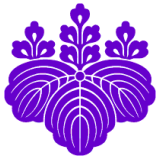


筑波大学 知能ロボット研究室
University of Tsukuba, Intelligent Robot Laboratory

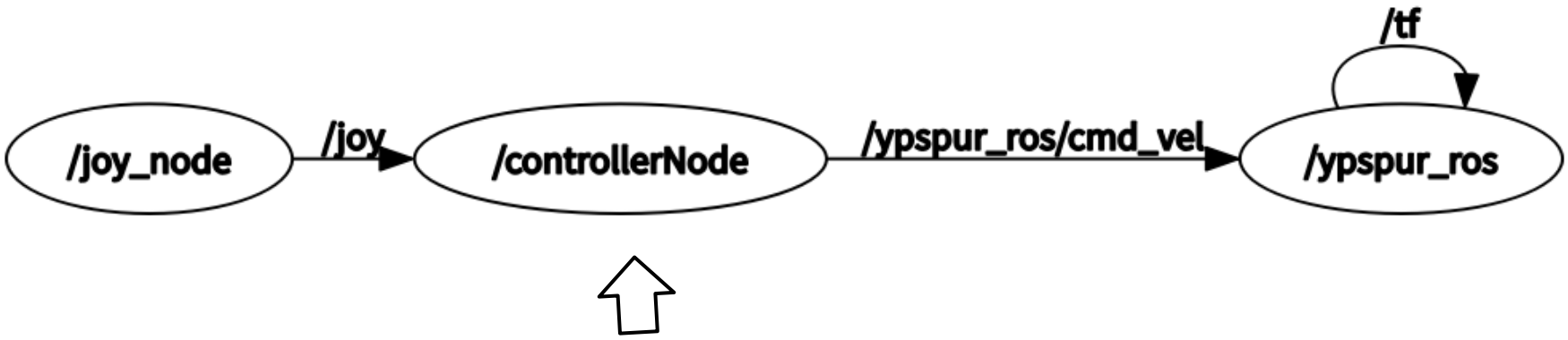
速度、角速度の指令と値の確認

資料作成：TCB,OKW
発表：及川 裕介（OKW）

2023年度第4回山彦セミナー　2022-04-12

今回の目標

- Launchファイルの確認
- Pythonファイルを書き換えて山彦を動かす
- 同じ処理のc++ノードを作り山彦を動かす



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で動かせる
  - 前回のセミナーでpyhtonノードを作成
- 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
  - 公式ROS Tutorialに必ずしも沿わない

Contents

Launchファイルの確認

```
$ 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

Contents

Pythonファイルの編集

- 前回作成した<file_name>.pyを編集

```
import rospy
from sensor_msgs.msg import Joy
from std_msgs.msg import Float32
from geometry_msgs.msg import Twist #追加
from rospy.exceptions import ROSInterruptException
.........
#.publish(hoge)メソッドでhogeをトピックにPublish
self.pub = rospy.Publisher('joy_state', Float32, queue_size=10)
self.cmd_vel_pub = rospy.Publisher('ypspur_ros/cmd_vel', Twist, queue_size=1) #追加
.........
#Subscribeしたコントローラの入力を標準出力
for i in range(4):
        print(str(i+1), joy_msg.axes[i])
print("¥n")
#以下を追加
twist = Twist()
twist.linear.x = joy_msg.axes[1]*0.2
twist.angular.z = joy_msg.axes[2]*0.2
self.cmd_vel_pub.publish(twist)
```

Contents

ノードの起動&トピックの確認

1. ターミナル① `$ roscore`
2. ターミナル② `$ rosrun joy joy_node`
3. ターミナル③

```
$ rosrun <package_name> <file_name>.py
```

4. ターミナル④

```
$ rostopic echo /ypspur_ros/cmd_vel
```

- ↑ジョイスティックを倒して
  何かしら値が表示されることを確認する！！！！！！！
- /cmd_velの内容は
  linear.x：前進速度指令
  angular.z：角速度指令
  他：0

Contents

山彦を動かす

1. ターミナル⑤

