アカデミック　スカラロボット
ROS 対応 SDK
取扱説明書

(2020.06.04)

Vstone®
ヴイストン株式会社

内容

1　はじめに / 注意事項

本書は、アカデミック スカラロボット(以降「ロボット」「ロボット本体」と記述)の ROS 対応 SDK(以降「本 SDK」と記述)に関する説明書兼チュートリアルです。ロボット本体の概要・お取り扱いにつきましては、ロボット本体に付属の各種資料をご参照ください。

本製品の使用にあたっては下記注意事項に従い、正しくご使用ください。

○ 本製品を無許可で複製、再配布、再販することはできません。ただし、著作権法で認められた範囲における複製については許可されます。
○ 本製品の対応環境は、ネイティブの Ubuntu 18. 04 と ROS Melodic です。それ以外の環境での使用についてはサポート対象外となる他、それによって生じたいかなる損害についても、製造元および販売元は何らの責任を負いません。
○ 本製品の使用にあたっては、 本製品に含まれない公開ライブラリおよびアプリケーションを多数使用する必要 がございます。本製品に含まれないライブラリについてはサポート対象外となる他、その使用によって生じたいかなる損害についても、製造元および販売元は何らの責任を負いません。
○ 本製品を使用する PC はお客様にてご準備ください。 Ubuntu のデバイスドライバの対応状況等により、一部の機能が正常に動作しない可能性がございますが、デバイス固有の問題についてはサポート対象外となる他、それによって生じたいかなる損害についても、製造元および販売元は何らの責任を負いません。

2 ROS について

2.1 ROS の概要

ROS（Robot Operation System）は、OSRF（Open Source Robotics Foundation）によって開発・メンテナンスされているロボット用のミドルウェアです。分散処理が求められる複雑なロボットシステムを制御できる性能を備えており、世界中の研究者や開発者が作成した豊富なライブラリを使用することができます。ロボット制御システムの作成を効率化できることから、人型ロボットから車両型ロボット、水上・水中ロボットやドローンに至るまで、幅広い分野で活用されています。

ROS の特徴のひとつが、BSD ライセンスに基づくオープンソースとして公開されており、誰でも開発に参加し貢献できることです。ROS には強力な開発者コミュニティが存在し、誰でも使用可能な 5000 以上のライブラリのほとんどは、OSRF ではなくコミュニティによって開発・メンテナンスされています。

ROS 本体が BSD ライセンスによって提供されているため､ROS を用いて開発した成果物は、商用利用することが可能です。ROS を用いて動作する様々なロボットが発売されており、メガローバーもそのひとつです。ただし、ライブラリによっては BSD ではないライセンスによって提供されているものも存在するため、商用利用ではご注意ください。

2.2 ROS が提供する機能

ROS によって提供される主要な機能について、説明します。

- メッセージ通信

  ROS を用いて構成されるロボットシステムは分散処理が基本となっており、ユーザは「ノード」と呼ばれるプログラムを複数立ち上げることでシステムを作成します。例えば、ゲームパッドで操作できる台車ロボットであれば、「ゲームパッドの入力値を取得するノード」、「入力値に従って移動指令を出すノード」、「移動指令をもとにモータを回転させるノード」などが必要となります。当然、ノード間で情報を通信する仕組みが求められます。各ノード間で、センサ入力値の情報やカメラの映像、制御指令値といったデータをやり取りするために、ROS では Pub/Sub 方式によるメッセージ通信が提供されています。開発者はわずか数行のコードにより、任意の情報をパブリッシュ（配信）したり、サブスクライブ（購読）したりすることができます。メッセージには、構造体のような型が定められているため、ROS ノード同士であれば互換性を気にする必要はありません。また、型は自作することもできます。

- パッケージ管理

  ROS のライブラリやプログラムは、パッケージという単位で管理されています。パッケージの中にはノードやその設定ファイル、起動スクリプトなどが含まれており、ユーザは使用したい機能を持つパッケージをインターネット上からダウンロードし、ローカル環境に組み込むことができます。パッケージの追加や削除といった操作は非常に簡単に行う

事ができます。また、ユーザが独自の制御プログラムを開発する際には、まずパッケージを作成し、その中で開発を行う事になります。

- デバイスドライバ

  ROS では様々なデバイスのドライバがパッケージの形式で提供されています。対応しているデバイスであれば、パッケージを導入し、デバイスを接続するだけで使用することができます。

- ハードウェア抽象化

  ROS による制御システムは複数のノードによる分散処理によって動作します。これにより、ハードウェアが異なるロボットでも、上流の制御システムは同じものが使用できます。

- ライブラリ

  ROS では、5000 を超える公開ライブラリを使用することができます。それらはデバイスドライバのようなものから、経路計画や動作生成といったものまで様々です。

- 視覚化ツール

  ROS には、いくつかの視覚化ツールが存在しており、ロボットの操縦 UI やデバッグ等のために活用されています。中でも「Rviz」は強力なツールです。3D 表示機能を持つこのツールは、実に多くのパッケージで情報を表示するために使われています。表示情報のカスタマイズが容易ですので、ユーザオリジナルの UI を作成することも容易です。

2.3 ROS に関する情報の集め方

ROS を用いた開発を行う際には、使用するパッケージの情報など、様々な情報が必要になります。ROS やその開発に関する情報は書籍から集めることもできますが、ここではインターネットから情報を集める際に参考になるサイトをいくつかご紹介します。

- ROS Wiki - http://wiki.ros.org/

  ROS の公式 Wiki です。ROS のインストール方法からチュートリアル、各公開パッケージの情報まで、様々な情報が公開されています。ただ、パッケージの情報などが更新されないまま古くなっていることもありますので、ご注意ください。

