

# ROSを用いた移動ロボットの知覚制御

知能機械工学専攻 中村友昭, 長井隆行

# 実験の流れ

---

## ■ 1週目（今日）

### ■ 3限 13:00 – 14:30

#### ■ イントロダクション

#### ■ サンプルプログラムを用いた演習

### ■ 4限 14:40 – 16:10

#### ■ 課題の実装

## ■ 2週目（来週）

### ■ 各自レポートの作成(自習)

※実験室に来る必要はありません



# 出席確認とグループ登録

## ■ 「デスクトップ¥M科グループ登録」を開く

### ■ アンケートに回答

- 実験日
- 班番号 A-X
- グループ番号 1-3
- 全員の学籍番号と氏名

### ■ 全てを記入したら送信をClick

質問 回答 18

### M科実験 グループ登録

出席確認を兼ねています。必ず入力してください。  
全てを入力したら送信ボタンを押してください

実験日 \*

年月日

班番号 A~X \*

記述式テキスト (短文回答)

グループ番号 \*

教員から指定されたグループ番号を選んでください

☐ グループ1

☐ グループ2

☐ グループ3

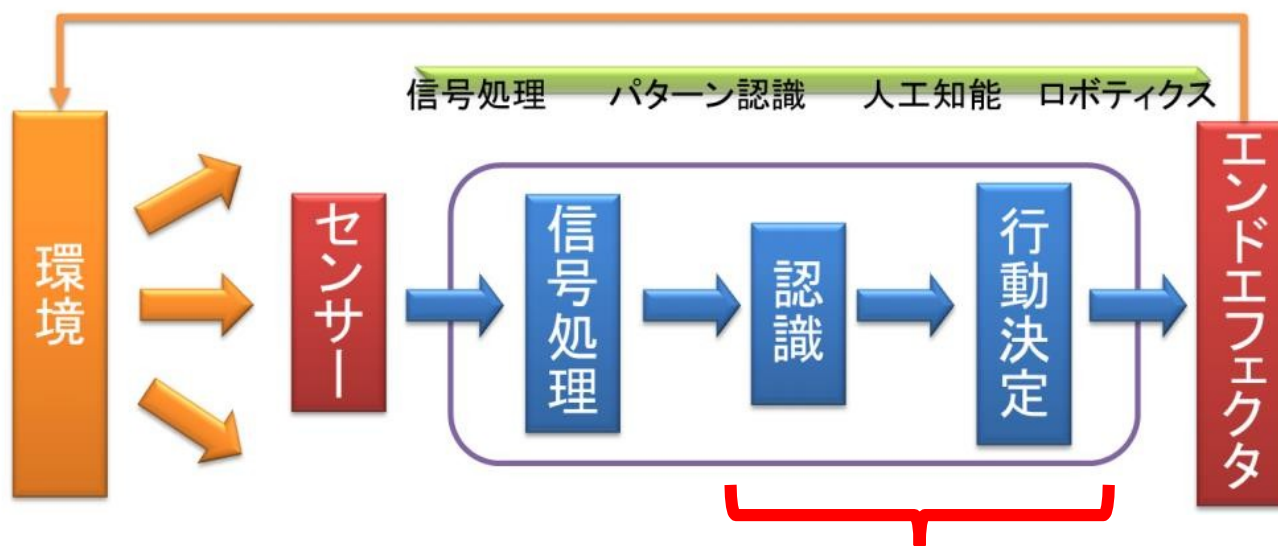
グループ全員の学籍番号と氏名 \*

例：123456 電通太郎, 456789電通次郎, 789123電通花子

記述式テキスト (短文回答)

## この実験の目的

### ■ 複数のセンサを統合したロボットシステム開発を行う

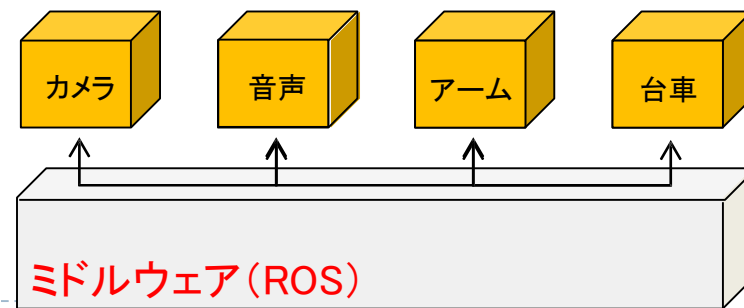


ここを皆さんに作ってもらいます

### ■ ミドルウェア(ROS)を使うことでロボットシステム開発が容易に行えることを体験する

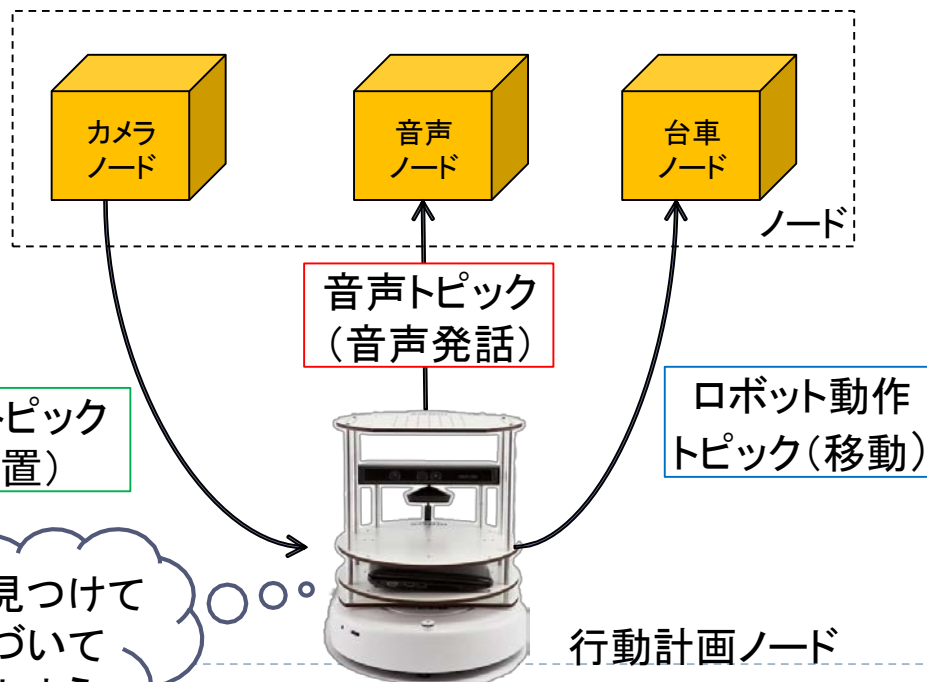
# ROS (Robot Operating System)

- Willow Garageが開発したロボット用OS
  - Ubuntuで動作しロボット開発に必要な様々な機能を提供
  - ROSの一部機能はWindowsで実行可
- ミドルウェアとは？
  - OSとアプリケーションの間に位置
  - 様々なソフトが利用可能な標準化されたインタフェース（通信機能）を提供
  - 簡単にソフト同士が通信可
- 利点
  - 独立して開発が可能
  - 一部が壊れてもシステム全体は停止しない
  - プログラムの再利用が可能



# ROSによるプログラム間の通信

- 様々なソフトウェアが様々な情報を送受信して動作
  - ノードやトピックの管理を行うソフトウェア = **ROSCore**
  - ソフトウェア = ノード
  - 情報 = トピック



ROSCoreにより管理

- どんなノードがあるか
- どのノードがどのトピックを発行(**Publish**)しているか
- どのノードがどのトピックを要求(**Subscribe**)しているか

# Python

---

## ■ この実験ではPythonというプログラミング言語を使用 利点

- コンパイルなしでソースコードをそのまま実行可能
- 型宣言が不要
- ポインタがない
- C言語に比べてソースコードが短く, シンプルになる
- C言語に比べて高機能

