

2022/8/25 ROS Japan Users Group #47 シミュレータ特集！

# Choreonoid（コレオノイド）で学ぶ ROSとロボットシミュレーション

株式会社コレオノイド代表

中岡 慎一郎

# 講師紹介



## 中岡 慎一郎

東京大学（2001年～）、産総研（2006～）にて18年間  
ヒューマノイドロボットの研究に従事

2019年より株式会社コレオノイド代表取締役

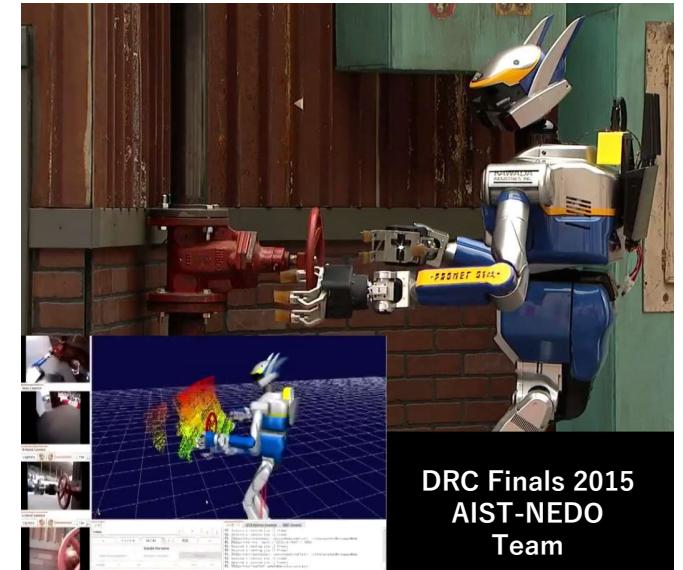
### 研究内容：二足歩行ヒューマノイドロボットの全身動作の操り



人全身動作模倣手法（2005）

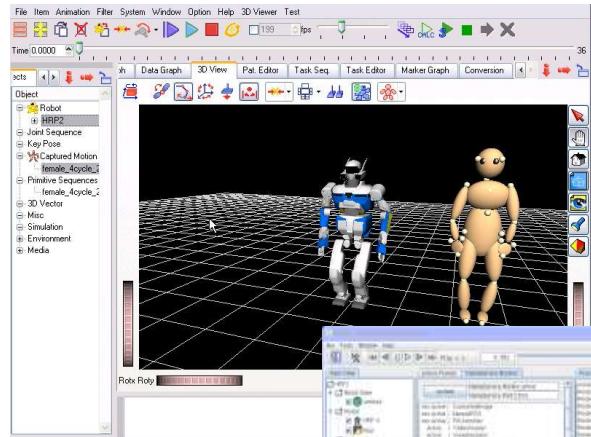


全身動作振付手法（2010）



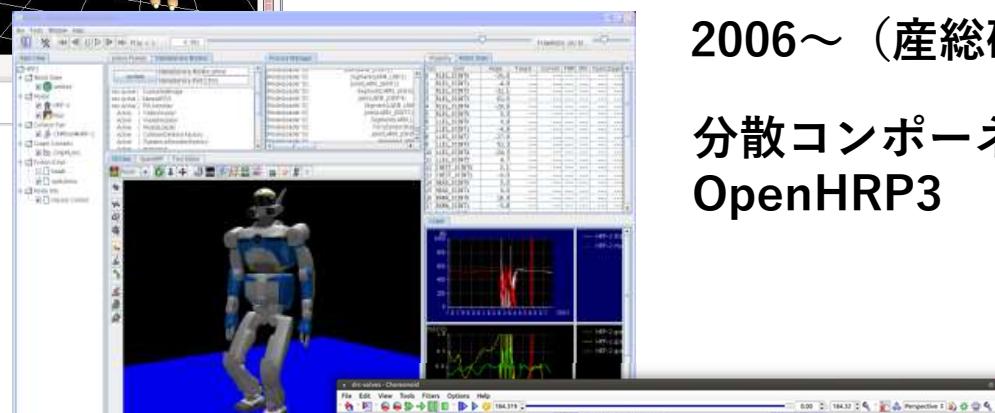
半自律遠隔操作タスク遂行手法  
(2015)

# 開発したロボットシミュレータ



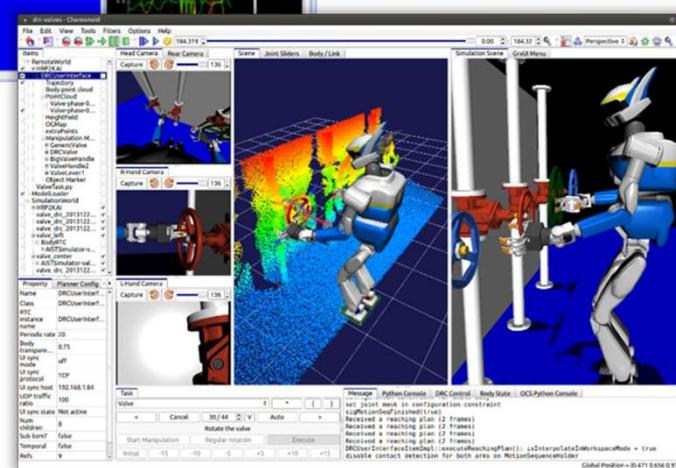
2003～（博士課程）

動作解析／生成用シミュレータ



2006～（産総研入所後）

分散コンポーネント型ロボットシミュレータ  
OpenHRP3



2008～

統合ロボットシミュレータ  
Choreonoid

# 本日のお題

## Choreonoidで学ぶROSとロボットシミュレーション

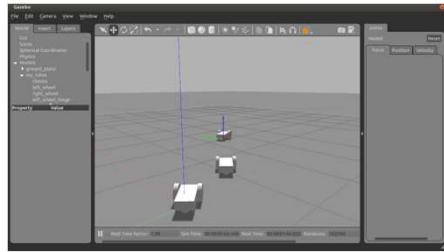
- ROS入門者
  - ROSの基本的な仕組みとその利用方法が分かる
- ロボットシミュレータ入門者
  - ロボットのシミュレーションがどのようなものか分かる
- ROS、ロボットシミュレータ経験者
  - Choreonoidがどのようなものか分かる

※ 本講義はROS1を対象とします

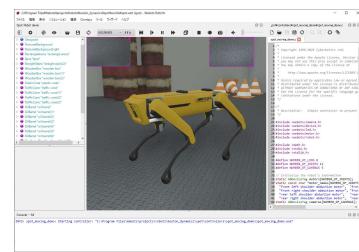
# Choreonoid?

