

NEDO講座

第1部 ROSの基礎

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

本講座の概要

- SEED-Noidを対象として，自律移動機能やマニピュレーション機能を運用するための基礎について学ぶ。
なお，ROS 1 を対象にした説明・演習を行う。
 - ROSの基礎(オプション)
 - SEED-Noidの運用の基礎
 - SEED-Noidを用いた地図生成とナビゲーション
 - SEED-Noidを用いたマニピュレーション
- 上記項目を部構成とし，それぞれ別々の動画で説明をさせていただきます。

第1部の目的

- ROSの概要を把握し，SEED Noidのシステムを運用する上での基礎を固める.
- ROSをよく理解している方は飛ばしていただいて大丈夫です.
- 逆に，ROSなどのモジュールベースのプラットフォームに不慣れな方は，キーワードを知る意味でも一度ご確認ください.

第2部以降はここで紹介したキーワードが多用されますので，逆引きでもご利用ください.

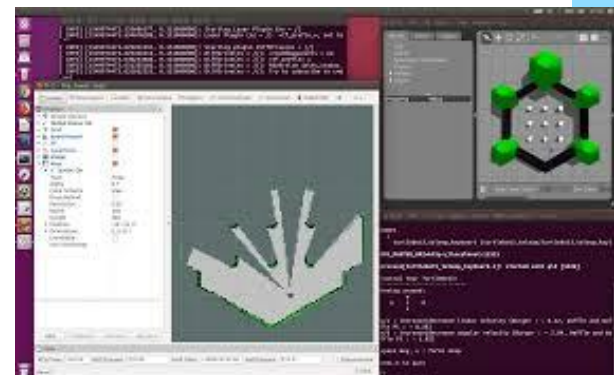
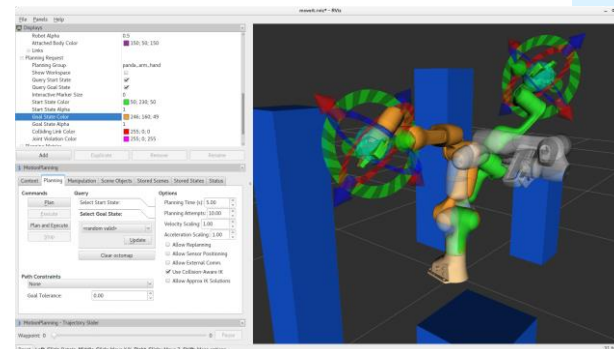
ROSとは？

ロボットのための

オープンソースのソフトウェアフレームワーク(ミドルウェア)

ROS

- 多くのハードウェアをサポート。
- ネットワーク透過性
- 多様な開発ツール
- 豊富なソフトウェアリソースとドキュメント



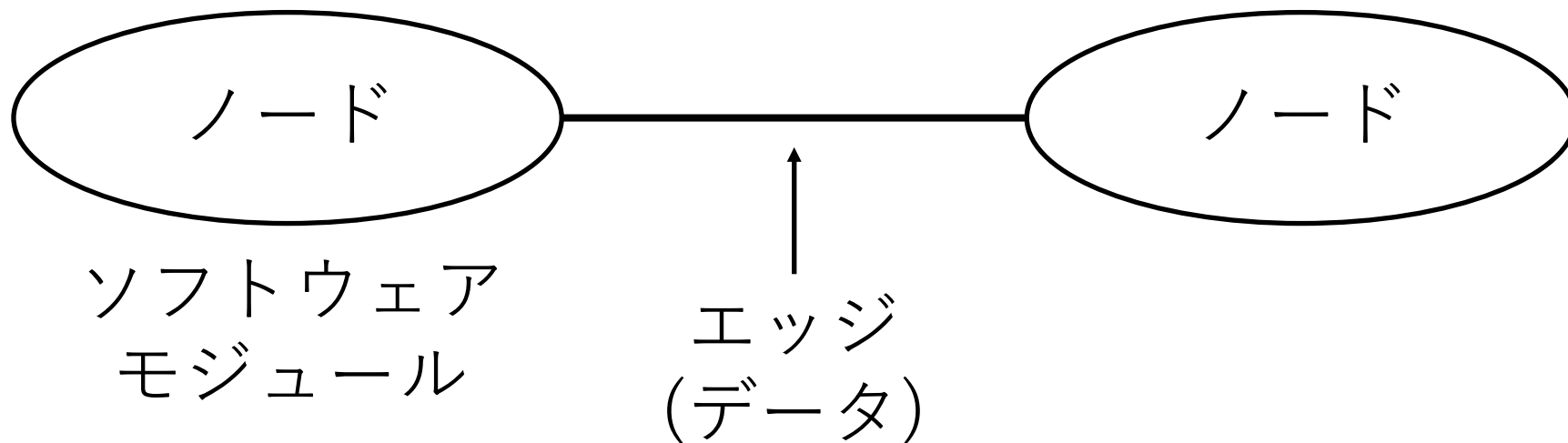
ROSの基礎(ROS Graph)