- ROS Wiki(ja) - http://wiki.ros.org/ja

  ROSWiki の日本語訳版です。現在も有志による精力的な翻訳作業が行われていますが、古い情報も多いので、英語版とあわせて確認した方がよいでしょう。

- ROS Answers - https://answers.ros.org/questions/

  ROS の Q&A フォーラムです。パッケージを使用した際のエラーの解消法など、様々な情報が蓄えられています。

3　ROS のインストール

ROS Melodic を Ubuntu にインストールする方法を説明します。ここで述べる手順は、ROS Wiki で紹介されているインストール方法から一部を抜粋したものです。より詳しい情報が欲しい方は、下記ページを確認してください。

ROS Melodic の Ubuntu へのインストール - http://wiki.ros.org/melodic/Installation/Ubuntu

ROS のファイルはインターネットから取得するため、インターネット接続が必要です。PC をインターネットに接続してください。ネットワークの接続制限があったり、プロキシが設定されている環境では、ファイルのダウンロードに失敗することがあります。

① 端末（シェル）を立ち上げる

ランチャー上部の「コンピュータの検索」から“端末”を検索するか、ショートカットキー“Ctrl+Alt+T”を使用して、新しい端末（シェル）を起動します。

図 3-1　コンピュータの検索

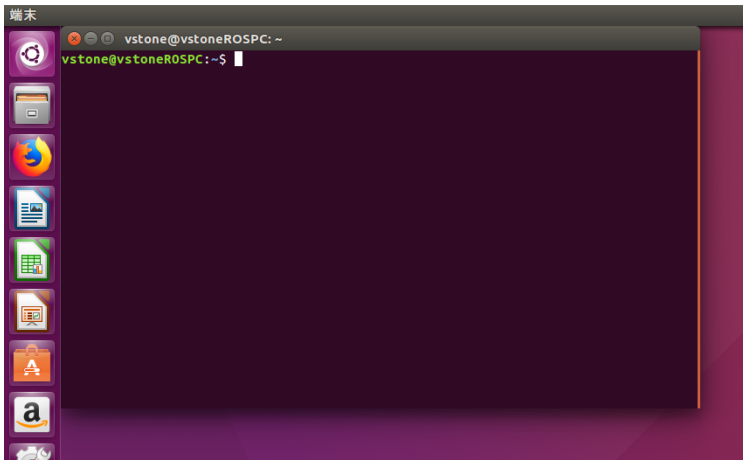


図 3-2　端末（シェル）

② sources.list を設定する

以下の青枠内のコマンドを端末にコピー&ペーストし、エンターキーを押して実行してください。コマンドラインへの貼り付けは右クリックまたは “Ctrl+Shift+V” で行えます。ひとつの青枠内にひとつのコマンドを書いていますので、複数行に分かれていてもまとめてコピーしてください。コピー&ペーストできない場合は手で入力してください。

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

③ 鍵を設定する

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

④ インストール

先にパッケージインデックスをアップデートしておきます。

```
sudo apt update
```

ROS Melodic をインストールします。ネットワーク環境によってはかなり時間がかかります。

```
sudo apt install ros-melodic-desktop-full
```

ROS のライブラリをバイナリでインストールする場合、上記のように「ros-バージョン-パッケージ名」で指定します。以下、バージョンに相当する箇所は、melodic をご利用の場合は「melodic」に置き換えて入力してください。

⑤ rosdep の初期化

以下の 2 つのコマンドを順に実行してください。

```
sudo rosdep init
```

```
rosdep update
```

⑥　環境設定

パスの設定を行います。Melodic の場合、以下の 2 つのコマンドを順に実行してください。

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

⑦　パッケージ構築のための依存ツールのセットアップ

```
sudo apt install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

以上で、ROS Melodic 本体のインストールは完了です。ROS の基幹プログラムを立ち上げるコマンド“roscore”を使って、起動することを確認しておきましょう。

```
roscore
```

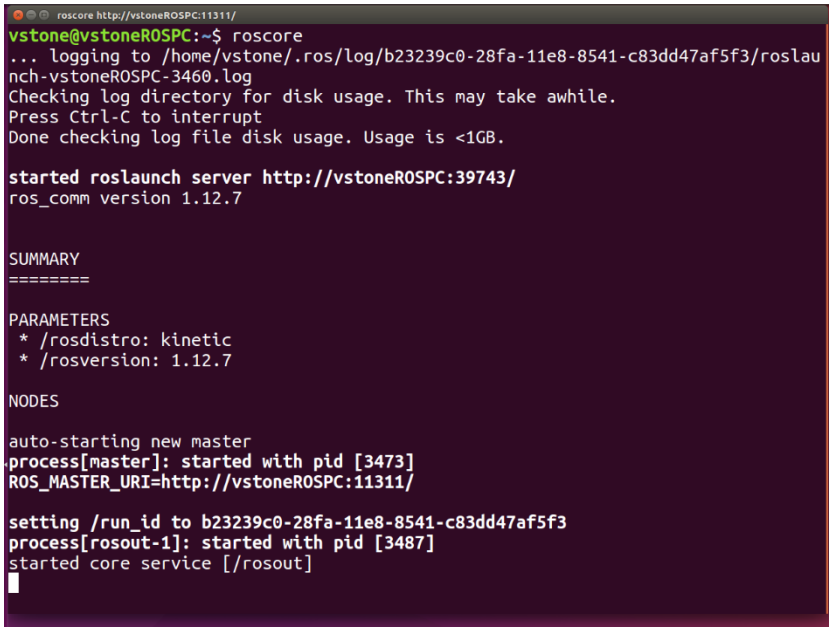


図 3-3 roscore 起動時のシェル

上のようなメッセージが表示されれば、ROS のインストールは正常に行われています。

roscore は、ROS の各サービスを提供する基幹プログラムです。ROS を使用する際には、必ずこの roscore を起動しておく必要があります。

4　開発環境のセットアップ

本 SDK における ROS の各種パッケージのセットアップを行います。下記の手順に従って作業を行ってください。

4.1 catkin ワークスペースの作成

ROS のビルドシステムである catkin のためのワークスペースを作成します。この作業は、ROS Wiki に掲載されているチュートリアル「ROS 環境のインストールとセットアップ」（http://wiki.ros.org/ja/ROS/Tutorials/InstallingandConfiguringROSEnvironment）にも記載されています。

端末を起動して、以下のコマンドを順番に実行してください。

・フォルダの作成

