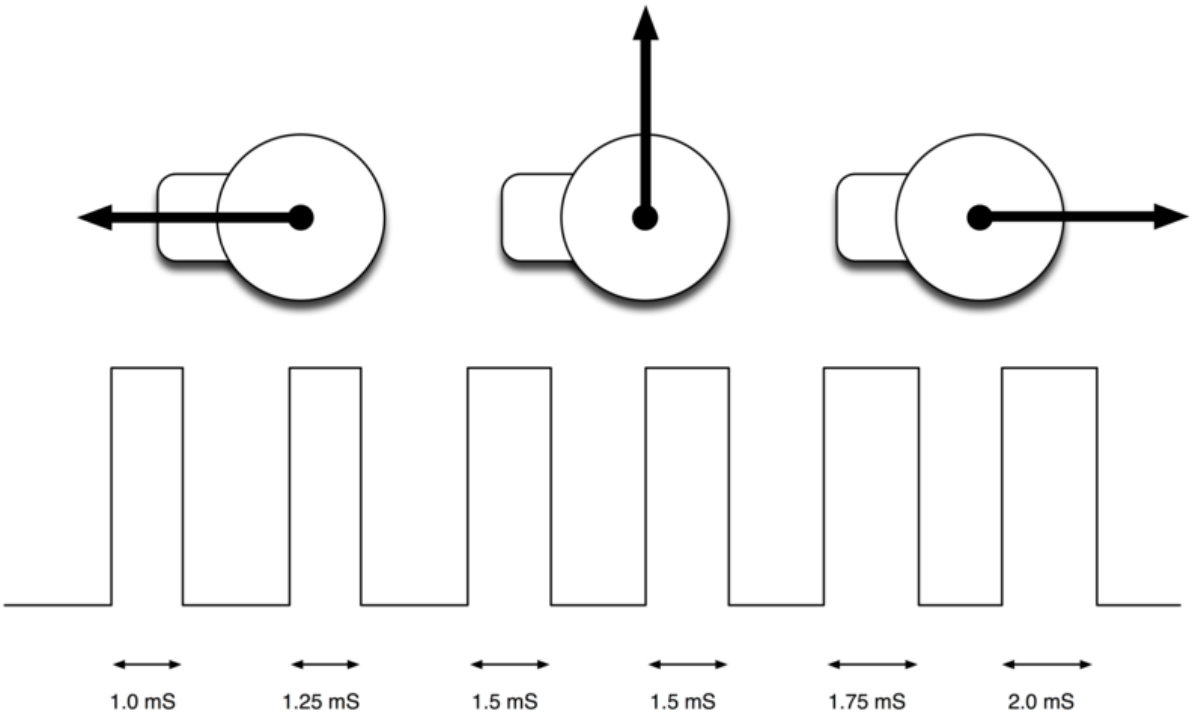
채명석

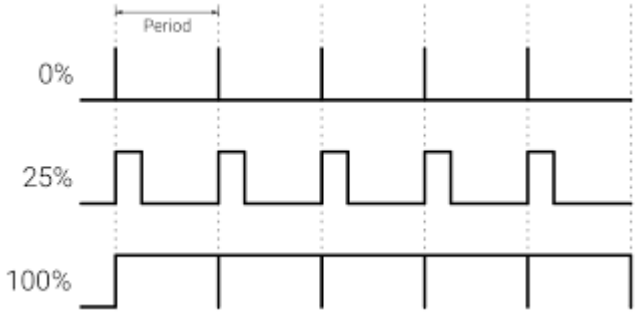
1. 아두이노 프로그래밍

・서보모터 제어(사용기술 PWM)

서보모터 - 서보 모터의 위치는 펄스의 길이에 따라서 설정됩니다. servo는 대략 매 20ms마다 펄스를 받게 되는데, 만약 이 펄스가 1ms동안 high이면 각은 0 이며, 1.5ms동안 high이면 중간위치에 위치하게 되고 2ms인 경우는 180도가 되게 됩니다.



▼ Pulse Width Modulation - 펄스 폭 변조라고 말하는 PWM은 펄스의 폭을 컨드롤하는 기제어방법이다. 출력되는 전압값을 일정한 비율(duty)동안 High를 유지하고, 나머지는 Low를 출력하여 아래와 같은 사각파의 출력을 만들어 낸다.

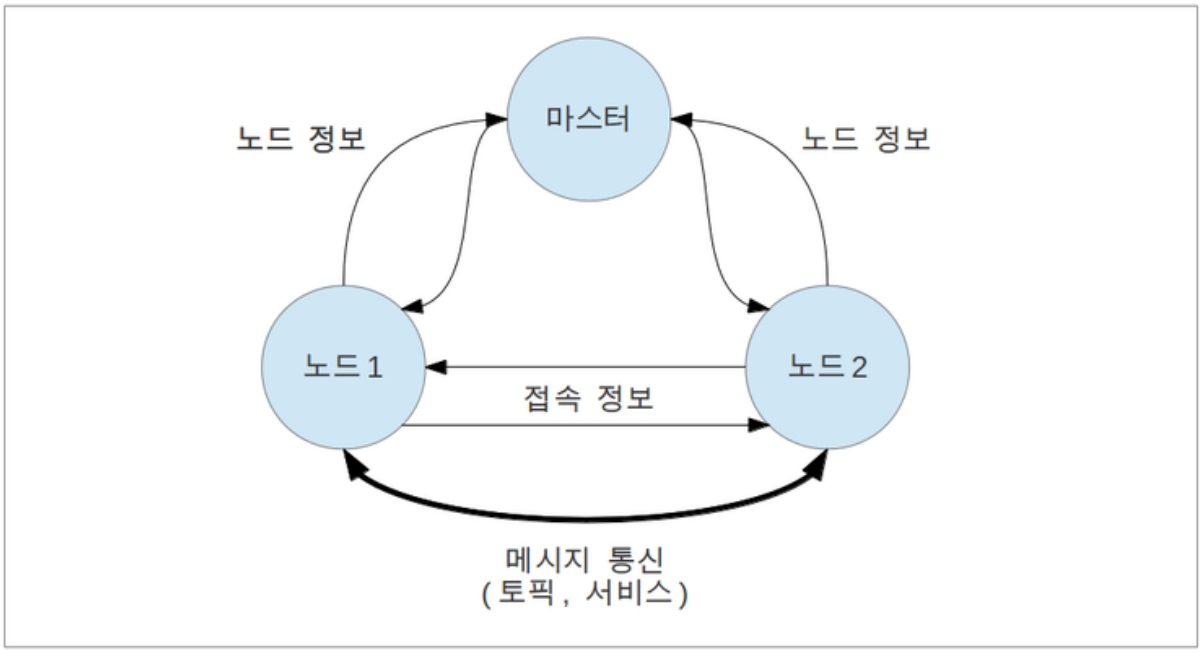


2. ROS 시리얼 통신

· ROS 개념 정리

1. 메시지 통신

- ROS의 핵심 개념은 **노드간의 메시지 통신**이다. 목적에 따라 세분화된 최소 단위의 실행 프로그램인 노드는 다른 노드와 처리 결과 값을 주고 받게 됨으로써 하나의 커다란 프로그램이 된다. 이를 위하여 메시지 통신에는 단 방향으로 연속적으로 메시지를 송수신하는 **토픽** 메시지 통신과 쌍 방향으로 요청과 응답 형태의 정해진 작업을 수행하게 하는 목적의 **서비스** 메시지 통신이 있다.
- 이러한 노드간의 메시지 통신에는 접속이 필요한데, 노드간의 접속을 돕는 것이 마스터이다. 마스터는 노드의 이름, 토픽 및 서비스의 이름, URI 주소와 포트, 매개변수 등의 네임 서버와 같은 역할을 하게된다. 즉, 노드는 생성과 동시에 마스터에 자신의 정보를 등록하게 되고, 다른 노드가 마스터를 통해 접속하려는 노드의 정보를 마스터로부터 취득한다. 그 후, 노드와 노드가 직접 접속하여 메시지 통신을 하는 것이다.
- 이를 그림으로 나태내면 다음의 그림과 같다. 마스터는 노드들의 정보를 관리하며, 각 노드는 필요에 따라 다른 노드와 접속, 메시지 통신을 하게 된다. 여기서 우리는 가장 중요한 **마스터, 노드, 토픽, 서비스, 메시지의 흐름**에 대해서 알아보도록 하자.



2. 마스터 구동

- 노드들간의 메시지 통신에서 연결 정보를 관리하는 마스터는 로스를 사용하기 위해서 제일 먼저 구동해야하는 필수 요소이다. 다음과 같이 "roscore"라는 실행 명령어로 로스 마스터는 구동되며, XMLRPC으로 서버를 구동하게 된다. 마스터는 노드간의 접속을 위하여 노드들의 이름, 토픽 및 서비스의 이름, 메시지 형태, URI 주소 및 포트를 등록받고, 요청이 있을 경우 이 정보를 다른 노드에게 알려주는 역할을 한다.