大前提

ROSでは、**多数のプログラムが独立して存在しており、ネットワークを介してお互いに通信しながら連携する。**

ROS Graph

グラフ理論で示されるような、**ノード**と**エッジ**を用いてシステムがグラフとして表現・構成される。



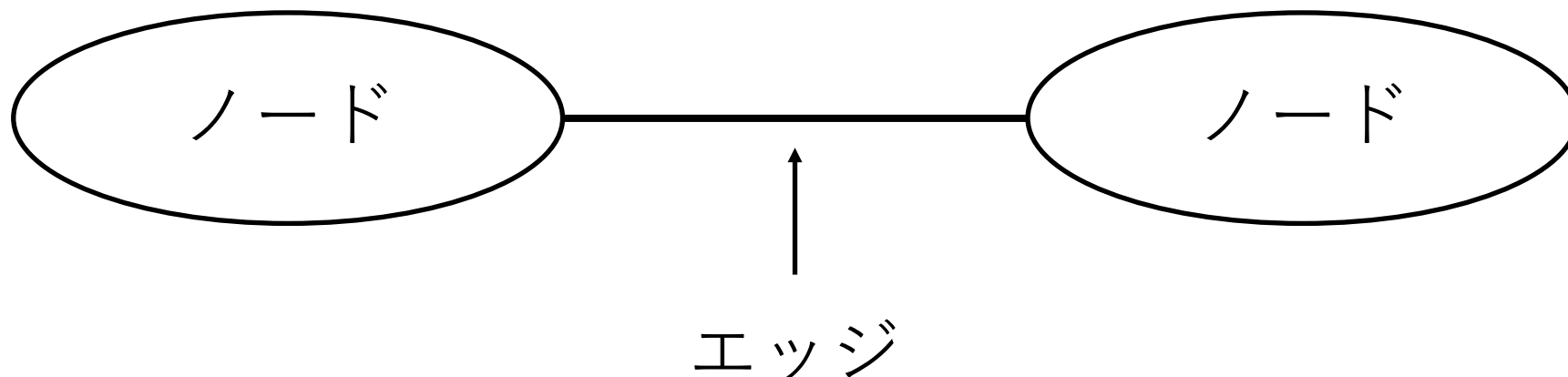
ROSの基礎(ROS Graph)