C言語

```
#include <stdio.h>
void main()
{
    printf("Hello World!");
}
```

Python

```
print "Hello World!"
```

# Python 制御構文

**{ }を使用せずインデントでブロックを表現**

## ■ if文

条件判定

例) 条件が真ならば処理1,  
条件が偽ならば処理2,  
を実行

C言語

```
if( 条件 ){  
    処理1;  
}else{  
    処理2;  
}
```

Python

```
if 条件 :  
    処理1  
else:  
    処理2
```

C言語

If

Python

if

else if

elif

else

else

式1 & 式2

式1 and 式2

式1 | 式2

式1 or 式2

!=式1

not 式1

## ■ while文

繰り返し処理

例) 処理1と処理2の  
無限ループ

C言語

```
while(1){  
    処理1;  
    処理2;  
}  
処理3;
```

Python

```
while 1:  
    処理1  
    処理2  
処理3
```

## ■ for文

繰り返し処理

例) 処理1が10回  
繰り返される

C言語

```
for( int i=0 ; i<10 ;i++ )  
{  
    処理1;  
}
```

Python

```
for i in range(10):  
    処理1
```



# 動画を適宜見ながらグループごとに進める

---

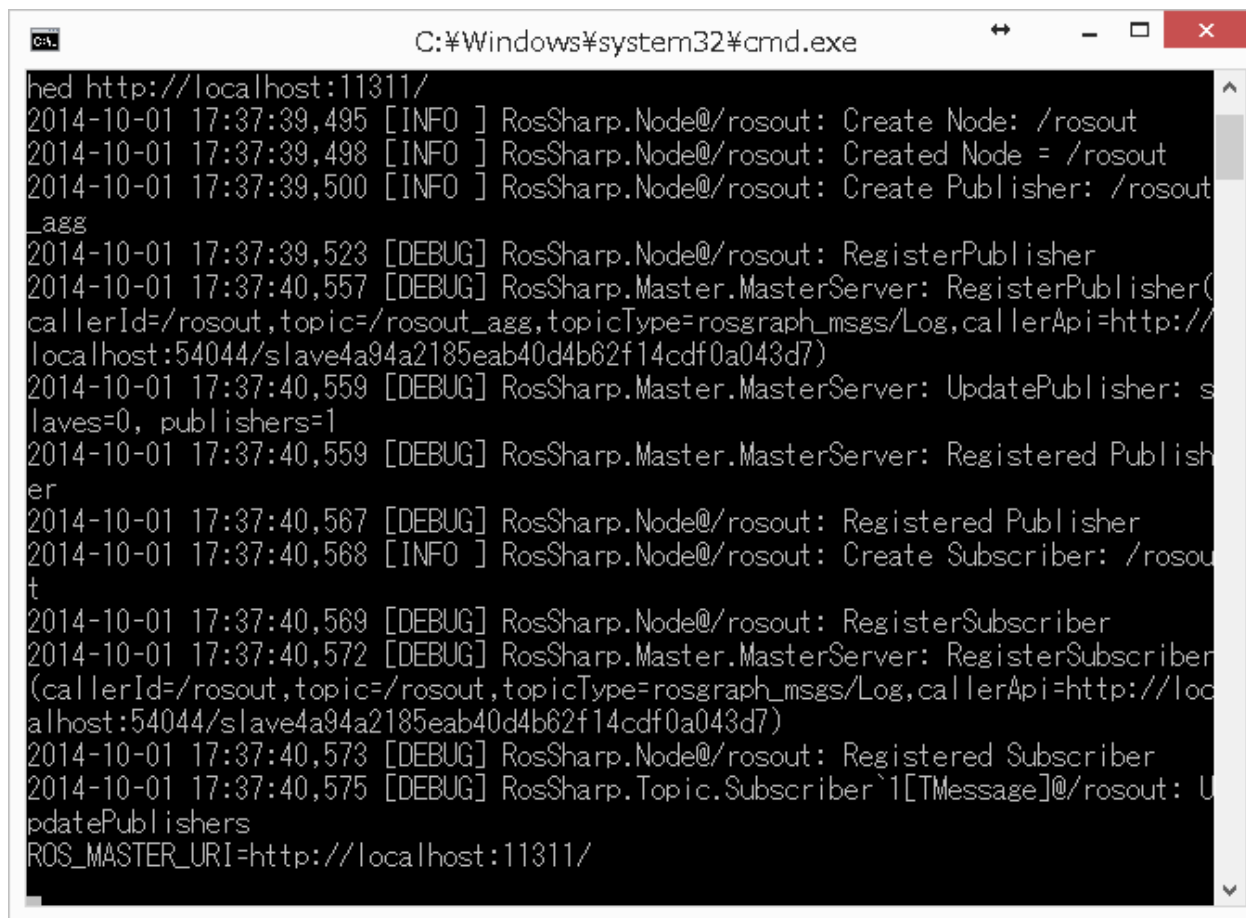
- ▶ 以下のページには動画へのリンクがある
  - ▶ ①ROSによる通信の基本
  - ▶ ②音声認識を使う
  - ▶ ③画像(Kinect)を使う
  - ▶ ④ロボットを動かす
- ▶ 適宜動画を見ながら実際に実行してみる
- ▶ それぞれのページで何をやっているのかを理解する
- ▶ プログラムにはPythonを用います
- ▶ 分からないことは検索や質問をする



# ①簡単なサンプルを動かしてみよう！

## ROSCOREの起動

### 「StartRoscore.bat」をダブルクリック



```
C:\Windows\system32\cmd.exe

hed http://localhost:11311/
2014-10-01 17:37:39,495 [INFO ] RosSharp.Node@/rosout: Create Node: /rosout
2014-10-01 17:37:39,498 [INFO ] RosSharp.Node@/rosout: Created Node = /rosout
2014-10-01 17:37:39,500 [INFO ] RosSharp.Node@/rosout: Create Publisher: /rosout
_agg
2014-10-01 17:37:39,523 [DEBUG] RosSharp.Node@/rosout: RegisterPublisher
2014-10-01 17:37:40,557 [DEBUG] RosSharp.Master.MasterServer: RegisterPublisher(
callerId=/rosout,topic=/rosout_agg,topicType=rosgraph_msgs/Log,callerApi=http://lo
localhost:54044/slave4a94a2185eab40d4b62f14cdf0a043d7)
2014-10-01 17:37:40,559 [DEBUG] RosSharp.Master.MasterServer: UpdatePublisher: s
laves=0, publishers=1
2014-10-01 17:37:40,559 [DEBUG] RosSharp.Master.MasterServer: Registered Publish
er
2014-10-01 17:37:40,567 [DEBUG] RosSharp.Node@/rosout: Registered Publisher
2014-10-01 17:37:40,568 [INFO ] RosSharp.Node@/rosout: Create Subscriber: /rosou
t
2014-10-01 17:37:40,569 [DEBUG] RosSharp.Node@/rosout: RegisterSubscriber
2014-10-01 17:37:40,572 [DEBUG] RosSharp.Master.MasterServer: RegisterSubscriber
(callerId=/rosout,topic=/rosout,topicType=rosgraph_msgs/Log,callerApi=http://loc
localhost:54044/slave4a94a2185eab40d4b62f14cdf0a043d7)
2014-10-01 17:37:40,573 [DEBUG] RosSharp.Node@/rosout: Registered Subscriber
2014-10-01 17:37:40,575 [DEBUG] RosSharp.Topic.Subscriber`1[TMessage]@/rosout: U
pdatePublishers
ROS_MASTER_URI=http://localhost:11311/
```