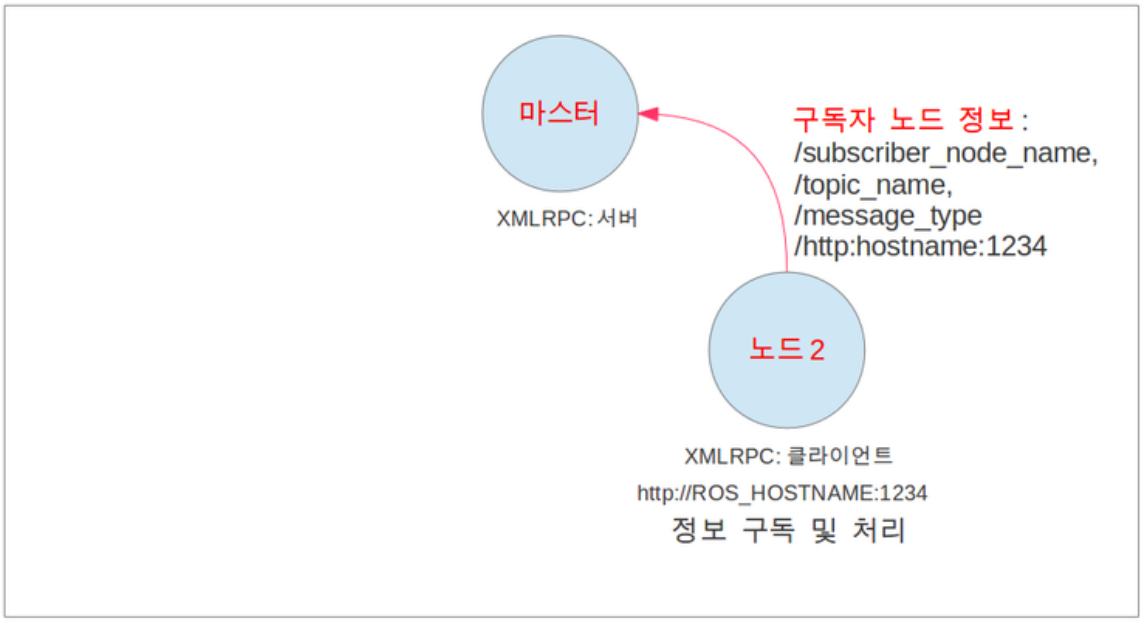
▼ roscore

3. 구독자 노드(Node) 구동

- 구독자 노드는 다음과 같이 "rosrun" 및 "roslaunch" 라는 실행 명령어로 구동된다. 구독자 노드는 구동과 함께 마스터에 자신의 구독자노드이름, 토픽이름, 메시지형태, URI 주소 및 포트를 등록한다. 마스터와 노드는 XMLRPC 를 이용하여 통신하게 된다.

▼ rosrun 패키지이름 노드이름

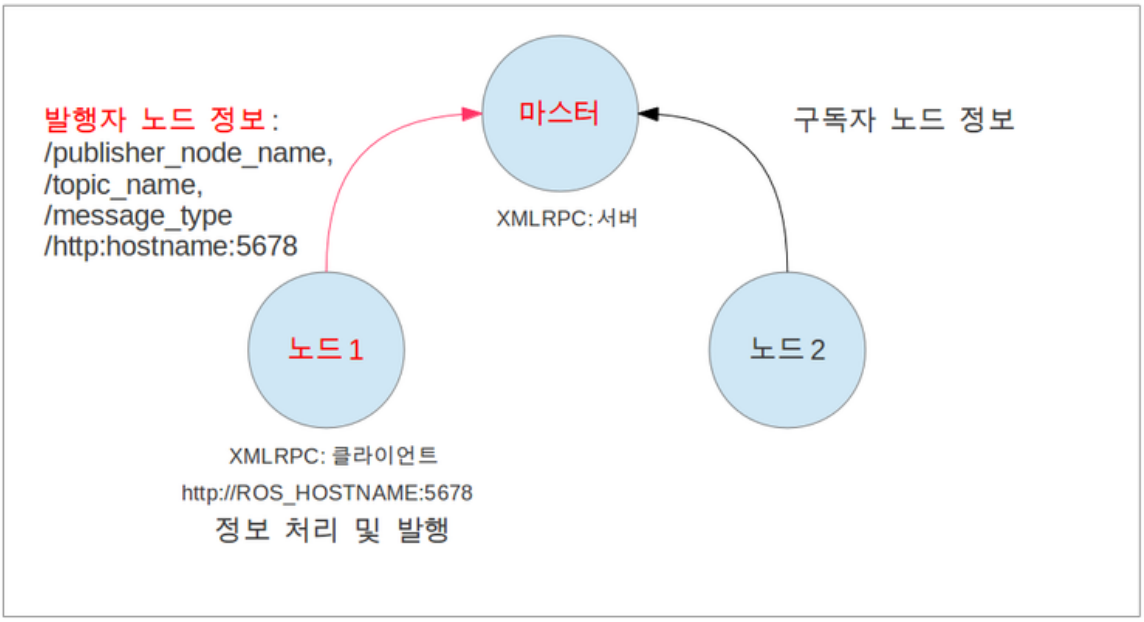
▼ roslaunch 패키지이름 런치이름



4. 발행자 노드(Node) 구동

- 발행자 노드는 구독자 노드와 마찬가지로 "rosrun" 및 "roslaunch" 라는 실행 명령어로 구동한다. 발행자 노드는 구동과 함께 마스터에 자신의 발행자노드이름, 토픽이름, 메시지형태, URI 주소 및 포트를 등록한다. 마스터와 노드는 XMLRPC 를 이용하여 통신하게 된다.

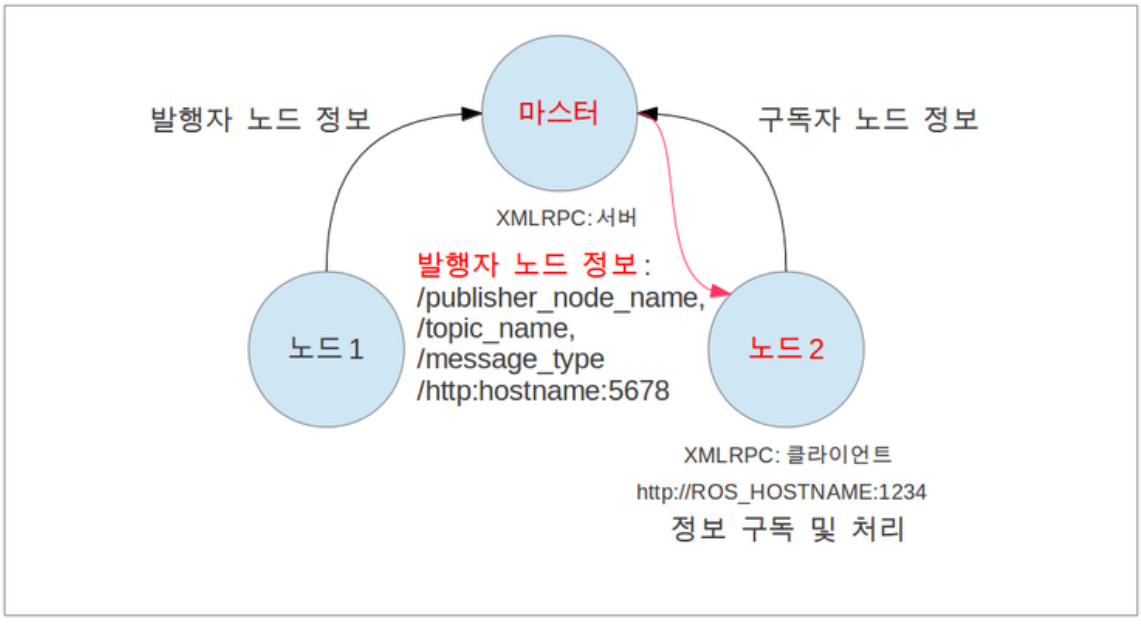
▼ 발행자 노드 정보



5. 발행자 정보 알림

- 마스터는 구독자 노드에게 새로운 발행자 정보를 알린다. 마스터와 노드는 XMLRPC를 이용하여 통신하게 된다.