```
mkdir -p ~/catkin_ws/src
```

・作成した src フォルダに移動

```
cd ~/catkin_ws/src
```

・ワークスペース作成

```
catkin_init_workspace
```

これでワークスペース「catkin_ws」ができあがりました。ワークスペースの src フォルダ内にパッケージフォルダを配置していくことができます。作成したばかりの src フォルダは空ですが、~/catkin_ws で以下のコマンドを実行することで、ワークスペースをビルドすることができます。

・(src フォルダに居る場合) ~/catkin_ws に移動

```
cd ~/catkin_ws
```

・ワークスペースをビルド

```
catkin_make
```

このワークスペース内で作成したパッケージを動作させるためには、ワークスペースをオーバーレイする必要があります。オーバーレイについては ROS Wiki にある catkin のチュートリアル「workspace_overlaying」(http://wiki.ros.org/catkin/Tutorials/workspace_overlaying)を確認してください。オーバーレイを行うためには、~/catkin_ws で次のコマンドを実行します。

```
source devel/setup.bash
```

ただしこの状態では、端末を新しく起動するたびにオーバーレイの作業を行う必要があります。端末起動時に自動的にオーバーレイが実行されるようにするためには、以下のコマンドを用いて設定を.bashrc に書き込みます。

```
echo "source /home/ユーザ名/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

以上で、ワークスペースの作成は完了です。

4.2 必要なパッケージのインストール

続いて、本 SDK に必要なパッケージ類をインストールします。それぞれ、記載のコマンドを端末より入力してください。

joint_state_publisher のパッケージをインストールします。

```
sudo apt install ros-melodic-joint-state-publisher-gui ros-melodic-joint-state-publisher
```

ros_control のパッケージをインストールします。長いですが 1 コマンドになります。

```
sudo apt install ros-melodic-ros-control ros-melodic-ros-controllers ros-melodic-joint-state-controller ros-melodic-effort-controllers ros-melodic-position-controllers ros-melodic-joint-trajectory-controller
```

MoveIt のパッケージをインストールします。

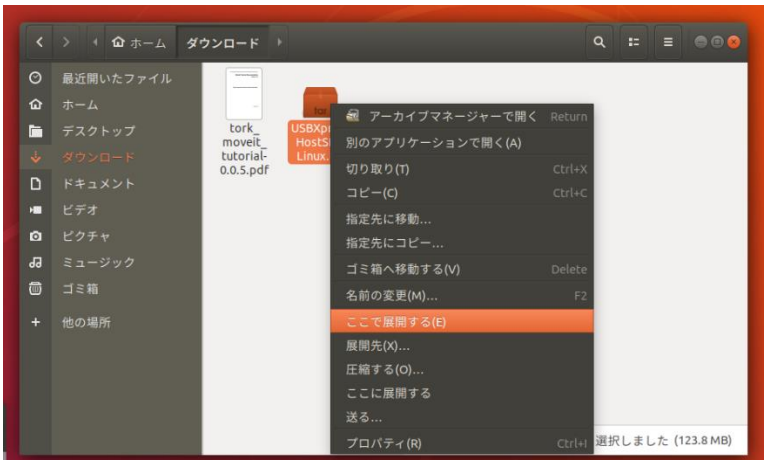
```
sudo apt install ros-melodic-moveit ros-melodic-moveit-ros-visualization
```

4.3 Slllicon Lab 通信ライブラリの導入

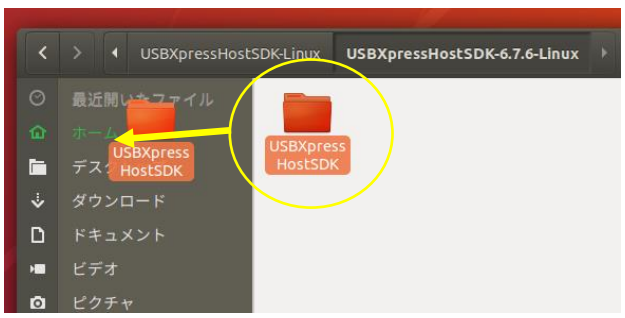
ロボット本体との通信を行うため、sillicon lab より公開されている通信ライブラリをインストールします。

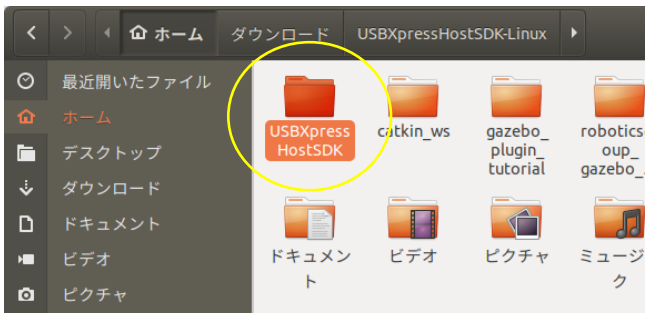
下記の URL のファイルをダウンロードしてください。
https://www.silabs.com/documents/public/software/USBXpressHostSDK-Linux.tar