←こんな画面が出れば成功

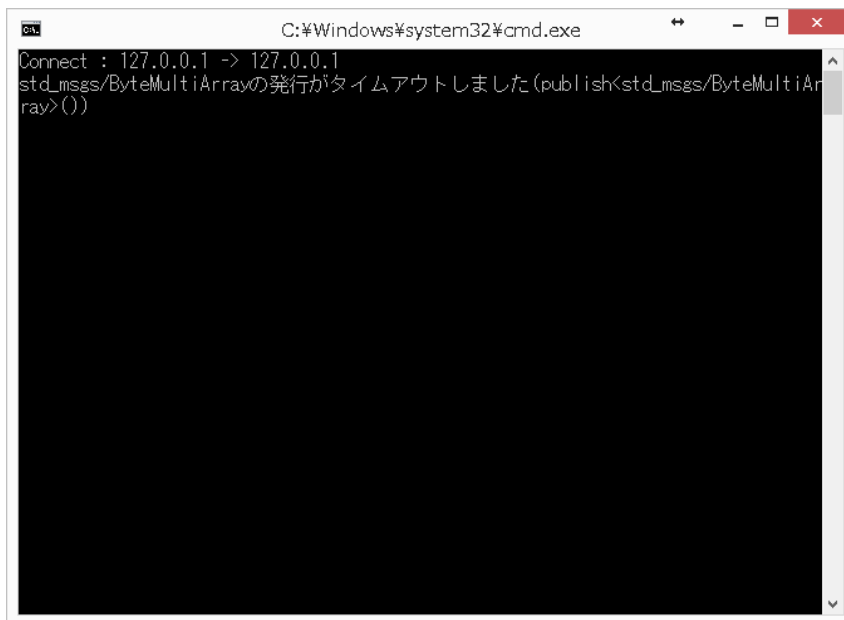
# ①簡単なサンプルを動かしてみよう！

## 文字列を送受信するサンプル

「sample¥SimpleSender.py」をダブルクリック

「sample¥SimpleReciever.py」をダブルクリック

↓こんな2つの画面が出れば成功↓



```
C:\Windows\system32\cmd.exe
Connect : 127.0.0.1 -> 127.0.0.1
std_msgs/ByteMultiArrayの発行がタイムアウトしました (publish<std_msgs/ByteMultiArray>())
```

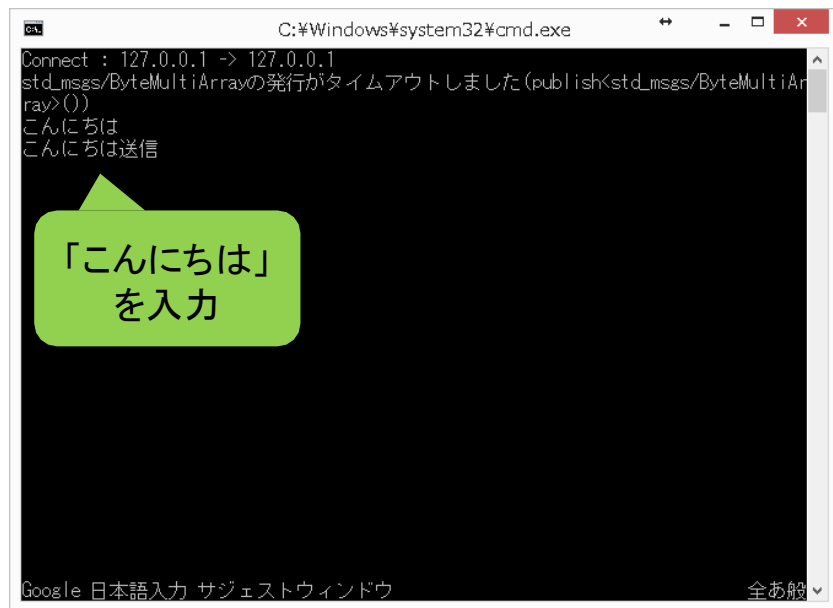


```
C:\Windows\system32\cmd.exe
Connect : 127.0.0.1 -> 127.0.0.1
```

# ①簡単なサンプルを動かしてみよう！

## 文字列を送受信するサンプル

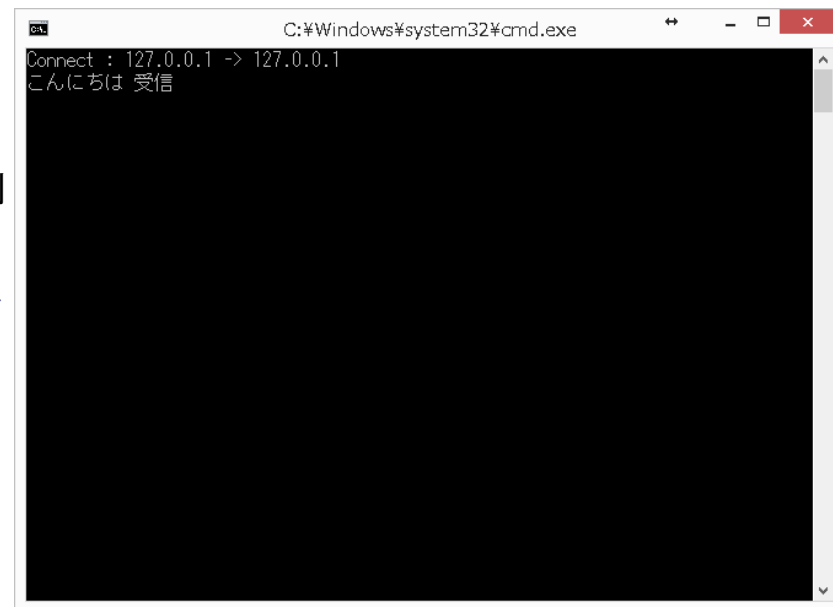
SimpleSenderに文字を入力してEnter



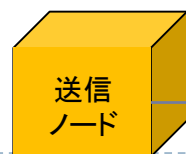
```
C:\Windows\system32\cmd.exe
Connect : 127.0.0.1 -> 127.0.0.1
std_msgs/ByteMultiArrayの発行がタイムアウトしました (publish<std_msgs/ByteMultiAr
ray>())
こんにちは
こんにちは送信
```

「こんにちは」  
を入力

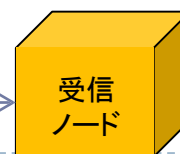
文字列  
転送



```
C:\Windows\system32\cmd.exe
Connect : 127.0.0.1 -> 127.0.0.1
こんにちは 受信
```



文字列トピック  
(配列)



