体験！
小型ドローンのプログラム制御

初級コース

株式会社想画 秋山

2020/02/20版

INDEX

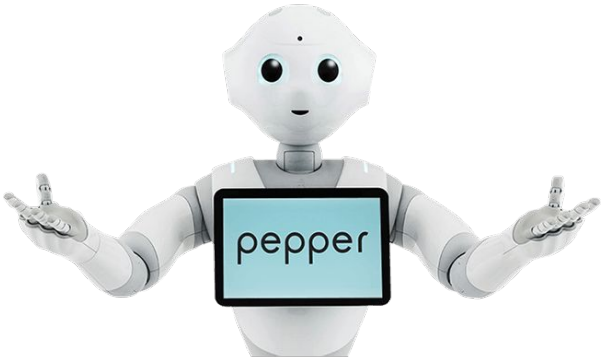
初級コース

ROSとは

Robot Operating System

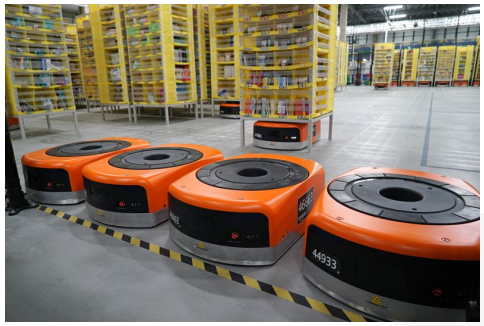
ROSとは

Robot Operating System



はじめに

- ROSは、既に多くの身近なロボットシステムで稼働しています。



Amazon Roboticsの物流補助ロボット
Sonyの家庭向けロボットaibo
パナソニックのトマト収穫用ロボット
iRobotのロボット掃除機Roomba
ソフトバンクの人型ロボットPepper
など…

ROSとは

Robot Operating System

開発ツールやライブラリが内包されたオープンソフトウェア

- **ROS普及以前**：メーカーや研究者が独自のロボットシステムを開発

  開発ノウハウが一般化しない

  技術的なブレイクスルーが生まれない

  開発コストの高騰

- **ROS普及以降**：過去に開発された技術やノウハウが公開され、

  誰もが利用できる

  ロボットシステムの開発や進歩速度が格段にUP

ROSとは

Robot Operating System

ロボットに欠かせない分散処理システム

- **集中管理型**：メインプロセッサがすべての情報処理と命令を行う

  システム構築が簡単

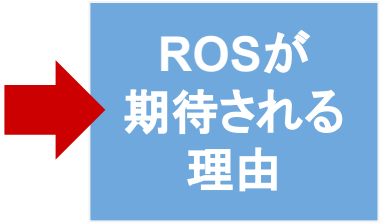
  一度に多くの情報を高速かつ連続で処理しなければならない場合、動作遅延やシステム停止を招く可能性がある

- **分散処理型**：複数のプロセッサが情報処理と命令を行う

  システム構築が集中管理型よりも難しい

  メンテナンスが手間

  安定したスピードや動作が見込める

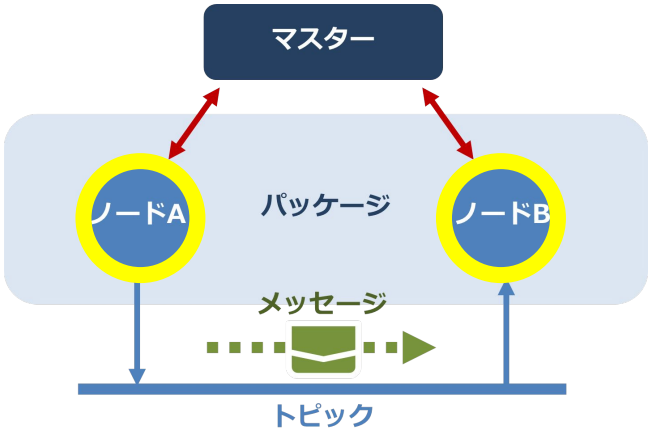


ROSとは

Robot Operating System

ROSの構成要素「ノード」

- **ノード**：一般的なコンピュータの**プログラム**に相当。ノード同士が情報のやり取りを行うことで、ロボットの制御を行う

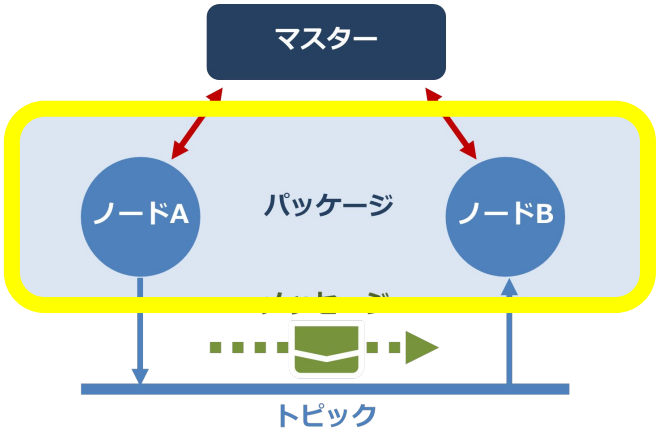


ROSとは

Robot Operating System

ROSの構成要素「パッケージ」

- **パッケージ**：関連する複数の**ノード**をまとめたもの

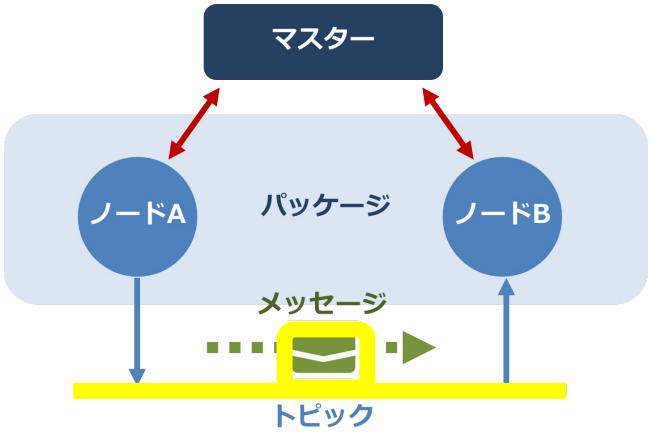


ROSとは

Robot Operating System

ROSの構成要素「トピックとメッセージ」

- **トピックとメッセージ**：ノード同士が情報をやり取りするための**回路の役割を担うのがトピック**。トピックを介して送られる**情報はメッセージ**と呼ばれる。ノードはトピックを通じてメッセージをやり取りすることで、プログラムを実行。

