

ロボットミドルウェア

安藤慶昭

国立研究開発法人産業技術総合研究所

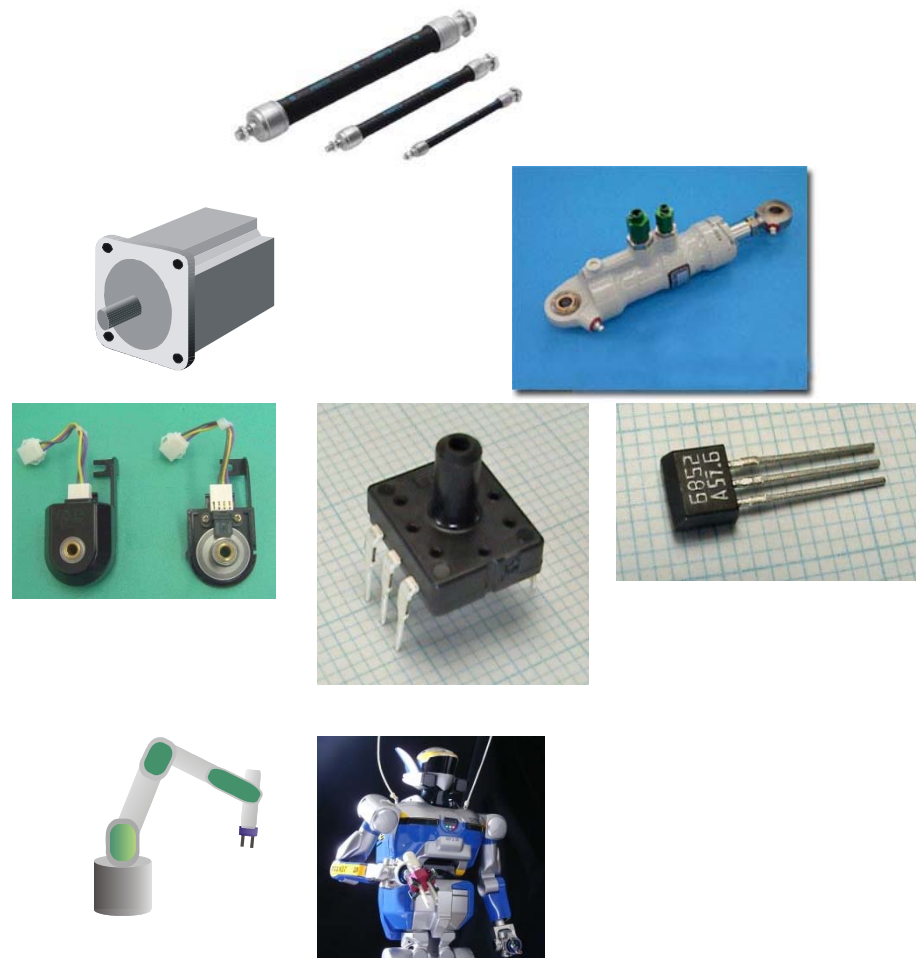
ロボットイノベーション研究センター

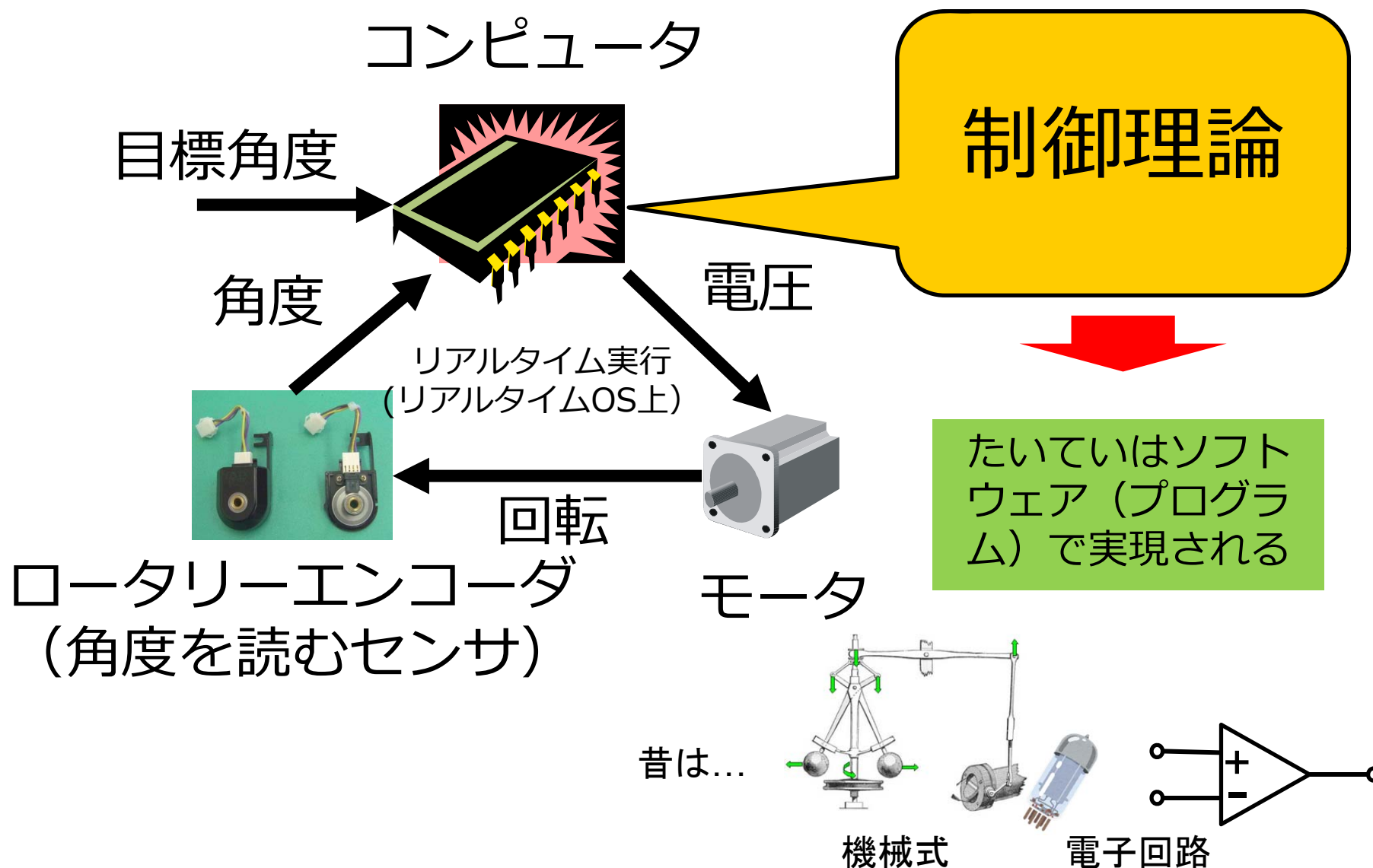
ロボットソフトウェアプラットフォーム研究チーム長

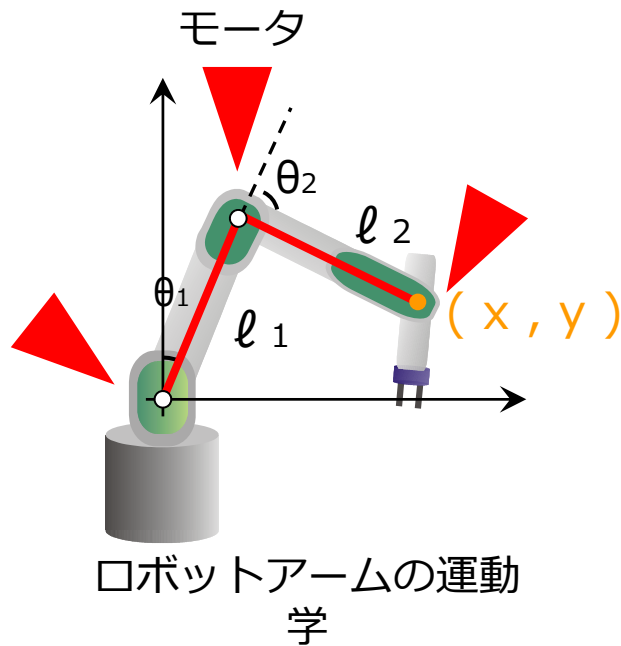


- ロボットとソフトウェア
- ロボットミドルウェアとは？
- RTミドルウェア：OpenRTM-aist
- 様々なミドルウェア/プラットフォーム
- 終わりに

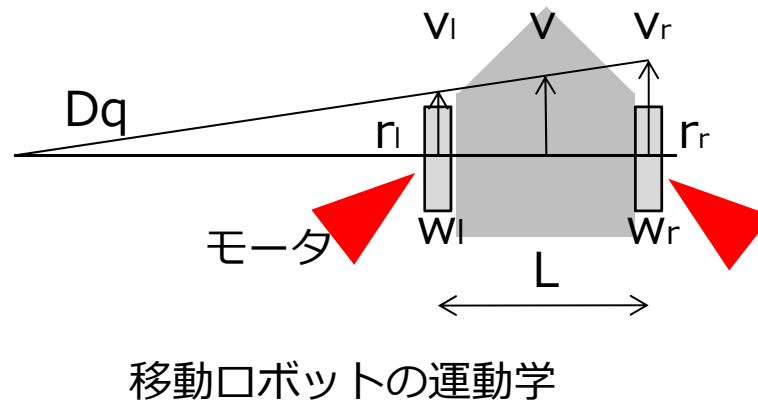
- センサ
 - エンコーダ
 - 力・磁気・加速度など
 - アクチュエータ
 - モータなど
 - 機構（メカ）
- +
- コンピュータ
 - ソフトウェア





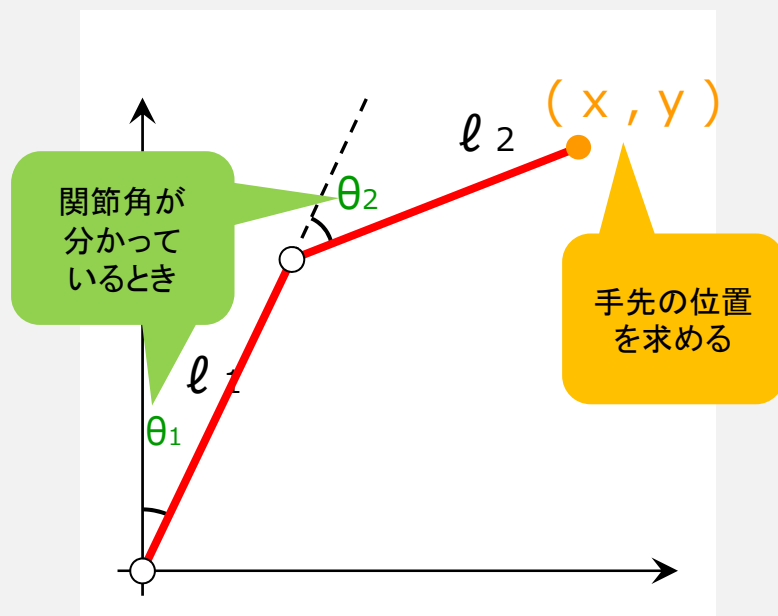


運動学 (kinematics) とは位置の移動とその時間変化を対応させて考える学問であり、数理的な手法であり、物理学の原理を基礎としていない。一般的に物体の状態は、移動と回転の運動学の両方を兼ね合わせて記述する。



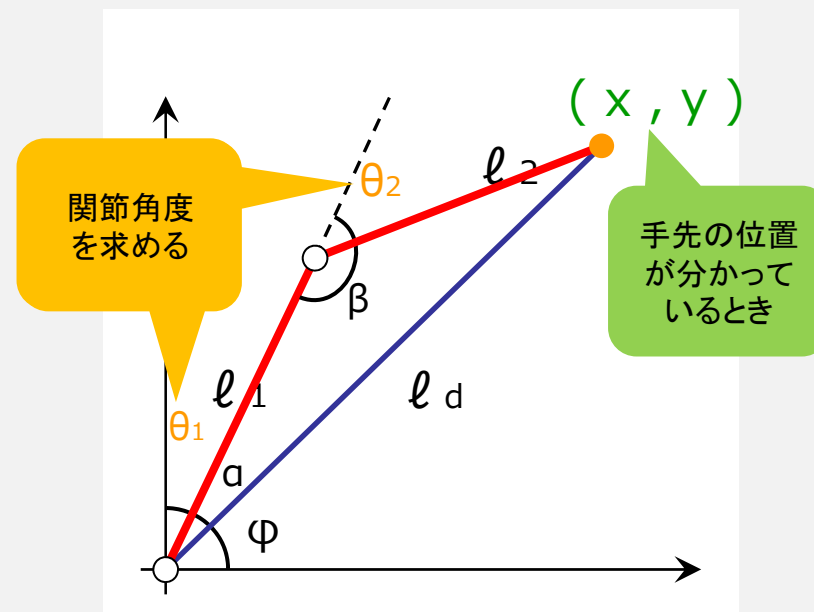
ロボットの機構（メカ）の運動を数式で（プログラムとして）記述する

● 順運動学



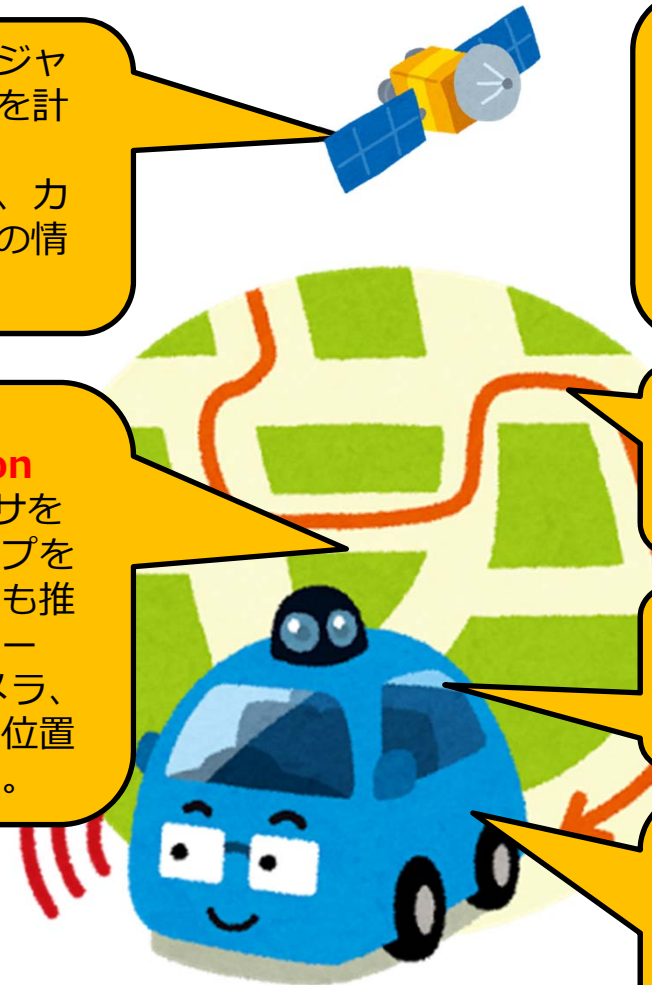
関節の角度がわかっているとき、XY座標を求めることができる

● 逆運動学



XY座標がわかっているとき、関節の角度を求めることができる

注：どの場合も腕の長さははじめから与えられている



内界センサ：車輪の角速度やジャイロ等ロボットの内部の情報を計測するセンサ

外界センサ：レーザーや音波、カメラ、GPS等ロボットの外部の情報を取得するセンサ

SLAM (スラムと読む、Simultaneous Localization and Mapping)：外界センサを用いて、ロボット周辺のマップを作成しながら同時に自己位置も推定する技術。センサには、レーザー（2次元、3次元）やカメラ、音波などが用いられる。相対位置を比較的安定的に推定できる。

自己位置推定 (Localization, ローカリゼーション、ローカライゼーション)：種々のセンサを利用し、ロボットの現在の位置を推定する技術。移動ロボットを制御するために最も基本的かつ必要とされる技術。

パスプランニング (Path Planning)：与えられたマップ上で、現在位置から目的地までの経路を計画する方法。

ナビゲーション(Navigation)：現在位置を推定しながらロボットを目的地まで移動させること。

デッドレコニング (Dead Reckoning)：車輪のエンコーダやジャイロ等内界センサのみ利用する自己位置を推定手法。誤差が蓄積するため長時間使用できない。オドメトリ(Odometry)と呼ばれることもある。

