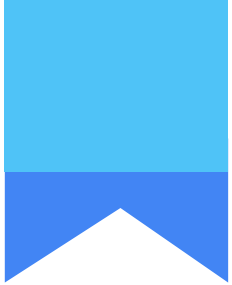
ROS SERIAL and OpenCR

- **OpenCR로 ROS 시리얼 구현하기**

Yoonseok Pyo

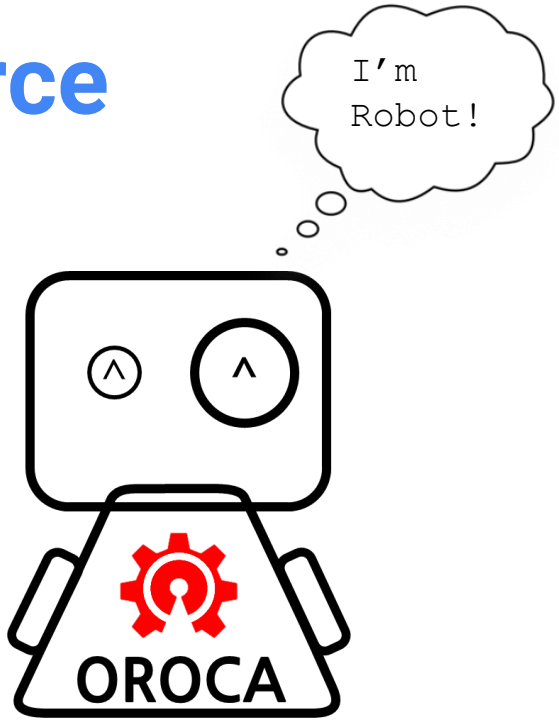


Robot Operating System

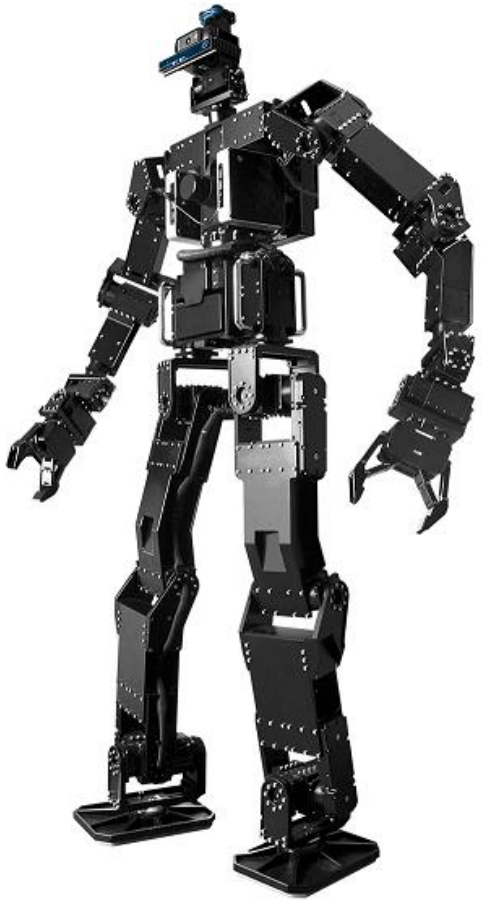
Best tool for your robot!

https://youtu.be/Z70_3wMFO24

What is the Best Computing Resource for Your Robot?



intel.com / st.com



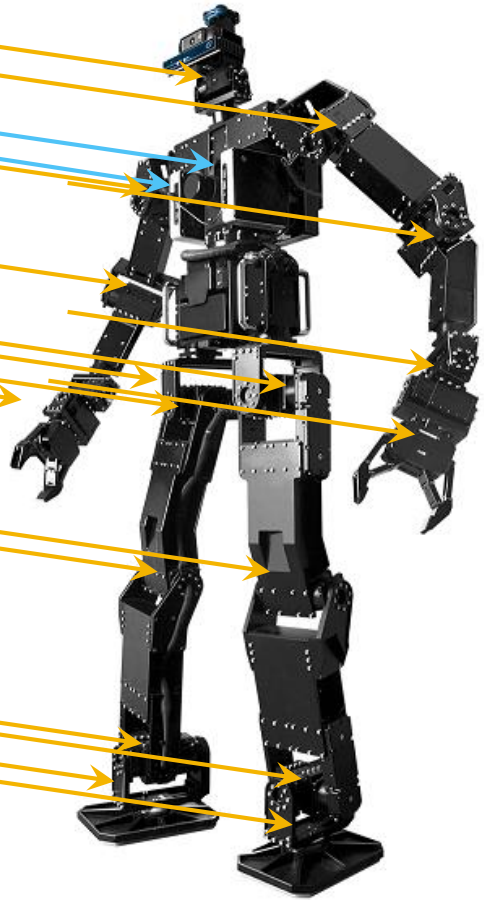
THORMANG3

Intel NUC
for Sensor and Communication

THORMANG3

Intel NUC
for Sensor and Communication

ARM Cortex-M
for Motor control and Analog Sensors



Small embedded systems are everywhere in ROBOT!

