

SLAM

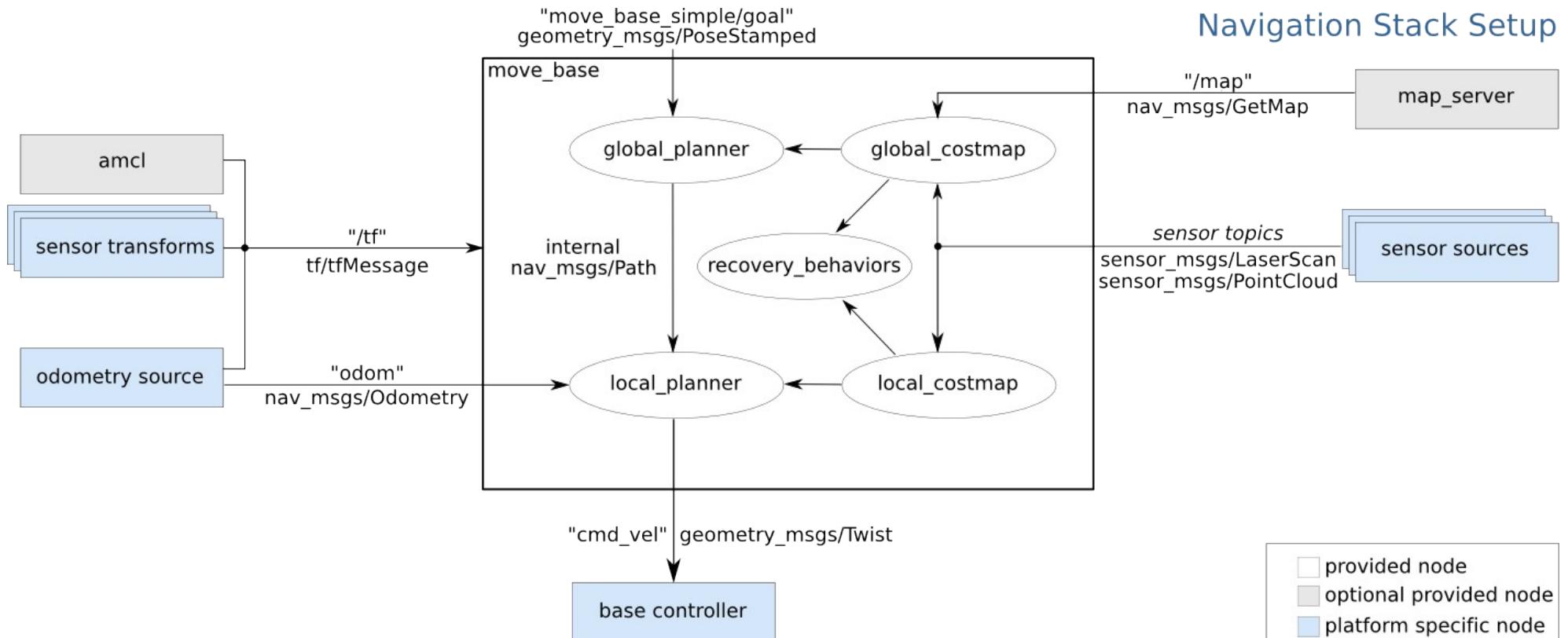
Presenter: Pyae Soan Aung

Company: ROM Robotics

Location: Yangon

Simultaneous Localization And Mapping

How works?



Simultaneous Localization And Mapping

Outline

- Gmapping [gmapping]
- Localization [amcl]
- Path Planning [move_base]

How to install ?

```
~$ sudo apt install ros-version-slam-gmapping  
~$ sudo apt install ros-version-map-server  
~$ sudo apt install ros-version-amcl  
~$ sudo apt install ros-version-move-base  
~$ sudo apt install ros-version-navigation
```

OR

<https://github.com/ros-planning/navigation/tree/noetic-devel>

Replace with
slam-toolbox

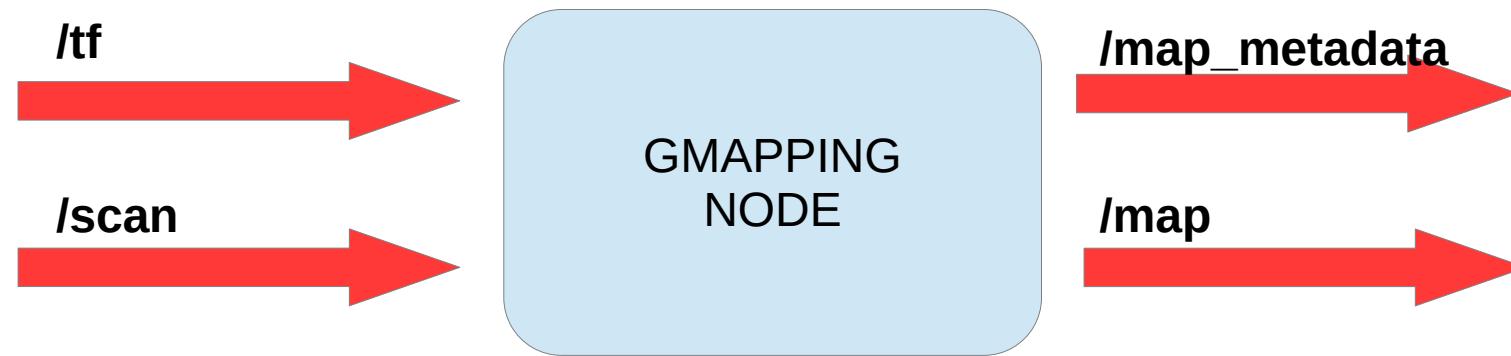
Gmapping

Outline

- **Creating map**
- **Saving map**
- **Running map**
- **How map server work**
- **Parameter**

Gmapping

Gmapping Node



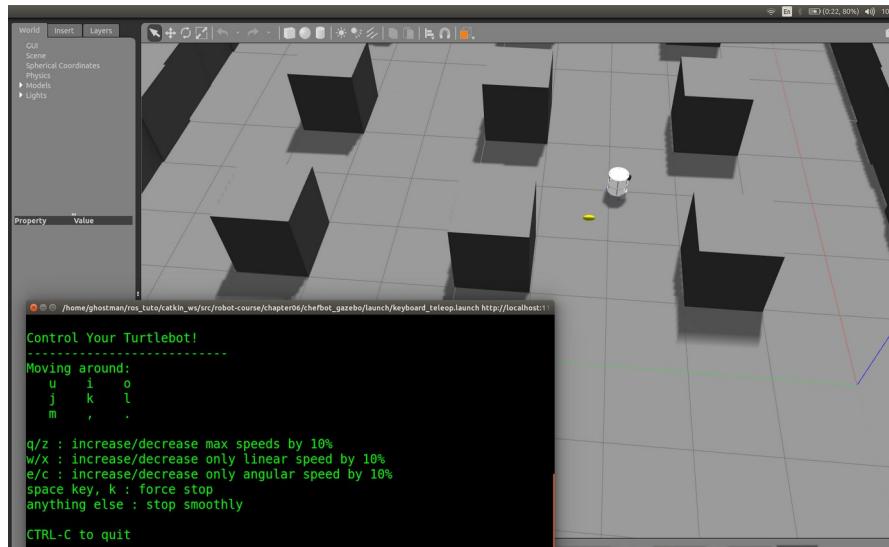
Parameter အများအပြားရှိပြီး base_frame,odom_frame,map_frame စစ်ဆေးပါ။
max_range က lidar ခဲ့ maximum range ထက်များဖို့လိုပြီး max_urange (ကတေသာ့) lidar ခဲ့
Maximum range ထက်နဲ့သင့်ပါတယ်။

Creating map

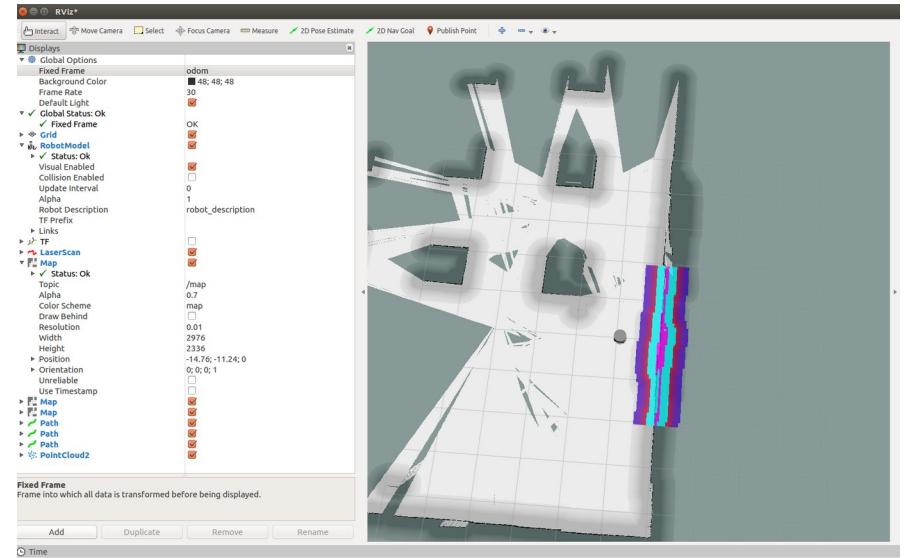
```
~$ roslaunch chefbot_gazebo chefbot_playground.launch  
~$ roslaunch chefbot_gazebo gmapping_demo.launch  
~$ roslaunch chefbot_gazebo amcl_demo.launch  
~$ roslaunch chefbot_gazebo keyboard_teleop.launch  
~$ rosrun rviz rviz
```

