ROS framework으로 젯봇 프로그래밍

여기서는 기존의 젯봇 프로그래밍을 ROS Framework으로 하는 방법을 생각해 본다.
본 장은 주로 jetbot_ros 을 참고했다.
앞서 글에서 진행한 기본 SD card를 통해 CUDA, cuDNN, TensorRT 가 설치되어 있음을 가정한다.
바로 ROS Melodic을 설치해 보자

ROS Melodic 설치

아래 설명을 따라하는 것으로 충분하지만 자세한 설치 방법은 ROS Wiki를 참조한다.

```
# 먼저 ROS 패키지를 받아올 저장소(repository)를 Ubuntu의 패키지 매니저인 apt 설정화일로 추가등록한다.
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
# 이후 패키지의 무결성 검증을 위한 key를 키서버로부터 받아온다.
# 만약 proxy 뒤에서 진행하여 key를 가져오는 데 문제가 있다면
# curl을 사용하여 일단 key를 화일로 가져오고 apt-key로 등록하는 방법도 있다.
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

# 기본 ROS 패키지를 설치한다.
$ sudo apt update
$ sudo apt install ros-melodic-ros-base

# 터미널을 열때마다 ROS framework을 사용하기 위한 환경변수가 설정되도록 bash 쉘 스크립트에 source 문을 추가한다.
sudo sh -c 'echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc'

# 본 장에서 설치하고자 하는 ROS 패키지가 의존하는 라이브러리 추가
$ sudo apt-get install ros-melodic-image-transport
$ sudo apt-get install ros-melodic-image-publisher
$ sudo apt-get install ros-melodic-vision-msgs
```

jetson-inference 엔진 설치

이미지 추론 엔진을 설치한다. 다양한 기훈련 모델의 C++ API를 제공된다. 나중에 ROS 노드에서 활용된다.
아래를 따라하는 것으로 충분하지만 자세한 설치 방법은 dusty-nv/jetson-inference 을 참고한다.

```
# 본 엔진을 빌딩하기 위해 필요한 git, cmake를 설치한다.
# 대부분 기본으로 이미 설치되어 있을 것이다.
sudo apt-get install git cmake

# 추론 엔진의 소스코드를 내려받는다.
cd ~
git clone -b onnx https://github.com/dusty-nv/jetson-inference
cd jetson-inference
git submodule update --init

# 소스코드를 빌딩한다.
# jetson nano에서 실행하면 상당한 시간이 필요하다.
mkdir build
cd build
cmake ../
make

# 빌딩한 추론 라이브러리를 표준 디렉토리로 이동시킨다.
sudo make install
```

적절한 JetPack 버전을 설치하지 않을 시 위 작업중 cmake ../ 가 완료되지 않고 다음과 같은 오류메시지를 볼 수 있다.

```
-- jetson-utils:  building as submodule, /home/jetbot/jetson-inference
qmake: could not exec '/usr/lib/aarch64-linux-gnu/qt4/bin/qmake': No such file or directory
CMake Error at /usr/share/cmake-3.10/Modules/FindQt4.cmake:1320 (message):
  Found unsuitable Qt version "" from NOTFOUND, this code requires Qt 4.x
Call Stack (most recent call first):
  utils/CMakeLists.txt:12 (find_package)
```

이 경우 다음과 같이 필요한 라이브러리를 설치하고 다시 cmake ../ 를 진행한다.

```
sudo apt install libqt4-dev
```

make 를 실행하는 중 다음과 같은 오류 메시지가 나오면

```
/home/jetbot/jetson-inference/utils/display/glUtility.h:27:10:
```

```
 fatal error: GL/glew.h: No such file or directory
 #include <GL/glew.h>
          ^~~~~~~~~~
compilation terminated.
```

다음과 같이 필요한 라이브러리를 설치하고 다시 make 를 실행한다.

```
sudo apt install libglew-dev
```

catkin 작업공간 생성

젯봇에 설치할 패키지를 위한 작업공간을 만든다. 여기서는 다음의 2개의 패키지를 설치할 것이다.

