実習で学ぶ初めてのROS

九州大学大学院
システム情報科学研究院 教授
倉爪　亮

KYUSHU UNIVERSITY

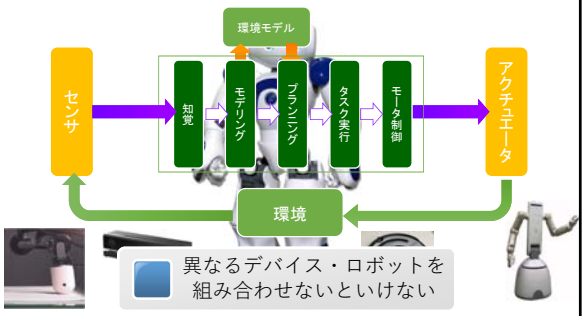
2016/11/22

内容

- http://irvs.github.io/rosbook_jp/
- 1～7章まで
  - 1 ROS入門
  - 2 ROSのインストール
  - 3 ROSの基本知識
  - 4 ROS コマンド
  - 5 ROSツール
  - 6 ROSプログラミングの基本
  - 7 パッケージの導入方法

2016/11/22

ロボットプログラミング



異なるデバイス・ロボットを
組み合わせないといけない

2016/11/22

ロボットプログラミング

- ロボットの構成機器ごとにプログラムを作成

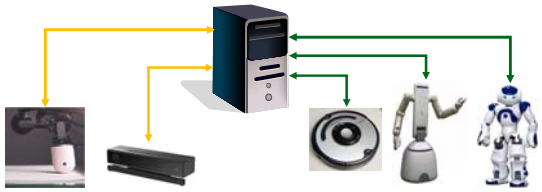
- 問題点：特定のデバイスに依存したハードコーディング
  - ✓拡張性
  - ✓再利用性
  - ✓対故障性

その都度作成→非効率

2016/11/22

RTミドルウェア

- ロボットの構成機器をソフトウェアレベルでモジュール化
- モジュール間のインターフェースを統一

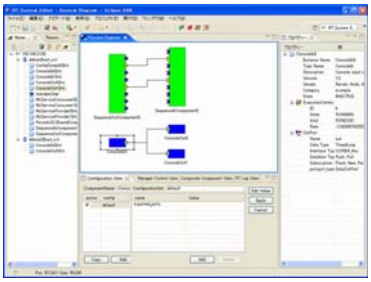


開発が効率化

2016/11/22

RTミドルウェア

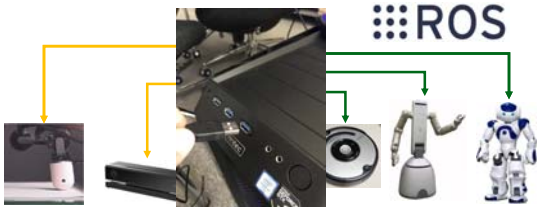
- OpenRTM：産総研が開発
  http://www.openrtm.org/openrtm/ja/content/openrtm-aist-official-website



2016/11/22

Robot Operating System (ROS)

- ROS　http://www.ros.org/
- ロボットの構成機器をソフトウェアレベルでモジュール化
- モジュール間のインターフェースを統一



2016/11/22

RT Middleware(OpenRTM)とROS

- 目指すものは一緒

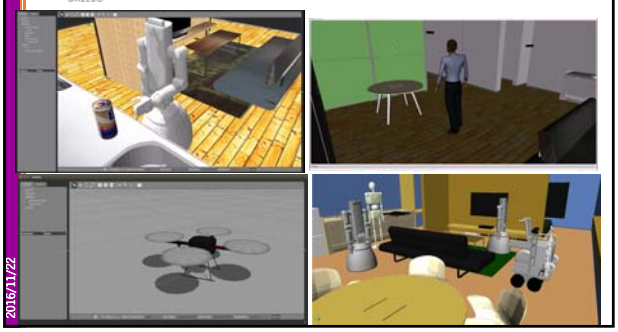
- 機能的にはほぼ差はない！
  - 細かい差はある
  - ROSはトピック通信やサービス通信が定義されている
  - RT Middlewareはポート間の接続で通信を規定
  - ROSはOpen Source Robotics Foundation (OSRF) が管理
  - RT MiddlewareはObject Management Group (OMG) で規格化
  - ROSはノード，RT MiddlewareはRTC (RT Component)
  - OpenRTMはエディタがある
  - ROSは3D Viewerや物理シミュレータが提供されている
  -
  -

2016/11/22

強力なツール群

Gazebo　動力学シミュレータ

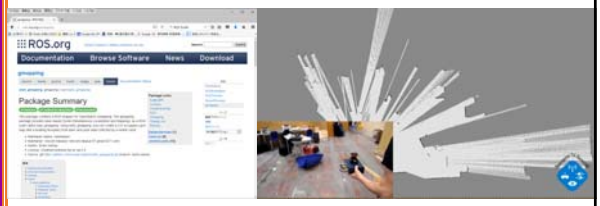
Rviz　3次元可視化ソフト



2016/11/22

地図生成＆位置同定（SLAM）

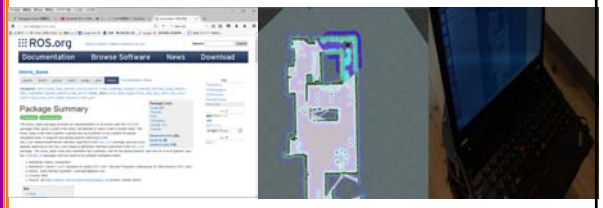
gmapping



2016/11/22

自律移動と位置同定（Navigation Stack）

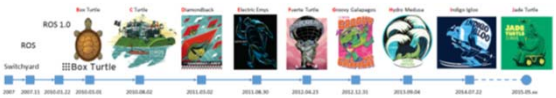
move_base & AMCL



2016/11/22

Robot Operating System (ROS)

- 歴史
  - M. Quigley (Stanford Univ., AI Lab, 2007.5~)
  - WillowGarage (2007.11~)
  - Open Source Robotics Foundation (2011~)

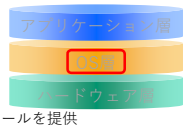


- 最新バージョンは Kinetic Kame (2016/5)
  →本講習では 安定している Indigo Iglooを使用
  対応オペレーティングシステム：
  Ubuntu 14.04 Trusty Tahr（LTS）

2016/11/22

Robot Operating System (ROS)

- ソフトウェアプラットホーム
  - アプリケーション開発のための（疑似的な）オペレーティングシステム
    - ハードウェア抽象化
    - 低レベルデバイス制御
    - ロボットが一般的に利用する機能実装
    - メッセージ通信
    - パッケージ管理　など
  - アプリケーション開発を支援するコマンドやツールを提供
    - コマンド：catkin_make, rostopic, roscore, rosrun, etc..
    - ツール：gazebo(シミュレーション), Rviz (ビューア), etc..



Robot Operating System (ROS)

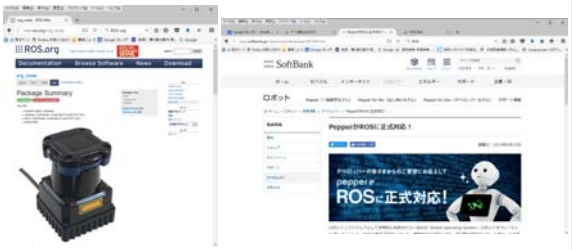
- 様々なOS上で動く「メタOS」
  - OS : Ubuntu, OS X, Windows, Debian, Fedoraなど
- プログラムを「ノード」としてモジュール化
  - ノード＝実行ファイル
  - ノード間の通信手順を取り決め

ノードA：車輪のモータを制御
ノードB：経路を計画、など

- ロボット開発のための機能部品を、
  “パッケージ”＝ノード　＋　メッセージ（データ形式）＋
  サービス＆トピックス（ROSインターフェース）
  “スタック”　＝パッケージの集合
  としてWebで配布
- ライブラリ群を容易にシェアできる仕組みを提供
- オープンソース・BSDライセンス
  - https://ja.osdn.net/projects/opensource/wiki/licenses%2Fnew_BSD_license

Robot Operating System (ROS)

- 世界中の研究者・技術者が開発したソフトを利用可能
- 多くのセンサ，アクチュエータ，ロボットをサポート



Robot Operating System (ROS)

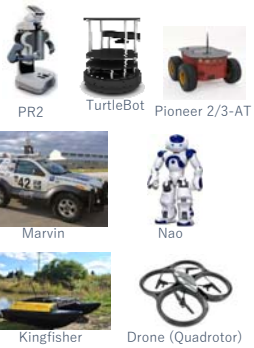
- センサパッケージ
  - 1D range finders
  - 2D range finders
  - 3D Sensors (range finders & RGB-D cameras)
  - Audio / Speech Recognition
  - Cameras
  - Force/Torque/Touch Sensors
  - Motion Capture
  - Pose Estimation (GPS/IMU)
  - Power Supply
  - RFID
  - Sensor Interfaces（マインドストーム、ワンボードマイコンなど）

＊ http://wiki.ros.org/Sensors

Robot Operating System (ROS)

- ロボットパッケージ（約120体）
  - Mobile manipulator
  - Mobile robot
  - Manipulator
  - Autonomous car
  - Humanoid
  - UAV（無人飛行機）
  - AUV（自律型無人潜水機）
  - UWV（無人航行船）
  - Others（マインドストームなど）

＊ http://wiki.ros.org/Robots



Robot Operating System (ROS)

- 処理パッケージ
  - 物体認識
  - 経路計画
  - 動作計画
  - スケジューラ
  - SLAM
  - その他たくさん

＊ http://wiki.ros.org/Robots

- 便利なツール
  - ３次元可視化ソフト　rviz
  - シミュレータ　Gazebo
  - 管理ソフト rqt
  - ライブラリ
    - PCL, OpenCV, OpenRAVE

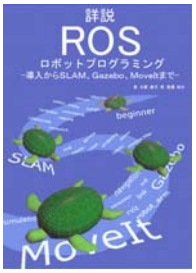
GAZEBO

ROSの日本語解説書

- Webで無料公開
- http://irvs.github.io/rosbook_jp/

詳説 ROSロボットプログラミング
-導入からSLAM・Gazebo・MoveItまで-

著者：表 允晢, 倉爪 亮, 渡邊 裕太
出版日：2015年 11月 30日（初版）
ISBNコード：9784990873608
フォーマット：PDF版
ページ数：342p



