



OBODROID WORKSHOP

Day 1 - Introduction to
ROS Navigation Stack



INTRO TO



NAVIGATION STACK



THEPPASITH NISITSUKCHAROEN

Mail : theppasith.n@obodroid.com
Web : medium.itutor.name



THIRUCH RAKTHAO

Mail : thiruch.r@obodroid.com



obodroid
Corporation Limited

WORKSHOP
2021

ก่อนอื่น . . .

ลงโปรแกรมกันเถอะ

ช่วงเวลาแห่งการก็อปโค๊ด !

ມີຂອງຄຣບແລ້ວຫົວໜ້າຍ !

- Ubuntu 18.04 (Bionic Beaver)
- ROS : Melodic
- ROS : Install Navigation Stack Package <ros-melodic-navigation>
 - > sudo apt install ros-melodic-navigation
 - > sudo apt install ros-melodic-robot-localization
- Simulator : Ignition Dome
 - > https://ignitionrobotics.org/docs/dome/install_ubuntu
- Simulator to ROS : ros-ign-bridge
 - > ລອງທ່າຕາມ https://github.com/ignitionrobotics/ros_ign
- Catkin tools ຕ້ວໃໝ່
 - > sudo apt install python-catkin-tools

มีของครบแล้วหรือยัง !

Learning Material

- สร้าง Catkin workspace
 - สร้าง Folder ใน home ชื่อ workspace ก็ได้นะ
 - สร้าง Folder workspace/src
 - ใช้ catkin init
 - (src)> git clone https://bitbucket.org/obodroid/ignition_playground
 - (src)> git clone https://bitbucket.org/obodroid/ignition_navigation
- Build Workspace
 - (workspace root) > catkin build
[เหมือน catkin_make แหลก แต่อันนี้ใหญ่กว่า ใช้สะดาวกกว่า]
- Source Workspace เข้าไปใน bashrc
 - ไฟล์ ~/.bashrc (มีจุดหน้าไฟล์)
 - เพิ่ม ต่อท้าย File

```
source ~/workspace-devel/setup.bash
export IGN_GAZEBO_RESOURCE_PATH=/home/<username>/workspace/src/ignition_playground
```
- ลอง Run
 - > roslaunch ignition_playground block.launch
 - > roslaunch ignition_playground offline_team.launch
 - > roslaunch ignition_navigation bringup.launch

ມີຂອງຄຣບແລ້ວທີ່ອຢັ້ງ !

Learning Material

- ລອງ Run
 - > roslaunch ignition_playground block.launch
 - > roslaunch ignition_playground offline_team.launch
 - > roslaunch ignition_navigation bringup.launch

เสร็จแล้ว

กลับบ้านได้

. . . (ล้อเล่น)



OPERATING SYSTEM



Program นี้



Program นี้

The ROS logo consists of a white graphic element to the left of the word "ROS". The graphic element is composed of four rows of three circular dots each, creating a 4x3 grid pattern.

ROS

พื้นฐานของ Robot Operating System

- Stanford Robotics Lab -> Willow Garage
- ทำมาใช้กับหุ่นยนต์ PR2
- Reusable Component
- ไม่ Reinvent the wheel บ่อยๆ เพราะเสียเวลา
- Messaging between Processes
- Message Data Structure
- มี ROS Standard ทำให้ทุกๆ คนเขียนคล้ายๆ กัน
- Community-Driven development

ROS



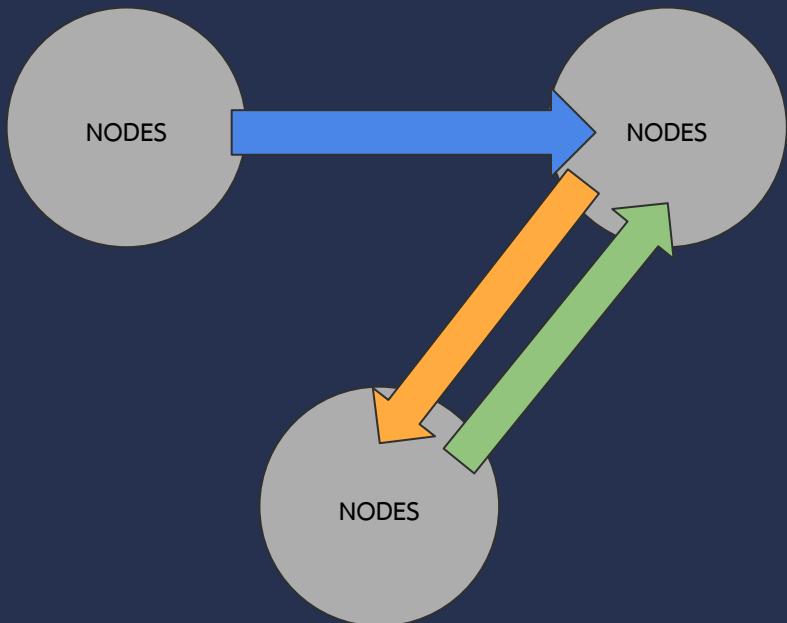
Willow
Garage



พื้นฐานของ Robot Operating System

ROS Computation Graph System

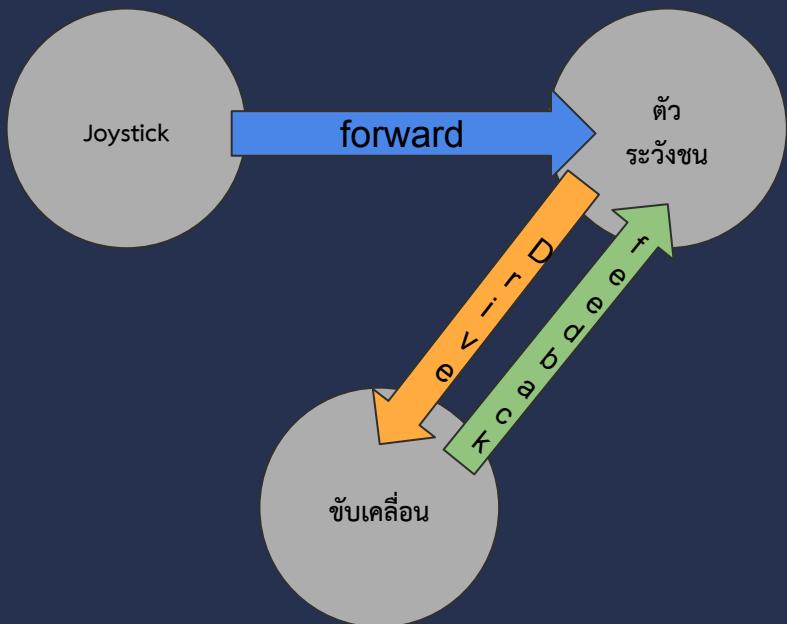
- Nodes
 - แบ่ง Software ออกเป็นเรื่องๆ
- Messaging system
 - รับส่งข้อมูลความกันระหว่าง Nodes
 - Publish and Subscribe (pub/sub)
 - Service server and Service Client
 - Action Library (actionlib)
- Build System
 - ทำให้จัดการงานใหญ่ได้ง่าย
 - catkin_make , catkin build (catkin-tools)



พื้นฐานของ Robot Operating System

ROS Computation Graph System

- Nodes
 - แบ่ง Software ออกเป็นเรื่องๆ
- Messaging system
 - รับส่งข้อมูลความกันระหว่าง Nodes
 - Publish and Subscribe (pub/sub)
 - Service server and Service Client
 - Action Library (actionlib)
- Build System
 - ทำให้จัดการงานใหญ่ได้ง่าย
 - catkin_make , catkin build (catkin-tools)



แต่วันนี้

เราจะใช้ Node หลายๆ อัน

ทำให้หุ่นยนต์ได้เดินเองกัน !

หัวข้อวันนี้ !

จบวันนี้อย่างให้ทำแบบนี้ได้

Part 1 : Introduction and Components

- ROS Navigation Stack คืออะไร
 - ทำไมเราต้องใช้ Navigation Stack
 - เตรียมของที่ต้องใช้ใน Navigation Stack
 - Component ต่างๆใน Navigation Stack
- แบบคร่าวๆ

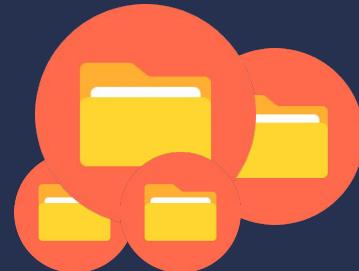
Part 2 : Component Detail

- Component แบบละเอียด
อย่างไรทำงานยังไง
- AMCL node , move_base node

เริ่มได้

ROS Navigation Stack

- ROS Stack คือ กลุ่ม Packages (Nodes) ที่ช่วยกันทำงานให้ญี่ๆ
 - มี หลาย Packages ย่อยๆ มารวมกัน
 - เช่น Navigation Stack , Manipulation Stack (moveit)
- Navigation Stack เป็นการรวม Packages ที่ช่วยให้หุ่นยนต์ทำ Autonomous Navigation ได้นั่นเอง
- หน้าที่ : รับคำสั่งให้เดินแบบอัตโนมัติไปตามแผนที่ , หลบสิ่งกีดขวางได้เอง



Why ROS Navigation Stack ?



Community-based
Development

มีคนเทพๆ มาเขียน + แก้บัค

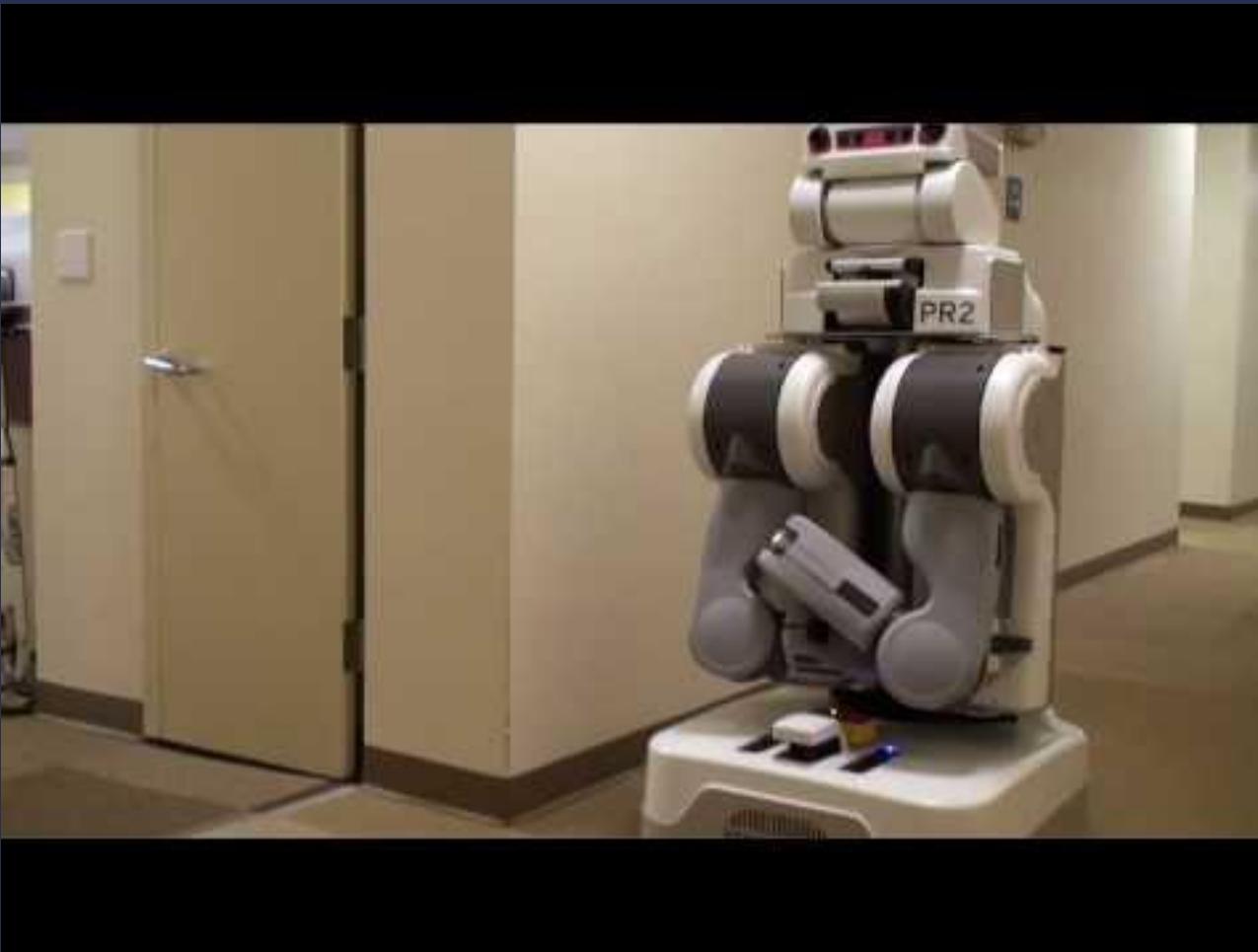


Basic Navigation
ไม่ต้อง Code มาก
แค่ปรับ Config

navigation
ROS Navigation stack. Code for somewhere else.

robotics navigation ros

C++ 1,390 ⌂ 1,228 ⭐ Forks : 1,390 ครั้ง
Downloads : . . .
(เยอะเหละ หาข้อมูลไม่ทัน 555)



ปล. นีเป็น
วิดีโอ ตอนปี 2010

ROS Navigation Stack : Components

รู้ว่าตัวเองอยู่ตรงไหน

ทำให้หุ่นยนต์ขับตามแผน

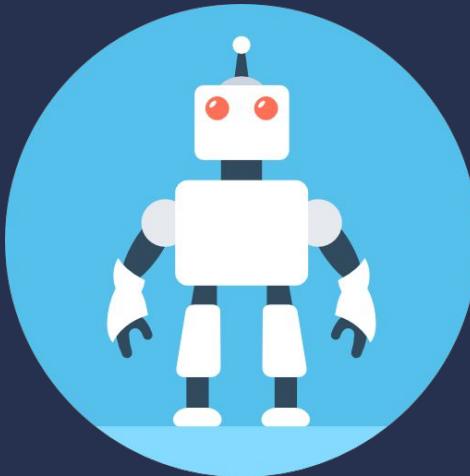
วางแผนเส้นทางการเดิน

หลบหลีกสิ่งกีดขวาง

สร้างแผนที่สิ่งกีดขวาง

ตอนเวลาพัง ทำยังไงดี?

ความรู้เกี่ยวกับแผนที่



ROS Navigation Stack : Components

รู้ว่าตัวเองอยู่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

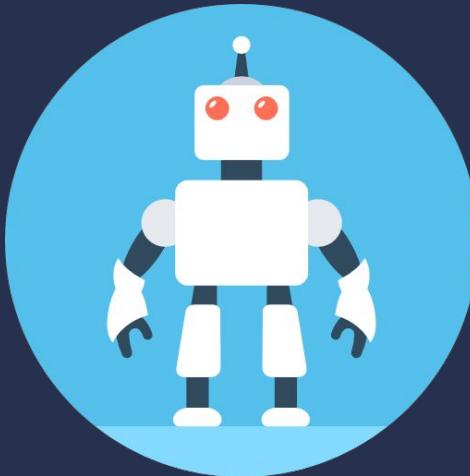
สร้างแผนที่สิ่งกีดขวาง
(OBSTACLE SPACE)

ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

ทำให้หุ่นยนต์ขยับตามแผน
(MOTION EXECUTION)

หลบหลีกสิ่งกีดขวาง
(OBSTACLE AVOIDANCE)

ตอนเวลาพัง ทำยังไงดี?
(RECOVERY BEHAVIOR)



ROS Navigation Stack : Components

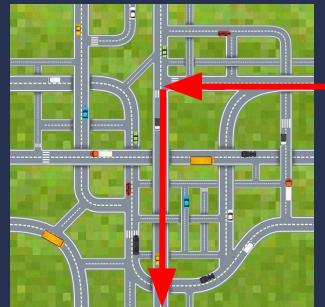
ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)



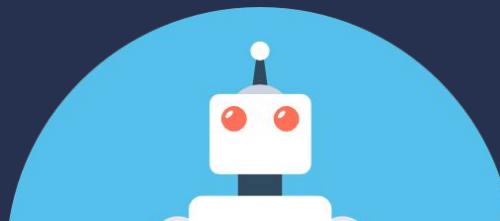
รู้ว่าตัวเองอยู่ตรงไหน
(LOCALIZATION)



วางแผนเส้นทางการเดิน
(MOTION PLANNING)



ทำให้หุ่นยนต์ขับตามแผน
(MOTION EXECUTION)



ROS Navigation Stack : Components

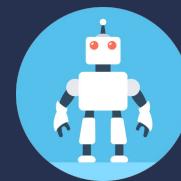


สมมติว่าเราไปเที่ยวต่างเมือง ไม่คุ้นทางเลย

- มีแผนที่ให้ 1 อัน
- มีคนเดาๆนั่งบอกว่า
เราอยู่ตรงไหนในแผนที่
- เราอยากจะเดินทางไปที่ๆหนึ่ง

ลองคิดดู . . ว่าจะเกิดอะไรขึ้นบ้าง ?

