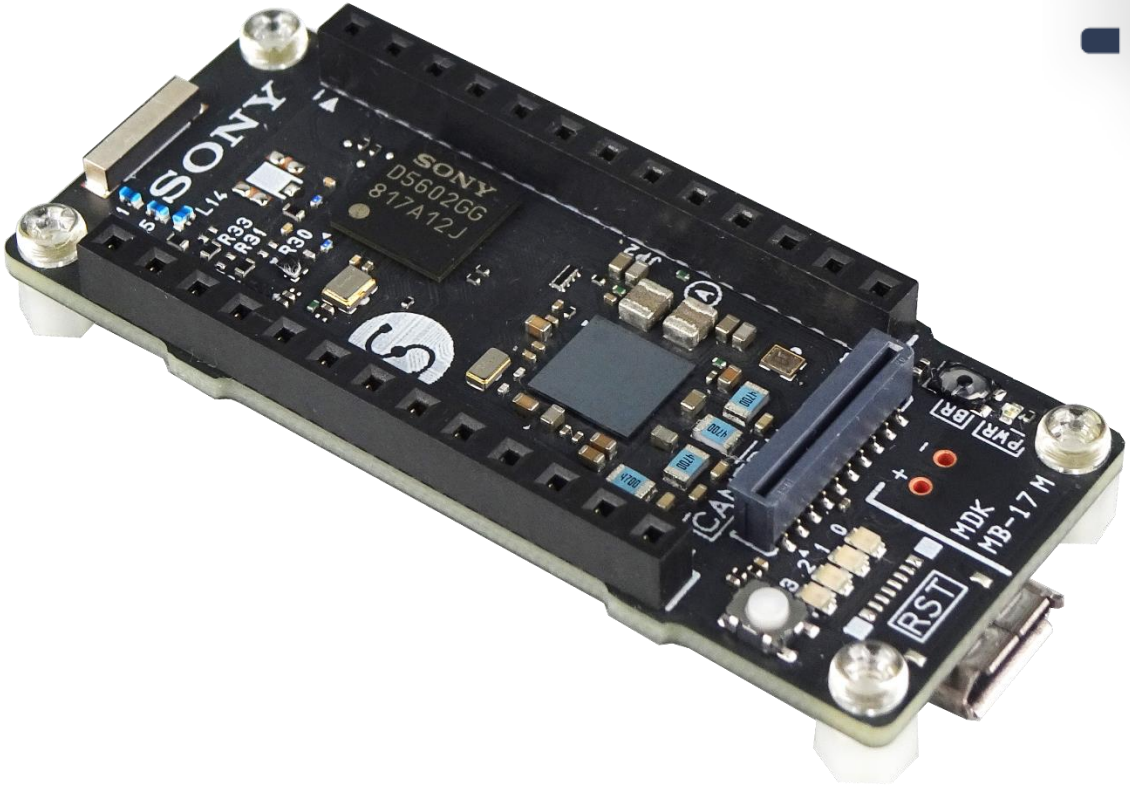
Get Started micro ROS programming with Sony SPRESENSE™

Sony Semiconductor Solutions Corporation
Yoshinori Oota

Contents



μROS

micro-ROS puts ROS 2 onto microcontrollers

micro-ROS とは？

micro-ROS and ROS2

What is ROS?

ROS stands for Robot Operation System, and in a broad sense refers to various tools for robot development. ROS itself is a library released in 2010 that is responsible for a communication system for cooperative operation between a robot and a PC.

What is ROS2?

ROS1 was designed for research purposes and was only intended to communicate with stand-alone robots. As applications expanded, there were more requests for control of multiple robots, support for microcontrollers, and robust communication. ROS2 with these features was released in 2017.

What is micro-ROS?

micro-ROS is a library for RTOS to connect ROS2 and microcontroller systems. ROS2 running on the host PC can cooperate with microcontrollers implementing micro-ROS through a bridge program called micro-ROS-agent.

micro-ROS and ROS2

Difference between ROS and ROS2

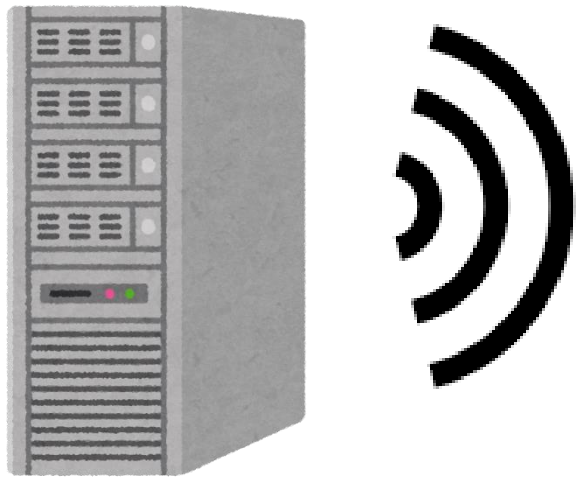
https://design.ros2.org/articles/why_ros2.html

	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 and ROS2

Application using micro-ROS and ROS2

ROS2



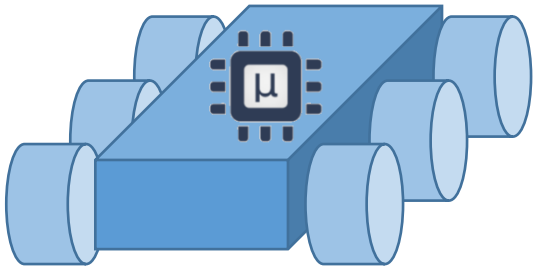
Coordinated control of multiple micro-ROS

micro-ROS and ROS2

micro-ROS and ROS2 Applications

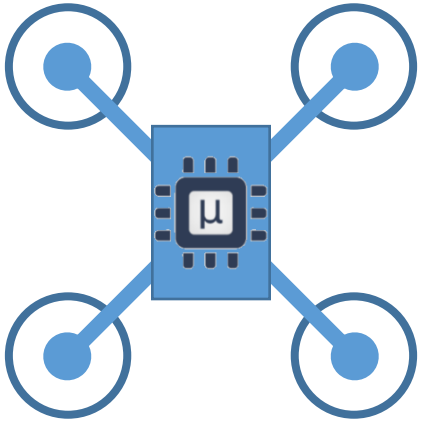
ROS2 is being applied to Automated Guided Vehicle robots and robotic arms in factories, as well as drones and camera systems.

AGVs
（Automated Guided Vehicle）



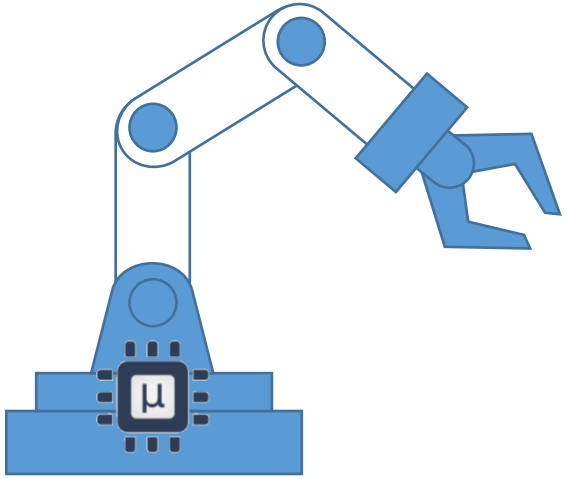
micro-ROS

Drones



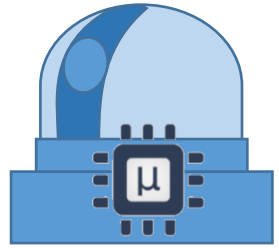
micro-ROS

Robot Arms



micro-ROS

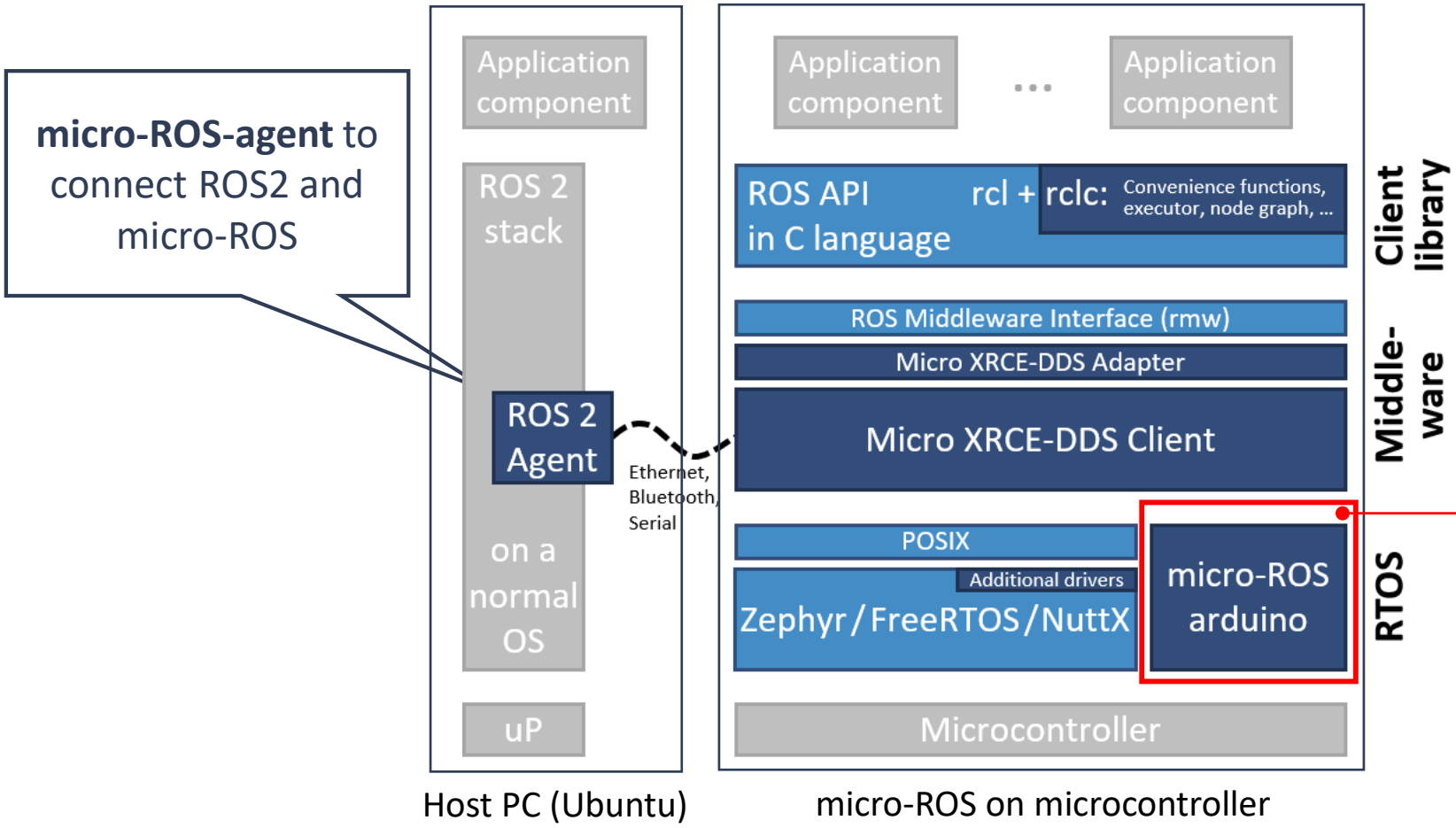
Camera Systems



micro-ROS

micro-ROS and ROS2

Software stack of micro-ROS and ROS2



3 Componens

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

Spresense uses the micro-ROS arduino library

micro-ROS and ROS2 Communication

<ROS2 Elements>

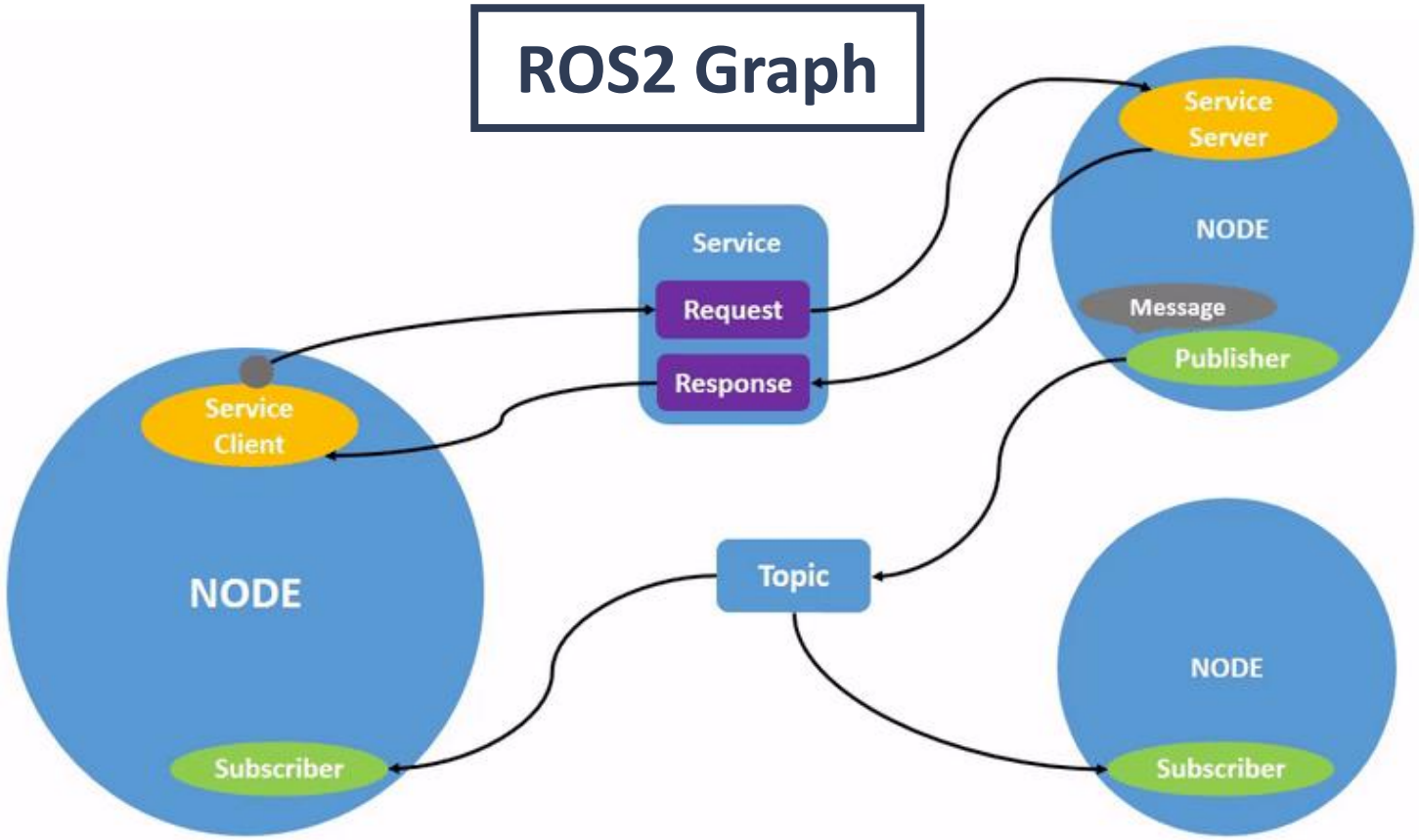
Node

Topic

- Subscriber
- Publisher

Service

- Service Client
- Service Server



ROS2 graph is a network structure of ROS2 elements that process data

micro-ROS and ROS2 Communication

<ROS2 Elements>

Node

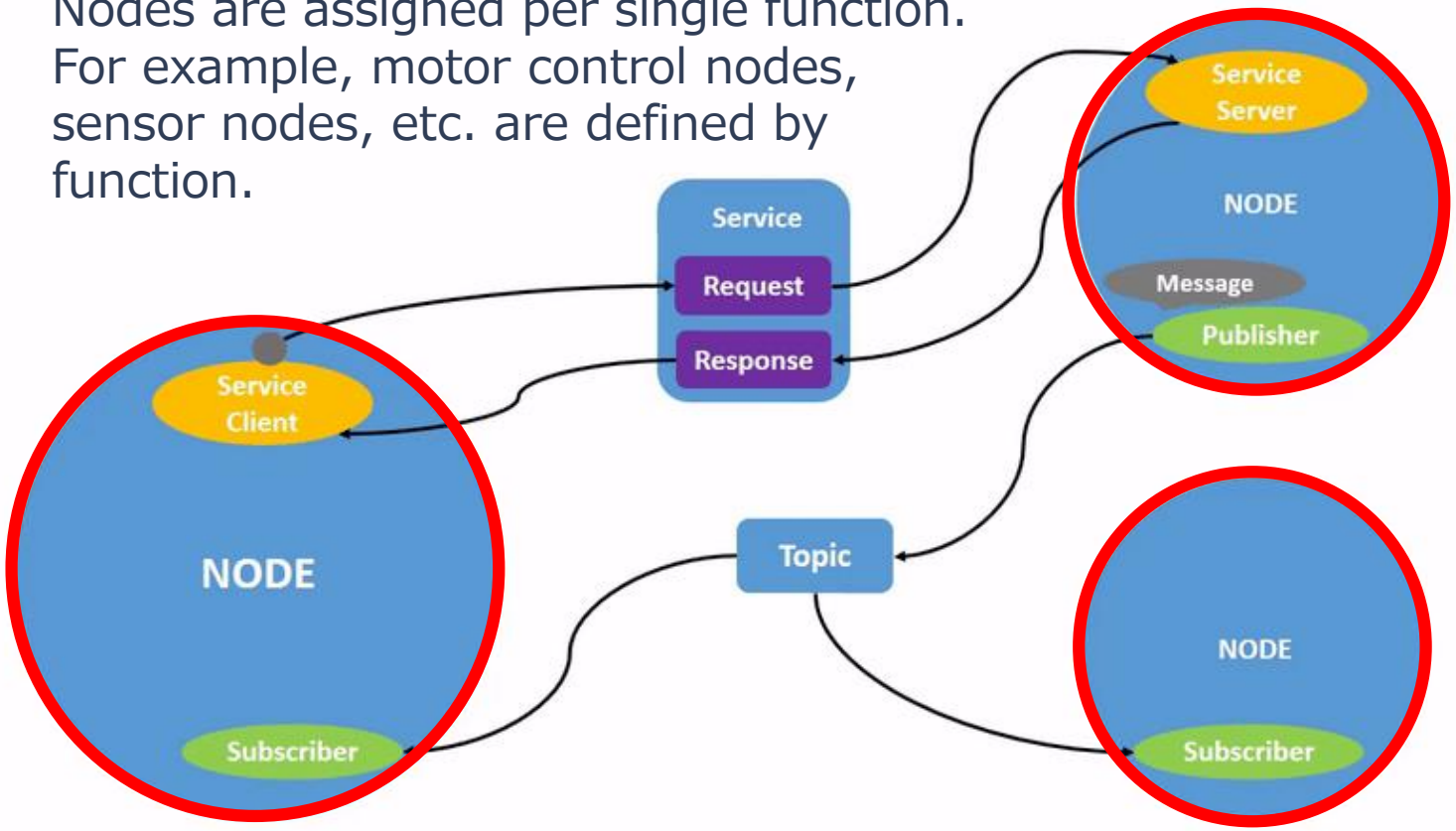
Topic

- Subscriber
- Publisher

Service

- Service Client
- Service Server

Nodes are assigned per single function.
For example, motor control nodes, sensor nodes, etc. are defined by function.