ROSのインストール

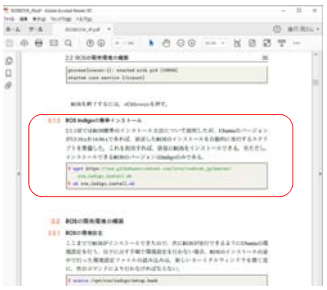
ROSのインストール

- http://robotics.ait.kyushu-u.ac.jp/books/ROSBOOK_JP.pdf
- 29ページから37ページを実行して，ROSをインストールしてみましょう
  - ROS Indigoのインストール
    - NTP(Network Time Protocol) 設定
    - ROSリポジトリアドレスの追加
    - キーの設定
    - パッケージインデックスの更新
    - ROS Indigo Iglooのインストール
    - rosdepの初期化
    - rosinstallのインストール
    - 環境設定ファイルのロード
    - 作業フォルダの作成と初期化
    - テスト
  - ROSの開発環境の構築

・35ページ「2.1.2 ROS Indigoの簡単インストール」が楽
・https://github.com/irvs/ros_tms/wiki/installにもあります

ROSのインストール 2.1.2 ROS Indigoの簡単インストール

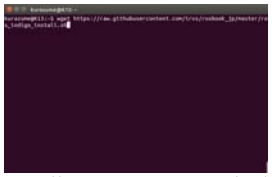
35ページ「2.1.2 ROS Indigoの簡単インストール」



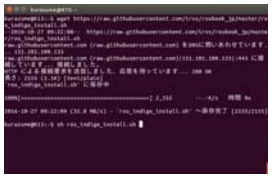
ROSのインストール 2.1.2 ROS Indigoの簡単インストール

https://github.com/irvs/ros_tms/wiki/install

ROSのインストール 2.1.2 ROS Indigoの簡単インストール



$ wget https://raw.githubusercontent.com/irvs/rosbook_jp/master/ros_indigo_install.sh



$ sh ros_indigo_install.sh

ROSのインストール 2.1.2 ROS Indigoの簡単インストール

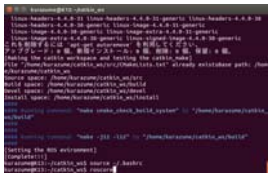
パスワード入力

[Completed!!!]と出たら終了

2016/11/22

ROSのテスト 2.2.2 ROSの動作テスト

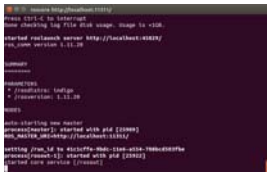
$ source ~/.bashrc



$ roscore

2016/11/22

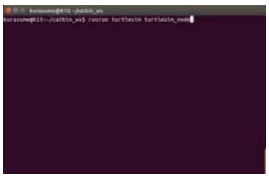
ROSのテスト 2.2.2 ROSの動作テスト



左の画面が出たら成功

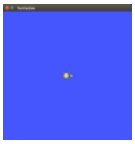
2016/11/22

ROSのテスト 2.2.2 ROSの動作テスト



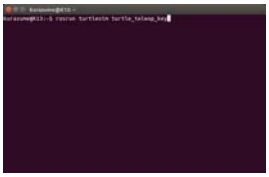
別のターミナルを開いて
「Ctrl+Alt+T」

$ rosrun turtlesim turtlesim_node



2016/11/22

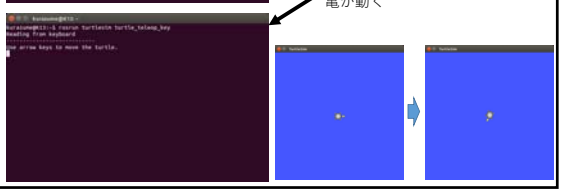
ROSのテスト 2.2.2 ROSの動作テスト



別のターミナルを開いて
「Ctrl+Alt+T」

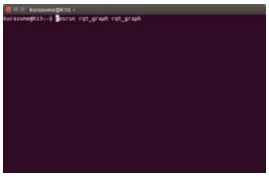
$ rosrun turtlesim turtle_teleop_key

左の画面で矢印キーを押すと
亀が動く



2016/11/22

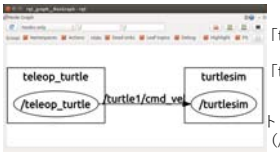
ROSのテスト 2.2.2 ROSの動作テスト



別のターミナルを開いて
「Ctrl+Alt+T」

$ rosrun rqt_graph rqt_graph

現在実行中のノードの情報を
見ることができる



「teleop_turtle」の名で
turtle_teleop_keyノードと
「turtlesim」の名で
turtlesim_nodeノードが実行中

トピックメッセージ通信
（/turtle1/cmd_vel）が行われている

2016/11/22

ROSの基本用語

ROSの基本用語

- マスター（master）
  - roscoreコマンドで実行されるサーバ（ネームサーバ）
  - ノード間の接続（通信）を管理
- ノード（node）
  - 最小の実行プログラム
  - 配信者（publisher）、購読者（subscriber）がある
- パッケージ（package）
  - ノードの集合
  - パッケージ内のノードを接続することで特定の機能を提供
- メタパッケージ（metapackage）
  - パッケージの集合

ROSの基本用語

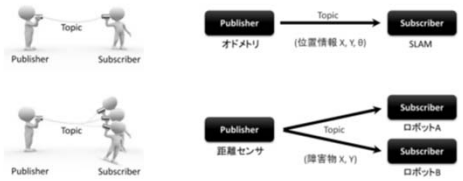
- メッセージ（message）
  - ノード間でやり取りされる情報
  - トピック通信とサービス通信がある
- トピック（topic）
  - 一方向で非同期方式のメッセージ送受信方式
  - 送信側＝配信者　・　受信側＝購読者
- サービス（service）
  - 双方向で同期方式のメッセージ通信方式
  - リクエスト（request）とレスポンス（response）

- 配信，配信者（publish and publisher）が送信
- 購読，購読者（subscribe and subscriber）が受信

ROSの基本用語

トピックメッセージ通信

一方向の通信

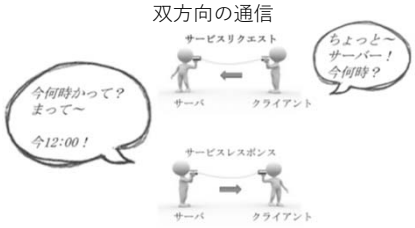


一対一，一対多も可能

ROSの基本用語

サービスメッセージ通信

双方向の通信



一対一のみ

ROSの基本用語

- rosrun
  - １つのノードを起動するコマンド
- roslaunch
  - 複数のノードを一度に起動するコマンド
- rostopic
  - トピック通信を監視するコマンド

ROSの基本用語

- パラメータ（parameter）
  - ノード実行中に変更可能な変数
- パラメータサーバ（parameter server）
  - 全てのノードのパラメータを一括管理するサーバ
  - マスターと同時に起動される

2016/11/22

ROSコマンド

2016/11/22

ROSコマンド

- シェル（shell）環境で、コマンドを使って処理を行う
  - ✓ファイルシステムの利用
  - ✓ソースコードの編集
  - ✓ビルド & デバッグ
  - ✓パッケージ管理
- UbuntuでROSを開発するには..
  - ➡ Linuxコマンド ＋ ROSコマンドの習得

2016/11/22

ROSコマンド

- ROSシェルコマンド
  - roscd　：ROSパッケージまたはスタックへ移動
- ROS実行コマンド
  - roscore　：master（ネームサービス）＋ rosout（stdout/ stderr）＋ parameter server
  - rosrun　：パッケージの１つのノードを実行
  - roslaunch　：パッケージの複数個のノードを実行
- ROS情報コマンド
  - rosnode　：ROSのノード情報を取得
  - rostopic　：ROSトピックの情報を取得
  - rosservice　：ROSサービス情報を取得
- ROS catkinコマンド
  - catkin_create_pkg：catkinビルドシステムによるパッケージの自動生成
  - catkin_init_workspace：catkinビルドシステムの作業フォルダの初期化
  - catkin_make　：catkinビルドシステムをベースにしたビルド命令

2016/11/22

ROSコマンド

- LinuxのbashシェルコマンドをROSで使用
  ros（接頭辞）＋ 接尾辞（cd，pd，d，ls，ed，cp，runなど）
- roscd：ROSディレクトリに移動
  $ roscd［パッケージ名］

  使用例：
  $ roscd turtlesim

  （結果）
  /opt/ros/indigo/share/turtlesim$

2016/11/22

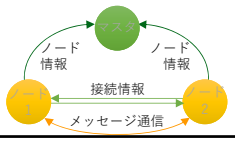
ROS実行コマンド

サーバの起動

- roscore：ノード間のメッセージ通信時に接続情報を管理するマスタ（ネームサーバ）

  $ roscore

  （結果）
  - ✓ XMLRPCでサーバを起動
  - ✓ ノード間の接続のためにノードの名前、トピック、サービスの名前、メッセージの形式、URIアドレスとポートを登録
  - ✓ 登録された情報を、要求に応じて他のノードに通知



2016/11/22

roscoreの動作画面

```
.. logging to /home/#####/.ros/log/****.log          ←ログを保存するファイル
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt                    ←「Ctrl-C」でROSコアを終了できる
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://#####                ← roslaunch server の情報
ros_comm version 1.11.9

SUMMARY
========
PARAMETERS                ← /rosdistro および /rosversionのパラメータサーバ
 * /rosdistro: indigo
 * /rosversion: 1.11.9

NODES
auto-starting new master
process[master]: started with pid [*****]
ROS_MASTER_URI=http://#####:11311/          ← ROS_MASTER_URI の情報
setting /run_id to *****
process[rosout-1]: started with pid [#####]
started core service [/rosout]     ← /rosoutのサービス、 /rosoutノードの実行
```

2016/11/22

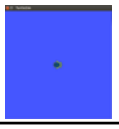
ROS実行コマンド

一つのノードの実行

- rosrun：指定されたパッケージの一つのノードを実行
  $ rosrun［パッケージ名］［ノード名］

  使用例：
  $ rosrun turtlesim turtlesim_node
  ✓ パッケージ「turtlesim」のノード「turtlesim_node」をデフォルトの「/turtlesim」という名前で実行

  （結果）
  [INFO] [#####.*****]: Starting turtlesim with node name /turtlesim
  [INFO] [#####.*****]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]



2016/11/22

ROS実行コマンド

一つのノードの実行

- rosrun：指定されたパッケージの一つのノードを名前を変えて実行
  $ rosrun［パッケージ名］［ノード名］ __name:=[名前]

  _が２つ

  使用例：
  $ rosrun turtlesim turtlesim_node __name:=my_turtle
  ✓ パッケージ「turtlesim」のノード「turtlesim_node」を「/my_turtle」という名前で実行

2016/11/22

ROS実行コマンド

複数のノードの同時実行

- roslaunch：パッケージの複数個のノードを実行
  ROSでは、パラメータ設定も同時にできる
  $ roslaunch［パッケージ名］［ランチファイル名］

  使用例：
  $ roslaunch openni_launch openni.launch
  ✓ 「openni_launch」パッケージの「openni.launch」を実行

  （結果）
  ✓ 20以上のノードと10個のパラメータサーバが実行

  ＊もしopenNI関連のパッケージが未インストールなら
  $ sudo apt-get install ros-indigo-openni-camera ros-indigo-openni-launch