ROS Navigation Stack : Components



ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

รู้ว่าตัวเองอยู่ที่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขยับตามแผน
(MOTION EXECUTION)



ROS Navigation Stack : Components

ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

รู้ว่าตัวเองอยู่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขยับตามแผน
(MOTION EXECUTION)

รู้ว่าตัวเองอยู่เดาๆ ในแผนที่แล้ว



ROS Navigation Stack : Components

ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

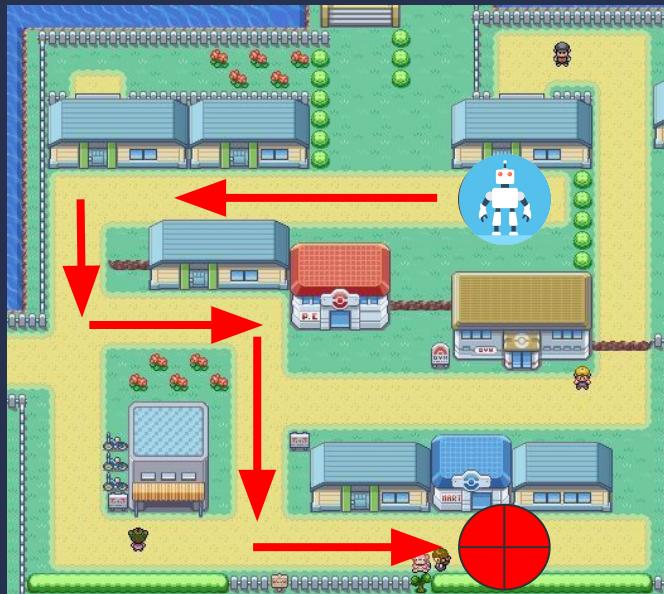
รู้ว่าตัวเองอยู่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขยับตามแผน
(MOTION EXECUTION)

รู้ว่าจะไป哪儿ไหน +

วางแผนว่าจะเดินไปตามถนนไหน ทางไหนตัด
ไม่ได้ ทางไหนดี เดินใกล้ เดินง่าย



ROS Navigation Stack : Components

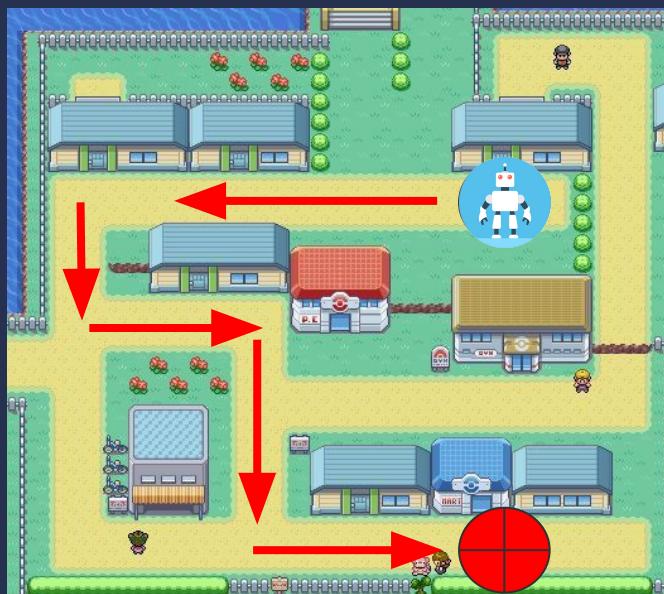
ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

รู้ว่าตัวเองอยู่ที่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขยับตามแผน
(MOTION EXECUTION)

เดินตามแผนไป
ด้วยการก้าวขา
การเดิน การวิ่ง



ROS Navigation Stack : Components

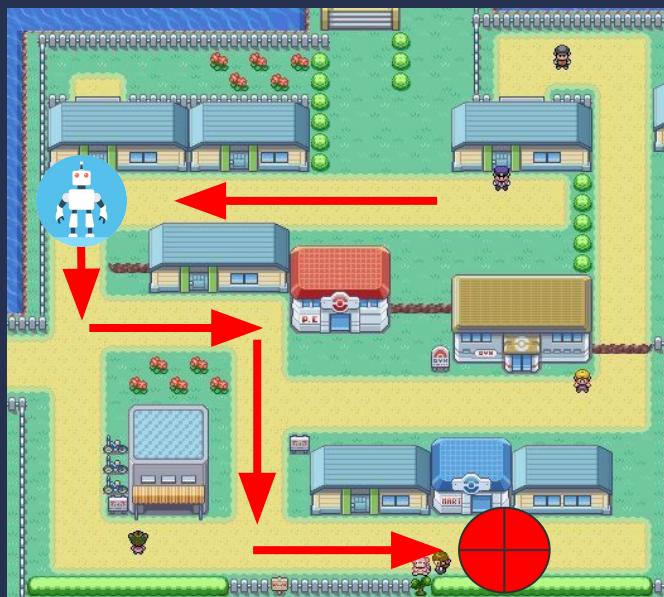
ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

รู้ว่าตัวเองอยู่ที่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขยับตามแผน
(MOTION EXECUTION)

เดินตามแผนไป
ด้วยการก้าวขา
การเดิน การวิ่ง



ROS Navigation Stack : Components

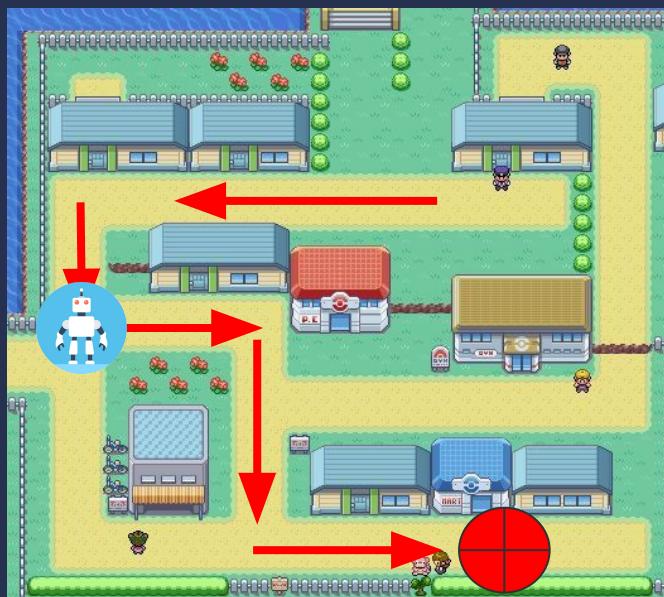
ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

รู้ว่าตัวเองอยู่ที่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขยับตามแผน
(MOTION EXECUTION)

เดินตามแผนไป
ด้วยการก้าวขา
การเดิน การวิ่ง



ROS Navigation Stack : Components

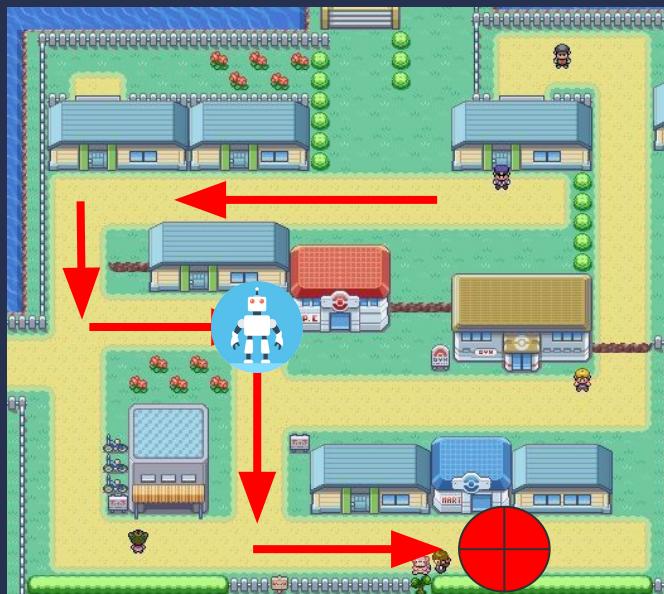
ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

รู้ว่าตัวเองอยู่ที่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขยับตามแผน
(MOTION EXECUTION)

เดินตามแผนไป
ด้วยการก้าวขา
การเดิน การวิ่ง



ROS Navigation Stack : Components

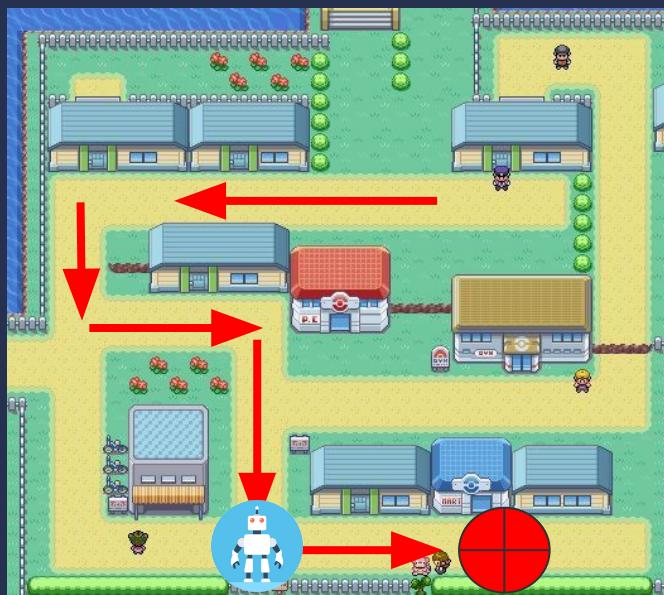
ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

รู้ว่าตัวเองอยู่ที่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขยับตามแผน
(MOTION EXECUTION)

เดินตามแผนไป
ด้วยการก้าวขา
การเดิน การวิ่ง



ROS Navigation Stack : Components

ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

รู้ว่าตัวเองอยู่ที่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขยับตามแผน
(MOTION EXECUTION)

เดินตามแผนไป
ด้วยการก้าวขา
การเดิน การวิ่ง



เกิดสิ่งไม่คาดคิด

- แล้วถ้าเส้นทางปิดล่ะ ?



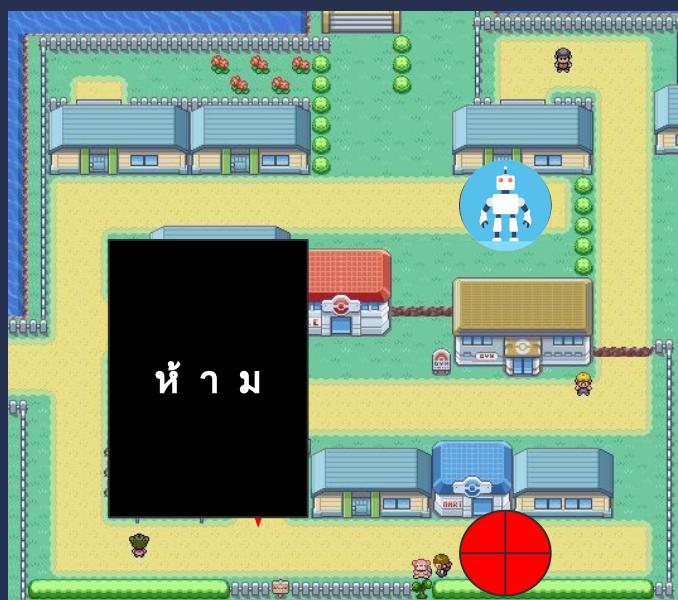
ROS Navigation Stack : Components

ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

รู้ว่าตัวเองอยู่ที่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขับตามแผน
(MOTION EXECUTION)



เกิดสิ่งไม่คาดคิด

- แล้วถ้าเส้นทางปิดล่ะ ?

สร้างแผนที่สิ่งกีดขวาง
(OBSTACLE SPACE)

หลบหลีกสิ่งกีดขวาง
(OBSTACLE AVOIDANCE)



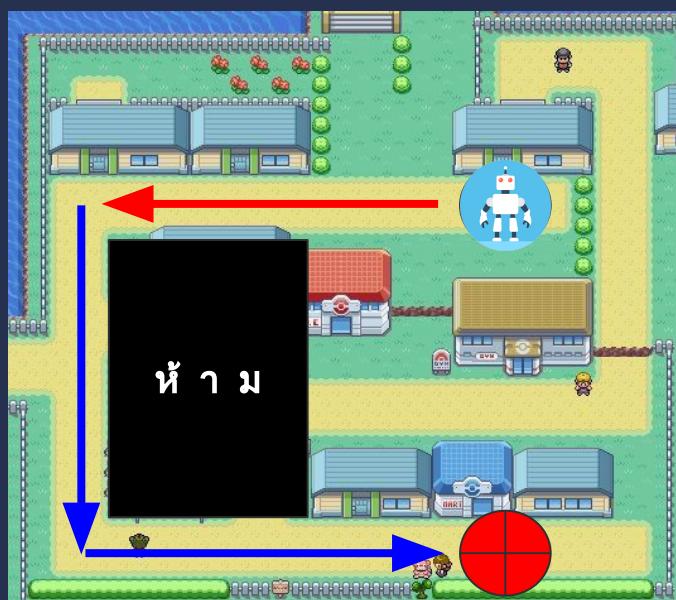
ROS Navigation Stack : Components

ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

รู้ว่าตัวเองอยู่ที่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขับตามแผน
(MOTION EXECUTION)



เกิดสิ่งไม่คาดคิด

- แล้วถ้าเส้นทางปิดล่ะ ?

สร้างแผนที่สิ่งกีดขวาง
(OBSTACLE SPACE)

หลบหลีกสิ่งกีดขวาง
(OBSTACLE AVOIDANCE)



เกิดสิ่งไม่คาดคิด

- แล้วถ้าเดินทางอยู่เมืองใดมาตั้งขวางทางทันทีเลย
จะทำยังไงดี ?

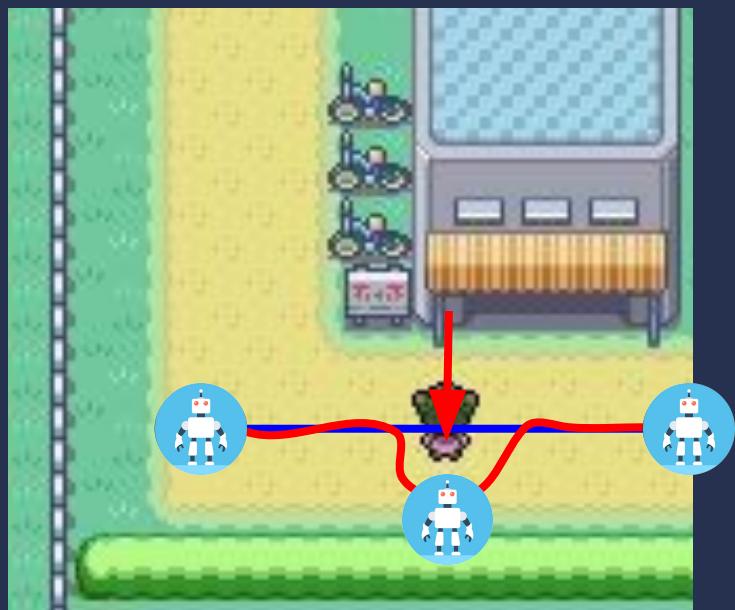
ROS Navigation Stack : Components

ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

รู้ว่าตัวเองอยู่ที่ตรงไหน
(LOCALIZATION)

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

ทำให้หุ่นยนต์ขับตามแผน
(MOTION EXECUTION)



สร้างแผนที่สิ่งกีดขวาง
(OBSTACLE SPACE)

หลบหลีกสิ่งกีดขวาง
(OBSTACLE AVOIDANCE)

ROS Navigation Stack : Components

รู้ว่าตัวเองอยู่ตรงไหน
(LOCALIZATION)

- AMCL

วางแผนเส้นทางการเดิน
(MOTION PLANNING)

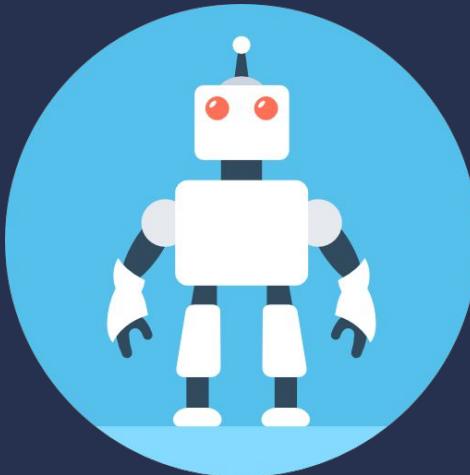
- GLOBAL PLANNER

สร้างแผนที่สิ่งกีดขวาง
(OBSTACLE SPACE)

- GLOBAL/LOCAL COSTMAP

ความรู้เกี่ยวกับแผนที่
(MAP KNOWLEDGE)

- MAP SERVER



ทำให้หุ่นยนต์ขับตามแผน
(MOTION EXECUTION)

- GLOBAL/LOCAL PLANNER

หลบหลีกสิ่งกีดขวาง
(OBSTACLE AVOIDANCE)

