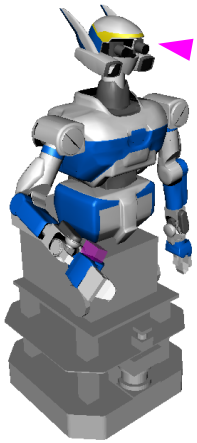
OpenRAVE

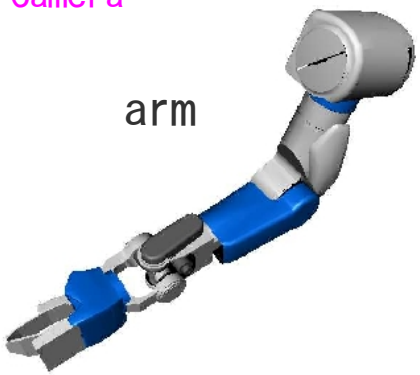
出杏光　魯仙
デアンコウ　ロセン

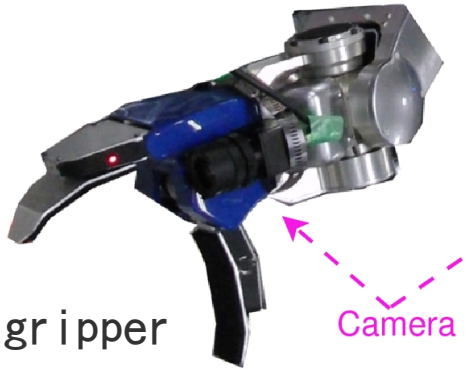
講義の内容

- 今回:
  - 動作アルゴリズムの紹介
  - OpenRAVEのロボット動作計画アルゴリズム
  - OpenRAVEアーキテクチャ
- 次回:
  - OpenRAVEコードとスクリプトのデモ

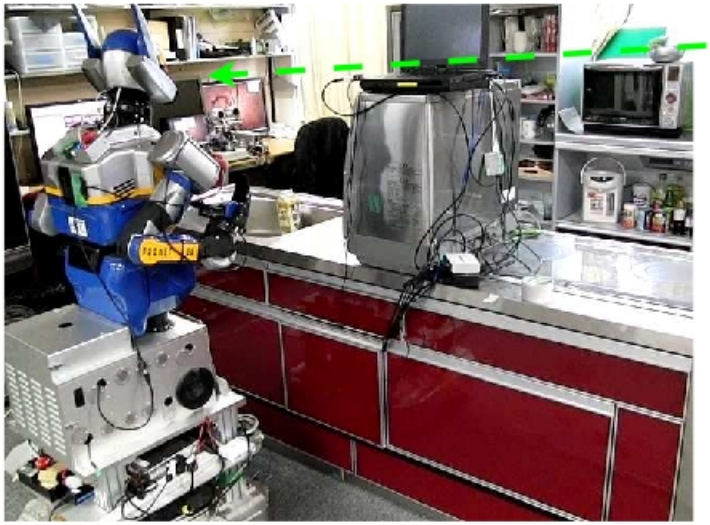
ロボットシステムの例





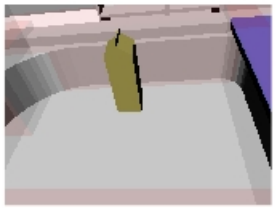


Target object

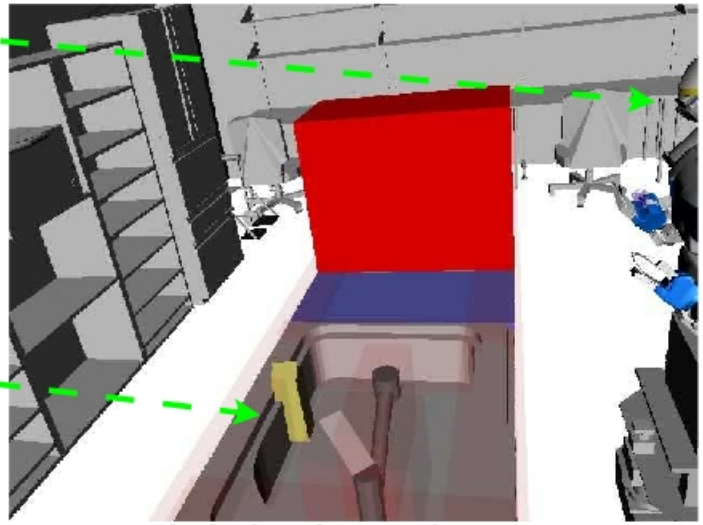


Initial Real Scene

Head Camera

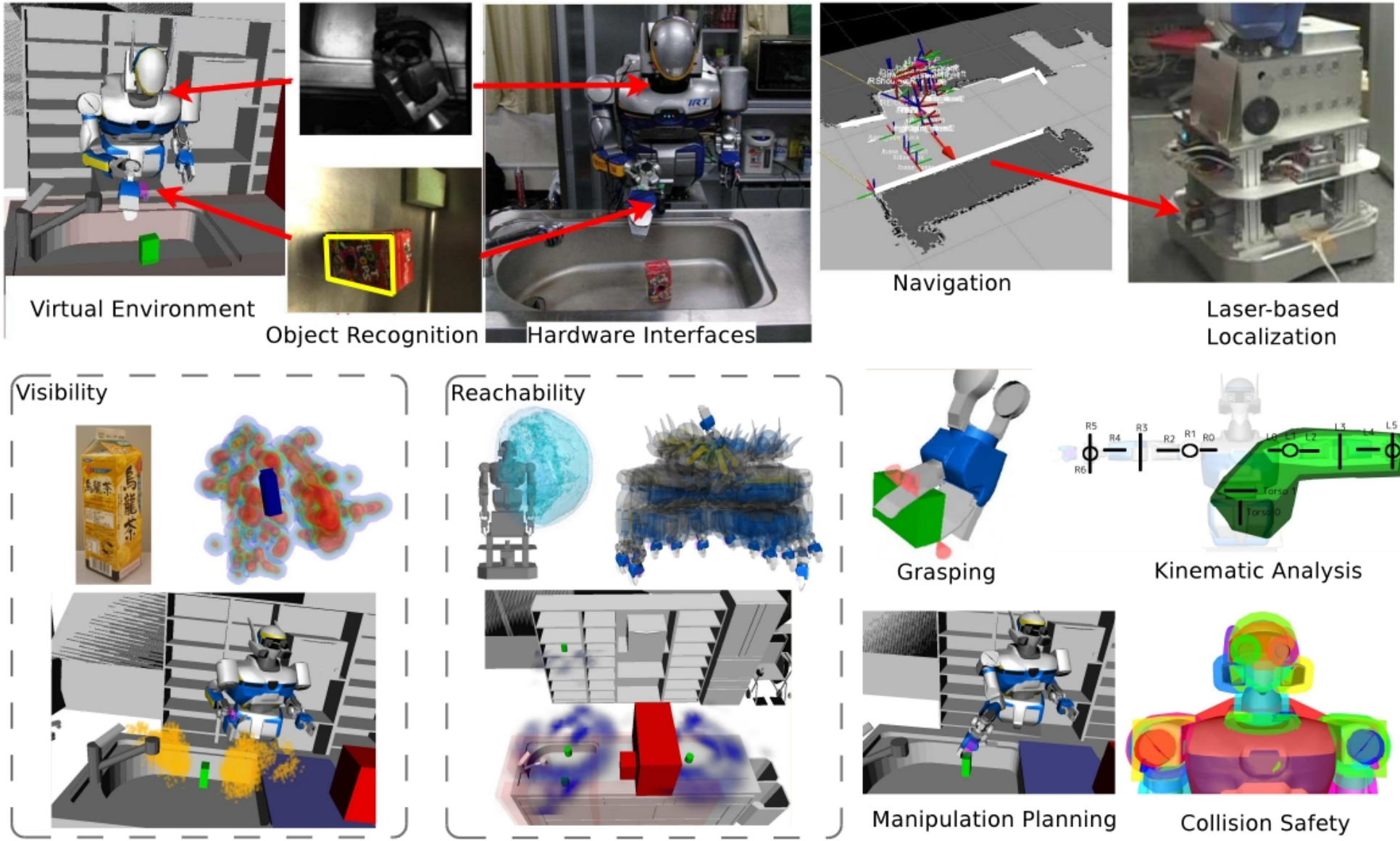


Half of Object is
Colliding with Sink



Initial Virtual Scene

ロボットシステムのコンポーネント



OpenRAVEって何？

ロボット動作と3次元環境の解析における
再利用可能な適合性の高い

計算エンジン

OpenRAVEの第一目的

生産工程における
ロボット導入の完全自動化

OpenRAVEの重要な点

- オープンソース：LGPLとApache V2.0ライセンス
- 6年間開発
- Linux、Windows、Mac OSXに対応
- 180論文以上に参考されている
- 世界で300人以上の技術者に使用
- 200種以上のロボットが使われた実績
- 50以上すぐに使えるサンプル
- ドキュメントの自動生成
- 自動テストシステムの使用

動作計画の前提条件

- 障害物
  - 3次元センサ
  - CADモデル
- 対象物体の位置姿勢と形状
  - 認識
- ロボットの位置姿勢
  - 自己位置同定
- ロボットの動作モデル
  - 制御・コントローラ
  - 関節のリミット、最大速度、最大加速度
- ロボットの形状
- はっきりした目的
  - タスク仕様
  - 上流の知能モジュール

OpenRAVEで利用可能な技術

複雑な運動学のモデリング

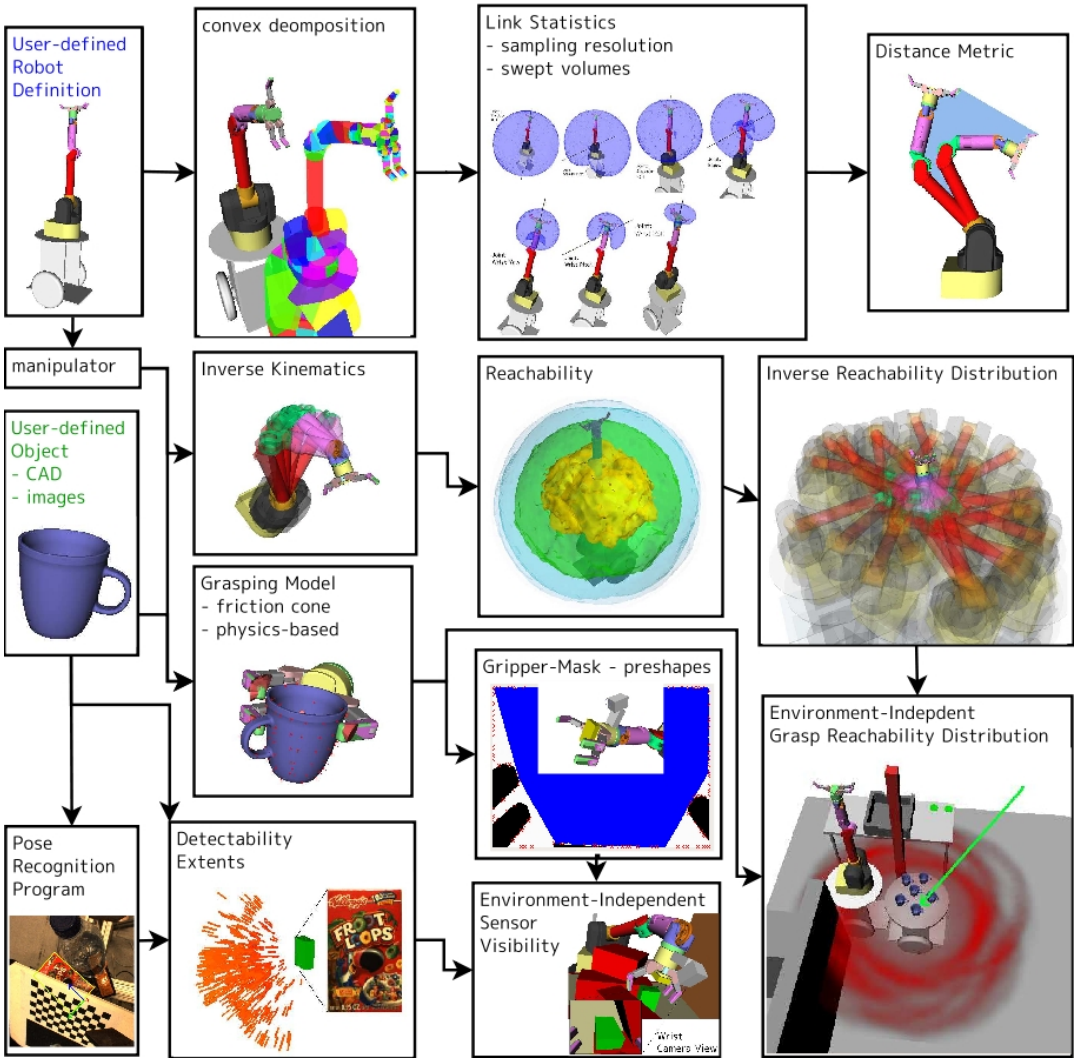
Bobcat:
8 joints, 1 DOF

Welding Tooltip
4 joints, 1 DOF

動作と作業目標をつなぐ
ロボット知識ベース

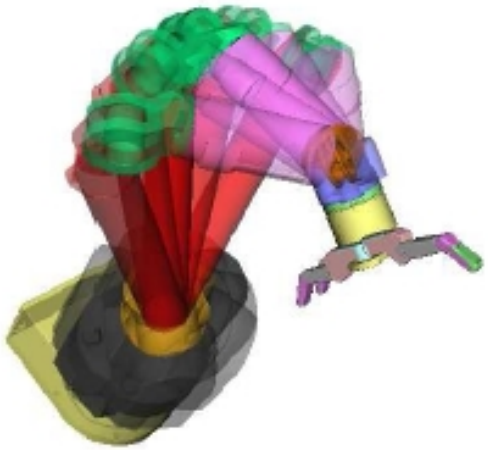
- 逆運動学
- 把持シミュレーション
- 掃引体積
- 到達範囲
- 位置分布
- 視認性

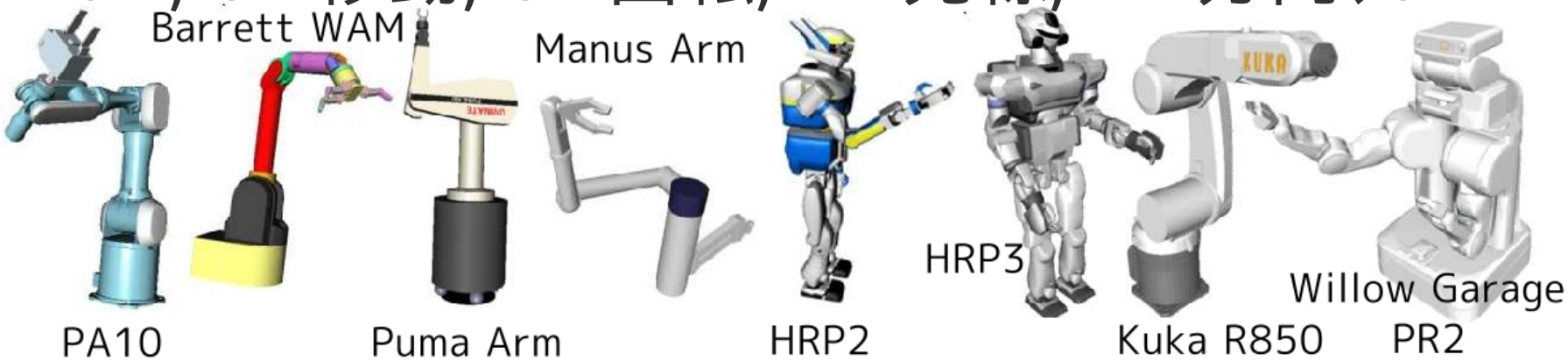
openrave.py --listdatabases



運動学の解析的な解

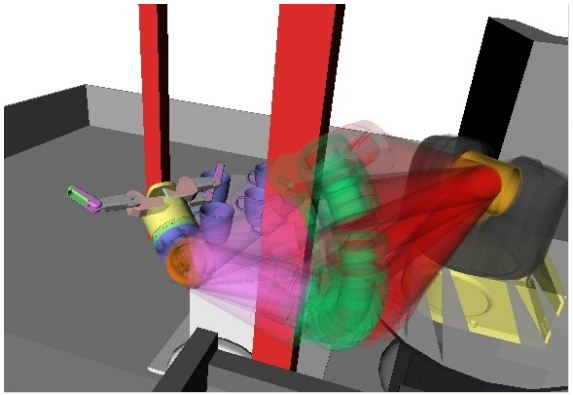
- **ikfast -** 最適化かつ安定化されたC++生成
- ゼロ除算の検出
- degenerateケースの検出
  - 軸の整列
- 全解の返し ~**6マイクロ秒**
- 6D, 3D移動, 3D回転, 4D光線, 2D方向、3D

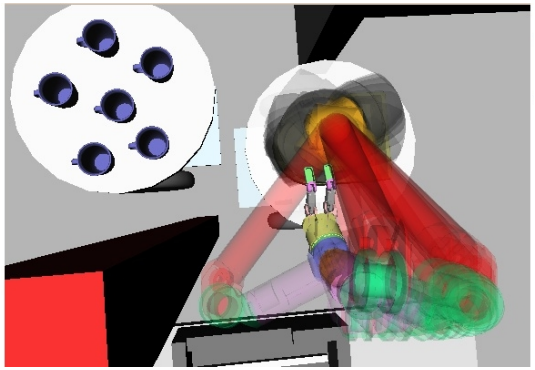


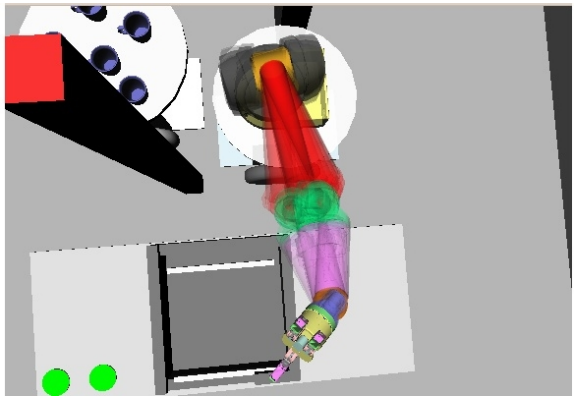


逆運動学の機能

- 6関節を予めに選択
- 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



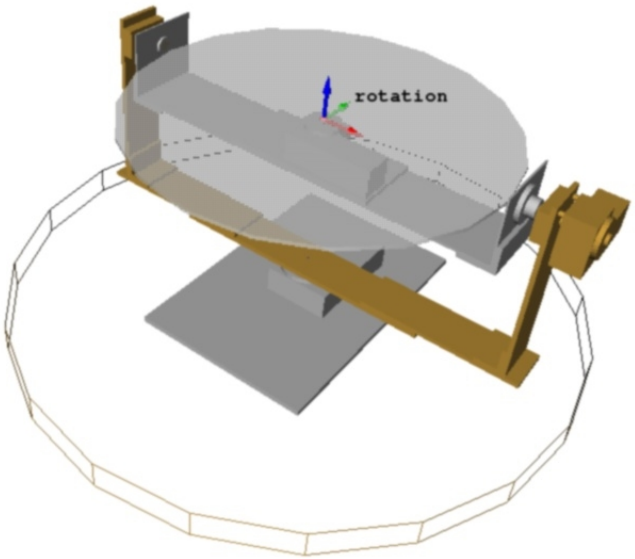




3D回転

- 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.zae
--iktype=rotation3d --iktests=100
```



3D移動

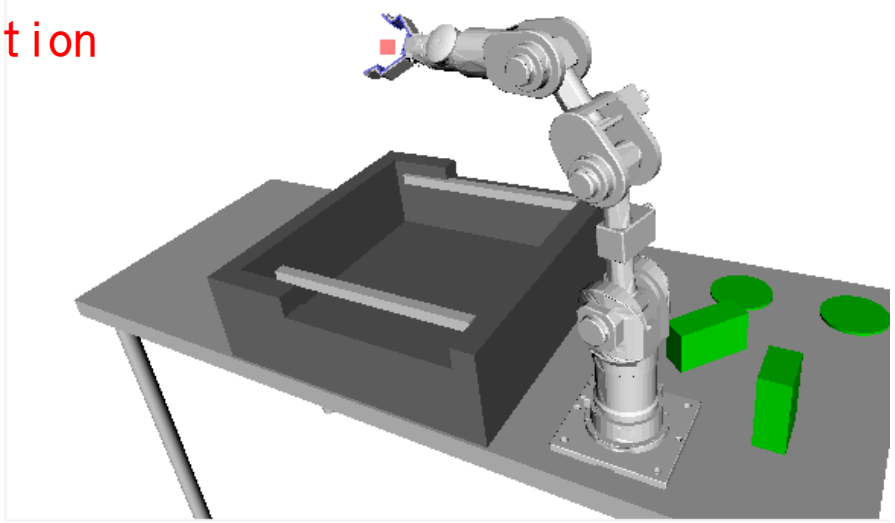
- ロボット上の3D位置を世界座標系の3D位置に合わせる

生成

