Developers Note for ADAMR2 Project

一関高専 専攻科 生産工学専攻 2 年 藤野航汰

2020 年 11 月 4 日

目次

第 1 章

URDF を記述する

本章では，ROS においてロボットの構造をモデリングするためのフォーマットである URDF(Unified Robot Description Format) と，URDF をプログラマブルに記述することのできるマクロパッケージである xacro[*1] について解説します．

1.1 URDF とは？

URDF についての説明を MathWorks の Web サイトから引用します．[1]

> URDF (Unified Robotics Description Format) は、製造業の組み立てライン用ロボット マニピュレーター アームや遊園地用のアニマトロニクス ロボットなどのマルチボディ システムをモデル化するために、学術界や産業界で使用される XML 仕様です。

URDF では，ロボットのモデルをリンク (Link) とジョイント (Joint) からなるツリー構造で表現します．ボディやホイール，センサ等，駆動しないブロックをリンクとして扱い，リンクとリンクとの接続 (固定，回転，直動，etc) をジョイントとして扱います．URDF によってモデリングされたロボットは，最終的に図 1.1[2] のような構造になります．

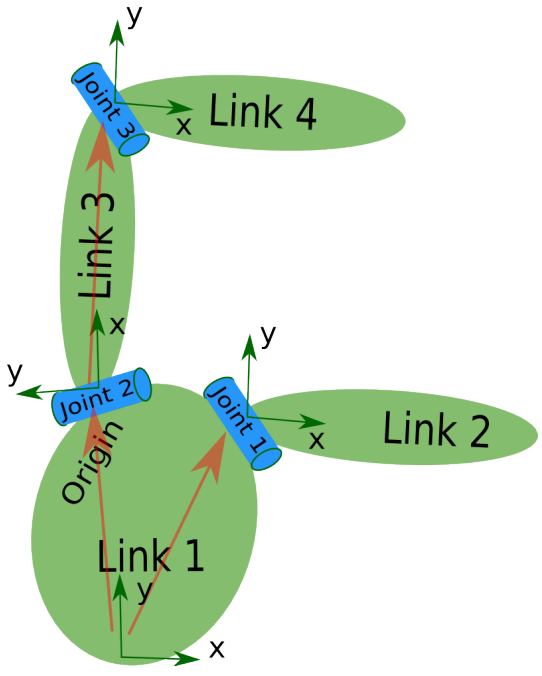

Fig.1.1 Tree Structure of URDF

URDF を記述することで，ロボットを構成する各リンクの位置関係やジョイントの属性を表すことができます．また，URDF の各リンク/ジョイントに詳細なオプションを追加することで，シミュレーション用モデル

[*1] http://wiki.ros.org/xacro

を作成することもできます．

1.2 何故 URDF を書くのか？

ROS の教本やチュートリアルでよく紹介されている URDF ですが，実のところ URDF を書かなくてもロボットの実機を動かすシステムを構築することができます．URDF が提供するのは，ロボット座標系におけるセンサやアクチュエータ等の位置関係を示す座標変換であり，これは TF パッケージを用いることでも実現できるものです．言い換えれば，TF を直接発行してロボットの各コンポーネントの座標変換を提供することができれば URDF を記述する必要は無いという事です．

では，URDF を記述する理由はあるのでしょうか？これはあくまで筆者の考えですが，以下のような利点があると考えます．

1. ロボットの各コンポーネントの位置関係を一括で管理することができる
2. ロボットの詳細な構造を可視化することができる
3. 実機と同じモデルでシミュレーションを行うことができる

URDF を記述することの大きな利点の 1 つとして，ロボットのモデルの可視化を行うことができるという点があると考えます．リンクやジョイントの座標変換を提供するだけなら TF パッケージを使うことで実現できるのですが，その場合モデルの可視化を行うことができません．そのため，入力した数値が正しいのかどうかを検証するのが困難になります．例えば，ロボットのハードウェアの設計を変更して，センサの位置が代わってしまったという場合を考えます．TF パッケージと URDF のどちらを使う場合でも，新しいセンサの位置を 3D CAD の設計データから計算して数値を導出するのは同じですが，TF パッケージを使う場合は可視化の手段が乏しいため，その数値が正しいかどうかを検証することが難しくなります．一方で URDF を使う場合は，URDF ファイルを編集[*2]し，更新したモデルを rviz で可視化することで，センサの位置が正しい位置にいるのかどうかを目で確かめることができます．

また，副産物的な考え方ですが，実機用の URDF ファイルにシミュレーションのためのオプションを追記することで，ロボットのシミュレーション環境を簡単に整えることが可能になります．ros_control[*3]のフレームワークと合わせることで実機とシミュレーションで同一のコントローラを使ってロボットを動かすことができるため，SLAM や Navigation 等のアプリケーションの開発を効率よく進められるようになります．

以上の理由から，ここでは URDF を記述してロボットのモデリングを行うことを強く推奨します．次の小節移行から，ADAMR2 で実際に使用している xacro ファイルをもとに，URDF によるロボットの実践的なモデリングについて解説します．

1.3 ADAMR2 の実機用 URDF モデリング

早速 xacro を用いて実際に ADAMR2 の実機用 URDF モデリングを行います．xacro を用いるため，URDF を直接記述することはありませんが，URDF の記述に関する基礎知識が必要になります．このセクションでは URDF 記述のチュートリアルについては取り扱わず，より実践的な内容を説明します．URDF モデリングのチュートリアルは Web 上に大量に存在します．この小節を読む前に，そちらを参照して基礎知識を身に付けてください．

これからモデリングするロボットのスケッチを図 1.2 に，リンクとジョイントのグラフを図に示します．

トップレベルのリンクとして base_footprint(空のリンク) があり，その下に base_link があります．ホイール，センサ，キャスターのリンクはすべて base_link の子リンクです．

1.4 URDF モデリング用パッケージの作成

1.4.1 ディレクトリ構成

ここから実際に URDF モデリングを行っていきます．まずは，URDF モデルを置くための ROS パッケージを作成します．慣習的に，ロボットの URDF モデルは*_description(*はロボットもしくはプロジェクトの名前) という名前の ROS パッケージに配置します．ADAMR2 プロジェクトならば「adamr2_description」という名前になります．URDF モデリングを始める前に，パッケージを作っておきましょう．ビルド依存パッケージは無いので，パッケージ作成コマンドにオプションは必要ありません．