THORMANG3

Scope of computing resources : computation vs control





	8/16-bit MCU	32-bit MCU		ARM A-class (smartphone without screen)	x86 (laptop without screen)
		"small" 32-bit MCU	"big" 32-bit MCU		
Example Chip	Atmel AVR	ARM Cortex-M0	ARM Cortex-M7	Samsung Exynos	Intel Core i5
Example System	Arduino Leonardo	Arduino M0 Pro	SAM V71	ODROID	Intel NUC
MIPS	10's	100's	100's	1000's	10000's
RAM	1-32 KB	32 KB	384 KB	a few GB (off-chip)	2-16 GB (SODIMM)
Max power	10's of mW	100's of mW	100's of mW	1000's of mW	10000's of mW
Peripherals	UART, USB FS, ...	USB FS	Ethernet, USB HS	Gigabit Ethernet	USB SS, PCIe

Morgan Quigley (OSRF) "ROS 2 on “small” embedded systems", ROSCon2015

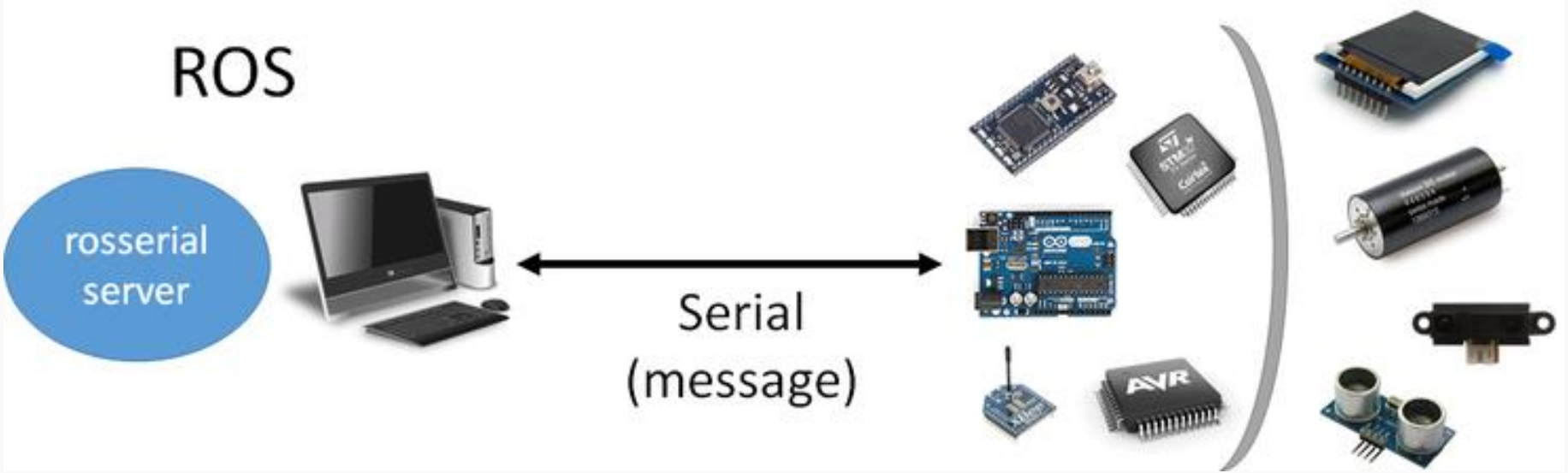
Communication between the computer and the microcontroller

컴퓨터에서 동작하는 ROS 와 8051, AVR, ARM Cortex-M 등의 MCU(Micro Controller Unit)간의 메시지 통신을 위해 rosserial 이라는 ROS의 패키지가 제공되고 있다.

ROS 대부분의 패키지들은 상위레벨의 소프트웨어가 중심인데 로봇을 실제로 동작하기 위해서는 하드웨어와 연결하여 하위레벨 제어가 필요해진다. 즉, 로봇의 이동 명령을 상위레벨의 소프트웨어에서 만들었다고 하더라도 결국은 모터를 제어하는 하위레벨이 필요하다는 것이다. 이때에는 실시간 제어가 가능하도록 위에서 언급한 MCU 들을 이용하게 되는데 이때의 중간 중계역할을 rosserial이 담당한다.

이 세미나에서는 간단히 아두이노의 개발 환경과 아두이노&ROS 관련 라이브러리 생성, pubsub 예제를 이용하여 OpenCR 에서 퍼블리시, 서브스크라이브를 rosserial을 이용하여 컴퓨터의 터미널창에서 확인해보는 과정에 대해서 알아보도록 하자.

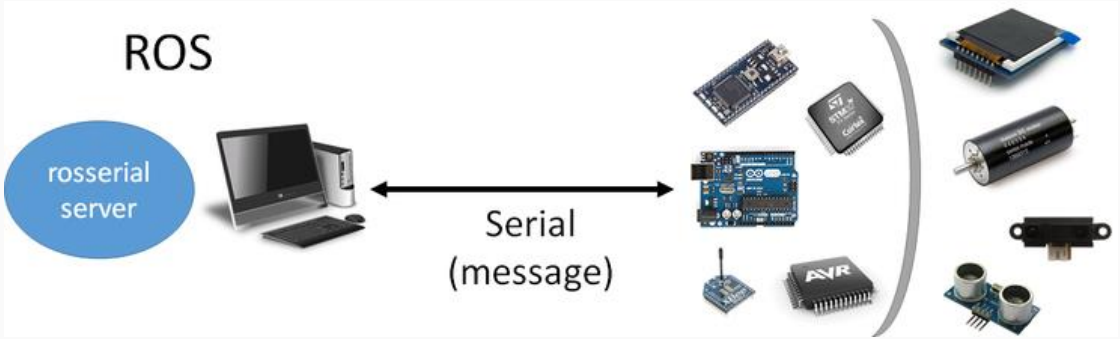
Communication between the computer and the microcontroller



rosserial for embedded system to control the hardware parts of ROBOT!