ダウンロードしたらファイルを展開します。GUI より展開する際は、一度展開して生成される tar.gz ファイルを更に展開してください。

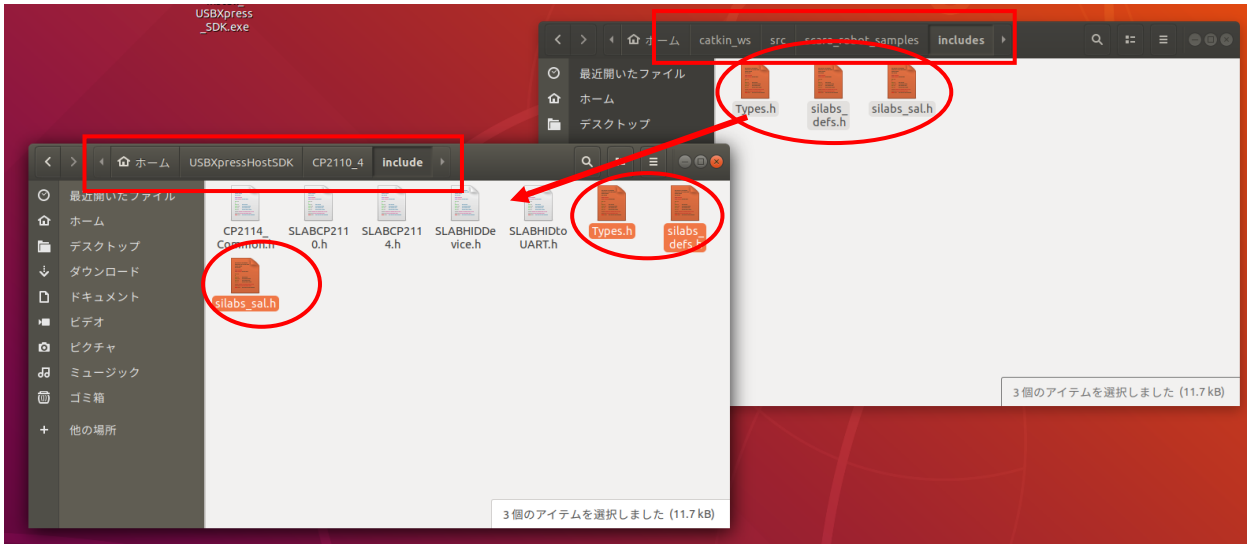


展開すると「USBXpressHostSDK」のフォルダができるので、このフォルダを「~/」にコピーしてください。GUI の場合「ホーム」にフォルダごとコピーします。





続いて、通信ライブラリをビルドするのに必要なヘッダファイルを本 SDK よりコピーします。本 SDK の「scara_robot_samples/includes」ディレクトリにある「Types.h」「silabs_defs.h」「silabs_sal.h」の各ファイルを、先ほどホームに移動した「USBXpressHostSDK」フォルダ内の「CP2110_4/include」フォルダにコピーしてください。



続いて、ライブラリへのパスを設定します。端末より下記のコマンドを入力してください。長いですが 1 行のコマンドです。

```
echo “export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/USBXpressHostSDK/CP2110_4/lib/x86_64” >> ~/.bashrc
```

4.4 サンプルパッケージのダウンロードとビルド

GitHub で公開している本 SDK のパッケージをダウンロードしビルドします。以下のコマンドを順番に実行してください。

・~/catkin_ws/src フォルダに移動します

```
cd ~/catkin_ws/src
```

・GitHub よりソースコードをクローンします

```
git clone https://github.com/vstoneofficial/scara_ros_samples.git
```

・ビルドします。独自に設定している ROS メッセージの関係上、下記のように 2 度ビルドを行います。

```
cd ~/catkin_ws
```

```
catkin_make -DCATKIN_BLACKLIST_PACKAGES="scara_robot_samples"
```

・一度ビルド時のキャッシュをクリアします。

```
catkin_make -DCATKIN_BLACKLIST_PACKAGES=""
```

・再度ビルドします。表示が[100%]になればビルド完了です。

```
catkin_make
```

ROS の公開ライブラリの多くは GitHub でソースコードを公開しています。GitHub で公開されているものについては、同じ手順でワークスペースに追加、ビルドして使用することができます。

4.5 gazebo_mimic_joint プロジェクトのダウンロードとビルド

Gazebo によるシミュレーションで必要となる gazebo_mimic_joint を github よりクローンしてビルドします。

・~/catkin_ws/src フォルダに移動します

```
cd ~/catkin_ws/src
```

・GitHub より gazebo_mimic_joint のソースコードをクローンします

```
git clone https://github.com/roboticsgroup/roboticsgroup_gazebo_plugins.git
```

・ビルドします、表示が[100%]になればビルド完了です。

```
cd ~/catkin_ws
```

```
catkin_make
```

4.5 パッケージの内容について

本 SDK は「scara_robot_samples」と「scara_robot_moveit」の二つのパッケージに大きく分かれております。前者は USB 通信制御や本体の構造に関する基本設定及びシンプルな関節の制御部分に絞った基本処理、後者は ROS 用の動作計画フレームワーク「MoveIt」を用いたサンプルパッケージになります。それぞれの内容について簡単に説明します。

- scara_robot_samples パッケージ

  - ノード

  scara_robot_samples には、「scarasubscriber」「jointstatebridge」「teaching」というノードが存在します。

「scarasubscriber」は、アームの各関節の角度・位置情報を受け取って実機のロボット本体に反映させるノードです。「jointstatebridge」は、rviz 等で出力される joint_state を scarasubscriber に伝達するノードです。「teaching」は、実機の関節より角度・位置を読み取って配信するノートです。

それぞれのソースコードは scara_robot_samples/src に存在します。ソースコードを書き換えてビルドすれば、機能を変更することができます。また、同フォルダに存在する「scara_lib.cpp」「scara_lib.h」は、ノードではありませんが、ロボット本体と通信するための基本処理であり、各種ノードより参照されます。

  - launch ファイル

    launch フォルダ内に存在する拡張子が「launch」のファイルは、ROS で使用できる非常に便利なファイルです。ROS で制御システムを構築するためには数多くのノードを起動する必要がありますが、それをひとつひとつ手動でやるのは大変です。launch ファイルを使えば複数のノードを同時に起動することができます。

    他にも、パラメータの設定や remap 機能など、便利な機能を使用することができますので、積極的に利用しましょう。

    scara_robot_samples の launch フォルダ内には、サンプルプログラムを実行するための複数の launch ファイルが格納されています。各 launch ファイルについては、5 章以降で説明します。