[*2] 実際は xacro で記述した後に URDF をエクスポートすることが多いので，直接 URDF ファイルを編集することはありません．
[*3] http://wiki.ros.org/ros_control

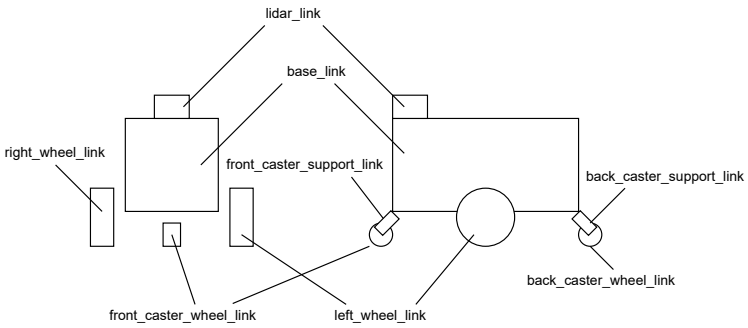

Fig.1.2　Link Structure of Diff-Drive Mobile Robot

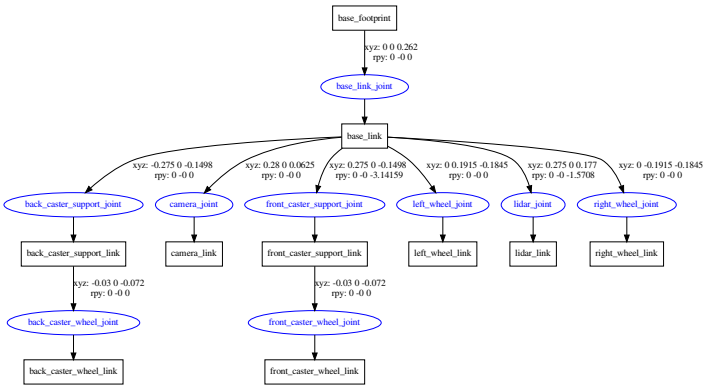

Fig.1.3　URDF Tree of ADAMR2

Code 1.1　Create a Package to put the URDF Model in

```
catkin create pkg adamr2_description
```

パッケージが作成できたら，中にいくつかのディレクトリを作成します．adamr2_description パッケージのディレクトリ構成はコード 1.2 のようにします．

Code 1.2　Directory Structure of adamr2_description

```
adamr2_description/
   ├ launch/
   ├ meshes/
   ├ rviz/
   ├ urdf/
   ├ package.xml
   └ CMakeLists.txt
```

launch/ディレクトリには，URDF モデルの可視化を行うための launch ファイルを置きます．meshes/ディレクトリには，STL 形式のロボットの 3D モデルを置きます．rviz/ディレクトリには，URDF モデル可視化時の rviz の設定ファイルを置きます．そして，urdf/ディレクトリに xacro ファイルを格納していきます．

urdf/ディレクトリの中身のディレクトリ構造はコードのようになります．

Code 1.3　Directory Structure of urdf/

```
urdf/
   ├ base/
   │    └ base.xacro
   ├ caster/
   │    └ caster.xacro
   ├ lidar/
   │    └ lidar.xacro
```

```
    ├ wheel/
    │    ├ transmission.xacro
    │    └ wheel.xacro
    └ robot.xacro
```

robot.xacro がルートファイルです．ルートファイルから各コンポーネントの xacro ファイルをインクルードし，ロボットのモデルを作成します．このファイルは最終的に xacro パッケージのノードを用いて URDF ファイルに変換されます．

ロボットの主要コンポーネントであるボディ，キャスター，LiDAR，ホイールのそれぞれに対してディレクトリを作り，その中に各々の xacro ファイルを格納します．ここで，キャスターはロボットの前後に 1 つずつ，ホイールはロボットの左右に 1 つずつ存在しますが，それぞれに対応する xacro ファイルを複数作成する必要はありません．共通のモジュールとして xacro ファイルを作成しておき，ルートファイルから読み込む際に名前や位置を付けることで各リンクを定義します．また，wheel には transmission.xacro というファイルがありますが，これは ros_control を用いたコントローラを作る際に必要となるオプションを設定するためのファイルです．

1.4.2 ルートファイルの作成

では早速，最初の xacro ファイルを作成してみましょう．ルートファイルである robot.xacro を記述してみます．urdf/ディレクトリの中に robot.xacro という名前の空のファイルを作成し，まずコード 1.4 のように記述します．

Code 1.4 robot.xacro

```
<?xml version="1.0"?>
<robot name="adamr2" xmlns:xacro="http://ros.org/wiki/xacro">

</robot>
```

URDF のルートファイルは，robot という要素のタグで始まる必要があり，他の全ての要素はこの robot タグの中に入っていなければなりません．[3] robot タグの name 属性にはロボットの名前を指定します．ここで指定した名前は後々使用することになるので，一意な名前をつけておきましょう．また，xmlns:xacro 属性にも値を設定しておきます．この値は ROS 固有のものなので，コピペして構いません．1 行目の文は XML のバージョン指定です．1.0 を指定しておきます．

これだけでは何も起きないので，robot タグ内に要素を追加してみましょう．コード 1.5 のように追記してみます．

Code 1.5 robot.xacro

```
<?xml version="1.0"?>
<robot name="adamr2" xmlns:xacro="http://ros.org/wiki/xacro">
  <link name="base_footprint"/>
</robot>
```

link タグを記述することにより，ロボットにリンクを追加することができます．本来，link タグは inertial，visual，collision の要素を持つのですが [?]，base_footprint リンクは実体を持たない空のリンクなので，空要素タグで記述します．

link 空要素タグの中で name 属性に値を代入しています．この name 属性につけた名前がリンクの名前となり，他のリンクから参照できるようになります．

1.4.3 URDF への変換と構文チェック

まだ 1 つのリンク (しかも空のリンク) しか追加していませんが，ひとまず URDF の構文チェックを行ってみましょう．ROS のツールに check_urdf というものがあり，これを使用すると URDF が正しい構造になっているかどうかをチェックすることができます．

しかし，xacro ファイルのままではチェックを行うことができません．[*4] そのため，まず xacro ファイルから

[*4] チェックツールを実行すること自体は可能ですが，URDF に含まれるリンク等が展開されないのでチェックにはなりません．

URDF ファイルに変換し，変換したファイルをチェックにかけることになります．
xacro ファイルを URDF ファイルに変換するには，xacro パッケージのノードを使用します．roscore を起動した状態で，コードのコマンドを実行します．