rosserial 은 아래의 그림과 같이 일반 컴퓨터와 제어에 많이 사용되는 마이크로컨트롤러간의 메시지 통신을 위해 제공되는 ROS의 패키지이다. 이 두 가지(컴퓨터↔마이크로컨트롤러)의 요소는 서로 다른 하드웨어이고 마이크로컨트롤러의 경우 대부분 UART와 같은 시리얼 통신만을 제공하는 경우가 많아 ROS 에서 사용하는 ROSTCP 등을 사용 못하는 경우가 많다.

이 경우의 통신을 rosserial의 rosserial-server가 하드웨어적으로 시리얼(경우에 따라서는 Xbee, 시리얼형 타입의 블루투스도 포함)로 연결된 **두 하드웨어의 중계자 역할**을 하게된다.

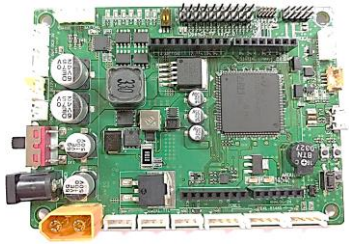


rosserial for embedded system to control the hardware parts of ROBOT!

예를들어, 마이크로컨트롤러에 연결된 센서값을 ADC로 디지털값화 시킨 후 시리얼로 전송하게 되면, 컴퓨터의 rosserial-server 노드는 시리얼 값을 받아서 ROS에서 사용하는 토픽(Topic)으로 변환하여 전송하게 된다.

반대로, ROS 의 다른 노드에서 모터 제어 속도값을 토픽으로 전송하면 rosserial-server 노드는 이를 시리얼로 마이크로컨트롤러에게 전송하고 연결된 모터를 직접 제어하게 되는 것이다.

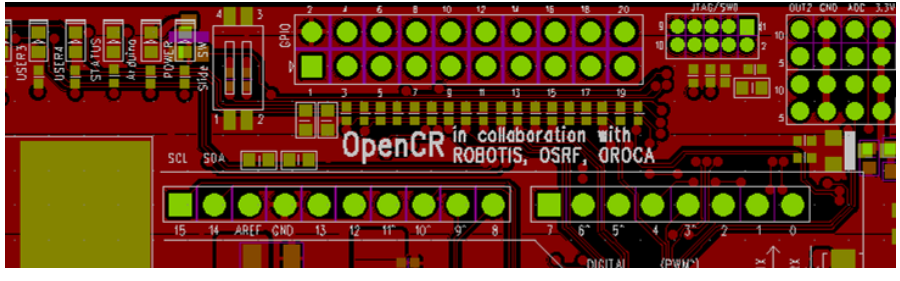
SBC를 포함한 일반 컴퓨터의 경우, 제어에 있어서 실시간이 보장 못하는데 이렇게 보조 하드웨어 제어기로 마이크로컨트롤러를 사용하면 모터 제어단의 실시간성을 확보할 수 있어서 많이 사용되고 있다.



rosserial

OpenCR + ROSSERIAL

이하 설명에서는 컴퓨터에 Ubuntu 16.04 LTS OS와 ROS Kinetic 버전이 설치되어 있다는 전제 하에 OpenCR + rosserial 를 사용하는 법을 설명한다. F/W는 Arduino IDE 1.6.12 에서 개발한다.



rosserial install

다음 명령어로 rosserial과 아두이노 계열 지원 패키지를 설치하도록 하자.

그 이외에도 ros-kinetic-rosserial-windows ros-kinetic-rosserial-xbee, ros-kinetic-rosserial-embeddedlinux 등이 있는데 이는 필요할 경우에만 설치하도록 하자.

```
$ sudo apt-get install ros-kinetic-rosserial ros-kinetic-rosserial-server ros-kinetic-rosserial-arduino
```

Build the ros libs for rosserial

다음으로 아두이노 IDE 에서 사용가능한 라이브러리를 생성하도록 하자.

다음과 같이 아두이노 IDE의 개인폴더의 라이브러리 폴더로 이동한 후 기존 ros_lib 를 모두 삭제한다(처음에는 ros_lib가 없으나 추가로 생성할때는 꼭 기존 폴더를 삭제하도록 하자).

그 뒤, rosserial_arduino 패키지의 make_libraries.py 파일을 실행하여 ros_lib를 생성하자.

```
$ cd ~/Arduino/libraries/
$ rm -rf ros_lib
$ rosrun rosserial_arduino make_libraries.py .
```

Build the ros libs for rosserial

make_libraries.py 실행하게 되면 Arduino IDE 에서 ROS 프로그래밍이 가능하도록 ros.h와 같은 기본 헤더파일들과 설치된 ROS 패키지 및 catkin_ws 에 설치된 개인이 작성한 패키지들의 메시지 파일등의 헤더파일 등이 생성된다.

이는 나중에 아두이노 IDE에서 불러와서 사용할 수 있다.
궁금한 경우 ~/Arduino/libraries/ros_lib 폴더에 생성된 파일들을 확인해보도록 하자.