ロボットアーム：物体を把持するなどしてある位置からある位置まで移動させることを目的としたロボット。マニピュレータとも呼ばれる。

ティーチング・プレイバック：ロボットにあらかじめ覚えさせた動作を繰り返しさせること、またそうした利用方法。

ティーチング：ロボットに繰り返し動作させるための動き（手先の軌道）を覚えさせる作業。

位置制御：手先や関節を目的に位置または角度まで動かす制御。ほとんどのロボットアームは位置制御で利用されている。

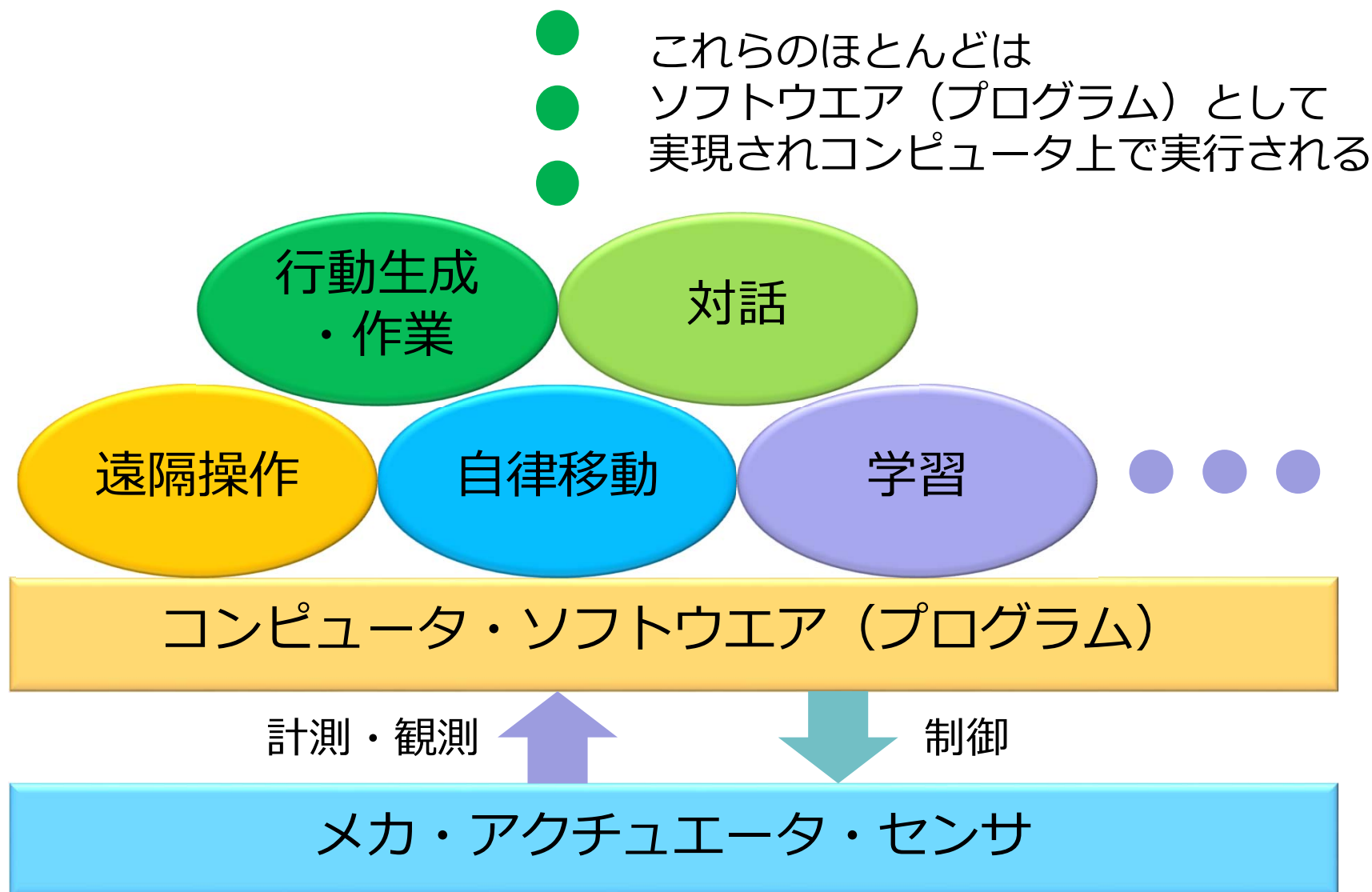
速度制御：手先や関節を目的の速度で制御する方法。

力制御：力センサやモータの電流を利用することで、手先や関節にかかる力を制御する方法。主に接触、倣い動作（コンプライアンス制御）が必要な作業で利用される。

自由度 (Degree of freedom)：制御できる軸の数。3次元空間内で、物体を任意の位置姿勢へ移動させるためには、6自由度以上が必要。

エンドエフェクタ (End-effector 終端(or 末端、手先) 効果器)：アームの先端に取り付けられた外界に働きかける部分。≡ロボットハンド、グリップなどとも呼ばれる。

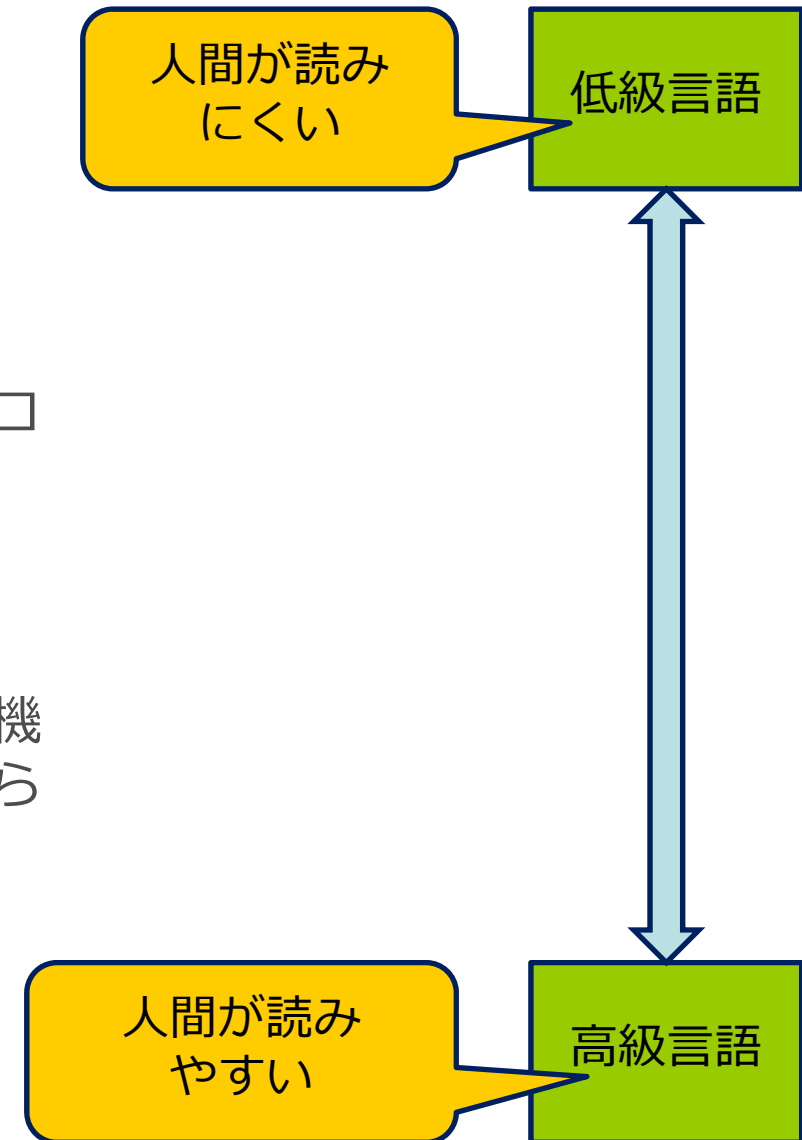
プランニング：現在位置から目的位置まで（主に手先を）どのようにアームを制御し動かすかを計画すること、またはその手法。



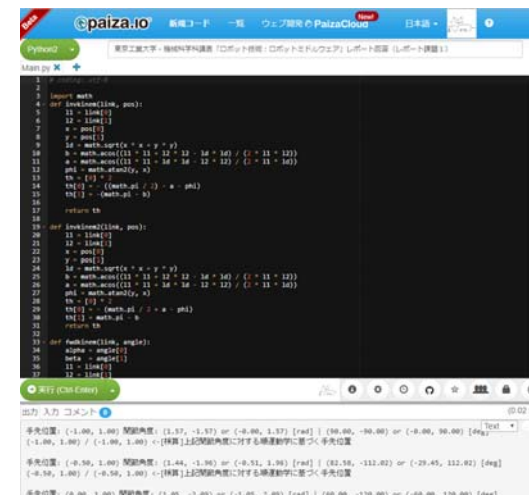
- プログラムとは？
 - コンピュータにやらせたいことを書いた手順書
 - 手順 = アルゴリズム (算法)
 - アルゴリズムを何らかの方法で書き表した物 = プログラム
 - コンピュータは機械語しか理解できない
 - 人間には機械語はわかりづらい

- プログラミング言語
 - プログラムはプログラミング言語で書く
 - 書いたもの：ソースコード (source code, テキストファイル)
 - プログラミング言語にはいろいろな種類がある
 - 何言語を習った？：C言語, Java, Python
 - それぞれ特徴、長所・短所がある (適材適所)

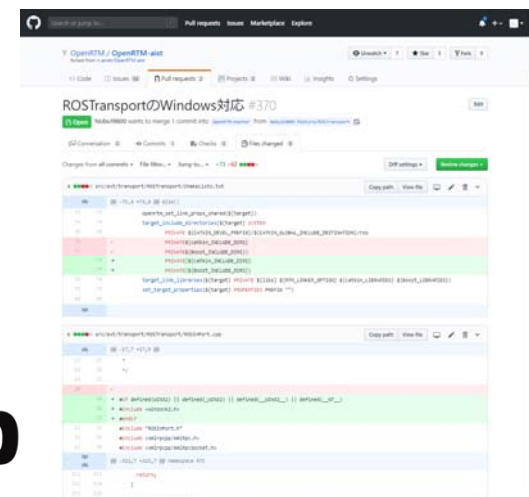
- 機械語
- アセンブリ言語
- コンパイル型言語
 - プログラムをすべて機械語に変換（コンパイル）してから実行
 - C、C++、FORTRAN、Pascal など
- 中間言語型
 - プログラムを仮想コンピュータ用の機械語（バイトコード）に翻訳してから仮想コンピュータ上で実行
 - Java、.NET（C#など）
- インタプリタ型
 - プログラムを一行ずつ実行
 - Ruby、Python、Perl、BASIC



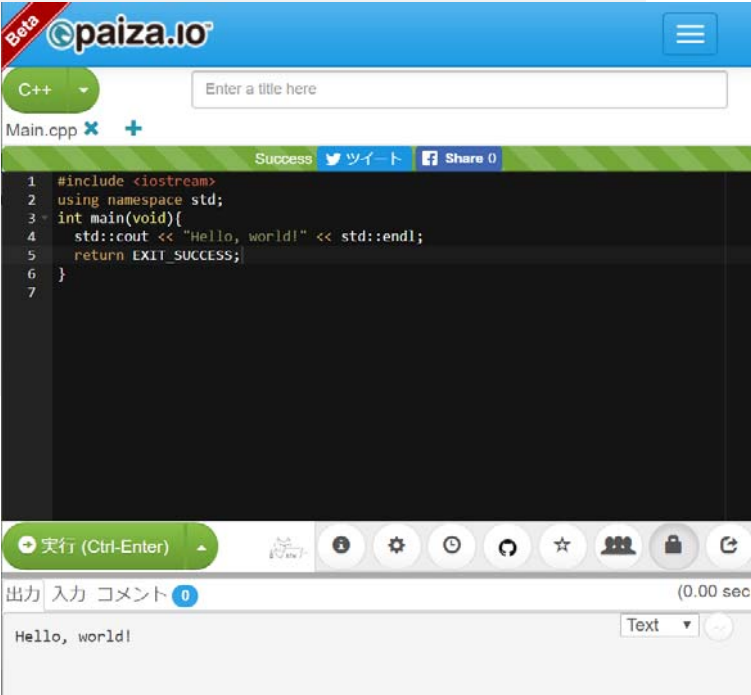
- 文法を覚える
- コンパイル、実行の方法を覚える
 - 実際にプログラムを書いてみる
 - ブラウザで試すこともできる (例: paiza.io)
- 関数やライブラリの使い方を覚える
 - いっぺんに覚える必要はない、少しずつ
- デバッグの方法を覚える
 - 間違いを見つけ、修正する
- 人のプログラムを読む
 - 今や、多くのソースコードを入手可能 (例: github.com)
 - 簡単に優れたプログラムを利用・読むことができる
 - 読むことは書くことより難しい！



プログラミングの敷居はここ数年で一気に下がった。ぜひ、試してみてください！！



```
#include <cstdlib>
#include <iostream>
int main ()
{
    std::cout << "Hello, world!" << std::endl;
    return EXIT_SUCCESS;
}
```



The screenshot shows the paiza.io web interface. At the top, there's a blue header with the paiza.io logo and a 'Beta' badge. Below the header, there's a green bar with 'C++' selected and a text input field 'Enter a title here'. The main area is a dark-themed code editor showing the following C++ code:

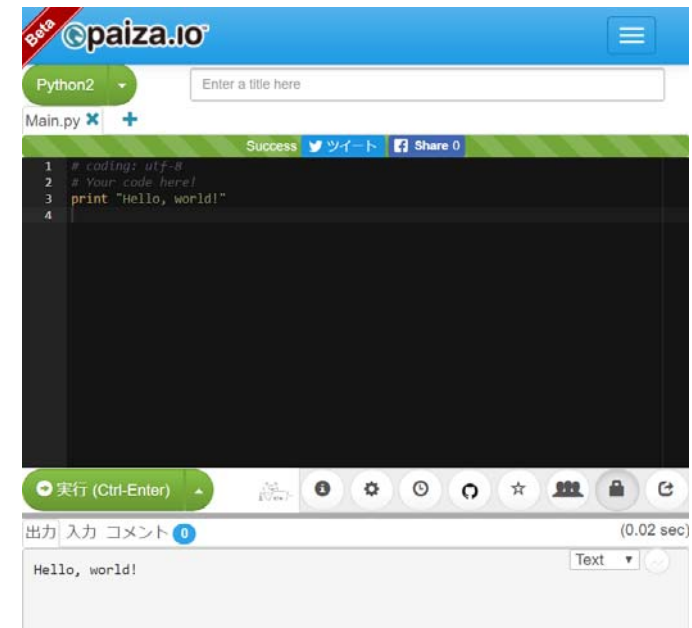
```
1 #include <iostream>
2 using namespace std;
3 int main(void){
4     std::cout << "Hello, world!" << std::endl;
5     return EXIT_SUCCESS;
6 }
7
```

Below the code editor, there's a green bar with '実行 (Ctrl-Enter)' and several icons. The output section shows 'Hello, world!' and a 'Text' dropdown menu. The status bar at the bottom indicates '(0.00 sec)'.

```
print "Hello, world!"
```

(Version 3以降は)

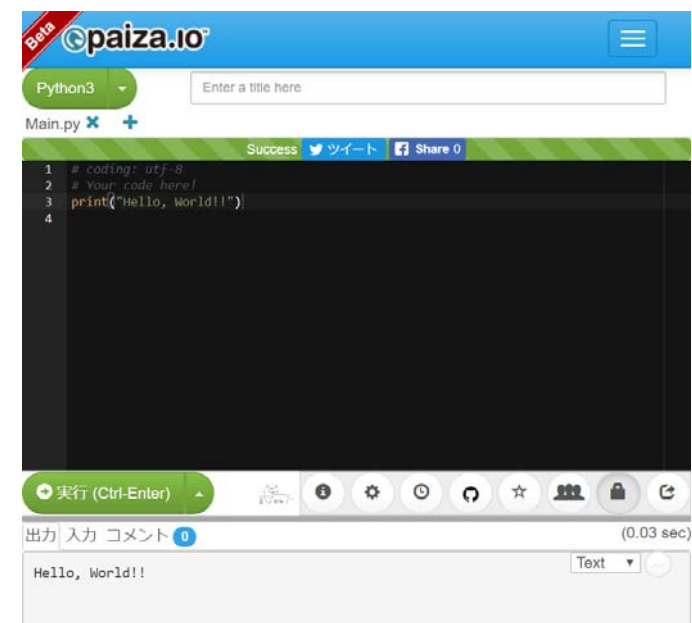
```
print("Hello, world!")
```



The screenshot shows the paiza.io interface for Python2. The code editor contains the following code:

```
1 # coding: utf-8
2 # Your code here!
3 print "Hello, world!"
4
```

The execution button is labeled "実行 (Ctrl-Enter)". The output area shows "Hello, world!" and the execution time is "(0.02 sec)".



