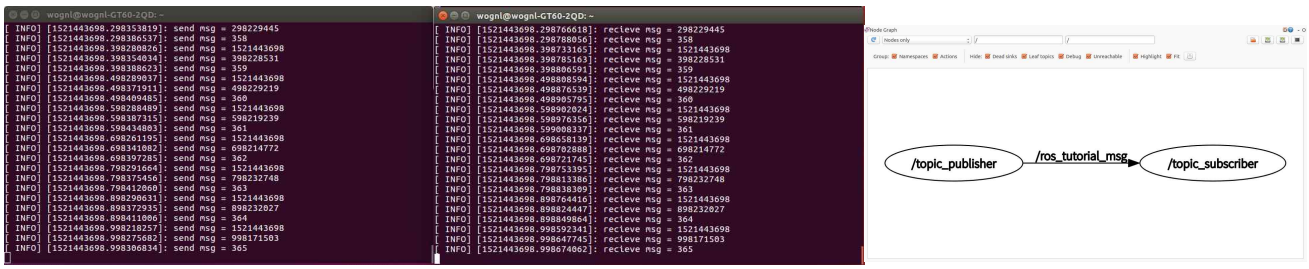
20140496 장재휘 / Team 7 (김진환 교수님) / ROS

[Week 1 - ROS Basic]

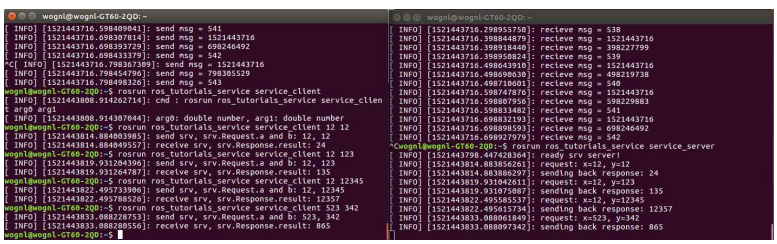
In the very first week of the sesssion is about basic of ROS. We start off with installing Linux operating system and ROS workspaces. As Ubuntu 16.04 is the most common Linux based operating system, all of the works are done in Ubuntu 16.04. We can easily install the ROS package by referring ROS wiki - http://wiki.ros.org/ko/groovy/Installation/Ubuntu.

ROS systems are made of various nodes which do individual operation. Each node communicate each other in three ways - Topic, Service and Action. Basically nodes are generated in C++ language. Cmakelist.txt file contains all the information about compiling C++ files. Thus we have to be accurate on grammers at every sequence.

Our first assignment is to demonstrate those communication with basic examples. Publisher node spreads out the messages under certain topic. And Subscriber node with the same topic can receive those messages. Our topic communication example is to publish the information of current time and subscribe it. And for the bonus task, we make one more consecutive publish-subscription node. I think most important part is 'NodeHandle' function. We can imprint the publication or subscription function with 'NodeHandle' function. I think I may search up 'NodeHandle' function more for a possible advanced task.



And the next assignment is to demonstrate the communication with service. Unlike topic communication, it only works when the client node asks for it. Thus server node can send back the information as a consequence. Our example is to request simple addition work from client node and server node calculate and send the result back.



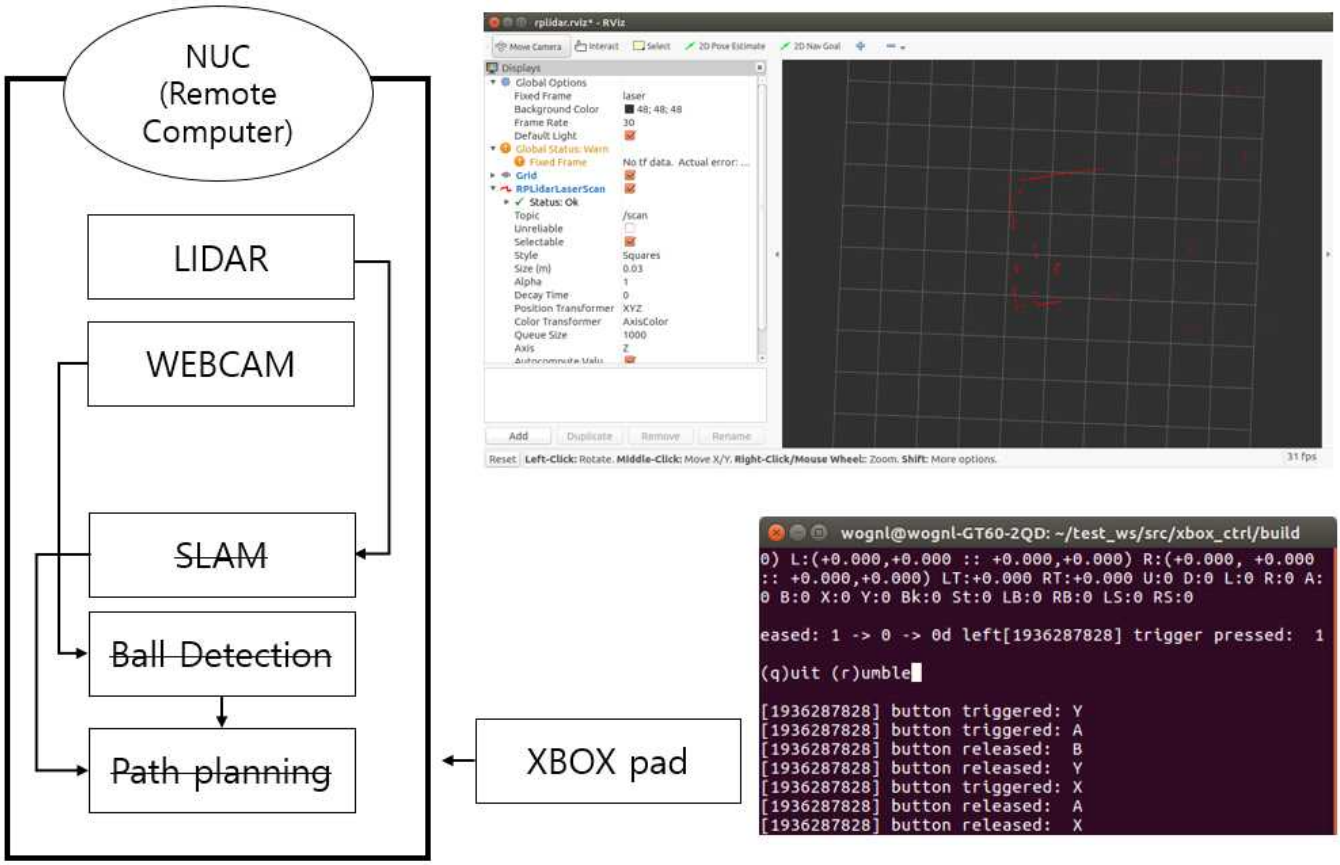
The last assignment is to use roslauncher. The roslauncher helps users to make preset form of

