Robot Operation System

Lecture material based on the book, ROS Robot Programming

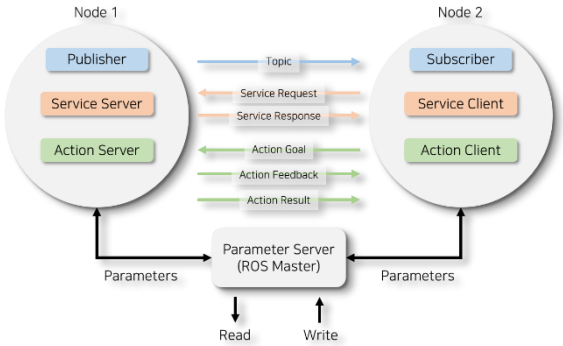
2018 Fall

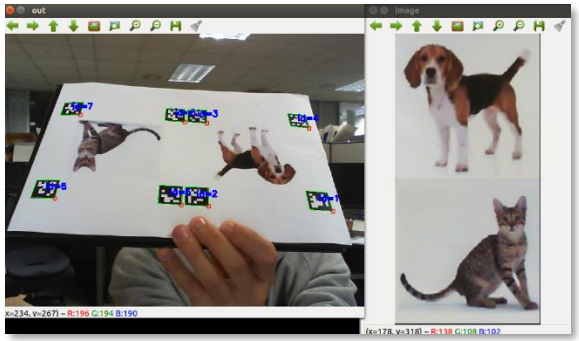
ME401 Capstone Design
ME491 Programming for Autonomous Mobile System

KAIST Department of Mechanical Engineering

T.A. Dongha Chung chungdongha@kaist.ac.kr

CONTENTS





Some theoretical background + Code explanation

Do not just copy & paste the code without understanding them

`Is for terminal command`

`Is for terminal output`

`Is for script`

Prerequisite

New and shiny is not always the best

ubuntu® 16.04.5 Xenial Xerus – [Eng] + 

ROS Kinetic

How to setup ubuntu 16.04.5 OS :
1. Download iso file from http://releases.ubuntu.com/16.04/
2. Make bootable usb : https://tutorials.ubuntu.com/tutorial/tutorial-create-a-usb-stick-on-windows#3
3. Setup : https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop#4

How to setup ROS in ubuntu :
1. Follow instruction and select "**Desktop-full install**" : http://wiki.ros.org/kinetic/Installation/Ubuntu

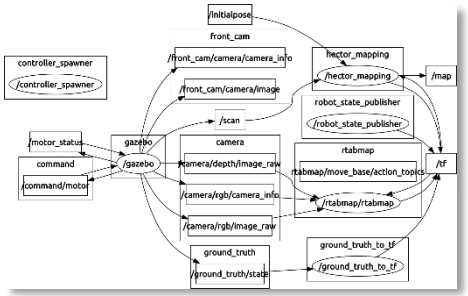
※It is important to install ROS with "Desktop-full install" option

---

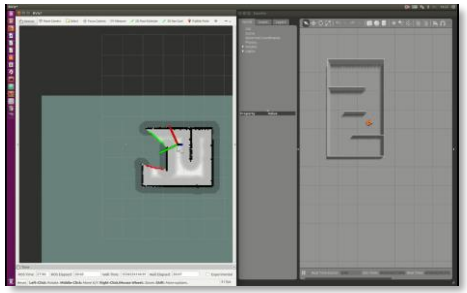
Additional (recommended for your convenience) setup
Ubuntu hangul(한글) : http://hochulshin.com/ubuntu-1604-hangul/
Atom (a code editing program) : https://codeforgeek.com/2014/09/install-atom-editor-ubuntu-14-04/
Terminator (splitable termianl) : https://askubuntu.com/questions/829045/how-do-i-install-terminator

Introduction to ROS

ROS is an open-source, meta-operating system for your robot



Plumbing

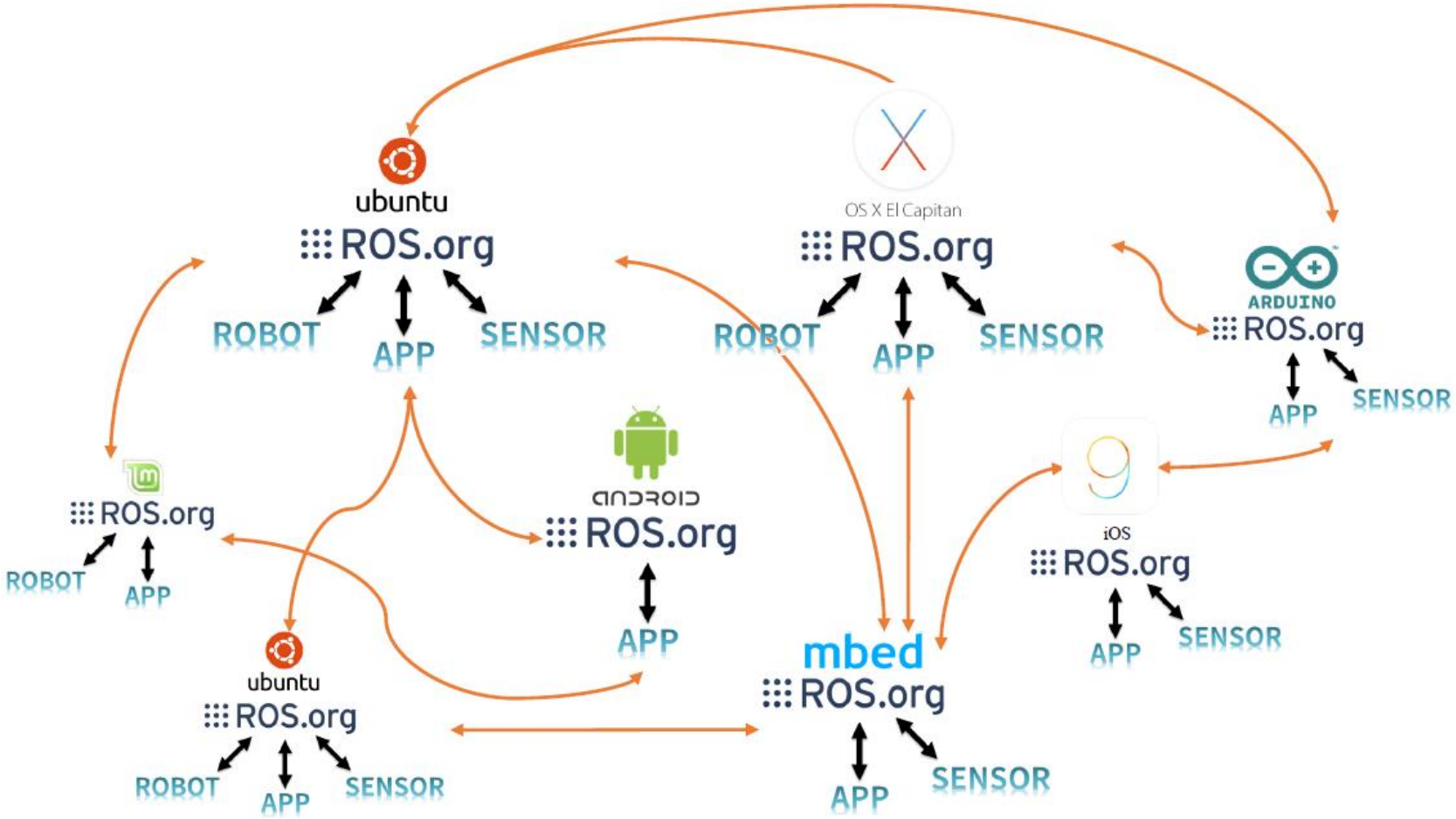


Tools

Algorithms

Introduction to ROS

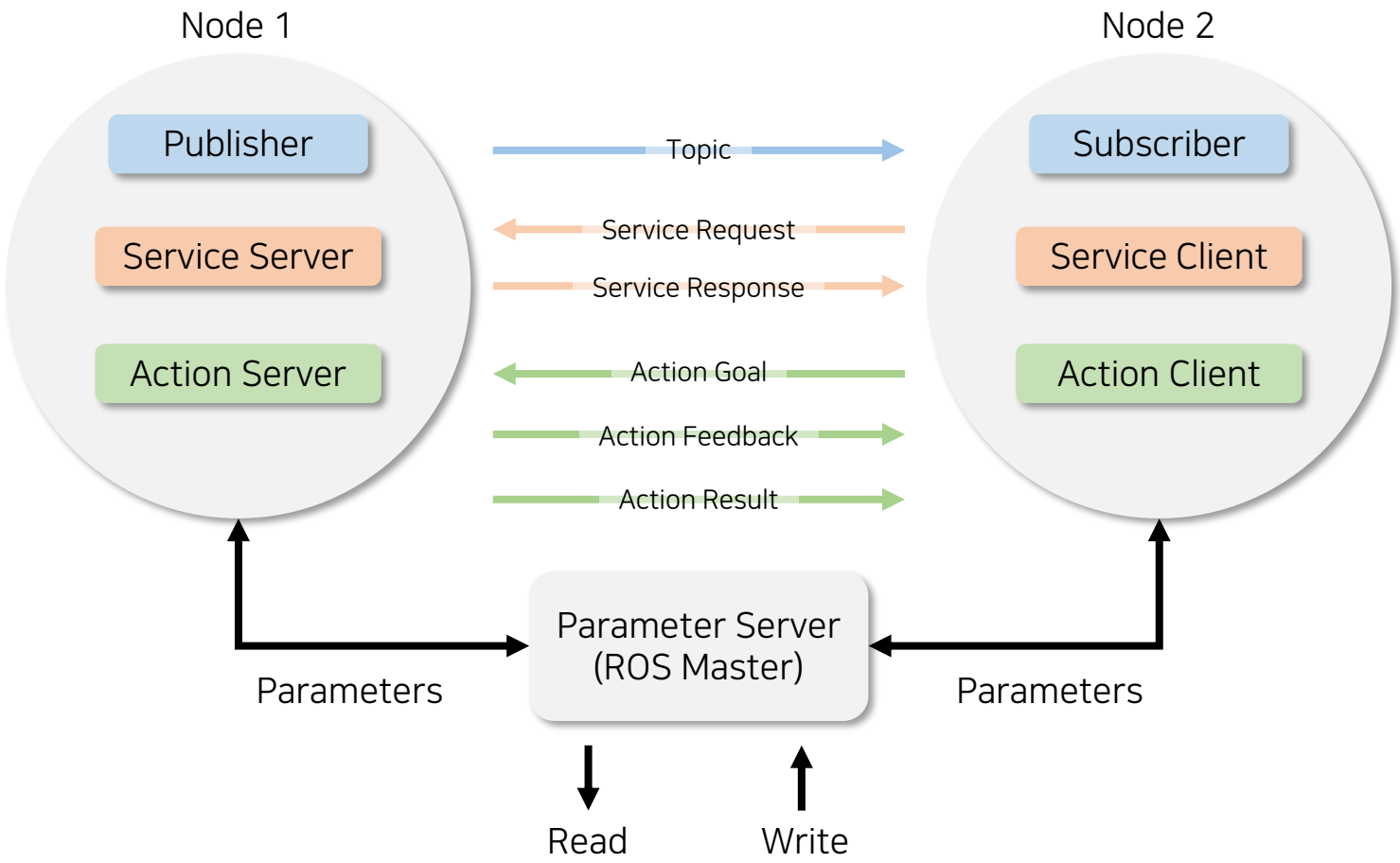
Compatible with different OS



Making nodes and connecting them

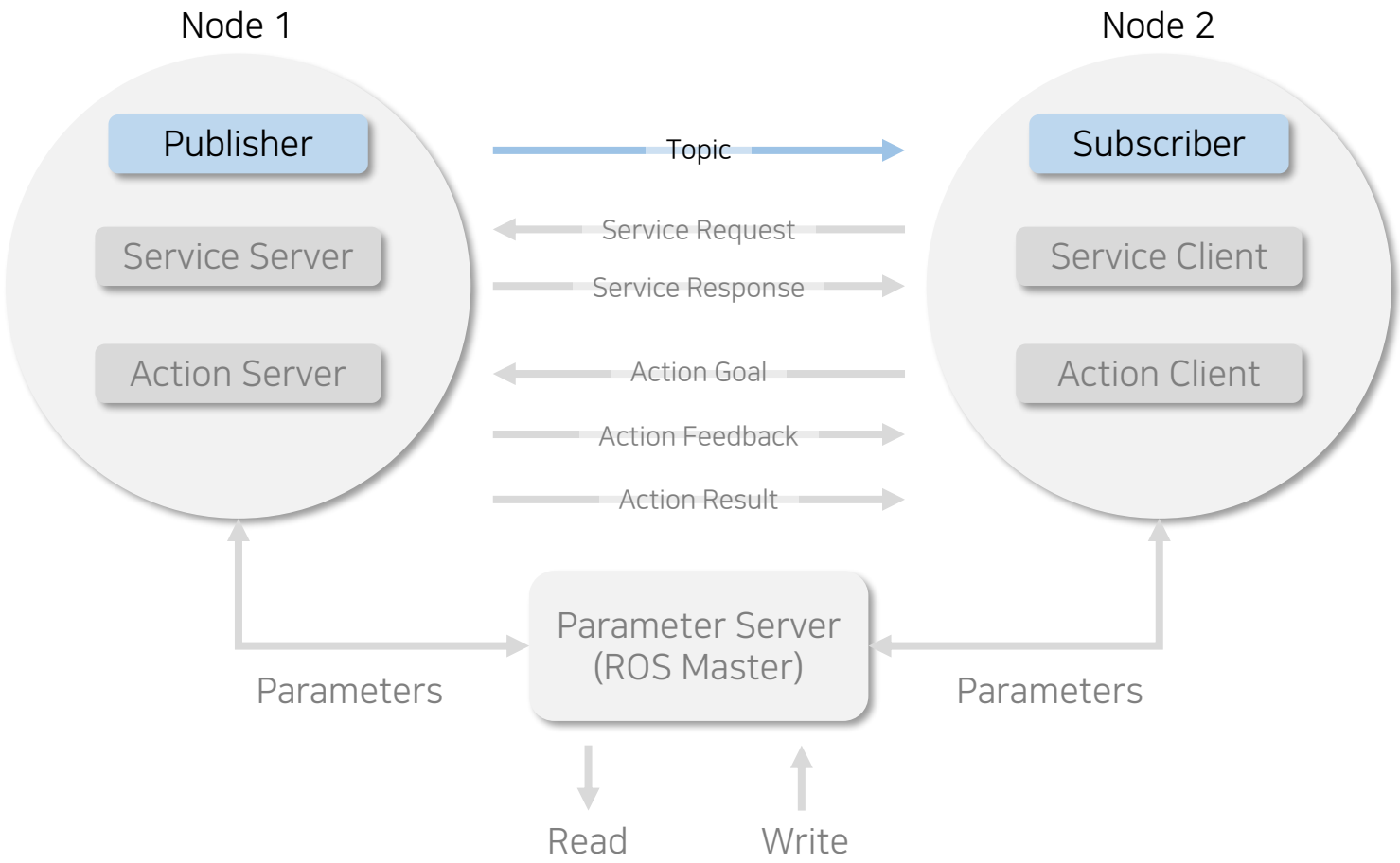
Communications between nodes & ROS master

Chapter 7 : https://github.com/robotpilot/ros-seminar/blob/master/07_ROS_기본_프로그래밍.pdf



Making nodes and connecting them

Publish & Subscribe between nodes



Making nodes and connecting them

Let's start with making workspace

```
~$ cd                                                    -  Goes to home directory
~$ mkdir catkin_make                                     -  Make a folder named "catkin_ws"
~$ cd catkin_make                                        -  Goes inside the folder "capstone_simulation"
~$ mkdir src                                             -  Important! : the folder name should be "src"
~$ catkin_make init                                      -  Initialize the workspace
~$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc  -  Include the bash file in workspace
~$ source ~/.bashrc                                      -  Source ~/.bashrc
```

Now, let's create a package named "ros_tutorials_topic"

You are now at ~/catkin_ws

```
~$ cd src                                                -  Goes to src
~$ catkin_create_pkg ros_tutorials_topic message_generation std_msgs roscpp
```

- Package name
- Dependency 1 "We are going to generate messages"
- Dependency 2 "We are going to use standard messages"
- Dependency 3 "We are going to use c++ language"

https://github.com/dhchung/me491_ros

Making nodes and connecting them

Let's start with making workspace

You are now at ~/catkin_ws/src

```
~$ cd ros_tutorials_topic      -  Goes to ros_tutorials_topic
~$ ls                          -  List segments
```

You will see :

```
include                        -  Folder containing header files
src                            -  Folder containing source codes
CMakeLists.txt                 -  File for build settings
package.xml                    -  File for package settings
```

Make some changes in package.xml (Optional)

```
~$ gedit package.xml           -  Open file using text modifying program 'gedit'
```

Making nodes and connecting them

package.xml

Description of the package

```
<?xml version="1.0"?>
<package format="2">
<name>ros_tutorials_topic</name>
<version>0.0.0</version>
<description>The ros_tutorials_topic package</description>
<maintainer email="chungdongha@kaist.ac.kr">Dongha Chung</maintainer>
<license>TODO</license>  <buildtool_depend>catkin</buildtool_depend>
<build_depend>message_generation</build_depend>
<build_depend>roscpp</build_depend>
<build_depend>std_msgs</build_depend>
<build_export_depend>roscpp</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>roscpp</exec_depend>
<exec_depend>std_msgs</exec_depend>
<export>  </export>
</package>
```

Save & exit

Make some changes in CMakeLists.txt (Not optional)

```
~$ gedit CMakeLists.txt                    -  Open file using text modifying program ‘gedit’
```

Making nodes and connecting them

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_tutorials_topic)

## Packages required for catkin build
## If the dependent packages (message generation roscpp std_msgs) does not exist : error
find_package(catkin REQUIRED COMPONENTS message_generation roscpp std_msgs)
## Declare message : MsgTutorial.msg
add_message_files(FILES MsgTutorial.msg)
## Set the dependent message : std_msgs
generate_messages(DEPENDENCIES std_msgs)
## Options for catkin package : dependencies on the system and catkin build
catkin_package(
  LIBRARIES ros_tutorials_topic
  CATKIN_DEPENDS std_msgs roscpp)
## Setting the include directories
include_directories(${catkin_INCLUDE_DIRS})
## Options for the node, 'topic_publisher'
## Settings for executable file, target link library, and additional libraries
add_executable(topic_publisher src/topic_publisher.cpp)
add_dependencies(topic_publisher ${${PROJECT_NAME}_EXPORTED_TARGETS}${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_publisher ${catkin_LIBRARIES})
## Options for the node, 'topic_subscriber'
add_executable(topic_subscriber src/topic_subscriber.cpp)
add_dependencies(topic_subscriber ${${PROJECT_NAME}_EXPORTED_TARGETS}${catkin_EXPORTED_TARGETS})
target_link_libraries(topic_subscriber ${catkin_LIBRARIES})
```

[1]

[2]

[3]

Save & exit

Making nodes and connecting them

Making messages : [1]

```
## Declare message : MsgTutorial.msg
add_message_files(FILES MsgTutorial.msg)
```

[1]

```
                                        You are now at ~/catkin_ws/src/ros_tutorials_topic
~$ mkdir msg                            -  Make a directory named msg
~$ cd msg
~$ gedit MsgTutorial.msg                -  Create a msg file named MsgTutorial
```

MsgTutorial.msg

```
time stamp                              -  time message named ‘stamp’
int32 data                              -  int32 message named ‘data’
```

```
Save & exit
```

Making nodes and connecting them

Making publisher node [2]

[2]

```
## Options for the node, 'topic_publisher'
add_executable(topic_publisher src/topic_publisher.cpp)
```

```
~$ cd ..                                          -  Goes back to upper directory
~$ cd src
~$ gedit topic_publisher.cpp                      -  Create a cpp file named topic_publisher
```

topic_publisher.cpp

```
#include "ros/ros.h"                              -  Basic ROS header file
#include "ros_tutorials_topic/MsgTutorial.h"      -  MsgTutorial header file
int main(int argc, char **argv)                   -  Main function of the node
{
  ros::init(argc, argv, "topic_publisher");       -  Initialize the node name as "topic publisher"
  ros::NodeHandle nh;                             -  Declare the node handle for communication
                                                     with ROS system

  ros::Publisher ros_tutorial_pub = nh.advertise<ros_tutorials_topic::MsgTutorial>("ros_tutorial_msg", 100);
                                                  -  Declare the publisher 'ros_tutorial_pub :
                                                       -  We will use MsgTutorial message
                                                       -  Publisher queue size is 100

  ros::Rate loop_rate(10);                        -  Set the loop rate as 10 : 10Hz
  ros_tutorials_topic::MsgTutorial msg;           -  Declare a message variable msg
  int count = 0;                                  -  Initialize the variable 'count'

  while(ros::ok())                                -  While the node is running do :
  {…}
…
```

Making nodes and connecting them

Making publisher node [2]

topic_publisher.cpp

```
...
  while(ros::ok())
  {
    msg.stamp = ros::Time::now();                  - Set the message ‘stamp’ as current time
    msg.data = count;                              - Set the message ‘data’ as ‘count’
    ROS_INFO("send msg = %d", msg.stamp.sec);      - Print out stamp.sec
    ROS_INFO("send msg = %d", msg.stamp.nsec);     - Print out stamp.nsec
    ROS_INFO("send msg = %d", msg.data);           - Print out data
    ros_tutorial_pub.publish(msg);                 - Publish the message
    loop_rate.sleep();                             - Goes into sleep regarding to the loop rate
    ++count;                                       - Add 1 to variable ‘count’
  }
  return 0;
}
```

Save & exit

Making nodes and connecting them

Making subscriber node [3]

```
## Options for the node, 'topic_subscriber'
add_executable(topic_subscriber src/topic_subscriber.cpp)
```

[3]

topic_subscriber.cpp

```
#include "ros/ros.h"
#include "ros_tutorials_topic/MsgTutorial.h"

void msgCallback(const ros_tutorials_topic::MsgTutorial::ConstPtr& msg)
{
  ROS_INFO("recievemsg= %d", msg->stamp.sec);
  ROS_INFO("recievemsg= %d", msg->stamp.nsec);
  ROS_INFO("recievemsg= %d", msg->data);
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "topic_subscriber");
  ros::NodeHandle nh;
  ros::Subscriber ros_tutorial_sub= nh.subscribe("ros_tutorial_msg", 100, msgCallback);
  ros::spin();
  return 0;
}
```

- Callback function which will run when message is received
- Main function of the node
- Initialize the node name as 'topic_subscriber'
- Declare node handle
- Declare the subscriber 'ros_tutorial_sub :
  - We will use MsgTutorial message
  - Subscriber queue size is 100

Save & exit

Making nodes and connecting them

Build the package & run

```
~$ cd ~/catkin_ws                                   -  Goes to catkin_ws directory
~$ catkin_make                                      -  Execute catkin build
```

On one terminal

```
~$ roscore                              -  Run ROS Master, parameter server, and rosout logging node
```

On other terminal

```
~$ rosrun ros_tutorials_topic topic_publisher    -  Run the ros node topic_publisher
```

Package name — Node name

On another terminal

```
~$ rosrun ros_tutorials_topic topic_subscriber   -  Run the ros node topic_subscriber
```

Package name — Node name

On another terminal

