Mobile Robotics & Intelligence Lab.

Hyundai NGV
Autonomous Driving for Mobile Robot

Practice Session

Mobile Robot Navigation and SLAM

TA. Eunghyun Kim, Sol Han

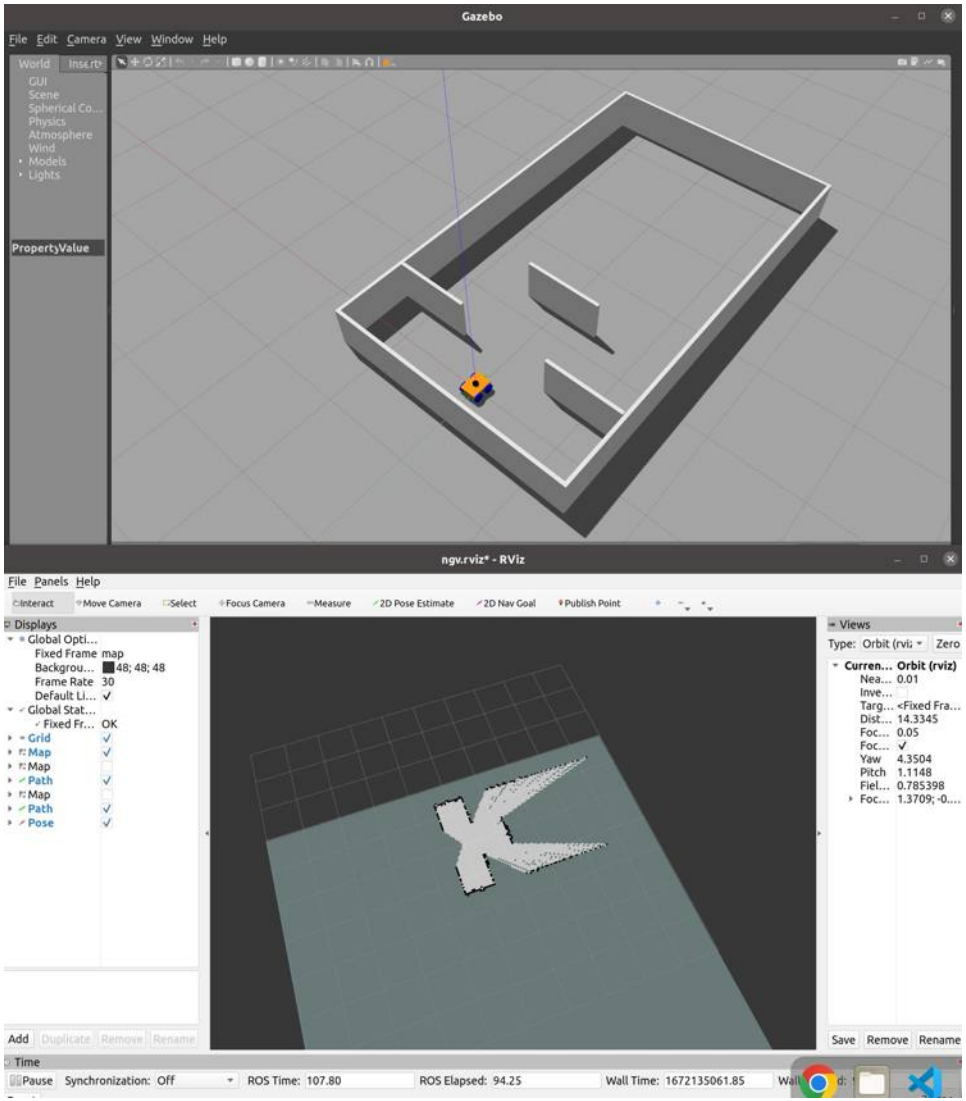
Mobile Robotics & Intelligence Lab.

Department of ME, KAIST

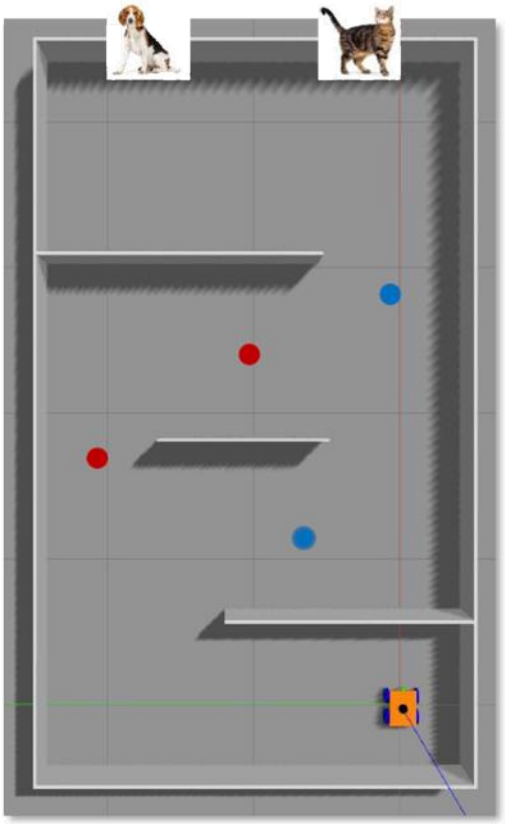
Original Material from Dongha Chung & Junwoo Jang

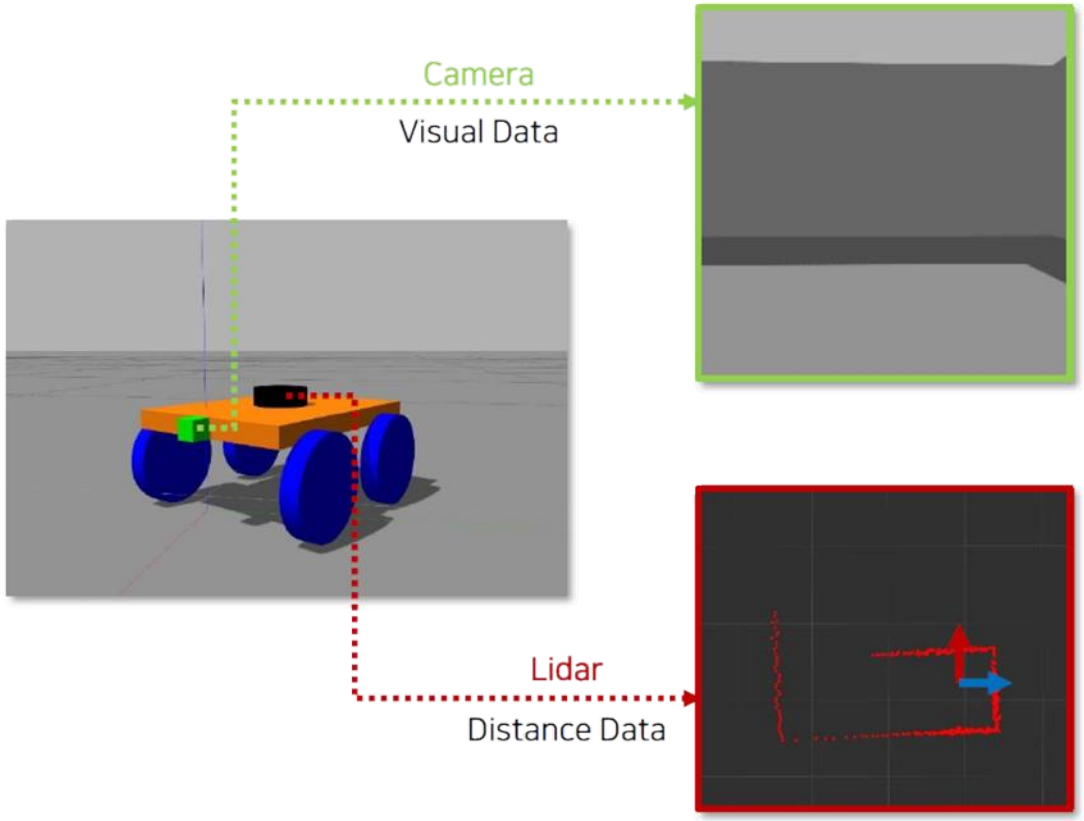
ENGINEERING FOR MANKIND KAIST

Preview


Gazebo
File Edit Camera View Window Help
World
GUI
Scene
Physics
Atmosphere
Wind
Models
Lights
Property Value
ngv.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Opti...
Fixed Frame map
Frame Rate 30
Grid
Map
Path
Pose
Views
Orbit (rviz)
Add
Duplicate
Remove
Rename
Save
Remove
Rename
Time
Pause
Synchronization: Off
ROS Time: 107.80
ROS Elapsed: 94.25
Wall Time: 1672135061.85

What do we have to do...?



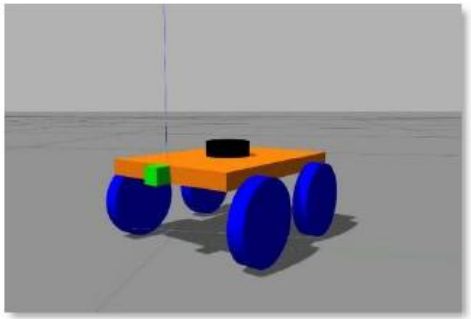


CONTENTS

01. Construction of simulation environment using Gazebo

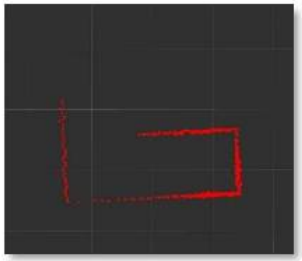
02. Simulating SLAM using Hector SLAM package

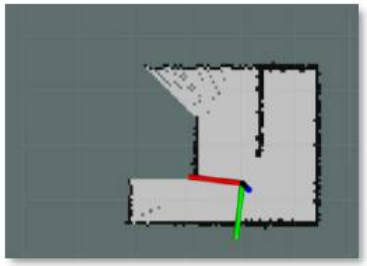
03. Finding frontiers of the map

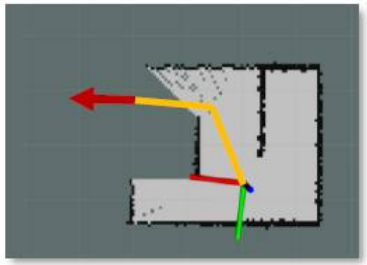


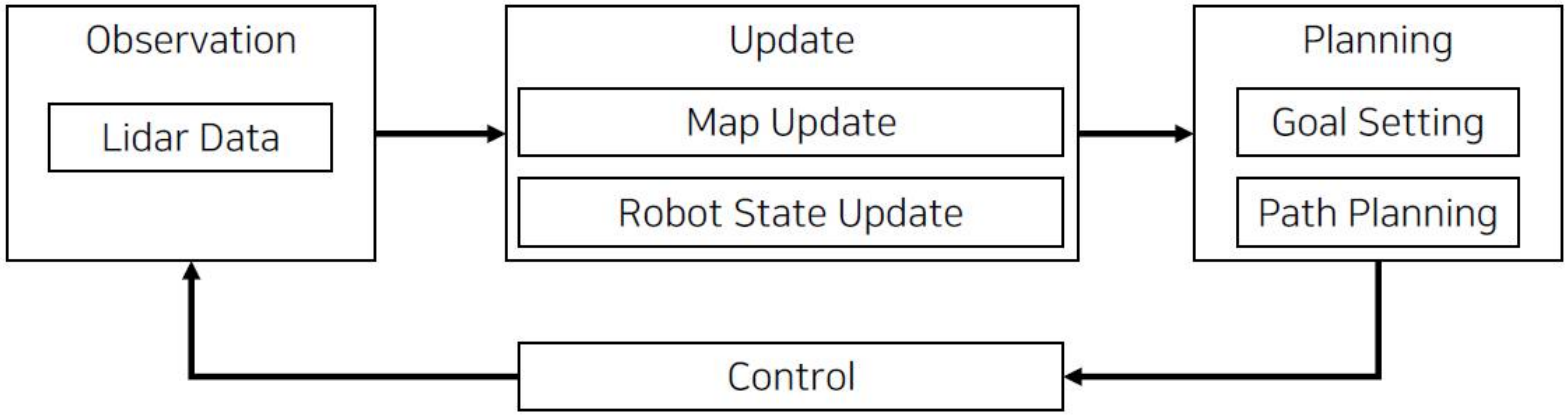
Before making the robot move, we should know :

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









Phase 1 : Download and Run the code

With minimum explanation

01

Construction of simulation environment using Gazebo

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.

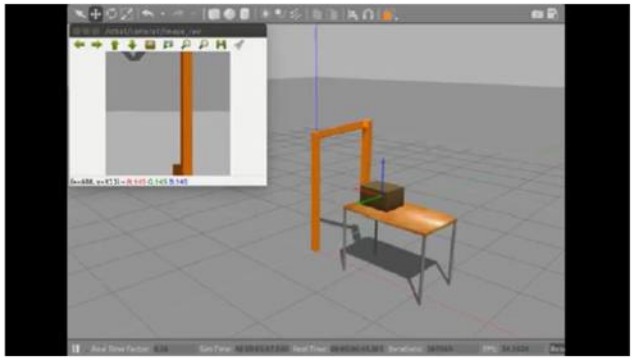
Plugins
Develop custom plugins for robot, sensor, and environmental control. Plugins provide direct access to Gazebo's API.

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

Download Code and Execute

```
$ git clone https://github.com/Sol-Han/Hyundai_NGV_Mobile_Robot.git
```

cp : copy files
cp file1 file2 … /dir : copy files to dir
cp –r : copy directory
cp –r /dir1 /dir2 : copy directory1 to directory

Let's copy GazeboMap directory to ~/.Gazebo/models/

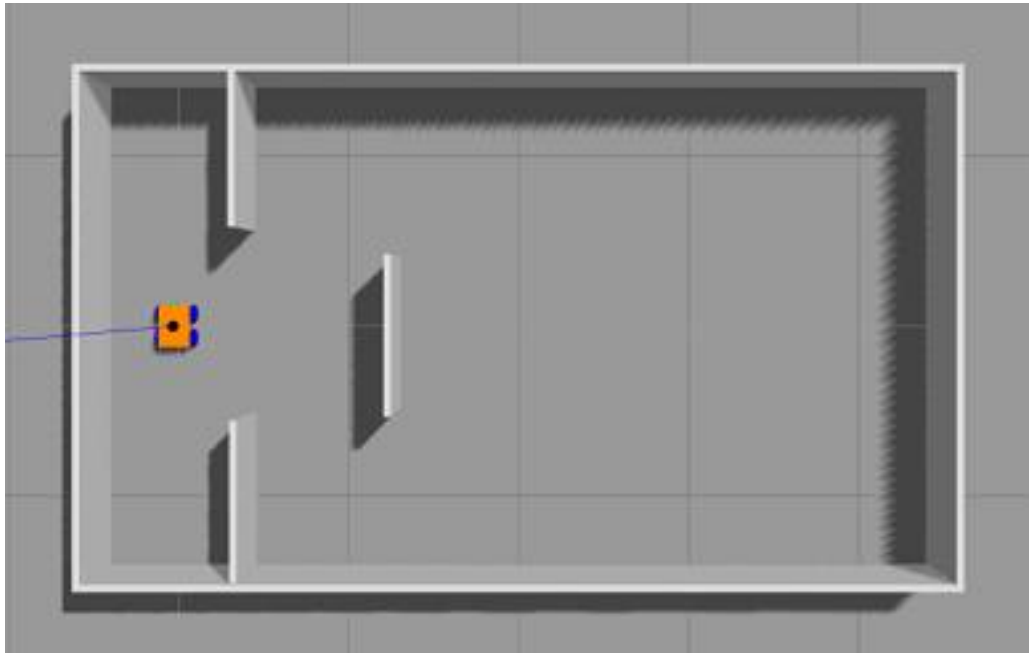
```
$ cd ~/Hyundai_NGV/SLAM/src
$ cp –r ./GazeboMap/simple_map ~/.gazebo/models/
      (you are now at ~~/Hyundai_NGV/SLAM)

$ cd ~/.gazebo/models
$ ls
```

Hidden directory

Download Code and Execute

Copy all source code files to catkin workspace
$ cd (catkin workspace)
$ catkin_make
$ roscore

Ctrl +Alt + t (new terminal)
$ roscd m_robot_gazebo
$ roslaunch m_robot_gazebo m_robot_world.launch

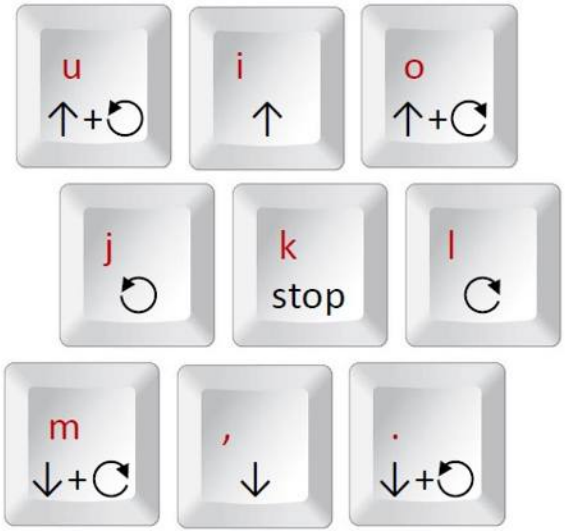


catkin_make error
- source check (bashrc)