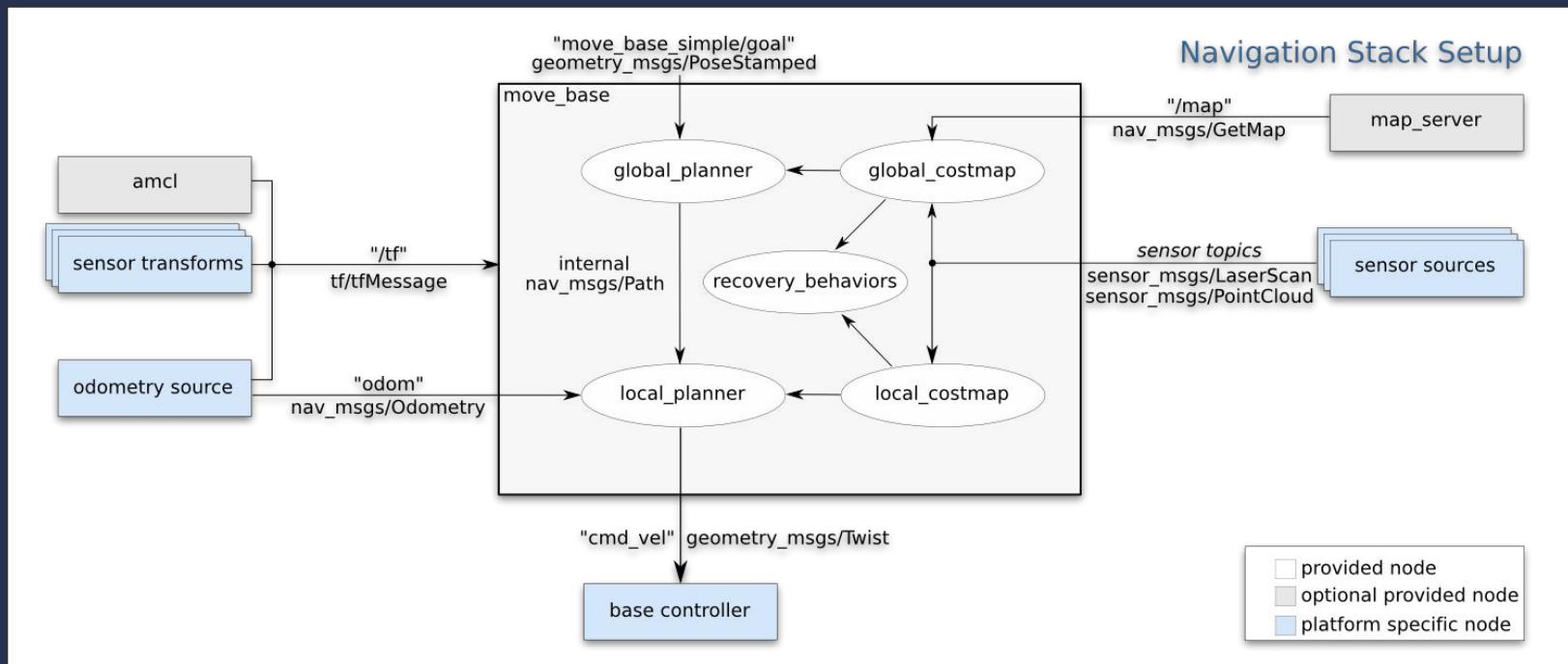
- LOCAL PLANNER

ตอนเวลาพัง ทำยังไงดี?
(RECOVERY BEHAVIOR)

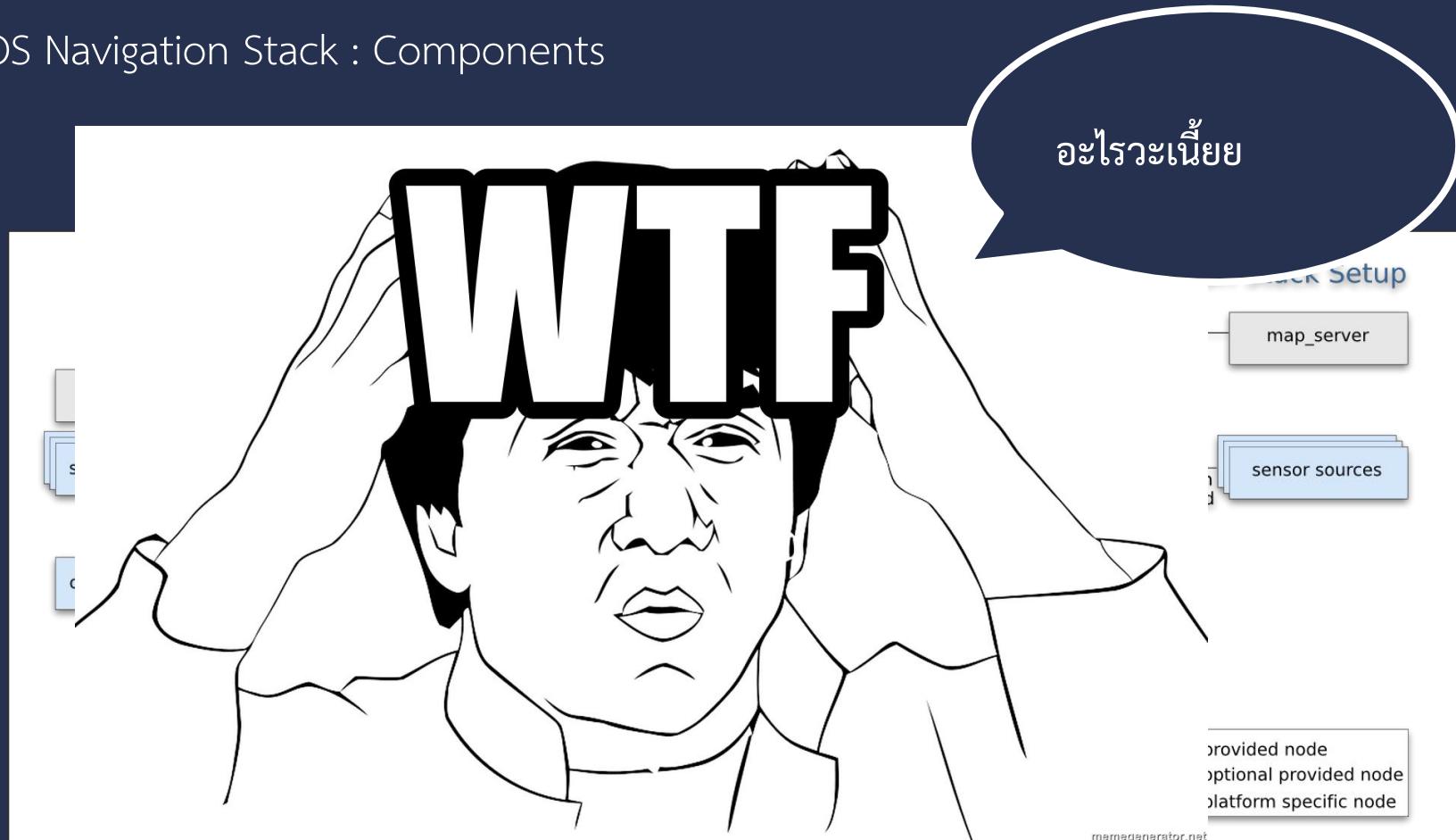
- MOVEBASE RECOVERY BEHAVIOR

ROS Navigation Stack : Components

ถ้าดูรูปจาก Web ROS เลย
Navigation Stack หน้าตาจะเป็นแบบนี้ !



ROS Navigation Stack : Components



ROS Navigation Stack : Components (Nodes)

เพื่อความง่าย แบ่งเป็น 3 ส่วนหลักๆ

รู้ว่าตัวเองอยู่ตรงไหน

AMCL

เดินไปจุดหมายด้วยตัวเอง

MOVE_BASE

MAP_SERVER

โหลดไฟล์รูป -> แผนที่

ROS Navigation Stack : Components (Nodes)

แล้วมันเกี่ยวกับการเดินแบบกี้ยังไง ?



รู้ว่าตัวเองอยู่ตรงไหน



เดินไปจุดหมายด้วยตัวเอง



MAP_SERVER

โหลดไฟล์รูป -> แผนที่

ROS Navigation Stack : Components (Nodes)

รู้ว่าตัวเองอยู่ตรงไหน



เดินไปจุดหมายด้วยตัวเอง

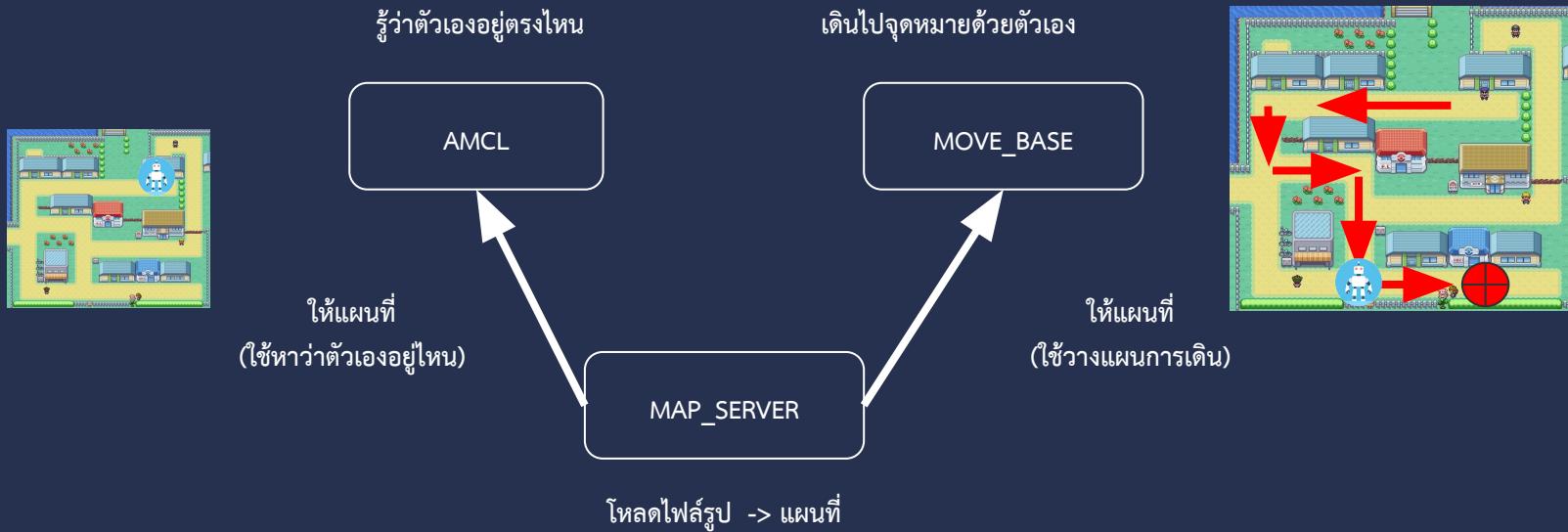


MAP_SERVER

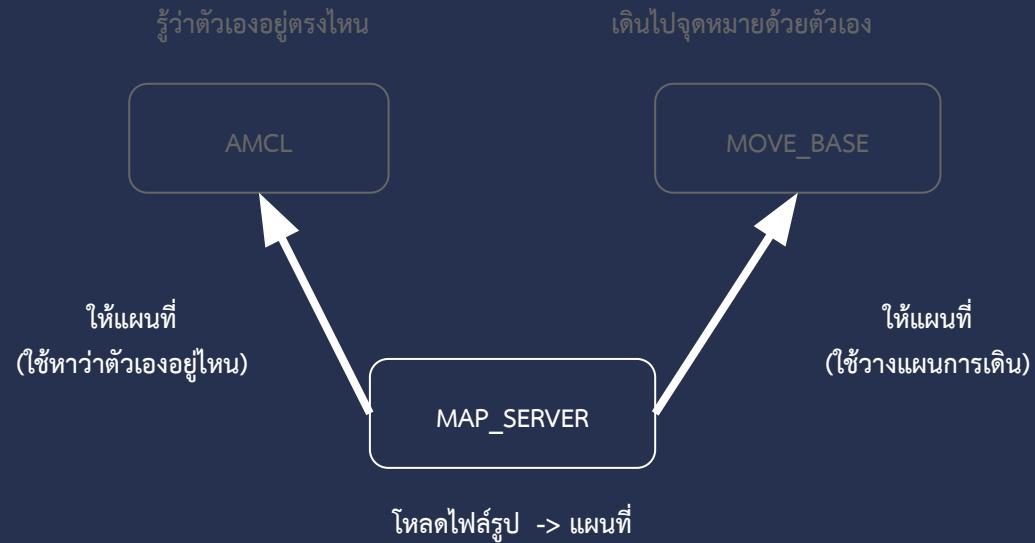


โหลดไฟล์รูป -> แผนที่

ROS Navigation Stack : Components (Nodes)



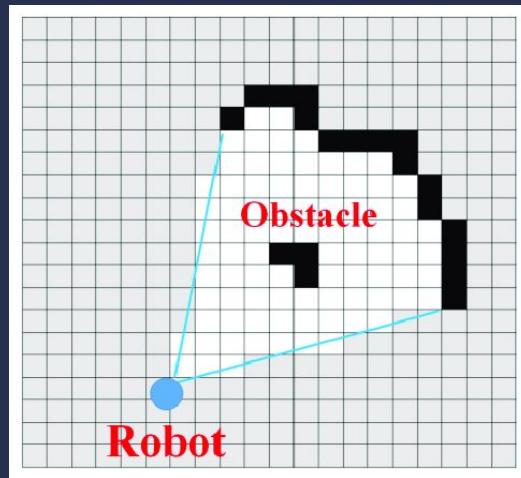
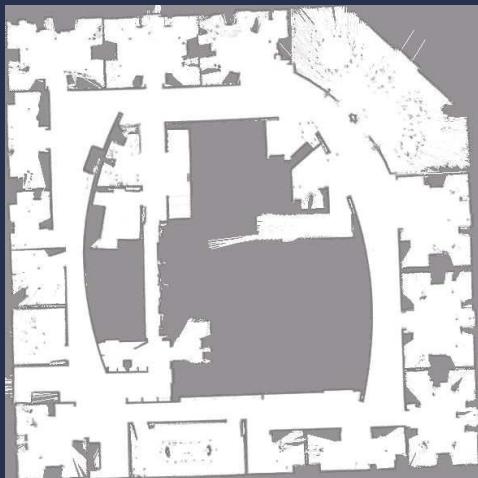
ROS Navigation Stack : Components (Nodes)



ROS Navigation Stack : Map Server

แผนที่ (Map) : ใช้เพื่อคุ้น และ อ้างอิงตำแหน่งหุ่นยนต์ !

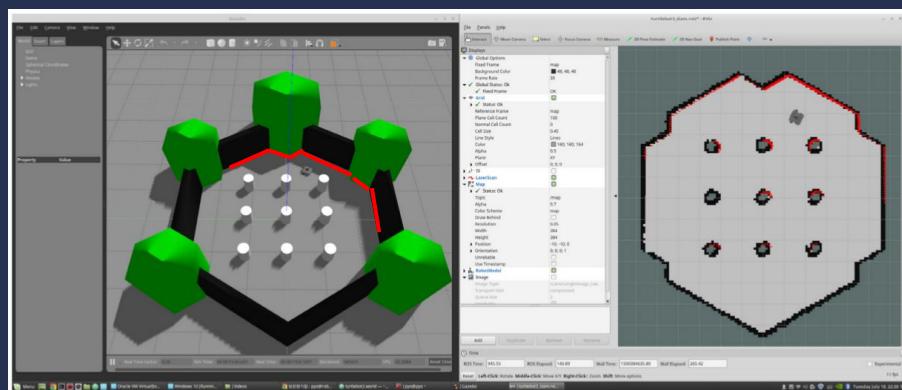
- จะเป็นแบบ 2D, 3D ก็เป็นไปได้
- ROS Navigation Stack เราจะใช้ 2D TOP VIEW ที่เรียกว่า Occupancy Grid



ROS Navigation Stack : Map Server

map_server : แปลงรูปภาพ MAP (png, pgm) ไปเป็น Occupancy Grid และส่งให้คนอื่นๆ (pub /map)

- 2D Array ที่แต่ละช่องมี 3 STATES
 - FREE พื้นที่โล่งๆ เดินได้
 - OCCUPIED มีของอยู่แล้วในนั้น (สิ่งกีดขวาง , กำแพง)
 - UNKNOWN พื้นที่ที่ไม่แน่ใจว่ามีอะไรในนั้น (อาจว่าง, ไม่ว่าง ก็ได้)
- 1 ช่อง เท่ากับความยาวเท่าไรในโลกจริง ?
 - RESOLUTION หน่วยเป็น meter/pixel เช่น $0.05 \text{ m}/\text{px} = 1 \text{ ช่อง}$ มีขนาด $5 \times 5 \text{ cm}$.



