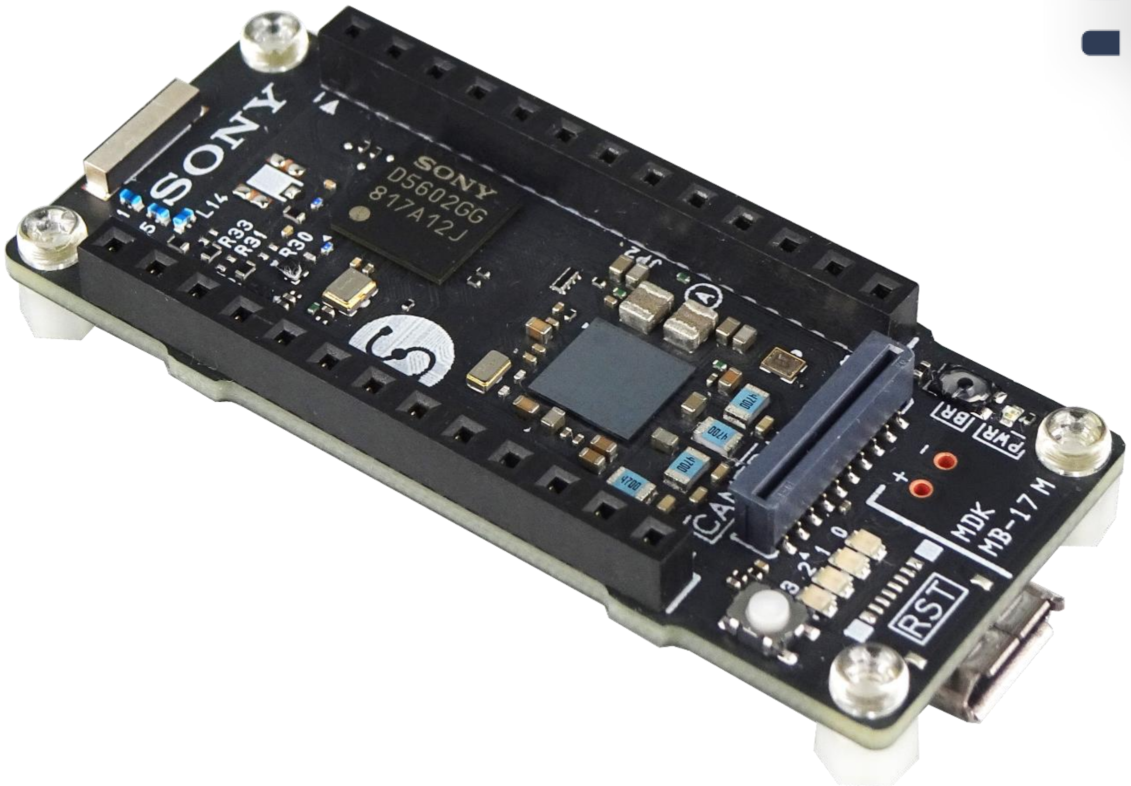
SPRESENSE™ とmicro-ROS で始めるロボットプログラミング

ソニーセミコンダクタソリューションズ(株)
太田　義則

本日のセミナーの流れ

- **micro-ROSとは？**
  - micro-ROSとROS2
  - micro-ROSとROS2の通信
- **Spresenseによる micro-ROS**
  - 環境セットアップ
    - ROS2 セットアップ
    - micro-ROS-agent セットアップ
    - Spresense/micro-ROS arduino セットアップ
  - micro-ROS の実装
    - micro-ROS publisher の実装
    - micro-ROS subscriber の実装
    - micro-ROS server の実装
    - micro-ROS client の実装
- **Spresense による TurtleBot の実装**
  - TurtleBot の動作仕様
  - TurtleBot の使用部材
  - TurtleBot の回路構成
  - TurtleBot の実装
  - TurtleBot の操作



µROS

micro-ROS puts ROS 2 onto microcontrollers

micro-ROS とは？

micro-ROS と ROS2

ROSとは？

ROS とは、Robot Operation System の略であり、広い意味ではロボット開発のための各種ツールのことを指します。ROS そのものは、2010年にリリースされた、ロボットとPC間で協調動作をするための通信システムを担うライブラリです。

ROS2とは？

ROS1 は研究用に作られたもので、単体のロボットとの通信しか意識されていませんでした。利用用途が広がるにつれて、複数台ロボットの制御や、マイクロコントローラへの対応、ロバストな通信を備えたものを要望する声が増え、それらを備えた ROS2 が2017年にリリースされました。

micro-ROS とは？

micro-ROS とは ROS2 とマイクロコントローラシステムをつなぐためのRTOS向けライブラリです。ホスト上で動作する ROS2 は、micro-ROS-agent というブリッジプログラムによって micro-ROS が実装されたマイクロコントローラと協調動作することができます。

micro-ROS と ROS2

ROS と ROS2 の違い

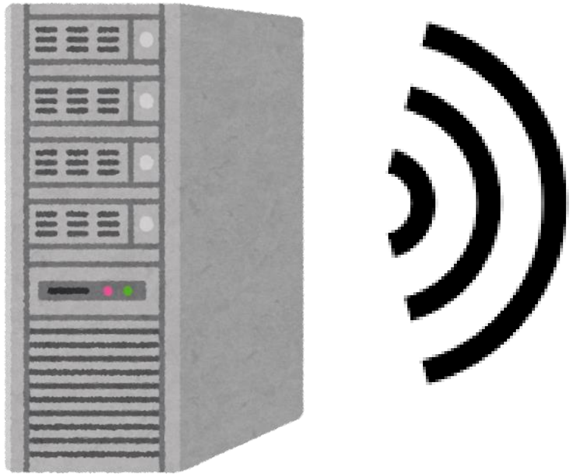
https://design.ros2.org/articles/why_ros2.html

	ROS	ROS2
Controllable robots	a single robot	Teams of multiple robots
Computing power requirements	workstation-class computational resources on board	Small embedded platforms (micro-ROS)
Real time	no real-time requirements	Real-time systems
Network environment	excellent network connectivity r equired	Non-ideal networks
Use case	applications in research, mostly academia	Production environments
Flexibility	maximum flexibility, with nothing prescribed or proscribed	Prescribed patterns for building and structuring systems

micro-ROS と ROS2

micro-ROS と ROS2 のアプリケーション

ROS2



複数の micro-ROS を協調制御

micro-ROS

micro-ROS

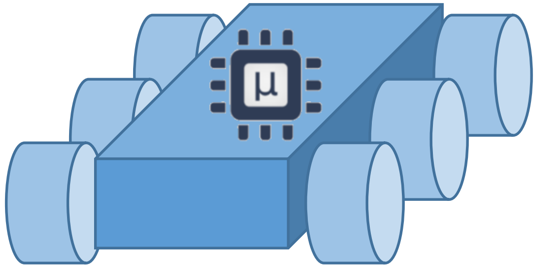
micro-ROS

micro-ROS と ROS2

micro-ROS と ROS2 のアプリケーション

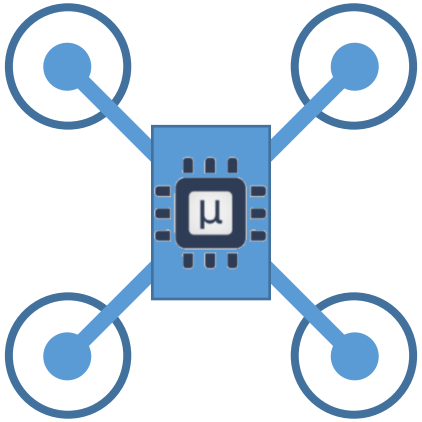
ROS2 は工場での自走搬送ロボットやロボットアーム、またドローンやカメラシステムへ応用されつつあります

