

# 제5회 국민대학교 자율주행 경진대회

## 예선 과제 안내

---

---

제 5회 자율주행 경진대회

---

예선과제 안내

---

일정 : 2022년 5월 25일

---

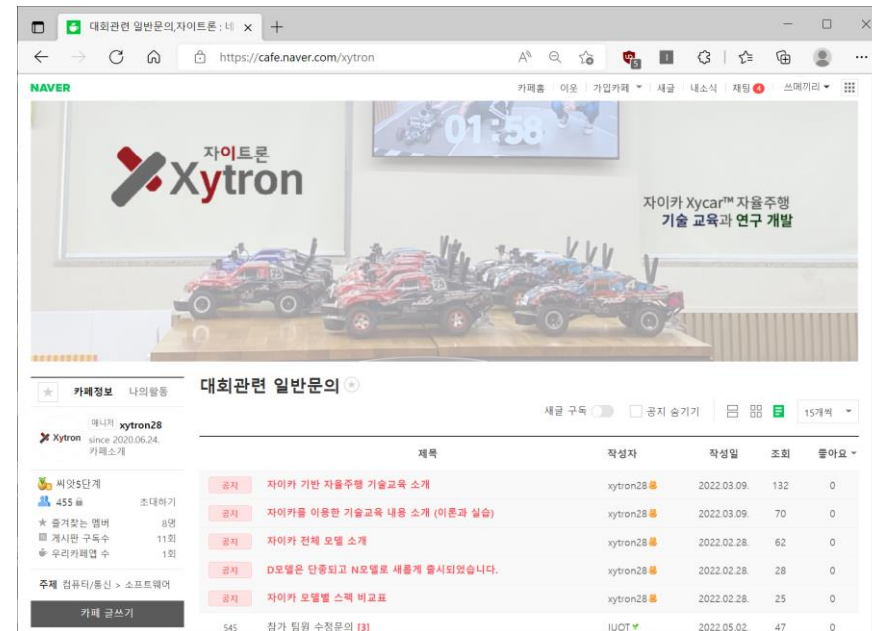
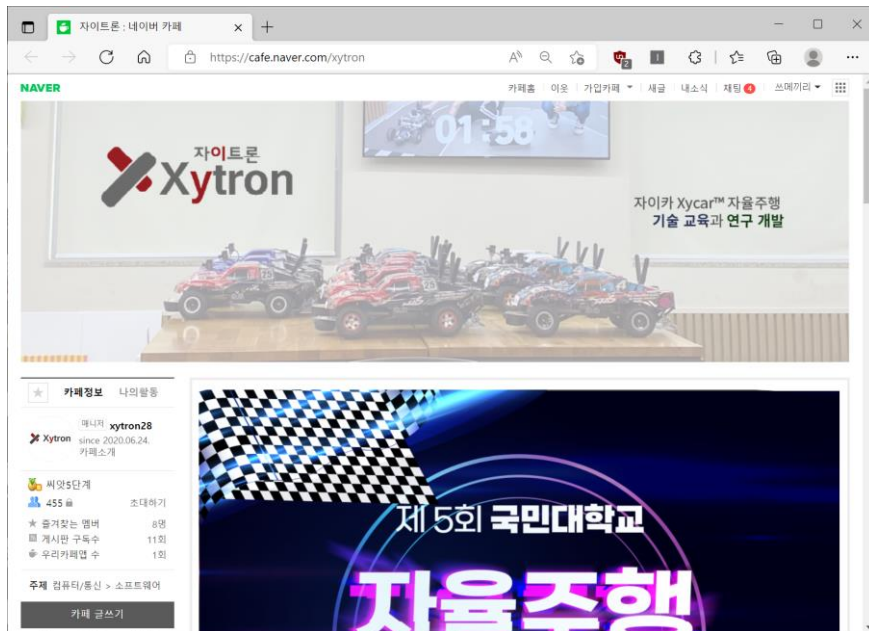
# 예선 안내

---

- 지원 자격
  - 서류심사 통과팀에게 예선 관련 자료를 배포합니다. (시뮬레이터, ROS 패키지, 설치 매뉴얼)
  - 예선 과제 심사를 통과한 팀에게 본선 진출 자격을 부여합니다.
- 과제
  - (1) 주행 시뮬레이터 환경에서 차선을 벗어나지 않고 목적지까지 주행하는 자율주행SW를 구현합니다.
  - (2) 주차 시뮬레이터 환경에서 AR태그를 이용하여 정확하게 주차하는 자율주차SW를 구현합니다.
  - (3) 다양한 미션을 수행하는 자율주행SW를 제작하는데 필요한 SW설계서를 작성합니다.
- 평가
  - (1)번 과제 : 차선을 벗어나지 않으며 목적지에 최대한 가까이 도달하는지와 주행 품질로 평가합니다.
  - (2)번 과제 : 4개 지정위치에서 출발하여 주차구역에 정확하게 주차하는지 여부로 평가합니다.
  - (3)번 과제 : SW 설계서의 완성도, 구체성, 타당성, 창의성 검토하여 평가합니다.

- 문의처

- 국민대 자율 주행 대회 홈페이지 게시판
- 자이트론 네이버 카페 - 제5회 국민대 대회 게시판  
( <https://cafe.naver.com/XYTRON> )
- 리눅스와 ROS의 설치와 관련된 문의는 스스로 해결해야 합니다.



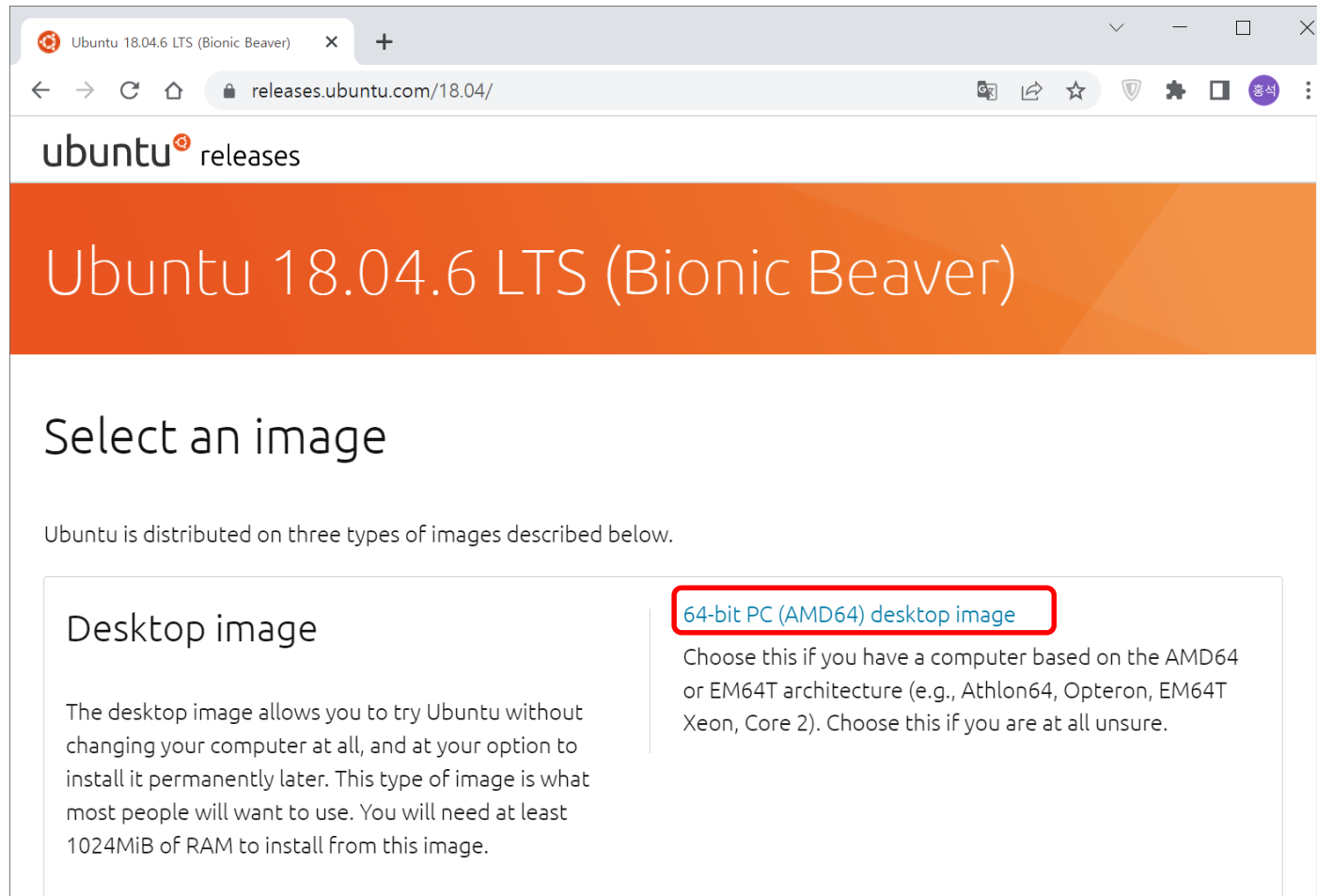
# 개발 환경 설정

---

리눅스 Ubuntu 18.04 설치  
ROS Melodic 설치  
의존 패키지 설치

# Ubuntu 18.04 설치

- 노트북이나 PC에 Ubuntu 18.04를 설치합니다.
  - 64bit PC 버전을 다운받아서 설치하면 됩니다.





# ROS 환경 설정

- ROS 작업에 필요한 환경변수를 설정하세요.
  - 홈 디렉토리에 있는 .bashrc 파일을 수정하면 됩니다.
    - ▶ `$ cd` (홈디렉토리로 이동)
    - ▶ `$ sudo gedit ~/.bashrc` (아래 내용을 마지막에 추가)
    - ▶ `$ source .bashrc` (수정한 내용을 시스템에 반영)

## .bashrc 파일의 내용

```
...  
alias cm='cd ~/catkin_ws && catkin_make'  
source /opt/ros/melodic/setup.bash  
source ~/catkin_ws/devel/setup.bash  
export ROS_MASTER_URI=http://localhost:11311  
export ROS_HOSTNAME=localhost
```

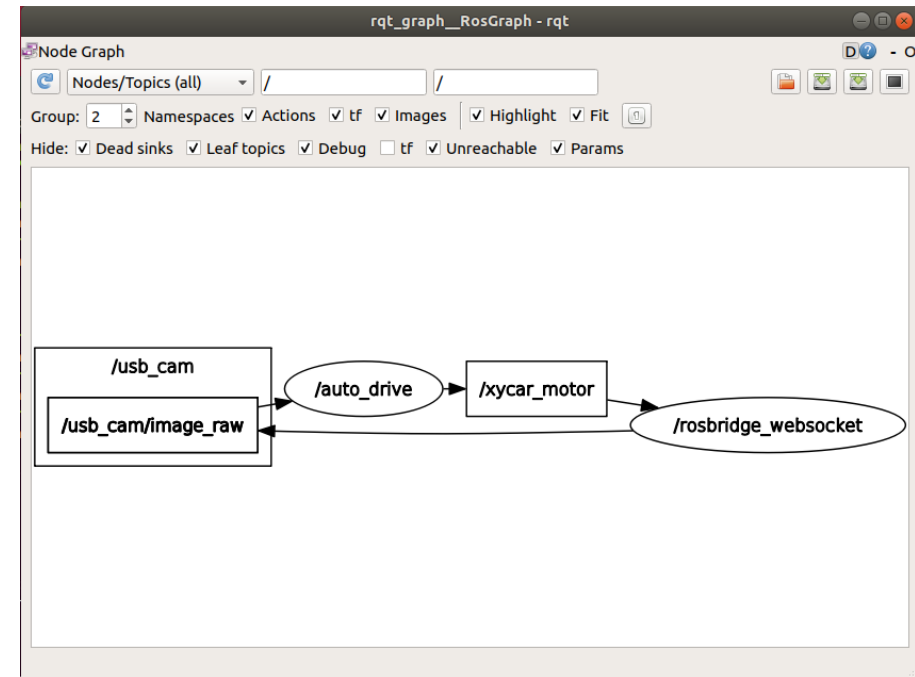
# 주행 시뮬레이터 의존 패키지 설치

- ROS Bridge 서버 설치 : 시뮬레이터와 ROS 시스템 간에 Web 소켓 통신을 담당.
  - `$ sudo apt update`
  - `$ sudo apt install ros-melodic-rosbridge-server`
  - `$ cm`



```
hongscho@sparc18: ~  
File Edit View Search Terminal Help  
hongscho@sparc18:~$ sudo apt install ros-melodic-rosbridge-server
```