Real World

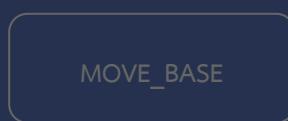
Occupancy Grid

ROS Navigation Stack : Components (Nodes)

รู้ว่าตัวเองอยู่ตรงไหน



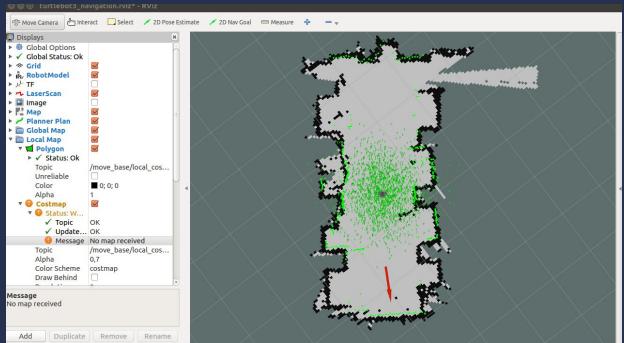
เดินไปจุดหมายด้วยตัวเอง



MAP_SERVER

โหลดไฟล์รูป -> แผนที่

ROS Navigation Stack : AMCL



รู้ว่าตัวเองอยู่ตรงไหน

AMCL



AMCL เป็น Node ที่รับข้อมูล แผนที่ และ เชนเชอร์เข้าไป

จะทำหน้าที่ ามาว่า เรายู่ตรงไหนในแผนที่ !

โดยใช้วิธีแยกย่างไปทั่วๆ และหาว่าเราอยู่แฉะไหน !

ROS Navigation Stack : AMCL



ROS Navigation Stack : Components (Nodes)

รู้ว่าตัวเองอยู่ตรงไหน

AMCL

เดินไปจุดหมายด้วยตัวเอง

MOVE_BASE

MAP_SERVER

โหลดไฟล์รูป -> แผนที่

ROS Navigation Stack : move_base

เดินไปจุดหมายด้วยตัวเอง

เมื่อเรารู้ว่าเราอยู่ไหนในแผนที่

Node นี้จะช่วย

- วางแผนการเดินทางระยะไกล
 - ด้วยแผนที่ + ข้อมูลภูมิศาสตร์
 - ได้เส้นทาง(ระยะไกลๆ)ออกมาก
 - เมื่อันกับ : Google Map
- วางแผนการเดินทางระยะใกล้
 - ด้วยแผนที่ของสิ่งแวดล้อมรอบตัว
 - ระวังเรื่องการโดนขวาง และตัดหน้า
 - ค่อยเดินตามทาง
 - เมื่อันกับ : เราที่ดูเส้นทาง และ ขับตาม



ลองเล่นกันเถอะ !

ROS Navigation Stack : เริ่มใช้งาน

ใช้ ROSLAUNCH RUN ไฟล์ตามลำดับต่อไปนี้
(สลับลำดับอาจทำให้เพี้ยนๆไปได้ดันนะ)

Simulation

1. [Terminal 1] เปิด Simulation

```
> rosrun ignition_playground block.launch
```

2. [Terminal 2] เปิดตัวควบคุมทุนใน Simulation

```
> rosrun ignition_playground offline_team.launch
```

จะได้ Visualizer RVIZ และมาด้วย

แต่ยังไม่มีอะไรขึ้นบนจอ

คำสั่ง ROS CLI (command line interface)
(คำสั่งที่พิมพ์ใน Terminal ได้)

พื้นฐาน ลองเล่นก็ได้นะ ไม่ผิดๆ

```
> roscore  
> rosrun <package_name> <executable_name>  
> rosrun <package_name> <launch_file>  
> rosnode list  
> rostopic list  
> rviz
```

ROS Navigation Stack : เริ่มใช้งาน

ใช้ ROSLAUNCH RUN ไฟล์ตามลำดับต่อไปนี้

(สลับลำดับอาจทำให้เพี้ยนๆไปได้บ้าง)

Robot

1. [Terminal 3] Map_server

```
> rosrun ignition_navigation map_server.launch
```

2. [Terminal 4] amcl

```
> rosrun ignition_navigation amcl.launch
```

3. [Terminal 5] move_base

```
> rosrun ignition_navigation move_base.launch
```

ROS Navigation Stack : เริ่มใช้งาน

ใช้ ROSLAUNCH RUN ไฟล์ตามลำดับต่อไปนี้
(สลับลำดับอาจทำให้เพี้ยนๆไปได้นะ)

การเปิด Simulation

1. [Terminal 1] เปิด Simulation
> rosrun ignition_playground block.launch
2. [Terminal 2] เปิดตัวควบคุมหุ่นใน Simulation
> rosrun ignition_playground offline_team.launch
จะได้ Visualizer RVIZ แสดงมาด้วย
แต่ยังไม่มีอิเล็กทรอนิกส์บนหุ่น

การเปิด Navigation Stack Nodes

3. [Terminal 3] map_server
> rosrun ignition_navigation map_server.launch
4. [Terminal 4] amcl
> rosrun ignition_navigation amcl.launch
5. [Terminal 5] move_base
> rosrun ignition_navigation move_base.launch

พักเบรค !

กลับมาเจอกันตอน 14:00 น. เด้อ

Next : PART 2 - Deeper Detail

PART 2 : Navigation Parameters

เจาะลึก (มากขึ้น)

แต่ละอัน ทำงานยังไงบ้าง

ROS Navigation Stack : Components (Nodes)

รู้ว่าตัวเองอยู่ตรงไหน

AMCL

เดินไปจุดหมายด้วยตัวเอง

MOVE_BASE

MAP_SERVER

โหลดไฟล์รูป -> แผนที่

ROS Navigation Stack : Requirement

- SETUP ลักษณะหุ่นยนต์ของเรา (Robot Configuration)

1. TRANSFORMATION FRAME (TF)

- รู้ว่า Sensor เราติดอยู่ตรงไหน
 - ต้องอยู่ตรงพื้นอยู่ตรงไหน (หุ่นบางตัวยกขา ยกล้อได้)
ต้องมีตำแหน่งอ้างกับพื้น

2. MAP (ข้อมูลแผนที่)

- ใช้รูปที่จะถูกมาเป็น Occupancy Grid
- อยู่ใน PART 1 นะจ๊ะ

3. ODOMETRY

- ข้อมูลจากล้อ, จากภายในหุ่นเอง ,
คำนวนผลลัพธ์ตำแหน่งของหุ่น แบบไม่ได้พิงสิ่งแวดล้อม

ROS Navigation Stack : Requirement

- SETUP ลักษณะหุ่นยนต์ของเรา (Robot Configuration)

1. TRANSFORMATION FRAME (TF)

- รู้ว่า Sensor เราติดอยู่ตรงไหน
 - ลักษณะหุ่นยนต์ของเรา (หุ่นบางตัวยกขา ยกล้อได้)
ต้องมีตำแหน่งอ้างกับพื้น

2. MAP (ข้อมูลแผนที่)

- ใช้รูปที่จะถูกนำมาเป็น Occupancy Grid
- อยู่ใน PART 1 นะจ๊ะ

3. ODOMETRY

- ข้อมูลจากล้อ, จากภายนอกหุ่นยนต์,
คำนวณผลลัพธ์ตำแหน่งของหุ่นยนต์แบบไม่ได้พิงสิ่งแวดล้อม

รู้ว่า Sensor อยู่ไหน : Transformation Frame Library (TF)

- Frame Of Reference
 - นุ่มมองของเรานั้นโลก Geometry
 - ถ้าเราอยู่ตุ่นนี้ เที่ยวนะแบบนี้ ตุ่นนั้นจะเห็นของยังไง ?

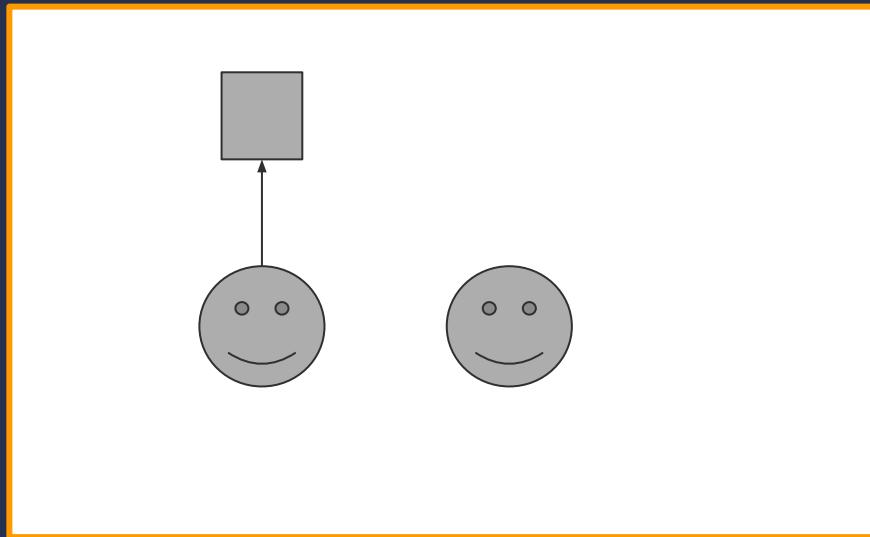
นาย A อยู่哪儿^ๆ
กลางห้องไปทางซ้าย



รู้ว่า Sensor อยู่ไหน : Transformation Frame Library (TF)

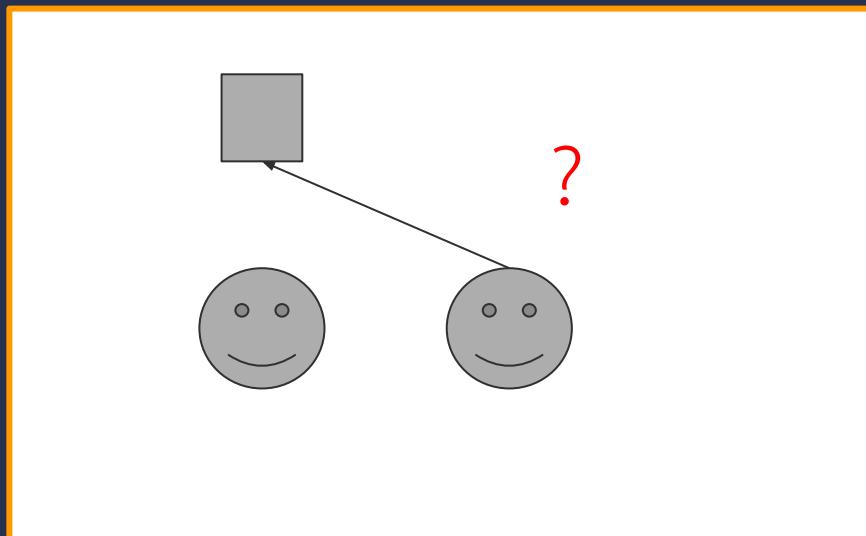
- Frame Of Reference
 - นูมของเรานะในโลก Geometry
 - ถ้าเราอยู่ตรงนี้ เที่ยงของแบบนี้ ตรงนั้นจะเที่ยงของยังไง ?

นาย A เที่ยงกล่อง
1 เมตร ด้านหน้า



รู้ว่า Sensor อยู่ไหน : Transformation Frame Library (TF)

- Frame Of Reference
 - นูมของเรานะในโลก Geometry
 - ถ้าเราอยู่ตรงนี้ เที่ยงของแบบนี้ ตรงนั้นจะเที่ยงของยังไง ?

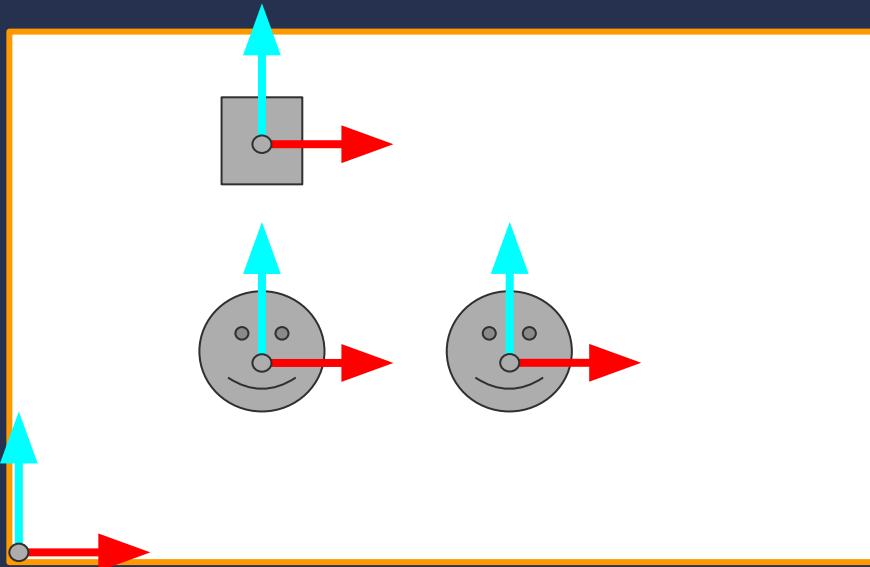


นาย B จะเที่ยงก่อต่องอยู่
ตรงไหน

รู้ว่า Sensor อยู่ไหน : Transformation Frame Library (TF)

- Frame Of Reference

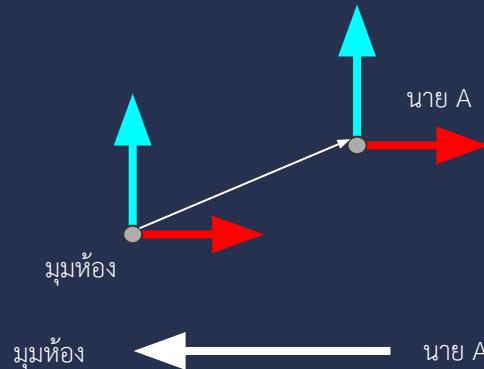
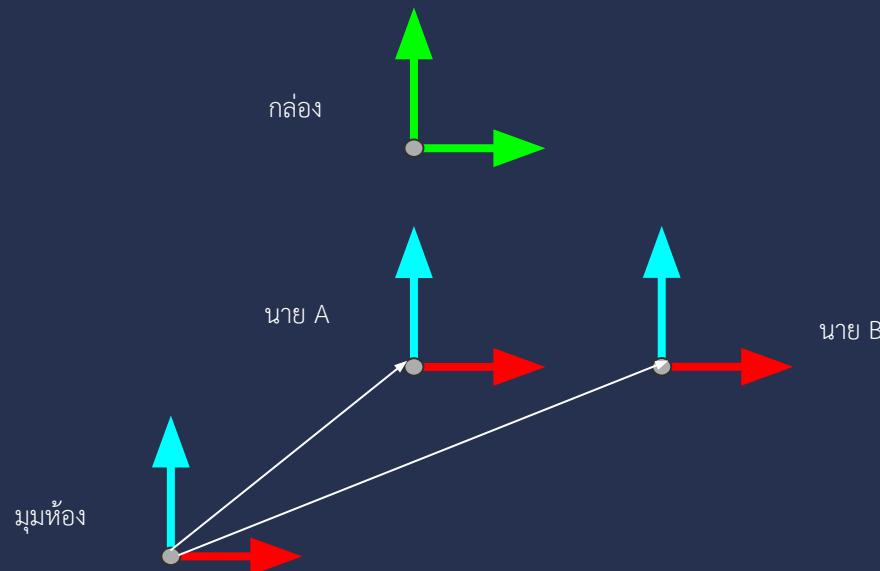
- นูมของเรานะในโลก Geometry
- ถ้าเราอยู่ตรงนี้ เที่ยงของแบบนี้ ตรงนั้นจะเที่ยงของยังไง ?



รู้ว่า Sensor อยู่ไหน : Transformation Frame Library (TF)

- Frame Of Reference

- มุมมองของเราในโลก Geometry
- ถ้าเราอยู่ตุ่นนี้ เที่ยวนะแบบนี้ ตรงนั้นจะเห็นของยังไง ?

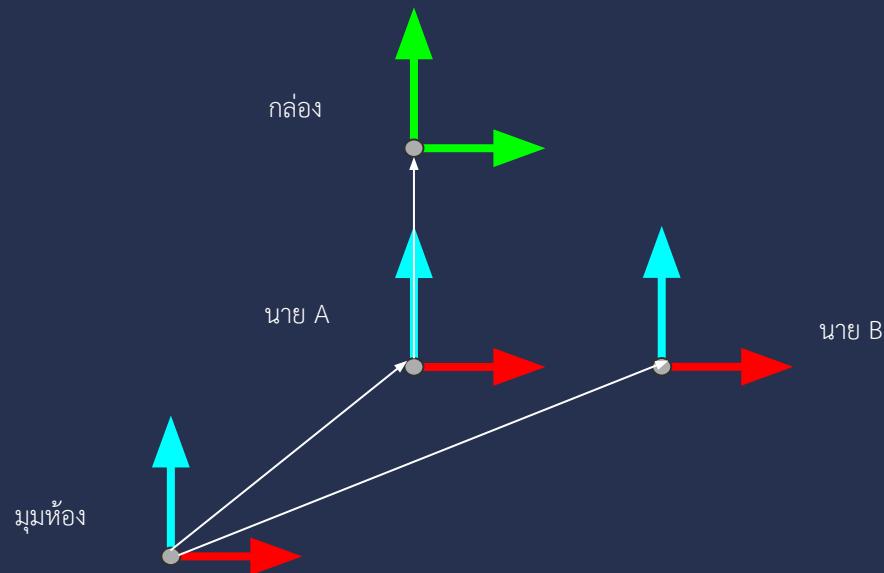


เส้นนี้เรียกว่า
ความสัมพันธ์
จาก นาย A ไปยัง มุมห้อง

รู้ว่า Sensor อยู่ไหน : Transformation Frame Library (TF)

- Frame Of Reference

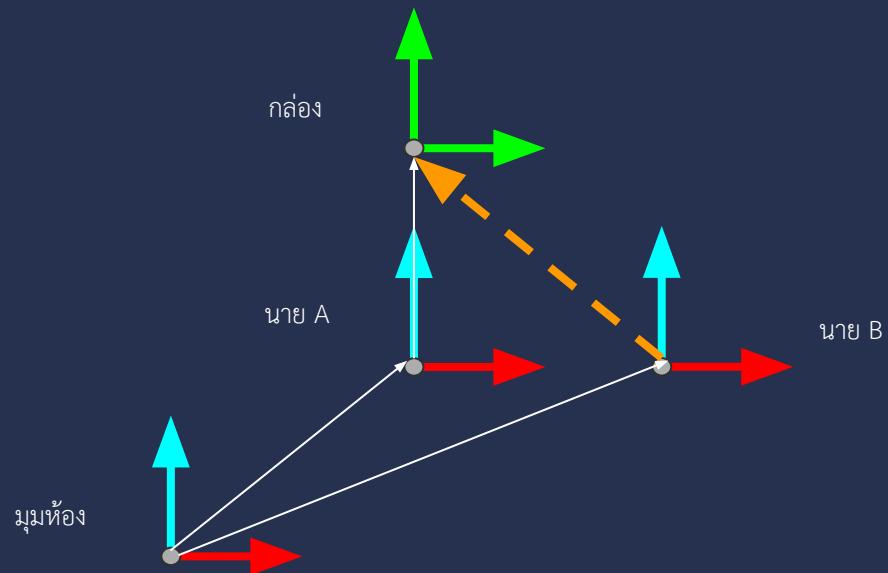
- นุ่มมองของเรานั้นโลก Geometry
- ถ้าเราอยู่ต่างนี้ เที่ยวนะแบบนี้ ตรงนั้นจะเห็นของยังไง ?



