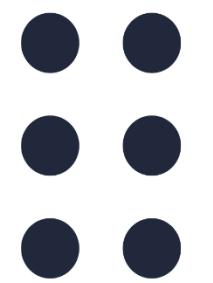


## 18コマ目

ROS(Robot Operating System)の紹介,  
今後のロボット制御について



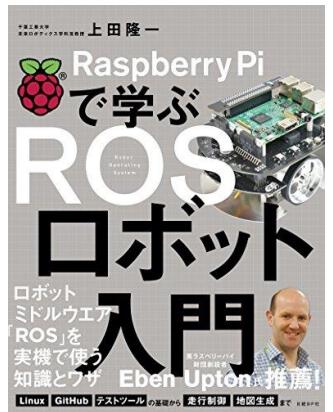
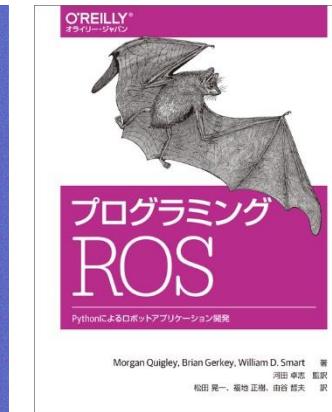
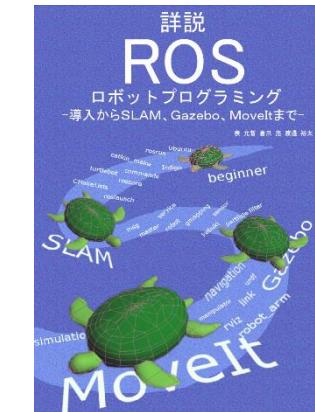
# ROSの概要、紹介

 ROS



# 参考資料

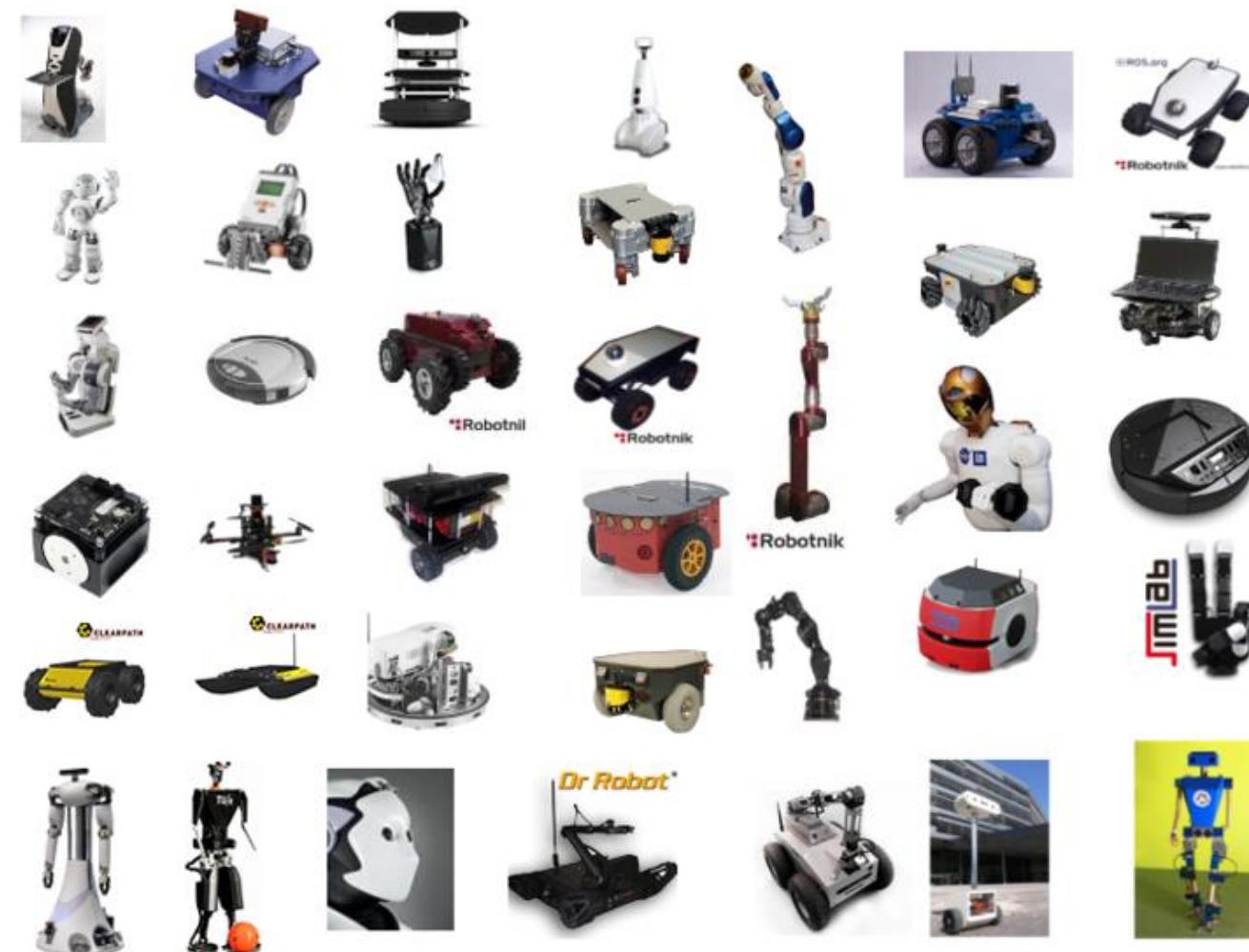
- 詳説 ROSロボットプログラミング
  - [http://irvs.github.io/rosbook\\_jp/](http://irvs.github.io/rosbook_jp/)
  - 著者：表允暫, 倉爪亮, 渡邊裕太
- プログラミングROS
  - 著者：Morgan Quigley, Brian Gerkey, William D. Smart  
監訳：河田卓志, 訳：松田晃一, 福地正樹, 由谷哲夫
- Raspberry Piで学ぶ ROSロボット入門
  - 著者：上田隆一



# ROS(Robot Operating System)

- ロボットソフトウェア開発向けのフレームワーク
  - ツールやライブラリ、規約を集めたもの
  - 多種多様なロボットプラットフォームに対して、複雑でロバストなロボットの動作の開発を簡単に行うことが可能
  - 大規模な協力体制を簡単に実現するための特徴を持つ
- ロバストで汎用的なロボットのソフトウェアを開発することが困難
  - 現実世界の複雑なタスクや多様な環境条件を取り扱うため

# ROSを導入したロボット例



# ROSを使用可能なセンサ例



# ROSの歴史

- 2000年代中頃：スタンフォード大学内のプロジェクト
  - スタンフォード人工知能ロボット (STAIR : STanford AI Robot) や  
パーソナルロボット (PR : Personal Robot) プログラムなどの柔軟でダイナミックなソフトウェアシステムの初期のプロトタイプ
- 2007年：Willow Garage社
  - 上記のプロジェクトを拡張したソフトウェアを開発  
このソフトウェアは、制約が緩いBSDオープンソースライセンスでオープンに開発され、ロボット研究コミュニティの中で徐々に使用
- **現在：オープンソースによるエコシステム**
  - 世界中の数万人のユーザで構成され、机上の趣味のプロジェクトから大規模な工業用の自動化システムまでの幅広い分野で利用

# ROSの哲学

- いくつかの重要な点でUnixにおけるソフトウェア開発の哲学に従う
1. P2P (ピアツーピア)
    - 互いに接続された小さなプログラムで構成され，中央にルーティングサービスは存在しないが，互いにメッセージを送り合う
    - データの量が多くなった場合にも，より対応できるシステム
  2. ツールベース
    - 小さな汎用的なプログラムを組み合わせることで，複雑なソフトウェアを構成
    - 標準的な統合開発環境や実行環境を持たない
      - ソースコードツリーのナビゲーション，システムの相互接続状況の可視化，データストリームのグラフィカルな表示，ドキュメント生成，データのロギングなどはすべて独立したプログラムで実現

# ROSの哲学

## 3. 多言語対応

- ROSのソフトウェアモジュール、クライアントライブラリはどんなプログラム言語でも記述可能

## 4. 薄く実装

- 最初にスタンドアローンのモジュールを作成し、その後、他のROSモジュールとのメッセージを送受信を可能にするラップレイヤを設けることを推奨
  - 他のアプリケーションのために作られたROS以外のソフトウェアを再利用が可能
  - 標準の継続的インテグレーションツールによる自動化テストの作成が簡単

## 5. 無償かつオープンソース

- BSDライセンスのもとに公開されており、商用と非商用の利用が可能

# ROSのインストール

- LinuxOSの1つであるUbuntuが最適な実行環境
- Ubuntuのバージョンに合わせた、推奨ROSバージョンが存在
  - Ubuntu 14.04 LTSの場合 : ROS Indigo Igloo
  - Ubuntu 16.04 LTSの場合 : ROS Kinetic Kame
    - 上記の2種類が良く使用されている環境である
- この講習会ではRaspbianにROSのソースコードからビルドをする方法を紹介

# ROSの基本

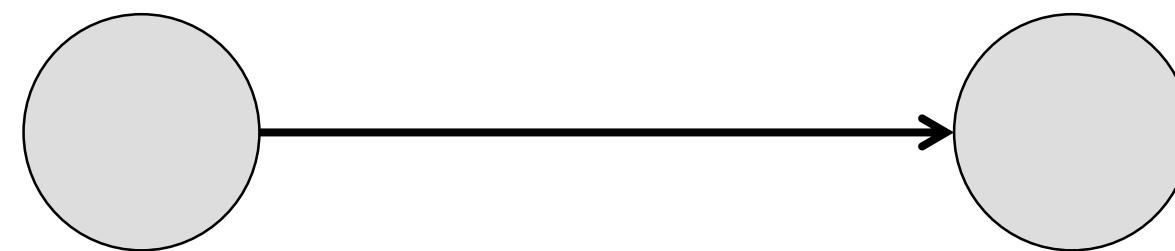


# ROSノードとマスター

- ROSノード
  - ROSの最小の構成要素であり，個々の実行プログラム
  - ノードを互いに接続することで，情報のやりとり（メッセージ通信）を行う
- ROSマスター
  - ノード間の接続とメッセージ通信の実現に必要なサーバ（ネームサーバ）
  - マスターが起動すると，マスターは各ノードの名前を登録し，ノード間で必要な情報を送受信する
  - マスターが起動していなければ，ノード間の接続やメッセージ通信を行うことができない

# メッセージ通信

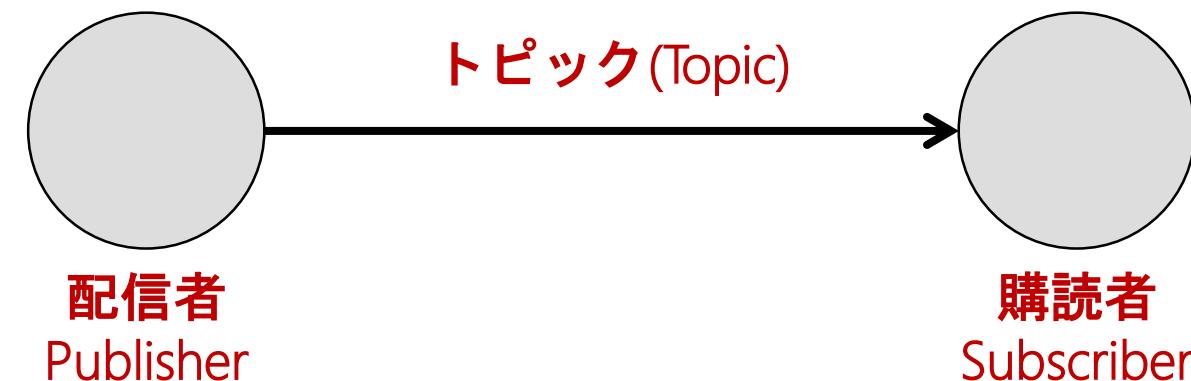
- ・作業目的に応じて細分化されたノードは、他のノードと処理結果を送受信（メッセージ通信）することで、一つの大きなプログラムになる