  - configuration ファイル

    configuration_files フォルダ内に存在する、拡張子が「yaml」や「lua」のファイルです。サンプルプログラム実行時に使用する様々なパラメータ等について記載された設定ファイルです。

- CMakeLists.txt と package.xml

  CMakeLists.txt と package.xml は ROS パッケージに必須な、catkin の設定ファイルです。詳しくは下記のサイトを参照してください。

  CMakeLusts.txt: http://wiki.ros.org/catkin/CMakeLists.txt
  Package.xml: http://wiki.ros.org/catkin/package.xml

- scara_robot_moveit パッケージ

  - launch ファイル

    scara_robot_samples と同じく、launch フォルダに各種設定を一度に行う launch ファイルを多数配置しています、多くのファイルは、assistant で自動生成されるサンプルです。

  - configuration ファイル

    「MoveIt」で使用するための ros_controller などのファイルを配置しています。こちらも、多くのファイルが assistant で自動生成されるサンプルです。

5　ロボット本体と ROS との接続

ロボット本体を PC と通信させる場合は、下記の手順でデバイスの権限を変更する必要があります。まず、ロボット本体に付属の AC アダプタ及び USB ケーブルを接続し、続いて PC と USB ケーブルで接続してください。接続したら、PC より端末を立ち上げて「lsusb」と入力します。

```
lsusb
```

入力すると、画面上に現在接続されているデバイス一覧が表示されます。この中から、「CP210x UART Bridge」と言う表記があるものを確認してください。デバイスを確認できたら、同一行にある「Bus ***」と「Device ***」の番号をそれぞれ確認してください。

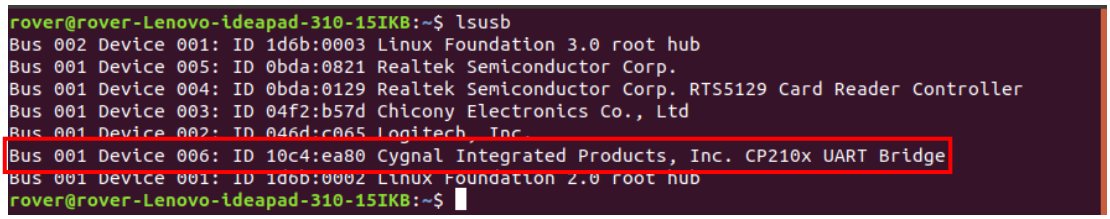


図 5-1　この場合、Bus:001、Device:006 となる

番号を確認したら、続いて端末より下記の通り入力してください。

```
cd /dev/bus/usb/(「Bus」の後に記載された番号)
sudo chmod 666 (「Device」の後に記載された番号)
```

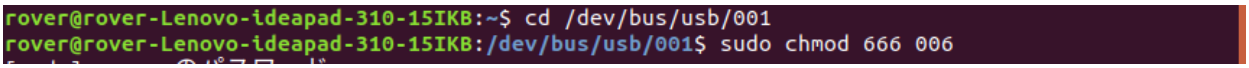


図 5-2　先ほどの番号を元にデバイスの権限を変更

デバイスの権限の変更は、ロボット本体を接続し直したり、PC をシャットダウン・スリープさせたりすることで、ID が変更されたり設定が解除される場合があります。これらの操作を行った後は、再度デバイスの権限を変更してください。

正しく通信できるようになっているか、一度サンプルの launch ファイルを使って確認してみます。端末で下記のコマンドを実行してください。

```
roslaunch scara_robot_samples rviz_teaching.launch
```

実行すると、PC の画面上に rviz によるロボットの 3D モデルが表示され、PC に接続したロボット本体に合わせて動きます。手でロボットのモータを動かして、画面上の 3D モデルがそれに追従すれば、正しく通信ができています。

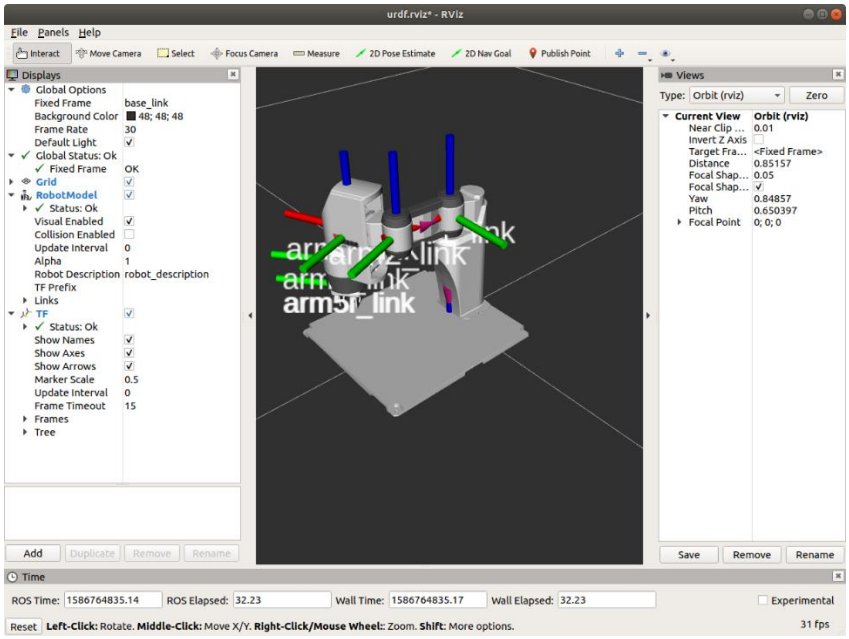


図 5-3　画面に 3D モデルが表示され、ロボット本体の動きを追従する

各関節の原点に座標軸を示す赤・青・緑の棒が表示されますが、ウィンドウ左の「TF」のチェックを外すと表示を消すことができます。

6　ロボット本体との通信

本 SDK に含まれる launch ファイルを元に、SDK に含まれる各種機能について説明してきます。

6.1 起動用 launch ファイルの実行

先ほどの通信確認でも利用しましたが、本 SDK では launch と呼ばれるファイルによって機能を容易に確認できるようになっています。まずはロボット本体との基本的な通信処理、及び rviz による 3D モデルの表示について、launch を起動して確認してみましょう。