รู้ว่า Sensor อยู่ไหน : Transformation Frame Library (TF)

- Frame Of Reference

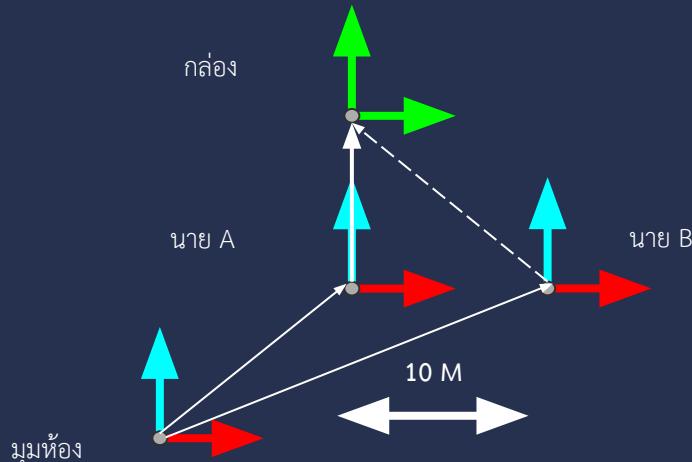
- นุ่มมองของเรานั้นโลก Geometry
- ถ้าเราอยู่ต่างนี้ เที่ยวนะแบบนี้ ตรงนั้นจะเห็นของยังไง ?



รู้จ้า Sensor อุปกรณ์ : Transformation Frame Library (TF)

- “TF” Transformation Frame (in ROS)

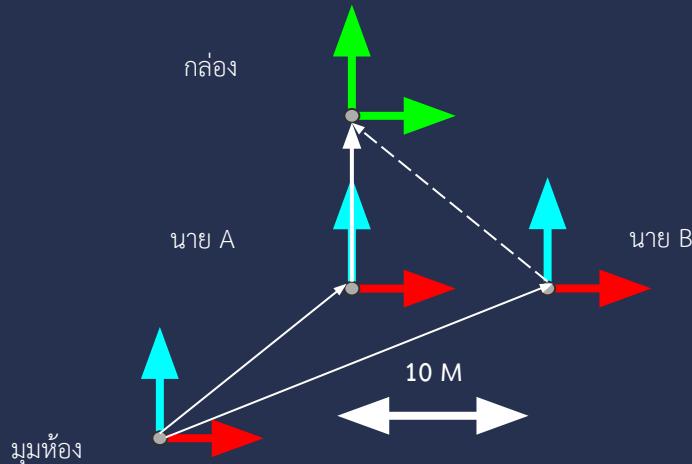
- เป็น ทั้ง Node และ Library
- TF Node หน้าที่ = สร้าง Frame of Reference
- TF Library หน้าที่ = ช่วยคำนวณเวลาเปลี่ยนมุมของ Frame of Reference
- มีความสัมพันธ์ของ Frame แบบ ต้นไม้ (Single Parent, Multiple Child)



$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -10 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

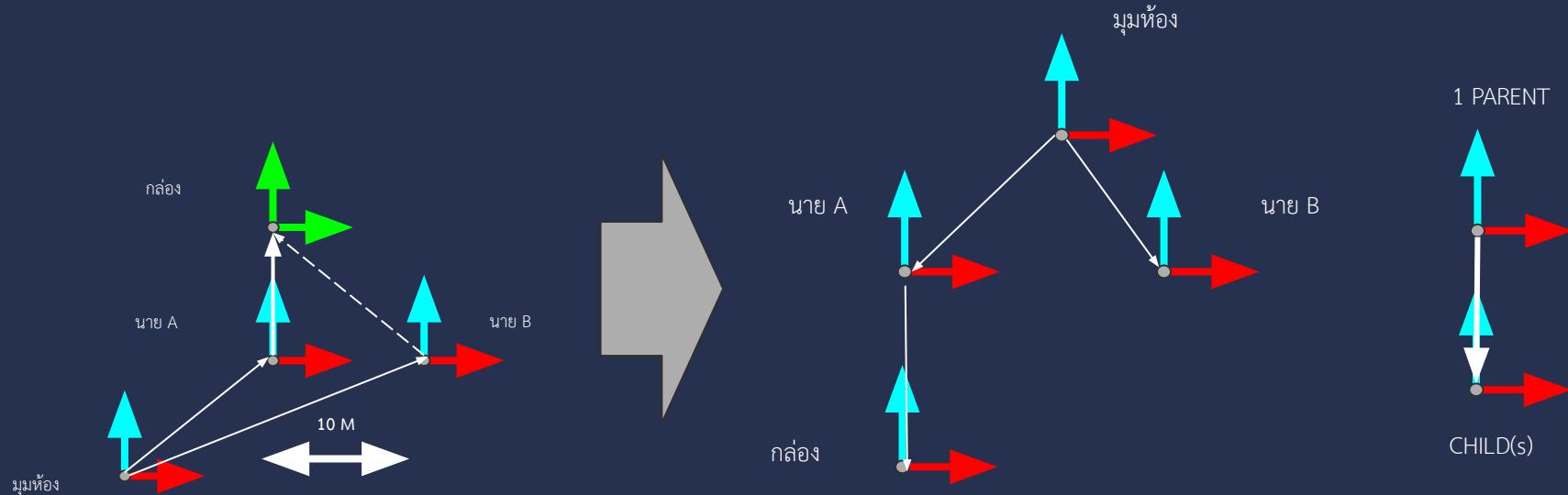
รู้ว่า Sensor อุปกรณ์ไหน : Transformation Frame Library (TF)

- Transformation Frame (in ROS)
 - เป็น ทั้ง Node และ Library
 - TF Node หน้าที่ = สร้าง Frame of Reference
 - TF Library หน้าที่ = ช่วยคำนวณเวลาเปลี่ยนมุมของ Frame of Reference
 - มีความสัมพันธ์ของ Frame แบบ ต้นไม้ม (Single Parent, Multiple Child)



pose_b_see_box = tf.lookupTransform("นาย B", "BOX")

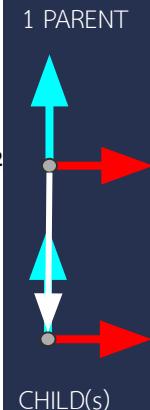
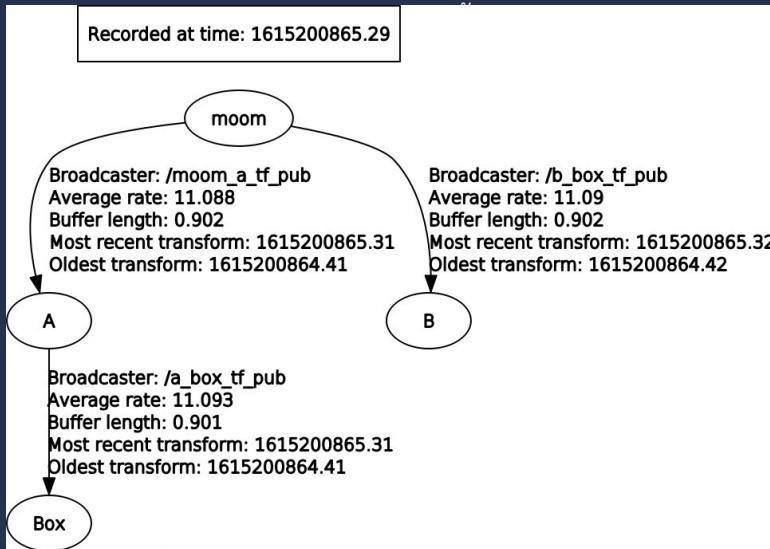
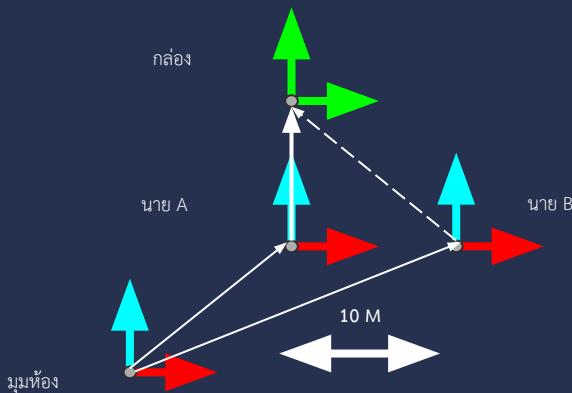
รู้ว่า Sensor อยู่ไหน : Transformation Frame Library (TF)



**สิ่งนี้เรียกว่า TF TREE

เวลาพูดถึง TF ต้องพูดเป็นคู่ๆ

รู้ว่า Sensor อยู่ไหน : Transformation Frame Library (TF)



**สิ่งนี้เรียกว่า TF TREE

เวลาพูดถึง TF ต้องพูดเป็นคู่ๆ

แล้ว

ต้องรู้จัก TF ไปทำไม่ล่ะ ?

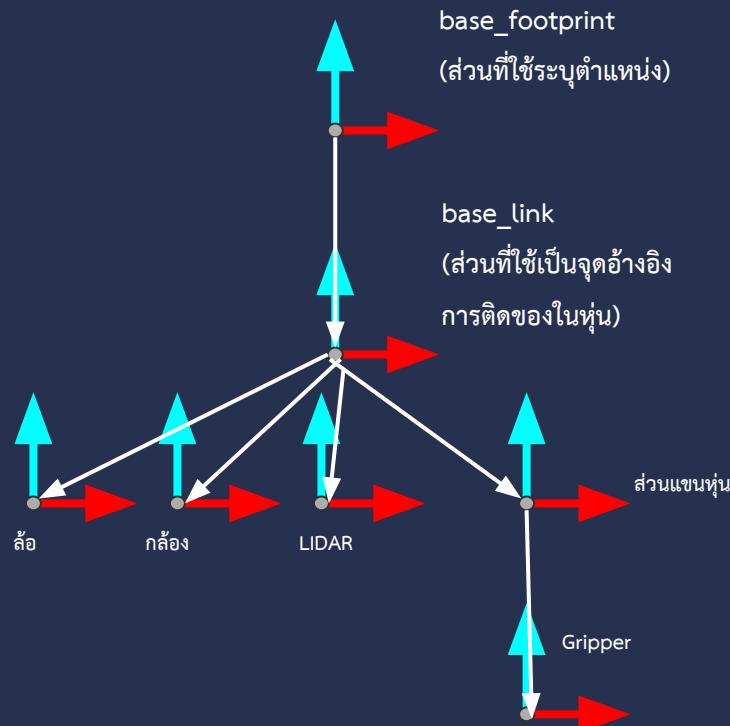
รู้ว่า Sensor อุปกรณ์ไหน : Transformation Frame Library (TF)

- ในหุ่นยนต์เรา ROS มีมาตรฐานการเรียกชื่อ

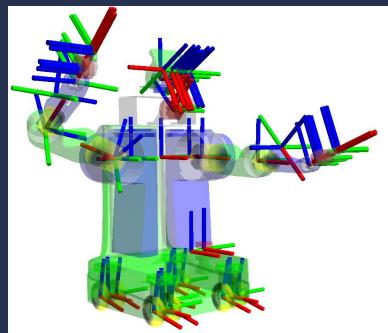
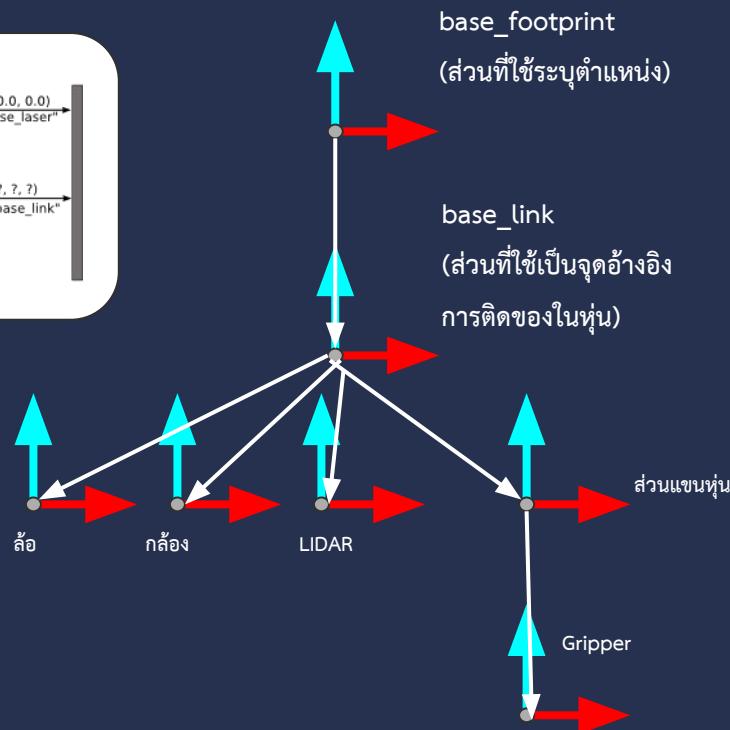
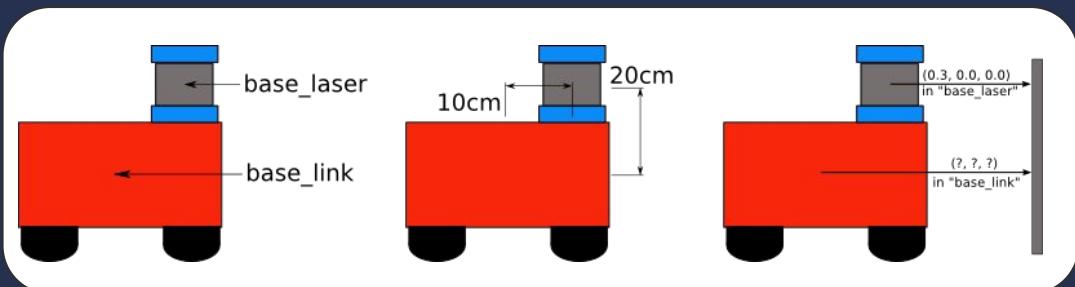
Frame of Reference ต่างๆอุปกรณ์

(เพื่อให้ทุกคนเข้าใจตรงกัน)

- base_footprint
คือ ส่วนที่ใช้ระบุตำแหน่งของหุ่นในแผนที่
- base_link
คือ ส่วนที่ใช้เป็นจุดอ้างอิง
การติดของในหุ่น
(มักไว้ติดของ)



รู้ว่า Sensor อยู่ไหน : Transformation Frame Library (TF)



ROS Navigation Stack : Requirement

- SETUP ลักษณะหุ่นยนต์ของเรา (Robot Configuration)

1. TRANSFORMATION FRAME (TF)

- รู้ว่า Sensor เราติดอยู่ตรงไหน
- ถ้าอยู่ตรงไหน พื้นอยู่ตรงไหน (หุ่นบางตัวยกขา ยกล้อได้)
ต้องมีตำแหน่งอ้างกับพื้น

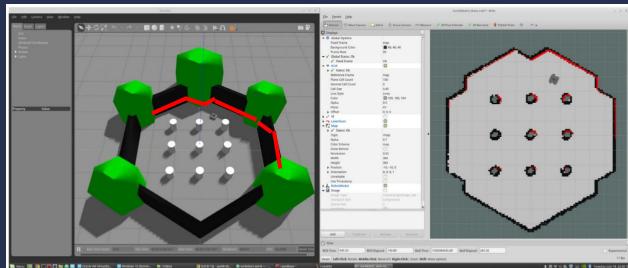


2. MAP (ข้อมูลแผนที่)

- ใช้รูปที่จะกลายมาเป็น Occupancy Grid
- อยู่ใน PART 1 นะจ๊ะ

3. ODOMETRY

- ข้อมูลจากล้อ, จากภายนอกหุ่นเอง ,
คำนวณผลลัพธ์ตำแหน่งของหุ่น แบบไม่ได้พึ่งสิ่งแวดล้อม



