手先カメラを用いた双腕ロボットによる
マニピュレーションシステム
操作手順書

東京大学　情報理工学系研究科　稲葉研究室

平成 24 年 2 月 15 日

目 次

第1章　システム概要

本サービスは，工場での部品整理をイメージしたものである．具体的には手先のカメラを用いて作業台上の部品を認識し, 両手で箱に整理して入れる機能を実現します．手を動かすことで複数の対象物を認識，両手の干渉を考慮して同時にアプローチできる対象物を選択します.

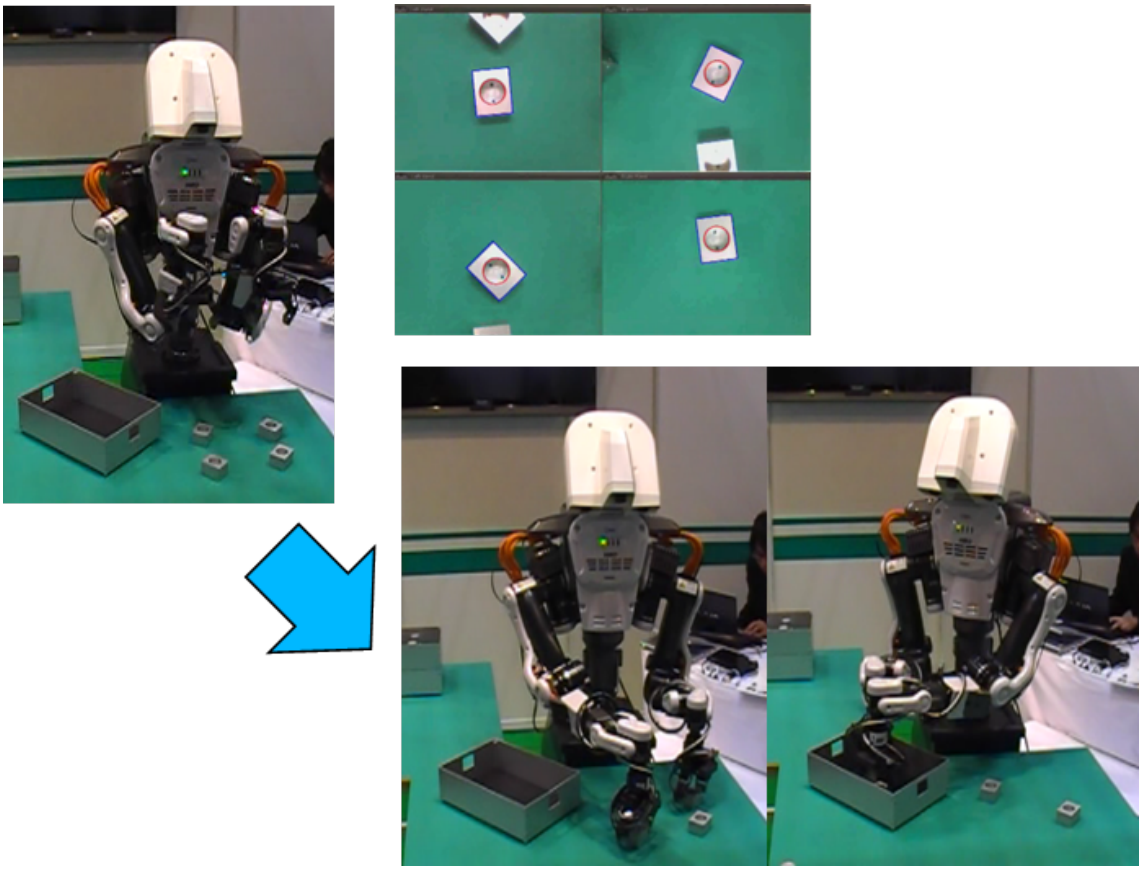


図 1.1: サービスイメージ

1.1　全体のモジュール構成

以下に，本システムで利用するモジュールの一覧を示します．これらを入手し，実行可能な環境を整えてください.