- ros_deep_learning 패키지
- jetbot_ros 패키지

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws
$ catkin_make

# 방금 만든 catkin 작업공간을 ROS에서 사용하기 위한 환경변수가
# 터미널을 열때마다 자동 설정하도록 .bashrc 화일에 source 실행문을 추가
$ sudo sh -c 'echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc'
```

ros_deep_learning 패키지 설치

이 패키지는 앞서 설치한 추론 엔진을 사용하며 다음의 3개의 노드를 제공한다.

- imagenet 노드: 이미지 메시지를 구독하여 이미지의 종류를 구별한다.
- detectnet 노드: 이미지 메시지를 구독하여 이미지 내의 여러 사물과 그 위치를 식별한다.
- segnet 노드: 이미지 메시지를 구독하여 이미지 안의 여러 사물 사이의 경계를 식별한다.

다음과 같이 이 패키지를 설치한다. 본 패키지에 대한 자세한 설명은 dusty-nv/ros_deep_learning 을 참고한다.

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/dusty-nv/ros_deep_learning
$ cd ../
$ catkin_make
```

imagenet 노드 테스트
본 패키지에서 제공하는 imagenet 노드를 다음과 같이 테스트해본다.
테스트 입력으로 orange_0.jpg 그림화일을 사용했다.

```
$ roscore
$ rosrun image_publisher image_publisher __name:=image_publisher ~/jetson-
inference/data/images/orange_0.jpg &
$ rosrun ros_deep_learning imagenet /imagenet/image_in:=/image_publisher/image_raw
_model_name:=googlenet
```

잠시 기다려면 다음과 같은 결과를 얻을 수 있다. (결과가 계속 스크롤되는 것을 멈추려면 Ctrl-C를 누루면 된다.)

```
...
[ INFO] [1570620596.934852418]: converting 1920x1920 bgr8 image
class 0950 - 0.978909  (orange)
class 0951 - 0.020962  (lemon)
[ INFO] [1570620597.009793914]: classified image, 0.978909 orange (class=950)
```

즉 아래의 orange_0.jpg 그림을 97.8%의 확률로 class 0950 (orange) 으로 인식한다는 의미이다.
차순위 확률로 class 0951 (lemon) 을 2.09%의 확률로 인식했다.

이번에는 입력 이미지로 granny_smith_0.jpg 화일을 입력하여 실행해 보았다.

결과는 다음과 같다.

```
[ INFO] [1570621425.684989610]: converting 400x400 bgr8 image
class 0948 - 0.999713  (Granny Smith)
[ INFO] [1570621425.717892303]: classified image, 0.999713 Granny Smith (class=948)
```

detectnet 노드의 테스트

이번에는 detectnet 노드를 다음과 같이 테스트해 본다.

```
$ rosrun image_publisher image_publisher __name:=image_publisher ~/jetson-
inference/data/images/peds-004.jpg
$ rosrun ros_deep_learning detectnet /detectnet/image_in:=/image_publisher/image_raw
_model_name:=pednet
```

발행한 이미지는 peds-004.jpg로 다음과 같다.

결과는 아래와 같다. 위 결과는 5명의 Object가 탐지되었고 모두 class #0로 사람(person)으로 판정된다. 위 그림에서 서있는 5명의 사람에 해당한다.

안타깝게도 github에서 detectnet 노드의 코드를 살펴보니 인식된 object를 boxing하여 합성된 이미지를 출력하는 코드는 구현되어 있지 않다.

```
[ INFO] [1570623639.552472411]: converting 1024x611 bgr8 image
[ INFO] [1570623639.733503133]: detected 5 objects in 1024x611 image
object 0 class #0 (person)  confidence=0.860840
object 0 bounding box (692.125000, 43.632202)  (841.000000, 460.040039)  w=148.875000
h=416.407837
object 1 class #0 (person)  confidence=0.901855
object 1 bounding box (851.312500, 59.966309)  (1013.937500, 490.619873)  w=162.625000
h=430.653564
object 2 class #0 (person)  confidence=1.095703
object 2 bounding box (16.562500, 13.425293)  (227.250000, 559.238037)  w=210.687500  h=545.812744
object 3 class #0 (person)  confidence=0.683105
object 3 bounding box (374.218750, 34.756592)  (619.234375, 598.618896)  w=245.015625
h=563.862305
object 4 class #0 (person)  confidence=0.908203
object 4 bounding box (549.078125, 129.889709)  (617.781250, 318.999878)  w=68.703125
h=189.110168
```