# ロボットシミュレータ色々

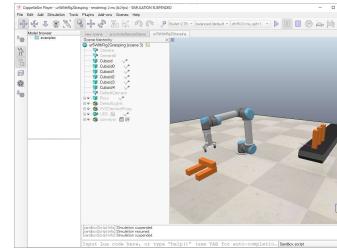
## 汎用ロボットシミュレータ



Gazebo

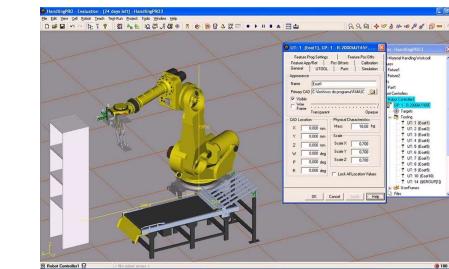


Webots



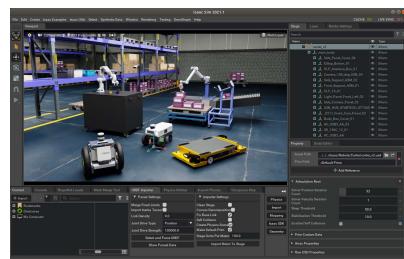
CoppeliaSim

## 産用ロボ教示用 シミュレータ

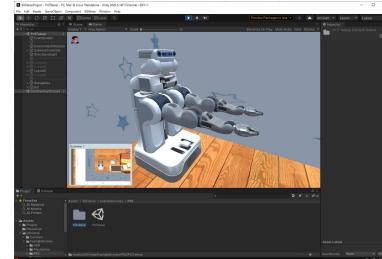


RoboGuide

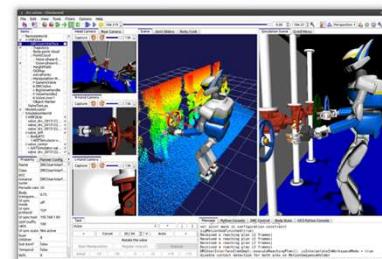
## ゲームエンジン



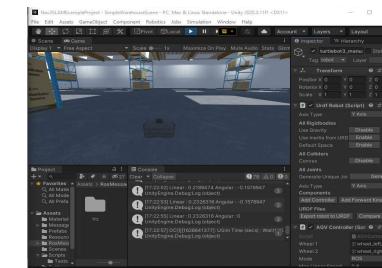
Issac Sim



SigVerse



Choreonoid



Unity

# Choreonoid?

- 実は歴史と実績のある本格的ロボットシミュレータ

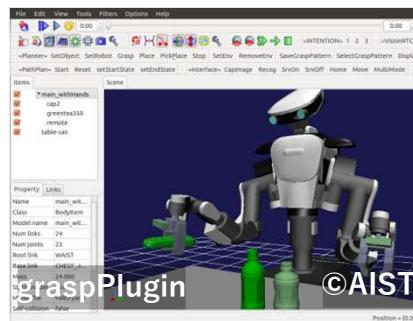
# 主な利用実績

2009年～



二足歩行ヒューマノイド  
ロボットの研究開発

2009年～



マニピュレータの把持／  
軌道／作業計画の生成

2010年～



DC-EXPO 2010 ©AIST

2014年～



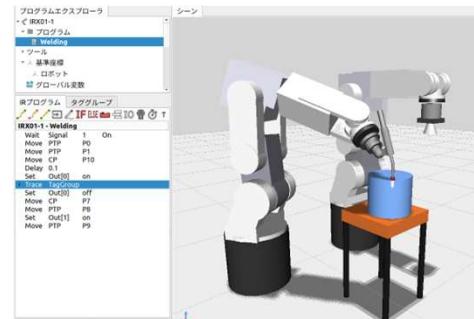
DRC Finals ©DARPA

2015年～



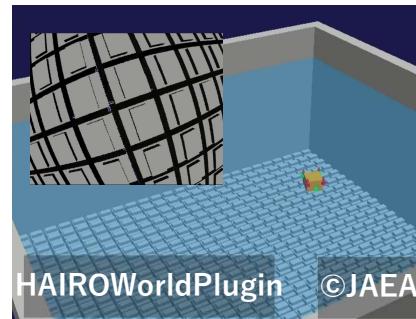
シミュレーションによる  
災害対応ロボット競技会

2019年～



産業用ロボットの  
動作教示

2020年～



HAIROWorldPlugin ©JAEA

廃炉のための  
遠隔操作技術開発

2021年～

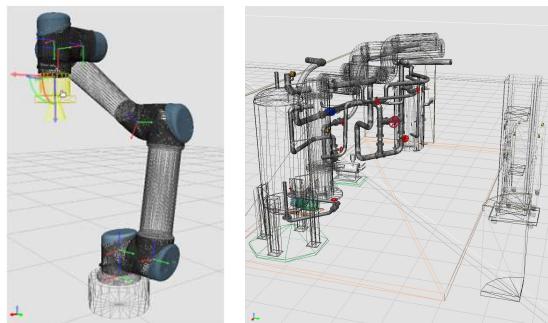


福祉機器の  
リスクアセスメント

ロボットシミュレータとしての実用性は多くの実応用で実証済み！

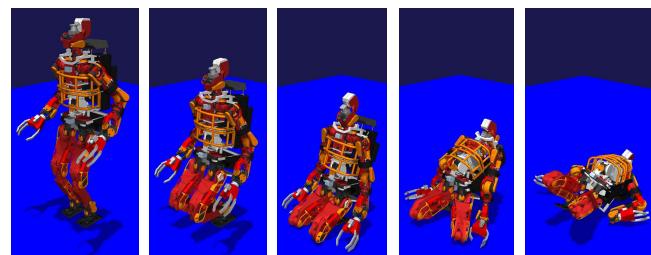
# 主な機能

## ロボット／環境モデルの読み込み／可視化／操作



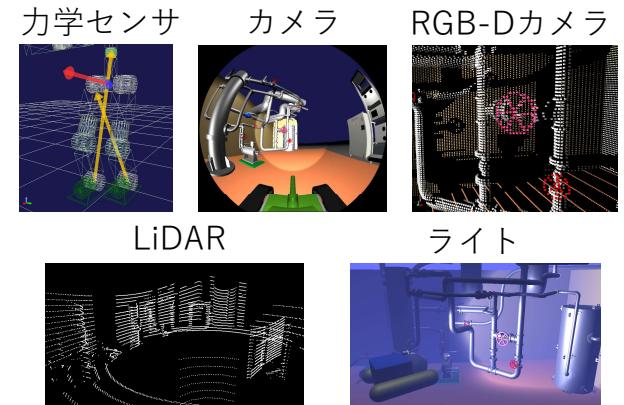
- ・機構構造／状態の可視化
- ・配置／姿勢の変更(運動学／逆運動学)