AGV（自動搬送車）



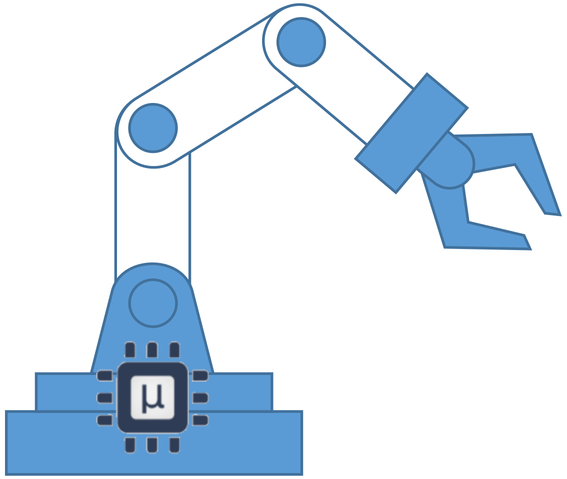
micro-ROS

ドローン



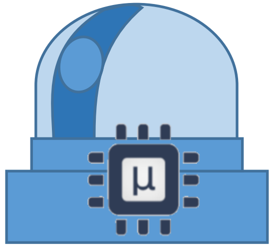
micro-ROS

ロボットアーム



micro-ROS

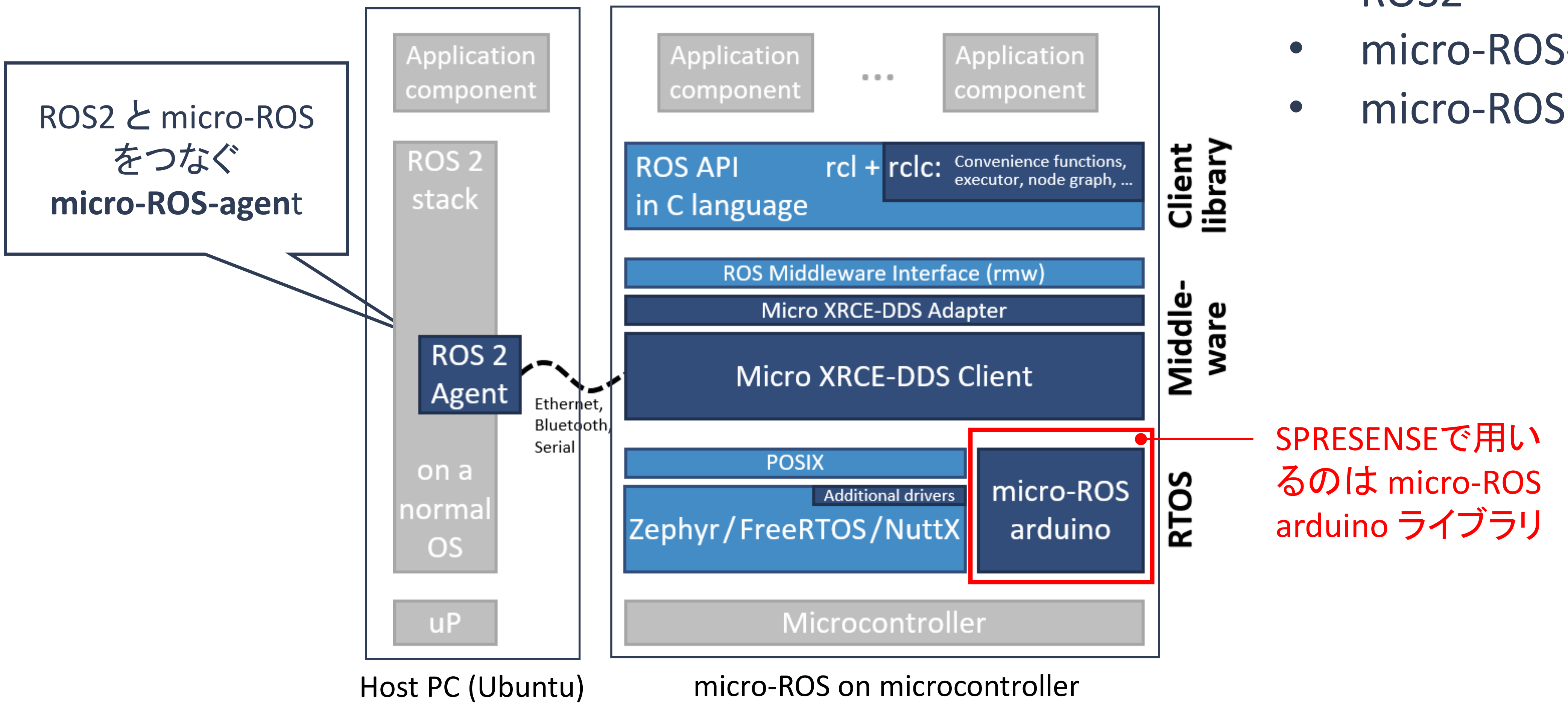
カメラシステム



micro-ROS

micro-ROS と ROS2

micro-ROS と ROS2 の構造



３つのコンポーネント

- ROS2
- micro-ROS-agent
- micro-ROS

micro-ROS と ROS2 の通信

<ROS2要素>

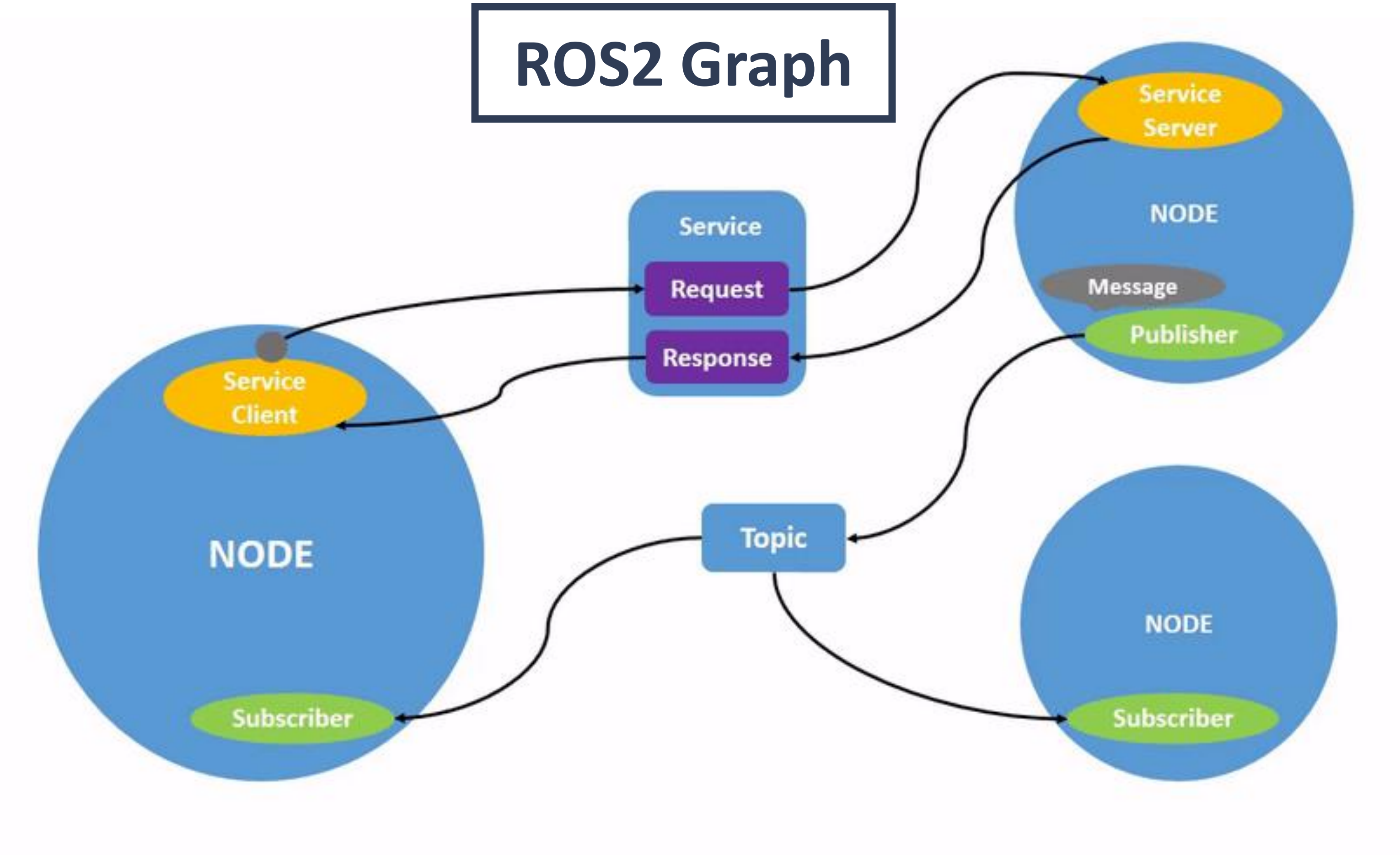
Node

Topic

- Subscriber
- Publisher

Service

- Service Client
- Service Server



ROS2グラフは、データを処理するROS2要素のネットワーク構造

micro-ROS と ROS2 の通信

<ROS2要素>

Node

Topic

Subscriber

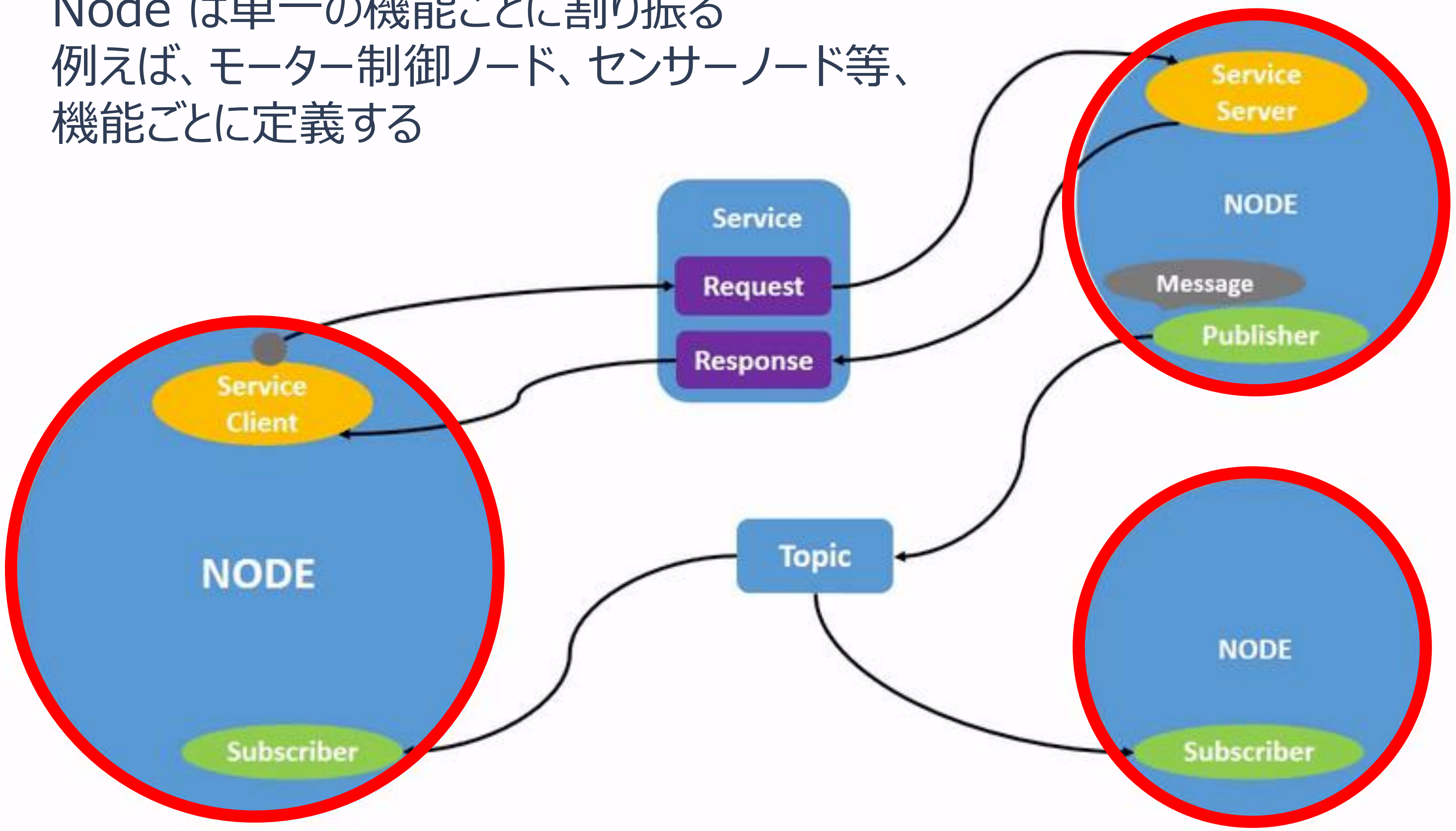
Publisher

Service

Service Client

Service Server

Node は単一の機能ごとに割り振る
例えば、モーター制御ノード、センサーノード等、
機能ごとに定義する



Node 同士は、Topic, Service 等を使って通信する

micro-ROS と ROS2 の通信

<ROS2要素>

Node

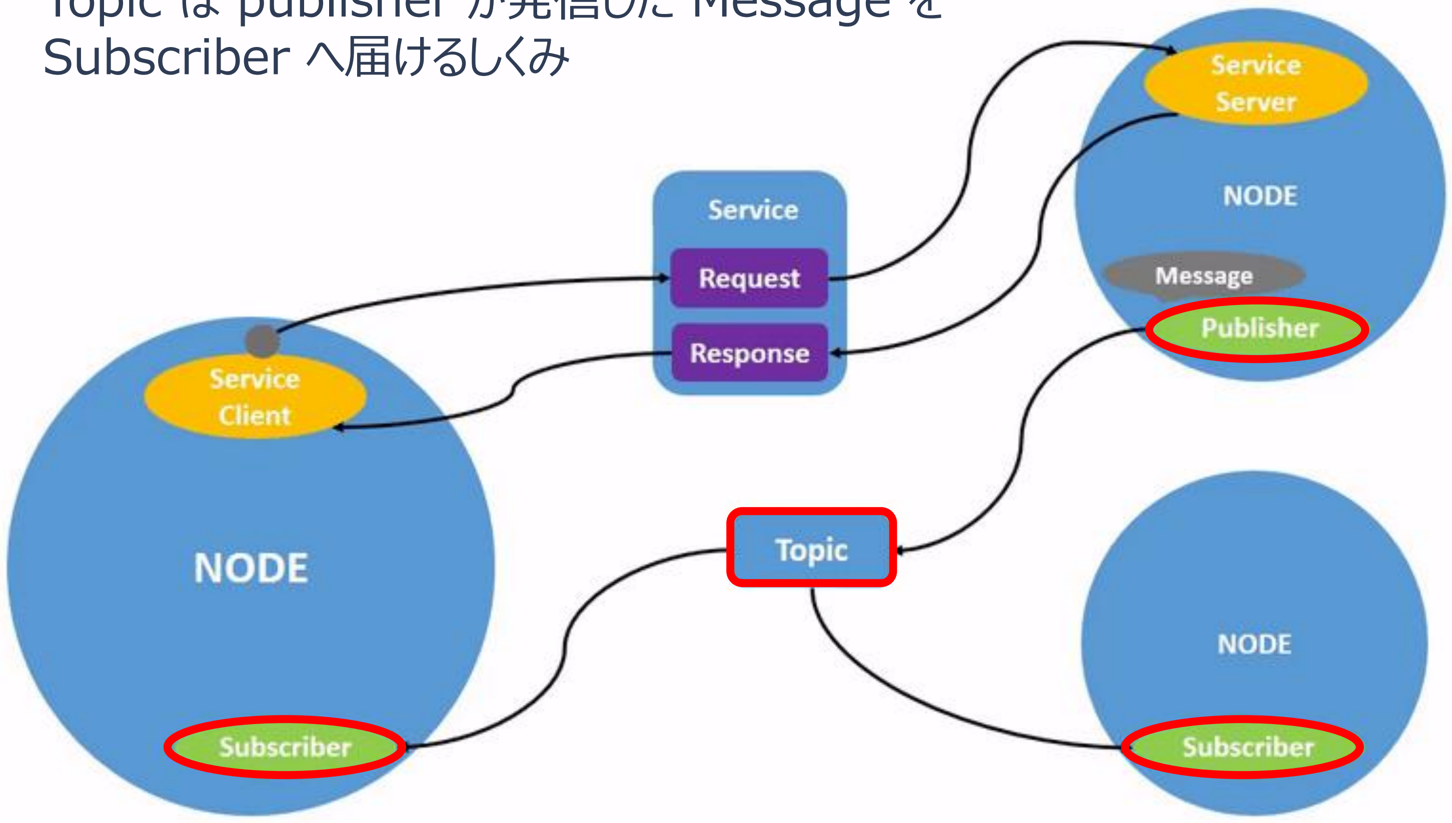
Topic

- Subscriber
- Publisher

Service

- Service Client
- Service Server

Topic は publisher が発信した Message を Subscriber へ届けるしくみ



Topic を Subscribe している全ての Subscriber に Message が伝達される

micro-ROS と ROS2 の通信

<ROS2要素>

Node

Topic

Subscriber

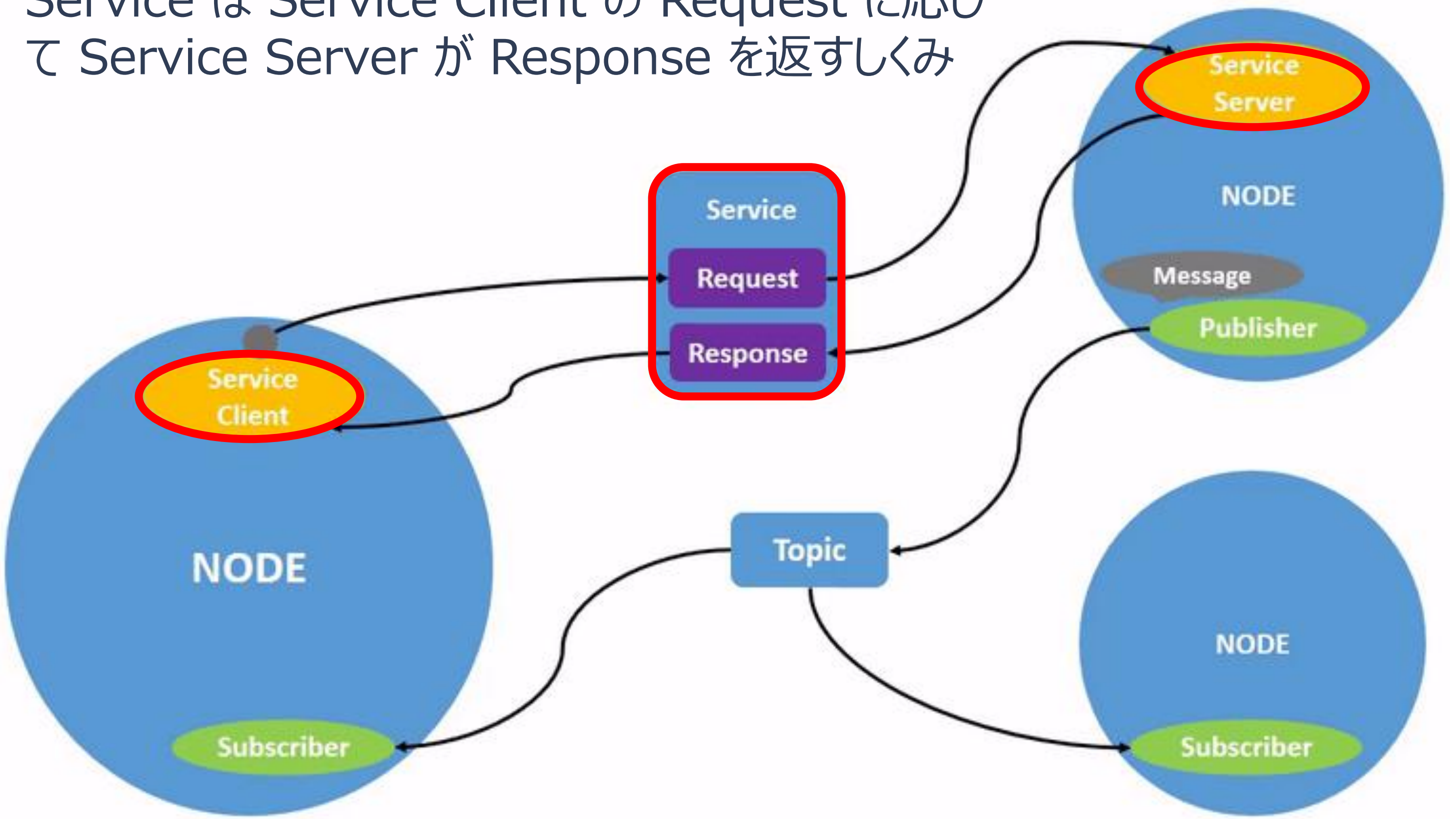
Publisher

Service

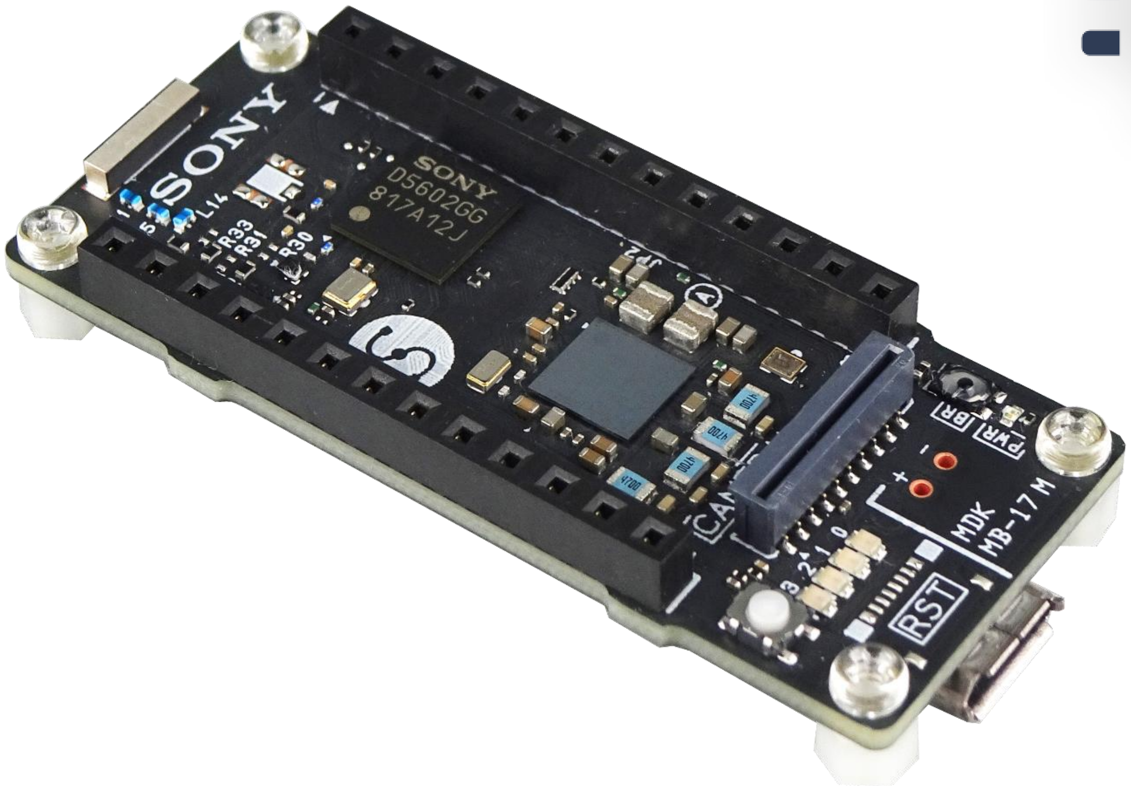
Service Client

Service Server

Service は Service Client の Request に応じて Service Server が Response を返すしくみ



Topic と異なり Service Client / Server の 1 対 1通信となるところが異なる



micro-ROS puts ROS 2 onto microcontrollers

Spresense による micro-ROS

Spresense の開発環境セットアップ

ROS2 セットアップ

ROS2は “humble”を使用します。humble は Ubuntu 22.04 TLS が対象です。Host PC には、Ubuntu 22.04 TLS をインストールされたものをお使いください。

micro-ROS-agent セットアップ

micro-ROS-agent は、コンパイルをしてもよいですが、簡単に扱うために Docker イメージを使います。セットアップの際に Docker をインストールする必要があります。

Spresense / micro-ROS-arduino セットアップ

Spresense 用の micro-ROS-arduino を GitHub に準備しました。ダウンロードする際に、ダウンロードしようとしているブランチが humble-spresense であることを確認してください。

Spresense の開発環境セットアップ

ROS2 セットアップ

Ubuntu 22.04TLS をインストールした PC で次のサイトを参考にしてインストールを行ってください
https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html

【STEP1】PPA (Personal Package Archive) に ROS のパッケージを追加

```
$ sudo apt update && sudo apt install curl gnupg lsb-release

$ sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg

$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(source /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

Spresense の開発環境セットアップ

ROS2 セットアップ

【STEP2】ROS2 パッケージをインストール

```
$ sudo apt update && sudo upgrade
$ sudo apt install ros-humble-desktop
$ sudo apt install ros-humble-ros-base
```

【STEP3】ROS2 の環境変数を登録

```
$ source /opt/ros/humble/setup.bash
```

Spresense の開発環境セットアップ

ROS2 セットアップ

【STEP4】動作確認

```
$    ros2 run demo_nodes_cpp talker
[INFO] [1659857287.397953074] [talker]: Publishing: 'Hello World: 1‘
[INFO] [1659857288.397938888] [talker]: Publishing: 'Hello World: 2‘
[INFO] [1659857289.397927192] [talker]: Publishing: 'Hello World: 3‘
[INFO] [1659857290.397927403] [talker]: Publishing: 'Hello World: 4‘
[INFO] [1659857291.397948089] [talker]: Publishing: 'Hello World: 5‘
[INFO] [1659857292.397923946] [talker]: Publishing: 'Hello World: 6'
```

【STEP4】別ターミナルを開いて次のコマンドを実行

```
$    source /opt/ros/humble/setup.bash && ros2 run demo_nodes_py listener
[INFO] [1659857290.424806294] [listener]: I heard: [Hello World: 4]
[INFO] [1659857291.399981669] [listener]: I heard: [Hello World: 5]
[INFO] [1659857292.400563587] [listener]: I heard: [Hello World: 6]
```

Spresense の開発環境セットアップ

micro-ROS-agent セットアップ

【STEP1】docker をインストール

```
$   sudo apt install docker docker-compose
```

Spresense の開発環境セットアップ

micro-ROS-agent セットアップ

【STEP2】Spresense を接続し、micro-ROS-agent の docker イメージをインストールし起動する

```
$   sudo docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-ros-agent:humble  serial   --dev
    /dev/ttyUSB0
humble: Pulling from microros/micro-ros-agent
   ... skip ...
4f4fb700ef54: Pull complete
63c52ba960a7: Pull complete
   ... skip ...
Digest: sha256:69e2b0d22a1e6cf33ca0a984a1ee7085133982b8bda1f9de305dc3a249a6405d
Status: Downloaded newer image for microros/micro-ros-agent:humble
[1659402842.881086] info    | TermiosAgentLinux.cpp | init                | running...        | fd: 3
[1659402842.881813] info    | Root.cpp              | set_verbose_level   | logger setup      | verbose_level: 4
```

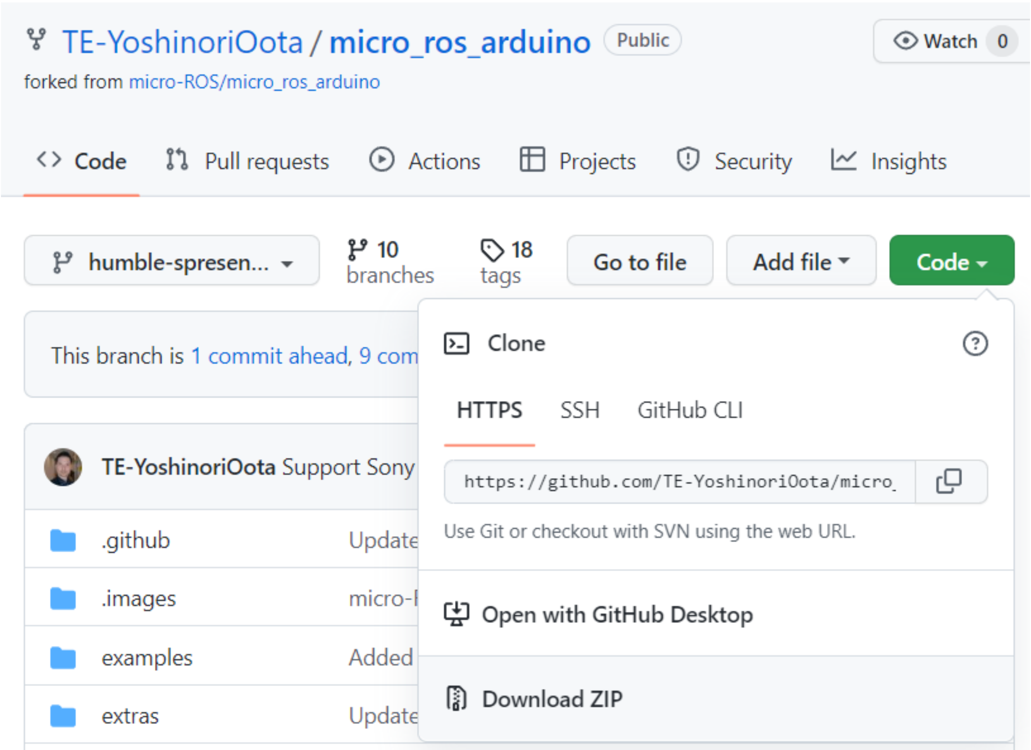
micro-ROS-agent の docker image のダウンロード

Spresense の開発環境セットアップ

うまくいかない場合は、"micro_ros_arduino-humble-spresense.zip" を解凍して Arduino のスケッチフォルダの"libraries"フォルダにフォルダー毎コピーしてください

Spresense / micro-ROS-arduino セットアップ

【STEP1】micro_ROS_arduino のダウンロード

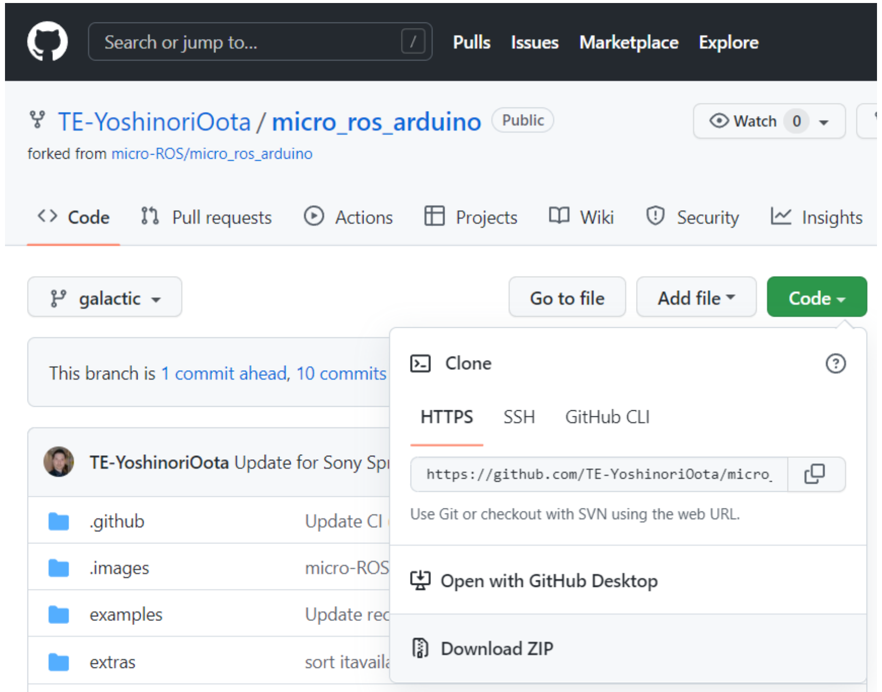


【STEP2】micro_ROS_arduino のインストール

Spresense の開発環境セットアップ

Spresense / micro-ROS-arduino セットアップ

【STEP3】Spresense_microROS_seminar 資料 のダウンロード



【STEP4】sketchesの中にある”spresense_micro-ROS_ publisher_serial を書き込み

Spresense の開発環境セットアップ

Spresense / micro-ROS-arduino セットアップ

Spresense を接続したままで、Ubuntu上で micro-ROS-agent を起動し、Spresense をリセット

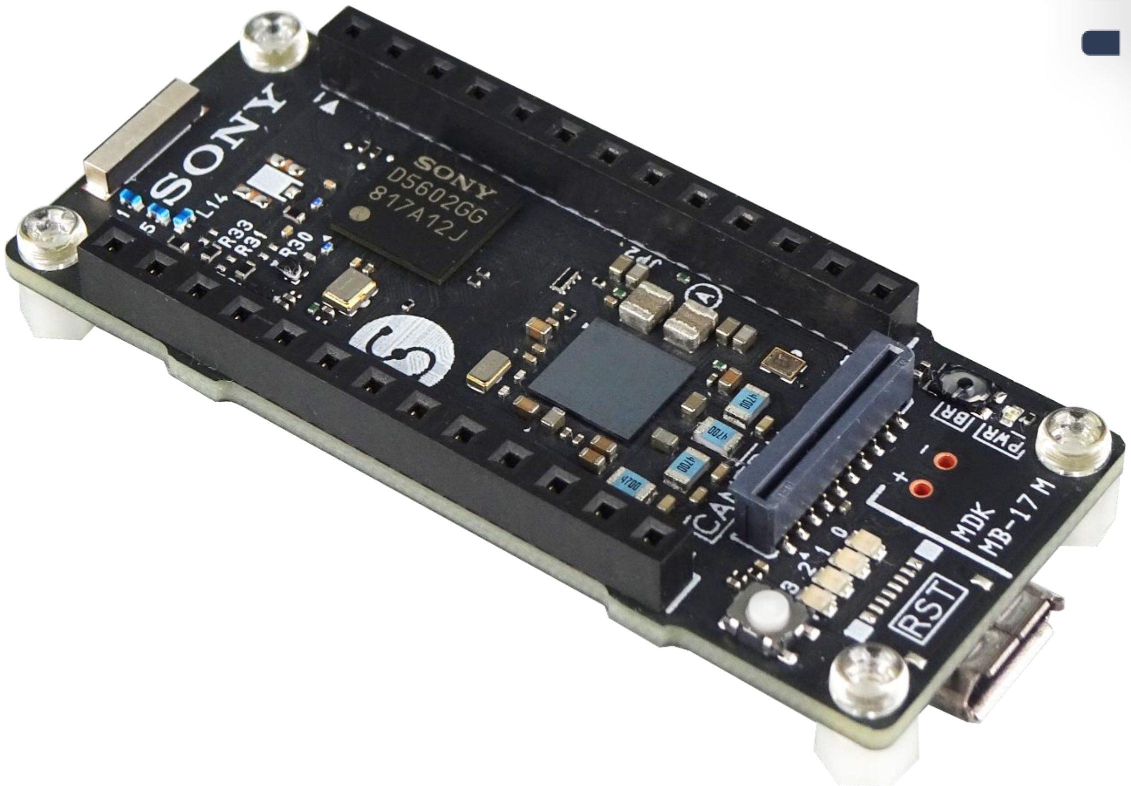
```
$   source /opt/ros/humble/setup.bash
$   sudo docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-ros-agent:humble serial --dev /dev/ttyUSB0
[1659348844.975038] info    | TermiosAgentLinux.cpp | init                  | running...          | fd: 3
[1659348844.975462] info    | Root.cpp              | set_verbose_level     | logger setup        | verbose_level: 4
[1659348850.359973] info    | Root.cpp              | create_client         | create              | client_key: 0x28E804DA, session_id: 0x81[1659348850.360103] info    | SessionManager.hpp | establish_session     | session established  | client_key: 0x28E804DA, address: 0
[1659348850.390805] info    | ProxyClient.cpp       | create_participant    | participant created  | client_key: 0x28E804DA, participant_id: 0x000(1)
[1659348850.407067] info    | ProxyClient.cpp       | create_topic          | topic created        | client_key: 0x28E804DA, topic_id: 0x000(2), participant_id: 0x000(1)
[1659348850.416714] info    | ProxyClient.cpp       | create_publisher      | publisher created    | client_key: 0x28E804DA, publisher_id: 0x000(3), participant_id: 0x000(1)
[1659348850.428346] info    | ProxyClient.cpp       | create_datawriter     | datawriter created   | client_key: 0x28E804DA, datawriter_id: 0x000(5), publisher_id: 0x000(3)
```

Spresense の開発環境セットアップ

Spresense / micro-ROS-arduino セットアップ

別のターミナルを開き、Ubuntu 上で spresense_micro-ROS_publisher が発信したトピックを確認

```
$ source /opt/ros/humble/setup.bash
$ ros2 topic list
/my_topic
/parameter_events
/rosout
$ ros2 topic echo /my_topic
data: 48
---
data: 49
---
```



micro-ROS puts ROS 2 onto microcontrollers

micro-ROS の実装

micro-ROS と ROS2 の通信f

<ROS2要素>

Node

Topic

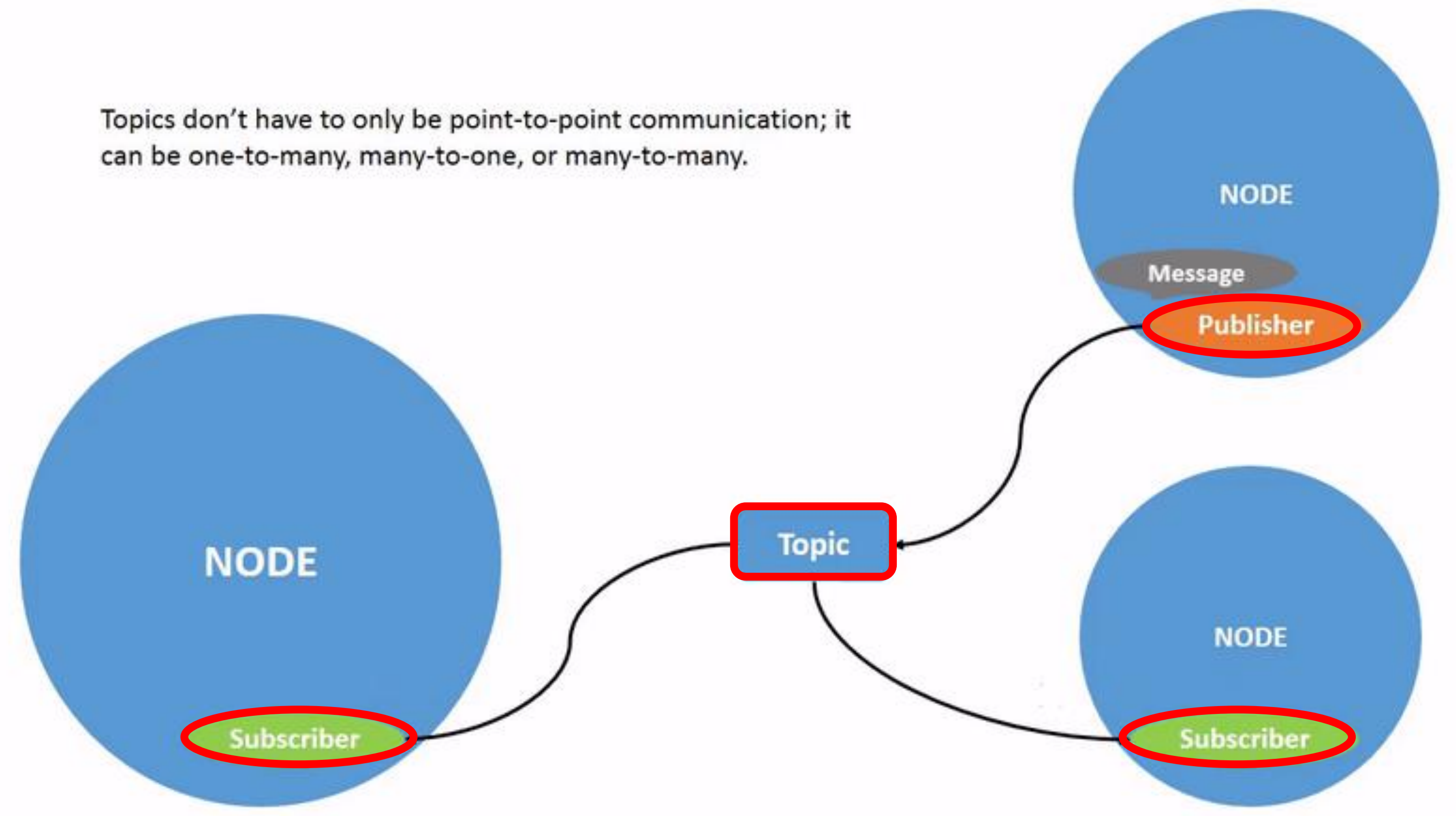
Subscriber

Publisher

Service

Service Client

Service Server



伝送路やプラットフォームを気にする必要ない
発信と受信するNodeを動的に入れ替えることができる

micro-ROS の実装（Topic Publisher）

<ROS2要素>

Node

Topic

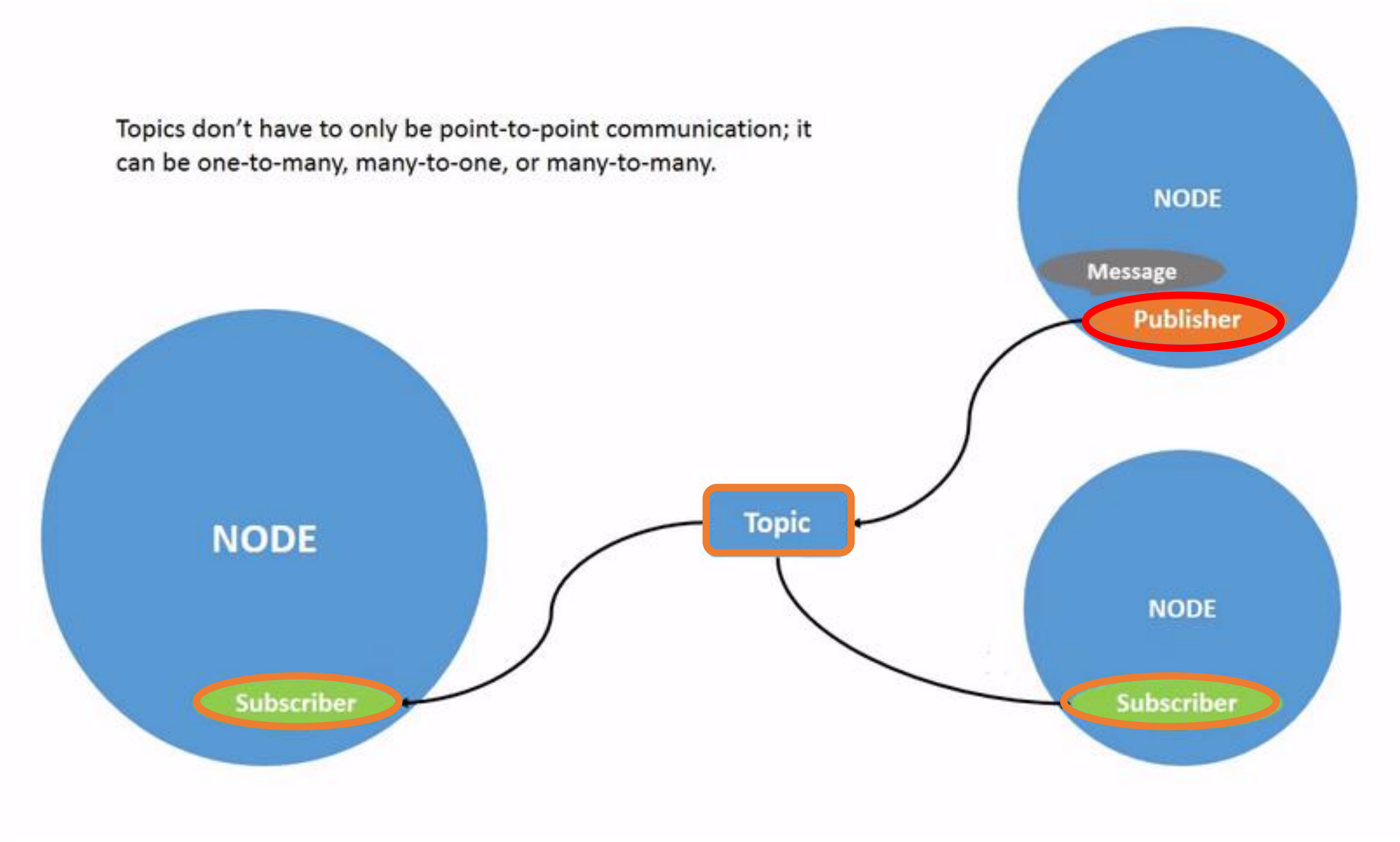
Subscriber

Publisher

Service

Service Client

Service Server



micro-ROS の実装（Topic Publisher）

Serial（default）実装

```
#include <micro_ros_arduino.h>
#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <std_msgs/msg/int32.h>

static rcl_allocator_t allocator;          // memory allocator
static rclc_support_t support;             // context structure
static rcl_node_t node;                    // node instance
static rcl_publisher_t publisher;          // publisher instance
static std_msgs__msg__Int32 msg;           // message for topics

#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}

void error_loop() {
  while(1){ digitalWrite(LED0, !digitalRead(LED0)); delay(100); }
}
```

helper functions
エラー発生時にLED0を点滅させる

```
void setup() {

  set_microros_transports();
  delay(2000);

  allocator = rcl_get_default_allocator();
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // create node
  RCCHECK(rclc_node_init_default(&node, "my_node_serial", "", &support));

  // create publisher
  RCCHECK(rclc_publisher_init_default(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "my_publisher_serial"));

  msg.data = 0;

  digitalWrite(LED0, HIGH);
}

void loop() {
  delay(100);
  RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
  msg.data++;
}
```

シリアル通信を初期化

メモリマネージャーなど各種設定

nodeの生成

publisherの生成

msg をパブリッシュ

micro-ROS の実装（Topic Publisher）

Serial 実装

```
void setup() {
  set_microros_transports();
  delay(2000);

  allocator = rcl_get_default_allocator();
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // create node
  RCCHECK(rclc_node_init_default(&node, "my_node_serial", "", &support));

  // create publisher
  RCCHECK(rclc_publisher_init_default(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "my_publisher_serial"));

  msg.data = 0;

  digitalWrite(LED0, HIGH);
}

void loop() {
  delay(100);
  RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
  msg.data++;
}
```

WiFi実装

```
void setup() {
  rmw_uros_set_custom_transport(
    false, NULL,
    arduino_wifi_transport_open,
    arduino_wifi_transport_close,
    arduino_wifi_transport_write,
    arduino_wifi_transport_read );
  ... snip ...

