
「ROS」の基礎と ROS 2プログラミングの実践

5. 新機能の利用と実践

高瀬 英希

(京都大学／JSTさきがけ)

takase@i.kyoto-u.ac.jp



プログラム・スケジュール

5. 新機能の利用と実践 [day2 13:30-15:00]

- ROS 2の新機能の利用 [実習]
- ROS 2による実ロボットの操作とシミュレーション

ROS 2の新機能

- DDSの切り替え
- QoSとLifecycleの制御
- ros1bridgeの利用
- micro-ROS

- TurtleBot3
 - ROS 2による実ロボットの操作
 - Gazeboによるシミュレーション

ROS 2対応DDSの実装

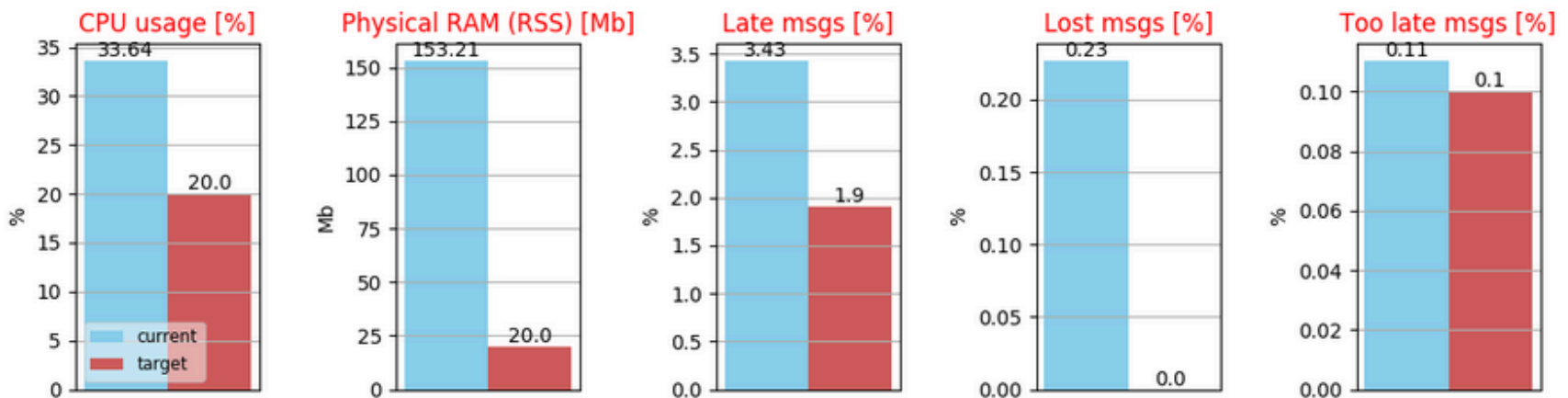
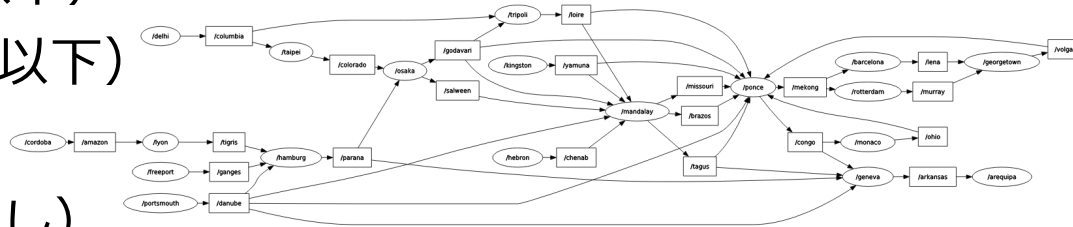
REP-2000 を基に
作成・整理

