

ROS勉強会 < 入門編 >

発表日：2013年8月2日

作成者：前川 大輝
野崎 耕平

目次

- 第零章：この勉強会の位置づけ
- 第一章：ROSとは
- 第二章：RGBDセンサの制御
- 第三章：rviz
- 第四章：3D点群処理

第零章：この勉強会の位置づけ

- あくまでも入門編なのでわかりやすさを重視
- ROSを用いることで何ができるのかについての説明
 - 詳細は公式ドキュメントを参照

第一章：ROSとは

■ ロボット制御用のミドルウェア

→ Willow Garage社が開発・保守している

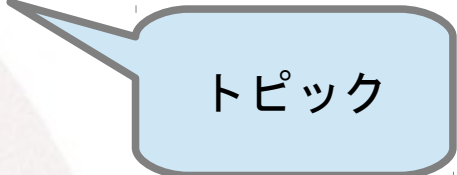
通信の基本単位

- ソフトウェアの単位 → ノード

- ロボットのセンサやアクチュエータなどを別々のノードとして定義
→ それを組み合わせ分散システムを構築

ノードの種類

- ノード
 - パブリッシャー (データ送信)
 - サブスクライバー (データ受信)



トピック

ノードの種類

- ノード → サービス(特定の処理)
- サービスは利用側(多くはノード)からリクエストを受け取ると特定の処理を行うもの

パラメータ

- トピック以外のデータを送受信するための仕組み
- パラメータサーバがデータを保持
 - 他のノードからの変更は通知される

マスター

- ノードやトピックの管理を行う
- roscore というプロセスにより提供される

パッケージ

- 機能ごとに分類されたノードやライブラリの単位のこと
- パッケージを管理するための便利なコマンドが多数提供されている
- ロボット用のオープンソースパッケージが多数公開されている

第二章： *RGBD* センサの制御

ノード間通信の具体例として

ROS上でXtionを取り扱う方法を解説

パッケージのインストール

```
sudo ros-groovy-openni-camera  
sudo ros-groovy-openni-launch  
sudo ros-groovy-openni-tracker
```

OpenNI をROS向けに改良したもの

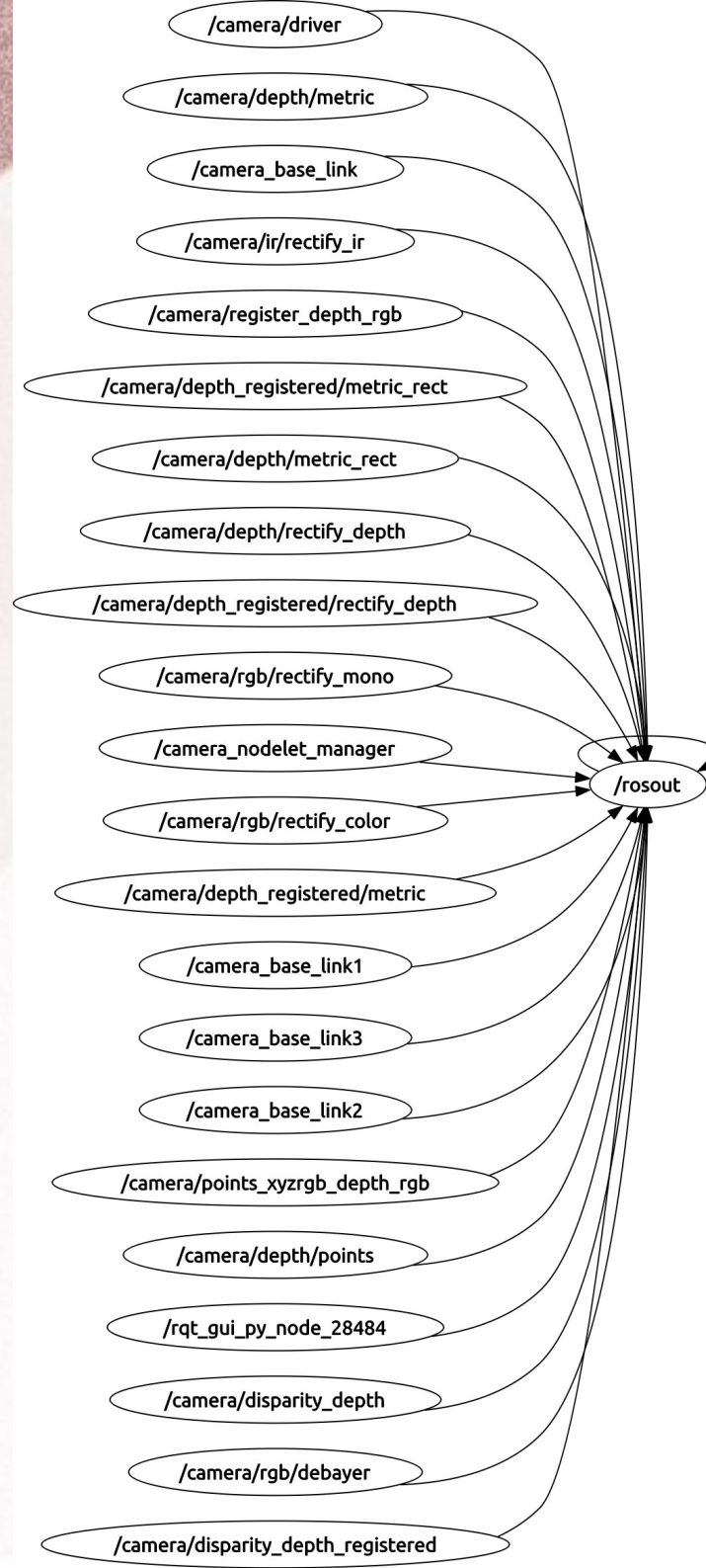
*Xtion*からの情報を配信

```
roslaunch openni_launch openni.launch
```