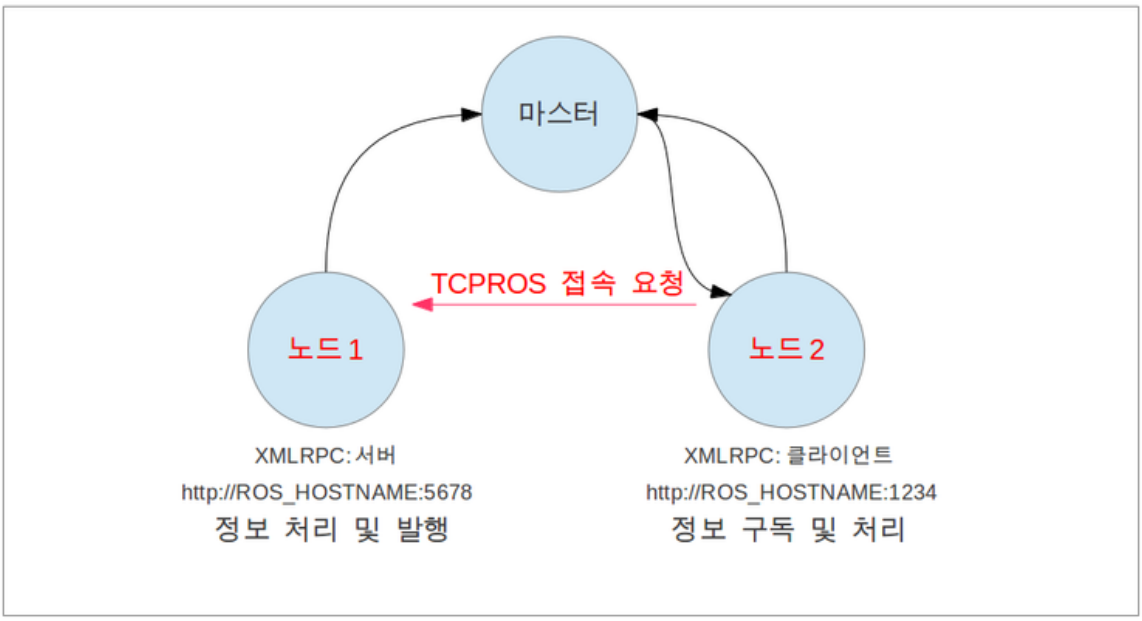
▼ 발행자 정보 알림



6. 발행자 노드에 접속 요청

- 구독자 노드는 마스터로부터 받은 발행자 정보를 기반으로 발행자 노드에게 직접 접속 요청을 한다. 이때에 전송하는 정보로는 자신의 구독자 노드 이름, 토픽이름, 메시지방식(TCPROS 또는 UDPROS)이 있다. 발행자 노드와 구독자 노드는 XMLRPC 를 이용하여 통신하게 된다.

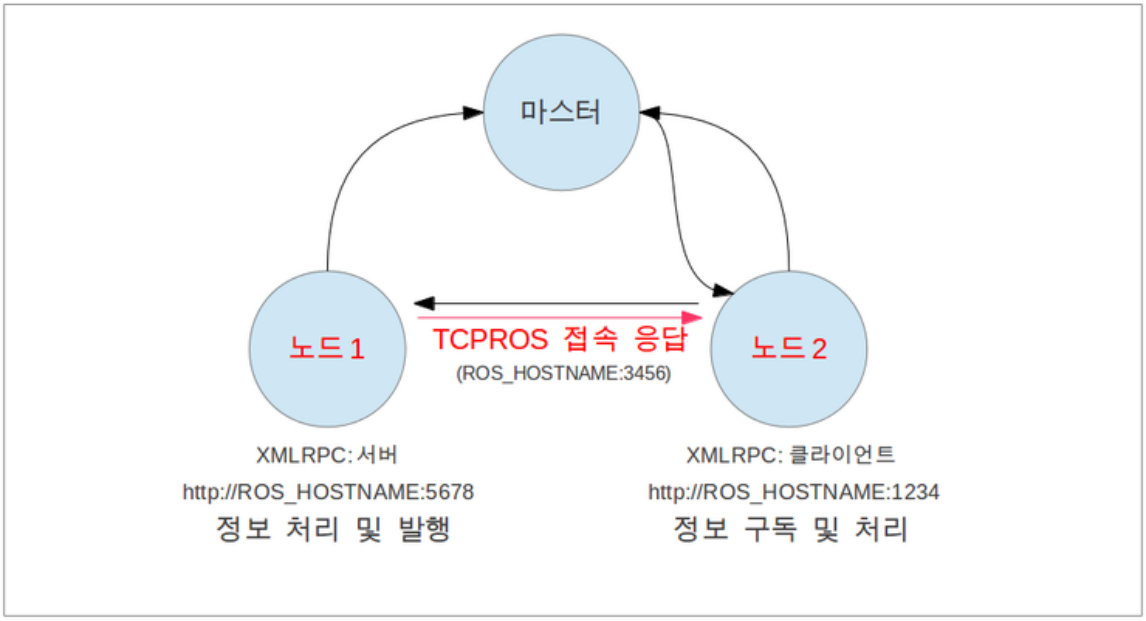
▼ 발행자 노드 접속 요청 과정



7. 발행자 노드에 접속 응답

- 발행자 노드는 구독자노드에게 접속 응답에 해당되는 자신의 TCP 서버의 정보인 URI주소와 포트를 전송한다. 발행자 노드와 구독자 노드는 XMLRPC 를 이용하여 통신하게 된다.

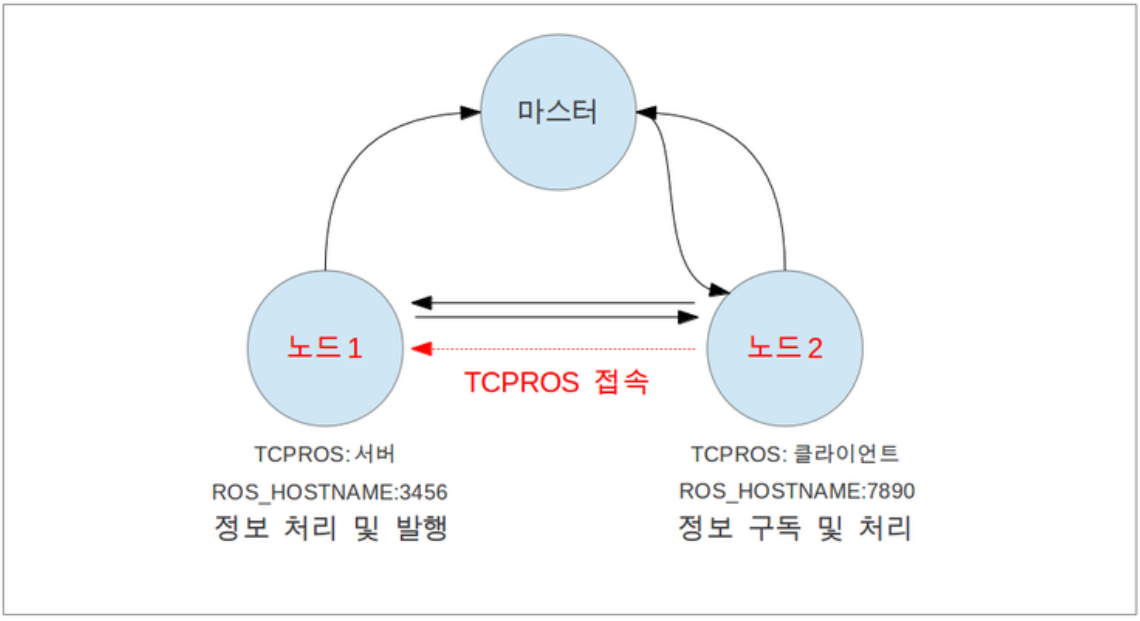
▼ 발행자 노드 접속 응답 과정



8. TCP 접속

- 구독자 노드는 TCPROS를 이용하여 발행자노드에 대한 클라이언트를 만들고, 발행자노드와 직접 연결한다. (노드간의 통신 방식으로는 TCPROS라 하는 TCP/IP 방식을 이용한다.)

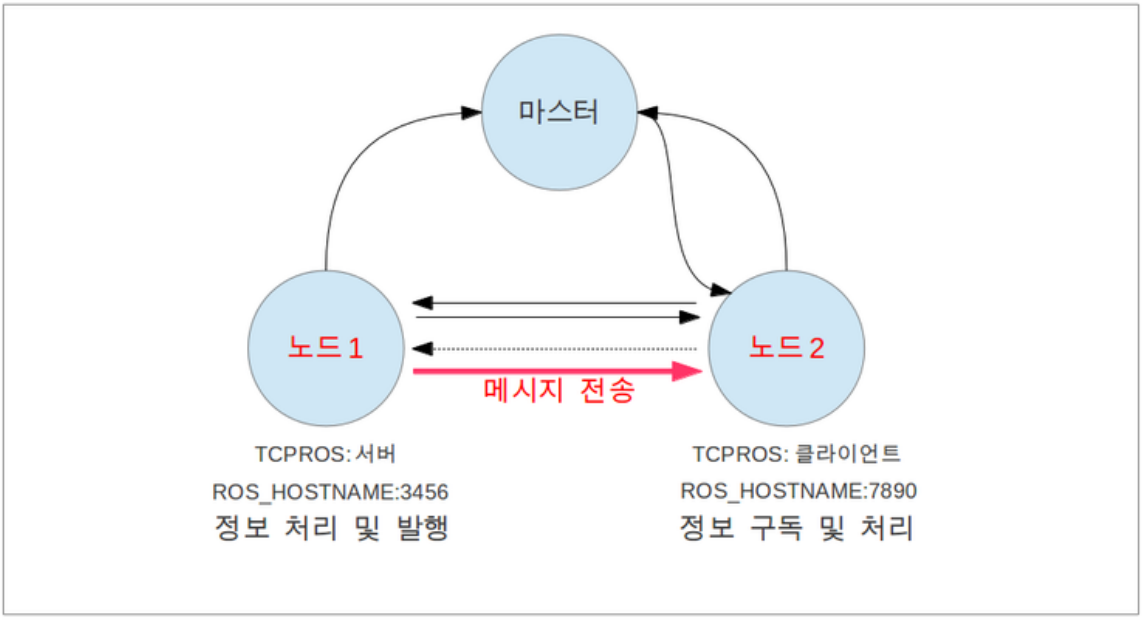
▼ TCP 접속



9. 메시지 전송

- 발행자 노드는 구독자 노드에게 정해진 메시지를 전송한다. (노드간의 통신 방식으로는 TCPROS라 하는 TCP/IP 방식을 이용한다.)