Code 1.6　Conversion from xacro to urdf

```
rosrun xacro xacro robot.xacro > robot.urdf
```

このコマンドによって，robot.xacro と同じディレクトリに robot.urdf が生成され，コード 1.7 に示すような内容が書き込まれます．

Code 1.7　Generated URDF file

```
<?xml version="1.0" encoding="utf-8"?>
<!-- =================================================================================== -->
<!-- |    This document was autogenerated by xacro from robot.xacro                    | -->
<!-- |    EDITING THIS FILE BY HAND IS NOT RECOMMENDED                                  | -->
<!-- =================================================================================== -->
<robot name="adamr2">
  <link name="base_footprint"/>
</robot>
```

生成された URDF ファイルの先頭に書かれている通り，このファイルを直接編集することは推奨されません．生成元となった xacro ファイルを編集して修正作業等を行います．
現時点での robot.xacro は空のリンクが 1 つだけのシンプルなファイルなので，URDF に変換しても大した違いはありません．とりあえずこの URDF ファイルをチェックにかけてみましょう．コード 1.8 を実行します．

Code 1.8　Check URDF

```
check_urdf robot.urdf
```

このコマンドは ROS ノードではないため，roscore が起動していなくても使うことができます．正しく記述できている場合，コード 1.9 に示すようなメッセージがターミナルに表示されます．

Code 1.9　Check Result

```
robot name is: adamr2
---------- Successfully Parsed XML ---------------
root Link: base_footprint has 0 child(ren)
```

ここまでの作業で，xacro ファイルの記述と URDF への変換，及び構文チェックまでを行うことができました．URDF を xacro によって記述するときは，このように URDF に変換してチェック作業を行いながら書くことになります．

1.4.4　URDF の可視化

urdf_to_graphiz コマンドを使えば，作成した URDF をグラフにして可視化することができます．コード 1.10 に示すコマンドを実行することで，URDF のツリー構造を可視化した PDF 画像ファイルを得ることができます．

Code 1.10　Visualize URDF Tree

```
urdf_to_graphiz robot.urdf
```

出力結果は図 1.4 のようになります．現時点では base_footprint リンクしか定義していないので，グラフのノードは 1 つだけです．

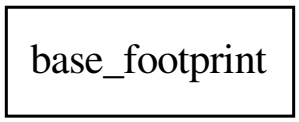

Fig.1.4　URDF Tree (base_footprint Only)

1.5　base_link の作成と追加

節 1.4 までの作業で，ルートファイル robot.xacro の作成と base_footprint リンクの定義を行いました．この節ではロボットのボディのリンクである base_link の定義とルートファイルへの追加の作業を行います．

1.5.1　base.xacro の作成

まず，base_link の要素や属性の定義を記述するファイルを作成します．urdf/ディレクトリ以下に base/ディレクトリを作成し，base.xacro という名前のファイルを作成します．そして，まずはコード 1.11 のような内容を記述します．

Code 1.11　base.xacro

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:macro name="base" params="parent *joint_origin">
  ..
```

ルートファイルと同じく，robot タグをトップレベルに記述し，その中に他の全ての要素を記述していきます．ただし，コンポーネントファイルには name 属性を書く必要はありません．名前空間だけ指定しておきましょう．

robot タグの中には，xacro:macro タグが書かれています．これが xacro におけるマクロの定義であり，ルートファイルから呼び出されるものです．マクロの中に link や joint 等のタグを記述しておけば，ルートファイルから呼び出されたときにそれらが展開される，という仕組みです．xacro:macro タグには name 属性と params 属性があります．name 属性にはマクロの名前を指定します．params 属性には，そのマクロが取る引数を指定することができます．引数は複数設定することができ，半角スペースで区切って記述します．コード 1.11 では，引数として parent(親リンク名の指定) と*joint_origin [*5](座標原点の指定) の 2 つを取るように設定しています．

base.xacro ファイルでは，このマクロを読み込んだら「base_link の定義」と「base_footprint と base_link を繋ぐジョイント base_link_joint の定義」が行われるようにマクロを定義します．

1.5.2　joint の設定

この小節では base_link_joint の定義を行います．コード 1.12 のようにマクロタグ内に joint タグを追記します．

Code 1.12　Configurate base_link_joint

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:macro name="base" params="parent *joint_origin">
```

[*5] アスタリスクが先頭に付くパラメータは「ブロックパラメータ」と呼ばれるパラメータで，任意の要素を展開することができます．

```
    <joint name="base_link_joint" type="fixed">
      <xacro:insert_block name="joint_origin"/>
      <parent link="${parent}"/>
      <child link="base_link"/>
    </joint>
  </xacro:macro>
</robot>
```

joint タグは name と type の 2 つの属性を持ち，また以下の要素を持ちます．[4]

- origin：親リンクから見たジョイントの原点座標
- parent：親リンク名
- child：子リンク名
- axis：ジョイントの軸．回転ジョイントの場合は回転軸を表す
- calibration：ジョイントの絶対位置を校正するために使用される，ジョイントの基準位置
- dynamics：ジョイントの物理的特性を指定するための要素
- limit：ジョイントの角度，力，速度のリミットを指定するための要素
- mimic：他のジョイントの動きを真似させるときに使用する要素
- safety_controller：ジョイントが安全に動作するために設定するソフトなリミット

このうち，ジョイントの定義に必須なのは parent と child のみで，他の要素は指定しなくてもジョイントを定義することができます．

ここでは，ジョイント名を「base_link_joint」，ジョイントタイプを「fixed」とし，要素には parent と child，そして origin を設定します．

child 要素は base_link です．base_link はまだ存在せず，1.5.3 で定義します．parent 要素にはマクロが引数として受け取った値をそのまま使用します．引数をマクロ内で利用するには，${PARAMETER_NAME } の書式を使用します．

joint タグの origin 要素で指定するのは，親リンクから見たジョイントの原点座標です．図 1.5.2 のように，ジョイントが子リンクの原点になります．ここでは base_link が子リンクなので，base_link の座標系原点は base_link_joint の位置になるという事になります．