OpenCR의 시리얼 통신은 기본 아두이노 보드와 조금 틀려서 설정을 하나 변경할 필요가 있다.
~/Arduino/libraries/ros_lib/ArduinoHardware.h 의 59번째 라인의 아래 내용을 변경하자.

(수정전) #define SERIAL_CLASS HardwareSerial

(수정후) #define SERIAL_CLASS USBSerial

Communicating ROS messages between the computer and the OpenCR board

마지막으로 컴퓨터와 OpenCR 보드간의 ROS 메시지 통신을 해보도록 하겠다.
OpenCR 보드를 컴퓨터의 USB단자에 연결하고 다음과 같이 아두이노 IDE를 실행하자.

```
$ arduino
```

그 다음 메뉴에서 [File] > [Sketchbook] > [libraries] > [ros_lib] > [pubsub] 를 선택하고 업로드 버튼을 클릭하여 OpenCR 보드에 업로드를 하도록 하자. 이는 0.5초 단위로 "hello world!"라는 string 타입의 메시지를 퍼블리시(publish)하고 toggle_led 라는 토픽 이름의 LED 제어 신호를 서브스크라이브(subscribe)하는 프로그램이다.

자세한 동작 및 코드 설명에 대해서는 우선 실행을 먼저 하고 눈으로 동작을 확인한 후에 이어서 하도록 하겠다.

Example of rosserial: pubsub

pubsub

```
/*
 * rosserial PubSub Example
 * Prints "hello world!" and toggles led
 */

#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Empty.h>

ros::NodeHandle  nh;


void messageCb( const std_msgs::Empty& toggle_msg){
  digitalWrite(13, HIGH-digitalRead(13));   // blink the led
}

ros::Subscriber<std_msgs::Empty> sub("toggle_led", messageCb );



std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg);

char hello[13] = "hello world!";

void setup()
{
  pinMode(13, OUTPUT);
  nh.initNode();
  nh.advertise(chatter);
  nh.subscribe(sub);
}

void loop()
{
  str_msg.data = hello;
  chatter.publish( &str_msg );
  nh.spinOnce();
  delay(500);
}
```

File Edit Sketch Tools Help

pubsub

```
/*
 * rosserial PubSub Example
 * Prints "hello world!" and toggles led
 */

#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Empty.h>

ros::NodeHandle  nh;


void messageCb( const std_msgs::Empty& toggle_msg){
  digitalWrite(13, HIGH-digitalRead(13));   // blink the led
}

ros::Subscriber<std_msgs::Empty> sub("toggle_led", messageCb );



std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg);

char hello[13] = "hello world!";

void setup()
{
  pinMode(13, OUTPUT);
  nh.initNode();
  nh.advertise(chatter);
  nh.subscribe(sub);
}

void loop()
{
  str_msg.data = hello;
  chatter.publish( &str_msg );
  nh.spinOnce();
  delay(500);
}
```

Done uploading.

```
file name : /tmp/build5f7ff3ea95920cf4d8968ef6ad1e729e.tmp/pubsub.pde.bin
file size : 37 KB
Open port OK
Clear Buffer Start
Clear Buffer End
>>
Board Name : OpenCR R1.0
Board Ver  : 0x16071900
Board Rev  : 0x00000000
>>
flash_erase : 0 : 0.949000 sec
flash_write : 0 : 0.218000 sec
CRC OK 38AC22 38AC22 0.001000 sec
jump_to_fw
```

1 OpenCR Board, OpenCR Bootloader on /dev/ttyACM0

Example of rosserial: pubsub

다음 예제와 같이 roscore를 실행 후,
별도의 다른 터미널창에서 rosserial_python패키지의 serial_node.py를 실행하자.

```
$ roscore
```

```
$ rosrun rosserial_python serial_node.py
_port:=/dev/ttyACM0
```

Example of rosserial: pubsub

이제는 OpenCR 보드에서 퍼블리시하고 있는 메시지를 확인해보자.
다음과 같이 rostopic echo 명령어로 "chatter" 라는 토픽의 메시지를 확인해보자.
0.5초 단위로 메시지를 받고 있음을 확인할 수 있다.

```
$ rostopic echo /chatter
data: hello world!
---
data: hello world!
---
data: hello world!
```

