Torobo Arm ROS
ユーザーマニュアル

Ver. 0.0.2

東京ロボティクス株式会社
http: //robotics.tokyo

目次

1. はじめに

このマニュアルは Torobo Arm を ROS 環境で使用する方法を説明しています。このマニュアルをよくお読みになり、内容を十分理解された上で操作を行っていただくようお願いいたします。

1.1. ROS とは？

Robot Operation System(ROS)はロボット開発のためのフレームワークです。Operation System という名前がついていますが、実際にはミドルウェア、ビルド環境、パッケージ間の依存関係管理等が一体となった開発環境となっています。現在は Open Source Robotics Foundation(OSRF)によって管理されており、世界で最も多く使用されているロボット開発のためのフレームワークとなっています。多くの研究成果が ROS 用のパッケージとして公開されており、Torobo Arm の ROS パッケージをこれらと組み合わせて利用することで複雑なロボットアプリケーションを用意に構築することができます。

ROS そのものの概念については、ROS 公式の Wiki や Tutorial を参照してください。

1.2. Torobo Arm ROS で使用するハードウェア

Torobo Arm を ROS 環境で利用する場合、下記の環境が必要となります。

	**Torobo Arm 本体および付属物品** ToroboArm を Windows 環境で使用する場合と同様に、下記を用意してください。 ・ 本体 ・ マスターコントローラ ・ 非常停止スイッチ ・ アーム固定用台座 ・ 電源 ・ USB ケーブル（Type A – Type B）

	**ROS がインストールされた制御用 PC** ROS のインストールされた PC をご自身で用意してください。なお、東京ロボティクスが提供する各種コンポーネントは Ubuntu 14.04LTS に ROS Indigo Igloo をインストールした環境で動作確認を行っています。 なお、Torobo Arm と USB ケーブルで接続するため、USB ポートのある PC を用意してください。また、本マニュアルでは制御用 PC にインターネット接続があることを想定して解説を行っていきます。

※　ROS のインストール手順は公式 Wiki をご確認下さい。
http://wiki.ros.org/ja/indigo/Installation/Ubuntu

1.3.　Torobo Arm ROS で使用するソフトウェア

Torobo Arm を Linux から利用するためには、下記のドライバと制御用ソフトウェアが必要です。

Linux 用ドライバ及びコマンドラインツール

ToroboArmManager	Torobo Arm を Linux から利用するためのドライバです。
ToroboArmCommandInterface	Torobo Arm のためのコマンドラインツールです。基本的な制御指令を Linux のターミナル画面から送信することができます。

また、Torobo Arm の ROS 用パッケージは以下の構成要素からなります。

ROS 用パッケージ

toroboarm_robot	ROS 用パッケージを全てまとめたメタパッケージ。 以下の全てのパッケージが含まれます。
toroboarm_bringup **or toroboarm_seven_bringup**	Torobo Arm 関連コンポーネントの起動手順をまとめたパッケージ。
toroboarm_control **or toroboarm_seven_control**	Torobo Arm の制御設定をまとめたパッケージ。
toroboarm_desctiption **or toroboarm_seven_description**	Torobo Arm の構成モデル情報がまとめられたパッケージ。

toroboarm_driver **or toroboarm_seven_driver**	ToroboArmManager と ROS 環境の接続用ソフトウェア。
toroboarm_gazebo **or toroboarm_seven_gazebo**	Torobo Arm を物理シミュレータ Gazebo で使用するための各種設定をまとめたパッケージ。Gazebo については後ほど詳説します。
toroboarm_moveit_config **or** **toroboarm_seven_ moveit_config**	軌道計画ソフトウェア MoveIt!で Torobo Arm を動かすための各種設定をまとめたパッケージ。MoveIt!については後ほど詳説します。

全てのソフトウェアは東京ロボティクスの公開リポジトリから取得することができます。

1.4. 本マニュアルの構成

本マニュアルは、以下のような構成をとります。

基本事項：1 章、2 章、5 章

ロボットアームやマニュアルの基本事項について説明しています。

Linux で使うための基本設定：3 章

Torobo Arm を Linux で使用するために必要なソフトウェアの導入方法や使用方法について説明しています。

ROS パッケージを使う：4 章

Torobo Arm のために用意された各種 ROS パッケージの導入方法や使用方法について説明しています。

2. 安全注意事項

2.1. 表示の説明

本マニュアルでは安全に関する内容により下記のシンボルマークを使用しています。安全に関するシンボルマークのある記述は、重要な内容を記載しておりますので必ずお読みになった上で記載事項を順守していただくようお願いします。また、製品にも同様の基準でシンボルマークが貼付されています。

	**危険** 取扱いを誤った場合に、危険な状況が起こりえて、死亡または重症を受けるなどの重大事故につながる際の表記。
	**危険** 取扱いを誤った場合に、感電等の電気的に危険な状況が起こりえて、死亡または重症を受けるなどの重大事故につながる際の表記。
	**警告** 取扱いを誤った場合に、危険な状況が起こりえて、死亡または重症を受けるなどの重大事故につながる可能性がある場合の表記。
	**警告** 取扱いを誤った場合に、感電等の電気的に危険な状況が起こりえて、死亡または重症を受けるなどの重大事故につながる可能性がある場合の表記。
	**警告** 表面が高温になっており、触ると火傷などの事故につながる可能性がある場合の表記。
	**注意** 取扱いを誤った場合に、怪我や事故につながる可能性がある場合の表記。 ※この場合も。状況によっては重大な事故に結びつく可能性がありますので、必ずお守り下さい。
	**要件・注記** 操作に必要な指示、取扱いする上で参考すべき注意表記。

2.2. ソフトウェアの安全性について（重要）

Torbo Arm は、初期実装として弊社が提供するサンプルプログラム（制御 PC およびマスターコントローラ）によって動作します。**サンプルプログラムは、ロボットアームユーザーの利便性を向上させるためのものであり、安全な動作を保証するものではありません。**そのため、ロボットアームを動作させる際は、必ずアームの可動域の外に出て行うようにお願いします。人とロボットの協働など、ロボットとの物理的接触を前提にする場合は、安全な環境で十分にテストを行った上で、ユーザーご自身の責任において行って下さい。

危険

サンプルプログラムの予期せぬ不具合、想定されてない使用方法などで、ロボットアームが暴走する恐れがあります。

ユーザーの責任のもと、人とロボットの物理的接触を前提としたタスクを設定する場合は、必ず非常停止ボタンを瞬時に押せる状態にしておいて下さい。

2.3. 禁止事項

ロボットアーム・マスターコントローラともに分解はしないで下さい。分解された場合は、有償・無償ともに修理の対象外とさせていただきます。

警告

ロボットアームやマスターコントローラを分解すると、感電の危険があります。有償・無償ともに修理の対象外ともなりますので、分解はお控え下さい。

2.4. 免責事項

当社が提供するサンプルプログラムによって生じたいかなる損害に関しても、当社は一切責任を負いません。

当社の責任以外の火災、第三者による行為、その他の事故・お客様の故意または過失、誤用、その他の異常な条件下での使用により生じた損害に関して、当社は一切責任を負いません。