```
~$ rqt_graph                                        -  Shows the nodes and their connections
```

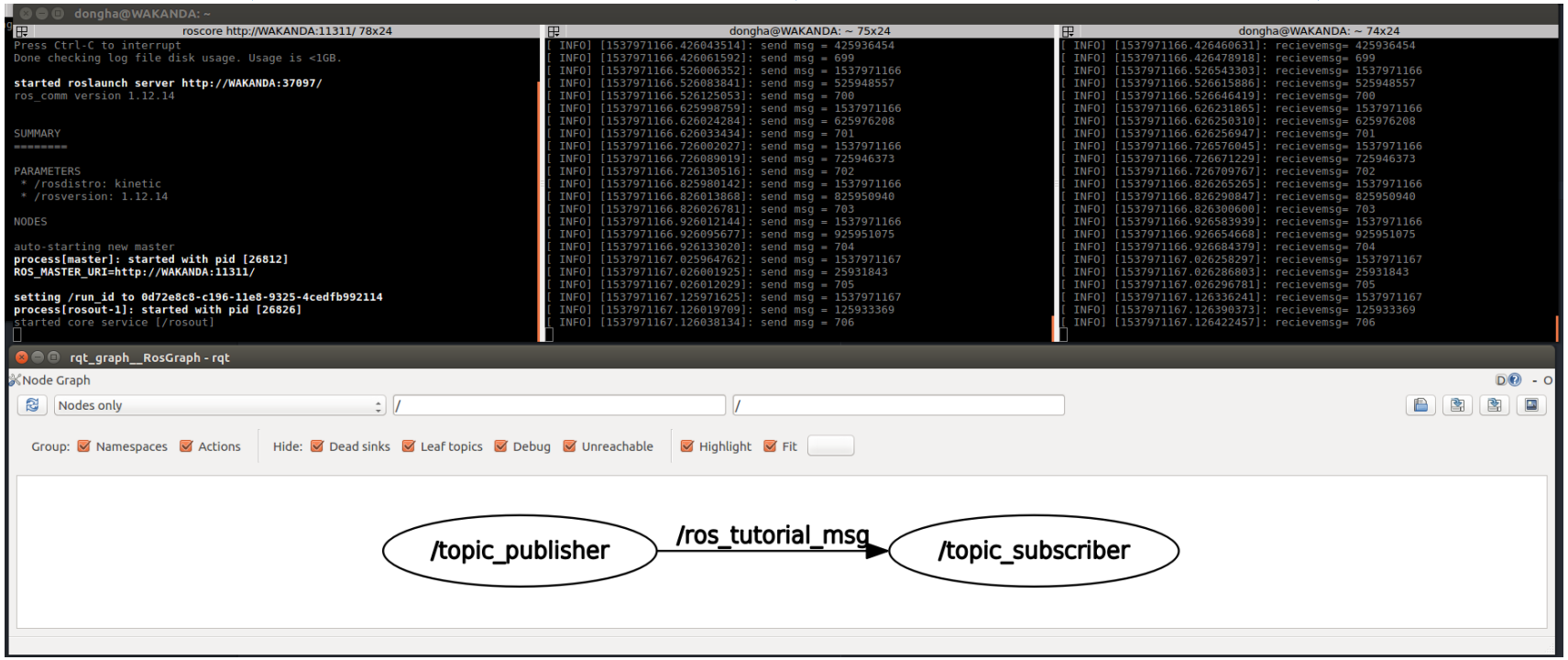
Making nodes and connecting them

Output

roscore

topic_publisher node

topic_subscriber node



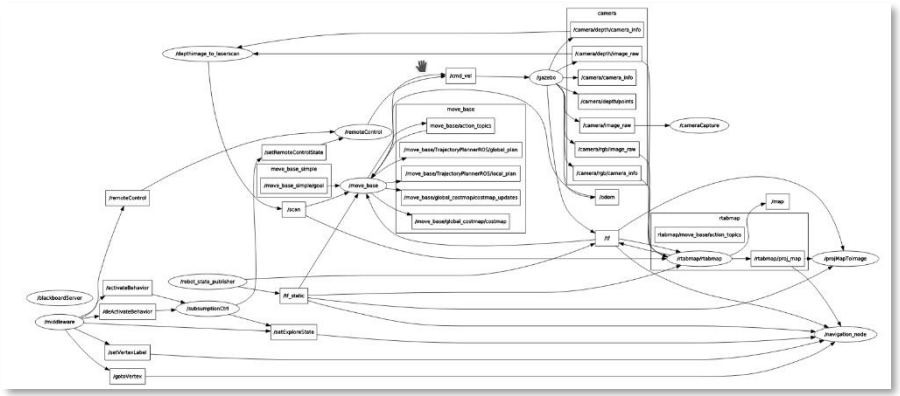
rqt_graph

Making nodes and connecting them

Publish & Subscribe between nodes

For much complex systems…

Node1, Node2, Node3, … Node n [1]



Publish-Subscribe relationship (rqt_graph) [2]

[1] https://videohive.net/item/black-lines-and-dots-clean-background/16910801
[2] https://medium.com/@genefoxwell/marvins-head-pt-1-faf260831883

For more information…

Dr. Pyo's lecture note

https://github.com/robotpilot/ros-seminar

Getting the cat image and dog image from webcam

Problem with using CNN for webcam image

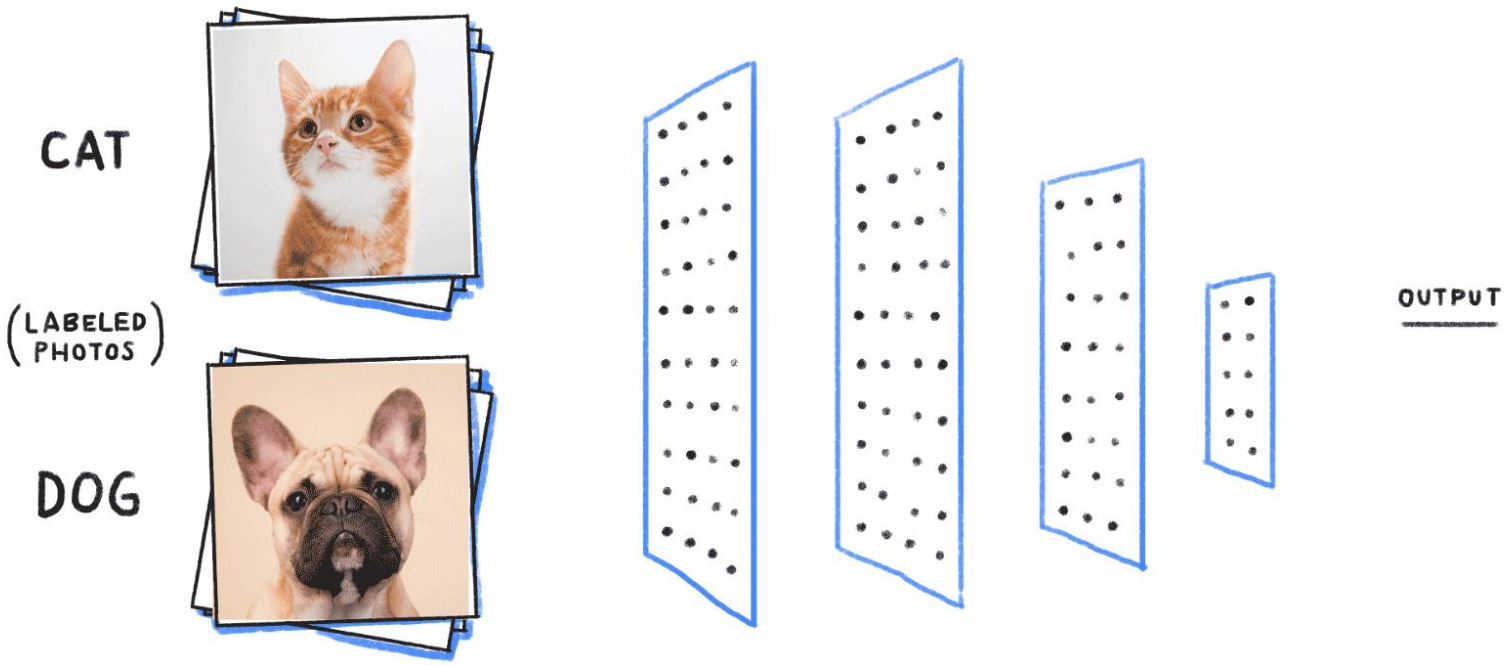
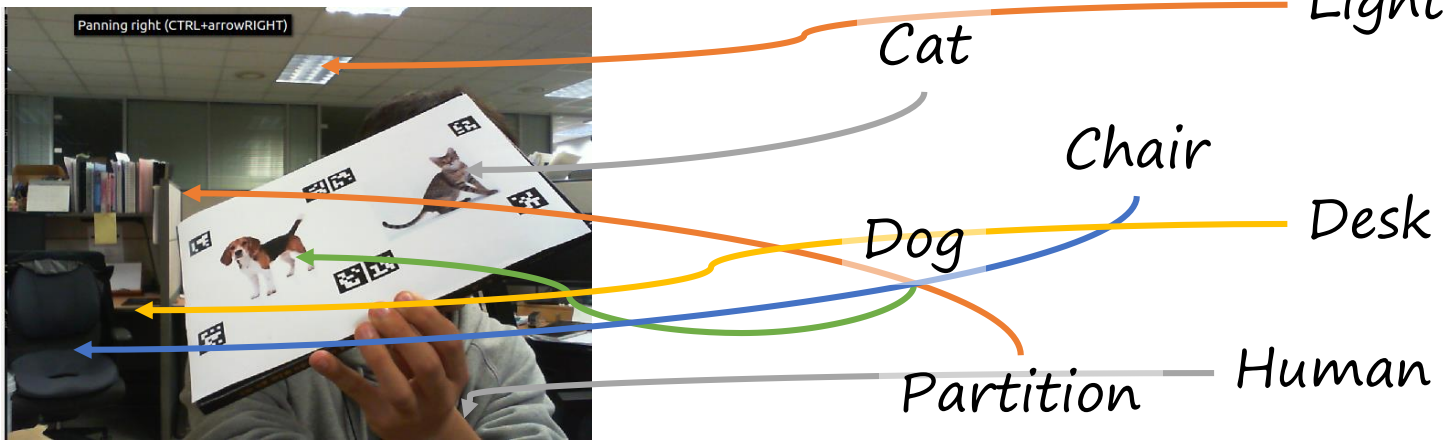


Image from https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8

Getting the cat image and dog image from webcam

Problem with using CNN for webcam image



Too many information in one image

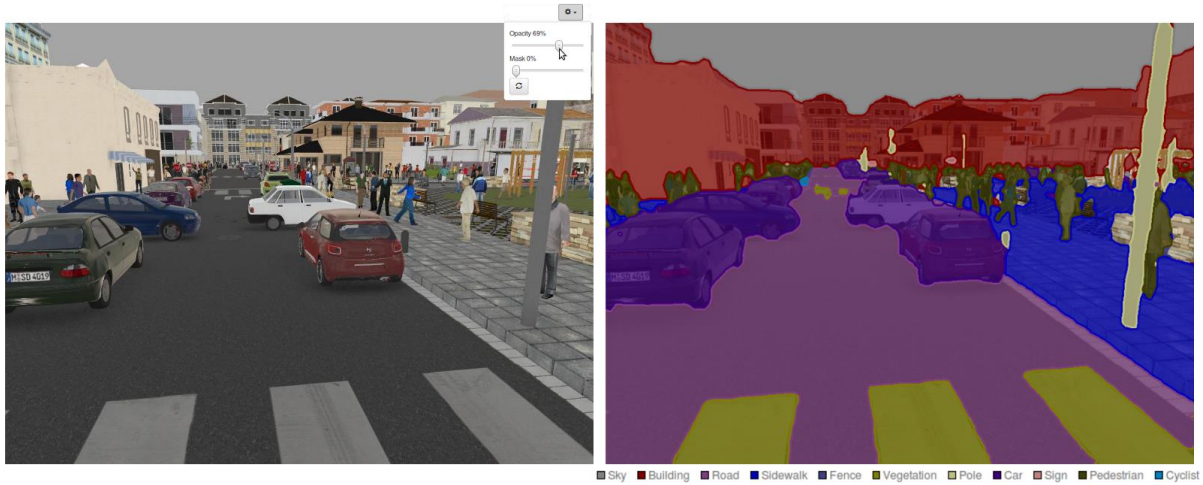


Image segmentation : https://medium.com/nanonets/how-to-do-image-segmentation-using-deep-learning-c673cc5862ef

Assignment : Getting warped cat & dog images from webcam

Getting the cat image and dog image from webcam

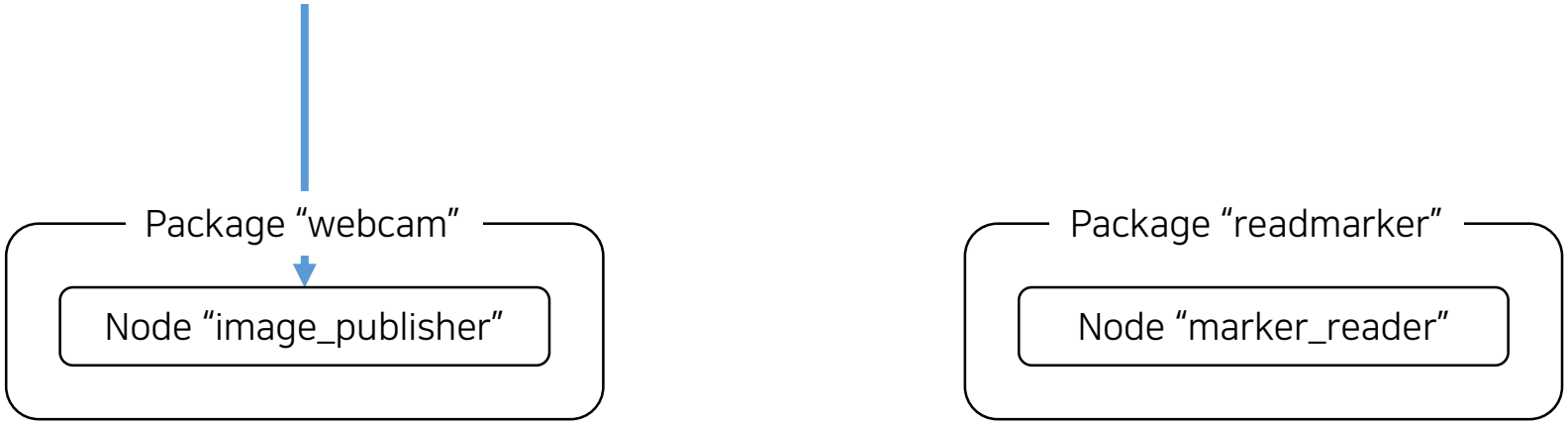
From webcam to readmarker

Capture a video frame from webcam 0 (if it does not work, try 1, 2, 3…)

```
VideoCapture cap = cv::VideoCapture(0);
```

Convert the image into Mat format

```
cap >> frame;
```



Getting the cat image and dog image from webcam

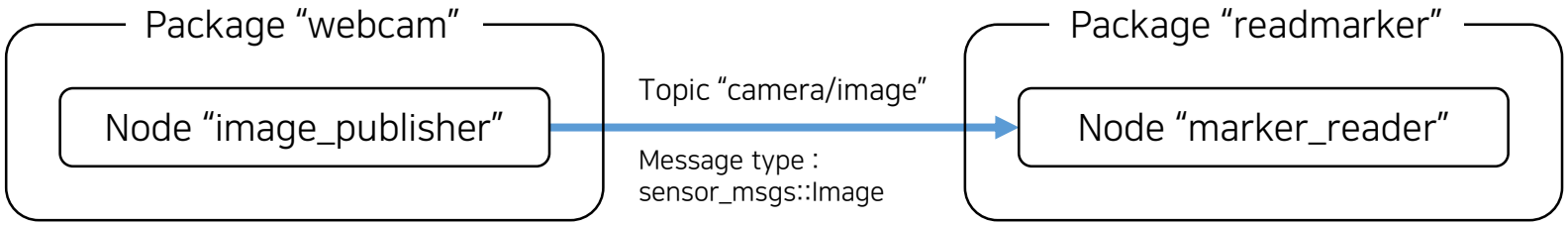
From webcam to readmarker

Convert the OpenCV Mat format to ROS message

```
msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8", frame).toImageMsg();
```

Publish the message

```
pub.publish(msg);
```



Getting the cat image and dog image from webcam

From webcam to readmarker

Crop and warp the image when the image is subscribed

```
void imageCallback(const sensor_msgs::ImageConstPtr& msg);
```

Convert the incoming ROS image message to OpenCV CvImage

```
cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
```

Convert the OpenCV CvImage to Mat

```
cv_ptr->image.copyTo(inputImage);
```

Detect markers, crop, warp, & publish

```
...
```

Package “webcam”

Node “image_publisher”

Package “readmarker”

Node “marker_reader”

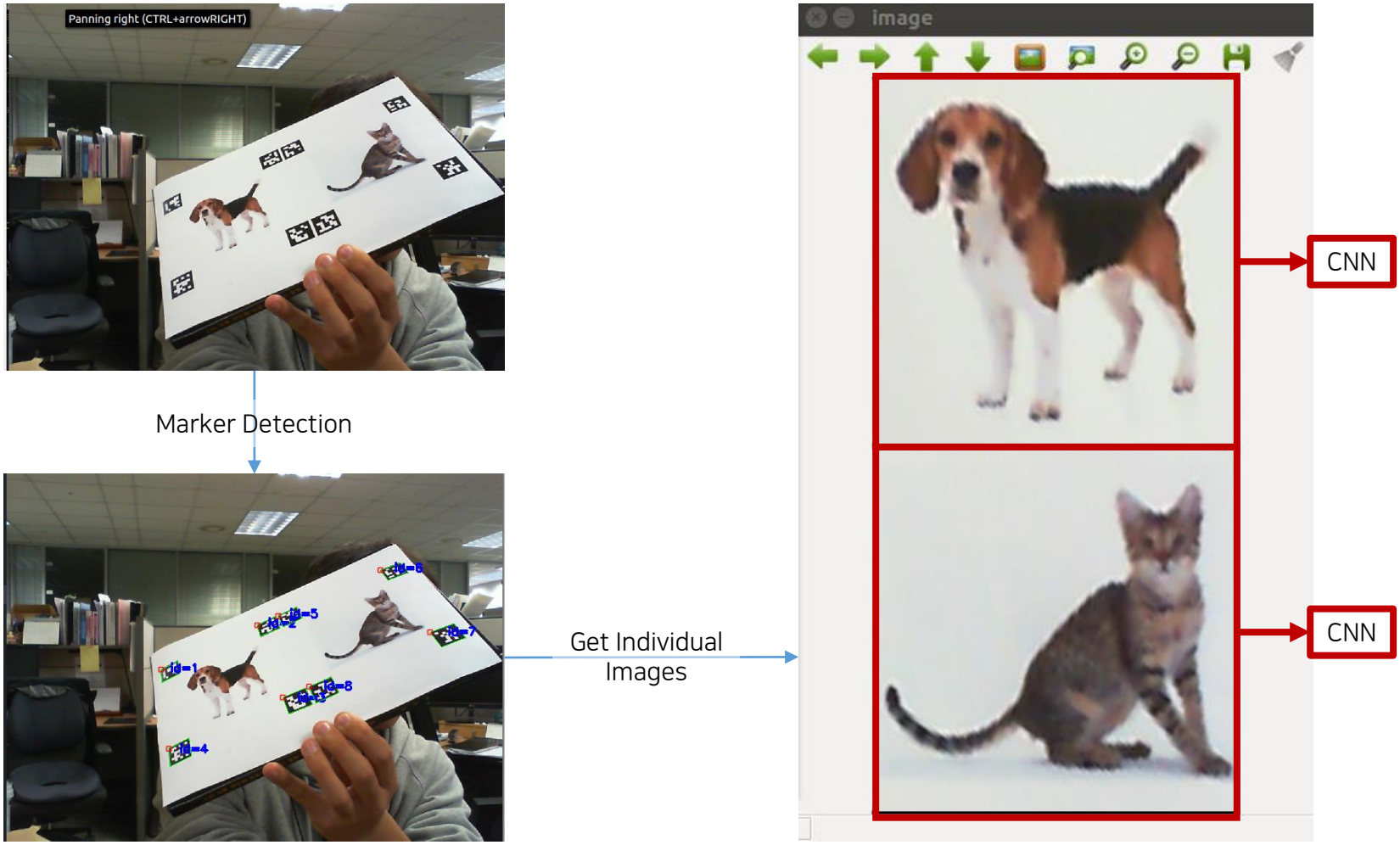
Topic “cropped_img”

Message type :
1. int32
2. sensor_msgs::CompressedImage

Out

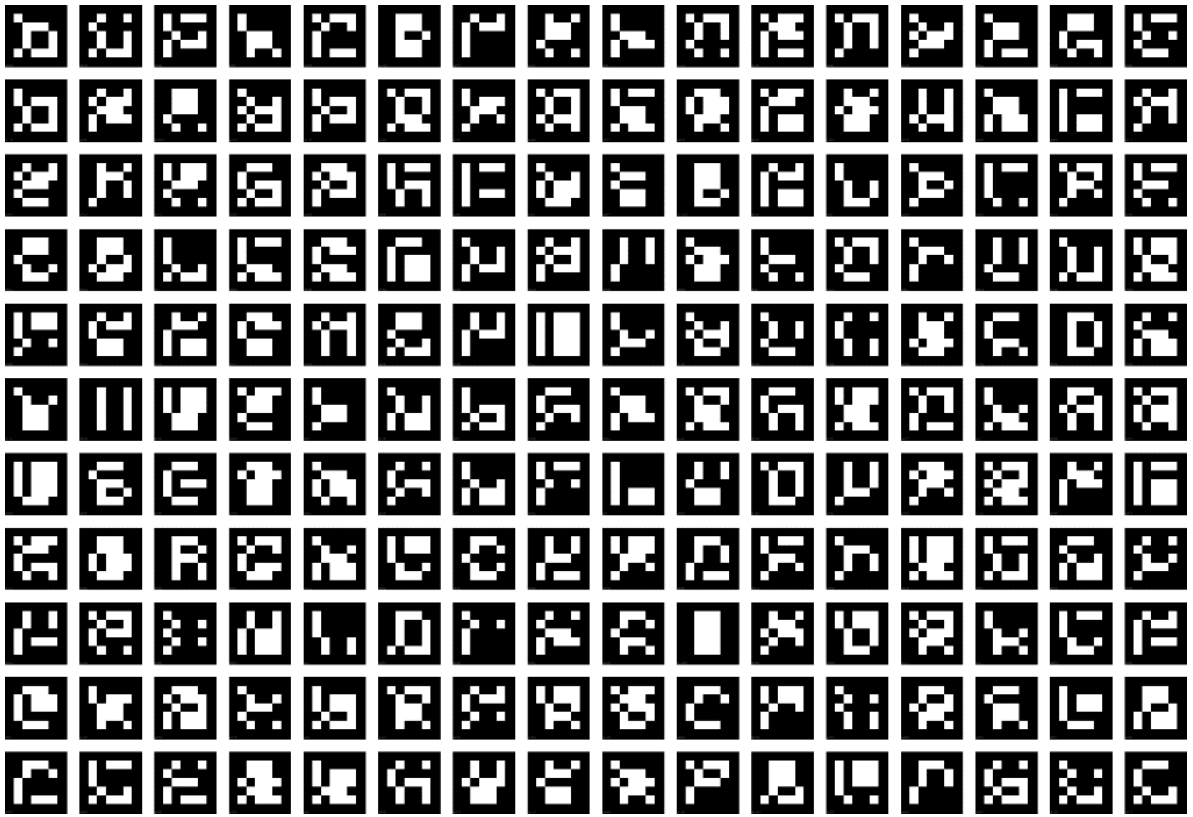
Getting the cat image and dog image from webcam

Use the AruCo markers to crop the images



Getting the cat image and dog image from webcam

AruCo in OpenCV



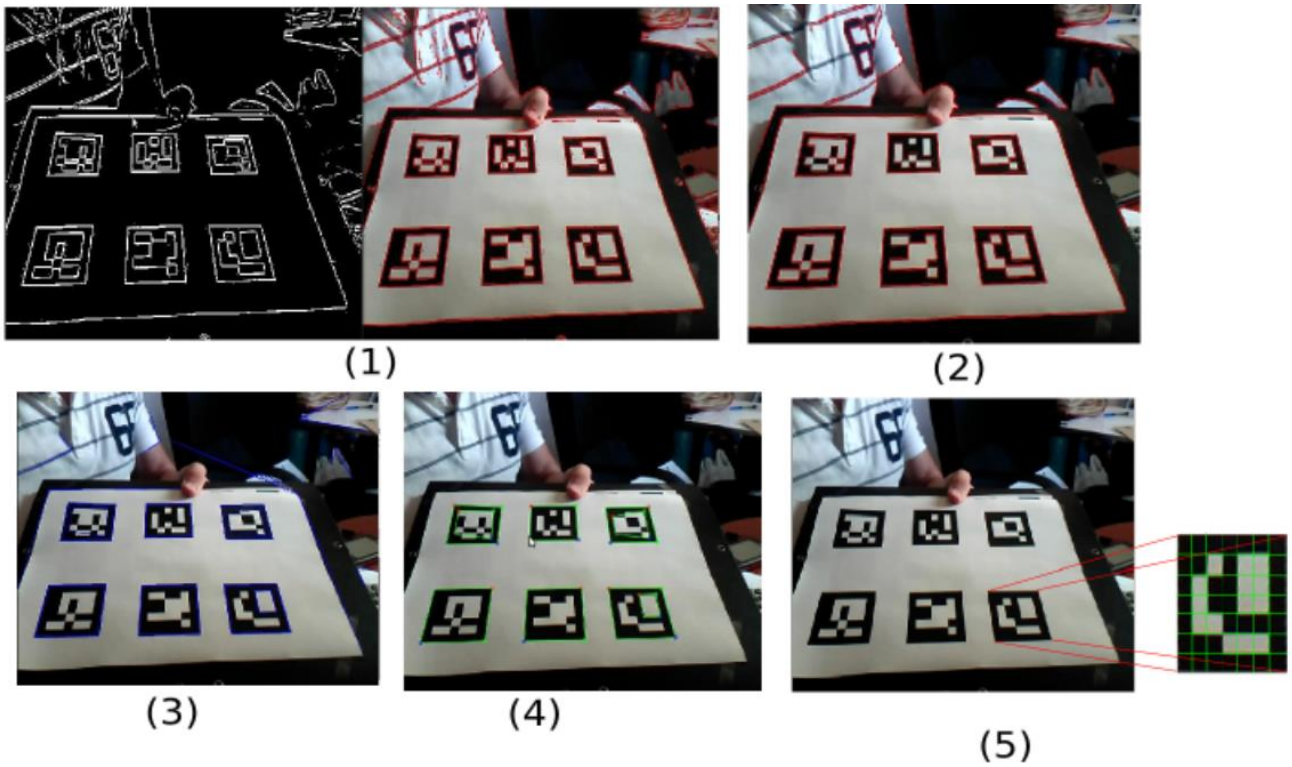
AruCo markers

Predefined marker dictionary

Making AruCo markers -> see package aruco

Getting the cat image and dog image from webcam

AruCo in OpenCV



1. Apply an Adaptive Thresholding so as to obtain borders
   Find contours. After that, not only the real markers are detected but also a lot of undesired borders.
2. Remove borders with an small number of points
3. Polygonal approximation of contour and keep the concave contours with exactly 4 corners
   Sort corners in anti-clockwise direction
4. Remove too close rectangles
5. Marker Identification

[1] https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit#

Getting the cat image and dog image from webcam

Use the AruCo markers to crop the images

Marker Detection

Detect the markers

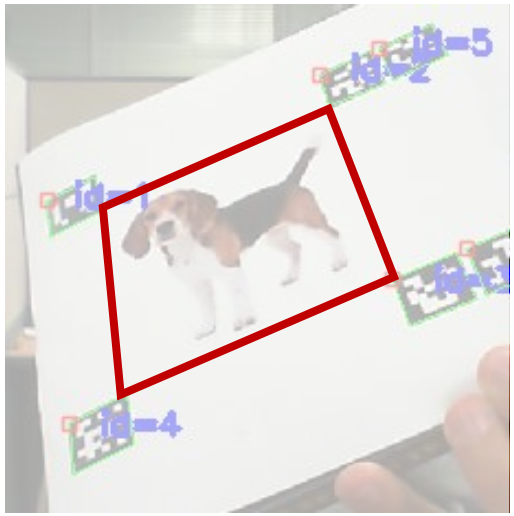
```
aruco::detectMarkers(inputImage, dictionary, markerCorners, markerIds)
```

Check if markers 1~4 & 5~8 are detected

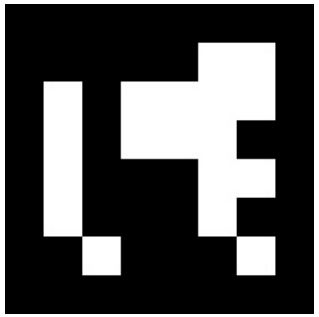
```
b_image1 and b_image2 is used to check if the images are available
```

If image 1 (marker 1~4) is available : b_image1 = 1,
Get the coordinates of image 1 coordinates and find homography

```
if(b_image1==1){…}
```



`markerCorners[i][0]` `markerCorners[i][1]`



`markerCorners[i][3]` `markerCorners[i][2]`

Getting the cat image and dog image from webcam

Use the AruCo markers to crop the images

Marker Detection

Detect the markers

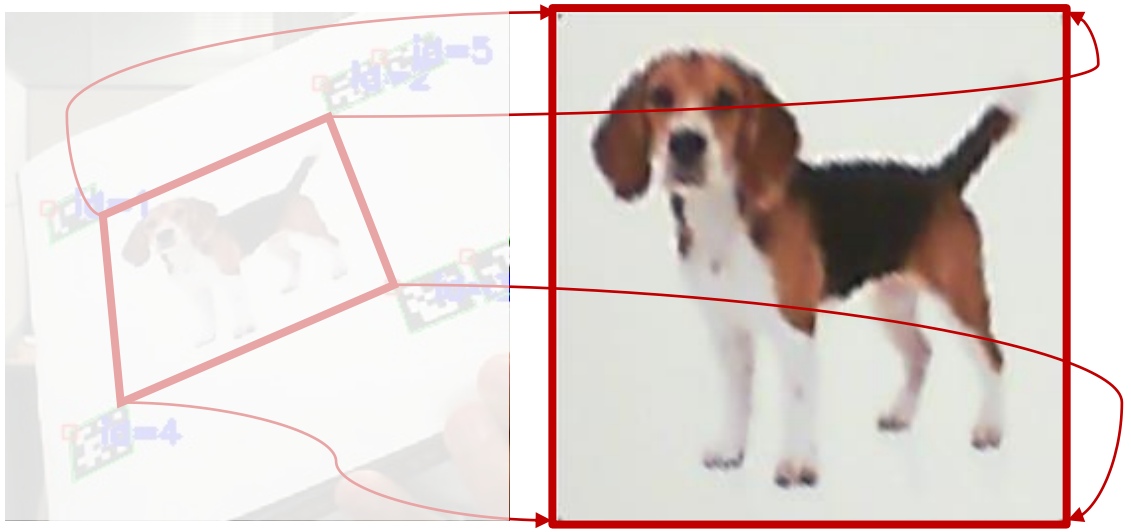
```
aruco::detectMarkers(inputImage, dictionary, markerCorners, markerIds)
```

Check if markers 1~4 & 5~8 are detected