Example of rosserial: pubsub

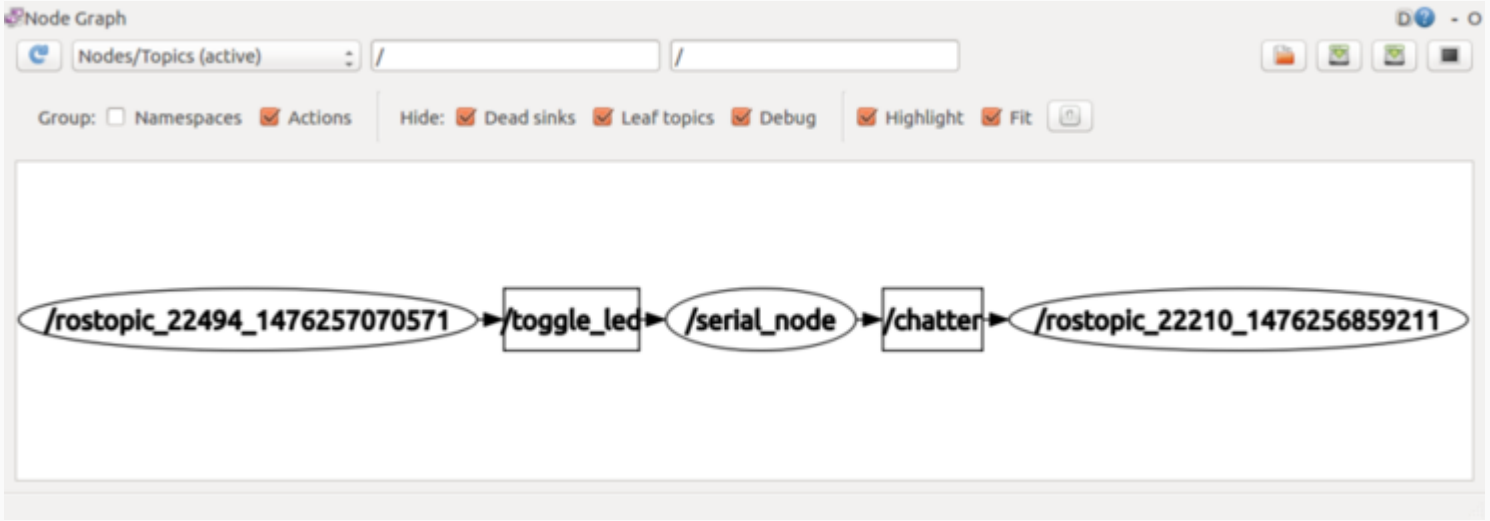
이제는 컴퓨터에서 std_msgs/Empty 타입의 "toggle_led"라는 토픽으로 OpenCR 보드에게 led 토글 신호를 퍼블리시하고 OpenCR 보드에서는 이를 서브스크라이브하여 led를 on/off 되는지 확인해 보자. 다음의 명령어를 보내게 되면 OpenCR 보드는 led가 제어되는 것을 볼 수 있다. 한번 명령어를 보내면 led 는 켜졌다가 다시 한번 명령어를 보내면 꺼진다. 말 그대로 토글(toggle) 작업을 하게 된다.

```
$ rostopic pub -1 /toggle_led std_msgs/Empty "{}"
publishing and latching message for 3.0 seconds

$ rostopic pub -1 /toggle_led std_msgs/Empty "{}"
publishing and latching message for 3.0 seconds
```

Example of rosserial: pubsub

이 퍼블리시와 서브스크라이브 메시지 통신을 rqt_graph로 확인하면 다음과 같다.

```
$ rqt_graph
```



Example of rosserial: pubsub

이러한 퍼블리시와 서브스크라이브 관련 프로그램은 일반적인 ROS 프로그래밍과 같은 방법으로 작성된다. 다만 다른 것은 그 대상이 OpenCR 라는 마이크로컨트롤러 보드라는 것 뿐이다. 일단, 위에서 실행한 예제 프로그램(pubsub)을 살펴보자.

```
#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Empty.h>

ros::NodeHandle  nh;

void messageCb( const std_msgs::Empty& toggle_msg) {
  digitalWrite(13, HIGH-digitalRead(13));   // blink the led
}

ros::Subscriber<std_msgs::Empty> sub("toggle_led", messageCb );
std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg);

char hello[13] = "hello world!";
```

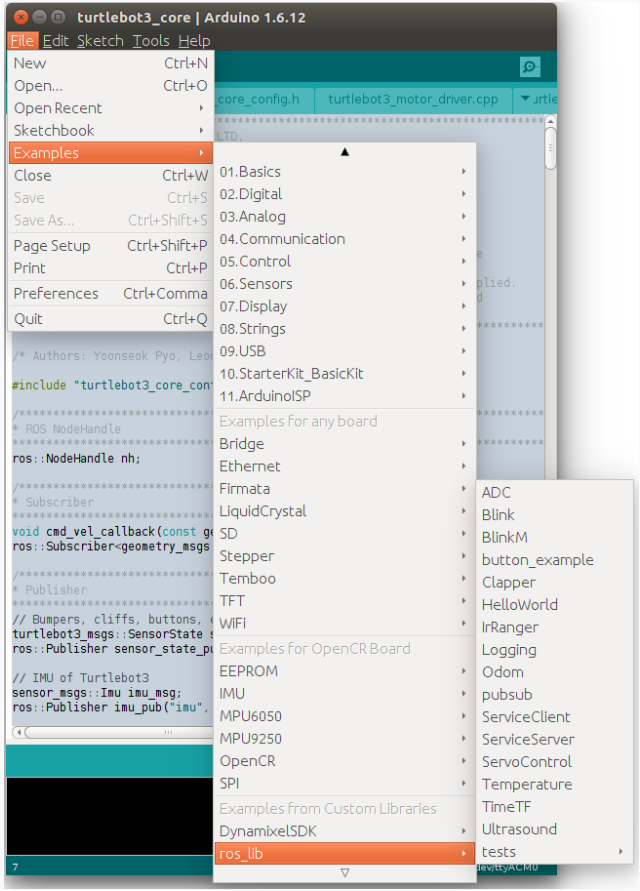
```
void setup() {
  pinMode(13, OUTPUT);
  nh.initNode();
  nh.advertise(chatter);
  nh.subscribe(sub);
}

void loop() {
  str_msg.data = hello;
  chatter.publish( &str_msg );
  nh.spinOnce();
  delay(500);
}
```