터미널 창을 연 후 명령입력  
`$ sudo apt install ros-melodic-rosbridge-server`





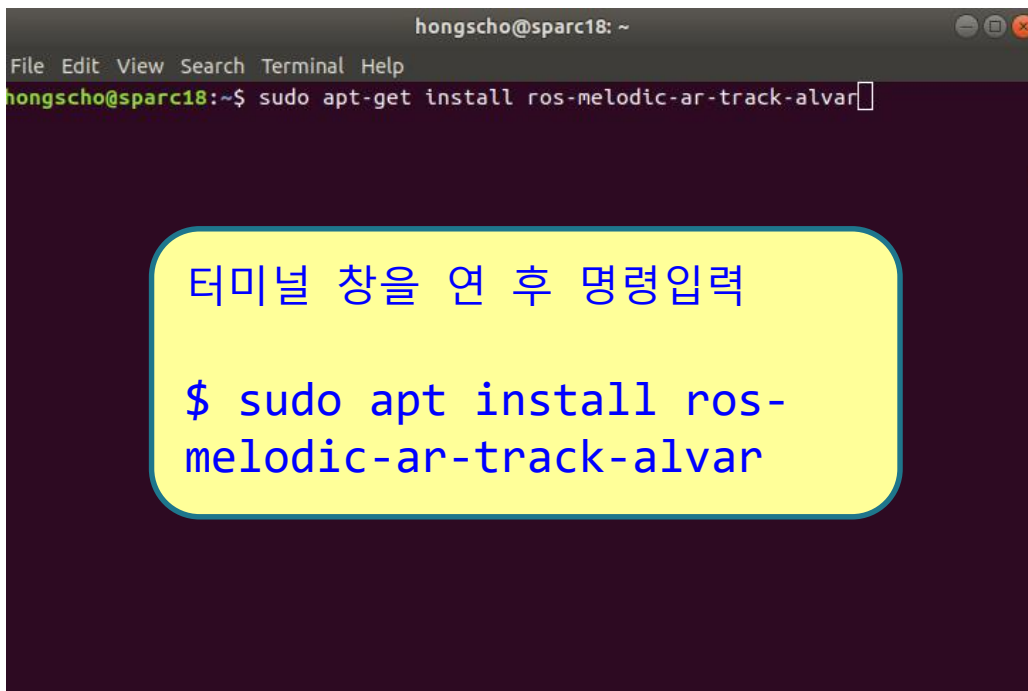
# 주차 시뮬레이터 의존 패키지 설치

---

- 제공된 주차 시뮬레이터를 사용하려면 아래 의존 패키지들의 설치가 필요합니다.
- 설치할 의존 패키지들 (여기 순서대로 설치하세요)
  - ar-track-alvar v0.7.1                      - Alvar AR Tag 전용 ROS 패키지
  - pygame v1.9.6                              - 파이썬 게임 제작 라이브러리
  - pillow v6.2.2                                - 파이썬 이미지 처리 라이브러리

# ar-track-alvar 설치 방법

- ar-track-alvar for ROS Melodic
  - `$ sudo apt update`
  - `$ sudo apt-get install ros-melodic-ar-track-alvar`



A terminal window titled 'hongscho@sparc18: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The command `hongscho@sparc18:~$ sudo apt-get install ros-melodic-ar-track-alvar` is entered. A yellow callout box with a blue border contains the text: '터미널 창을 연 후 명령입력' and '\$ sudo apt install ros-melodic-ar-track-alvar'.



ROS.org  
ar\_track\_alvar

이 패키지는 오픈 소스 AR 태그 추적 라이브러리인 [Alvar](#)을 ROS 래퍼입니다.

ar\_track\_alvar에는 4 가지 주요 기능이 있습니다.

1. 다양한 크기, 해상도 및 데이터 / ID 인코딩의 AR 태그 생성
2. 개별 AR 태그의 포즈를 식별하고 추적하여 선택적으로 더 나은 포즈 추정을 위해 키 벡트 깊이 데이터를 통합합니다 (키 벡트가 사용 가능한 경우).
3. 여러 태그로 구성된 "번들"의 자세를 식별하고 추적합니다. 이를 통해보다 안정적인 포즈 추정, 패색 견고성 및 다면체 추적이 가능합니다.
4. 카메라 이미지를 사용하여 번들의 태그 간 공간 관계를 자동으로 계산하므로 사용자는 번들 기능을 사용하기 위해 XML 파일에서 태그 위치를 수동으로 측정하고 입력 할 필요가 없습니다 (\*\* 현재 작동하지 않음-아래 참조).

Alvar는 다른 ROS AR 태그 패키지의 기반이 된 ARToolkit보다 훨씬 새롭고 고급입니다. Alvar는 다양한 조명 조건을 처리 할 수있는 적응형 임계 값보다 안정적인 포즈 추정을위한 광학 흐름 기반 추적 및 태그 수가 증가해도 크게 느려지지 않는 향상된 태그 식별 방법을 제공합니다.

# pip 설치

- pygame과 pillow를 설치하려면 pip(python package index)가 필요합니다.
- pip는 파이썬으로 작성된 패키지 소프트웨어를 설치, 관리하는 패키지관리 도구입니다.
- pip는 ubuntu 18.04의 기본 패키지가 아니므로 직접 설치해야 합니다.

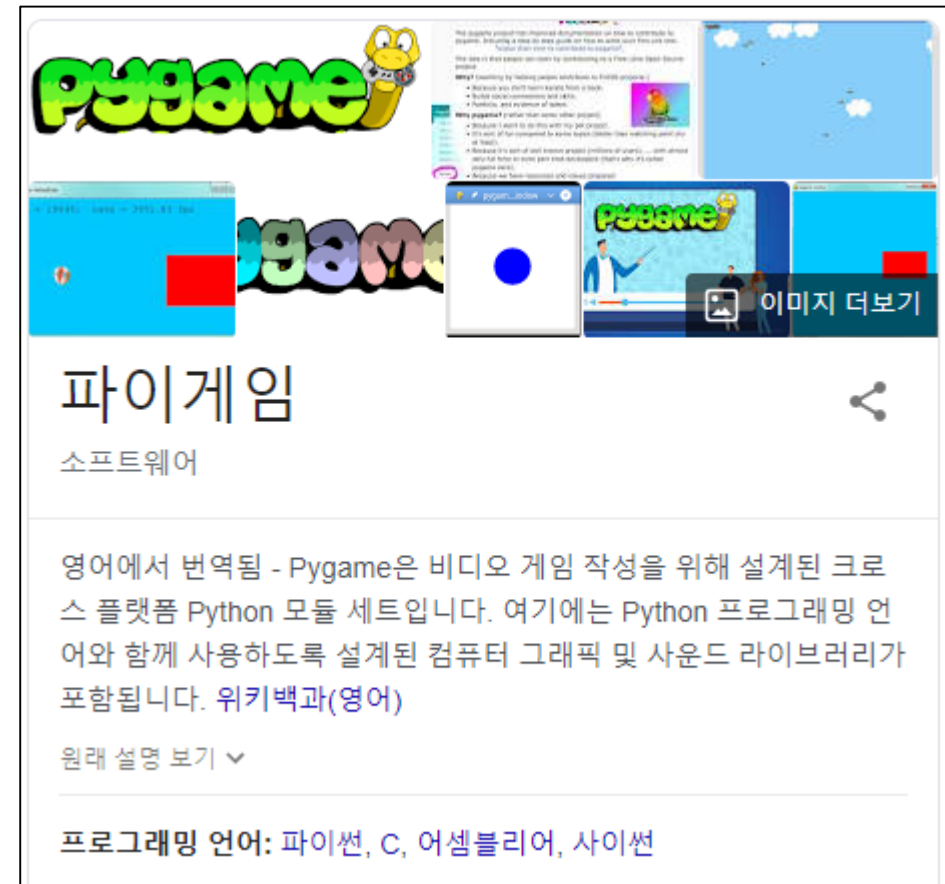
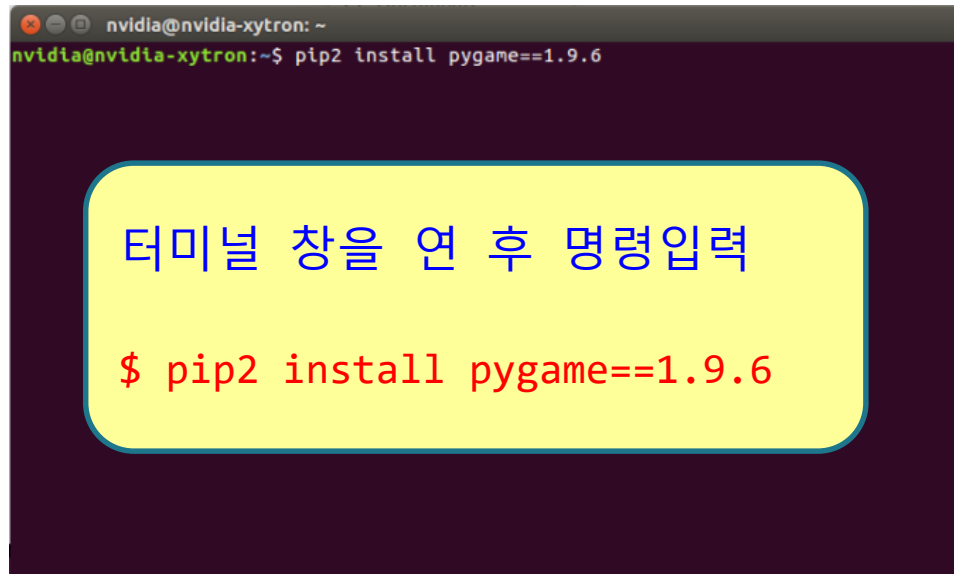
(pip 설치 방법)

```
$ sudo apt update
```

```
$ sudo apt-get install python-pip
```

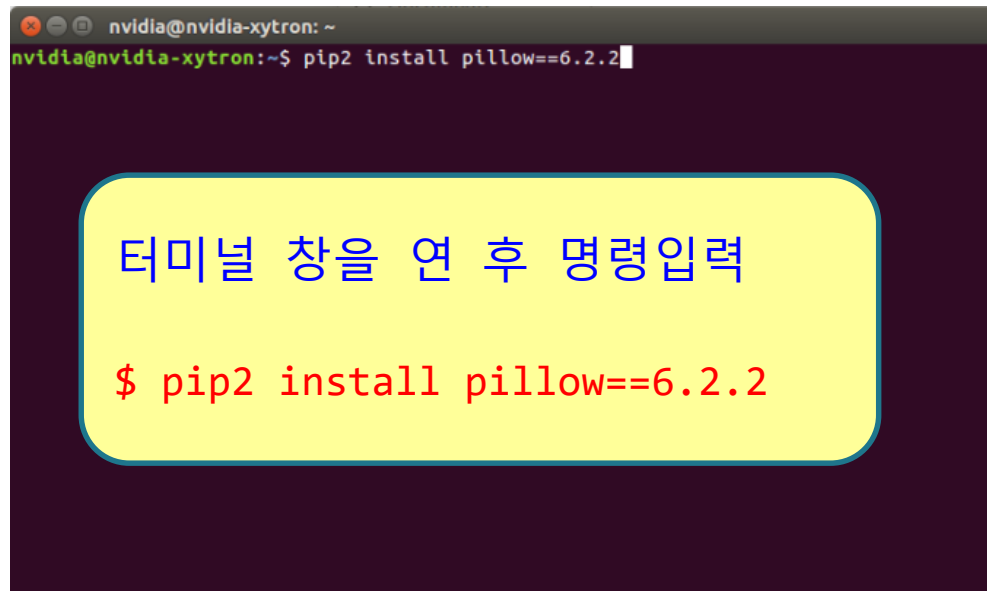
# pygame 설치

- pygame 1.9.6 버전 설치
  - `$ pip2 install pygame==1.9.6`



# pillow 설치

- pillow 6.2.2 버전 설치
  - \$ pip2 install pillow==6.2.2

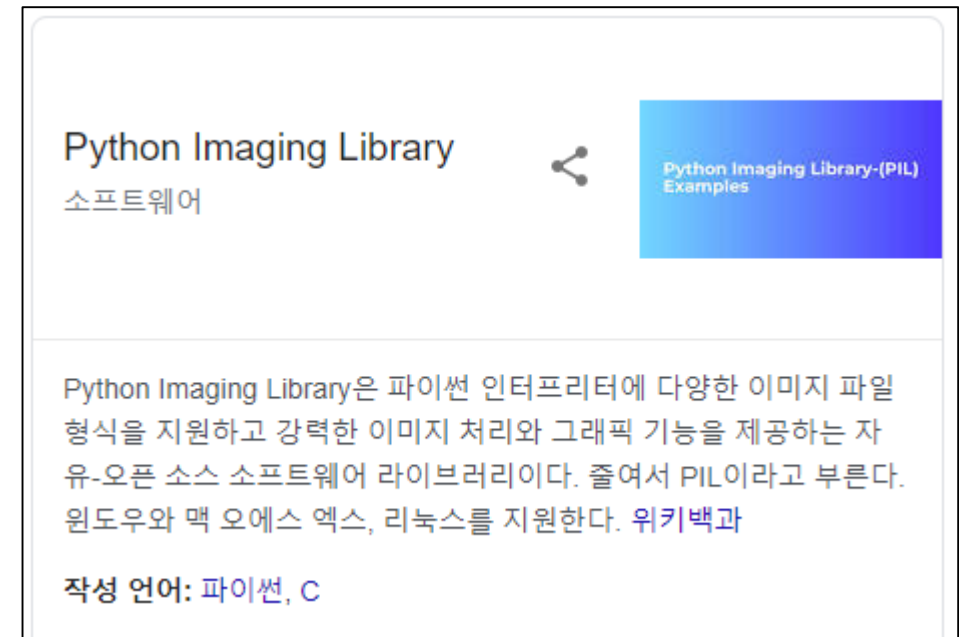


A terminal window with a dark background. The prompt is 'nvidia@nvidia-xytron: ~'. The command entered is 'pip2 install pillow==6.2.2'. A yellow rounded rectangle is overlaid on the terminal, containing the text '터미널 창을 연 후 명령입력' and the command '\$ pip2 install pillow==6.2.2'.

```
nvidia@nvidia-xytron: ~  
nvidia@nvidia-xytron:~$ pip2 install pillow==6.2.2
```

터미널 창을 연 후 명령입력

\$ pip2 install pillow==6.2.2



A screenshot of the Python Imaging Library (PIL) page. The page has a white background with a blue header. The header contains the text 'Python Imaging Library' and '소프트웨어'. To the right of the header is a blue button with the text 'Python Imaging Library-(PIL) Examples'. Below the header is a paragraph of text: 'Python Imaging Library은 파이썬 인터프리터에 다양한 이미지 파일 형식을 지원하고 강력한 이미지 처리와 그래픽 기능을 제공하는 자유-오픈 소스 소프트웨어 라이브러리이다. 줄여서 PIL이라고 부른다. 윈도우와 맥 오에스 엑스, 리눅스를 지원한다. 위키백과'. At the bottom, it says '작성 언어: 파이썬, C'.

