エージェントシステム
RTM/ROS相互運用
RTM第三回
2011/6/15

情報システム工学研究室
特任講師　吉海智晃

今日のトピック

- 独自データ形式によるデータ通信
  - 基本型，拡張型，独自型の定義
  - マルチキャプチャ環境を想定したカメラ用IFのIDL
    Img.dllを利用したデータ通信
- RTMとROSの相互接続利用
  - オープンソースロボット用スクリプト言語EusLispを利用した実現(rtmeus/roseus)
- RTM事例紹介
  - 富士通ビジョンボード

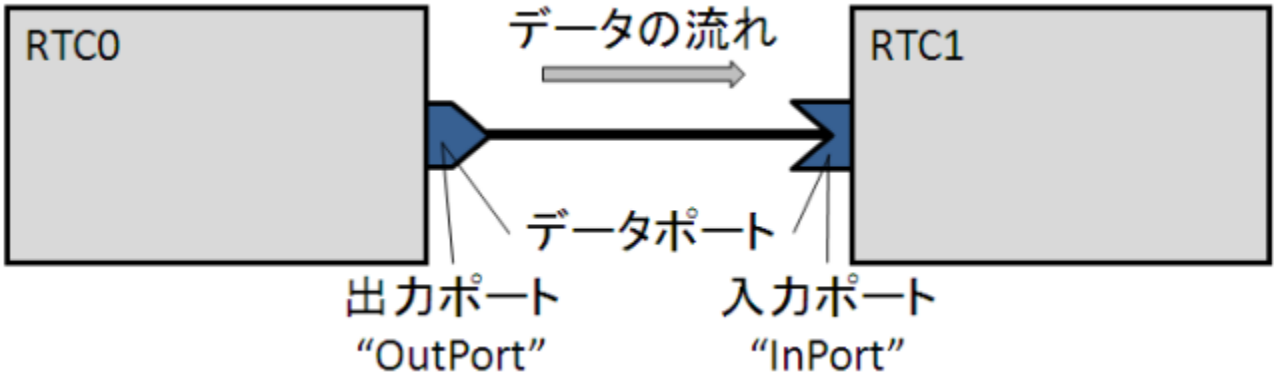
今日のトピック

- 独自データ形式によるデータ通信
  - 基本型，拡張型，独自型の定義
  - マルチキャプチャ環境を想定したカメラ用IFのIDL
    Img.dllを利用したデータ通信
- RTMとROSの相互接続利用
  - オープンソースロボット用スクリプト言語EusLispを利用した実現(rtmeus/roseus)
- RTM事例紹介
  - 富士通ビジョンボード

データポートで利用可能な組込型

- InPort/OutPort: 連続的データ授受用のインタフェース
  - **データ入力ポート(InPort)**
  - **データ出力ポート(OutPort)**

Official webドキュメントより転載



- **基本型**
  - **タイムスタンプ+データ単体**
    - **TimedShort, TimedLong, TimedDoubleなど**
  - **タイムスタンプ+データシーケンス**
    - **TimedShortSeq, TimedLongSeq, TImedDoubleSeqなど**

データポートで利用可能な基本型一覧

/usr/include/rtm/idl/BasicDataType.idl（或いは/usr/local/include/rtm/idl/）に定義されている

TimedShort / TimedShortSeq	タイムスタンプと short int 型
TimedUShort　/ TimedUShortSeq	タイムスタンプと unsigned short int 型
TimedLong / TimedLongSeq	タイムスタンプと long int 型
TimedULong / TimedULongSeq	タイムスタンプと unsigned long int 型
TimedFloat　/ TimedFloatSeq	タイムスタンプと float 型
TimedDouble / TimedDoubleSeq	タイムスタンプと double 型
TimedString / TimedStringSeq	タイムスタンプと string 型
TimedWString / TimedWStringSeq	タイムスタンプと wstring 型
TimedChar / TimedCharSeq	タイムスタンプと char 型
TimedWChar / TimedWCharSeq	タイムスタンプと wchar 型
TimedOctet / TimedOctetSeq	タイムスタンプと バイト 型
TimedBool / TimedBoolSeq	タイムスタンプと bool 型

Official webドキュメントのものを改変して転載

基本型の使い方（C++）

- ヘッダファイルのクラス定義中の記述

```
protected:
  TimedLong m_out;
  OutPort<TimedLong> m_outOut;
```

- Cppファイルのクラスコンストラクタの記述

```
ConsoleIn::ConsoleIn(RTC::Manager* manager)
    : RTC::DataFlowComponentBase(manager),
      m_outOut("out", m_out)
{
```

- CppファイルのonInitialize()の記述

```
addOutPort("out", m_outOut);
```

後は，OnExecute()にて前回講義のサンプルの通りに記述すれば良い

基本型の使い方（Python）

- クラス定義内のOnInitialize(self)の記述

  self._data = RTC.TimedLong(RTC.Time(0,0),0)

  self._outport = OpenRTM_aist.OutPort("out", self._data)

  self.addOutPort("out", self._outport)

  後は，OnExecute(self)にて前回講義のサンプルの
  通りに記述すれば良い

データポートで利用可能な拡張型

- /usr/include/rtm/idl/ (or /usr/local/include/rtm/idl)
  以下のExtendedDataTypes.idl，InterfaceDataTypes.idl
  に定義がある
- C++では，
  - #include <rtm/idl/InterfaceDataTypesSkel.h>
  をして，後は基本型と同じように使える
- Pythonでは，特別なものをインポートする必要はない（基本型と同様，'import RTC' があれば良い）

Pythonで拡張型を使う場合の例

- TimedPoint2D(/usr/include/rtm/idl/ExtendedDataTypes.idl)

```
struct Point2D
{
   double x;
   double y;
};
struct Timed Point2D
{
   Timed tm;
    Point2D data;
};
```

- OnInitialize(self)の記述

```
p2d = RTC.Point2D(0, 0)
self._data = RTC.TimedLong(RTC.Time(0,0), p2d)
self._outport = OpenRTM_aist.OutPort("out", self._data)
self.addOutPort("out", self._outport)
```

- OnExecute(self)の記述

```
self._data.data.x = 10
self._data.data.y = 20
self._outport.write()
```

独自型を利用する場合（C++)

- 基本的には，Idlを用意して，omniidlにてコンパイルする
  - xxx.idlの中に
    - #include <BasicDataType.idl>
      の1行を含める
    - RTC::Time tmを含むTimedxxx
      というデータ型にする
  - xxx.idlからxxx.hhを生成し，ヘッダファイルに読込む
  - xxx.idlからxxxSkel.oを生成し，実行ファイル，共有ライブラリにリンクする

RTCBuilder雛形から独自型を使う（C++）

- idltest.idlを読み込ませる場合，雛形Makefile中のSKEL_OBJにオブジェクトファイルを指定

  SKEL_OBJ = idltestSkel.o

- Makefileに以下の記述を追加

```
idltestSkel.cpp : idltest.idl
	$(IDLC) $(IDLFLAGS) $<
	$(WRAPPER) $(WRAPPER_FLAGS) --idl-file=$<
idltestSkel.h : idltest.idl
	$(IDLC) $(IDLFLAGS) $<
	$(WRAPPER) $(WRAPPER_FLAGS) --idl-file=$<
idltestSkel.o: idltestSkel.cpp idltestSkel.h idltestStub.h
	$(CXX) $(CXXFLAGS) -c -o $@ $<
```

CORBAでサーバとクライアントを接続する仕組みとして，omniidlはidlからスケルトン(Skel)とスタブ(Stub)のコードを作ってくれる.
サーバ側がSkel, クライアント側がStubだが今は，どちらをリンクしても良い.