ros_libs

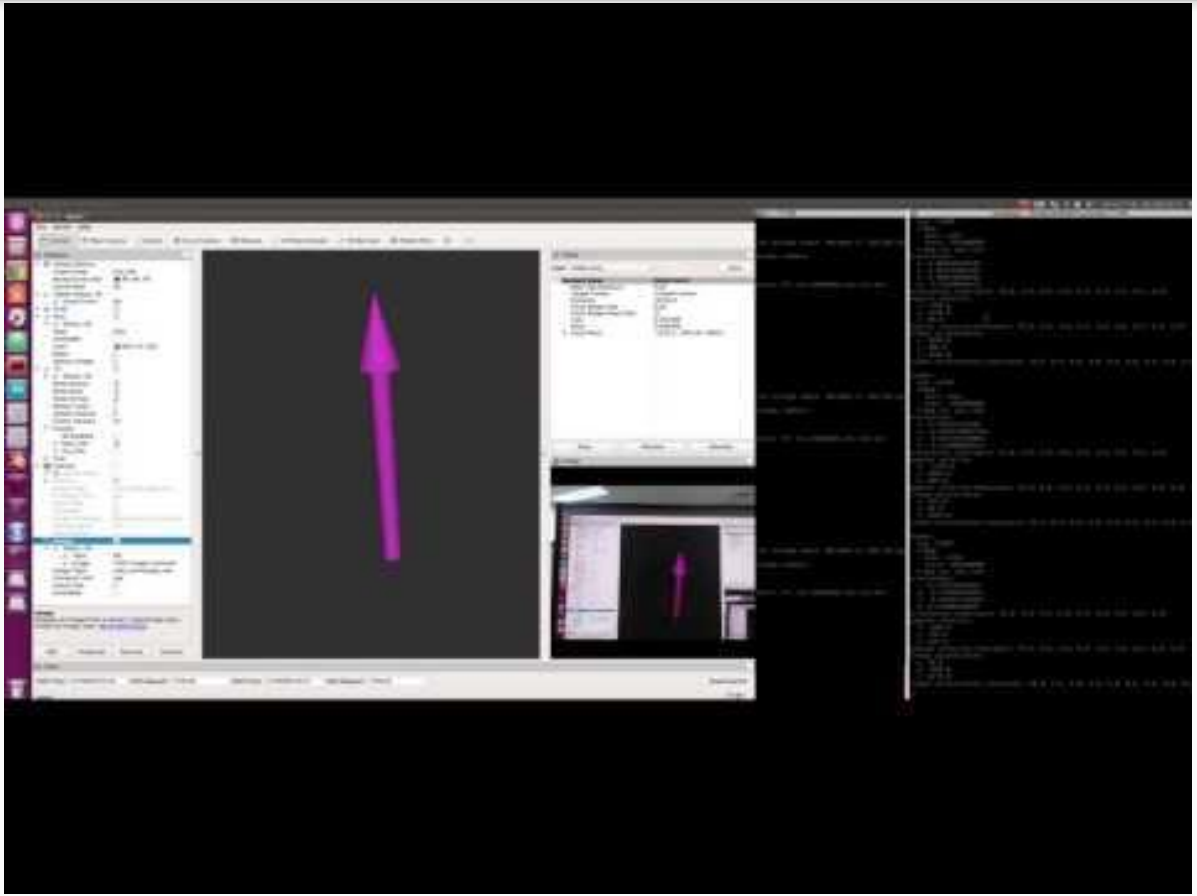
예제 프로그램(pubsub)이외에도

ADC, Blink, BlinkM, button_example,
Clapper, HelloWorld, IrRanger, Logging,
Odom, pubsub, ServiceClient, ServiceServer,
ServoControl, Temperature, TimeTF,
Ultrasound, tests

등이 준비 되어 있다.