- app-recog
- CameraComp
- (LoadPictureComp)

- iv_plan_hironx
- HiroNXInterface

ファイルシステム上の場所は必ずしも重要ではないですが，1つのディレクトリにこれらを置いておくと，本ドキュメントとあわせて理解しやすいです．これらのモジュールは，1台の計算機上で実行しても，動作生成部を別の計算機で実行してもどちらでも構いません．認識部は通信負荷を考えるとカメラがつながっているホスト上で実行することを推奨します．

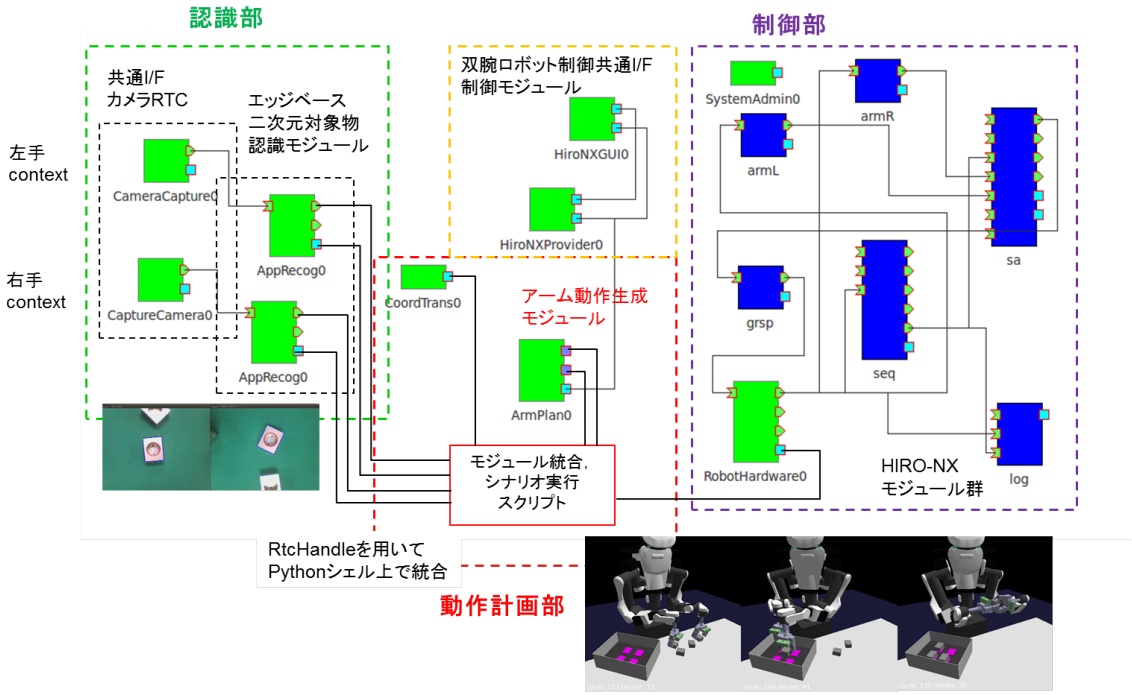


図 1.2: 全体のモジュール構成

1.2 認識部

1.2.1 エッジベース二次元対象物認識モジュール (AppRecog)

http://openrtm.org/openrtm/ja/project/NEDO_Intelligent_PRJ_HiroAccPrj_5002

HiroNX の手先に取り付けられた USB カメラで対象物を認識するために利用します．両手先のカメラそれぞれでカメラモジュールとともに実行します．使い方はモジュール付属のドキュメントを参照してください．

1.2.2 カメラ共通 I/F 準拠の画像キャプチャモジュール (CameraComp)

http://www-arailab.sys.es.osaka-u.ac.jp/CameraIF/

大阪大学により開発され画像キャプチャモジュール CameraComp をダウンロード，コンパイルします．利用方法は，本モジュールのドキュメントおよび，上記 AppRecog 付属のドキュメントを参照してください．

1.3 動作生成部

1.3.1 (VPython 版)HiroNX 動作生成モジュール

認識結果を世界座標への変換，HiroNX の動作の生成，タスク記述を行なうために利用します．詳しくは，モジュール付属のドキュメントを参照してください．

1.3.2 HiroNXInterface

http://www.openrtm.org/openrtm/en/node/4645

双腕ロボットの制御コマンドの共通インタフェースに準拠する HiroNX 用制御モジュールです．詳細については，開発元である産業技術総合研究所のドキュメントを参照してください．

第2章　準備

必要に応じてデモ環境，ロボットごとに以下の作業を行います．

2.1　力センサの有無，テーブルの高さ

手首に力センサがある場合とない場合で，ロードするモデルを変更する必要があります．サンプルプログラムは，力センサがある場合に手首のリンク長を変更，円筒形状を挿入したものです．力センサがない場合は，`HiroNX` インスタンスを生成するときに，オプション `forcesensors=False` を指定します．

