

エージェントシステム第3講義

ソフトウェアエンジニアリング
最新ツールと手法

2011-05-11

出杏光 魯仙
であんこう ろせん

内容

- ソフトウェアエンジニアリングで重要な所
 - テスト
 - Continuous Integration方法
 - ドキュメント
 - リリース
- アナウンス
 - 個人レポジトリ作成と宿題提出
 - 宿題

アンケート

- 面白いプログラムを書いたことがありますか？
- 2年以上に使い続けたことがありますか？
- 友達と知り合いに使われていますか？
- 10人以上に使われていますか？
- 100人以上？
- 1000人以上？
- なぜ難しくなっているでしょう？

日本発で世界有名なオープンソース



Jenkins

A Jenkins community resource



テスト
サーバー

川口耕介



言語

松本行弘

ロボット・人工知能オープンソース

ROS.org

[About](#) | [Support](#) | [answers.ros.org](#)

Search:

Documentation

Browse Software

News

Download

Documentation

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.

ROS:

[Install](#)

Install ROS on your machine.

Wiki

[ROS](#)
[StackList](#)
[RecentChanges](#)
[openrave](#)
[Documentation](#)

ロボット
統合

Player

navigation

- Main page
- Manuals
- Project Page
- Recent changes
- Random page

Main Page

About the Player Project

The Player Project creates Free Software that enables research in robot and sensor systems. The Player robot server is probably the most widely used robot control interface in the world, and supports a wide variety of hardware. Client libraries in C, C++, Python, and Ruby are officially supported, while Java, Ada, Octave, and others are supported by third parties. Its simulation

Current Player version: **3.0.2**
(Changelog)
Current Stage version: **4.0.0**
(Changelog)
Current Gazebo version:
0.10.0 (Changelog)

ロボット
ドライバー

[Home](#)

News

Projects

2D-I-SLSJF
CAS-Toolbox
CEKF-SLAM
DP-SLAM
EKFMonoSLAM
FLIRTLib

OpenSLAM
Give your algorithm to the community

What is OpenSLAM.org?

The simultaneous localization and mapping (SLAM) problem has been intensively studied in the robotics community in the past. Different techniques have been proposed but only a few of them are available as implementations to the community.

2D・3D
地図作成

日本発のロボット・人工知能オープンソース

- ユーザーがほとんど日本国内
- OpenRTM通信ミドルウェア



OpenRTM-aist
The power to connect

ホーム ダウンロード ドキュメント コミュニティ 研究・開発

OpenRTM-aist クイックスタートページ
OpenRTM-aistを10分で始めよう!

ナビゲーション
ホーム
ダウンロード

ホーム
OpenRTM-aist official website
ここは(独)産業技術総合研究所 知能システム研究部門 OpenRTM-aistの公式Webサイトです。
現在の最新RELEASEバージョン: 1.0.0-RELEASE
現在の最新RELEASEバージョンは OpenRTM-aist-1.0.0-RELEASE です。 [こちら](#)からダウンロードできます。現在の各言語、ツールのバージョンは以下の通りです。
• C++: 1.0.0-RELEASE

Download
OpenRTM-aist-1.0.0

ダウンロード
最新リリース

C++	1.0.0-RELEASE
Java	1.0.0-RELEASE
Python	1.0.0-RELEASE
Tools	1.0.0-RELEASE

- HARK音声処理



Languages
English

Navigation
Top
Updates
Softwares
Documentation
Related Papers
FAQ
People
Support

Top Diff List Farm Source Search Help RSS Login

HARK

MainPage

News
HARK
HRI-JP Audition for Robots with Kyoto University (HARK)
Features
Download and Installation
Related Work and Project

News
Nov. 25th 2010, HARK 1.0.0 released

ソフトウェアの 成功と普及の原因は？

- 1。使用条件
- 2。信頼性
- 3。拡張可能

ソフトウェアエンジニアリング

プロセス

要素

要求分析

企画

実装

公開

メンテナンス

開発の進め方

バグ・フィーチャー
マネージメント

この講義

品質管理

役割分担

宣伝

誰でも
使用、信頼、拡張出来る
ソフトウェアの実現
が
現在の技術と
将来のソフトウェア
の柱になる

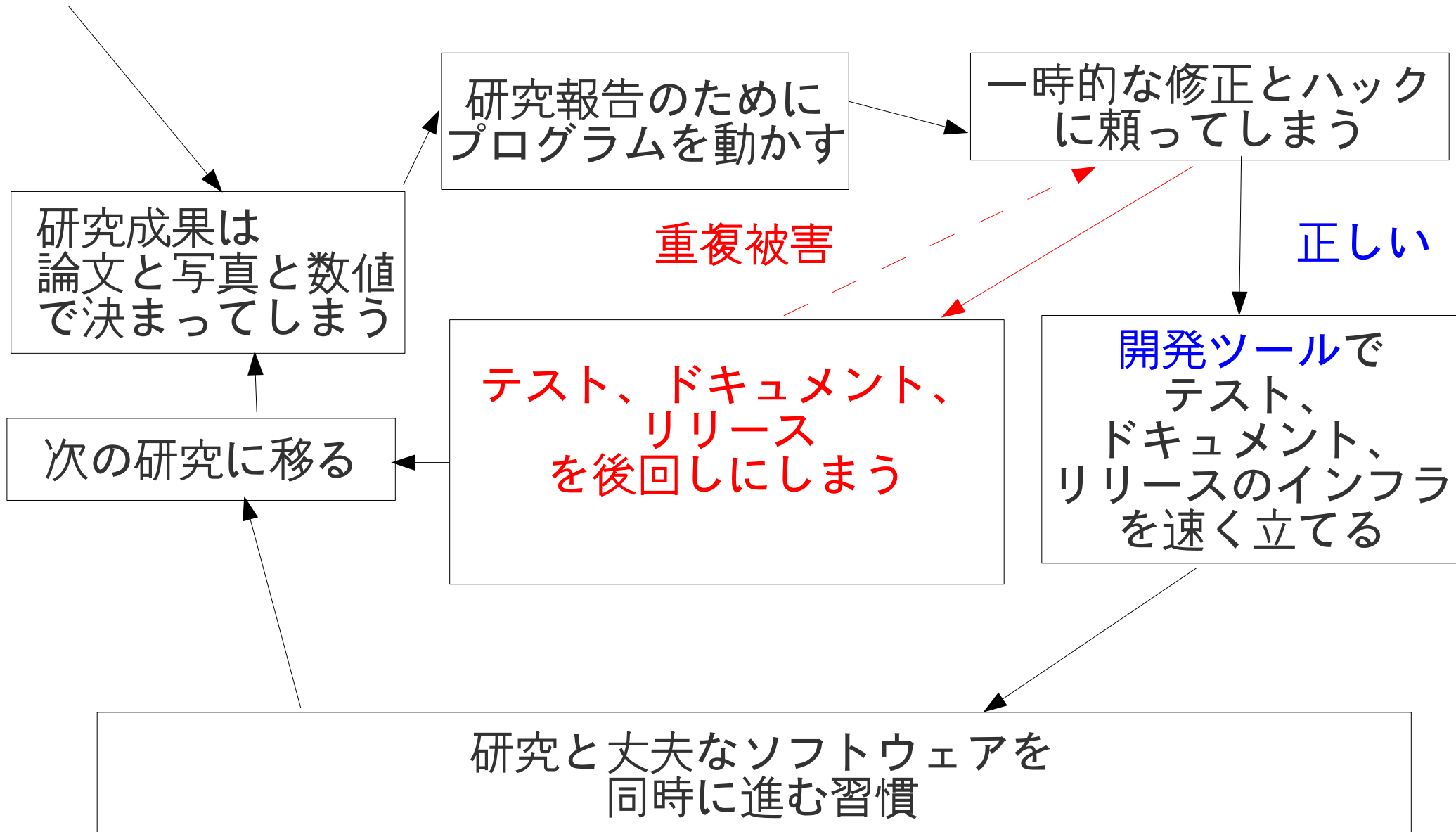
丈夫なソフトウェア目標

- 使用してもらえるために
 - **リリース**版を頻繁に出す
 - 変更された分を明確にする
 - **ドキュメント**：APIとチュートリアル
- 信頼してもらえるために
 - 書いてある機能が100%動く保証が出来る**テスト**
 - **ドキュメント**：テスト方法
- 拡張してもらえるために
 - API：アプリケーションプログラミングインタフェース
 - **ドキュメント**：開発者むけ・メモ

開発プロセス重要要因

- テスト
 - すべての機能が動いている保証
 - テスト結果がわかりやすい
- ドキュメント
 - HTML、PDF、WIKIの内容とコード・実装との同期
- リリース
 - 自動的なパッケージ作成
 - すぐ試せるデモ・実行例
- 総合注意点
 - メインテ・管理しやすい？