```
b_image1 and b_image2 is used to check if the images are available
```

If image 1 (marker 1~4) is available : b_image1 = 1,
Get the coordinates of image 1 coordinates and find homography

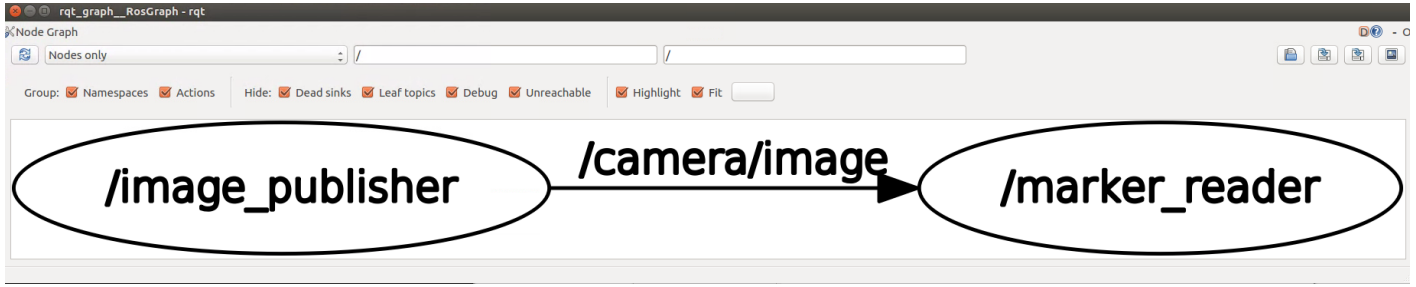
```
if(b_image1==1){…}
```



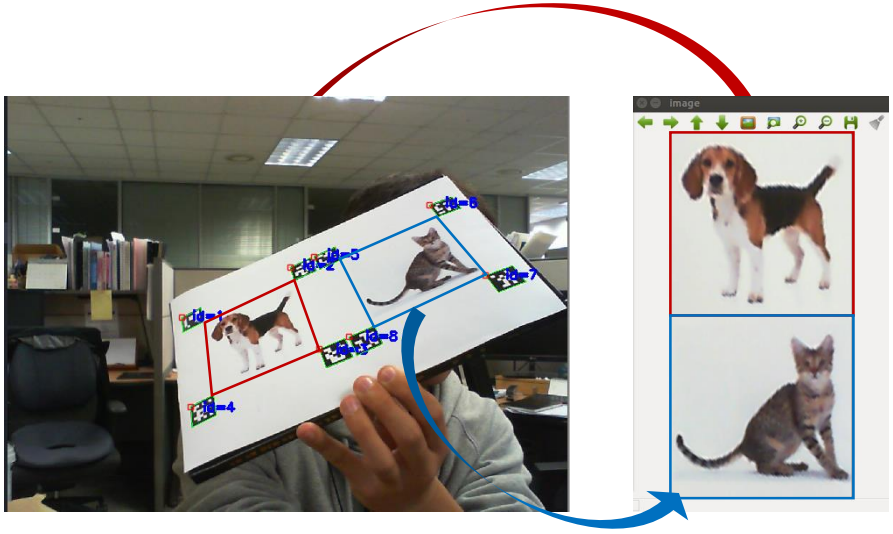
Homography (transformation matrix) : $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$

In this assignment

1. Know how the images are published / subscribed



2. Know how the dog image and cat image is cropped and warped



Review codes : webcam.cpp & readmarker.cpp

Appendix : Sample image

1

2

5

6

4

3

8

7

https://github.com/dhchung/me491_ros

SLAM: Simultaneous Localization & Mapping

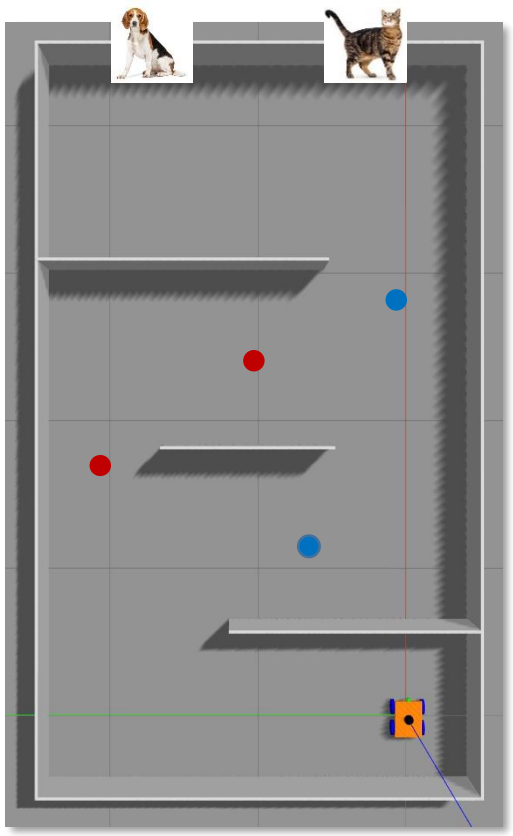
2018 Fall

ME401 Capstone Design
ME491 Programming for Autonomous Mobile System

KAIST Department of Mechanical Engineering

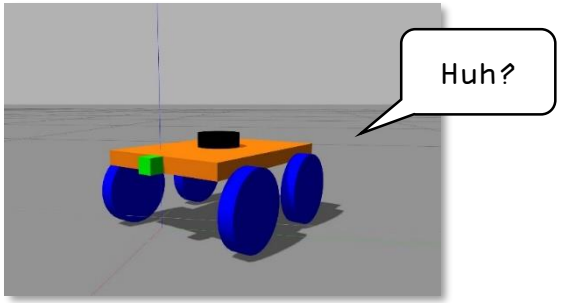
T.A. Dongha Chung chungdongha@kaist.ac.kr

What do we have to do...?



Capstone design 2 mission :

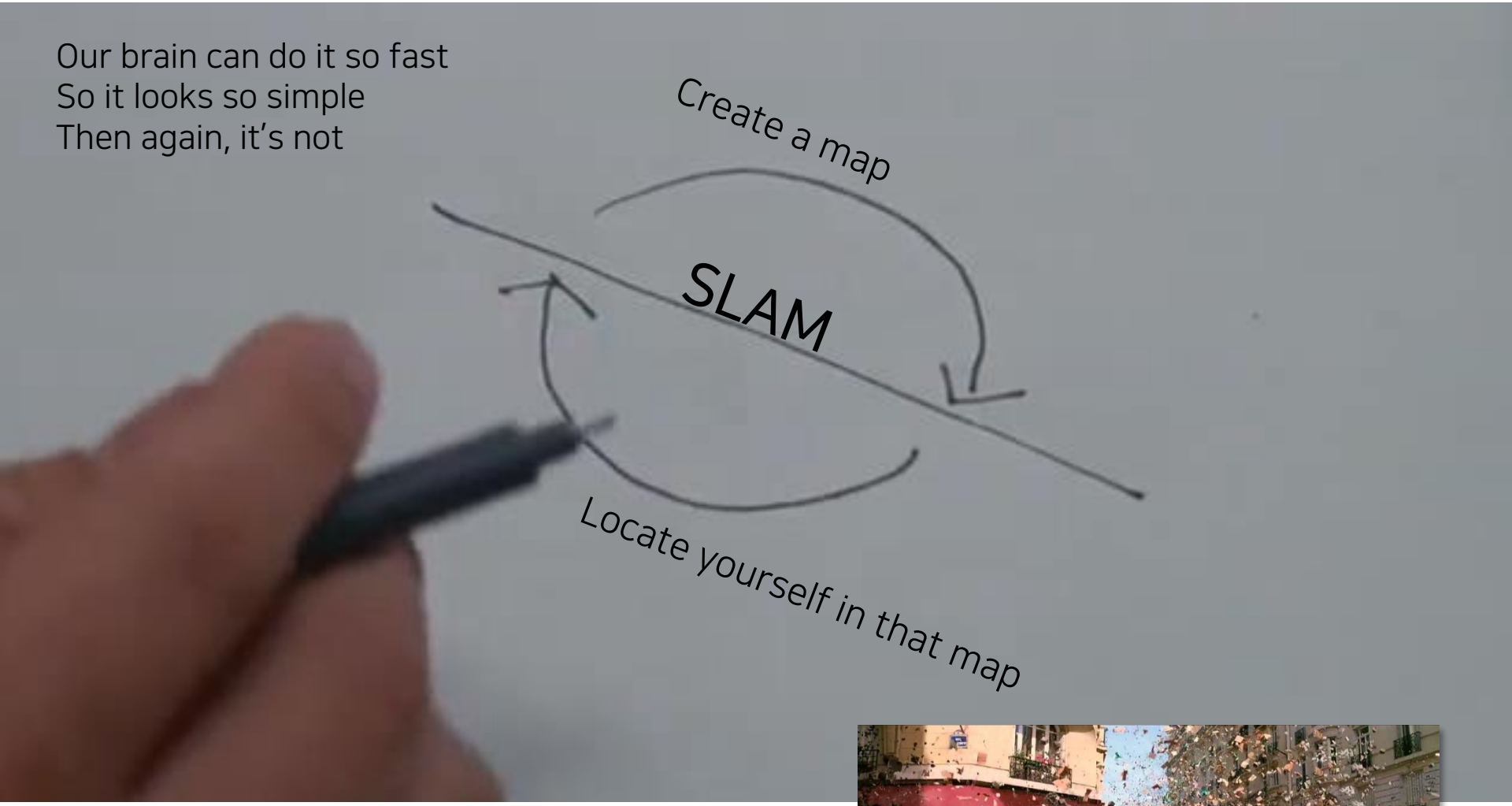
A. Pick up the balls
B. Drop the balls on the designated areas
   (i.e., red balls → cat, blue balls → dog)



Not that simple for the robots

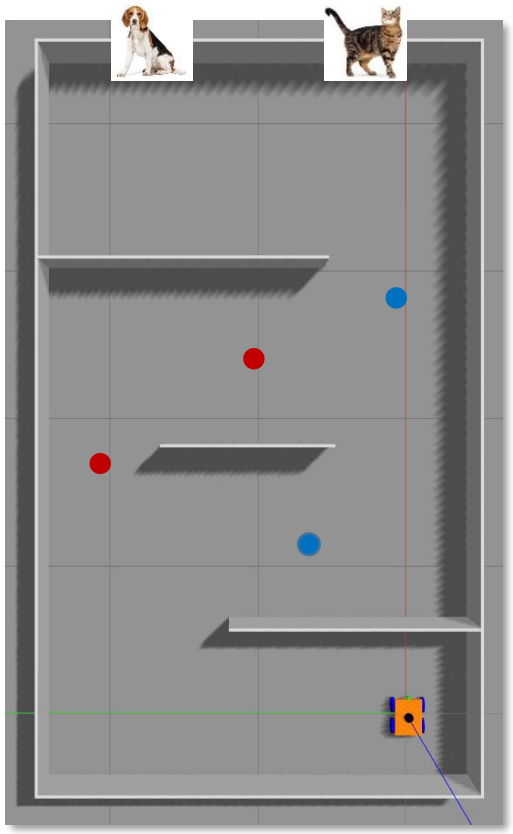
What do we have to do…?

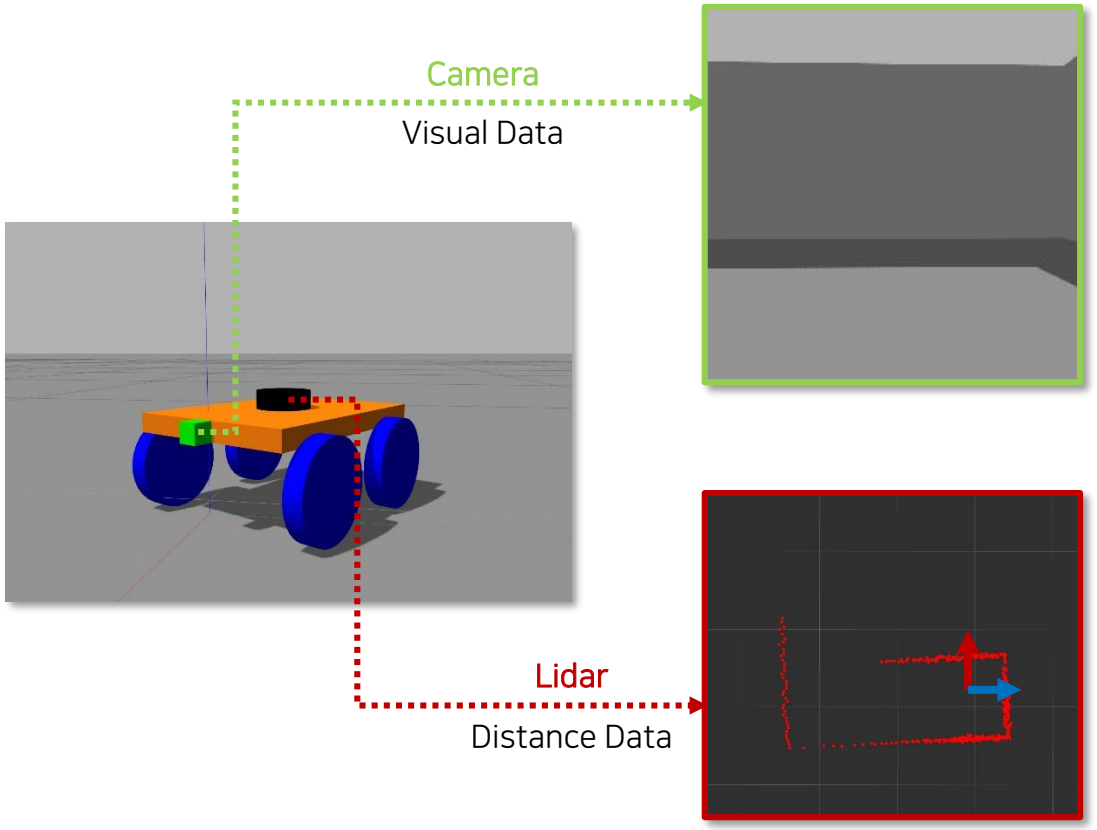
Our brain can do it so fast
So it looks so simple
Then again, it's not



A scene from the movie, Inception
What Mr. Cobb drew before the street blows up

What do we have to do...?





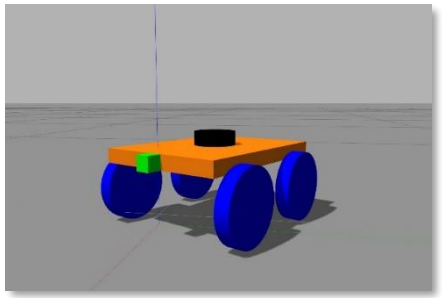
With given data, the robot have to process it to get useful informations :

Map data,

Ball location data,

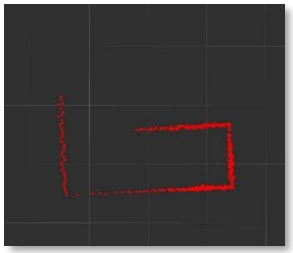
Goal position data, ··· , etc.

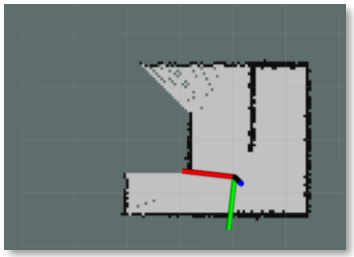
What do we have to do…?

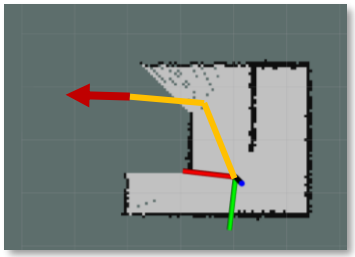


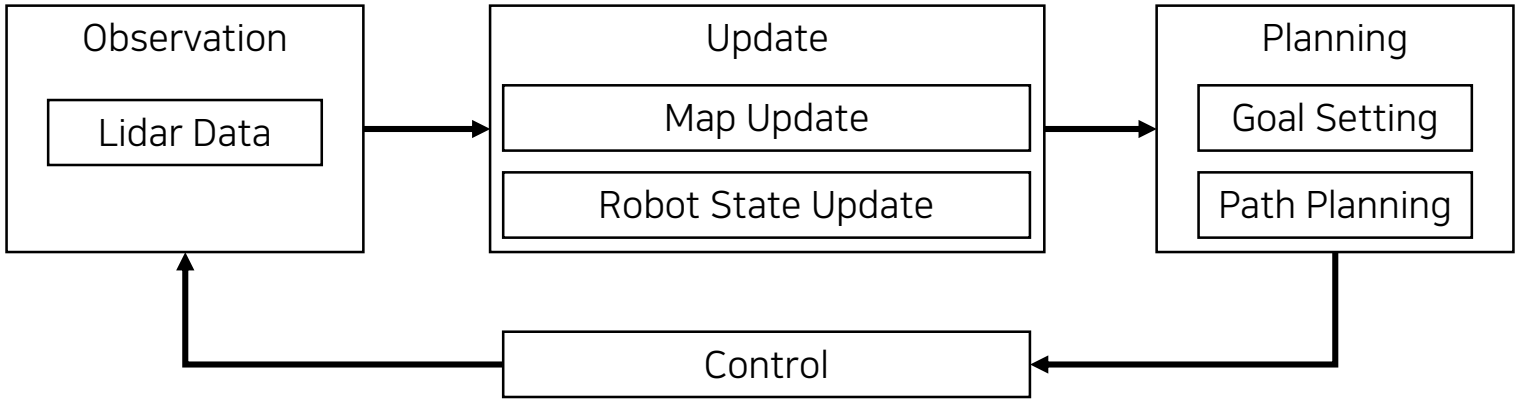
Before making the robot move, we should know :

A. How the map looks like
B. Where the robot is
C. Where the robot should go

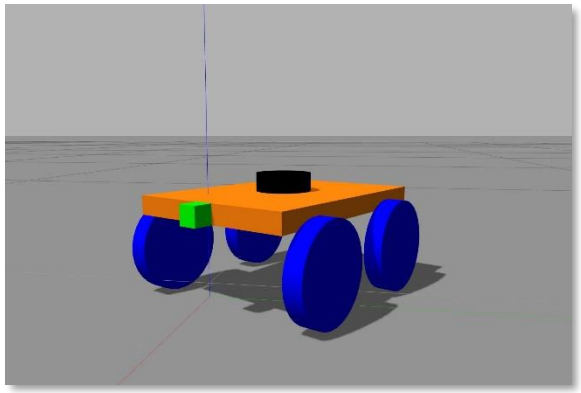


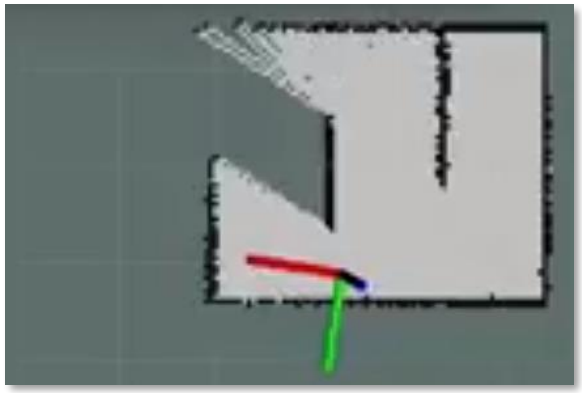


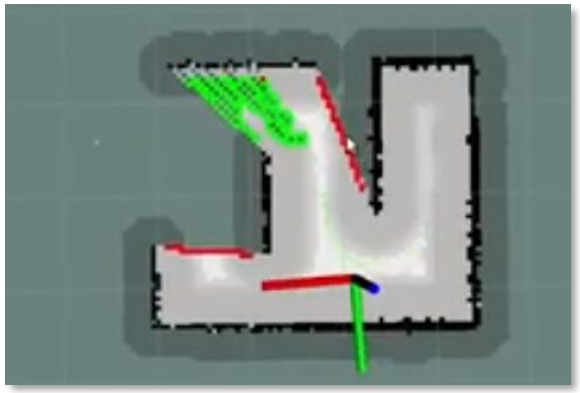




CONTENTS







Some theoretical background + Code explanation

Do not just copy & paste the code without understanding them

```
Is for terminal command
```

```
Is for terminal output
```

```
Is for script
```

01

Construction of simulation environment using Gazebo

KAIST Department of Mechanical Engineering

T.A. Dongha Chung chungdongha@kaist.ac.kr

About Gazebo

GAZEBO

Features

Dynamics Simulation
Access multiple high-performance physics engines including ODE, Bullet, Simbody, and DART.

Advanced 3D Graphics
Utilizing OGRE, Gazebo provides realistic rendering of environments including high-quality lighting, shadows, and textures.

Sensors and Noise
Generate sensor data, optionally with noise, from laser range finders, 2D/3D cameras, Kinect style sensors, contact sensors, force-torque, and more.

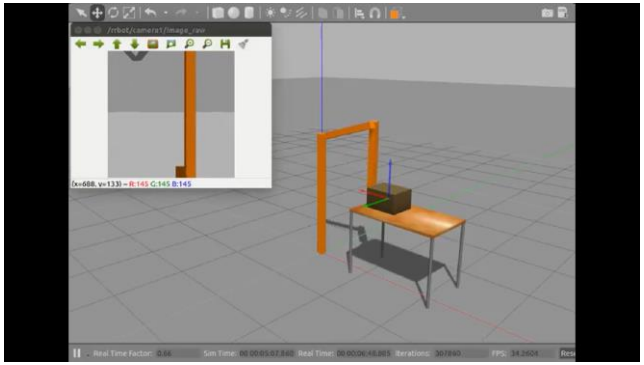
Plugins
Develop custom plugins for robot, sensor, and environmental control. Plugins provide direct access to Gazebo's API.

Robot Models
Many robots are provided including PR2, Pioneer2 DX, iRobot Create, and TurtleBot. Or build your own using SDF.

TCP/IP Transport
Run simulation on remote servers, and interface to Gazebo through socket-based message passing using Google Protobufs.

Cloud Simulation
Use CloudSim to run Gazebo on Amazon AWS and GzWeb to interact with the simulation through a browser.

Command Line Tools
Extensive command line tools facilitate simulation introspection and control.



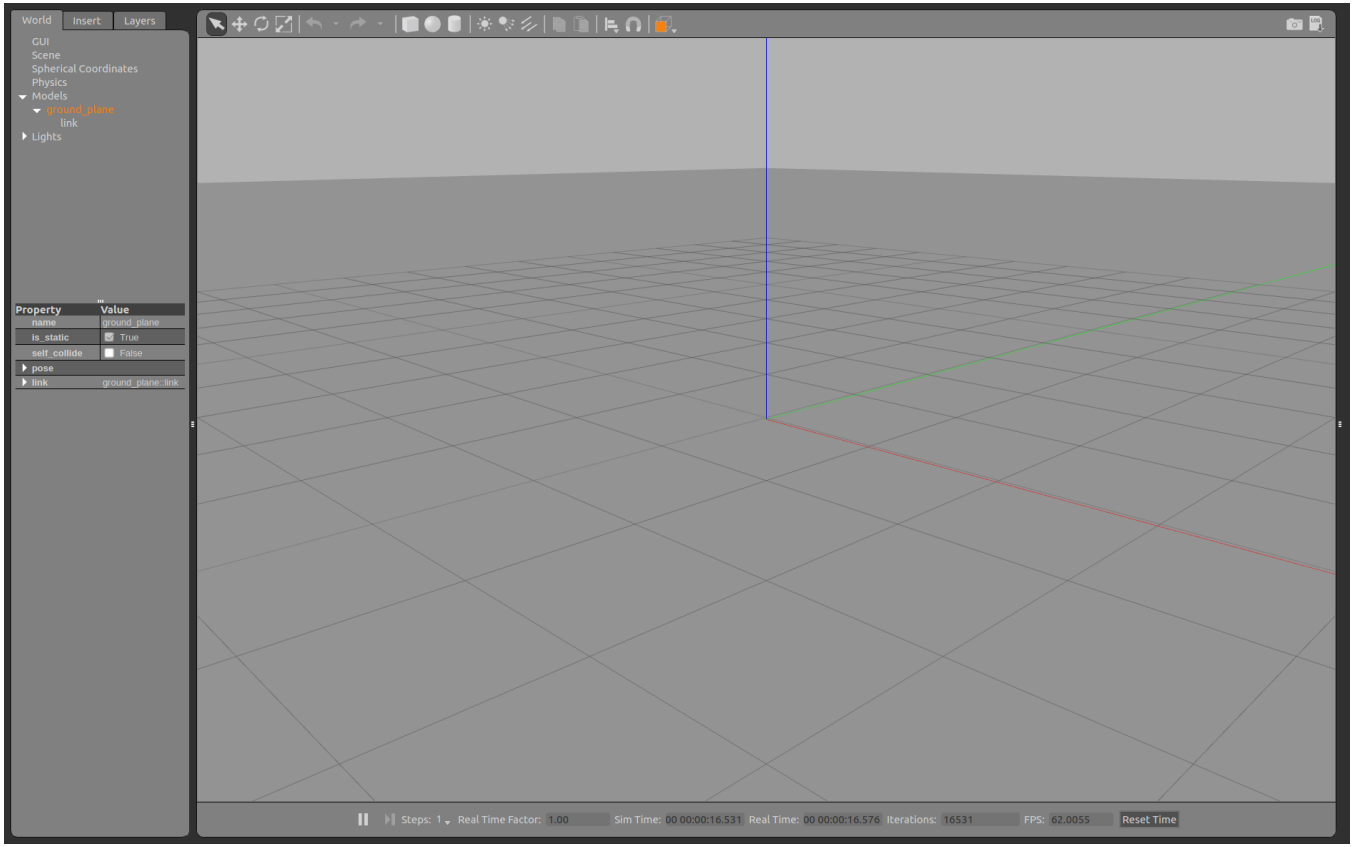
https://www.youtube.com/watch?v=OKOyTQQcrLw

https://www.youtube.com/watch?v=97JRYhKLhSY

Making a structure

First, let's make the maze structure

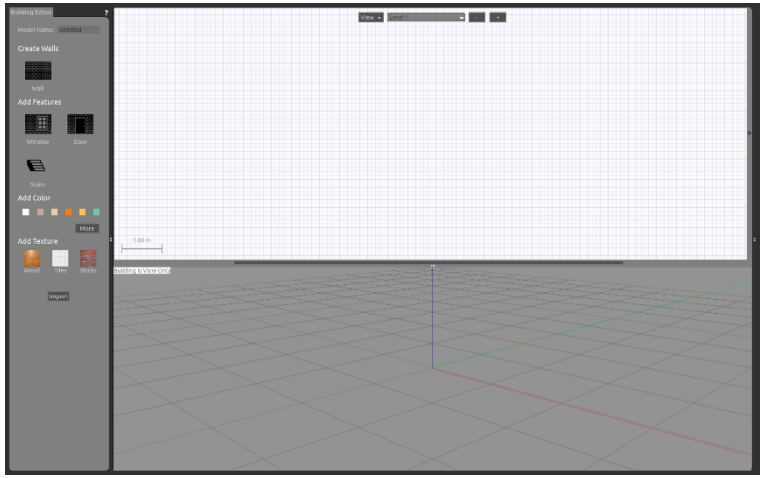
```
~$ rosrun gazebo_ros gazebo
```



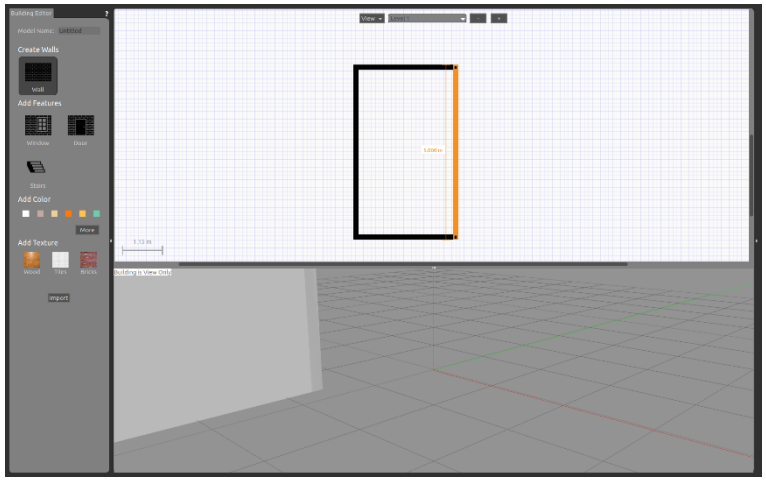
Press ctrl+b to enter the building editor