ノード

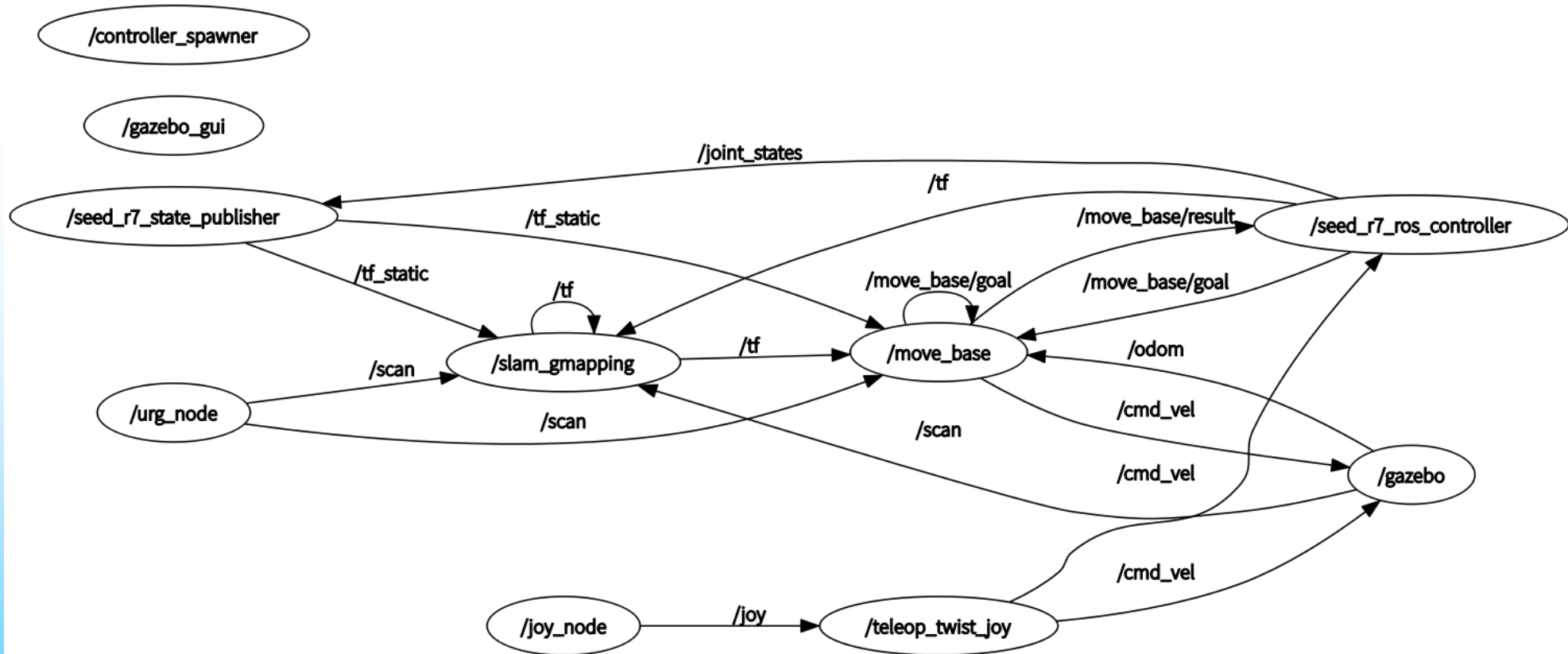
ハードウェアを制御したり，アルゴリズムなどのソフトウェアモジュール．他のノードへのデータの送信や，データの受信が可能．

エッジ

ソケット通信(TCP/IP)によるノード間の通信を担う．



SEED-Noidの地図生成時のROS Graphの例



ROSのツールである「rqt_graph」で確認可能

ソフトウェアモジュールがノードとして存在する意義

従来のロボットプログラム

従来のロボットソフトウェアは**密な連携**(1つのプログラムのみ)で動作しており、ソフトウェアの実行は容易であるが、**不具合発生時にシステム全体がダウン**。

ROSのシステム

(きちんと作られていれば)システムを構成する1つのノードが不具合でダウンしても、**システム全体のダウンにつながることは回避可能**。

(こういったシステムを**疎な連携**のシステムという)

ROSの基礎(roscore)

roscoreは各ノードが関連するノードと情報を送受信するための、接続情報を提供する。

- 全てのノードはroscoreに接続する必要がある。
- 新しく起動されたノードはroscoreに接続し、データを送受信するノードと直接接続するための情報を提供される。

roscoreはサーバ-クライアントモデルとエージェントモデルのハイブリッドの機能を持つ。

ノード間で直接メッセージを送信するための
ネームサービスとしての機能を提供

ROSの基礎(roscore)

複数のPCでシステムを構築していて、1つのroscoreに接続する場合、以下の様に設定すれば良い。

```
export ROS_MASTER_URI=http://hostname:11311
```

ここで、hostnameは各環境に合わせて設定。
IPアドレスを直接指定することも可能。

roscoreのパラメータサーバとしての機能

roscoreはノードの設定を行うためのパラメータ
(roscparam)サーバとしての機能を持つ。
(ノードのパラメータ情報についても管理している。)

catkin(キャッキン)はROS用のビルドシステム
CMakeとPythonスクリプトを合わせ、拡張したもの。

以前はcatkin_makeというコマンドが利用されていた
が、現在推奨はcatkinというコマンド。
(Pythonのパッケージとして提供されている。)

catkinコマンドでは、パッケージのビルド時の他の
パッケージとの相互参照を避けるために、各パッケー
ジを独立・並列で処理を行ってくれる。

```
$catkin [global options] <verb> [verb arguments and option]
```

catkinのサブコマンド(Verb) リスト

サブコマンド リスト	概要
build	(後述する)catkinワークスペース内のパッケージをまとめてビルドする。
config	catkinワークスペースの設定の表示や操作をする。
clean	catkinワークスペースの中身をクリーンする。
create	catkinワークスペース要素を作成
env	現在の環境変数の表示や、変更を行う
init	catkinワークスペースの初期化
list	catkinワークスペース内のパッケージの探索やリストの表示
locate	catkinワークスペース内の重要なディレクトリの場所を登録
profile	設定プロファイルの管理