目的達成の困難さ



丈夫なソフトウェアは
更にいい研究・技術を
生み出せる

テストシステム

テストシステム概要

- 信頼性の価値が高いため、企業秘密が多い
 - Intelの効率的なテスト技術でCPUを速く市場に出せる
 - Microsoftでテストを行っているプログラマーが40%
- 種類
 - 単体テスト：機能ごと
 - 結合テスト：コンポーネントを組み合わせる
 - システムテスト
 - 他のプログラム、ハードウェア、通信ネット、データベースでテストする
- 目的
 - 回帰テスト (regression testing)
 - 受け入れテスト (acceptance testing)

一般テスト項目

- 全チュートリアルとプログラムが実行出来るか
- 全ターゲットOSで実行可能か
 - Linux: Ubuntu, Fedora Core, Debian, Redhat
 - Windows XP, Vista, 7
- 様々なコンパイラ
 - GCC 3.3, 4.0, 4.4
 - Visual Studio 20XX, MinGW, Intel Compiler
- 違うバージョンのライブラリ
 - Boost C++: 1.34, 1.40, 1.44
 - OpenCV: 2.0, 2.2
- 分散処理
 - ライブラリが一つのスレッドで走るにも関わらず、ユーザの環境がそうでもない

Jenkinsツールで
簡単に設定出来る

Pythonテストサンプル 1

- 逆行列関数をpythonのnumpyライブラリでテスト

```
def test_inv():  
    for i in range(5000):  
        N = random.randint(100)+1  
        T = random.rand(N,N)-0.5  
        Tinv = linalg.inv(T)  
        shouldbezero = dot(T,Tinv)-eye(N)  
        error = sum(abs(shouldbezero))  
        assert( error <= N*N*1e-12 )
```

- python-noseツールで起動する
 - nosetests test.py

```
=====  
FAIL: test.test_inv  
=====
```

Traceback (most recent call last):
 File "/home/rdiankov/python-nose-test/src/nose/nose/case.py", line 187, in runTest
 self.test(*self.arg)
 File "/home/rdiankov/research/test.py", line 10, in test_inv
 assert(error <= N*N*1e-12)
AssertionError

Pythonテストサンプル 1

- 行列のサイズでテストを分ける

```
def myinv(N):  
    for i in range(10000*N):  
        T = random.rand(N,N)-0.5  
        Tinverse = linalg.inv(T)  
        shouldbezero = dot(T, Tinverse)-eye(N)  
        error = sum(abs(shouldbezero))  
        assert( error <= N*N*1e-12 )  
  
def test_myinv():  
    for N in range(10):  
        yield myinv, N
```

- 結果（4行以上の行列がよく失敗する）：

```
-----  
FAIL: test_myinv(4,)
```

```
-----  
Traceback (most recent call last):
```

```
File "/home/rdiankov/python-nose-test/src/nose/nose/case.py", line 187, in runTest
```

```
    self.test(*self.arg)
```

```
File "/home/rdiankov/research/test.py", line 33, in myinv
```

```
    assert( error <= N*N*1e-12 )
```

```
AssertionError
```

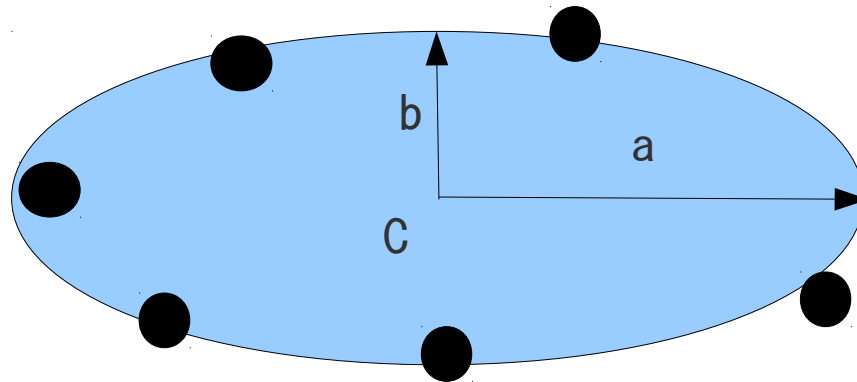
```
Ran 10 tests in 21.485s
```

テストサンプル 1 分散処理

- Python-noseで並列を行うために入力する
 - `_multiprocess_can_split_ = True`
- マルチコアで実行する
 - `nosetests test.py --processes=6`
 - 6コアと9.1秒
 - 1コアは21秒

テストサンプル 2

- 楕円を点群から抽出する関数
 - `cx, cy, rotation, a, b, error <= fitEllipse(points)`



- テスト流れ
 - `cx, cy, rotation, a, b, error`をランダムに選んで`points`を作成する
 - `fitEllipse(points)`の出力と入力を比較する
 - 楕円でない点群も作成する
 - 全部の`points`が一定値のケースもテスト（例外）

ロボットナビゲーションのテスト

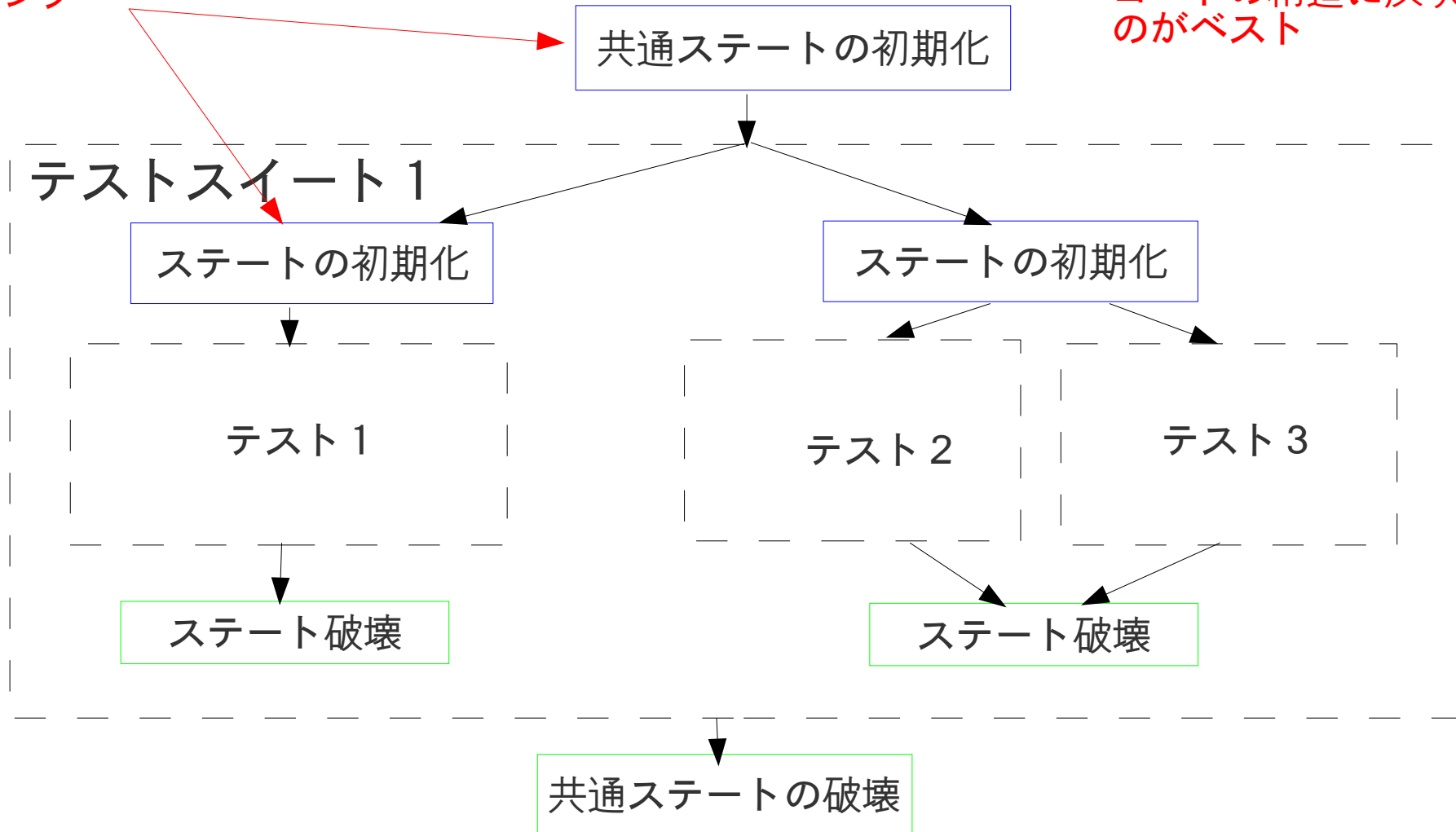
- Willow Garage
- 2日間で自律に
オフィスを回る



テストの構造

テスト
ジグ

コードの構造に反映する
のがベスト



C++サンプル: GoogleTest

<http://code.google.com/p/googletest/>

- `fitting.h`
`/// \brief 楕円を近似する`
`double fitEllipse(const cv::Mat& points, cv::RotatedRect& ellipse);`