Rviz ပွင့်လာရင် map သစ္နဲ့ laser scan တဲ့ add ပြီး topic ရွေးပေးပါ။ ပြီးရင် simulation robot ကို Keyboard နဲ့လှည့်ပါတ်မောင်းပြီး (များများမောင်းဖို့လိုပါတယ်) map data များရလာအောင်လုပ်ပါ။

Creating map



Real Robot (or) Simulation



Visualizing data

Saving a map

Robot ကို မောင်းပြီး ရလာတဲ့ map
ကို save ဖို့

```
~$ rosrun map_server map_saver -f mymap
```

```
~$ ls
```

```
~$ mymap.pgm mymap.yaml
```

မြေပုံ ရပြုမိုလို နောက်တခါဆိုရင် Gmapping ကို
မလိုတော့ပါဘူး။ Rviz ထဲက setting တွေကို
လည်း save ထားသင့်ပါတယ်။



Using a map

```
~$ roslaunch chefbot_gazebo chefbot_playground.launch
```

```
~$ roslaunch chefbot_gazebo amcl_demo.launch
```

```
~$ roslaunch chefbot_gazebo keyboard_teleop.launch
```

```
~$ rosrun map_server map_server mymap.yaml
```

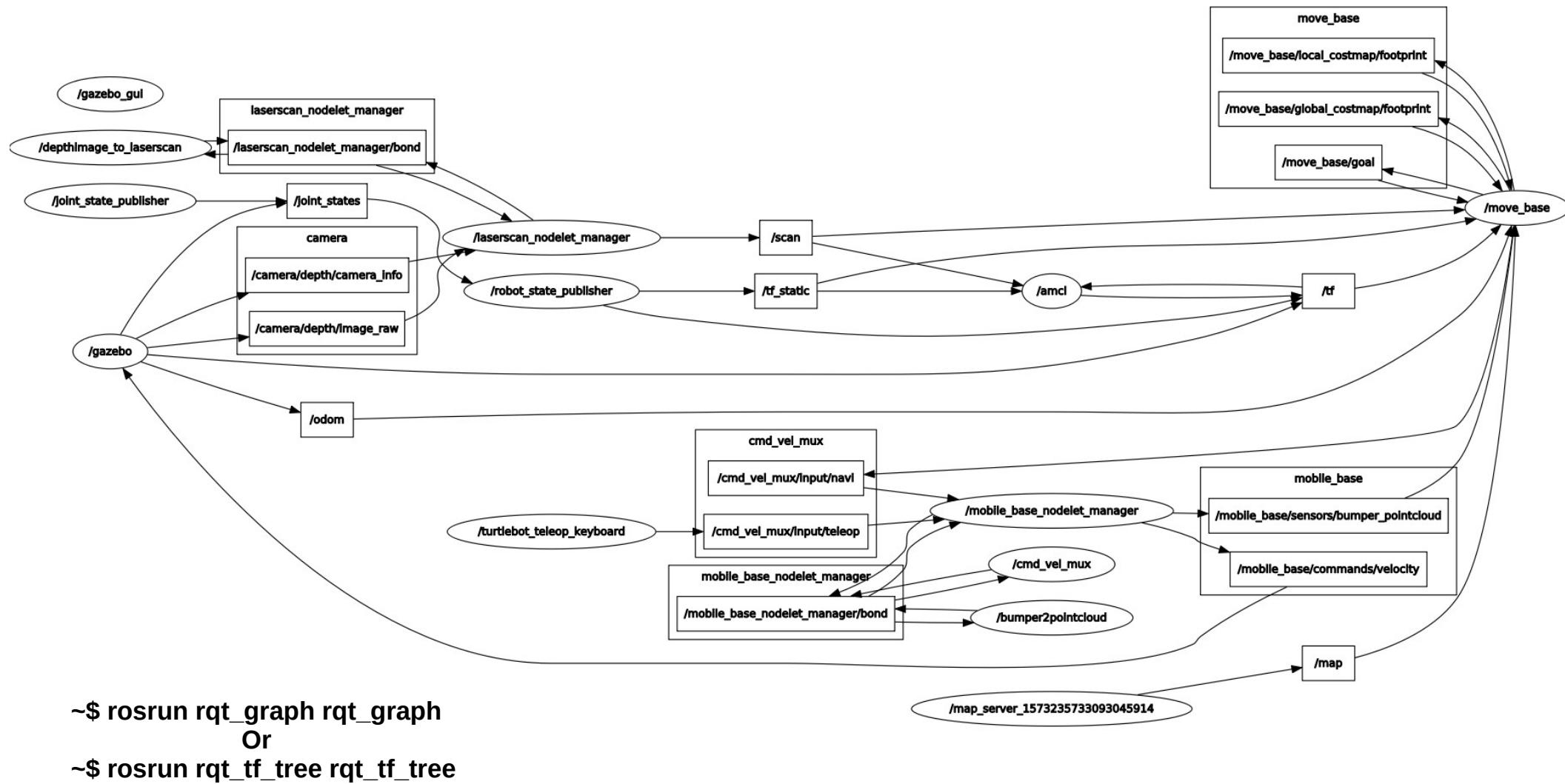


Using a map

- map server သည် static map ကို service server နဲ့ရော topic နဲ့ရောထုတ်ပေးပါတယ်။
- မြေပံ့ကောင်းကောင်းရဖို့ဆိုရင် /scan နဲ့ /tf ကောင်းဖို့လိုပါတယ်။
- သူလိုတဲ့ transform ၂ခုက base_link → lidar (or camera) ရယ် odom → base_link ရယ်ပါ။
- ဖြစ်သင့်တဲ့ frame တွေက map → odom → base_link ပါ။
- gmapping က map → odom tf ထုတ်ပေးပေမဲ့ map_server ကတော့ tf ထုတ်မပေးပါဘူး။
- ဒီအဆင့်တွေမှာ tf frame တွေကို စစ်ဆေးဖို့လိုပါမယ်။
- map_server သို့မဟုတ် gmappingng ဆိုက move_base,rviz,amcl တိုက မြေပံ့ရယူပါတယ်။

ခု သင်ခန်းစာက Simulation ထဲမှာ lidar တပ်ထားတာမဟုတ်ပဲ 3d camera တပ်ထားတာဖြစ်ပါတယ်။ သဆီက ထွေက်လာတဲ့ depth image ကို depthimage_to_laserscan ဆိုတဲ့ package/node နဲ့ laser data ပြောင်းလို့ ရပါတယ်။

ဒီနည်းနဲ့ lidar မရှိဘဲ 3d camera ရှိသူများ SLAM ကို အသုံးပြန်ပါတယ်။

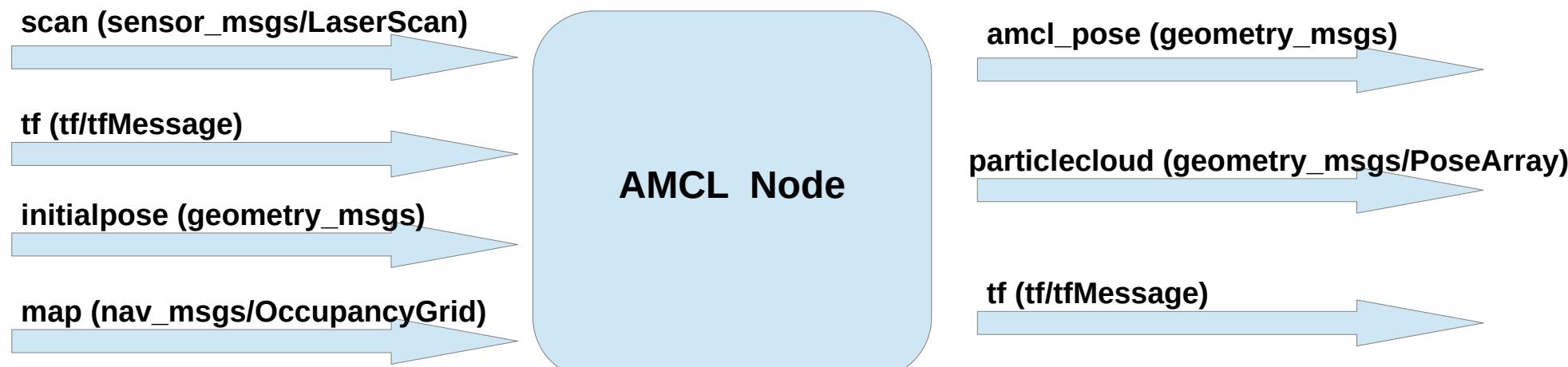


```
~$ rosrun rqt_graph rqt_graph
Or
~$ rosrun rqt_tf_tree rqt_tf_tree
```

Localization with AMCL

Adaptive Monte Carlo Localization

- Robot ဟာ မြေပံ့ပေါ်မှာ သူ၏ position နဲ့ orientation ကို သိဖို့ own position ကို ရှာပါတယ်။ ဒါမှသာ Path planning ကိုလုပ်နိုင်မှာဖြစ်ပါတယ်။



AMCL

What is Monte Carlo?