The screenshot shows the paiza.io interface for Python3. The code editor contains the following code:

```
1 # coding: utf-8
2 # Your code here!
3 print("Hello, world!")
4
```

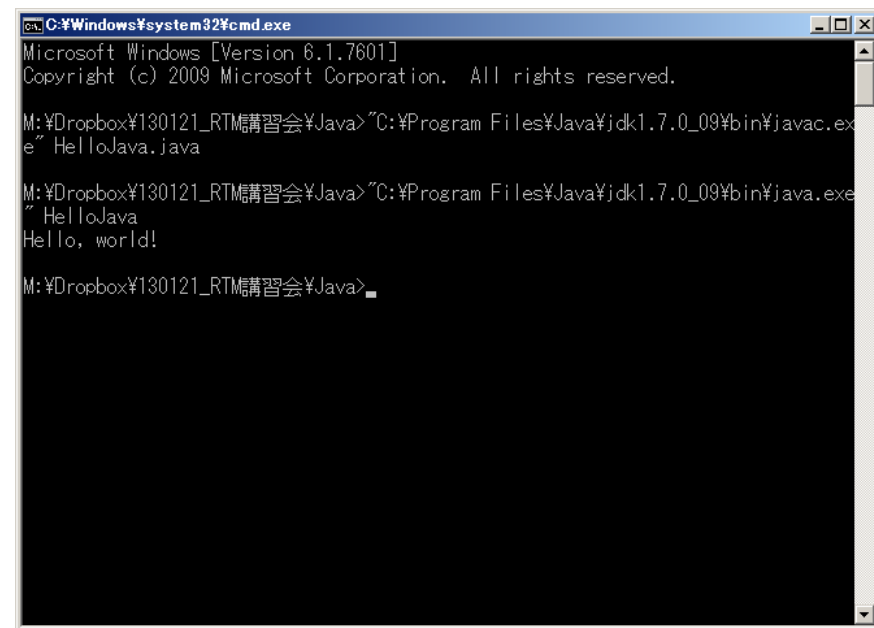
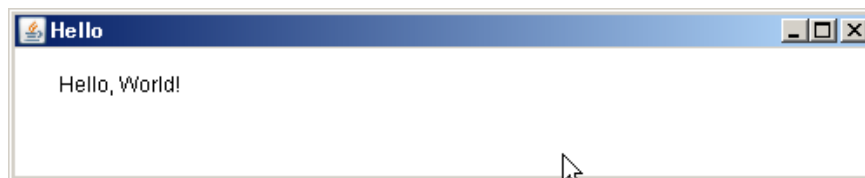
The execution button is labeled "実行 (Ctrl-Enter)". The output area shows "Hello, World!!" and the execution time is "(0.03 sec)".