`/// \brief 楕円を画像から抽出する`
`double fitEllipseFromImage(cv::Mat& image, cv::RotatedRect& ellipse);`

• テスト

```
#include <gtest/gtest.h>
#include "fitting.h"

TEST(Fitting, Ellipse)
{
    cv::Mat points;
    cv::RotatedRect input, output;
    double noise = 0;
    double pixelerror = 0.5;
    // pointsのサンプリング
    double fiterror = fitEllipse(points, output);
    ASSERT_NEAR(input.center.x, output.center.x, pixelerror);
    // ...
    EXPECT_NEAR(input.angle, output.angle, pixelerror);
    ASSERT_TRUE(fiterror <= noise*1.5);
}
```

GoogleTestでROSの画像処理テスト

- ジグでROSの初期化

```
class TestImageResults : public testing::Test
{
protected:
    boost::shared_ptr<ros::NodeHandle> nh_;
    ros::Subscriber posesub_;
    geometry_msgs::Pose2D realpose_;
    bool receivedpose_;

    virtual void SetUp() {
        receivedpose_ = false;
        nh_.reset(new ros::NodeHandle());
        posesub_ = nh_->subscribe("ellipse", 1, &TestImageResults::posecb, this);
    }
    virtual void TearDown() {
        posesub_.shutdown();
        nh_.reset();
    }
    void posecb(const geometry_msgs::Pose2DConstPtr msg) {
        ASSERT_NEAR(msg->x, realpose_.x, 0.5);
        ASSERT_NEAR(msg->y, realpose_.y, 0.5);
        EXPECT_NEAR(msg->theta, realpose_.theta, 0.2);
        receivedpose_ = true;
    }
};
```


GoogleTestでROSの画像処理テスト

- ジグのメンバー関数としてテストを定義する

```
TEST_F(TestImageResults, SendImage) {  
    ros::NodeHandle nh;  
    image_transport::ImageTransport imgtrans(nh);  
    image_transport::Publisher image_pub = imgtrans.advertise("image", 1);  
    realpose_.x = 100;  
    realpose_.y = 200;  
    realpose_.theta = 0;  
    receivedpose_ = false;  
    cv_bridge::CvImagePtr cv_ptr;  
    // realpose_で楕円を表示した画像を作成する (cv::Ellipse)  
    image_pub.publish(cv_ptr->toImageMsg()); // 画像を送信し  
    // 結果を待つ  
    for(int i = 0; i < 500; ++i) {  
        if( receivedpose_ ) {  
            break;  
        }  
        usleep(10000); // 10マイクロ秒で待つ  
    }  
    ASSERT_TRUE(receivedpose_);  
}
```

Pythonのテストライブラリ

- unittest : <http://docs.python.org/library/unittest.html>
 - 標準、使いにくい
- doctest : <http://docs.python.org/library/doctest.html>
 - ドキュメントにもなれる、柔軟性が低い
- py.test : <http://pylib.org/>
 - 柔軟性が高い
- python-nose : <http://code.google.com/p/python-nose>
 - 柔軟性が高い、プラグイン型、分散処理

python-noseでROS画像処理のテスト

```
import roslib; roslib.load_manifest('opencv_fitting')
import rospy
import nose
from geometry_msgs.msg import Pose2D
from image_msgs.msg import Image
import cv
from cv_bridge import CvBridge

class TestImageResults(object):
    def setup(self):
        self.realpose = None
        self.received = False
        rospy.init_node('testfitting', anonymous=True)
        self.subellipse = rospy.Subscriber("ellipse", Pose2D, self.posecb)
        self.pub = rospy.Publisher('image', Image)
        self.bridge = CvBridge()

    def teardown(self):
        self.subellipse.unregister()

    def posecb(self, msg):
        assert( abs(msg.x - self.received.x) <= 0.5)
        assert( abs(msg.y - self.received.y) <= 0.5)
        assert( abs(msg.theta - self.received.theta) <= 0.2)
        self.received = True

    def test_sendimage(self):
        self.realpose = Pose2D(100, 100, 0.5)
        I = cv.CreateImage([256, 256], cv.IPL_DEPTH_8U, 1)
        cv.Ellipse(I, center=(self.realpose.x, self.realpose.y), axes=(100, 50), angle=
self.realpose.theta, start_angle=0, end_angle=6.28, color=(0, 0, 0))
        self.pub.publish(bridge.cv_to_imgmsg(I))
        # wait for results
```

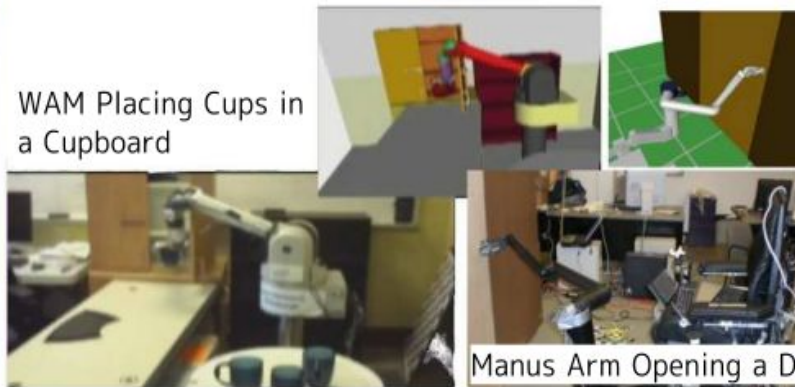
python-noseの特徴

- コアの機能
 - 複雑なテスト構造
 - テストの収集
 - ランタイムでテストの生成
- プラグインの機能
 - Coverageが簡単に図れる
 - 全テストが個別のプロセスで走れる
 - テスト結果をXMLに出力する（Jenkinsのため）
 - エラー発生の際に実際の変数の値も出力
 - 時間の制限
- C++のコードもテスト可能！
 - Boost Pythonでbindingを作れば

