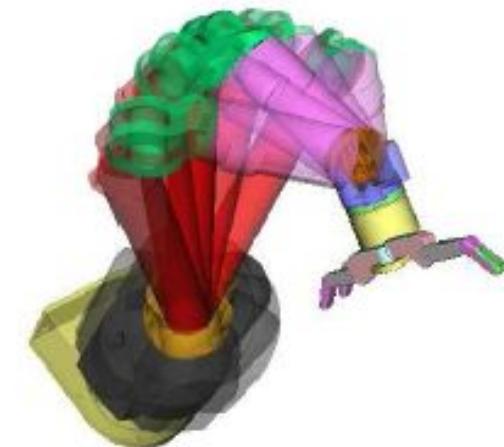
# 接触のみの把持



# 運動学の解析的な解

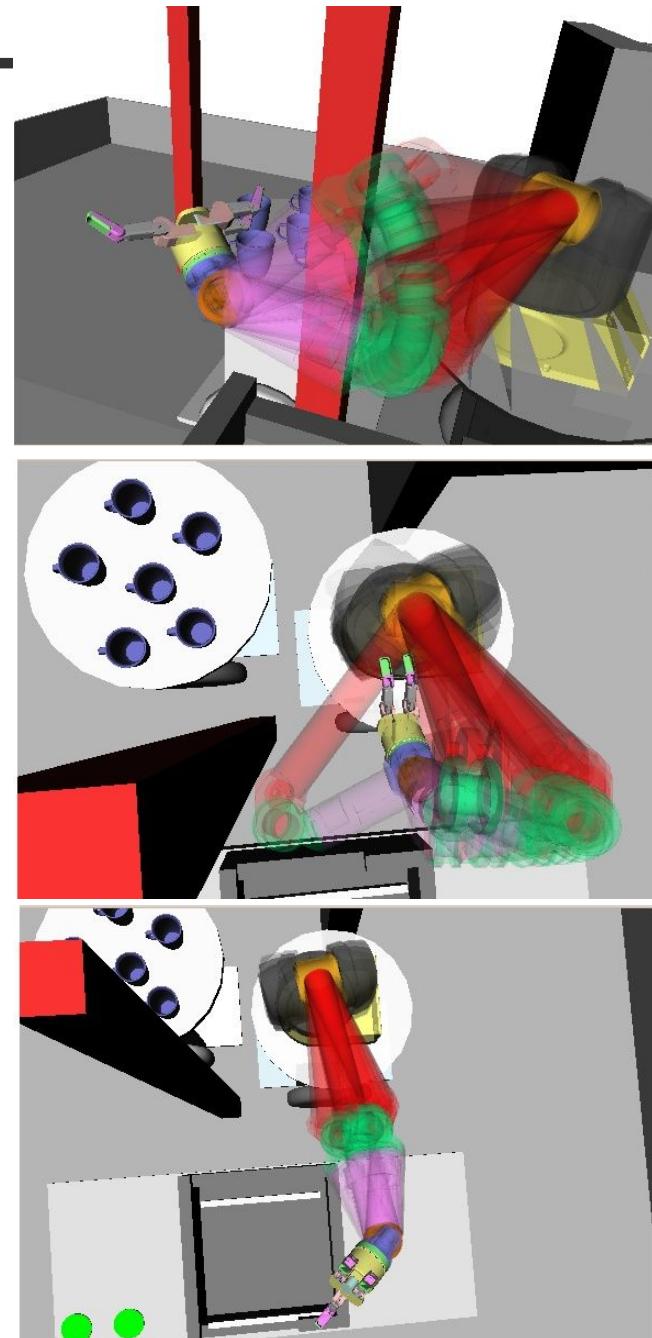
[Low,Dubey (1987)], [Raghavan, Roth (1990)], [Manocha,Zhu (1994)]

- **ikfast** - 最適化かつ安定化されたC++生成
- ゼロ除算の検出
- degenerateケースの検出
  - 軸の整列
- 全解の返し ~6マイクロ秒
- 6D, 3D移動, 3D回転, 4D光線, 2D方向、3D



# 冗長なマニピュレ-

- 6関節を選択
- 逆運動学関数の入力
  - 6D姿勢
  - 残りの関節度
- 含まれるチェック
  - 環境障害物衝突
  - 自己衝突
  - 関節のリミット
  - カスタム・フィルター



# IKの生成方法

openrave.py --database inversekinematics

成功率の試験

openrave.py --database inversekinematics --usecached --iktests=100

新しいロボットを指定する

openrave.py --database inversekinematics --robot=robots/pa10.robot.xml

新しいロボットの特定のマニピュレーターを指定する

openrave.py --database inversekinematics --robot=robots/pr2-beta-static.robot.xml manipname=leftarm

フリー関節の選択：

openrave.py --database inversekinematics --robot=robots/pa10.robot.xml --freejoint=S3

関節の出力：

openrave.py -p 'print env.GetBodies()[0].GetJoints()' robots/pa10.robot.xml

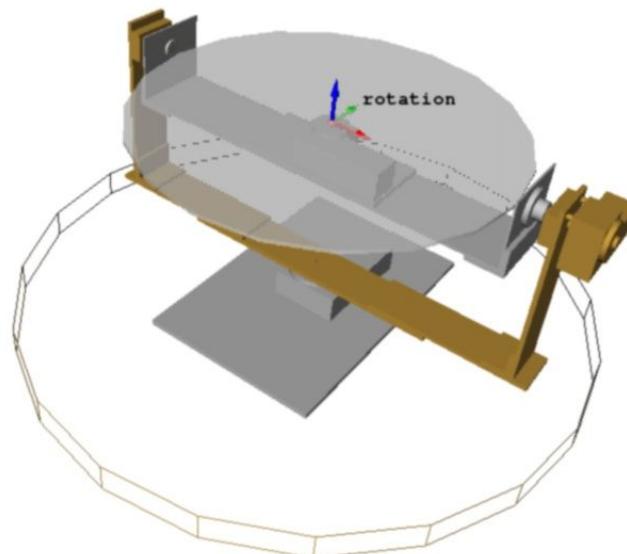
ファイル名：

openrave.py --database inversekinematics --getfilename

# 3D回転

- ・ロボットリンクの3D回転を世界座標系の3D回転に合わせる

```
openrave.py --database inversekinematics --robot=robots/rotation_stage.robot.xml  
--iktype=rotation3d --iktests=100 --usecached
```



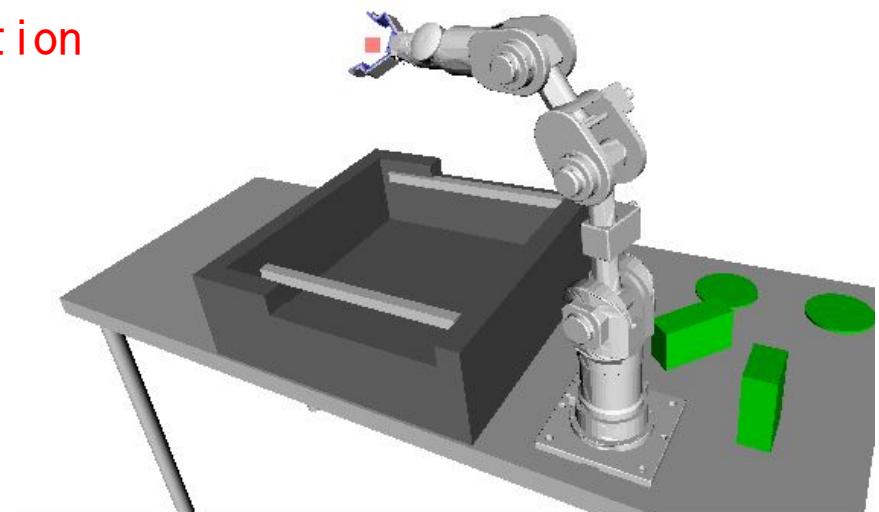
# 3D移動

- ・ロボット上の3D位置を世界座標系の3D位置に合わせる

```
openrave.py --database inversekinematics --robot=robots/katana.robot.xml  
--iktype=translation3d --freejoint=j4 --freejoint=j5 --iktests=100  
--usecached
```

サンプリングの例

```
openrave.py --example tutorial_iktranslation
```



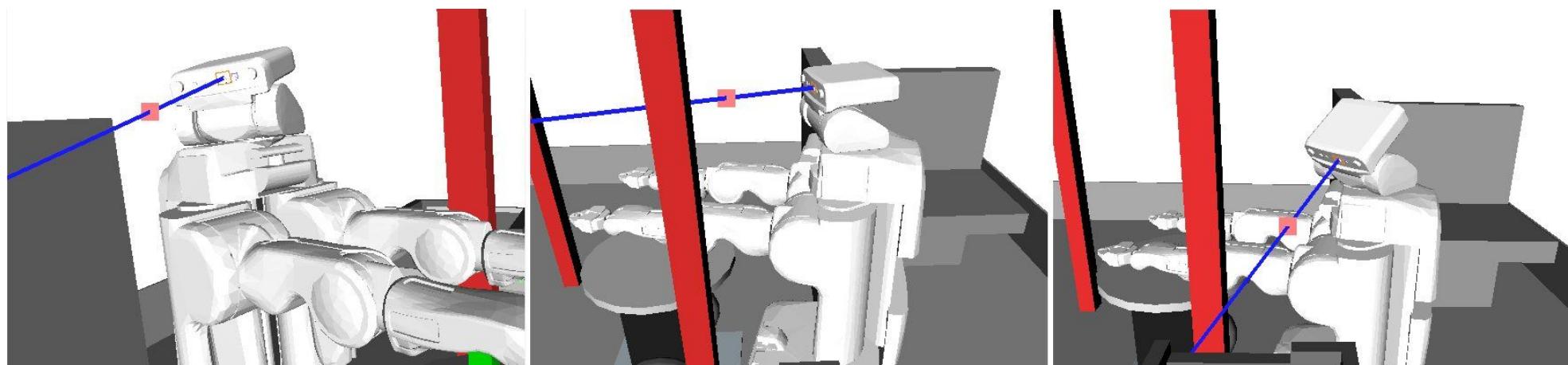
# 3D対象

- 対象の位置に光線ベクトルを合わせる

```
openrave.py --database inversekinematics --robot=robots/pr2-beta-static.robot.xml --manipname=head --iktype=lookat3d --usecached --iktests=100
```