本製品の使用または使用不能から生ずる付随的な損害（事業利益の損失・事業の中断など）に関して、当社は一切責任を負いません。

本マニュアルの記載内容を守らないことにより生じた損害に関して、当社は一切責任を負いません。

3. Linux 用ドライバ及びコマンドラインツール

3.1. 全体構成

別紙「Torobo Arm マニュアル」では、Master Controller を Windows PC に接続して使用する場合の方法について「3.まずは動かしてみよう」で解説しています。Torobo Arm を ROS 環境で使用するためには、Linux(Ubuntu)に接続するため、専用のソフトウェアを導入する必要があります。下図に関連ソフトウェアの構成と役割について示します。

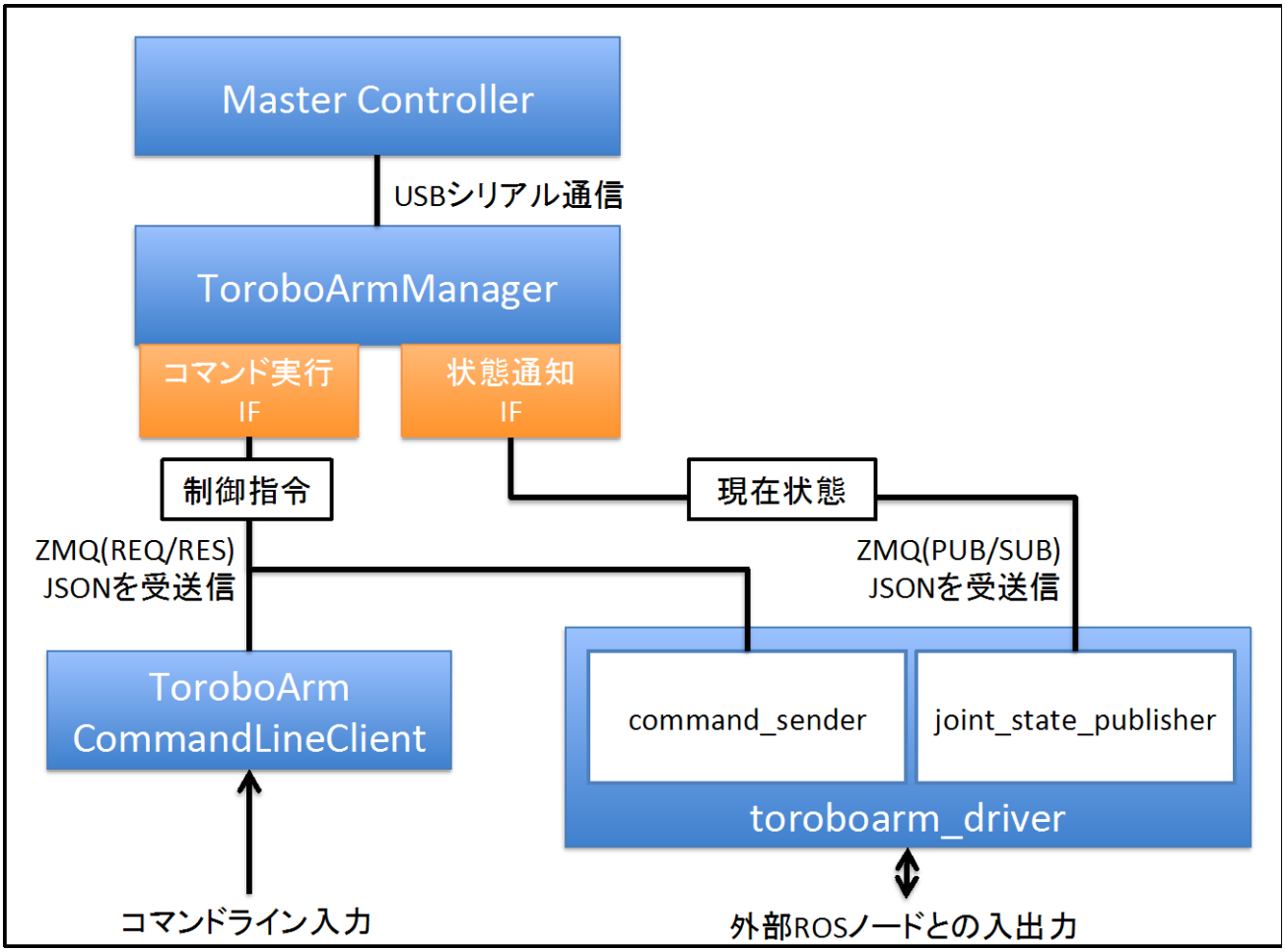

FIg 1 各ソフトウェアの通信構成

Master Controller と Ubuntu PC は Windows の場合と同様に USB ケーブルで接続します。Master

Controller と通信して各種制御指令を実行したり、現在状態を確認したりするドライバの役割を果すのが ToroboArmManager です。ToroboArmManager には以下の３つの通信インタフェースが用意されています。

（１） MasterController との通信インタフェース
こちらの仕様は MasterController のアプリの仕様に準拠していますので、「Torobo Arm マニュアル」の「8.3.1 制御用 PC とマスターコントローラの通信」をご確認下さい。

（２） コマンド実行インタフェース
Linux マシン上の外部アプリからコマンド実行指令を受け取るためのインタフェースです。通信は ZeroMQ の REQ/RES 方式で実装されており、特定の書式の JSON を受け取るとその内容に応じたコマンドを MasterController に送信します。

（３） 状態通知インタフェース
Linux 上の外部アプリへマシンの現在状態を配信する通信インタフェースです。通信は ZeroMQ の PUB/SUB 方式で実装されており、クライアント側から購読要求があると、現在状態を記述した JSON を一定周期で配信します。

この ToroboArmManager に対して Linux 上からコマンドを送信することのできるサンプルアプリケーションとして、コマンドラインツールとして利用が可能な ToroboArmCommandLineClient を用意しました。ToroboArmManager と合わせてこちらのソフトウェアをインストールすることで、ターミナル画面上から Linux コマンドを入力する要領で Torobo Arm を制御することが可能になります。

本章ではこれらのツールのインストール方法について解説するとともに、Linux 上での基本的な動作確認方法について解説します。

3.2. インストール方法

3.2.1. ToroboArmManager

依存パッケージを下記のコマンドでインストールします。

```
$ sudo apt-get install libjsoncpp-dev libjsoncpp0 libzmq3-dev
```

次に、ホームディレクトリ以下にソースを配置しビルドして下さい。bin ディレクトリ以下に実行ファイルが出来ていればインストール完了です。ここでは提供されたファイルがホームディレクトリの toroboarmmanager に展開されたことを想定します。

```
$ cd ~/toroboarmmanager
$ vim src/CommonDefine.h
※ここで#define の SLAVE_MCU_NUM および MAX_JOINT_NUM がお使いのアームの軸数と一致しているか確認して下さい。一致していない場合は修正して下さい。
$ make clean
```

```
$ make
```