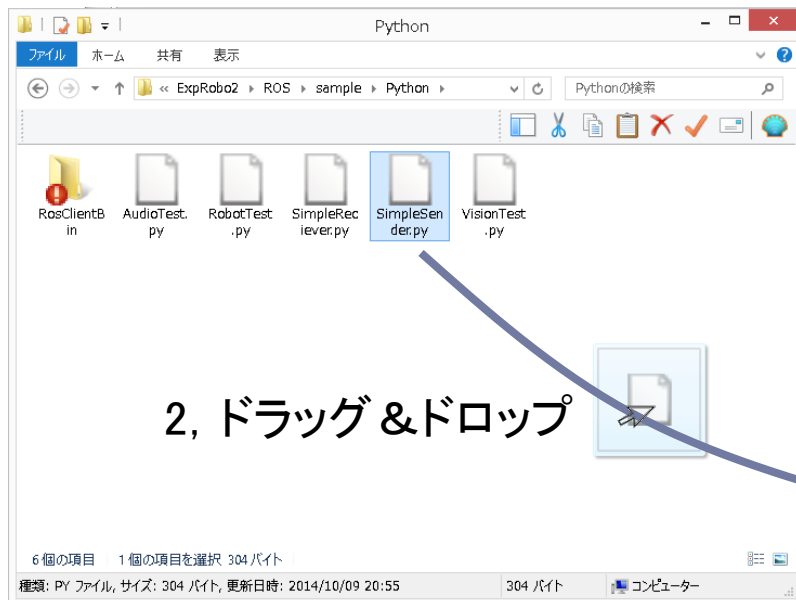
文字列トピックのPublish

文字列トピックのSubscribe

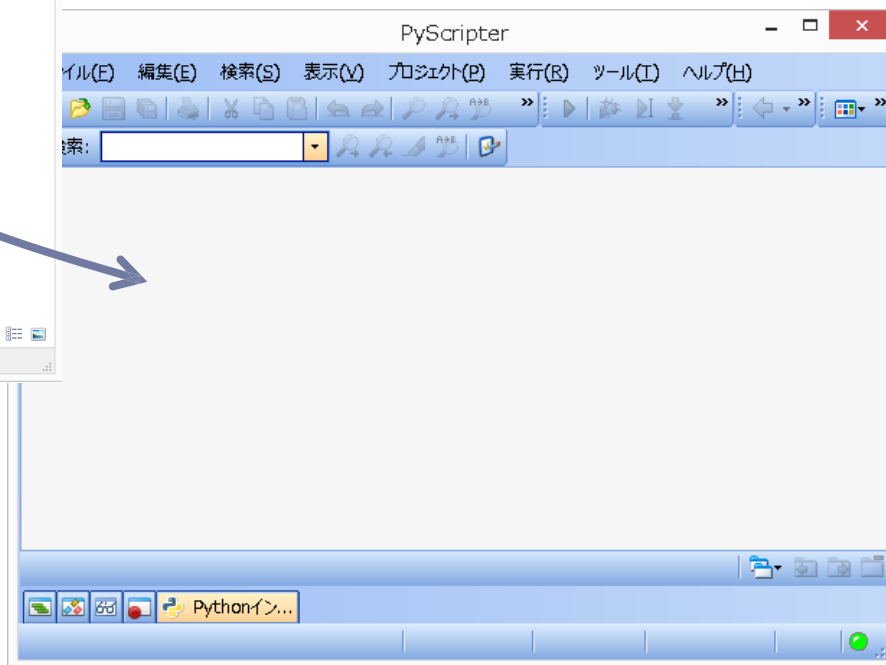
# ①送信側のソースコード

## 「sample¥SimpleSender.py」をPyScripterで開く

1, ダブルクリック



2, ドラッグ & ドロップ



※[ファイル]→[開く]で開いてもいいです

※プログラミング用のメモ帳みたいなもの

# ①送信側のソースコード

```
# coding: shift-jis
from RosClientBin import *

client = RosClient()
client.Connect( "localhost" , "11311" )
client.Publish[ByteArray]()
```

ROS利用するための  
ファイルを読み込み(import)

ノードの名前, Publishするトピックの設定等  
ROSCoreに通知

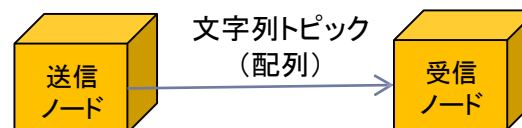
```
while 1:
    print "何か入力してください"
    str = raw_input()

    msg = ByteArray()
    msg.data = ToBytes( str )
    client.Send( msg )
```

文字列入力 文字列を送るための変  
数(箱)を用意

変数のdataに文字列を代入

↑  
文字列トピックを送信



文字列トピックのPublish

文字列トピックのSubscribe

## ▶ While文

- ▶ 繰り返し処理
- ▶ Pythonでは{ }を使用せずインデントでブロックを表現

### C言語

```
while(1){
    処理1;
    処理2;
}
処理2;
```

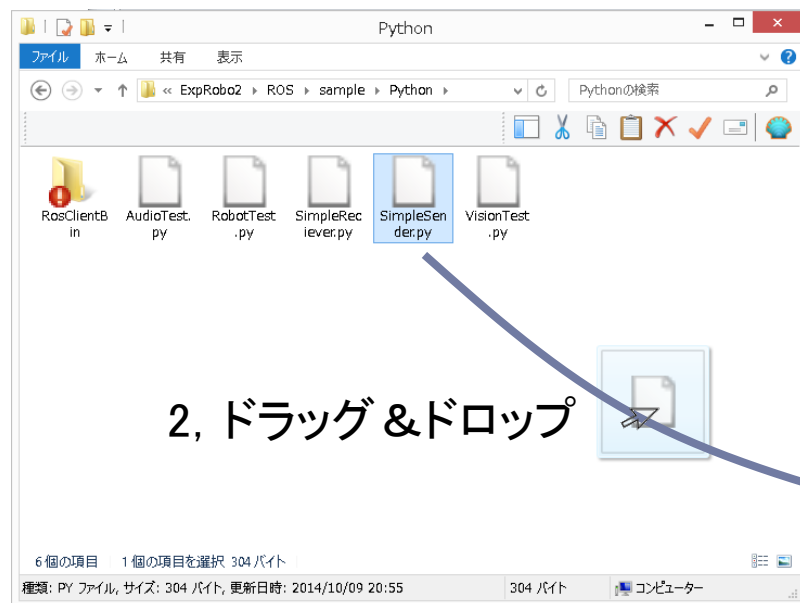
### Python

```
while 1:
    処理1
    処理2
    処理3
```

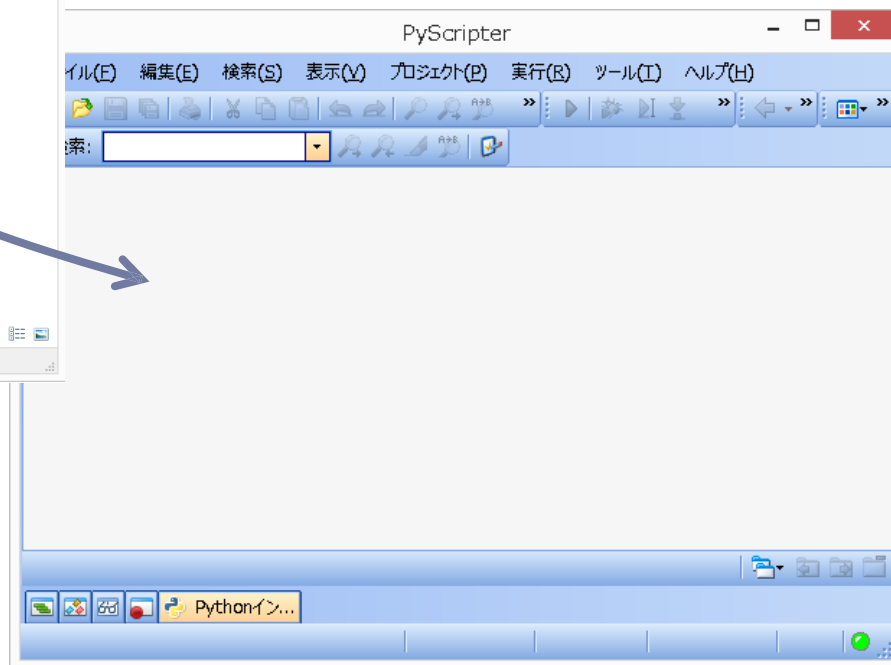
※処理1と処理2が  
永久に繰り返される

# ①受信側のソースコード

## 「sample¥SimpleReciever.py」をPyScripterで開く



2, ドラッグ & ドロップ



※[ファイル]→[開く]で開いてもいいです

# ①受信側のソースコード

```
# coding: shift-jis
from RosClientBin import *
```

```
client = RosClient()
client.Connect( "localhost" , "11311" )
client.Subscribe[ByteArray]()
```

```
while 1:
```

```
    msg = client.GetLastMsg[ByteArray]()
```

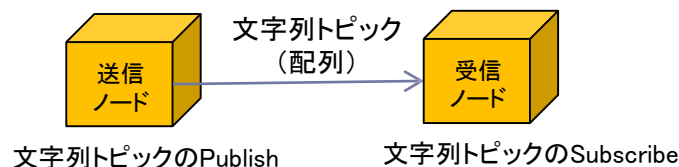
```
    if msg:
        print ToString(msg.data)
```