Python Imaging Library  
소프트웨어

Python Imaging Library-(PIL)  
Examples

Python Imaging Library은 파이썬 인터프리터에 다양한 이미지 파일 형식을 지원하고 강력한 이미지 처리와 그래픽 기능을 제공하는 자유-오픈 소스 소프트웨어 라이브러리이다. 줄여서 PIL이라고 부른다. 윈도우와 맥 오에스 엑스, 리눅스를 지원한다. 위키백과

작성 언어: 파이썬, C

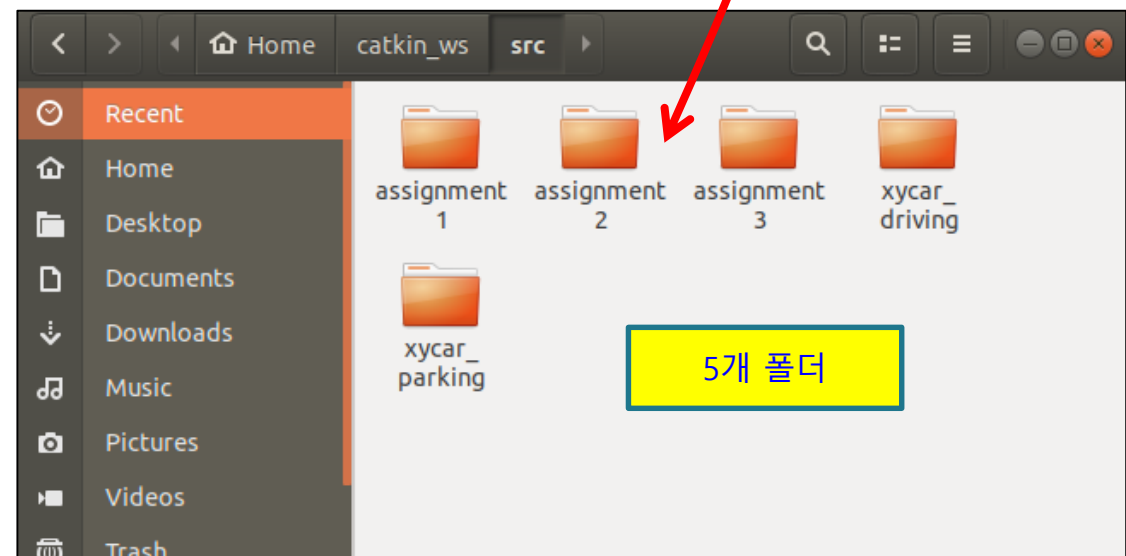
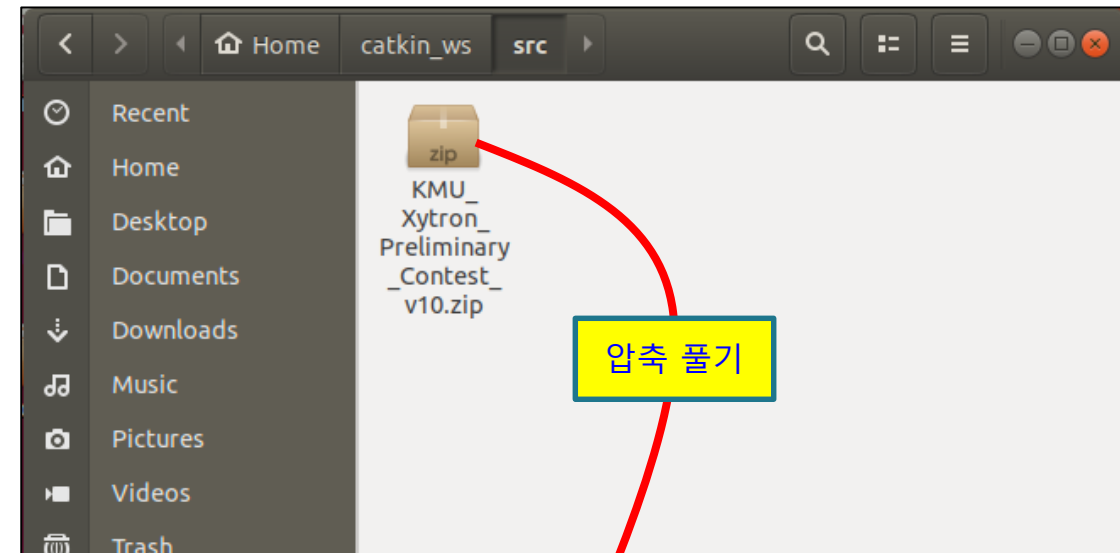
# 과제 수행 준비

---

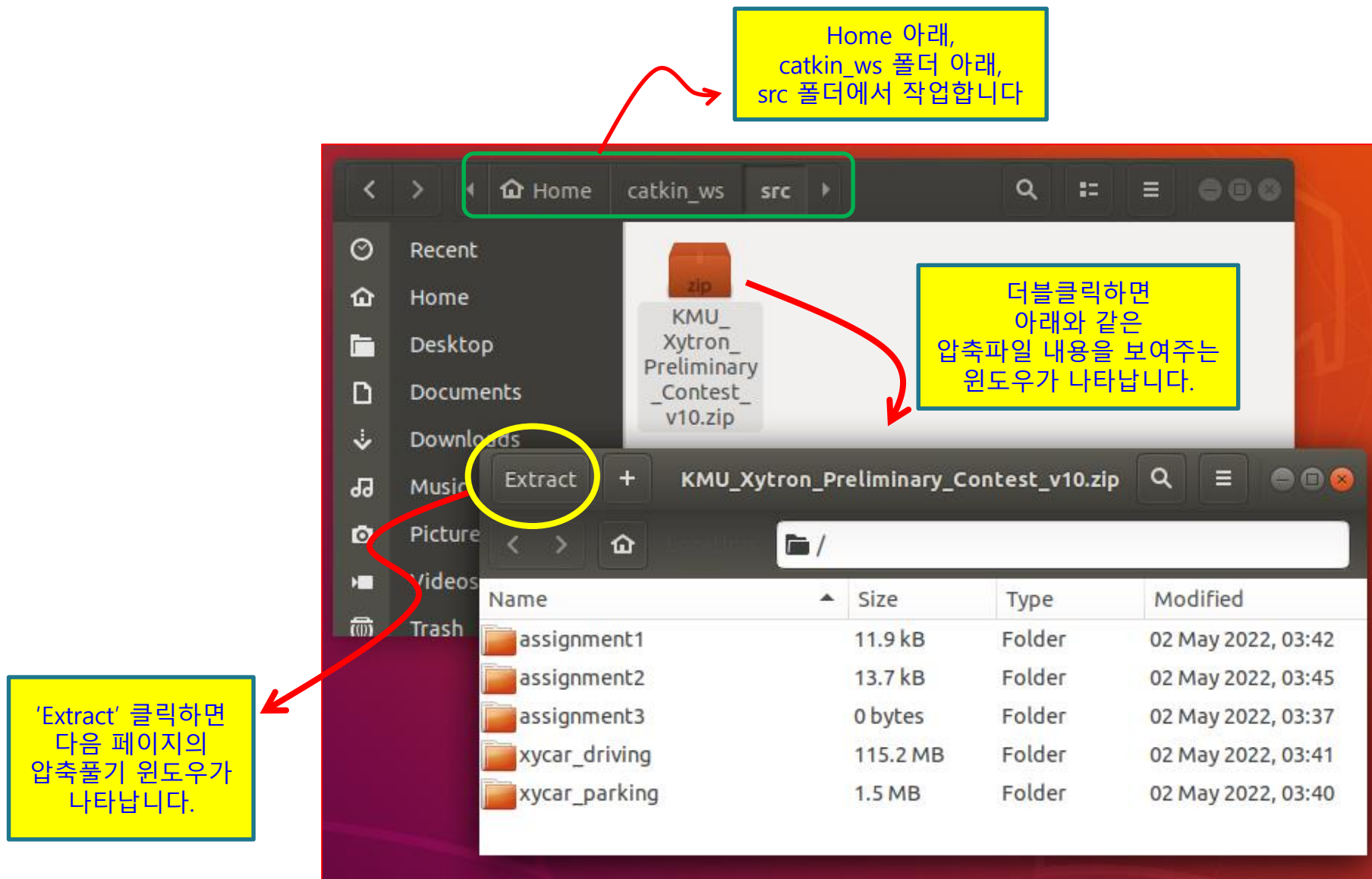
ROS 워크스페이스에  
배포된 압축 파일 풀고 빌드하기

# 배포 파일 압축 풀기

- 배포 파일
  - KMU\_Xytron\_Preliminary\_Contest\_v10.zip
- 배포 파일 복사
  - ~/catkin\_ws/src 폴더에 복사합니다.
- 압축 풀기
  - \*\*\*.zip 파일을 더블클릭 해서 압축을 풉니다.
- 확인
  - 모두 5개의 폴더가 생성됩니다.
  - \$ cm (빌드합니다)