2016/11/22

ROS情報コマンド

- トピック、サービス、ノード、パラメータなどの情報を確認する
  - rosnode ：ROSのノード情報を取得
  - rostopic ：ROSトピックの情報を取得
  - rosservice ：ROSサービス情報を取得
  - rosparam ：ROSパラメータ情報を取得・変更
  - rosbag ：ROSメッセージを記録・再生

- 事前準備
  1. 現在開いている端末を全て閉じ、新しい端末を開く
  2. 以下のコマンドを実行
     $ roscore
     $ rosrun turtlesim turtlesim_node
     $ rosrun turtlesim turtle_teleop_key

roscore の実行

turtlesimパッケージの turtlesim_nodeノードの実行

turtlesimパッケージの turtle_teleop_keyノードの実行

別のターミナルを開いて「Ctrl+Alt+T」

- turtlesim_node：青色画面に亀が表示
- turtle_teleop_key：矢印キーで亀を操作

2016/11/22

ROS情報コマンド

rosnodeコマンド

- rosnode：ノード情報を取得

- ノード
  - 定義：最小単位の実行プロセッサ＝一つの実行可能なプログラム
    例：ロボット制御
    ノード：センサドライブ、モータ駆動、エンコーダ入力、
    障害物判断、ナビゲーション、など
  - マスターにノード情報を登録
    ノード名、発行者名、加入者名、トピック名、サービス名、メッセージ形式、URIアドレス、ポート
  - 通信方法
    ✓ XMLRPC：マスタ・ノード間の通信
    ノード間の接続要求・応答
    ✓ TCPROS：ノード間の通信、メッセージ通信 （XMLPCとTCP/IP通信系）

2016/11/22

ROS情報コマンド

rosnodeコマンド

- rosnode list：ROSコアに接続された全てのノードのリストを表示
  $ rosnode list

  /rosout
  /teleop_turtle
  /turtlesim

- rosnode ping [ノード名]：指定されたノードとの接続をテスト
  $ rosnode ping /turtlesim

  rosnode : node is [/turtlesim]
  pinging /turtlesim with a timeout of 3.0s
  xmlrpc reply from http://####:45470/ time=0.344992ms
  ：

＊もしそのノードと接続されていない場合は
ERROR：connection refused to[http://xxx.xxx.xxx.xxx:xxxxx/]
のようなエラーメッセージが表示

2016/11/22

ROS情報コマンド

rosnodeコマンド

- rosnode info [ノード名] ：指定されたノードの情報を確認
  $ rosnode info /turtlesim

  Node[/turtlesim]
  Publications:
  ＊ /turtle1/color_sensor [turtlesim/Color]
  ...省略...

  Subscriptions:
  ＊ /turtle1/cmd_vel [geometry_msgs/Twist]
  ...省略...

  Services:
  ＊ /turtle1/teleport_absolute
  ...省略...

2016/11/22

ROS情報コマンド

rosnodeコマンド

- rosnode machine [PC名またはIP]：PC上で実行中のノードを表示
  $ rosnode machine xxx.xxx.xxx.xxx

  /rosout
  /teleop_turtle
  /turtlesim

- rosnode kill [ノード名] ：指定されたノードの停止
  $ rosnode kill /turtlesim

  killing /turtlesim
  killed

- rosnode cleanup：接続情報が確認できないノードの登録情報の削除
  $ rosnode cleanup

＊端末ウィンドウで、［Ctrl + C］で直接ノードをシャットダウンも可

2016/11/22

ROS情報コマンド

rostopicコマンド

- rostopic：ROSトピックの情報を取得
- トピック
  - 定義：発行者ノードが発行するメッセージにつくタグ・キーワード
  - トピックの受信を希望するノードは、そのトピックを発行している発行者ノードの情報を受け取る
  - 受信した情報に基づいて、受信者ノードは発行者ノードと直接接続して、メッセージを送受信または要求・応答受ける
  - 非同期式通信：必要に応じてデータの送受信が可能
  - 一度の接続で継続的にメッセージの送受信

一旦実行を停止し、再度実行
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key

2016/11/22

ROS情報コマンド

rostopicコマンド

- rostopic list：現在アクティブなトピックの一覧を表示
  $ rostopic list

  /rosout
  /rosout_agg
  /turtle1/cmd_vel
  /turtle1/color_sensor
  /turtle1/pose

- rostopic list -v：現在送受信されている全てのトピックのリストを表示
  $ rostopic list -v

  /Published topics：
  ＊ /turtle1/color_sensor [turtlesim/Color] 1 publisher
  ：
  Subscribed topics：
  ＊ /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber

2016/11/22

ROS情報コマンド

rostopicコマンド

- rostopic echo [トピック名]：指定したトピックのメッセージの内容をリアルタイムに表示
  $ rostopic echo /turtle1/pose

  x：5.35244464874 ← 座標値
  y：5.544444561
  theta：0.0 ← 姿勢（向き）
  linear_velocity：0.0 ← 速度
  angular_velocity：0.0 ← 角速度

- rostopic find [タイプ名]：指定したタイプのメッセージを使用するトピックを表示
  $ rostopic find turtlesim/Pose

  /turtle1/pose

2016/11/22

ROS情報コマンド

rostopicコマンド

- rostopic type [トピック名]：指定したトピックのメッセージタイプを表示
  $ rostopic type /turtle1/pose

  turtlesim/Pose

- rostopic bw [タイプ名]：指定したトピックのメッセージデータの帯域幅(bandwidth) を表示
  $ rostopic bw /turtle1/pose

  subscribed to [/turtle1/pose]
  average：1.27KB/s
  mean: 0.02KB min: 0.02KB max: 0.02KB window: 62
  …省略…

ROS情報コマンド

rostopicコマンド

- rostopic hz [トピック名]：指定したトピックのメッセージ発行周期を表示
  $ rostopic hz /turtle1/pose

  subscribed to [/turtle1/pose]
  average rate: 62.502
  min: 0.016s max: 0.016s std dev: 0.00005s window: 62
  …省略…

- rostopic info [タイプ名]：指定したトピックの情報を表示
  $ rostopic info /turtle1/pose

  Type: turtlesim/Pose ← メッセージタイプ
  Publishers:
  ＊ /turtlesim（http://xxxx:42443/）← 発行者ノード
  Subscribers: None ← 購買者ノード

ROS情報コマンド

rostopicコマンド

- rostopic pub [トピック名] [メッセージタイプ] [パラメータ]：指定したトピックの名前でメッセージを発行
  $ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'

  publishing and latching message for 3.0 second

  《オプションの説明》
  - -1 :メッセージを1回発行
  - /turtle1/cmd_vel：指定したトピック名
  - geometry_msgs/ Twist：発行されているメッセージタイプ名
  - -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]' :
    並進速度 角速度

ROS情報コマンド

rosserviceコマンド

- rosservice：サービスの情報を取得
- サービス
  - 種類：要求があった場合に応答をするサービスサーバ
    要求して応答を受信するサービスクライアント
  - 同期式通信：要求があった時のみデータを送受信
  - 1回限りのメッセージ通信＝サービスの要求と応答が完了すると，ノード間の接続は切断

一旦実行を停止し、再度実行
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key

ROS情報コマンド

rosserviceコマンド

- rosservice list：アクティブなサービスの一覧を表示
  $ rosservice list

  /clear
  /kill
  /reset
  /rosout/get_loggers
  /rosout/set_logger_level
  /spawn
  /teleop_turtle/get_loggers
  /teleop_turtle/set_logger_level
  /turtle1/set_pen
  /turtle1/teleport_absolute
  /turtle1/teleport_relative
  /turtlesim/get_loggers
  /turtlesim/set_logger_level

ROS情報コマンド

rosserviceコマンド

- rosservice type [サービス名]：サービスタイプを表示
  $ rosservice type /turtle1/set_pen

  turtlesim/SetPen

- rosservice find [サービスタイプ]：
  指定したサービスタイプのサービスを検索
  $ rosservice find turtlesim/SetPen

  /turtle1/set_pen

- rosservice uri [サービス名]：ROSRPC uriサービスを出力
  $ rosservice uri /turtle1/set_pen

  rosrpc://xxx.xxx.xxx.xxx:49714

ROS情報コマンド

rosserviceコマンド

- rosservice args [サービス名]：サービスパラメータを出力
  $ rosservice args /turtle1/set_pen

  r g b width off

- rosservice call [サービス名] [パラメータ]：
  指定したサービスパラメータの設定
  $ rosservice call /turtle1/set_pen 255 0 0 5 0

  《オプション》
  - r = 255, g = 0, b = 0 : ペンの色
  - width = 5：線の太さ
  - off = 0：線が見えるようにする

2016/11/22

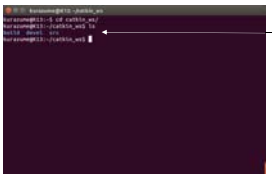
ROS　実習

2016/11/22

一度リブートしておきましょう

パッケージの作成

- ワークディレクトリ　~/catkin_ws
  $ cd ~/catkin_ws
  $ ls



build devel src の3つの
フォルダがある

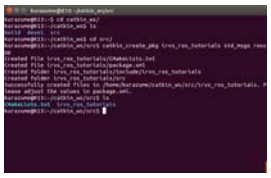
プログラム開発で使うのは
srcフォルダ

build　ビルド関連ファイル
devel　msg又はsrvのヘッダーファイルとユーザーパッケージのライブラリ、実行ファイル
src　ユーザーパッケージ

2016/11/22

パッケージの作成

- src ディレクトリ内に irvs_ros_tutorial パッケージを作成
  $ cd ~/catkin_ws/src
  $ catkin_create_pkg irvs_ros_tutorials std_msgs roscpp



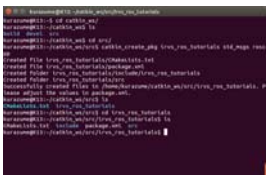
パッケージが使うパッケージ

srcフォルダの下に
CMakeLists.txt
irvs_ros_tutorialフォルダ
ができる

2016/11/22

パッケージの作成

- irvs_ros_tutorialディレクトリ内を見てみる
  $ cd irvs_ros_tutorials
  $ ls



Irvs_ros_tutorialの中には
include フォルダ
src フォルダ
CMakeLists.txt
package.xml
がある

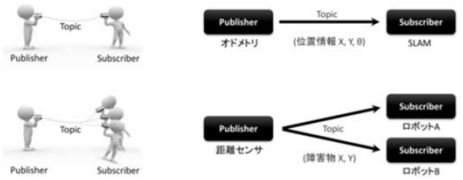
include　→　ヘッダーファイルフォルダ
src　→　ソースコードフォルダ
CMakeLists.txt　→　ビルド設定ファイル
package.xml　→　パッケージの設定ファイル