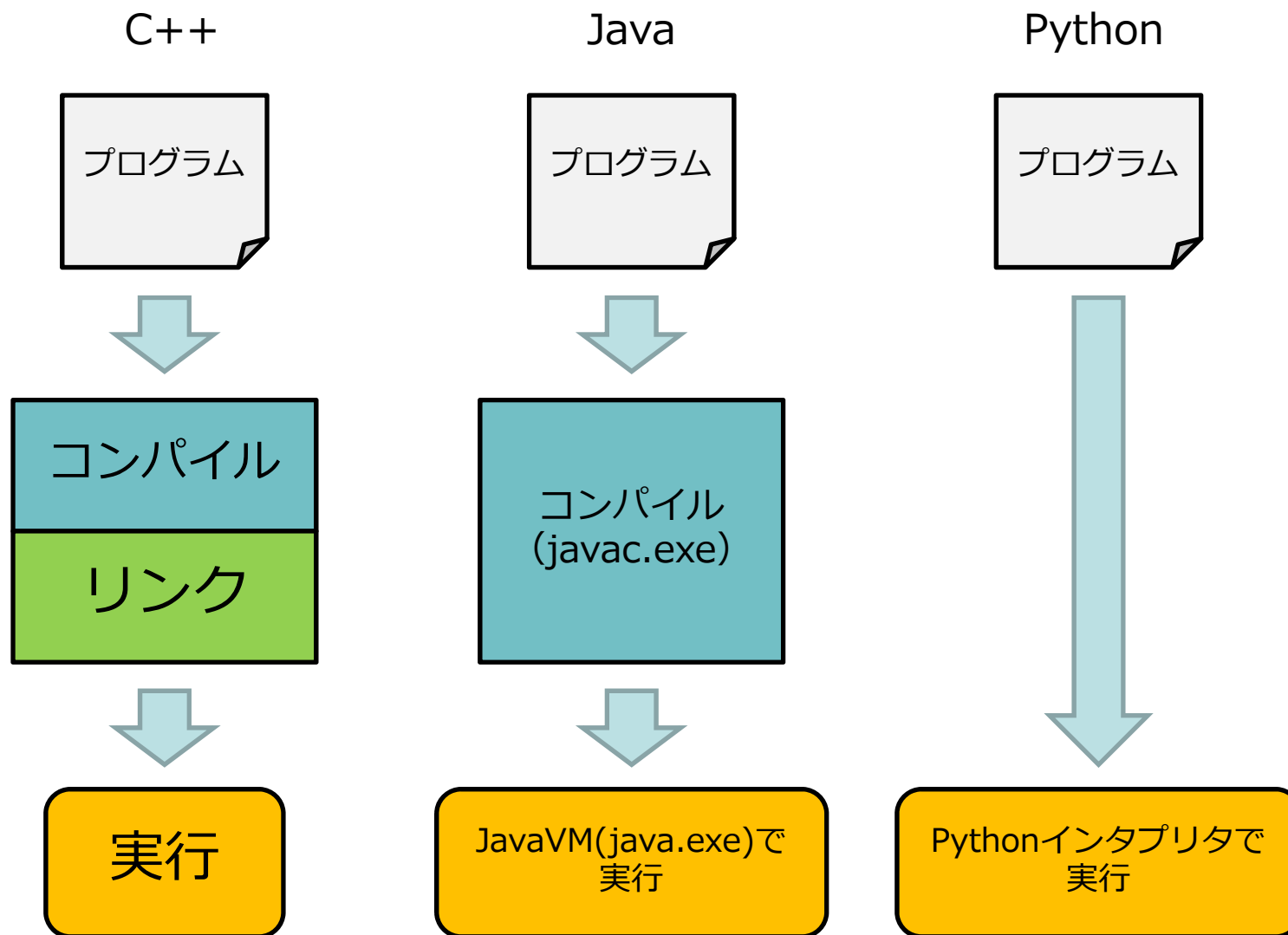
```
import java.awt.*;
import java.awt.event.*;

public class HelloFrame extends Frame {
    HelloFrame(String title) {
        super(title);
    }
    public void paint(Graphics g) {
        super.paint(g);
        Insets ins = this.getInsets();
        g.drawString("Hello, World!", ins.left + 25, ins.top + 25);
    }
    public static void main(String[] args) {
        HelloFrame fr = new HelloFrame("Hello");

        fr.addWindowListener(
            new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    System.exit(0);
                }
            }
        );
        fr.setResizable(true);
        fr.setSize(500, 100);
        fr.setVisible(true);
    }
}
```

```
class HelloJava {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```





※VM=仮想マシン

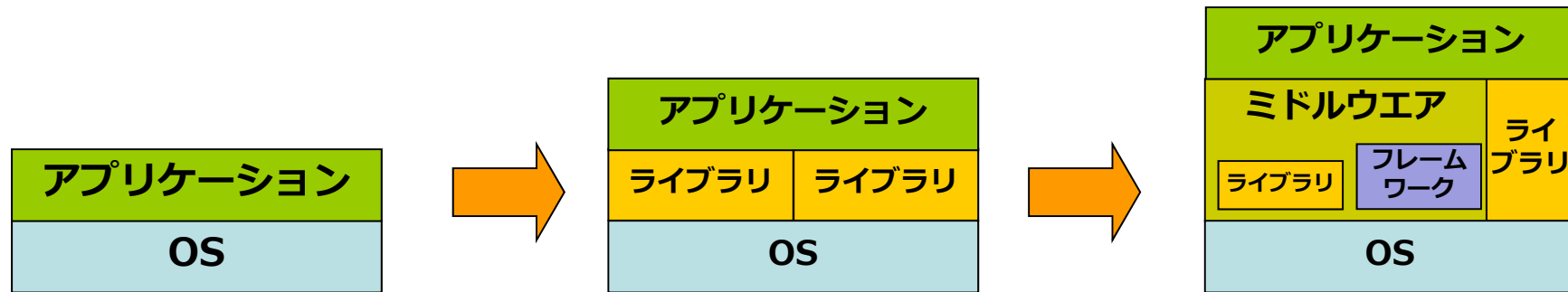
※インタプリタ：プログラムを一行ずつ読み込みながら逐次実行するプログラム

| | 実行速度 | リアルタイム実行 | コンパイル | オブジェクト指向 | 手軽さ |
|--------|------|----------|-------|----------|-----|
| C/C++ | ◎ | ◎ | 必要 | △／○ | △ |
| Java | ○ | △ | 必要 | ◎ | ○ |
| Python | × | × | 不要 | ○ | ◎ |

| | 移植性 | 動的処理 | GUIの作りやすさ | 学習のしやすさ | 大規模開発 |
|--------|-----|------|-----------|---------|-------|
| C/C++ | △ | × | × | △／× | ○／◎ |
| Java | ○ | △ | △ | ○ | ◎ |
| Python | ○ | ○ | ○ | ◎ | △ |

用途・目的に応じて適切な
プログラミング言語を選択することが重要

ロボットミドルウェアとは？



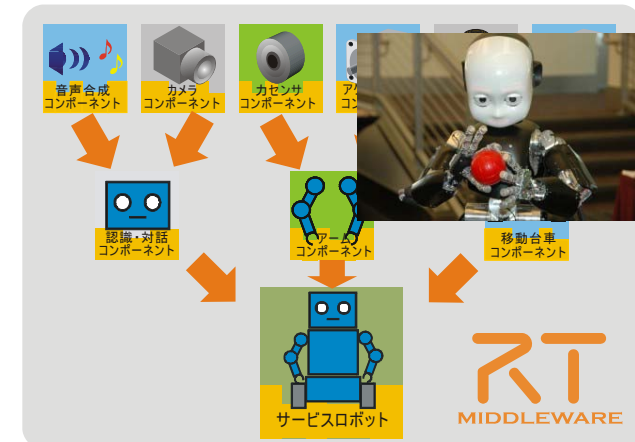
はじめはOSの上に直接アプリケーションソフトウェアを構築していたが…

アプリケーションが複雑化してくると、よく使う機能、共通の機能は、「**ライブラリ**」としてまとめて、使いまわすようになる。

さらに複雑化すると、特定用途のアプリケーション向けに、**便利機能**（ライブラリ、フレームワーク、etc.）をまとめたレイヤ「**ミドルウェア**」がOSとアプリの間（ミドル）に出現。

ミドルウェアの代表例：データベース、通信ミドルウェア、Webサーバ+アプリケーション実行サーバ、等

- ロボットシステム構築を効率化するための共通機能を提供する**基盤ソフトウェア**
 - － 「ロボットOS」と呼ばれることもある
 - － インターフェース・プロトコルの共通化、標準化
 - － 例として
 - モジュール化フレームワークを提供
 - モジュール間通信機能を提供
 - パラメータの設定、配置、起動等を管理
 - OSや言語間連携・相互運用を実現
- 2000年ごろから開発が活発化
 - － 世界各国で様々なミドルウェアが開発・公開されている



RTミドルウェア

ROS

ROS1/ROS2



OROCOS

YARP

YARP

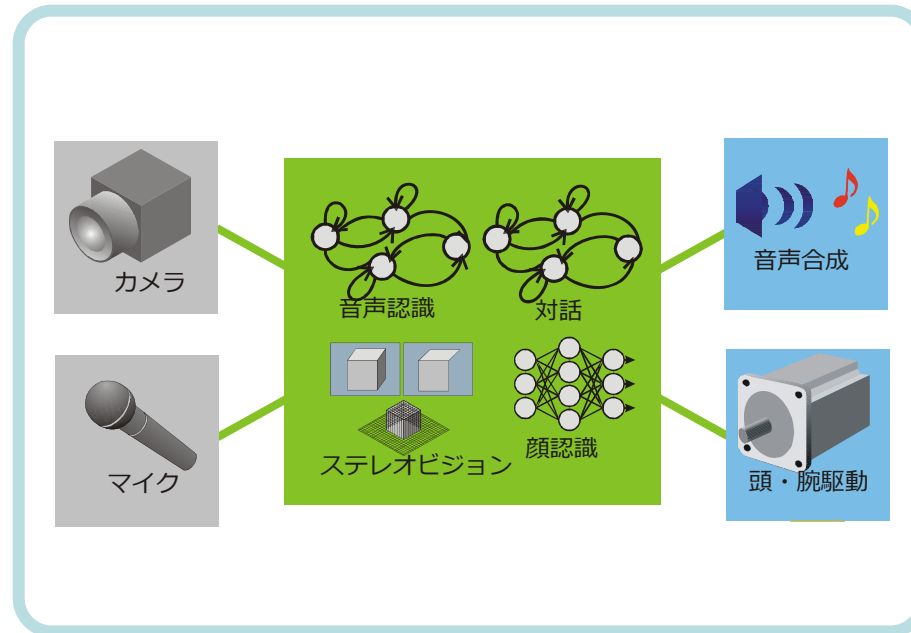


ORiN

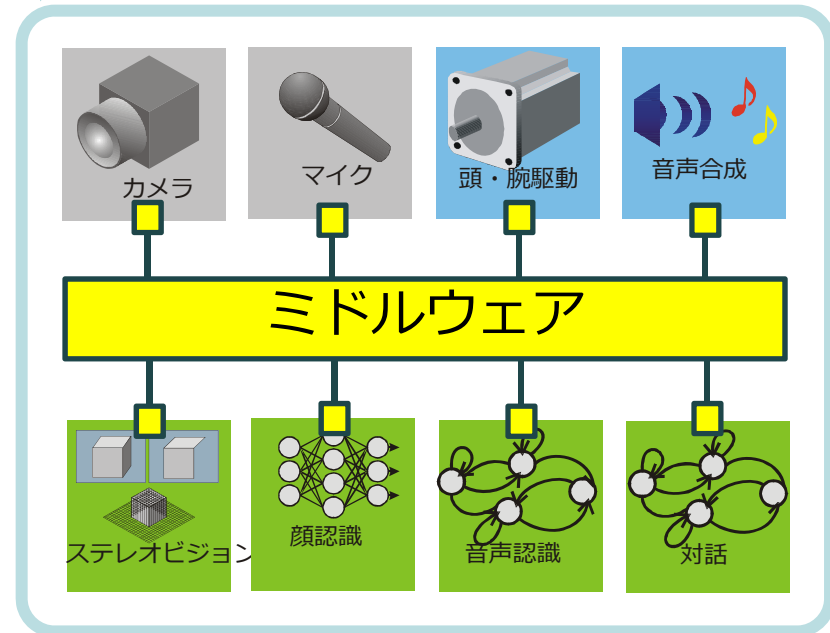


NAOqi

従来型開発



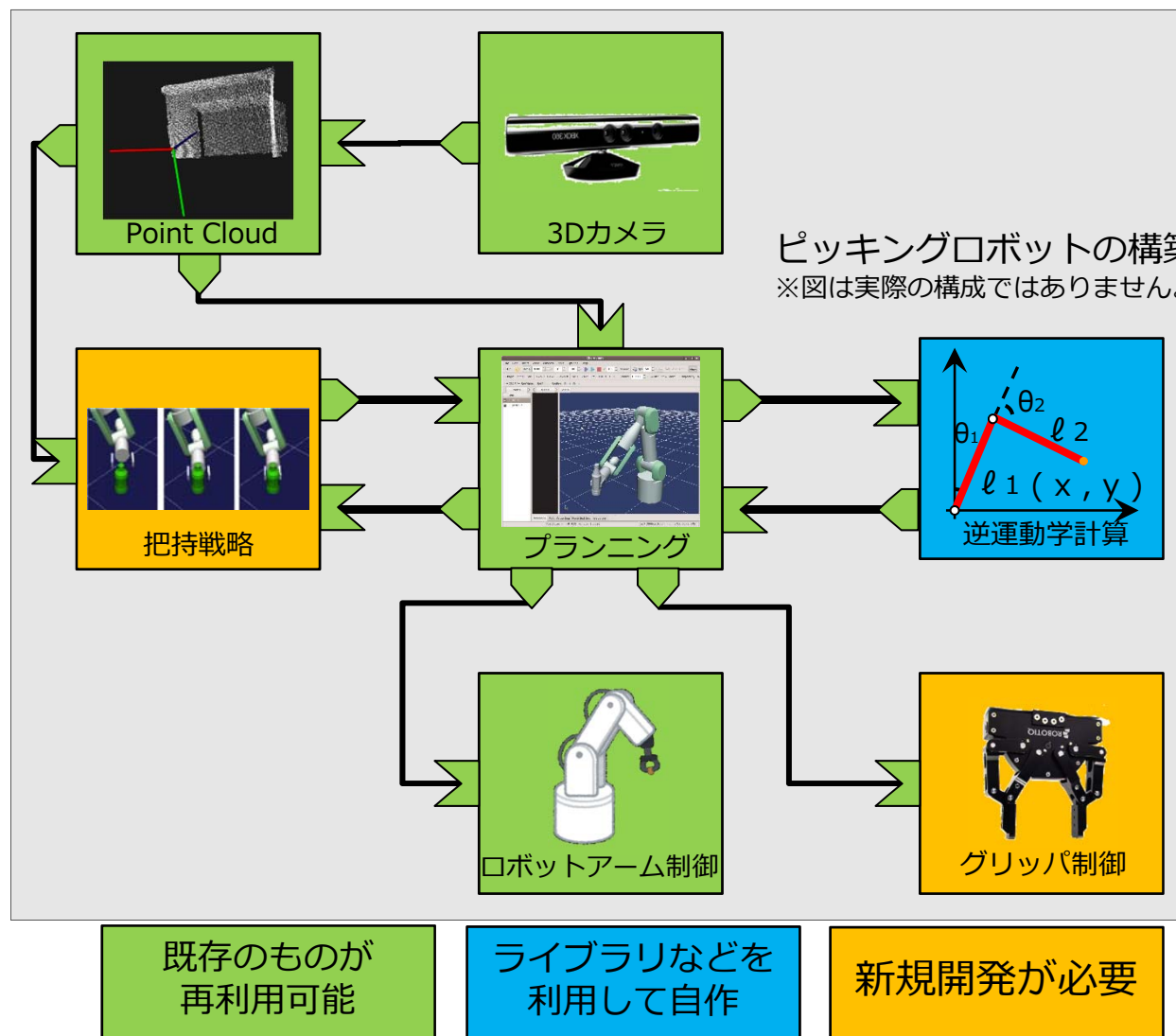
コンポーネント指向開発



- ✓ 様々な機能を融合的に設計
- ✓ 実行時効率は高いが、柔軟性に欠ける
- ✓ 複雑化してくると開発が困難に

- ✓ 大規模複雑な機能の分割・統合
- ✓ 開発・保守効率化（機能の再利用等）
- ✓ システムの柔軟性向上

- 再利用性の向上
 - 同じモジュールを様々なシステムに使いまわせる
- 選択肢の多様化
 - 同じ機能を持つ多様なモジュールを利用可能
- 柔軟性の向上
 - モジュールの接続構成を変えるだけで様々なシステムを構築可能
- 信頼性の向上
 - モジュール単位でテスト可能なため信頼性の向上につながる
- 堅牢性の向上
 - システムがモジュールで分割されているので、一つの問題が全体に波及しにくい



ミドルウェアを利用すると、**既存のモジュール**が利用できる

開発するときに**新規に作らなければならない部分**は少なくて済む

RTミドルウェア OpenRTM-aist



RTとは?

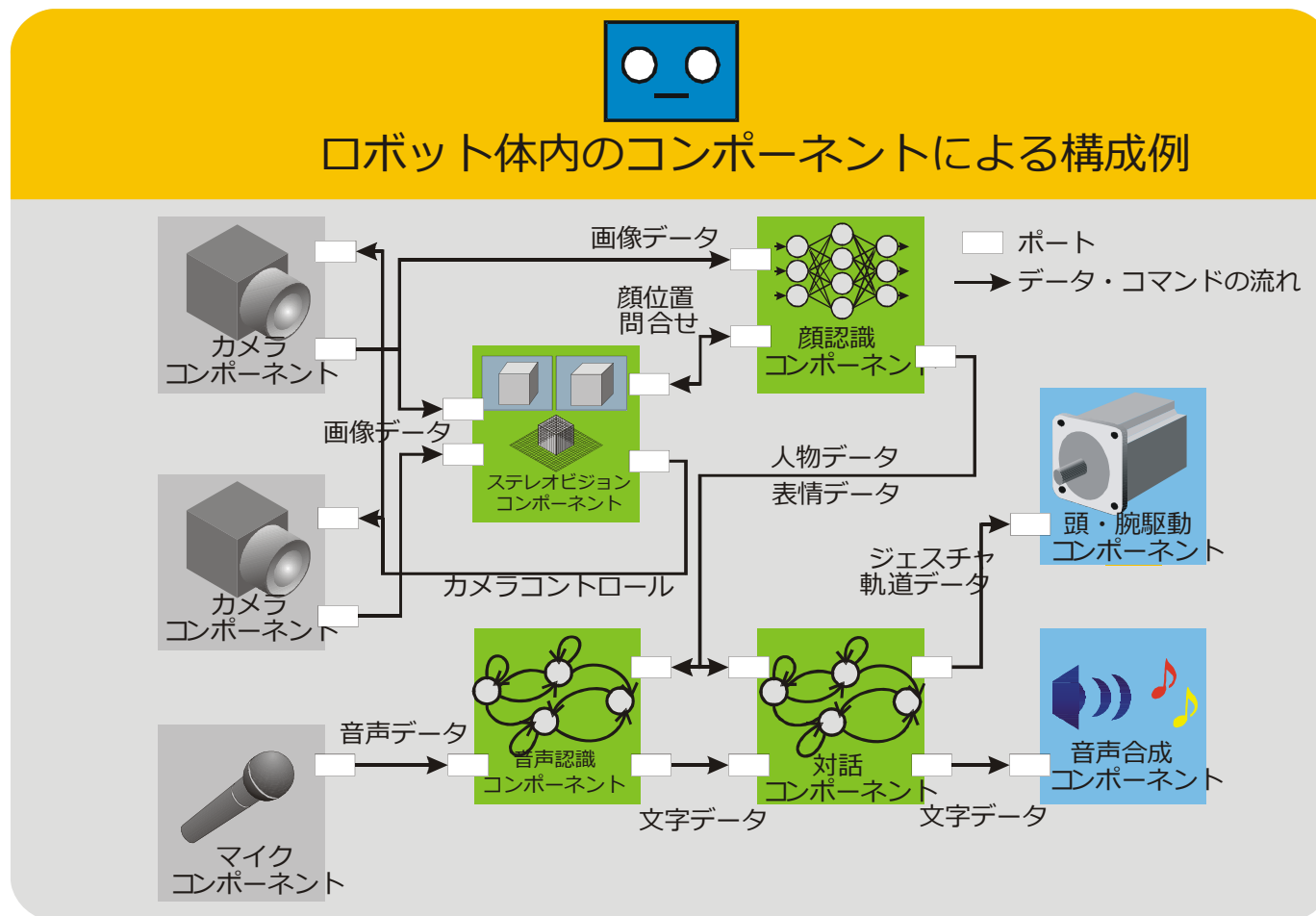
- RT = Robot Technology cf. IT
 - ≠Real-time
 - 単体のロボットだけでなく、さまざまなロボット技術に基づく機能要素をも含む (センサ、アクチュエータ, 制御スキーム、アルゴリズム、etc….)

産総研版RTミドルウェア

OpenRTM-aist

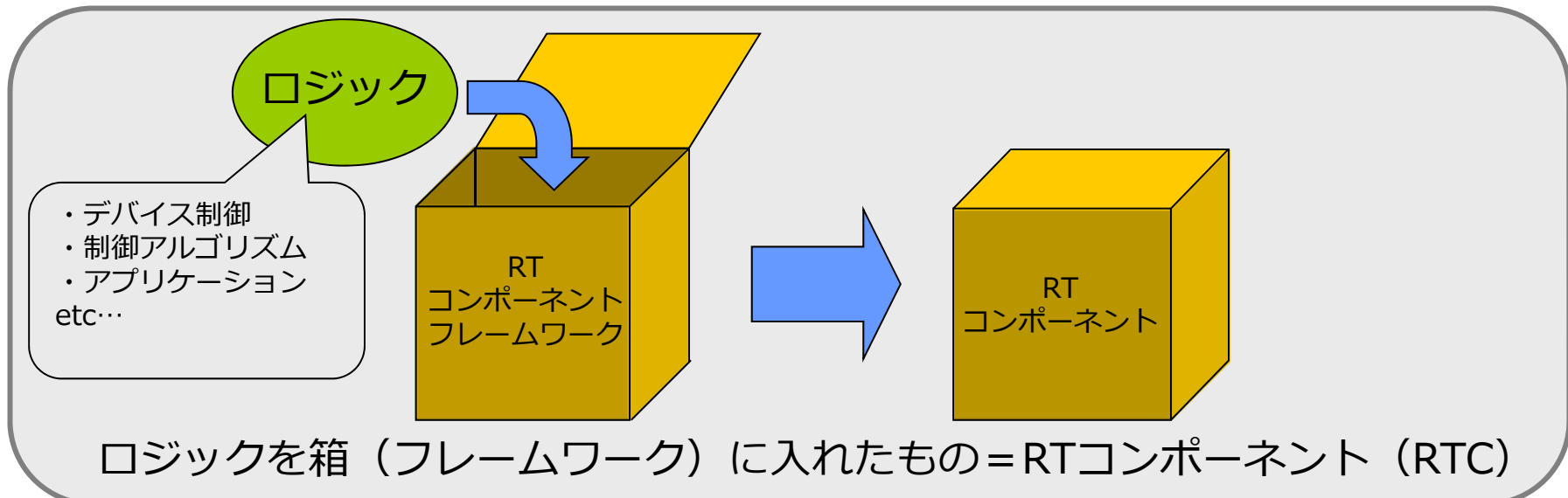
- RT-Middleware (RTM)
 - RT要素のインテグレーションのためのミドルウェア
- RT-Component (RTC)
 - RT-Middlewareにおけるソフトウェアの基本単位

ロボットシステムの構造



多くの機能要素（コンポーネント）から構成される

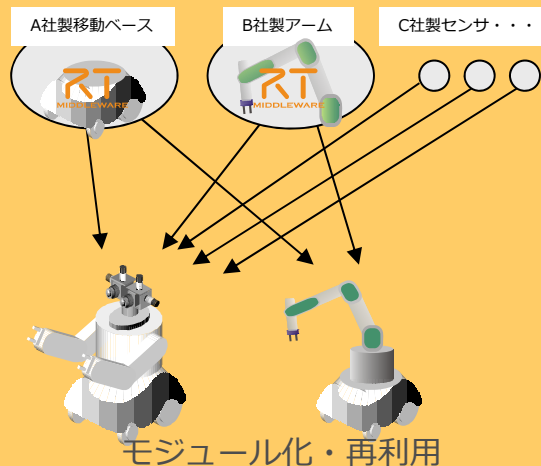
RTミドルウェアとRTコンポーネント



モジュール化のメリットに加えて

- ソフトウェアパターンを提供
 - ロボットに特有のソフトウェアパターンを提供することで、体系的なシステム構築が可能
- フレームワークの提供
 - フレームワークが提供されているので、コアのロジックに集中できる
- 分散ミドルウェア
 - ロボット体内LANやネットワークロボットなど、分散システムを容易に構築可能

コストの問題



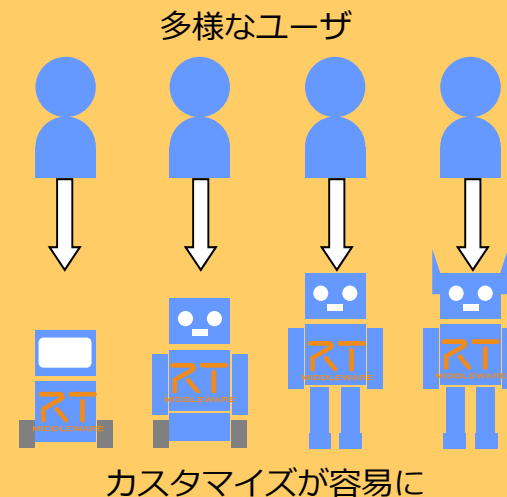
ロボットの低コスト化

技術の問題



最新技術を利用可能

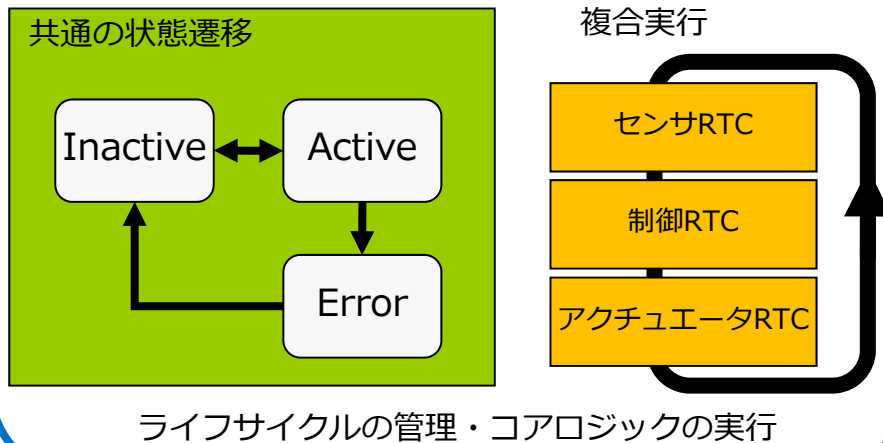
ニーズの問題



多様なニーズに対応

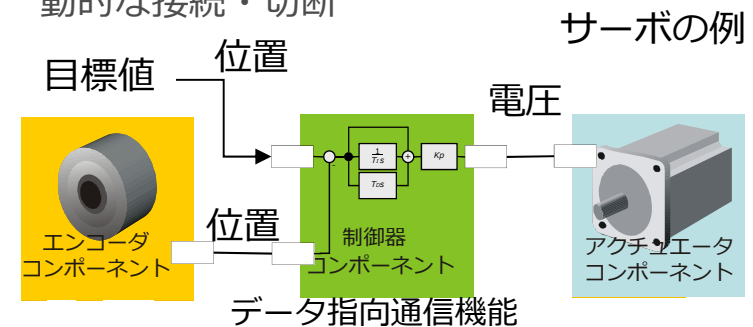
ロボットシステムインテグレーションによるイノベーション

アクティビティ・実行コンテキスト



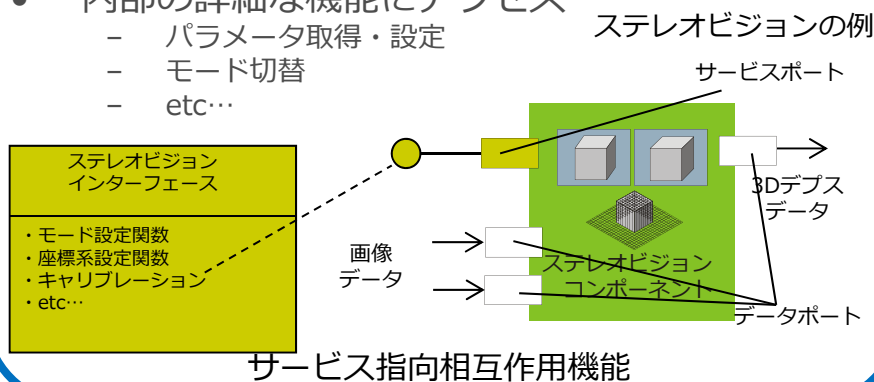
データポート

- データ指向ポート
- 連続的なデータの送受信
- 動的な接続・切断



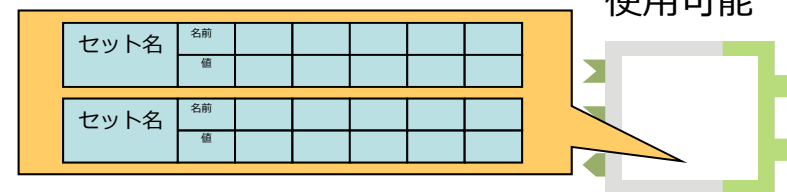
サービスポート

- 定義可能なインターフェースを持つ
- 内部の詳細な機能にアクセス
 - パラメータ取得・設定
 - モード切替
 - etc...



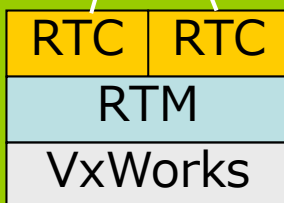
コンフィギュレーション

- パラメータを保持する仕組み
 - いくつかのセットを保持可能
 - 実行時に動的に変更可能
- 複数のセットを動作時に切り替えて使用可能

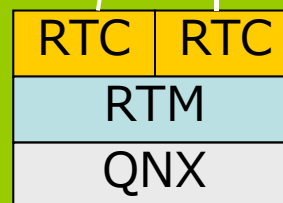


RTMにより、
ネットワーク上に
分散するRTCを
OS・言語の壁を
越えて接続する
ことができる。

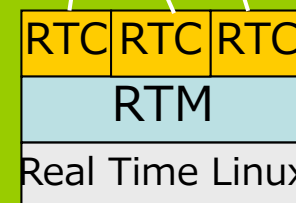
ロボットA



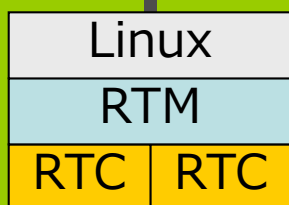
ロボットB



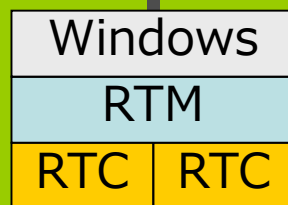
ロボットC



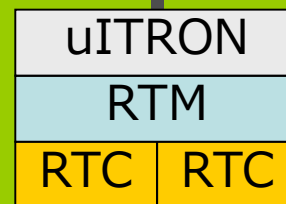
ネットワーク



アプリケーション



操作デバイス



センサ

RTC同士の接続
は、プログラム
実行中に動的に
行うことができる。

Date: September 2012



Robotic Technology Component (RTC)

Version 1.1

Normative reference: <http://www.omg.org/spec/RTC/1.1>
 Machine consumable files: <http://www.omg.org/spec/RTC/20111205/>
 Normative:
<http://www.omg.org/spec/RTC/20111205/rtc.xml>
<http://www.omg.org/spec/RTC/20111205/rtc.h>