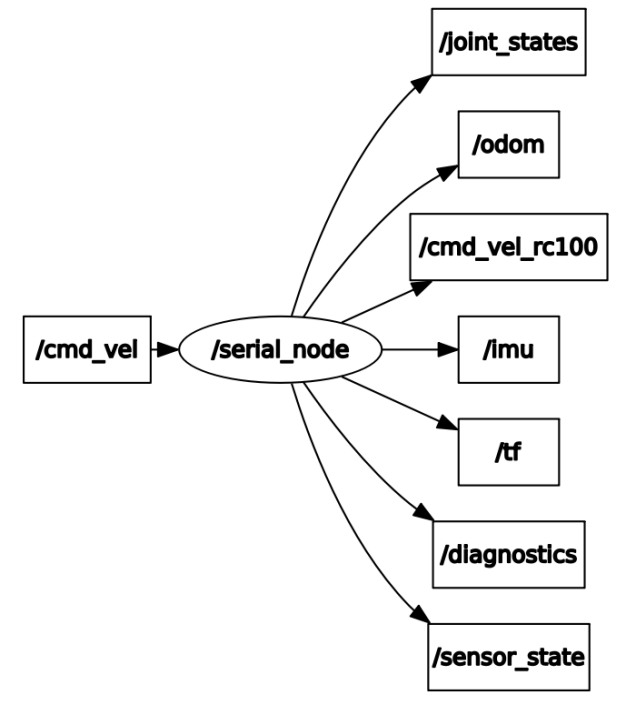
Practical examples: IMU on RViz

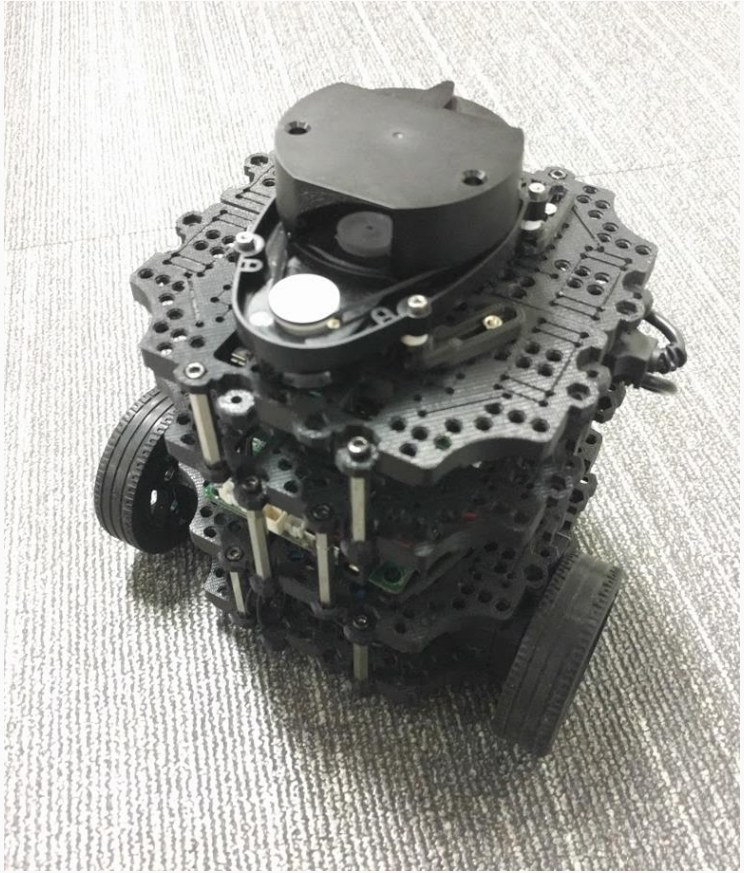


https://youtu.be/wXN_7oRHst0

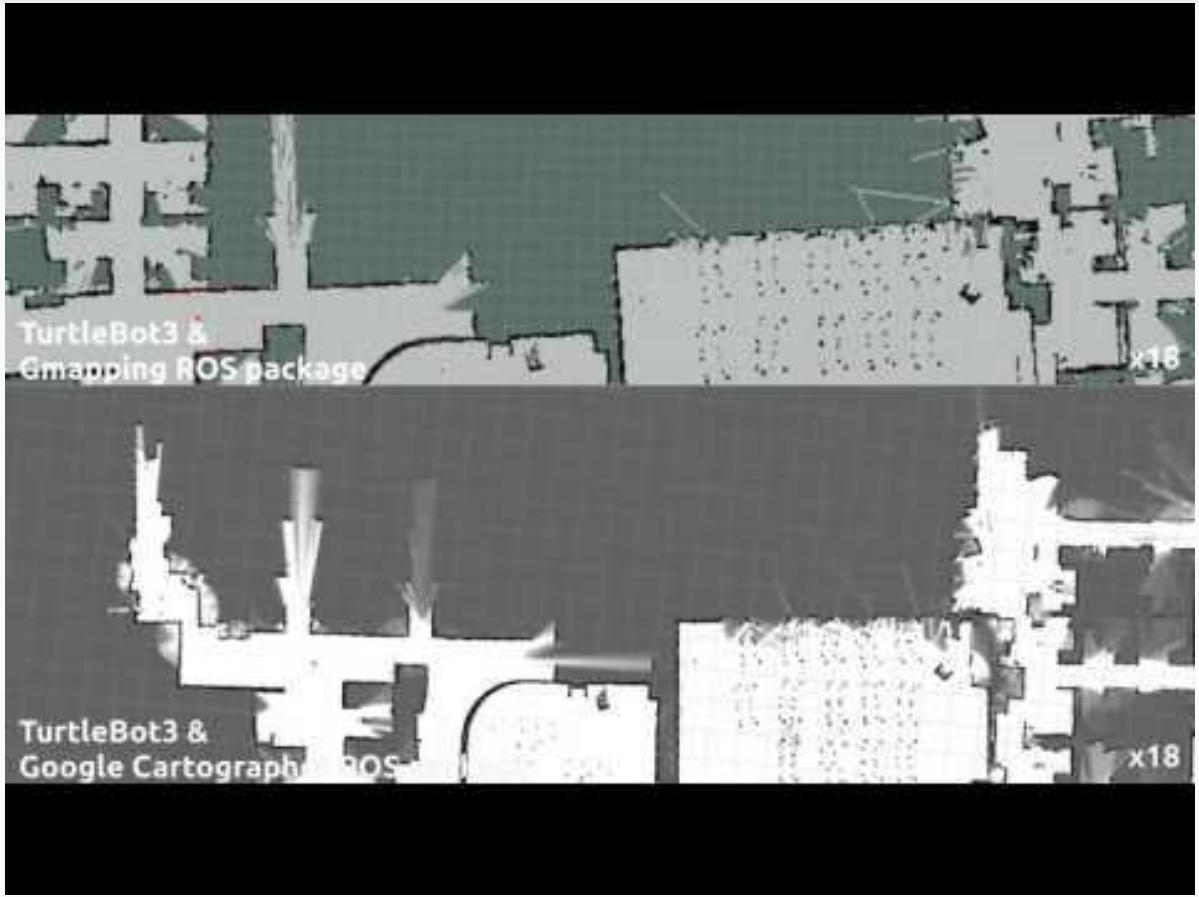
Practical examples: TurtleBot3 with OpenCR + rosserial