How to control the vehicle

$ sudo apt-get install ros-noetic-teleop-twist-keyboard

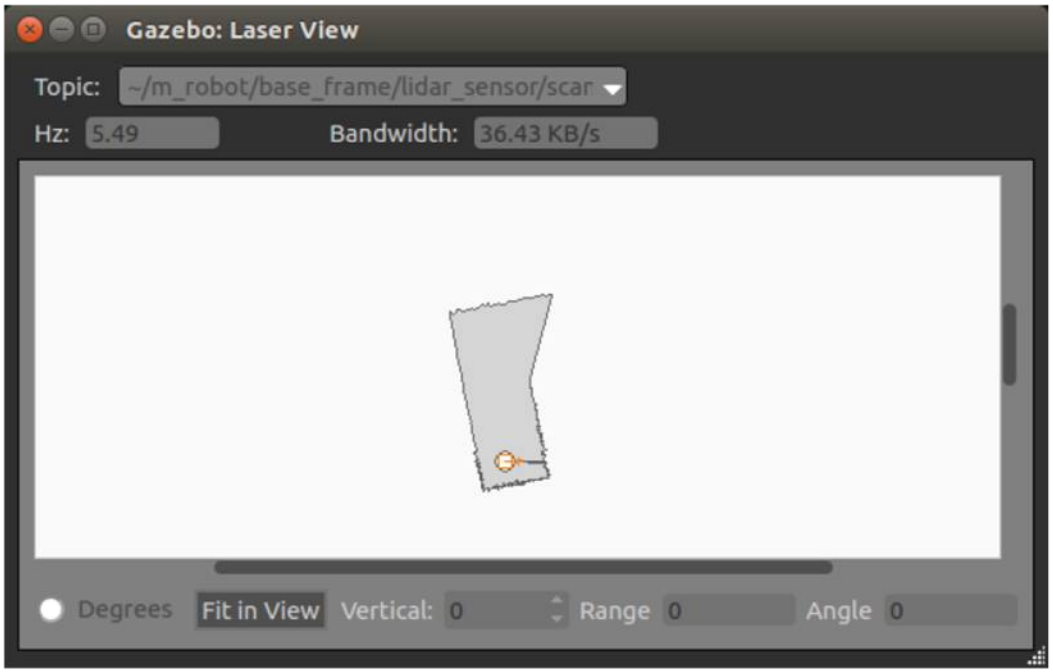
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py



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

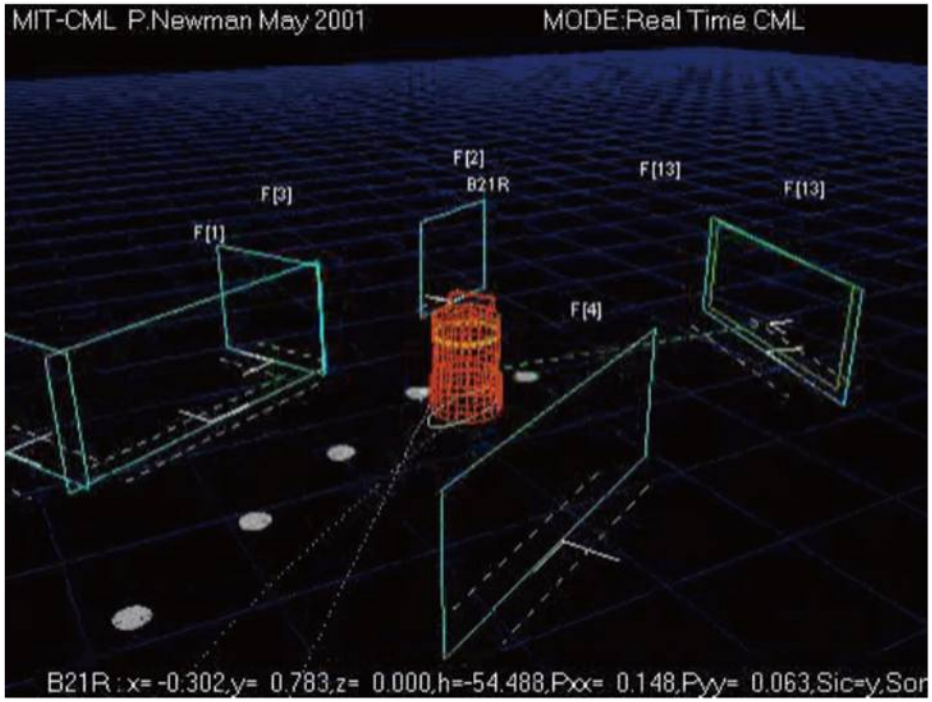
Simulating SLAM using Hector SLAM package

SLAM: Simultaneous Localization and Mapping

A robot moving in an unknown environment

Use sensor data to:

- **build a map** of the environment
- and **at the same time**
- use the map to compute the **robot location**



Localization can be done with map
Mapping can be don localization

What do we have to do first? -> Simultaneously

Team Hector

TECHNISCHE UNIVERSITÄT DARMSTADT

1st place in the European Micro Air Vehicle Conference (EMAV) in category "Outdoor Autonomy", Sep. 2009, Delft

3rd place (out of 12 & 16 teams) at the SICK Company's Robot Day, October 2009 & 2010, Waldkirch

2nd place (out of 27) "Best in Class Autonomy" at RoboCup 2010, Singapore

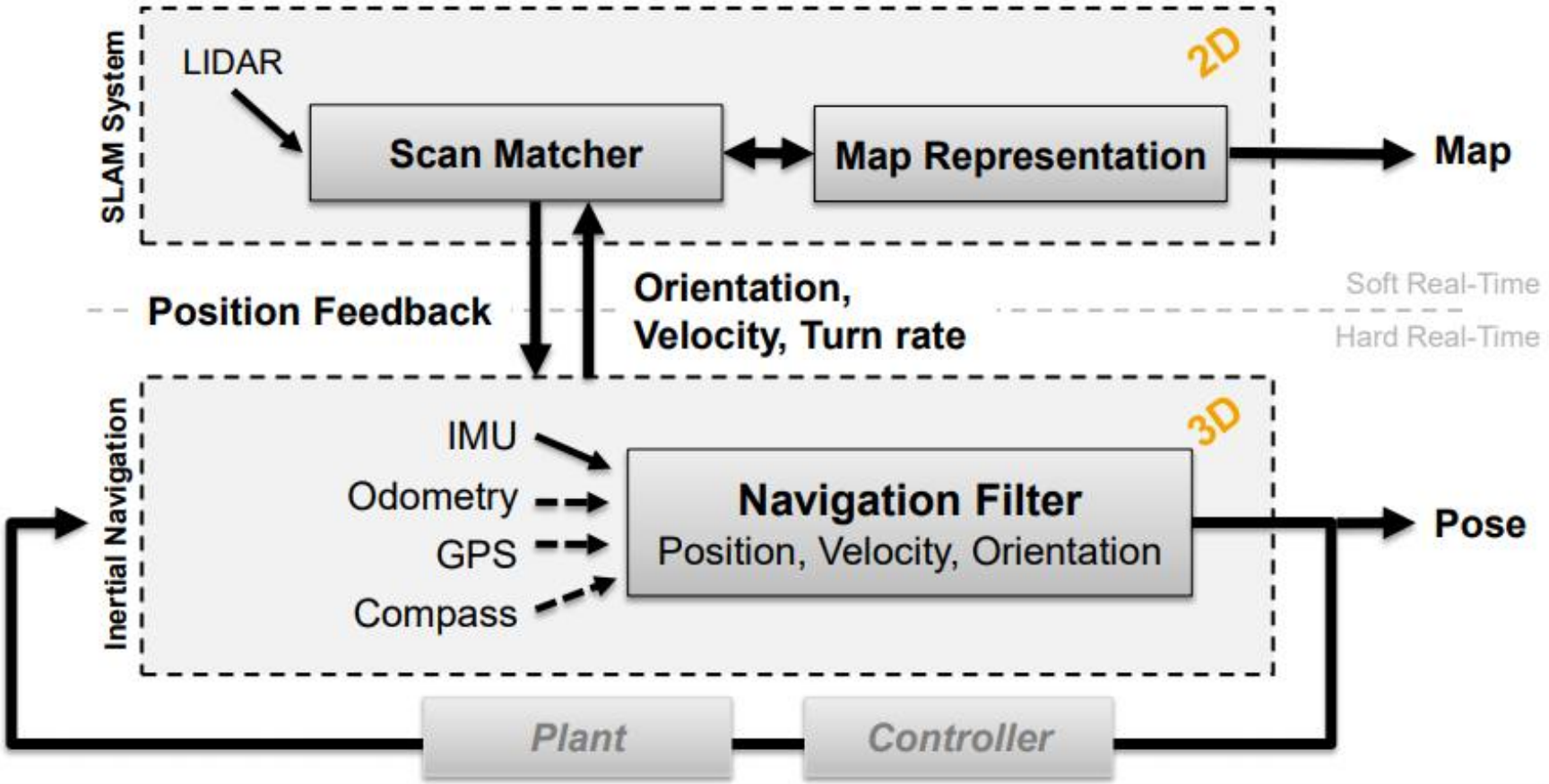
Winner and "Best in Class Autonomy" at Robocup 2011 GermanOpen

2nd place "Best in Class Autonomy" at RoboCup 2011, Istanbul

Appendix: Localization and Mapping
Combine SLAM and EKF

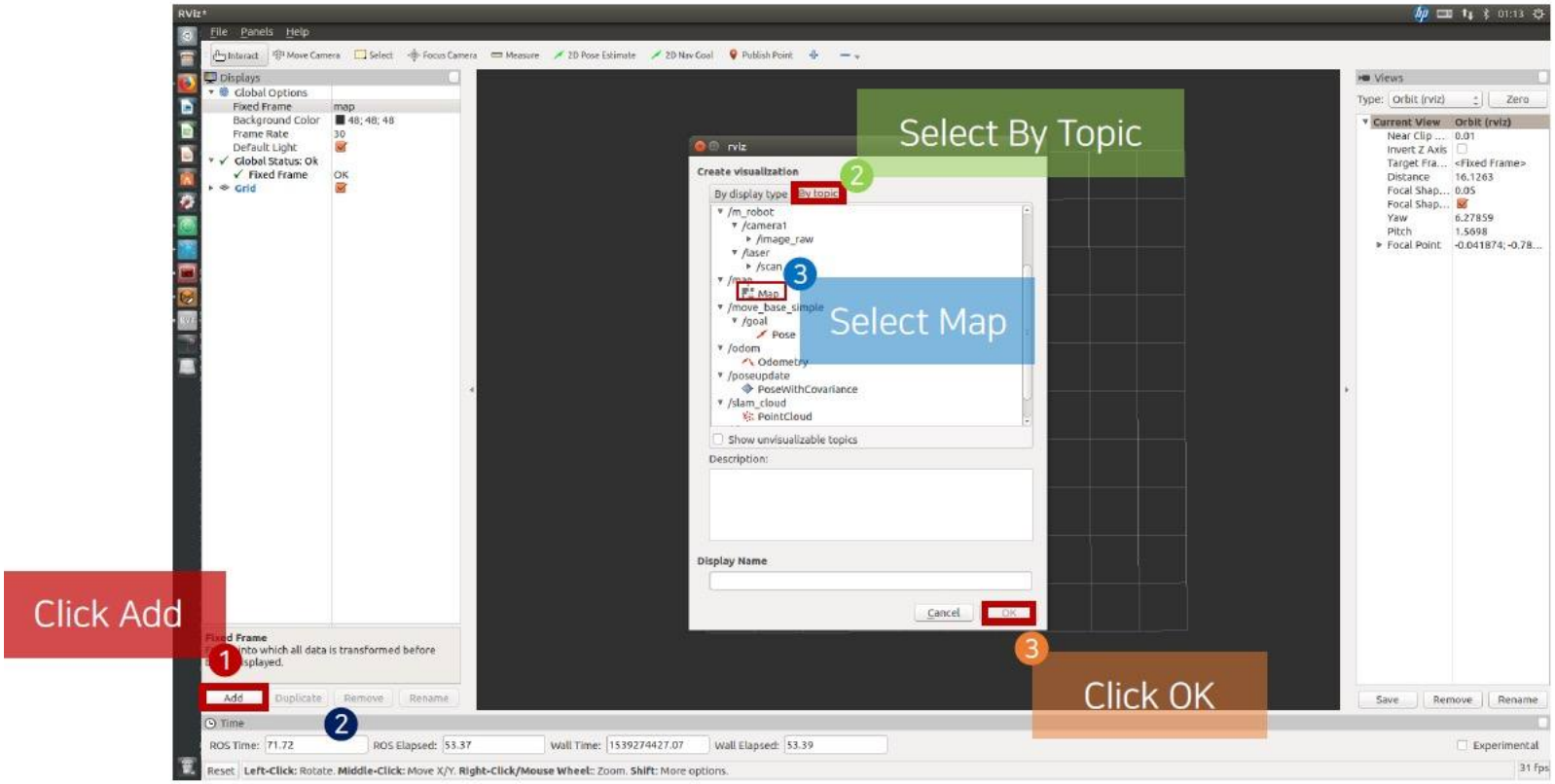
TECHNISCHE UNIVERSITÄT DARMSTADT

- **Our approach:** Couple both localization approaches in a loose manner



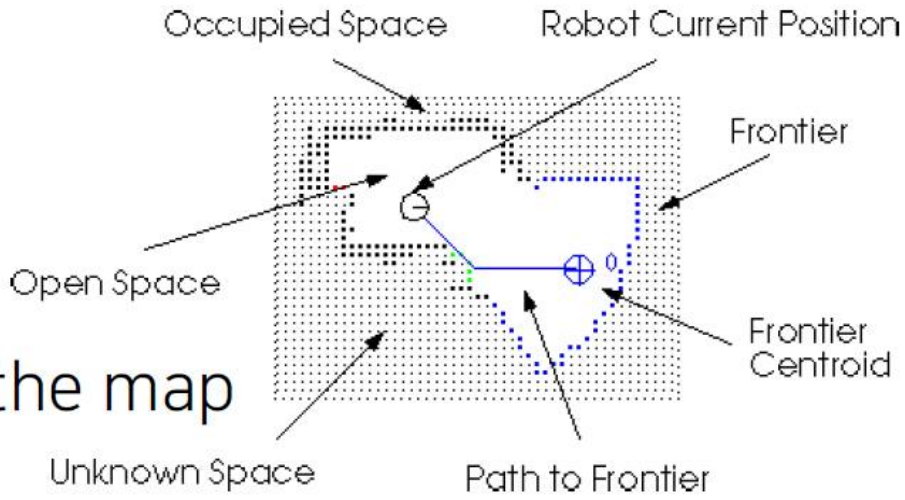
Download code

$ git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam
Copy to catkin workspace
$ roslaunch find_frontier hector_mapping.launch
$ rosrun rviz rviz