Xtionのデータがトピックにパブリッシュされる

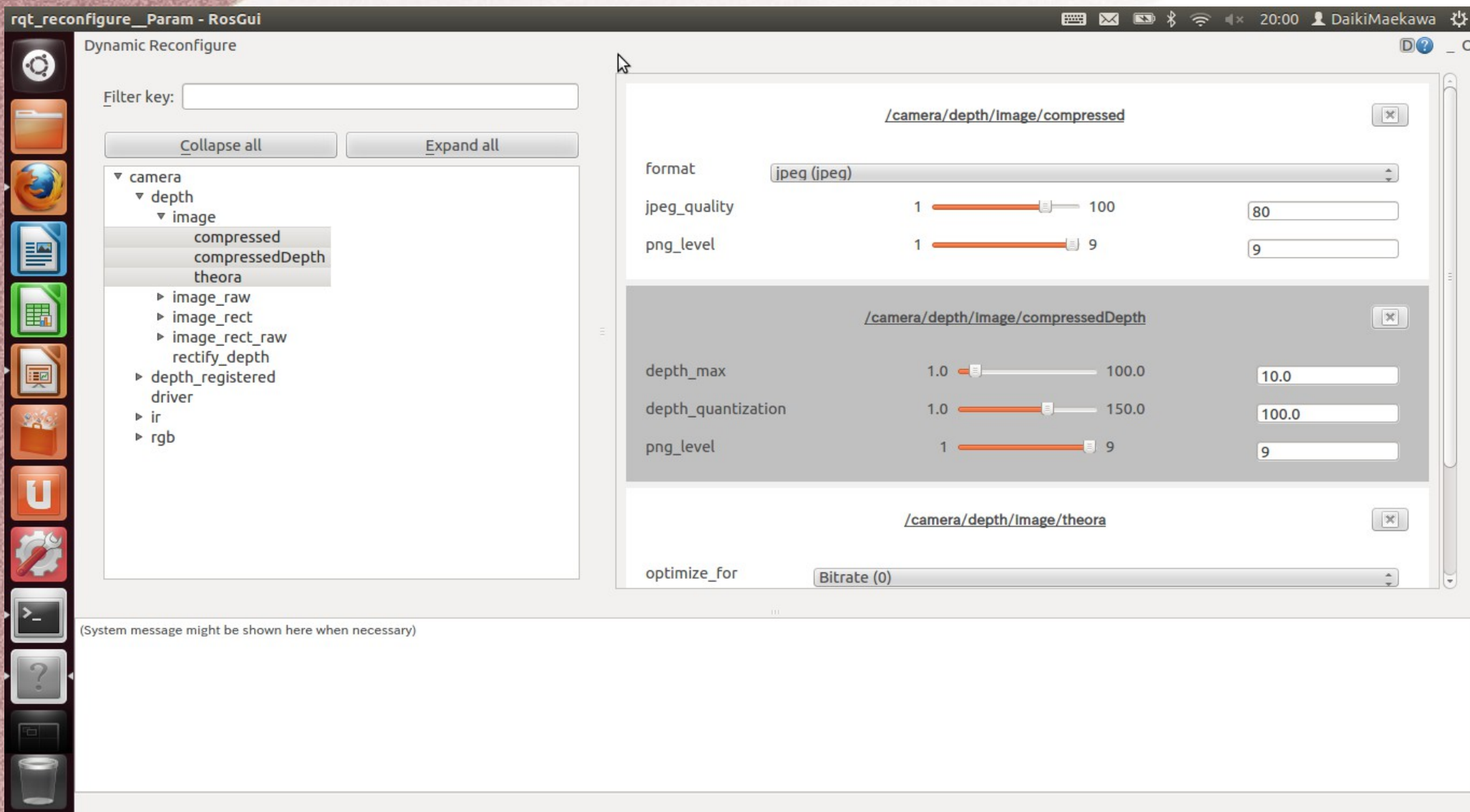
*Xtion*からの情報を配信

rqt_graphでノード間通信をグラフ化すると…



ノードのパラメータ編集

rosrun rqt_reconfigure rqt_reconfigure



データを視覚化

以下のトピックに注目

- デプス画像 → `/camera/depth/image`
- RGB画像 → `/camera/rgb/image_color`

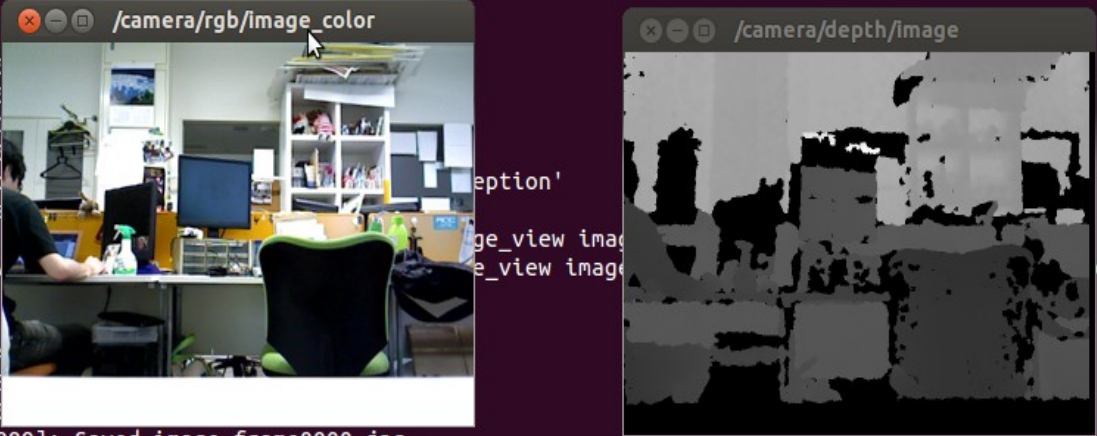
データを視覚化

```
■roslaunch image_view image_view  
  image:=/camera/depth/image &  
  roslaunch image_view image_view  
    image:=/camera/rgb/image_color
```