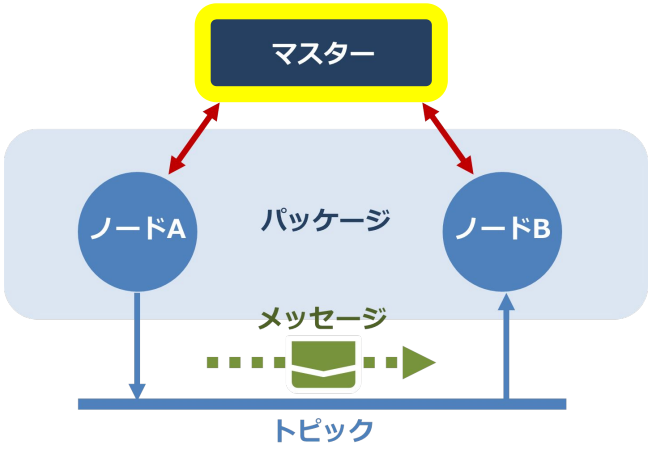


ROSとは

Robot Operating System

ROSの構成要素「マスター」

- **マスター**：ノード同士が情報のやり取りをする際に、お互いの**ノードやトピックを識別**し、それぞれの**ノードを管理する**役割を担う。

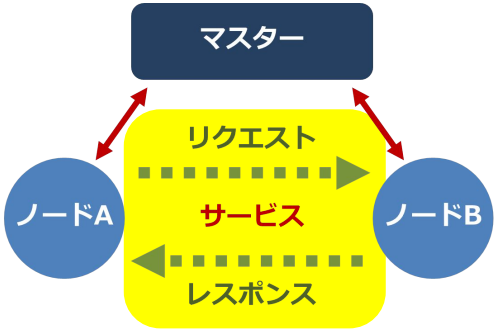


ROSとは

Robot Operating System

ROSの構成要素「サービス」

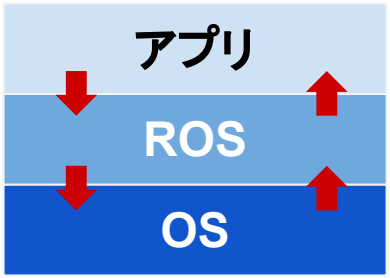
- **サービス**：ノード同士が情報をやり取りするためのもう一つの方法。ノードが**リクエスト**を送り、他のノードが処理をした後の**レスポンス**を受取るような通信であり、2つのノード間で1対1の**同期通信**を行うことができる。



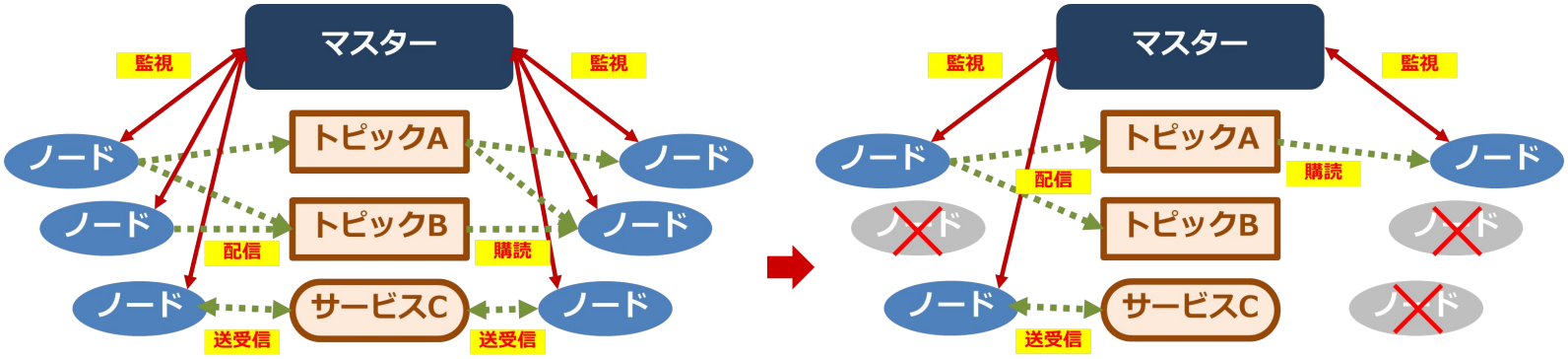
ROSとは

Robot Operating System

まとめ



- OSとアプリケーションの仲立ちを行うミドルウェアに近い
- 必ずノードを管理するマスターが必要（ただし、1つ）
- ノードはいくつでも立ち上げることができる
- ノード同士は非同期通信のトピックや同期通信のサービスを介して情報をやり取りできる
- 1つのノードは、同時に複数のトピックとの送信も受信もできる
- 各ノードは分離されたプログラムであり、1つがダウンしても他のノードへの影響がない



ROSとは

Robot Operating System

ROSで使えるライブラリや開発ツール

ROSには、既に開発済みの多くの**ロボット用ライブラリや開発ツール**が用意されており、全て無料でダウンロードして使用することができます。

- **開発ツール**:
  - 3D可視化ツール：Rviz
  - 3Dロボットシミュレータ：Gazebo
  - 開発用GUIフレームワーク：rqt
- **ライブラリ**:
  - 画像フォーマット変換：CvBridge
  - Python用ROSクライアントライブラリ：Rospy

ROSとは

Robot Operating System

ROS2について

ROS開発がスタートした2007年当時と現在では、ロボットを取り巻く環境は大きく変化しています。そこで、既に安定稼働している多くのROSシステムへの影響を考慮し、既存のROSと切り離した次世代バージョン「ROS 2」が開発されることになりました。

- **ROSとの主な違い**:
  - 通信ライブラリがData Distribution Service (DDS) に変更
    →rosmasterが必要なくなり、node同士で通信ができる
    →セキュア通信、Quality of Service (QoS) 通信、リアルタイム通信、ノード間の相互発見機能
  - サポートOSがUbuntu16.04(LTS)、Mac OS X Elcapitan、Windows 10に
  - LaunchファイルがXMLからPythonに変更
  - ビルド管理ツールがCatkin（CMake）からcolconに変更

※ROS 2 Design http://design.ros2.org/

※ROS 2 Overview https://index.ros.org/doc/ros2/

ROSとは

Robot Operating System

Bebop2の紹介

- 本セミナーで使用する小型ドローン**Bebop2**は、専用のパッケージ（**bebop_autonomy**）を使用することで、**ROS**による制御が可能になります。

Parrot Bebop 2

技術仕様（Parrot社HPより抜粋）

- 重量：500g
- サイズ：38 x 33 x 9 cm（プロペラ含）
- 飛行時間：2750mAhバッテリーにより、約25分
- ビデオ解像度：1920 x 1080p (30fps)

https://www.parrot.com/jp/doron/parrot-bebop-2

初級コース

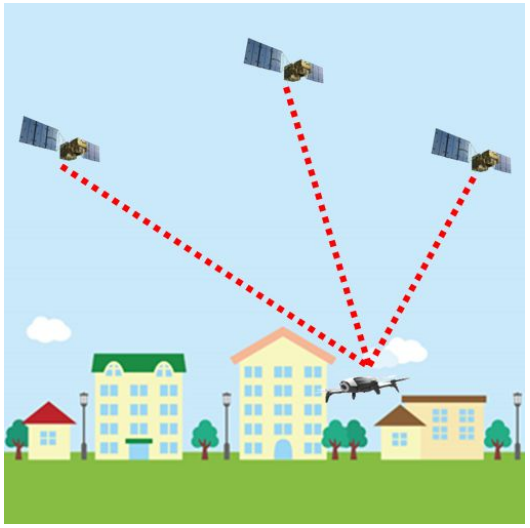
室内飛行の注意点

室内飛行の注意点

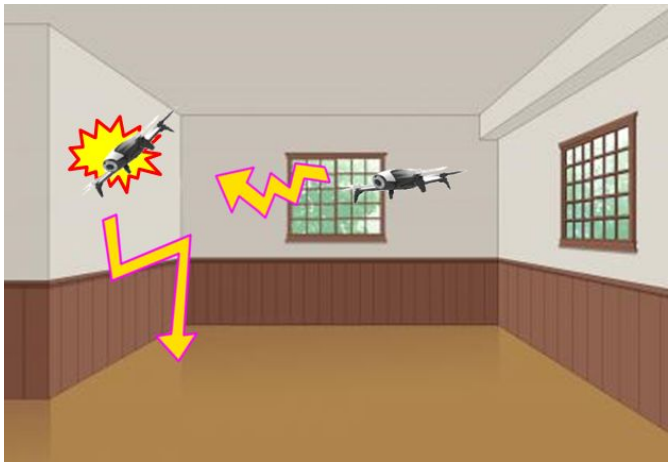
GPS環境下と非GPS環境下での飛行の違い

- GPSあり：自己位置推定が可能

  ⇒地図上に自分の位置を表示できる



- GPSなし：正確な自己位置推定が不可能



※現在地図上のどこに自分がいるのかわからないため、風に流されても元の場所に戻る事ができない。

室内飛行の注意点

ビジョンポジショニングシステム

ある程度、飛行の安定性を保つことが可能となるシステムで、一部の市販のドローンに搭載されています。部屋の明るさや床の模様などにより必ずしも万能な仕組みではありません。