Middleware Library	Provider	Support	ライセンス・備考等
rmw_fastrtps_cpp	eProsima Fast-RTPS	Tier 1	Apache 2.0, ROS 2デフォルト, 軽量実装
rmw_connext_cpp	RTI Connex	Tier 1	商用／研究(機能制限)ライセンス, No.1 ベンダ, PF/Arch.サポートは限定的
rmw_cyclonedds_cpp	Eclipse Cyclone DDS	Tier 2	Eclipse Public License 2.0(Open), 高性能・高信頼な評価結果
rmw_opensplice_cpp	ADLINK OpenSplice	Tier 2	商用／オープン(機能制限, LGPLv6.4), PFサポートは限定的, Foxyで対象外
rmw_fastrtps_ dynamic_cpp	eProsima Fast-RTPS	Tier 2	Apache 2.0, 型の実行時の解釈・変換 (type introspection)をサポート

- 実装機能やライセンス形態で複数のDDSから通信層を選択できる
- 異種DDSを選択したノード間で通信できる（rmw層で抽象化）
 - Tier 1: Open Robotics公式の手厚いサポート
 - Tier 2: 公式だが限定的なサポート

ROS2 for consumer robotics: the iRobot use-case

- 対象 : Raspberry Pi 2 (900MHz A7x4, 1GB RAM)
 - CPU使用率 (目標20%以下)
 - RAM使用量 (目標20MB以下)
 - 通信レイテンシ
 - メッセージ (目標欠落無し)
- Fast-RTPSの評価結果



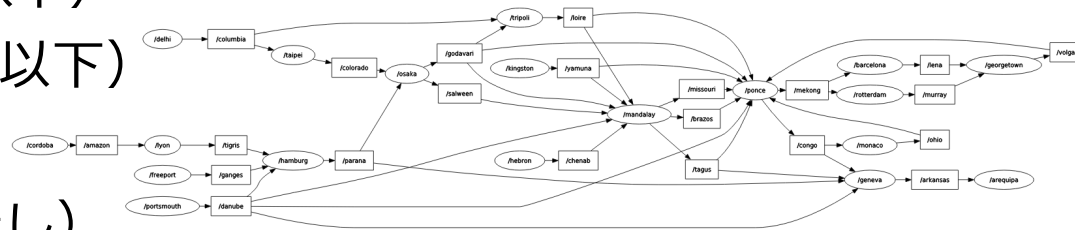
https://roscon.ros.org/2019/talks/roscon2019_irobot_usecase.pdf

<https://www.infoq.com/jp/news/2019/12/ros2-linux-embedded-platform/>

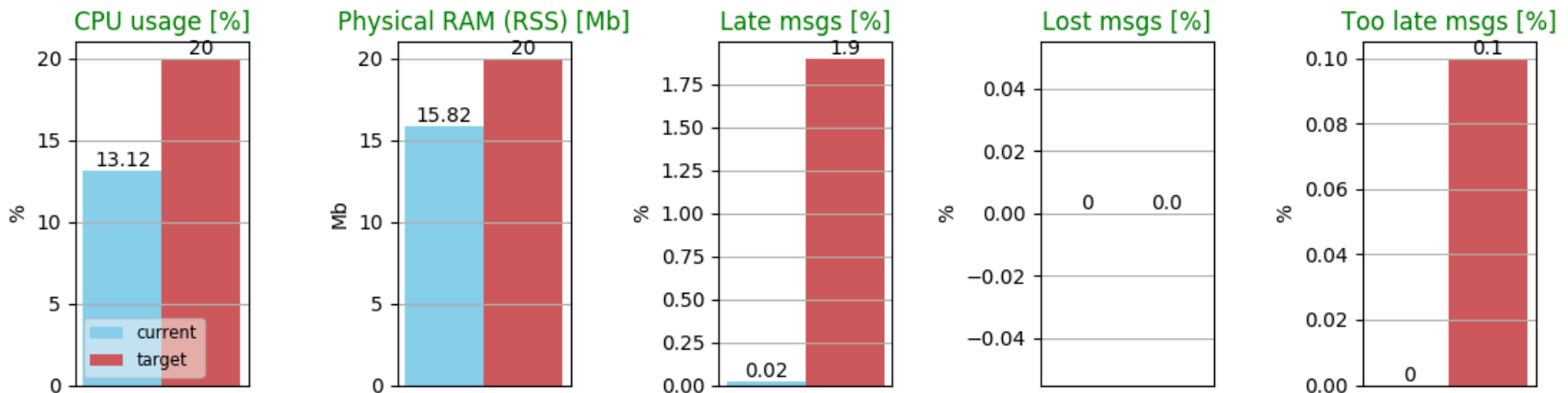
ROS2 for consumer robotics: the iRobot use-case