imageというトピックをそれぞれのデータに合わせてリマップ

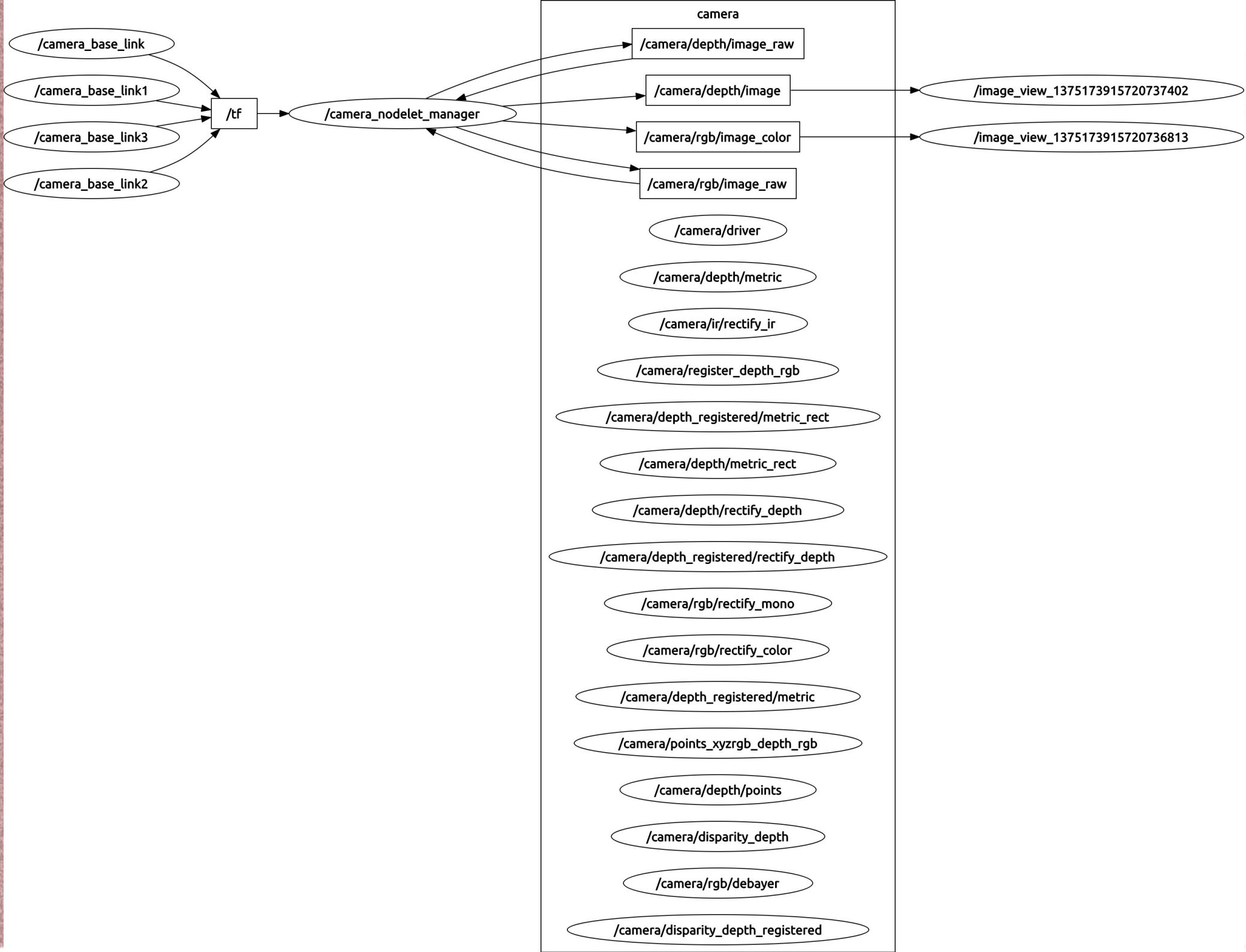
データを視覚化

```
/camera/rgb/image_color
daikimaekawa@daikimaekawa-VPCSB1AGJA:~$ rosrun image_view image_view image:=/camera/depth/image & rosrun image_view image_view image:=/camera/rgb/image_color
[1] 22447
^Cdaikimaekawa@daikimaekawa-VPCSB1AGJA:~$
daikimaekawa@daikimaekawa-VPCSB1AGJA:~$
daikimaekawa@daikimaekawa-VPCSB1AGJA:~$ terminate called after throwing an instance of 'boost::exception_detail::clone_impl<boost::exception_detail::error_info_injector<boost::lock_error> >'
what(): boost::lock_error
^C
[1]+ 中止 (コアダンプ) rosrun image_view image_view image:=/camera/depth/image
daikimaekawa@daikimaekawa-VPCSB1AGJA:~$ rosrun image_view image_view image:=/camera/depth/image & rosrun image_view image_view image:=/camera/rgb/image_color
[1] 24399
[ INFO] [1375174453.8219570]
[ INFO] [1375174454.4532240]
[ INFO] [1375174470.3735090]
[ INFO] [1375174491.4540530]
terminate called after throwing an instance of 'cv_bridge::Exception'
what(): Unrecognized image encoding
[1]+ 中止
daikimaekawa@daikimaekawa-VPCSB1AGJA:~$ rosrun image_view image_view image:=/camera/depth/image & rosrun image_view image_view image:=/camera/rgb/image_color
[1] 24951
[ INFO] [1375174524.8096990]
[ INFO] [1375174525.2107560]
[ INFO] [1375174525.5827840]
[ INFO] [1375174527.326516999]: Saved image frame0000.jpg
[ INFO] [1375174527.696254461]: Saved image frame0001.jpg
[ INFO] [1375174528.057296041]: Saved image frame0002.jpg
[ INFO] [1375174531.653121599]: Saved image frame0003.jpg
[ INFO] [1375174537.972268330]: Saved image frame0004.jpg
[ INFO] [1375174538.109678752]: Saved image frame0005.jpg
daikimaekawa@daikimaekawa-VPCSB1AGJA:~$ terminate called after throwing an instance of 'cv_bridge::Exception'
what(): Unrecognized image encoding [32FC1]
^C
[1]+ 中止 (コアダンプ) rosrun image_view image_view image:=/camera/depth/image
daikimaekawa@daikimaekawa-VPCSB1AGJA:~$ rosrun image_view image_view image:=/camera/depth/image & rosrun image_view image_view image:=/camera/rgb/image_color
[1] 25091
[17:56] 0 bash 1 bash 2 bash
```