ノードの名前, **Subscribe**するトピックの設定等  
ROSCoreに通知

データを受信

受信に成功したか判定

受信したデータを 配列  
から文字列へ変換して表示



## ▶ if文

### ▶ 条件判定

C言語

```
if( 条件 ){
    処理1;
}else{
    処理2;
}
```

Python

```
if 条件:
    処理1
else:
    処理2
```

※条件が真ならば処理1が, 条件が  
偽ならば処理2が実行される

## ▶ print

### ▶ データを表示する関数

### ▶ dとか, lfとか指定しなくても自動的に表示

C言語

```
printf("d lf", 10, 9.9);
```

Python

```
print 10, 9.9
```

※カンマで複数  
続けることが可能



# 送信(Publish)の基本

---

```
# coding: shift-jis
from RosClientBin import *

client = RosClient()
client.Connect( "localhost" , "1"
client.Publish[ByteArray]()

while 1:
    print "何か入力してください "
    str = raw_input()

    msg = ByteArray()
    msg.data = ToBytes( str )
    client.Send( msg )
```

- ▶ Publishするトピックを通知する
  - ▶ トピック=変数の型
  - ▶ client.Publish[\*\*\*]();
- ▶ 送信するトピック(変数)を準備
  - ▶ 変数名=型名();
  - ▶ r = RobotInfo();
  - ▶ **変数の宣言がC言語と違うので注意！**
- ▶ データ(構造体)に値を入れる
  - ▶ client.Send( \*\*\* )で送信



# 受信 (Subscribe) の基本

---

```
# coding: shift-jis
from RosClientBin import *

client = RosClient()
client.Connect( "localhost" , "11311" )
client.Subscribe[ByteArray]()

while 1:
    msg = client.GetLastMsg[ByteArray]()

    if msg:
        print ToString(msg.data)
```

- ▶ Subscribeするトピックを通知する
  - ▶ トピック=変数の型  
client.Subscribe[\*\*\*]();
- ▶ データを受信する
  - ▶ msg = client.GetLastMsg[\*\*\*]()
  - ▶ 受信に成功していればmsgが存在する

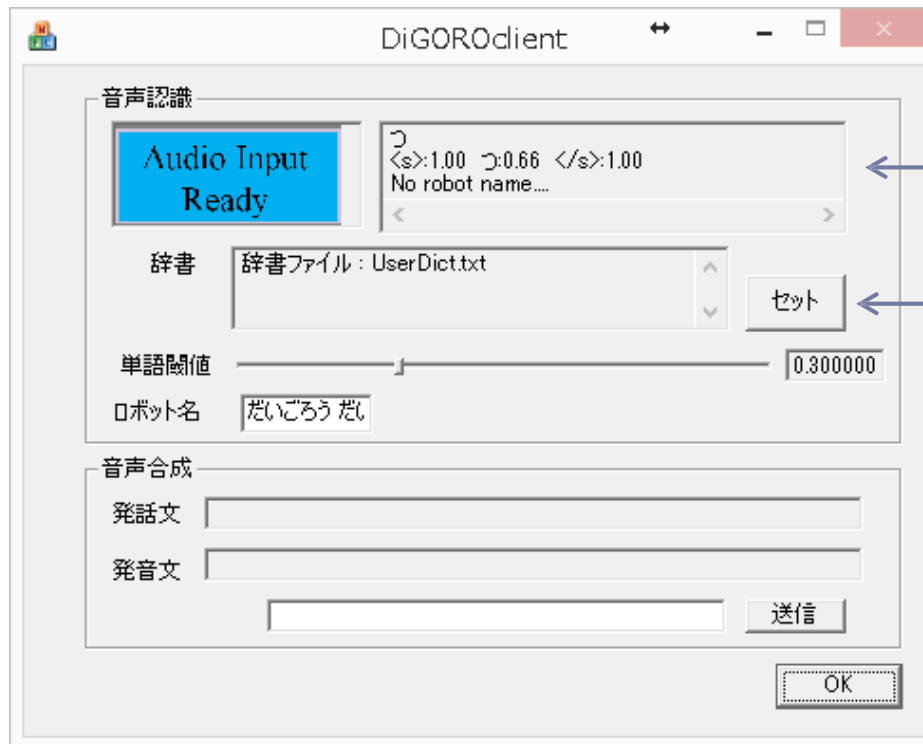
```
if msg:
    print "データがある！"
else:
    print "データがない..."
```



## ②音声ノードの使い方-起動-

音声の合成・認識を行うノード

起動方法:「StartAudio.bat」をダブルクリック



認識結果

認識辞書設定

## ②音声ノードの使い方 -認識辞書-

### 認識する単語の追加

```
[grammar]↓  
// 文法を書く↓  
S : DIGORO HELLO↓  
S : NOISE↓
```

DIGORO HELLO  
に単語が代入され認識される

```
[words]↓  
// 単語の定義↓  
// 漢字でも可能だが、できれば平仮名で書く↓  
// ●既知の誤変換↓  
// 中村⇒ちゅうそん↓  
// 明後日⇒みょうごにち↓  
// 私⇒わたくし↓  
// 行って⇒おこなって↓
```

DIGOROに代入される単語候補  
(この場合一つだけ)

```
% DIGORO↓  
だいごろう、↓
```

```
% NOISE↓  
つ↓  
っ↓
```

```
% HELLO↓  
こんにちは↓  
こんばんは↓  
おはよう↓  
おなかがすいた↓
```

HELLOに代入される単語候補  
ここに単語を追加すると 色々  
認識ができるようになる

この場合...

- だいごろう、こんにちは
- だいごろう、こんばんは
- だいごろう、おはよう
- だいごろう、おなかがすいた

が認識可能

## ②音声ノードの使い方 - ソースコード -

### 「sample¥AudioTest.py」の使い方

### 「sample¥AudioTest.py」を開く

```
# coding: shift-jis
from RosClientBin import *
```

```
client = RosClient()
client.Connect( "localhost" , "11311" )
client.Subscribe[SpeechInfo]()
client.Publish[SpeechOrder]()
```

```
while 1:
```

```
    # 音声認識結果の取得
```

```
    info = client.GetLastMsg[SpeechInfo]()
```

```
    if info:
```

```
        print ToString(info.recSpeech), info.isSpeaking
```

```
        if ToString(info.recSpeech).find("こんにちは")!=-1:
```

```
            # 音声発話命令を送信
```

```
            order = SpeechOrder()
```

```
            order.utterance = ToBytes("こんにちは")
```

```
            client.Send( order )
```

音声合成命令を送信

送受信(Publish, Subscribe)する情報の通知

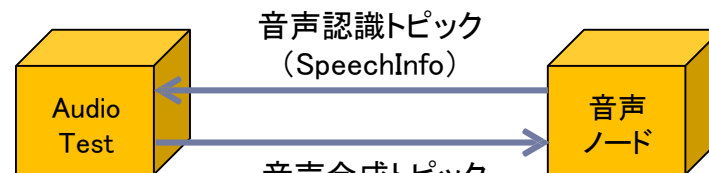
- SpeechInfo: 音声認識トピック(受信)
- SpeechOrder: 音声合成トピック(送信)

音声認識トピックSpeechInfoを受信

配列を文字列へ変換 文字列の中に「こ

んにちは」があるか検索

「こんにちは」を配列に変換して  
order.utteranceに代入



音声認識トピックのSubscribe  
音声合成トピックのPublish

音声認識トピックのPublish  
音声合成トピックのSubscribe

## ②課題1

1. まずはAudioTest.pyを起動して、マイクに向かって「だいごろう、こんにちは」と試してみよう
2. 「音声認識辞書.txt」を書き換えて、音声ノードにセットして、認識できるか試してみよう
3. AudioTest.pyを書き換えてチャットシステムを作ろう
  - ▶ 実行したプログラムは再起動しないと書き換えが有効にならないので注意！
  - ▶ 簡単なやり取りができるおしゃべりロボットにしてみましょう

```
↓
% DIGORO↓
だいごろう、 ↓
↓
% NOISE↓
つ↓
つ↓
↓
% HELLO↓
こんにちは↓
こんばんは↓
おはよう↓
おなかがすいた↓
```