```
openrave.py --database inversekinematics --robot=robots/neuronics-katana.zae
--iktype=translation3d --freejoint=j4 --freejoint=j5 --iktests=10
```

サンプリングの例

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

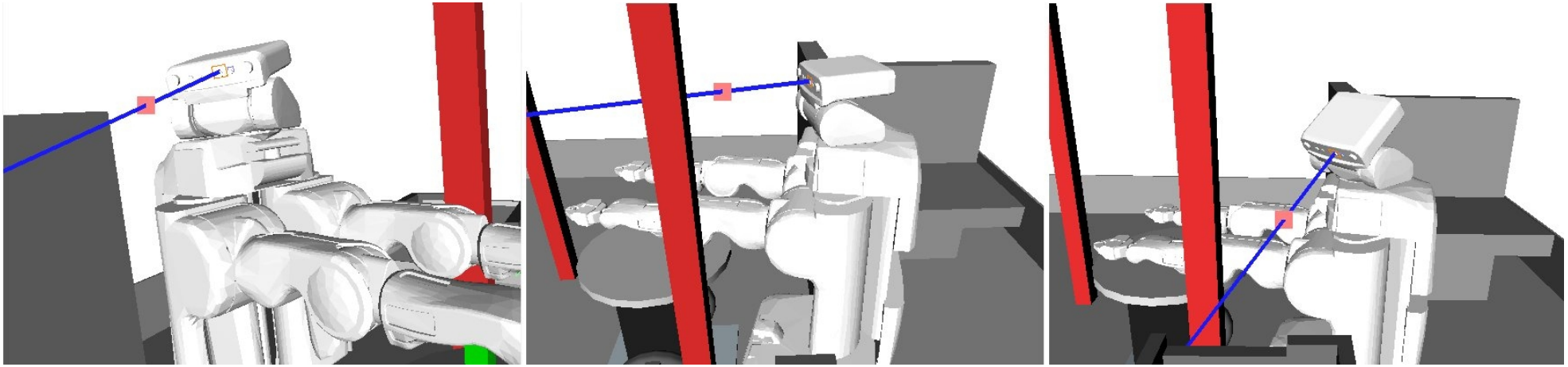


3D対象

- 対象の位置に光線ベクトルを合わせる

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

サンプリングの例
```
openrave.py --example tutorial_iklookat
```



5D IK:位置と方向

IKの生成方法

```
openrave.py --database inversekinematics
```

成功率の試験

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

新しいロボットを指定する

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

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.zae
--manipname=leftarm
```

フリー関節の選択：

```
openrave.py --database inversekinematics --robot=robots/mitsubishi-pa10.zae --
freejoint=S3
```

関節の出力：

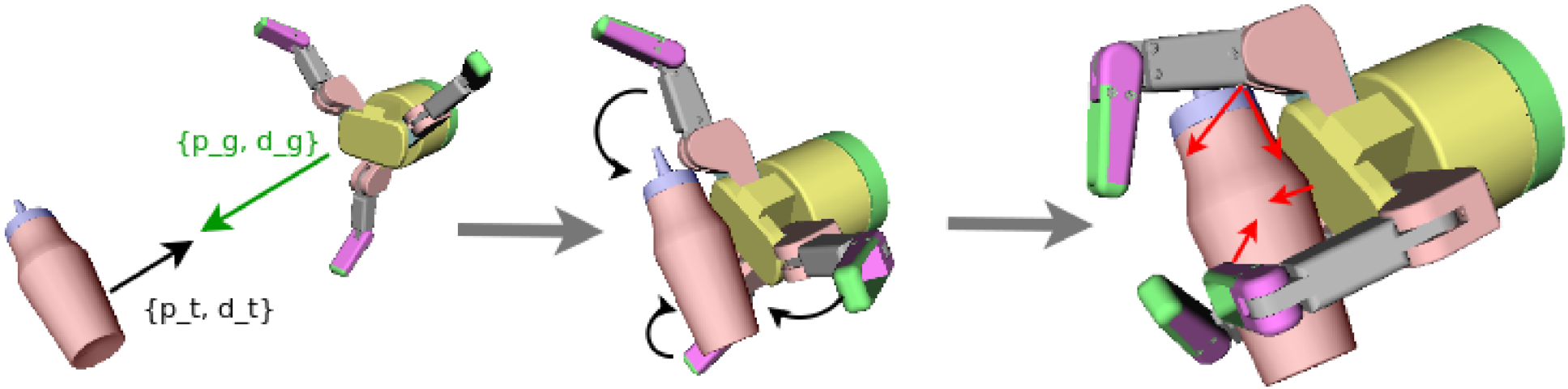
```
openrave.py -p 'print env.GetBodies()[0].GetJoints()' --robots/mitsubishi-
pa10.zae
```

ファイル名：

```
openrave.py --database inversekinematics --getfilename
```

把持シミュレーション

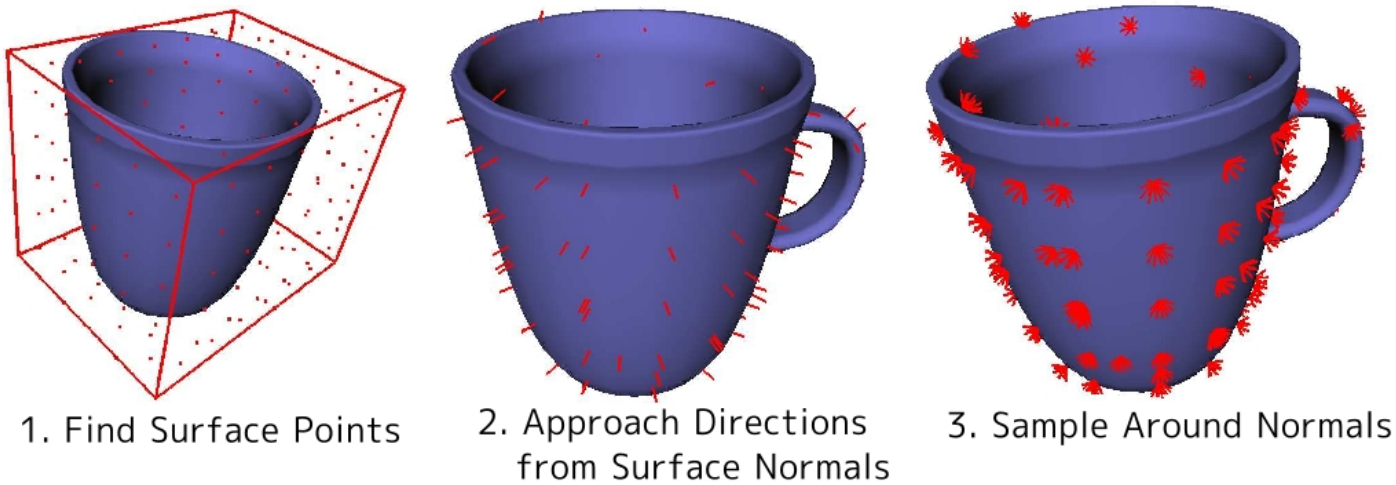
- 戦略
- 把持空間のパラメーター化
- 把持安定性測定法
- 把持セットの作成



1．対象物体に迫る　2．指を閉じる　3．安定性を計算

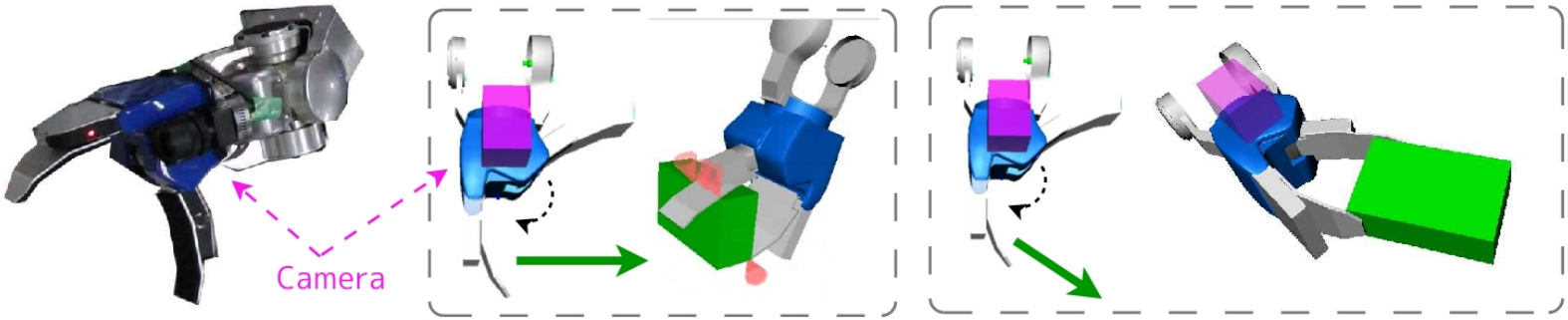
把持空間のパラメター化:8自由度

物体表面上
サンプリング
(4DOF)



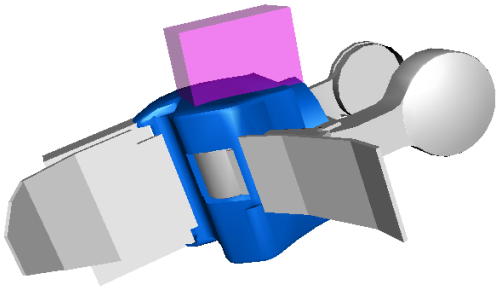
ロボット
表面上

(4DOF)



Stand-off (1DOF)

把持セット



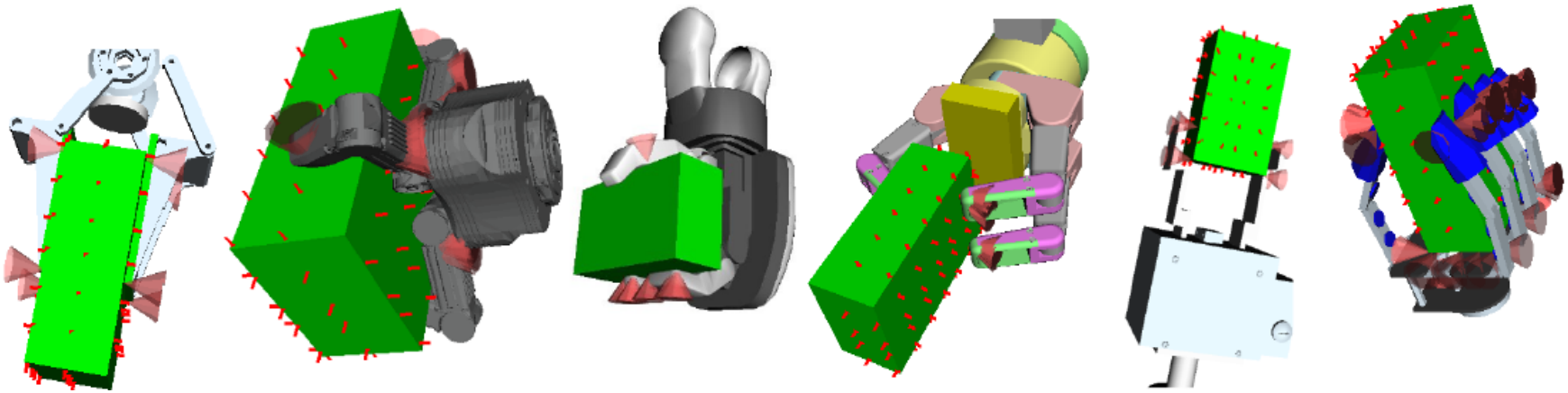
```
openrave.py --database grasping
```

把持セットの表示

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

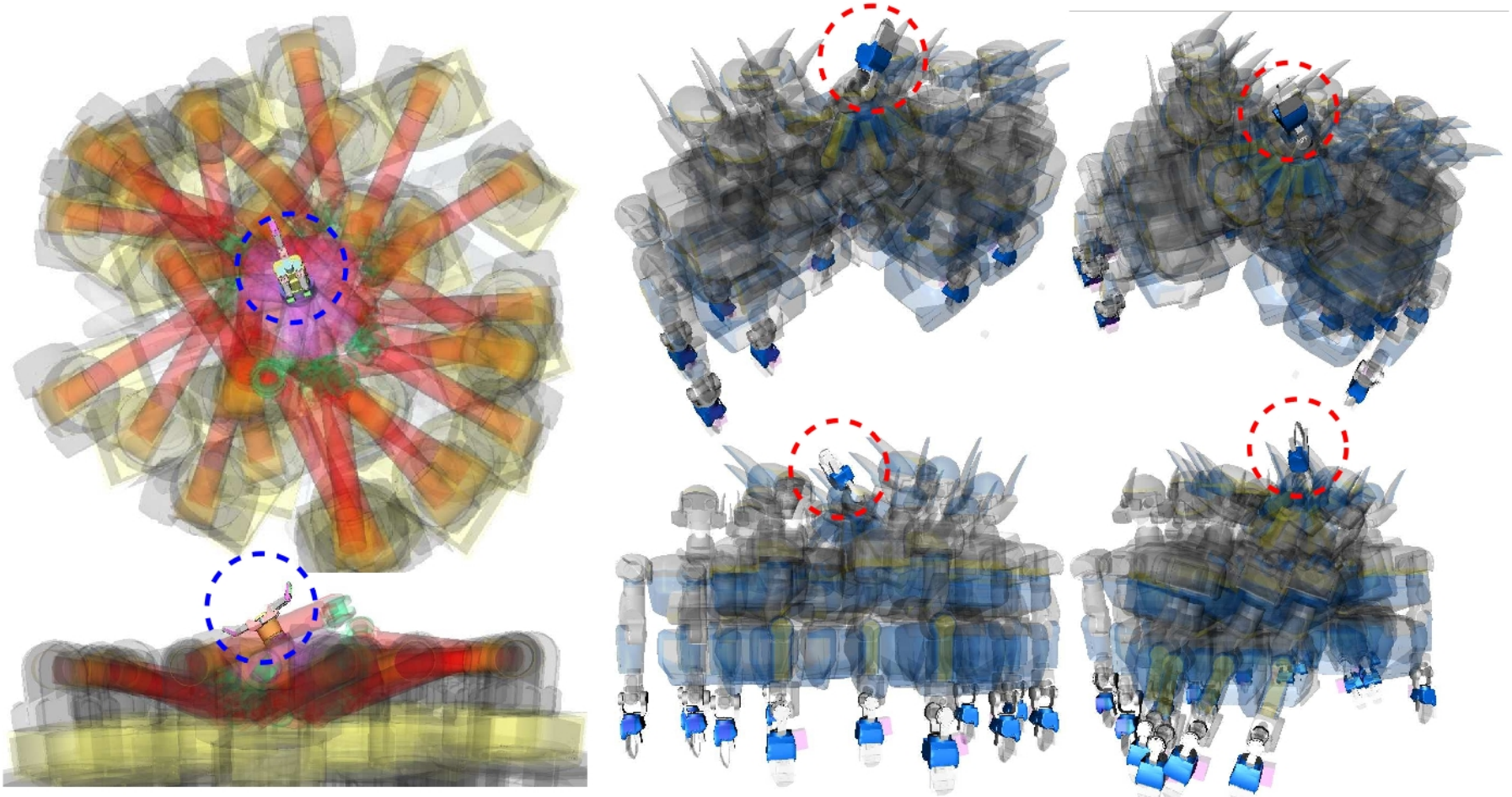
ノイズを加える

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



投影された逆運動学分布

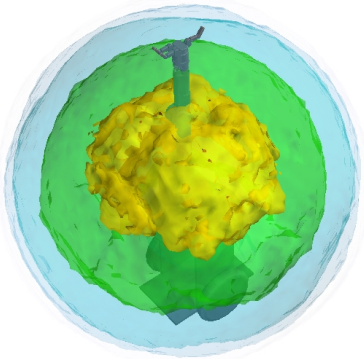
```
openrave.py --database inversereachability --robot=robots/mitsubishi-pa10.zae
```