データを視覚化

rqt_graphでノード間通信をグラフ化すると…



骨格情報の取得

■ `roslaunch openni_tracker openni_tracker`

データの記録と再生

- デバックの度にXtionを用意し、体を動かすのは面倒
 - ROSなら簡単にデータを記録、再生できる

ROS上のデータにあたるトピックが対象

トピックの記録

- パブリッシュされているトピックを記録

- `rosbag record トピック名`

- 終了はCtrl-C

- 「年月日 + 時間 + .bag」 ファイル(バグファイル)を生成

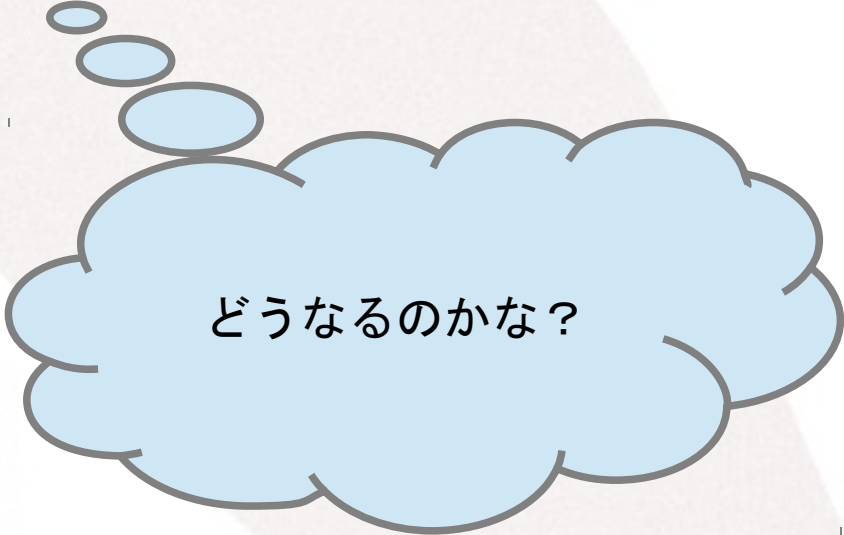
パブリッシュされているすべてのトピックを記録したい場合
`rosbag record -a`

演習 1

- 1) 配布されたバグファイルの内容を表示してから再生してみよう
- 2) バグファイルに記録された画像データをimage_viewに表示してみよう
- 3) rvizを用いて骨格情報を視覚化してみよう

バグファイルを調べる

■rosbag info バグファイル名

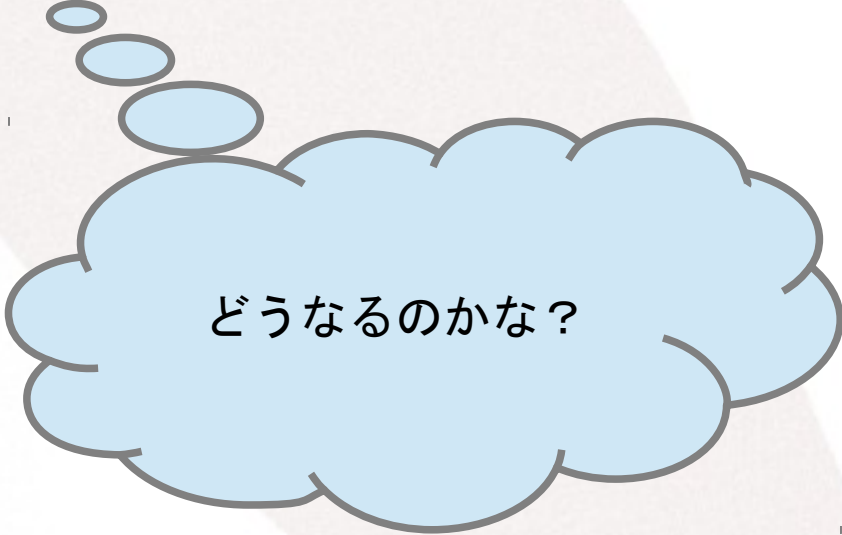


どうなるのかな？

バグファイルの再生

■ `rosbag play xtion_data.bag`

SPACE	: 一時停止
s	: ステップ



どうなるのかな？

データを視覚化

以下のトピックに注目

■デプス画像 → `/camera/depth/disparity`

データを視覚化

```
■rosrun image_view disparity_view  
image:=/camera/depth/disparity
```