## 運動学／動力学シミュレーション



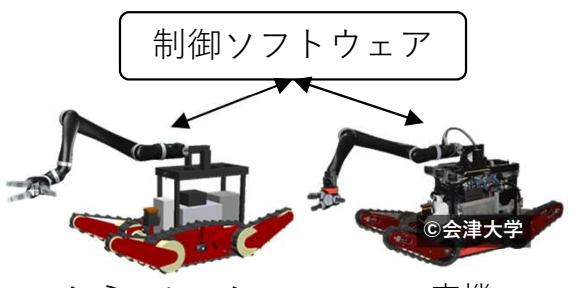
- ・多様な機構に対応  
マニピュレータ、脚、車輪、クローラ、マルチコプタ、etc.
- ・用途に適した物理エンジンを選択可能

## センサ／デバイスシミュレーション



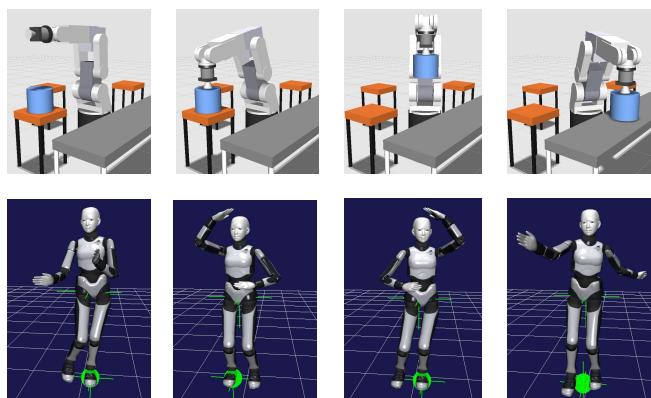
多様なセンサ／デバイスに対応

## 制御ソフトウェアの接続



- ・様々な規格(API/ミドルウェア)に対応
- ・実機と共通のソフトウェアを利用可能

## 動作教示／振付



ロボットの動作を作成・編集することが可能

## スクリプティング

```
worldItem = WorldItem()  
RootItem.instance.addChildItem(worldItem)  
  
robotItem = BodyItem()  
robotItem.load("${SHARE}/model/SR1/SR1.b  
ody")  
robot = robotItem.body  
robot.rootLink.setTranslation([0.0, 0.0,  
0.7135])  
robot.calcForwardKinematics()  
robotItem.storeInitialState()  
  
simulatorItem = AISTSimulatorItem()
```

各種操作／シミュレーションを自動化

仮想環境でロボットを扱う様々な機能を統合的に利用可能！

# Choreonoid?

- 実は歴史と実績のある本格的ロボットシミュレータ
- 圧倒的な軽さ
- 「痒い所に手が届く」 基本機能
  - タイムバー、個別リロード、etc.
- 尋常ではない柔軟性・拡張性

# デモ

- Jaxon歩行
  - <https://github.com/choreonoid/jaxon2-draft>
- HRP-4C未夢ダンス
  - Turn to the future
  - 作曲：本崎朝人
  - 作詞：中岡慎一郎

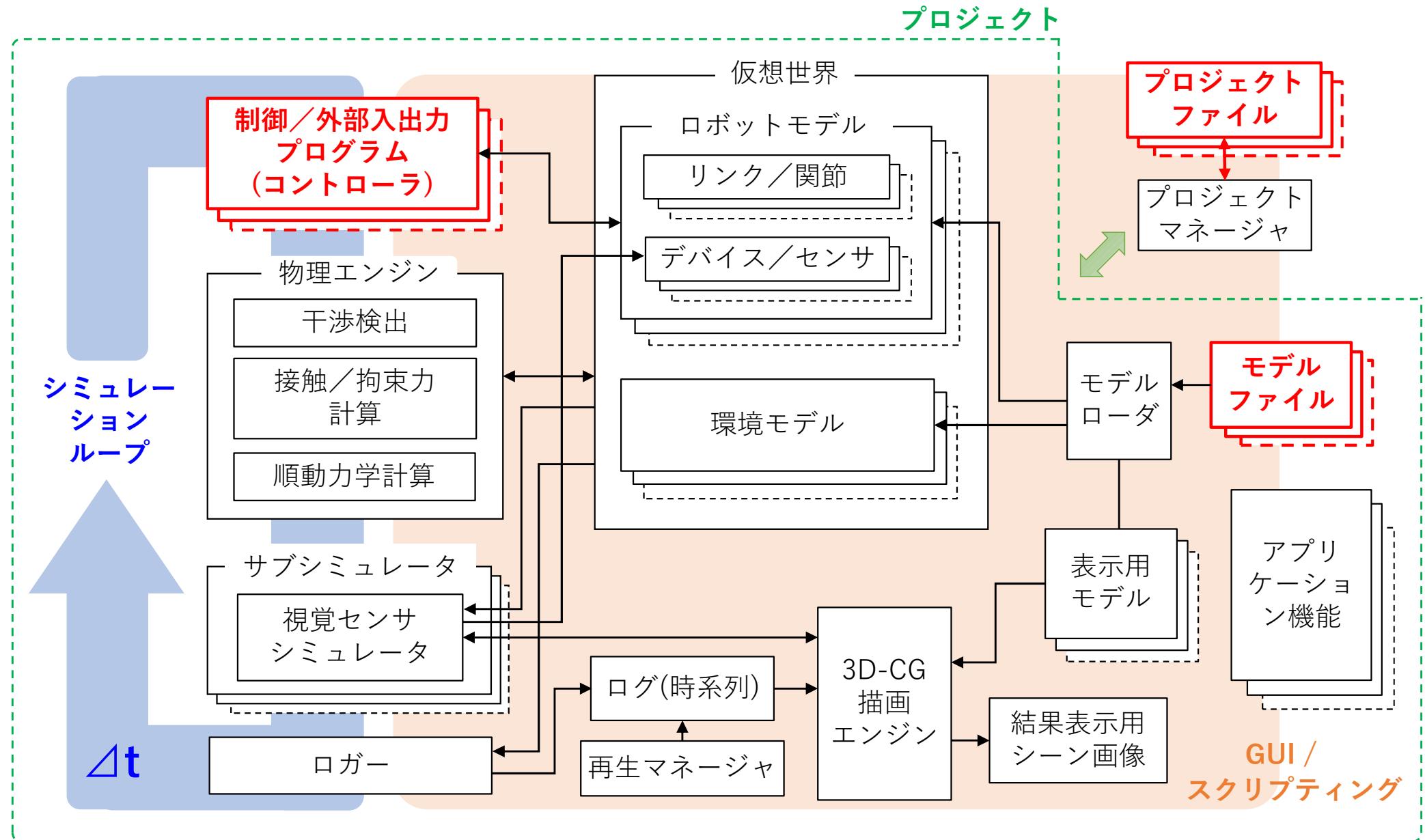
# 課題

- 普及度、知名度
- コミュニティの充実
- マニュアル、掲示板、Githubリポジトリ等は用意しているが…
- リリース版
- バイナリパッケージ／インストーラ
- YouTube解説動画
- ROS対応機能は発展途上