また，シミュレータ内シーンはiv_plan/src/scene_objects.pyで定義されたPythonのデータ構造を解釈して生成されます．デフォルトで高さ700[mm]の東大仕様テーブル(table_scene())と高さ735[mm]のAIST仕様テーブル(table_scene_aist())が用意されています．それ以外のシーンを作成したい場合は，scene_objects.pyを参考に，シーン記述を作成し，env.load_scene()で環境にロードしてください．

2.2　キャリブレーション

ロボットの個体差を修正する作業です．カメラの内部パラメータおよび，ロボットのリンクに対するカメラの取り付け位置を計算します．デフォルト値はHiroNX14号機のものであり，精度を出すには各機体ごとに行う必要があります．

2.2.1　カメラのキャリブレーション

RT-middlewareのコンポーネントでもROSのノードでもよいので単眼カメラのキャリブレーションを行い，CameraCompが読み込めるようにします．本システムにおいて，run.shスクリプトでRTCを起動する場合，scriptsディレクトリにあるcamera_left.ymlおよびcamera_right.ymlにそれぞれのカメラの内部パラメータを記述します．その後，エッジベース二次元対象物認識モジュールにおいて，対象物の位置，距離が正しく出力されているかどうか確認しておいてください．キャリブレーションを行なうときは，以下の関数を実行して手先カメラを正面に向ける(図2.1)と作業が容易になります．

```
$ ipython
>>> from hironx_calib import *
>>> handcam_calib_pose()
>>> sync()
```

図 2.1: カメラキャリブレーション用の姿勢

2.2.2 カメラ取り付け位置のキャリブレーション

次に，カメラの取り付け位置を計算します．キャリブレーションプログラムは，学習データとしてロボットの各姿勢におけるチェッカーボードの姿勢を入力とします (図 2.2)．チェッカーボードの認識には ROS の checkerboard_detection ノードを使用し，tf として publish されたチェッカーボードの姿勢を python プログラムで受信します．具体的な手順は，付録に記載します．

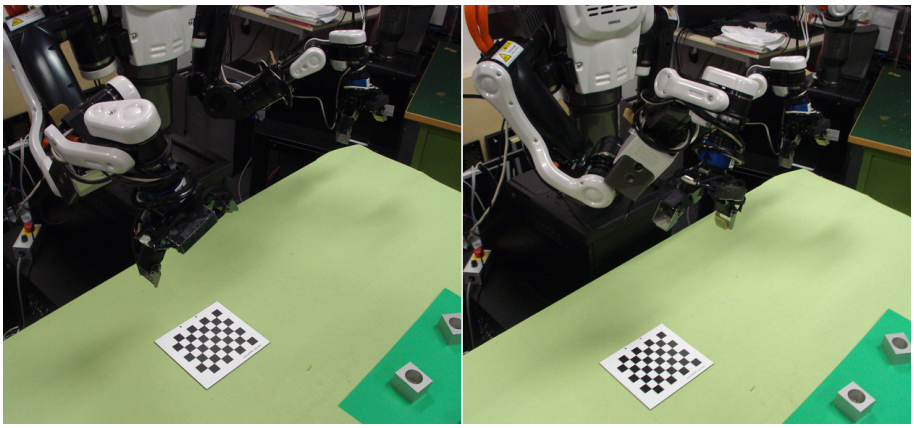


図 2.2: 手先を動かしたチェッカーボードの検出

第3章　デモの実行

3.1　制御系モジュールの起動と接続

```
$ cd iv_plan/scripts
$ ./run-hironx-interface.sh
```

これにより，両手に対応するカメラ，認識モジュール，HiroNXInterface 制御モジュールが起動，接続され，さらに各モジュールが activate されます．HiroNXInterface モジュールは起動後 GUI 上の「RTC Status」が緑になった状態で「Set up Robot」ボタンを押し，RTC まわりの初期化を行う必要がある点に注意してください．起動した GUI から以下を実行しておきます．

1. ジョイントキャリブレーション
2. 初期姿勢へ移行
3. 指のサーボオン　(ついでに開閉動作確認)

3.2　認識系モジュールの起動と接続

次に，HiroNX 両手の手先カメラのキャプチャ及び，AppRecog 認識モジュールを起動します．左右で同じ名前のコンポーネント群を起動するため，別々の RTC 名前空間に起動します．

```
$ ./run.sh
```