サンプリングの例

```
openrave.py --example tutorial_iklookat
```



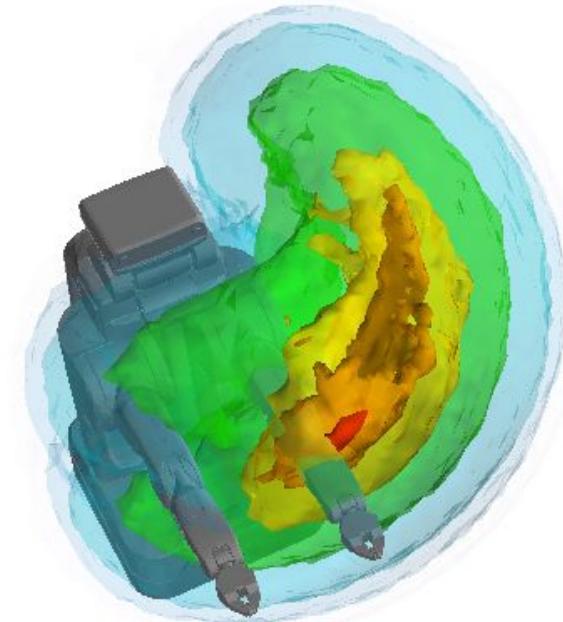
# ikfastの計算速度

	Computation	Success	Discretization
Mitsubishi PA10	$7\mu s$	100%	0.1 radians
Barrett WAM	$6\mu s$	100%	0.1 radians
Puma Arm	$6\mu s$	100%	
Manus Arm	$6\mu s$	100%	
HRP2	$6\mu s$	99.5%	0.1 radians
Kuka R850	$5\mu s$	100%	
Willow Garage PR2	$6\mu s$	98.2%	0.1 radians
Willow Garage PR2	$6\mu s$	99.2%	0.02 radians
WAM Ray (Figure 4.7)	$4\mu s$	99.99%	

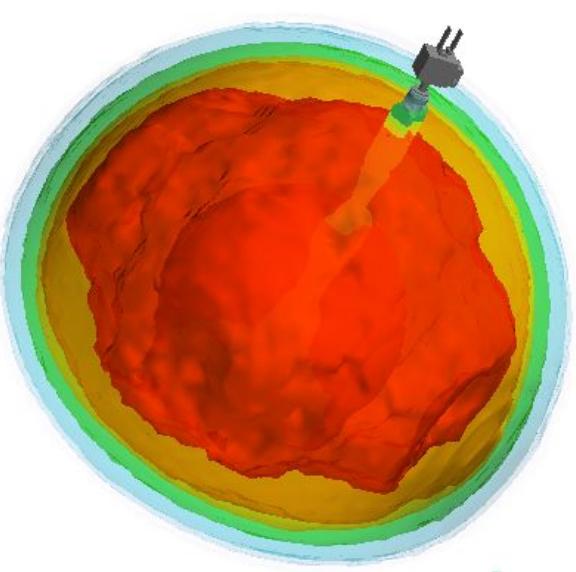
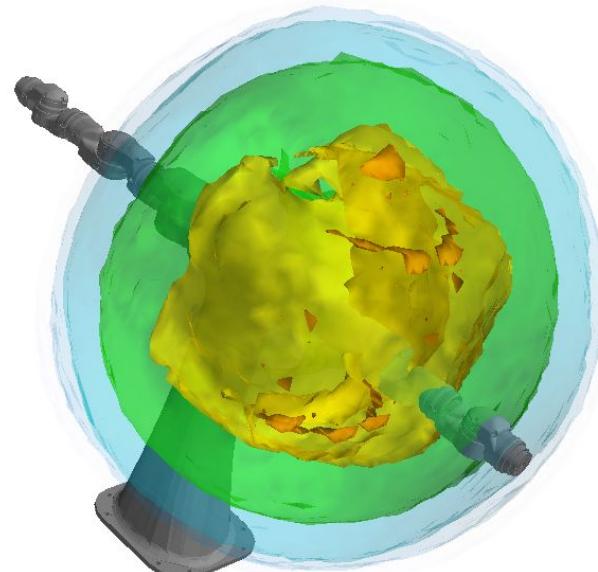
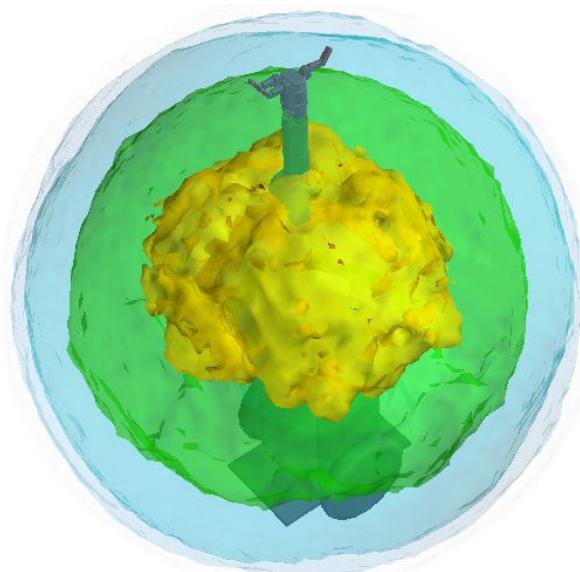
# 運動学の到達範囲

[Zacharias (2007-2010)]

- ・ハンドの6D位置姿勢のマップ



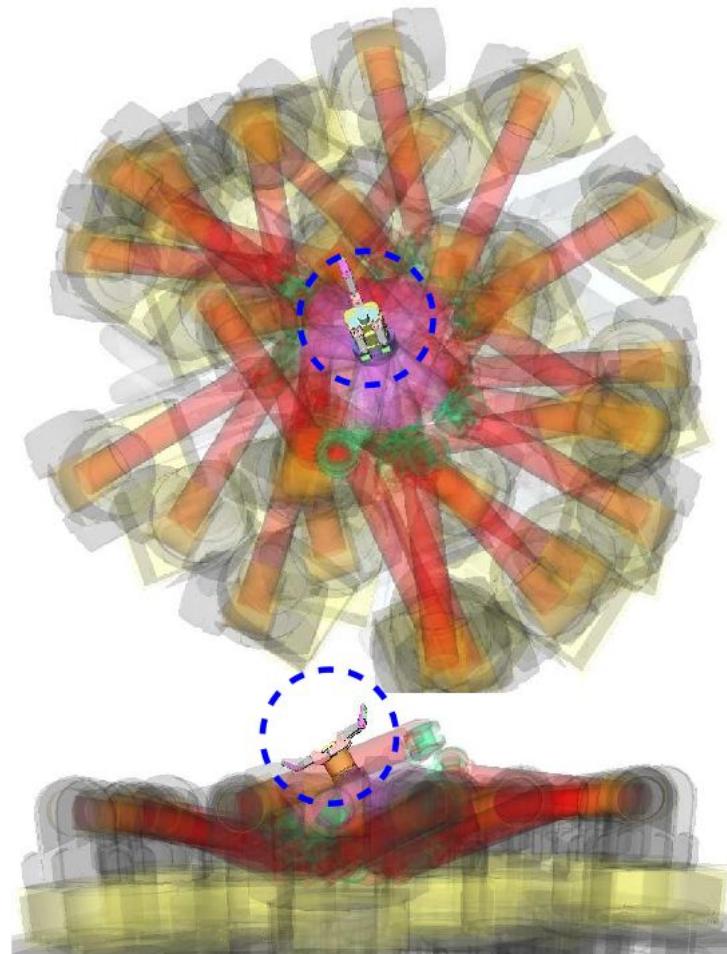
```
openrave.py --database kinematicreachability --robot=robots/pa10.robot.xml
```



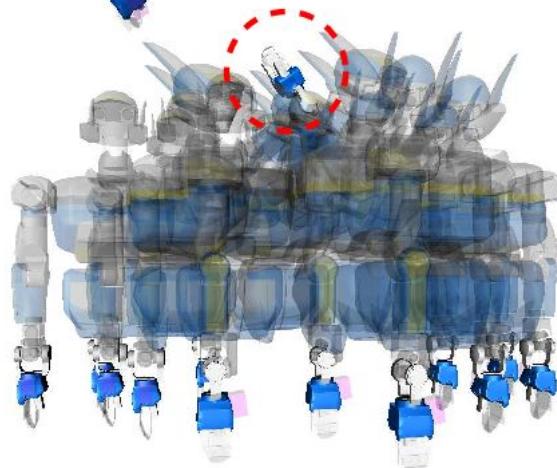
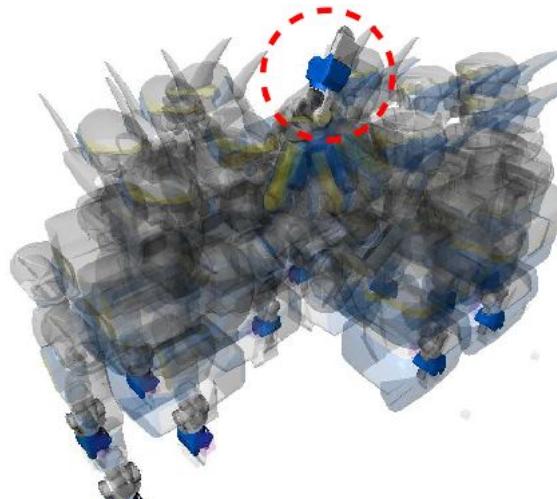
# 投影された逆運動学分布