ここに認識したい  
文を追加

```
while 1:
    # 音声認識結果の取得
    info = client.GetLastMsg[SpeechInfo]()

    if info:
        print ToString(info.recSpeech), info.isSpeaking
        if ToString(info.recSpeech).find("こんにちは")!=-1:
            # 音声発話命令を送信
            order = SpeechOrder()
            order.utterance = ToBytes("こんにちは")
            client.Send( order )
```

ここをコピー  
して書き換え

「音声認識辞書.txt」の下の方

「AudioTest.py」の下の方

### ③ ビジョンノードの使い方 - 起動 -

様々な画像認識を行うノード

起動方法: 「StartVision.bat」をダブルクリック

タブで各処理の結果表示を切り替え

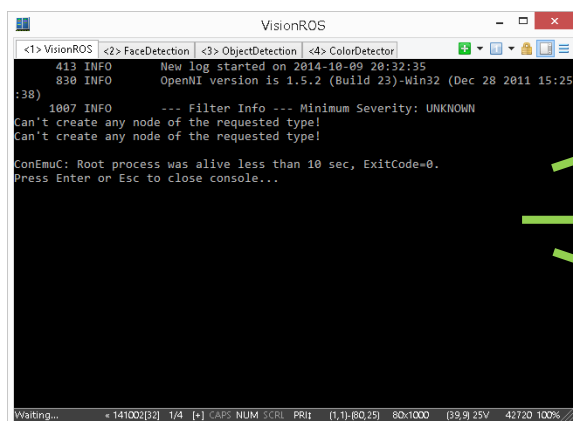
VisionROS: 骨格検出

FaceDetection: 顔検出

ObjectDetection: 物体検出

ColorDetection: 色検出

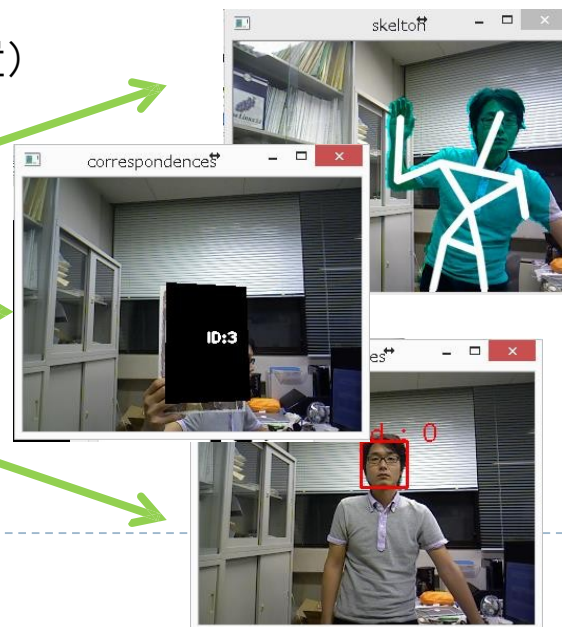
タブを切り替えて  
F1(F2)キーで検出結果画像表示



骨格(関節位置)  
検出

物体認識

顔検出



### ③ ビジョンノードの使い方 - ソースコード -

- ▶ 「sample¥VisionTest.py」を使ってみる (ビジョンノードから情報を受け取って処理するプログラム)
- ▶ 「sample¥VisionTest.py」のソースコードを開く

```
client = RosClient()  
client.Connect( "localhost" , "11311" )  
client.Subscribe[Skeltons]()  
client.Subscribe[Objects]()  
client.Subscribe[Faces]()
```

```
while 1:  
    # 骨格情報を取得  
    skeltons = client.GetLastMsg[Skeltons]()  
    if skeltons:  
        if skeltons.data.Count:  
            rightUp = False  
            leftUp = False  
            rightFront = False  
            leftFront = False  
            rightSide = False  
            leftSide = False
```

```
# 手の座標と頭の座標を基準に手の状態を識別  
if skeltons.data[0].joints[Skelton.SKELETON_RIGHT_H]  
if skeltons.data[0].joints[Skelton.SKELETON_LEFT_H]  
  
if skeltons.data[0].joints[Skelton.SKELETON_HEAD].z  
if skeltons.data[0].joints[Skelton.SKELETON_HEAD].z  
  
if skeltons.data[0].joints[Skelton.SKELETON_RIGHT_H]  
if skeltons.data[0].joints[Skelton.SKELETON_LEFT_H]
```

```
print "右手:",  
if rightUp:  
if rightFront:  
if rightSide:  
print  
print "上",  
print "前",  
print "横",
```

Visionノードからの情報をSubscribe

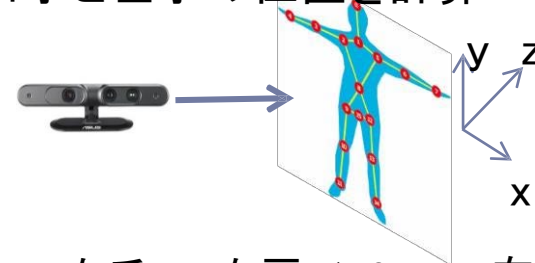
Skeltons: 骨格検出結果

Objects: 物体検出結果

Faces: 顔検出結果

骨格検出結果を受信

右手と左手の位置を計算



右手y - 右肩y > 0 : 右上

右手x - 右肩x < -300 : 右手横

右手と左手の位置の表示

• 上, 横, 前



# ③ ビジョンノードの使い方 - ソースコード -

## 物体認識結果の受信



```
# 物体検出結果受信
objects = client.GetLastMsg[Objects]()
if objects:
    for i in range( objects.data.Count ):
        print "物体検出 id:", objects.data[i].id, " pos:", objects.data[i].pos.x, objects.data[i].pos.y, objects.data[i].pos.z

# 顔検出結果受信
faces = client.GetLastMsg[Faces]()
if faces:
    for i in range( faces.data.Count ):
        print "顔検出 id:", faces.data[i].id, " pos:", faces.data[i].pos.x, faces.data[i].pos.y, faces.data[i].pos.z
```



## 顔検出結果の受信

- objects.data は配列
  - objects.data.Count で配列の要素数
  - objects.data[i] でi個目の物体の情報にアクセス
  - objects.data[i].pos.x : x座標
  - objects.data[i].pos.y : y座標
  - objects.data[i].pos.z : z座標
  - objects.data[i].id : 物体毎のID
- 
- Faces.data は配列
  - faces.data.Count で配列の要素数
  - faces.data[i] でi個目の物体の情報にアクセス
  - faces.data[i].pos.x : x座標
  - faces.data[i].pos.y : y座標
  - faces.data[i].pos.z : z座標

## ▶ for文

### ▶ 指定回数繰り返し

#### C言語

```
for( int i=0 ; i<10 ;i++)
{
    処理1;
}
```

#### Python

```
for i in range(10):
    処理1
```

※処理1が10回  
繰り返される

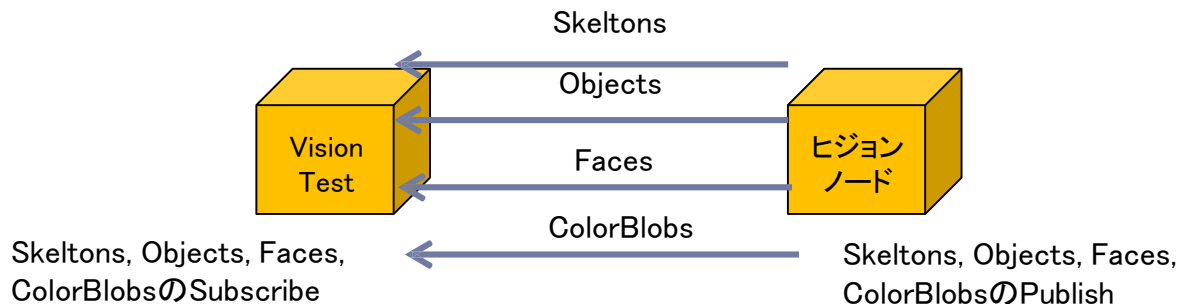
### ③ ビジョンノードの使い方 - ソースコード -

物体認識結果の受信



```
# 色検出結果を受信
color = client.GetLastMsg[ColorBlobs]()
if color:
    for i in range( color.data.Count ):
        print "色検出 id:" , color.data[i].id, " pos:", color.data[i].pos.x, color.data[i].pos.y, color.data[i].pos.z
```