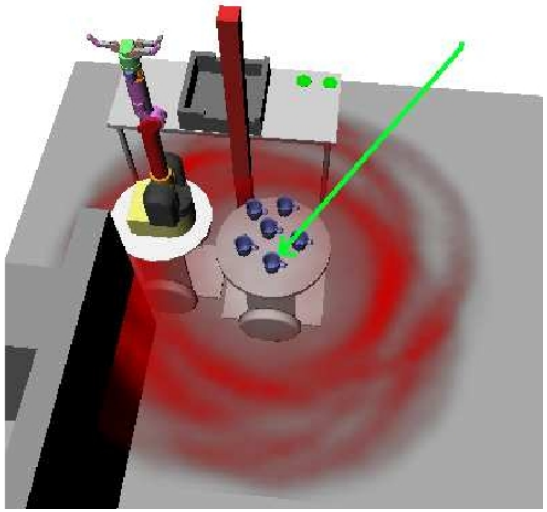


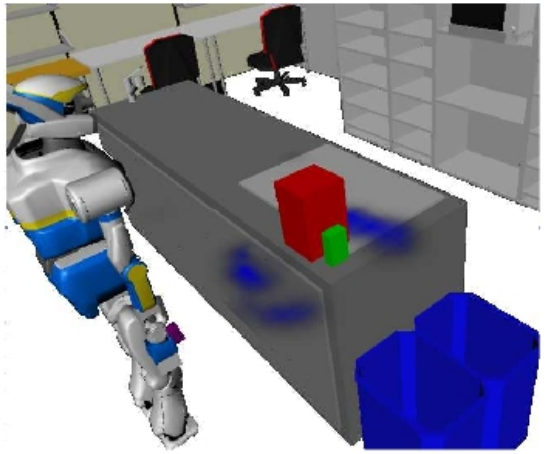
7 DOF Arm　　7 DOF Arm　　8 DOF Arm+Chest

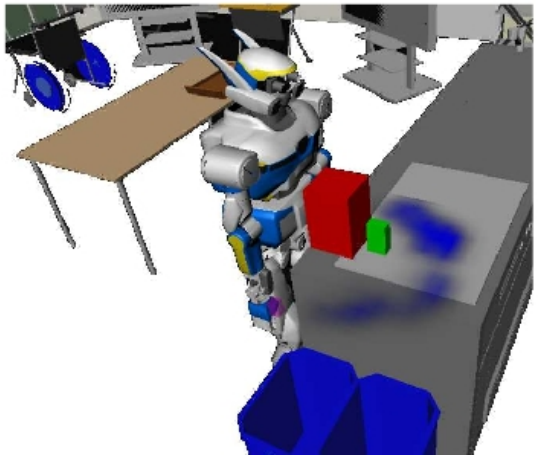
把持可能位置姿勢空間

```
openrave.py --example tutorial_inversereachability
```

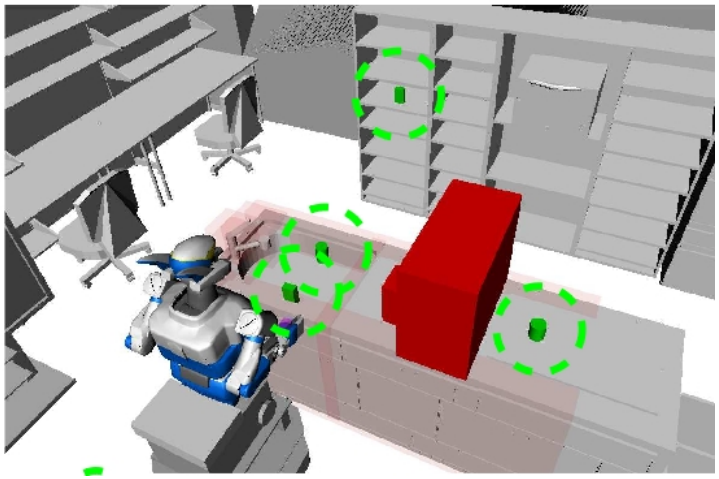








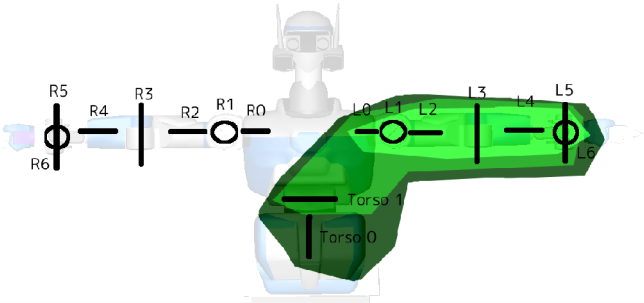
把持可能位置姿勢空間



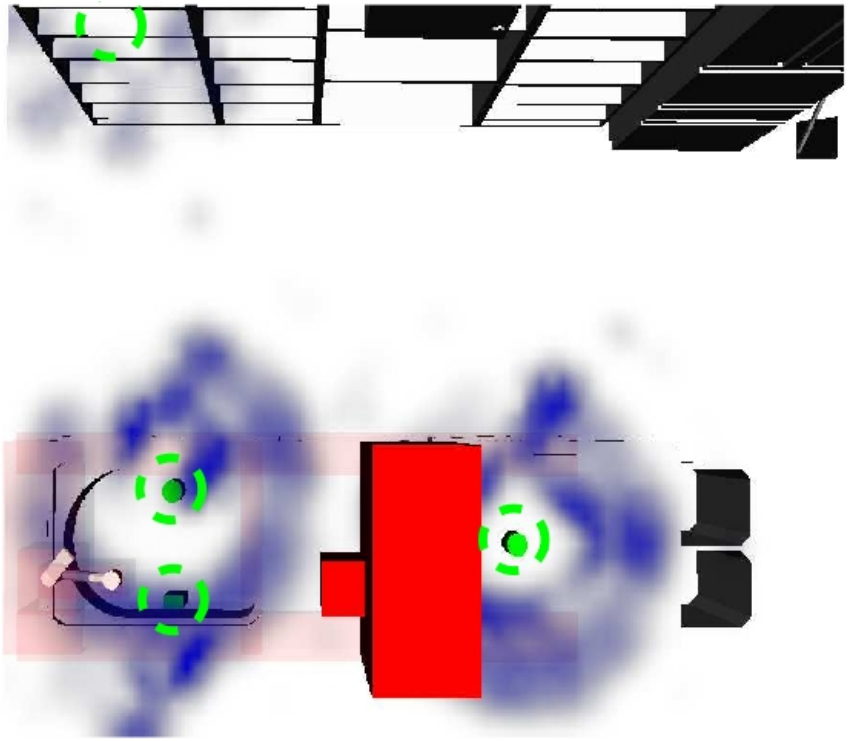
4物体

＋

6アーム



＝

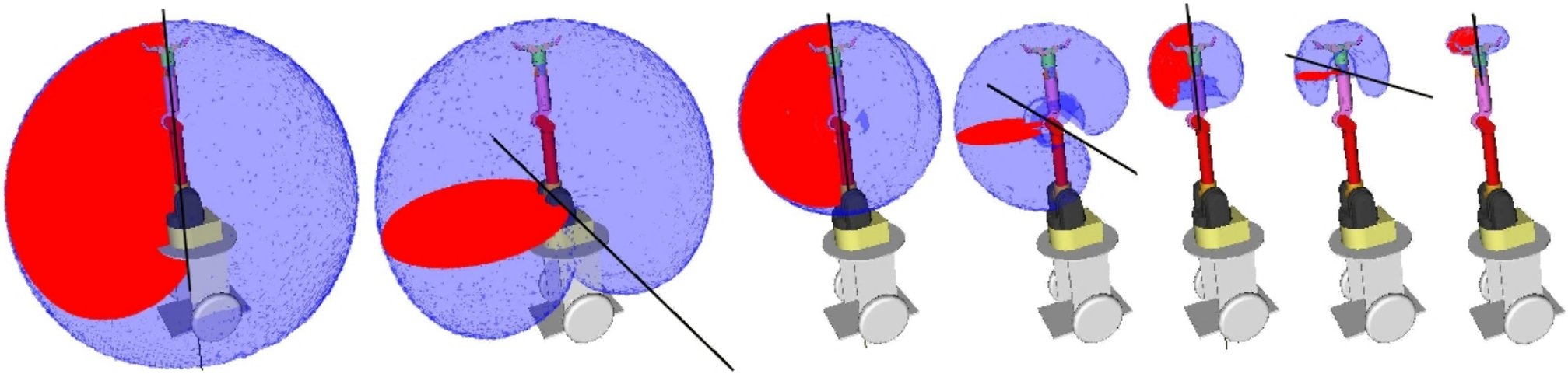


XY+回転の胴体分布

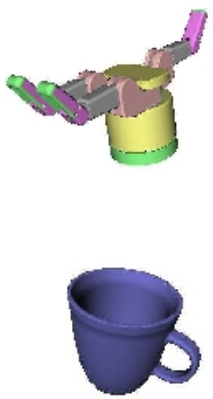
凸形状分解：Distance Metric

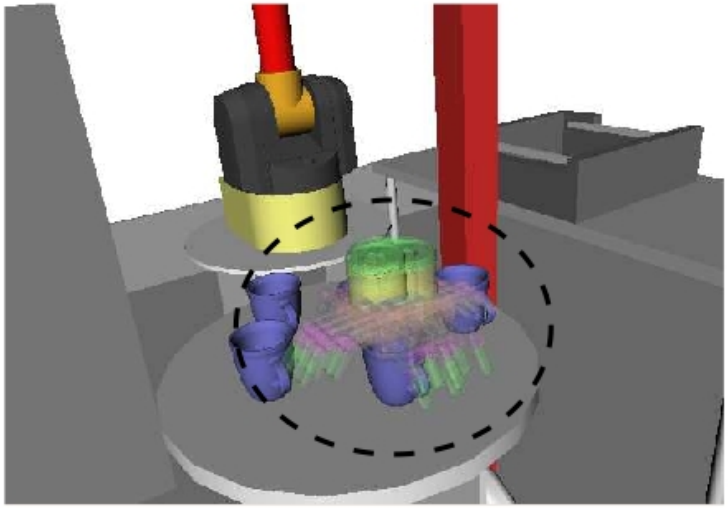
掃引体積

$$\omega_i = \left(\textsc{CrossSection}(SV_i) * \sum_{j \in DependentLinks_i} LV_j \right)^{\frac{1}{3}}$$

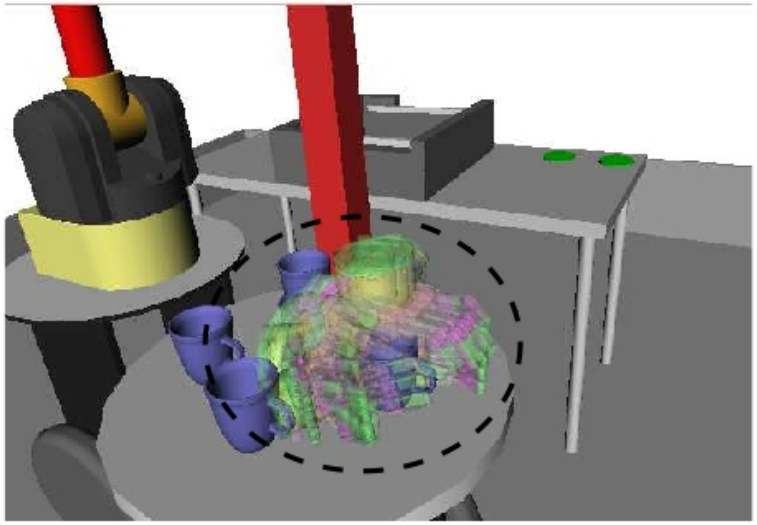


把持の選択

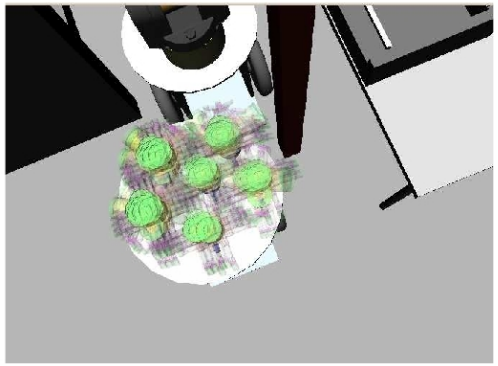


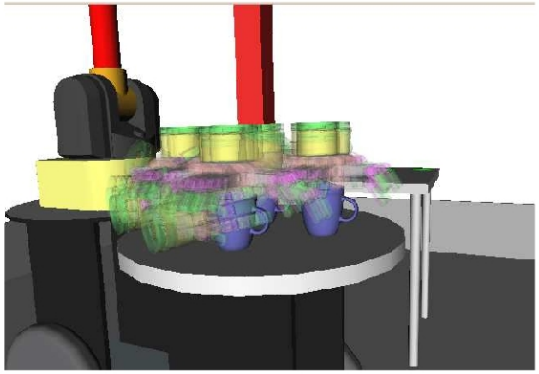


Reachable (has IK)