```
ctrl+b
```

Making a structure

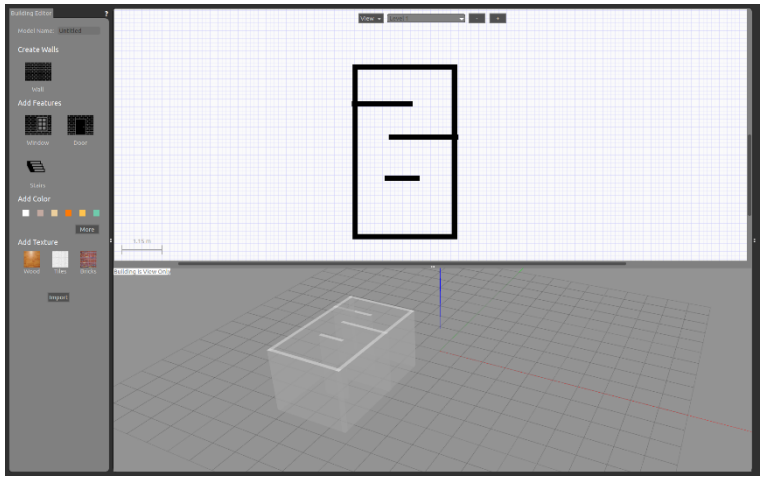


Click on the wall button on the top left, and create a 3 by 5 rectangular structure

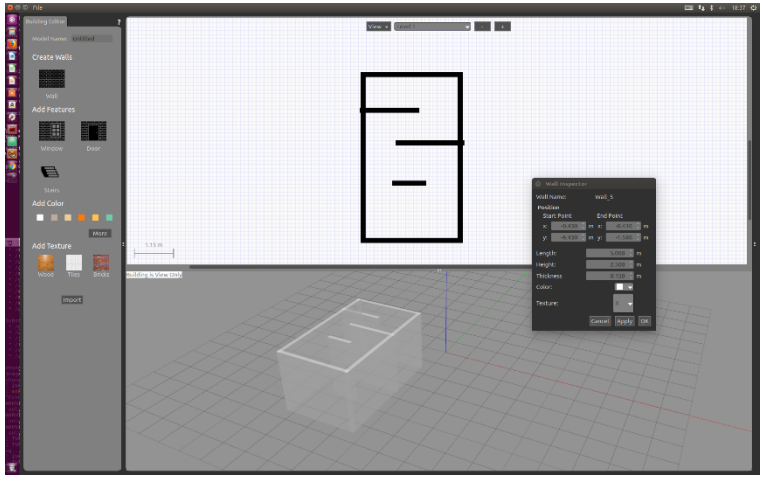


Making a structure

Make a maze of your own. Mind the gap between the walls so that the robot can pass through

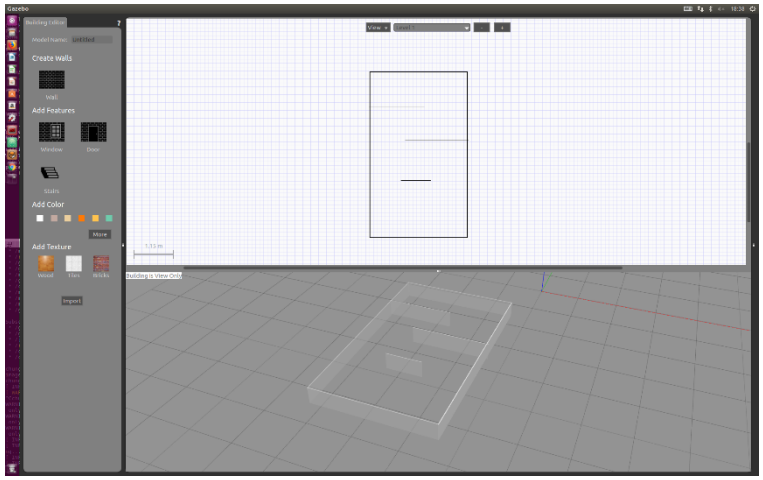


Right click the wall and open the wall inspector

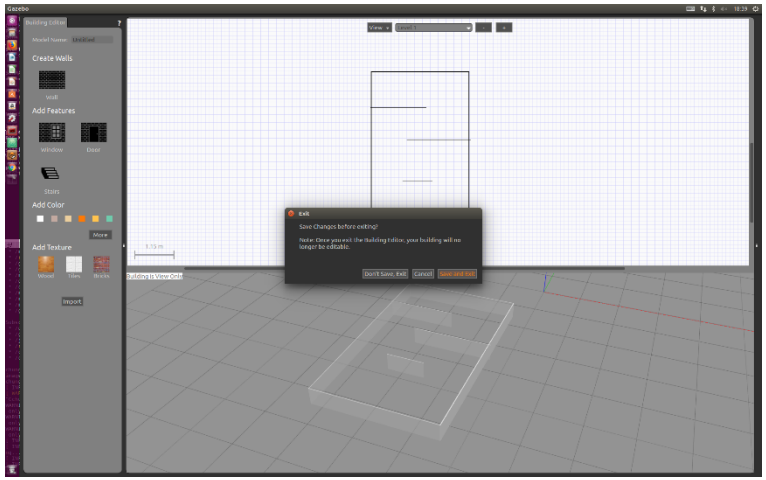


Making a structure

Set the height as 0.4m and thickness as 0.02m

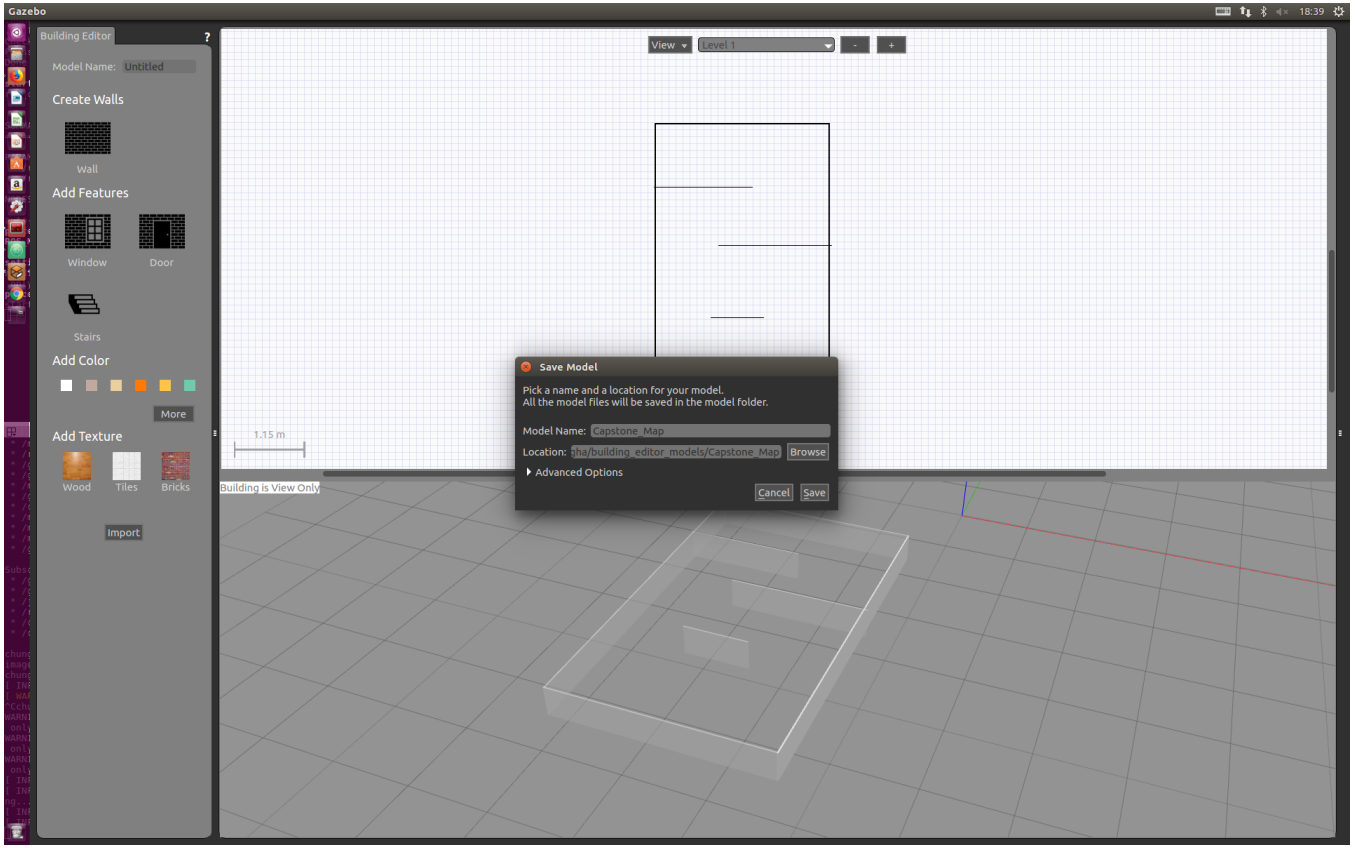


Close the building editor by clicking File -> Exit building Editor or by ctrl+x



Press Save and Exit

Making a structure



Set the model name as “Capstone_Map” and press Save
Then the model will be saved in ~/building_editor_models

Making a structure

Copy the building model into gazebo folder

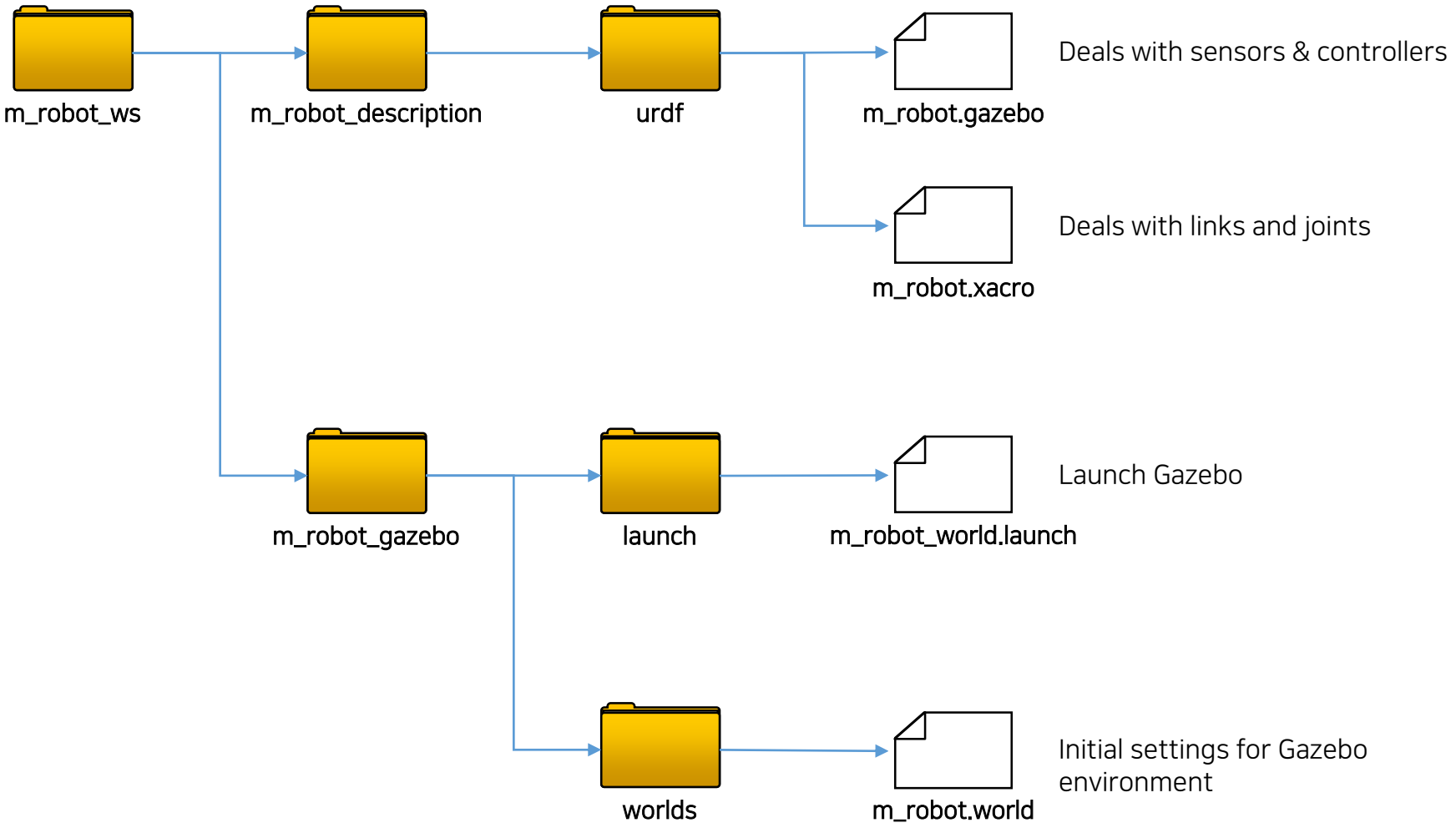
```
~$ cd ~/building_editor_models
~$ cp -r Capstone_Map ~/.gazebo/models
~$ cd ~/.gazebo/models
~$ ls
```

- Goes to building_editor_models folder
- Copy the building model into gazebo model folder

Then you will see :

```
Capsonte_Map
```

Overall structure



This lecture note is based on the contents in http://moorerobots.com/

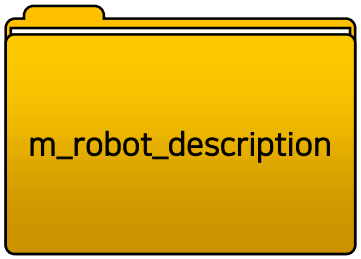
Creating workspace

Create a workspace (name it whatever you want)

```
~$ cd                                  - Goes to home directory
~$ mkdir capstone_simulation           - Make a folder named “capstione_simulation”
~$ cd capstone_simiulation             - Goes inside the folder “capstone_simulation”
~$ mkdir src                           - Important! : the folder name should be “src”
~$ cd src                              - Goes inside the folder “src”
~$ mkdir m_robot_ws                    - Make a folder named “m_robot_ws”
```

(tips : **cd** is for change directory, and **mkdir** is for make directory)

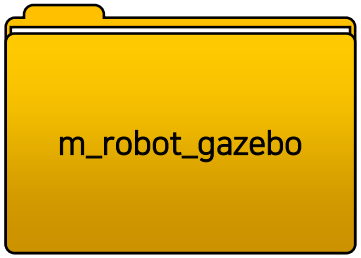
Now, we are going to make two catkin packages :



Contains the description of the robot

- Links : shape, color, mass, inertial, etc.
- Joints : joints between links (revolute joint, fixed joint, etc.)
- Sensors : encoder, camera, lidar, etc.

In this lecture, we are going to use URDF (Unified Robot Description Format)
(Another way of describing the robot : sdf format)



Contains the settings in Gazebo

- Light, view, and other environmental factors in Gazebo
- Robot’s initial position
- Positions of other objects that should be included

Creating package

Create two catkin packages

You are now at ~/capstone_simulation/src

```
~$ cd m_robot_ws                                  - Goes inside the folder “m_robot_ws”
~$ catkin_create_pkg m_robot_description          - Create “m_robot_description” catkin package
~$ catkin_create_pkg m_robot_gazebo               - Create “m_robot_gazebo” catkin package
```

Before we make changes, let’s include the source to bashrc

```
~$ cd ~/capstone_simulation                       - Goes back to capstone_simulation directory
~$ catkin_make                                    - Build the source files using catkin_make
~$ ls                                             - List segments in “capstone_simulation”
```

(tips : **~/** indicates the home directory, **ls** means “list segments”)

Then you will see :

```
build  devel  src
```

In devel folder, you can also see “setup.bash” file
Let’s include this “setup.bash” file in bashrc

```
~$ echo “source ~/capstone_simulation/devel/setup.bash” >> ~/.bashrc
```

- Add a line “source ~/capstone_simulation/devel/setup.bash” into bashrc

Robot description

Now, let's create some source files in m_robot_description

You are now at ~/capstone_simulation

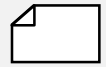
```
~$ cd src/m_robot_ws/m_robot_description
~$ mkdir urdf
~$ gedit m_robot.xacro
```

- Goes inside the folder "m_robot_description"
- Create a folder named "urdf"
- Open "m_robot.xacro" using gedit

Or with atom, or any other program you ever want

(tips : m_robot.gazebo file is not created unless you save your file)

m_robot.xacro file uses a xml file format which is similar to html
The links and joints are described within <robot> class that is as :



m_robot.xacro

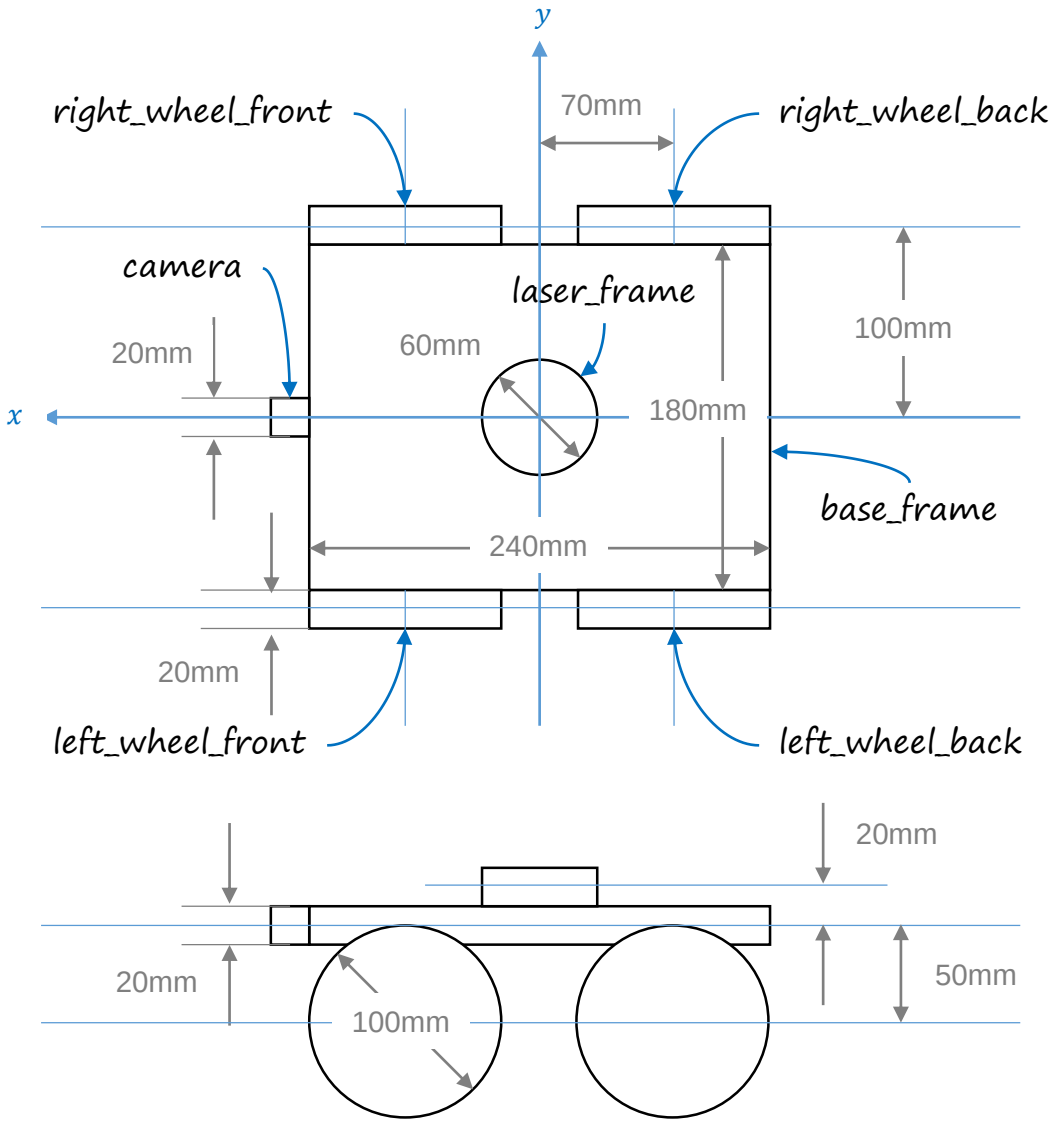
```
<?xml version='1.0'?>
<robot name = "m_robot" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <link name = "link 1"
  <!-- link description -->
  </link>
  ...
  <joint type = "continuous" name = "joint1">
  <!-- joint description -->
  </joint>
  ...
</robot>
```

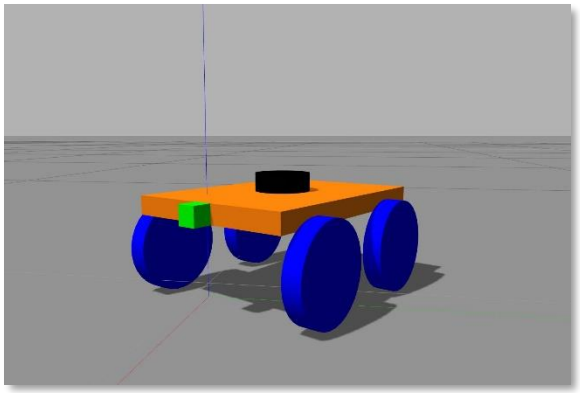
Notice that the classes are closed with /
<classname> : starts class
</classname> : closes class

Some classes open and close at the same line
<classname param1 = "val1" param2 = "val2" ⋯ />

Robot description

Design the robot yourself before making it





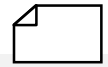
base_frame
- $m = 5.0kg$
- Inertia
  - $I_{xx} = 0.0135kg \cdot m^2$
  - $I_{yy} = 0.0240kg \cdot m^2$
  - $I_{zz} = 0.0375kg \cdot m^2$

wheel
- $m = 2.5kg$
- Inertia
  - $I_{xx} = 0.00165kg \cdot m^2$
  - $I_{yy} = 0.00165kg \cdot m^2$
  - $I_{zz} = 0.00313kg \cdot m^2$

Camera and lidar
- Mass & Inertia : all zero

The link names are used later in other packages, so try not to change them

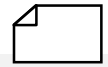
Robot description : base_frame link



m_robot.xacro

```
<?xml version='1.0'?>
<robot name = "m_robot" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <link name='base_frame'>                          -  Open the <link> class, name it “base_frame”
    <pose>0 0 0.1 0 0 0</pose>                      -  Pose of the link in world coordinate system
    <inertial>                                                (x,y,z,roll,pitch,yaw)
      <mass value="5.0"/>                           -  Set the mass value as 5.0kg
      <origin xyz="0.0 0 0.0" rpy=" 0 0 0"/>        -  The frame where the origin is defined
      <inertia                                                (w.r.t the base_frame)
          ixx="0.0135" ixy="0" ixz="0"              -  Set the inertial values
          iyy="0.024" iyz="0"
          izz="0.0375"/>
    </inertial>
    <collision name='base_frame_collision'>         -  Collision class is for defining the link which
      <origin xyz="0 0 0" rpy=" 0 0 0"/>               dynamically interacts within the simulator
      <geometry>                                    -  Description of the collision area
        <box size=".24 .18 .02"/>
      </geometry>
    </collision>
    <visual name='base_frame_visual'>               -  Visual class is for visualizing the link
      <origin xyz="0 0 0" rpy=" 0 0 0"/>            -  Set the origin and geometry same as collision
      <geometry>                                    -  Description of how the base_frame looks like
        <box size=".24 .18 .02"/>
      </geometry>
    </visual>
  </link>                                           -  The units are meter for distance, and
…                                                      radian for angle
…
```

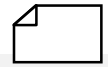
Robot description : left_wheel_front link



m_robot.xacro

```
…
…
  <link name="left_wheel_front">
    <collision name="left_wheel_front_collision">
      <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/> -  Rotate the wheel 90 degrees in yaw and pitch
      <geometry>
        <cylinder radius="0.05" length="0.02"/>   -  Notice the geometry type is cylinder
      </geometry>
    </collision>
    <visual name="left_wheel_front_visual">
      <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/> -  Rotate the wheel 90 degrees in yaw and pitch
      <geometry>
        <cylinder radius="0.05" length="0.02"/>
      </geometry>
    </visual>
    <inertial>
      <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/> -  Rotate the wheel 90 degrees in yaw and pitch
      <mass value="2.5"/>
      <cylinder_inertia m="2.5" r="0.05" h="0.02"/>
      <inertia
        ixx="0.001646" ixy="0.0" ixz="0.0"
        iyy="0.001646" iyz="0.0"
        izz="0.003125"/>
    </inertial>
  </link>…
…                                                -  Repeat this for other wheels
```

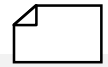
Robot description : camera link



m_robot.xacro

```
…
…
  <link name="camera">
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.02 0.02 0.02"/>
      </geometry>
    </collision>

    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.02 0.02 0.02"/>
      </geometry>
    </visual>

    <inertial>
      <mass value="0" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0" />
    </inertial>
  </link>
…
…
```

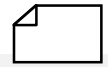
Robot description : laser_frame link



m_robot.xacro

```
...
...
  <link name="laser_frame">
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <cylinder radius=".03" length=".02"/>
      </geometry>
    </collision>

    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <cylinder radius=".03" length=".02"/>
      </geometry>
    </visual>

    <inertial>
      <mass value="0" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0=" />
    </inertial>
  </link>
...
...
```

Robot description : joints



m_robot.xacro

```
…
…
  <joint type="continuous" name="left_wheel_front_hinge">
    <origin xyz="0.07 0.1 -0.05" rpy="0 0 0"/>
    <child link="left_wheel_front"/>
    <parent link="base_frame"/>
    <axis xyz="0 1 0" rpy="0 0 0"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>
…
…
…
  <joint name="camera_joint" type="fixed">
    <origin xyz=".13 0 0" rpy="0 0 0"/>
    <parent link="base_frame"/>
    <child link="camera"/>
    <axis xyz="0 1 0" rpy="0 0 0"/>
  </joint>
…
  <joint name="lidar_joint" type="fixed">
    <axis xyz="0 0 1" />
    <origin xyz="0 0 0.02" rpy="0 0 0"/>
    <parent link="base_frame"/>
    <child link="laser_frame"/>
  </joint>
</robot>
```

Wheel joint
- Open joint with continuous type
- Locate it w.r.t the parent link
- Set the child link
- Set the parent link
- Set the rotation axis
- Parameters (such as PID controller)
- Damping and friction properties
- Close the joint class

Repeat it for all wheels

Camera joint
- Open joint with fixed type
- Locate it w.r.t the parent link

Lidar joint

- Close the robot class

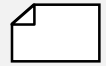
Robot description : sensors & controllers

Now, save the m_robot.xacro file, and open m_robot.gazebo file with gedit

You are now at ~/capstone_simulation/src/m_robot_ws/m_robot_description

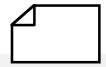
```
~$ gedit m_robot.gazebo
```

m_robot.gazebo file also uses a xml file format


m_robot.gazebo

```
<?xml version='1.0'?>
<robot>
  <gazebo>
    <plugin name="skid_steer_drive_controller" filename="libgazebo_ros_skid_steer_drive.so">
      …                                          -  This plugin is used to control 4-wheeled robot
    </plugin>
  </gazebo>
…
  <gazebo reference="camera">
    <sensor type="camera" name="camera1">        -  Setup a camera sensor on the “camera” link
      …                                          -  and name it as “camera1”
    </sensor>
  </gazebo>
…
 <gazebo reference="laser_frame">
    <sensor type="ray" name="lidar_sensor">      -  Setup a ray sensor on the “laser_frame” link
    </sensor>                                    -  and name it as “lidar_sensor”
  </gazebo>
</robot>
```

Robot description : controller – skid steer drive controller



m_robot.gazebo

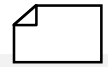
```
…
…
  <gazebo>
    <plugin name="skid_steer_drive_controller" filename="libgazebo_ros_skid_steer_drive.so">

      <updateRate>100</updateRate>                                  -  Control rate

      <leftFrontJoint>left_wheel_front_hinge</leftFrontJoint>       -  Assign the joints to
      <leftRearJoint>left_wheel_back_hinge</leftRearJoint>             Corresponding joint in
      <rightFrontJoint>right_wheel_front_hinge</rightFrontJoint>       skid steer drive controller
      <rightRearJoint>right_wheel_back_hinge</rightRearJoint>

      <wheelSeparation>0.2</wheelSeparation>                        -  Set the wheel separation
      <wheelDiameter>0.10</wheelDiameter>                           -  Set the wheel diameter
      <torque>8</torque>                                            -  Set the torque of the wheel
      <commandTopic>cmd_vel</commandTopic>                          -  Topic mane
                                                                       This will make the node to
                                                                       subscribe to “cmd_vel” topic
                                                                       which the message type is
                                                                       geometry_msgs/twist

      <robotBaseFrame>base_frame</robotBaseFrame>                   -  Assign the robot base frame
    </plugin>                                                       -  Close plugin class
  </gazebo>                                                         -  Close gazebo class
…
For more information about gazebo plugins, please refer to :
          http://gazebosim.org/tutorials?tut=ros_gzplugins
```

Robot description : sensor – camera

m_robot.gazebo

```
...
...
  <gazebo reference="camera">
    <material>Gazebo/Green</material>                  -  Set the camera link color
    <sensor type="camera" name="camera1">              -  Sensor type and its name
      <update_rate>30.0</update_rate>                  -  FPS of the camera
      <camera name="head">                             -  Camera name
        <horizontal_fov>1.3962634</horizontal_fov>     -  Set the field of view [rad]
        <image>
          <width>800</width>                           -  Image size
          <height>800</height>
          <format>R8G8B8</format>                      -  Image format
        </image>
        <clip>                                         -  Clipping parameters
          <near>0.05</near>                               see this
          <far>300</far>
        </clip>
      </camera>
      <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
        <alwaysOn>true</alwaysOn>                      -  Setup the camera plugin
        <cameraName>m_robot/camera1</cameraName>
        <imageTopicName>image_raw</imageTopicName>     -  Topic name
      </plugin>                                        Image:= /m_robot/camera1/image_raw
    </sensor>
  </gazebo>
...
```

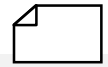
Robot description : sensor – lidar



m_robot.gazebo

```
…
…                                                                    Lidar is programmed based on rplidar A1 model
  <gazebo reference="laser_frame">
    <material>Gazebo/Black</material>                                -   Set the laser_frame link color
    <sensor type="ray" name="lidar_sensor">                          -   Sensor type and its name
      <visualize>false</visualize>                                   -   If true, you can see the ray in Gazebo
      <update_rate>5.5</update_rate>                                 -   Update rate
      <ray>
        <scan>
          <horizontal>
            <samples>359</samples>                                   -   Number of samples
            <min_angle>0</min_angle>                                 -   Minimum angle : 0 degree
            <max_angle>6.265732015</max_angle>                       -   Maximum degree : 359 degree
          </horizontal>
        </scan>
        <range>
          <min>0.15</min>                                            -   Minimum detectable range : 15cm
          <max>6.0</max>                                             -   Maximum detectable range : 6m
          <resolution>0.005</resolution>                             -   Resolution
        </range>
        <noise>                                                      -   Add some Gaussian noise
          <type>gaussian</type>
          <mean>0.0</mean>
          <stddev>0.02</stddev>
        </noise>
      </ray>
      <plugin name="gazebo_ros_lidar_controller" filename="libgazebo_ros_laser.so">
                                                                     -   Setup the lidar plugin
        <topicName>/m_robot/laser/scan</topicName>                   -   Topic name
      </plugin>
    </sensor>
</gazebo>
…
```

Robot description : material colors



m_robot.gazebo

```
...
...
  <gazebo reference="base_frame">
    <material>Gazebo/Orange</material>
  </gazebo>

  <gazebo reference="left_wheel_front">
    <material>Gazebo/Blue</material>
  </gazebo>

  <gazebo reference="right_wheel_front">
    <material>Gazebo/Blue</material>
  </gazebo>

  <gazebo reference="left_wheel_back">
    <material>Gazebo/Blue</material>
  </gazebo>