imageというトピックからdisparityにリマップ

データを視覚化

/camera/depth/disparity

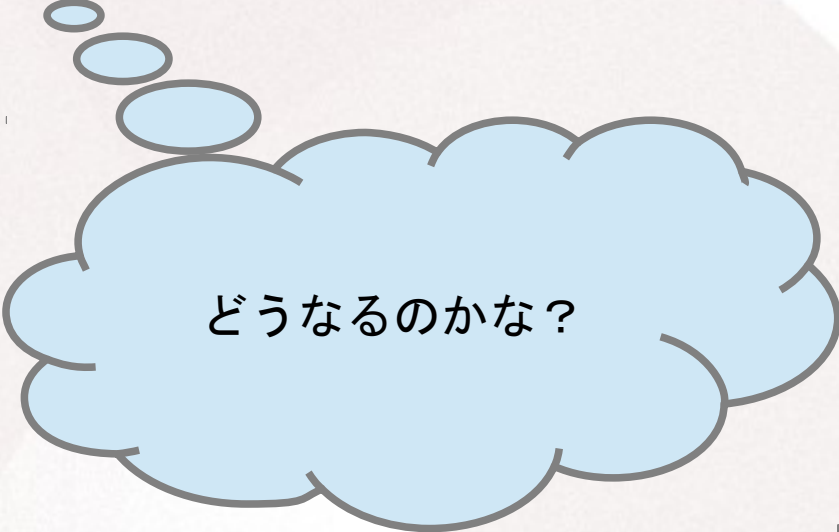
04:38



骨格情報の視覚化

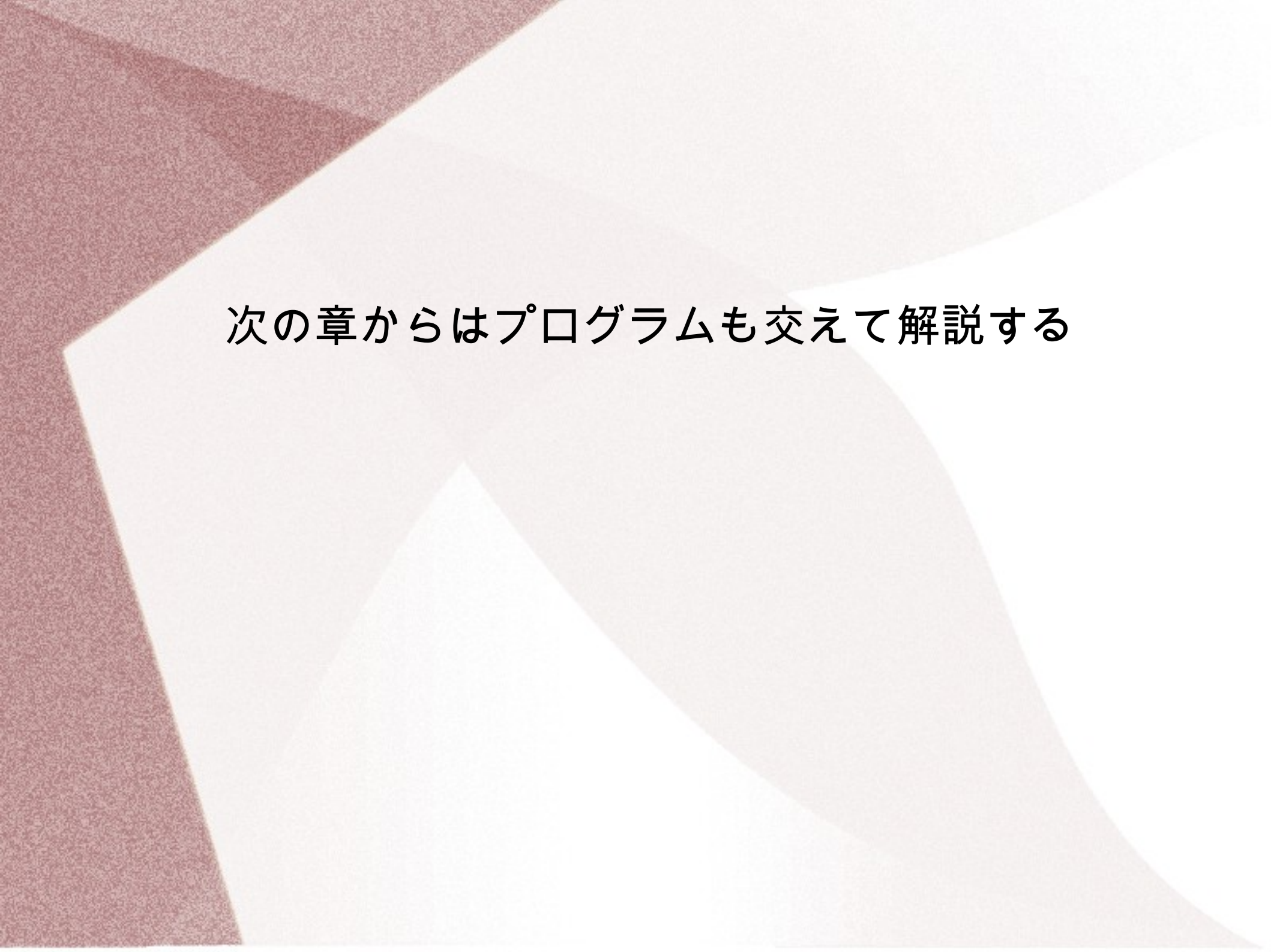
■視覚化

→ `roslaunch rviz rviz`



どうなるのかな？

ここまで一行もソースコードを書いていない...



次の章からはプログラムも交えて解説する

第三章：*rviz*

骨格情報を表示する際に用いたrvizについて解説する

URGでも使ってみよう

実演

*rviz*とは

ROS

[要出典] Visualization [要出典]

データを 3次元マップ上に 可視化

みなさんも
使ってみよう

■rvizを起動

→ `roslaunch rviz rviz`

File Panels Help



Interact



Move Camera



Select



Focus Camera



Measure



2D Pose Estimate



2D Nav Goal



Publish Point



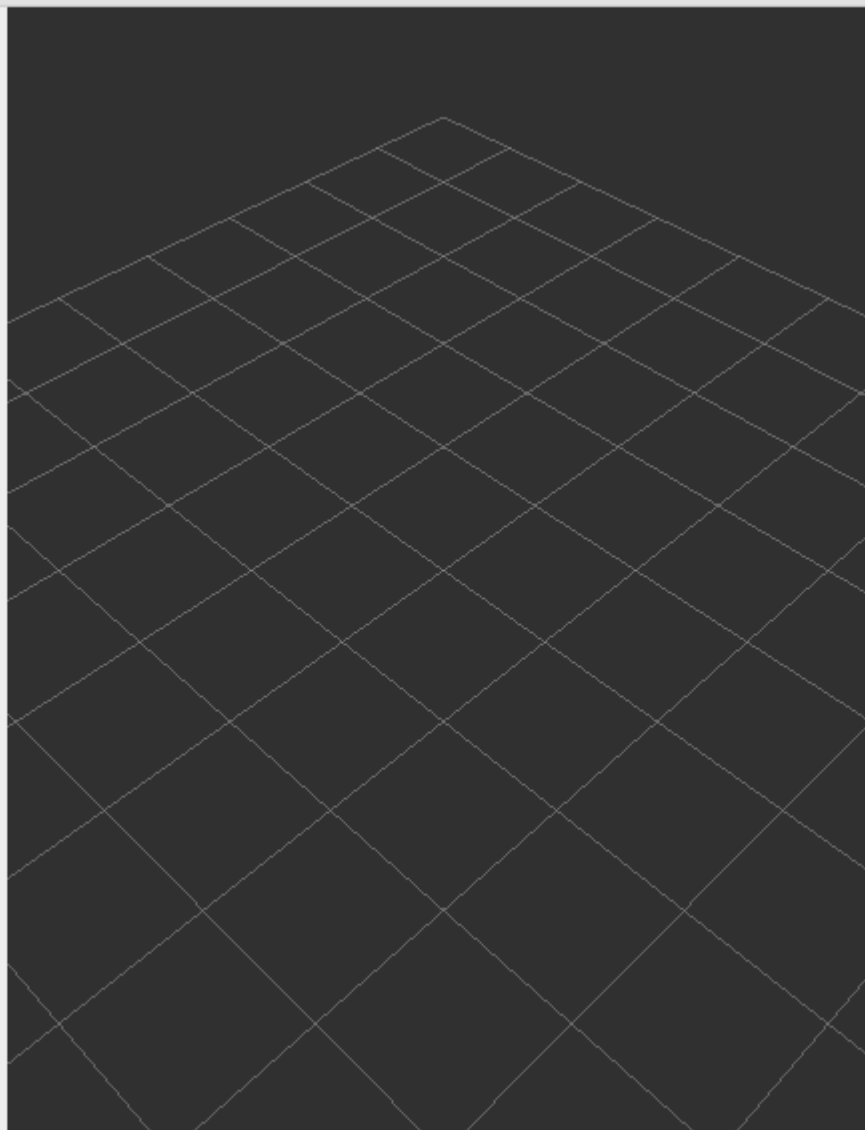
Displays

- Global Opt...
 - Fixed Frame map
 - Background ... 48; 48; 48
 - Frame Rate 30
- Global Sta...
 - Fixed Fr... No tf data. Act...
- Grid ☒

Add

Remove

Rename



Views

Type: Orbit (rviz) Zero

- Current View Orbit (rviz)
 - Near Clip D... 0.01
 - Target Frame <Fixed Frame>
 - Distance 10
 - Yaw 0.785398
 - Pitch 0.785398
 - Focal Point 0; 0; 0

Save

Remove

Rename

Time

ROS Time: 1375319247.53 ROS Elapsed: 70.67 Wall Time: 1375319247.56 Wall Elapsed: 70.67 ☐ Experimental

Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

30 fps

File Panels Help

Interact

Move Camera

Select

Focus Camera

Measure

2D Pose Estimate

2D Nav Goal

Publish Point

+

-

Displays

Global Opt...