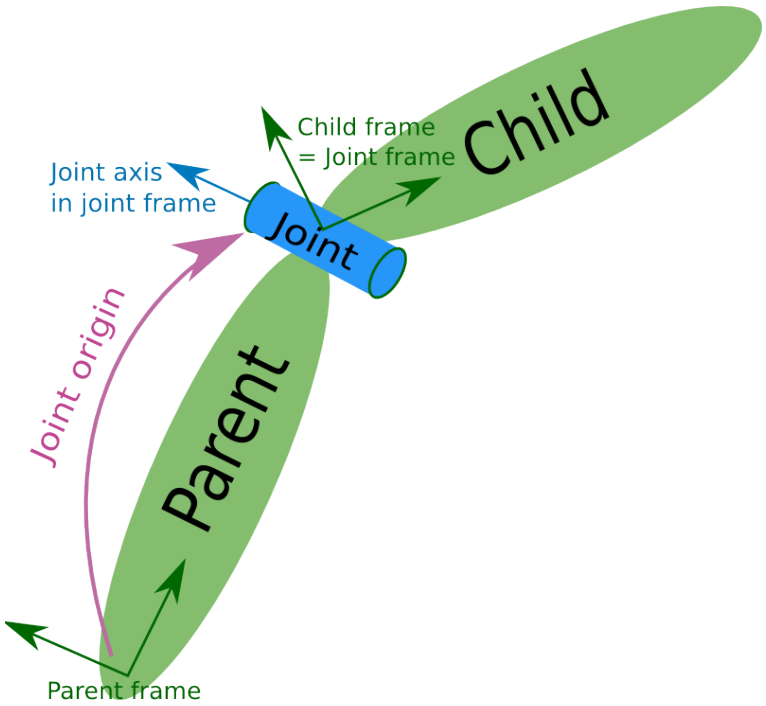

Fig.1.5　Coordinate Transformation Relationship Between Link and Joint[4]

origin 要素もマクロが引数として受け取ったものを使用します．ただし，origin 要素は単なる値ではないため，parent 要素のようにそのまま使うことはできません．そこで使用するのがブロックパラメータと呼ばれる引数です．これを使用することで，ルートファイルからジョイントの origin 要素を定義できるようになります．

コード 1.12 では*joint_origin という名前でブロックパラメータを定義しています．ルートファイルから読み込む際の詳細はで説明します．

ジョイントの設定は以上で完了です．次は link タグの設定を行います．

1.5.3 link タグの設定

ここから link タグを定義して，base_link の要素を記述していきます．link タグは inertial，visual，collision の 3 つの要素を持ちます．link タグ及び各要素の詳細について知りたい方は，ROS Wiki のページ[*6]を参照してください．

Gazebo シミュレーションを行う際は全ての要素について定義する必要がありますが，rviz で可視化するだけなら visual 要素を設定するだけで事足ります．本章では実機用 URDF の記述を目的としているため，ここではシミュレーション用の設定については説明しません．シミュレーション用の設定については後の章で解説します．

visual 要素はモデルの見た目の要素を定義するタグです．コード 1.13 のようにマクロタグ内に link タグを追記し，更にその中に visual 要素を記述します．

Code 1.13 link Element

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:macro name="base" params="parent *joint_origin">
    <joint name="base_link_joint" type="fixed">
      <xacro:insert_block name="joint_origin"/>
      <parent link="${parent}"/>
      <child link="base_link"/>
    </joint>