PC とロボット本体を接続し、通信できるようにデバイスの権限を変更したら、端末より下記のコマンドを入力してください。

```
roslaunch scara_robot_samples rviz_controller.launch
```

実行すると、下図のような画面が開き、またロボット本体が 3D モデルと同様アームを前に伸ばしたポーズでモータに力が入ります。Rviz と共に joint_state_publisher という小さいウィンドウも開き、ウィンドウ内のスライダーをマウスで操作すると、3D モデル及びロボット本体が操作に追従して動きます。

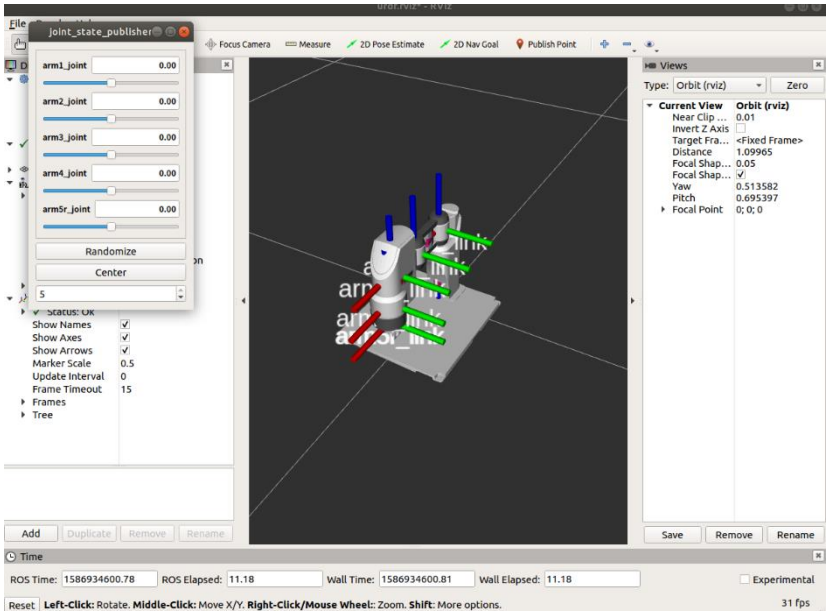


図 6-1 rviz_controller.launch の起動後

3D モデルの表示領域内をマウスでドラッグすると、視点の位置や角度を変更できます。ドラッグ時に押しているマウスのボタンに応じて変更するパラメータが変わります(左クリック:回転、右クリック:拡大/縮小、ホイールクリック:位置)。

起動したプログラムを終了させる場合は、ウィンドウ右上の×印をクリックしてウィンドウを閉じるか、launch ファイルを実行した端末上で Ctrl+C キーを押してください。

次に、ロボット本体から情報を読み取るサンプルの launch を起動します。こちらは前述の動作テストでも使用したもので、端末より下記の通り入力してください。

```
roslaunch scara_robot_samples rviz_teaching.launch
```

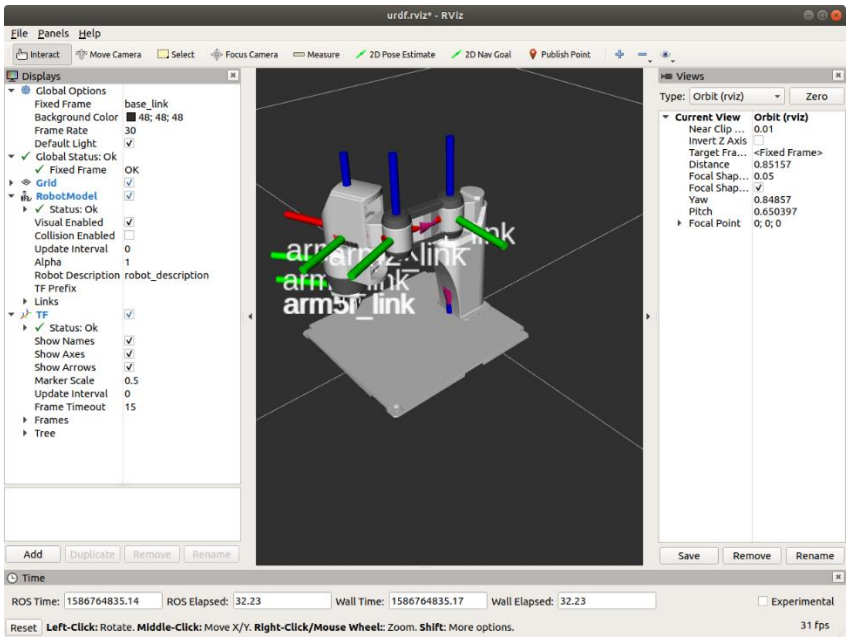


図 5-2 rviz_teaching.launch の起動後

このサンプルではロボット本体に追従して 3D モデルのポーズが変化しますが、これはロボット本体より逐次現在のモータ角度を読み取って 3D モデルにフィードバックしています。

6.2 メッセージの仕様

ROS では、各ノードが様々なメッセージをやり取りすることで動作を実現しています。本 SDK では、ロボット本体との通信用として scarasubscriber 及び teaching というノードが含まれており、それぞれ下記のメッセージをサブスクライブ（購読）及びパブリッシュ（配信）しています。ここでは、各メッセージの仕様について解説します。

なお、ノードのソースは scara_robot_samples の src フォルダに入っており、プログラムを改良してメッセージの内容を追加・変更することが可能です。