AMCL

```
~$ roslaunch chefbot_gazebo chefbot_playground.launch  
~$ roslaunch chefbot_gazebo amcl_demo.launch  
~$ roslaunch chefbot_gazebo keyboard_teleop.launch  
~$ rosrun rviz rviz
```

AMCL

amcl_demo.launch

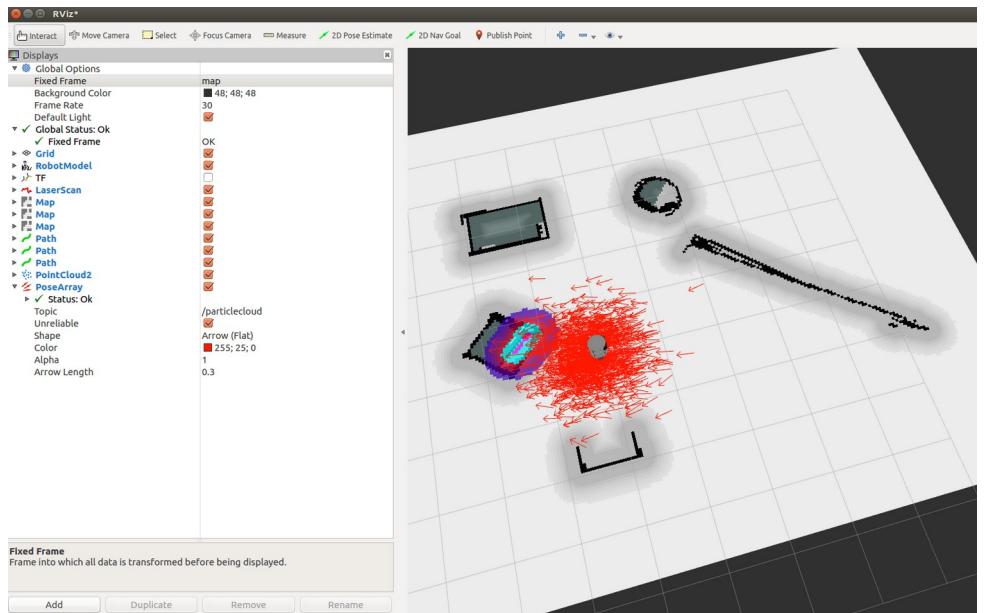
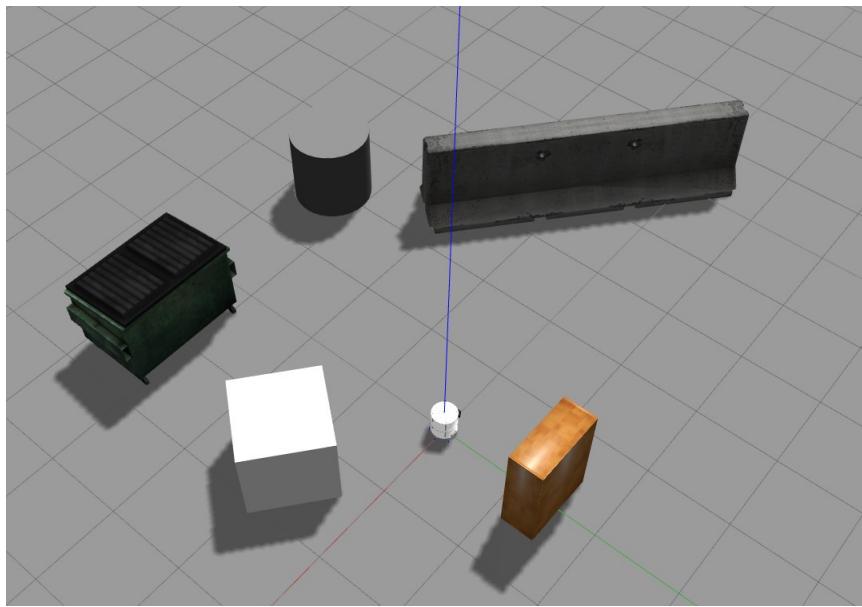
```
<launch>
    <!-- Map server -->
    <arg name="map_file" default="$(find chefbot_gazebo)/maps/playground.yaml"/>

    <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

    <!-- Localization -->
    <arg name="initial_pose_x" default="0.0"/>
    <arg name="initial_pose_y" default="0.0"/>
    <arg name="initial_pose_a" default="0.0"/>
    <include file="$(find chefbot_bringup)/launch/includes/amcl.launch.xml">
        <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>
        <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>
        <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>
    </include>

    <!-- Move base -->
    <include file="$(find chefbot_bringup)/launch/includes/move_base.launch.xml"/>
</launch>
```