```
$ 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
```

左スティックを動かすと山彦が動く！！！！！！！

トピックの確認(再)

- ypspur_rosノードが/ypspur_ros/cmd_vel を受け取っていることを確認する

```
$ rostopic echo /ypspur_ros/cmd_vel
```

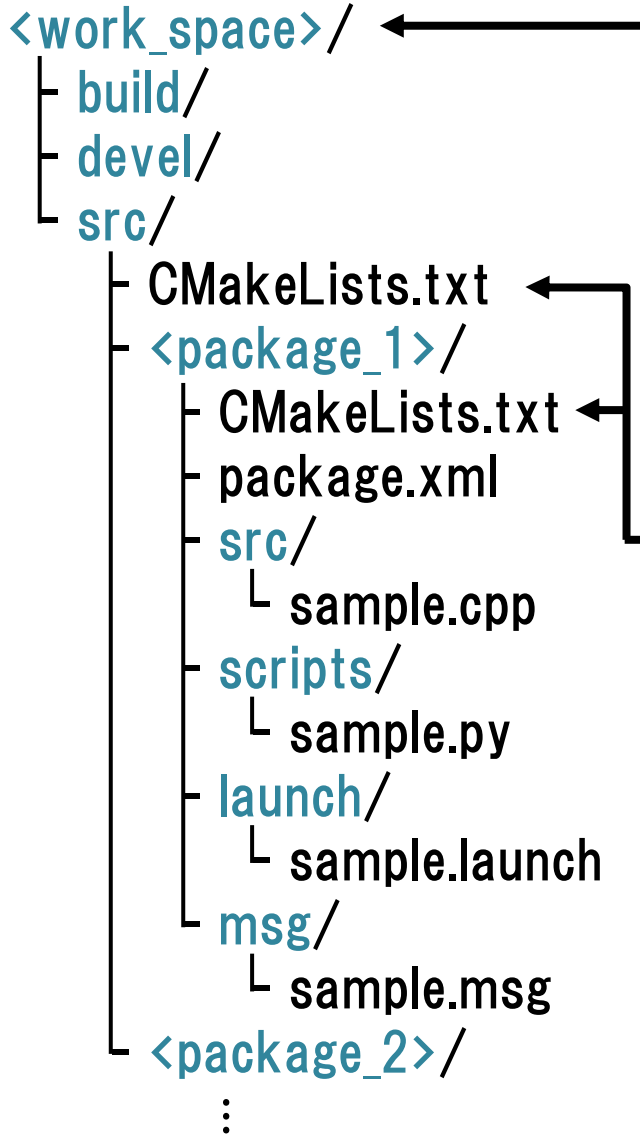
または

```
$ rqt_graph
```

- 確認が終わったら、roscore,rosrun,rostopicなど起動しているものを終了してください

Contents

ROSのファイル構成



<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るとよい
- ROS本体にあるパッケージは入れなくても使用可能

CMakeLists.txtが複数あることに注意

<work_space>/src/CMakeLists.txt（上）は通常は編集しない

ROSのファイル構成

```
<work_space>/
 ├ build/
 ├ devel/
 └ src/
    ├ CMakeLists.txt
    ├ <package_1>/
    │  ├ CMakeLists.txt
    │  ├ package.xml
    │  ├ src/
    │  │  └ sample.cpp
    │  ├ scripts/
    │  │  └ sample.py
    │  ├ msg/
    │  │  └ sample.msg
    │  └ launch/
    │     └ sample.launch
    └ <package_2>/
       ⋮
```

<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
- <worl_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

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