Nodes communicate with each other using Topic and Service.

micro-ROS and ROS2 Communication

<ROS2 Elements>

Node

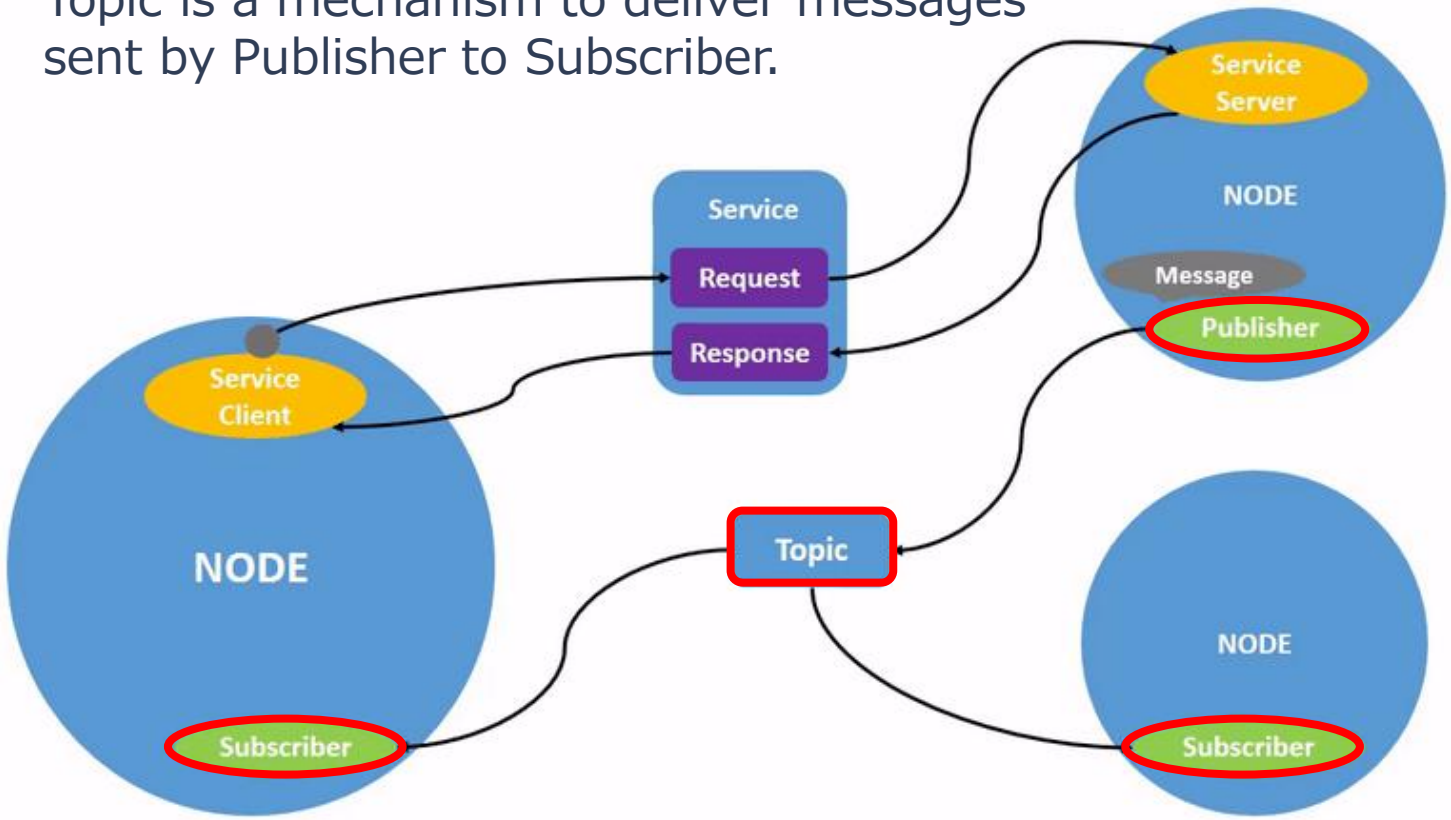
Topic

- Subscriber
- Publisher

Service

- Service Client
- Service Server

Topic is a mechanism to deliver messages sent by Publisher to Subscriber.



Messages are sent to all Subscribers who have subscribed to the Topic.

micro-ROS and ROS2 Communication

<ROS2 Elements>

Node

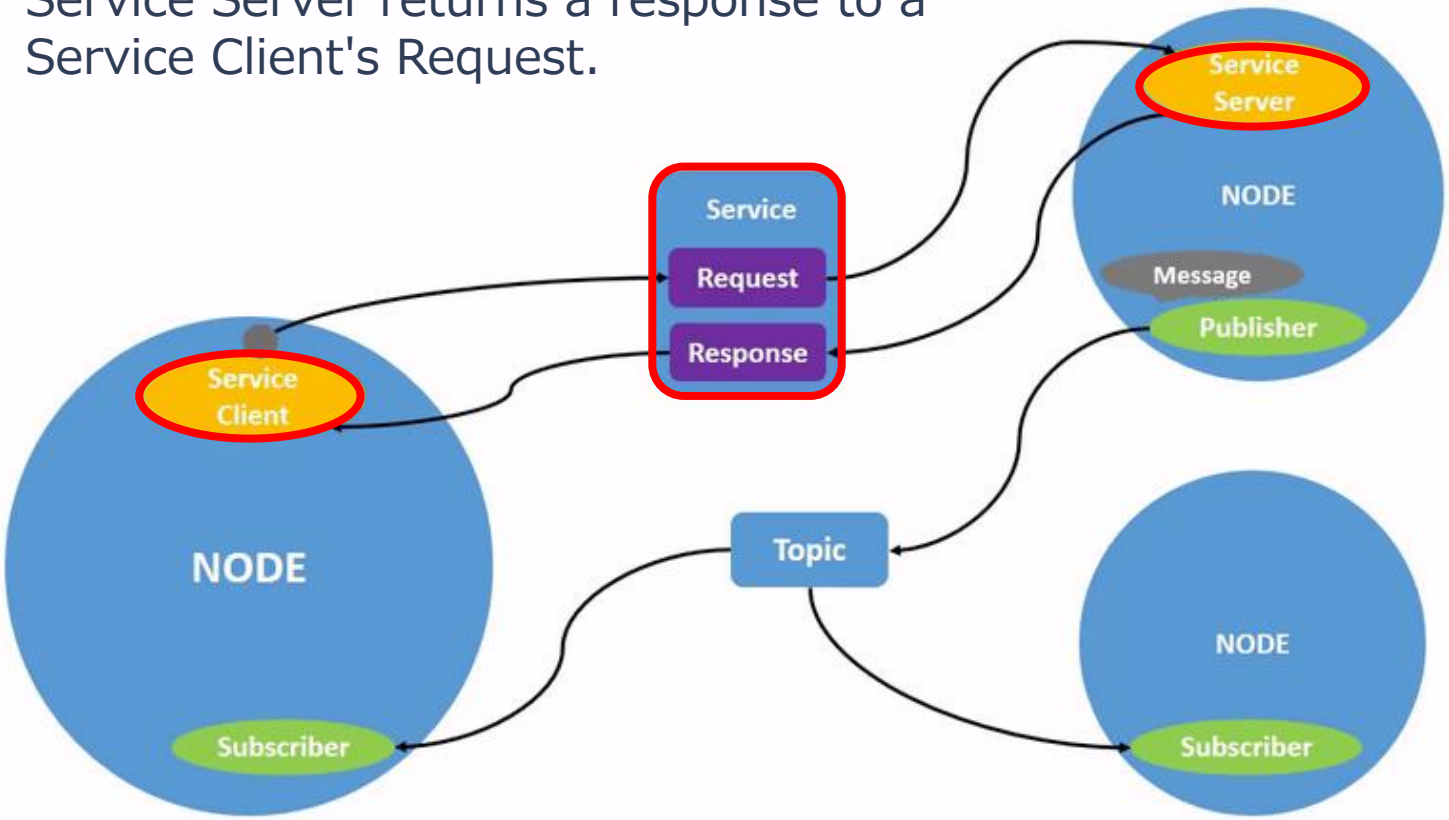
Topic

- Subscriber
- Publisher

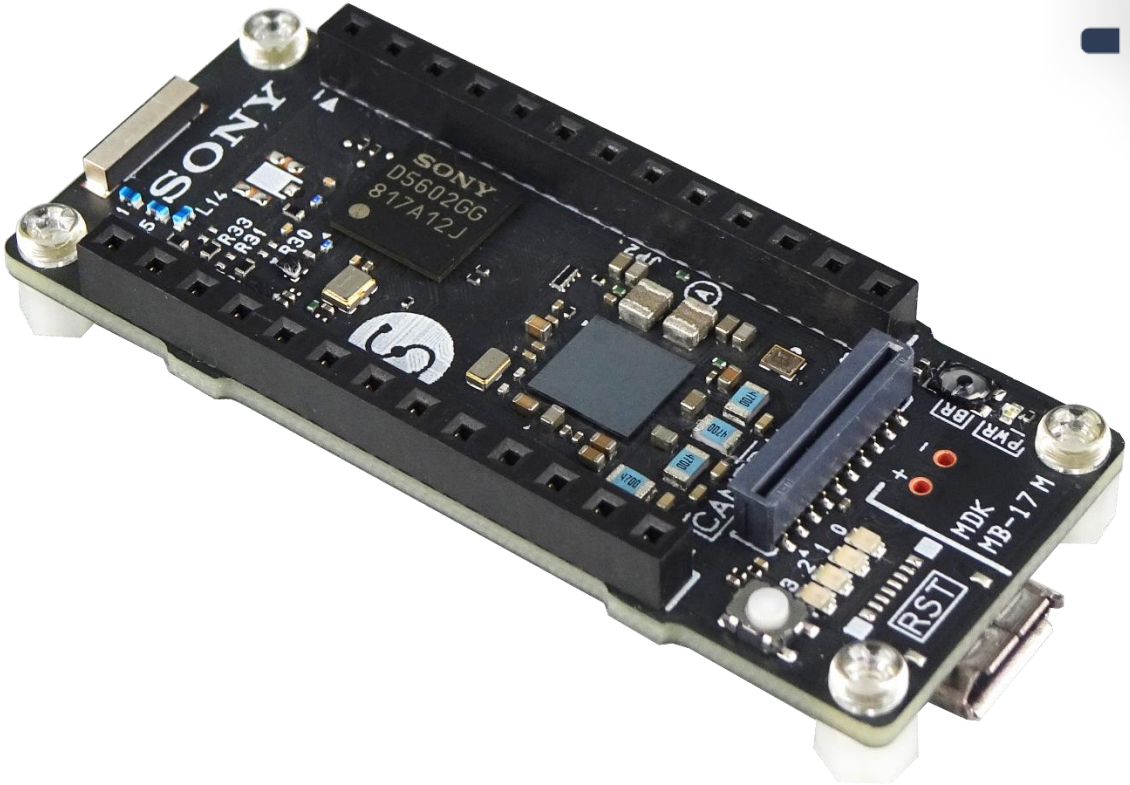
Service

- Service Client
- Service Server

Service Server returns a response to a Service Client's Request.



Unlike Topic, the difference is that it is a one-to-one communication between Service Client / Server.



micro-ROS

micro-ROS puts ROS 2 onto microcontrollers

micro-ROS on Spresense

Development Environment

ROS2

ROS2 uses "humble". humble is for Ubuntu 22.04 LTS. The Host PC must have Ubuntu 22.04 LTS installed.

micro-ROS-agent

micro-ROS-agent can be compiled and generated, but for ease of use, we will use a Docker image. Docker must be installed during setup.

Spresense / micro-ROS-arduino

We have prepared micro-ROS-arduino for Spresense on GitHub. When downloading, make sure the branch you are trying to download is “humble-spresense”.

Development Environment

ROS2 Setup

Please refer to the following site for installation on a PC with Ubuntu 22.04 TLS installed
https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html

【STEP1】Added ROS packages to PPA (Personal Package Archive)

```
$ 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
```

Development Environment

ROS2 Setup

【STEP2】Install ROS2 package

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

【STEP3】Register ROS2 environment variables

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

Development Environment

ROS2 Setup

【STEP4】Open a terminal and type commands below to check the response

```
$    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】Open another terminal and type commands to check the communication work properly

```
$    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]
```

Development Environment

micro-ROS-agent Setup

【STEP1】Install docker

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

Development Environment

micro-ROS-agent Setup

【STEP2】Connect Spresense, install and launch the micro-ROS-agent docker image.
The docker image starts to download automatically when the docker is installed properly.

```
$   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
```

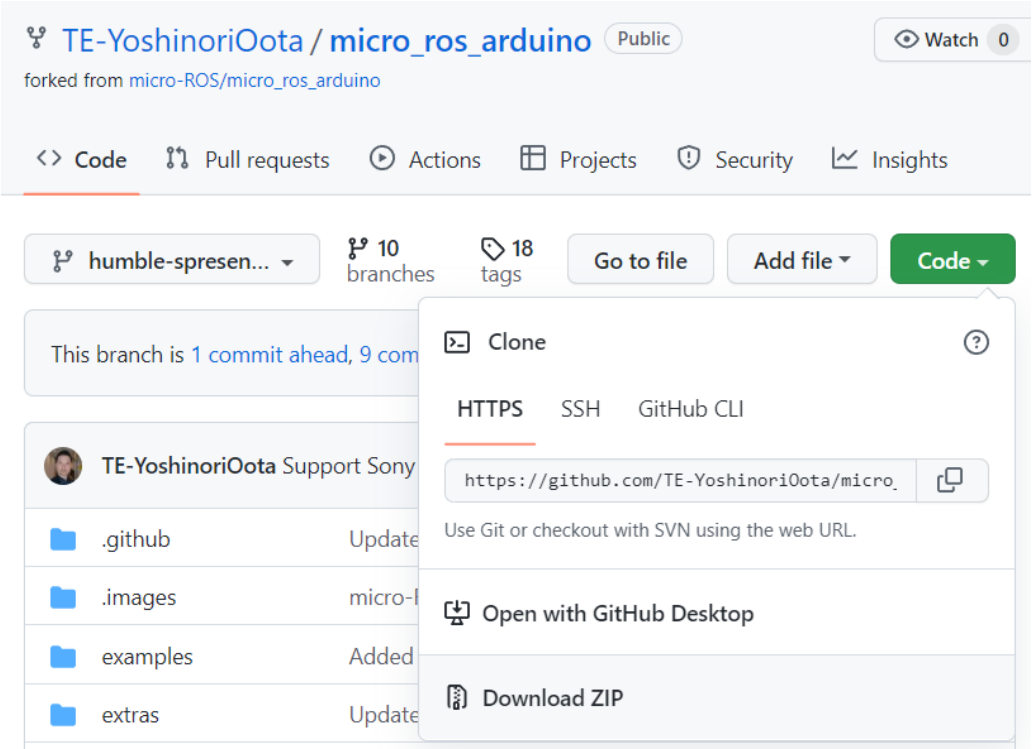
Download the docker image of micro-ROS-agent

Spresense Development Environment

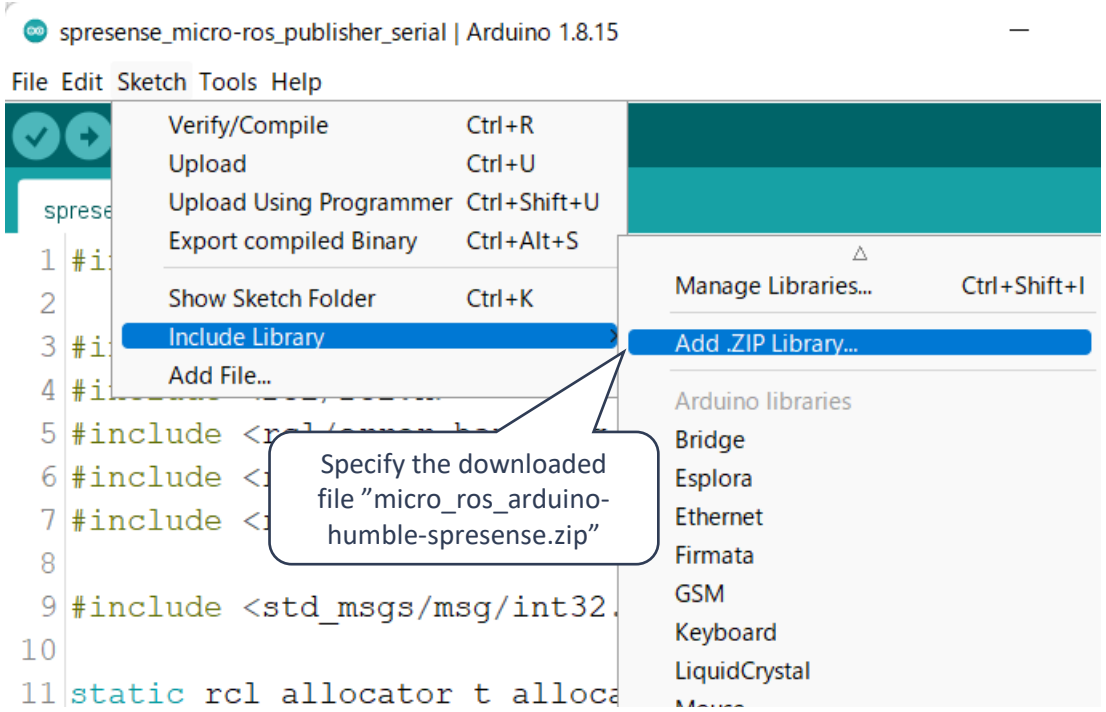
Setup Spresense / micro-ROS-arduino

If it does not work, unzip "micro_ros_arduino-humble-spresense.zip" and copy it to the "libraries folder of your Arduino sketch folder.

【STEP1】Download micro_ROS_arduino



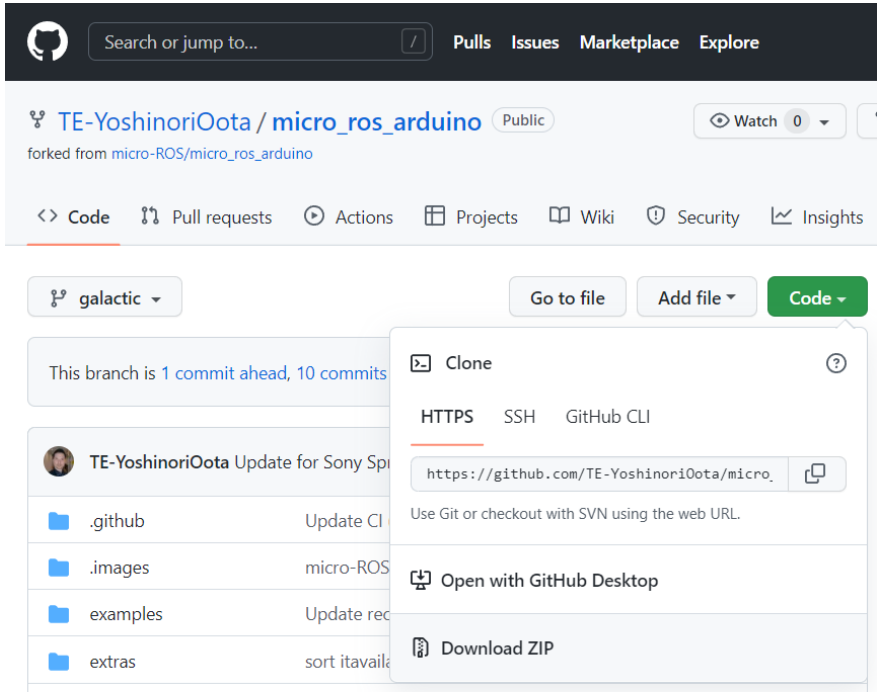
【STEP2】Install micro_ROS_arduino



Spresense Development Environment

Setup Spresense / micro-ROS-arduino

【STEP3】Download the materials of “Spresense_microROS_seminar”



【STEP4】Flush "spresense_micro-ROS_ publisher_serial" in the sketches folder

Spresense Development Environment

Setup Spresense / micro-ROS-Arduino (Serial Communication)

Start micro-ROS-agent on Ubuntu with Spresense connected and reset Spresense

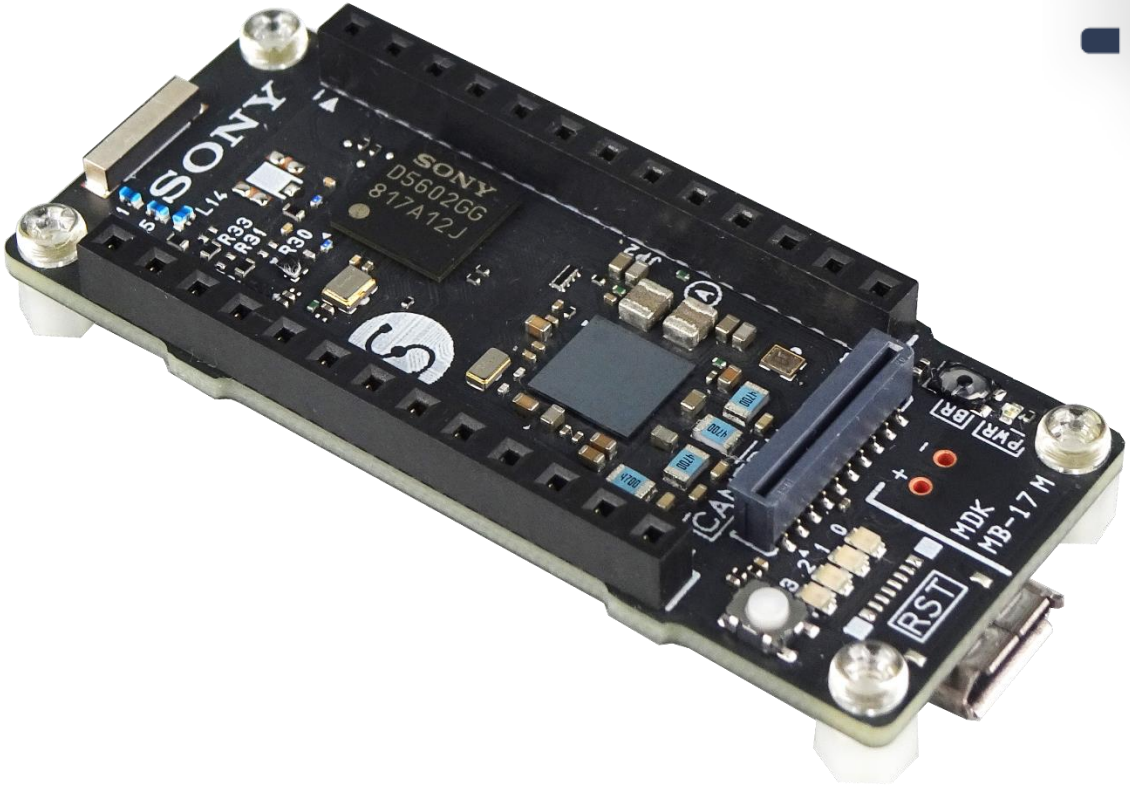
```
$    source /opt/ros/humble/setup.bash
$    sudo docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-ros-agent:humble serial --dev /dev/ttyUSB0
[1659348844.975038] info     | TermiosAgentLinux.cpp | init                 | running...           | fd: 3
[1659348844.975462] info     | 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 Development Environment