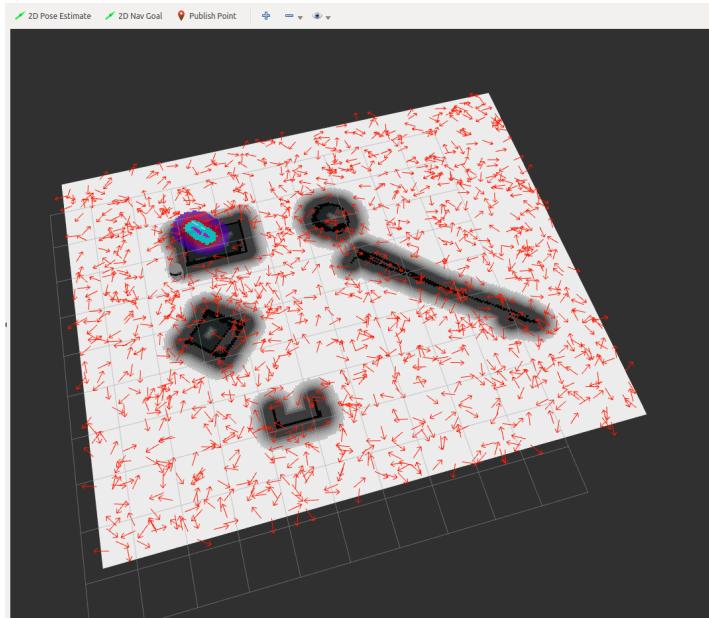
AMCL



Rviz – add → PoseArray → topic → particleCloud

AMCL

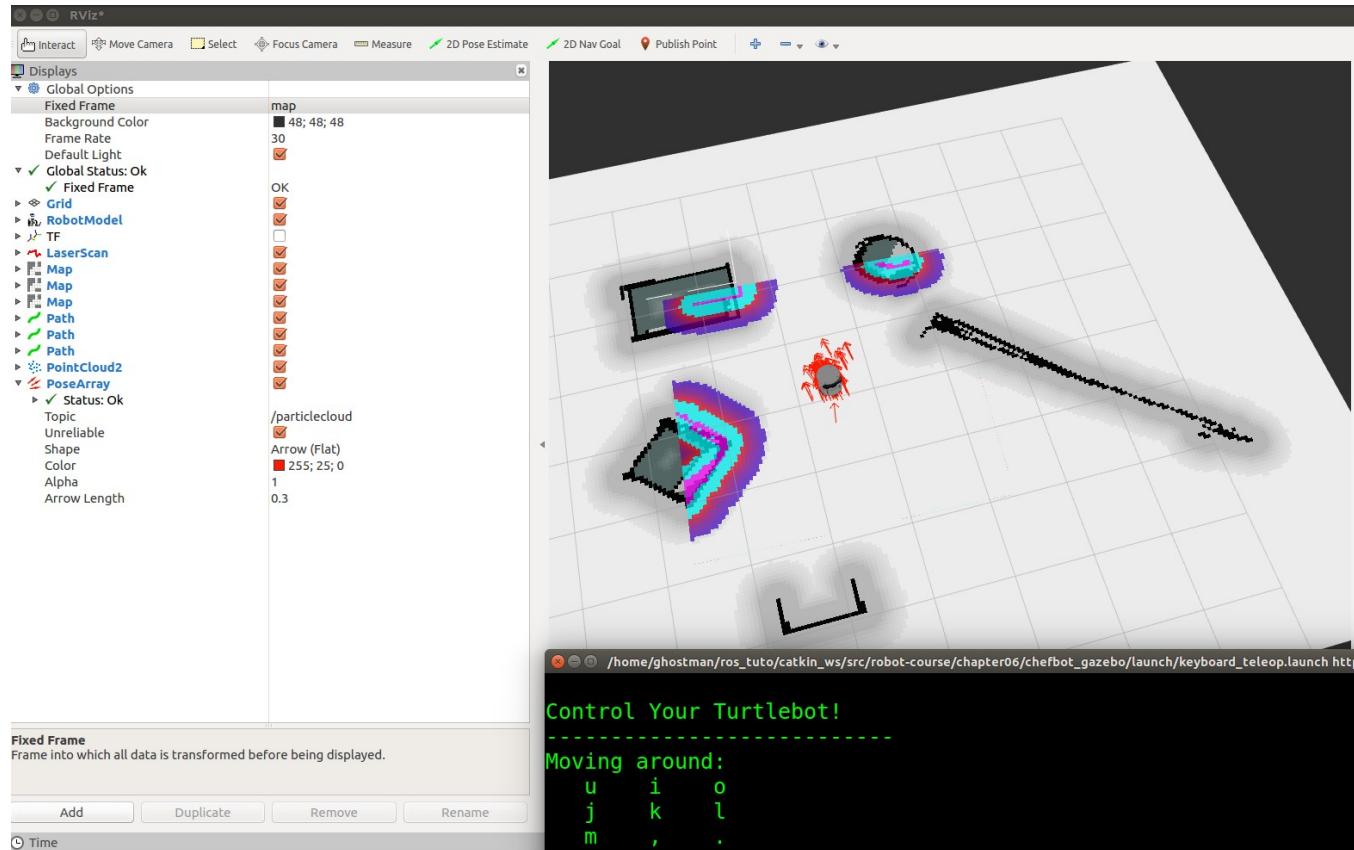
Reset Localization



`~$ rosservice call /global_localization "{}"`

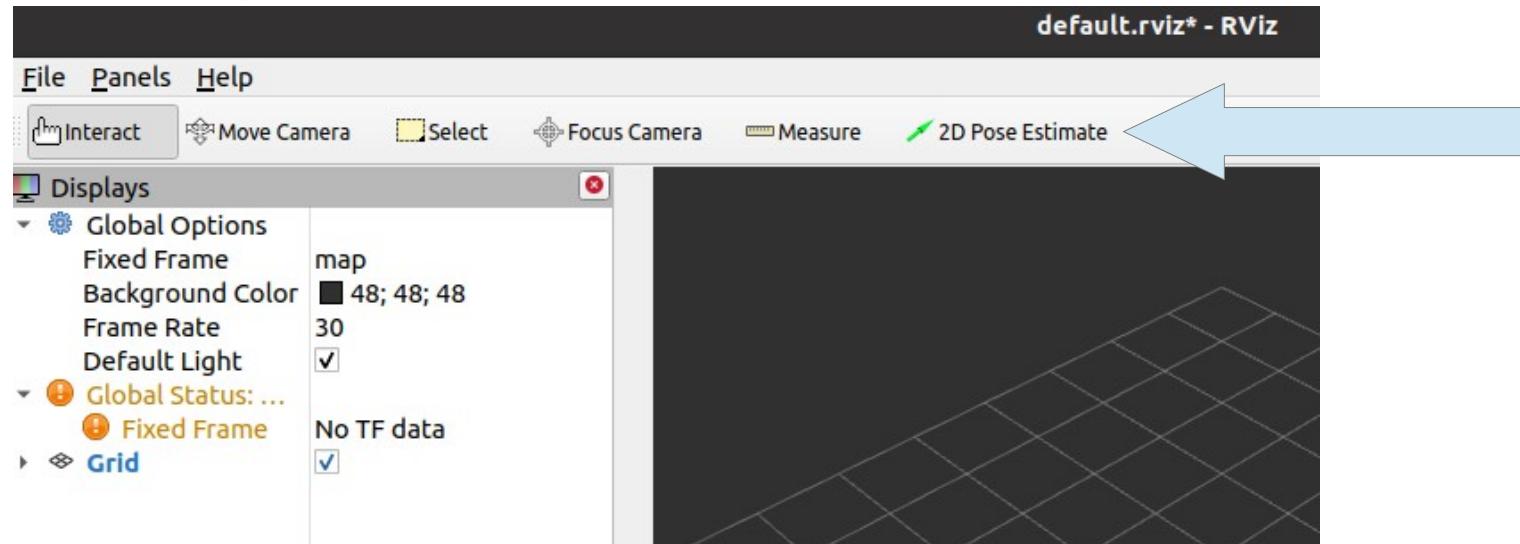
AMCL

After driving with keyboard



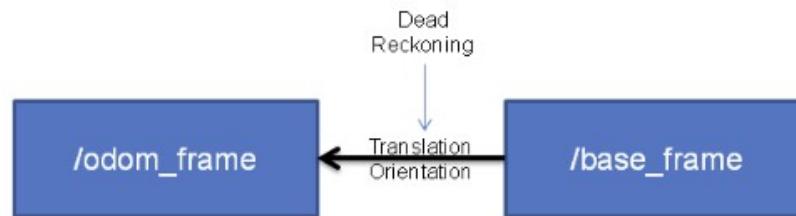
AMCL

တကယ်လို **robot** ကို မြေပံပေါ်မှာ နေရာချတာ **position** သို့မဟုတ် **orientation** လွှဲနေတယ်ဆိုရင် **rviz** မှာ **2D Pose Estimate** ကို နှုပ်ပြီး **robot** ရဲ့ **own position** နေရာချပေးနိုင်သလို **mouse** ကို **drag** လုပ်ပြီး **orientation** နေရာချပေးနိုင်ပါတယ်။
ပြီးမှ **robot** ကို မောင်းနှင်ပေးပြီး **localization** ကို ပိုမိုတိကျအောင် လုပ်ရမှာပါ။
များများမောင်းလေ ပိုမိုတိကျလေလေဖြစ်ပါတယ်။



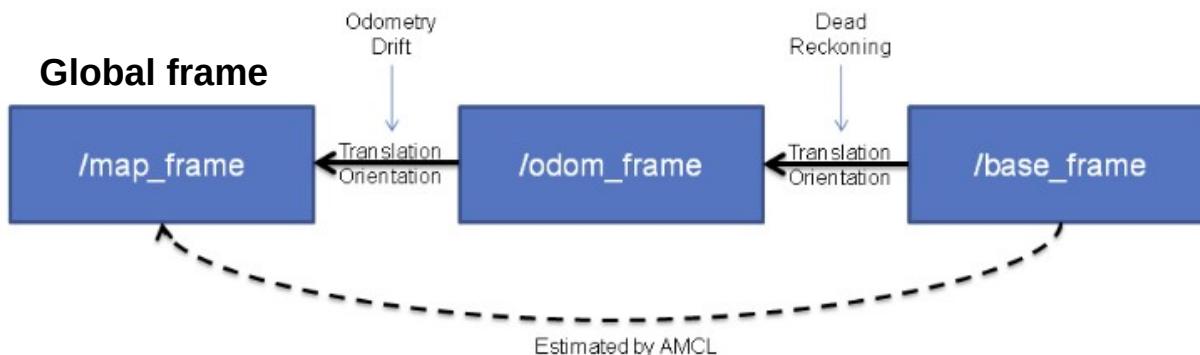
AMCL

Odometry Localization



Odom → base_link
ကို Odometry မှထွတ်လုပ်

AMCL Map Localization



map → base_link
ကို AMCL မှထွတ်လုပ်

AMCL

Parameter များစွာပါဝင်ပြီးတော့ အောက်ပါ parameter များကို စစ်ဆေးသင့်ပါတယ်။ roswiki မှာလဲ သွားရောက်ဖတ်ကြည့်သင့်ပါတယ်။

```
<param name="odom_model_type" value="diff"/>  
  
<param name="odom_frame_id" value="odom"/>  
  
<param name="base_frame_id" value="base_footprint"/>  
  
<remap from="scan" to="$(arg scan_topic)"/>  
  
<param name="use_map_topic" value="$(arg use_map_topic)"/>
```

Move Base

ခုသင်ခန်းစာမျာတော့ Navigation stack မှာ path planning လုပ်ပေးတဲ့ move_base node အကြောင်းပဲဖြစ်ပါတယ်။

```
~$ roslaunch chefbot_gazebo chefbot_playground.launch
```

```
~$ roslaunch chefbot_gazebo amcl_demo.launch
```