# シミュレーションするにあたって

- 用意しなければならないもの
  - モデル
  - 入出力／制御プログラム
- 必要な情報
  - モデル記述形式
  - 入出力API

# シミュレータの構成



# チュートリアル（実演）

- モバイルロボットのシミュレーション
  - モデルの作成
  - ROS通信を用いた制御と可視化
  - センサの導入
  - 遠隔操作
- アンケート
  - ROSのことはよく分かっていない
  - ROSのトピックとかの基本的なところは分かるけどまだ初心者
  - 仕事や趣味でROSを活用している中級者～上級者

# 情報

- Choreonoid公式サイト
  - <https://choreonoid.org/>
- マニュアル
  - Choreonoid 最新版（開発版）マニュアル
    - Tankチュートリアル、ROS版Tankチュートリアル
- ソースコード
  - <https://github.com/choreonoid/choreonoid>
  - [https://github.com/choreonoid/choreonoid\\_ros](https://github.com/choreonoid/choreonoid_ros)
  - チュートリアルのリポジトリ（準備中）
  - master最新版（昨日更新）を使用
- 掲示板
  - <https://discourse.choreonoid.org/>

# インストール（ビルド）

- masterブランチの最新版をビルドして利用するのがおすすめ
- Ubuntuで通常のビルドを行うのは比較的簡単

```
sudo apt install git
git clone https://github.com/choreonoid/choreonoid.git
cd choreonoid
./misc/script/install-requisites-ubuntu-20.04.sh
cmake -B build; cmake --build build --parallel
```

※ make install しなくてもビルドディレクトリのバイナリをそのまま実行できます

# ROS環境へのインストール

- ROSパッケージとしてビルド
  - choreonoid (本体)
  - choreonoid\_ros (ROSプラグイン)
  - Catkinワークスペース上にソースを展開してビルド

※ マニュアルの「ROSとの連携」 - 「Choreonoid関連パッケージのビルド」参照

<https://choreonoid.org/ja/documents/latest/ros/build-choreonoid.html>

# Choreonoid ノードの起動

- ROS ノードとしてChoreonoidを起動

端末1

```
roscore
```

端末2

```
rosrun choreonoid_ros choreonoid
```

# シミュレーション用パッケージの作成

- パッケージ雛形の作成

```
cd catkin_ws/src  
catkin create pkg rosjp_ug47
```

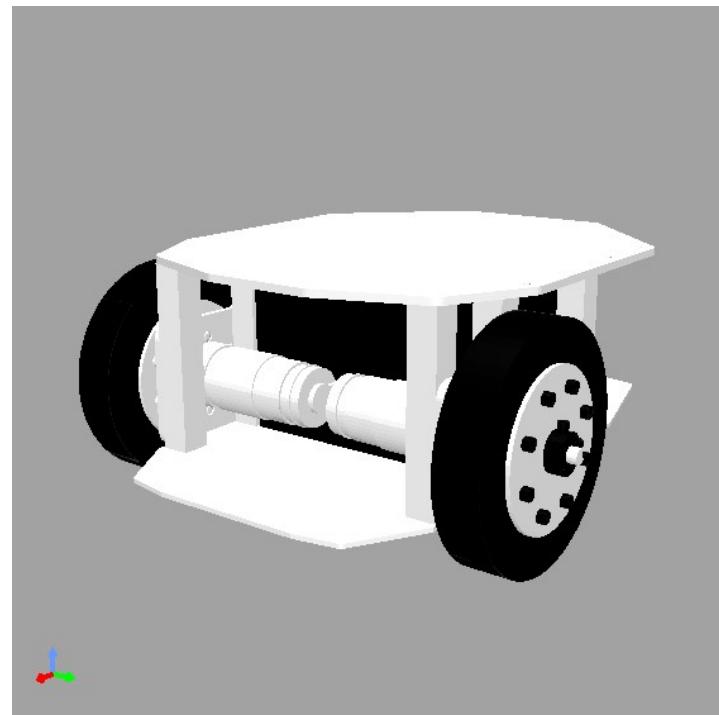
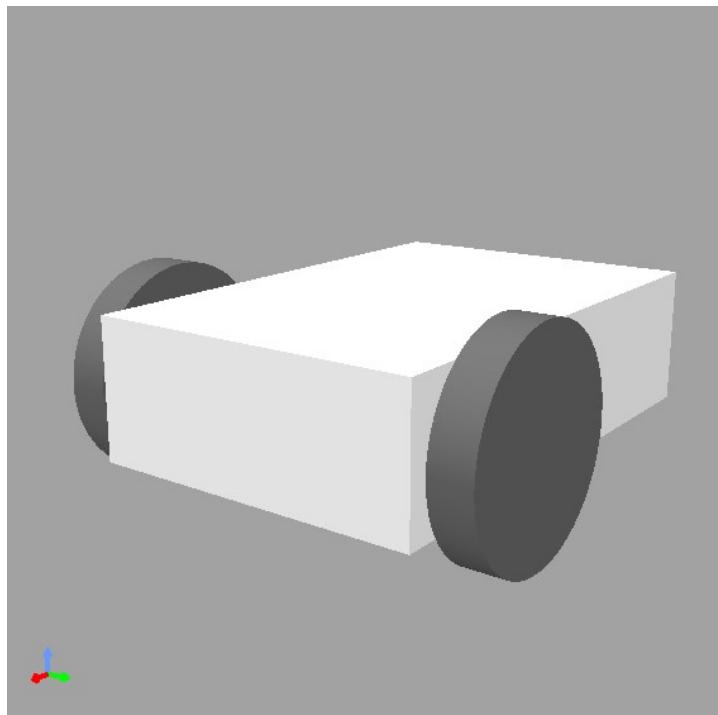
※ Githubにアップしたリポジトリは  
“choreonoid\_ros\_mobile\_robot\_tutorial”  
としています。

- package.xmlの編集

```
...  
<depend>choreonoid</depend>  
<depend>choreonoid_ros</depend>  
...  
<export>  
  <build_type>cmake</build_type>  
</export>  ※ デフォルトでは "catkin"  
...
```

# モデルの作成

- ・車輪型モバイルロボットを作る



※ ヴィストン社 メガローバー2.1

# モデルファイル（記述形式）

- Body形式
  - Choreonoidネイティブ
  - YAMLで記述
- URDF
  - ROS標準
  - XMLで記述
- Xacro
  - XMLマクロ言語、ROS標準
  - URDFの記述を効率化

# 車体の記述 (Body形式)

```
format: ChoreonoidBody
formatVersion: 1.0
angleUnit: degree
name: MobileRobot
rootLink: Chassis

links:
-
  name: Chassis
  joint_type: free
  center_of_mass: [ -0.061, 0, -0.0020 ]
  mass: 14.0
  inertia: [ 0.11, 0, 0,
              0, 0.20, 0,
              0, 0, 0.27 ]
  material: Slider
  elements:
  -
    type: Shape
    translation: [ -0.1, 0, 0.08 ]
    geometry:
      type: Box
      size: [ 0.36, 0.24, 0.1 ]
  -
    type: Shape
    translation: [ -0.255, 0, 0.02 ]
    geometry:
      type: Cylinder
      height: 0.01
      radius: 0.02
```