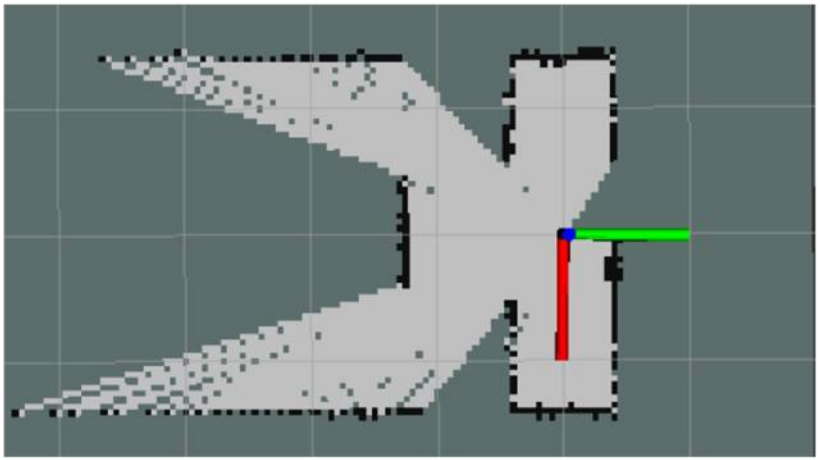
03

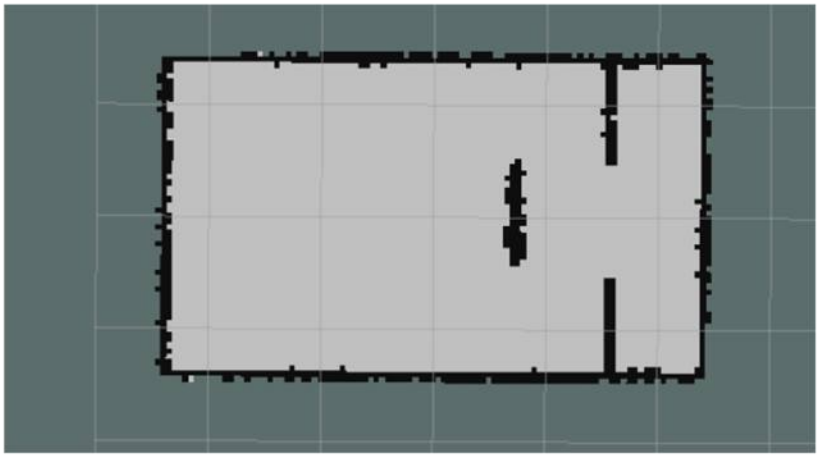
Finding frontiers of the map

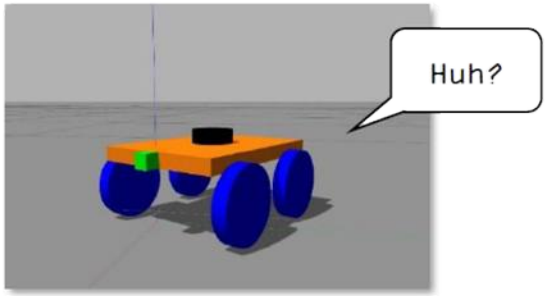


Objective

Build the map as closed form autonomously



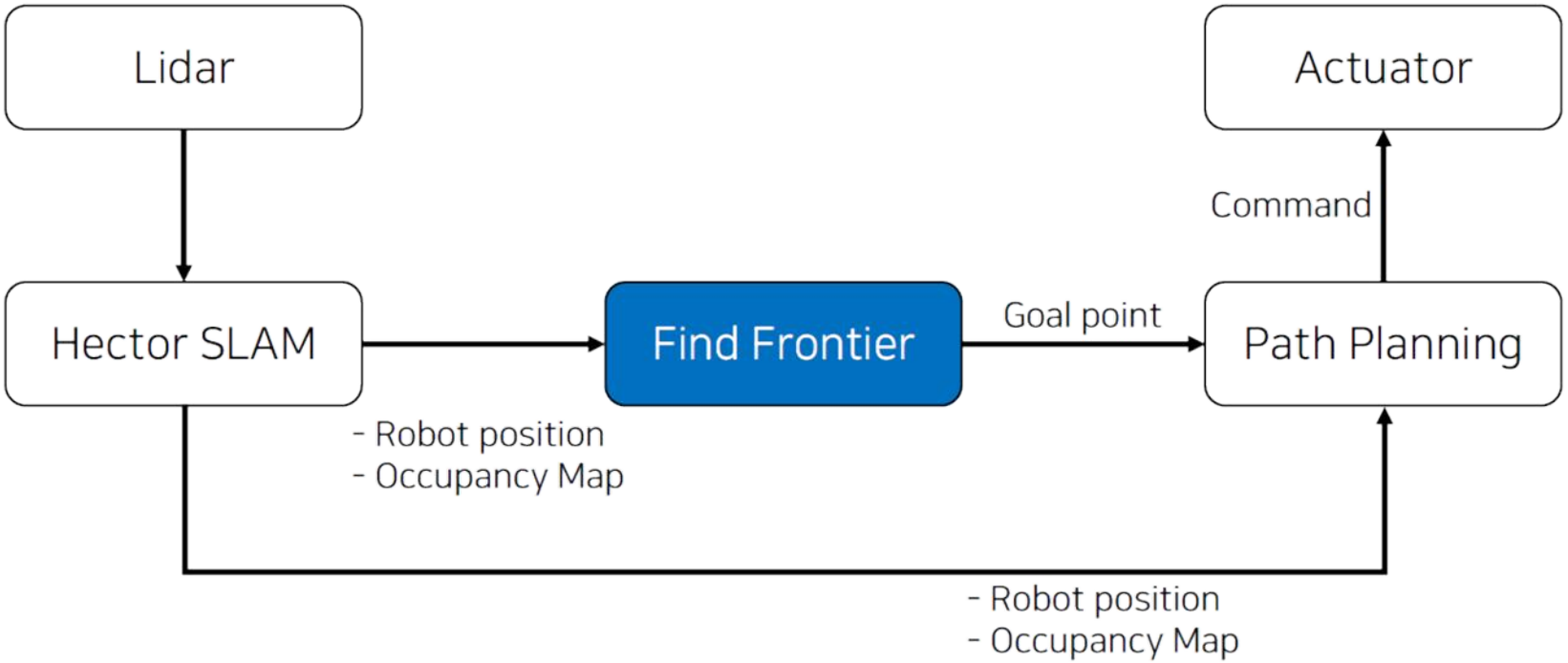




Set the goal point to explore all the space in a closed space

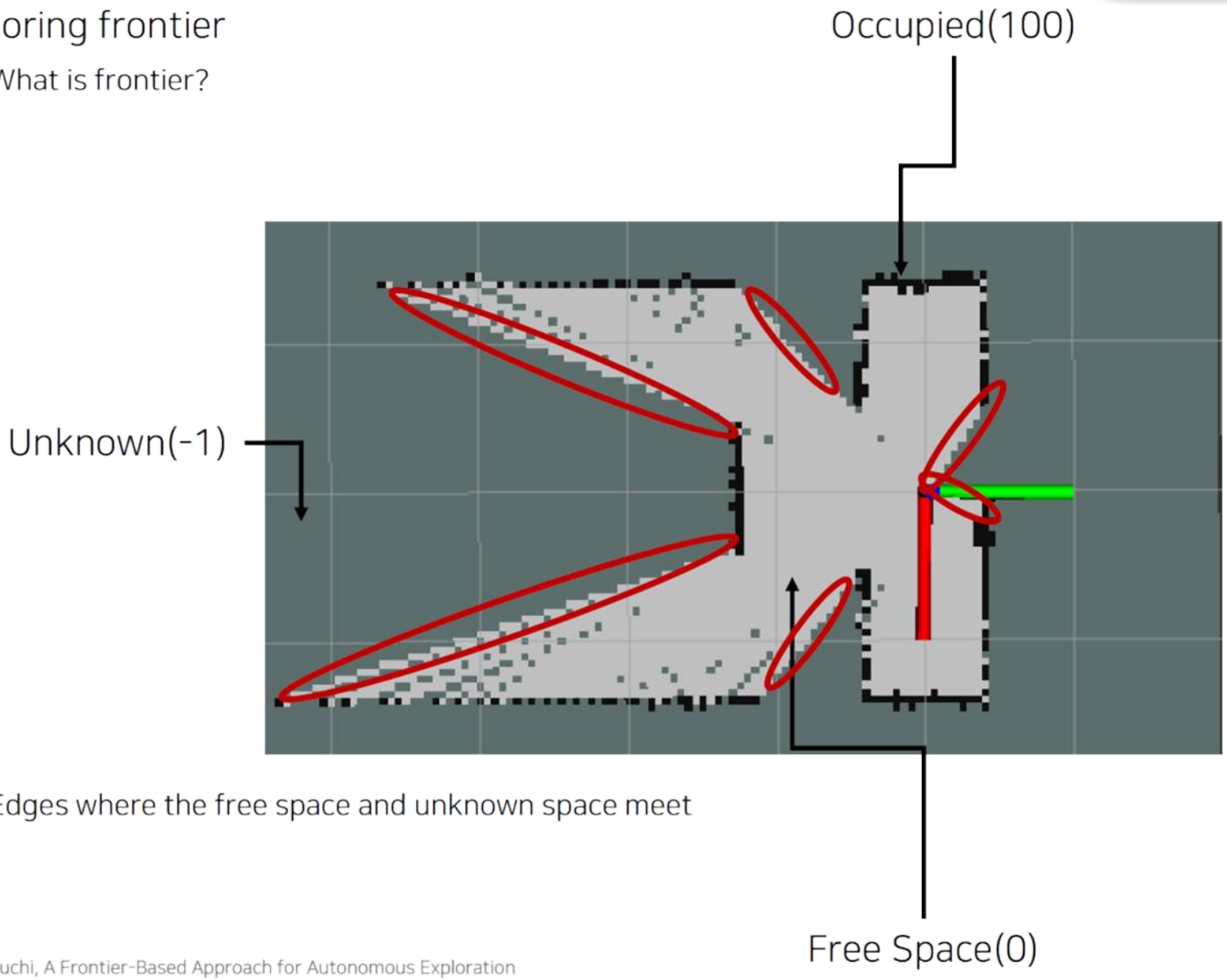
Exploring frontier

What we need to do



Exploring frontier

What is frontier?

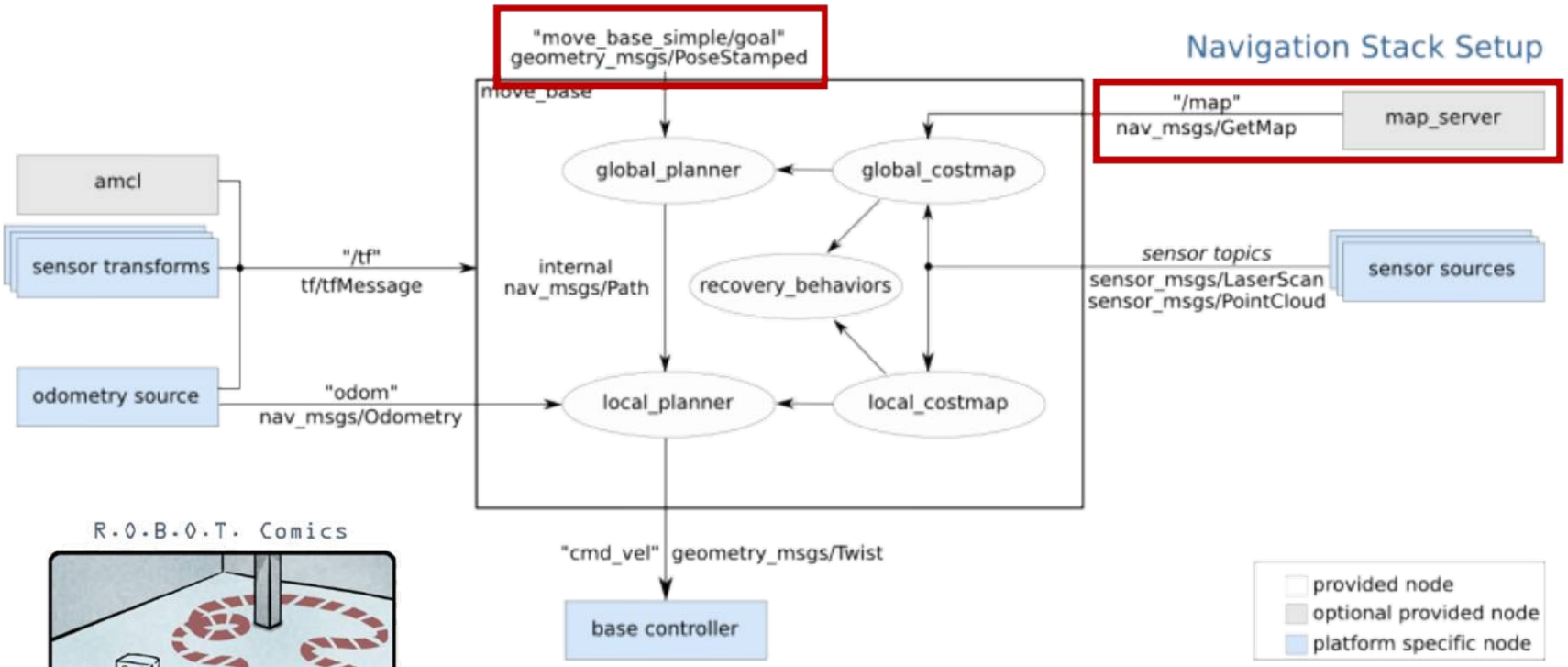


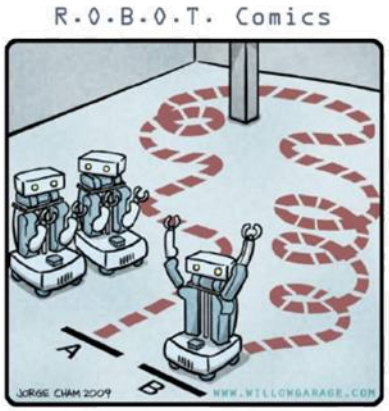
Edges where the free space and unknown space meet

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

About move_base package

Visualization results in rviz





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

Execute the code

$ sudo apt-get install ros-noetic-move-base
$ sudo apt-get install ros-noetic-dwa-local-planner
$ rosrun rviz rviz

$ roslaunch find_frontier move_base.launch
$ rosrun find_frontier find_frontier_node

$ rostopic pub /move_base_simple ….

```
jjw@JJW:~/catkin_ws$ rostopic pub /move_base_simple/goal geometry_msgs/PoseStamped "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: 'map'
pose:
  position:
    x: 1.0
    y: -2.0
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.14
    w: 0.98"
```

Orientation is represented using quaternion. If you want to know yaw angle you have to convert it.

Summary

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

Useful tools
- rviz, rqt, rqt_graph
- rostopic list (–s –p), rostopic pub, rostopic echo
- catkin_make, catkin_make clean

Required
- $ git clone https://github.com/Sol-Han/Hyundai_NGV_Mobile_Robot.git
- $ git clone https://github.com/tu-darmstadt-ros-pkg/hector_slam

move source code & catkin_make
- $ sudo apt-get install ros-noetic-teleop-twist-keyboard
- $ sudo apt-get install ros-noetic-move-base
- $ sudo apt-get install ros-noetic-dwa-local-planner

If there is a problem, Simple and Fast way is reinstalling ros.

$ sudo apt-get remove ros-* (remove ros)

And install ros again
wiki.ros.org/noetic/Installation/Ubuntu

Phase 2 : Knowing the detailed procedure

Gazebo Model

Let's look at the file m_robot_world.launch

```
<launch>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

  <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="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro.py '$(find m_robot_description)/urdf/
m_robot.xacro'" />

  <node name="m_robot_spawner" pkg="gazebo_ros" type="spawn_model" output="screen"
    args="-urdf -param robot_description -model m_robot "/>
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
    <param name="use_gui" value="FALSE"/>
  </node>
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>

</launch>
```

Load m_robot.world

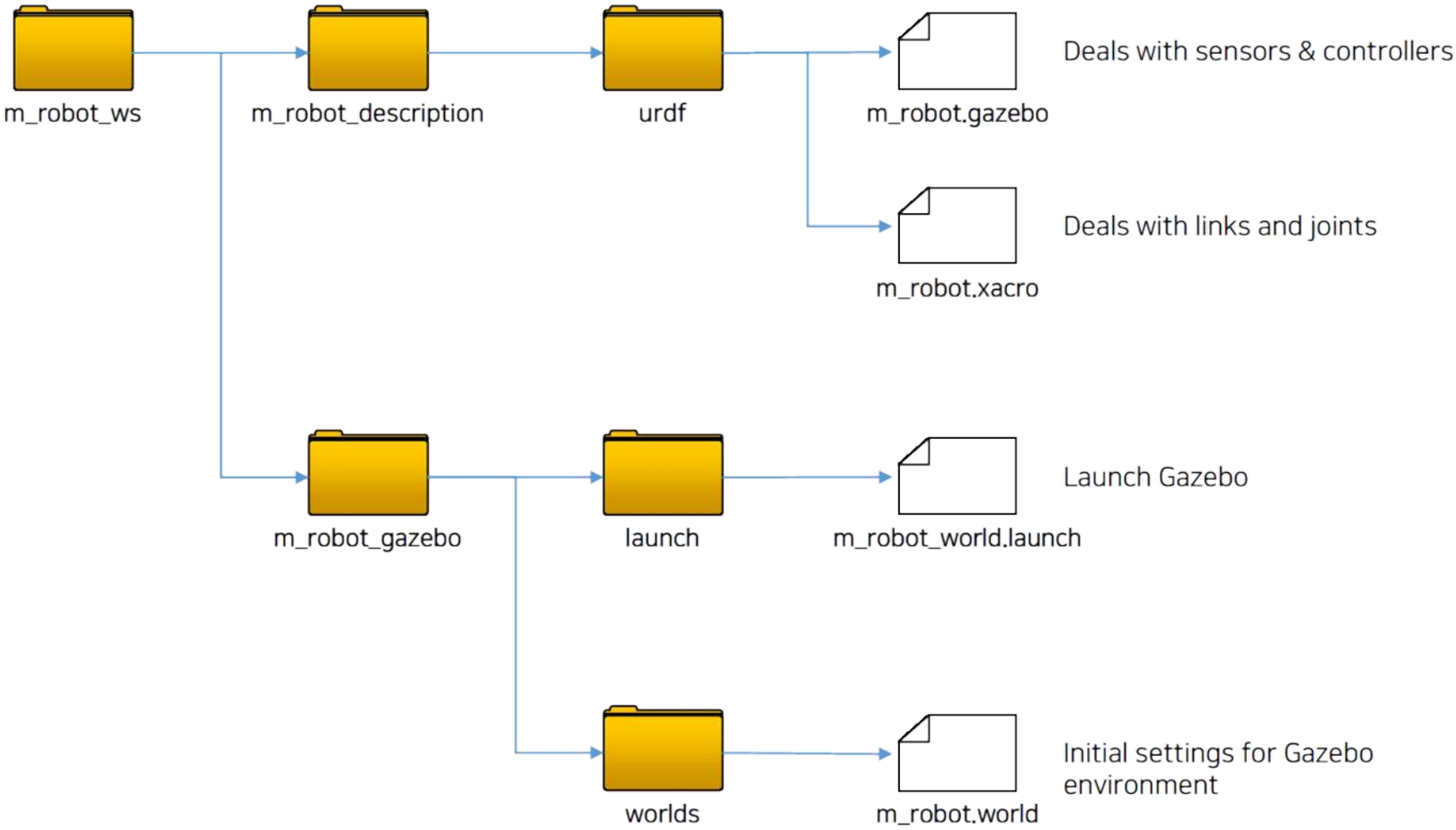
Load m_robot.gazebo, m_robot.xacro

Then, let's look at the file m_robot.world

```
-<sdf version="1.4">
 -<!--
    We use a custom world for the m_robot so that the camera angle is launched correctly
  -->
 -<world name="default">
  -<!--
    include>
          <uri>model://willowgarage</uri>
        </include
   -->
  -<include>
     <uri>model://ground_plane</uri>
   </include>
   <!-- Global light source -->
  -<include>
     <uri>model://sun</uri>
   </include>
   <!-- Focus camera on tall pendulum -->
  -<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>
     <pose>0 -2.0 0.0 0.0 0.0 0.0</pose>
   </include>
 </world>
</sdf>
```