Setup Spresense / micro-ROS-arduino

Open another terminal on Ubuntu to check the topic sent out by Spresense

```
$   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 implementation

micro-ROS (Topic Publisher/Subscriber)

<ROS2 Elements>

Node

Topic

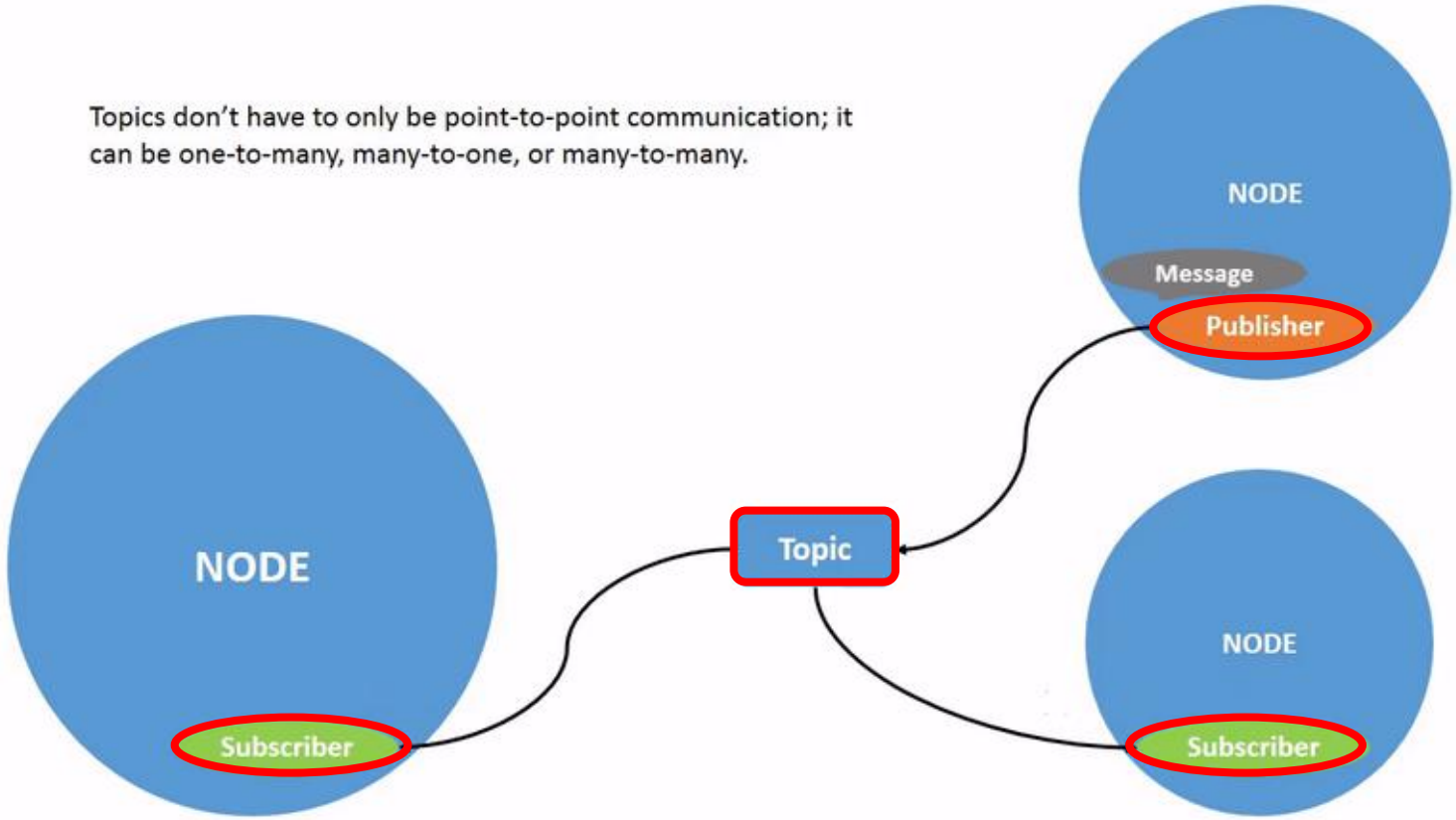
Subscriber

Publisher

Service

Service Client

Service Server



micro-ROS (Topic Publisher)

<ROS2 Elements>

Node

Topic

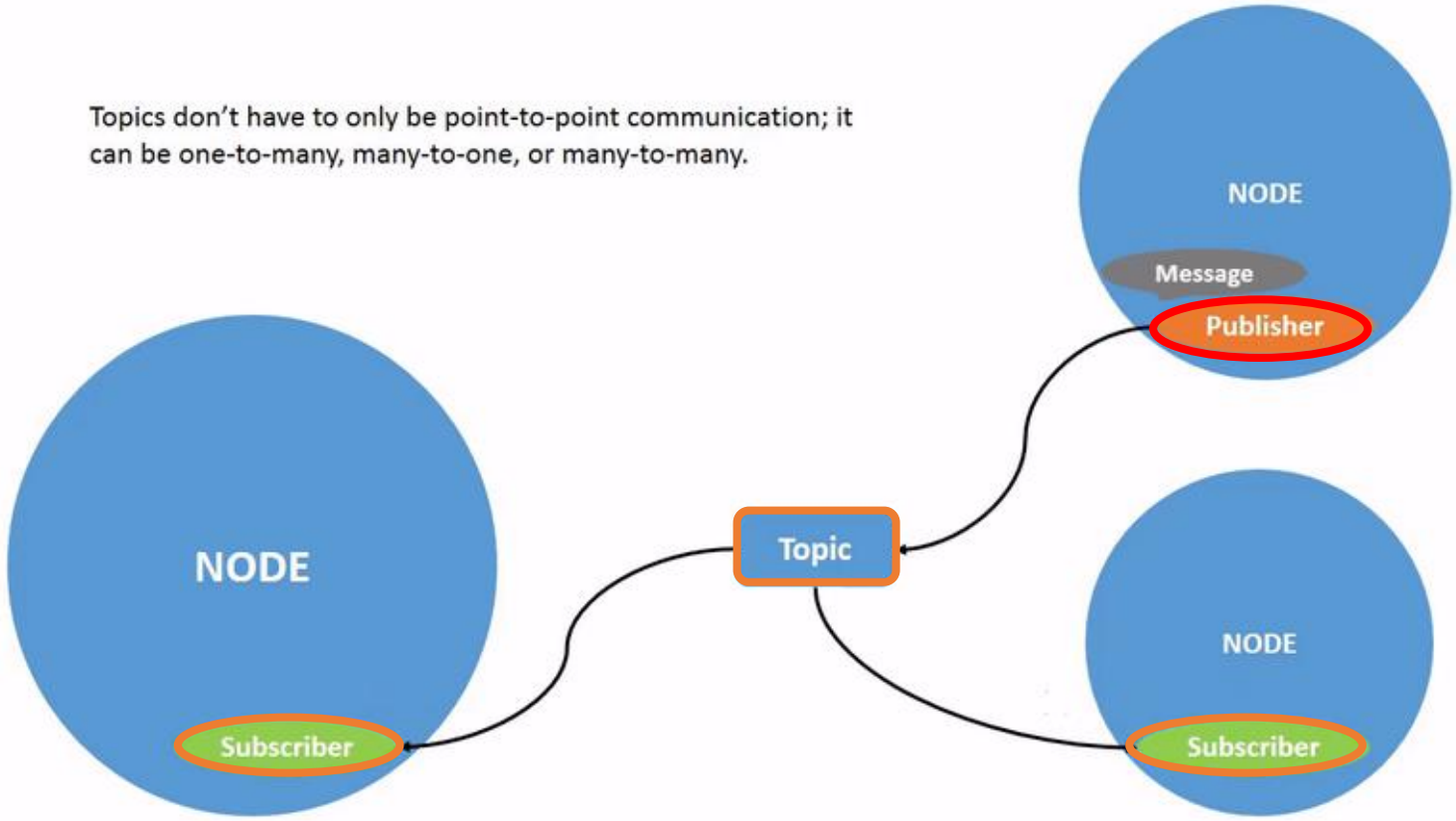
Subscriber

Publisher

Service

Service Client

Service Server



micro-ROS (Topic Publisher)

Implementation for serial communication

```
#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 function>
                                   LED0 blinks when an error occurs
```

```
void setup() {

  set_microros_transports();                       Initialize serial communication
  delay(2000);

  allocator = rcl_get_default_allocator();         Various settings including
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));   memory manager

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

  // create publisher                              Create the 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);                                      Publish msgs every 100 milliseconds
  RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
  msg.data++;
}
```

micro-ROS (Topic Publisher)

Serial Implementation

```
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++;
}
```

Wi-Fi Implementation

```
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++;
}
```

Replace set_microros_transports() with rmw_uros_set_custom_transport()

The functions of “arduino_wifi_transport_xxx” need to be implemented by your self.

micro-ROS (Topic Publisher)

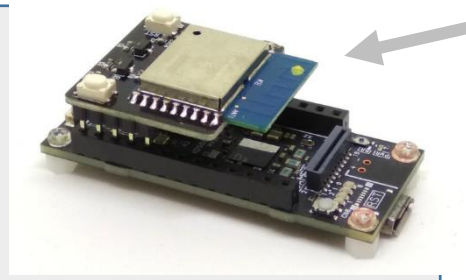
Implementation example of “arduino_wifi_transport_xxx” using GS2200 Wi-Fi Add-on board

```
#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 (Topic Publisher)

Implementation example of “arduino_wifi_transport_xxx” using [ESP8266 Wi-Fi Add-on Board]()



```
#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”
```

Using Library: https://github.com/YoshinoTaro/ESP8266ATLib-for-Spresense

micro-ROS (Topic Publisher)

Operation Check

Start micro-ROS-agent on Ubuntu with Spresense connected and reset the Spresense

```
$ 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)
```

For serial communication, replase as follows:
serial --dev /dev/ttyUSB0

micro-ROS (Topic Publisher)

Operation Check

Open another terminal on Ubuntu to check the topic sent out by Spresense

```
$    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 (Topic Subscriber)

<ROS2 Elements>

Node

Topic

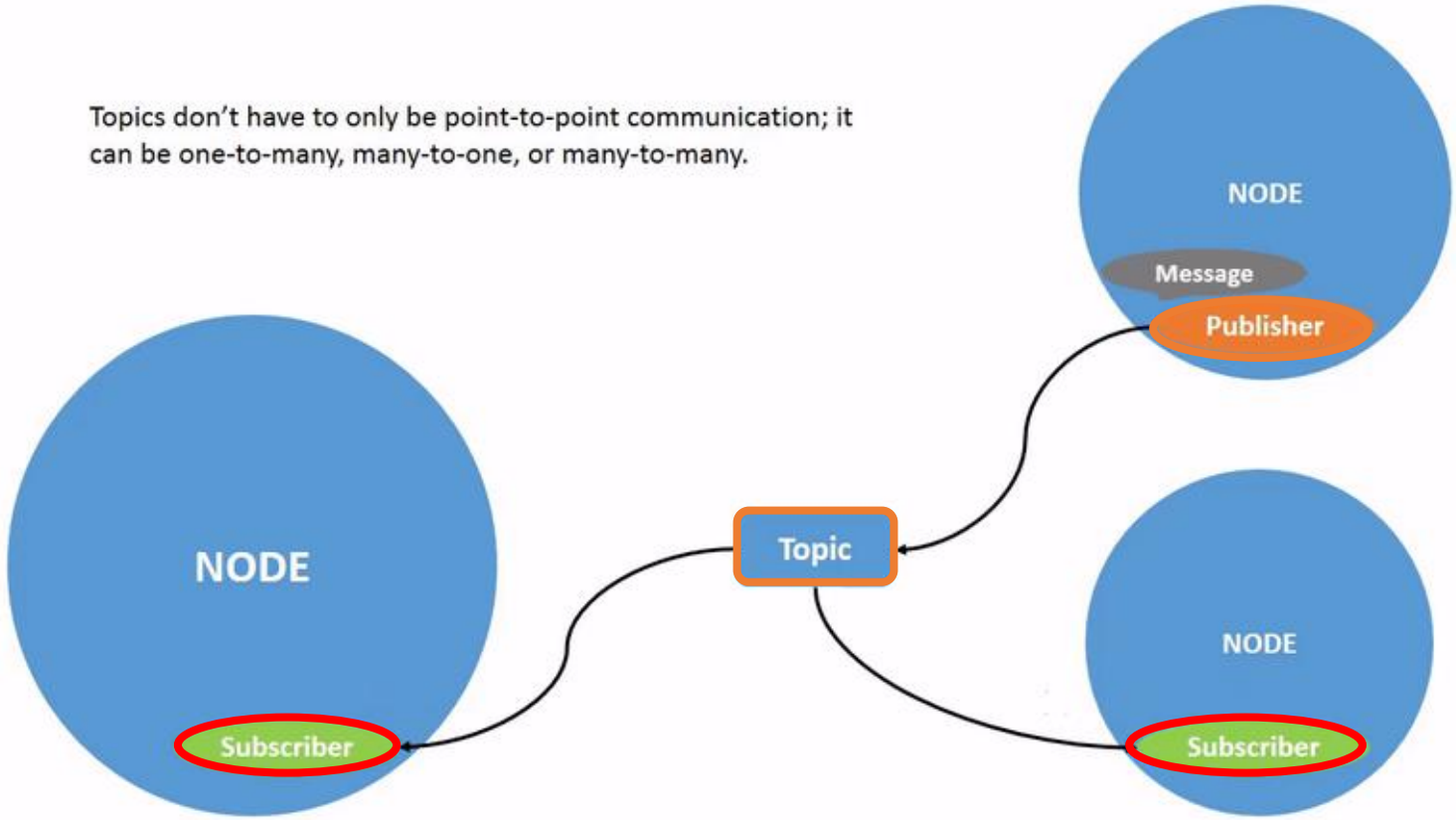
Subscriber

Publisher

Service

Service Client

Service Server



micro-ROS (Topic Subscriber)

Subscription Implementation Example

```
#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>
                                                     Callback function called when new data arrives
```

```
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,                    Create subscribe
    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,                 Register subscription task
                &msg,  &subscription_callback, ON_NEW_DATA));                    in Executor

  digitalWrite(LED0, HIGH);
}

void loop() {
  delay(100);                                                                    Check for TOPIC
  rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));                         every 100 milliseconds
}
```

micro-ROS (Topic Subscriber)

Operation Check

Start micro-ROS-agent on Ubuntu with Spresense connected and reset the Spresense

```
$    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)
```

For WIFi, set as follows: **udp4 -p 8888**

micro-ROS (Topic Subscriber)

Operation Check

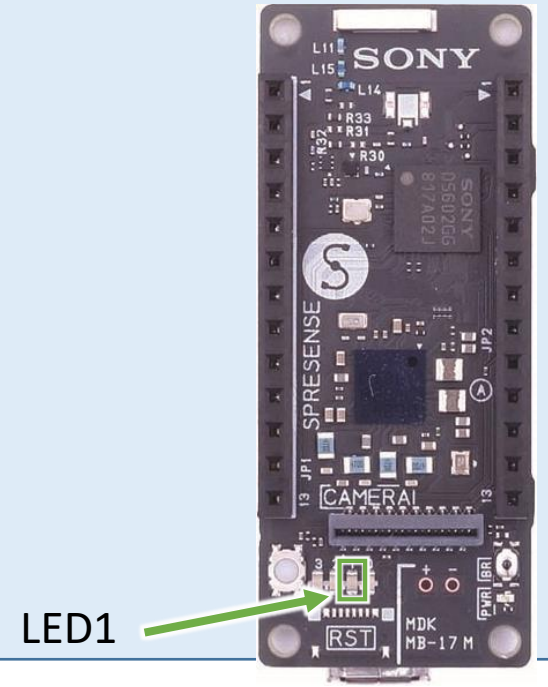
Open another terminal on Ubuntu and turns LED1 of Spresense on and off

```
$    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 ON

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

LED1 OFF



micro-ROS (Image Publisher)

Advanced

<ROS2 Elements>

Node

Topic

Subscriber

Publisher

Service

Service Client

Service Server

Spresense Camera

Acquire images by Spresense Camera and publish the images as Topics

micro-ROS (Image Publisher)

Advanced

Image Publisher Implementation-1

```
#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"));

  static uint8_t img_buff[20000] = {0};                    Initialize the compressed_image buffer
  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;

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

  digitalWrite(LED0, HIGH);
}
```

micro-ROS (Image Publisher)

Advanced

Image Publisher Implementation-2

```
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);                                   Take a photo every second and Publish the image as a topic
}
```

micro-ROS (Image Publisher)

Advanced

Operation Check

Open another terminal on Ubuntu and launch rqt_image_view

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

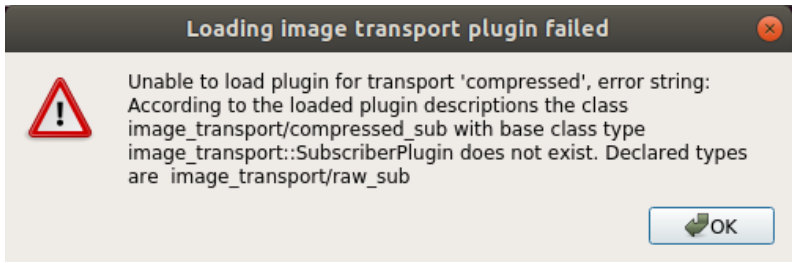
micro-ROS (Image Publisher)

Advanced

Operation Check

After a while, you can see the image as shown in the image on the right. ➡

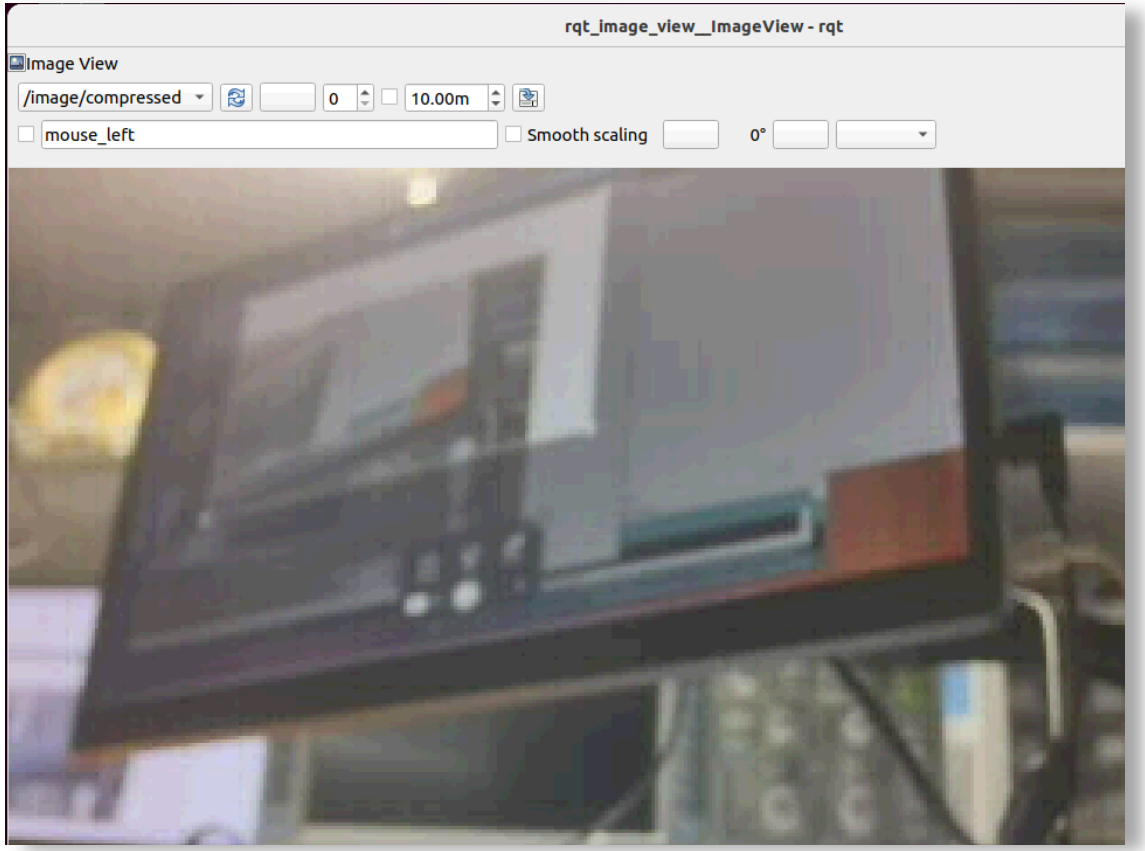
If you change the environment, such as updating Distro in ROS, you may get the following error