ここで，左右のカメラが逆になっていないかどうか確認してください．逆になっている場合は，CaptureCameraLeft.conf と CaptureCameraRight.conf に記述してある conf.default.int_camera_id の番号を逆にして，run.sh を再実行してください．左右それぞれのモジュール群は起動した端末で Ctrl+c を押すことで終了します．

iv_plan/scripts にあるスクリプトは，ローカルのファイルシステム非依存な方法でモジュールの場所を検出するため，ROS のコマンドである rospack を使います．rospack を利用できない場合は，適宜，スクリプトを編集するか同等の機能を実現するコマンドに置き換えてください．もし，コンポーネントが上手く接続および activate されない場合は，./comcon.sh および./comact.sh を再度実行してみてください．

カメラが認識できていない可能性がある場合には，xawtv コマンドなどでカメラ自体が認識されているか確認してください．認識関係のモジュールを登録するネームサーバは scripts ディレクトリの rtc_left.conf および rtc_right.conf で指定します．

3.3　動作生成系モジュールの起動とモジュール間通信の確認

以下のコマンドでデモプログラムを起動し，

- HiroNX 本体のモジュールと通信し，関節角度の取得および，動作コマンドの送信を行なうことができること
- 認識器と通信し認識結果を取得できること

を確認します.

```
$ cd iv_scenario/src
$ ipython demo_wexpo.py

>>> rr.get_joint_angles() # 実機の関節角度を取得
>>> r.prepare()           # シミュレータのモデルの姿勢を変更
>>> sync()                # 実機をモデルにあわせる(実機が動くので注意)

図???のように，認識できる位置に対象物を置いて
>>> rr.detect(sensor='rhandcam') # 右手ハンドカメラでの認識
>>> rr.detect(sensor='lhandcam') # 左手ハンドカメラでの認識

認識した後，その結果を世界座標への変換する
>>> detect(sensor='rhandcam')
>>> detect(sensor='lhandcam')

>>> f = detect(sensor='rhandcam')
>>> show_frame(f) # 認識結果の表示
# シミュレータ内で認識位置に箱A0を移動させる
>>> env.get_object('A0').locate(f)
```

モジュールと通信できない場合は，iv_scenario/src にある rtc.conf で正しくネームサーバが指定されているか，また，iv_scenario/interface_wexpo.py に書かれている各モジュールのパスが正しいかどうか確認してください.

3.4 デモの実行

図 3.1 のように作業台に対象物を 4 つ並べ，作業台上で両手を動かし，対象物を検出できるかどうかを確認してみます.

```
>>> look_for()
```

上手く検出できれば，シミュレータ内の箱 4 つが正しい位置に移動します (図 3.2(a)). (b) では 1 つ，(c) では 2 つ認識に失敗しています．上手く検出されないときは，画像中で対象物が正しく認識されているかチェックしてください.

検出時の初期位置へロボットの姿勢を持っていきたいときには, 以下のコマンドを実行します.

```
>>> go_scan_pose()
```

これで準備が整ったので，実際にデモを実行します.

```
>>> demo()
```

作業台をスキャンし，対象物を 4 つ検出した後，最初は両手で，残りの 2 つは適宜持ち替えを行なって箱に配置します．対象物を 4 つ検出できなかったときは，動作には入りません.

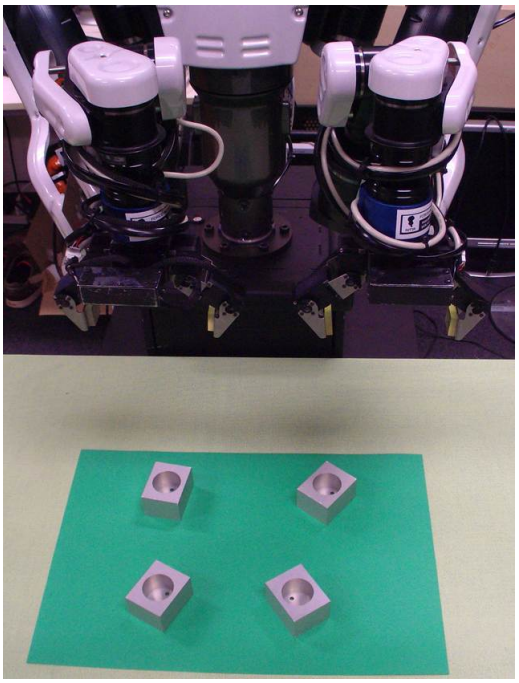
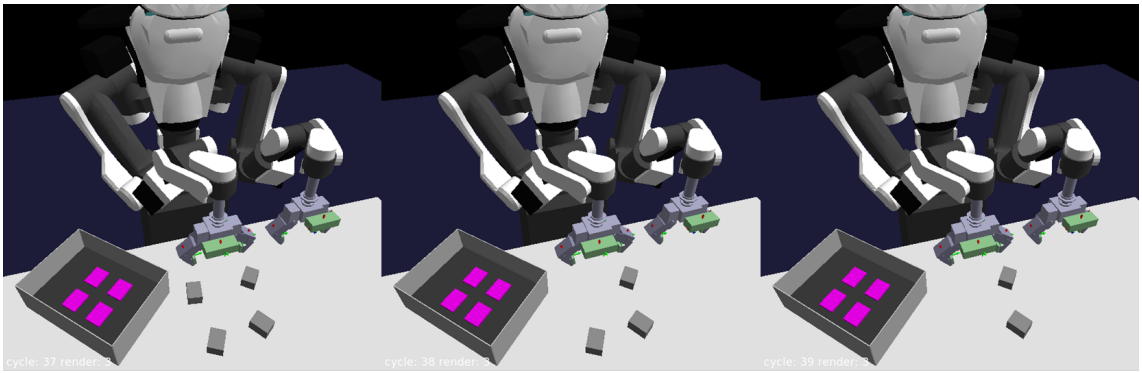