ros_graph로 현재의 발행과 구독 상황을 보면 다음과 같다. 즉 image_publisher 노드가 /Image_raw 토픽으로 메시지를 발행하고 이를 detectnet 노드가 구독하는 상황이다.

ROS 노드에서 deep learning 추론 API 사용 방법

deep learning 추론 라이브러리를 ROS 노드에서 어떻게 호출되는지 살펴보겠다.

imagenet 노드를 구현한 에서 이미지를 구독했을 때 void img_callback(const sensor_msgs::ImageConstPtr& input) 함수가 불려진다.

이 함수는 아래와 같다. 처음 부분은 입력 이미지를 추론 엔진이 사용하는 입력 포맷으로 변환하는 부분이다.

이후 이미지를 분류하기 위해 deep learning 추론 엔진을 부르는 부분은 *net->Classify();* 이다.

4개의 파라미터를 요구하는데 첫번째는 이미지이고 두번째, 세번째는 이미지의 size 정보이다. 나머지는 추론 확률을 받기 위한 변수이다.

이 함수는 분류한 이미지의 레이블 번호를 반환한다. 자세한 정보는 를 참고한다.

```
void img_callback( const sensor_msgs::ImageConstPtr& input )
{
    // convert the image to reside on GPU
    if( !cvt || !cvt->Convert(input) )
    {
        ROS_INFO("failed to convert %ux%u %s image", input->width, input->height, input->encoding.c_str());
        return;
    }

    // classify the image
    float confidence = 0.0f;
    const int img_class = net->Classify(cvt->ImageGPU(), cvt->GetWidth(), cvt->GetHeight(), &confidence);

    // verify the output
    ...
}
```

위 코드에서 net 변수를 초기화 하는 방법은 다음과 같다.

```
#include <jetson-inference/imageNet.h>
imageNet::NetworkType model = imageNet::NetworkTypeFromStr("googlenet"));
imageNet* net = imageNet::Create(model);
```

처음에는 ROS의 python2 code로 구현한 노드가 어떻게 deep learning 의 python3 API를 부르는지 궁금했는데 실상은 python을 사용하지 않고 둘다 C++로 코딩되어 있었다.

jetbot_ros 패키지 설치

이 패키지는 다음의 3개의 노드를 제공한다.

- jetbot_motors.py 노드: 젯봇의 모터 제어 메시지를 구독하여 모터를 제어하는 노드
- jetbot_oled.py 노드: 문자열 메시지를 구독하여 OLED display 에 표시하는 노드
- jetbot_camera 노드: 카메라에서 이미지 프레임을 얻어 이를 발행하는 노드

이 패키지를 설치하기 전에 python2 코드로 모터를 구동할 수 있게 해주는 Adafruit 모터 드라이버 라이브러리와 OLED 디스플레이 드라이버를 먼저 설치해야 한다. jetbot에는 이미 기본으로 "기본 동작 확인" 장에서 사용한 jetbot이라는 모터 제어 라이브러리가 있으나 이는 python3 코드로 작성되어야 이 라이브러리를 사용할 수 있다. 그러나 ROS는 python3를 지원하지 않기 때문에 따로 python2용 드라이버를 설치해야 한다.

```
$ sudo apt install python-pip
$ pip install Adafruit-MotorHAT
$ pip install Adafruit-SSD1306
```

이후 패키지 설치는 다음과 같이 한다.

```
$ cd ~/workspace/catkin_ws/src
$ git clone https://github.com/dusty-nv/jetbot_ros
$ cd ../
$ catkin_make
```

jetbot_motors.py 노드 테스트

jetbot_motors.py 노드는 다음의 3개의 토픽을 구독한다.