3.2.2. ToroboArmCommandLineClient

依存パッケージを下記のコマンドでインストールします。

```
$ sudo apt-get install python-pip
$ sudo pip install pyzmq docopt
```

次に、ホームディレクトリ以下にソースを配置して下さい。ここでは提供されたファイルがホームディレクトリの toroboarmcommandlineclient に展開されたことを想定します。

```
$ cd ~/toroboarmcommandlineclient
$ chmod +x armctl move_homepos_*
```

armctl は Python で書かれていますので、コンパイルの必要はありません。move_homepos_6axis.sh および move_homepos_7axis.sh はそれぞれ 6 軸用、7 軸用の armctl コマンドを使用したホームポジション回帰のバッチコマンドです。最後に、実行パスにスクリプトを追加しておきましょう。

```
$ echo “export PATH=~/toroboarmcommandlineclient:$PATH” >> ~/.bashrc
$ source ~/.bashrc
```

3.3. 動作確認

インストールしたソフトウェアを使って、Torobo Arm の接続確認をしてみましょう。マスターコントローラと PC を USB ケーブルで接続した上で、別紙「Torobo Arm マニュアル」の「3.4.1 マスターコントローラの電源を入れる」「3.4.2 書き込みスイッチを確認する」の手順を実行して下さい。

下記のコマンドを実行し、デバイスが接続されたかを確認して下さい。(ttyUSB*が存在していれば、Ubuntu はマスターコントローラと接続されています。)

```
$ ls /dev/ttyUSB*
/dev/ttyUSB0
sudo chmod 666 /dev/ttyUSB0
```

ターミナルを２つ開き、１つの画面で ToroboArmManager を起動します。

```
$ cd ~/toroboarmmanager/bin
$ ./armmng
```

さらにもう１つのターミナルから下記のコマンドを実行してみて下さい。これは、全関節のサーボを ON にするコマンドです。

```
$ armctl --reset --joint=all
$ armctl --servo on --joint=all
```

ToroboArmManager のターミナル画面で状態が変わったことが確認できれば、２つのソフトウェア間での通信が成功しています。また、ToroboArm に触って重力補償制御が正常に動いているか確認しましょう。

3.4. コマンドリファレンス

ToroboArmCommandLineClient から実行可能なコマンドを以下にまとめます。詳しい使用方法は、”armctl -h”で表示されるヘルプ画面を御覧ください。

状態確認	armctl --state
サーボ ON/OFF	・ 全関節サーボ ON armctl --servo ON --joint=all ・ 全関節サーボ OFF armctl --servo OFF --joint=all ・ 特定関節のみ armctl --servo ON --joint=1
全関節サーボオフ	armctl --quit
リセット	armctl --reset --joint=all
ブレーキ ON/OFF	※ブレーキ付アームのみ ・ 全関節ブレーキ ON armctl --brake ON --joint=all ・ 全関節ブレーキ OFF armctl --brake OFF --joint=all ・ 特定関節のみ armctl --brake ON --joint=1
制御モード変更	・ 重力補償＆外力追従モード armctl --mode 5 --joint=all ・ ダイナミクス付軌道制御モード armctl --mode 20 --joint=all
制御ゲイン変更	・ PID ゲインの変更 armctl --kp 0.01 --joint=1 armctl --ki 0.02 --joint=2 armctl --kd 0.05 --joint=3 ・ Dumper の変更 armctl --da 0.01 --joint=all
PID 制御（非推奨）	・ 電流制御（値によっては急激な動き） armctl --cur 0.1 --joint=1 ・ 位置制御（値によっては急激な動き） armctl --pos 0.1 --joint=2 ・ 速度制御（値によっては急激な動き） armctl --vel 0.1 --joint=3

	・ トルク制御（値によっては急激な動き） armctl --tor 0.1 --joint=4
軌道制御	・ 目標姿勢の指定（関節角度・時間を指定） 45 度まで 3 秒で移動（曲線加減速） armctl --tpts 45 3 --joint=1 ・ 目標姿勢の指定（関節角度・時間を指定） 45 度まで 3 秒で移動（台形加減速） armctl --tptl 45 3 --joint=1 ・ 軌道制御の実行 armctl --ts --joint=1 ・ 軌道の削除 armctl --tc --joint=1 ※マイナスの値をセットする場合は符号の代わりに m を付与して下さい（マイナス 30 度：m30）
PVT 制御	・ PVT 点のセット （CSV ファイルを絶対パスで指定） armctl --tpvt /home/torobo/test.csv ・ PVT 制御の実行 armctl --ts --joint=all ・ PVT 点のクリア armctl --tc --joint=all

※ joint には 1~8 または all が指定可能

4. ROS パッケージ

本章では、ROS パッケージのインストール方法について解説した後、実際に動作を見ながら各コンポーネントの役割について学んでいきます。6 軸タイプと 7 軸タイプでは異なるパッケージを使用するため、7 軸タイプをご使用の場合は、以降の説明において、toroboarm_xxx というパッケージ名を適宜 toroboarm_seven_xxx と読み替えて下さい。

4.1. インストール方法

catkin ワークスペースが作成されていない場合、下記のコマンドを実行してみて下さい。

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
$ cd ../
$ catkin_make
```

次に、catkin ワークスペース内にパッケージを配置し、ビルドしましょう。