2016/11/22

トピック通信の実習

2016/11/22

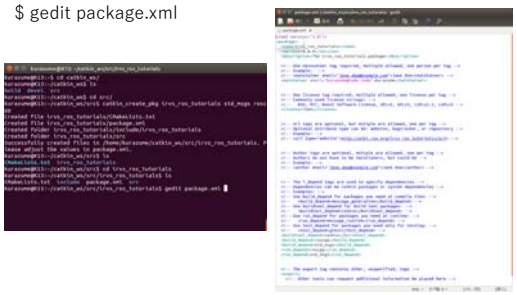
ROSの基本用語
トピックメッセージ通信



一対一，一対多も可能

トピック通信

- パッケージの設定ファイル(package.xml)の確認
  $ gedit package.xml



トピック通信

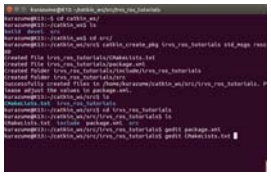
$ gedit package.xml

```
<!-- The *_depend tags are used to specify dependencies -->
<!-- Dependencies can be catkin packages or system dependencies -->
<!-- Examples: -->
<!-- Use build_depend for packages you need at compile time: -->
<!--   <build_depend>message_generation</build_depend> -->
<!-- Use buildtool_depend for build tool packages: -->
<!--   <buildtool_depend>catkin</buildtool_depend> -->
<!-- Use run_depend for packages you need at runtime: -->
<!--   <run_depend>message_runtime</run_depend> -->
<!-- Use test_depend for packages you need only for testing: -->
<!--   <test_depend>gtest</test_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>roscpp</build_depend>
<build_depend>std_msgs</build_depend>
<run_depend>roscpp</run_depend>
<run_depend>std_msgs</run_depend>

<!-- The export tag contains other, unspecified, tags -->
<export>
  <!-- Other tools can request additional information be placed here -->

</export>
</package>
```

依存するパッケージ
catkin_create_pkg で書いたもの
std_msgs, roscpp

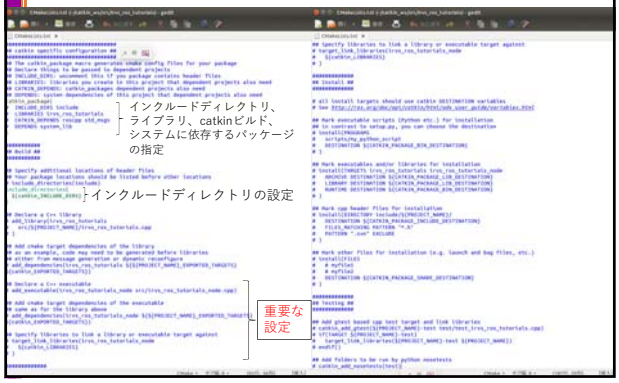
トピック通信

- ビルド設定ファイル(CMakeLists.txt)の修正
  $ gedit CMakeLists.txt



CMakeLists.txtの中身（オリジナル）

CMakeLists.txtの中身（オリジナル）



CMakeLists.txtの変更

CMakeLists.txtの変更

書き換えたものを保存して終了

CMakeLists.txtの変更

ros_tutorial_msg_publisherノード
ros_tutorial_msg_subscriberノードを追加の設定

```
## ros_tutorial_msg_publisherノードの設定
##実行ファイル、ターゲットリンクライブラリ、追加の依存関係などを設定
add_executable(ros_tutorial_msg_publisher src/ros_tutorial_msg_publisher.cpp)
target_link_libraries(ros_tutorial_msg_publisher ${catkin_LIBRARIES})
add_dependencies(ros_tutorial_msg_publisher irvs_ros_tutorials_generate_messages_cpp)

## ros_tutorial_msg_subscriberノードの設定
##実行ファイル、ターゲットリンクライブラリ、追加の依存関係などを設定
add_executable(ros_tutorial_msg_subscriber src/ros_tutorial_msg_subscriber.cpp)
target_link_libraries(ros_tutorial_msg_subscriber ${catkin_LIBRARIES})
add_dependencies(ros_tutorial_msg_subscriber irvs_ros_tutorials_generate_messages_cpp)
```


メッセージファイルの作成

```
$ cd ~/catkin_ws/src/irvs_ros_tutorials
$ mkdir msg
$ cd msg
$ gedit msgTutorial.msg
```

msgTutorial.msgの中身
int32 data

保存して終了

配信者ノードの作成

```
$ roscd irvs_ros_tutorials/src
  （あるいは　$ cd ~/catkin_ws/src/irvs_ros_tutorials/src）
$ gedit ros_tutorial_msg_publisher.cpp
```

http://robotics.ait.kyushu-u.ac.jp/~kurazume/ROS/ros_tutorial_msg_publisher.cpp
ユーザ　：　temp
パスワード：　temp

ros_tutorial_msg_publisher.cpp

http://irvs.github.io/rosbook_jp/　134頁

```
// ROSメインヘッダーファイル
// ROSプログラミングを行う際に必要となるROSファイルのインクルードを行う。
// 後述するROS_INFO関数などを使用できるようになる。
#include "ros/ros.h"

// msgTutorialメッセージファイルのヘッダー
// CMakeLists.txtでビルド後に自動的に生成されるように設定した
// メッセージファイルのヘッダーをインクルードする。
#include "irvs_ros_tutorials/msgTutorial.h"

// 配信者ノードのメイン関数
int main(int argc, char **argv)
{
        // ノード名の初期化
        ros::init(argc, argv, "ros_tutorial_msg_publisher");
        // ROSシステムとの通信のためのノードハンドルを宣言
        ros::NodeHandle nh;

        // 配信者ノードの宣言
        // irvs_ros_tutorialsパッケージのmsgTutorialメッセージファイルを
        // 利用した配信者ros_tutorial_pubを宣言する。
        // トピック名をros_tutorial_msg とし、配信者キュー( queue )の
        // サイズを100に設定する。
        // 配信者キューには、メッセージを送る際、メッセージデータを蓄積する。
        // http://wiki.ros.org/msg
        ros::Publisher ros_tutorial_pub =
        nh.advertise<irvs_ros_tutorials::msgTutorial>("ros_tutorial_msg", 100);

        // ループの周期を設定する。"10"は10Hzを表し、0.1秒間隔で繰り返される
        // http://wiki.ros.org/roscpp/Overview/Time
        ros::Rate loop_rate(10);

        // メッセージに使用する変数の宣言
        int count = 0;

        // ros::ok()はROSの動作が正常であるならtrueを返す関数である。
        while (ros::ok())
        {
                // msgTutorialメッセージファイル形式でmsg変数を宣言する。
                irvs_ros_tutorials::msgTutorial msg;

                // 変数countを使用して、メッセージの値を定める。
                msg.data = count;

                // ROS_INFOというROS関数を使用して、count変数を表示する。
                ROS_INFO("send msg = %d", count);

                // メッセージを発行する。約0.1秒間隔で発行される。
                ros_tutorial_pub.publish(msg);

                // よって定められたループサイクルになるように、スリープに入る
                loop_rate.sleep();

                // count変数に1ずつ増加
                ++count;
        }
        return 0;
```

配信者ノードの作成

$ roscd irvs_ros_tutorials/src
（あるいは　$ cd ~/catkin_ws/src/irvs_ros_tutorials/src）
$ gedit ros_tutorial_msg_subscriber.cpp

http://robotics.ait.kyushu-u.ac.jp/~kurazume/ROS/ros_tutorial_msg_subscriber.cpp
ユーザ　　：　temp
パスワード：　temp

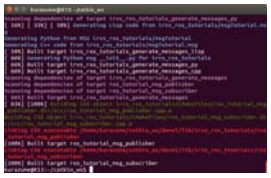
ros_tutorial_msg_publisher.cpp

http://irvs.github.io/rosbook_jp/　136頁

```
// ROSメインヘッダーファイル
// ROSプログラミングを行う際に必要となるROSファイルのインクルードを行う。
// 後述するROS_INFO関数なども使用できるようになる。
#include "ros/ros.h"

// msgTutorialメッセージファイルのヘッダー
// CMakelists.txtでビルド後に自動的に生成されるように設定した
// メッセージファイルのヘッダーをインクルードする。
#include "irvs_ros_tutorials/msgTutorial.h"

// メッセージを受信したときに動作するコールバック関数を定義
// irvs_ros_tutorialsパッケージのmsgTutorialメッセージを受信する
void msgCallback(const irvs_ros_tutorials::msgTutorial::ConstPtr& msg)
{
        // 受信したメッセージを表示する。
        ROS_INFO("recieve msg: %d", msg->data);
}
// 購読者ノードのメイン関数
int main(int argc, char **argv)
{
        // ノード名の初期化
        ros::init(argc, argv, "ros_tutorial_msg_subscriber");
        // ROSシステムとの通信のためのノードのハンドルを宣言
        ros::NodeHandle nh;

        // 購読者ノードの宣言
        // irvs_ros_tutorialsパッケージのmsgTutorialメッセージファイルを
        // 利用した購読者ros_tutorial_subを宣言する。
        // トピック名をros_tutorial_msg とし、購読者キュー( queue )の
        // サイズを100に設定する。
        // 購読者キューには、配信者から送信されてくるメッセージが蓄積される。
        ros::Subscriber ros_tutorial_sub = nh.subscribe("ros_tutorial_msg",
        100, msgCallback);

        // メッセージが受信されるまで待機し、受信が行われた場合、
        // コールバック関数を実行する。
        ros::spin();

        return 0;
}
```


トピック通信

• ビルド
$ cd ~/catkin_ws
$ catkin_make

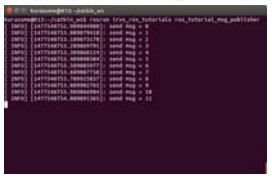


「100%」と表示されたら成功

トピック通信

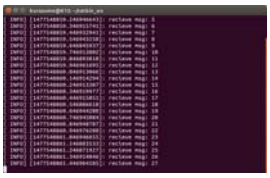
• 実行
新しいターミナルを開いて（「Ctrl+Alt+T」）
$ roscore

新しいターミナルを開いて（「Ctrl+Alt+T」）
$ rosrun irvs_ros_tutorials ros_tutorial_msg_publisher



トピック通信

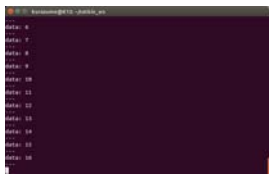
• 実行
新しいターミナルを開いて（「Ctrl+Alt+T」）
$ rosrun irvs_ros_tutorials ros_tutorial_msg_subscriber