Real World

Occupancy Grid

ROS Navigation Stack : Requirement

- SETUP ลักษณะหุ่นยนต์ของเรา (Robot Configuration)

1. TRANSFORMATION FRAME (TF)

- รู้ว่า Sensor เราติดอยู่ตรงไหน
- ต้องอยู่ตรงไหน พื้นอยู่ตรงไหน (หุ่นบางตัวยกขา ยกล้อได้)
ต้องมีตำแหน่งอ้างกับพื้น

2. MAP (ข้อมูลแผนที่)

- ใช้รูปที่จะถ่ายมาเป็น Occupancy Grid
- อยู่ใน PART 1 นะจ๊ะ

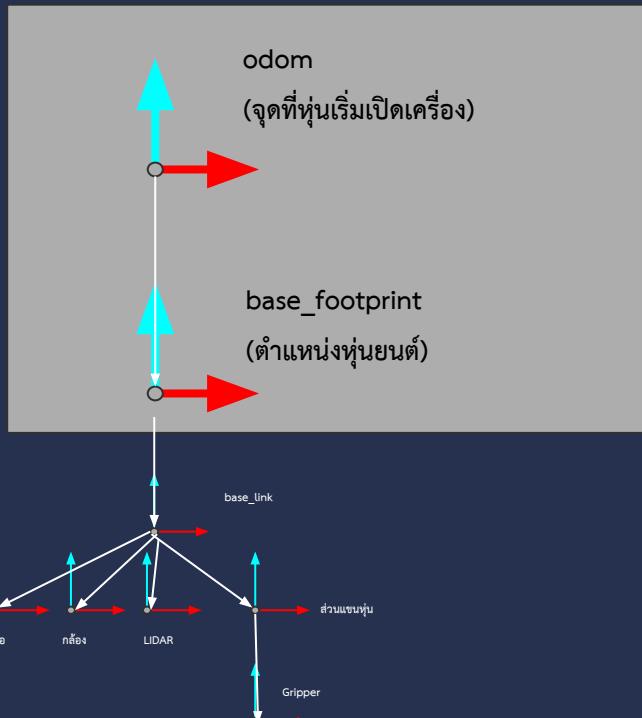
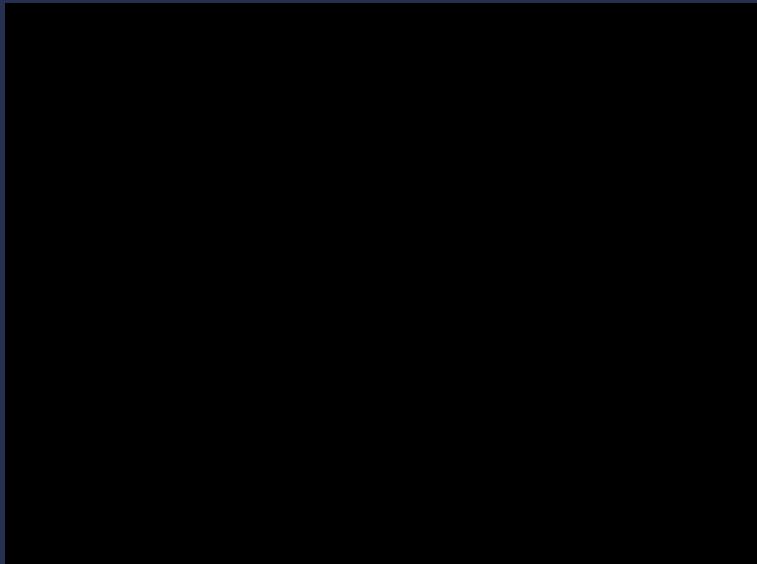
3. ODOMETRY

- ข้อมูลจากล้อ, จากภายในหุ่นเอง ,
คำนวนผลลัพธ์ตำแหน่งของหุ่น แบบไม่ได้พิงสิ่งแวดล้อม

Odometer : วัดว่าหุ่นเคลื่อนที่มาเท่าไร (โดยไม่พึงโลกภายนอก)

TF odom->base_footprint

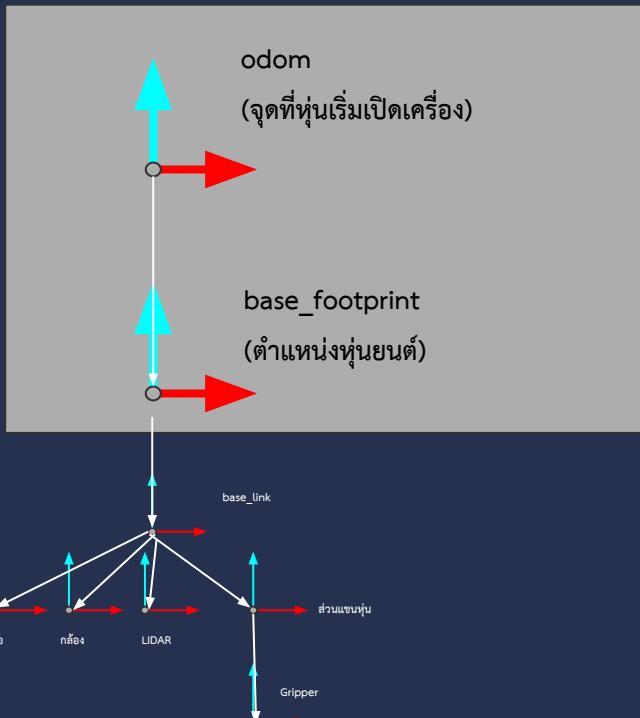
- การวัดว่าตั้งแต่เปิดเครื่องมา หุ่นไปอยู่ที่ไหน
- ไม่พึงสิ่งแวดล้อมภายนอก



Odometer : วัดว่าหุ่นเคลื่อนที่มาเท่าไร (โดยไม่พึงโลกภายนอก)

TF odom->base_footprint

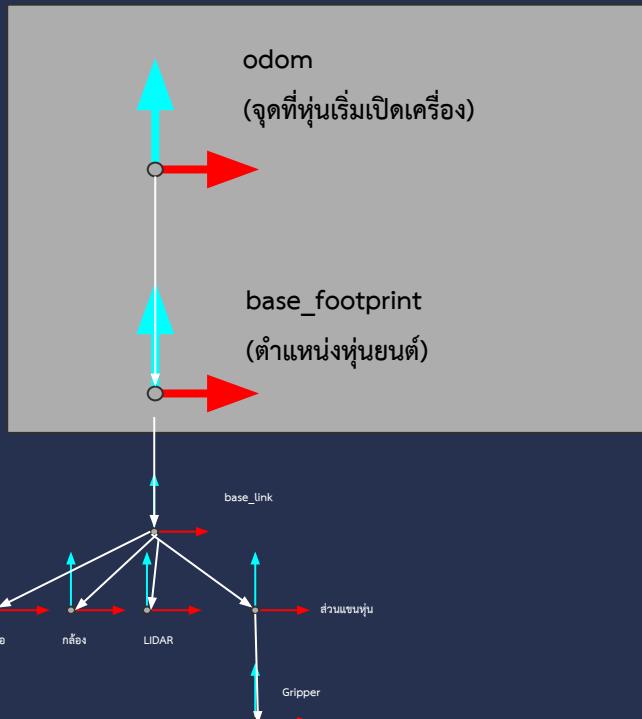
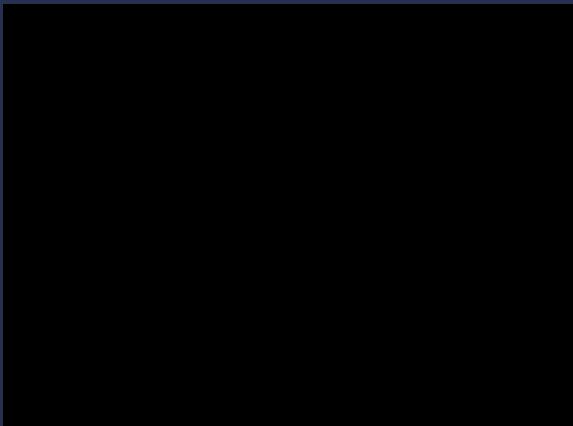
- การวัดว่าตั้งแต่เปิดเครื่องมา หุ่นไปอยู่ที่ไหน
- ไม่พึงสิ่งแวดล้อมภายนอก



Odometer : วัดว่าหุ่นเคลื่อนที่มาเท่าไร (โดยไม่พึงโลกภายนอก)

TF odom->base_footprint

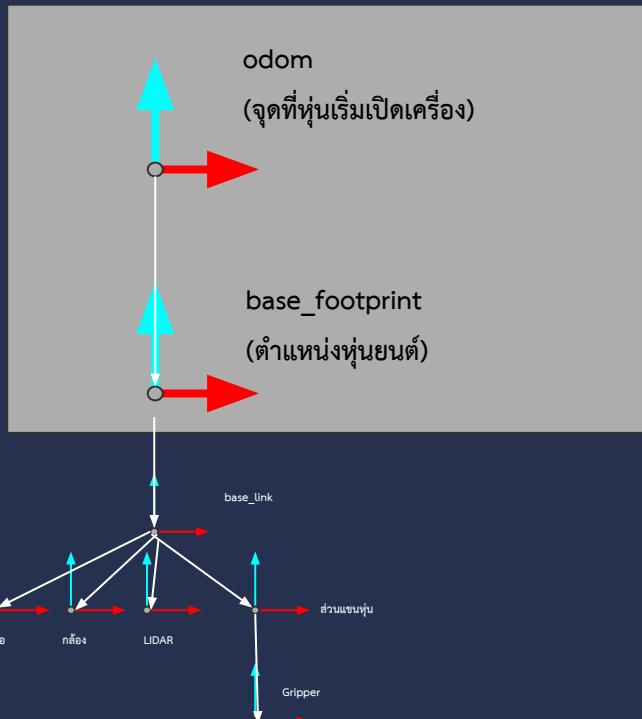
- การวัดว่าตั้งแต่เปิดเครื่องมา หุ่นไปอยู่ที่ไหน
- ไม่พึงสิ่งแวดล้อมภายนอก
- โคนหลอกได้ (เดียว localization จะมาช่วย)
 - พื้นลีน = ล้อหมุนไปข้างหน้า BUT หุ่นอยู่กับที่



Odometer : วัดว่าหุ่นเคลื่อนที่มาเท่าไร (โดยไม่พึงโลกภายนอก)

ปกติวัดยังไง ?

- Wheel Encoder ติดที่ล้อ
 - เพื่อหาว่าล้อหมุนไปเท่าไร
 - นำไปคิดสมการ Inverse Kinematics ของ Model
 - เขียน Node เพิ่มเติมมาเอง
- IMU ติดที่ตัวหุ่น
 - เพื่อหา Acceleration ของทั้งก้อนหุ่น
- ในงานนี้เราใช้ข้อมูลจาก Simulator



ROS Navigation Stack : Requirement (Formal)

- Transformation Frames ของ Sensor บันทุณยนต์
 - Required TF frame name ([REP105 - Coordinate Frames for Mobile Platforms](#))
 - TF : odom -> base_footprint
 - TF : base_footprint -> lidar_frame
- Sensor Data
 - ข้อมูล LIDAR 2D
 - Topic : /scan
 - Frame_id : "lidar_frame"
 - Odometry Data
 - Topic : /odom

มาดูกันว่า

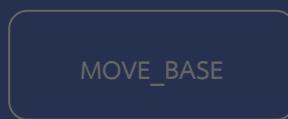
แต่ละ Node ใน Navigation Stack
ทำงานยังไงบ้าง ?

ROS Navigation Stack : Components (Nodes)

รู้ว่าตัวเองอยู่ตรงไหน



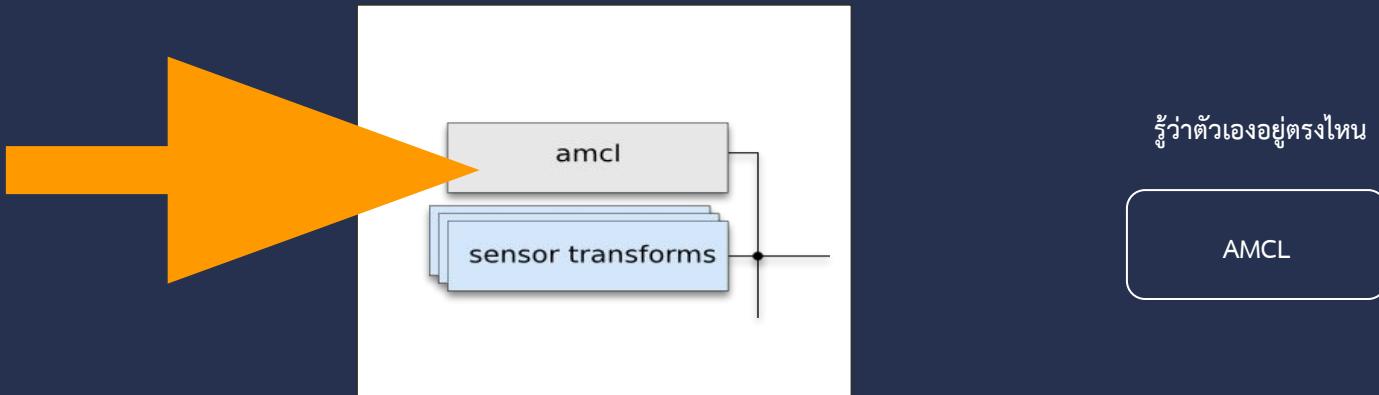
เดินไปจุดหมายด้วยตัวเอง



MAP_SERVER

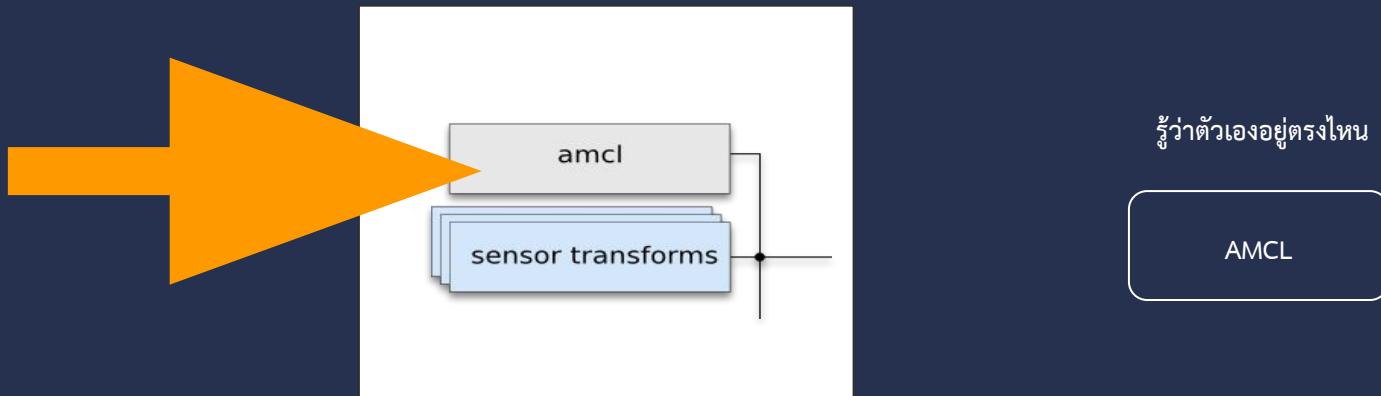
โหลดไฟล์รูป -> แผนที่

ROS Navigation Stack : AMCL



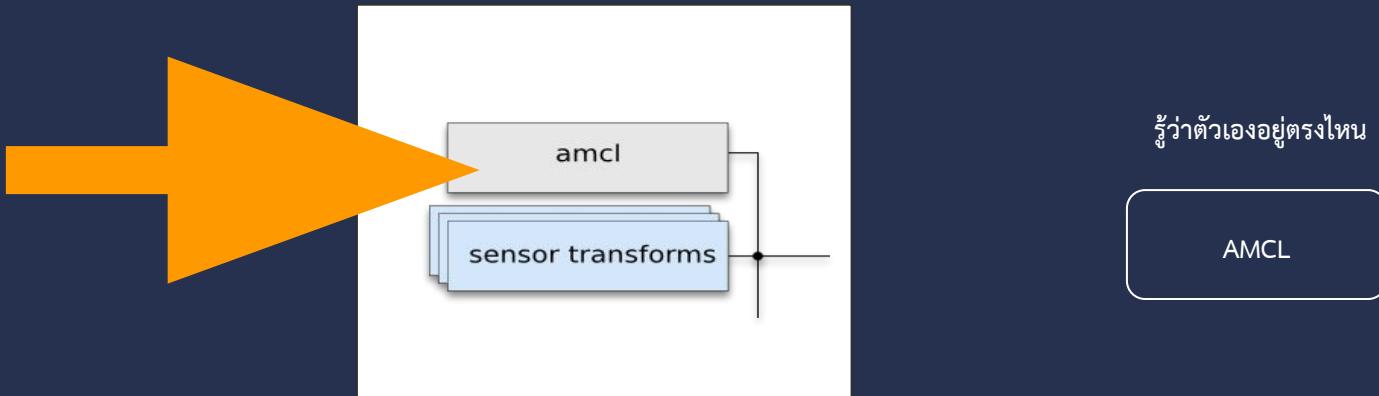
- Localization Node : โปรแกรมที่ให้ตำแหน่ง จาก จุดอ้างอิง เป้าที่นุยนต์บนโลกให้
- จุดอ้างอิง = map (แผนที่)