```
<work_space>/
  ├ build/
  ├ devel/
  └ src/
      ├ CMakeLists.txt
      ├ <package_1>/
      │   ├ CMakeLists.txt
      │   ├ package.xml
      │   ├ src/
      │   │   └ sample.cpp
      │   ├ scripts/
      │   │   └ sample.py
      │   ├ msg/
      │   │   └ sample.msg
      │   └ launch/
      │       └ sample.launch
      └ <package_2>/
          ⋮
```

今回の編集箇所

package.xmlは必要に応じて編集(今回は編集しない)

C++ノードの作成

- Internal からymbc_seminar.zipをダウンロード、展開
- ymbc_seminar/src/joy_to_cmd_node.cpp を前回作成したパッケージのsrcディレクトリにコピー
- Cmakelists.txtの編集
  137行目に以下を追加

```
add_executable(joy_to_cmd_node src/joy_to_cmd_node.cpp)
target_link_libraries(joy_to_cmd_node ${catkin_LIBRARIES})
```

add_executable()　　：実行ファイル作成のターゲットを指定
target_link_libraries() :ターゲットが私用するライブラリを指定

ROSノードを作る（C++）

ソースコードの解説

#include “ros/ros.h”

rosを使うなら必要

#include “geometry_msgs/Twist.h”

使用するメッセージは
”<package>/<message>.h”で指定

ros::init(argc, argv, “sample_node”);

ROSの初期化．おまじない
第3引数：このプログラムで起動するノードの”ノード名”

ros::NodeHandle _nh;

ノードのハンドラ．おまじない

ROSノードを作る（C++）

ソースコードの解説

```
_joy_sub = _nh.subscribe("/joy", 1, &Joy2Cmd::joyCallback, this);
```

subscribe機能のインスタンス
第1引数：subscribeするトピック名
第2引数：キューサイズ（メッセージが早すぎるときのもの）
第3引数：メッセージが送られてきたときに実行する関数名
第4引数：オブジェクト自信を指すポインタ

```
_cmd_vel_pub = _nh.advertise<geometry_msgs::Twist>("/ypspur_ros/cmd_vel", 1, this);
```

publish機能のインスタンス
<>内：送信するメッセージ型の"<package>::<message>"
第1引数：publishするトピック名
第2引数：キューサイズ（メッセージが早すぎるときのもの）
第3引数：オブジェクト自信を指すポインタ

catkin_make

```
$ cd ~/<workspace_name>/
$ catkin_make
$ source devel/setup.bash
```

↑catkin_makeする度に必ず実行する

```
$ rosrun <opackage_name> joy_to_cmd_node
```

Contents

課題

- C++ノードを使って山彦を動かしてみよう！

課題

- C++ノードを使った山彦を動かすlaunchファイルを作ってみよう！！

課題解答例