  // create publisher
  RCCHECK(rclc_publisher_init_default(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "my_publisher_serial"));

  msg.data = 0;

  digitalWrite(LED0, HIGH);
}

void loop() {
  delay(100);
  RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
  msg.data++;
}
```

set_microros_transports() を
rmw_uros_set_custom_transport() に
置き換え

micro-ROS の実装（Topic Publisher）

UDP（default）実装（GS2200）

```
#include <micro_ros_arduino.h>
#include <uxr/client/transport.h>
#include <rmw_microros/rmw_microros.h>
#define ATCMD_CHECK(fn) { while(fn != ATCMD_RESP_OK); }

ATCMD_NetworkStatus net_status;
extern uint8_t ESCBuffer[];
extern uint32_t ESCBufferCnt;
const char server_ip[16] = “xxx.xxx.xxx.xxx";
const char server_port[6] = "8888";
const char client_port[6] = "10001";
uint8_t client_id = 0;


extern “C” {
 bool arduino_wifi_transport_open(struct uxrCustomTransport* transport) {
  ATCMD_REGDOMAIN_E regDomain;
  const char ssid[32] = “ssid”;
  const char pswd[32] = “passwd”;
  char macid[20];

  Init_GS2200_SPI();
  while (Get_GPIO37Status()) {  …snip… }
  … skip …
  return true;
 }
```

```
bool arduino_wifi_transport_close(struct uxrCustomTransport* transport) {
  return true;
}

size_t arduino_wifi_transport_write(struct uxrCustomTransport* transport,
                                    const uint8_t* buf, size_t len, uint8_t* errcode) {
  (void)errcode;
  if (len == 0) return len;
  WiFi_InitESCBuffer();
  if (AtCmd_SendBulkData(client_id, buf, len) != ATCMD_RESP_OK)  return 0;
  return len;
}

size_t arduino_wifi_transport_read(struct uxrCustomTransport* transport, uint8_t* buf,
                                   size_t len, int timeout, uint8_t* errcode) {
  (void) errcode;
  int res = 0;
  if (AtCmd_RecvResponse() == ATCMD_RESP_BULK_DATA_RX) {
   if (Check_CID(client_id)) {
     ConsolePrintf( "Receive %d bytes¥r¥n", ESCBufferCnt-1, ESCBuffer+1 );
     memcpy(buf, ESCBuffer+1, ESCBufferCnt-1);
     res = ESCBufferCnt-1;
  }
  WiFi_InitESCBuffer();
  return res;
 }
} // extern “C”
```

micro-ROS の実装（Topic Publisher）

UDP（default）実装（ESP8266） 利用ライブラリ：https://github.com/YoshinoTaro/ESP8266ATLib-for-Spresense

```
#include "ESP8266ATLib.h“
#include "IPAddress.h“
#define BAUDRATE 115200

#include <micro_ros_arduino.h>
#include <uxr/client/transport.h>
#include <rmw_microros/rmw_microros.h>


#include "ESP8266ATLib.h“
const char server_ip[16] = "192.168.xxx.xxx";
const char server_port[6] = "8888";

extern "C" {
 #define BAUDRATE 115200

 bool arduino_wifi_transport_open(struct uxrCustomTransport* transport) {
  bool result = false;
  esp8266at.begin(BAUDRATE);
  result = esp8266at.espConnectAP(“ssid",“passwd");
  result = esp8266at.setupUdpClient(server_ip, server_port);
  return result;
 }
```

```
 bool arduino_wifi_transport_close(struct uxrCustomTransport* transport) {
  return true;
 }

 size_t arduino_wifi_transport_write(struct uxrCustomTransport* transport,
                                     const uint8_t* buf, size_t len, uint8_t* errcode) {
  (void)errcode;
  bool result = esp8266at.sendUdpMessageToServer(buf, len);
  if (result) return len;
  else return 0;
 }

 size_t arduino_wifi_transport_read(struct uxrCustomTransport* transport, uint8_t* buf,
                                    size_t len, int timeout, uint8_t* errcode) {
  (void) errcode;
  int res = 0;
  uint32_t start_time = millis();
  do {
   res = esp8266at.espListenToUdpServer(buf, len);
  } while(!res && (millis() - start_time < timeout));
  return res;
 }
} // extern “C”
```

micro-ROS の実装（Topic Publisher）

UDP動作確認

Ubuntu のターミナル上で micro-ROS-agent を起動し、Spresense をリセット

通信方式を指定する（シリアル通信の場合は次の通り）

serial --dev /dev/ttyUSB0

```
$ source /opt/ros/humble/setup.bash
$ export ROS_DOMAIN_ID=30
$ sudo docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-ros-agent:humble udp4 -p 8888
[1659348844.975038] info | TermiosAgentLinux.cpp | init | running... | fd: 3
[1659348844.975462] info | Root.cpp | set_verbose_level | logger setup | verbose_level: 4
[1659348850.359973] info | Root.cpp | create_client | create | client_key: 0x28E804DA, session_id:
0x81[1659348850.360103] info | SessionManager.hpp | establish_session | session established | client_key: 0x28E804DA, address:
0
[1659348850.390805] info | ProxyClient.cpp | create_participant | participant created | client_key: 0x28E804DA, participant_id:
0x000(1)
[1659348850.407067] info | ProxyClient.cpp | create_topic | topic created | client_key: 0x28E804DA, topic_id: 0x000(2),
participant_id: 0x000(1)
[1659348850.416714] info | ProxyClient.cpp | create_publisher | publisher created | client_key: 0x28E804DA, publisher_id:
0x000(3), participant_id: 0x000(1)
[1659348850.428346] info | ProxyClient.cpp | create_datawriter | datawriter created | client_key: 0x28E804DA, datawriter_id:
0x000(5), publisher_id: 0x000(3)
```

micro-ROS の実装（Topic Subscriber）

<ROS2要素>

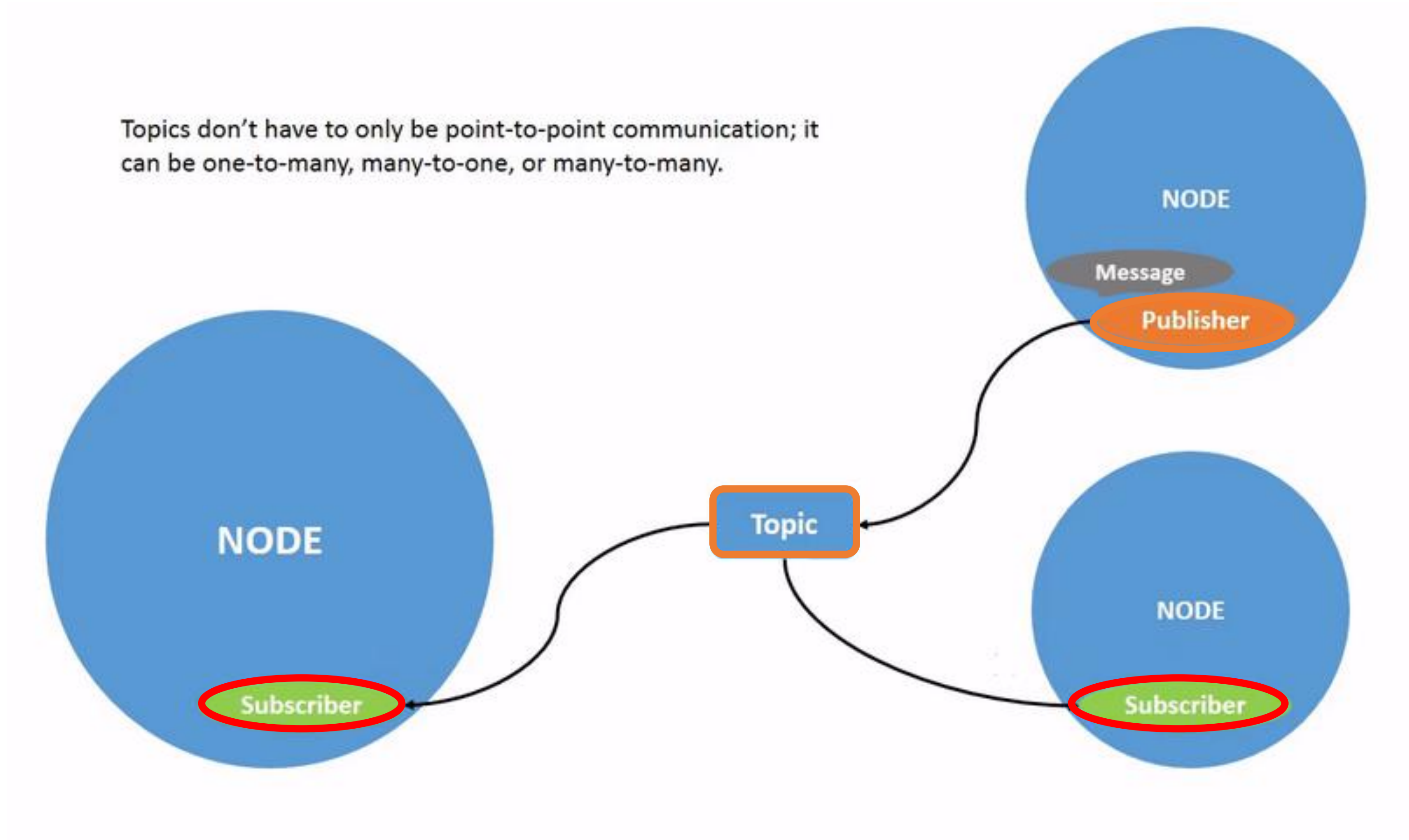
Node

Topic

- Subscriber
- Publisher

Service

- Service Client
- Service Server



micro-ROS の実装 (Topic Subscriber)

Subscription 実装（Serialの場合）

```
#include <micro_ros_arduino.h>
#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <std_msgs/msg/int32.h>

static rcl_allocator_t allocator;            // memory allocator
static rclc_support_t support;               // context structure
static rcl_node_t node;                      // node instance
static rcl_subscription_t subscriber;        // subscriber instance
static rclc_executor_t executor;             // function executor instance
static std_msgs__msg__Int32 msg;             // message for topics

#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}

void error_loop() {
  while(1){ digitalWrite(LED0, !digitalRead(LED0)); delay(100); }
}

void subscription_callback(const void * msgin) {
  const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msgin;
  digitalWrite(LED1, (msg->data == 0) ? LOW : HIGH);
}
```

subscription callback function
新しいデータが来た時に呼ばれるコールバック関数

```
void setup() {
  set_microros_transports();
  allocator = rcl_get_default_allocator();
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));
  RCCHECK(rclc_node_init_default(&node, "my_node_serial", "", &support));

  RCCHECK(rclc_subscription_init_default(&subscriber,   &node,
   ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),   "my_subscriber_serial"));

  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  RCCHECK(rclc_executor_add_subscription(&executor, &subscriber,
             &msg,  &subscription_callback, ON_NEW_DATA));

  digitalWrite(LED0, HIGH);
}

void loop() {
  delay(100);
  rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));
}
```

subscriber の生成

subscription タスクを
Executorに登録

100ミリ秒毎に
topicの有無を確認

micro-ROS の実装 (Topic Subscriber)

Subscriber の動作確認

【STEP1】Spresense を接続したままで、Ubuntu上で micro-ROS-agent を起動し、Spresense をリセット

WIFiの場合は次のように設定: udp4 -p 8888

```
$    source /opt/ros/humble/setup.bash
$    sudo docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-ros-agent:humble serial --dev /dev/ttyUSB0
[1659407603.208276] info     | TermiosAgentLinux.cpp | init                  | running...           | fd: 3[1659407603.208943] info     | Root.cpp           |
set_verbose_level        | logger setup           | verbose_level: 4
[1659407603.985348] info     | Root.cpp           | create_client           | create               | client_key: 0x44BDFA8B, session_id:
0x81[1659407603.986448] info     | SessionManager.hpp | establish_session       | session established     | client_key: 0x44BDFA8B, address: 0
[1659407605.450670] info     | ProxyClient.cpp     | create_participant       | participant created     | client_key: 0x44BDFA8B, participant_id: 0x000(1)
[1659407605.464926] info     | ProxyClient.cpp     | create_topic            | topic created          | client_key: 0x44BDFA8B, topic_id: 0x000(2),
participant_id: 0x000(1)
[1659407605.474488] info     | ProxyClient.cpp     | create_subscriber       | subscriber created      | client_key: 0x44BDFA8B, subscriber_id: 0x000(4),
participant_id: 0x000(1)
[1659407605.485901] info     | ProxyClient.cpp     | create_datareader       | datareader created      | client_key: 0x44BDFA8B, datareader_id:
0x000(6), subscriber_id: 0x000(4)
```

micro-ROS の実装 (Topic Subscriber)

Subscriber の動作確認

【STEP2】別ターミナルで次のコマンドを実行して Spresense の LED1 の点灯・消灯を操作

```
$    source /opt/ros/humble/setup.bash
$    ros2 topic list
/my_subscriber
/parameter_events
/rosout
$    ros2 topic pub --once /my_subscriber std_msgs/Int32 "data: 1“
publisher: beginning loop
publishing #1: std_msgs.msg.Int32(data=1)
```

LED1を点灯

```
$    ros2 topic pub --once /my_subscriber std_msgs/Int32 "data: 0“
publisher: beginning loop
publishing #1: std_msgs.msg.Int32(data=0)
```

LED1を消灯

micro-ROS の実装（Image Publisher）

Advanced

<ROS2要素>

Node

Topic

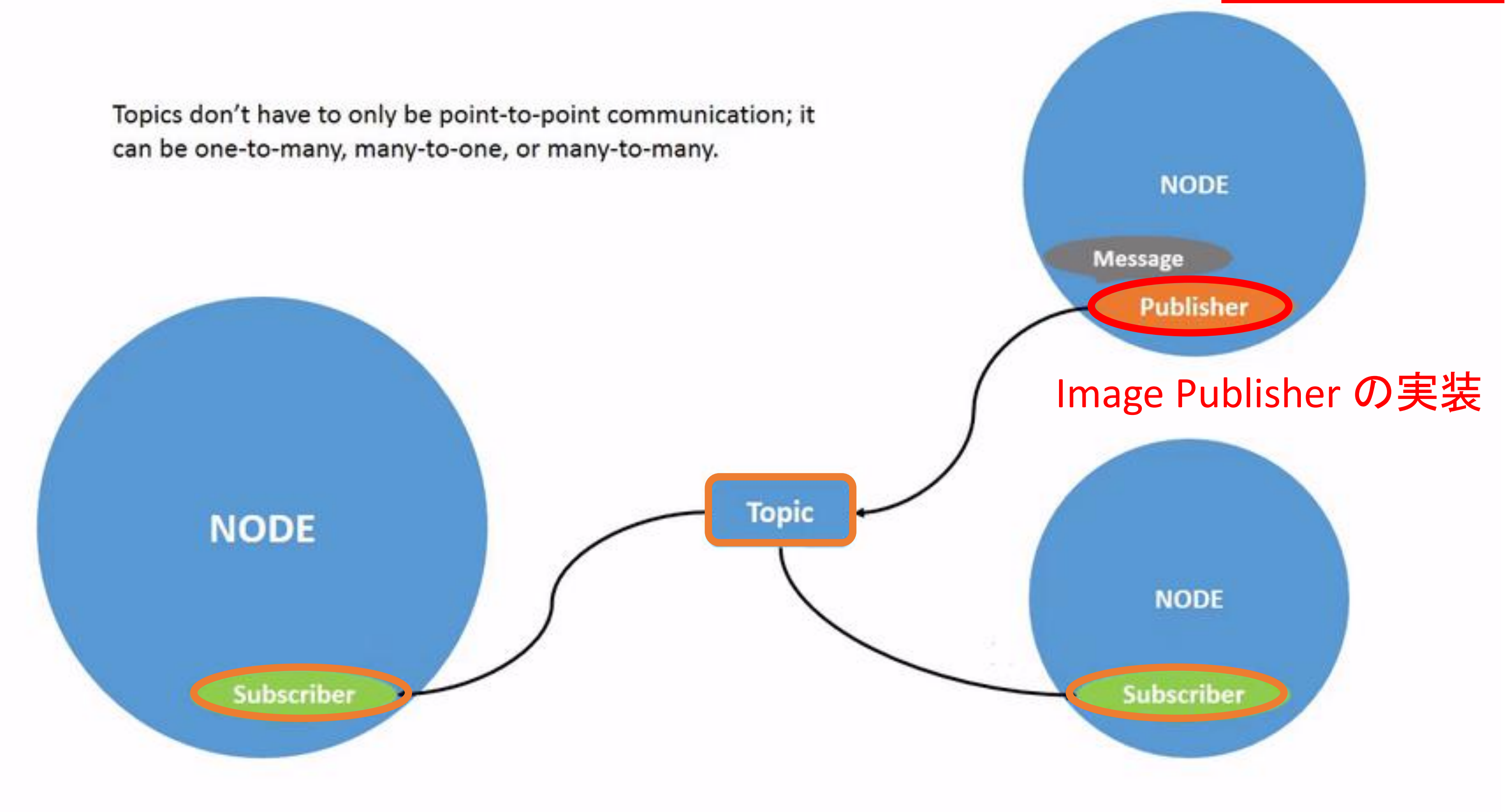
Subscriber

Publisher

Service

Service Client

Service Server



Spresenseカメラから画像を取得し、Topicとして画像を配信します

micro-ROS の実装（Image Publisher）

Advanced

Image Publisherの実装

```
#include <micro_ros_arduino.h>
#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <sensor_msgs/msg/compressed_image.h>

#include <Camera.h>

static rcl_allocator_t allocator;
static rclc_support_t support;
static rcl_node_t node;
static rcl_publisher_t publisher;
static sensor_msgs__msg__CompressedImage msg_static;

#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}
void error_loop() {
  while(1){ digitalWrite(LED0, !digitalRead(LED0)); delay(100); }
}

void setup() {

  set_microros_transports();

  allocator = rcl_get_default_allocator();

  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  RCCHECK(rclc_node_init_default(&node, "my_node_serial", "", &support));
```

```
RCCHECK(rclc_publisher_init_default(&publisher,   &node,
  ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, CompressedImage),
  "image/compressed"));
```

compressed_image の初期化

```
static uint8_t img_buff[20000] = {0};
static char    frame_id_data[30] = {0};
static char    format_data[6] = {0};

msg_static.header.frame_id.capacity = 7;
msg_static.header.frame_id.data=frame_id_data;
memcpy(msg_static.header.frame_id.data, "myframe", 7);
msg_static.header.frame_id.size = 7;
msg_static.data.capacity = 20000;
msg_static.data.data=img_buff;
msg_static.data.size = 0;
msg_static.format.capacity = 4;
msg_static.format.data=format_data;
memcpy(msg_static.format.data, "jpeg", 4);
msg_static.format.size = 4;
```

Spresenseカメラの初期化

```
// Camera setup
theCamera.begin();
theCamera.setStillPictureImageFormat(
  CAM_IMGSIZE_QVGA_H, CAM_IMGSIZE_QVGA_V, CAM_IMAGE_PIX_FMT_JPG);

digitalWrite(LED0, HIGH);
}
```

micro-ROS の実装（Image Publisher）

Advanced

Image Publisherの実装

```
void loop() {
  digitalWrite(LED2, LOW);
  digitalWrite(LED3, LOW);
  CamImage img = theCamera.takePicture();
  if (img.isAvailable()) {
    if (img.getImgSize() > msg_static.data.capacity) {
      digitalWrite(LED3, HIGH);     // When the image size is too large, LED3 is on
    } else {
      msg_static.data.size = img.getImgSize();
      memcpy(msg_static.data.data, img.getImgBuff(), img.getImgSize());
      RCSOFTCHECK(rcl_publish(&publisher, &msg_static, NULL));
    }
  } else {
    digitalWrite(LED2, HIGH);       // When taking a picture is failed, LED2 is on
  }
  sleep(1);
}
```

１秒ごとに写真撮影をして画像をトピックとしてPublish

micro-ROS の実装 (Image Publisher)

画像をホストPCで確認する

Advanced

【STEP１】 rqt_image_view の起動

```
$ source /opt/ros/humble/setup.bash
$ ros2 run rqt_image_view rqt_image_view
```

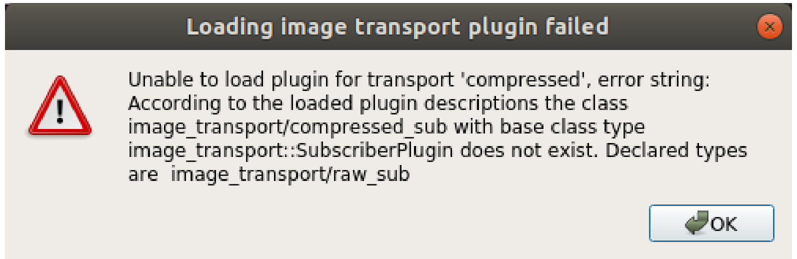
micro-ROS の実装 (Image Publisher)

Advanced

画像をホストPCで確認する

【STEP２】 rqt_image_view で画像確認

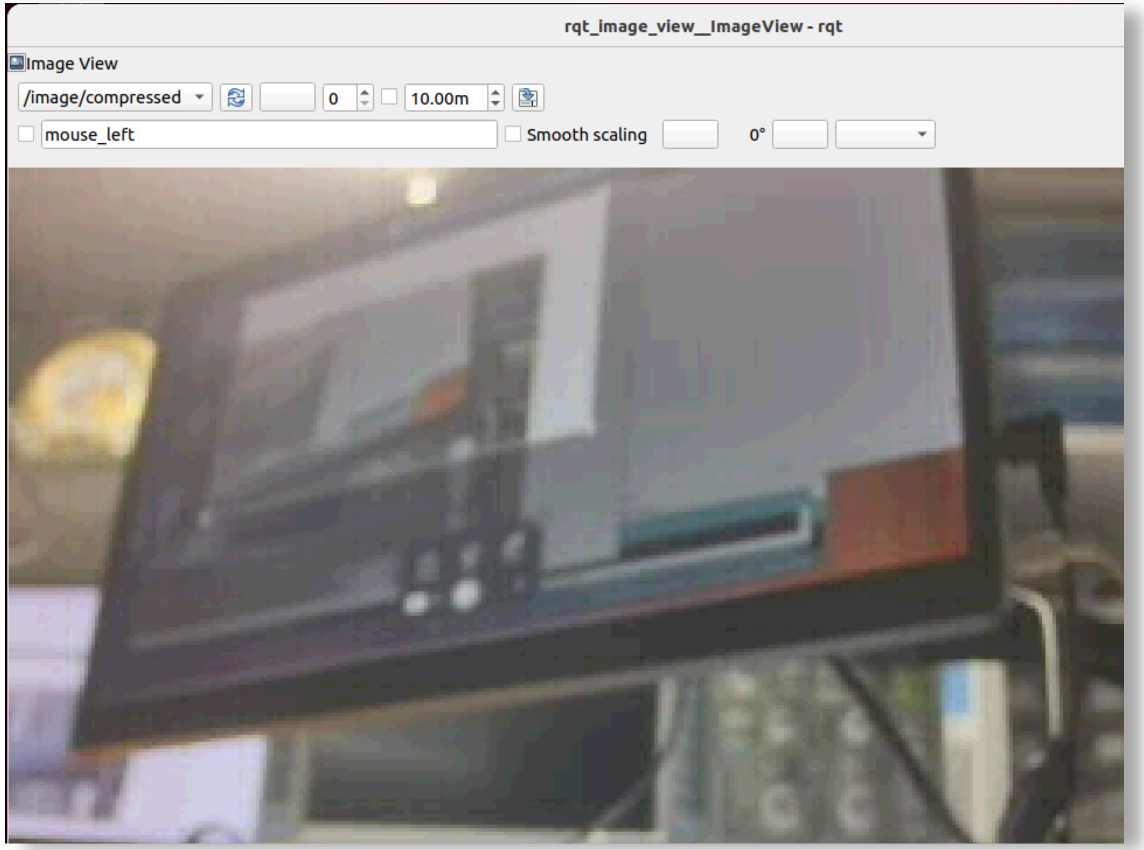
ROSのDistroを更新等、環境を変えると、次のようなエラーが出る場合があります



その場合は、次のコマンドでプラグインをインストールし直してください。

