SEED Solutions
Smart RT System for Service Robots

NEDO講座
第２部
ROSでのSEED-Noid制御基礎

THK株式会社
産業機器統括本部 技術本部
事業開発統括部 永塚BU

第２部の目的

- ROS上でSEED-Noidを動作させるための基本手順について学ぶ.
- 最初に基本的なノードを起動するlaunchファイルを眺めながら，SEED-Noidの制御のために動作しているノードについて概説する.
- 続いて，SEED-Noidに関連するROSノードの起動と，最終的に目指す動作をイメージするためのデモの動作を行う.

SEED-Noidのlaunchファイル

seed_r7_bringup.launch

SEED-Noidのベースとなるノードの起動を行うlaunchファイル.

このlaunchファイルを通じて，launchファイルの書き方と，SEED-Noidで利用しているROSパッケージの基礎を理解する.

以降の説明はファイルを開いて，見比べながら見てください.

```
$cd ~/ros/melodic/src/seed_r7_ros_pkg/seed_r7_bringup/launch
$gedit seed_r7_bringup.launch
```

launchファイル概説(1)

1行目　　<?xml version="1.0"?>
XMLファイルであることの宣言．XMLファイルでは必ず最初にこの宣言をする．
(HTMLでも同様の定義をするし，Pythonのスクリプトでも1行目で同様の定義をする．)

2行目　　<launch>
58行目　　</launch>
ROSのlaunchファイルである事を定義するタグ．

XMLなどではこのように”<>”で囲ったものをタグといい，どこからどこまでがこのタグの表す内容かわかるように，”<launch>” ~ “</launch>”のように定義の開始と終了がわかるようにする．
なお，1行で書く場合は<aa ...　/>とすることもできる．

launchファイル概説(2)

3行目～15行目

```
<!-- parameters when using models in official package -->
<arg name="robot_model"         default="typef"/>
<arg name="robot_model_plugin" default="seed_r7_robot_interface/$(arg robot_model)"/>

<!-- parameters when using models in user packages -->
<arg name="csv_config_dir"      default="$(find seed_r7_description)/csv"/>
<arg name="robot_joint_config"  default="$(find seed_r7_robot_interface)/$(arg robot_model)/config/joint_settings.yaml"/>
<arg name="controller_settings" default="$(find seed_r7_description)/$(arg robot_model)/controller_settings.yaml"/>
<arg name="moveit_config_pkg"   default="$(find seed_r7_moveit_config)/../seed_r7_$(arg robot_model)_moveit_config"/>

<!-- other settings -->
<arg name="controller_rate"     default="50"/>
<arg name="overlap_scale"       default="2.0"/>
```

ここで示される<arg>タグは引数の定義を行っている．例として，”robot_model_plugin”という引数に，デフォルト値として，”seed_r7_robot_interface/typef”というものをいれるということを宣言している．

なお，ここで言う”typef”とはSEED-Noidのモデルであり，SEED-Noid-Moverの標準のモデルを表している．
(カスタマイズ品にはそれぞれ別の名称がつけられている．)

http://wiki.ros.org/roslaunch/XML/arg

launchファイル概説(3)

17行目～21行目

```
<!-- controller settings / load from yaml -->
<rosparam command="load" file="$(arg controller_settings)" />
<rosparam command="load" file="$(arg robot_joint_config)" />
<rosparam command="load"
    file="$(find seed_r7_robot_interface)/$(arg robot_model)/config/extra_controller_settings.yaml" />
```

rosparamはROSノードの外部から変更可能なパラメータ.
<rosparam>タグはrosparamの書かれたYAMLファイルを読み込ませることができる.

YAMLとは，XMLのようなメタ言語の1つであるが，XMLのように明示的にタグを設定せずに，インデントなどで階層構造を示すことができるメタ言語.