独自型を利用する場合(Python)

- idlからomniidlを使ってpython用のコードを生成

  omniidl -bpython -I/usr/include/rtm/idl ./xxx.idl

BasicDataType.idlがある場所を-Iで指定することが必要

- xxx_idl.pyとxxx/, xxx__POA/が生成されるのでコードの中でxxxをimportすれば良い．サービスを使う場合はxxx__POAも必要

例えば，idltest.idlの場合

import Idltest

として，

idmsg = Idltest.Idmsg(0, "None")

self.RecvMsg = Idltest.TimedIdmsg(RTC.Time(0,0), idmsg)

のようにできる

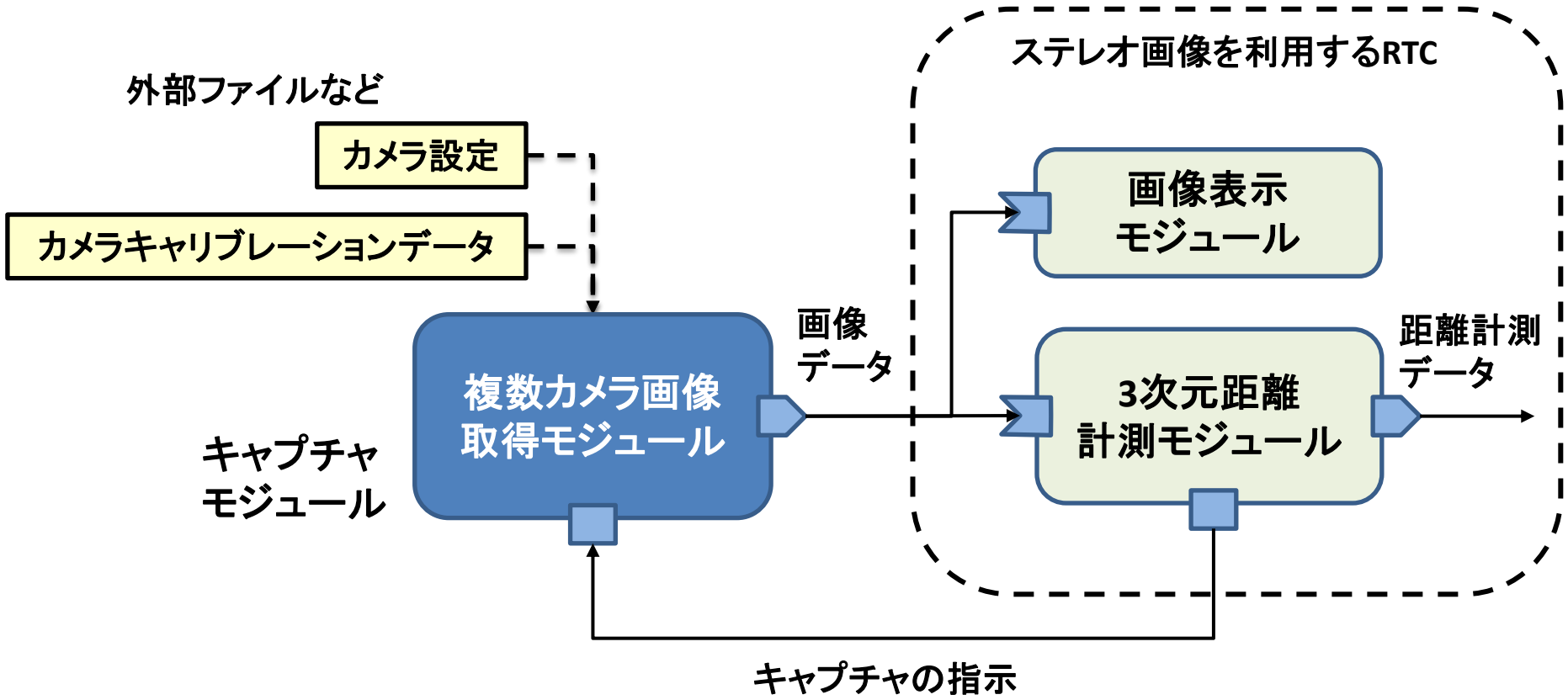
さらに詳細は，wikiのサンプルにありますので，そちらを参照してください

今日のトピック

- 独自データ形式によるデータ通信
  - 基本型，拡張型，独自型の定義
  - マルチキャプチャ環境を想定したカメラ用IFのIDL
    Img.dllを利用したデータ通信
- RTMとROSの相互接続利用
  - オープンソースロボット用スクリプト言語EusLispを利用した実現(rtmeus/roseus)
- RTM事例紹介
  - 富士通ビジョンボード

独自型を利用した通信コンポーネントサンプル

- マルチキャプチャ環境を想定したカメラデータ通信用データ型（次世代知能化プロジェクトにて検討中）



データポート通信に用いるデータ型（Img.idlより抜粋）

```
struct CameraImage
  {
    RTC::Time captured_time;                    キャプチャされた時間
    ImageData image;                            画像生データ
    CameraIntrinsicParameter intrinsic;         カメラ内部パラメータ
    Mat44 extrinsic;                            カメラ外部パラメータ
  };
struct MultiCameraImage
  {
    sequence<CameraImage> image_seq;            カメラ画像データのシーケンス
    long camera_set_id;                         カメラID
  };

struct TimedMultiCameraImage
  {
    RTC::Time tm;
    MultiCameraImage data;
    long error_code;
  };
```

サービスポートからのキャプチャ指示(Img.idl)

- 複数カメラ画像取得のインタフェース(CameraCaptureService):
  - 1枚キャプチャ

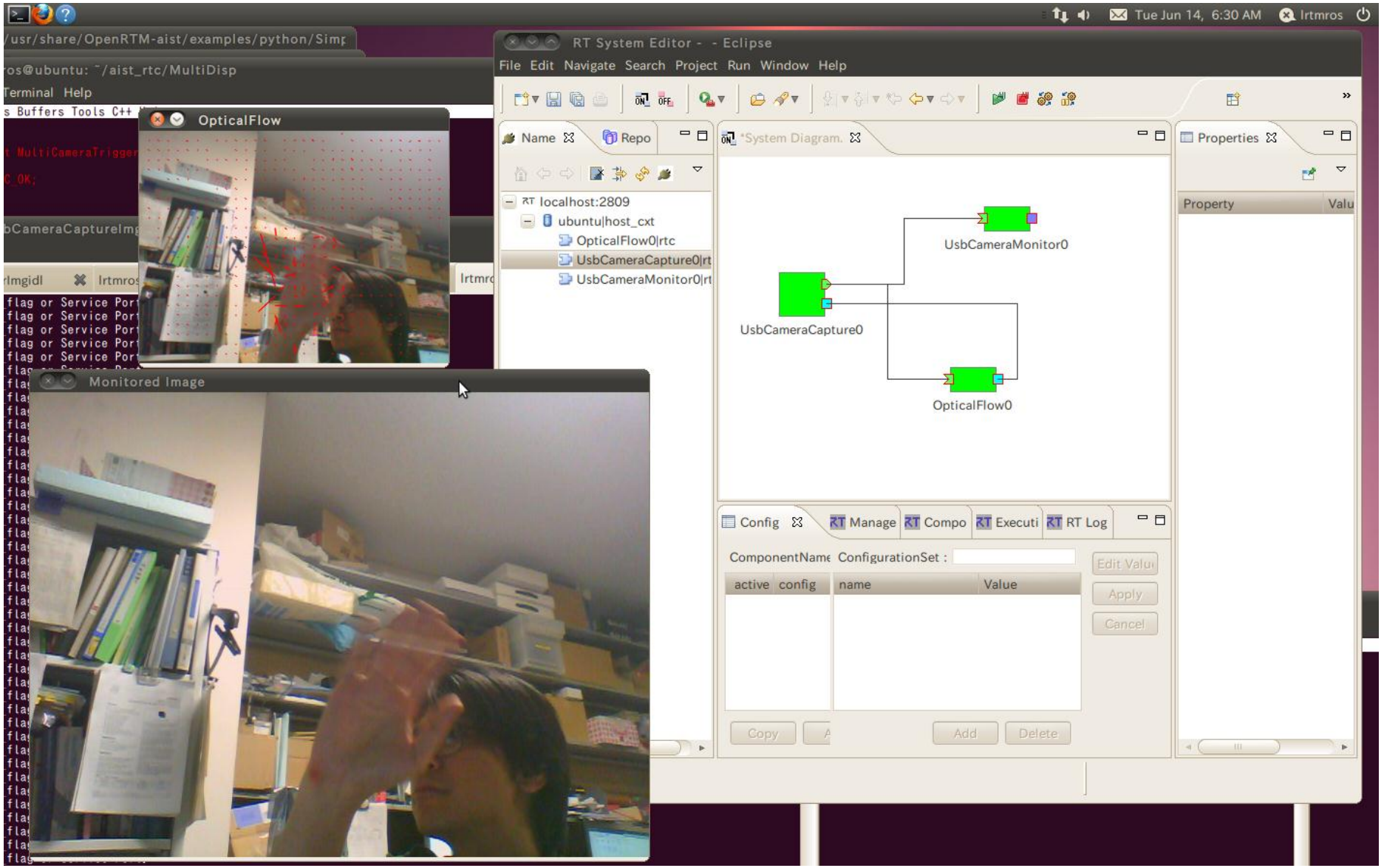
    oneway void take_one_frame()
  - num枚キャプチャ

    oneway void take_multi_frames(in long num)
  - 連続撮影開始

    oneway void start_continuous();
  - 連続撮影停止

    oneway void stop_continuous();