# モデルの読み込み

- 「ファイル」 – 「読み込み」 – 「ボディ」
- リロード機能
  - コンテキストメニューの「再読み込み」
  - 選択状態でCtrl + R

# 車輪の追加

```
-  
  name: RightWheel  
  parent: Chassis  
  translation: [ 0, -0.142, 0.075 ]  
  joint_type: revolute  
  joint_id: 0  
  joint_axis: [ 0, 1, 0 ]  
  center_of_mass: [ 0, 0, 0 ]  
  mass: 0.8  
  inertia: [ 0.01, 0, 0,  
             0, 0.02, 0,  
             0, 0, 0.01 ]  
  material: Tire  
  elements:  
    - &TireShape  
      type: Shape  
      geometry:  
        type: Cylinder  
        height: 0.03  
        radius: 0.076  
        division_number: 60  
      appearance:  
        material:  
          diffuseColor: [ 0.4, 0.4, 0.4 ]
```

- リロード(Ctrl + R)して更新
- 左側の車輪も同様に追加

※ 完成版はリポジトリの  
"robots/mobile\_robot.body"

# モデルの操作

- ・シーンビューを編集モードに
- ・車体の移動・回転
- ・車輪の回転
- ・関節変位ビュー上のスライダ（ダイアル）操作
- ・チェックで表示／非表示

# シミュレーションの実行

- 以下のアイテムを追加
  - ワールド
  - AISTシミュレータ

```
+ World
  + MobileRobot
  + AISTSimulator
```

※ 親子関係に注意！

- シミュレーション開始ボタンを押す
- モデルが落下する
  - cf. 「重力加速度」プロパティを"0 0 0"にしてみる

# なぜWorldもSimulatorもアイテム？

- World
  - ひとつのChoreonoid上で複数の仮想世界を持つ
    - 仮想世界でシミュレーション設定を変える
    - 複数シミュレーションを同時に実行する
    - シミュレーション結果を重ねて表示・比較する
- Simulator
  - 物理エンジンの実装を切り替えられる（追加できる）
    - ODE、Bullet、PhysX、AGX、SpringHead、etc.
    - 物理計算に関する複数の設定を持つ

# 参考: 物理エンジンODEの利用

- CMakeで “BUILD\_ODE\_PLUGIN” をONにして  
再ビルド

```
catkin config --cmake-args -DBUILD_ODE_PLUGIN=ON  
catkin build
```

- 「ファイル」 - 「新規」 - 「ODEシミュレータ」
  - World以下に配置
- “ODESimulator”アイテムを選択してシミュレー  
ションを実行

# 床の追加

- 床モデルを読み込む
  - choreonoid-1.8/model/misc/floor.body

```
+ World
  + MobileRobot
  + Floor
  + AISTSimulator
```

- 落ちなくなる
- 車体をドラッグして引っ張ってみる

# メッシュファイルの利用

```
- name: Chassis
  joint_type: free
  center_of_mass: [ -0.061, 0, -0.0020 ]
  mass: 14.0
  inertia: [ 0.11, 0, 0,
              0, 0.20, 0,
              0, 0, 0.27 ]
  material: Slider
  elements:
    -
      type: Resource
      uri: “..../meshes/vmega_body.dae”
-
```

```
- name: RightWheel
  parent: Chassis
  translation: [ 0, -0.142, 0.075 ]
  joint_type: revolute
  joint_id: 0
  joint_axis: [ 0, 1, 0 ]
  center_of_mass: [ 0, 0, 0 ]
  mass: 0.8
  inertia: [ 0.01, 0, 0,
              0, 0.02, 0,
              0, 0, 0.01 ]
  material: Tire
  elements:
    -
      type: Resource
      uri: “..../meshes/vmega_wheel.dae”
```

uri: “package://パッケージ名/meshes/vmega\_body.dae” と書いてもOK!

※ 完成版はリポジトリの “robots/mobile\_robot\_mesh.body”

[https://github.com/vstoneofficial/megarover\\_samples](https://github.com/vstoneofficial/megarover_samples)

のメッシュファイルを利用

# URDF版 (1/2)

```
<?xml version="1.0" ?>

<robot name="MobileRobot">

  <link name="Chassis">
    <visual>
      <geometry>
        <mesh filename="package://rosjp_ug47/robots/vmega_body.dae"/>
      </geometry>
    </visual>
    <collision>
      <geometry>
        <mesh filename="package://rosjp_ug47/robots/vmega_body.dae"/>
      </geometry>
    </collision>
    <inertial>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <mass value="14.0"/>
      <inertia ixx="0.11" ixy="0" ixz="0" iyy="0.20" iyz="0" izz="0.27"/>
    </inertial>
  </link>

  <link name="RightWheel">
    <visual>
      <geometry>
        <mesh filename="package://rosjp_ug47/robots/vmega_wheel.dae"/>
      </geometry>
    </visual>
    <collision>
      <geometry>
        <mesh filename="package://rosjp_ug47/robots/vmega_wheel.dae"/>
      </geometry>
    </collision>
    <inertial>
      <mass value="0.8"/>
      <inertia ixx="0.01" ixy="0" ixz="0" iyy="0.02" iyz="0" izz="0.01"/>
    </inertial>
  </link>
```

# URDF版 (2/2)

```
<link name="LeftWheel">
  <visual>
    <geometry>
      <mesh filename="package://rosjp_ug47/robots/vmega_wheel.dae"/>
    </geometry>
    <origin xyz="0 0 0" rpy="0 0 3.14159265"/>
  </visual>
  <collision>
    <geometry>
      <mesh filename="package://rosjp_ug47/robots/vmega_wheel.dae"/>
    </geometry>
    <origin xyz="0 0 0" rpy="0 0 3.14159265"/>
  </collision>
  <inertial>
    <mass value="0.8"/>
    <inertia ixx="0.01" ixy="0" ixz="0" iyy="0.02" iyz="0" izz="0.01"/>
  </inertial>
</link>

<joint name="RightWheel" type="continuous">
  <parent link="Chassis"/>
  <child link="RightWheel"/>
  <origin xyz="0 -0.142 0.075" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
</joint>

<joint name="LeftWheel" type="continuous">
  <parent link="Chassis"/>
  <child link="LeftWheel"/>
  <origin xyz="0 0.142 0.075" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
</joint>

</robot>
```

※ 完成版はリポジトリの  
"robots/mobile\_robot.urdf"

# URDFファイルの読み込み

- 「ファイル」 — 「読み込み」 — 「ボディ」
- ダイアログ下部の「ファイルの種類」コンボ  
ボックスで「URDF」を選択
- URDF (xacro) ファイルを選択