<https://catkin-tools.readthedocs.io/en/latest/index.html>

ROSのパッケージが格納される作業用ディレクトリ
複数のワークスペースを生成できるが、1度に利用できる
ワークスペースは1つのみ。

ワークスペースを利用するためには、以下の様に、
ワークスペースとしての初期化を行う必要がある。

```
$catkin init
```

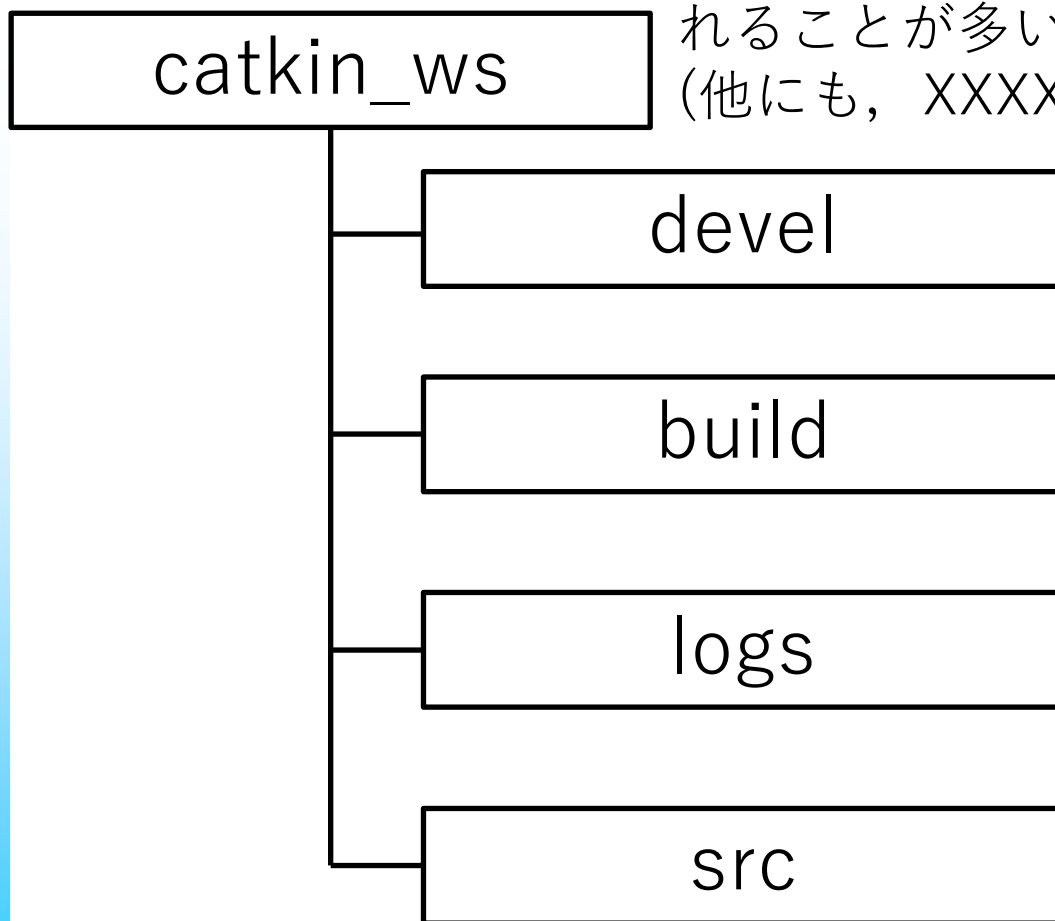
このコマンドを実行すると、隠しディレクトリである
“.catkin_tools”を生成する。どこのディレクトリでも
初期化は可能。

なお、catkin buildとしても初回の場合“.catkin_tools”
が作られる。

ROSの基礎(Workspaces)

「catkin init」を実行した後でのディレクトリ構成

ROS用ディレクトリ。
慣例的に、"catkin_ws"というディレクトリが使われることが多い。
(他にも、XXXX_wsとする場合もある。)



ワークスペースで用いられるシステムの設定を行う環境設定ファイルを格納

ビルドした実行ファイルやライブラリを格納。

Pythonの場合は気にしなくて良い。
(後述する)ROSパッケージ毎にログを保存。エラーなどの発生時にはここをみると言われる。

ROSパッケージが格納されているディレクトリ。

ROSのソフトウェアはROSパッケージとしてまとめられる。

パッケージは”src”以下に置かれることを想定。
(gitなどからダウンロードしてきたパッケージは必ず”src”の下に置くようにすること。)

新しいパッケージを作る場合は”src”ディレクトリで以下の様なコマンドを実行する。
(sampleというパッケージを作る場合。)

```
$catkin create pkg sample
```

ROSパッケージを実行するためのコマンド.