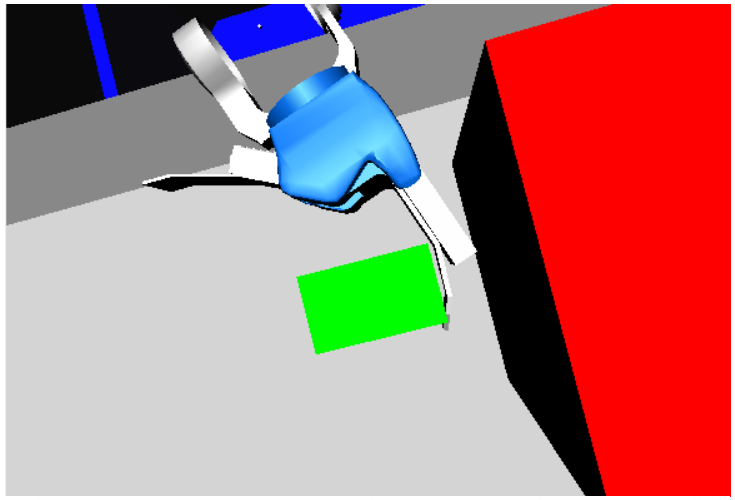


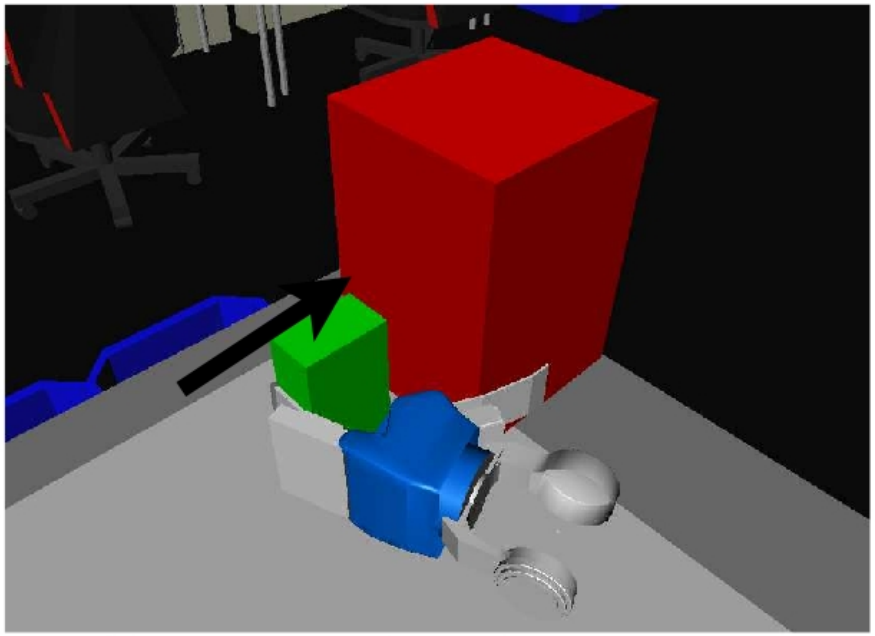
Unreachable



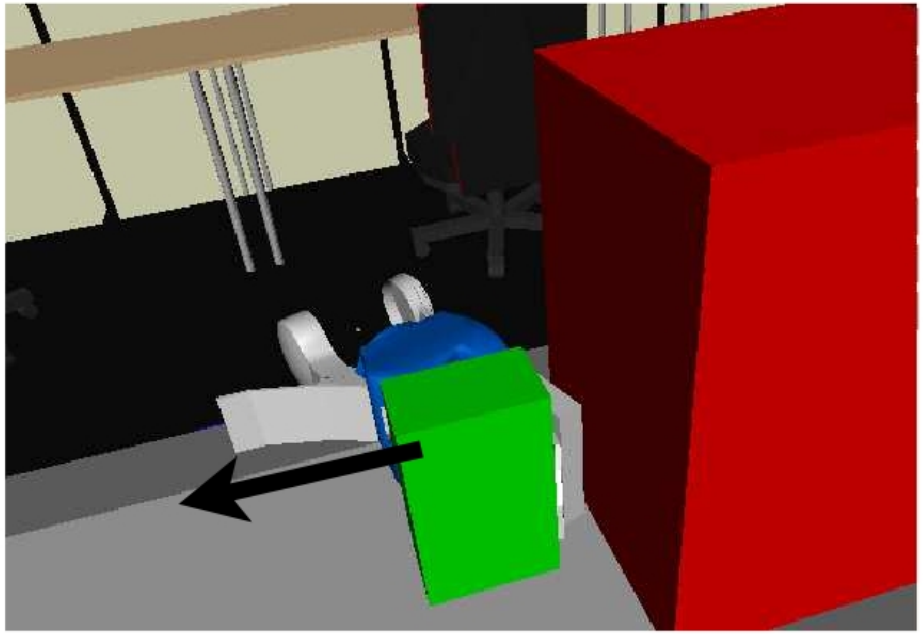


把持の検証



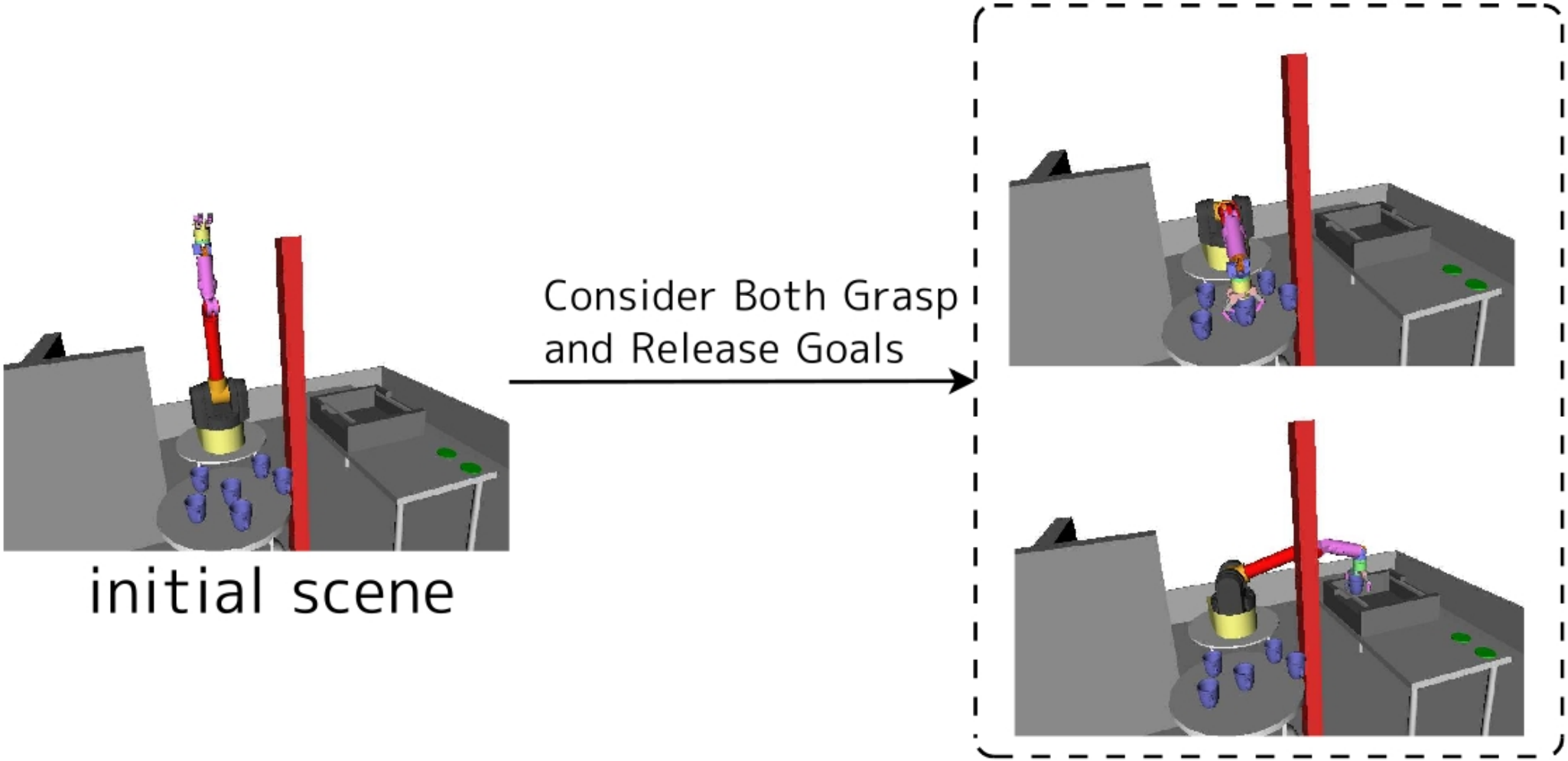


Obstacle Blocks Finger

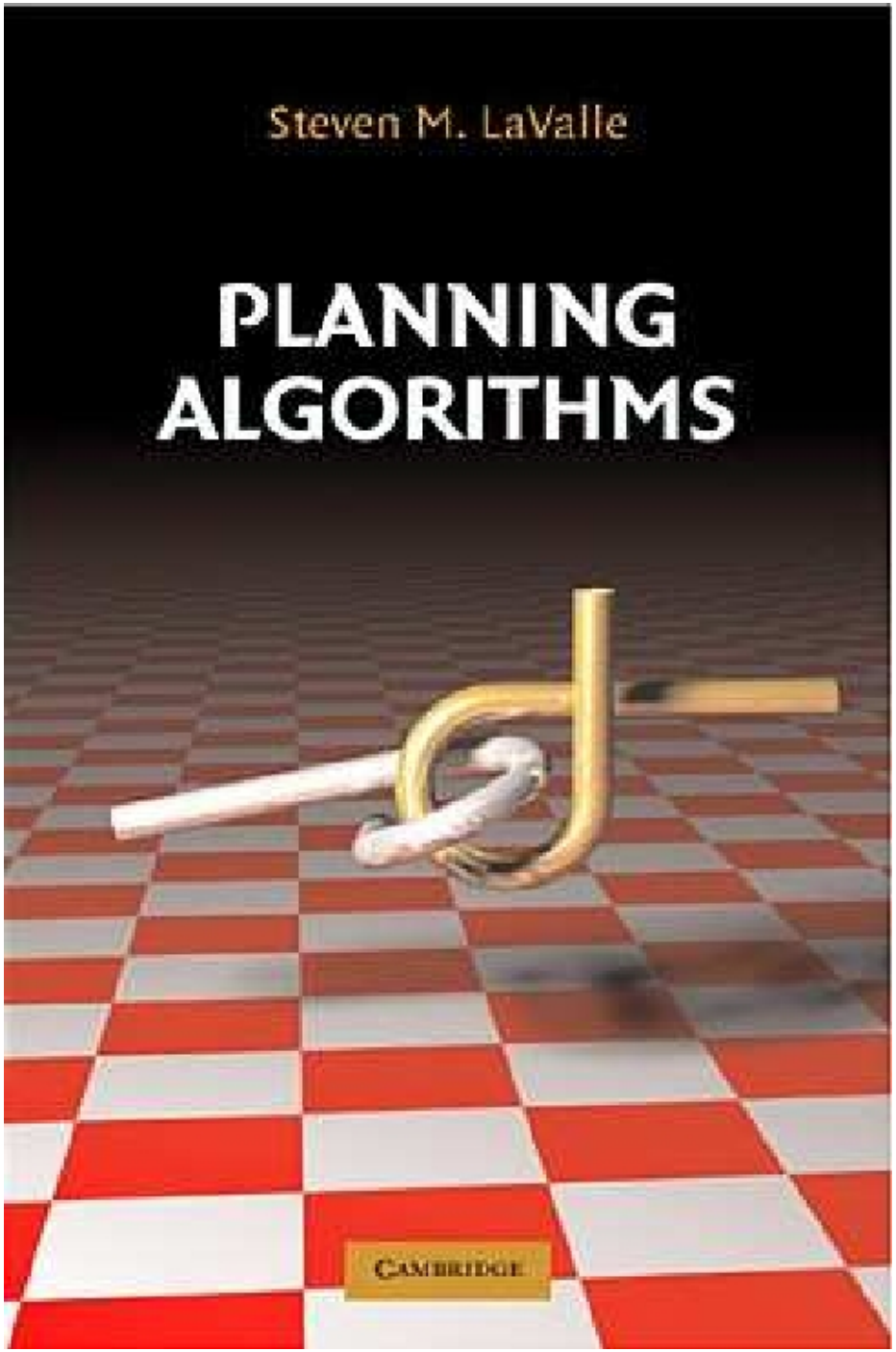


Tight Place

把持選択時に目的地も考慮する

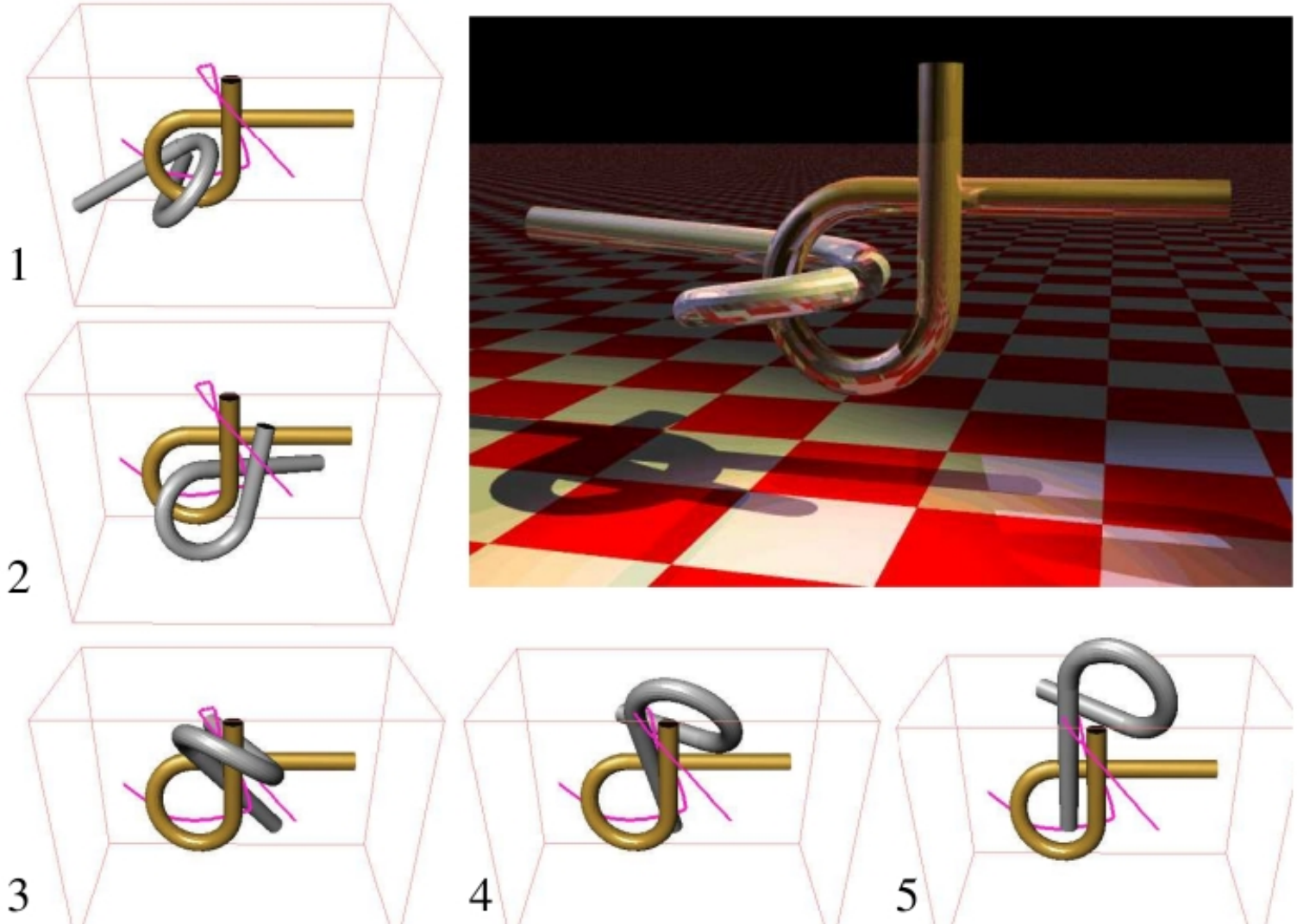


動作計画アルゴリズム紹介



計画

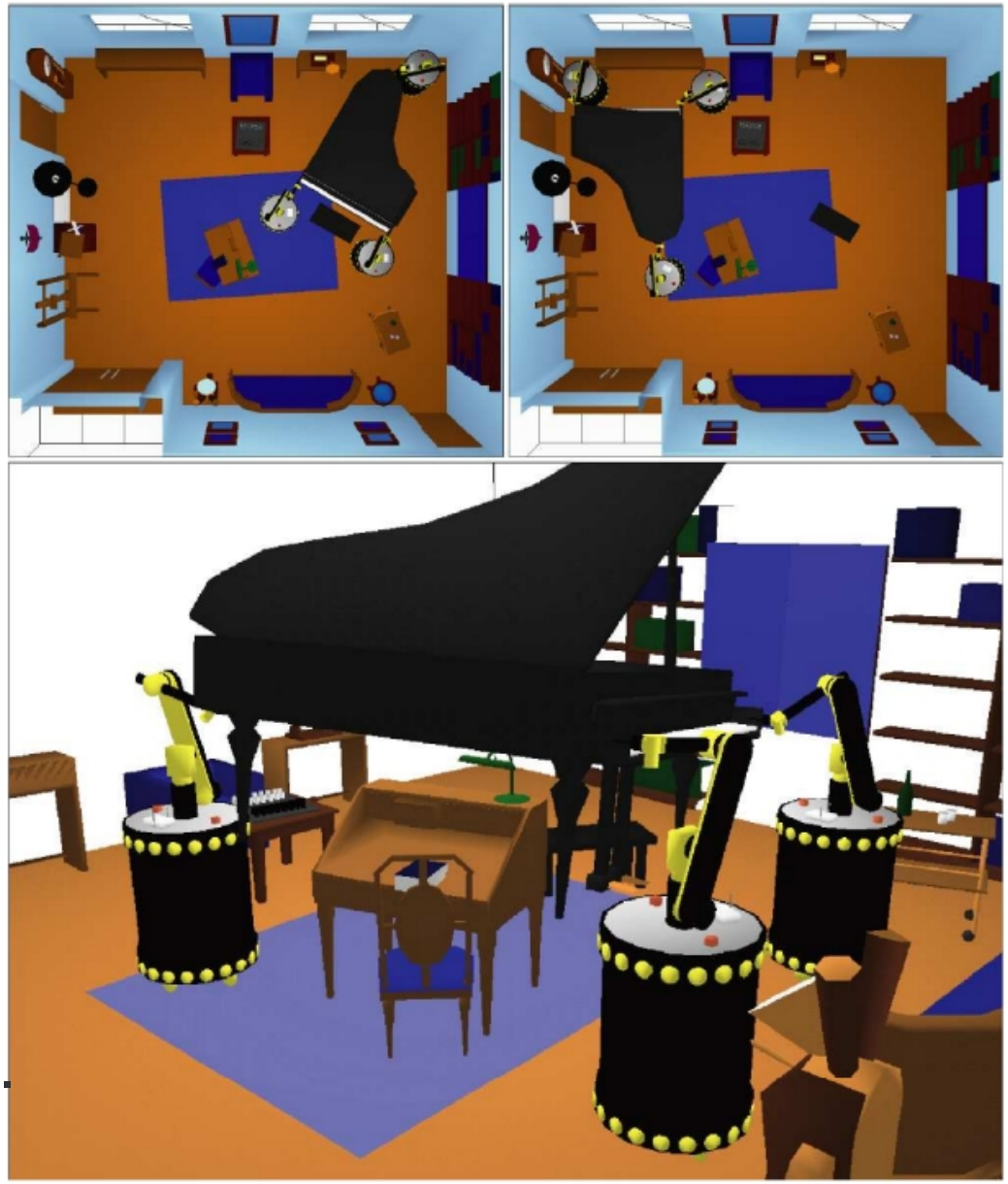
衝突せずに引き離す経路



Steven LaValle. Planning Algorithms. Cambridge University, 2006

物運び

- ロボットによって運ばれる物
  - ロボットの制約条件
  - 3D障害物
- 状態
  - 位置姿勢
  - 関節の値



Steven LaValle. Planning Algorithms. Cambridge University, 2006

A*探索:最短の距離

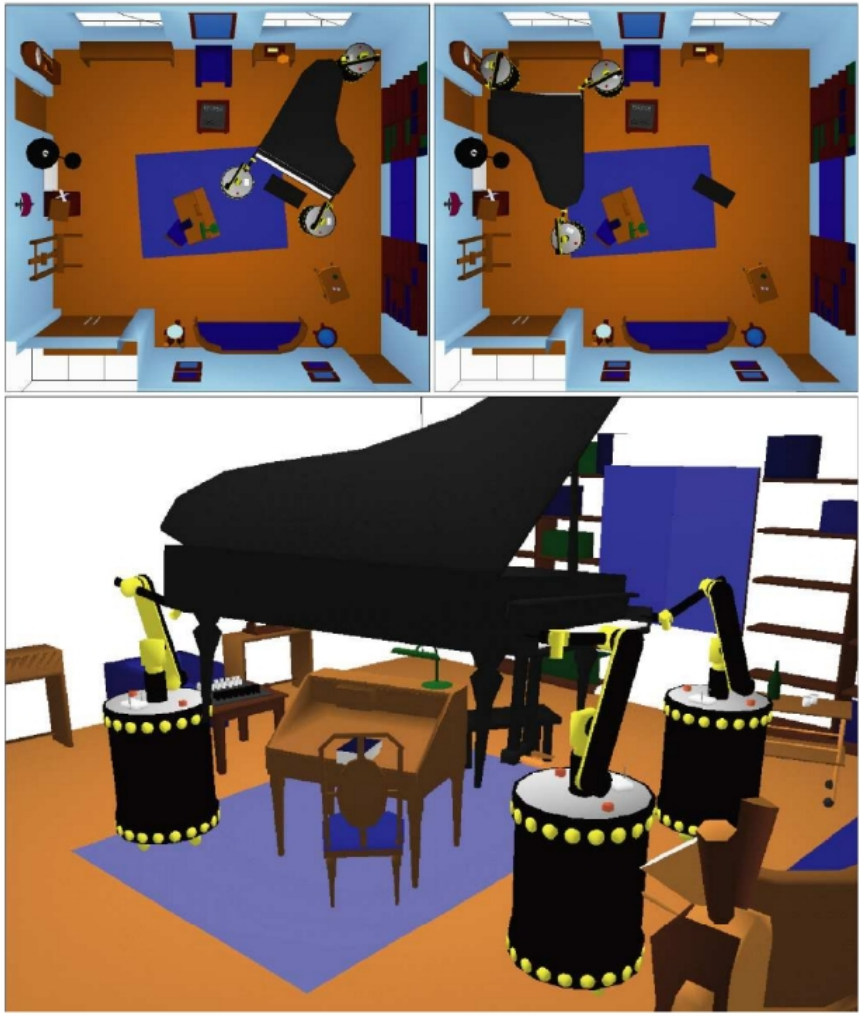
$$f(n) = g(n) + h(n)$$

- 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：$h(x) \leq d(x,y) + h(y)$

Configuration Space
コンフィグレーションスペース

- ロボットの位置姿勢
- ロボットの関節の値
- ピアノの位置姿勢?



Steven LaValle. Planning Algorithms.
Cambridge University, 2006

コンフィグレーションスペース

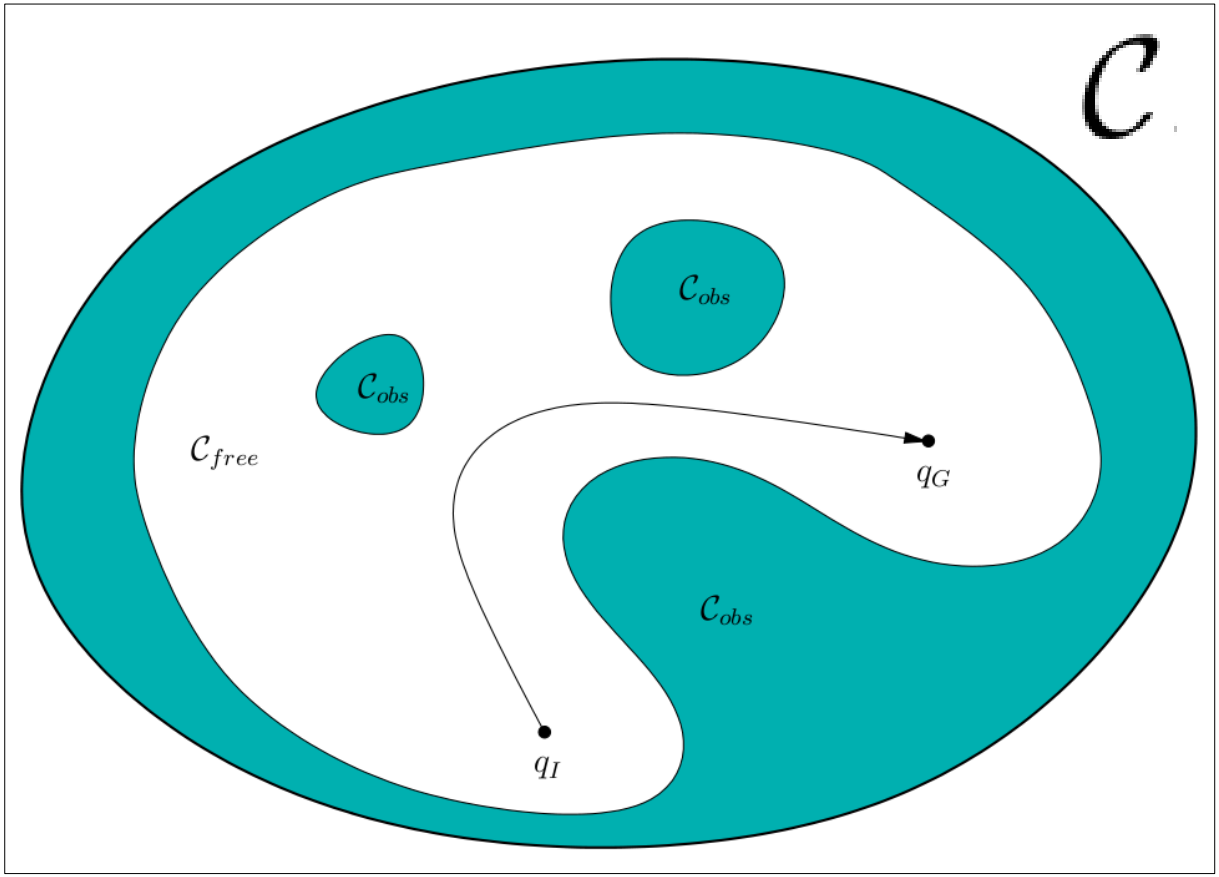
$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}$$