- 対象 : Raspberry Pi 2 (900MHz A7x4, 1GB RAM)

- CPU使用率 (目標20%以下)
- RAM使用量 (目標20MB以下)
- 通信レイテンシ
- メッセージ (目標欠落無し)



- CycloneDDSの評価結果



https://roscon.ros.org/2019/talks/roscon2019_irobot_usecase.pdf

<https://www.infoq.com/jp/news/2019/12/ros2-linux-embedded-platform/>

CycloneDDSへの切り替え

- インストール

```
# DDS本体と対応するRMWのインストール
$ sudo apt install ros-dashing-rmw-cyclonedds-cpp
# 環境設定
$ export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
```

- 異種DDSのノード間で通信

```
# 出版者側はCycloneDDS
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
$ ros2 run pubsub_topic talker
```

```
# 購読者側はFast-RTPS (デフォルト)
$ ros2setup
$ . ~/ros2_ws/install/local_setup.bash
$ ros2 run pubsub_topic listener
```

QoS Control

- 通信経路の品質を指定する
 - Pub/Sub node間で互換性が必要
(SubのQoSプロファイルのほうが厳しい必要がある)
- 指定できるパラメータの例 (下線はデフォルト設定)
 - History (履歴): Keep last (任意の値数を保持) or Keep all
 - Depth (深さ): Size of the queue (Keep lastの個数 10)
 - Reliability (信頼性): Best effort or Reliable (リトライ試行)
 - Durability (耐久性): Transient local or Volatile (遅延非許容)

Profile	History policy	Reliability	Durability
Default	Keep last 10	Reliable	Volatile
Services	Keep last 10	Reliable	Volatile
Sensor data	keep last 5	Best effort	Volatile
Parameters	keep last 1000	Reliable	Volatile

詳細 : `rmw_qos_profile_t`

- メンバ (太字はDashingより・下線はデフォルト設定)
 - history (履歴): Keep last (任意の値数を保持) or Keep all
 - depth (深さ): Size of the queue (Keep lastの個数 10)
 - reliability (信頼性): Best effort or Reliable (リトライ試行)
 - durability (耐久性): Transient local or Volatile (遅延非許容)
 - **deadline** (許容更新周期): Period ({0, 0})
 - **lifespan** (存在期間): Age ({0, 0})
 - **liveliness** (生存確認): Default, Automatic or Manual
 - **liveliness_lease_duration** (保証時間): Time ({0, 0})
 - **avoid_ros_namespace_conventions**:
ROS固有の名前空間をDDSのものに変換するか true or false

詳細：QoSプロファイルの互換性

- Reliability (信頼性)

Publisher	Subscriber	Connection	Result
Best effort	Best effort	Yes	Best effort
Best effort	Reliable	No	--
Reliable	Best effort	Yes	Best effort
Reliable	Reliable	Yes	Reliable

- Durability (信頼性)

Publisher	Subscriber	Connection	Result
Volatile	Volatile	Yes	Volatile
Volatile	Transient local	No	--
Transient local	Volatile	Yes	Volatile
Transient local	Transient local	Yes	Transient local

QoS Controlの実装例

```
// historyサイズを指定（残りのプロファイルはデフォルト）
pub = node->create_publisher<MsgT>("chatter", 10);
sub = node->create_subscription<MsgT>("chatter", 10, chatterCallback);
pub = node->create_publisher<MsgT>("chatter", rclcpp::QoS(10));

// 定義済みプロファイルを使用
pub = node->create_publisher<MsgT>("chatter", rclcpp::ServicesQoS());
sub = node->create_subscription<MsgT>("chatter",
                                     rclcpp::SensorDataQoS(), chatterCallback);

// 定義済みプロファイルをカスタマイズ
rclcpp::QoS qos(rclcpp::KeepLast(10), rmw_qos_profile_sensor_data);
pub = node->create_publisher<MsgT>("chatter", qos, auto);
```