OpenRAVEで動作計画の技術

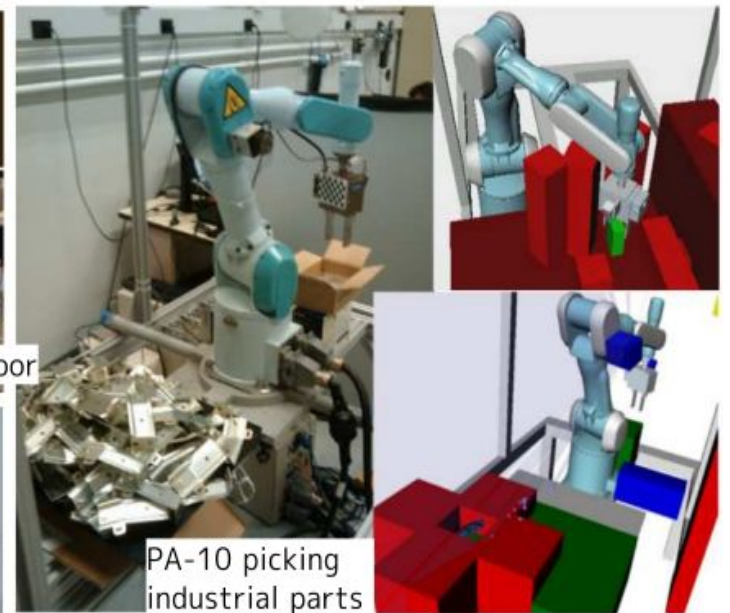


WAM Loading a Dish Rack



WAM Placing Cups in a Cupboard

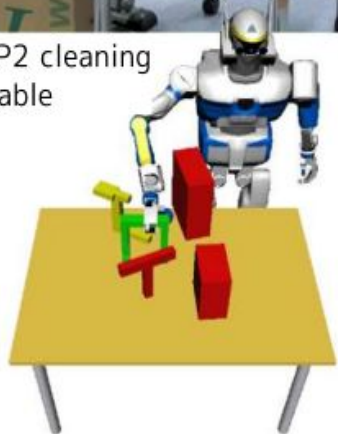
Manus Arm Opening a Door



PA-10 picking industrial parts

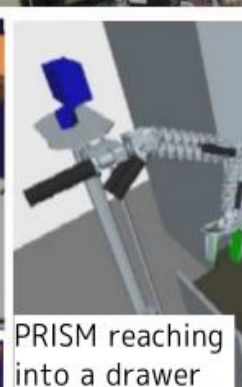


HRP2 cleaning a table

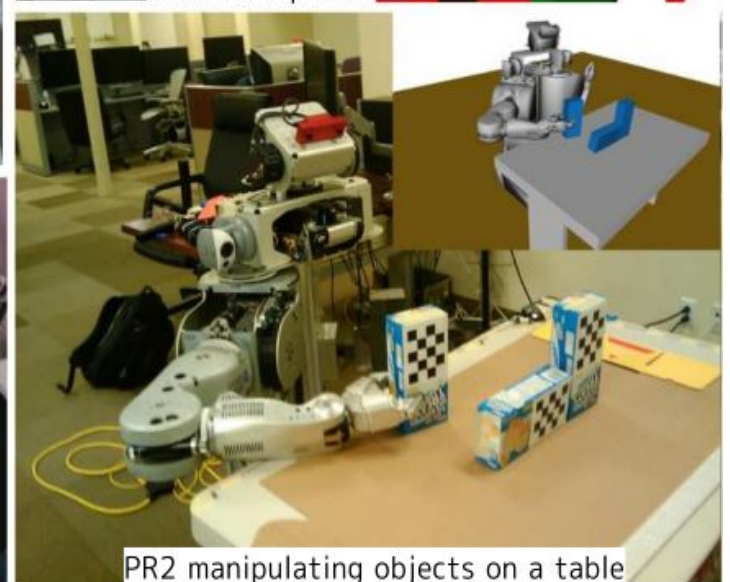


WAM Opening a Cupboard

WAM Opening a Refrigerator



PRISM reaching into a drawer



PR2 manipulating objects on a table

OpenRAVEのテスト仕組み

- 基本数学のテスト
 - 整合性がとれているか
 - $f(\text{finv}(X)) == X$, $f(X+Y) = f(X) + f(Y)$
- 運動学のテスト
 - 様々なロボット構造で試す
 - リンクの位置姿勢と関節の値
- 逆運動学のテスト
 - 書くロボットと書くIKの種類
- プログラム・デモの実行テスト
 - ユーザが最初に試すプログラムが無事に終わっているか

テストを書く時に

- 独立性
 - エラー発生の際に、原因に速く絞れる仕組み
- 再現性
 - 失敗がまれな時でもいつも失敗させる
- 再利用性
 - 10個の環境があれば一つのテストを10回実行する
- OSに依存しないように
- 分散処理の環境で試す

Continuous Integration

品質管理の連続的なプロセス

Jenkins

- チェックアウト、ビルド、実行、テストのスクリーンリポートをジョブで管理する
- PCクラスターの登録、パラレルに走らせる
- エラー発生時にメール送信
- テスト結果を履歴に保存する

The screenshot shows the Jenkins web interface. At the top, there's a blue header with the Jenkins logo and a search bar. Below the header, there's a sidebar on the left with links like 'New Job', 'Manage Jenkins', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'My Views', and 'Dependency Graph'. The main area displays a table of jobs with columns for status (S), weather icon (W), job name, last success, last failure, and last duration. The jobs listed are 'openrave', 'openrave_all', 'openrave_documentation', 'openrave_linux', 'openrave_publish_latest', and 'openrave_windows'. At the bottom, there's a 'Build Queue' section showing 'No builds in the queue.' and a 'Build Executor Status' section showing two executors: 'Master (offline) dulse' and 'wakame'.

S	W	Job ↓	Last Success	Last Failure	Last Duration
		openrave	12 days (#84)	12 days (#83)	2 hr 20 min
		openrave_all	9 days 17 hr (#30)	N/A	0.57 sec
		openrave_documentation	9 days 17 hr (#15)	25 days (#11)	6 min 49 sec
		openrave_linux	1 day 14 hr (#38)	1 day 15 hr (#37)	14 min
		openrave_publish_latest	9 days 17 hr (#28)	1 mo 3 days (#9)	3 min 39 sec
		openrave_windows	9 days 17 hr (#68)	9 days 19 hr (#66)	8 min 53 sec

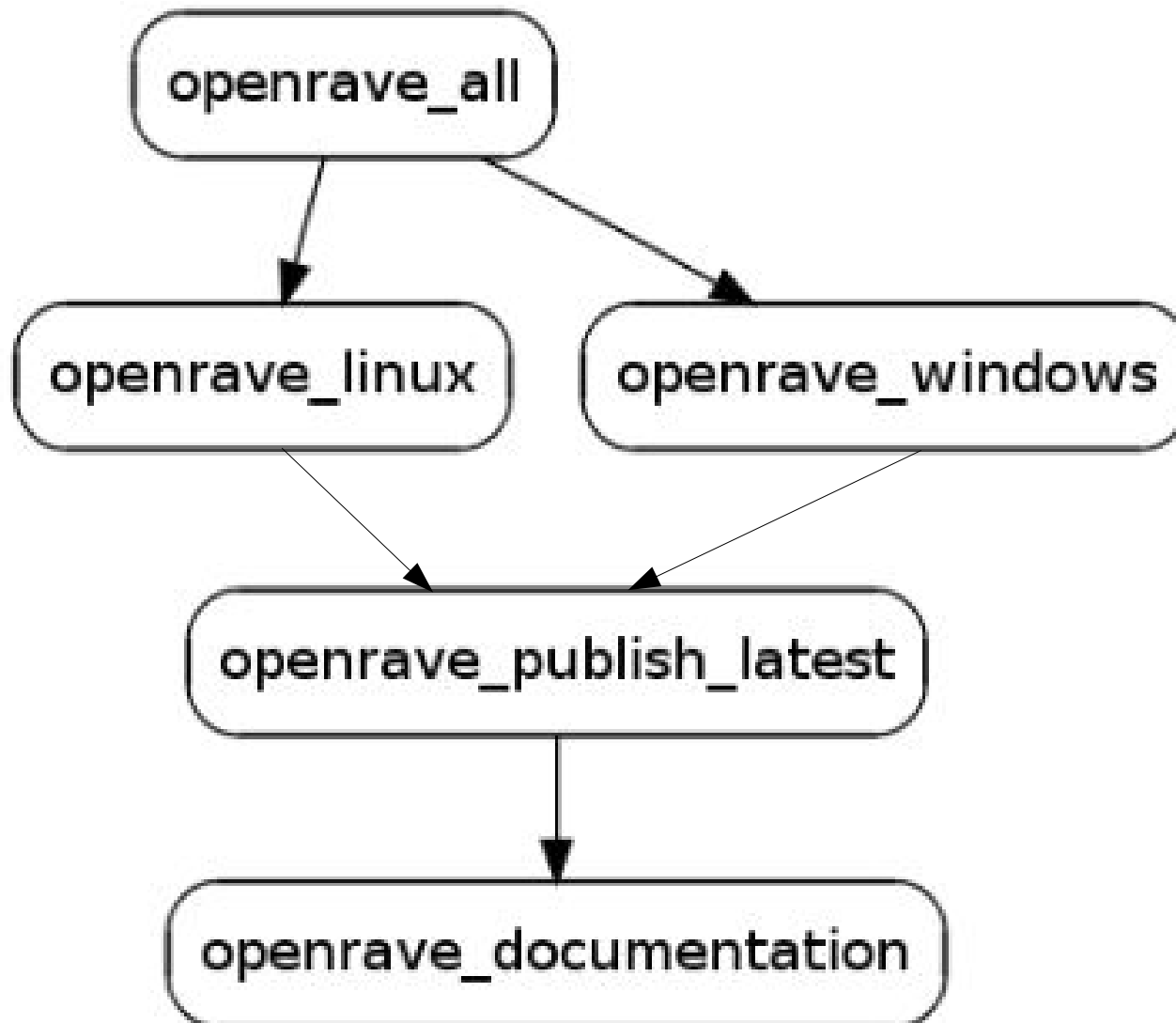
Icon: [S](#) [M](#) [L](#)

Legend: for all for failures for just latest builds

Jenkinsのジョブ

- ネットからコードのゲット（定期的に）
- コードのビルド：ビルド環境の設定
 - chroot, LD_LIBRARY_PATH, PATH
- 特定されたテストを走らせる
- テスト結果によって下流でジョブを起動出来る
- 下流のジョブが無事に終われば上流に報告する
 - テスト実行の依存関係
- ファイルをSSHで転送出来る

OpenRAVEのJenkinsジョブ



openrave_allジョブ

Project name

Description

☐ Discard Old Builds ?

☐ This build is parameterized ?

☐ Use Subversion Release Manager ?

Trac website ?

