エージェントシステム
RTM/ROS相互運用
RTM 第４回

2011/06/29（水） 13:00 – 14:30

花井　亮

講義の概要

- HIRO-NXの事例紹介
  - HIRO-NX実機の認識から実行までを
    つなぐシステム
  - システム設計、ツール選択
  - 各機能に関連するRT-component
  - 泥臭い話

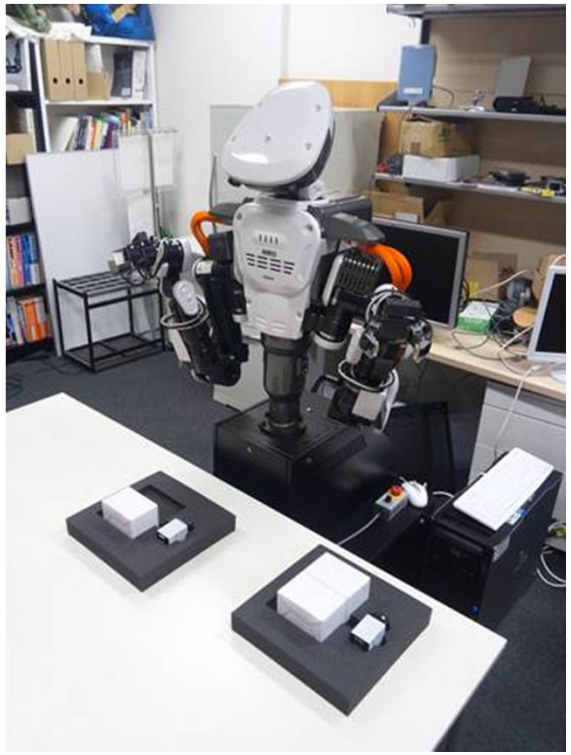
- OpenHRP3-GRX上でHIRO-NXを動かす試み

背景

- 知能化プロジェクト（NEDO）
  - RT-middlewareで再利用性高いシステム構築
- 加速（知能化プロジェクトのサブプロジェクト）における目標
  - 双腕ロボットHIRO-NX上で、視覚に基づくタスク実行をRTCベースで実現
  - 複数の大学、研究機関が連携
  - インタフェースを決めて統合を計画
- RTM-ROS連携

HIRO-NX

- 今年から利用開始
  - 大学向けのコードネームHIRO-NX
- 川田工業製
  - 首:2、腕:6(x2)、腰(yaw):1
    の計15自由度
  - ハンド左右各4自由度
  - 頭部ステレオカメラ、両手にUSBカメラ



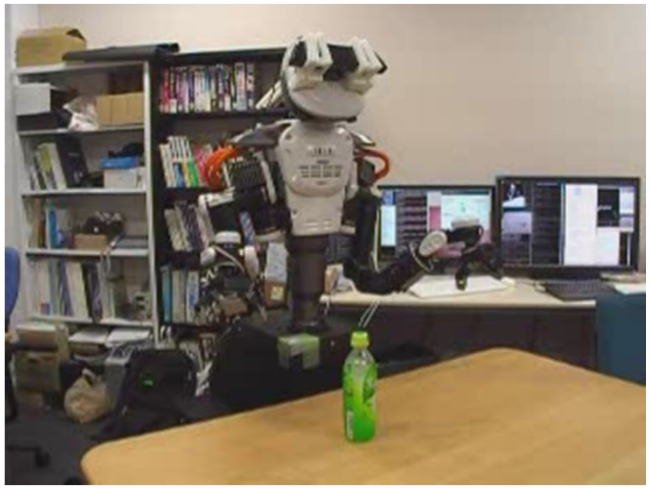
- General Robotics (GRX)がベースのソフトウェアを開発
- OpenRAVE(本講義)で動作計画に使ったロボット

HIRO-NX動画(1/2)

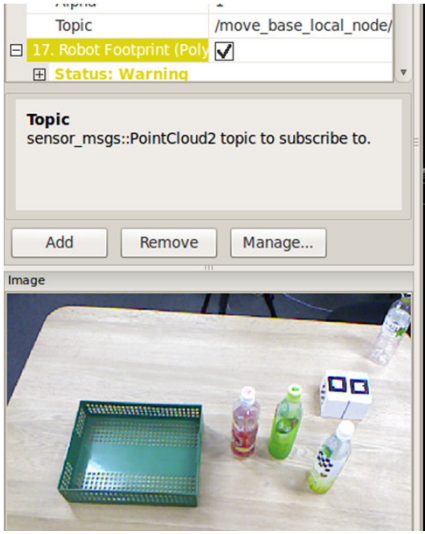
頭部カメラで対象物を
認識し、再配置

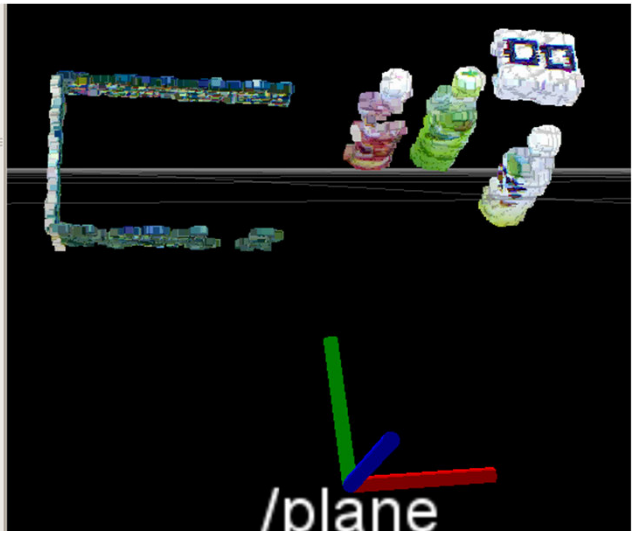
- 手首を動かしながら
  ハンドカメラで対象物を
  認識
- 箱状物体を積み重ねる

HIRO-NX動画(2/2)