障害物

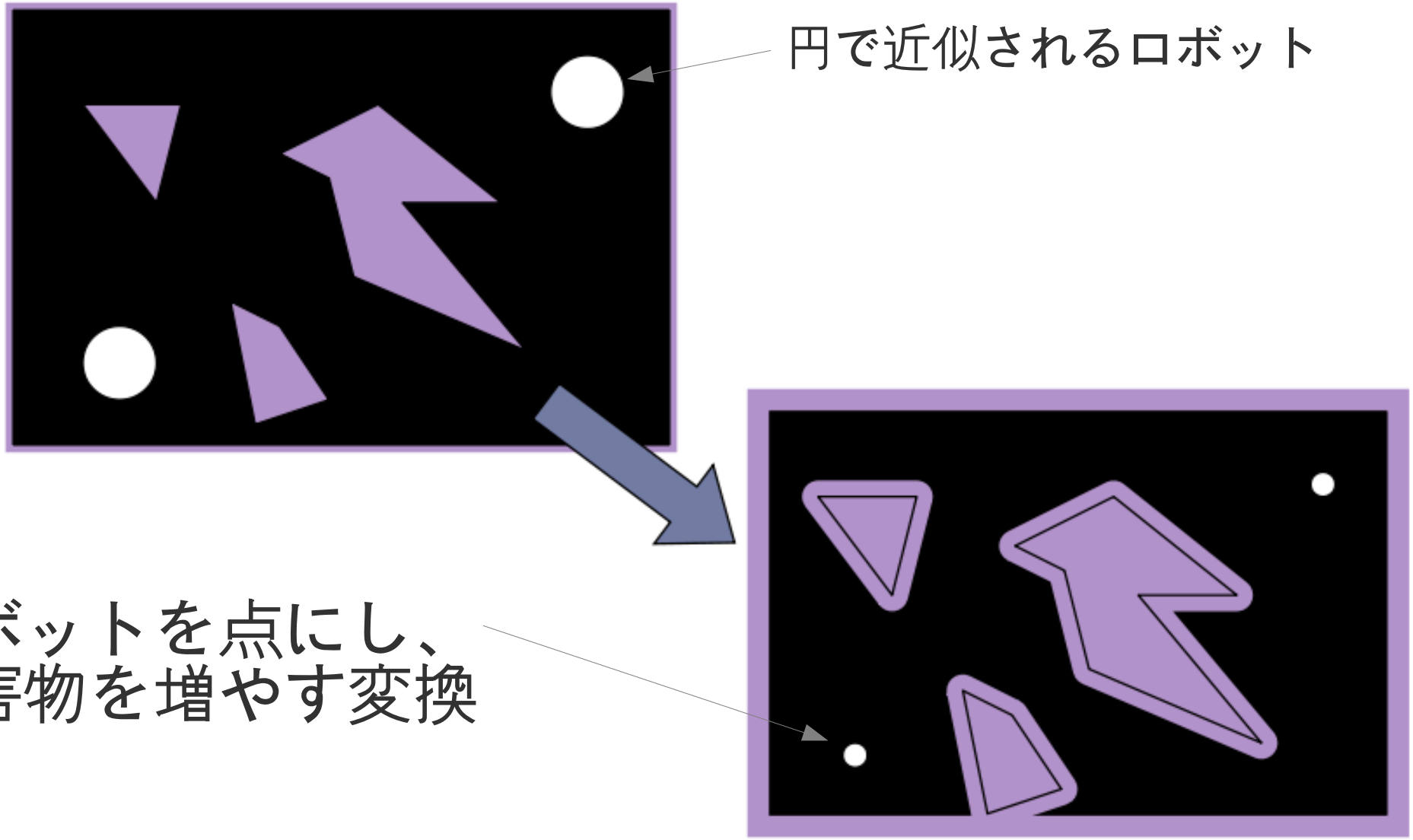
ロボット領域

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

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

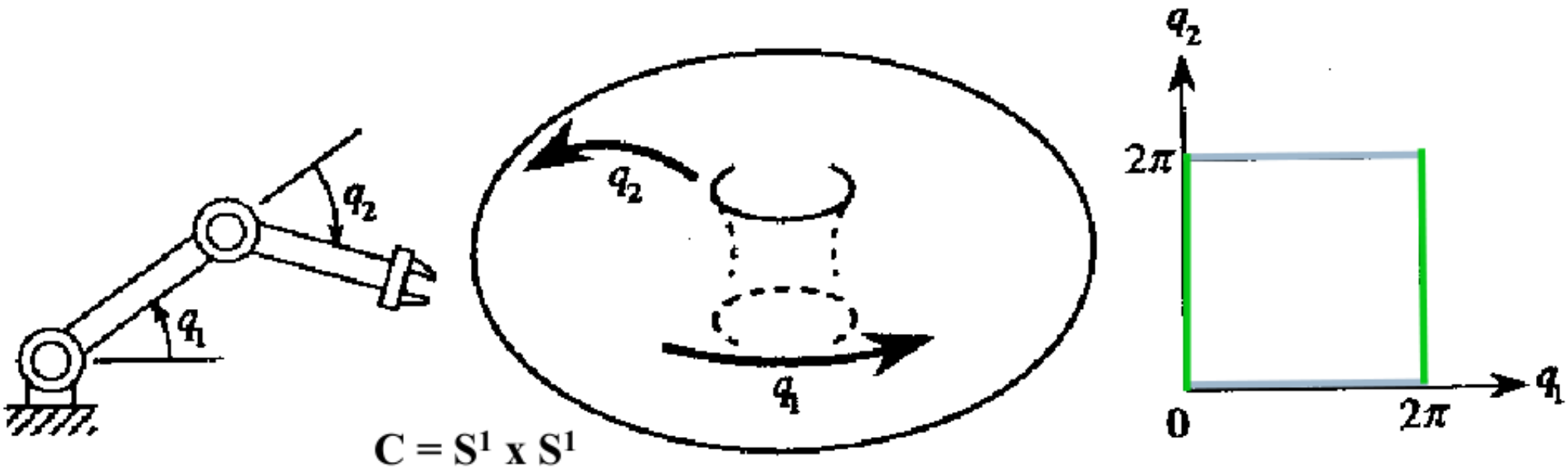


2Dのコンフィグレーションスペース



2関節ロボット
コンフィグレーションスペース

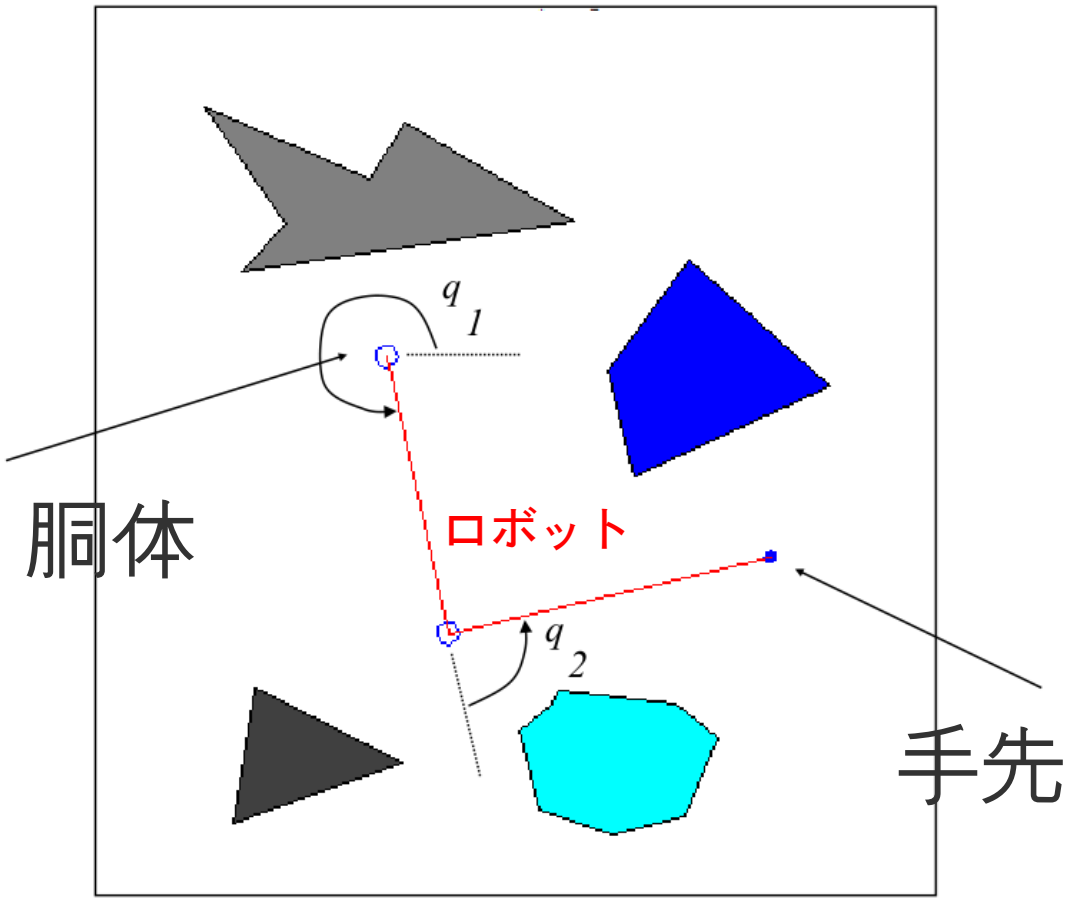
- 二つの関節は円環面（torus）の表面を描く



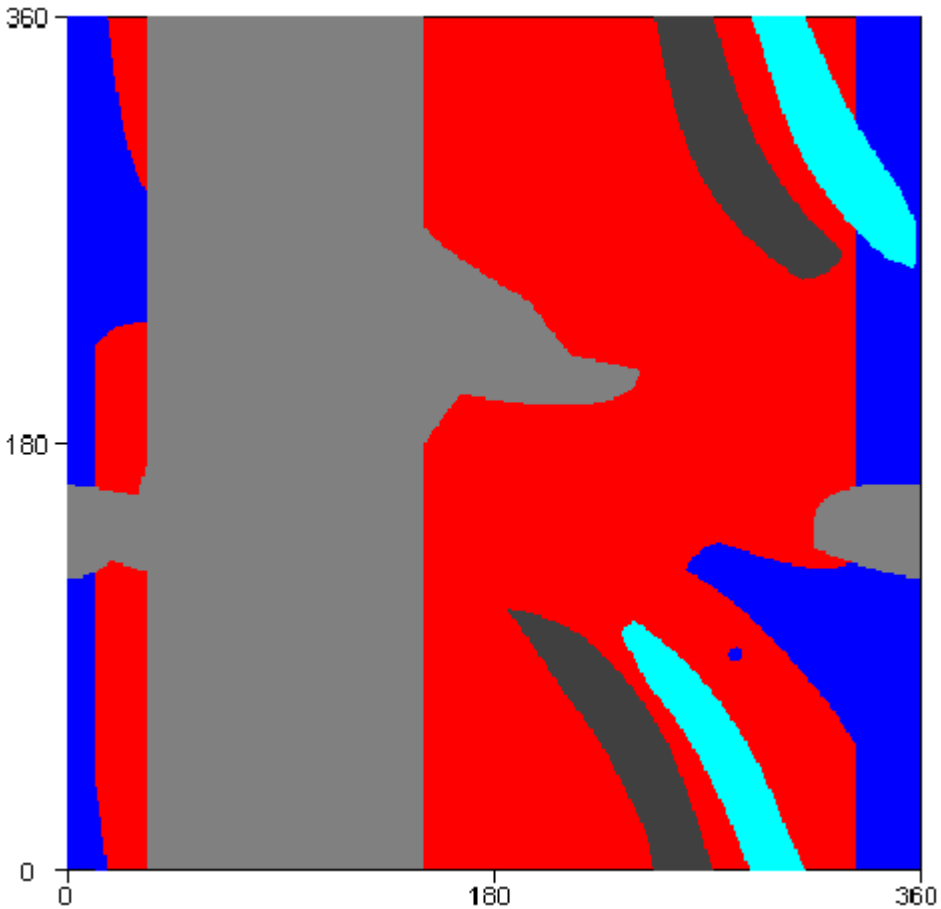
2D作業空間、2関節ロボット

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

作業空間

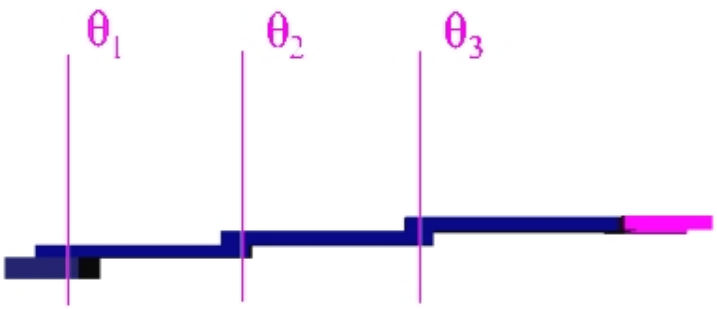


コンフィグレーション
スペース



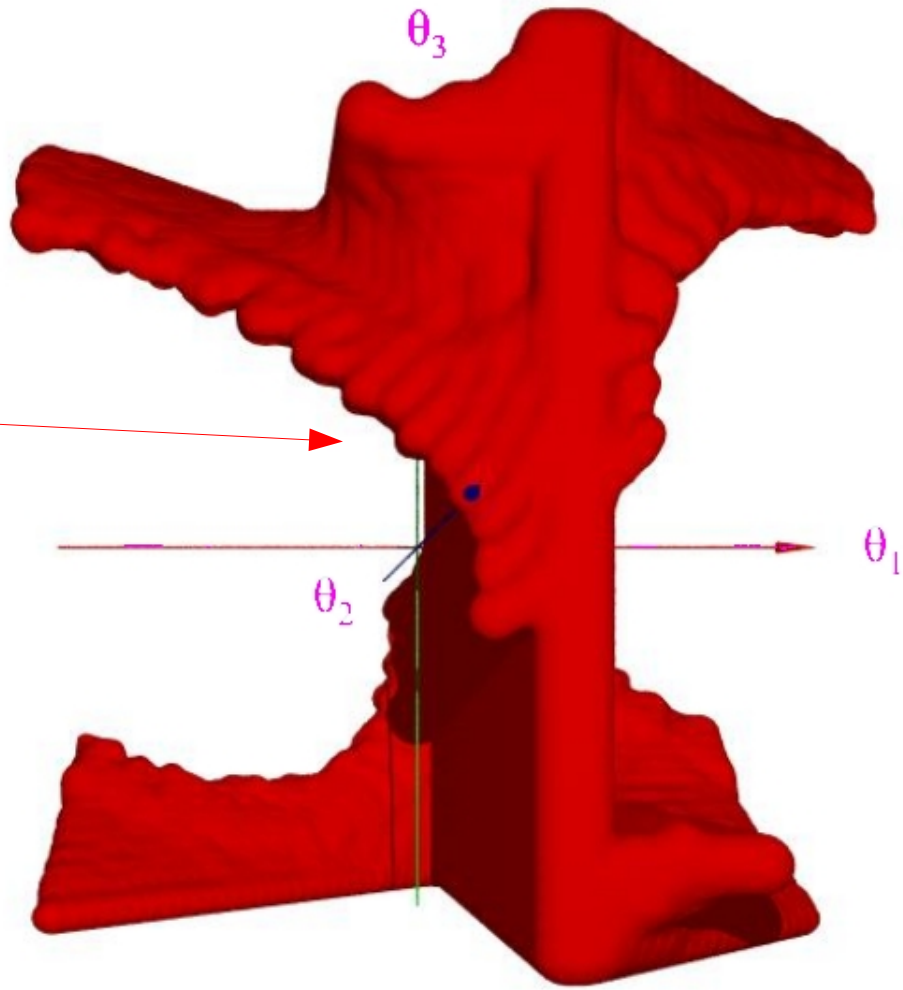
2D作業空間、3関節ロボット

作業空間

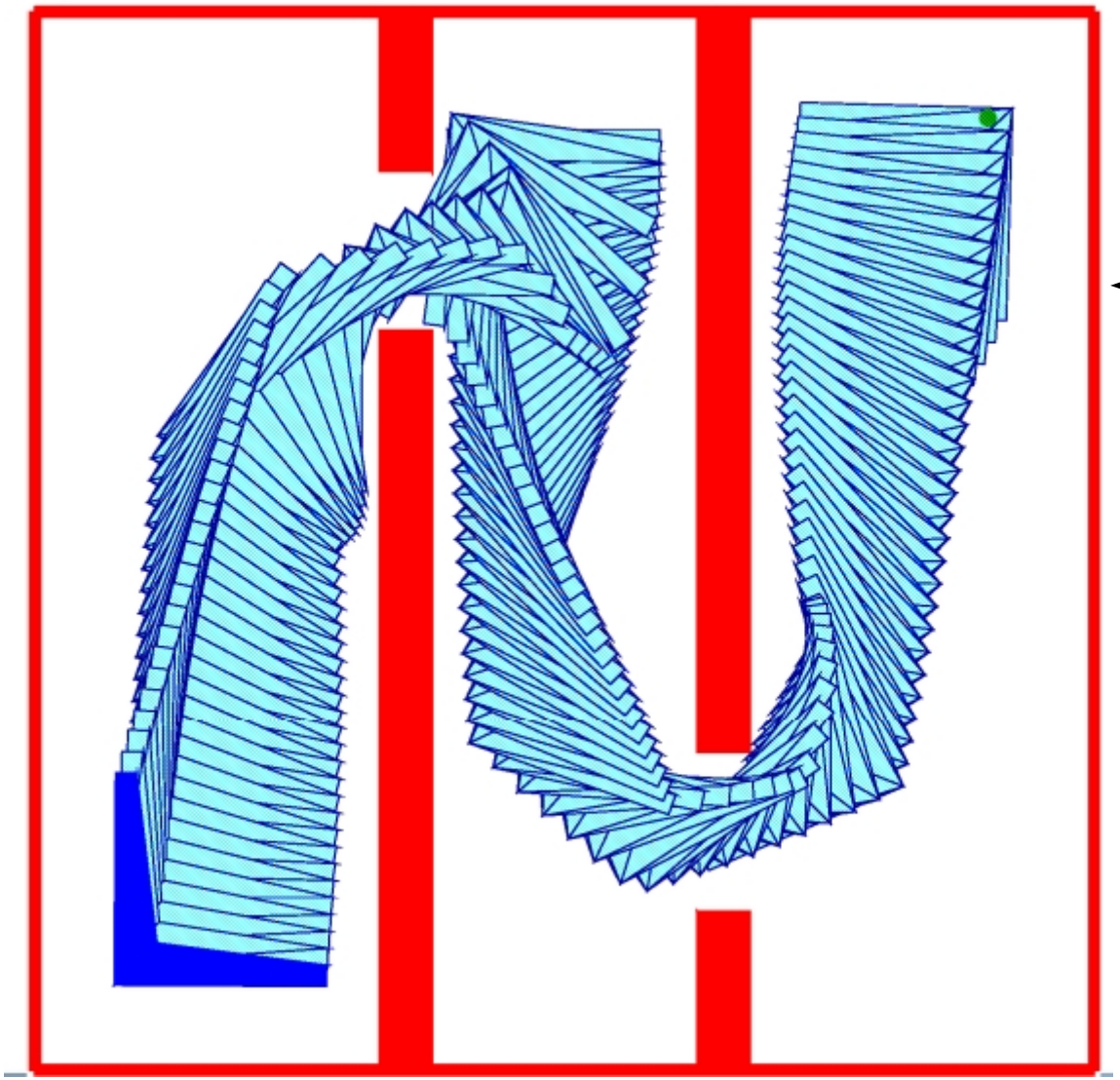


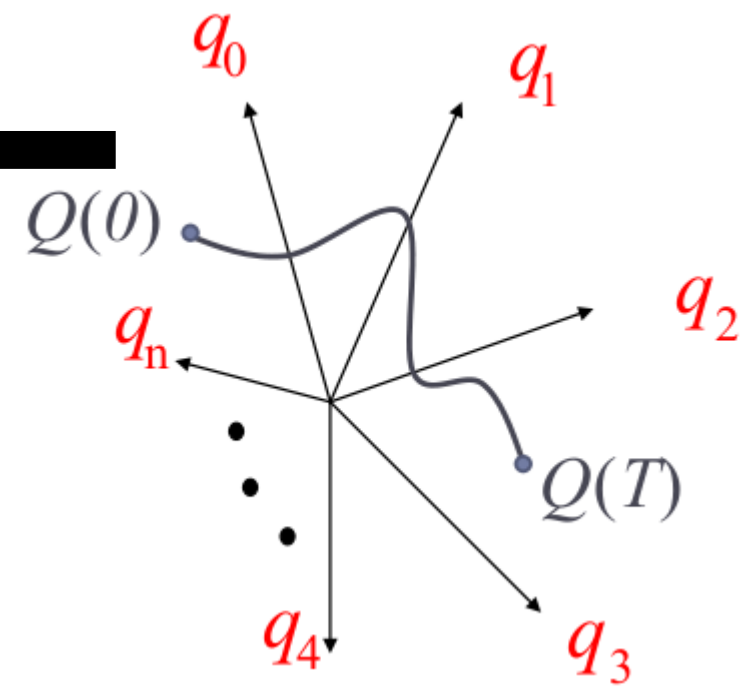
障害物

コンフィグレーション
スペース



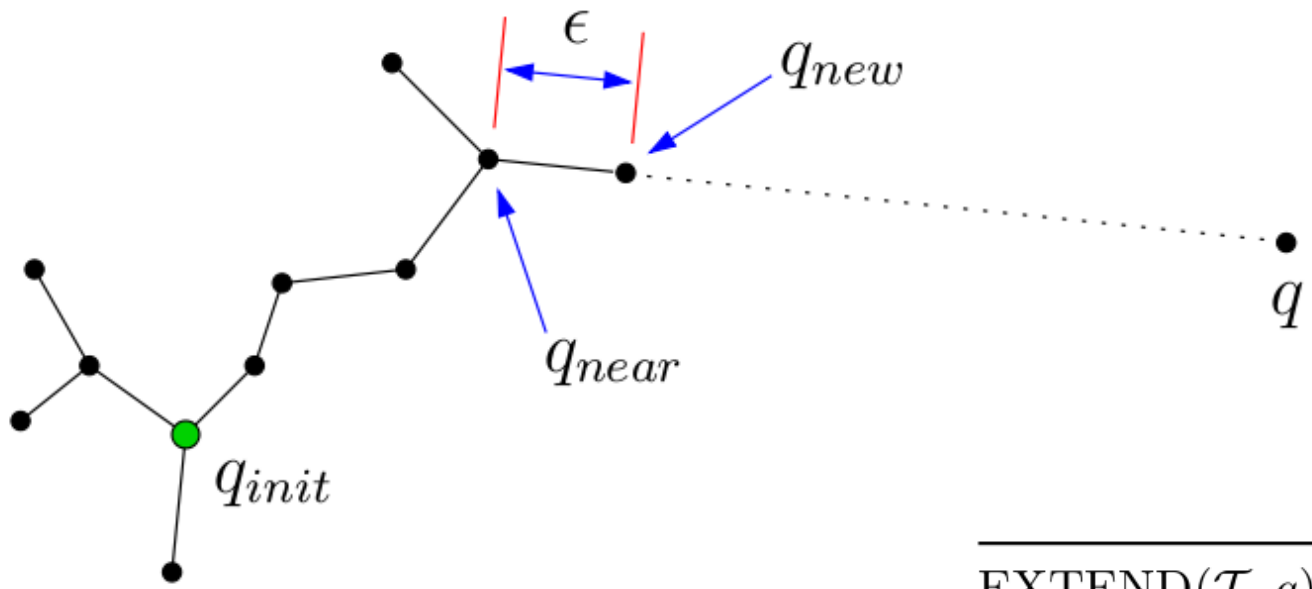
動作計画の目的:
コンフィグレーションスペースの経路





Rapidly-exploring Random Trees (RRTs)

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

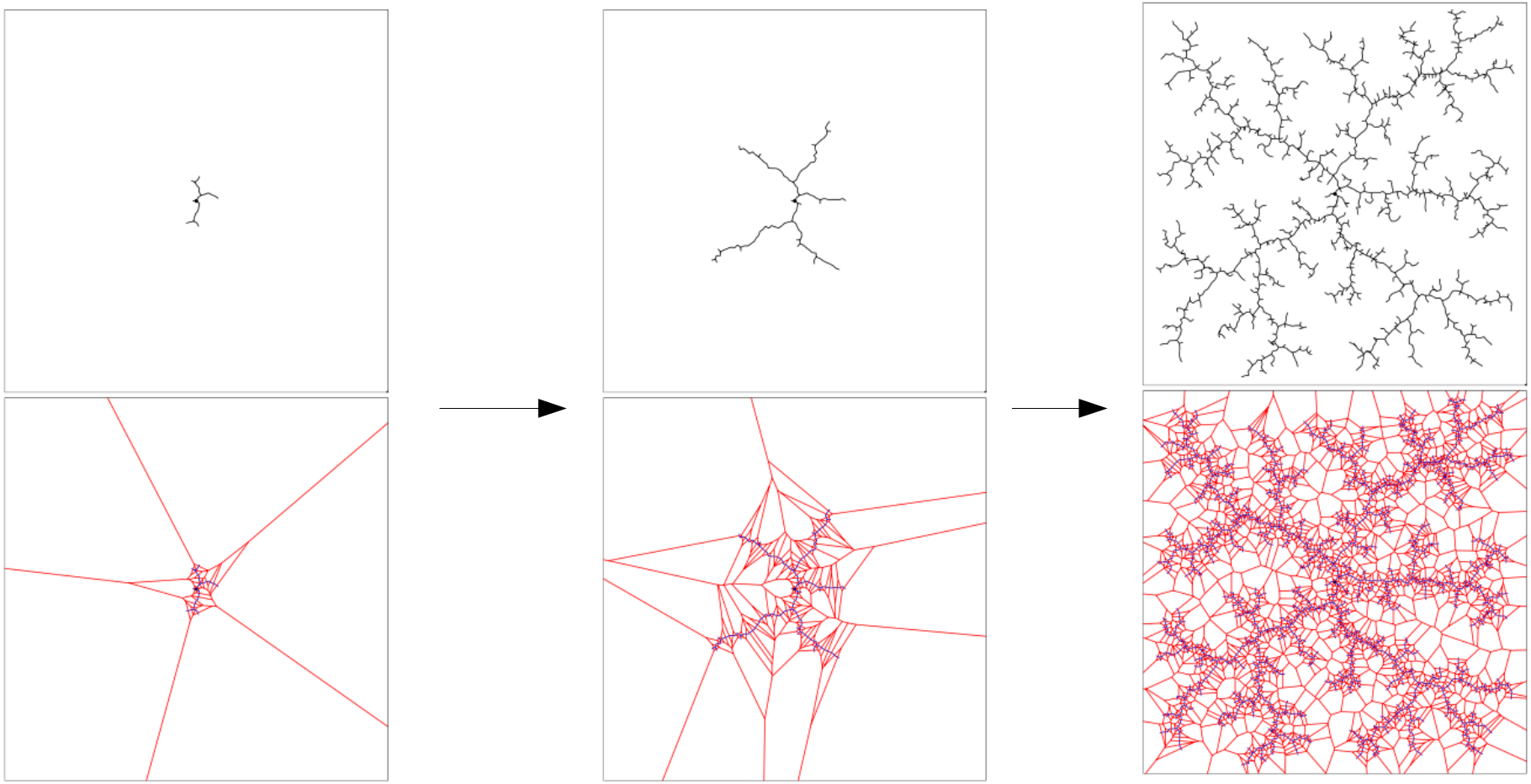