例題 : QoS for image_tools

```
$ ros2setup
$ ros2 run image_tools showimage [--option]
```

```
$ ros2setup
# -b はダミー動画を表示
$ ros2 run image_tools cam2image -b
```

```
# 送信パケットを一定の割合で廃棄
$ sudo tc qdisc add dev lo root netem loss 5%
~~~
# 設定をもとに戻す
$ sudo tc qdisc delete dev lo root netem loss 5%
```

<https://index.ros.org/doc/ros2/Tutorials/Quality-of-Service/>

```
$ ros2 run image_tools showimage -- -h
Right after init
```

Usage:

- h: This message.
- r: Reliability QoS setting:
 - 0 - best effort
 - 1 - reliable (default)
- d: Depth of the queue: only honored if used together with 'keep last'. 10 (default)
- k: History QoS setting:
 - 0 - only store up to N samples, configurable via the queue depth (default)
 - 1 - keep all the samples
- s: Camera stream:
 - 0 - Do not show the camera stream
 - 1 - Show the camera stream
- t TOPIC: use topic TOPIC instead of the default

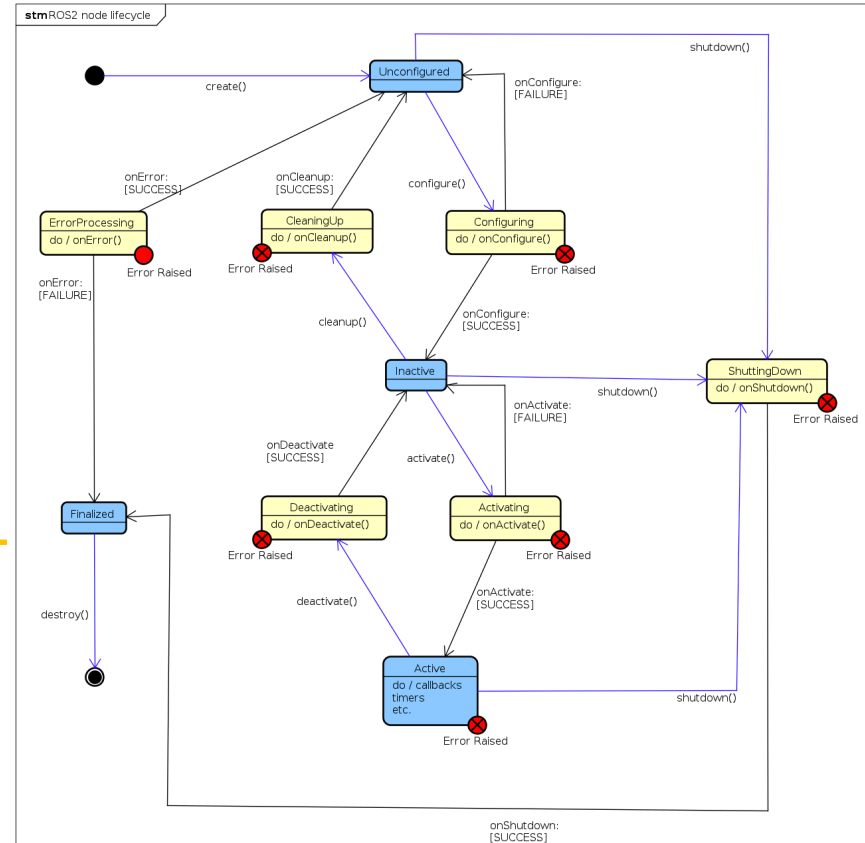
例題 : Lifecycle