現在はROSで利用
OpenNIノード
⇒ Point Cloud Library(PCL)でフィルタ
⇒ 平面検出（JSKのROSパッケージ）で
テーブル除去





HIRO-NXでやろうとしていること

- パレタイジング、（お菓子の箱詰め）
  - Pick&placeを高い精度で行う
  - 共通インタフェースによる再利用性

- Pick&placeを補うスキル
  - ロバストなタスク実現
- 人との協調作業
- プランニングの問題として解く

ハンドカメラの画像

複数物品が近接する場合のアプローチ例

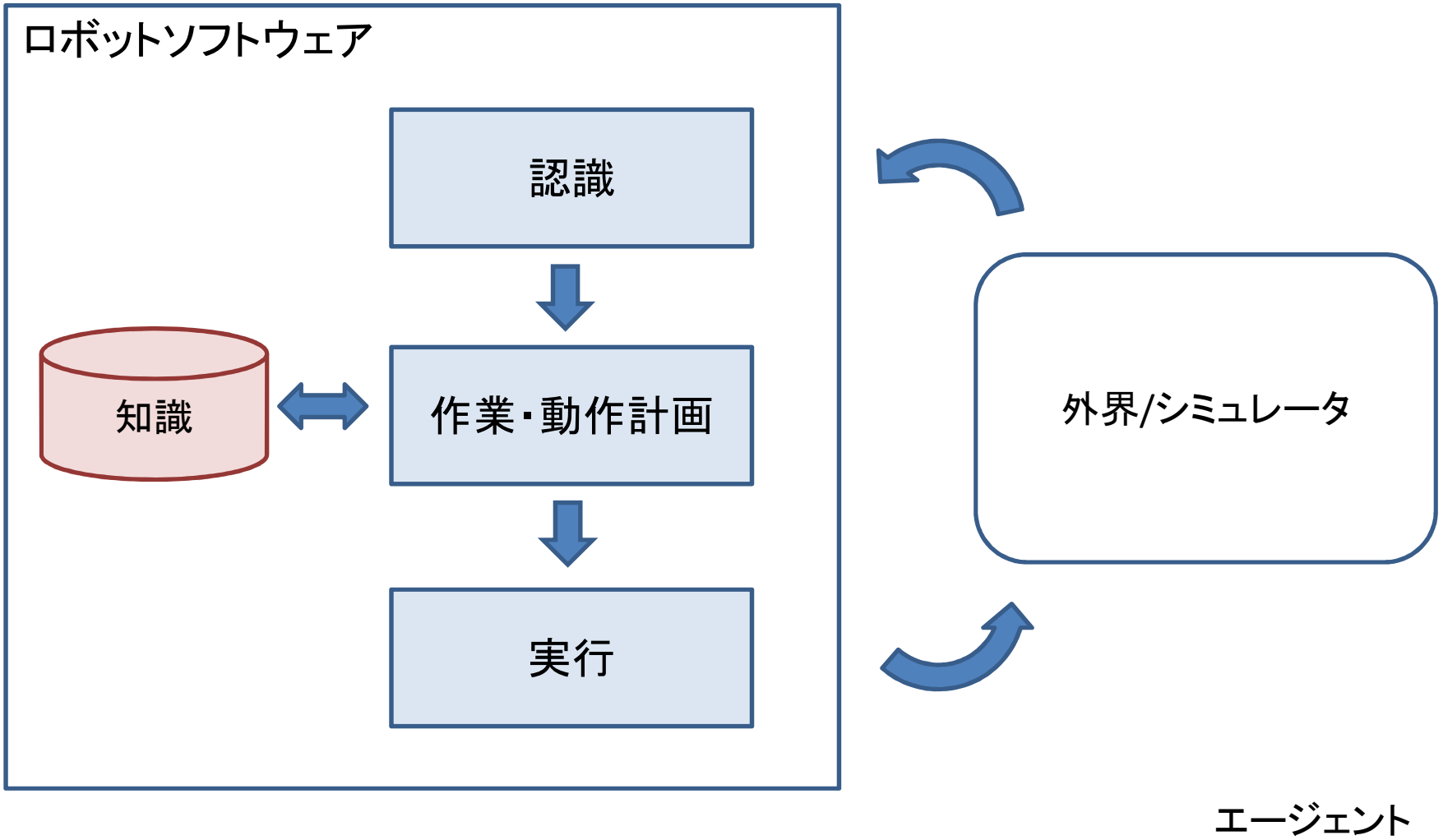
邪魔な物を
掴んで
少し移動
させてから
掴む

他のものを
どかしながら掴む

邪魔なものを
押して移動させてから
掴む

ARマーカで認識
動作は作り込み

機能的な描像



システム設計

- RT-middlewareとROSを使う
  - 再利用性
  - 講義で登場した多くのツール、概念を利用する事例
  - （類似機能を整理）

ロボットシステム開発におけるミドルウェア

- RT-middlewareとROS

- 計算モデル、通信モデル
  - 類似
  - RTMは実行コンテキスト、状態をミドルウェアレベルで取り入れている

- Deployment
  - ソフトウェアと実行に必要なものの入手、コンパイル、配置、プログラムの起動、終了というプロセス
  - ROSが便利

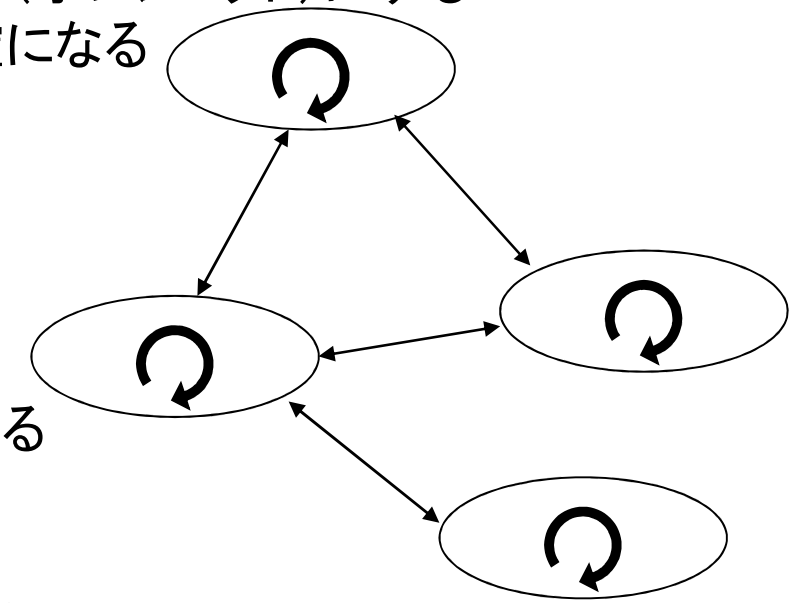
計算モデル

- 分散システム
  - 開発プロセスの分離
  - システム全体でのロバスト性
  - 複数プロセッサの自然な利用

- 計算モデル
  - 手続き型
    - C言語、PASCAL
    - 処理の列
  - オブジェクト指向
    - データをそれに対する操作をひとまとめ
    - あくまでデータとコード
  - 並行オブジェクト
    - 各オブジェクトに制御フローを持たせる
    - オブジェクト+制御（コンテキスト）