    <link name="base_link">
      <visual>
        <origin xyz="0.0 0.0 0.0"/>
        <geometry>
          <mesh filename="package://adamr2_description/meshes/base_link.STL"/>
        </geometry>
        <material name="red">
          <color rgba="1.0 0.0 0.0 1.0"/>
        </material>
      </visual>
    </link>
  </xacro:macro>
</robot>
```

visual 要素は更に origin，geometry，material 要素を持ちます．geometry 要素ではリンクの見た目を表す形状を定義します．長方形，球，円柱等のプリミティブな図形を設定できる他，STL ファイルを読み込んで設定することができます．実際，コード 1.13 では mesh 要素に STL ファイルを指定して形状を設定しています．STL ファイルは 1.5.4 で作成します．

origin 要素では geometry 要素で指定した図形の原点の座標を設定することができます．リンクの座標系から見た位置，即ちこのリンクが所属するジョイントの原点からみた位置で指定することになります．このドキュメントでは，基本的にリンクの位置は joint タグの位置を設定することで決定します．そのため，link タグの origin 要素の値はほとんどの場合でゼロになります．

material 要素ではリンクの色を設定することができます．ただし，プリミティブ図形または STL ファイルを使用する場合は，単色でしか設定できません．リンクの見た目の色を詳細に決定したい場合は，COLLADA ファイル (拡張子は*.dae) を用意する必要があります．ロボットの開発にとってあまり本質的ではないと考えたので，このドキュメントでは COLLADA ファイルの作成については説明しません．

1.5.4 メッシュファイル (STL 形式) の作成と割り当て

visual 要素の mesh に指定する STL ファイルを，3D-CAD ソフトである SolidWorks2019 を用いて作成します．SolidWorks にはパーツファイルを STL にエクスポートする機能があるので，これを使います．

[*6] http://wiki.ros.org/urdf/XML/link

ROS REP:103[*7]にあるように，ROS では長さの単位を全てメートルで扱います．しかし，3D-CAD で設計を行うときはミリメートル単位でモデリングするのが一般的です．従って，そのまま STL エクスポートすると，ROS から読み込んだときにモデルの大きさが 1000 倍になってしまいます．また，ROS ではロボットの座標軸の向きは「X 軸がロボットの前進方向，Z 軸は鉛直上向き方向，Y 軸は右手座標系となるような向き」と決められています．3D-CAD のデフォルト座標軸がこのような向きになっていることは少ないため，STL ファイルをエクスポートした後に Blender 等の 3D-CG ファイルでスケールや向きを調整しなければならない場合が殆どです．

しかし，SolidWorks には STL ファイルをエクスポートする際に，単位系や座標系を指定することができる機能があります．そのため，適切な設定を行いさえすれば 3D-CG ソフトで後処理を行うことは必要ありません．

図 1.5.4 に，base_link に対応付けるロボットのボディの 3D-CAD モデルを示します．これはアセンブリファイルをパーツファイルに変換したもので，筐体の他，アクチュエータ (ホイールは含まない) も内包しています．3D-CAD の座標軸が図の左下に表示されていますが，X 軸はロボットの前方を向いているものの，Z 軸が鉛直上向きになっていません．また，ミリメートル単位で設計を行ったため，単位変換を行う必要があります．

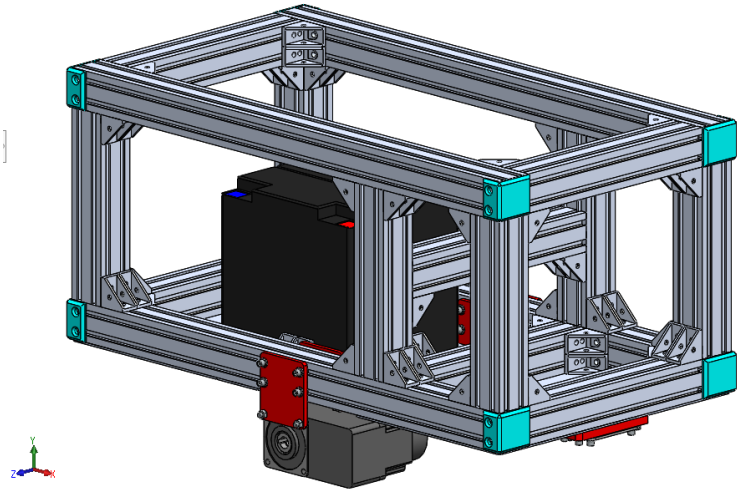

Fig.1.6　3D Model of base_link

STL ファイルをエクスポートする前の準備として，まず座標系を定義しましょう．今回のモデルでは，原点はアルミフレームが描く直方体の中心に位置しているので，STL ファイルの原点にしたい場所に新たに点を打つ必要はありません．まずは座標系のもとになる軸を定義します．フィーチャー → 参照ジオメトリ → 軸を選択します．そして，図 1.5.4 のように 2 つの平面を選択して，X 軸，Y 軸，Z 軸の 3 本を新たに定義します．軸を追加したら，座標系を定義します．先ほど追加した軸を使って，ROS が推奨する座標軸の向きになるように設定します．少し見づらいですが，図 1.5.4 では X 軸に軸 1 を，Z 軸に軸 3 を対応付けた結果，望ましい座標軸の向きになっていることがわかります．

座標系の定義ができたので，STL ファイルをエクスポートしましょう．指定保存を開き，ファイルの種類 (拡張子) に「STL」を選択します．STL を選択すると，「ｵﾌﾟｼｮﾝ」ボタンが表示されるので，それをクリックします．

オプションを開くと，図 1.5.4 のような画面が表示されます．ここで単位と出力座標系を設定します (1.5.4 で赤線を引いたところ)．単位をメートルに設定し，出力座標系に先ほど作成した座標系 (ここでは「座標系 1」という名前) を設定します．お好みで解像度の設定もできます．「粗表示」を選ぶとモデルが粗くなってしまいますが，ファイルの容量を減らすことができます．

この設定で「OK」をクリックし，「保存」を選択することで，ROS でそのまま使用できる STL ファイルを作成することができます．この STL ファイルを，adamr2_description/meshes/ディレクトリに base_link.STL という名前で保存します．コード 1.13 が正しく記述されていれば，STL ファイルを読み込むことができるようになります．

[*7] https://www.ros.org/reps/rep-0103.html

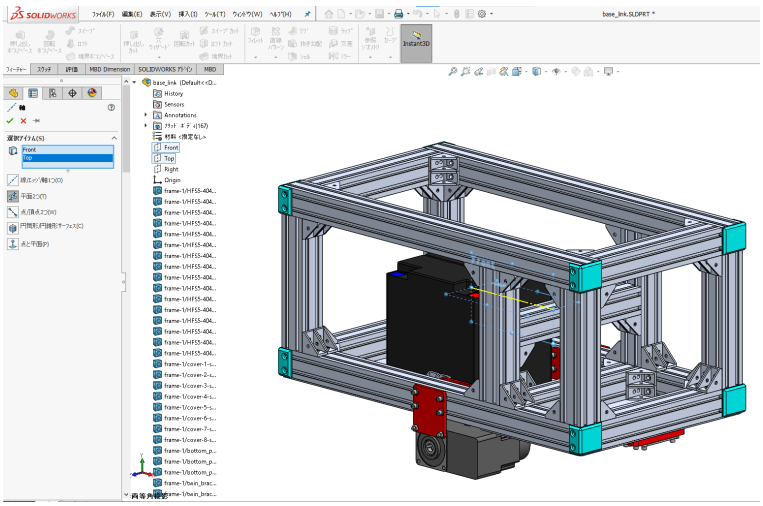

Fig.1.7　Add Axes for STL Export

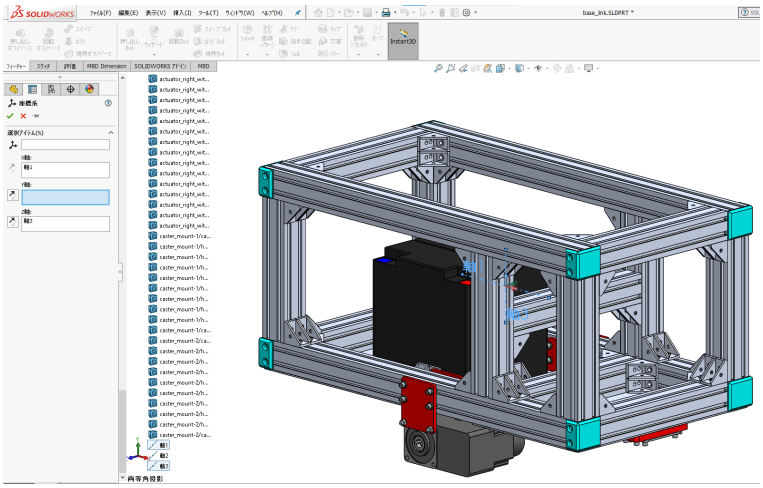

Fig.1.8　Add Coordinate System for STL Export

1.5.5　ルートファイルからインクルードする

ここまでの作業で base_link の定義を完了することができたので，これをルートファイルから読み込んでロボットモデルに組み込みましょう．コード 1.14 に，robot.xacro にいくつかの文を追記します．

Code 1.14　Add base_link to Robot Model

```
<?xml version="1.0"?>
<robot name="adamr2" xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:include filename="$(find adamr2_description)/urdf/base/base.xacro"/>

  <link name="base_footprint"/>

  <xacro:base parent="base_footprint">
    <origin xyz="0.0 0.0 0.262"/>
  </xacro:base>