```
$   sudo apt install ros-humbe-image-transport-plugins
```

rqt_image_view の画面（例）



micro-ROS と ROS2 の通信

<ROS2要素>

Node

Topic

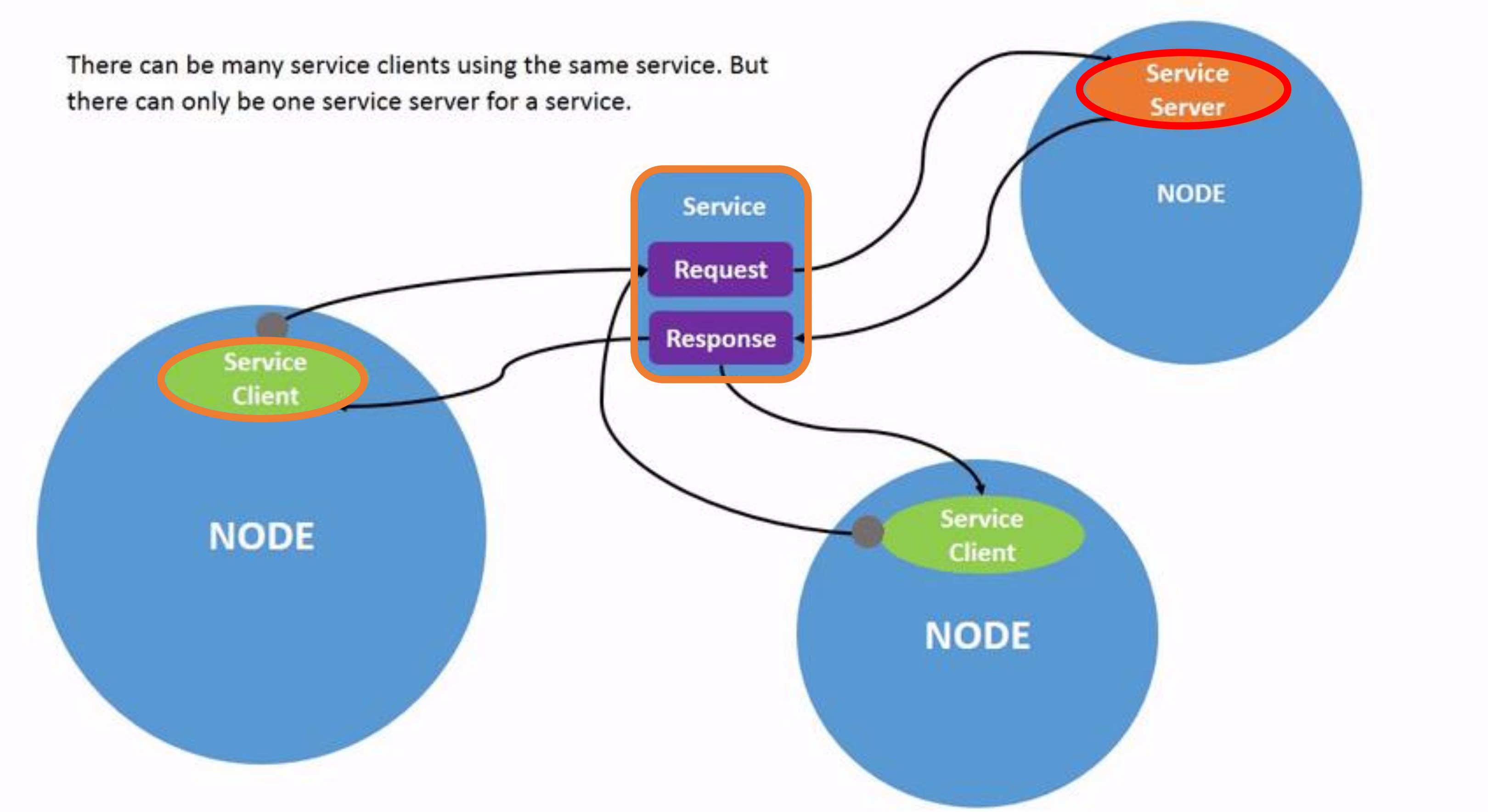
Subscriber

Publisher

Service

Service Client

Service Server



自分で定義した Service を組み込むには micro-ROS library のリビルドが必要です。[GalacticではROS2との通信に不具合があります]。Humble でも若干問題があるようです（後述）。micro-ROS-agent の実装がまだ安定してないようなので、使う場合は注意してください。

micro-ROS と ROS2 の通信

<ROS2要素>

Node

Topic

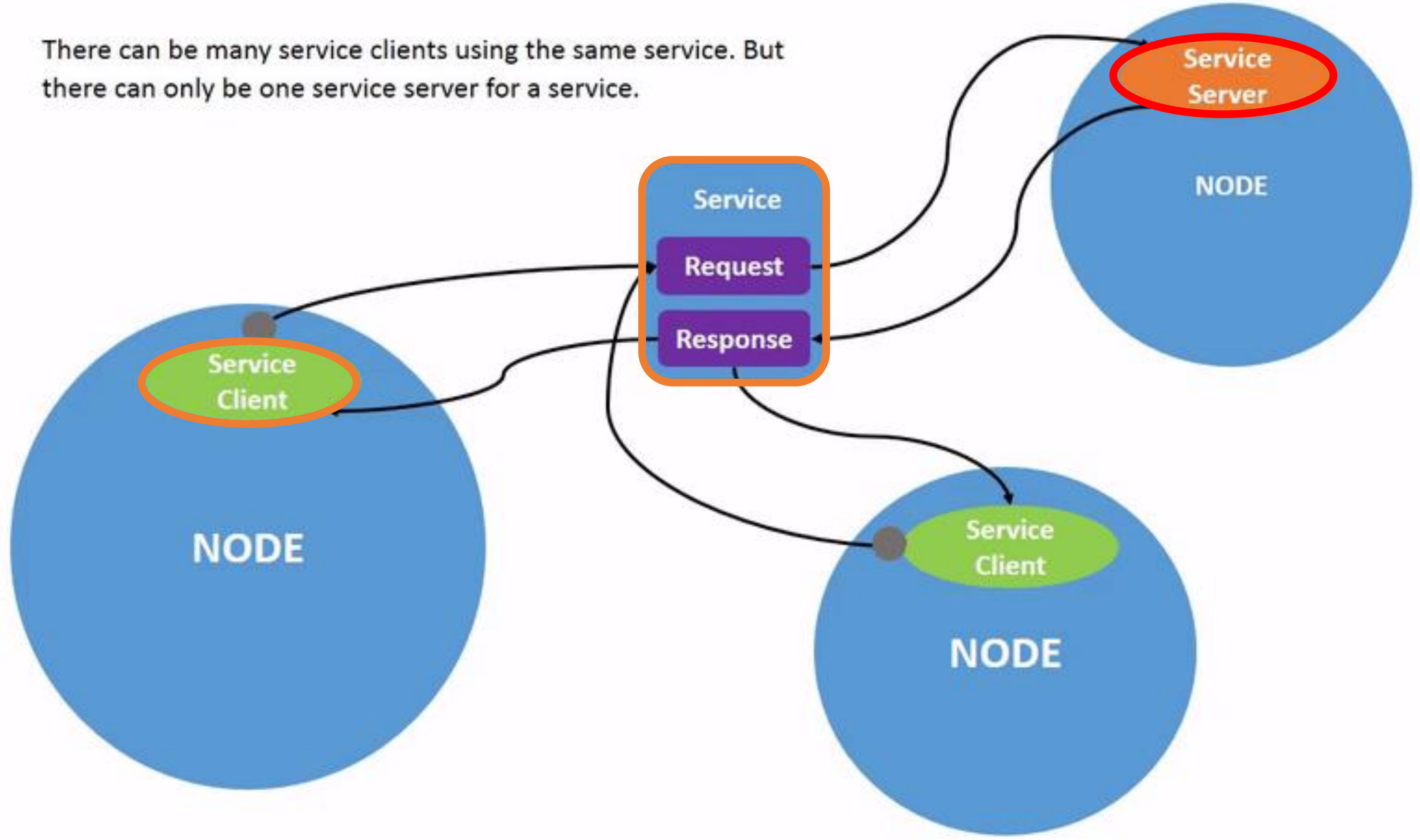
Subscriber

Publisher

Service

Service Client

Service Server



micro-ROS の実装 (Service Server)

Service Server 実装

```
#include <micro_ros_arduino.h>
#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <std_msgs/msg/int32.h>
#include “std_srvs/srv/trigger.h”

static rcl_allocator_t allocator;          // memory allocator
static rclc_support_t support;             // context structure
static rcl_node_t node;                    // node instance
static rcl_service_t service;              // service instance
static rclc_executor_t executor;           // function executor instance

#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}
void error_loop() {
  while(1){ digitalWrite(LED0, !digitalRead(LED0)); delay(100); }
}

std_srvs__srv__Trigger_Response res;
std_srvs__srv__Trigger_Request req;
const int capacity = 32;
uint8_t data[capacity] = {0};              コールバック関数用のパラメータ

void service_callback(const void* req, void* res) {
  static bool result = false;
  static int cnt = 0;
  std_srvs__srv__Trigger_Response* res_in = (std_srvs__srv__Trigger_Response*)res;
  std_srvs__srv__Trigger_Response * res_in = (std_srvs__srv__Trigger_Response *) res;
  sprintf(data, "Response[%d]", cnt);
  res_in->success = !result;    result = res_in->success;
  res_in->message.capacity=capacity;
  res_in->message.size = strlen(data);
  res_in->message.data = data;
  printf("Send Response: %d %s¥n", res_in->success, res_in->message.data);
  ++cnt;
  digitalWrite(LED1, result);              クライアントからアクセスがあったときに呼ばれる
}                                          コールバック関数

void setup() {
  set_microros_transports();
  allocator = rcl_get_default_allocator();
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));
  RCCHECK(rclc_node_init_default(&node, "my_node_serial", "", &support));

  RCCHECK(rclc_service_init_default(&service, &node,       サービスの登録
    ROSIDL_GET_SRV_TYPE_SUPPORT(std_srvs, srv, Trigger), “srv_trigger_serial"));
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  RCCHECK(rclc_executor_add_service(&executor, &service, &req, &res, service_callback));

  digitalWrite(LED0, HIGH);
}

void loop() {
  delay(100);
  rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));
}
```

micro-ROS の実装 (Service Server)

Service Server の動作確認

【STEP1】Spresense を接続したままで、Ubuntu上で micro-ROS-agent を起動し、Spresense をリセット

WIFi の場合は次のように設定：udp4 –p 8888

```
$    source /opt/ros/humble/setup.bash
$    sudo docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-ros-agent:humble serial --dev /dev/ttyUSB0
[1659860178.557416] info    | TermiosAgentLinux.cpp | init                  | running...            | fd: 3
[1659860178.557910] info    | Root.cpp          | set_verbose_level       | logger setup          | verbose_level: 4
[1659860178.806242] info    | Root.cpp          | create_client           | create                | client_key: 0x24C8DAAF, session_id: 0x81
[1659860178.806378] info    | SessionManager.hpp | establish_session      | session established   | client_key: 0x24C8DAAF, address: 0
[1659860178.971228] info    | ProxyClient.cpp    | create_participant     | participant created   | client_key: 0x24C8DAAF, participant_id: 0x000(1)
[1659860178.998719] info    | ProxyClient.cpp    | create_replier         | replier created       | client_key: 0x24C8DAAF, requester_id: 0x000(7), participant_id: 0x000(1)
```

micro-ROS の実装 (Service Server)

Service Server の動作確認

【STEP2】次のコマンドを使って Service Server にアクセス

```
$    ros2 service list
/srv_trigger_serial
```

Topic名は、プログラムで設定した名前になります
例）srv_trigger_esp8266 or srv_trigger_gs2200

```
$    ros2 service call /srv_trigger_serial std_srvs/srv/Trigger
requester: making request: std_srvs.srv.Trigger_Request()

response:std_srvs.srv.Trigger_Response(success=True, message='Response[0]')
```

LED1が点灯

```
$    ros2 service call /srv_trigger_serial std_srvs/srv/Trigger
requester: making request: std_srvs.srv.Trigger_Request()

response:std_srvs.srv.Trigger_Response(success=False, message='Response[1]')
```

LED1が消灯

micro-ROS と ROS2 の通信

<ROS2要素>

Node

Topic

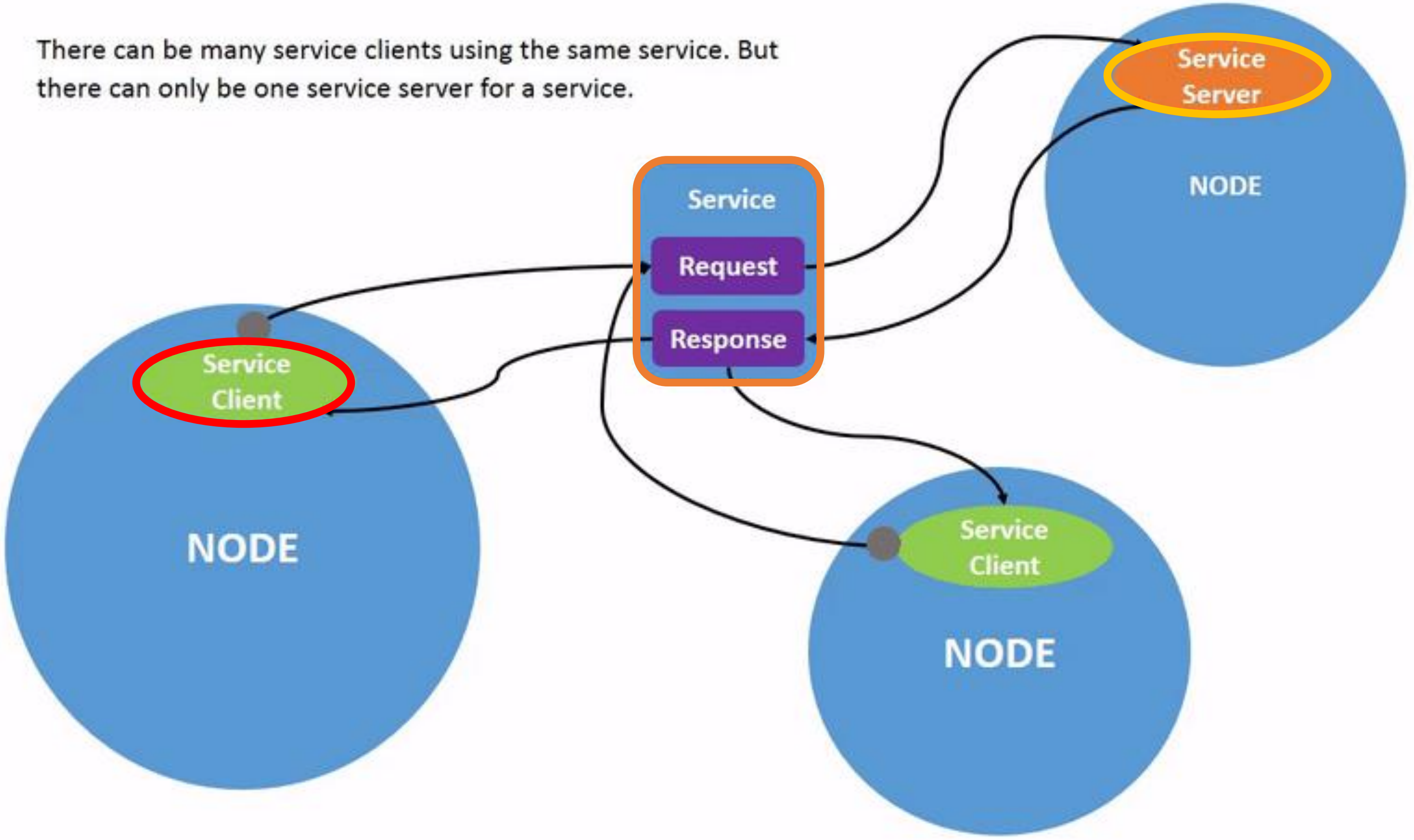
Subscriber

Publisher

Service

Service Client

Service Server



micro-ROS の実装 (Service Client)

Service Client 実装

```
#include <micro_ros_arduino.h>
#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
#include <std_srvs/srv/trigger.h>

static rcl_allocator_t allocator;            // memory allocator
static rclc_support_t support;               // context structure
static rcl_node_t node;                      // node instance
static rcl_client_t client;                  // client instance
static rclc_executor_t executor;             // function executor instance
std_srvs__srv__Trigger_Request req;

#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}
void error_loop() {
  while(1){ digitalWrite(LED0, !digitalRead(LED0)); delay(100); }
}

void  client_callback(const void* res) {
  digitalWrite(LED1, !digitalRead(LED1));
  std_srvs__srv__Trigger_Response* res_in = (std_srvs__srv__Trigger_Response*)res;
  printf("Received service response: %d¥n", res_in->success);
  char* ptr = res_in->message.data;
  ++ptr; // The counter measure to micro-ROS-agent bug. (The head of pointer sets null)
  printf("Reeived service message %s (size: %d)¥n", res_in->message.data, res_in->message.size);
}
```

サーバーからレスポンスが返って来た時に呼ばれるコールバック関数

```
void setup() {
  set_microros_transports();
  allocator = rcl_get_default_allocator();
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));
  RCCHECK(rclc_node_init_default(&node, "my_node_serial", "", &support));

  RCCHECK(rclc_client_init_default(&client, &node,
   ROSIDL_GET_SRV_TYPE_SUPPORT(std_srvs, srv, Trigger), "srv_trigger_py"));
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  std_srvs__srv__Trigger_Response res;
  RCCHECK(rclc_executor_add_client(&executor, &client, &res, client_callback));
}
```

クライアントの登録

```
void loop() {
  sleep(2); // Sleep a while to ensure DDS matching before sending a request
  int64_t seq;
  RCCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(2000)));
  RCCHECK(rcl_send_request(&client, &req, &seq));
}
```

クライアントリクエストの発信

micro-ROS の実装 (ROS2 Server)

テスト用ROS2サーバーの実装

【STEP1】ROS2用ワークディレクトリを生成し、パッケージを生成

```
$   source /opt/ros/humble/setup.bash
$   mkdir ros2_ws && mkdir ros2_ws/src && cd ros2_ws
$   rosdep install -i --from-path src --rosdistro humble -y
$   ros2 pkg create --build-type ament_python py_srv --dependencies rclpy std_srvs
going to create a new package
package name: py_srv
destination directory: /home/user/ros2_ws/src
…
build type: ament_python
dependencies: ['rclpy', 'std_srvs']
creating folder ./py_srv
creating ./py_srv/package.xml
…
[WARNING]: Unknown license 'TODO: License declaration'.  This has been set in the package.xml, but no LICENSE file has been created.
It is recommended to use one of the ament license identitifers:
…
```

micro-ROS の実装 (ROS2 Server)

【STEP2】trigger_service.py の実装

ros2_ws/src/py_srv/py_srv/trigger_service.py

```
from std_srvs.srv import Trigger
import rclpy
from rclpy.node import Node

class TriggerService(Node):
  def __init__(self):
    self.result = 0
    self.cnt = 0
    super().__init__('trigger_service')
    self.srv = self.create_service(Trigger, 'my_trigger', self.trigger_callback)

  def trigger_callback(self, request, response):
    self.get_logger().info('incoming request')
    response.success = not self.result
    self.result = response.success
    response.message = str(" Response[" + str(self.cnt) + "]")
    self.get_logger().info(response.message)
    self.cnt = self.cnt + 1
    return response

def main():
  rclpy.init()
  trigger_service = TriggerService()
  rclpy.spin(trigger_service)
  rclpy.shutdown()

if __name__ == '__main__':
  main()
```

ROS2→micro-ROSへService Response で文字列を送信する時に、文字列先頭に(null) が設定されます。先頭はスペースなど読む必要のない文字を入れてください。

【STEP3】setup.py の変更

ros2_ws/src/py_srv/setup.py

```
from setuptools import setup

package_name = 'py_srv'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='ystaro',
    maintainer_email='ystaro@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
        'service=py_srv.trigger_service:main',      追加
        ],
    },
)
```

コマンドライン引数に 'service' が指定されると
trigger_service の main が呼ばれる

micro-ROS の実装 (Service Client)

テスト用サーバーの実装

【STEP5】ROS2の Trigger Service をビルドし、コマンドを実行できるように登録

```
$    cd ~/ros2_ws
$    rosdep install -i --from-path src --rosdistro humble -y
$    colcon build  --packages-select py_srv
Starting >>> py_srv
--- stderr: py_srv
/usr/lib/python3/dist-packages/setuptools/command/install.py:34: SetuptoolsDeprecationWarning: setup.py install is deprecated. Use build and pip and other standards-based tools.
  warnings.warn(
---
Finished <<< py_srv [2.76s]

Summary: 1 package finished [3.25s]
  1 package had stderr output: py_srv
$    . install/setup.bash
```

micro-ROS の実装 (Service Client)

テスト用サーバーの実装

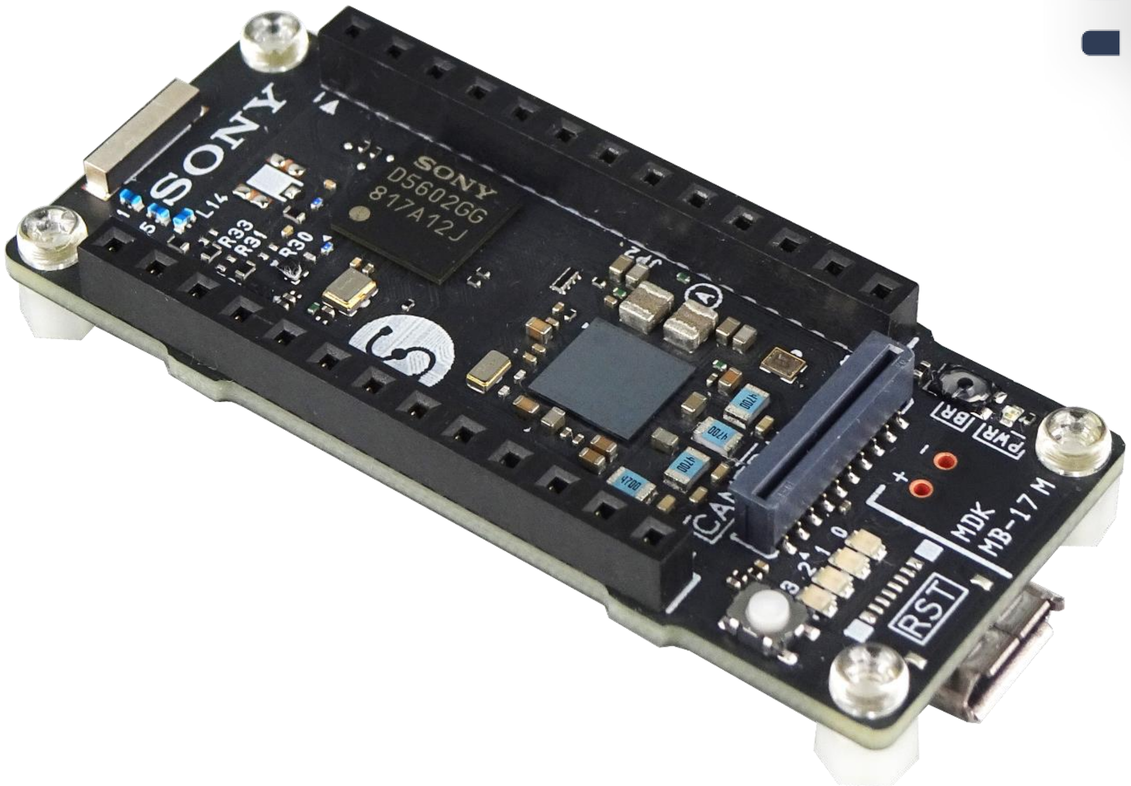
【STEP 5】Arduino IDE シリアルコンソールを開く

ESP8266版で確認してください

```
Send Trigger to server.
Send Trigger to server.
Send Trigger to server.
Send Trigger to server.
Received service response  1
Received service message  Response[0], 12
Send Trigger to server.
Received service response  0
Received service message  Response[1], 12
Send Trigger to server.
Received service response  1
Received service message  Response[2], 12
Send Trigger to server.
Received service response  0
Received service message  Response[3], 12
Send Trigger to server.
Received service response  1
Received service message  Response[4], 12
```

【STEP 6】Trigger Service の実行