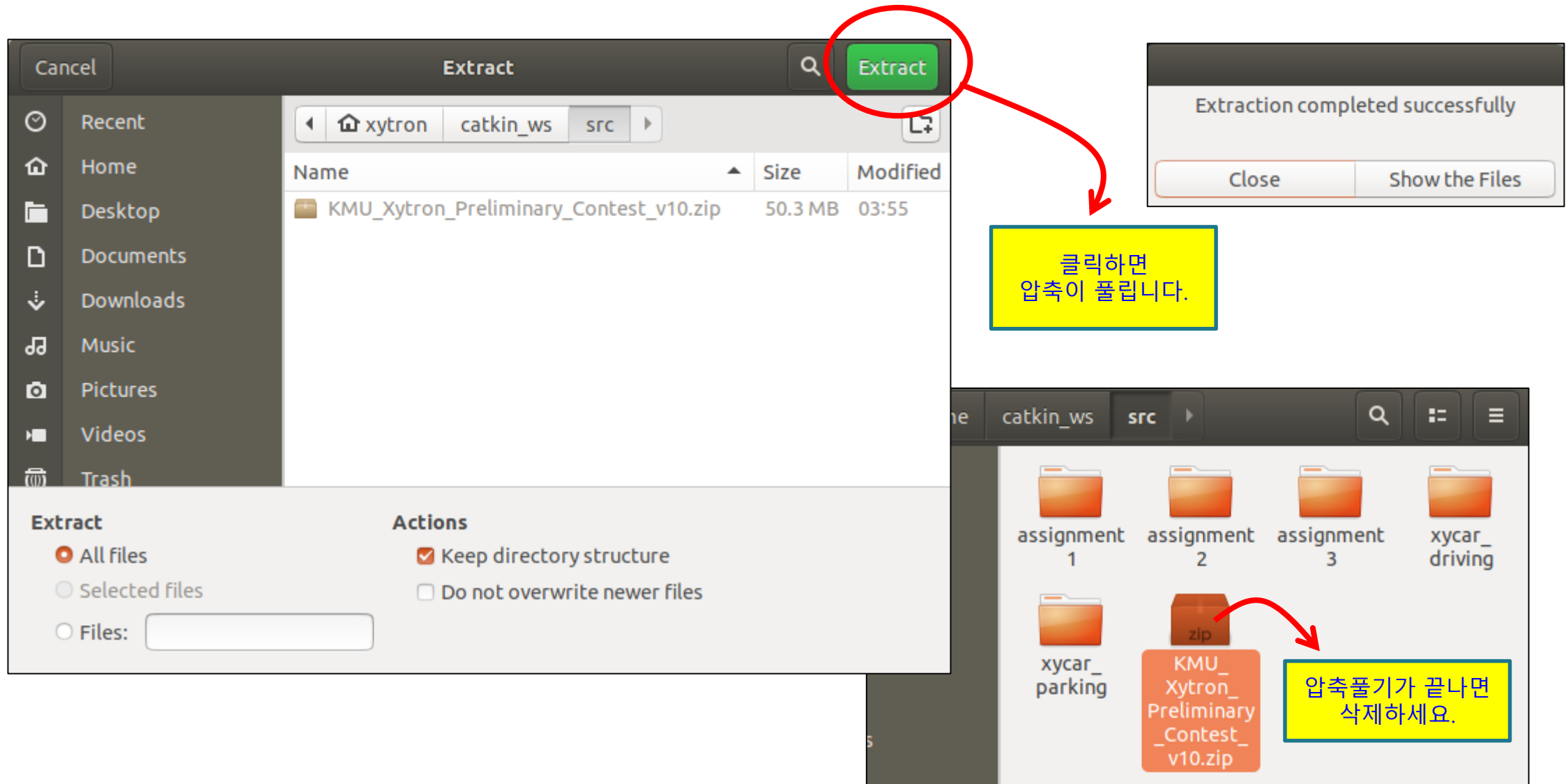


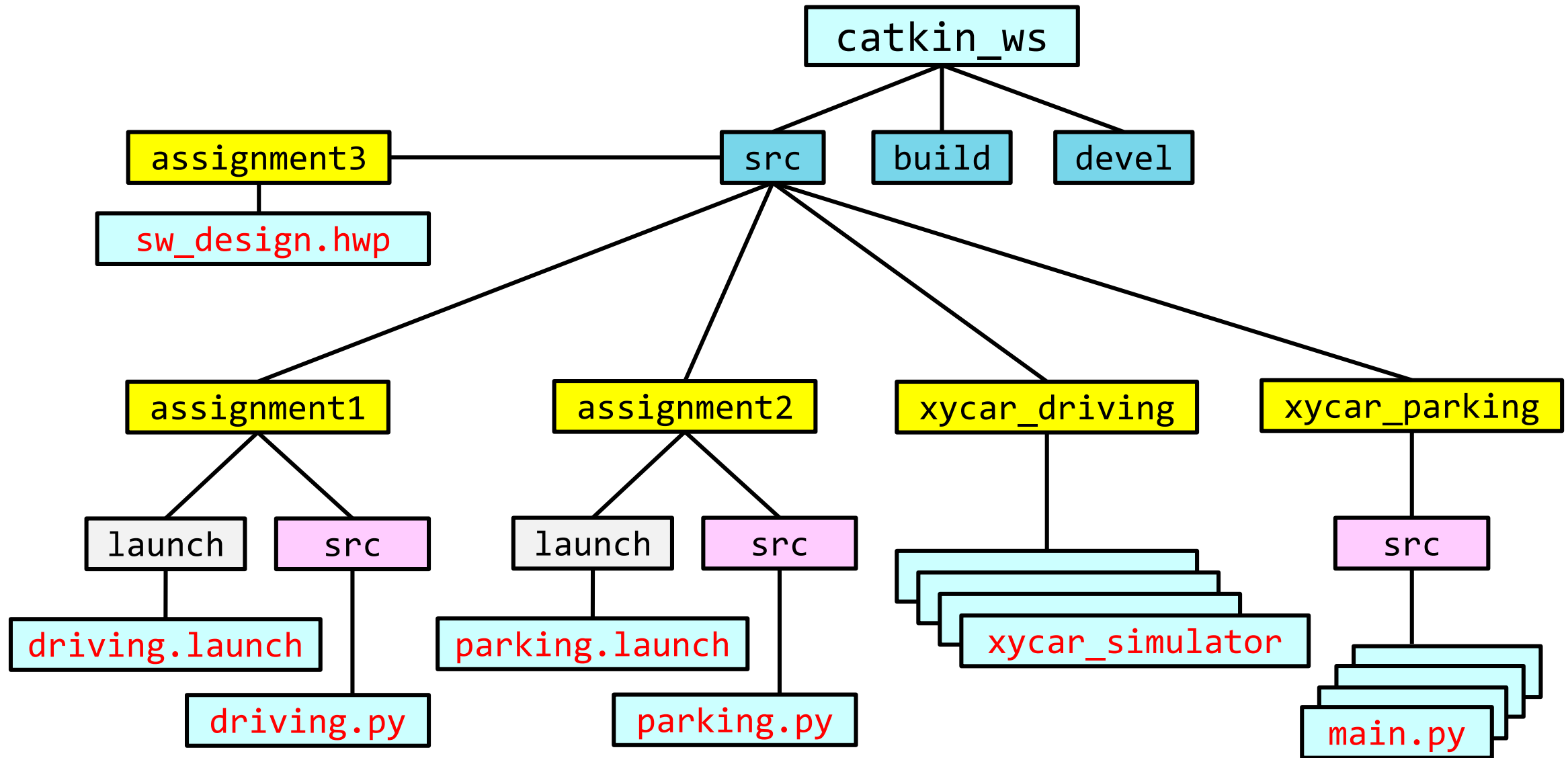
# 배포 파일 압축 풀기





# 배포 파일 압축 풀기





# 과제 #1

---

차선을 벗어나지 않고 목적지까지 주행하는  
자율주행SW 구현

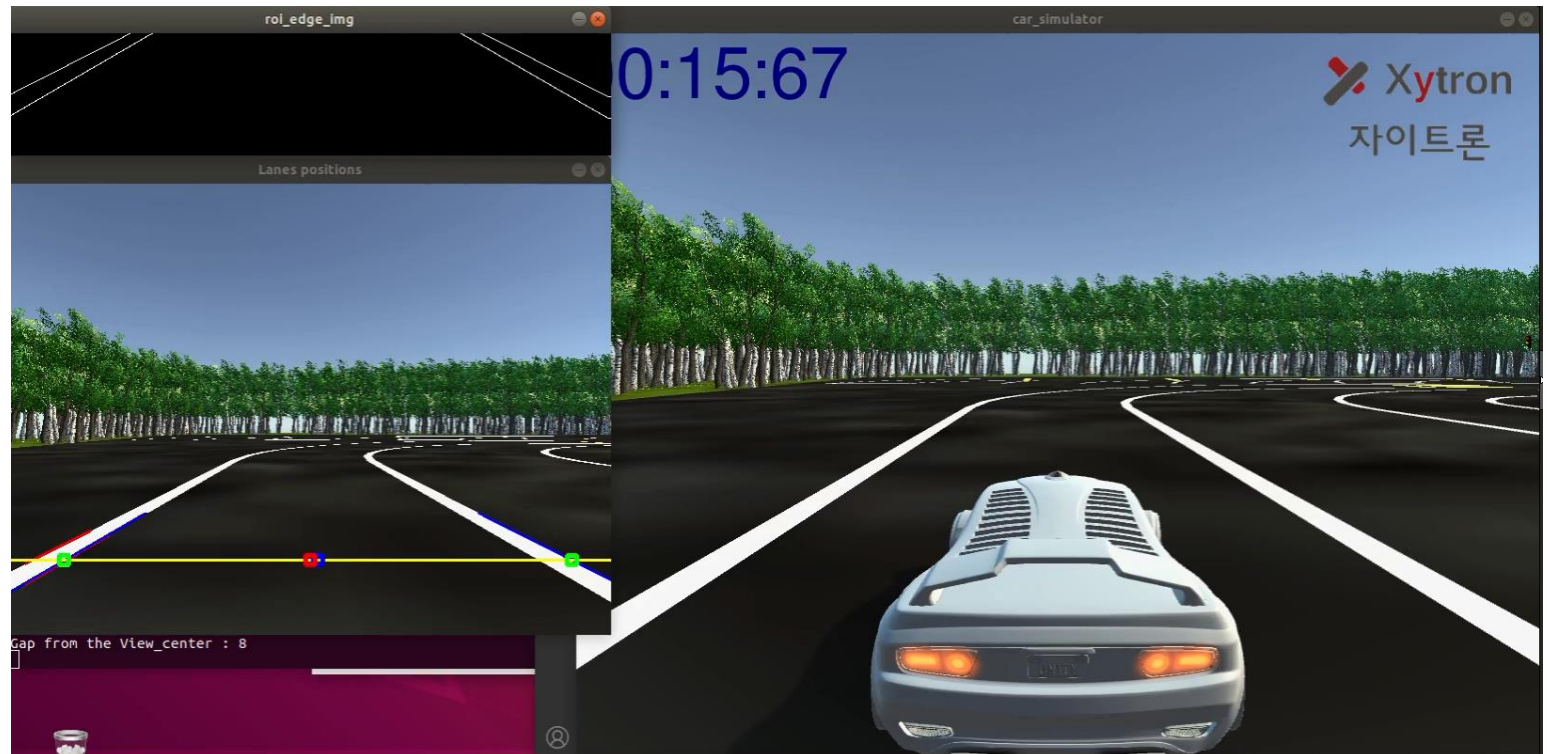
# 과제 설명

- 시뮬레이터상에서 자동차가 차선을 벗어나지 않고 주행하게 만드는 것이 목표입니다.
- 카메라로부터 640x480 크기의 영상을 ROS 토픽으로 받을 수 있습니다.
- 차량 속도와 조향각을 제어하는 ROS 토픽을 발행할 수 있습니다.

시뮬레이터가 제공하는  
카메라 이미지를 영상처리하여  
차선의 위치를 찾고,

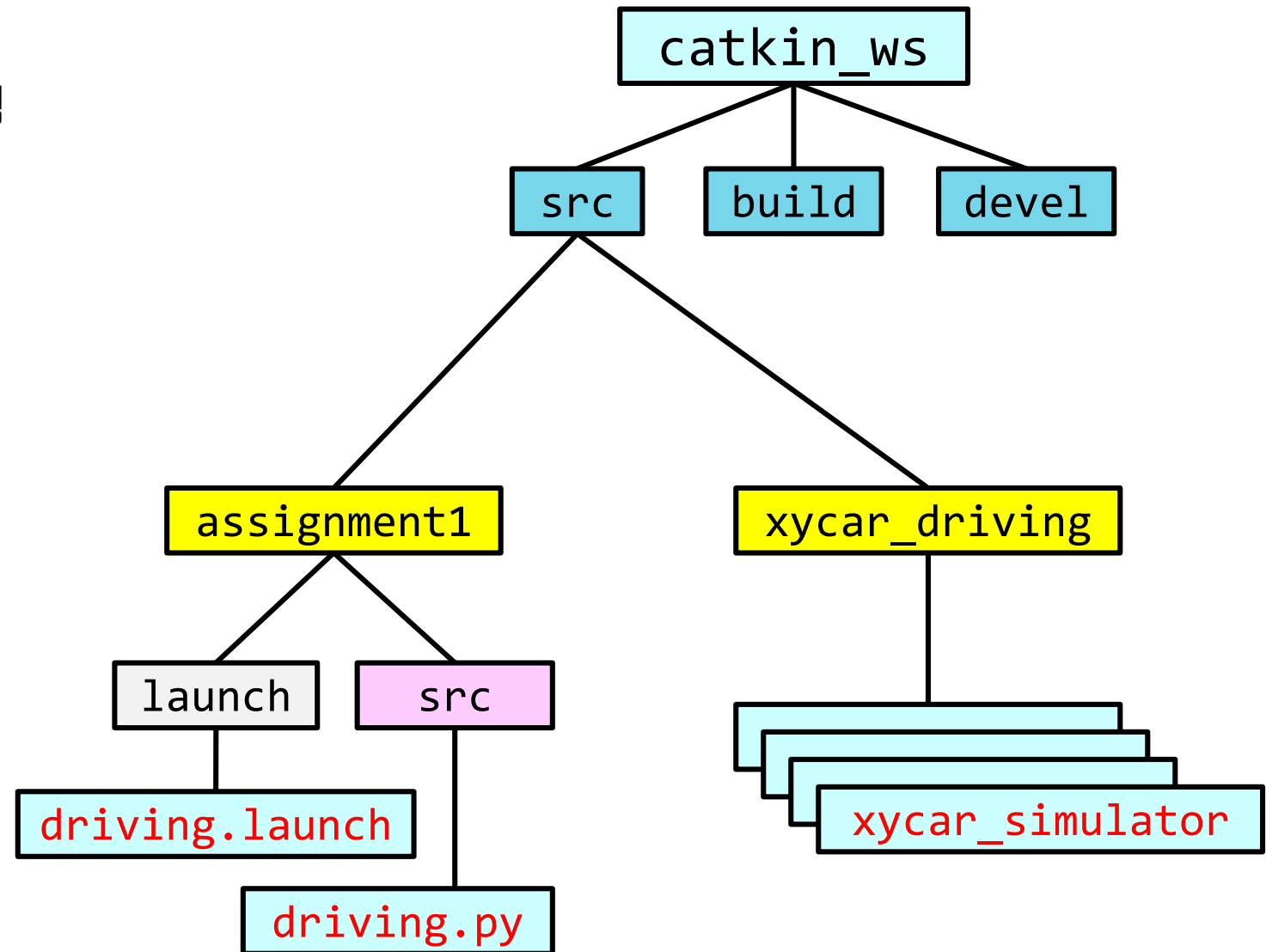
이를 계산해 차선을 벗어나지  
않고 주행할 수 있는 조향각을  
찾아내고,

모터제어 토픽을 시뮬레이터로  
보내 차량이 차선을 벗어나지  
않도록 운전하면 됩니다.