サンプルを動かしてみよう



今日のトピック

- 独自データ形式によるデータ通信
  - 基本型，拡張型，独自型の定義
  - マルチキャプチャ環境を想定したカメラ用IFのIDL Img.dllを利用したデータ通信
- RTMとROSの相互接続利用
  - オープンソースロボット用スクリプト言語EusLispを利用した実現(rtmeus/roseus)
- RTM事例紹介
  - 富士通ビジョンボード

RTMとROSの相互接続利用方法

- 両方の口を備えたコンポーネント/ノードによる実現
  - ROSシステム側からの構築： 6/8 ROS(3)
  - Pythonで双方の必要なファイルをimportして書く
- ROSトランスポートの利用による実現
- オープンソースロボット用スクリプト言語EusLispを利用した実現
  - 双方のインタフェースであるrtmeus, roseusを介した相互接続

ROSトランスポート

- 産総研のジェフさんが開発，公式ページで公開している

http://www.openrtm.org/openrtm/ja/content/ros%E3%83%88%E3%83%A9%E3%83%B3%E3%82%B9%E3%83%9D%E3%83%BC%E3%83%88rosport

- RTM側から
  - アウトポート：ROS側へのtopicのpublish
  - インポート: ROS側からのtopicのsubscribe

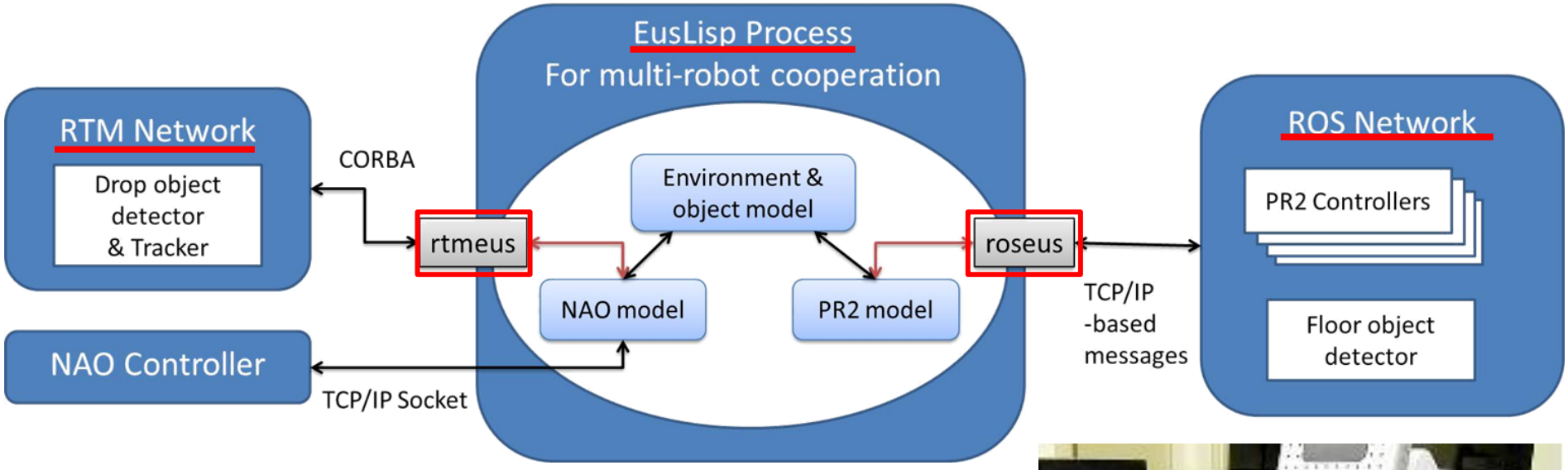
  として見える.
  - #include <rtm/RosOutPort.h>

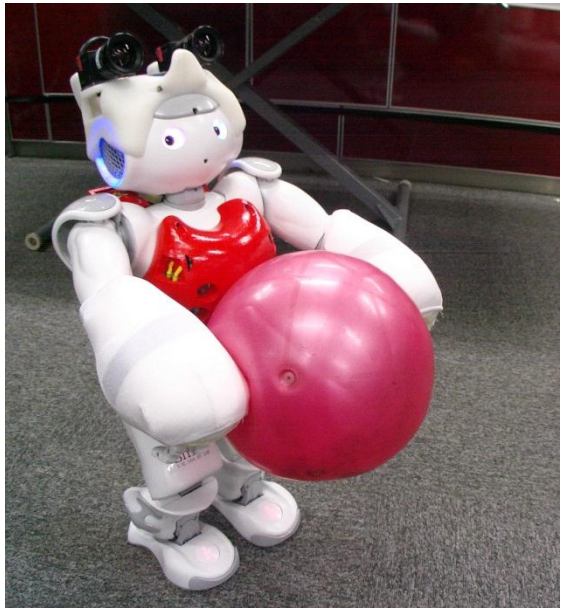
    RosOutPort<numbers::Num> _numOut;