nodes so they can recall every time they want.

[Week 2 – Basic Robotic built in codes]

The most powerful advantage of ROS is that we can use programs that is made by others. In the second week of the session is to explain the codes that TAs used in Naver labs.

We followed several codes that activate Webcam, RPLIDAR and XBOX controllers.
Information of Webcam should be treated with OpenCV programs. Especially boundary detecting tehcnology would be implemented. RPLIDAR data will be handled with RVIZ software which helps us to visualize the data taken from RVIZ. Since we don't have SLAM or pathfinding programs yet, we will replace those output data with XBOX controllers. I think we can implement other open source code for SLAM or Pathfinding algorithms.



RPLIDAR data show the structure of my room. and XBOX_ctrl package receives all the inputs from the XBOX controller(XBOX pad).

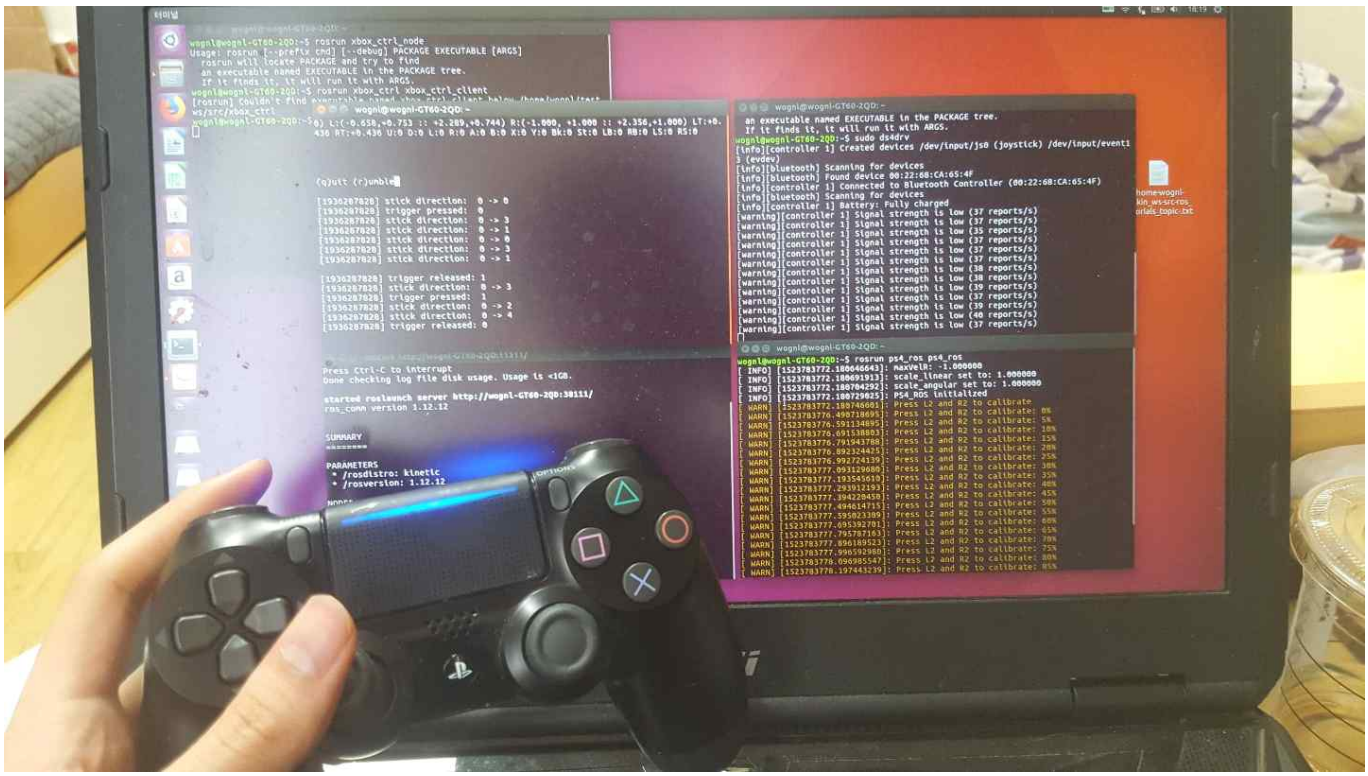
[Week 3 - Basic Robotic built in codes]

In this week, I figured out how to use PS4 Dualshock instead of XBOX360. As XBOX360 device needs additional receiving device, it would be a bit bothersome in general. Also, this device would be used when the XBOX controller is in use for other team (LabView / Solidworks etc).