```
<launch>
<node name="ypspur_ros" pkg="ypspur_ros" type="ypspur_ros">
<param name="param_file"
value="/home/<user>/researches/programs/platform/yp-robot-
params/robot-params/<山彦の機種>.param"/>
</node>
<node name="joy_node" pkg="joy" type="joy_node"/>
<node name="yourNode" pkg="<package_name>"
type=”joy_to_cmd_node"/>
</launch>
```

- 終わり
  移行前年度のスライドから抜粋

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で山彦を動かす

課題2 解答

1. 山彦が0.1[m/s]で前進する
2. 結果は以下の通り
    1. /cmd_velの内容は
        linear.x：前進速度指令
        angular.z：角速度指令
        他：0
    2. /odomの内容は
        linear.x：前進速度
        angular.z：角速度
        他：0
    3. ypspur_ros_bridgeでは以上の他に
        座標x, y, 姿勢も表示される

Contents

ROSパッケージを作る

パッケージの作成

- $ cd ~/<work_space>/src
  カレントディレクトリの移動
- $ catkin_create_pkg <package_name> std_msgs rospy roscpp
  パッケージの作成
- $ cd ~/<work_space>/
  カレントディレクトリの移動
- $ catkin_make
  ワークスペースのビルド

！重要！
パッケージ名の
1文字目は必ず小文字に
（バグる）

ROSノードを作る（C++ソースファイル）

ソースファイルの作成

- $ cd ~/<work_space>/src/<package>
- $ gedit src/<source_file_name>
  ※<source_file_name>は”〇〇.cpp”となるように
- 開いた空のファイルに
  山セミページの今回の資料のzipの
  sample_pkg/src/sample_src.cpp
  の内容をコピペ

ROSノードを作る（C++）

ソースコードの解説

#include “ros/ros.h”

rosを使うなら必要

#include “std_msgs/Int64.h”

#include “geometry_msgs/Twist”

使用するメッセージは
”<package>/<message>.h”で指定

ros::init(argc, argv, “sample_node”);

ROSの初期化．おまじない
第3引数：このプログラムで起動するノードの”ノード名”

ros::NodeHandle nh;

ノードのハンドラ．おまじない

ROSノードを作る（C++）

ソースコードの解説

ros::Subscriber sub = nh.subscribe(“sample_topic”, 1000, callback);

subscribe機能のインスタンス
第1引数：subscribeするトピック名
第2引数：キューサイズ（メッセージが早すぎるときのもの）
第3引数：メッセージが送られてきたときに実行する関数名

ros::Publisher pub = nh.advertise<geometry_msgs::Twist>(“cmd_vel”, 1000);

publish機能のインスタンス
<>内：送信するメッセージ型の”<package>::<message>”
第1引数：publishするトピック名
第2引数：キューサイズ（メッセージが早すぎるときのもの）

ROSノードを作る（C++）

ソースコードの解説

geometry_msgs::Twist pub_msg;

publishするメッセージのインスタンス生成
<package>::<message> <instance>;

ros::Rate loop_rate(10);

後述

ros::ok()

ROSが正常に動作すれば true
コマンドからCtrl+cが送られる等のとき false
つまりwhile(ros::ok()){}は,
rosが正常に動作する限り繰り返すということ

ROSノードを作る（C++）

ソースコードの解説

ros::spinOnce()

nh.subscribe()で設定した関数にアクセス
この関数が呼ばれたとき更新されたトピックがあれば，
そのトピックをsubscribeするsubscriberの関数を実行する