データがpublisherからsubscriberへ送られている

トピック通信

• 実行
新しいターミナルを開いて（「Ctrl+Alt+T」）
$ rostopic echo /ros_tutorial_msg



送られているデータを見ることができる

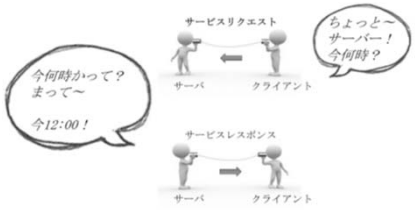
トピック通信

- 接続の確認
  $ rqt_graph

サービス通信の実習

ROSの基本用語

サービスメッセージ通信



一対一のみ

サービス通信

- パッケージの設定ファイル(package.xml)の修正
  $ roscd irvs_ros_tutorials
  （あるいは　$ cd ~/catkin_ws/src/irvs_ros_tutorials）
  $ gedit CMakeLists.txt

CMakeLists.txtの変更

CMakeLists.txtの変更

ros_tutorial_srv_serverノード
ros_tutorial_srv_clientノードを追加の設定

```
## ros_tutorial_srv_serverサービスサーバノードの設定
##実行ファイル、ターゲットリンクライブラリ、追加の依存関係などを設定
add_executable(ros_tutorial_srv_server src/ros_tutorial_srv_server.cpp)
target_link_libraries(ros_tutorial_srv_server ${catkin_LIBRARIES})
add_dependencies(ros_tutorial_srv_server irvs_ros_tutorials_generate_messages_cpp)

## ros_tutorial_srv_clientサービスクライアントノードの設定
##実行ファイル、ターゲットリンクライブラリ、追加の依存関係などを設定
add_executable(ros_tutorial_srv_client src/ros_tutorial_srv_client.cpp)
target_link_libraries(ros_tutorial_srv_client ${catkin_LIBRARIES})
add_dependencies(ros_tutorial_srv_client irvs_ros_tutorials_generate_messages_cpp)
```

メッセージファイルの作成

$ roscd irvs_ros_tutorials
$ mkdir srv
$ cd srv
$ gedit srvTutorial.srv

srvTutorial.srvの中身
int64 a
int64 b
---
int64 result

保存して終了

サービスサーバノードの作成

$ roscd irvs_ros_tutorials/src
$ gedit ros_tutorial_srv_server.cpp

http://robotics.ait.kyushu-u.ac.jp/~kurazume/ROS/ros_tutorial_srv_server.cpp
ユーザ　：　temp
パスワード：　temp

ros_tutorial_srv_server.cpp

http://irvs.github.io/rosbook_jp/　142頁

```
// ROSメインヘッダーファイル
// ROSプログラミングを行う際に必要となるROSファイルのインクルードを行う。
// 後述するROS_INFO関数などを使用できるようになる。
#include "ros/ros.h"

// srvTutorial サービスファイルのヘッダー
// CMakelists.txtでビルド後に自動的に生成されるように設定した
// サービスファイルのヘッダーをインクルードする。
#include "irvs_ros_tutorials/srvTutorial.h"

// サービスリクエストがある場合は、以下の処理を実行する。
// サービスリクエストは、req、サービスのレスポンスは、resに設定した。
bool calculation(irvs_ros_tutorials::srvTutorial::Request &req,
irvs_ros_tutorials::srvTutorial::Response &res)
{
        // サービスリクエストで受けたaとbの値を加えて、
        // サービスのレスポンス値に格納する。
        res.result = req.a + req.b;

        // サービスリクエストで受けたa、bの値の表示、および
        // サービスレスポンスに対応するresultの値を出力する。
        ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
        ROS_INFO("sending back response: [%ld]", (long int)res.result);
        return true;
}

// サービスサーバノードのメイン関数
int main(int argc, char **argv)
{
        // ノードの初期化
        ros::init(argc, argv, "ros_tutorial_srv_server");
        // ROSシステムとの通信のためのノードのハンドルを宣言
        ros::NodeHandle nh;

// サービスサーバ宣言
// irvs_ros_tutorialsパッケージのsrvTutorialサービスファイルを利用した
// サービスサーバros_tutorial_service_serverを作成する。
// サービス名はros_tutorial_srvで、サービスリクエストがあったとき、
// calculation という関数を実行するように設定している。
ros::ServiceServer ros_tutorial_service_server =
nh.advertiseService("ros_tutorial_srv", calculation);

// サービスサーバが実行したことを表示する。
ROS_INFO("ready srv server!");

// サービスリクエストを待機する
ros::spin();

return 0;
}
```


サービスクライアントノードの作成

$ roscd irvs_ros_tutorials/src
（$ cd ~/catkin_ws/src/irvs_ros_tutorials/srcと同じ）
$ gedit ros_tutorial_srv_client.cpp

http://robotics.ait.kyushu-u.ac.jp/~kurazume/ROS/ros_tutorial_srv_client.cpp
ユーザ　：　temp
パスワード：　temp

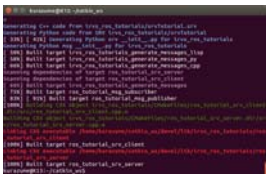
ros_tutorial_srv_client.cpp

http://irvs.github.io/rosbook_jp/　144頁

```
// ROSメインヘッダーファイル
// ROSプログラミングを行う際に必要となるROSファイルのインクルードを行う。
// 後述するROS_INFO関数などを使用できるようになる。
#include "ros/ros.h"

// srvTutorialサービスファイルのヘッダー
// CMakelists.txtでビルド後に自動的に生成されるように設定したサービスファ
// イルのヘッダーをインクルードする。
#include "irvs_ros_tutorials/srvTutorial.h"

// atoll関数を使用するためのライブラリ
#include <cstdlib>

// サービスクライアントノードのメイン関数
int main(int argc, char **argv)
{
        // ノード名の初期化
        ros::init(argc, argv, "ros_tutorial_srv_client");

        // 入力値エラー処理
        if (argc != 3)
        {
        ROS_INFO("cmd: rosrun ros_tutorial ros_tutorial_service_client arg0 arg1");
        ROS_INFO("arg0: double number, arg1: double number");
        return 1;
        }
                                                        }
        // ROSシステムとの通信のためのノードのハンドル宣言
        ros::NodeHandle nh;

        // サービスクライアント宣言。
        // irvs_ros_tutorialsパッケージのsrvTutorialサービスファイルを利用した。
        // サービスクライアントros_tutorial_service_clientを作成する。
        // サービス名は「ros_tutorial_srv」である。
ros::ServiceClient ros_tutorial_service_client =
nh.serviceClient <irvs_ros_tutorials::srvTutorial>("ros_tutorial_srv");

// srvという名前でsrvTutorialサービスを利用する
// サービスのファイルを宣言する。
irvs_ros_tutorials::srvTutorial srv;

// サービスリクエスト値をそれぞれのa、bに格納する。
srv.request.a = atoll(argv[1]);
srv.request.b = atoll(argv[2]);

// サービスをリクエストし、レスポンスが返された場合、
// レスポンス値を表示する。
if (ros_tutorial_service_client.call(srv))
{
                ROS_INFO("send srv request.a and b: %ld, %ld",
                (long int)srv.request.a, (long int)srv.request.b);
                ROS_INFO("receive srv.response.result: %ld",
                (long int)srv.response.result);
}
else
{
                ROS_ERROR("Failed to call service ros_tutorial_srv");
                return 1;
}
return 0;
}
```


サービス通信

- ビルド
  $ cd ~/catkin_ws
  $ catkin_make



「100%」と表示されたら成功

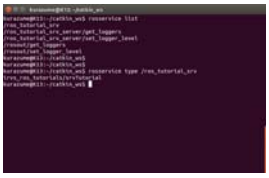
サービス通信

- 実行
  もしroscoreが一つも動いていなければ
  新しいターミナルを開いて（「Ctrl+Alt+T」）
  $ roscore
  新しいターミナルを開いて（「Ctrl+Alt+T」）
  $ rosrun irvs_ros_tutorials ros_tutorial_srv_server

2016/11/22

サービス通信

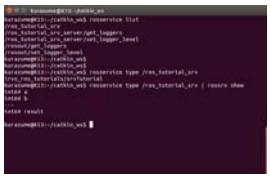
- 確認
  新しいターミナルを開いて（「Ctrl+Alt+T」）
  $ rosservice list
  /ros_tutorial_srvがあることを確認
  $ rosservice type /ros_tutorial_srv
  /ros_tutorial_srvの型が srvTutorialであることを確認



2016/11/22

サービス通信

- 確認
  $ rosservice type /ros_tutorial_srv | rossrv show
  srvTutorialの型の中身が確認できる
  int64 a
  int64 b　　入力
  ---
  int64 result　　出力

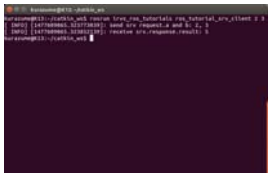


2016/11/22

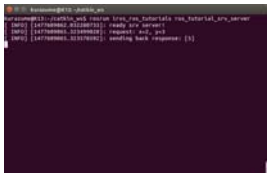
サービス通信

- 実行
  新しいターミナルを開いて（「Ctrl+Alt+T」）
  $ rosrun irvs_ros_tutorials ros_tutorial_srv_client 2 3

ros_tutorial_srv_clientの画面



ros_tutorial_srv_serverの画面



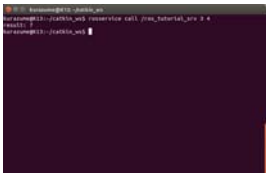
答えの　5　が返ってきた

2016/11/22

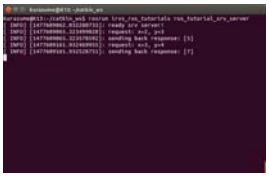
サービス通信

- サービスクライアントノードを使わない方法
  新しいターミナルを開いて（「Ctrl+Alt+T」）
  $ rosservice call /ros_tutorial_srv 3 4

ros_tutorial_srv_clientの画面



ros_tutorial_srv_serverの画面



答えの　7　が返ってきた

2016/11/22

パラメータの実習

2016/11/22

ROSの基本用語

- パラメータ（parameter）
  - ノード実行中に変更可能な変数
- パラメータサーバ（parameter server）
  - 全てのノードのパラメータを一括管理するサーバ
  - マスターと同時に起動される

パラメータを利用したノードの作成

$ roscd irvs_ros_tutorials/src
$ gedit ros_tutorial_srv_server.cpp

http://robotics.ait.kyushu-u.ac.jp/~kurazume/ROS/param/ros_tutorial_srv_server.cpp
ユーザ　：　temp
パスワード：　temp