- ・ノード間のメッセージ通信には2種類ある
  1. トピックメッセージ通信
  2. サービスメッセージ通信

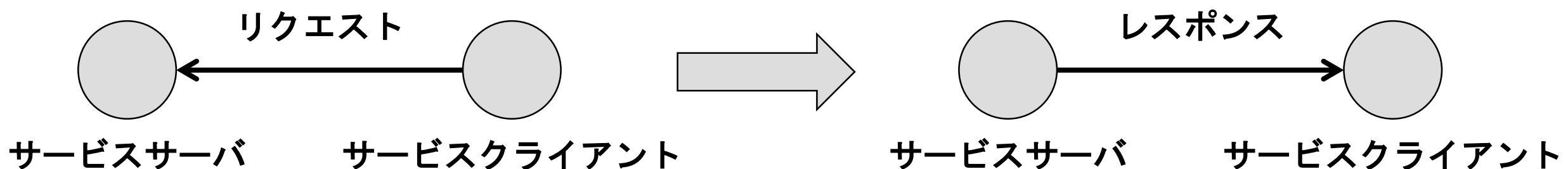
# トピックメッセージ通信

- 情報を送信する配信者(Publisher)と情報を受信する購読者(Subscriber)が、トピック(Topic)とよばれる形式でメッセージを送受信する方法
  - 一つの配信者から複数の購読者への通信
  - 複数の配信者から一つの購読者への通信
  - 一つの配信者から一つの購読者への通信
  - 多数の配信者から多数の購読者への通信



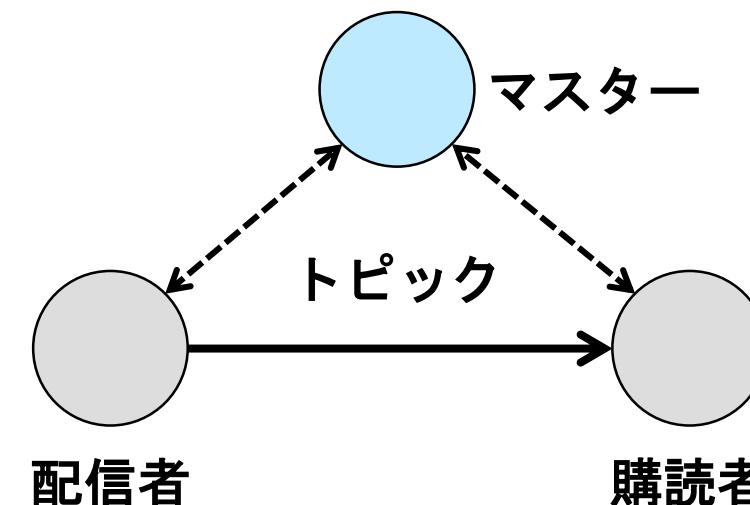
# サービスメッセージ通信

- ・サービスをリクエスト (request) するサービスクライアント (service client) と、レスポンス (response) を返すサービスサーバ (service server) 間における  
双向サービスメッセージ通信
  - ・クライアントがサーバにリクエストすると、サーバはクライアントにそのリクエストに応じたレスポンスを返す
  - ・一対一の通信のみが可能



# マスターによるノードの管理

- 配信者，購読者，サービスサーバ，サービスクライアントの情報は，それぞれ異なるノードの中に存在する
- これらのノードがメッセージ通信をするには，ノードを接続する
- このノード間の接続を管理するのがマスター



# マスターの実行

- ・ノード間のメッセージ通信の接続情報を管理するマスターを最初に実行する必要がある
- ・`roscore`コマンドによりサーバ（マスター）を動作させる

\$ **roscore**

- ・ノード間の接続のために、ノードの名前、トピックとサービスの名前、メッセージの形式、URIアドレスとポートを登録し、他のノードから問い合わせがあったときに、必要なノードの情報を送信する

# 購読者ノード，配信者ノードの実行

- ・ 購読者ノードおよび配信者ノードは， rosrun または roslaunch コマンドで実行できる
  - ・ 購読者ノードは実行時に， マスターに自分の購読者ノード名， 購読しようとするトピック名， メッセージの形式， URIアドレスとポートを登録する
  - ・ 配信者ノードは実行時に， マスターに自分の配信者ノード名， 配信しようとするトピック名， メッセージの形式， URIアドレスとポートを登録する

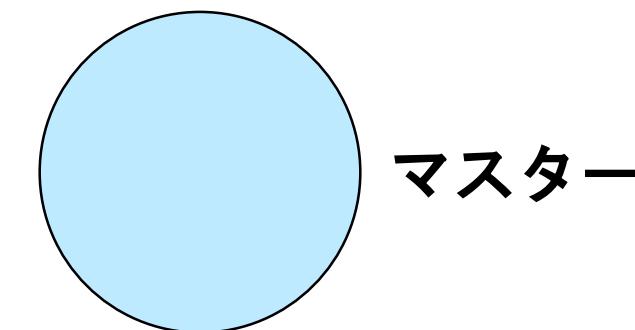
```
$ rosrun [PACKAGE_NAME] [NODE_NAME]  
$ roslaunch [PACKAGE_NAME] [NODE_NAME]
```

# ROSノードの接続

- マスターは、購読者ノードに購読者が接続したい配信者ノード名やトピック名などの情報を送信する
- 購読者ノードは、マスターから受信した配信者の情報に基づいて、配信者ノードに直接接続を要請する
  - この際に送信される情報には、自分の購読者ノード名、トピック名、メッセージ方式が含まれる
- 購読者ノードは配信者ノードに対するクライアントを生成し、配信者ノードと直接接続する
  - 購読者ノードと配信者ノード間はTCP/IP方式で通信する
  - 接続後、配信者ノードは、購読者ノードにトピックメッセージを送信する