並行オブジェクトモデル

- オブジェクト志向
  - データとそれに対する操作をひとまとめ(オブジェクト)にする
  - 内部データと外部インタフェースが明確になる
  - 部品として再利用性があがる



- 分散オブジェクト
  - オブジェクトを複数プロセス(計算機)に分散させたもの
  - 外部インタフェースがそのまま通信となる
- 並行オブジェクト
  - 各オブジェクトが独立した制御フローをもつ
  - 通信データはキューに入り、各オブジェクトのコンテキストで処理される

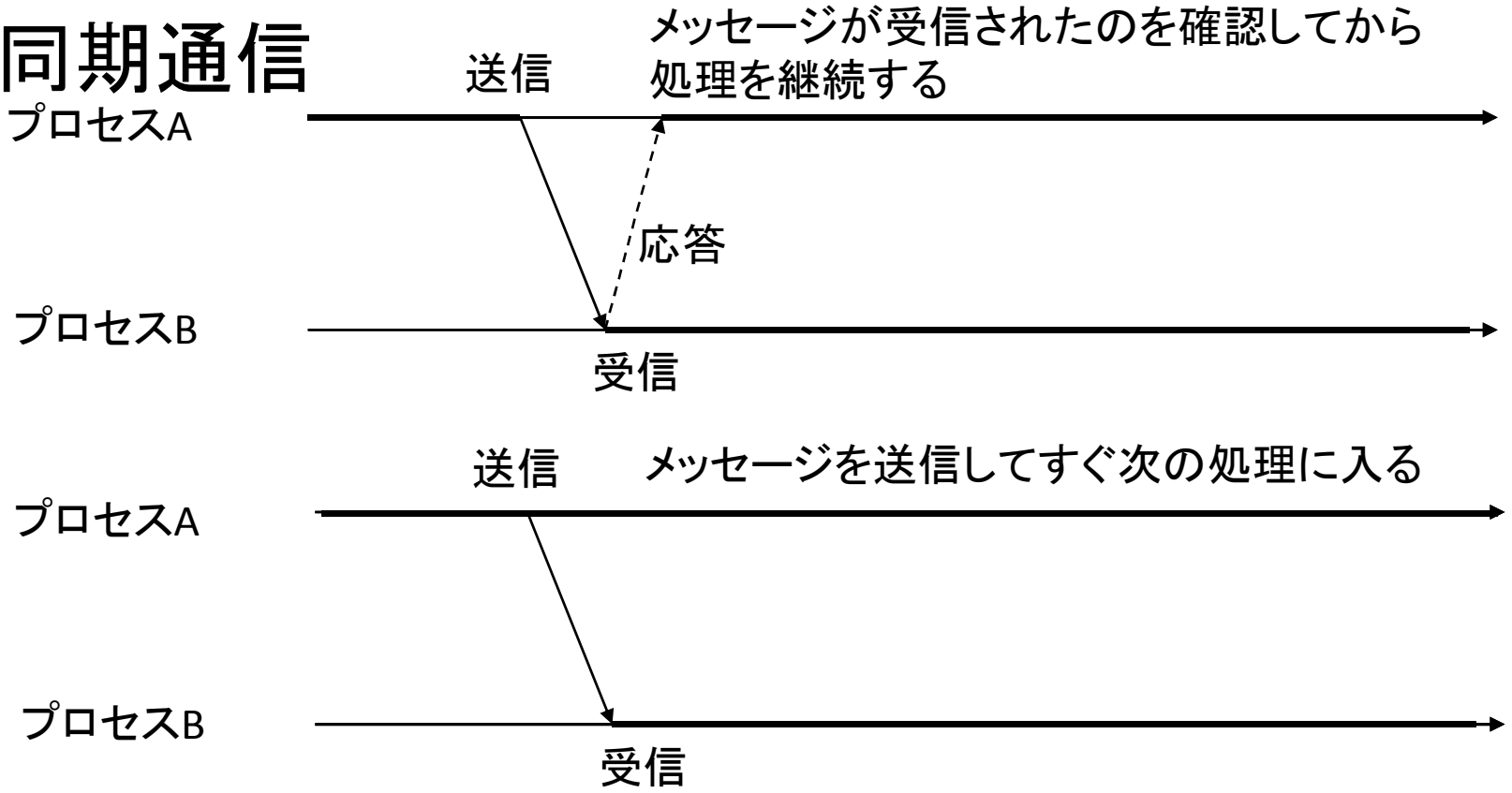
通信モデル

- 通信モデル
  - 同期、非同期
  - メッセージパッシング、リモート関数呼出し

- どちらも非同期メッセージパッシングと、
  同期関数呼出し

通信（メッセージング）