</robot>
```

3 行目に追記したコマンド xacro:include で，base.xacro ファイルを読み込んでいます．これにより，base.xacro で定義したマクロを使用できるようになります．そして，7 行目から 9 行目で xacro:base マクロを使って base_link と base_link_joint を定義しています．マクロタグ内で origin 要素の記述を行っているの

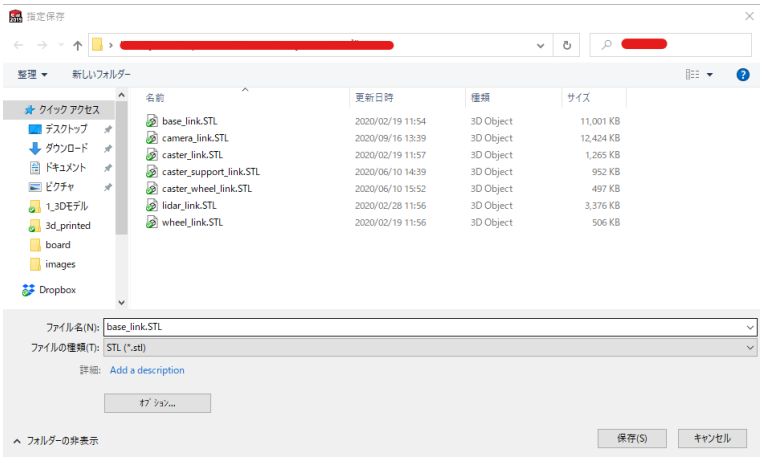

Fig.1.9　Save as STL File

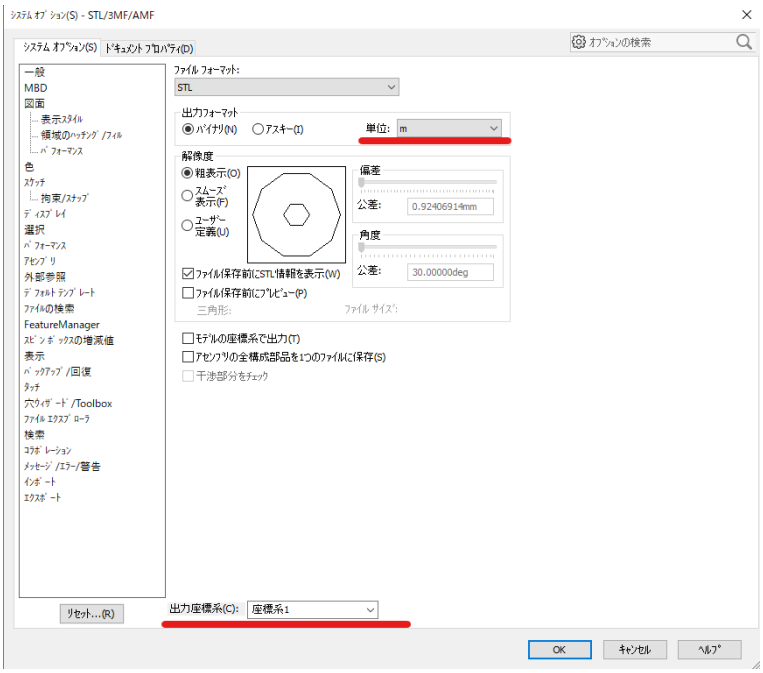

Fig.1.10　STL Export Options

は，マクロがブロックパラメータとして origin 要素を引数としているからです．こうすることにより，設計変更が発生してリンクの場所を変更しなければならなくなっても，robot.xacro を編集するだけで対応することが出来るようになります．

robot.xacro を編集したら，念の為 URDF チェックをしておきましょう．URDF の構文チェックの手順は，1.4.3 の手順を参照してください．URDF を正しく記述できていれば，コード 1.15 のように表示されるはずです．

Code 1.15　Result of Check URDF

```
robot name is: adamr2
---------- Successfully Parsed XML ---------------
root Link: base_footprint has 1 child(ren)
    child(1):  base_link
```

1.5.6 rviz による可視化

せっかくリンクを追加したので，rviz で可視化してみましょう．launch/ディレクトリを作成し，その中に rviz を起動するための launch ファイル display.launch を作成し，コード 1.16 の内容を記述します．

Code 1.16 display.launch

```
<launch>
  <arg name="model" default="$(find adamr2_description)/urdf/robot.xacro"/>

  <param name="robot_description" command="$(find xacro)/xacro $(arg model)"/>

  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher
    "/>
  <node name="rviz" pkg="rviz" type="rviz"/>
</launch>
```

param タグで robot.xacro を展開し，robot_description という名前のパラメータに登録します．ROS ではロボットのリンクやジョイントの状態を，robot_description という名前のパラメータで一元管理しています．URDF の情報をこのパラメータに登録することで，URDF で定義したロボットのモデルを ROS に伝えます．

また，この launch ファイルでは 2 つのノードを起動しています．1 つは robot_state_publisher ノードです．このノードは robot_description パラメータの情報を解釈して TF を投げてくれるノードです．2 つ目は rviz です．

launch ファイルを作成したら，一旦パッケージをビルドしておきましょう．パッケージをビルドすることで，そのパッケージに launch ファイルが存在することをシステムに登録できます．[*8]ROS のワークスペースに移動して，コード 1.17 のコマンドを実行します．

Code 1.17 Build adamr2_description Package

```
cd ~/catkin_ws/src
catkin build adamr2_description
source ~/catkin_ws/devel/setup.bash
```

パッケージをビルドしたら，launch ファイルを実行してみましょう．

Code 1.18 Launch display.launch