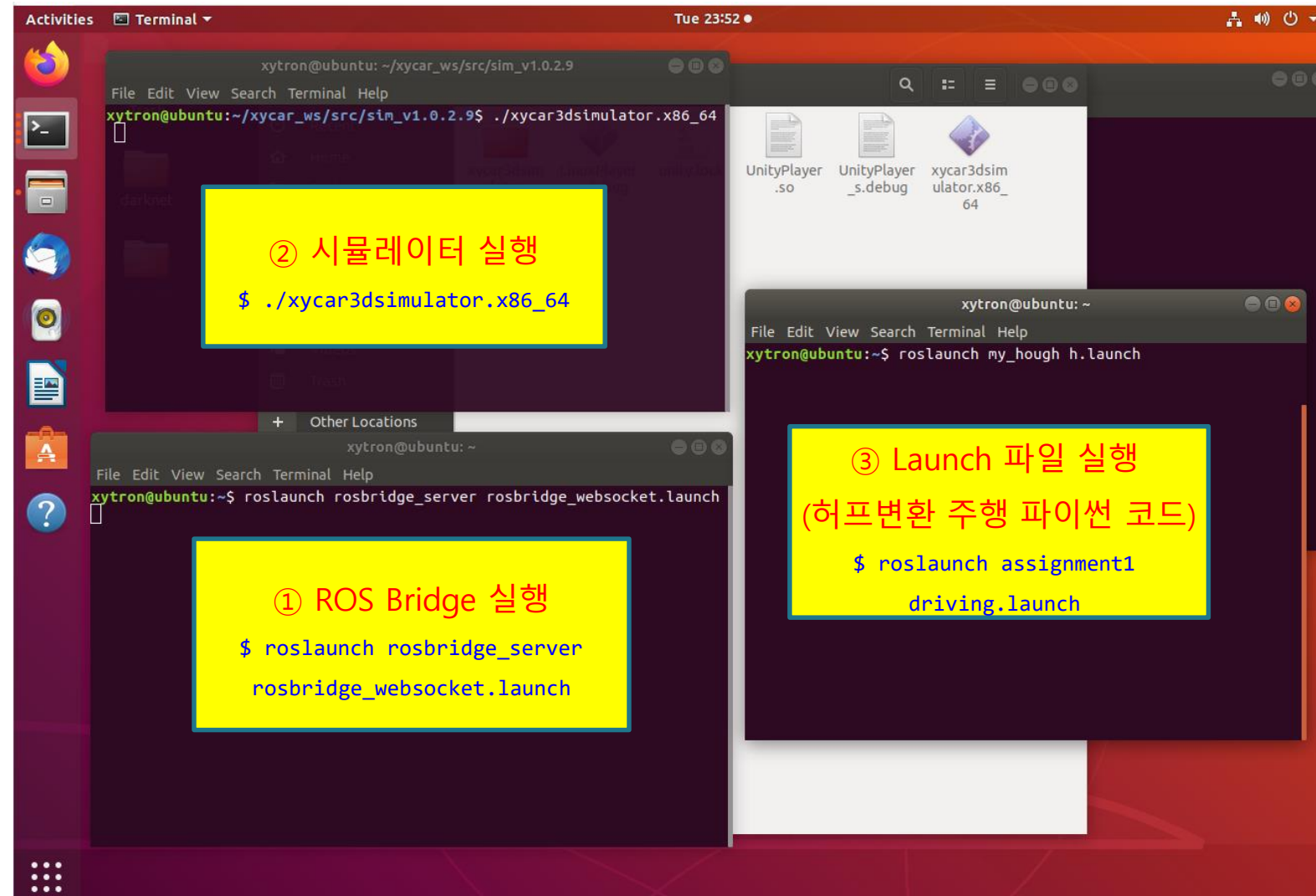


# 과제 수행에 필요한 폴더와 파일들

- driving.py
  - 차량을 운전하는 파이썬 프로그램
- driving.launch
  - ROS 런치파일
- xycar\_simulator
  - 시뮬레이터 실행 파일



- 터미널 3개 열어서 선수대로 하나씩 실행합니다. (다음 페이지에서 상세히 설명합니다)



# 실행 방법

- 터미널 3개 열어서 순서대로 하나씩 실행합니다.

## ① ROS Bridge 실행

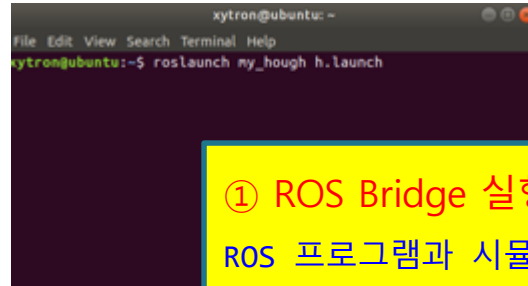
```
$ roslaunch rosbridge_server rosbridge_websocket.launch
```

## ② 시뮬레이터 실행 (파일이 있는 폴더로 가서 실행)

```
$ ./xycar3dsimulator.x86_64
```

## ③ Launch 파일 실행 (자신이 작성한 자율주행 파이썬 코드)

```
$ roslaunch my_hough h_drive_sim.launch
```



## ① ROS Bridge 실행

ROS 프로그램과 시뮬레이터를 연결해 준다.



## ③ Launch 파일 실행

영상처리를 통해 차선을 찾고 조향각을 결정해서 모터제어 토픽을 발행한다.



## ② 시뮬레이터 실행

카메라 영상을 ROS 토픽으로 외부로 보낸다. / 외부로부터 ROS 모터제어 토픽을 받아서 차량을 움직인다. / 차선준수 여부를 모니터링해서 재출발 여부를 결정하고 조치한다.

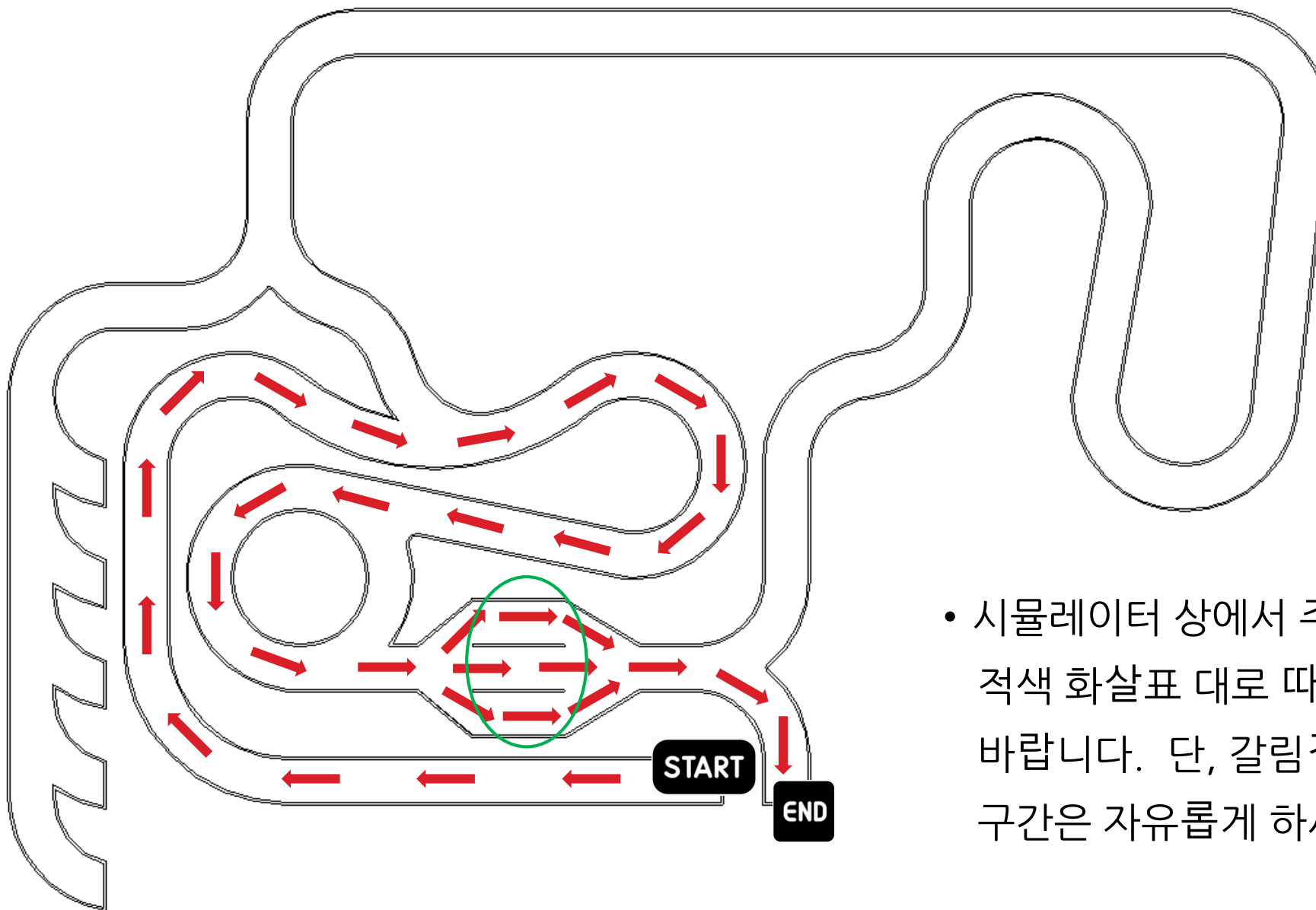


# 주행 시뮬레이터 사용법

- 처음 시작은 자동주행 모드입니다. (파이썬 프로그램에 의해 차량이 움직입니다)
- 스페이스바를 누르면 수동조종 주행모드로 들어갑니다. (사용자가 상하좌우 키로 운전)
- 다시 스페이스바를 누르면 자동주행 모드로 들어갑니다. (프로그램에 의해 운전)



# 주행 시뮬레이터 경로



- 시뮬레이터 상에서 주행 경로는 적색 화살표 대로 따라 주시기 바랍니다. 단, 갈림길 (녹색 표시) 구간은 자유롭게 하세요)

# 시뮬레이터가 제공하는 카메라 영상 데이터 토픽

- /usb\_cam/image\_raw 토픽
  - 타입 = sensor\_msgs/Image
  - 구성 =

```
std_msgs/Header header
```

```
uint32 seq
```

```
time stamp
```

```
string frame_id
```

```
uint32 height
```

```
uint32 width
```

```
(중간 생략)
```

```
uint8[ ] data
```

헤더  
(시퀀스번호와 시간, 아이디  
정보를 담는다)

영상의 높이/폭 정보

영상데이터 저장공간

# 시뮬레이터로 보내야 하는 자동차 속도/조향각 제어 토픽

- /xycar\_motor 토픽

- 타입 = xycar\_msgs/xycar\_motor
- 구성 =

```
std_msgs/Header header
uint32 seq
time stamp
string frame_id
```

헤더  
(시퀀스번호와 시간, 아이디  
정보를 담는다)

```
int32 angle
```

핸들 조향각 (-50~50)

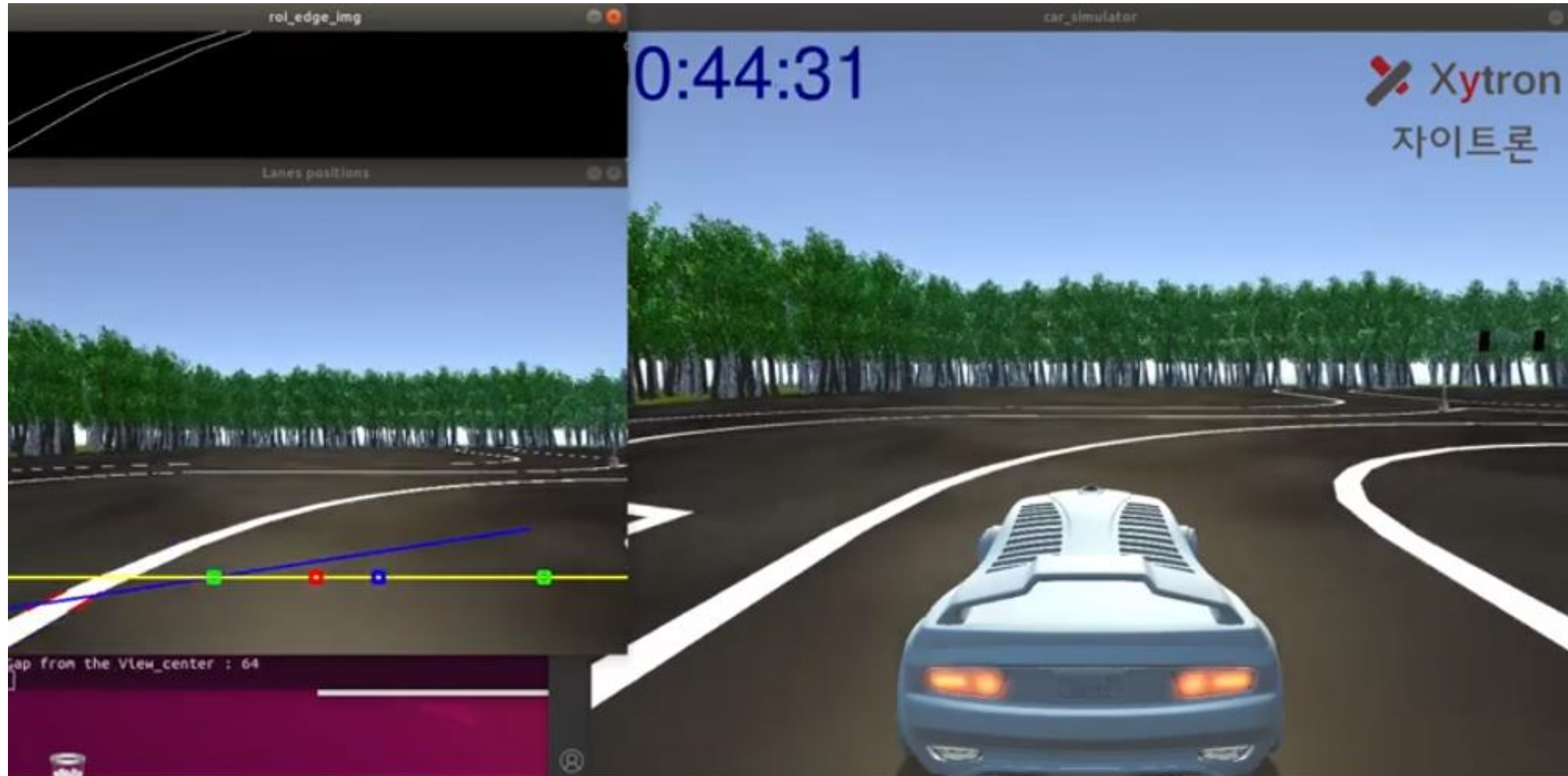
angle : 좌회전 MAX(-50) ~ 직진 (0) ~ 우회전 MAX (50)

```
int32 speed
```