```
$ cd ~/catkin_ws/src
src 内に提供された ROS パッケージの toroboarm_robot ディレクトリをコピーします。
$ cd ~/catkin_ws
$ rosdep install --from-paths src -iy
$ sudo apt-get install ros-indigo-moveit-full
$ catkin_make
$ source devel/setup.bash
```

エラーが表示されなければ、インストールは完了です。

4.2. ToroboArm の構成モデルを確認する

ROS では、ロボットモデルを Unified Robot Description Format (URDF)という XML 形式ファイルで記述します。このファイルは ROS 上の様々なアプリケーションから読み込まれ、ロボットの状態を管理するのに用いられます。

URDF には各リンクの形状や関節の配置などの構造情報や、ロボットに内蔵されたセンサの種類等が

記述されます。

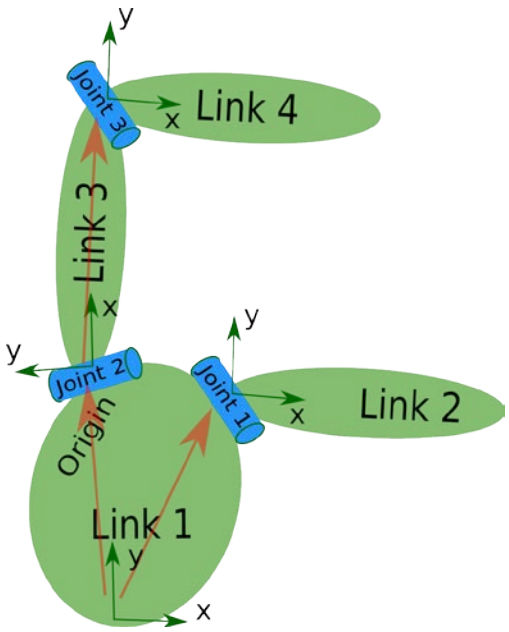

FIg 2 ロボットの構造

（http://wiki.ros.org/urdf/Tutorials/Create%20your%20own%20urdf%20file?action=AttachFile&do=get&target=link.png）

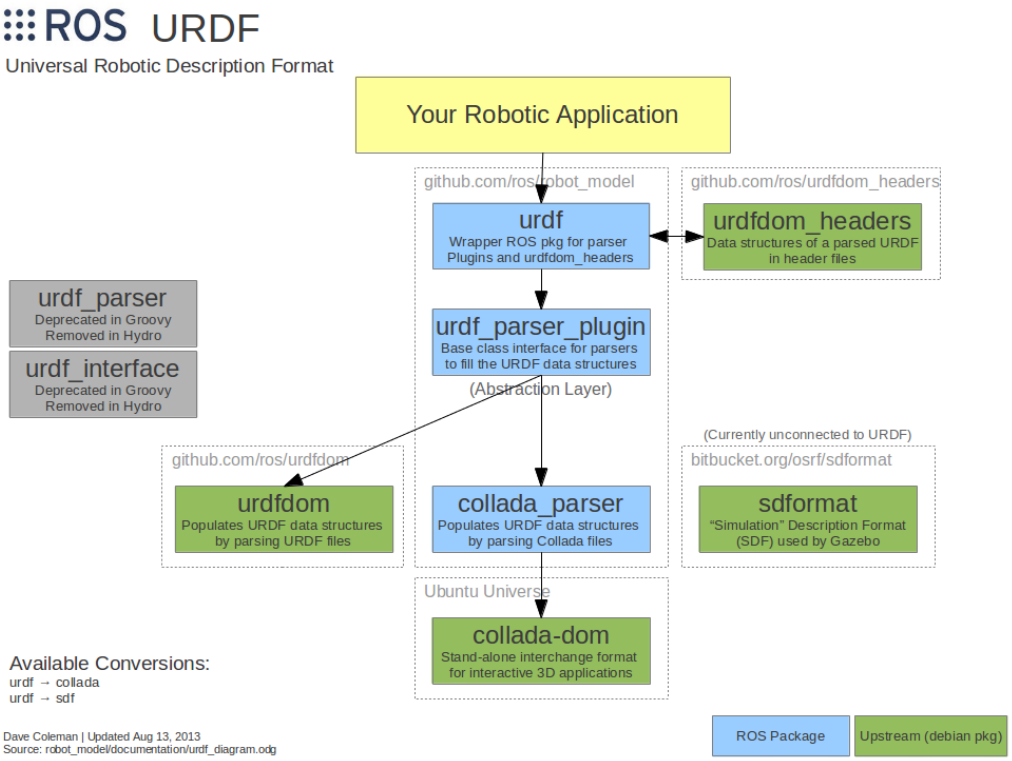

FIg 3 URDF の使用

(http://wiki.ros.org/urdf?action=AttachFile&do=get&target=urdf_diagram.png)

今回は toroboarm_description 内に Torobo Arm の URDF を作成しました。下記のコマンドを実行すると、ROS 標準の可視化環境 Rviz 上でモデルを確認することができます。

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roslaunch toroboarm_description display_toroboarm.launch
```

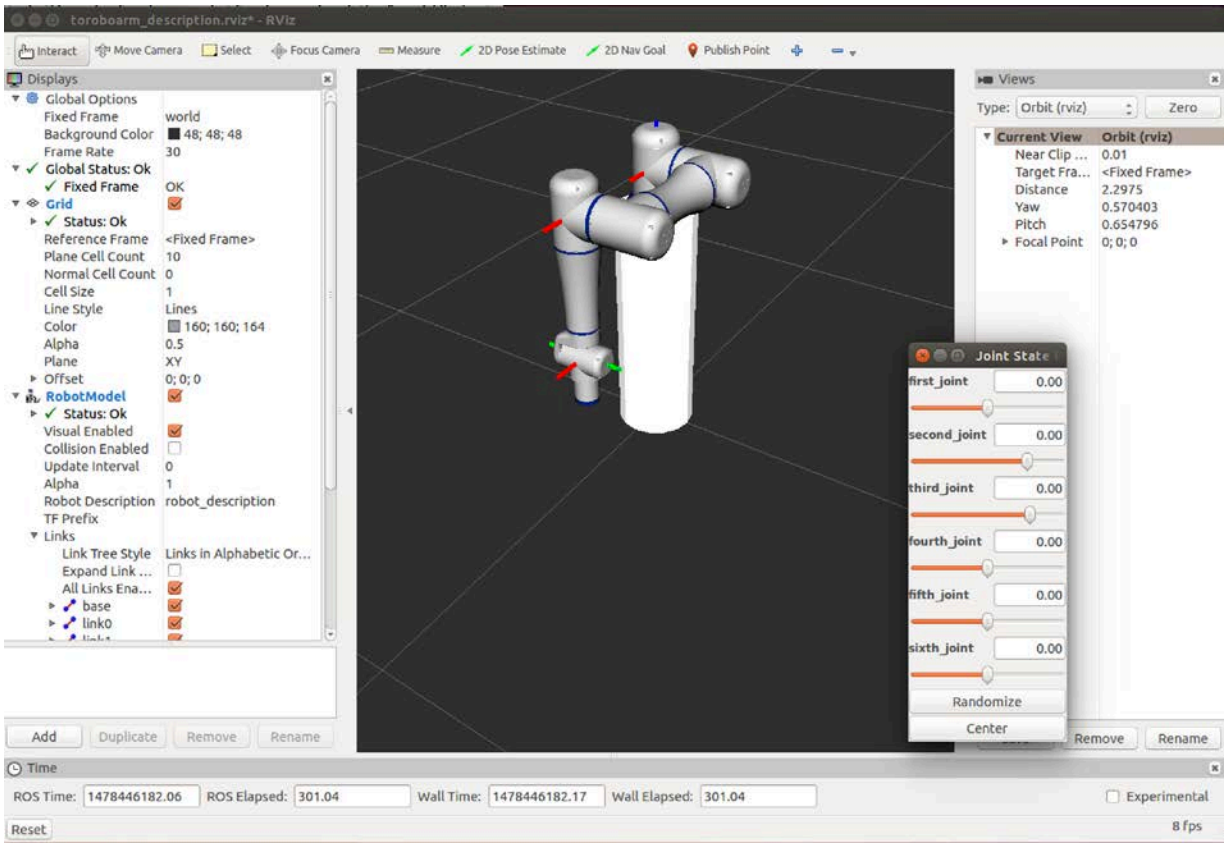

FIg 4 Rviz に表示された Torobo Arm のモデル

Rviz と一緒に表示されたゲージを操作すると、モデルが動作する様子を確認することができます。

4.3. ”MoveIt!”を用いた軌道計画

Torobo Arm の様な多関節ロボットのための軌道計画ソフトウェアとして、ROS では”MoveIt!”が広く使われています。MoveIt!を用いると、前述の Rviz の画面上から、または API を介して C++/Python のコード上から軌道計画を実行することが可能です。障害物回避や物体把持など複雑なタスクを標準でサポートしており、従来開発者の頭を悩ませていた軌道計画・実行の複雑なフローを大幅に簡略化することが可能です。