We should use basically same xbox_ctrl package in order to keep the coherence. I installed ps4 Ubuntu driver and ps4_ros package. Then, ps4_ros package would generate the same signal inputs those from the XBOX controller. All the process are based on https://github.com/solbach/ps4-ros.

First, we install Dualshock4 driver. Then, install ros-joy package (currently the package is updated to ros-kinetic-joy package). Finally, install ps4_ros package developed by solbach.

We can check that signal from Dualshock gives compatible result from XBOX controller



[Now I can use DS4 instead of XBOX controller]

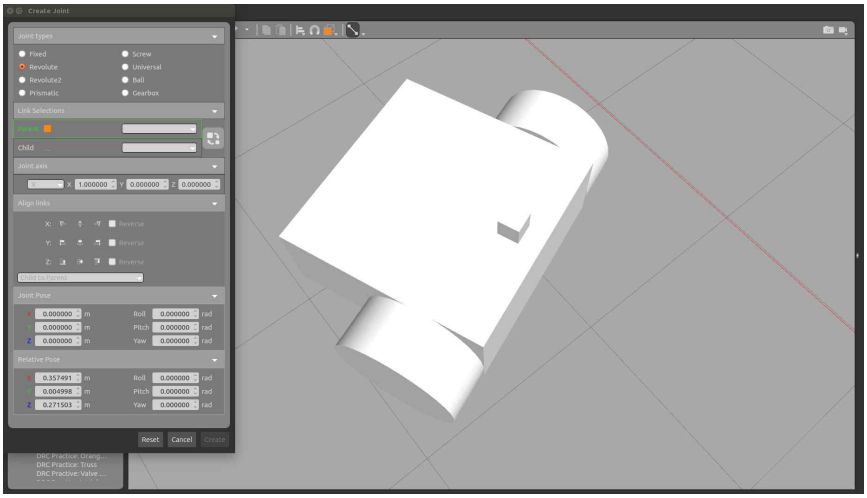
[Week 4 - Basic Gazebo Tutorials]

Gazebo is a 3D dynamic simulator for robots in various environments. It has its own physics engine with high fidelity and accuracy.

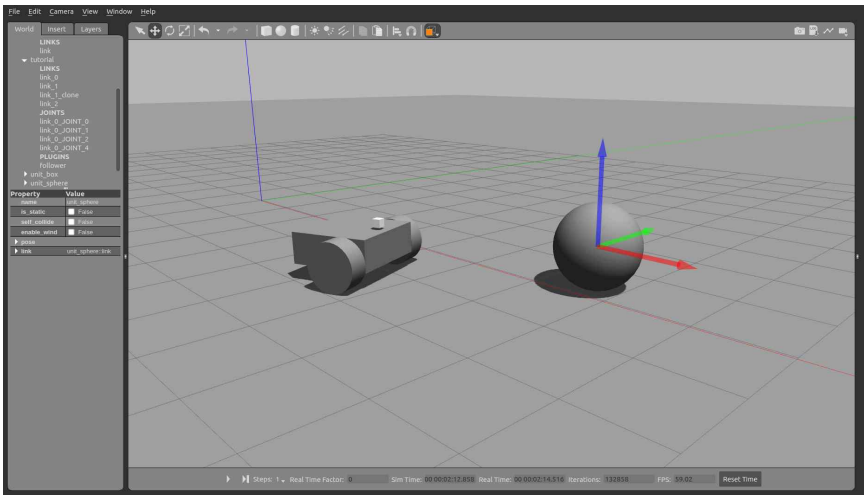
we may use Gazebo to simulate the prototype of our robot. We may test some coded algorithms such as path finding or mapping other than physical behaviors of the robots.

As it is an open source program, we can easily install Gazebo with detailed instruction form the offical website: http://gazebosim.org/tutorials.

We tested the Gazebo program with basic tutorials given.