<http://www.omg.org/spec/RTC/20111205/rtc.idl>
 Non-normative:
<http://www.omg.org/spec/RTC/20111205/rtc.eap>

標準化履歴

- 2005年9月
Request for Proposal 発行(標準化開始)
- 2006年9月
OMGで承認、事実上の国際標準獲得
- 2008年4月
OMG RTC標準仕様 ver.1.0公式リリース
- 2012年9月
ver. 1.1改定
- 2015年9月
FSM4RTC(FSM型RTCとデータポート標準) 採択

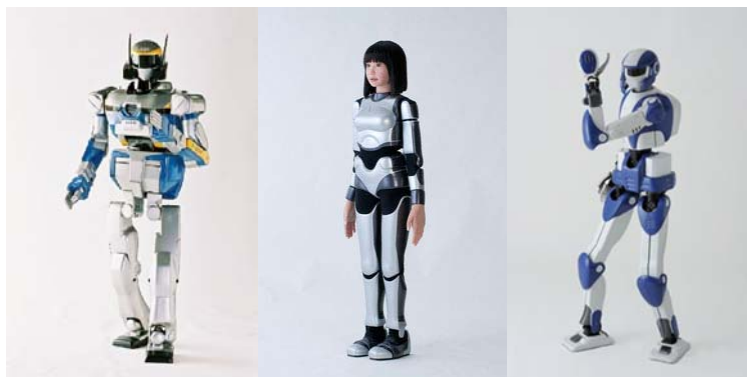
OMG国際標準

- 標準化組織で手続きに沿って策定
 → 1組織では勝手に改変できない安心感
 → 多くの互換実装ができつつある
 → 競争と相互運用性が促進される

RTミドルウェア互換実装は10種類以上

| 名称 | ベンダ | 特徴 | 互換性 |
|----------------|----------------|----------------------------------|-----|
| OpenRTM-aist | 産総研 | NEDO PJで開発。参照実装。 | --- |
| HRTM | ホンダ | アシモはHRTMへ移行中 | ◎ |
| OpenRTM.NET | セック | .NET(C#, VB, C++/CLI, F#, etc..) | ◎ |
| RTM on Android | セック | Android版RTミドルウェア | ◎ |
| RTC-Lite | 産総研 | PIC, dsPIC上の実装 | ○ |
| Mini/MicorRTC | SEC | NEDOオープンイノベーションPJで開発 | ○ |
| RTMSafety | SEC/AIST | NEDO知能化PJで開発・機能安全認証取得 | ○ |
| RTC CANOpen | SIT, CiA | CAN業界RTM標準 | ○ |
| PALRO | 富士ソフト | 小型ヒューマノイドのためのC++ PSM 実装 | × |
| OPRoS | ETRI | 韓国国家プロジェクトでの実装 | × |
| GostaiRTC | GOSTAI, THALES | ロボット言語上で動作するC++ PSM実装 | × |

特定のベンダが撤退しても
ユーザは使い続けることが可能



HRPシリーズ: 川田工業、AIST

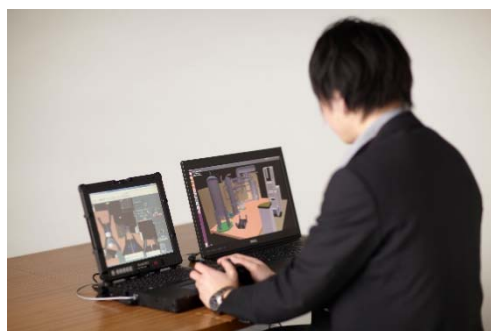


S-ONE : SCHAFT



DAQ-Middleware: KEK/J-PARC

KEK: High Energy Accelerator Research Organization
J-PARC: Japan Proton Accelerator Research Complex



災害対応ロボット操縦シミュレータ : HIRO, NEXTAGE open: Kawada Robotics
NEDO/千葉工大



RAPUDA : Life Robotics



ビュートローバーRTC/RTC-BT(VSTONE)



OROCHI (アールティ)



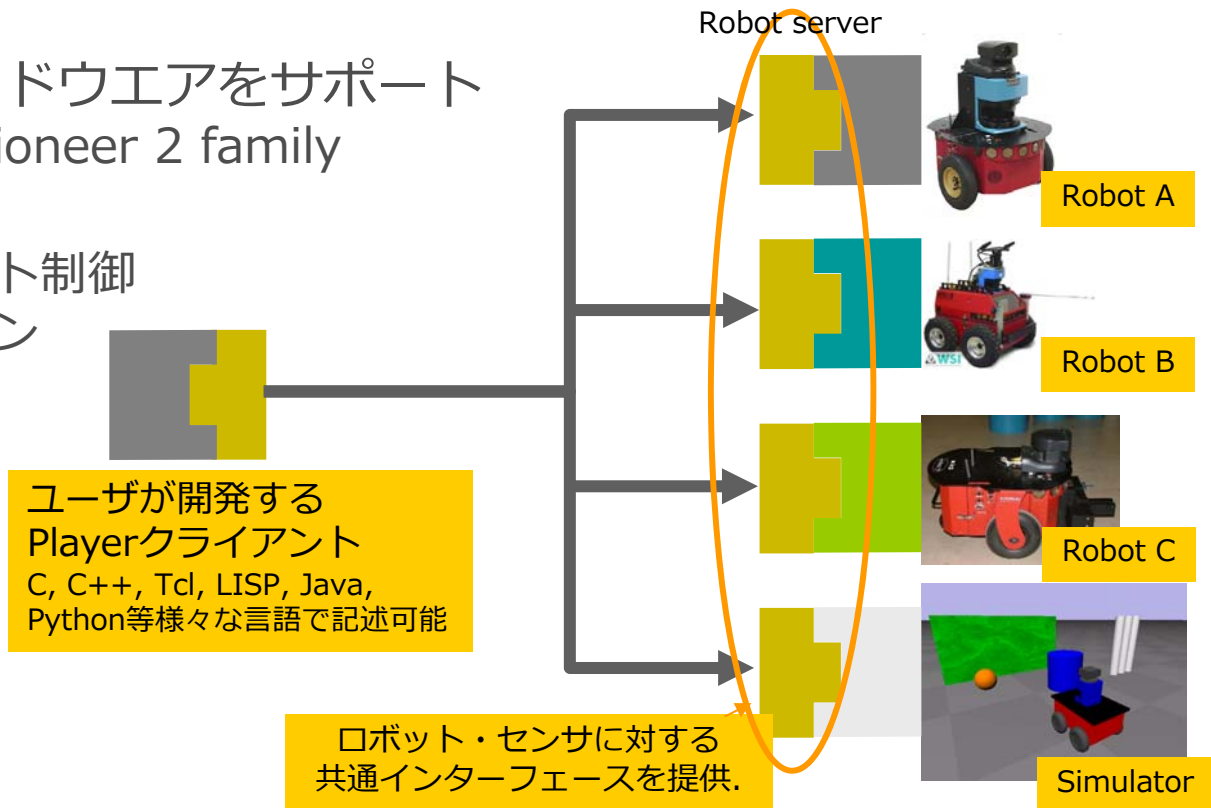
新日本電工他: Mobile SEM

Player/Stage and Gazebo

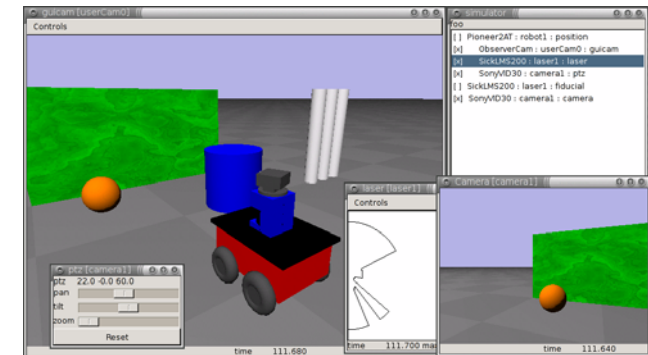
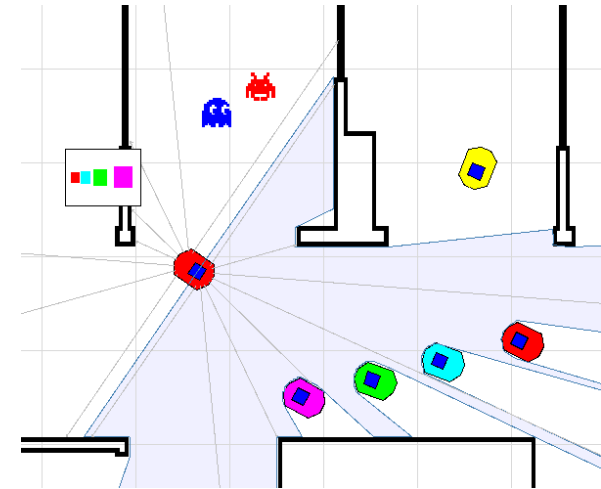
<http://playerstage.sourceforge.net/>



- 最も古い研究用ロボットミドルウェア/ロボットプラットフォームの一つ
- USC Robotics Labで開発
- Player
 - ロボット用ネットワークサーバ
 - 主に移動ロボットへの共通インターフェースを提供
 - 多くのロボットハードウェアをサポート
 - ex. ActivMedia Pioneer 2 family
 - 研究用途
 - マルチエージェント制御
 - センサフュージョン
 - パスプランニング
 - 衝突回避
 - etc...



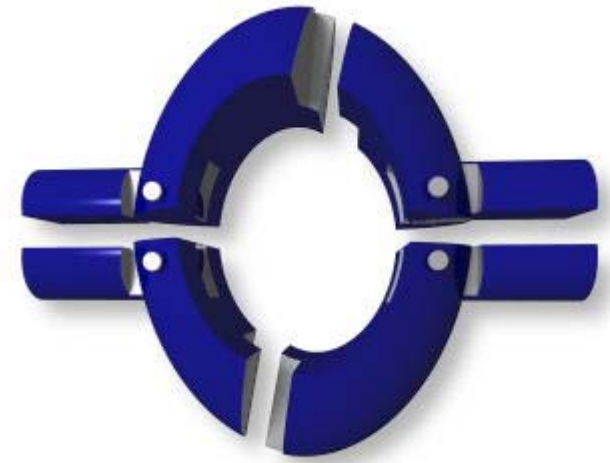
- Stage
 - 2次元ビットマップベースのシミュレータ
 - 移動ロボット、センサ、オブジェクト群をシミュレート
 - マルチエージェントシステムの研究のために設計されている。
- Gazebo
 - 屋外環境のマルチロボットシミュレーション環境を提供。
 - 3次元環境のロボット、センサ、オブジェクト群のシミュレーション環境を提供する。
 - 後述のROSの動力学シミュレータとして今も利用されている
 - 現在はDARPAの支援を受けてOSRF[†]において開発を継続中



[†] OSRF: OpenSource Robotics Foundation

“Stage” は2次元、“Gazebo”は3次元の移動ロボットシミュレーション環境を提供

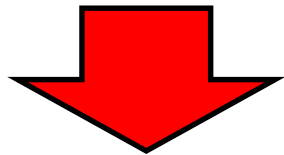
OROCOS



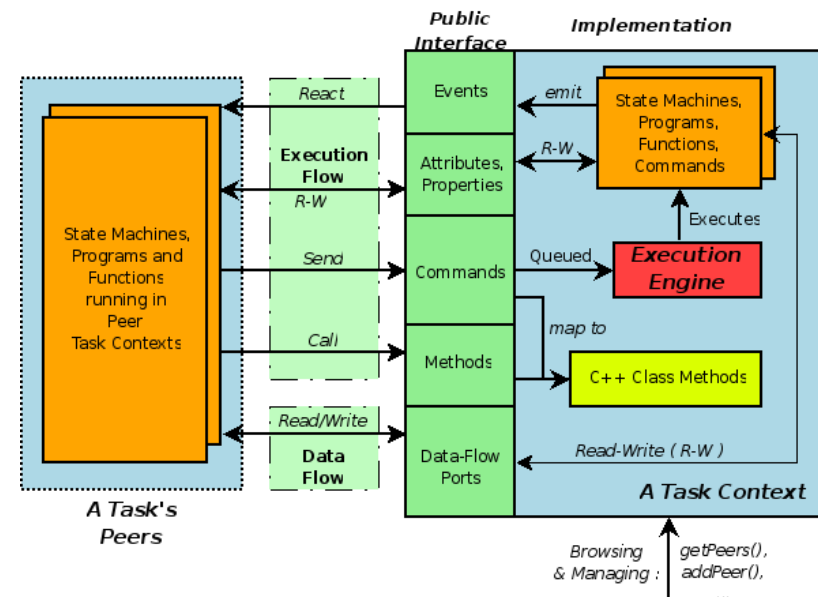
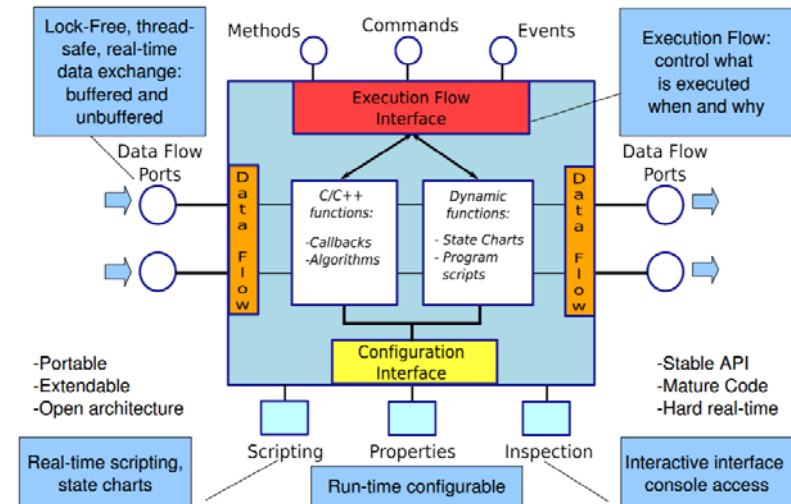
<http://www.oroocos.org/>

- EUプロジェクトで開発(2001-2003)
 - K.U.Leuven(ベルギー)
 - LAAS Toulouse(フランス)→ORCAへ
 - KTH Stockholm(スウェーデン)
- ハードリアルタイムのソフトウェアフレームワーク
- ロボットの制御に必要なライブラリ集（運動学、リアルタイム制御、etc…）
- 最近はコンポーネントベース開発のフレームワークも提供
- ツールによるモデルベース開発

- コンポーネントモデル
 - データフローポート
 - 種々のサービスインターフェース
 - コンフィギュレーション機能
 - コールバックベースのロジック実行フレームワーク
- リアルタイム実行の枠組み
 - モジュール間の密結合
 - ROS（後述）と連携してリアルタイム実行の枠組みとして今でも利用されている

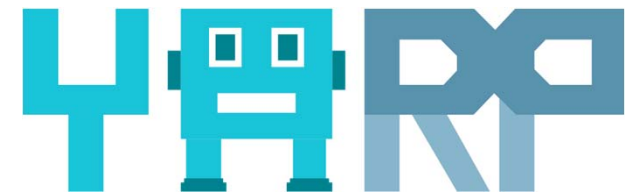


コンポーネントモデルはほぼ
RTCと同じ

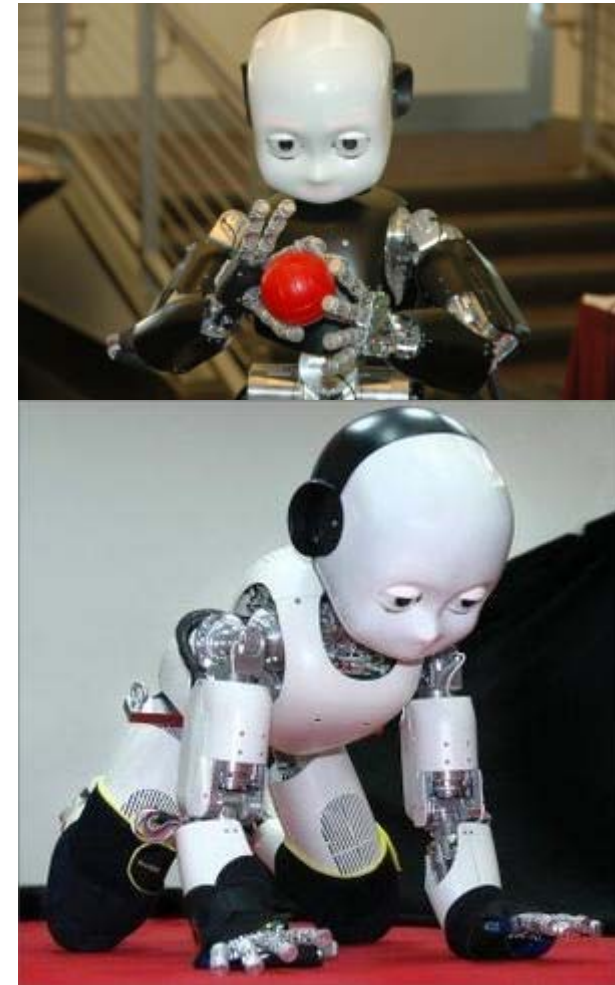


YARP

<https://www.yarp.it/>



- IIT (Istituto Italiano di Tecnologia) で開発されたiCubのためのソフトウェアプラットフォーム
 - iCub: EUプロジェクトRobotCub,
 - 53DOFの赤ちゃんの様なヒューマノイド
- コンポーネントフレームワークは無し
 - Mainから書き始める
 - 原則1プロセス1モジュール
- 多様な伝送方式のデータポートを提供
 - TCP, UDP, multicast
 - Carrier: 様々なマーシャリング、プロトコルを利用可能
- 簡単なRPCもある
- 独自のマーシャリング方式
- ノード間の利用にはネームサービスを利用
- CUIツール: yarp
 - 接続制御、モジュール制御



Microsoft Robotics Studio

<http://www.microsoft.com/robotics/>

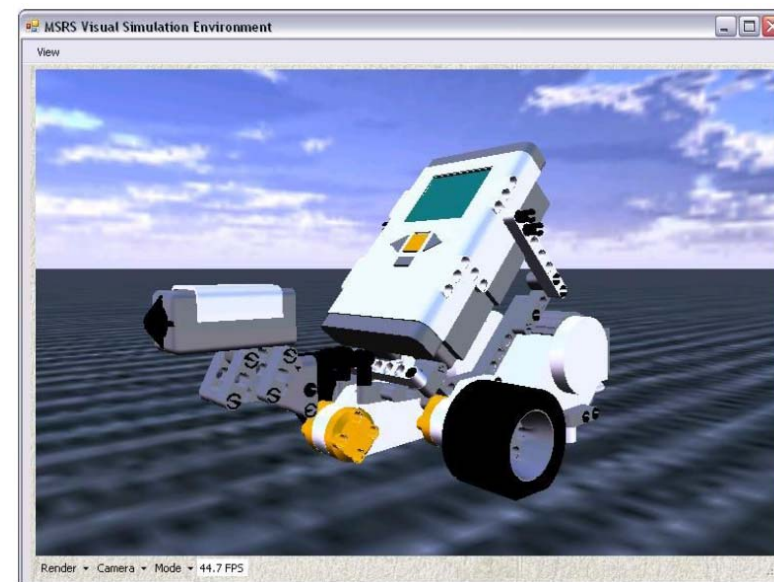
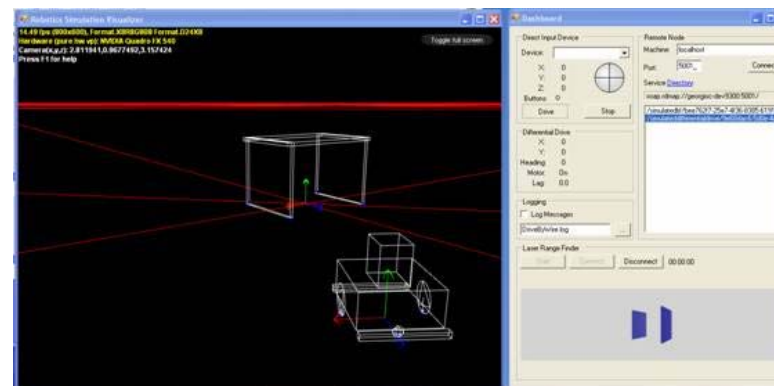


Microsoft®
Robotics Developer Studio 4

- **すでに終了している**
- 高レベルのビヘイビアやハードウェアのサービス志向の抽象化を提供
- 再利用性
- シミュレータエンジンを提供
 - AGEIA PhysXも使える
- ブラウザインターフェース
- Windowsのみで利用可能
 - ただし、ある種のハードウェアに対応
 - LEGO NXT, i-Robot, ActivMedia等

DSSP

- DSSP : Decentralized Software Services Protocol
- SOAPベースのプロトコル
- アプリケーションはサービスの集合体として定義される。
- サービスとは：アプリケーションのライフタイム中であれば、生成、操作、モニタリング、削除可能な軽量なエンティティ。



OPEN-R

(旧AIBOのミドルウェア)



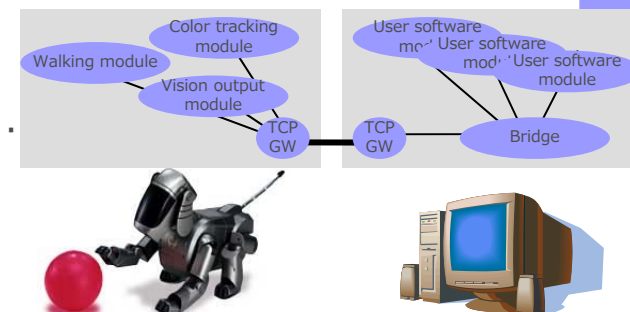
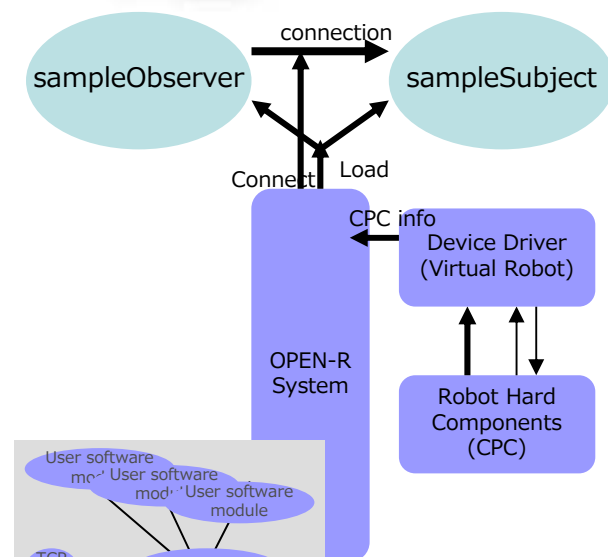
- すでに終了している
- 開発元：SONY
- AIBO, SDR-[3,4]X, QURIO等の制御アーキテクチャ

ハードウェアコンポーネント

- CPC (Configurable Physical Component)
 - 各コンポーネントは自身の機能、位置、ダイナミクスパラメータなどの情報を内蔵している。

ソフトウェアコンポーネント

- コンポーネント指向
 - スケーラビリティ
 - ポータビリティ
 - 開発環境とロボット間
 - インターオペラビリティ
 - 異なるタイプのロボットに対して同じインターフェースを提供
 - スタイル・フレキシビリティ
 - 車輪型、4足歩行型、2足歩行型...
 - コンカレント開発



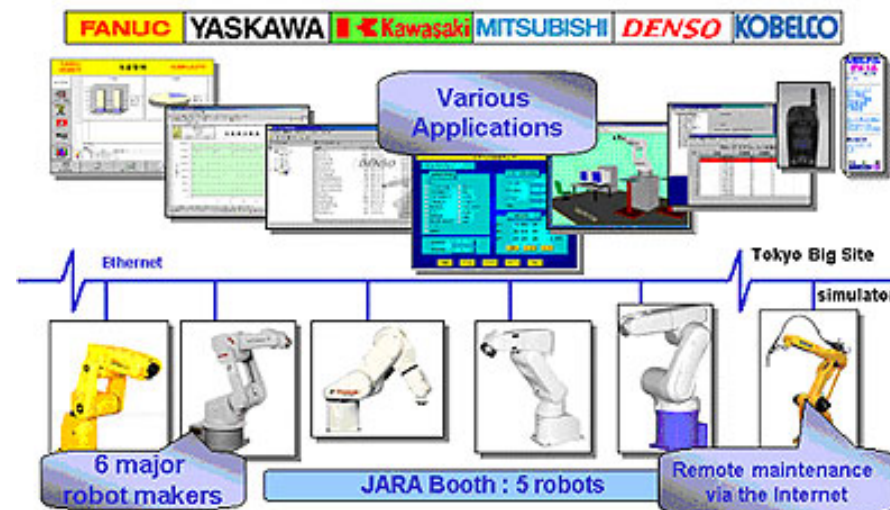
ORiN

(産業用ロボット向けミドルウェア)