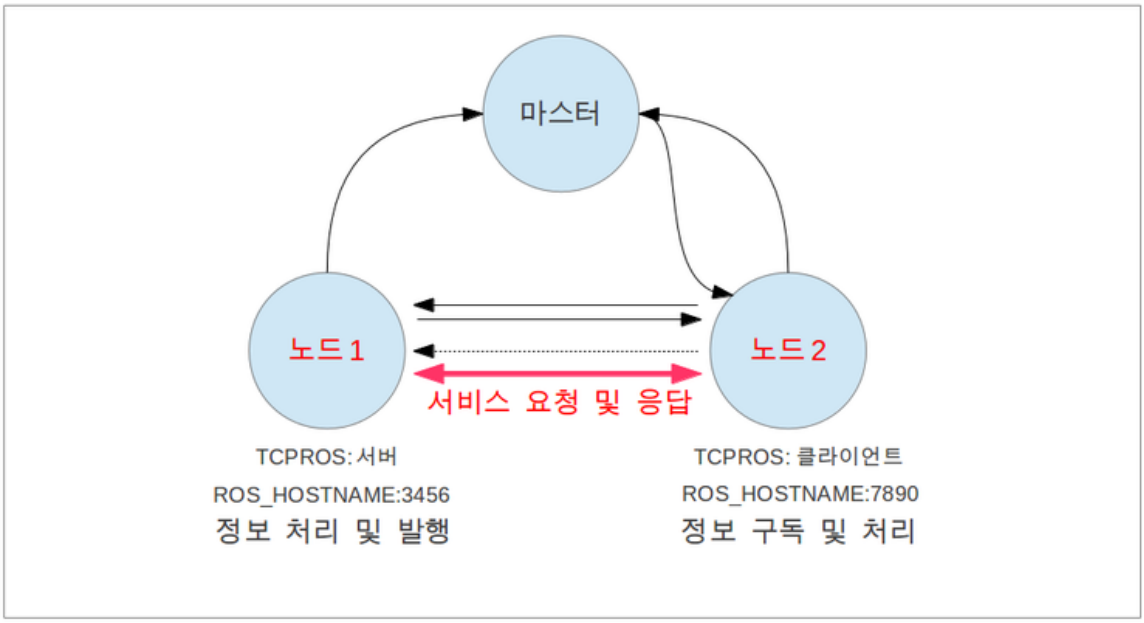
▼ 메시지 전송 과정



10. 서비스 요청 및 응답

- 위에서 설명한 내용은 메시지 통신중에 토픽에 해당된다. 토픽의 경우, 발행자 및 구독자가 중지하지 않는 이상, 메시지가 연속적으로 발행되고 구독하게 된다. 서비스의 경우에는 서비스를 요청하는 서비스 클라이언트와 서비스 요청을 받고 정해진 프로세스를 수행 및 응답하는 서비스 서버로 구분된다. 서비스는 토픽과 달리 1회에 한해 접속, 서비스 요청, 서비스 응답이 수행되고 서로간의 접속을 끊는다. 다시 필요한 경우 접속부터 다시 진행해야 한다.

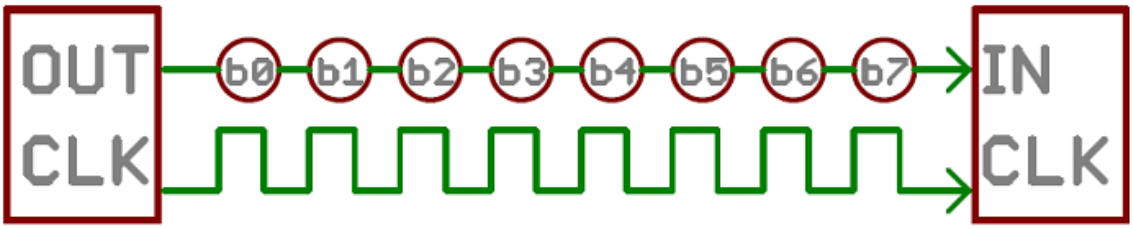
▼ 서비스 요청 및 응답



Searial 통신

- ROS는 로봇 응용프로그램을 개발할 때 필요한 하드웨어 추상화, 하위 디바이스 제어, 센싱, 인식, 슬램, 네비게이션 등의 기능 구현 및 메시지 전달, 패키지 관리, 개발 환경에 필요한 라이브러리와 다양한 개발 및 디버깅 도구를 제공하는 오픈 소스 기반의 로봇 메타 OS이다.

▼ 시리얼 통신(Serial Communication) : 임베디드 시스템은 각종 프로세서와 회로들 간에 서로 통신하며 동작하는 것이 핵심이기 때문에 데이터를 주고 받는 표준 프로토콜들이 필요한 것은 당연합니다. 수많은 프로토콜들이 있지만 일반적으로, 크게 두 개의 카테고리로 나눌 수 있습니다. 바로 패러럴(병렬, parallel)과 시리얼(직렬, serial) 입니다.



동기식 시리얼 인터페이스는 데이터 라인(data line)의 동작을 클럭 시그널(clock signal)라인의 동작과 동기화 시킵니다. 그리고 시리얼 버스에 함께 연결된 장치들이 이 클럭을 공유합니다. 이 방식은 직관적이고 종종 보다 빠르게 동작하지만 하나의 라인을 더 필요로 합니다. 대표적으로 SPI, I2C 프로토콜이 이런 방식을 사용합니다.

비동기식은 데이터가 외부 클럭 시그널(external clock signal)의 도움없이 동작하는 것을 의미합니다. 대신 데이터를 안정적으로 전송, 수신할 수 있도록 처리를 해줘야 합니다. 이 문서에서 다루는 시리얼(Serial, UART) 통신은 비동기식 시리얼 통신에 초점이 맞

춰져 있다. 일반적으로 시리얼 통신이라 부르는 것들이 대부분이 비동기식 시리얼 통신을 말함. GPS, 블루투스, XBee, Serial LCD 등이 비동기식 시리얼 통신을 사용합니다. 아두이노에서 말하는 시리얼, UART통신이라 부르는것도 동기식 시리얼 통신이라 생각하면 편함

▼ 시리얼 동작 방식 : 비동기식 시리얼 프로토콜은 몇 가지 규칙에 기반해서 동작합니다.

- Data bits
- Synchronization bits
- Parity bits
- And Baud rate

- Baud Rate (통신 속도)

Baud rate 는 시리얼 라인으로 전송되는 데이터 속도를 말합니다. Bits-per-second (bps) 단위로 표시하는데 이 값을 바탕으로 1 bit가 전송되는데 필요한 시간을 알 수 있습니다. 즉 1bit 데이터를 전송할 때 시리얼 라인을 high/low 상태로 유지하는 시간이며, 데이터를 받기 위해 시리얼 라인을 체크하는 시간입니다.

Baud rates 값은 어떻게든 설정할 수 있지만 통신 속도에 크게 영향을 받지 않는 경우 일반적으로 9600 bps를 사용합니다. 다른 표준 baud are 값으로 1200, 2400, 4800, 19200, 38400, 57600, 115200 을 사용할 수도 있습니다.

값이 높을수록 전송/수신 속도가 빠르지만 115200 를 초과할 수는 없습니다. 많은 마이크로 컨트롤러에서 이 값이 상한선으로 사용됩니다. Baud rate 값이 너무 높거나 양쪽의 설정값이 틀릴 경우 데이터 수신에 문제가 발생합니다.

- Framing the data (데이터 구조)

▼ 전송에 사용되는 데이터 패킷은 아래와 같이 구성됩니다. 각각의 구성요소는 Start bit를 제외하고 가변적인 크기를 가질 수 있습니다.

- Data chunk (데이터 영역)

실제 전송할 데이터를 말합니다. 5~9 bit 를 사용할 수 있는데 8 bit가 기본이긴 하지만다른 사이즈를 가질수도 있습니다. 예로 7 bit 크기인 경우는 7-bit ASCII 문자를 보내는데 적합니다.