[Stulp, Beetz (2009)], [Asfour et al (2010)]

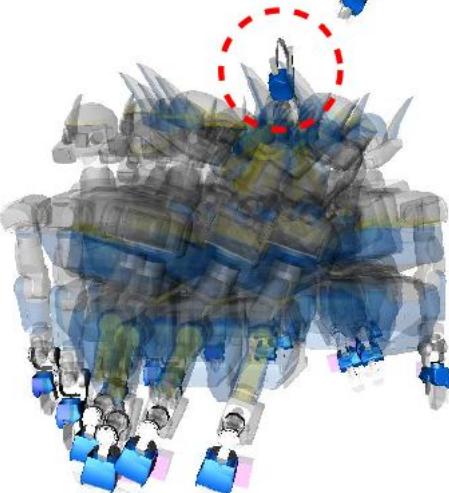
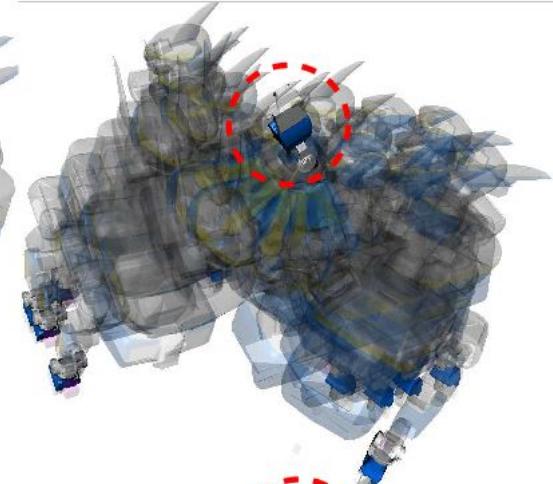
openrave.py --database inversoreachability --robot=robots/pa10.robot.xml



7 DOF Arm



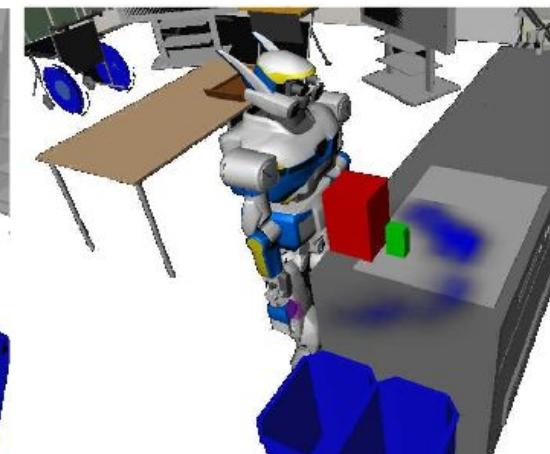
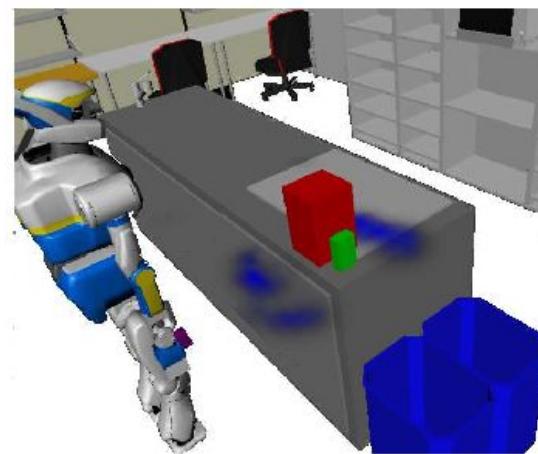
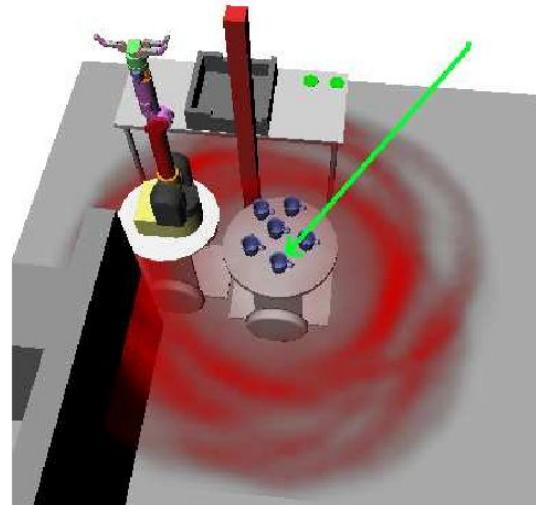
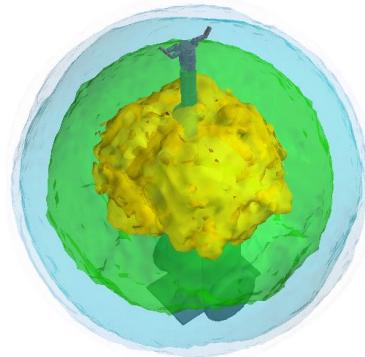
7 DOF Arm