- blobs.data は配列
- blobs.data.Count で配列の要素数
- blobs.data[i] でi個目の物体の情報にアクセス
- blobs.data[i].pos.x : x座標
- blobs.data[i].pos.y : y座標
- blobs.data[i].pos.z : z座標
- blobs.data[i].id : 色の種類を表すID



## ③課題2

- ▶ 「sample¥VisionTest.py」を書き換えて、ロボットが見てるシーンを音声で説明できるようにしてみましょう
  - ▶ 人が手を上げると「右手を上げてる人がある」と発話等
  - ▶ ヒント:
    - ▶ AudioText.pyの発話部分をVisionTest.pyにコピーして発話内容変更
    - ▶ 連続でしゃべり続けてしまう場合には発話後に「time.sleep(5)」(5秒間待機する関数)を入れる

```
if skeltons.data[0].joints[Skelton.SKELETON_RIGHT_HAND].x - skeltons.data[0].joints[Skelton.SKELETON_LEFT_HAND].x > 0.1:
    print "右手:",
    if rightUp:
        print "上",
        # 空欄

if rightFront:    print "前",
if rightSide:    print "横",
print

print "左手:",
if leftUp:        print "上",
if leftFront:    print "前",
if leftSide:    print "横",
print
```

VisionTest.py

```
client.Connect( "localhost" , "11311" )
client.Subscribe[SpeechInfo]()
client.Publish[SpeechOrder]()

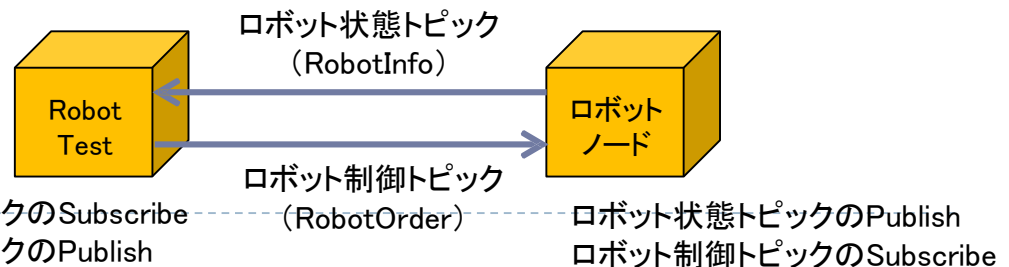
while 1:
    # 音声認識結果の取得
    info = client.GetLastMsg[SpeechInfo]()

    if info:
        print ToString(info.recSpeech), info.isSpeaking
        if ToString(info.recSpeech).find("こんにちは")!=-1:
            # 音声発話命令を送信
            order = SpeechOrder()
            order.utterance = ToBytes("こんにちは")
            client.Send( order )
```

AudioTest.py

## ④ロボットノードの使い方-起動-

- ▶ ロボットを制御するノード
- ▶ 起動方法:「StartRobot.bat」をダブルクリック

[illegible]

## ④ ロボットノードの使い方 - ソースコード -

「sample¥RobotTest.py」を使ってみる（ロボットノードと情報をやり取りして実際にロボットを動かす）

sample¥RobotTest.py」のソースコードを開く

```
client = RosClient()  
client.Connect( "localhost" , "11311" )  
client.Subscribe[RobotInfo]()  
client.Publish[RobotOrder]()
```

```
while 1:  
    if kbhit():  
        c = 0  
        while kbhit():  
            c = ord(getch())  
  
        order = RobotOrder()  
        if c==72:# ↑  
            print "直進"  
            order.kind = RobotOrder.ORDER_MOVE_FORWARD;  
            order.data.Add( 0.1 );  
        elif c==80:# ↓  
            print "後進"  
            order.kind = RobotOrder.ORDER_MOVE_FORWARD;  
            order.data.Add( -0.1 );  
        elif c==75: # ←  
            print "左回転"  
            order.kind = RobotOrder.ORDER_ROTATE;  
            order.data.Add( 0.5 );  
        elif c==77: # →  
            print "右回転"  
            order.kind = RobotOrder.ORDER_ROTATE;  
            order.data.Add( -0.5 );  
        else:  
            print "停止";  
            order.kind = RobotOrder.ORDER_STOP;  
  
    client.Send( order )
```

入力されたキーを取得

入力されたキーに応じて ロボット制御  
トピックRobotOrderに代入

直進命令とその速度(m/s)

回転命令とその速度(rad/s)

ロボット制御トピックを送信

## ④課題3

---

- ▶ 「sample¥AudioTest.py」(←**ファイル名に注意!**)を書き換えて音声でロボットを操作できるようにしましょう
    - ▶ 「だいごろう, まえにすすんで」で前進する等
    - ▶ 複数の操作ができるようにしましょう
  - ▶ ロボットが止まれるように「止まれ」などで停止もできるようにしましょう
  - ▶ 「了解」などロボットの発話を入れてみていいでしょう
  - ▶ **ロボットが停止できなくなった場合は, ロボットを持ち上げて, USBケーブルを抜いてください**
  - ▶ 次のページに**ヒント**があります
-

## ④課題3

### ▶ ヒント:

- ▶ AudioTestの発話部分をロボットの移動命令へ変更
- ▶ AudioNodeとAudioTestを書き換えて「まえにすすんで」等を認識できるように変更
- ▶ 色々な命令が実行できるようにしてみましょう！

```
while 1:  
    # 音声認識結果の取得  
    info = client.GetLastMsg[SpeechInfo]()  
  
    if info:  
        print ToString(info.recSpeech), info.isSpeaking  
        if ToString(info.recSpeech).find("こんにちは")!=-1:  
            # 音声発話命令を送信  
            order = SpeechOrder()  
            order.utterance = ToBytes("こんにちは")  
            client.Send( order )
```

AudioTest.py

認識文・認識文字列を変更

ロボットへの  
命令へ変更

直進命令代入

RobotTest.py

```
c = ord(getch())  
  
order = RobotOrder()  
if c==72:# ↑  
    print "直進"  
    order.kind = RobotOrder.ORDER_MOVE_FORWARD;  
    order.data.Add( 0.1 );  
elif c==80:# ↓  
    print "後進"  
    order.kind = RobotOrder.ORDER_MOVE_FORWARD;  
    order.data.Add( -0.1 );  
elif c==75:# ←  
    print "左回転"  
    order.kind = RobotOrder.ORDER_ROTATE;  
    order.data.Add( 0.5 );  
elif c==77:# →  
    print "右回転"
```

## ④ ロボットノードの使い方 - ソースコード -

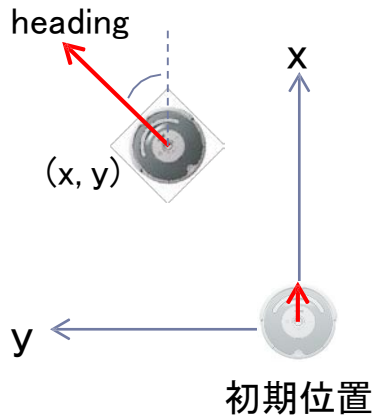
ロボット状態トピックを取得



```
info = client.GetLastMsg[RobotInfo]()
if info:
    print "x:", info.posx , " y:" , info.posy , " theta:" , info.heading
    print "衝突(左, 中心, 右):" , info.isClisionDetectedL , info.isClisionDetectedC , info.isClisionDetectedR
    print "-----"
```