コマンドで指定されたプログラムに対するパッケージを探したり, 入力されたパラメータをプログラムに渡す事ができる.

```
$roslaunch <パッケージ名> <実行プログラム名> [パラメータ等]
```

“roslaunch”は1つのROSノードの起動には良いが, 大規模なシステムでは, 手間がかかることから, 後述する“roslaunch”が使われる.

詳しくは, 第2部の演習で確認.

ROSでは、ノード名や送受信するメッセージ(Topic), パラメータ等は全てがユニークな名前を持つ。

例

Cameraのノードでは、imageというトピックと、frame_rateというパラメータを持つ。

名前空間(Namespace)は名前が重なることを避けるために用いられる。

例

Cameraが2つあるとき、名前空間として、leftとrightを設け、right/image, left/imageというように名前空間をわけて定義できる。

ROSの基礎(Remapping)

ソースコードを変更することなく，既存のノードでメッセージを受け取れるようにRemappingを使う。

例

```
$/image_view image:=right/image
```

デフォルトはimageだが，それをright/imageというメッセージを受け取るようにRemapしている。

メッセージだけでなく，ノードのNamespaceのRemappingもできる。

例

```
$/camera __ns:=right/image
```

ROSの基礎(Remapping)

同様にノード名の変更も可能

例

```
$/camera __name:=camera1
```

Remappingを利用することで、ROSノードのソースコードに手を加えることなく、実行時に与えるパラメータで運用を工夫できる。

ROSの基礎(roslaunch)

複数のノードを起動するために用いられるコマンド。
XMLファイルで作成する。
文法などのは基本的にXMLのルールに従い、ROS用の
タグが設定されている。

例

```
$roslaunch PACKAGE LAUNCH_FILE
```

詳細は、演習を通じて学びます。

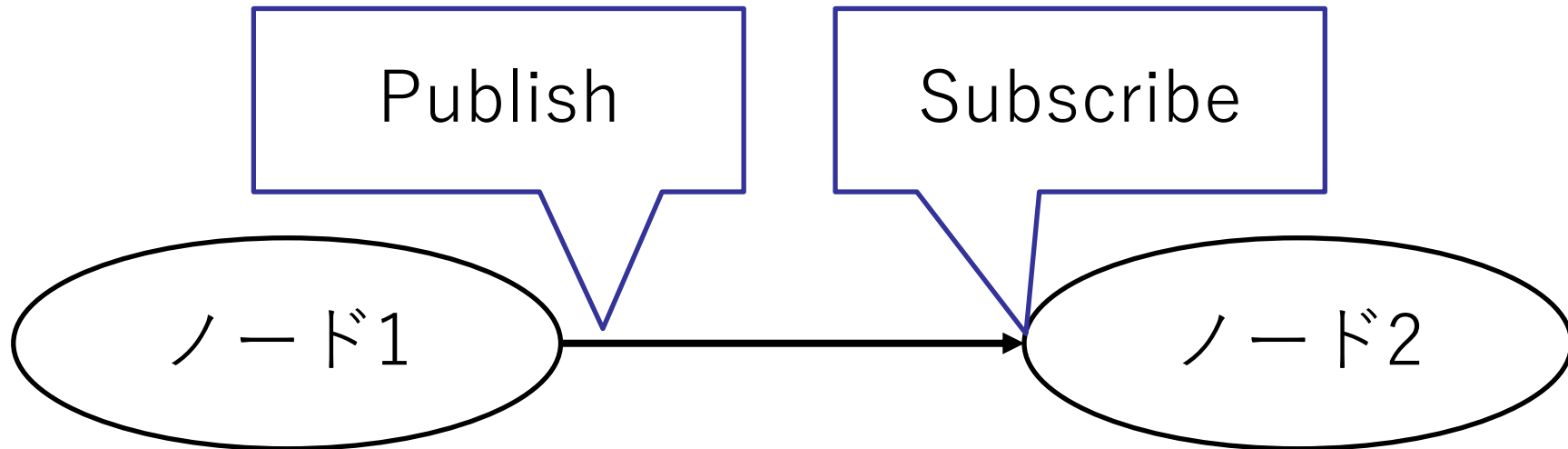
ROSが広く使われている理由にもなっている重要な機能の1つ。Transformの略。