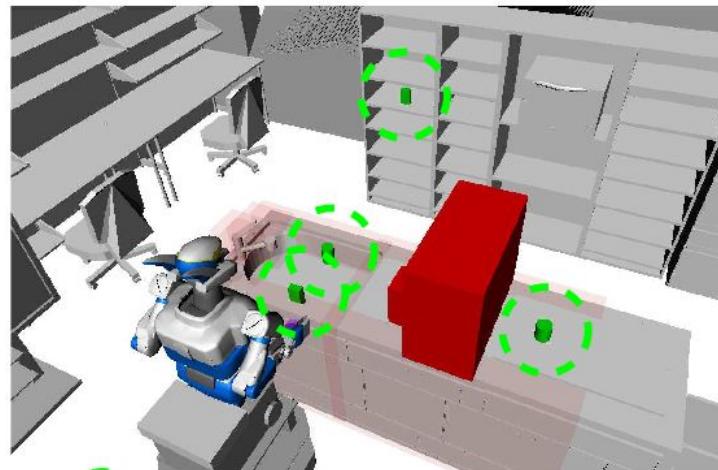
8 DOF Arm+Chest

# 把持可能位置姿勢空間

openrave.py --example tutorial\_inversereachability



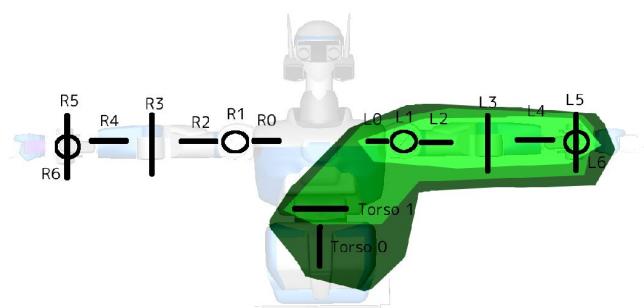
# 把持可能位置姿勢空間



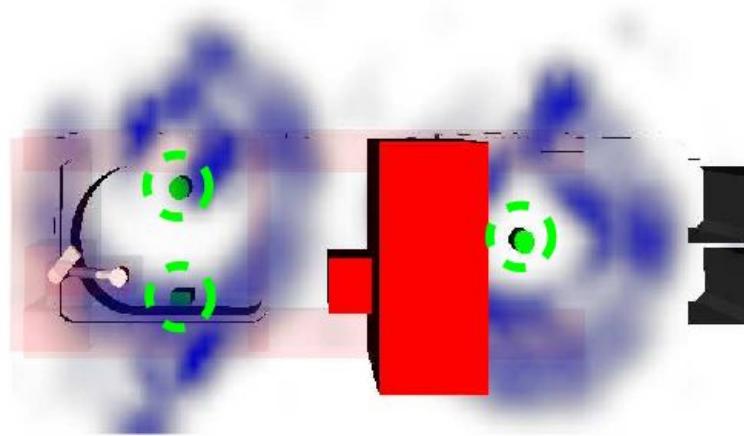
4物体

+

6アーム

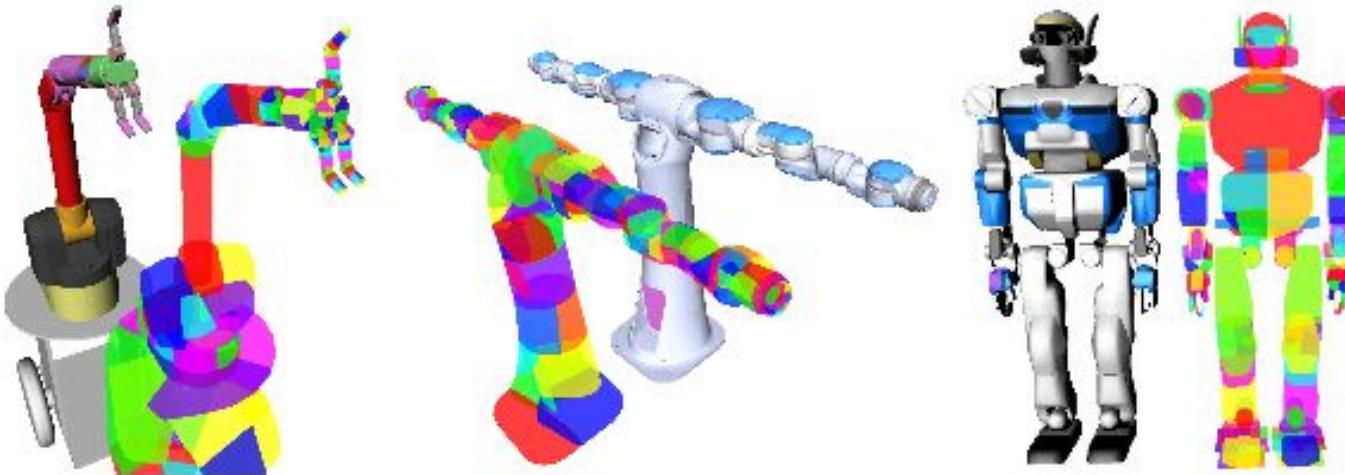


=



XY+回転の胴体分布

# 凸形状分解



## Running the Generator

```
openrave.py --database convexdecomposition --robot=robots/barrettsegway.robot.xml
```

## Showing the Decomposition

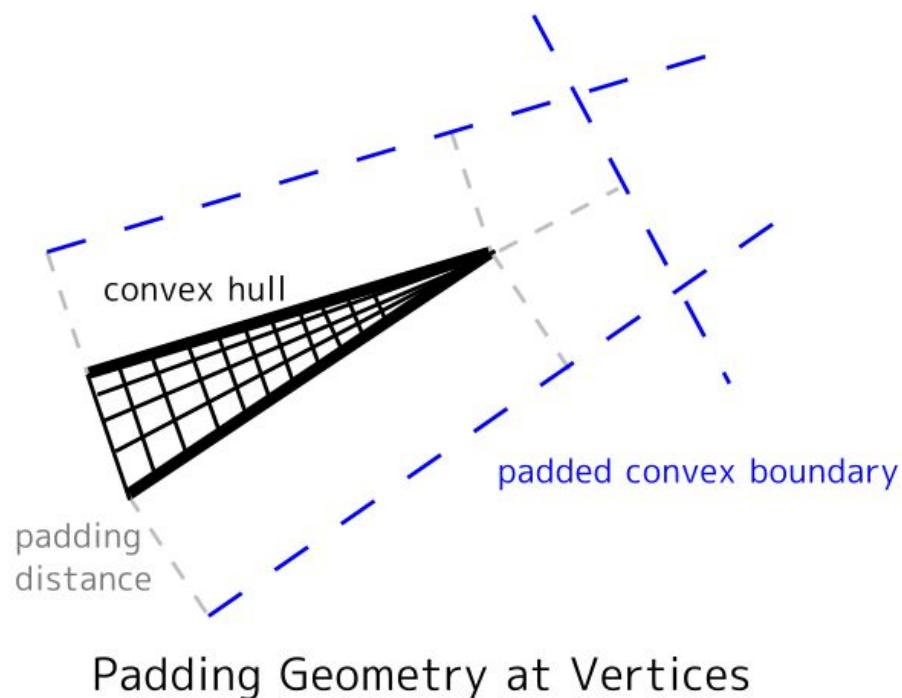
```
openrave.py --database convexdecomposition --robot=robots/barrettsegway.robot.xml --show
```

## Usage

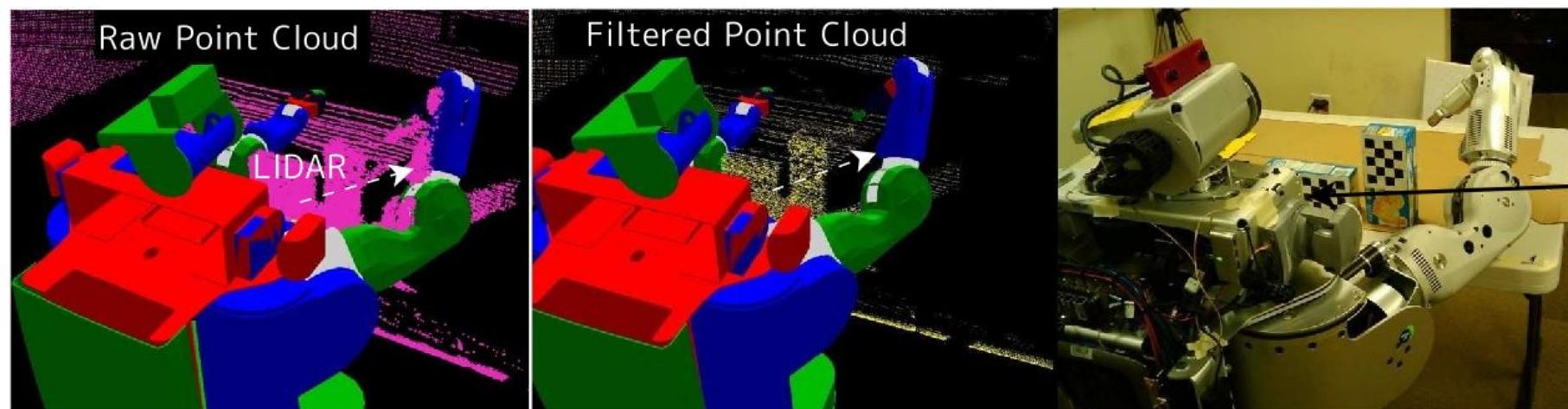
Dynamically load the convex hulls for a robot:

```
cdmodel = openravepy.databases.convexdecomposition.ConvexDecompositionModel(robot)
if not cdmodel.load():
    cdmodel.autogenerate()
```

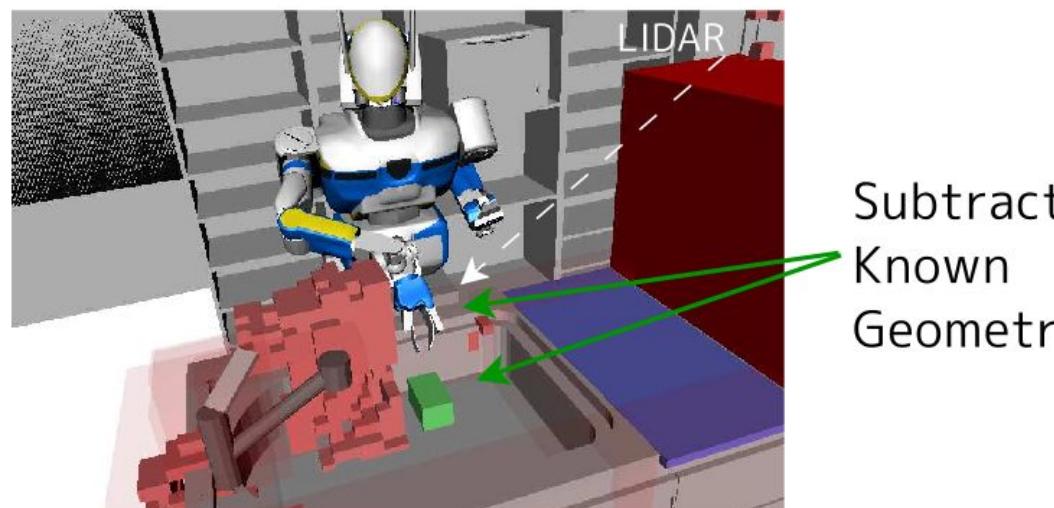
# 凸形状分解：パッディング



# 凸形状分解：体積



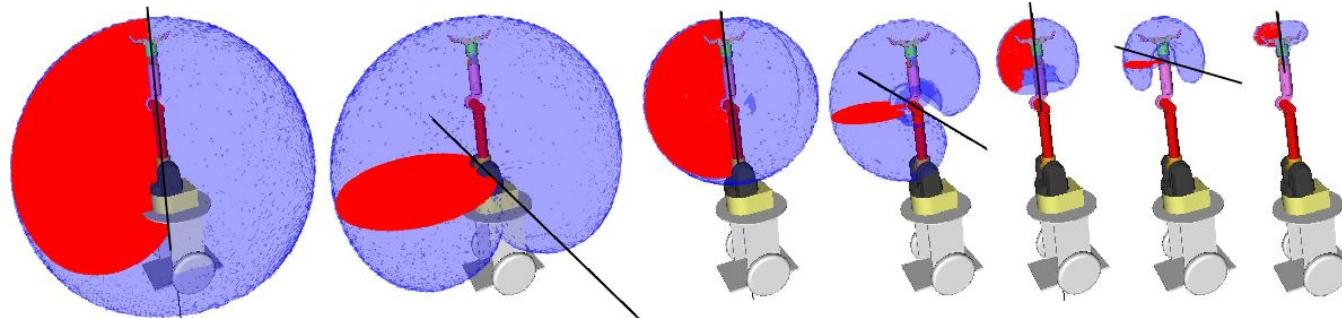
on-board  
spinning  
laser



# 凸形状分解 : Distance Metric

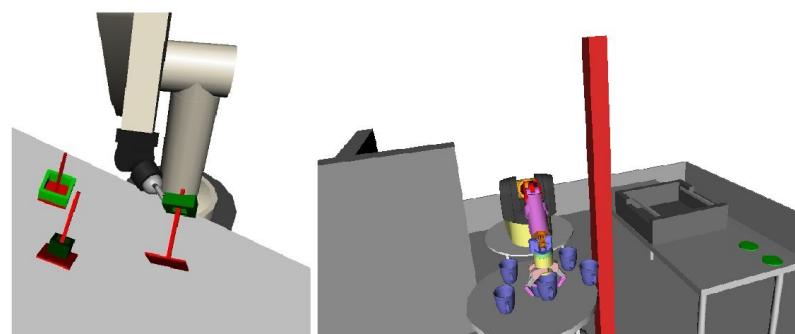
## 掃引体積

$$\omega_i = \left( \text{CROSSSECTION}(SV_i) * \sum_{j \in \text{DependentLinks}_i} LV_j \right)^{\frac{1}{3}}$$