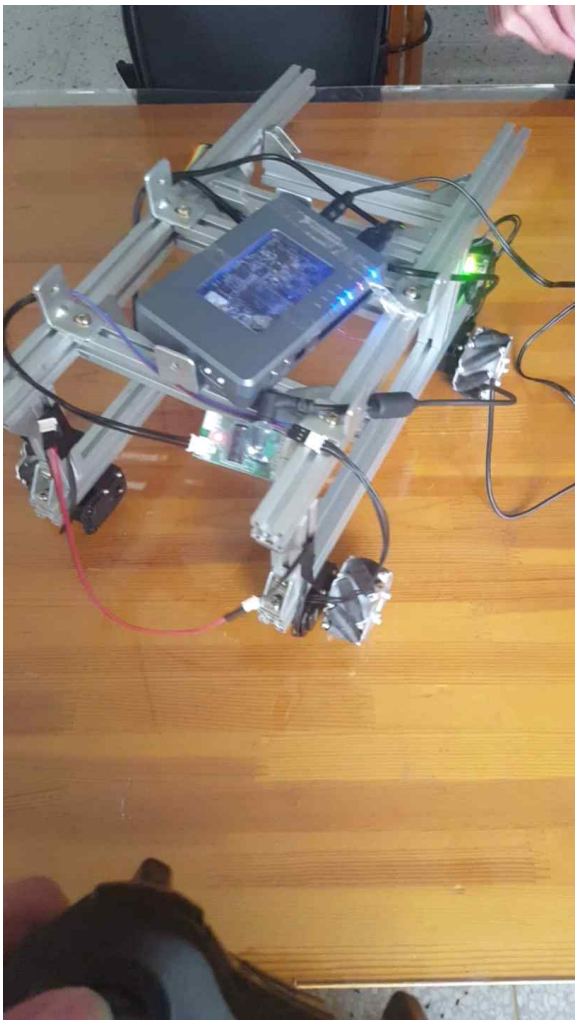


In Gazebo tutorial session, we are followed to make three-wheeled vehicles mounted with depth sensor.



Generated robot follows the object(Sphere) based on the data collected from the depth sensor.

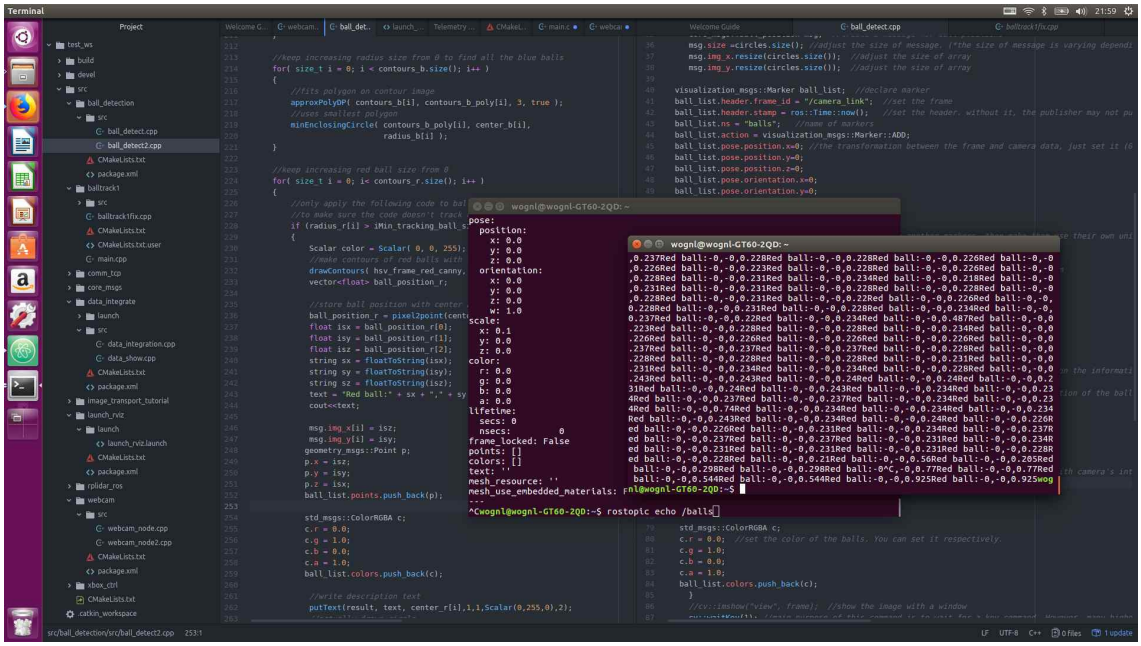
[Week 5&6 - ROS Integration]