- ドローン下部の超音波センサーで床の凹凸をチェック、常に一定高度を維持できる
- ドローン下部のカメラで床の模様の変をチェック、操縦による変化なのか、流されているのかを判断し位置を補正できる

空調の風が強くあたる場所での飛行は思わぬ異常飛行を始める原因になります。

非GPS環境下では、ドローンから目を離さず、常にコントローラーを操作できる状態を心がけてください。

室内飛行の注意点

室内特有の危険について

- **室内は決して無風ではない**

  ドローンはプロペラから発生する強力な風の力で飛行しています。そのため、自身の風の壁から吹き返し、室内空調の風などが影響し、飛行が不安定になる場合があります。

- **プロペラが止まる原因になるものが沢山**

  プロペラからの風が原因で、壁のポスターが突然めくれる、カーテンの裾がドローンを巻き込む、蛍光灯のスイッチ紐が巻き付くなど、室内にはプロペラの回転に影響を及ぼす多くの障害物が存在しています。

- **室内でのドローンは思った以上に高速で移動する**

  わずかな出力のアップ操作でも、急激にスピードが早まり、あっという間に距離を移動してしまいます。また空気との摩擦抵抗が小さいため、急に止まることはできません。壁や人との衝突には、屋外以上に注意が必要です。

初級コースの目的

初級コースの目的

なぜ、ROSを使う必要があるのか？

- 将来的に、ドローンの自律制御に挑戦したい時の課題

➔ ほとんどの付属アプリでは、GPSを利用した自律飛行しか用意されていない
➔ 非GPS環境下やGPS以外の自己位置推定技術を利用する場合は非常に高価

↓

自分で制御系のプログラムを開発する必要がある

- ROSを利用した制御システム開発のメリット

➔ 開発環境が無料で構築できる。（例：Ubuntu＋ROS＋Python＋OpenCV）
➔ ドローンや周辺機器との通信レベルの実装作業から開放される
➔ シュミレーションや可視化ツールが充実している

↓

自律制御系の開発だけに専念することができる

初級コースの目的

本日のセミナーでは行わないこと

- OS、ミドルウェアの環境構築手順の実践しません
  - ➔ 具体的なインストール手順は資料を使って説明します
  - ➔ 初級コースでは、参加者があらかじめROS（Kinetic）までインストール済みのUbuntuPCを用意していただくか、こちらで用意したBebop専用パッケージまでインストール済みのUbuntuPCを使い、「演習用パッケージの作成」から実施いたします。

- プログラム言語の学習は行いません
  - ➔ 演習で使用する言語はPythonですが、言語の使い方自体は学習しません
  - ➔ 初級コースでは既存のスクリプト（Python、XML）をベースに、パラメーターの書き換えや機能追加用のスクリプトの追加方法を学びます

- ドローンの自律飛行制御は行いません
  - ➔ 初級コースではキーボードを使った手動のリモート操作方法を学びます
  - ➔ 中級コースではドローンカメラを使って飛行指示を与える方法を学びます
  - ➔ 上級コースではARマーカーを自己位置推定に使った **自律飛行**を学びます

初級コースの目的

本日のセミナーでは何ができるようになるのか？

- パソコンのキーボードでドローンの操縦ができるようになります。
- デモンストレーション

➔ **テスト場への移動をお願いいたします**

初級コース

Ubuntuのインストール

Ubuntu16.04LTS

Ubuntuのインストール

Ubuntu16.04LTS

手順①

- Ubuntu 16.04 LTS 日本語 Remix版のダウンロード

手順②

- インストールディスクの作成（Windows10の場合）

手順③

- Ubuntuをインストール（Windows10とのデュアルブート）

手順④

- Ubuntuのアップデート

Ubuntuのインストール

Ubuntu16.04LTS

手順①

- Ubuntu 16.04 LTS 日本語 Remix版のダウンロード

以下のページにアクセスし、**ubuntu-ja-16.04-desktop-amd64.iso（ISOイメージ）**をダウンロードします。

https://www.ubuntulinux.jp/News/ubuntu1604-ja-remix

※現在、Ubuntu 16.04についてはisoイメージでの配布のみ。

Ubuntuのインストール

Ubuntu16.04LTS

手順②

- インストールディスクの作成（Windows10の場合）

書き込み用のDVDディスクを用意し、インストールディスクを作成します（ダウンロードしたISOファイルを右クリックし、メニューの二番上にある「ディスクイメージの書き込み」を選択）。

書き込みが完了後、作成したインストールディスクからPCを起動します。

※DVDから起動する方法は、お使いのPCのマニュアルを確認する。
※PCに光学ドライブが付属していない場合、専用のアプリを使ってisoイメージをUSBメモリーに書き込みインストールディスクの代わりにすることが可能（ブータブルUSBメモリの作成）。

ブータブルUSBメモリの作成方法
例）インストール用のUSBを作成できるソフトウェアとして「Rufus」を使う

※「Rufus」のダウンロードと使用方法は下記公式ページに記載されている。
https://rufus.ie/ja_JP.html

Ubuntuのインストール

Ubuntu16.04LTS

手順③

- Ubuntuをインストール（Windows10とのデュアルブート）

インストール作業については、詳しく解説されたページを参考に、慎重に作業を行ってください。

インストール用DVD（またはUSBメモリ）からPCを起動後、GRUBメニュー画面が表示されるので、「Install Ubuntu」を選択します。
各種オプションの選択画面が表示されたら、順に設定作業をすすめていきます。

Windowsを残してデュアルブートで使用する場合は、必ず、**インストールの種類**を選択する画面で、**＜それ以外＞**を選択して次へ進んでください。

パーティションを構成する際は、（引き続き**ROS**のインストールを考慮し）最低でも20GBのサイズを割り当ててください。それ以外は特別な条件はありません。

全設定が終わると、ファイルのコピーが始まるので終わるまでしばらく待ちます。

画面に「インストールは完了しました」のメッセージが表示されたら、DVDを入れたまま「今すぐ再起動する」ボタンをクリックします。ディスクが自動的に排出されたら「Enterキー」をクリックします。

※USBメモリからインストールを行った場合は、システムが一旦終了し、画面が暗くなるまで待ってからメモリを抜き「Enterキー」を押す。

Ubuntuのインストール

Ubuntu16.04LTS

手順④

- Ubuntuのアップデート

ホーム画面にターミナル（端末）アイコンがない場合はDashボタンをクリックし「ter」と入力。
「端末」アイコンが表示されるのでそれをクリックして起動してください。

Ubuntuへログイン後、デスクトップ画面からターミナルを起動します。
ターミナルで次の2つのコマンドを順番に入力し、Ubuntuを最新の状態に更新します。

```
$ sudo apt update
$ sudo apt upgrade
```