```
$ ros2setup  
$ ros2 run lifecycle lifecycle_talker
```

```
$ ros2setup  
$ ros2 run lifecycle lifecycle_listener
```

```
$ ros2setup  
$ ros2 run lifecycle lifecycle_service_client
```

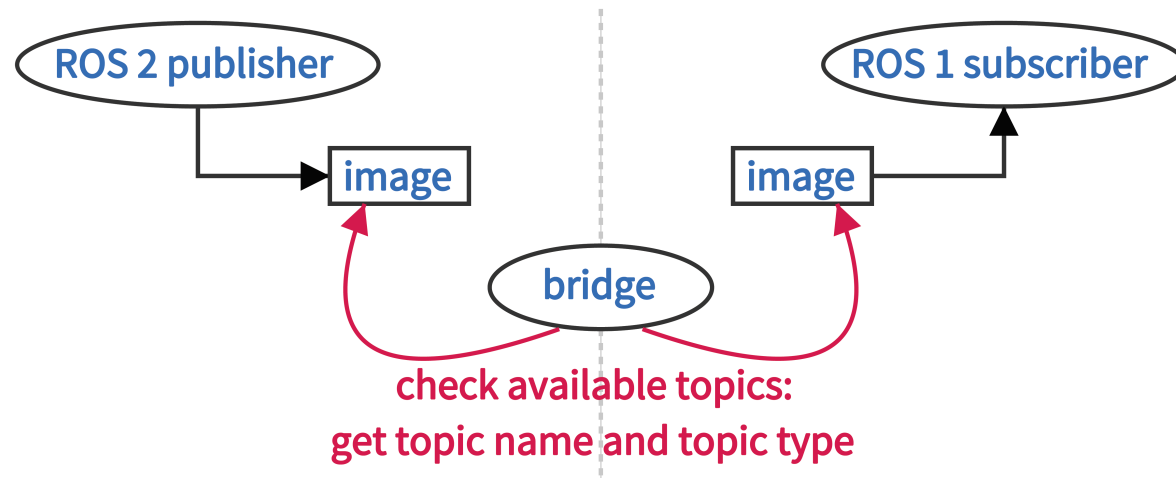
```
$ ros2 lifecycle get /lc_talker  
$ ros2 lifecycle set /lc_talker configure  
$ ros2 service call /lc_talker/get_state ¥  
  lifecycle_msgs/GetState  
$ ros2 service call /lc_talker/change_state ¥  
  lifecycle_msgs/ChangeState ¥  
  "{transition: {id: 1}}"  
$ ros2 msg show lifecycle_msgs/msg/Transition
```



<https://index.ros.org/p/lifecycle/>

ros1bridge

- ROS 1とROS 2を共存させる仕組み
- bridgeノードが topic/service の“橋渡し”をする
 - dynamic_bridge: 双方に同名のトピック／サービスが存在する時のみブリッジを作成
 - static_bridge: 常にブリッジを作成



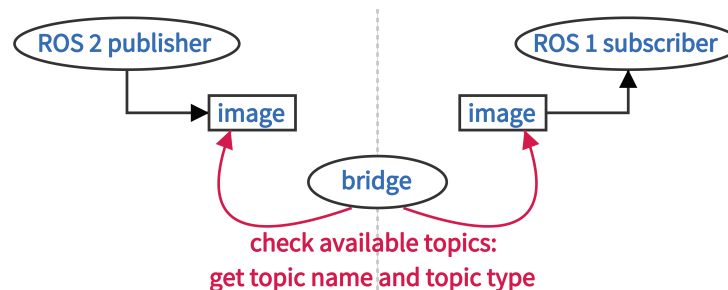
topic の ros1bridge

```
$ ros1setup
$ ros2setup
$ export ROS_MASTER_URI=http://localhost:11311
$ ros2 run ros1_bridge dynamic_bridge
```

```
$ ros1setup
$ roscore
```

```
$ . ~/ros2_ws/install/setup.bash
$ ros2 run pubsub_topic talker
~~~
$ ros2 run pubsub_topic listener
~~~
$ ros2 run image_tools cam2image
```

