筑波大学 知能ロボット研究室
University of Tsukuba, Intelligent Robot Laboratory

「ROS」の使い方

ROS.org

資料作成：FJY, AZM, DME, YMG, HND, KAT, TCB, NSK
発表：西村 浩毅(NSK)

2022年度第3回山彦セミナー

今回の目標

- ROSの概要、システムを理解する
- ROSで山彦を動かす
- ROSのプログラミングをする

以上は山彦セミナー課題に
取り組むうえで必須の事項である

注意点

- 今回のセミナーの最低要件
  - ROSがインストール済み
  - 山彦をypspur-rosで動かせる

  ↑前回のセミナーで完了のはず
  まだの人は早急に行うこと
- 注意点
  - 本カリキュラムは今年が初
    不手際があればすみません
  - 知ってる人は先をやってOK
  - ypspur_rosの導入が今年からなので、不手際があるかもしれない
  - 公式ROS Tutorialに必ずしも沿わない

注意点

- 質問事項
  - Unixコマンドラインわかる？
  - C++/Pythonのクラスわかる？

Contents

ROSの概要

- ROSとは
  - ロボット用ソフトウェアの開発環境と通信フレームワークを提供
    ※ROS2ってのもある
  - オープンソース（無料）
  - 複数の対応言語・対応環境
    - メインはC++とPython
    - Ubuntu(推奨)、Windows、MacOS
- メリット
  - パッケージ導入が楽
    - 共同開発しやすい
    - 公開パッケージの導入が楽
    - パッケージの取捨選択が楽
- デメリット
  - 公式チュートリアルが初学者向けではない
    今回の山セミ後見ることを推奨

ROSの概要

- ROSのパッケージの例
  - 視覚化ツール「rviz」
    - ROS標準搭載のビューア
    - センサデータやオドメトリを同時に複数表示可能
  - 地図作成ツール「gmapping」
    - SLAMによる2次元グリッドマップが作成可能
  - 自己位置推定ツール「amcl」
    - 自己位置推定を行ってくれる
  - ナビゲーションツール「move_base」
    - 経路計画を行ってくれる

ROSの仕組み

- 基本概念としては4つ
  - Roscore:
    - ROSの基本的なプログラムやNodeの集まり
  - Node:
    - ROSにおけるプログラムの基本単位
  - Topic:
    - Node間の通信を行うための機構
  - Service:
    - Node間の関数呼び出し

ROSの仕組み(roscore)

- NodeやTopic、Serviceの管理を行う
  - ROS Master：Nodeの名前登録、解決を行う
  - ROS Parameter Server：パラメータを共有
  - rosout：ログ記録用のNode
- roscoreの起動後,各Nodeを実行する

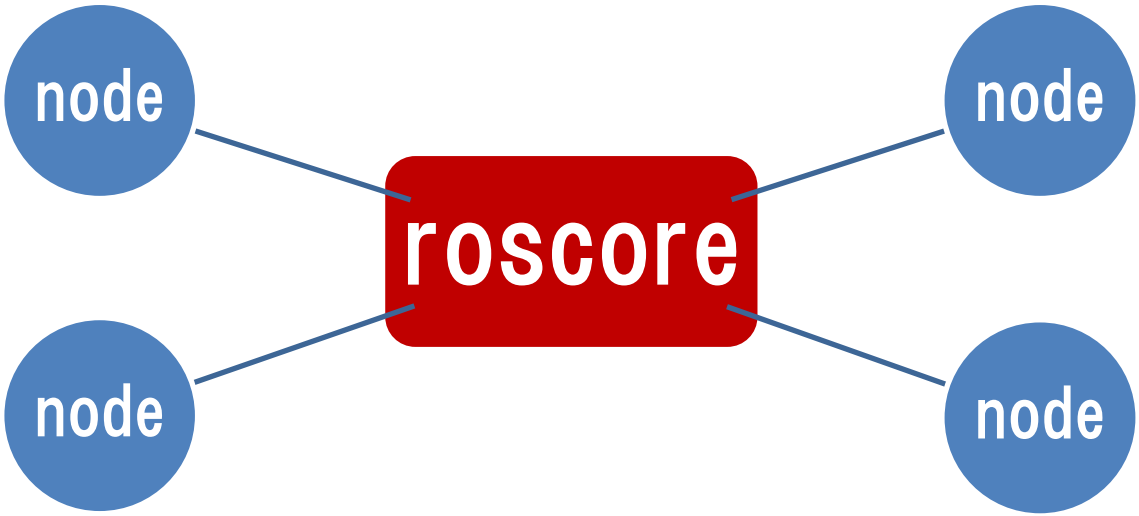
```
$ roscore
$ rosrun “ﾊﾟｯｹｰｼﾞ名” “ﾉｰﾄﾞ名” “引数”
```