  というようにRTM側から定義して使える
- ただし，現在公式ページで配布しているのはcturtle用

試したい人のために，後で，wikiにdiamondback用のpatchとやり方を上げます

RTM/ROS相互接続システムの例



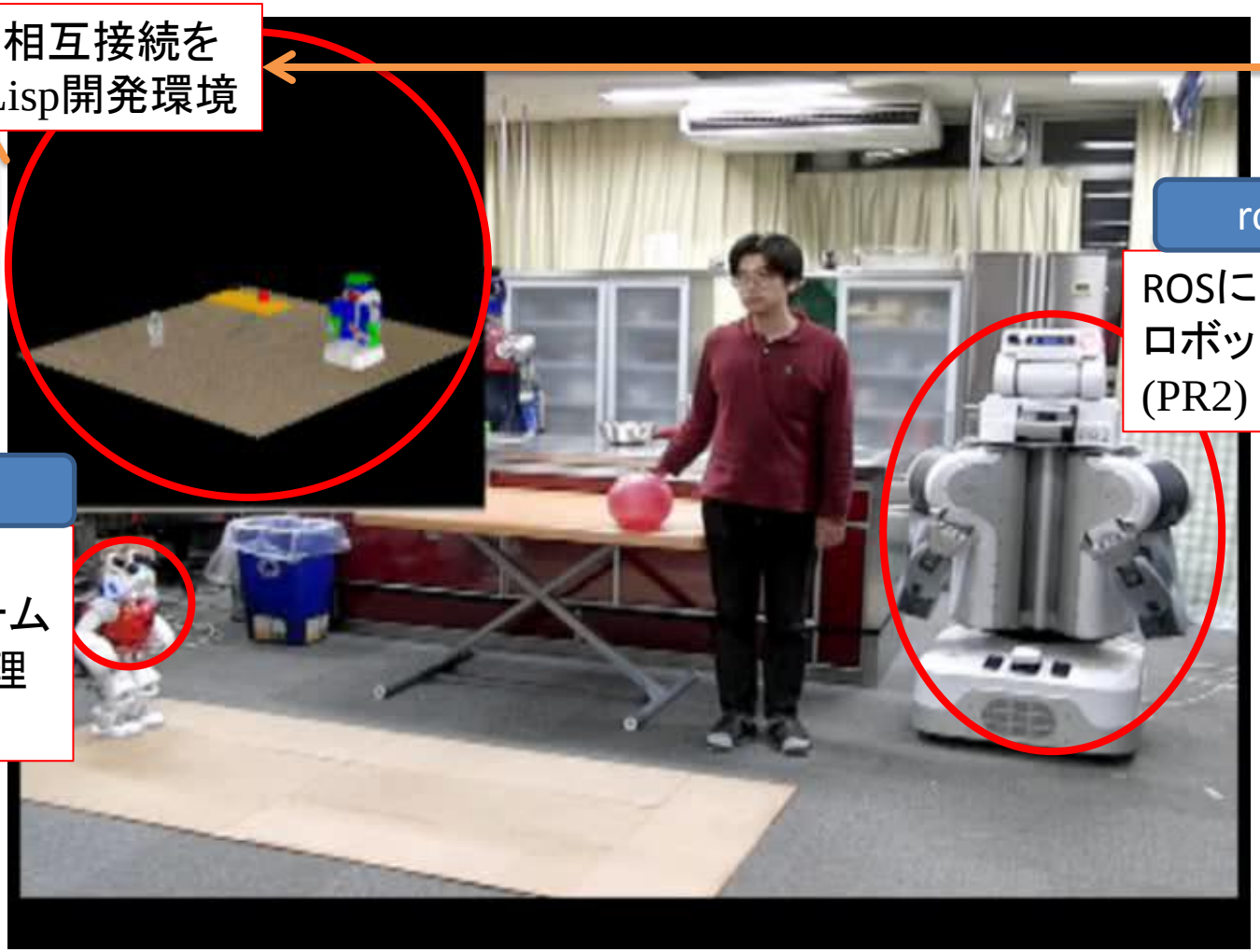


RTM対応画像処理
モジュール搭載NAO

PR2（ROSシステム）

RTM/ROS接続による2台ロボット協調作業アプリケーション

RTMとROSの相互接続を実現するEusLisp開発環境



roseus

ROSによるロボットシステム(PR2)

rtmeus

RTMによるロボットシステム(組込画像処理ボード)

rtmeus：
EusLisp によるRTMインタフェース

- OpenRTM-aist-1.0.0-Release対応
- rtcshellの利用によりEusLispからのRTMシステムインタフェース機能を実装
- 独自機能としては，
  - 汎用入出力インタフェースのためのサービスポート拡張
    （文字列型引数，返り値によるConsumer/Provider記述）
  - コンポーネント再利用性向上のためのデータポート拡張
    （配列構造のプロファイルをデータポートを通じてやり取りする）
  - EusLispからのコンポーネント制御
- http://code.google.com/p/rtmeus/

にて公開しています

汎用入出力インタフェースのための サービスポートの拡張

- 汎用入出力のためのConsumer / Provider
  - 引数：文字列型1（実行関数名文字列）
    文字列型2（実行関数への引数）
  - 返り値：文字列型x1（実行関数からの返り値）

- 引数の解釈，返り値の生成はバインドされた実行関数内で行う

コンポーネント再利用性向上のための データポートの拡張

- データ型，個数を起動時にコンフィギュレーションファイルから読み込むことで再利用性の高いRTCを実現

- 配列構造に関するデータをプロファイルとして，コンポーネント間でやりとりできるよう拡張

- 実際にコンポーネントの接続を行う時に接続の型を決めることが可能になる

rtmeus sample codes

- コンポーネント参照，接続，activateの例

```
(require :rtmeus
         (format nil "~A/eus/common/rtmeus.l" (unix:getenv 'RTMEUSDIR)))
;;
(setq *rtm* (instance RTMeus :init "localhost"))
(setq hoge (send *rtm* :resolve-component "UsbCameraAcquire0.rtc"))
(setq flip (send *rtm* :resolve-component "Flip0.rtc"))
(setq fuga (send *rtm* :resolve-component "UsbCameraMonitor0.rtc"))
(send *rtm* :connect-components hoge "imageout" flip "originalImage")
(send *rtm* :connect-components flip "flippedImage" fuga "ImageData")
(send *rtm* :all-activate)
```

今日のトピック

- 独自データ形式によるデータ通信
  - 基本型，拡張型，独自型の定義
  - マルチキャプチャ環境を想定したカメラ用IFのIDL
    Img.dllを利用したデータ通信
- RTMとROSの相互接続利用
  - オープンソースロボット用スクリプト言語EusLispを利用した実現(rtmeus/roseus)
- RTM事例紹介
  - 富士通ビジョンボード

RTコンポーネントの利用が可能な組込み画像処理モジュール

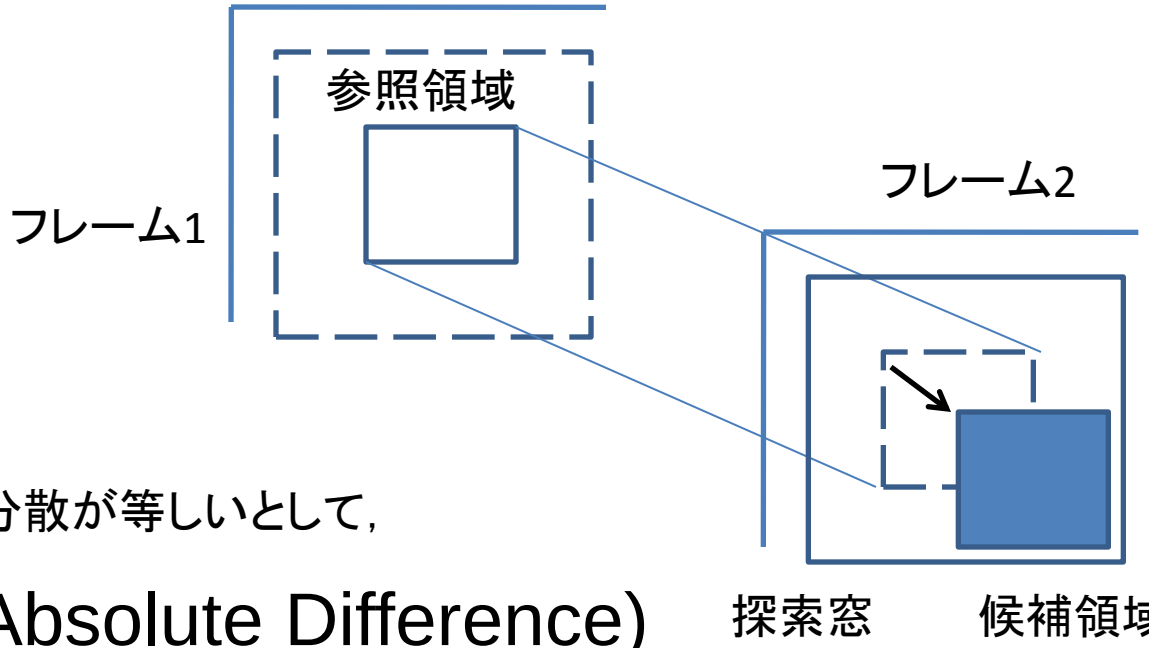
- 組込みOS搭載ハードウェア画像処理モジュール

次世代知能化プロジェクトの成果として富士通研究所株式会社が開発，2009年より富士通九州ネットワークテクノロジーズ株式会社から販売

- PowerPC440EPx（666MHz）＋ ImageProcessingLSI(200MHz)
- OS：組込Linux
- Ether IEEE 1394カメラ or Analog NTSCカメラ入力
- C，C++でアプリが書ける
- イーサネット経由でRTコンポーネントが走る
- 画像処理モジュール用RTMは株式会社セックより提供予定