데이터 크기가 설정되면 데이터의 엔디안 처리에 대해서도 송신, 수신측에서 합의가 되어야 합니다

- Synchronization bits(동기화 비트)

Synchronization bits 는 2개 혹은 3개의 특수한 비트로 전송되는 데이터 패킷의 시작과 끝을 알립니다. 위 그림에서 start bit, stop bit가 해당됩니다. Start bit는 1 bit, stop bit는 1~2 bit 로 설정할 수 있습니다. (일반적으로 stop bit는 1bit를 사용)

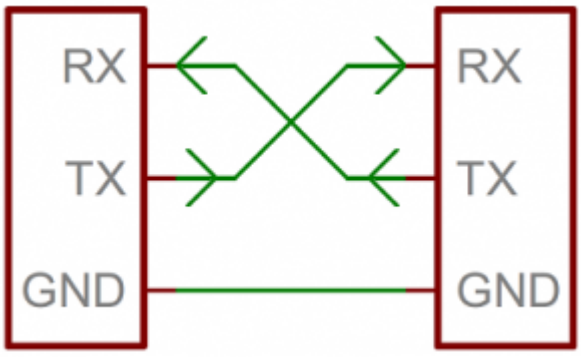
Start bit 는 idle 상태(데이터 전송이 없는 상태)에서 active 상태로의 변화(1–>0)로 표시되며 stop 비트는 반대로 idle 상태로 변화함(1)을 의미합니다.

- Parity bit(패리티 비트)

Parity(패리티) 비트는 매우 단순한 저레벨 에러 보정 방법으로 홀수 또는 짝수(odd or even)로 체크합니다. 데이터 영역에 해당하는 5-9 bit 를 모두 더해서 홀수, 단수인지를 패리티 비트(0 또는 1)에 기록하고 전송합니다. 수신측에서도 마찬가지의 작업으로 패리티 비트와 비교해서 수신 데이터에 문제가 있는지 체크합니다. 예로 **0b01011101** 데이터를 전송하면 5개의 1이 있으므로 홀수이고 패리티 비트는 1로 설정됩니다.

패리티 비트는 선택사항(optional)이며 잘 사용되지 않습니다. 노이즈에 취약한 환경에서 써볼만 하지만 송신, 수신측에서 추가적으로 연산에 대한 부담을 해야합니다. 그리고 문제 발생시 데이터를 다시 받을 수 있도록 처리해줘야 합니다.

▼ 연결 방법 하드웨어



RX, TX 라는 이름은 각 장치 자신의 입장에서 바라봤을 때 라인이 담당하는 역할입니다. 따라서 두 장치를 연결할 때는 TX-RX, RX-TX로 엇갈리게 연결되어야 합니다.

▼ ※ 참고로 공부하면 좋을 것

- How to read a schematic : https://learn.sparkfun.com/tutorials/how-to-read-a-schematic
- Analog vs Digital : https://learn.sparkfun.com/tutorials/analog-vs-digital
- Logic-Levels : https://learn.sparkfun.com/tutorials/logic-levels
- Binary : https://learn.sparkfun.com/tutorials/binary
- Hexadecimal : https://learn.sparkfun.com/tutorials/hexadecimal
- ASCII : https://en.wikipedia.org/wiki/ASCII

3. ROS1 젯슨나노 활용

4. ROS SLAM/NAVI

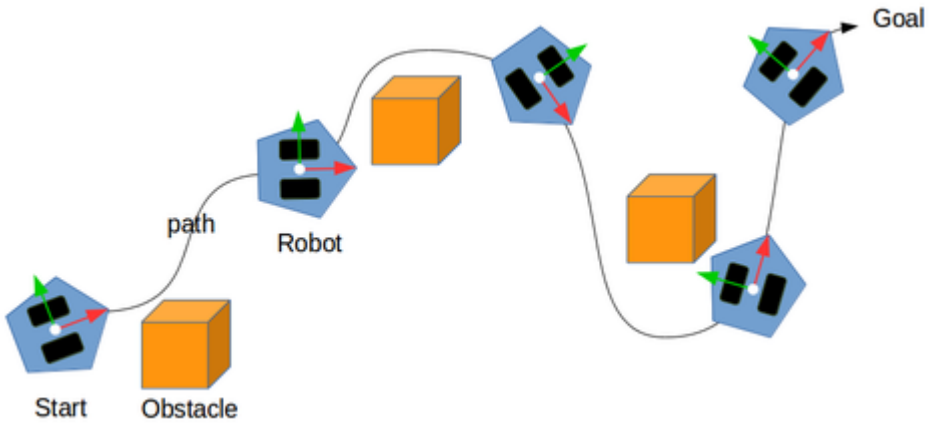
· 내비게이션(navigation)

내비게이션(navigation), 우리 말로는 "**길도움이**" 이라고 하는 건데 이 단어 들어보지 못한 현대인은 없을 듯 싶다. 바로, 우리 생황에 친숙하게 사용하고 있는 자동차에 달린 내비게이션으로 줄여서 "**내비**"라고도 불리우고 있다. 아마 이는 내비게이션보다는 "**내비게이터(navigator)**"라고 보는 것이 좋을 듯 싶다. 아무튼, 이는 운전할 때, 주어진 단말기의 지도에 목적지를 설정해 주면 현재 위치에서 목적지까지의 정확한 거리, 소요시간 등이 확인 가능하고, 도중에 거쳐야 하는 장소 및 이용하는 도로 등을 세부적으로 설정까지할 수 있도 있다. 더불어 아름다운 목소리의 아가씨(아저씨)가 가본적도 없는 미지의 목적지까지의 우리의 길을 안내하게 된다.

· 땔래야 땔 수 없는 SLAM과 내비게이션(navigation)

이 모바일 로봇의 기본이자 꽃은 바로 내비게이션이라고 할 수 있겠다. 내비게이션은 로봇이 정해진 목적지까지 이동하는 것으로, 말이야 쉽지만 로봇 자기 자신이 어디에 있는지, 그리고 주어진 환경의 지도를 가지고 있어야 한다는 것, 다양한 경로중에 최적화된 경로는 어떤 것인지, 그리고 도중에 로봇에게 장애물이 되는 벽, 가구, 물체 등을 피하여 이동하는 등 어디 하나 쉬운 미션이 하나도 없다.

▼ 그림. 1



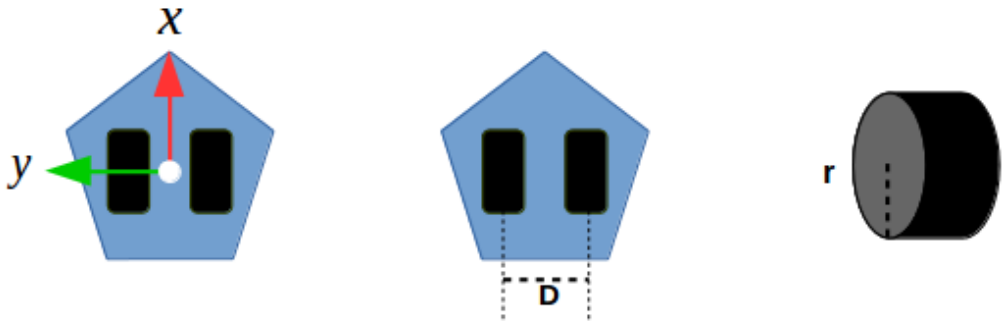
▼ 로봇에 있어서 내비게이션을 위해서는 어떠한 것 들이 필요할까? 내비게이션 알고리즘에 따라 틀리겠지만 필수적으로는 아래의 기능 정도는 갖추고 있어야 할 것이다.

▼ 지도

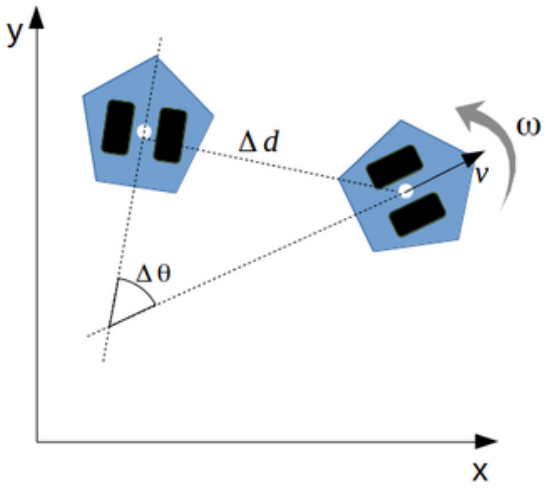
첫 번째는 지도이다. 내비게이터는 구매시 부터 매우 정확한 지도가 탑재되어 있고 이 지도를 기반으로 목적지까지 안내 받을 수 있게 된다. 그러나, 서비스 로봇이 사용되는 실내에서 지도가 따로 있을까? 내비게이터와 마찬가지로 지도가 필요함으로 사람이 직접 지도를 작성하여 로봇에게 주던지, 로봇 스스로가 지도를 작성하여야 할 것이다. 로봇이 스스로 혹은, 약간의 인간의 도움을 받아 지도를 작성하기 위해서 등장한 것이 바로**SLAM(Simultaneous localization and mapping)**이다. 우리 말로는 " **동시적 위치추정 및 지도 작성**" 이라고 표현할 수 있겠다. 이는 로봇이 미지의 임의 공간을 이동하면서 주변을 센싱하며 현재 위치를 추정하는과 동시에 그 과정에서 지도를 작성하는 방법이다.