```
$   ros2 run py_srv service
[INFO] [1659965392.183654203] [trigger_service]: incoming request
[INFO] [1659965392.184600874] [trigger_service]:  Response[0]
[INFO] [1659965394.530636585] [trigger_service]: incoming request
[INFO] [1659965394.531518345] [trigger_service]:  Response[1]
[INFO] [1659965396.930469210] [trigger_service]: incoming request
[INFO] [1659965396.931471882] [trigger_service]:  Response[2]
[INFO] [1659965399.330824649] [trigger_service]: incoming request
[INFO] [1659965399.332065761] [trigger_service]:  Response[3]
[INFO] [1659965401.731314008] [trigger_service]: incoming request
[INFO] [1659965401.732427844] [trigger_service]:  Response[4]
```



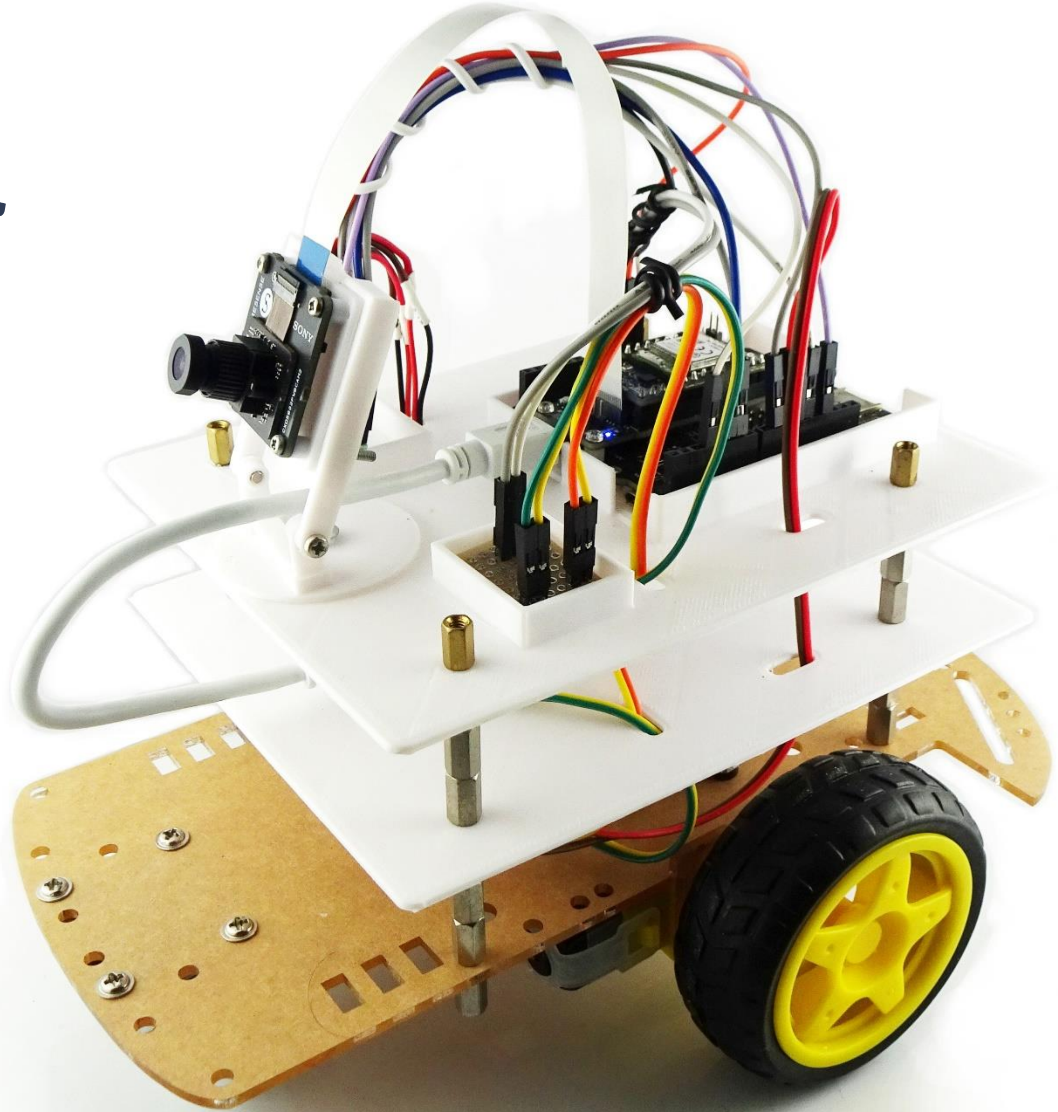
µROS

micro-ROS puts ROS 2 onto microcontrollers

Spresense による TurtleBot の実装

SprTurtleBotの制作

SprTurtleBot0 の外観



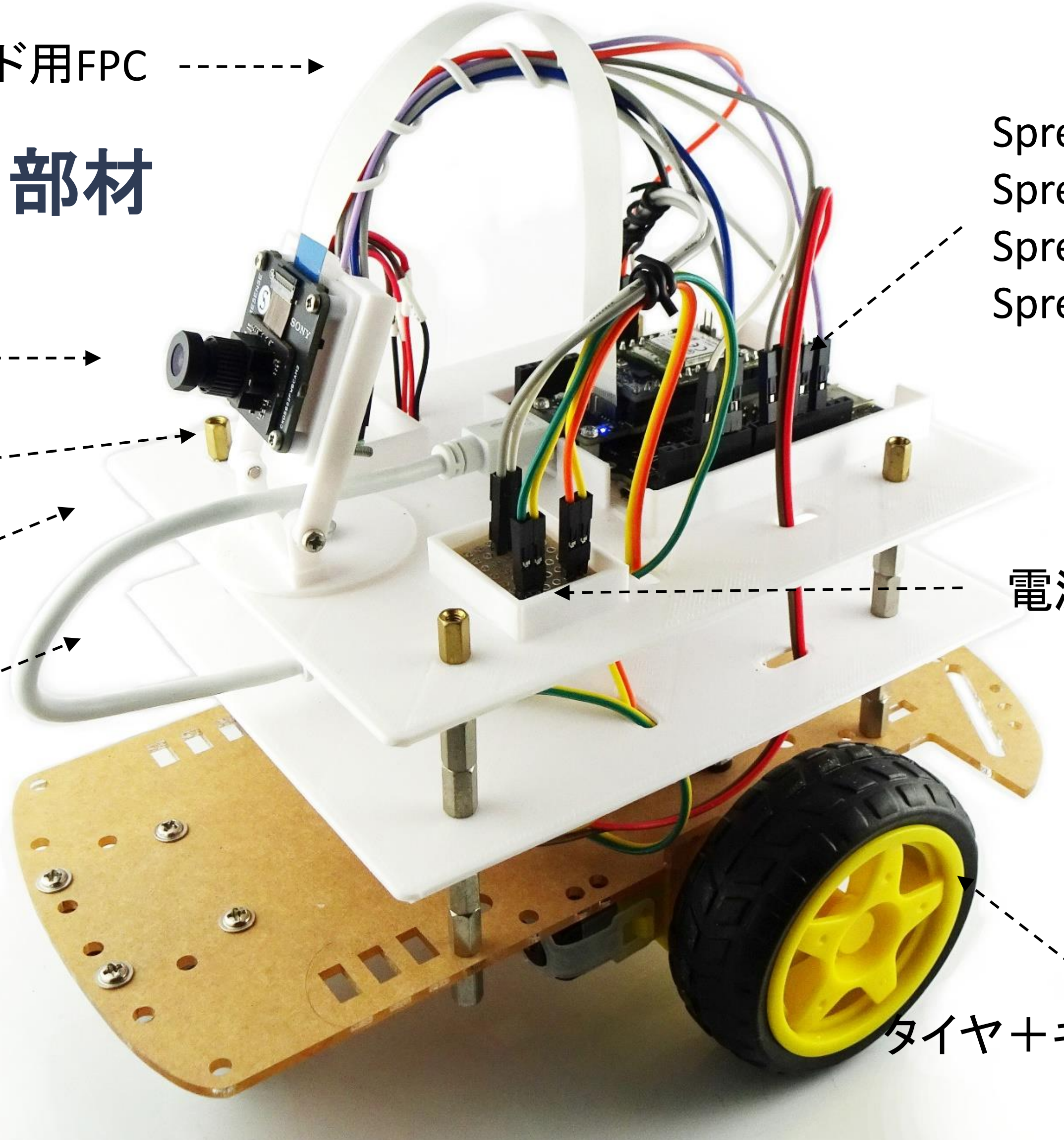
SprTurtleBotの制作

SprTurtleBot0 の使用部材

部品	員数	役割	備考
Spresense メインボード	1	SprTurtleBot0のコントローラ	SPRESENSEメインボード[CXD5602PWBMAIN1]
Spresense 拡張ボード	1	I/Oボード	SPRESENSE拡張ボード[CXD5602PWBEXT1]
Spresense HDRカメラボード	1	SprTurtleBot0のカメラ	SPRESENSE HDRカメラボード[CXD5602PWBCAM2W]
Spresense WiFi アドオンボード	1	ROS2通信用ボード	SPRESENSE Wi-Fi Add-onボード iS110B SPRESENSE用Wi-Fi add-onボード
Spresense センサーアドオンボード	1	加速度・ジャイロセンサー	SPRESENSE用3軸加速度・3軸ジャイロ・気圧・温度センサ
カメラボード用FPC	1	カメラボードとメインボードを接続するFPC	Molex 15166-0123
台座＋モーター＋タイヤキット	1	基本シャーシ	2WDロボットスマートカーシャーシ 2輪駆動 DIY教材
エンコーダー	2	タイヤの回転数をカウント	IR赤外線スロット付きオプトカプラーモーター速度検出
モータードライバ	1	モーター駆動用回路	DRV8835使用ステッピング＆DCモータドライバモジュール
ユニバーサル基板	1	電源分配用基板	両面スルーホールガラスコンポジット・ユニバーサル基板
基板台座	1	Spresense格納用台座	3Dプリント
バッテリー台座	1	バッテリー格納用台座	3Dプリント
その他（ワイヤー、スペーサ、モバイルバッテリー）	多数	ワイヤー（メス・オス）、スペーサー基板台座	

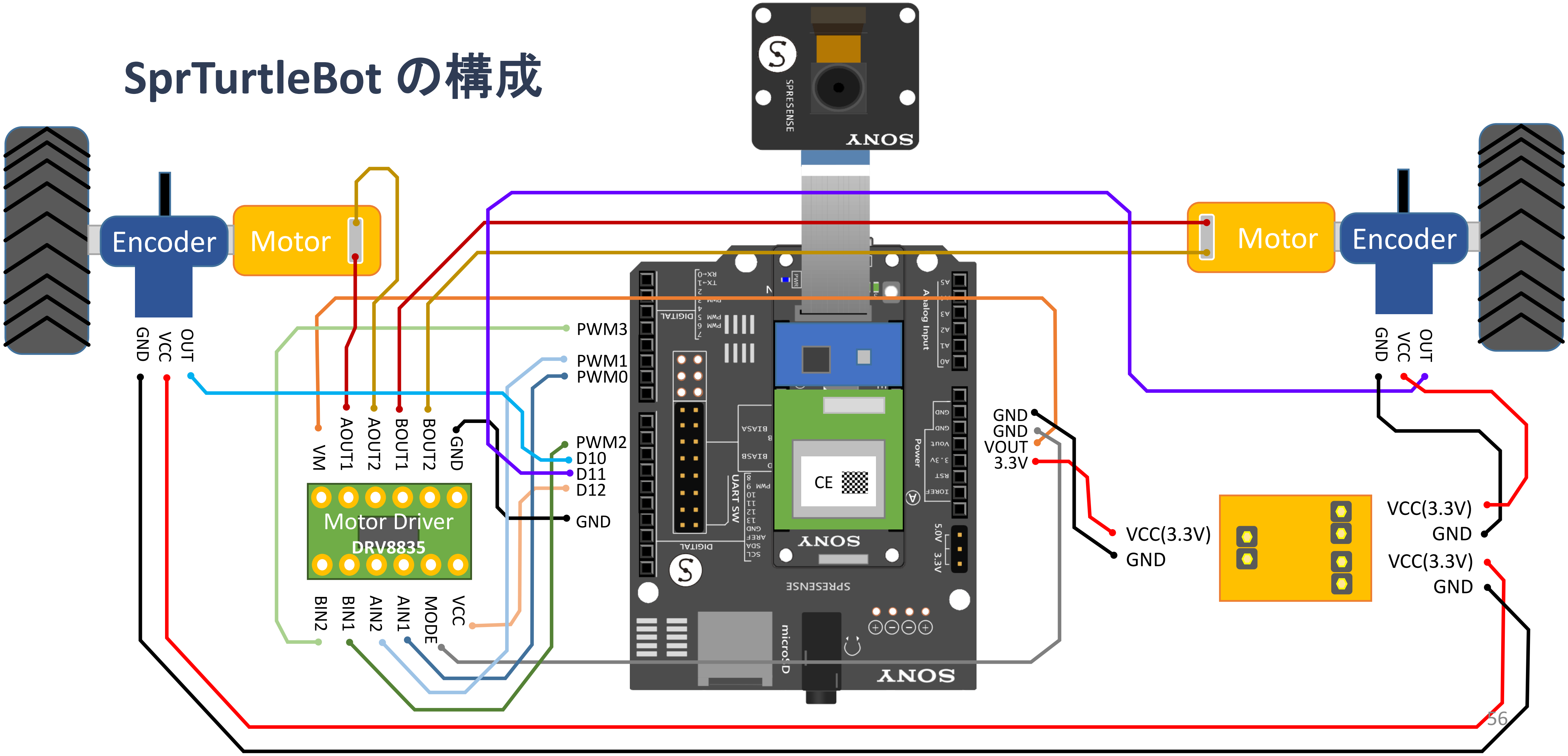
SprTurtleBotの制作

SprTurtleBot0 の使用部材



SprTurtleBotの制作

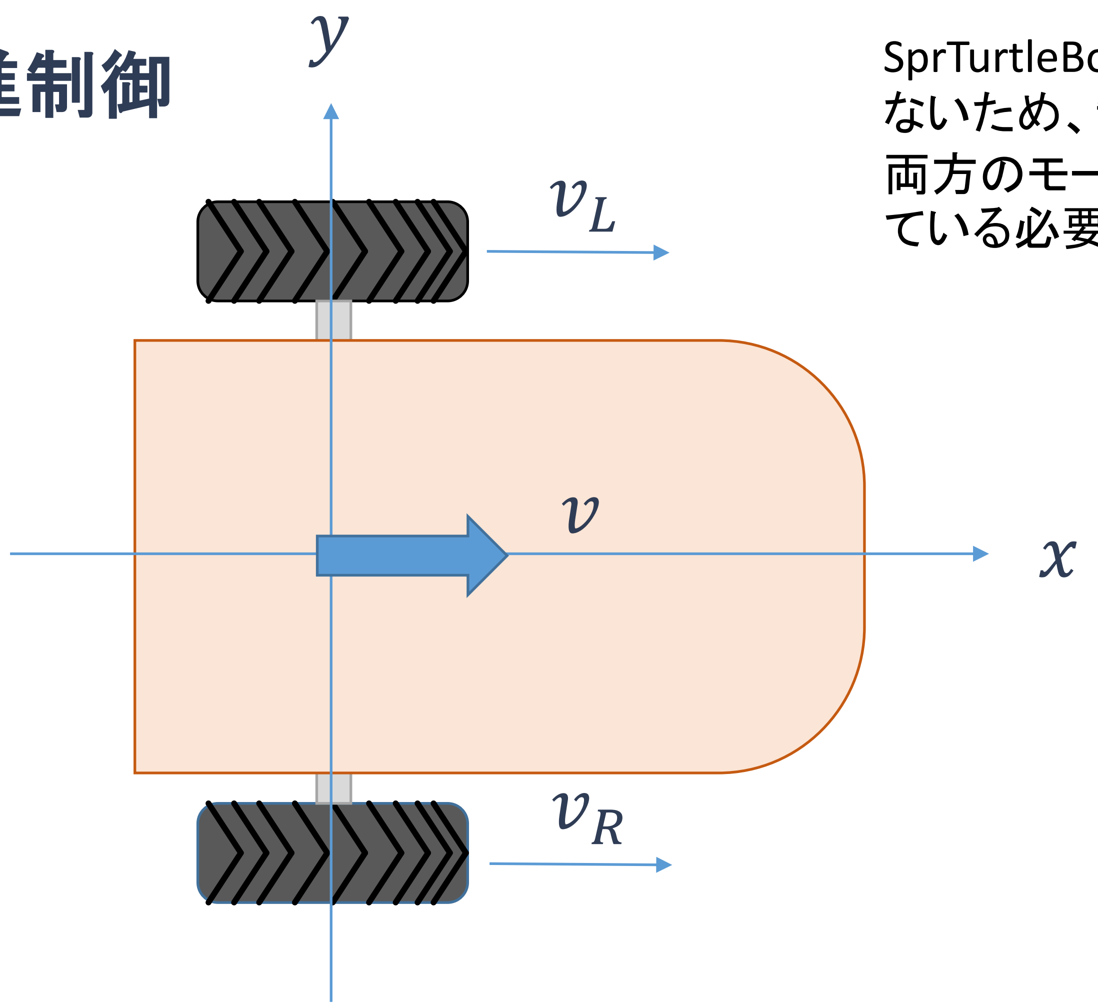
SprTurtleBot の構成



SprTurtleBotの制作

SprTurtleBot の並進制御

$$v = v_L = v_R$$



SprTurtleBotの場合、前後しか動かないため、制御は x 方向のみとなる。両方のモーターが同じ回転数になっている必要がある。

SprTurtleBotの制作

SprTurtleBot の回転制御

$v_R = (L + d)\omega$ ・・・①
$v_L = (L - d)\omega$ ・・・②
$v = L\omega$ ・・・③

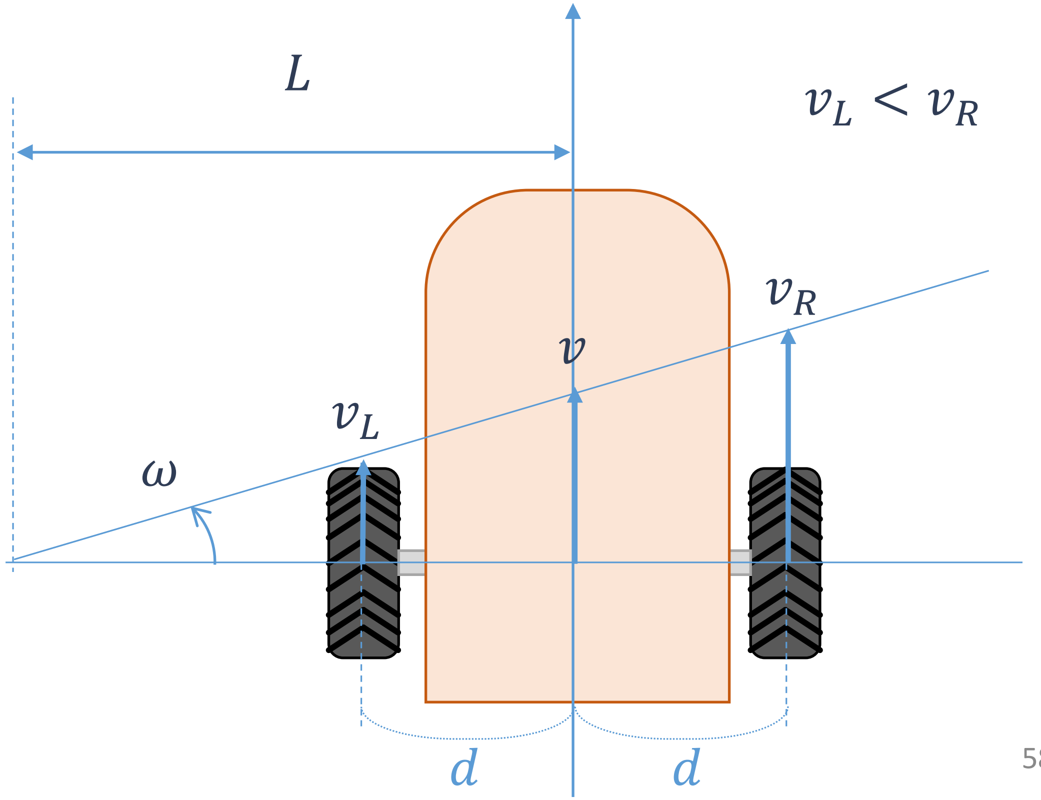
①、②から、Lは未知なのでdについて解くと

$$\omega = \frac{(v_R - v_L)}{2d} \quad \cdots ④$$

$$L = \frac{(v_R + v_L)d}{(v_R - v_L)} \quad \cdots ⑤$$

③、④、⑤から、 $v = \frac{(v_R + v_L)}{2}$

左右の速度が異なると、回転方向の速度ωが発生する。



SprTurtleBotの制作

TurtleBot の自己位置推定

$\Delta\theta$ が小さい場合、$T' \cong T$

$$\Delta x = T' \cos\left(\theta_i + \frac{\Delta\theta}{2}\right) \quad \cdots ⑥$$

$$\Delta y = T' \sin\left(\theta_i + \frac{\Delta\theta}{2}\right) \quad \cdots ⑦$$

一方で、

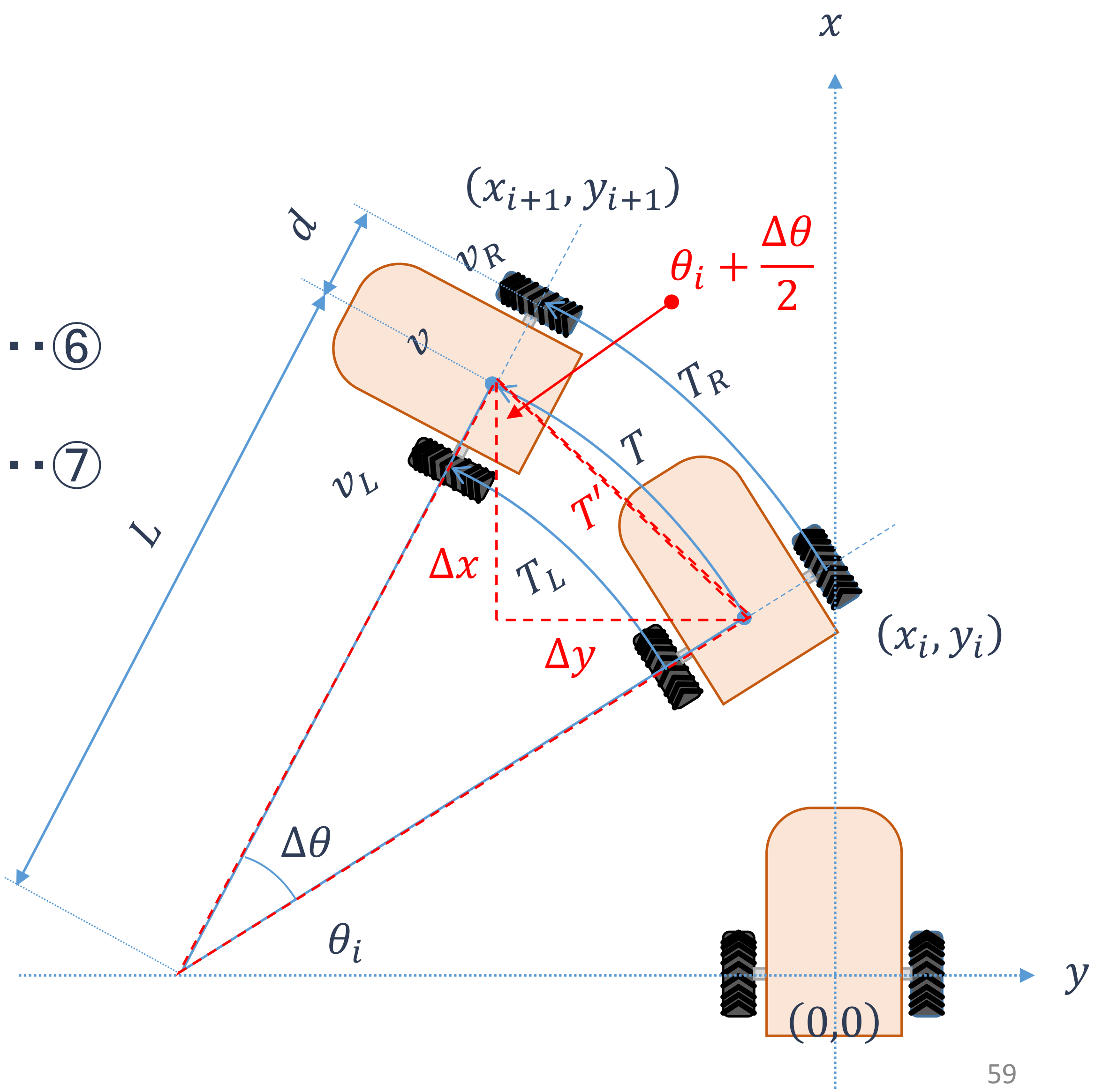
$$T = L\Delta\theta \quad \cdots ⑧$$

$$\Delta\theta = \omega\Delta t \quad \cdots ⑨$$

式④、⑤と、
式⑧、⑨から

$$T = \frac{(v_R + v_L)\Delta t}{2} \quad \cdots ⑩$$

$$\Delta\theta = \frac{(v_R - v_L)\Delta t}{2d} \quad \cdots ⑪$$

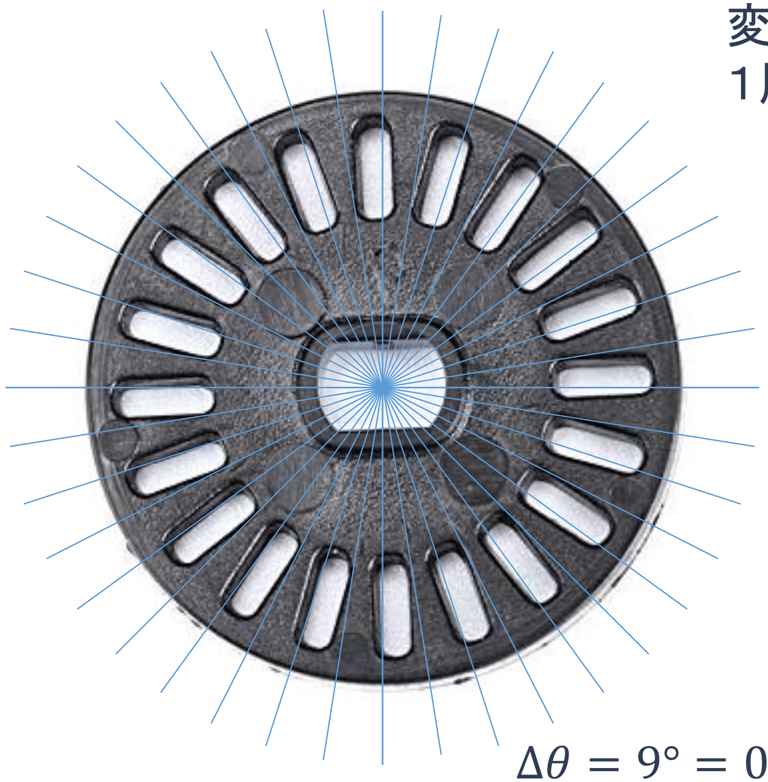


SprTurtleBotの制作

SprTurtleBot の速度の算出

$$v = \frac{\Delta\theta \times EncCount \times D}{\Delta t}$$

変化があったときにカウント
1周（40カウント）

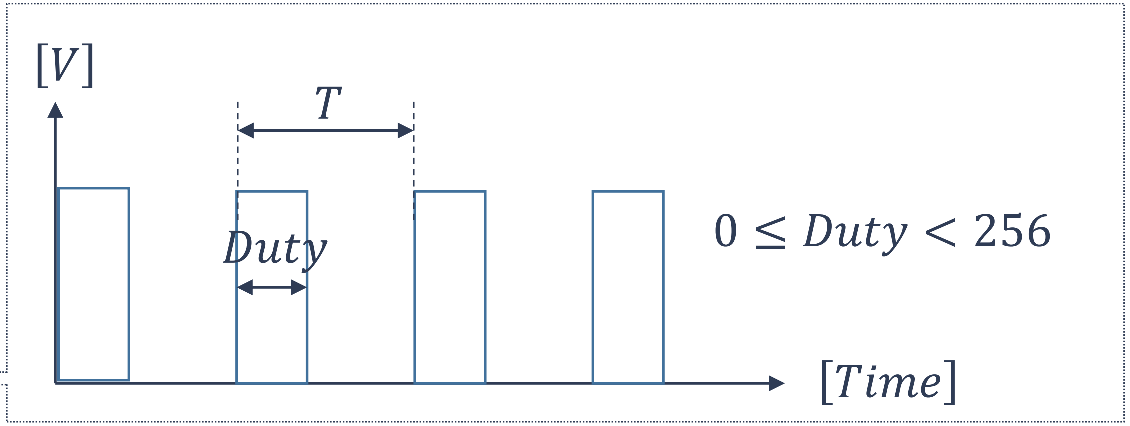


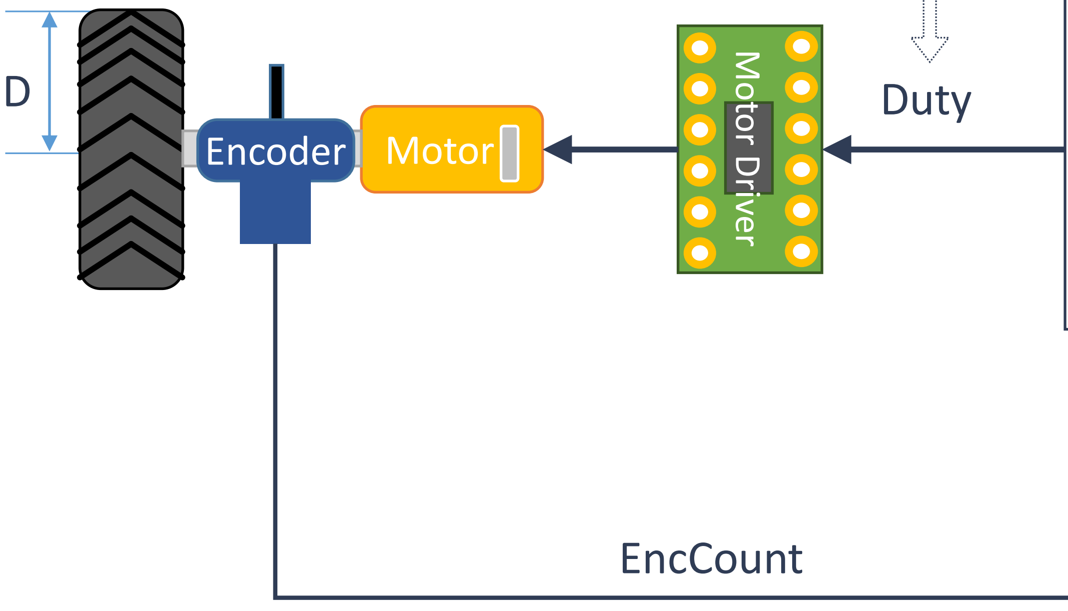
$\Delta\theta = 9° = 0.15707963\ (rad)$

$D = 0.06m$

SprTurtleBotの制作

SprTurtleBot の速度制御





ターゲット速度：v_t

$$\mathrm{Duty} = K_p \dot{v}[n] + K_I \sum_{i=0}^{n} \dot{v}[i] + Kd(\dot{v}[n] - \dot{v}[n-1])$$

$$\dot{v}[i] = v_t - v[i]$$

$$n > 1$$

$v[n]$

$$v = \frac{\Delta\theta \times EncCount \times D}{\Delta t}$$

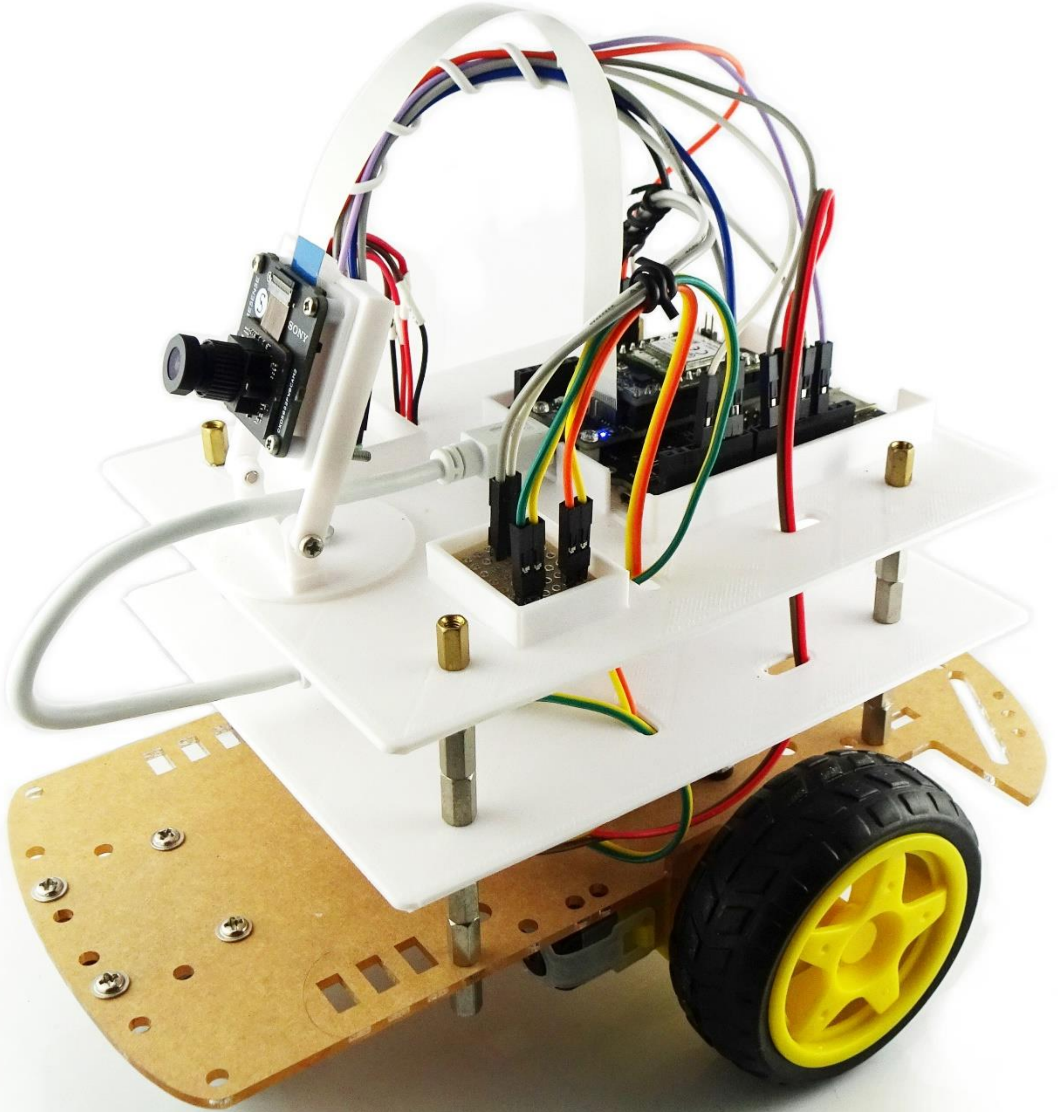
SprTurtleBotの制作

SprTurtleBot のPID係数の調整

- ブラシモーターは個体ばらつきが大きいため、左右が同じレスポンスとなるよう個別に調整
- 調整は、自重による必要トルクを考慮するため、必ず地面に走らせた状態で行う
- 調整値の目安は入力と出力の桁数を参考にする（調整値桁数＝出力桁数／入力桁数）
- 初動時は摩擦が大きくトルクが必要になるため、出力トルクがあがるように調整

SprTurtleBotの制作

PID係数の調整



SprTurtleBotの制作

SprTurtleBot のPID調整プログラムの構成

Main Core

Main_Rover_PID_Adjustment.ino

pwm_control.ino

Sub Core

Sub_WiFi_connection.ino

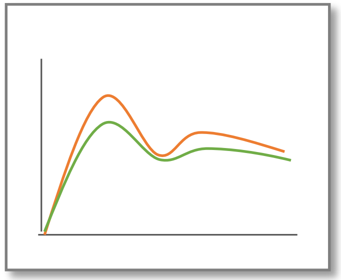
wifi_control.ino

左右Duty設定値

PID係数
ターゲット速度

Ubuntu 22.04LTS (jelly fish)

pid_visualizer.py



Duty設定値グラフ

R_KP:右輪P係数
R_KI:右輪I係数
R_KD:右輪D係数
L_KP:左輪P係数
L_KI:左輪I係数
L_KD:左輪D係数
Vt:ターゲット速度

SprTurtleBotの制作

Main_Rover_PID_Adjustment.inoの実装（抜粋）