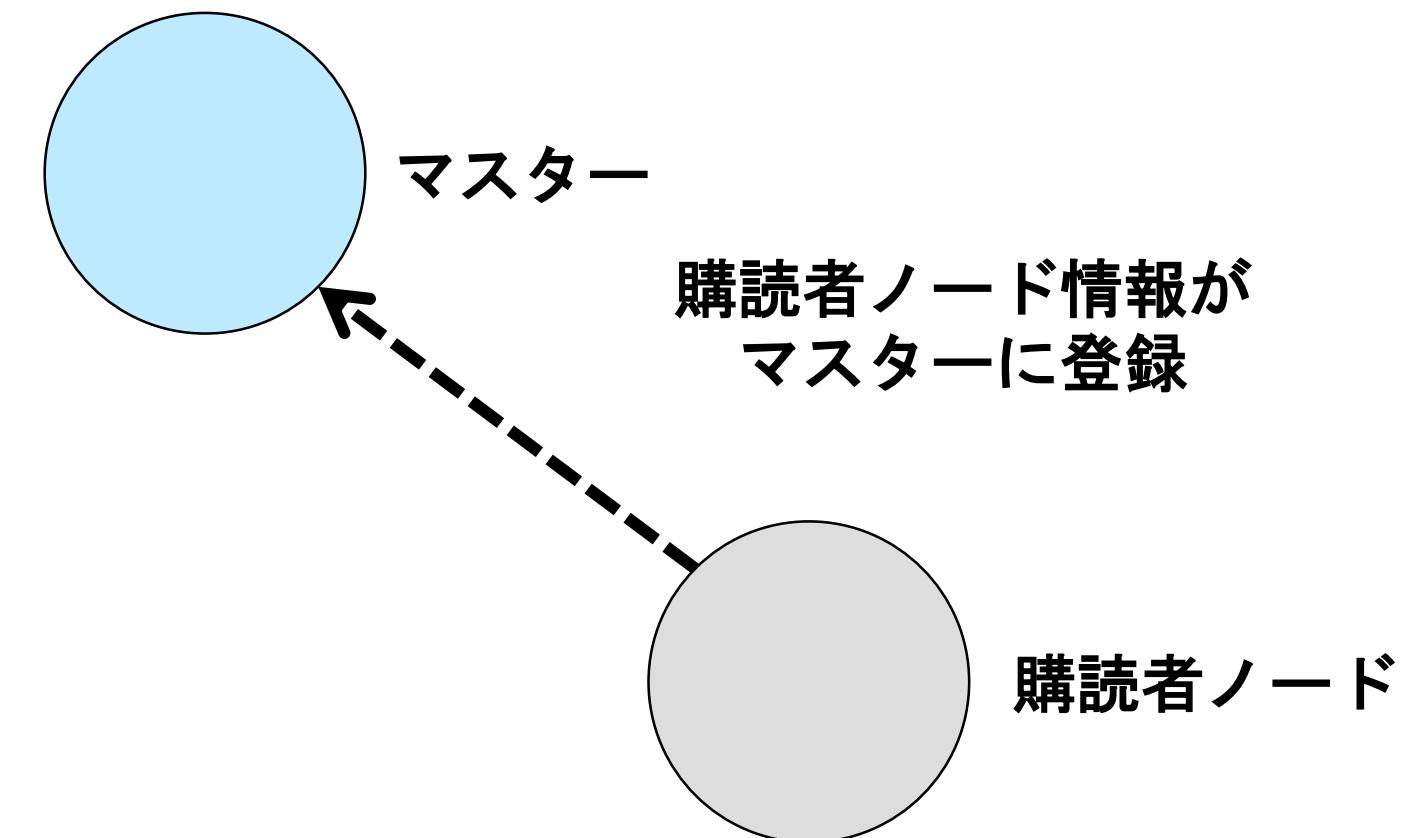
# ROSのトピックメッセージ通信

1. マスターを実行



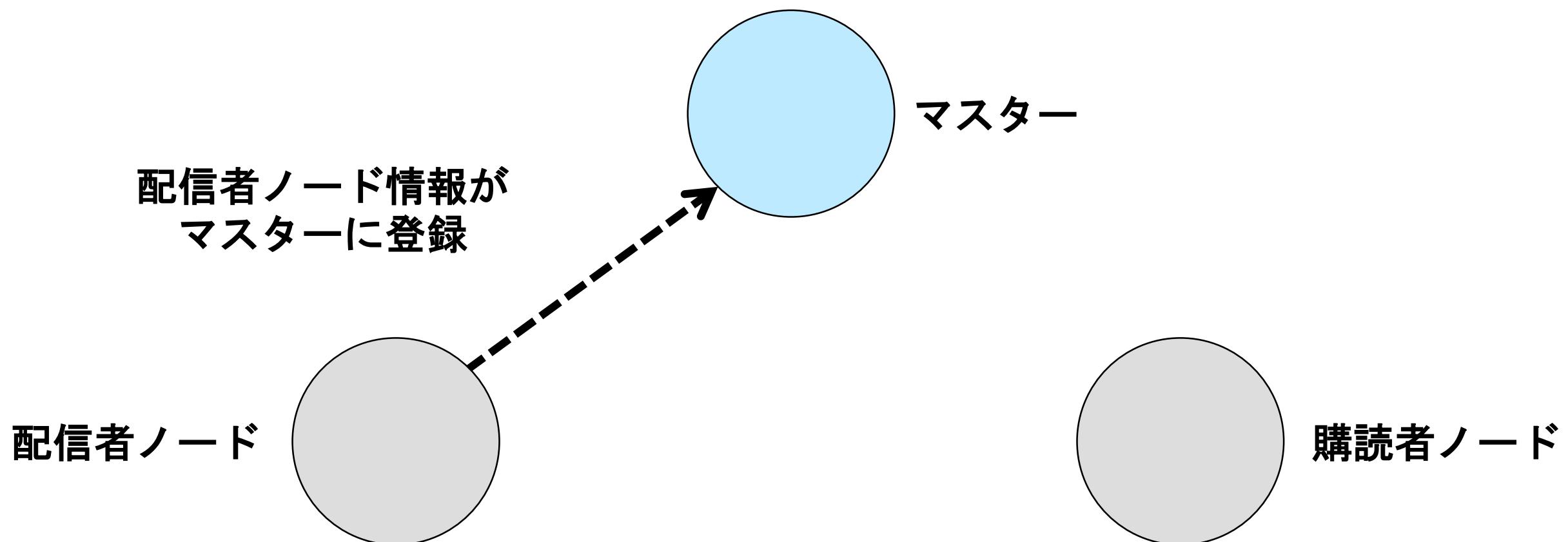
# ROSのトピックメッセージ通信

## 2. 購読者ノードを実行



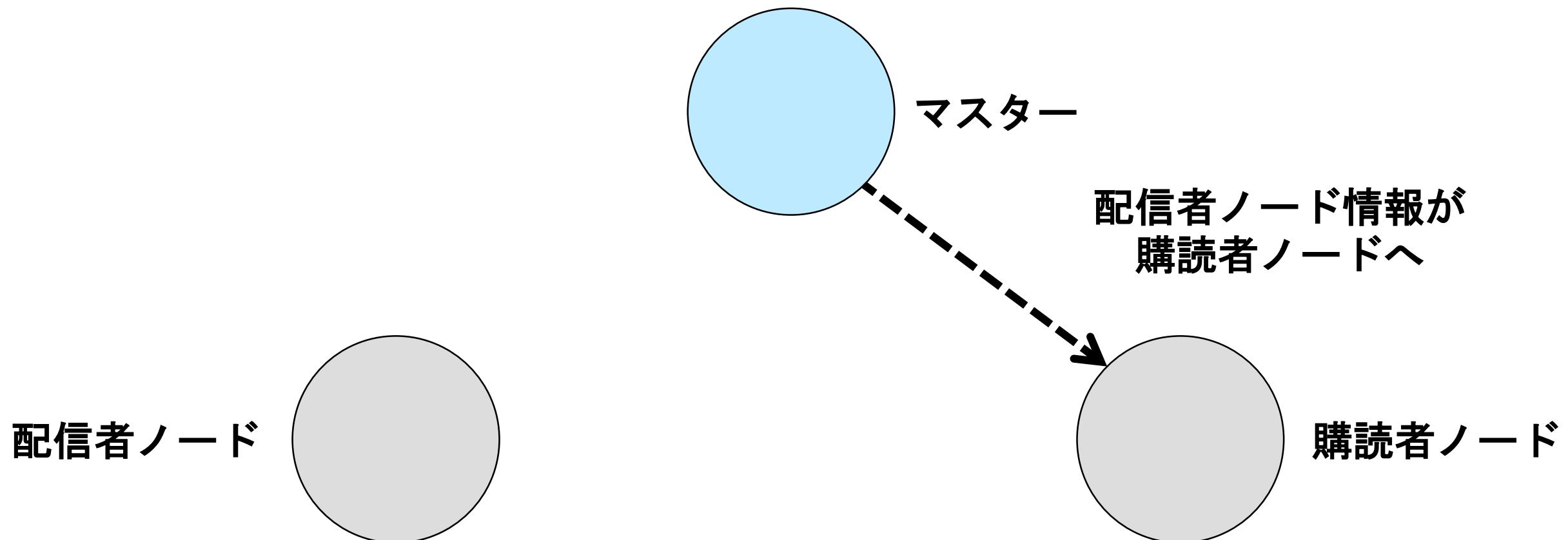
# ROSのトピックメッセージ通信

## 3. 配信者ノードを実行



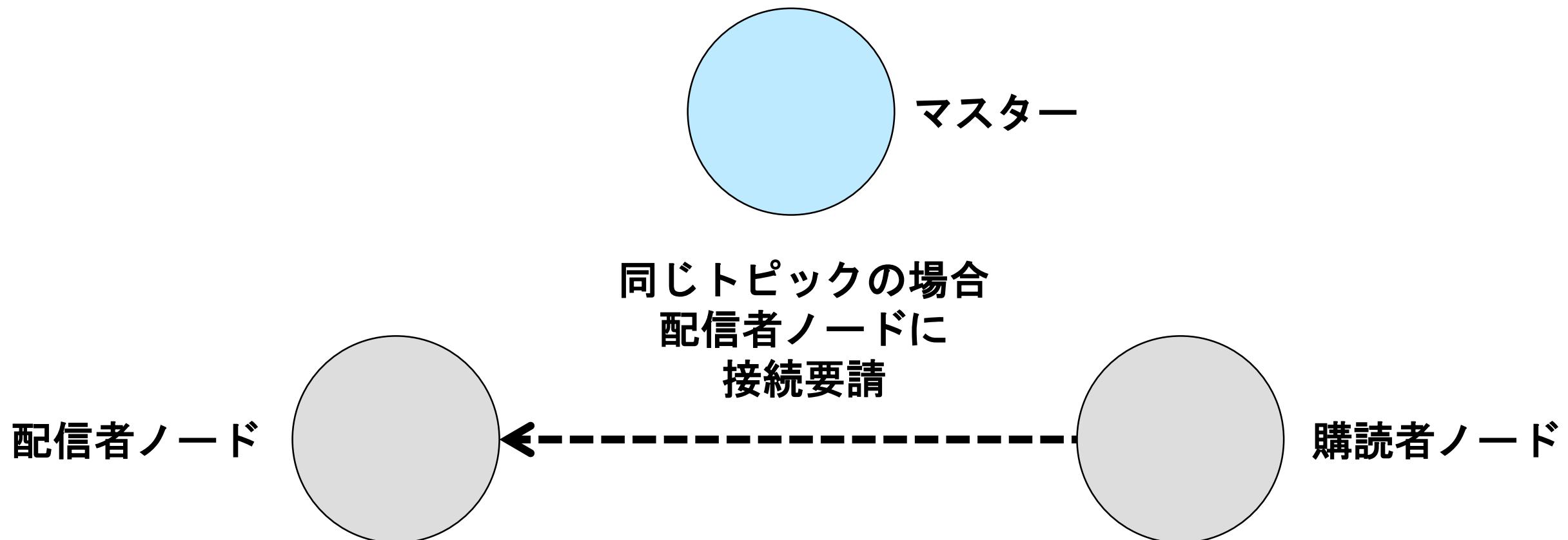
# ROSのトピックメッセージ通信

4. マスターから購読者ノードに配信者ノード情報が送信



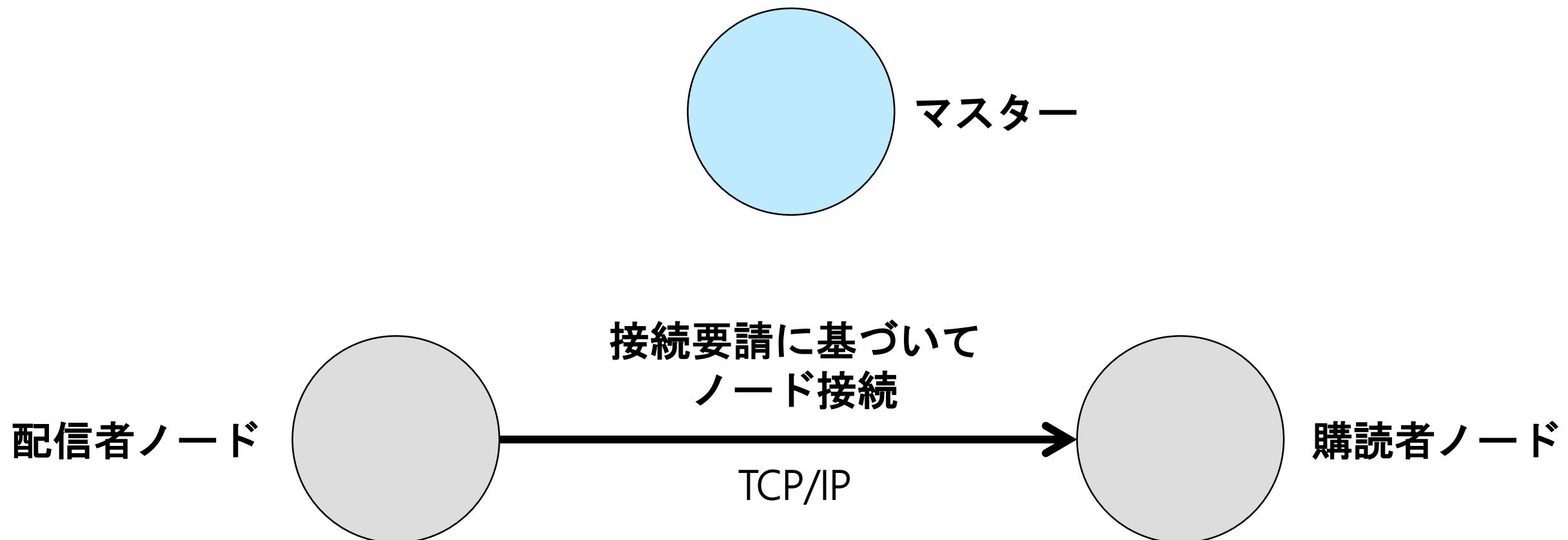
# ROSのトピックメッセージ通信

5. 購読者ノードに届いた情報に基づいて配信者ノードへ接続要請



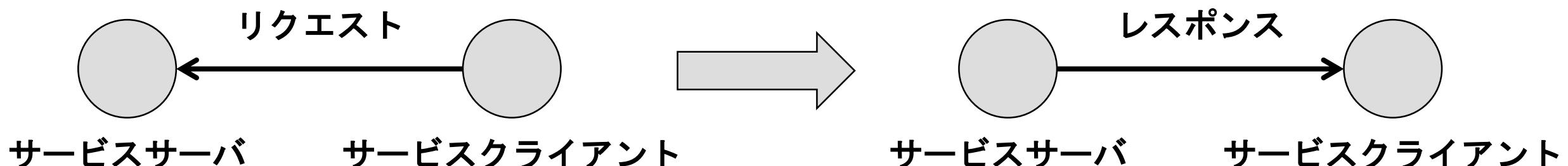
# ROSのトピックメッセージ通信

- 接続要請に基づいて、ノード間接続を行う



# ROSのサービスメッセージ通信

- ・サービスサーバとクライアントの接続は、配信者ノードと購読者ノード間の接続と同様にTCP/IP接続
  - ・サービスメッセージ通信はトピックメッセージ通信とは異なり、1回限り接続して、リクエストとレスポンスを行い、その後お互いの接続を切断する
  - ・再度通信するときは、再び接続を行う必要がある