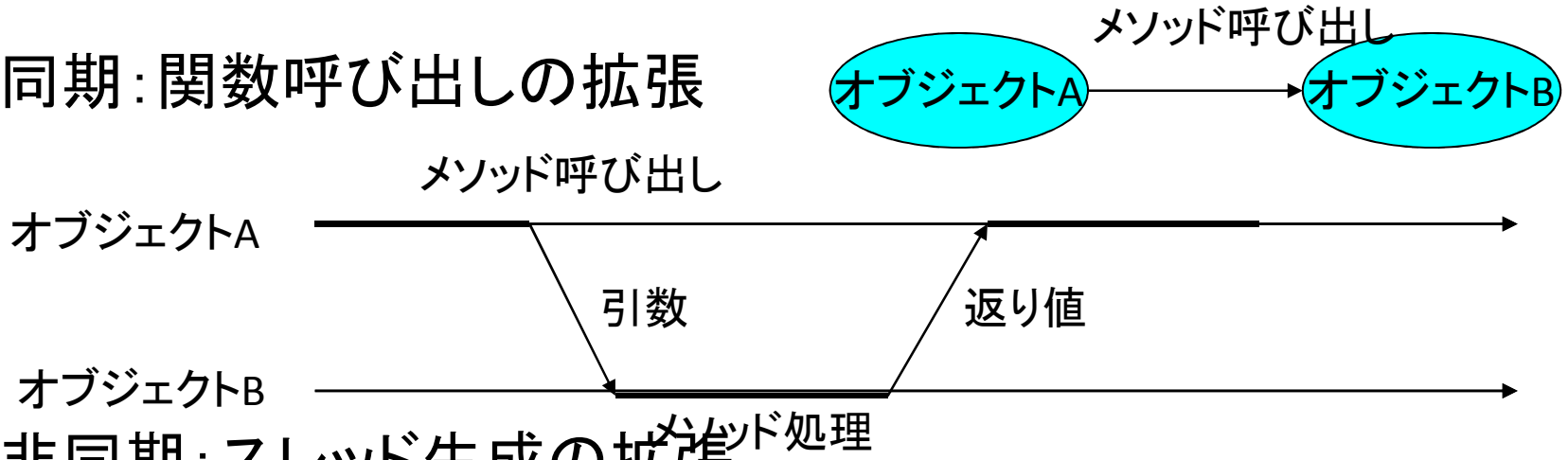
- 同期通信



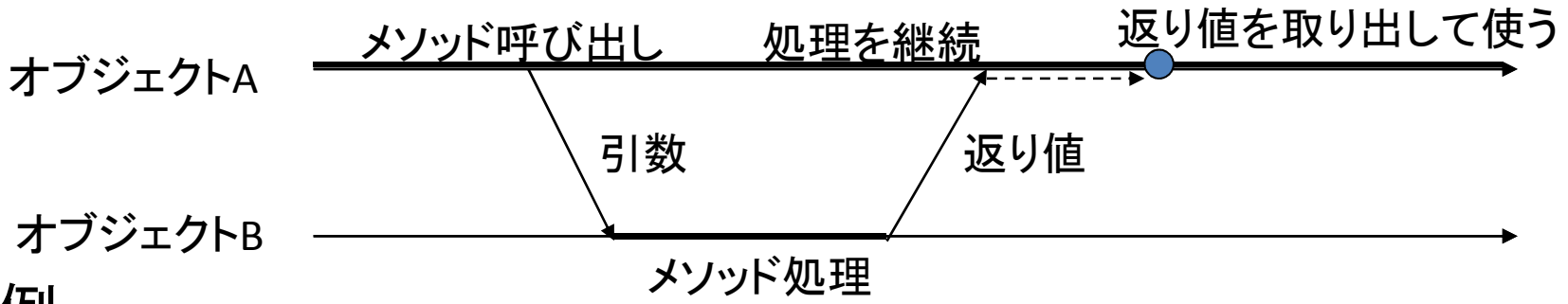
- 非同期通信

通信（リモートメソッド呼び出し）

- 同期：関数呼び出しの拡張



- 非同期：スレッド生成の拡張



- 例
  - CORBA (Common Object Request Broker Architecture)
  - Java RMI (Remote Method Invocation)

ノード間の接続

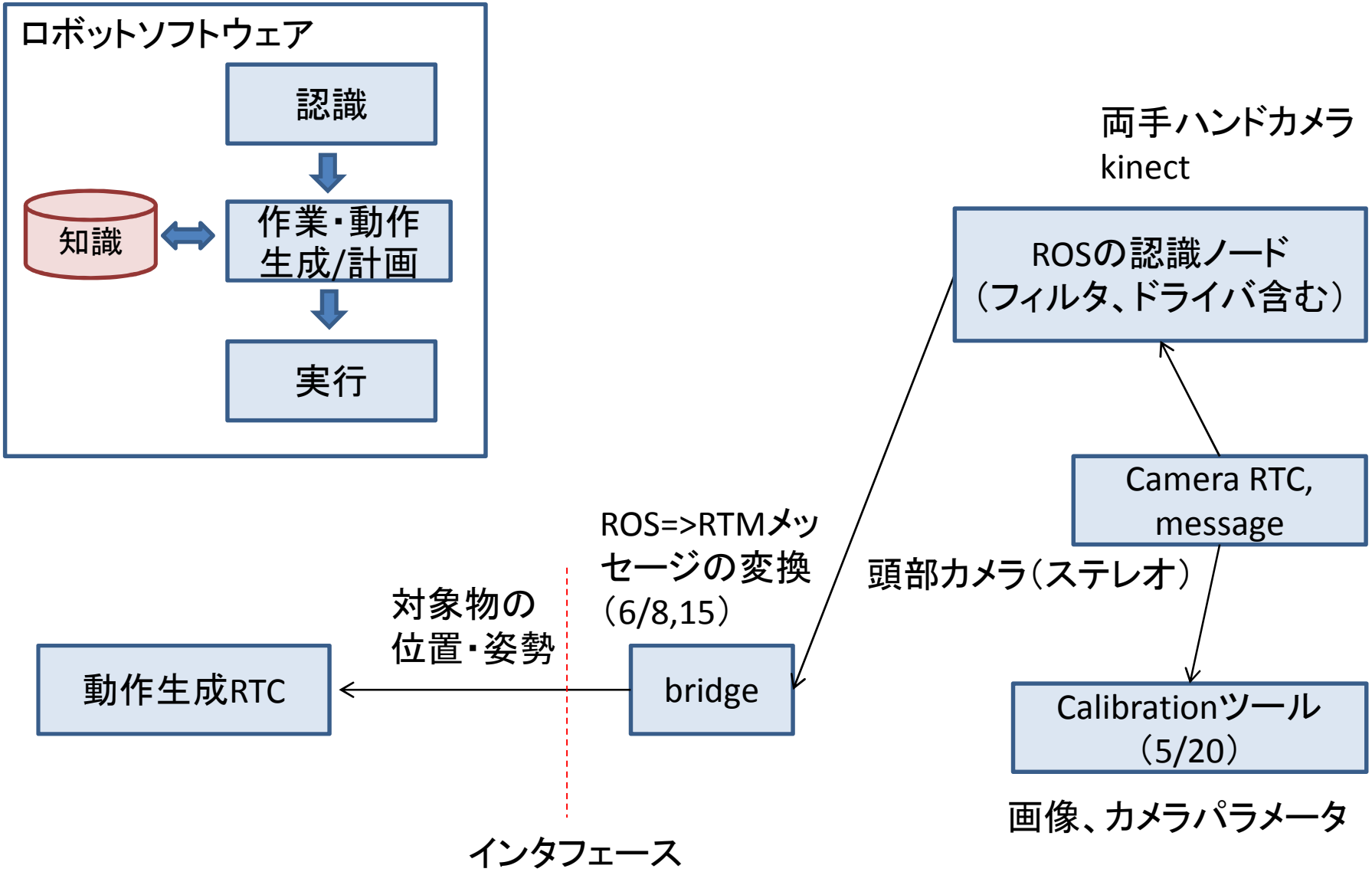
- ROS: 名前、rename
- RTM: 明示的接続

- rtshell, rtc-handle, rtm.py, etc.