In that case, reinstall the plug-in with the following command

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

The screen shot of rqt_image_view



micro-ROS (Service Server/Client)

<ROS2 Elements>

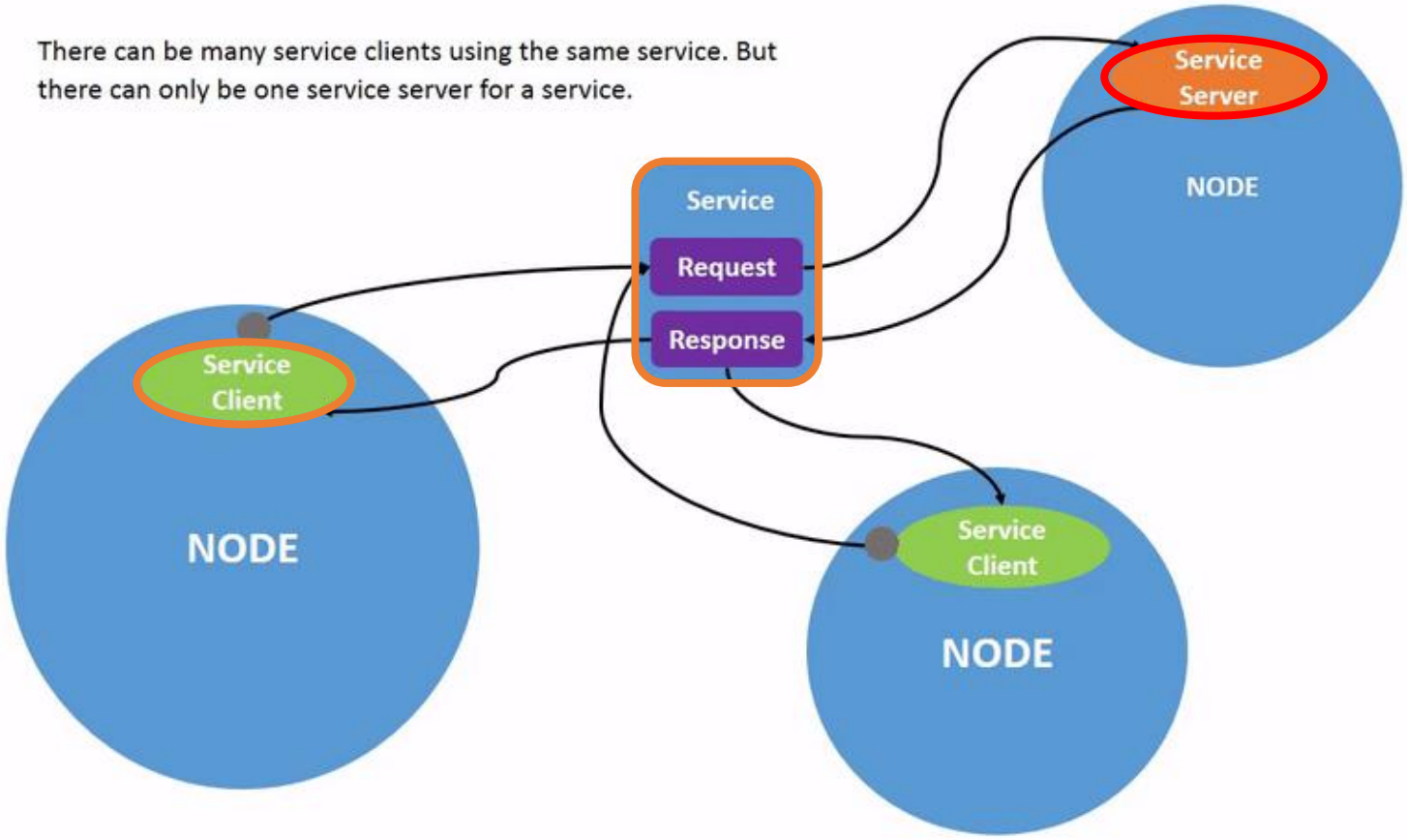
Node

Topic

- Subscriber
- Publisher

Service

- Service Client
- Service Server



⚠ You will need to rebuild the arduino micro-ROS library to incorporate your own service definitions. “galactic” has a problem communicating with ROS2, and Humble has some problems (description later). micro-ROS-agent implementation is not stable at this moment(Jul. 22), so be careful if you use it.

micro-ROS (Service Server)

<ROS2 Elements>

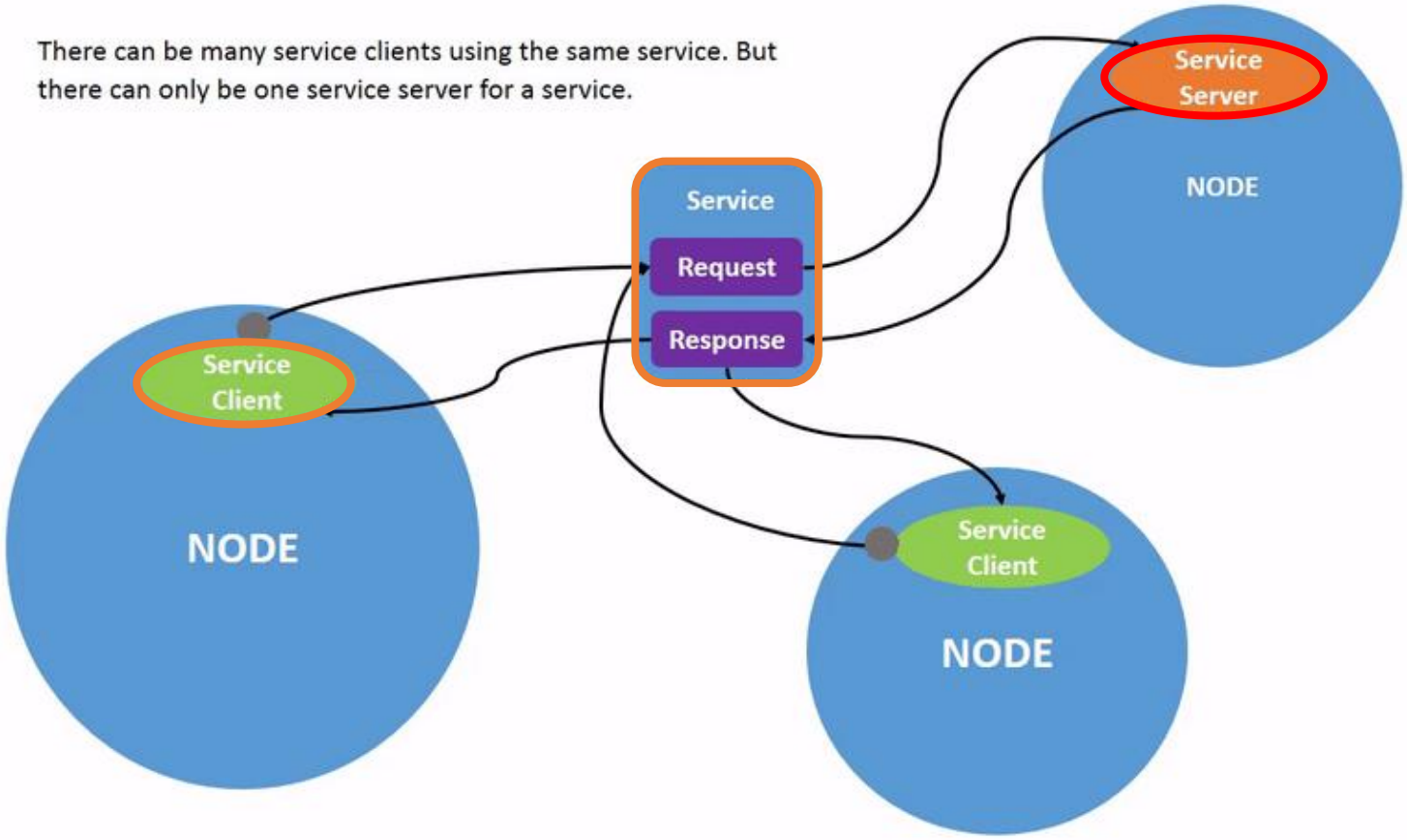
Node

Topic

- Subscriber
- Publisher

Service

- Service Client
- Service Server



micro-ROS (Service Server)

Service Server Implementation

```
#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};                        Parameters for the callback function below

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);
}                                                    Callback function called when accessed by client

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,          Register the service in Executor
    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);                                                          Check requests
  rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));               every 100 milliseconds
}
```

micro-ROS (Service Server)

Operation Check

Start micro-ROS-agent on Ubuntu with Spresense connected and reset the Spresense

For WIFi, set as follows: **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 (Service Server)

Operation Check

Access the Service Server with the following command

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

The Topic name will be the name set in the program
Example: 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 ON

```
$    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 OFF

micro-ROS (Service Client)

<ROS2 Elements>

Node

Topic

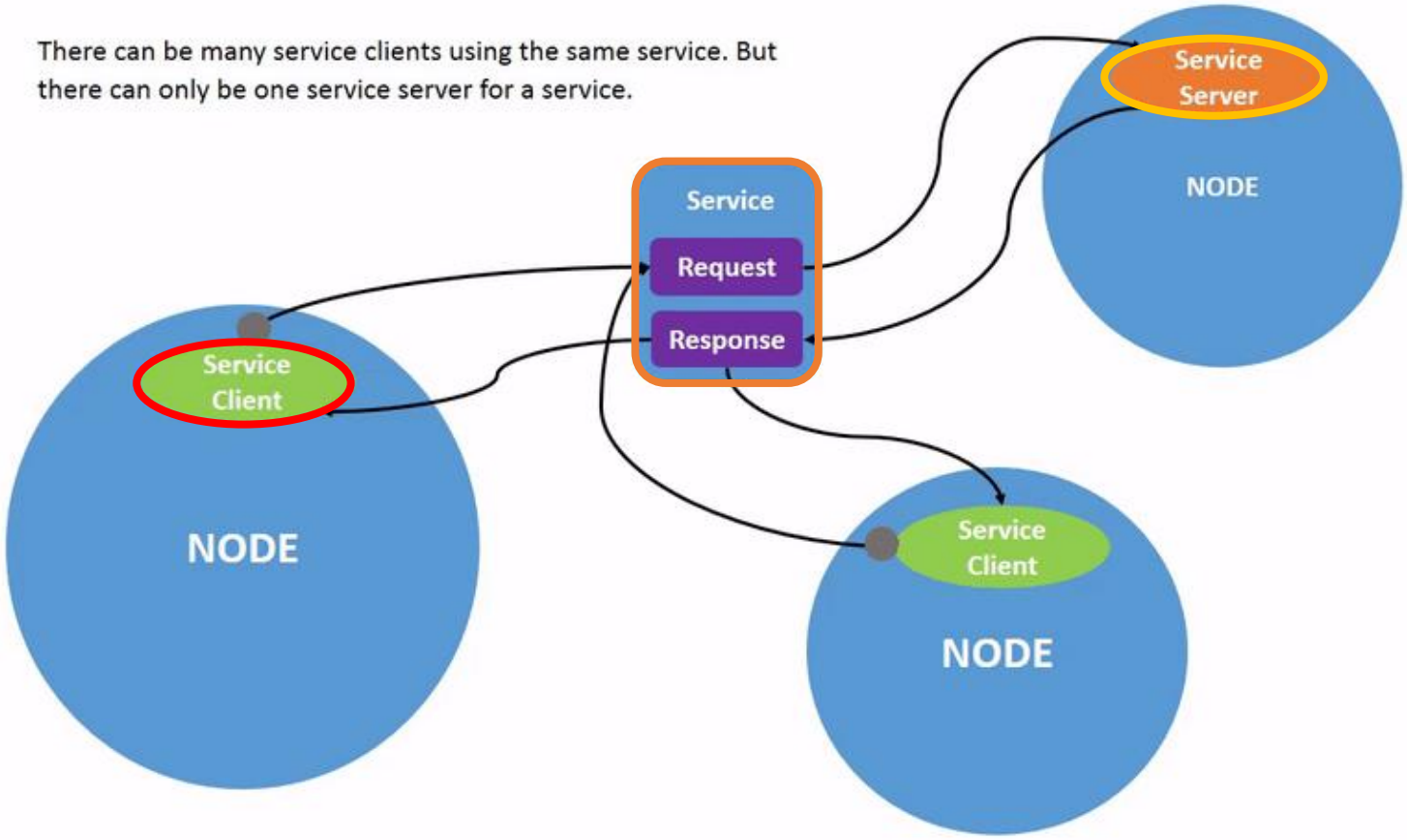
Subscriber

Publisher

Service

Service Client

Service Server



micro-ROS (Service Client)

Service Client Implementation

```
#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);
}
```

Callback function called when a response is received from the server

```
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));
}
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));
}
```

Register the client in Executor

Send client request every 100 milliseconds

micro-ROS (Service Server)

Implementation of a ROS2 Trigger service server for the test

Create a work directory for ROS2 project and generate the package for the service server

```
$ 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 (ROS2 Server python based)

Implementation of trigger_service.py

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()
```

When sending a string from ROS2 (humble/July ‘22) to micro-ROS by Service Response, (null) is set at the beginning of the string. Please put a space or other unnecessary character at the beginning of the string.

Implementation of setup.py

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
        ],
    },
)
```

If 'service' is specified in the command line argument, “main” of ”trigger_service” is called

micro-ROS (Service Client)

Implementation of a ROS2 Trigger service server for the test

Build Trigger Service for ROS2 and register it to run commands

```
$    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 (Service Client)

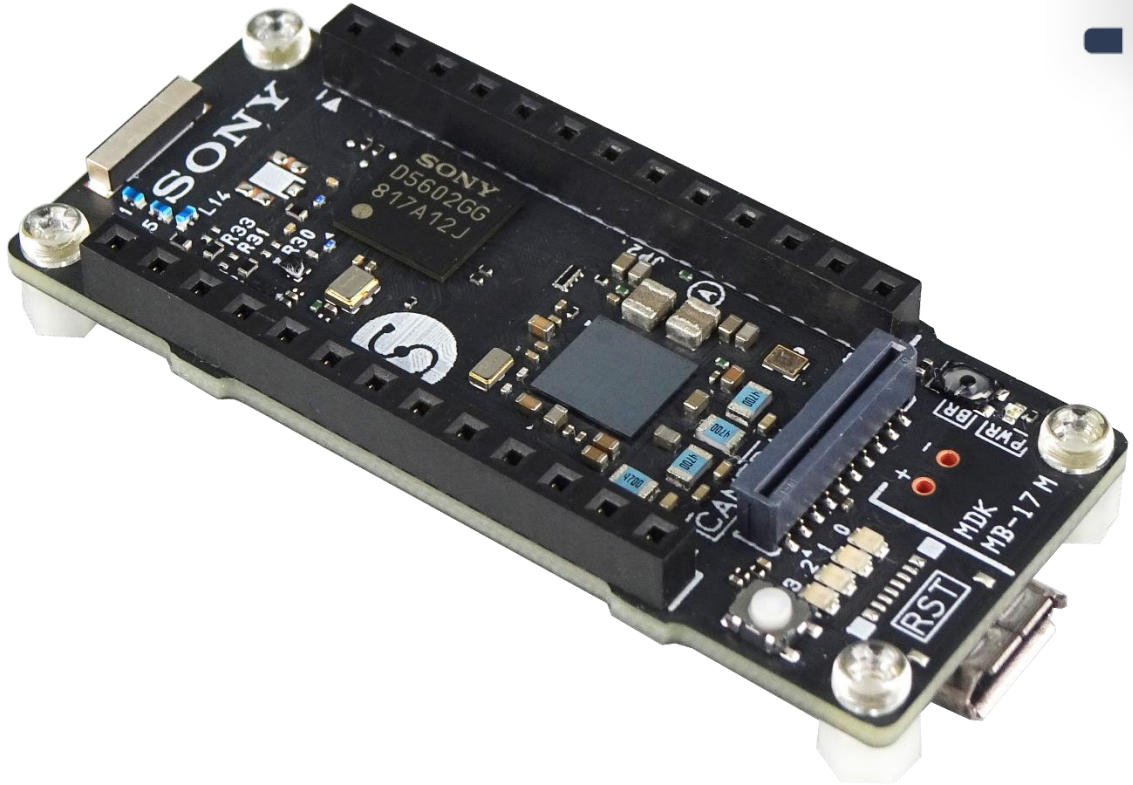
Operation Check

Open Serial Console on Arduino IDE

```
Send Trigger to server.                    You'd better check it with ESP8266
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
```

Launch Trigger Service of ROS2 on Ubuntu