```
BUILD_RRT(q_init)
    T.init(q_init);
    for k = 1 to K do
        q_rand ← RANDOM_CONFIG();
        EXTEND(T, q_rand);
    Return T
```

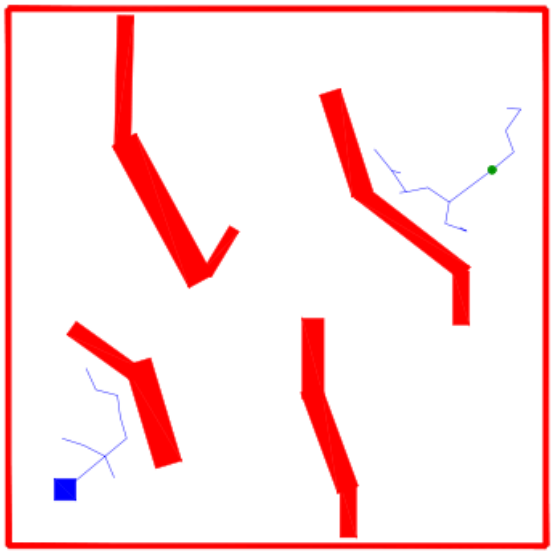
```
EXTEND(T, q)
    q_near ← NEAREST_NEIGHBOR(q, T);
    if NEW_CONFIG(q, q_near, q_new) then
        T.add_vertex(q_new);
        T.add_edge(q_near, q_new);
        if q_new = q then
            Return Reached;
        else
            Return Advanced;
    Return Trapped;
```

RRTsの探索傾向

- 不探索空間への生きがち

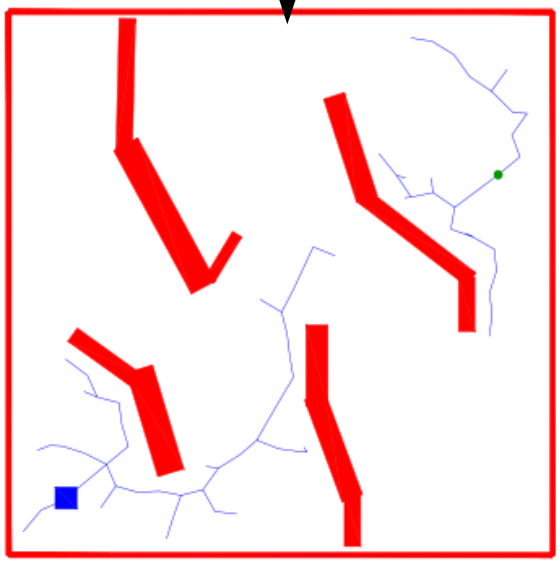


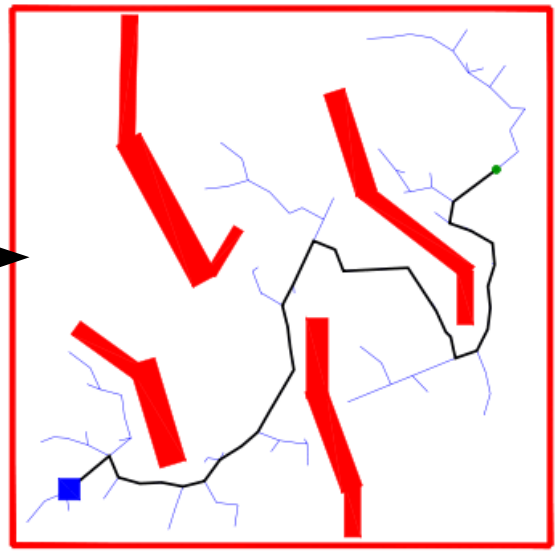
BiRRTs（両進探索）

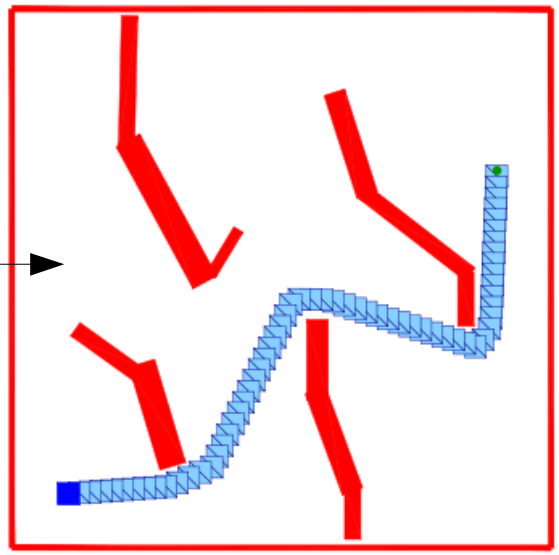


RRT_CONNECT_PLANNER(q_{init}, q_{goal})

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

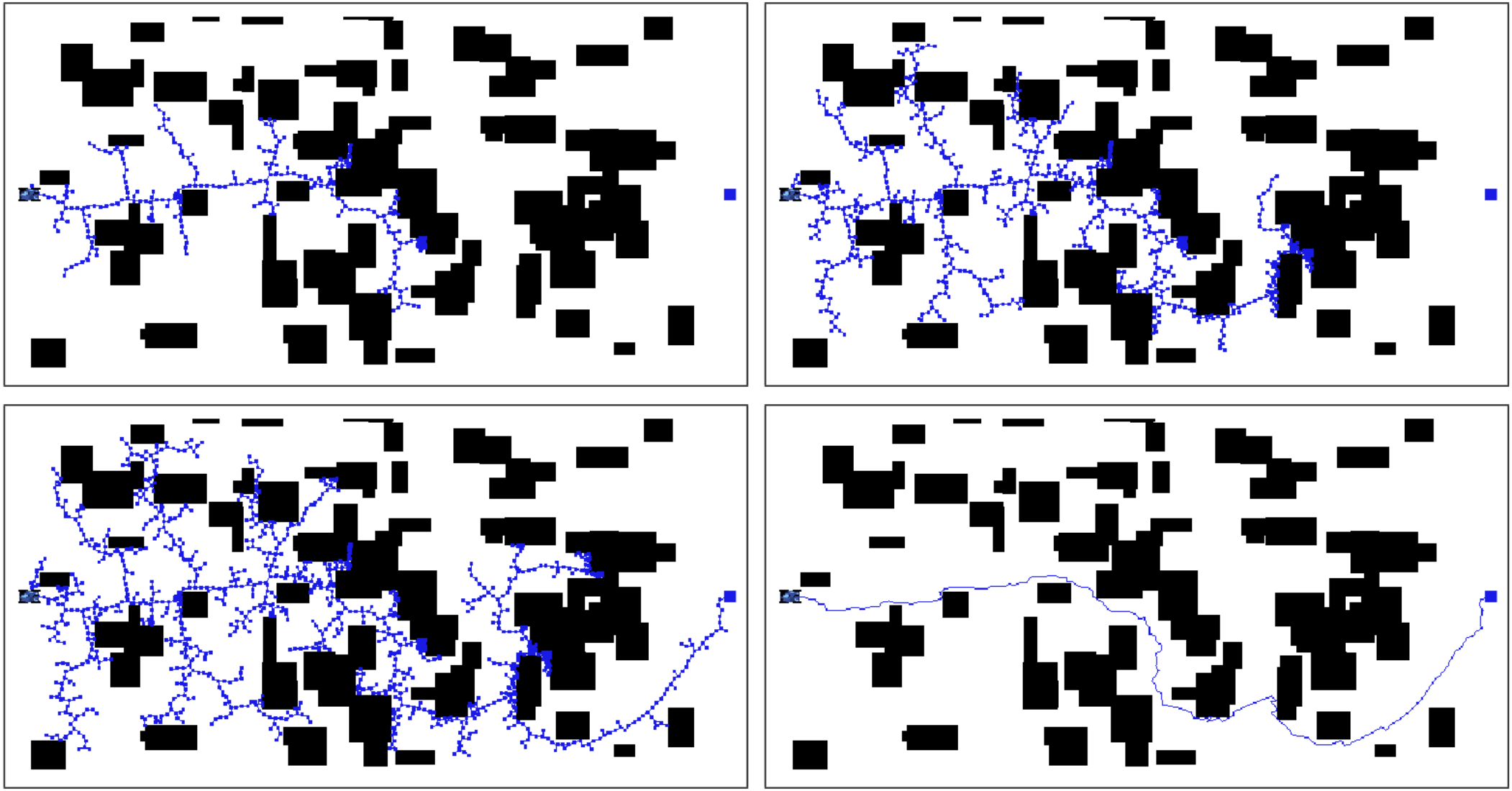




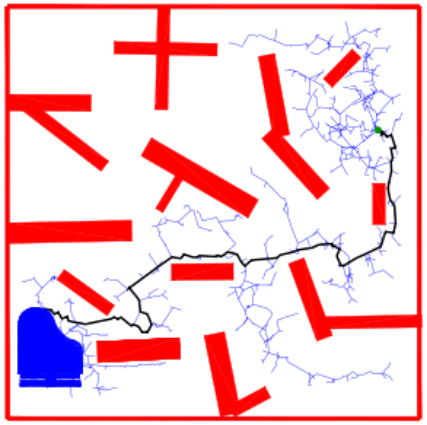


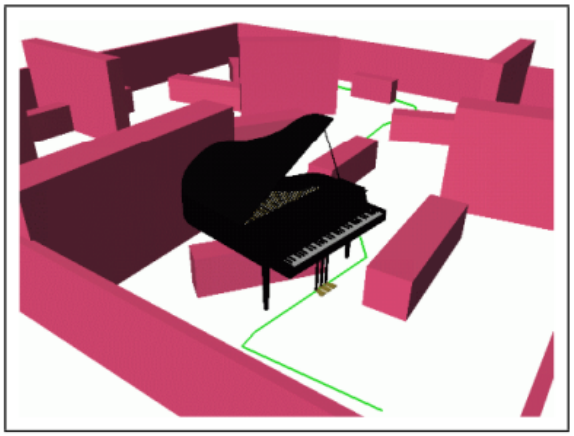
$\mathcal{T}, q$);
$nced$)