〔サブスクライブ〕

- /scara_controller/command

  ロボット本体の各関節の位置・角度情報です。メッセージは各関節(5 個)に応じた変数を持ち、関節によって角度(360 度法)・位置(mm 単位)をそれぞれ指定します。表 6-1 に詳細を示します。

表 6-1　/scara_controller/command の詳細

メッセージ名	/scara_controller/command
型	scara_robot_samples/Axes
内容	axis1:　　//　第一関節の角度（float64） axis2:　　//　第二関節の角度（float64） axis3:　　//　第三関節の高さ(mm)（float64） axis4:　　//　第四関節の角度（float64） axis5:　　//　第五関節の幅(mm)（float64）

- /scara_controller/position

  ロボット本体の手先の座標です。X/Y/Z の三次元(右手系)の座標で指定します。ロボットの原点についてはロボット本体の資料をご参照ください(ロボット本体の資料は左手系の座標表記となっています)。表 6-2 に詳細を示します。

表 6-2　/ scara_controller/position の詳細

メッセージ名	/scara_controller/position
型	geometry_msgs/Point
内容	x:　　//　手先位置の x 座標（float64） y:　　//　手先位置の y 座標（float64） z:　　//　手先位置の z 座標（float64）

- /scara_controller/poweron

  ロボット本体のモータの ON/OFF 情報です。Bool 値で表現され、ON=True、OFF=False です。シミュレーションではこのメッセージはサブスクライブされません。表 6-3 に詳細を示します。

表 6-3　/ scara_controller/poweron の詳細

メッセージ名	/scara_controller/poweron
型	geometry_msgs/Bool
内容	data:　　//　モータの ON/OFF（bool）

〔パブリッシュ〕

- /scara_controller/potentiometer

  ロボット本体の各関節の位置・角度情報です。各関節(5 個)に応じた変数を持ち、関節によって角度(360 度法)・位置(mm 単位)をそれぞれ指定します。表 6-4 に詳細を示します。

表 6-4　/scara_controller/ potentiometer の詳細

メッセージ名	/scara_controller/ potentiometer
型	scara_robot_samples/Axes
内容	axis1:　　//　第一関節の角度（float64） axis2:　　//　第二関節の角度（float64） axis3:　　//　第三関節の高さ(mm)（float64） axis4:　　//　第四関節の角度（float64） axis5:　　//　第五関節の幅(mm)（float64）

- /scara_controller/readposition

  現在のロボットの手先情報を記録しているメッセージです。座標は X/Y/Z の三次元(右手系)で表現します。表 6-5 に詳細を示します。

表 6-5 /scara_controller/readposition の詳細

メッセージ名	/scara_controller/readposition
型	geometry_msgs/Point
内容	x:　　//　手先位置の x 座標（float64） y:　　//　手先位置の y 座標（float64） z:　　//　手先位置の z 座標（float64）

7　シミュレーションサンプル

本 SDK には、gazebo によるロボット本体のシミュレーションについても含まれています。シミュレーションに関するサンプルは、端末より下記のように入力して launch ファイルを実行して下さい。

```
roslaunch scara_robot_samples gazebo_robot_cntroller.launch
```

実行すると、rviz_controller.launch と同様に Rviz と joint_state_publisher のウィンドウが表示され、またこれとは別にシミュレーション画面が開きます。

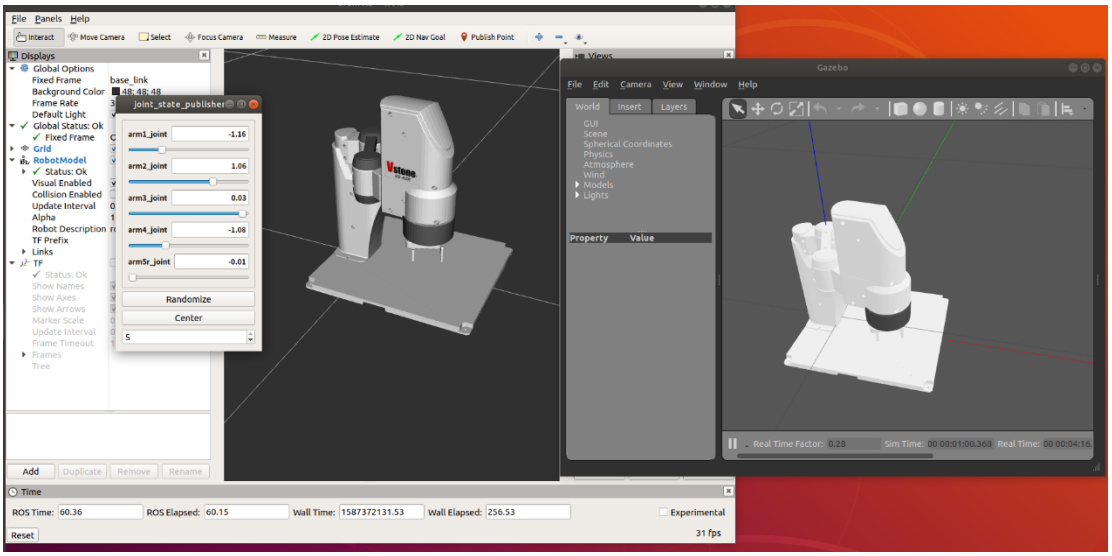


図 7-1 シミュレーションサンプルの実行画面例

joint_state_publisher のスライダーを操作すると、シミュレーション上のロボットが追従して動作します。なお、モータの駆動特性等について詳細を再現していないため、ロボットの実機と若干の差異が発生します。

パブリッシュ/サブスクライブしているメッセージはロボット実機を操作するものをそのまま利用できます。ただし、/scara_controller/poweron のみシミュレータでは対応しておりません。

8　MoveIt サンプル

「MoveIt」は、ROS 用の動作計画フレームワークです。主に物体の移動・運搬における経路設定を算出するために使用され、グラフィカルな環境・操作性で直感的に移動経路を指定できます。本 SDK では、MoveIt の基本的な設定とサンプルを scara_robot_moveit のパッケージにまとめています。

scara_robot_moveit のパッケージは、MoveIt の moveit_setup_assistant で作成されるパッケージをほぼそのまま収録しているため、本説明書で紹介している以外のサンプルも含まれますが、ロボット本体との連動が行われなかったり、環境によっては正常に動作しない場合があります。