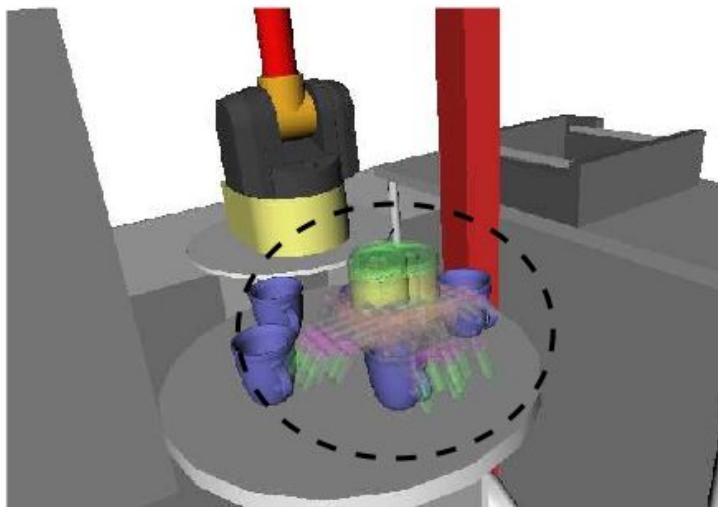
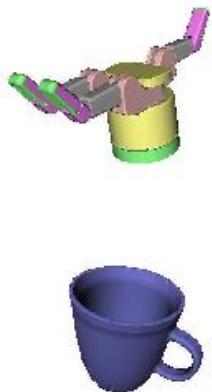
## 動作計画の処理時間

	Uniform Weights	Swept Volume Weights ( $\omega_i$ )
WAM scene	2.89s	1.24
Puma Scene	0.71s	0.57s

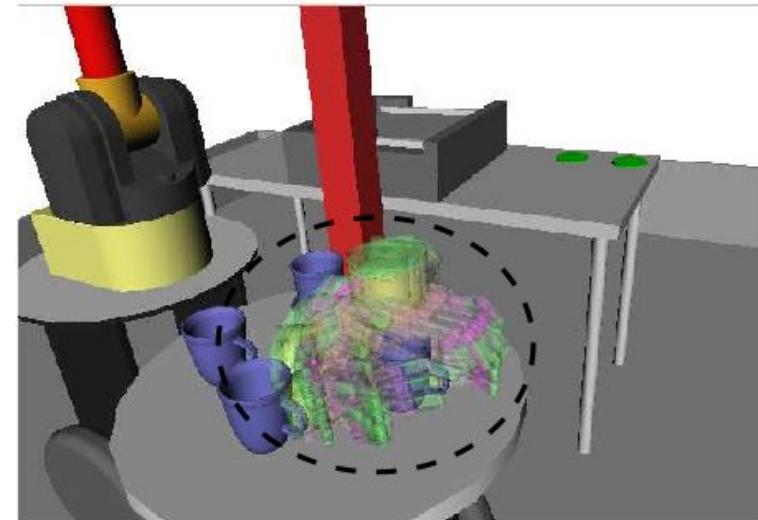


# 把持の選択

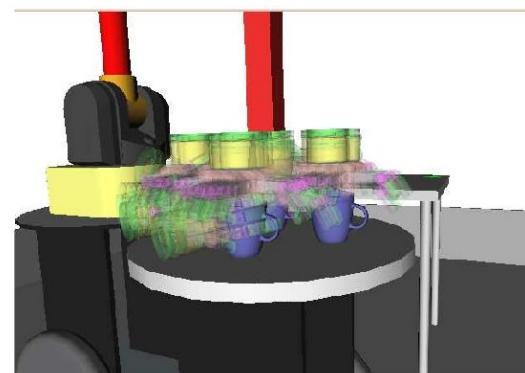
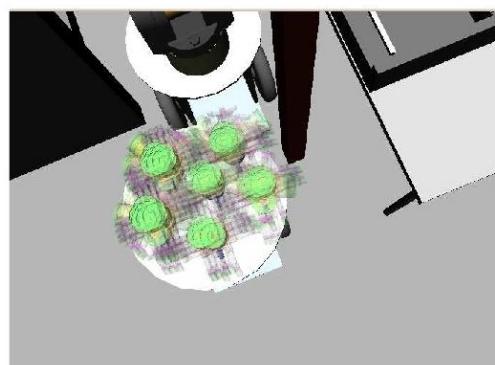
[Berenson, Diankov (2007)]



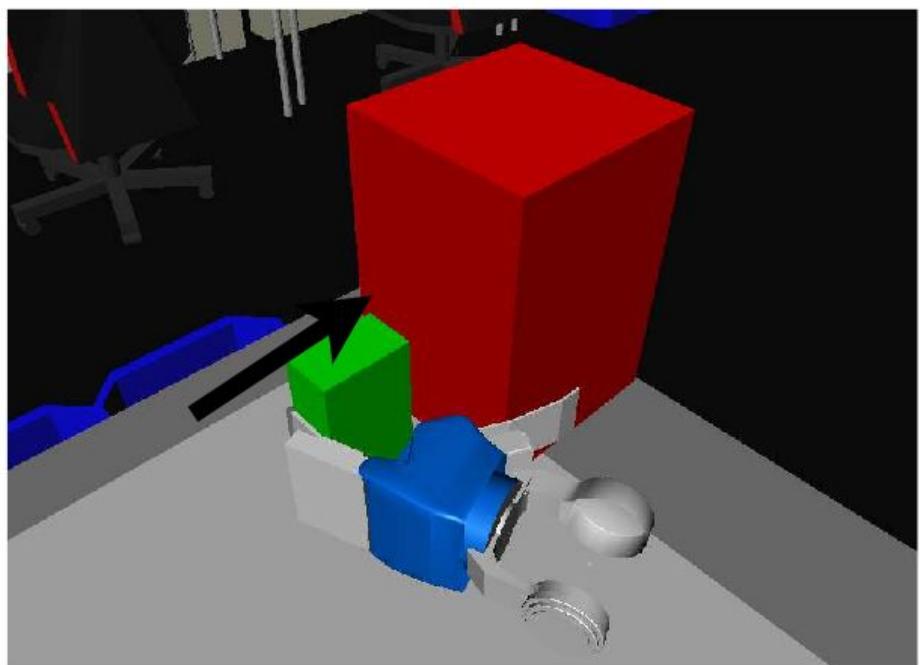
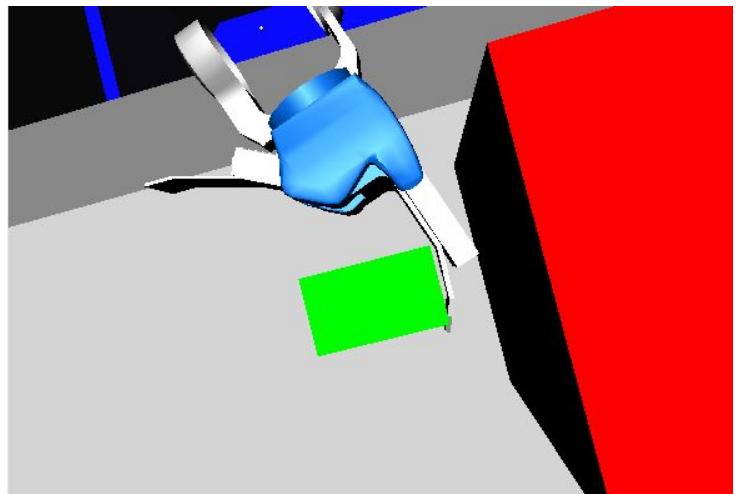
Reachable (has IK)