<http://www.orin.jp/>

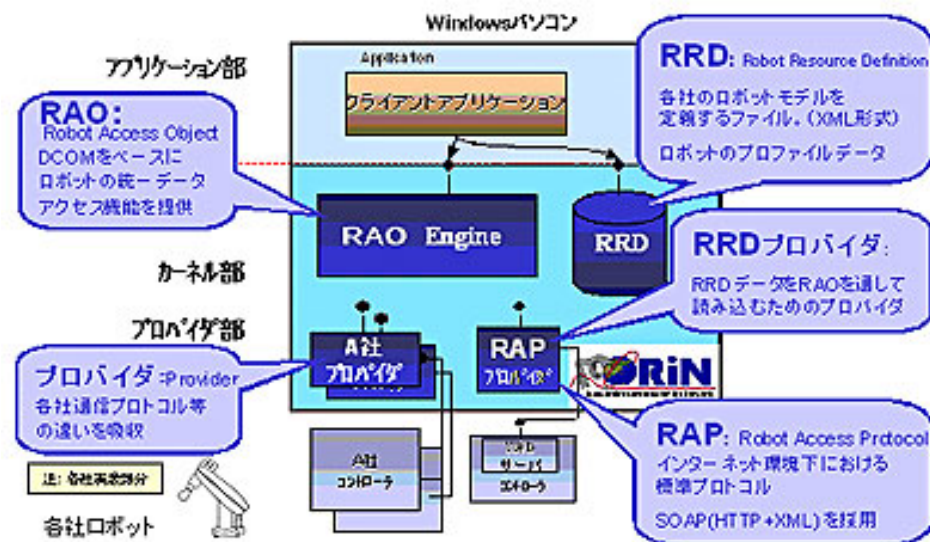


- 日本のFAロボット標準
 - FAロボットコントローラを抽象化
 - マルチベンダロボットシステムを容易に実現可能
 - メンバー (ORiN コンソーシアム)
 - FANUC, YASUKAWA, Kawasaki, MITSUBISHI, DENSO, KOBELCO)



- ポリシー

- 拡張性
 - ベンダ特有のオプションを定義可能
- ネットワークプロトコル
 - DCOM (言語非依存)
- 環境 : PC&Windows



RSNP (Robot Service Network Protocol) (ロボット用クラウドサービスフレームワーク)

<http://robotservices.org/>

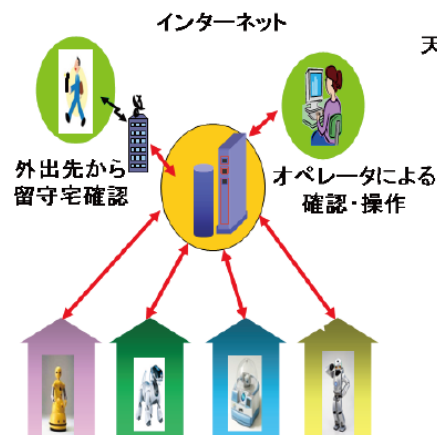
<http://www.robotservices.org/wiki/jp/>



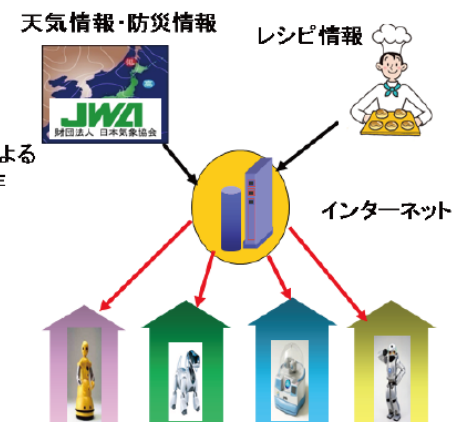
- RSi(ロボットサービスイニシアチブ) が主導
 - 2004年発足
 - ロボットをプラットフォームとしたインターネットと連携した新たなビジネス創出を目指す
- RSiサーバ等で提供するサービスを各地のロボットを介して提供
 - 天気予報、見守り、ロボットマップ、各種ロボット制御
- SOAPを利用
- 疑似Push機能を利用しFirewall越し通信を実現
- 主としてロボットとインターネット(クラウド)との連携に利用
 - cf. RTC



例1 遠隔操作によるサービス
(見守りサービス等)



例2 情報提供サービス

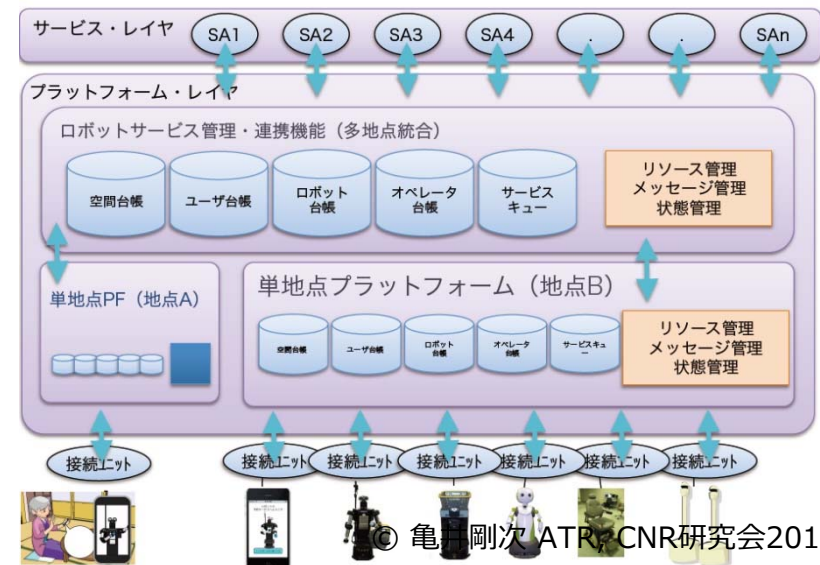


UNR Platform (Ubiquitous Network Robot Platform) (ロボットサービスフレームワーク)

<http://www.irc.atr.jp/std/UNR-Platform.html>

 **ATR** Advanced
elecommunications
research Institute International

- ロボット、スマートフォンアプリ、環境センサがネットワークを介して連携し、多地点で人々にサービスを提供することを目指す
 - ARTにより開発、配布
 - 一部はOMG RoIS (Robot Interaction Service)標準に準拠
- 個々のロボット仕様を気にせずアプリケーションを記述可能
- アプリと下位コンポーネントのデータのやり取りを仲介
- RoIS: ロボット対話サービスに必要な機能を標準化
- UNRプラットフォームは各種RoISサービスをクライアントの要求に応じて仲介
- ロボット側はRTMやROSなど何を利用していてもよい



ROS(Robot Operating System)



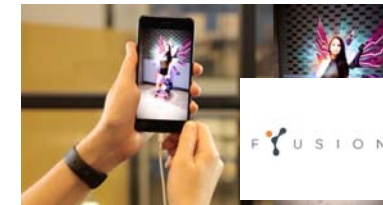
<http://wiki.ros.org/>



- Willow Garage
 - 2007年設立のロボットベンチャー（米、Menlo Park）
 - 2014年事業停止
 - Scott Hassanが出資
 - googleの初期エンジンの作者の一人
 - ビジネスモデル
 - ソフト：ROS（無償）＋ハード：PR2 を販売
 - PR2を10台無償で大学などに配布
 - スピンアウト創出を狙う



Industrial Perception, Inc.



Open Source Robotics Foundation

概要：

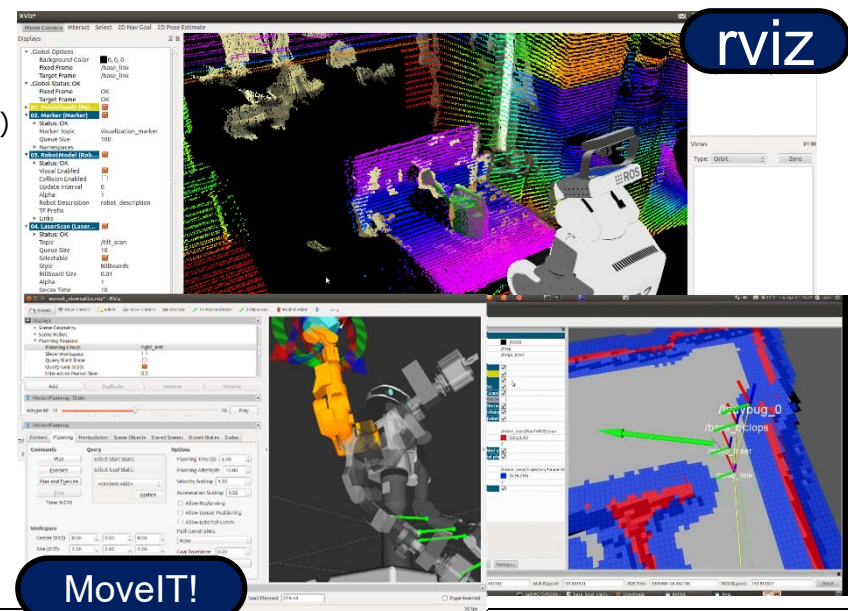
- 米国ベンチャー Willow Garageが開発したロボット用OS。
- 2007年から開発。
- オープンソースとして広くソースコードを公開。
- 現在はOSRF(オープンソースロボティクス財団, 現在はOpen Roboticsに改名) が管理。
- ロボット機能要素（センサ、モータ、アーム、移動機能）をモジュール化。個別に動くプログラムを連携する通信部分を提供。（RTMとコンセプトは同じだが、リアルタイム・密結合機能はない）
- Linuxのソフトウェア管理機構を活用し（Linuxに慣れた人なら）インストールが容易。

ユーザ：

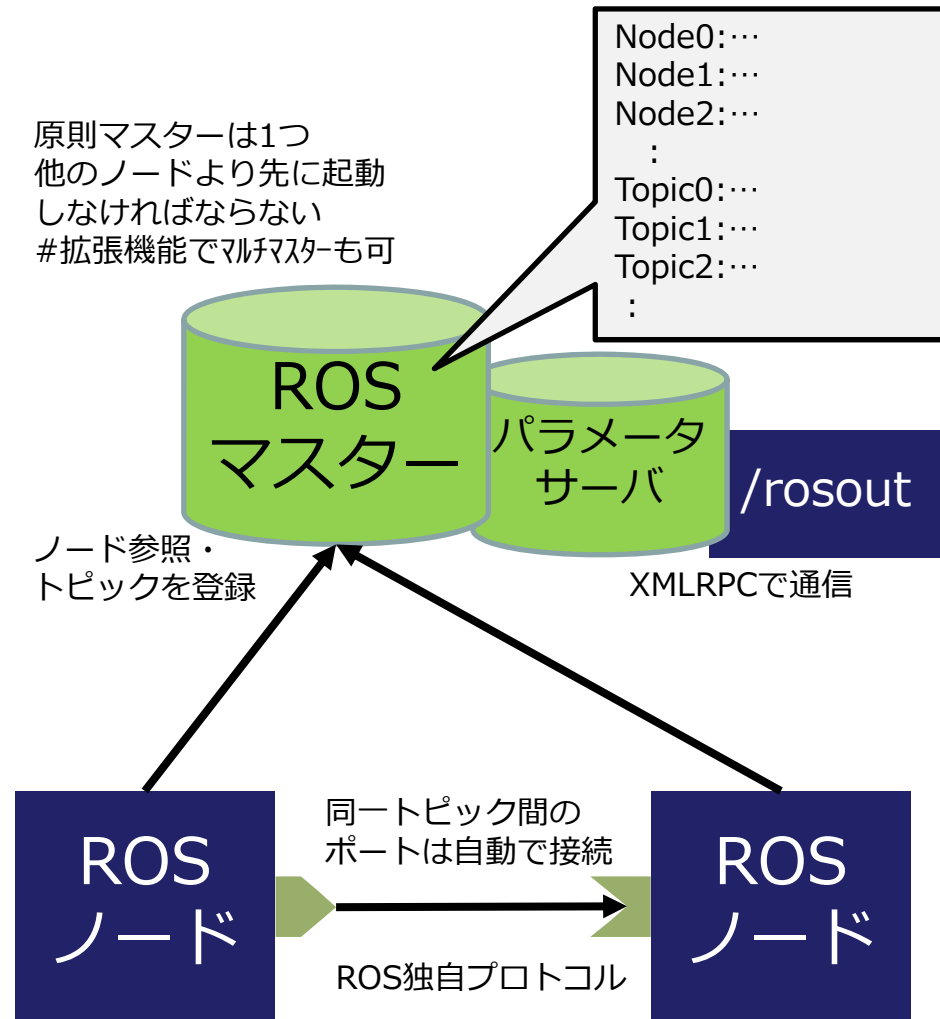
- Savioke、Fetchロボティクス、Clearpath Robotics、トヨタなどがロボット用OSとして採用。
 - 学術分野ではデファクトスタンダードとなっている

特徴：

- Ubuntu Linux + 上で動作（コマンド入力による操作が基本）
 - 他のOSは公式にはサポートしていない
- キラーアプリケーションにより人気獲得
 - rviz: ロボットの様々な状態を3Dで表示
 - MoveIT!: アームの軌道計画
 - Navigation Stack: 地図作成・経路計画
- 現在次バージョン：ROS2に移行中
 - RTM同様の密結合・リアルタイム機能含む
 - 通信はOMGのDDS標準を利用（RTMと互換性有）



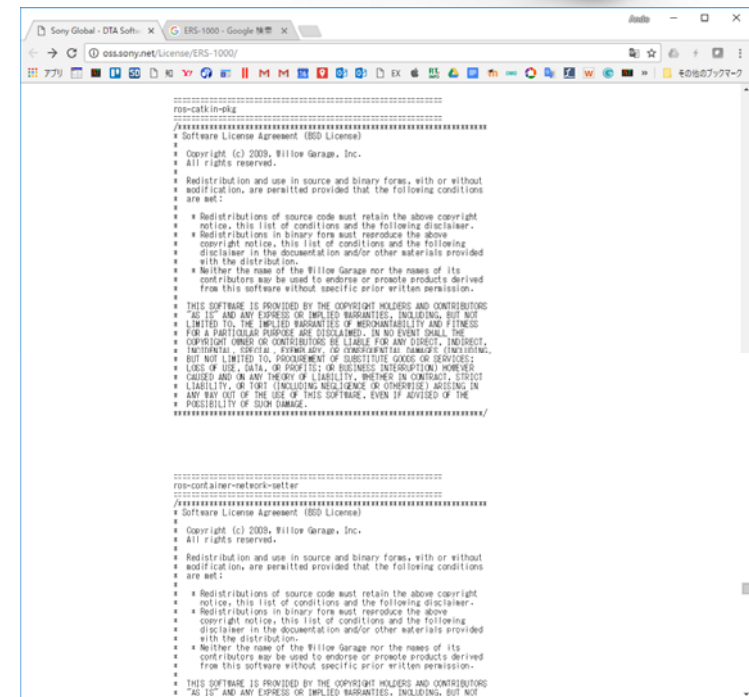
- **ノード**：ROSのモジュール化単位
 - 1ノード=1プロセス
- **マスター**：ノードの参照やトピックを保持するネームサービス
 - システム全体で原則一つ→SPOF (Single Point of Failure)
 - 他のノードより先に起動しなければならない
- **パラメータサーバ**：ノードのパラメータを保持するデータベース
 - マスター内で動作、ノードからはXMLRPCでアクセス
- **rosout**：ノードに対してstdout, stderrのような役割を果たす



- aibo
 - 新AiboではOpen-RではなくROSを利用している模様
 - ROS kinetic (バージョン) を利用
 - 約500以上のオープンソースソフトウェア (ROS含む) を利用している
 - ライセンスをWebサイト上で明記



- ROSに対して手を加えている
 - 通信効率化の追加モジュール開発
- SDK (ソフトウェア開発キット)
 - 将来的にユーザに提供
 - ROSベースSDKの予定



<http://oss.sony.net/License/ERS-1000/>

ROS

- UNIXユーザに受けるツール（特にCUI）が豊富
 - →基本はLinuxのみサポート
- 独自のパッケージ管理システムを持つ
- 実装方法が比較的自由
 - コア部分の使用が固まっていない
 - コンポーネントモデルがない
- ノード（コンポーネント）の質、量ともに十分
 - WGが直接品質を管理しているノードが多数
- ユーザ数が多い
- メーリングリストなどの議論がオープンで活発
- 英語のドキュメントが豊富

RTM

- 対応OS・言語の種類が多い
 - Windows、UNIX、uITRON、T-Kernel、VxWorks、QNX
 - Windowsでネイティブ動作する
- 日本製
 - 国外のユーザが少ない
 - 英語ドキュメントが少ない
- GUIツールがあるので初心者向き
- 仕様が標準化されている
 - OMGに参加すればだれでも変更可
 - サードパーティー実装が作りやすい
 - すでに10程度の実装あり
- コンポーネントモデルが明確
 - オブジェクト指向、UML・SysMLとの相性が良い
 - モデルベース開発
- IEC61508機能安全認証取得
 - RTMSafety

第三者による比較：<http://ysuga.net/?p=146>

【ユースケースの変化】

- 複数のロボット
- 組込みCPU
- リアルタイム
- 理想的でない通信環境
- 製品向け使用
- あらかじめ規定されたパターンにのっとった構造化したシステム構成



【ROS1では】

- 単体のロボット
- 強力なCPU
- 非リアルタイム
- 途切れない通信環境
- 研究向け
- 自由な枠組み（main関数）



【新たな技術】

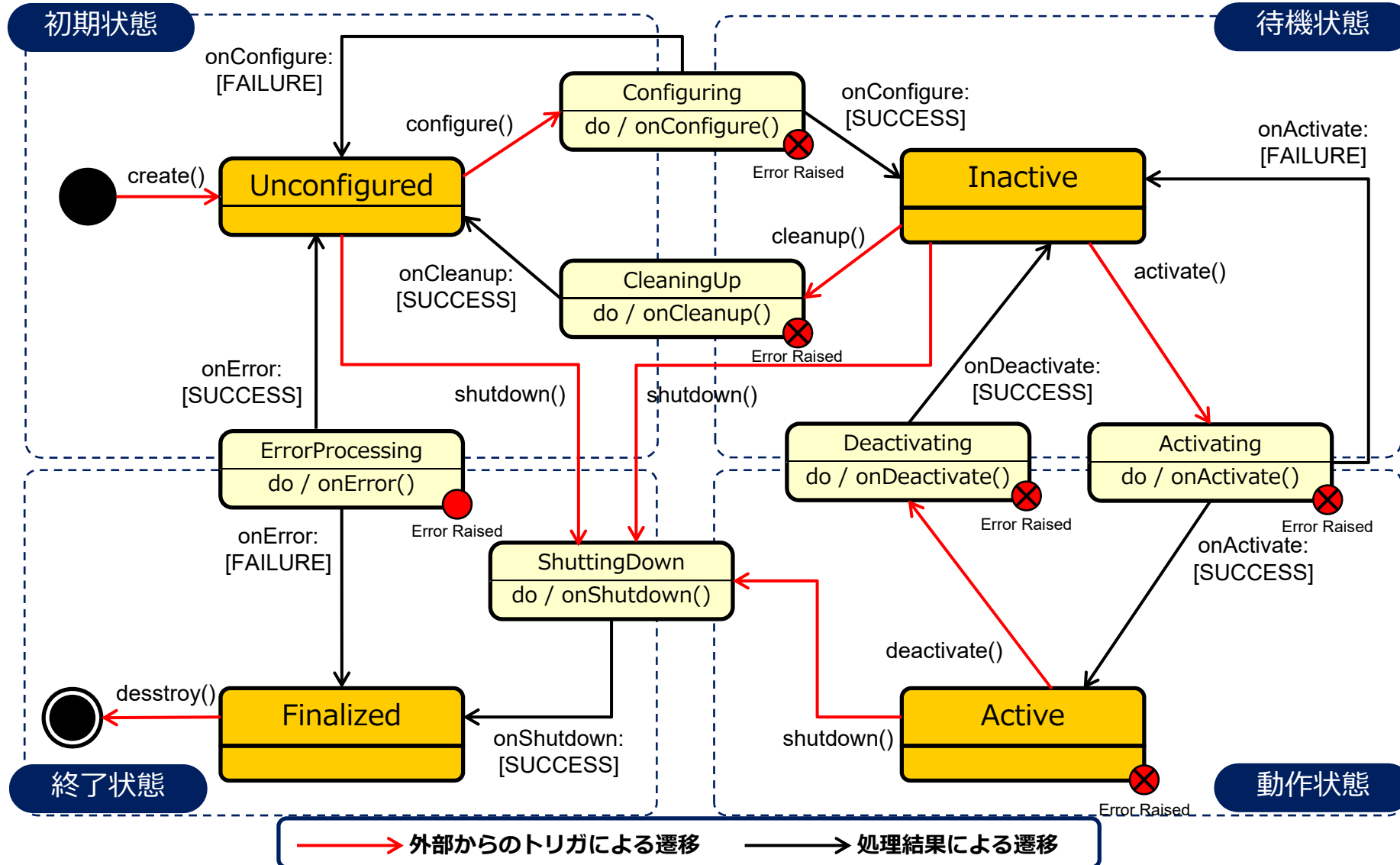
- Zeroconf (avahi, bonjour, UPnP等)
- Protocol Buffers
- ZeroMQ (and the other MQs)
- Redis(次世代高速key-valueデータストア)
- WebSockets
- **DDS (Data Distribution Service).**

ROSを大幅に改良したROS2への移行を発表
(互換性なし)

http://design.ros2.org/articles/why_ros2.html

- 通信：DDS(OMG標準のpub/sub通信ミドルウェア)
 - ミドルウェア層を作製し複数のDDS実装（製品も含む）を使えるように
 - 単一障害点がない（ROS1ではmasterが落ちると×）
 - QoS制御が可能に(History, Depth, Reliability, Durability)
- コンポーネントモデルを導入
- 複数のOSに対応（Windows、MacOS）
- ROS1とはブリッジで通信
 - 直接は通信できない、互換性なし
- セキュリティ対応（DDSセキュリティを利用）
- リアルタイム実行可能（Linuxのみ）
- 組込み対応

http://design.ros2.org/articles/ros_middleware_interface.html

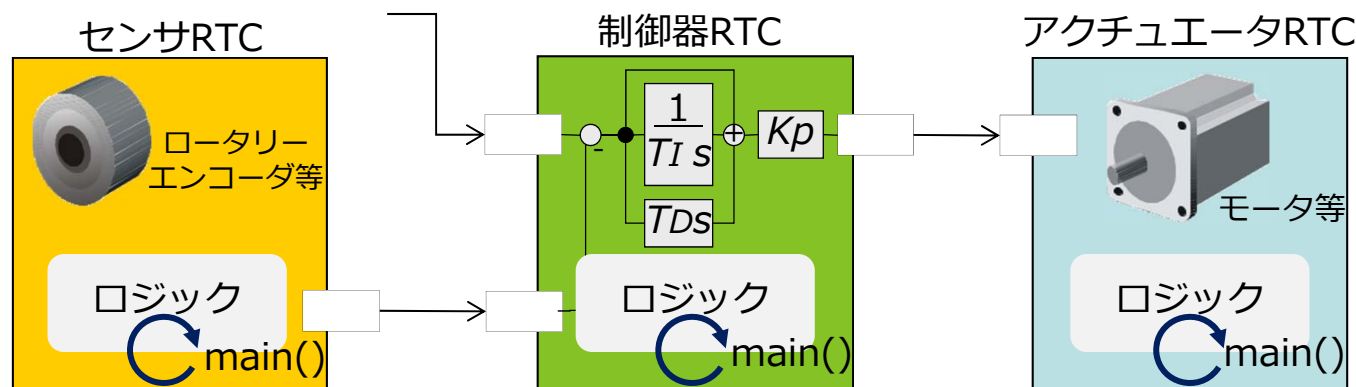


過渡状態を状態とみなしている以外はRTCをほぼ同じ