ROS Navigation Stack : AMCL



- Localization Node : โปรแกรมที่ให้คำแนะนำ จาก จุดอ้างอิง ไปหา ที่นี่ บนโลกได้
 - จุดอ้างอิง = map (แผนที่)
-
- Localization Node : โปรแกรมที่ให้คำแนะนำ จาก แผนที่ ไปหา ที่นี่ บนโลกได้
 - คำแนะนำที่นี่ = base_footprint

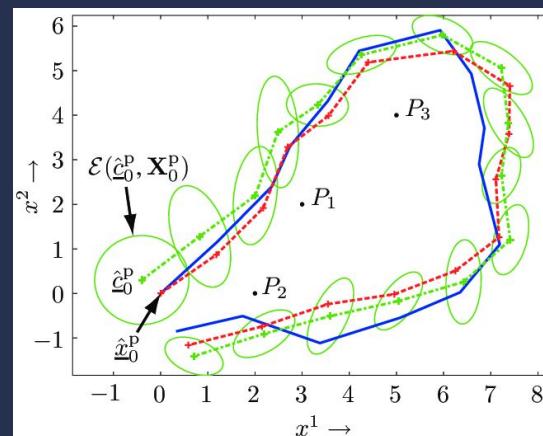
ROS Navigation Stack : AMCL



- Localization Node : โปรแกรมที่ให้คำแนะนำ จาก จุดอ้างอิง ไปหา หุ่นยนต์บนโลก ได้
 - จุดอ้างอิง = map (แผนที่)
-
- Localization Node : โปรแกรมที่ให้คำแนะนำ จาก แผนที่ ไปหา หุ่นยนต์บนโลก ได้
 - คำแนะนำหุ่นยนต์ = base_footprint
-
- Localization Node : โปรแกรมที่ให้คำแนะนำ จาก แผนที่ ไปหา base footprint ได้

AMCL : Adaptive Monte-Carlo Localization

- Localization : การหาว่าเราอยู่ตรงไหน เทียบกับแผนที่
 - Deterministic (ทำเหมือนเดิม จะได้ผลลัพธ์ เมื่อเดินทางไปแล้ว)
 - Stochastic (ใช้การสุ่มเข้ามาเกี่ยวข้อง ทำทุกอย่างแบบเดิม อาจได้ผลลัพธ์ไม่เหมือนเดิม)

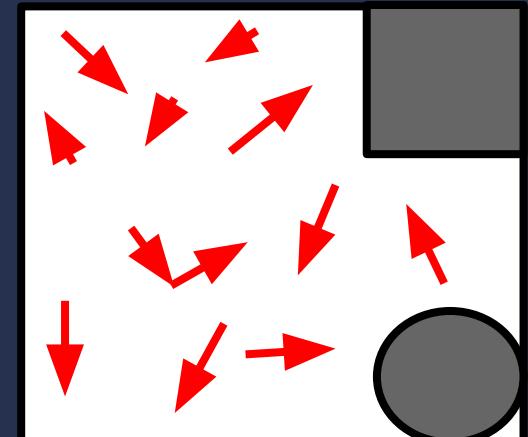


AMCL : Adaptive Monte-Carlo Localization

- Monte-Carlo เป็นบ่อนในโปรดักส์
- เป็นบ่อน = เสี่ยงโชค = ทดลองแบบสุ่มๆ (Random)
- ถ้าทำ Monte-Carlo Localization
วิธีทั่วๆไป คือ Particle Filter Localization



Monte Carlo Casino



AMCL : Adaptive Monte-Carlo Localization

Adaptive ยังไง ?

- ด้านในการสุ่ม จะมีการแยกร่างหุ่นยนต์ออกเป็นหลายๆ ตัว
= กิน Resource เยอะ (โดยเฉพาะ RAM)
- มันเลยมีวิธีการด้านใน ที่ช่วยทำให้การคำนวณลดปริมาณลง
เมื่อตำแหน่งถูกต้องแล้ว (ใช้ร่างแยกน้อยลง)

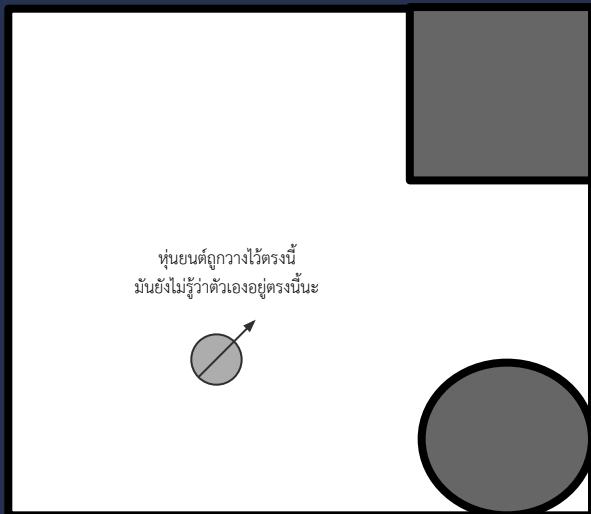
แล้ว

Particle Filter ใน AMCL

มันทำงานยังไงล่ะ ?

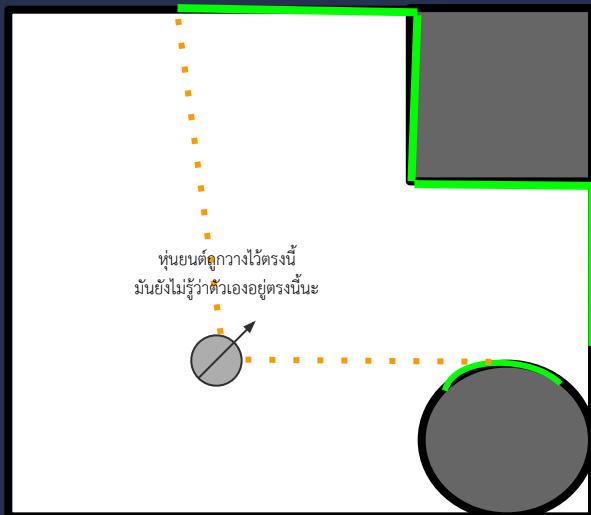
AMCL : Adaptive Monte-Carlo Localization

- เริ่มต้นจาก เรา มีแพนที่ 2D
- เรา มีลูกศร แทนตำแหน่งของหุ่นยนต์
- เรา มีข้อมูล LiDAR2D (Sensor ที่วัดอ่าน Environment)

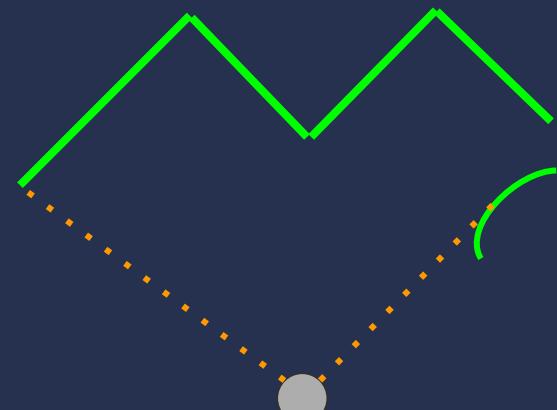


AMCL : Adaptive Monte-Carlo Localization

- เริ่มต้นจาก เรา มีแพนที่ 2D
- เรา มีลูกรศร แทนตำแหน่งของหุ่นยนต์
- เรา มีข้อมูล LiDAR2D (Sensor ที่วัดอ่าน Environment)

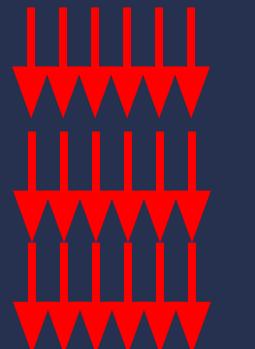
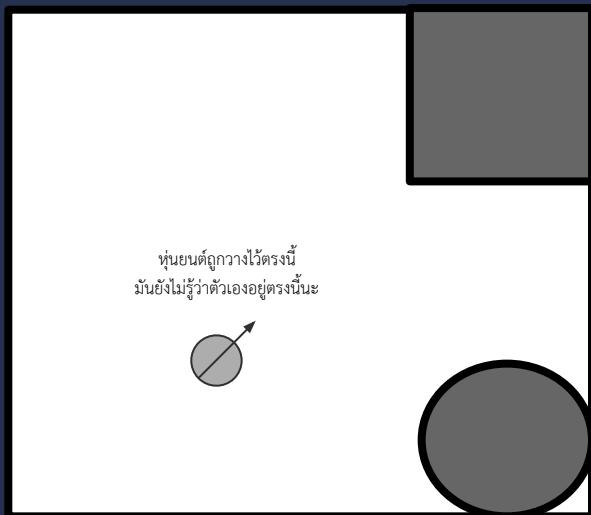


LiDAR 2D อ่านค่าได้ แบบนี้

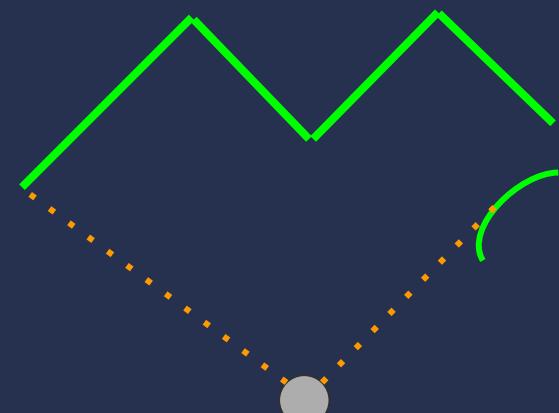


AMCL : Adaptive Monte-Carlo Localization

- เริ่มต้นจาก เรา มีแพนที่ 2D
- เรา มีลูกศร แทนตำแหน่งของหุ่นยนต์
- เรา มีข้อมูล LiDAR2D (Sensor ที่วัดอ่าน Environment)



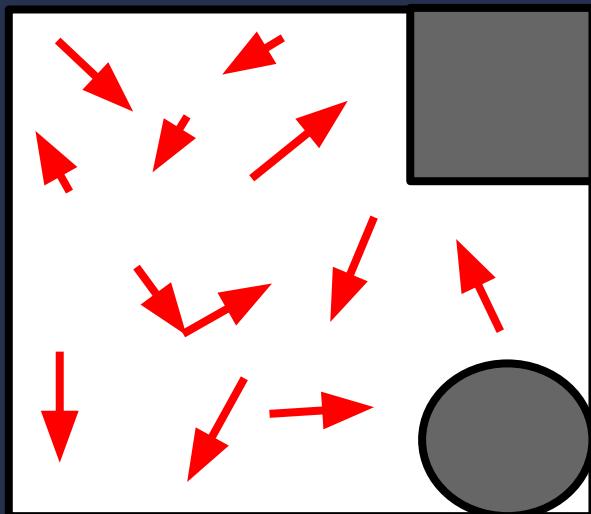
1 ลูกศร
แทนตำแหน่งของหุ่นยนต์



LIDAR 2D อ่านค่าได้ แบบนี้

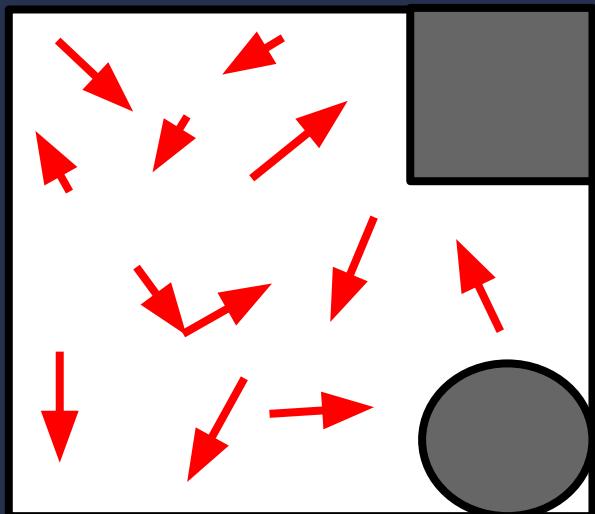
AMCL : Adaptive Monte-Carlo Localization

- เราไม่รู้ตำแหน่งเริ่มต้น
- เริ่มทำการประยุกต์ลงไปที่ว่างๆ สุ่มๆ ทั่วๆ
- ลองคิดว่าตัวเองเป็นลูกศรนั้น
 - เราควรจะเห็นอะไร
 - แล้วตอนนี้อ่าน Sensor (LiDAR 2D) และเห็นอะไร
 - มันเหมือนแผนที่ ที่เราคิดไว้ไหม ? ให้คะแนน !
- ลูกศรที่มีคะแนนเยอะ รอบต่อไป
เราจะมาประยุกต์เฉพาะนั้นเยอะขึ้น
(ก็ตรงนั้นดูเหมือนจะถูกนิ ก็มั่วไปทางถูกๆ ใจ 555)



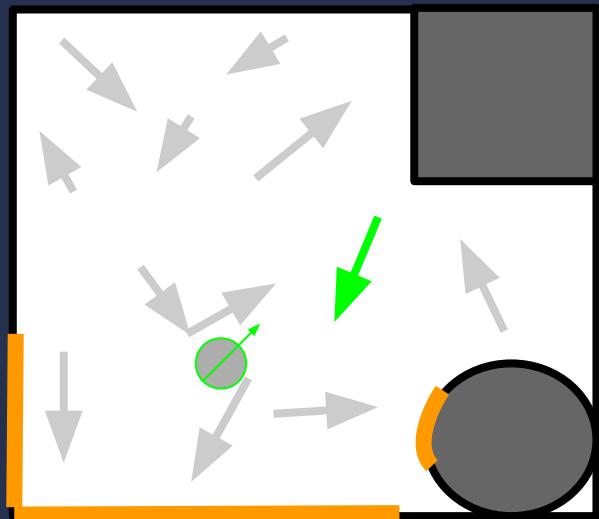
AMCL : Adaptive Monte-Carlo Localization

- เราไม่รู้ตำแหน่งเริ่มต้น
- เริ่มทำการประยุกต์ลงไปที่ว่างๆ สุ่มๆ ทั่วๆ
- ลองคิดว่าตัวเองเป็นลูกศรนั้น
 - เราควรจะเห็นอะไร
 - แล้วตอนนี้อ่าน Sensor (LiDAR 2D) และเห็นอะไร
 - มันเหมือนแผนที่ ที่เราคิดไว้ไหม ? ให้คะแนน !
- ลูกศรที่มีคะแนนเยอะ รอบต่อไป
เราจะมาประยุกต์เฉพาะนั้นเบื่องขึ้น
(ก็ตรงนั้นดูเหมือนจะถูกนิ ก็มัวไปทางถูกๆ ใจ 555)

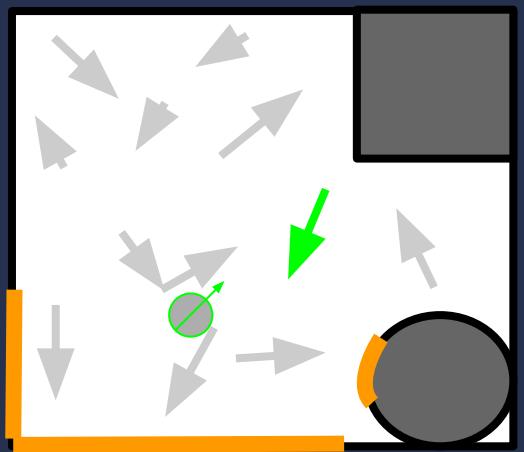


AMCL : Adaptive Monte-Carlo Localization

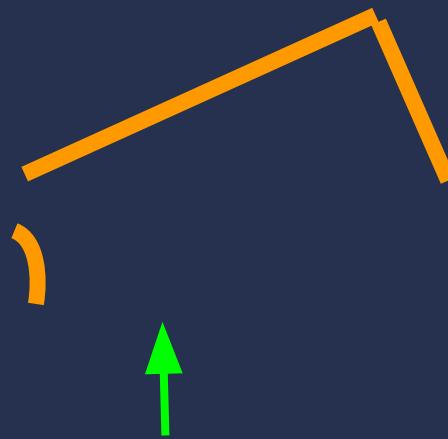
- เราไม่รู้ตำแหน่งเริ่มต้น
- เริ่มทำการประยุกศรถลงไปที่ว่างๆ สุ่มๆ ทั่วๆ
- ลองคิดว่าตัวเองเป็นลูกศรนั้น
 - เราควรจะเห็นอะไร
 - แล้วตอนนี้อ่าน Sensor (LiDAR 2D) และเห็นอะไร
 - มันเหมือนแผนที่ ที่เราคิดไว้ไหม ? ให้คะแนน !
- ลูกศรที่มีคะแนนเยอะ รอบต่อไป
เราจะมาประยุกศรถวนนั้นเยอะขึ้น
(ก็ต้องนั่นดูเหมือนจะถูกนิ ก็มัวไปทางถูกๆ ใจ 555)