# ROSツールの紹介

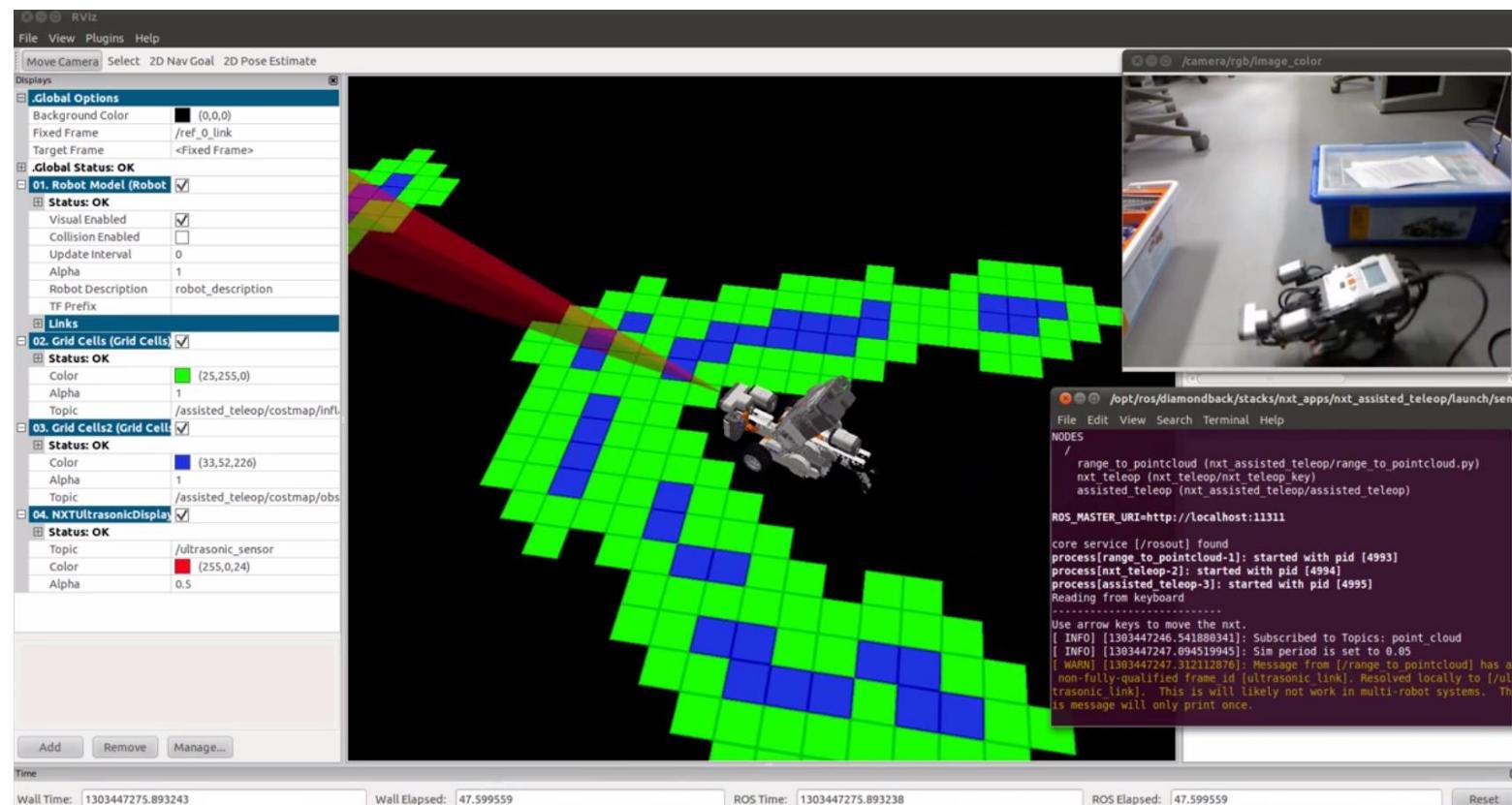


# RViz

- 3次元可視化ツール
  - ROSネットワーク上のデータを3次元的に可視化でき、センサの計測データの確認や3次元コンピュータグラフィックスによるロボットの動作確認などに利用
- 使用例
  - レーザレンジファインダ (LRF, Laser Range Finder) などの距離センサから得られる距離データ
  - KinectやXtionなどの3次元センサから得られる点群データ (PCD, Point Cloud Data)
  - カメラから取得したカラー画像