エラー処理が明確に、終了処理が一体になっている

ROS1

モジュール
(ノード)
の粒度

大

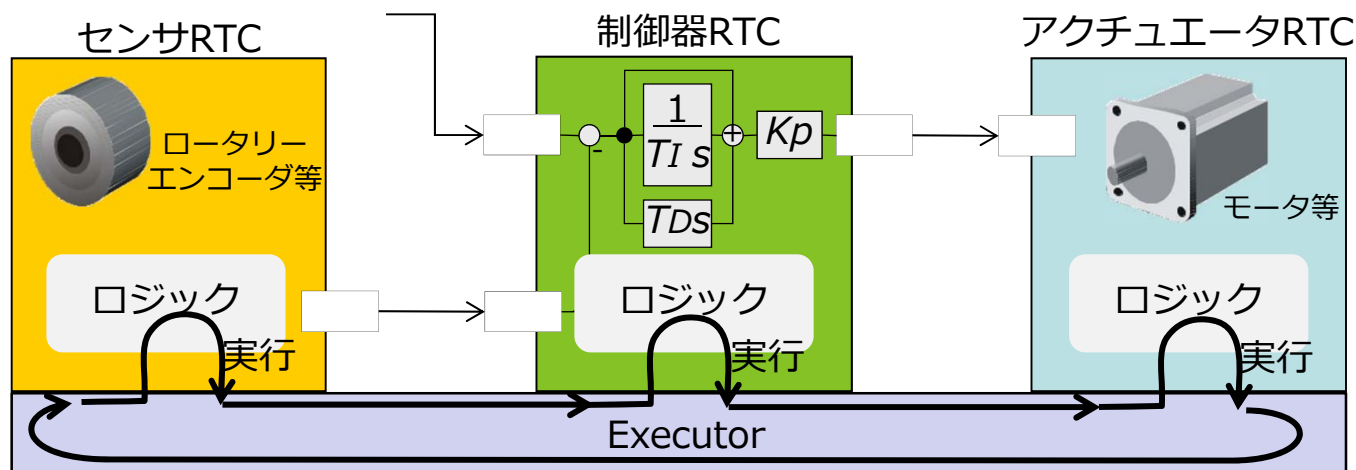


ノードの実行タイミング・順序は制御できない
複数のノードを密結合してリアルタイムシステムを構成できない

ROS2

モジュール
(ノード)
の粒度

小



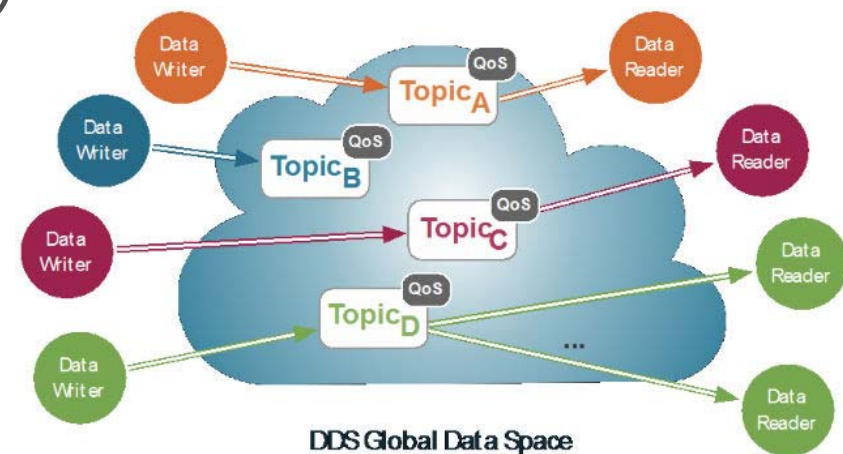
ノードの実行タイミング・順序をExecutorで決定できる
複数のノードを密結合してリアルタイムシステム化可能

- Alpha1 (2015.8)
 - DDS, 基本的なpub/sub messagingとservice 対応
 - ROS1とのブリッジ
- Alpha2 (2015.10)～Alpha8(2016.10)
- Beta1(2016.12)～Beta3(2017.9)
- Ardent Apalone (2017.12)
 - コードネーム：Ardent(熱烈) Apalone (アパロンカメ、北米原産のスッポン科のカメ)
 - ディスカバリ機能、ライフサイクル、コマンドラインツール等
- Bouncy Bolson (2018.6)
 - コードネーム：Bouncy (弾む) Bolson (メキシコゴファーガメ、Bolson tortoise)
 - Launchシステム、バイナリパッケージ提供
- Crystal Clemmys (2018.12)
 - コードネーム：Crystal(透明な) Clemmys (キボシイシガメ Clemmys guttata)
 - 周辺ツール (gazebo、rqt、rosviz) のサポート



ROS1同様にカメにまつわるコードネーム

- OMG(Object Management Group)で標準化されたpub-sub型データバスミドルウェア
- Pub/sub型通信
 - トピックが同じ送受信者間でデータが配信される仕組み
- SPOF（単一障害点）がない discoveryメカニズム
- OMGのミドルウェアTF(MARS)で最もアクティブに活動しているTFで策定
 - 関連標準仕様は10程度



| | Target | Open spec/source | Real- time | Language | OS | modularity | communica tion |
|--------------|------------------------|---------------------|---------------|--|---|-----------------|----------------------|
| OpenRTM-aist | Universal | ○/○ OMG RTC | ○ | C++, C, Python, Java, .NET, Android | UNIX, Mac OS X, Windows, uITRON, QNX, VxWorks | ○ CBSD | CORBA |
| ROS | Universal | △/○ | ○ | C++, Python, Java, LISP, Matlab | Linux, (OS X, Windows) | × Free style | original protocol |
| OROCOS | Universal | △/○ | ○ | C++, (scripting: Lua) | Linux, Windows, Etc.. | ○ CBSD | Ice, CORBA |
| OPRoS | Universal | ○/○ OMG RTC | ○ | C++ | Windows, Linux | ○ CBSD | Original protocol |
| YARP | Humanoid/ Universal | △/○ | × | C++, Java, Python, Lua, Matlab | Linux, Windows, Mac OS X | △ Free style | Original protocol |
| ORCA/ORCA2 | Universal | △/○ | ○ | C++, Python | RTLinux, Other | ○ CBSD | CORBA, CURD, Ice |
| MSRS | Universal | △/△ | × | .NET (C++, C#, VB, etc.) | Windows | △ SOA | DSS・SOAP |

| | Target | Open spec/source | Real- time | Language | OS | modularity | communica tion |
|--|---------------------|------------------------|---------------|--|---|------------|----------------------|
| OpenRTM-aist | Universal | ○/○ OMG RTC | ○ | C++, C, Python, Java, .NET, Android | UNIX, Mac OS X, Windows, uITRON, QNX, VxWorks | ○ CBSD | CORBA |
| ORiN | FA robots | ○/× ISO ORiN | × | C++ | Windows | △ OOP | DCOM, CORBA |
| RSNP | Internet Service | ○/× RSi RSNP | × | Java | Java VM | △ OOP | SOAP |
| UNR Platform | Internet Service | ○/○ OMG RoIS/RLS | × | Java | Java VM | △ OOP | SOAP |
| PlayerStage | Mobile robot | △/○ | ○ | C, C++, Tcl, LISP, Java, and Python | Linux, Etc.. | △ PO | original protocol |
| OPEN-R | AIBO, SDR3X | ○/× | × | C++ | Linux | △ OOP | original protocol |
| Open Robot Controller Architecture | Universal | △/× | × | Java, Python | Linux, Etc.. | △ OOP | HORB |

- ロボット新戦略とロボットOS・ミドルウェア
- RTミドルウェアの目的、アーキテクチャ、応用
- その他のロボットOS/ミドルウェアの動向

詳しくは…

検索

Github Noriaki Ando



本日の資料はgithubに掲載します。
https://n-ando.github.io/titech_robotics

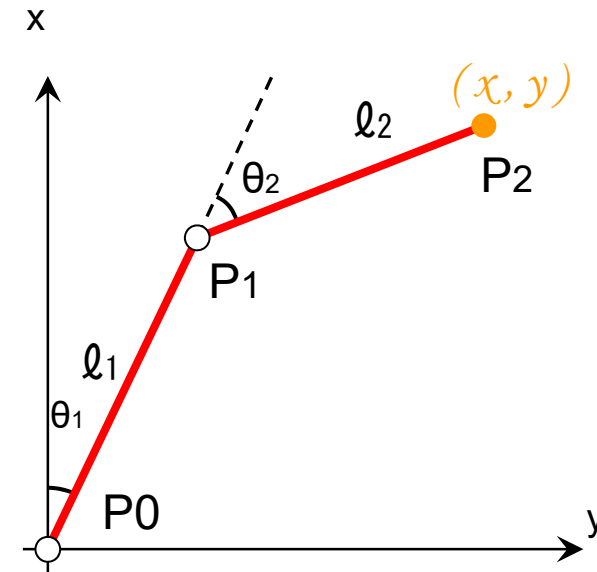
課題1：順運動学（2自由度）

～関節角度から手先位置を求める～

- 右の2自由度アームの順運動学を求めよ。

点 P_2 (x, y)

の値を $l_1, l_2, \theta_1, \theta_2$ で表す。



$$x_2 = l_2 \times \cos(\theta_1 + \theta_2) \quad x =$$

$$y_2 = l_2 \times \sin(\theta_1 + \theta_2) \quad y =$$

課題2：逆運動学（2自由度）

～手先位置から関節角度を求める～

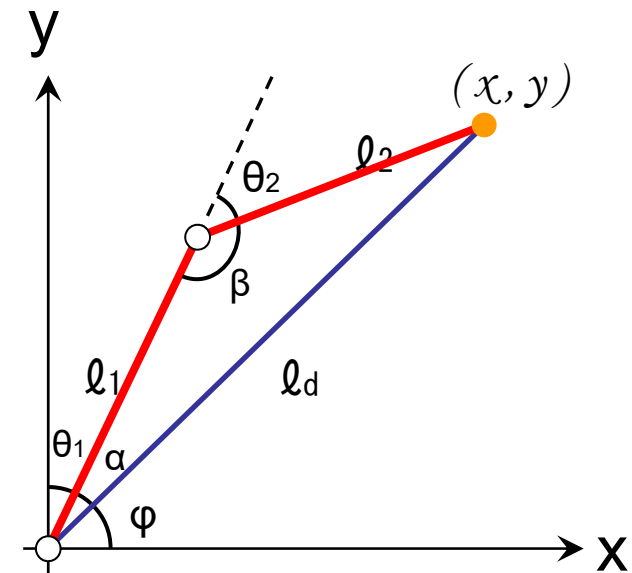
- 右のアームの逆運動学を求めよ。

ヒント： θ_1 , θ_2 は以下の式で表される。

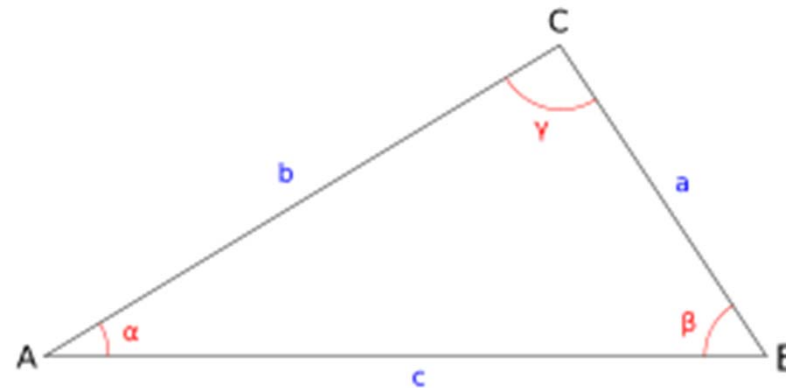
$$\theta_1 = \frac{\pi}{2} - \alpha - \varphi$$

$$\theta_2 = \pi - \beta$$

α と β と φ を l_1 , l_2 , l_d で表現する。



余弦定理



余弦定理：
三角形の角と辺の関係式

$$\cos \alpha = \frac{b^2 + c^2 - a^2}{2bc}$$

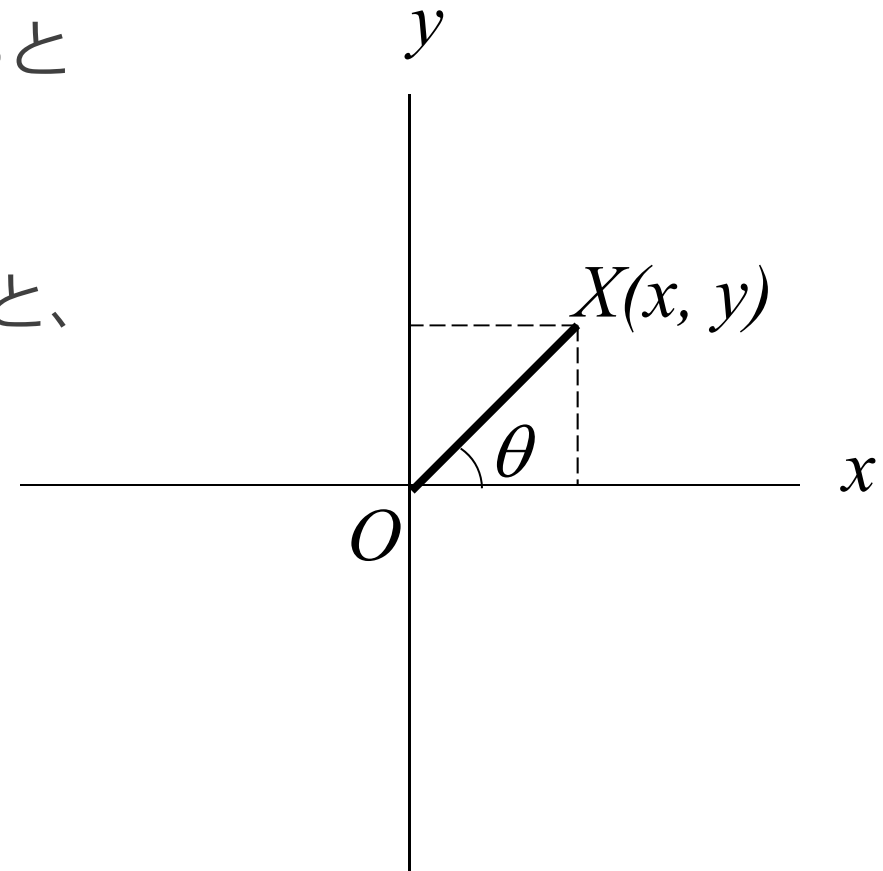
逆関数

OX が x 軸となす角度を θ とすると

$$\tan \theta = \frac{y}{x}$$

なので、逆関数 \arctan を使うと、

$$\theta = \arctan \frac{y}{x}$$



$\sin \theta = x$ のとき $\arcsin x = \theta$

$\cos \theta = x$ のとき $\arccos x = \theta$

$\tan \theta = x$ のとき $\arctan x = \theta$

レポート課題（１）

1. ロボット制御に必要な以下のプログラムを示せ

2自由度のアームの逆運動学を計算する以下の仕様の関数のPythonプログラムを作成し、次のプログラムを完成させ、実行結果を示せ（40点）

関数: `th = invkinem(link, pos)` を作成

- `th`: 2つの関節の角度[deg]
- `link`: 2つのリンク長[m]
- `pos`: 手先位置[m]

引数、戻り値はいずれも要素数2の配列とする

Pythonで利用できる定数・関数

- `math.pi`: 円周率
- `math.sqrt(x)`: `x`の平方根
- `math.acos(x)`: `cos` の逆関数
- `math.atan(y, x)`: `tan` の逆関数