システムで用いられる座標系を管理し，これらの座標系の間での座標変換を行ってくれる。

tfのtopicに合わせて，tf/tfMessageを利用する。
この中には，変換，座標系の名前の特定，相対座標や回転，時間などの管理を行う。

ROSにおける基本的な送受信するメッセージ。
Publisher/Subscriberモデルで実装されている。

直訳すると、出版社/購読者のモデル。Publisherはデータの配信を行い、読みたいSubscriberがデータを受信するというモデル。疎な連携を実現。



Topicはpublisherとsubscriberの間でメッセージのデータ型が一緒である必要がある。

サービスはあるノードから他のノードの持つ関数を実行することができる機能。同期式。
サーバとクライアントに分けて実装する必要がある。

サーバ

サービスをリクエストされた場合に実行されるコールバックを定義し，サービスを発行する。

クライアント

サービスにアクセスし，サービスを実行する。

サービスは，サービス定義ファイル(.srv)に入力と返り値を定義する．

このときのファイル名はサービス名になるので，気をつける．

サービスを定義し，パッケージのビルドを行うと，以下の3つのファイルが生成される．

(ここでは，TestService.srvと言う名前にする．)

- TestService
- TestServiceRequest(関数呼び出し)
- TestServiceResponse(呼び出しに対する応答)

サービスは同期呼び出しなのに対して，Actionは非同期呼び出し。
時間がかかり，ゴールに依存した行動を実行するのに利用される。

同期呼び出し

関数コールが行われたら，処理が終わるまで呼び出し元での処理をブロックする。

非同期呼び出し

関数コールが行われても，処理が終わるまで待たずに呼び出し元での処理が継続される。

アクションでは、以下の3つの機能が利用できる。

- goal(goalの設定。)
- Result (アクションの結果を受け取れる)
- Feedback(現在の状態を知ることができる。)

アクションはtopicを用いて実装する。

ROSの実装の流れ

1. ノードを設計する.

- ① 何の処理をするのか.
- ② 何のパッケージが必要かを考える.
- ③ 入出力のデータがなにか, 内部で実行前に設定したいパラメータは? 外部からパラメータ等の設定変更をする必要があるか? 外部からの関数呼び出しの必要性は?
- ④ 処理を実装する.

2. ノードを単体実行する.

- rostopic echoなどで動作の確認

3. ノードをシステムで実行する.

- roslaunchを作成し, 実行する.

ROSノードの再利用の流れ

1. 利用したいROSノードの取得

- Githubなどに公開されているROSノードを取得。
Workspace以下のsrcの中に取得したファイル一式を配置。

2. ROSノードをビルドする

- Catkinコマンドを利用して、ダウンロードしてきたノードをビルド

```
$catkin build [ダウンロードしたパッケージ名]
```

3. ROSノード利用する

- 公開されているROSノードの多くはlaunchファイルを持つため、launchファイルを利用してノードを起動する。

- ROSの基礎についての概説

演習

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

この演習の目的

ROSノードの開発手順や、実行方法について、実際に手を動かしながら、体感していただきます。

この演習で開発するROSノードはROSでの”Hello World”のような位置づけです。

内容自体はシンプルですが、基本が詰まっていますので、ソースコードの構成やビルド方法、運用を簡便にするための仕組みについての理解を深めて下さい。

演習 簡単なノードを作ろう

パッケージの生成

```
$cd ~/ros/melodic/src  
$catkin create pkg beginner_tutorials
```

ここでは、プログラムのコードを書くエディタとしてgeditを 사용합니다. (Windowsのメモ帳ライクで使いやすと思われるため.)
すでに使いやすいいディタがある場合はそちらを利用.

```
$cd ~/ros/melodic/src/beginner_tutorials  
$mkdir src  
$cd src  
$gedit talker.cpp
```

ソースコードなどは下記も参照

<http://wiki.ros.org/ja/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29>

ソースコード (talker.cpp)

```
1 #include <ros/ros.h>
2 #include <std_msgs/String.h>
3 #include <sstream>
4
5 int main(int argc, char **argv)
6 {
7     ros::init(argc, argv, "talker");
8     ros::NodeHandle n;
9     ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
10    ros::Rate loop_rate(10);
11
12    int count = 0;
13    while(ros::ok())
14    {
15        std_msgs::String msg;
16        std::stringstream ss;
17        msg.data = ss.str();
18        ROS_INFO("%s", msg.data.c_str());
19        chatter_pub.publish(msg);
20        ros::spinOnce();
21
22        loop_rate.sleep();
23        ++count;
24    }
25    return 0;
26 }
```