局所相関演算をハードウェアチップ化

- 局所相関演算



各小領域の輝度値の平均と分散が等しいとして，

- SAD相関(Sum Absolute Difference)

$$Corr(u,v)=\sum_{x,y\in R}\left|I_{f1}(x,y)-I_{f2}(x+u,y+v)\right|$$

- SSD相関(Sum of Squared Difference)

$$Corr(u,v)=\sum_{x,y\in R}\left\{I_{f1}(x,y)-I_{f2}(x+u,y+v)\right\}^2$$

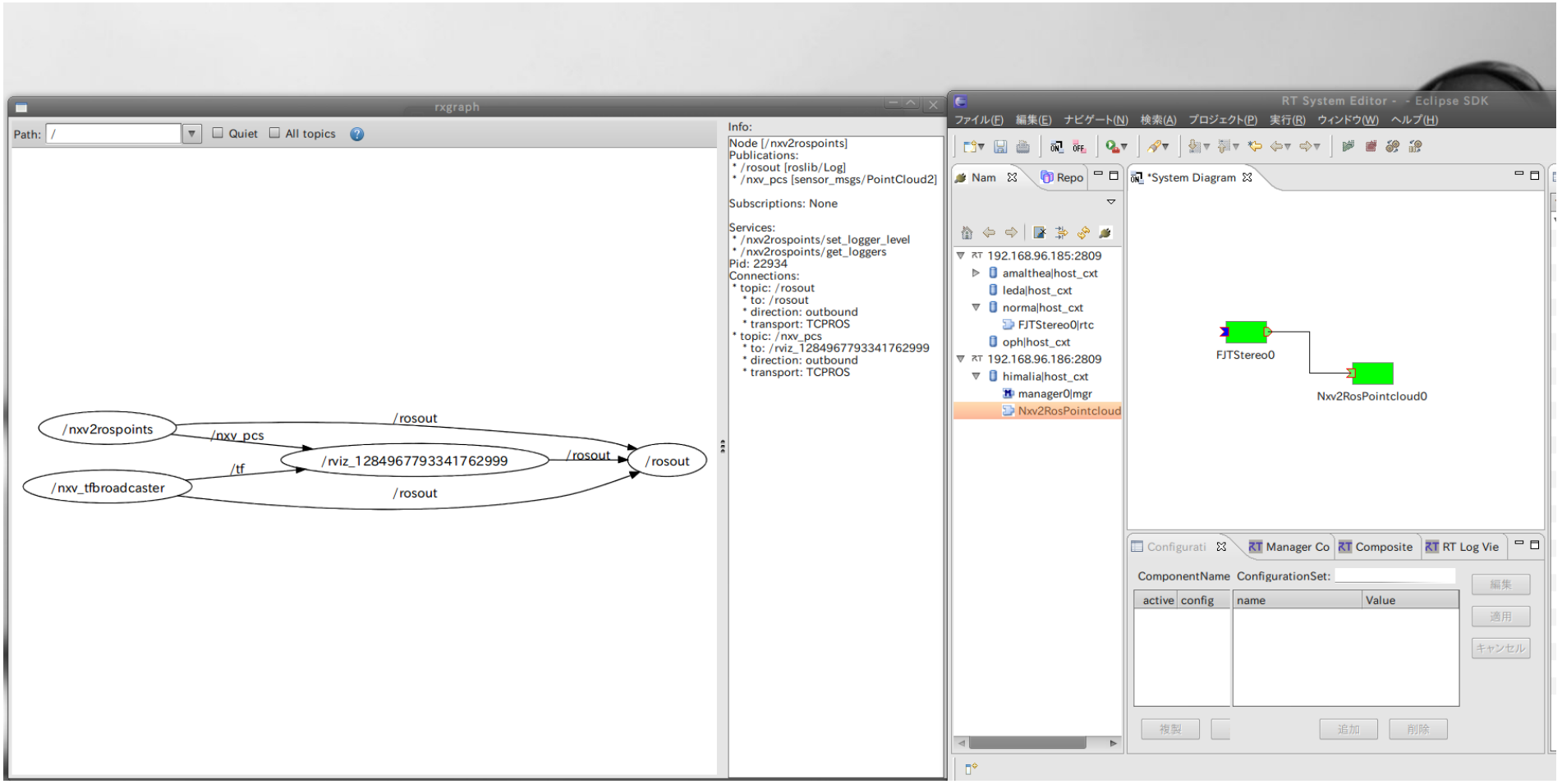
局所相関演算により可能な視覚処理

- 記憶との間の相関演算
  - 参照画像に対するテンプレートマッチング
- 左右両眼画像の間での相関演算
  - ステレオ立体視によるデプスマップの生成
- 時間方向の変化に対する相関演算
  - 画像内でのオプティカルフロー
- 左右両眼と時間方向の双方への相関演算
  - 3次元オプティカルフローの生成