# Bodyファイル vs URDF

- Choreonoid読込用としてはBodyが望ましい
  - 記述しやすく読みやすい
  - Choreonoid上の要素を全て記述できる
- URDFからBodyへの変換
  - URDFで読み込んで「ファイル」 – 「選択アイテムに名前を付けて保存」
  - ただしメッシュデータは展開される
- BodyからURDFへの変換にも対応させたい

# 入出力API

- 「コントローラアイテム」が提供
- シンプルコントローラアイテム
  - ネイティブAPI
  - C++で自前で実装
  - roscppを用いることでROS通信も可能
- BodyROSアイテム
  - ロボット／センサの状態をROS出力
- ROSControlアイテム
  - ROS標準のros\_controlによる入出力／制御

# ロボット状態のROS出力

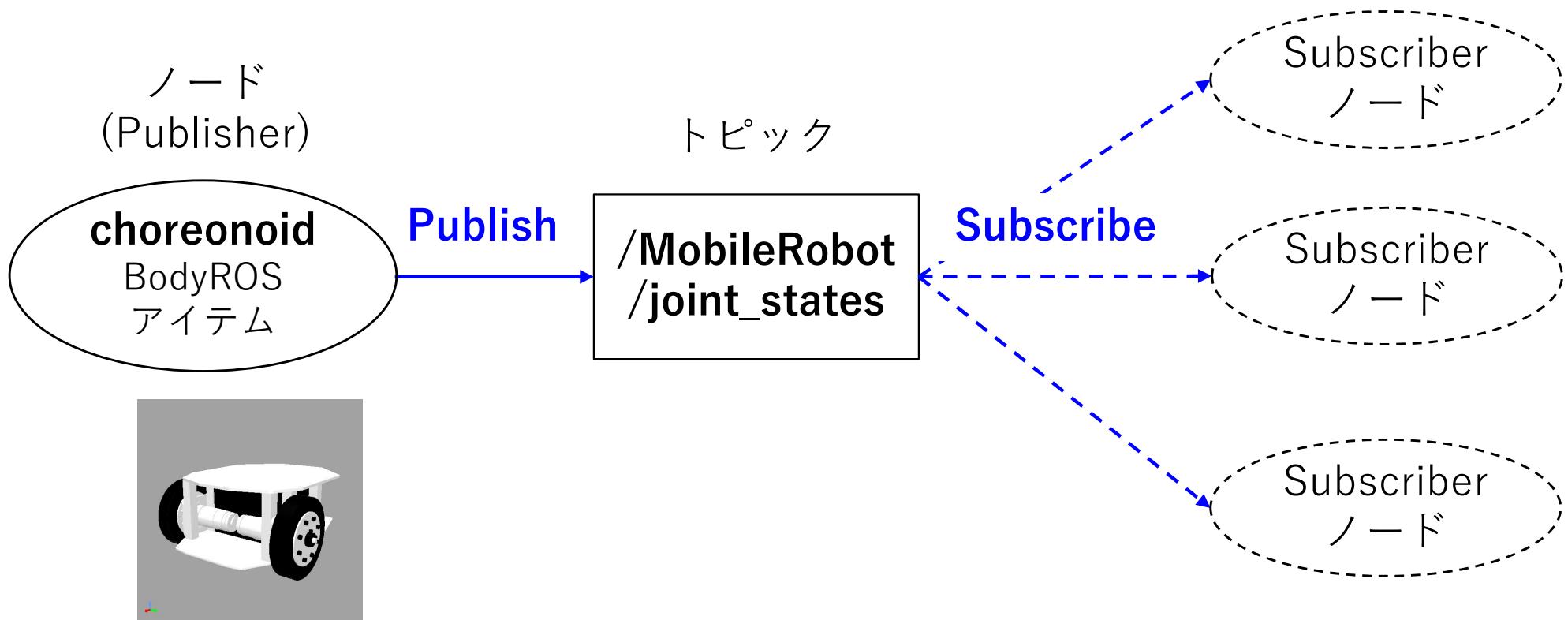
- BodyROSアイテムを追加する

```
+ World
  + MobileRobot
    + BodyROS
  + Floor
  + AISTSimulator
```

- 車軸回転量がROSトピックとしてPublish
  - /MobileRobot/joint\_states

```
rostopic echo /MobileRobot/joint_states
```

# トピック、Publisher、Subscriber



# RVizからSubscribe

“launch/display.launch”

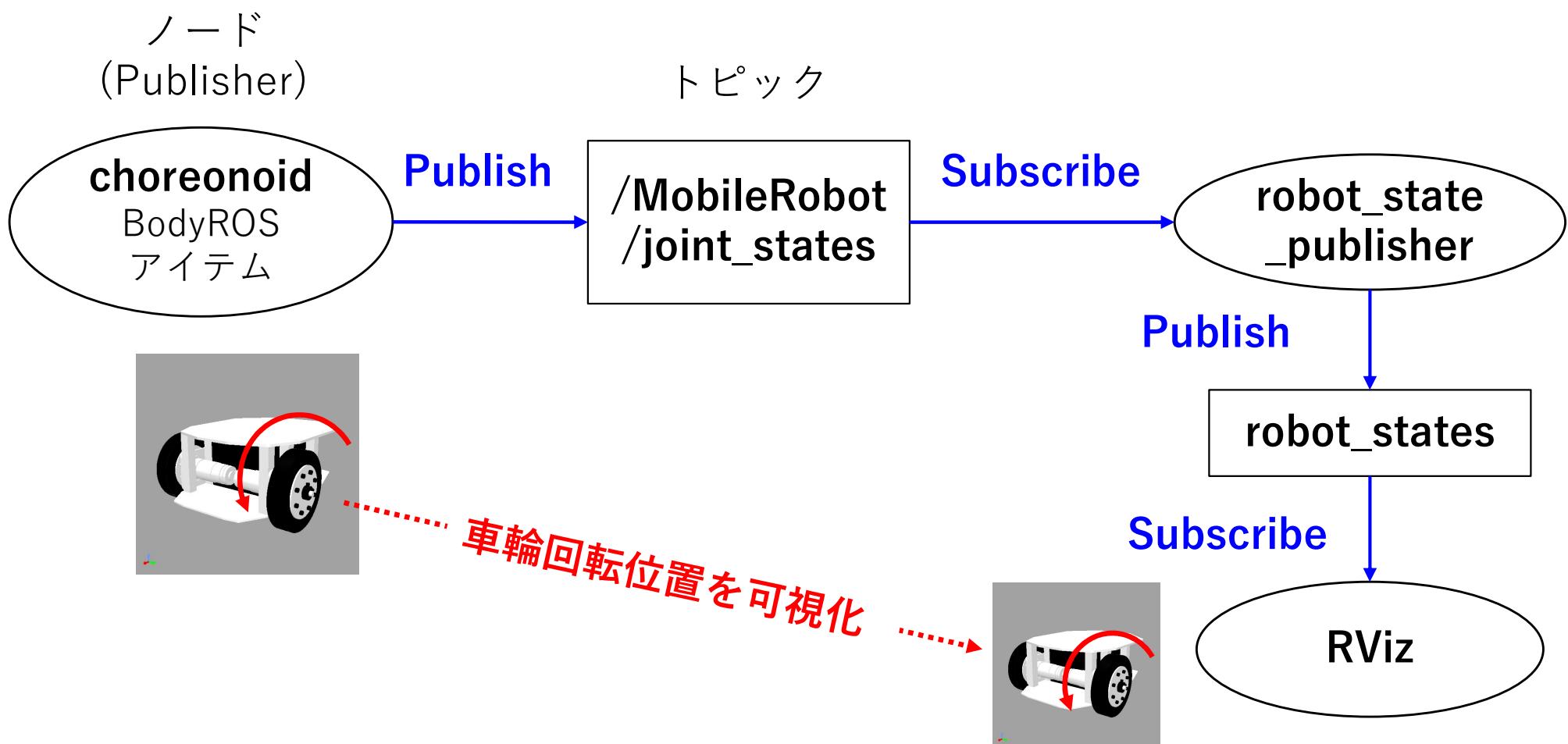
※ Githubにアップしたリポジトリではパッケージが  
“choreonoid\_ros\_mobile\_robot\_tutorial”となります。

```
<launch>
  <arg name="model" default="$(find rosjp_ug47)/robots/mobile_robot.urdf"/>
  <arg name="rvizconfig" default="$(find rosjp_ug47)/config/mobile_robot.rviz" />
  <param name="robot_description" command="$(find xacro)/xacro $(arg model)" />
  <remap from="/joint_states" to="/MobileRobot/joint_states" />
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)" required="true" />
  <node pkg="rqt_graph" name="rqt_graph" type="rqt_graph" />
</launch>
```