HIRO-NXの場合

- ベースはRT-middlewareで提供される
  - HIRO-NXのハードウェアまわり、シミュレータ(OpenHRP3)
- 認識、動作計画等の上位層のRT-componentが今後提供される
- 必要に応じてROSのパッケージを利用する

こういう構成を作る


ロボットソフトウェア
認識
知識
作業・動作
生成/計画
実行
両手ハンドカメラ
kinect
ROSの認識ノード
（フィルタ、ドライバ含む）
Camera RTC,
message
頭部カメラ（ステレオ）
ROS=>RTMメッ
セージの変換
（6/8,15）
対象物の
位置・姿勢
動作生成RTC
bridge
Calibrationツール
（5/20）
画像、カメラパラメータ
インタフェース

パッケージ構成図

- ソース
  - svn co http://rtm-ros-robotics.googlecode.com/svn/trunk/agentsystem_hironx_samples
- 環境設定
  - Export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:<rtm-ros-robotics>/agentsystem_hironx_samples
- コンパイル（RTM, ROS等はインストール済みという前提）
  - roscd iv_plan
  - cd externals; make
  - roscd iv_plan; rosmake
  - （ar_poseは標準でないかもしれない。その場合、とりあえずmanifest.xmlから除く）

ディレクトリ構成

agentsystem_hironx_samples

- /iv_plan
  - メイン
  - /externals
    - 外部ライブラリ
    - ロボットモデル
    - VPython最新版（可視化）, ikfast, PQP
- /iv_scenario
  - シナリオ関係を入れる予定
- /iv_idl
  - HIRO-NXシステムで必要なIDLファイル群
- /iv_sense
- /iv_bridges
  - 認識・ROSとRTMの仲介
- /rmrc_geo_model
  - 座標表現ライブラリ
- /rtc_handle
  - Pythonから対話的にRTC管理を行うツール