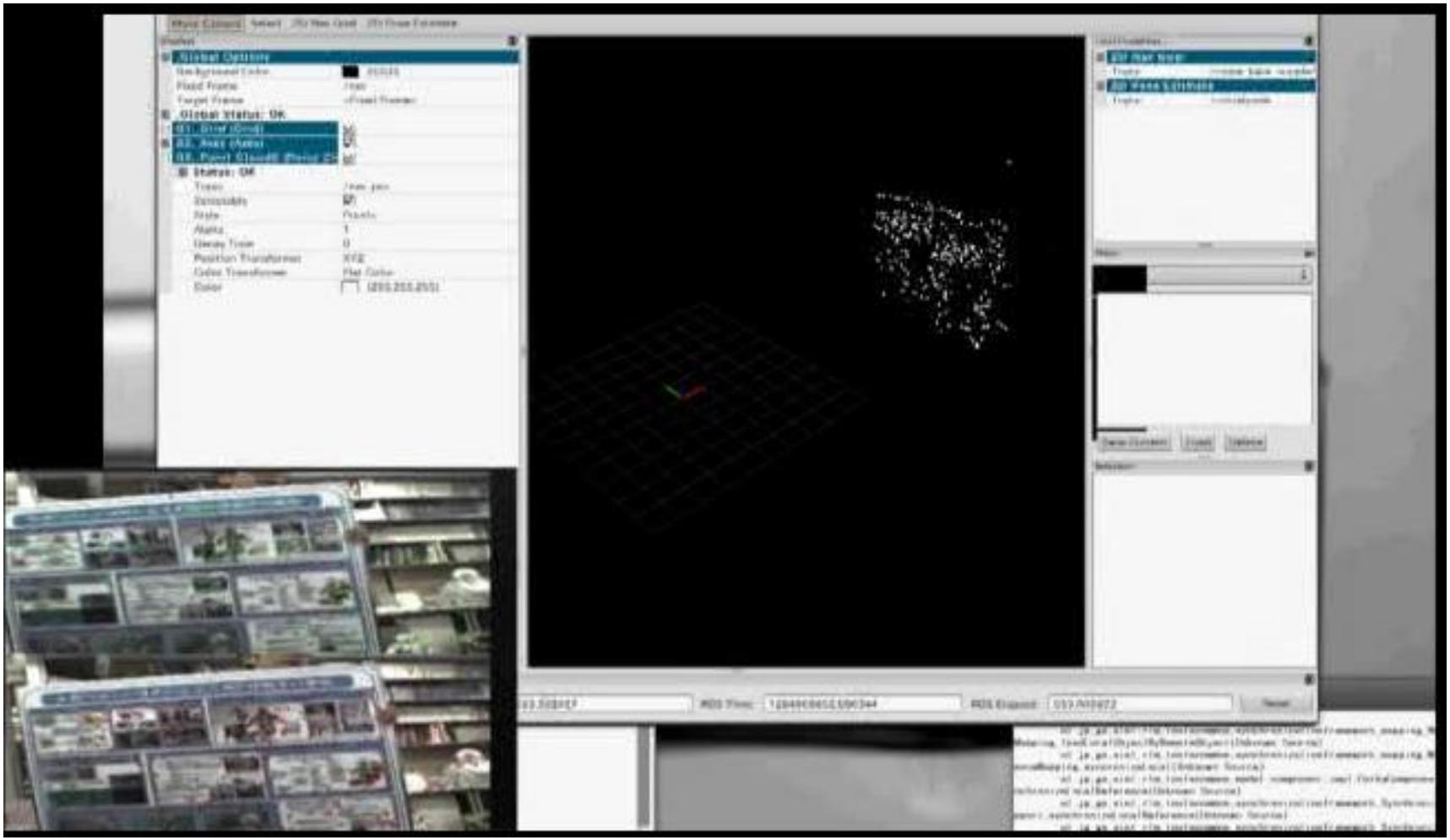
2次元Optical Flowのデモンストレーション

富士通ビジョンボードとROSとの連携

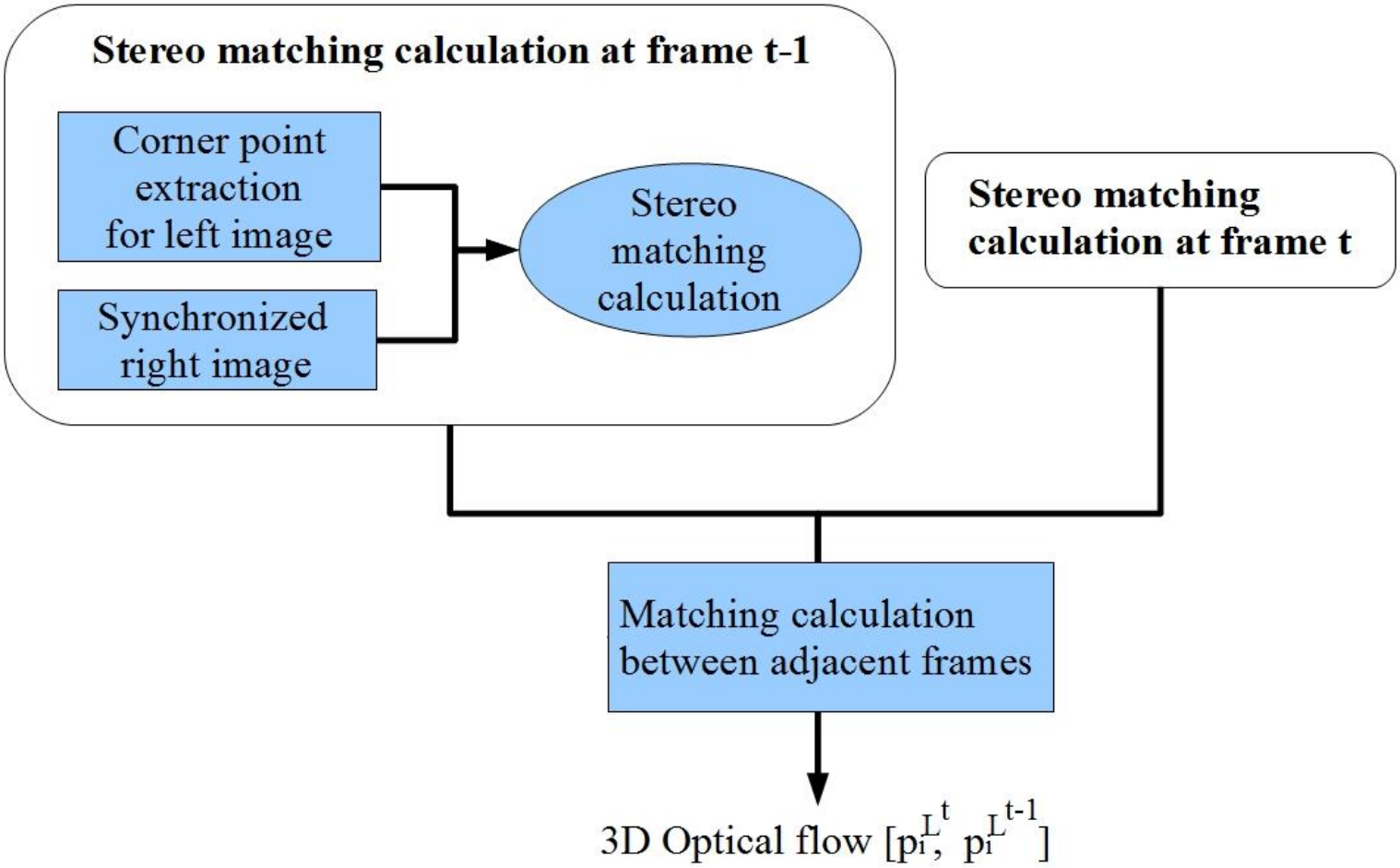
- ステレオ三次元点→ポイントクラウド変換



ステレオ三次元点からポイントクラウドへの変換

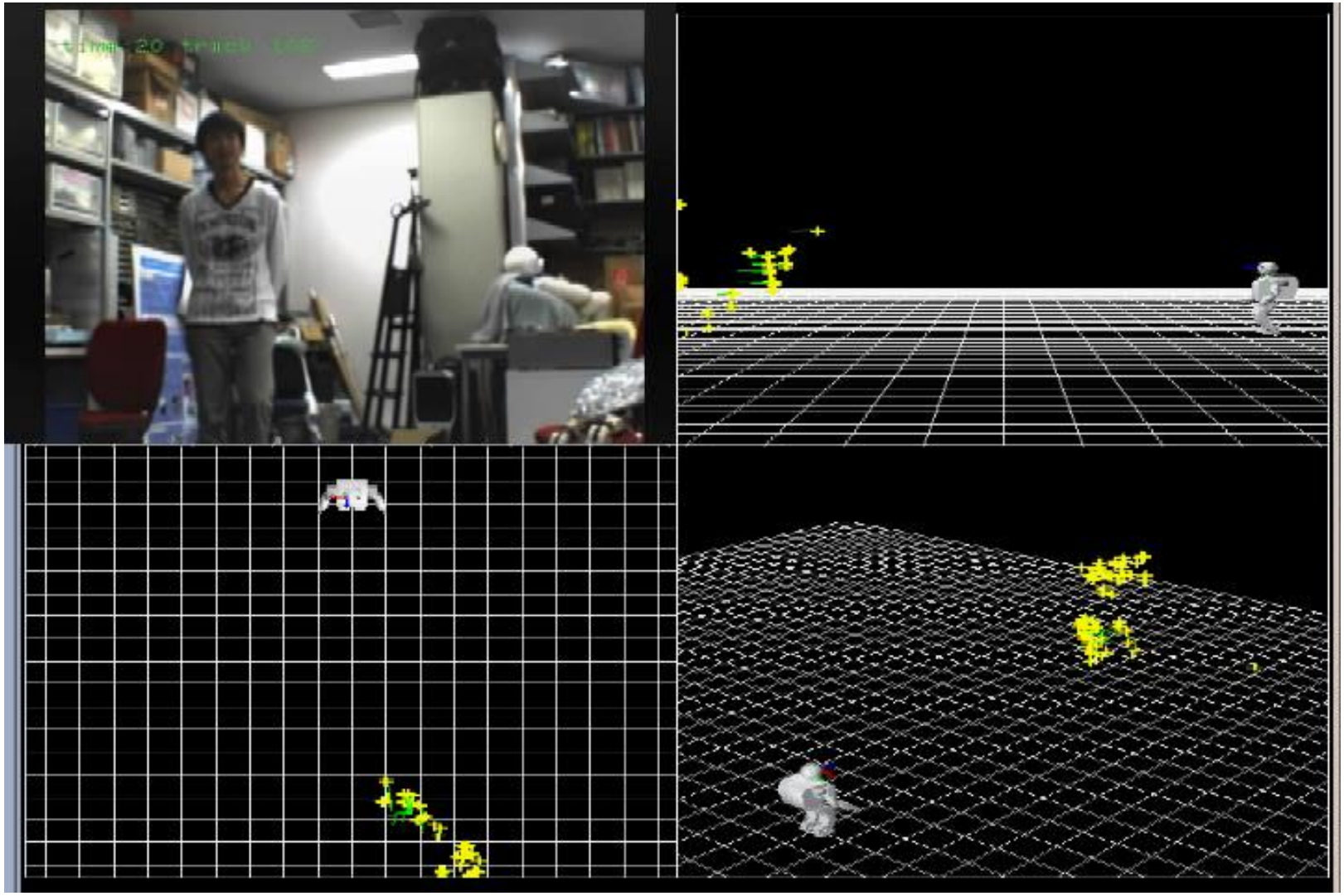


特徴点追跡型3次元フロー生成コンポーネント

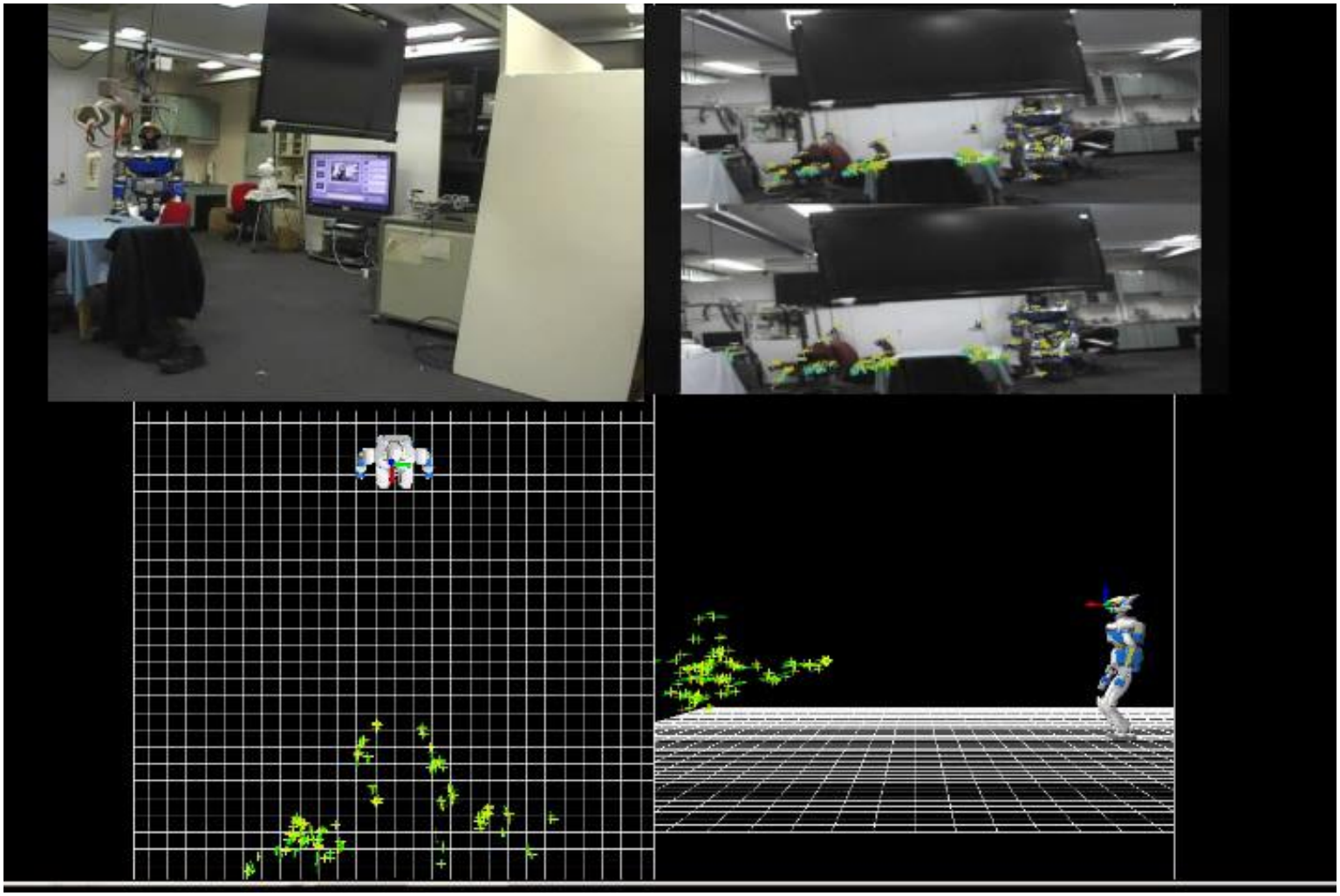


- コーナー特徴点によりステレオマッチングした点群の追跡により，3次元的なフローを生成
- 150点以上の特徴点群に対して，30[fps]での計算が十分可能