▼ 로봇의 위치 계측/추정하는 기능

두 번째로 로봇 스스로 위치를 계측/추정할 수 있어야 한다. 자동차의 경우 GPS를 이용하여 자기 위치를 추정하지만, 실내에서는 GPS를 사용할 수 없고 만약에 사용할 수 있다고 하더라도 정밀한 이동을 위해서는 오차가 큰 GPS 는 쓸모가 없다. 그래서, 로봇은 일반적으로 자신이 움직이는 이동량을 바퀴의 회전축의 회전량을 가지고 측정하게 된다. 하지만, 바퀴의 회전량이라는게 추측 항법(dead reckoning, 데드레커닝)이라고 하여 오차가 꼬 발생한다. 그래서, IMU 센서 등으로 관성 정보를 취득하여 위치보상을 통해 그 오차를 줄여주는 방법이 사용되고는 한다.



추측 항법에 대해서 간략히 설명하겠다. 만약에, 위와 같이 로봇이 있을때, 바퀴간의 거리는 D, 바퀴의 반지름은 r 이라 하자. 그 후, 좌/우측 모터의 엔코더 값을 이용하여 수식(1)(2)(3)(4)로 변환 과정을 거치고 수식(5),(6)과 같이 로봇의 선속도(linear velocity: v)와 각속도(angular velocity: w)를 구할 수 있다. 그리고 최종적으로 Runge-Kutta 공식을 이용하여 식(7),(8), (9) 와 같이 이동한 위치, 각도의 근사값을 구할 수 있다.



$$V_l = \frac{E_{lc} - E_{lp}}{T_e} \frac{\pi}{180} \quad (rad/s) \tag{1}$$

$$V_r = \frac{E_{rc} - E_{rp}}{T_e} \frac{\pi}{180} \quad (rad/s) \tag{2}$$

$$V_l = v_l r \quad (m/s) \tag{3}$$

$$V_r = v_r r \quad (m/s) \tag{4}$$

$$v = \frac{V_r + V_l}{2} \quad (m/s) \tag{5}$$

$$\omega = \frac{V_r - V_l}{D} \quad (rad/s) \tag{6}$$

$$x_{k+1} = x_k + T_e v_k \cos\left(\theta_k + \frac{T_e \omega_k}{2}\right) \tag{7}$$

$$y_{k+1} = y_k + T_e v_k \sin\left(\theta_k + \frac{T_e \omega_k}{2}\right) \tag{8}$$

$$\theta_{k+1} = \theta_k + \omega_k T_e \tag{9}$$

E_{lc} E_{lp}: 좌측 모터의 엔코더 값 (현재와 이전값)
E_{rc} E_{rp}: 우측 모터의 엔코더 값 (현재와 이전값)
T_e : 경과시간
v_l v_r : 좌 / 우측 휠의 각속도
r : 휠의 반지름
V_l V_r : 좌 / 우측 휠의 선속도
v : 로봇의 선속도
ω : 로봇의 각속도

▼ 벽, 물체 등의 장애물의 계측하는 기능

세 번째로는 벽, 물체 등의 장애물을 파악하는 방법인데 이는 센서가 절대적으로 사용된다. 거리센서, 비전센서 등 다양한 종류의 센서가 사용되는데 거리센서는 LRF, 초음파센서, 적외선 거리센서 등이 사용되며 비전센서에는 스테레오 카메라, 모노카메라, 전방향 옴니 카메라, 그리고 최근에 Depth camera 로 많이 사용되는 Kinect, xtion 등이 장애물 파악하는데에 사용된다. 이 센서에 대한 내용은 대표적인 센서를 대상으로 이전 강좌로 설명하였다. 목차를 참고하여 보도록 하자.

▼ 목적지까지의 최적 경로를 계산하고 주행하는 기능

네 번째로는 목적지까지의 최적 경로를 계산하고 주행하는 기능인데, 이것이 바로 이 강좌 처음에 말한 **내비게이션(Navigation)** 이다. 이는 경로 탐색/계획 (path search and planning)이라고 하는데 A* 알고리즘, 전자자기장 알고리즘, 파티클, 그래프 방식 등 매우 다양하다

5. Open CV를 활용한 카메라 활용

Camera 출력

내장 카메라 또는 외장 카메라에서 이미지를 얻어와 프레임을 재생할 수 있습니다.

Main Code

```
import cv2

capture = cv2.VideoCapture(0)
capture.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

while True:
    ret, frame = capture.read()
    cv2.imshow("VideoFrame", frame)
    if cv2.waitKey(1) > 0: break

capture.release()
cv2.destroyAllWindows()
```

Detailded Code

```
capture = cv2.VideoCapture(0)
```

`cv2.VideoCapture(n)`을 이용하여 **내장 카메라** 또는 **외장 카메라**에서 영상을 받아옵니다.

`n`은 **카메라의 장치 번호**를 의미합니다. 노트북을 이용할 경우, 내장 카메라가 존재하므로 카메라의 장치 번호는 `0`이 됩니다.

카메라를 추가적으로 연결하여 **외장 카메라**를 사용하는 경우, 장치 번호가 `1~n`까지 변화합니다.

```
capture.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
```

`capture.set(option, n)`을 이용하여 카메라의 속성을 설정할 수 있습니다.

`option`은 **프레임의 너비와 높이**등의 속성을 설정할 수 있습니다.

`n`의 경우 해당 **너비와 높이의 값**을 의미합니

```
while True:
    ret, frame = capture.read()
    cv2.imshow("VideoFrame", frame)
    if cv2.waitKey(1) > 0: break
```

`while`문을 이용하여 **영상 출력을 반복합니다.**

`capture.read()`를 이용하여 `카메라의 상태` 및 `프레임`을 받아옵니다.

`ret`은 카메라의 상태가 저장되며 정상 작동할 경우 `True`를 반환합니다. 작동하지 않을 경우 `False`를 반환합니다.

`frame`에 현재 프레임이 저장됩니다.

`cv2.imshow("윈도우 창 제목", 이미지)`를 이용하여 **윈도우 창**에 **이미지**를 띄웁니다.

`if`문을 이용하여 **키 입력**이 있을 때 까지 `while`문을 반복합니다.

`cv2.waitkey(time)`이며 `time`마다 키 입력상태를 받아옵니다.

키가 입력될 경우, 해당 키의 `아스키 값`을 반환합니다.

즉, 어떤 키라도 누를 경우, `break`하여 `while`문을 종료합니다

- Tip : `time`이 `0`일 경우, 지속적으로 검사하여 **프레임이 넘어가지 않습니다.**

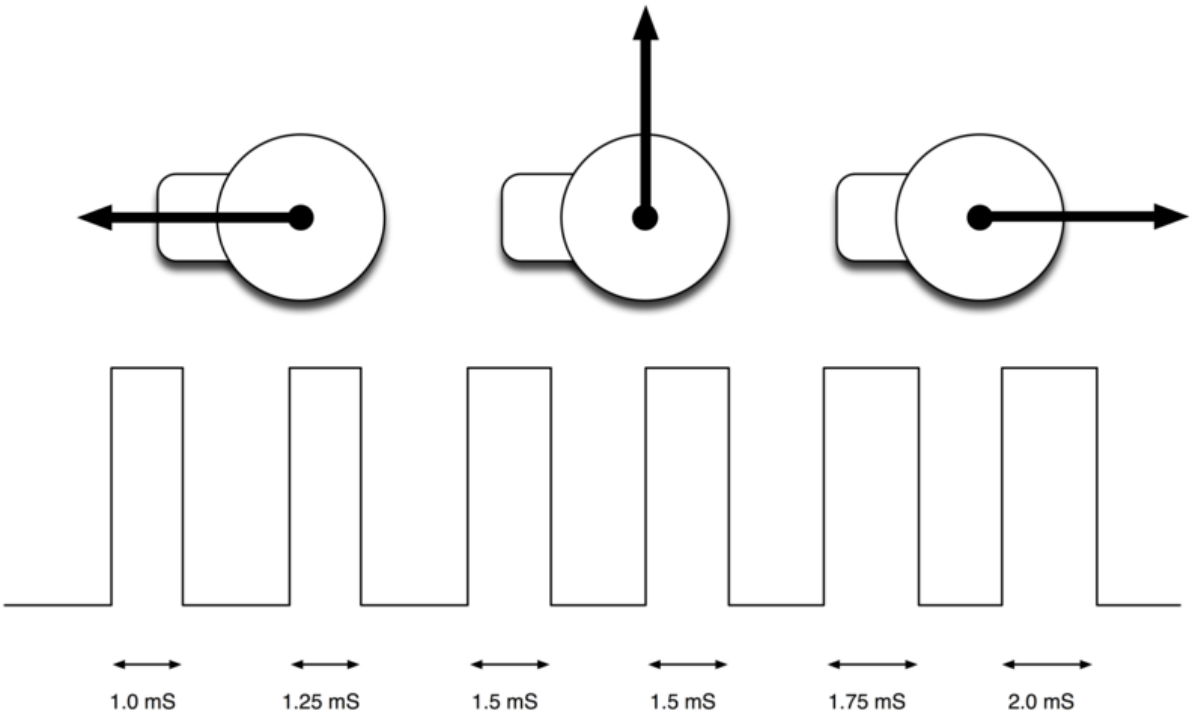
- Tip : `if cv2.waitKey(1) == ord('q'): break`으로 사용할 경우, `q`가 입력될 때 `while`문을 종료합니다.