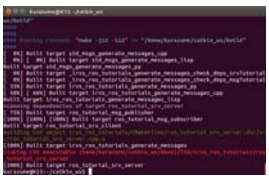
ros_tutorial_srv_server.cpp

http://irvs.github.io/rosbook_jp/　149頁

```
// ROSメインヘッダーファイル
// ROSプログラミングを行う際に必要となるROSファイルのインクルードを行う。
// 後述するROS_INFO関数などを使用できるようになる。
#include "ros/ros.h"

// srvTutorial サービスファイルのヘッダー
// CMakeLists.txtでビルド後に自動的に生成されるように設定したサービスファ
// イルのヘッダーをインクルードする。
#include "irvs_ros_tutorials/srvTutorial.h"

//パラメータのオプション
#define PLUS 1 // 加算
#define MINUS 2 // 減算
#define MULTIPLICATION 3 // 乗算
#define DIVISION 4 // 割算

int g_operator = PLUS;

// サービスリクエストがある場合は、以下の処理を実行する。
// サービスリクエストは、req、サービスのレスポンスは、resに設定した。
bool calculation(irvs_ros_tutorials::srvTutorial::Request &req,
                 irvs_ros_tutorials::srvTutorial::Response &res)
{
                 //パラメータ値に基づいて演算子を変更し、サービスリクエストを受けた
                 //aとbの値を計算して、結果をサービスレスポンス値に保存する。
                 switch (g_operator){
                 case PLUS:
                                  res.result = req.a + req.b; break;
                 case MINUS:
                                  res.result = req.a - req.b; break;
                 case MULTIPLICATION:
                                  res.result = req.a * req.b; break;
                 case DIVISION:
                                  if (req.b == 0){
                                                   res.result = 0; break;
                                  }
                                  else{
                                                   res.result = req.a / req.b; break;
                                  }
                 default:
                                  res.result = req.a + req.b; break;
                 }

                 // サービスリクエストで受けた、bの値の表示、および
                 // サービスレスポンスに対応するresultの値を出力する。
                 ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
                 ROS_INFO("sending back response: [%ld]", (long int)res.result);
                                  return true;
}

// サービスサーバノードのメイン関数
int main(int argc, char **argv)
{
                 //ノード名の初期化。
                 ros::init(argc, argv, "ros_tutorial_srv_server");
                 // ROSシステムとの通信のためのノードのハンドルを宣言
                 ros::NodeHandle nh;

                 // ROSシステムとの通信のためのノードのハンドルを宣言(Private)
                 // http://wiki.ros.org/Names
                 ros::NodeHandle nh_priv("~");

                 // パラメータの初期設定, http://wiki.ros.org/rosparam
                 nh_priv.setParam("calculation_method", PLUS);

                 //サービスサーバ宣言。
                 // irvs_ros_tutorialsパッケージのsrvTutorialサービスファイルを利用した
                 // サービスサーバ"ros_tutorial_service_server"を作成する。
                 // サービス名はros_tutorial_srv下、サービスリクエストがあったとき、
                 // calculationという関数を実行するように設定している。
                 ros::ServiceServer ros_tutorial_service_server =
                 nh.advertiseService("ros_tutorial_srv",calculation);

                 //サービスサーバが実行したことを表示する。
                 ROS_INFO("ready srv server!");

                 // ループの周期を設定する。"10"は10Hzを表し、
                 // 0.1秒間隔で繰り返される
                 ros::Rate r(10);

                 while (ros::ok())
                 {
                                  // 演算子をパラメータから取得した値に変更する。
                                  nh_priv.getParam("calculation_method",g_operator);

                                  // コールバック関数の処理ルーチン
                                  ros::spinOnce();

                                  // よって定められたループサイクルになるように、スリープに入る
                                  r.sleep();
                 }
                 return 0;
}
```


パラメータの使用例

- ビルド
  $ cd ~/catkin_ws
  $ catkin_make



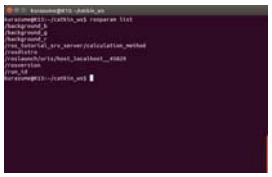
「100%」と表示されたら成功

パラメータの使用例

- 実行
  もしroscoreが一つも動いていなければ
  新しいターミナルを開いて（「Ctrl+Alt+T」）
  $ roscore
  新しいターミナルを開いて（「Ctrl+Alt+T」）
  $ rosrun irvs_ros_tutorials ros_tutorial_srv_server

パラメータの使用例

- 実行
  新しいターミナルを開いて（「Ctrl+Alt+T」）
  $ rosparam list



/ros_tutorial_srv_server/calculation_method というパラメータがあることを確認

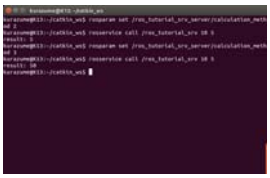
パラメータの使用例

- 実行
  新しいターミナルを開いて（「Ctrl+Alt+T」）
  $ rosparam set /ros_tutorial_srv_server/calculation_method 2
  $ rosservice call /ros_tutorial_srv 10 5　→パラメータを減算に変更

答えの　5　が返ってきた

パラメータの使用例

- 実行
  $ rosparam set /ros_tutorial_srv_server/calculation_method 3
  $ rosservice call /ros_tutorial_srv 10 5　→パラメータを乗算に変更



答えの　50　が返ってきた

roslaunchを用いた複数ノードの起動

roslaunchを用いた複数ノードの起動

- rosrun
  - 一つのノードを実行するコマンド
- roslaunch
  - 複数のノードを実行するコマンド
- roslaunchファイル
  - *.launch
    - 起動するノードを設定
    - ノードの実行時のパラメータの変更
    - ノード名の変更
    - ノードの名前空間の設定
    - ROS_ROOTとROS_PACK
    - AGE_PATHの設定、環境変数の変更なども可能

roslaunchを用いた複数ノードの起動

- ２つの配信者ノードと２つの購読者ノードを同時に起動したい
- それぞれの組を独立に通信させたい

ros_tutorial_msg_publisher → ros_tutorial_msg_subscriber

ros_tutorial_msg_publisher → ros_tutorial_msg_subscriber

- 制約　「実行ノード名は，ユニークでなくてはならない」

roslaunchを使う

roslaunchファイルの作成

$ roscd irvs_ros_tutorials
$ mkdir launch
$ cd launch
$ gedit union.launch

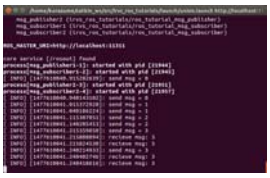
http://robotics.ait.kyushu-u.ac.jp/~kurazume/ROS/union.launch
ユーザ　：　temp
パスワード：　temp

```
<launch>
<node pkg = "irvs_ros_tutorials" type = "ros_tutorial_msg_publisher" name = "msg_publisher1" />
<node pkg = "irvs_ros_tutorials" type = "ros_tutorial_msg_subscriber" name = "msg_subscriber1" />
<node pkg = "irvs_ros_tutorials" type = "ros_tutorial_msg_publisher" name = "msg_publisher2" />
<node pkg = "irvs_ros_tutorials" type = "ros_tutorial_msg_subscriber" name = "msg_subscriber2" />
</launch>
```

roslaunchの実行

$ cd ~/catkin_ws
$ roslaunch irvs_ros_tutorials union.launch --screen

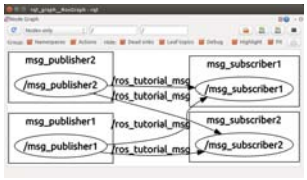
画面に出力するオプション



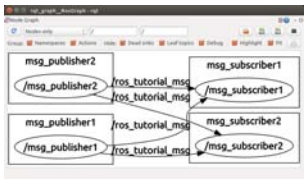
roslaunchの実行

新しいターミナルを開いて（「Ctrl+Alt+T」）
$ rosnode list

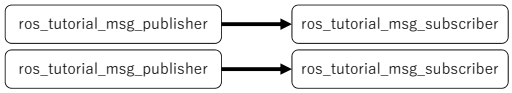
$ rqt_graph



roslaunchを用いた複数ノードの起動



• 以下のようになっていない！



ネームスペースを使う

roslaunchファイルの作成

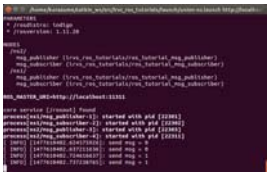
$ roscd irvs_ros_tutorials/launch
$ gedit union-ns.launch

http://robotics.ait.kyushu-u.ac.jp/~kurazume/ROS/union-ns.launch
ユーザ　：　temp
パスワード：　temp

```
<launch>
<group ns = "ns1">          ← ネームスペース１
<node pkg = "irvs_ros_tutorials" type = "ros_tutorial_msg_publisher" name = "msg_publisher" />
<node pkg = "irvs_ros_tutorials" type = "ros_tutorial_msg_subscriber" name = "msg_subscriber" />
</group>
<group ns = "ns2">          ← ネームスペース２
<node pkg = "irvs_ros_tutorials" type = "ros_tutorial_msg_publisher" name = "msg_publisher" />
<node pkg = "irvs_ros_tutorials" type = "ros_tutorial_msg_subscriber" name = "msg_subscriber" />
</group>
</launch>
```


roslaunchの実行

$ cd ~/catkin_ws
$ roslaunch irvs_ros_tutorials union-ns.launch --screen



roslaunchの実行

新しいターミナルを開いて（「Ctrl+Alt+T」）
$ rosnode list

$ rqt_graph

パッケージのインストール

パッケージのインストール

- ROSの特徴の一つ
- “公開されているパッケージの導入が簡単”

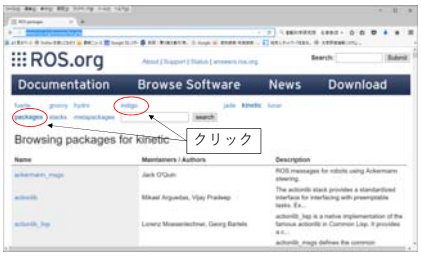
- 例えば
  - PR2を使いたいとき
    sudo apt-get install ros-indigo-pr2*
  - 北陽電機のレーザレンジファインダを使いたいとき
    sudo apt-get intall ros-indigo-urg-node

    あるいは
    cd ~/catkin_ws/src
    git clone https://github.com/ros-drivers/urg_node.git

PR2

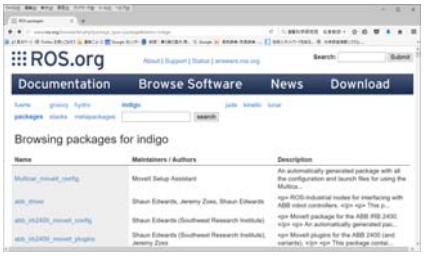
パッケージのインストール

- 公開パッケージを探す
  http://www.ros.org/browse/list.php



パッケージのインストール

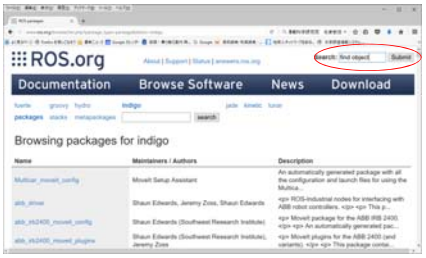
- 公開パッケージを探す
  http://www.ros.org/browse/list.php