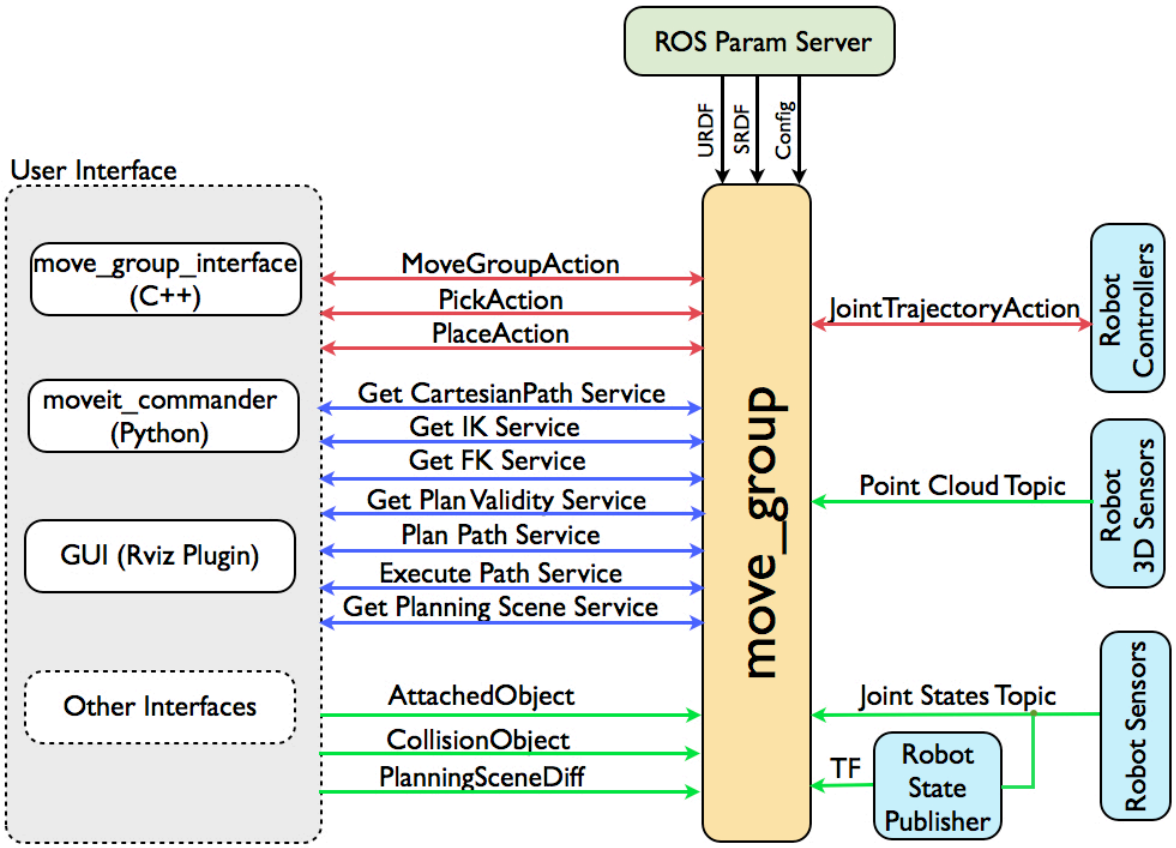

FIg 5 MoveIt のインタフェース

(http://moveit.ros.org/documentation/concepts/)

MoveIt!は軌道計画のために URDF、SRDF(Semantic Robot Description Format)と幾つかのコンフィグ情報を読み込みます。SRDF は MoveIt!に付属の Setup Assistant というソフトウェアに URDF を読み込ませ、画面の指示に従って幾つかの設定情報を入力することで自動的に作成することができます。今回提供されているパッケージには東京ロボティクスが Setup Assistant を使って出力したコンフィグ情報”toroboarm_moveit_config”が含まれているため、すぐに MoveIt!を使い始めることができます。

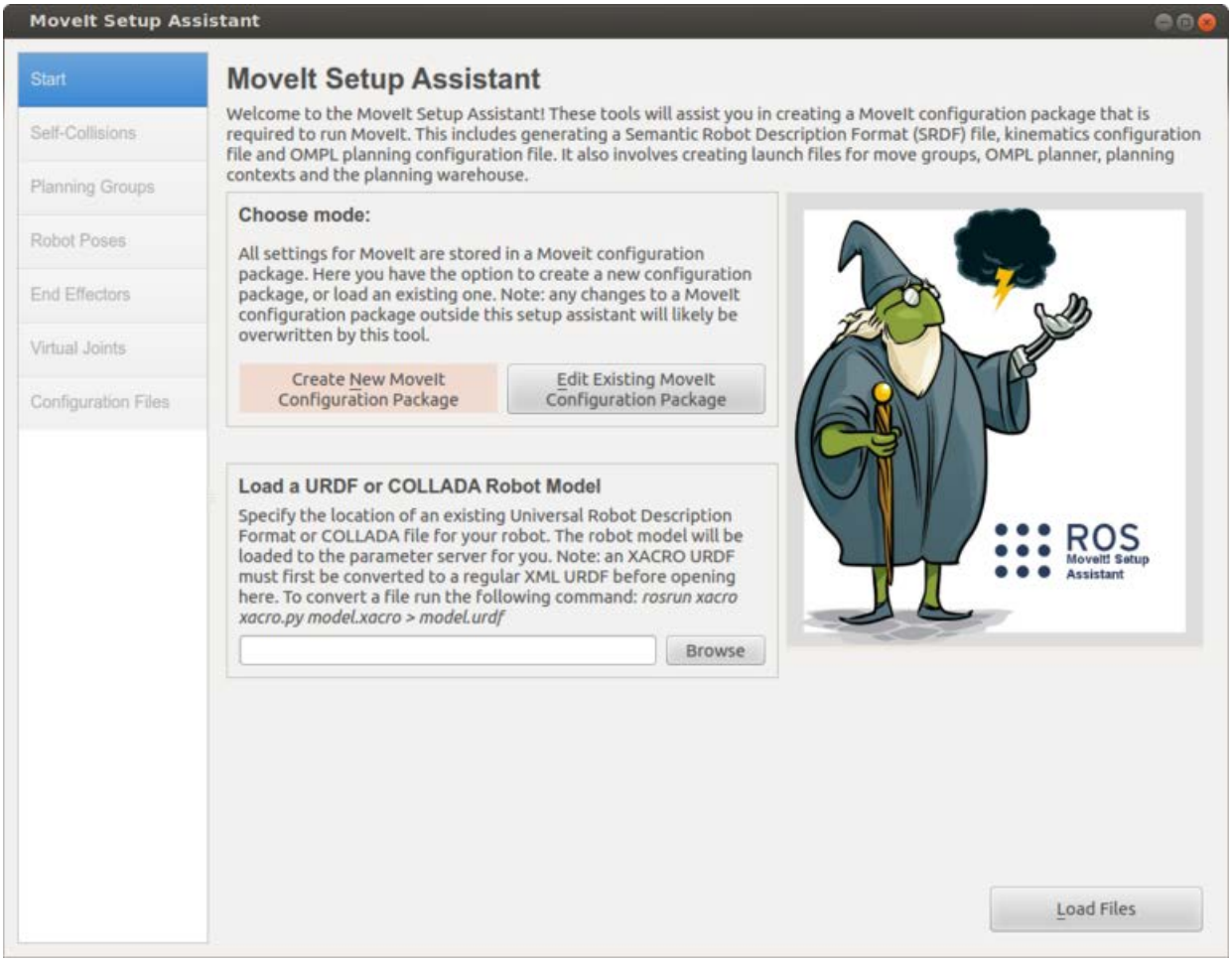

FIg 6 MoveIt! Setup Assistant

(http://docs.ros.org/hydro/api/moveit_setup_assistant/html/_images/setup_assistant_start.png)