Fixed Frame map

Background ... 48; 48; 48

Frame Rate 30

Global Sta...

Fixed Fr... No tf data. Act...

Grid ☒

Add

Remove

Rename

Views

Type: Orbit (rviz)

Zero

Current View Orbit (rviz)

Near Clip D...	0.01
Target Frame	<Fixed Frame>
Distance	10
Yaw	0.785398
Pitch	0.785398
Focal Point	0; 0; 0

Save

Remove

Rename

Time

ROS Time: 1375319247.53

ROS Elapsed: 70.67

Wall Time: 1375319247.56













Wall Elapsed: 70.67

☐ Experimental

Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

30 fps

Display Type

- ▼  rviz
 -  Axes
 -  Camera
 -  DepthCloud
 -  Effort
 -  Grid
 -  GridCells
 -  Group
 -  Image
 -  InteractiveMarkers
 -  LaserScan
 -  Map






















Description:

Display Name

Cancel

OK

Display Type

- ▼  rviz
 -  Axes
 -  Camera
 -  DepthCloud
 -  Effort
 -  Grid
 -  GridCells
 -  Group
 -  Image
 -  InteractiveMarkers
 -  **LaserScan**
 -  Map
 -  Marker
 -  MarkerArray
 -  Odometry
 -  Path
 -  PointCloud
 -  PointCloud2
 -  PointStamped
 -  Polygon
 -  Pose

Description:

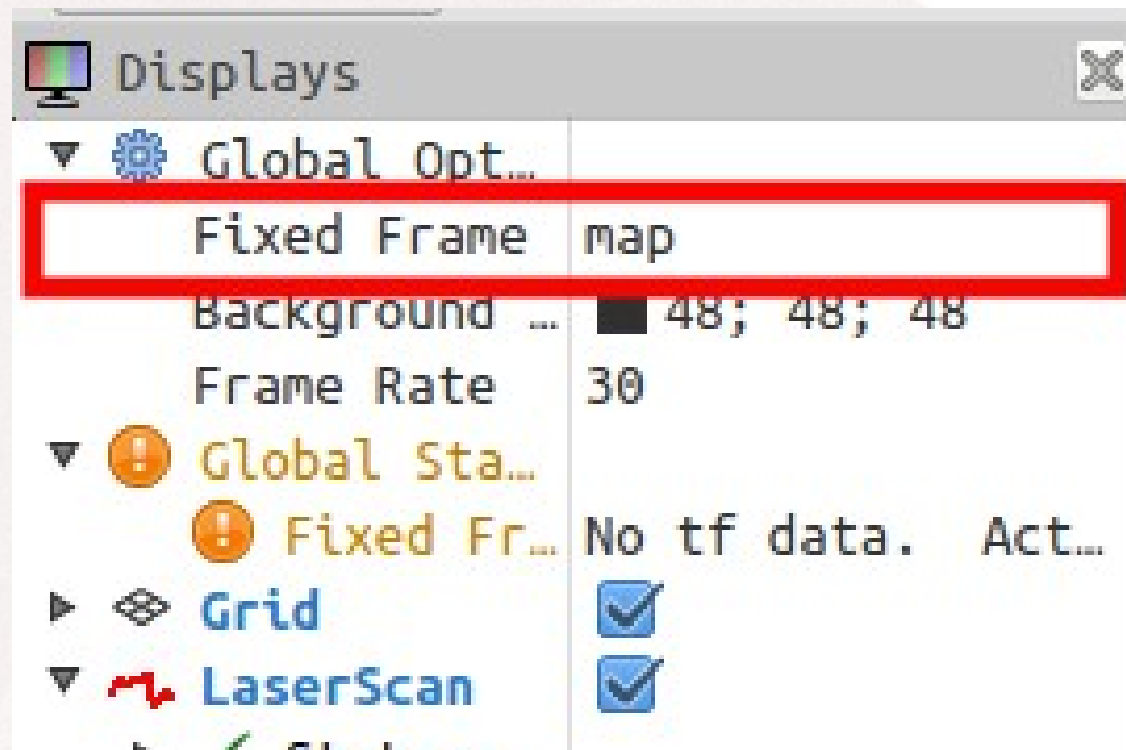
Displays the data from a `sensor_msgs::LaserScan` message as points in the world, drawn as points, billboards, or cubes.
[More Information.](#)

Display Name

■ バグファイルの再生

→ `rosbag play urg_data.bag`

frameの設定が必要









送信データ確認

```
rostopic echo  
  /hokuyo_node/parameter_descriptions
```

↑ 一行で

→ frame_id が対応する名前

▼  Global Opt...	
Fixed Frame	laser
Background ...	■ 48; 48; 48
Frame Rate	30
▼  Global Sta...	
 Fixed Fr...	No tf data. Act...
▶  Grid	<input checked="" type="checkbox"/>
▼  LaserScan	<input checked="" type="checkbox"/>
▶  Status: ...	
Topic	/scan
Selectable	<input checked="" type="checkbox"/>
Style	Flat Squares
Size (m)	0.05
Alpha	1

トピックを送信する
プログラムを
書いてみよう

パッケージ作成

```
catkin_create_pkg exercise  
roscpp visualization_msgs tf
```

catkin_create_pkg exercise
roscpp visualization_msgs tf

わかりやすい^[要出典] プログラムの流れ

1. 初期設定
2. 送信データ作成
3. 送信

1. 初期設定

2. 送信データ作成

3. 送信

1. 初期設定

使用するヘッダファイル

```
#include <cmath>
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <visualization_msgs/Marker.h>
```