ここでは，SEED-Noidに関連するコントローラの設定や，各種YAMLファイルを読み込ませている.
(各YAMLファイルの詳細などは各自で調べて見てください．)

http://wiki.ros.org/roslaunch/XML/rosparam

Launchファイル概説(4)

23行目～30行目

```
<node pkg="seed_r7_ros_controller" type="seed_r7_ros_controller" name="seed_r7_ros_controller" output="screen">
  <param name="port_lower" value="/dev/aero_lower"/>
  <param name="port_upper" value="/dev/aero_upper"/>
  <param name="csv_config_dir" value="$(arg csv_config_dir)"/>
  <param name="robot_model_plugin" value="$(arg robot_model_plugin)"/>
  <param name="controller_rate" value="$(arg controller_rate)"/> <!-- [ Hz ] ( rate of read/write cycle) -->
  <param name="overlap_scale"   value="$(arg overlap_scale)"/>   <!-- scaling of target time -->
</node>
```

ここでは，SEED-Noid本体のハードウェア制御を行うノードを起動している．
“pkg”はパッケージ名，”type”はノードの実行ファイル名，”name”はroscoreに登録するノード名を表す．
“output”は標準出力，標準エラー出力の表示先を示しており，”screen”はlaunchを実行したターミナルに出力するという宣言．

Launchファイル概説(5)

32行目～36行目

```
<rosparam>
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50
</rosparam>
```

ちょっと特殊なインラインでの書き方.
YAMLの文法に従い，直接値を書き込む形での記述になっている.
わざわざ設定をファイルで書く必要がない場合や，高頻度でパラメータを変えない場合などはこのような記述も利用可能.

Launchファイル概説(6)

38行目～40行目

```
<include file="$(arg moveit_config_pkg)/launch/planning_context.launch">
  <arg name="load_robot_description" value="true"/>
</include>
```

<include>タグは，他のlaunchファイルを読み込むときに利用する.
<include>タグで指定したlaunchファイルの中の変数を変更したい場合は<arg>タグでその変数名を指定し，値を代入して上げることで変更ができる.

Launchファイル概説(7)

42行目

```
<node name="seed_r7_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />
```

robot_state_publisherというノードを
seed_r7_state_publisherと名前を変更してノードの起動をする.

robot_state_publisherはROSにおける特殊な(?)ノードの1つで,
robot_controllerから受け取る現在の各エンコーダ情報をロボットの構造に合わせて変換しpublishしてくれるノード.
ロボットのモデル(urdfファイル)の構造に従い，各関節の3次元位置・姿勢を取得できる.

以下の様に，分けられたトピックを出力
可動関節　tf
非可動関節　tf_static

Launchファイル概説(7)

44行目～56行目

```
<!-- spawing joint controllers -->
<!-- names of controllers depend on robot type (described at controller_settings.yaml) -->
<group if="$(eval not robot_model.endswith('arm') )"> <!-- in case of dual arm -->
  <node name="cm_spawner" pkg="controller_manager"
        type="spawner" args="joint_state_controller larm_controller rarm_controller
                             head_controller waist_controller lifter_controller
                             lhand_controller rhand_controller" />
</group>
<group if="$(eval robot_model.endswith('arm') )"> <!-- in case of single arm -->
  <node name="cm_spawner" pkg="controller_manager"
        type="spawner" args="joint_state_controller arm_controller hand_controller
                             lifter_controller" />
</group>
```

ここでは，双腕の場合と短腕の場合でcontroller_managerにわたすハードウェア要素名を変更している．

Groupタグは通常，同じ機能のノードを起動する際に，名前空間を変更する場合などに利用されるが，ここでは，指定されたハードウェアに合わせた構成を読み込むための条件分岐をするために利用している．

Launchファイル概説(8)

44行目～56行目

```
<!-- spawing joint controllers -->
<!-- names of controllers depend on robot type (described at controller_settings.yaml) -->
<group if="$(eval not robot_model.endswith('arm') )"> <!-- in case of dual arm -->
  <node name="cm_spawner" pkg="controller_manager"
        type="spawner" args="joint_state_controller larm_controller rarm_controller
                             head_controller waist_controller lifter_controller
                             lhand_controller rhand_controller" />
</group>
<group if="$(eval robot_model.endswith('arm') )"> <!-- in case of single arm -->
  <node name="cm_spawner" pkg="controller_manager"
        type="spawner" args="joint_state_controller arm_controller hand_controller
                             lifter_controller" />
</group>
```

controller__managerは，ロボットのハードウェア制御において，リアルタイム性が求められる要素を複数同時に制御することができる仕組み(パッケージ，ノード)