※途中でパスワードの入力が求められた場合、管理者用パスワードを入力しエンターキーを押す。

初級コース

Ubuntuの使い方

Ubuntuの使い方

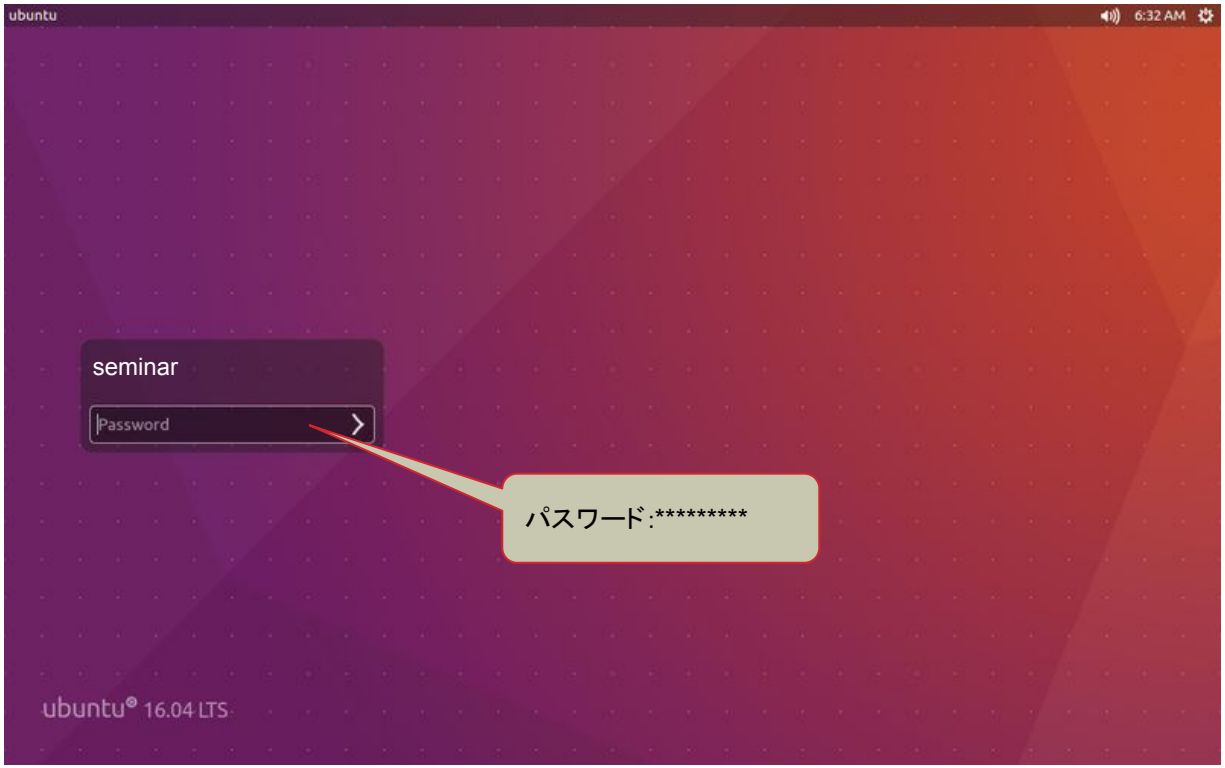
基本操作

- ログインについて
- ターミナルの起動
- コマンドの入力方法
- ファイルの編集方法
- 資料へのアクセス

Ubuntuの使い方

基本操作

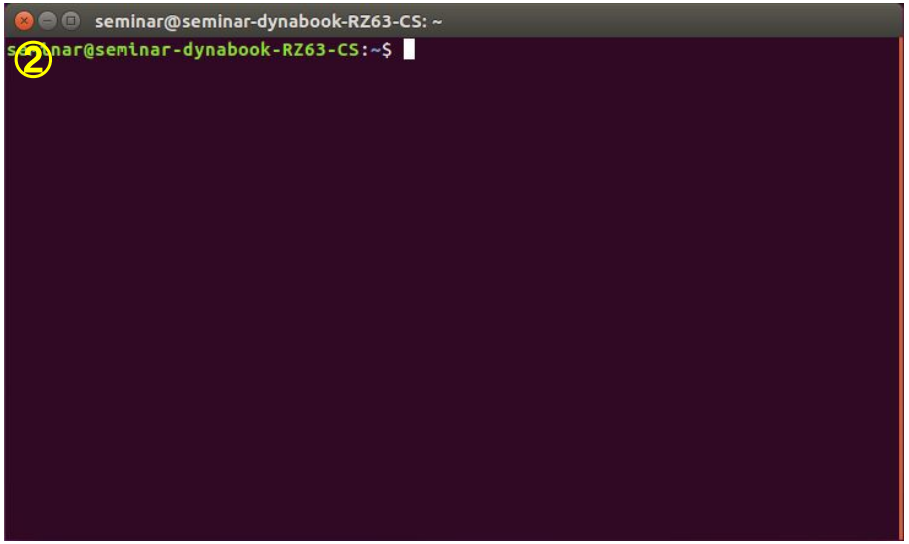
- ログインについて：パスワードを入力しEnterキーを押す



Ubuntuの使い方

基本操作

- ターミナル（端末）の起動：アイコン（①）をクリック

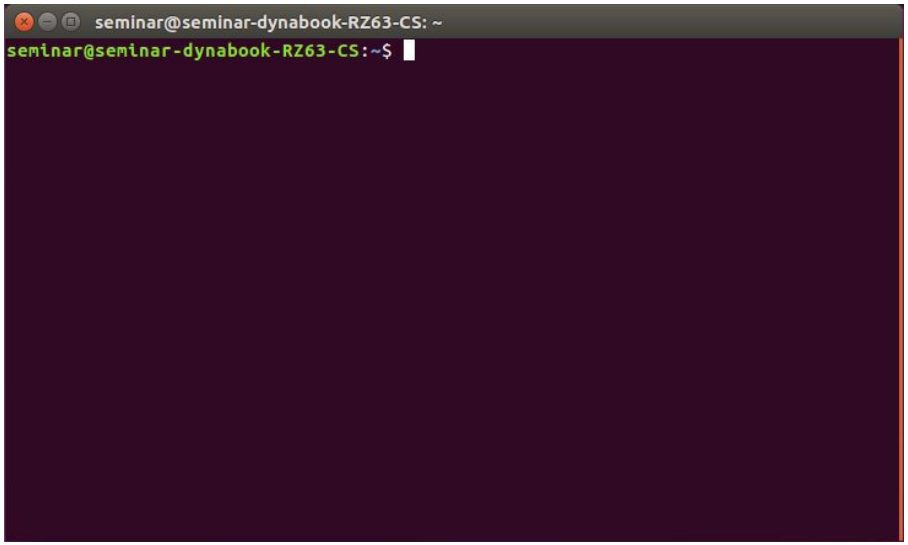


- ターミナル（端末）の終了：左上のクローズボタン（②）をクリックするか「exit」コマンドを実行する

Ubuntuの使い方

基本操作

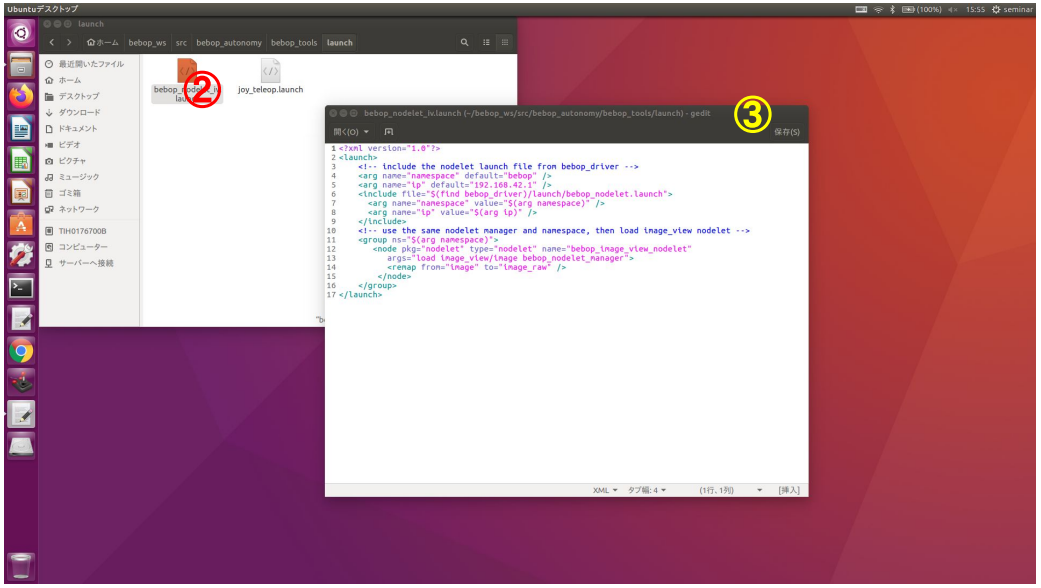
- コマンドの入力方法
  - 直接キーを入力するか、コピーしたテキストをCtrl＋Shift＋vでペーストすることができます。
  - 「半／全角」ボタンで、日本語入力のON・OFFができます。