```
roslaunch adamr2_description display.launch
```

launch ファイルを実行すると，rviz が起動し，図 1.5.6 のような画面が表示されます．

起動した直後はオプションを何も設定していないので，何も表示されません．ロボットのモデルを表示するためには，左下の「Add」ボタンを押し，「RobotModel」を追加します (図 1.5.6)．

そして，Fixed Frame の値を base_footprint に変更します．デフォルトの値は map というフレームが割り当てられていますが，ロボットのモデルには map というフレームは存在しません．ロボットモデルのルートとなるフレーム名は base_footprint に設定しているので，プルダウンメニューを開き，base_footprint を選択します．

ここまで設定すれば，図 1.5.6 のようにロボットのモデルが正しく表示されるはずです．base_link がホイールの大きさを考慮した高さに置かれていることが分かると思います．

本節では，コンポーネントとしての xacro ファイルの記述の仕方，SolidWorks を利用した STL ファイルの作成方法，ルートファイルからのインクルードの仕方，及び rviz を使った URDF モデルの可視化について説明しました．センサやホイール等のコンポーネントも，本節で説明した手順の通りに追加することができます．次節以降の作業を行う際にづまづいたときは，本節に戻って手順を確認してください．

1.6 caster_*_link の作成と追加

ロボットのモデルに自在キャスターを追加しましょう．実機用のモデルには必要のないリンクですが，後々シミュレーションを行うときに必要になるので，今のうちに追加しておきます．

[*8] ROS1 では，ROS2 のように launch ファイルを変更する度にビルドする必要はありません．ここでの作業は launch ファイルの存在をシステムに知らせるためのものです．

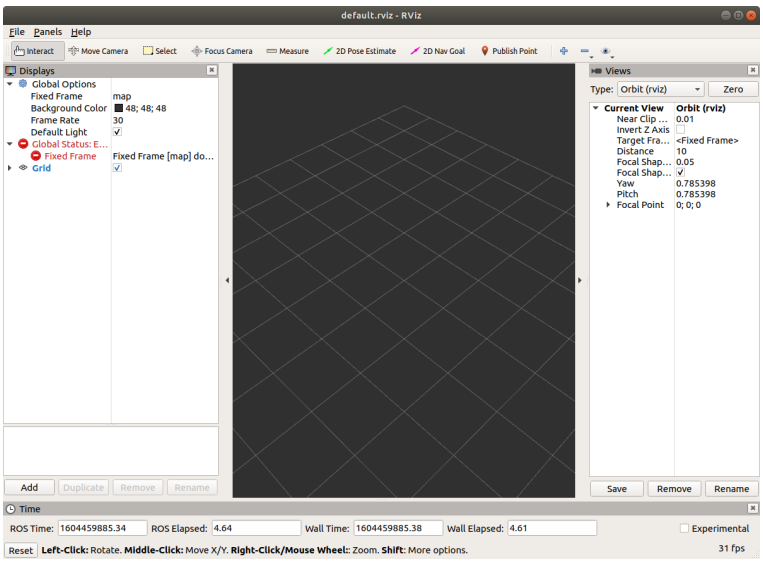

Fig.1.11　Screenshot of rviz Right After It Started

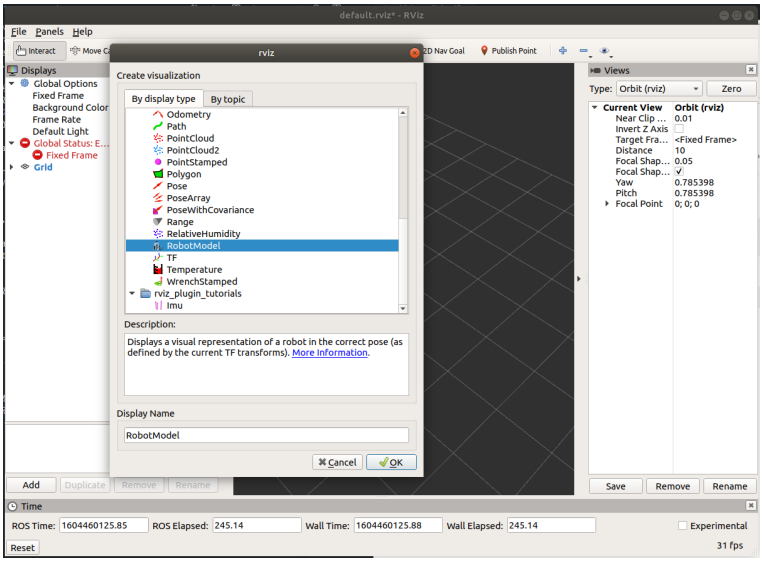

Fig.1.12　Add RobotModel to rviz

自在キャスターは 2 つのパーツから構成されます．1 つ目はキャスターの車輪パーツ，2 つ目は車輪を支えるサポートパーツです．シミュレーションを行うことを考えて，これらのパーツを別々のリンクとして定義することにします．

1.6.1　caster.xacro の作成

urdf/ディレクトリ以下に caster/ディレクトリを作り，その中に caster.xacro という名前のファイルを作成します．そして，コード 1.19 の内容を記述します．

Code 1.19　caster.xacro

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:macro name="caster" params="prefix parent *joint_origin">
    <joint name="${prefix}_caster_support_joint" type="fixed">
      <xacro:insert_block name="joint_origin"/>
      <parent link="${parent}"/>
      <child link="${prefix}_caster_support_link"/>
```

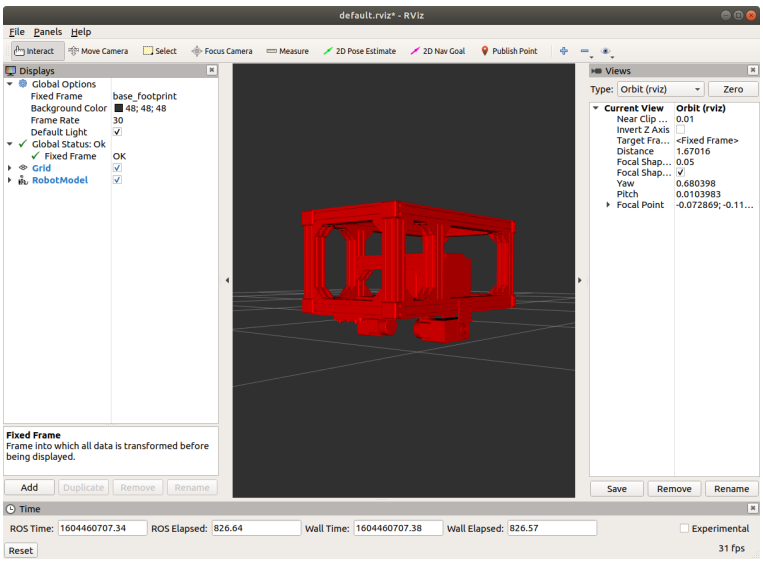

Fig.1.13　base_link is Correctly Displayed

```
    </joint>

    <link name="${prefix}_caster_support_link">
      <visual>
        <origin rpy="0.0 0.0 0.0"/>
        <geometry>
          <mesh filename="package://adamr2_description/meshes/caster_support_link.STL"/>
        </geometry>
        <material name="red">
          <color rgba="0.0 0.0 0.0 1.0"/>
        </material>
      </visual>
    </link>

    <joint name="${prefix}_caster_wheel_joint" type="fixed">
      <origin xyz="-0.030 0.0 -0.072" rpy="0.0 0.0 0.0"/>
      <parent link="${prefix}_caster_support_link"/>
      <child link="${prefix}_caster_wheel_link"/>
    </joint>

    <link name="${prefix}_caster_wheel_link">
      <visual>
        <origin rpy="0.0 0.0 0.0"/>
        <geometry>
          <mesh filename="package://adamr2_description/meshes/caster_wheel_link.STL"/>
        </geometry>
        <material name="red">
          <color rgba="0.0 0.0 0.0 1.0"/>
        </material>
      </visual>
    </link>
  </xacro:macro>
</robot>
```