☒ Promote builds when... ?

Promotion process

Name

Icon

Criteria

☒ When the following downstream projects build successfully ?

Job names

☐ Trigger even if the build is unstable

☐ Only when manually approved ?

☐ When the following upstream promotions are promoted ?

☐ If the build is a release build ?

☐ If the build is a release build ?

Actions

☒ Build other projects ?

Projects to build

Jenkinsの設定

- ブラウザーから全設定が出来る
- 中身はXMLとして保存されている
- Pythonからの直接設定出来ます
 - <http://www.ros.org/wiki/hudson>
 - ジョブの登録、ジョブの状態、PCの登録、

```
import roslib; roslib.load_manifest("hudson")
import hudson
hudson.create_job(jobname, config_xml)
```

openrave_11ジョブXML上

```
<project>
  <actions/>
  <description>triggers all openrave tests and if successful builds the &quot;rele
  <keepDependencies>>false</keepDependencies>
  <properties>
    <hudson.plugins.trac.TracProjectProperty>
      <tracWebsite>https://sourceforge.net/apps/trac/openrave/</tracWebsite>
    </hudson.plugins.trac.TracProjectProperty>
    <hudson.plugins.promoted_builds.JobPropertyImpl>
      <activeProcessNames>
        <string>build_promotion</string>
      </activeProcessNames>
    </hudson.plugins.promoted_builds.JobPropertyImpl>
  </properties>
  <scm class="hudson.scm.NullSCM"/>
  <canRoam>true</canRoam>
  <disabled>>false</disabled>
  <blockBuildWhenDownstreamBuilding>>false</blockBuildWhenDownstreamBuilding>
  <blockBuildWhenUpstreamBuilding>>false</blockBuildWhenUpstreamBuilding>
  <triggers class="vector"/>
  <concurrentBuild>>false</concurrentBuild>
  <builders>
    <hudson.tasks.Shell>
      <command># have to create the publish directory since will be copying result
        ssh rdiarkov@dulse &quot;mkdir -p /var/lib/jenkins/workspace/openrave_publ
        ve_publish_latest/*.exe&quot;;
      </command>
    </hudson.tasks.Shell>
  </builders>
```

openrave_aliジョブXML下

```
<publishers>
  <hudson.tasks.BuildTrigger>
    <childProjects>openrave_linux, openrave_windows</childProjects>
    <threshold>
      <name>SUCCESS</name>
      <ordinal>0</ordinal>
      <color>BLUE</color>
    </threshold>
  </hudson.tasks.BuildTrigger>
  <hudson.tasks.test.AggregatedTestResultPublisher>
    <includeFailedBuilds>true</includeFailedBuilds>
  </hudson.tasks.test.AggregatedTestResultPublisher>
</publishers>
<buildWrappers/>
</project>
```

- openrave_linux, openrave_windowsを実行する

テストシステムの流れ

- テストはそれぞれの言語で書く
 - 細かい分散処理もそのテストツールで書く
- Jenkinsで
 - コードの定期的なチェックアウト
 - 全テストの実行、結果の収集
 - それぞれのOSパッケージ作成
 - テスト結果でドキュメント生成
 - 正式なリリースをアップロードする
 - パッケージもドキュメントも

テストシステム立ち上げのお勧め

- Jenkinsのサーバーをインストール
- Virtual Machineで全ターゲットOSを起動
- Python-noseかgoogletestで全テストを書く

例 : ROS <http://build.willowgarage.com>
OpenRAVE <http://openrave.org/testing>

ドキュメントシステム

ドキュメントシステム概要

- 目的
 - 「このソフトに何が入っているか、どうやって使えばよいか」を自動的に答えられるシステム
- 問題
 - コードの頻繁更新・変化
 - 変化に伴ってドキュメントの書く場所を理解する

ドキュメント方法：論文

- 利点
 - 概念・ニーズ・比較・結果が伝わる
 - 新規性が伝わりやすい
- 欠点
 - 時代遅れになってしまう
 - 詳細を書かないことになっている
 - コードの使い方、便利ツールの関係
 - 少数人で書く
 - 論文の結果を再現出来る方法を書かない

例：Quigley M, Gerkey B, Conley K, Faust J, Foote T, Leibs J, Berger E, Wheeler R, Ng A: An open-source robot operating system (ROS). In: ICRA Workshop on Open Source Software in Robotics, 2009.

ドキュメント方法：Wiki

- 利点

- 誰にでも更新出来るのでコミュニティが現れる
- 情報が整理しやすい
 - コンテンツ・ページの作成、検索ツール
- 世界のニーズとともに変化出来る

- 欠点

- コードとの同期が失われる
- Wikiが大きくなって古い情報を削除のが難しい
- 全データを保存しにくい
 - WikiはDBで管理されているからログインが必要

例：COLLADA <https://collada.org/mediawiki>
ROS <http://ros.org>

ドキュメント方法：コード挿入

- コードの中にコメントとして挿入し、ツールでHTML、PDF、LATEX、XML等を生成する
 - ツール：Sphinx、Doxygen
- 利点
 - 管理しやすい
 - コードとの同期がとれている
 - コードの変更時に側のドキュメントも更新する
 - ドキュメントされていないものに警告が出る
- ツールによって出力の柔軟性が変わる
 - Sphinxの柔軟性がWikiレベル以上

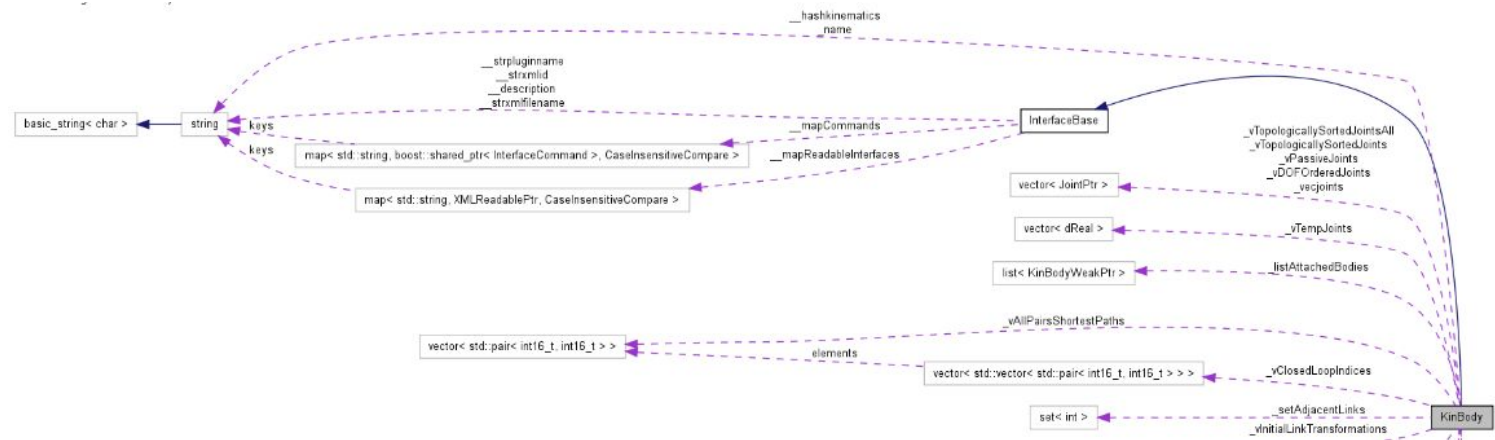
Doxygen : 言語・ツール

- C++ドキュメント作成・管理
- 継承と依存関係図
- ソースコードからのリンク作成



Getting Started

- **Building and Installing**
- **Basic Usage and Loading Environments**
- Environment Variables



Doxygen関数例

```
/** \brief Sets the velocity of the base link and each of the joints.  
  
Computes internally what the corresponding velocities of each of the links should be in order to  
achieve consistent results with the joint velocities. Sends the velocities to the physics engine.  
Velocities correspond to the link's coordinate system origin.  
\param[in] linearvel linear velocity of base link  
\param[in] angularvel angular velocity rotation_axis*theta_dot  
\param[in] vDOFVelocities - velocities of each of the degrees of freedom  
\param checklimits if true, will explicitly check the joint velocity limits before setting the values.  
*/  
virtual bool SetDOFVelocities(const std::vector<dReal>& vDOFVelocities, const Vector& linearvel,  
                             const Vector& angularvel, bool checklimits = false);
```



```
bool SetDOFVelocities ( const std::vector< dReal > & vDOFVelocities,  
                        const Vector & linearvel,  
                        const Vector & angularvel,  
                        bool checklimits = false  
                      )  
[virtual]
```

Sets the velocity of the base link and each of the joints.