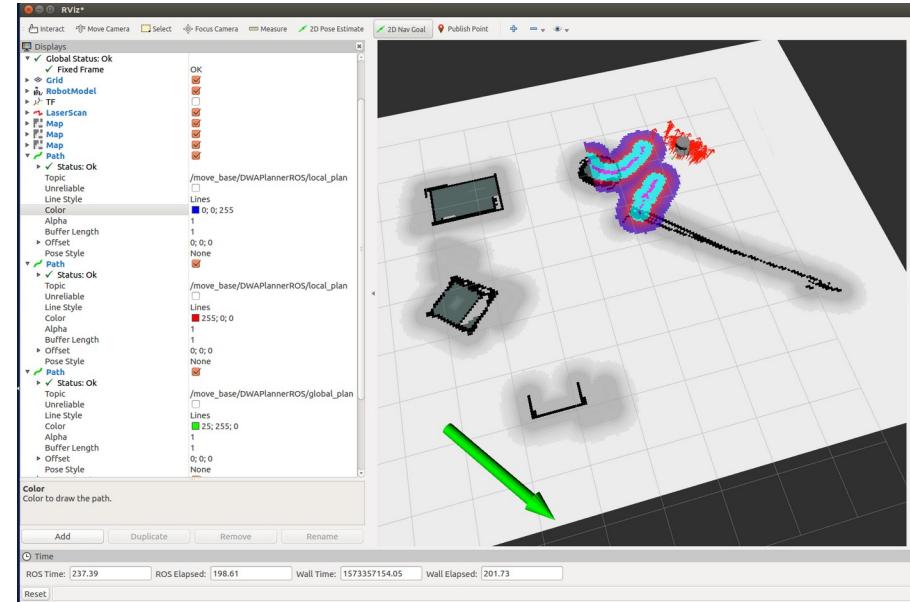
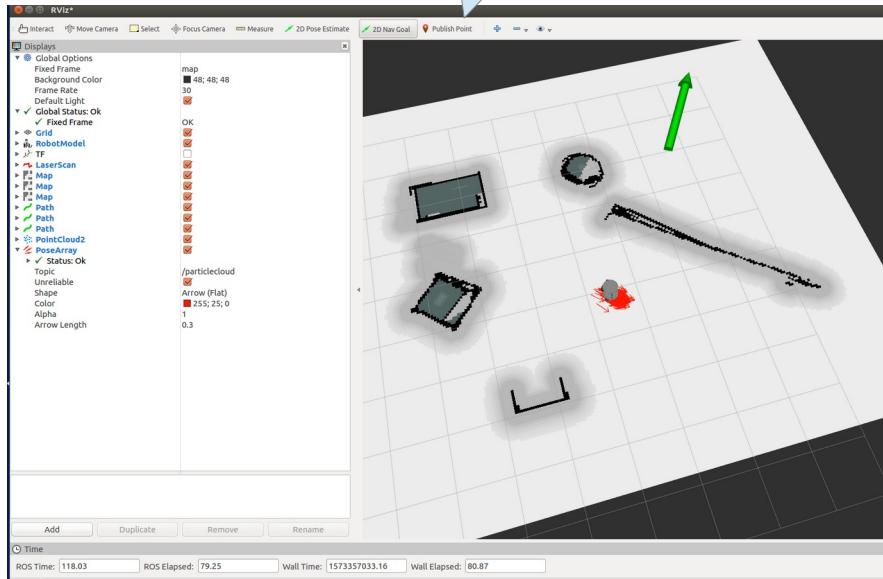
```
~$ rosrun rviz rviz
```

chefbot_gazebo/rviz/map.rviz

Rviz → map → topic → global_costmap
→ map → topic → local_costmap
→ path → topic → move_base/NavfnROS/plan (or) global_plan
→ path → topic → local_plan

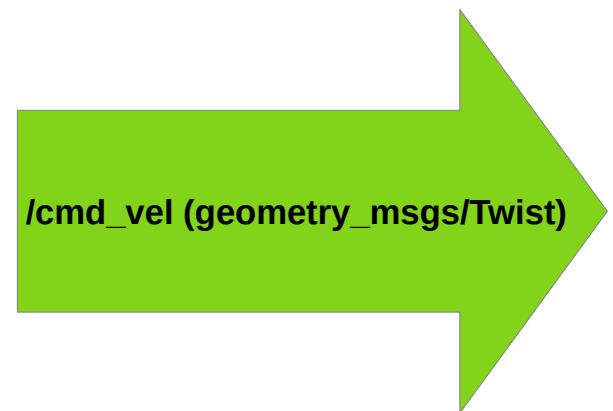
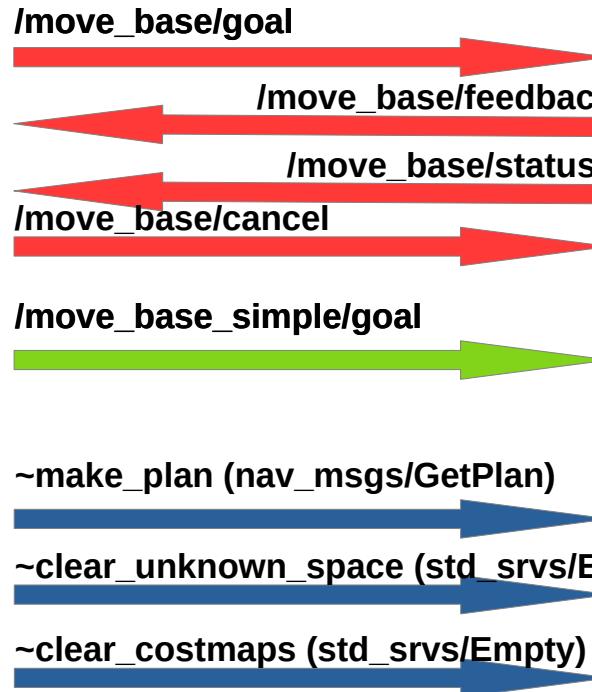
Move Base

2D Pose Goal



Move Base

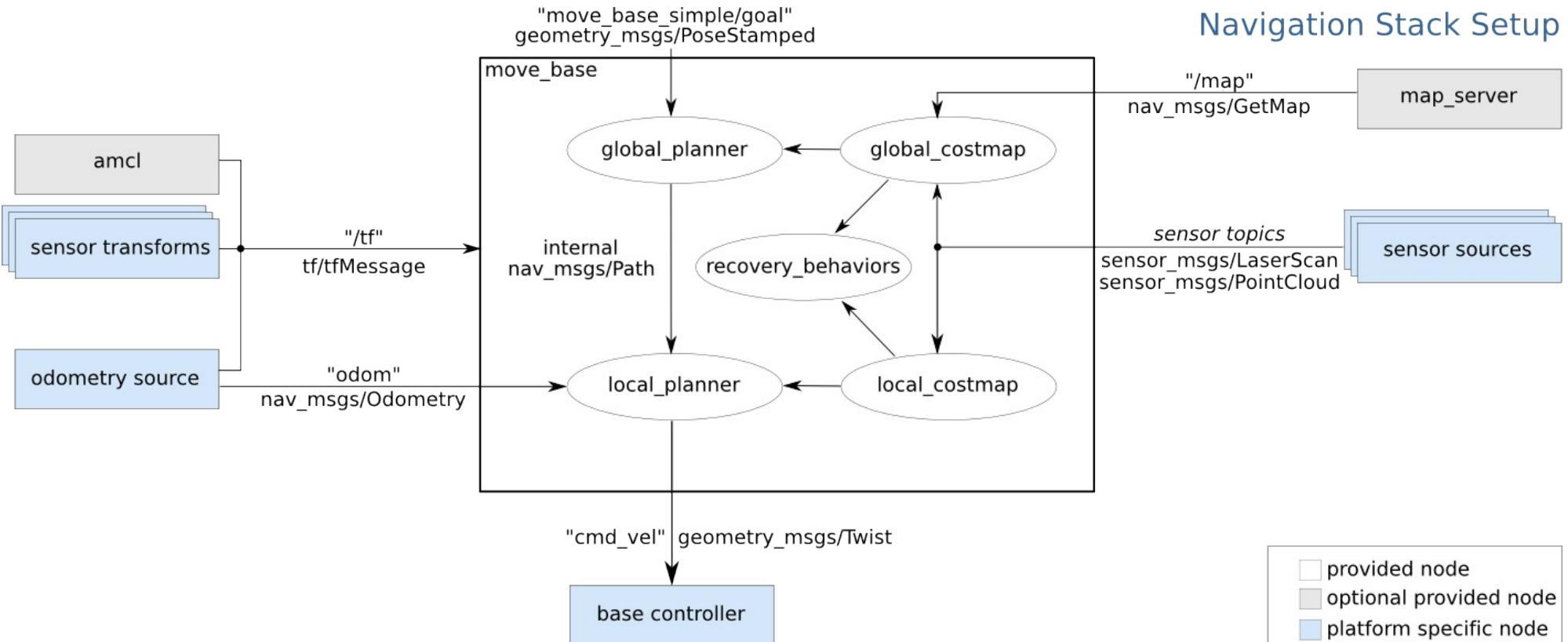
How move_base works?



- action
- topic
- service

Move Base

How move_base works?