以下を実行

```
roslaunch rosjp_ug47 display.launch
```

# トピック、Publisher、Subscriber



# 仮想世界全体情報のPublish

- WorldROSアイテムを追加する

- + World
  - + MobileRobot
  - + BodyROS
- + Floor
- + AISTSimulator
- + **WorldROS**

# Clock トピック

- 他のROSノードと時刻を共有（同期）
- WorldROSアイテムがPublish
- rosgraph\_msgs/Clock型の/clock トピック
- ROSパラメータ /use\_sim\_time を true に

```
rosparam set /use_sim_time true
```

または

```
<launch>
  <param name="/use_sim_time" value="true" />
  ...
</launch>
```

# 制御の仕方

- モデルにコントローラアイテムを紐付ける
- シンプルコントローラアイテム
  - C++(roscpp)を用いて自分で実装
- ROSControlアイテム
  - ros\_controlのモジュールを利用

# シンプルコントローラの導入

- ・該当アイテムを追加

```
+ World
  + MobileRobot
    + BodyROS
    + SimpleController
  + Floor
  + AISTSimulator
  + WorldROS
```

- ・「コントローラモジュール」プロパティでコントローラ本体を指定

# コントローラ本体の作成

- src/MobileRobotController.cppを作成
- ビルド用のCMakeLists.txtも作成
- catkin buildでビルド
- MobileRobotController.so が生成される

「コントローラモジュール」プロパティに設定

# 操作指令値: Twist メッセージ型

- geometry\_msgs/Twist
- 速度 + 角速度

```
rosmsg show geometry_msgs/Twist
```

```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

# クラス定義

```
#include <cnoid/SimpleController>
#include <ros/node_handle.h>
#include <geometry_msgs/Twist.h>
#include <mutex>

using namespace std;
using namespace cnoid;

class MobileRobotController : public SimpleController
{
    std::unique_ptr<ros::NodeHandle> node;
    ros::Subscriber subscriber;
    geometry_msgs::Twist command;
    std::mutex commandMutex;
    Link* wheels[2];
    double dq_target[2];

    virtual bool initialize(SimpleControllerIO* io) override;
    void commandCallback(const geometry_msgs::Twist& twist);
    virtual bool control() override;
};

CNOID_IMPLEMENT_SIMPLE_CONTROLLER_FACTORY(MobileRobotController)
```

# 初始化／通信 (Subscribe) 处理

```
bool MobileRobotController::initialize(SimpleControllerIO* io)
{
    auto body = io->body();
    wheels[0] = body->joint("RightWheel");
    wheels[1] = body->joint("LeftWheel");

    for(int i=0; i < 2; ++i){
        auto wheel = wheels[i];
        wheel->setActuationMode(JointTorque);
        io->enableInput(wheel, JointVelocity);
        io->enableOutput(wheel, JointTorque);
    }

    node = make_unique<ros::NodeHandle>(body->name());
    subscriber = node->subscribe(
        "/cmd_vel", 1, &MobileRobotController::commandCallback, this);

    return true;
}

void MobileRobotController::commandCallback(const geometry_msgs::Twist& msg)
{
    std::lock_guard<std::mutex> lock(commandMutex);
    command = msg;
}
```

# 車輪の制御（目標角速度に基づくトルク指令）

```
bool MobileRobotController::control()
{
    constexpr double wheelRadius = 0.076;
    constexpr double halfAxeWidth = 0.142;
    constexpr double kdd = 5.0;
    double dq_target[2];

    {
        std::lock_guard<std::mutex> lock(commandMutex);
        double dq_x = command.linear.x / wheelRadius;
        double dq_yaw = command.angular.z * halfAxeWidth / wheelRadius;
        dq_target[0] = dq_x - dq_yaw;
        dq_target[1] = dq_x + dq_yaw;
    }

    for(int i=0; i < 2; ++i) {
        auto wheel = wheels[i];
        wheel->u() = kdd * (dq_target[i] - wheel->dq());
    }

    return true;
}
```

# rostopicコマンドによる操作

```
rostopic pub -1 /cmd_vel geometry_msgs/Twist  
'{ linear: { x: 0.5, y: 0, z: 0 }, angular: { x: 0, y: 0, z: 0 } }'
```

前後方向速度

旋回角速度

※ コロン(:)の後のスペースに注意！

# ゲームパッドによる操作

- joyトピックをTwist型に変換

launch/teleop\_twist\_joy.launch

```
<launch>
  <node pkg="joy" type="joy_node" name="joy" respawn="true">
    <param name="autorepeat_rate" value="20" />
  </node>
  <node pkg="teleop_twist_joy" type="teleop_node"
        name="teleop_node" output="screen">
    <param name="scale_angular" value="-2.0" />
  </node>
</launch>
```

PS5のDualSenseコントローラ（ゲームパッド）の場合は  
×ボタンを押しながら左アナログスティックを操作

# 視覚センサの追加