1. /jetbot_motors/cmd_dir : 진행 방향 (degree [-180.0, 180.0], speed [-1.0, 1.0])
2. /jetbot_motors/cmd_raw : 왼쪽/오른쪽 모터의 속도(speed [-1.0, 1.0], speed [-1.0, 1.0])
3. /jetbot_motors/cmd_str : String으로 표현된 진행 명령 (left/right/forward/backward/stop)

그런데, 이 글을 작성하는 현재는 3번 토픽만 구현되어 있다.
jetbot_motors.py 노드를 테스트하기 위해 다음과 같이 이 노드를 시작한다. (roscore는 동작중이라 가정)

```
$ rosrun jetbot_ros jetbot_motors.py
```

다음과 같이 메시지를 발행한다. 모터가 메시지에 따라 움직이는 것을 볼 수 있다.

```
$ rostopic pub /jetbot_motors/cmd_str std_msgs/String --once "forward"
$ rostopic pub /jetbot_motors/cmd_str std_msgs/String --once "backward"
$ rostopic pub /jetbot_motors/cmd_str std_msgs/String --once "left"
$ rostopic pub /jetbot_motors/cmd_str std_msgs/String --once "right"
$ rostopic pub /jetbot_motors/cmd_str std_msgs/String --once "stop"
```

만약 동작하지 않으면 다음과 같이 jetbot 사용자를 i2c 그룹에 포함시키고 다시 부팅한 다음 실행해 본다.
가이드에 이것을 먼저 실행하도록 설명되어 있으나 실제로 해보니 이를 하지 않아도 실행되었다.

```
$ sudo usermod -aG i2c $USER
```

jetbot_oled.py 노드 테스트

jetbot_oled.py 노드는 /jetbot_old/user_text 토픽을 구독하여 구독된 메시지를 OLED display에 출력한다.
테스트는 다음과 같이 할 수 있다. "Auroca!"가 출력된다.

```
$ rosrun jetbot_ros jetbot_oled.py &
$ rostopic pub /jetbot_oled/user_text std_msgs/String --once "Auroca!"
```

jetbot_camera 테스트
jetbot_camera 노드를 다음과 같이 실행해 보자.

```
$ rosrun jetbot_ros jetbot_camera
```

이후 카메라로 들어오는 비디오 프레임은 /jetbot_camera/raw 토픽으로 발행된다.
이때 타입은 sensor_msgs::Image 이며 BGR8 형식으로 인코딩된 이미지이다.
안타깝게도 발행되는 이미지를 볼수 있는 구독 노드가 참고 자료에서 주어지지 않았다.
하지만 `rqt_image_view` 도구를 사용하면 쉽게 볼 수 있다.
다음을 실행하고 /jetbot_camera/raw를 구독해 본다.

```
$ rqt_image_view
```

이 이미지 메시지 타입에 대한 자세한 정보는 `sensor_msgs/Image` 참고한다.

마치며

지금까지 소개한 ROS Framework의 노드를 활용하면 앞장의 충돌회피, 물체따라가기, 길 따라가기를 ROS framework으로 재작성할 수 있다.
즉 jetbot_camera 노드로 카메라 이미지를 발행하고 이를 imagenet/detectnet/segnet 노드로 실시간 분석하여 적절하게 모터제어 메시지를 발행한다. 최종적으로 jetbot_motor.py 노드가 이 메시지를 구독하여 모터를 움직이게 하는 식이다.
이를 그래프로 도시하면 아래와 같다.

이를 위해서는 imagenet/detectnet/segnet 노드가 이미지의 분석결과로부터 어떻게 모터를 제어할 지에 대한 제어 전략 구현과 jetbot_motors.py 노드로 메시지를 발행하는 코드를 추가하는 수정이 필요하다.
기존의 프로그래밍 방법에 비해 ROS를 사용하여 프로그래밍하는 방법의 장점은 여기서는 사실 크지 않을 것 같다. 다만 ROS 프로그래밍을 연습하는 예로서 의미는 찾을 수 있다.