

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 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

News

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

MainPage

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
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

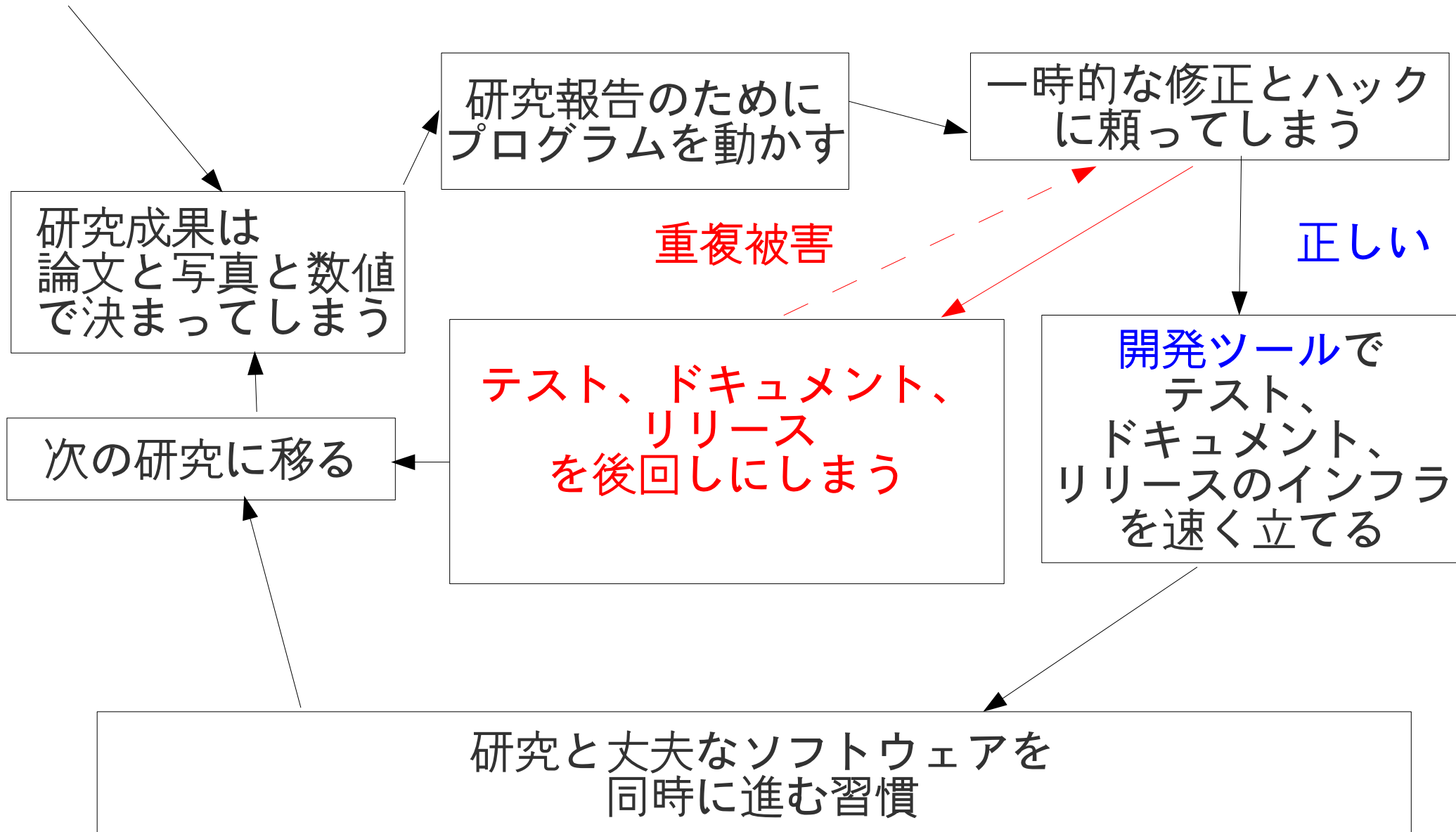
丈夫なソフトウェア目標

- 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
```

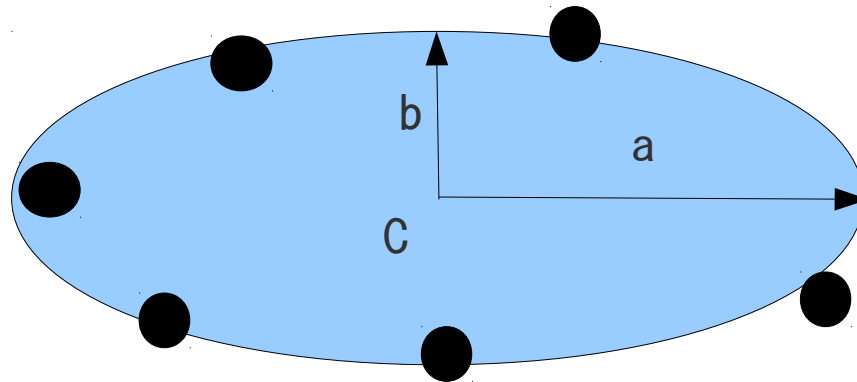
```
Ran 10 tests in 21.485s