```
-  
  type: Transform  
  translation: [ 0, 0, 0.2 ]  
  elements:  
    -  
      type: RangeSensor          取付位置  
      name: VLP_16  
      optical_frame_rotation: [ [ 1, 0, 0, 90 ], [ 0, 1, 0, -90 ] ]  
      yaw_range: 360.0  
      yaw_step: 0.4  
      pitch_range: 30.0  
      pitch_step: 2.0  
      scan_rate: 20  
      max_distance: 100.0  
      detection_rate: 0.9  
      error_deviation: 0.01  
  
    -  
      type: Camera                LiDAR  
      name: RealSense  
      optical_frame_rotation: [ [ 1, 0, 0, 90 ], [ 0, 1, 0, -90 ] ]  
      format: COLOR_DEPTH  
      field_of_view: 62  
      width: 320  
      height: 240  
      frame_rate: 30  
      detection_rate: 0.9  
      error_deviation: 0.005  
  
      -  
        type: Transform  
        translation: [ 0, 0, 0.2 ]  
        elements:  
          -  
            type: RangeSensor  
            name: VLP_16  
            optical_frame_rotation: [ [ 1, 0, 0, 90 ], [ 0, 1, 0, -90 ] ]  
            yaw_range: 360.0  
            yaw_step: 0.4  
            pitch_range: 30.0  
            pitch_step: 2.0  
            scan_rate: 20  
            max_distance: 100.0  
            detection_rate: 0.9  
            error_deviation: 0.01  
  
          -  
            type: Camera  
            name: RealSense  
            optical_frame_rotation: [ [ 1, 0, 0, 90 ], [ 0, 1, 0, -90 ] ]  
            format: COLOR_DEPTH  
            field_of_view: 62  
            width: 320  
            height: 240  
            frame_rate: 30  
            detection_rate: 0.9  
            error_deviation: 0.005  
  
※ 完成版はリポジトリの  
"robots/mobile_robot_sensors.body"
```

# 視覚センサシミュレーション

- GLビジョンシミュレータアイテムを追加

```
+ World
  + MobileRobot
    + BodyROS
    + SimpleController
  + Floor
  + AISTSimulator
    + GLVisionSimulator
  + WorldROS
```

# RVizによるセンサ情報の可視化

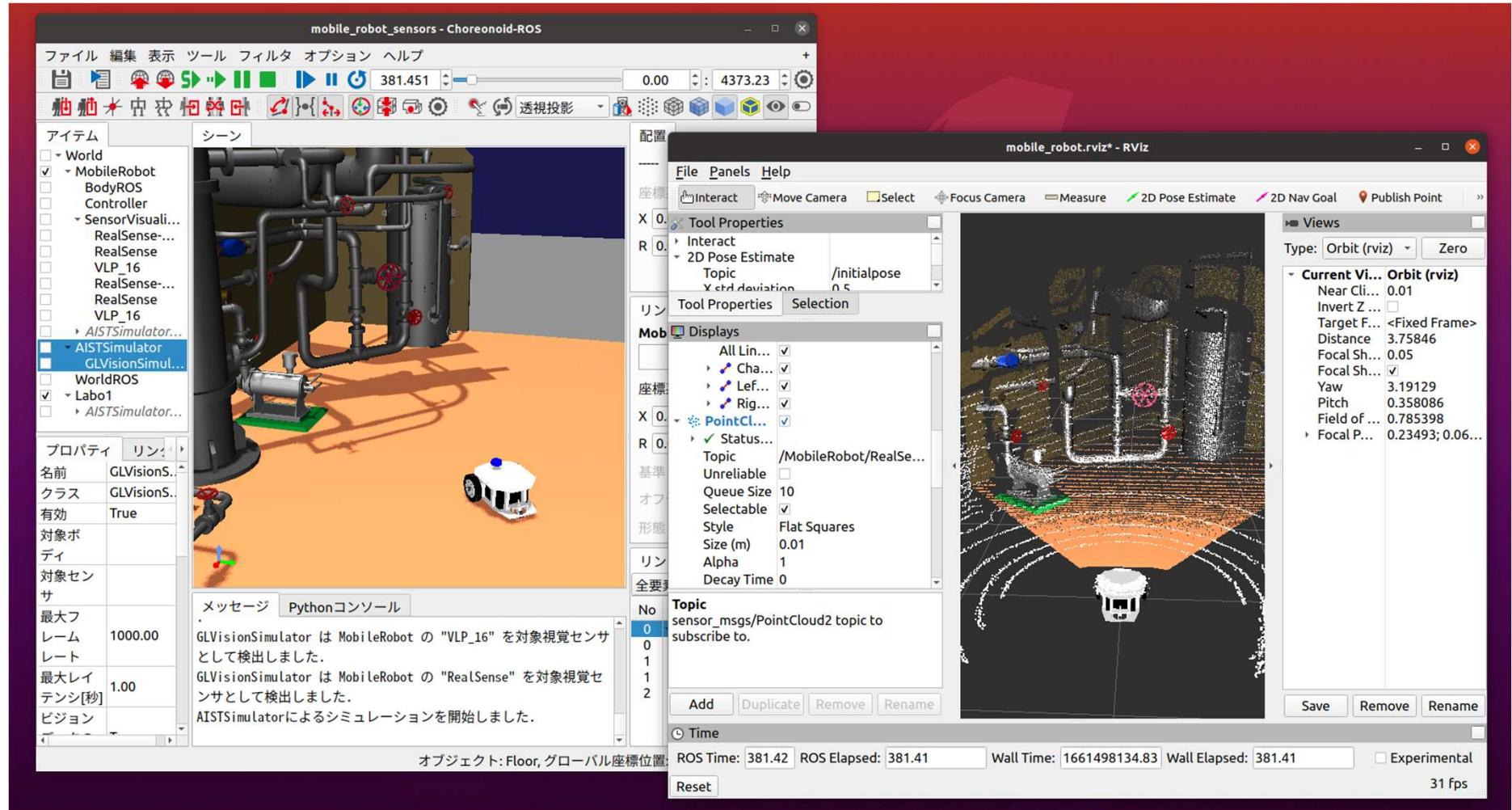
launch/display\_sensors.launch

```
<launch>
  <arg name="model" default="$(find rosjp_ug47)/robots/mobile_robot.urdf"/>
  <arg name="rvizconfig" default="$(find rosjp_ug47)/config/mobile_robot.rviz" />
  <param name="robot_description" command="$(find xacro)/xacro $(arg model)" />
  <remap from="/joint_states" to="/MobileRobot/joint_states" />
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />

  <node pkg="tf" type="static_transform_publisher" name="VLP_16_broadcaster" args="0 0 0.2 0 0 0 Chassis VLP_16 100" />
  <node pkg="tf" type="static_transform_publisher" name="RealSense_broadcaster" args="0 0 0.2 0 0 0 Chassis RealSense 100" />

  <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig) " required="true" />
  <node pkg="rqt_graph" name="rqt_graph" type="rqt_graph" />
</launch>
```

# 遠隔操作の実現



# 高度な使用例（デモ）

- World Robot Summit 災害対応競技