1. 初期設定

```
ros::init(argc, argv, "test");  
// ノード作成  
ros::NodeHandle n;  
// publisher(データ送信部)作成  
ros::Publisher marker_pub =  
    n.advertise<visualization_msgs::Marker>  
        ("visualization_marker", 10);  
// データ送信周期設定 (秒間30回)  
ros::Rate r(30);
```

1. 初期設定

```
// 送信するメッセージの格納先  
visualization_msgs::Marker line_strip;  
// frame_id (URGのlaser) 好きな名前で  
line_strip.header.frame_id = "practice";  
// ネームスペース, これも好きな名前で  
line_strip.ns = "practice";  
line_strip.id = 1;
```

1. 初期設定

```
// マーカーのタイプを連続線に
line_strip.type =
    visualization_msgs::Marker::LINE_STRIP;
line_strip.action =
    visualization_msgs::Marker::ADD;
// 線の太さ
line_strip.scale.x = 0.1;
// 色の設定（青）
line_strip.color.b = 1.0;
line_strip.color.a = 1.0;
```


1. 初期設定

2. 送信データ作成

3. 送信

2. 送信データ作成

// 現在の時刻を設定

```
line_strip.header.stamp = ros::Time::now();
```

// 描画する座標の姿勢を指定

```
float phai = 0.0;
```

```
line_strip.pose.orientation.z = sin(phai/2);
```

```
line_strip.pose.orientation.w = cos(phai/2);
```

2. 送信データ作成

```
for (int i=0; i<=10; ++i) {  
    // 星型の各頂点を計算  
    float radius = i%2 ? 6.0 : 3.0;  
    float theta =  
        (72.0*(i/2) + (i%2)*36.0)*M_PI/180.0;  
    geometry_msgs::Point vertex;  
    vertex.x = radius*cos(theta);  
    vertex.y = radius*sin(theta);  
    // データを格納  
    line_strip.points.push_back(vertex);  
}
```


1. 初期設定

2. 送信データ作成

3. 送信

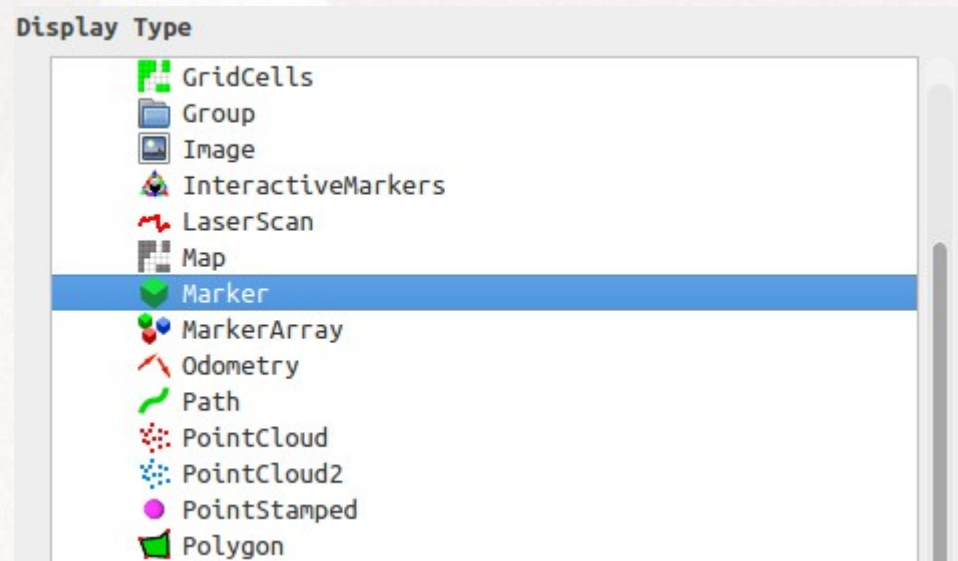
3. 送信

```
// データ送信  
marker_pub.publish(line_strip);
```

rvizで見てみよう

frame_id → practice

Markerを追加



ループしてみよう

```
while (ros::ok()) {  
    // 送信データ作成部(2)  
    // 送信(3)  
    // 今回送ったデータをクリア  
    line_strip.points.clear();  
    // 図形を回転させる  
    phai += 0.05;  
    if (phai > 2*M_PI) phai -= 2*M_PI;  
    // 周期のための待ち合わせ  
    r.sleep();  
}
```


この章 → rvizを通してパブリッシャーの作り方を学んだ

次の章 → pclを通してサブスクライバーについて学習する

第四章： 3D点群処理

ROS上でのPointCloudLibrary(PCL)の扱い方を説明

*PCL*とは

- 3D点群データ処理を集めたオープンソースライブラリ
 - Kinectなどの安価な3Dセンサの登場により注目を集めた

データ形式

- PCL専用の点群データ形式(.pcd)を用いる

演習2

1) Xtionから取得したデータを3次元空間として表示せよ

/camera/depth_registered/pointsを
収集したバグファイルを配布

プログラムの編集

■roscd パッケージ名 ファイル名

■roscd cloud_view xtion_cloud_view.cpp

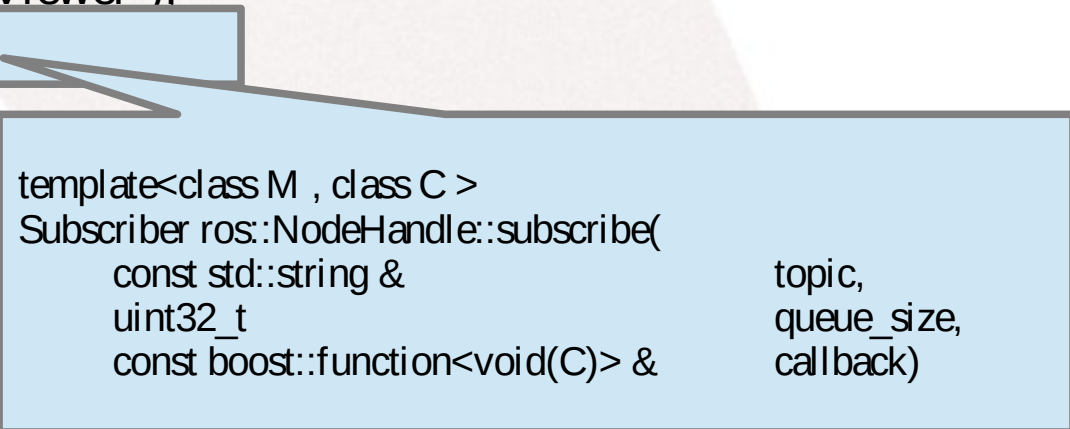
サブスクライバーの設定は？

```
class RosCloudViewer{
    pcl::visualization::CloudViewer m_viewer;
    ros::Subscriber m_pcSub;

public:
    typedef pcl::PointCloud<pcl::PointXYZRGB> PointCloud;
    RosCloudViewer(ros::NodeHandle &node) :
        m_viewer("RGB-D Sensor Cloud Viewer"),
        m_pcSub(?)
    {
    }

    void run(){
        ros::spin();
    }

private:
    void receivePointCloudCb(const PointCloud::ConstPtr &msg){
        m_viewer.showCloud(msg->makeShared());
    }
};
```



```
template<class M , class C >
Subscriber ros::NodeHandle::subscribe(
    const std::string &                topic,
    uint32_t                        queue_size,
    const boost::function<void(C)> &    callback)
```