On the other hand, m_robot.gazebo, m_robot.xacro describes mobile robot

Overall structure



This vehicle uses “skid_steer_drive_controller” (in m_robot.gazebo)

```
-<plugin name="skid_steer_drive_controller" filename="libgazebo_ros_skid_steer_drive.so">
    <updateRate>100</updateRate>
    <leftFrontJoint>left_wheel_front_hinge</leftFrontJoint>
    <leftRearJoint>left_wheel_back_hinge</leftRearJoint>
    <rightFrontJoint>right_wheel_front_hinge</rightFrontJoint>
    <rightRearJoint>right_wheel_back_hinge</rightRearJoint>
    <wheelSeparation>0.2</wheelSeparation>
    <wheelDiameter>0.10</wheelDiameter>
    <torque>8</torque>
    <commandTopic>cmd_vel</commandTopic>
    <robotBaseFrame>base_frame</robotBaseFrame>
  </plugin>
</gazebo>
```

- There are many kinds of controllers. http://wiki.ros.org/steer_drive_controller
- This driver subscribes topic **cmd_vel (Fomat : geometry_msgs/Twist)**
- **Robot will move following /cmd_vel command until receiving new command**

- We used teleop_twist_keyboard package which sends topic /cmd_vel

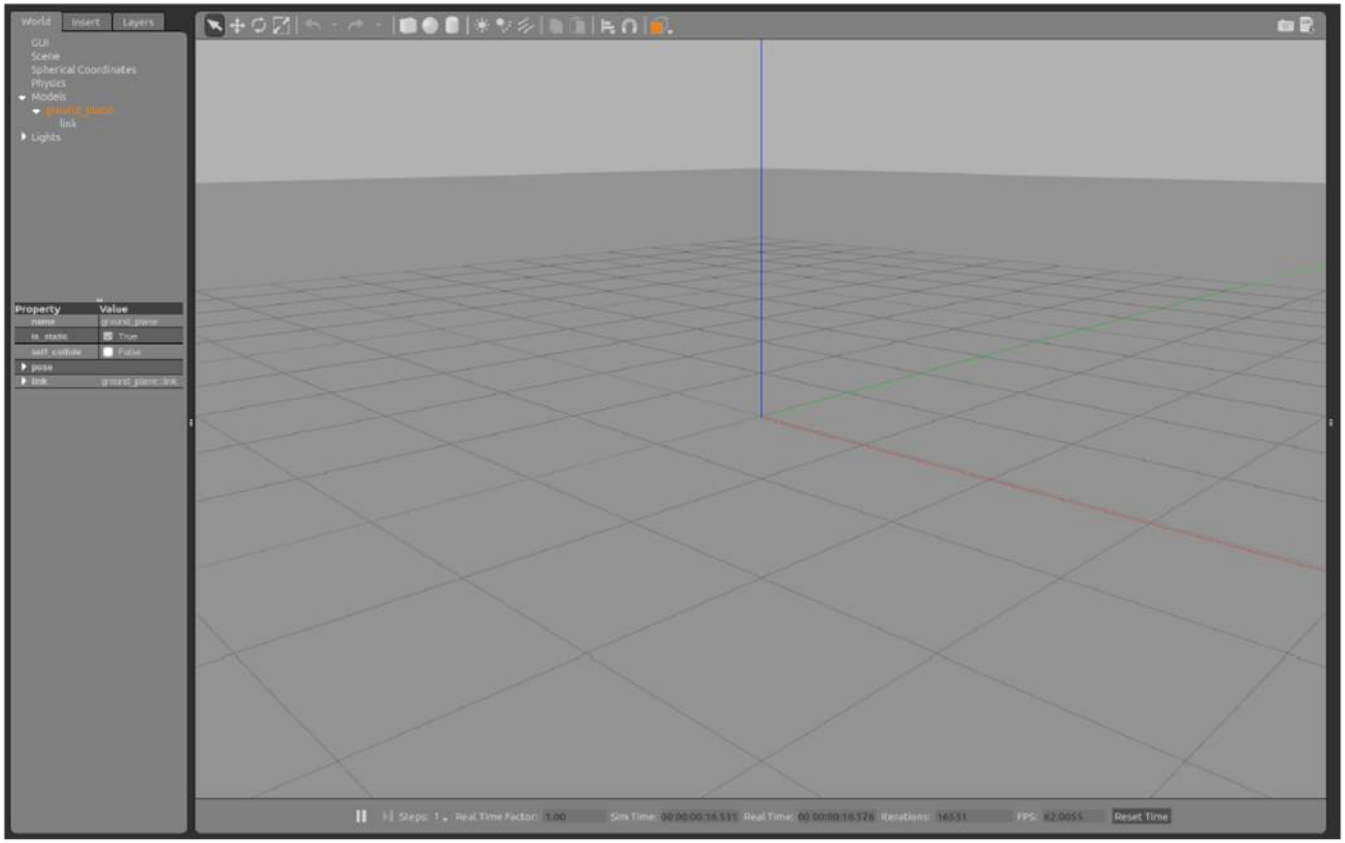
- Therefore, you can directly input the desired velocity of the vehicle.
- rostopic pub /cmd_vel ….. Using rqt

Customize

Making a structure

First, let's make the maze structure

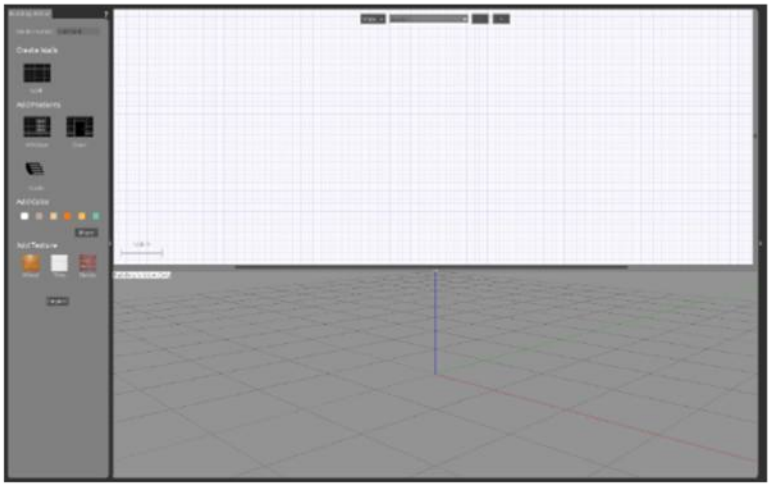
```
~$ rosrun gazebo_ros gazebo
```



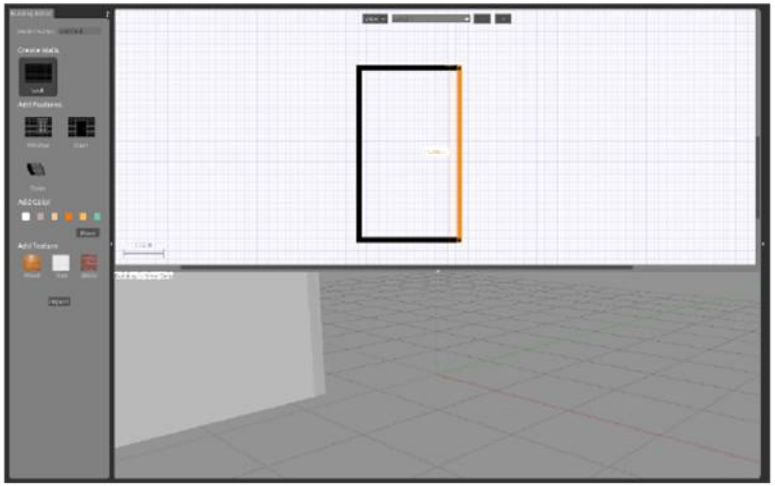
Press ctrl+b to enter the building editor

```
ctrl+b
```

Making a structure

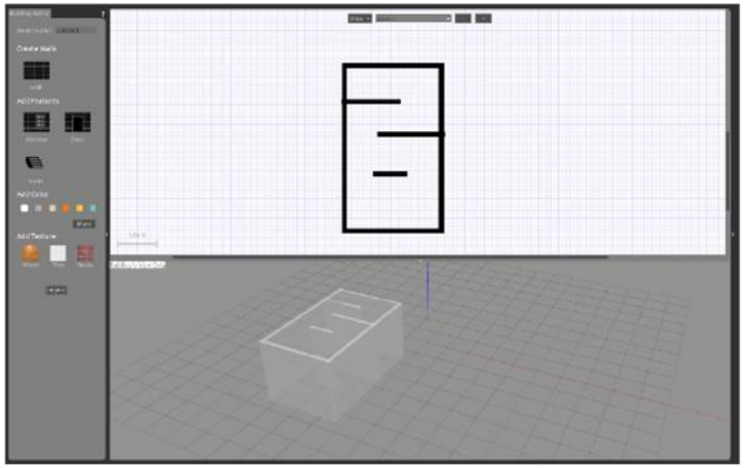


Click on the wall button on the top left, and create a 3 by 5 rectangular structure

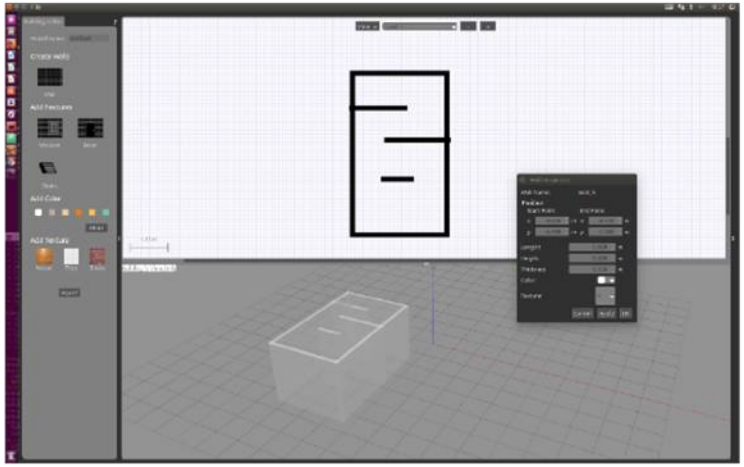


Making a structure

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

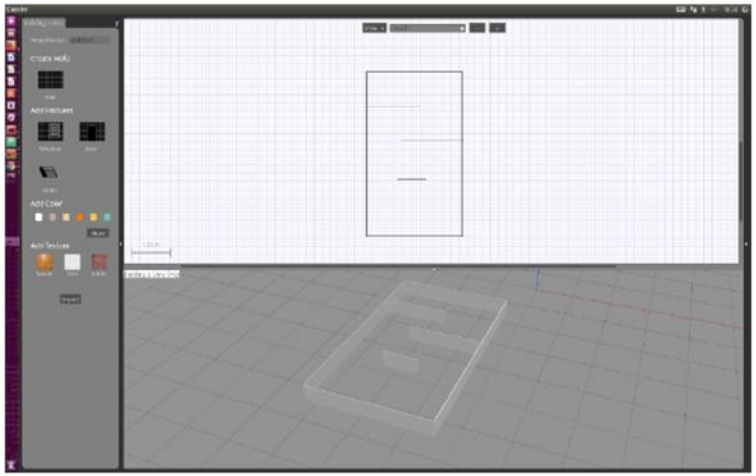


Right click the wall and open the wall inspector

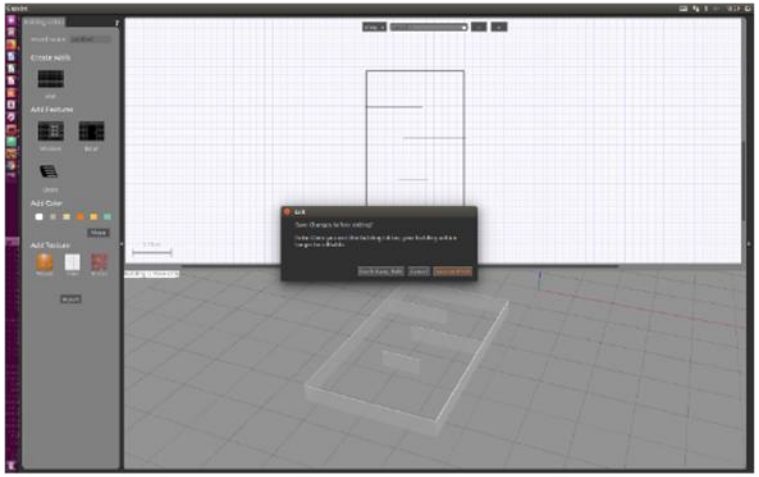


Making a structure

Set the height as 0.4m and thickness as 0.02m

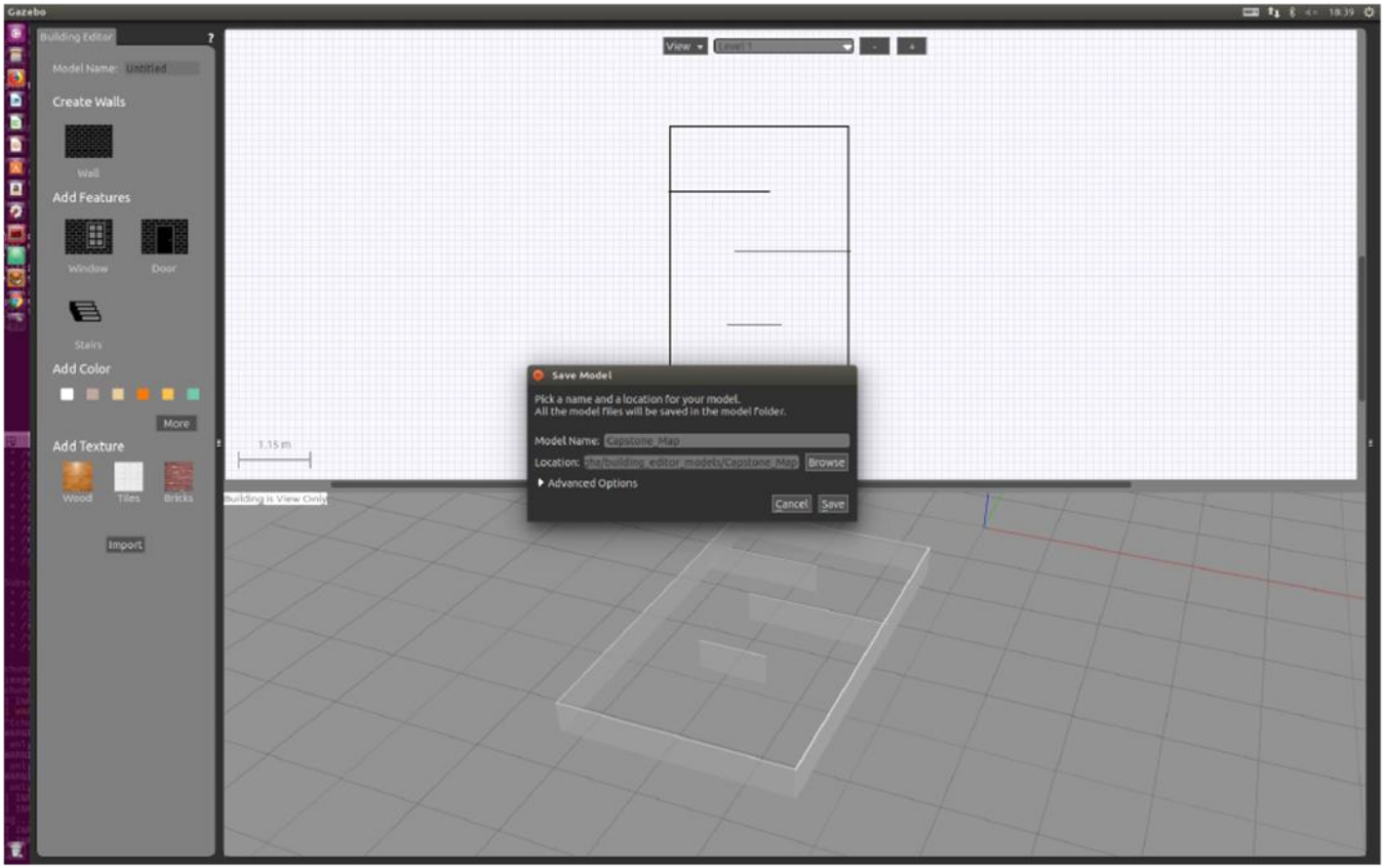


Close the building editor by clicking File -> Exit building Editor or by ctrl+x



Press Save and Exit

Making a structure



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

$ cp –r ~/bulding_editor_models“---” ~/.gazebo/models

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

m_robot design : m_robot.xacro

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 :

```
<?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>
```