```
float R_Kp = KP;  float R_Ki = KI;  float R_Kd = KD;
float L_Kp = KP;  float L_Ki = KI;  float L_Kd = KD;
float R_Vt = 0.0;  float L_Vt = 0.0;

volatile uint32_t R = 0;
volatile uint32_t L = 0;
void Encoder0() {  ++R; }
void Encoder1() {  ++L; }

void setup() {
  …
  attachInterrupt(R_EN, Encoder0, CHANGE);   // 右エンコーダに変化があったらカウント
  attachInterrupt(L_EN, Encoder1, CHANGE);   // 左エンコーダに変化があったらカウント
  …
}
```

エンコーダーの変化をカウント

```
float calc_speed(uint32_t enc_count, uint32_t duration_ms, float* mileage) {
  static const float D  = 0.0325;                  // tire radius (m)
  static const float Enc_Theta = 9.0;              // a unit degree of the edge encoder
  float rotation_ = enc_count*Enc_Theta;           // rotation (degree)
  float rot_radian_ = PI*rotation_/180.0;          // convert dgree to radian
  float mileage_ = rot_radian_*D;                  // (m)
  float tire_speed_ = mileage_*1000.0/(float)(duration_ms);
  *mileage = mileage_;
  return tire_speed_;
}
```

タイヤ速度計算

```
void loop() {
  …
  uint32_t current_time = millis();
  uint32_t duration = current_time - last_time;
  last_time = current_time;

  noInterrupts();
  uint32_t cur_R = R; R = 0;  uint32_t cur_L = L; L = 0;  // カウンターをリセット
  interrupts();
  float R_Vm = calc_speed(cur_R, duration, &R_mileage);          // 右輪スピード
  float L_Vm = calc_speed(cur_L, duration, &L_mileage);          // 左輪スピード
  …
  float duration_sec = duration/1000.0;
  float R_err = R_Vt - R_Vm;
  float L_err = L_Vt - L_Vm;
  cache_R_err_integ += (R_err + R_last_err)*0.5*duration_sec;
  cache_L_err_integ += (L_err + L_last_err)*0.5*duration_sec;
  float R_derr = (R_err - R_last_err) /duration_sec;
  float L_derr = (L_err - L_last_err) /duration_sec;
  int32_t R_duty = (int32_t)(R_Kp*R_err + R_Ki*cache_R_err_integ + R_Kd*R_derr);
  int32_t L_duty = (int32_t)(L_Kp*L_err + L_Ki*cache_L_err_integ + L_Kd*L_derr);
  R_last_err = R_err; L_last_err = L_err;
  …
  pwm_control(R_duty);
  pwm_control(L_duty);
  …
}
```

PID制御

SprTurtleBotの制作

pid_visualizer.py の実装（抜粋）

```
…
history = collections.deque(maxlen=100)

def recv_run():
  while True:
      data = client.recv(buffer_size)
      data = data.decode()
      l = [int(y.strip()) for y in data.split(',')]
      history.append(l)
      time.sleep(0.01)
```

SprTrutlBotから送られてくるデータを受信

```
def cmd_input():
  while True:
    line = input("input: ")
    client.send(bytes(line, 'utf-8'))
    print("[*] Send Data : {}".format(line))
```

ユーザーが入力したパラメータを取得・送信

```
if __name__=="__main__":
  client, address = tcp_server.accept()
  print("Connected!! [ SprTrutleBot IP : {}]".format(address))

  t1 = threading.Thread(target=cmd_input)
  t1.start()
```

ユーザー入力スレッドを開始

```
  plt.ion()
  fig,ax = plt.subplots()

  t2 = threading.Thread(target=recv_run)
  t2.start()
```

データ受信スレッドを開始

```
  while True:
    try:
      for i in range(30): # frame
        x = list(range(i-len(history), i))
        plt.plot(x,history)
        plt.xlabel("frame")
        plt.ylabel("torque")
        plt.draw()
        plt.pause(0.1)
        plt.cla()
    except KeyboardInterrupt:
      plt.close()
```

グラフ描画

SprTurtleBotの制作

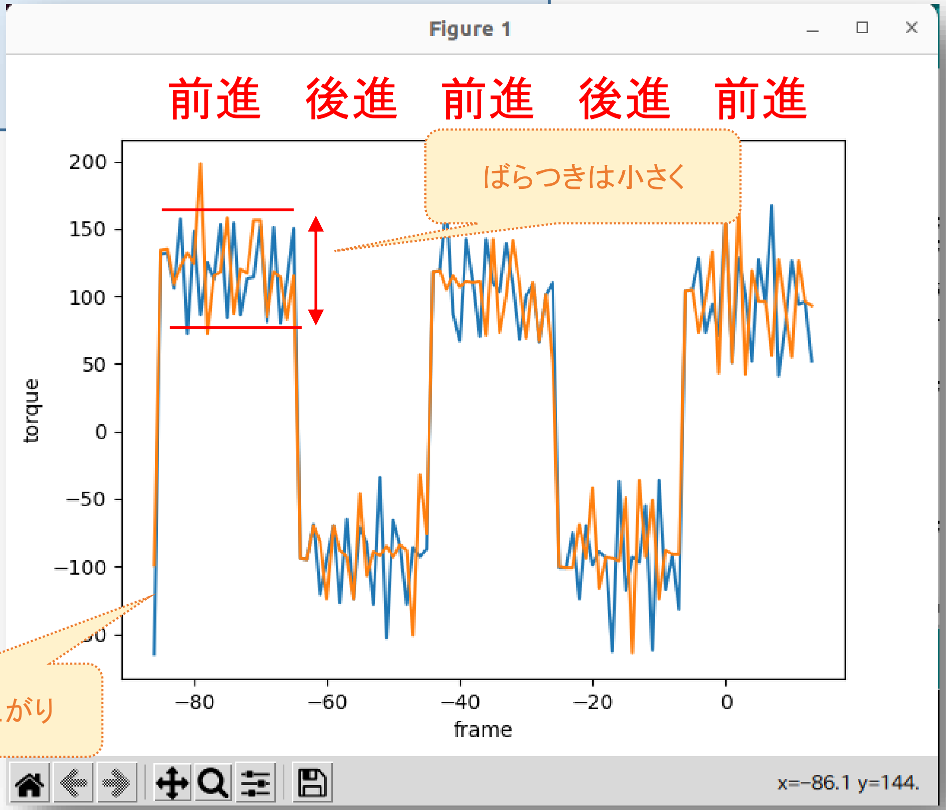
```
$    python3 pid_visualizer.py
Wainting for SprTurtleBot access
Please reset SPrTurtleBot and wait for a moment…
Connected!! [192.168.2.105]
input: 200,30,5,200,30,5,1.0,0
```

ローバーをリセットすると「Connected!!」のメッセージが現れグラフが表示されます

PIDの係数を設定します。引数は次の順番になっている。

R_Kp, R_Ki, R_Kd, L_Kp, L_Ki, L_Kd, Speed(m/s), Rot

Speed は、正の値を指定すると直進、負の値を指定すると後進します。
Rotは、0:直進, 1:逆時計方向回転, -1:時計方向回転になります。



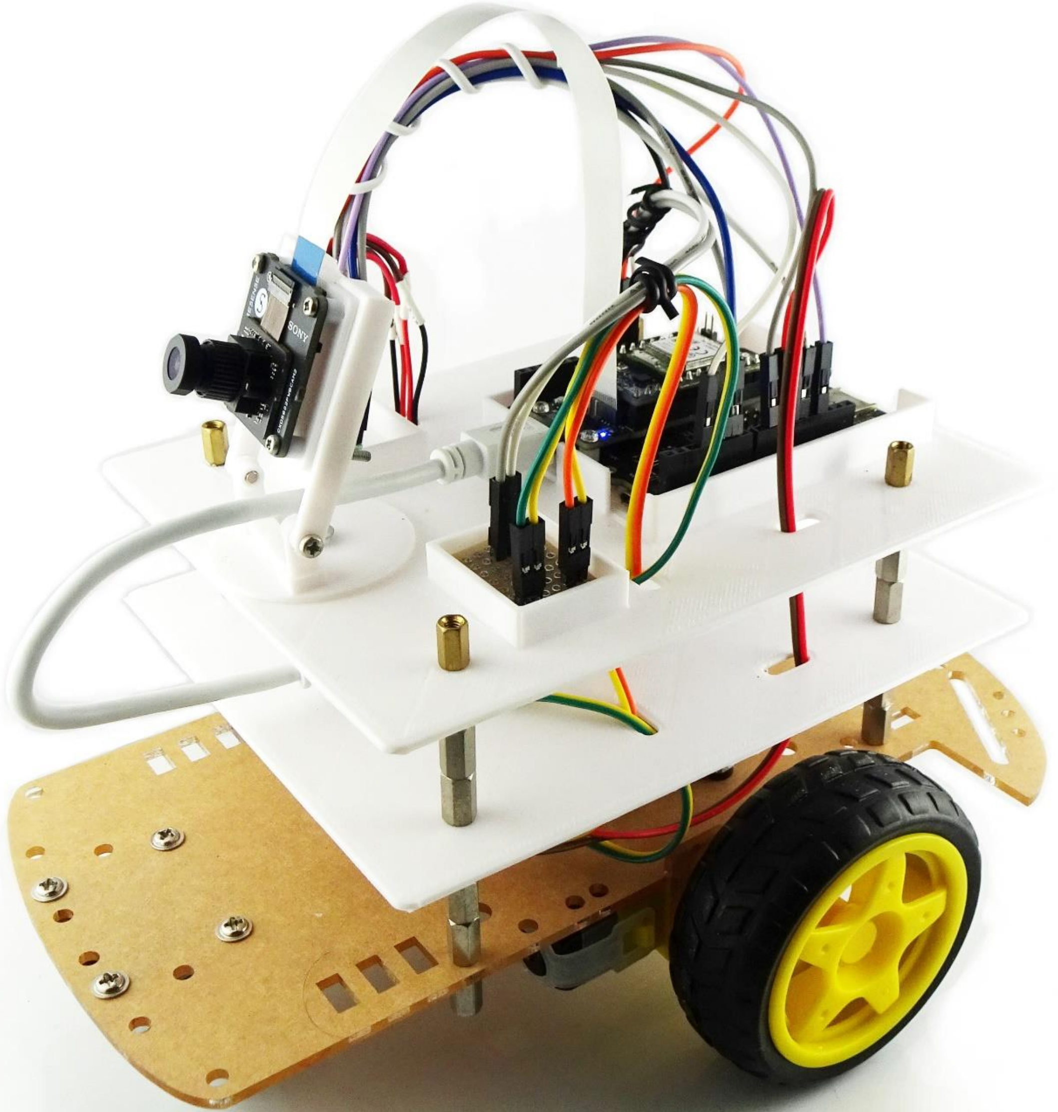
急峻な立ち上がり

係数の値は出力値と入力値のレンジを参考にすると素早く値を見つけられる。この場合、出力レンジ(0-255)、入力レンジ（1.0-5.0m/sとすると） Kpの係数は200程度とあたりをつけることができる。係数はKpの値のみで調整し(Ki,Kdは一旦ゼロ)、その後に波形を見ながら、Ki,Kdの値を調整する。

SprTurtleBotの制作

ROSとの接続

micro-ROS puts ROS 2 onto microcontrollers



SprTurtleBotの制作

SprTurtleBot の動作仕様

動作	Topic名	pub/sub	メッセージ型	備考
モーターオン	motor_power	subscriber	std_msgs/msg/Bool	モータードライバのOn/Offを指示
リセット	reset	subscriber	std_msgs/msg/Empty	システムの再起動を指示
進行制御	cmd_vel	subscriber	geometry_msgs/msg/Twist	並進、回転速度を制御を指示
写真撮影	take_picture	subscriber	std_msgs/msg/Empty	写真撮影を指示
写真データ	image/compressed	publisher	sensor_msgs/msg/CompressedImage	写真データを配信
オドメトリ	odom	publisher	nav_msgs/msg/Odometry	エンコーダから得られたオドメトリデータを配信

SprTurtleBotの制作

SprTurtleBot のプログラムの構成

Main Core

Main_micro-ROS_subpub

Sub Core

Sub_Rover_Control.ino

pwm_control.ino

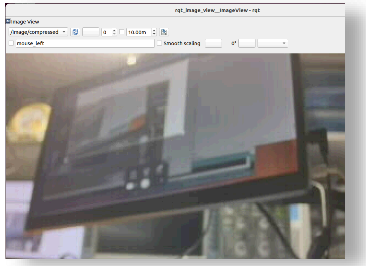
写真データ
オドメトリ

モーターオン
リセット
進行制御
写真撮影

Ubuntu 22.04LTS (jelly fish)

ROS2

写真データ

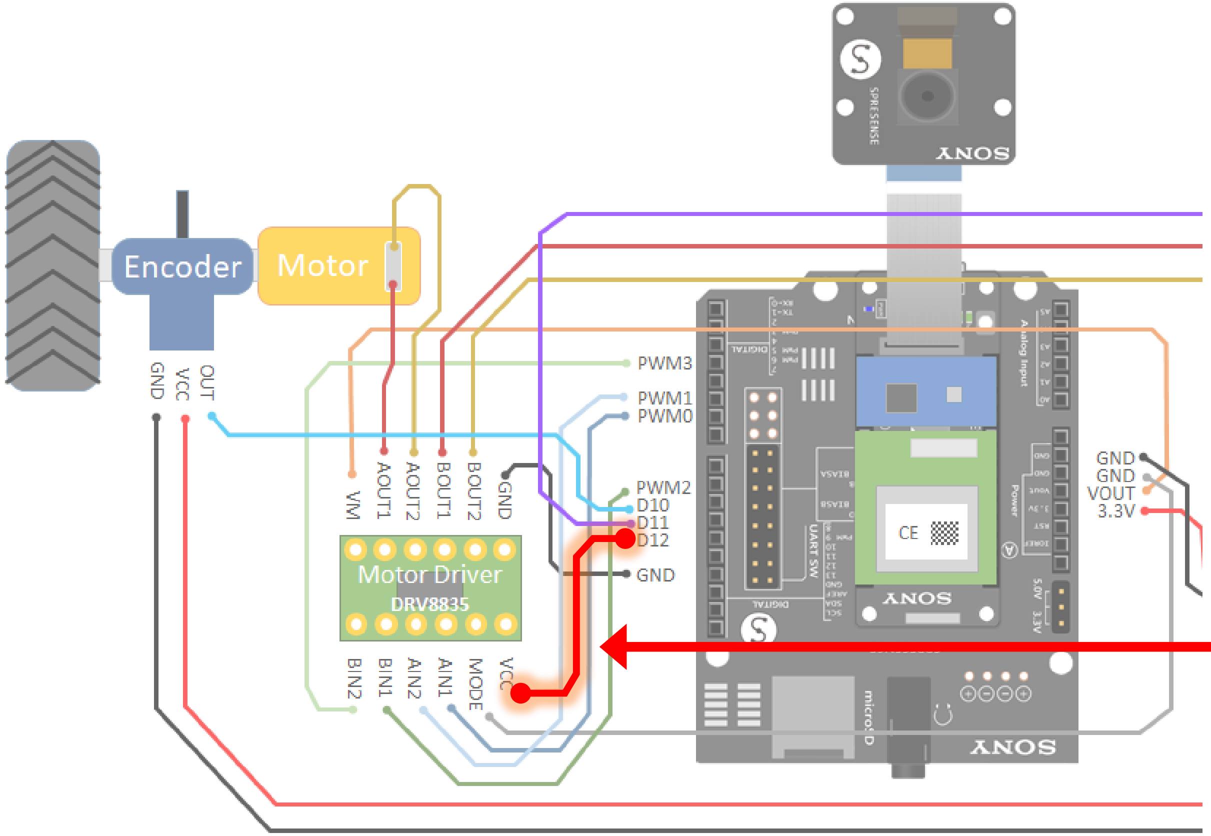


rqt_image_view

モーターオン
リセット
進行制御
写真撮影

SprTurtleBotの制作（モータースイッチ）

モータースイッチの制御



D12ピンを HIGH にすると、ロジック電源が
オンになり、モータードライバが有効になる

SprTurtleBotの制作（モータースイッチ）

Main_micro-ROS_subpub.ino の実装（抜粋）

```
…
static rcl_subscription_t   msw_subscriber;  // motor switch subscriber
…

void msw_callback(const void * msgin) {
  std_msgs__msg__Bool* msg = (std_msgs__msg__Bool*)msgin;
  int8_t sndid = MOTOR_POWER_MSG;
  static std_msgs__msg__Bool msgout;
  memcpy(&msgout, msg, sizeof(std_msgs__msg__Bool));
  MP.Send(sndid, &msgout, subcore);
}
```

ROS2からモーターオンが来たらサブコアに伝える

```
void setup() {
…
  rclc_subscription_init_default(&msw_subscriber, &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Bool), "motor_power");

  rclc_executor_init(&executor, &support.context, 5, &allocator);
  rclc_executor_add_subscription(
    &executor, &msw_subscriber,  &msg, &msw_callback, ON_NEW_DATA));
…
}

void loop() {
  delay(100);
  rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));
}
```

モータースイッチ Subscriber の登録

Sub_Rover_Control.ino の実装（抜粋）

```
#define MOTOR_SW 12

void loop() {
  int8_t recvid; void* msgin;
  int ret = MP.Recv(&recvid, &msgin);
  if (ret > 0) {
    if (recvid == MOTOR_POWER_MSG) {
      std_msgs__msg__Bool* msg = (std_msgs__msg__Bool*)msgin;
      if (msg->data == true && motor_power == false) {
        digitalWrite(MOTOR_SW, HIGH);                  // MOTOR SW ON
        motor_power = true;
        R_Vt = L_Vt = 0.0;
        R_err_integ = L_err_integ = 0.0;
      } else if (msg->data == false && motor_power == true) {
        digitalWrite(MOTOR_SW, LOW);                   // MOTOR SW OFF
        motor_power = false;
        VRt = 0.0; VLt = 0.0;
        R_err_integ = L_err_integ = 0.0;
      }
    …
    }
  …
  }
…
}
```

モータースイッチ（D12ピン）の制御

SprTurtleBotの制作（モータースイッチ）

ホストPCで動作を確認する

モータースイッチオン

```
$    source /opt/ros/humble/setup.bash
$    ros2 topic list
motor_power
$    ros2 topic pub  --once /motor_power std_msgs/msg/Bool "{data: 1}"
```

モータースイッチオフ

```
$    source /opt/ros/humble/setup.bash
$    ros2 topic list
motor_power
$    ros2 topic pub  --once  /motor_power std_msgs/msg/Bool "{data: 0}"
```

SprTurtleBotの制作（リセット制御）

Main_micro-ROS_subpub.ino の実装（抜粋）

```
…
static rcl_subscription_t   res_subscriber;  // reset subscriber
…

void res_callback(const void * msgin) {
  LowPower.reboot();                                   // システムリブート
}
                                   ROS2からリセット要求が来たらシステムをリセット

void setup() {
  LowPower.begin();
…
  rclc_subscription_init_default(&reset_subscriber, &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Empty, “reset");

  rclc_executor_init(&executor, &support.context, 3, &allocator);
  rclc_executor_add_subscription(
    &executor, &msw_subscriber,  &msg, &res_callback, ON_NEW_DATA));
…                                                      リセット subscriber を登録
}
```

SprTurtleBotの制作（リセット制御）

ホストPCで動作を確認する

システムリセット

```
$    source /opt/ros/humble/setup.bash
$    ros2 topic list
reset
$    ros2 topic pub /reset std_msgs/msg/Empty
```

SprTurtleBotの制作（進行制御）

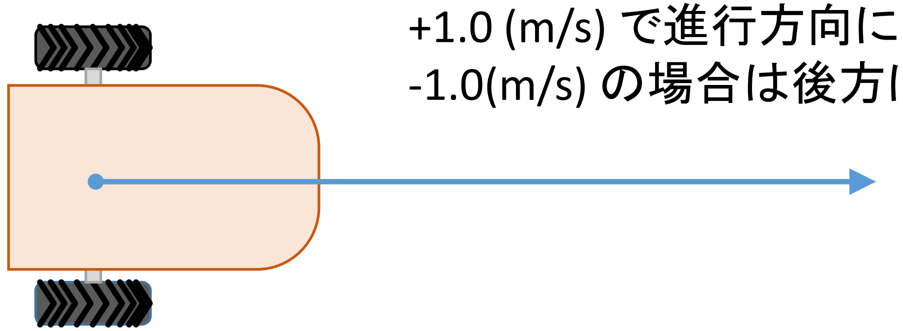
geometry_msgs/msg/Twist に設定する内容

トピック名	指定型	パラメータ	備考	単位
spr_turtle/cmd_vel	linear	x	進行方向もしくは逆方向に指定した速度で進行する	(m/s)
		y	無効	
		z	無効	
	angular	x	無効	
		y	無効	
		z	反時計周りもしくは時計回り方向に指定した角速度で回転する	(rad/s)

SprTurtleBotの制作（進行制御）

geometry_msgs/msg/Twist による制御（並進制御）

```
$    source /opt/ros/humble/setup.bash
$    ros2 topic list
sprturtle/cmd_vel
$    ros2 topic pub --once /spr_turtle/cmd_vel geometry_msg/msg/Twist “{linear: {x: 1.0, y: 0.0, z:0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}”
```

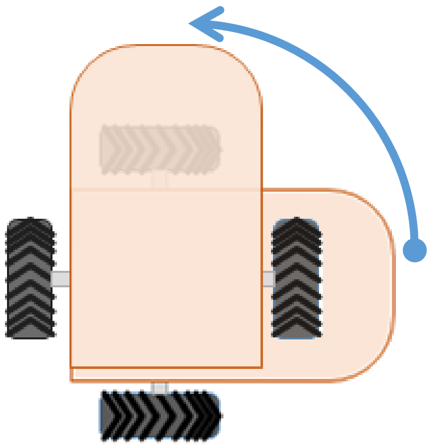


+1.0 (m/s) で進行方向に1秒間進む
-1.0(m/s) の場合は後方に1秒間進む

SprTurtleBotの制作（進行制御）

geometry_msgs/msg/Twist による制御（回転制御）

```
$    source /opt/ros/humble/setup.bash
$    export ROS_DOMAIN_ID=30; export ROS_DISTRO=humble
$    ros2 topic list
sprturtle/cmd_vel
$    ros2 topic pub --once /spr_turtle/cmd_vel geometry_msg/msg/Twist “{linear: {x: 0.0, y: 0.0, z:0.0}, angular: {x: 0.0, y: 0.0, z: 1.57}}”
```



1.57 (rad/s) ≒90 (degree/s) で
反時計方向に1秒間動作する

SprTurtleBotの制作（進行制御）

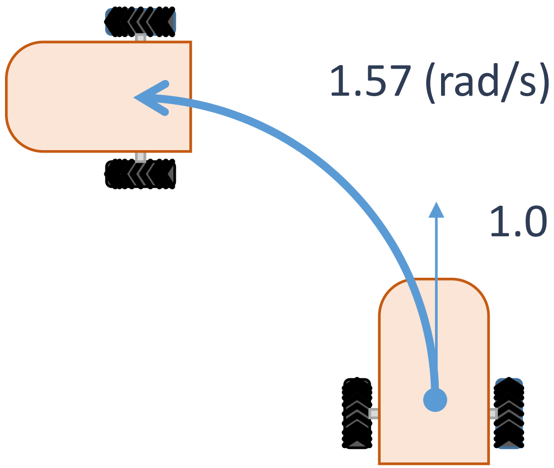
geometry_msgs/msg/Twist による制御（並進＆回転制御）