Ubuntuの使い方

基本操作

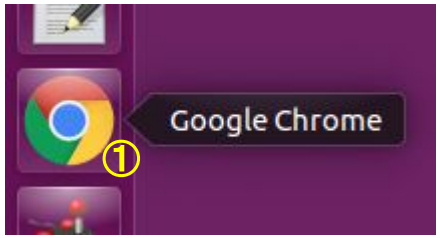
- ファイルの編集方法（テキストファイルの場合）
  - ファイルアイコン（①）をクリックし、目的のファイルを探していきます。
  - 目的のファイル（②）をダブルクリックすると、既定のエディター（gedit）が自動的に起動し編集画面が開きます。
  - 保存ボタン（③）をクリックすると、変更した内容が保存できます。



Ubuntuの使い方

基本操作

- 資料へのアクセス
  - Chrome（①）を起動し、ブックマークバーに登録済みの「ROSセミナー」（②）ボタンをクリックします。



初級コース

ROSのインストール

Ubuntu16.04LTS＋ROS Kinetic

ROSのインストール

Ubuntu16.04LTS＋ROS Kinetic

ROS Kinetic の Ubuntu へのインストール

- Gitのインストール
- ROSパッケージのダウンロード準備①
- ROSパッケージのダウンロード準備②
- ROSパッケージのインストール
- rosdepの初期化とROSの環境設定
- ROS起動テスト

演習用ワークスペースの作成

- catkinのインストール
- 作業ディレクトリの作成と設定

ROSのインストール

Ubuntu16.04LTS＋ROS Kinetic

ROS Kinetic の Ubuntu へのインストール

- Gitのインストール

次のコマンドを入力し、Git（圧縮ファイルの解凍プログラム）をインストールします。

```
$ sudo apt update
$ sudo apt install git
```

※途中でパスワードの入力が求められた場合、管理者用パスワードを入力しエンターキーを押す。

ROSのインストール

Ubuntu16.04LTS＋ROS Kinetic

ROS Kinetic の Ubuntu へのインストール

- ROSパッケージのダウンロード準備①

次のコマンドを実行し、packages.ros.orgからROSパッケージをダウンロードできるようにします。

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

※途中でパスワードの入力が求められた場合、管理者用パスワードを入力しエンターキーを押す。

ROSのインストール

Ubuntu16.04LTS＋ROS Kinetic

ROS Kinetic の Ubuntu へのインストール

- ROSパッケージのダウンロード準備②

次のコマンドを実行し、リポジトリを認証する鍵情報をローカルキーチェーンに取り込んでおきます。

```
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

※途中でWindowsセキュリティの警告が表示された場合は、「アクセスを許可する」をボタンを押す。

ROSのインストール

Ubuntu16.04LTS＋ROS Kinetic

ROS Kinetic の Ubuntu へのインストール

- ROSパッケージのインストール

次のコマンドを入力し、ROS（Kinetic）パッケージと各種ライブラリやツールをインストールします。

```
$ sudo apt update
$ sudo apt install ros-kinetic-desktop-full
```

※ディスクの空き容量が少なく、容量不足のエラーが出てしまう場合、以下のコマンドを使用GUIツールが含まれていない最小構成のバージョンをインストールする。
→ $ sudo apt-get install ros-kinetic-ros-base

ROSのインストール

Ubuntu16.04LTS＋ROS Kinetic

ROS Kinetic の Ubuntu へのインストール

- rosdepの初期化とROSの環境設定

ROSの依存関係をチェックした後、ROSを使いやすくするための設定を行います。

```
$ sudo rosdep init
$ rosdep update
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

※途中で、[Y/n]（Yes/No）の入力が求められた場合、Y（またはｙ）を入力しエンターキーを押す。
※rosdep は、ROSにおけるシステムの依存関係を管理し、必要な依存物をダウンロードしてインストールするためのツール。

ROSのインストール

Ubuntu16.04LTS＋ROS Kinetic

ROS Kinetic の Ubuntu へのインストール

- ROS起動テスト

新規にターミナルを起動し、以下のコマンドを実行します。

```
$ roscore
```

※エラーが表示されなければ、ROSのインストールは完了。
※終了する場合は、ターミナルで**Ctrl+C**を実行しスクリプトを停止させた後、**exit**コマンドでターミナルを閉じる。

ROSのインストール

Ubuntu16.04LTS＋ROS Kinetic

演習用ワークスペースの作成

- catkinのインストール

次のコマンドを入力し、catkin（ROSパッケージビルドツール）をインストールします。

```
$ sudo apt install python-catkin-tools
```

※途中でパスワードの入力が求められた場合、管理者用パスワードを入力しエンターキーを押す。
※途中で、[Y/n]（Yes/No）の入力が求められた場合、Y（またはｙ）を入力しエンターキーを押す。

ROSのインストール

Ubuntu16.04LTS＋ROS Kinetic

演習用ワークスペースの作成

- 作業ディレクトリの作成と設定

次のコマンドを入力し、作業ディレクトリ（本セミナーでの演習用ワークスペース）を作成します。
（ワークスペース名：**bebop_ws**）

```
$ mkdir -p ~/bebop_ws/src && cd ~/bebop_ws
$ catkin build
$ source devel/setup.bash
$ echo "source ~/bebop_ws/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

※途中で、[Y/n]（Yes/No）の入力が求められた場合、Y（またはｙ）を入力しエンターキーを押す。

初級コース

Bebop2専用パッケージのインストール

Bebop2専用パッケージのインストール

bebop_autonomy のインストール

- bebop_autonomyのダウンロード
- bebop_autonomyのビルド

Bebop2専用パッケージのインストール

bebop_autonomy のインストール

- bebop_autonomyのダウンロード

bebop_autonomyと遠隔操作用のソースファイルを作業ディレクトリ内のsrcフォルダー に配置します。

```
$ git clone https://github.com/AutonomyLab/bebop_autonomy.git src/bebop_autonomy
$ git clone https://github.com/ros-teleop/teleop_tools.git src/bebop_autonomy
```

※途中で、[Y/n]（Yes/No）の入力が求められた場合、Y（またはｙ）を入力しエンターキーを押す。

Bebop2専用パッケージのインストール

bebop_autonomy のインストール

- bebop_autonomyのビルド

まず、依存関係を確認してからインストールし、最後にドライバをビルドします。

```
$ rosdep update
$ rosdep install --from-paths src -i
$ catkin build
```

※途中で、[Y/n]（Yes/No）の入力が求められた場合、Y（またはｙ）を入力しエンターキーを押す。
※.bashrcファイルの中に、以下の2行が追記されているかを確認しておく。

- source /opt/ros/kinetic/setup.bash
- source ~/bebop_ws/devel/setup.bash

初級コース

演習1.
ROSコマンドを体験する

演習1.
ROSコマンドを体験する

代表的なROSコマンドの種類と使い方

- roscore
- rosrun
- roslaunch
- rqt_graph

その他のコマンド

- rosgraph
- rqt_graph
- rostopic

演習1.
ROSコマンドを体験する

代表的なROSコマンドの種類と使い方

- roscoreについて

ROSノードが通信を行うために必要な機能の実行を開始します。

roscoreはMaster、Parameter Server、rosoutのログ用ノードを起動します