```
$ source ~/catkin_ws/devel/setup.bash
$ rosrunc pubsub_topic listener
~~~
$ rosrunc pubsub_topic talker
~~~
$ rqt_image_view /image
```



独自定義msg/srvのブリッジ

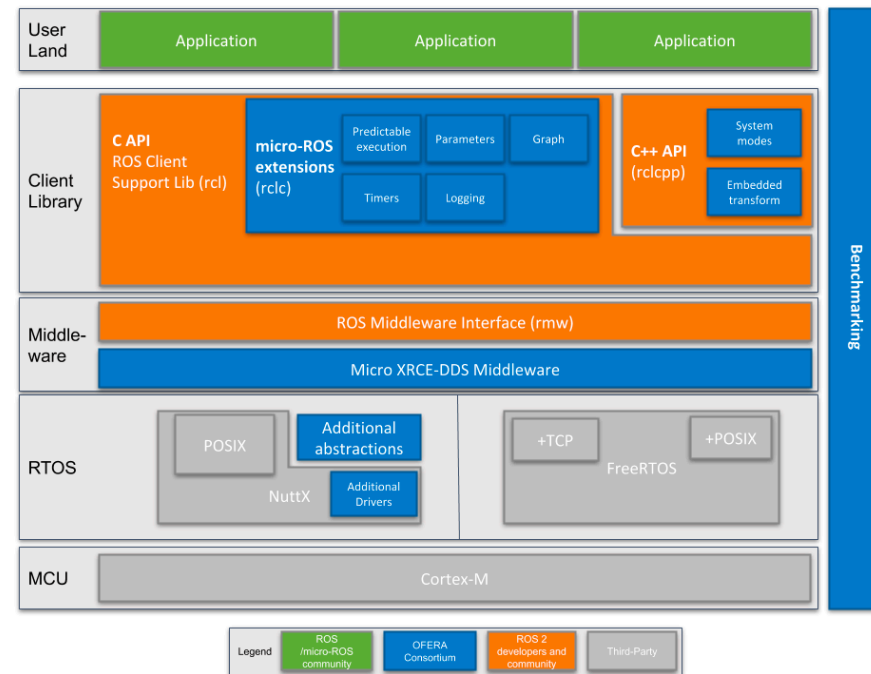
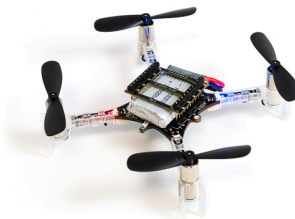
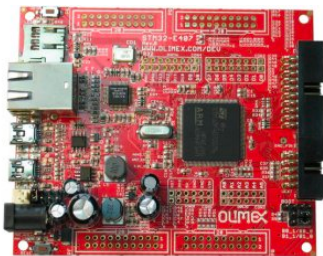
- YAMLによるブリッジルールの記述, および, `ros1_bridge` 自体のソースビルドが必要となる
- 参考情報 :
 - [公式の解説 GitHubのdoc/index.rst](#)
 - `ros_study_types` での `Human.msg` への適用例
 - ✓ [Git差分ログ](#)
 - ✓ [詳細な解説 \(Evernote\)](#)

micro-ROS ROS

- ROS 2をマイコンで動作させる試み
- RTOSとDDS-XRCEによってROS 2通信機能を提供
 - DDS for eXtremely Resource Constrained Environments

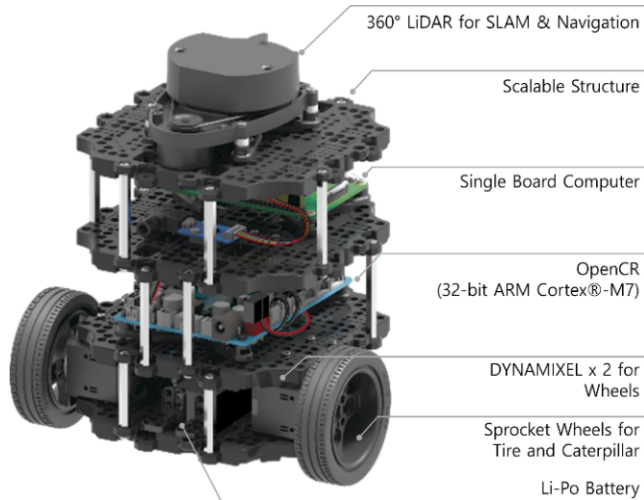
• サポート対象

RTOS	Platform
NuttX	Olimex STM32-E407, STM32F4Discovery
FreeRTOS	Crazyfile 2.1
FreeRTOS	Olimex STM32-E407

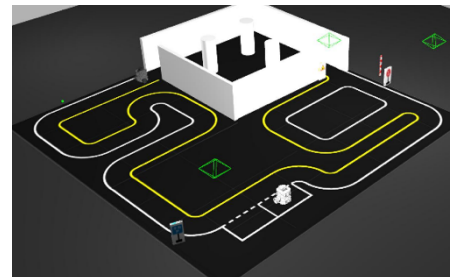
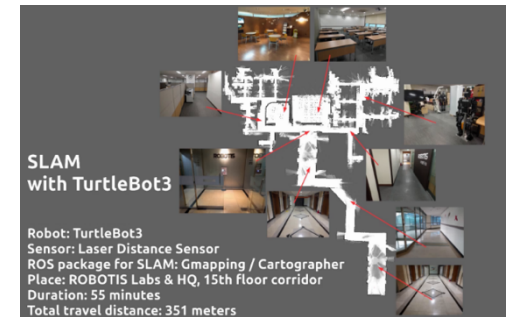


学習用ロボット : TurtleBot3

- ROS公式の研究・教育用ロボット組立キット
 - テーブルトップでROSやSLAMの学習が出来る
 - OSRF (Open Robotics)からのリクエストで誕生
 - 改造の自由度が高く、ROS対応ロボット作成が容易



- **ROS Education**
- **SLAM**
- **Navigation**
- **Autonomous Driving**



TurtleBot3によるデモ

- 環境設定・インストール
 - [install tb3.md](#) : PC側のみの手順（実習PCに構築済）
 - 実機側の設定は [公式ページ](#) を参照のこと
- 実施項目
 - rviz2: ロボット状態とセンサ値の可視化
 - teleop: 移動方向の操作
 - cartographer: 自己位置推定と地図生成（SLAM）
 - navigation2: 地図ベースによる経路計画の決定
 - **Gazebo: PC上のみでのシミュレーション**

TurtleBot3によるデモ

- 実機の実機操作

- 実機でのROS 2システムの立ち上げ