```
$    source /opt/ros/humble/setup.bash
$    export ROS_DOMAIN_ID=30; export ROS_DISTRO=humble
$    ros2 topic list
sprturtle/cmd_vel
$    ros2 topic pub --once /spr_turtle/cmd_vel geometry_msg/msg/Twist “{linear: {x: 1.0, y: 0.0, z:0.0}, angular: {x: 0.0, y: 0.0, z: 1.57}}”
```

1.0 (m/s) で進行方向に、1.57 (rad/s)の回転速度で動作　→　どういうこと？

SprTurtleBotの制作（進行制御）

geometry_msgs/msg/Twist による制御（並進＆回転制御）



この場合は、 $v_R = 1.21$

$v_L = 0.79$

p.59から $v = \frac{(v_R + v_L)}{2}$ $\omega = \frac{(v_R - v_L)}{2d}$

この2つの式をv_R、v_Lについて整理すると、

$$\boxed{\begin{aligned} v_R &= v + d \times \omega \\ v_L &= v - d \times \omega \end{aligned}}$$

⇦ この式は並進、回転でも使える！

SprTurtleBotの制作（進行制御）

Main_micro-ROS_subpub.ino の実装（抜粋）

```
...
static rcl_subscription_t   cmd_subscriber;  // command subscriber
geometry_msgs__msg__Twist cmd_vel;    // command message
...
void cmd_vel_callback(const void * msgin) {
  geometry_msgs__msg__Twist* cmd_ = (geometry_msgs__msg__Twist*)msgin;
  int8_t sndid = 101;
  static geometry_msgs__msg__Twist cmdout;
  memcpy(&cmdout, cmd_, sizeof(geometry_msgs__msg__Twist));
  MP.Send(sndid, &cmdout, subcore);
}
```

ROS2から来たコマンドの内容をそのままサブコアに伝える

```
void setup() {
...
  RCCHECK(rclc_subscription_init_default(
    &cmd_subscriber,   &node,  ROSIDL_GET_MSG_TYPE_SUPPORT(geometry_msgs, msg, Twist),
    "cmd_vel"));
...
  RCCHECK(rclc_executor_init(&executor, &support.context, 5, &allocator));
  RCCHECK(rclc_executor_add_subscription(&executor, &cmd_subscriber, &cmd_vel,
    &cmd_vel_callback, ON_NEW_DATA));
...
}

void loop() {
 delay(100);
 RCCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```

Sub_Rover_Control.ino の実装（抜粋）

```
...
void loop() {
  int8_t recvid; void* msgin;
  int ret = MP.Recv(&recvid, &msgin);
  if (ret > 0) {
    ... snip ...

    } else if (recvid == COMMAND_MSG) {
      if (motor_power == false)  return;
      geometry_msgs__msg__Twist* cmd = (geometry_msgs__msg__Twist*)msgin;
      VRt = cmd->linear.x + cmd->angular.z*d;
      VLt = cmd->linear.x - cmd->angular.z*d;
      start_time = millis();  // record the start time of the initiation of the move.
    }
```

進行方向と回転速度を取得し、左右のタイヤ速度に変換

```
  }
....
 if (current_time - start_time > 1000) { // stopped after 1sec of the initiation
    pwm_control(0, 0);  pwm_control(1, 0);
    pwm_control(2, 0);  pwm_control(3, 0);
    VRt = VLt = 0.0;    R_Vm = L_Vm = 0.0;   R_duty = L_duty = 0;
    R_last_err = L_last_err = 0.0;    R_err_integ = L_err_integ = 0.0;
    delay(DELAY_TIME);
    return;
  }
```

動作開始後、1秒後に停止

```
  ...
}
```

SprTurtleBotの制作（写真撮影）

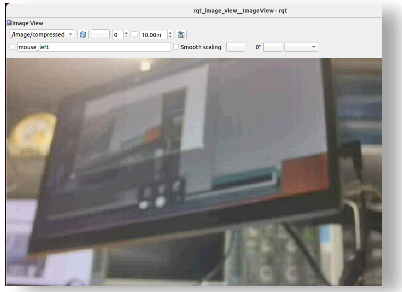
take_picture のトピックを受信したら撮影して画像をパブリッシュ

/take_picture
std_msgs/msg/Empty

ROS2 → Main_micro-ROS_subpub

ROS2 ← Main_micro-ROS_subpub

/image/compressed
sensor_msgs/msg/CompressedImage



rqt_image_view

SprTurtleBotの制作（写真撮影）

Main_micro-ROS_subpub.ino の実装（抜粋）

```
...
static rcl_subscription_t   pic_subscriber;  // camera shutter subscriber
static rcl_publisher_t      img_publisher;   // camera image publisher
std_msgs__msg__Empty pic;            // take picture message
sensor_msgs__msg__CompressedImage msg_static;  // camera image entity
...

void pic_callback(const void * msgin) {
  digitalWrite(LED2, HIGH);
  CamImage img = theCamera.takePicture();
  if (img.isAvailable()) {
    if (img.getImgSize() > msg_static.data.capacity) {
      // Error message: image size is too big
    } else {
      msg_static.data.size = img.getImgSize();
      memset(msg_static.data.data, NULL, img_buffer_size*sizeof(uint8_t));
      memcpy(msg_static.data.data, img.getImgBuff(), img.getImgSize());
      // publish image
      RCCHECK(rcl_publish(&img_publisher, &msg_static, NULL));
    }
  }
  digitalWrite(LED2, LOW);
}
```

ROS2から写真撮影の要求が来たらカメラで撮影し、トピックとして配信する

```
void setup() {
...
RCCHECK(rclc_subscription_init_default(&pic_subscriber,   &node,
  ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Empty),  "take_picture"));
RCCHECK(rclc_publisher_init_default(&img_publisher,  &node,
  ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, CompressedImage),
  "image/compressed"));
```

take_pictureサブスクリプション/imageパブリッシュの生成

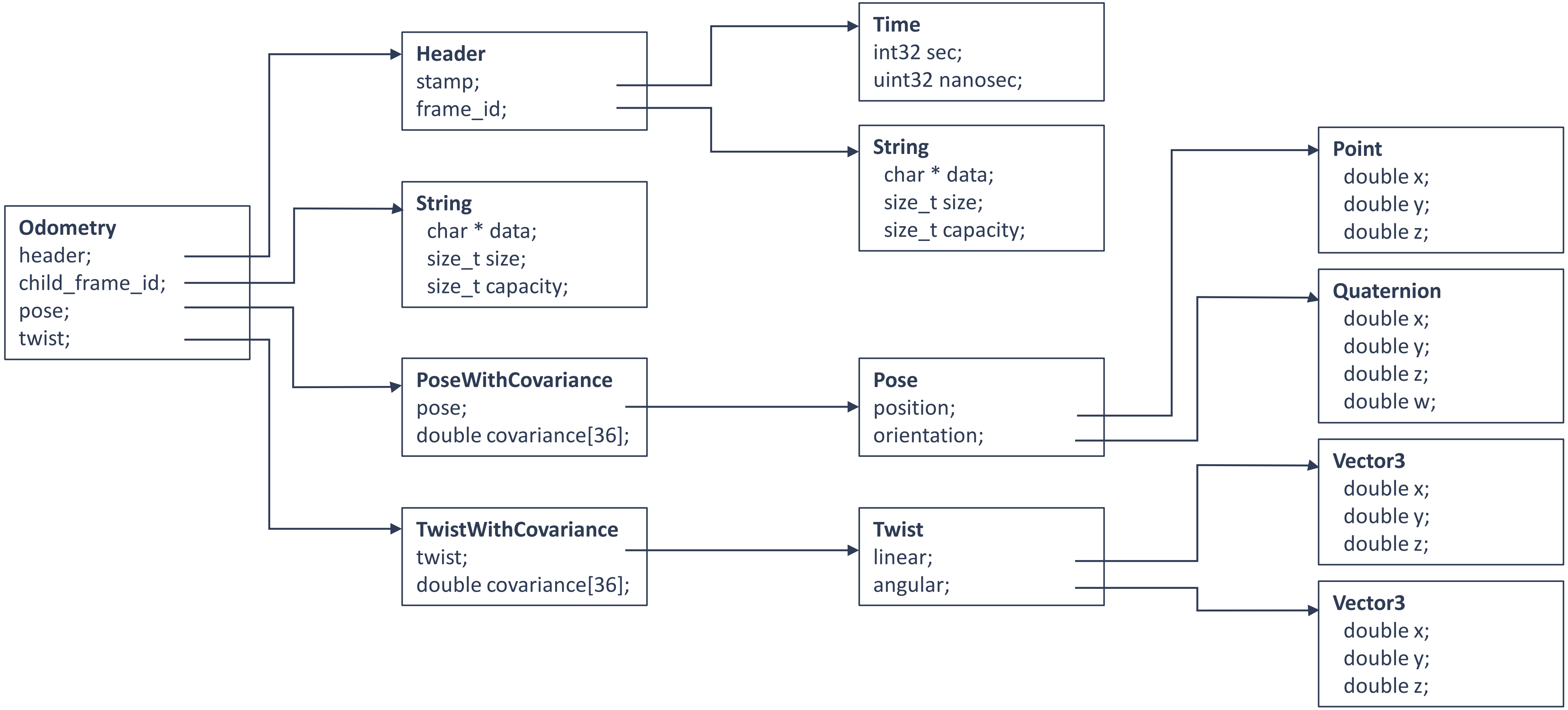
```
 // create buffer for img_publisher
 static uint8_t img_buff[img_buffer_size] = {0};
 msg_static.header.frame_id.data=frame_id_data;
 ....
 msg_static.format.data = format_data;
 sprintf(msg_static.format.data, "jpeg");
 msg_static.format.size = 4;
```

パブリッシュ用画像バッファの確保

```
 RCCHECK(rclc_executor_init(&executor, &support.context, 5, &allocator));
 RCCHECK(rclc_executor_add_subscription(&executor, &pic_subscriber, &pic,
  &pic_callback, ON_NEW_DATA));
 ...
 theCamera.begin();
 theCamera.setStillPictureImageFormat(
  CAM_IMGSIZE_QVGA_H, CAM_IMGSIZE_QVGA_V, CAM_IMAGE_PIX_FMT_JPG);
}
```

SprTurtleBotの制作（オドメトリ）

nav_msgs/msg/Odometry の構造



SprTurtleBotの制作（オドメトリ）

nav_msgs/msg/Odometry に設定する情報

データ	型	設定データ	単位
odometry.header.stamp.sec	int32	起動後の経過時間秒	(sec)
odometry.header.stamp.nanosec	uint32	起動後の経過時間の小数点以下	(nano sec)
odometry.header.frame_id.data	char*	“odom” を設定	
odometry.header.frame_id.size	size_t	4 を設定	
odometry.header.frame_id.capacity	size_t	終端NULLを考慮し、5を設定	
odometry.pose.pose.position.x	double	起動時を原点としたX方向の移動距離	(m)
odometry.pose.pose.position.y	double	起動時を原点としたY方向の移動距離	(m)
odometry.pose.pose.orientation.z	double	クオータニオンのz項	
odometry.pose.pose.orientation.w	double	クオータニオンの2項	
odometry.twist.twist.angular.z	double	起動時からのz方向の回転角	(degree)

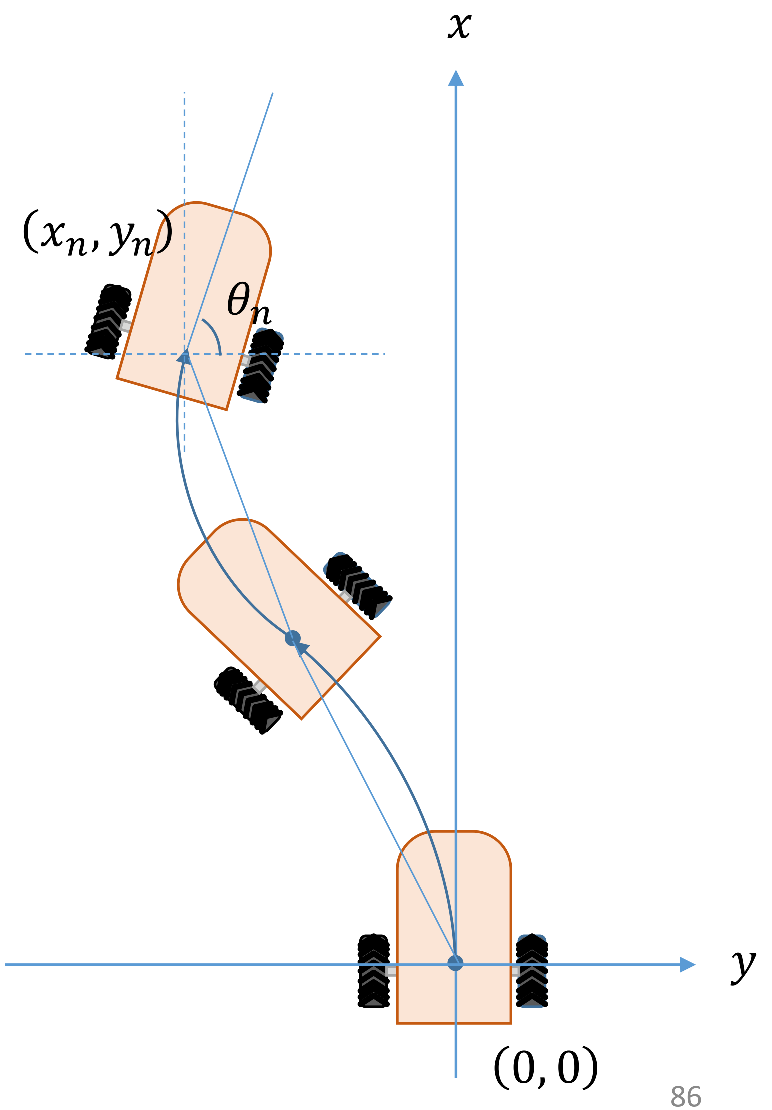
SprTurtleBotの制作（オドメトリ）

p.59の式⑥、⑦、⑩、⑪から、現在をn回目の測定とすると、

$$\theta_n = \sum_{i=0}^{n} \left\{ \frac{(v_{Ri} - v_{Li})\Delta t_i}{2d} \right\}$$

$$x_n = \sum_{i=0}^{n} \left\{ \frac{(v_{Ri} + v_{Li})\Delta t_i}{2} \cos\left(\sum_{j=0}^{i-1} \theta_j + \frac{(v_{Ri} - v_{Li})\Delta t}{4d} \right) \right\}$$

$$y_n = \sum_{i=0}^{n} \left\{ \frac{(v_{Ri} + v_{Li})\Delta t_i}{2} sin\left(\sum_{j=0}^{i-1} \theta_j + \frac{(v_{Ri} - v_{Li})\Delta t}{4d} \right) \right\}$$



またクオータニオンは、X軸、Y軸の回転がないことを前提とすると、次式で表現できる

$$q_{zn} = sin\left(\frac{\theta_n}{2}\right) - cos\left(\frac{\theta_n}{2}\right)$$

$$q_{wn} = cos\left(\frac{\theta_n}{2}\right) + sin\left(\frac{\theta_n}{2}\right)$$

SprTurtleBotの制作（オドメトリ）

Main_micro-ROS_subpub.ino の実装（抜粋）

```
#include <nav_msgs/msg/odometry.h>
static rcl_publisher_t odpm_publisher;
static nav_msgs__msg__Odometry odometry;
#define REQ_ODOM       102
strcut rover_odm {
  float odm_ang_z;
  float odm_pos_x;
  float odm_pos_y;
  float odm_qt_qz;
  float odm_qt_qw;
};
…
void timer_callback(rcl_timer_t * timer, int64_t last_call_time) {
  int8_t sndid=REQ_ODOM;
  struct rover_odm* rover_odm;
  if (timer != NULL) {
    MP.Send(sndid, snd_empty, subcore);
    int ret = MP.Recv(&recvid, &rover_odm, subcore);
    if (ret >= 0) {
      odm.twist.twist.angular.z = rover_odm->odm_ang_z;
      odm.pose.pose.position.x = rover_odm->odm_pos_x;
      odm.pose.pose.position.y = rover_odm->odm_pos_y;
      odm.pose.pose.orientation.z = rover_odm->odm_qt_qz;
      odm.pose.pose.orientation.w = rover_odm->odm_qt_qw;
      …
      RCSOFTCHECK(rcl_publish(&odm_publisher, &odm, NULL));
    }
  }
}
```

オドメトリを１秒毎にパブリッシュするためのタイマー

```
void setup {
  …
  RCCHECK(rclc_publisher_init_default(&odm_publisher, &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(nav_msgs, msg, Odometry), "odom"));
  …
  const uint32_t timer_timeout = 1000;
  RCCHECK(rclc_timer_init_default(&timer, &support, RCL_MS_TO_NS(timer_timeout),
    timer_callback));
  …
  RCCHECK(rclc_executor_init(&executor, &support.context, 5, &allocator));
  RCCHECK(rclc_executor_add_timer(&executor, &timer));
  …
}

void loop() {
  delay(100);
  RCCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```

タイマーを登録

SprTurtleBotの制作（オドメトリ）

Sub_Rover_Control.ino の実装（抜粋）

```
#define REQ_ODOM      102
struct rover_odm {
  float odm_ang_z;
  float odm_pos_x;
  float odm_pos_y;
  float odm_qt_qz;
  float odm_qt_qw;
};
struct rover_odm rover_odm;
…
void loop() {
  int8_t recvid; void* msgin;
  int ret = MP.Recv(&recvid, &msgin);
  if (ret > 0) {
    if (recvid == MOTOR_POWER_MSG) {
      …
    } else if (recvid == COMMAND_MSG) {
      …
    } else if (recvid == REQ_ODOM) {
      int8_t sndid = REQ_ODOM;
      MP.Send(sndid, &rover_odm);
    }                                   メインコアからオドメトリの要求があったら値を返す
   …
   R_Vm = calc_speed(cur_R, duration, &R_mileage, VRt);
   L_Vm = calc_speed(cur_L, duration, &L_mileage, VLt);
```

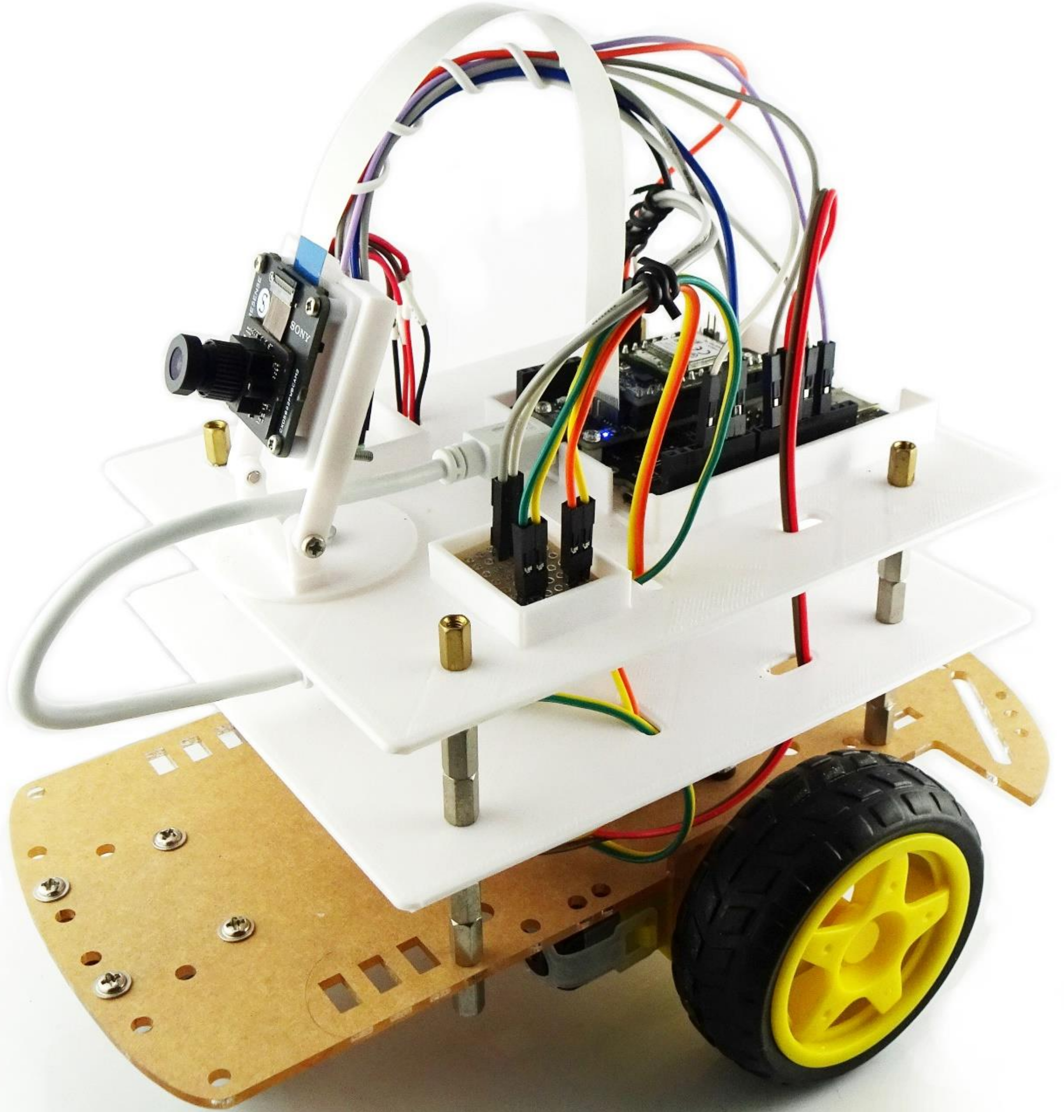
```
 if (abs(R_Vm) > 0.0 || abs(L_Vm) > 0.0) {
   static float odm_ang_z = 0.0;
   static float odm_pos_x = 0.0;
   static float odm_pos_y = 0.0;
   static float odm_qt_qz = 0.0;
   static float odm_qt_qw = 0.0;

   float duration_sec = (float)duration/1000;
   float last_odm_ang_z = odm_ang_z;
   odm_ang_z += (R_Vm - L_Vm)*duration_sec/(2.*d);
   if (odm_ang_z > 2.*PI) odm_ang_z -= 2.*PI;
   odm_pos_x += (R_Vm + L_Vm)*duration_sec/2.*arm_cos_f32(last_odm_ang_z+odm_ang_z/2);
   odm_pos_y += (R_Vm + L_Vm)*duration_sec/2.*arm_sin_f32(last_odm_ang_z+odm_ang_z/2);
   odm_qt_qz = arm_sin_f32(odm_ang_z/2) - arm_cos_f32(odm_ang_z/2);
   odm_qt_qw = arm_cos_f32(odm_ang_z/2) + arm_sin_f32(odm_ang_z/2);

   rover_odm.odm_ang_z = odm_ang_z;
   rover_odm.odm_pos_x = odm_pos_x;
   rover_odm.odm_pos_y = odm_pos_y;
   rover_odm.odm_qt_qz = odm_qt_qz;
   rover_odm.odm_qt_qw = odm_qt_qw;

 }                                   オドメトリの値を計算して、構造体に格納する
 …
 delay(DELAY_TIME);
}
```

SprTurtleBotの制作（オドメトリ）

nav_msgs/msg/Odometry の取得

```
$    source /opt/ros/humble/setup.bash
$    ros2 topic list
odom
$    ros2 topic echo /odom
header:
  stamp:
    sec: 54
    nanosec: 258601
  frame_id: odom
child_frame_id: '‘
pose:
  pose:
    position:
      x: 3.55720114708
      y: 0.655082702637
      z: 0.0
```

```
    orientation:
      x: 0.0
      y: 0.0
      z: 0.113450162113
      w: 0.993543684483
  covariance:  - 0.0  - 0.0 .... - 0.0
twist:
  twist:
    linear:
      x: 0.0
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: -0.00472585950047
  covariance:  - 0.0  - 0.0  ... - 0.0
```

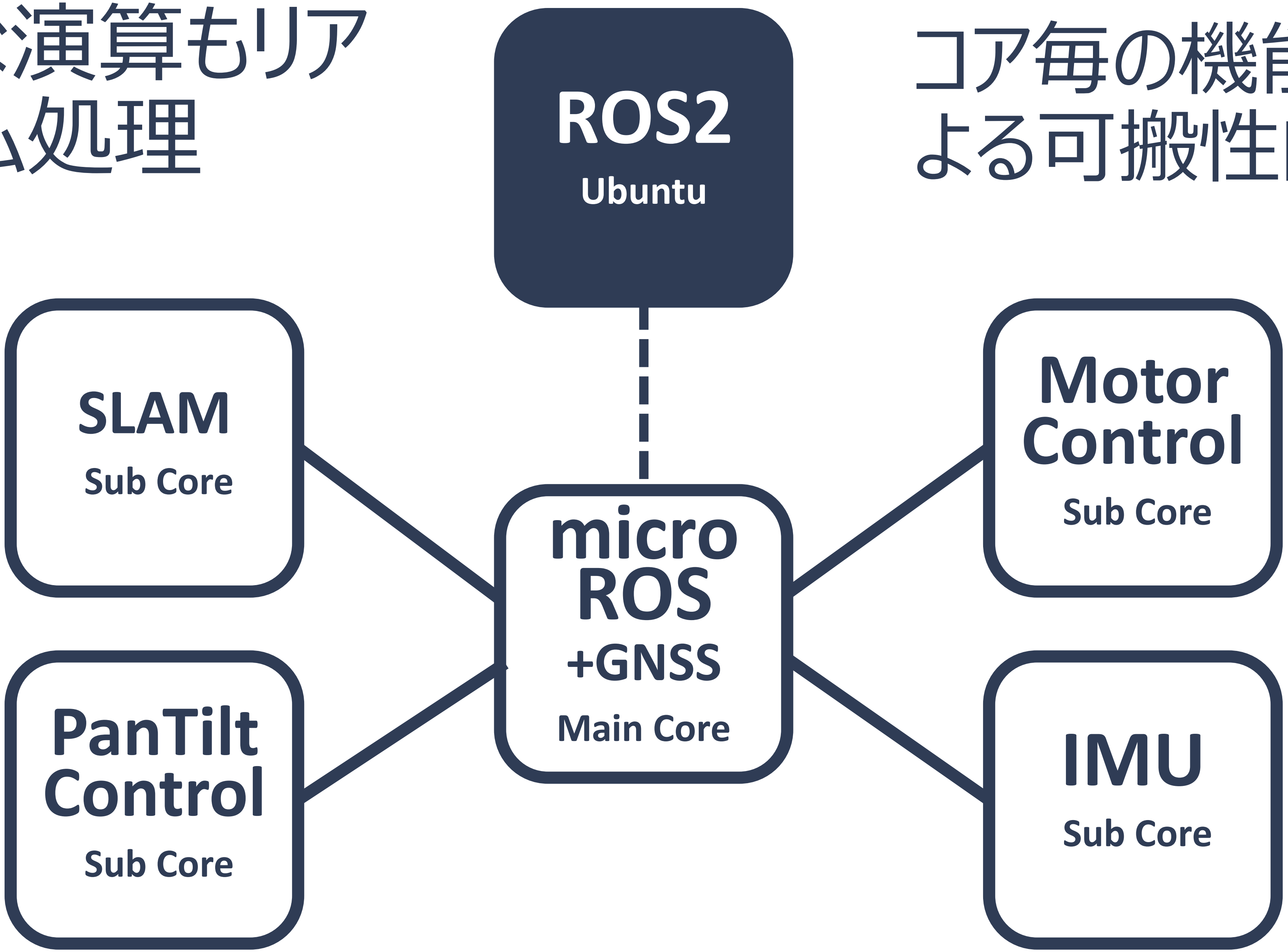


micro-ROS puts ROS 2 onto microcontrollers

SprTurtleBot の完成度を高める

複雑な演算もリアルタイム処理

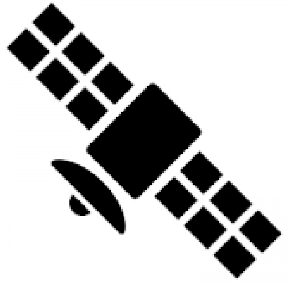
コア毎の機能分散による可搬性向上

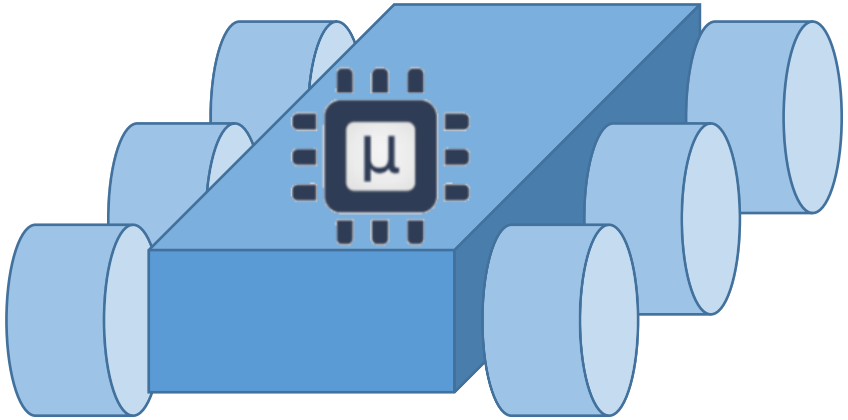


測位機能の拡張