```
$    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]
```



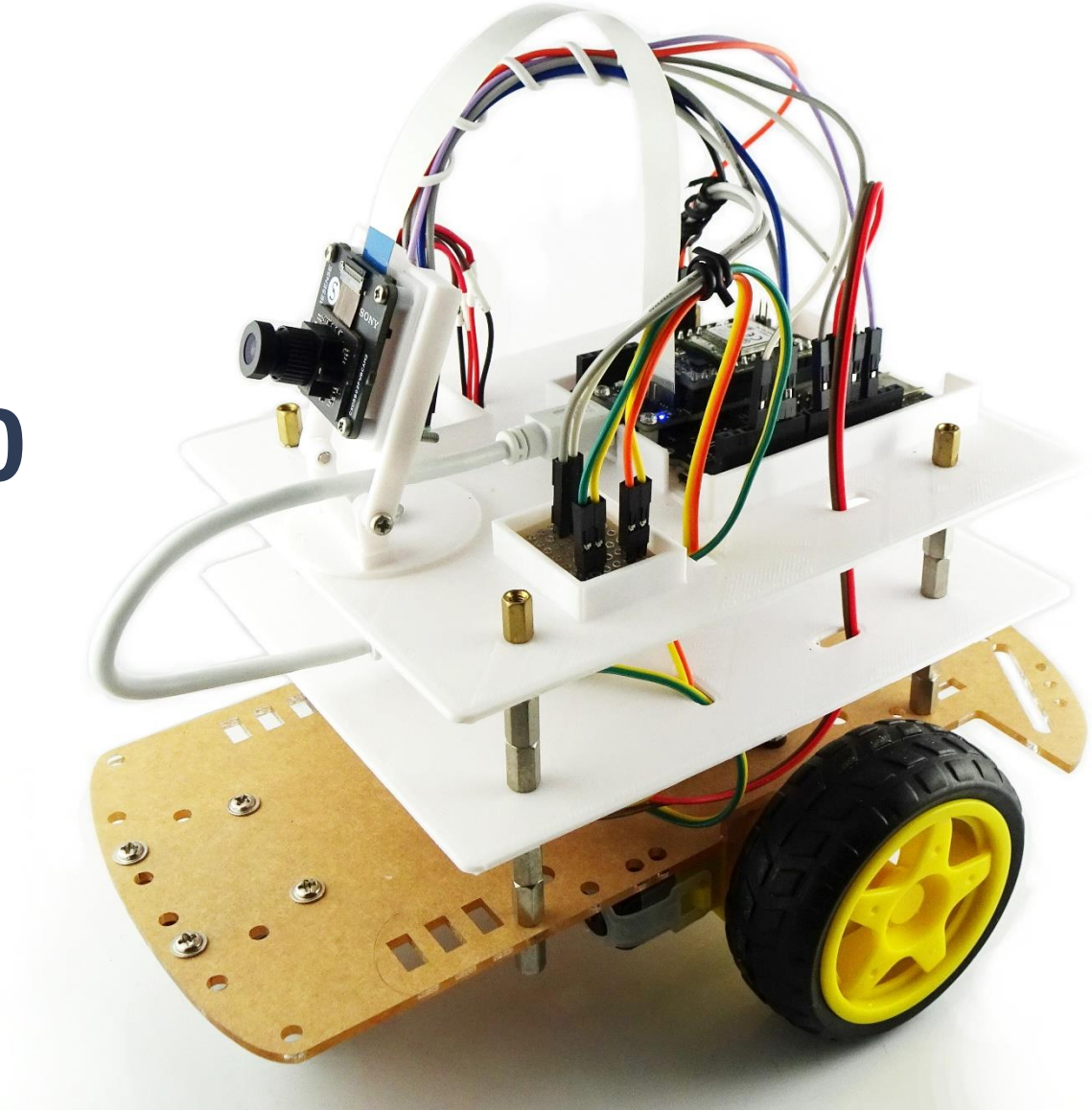
micro-ROS

micro-ROS puts ROS 2 onto microcontrollers

Implementation of Simple TurtleBot using Spresense

Make SprTurtleBot

SprTurtleBot0



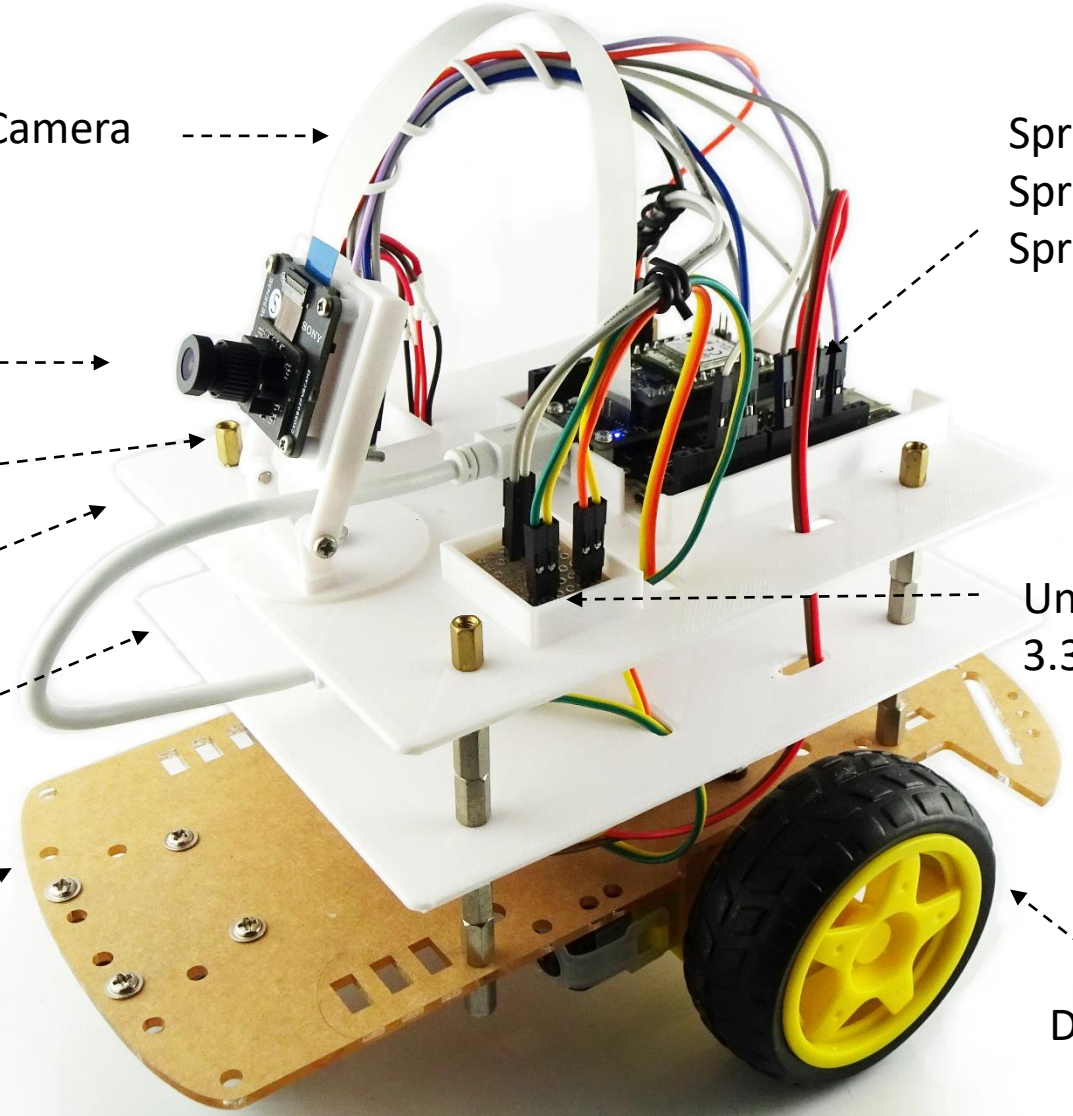
Make SprTurtleBot

Components of SprTurtleBot0

Component	Number of items	The role	Memo
Spresense Main Board	1	Controller	SPRESENSE Main BoardCXD5602PWBMAIN1]
Spresense Extension Board	1	I/O Board	SPRESENSE Extension Board [CXD5602PWBEXT1]
Spresense Camera Board	1	Camera	SPRESENSE Camera Board [CXD5602PWBCAM2W]
Spresense WiFi Add-on Board	1	Communication to ROS2	SPRESENSE Wi-Fi Add-on iS110B
FPC for Spresense Camera	1	FPC to connect Camera and Main Board	Molex 15166-0123
Chassis, Motors and Tires	1	Basic Chassis	Robot Car Chassis Kit with Motors
Encoder	2	To count tire rotations	Speed Measuring Module with Photoelectric Encoders
Motor Driver	1	Motor Drive Circuit	DRV8835 Dual Motor Driver
Universal PCB board	1	To make a circuit to distribute 3.3V power line	Mini Solderable BreadBoard
Chassis to mount Spresense	1	3D printed chassis	
Chassis to mount a mobile battery	1	3D printed chassis	
Others (wires, spacers)	Many	Wires to connect circuits and Spacers to build up chassis	

Make SprTurtleBot

Configuration of SprTurtleBot0



FPC for Spresense Camera

Spresense Camera Board

Motor Driver

Chassis for Spresense Board

Chassis for Battery Board

Base Chassis + Encoders

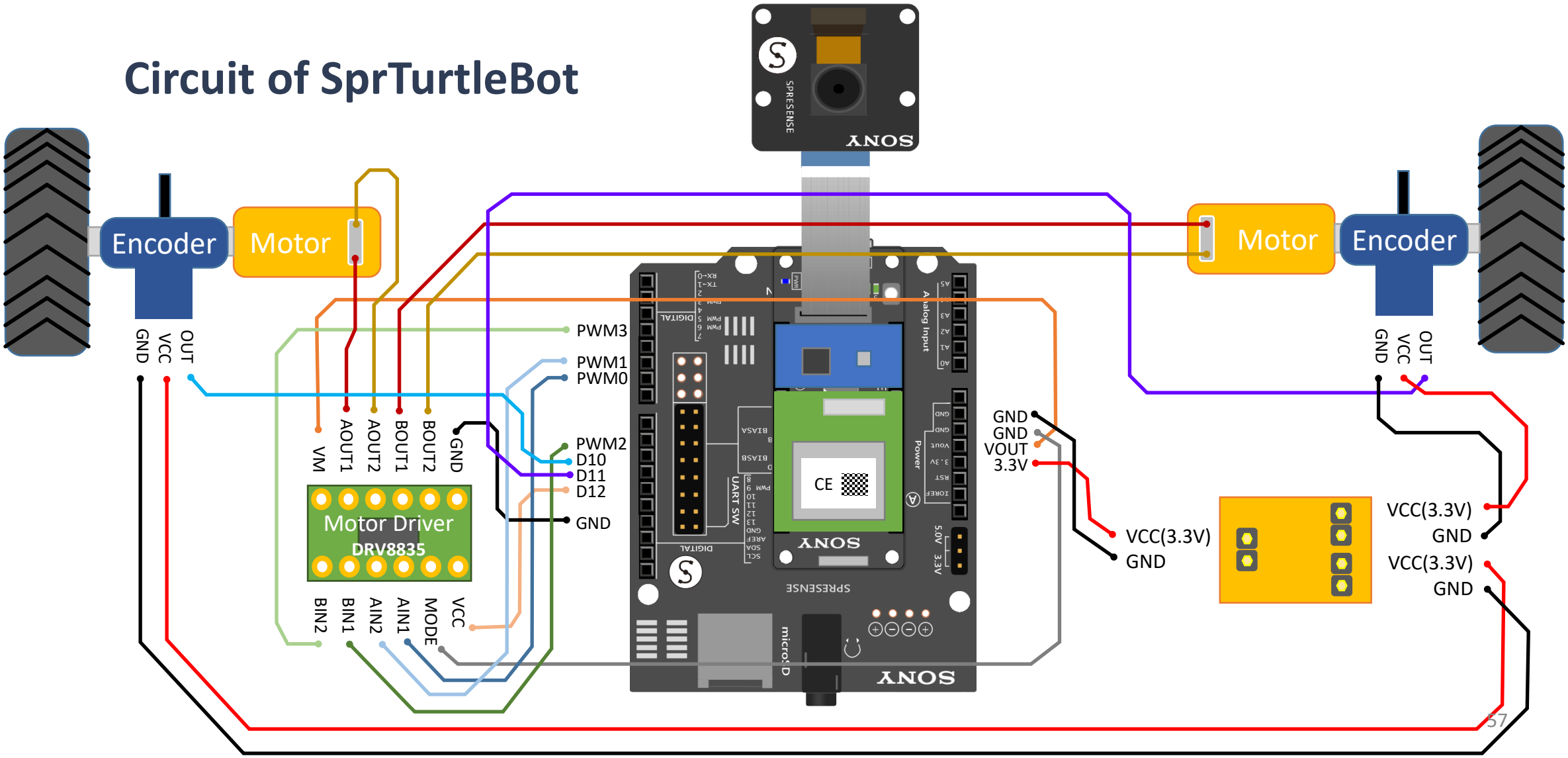
Spresense Main Board
Spresense Extension Board
Spresense WiFi Add-on Board

Universal PCB to distribute
3.3V power lines

DC Motor with Tire

Make SprTurtleBot

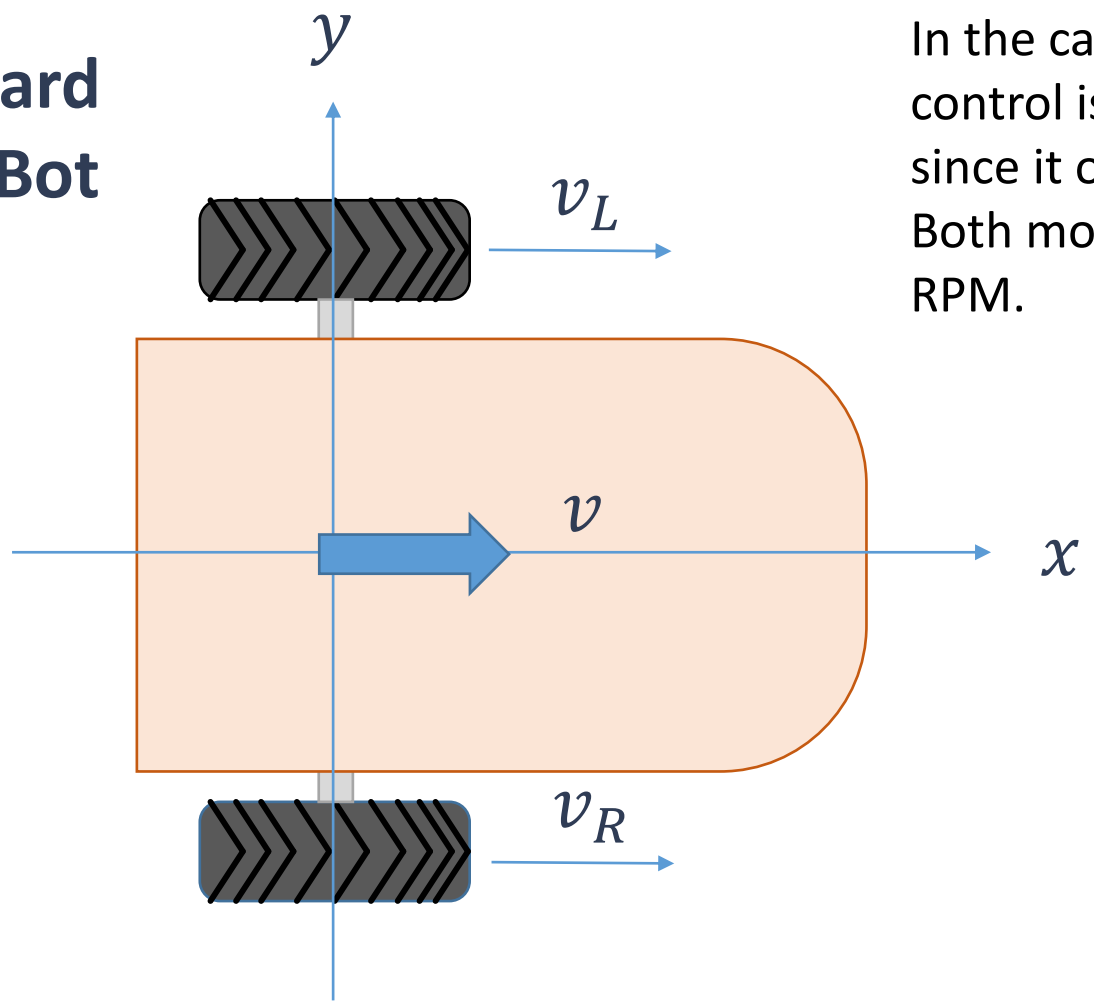
Circuit of SprTurtleBot



Make SprTurtleBot

Forward and backward control of SprTurtleBot

$$v = v_L = v_R$$



In the case of the SprTurtleBot, the control is only in the x direction, since it only moves back and forth. Both motors must have the same RPM.

Make SprTurtleBot

Rotation control of SprTurtleBot

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

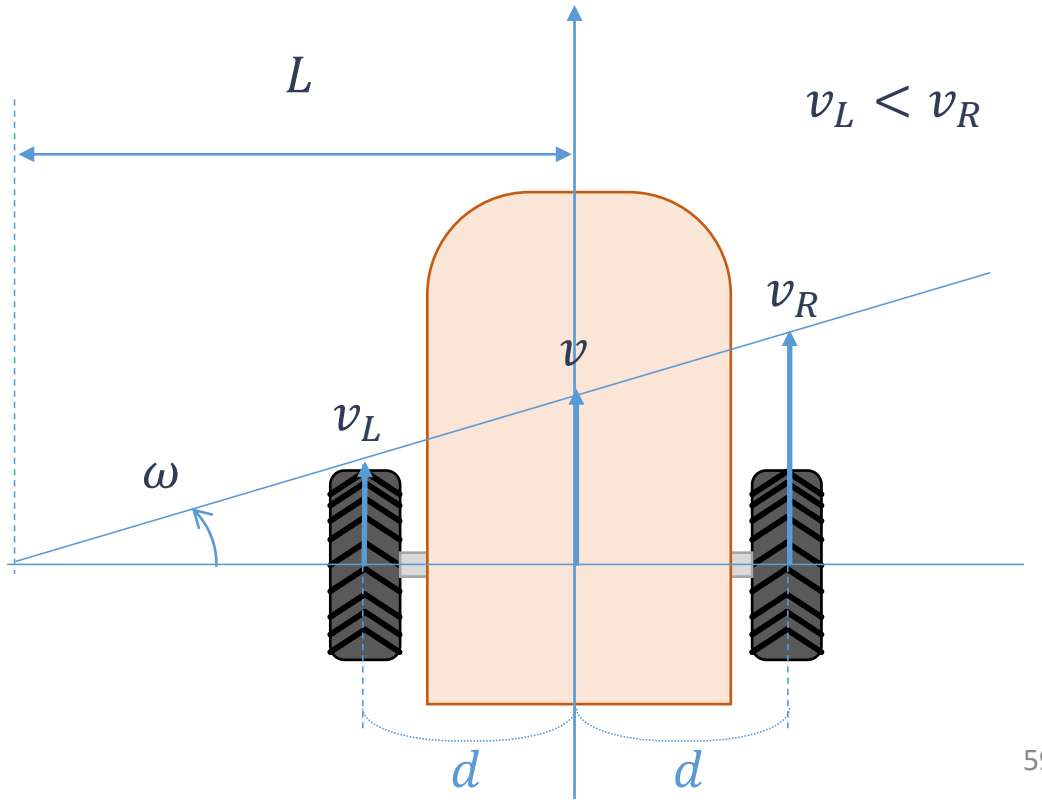
From (1) and (2), solve for d

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

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

From (3), (4), (5) $v = \frac{(v_R + v_L)}{2}$

Different speeds on the left and right sides cause velocity ω in the direction of rotation.



Make SprTurtleBot

Self positioning estimation of SprTurtleBot

$If\ \Delta\theta\ is\ small\ ,\ \ T' \cong T$

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

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

On the other hand,

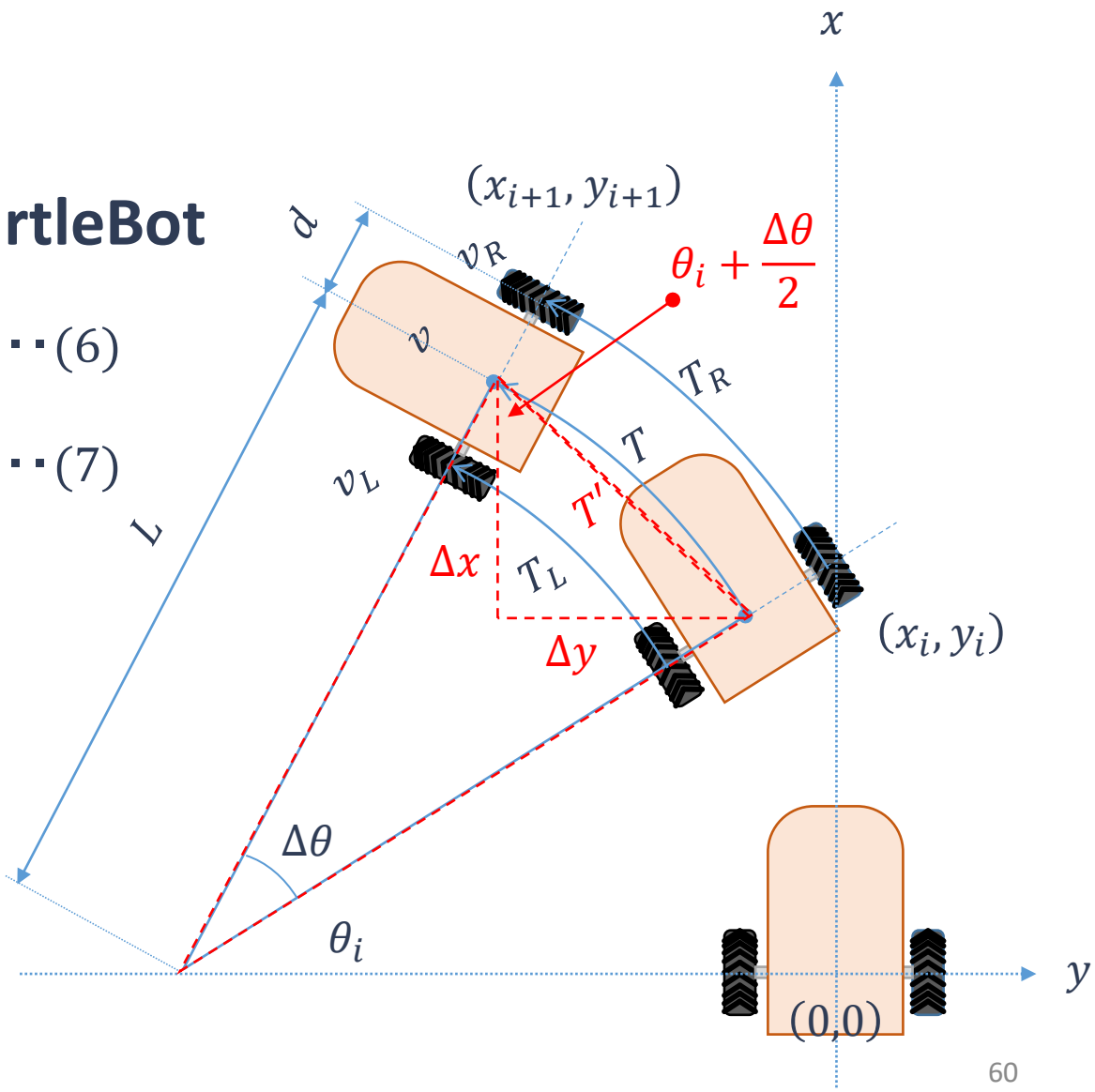
$$T = L\Delta\theta \quad \cdots(8)$$

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

From (4), (5) and (8), (9)

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

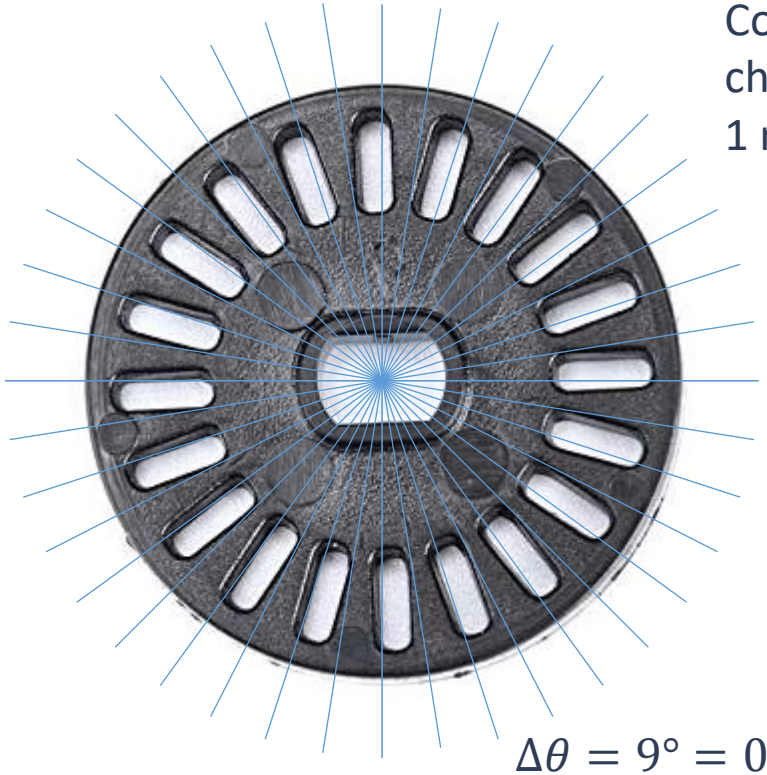
$$\Delta\theta = \frac{(v_R - v_L)\Delta t}{2d} \quad \cdots(11)$$



Make SprTurtleBot

Calculation speed of SprTurtleBot

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



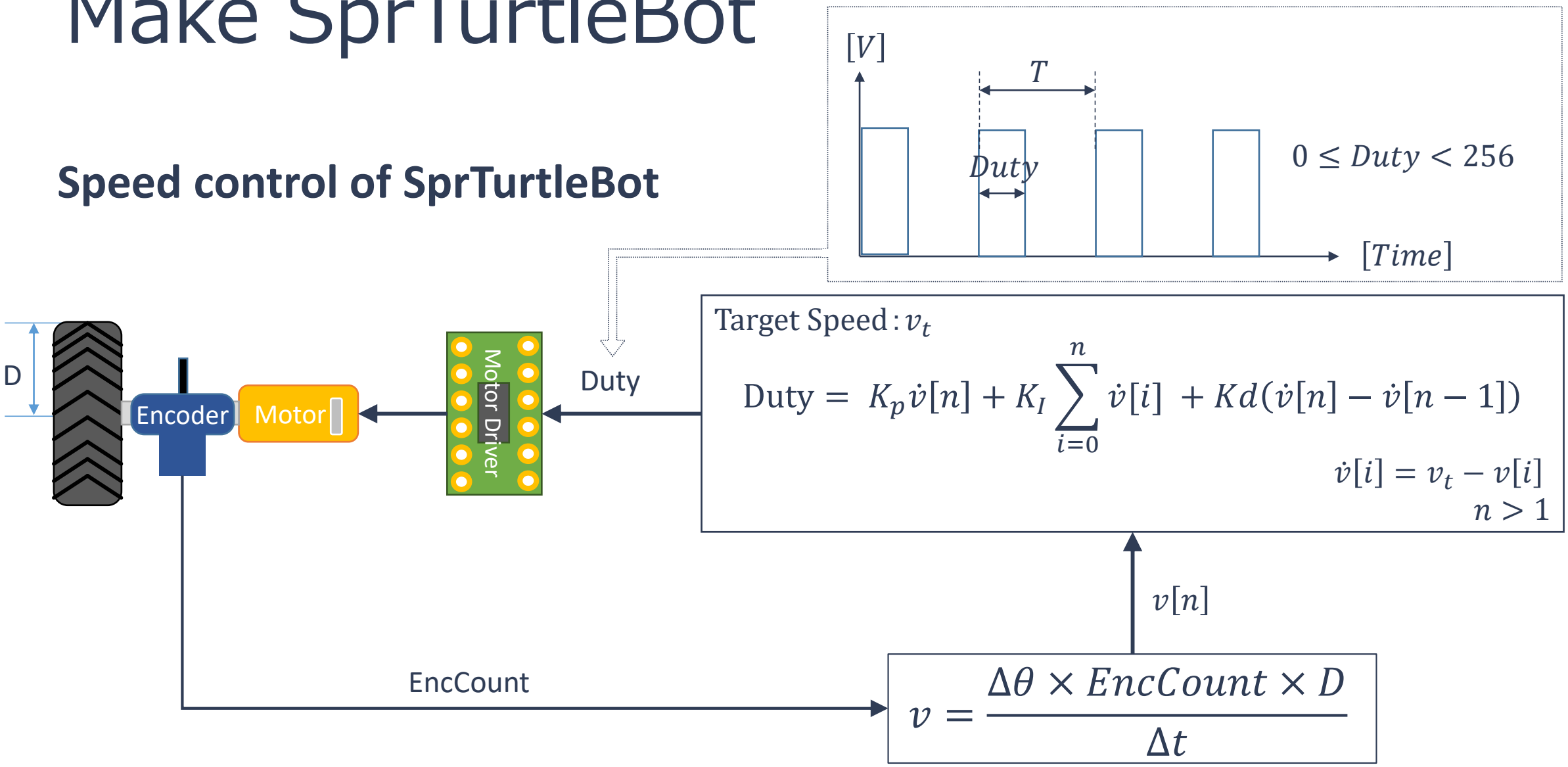
Counting when a sensor change occurred
1 round (40 counts)

$D = 0.06m$

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

Make SprTurtleBot

Speed control of SprTurtleBot



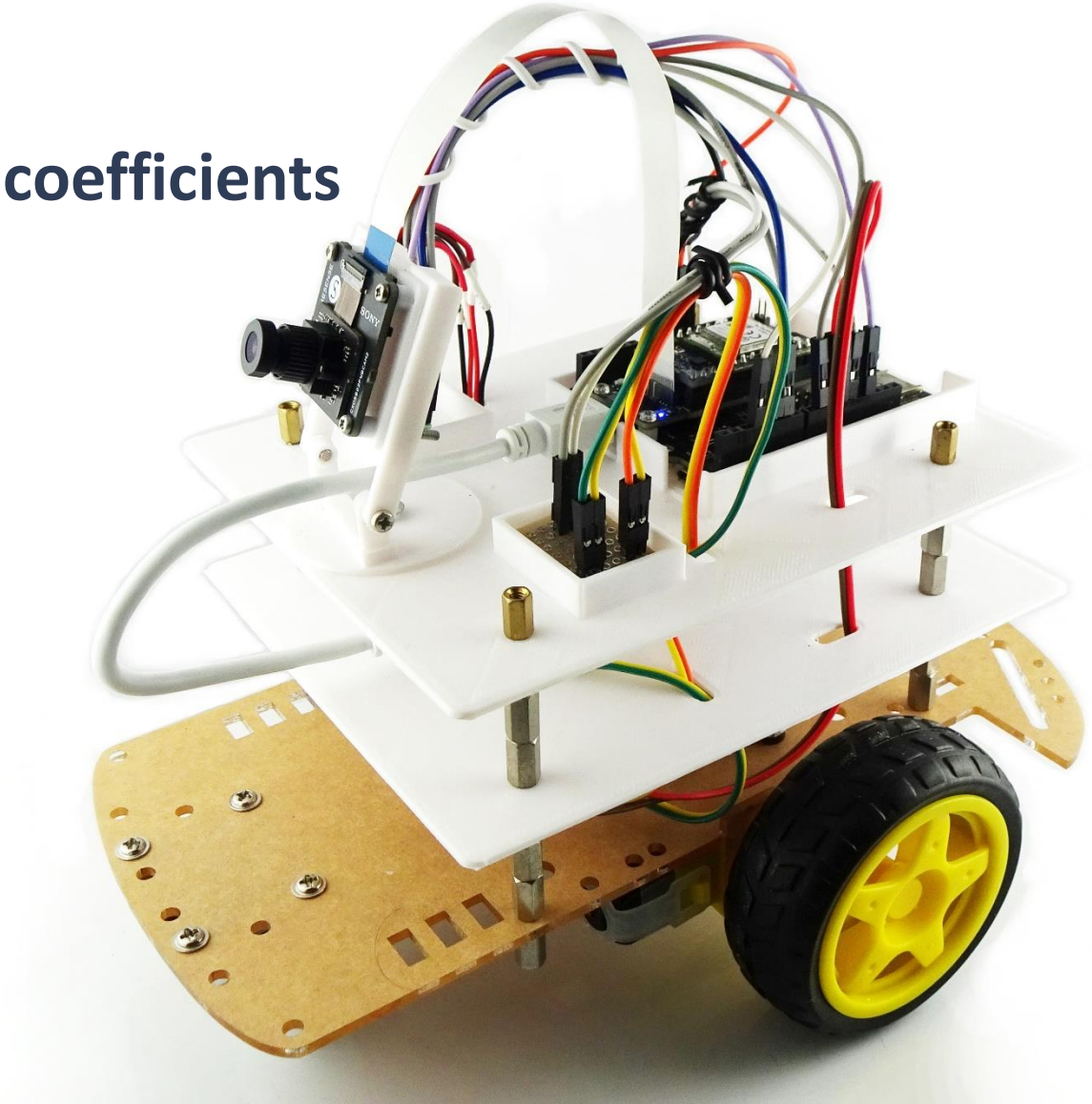
Make SprTurtleBot

Adjustment of PID coefficients for SprTurtleBot

- Adjust brush motors individually to achieve the same response on both sides due to the significant variation in the characteristics of brush motors.
- Adjustment should always be made with the car running on the ground to take into account the required torque due to its own weight and friction.
- The number of input and output digits is used as a reference for the adjustment value (number of adjustment value digits = number of output digits / number of input digits)
- Adjustment to increase output torque because of high friction and torque required at initial startup.

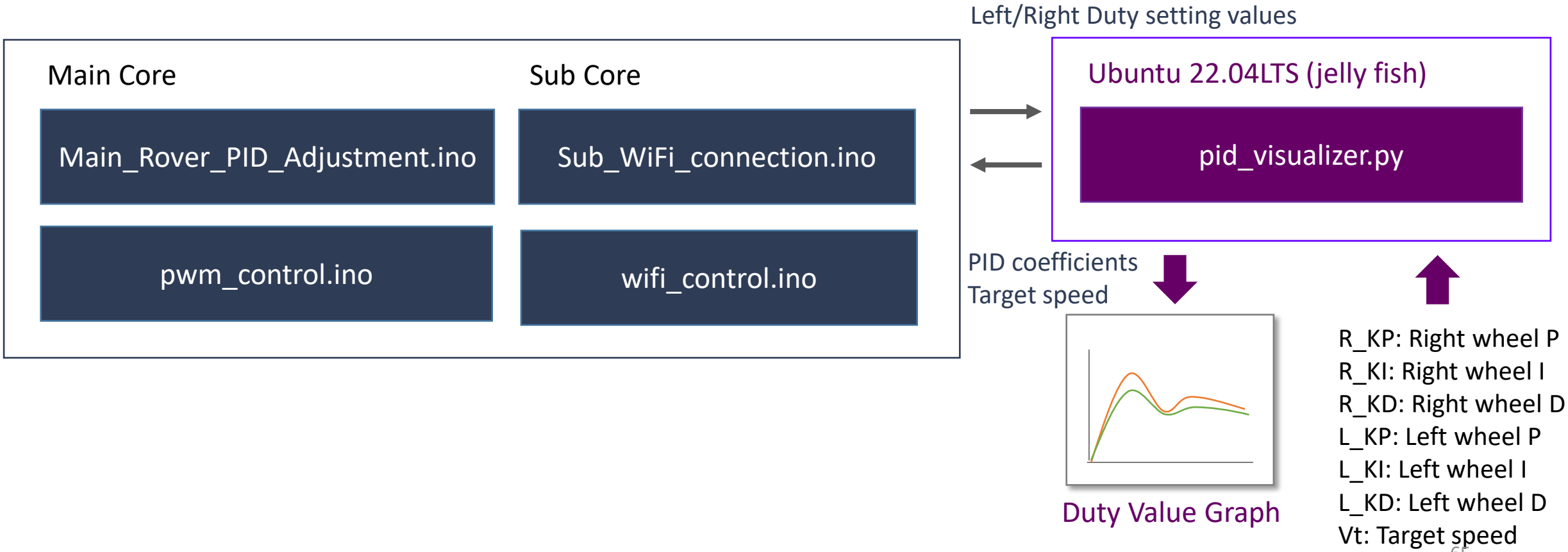
Make SprTurtleBot

Adjustment of PID coefficients



Make SprTurtleBot

Configuration of SprTurtleBot's PID adjustment program



Make SprTurtleBot

Main_Rover_PID_Adjustment.ino