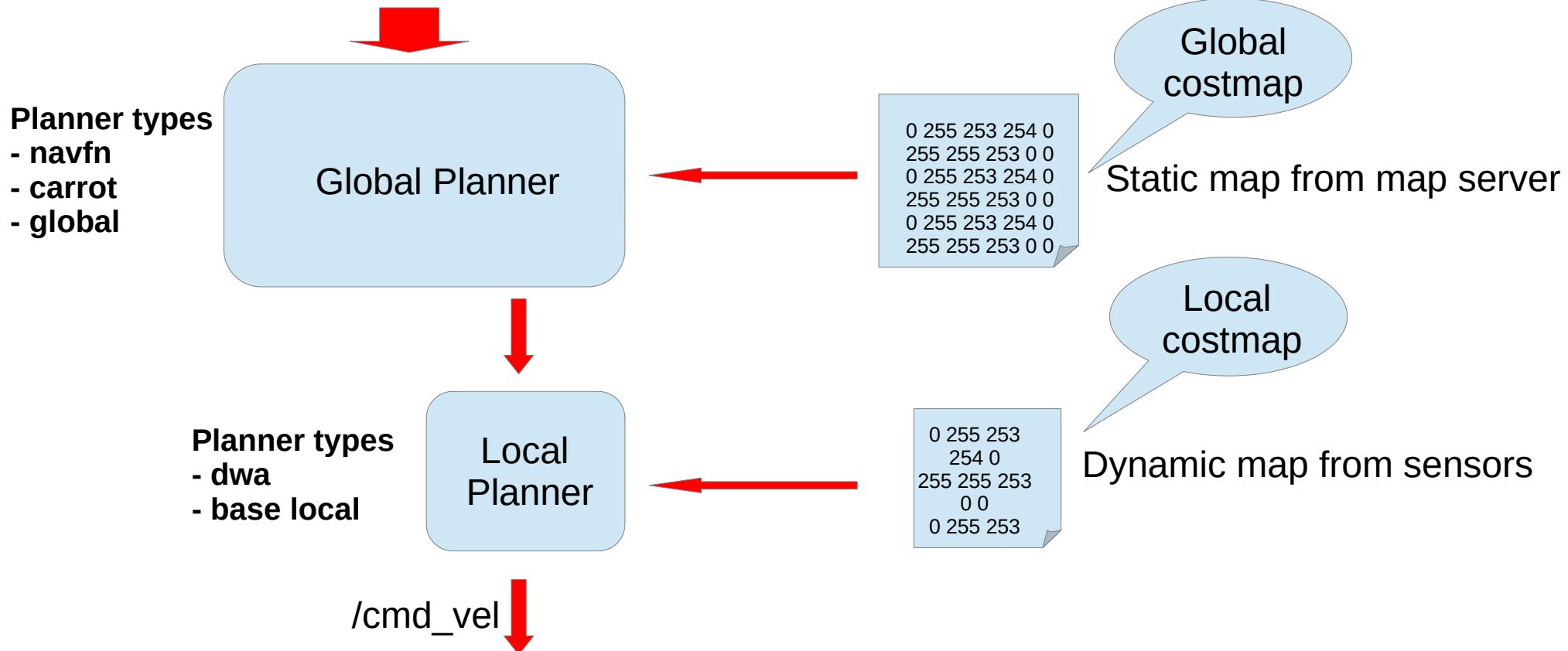


Move Base

How move_base works?

- လက်ရှိ position မှသည် goal ကိုသားဖို့ velocity command တွက်ထုတ်
- goal ပေးဖို့ action goal ကိုအသုံးပြန်တယ်။
- ၁ မီတာ ပတ်လည် Goal ရေးကြည့်ပါ။
- ~\$ rostopic echo /move_base/goal
- velocity command ပေးတဲ့ frequency သည် controller_frequency
- move_base ဟာ velocity တွေတွက်လာဖို့ path တွေအရင်တွက်ရပါတယ်။
- path တွက်တဲ့ ကောင်ကို planner လိုခေါ်ပြီး global planner နဲ့ local planner နှစ်မျိုးရှိတယ်။
- move_base ဟာ goal တစ်ခုကို ရပြီဆိုတာနဲ့ global planner ဆိုကို ပိုလိုက်ပါတယ်။

Move Base



250 = no info, 254 = obstacle, 253 = no ob but collision

Move Base

How planner works?

- global planner သည် goal position ကိုရရှိတဲ့အခါ global costmap ကိုအသုံးပြု၍ global Path တစ်ခုထုတ်လုပ်ပါတယ်၊ ပြီးတာနဲ့ path ကို local planner ဆီပိုပါတယ်။
- local planner ဟာရရှိလာတဲ့ path ကိုအစိတ်အပိုင်းအသေးလေးတွေခဲ့ထုတ်ပြီး local path ထုတ်လုပ်ပါတယ်။ အဲဒီ local path အသေးလေးတွေအတိုင်း velocity ကိုတွက်ထုတ်ပါတယ်။ local path ထုတ်တဲ့အခါမှ laser data ကိုပါရယူပြီးတော့ အထိအခိုက်မရှိတဲ့ local path ကိုတွက်ထုတ်ထာဖြစ်ပါတယ်။
- robot ရွှေလျားသွားတာနဲ့အမျှ local plan ကိုထပ်တလဲလဲပြန်လည်တွက်ချက်ပါတယ်။
- တကယ်တော့ local planner တို့global planner တို့ဆုံးတာ move base node code ထဲက Logic တွေပဲဖြစ်ပါတယ်။

Move Base

Clear costmap

- တကယ်တမ်း real hardware robot ကို အသုံးပြုတဲ့အခါ robot ဟာ နေရာမှာ 360 degree အဆက် မပြတ်လည်ပါတ်နေတာမျိုးဖြစ်နိုင်ပါတယ်။ ဒါဟာ move_base node ရဲ့ recovery behaviour ဖြစ်ပါတယ်။
- recovery ဖြစ်ခြင်းအကြောင်းအရင်းတရာ့၍ robot ဟာ goal ကို မရောက်နိုင်တော့ခြင်းနဲ့ costmap များ ရှုပ်ထွေးလာခြင်းကြောင့် ဖြစ်ပါတယ်။ robot ဟာ သူ့ရဲ့ orientation ကို ထိန်းသိမ်းဖို့ recovery လုပ်ရခြင်းဖြစ်တတ်ပါတယ်။
- recovery နှစ်မျိုးရှိပြီး rotate recovery နဲ့ clear costmap recovery ပါ။ clear costmap လုပ်ဖို့

```
~$ rosservice call /move_base/clear_costmaps "{}"
```

Launch

Move Base

```
<launch>

  <arg name="odom_topic" default="odom" />

  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find chefbot_bringup)/param/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find chefbot_bringup)/param/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find chefbot_bringup)/param/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find chefbot_bringup)/param/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find chefbot_bringup)/param/base_local_planner_params.yaml" command="load" />
    <rosparam file="$(find chefbot_bringup)/param/dwa_local_planner_params.yaml" command="load" />
    <rosparam file="$(find chefbot_bringup)/param/move_base_params.yaml" command="load" />

    <remap from="cmd_vel" to="/cmd_vel_mux/input/navi"/>
    <remap from="odom" to="$(arg odom_topic)"/>
  </node>
</launch>
```

Move Base

Parameter များ (base_local_planner_params.yaml)

- controller_frequency:3.0 – controller_frequency ကိုတော့ တစ်စက်နံ့မှ Planning Process ကို ဘယ်နှစ်ကိုမြဲ Update လုပ်မလဲဆိုတာပဲ ဖြစ်တာပါ။ ဒီတန်ဖိုး အရမ်းများသွားခြင်းက CPU ကို Overload ဖြစ် စေနိုင်တဲ့အတွက် မိမိကျွန်ုပါ၍ CPU နဲ့ ကိုက်ညီ အောင် ထားပေးဖို့လိုအပ်ပါတယ်။ သာမန် Laptop တွေအတွက်တော့ ဒီတန်ဖိုးကို ၃ ကနေ ၅ ကြားထဲမှာပဲ ထားသင့်ပါတယ်။
- max_vel_x:0.3 – Robot ရဲ့ အများဆုံး Linear Velocity ဖြစ်ပါတယ်။ သူရဲ့ Unit က m/s ဖြစ်ပါတယ်။ 0.3 ဆိုတော့ 30 cm/s ဖြစ် မာပေါ့။ Indoor Robot တွေအတွက်တော့ ဒီ Speed က အသင့်လျှော်ဆုံးဖြစ်ပါလိမ်မယ်။ Chapter 9 မှာတုန်းကတော့ ကျွန်ုပ်တော်က 0.5 ထားခဲ့ပါတယ်။ ကွန်ပြုတာ Simulation ထဲမှာပဲ Run မှာဖြစ်လဲ 0.5 သတ်မှတ်ပေးခဲ့တာဖြစ်ပါတယ်။
- min_vel_x:0.5 – Robot ရဲ့ အနည်းဆုံး Linear Velocity ဖြစ်ပါတယ်။
- max_vel_theta:1.0 – Robot ရဲ့ Rotational Velocity ပဲဖြစ်ပါတယ်။ ဒီတန်ဖိုးကို အများကြီး သတ်မှတ်ပေးတာကလည်း Goal Orientation ကနေ လဲချော်သားစေနိုင်တဲ့အတွက် Robot Size နဲ့ ကိုက်ညီတဲ့ပေါက်ကို သတ်မှတ်ပေးတာ အကောင်းဆုံးဖြစ်ပါတယ်။ သူရဲ့ Unit က radian/s ဖြစ်ပါတယ်။
- min_vel_theta:-1.0 – Root ရဲ့ Minimum Rotational Velocity ဖြစ်ပါတယ်။
- min_in_place_vel_theta: 0.5 – Robot ရဲ့ Minimum In_place Rotational Velocity ဖြစ်ပါတယ်။
- escape_vel: 0.1 – ကျွန်ုပ်ကျော်းကျုပ်ကျုပ် နေရာတွေထဲ ရောက်သွားတဲ့အခါမှာ Robot ရဲ့ Velocity ဖြစ်ပါတယ်။ သူရဲ့တန်ဖိုးက အနတ် တန်ဖိုးဖြစ်နေပြီး Robot ကုန်နောက်ဆုတ်ဖူး ရည်ရွယ်ထားတာဖြစ်ပါတယ်။ နေရာကျော်းတဲ့နေရာမှာ နောကဆုတ်လိုက် ရွှေတုံးလုံကနဲ့ မောင်းရန် အတွက်ဖြစ်ပါတယ်။