```
... logging to ~/.ros/log/9cf88ce4-b14d-11df-8a75-00251148e8cf/roslaunch-machine_name-13039.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://machine_name:33919/
ros_comm version 1.4.7

SUMMARY
========

PARAMETERS
 * /rosversion
 * /rosdistro

NODES

auto-starting new master
process[master]: started with pid [13054]
ROS_MASTER_URI=http://machine_name:11311/

setting /run_id to 9cf88ce4-b14d-11df-8a75-00251148e8cf
process[rosout-1]: started with pid [13067]
started core service [/rosout]
```

演習1.
ROSコマンドを体験する

代表的なROSコマンドの種類と使い方

- roscoreを実行する

新規にターミナルを起動し、以下のコマンドを実行します。

```
$ roscore
```

※次のコマンドの実行で必要となるので、roscoreは終了しない。

演習1.
ROSコマンドを体験する

代表的なROSコマンドの種類と使い方

- rosrunについて

使用方法:rosrun <パッケージ名> <ノード名> <オプション>

最初にフルパスまたはcd / roscdを指定することなく、任意の実行ファイルを起動したり、ノードを立ち上げることができます。また、変数に値を設定したり、ノード名を置き換えたり、参照するトピック名を変更することができます。

基本形
$ rosrun package node

パラメーターの渡す場合
$ rosrun package node _parameter:=value

トピック名を変更する場合(例: image_view)
$ rosrun image_view image_view image:=/usb_cam/image_raw

※最初にroscoreを起動しておく必要がある。

演習1.
ROSコマンドを体験する

代表的なROSコマンドの種類と使い方

- rosrunを実行する（トピックの配信）

roscoreが別のターミナルで起動していることを確認してください。

2つ目のターミナルを起動し、以下のコマンドを順番に実行します。

USBカメラ用パッケージのインストール（未インストールの場合のみ）
```
$ sudo apt install ros-kinetic-usb-cam
```

ノード（usb_cam_node）を起動し、トピックの配信を開始
```
$ rosrun usb_cam usb_cam_node
```

※途中でパスワードの入力が求められた場合、管理者用パスワードを入力しエンターキーを押す。
※途中で、[Y/n]（Yes/No）の入力が求められた場合、Y（またはｙ）を入力しエンターキーを押す。

演習1.
ROSコマンドを体験する

代表的なROSコマンドの種類と使い方

- rosrunを実行する（トピックの受信）

roscoreが別のターミナルで起動していることを確認してください。

3つ目のターミナルを起動し、以下のコマンドを順番に実行します。

```
image_viewノードを起動（usb_cam_nodeのトピックを受信）
$ rosrun image_view image_view image:=/usb_cam/image_raw
```

※終了する場合は、ターミナルで**Ctrl+C**を実行しスクリプトを停止させた後、**exit**コマンドでターミナルを閉じる。（それぞれのターミナルで実行）

演習1.
ROSコマンドを体験する

代表的なROSコマンドの種類と使い方

- roslaunchについて

使用方法 :roskaunch <パッケージ名> <XMLファイル名>

Launchファイルは記述形式にXMLを使用しています。複数のrosrunを同時に実行し、それぞれに引数の設定やリマップ操作などを指定することができます。

```
<?xml version="1.0"?>
<launch>
    <arg name="namespace" default="bebop" />
    <arg name="ip" default="192.168.42.1" />
    <arg name="drone_type" default="bebop1" /> <!-- available drone types: bebop1, bebop2 -->
    <arg name="config_file" default="$(find bebop_driver)/config/defaults.yaml" />
    <arg name="camera_info_url" default="package://bebop_driver/data/$(arg drone_type)_camera_calib.yaml" />
    <group ns="$(arg namespace)">
        <node pkg="bebop_driver" name="bebop_driver" type="bebop_driver_node" output="screen">
            <param name="camera_info_url" value="$(arg camera_info_url)" />
            <param name="bebop_ip" value="$(arg ip)" />
            <rosparam command="load" file="$(arg config_file)" />
        </node>
        <include file="$(find bebop_description)/launch/description.launch" />
    </group>
</launch>
```

※roslaunchはroscoreを自動で起動するため、事前にroscoreを起動させておく必要はない。

演習1.
ROSコマンドを体験する

代表的なROSコマンドの種類と使い方

- roslaunchを実行する（Bebop2と通信を開始し、トピックを配信）

roslaunchはroscoreを自動で起動するため、事前にroscoreを起動させておく必要はありません

Bebop2後方のPowerスイッチを押し、ライトの点滅が完了するまで待機します。
ライトが点灯したらWiFiのアクセスポイントを＜Bebop2-******（6桁の数値）＞に切り替え、新規にターミナルを起動し、以下のコマンドを実行します。

```
bebop_autonomy（Bebop2との通信用パッケージ）を起動
$ roslaunch bebop_driver bebop_node.launch
```

※次のコマンドの実行で必要となるので、roslaunchは終了しない。

演習1.
ROSコマンドを体験する

代表的なROSコマンドの種類と使い方

- rosrunを実行する（トピックの受信）

roslaunchが別のターミナルで起動していることを確認してください。

2つ目のターミナルを起動し、以下のコマンドを実行します。

```
image_viewノードを起動（bebop_nodeのトピックを受信）
$ rosrun image_view image_view image:=/bebop/image_raw
```

※終了する場合は、ターミナルで**Ctrl+C**を実行しスクリプトを停止させた後、**exit**コマンドでターミナルを閉じる。（それぞれのターミナルで実行）

演習1.
ROSコマンドを体験する

その他のコマンド

- rosgraph

現在動作しているノードの情報を一挙に表示するためのコマンドです。

3つ目のターミナルを起動し、以下のコマンドを実行します。

```
$ rosgraph
```

※終了する場合は、ターミナルで**Ctrl+C**を実行しスクリプトを停止させた後、**exit**コマンドでターミナルを閉じる。（それぞれのターミナルで実行）

演習1.
ROSコマンドを体験する

その他のコマンド

- rqt_graph

rqt_graph はrqt_tools パッケージの一部で、システムの状況を可視化するツールです。

4つ目のターミナルを起動し、以下のコマンドを順番に実行します。

rqt_graphのインストール（未インストールの場合のみ）
```
$ sudo apt-get install ros-kinetic-rqt
$ sudo apt-get install ros-kinetic-rqt-common-plugins
```

rqt_graphの起動
```
$ rqt_graph
```

※終了する場合は、ターミナルで**Ctrl+C**を実行しスクリプトを停止させた後、**exit**コマンドでターミナルを閉じる。（それぞれのターミナルで実行）

演習1.
ROSコマンドを体験する

その他のコマンド

- rostopic

アクティブなトピックの情報を表示するツールです。

5つ目のターミナルを起動し、以下のコマンドを順番に実行します。

```
アクティブなトピックの一覧を表示
$ rostopic list

指定したトピックのメッセージの型を表示
$ rostopic type /bebop/image_raw

指定したトピックのメッセージの内容を表示
$ rostopic echo /bebop/image_raw
```

※「rostopic echo」コマンドを終了する場合は、ターミナルで**Ctrl+C**を実行する

初級コース

演習2.
既存パッケージを利用する

演習2.
既存パッケージを利用する

Bebop2をノンプログラミングで飛行制御する

- bebop_autonomyを起動する
- teleop_twist_keyboardを起動する
- 離陸用トピックを送信する
- 着陸用トピックを送信する
- 資料：キーボードの使い方

演習2.
既存パッケージを試す

Bebop2をノンプログラミングで飛行制御する

- bebop_autonomyを起動する

roslaunchはroscoreを自動で起動するため、事前にroscoreを起動させておく必要はありません

Bebop2後方のPowerスイッチを押し、ライトの点滅が完了するまで待機します。
ライトが点灯したらWiFiのアクセスポイントを＜Bebop2-******（6桁の数値）＞に切り替え、新規にターミナルを起動し、以下のコマンドを実行します。