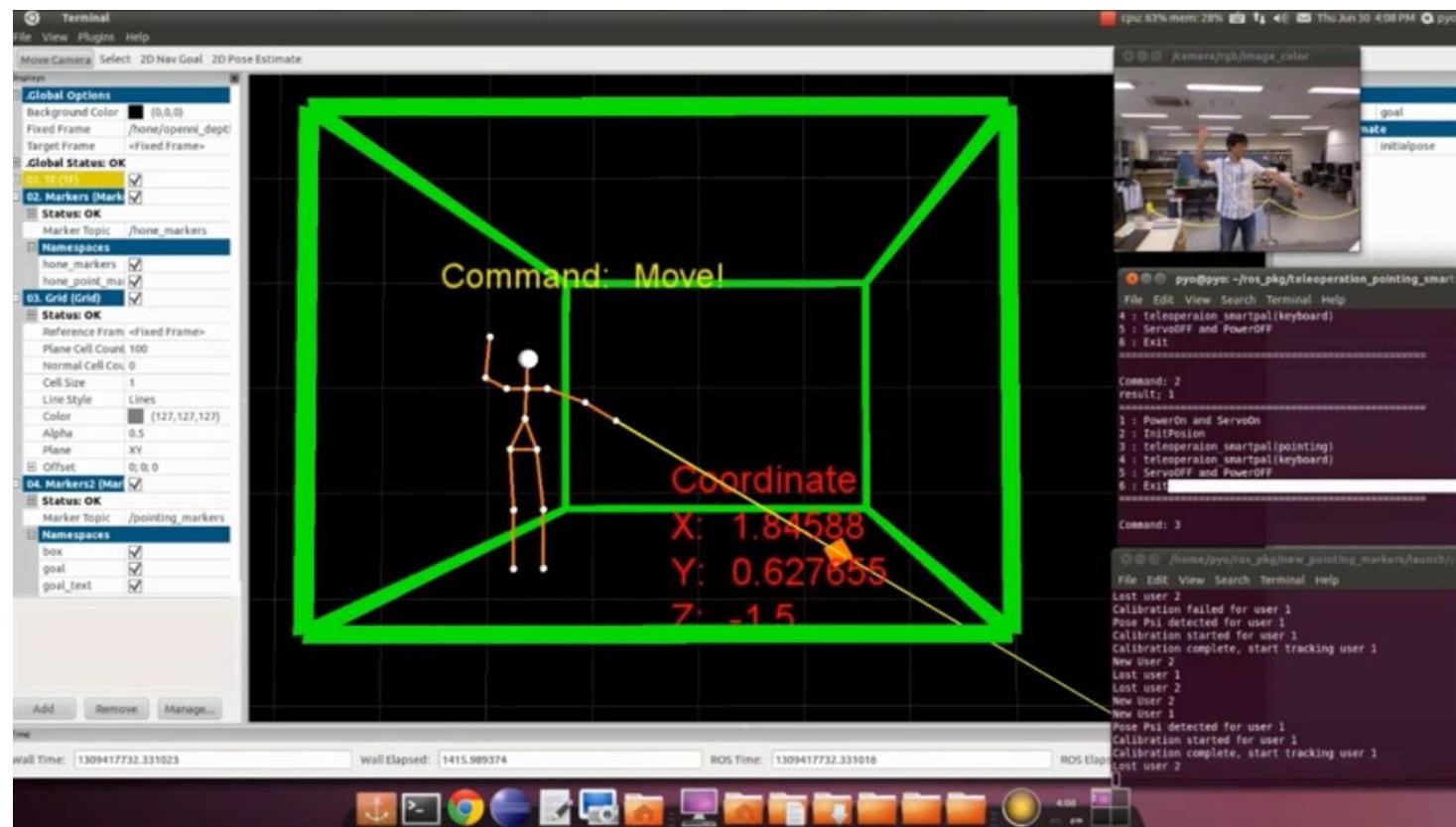
# RVizの使用例

- LEGOロボットと超音波センサを用いた簡単なマップの作成



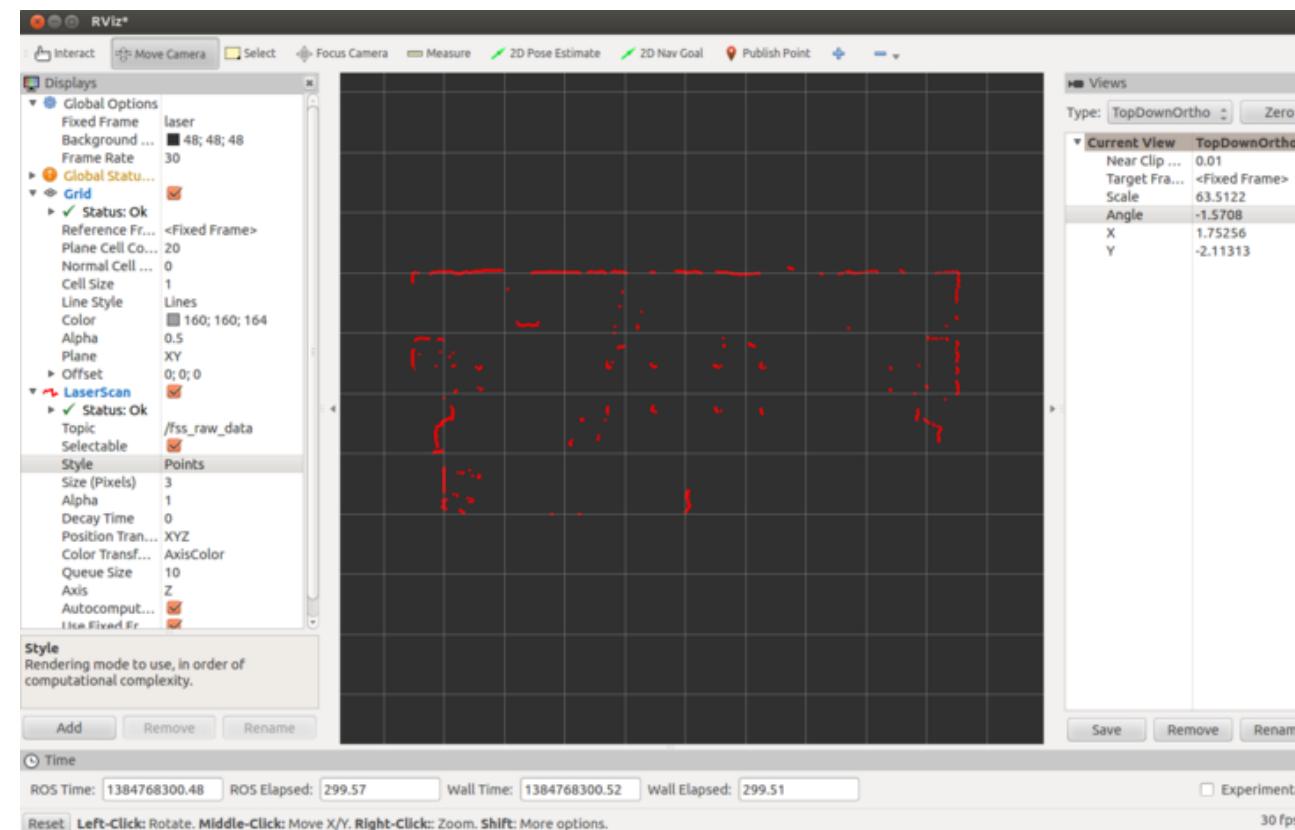
# RVizの使用例

- Kinectから人の骨格を取得し、ロボット制御に利用



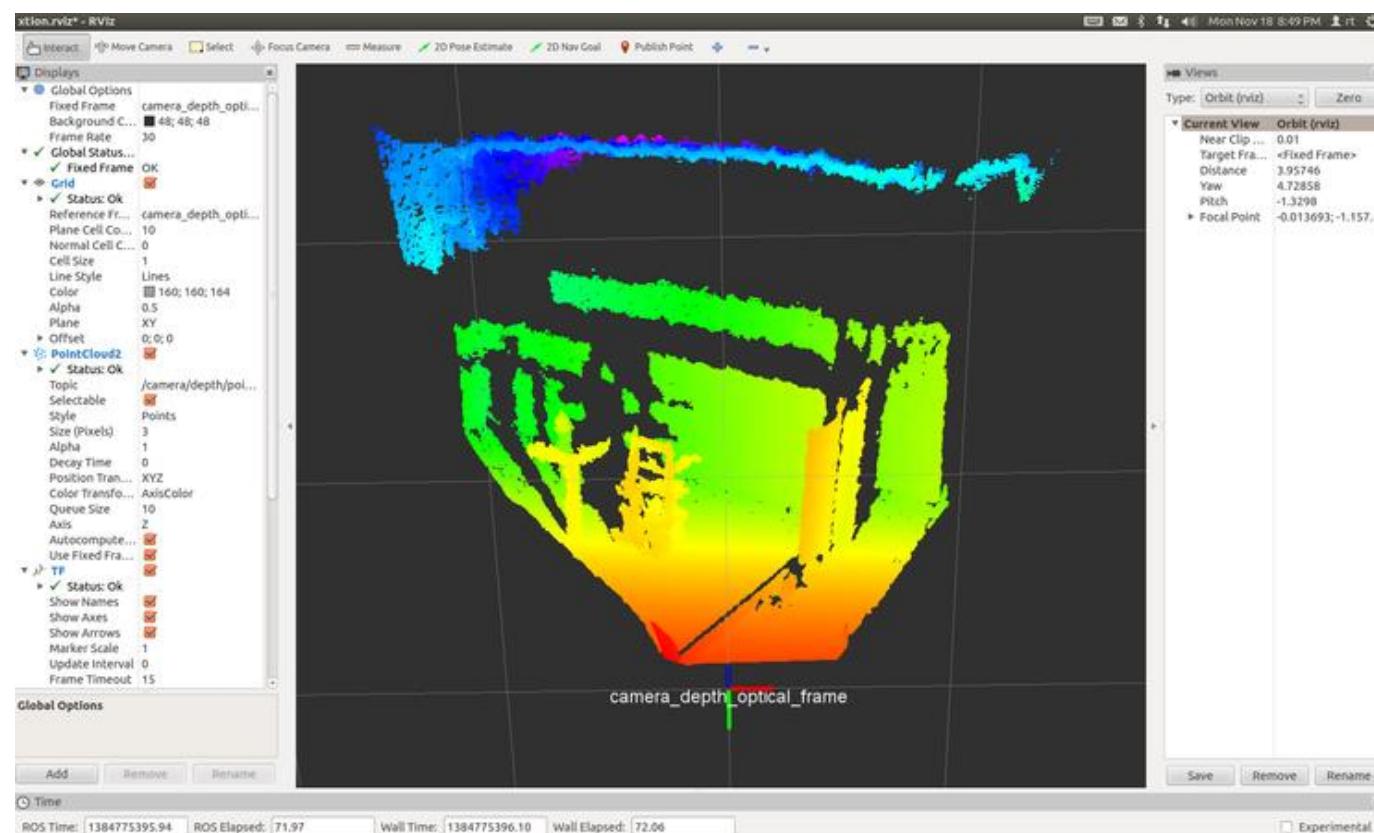
# RVizの使用例

- LRFを用いた距離計測



# RVizの使用例

- Xtionセンサから取得した3次元距離値の可視化

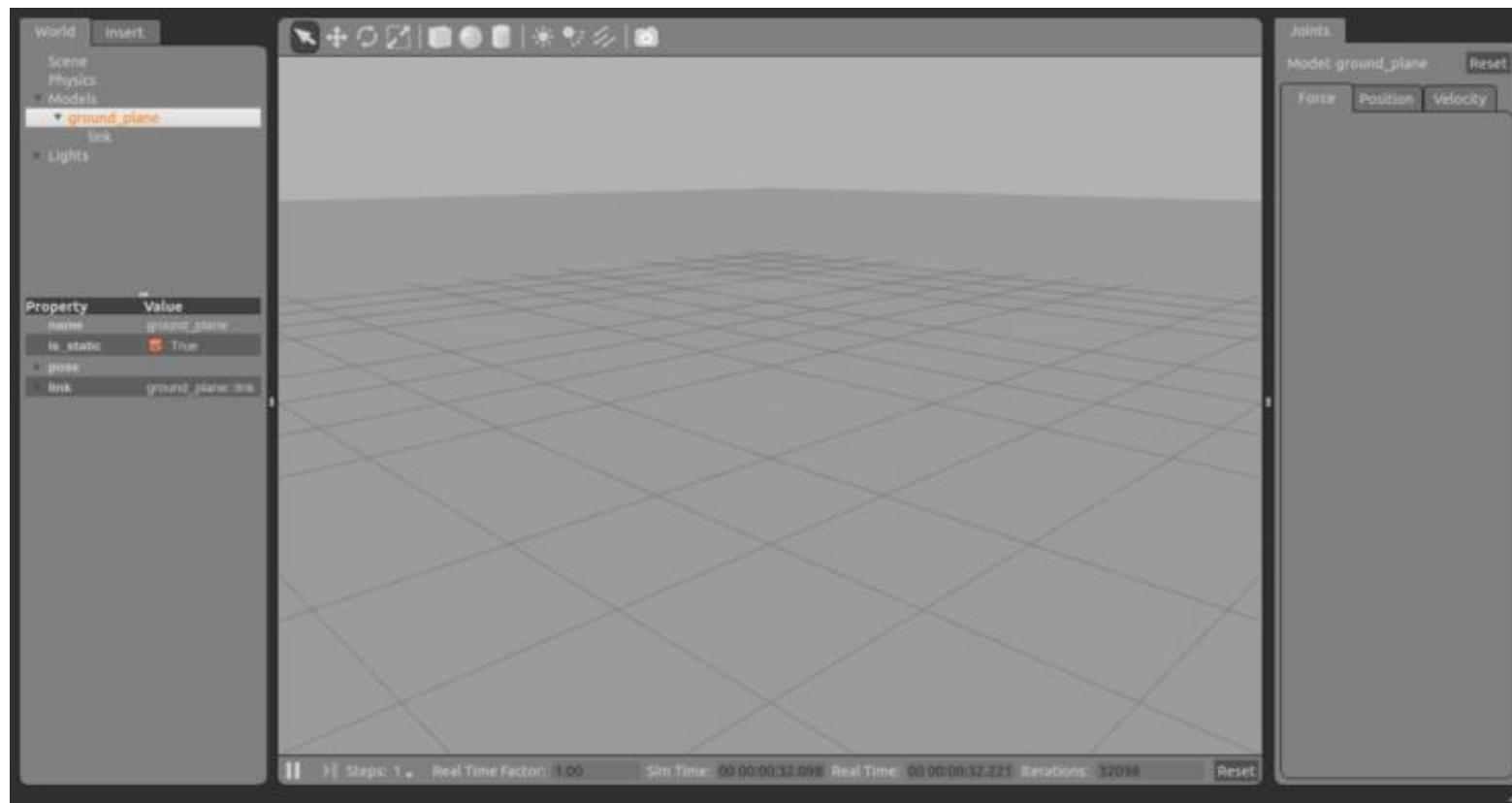


# Gazebo

- 物理エンジンを搭載したロボットアプリケーション開発のための3次元シミュレータ
  - 米国DARPA Robotics Challengeの公式シミュレータに選定
  - ROSへの導入も容易
  - 様々な物理エンジンによる動力学シミュレーション
  - 高度な3次元グラフィックス機能
  - 多様なセンサ， LRF， カメラ， デプスカメラ， 接触センサなどを再現
    - 検出されたセンサデータへのノイズの付加も可能
  - 多様なロボットモデルを導入可能
    - 独自のロボットも追加可能