모바일 로봇에 사용할 때의 예제

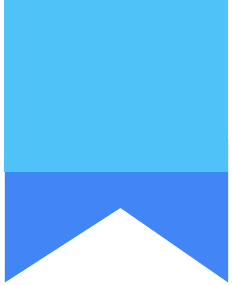




Practical examples: TurtleBot3 with OpenCR + rosserial



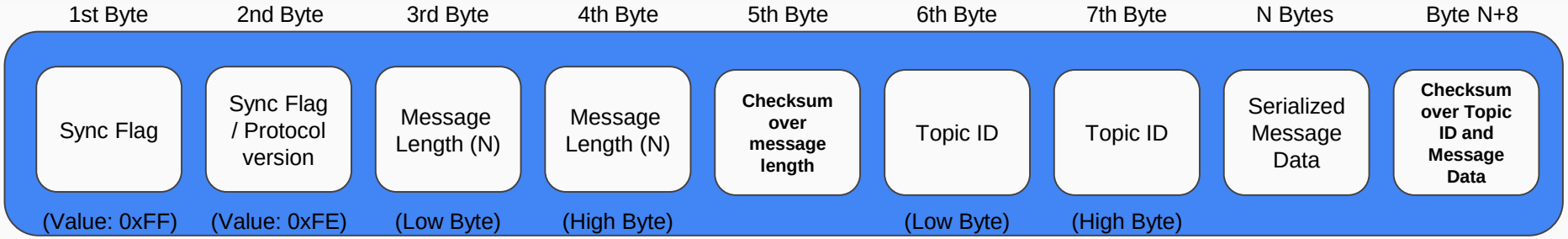
https://youtu.be/lkW4-dG2BCY



Let's get to know more about ROSSERIAL!

- Protocol
- Limitations
- Future

Protocol of rosserial (http://wiki.ros.org/rosserial/Overview/Protocol)



Sync Flag	**oxFF**
Sync Flag / Protocol version	**0xFF** on ROS Groovy, **0xFE** on ROS Hydro, Indigo, Jade and Kinetic
Message Length (N)	Message Length, 2 Byte = Low Byte + High Byte
Checksum over message length	Checksum = 255 - ((Message Length Low Byte + Message Length High Byte) %256)
Topic ID	2 Byte = Low Byte + High Byte ID_PUBLISHER=0, ID_SUBSCRIBER=1, ID_SERVICE_SERVER=2, ID_SERVICE_CLIENT=4, ID_PARAMETER_REQUEST=6, ID_LOG=7, ID_TIME=10, ID_TX_STOP=11
Serialized Message Data	Message Data like IMU, TF, GPIO
Checksum over Topic ID and Message Data	Checksum = 255 - ((Topic ID Low Byte + Topic ID High Byte + data byte values) % 256)

Limitations of rosserial (http://wiki.ros.org/rosserial/Overview/Limitations)

1. Maximum Size of a Message, Maximum Number of Publishers/Subscribers

ros::NodeHandle_<HardwareType, MAX_PUBLISHERS=25, MAX_SUBSCRIBERS=25, IN_BUFFER_SIZE=512, OUT_BUFFER_SIZE=512> nh;

: 실제 노드핸들의 선언에서 퍼블리셔의 갯수, 서브스크라이버의 갯수, 수신버퍼, 송신버퍼를 설정하게 되는데 이는 사용하는 MCU에 의존하게 된다.

2. Float64

: 아두이노의 한계로 64-bit float은 32-bit 데이타형으로 변환된다.

Limitations of rosserial (http://wiki.ros.org/rosserial/Overview/Limitations)

3. Strings

: 메모리 사용 문제로 String 대신에 "char *"를 사용한다.

4. Arrays

: 메모리 사용 문제로 길이를 지정하여 사용한다.

5. Speed and bandwidth

: serial의 하드웨어적인 제어로 인하여 전송 속도와 대역폭에 한계가 존재한다.

OpenCR's Challenge

- rosserial 에서의 한계 =8Bit 계열의 MCU / Serial (UART)

- OpenCR은 32Bit 계열의 MCU이기에 이를 하드웨어적 성능으로 극복할 수 있으며, serial이 아닌 USB Stack 및 Ethernet Stack 을 이용하면 시리얼 통신에만 묶여 있던 제약을 해결할 수 있을 것으로 예상된다.

- 이 숙제를 풀어내는 것이 OpenCR이 “Open-source Control module for ROS”으로 거듭날 수 있는 길이라고 생각한다.

OpenCR 개발팀

@iMachine님, @BlueSky7님, @Baram님, @박차장님

수고하셨습니다.

Open-source **C**ontrol module for **R**OS