ソースコード (listener.cpp)

続いて, talkerからのトピックを受け取るノードを作成

```
$gedit listener.cpp
```

以下のコードを入力する.

```
1 #include <ros/ros.h>
2 #include <std_msgs/String.h>
3
4 void chatterCallback(const std_msgs::String::ConstPtr& msg)
5 {
6     ROS_INFO("I heard: [%s]", msg->data.c_str());
7 }
8
9 int main(int argc, char **argv)
10 {
11     ros::init(argc, argv, "listener");
12     ros::NodeHandle n;
13
14     ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
15     ros::spin();
16
17     return 0;
18 }
```

CMakeLists.txtの編集

ビルドするための設定をCMakeLists.txtに記述する。

```
$cd ~/ros/melodic/src/beginner_tutorials  
$gedit CMakeLists.txt
```

いろいろとコメントが入っていますが，消していただき，以下の
ようにCMakeLists.txtを書き換えてください。

```
1 cmake_minimum_required(VERSION 3.0.2)  
2 project(beginner_tutorials)  
3  
4 find_package(catkin REQUIRED COMPONENTS roscpp std_msgs)  
5  
6 include_directories(include ${catkin_INCLUDE_DIRS})  
7 |  
8 add_executable(talker src/talker.cpp)  
9 target_link_libraries(talker ${catkin_LIBRARIES})  
10  
11 add_executable(listener src/listener.cpp)  
12 target_link_libraries(listener ${catkin_LIBRARIES})  
13
```

作成したノードのビルドと実行

```
$cd ~/ros/melodic
```

```
$catkin build # エラーが出たら適宜対応
```

ターミナルを3個起動する

ターミナル1

```
$roscore
```

ターミナル2

```
$cd ~/ros/melodic/  
$source devel/setup.bash  
$roslaunch beginner_tutorials talker
```

ターミナル3

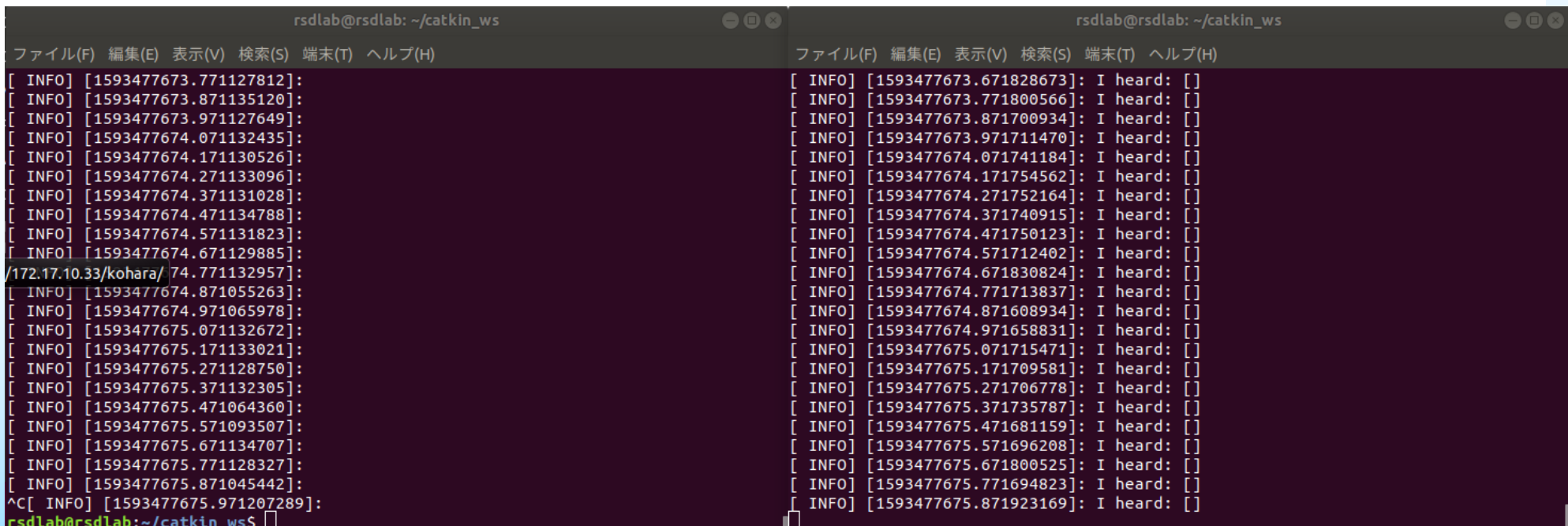
```
$cd ~/ros/melodic  
$source devel/setup.bash  
$roslaunch beginner_tutorials listener
```