```

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
 - $cx, cy, rotation, a, b, error \leq \text{fitEllipse(points)}$



- 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
 - $\text{fitEllipse(points)}$ 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

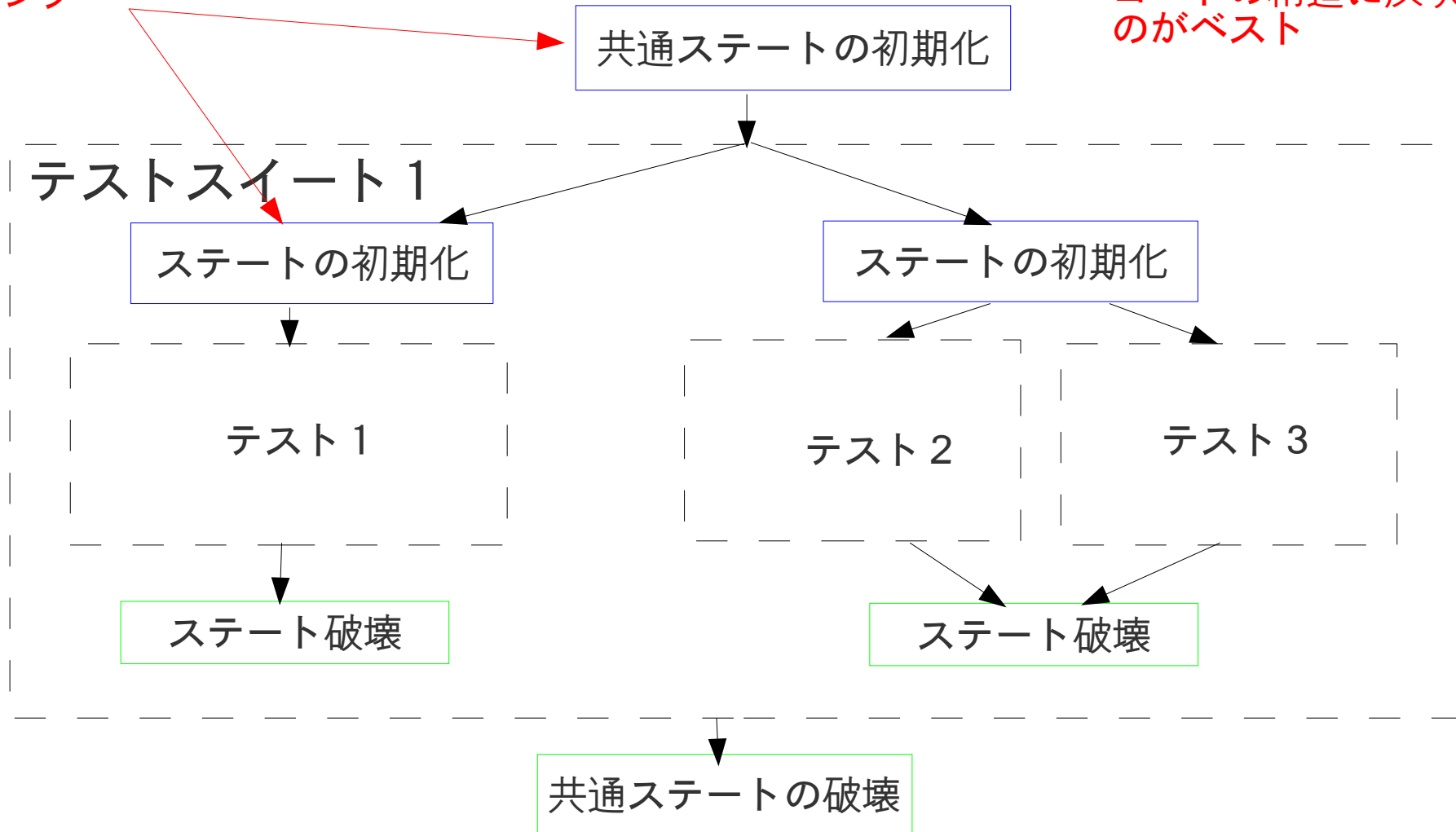
- Willow Garage
- 2日間で自律に
オフィスを回る



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()

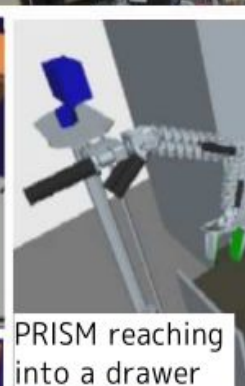