  <gazebo reference="right_wheel_back">
    <material>Gazebo/Blue</material>
  </gazebo>
</robot>
```

- Setup the link colors

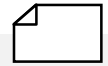
- Save and close gedit

Setting up Gazebo environment

Now, let's create some source files in m_robot_gazebo

```
~$ cd ~/capstone_simulation/src/m_robot_ws/m_robot_gazebo
~$ mkdir launch worlds                          -  Create folders named "launch" and "worlds"
~$ cd worlds
~$ gedit m_robot.world
```

Setup for Gazebo environment



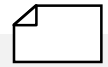
m_robot.world

```
<?xml version="1.0" ?>
<sdf version="1.4">                                 - Open sdf class
  <world name="default">                            - Open world class named "default"
    <include>
      <uri>model://ground_plane</uri>               - Include a "ground_plane" model
    </include>
    <include>
      <uri>model://sun</uri>                        - Include a light source model "sun"
    </include>
    <gui fullscreen='0'>
      <camera name='user_camera'>
        <pose>0 0 7 0 1.570796 0</pose>             - Set the camera pose (user view)
        <view_controller>orbit</view_controller>
      </camera>
    </gui>
</world>                                            - Close world class
</sdf>                                              - Close sdf class
```

Setting up Gazebo launch file

```
~$ cd ~/capstone_simulation/src/m_robot_ws/m_robot_gazebo
~$ cd launch
~$ gedit m_robot_world.launch
```

Setup the Gazebo launch file




```
<launch>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find m_robot_gazebo)/worlds/m_robot.world"/>
    <arg name="debug" value="false" />
    <arg name="gui" value="true" />
    <arg name="paused" value="false"/>                    -   Setup the Gazebo startup parameters
    <arg name="use_sim_time" value="true"/>
    <arg name="headless" value="false"/>
  </include>
  <param name="robot_description"
    command="$(find xacro)/xacro.py '$(find m_robot_description)/urdf/m_robot.xacro'" />
                                                          -   Assign the robot description to m_robot.xacro
  <node name="m_robot_spawner" pkg="gazebo_ros" type="spawn_model" output="screen"
    args="-urdf -param robot_description -model m_robot "/>
                                                          -   Spawn the model into Gazebo
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
    <param name="use_gui" value="FALSE"/>                 -   Joint_state_publisher sends joint states to
  </node>                                                     robot_state_publisher
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>
</launch>                                                 -   Then the robot_state_publisher sends tf of the robot
                                                            (topic name : tf)
```

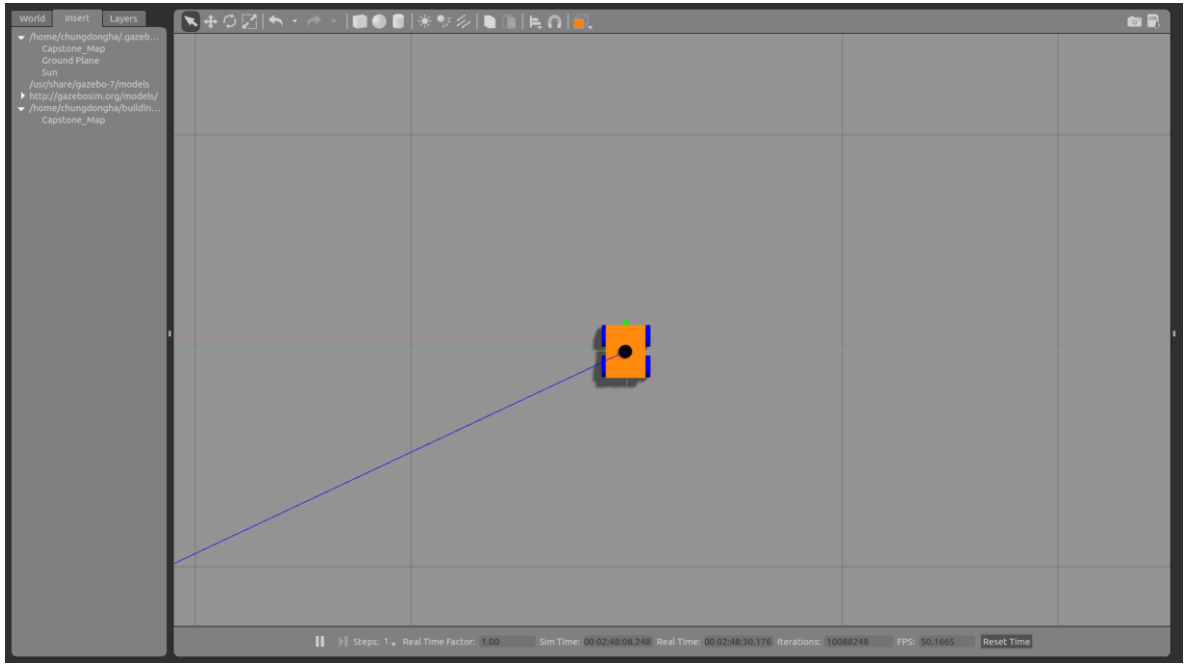
Check the launch file

Build the project

```
                                                    You are now at ~/capstone_simulation
~$ catkin_make                                      -  Build the project
~$ roscore
```

Open another terminal and launch m_robot_world.launch

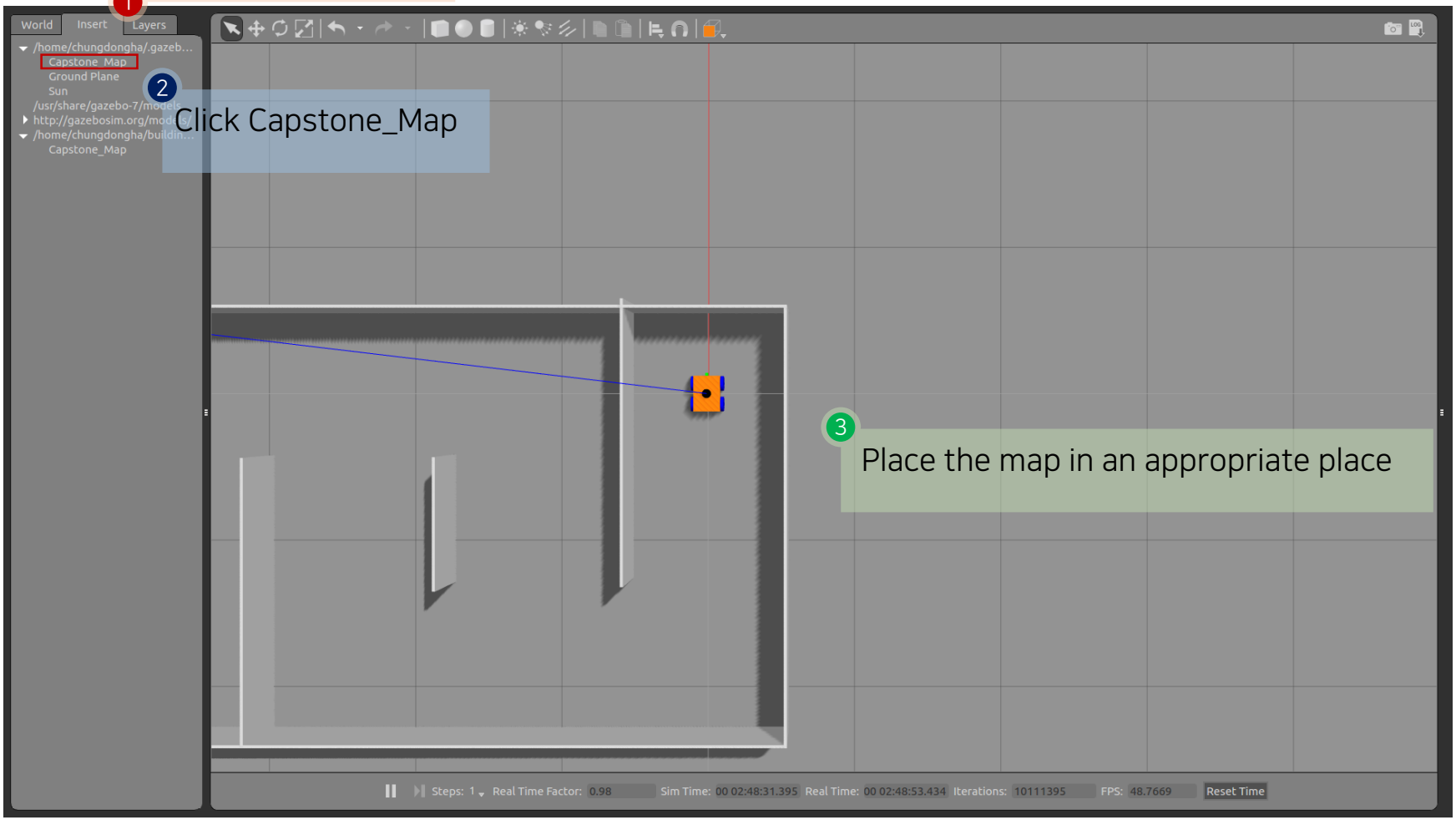
```
~$ roslaunch m_robot_gazebo m_robot_world.launch
```



Now, let's insert the map

Setting the map pose

1 Click the Insert tab

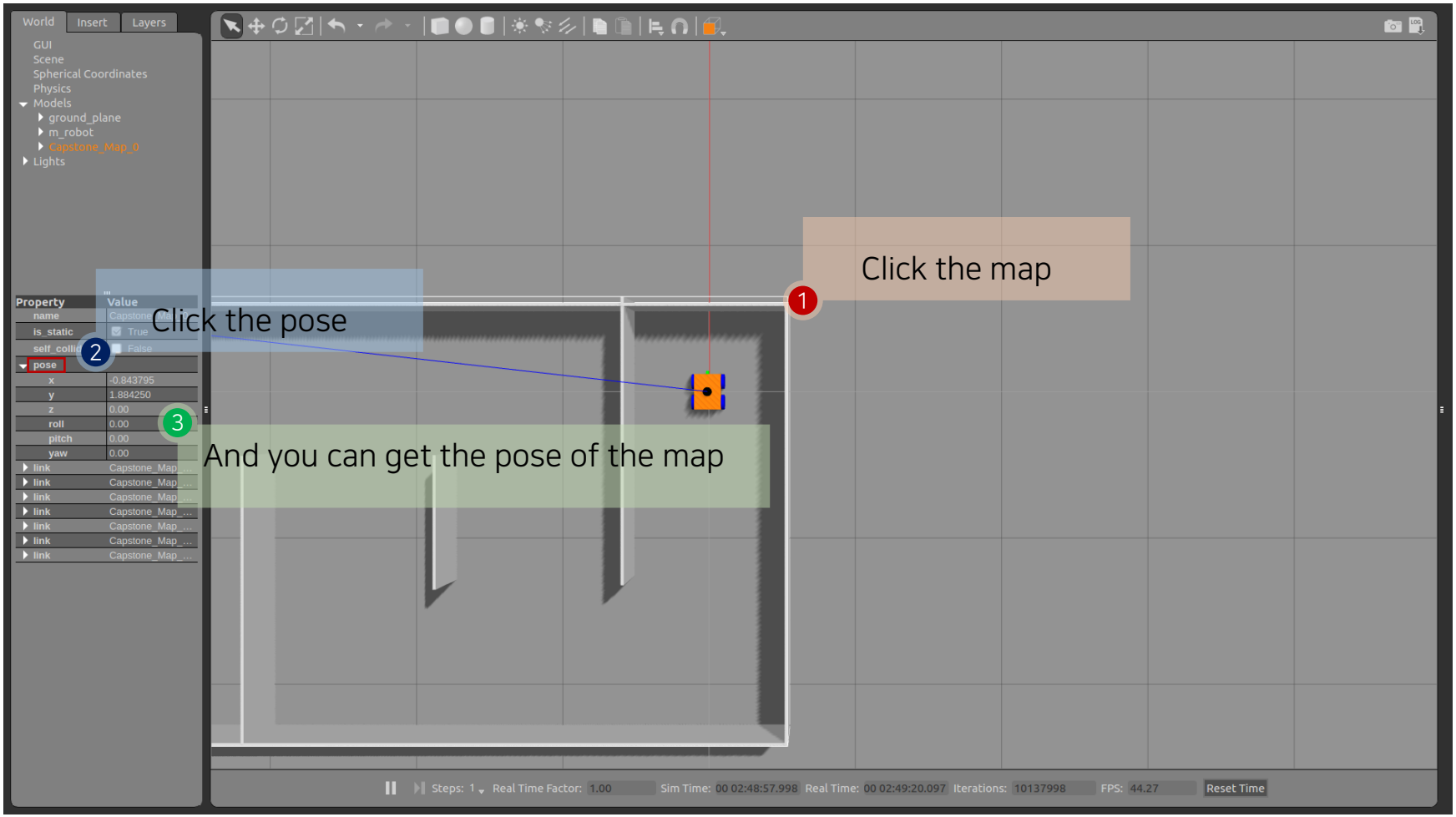


2 Click Capstone_Map

3 Place the map in an appropriate place

Now, let's insert the map

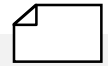
Setting the map pose



Now let's include the map in the launch file

Setting the map model in launch file

```
~$ cd ~/capstone_simulation/src/m_robot_ws/m_robot_gazebo/worlds
~$ gedit m_robot.world
```



m_robot.world

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://sun</uri>
    </include>
    <gui fullscreen='0'>
      <camera name='user_camera'>
        <pose>0 0 7 0 1.570796 0</pose>
        <view_controller>orbit</view_controller>
      </camera>
    </gui>
    <include>
      <uri>model://Capstone_Map</uri>              - Include the map model
      <pose>-1.0 2.0 0.0 0.0 0.0 0.0</pose>        - Set the model pose in world coordinate system
    </include>
</world>
</sdf>
```

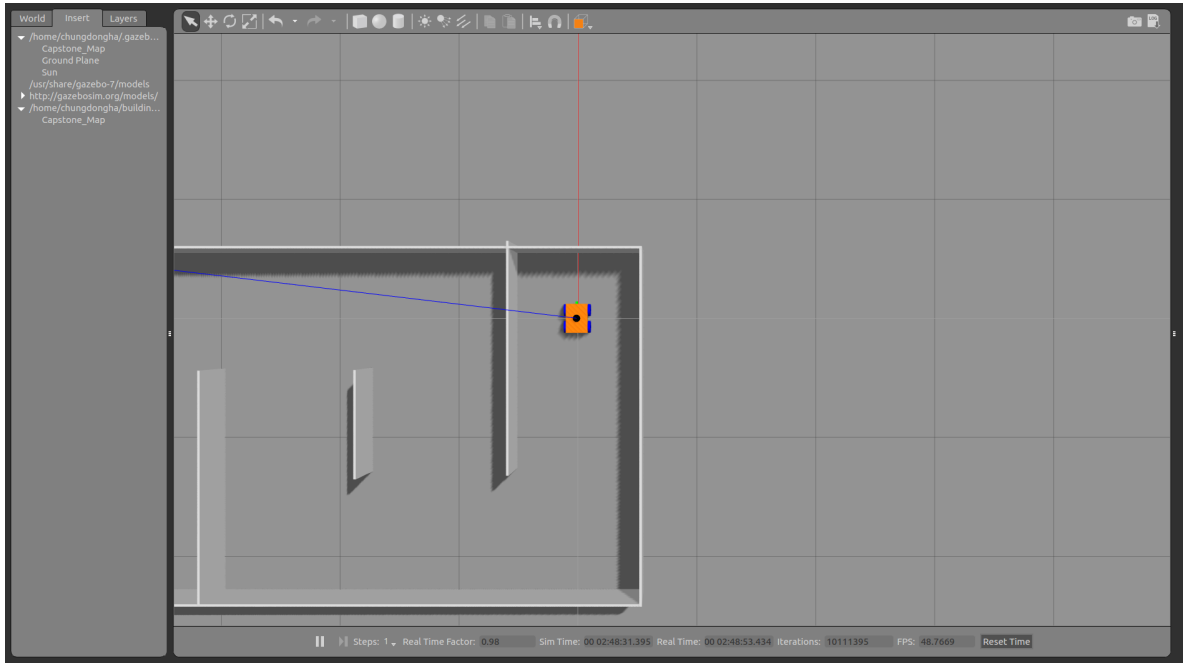
Check the launch file

Build the project

```
                                        You are now at ~/capstone_simulation
~$ catkin_make                          -  Build the project
~$ roscore
```

Open another terminal and launch m_robot_world.launch

```
~$ roslaunch m_robot_gazebo m_robot_world.launch
```

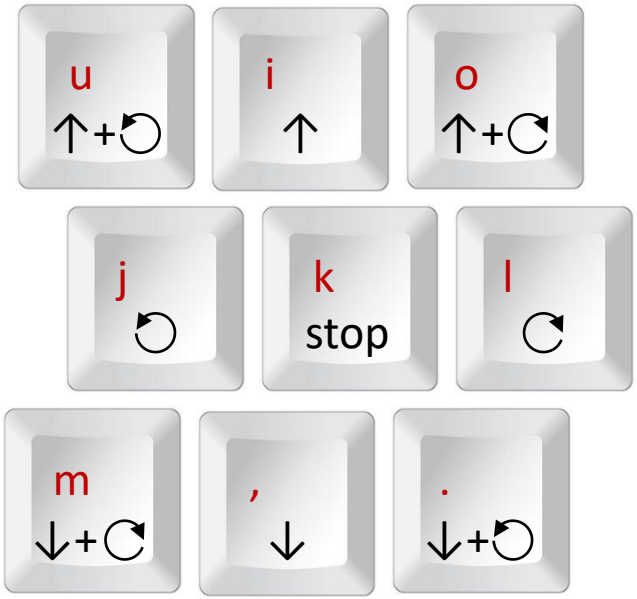


All set, ready to go

Check the controllers

```
~$ rosrun teleop_twist_keyboard teleop_twist_keyboard
```

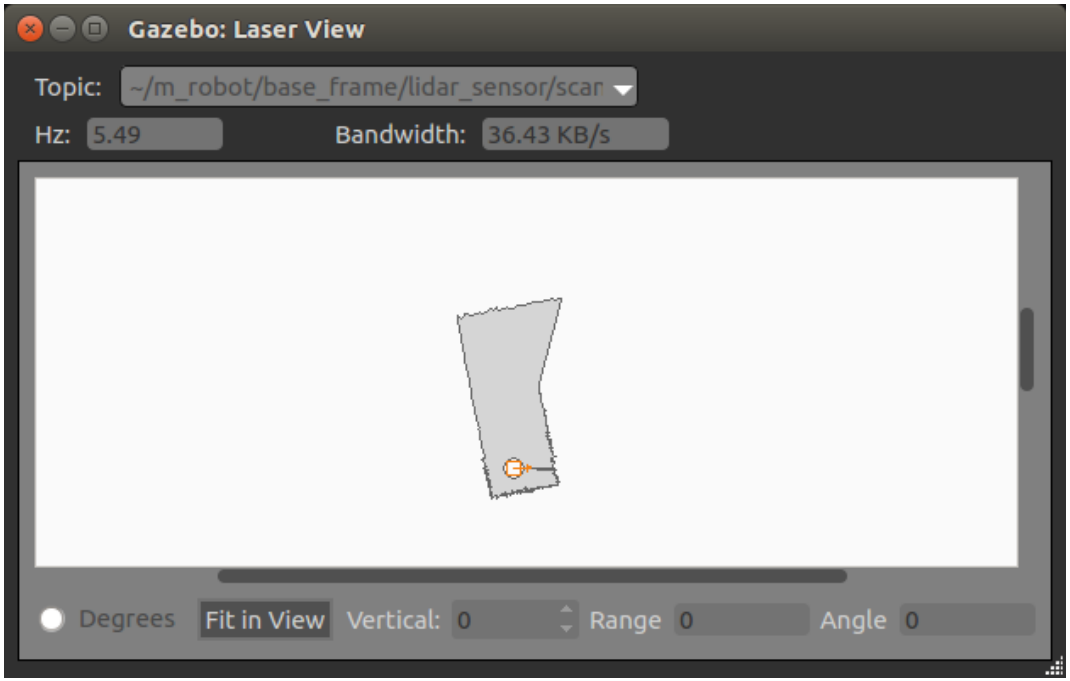
If you don't have teleop_twist_keyboard package, install it from :
http://wiki.ros.org/teleop_twist_keyboard



All set, ready to go

Check the sensors : In Gazebo

```
Window -> Topic Visualization (or ctrl+t)
Lidar : Double click the topic under gazebo.msgs.LaserScanStamped
Camera : Double click the topic under gazebo.msgs.ImageStamped
```



Also check the topics in the terminal

```
~$ rostopic list -v
```

02

Simulating SLAM using Hector SLAM package

KAIST Department of Mechanical Engineering

T.A. Dongha Chung chungdongha@kaist.ac.kr

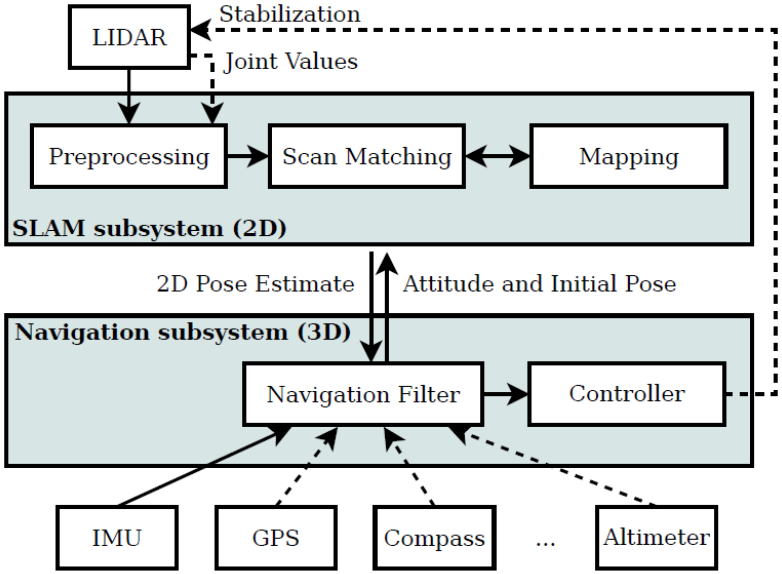
About Hector SLAM

RoboCup 2011 Rescue Arena

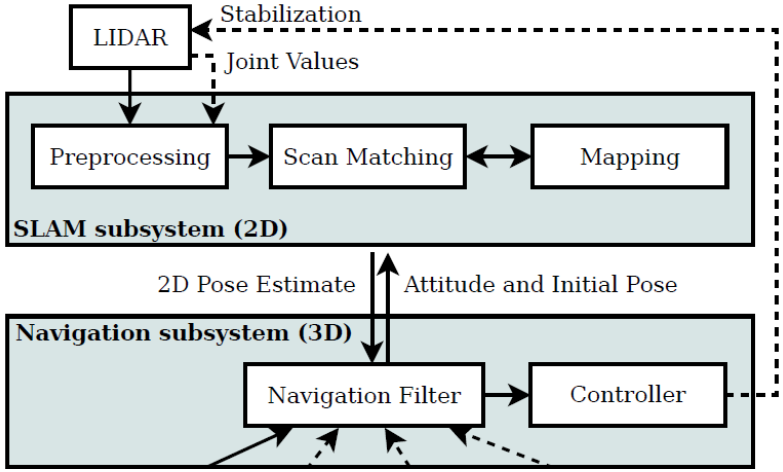
Youtube : https://www.youtube.com/watch?v=F8pdObV_df4&list=PL0E462904E5D35E29

About Hector SLAM

Overall Algorithm



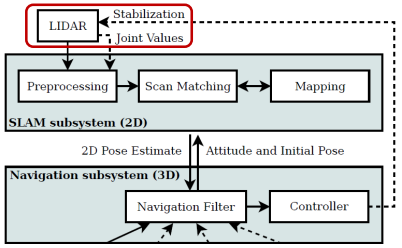
Hector SLAM framework



Stefan Kohlbrecher, et. al. A flexible and scalable SLAM system with full 3D motion estimation
2011 IEEE International Symposium on Safety, Security, and Rescue Robotics

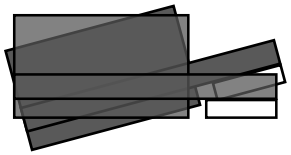
About Hector SLAM

Algorithm Step 1. Transformation of scan data to “satablized frame”



Lidar Scan Data

Lidar scan data



Attitude data

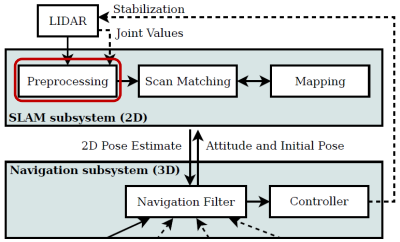
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R^{-1} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

2D Scan data in Stabilized Frame

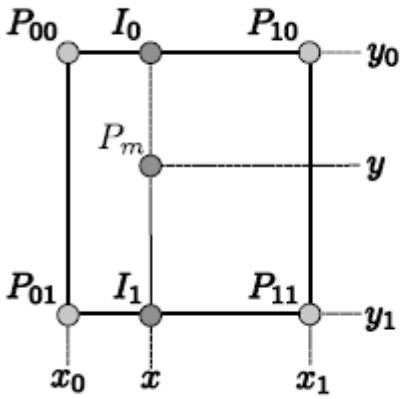
Lidar attitude data (ϕ, θ, ψ)

Stefan Kohlbrecher, et. al. A flexible and scalable SLAM system with full 3D motion estimation
2011 IEEE International Symposium on Safety, Security, and Rescue Robotics

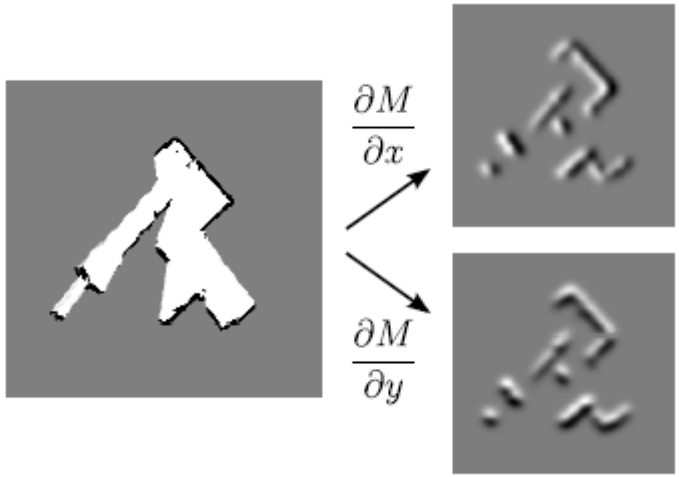
About Hector SLAM



Algorithm Step 2. Interpolation of occupancy grid map



Point to be approximated



Gradient of the occupancy grid map

Discrete occupancy grid map → Continuous occupancy grid map

$$M(P_m) \approx \frac{y-y_0}{y_1-y_0}\left(\frac{x-x_0}{x_1-x_0}M(P_{11})+\frac{x_1-x}{x_1-x_0}M(P_{01})\right) + \frac{y_1-y}{y_1-y_0}\left(\frac{x-x_0}{x_1-x_0}M(P_{10})+\frac{x_1-x}{x_1-x_0}M(P_{00})\right)$$

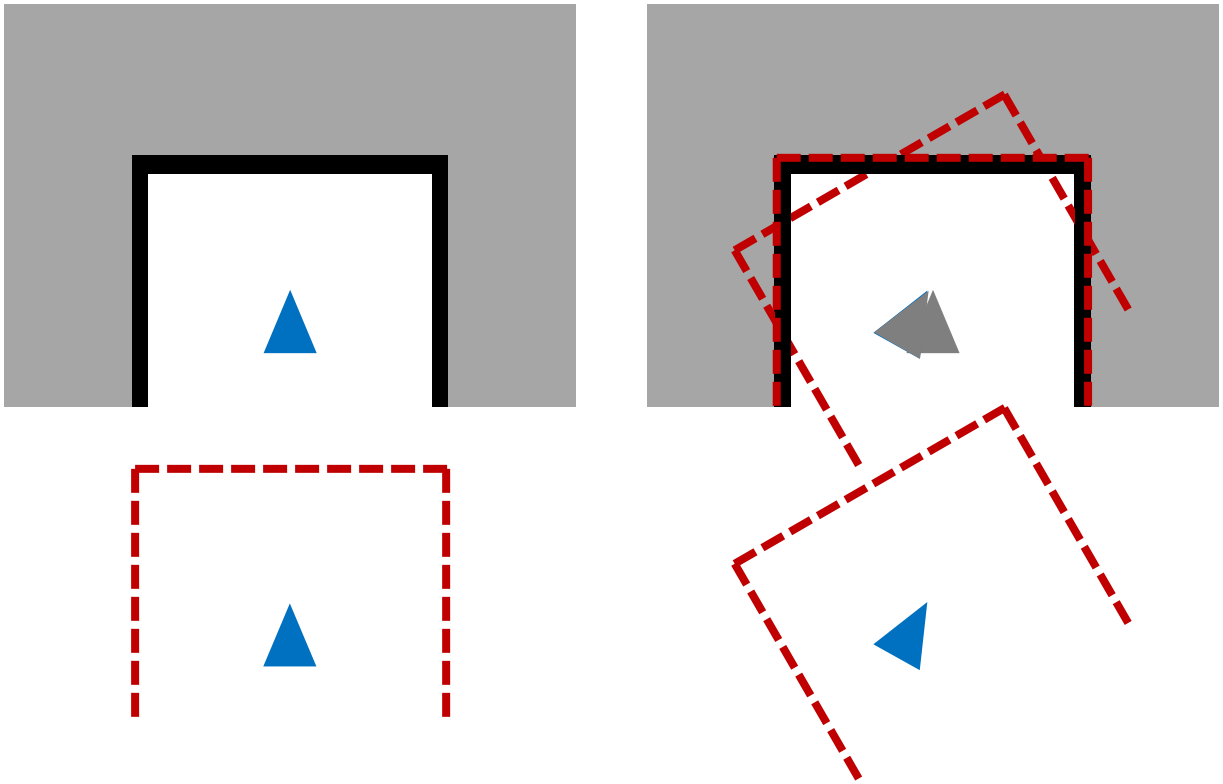
$$\frac{\partial M}{\partial x}(P_m) \approx \frac{y-y_0}{y_1-y_0}(M(P_{11})-M(P_{01})) + \frac{y_1-y}{y_1-y_0}(M(P_{10})-M(P_{00}))$$