- acc_lim_x: 0.8 – Max Linear Acceleration ဖြစ်ပြီး သူ့ရဲ့ Unit ကတေသာ meter per second square ဖြစ်ပါတယ်။
- acc_lim_y: 0.0 – Y Axis အတိုင်း Max Linear Accelerating ဖြစ်ပြီး ဒီမှာတော့ 0 လိုပေးထားပါတယ်။ ကျွန်တော်တို့၏ Robot က Differential Drive မှုဖြစ်ပါတယ်။ တွေး Y Direction အတိုင်းသွားနှင့်တဲ့ Omni Wheel နဲ့ Robot တွေမှာတော့ ဒီတန်ဖိုးကို သတ်မှတ်ပေးဖို့ လိုအပ်ပါလိမ့်မယ်။
- Holonomic_robot:False – Omni-directional Drive Robot တွေအတွက်ဖြစ်ပါတယ်။ တွေးအမျိုးအစားဆိုရင်တော့ False
- xy_goal_tolerance:0.1 – Target နဲ့ Robot လက်ရှိနေရာကြားက Distance ဖြစ်ပါတယ်။ ဒီတန်ဖိုးက တော်တော်လေးယ်နေရင်တော့ သင့်ရဲ့ Robot က ဒီတန်ဖိုးမရောက်ရောက်အောင် ညိုနေရတဲ့ အတွက် Target ကိုရောက်နေရင်တောင် တဲ့လည်လည် နဲ့ ရပ်တော့မှာမဟုတ်ပါဘူး။ ဒါကြောင့် ဒီတန်ဖိုးကို ကိုယ့်ရဲ့ map_size နဲ့ ညိုပြီးထားဖို့ လိုအပ်ပါတယ်။
- pdist_scale:0.8 – Global Path ပေါ်မှာပဲ မူတည်ပြီး သွားတဲ့ စကေးဖြစ်ပါတယ်။ သူကတော့ Map အတိုင်းပဲ Goal ကိုတဖြည်းဖြည်း ရေအောင် ချဉ်းကပ်ပါတယ်။ ဒီတန်ဖိုးကို gdist_scale ထက်ပုံကြီးထားရင်တော့ တွက်ထုတ်လို့ရတဲ့ Global Path အတိုင်းပဲနဲ့စပ်စပ် သွားပါမယ်။
- gdist_scale:0.4 – သူကတော့ Goal ရောက်ဖို့ကိုပဲ အဓိက ထားပါတယ်။ တွက်ထုတ်လိုက်တဲ့ Path အတိုင်းသွားချင်မှုသွားပါလိမ့်မယ်။ ဒီတန်ဖိုးကို _scale ထက်ပုံကြီးတာထားလိုက်ရင် လိုချင်တဲ့ Path အတိုင်းသွားတော့မှာမဟုတ်ပါဘူး။
- occdist-scale:0.1-Obstacle တွေကို ရှောင်ဖို့ ရှားဖို့ Scale ဖြစ်ပါတယ်။ အနီးစပ်ဆုံး ဘယ်လောက်အကွာအဝေးကနေ ရှောင်မလဲဆိုတာ သတ်မှတ်ပေးပါတယ်။ sim_time:1.0 – အချိန်သယ်လောက် ကြိုးတင်ပြီး Path ကို တွက်ထုတ်မလဲဆိုတာ ဖြစ်ပါတယ်။ သူတန်ဖိုးကိုတော့ Seconds နဲ့သတ်မှတ်ပါတယ်။
- dwa:true-Trajectories ကို ကြိုးတင်ပြီး Simulation လုပ်တဲ့အချိန်မှာ Dynamic Window Approach ကိုသုံးမသုံး

Move Base

costmap_common_params.yaml

- `robot_radius:0.165` – စက်ဝိုင်းပံ့သဏ္ဌာန်ရှိတဲ့ Robot တွေအတွက် ဖြစ်ပါတယ်။ X,Y နေရာတွေမှာ Robot Center ကနေ တိုင်းတာလို့ရတဲ့ Coordinate တန်ဖိုးကုတ္တော့ Meterနဲ့ဖော်ပြုပါတယ်။
- `Footprint: [[x0,y0],[x1,y1],[x2,y2],[x3,y3],etc]` – အခြားသော ပံ့သဏ္ဌာန်ရှိတဲ့ Robot တွေအတွက် ဖြစ်ပါတယ်။ X, Y နေရာတွေ မှာ Robot Center ကနေ တိုင်းတာလို့ရတဲ့ Coordinate တန်ဖိုးတွေကို ထည့်ပေးရမှာ ဖြစ်ပါတယ်။ ကျွန်ုတော့ အရုပ်ကတော့ စက်ဝိုင်းပံ့သဏ္ဌာန်ဖြစ်တဲ့အတွက် ဒီတန်ဖိုးတွေကို မသတ်မှတ်ပေးထားပါဘူး။
- `inflation_radius:0.3` – Obstacle ရဲ့ Inflation Radius ဖြစ်ပါတယ်။ တကယ်လို့ သင့်ရဲ့ Robot ဟာ အကျဉ်းအကျပ်ထဲမှာ သွားဖို့ ခက်တဲ့ Design ဖြစ်နေရင် ဒီတန်ဖိုးကို လျော့ပေးဖို့ လိုအပ်ပါတယ်။

Move Base

global_costmap_params.yaml

- global_frame: /map – Global Cost Map အတွက်ကိုတော့ Map Frame ကိုပဲ global_frame အဖြစ်သုံးထားပါတယ်။
- robot_base_frame: /base_footprint – ဒါကတော့ သင်ဆွဲလာတဲ့ urdf ဖိုင်မှာ သတ်မှတ်ထားသလိုပဲ ဖြစ်ပါတယ်။ Turtlebot အတွက်က /base_footprint ဖြစ်ပါတယ်။ ဥပမာ ကျွန်တော့ Robot ဆိုရင်တော့ /base_link ဖြစ်ပါတယ်။
- update_frequency: 1.0 – Global Map မှာ Sensor Information တွေကို Update လုပ်တဲ့ Frequency ဖြစ်ပါတယ်။ ဒီတန်ဖိုးကို များများ ပေးခြင်းက သင့်ကွန်ပြုတာရဲ့ CPU ကို Overload ဖြစ်စေနိုင်ပါတယ်။ သာမဏ် ကွန်ပြုတာတွေအတွက်ကိုတော့ 1 နဲ့ 5 ကြားမှာ ထားပေးတာ အကောင်းဆုံးဖြစ်ပါတယ်။
- publish_frequency: 0 – ဒါကတော့ Map အသေဖြစ်တဲ့အတွက် Update လုပ်ဖို့ မလိုအပ်ပါဘူး။
- static_map: true – Global Map ကတော့ အမြဲ Static ဖြစ်နေတဲ့အတွက် True ပဲ ပေးထားပါမယ်။
- Rolling_window: false – ကျွန်တော်တို့က Global Map ကို အရင်ထုတ်ပြီးမှ Robot ကို စေခိုင်းတာဖြစ်တဲ့အတွက် ဒီတန်ဖိုးကိုလဲ False ပဲ ပေးထားမှာ ဖြစ်ပါတယ်။
- transform_tolerance: 1.0 – Coordinate Frame awG Transform ပြုလုပ်ဖို့ရာအတွက် Delay Time ဖြစ်ပါတယ်။ Robot နဲ့ Workstation ဟာ Wireless Connection ဖြစ်မယ်ဆိုရင်တော့ ဒီတန်ဖိုးကတော့ အသင့်လျှော်ဆုံး ဖြစ်ပါတာ