```
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);   // Count any change in the right encoder.
  attachInterrupt(L_EN, Encoder1, CHANGE);    // Count any change in the left encoder.
…
}
```

Counting Encoder Changes

```
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_;
}
```

Tire Speed Calculation

```
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;  // reset counter
  interrupts();
  float R_Vm = calc_speed(cur_R, duration, &R_mileage);   // Right wheel speed
  float L_Vm = calc_speed(cur_L, duration, &L_mileage);    // Left wheel speed
  …
  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 Control

SprTurtleBotの制作

pid_visualizer.py

```
...
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)
```

Receive data sent from SprTrutlBot

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

Retrieve and send parameters entered by the user

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

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

Start user input thread

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

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

Start data receiving thread

```
  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()
```

Display Graph

Make SprTurtleBot

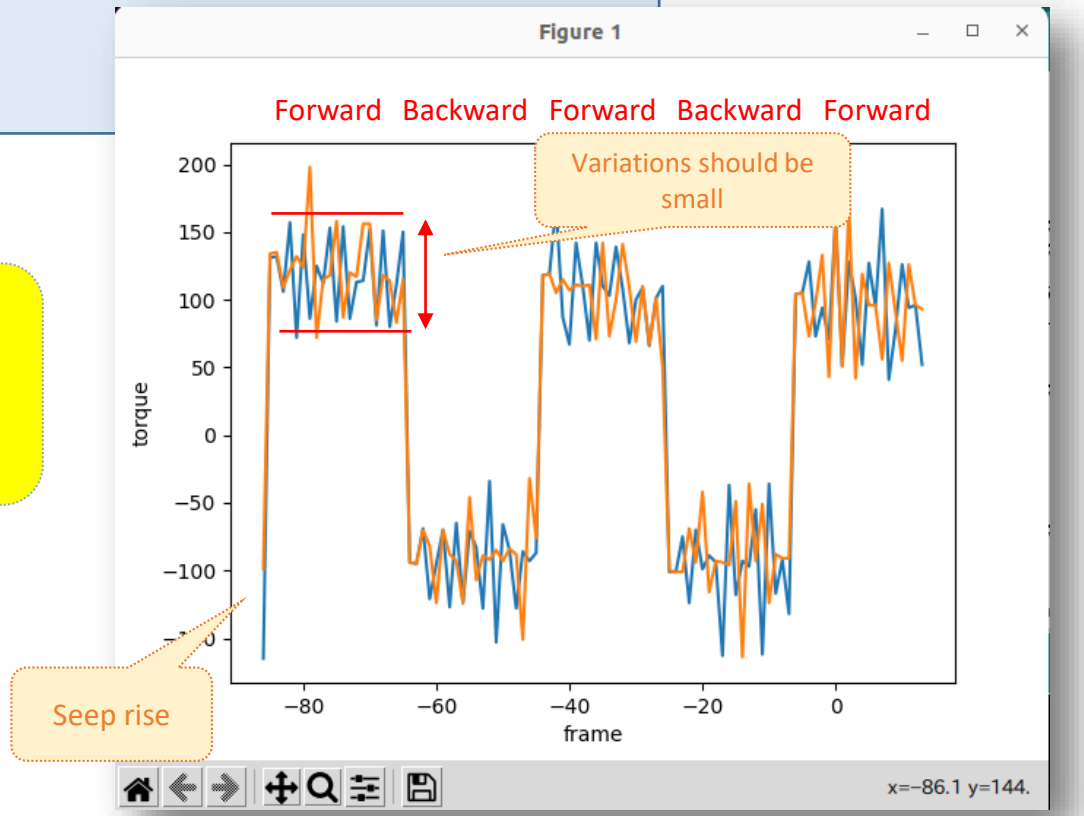
```
$    python3 pid_visualizer.py
Wainting for SprTurtleBot access
Please reset SPrTurtleBot and wait for a moment…
Connected!! [192.168.2.105]
input: 200,30,5,200,30,5,1.0,0
```

When the rover is reset, the "Connected! message appears and the graph is displayed

Sets the PID coefficient. The arguments are in the following order.

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

where a positive value specifies straight ahead and a negative value backward.
Rot is 0: no rotation, 1: counterclockwise rotation, -1: clockwise rotation.

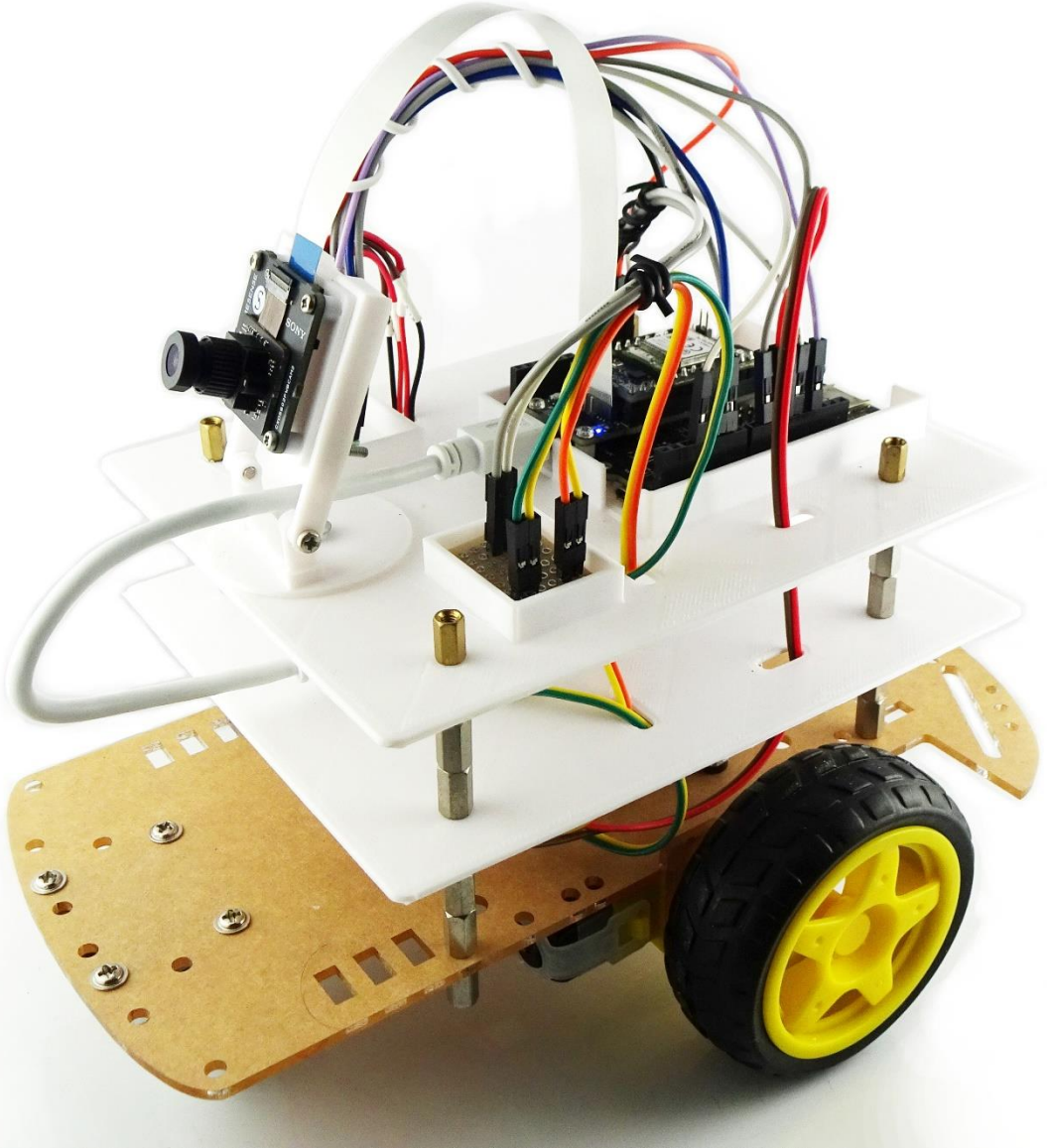


The value of the coefficient can be quickly found by referring to the range of output and input values. In this case, assuming the output range (0-255) and input range (1.0-5.0 m/s), the coefficient of Kp can be estimated to be around 200. The coefficients are adjusted only by the value of Kp at first (Ki and Kd are once set to zero), and then the values of Ki and Kd are adjusted while watching the waveform.