RRT探索例



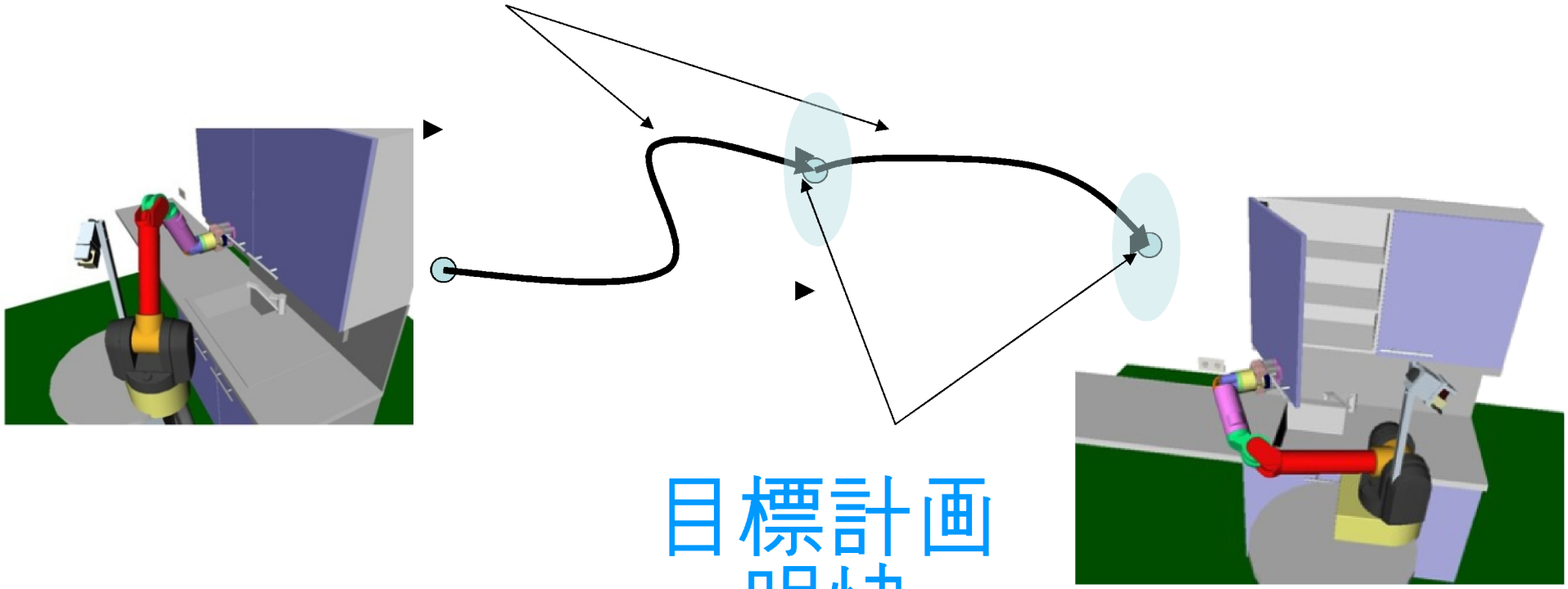
コンフィグレーションスペースの柔軟性





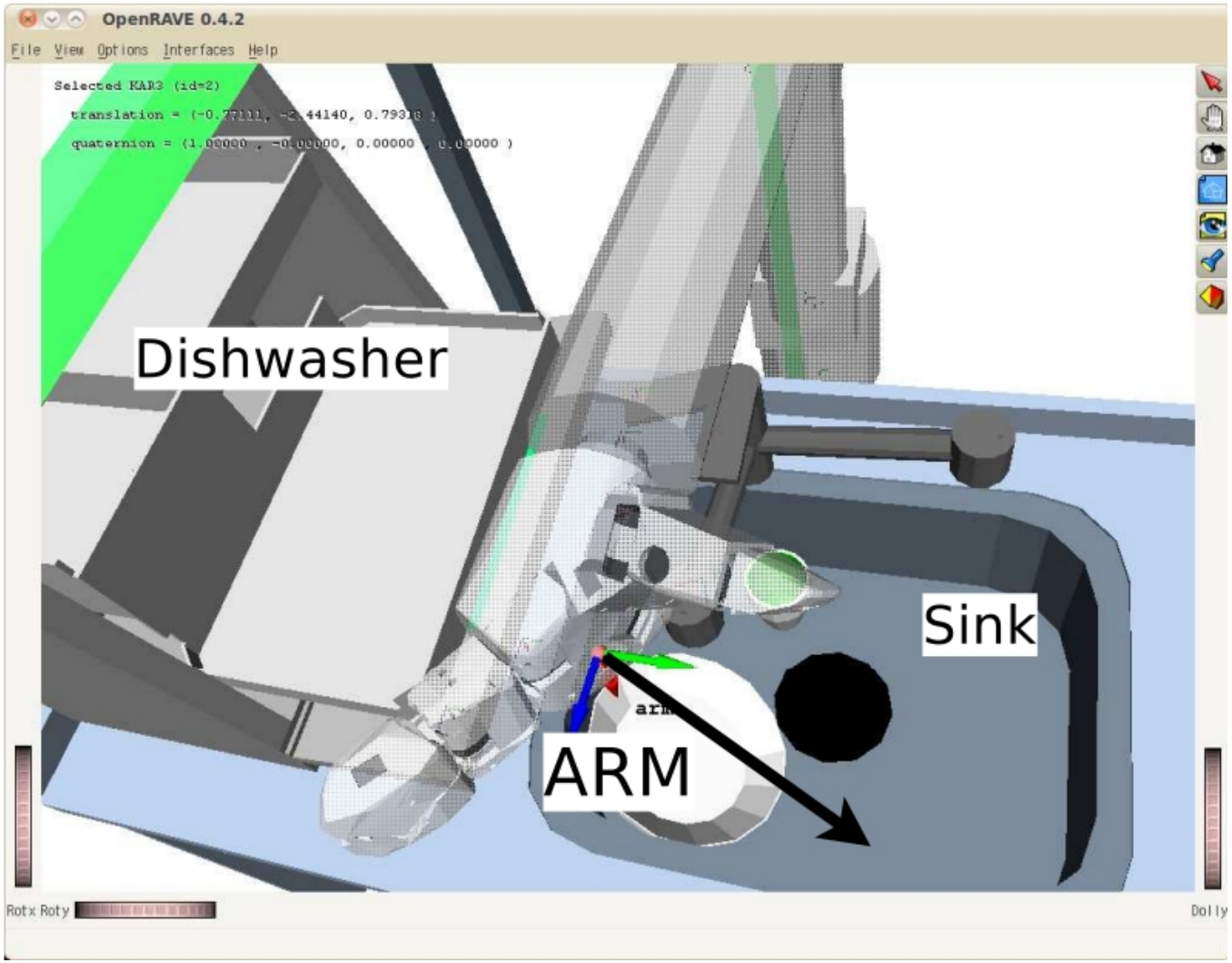
計画の二段階

動作経路（BiRRT, Delayed D*, Anytime RRTs）



目標計画
- 明快
- 暗黙

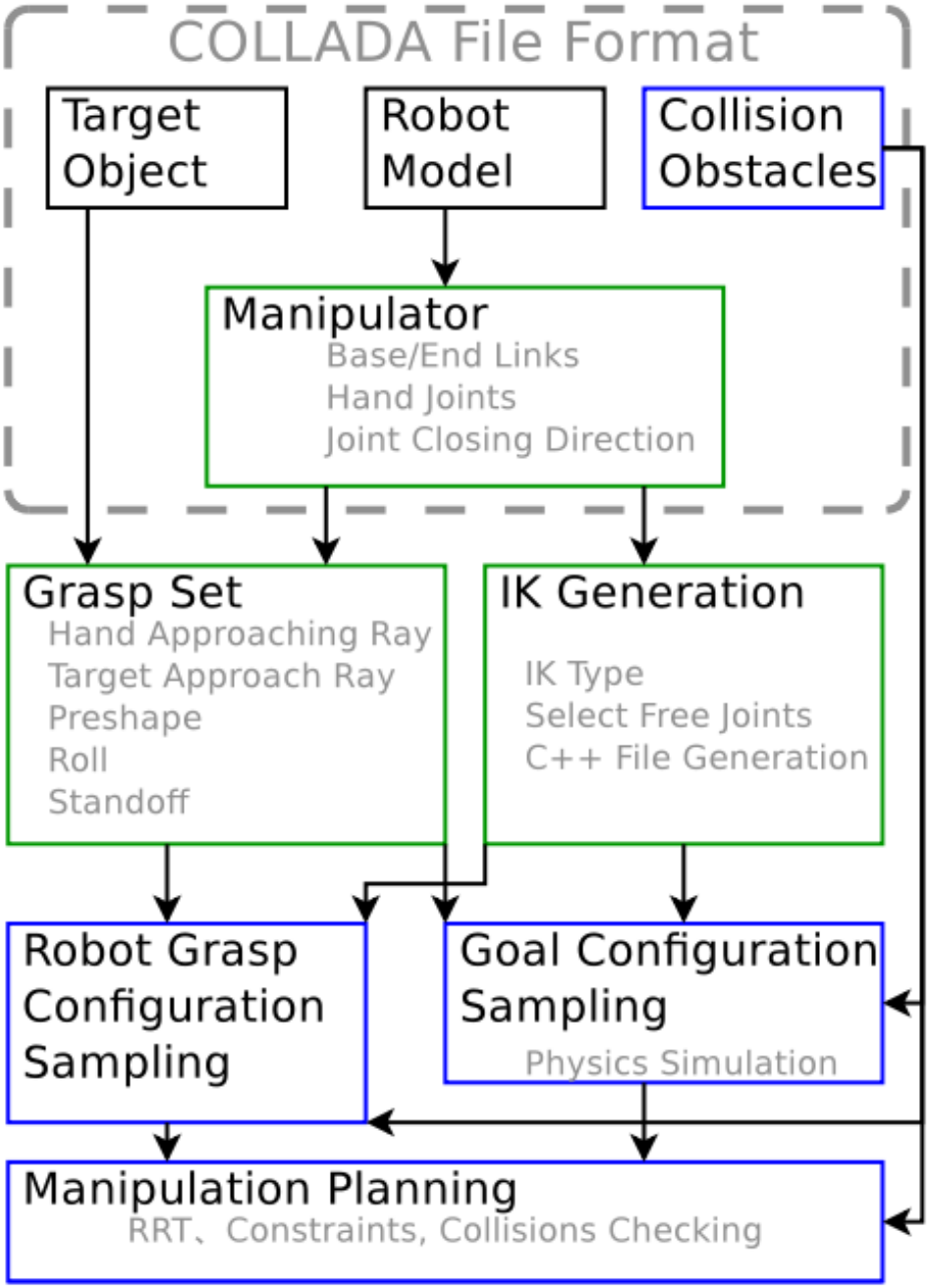
食器洗い機の搭載
マニピュレーション動作を作る開発過程



数時間で実現されたデモ

- 安定な把持
- マニピュレーション動作
- 障害物回避
- 逆運動学
- プランニングパラメータ

Pick-and-Place Architecture



逆運動学の生成

```
openrave.py --database inversekinematics --robot=myrobot.dae  --manipname=arm
--iktests=100 --iktype=transform6d --freejoint=base_joint
```

把持セットの生成

```
openrave.py --database grasping --robot=myrobot.dae --manipname=arm --target=myobject.dae
--friction=0.1 --preshape="-0.685 -1.48 0 0 0" --manipulatordirection="0.09 0.8  0.58"
--graspingnoise=0.01
```

ロボットコンフィグレーションサンプリング

openravepy.databases.grasping.GraspingModel.computeValidGrasps

目的のロボットコンフィグレーションサンプリング

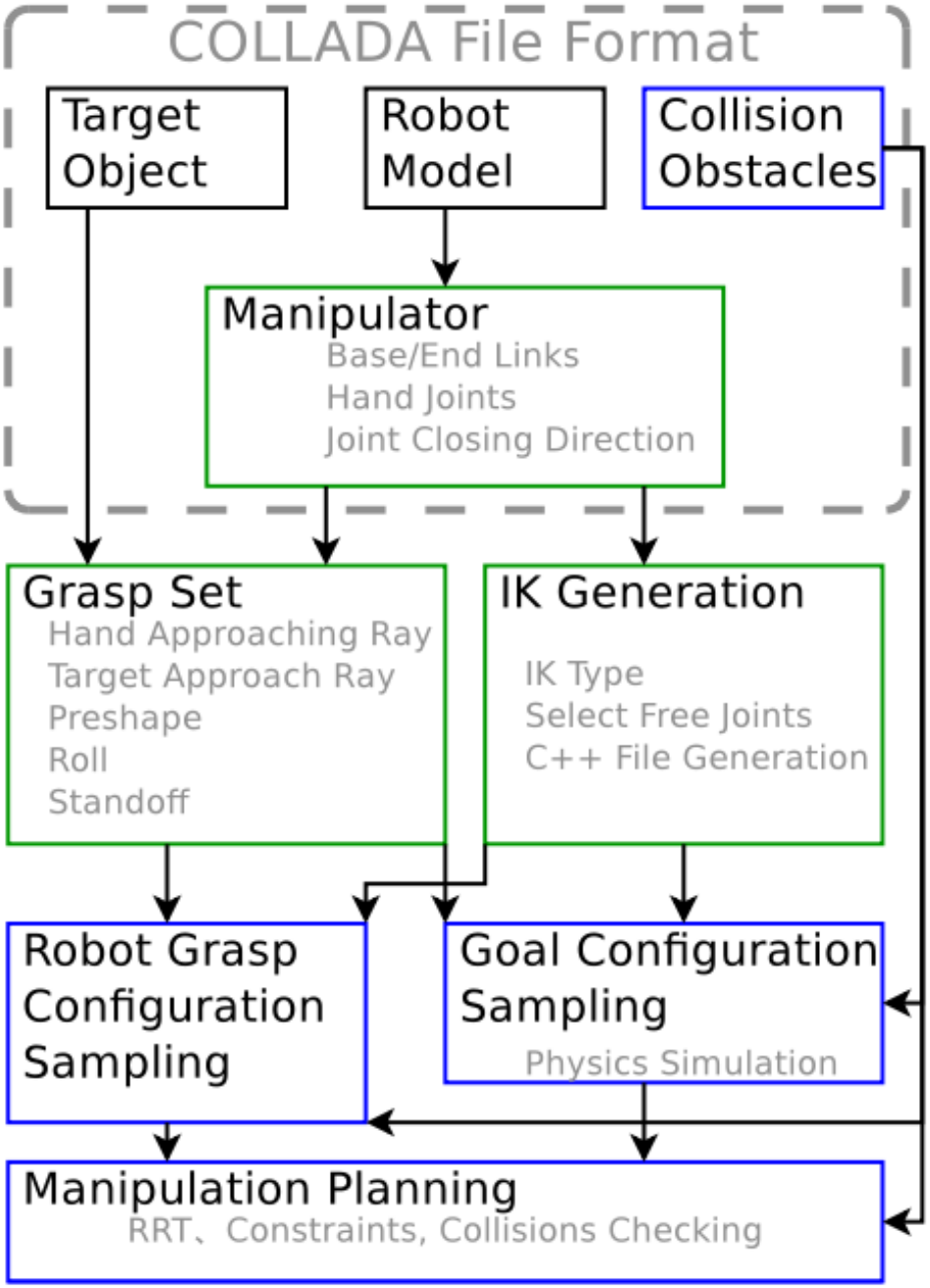
マニピュレーション動作 (6D IK)

openrave.py --example graspplanning

マニピュレーション動作 (5D IK)

openrave.py --example graspplanning --scene=data/katanatable.env.xml

Pick-and-Place Architecture

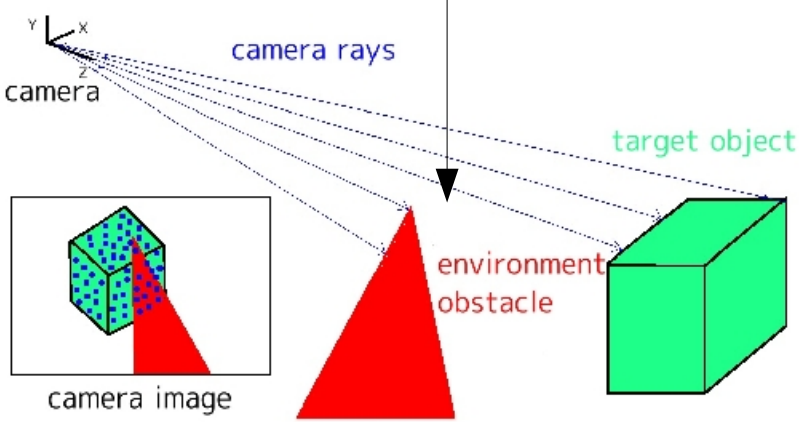


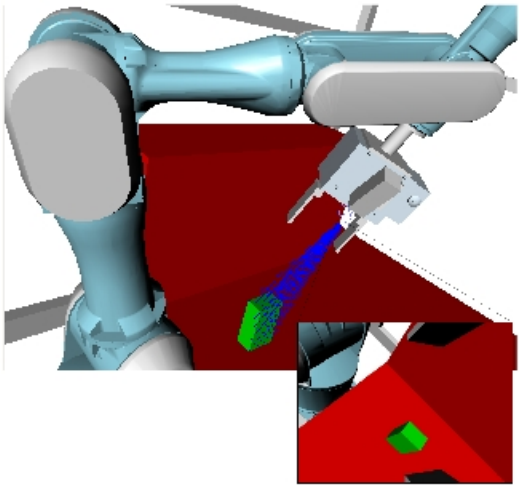
ハンドの制約条件

```
openrave.py --example constraintplanning
```