Indigoバージョンで使えるパッケージが表示される

パッケージのインストール(find_object_2d)

- 例　Search： に find object と入れて，物体検出パッケージを探してみる



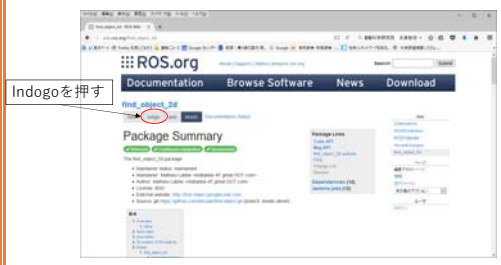
数回押す（結果が異なる）

パッケージのインストール(find_object_2d)

- 例　Search： に find object と入れて，物体検出パッケージを探してみる

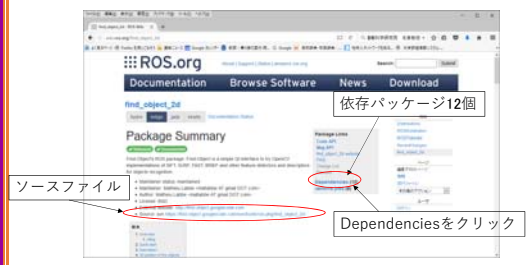
パッケージのインストール(find_object_2d)

- 例　Search： に find object と入れて，物体検出パッケージを探してみる



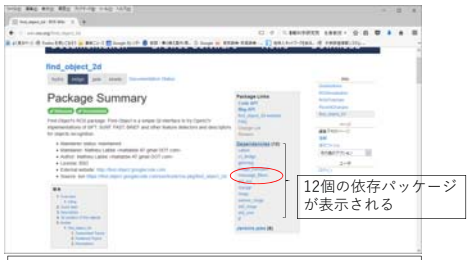
パッケージのインストール(find_object_2d)

- 例　Search： に find object と入れて，物体検出パッケージを探してみる



パッケージのインストール(find_object_2d)

- 例　Search： に find object と入れて，物体検出パッケージを探してみる



依存パッケージが全部インストールされている必要がある

パッケージのインストール(find_object_2d)

- パッケージの確認方法

・rospack listコマンドによる確認

```
$ rospack list
actionlib /opt/ros/indigo/share/actionlib
actionlib_msgs /opt/ros/indigo/share/actionlib_msgs
actionlib_tutorials /opt/ros/indigo/share/actionlib_tutorials
...
```

（出力されたリストに必要なパッケージが含まれているかを確認）

rospack findコマンドによる確認（インストールされている場合）

```
$ rospack find cv_bridge
/opt/ros/indigo/share/cv_bridge
```

rospack findコマンドによる確認（インストールされていない場合）

```
$ rospack find cv_bridge
[rospack] Error: package 'cv_bridge' not found
```


パッケージのインストール(find_object_2d)

- パッケージの確認方法

インストールされていない場合には，それぞれのWikiページでインストール方法を確認し，以下のようにインストールする。

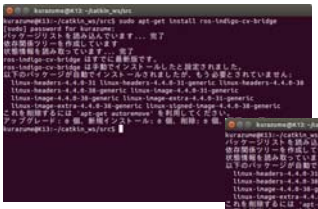
```
$ sudo apt-get install ros-indigo-cv-bridge
```

また，Wikiページ（http://wiki.ros.org/find_object_2d）の「2. Quick start」には，find_object_2dパッケージではuvc_cameraパッケージ（http://wiki.ros.org/uvc_camera）を利用することが記述されている。そこで，uvc_cameraパッケージをインストールする。

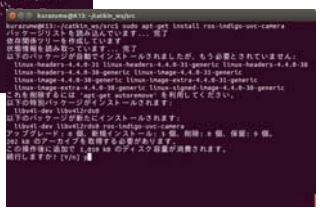
```
$ sudo apt-get install ros-indigo-uvc-camera
```


パッケージのインストール(find_object_2d)

ros-indigo-cv-bridge

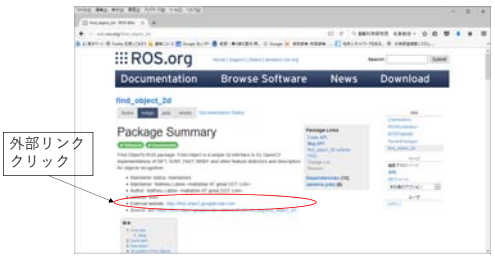


ros-indigo-uvc-camera



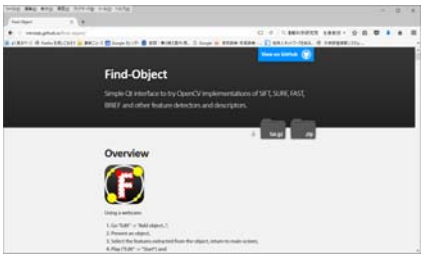
パッケージのインストール(find_object_2d)

find_object_2dをインストールする



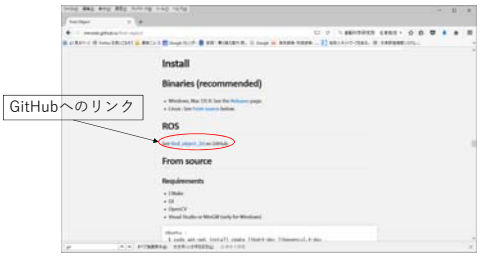
パッケージのインストール(find_object_2d)

find_object_2dをインストールする



パッケージのインストール(find_object_2d)

find_object_2dをインストールする

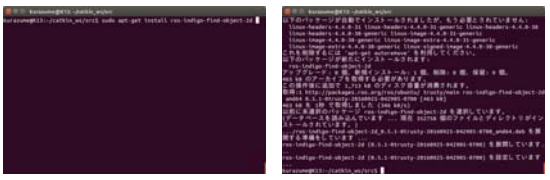


パッケージのインストール(find_object_2d)

find_object_2dをインストールする

パッケージのインストール(find_object_2d)

find_object_2dをインストールする

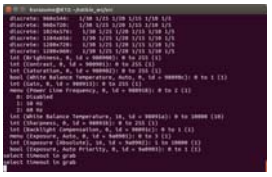


インストール終了！

パッケージのインストール(find_object_2d)

USBカメラを起動する

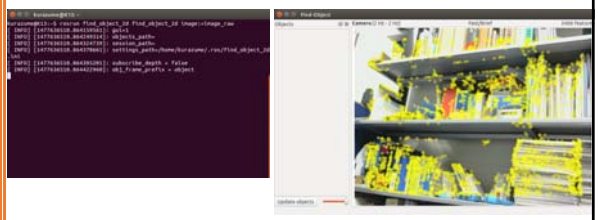
```
$ roscore
$ rosrun uvc_camera uvc_camera_node
```



パッケージのインストール(find_object_2d)

find_object_2dを実行する

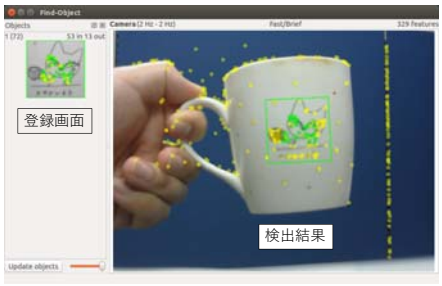
```
$ rosrun find_object_2d find_object_2d image:=image_raw
```



2016/11/22

パッケージのインストール(find_object_2d)

find_object_2dを実行する



2016/11/22

パッケージのインストール方法

- 公開パッケージの導入方法
  - apt-get を使う　（コンパイル済みファイルを導入）

    $ sudo apt-get install ros-indogo-XXXX

  - git clone を使う（ソースからコンパイルして導入）

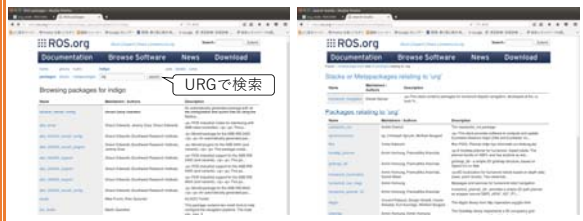
    $ cd ~/catkin_ws/src
    $ git clone XXX.git
    $ cd ~/catkin_ws
    $ catkin_make

  - その他（例えば svnとか）

2016/11/22

パッケージのインストール(urg_node)

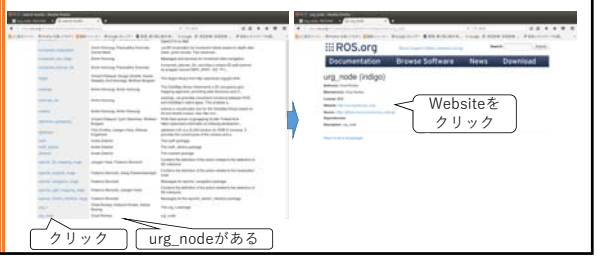
- 例　北陽電機のレーザレンジファインダ(URG)を導入
- 公開パッケージを探す
  http://www.ros.org/browse/list.php



2016/11/22

パッケージのインストール(urg_node)

- 例　北陽電機のレーザレンジファインダ(URG)を導入
- 公開パッケージを探す
  http://www.ros.org/browse/list.php



2016/11/22

パッケージのインストール(urg_node)

- 例　北陽電機のレーザレンジファインダ(URG)を導入

2016/11/22

パッケージのインストール(urg_node)

- 公開パッケージの導入方法①
  - apt-get を使う　（コンパイル済みファイルを導入）

$ sudo apt-get install ros-indogo-urg-node

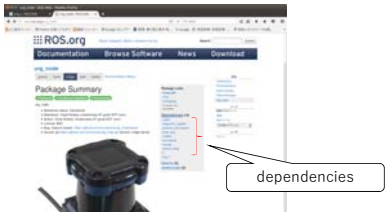
パッケージのインストール(urg_node)

- 公開パッケージの導入方法②
  - git clone　を使う（ソースからコンパイルして導入）

$ cd ~/catkin_ws/src
$ git clone https://github.com/ros-drivers/urg_node.git

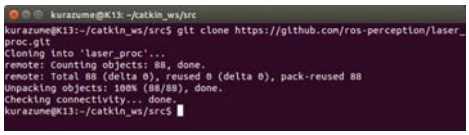
パッケージのインストール(urg_node)

- 公開パッケージの導入方法②
  - git clone　を使う（ソースからコンパイルして導入）
    dependencies を見ると，laser_proc, urg_cを入れる必要あり

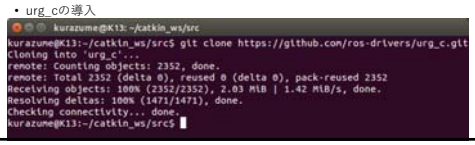


パッケージのインストール(urg_node)