  - これがrosの基本的な使い方
- 複数のNodeをまとめて実行することも可能
  - roslaunch(後ほど説明)

```
$ roslaunch “ﾊﾟｯｹｰｼﾞ名” “ﾗﾝﾁ名”.launch
```

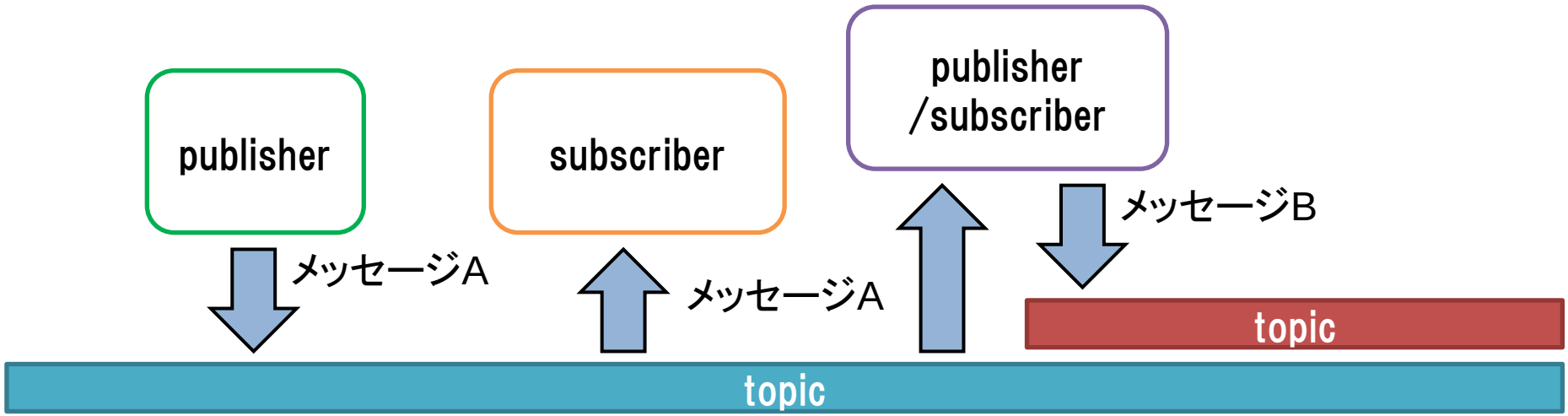
Node

- Node：ROSにおけるプログラムの単位
- 各センサやロボットを一つのノードとして扱う
- Node間の通信はroscoreを通じて行う



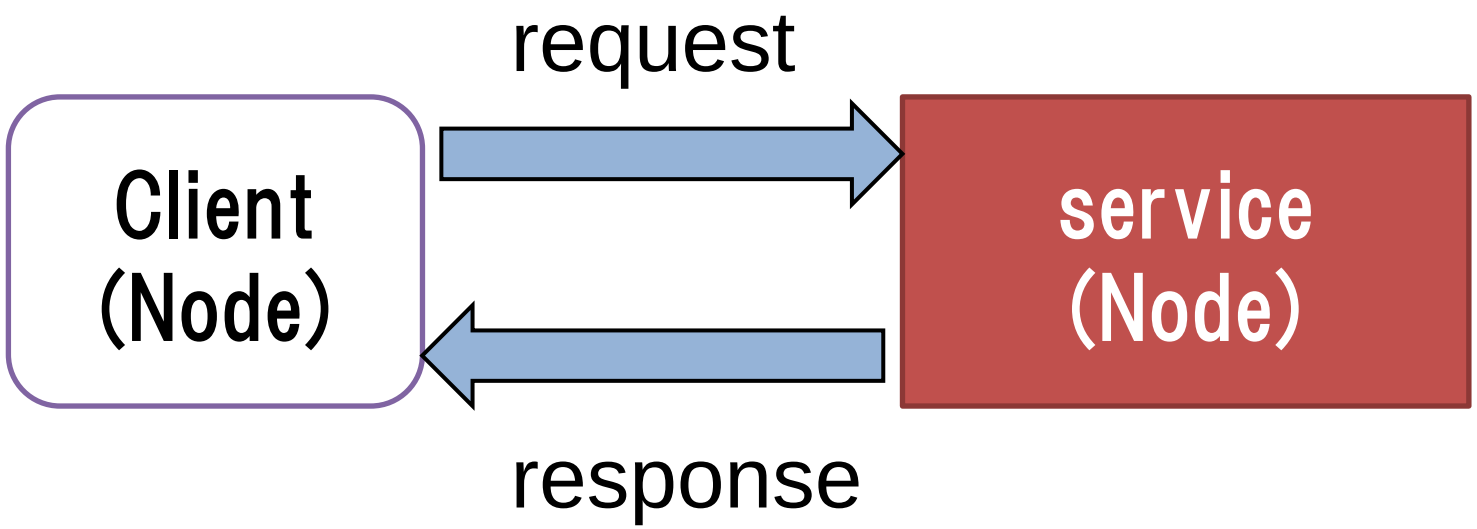
Topic

- Node間の通信を行うための機構
- Nodeには<Publisher>と<Subscriber>が存在
  - <Publisher>はデータをTopic経由でROSに流す
  - <Subscriber>は必要なデータをTopic経由で受信
  - Publishされたデータはすべての<Subscriber>に一斉送信される



Service

- Node間のrequest-response型の通信機能
- Node間の関数呼び出しのようなもの
- データの送受信のみでなくClient側から処理を行わせることが可能



Contents

山彦でのROS使用方法(ypspur-ros)

- ypspur-rosを使用する
- ROSの”cmd_vel”トピックをsubscribeしてYP-spurからモータコントローラへ目標速度を送る
- YP-Spurのオドメトリ情報をROSにpublishする
- **とにかく、”cmd_vel”というトピックに司令値だけ送ればロボットを動かしてくれるインターフェースになってる！すごい！**

山彦でのROS使用方法(ypspur-ros)

- ypspur-rosの使い方
  1. 1つ目のターミナルで `$ roscore`
  2. 2つ目のターミナルで

```
$ rosrun ypspur_ros ypspur_ros _param_file:=/home/
        <user>/researches/programs/platform/yp-robot-
params/robot-params/<ロボットの種類>.param
```