早速､下記のコマンドを実行して MoveIt!を使ってみましょう｡MoveIt!のプラグインを読み込んだ Rviz の画面が立ち上がるはずです。

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roslaunch toroboarm_moveit_config demo.launch
```

Planning タブから”Select Goal State”で”home_pose”を選択して”update”を押下して下さい。これで、目標姿勢が新たにセットされました。Commands フィールドの”Plan and Execute”を押下すると、起動が計画され、画面上の Torobo Arm モデルが駆動します。

目標状態の設定は、手先位置に表示されたグリーンのボールをドラッグしても実行することができます。様々な姿勢に動かして起動を計画することができるか確認してみてください。

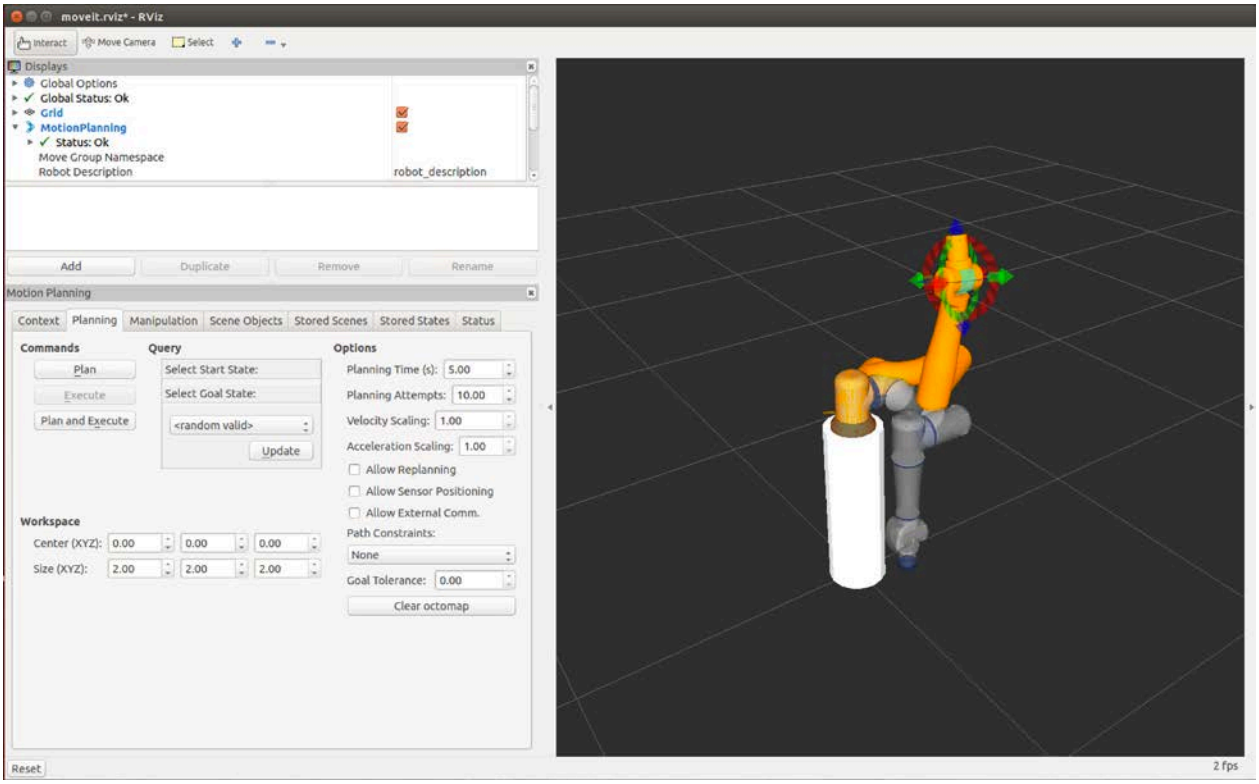

FIg 7 MoveIt!の画面に表示された Torobo Arm

4.4. シミュレーションの実行

ROS では物理シミュレータとして”Gazebo”というソフトウェアを使用します。実際の使い方について解説する前に、ROS におけるシミュレーションと実機制御の扱いについて概説します。

ROS では、シミュレーションと実機の両方で同じコントローラを使用することができるよう、制御対象のインタフェースの型を URDF 内に定義します（実際には URDF の transmittion タグ内に記述されています）。このように記述することで、そのインタフェースに対応した制御コントローラであれば、柔軟に交換することができるようになります。このように、シミュレーションと実機制御をシームレスに切り替えられる工夫が ROS には数多く実装されています。

Gazebo に話を戻しますが、ロボットを Gazebo 上でシミュレートするには、下記の設定が必要です。

・ Gazebo 用のパラメータを URDF に追記する。
・ 前述の HardwareInterface を URDF に追記する
・ シミュレーション中に使用するコントローラを設定する

今回、これらの設定は既に提供したパッケージに含まれており、新規に記述することなく直ぐにシミュレーションを体験することができるようになっています。各種設定内容について確認したいときは、gazebo のチュートリアル（http://gazebosim.org/tutorials/?tut=ros_control）を参考に、toroboarm_control.

toroboarm_description, toroboarm_gazebo の記述内容を確認してみて下さい。

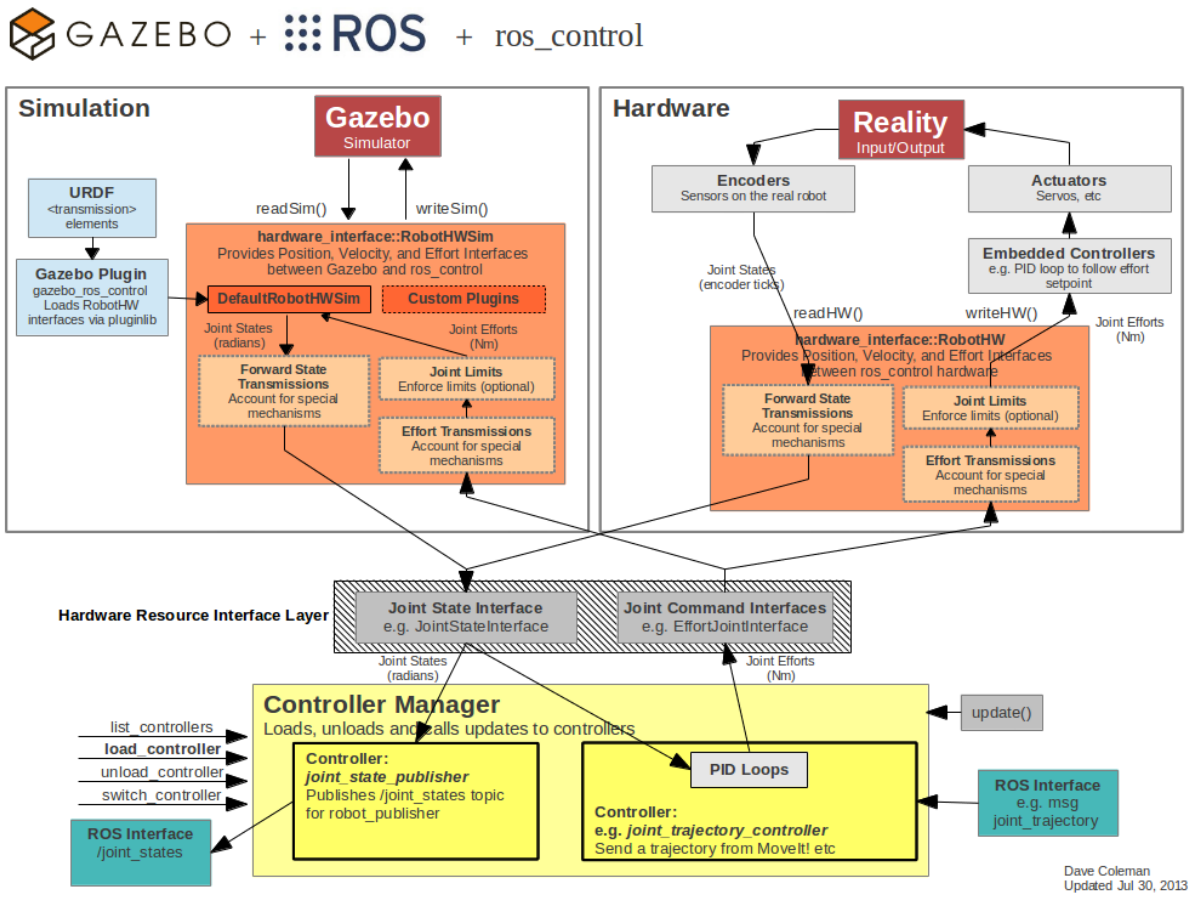

FIg 8 ROS におけるシミュレーションと実機制御の関連
(http://gazebosim.org/tutorials/?tut=ros_control)

それでは、実際にシミュレーションを実行するため、下記のコマンドを実行してみましょう。