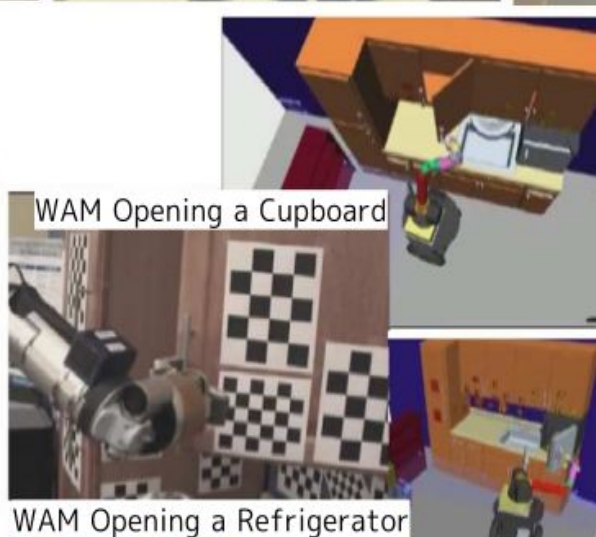
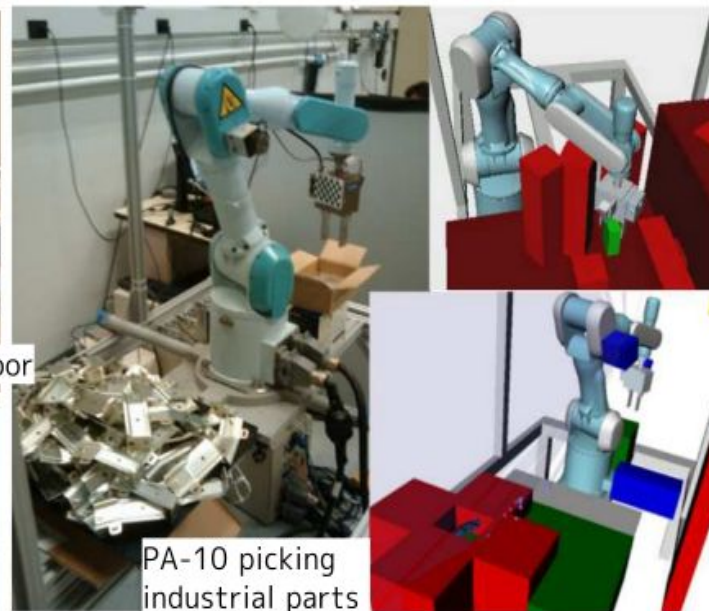
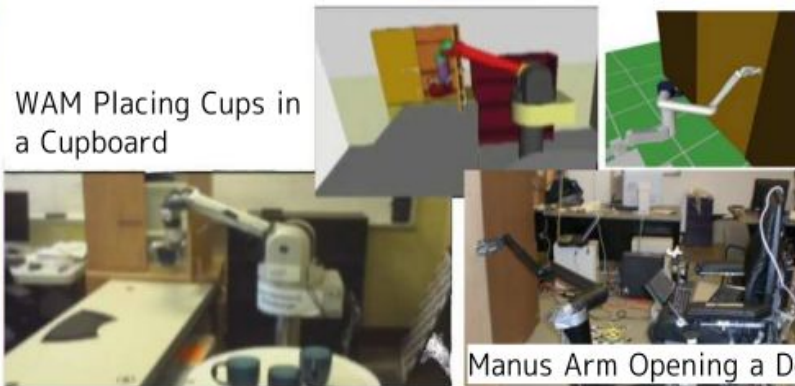
    def posecb(self, msg):
        assert( abs(msg.x - self.received.x) <= 0.5)
        assert( abs(msg.y - self.received.y) <= 0.5)
        assert( abs(msg.theta - self.received.theta) <= 0.2)
        self.received = True

    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で動作計画のテスト



OpenRAVEの基本テスト