Computes internally what the corresponding velocities of each of the links should be in order to achieve consistent results with the joint velocities. Sends the velocities to the physics engine. Velocities correspond to the link's coordinate system origin.

Parameters:

[in] linearvel linear velocity of base link

[in] angularvel angular velocity rotation_axis*theta_dot

[in] vDOFVelocities - velocities of each of the degrees of freedom checklimits if true, will explicitly check the joint velocity limits before setting the values.

Definition at line **2194** of file **KinBody.cpp**.

Doxygenの多言語サポート

```
/** \en \brief Computes the minimal chain of joints that are between two links in the order of linkindex1 to linkindex2

Passive joints are also used in the computation of the chain and can be returned.
Note that a passive joint has a joint index and dof index of -1.
\param[in] linkindex1 the link index to start the search
\param[in] linkindex2 the link index where the search ends
\param[out] vjoints the joints to fill that describe the chain
\return true if the two links are connected (vjoints will be filled), false if the links are separate

\ja \brief 2つのリンクを繋ぐ関節の最短経路を計算する。

受動的な関節は、位置関係が固定されているリンクを見つけるために調べられている
受動的な関節も返される可能性があるから、注意する必要があります。
\param[in] linkindex1 始点リンクインデックス
\param[in] linkindex2 終点リンクインデックス
\param[out] vjoints 関節の経路
\return 経路が存在している場合、trueを返す。
*/
virtual bool GetChain(int linkindex1, int linkindex2, std::vector<JointPtr>& vjoints) const;
```

```
bool GetChain ( int          linkindex1,
                int          linkindex2,
                std::vector< JointPtr > & vjoints
                )          const [virtual]
```

2つのリンクを繋ぐ関節の最短経路を計算する。

受動的な関節は、位置関係が固定されているリンクを見つけるために調べられている。受動的な関節も返される可能性があるから、注意する必要があります。

引数:

[in] linkindex1 始点リンクインデックス
[in] linkindex2 終点リンクインデックス
[out] vjoints 関節の経路

戻り値:

経路が存在している場合、trueを返す。

KinBody.cpp の 2774 行で定義されています。

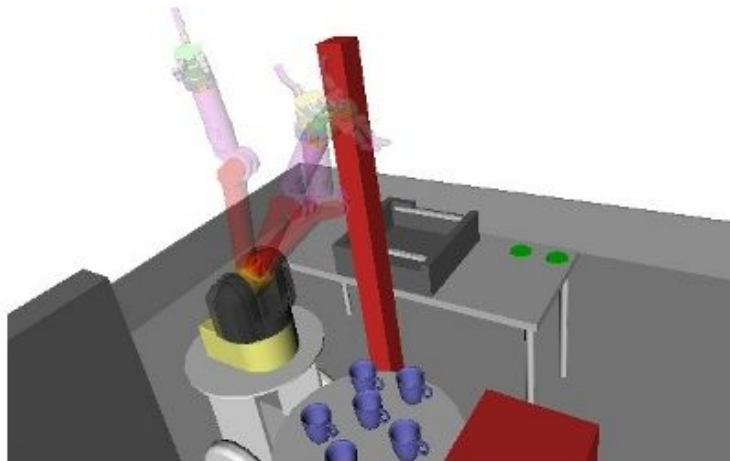
Doxygen実行例

Main Page	Related Pages	Modules	Namespaces	Classes	Files	Examples
						<input type="text" value="Search"/>

ortrajectory.cpp

Author:

Rosen Diankov



Robot moving in random configurations.

Shows how to send a cubically interpolated trajectory to the robot controller. The actual trajectory consists of two points: the current configuration and the target configuration.

```
TrajectoryBasePtr traj = env->CreateTrajectory(probot->GetDOF());
probot->GetDOFValues(q); // get current values
traj->AddPoint(TrajectoryBase::TPOINT(q,probot->GetTransform(),0.0f));
q[RaveRandomInt() %probot->GetDOF()] += RaveRandomFloat()-0.5; // move a random axis
traj->AddPoint(TrajectoryBase::TPOINT(q,probot->GetTransform(),2.0f));
traj->CalcTrajTiming(probot,TrajectoryBase::CUBIC,false,false); // initialize the trajectory structures
```

The demo also adds a collision check at the target point to make sure robot is going to a collision free configuration.

```
{
    RobotBase::RobotStateSaver saver(probot); // add a state saver so robot is not moved permanently
    probot->SetDOFValues(q);
    if( env->CheckCollision(RobotBaseConstPtr(probot)) ) {
        continue; // robot in collision at final point, so reject
    }
}
```

Python Bindingへの移転

- Breathe: <http://michaeljones.github.com/breathe/>

- DoxygenのドキュメントをXMLでPythonに移転出来る

- Pythonで入力すれば

```
import openravepy
help openravepy.KinBody.SetDOFVelocities
```

- PythonのヘルプとしてDoxygenが出ます

```
Help on method SetDOFVelocities:
```

```
SetDOFVelocities(...) unbound openravepy.openravepy.int.KinBody method
SetDOFVelocities( (KinBody) arg1, (object) dofvelocities) -> bool :
```

```
bool **SetDOFVelocities**\ (const std::vector< dReal > & vDOFVelocities, bool checklimits = false )
```

```
Sets the velocity of the joints.
```

```
*Parameters*
```

```
``vDOFVelocity`` -
```

```
- velocities of each of the degrees of freedom checklimits if true, will explicitly check the joint velocity limits before setting the values. Copies the current velocity of the base link and calls SetDOFVelocities(linearvel, angularvel, vDOFVelocities)
```

```
SetDOFVelocities( (KinBody) arg1, (object) dofvelocities, (object) linear, (object) angular) -> bool :
```

```
bool **SetDOFVelocities**\ (const std::vector< dReal > & vDOFVelocities, const Vector & linearvel, const Vector & angularvel, bool checklimits = false )
```

reStructuredText : 言語

Sphinx : コンパイラー役

- Pythonの標準ドキュメント言語
 - <http://docutils.sourceforge.net/rst.html>
- Doxygenと一緒にコードに挿入出来ます
- Doxygenより柔軟性がはるかに高い
- Sphinxで非常に綺麗なドキュメントが作成可能

Sphinx関数例

```
@docstring.dedent_interpd
```

```
def plot(self, *args, **kwargs):
```

```
    """
```

```
    Plot lines and/or markers to the
    :class:`~matplotlib.axes.Axes`. *args* is a variable length
    argument, allowing for multiple *x*, *y* pairs with an
    optional format string. For example, each of the following is
    legal::
```

```
        plot(x, y)           # plot x and y using default line style and color
        plot(x, y, 'bo')     # plot x and y using blue circle markers
        plot(y)              # plot y using x as index array 0..N-1
        plot(y, 'r+')         # ditto, but with red plusses
```

```
    If *x* and/or *y* is 2-dimensional, then the corresponding columns
    will be plotted.
```

```
    An arbitrary number of *x*, *y*, *fmt* groups can be
    specified, as in::
```

```
        a.plot(x1, y1, 'g^', x2, y2, 'g-')
```

```
    Return value is a list of lines that were added.
```

```
    The following format string characters are accepted to control
    the line style or marker:
```

```
=====
character      description
=====
'-'            solid line style
'--'          dashed line style
```

```
matplotlib.pyplot.plot(*args, **kwargs)
```

Plot lines and/or markers to the [Axes](#). *args* is a variable length argument, allowing for multiple *x*, *y* pairs with an optional format string. For example, each of the following is legal::

```
plot(x, y)           # plot x and y using default line style and color
plot(x, y, 'bo')     # plot x and y using blue circle markers
plot(y)              # plot y using x as index array 0..N-1
plot(y, 'r+')         # ditto, but with red plusses
```

If *x* and/or *y* is 2-dimensional, then the corresponding columns will be plotted.

An arbitrary number of *x*, *y*, *fmt* groups can be specified, as in::


```
a.plot(x1, y1, 'g^', x2, y2, 'g-')
```

Return value is a list of lines that were added.