```
$ export TURTLEBOT3_MODEL=burger  
$ ros2 launch turtlebot3_bringup robot.launch.py
```

- Rviz2によるロボット状態とセンサ値の可視化

```
$ tb3setup  
$ ros2 launch turtlebot3_bringup rviz2.launch.py
```

- キーボード入力による操作

```
$ tb3setup  
$ ros2 run turtlebot3_teleop teleop_keyboard
```

TurtleBot3によるデモ

- 実機の実機操作

- SLAMによる地図生成 (Cartographer)

```
$ tb3setup  
$ ros2 launch turtlebot3_cartographer cartographer.launch.py
```

```
$ tb3setup  
$ ros2 run turtlebot3_teleop teleop_keyboard
```

```
$ tb3setup  
$ ros2 run nav2_map_server map_saver -f ~/map
```

- Navigation2による経路計画の決定と自律移動

```
$ tb3setup  
$ ros2 launch turtlebot3_navigation2 navigation2.launch.py map:=$HOME/map.yaml
```

TurtleBot3によるデモ

- Fake nodeの起動

```
$ tb3setup  
$ ros2 launch turtlebot3_fake_node turtlebot3_fake_node.launch.py
```

- Gazeboシミュレータでの操作

```
$ tb3setup  
$ ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

- Gazeboの起動後, 実機と同じ操作が可能