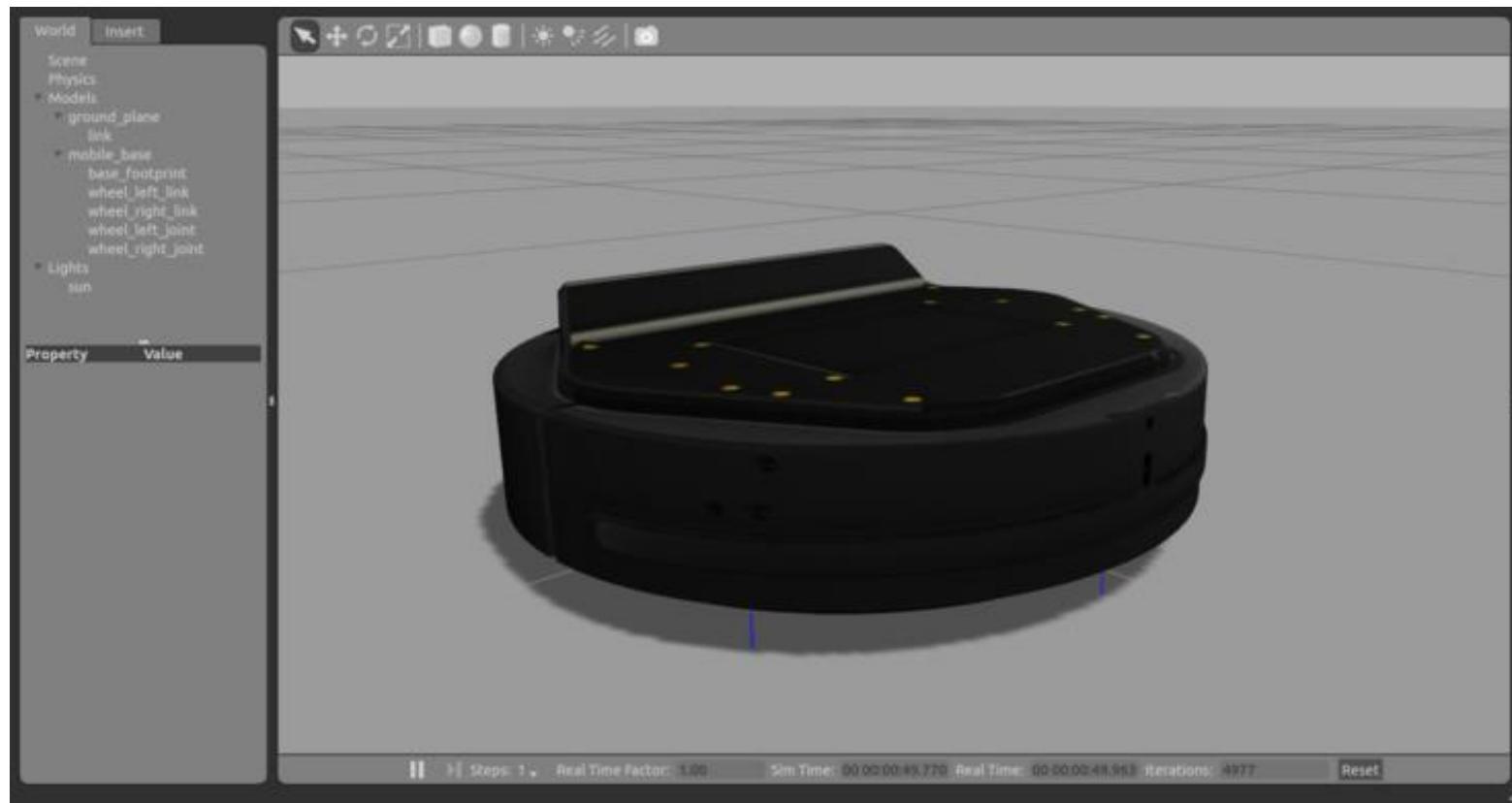
# Gazeboの例

- 初期画面



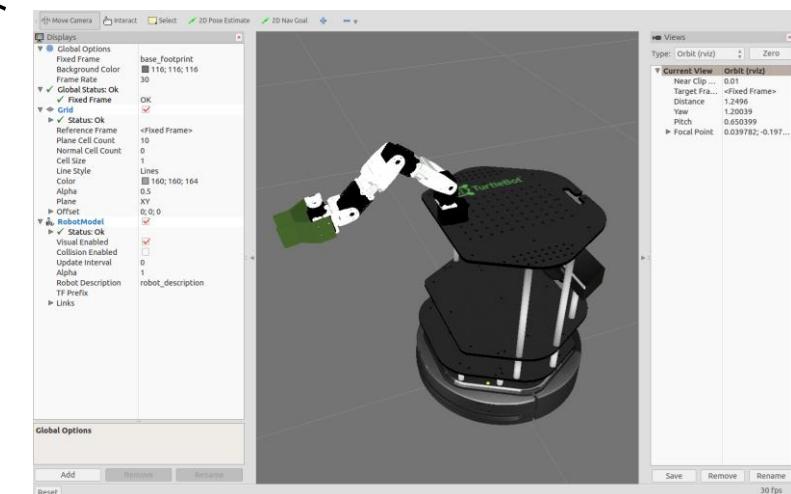
# Gazeboの例

- kobukiの3次元外観



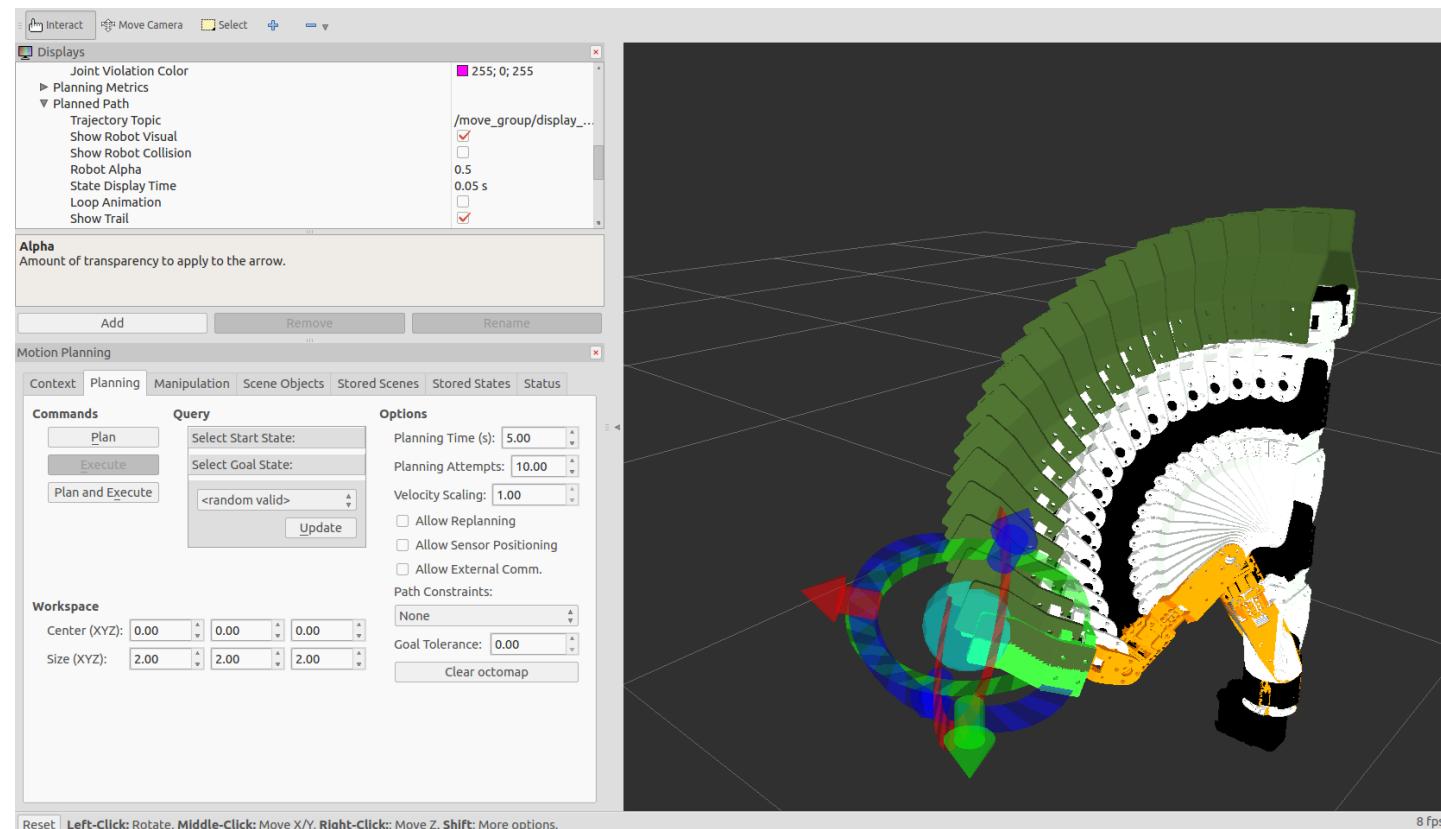
# Movelt

- 動作計画，3次元知覚，運動学計算，制御などを提供するパッケージ群
  - ロボットで頻繁に取り扱うロボットアームの順運動学計算と逆運動学計算が可能
  - 物体把持やピック＆プレイス作業などが簡単に実行
  - 65種以上のロボットアームの動作計画をサポート



# Movelitの例

- ・タートルボットアームが目標に向かって移動する様子

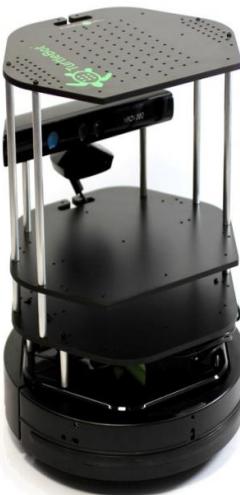


# ROSの使用例



# 使用例 | Turtlebot2(Kobuki)

- Turtlebot2をROSに適用
  - 下部分にはKobukiとよばれるロボット
  - KobukiとラップトップPCを有線接続し，PCから命令コマンドを送る
  - Kinectや3Dカメラを接続することが可能