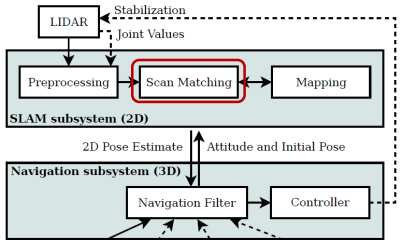
$$\frac{\partial M}{\partial y}(P_m) \approx \frac{x-x_0}{x_1-x_0}(M(P_{11})-M(P_{10})) + \frac{x_1-x}{x_1-x_0}(M(P_{01})-M(P_{00}))$$

Stefan Kohlbrecher, et. al. A flexible and scalable SLAM system with full 3D motion estimation
2011 IEEE International Symposium on Safety, Security, and Rescue Robotics

About Hector SLAM

Algorithm Step 3. Scan matching





Error covariance

$$\mathbf{H} = \left[\nabla M(\mathbf{S}_i(\boldsymbol{\xi}))\frac{\partial \mathbf{S}_i(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right]^T \left[\nabla M(\mathbf{S}_i(\boldsymbol{\xi}))\frac{\partial \mathbf{S}_i(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right]$$

$$\mathbf{R} = \mathrm{Var}\{\boldsymbol{\xi}\} = \sigma^2 \cdot \mathbf{H}^{-1}$$

$$\boldsymbol{\xi} = (p_x, p_y, \psi)^{\mathrm{T}} \longrightarrow \boldsymbol{\xi}^* = \underset{\boldsymbol{\xi}}{\operatorname{argmin}} \sum_{i=1}^{n} \left[1 - M(\mathbf{S}_i(\boldsymbol{\xi}))\right]^2$$

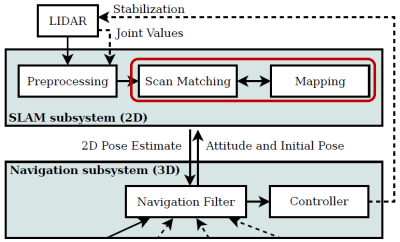
$$\mathbf{S}_i(\boldsymbol{\xi}) = \begin{pmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & cos(\psi) \end{pmatrix} \begin{pmatrix} s_{i,x} \\ s_{i,y} \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$

Gauss Newton method

Stefan Kohlbrecher, et. al. A flexible and scalable SLAM system with full 3D motion estimation
2011 IEEE International Symposium on Safety, Security, and Rescue Robotics

About Hector SLAM

Algorithm Step 3. Scan matching & Mapping



Multiresolution representation of the map

Scan Data $(s_x.s_y)$

Scan Data $(s_x.s_y)$
Pose (p_x^1, p_y^1, ψ^1)

Scan Data $(s_x.s_y)$
Pose (p_x^2, p_y^2, ψ^2)

Map $(M(x,y))$
Pose (p_x^f, p_y^f, ψ^f)

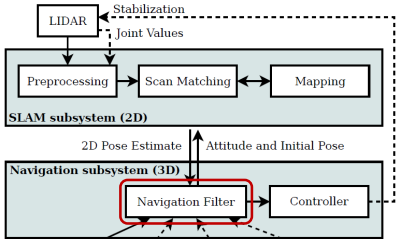
Pose (p_x^1, p_y^1, ψ^1)

Pose (p_x^2, p_y^2, ψ^2)

Stefan Kohlbrecher, et. al. A flexible and scalable SLAM system with full 3D motion estimation
2011 IEEE International Symposium on Safety, Security, and Rescue Robotics

About Hector SLAM

Algorithm Step 4. Navigation Filter



Measurement

$$z_{k+1} = h(x_{k+1}) + v_{k+1}$$

Time step k

$$\hat{x}_k, P_k$$

Kalman Gain

$$\tilde{y}_{k+1} = z_{k+1} - h(\hat{x}_{k+1}^-)$$
$$S_{k+1} = H_{k+1} P_{k+1}^- H_{k+1}^T + R_{k+1}$$
$$K_{k+1} = P_{k+1}^- H_{k+1}^T S_{k+1}^{-1}$$

Time step $k+1$

$$\hat{x}_{k+1} = \hat{x}_{k+1}^- + K_{k+1}\tilde{y}_{k+1}$$
$$P_{k+1} = (I - K_{k+1}H_{k+1})P_{k+1}^-$$

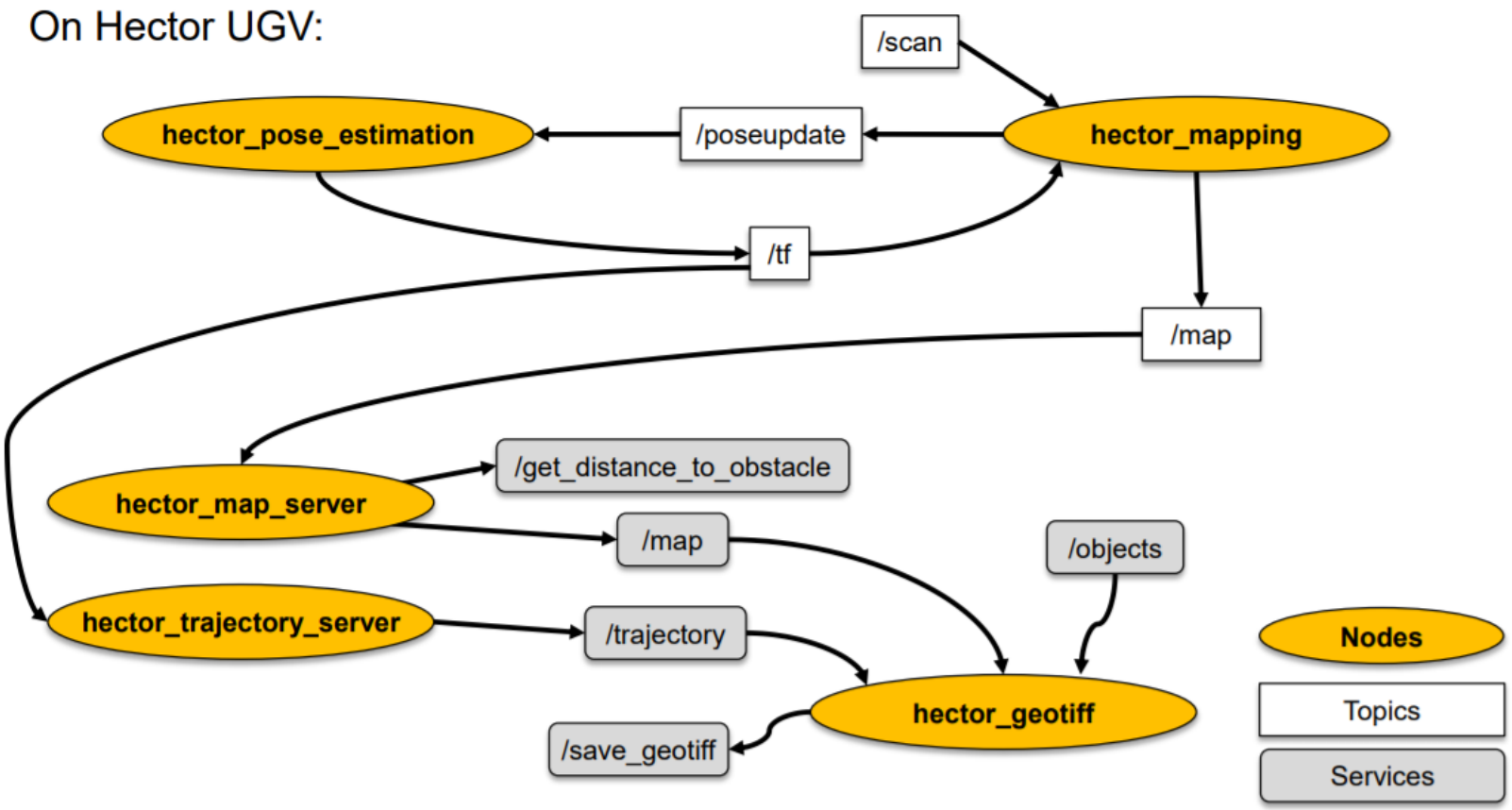
Prediction

$$\hat{x}_{k+1}^- = f(\hat{x}_k, u_k) + w_k$$
$$P_{k+1}^- = F_k P_k F_k^T + Q_k$$

Stefan Kohlbrecher, et. al. A flexible and scalable SLAM system with full 3D motion estimation
2011 IEEE International Symposium on Safety, Security, and Rescue Robotics

About Hector SLAM

Check the sensors : In Gazebo



Get Hector SLAM

Git clone from github

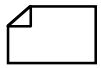
You are now at catkin_ws/src

```
~$ git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam
~$ cd hector_slam/hector_mapping/launch
~$ gedit mapping_default.launch
```

```
<?xml version="1.0"?>
<launch>
  <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/>
  Name to publish the tf between the scanmatcher and map
  <arg name="base_frame" default="base_footprint"/>
  Name of the base fame of the robot
  <arg name="odom_frame" default="nav"/>
  Name of the odometry frame
  <arg name="pub_map_odom_transform" default="true"/>
  Determine if the map to odometry tf should be published
  <arg name="scan_subscriber_queue_size" default="5"/>
  Queue size of the scan subscriber
  <arg name="scan_topic" default="scan"/>
  Name of the lidar scan topic
  <arg name="map_size" default="2048"/>
  Number of axis per axis (2048 X 2048 pixel)
  <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" output="screen">
…
</launch>
```

Change the parameters of Hector Mapping

Git clone from github

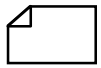


mapping_default.launch

```
…
  <param name="map_frame" value="map" />
  Name of the map_frame (to be published)
  <param name="use_tf_scan_transformation" value="true"/>
  Transformation of scan with attitude (Algorithm 1)
  <param name="use_tf_pose_start_estimate" value="false"/>
  Start with known pose
  <param name="map_resolution" value="0.050"/>
  Map resolution : 1 pixel = 0.05m
  <param name="map_start_x" value="0.5"/>
  <param name="map_start_y" value="0.5" />
  Starting point on the map (50%, 50%)
  <param name="map_multi_res_levels" value="2" />
  Multiresolution levels (Algorithm 3)
  <param name="update_factor_free" value="0.4"/>
  Factor to determine if the cell is free or not
  <param name="update_factor_occupied" value="0.9" />
  Factor to determine if the cell is occupied or not
  <param name="map_update_distance_thresh" value="0.4"/>
  Factor to determine if the map should be updated or not [m]
  <param name="map_update_angle_thresh" value="0.06" />
  Factor to determine if the map should be updated or not [rad]
…
```

Change the parameters of Hector Mapping

Git clone from github



mapping_default.launch

```
…
  <param name="laser_z_min_value" value = "-1.0" />
  Minimum height[m] relative to laser scanner frame
  <param name="laser_z_max_value" value = "1.0" />
  Minimum height[m] relative to laser scanner frame
…
</launch>
```

Change the parameters of Hector Mapping

For our robot simulation

```
~$ cd ~/me491_ros/src/find_frontier/launch
~$ gedit hector_mapping.launch
```

```
…
  <arg name="base_frame" default="base_frame"/>
  <arg name="odom_frame" default="base_frame"/>
  If there is no odometry frame, set it as base_frame
  <arg name="scan_topic" default="/m_robot/laser/scan"/>
  Gazebo is sending out topic /m_robot/laser/scan
  <arg name="map_size" default="200"/>
  Set map size as 10m by 10m (200*0.05 = 10)
  <param name="map_resolution" value="0.050"/>
  <param name="map_start_x" value="0.5"/>
  <param name="map_start_y" value="0.8"/>
  Starting point in the map (x : 50%, y : 80% of the total map)
…
```

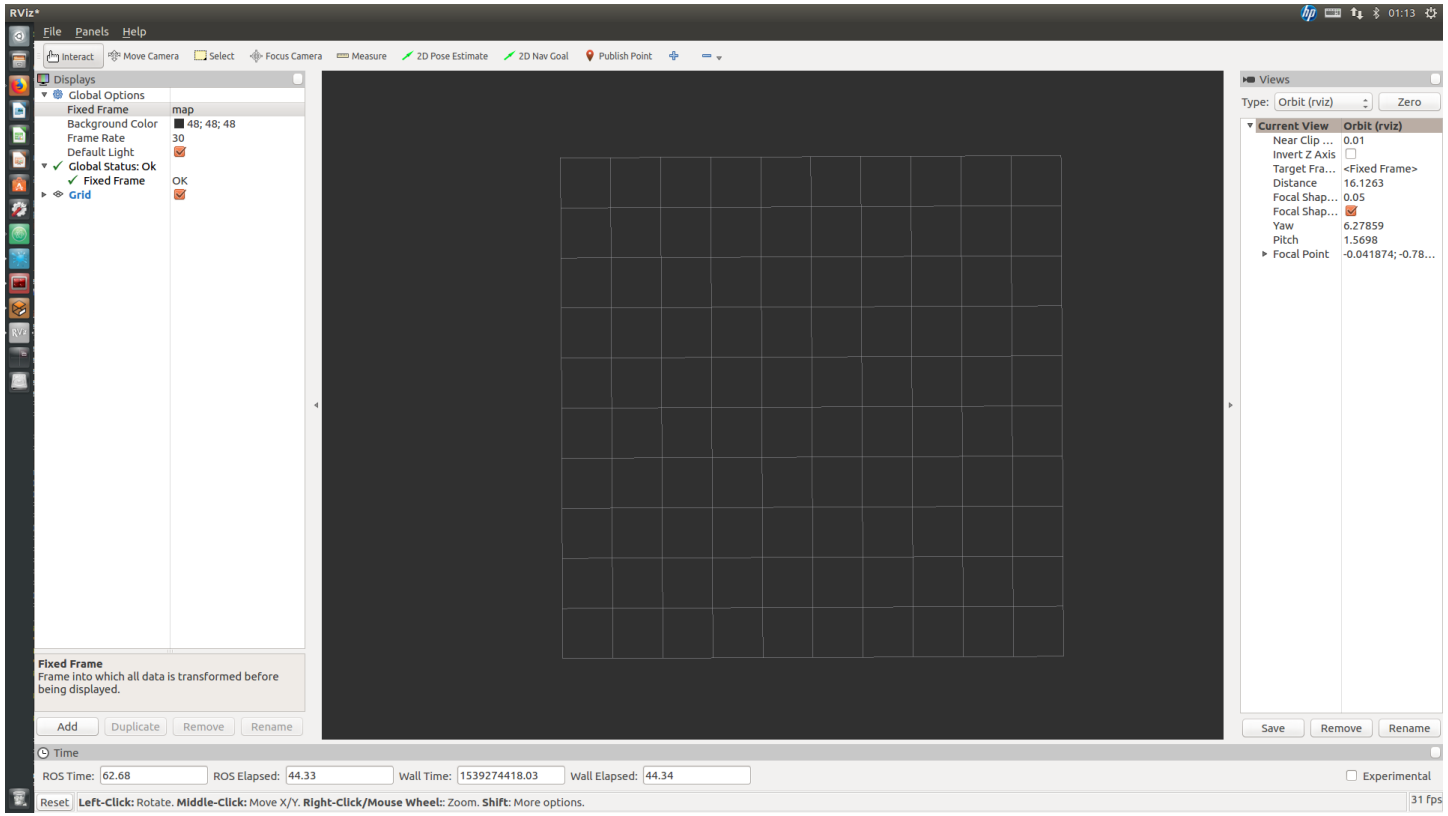
Now let's run it!

Visualization results in rviz

```
~$ roslaunch m_robot_gazebo m_robot_world.launch
```

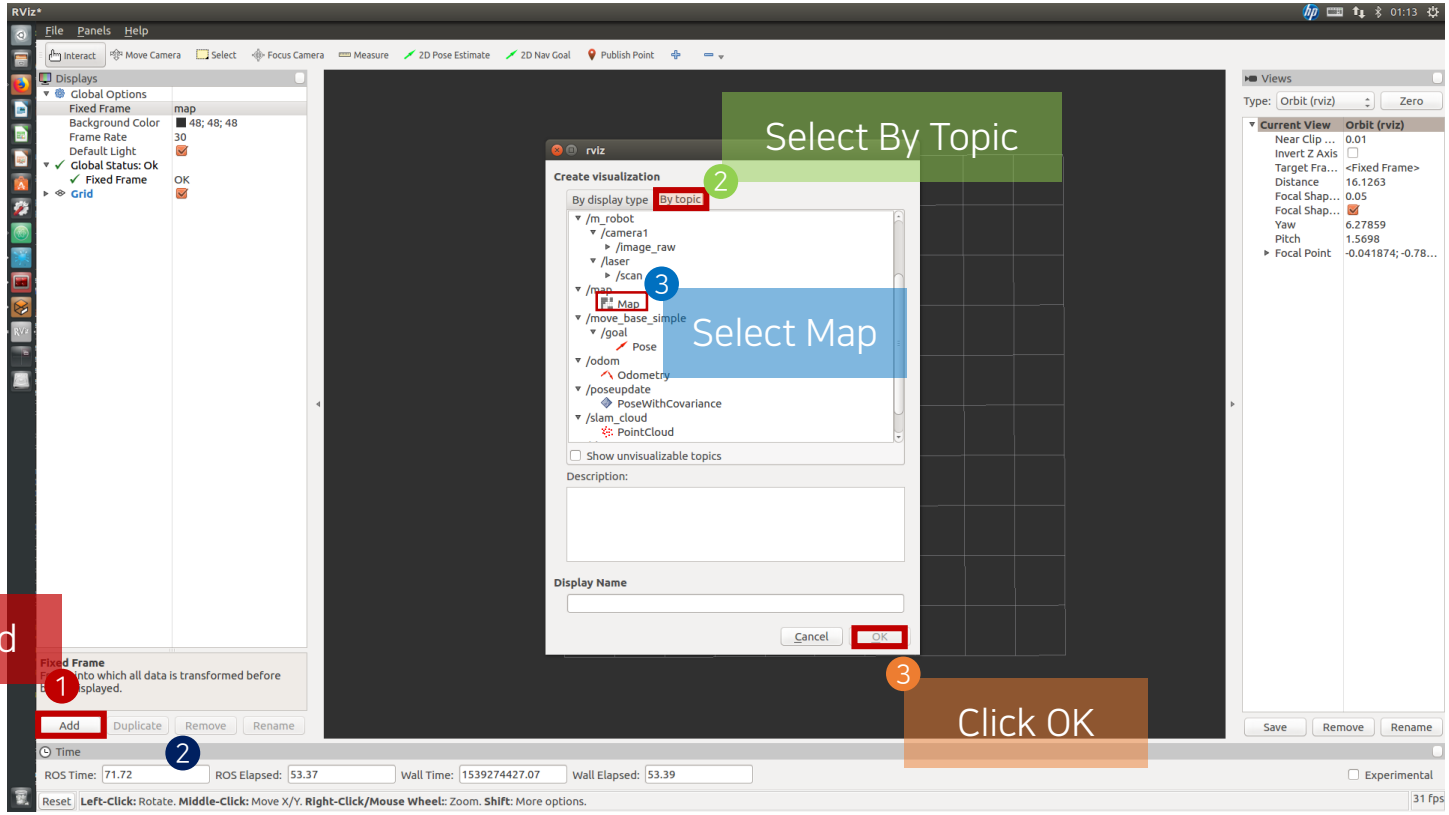
```
~$ roslaunch find_frontier hector_mapping.launch
```

```
~$ rosrun rviz rviz
```



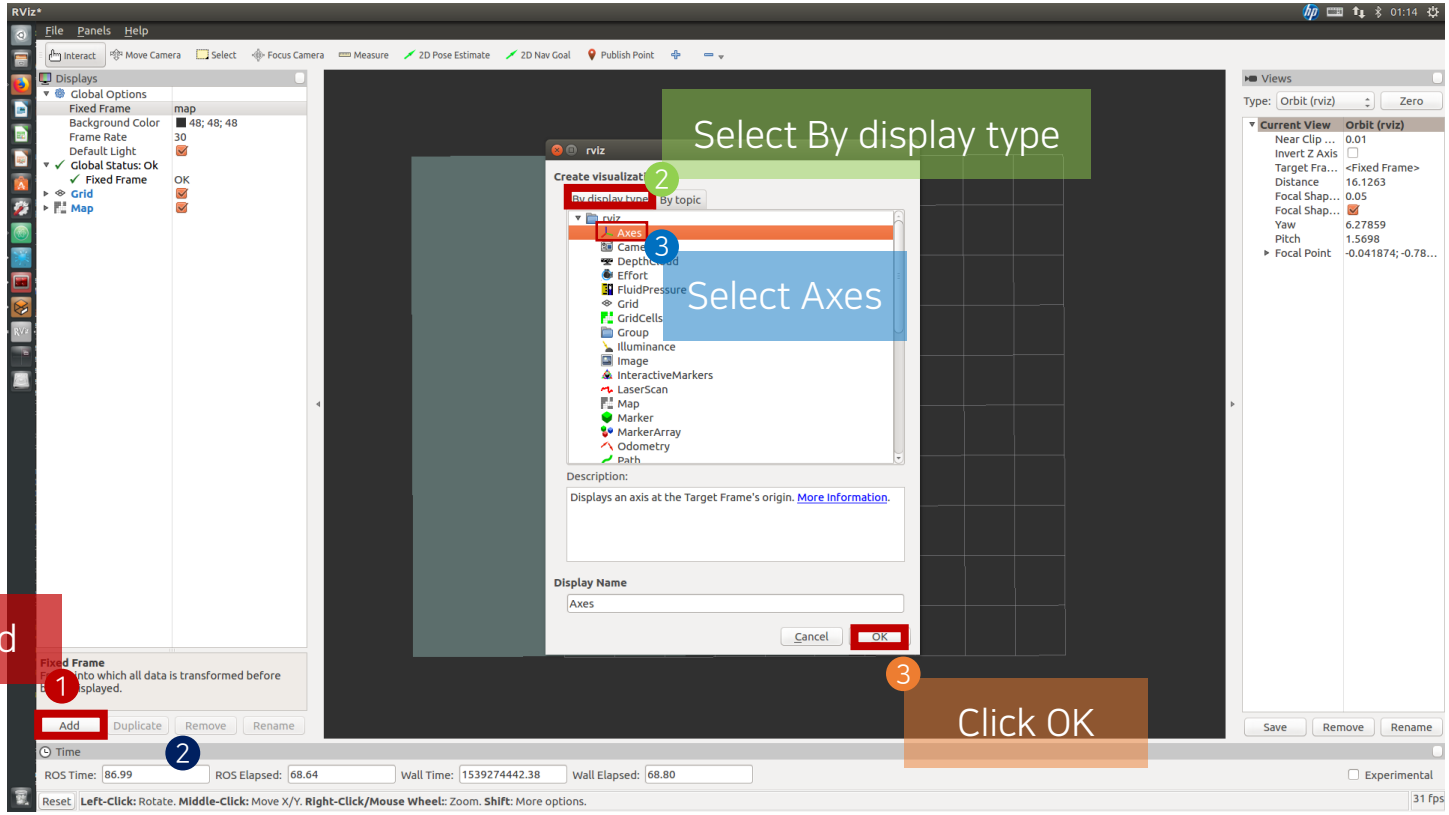
Now let's run it!

Visualization results in rviz : Show the map



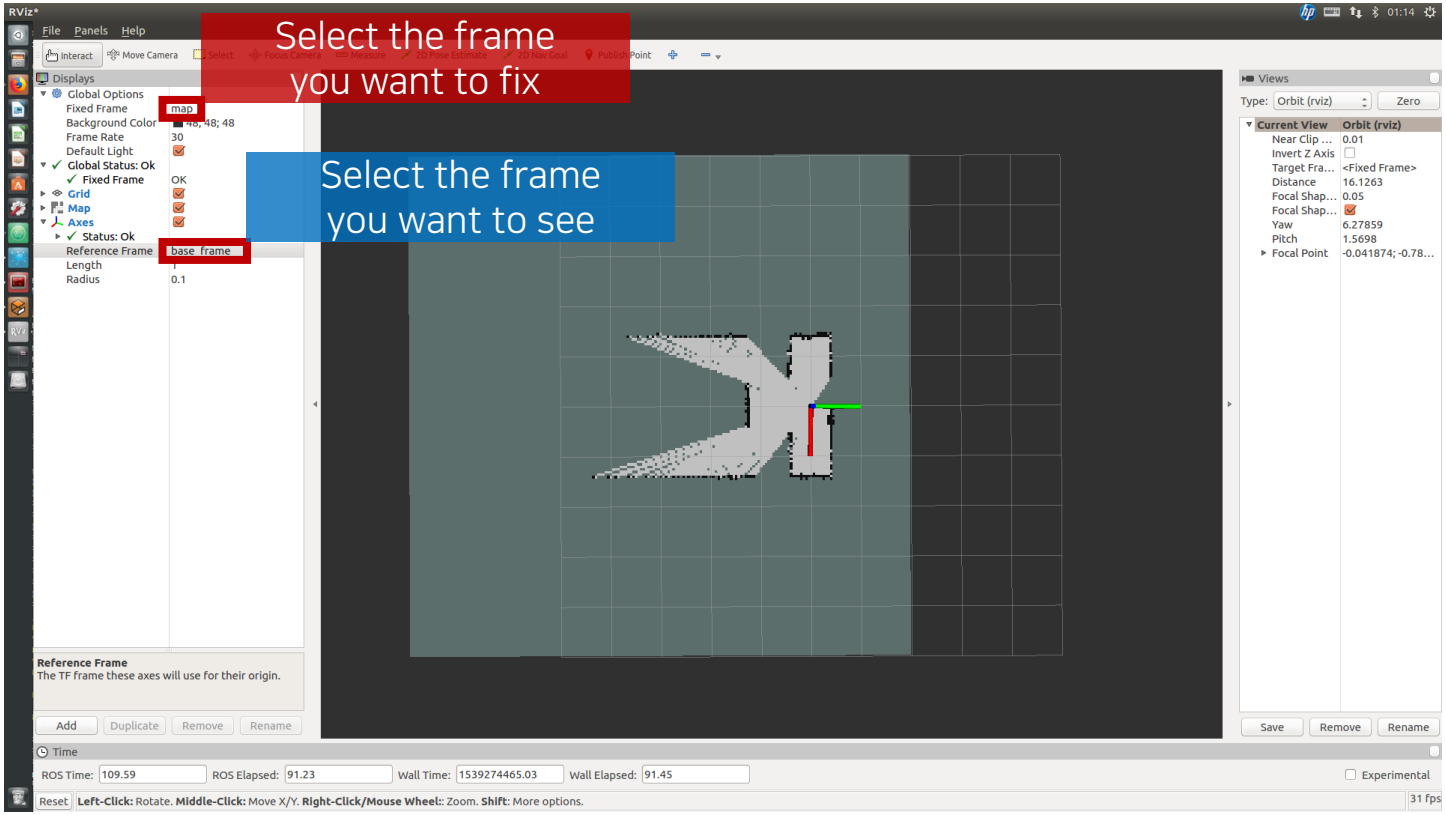
Now let's run it!