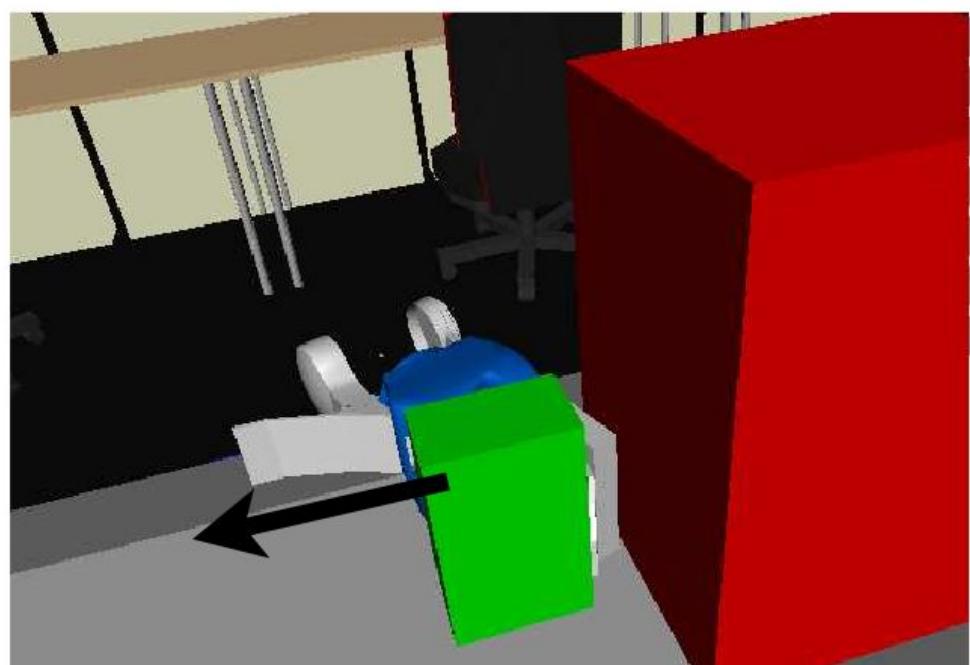
Unreachable



# 把持の検証

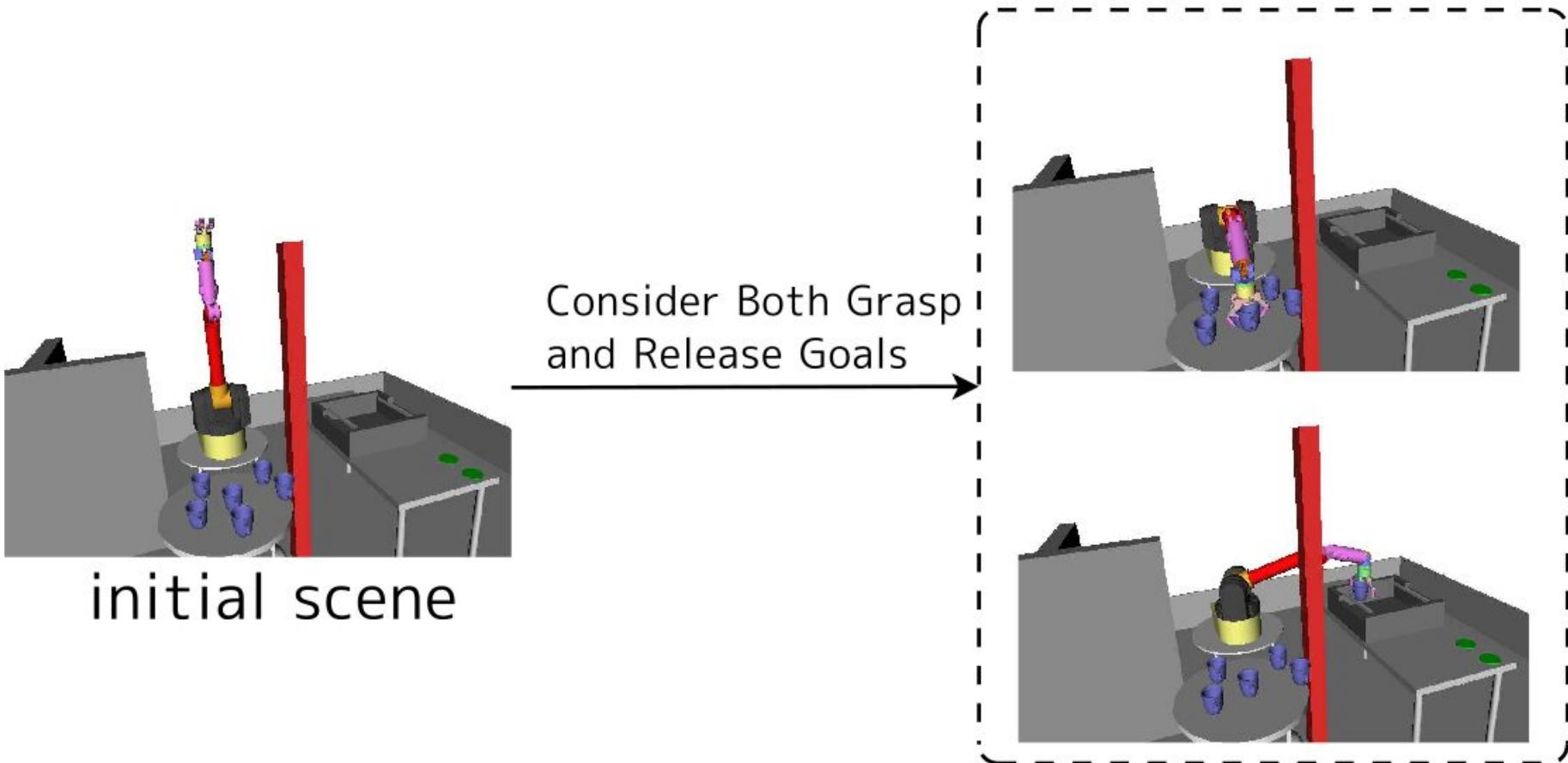


Obstacle Blocks Finger

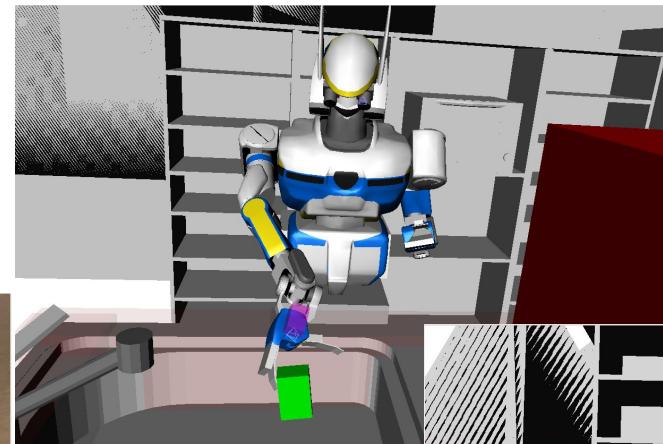
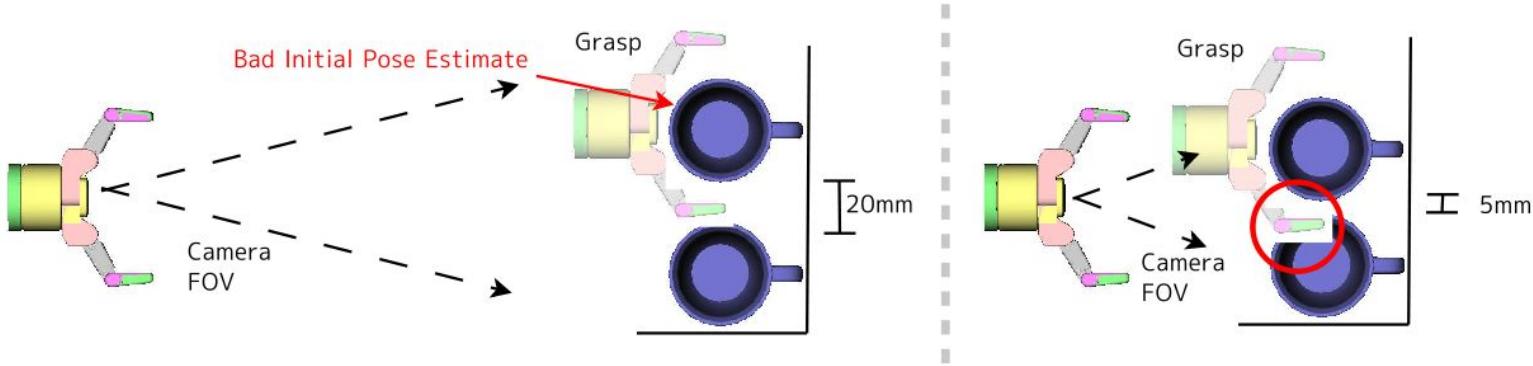