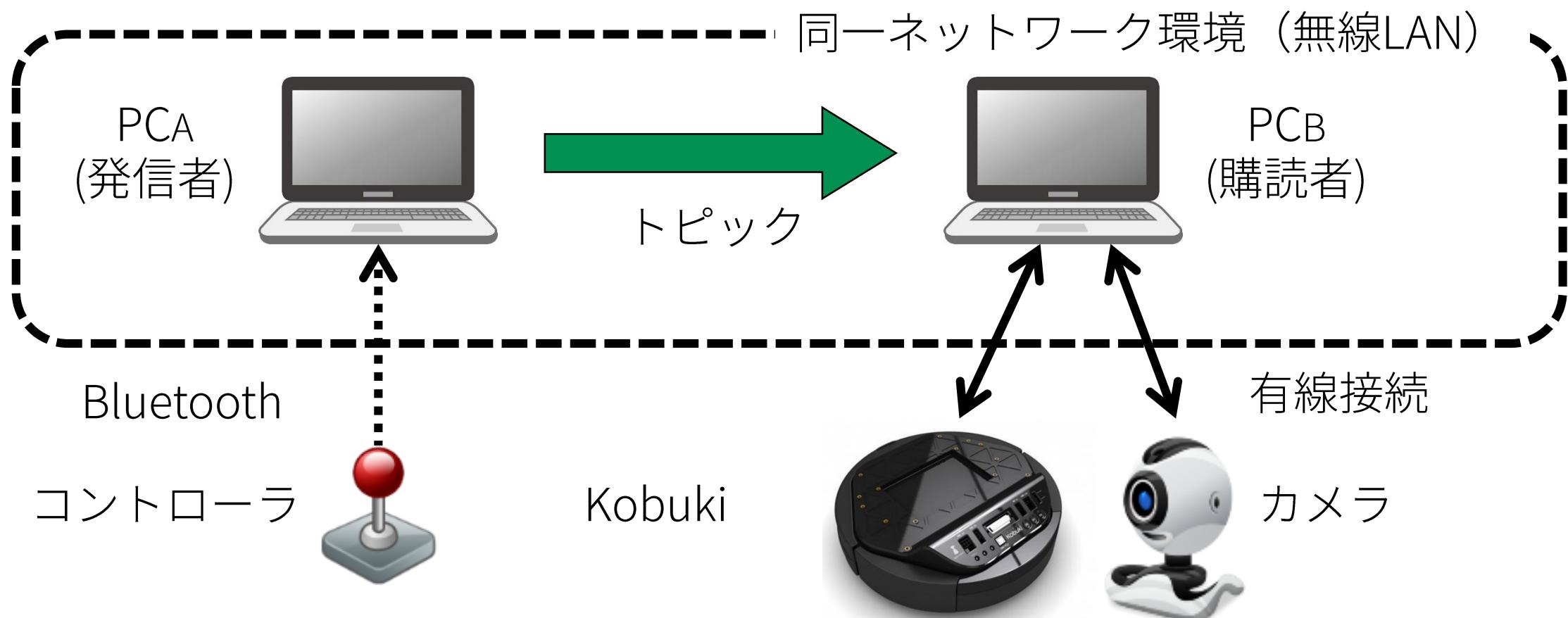
Tuttlebot2システム



Kobuki

# ROSによるTurtlebot2制御

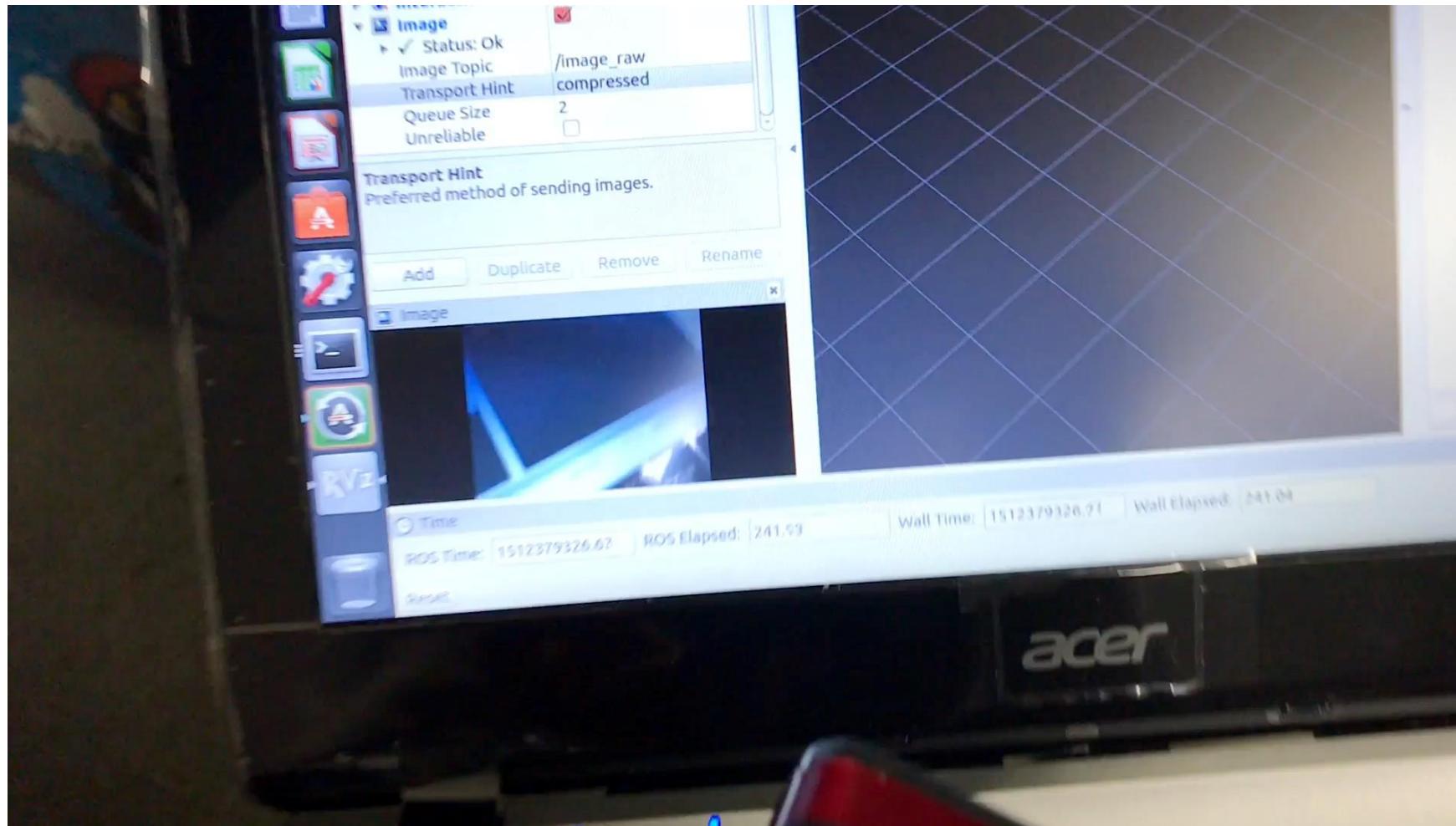
- システム構成図 (2台のPCを使用)