実行結果について

空の文字列しか送っていないため文字は表示されない。

talker

listener



```
rsdlab@rsdlab: ~/catkin_ws
ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[ INFO] [1593477673.771127812]:
[ INFO] [1593477673.871135120]:
[ INFO] [1593477673.971127649]:
[ INFO] [1593477674.071132435]:
[ INFO] [1593477674.171130526]:
[ INFO] [1593477674.271133096]:
[ INFO] [1593477674.371131028]:
[ INFO] [1593477674.471134788]:
[ INFO] [1593477674.571131823]:
[ INFO] [1593477674.671129885]:
/172.17.10.33/kohara/74.771132957]:
[ INFO] [1593477674.871055263]:
[ INFO] [1593477674.971065978]:
[ INFO] [1593477675.071132672]:
[ INFO] [1593477675.171133021]:
[ INFO] [1593477675.271128750]:
[ INFO] [1593477675.371132305]:
[ INFO] [1593477675.471064360]:
[ INFO] [1593477675.571093507]:
[ INFO] [1593477675.671134707]:
[ INFO] [1593477675.771128327]:
[ INFO] [1593477675.871045442]:
^C [ INFO] [1593477675.971207289]:
rsdlab@rsdlab:~/catkin_ws$

rsdlab@rsdlab: ~/catkin_ws
ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[ INFO] [1593477673.671828673]: I heard: []
[ INFO] [1593477673.771800566]: I heard: []
[ INFO] [1593477673.871700934]: I heard: []
[ INFO] [1593477673.971711470]: I heard: []
[ INFO] [1593477674.071741184]: I heard: []
[ INFO] [1593477674.171754562]: I heard: []
[ INFO] [1593477674.271752164]: I heard: []
[ INFO] [1593477674.371740915]: I heard: []
[ INFO] [1593477674.471750123]: I heard: []
[ INFO] [1593477674.571712402]: I heard: []
[ INFO] [1593477674.671830824]: I heard: []
[ INFO] [1593477674.771713837]: I heard: []
[ INFO] [1593477674.871608934]: I heard: []
[ INFO] [1593477674.971658831]: I heard: []
[ INFO] [1593477675.071715471]: I heard: []
[ INFO] [1593477675.171709581]: I heard: []
[ INFO] [1593477675.271706778]: I heard: []
[ INFO] [1593477675.371735787]: I heard: []
[ INFO] [1593477675.471681159]: I heard: []
[ INFO] [1593477675.571696208]: I heard: []
[ INFO] [1593477675.671800525]: I heard: []
[ INFO] [1593477675.771694823]: I heard: []
[ INFO] [1593477675.871923169]: I heard: []
```

talkerだけをctrl+cで落とすとわかるが、データがとまるので、正しくデータの送受信が行えていることがわかる。

Lanuchファイルの作成

sample.launchというファイルを生成する.

```
$cd ~/ros/melodic/src/beginner_tutorials/src  
$mkdir launch  
$cd launch  
$gedit sample.launch
```

ファイルの中身

```
<launch>  
  <node name =“talker” pkg=“beginner_tutorials” type=“talker” output=“screen” />  
  <node name =“listener” pkg=“beginner_tutorials” type=“listener” output=“screen” />  
</launch>
```

ここはTabを入れているので注意.

Launchファイルでのノードの実行

ターミナル1

```
$cd ~/ros/melodic  
$source devel/setup.bash  
$roslaunch beginner_tutorials sample.launch
```

launchファイルを生成することで、複数のターミナルを起動しなくてもノードの起動ができるため、システム起動の簡便化につながる事がわかる。

起動しているノードの確認

ノードを起動した状態で，以下のコマンドを入力

```
$rqt_graph
```

```
rqt_graph
```

現在起動しているroscoreに登録されているノードとその接続関係，やりとりされているtopic名を視覚的に確認できるツール。
ブラックボックスとして利用する場合，ノード同士の関係性を把握するためにも利用できる。

起動しているノードの確認

rqt_graphを起動すると下記のウィンドウが出る。