Make SprTurtleBot

Connection to ROS2

micro-ROS puts ROS 2 onto microcontrollers



Make SprTurtleBot

Operating Specification of SprTurtleBot

Operation	Topic name	pub/sub	Message Type	Memo
Motor ON/OFF	motor_power	subscriber	std_msgs/msg/Bool	Instructs On/Off of motor driver
Reset	reset	subscriber	std_msgs/msg/Empty	Instructs the system to reboot
Control	cmd_vel	subscriber	geometry_msgs/msg/Twist	Instructs forward/backward, rotation of SprTurtleBot
Take Picture	take_picture	subscriber	std_msgs/msg/Empty	Instructs to take picture
Picture Data	image/compressed	publisher	sensor_msgs/msg/CompressedImage	Instance of picture data
Odometry	odom	publisher	nav_msgs/msg/Odometry	Positioning data of SprTurtleBot

Make SprTurtleBot

Configuration of SprTurtleBot's control program

Main Core

Main_micro-ROS_subpub

Sub Core

Sub_Rover_Control.ino

pwm_control.ino

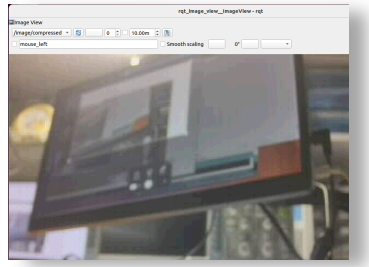
Picture Data
Odometry

Motor ON
Reset
Control
Take picture

Ubuntu 22.04LTS (jelly fish)

ROS2

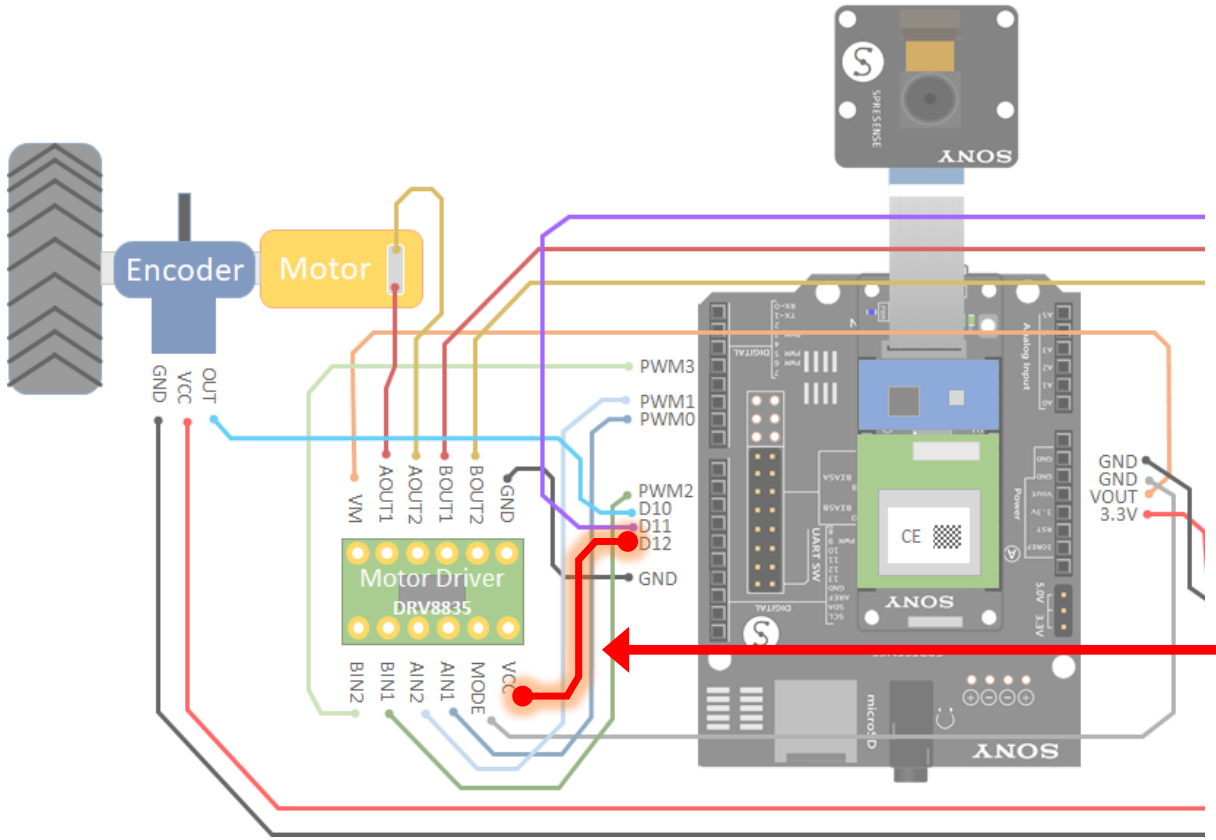
Pitcure Data



rqt_image_view

Motor ON/OFF
Reset
Control
Take picture

Make SprTurtleBot (Motor ON/OFF)



Setting pin D12 HIGH turns on logic power and enables the motor driver

Make SprTurtleBot (Motor ON/OFF)

Main_micro-ROS_subpub.ino (excerpt)

```
...
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);
}
```

When a motor message comes from ROS2, transfer the message to Subcore on Spresense

```
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));
}
```

Register the subscriber in the executor

Sub_Rover_Control.ino (excerpt)

```
#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;
      }
      ...
    }
    ...
  }
  ...
}
```

Motor Switch Pin (D12 pin) Control

Make SprTurtleBot (Motor ON/OFF)

Operation Check

Motor ON: instructs from ROS2 on Ubuntu

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

Motor OFF: instructs from ROS2 on Ubuntu

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

Make SprTurtleBot (Reset Control)

Main_micro-ROS_subpub.ino (excerpt)

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

void res_callback(const void * msgin) {
  LowPower.reboot();        // system reboot
}                                   Reset Spresense when a reset request comes from ROS2

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));
…                                   Register Reset subscriber in Executor
}
```

Make SprTurtleBot (Reset control)

Operation Check

System Resest: instructs from ROS2 on Ubuntu

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

Make SprTurtleBot (Control)

Structure of geometry_msgs/msg/Twist

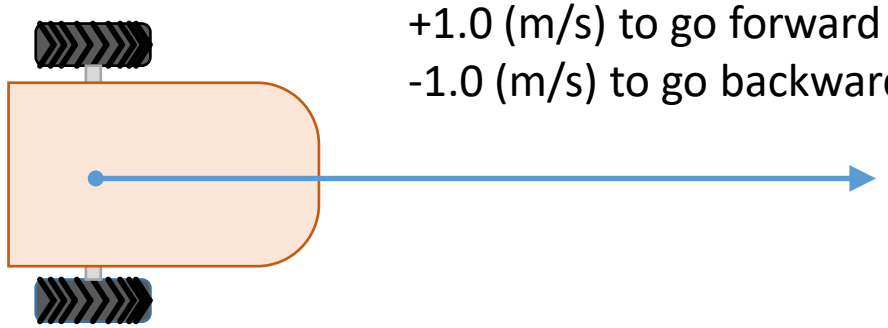
Topic name	Type	Parameter	Memo	Unit
spr_turtle/cmd_vel	linear	x	Move forward(+)/backward(-) at a specified speed	(m/s)
		y	N/A	
		z	N/A	
	angular	x	N/A	
		y	N/A	
		z	Rotate clockwise(+) or counterclockwise (-) at a specified speed	(rad/s)

Make SprTurtleBot (Control)

Operation Check

geometry_msgs/msg/Twist (Forward/Backward)

```
$    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}}”
```



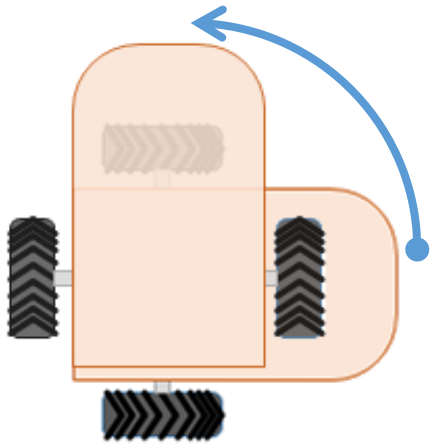
+1.0 (m/s) to go forward for 1 second
-1.0 (m/s) to go backward for 1 second

Make SprTurtleBot (Control)

Operation Check

geometry_msgs/msg/Twist (Rotation)

```
$    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

Make SprTurtleBot (Control)

Operation Check

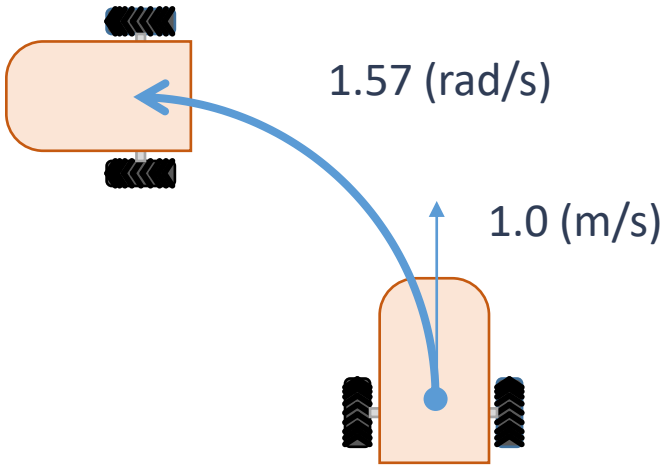
geometry_msgs/msg/Twist (Combination)

```
$    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}}”
```

Move forward at 1.0 (m/s), rotate at 1.57 (rad/s) → What does this mean?

Make SprTurtleBot (Control)

geometry_msgs/msg/Twist (Combination)



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

If we organize these two equations for v_R and v_L, we get

$$v_R = v + d \times \omega$$
$$v_L = v - d \times \omega$$

⇦ This equation works in forward/backward move and rotation move

In this case, $v_R = 1.21$

$v_L = 0.79$

Make SprTurtleBot (Control)

Main_micro-ROS_subpub.ino (excerpt)

```
…
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);
}
```

The message coming from ROS2 on Ubuntu is passed to the sub-core instantly.

```
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 (excerpt)

```
…
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.
    }
  }
  ….
```

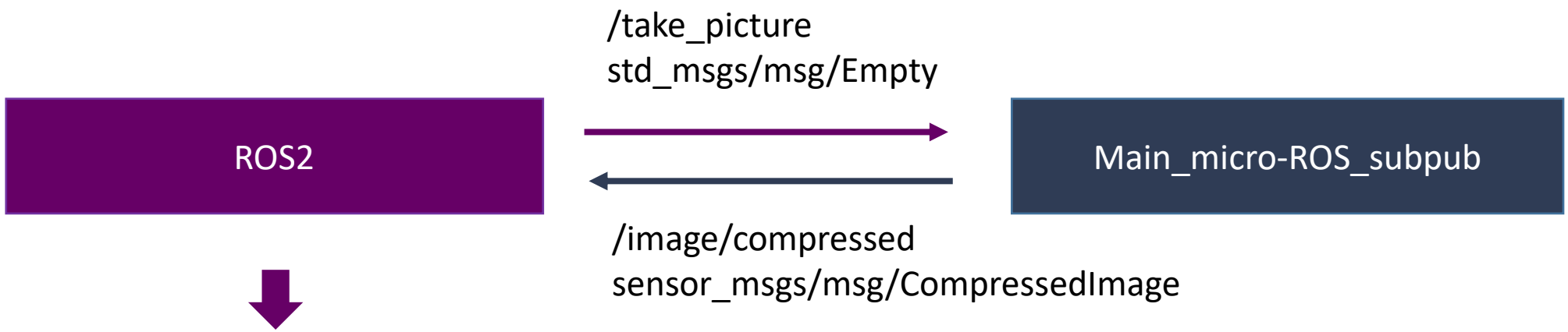
Get forward/backward speed and rotational speed from the message and converts to left and right tire speed

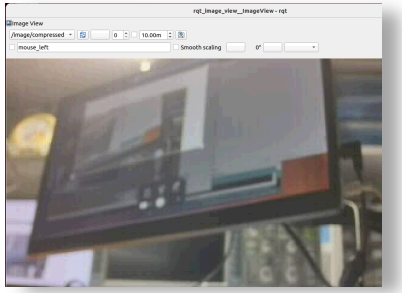
```
  if (current_time - start_time > 1000) { // stopped after 1sec of the initiation
    pwm_control(0, 0);  pwm_control(1, 0);
    pwm_control(2, 0);  pwm_control(3, 0);
    VRt = VLt = 0.0;    R_Vm = L_Vm = 0.0;   R_duty = L_duty = 0;
    R_last_err = L_last_err = 0.0;    R_err_integ = L_err_integ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

Stop 1 second after start of operation

Make SprTurtleBot (Take Picture)

When a “take_picture” topic is received, take a picture and publish the image





rqt_image_view

Make SprTurtleBot (Take Picture)

Main_micro-ROS_subpub.ino (excerpt)

```
...
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);
}
```

When ROS2 receives a "take_picture" request, take a picture by the camera and distributed as a topic.

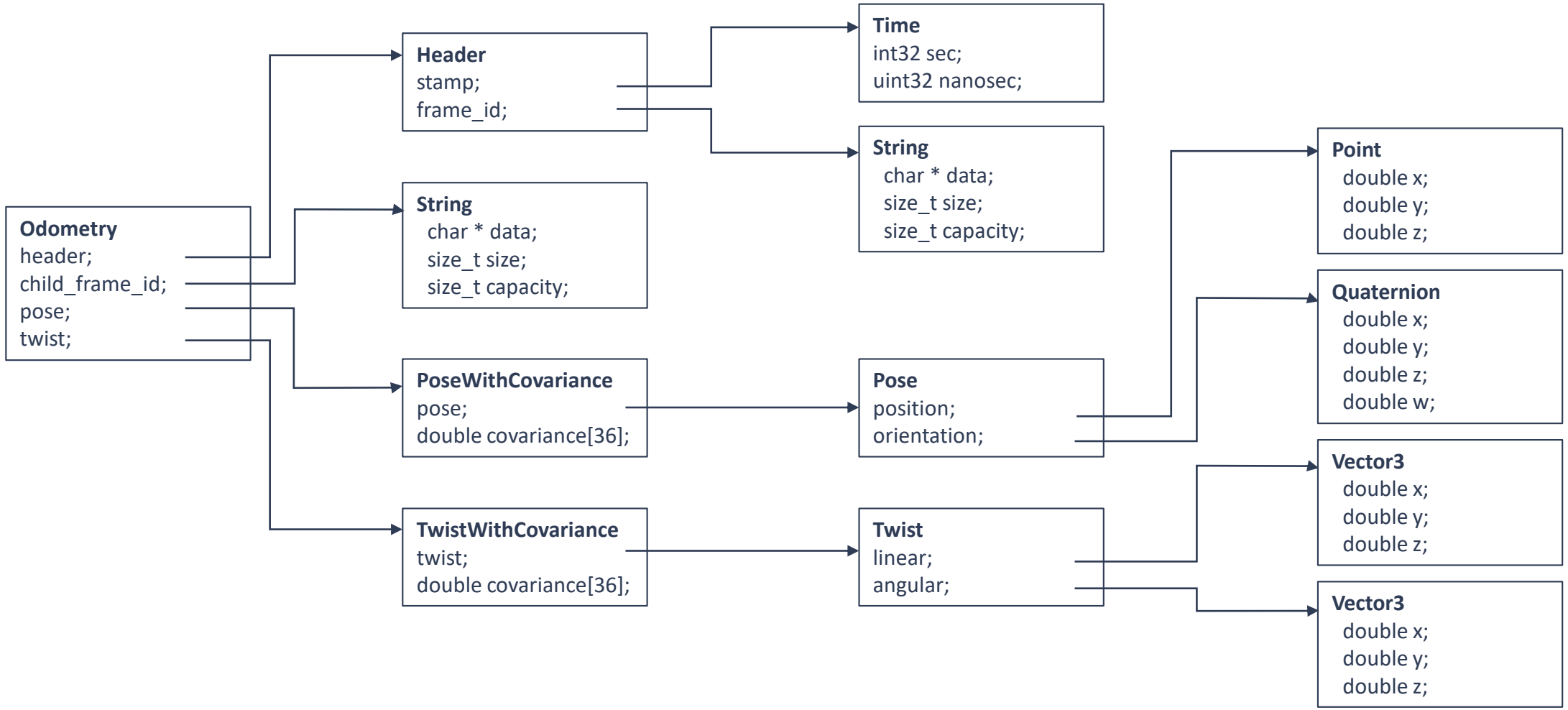
```
void setup() {
  ...
  RCCHECK(rclc_subscription_init_default(&pic_subscriber,   &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Empty),  "take_picture"));
  RCCHECK(rclc_publisher_init_default(&img_publisher,  &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, CompressedImage),
    "image/compressed"));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

  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);
}
```

Create the subscriber of “take_picture” and create the publisher of “compressed/image”

Register the subscriber in the executor

Make SprTurtleBot (Odometry)

Structure of nav_msgs/msg/Odometry



Make SprTurtleBot (Odometry)

Information that needs to be set in nav_msgs/msg/Odometry

Parameter	Type	set value	Unit
odometry.header.stamp.sec	int32	Elapsed time after startup in seconds	(sec)
odometry.header.stamp.nanosec	uint32	Decimal point of elapsed time after startup	(nano sec)
odometry.header.frame_id.data	char*	Constant character of “odom”	
odometry.header.frame_id.size	size_t	Constant number: 4	
odometry.header.frame_id.capacity	size_t	Constant number: 5	
odometry.pose.pose.position.x	double	Distance in the X axis from the starting point	(m)
odometry.pose.pose.position.y	double	Distance in the Y axis from the starting point	(m)
odometry.pose.pose.orientation.z	double	z-term of Quaternion	
odometry.pose.pose.orientation.w	double	second-term of Quaterniaon	
odometry.twist.twist.angular.z	double	Rotation in the Z axis from the starting point	(degree)

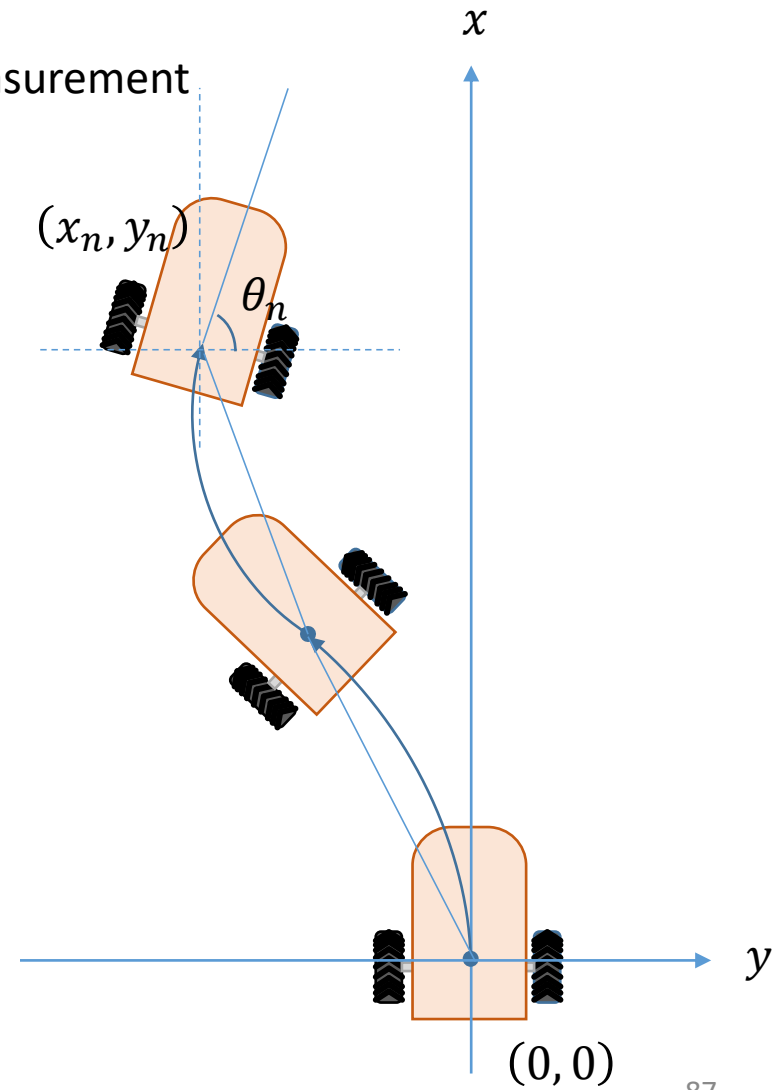
Make SprTurtleBot (Odometry)

From equations (6), (7), (10), and (11) on p59, if the present is the n[th] measurement

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

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

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



Assuming that the quaternion has no rotation in the X- and Y-axes, it can be expressed as the equation right.