Move Base

local_costmap_params.yaml

- **global_frame: /odom** – Local Costmap ကတော့ Robot သွားတာနဲ့ Sensor Data တွေအပေါ် မူတည်ပြီး ပြန်ဆွဲတဲ့ Map ဖြစ်ပါတယ်။ သူ့ကိုတော့ Robot လက်ရှိရောက်နေတဲ့ နေရာ Odometry Frame ကိုပဲ အသုံးပြုပါတယ်။
- **robot_base_frame: /base_footprint** – ဒါကတော့ သင်ဆွဲလာတဲ့ urdf ဖိုင်မှာ သတ်မှတ်ထားသလိုပဲ ဖြစ်ပါတယ်။ Turtlebot အတွက်က /base_footprint ဖြစ်ပါတယ်။ ဥပမာ ကျွန်တော့ Robot ဆိုရင်တော့ /base_link ဖြစ်ပါတယ်။
- **update_frequency: 3.0** – Sensor Data တွေနဲ့ Local Map ကို Update လုပ်တဲ့ အကြိမ်ဖြစ်ပါတယ်။ times per second နဲ့ သတ်မှတ်ပါတယ်။ ကွန်ပြုတာက နေးနေရင်တော့ ဒီတန်ဖိုးကို လျှော့ပေးဖို့ လိုအပ်ပါတယ်။
- **static_map: false** – Local Map ကို အချိန်နဲ့အမျှ Update လုပ်နေမှာ ဖြစ်တဲ့အတွက် ဒီတန်ဖိုးကို False ပဲ ပေးထားပါတယ်။
- **rolling_window: true** – Robot သွားနေသမျှ Update လုပ်နေမှာဖြစ်တဲ့အတွက် ဒီတန်ဖိုးကို True လို့ သတ်မှတ်ပေးထားပါတယ်။
- **width: 6.0** – Robot သွားနေတဲ့အချိန်မှာ Update လုပ်နေမယ့် Map ရဲ့ Width ဖြစ်ပါတယ်။ meters နဲ့ သတ်မှတ်ပေးပါတယ်။
- **height: 6.0** – Robot သွားနေတဲ့အချိန်မှာ Update လုပ်နေမယ့် Map ရဲ့ Height ဖြစ်ပါတယ်။ meters နဲ့ သတ်မှတ်ပေးပါတယ်။
- **resolution: 0.01** – Map ရဲ့ Resolution ဖြစ်ပါတယ်။ သင့် global_map ရဲ့ Resolution နဲ့ ကိုက်ညီဖို့ လိုအပ်ပါတယ်။
- **transform_tolerance: 1.0** – Coordinate Frame တွေ Transform ပြုလုပ်ဖို့ရာအတွက် Delay Time ဖြစ်ပါတယ်။ Robot နဲ့ Workstation ဟာ Wireless Connection ဖြစ်မယ်ဆိုရင်တော့ ဒီတန်ဖိုးကတော့ အသင့်လျှော်ဆုံး ဖြစ်ပါတယ်။

Move Base

More about

- Gmapping နဲ့ Amcl တူးက parameter များကို အသေးစိတ်မဖော်ပြခဲ့ပေမဲ့ move_base ကတော့ param များထာမို့ အသုံးဝင်နိုင်တဲ့ param များကိုဖော်ပြလိုက်ပါတယ်။ param များကို yaml ဖိုင်များမှာ သွားပြင်လို့ရသလို အခို့။ param များကို dynamic ပြင်လို့ရပါတယ်။

~\$ rosrun rqt_reconfigure rqt_reconfigure

- Planning algorithm များနဲ့ implement များကို သိချင်ရင်တော့ အောက်ပါ paper ကိုဖတ်သင့်ပါတယ်။
Maram Alajlan and Anis Koubaa ရဲ့ Writing global path planners Plugins in ROS:A tutorial
- Navigation stack ကို tuning လုပ်ဖို့ကတော့ အောက်ပါ paper ကို ဖတ်သင့်ပါတယ်။
<http://kaiyuzheng.me/documents/navguide.pdf>

Sending Goal example

```
#include <ros/ros.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <actionlib/client/simple_action_client.h>
#include <tf/transform_broadcaster.h>
#include <sstream>
#include <iostream>

typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> MoveBaseClient;

int main(int argc, char** argv){
    ros::init(argc, argv, "navigation_goals");

    MoveBaseClient ac("move_base", true);

    while(!ac.waitForServer(ros::Duration(5.0))){
        ROS_INFO("Waiting for the move_base action server");
    }

    move_base_msgs::MoveBaseGoal goal;

    goal.target_pose.header.frame_id = "map";
    goal.target_pose.header.stamp = ros::Time::now();
    try{
        goal.target_pose.pose.position.x = atof(argv[1]);
        goal.target_pose.pose.position.y = atof(argv[2]);
        goal.target_pose.pose.orientation.w = atof(argv[3]);
    }
```

Sending Goal example

```
catch(int e){  
  
    goal.target_pose.pose.position.x = 1.0;  
    goal.target_pose.pose.position.y = 1.0;  
    goal.target_pose.pose.orientation.w = 1.0;  
  
}  
ROS_INFO("Sending move base goal");  
ac.sendGoal(goal);  
  
ac.waitForResult();  
  
if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)  
    ROS_INFO("Robot has arrived to the goal position");  
else{  
    ROS_INFO("The base failed for some reason");  
}  
return 0;  
}
```

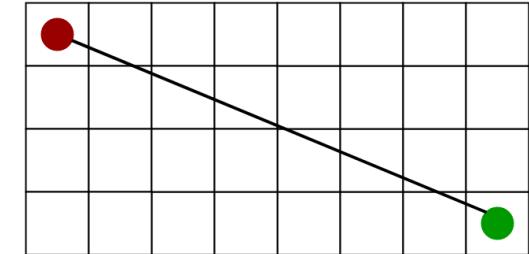
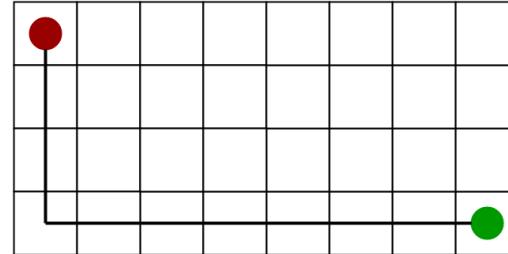
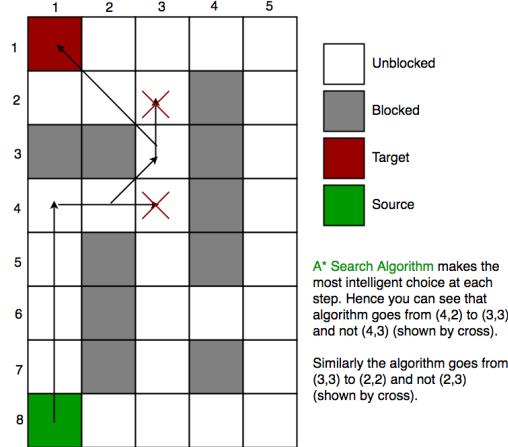
လွှဲလာစရုပ်

Presenter: Pyae Soan Aung

Company: ROM Robotics

Location: Yangon

A* algorithm example



<https://www.geeksforgeeks.org/a-search-algorithm/>

$$F = g + h$$

→ g = movement cost from start point to given point

→ h = given point to final destination

- h ကိုတွက်ထုတ်ရာမှာ အခိုင်ကုန်တာမိလို ခန့်မှန်းတဲ့နည်းလမ်းကိုသုံးရင်

c) Manhattan Distance

$$h = \text{abs}(\text{current_cell.x} - \text{goal.x}) + \text{abs}(\text{current_cell.y} - \text{goal.y})$$

j) Euclidean Distance

$$h = \sqrt{(\text{current_cell.x} - \text{goal.x})^2 + (\text{current_cell.y} - \text{goal.y})^2}$$

```

// A* Search Algorithm
1. Initialize the open list
2. Initialize the closed list
   put the starting node on the open
   list (you can leave its f at zero)
3. while the open list is not empty
   a) find the node with the least f on
      the open list, call it "q"
   b) pop q off the open list

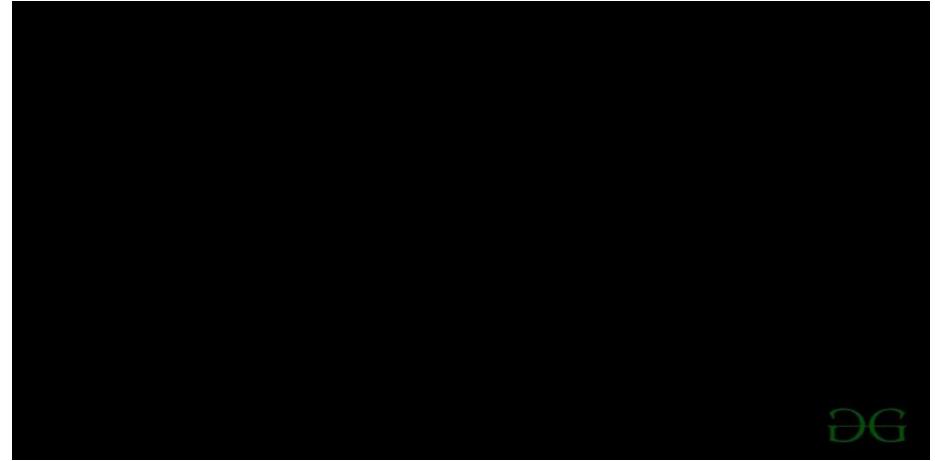
   c) generate q's 8 successors and set their
      parents to q

   d) for each successor
      i) if successor is the goal, stop search
         successor.g = q.g + distance between
                     successor and q
         successor.h = distance from goal to
                     successor (This can be done using many
                     ways, we will discuss three heuristics-
                     Manhattan, Diagonal and Euclidean
                     Heuristics)

         successor.f = successor.g + successor.h
      ii) if a node with the same position as
          successor is in the OPEN list which has a
          lower f than successor, skip this successor
      iii) if a node with the same position as
          successor is in the CLOSED list which has
          a lower f than successor, skip this successor
          otherwise, add the node to the open list
   end (for loop)

e) push q on the closed list
end (while loop)

```



<https://qiao.github.io/PathFinding.js/visual/>

Secret Behind A Successful Programmer