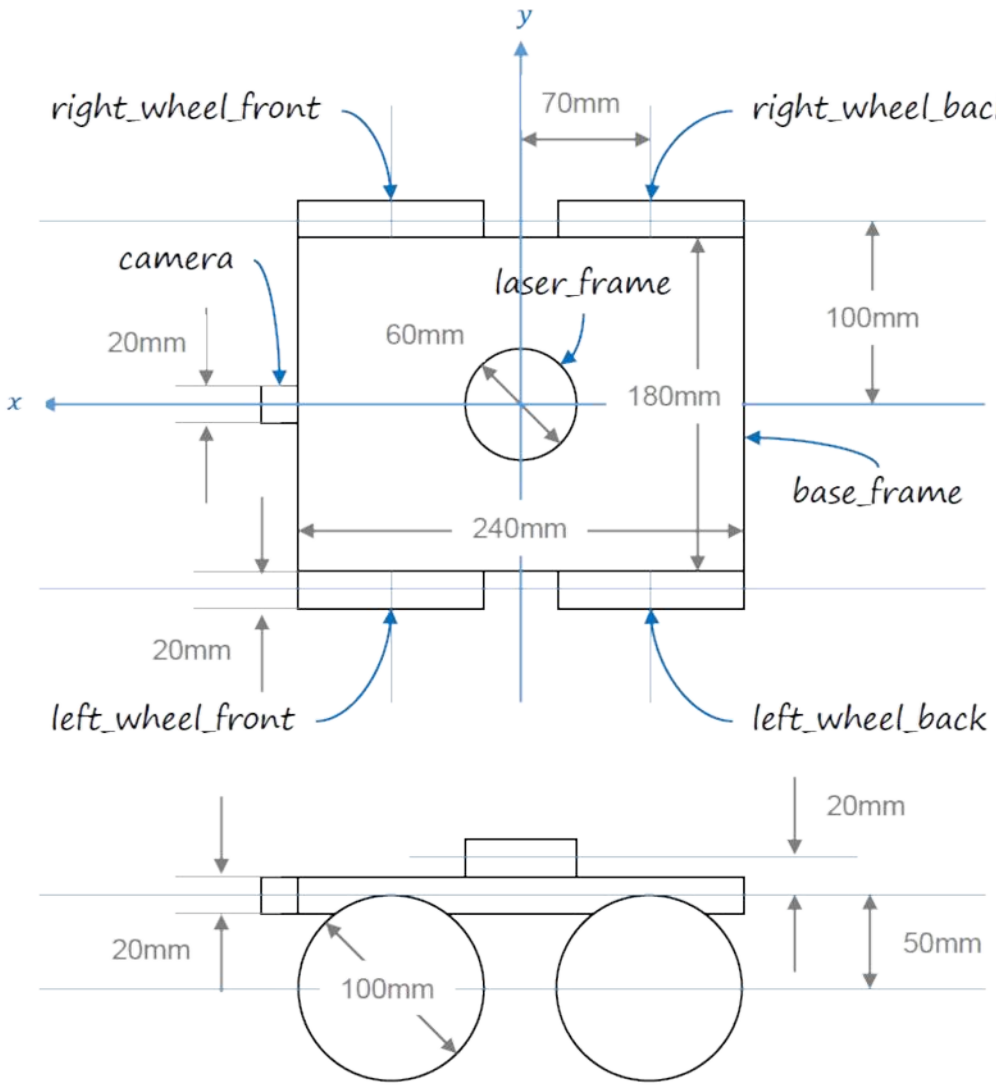
m_robot.xacro

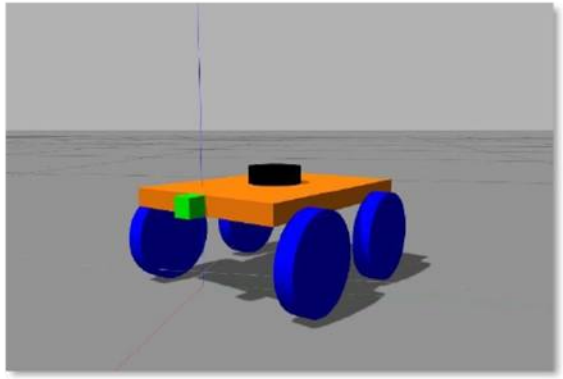
Notice that the classes are closed with /
<classname> : starts class
</classname> : closes class

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>
...
...
```

[제목 없음]

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

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

Setup for Gazebo environment


```
<?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

m_robot.world

```
<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)
```

SLAM & Find Frontier

Let's look at the file nav_run.launch

```
-<launch>
   <!--- Run hector mapping -->
   <include file="$(find find_frontier)/launch/hector_mapping.launch"/>
   <!--- Run Move Base -->
   <node pkg="find_frontier" type="find_frontier_node" respawn="false" name="find_frontier" output="screen"/>
   <include file="$(find find_frontier)/launch/move_base.launch"/>
   <!-- sychronize time -->
   <!--  <node name="rviz" pkg="rviz" type="rviz"/>  -->
   <rosparam> use_sim_time: true </rosparam>
 </launch>
```

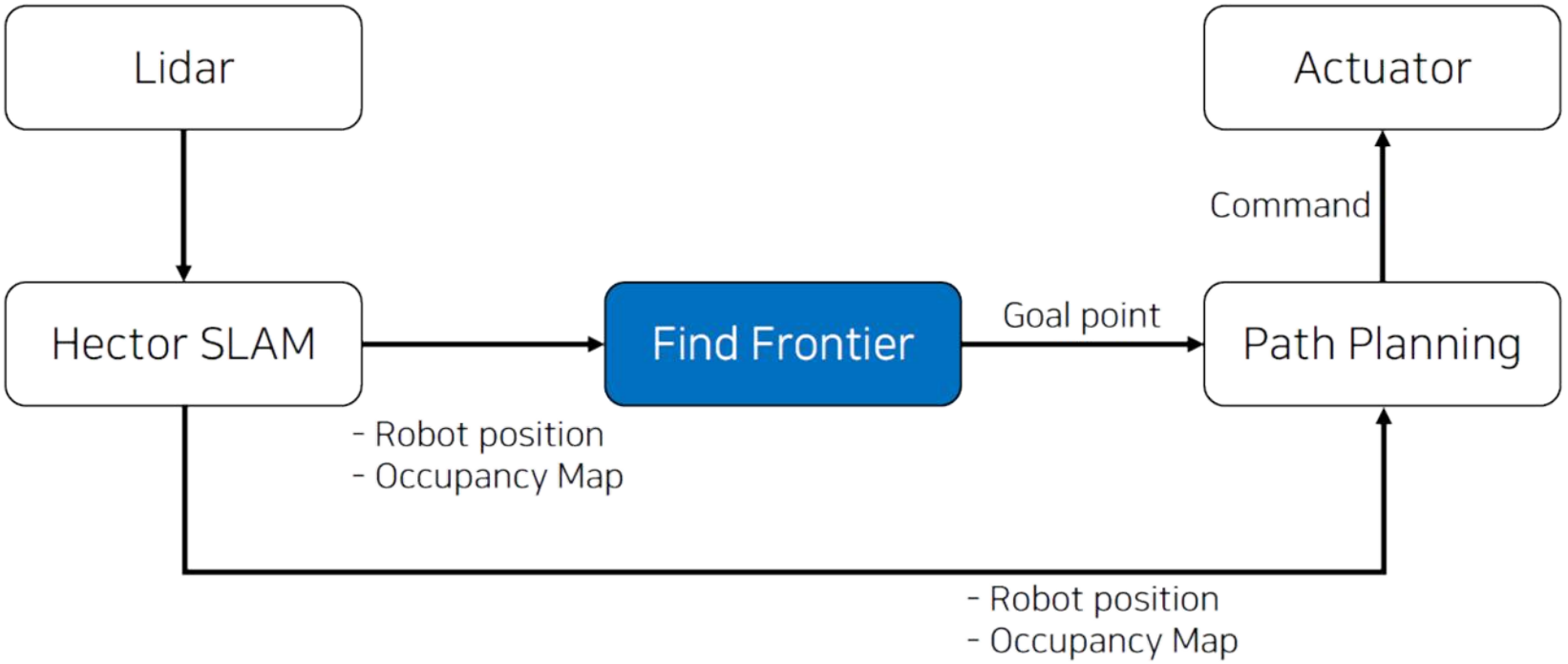
hector_slam (SLAM package) and move_base (Path planner) are public code
- hector_slam package was downloaded and built by catkin_make
- move_base was initially installed with ROS

SLAM informs the location of the vehicle and map
Path planner makes the vehicle move to desired position without collision

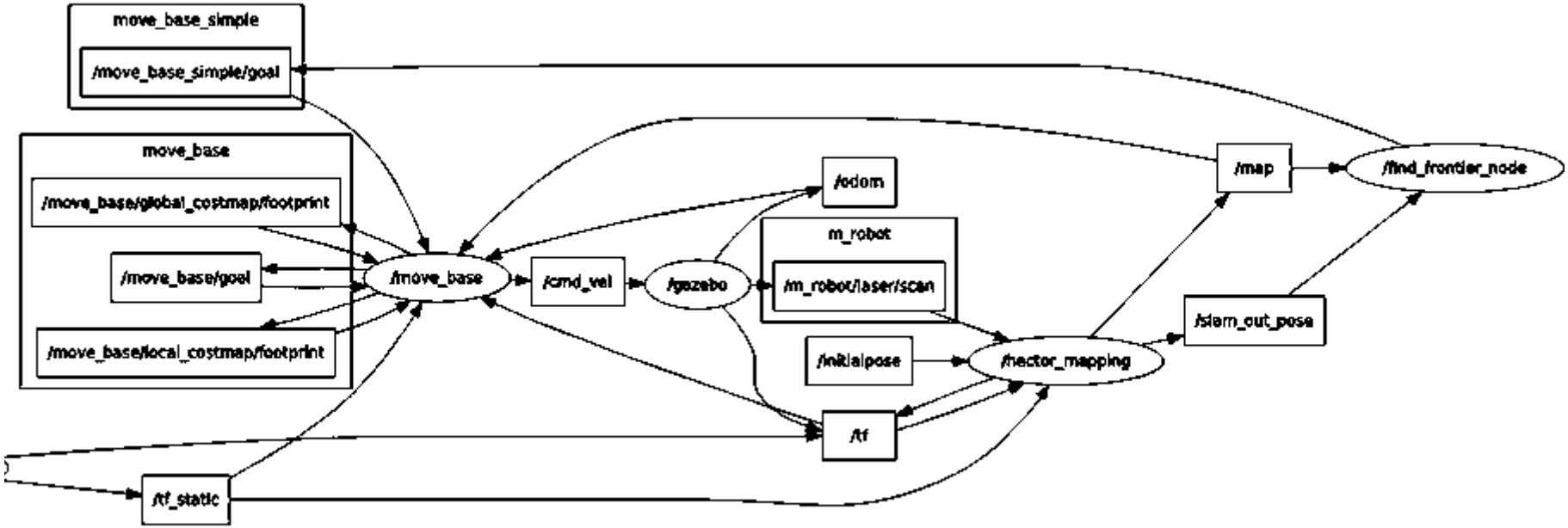
Find frontier decides to where to go (written by 2018 TA. Dongha Chung)
- This algorithm finds the largest undiscovered region (frontier) and publishes desired waypoint message (/move_base_simple/goal) to move_base node.
- Then move_base will command to robot (/cmd_vel) to move to destination while avoiding collision to obstacles.

Exploring frontier

What we need to do

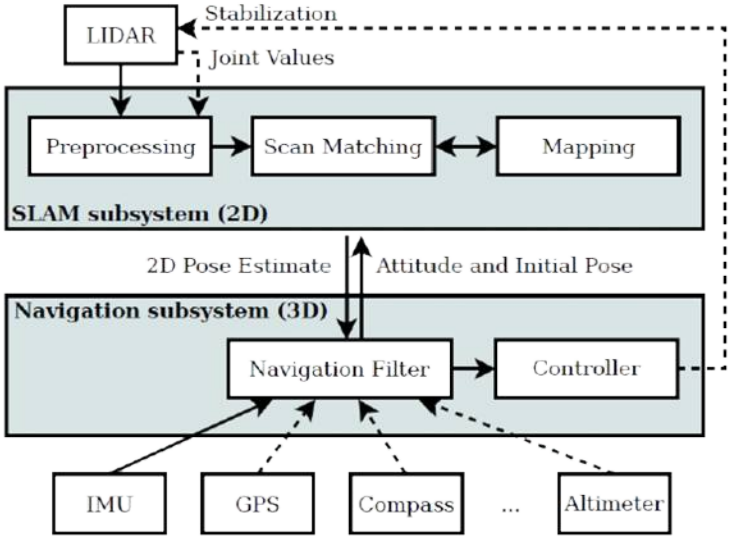


rqt_graph

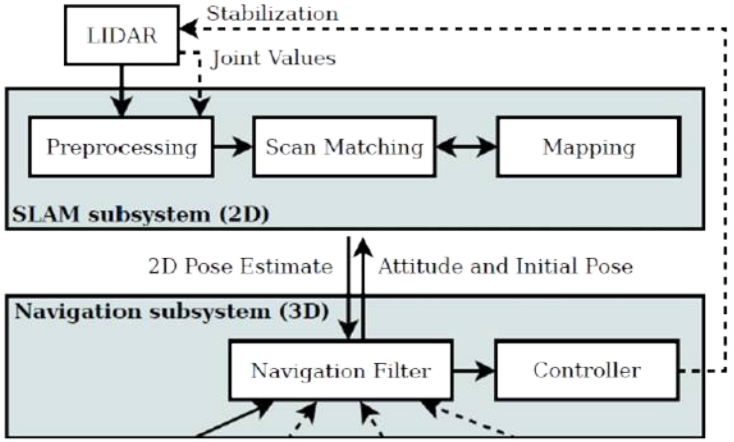

move_base_simple
/move_base_simple/goal
move_base
/move_base/global_costmap/footprint
/move_base/goal
/move_base/local_costmap/footprint
/move_base
/cmd_vel
/gazebo
/odom
m_robot
/m_robot/laser/scan
/initialpose
/hector_mapping
/tf
/tf_static
/map
/find_frontier_node
/slam_out_pose

About Hector SLAM

Overall Algorithm



Hector SLAM framework



Algorithm Step 1. Transformation of scan data to "stabilized frame"

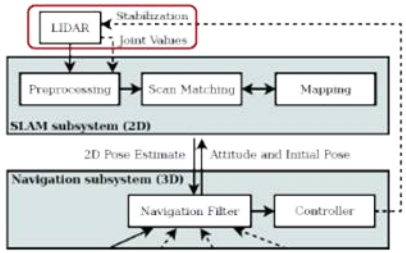
Algorithm Step 2. Interpolation of occupancy grid map

Algorithm Step 3. Scan matching & Mapping

Algorithm Step 4. Navigation Filter

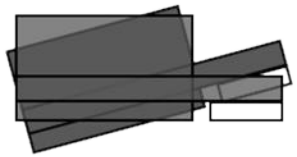
About Hector SLAM

Algorithm Step 1. Transformation of scan data to "satablized frame"



Lidar Scan Data

Lidar scan data



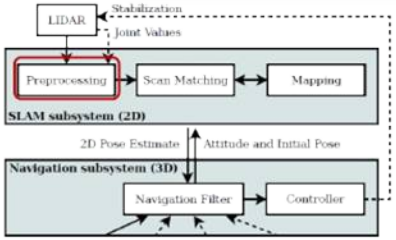
Attitude data

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

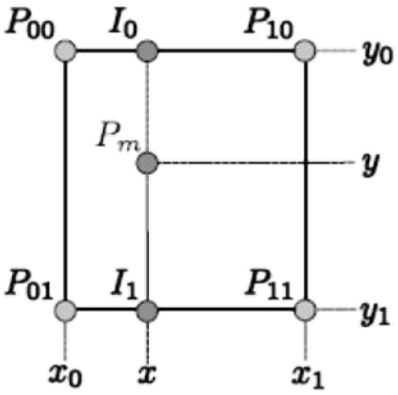
2D Scan data in Stabilized Frame

Lidar attitude data (ϕ, θ, ψ)