本 SDK 用のサンプルとして、IK(逆運動学)を用いて PC の画面上からロボット本体を制御する launch ファイルを収録しています。サンプルの実行の際は、ロボット本体を PC に接続し、通信の権限を設定してから、端末より下記の通り入力してください。

```
roslaunch scara_robot_moveit demo_real.launch
```

起動すると、PC 上に下記の画面を表示し、ロボット本体が画面上の 3D モデルと同じポーズになります。

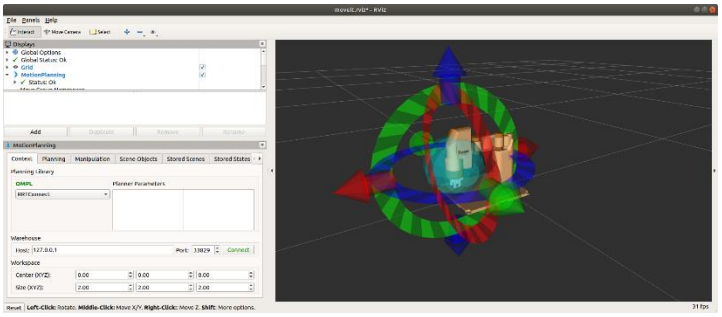


図 8-1　MoveIt のサンプル launch の起動画面

ロボットの手先付近に表示された球が、ロボットの移動目標位置を示します。この球及び球の周辺の矢印をマウスでドラッグすると、球の位置が移動し、それを追従するように画面上にオレンジ色の 3D モデルが表示されます。なお、ロボットが到達できない位置に球を移動させると追従されません。

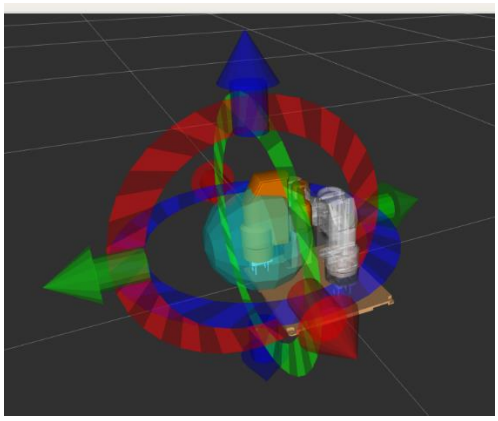


図 8-2　球を動かすとオレンジ色の 3D モデルが追従

ロボットアームの移動経路を算出する場合は、球を移動させてから、ウィンドウ左下の「MotionPlanning」内の「Planning」をクリックし、続いて「Command」内の「Plan」をクリックします。クリックすると、現在地から球の位置までの経路を自動で計算し、ロボットのモーションを生成します。生成されたモーションは画面上の 3D モデルに重なる形で描画されます。

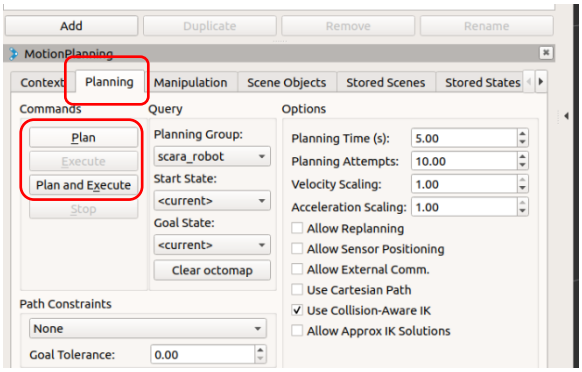


図 8-3　「Planning」内の「Plan」で移動経路を算出

生成したモーションを実行する場合は「Execute」をクリックします。クリックすると画面上の 3D モデル及びロボット本体が目標位置に向かって移動します。「Plan & Execute」をクリックすると、両方の操作が同時に行われます。

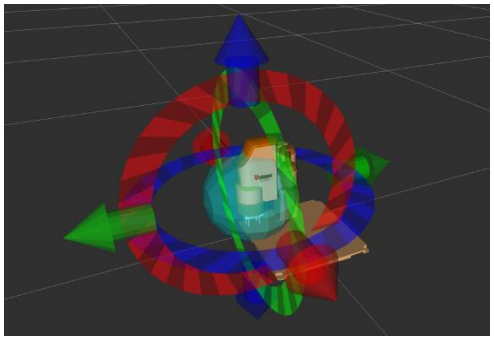


図 8-4　「Execute」で算出した経路を実行

9　動作確認バージョンについて

本 SDK の各サンプルが使用しているパッケージについては、以下のバージョンで動作確認を行っております。“apt”コマンドを用いてバージョン指定のオプションを付けずにバイナリを導入した場合、動作確認を行っていないバージョンが導入されシステムが正常に動作しない可能性がございます。また、Github からソースコードを取得してビルドする導入方法でも同様です。その際は、バージョン指定のオプションを用いて弊社にて動作確認を行っているバージョンのパッケージを導入してください。

表 9-1 動作確認パッケージのバージョン

パッケージ名	バージョン
joint-state-publisher	1.2.15
joint-state-publisher-gui	1.2.15
joint-state-controller	0.15.1
ros-control	0.17.0
effort-controllers	0.15.1
position-controllers	0.15.1
joint-trajectory-controller	0.15.1
moveit	1.0.2

商品に関するお問い合わせ

TEL: 06-4808-8701　　FAX: 06-4808-8702　　E-mail: infodesk@vstone.co.jp

受付時間　：9:00～18:00（土日祝日は除く）

ヴイストン株式会社　　www.vstone.co.jp

〒555－0012　大阪市西淀川区御幣島 2-15-28