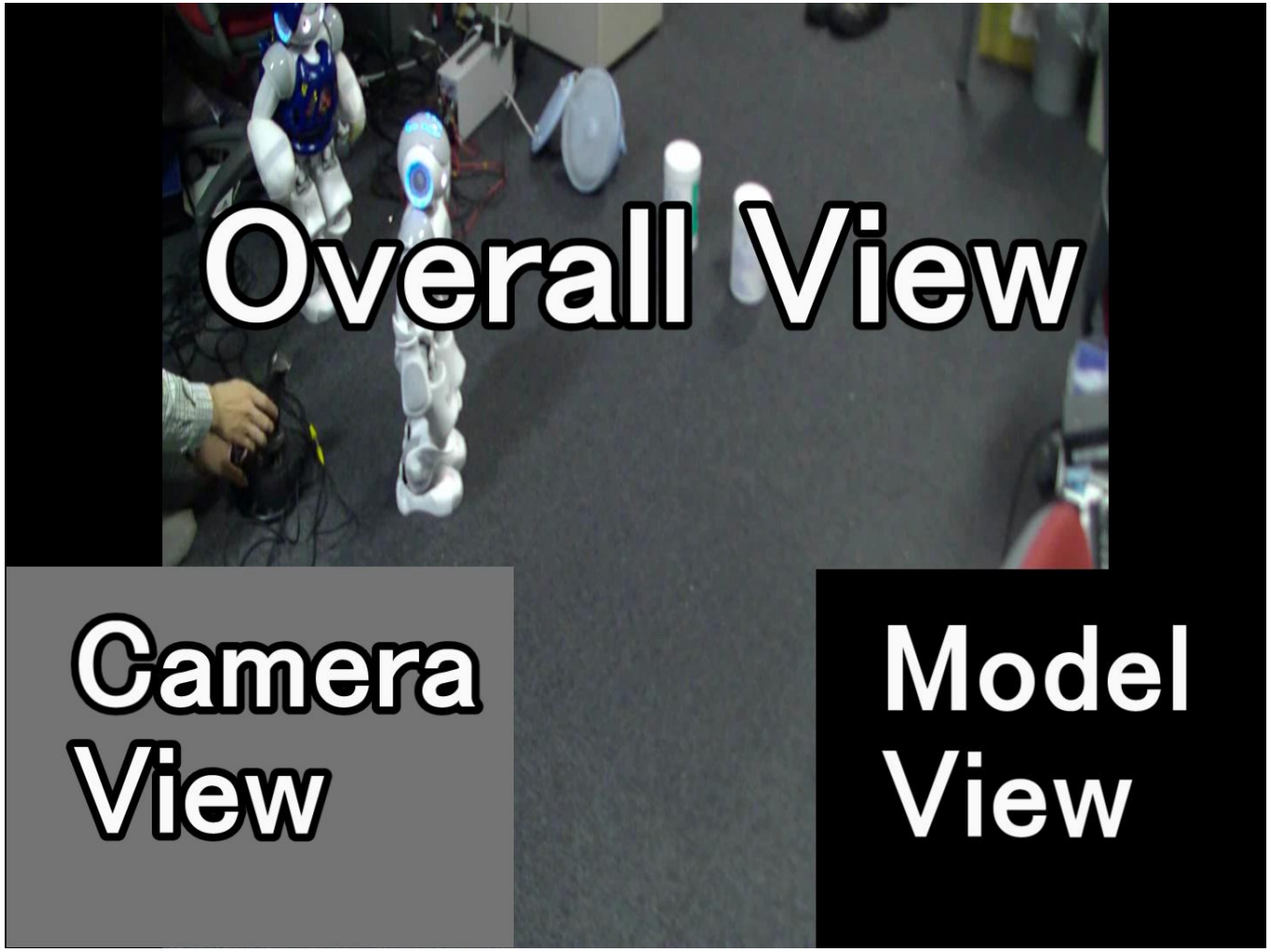
3次元フロー生成の様子（実演もします）



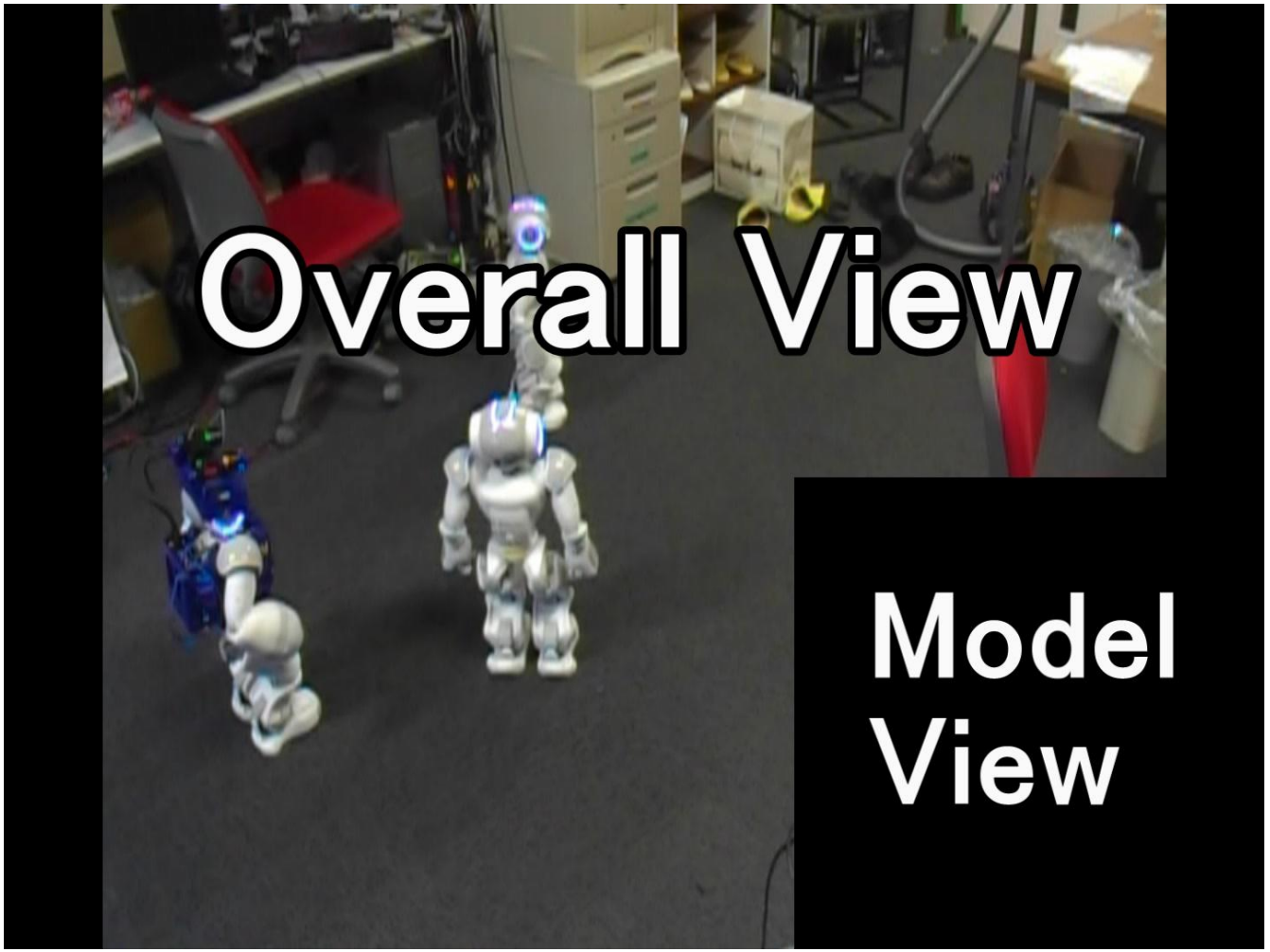
3次元フロー応用：自己運動の追跡



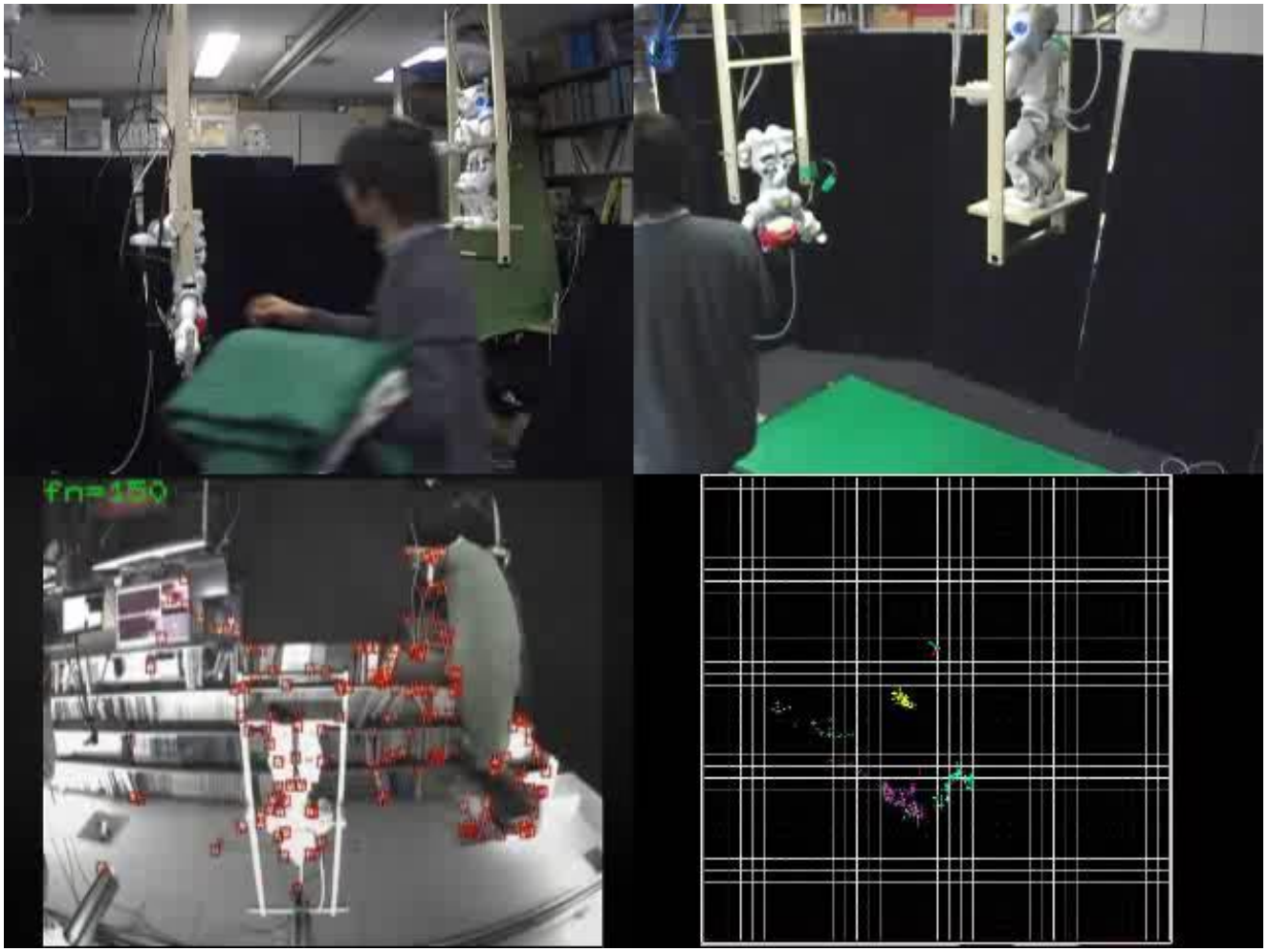
3次元フロー応用：自己運動と移動物体の分離認識による追従行動



３次元フロー応用：自己運動と複数移動物体の分離認識による追従行動



3次元フロー応用：自己運動と移動物体の分離認識による空中ブランコ



今回の宿題

- 基本課題（6点）
  - これまでに作った画像処理RTMをImg.idlを利用したものに改造し，サンプルのキャプチャ，モニタプログラムと接続せよ（2点）
  - 今回のキャプチャサンプルではカメラキャリブレーションデータの読込機能は実装していない．ROS側でrectifyした画像データをImg.idl形式に変換して，キャリブレーションされた画像データに対して上記のプログラムを接続するプログラムのソースと結果の動画を提出せよ（4点）
- 応用課題（4点）
  - これまで，ROS/OpenRAVEの回で動かしてきたサンプル，或いは個々が宿題として提出したものとの連携システムを
    - 6/18に解説したrtmros統合システム
    - rtm/rosの両方の口を持つノード/コンポーネントシステム
    - EusLispによるrtmeus/roseus統合システム（wikiにサンプルは載せます）

  のいずれかの方法で作成し，個人リポジトリで提出せよ．ただし，こちらの手元で試しづらいものは動画も一緒に提出すること．

いずれにせよ，面白いものができたら，是非メーリングリストに報告してください