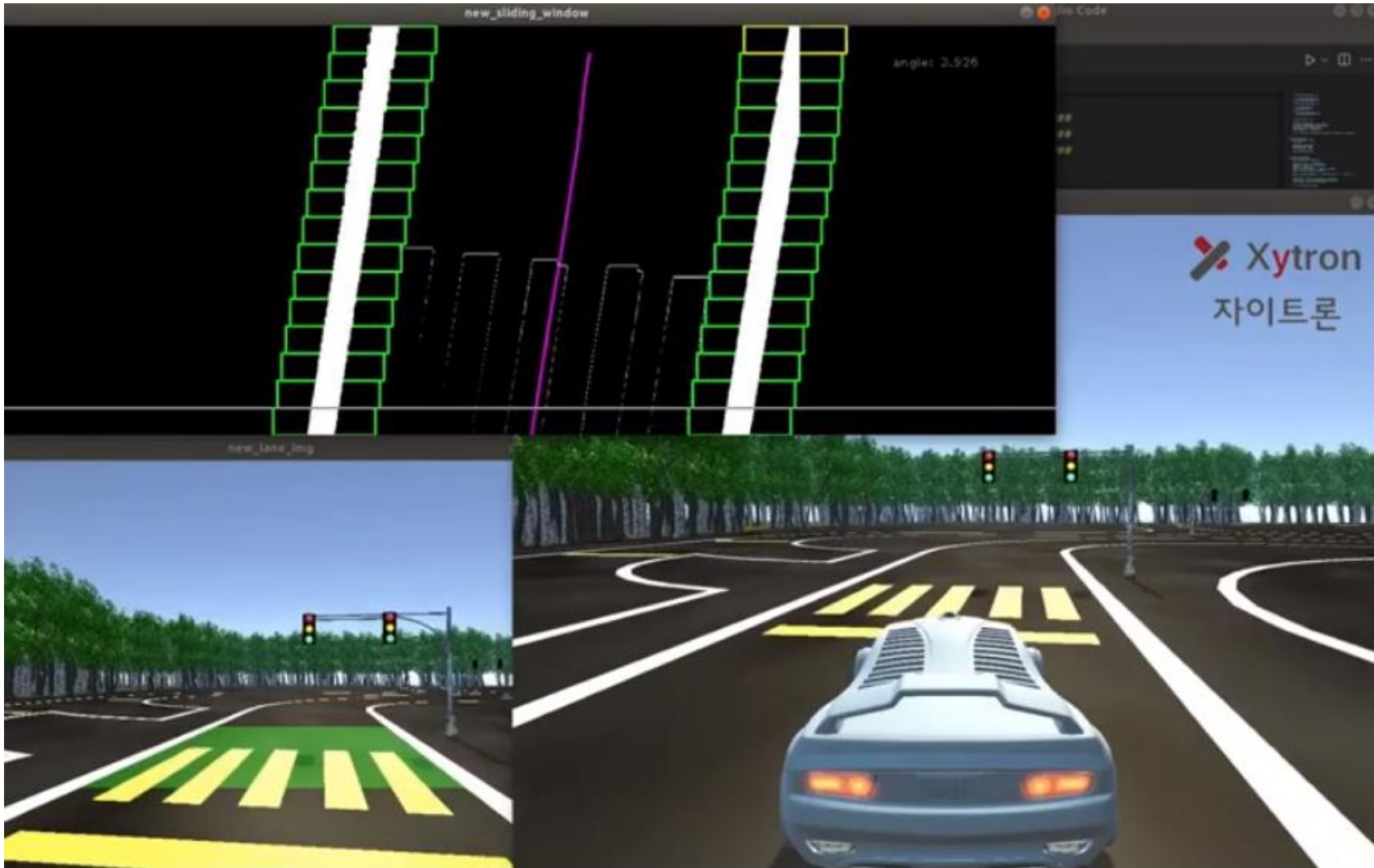
차량 속도 (-50~50)

speed : 후진 MAX(-50) ~ 정지 (0) ~ 전진 MAX (50)

## 예제 동영상 (1)



## 예제 동영상 (2)



# 결과물 제출

---

- 동영상 제출
  - 자동차가 주행하는 시뮬레이터 화면을 동영상 캡처툴로 (동영상 파일)로 만들어 제출하세요.
  - 동영상 파일이름은 “driving.mp4” 로 하세요.
- 소스코드 파일 제출
  - driving.py
  - 소스코드에 한글로 코멘트(설명) 상세히 붙여서 제출하세요.
  - 설명이 상세할수록 좋습니다.
  - 함수와 블록 단위로 설명하는 코멘트도 넣고, 한 줄 한 줄 개별코드를 설명하는 코멘트로 넣으세요.
  - 여러 팀원들이 힘을 합쳐 함께 작성하고, 디버깅하고, 수정하고, 개선한 코드라면 당연히 소스코드에 상세한 코멘트가 많이 들어갈 수 밖에 없겠지요. 코멘트 작성이 아무런 부담이 안 될 겁니다.

# 과제 #2

---

AR태그를 이용하여 주차구역에  
정확하게 주차하는 자율주차SW 구현



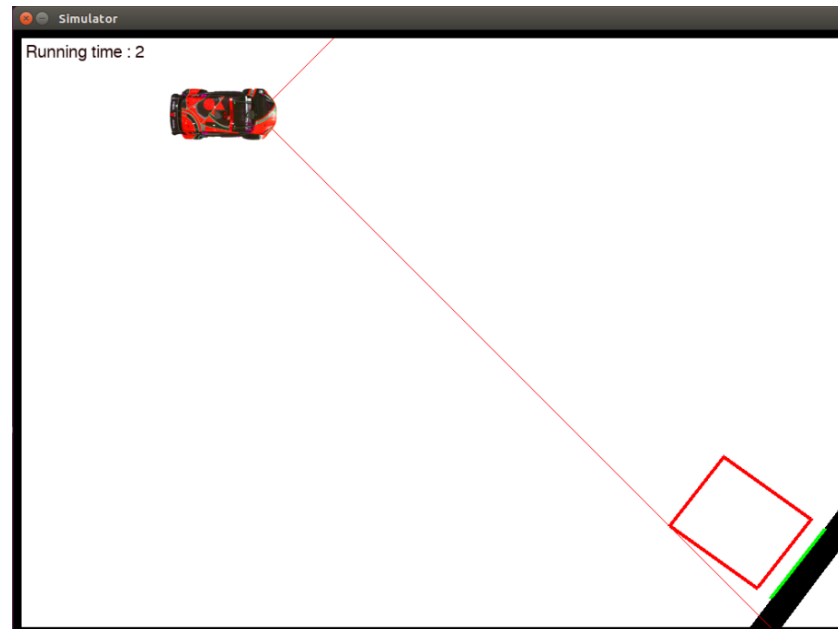
# 과제 설명

- 시뮬레이터상에서 자동차가 주차구역에 정확하게 주차하게 만드는 것이 목표입니다.
- AR 센서로부터 ARE태그의 거리와 자세정보를 ROS 토픽으로 받을 수 있습니다.
- 차량 속도와 조향각을 제어하는 ROS 토픽을 발행할 수 있습니다.
  - 주차구역에 정확하게 주차하면 주차구역이 녹색으로 바뀝니다. 주차 성공여부를 알 수 있습니다.

시뮬레이터가 제공하는  
ARE태그의 거리와 자세 정보를  
분석하여 주차구역까지의 거리  
와 차량과의 각도를 알아내고,

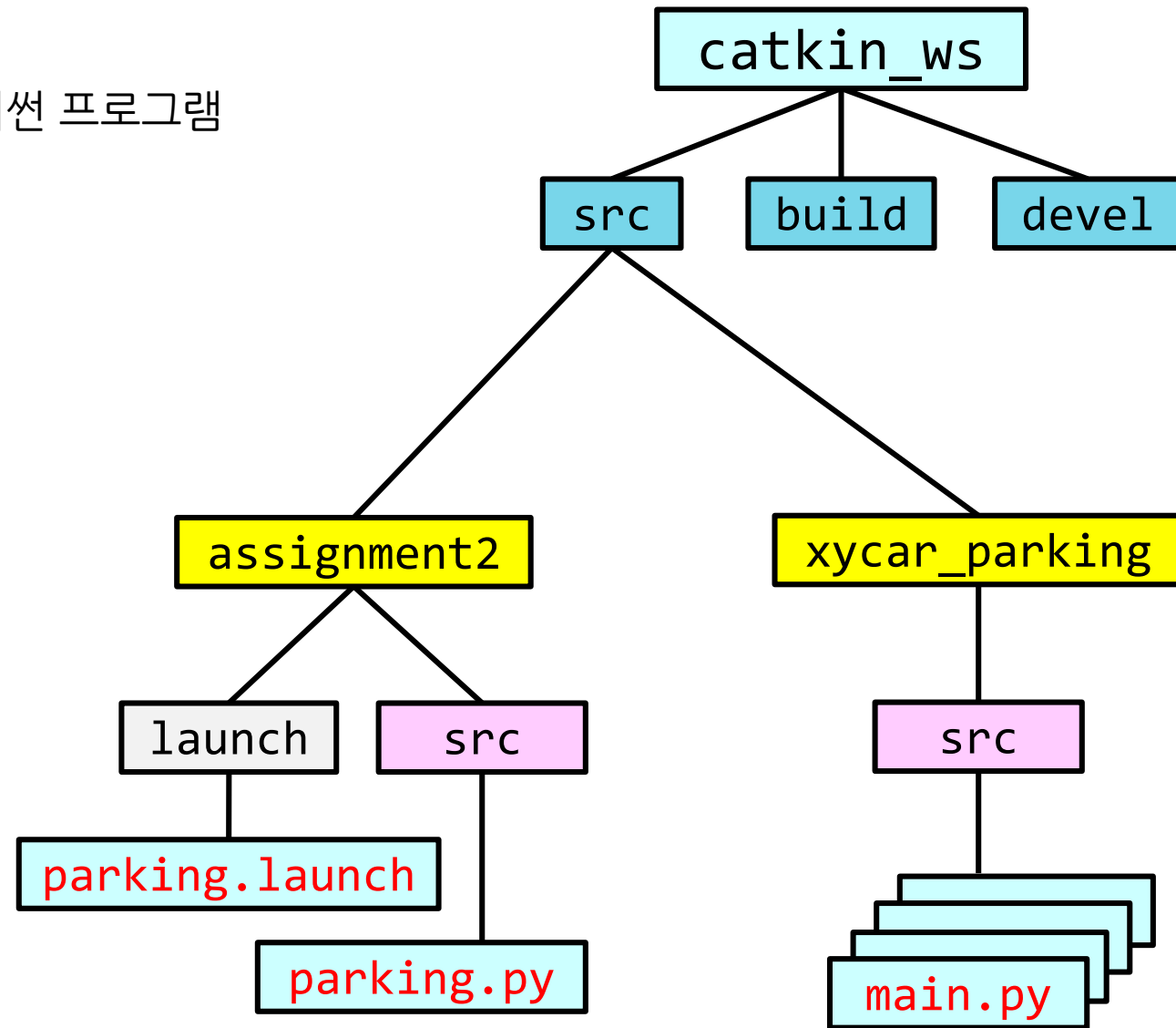
주차가 가능한 적절한  
주행 경로와 방법을 찾아내고,

모터제어 토픽을 시뮬레이터로  
보내 차량이 주차구역에  
정확하게 주차할 수 있도록  
운전하면 됩니다.