```
ros::Rate loop_rate(10);
loop_rate.sleep();
```

上側の関数の引数：周波数[Hz]
そこで設定した周波数になるように残り時間スリープする
ただ(1/周波数)秒スリープするのではなく，
そこまでの処理時間も勘案してスリープ時間を決定する．

ROSノードを作る（C++）

ソースコードの解説

pub_msg.linear.x = 2.0;

メッセージのインスタンスに代入

pub.publish(pub_msg);

引数のメッセージをpublishする

def callback(const std_msgs::Int64::ConstPtr& msg){}

subscriberの実体化の時に指定した，
メッセージが送られたときに実行する関数．
引数にトピックから読み込んだデータが代入されている．
引数はポインタのため，メンバの指定はmsg->data
であることに注意

ROSノードを作る（C++）

<package>/CMakeLists.txt の編集

- 13行目付近
  “std_msgs”の下に以下を追加
  geometry_msgs
  ※これは外部パッケージ(今回は”geometry_msgs”)
  　のメッセージを使うときに必要

- 119行目付近
  “# include”→“include”

- 138行目付近追加
  “# add_executable(${PROJECT_NAME}_ …(略)”の下に以下を追加
  add_executable(<実行ファイル名> src/<ソースファイル名>)

- 155行目付近
  # target_link _libraries(${PROJECT_NAME}_node
  #　 ${catkin_LIBRARIES}
  #)
  の下に以下を追加
  target_link_libraries(<実行ファイル名> ${catkin_LIBRARIES})

ROSノードを作る（C++）

<package>/package.xml の編集

- 55行目付近
  <build_depend>std_msgs</build_depend>
  の下に以下を追加
  <build_depend>geometry_msgs</build_depend>

- 62行目付近
  <exec_depend>std_msgs</exec_depend>
  の下に以下を追加
  <exec_depend>geometry_msgs</exec_depend>

※ これは外部パッケージ(今回は”geometry_msgs”)
のメッセージを使うときに必要

ROSノードを作る(各名前の注意点)

- ソースファイル名：
  〇〇.cppという名前
  プログラムのソースコードが書かれたファイル
  変更する場合CMakeLists.txtの再編集が必要
- 実行ファイル名：
  gccで言うコンパイル後に生成されるファイル
  windowsで言う〇〇.exe
  Pythonの場合ソースファイル名と同じ
  変更する場合CMakeLists.txtの再編集が必要で，
  起動コマンドも変わる
- ノード名：
  実行ファイルにより生成されたノードの名前
  ノードの起動前に様々な手法で**容易に変更可能**
  同じ実行ファイルで生成されたノードだとしても，
  ノード名は唯一無二でなければならない
  (詳しくはlaunchで)

ROSノードを作る(C++)

- ソースファイルの作成
- CMakeLists.txtの編集
- package.xmlの編集

が完了したら

- $ catkin_make

自作ノードを起動する(課題0)

課題0

以下を行って，自作ノードをビルドせよ

- ソースファイルの作成
- CMakeLists.txtの編集
- package.xmlの編集
- $ catkin_make

ROSの便利機能

ROSにはまだ触れていない機能が多くある

- roslaunch
- rosparam/param（yamlファイル）
- rosLogger
- rosbag
- サービス/クライアント
- rqt_console

今回はこの中でも使うことの多いlaunchを解説

roslaunch

roslaunchの機能

<package>/launch/〇〇.launchを作ることで...

- 1回のコマンドで複数のノードを起動可能
- roscoreが自動で起動
- ノード名やトピック名をビルドなしで変更可能
- ノードやトピックに名前空間(苗字)を追加可能
- ノードの引数やパラメータを事前に記入可能

roslaunch

launchファイルの作成

$ mkdir <package>/launch

$ gedit <package>/launch/<launch_name>
※<launch_name>は”〇〇.launch”となるように

- 開いた空のファイルに
  山セミページの今回の資料のzipの
  sample_pkg/launch/sample_launch.cpp
  の内容をコピペ

roslaunch

ソースコードの解説

- ファイルの最初と最後は<launch>と</launch>
- <node pkg="hoge" type="fuga" name="piyo"/>
  ノードを起動するコード
  - hoge: パッケージ名
  - fuga: 実行ファイル名
  - piyo: ノード名
    （ソースファイルで記述したものと同じでなくてよい）
  - これらをダブルクォーテーションで囲むことを忘れずに
- <remap from="hoge" to="fuga">
  トピック名等を変更するコード
  - hoge: 変更元のトピック名
  - fuga: 変更後のトピック名

roslaunch

- 起動方法