Tight Place

# 把持選択時に目的地も考慮する



# 位置姿勢の誤差



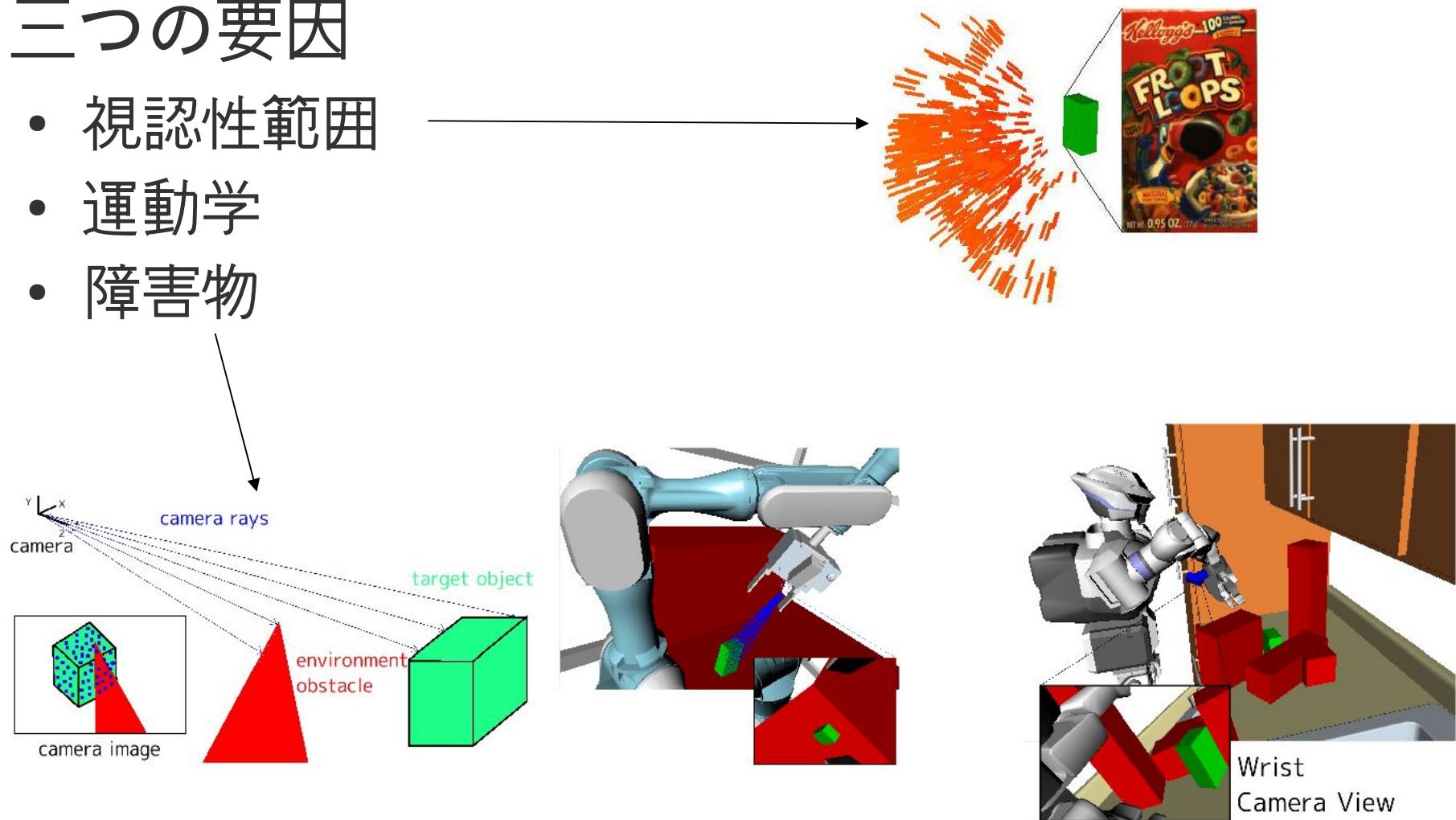
Head Camera Results

| 4cm

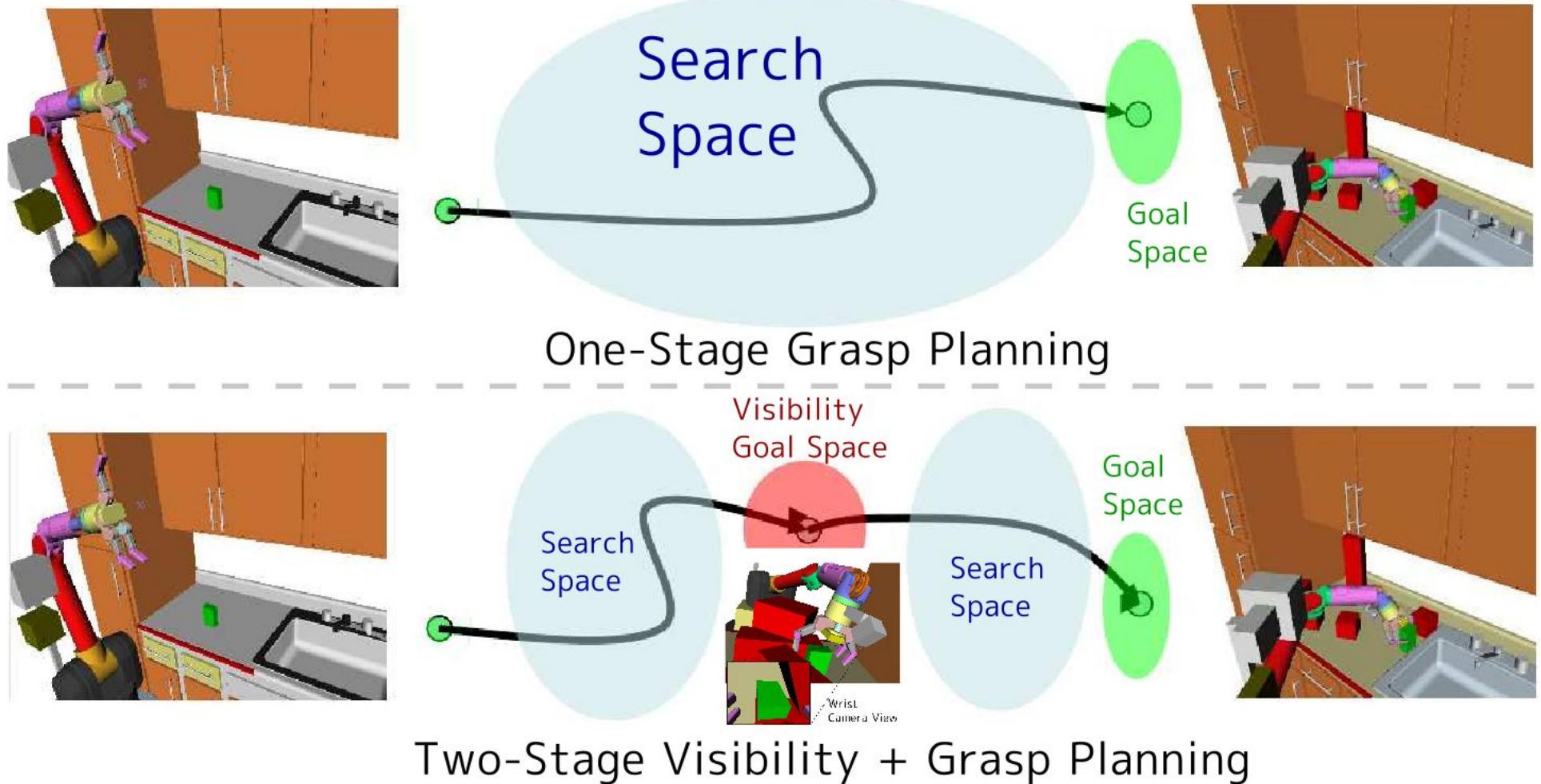
# 物体の視認性を目標にする

[Diankov, Kanade, Kuffner (2009)]