図 3.1: ４つの部品を並べた状態



(a) (b) (c)

図 3.2: 作業台上の対象物認識結果

付 録A　キャリブレーションメモ

checkerboard_detection および，camera_calibration パッケージを用意してください．ROS ノード起動用の launch ファイルセットのパッケージ Sense.tgz を使います．

A.1　カメラキャリブレーションの手順

```
$ roslaunch Sense rhand.launch
$ rosrun camera_calibration cameracalibrator.py --size 7x10 --square
0.025 image:=/rhand/usb_cam/image_raw
```
Sense/launch/rhand.launch の内部パラメータを更新する
```
$ roslaunch Sense lhand.launch
$ rosrun camera_calibration cameracalibrator.py --size 7x10 --square
0.025 image:=/lhand/usb_cam/image_raw
```
Sense/launch/lhand.launch の内部パラメータを更新する

A.2　カメラ取り付け位置キャリブレーションの手順

1. ROS のカメラノードにより画像および上記カメラパラメータがトピックとして publish され，checkerboard detection ノードでそれらを subscribe し，推定した tf が publish されるようにします．
   `$ roslaunch Sense rhand\_checkerboard.launch`
2. 机の上にチェッカーボードを置きます
3. 手を動かして学習データを取ります．`$ cd iv_scenario/src`
   ```
   $ ipython
   >>> from hironx_calib import *
   >>> rr.connect()
   >>> res = record_data(sensor='rhandcam')
   ```
   すべての姿勢で，チェッカーボードが視野に収まり，安定して認識できていることを確認します (図 A.1)．
4. 手首リンクからカメラへの座標変換を計算します．
   `>>> f = calibrate(res, link='RARM_JOINT5_Link', tf0=r.Trh_cam)`
5. 結果を確認します．
   `>>> play_data(res, link='RARM_JOINT5_Link', tf=f)`
6. 問題なければ hironx_params.py の Trh_cam を更新します．
7. 左手カメラについても同様に行います．

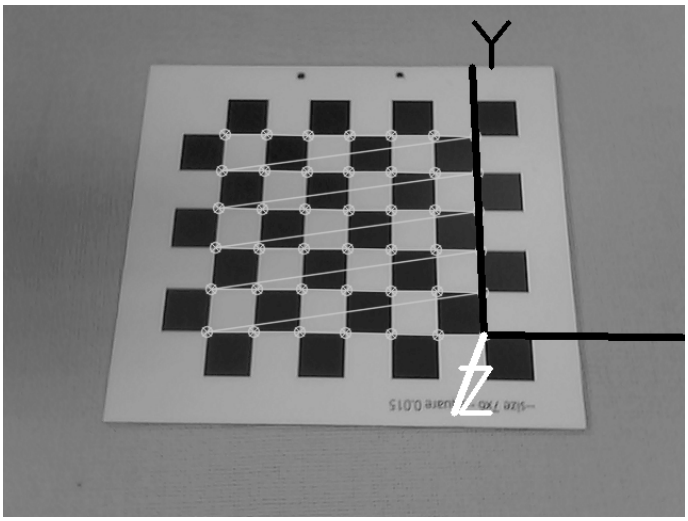


図 A.1: checkerboard detector