- 公開パッケージの導入方法②
  - git clone　を使う（ソースからコンパイルして導入）
    - laser_procの導入

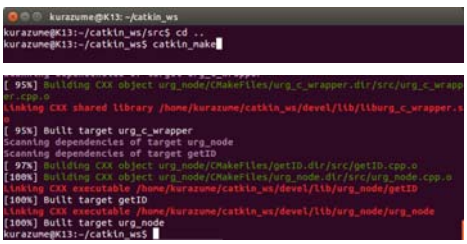


- urg_cの導入



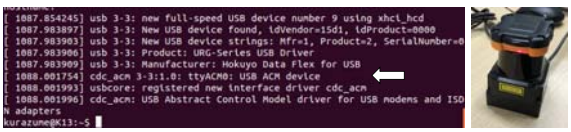
パッケージのインストール(urg_node)

- 公開パッケージの導入方法②
  - git clone　を使う（ソースからコンパイルして導入）
    $ cd ~/catkin_ws　（または cd ..）
    $ catkin_make



パッケージのインストール(urg_node)

- 実行例
  - TOPURG（UTM-30LX）をUSBに接続して接続ポートを確認
    $ dmesg



- /dev/ttyACM0　に接続されているので，パーミッションを設定
  $ sudo chmod a+rw /dev/ttyACM0

パッケージのインストール(urg_node)

- 実行例
  - roscoreを起動し，rosrunで実行
    $ roscore
    $ rosrun urg_node urg_node _serial_port:=/dev/ttyACM0

```
kurazume@K13:~
kurazume@K13:~$ sudo chmod a+rw /dev/ttyACM0
kurazume@K13:~$ rosrun urg_node urg_node _serial_port:=/dev/ttyACM0
[ INFO] [1479080790.037927525]: Connected to serial device with intensity and ID
: H0803605
[ INFO] [1479080790.287325896]: Streaming data.
```

  - トピックを確認
    $ rostopic list

```
kurazume@K13:~
kurazume@K13:~$ rostopic list
/diagnostics
/rosout
/rosout_agg
/scan
/urg_node/parameter_descriptions
/urg_node/parameter_updates
kurazume@K13:~$
```

urg_node関連

パッケージのインストール(urg_node)

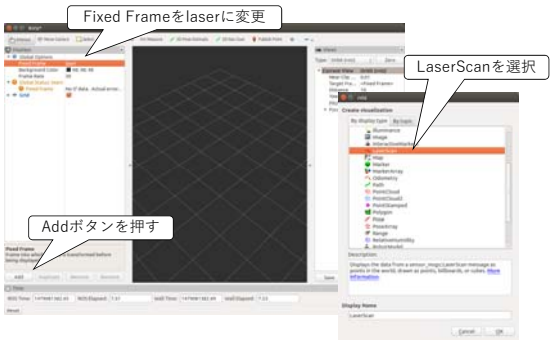
- 実行例
  - トピックの中身を見る
    $ rostopic echo /scan

```
kurazume@K13:~
9928474426, 0.13600008739997595, 0.13099999725818634, 0.1289999932050705, 0.1289
9999932050705, 0.12899999320507 05, 0.12700000405311584, 0.12600000202655792, 0.12
600000202655792, 0.12600000202655792, 0.12399999797344208, 0.12399999797344208,
0.12399999797344208, 0.12600000202655792, 0.12399999797344208, 0.12600000202655
92, 0.12700000405311584, 0.12700000405311584, 0.12700000405311584, 0.12700000405
311584, 0.12399999797344208, 0.11699999868869781, 0.11699999868869781, 0.1190000
0274181366, 0.11500000208616257, 0.11500000208616257, 0.11500000208616257, 0.112
```

  - rvizで確認
    $ rosrun rviz rviz （$ rviz のみでもいい）

```
kurazume@K13:~
kurazume@K13:~$ rviz
```


パッケージのインストール(urg_node)

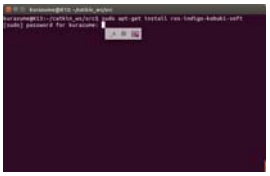


パッケージのインストール(urg_node)

Kobukiを用いたシミュレーション

Kobukiを用いたシミュレーション

- kibukiシミュレータをインストール
  $ sudo apt-get install ros-indigo-kobuki-soft

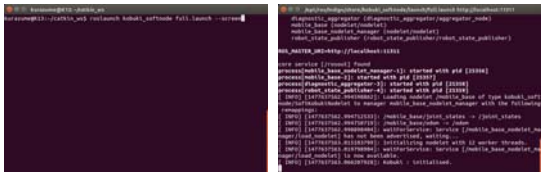


$ sudo apt-get install ros-indigo-kobuki-keyop

Kobukiを用いたシミュレーション

• kibukiシミュレータを実行

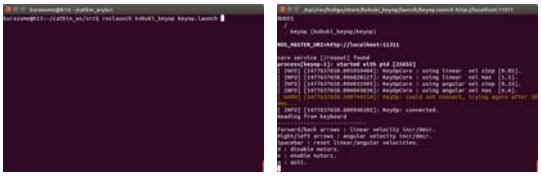
$ roslaunch kobuki_softnode full.launch --screen



Kobukiを用いたシミュレーション

• キーボード操作用ノードを実行

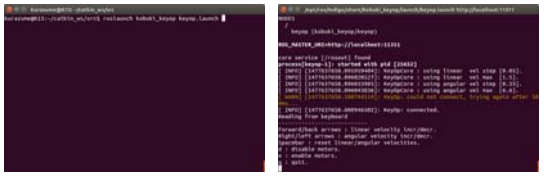
$ roslaunch kobuki_keyop keyop.launch



Kobukiを用いたシミュレーション

• キーボード操作用ノードを実行

$ roslaunch kobuki_keyop keyop.launch

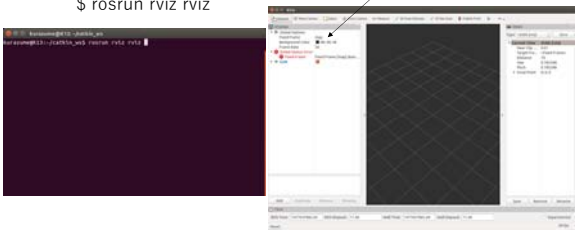


Kobukiを用いたシミュレーション

• 可視化ソフト rviz を起動

$ rosrun rviz rviz

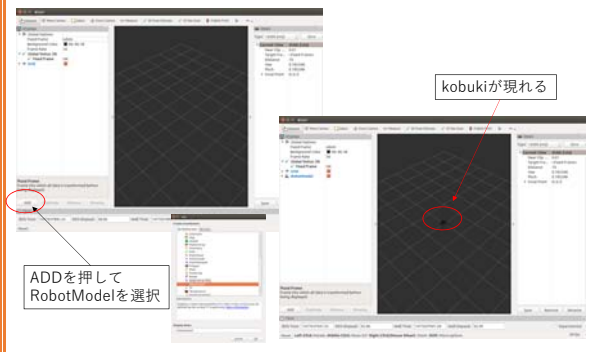
mapをodomに変更



Kobukiを用いたシミュレーション

kobukiが現れる

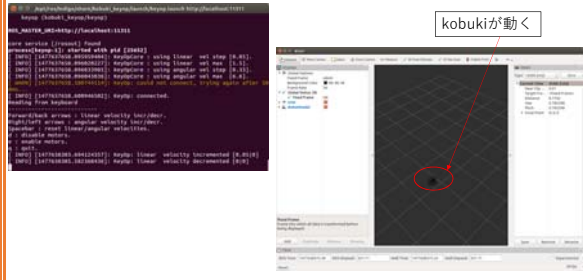
ADDを押して
RobotModelを選択



Kobukiを用いたシミュレーション

roslaunch kobuki_keyop keyop.launch
を実行しているターミナルで矢印キーを押す

kobukiが動く



Kobukiを用いたシミュレーション

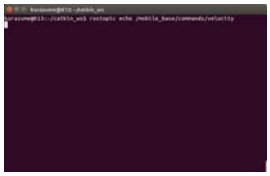
rostopic list でトピックを確認

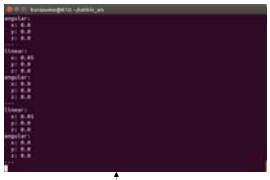
rqt_graphで接続を確認

/mobile_base/commands/velocity
トピックがある

Kobukiを用いたシミュレーション

/mobile_base/commands/velocityを出力

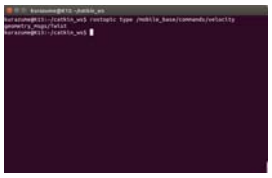




roslaunch kobuki_keyop keyop.launch
を実行しているターミナルで矢印キーを押す

Kobukiを用いたシミュレーション

rostopic type /mobile_base/commands/velocity で型がわかる



rosmsg show geometry_msgs/Twist で型の中身が見える

Kobukiを用いたシミュレーション

- 課題
  「/mobile_base/commands/velocityを出力する配信者を作りなさい」

パッケージの作成

- src ディレクトリ内に my_pkg パッケージを作成
  $ cd ~/catkin_ws/src
  $ catkin_create_pkg my_pkg geometry_msgs roscpp
  $ cd my_pkg
  $ gedit CMakeLists.txt

  … あとは自由に

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(my_pkg)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  geometry_msgs
  roscpp
)
...
## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )

catkin_package(
  INCLUDE_DIRS include
  LIBRARIES my_pkg
  CATKIN_DEPENDS geometry_msgs roscpp
  DEPENDS system_lib
)

...
```

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

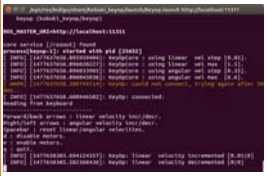
my_pkg_msg_publisher.cpp

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"

int main(int argc, char **argv)
{
	ros::init(argc, argv, "my_pkg_msg_publisher");
	ros::NodeHandle nh;

	ros::Publisher my_pkg_pub =
	nh.advertise<geometry_msgs::Twist>("/mobile_base/commands/velocity", 100);

	ros::Rate loop_rate(1);

	while (ros::ok())
	{
		geometry_msgs::Twist msg;

		msg.linear.x = 0.1;
		msg.linear.y = 0;
		msg.linear.z = 0;
		msg.angular.x = 0;
		msg.angular.y = 0;
		msg.angular.z = 0;

		ROS_INFO("send msg = %lf", msg.linear.x);

		my_pkg_pub.publish(msg);

		loop_rate.sleep();
	}
	return 0;
}
```

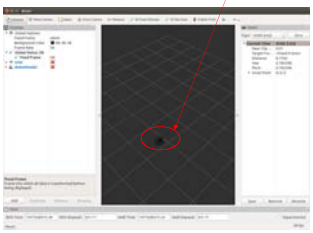
2016/11/22

Kobukiを用いたシミュレーション

roslaunch kobuki_keyop keyop.launch　を停止
$rosrun my_pkg my_pkg_msg_publisher



kobukiが動く



2016/11/22