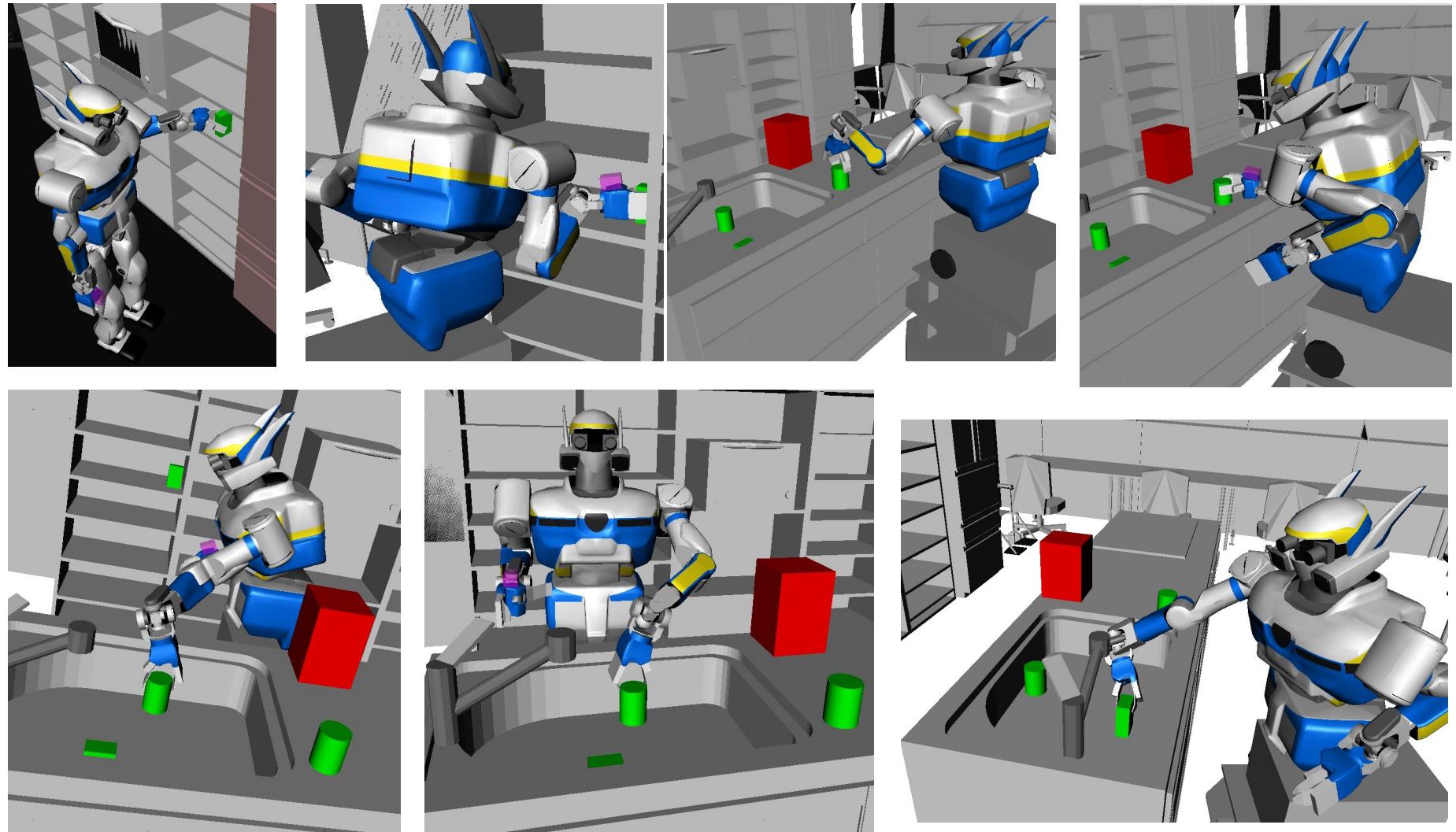
- 三つの要因
  - 視認性範囲
  - 運動学
  - 障害物



# 2段階の効率



# 胴体+アーム+把持のサンプリング



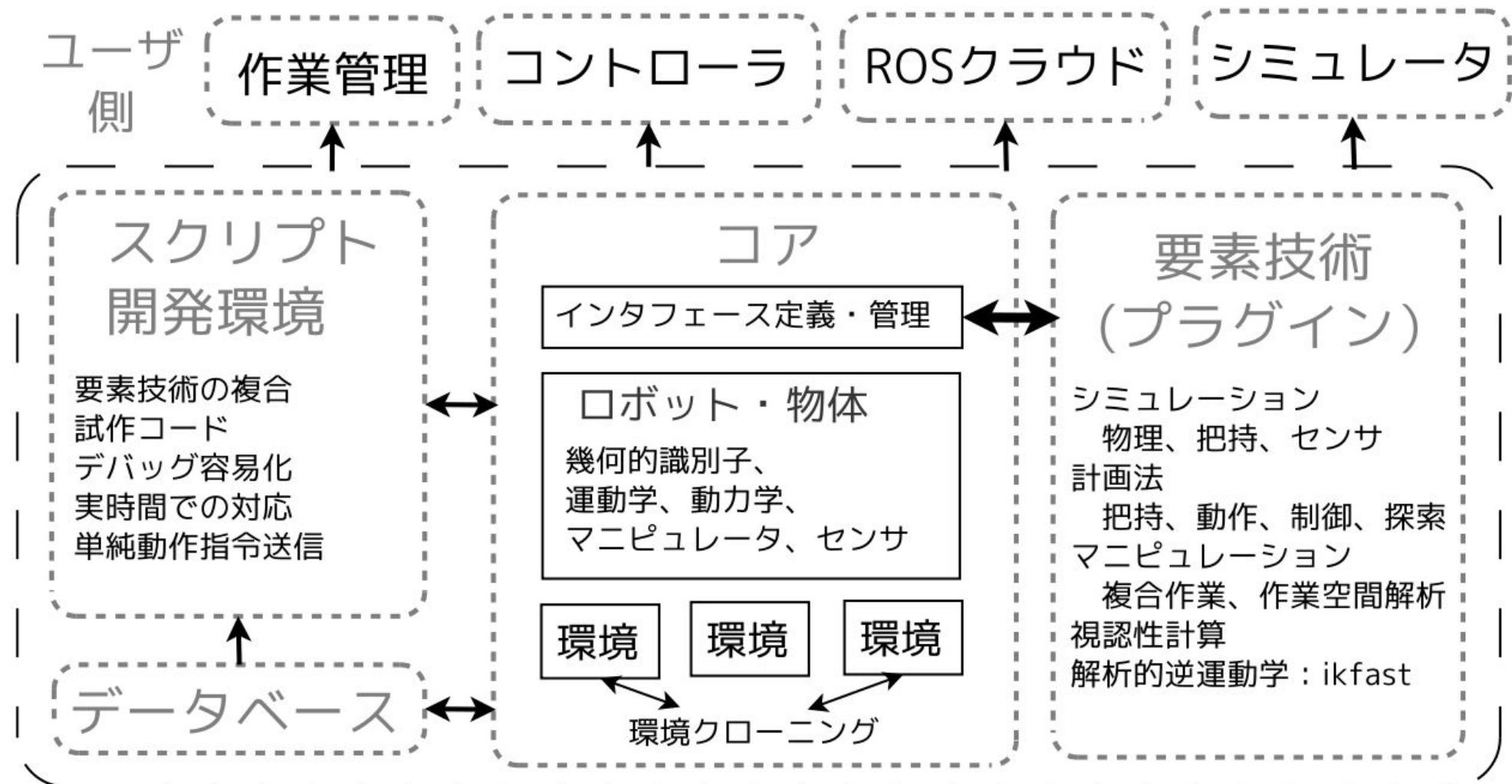
# 移動マニピュレーション動作計画



# 不明な物体の把持動作



# OpenRAVEアーキテクチャー



# Environment

- 役割
  - 物体・ロボットの生成、ロード、破壊
  - 衝突関数
  - 物理ライブラリとの同期
  - 表示ライブラリとの同期
  - クローン
  - メッシュの抽出

# Environmentロック

- 環境更新管理
  - ロックを持ったら環境が変わらない
- マルチスレッド対応
- ユーザ向けのRecursiveロック
  - スコープを気にせずにロック可能

# Environment クローン

- 新しいコピー
  - 速い
  - コピー対象物の調整
  - 静的なデータは参照される
- 実際のロボットデモ
  - 一つの環境：実世界との動機
  - 動作計画用のクローン
    - 他のユーザに干渉せずに長くロック可能

# Kinematics Bodyインタフェース

- リンクと関節のリスト
  - 階層ではない！
  - 関節の値はリンクの一姿勢から計算されている
  - 一部の構造で動作計画可能
  - 構造への探索
- 運動学・動力学
- ヤコビアンの計算
- 自己干渉
- 物体との接続
  - マニピュレーションへ必要不可欠

# Robotインターフェース

- マニピュレーター
  - Base and End Links forms arm sub-chain - IK
  - List of gripper joints - **grasping**
  - check collision or query links
    - gripper only
    - Independent links of the manipulator
- アクティブ関節
- 物体との剛体接地
- センサー設置

# 逆運動学のインターフェース

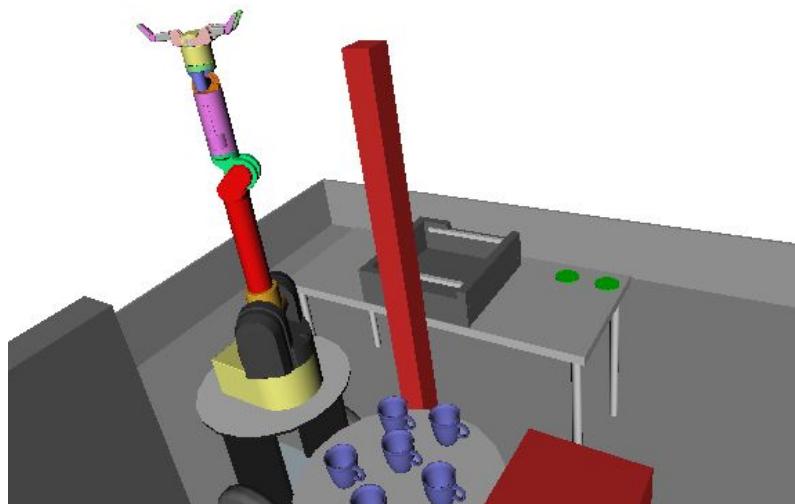
- 10個のIK種類
- 追加機能
  - 環境障害物
  - 自己干渉
  - 関節リミット
  - プランナー用

# OpenRAVE Python Scripting

[http://openrave.org/en/main/getting\\_started.html#getting-started](http://openrave.org/en/main/getting_started.html#getting-started)

# 環境のロード

```
from openravepy import *
env = Environment() # create openrave environment
env.SetViewer('qtcoin') # attach viewer (optional)
env.Load('data/lab1.env.xml') # load a simple scene
robot = env.GetRobots()[0] # get the first robot
print "Robot ",robot.GetName()," has ",robot.GetDOF()," joints with values:\n",robot.GetJointValues()
robot.SetJointValues([0.5],[0]) # set joint 0 to value 0.5
T = robot.GetLinks()[1].GetTransform() # get the transform of link 1
print "The transformation of link 1 is:\n",T
env.Destroy() # explicitly destroy the environment once done with it
```



# BiRRTの利用

```
from openravepy import *
env = Environment() # create the environment
env.Load('data/lab1.env.xml') # load a scene
env.SetViewer('qtcoin') # start the viewer
robot = env.GetRobots()[0] # get the first robot
manipprob = interfaces.BaseManipulation(robot) # create the interface for basic manipulation programs
res = manipprob.MoveManipulator(goal=[-0.75,1.24,-0.064,2.33,-1.16,-1.548,1.19]) # call motion planner
robot.WaitForController(0) # wait
env.Destroy()
```

# 環境ロックの掛け方

- `with`スコープ

```
env = Environment()  
# initialization code  
with env:  
    # environment is now locked  
    env.CheckCollision(...)
```

- ロボットの状態保存とロック

```
with robot:  
    robot.SetTransform(newtrans)  
    robot.SetActiveDOFs(...)  
    # do work  
  
# robot now has its previous state restored
```

# openrave.py program

- Jump into ipython after loading environment:

`openrave.py -i data/lab1.env.xml`

- Set default collision checkers/physics

`openrave.py --collision=pqp --viewer=qtcoin --physics=ode data/lab1.env.xml`

- サンプルの実行

`openrave.py --example grasplanning`

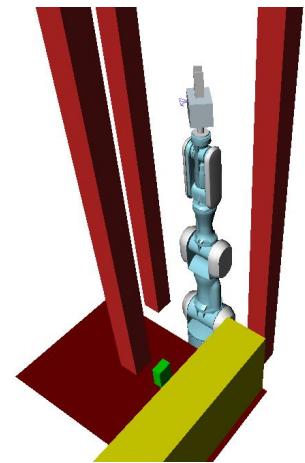
- Execute database generator: solve analytic IK, create C++ file

`openrave.py --database inversekinematics --robot=robots/pa10.robot.xml`

# openravepy特徴

- C++関数のone-to-one対応
- C++のエラーが起こればpythonに戻される
  - 再起動せずにエラー復帰
- 新しいプラグインのロード

# PA10で把持計画



```
openrave.py --example visibilityplanning
```

```
openrave.py data/pa10grasp2.env.xml
```

```
openrave.py --database grasping --robot=robots/pa10schunk.robot.xml  
--target=data/box_frootloops.kinbody.xml
```

```
openrave.py --database visiblitymodel --robot=robots/pa10schunk.robot.xml  
--target=data/box_frootloops.kinbody.xml
```

# PA10で把持計画

- 環境のロードと準備

```
# OpenRAVEをロード
from openravepy import *
env=Environment()
env.SetViewer('qtcoin')
env.Load('data/pa10grasp2.env.xml')

# モデルの作成
robot=env.GetRobots()[0]
manip=robot.GetActiveManipulator()
target=env.GetKinBody('frootloops')
gmodel=databases.grasping.GraspingModel(robot, target)
if not gmodel.load():
    gmodel.autogenerate()

ikmodel=databases.inversekinematics.InverseKinematicsModel(robot, iktype=IkParameterization.Type.Transform6D)
if not ikmodel.load():
    ikmodel.autogenerate()

basemanip = interfaces.BaseManipulation(robot)
taskmanip = interfaces.TaskManipulation(robot)
```

```
# 把持計画
taskmanip.ReleaseFingers()
validgrasps = gmodel.computeValidGrasps()[0]
gmodel.showgrasp(validgrasps[0])
for g in validgrasps: gmodel.showgrasp(g)

with env:
    for i,g in enumerate(validgrasps):
        with robot:
            Tgrasp = gmodel.getGlobalGraspTransform(g,collisionfree=True)
            sol = manip.FindIKSolution(Tgrasp,IkFilterOptions.CheckEnvCollisions)
            robot.SetDOFValues(sol,manip.GetArmIndices())
            if env.CheckCollision(robot):
                print i
            env.UpdatePublishedBodies()
            raw_input('pause')

# プランニング
basemanip.MoveToHandPosition(matrices=[gmodel.getGlobalGraspTransform(validgrasps[0],collisionfree=True)])
```

```
# 視認性
vmodel=databases.visibilitymodel.VisibilityModel(robot,target)
if not vmodel.load():
    vmodel.autogenerate()

validjoints = vmodel.computeValidTransform()
with robot:
    sol,index = validjoints[0]
    robot.SetJointValues(sol,manip.GetArmIndices())
env.UpdatePublishedBodies()
raw_input('pause')
```

# 第4宿題

- ・箱の6平面をハンドで接触する経路
  - ・左と右ハンドで経路を作る
- ・環境
  - ・障害物
  - ・箱の位置姿勢
- ・宿題演習セッション
  - ・月曜日18時から