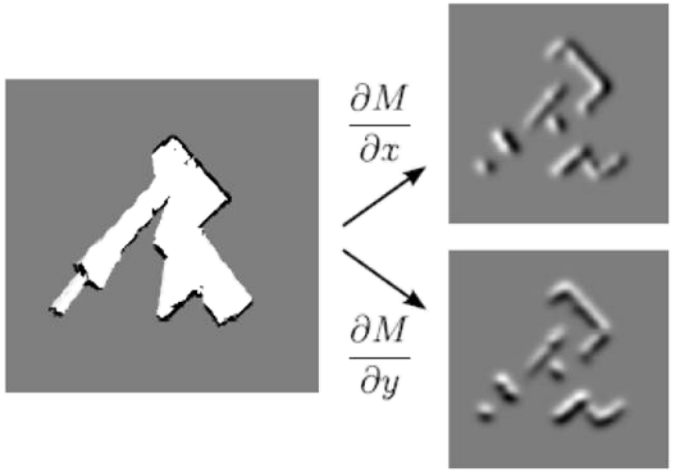
About Hector SLAM



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

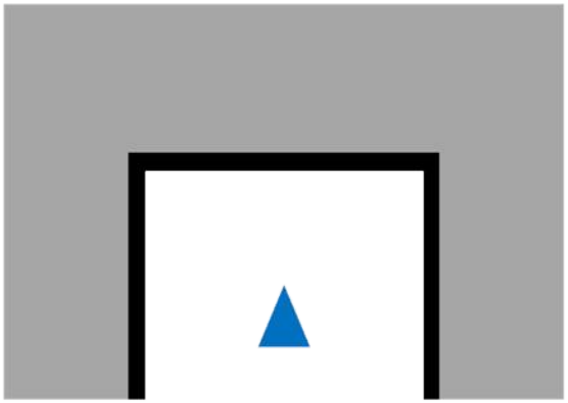
$$M(P_m) \approx \frac{y-y_0}{y_1-y_0}\left(\frac{x-x_0}{x_1-x_0}M(P_{11})+\frac{x_1-x}{x_1-x_0}M(P_{01})\right) + \frac{y_1-y}{y_1-y_0}\left(\frac{x-x_0}{x_1-x_0}M(P_{10})+\frac{x_1-x}{x_1-x_0}M(P_{00})\right)$$

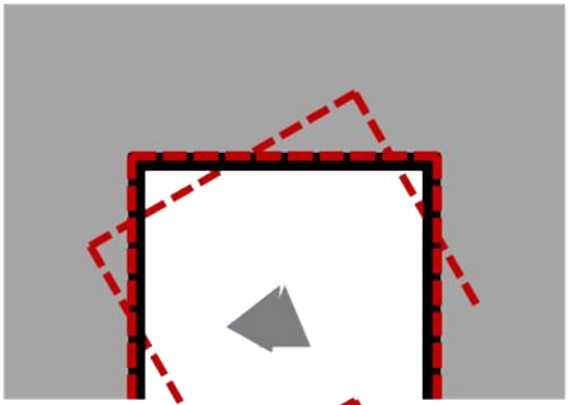
$$\frac{\partial M}{\partial x}(P_m) \approx \frac{y-y_0}{y_1-y_0}(M(P_{11})-M(P_{01})) + \frac{y_1-y}{y_1-y_0}(M(P_{10})-M(P_{00}))$$

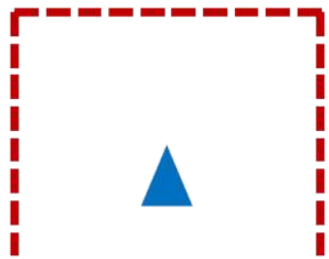
$$\frac{\partial M}{\partial y}(P_m) \approx \frac{x-x_0}{x_1-x_0}(M(P_{11})-M(P_{10})) + \frac{x_1-x}{x_1-x_0}(M(P_{01})-M(P_{00}))$$

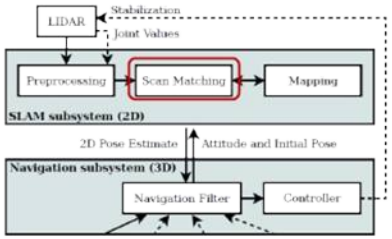
About Hector SLAM

Algorithm Step 3. Scan matching









Error covariance

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

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

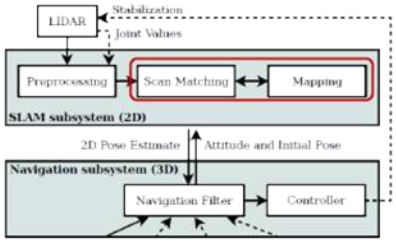
$$\boldsymbol{\xi} = (p_x, p_y, \psi)^{\mathrm{T}} \longrightarrow \boldsymbol{\xi}^* = \underset{\boldsymbol{\xi}}{\operatorname{argmin}} \sum_{i=1}^{n} \left[1 - M(\mathbf{S}_i(\boldsymbol{\xi}))\right]^2$$

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

Gauss Newton method

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)

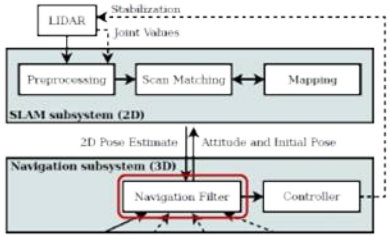
Map $(M(x, y))$
Pose (p_x^f, p_y^f, ψ^f)

Pose (p_x^1, p_y^1, ψ^1)

Pose (p_x^2, p_y^2, ψ^2)

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

$$\tilde{y}_{k+1} = z_{k+1} - h(\hat{x}^-_{k+1})$$
$$S_{k+1} = H_{k+1} P^-_{k+1} H^T_{k+1} + R_{k+1}$$
$$K_{k+1} = P^-_{k+1} H^T_{k+1} S^{-1}_{k+1}$$

Time step $k+1$

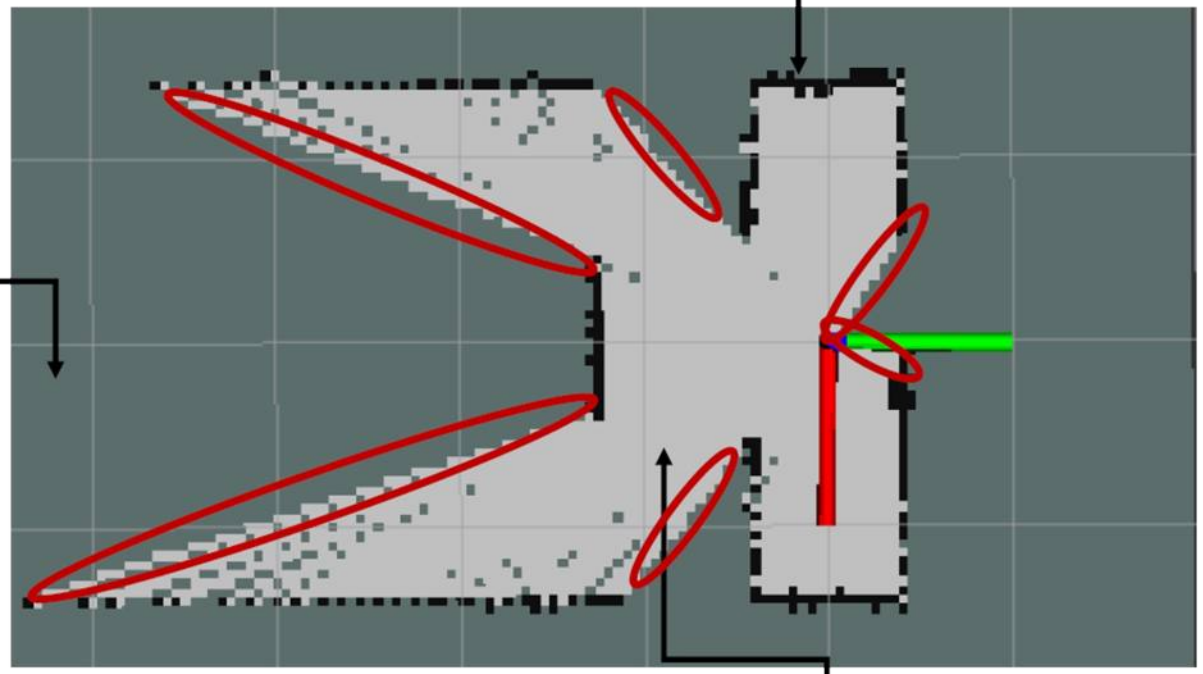
$$\hat{x}_{k+1} = \hat{x}^-_{k+1} + K_{k+1}\tilde{y}_{k+1}$$
$$P_{k+1} = (I - K_{k+1}H_{k+1})P^-_{k+1}$$

Prediction

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

Exploring frontier

What is frontier?