  3. “cmd_vel” トピックを publish する ROS のプログラムを実行

前回のコマンドの復習

- 前回山彦を動かしたコマンドは・・・
  1. ターミナル① `$ roscore`
  2. ターミナル②

```
$ rosrun ypspur_ros ypspur_ros _param_file:=/home/
        <user>/researches/programs/platform/yp-robot-
params/robot-params/<ロボットの種類>.param
```

  3. ターミナル③

```
$ rostopic pub /ypspur_ros/cmd_vel geometry_msgs/Twist [0.1, 0,
0] [0, 0, 0.3]
```

  - ↑/cmd_velにPublishをして動かしていた
  - とにかく、/cmd_velというトピックに
    指令をPublishすることで動かせる！！！！！！！

Tips (terminator)

- ROSではターミナルをたくさん起動する
  そのためターミナルを分割できるterminatorを
  newPCで入れた
- terminatorのショートカットキー
  - ctrl + alt + t: 起動
  - ctrl + shift + e: 縦分割
  - ctrl + shift + o: 横分割
  - ctrl + shift + w: 小窓削除
  - ctrl + tab: 小窓移動

ROSを使う前の事前確認

- ターミナルを起動し、以下のコマンドを入力

```
$ cat ~/.bashrc | grep ros
```

“source /opt/ros/noetic/setup.bash”と表示されるか確認。されない場合、

```
$ echo source /opt/ros/noetic/setup.bash >> ~/.bashrc
$ roscore
```

エラーが出ないか確認. 大丈夫ならCtrl+c

コピペ非推奨
Tab補完推奨

- joy入ってますか？（課題で使います）
  - ゲームパッドの入力をROSで読み取り、トピックに流すパッケージ

```
$ sudo apt update
$ sudo apt install ros-noetic-joy
```