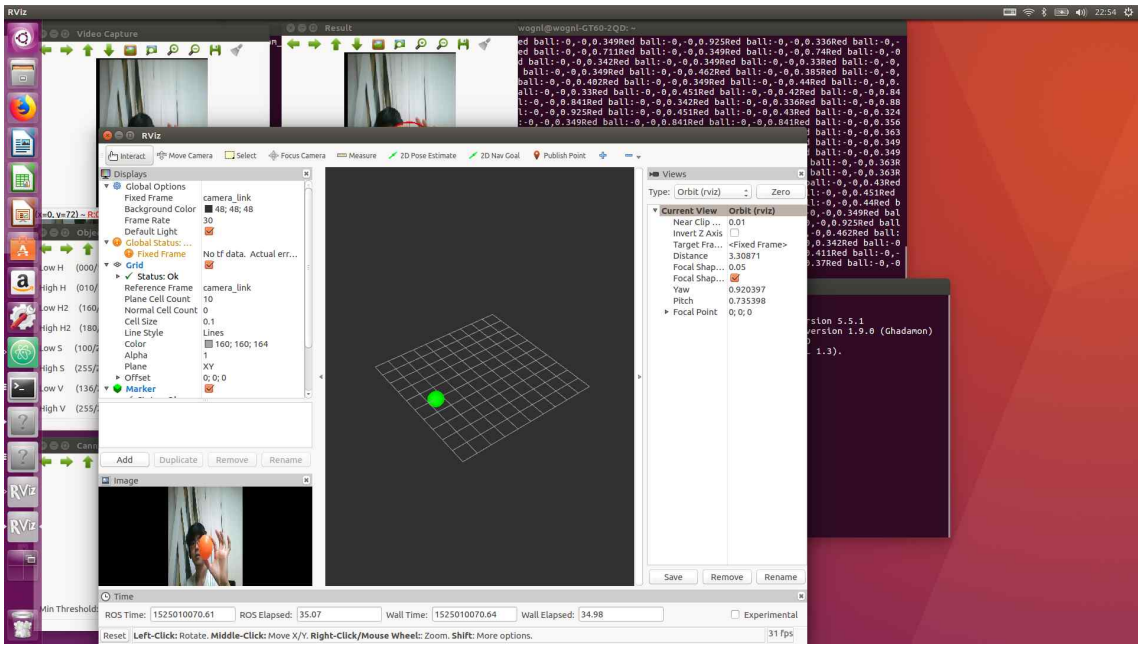


(전후좌우 + 회전이 가능한 4 DOF 조작)

구동 장치와의 연동을 위해 xbox_ctrl의 tcpip 데이터 통신을 테스트 하고 전송되는 데이터 형식을 파악한다. 전송되는 데이터는 24개의 float 데이터로 결정된다. 하지만 그 중에 사용되는 데이터는 4개의 Degree of Freedom으로 결정되므로 허비되는 트래픽 및 자원 소모를 줄일 수 있을 것이라고 생각된다.

OpenCV와의 융합을 진행하였다. 전체적인 구조는 webcam 노드는 그대로 두고 webcam노드로부터 받은 데이터를 ball_detection 노드에서 OpenCV의 코드를 이어붙이는 형식을 취한다.

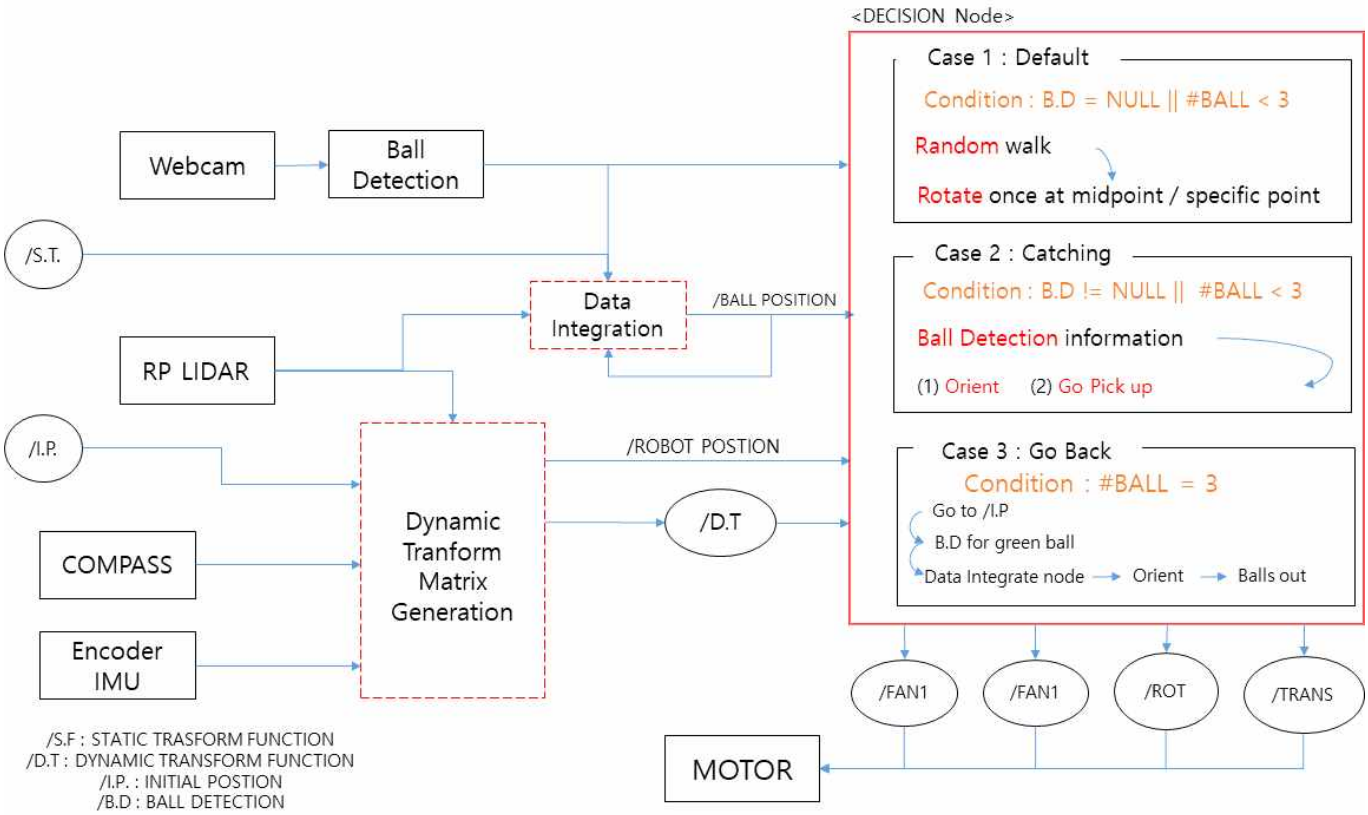




제일 먼저 OpenCV에서 사용되는 Libraries들을 사용하기 위해 적절한 Header를 선언해준다.
그 다음으로 이미지 처리에 사용되는 함수들을 선언하고 Implement해준다. 순서를 적절히 맞추어 오류가 나지 않도록 한다.
마지막으로 루프문을 매치 시켜 매 iteration마다 적절히 이미지 처리가 가능하도록 해준다.
이미지 처리를 통해 얻은 결과를 geometry_msg와 marker_msg 형식으로 변환을 시켜서 ros 서버 상의 topic으로 전송이 되는 것을 확인하고, 그 값을 rviz 상에서 찍었다.