```
$ roslaunch <package> <launch_name>
```

launchファイルは作成/編集してもビルド不要

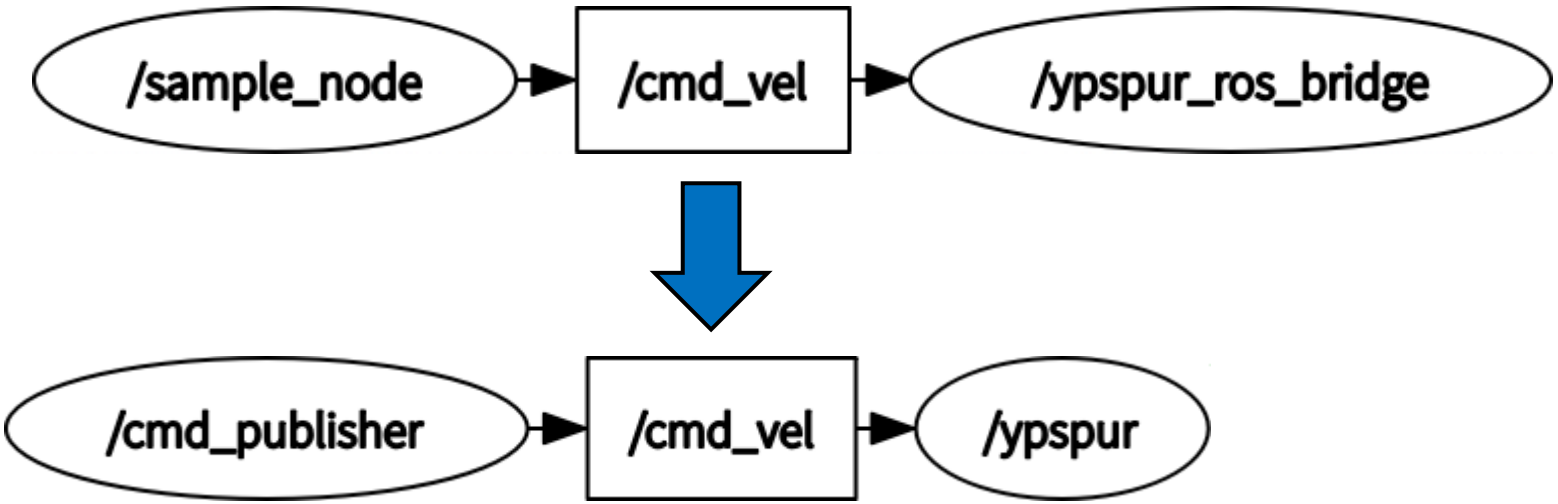
！重要！
ypspur_ros_bridgeより先に
ypspur-coodinatorを起動しないと
山彦は動かない
つまりlaunchより先に
ypspur-coodinatorを起動しないと
山彦は動かない

roslaunch

- 課題

  roslaunchでノードの起動,
  ノード名とトピック名の変更を行い,
  $ rqt_graph で確認せよ

- 解答



Contents

便利な外部パッケージ

- catkin build: catkin_makeの上位互換
- tf2: 座標管理/変換ツール
- urg_node: URGのデータをpublish
- rviz: トピックの3D可視化ツール
- gazebo: 3Dシミュレータ
- Navigation stack (move_base, amcl, gmapping): 自立移動ロボットの色々
- map_server: 地図データをpublish
- ROSはpython2で動くため，Pytho3で実行可能にするソフトを入れると便利(方法多数)

Contents

エラーマニュアル

- catkin_makeで”The specified source space ○○ does not exist”
  カレントディレクトリを<work_space>にする
- タブ補完がされない
  $ source ~/<work_space>/devel/setup.bash
- catkin_makeが通らない
  $ source ~/<work_space>/devel/setup.bash

発展課題

時間内にここまで全ての課題が終わった場合，
以下に挑戦せよ．これらは全て任意課題とする．

- オドメトリから自己位置を推定し，
  指定位置についたら停止するノードを作成せよ
- 自作のメッセージを使ったトピック通信をせよ
  （CMakeLists.txtとpackage.xmlを編集する必要があることに注意）
- param/rosparameを用いて，launchファイル内の数字を変えることで
  ビルドせずに挙動が変化するノードを作成せよ
- yamlファイルを用いてparam/rosparamを変更せよ
- catkin buildをインストールし便利さを体感せよ
- 公式ROS tutorialを見て今回触れていない部分を理解せよ
- tf2をbroadcat, listenするノードをそれぞれ作成せよ
- 今回のセミナーのわかりにくい点を発表者に伝えよ
  （他の旧人を介して伝えるのでも可）

参考

- ROSの公式チュートリアル
  http://wiki.ros.org/ja/ROS/Tutorials
- ROSの色々書いてあるQiita
  https://qiita.com/srs/items/5f44440afea0eb616b4a
- ロボ研のROS勉強会
  https://www.roboken.iit.tsukuba.ac.jp/tsukubachallenge-wiki/ros/study
  （旧
  https://www.roboken.iit.tsukuba.ac.jp/internal/event-wiki/study/ros/2015 ）