ロボット状態を表示



変数名	内容
info.posx	ロボットの現在位置のx座標(m)
info.posy	ロボットの現在位置のy座標(m)
info.heading	ロボットの現在位置の回転方向(rad)
info.ismoving	ロボットが動いているかどうか
info.isClisionDetectedL	左バンパーが接触しているかどうか
info.isClisionDetectedC	正面バンパーが接触しているかどうか
info.isClisionDetectedR	右バンパーが接触しているかどうか
info.isCliffDetectedL	左に段差があるかどうか
info.isCliffDetectedC	正面に段差があるかどうか
info.isCliffDetectedR	右に段差があるかどうか



## ④ロボットノードの使い方2-ソースコード-

- ▶ 「sample¥RobotTestDistAngle.py」を使ってみる
- ▶ 「sample¥RobotTestDistAngle.py」のソースコードを開く

```
#####
print "直進"
order = RobotOrder()
order.kind = RobotOrder.ORDER_MOVE_DIST
order.data.Add(0.5)
client.Send( order )
```

直進命令  
距離(m)

```
while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"
```

ロボットが 停止するまで  
ループ info.ismoving : 動作中か？

```
#####
print "回転"
order = RobotOrder()
order.kind = RobotOrder.ORDER_ROTATE_ANGLE
order.data.Add(1.507)
client.Send( order )
```

回転命令  
角度(rad)

```
while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"
```

ロボットが 停止するまで  
ループ info.ismoving : 動作中か？

```
#####
print "逆回転"
order = RobotOrder()
order.kind = RobotOrder.ORDER_ROTATE_ANGLE
order.data.Add(-1.507)
client.Send( order )
```

回転命令  
角度(rad)

```
while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"
```

```
#####
print "後進"
order = RobotOrder()
order.kind = RobotOrder.ORDER_MOVE_DIST
order.data.Add(-0.5)
client.Send( order )
```

直進命令  
距離(m)

```
while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"
```

## ④課題4

- ▶ sample¥RobotTestDistAngle.pyを書き換えて独自の複雑なモーションをロボットにさせてみよう！
  - ▶ ジグザグに動く, 多角形を描く等
- ▶ ヒント: 直進部分, 回転部分をコピペしてつなぎあわせる

コピペするときにインデントの量  
(左側のスペースの量)に注意！！

```
#####
print "直進"
order = RobotOrder()
order.kind = RobotOrder.ORDER_MOVE_DIST
order.data.Add(0.5)
client.Send( order )

while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"

#####
print "回転"
order = RobotOrder()
order.kind = RobotOrder.ORDER_ROTATE_ANGLE
order.data.Add(1.507)
client.Send( order )

while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"
```

```
#####
print "90度"
order = RobotOrder()
order.kind = RobotOrder.ORDER_ROTATE_ANGLE
order.data.Add(0.87266)
client.Send( order )

while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"

#####
print "0.5m直進"
order = RobotOrder()
order.kind = RobotOrder.ORDER_MOVE_DIST
order.data.Add(0.5)
client.Send( order )

while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"

#####
print "90度"
order = RobotOrder()
order.kind = RobotOrder.ORDER_ROTATE_ANGLE
order.data.Add(0.87266)
client.Send( order )

while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"

#####
print "0.5m直進"
order = RobotOrder()
order.kind = RobotOrder.ORDER_MOVE_DIST
order.data.Add(0.5)
client.Send( order )

while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"

#####
print "90度"
order = RobotOrder()
order.kind = RobotOrder.ORDER_ROTATE_ANGLE
order.data.Add(0.87266)
client.Send( order )

while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"

#####
print "0.5m直進"
order = RobotOrder()
order.kind = RobotOrder.ORDER_MOVE_DIST
order.data.Add(0.5)
client.Send( order )

while 1:
    time.sleep(2)
    info = client.GetLastMsg[RobotInfo]()
    if info:
        if info.ismoving==False:
            "停止"
            break
        else:
            print "移動中"
```

??度回転

??m直進

??度回転

??m直進

??度回転

??m直進

# 型一覧

## ■ 実験で使用する型の一覧

送信: 送信(Publish)すべきデータ

受信: 受信(Subscribe)すべきデータ

型	説明	送信	受信
List	可変長配列(高機能な配列) list[n] : n番目の要素の参照 list.Add(*) : 要素の追加 list.Count : 要素の数	X	X
ByteArray	配列. 文字列の送受信に利用. 文字列との変換が必要. s = ToString(arr.data) : 文字列に変換 arr.data = ToBytes("aaa") : 文字列からByte配列に変換	X	X
SpeechOrder	音声合成命令送信用の構造体. order.utterance : 認識結果文字列(ByteMultiArray)	O	X
SpeechInfo	音声認識受信用構造体. info.recSpeech : 認識結果文字列(ByteMultiArray) info.isSpeaking : ロボットが音声発話中かどうか?(bool)	X	O

型	説明	送信	受信
Pos3d	三次元位置を表す構造体: pos.x : x座標 (float) pos.y : y座標 (float) pos.z : z座標 (float)	X	X
Objects	検出された物体情報の受信用の構造体. objects.data : 物体情報 (ObjectDataのList)	X	O
Faces	検出された顔情報の受信用の構造体. faces.data : 顔情報 (ObjectDataのList)	X	O
ObjectData	物体又は顔の情報を表す構造体. obj.id : 物体や顔のID (int) obj.pos : 物体や顔の位置 (Pos3D)	X	X
Skeltons	人の骨格情報を受信するための構造体 skeltons.data : 骨格情報 (SkeltonDataのList)	X	O
SkeltonData	人の骨格情報を表す構造体 skelton.id : 人のID (int) skelton.joints : 各関節の位置 (Pose3DのList)	X	X
ColorBlobs	色検出結果を受信するための構造体 blobs.data: 色検出の結果 (ColorBlobのList)	X	O
ColorBlob	色検出の結果が格納される構造体 blob.id : 色ID (int) blob.pos : 三次元位置 (Pos3D)	X	X



型	説明	送信	受信
RobotOrder	ロボットへの命令送信用の構造体 kind : 命令の種類 (RobotOrder::ORDER_ROTATE等) RobotOrder::ORDER_MOVE_FORWARD RobotOrder::ORDER_ROTATE RobotOrder::ORDER_MOVE_DIST RobotOrder::ORDER_ROTATE_ANGLE RobotOrder::ORDER_STOP Data : 命令に必要な情報 (floatのList)	X	O
RobotInfo	ロボットの情報受信用の構造体 info.posx : ロボットのx座標 (float) info.posy : ロボットのy座標 (float) info.heading : ロボットの向いている方向 (float) info.ismoving : ロボットが動いているかどうか (bool) info.isClisionDetectedL : 左バンパーの接触状況 (bool) info.isClisionDetectedC : 正面バンパーの接触状況 (bool) info.isClisionDetectedR : 右バンパーの接触状況 (bool) info.isCliffDetectedL : 左側に段差があるか (bool) info.isCliffDetectedC : 正面に段差があるか (bool) info.isCliffDetectedR : 右側に段差があるか (bool)	O	X



# レポートについて

---

- 締め切り 2週間後の木曜日 24時
- 提出先 tnagai@ee.uec.ac.jp
- 件名 M科実験(学籍番号)  
(例) M科実験1234567
- ファイル名 学籍番号.pdf

## 注意

- メールの本文にも名前と学籍番号は記入すること
- レポートには表紙をつけること
- 講評は行いません