認識部分

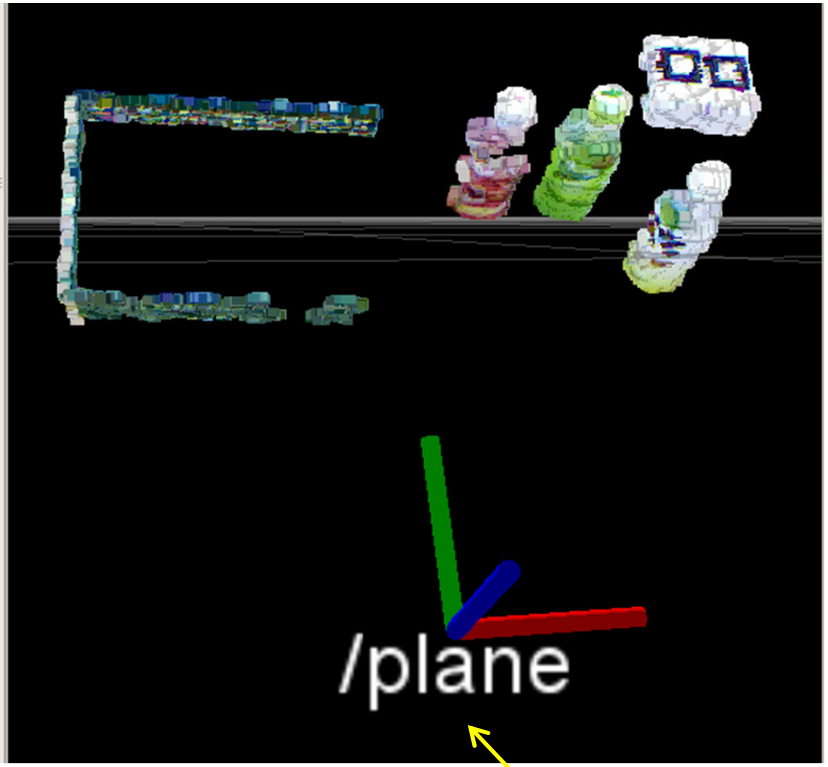
- ROSパッケージを利用
  - 導入が容易

- 切り換えが容易なのが分散ミドルウェアの利点
- RTMとROSを連携させるのが本講義の目的

- 今後、RT-componentに置き換えながら柔軟に運用していく予定
  - OpenVGR[産総研]など

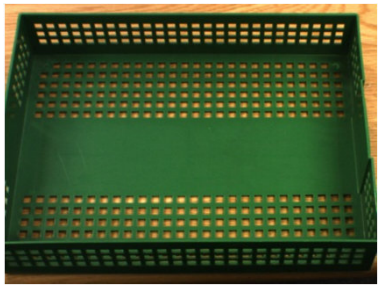
Kinect搭載

現在はROSで利用
OpenNIノード
⇒ Point Cloud Library(PCL)でフィルタ
⇒ 平面検出（JSKのROSパッケージ）で
テーブル除去



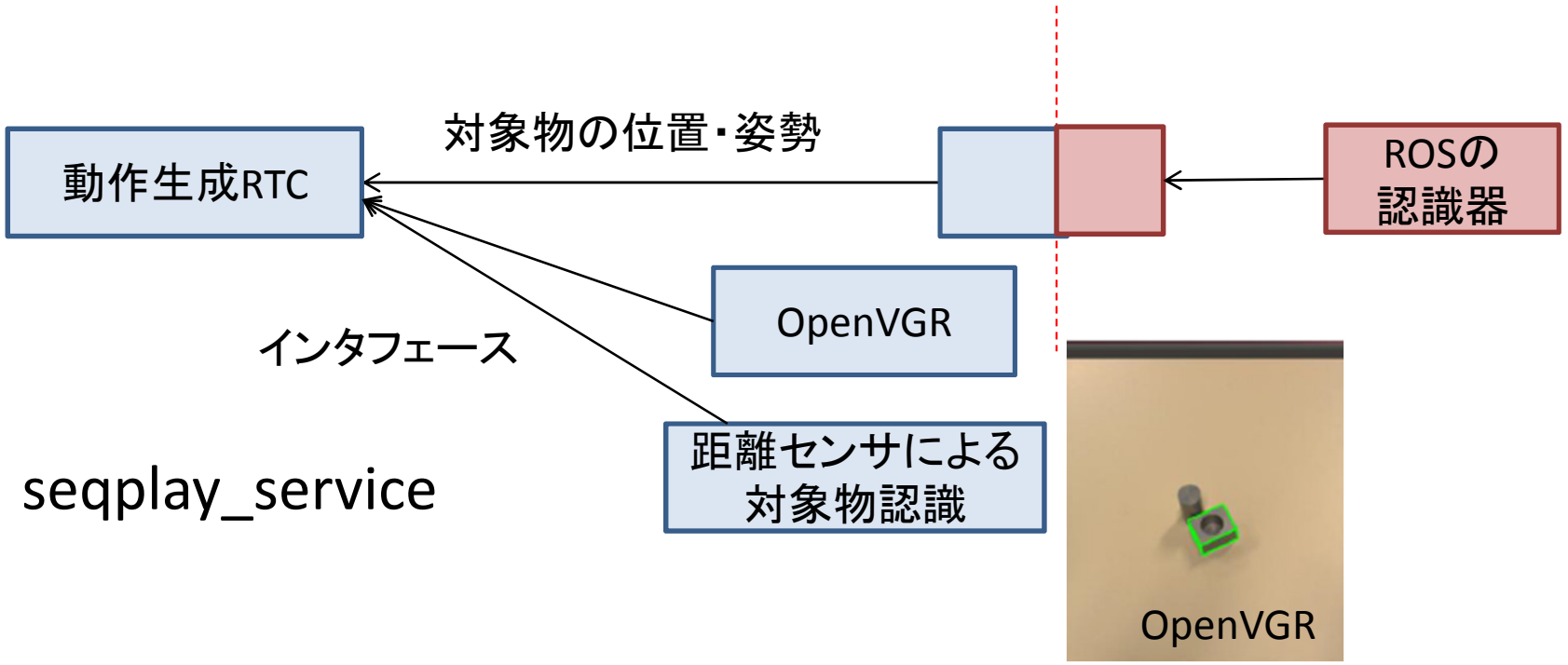
検出したテーブル面の向き

HIRO-NX頭部カメラ
Kinectの方がかなり広角



通信の橋渡し（bridge）

- 前回まで
  - 速度指令や点群の変換
- 認識部分と動作生成部分
  - ここは未実装
  - 今は直接ROSメッセージ, AR_Markers, tf, Poseを読んでいる
- seqplay_service



作業/シナリオ記述

- スクリプトで書きたいことが多い（トップレベル）
- 分散システムではタスク実行はリモートメソッド呼出しで書くのが自然
- 対話環境からRT-componentを操作する機能が必要
  - RTCの接続
  - RTCの状態遷移
  - RTCのサービス呼出し
  - RTCとのデータポート送受信

ツール

- Rtshell[Biggs、産総研]
  - python
- rtm.py (OpenHRP3、HIRO-NX)[金広、産総研]
  - jython
- Rtc-handle[末廣、電通大]
  - http://staff.aist.go.jp/t.suehiro/rtm/rtc_handle.html
  - python
- SDL[大橋、九工大]
  - Java

- 少しずつ異なる
  - 機能、実行環境、言語

- 他のツールとの組合せ、高品質のライブラリの充実度からPythonが便利
  - OpenRTM-aist-python, ROS, numpy, scipy, etc.

パッケージ管理

- なぜ必要か？

- どんなことが実現されているか？

- ツール
  1. ros_build
  2. rtm_build
  3. RTMExt

  HIRO-NXではROSのパッケージ管理、ビルドツールを利用

動作生成RTC概要

- コンセプト
  - Pythonによるコンパクトな実装
  - できるだけPythonで標準的に使われているパッケージを使う
- 概要
  - Pythonシェル上での対話的動作生成、要素機能の統合
  - 可視化（VPython）
  - 座標系表現（分解運動速度制御RTC[末廣'09]の一部）
  - 逆運動学計算（IK-fast[Diankov]）
  - RRT-connectによる動作計画
  - 干渉チェック（三角メッシュ（PQP）といくつかのprimitive）
  - VRML表記されたロボットモデルのロード
    （plyでパーサを記述）
- RTC wrapping
  - 作業WG共通インタフェース

記述例

- Sampleで説明
  - pickbox.py
  - sample_handcam.py
  - hanoi0.py

RTCとして使う

- IDL

```
#include <ExtendedDataTypes.idl>
interface ArmMotionService
{
   typedef sequence<double> DblSequence;
   boolean MoveArm(in RTC::Pose3D goal, in double handWidth,
                   in string arm, in boolean useTorso, in boolean checkCollision);
};
```

- roscd iv_plan/src; ./MplanComp.py
- 別ターミナルで
- roscd iv_scenario/src; ipython test.py