Visualization results in rviz : Show the robot position on the map



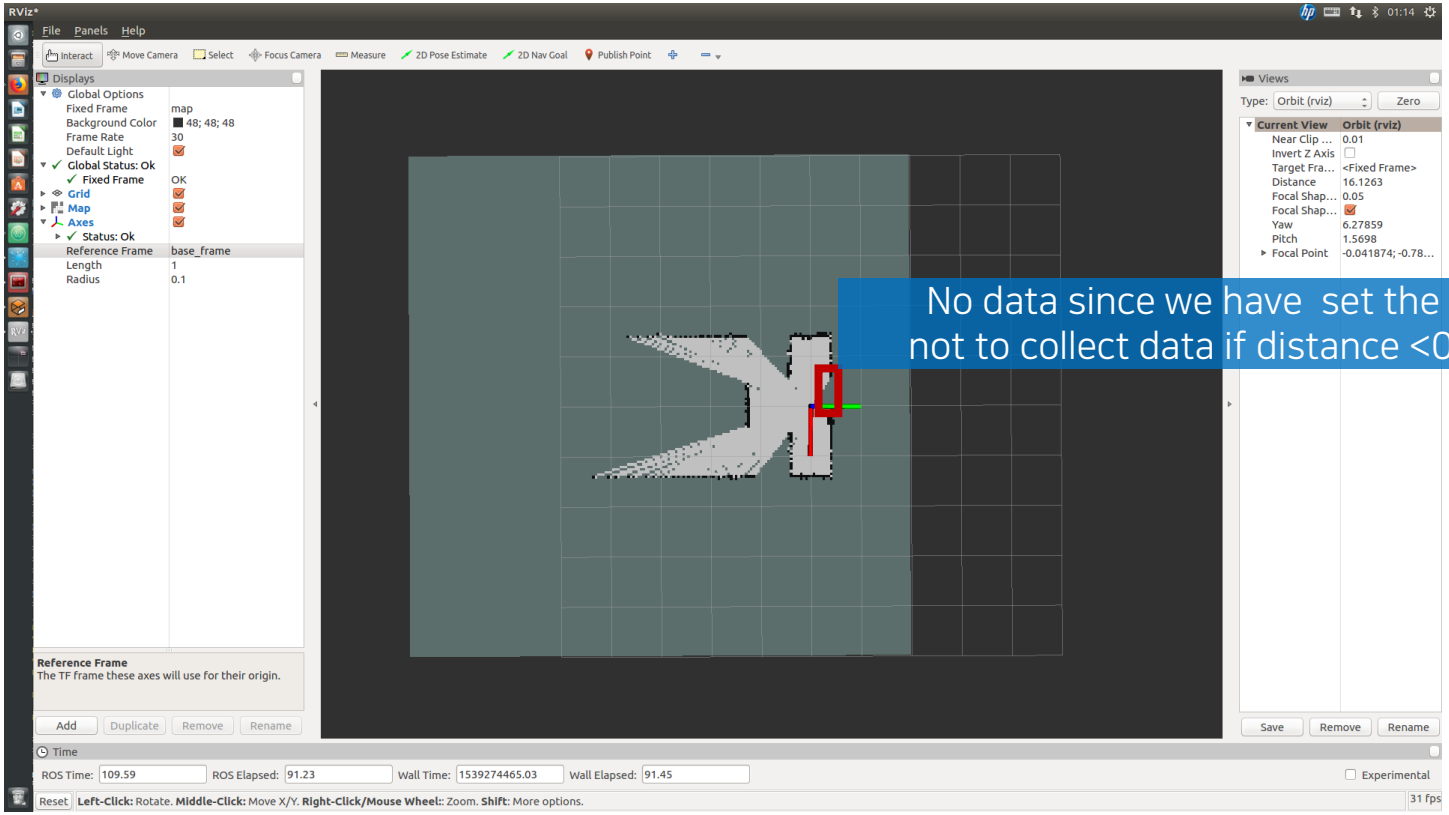
Now let's run it!

Visualization results in rviz



Now let's run it!

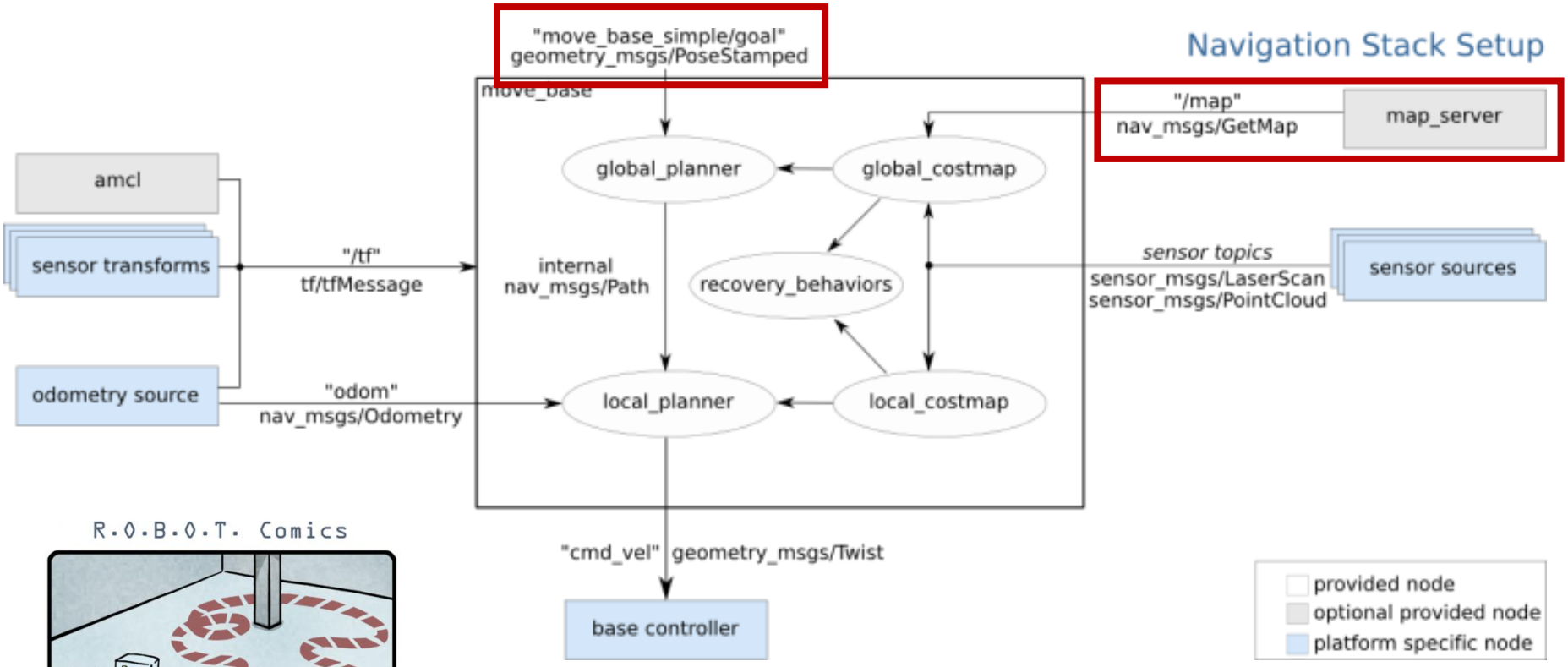
Visualization results in rviz

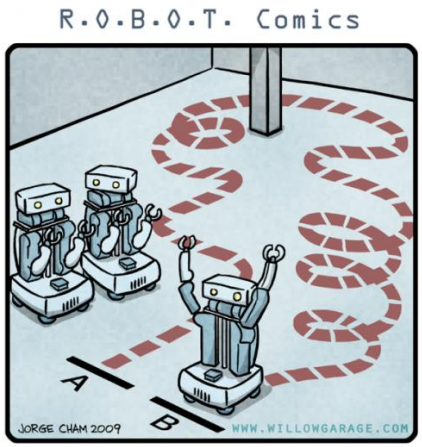


No data since we have set the lidar not to collect data if distance <0.15m

About move_base package

Visualization results in rviz





"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Parameters in move_base

For our robots

```
~$ cd ~/me491_ros/src/find_frontier/launch
~$ gedit move_base.launch
```

```
<?xml version="1.0"?>

<launch>
  <arg name="base_global_planner" default="navfn/NavfnROS"/>
  <arg name="base_local_planner" default="dwa_local_planner/DWAPlannerROS"/>
  Planner for global and local plan
…
    <param name="base_global_planner" value="$(arg base_global_planner)"/>
    <param name="base_local_planner" value="$(arg base_local_planner)"/>
    <rosparam file="$(find find_frontier)/config/planner.yaml" command="load"/>
    Get parameters from planner.yaml
            - Vehicle behavior : Maximum acceleration, velocity, goal tolerance, etc.,
…
</launch>
```

Parameters in move_base

For our robots

```
<?xml version="1.0"?>

…
    <!-- observation sources located in costmap_common.yaml -->
    <rosparam file="$(find find_frontier)/config/costmap_common.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find find_frontier)/config/costmap_common.yaml" command="load" ns="local_costmap" />
    Get parameters from costmap_common.yaml
            - Vehicle size (footprint), base frame name, etc.
    <!-- local costmap, needs size -->
    <rosparam file="$(find find_frontier)/config/costmap_local.yaml" command="load" ns="local_costmap" />
    Get parameters from costmap_local.yaml
            - Parameters for local costmap
    <!-- static global costmap, static map provides size -->
    <rosparam file="$(find find_frontier)/config/costmap_global_static.yaml" command="load" ns="global_costmap" />
    Get parameters from costmap_global.yaml
            - Parameters for global costmap
  </node>
</launch>
```

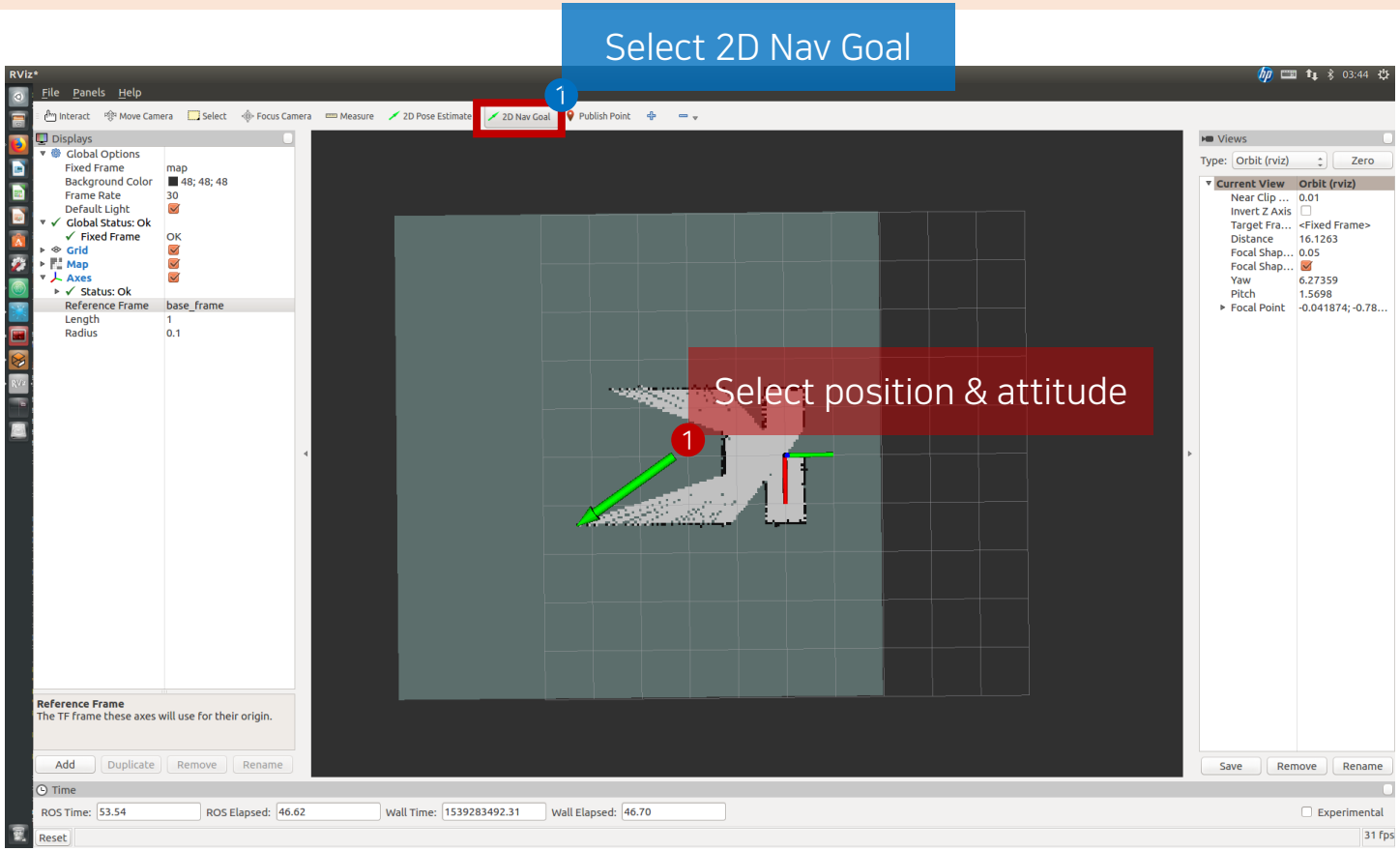
Use rviz to run path planning & tracking control

Visualization results in rviz

```
~$ roslaunch m_robot_gazebo m_robot_world.launch
```

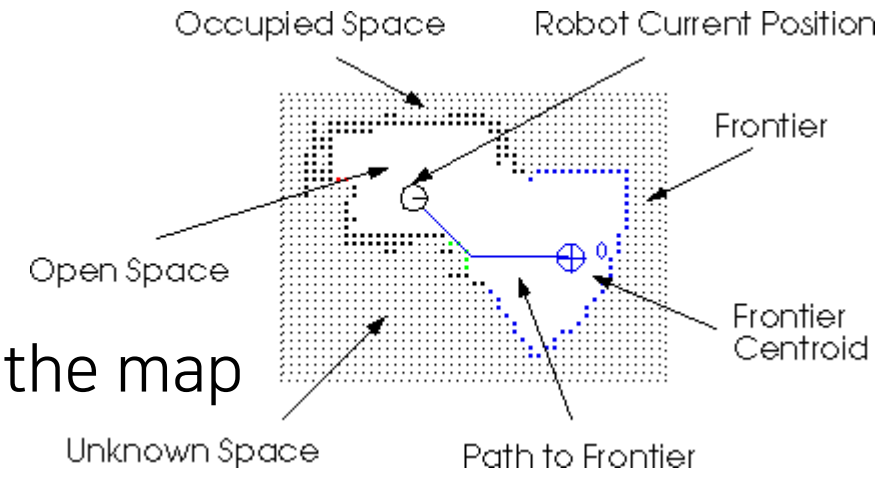
```
~$ roslaunch find_frontier hector_mapping.launch
```

```
~$ rosrun rviz rviz
```



03

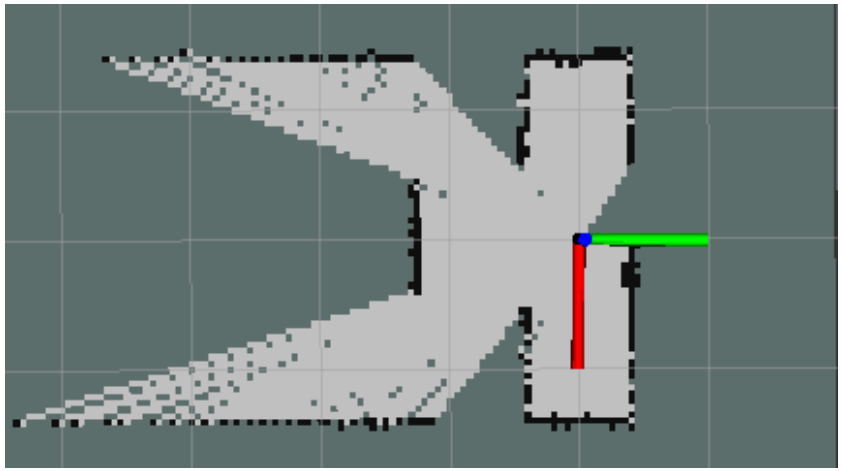
Finding frontiers of the map

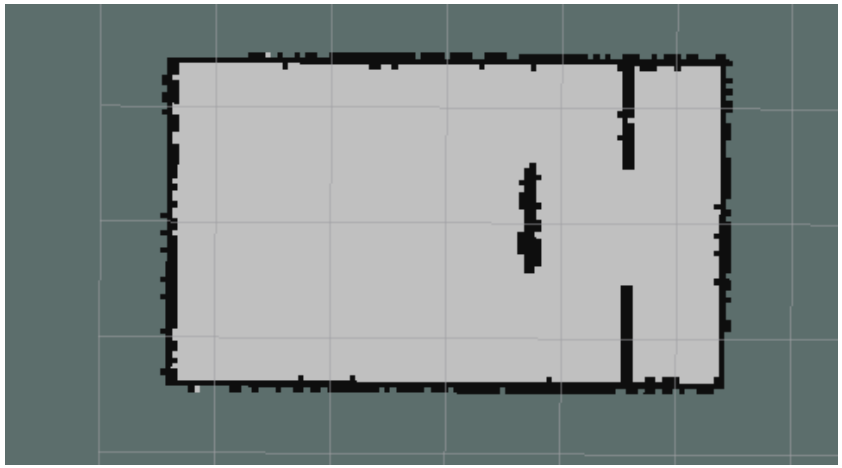


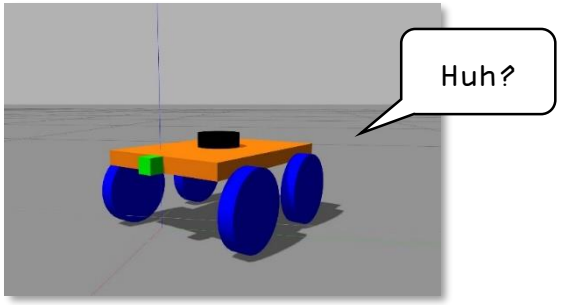
T.A. Dongha Chung chungdongha@kaist.ac.kr

Objective

Build the map as closed form autonomously





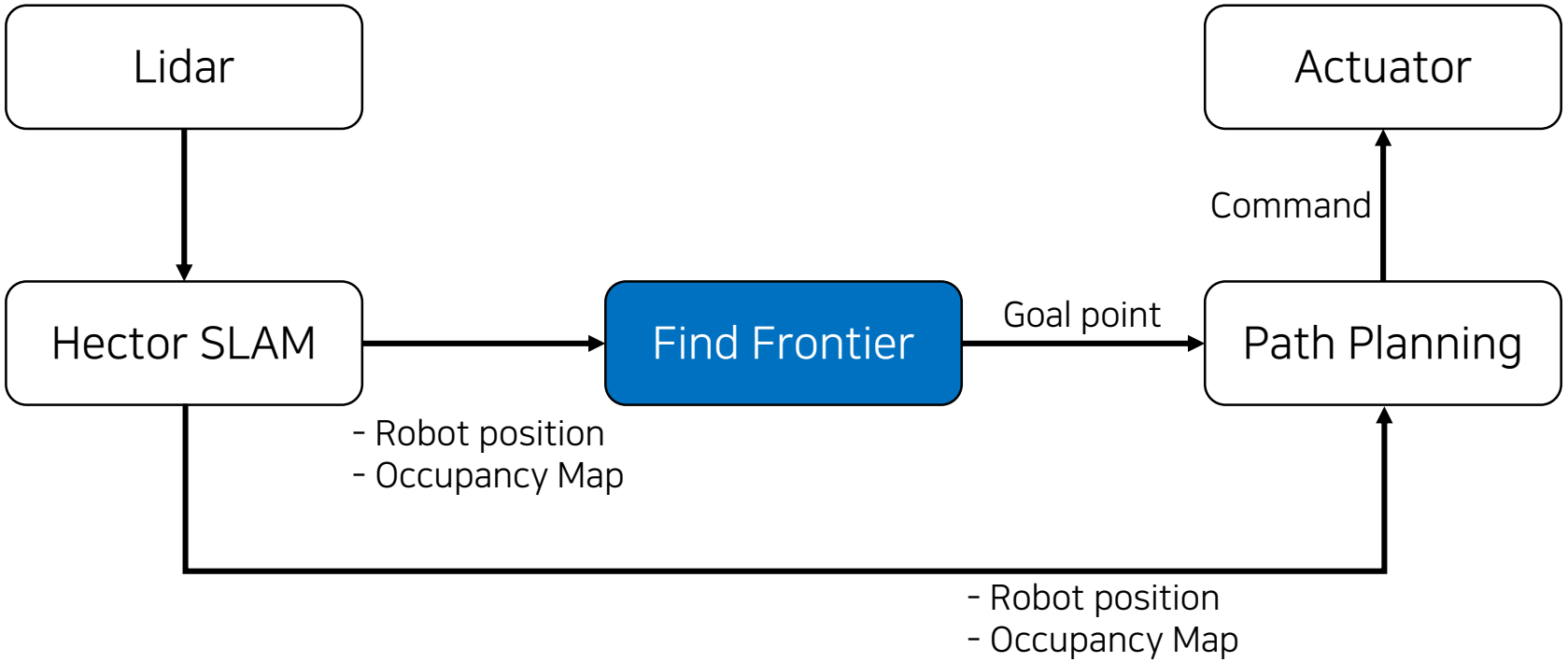


Set the goal point to explore all the space in a closed space

Brian Yamauchi, A Frontier-Based Approach for Autonomous Exploration
In Proceedings of the IEEE International Symposium on Computational Intelligence, Robotics and Automation

Exploring frontier

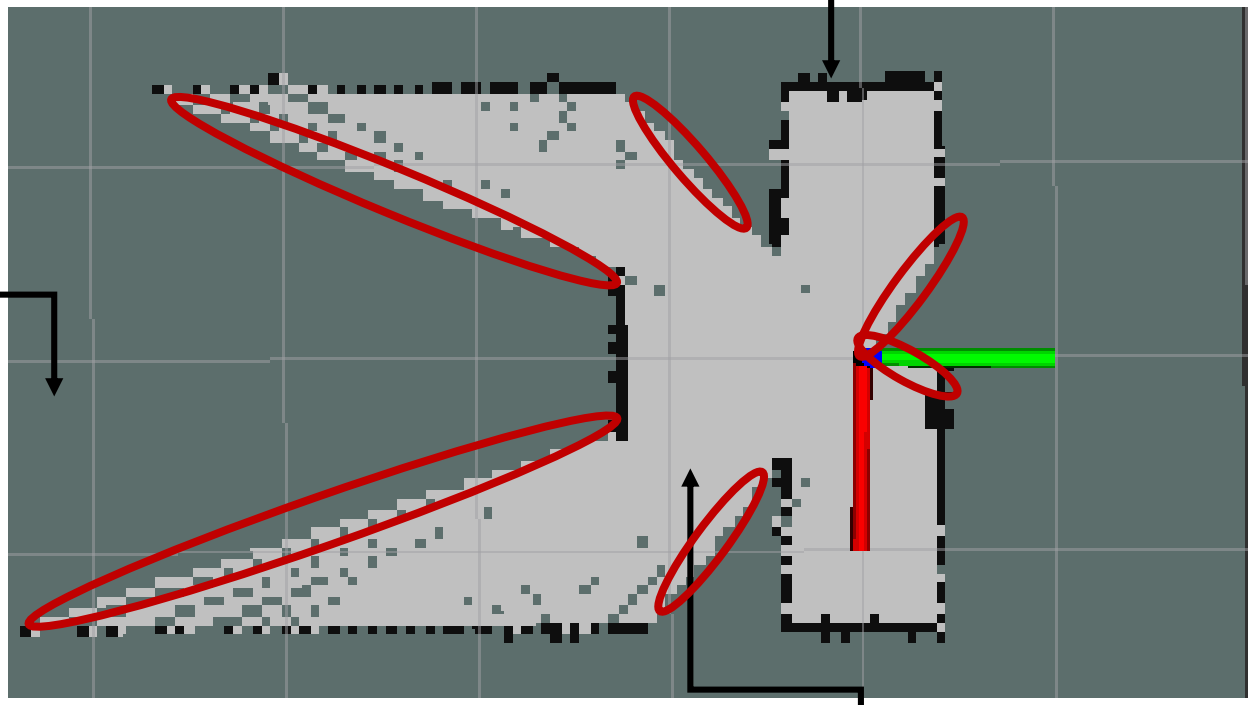
What we need to do



Brian Yamauchi, A Frontier-Based Approach for Autonomous Exploration
In Proceedings of the IEEE International Symposium on Computational Intelligence, Robotics and Automation

Exploring frontier

What is frontier?

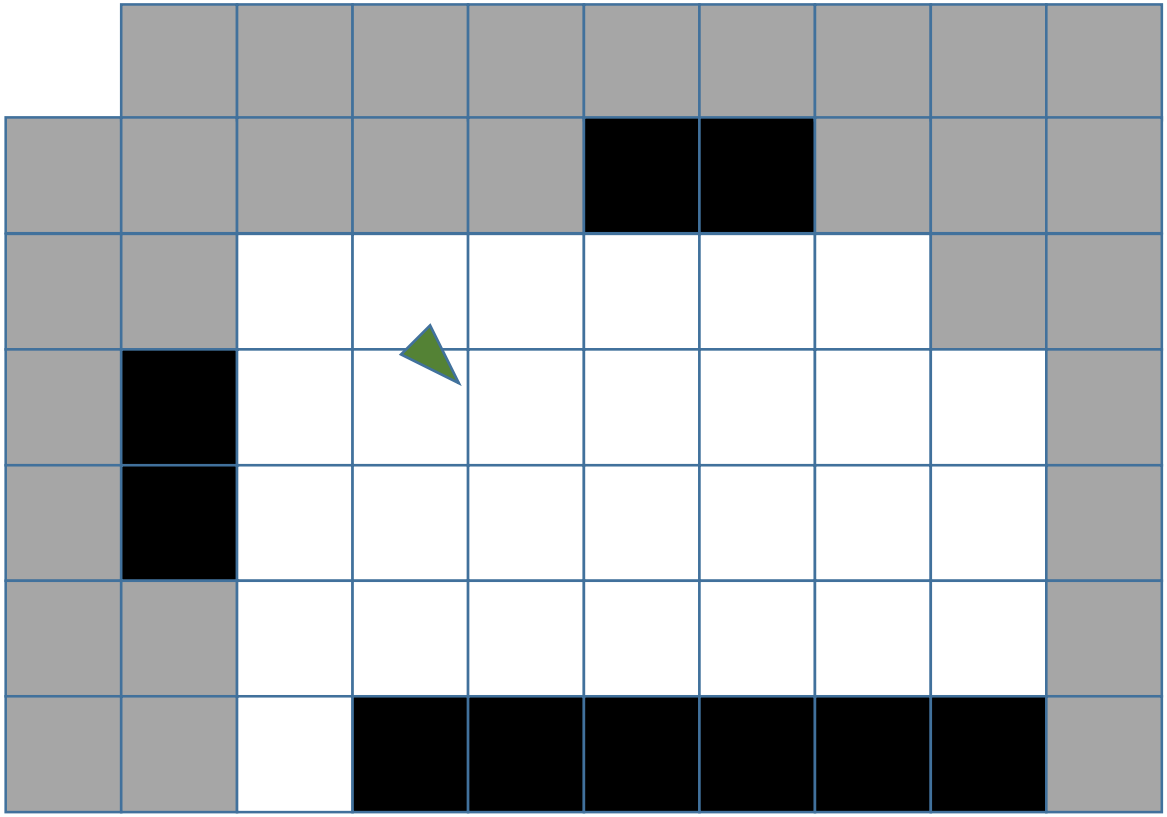


Edges where the free space and unknown space meet

Brian Yamauchi, A Frontier-Based Approach for Autonomous Exploration
In Proceedings of the IEEE International Symposium on Computational Intelligence, Robotics and Automation