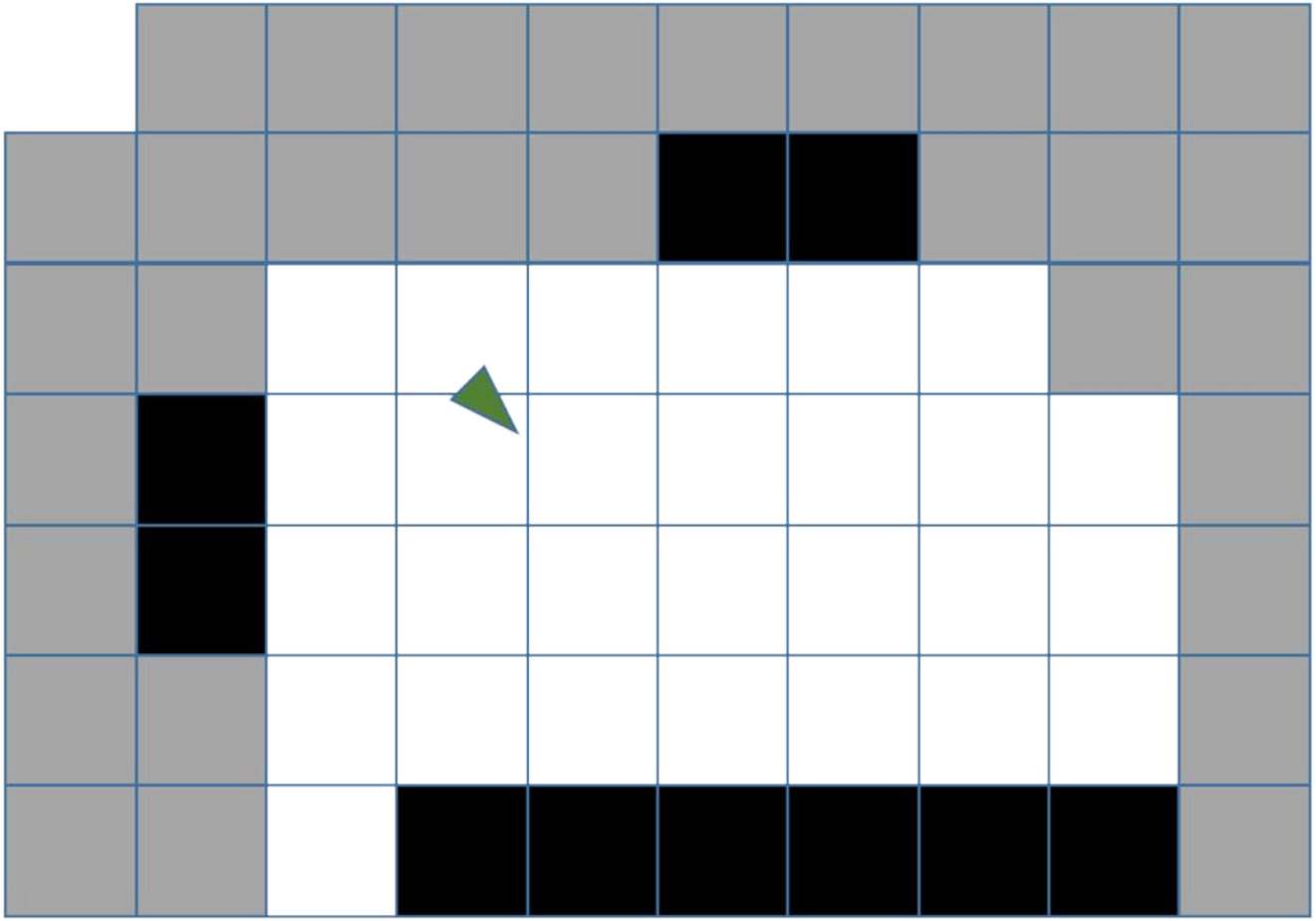


Edges where the free space and unknown space meet

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

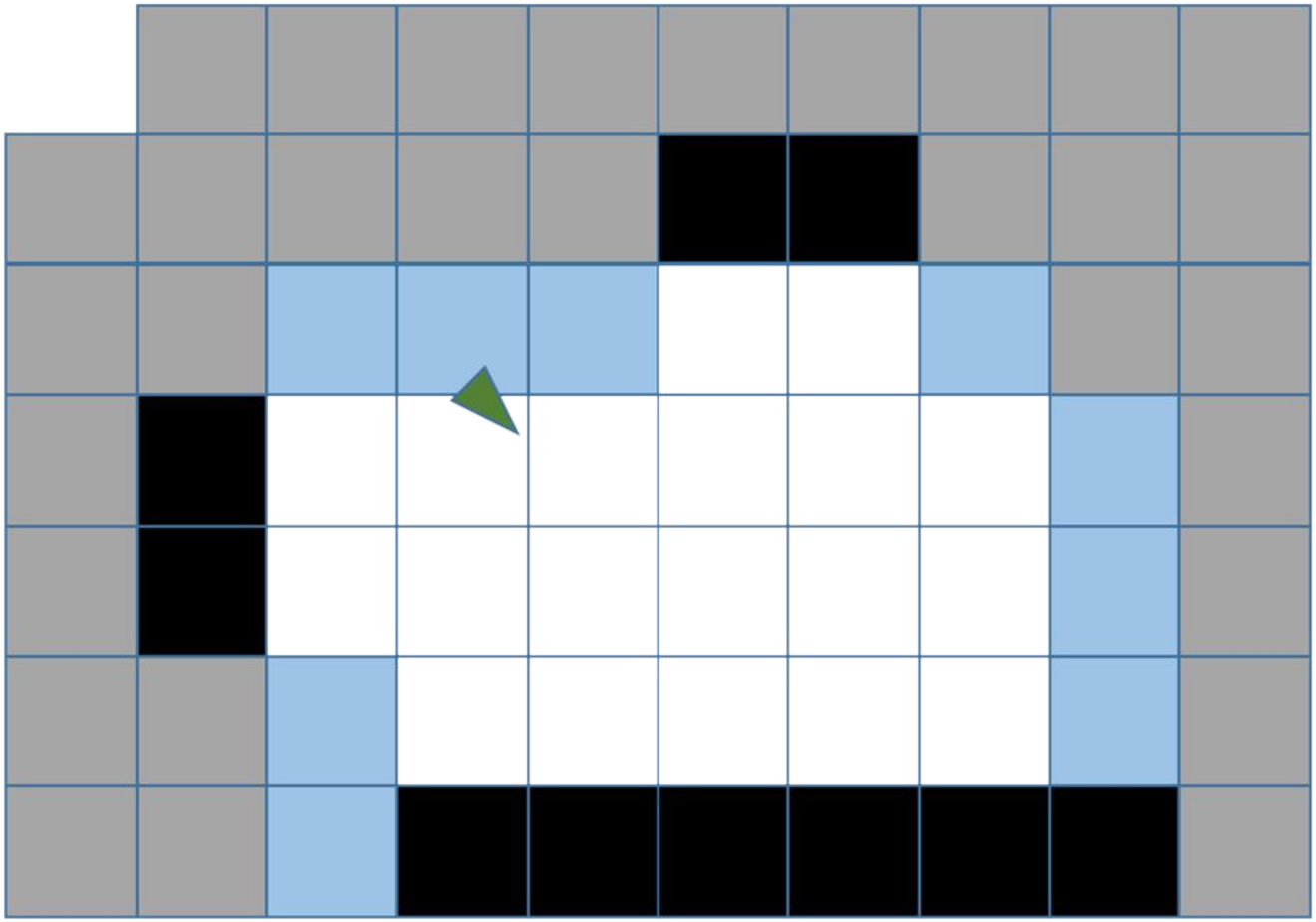
Exploring frontier

Algorithm overview



Exploring frontier

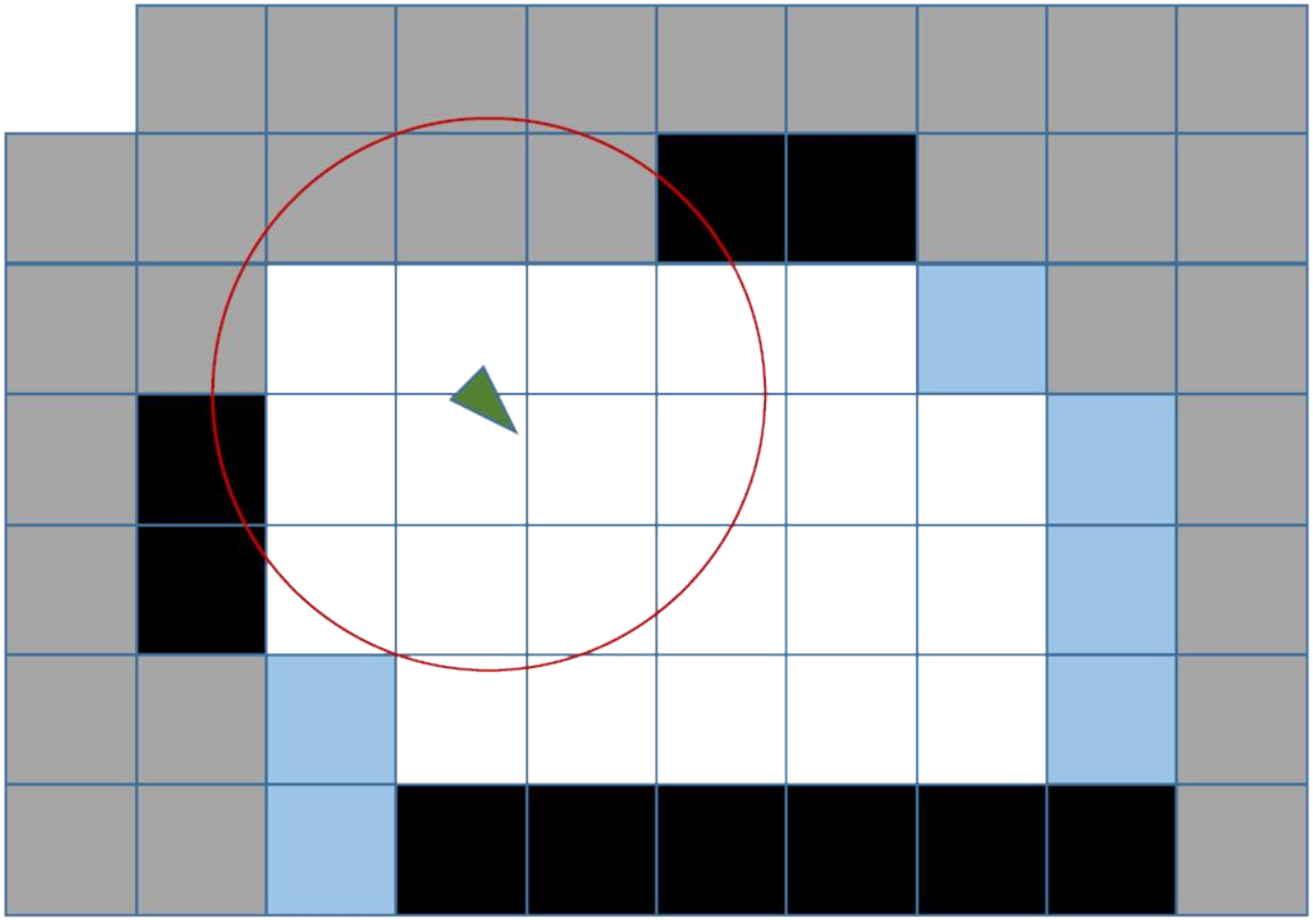
Algorithm overview : Candidate cells



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

Exploring frontier

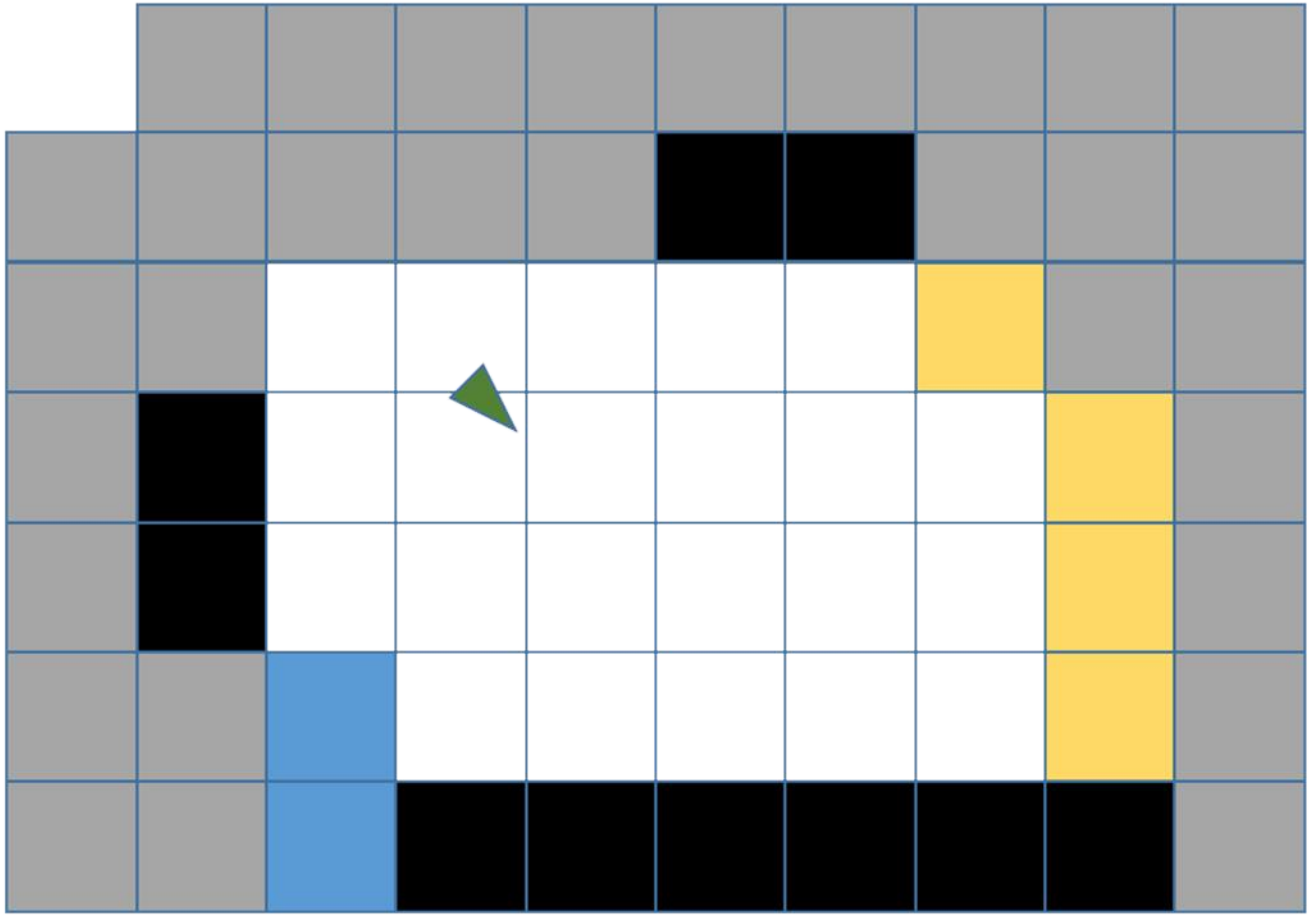
Algorithm overview : Candidate cells



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

Exploring frontier

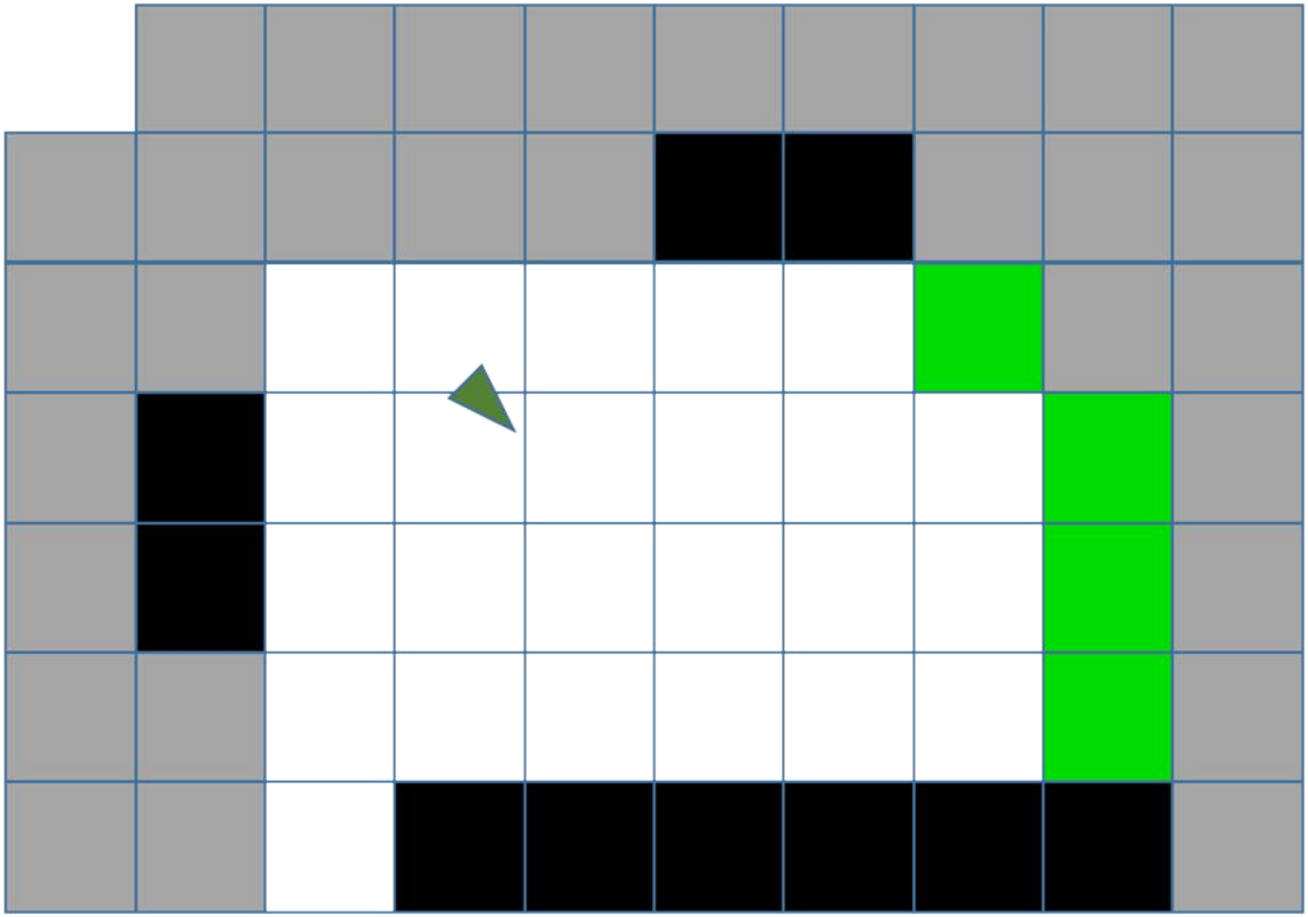
Algorithm overview : Candidate cells



Cluster the candidate cells that are connected

Exploring frontier

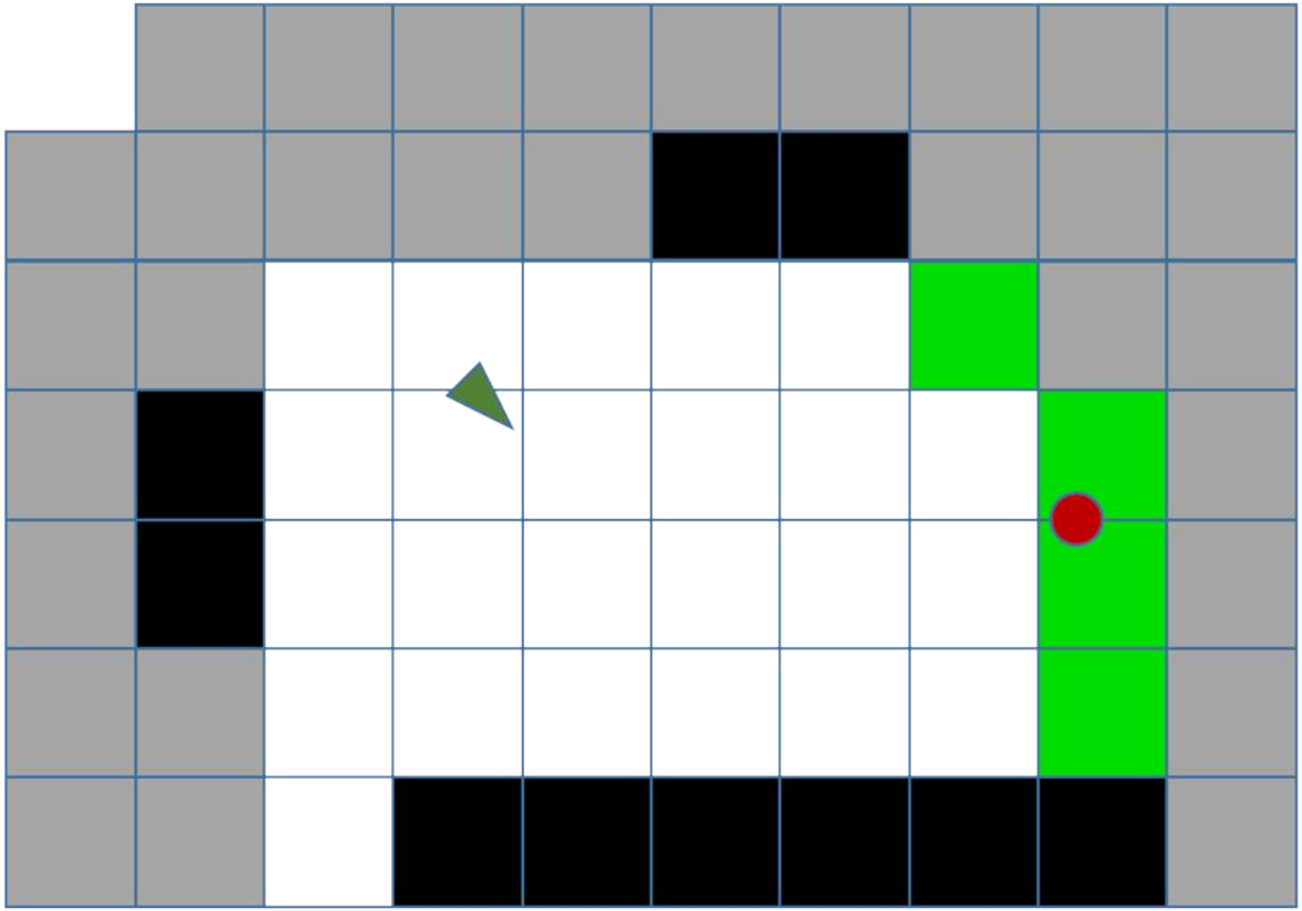
Algorithm overview : Candidate cells



Select the cluster that has most cells

Exploring frontier

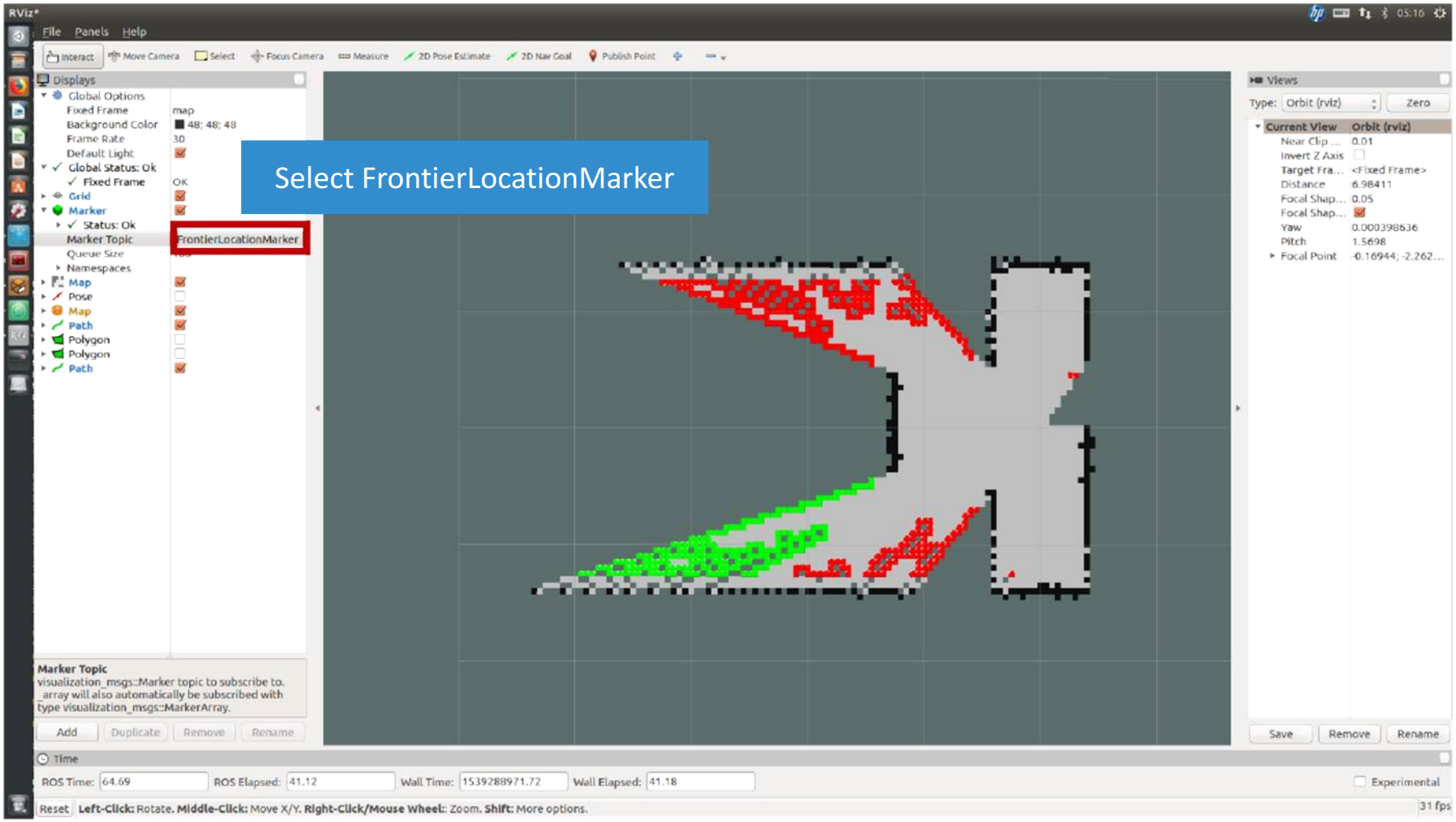
Algorithm overview : Candidate cells



Set the center of cluster to be the goal

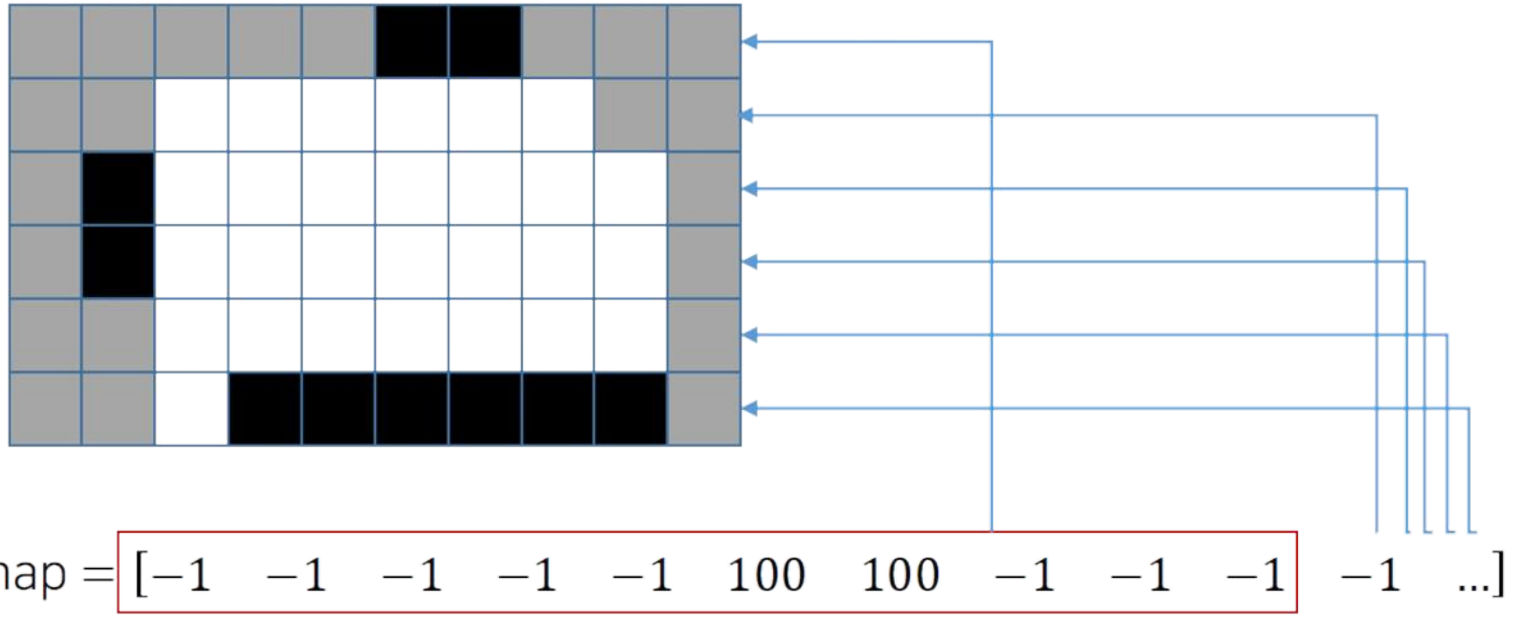
Exploring frontier

Run example



Exploring frontier

Code review



Customize

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]
…
```

[제목 없음]

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>
```

```
~$ 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)
…
```

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
```

Move_base

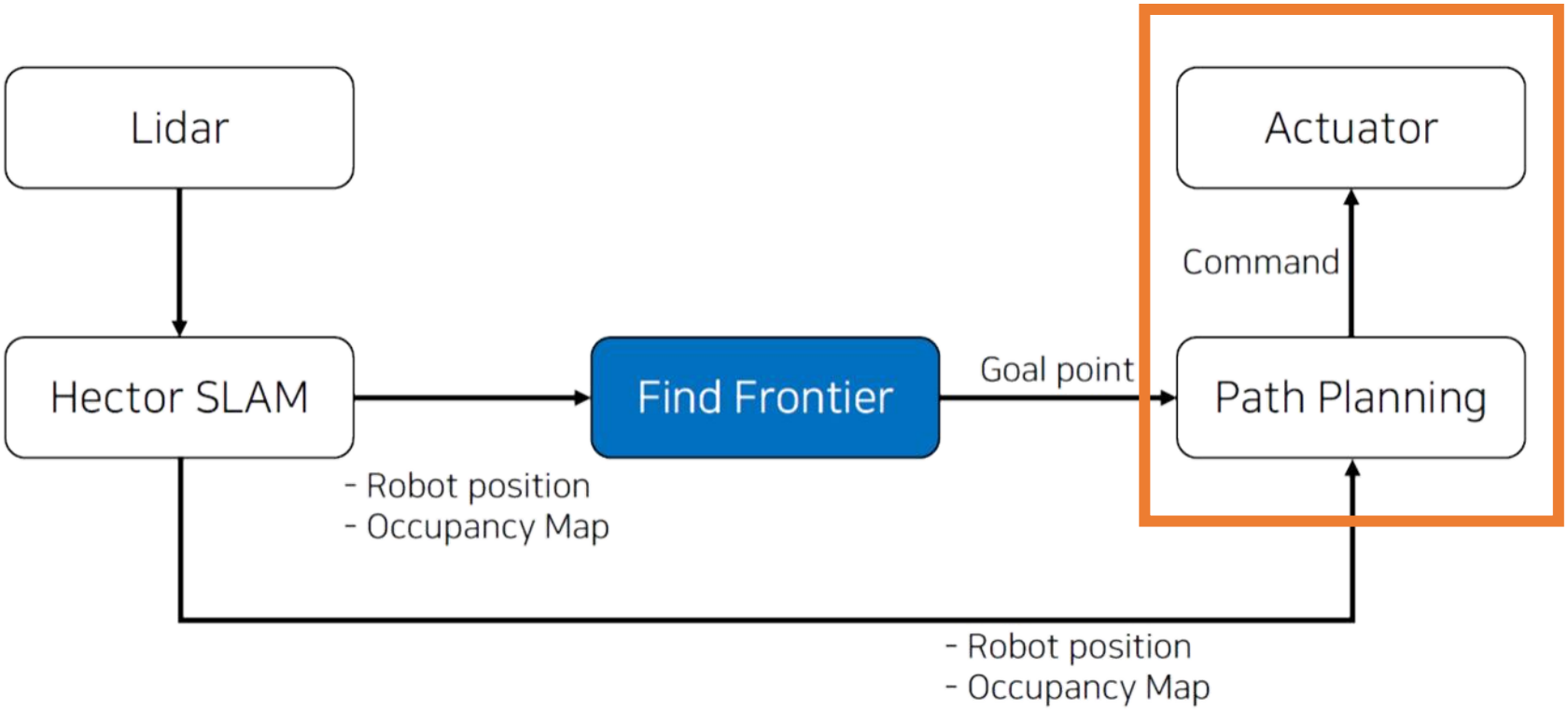
Let’s look at the file nav_run.launch

```
-<launch>
   <!--- Run hector mapping -->
   <include file="$(find find_frontier)/launch/hector_mapping.launch"/>
   <!--- Run Move Base -->
   <node pkg="find_frontier" type="find_frontier_node" respawn="false" name="find_frontier" output="screen"/>
   <include file="$(find find_frontier)/launch/move_base.launch"/>
   <!-- sychronize time -->