다음 주부터 이 데이터를 이용해 tcpip 통신으로 모터를 운전할 수 있도록 한다.

[Week 7&8 - ROS Integration II]



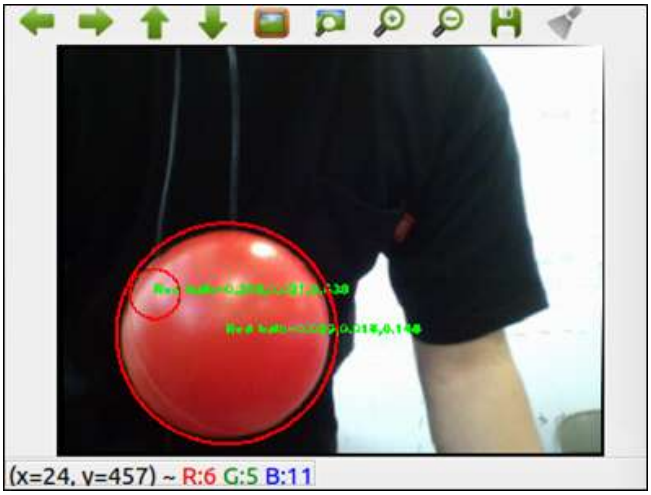
전반적인 ROS integration map을 구성하였다.

1차적인 목표로 case2: 공을 발견하여 움직이고 공을 집는 알고리즘을 구현 하도록 한다. 이를 구현한다면 전체적인 integration이 진행됨을 확인 시켜주고, case1, 3에도 응용하여 구현할 수 있다.

제일 먼저 TCPIP 통신을 통하여 모터로 구동정보를 전달을 구현했다. 기본적으로 xbox_ctrl 노드에서 사용된 TCPIP코드를 기반으로 제작되었다.
앞으로 xbox_ctrl 노드를 통해 자율주행과 수동주행을 변환할 수 있다.

그 다음으로 카메라로부터 인식된 데이터가 노이즈가 많기 때문에 데이터 선별 작업을 할 수 있는 노드를 추가했다. data_massage의 node2는 ball_detection으로부터 x,y,radius 데이터를 받고, 이 중에서 가장 큰 공을 선택하여 노이즈를 최소화하는 방향을 선택했다.

추가적으로 data_massage 패키지는 30Hz로 전달되는 위치 정보를 적절히 수정하여 myRIO의 100Hz 정보 수신율과 호환이 되도록 바꾸어 준다.

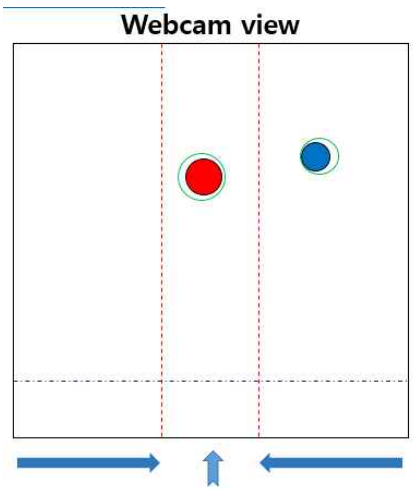


[공 하나에 여러 원이 인식되는 현상]

data_massage로 받은 x,y 데이터를 기반으로 로봇이 어느쪽으로 움직일지를 결정한다.
가장 간단하게 공이 가운데에 존재하면 전진, 왼편에 존재하면 좌회전, 오른편에 존재하면 우회전을 하도록 했다.

공을 집는 메카니즘으로는 공이 이미지 하단부에 위치하게 되면 ready_flag를 세우고, 이후 공이 이미지에서 없어지면 sweep_flag를 세운다. ready_flag와 sweep_flag가 동시에 세워지면 sweep이 진행되도록 코드를 구성했다.