```
$ cd ~/catkin_ws/
$ source devel/setup.bash
$ roslaunch toroboarm_bringup bringup_sim.launch
```

Gazebo の画面と MoveIt!の画面が２つ起動します。4.3 の要領で軌道計画を実行してみましょう。”Plan and Execute”を実行すると、Gazebo 中のモデルが動作することが確認できるはずです。

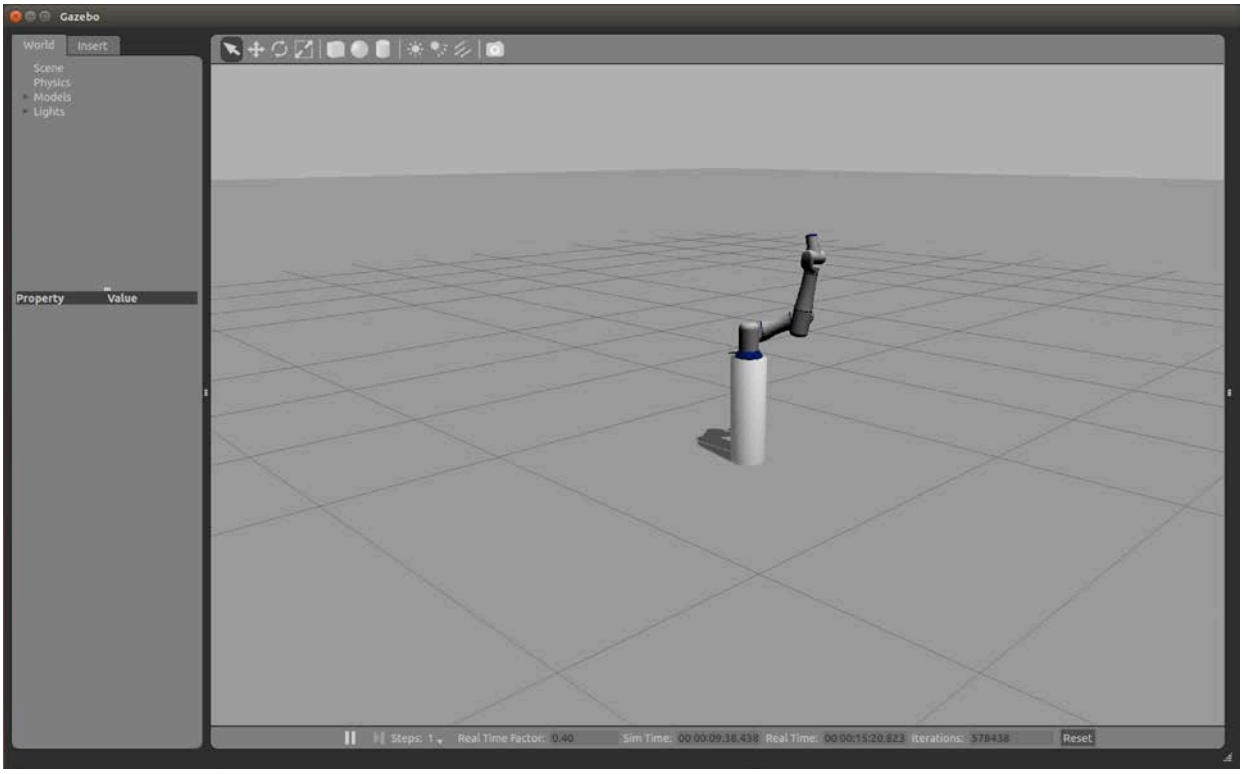

FIg 9 Gazebo 上に表示された Torobo Arm

4.5. 実機を制御する

最後に、MoveIt!で実機を制御してみましょう。

ROS ではロボットの制御ノードに対して制御命令を送信する際、Action という方法で情報がやり取りされます。Goal（目標状態）、Feedback（遷移中の状態）、Result（動作結果）の３つの情報をクライアント、サーバ間で事前に設定したインタフェース形式でやり取りをするというものです。

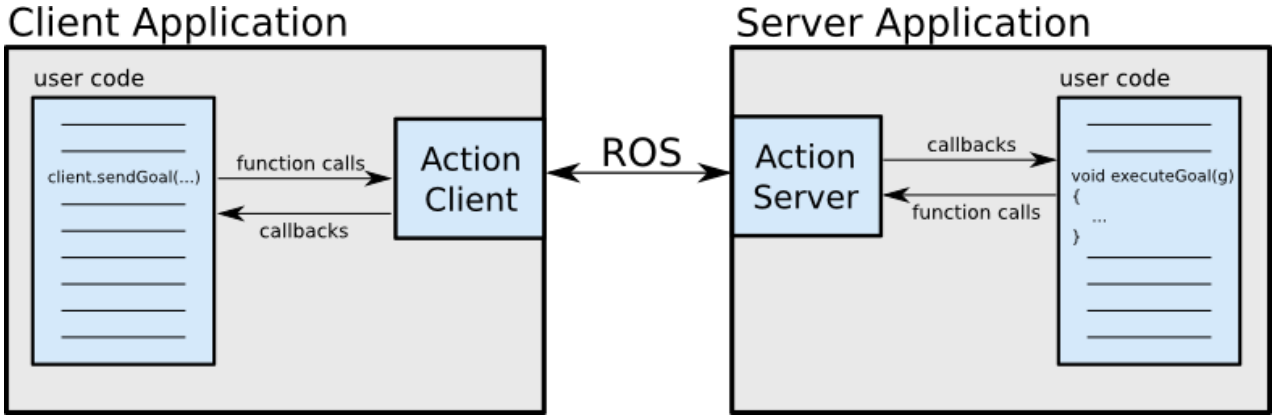

FIg 10 Action の概念図（http://wiki.ros.org/actionlib）

MoveIt!の場合も例外ではありません。MoveIt!は”Execute”を押下すると、軌道の経由点の情報がFollowJointTrajectryAction という型で制御ノードに渡されます。制御ノードはそれを受け取り、ロボットに制御命令を送信します。

今回提供した Torobo Arm 用のパッケージでは、toroboarm_control パッケージに含まれるtoroboarm_control_node が FollowJointTrajectryAction を受け取り、/joint_path_command というトピックにメッセージをパブリッシュすると、toroboarm_driver の command_sender がこれを受け取り、PVT制御で実行可能な CSV 形式に変換して Torobo Arm に実行するよう指令を送る様な構成になっています。

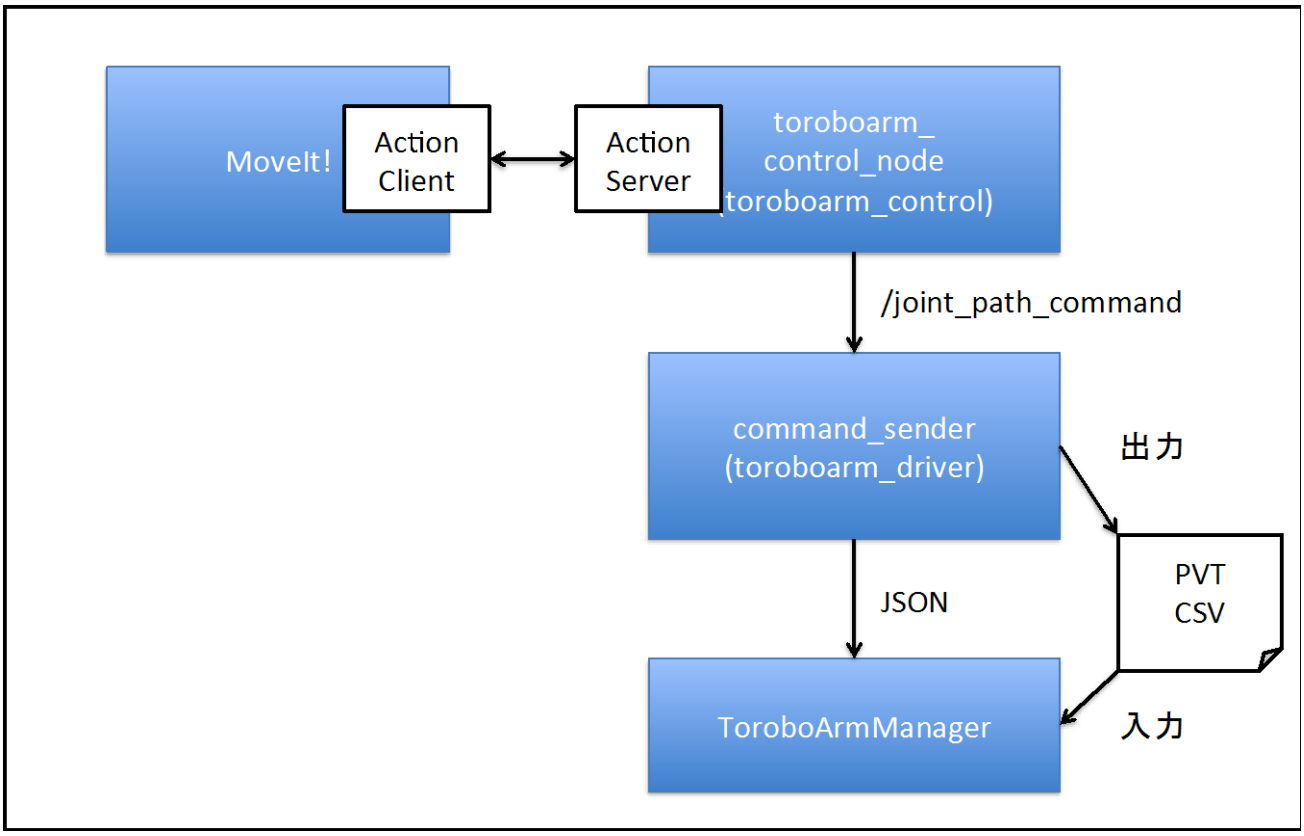

FIg 11 ToroboArm における実機制御の流れ

実機を動かすには、ToroboArmManager が起動している必要がありますので、事前にターミナルを起動して下記のコマンドを実行しておきましょう。