# 과제 수행에 필요한 폴더와 파일들

- parking.py
  - 차량을 운전하여 주차시키는 파이썬 프로그램
- parking.launch
  - ROS 런치파일
- main.py
  - 시뮬레이터 실행 파일



# 실행 방법

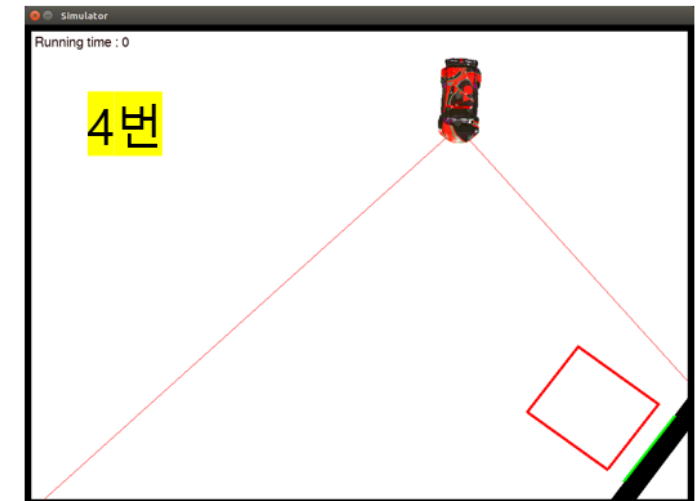
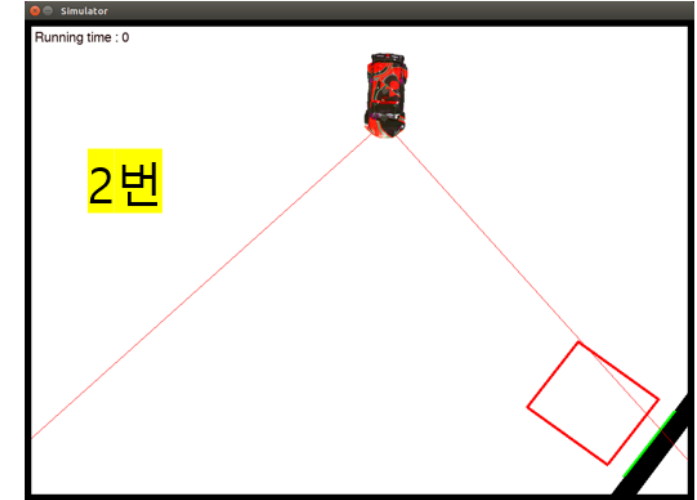
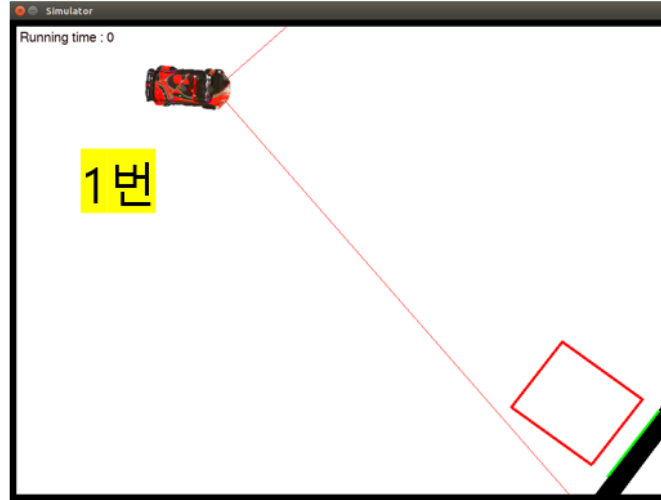
- 터미널 1개 열어서 ROS 런치파일을 실행합니다.
  - `$ roslaunch assignment2 parking.launch`

parking.launch

```
<launch>
  <node name="simulator" pkg="xycar_parking" type="main.py" output="screen"/>
  <node name="driver" pkg="assignment2" type="parking.py" output="screen" />
</launch>
```

# 시뮬레이터 사용법

- 출발위치는 총 4가지가 있으며, 시뮬레이터 창에서 키보드 숫자키 1~4를 누르면 출발위치가 바뀜.
- 주차구역 안에 정확히 안착하면 주차구역 선이 녹색으로 바뀜 (주차 성공)
- 주행하는 도중에 키보드 1~4를 누르면 해당 출발위치로 강제 이동됨. 디버깅시 편리함.



# 시뮬레이터가 제공하는 AR 태그 토픽

- /ar\_pose\_marker 토픽

- 타입 = ar\_track\_alvar\_msgs/AlvarMarkers

- 구성 =

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
```

```
uint32 id
uint32 confidence
```

(옆에서 계속)

```
geometry_msgs/PoseStamped pose
  std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
  geometry_msgs/Pose pose
  geometry_msgs/Point position
```

float64 x

float64 y

float64 z

```
geometry_msgs/Quaternion orientation
```

float64 x

float64 y

float64 z

float64 w

위치 정보 (X,Y 좌표값)

각도 정보 (Yaw 좌우회전 값)

# 시뮬레이터가 제공하는 AR 태그 토픽

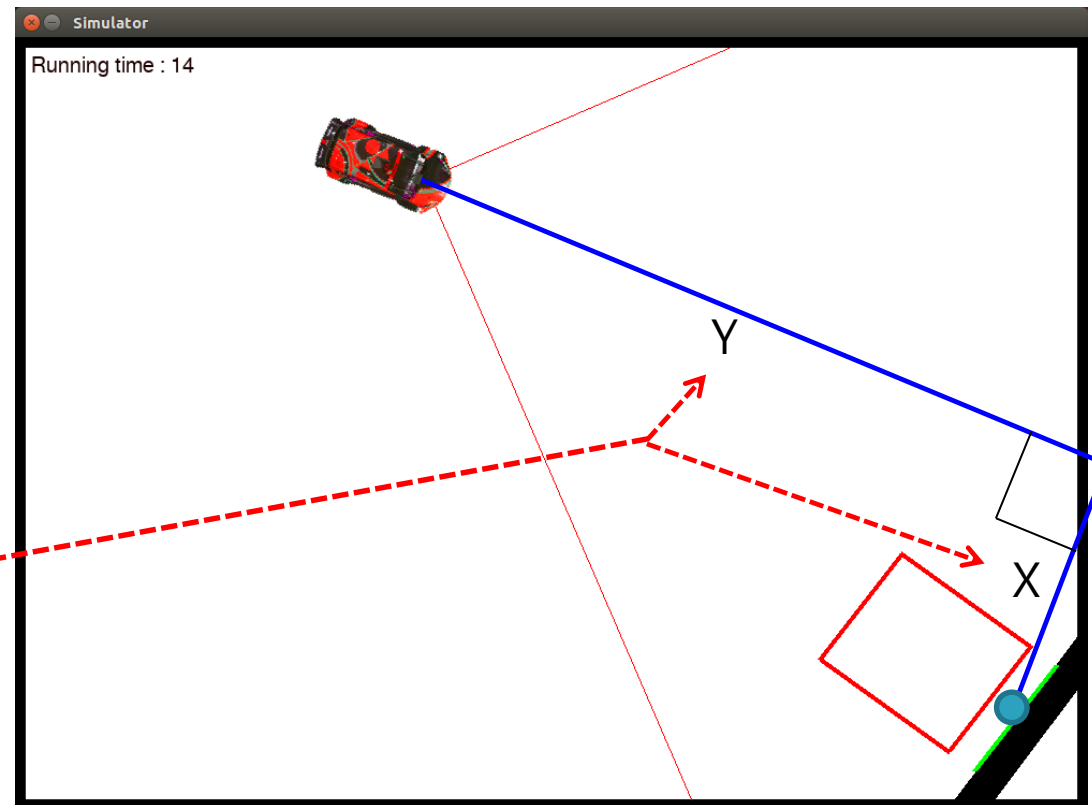
- 토픽 이름 = /ar\_pose\_marker
  - 카메라 화면에서 AR 태그가 어떤 위치에 있는지, 어떤 자세를 하고 있는지에 대한 정보를 /ar\_pose\_marker 토픽에 담아 전송한다.
  - 실제와는 달리 시뮬레이터가 보내는 정보의 거리 단위는 m(미터)가 아니고 pixel(픽셀) 이다.
  - 자세 정보는 쿼터니언으로 표시되므로, 오일러 방식으로 바꾸는 작업이 필요하다.

```
hscho@sparc: ~  
---  
header:  
  seq: 5773  
  stamp:  
    secs: 0  
    nsecs: 0  
  frame_id: ''  
markers:  
  -  
    header:  
      seq: 0  
      stamp:  
        secs: 1618466004  
        nsecs: 358340024  
      frame_id: "usb_cam"  
    id: 1  
    confidence: 0  
    pose:  
      header:  
        seq: 0  
        stamp:  
          secs: 0  
          nsecs: 0  
        frame_id: ''  
      pose:  
        position:  
          x: 335.838067754  
          y: 692.156433284  
          z: 0.0  
        orientation:  
          x: 0.0  
          y: 0.0  
          z: -0.0422303717216  
          w: 0.999107899931
```

# 시뮬레이터가 제공하는 AR 태그 토픽 - /ar\_pose\_marker

- 3차원 공간에서의 좌표값 (x, y, z)
  - pose.pose.position (x, y, z)
  - 픽셀 단위의 거리 정보

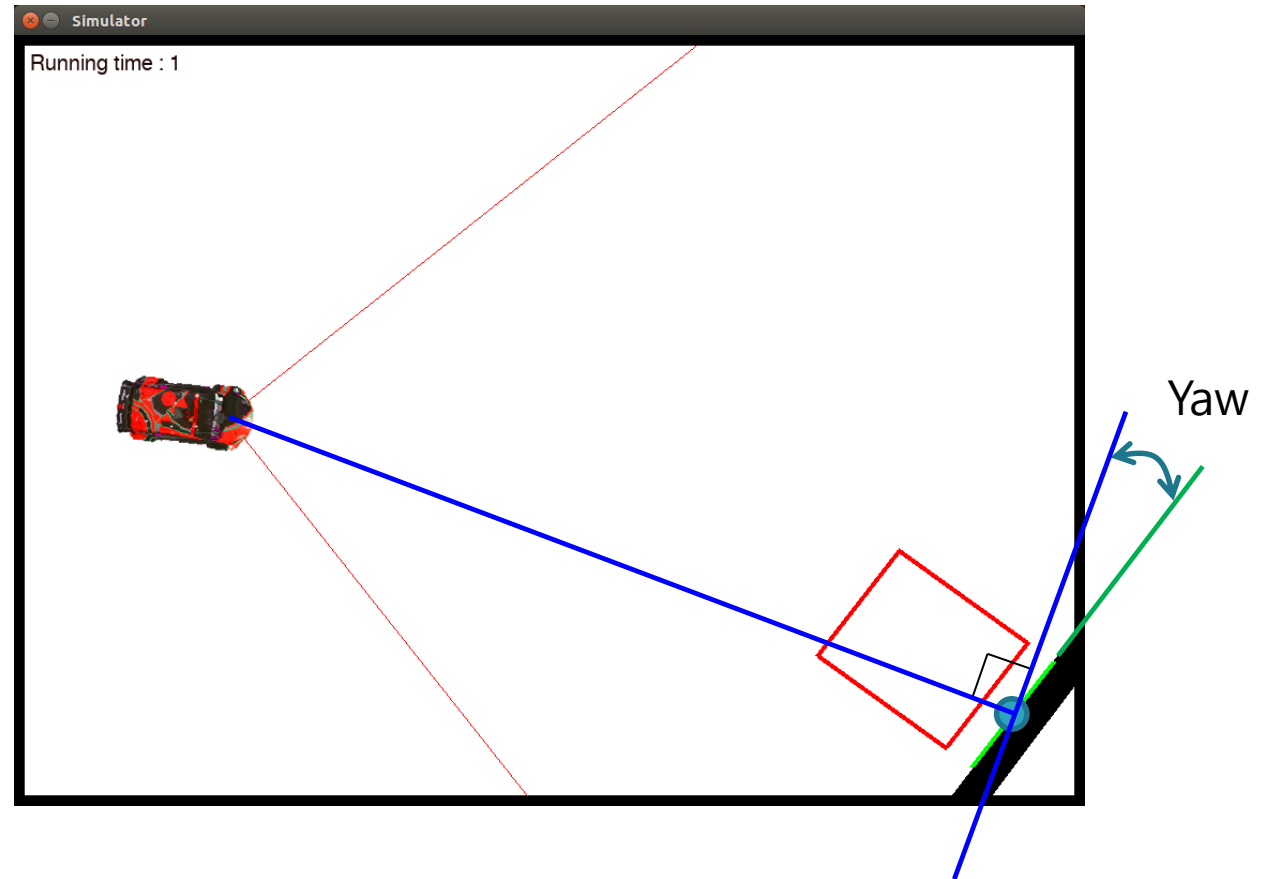
```
sungmin@machine: ~/
=====
roll  : 0.0
pitch : 0.0
yaw   : -4.6
x     : 368.0
y     : 688.0
z     : 0.0
('callback:', -50)
=====
roll  : 0.0
pitch : 0.0
yaw   : -5.0
x     : 310.0
y     : 694.0
z     : 0.0
```



# 시뮬레이터가 제공하는 AR 태그 토픽 - /ar\_pose\_marker

- 자세 정보값 (Roll, Pitch, Yaw)
  - pose.pose.orientation (x, y, z, w)
  - 하늘에서 보는 자동차의 진행 각도

```
sungmin@machine: ~/
=====
roll  : 0.0
pitch : -0.0
yaw   : 16.3
x : 203.0
y : 761.0
z : 0.0
('callback:', -50)
=====
roll  : 0.0
pitch : -0.0
yaw   : 16.2
x : 140.0
y : 754.0
z : 0.0
```





# 시뮬레이터로 보내야 하는 자동차 속도/조향각 제어 토픽

- /xycar\_motor 토픽

- 타입 = xycar\_msgs/xycar\_motor
- 구성 =

```
std_msgs/Header header
uint32 seq
time stamp
string frame_id
```

헤더  
(시퀀스번호와 시간, 아이디  
정보를 담는다)