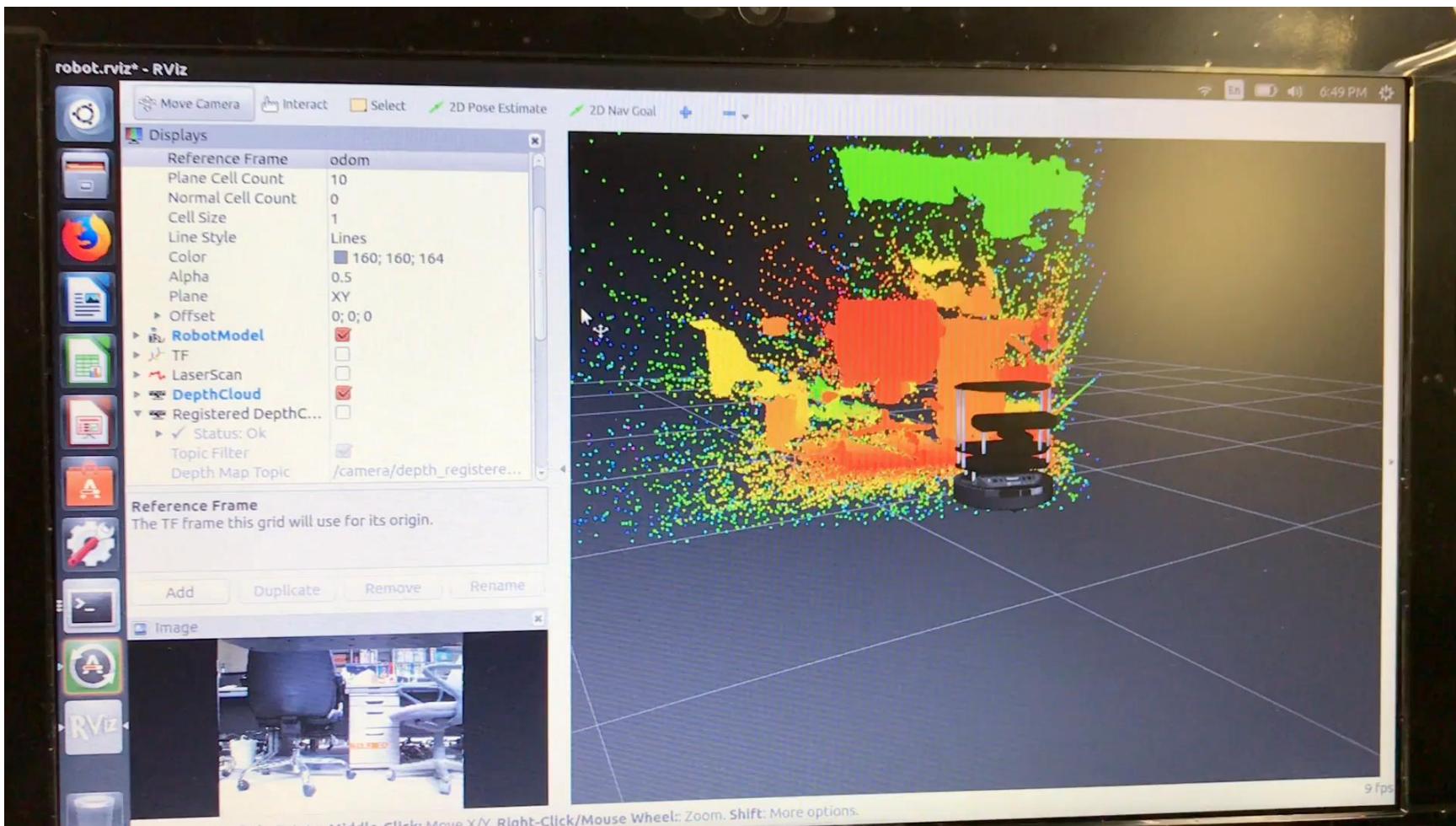
# コントローラによる操作



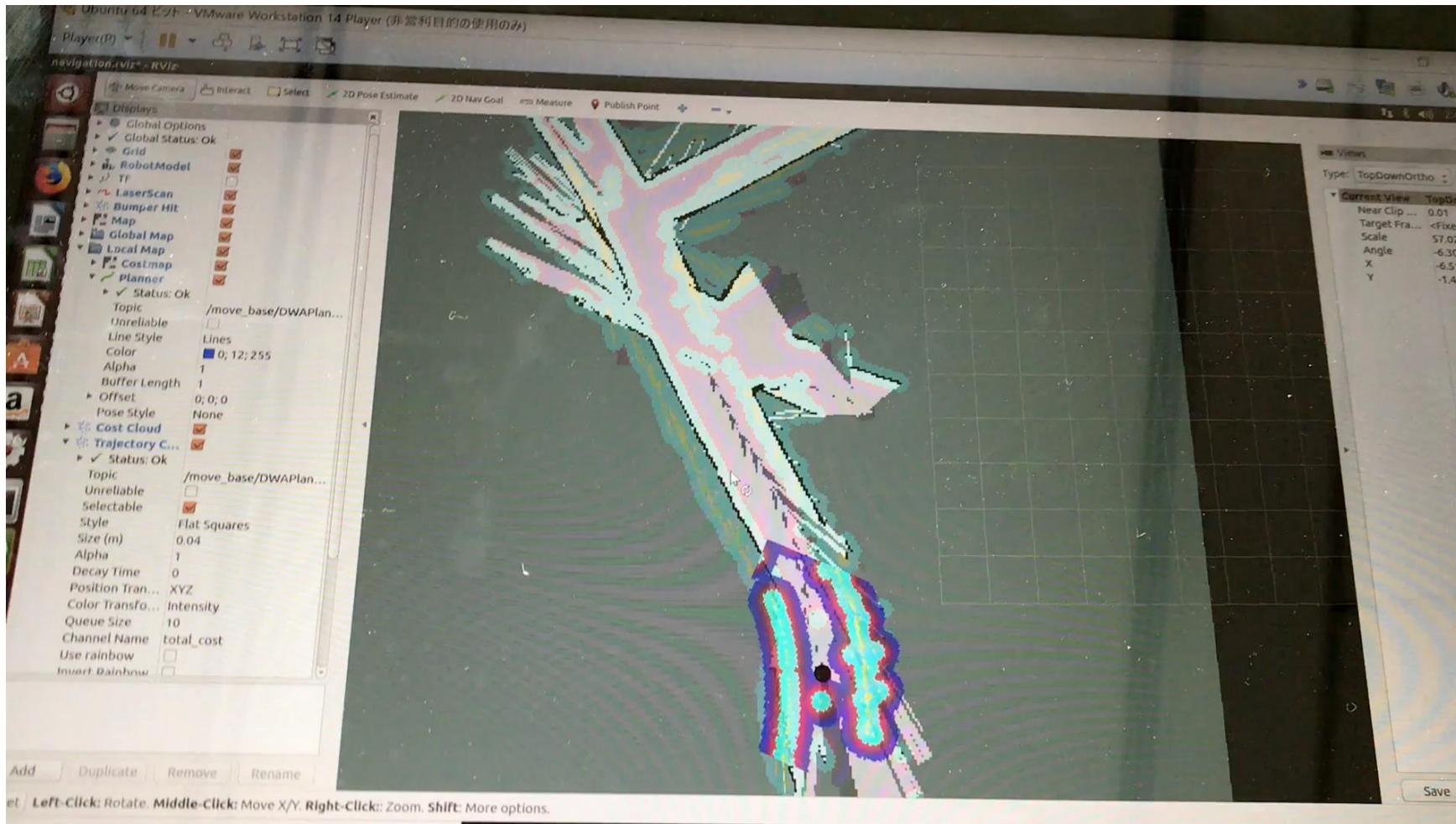
# USBカメラ表示例



# 3Dカメラ表示例



# SLAMを用いた地図生成

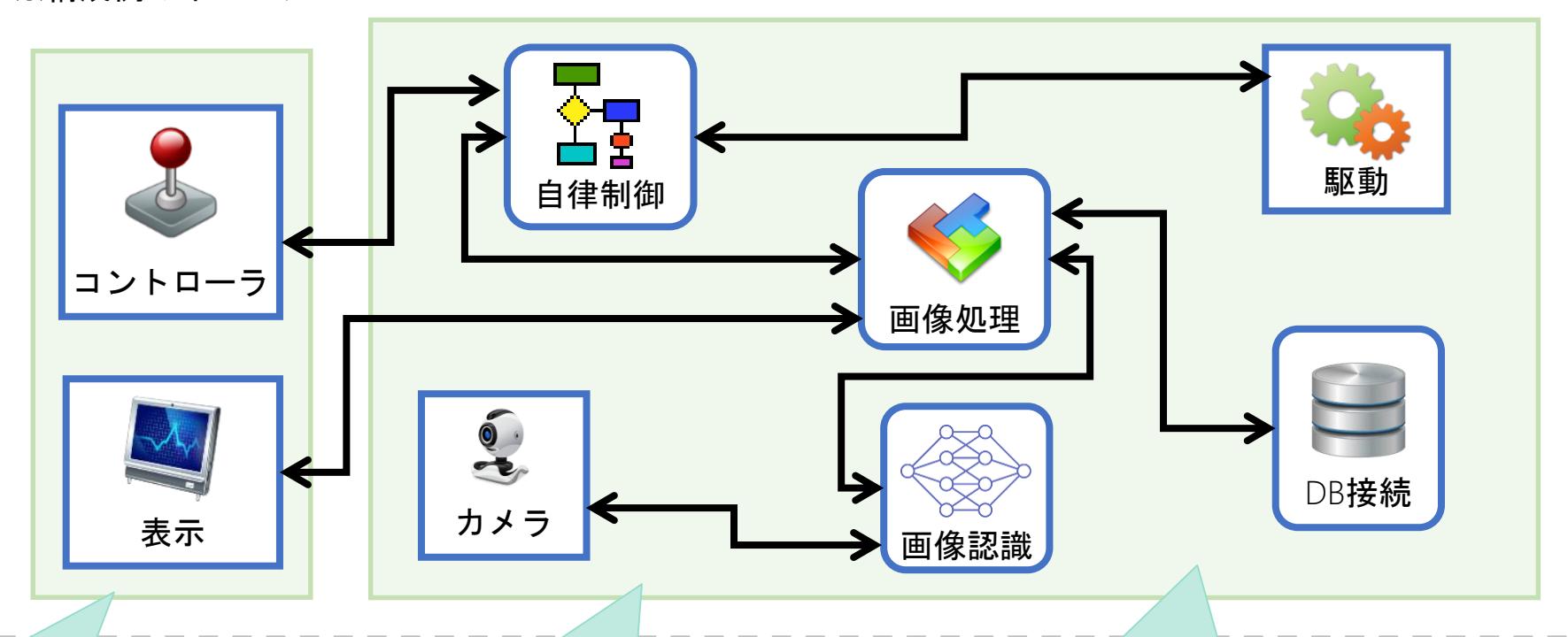


# 今後のロボット制御について



# 分散システムによるロボット開発

※構成例のイメージ



簡単に部品を切り替えできる  
多様性

デバイスやアルゴリズム単位で  
コンポーネント化

ネットワーク透過なオブジェクトを  
分散配置し機能を分割

標準化

ミドルウェア

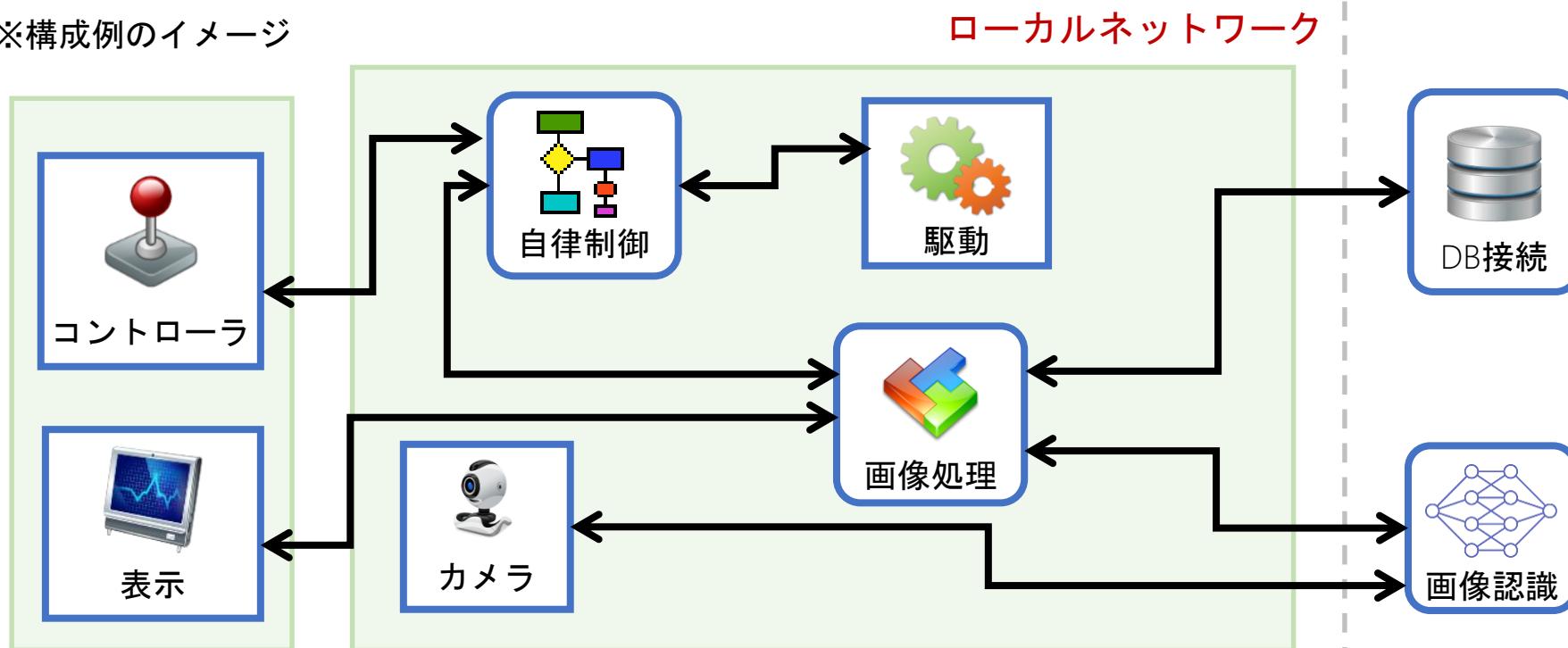


- ・ 多機能性 +
- ・ 高性能化 +
- ・ 低コスト化 +

# クラウドロボティクス

グローバルネットワーク：データ保存や高速処理を必要とする機能をクラウドへ

※構成例のイメージ



## 研究課題

ネットワーク  
アーキテクチャ

セキュリティ

安全性

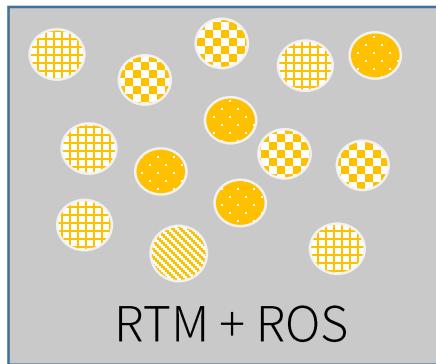
- 多機能性++
- 高性能化++
- 低コスト化++

# ロボット人材スキルレベル

スキル レベル	クラウド ロボティクス	ロボットSI	ロボットSW	ロボット回路	ロボット機構
レベル 5	クラウド（DB, AI, 機械学習など）を活用した知的ロボットネットワーク設計	HW・SWを含む実用的ロボットシステムを設計			
レベル 4	クラウドを活用したロボットシステムの設計（セキュリティ対策の基本を含む）	HW・SWを含む高度なロボットサブシステムを設計	実用的なロボットシステムSWを設計	実用的なロボットシステム回路を設計	実用的なロボットシステム機構を設計
レベル 3	クラウドを活用して Network, DB, AI, 機械学習などの基本をロボットに適用	HW・SWを含むロボットサブシステムの基本設計	ロボットシステムの基本 SW設計 AI, 機械学習などの要素技術習得	ロボットシステムの基本回路設計	ロボットシステムの基本機構設計
レベル 2	Network, DB, セキュリティなどのクラウド要素技術習得	ロボットミドルウェアの仕組みを理解し、ロボットシステムを実装	OSSなどを活用したサブシステムの機能実現	ロボットサブシステムの基本回路設計	ロボットサブシステムの基本機構設計
レベル 1	Network, DBの仕組みの理解	プログラムによる回路制御	C言語などによるプログラミングスキルの取得	ロボットサブシステムの回路理解	ロボットサブシステムの機構理解

# 今後のロボット制御

ロボットSWライブラリ  
Robot Software Library



ロボット開発  
Robot development



## <フィジカル Physical >

センシングデータ  
Sensing data



附加価値データ  
Added value



## <サイバー Cyber >

クラウド  
ロボティクス  
(CR)基盤

ロボットデータ  
レポジトリ  
Robot Data  
Repository



Learning/AI  
自由視点法,  
仮想俯瞰画像,  
VSM, NW, etc

シミュレータ  
Simulator

## <ロボット開発プロセス>

デザイン (CAD)  
HW + SW (RTC) モデル



シミュレータ  
Simulator



HW開発



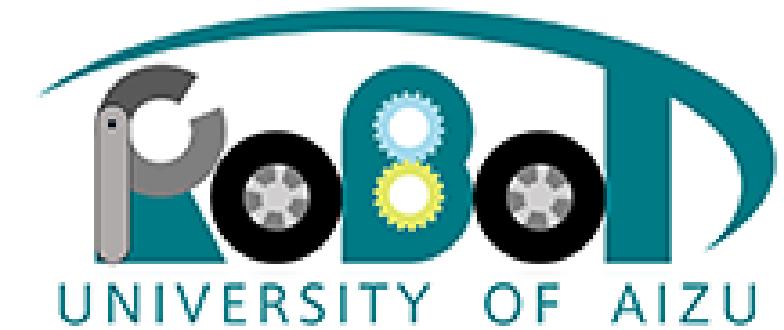
リリース

## ロボット設計・ロボットサービス技術者育成

HWとSWを理解した技術者 ("Dual-ware" Engineers)

Deep Learning・Data Scientist人材

# マウス型ロボットをROSで制御



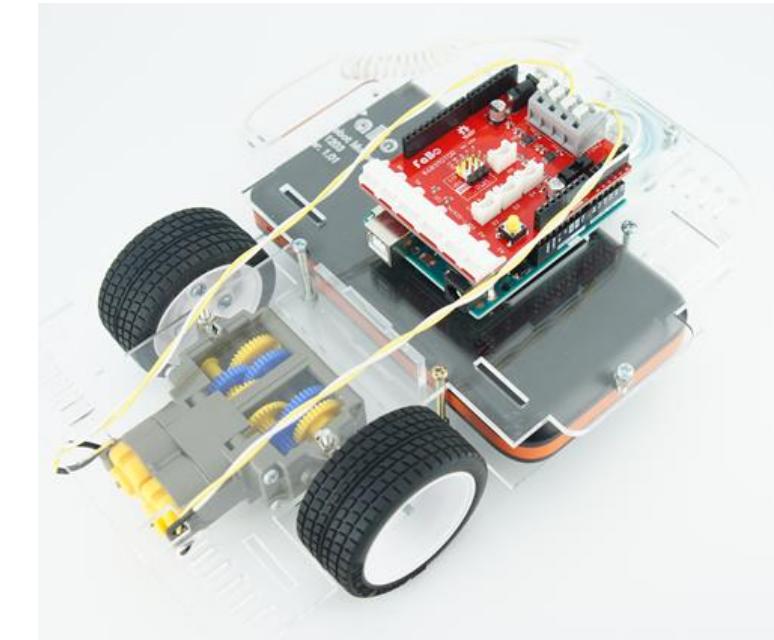
# マウス型ロボットをROSで制御

- 講習で使用しているマウス型ロボットをROSで制御
  - RaspbianにROSをソースコードからビルドして環境を構築
  - パッケージを作成してマウス型ロボットを制御



これまでの環境

ROS環境



# マウス型ロボットをROSで制御

- ・システム構成図（1台のRaspberry Piを使用）