   <!--  <node name="rviz" pkg="rviz" type="rviz"/>  -->
   <rosparam> use_sim_time: true </rosparam>
 </launch>
```

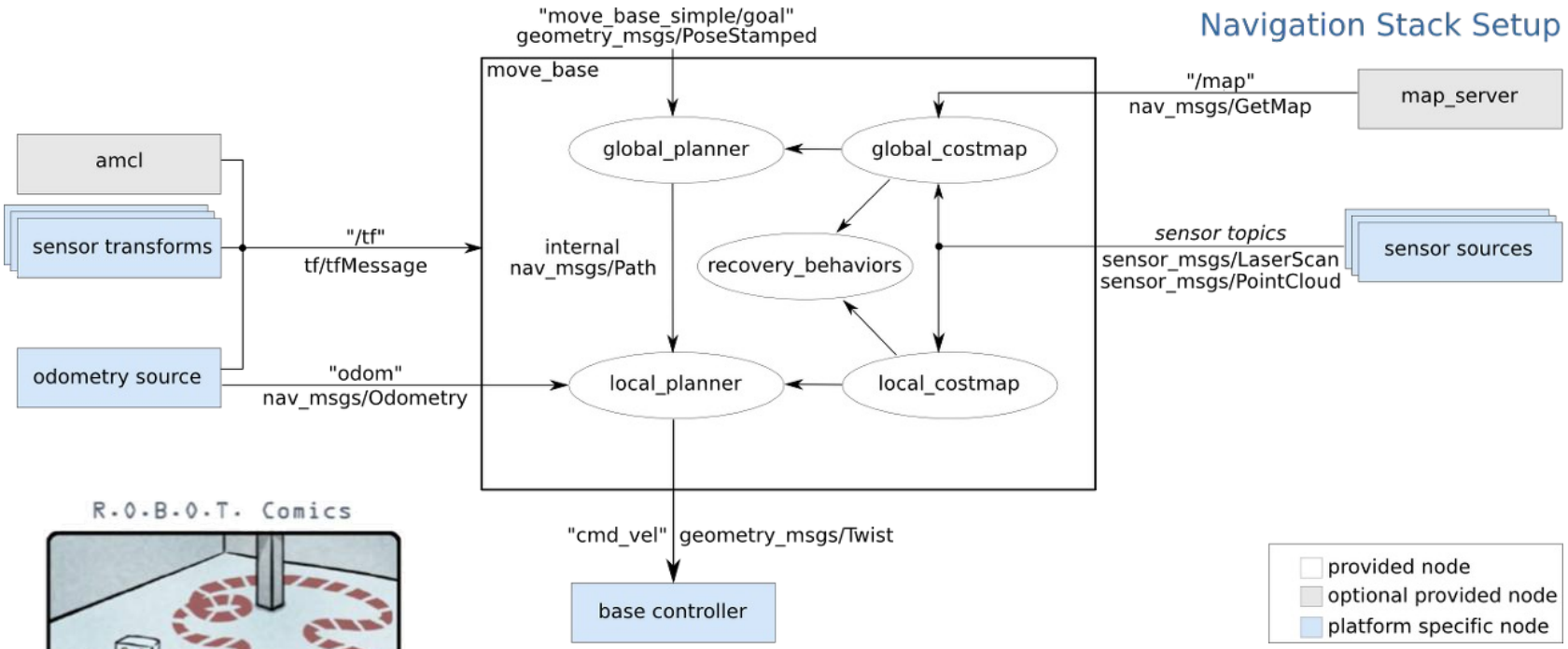
hector_slam (SLAM package) and move_base (Path planner) are public code
- hector_slam package was downloaded and built by catkin_make
- move_base was initially installed with ROS

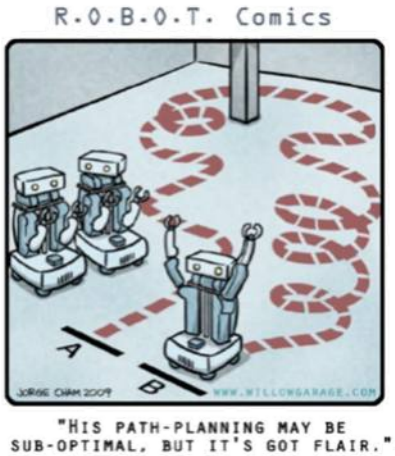
SLAM informs the location of the vehicle and map
Path planner makes the vehicle move to desired position without collision

Find frontier decides to where to go (written by 2018 TA. Dongha Chung)
- This algorithm finds the largest undiscovered region (frontier) and publishes desired waypoint message (/move_base_simple/goal) to move_base node.
- Then move_base will command to robot (/cmd_vel) to move to destination while avoiding collision to obstacles.

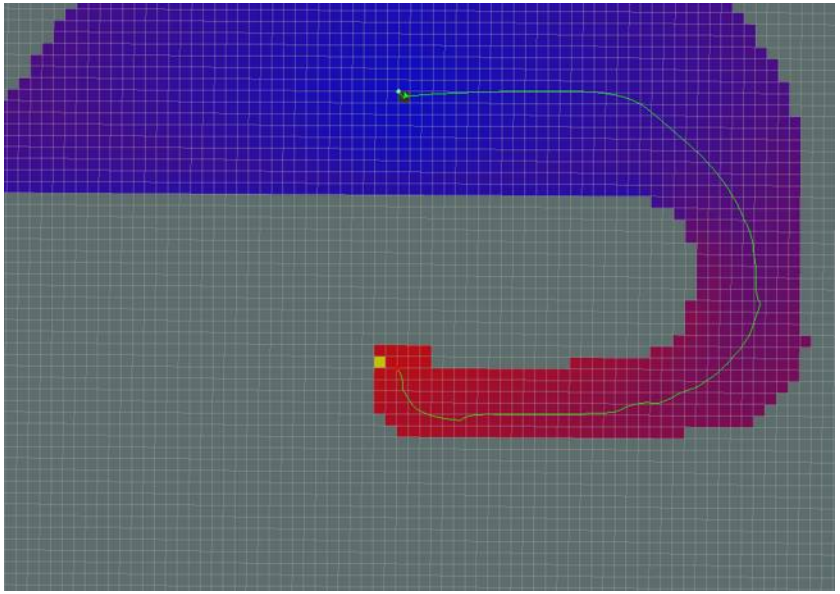

Lidar
Hector SLAM
- Robot position
- Occupancy Map
Find Frontier
Goal point
Actuator
Command
Path Planning
- Robot position
- Occupancy Map

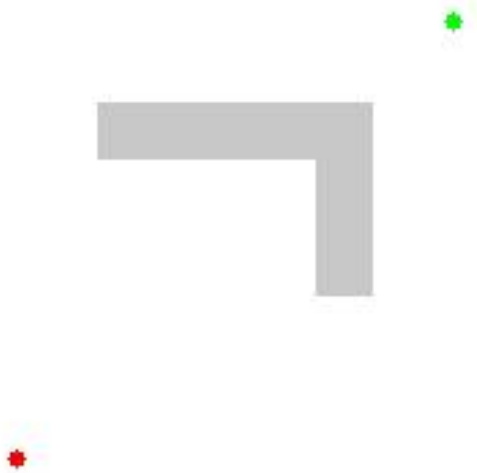
Move Base


Navigation Stack Setup
"move_base_simple/goal"
geometry_msgs/PoseStamped
move_base
"/map"
nav_msgs/GetMap
map_server
amcl
global_planner
global_costmap
sensor transforms
"/tf"
tf/tfMessage
internal
nav_msgs/Path
recovery_behaviors
sensor topics
sensor_msgs/LaserScan
sensor_msgs/PointCloud
sensor sources
odometry source
"odom"
nav_msgs/Odometry
local_planner
local_costmap
"cmd_vel" geometry_msgs/Twist
base controller
provided node
optional provided node
platform specific node

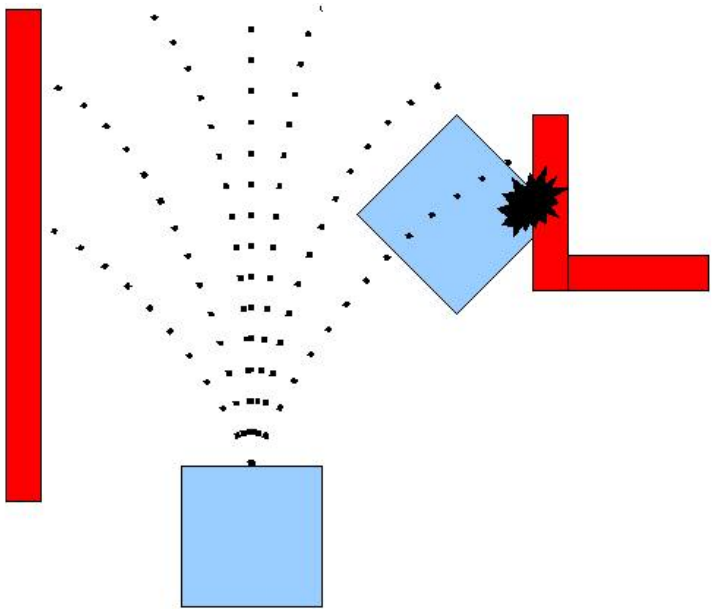

R.O.B.O.T. Comics
"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Global planner : Dijkstra Algorithm





Local planner : DWA approach



Customize

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>
```

Thank you!