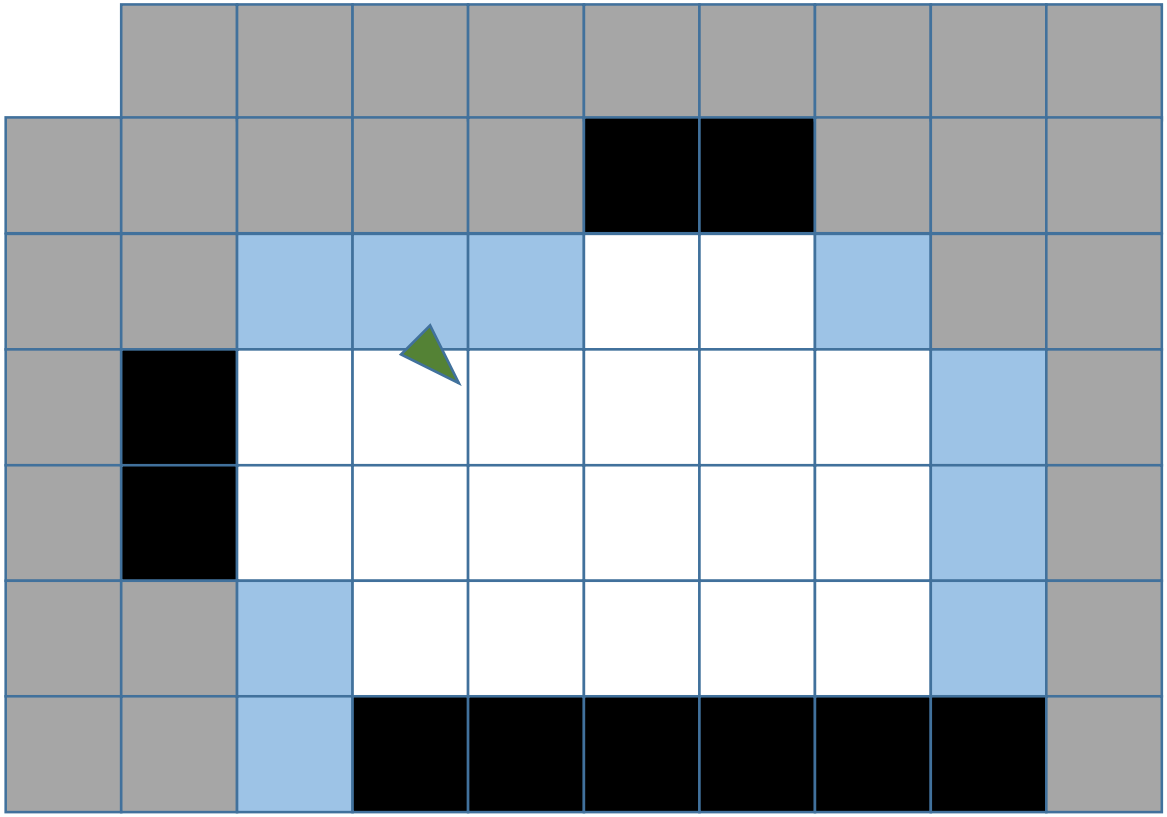
Exploring frontier

Algorithm overview



Exploring frontier

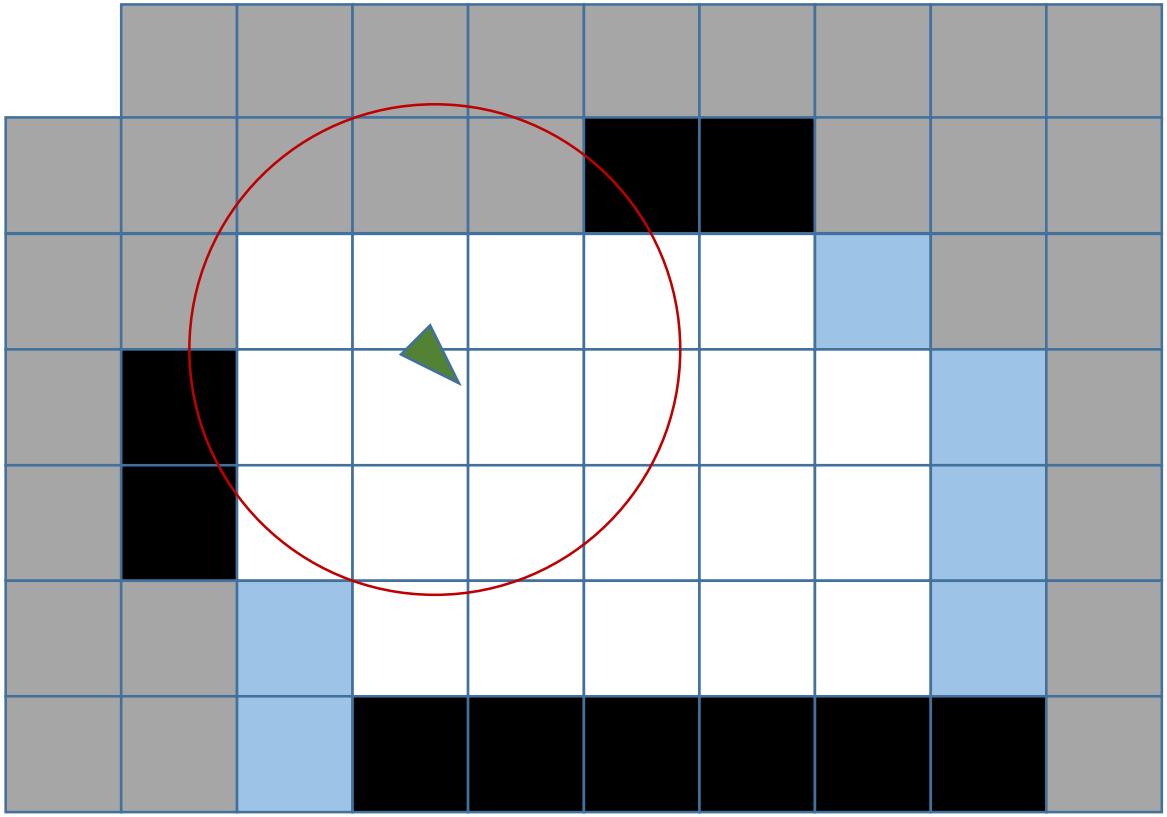
Algorithm overview : Candidate cells



If there exist an unexplored cell, the cell is a candidate for frontier

Exploring frontier

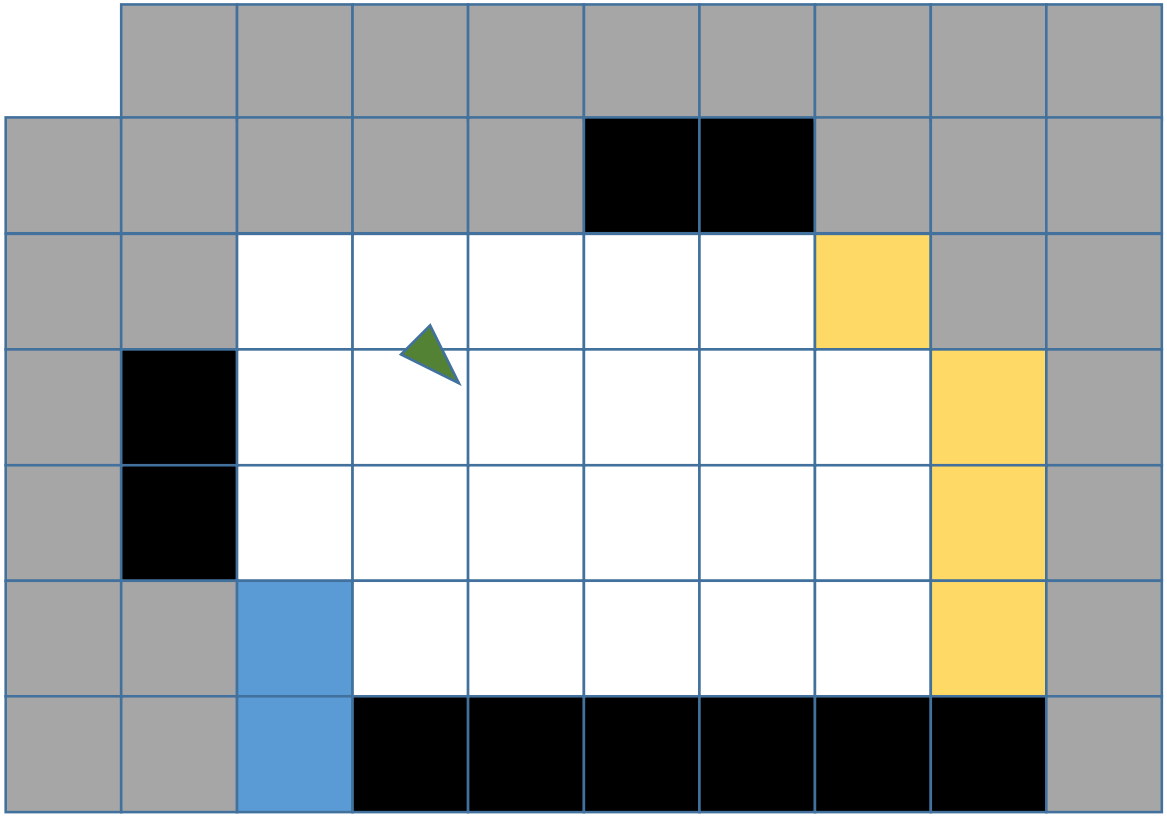
Algorithm overview : Candidate cells



The cells near the robot (distance < α) are not candidate cells

Exploring frontier

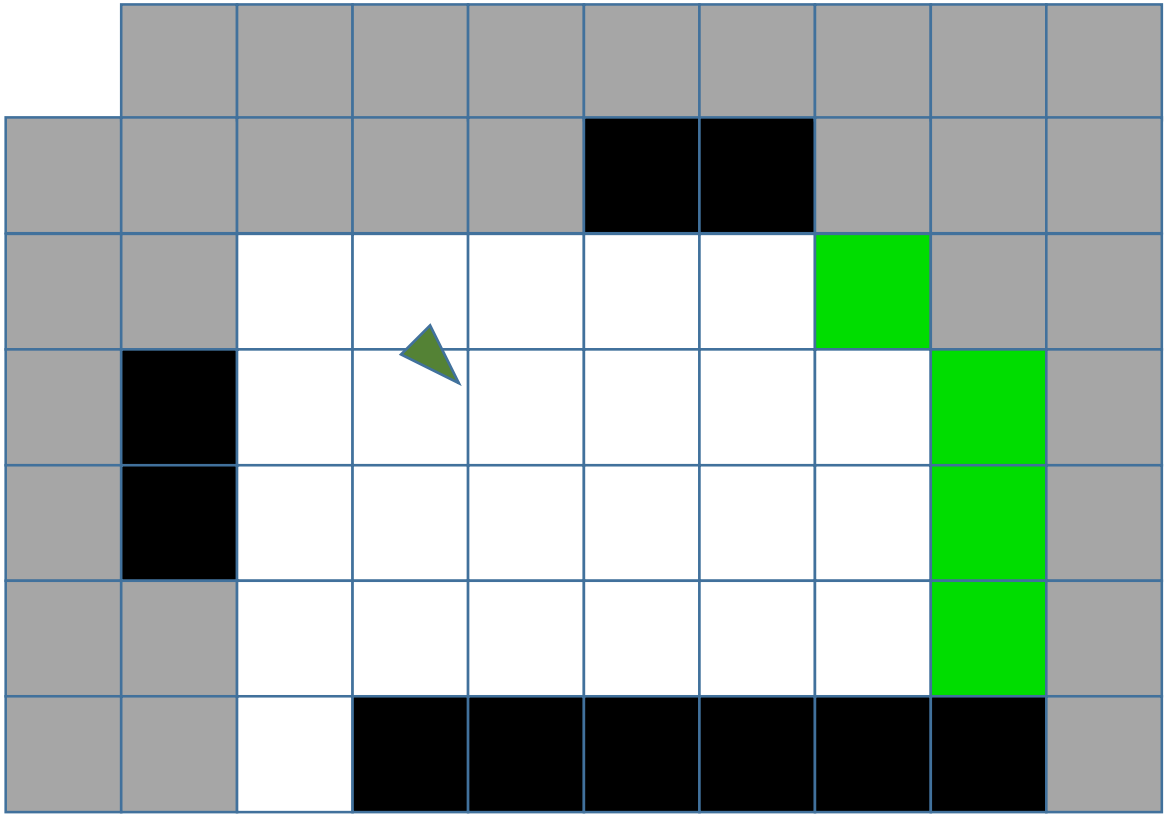
Algorithm overview : Candidate cells



Cluster the candidate cells that are connected

Exploring frontier

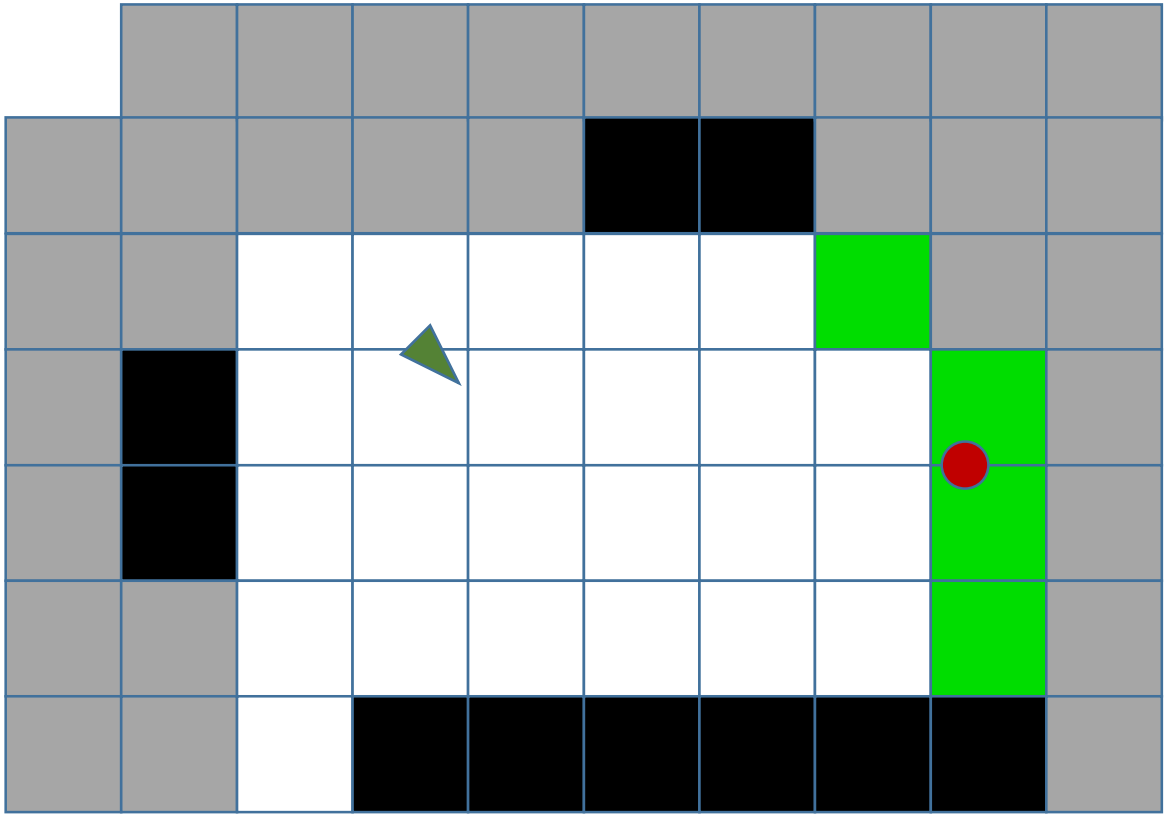
Algorithm overview : Candidate cells



Select the cluster that has most cells

Exploring frontier

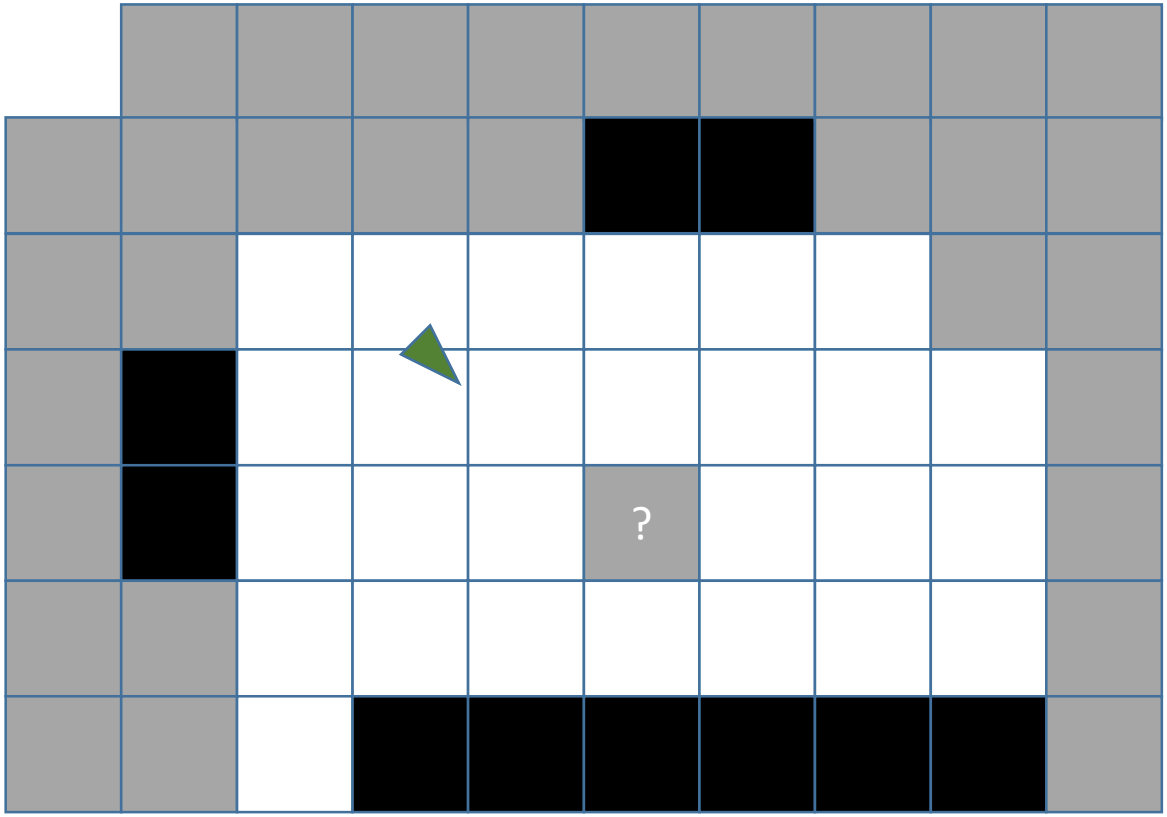
Algorithm overview : Candidate cells



Set the center of cluster to be the goal

Exploring frontier

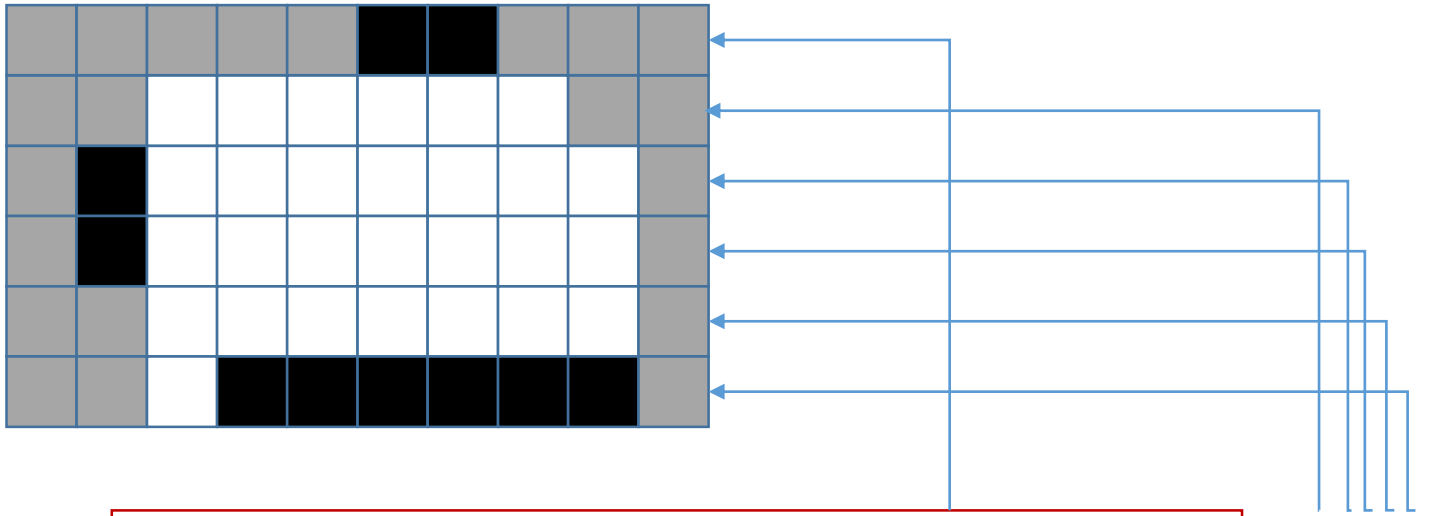
Algorithm overview : Candidate cells



Set the center of cluster to be the goal

Exploring frontier

Code review

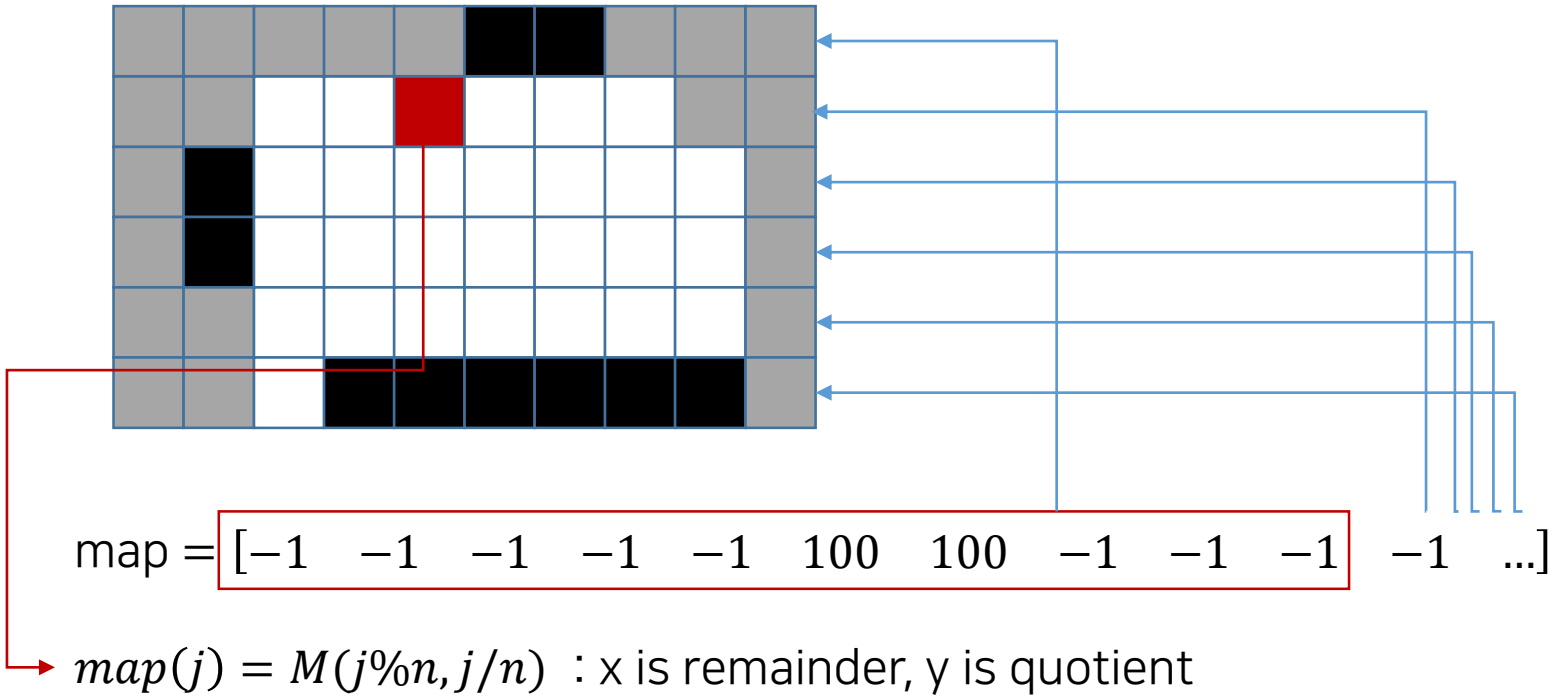


map = [−1 −1 −1 −1 −1 100 100 −1 −1 −1 −1 ...]

The map data is stored in $m \times n$ by 1 dimensional array

Exploring frontier

Code review



Exploring frontier

Code review

```
~$ cd ~/me491_ros/src/find_frontier/src
~$ gedit find_fronter.cpp
```

```
void mapConvert(const nav_msgs::OccupancyGrid::ConstPtr& msg)
Function activated when occupancy grid map message is received

void PoseUpdate(const geometry_msgs::PoseStampedConstPtr& pose)
Function activated when the robot pose is received

In main function
goal_pub = nh.advertise<geometry_msgs::PoseStamped>("move_base_simple/goal",1);
Publish the goal point as geometry_msgs::PoseStamped
```

Exploring frontier

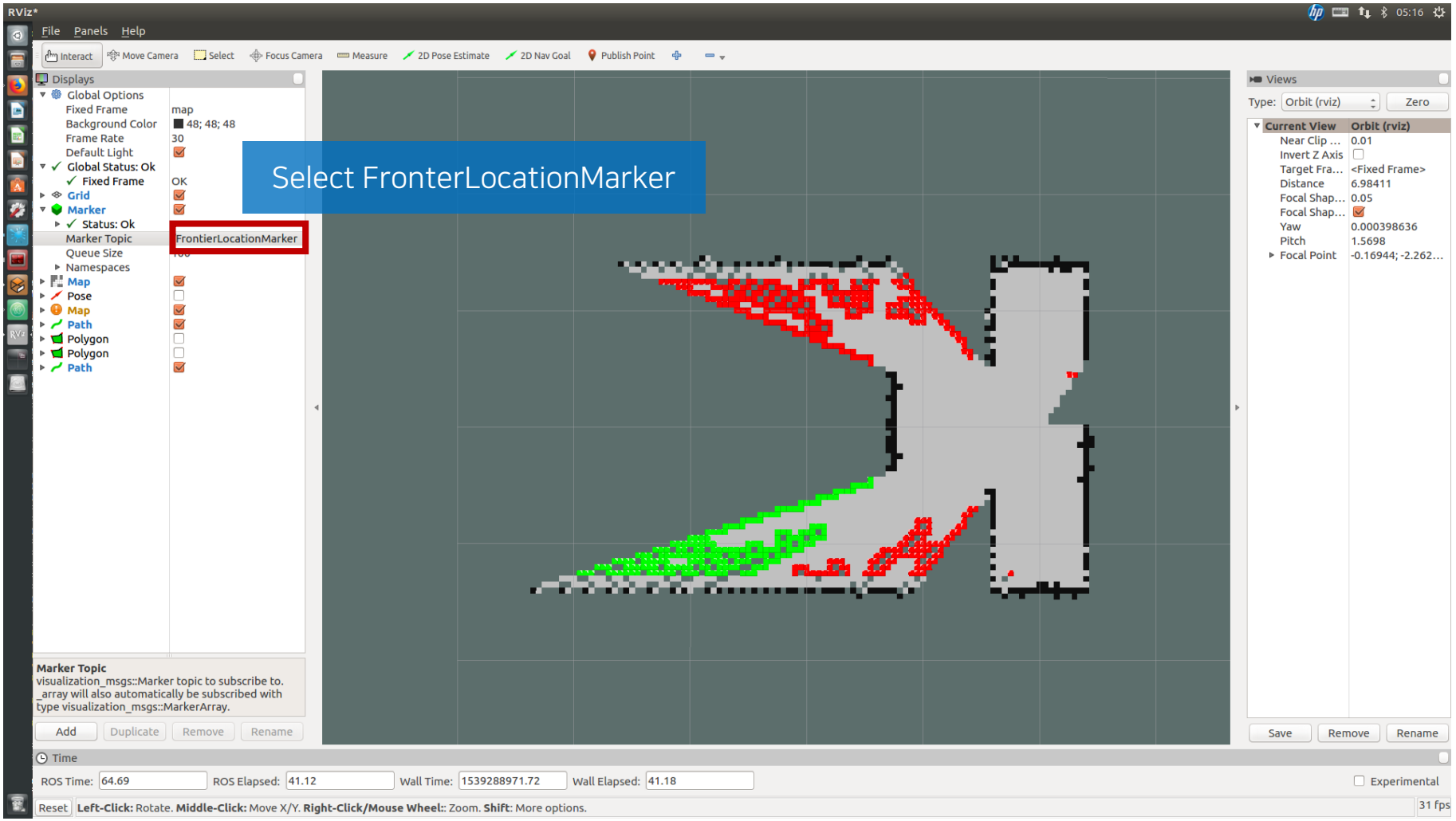
Code review

```
~$ roslaunch m_robot_gazebo m_robot_world.launch
~$ roslaunch find_frontier hector_mapping.launch
~$ rosrun find_frontier find_frontier
~$ rosrun rviz rviz
```

Brian Yamauchi, A Frontier-Based Approach for Autonomous Exploration
In Proceedings of the IEEE International Symposium on Computational Intelligence, Robotics and Automation

Exploring frontier

Run example



Exploring frontier

Code review

```
~$ roslaunch m_robot_gazebo m_robot_world.launch
~$ roslaunch find_frontier nav_run.launch
```

Brian Yamauchi, A Frontier-Based Approach for Autonomous Exploration
In Proceedings of the IEEE International Symposium on Computational Intelligence, Robotics and Automation