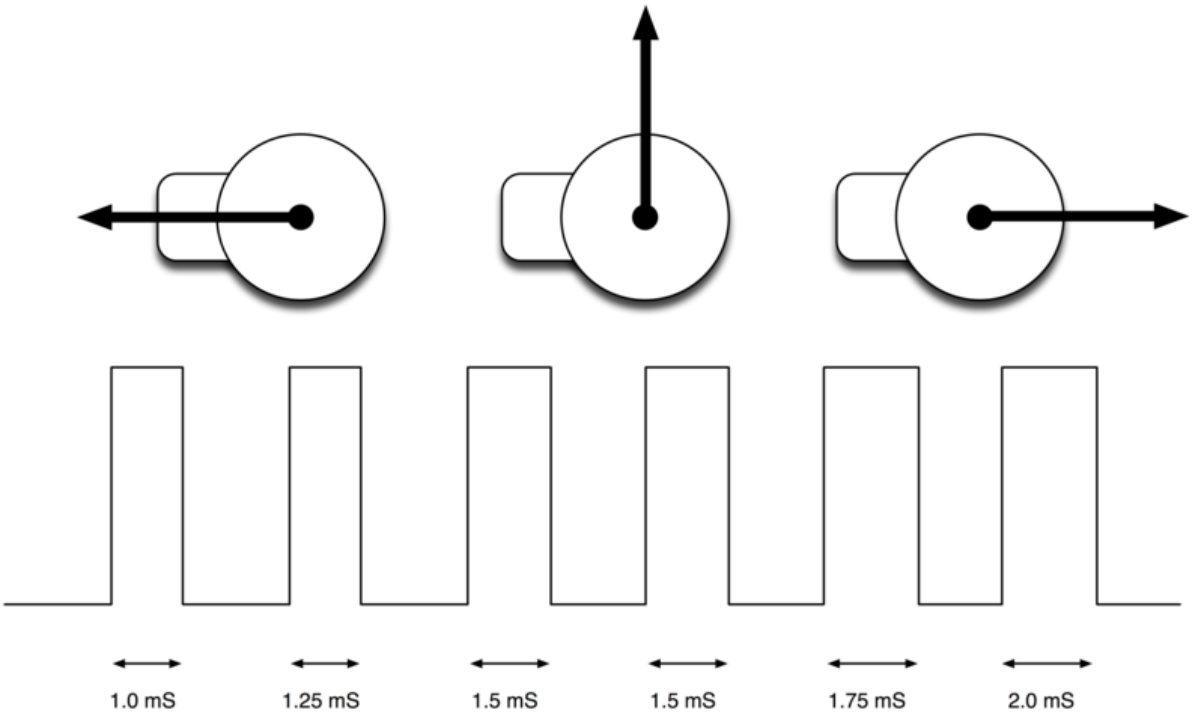
```
capture.release()
cv2.destroyAllWindows()
```

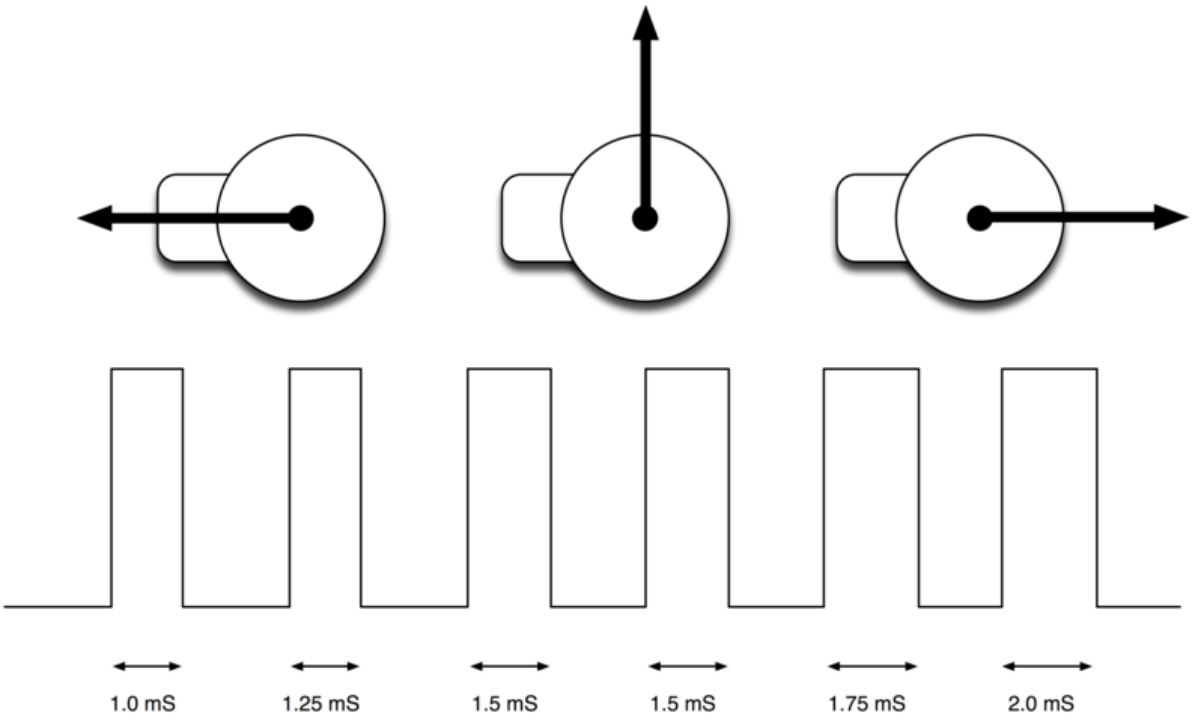
`capture.relase()`를 통하여 카메라 장치에서 받아온 **메모리를 해제합니다.**

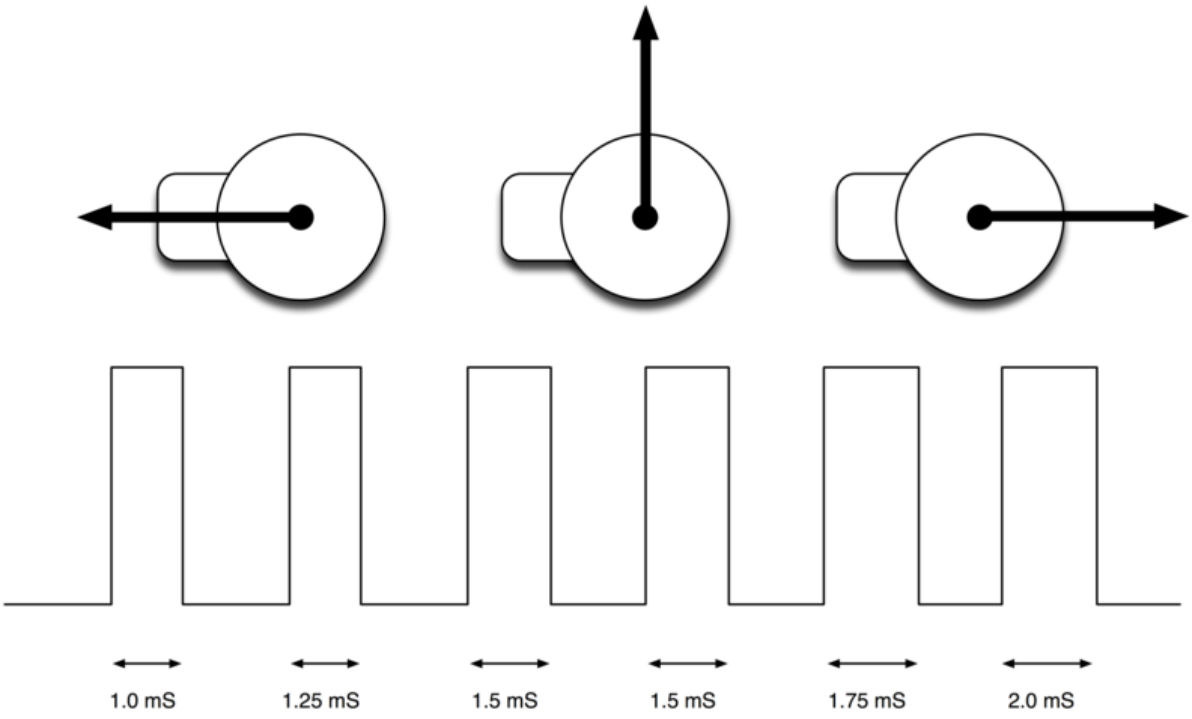
`cv2.destroyAllWindows()`를 이용하여 **모든 윈도우창을 닫습니다.**