```
bebop_autonomy（Bebop2との通信用パッケージ）を起動
$ roslaunch bebop_driver bebop_node.launch
```

※次のコマンドの実行で必要となるので、roslaunchは終了しない。

演習2.
既存パッケージを試す

Bebop2をノンプログラミングで飛行制御する

- teleop_twist_keyboardを起動する

teleop_twist_keyboardはキーボードの遠隔操作用パッケージです。

2つ目のターミナルを起動し、以下のコマンドを順番に実行します。

twist-keyboardのインストール（未インストールの場合のみ）
```
$ sudo apt install ros-kinetic-teleop-twist-keyboard
```

teleop_twist_keyboardノードを起動（bebop_nodeのトピックを受信）
```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
cmd_vel:=/bebop/cmd_vel
```

※途中でパスワードの入力が求められた場合、管理者用パスワードを入力しエンターキーを押す。
※途中で、[Y/n]（Yes/No）の入力が求められた場合、Y（またはｙ）を入力しエンターキーを押す。

演習2.
既存パッケージを試す

Bebop2をノンプログラミングで飛行制御する

- 離陸用トピックを送信する

3つ目のターミナルを起動し、以下のコマンドを実行します。

離陸用トピック
```
$ rostopic pub --once /bebop/takeoff std_msgs/Empty
```

※終了する場合は、ターミナルで**Ctrl+C**を実行しスクリプトを停止させた後、**exit**コマンドでターミナルを閉じる。（それぞれのターミナルで実行）
※1つ目のターミナルでroslaunchを使用しているため、roscoreは自動的に起動する。

演習2.
既存パッケージを試す

Bebop2をノンプログラミングで飛行制御する

- 着陸用トピックを送信する

4つ目のターミナルを起動し、以下のコマンドを実行します。

着陸用トピック
```
$ rostopic pub --once /bebop/land std_msgs/Empty
```

参考：救急停止用トピック（通常使用しない）
```
$ rostopic pub --once /bebop/reset std_msgs/Empty
```

※終了する場合は、ターミナルで**Ctrl+C**を実行しスクリプトを停止させた後、**exit**コマンドでターミナルを閉じる。（それぞれのターミナルで実行）
※1つ目のターミナルでoslaunchを使用しているため、roscoreは自動的に起動する。

演習2.
既存パッケージを試す

Bebop2をノンプログラミングで飛行制御する

- 資料：*キーボードの使い方*

（移動操作）

u：左方前進	**i**：前進	**o**：右方前進
j：左回転	**k**：停止	**l**：右回転
m：左方後退	**,**：後退	**.**：右方後退

（速度調整）

- 全体スピードの調整（**q**:+10%、**z**:-10%）
- 移動スピードの調整（**w**:+10%、**x**:-10%）
- 回転スピードの調整（**e**:+10%、**c**:-10%）

その他のキーで停止
Ctrl + Cでプログラムが終了

初級コース

演習3.
飛行プログラムを作る

演習3.
飛行プログラムを作る

Bebop2を自作プログムで飛行制御する

- 演習プログラム用パッケージを作成する
- 編集用Launchファイルを準備する
- 新規に作成するプログラムの起動スクリプトを追加する
- 編集用Pythonファイルを準備する
- 離着陸用トピックの利用機能を追加する
- Spaceキーで離着陸用トピックが交互に実行されるようにする
- キーボードコントローラーの起動
- 資料：*キーボードの使い方*
    1. 基本操作（離着陸、速度調整など）
    2. 移動操作（前後進、方向転換など）

演習3.
飛行プログラムを作る

Bebop2を自作プログムで飛行制御する

- 演習プログラム用パッケージの作成

独自に作成した機能を既存のパッケージと分けて管理すること可能になります。

次のコマンドを実行し、新規のパッケージを作成します。
（演習用パッケージ名：seminar）
最後に、rospackコマンドを使用し正しくビルドされているかの確認を行います。

```
$ cd ~/bebop_ws/src
$ catkin_create_pkg seminar roscpp rospy std_msgs
$ cd ~/bebop_ws/
$ catkin build
$ source devel/setup.bash
$ rospack find seminar
```

※正しくビルドされているとseminarパッケージのPATHが表示されます。
例）/home/seminar/bebop_ws/src/seminar
※エラーが出た場合は、手順に問題がないか見直してください。
例）Error: package ‘seminar’ not found

演習3.
飛行プログラムを作る

Bebop2を自作プログムで飛行制御する

- 新規実行ファイル（ros_example_1_3.py）の準備

公開されているキーボード操作用のプログラムをコピーし、それをベースに加工します。

以下のコマンドを実行し、編集作業を開始します。
（ベースプログラム名：teleop_twist_keyboard.py）
（新規プログラム名：：ros_example_1_3.py）

```
$ cd ~/bebop_ws/src
$ git clone https://github.com/ros-teleop/teleop_twist_keyboard.git seminar/src
$ mkdir -p ~/bebop_ws/src/seminar/scripts
$ cp ~/bebop_ws/src/seminar/src/teleop_twist_keyboard.py seminar/scripts/ros_example_1_3.py
$ gedit ~/bebop_ws/src/seminar/scripts/ros_example_1_3.py
```

※cpコマンド以降の操作は、デスクトップのファイルアイコンをクリックし、同様の操作をGUIを使って行うことも可能。

演習3.
飛行プログラムを作る

Bebop2を自作プログムで飛行制御する

- 新規ローンチファイル（ros_example_1_3.launch）の準備

既存のLaunchファイルをコピーし、それをベースに加工します。

以下のコマンドを実行し、編集作業を開始します。
（ベースファイル名：bebop_node.launch）
（新規ファイル名：ros_example_1_3.launch）

```
$ cd ~/bebop_ws/src
$ mkdir -p ~/bebop_ws/src/seminar/launch
$ cp bebop_autonomy/bebop_driver/launch/bebop_node.launch
seminar/launch/ros_example_1_3.launch
$ gedit ~/bebop_ws/src/seminar/launch/ros_example_1_3.launch
```

※デスクトップのファイルアイコンをクリックし、同様の操作をGUIを使って行うことも可能。

演習3.
飛行プログラムを作る

Bebop2を自作プログムで飛行制御する

- 新規ローンチファイル（ros_example_1_3.launch）を編集

実行プログラム「ros_example_1_3.py」が「bebop_autonomy」と同時に起動されるようにします。

16行目に以降に、赤文字部分のスクリプトを追記し、上書き保存をします。
（新規プログラム名：：ros_example_1_3.py）

```
（中略）
15:             </group>
16:             <node pkg="seminar" name="drone_controller"
type="ros_example_1_3.py">
17:             </node>
18:     </launch>
```

※エディターの「保存」ボタンをクリックしてから、編集画面を閉じる。

演習3.
飛行プログラムを作る

Bebop2を自作プログムで飛行制御する

- 新規実行ファイル（ros_example_1_3.py）を編集①

Empty型のメッセージを送信できるようにします。（離陸・着陸用トピックで使用）

9行目に、赤文字部分のスクリプトを追記します。

```
（中略）
8:      from geometry_msgs.msg import Twist
9:      from std_msgs.msg import Empty
（中略）
```

演習3.
飛行プログラムを作る

Bebop2を自作プログムで飛行制御する

- 新規実行ファイル（ros_example_1_3.py）を編集②

飛行状態を管理するため、変数「flying_mode」を定義します。初期値はFalse（非飛行状態）です。

69行目に、赤文字部分のスクリプトを追記します。