Contents

プログラムでROSから山彦を制御する

- ノードのプログラムを作ってロボットを制御する
  - ワークスペースの作成→パッケージの作成→プログラムの作成
  - launchによる複数ノードの起動

- 今回はPythonのプログラムを作る
  - C++のやり方は去年の資料とか見るといいと思います

ROSのファイル構成

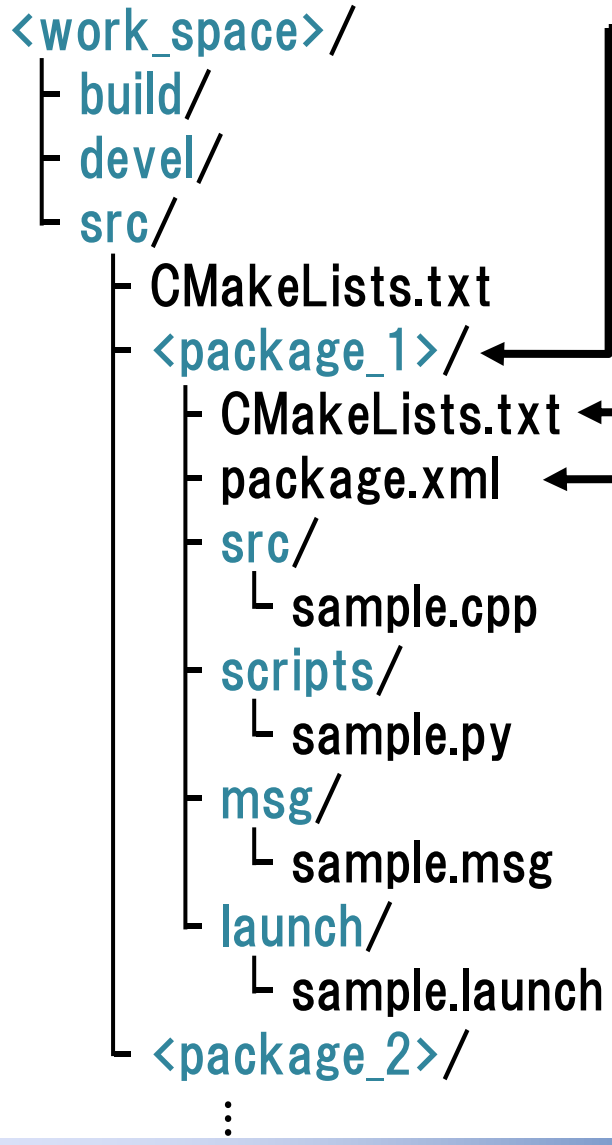
ワークスペースとパッケージ

<work_space>/
- build/
- devel/
- src/
  - CMakeLists.txt
  - <package_1>/
  - <package_2>/
  - ⋮

<work_space>/
- このディレクトリ下のパッケージの機能を実行可能
- 用途に合わせてワークスペースを複数使い分けてもよい
- ただし、ワークスペースがPC内に複数ある場合はコマンドで明示的に指示する必要がある。これがだるい
- ROS本体にあるパッケージはワークスペース下に入れなくても使用可能

<work_space>/src/<package>
- このワークスペース内のパッケージ
  次ページで解説

ROSのファイル構成



<package>/

- 目的に合わせてノードのソースファイル・メッセージ・設定ファイル等をまとめたもの
- <work_space>/src/に入れてビルドするだけで使用可能になる
- 基本的に開発はパッケージ単位で行う

CMakeLists.txt

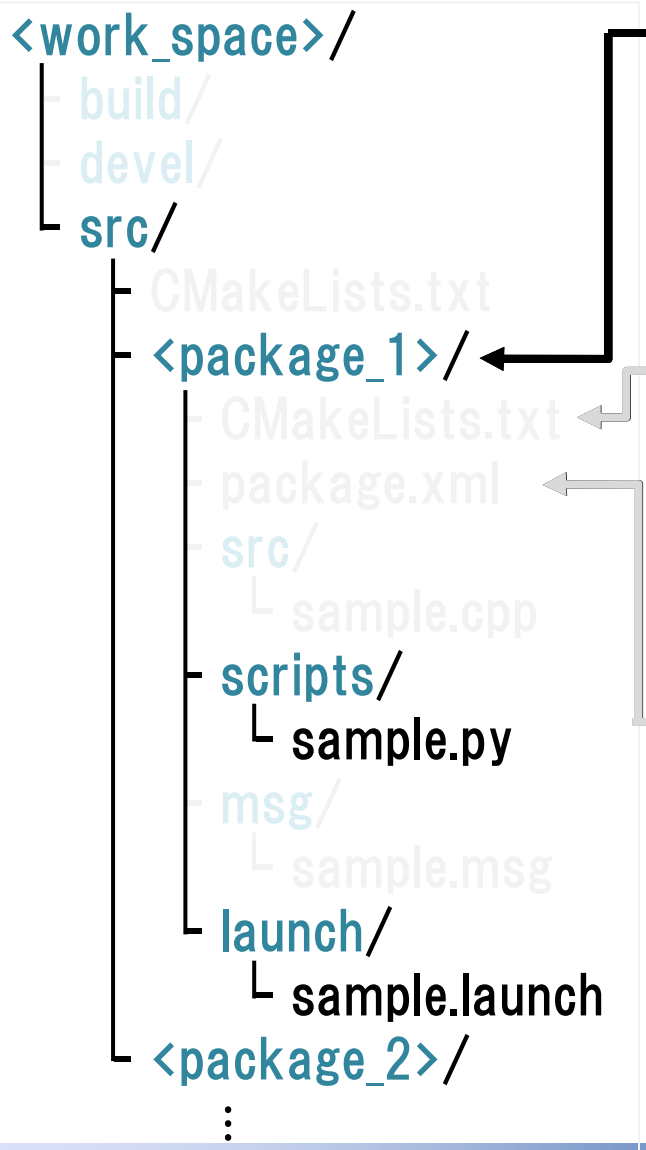
C++で書かれたノードの追加，
外部パッケージのメッセージの使用
等の際に編集する
<work_space>/src/下にも同名のファイルがあるが、こちらは基本的に編集しない