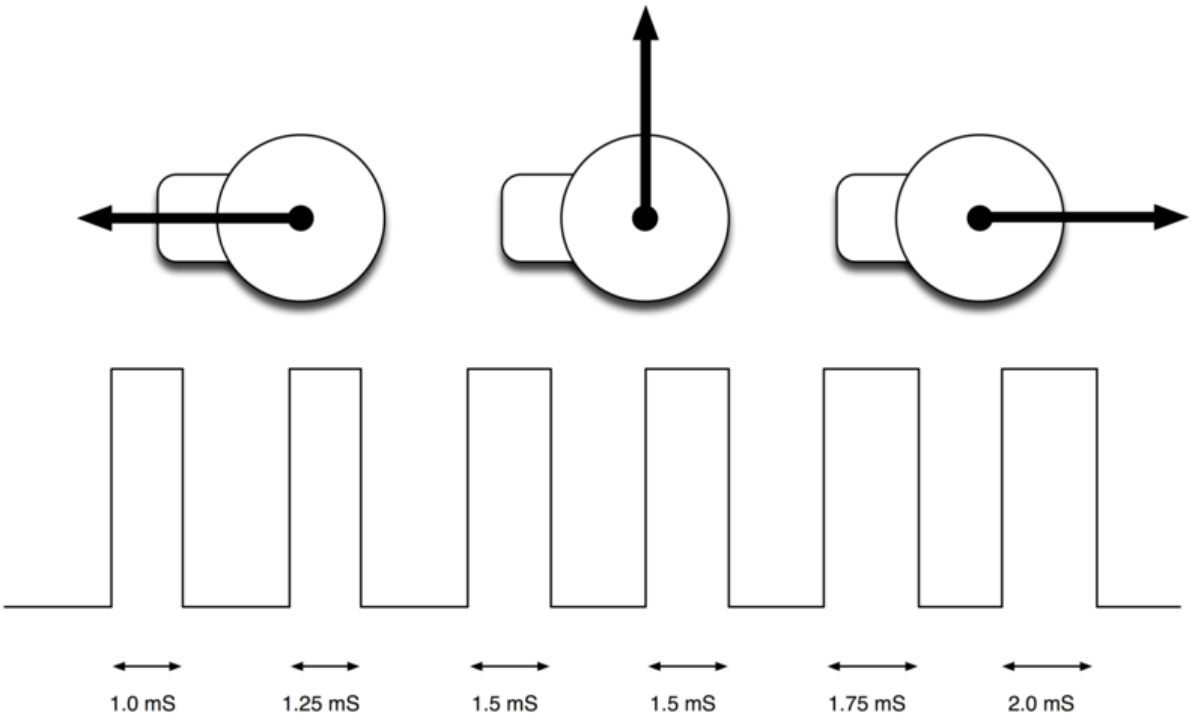
- Tip : `cv2.destroyWindow("윈도우 창 제목")`을 이용하여 **특정 윈도우 창만 닫을 수 있습니다.**

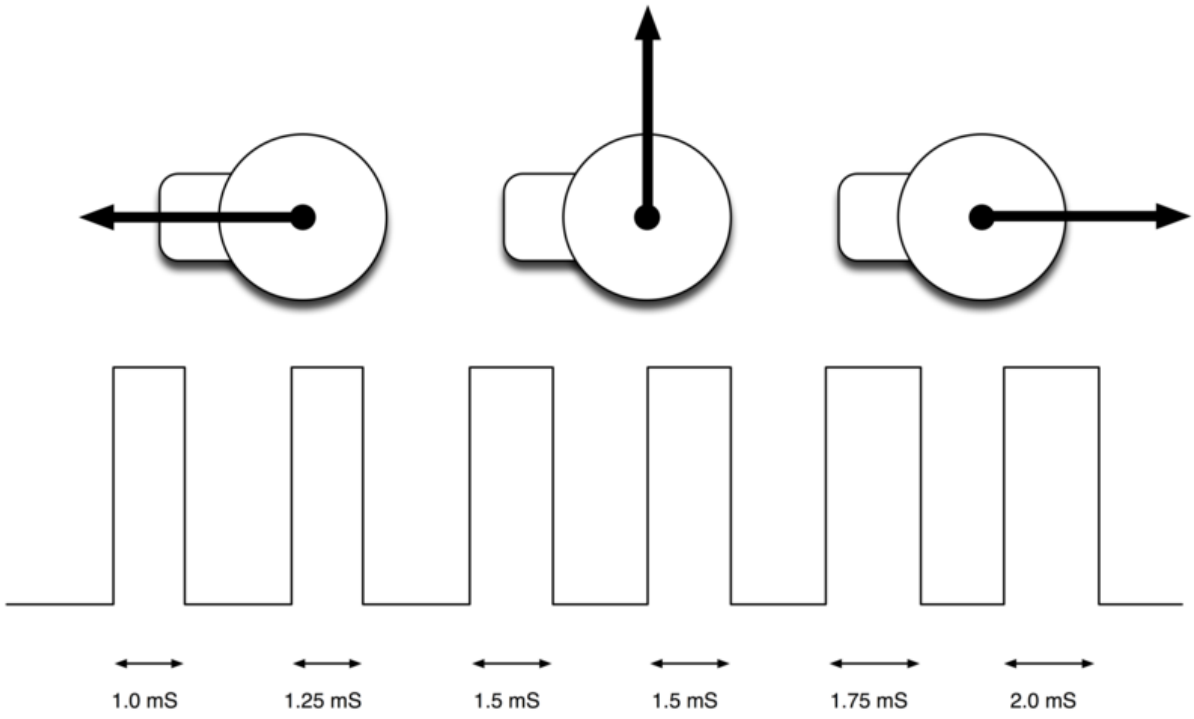


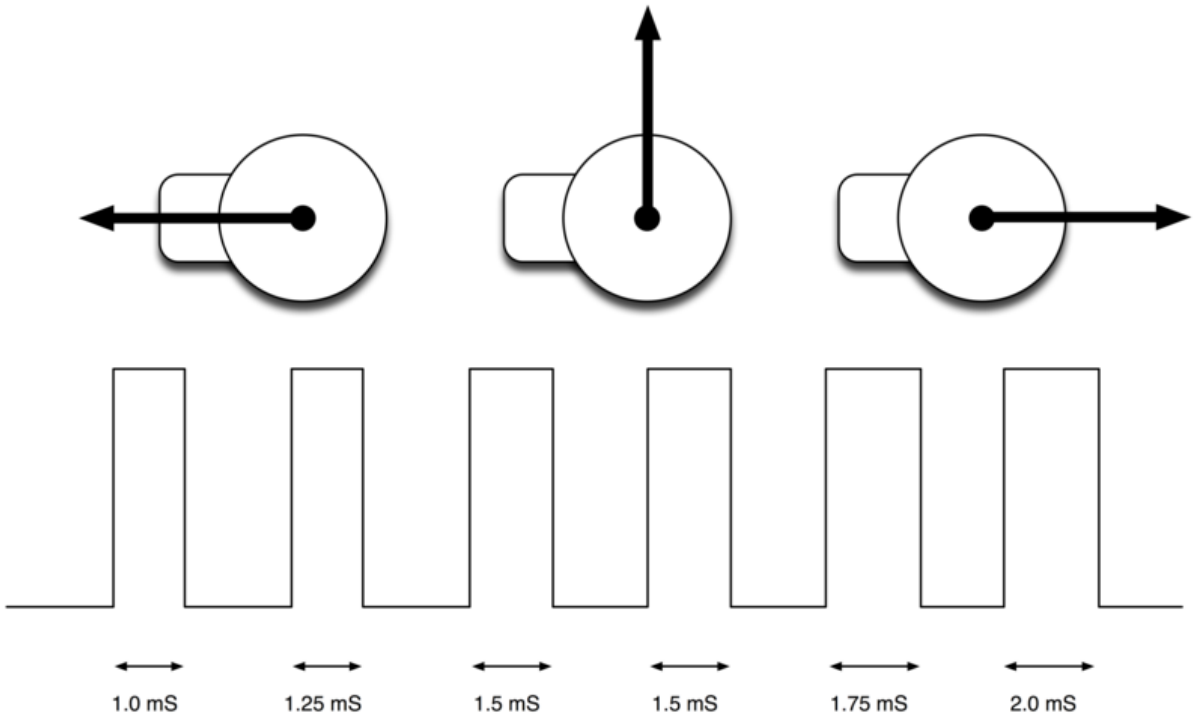

1.0 mS
1.25 mS
1.5 mS
1.5 mS
1.75 mS
2.0 mS


1.0 mS
1.25 mS
1.5 mS
1.5 mS
1.75 mS
2.0 mS


1.0 mS
1.25 mS
1.5 mS
1.5 mS
1.75 mS
2.0 mS


1.0 mS
1.25 mS
1.5 mS
1.5 mS
1.75 mS
2.0 mS


1.0 mS
1.25 mS
1.5 mS
1.5 mS
1.75 mS
2.0 mS


1.0 mS
1.25 mS
1.5 mS
1.5 mS
1.75 mS
2.0 mS