```
int32 angle
```

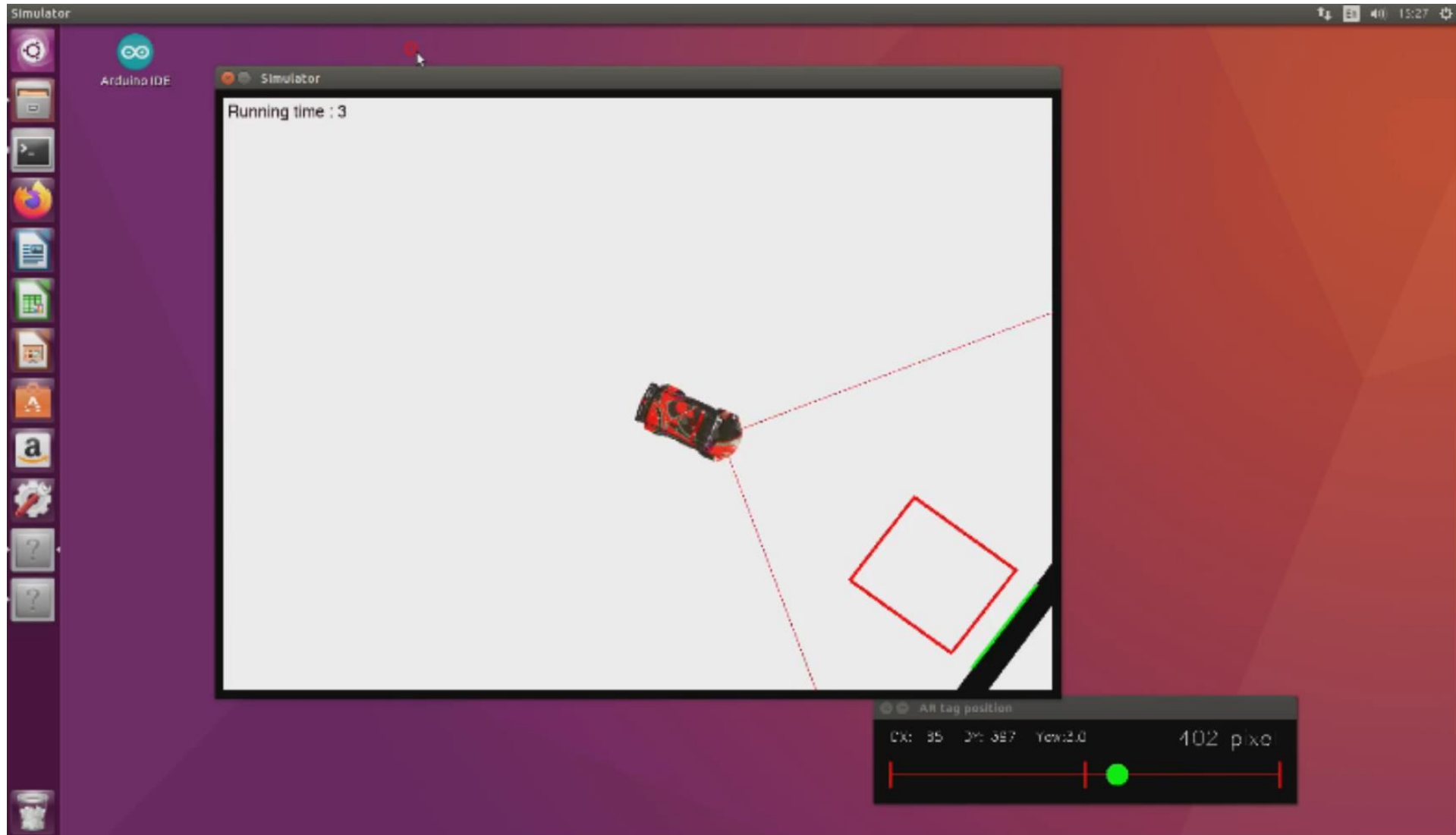
핸들 조향각 (-50~50)

angle : 좌회전 MAX(-50) ~ 직진 (0) ~ 우회전 MAX (50)

```
int32 speed
```

차량 속도 (-50~50)

speed : 후진 MAX(-50) ~ 정지 (0) ~ 전진 MAX (50)



# 결과물 제출

- 동영상 제출
  - 시뮬레이터에서 자동차가 주차하는 화면을 동영상 캡처툴로 (동영상 파일)로 만들어 제출하세요.
  - 동영상 파일이름은 “parking.mp4” 로 하세요.
- 소스코드 파일 제출
  - parking.py
  - 소스코드에 한글로 코멘트(설명) 상세히 붙여서 제출하세요.
  - 설명이 상세할수록 좋습니다.
  - 함수와 블록 단위로 설명하는 코멘트도 넣고, 한 줄 한 줄 개별코드를 설명하는 코멘트로 넣으세요.
  - 여러 팀원들이 힘을 합쳐 함께 작성하고, 디버깅하고, 수정하고, 개선한 코드라면 당연히 소스코드에 상세한 코멘트가 많이 들어갈 수 밖에 없겠지요. 코멘트 작성이 아무런 부담이 안 될 겁니다.

# 과제 #3

---

다양한 미션을 수행하는  
자율주행SW 개발을 위한 SW설계서 작성

# 과제 설명

- 6가지 미션에 대한 자율주행 SW를 제작함에 있어서 사전에 준비되어야 하는 SW 설계서를 작성해 보는 것이 목표입니다.
- 총 6개의 미션 중 첫번째 미션에 대한 SW 설계서는 예시로 미리 작성되어 있습니다.

미리 작성된 예시를 참고하여 나머지 5개 미션에 대한 SW 설계서를 작성하면 됩니다.

어떤 센서를 사용할지 적고,  
어떤 기능이 필요한지,  
그 기능을 어떤 방법으로 구현할 것인지 간단히 계획을 기술하면 됩니다.

뒤쪽에는 사전 조사를 통해 공부한 활용기술들에 대해서 간단히 그림 등으로 정리하면 소개하면 됩니다.

## 제5회 국민대 자율주행 경진대회 예선경기 #과제3 자율주행 SW 설계서

### 1. 참가팀 일반사항

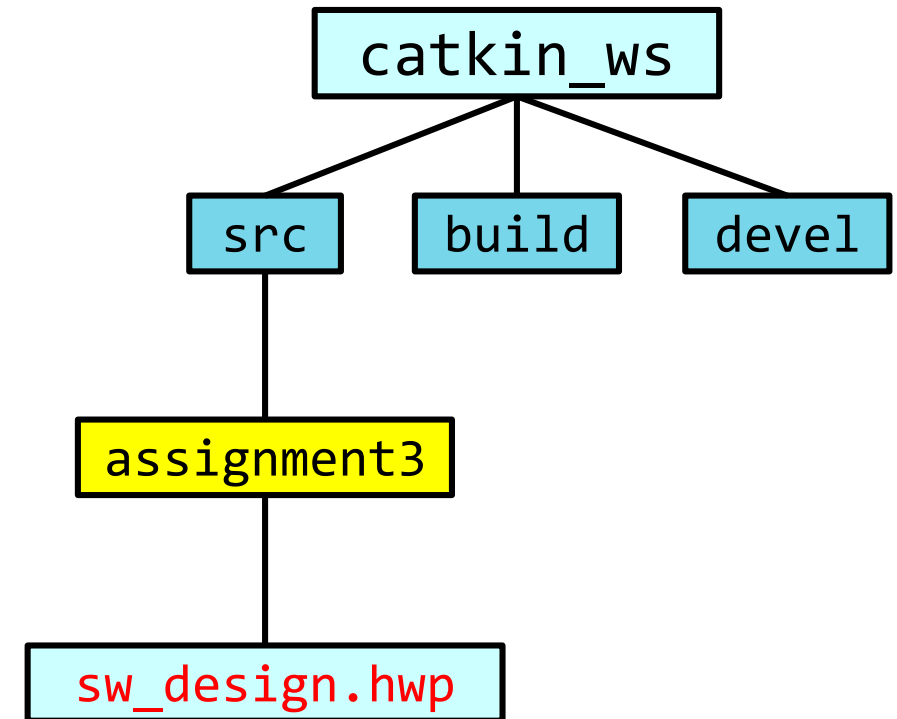
참가팀	팀명	고마자동차
	팀장	손흥민(국민대)
	팀원	류현진(국민대), 추신수(한성대), 김연아(한국항공대), 손연재(산업기술대)

### 2. 미션 완수를 위한 자율주행 SW 설계서

수행 미션 (1번)	차선을 벗어나지 않고 주행	
	<ul style="list-style-type: none"> <li>주행트랙에서 차량이 차선을 벗어나지 않고 주행해야 함. 코너와 곡선구간 차선 위에 세워둔 차선이탈 판정용 블록을 쓰러뜨리지 않아야 함.</li> </ul>	
	카메라	영상처리를 통해 차도에서 양쪽 차선의 위치를 파악하기 위해 사용
	초음파센서	주행 중에 전방에 있는 장애물과 충돌하는 것을 막기 위해 사용
	라이다	SLAM 기술로 차량이 트랙의 어디쯤 왔는지 위치를 파악하기 위해 사용
③ 사용하는 센서별 용도	IMU	차량의 현재 속도를 측정하기 위해 사용
	(1)	트랙에 그려진 차선을 인식하고 차량이 차선의 중간쯤에서 달려야 함.
	<ul style="list-style-type: none"> <li>카메라 ROS 토픽을 수신하여 OpenCV 이미지로 변환하여 영상처리를 진행한다.</li> <li>라이다 이미지를 회색톤으로 변환하고 다시 이진화 작업을 거쳐 차선을 회색으로 나</li> </ul>	

# 과제 수행에 필요한 폴더와 파일들

- sw\_design.hwp
  - 아래한글 파일
  - 설계서 작성을 위한 양식 파일
  - 첫번째 미션에 대한 예시 페이지 내용을 보고 나머지 비어 있는 부분을 채우면 됨



# SW 설계가 필요한 미션들

- 해결해야 할 미션 6가지

번호	미션 내용
1	<b>차선을 벗어나지 않고 주행</b> 주행트랙에서 차량이 차선을 벗어나지 않고 주행해야 함. 코너와 곡선구간 차선 위에 세워 둔 차선이탈 판정용 블록을 쓰러뜨리지 않아야 함.
2	<b>어두운 터널을 통과</b> 카메라 영상으로 차선을 인식하는 것이 불가능한 어두운 터널 안에서 거리센서(초음파센서 또는 라이다)를 이용해서 벽과 충돌하지 않으면서 주행하여 터널을 빠져나와야 함.
3	<b>장애물을 피해서 주행</b> 도로 위에 놓인, 차선 안쪽에 놓인 장애물과 충돌하지 않고 피해서 주행해야 함. 차선을 잠시 벗어날 수도 있으나 장애물을 모두 피한 후에는 정상적으로 차선 안쪽으로 되돌아와서 주행해야 함.
4	<b>정지선 정차 후 일정시간 후에 다시 재출발하여 주행</b> 정지선을 발견하면 지나치지 않고 정지선 앞에 정차해야 함. 정차 후 일정시간(3초)이 지나면 다시 출발하여야 함. 한계시간(5초)이 지나도록 계속 정차해 있으면 안됨.
5	<b>T자 수직주차 후 다시 출발</b> 후진으로 2대의 차량 사이에 수직으로 주차함. 주차 과정에서 양쪽 차량과 접촉사고가 나거나 후면 벽과 충돌하면 안됨. 3초 정차한 후 다시 빠져나와 주행트랙으로 복귀하여 계속 주행해야 함. 5초 이상 계속 주차하고 있으면 안됨.
6	<b>평행주차</b> 후진 또는 전진으로 트랙과 수평으로 놓인 주차구역에 주차함. 벽 또는 장애물과 충돌하면 안됨. 주차구역 앞쪽 벽에 붙여 놓은 AR코드를 이용하여 차량이 AR코드를 정면으로 바라보는 위치에 똑바로 정확히 주차해야 함.

# 결과물 제출

- 파일 제출
  - 아래한글 파일과, 그걸 출력한 PDF 파일을 함께 제출하세요.
  - 파일이름은 변경없이 그대로 제출하세요. “sw\_design.hwp”, “sw\_design.pdf”





# 과제 제출 방법

---

과제 제출 방법

# 제출 방법

---

- 제출 기한 : 2022년 6월 12일 (일) 23시59분 까지만 받습니다.
- 제출 방법
  - 이메일 : [hscho@xytron.co.kr](mailto:hscho@xytron.co.kr)
- 첨부파일 형식
  - 팀명-날짜.zip 으로 압축하여 보내주세요.  
(예) 씽씽카-2022-06-04.zip

# 제출 형태

- 3개의 과제 결과물을 압축해서 제출하세요.

씹씹카-2022-06-04.zip

서브폴더 만들지 말고 5개 파일을 하나의 폴더에 놓고서 한꺼번에 압축해서 만드세요.

- 제출 파일

	과제1	과제2	과제3
동영상	driving.mp4	parking.mp4	-
소스코드	driving.py	parking.py	-
문서	-	-	sw_design.hwp sw_design.pdf

- 문서는 아래아한글(hwp)과 pdf 파일로, 동영상은 mp4 포맷으로 제출



Q&A

---