package.xml

メッセージファイルを追加する等の際に編集する

src/ ：C++のソースコードのディレクトリ
scripts/ ：Pythonのソースコードのディレクトリ
msg/ ：msgの設定ファイルのディレクトリ
launch/ ：launchの設定ファイルのディレクトリ

ROSのファイル構成(今回使う部分)



<package>/

- 目的に合わせてノードのソースファイル・メッセージ・設定ファイル等をまとめたもの
- <work_space>/src/に入れて
  ビルドするだけで使用可能になる
- 基本的に開発はパッケージ単位で行う

CMakeLists.txt
C++で書かれたノードの追加,
外部パッケージのメッセージの使用
等の際に編集する
<work_space>/src/下にも同名のファイル
があるが、こちらは基本的に編集しない

package.xml
メッセージファイルを追加する等の際に編集する

src/ : C++のソースコードのディレクトリ
scripts/ : Pythonのソースコードのディレクトリ
msg/ : msgの設定ファイルのディレクトリ
launch/ : launchの設定ファイルのディレクトリ

課題1:ROSワークスペースを作る

- 以下コマンドでワークスペースを作る

```
$ mkdir –p ~/<work_space>/src          *ワークスペースとなるディレクトリの作成
$ cd ~/<work_space>/                   *ワークスペースとなるディレクトリの作成
$ catkin_make                          *ワークスペースのビルド
$ source devel/setup.bash              *ワークスペースのセットアップ
$ echo source ~/<work_space>/devel/setup.bash >> ~/.bashrc
 *bashrcに4つ目のコマンドを追加する。ターミナル起動時に自動で4. のコマンドを実行するように
```

- <work_space>は各人自由に変更してください
- catkin_makeはカレントディレクトリが<work_space>でないと
  エラーが出ます

課題2:ROSパッケージを作る

```
$ cd ~/<work_space>/src
  *<ワークスペース>/srcに移動
$ catkin_create_pkg <package_name> std_msgs sensor_msgs
rospy roscpp
  *パッケージの作成
$ cd ~/<work_space>/
   *ワークスペースに移動
$ catkin_make
   *ワークスペースのビルド
```

- 注:<package_name>の1文字目は必ず小文字にすること！(バグる)

課題3:ROSノードを作る（Python）

```
$ cd ~/<work_space>/src/<package_name>
  *パッケージのディレクトリに移動
$ mkdir scripts
  *scriptsディレクトリ(Pythonソースコードのディレクトリ)の作成
$ touch <file_name>.py
  *pythonファイルを作成する。<file_name>には好きな名前を入れる。これがノードになる
$ chmod +x <file_name>.py
  *ノードのファイルを実行可能にする。これをしないと後でエラーが出る。
```

どんな方法でもいいので、作成したノードのファイルに山セミのページからダウンロードしたsample.pyの内容をコピペする。

```
$ cd ~/<work_space>/
$ catkin_make
   *ワークスペースのディレクトリに移動してビルドする。
```