サブスクライバーの設定は？

```
class RosCloudViewer{
    pcl::visualization::CloudViewer m_viewer;
    ros::Subscriber m_pcSub;

public:
    typedef pcl::PointCloud<pcl::PointXYZRGB> PointCloud;
    RosCloudViewer(ros::NodeHandle &node) :
        m_viewer("RGB-D Sensor Cloud Viewer"),
        m_pcSub(node.subscribe<PointCloud>("points", 1,
            boost::bind(&RosCloudViewer::receivePointCloudCb, this, boost::lambda::_1)))
    {
    }

    void run(){
        ros::spin();
    }

private:
    void receivePointCloudCb(const PointCloud::ConstPtr &msg){
        m_viewer.showCloud(msg->makeShared());
    }

};
```


ROSの初期化方法は？

```
int main(int argc, char *argv[]){
```

?

```
  ros::NodeHandle node;  
  RosCloudViewer rViewer(node);
```

```
  rViewer.run();
```

```
  return 0;
```

```
}
```

第三章でノード作成にはどの関数を呼び出したのかを思い出そう

プログラムのビルド

- `cd ~/ros_pcl_tutorial/tutorial_ws`

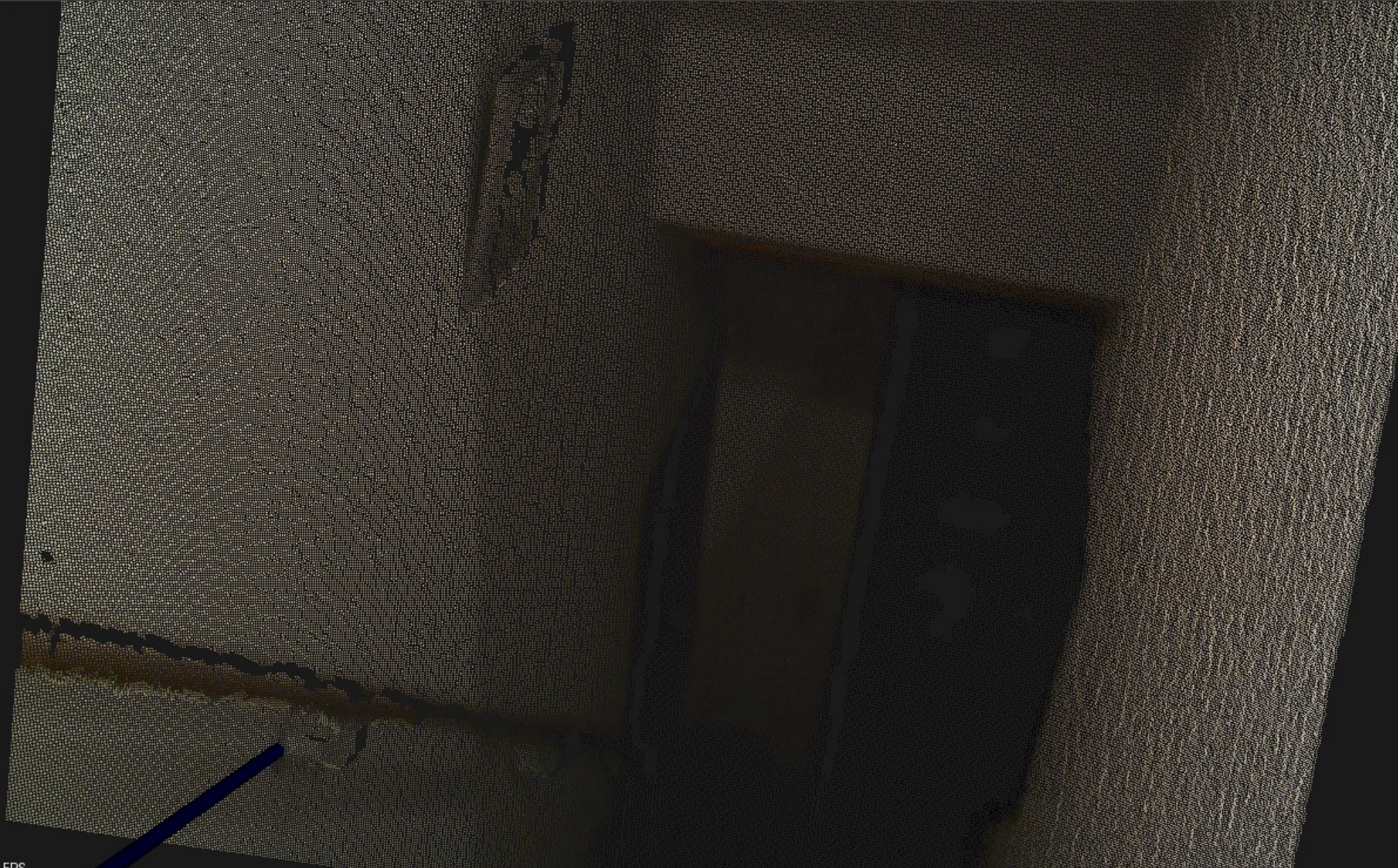
- `catkin_make`

プログラムの実行

- `roscore`

- `roslaunch cloud_view xtion_cloud_view`
 `points:=/camera/depth_registered/points`

- `rosbag play point_cloud.bag`



以上でROS勉強会<入門編>は終了です
ご清聴ありがとうございました