```
sensor_msgs/NavSatFix.msg
uint8 COVARIANCE_TYPE_UNKNOWN=0
uint8 COVARIANCE_TYPE_APPROXIMATED=1
uint8 COVARIANCE_TYPE_DIAGONAL_KNOWN=2
uint8 COVARIANCE_TYPE_KNOWN=3
std_msgs/msg/Header header
sensor_msgs/msg/NavSatStatus status
double latitude
double longitude
double altitude
double[9] position_covariance
uint8 position_covariance_type
```

```
sensor_msgs/NavSatStatus.msg
int8 STATUS_NO_FIX=-1
int8 STATUS_FIX=0
int8 STATUS_SBAS_FIX=1
int8 STATUS_GBAS_FIX=2
uint16 SERVICE_GPS=1
uint16 SERVICE_GLONASS=2
uint16 SERVICE_COMPASS=4
uint16 SERVICE_GALILEO=8
int8 status
uint16 service
```





IMUを活用した機能拡張

sensor_msgs/Imu.msg

```
Header header
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance # Row major about x, y, z axes
geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance # Row major about x, y, z axes
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance # Row major x, y z
```

geometry_msgs/Quaternion.msg

```
float64 x
float64 y
float64 z
float64 w
```

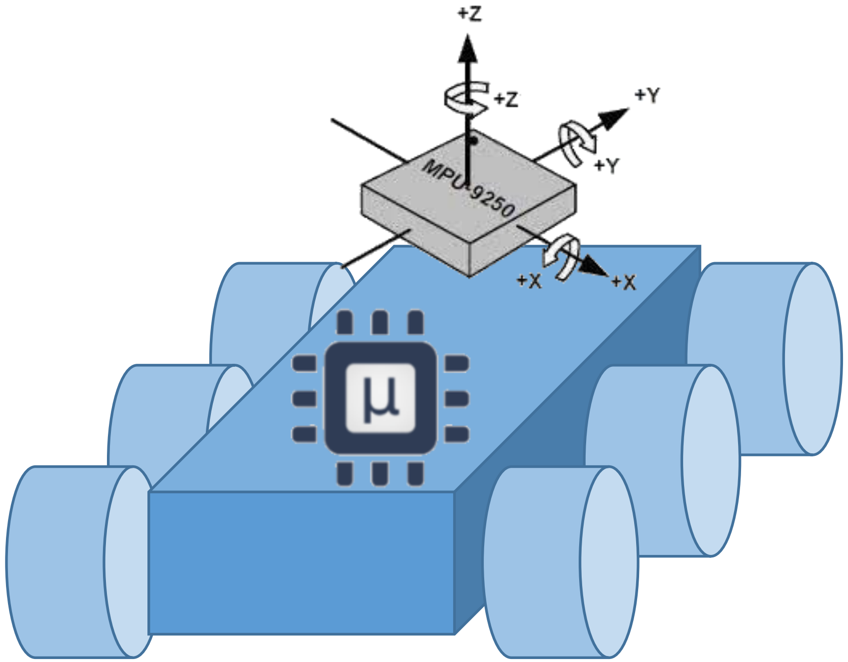
geometry_msgs/Vector3.msg

```
float64 x
float64 y
float64 z
```

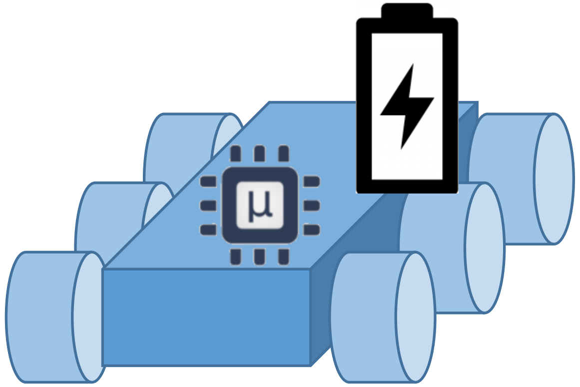
geometry_msgs/Vector3.msg

```
float64 x
float64 y
float64 z
```

IMU（6軸センサー）



バッテリー（+Fuel Gage）



バッテリー管理

sensor_msgs/BatteryState.msg

```
# Power supply status constants
uint8 POWER_SUPPLY_STATUS_UNKNOWN = 0
uint8 POWER_SUPPLY_STATUS_CHARGING = 1
uint8 POWER_SUPPLY_STATUS_DISCHARGING = 2
uint8 POWER_SUPPLY_STATUS_NOT_CHARGING = 3
uint8 POWER_SUPPLY_STATUS_FULL = 4
# Power supply health constants
uint8 POWER_SUPPLY_HEALTH_UNKNOWN = 0
uint8 POWER_SUPPLY_HEALTH_GOOD = 1
uint8 POWER_SUPPLY_HEALTH_OVERHEAT = 2
uint8 POWER_SUPPLY_HEALTH_DEAD = 3
uint8 POWER_SUPPLY_HEALTH_OVERVOLTAGE = 4
uint8 POWER_SUPPLY_HEALTH_UNSPEC_FAILURE = 5
uint8 POWER_SUPPLY_HEALTH_COLD = 6
uint8 POWER_SUPPLY_HEALTH_WATCHDOG_TIMER_EXPIRE = 7
uint8 POWER_SUPPLY_HEALTH_SAFETY_TIMER_EXPIRE = 8
# Power supply technology (chemistry) constants
uint8 POWER_SUPPLY_TECHNOLOGY_UNKNOWN = 0
uint8 POWER_SUPPLY_TECHNOLOGY_NIMH = 1
uint8 POWER_SUPPLY_TECHNOLOGY_LION = 2
uint8 POWER_SUPPLY_TECHNOLOGY_LIPO = 3
uint8 POWER_SUPPLY_TECHNOLOGY_LIFE = 4
uint8 POWER_SUPPLY_TECHNOLOGY_NICD = 5
uint8 POWER_SUPPLY_TECHNOLOGY_LIMN = 6
```

```
Header  header
float32 voltage                 # Voltage in Volts (Mandatory)
float32 temperature             # Temperature in Degrees Celsius (If unmeasured NaN)
float32 current                 # Negative when discharging (A)  (If unmeasured NaN)
float32 charge                  # Current charge in Ah  (If unmeasured NaN)
float32 capacity                # Capacity in Ah (last full capacity)  (If unmeasured NaN)
float32 design_capacity         # Capacity in Ah (design capacity)  (If unmeasured NaN)
float32 percentage              # Charge percentage on 0 to 1 range  (If unmeasured NaN)
uint8   power_supply_status     # The charging status as reported. Values defined above
uint8   power_supply_health     # The battery health metric. Values defined above
uint8   power_supply_technology # The battery chemistry. Values defined above
bool    present                 # True if the battery is present

float32[] cell_voltage          # An array of individual cell voltages for each cell in the pack
                                # If individual voltages unknown but number of cells known
                                # set each to NaN
float32[] cell_temperature      # An array of individual cell temperatures for each cell in the pack
                                # If individual temperatures unknown but number of cells known
                                # set each to NaN
string location                 # The location into which the battery is inserted. (slot number)
string serial_number            # The best approximation of the battery serial number
```

各種センサーのサポート

sensor_msgs/Illuminance.msg

```
Header header                # timestamp is the time the illuminance was measured
                             # frame_id is the location and direction of the reading
float64 illuminance          # Measurement of the Photometric Illuminance in Lux.
float64 variance             # 0 is interpreted as variance unknown
```

sensor_msgs/MagneticField.msg

```
Header header                # timestamp is the time the field was measured
                             # frame_id is the location and orientation of the field measurement
geometry_msgs/Vector3 magnetic_field     # x, y, and z components of the field vector in Tesla
                                         # If your sensor does not output 3 axes,
                                         # put NaNs in the components not reported.

float64[9] magnetic_field_covariance     # Row major about x, y, z axes
                                         # 0 is interpreted as variance unknown
```

sensor_msgs/Temperature.msg

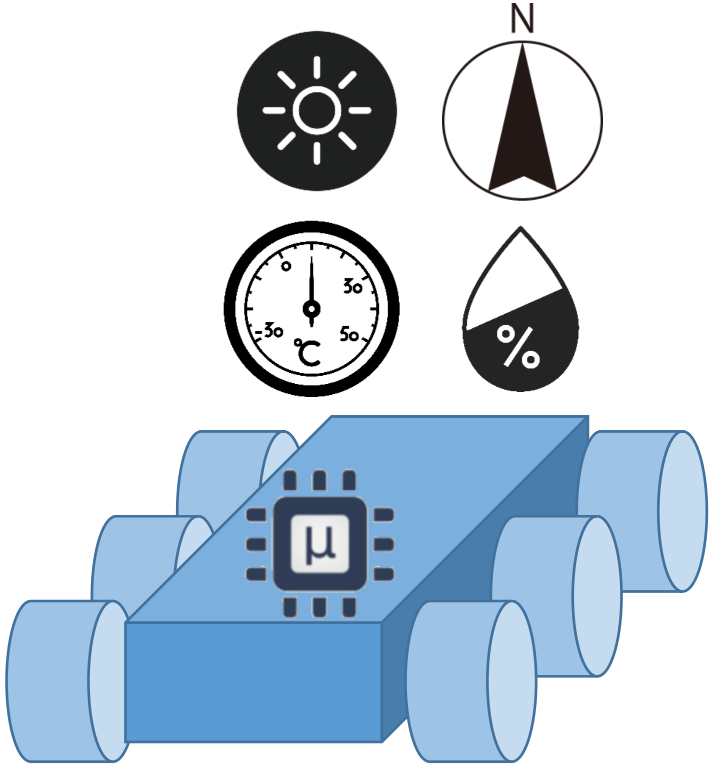
```
Header header                # timestamp is the time the temperature was measured
                             # frame_id is the location of the temperature reading
float64 temperature          # Measurement of the Temperature in Degrees Celsius
float64 variance             # 0 is interpreted as variance unknown
```

sensor_msgs/RelativeHumidity.msg

```
Header header                # timestamp of the measurement
                             # frame_id is the location of the humidity sensor
float64 relative_humidity    # Expression of the relative humidity from 0.0 to 1.0.
                             # 0.0 is no partial pressure of water vapor
                             # 1.0 represents partial pressure of saturation
float64 variance             # 0 is interpreted as variance unknown
```

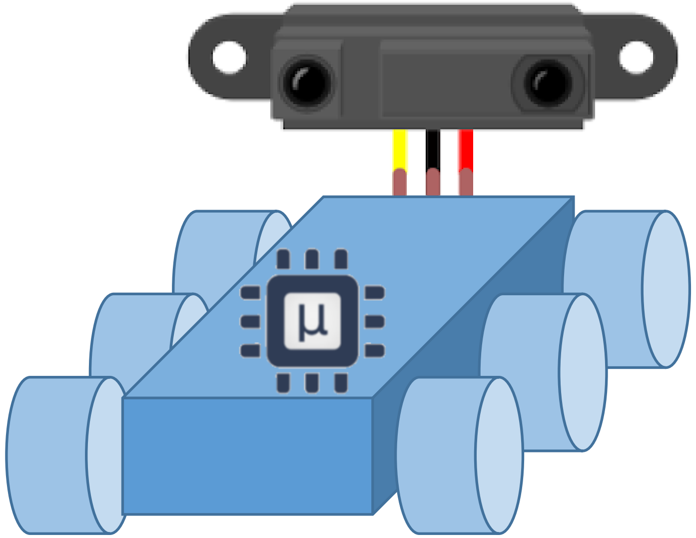
照度計
地磁気センサー

温度計
湿度計



測距センサーのサポート（赤外線、超音波）

```
sensor_msgs/Range.msg
Header header              # timestamp  is the time the ranger   returned the distance reading
uint8 ULTRASOUND=0
uint8 INFRARED=1
uint8 radiation_type        # the type of radiation used by the sensor (sound, IR, etc) [enum]
float32 field_of_view      # the size of the arc that the distance reading is valid for [rad]
                           # 0 angle corresponds to the x-axis of the sensor.
float32 min_range          # minimum range value [m]
float32 max_range          # maximum range value [m]
float32 range              # range data [m]
```



その他の情報の提供

sensor_msgs/TimeReference.msg

```
Header header        # stamp is system time for which measurement was valid
                     # frame_id is not used
time   time_ref      # corresponding time from this external source
string source        # (optional) name of time source
```

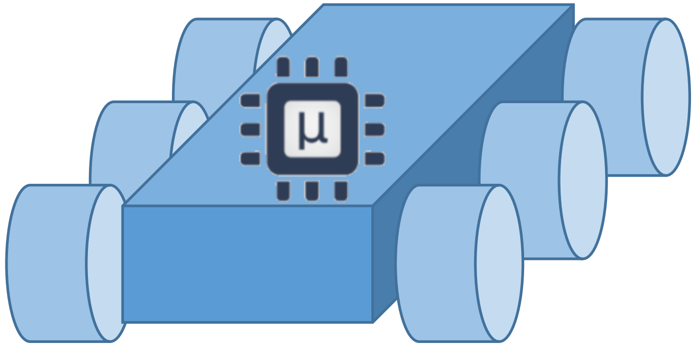
sensor_msgs/CameraInfo.msg

```
Header header        # Header timestamp should be acquisition time of image
uint32 height
uint32 width
float64[] D          # The distortion parameters
float64[9]  K        # 3x3 row-major matrix
float64[9]  R        # 3x3 row-major matrix
float64[12] P        # 3x4 row-major matrix
uint32 binning_x     # binning_x = binning_y = 0 is considered the same
uint32 binning_y     # binning_x = binning_y = 1 (no subsampling).
RegionOfInterest roi
```

sensor_msgs/RegionOfInterest.msg

```
uint32 x_offset
uint32 y_offset
uint32 height
uint32 width
bool do_rectify      # this should be False if the full image is captured (ROI not used),
                     # and True if a subwindow is captured (ROI used).
```

時計（GPS時計、電波時計）
カメラ情報



ROS2/micro-ROSをさらに知るために

ROS2ドキュメント（humble）
https://docs.ros.org/en/humble/index.html

ROS2チュートリアル（humble）
https://docs.ros.org/en/humble/Tutorials.html

micro-ROSドキュメント
https://micro.ros.org/

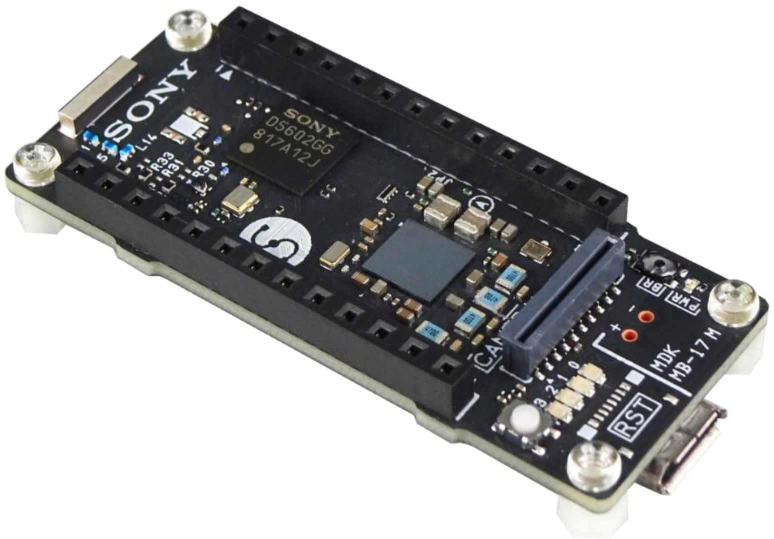
micro-ROSチュートリアル
https://micro.ros.org/docs/tutorials/core/overview/

ROS2 answers
https://answers.ros.org/questions/

SpresenseでROS2/micro-ROSの世界を拡げよう

µROS

micro-ROS puts ROS 2 onto microcontrollers



マルチコアによる豊富な計算能力とフレキシブルな分散設計

低消費電力によるバッテリーセーブ

測位機能と組み合わせた自律制御ロボット

SPRESENSE

SprTurtleBotの制作（オドメトリ）

nav_msgs/msg/Odometry に設定する内容

```
typedef struct nav_msgs__msg__Odometry {
  std_msgs__msg__Header header;
  rosidl_runtime_c__String child_frame_id;
  geometry_msgs__msg__PoseWithCovariance pose;
  geometry_msgs__msg__TwistWithCovariance twist;
} nav_msgs__msg__Odometry;
```

```
typedef struct std_msgs__msg__Header {
  builtin_interfaces__msg__Time stamp;
  rosidl_runtime_c__String frame_id;
} std_msgs__msg__Header;
```

```
typedef struct builtin_interfaces__msg__Time {
  int32_t sec;
  uint32_t nanosec;
} builtin_interfaces__msg__Time;
```

```
typedef struct rosidl_runtime_c__String {
  char * data;
  size_t size;
  size_t capacity;
} rosidl_runtime_c__String;
```

```
typedef struct geometry_msgs__msg__PoseWithCovariance {
  geometry_msgs__msg__Pose pose;
  double covariance[36];
} geometry_msgs__msg__PoseWithCovariance;
```

```
typedef struct geometry_msgs__msg__Pose {
  geometry_msgs__msg__Point position;
  geometry_msgs__msg__Quaternion orientation;
} geometry_msgs__msg__Pose;
```

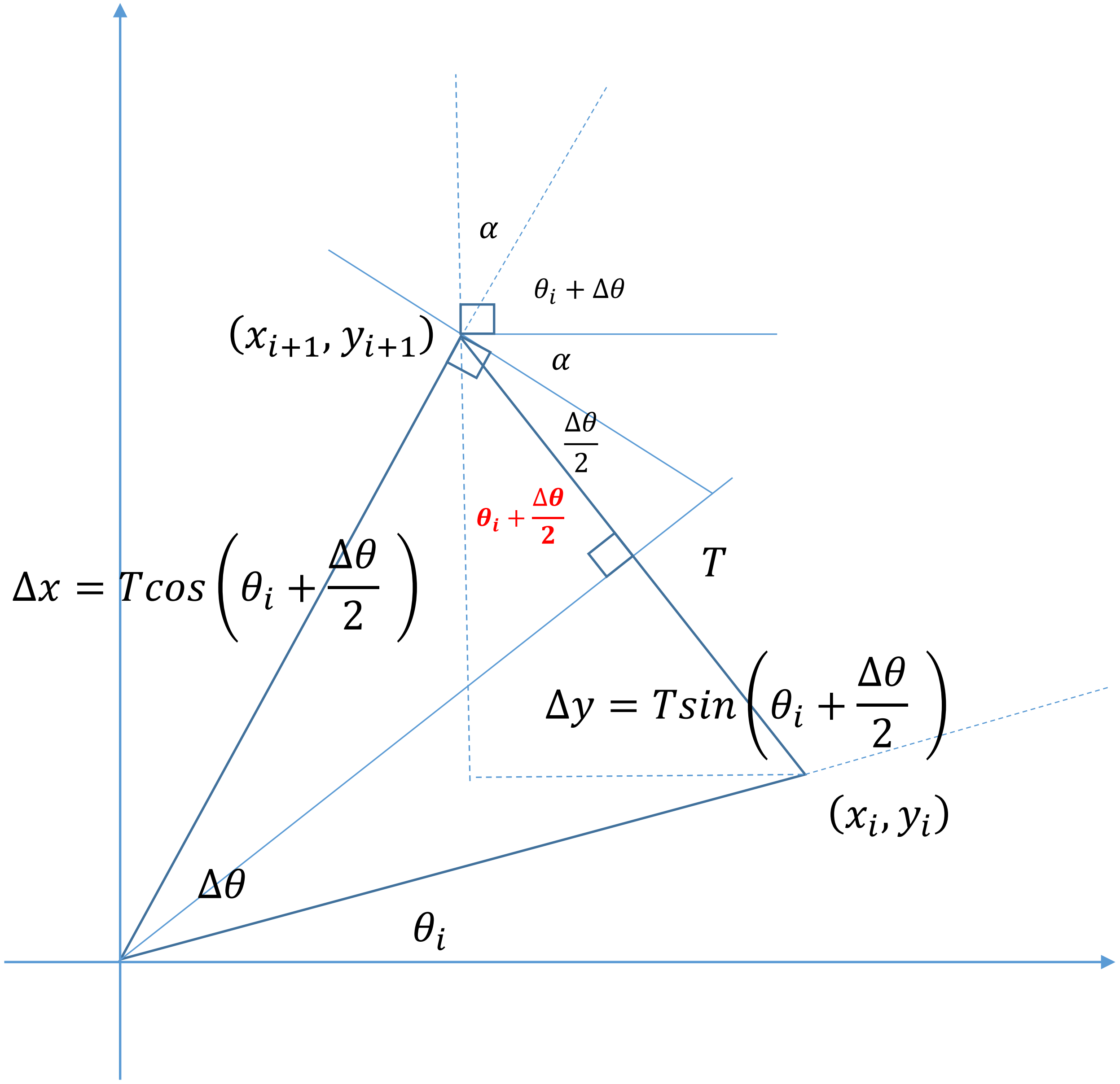
```
typedef struct geometry_msgs__msg__Point {
  double x;
  double y;
  double z;
} geometry_msgs__msg__Point;
```

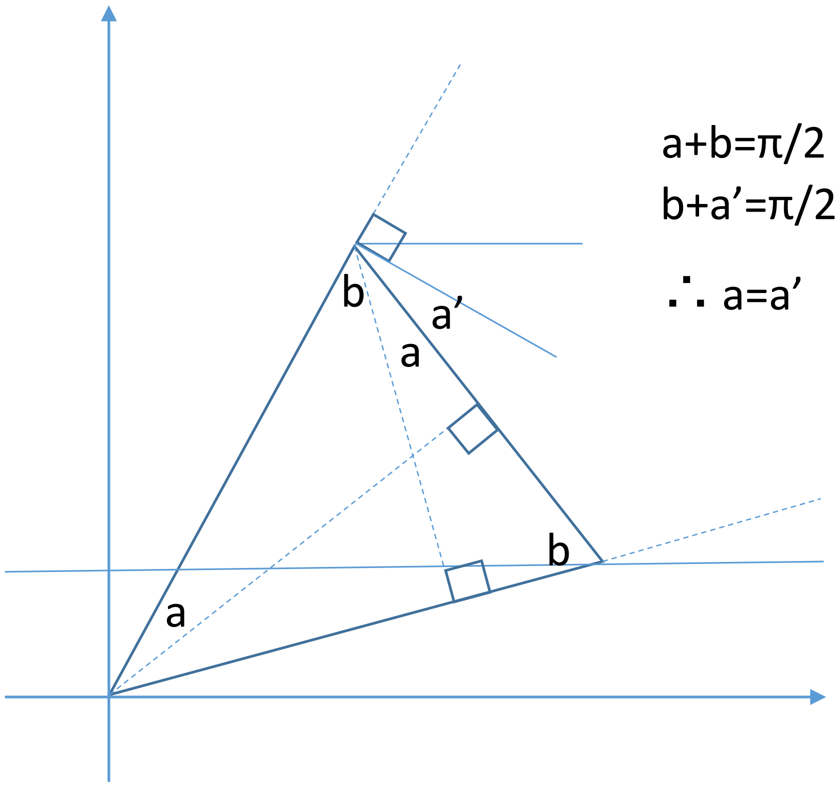
```
typedef struct geometry_msgs__msg__Quaternion {
  double x;
  double y;
  double z;
  double w;
} geometry_msgs__msg__Quaternion;
```

```
typedef struct geometry_msgs__msg__TwistWithCovariance {
  geometry_msgs__msg__Twist twist;
  double covariance[36];
} geometry_msgs__msg__TwistWithCovariance;
```

```
typedef struct geometry_msgs__msg__Twist {
  geometry_msgs__msg__Vector3 linear;
  geometry_msgs__msg__Vector3 angular;
} geometry_msgs__msg__Twist;
```

```
typedef struct geometry_msgs__msg__Vector3 {
  double x;
  double y;
  double z;
} geometry_msgs__msg__Vector3;
```


α
$\theta_i + \Delta\theta$
(x_{i+1}, y_{i+1})
α
$\frac{\Delta\theta}{2}$
$\boldsymbol{\theta_i + \frac{\Delta\theta}{2}}$
T
$\Delta x = T\cos\left(\theta_i + \frac{\Delta\theta}{2}\right)$
$\Delta y = T\sin\left(\theta_i + \frac{\Delta\theta}{2}\right)$
(x_i, y_i)
$\Delta\theta$
θ_i


b
a’
a
b
a

a+b=π/2
b+a’=π/2
∴ a=a’