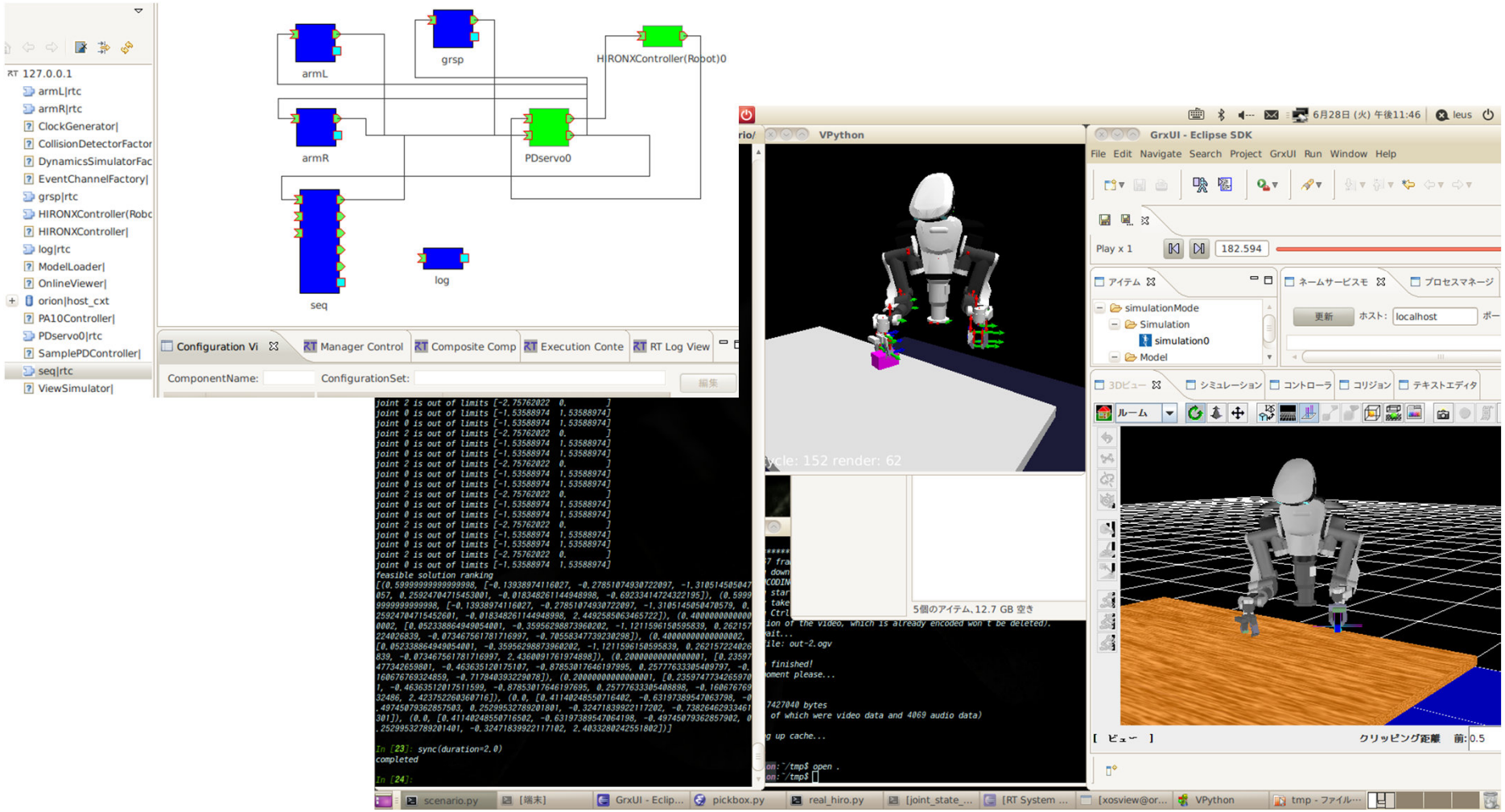
RTC-handleを用いたサービスを呼出し

- 起動
- Activate
- 使いたいサービスのポートを取得
- RTC:Pose3D型の目標値を生成
- サービス呼出し
  - frm=gen_goal_frm(y=-300)
  - plsvc.ref.MoveArm(frm, 100, ‘right’, False, False)