```
import math
def invkinem(link, pos):
    l1 = link[0]
    l2 = link[1]
    x = pos[0]
    y = pos[1]
    ld =
    b =
    a =
    phi =
    th = [0] * 2
    th[0] =
    th[1] =

    return th
link = (1.0, 1.0)
path = ((-1.0, 1.0), (-0.5, 1.0), (0.0, 1.0), (0.5, 1.0), (1.0, 1.0))
for pos in path:
    print invkinem(link, pos)
```

レポート課題（２）

2. ミドルウェアを利用したサンプルプログラムを示せ（40点）
 - a. ロボットミドルウェアを一つ選び、データの送信を行う手順・方法を調べ説明せよ。結果として、コメントを付したソースコード（完全である必要はないが、データ送信に必要な最低限の部分を示すこと。例えばRTMであればonExecute関数部分。）を添付せよ。
 - b. 同様に、データの受信を行う手順・方法を調べ説明せよ。結果として、コメントを付したソースコード（完全である必要はないが、データ受信に必要な最低限の部分を示すこと。例えばRTMであればonExecute関数部分。）を添付せよ。

Webページにはヒント掲載
コードに付記されたコメントを重視します。

3. 授業の感想（20点）

サービス課題。よく忘れる人がいます！！

解答

- レポート提出期限後に資料掲載サイトに解答を掲載します。

https://n-ando.github.io/titech_robotics