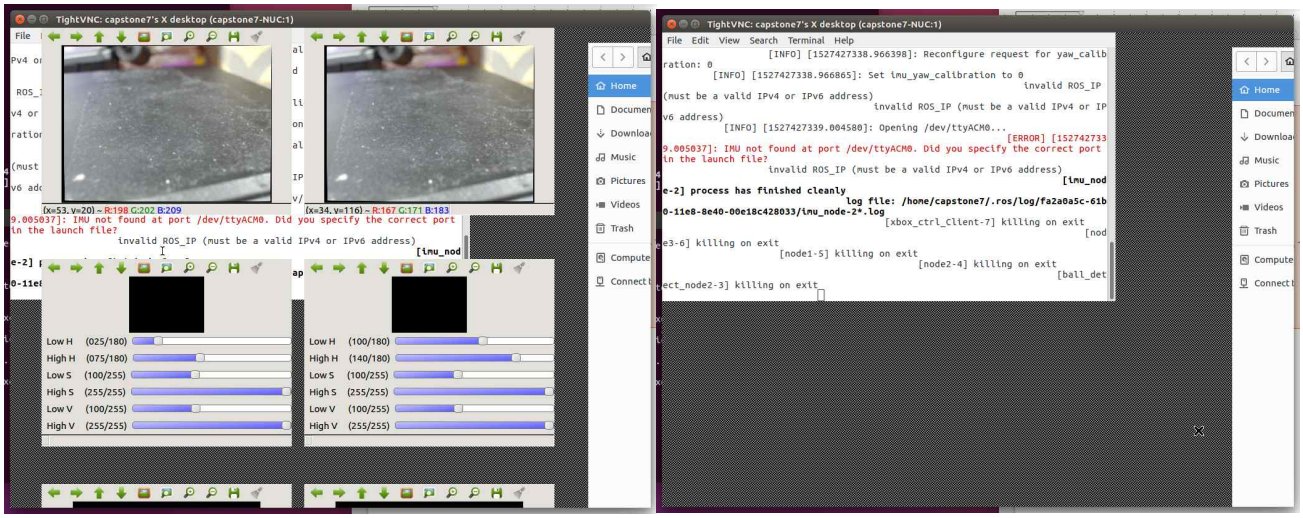
이후 공을 움직이는 알고리즘을 조금 더 매끄럽게 짤 것이고, 공을 집을 때의 오류가 줄어 들도록 디버깅을 하여 로봇 시행 성공률을 높이도록 한다.



[기본적인 구동 알고리즘]

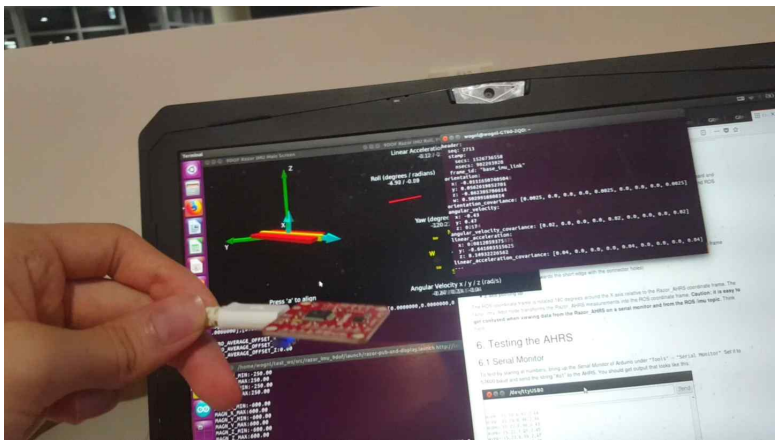
[Week 9&10 - ROS Integration III]

1. Run NUC with VNC server.



실제 경기장에서 NUC의 상태를 모니터로 확인을 할 수 없기에 vnc 서버를 이용하여 원격제어를 진행하기로 했다. Tightvncviewer 프로그램을 이용하여 진행 했으며, roscore를 비롯한 각종 ROS 명령어들을 전달하고, 모니터링이 가능한 것을 볼 수 있다.

2. IMU Implement



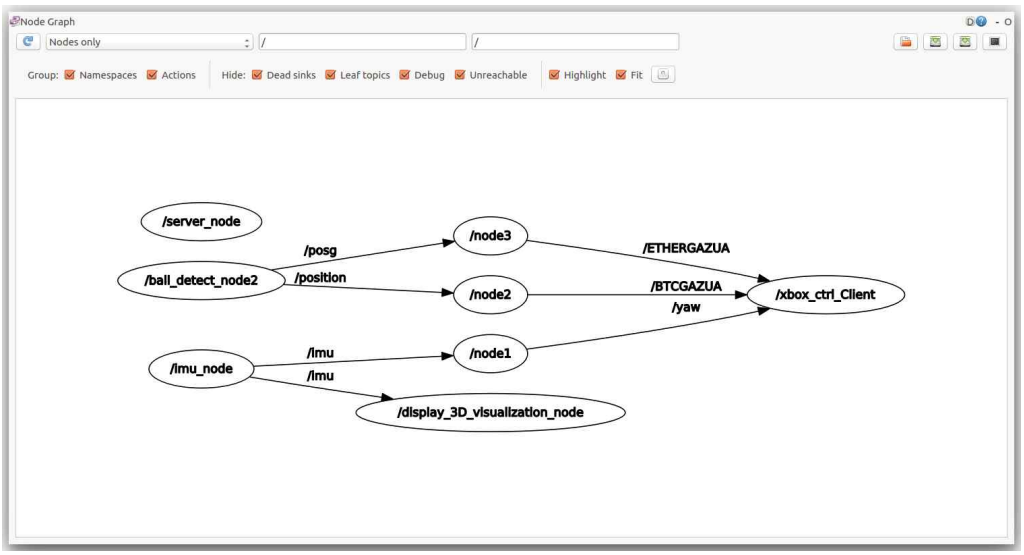
[IMU unit의 데이터를 ROS node로 전송하여 그래픽화 한 것]