```
（中略）
67:       }
68:
69:     flying_mode = False
70:
71:     def getKey():
（中略）
```

演習3.
飛行プログラムを作る

Bebop2を自作プログムで飛行制御する

- 新規実行ファイル（ros_example_1_3.py）を編集③

Twist型メッセージ（速度値）がトピック『bebop/cmd_vel』宛に配信されるよう変更します。

85行目の、赤文字部分のスクリプトを追記します。

```
（中略）
85:             pub = rospy.Publisher('/bebop/cmd_vel', Twist, queue_size = 1)
（中略）
```

演習3.
飛行プログラムを作る

Bebop2を自作プログムで飛行制御する

- 新規実行ファイル（ros_example_1_3.py）を編集④

Empty型メッセージが離着陸トピック『bebop/takeoff」「/bebop/land」宛に配信できるようにします。

86行目以降に、赤文字部分のスクリプトを追記します。

```
（中略）
86:             pubTakeoff = rospy.Publisher('/bebop/takeoff', Empty,
queue_size=10)
87:             pubLand = rospy.Publisher('/bebop/land', Empty,
queue_size=10)
88:
（中略）
```

演習3.
飛行プログラムを作る

Bebop2を自作プログムで飛行制御する

- 新規実行ファイル（ros_example_1_3.py）を編集⑤

Spaceキーで離着陸用トピックが交互に実行されるようにします。

104行目以降に、赤文字部分のスクリプトを追記します。

```
104:                if key == ' ':
105:                    if flying_mode:
106:                        flying_mode = False
107:                        pubLand.publish(Empty())
108:                    else:
109:                        flying_mode = True
110:                        pubTakeoff.publish(Empty())
111:                elif key in moveBindings.keys():
（中略）
```

※エディターの「保存」ボタンをクリックしてから、編集画面を閉じる。

roslaunch seminar ros_example_1_3.launch

演習3.
Pythonプログラムの作成

Bebop2を自作プログムで飛行制御する

- 自作プログラムの実行

Bebop2後方のPowerスイッチを押し、ライトの点滅が完了するまで待機します。
ライトが点灯したらWiFiのアクセスポイントを＜Bebop2-******（6桁の数値）＞に切り替え、
ターミナルを起動し、以下のコマンドを実行します。

```
$ roslaunch seminar ros_example_1_3.launch
```

※終了する場合は、ターミナルで**Ctrl+C**を実行しスクリプトを停止させた後、**exit**コマンドでターミナルを閉じる。

演習3.
Pythonプログラムの作成

Bebop2を自作プログムで飛行制御する

- 資料：キーボードの使い方　～1. 離着陸とスピード変更操作

（離着陸）
- スペースキーを押すと、「離陸」と「着陸」が交互に動作

離陸→着陸→離陸→着陸→（以降繰り返し）

（速度調整）
- 全体スピードの調整（**q**：+10%、**z**：-10%）
- 移動スピードの調整（**w**：+10%、**x**：-10%）
- 回転スピードの調整（**e**：+10%、**c**：-10%）

（その他）
- **Ctrl + C**でプログラムが終了

演習3.
Pythonプログラムの作成

Bebop2を自作プログムで飛行制御する

- 資料:キーボードの使い方　～2. 移動操作

（移動操作）

u：左方前進	**i**：前進	**o**：右方前進
j：左回転	**k**：停止	**l**：右回転
m：左方後退	**,**：後退	**.**：右方後退

U：左斜前進	**I**：前進	**O**：右斜前進
J：左移動	**K**：停止	**L**: 右移動
M：左斜後退	**<**：後退	**>**：右斜後退

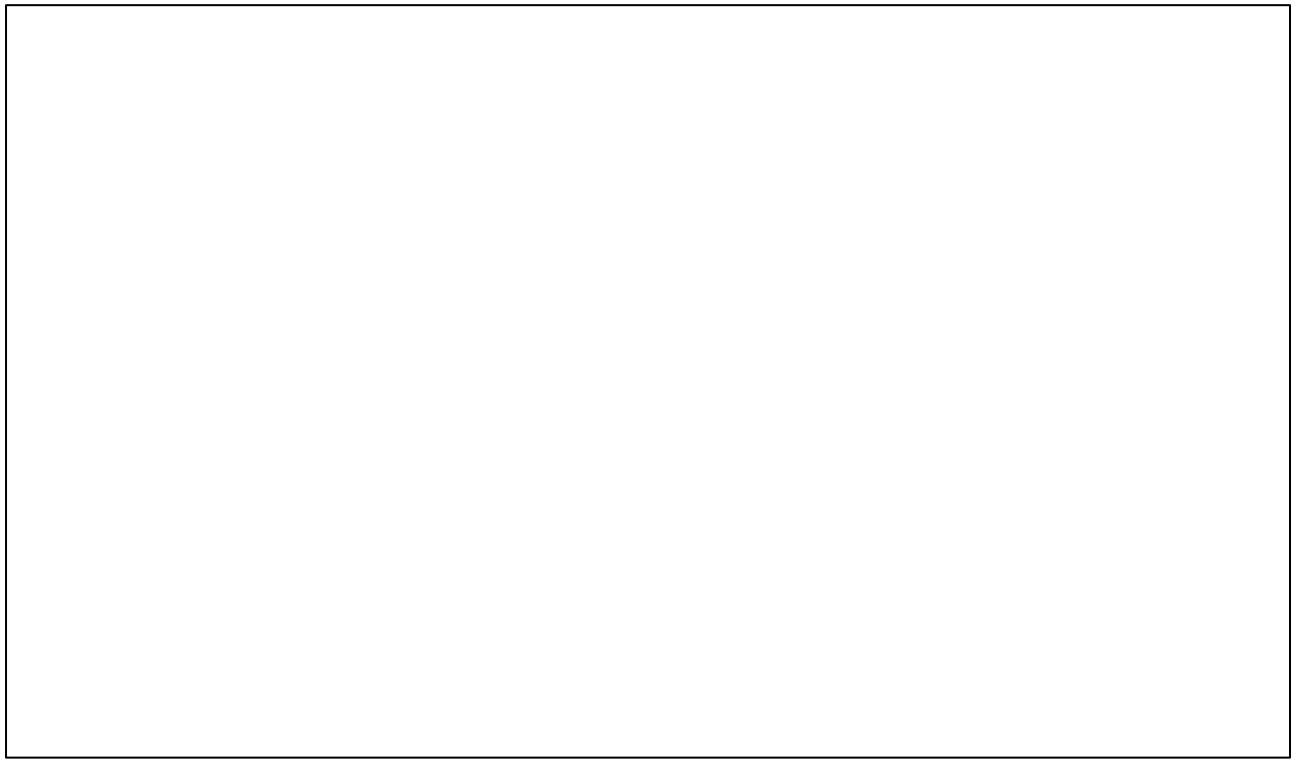
t：上昇
b：下降

- その他のキーで停止

演習3.
Pythonプログラムの作成

Bebop2を自作プログムで飛行制御する

- テキスト全文：ros_example_1_3.launch



演習3.
Pythonプログラムの作成

Bebop2を自作プログムで飛行制御する

- テキスト全文：ros_example_1_3.py

（移動操作）

初級コース

技術資料・
ダウンロードサイト

技術情報・ダウンロードサイト

ROS関連

- ROS公式ページ（日本語）：　http://wiki.ros.org/ja
- ROS（Kinetic）インストール：　http://wiki.ros.org/ja/kinetic/Installation/Ubuntu

Bebop2関連

- Parrot Bebop2（製品情報）：　https://www.parrot.com/jp/doron/parrot-bebop-2
- Bebop用ROSドライバー：　https://bebop-autonomy.readthedocs.io/en/latest/#

その他

- Ubuntu日本語Rimix：　https://www.ubuntulinux.jp/japanese
- teleop_twist_keyboard：　https://github.com/ros-teleop/teleop_twist_keyboard
- catkin buildの使い方：　https://catkin-tools.readthedocs.io/en/latest/index.html