```
$ cd ~/toroboarmmanager/bin
$ ./armmng
```

さらに、MoveIt!を実行する前に、下記のコマンドを実行してロボットを PVT 制御モードに変更しておきましょう。

```
$ armctl --reset --joint=all
$ armctl --servo ON --joint=all
$ armctl --mode 20 --joint=all
```

最後に下記のコマンドを実行すると、MoveIt!の画面が起動します。

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ chmod +x src/toroboarm_robot/toroboarm_driver/scripts/*
$ roslaunch toroboarm_bringup bringup_real.launch
```

シミュレーションの場合と同様に軌道を計画・実行すると、実際に実機が動作することが確認できるはずです。なお、**実機を動かす場合は”Plan and Execute”にしてしまうとどの様に動くか事前に分からず危険**なため、”Plan” を押下して軌道を確認してから”Execute”を押下するようにしましょう。

5. 著作権について

当社ロボットアームに付随するサンプルソースコードおよびユーザーマニュアルの著作権は当社に帰属します。これらの著作物は著作権法で保護されており、これら著作物の全部または一部を無断でいかなる手段によっても転載、配布、出版することは固くお断りいたします。

ソースコードに関しては、当社ロボットアームと併せて用いる場合に限り、複製と改変を認めます（配布は認めません）。なお、ソースコードとユーザーマニュアルの内容は、事前の通知なしに変更する場合がございますのでご承知おきください。