動作計画

- 簡易パレタイジング環境モデル
- RRT-connect

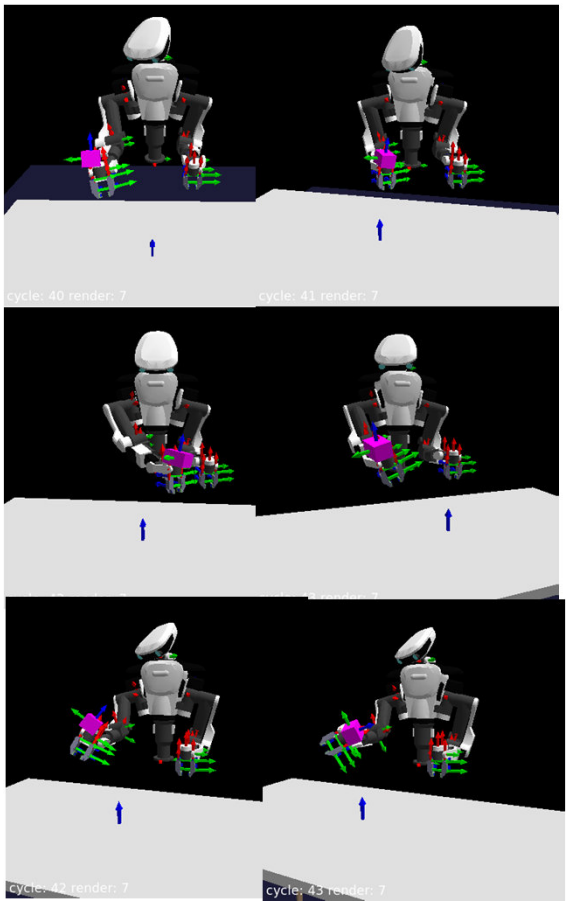
OpenHRP3の利用

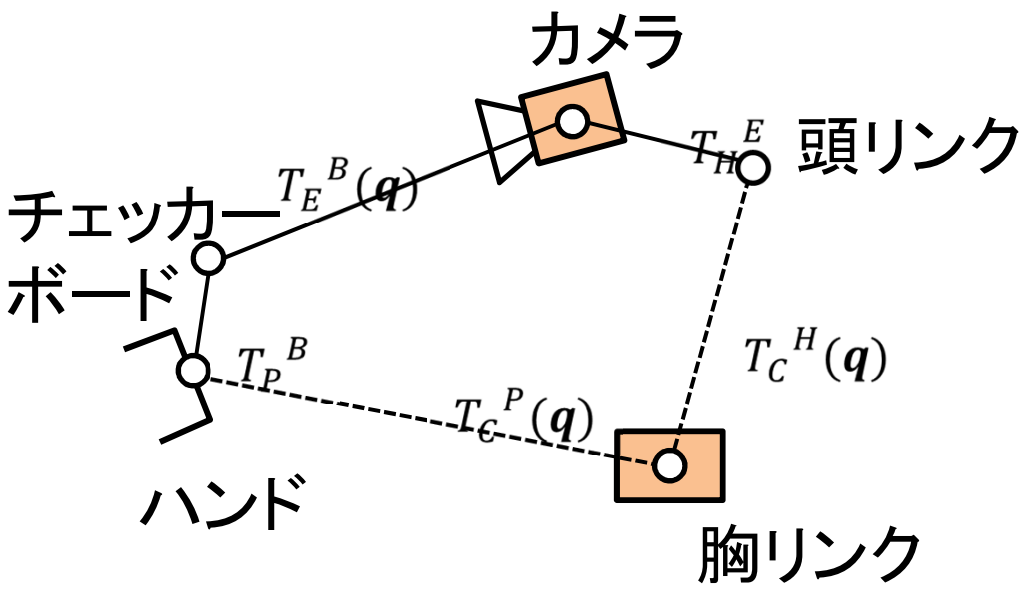


roslaunch iv_bridges seqplay_service.launch
roscd iv_plan/examples
ipython pickbox.py
ipython hanoi.py

Calibration

- カメラキャリブレーション
  - ROSのツール
- カメラ取り付け位置のキャリブレーション
  - ロボットモデルは十分精確と仮定
  - ハンドリンクにチェッカーボードを固定
  - 複数の姿勢でチェッカーボードを観察し、計12自由度を推定する





$$T_C{}^H(\boldsymbol{q})T_H{}^E T_E{}^B(\boldsymbol{q}) = T_C{}^P(\boldsymbol{q})T_P{}^B$$

$$\left\| T_C{}^H(\boldsymbol{q})\, T_H{}^E T_E{}^B(\boldsymbol{q}) - T_C{}^P(\boldsymbol{q})T_P{}^B \right\|$$

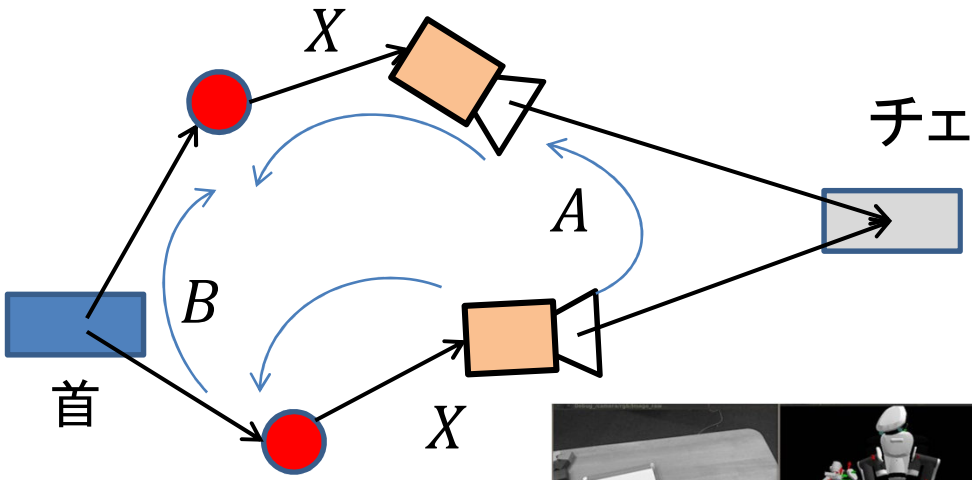
を最小化する$T_H{}^E$と$T_P{}^O$を求める

Calibration

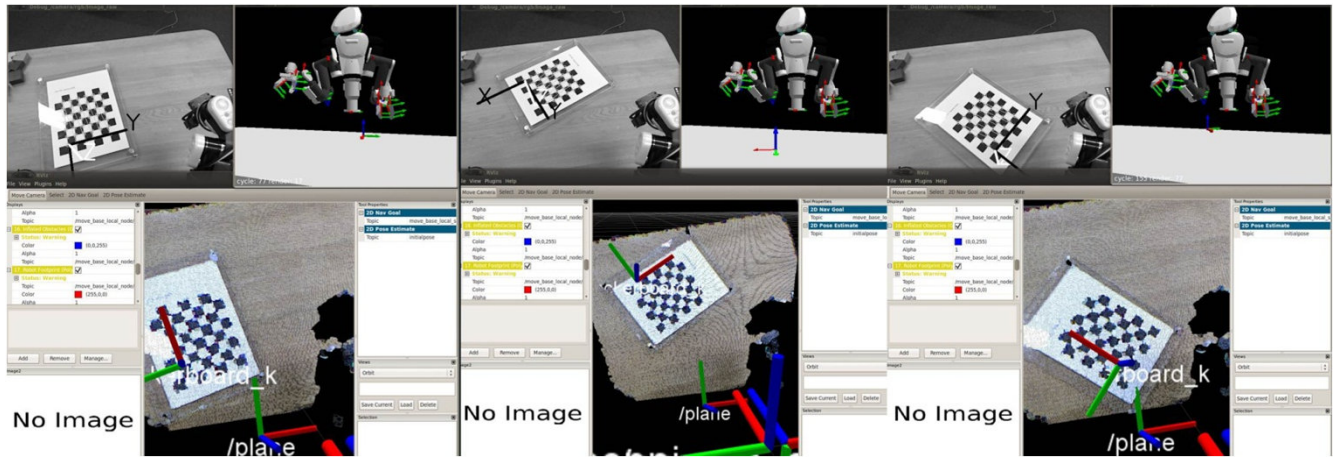
- ROSのカメラキャリブレーション
- ハンドアイキャリブレーション
  - 標準的手法

$$AX = XB$$

$$R_A R_X = R_X R_B$$
$$(R_A - \mathrm{I})t_X = R_X t_B - t_A$$



RGBカメラとdepthカメラの
オフセットはハンドチューン



まとめ

- RTM-ROS連携の事例
  - HIRO-NXの視覚から動作までをつなぐシステム
  - RTM-ROSで使えるものは使う（橋渡し、bridge）
  - DeploymentはROSベース（RTM-ROS連携の成果に移行中）
  - RT-component（RTC）は状態切り換え、接続が必要
  - 対話的環境からのRTCとの通信
    - Rtc-handleを利用
  - 動作生成
    - Pythonベースのシンプルなプログラム
    - RTCのインタフェースでwrapして使う