argsの中身を見ると文字から想像できるが，SEED-Noidを構成するハードウェア要素が全て指定されている．

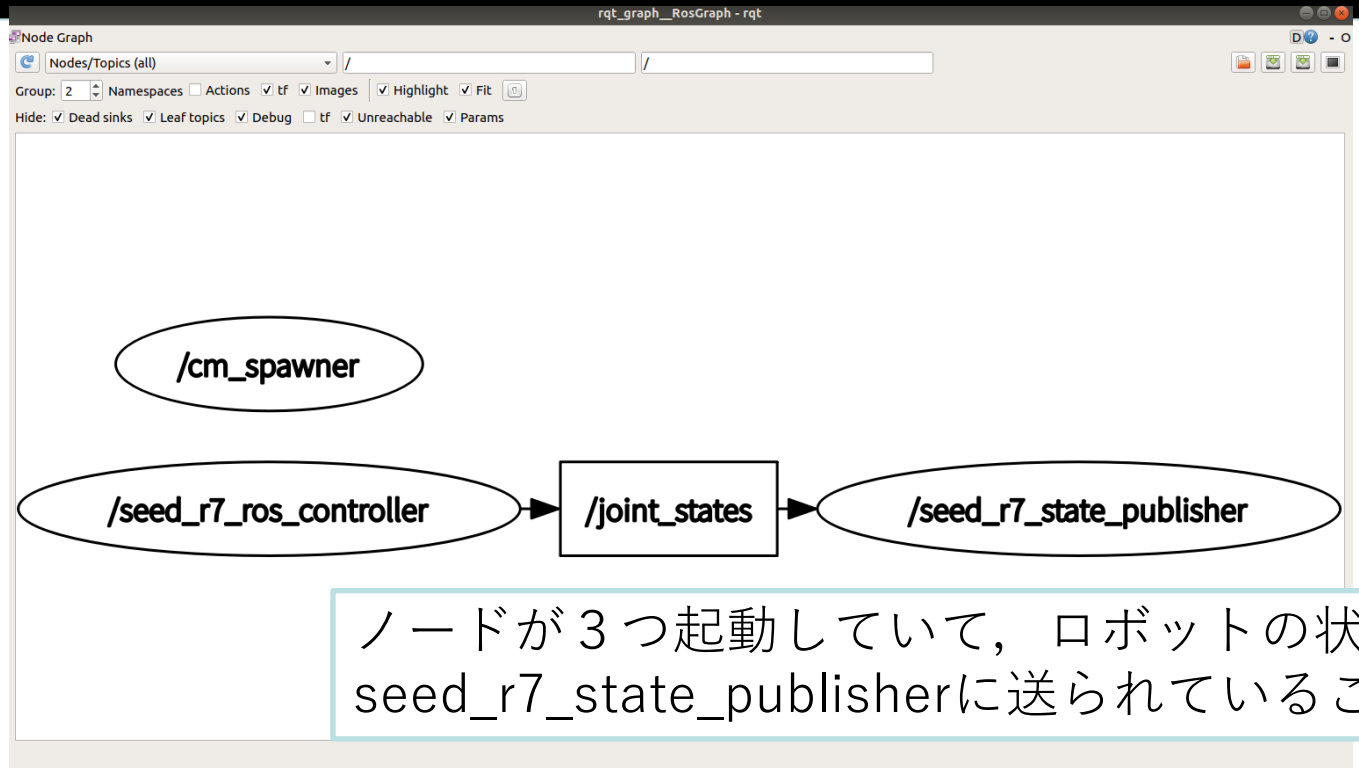
SEED-Noidのノードの起動

以下のコマンドでノードの起動を行う.

```
$roslaunch seed_r7_bringup seed_r7_bringup.launch
```

別のターミナルを起動して，以下のコマンドを実行
(以降，別のターミナルを起動する場合はターミナルXXとしています．)

```
$rqt_graph
```



ノードが３つ起動していて，ロボットの状態が，
seed_r7_state_publisherに送られていることがわかる.

本講座のゴールとなるデモの実行

実機を触ることはできないので，演習として，PC上でSEED-Noidの動作確認を行う．
ここで実行するサンプルと同等程度のことをできるようになることが今回の講座のゴール！

ターミナル１ *起動している場合は，無視

```
$roslaunch seed_r7_bringup seed_r7_bringup.launch
```

ターミナル2

```
$roslaunch seed_r7_navigation wheel_with_dummy.launch
```

ターミナル３

```
$roslaunch seed_r7_samples demo.launch
```

第２部のまとめ

SEED-Noidの制御のベースとなるlaunchファイルの理解

本講座のゴールとなるデモの実行・確認