골대로 로봇을 복귀시키기 위해서는 로봇의 위치를 절대좌표 기준으로 특정시킬 수 있어야 한다. 우리 조에서는 웹캠만으로 구현하기가 어렵다고 판단되어 IMU unit 중 compass 데이터를 이용하기로 했다. 사용된 모델은 Sparkfun 사의 SEN-14001 mpu 9250 이며, 내장된 가속도 센서, 자이로센서, 지자기센서를 이용하여 절대적인 방위를 계산할 수 있었다. 추후에 가속도 센서를 이용하여 진동 제어를 시도해보고자 한다.

```
lin_acc:        2.54726 0.176968        -9.78225        ang_vel:        -0.05   0.03    0.02
lin_acc:        2.59974 0.114914        -9.85848        ang_vel:        -0.05   0.03    0.02
lin_acc:        2.64762 0.143643        -9.62903        ang_vel:        -0.05   0.03    0.02
lin_acc:        2.4898  0.119894        -9.75352        ang_vel:        -0.05   0.03    0.02
lin_acc:        2.66218 0.2011  -9.79642        ang_vel:        -0.05   0.03    0.03
lin_acc:        2.55186 0.148622        -9.69607        ang_vel:        -0.05   0.03    0.03
lin_acc:        2.67635 0.0812059       -9.88721        ang_vel:        -0.05   0.03    0.03
lin_acc:        2.58557 0.0907821       -9.72479        ang_vel:        -0.05   0.03    0.02
lin_acc:        2.60932 0.0907821       -9.71522        ang_vel:        -0.05   0.03    0.03
lin_acc:        2.63345 0.0861855       -9.81558        ang_vel:        -0.05   0.03    0.02
lin_acc:        2.38945 0.229828        -9.68649        ang_vel:        -0.05   0.03    0.02
lin_acc:        2.51355 0.0861855       -9.94007        ang_vel:        -0.05   0.03    0.03
lin_acc:        2.63804 0.0812059       -9.72479        ang_vel:        -0.05   0.03    0.02
lin_acc:        2.50398 0.105338        -9.77267        ang_vel:        -0.05   0.02    0.02
lin_acc:        2.56143 0.0624366       -9.75352        ang_vel:        -0.05   0.03    0.02
lin_acc:        2.58557 0.12947 -9.77267        ang_vel:        -0.05   0.03    0.02
```

[IMU unit 가속도 센서 값을 저장 한 것, 이후 FFT 분석을 통해 진동을 분석하였다.]

3. ROS node graph



[rqt_graph를 통한 node 그래프]

ball_detect_node에서 파란색 공의 위치(/position)과 초록색 공의 위치(/posg)를 파악하고, IMU를 통하여 절대적 방위 정보를 얻는다. 이 데이터들을 사용하기 쉽게 가공 시킨뒤 (node1, node2, node3), xbox_ctrl_Client에서 정보를 통합하고 이를 이용하여 모터 제어를 실시한다.

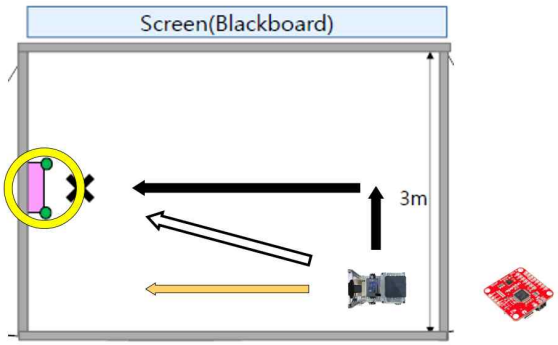
4. 최적화

a. webcam 노드와 ball_detect_node를 병합 시켜서 webcam에서 image transport에 낭비되는 자원을 절약하였다. 이를 통하여 카메라의 딜레이 및 해상도를 발전시킬 수 있었고, 로봇의 속도 한계와 정확도를 개선시킬 수 있었다.

b. 거리가 비슷한 공이 양쪽에 있을 경우, 오른 쪽에 있는 공을 주움과 동시에 다음 공을 찾을 때 좌회전을 하도록 최적화 시켰다.

[Week 11 - ROS Integration IV]

1. 귀환 알고리즘



기본적으로 IMU를 통하여 골대를 바라보는 방향을 기본 방향으로 설정을 했다.

a. 제일 먼저 초록색공을 회전하며 찾는다 (흰색 화살표)

b. 초록색 공을 찾았을 때, 기본 방향보다 오른쪽으로 틀어져 있는 경우, 기본 이동방향을 우측으로 설정한다.

c. 기본 방향을 바라보도록 한 후 (주황색 화살표) 기본 이동방향과 전진을 이용하여 초록색 공 쪽으로 이동한다 (검은 화살표)

d. 도착한 이후, 나침반 센서를 다시 이용하여 180도를 돌도록 한 후 공을 배출한다.

이와 같은 알고리즘을 사용하기 위해서는 초록색 공을 인식하는 알고리즘을 개선해야 한다.

2. 공 인식 개선

하나의 공만을 추적 할 때에는 가장 큰 반지름의 데이터를 추출하면 됐었으나, 두 개의 초록색 공을 인식 하기 위해서는 새로운 filtering이 필요하였다.

이를 위하여 겹치는 원들이 존재 할 때, 모든 원을 포함하는 거대한 사각형으로 합쳐 지도록 하였다. 이를 통하여 공의 중심 좌표와 반지름 정보는 보존하면서도 간단하게 여러개의 공을 인식할 수 있었다. 위와같은 프로세스를 통하여 두 개의 초록색 공 중점을 지날 수 있도록 알고리즘을 짤 수 있었다.