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

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

Make SprTurtleBot (Odometry)

Main_micro-ROS_subpub.ino (excerpt)

```
#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));
    }
  }
}
```

Timer for publishing odometry every second

```
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)));
}
```

Register Timer

Make SprTurtleBot (Odometry)

Sub_Rover_Control.ino (excerpt)

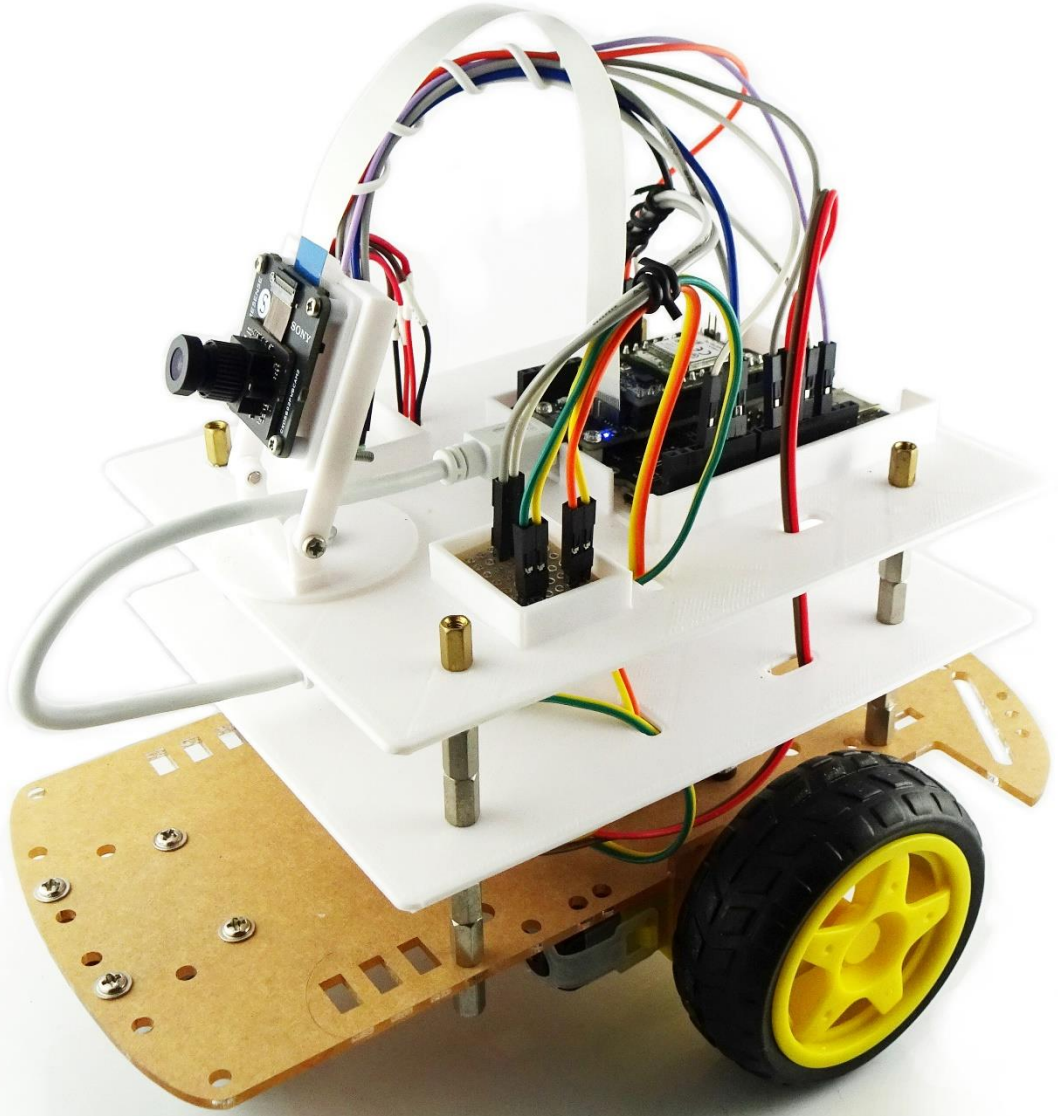
```
#define REQ_ODOM      102
struct rover_odm {
  float odm_ang_z;
  float odm_pos_x;
  float odm_pos_y;
  float odm_qt_qz;
  float odm_qt_qw;
};
struct rover_odm rover_odm;
…
void loop() {
  int8_t recvid; void* msgin;
  int ret = MP.Recv(&recvid, &msgin);
  if (ret > 0) {
    if (recvid == MOTOR_POWER_MSG) {
      …
    } else if (recvid == COMMAND_MSG) {
      …
    } else if (recvid == REQ_ODOM) {
      int8_t sndid = REQ_ODOM;
      MP.Send(sndid, &rover_odm);
    }                                   Returns a value when odometry is requested by the main core
    …
    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;
  }                                     Calculate odometry values and store them in the structure
  …
  delay(DELAY_TIME);
}
```

Make SprTurtleBot (Odometry)

Operation Check

```
$    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
```

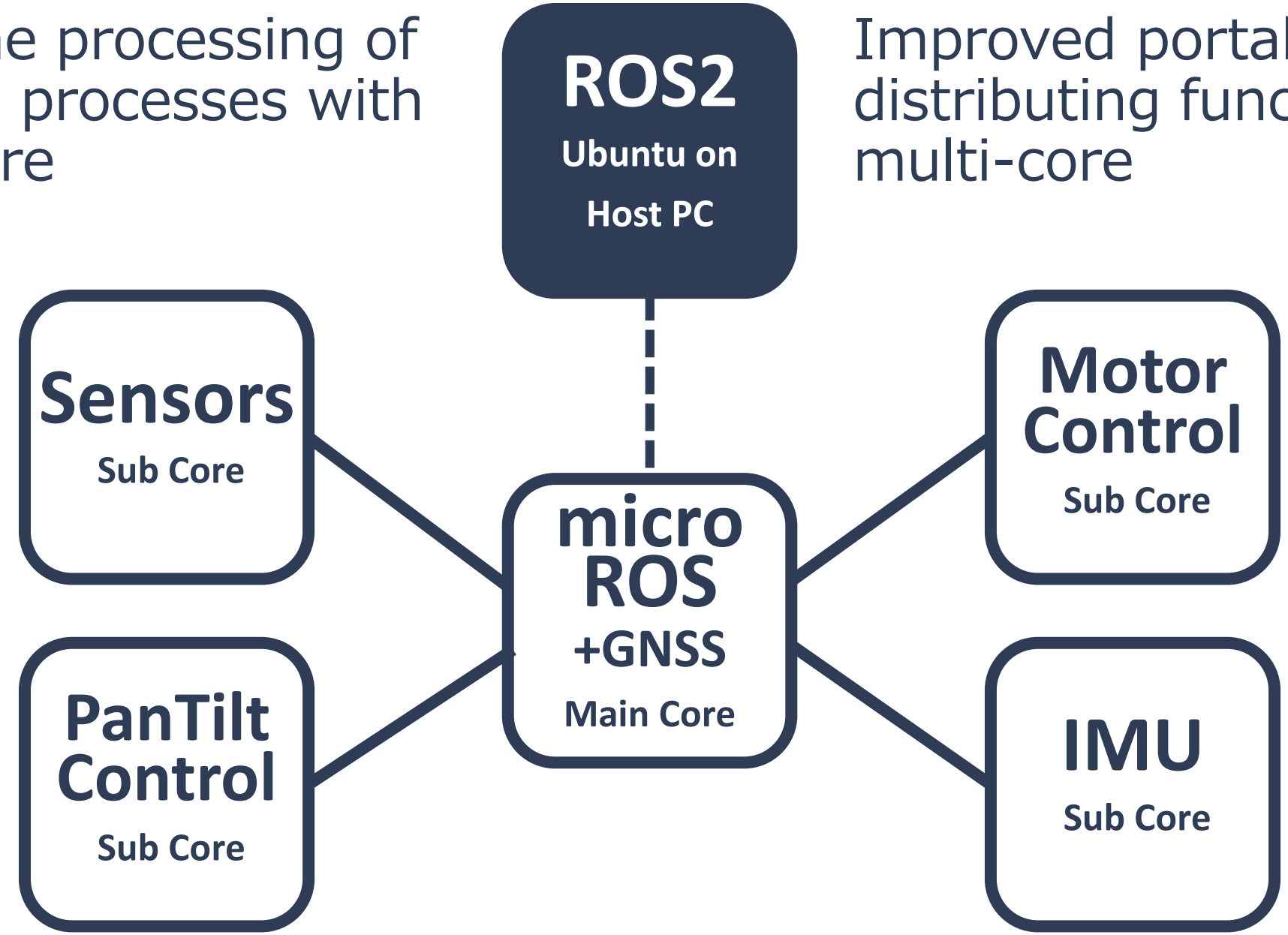


micro-ROS puts ROS 2 onto microcontrollers

SprTurtleBot Further Implementation

Real-time processing of complex processes with multi-core

Improved portability by distributing functions to multi-core



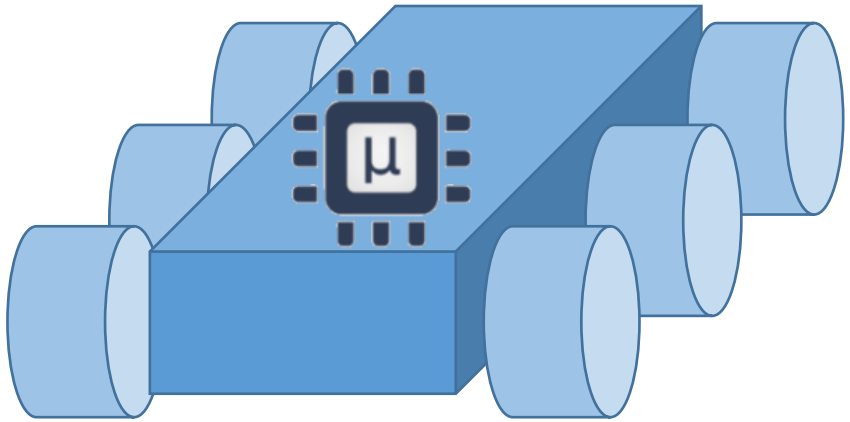
The msg structure of GNSS positioning data

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



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

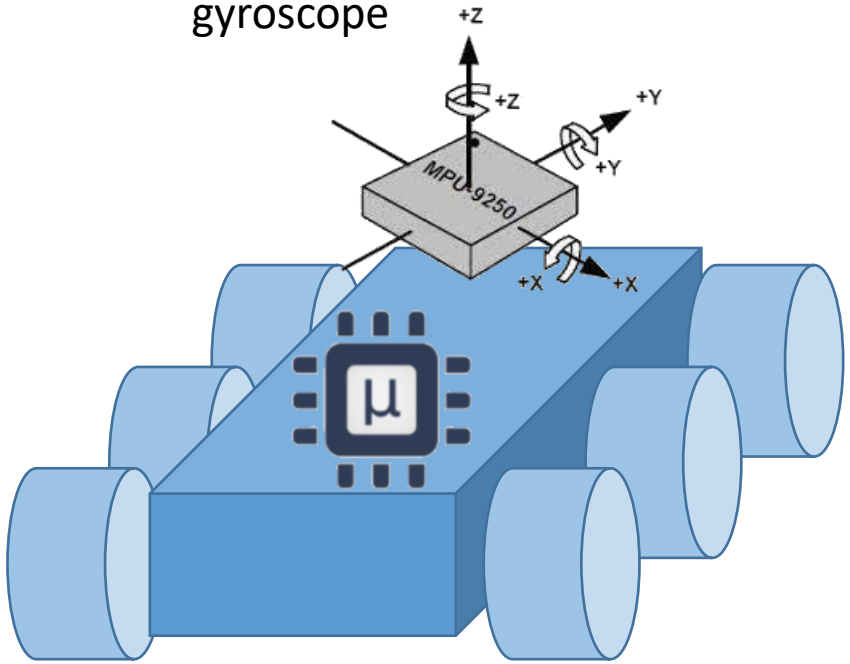
float64 x
float64 y
float64 z

geometry_msgs/Vector3.msg

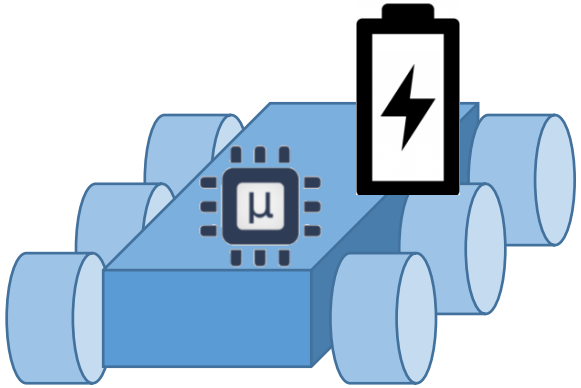
float64 x
float64 y
float64 z

The msg structure of IMU data

IMU (6DoF sensor)
accelerometer
gyroscope



バッテリー（+Fuel Gage）



The msg structure of Battery management

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
```

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

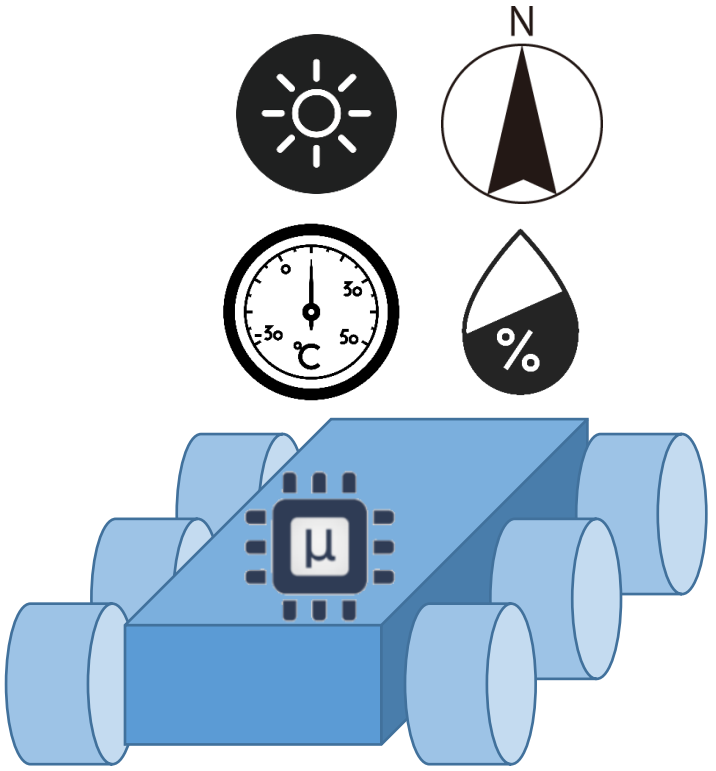
```
Header header                # timestamp is the time the temperature was measured
                             # frame_id is the location of the temperature reading
float64 temperature          # Measurement of the Temperature in Degrees Celsius
float64 variance             # 0 is interpreted as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

The msgs for various sensors

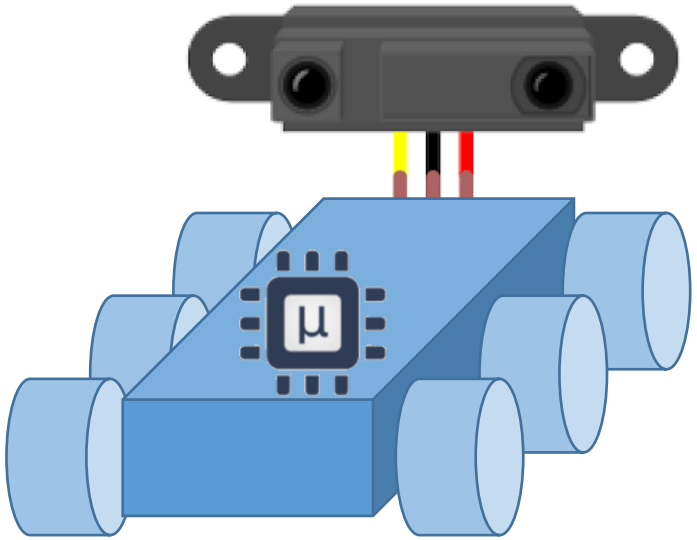
Illuminance	Temperature
MagneticField	Humidity



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]
```

The msg structure of distance sensor data (IR, Super sonic etc..)



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

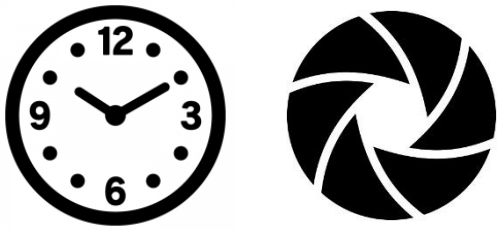
```
Header header                # Header timestamp should be acquisition time of image
uint32 height
uint32 width
float64[] D                  # The distortion parameters
float64[9]  K                # 3x3 row-major matrix
float64[9]  R                # 3x3 row-major matrix
float64[12] P                # 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
```

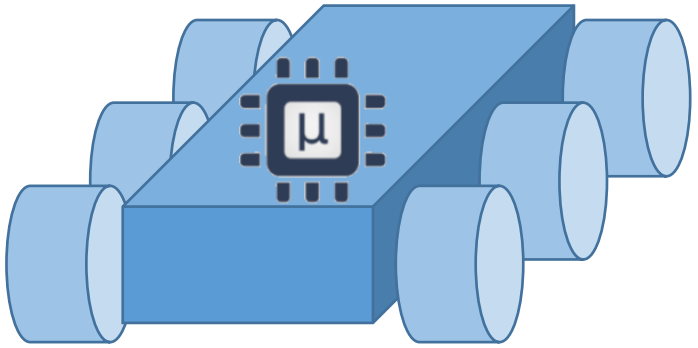
sensor_msgs/RegionOfInterest.msg

```
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).
```

Other msgs

Clock（GPS Clock, Radio Clock）
Camera information etc…





To learn more about ROS2/micro-ROS

ROS2 Documents (humble)
https://docs.ros.org/en/humble/index.html

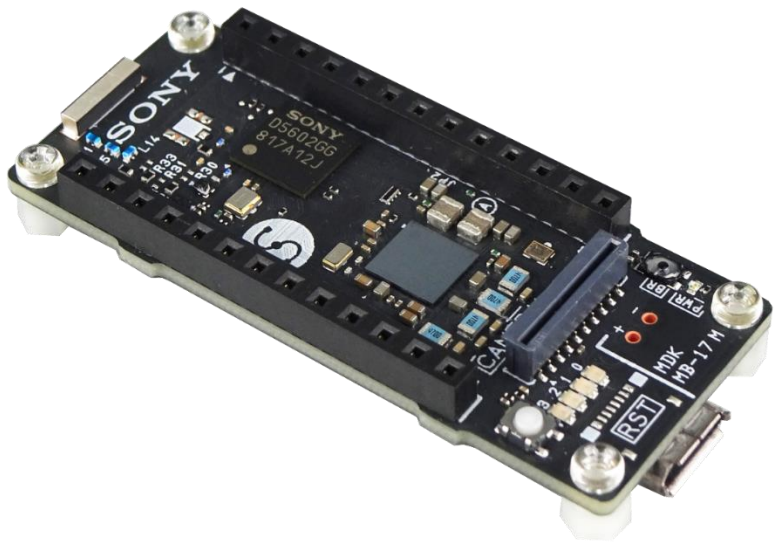
ROS2 Tutorials (humble)
https://docs.ros.org/en/humble/Tutorials.html

micro-ROS Documents
https://micro.ros.org/

micro-ROS Tutorials
https://micro.ros.org/docs/tutorials/core/overview/

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

Expand the world of ROS2/micro-ROS with Spresense!



- ✓ Rich computing power and flexible distributed design with multi-core
- ✓ Battery saving due to low power consumption
- ✓ Autonomous control robot combined with positioning capability

SPRESENSE