課題4:ROSノードを起動

```
(ターミナル1)$ roscore
(ターミナル2)$ rosrun ypspur_ros ypspur_ros
_param_file:=~/researches/programs/platform/yp-robot-
params/robot-params/<ロボットの種類>.param
  *roscoreとypspurの起動
(ターミナル3)$ rosrun joy joy_node
  *joy_nodeの起動(ゲームパッドの入力を受け取る)
(ターミナル4)$ rostopic echo joy
  *トピック「/joy」に流れる内容(ゲームパッドへの入力)を可視化
(ターミナル5)$ rosrun <package_name> <file_name>.py
  *自分のパッケージのノードを起動
(ターミナル6)$ rostopic echo /joy_state
  *トピック「/joy_state」に流れる内容を可視化
```

- 左スティックへの入力内容だけが抜き出されて別のトピックで表示されている

課題5:トピックの一覧を見る

- 課題4のノードを起動したまま、別のターミナルで以下のコマンドを実行する

```
$ rosnode list
  *現在起動しているノードの一覧が表示される
/controllerNode
/joy_node
/rosout
/ypspur_ros
$ rostopic list
  *現在流れているトピックの一覧が表示される
/diagnostics
/joy
/joy_state
…(省略)
```

  - デバッグに使える

プログラムの解説

- ControllerNodeはjoy(ゲームパッドの入力のトピック)をSubscribe

  ↓

- それが更新されるたびに(入力があるたびに)/joy_stateトピックへ入力内容をPublishしている

```
rospy.init_node("controllerNode")
```

```
self.sub = rospy.Subscriber('joy', Joy, self.joy_callback)
```

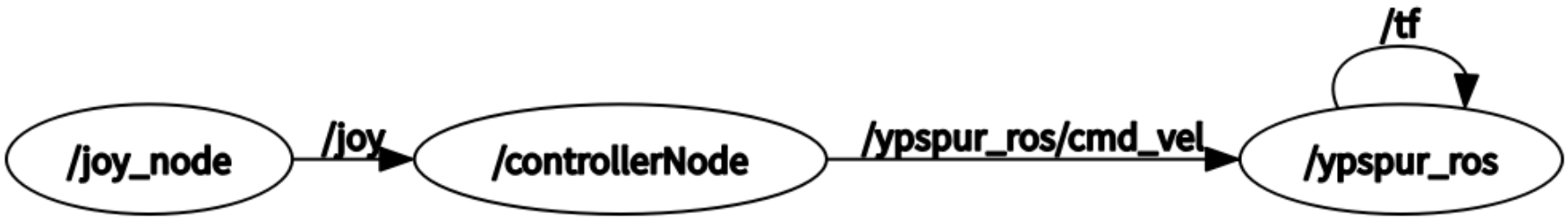
```
self.pub = rospy.Publisher('/joy_state', Float32,
queue_size=10)
```

```
def joy_callback(self, joy_msg):
    state = Float32()
    state.data = joy_msg.axes[1]
    self.pub.publish(state)
```

課題6:トピックの送受信を図で見る

- 課題4のノードを起動したまま、別のターミナルで以下のコマンドを実行する

```
$ rqt_graph
  *グラフが表示される
```



  - デバッグに使える

課題7:ノードをまとめて起動する(launch)

```
$ cd ~/<work_space>/src/<package_name>
  *パッケージのディレクトリに移動
$ mkdir launch
  *scriptsディレクトリ(Pythonソースコードのディレクトリ)の作成
$ touch <launch_file_name>.launch
  *pythonファイルを作成する。<launch_file_name>には好きな名前を入れる。
$ chmod +x <launch_file_name>.launch
  *launchファイルを実行可能にする。これをしないと後でエラーが出る。
```

どんな方法でもいいので、作成したlaunchファイルに山セミのページからダウンロードしたsample.launchの内容をコピペする。

課題7:ノードをまとめて起動する(launch)

- コピペしてきたlaunchファイルを書き換える

```
<launch>
    <node name="joy_node"   pkg="joy" type="joy_node"/>
    <node name="yourNode" pkg="<package_name>" type="<file_name>.py"/>
</launch>
```

  - <package_name>とか<file_name>を書き換えてください

課題7:ノードをまとめて起動する(launch)

```
$ roslaunch <package_name> <launch_file_name>.launch
  *作成したlaunchファイルが起動する
```

- このように、launchファイルを作ることでまとめてノードを起動でき、大変便利(roscoreも自動で起動してくれる)
- 色々オプションがあるのでlaunchファイルの書き方はネットで調べてください
- 参考:
- https://kazuyamashi.github.io/ros_lecture/ros_launch.html

M1の人はスプレッドシートへの記入を

- スプレッドシートにuアドレスを記入してください