センサ視認能力の最適化

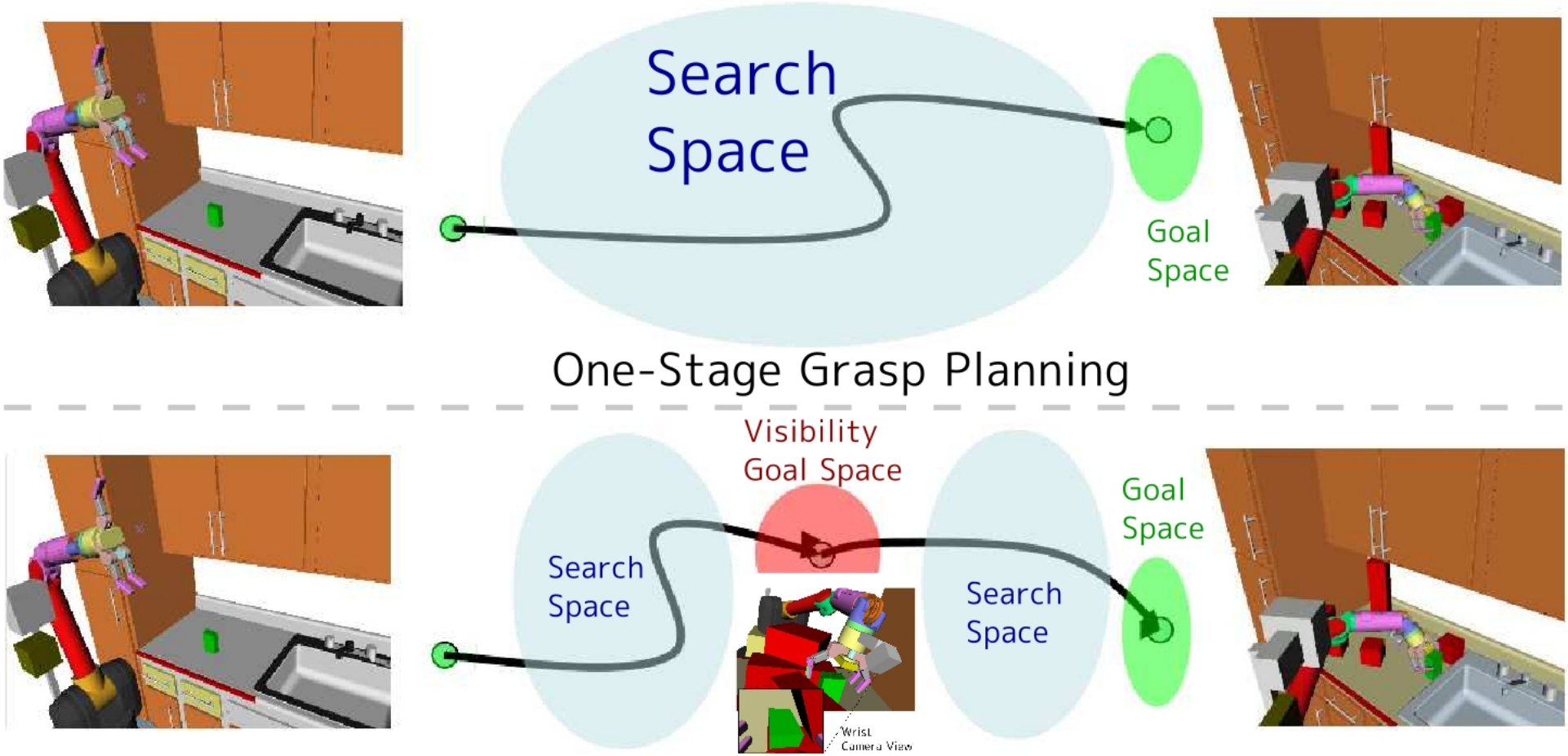
- 最も効率的なデータ取得可能
- 三つの要因
  - 視認性範囲
  - 運動学
  - 障害物



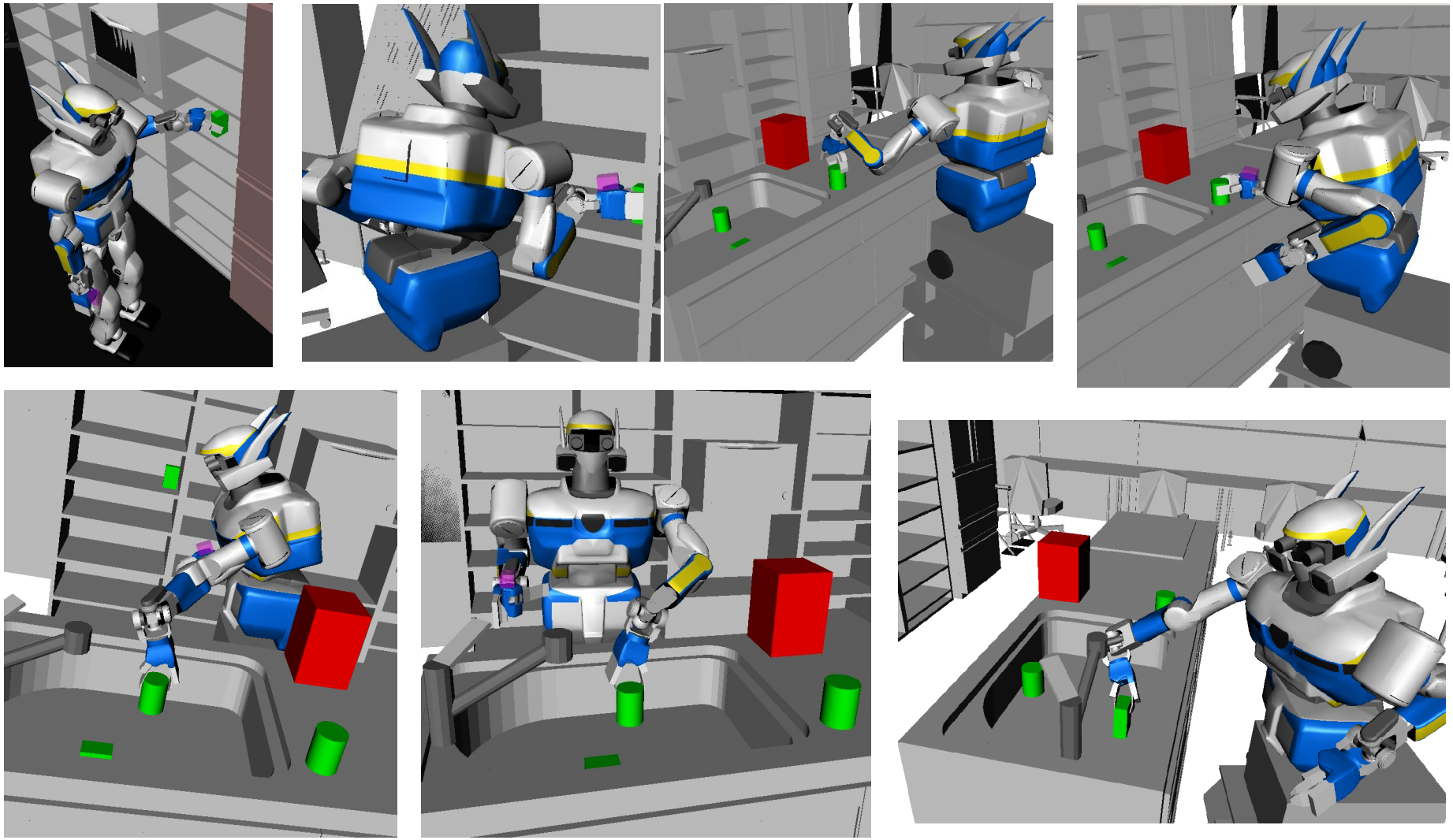


openrave.py --example visibilityplanning

2段階の効率


Search
Space
Goal
Space
One-Stage Grasp Planning
Visibility
Goal Space
Goal
Space
Search
Space
Search
Space
Wrist
Camera View
Two-Stage Visibility + Grasp Planning

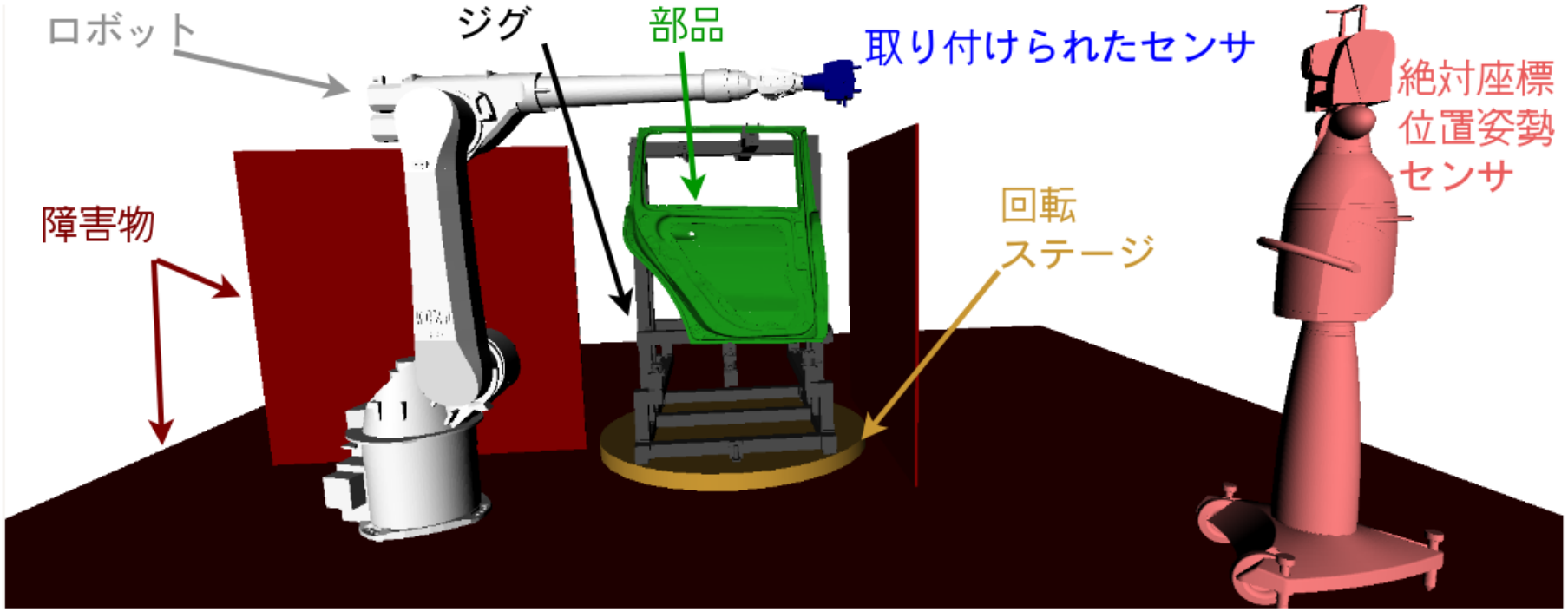
胴体＋アーム＋把持のサンプリング



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

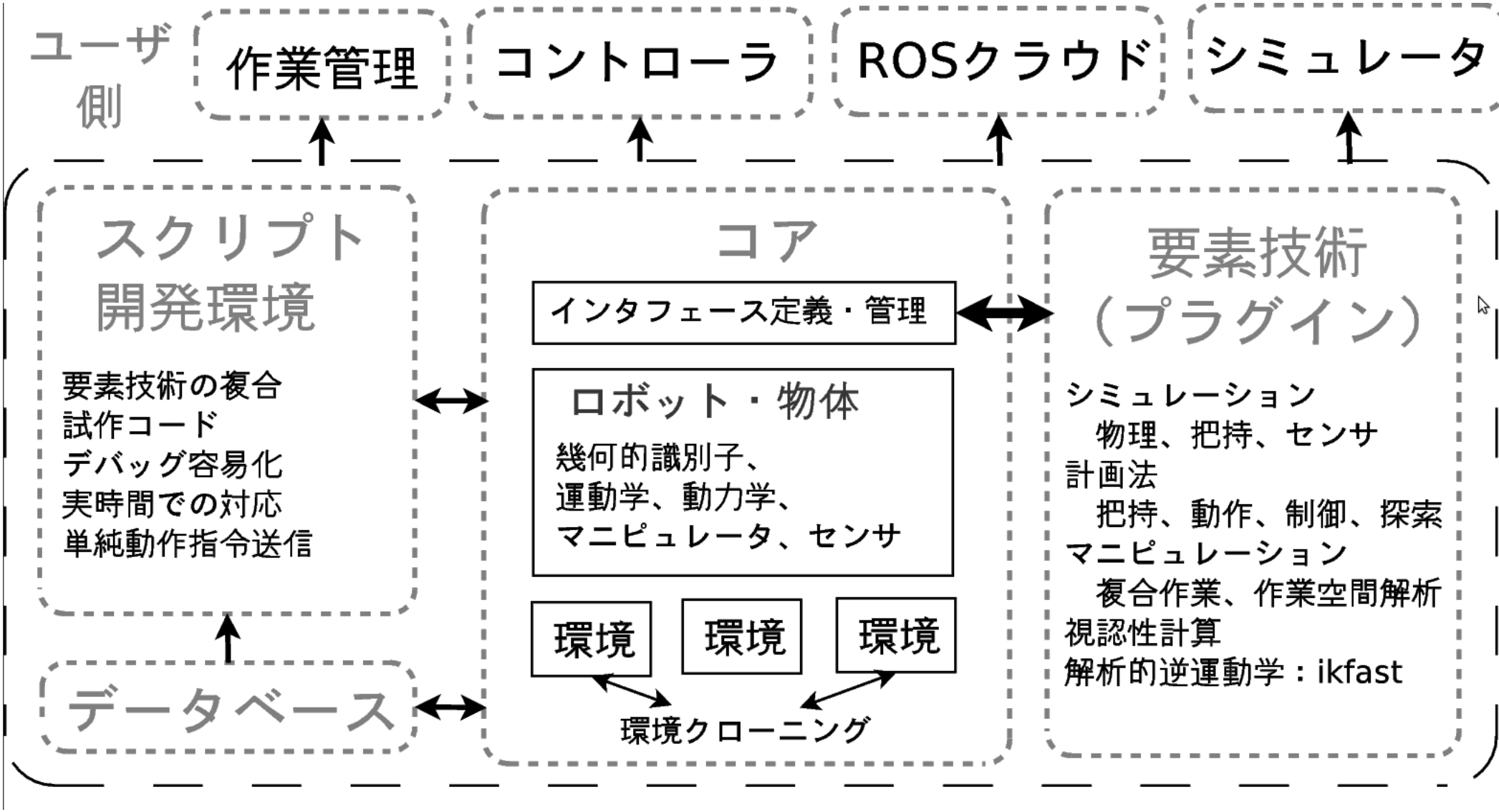
部品の形状検査



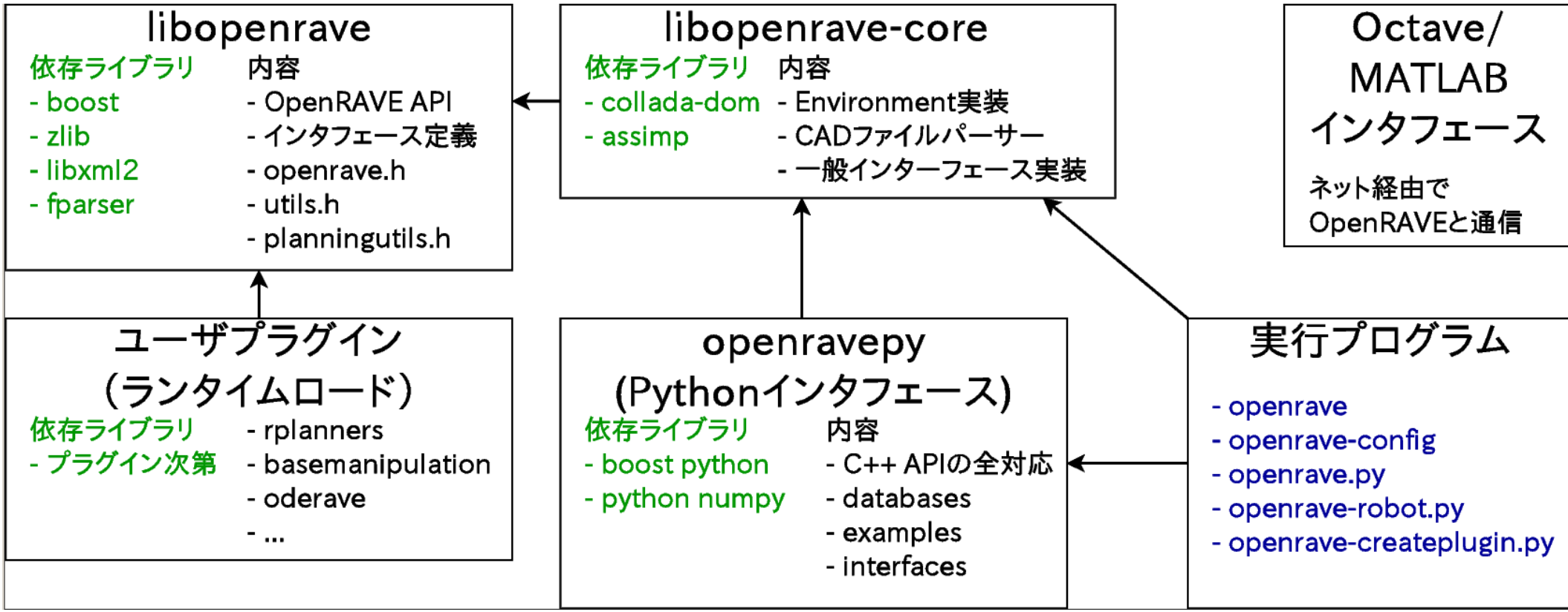
双腕の動作

OpenRAVE
アーキテクチャ

OpenRAVEアーキテクチャー一覧



OpenRAVEライブラリの構成



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インタフェース

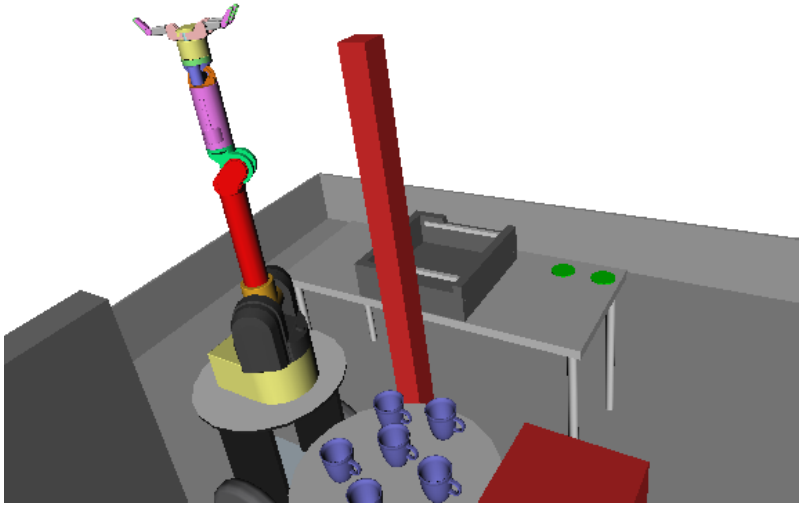
- マニピュレーター
  - Base and End Links forms arm sub-chain - IK使用
  - List of gripper joints
  - check collision or query links
    - gripper only
    - Independent links of the manipulator
- アクティブ関節
- 物体との剛体接地
- センサー設置

OpenRAVE Python Scripting

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/phisics

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

- サンプルの実行

```
openrave.py --example graspplanning
```

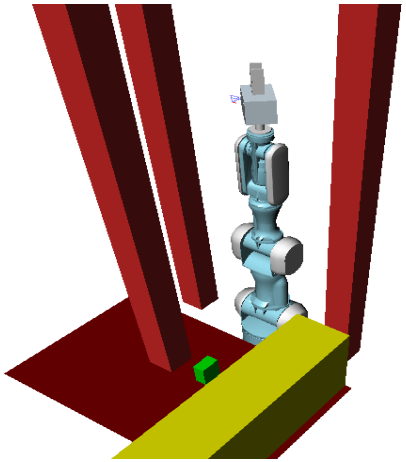
- Execute database generator: solve analytic IK, create C++ file

```
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 visibilitymodel --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 *
from numpy import * # 必要ないけれど、よく使うライブラリです
env=Environment()
env.SetViewer('qtcoin')
env.Load('data/pa10grasp2.env.xml')

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

ikmodel=databases.inversekinematics.InverseKinematicsModel(robot,iktype=IkParameterizationType.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]
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('prause')

# 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.SetDOFValues(sol,manip.GetArmIndices())
    env.UpdatePublishedBodies()
    raw_input('pause')
```