caster.xacro では，${prefix}_caster_support_link と${prefix}_caster_wheel_link の 2 つのリンクと，それらを繋ぐ${prefix}_caster_support_joint と${prefix}_caster_wheel_joint の 2 つのジョイントを定義するマクロ caster を定義しています．caster マクロは 3 つの引数を取ります．このうち，prefix はリンクやジョイントの名前の接頭辞です．自在キャスターはロボットの前後に 1 つずつ存在しますが，それらは全く同じパーツなので，xacro ファイルを複数定義するのは無駄になります．そのため，接頭辞を引数にとって区別することで，1 つの xacro ファイルで 2 つの部品を定義できるようにしています．

base_link と${prefix}_caster_support_link は${prefix}_caster_support_joint によって接続されます．ジョイン

トタイプは「Fixed」です．本来ならばこのジョイントは，Z 軸を回転中心とする連続回転ジョイントのタイプにする必要があります．しかし，実機用の URDF ファイルでこのような受動回転するコンポーネントのジョイントタイプを連続回転にすると，リンクの位置関係を取得できないために TF のエラーが発生します．そのため，実機用 URDF ではジョイントのタイプを「Fixed」にしています．${prefix}_caster_support_link と${prefix}_caster_wheel_link を繋ぐジョイント${prefix}_caster_wheel_joint も同様に固定タイプのジョイントとしています．

サポートパーツと車輪パーツの位置関係は一定なので，xacro ファイル内でローカルに${prefix}_caster_wheel_joint の原点を設定してしまっています．一方で，自在キャスターがロボットボディのどこに取り付けられるかはユーザー次第なので，${prefix}_caster_support_joint の原点はルートファイルから決定できるように，マクロのブロックパラメータとして設定しています．

1.6.2 STL ファイルの作成と割り当て

1.5.4 を参考に，3D-CAD ソフトから STL ファイルをエクスポートして meshes/ディレクトリに配置してください．

1.6.3 ルートファイルからインクルードする

作成したリンクをロボットモデルに組み込みましょう．robot.xacro にコード 1.20 のように追記します．

Code 1.20 Add Caster-Link to Robot Model

```
<?xml version="1.0"?>
<robot name="adamr2" xmlns:xacro="http://ros.org/wiki/xacro">
  <xacro:include filename="$(find adamr2_description)/urdf/base/base.xacro"/>
  <xacro:include filename="$(find adamr2_description)/urdf/caster/caster.xacro"/>

  <link name="base_footprint"/>

  <xacro:base parent="base_footprint">
    <origin xyz="0.0 0.0 0.262"/>
  </xacro:base>

  <xacro:caster prefix="back" parent="base_link">
    <origin xyz="-0.275 0.0 -0.1498"/>
  </xacro:caster>

  <xacro:caster prefix="front" parent="base_link">
    <origin xyz="0.275 0.0 -0.1498" rpy="0 0 ${radians(180)}"/>
  </xacro:caster>
</robot>
```

caster.xacro ファイルをインクルードし，xacro:caster マクロを使って前後の自在キャスターを定義しています．ロボットの前方に付けるキャスターは，向きを 180 度変えるために origin 要素の rpy 属性でヨー角を設定しています．

第 2 章

デバイス設定

2.1　F310/F710 ゲームパッドを ROS で利用する

2.1.1　製品の概要

Logicool Gamepad F310/F710 は，Logicool から発売されている USB 接続の PC 用ゲームパッドです.

Fig.2.1　Logicool F710 Gamepad

有線接続の F310 は 2310 円，無線接続の F710 は 4950 円と安価に入手することができ，ロボットの操縦用コントローラとして使用することもできます.

2.1.2　ゲームパッドの入力モード

310/F710 ゲームパッドは 2 つの入力モードを持っており，物理スイッチによってモードを切り替えることができます.

- DirectInput モード：他のゲームパッドと同じ挙動を示すモード
- XInput モード：Xbox360 コントローラを PC に繋いだときと同じ挙動を示すモード

モード切替スイッチは，F310 には本体背面に，F710 は本体側面上部に付いています.

Fig.2.2　Mode Selector Switch for F710

ROS パッケージの joy[*1]から F310/F710 ゲームパッドを使用するときは，DirectInput モードを使用する必要があります．

2.1.3 DirectInput モードのキーマッピング

DirectInput モードで ROS に接続した際の F310/F710 ゲームパッドのキーマッピングを表 2.1 及び表 2.2 に示します．

sensor_msgs/Joy[*2]型メッセージでは，ゲームパッドの各軸・各ボタンの信号の値が axes と buttons の 2 つのリストに格納されます．各リストに対してインデックスを指定することで，対応するボタン・軸のデータを得ることができます．

Table2.1 Button Mapping of F310/F710 Gamepad in ROS

Buttons	X	A	B	Y	LB	RB	LT	RT	BACK	START	LeftStick	RightStick
index	0	1	2	3	4	5	6	7	8	9	10	11

Table2.2 Axis Mapping of F310/F710 Gamepad in ROS

Axis	Left Horiz.	Left Vert.	Right Horiz.	Right Vert.	Arrow Horiz.	Arrow Vert.
index	0	1	2	3	4	5

[*1] http://wiki.ros.org/joy
[*2] http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/Joy.html

参考文献

[1] MathWorks. URDF 入門. https://jp.mathworks.com/help/physmod/sm/ug/urdf-model-import.html.

[2] Open Source Robotics Foundation. urdf/XML/model - ROS wiki. http://wiki.ros.org/urdf/XML/model.

[3] Open Source Robotics Foundation. urdf/XML/robot - ROS wiki. http://wiki.ros.org/urdf/XML/robot.

[4] Open Source Robotics Foundation. urdf/XML/joint - ROS wiki. http://wiki.ros.org/urdf/XML/joint.