- 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 navigation links like 'New Job', 'Manage Jenkins', 'People', 'Build History', etc. The main content area displays a table of jobs with columns for status, weather icon, 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

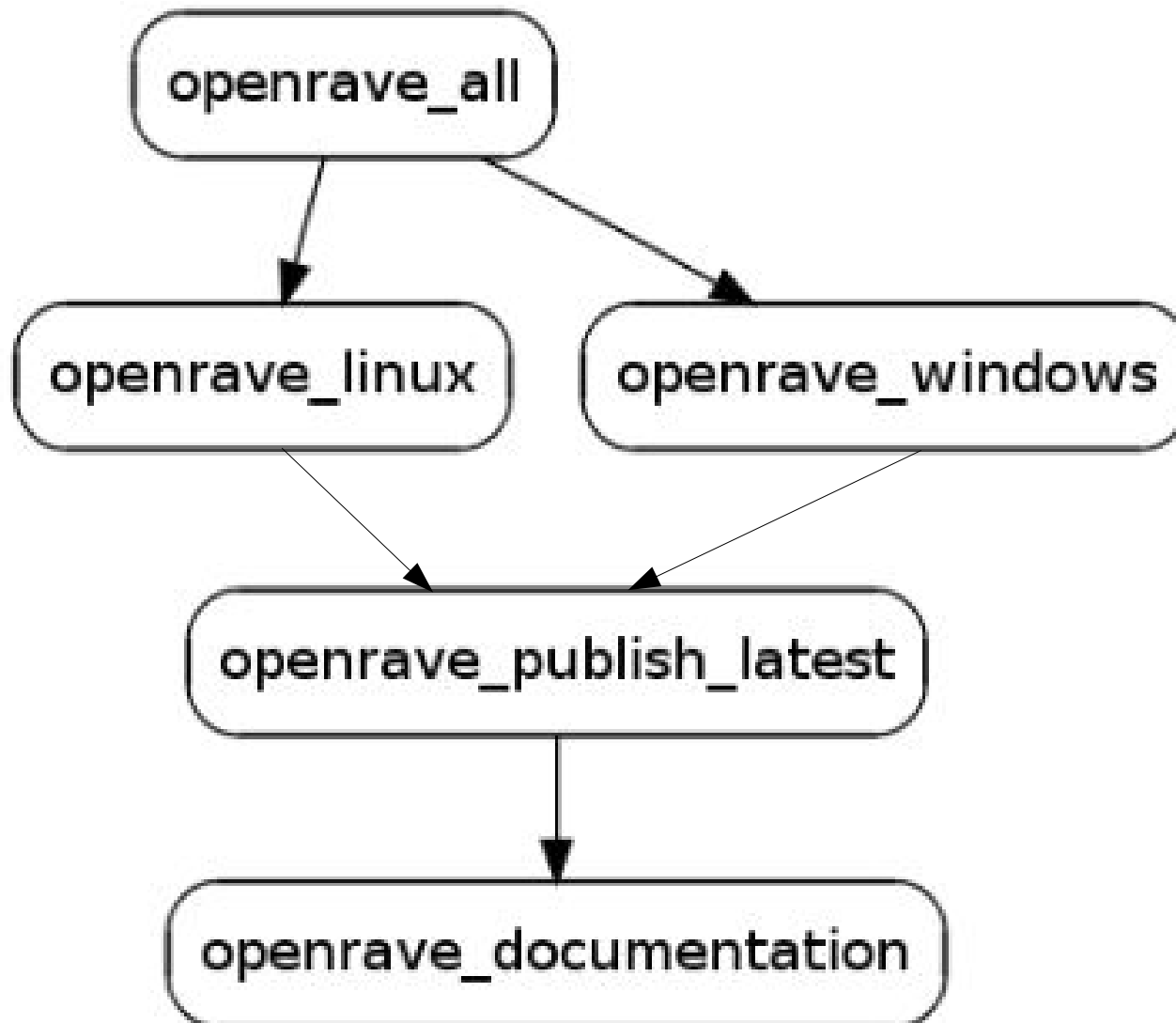
Icon: [S](#) [M](#) [L](#)

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>

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

- (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？