The following format string characters are accepted to control the line style or marker:

character	description
'-'	solid line style
'--'	dashed line style

実行例からギャラリーの自動作成



home | [examples](#) | [robots](#) | [plugins](#) | [database generators](#) | [wiki](#) | [contents](#) » [modules](#) | [index](#)

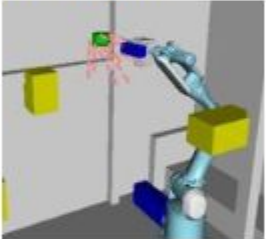
Examples

[Getting Started Tutorials](#)

[C++ Examples Page](#)


Python Examples

calibrationviews




Calibrates a camera attached on a robot by moving it around a pattern.

checkconvexdecomposition



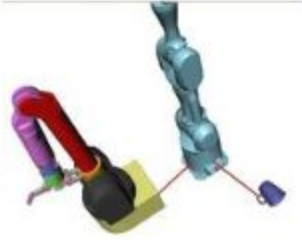
Builds the convex decomposition of the robot and plots all points inside its volume.

checkvisibility



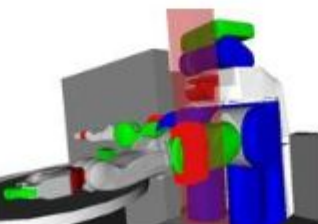
Computes the visibility extents of a camera and an object.

collision



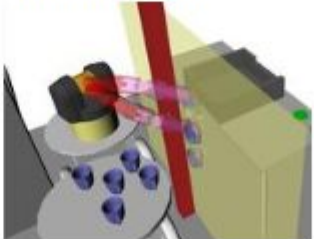
Check collision calls, use collision reports, and do distance queries.

collision2



Plot collision contacts.

constraintplanning



Shows how to use simple gradient-based jacobians to constrain the motion of the robot while planning.

Questions? Suggestions?

Join the [openrave-users](#) mailing list

Digest:

[Trac: Report bugs/request features](#)

Reference

[Core C++ API](#)

[Python API](#)

[Developers Guide](#)

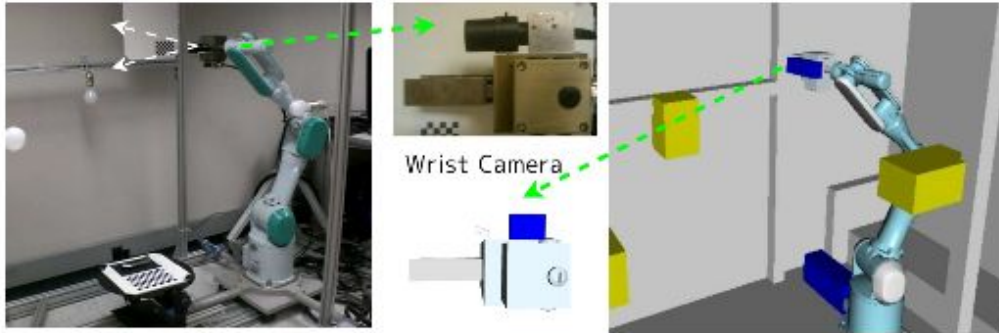
実行例ページ

[home](#) | [examples](#) | [robots](#) | [plugins](#) | [database generators](#) | [wiki](#) | [contents](#) » [openravepy Package](#) »

[examples Package](#) »

calibrationviews Module

Calibrates a camera attached on a robot by moving it around a pattern.



Running the Example:

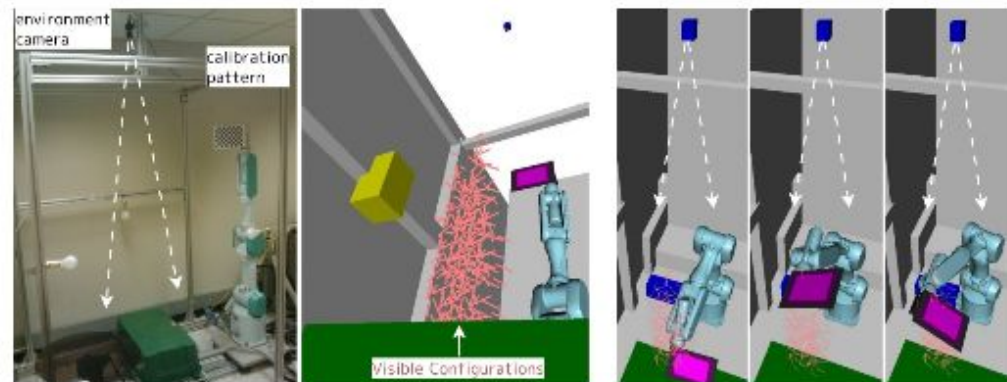
```
openrave.py --example calibrationviews
```

Description

The pattern is attached to the robot gripper and robot uses moves it to gather data. Uses [visibilitymodel](#) to determine which robot configurations make the pattern fully visible inside the camera view.

It is also possible to calibrate an environment camera with this example using:

```
openrave.py --example calibrationviews --scene=data/pal0calib_envcamera.env.xml --sensorrobot=ceiling
```



Command-line

Usage: openrave.py [options]

Views a calibration pattern from multiple locations.

Options:

-h, --help show this help message and exit
-scene=SCENE Scene file to load (default=data/pal0calib.env.xml)
-sensorname=SENSORNAME Name of the sensor whose views to generate (default is first sensor on robot)
-sensorrobot=SENSORROBOT Name of the robot the sensor is attached to (default=None)
-norandomize If set, will not randomize the bodies and robot position in the scene.
-novisibility If set, will not perform any visibility searching.
-posedist=POSEDIST An average distance between gathered poses. The smaller the value, the more poses robot will gather close to each other

OpenRAVE Environment Options:

-loadplugin=_LOADPLUGINS List all plugins and the interfaces they provide.
-collision=_COLLISION Default collision checker to use
-physics=_PHYSICS physics engine to use (default=None)
-viewer=_VIEWER viewer to use (default=qtcoin)
-server=_SERVER server to use (default=None).
-serverport=_SERVERPORT port to load server on (default=4765).
-l _LEVEL, -level=_LEVEL Debug level, one of (fatal,error,warn,info,debug,verbose)

Main Code

```
def main(env, options):  
    "Main example code."  
    env.Load(options.scene)  
    robot = env.GetRobots()[0]  
    sensorrobot = None if options.sensorrobot is None else env.GetRobot(options.sensorrobot)  
    env.UpdatePublishedBodies()  
    time.sleep(0.1) # give time for environment to update  
    self = CalibrationViews(robot, sensorname=options.sensorname, sensorrobot=sensorrobot, randomize=options.randomize)  
  
    attachedsensor = self.vmodel.attachedsensor  
    if attachedsensor.GetSensor() is not None and attachedsensor.GetSensor().Supports(Sensor.Type.Camera):  
        attachedsensor.GetSensor().Configure(Sensor.ConfigureCommand.PowerOn)  
        attachedsensor.GetSensor().Configure(Sensor.ConfigureCommand.RenderDataOn)  
  
    while True:  
        print "computing all locations, might take more than a minute..."  
        self.computeAndMoveToObservations(usevisibility=options.usevisibility, posedist=options.posedist)
```

Class Definitions

```
class openravepy.examples.calibrationviews.CalibrationViews(robot, sensorname=None,  
    sensorrobot=None, target=None, maxvelmult=None, randomize=False)[source]
```

```
computeAndMoveToObservations(waitcond=None, maxobservations=inf,  
    posedist=0.050000000000000003, usevisibility=True, **kwargs)[source]
```

Computes several configuration for the robot to move. If usevisibility is True, will use the visibility model of the pattern to gather data. Otherwise, given that the pattern is currently detected in the camera, move the robot around the local neighborhood. This does not rely on the visibility information of the pattern and does not create a pattern

```
computeLocalPoses(maxconvexdecomposition=0.5, maxposedist=0.10000000000000000)
```

www.openrave.org/en/main/openravepy/examples.checkconvexdecomposition.html

実行例のreStructuredText

Calibrates a camera attached on a robot by moving it around a pattern.

```
.. examplepre-block:: calibrationviews
```

Description

The pattern is attached to the robot gripper and robot uses moves it to gather data.
Uses :mod:`.visibilitymodel` to determine which robot configurations make the pattern fully visible inside the camera view.

It is also possible to calibrate an environment camera with this exapmle using:

```
.. code-block:: bash
```

```
openrave.py --example calibrationviews --scene=data/pal0calib_envcamera.env.xml --  
sensorrobot=ceilingcamera
```

```
.. image:: ../../images/examples/calibrationviews_envcamera.jpg  
   :width: 640
```

Calibration

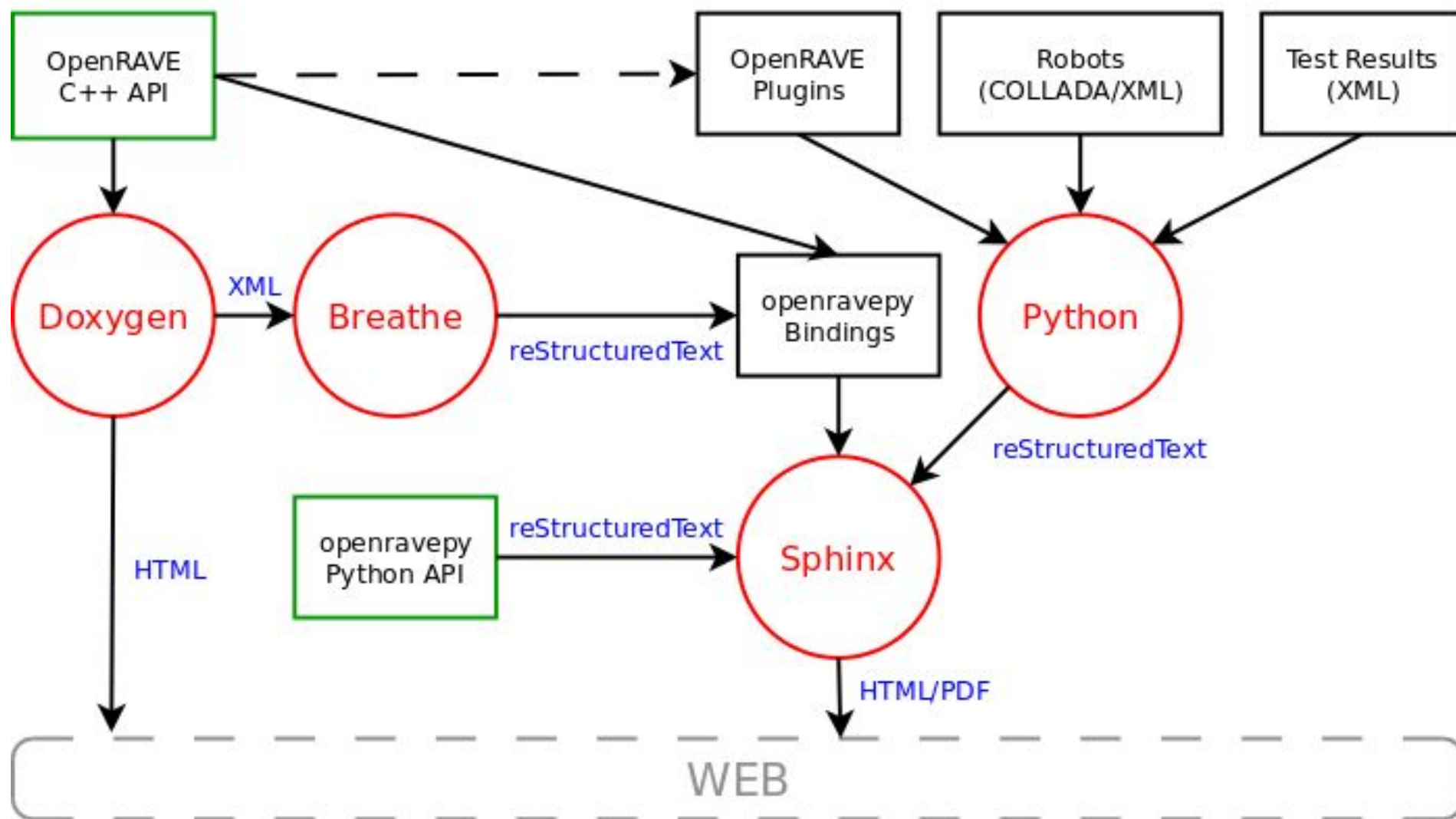
Although this example does not contain calibration code, the frames of reference are the following:

```
.. image:: ../../images/examples/calibrationviews_frames.jpg  
   :width: 640
```

****T_pattern^world**** and ****T_camera^link**** are unknown, while ****T_pattern^camera**** and ****T_link^world**** are known.

```
.. examplepost-block:: calibrationviews
```


OpenRAVE ドキュメント構造



ドキュメントシステム立ち上げのお勧め

- reStructuredTextで本も論文も書く
- コード挿入法で
 - コアのAPIとチュートリアルは
 - 本とシステム論文を書く
- Wiki法で
 - 他のシステムとの連携
 - 第三者のプロジェクト事例
 - ニュース

例 : OpenRAVE <http://openrave.org>

リリースシステム

リリースシステム概要

- ユーザにソースからビルドすると問題が起こる
- リリースパッケージの理想
 - 依存関係の解決
 - コンポーネント化
 - ユーザによって必要なものが異なる
 - 一つのコマンドで作成とアップロードを行う
 - ターゲットOSに対応する
 - ビルドシステムでインストールの設定を行える
- Ubuntu, Windows, RPMパッケージの作成
 - CMake

CMake/CPackでパッケージ作成

- CPackを使用し、OS対応の作成が出来る
 - `cpack -G NSIS`
 - `cpack -G DEB`
 - `cpack -G TGZ`
- CMakeLists.txt
 - `set(CPACK_XXX)`で設定を行う
 - コンポーネント化
 - `install(FILES myfile DESTINATION bin COMPONENT base)`
 - 依存関係
 - `set(CPACK_DEBIAN_PACKAGE_DEPENDS libboost-dev)`

Ubuntuパッケージ：Debian式

- Launchpadビルドファームの利用
 - ソースコードのSSH転送で自動的にパッケージがコンパイルされ、世界に使用出来るようになります
- ```
sudo add-apt-repository ppa:openrave/release
sudo apt-get update
sudo apt-get install openrave
```
- Debianソースパッケージ → バイナリーパッケージ
  - CMakeからのソースパッケージ作成
    - DebSourcePPA.cmakeを使用する
      - <https://openrave.svn.sourceforge.net/svnroot/openrave/trunk/modules-cmake/DebSourcePPA.cmake>

# Windowsパッケージ

- Nullsoft Scriptable Install System
  - <http://nsis.sourceforge.net/>
- Windows Registryの登録・調査
- 依存プログラムをネットからダウンロード
- 環境変数の設定

```
outFile "myinstaller.exe"
インストール先
InstallDir "$PROGRAMFILES\\MyProgram"

インストールのコンポーネント
section
 setOutPath $INSTDIR
 file test.txt
 writeUninstaller $INSTDIR\\uninstaller.exe
sectionEnd

アンインストール設定
section "Uninstall"
delete $INSTDIR\\uninstaller.exe
delete $INSTDIR\\test.txt
sectionEnd
```

# ROSパッケージ

<http://www.ros.org/wiki/release/Setup>

- ROSの依存関係の管理が違う
  - 機能単位はROSパッケージ
  - インストール単位はROSスタック
- WillowGarageのサーバーへリリースする

`roscpp release create.py stack_name stack_version distro_name`

- ビルドファーム
  - <http://build.willowgarage.com/>
- Ubuntuパッケージが出来上がり、ROSの正式なレポジトリに入る
- 注意：Ubuntu/Launchpadに収束しつつある



# リリースシステム立ち上げのお勧め

- CMakeで純粋なUbuntu/Windowsパッケージ作成
- ROSを使用するシステムを
  - コア：ROSに依存しない
    - 正式リリースし
  - 拡張機能：ROSとシステム使用
    - ROSスタックとしてWillowGarageにリリース
- 頻繁なリリース
  - テストシステムの速い段階の立ち上げ

# 開発の管理

- sourceforge, googlecode, launchpad
- オープンなソース管理ツール
  - subversion, git, mercurial, bazaar
- オープンなプロジェクト管理ツール
  - trac, redmine

# いいソフトウェアの重要点

- このツールを出来れば最初の段階でプロジェクトに入れる
  - 後で入れるのが非常に面倒です
- 開発者の責任
  - 自分が書いたコードの寿命
  - 波及効果
  - 時間の無駄を減らす

# 宿題用の個人レポジトリ

- これから宿題提出をSVNコミットで行う
  - メールしないでください！
- 作成と詳細情報

<http://code.google.com/p/rtm-ros-robotics/wiki/CreatePrivateRepository>

- 自分しかアクセス出来ない
- テストサーバーの立ち上げにも必要

# 宿題

[http://code.google.com/p/rtm-ros-robotics/wiki/Homework\\_3](http://code.google.com/p/rtm-ros-robotics/wiki/Homework_3)

- (1点) 課題
  - 画像から緑の楕円を抽出し、円の3次元位置を計算しているROSノードの作成
    - OpenCVにほとんどの機能が存在している
  - RVIZで円を表示する
- `agentsystem_ros_tutorials/opencv_fitting`を参照
  - `include/fitting.h`のAPI定義
  - `libfitting`ライブラリの作成
  - `fitting_node`のROSノード作成
  - `test/testfitting.cpp`のテスト登録

テスト対象



# 宿題

- (9点) 課題
  - ノードの単体・結合テスト
    - テストデータの作成
  - ドキュメント作成
    - プログラムの限界が分かるようにテスト結果も入れる。
  - Debianパッケージ
- 言語はC++かPython
- 一人プロジェクトになっているのでコピーされないようにご注意ください。
  - 教員・技術補佐が全コードをチェックします

14日（土）の宿題演習？