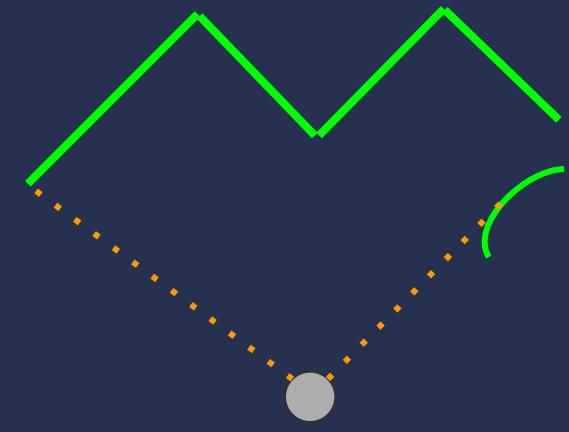
AMCL : Adaptive Monte-Carlo Localization



ที่ลูกศรคิด



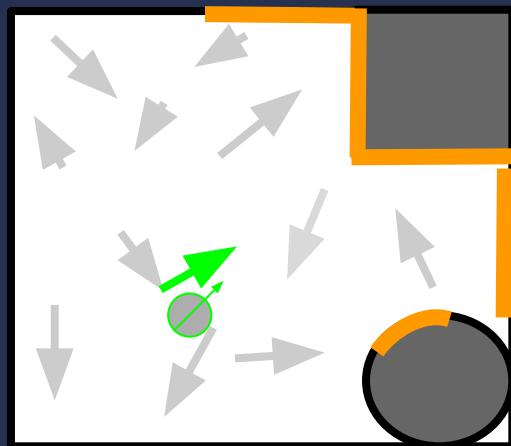
ที่ลูกศรคิด



ที่อ่านได้จริง

มัน MATCH ไหม ? = ดูແຍ່ນະ
ໃຫ້ຄະແນນນ້ອຍๆ

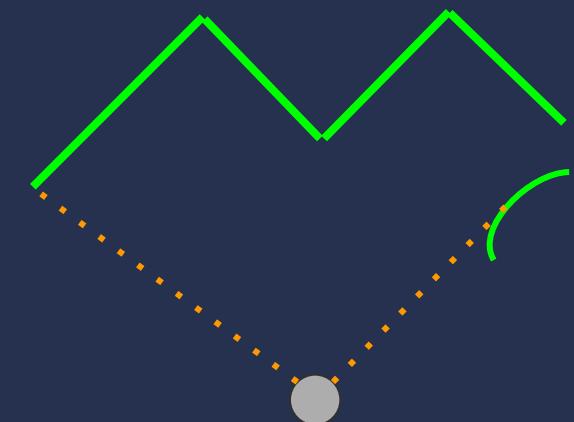
AMCL : Adaptive Monte-Carlo Localization



ที่ลูกศรคิด



ที่ลูกศรคิด

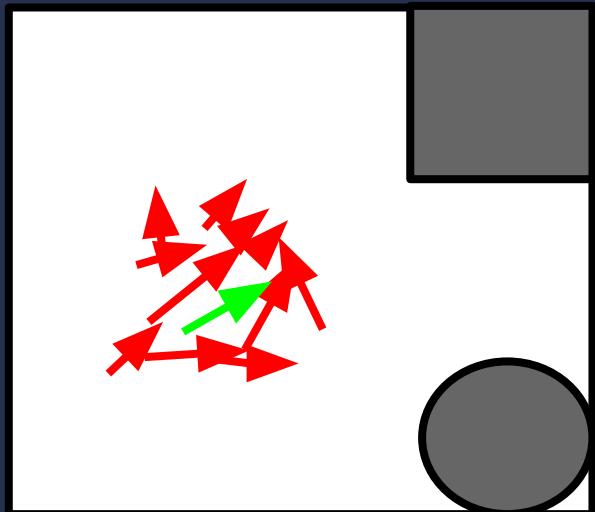


ที่อ่านได้จริง

มัน MATCH ใหม่ ? = ดูดีนะ
ให้คะแนนเยอะ

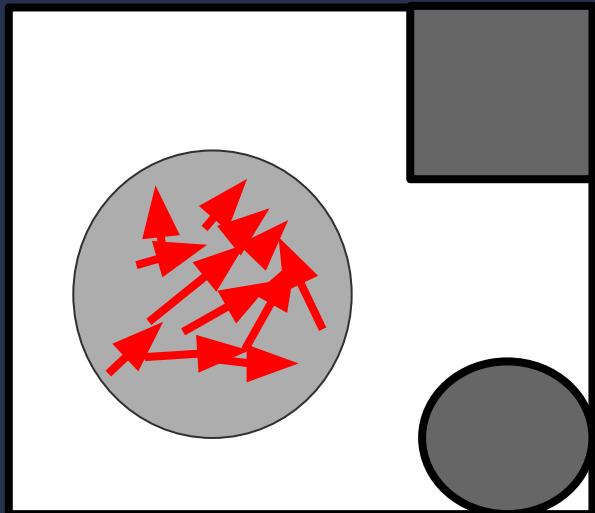
AMCL : Adaptive Monte-Carlo Localization

- เราไม่รู้ตำแหน่งเริ่มต้น
- เริ่มทำการประยุกศรลงไปที่ว่างๆ สุ่มๆ ทั่วๆ
- ลองคิดว่าตัวเองเป็นลูกศรนั้น
 - เราควรจะเห็นอะไร
 - แล้วตอนนี้อ่าน Sensor (LiDAR 2D) และเห็นอะไร
 - มันเหมือนแผนที่ ที่เราคิดไว้ไหม ? ให้คะแนน !
- ลูกศรที่มีคะแนนเยอะ รอบต่อไป
เราจะมาประยุกศรแล้วนั้นเยอะขึ้น
(ก็ตรงนั้นดูเหมือนจะถูกนิ ก็มั่วไปทางถูกๆ ใจ 555)



AMCL : Adaptive Monte-Carlo Localization

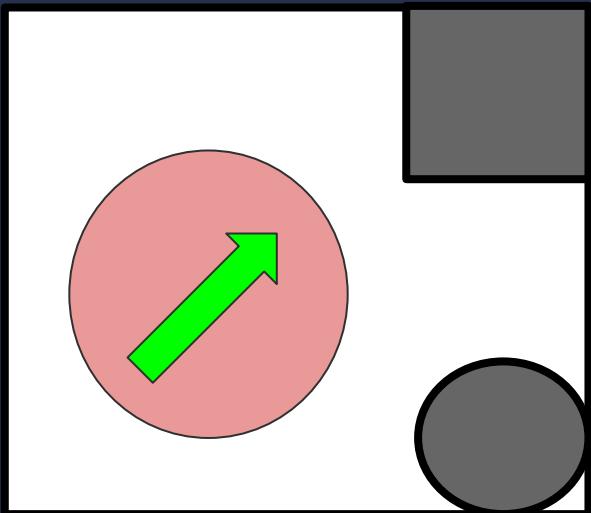
- เราไม่รู้ตำแหน่งเริ่มต้น
- เริ่มทำการประยุกศรลงไปที่ว่างๆ สุ่มๆ ทั่วๆ
- ลองคิดว่าตัวเองเป็นลูกศรนั้น
 - เราควรจะเห็นอะไร
 - แล้วตอนนี้อ่าน Sensor (LiDAR 2D) และเห็นอะไร
 - มันเหมือนแผนที่ ที่เราคิดไว้ไหม ? ให้คะแนน !
- ลูกศรที่มีคะแนนเยอะ รอบต่อไป
เราจะมาประยุกศรแล้วนั้นเยอะขึ้น
(ก็ตรงนั้นดูเหมือนจะถูกนิ ก็มั่วไปทางถูกๆ ใจ 555)



เริ่มรอบใหม่

AMCL : Adaptive Monte-Carlo Localization

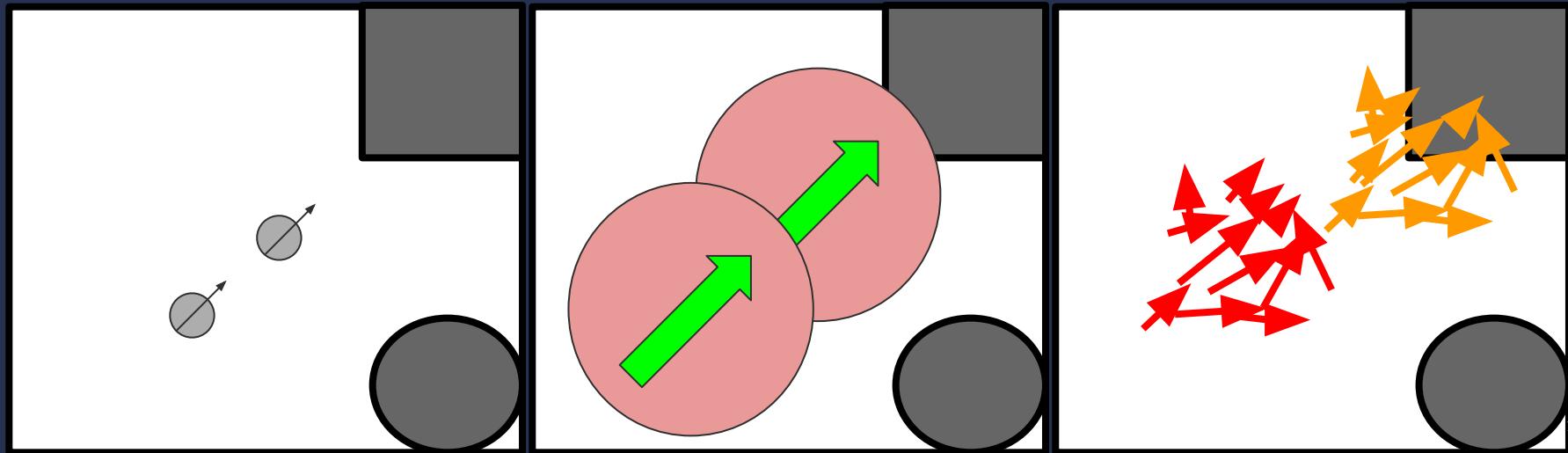
- เราไม่รู้ตำแหน่งเริ่มต้น
- เริ่มทำการประยุกต์ลงไปที่ว่างๆ สุ่มๆ ทั่วๆ
- ลองคิดว่าตัวเองเป็นลูกศรนั้น
 - เราควรจะเห็นอะไร
 - แล้วตอนนี้อ่าน Sensor (LiDAR 2D) และเห็นอะไร
 - มันเหมือนแผนที่ ที่เราคิดไว้ไหม ? ให้คะแนน !
- ลูกศรที่มีคะแนนเยอะ รอบต่อไป
เราจะมาประยุกต์แล้วนั้นเยอะขึ้น
(ก็ตรงนั้นดูเหมือนจะถูกนิ ก็มั่วไปทางถูกๆ ใจ 555)



ແຄວ່ານີ້ແລະມັງ
ສືເຂົ້າວິກຶດຕອບ

AMCL : Adaptive Monte-Carlo Localization

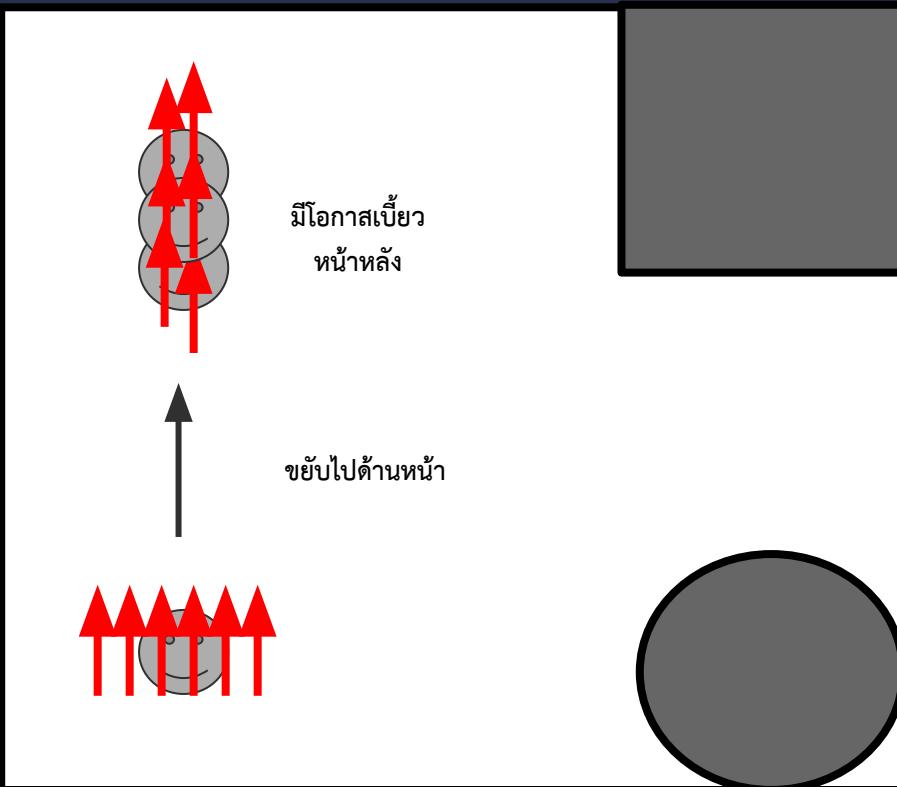
- เราเริ่มขยับหุ่นยนต์
- Odometry ไม่แม่นเสมอไป = มี Noise ใน odometry
- เวลาเราขยับ Particle จะขยับตามด้วยทีละทั้งหมด
- เราจะ Add Noise ให้กับ กลุ่ม Particle ทุกอันด้วย



AMCL : Adaptive Monte-Carlo Localization

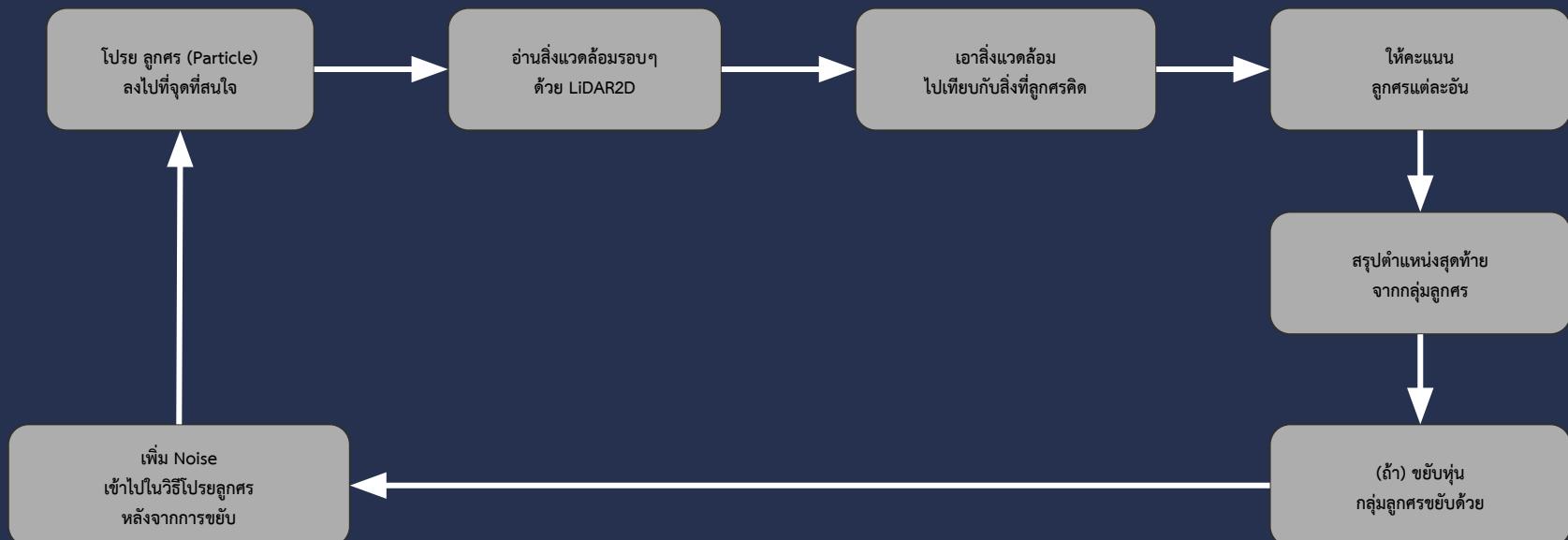
ODOMETRY NOISE

- เกิดจากการเคลื่อนที่ของหุ่นไม้ได้แม่นยำ
- AMCL สามารถช่วยจำลองความไม่แม่น
(ເຝືອຖາໄວ່)
 - หุ่นยนต์ชอบเดินเลย แนวหน้าหลัง
 - กินซ้าย กินขวา
 - หมุนเกิน หมุนขาด
- เราสามารถปรับ Parameter พากนີ້ได້
 - `odom_alpha`
- ວັນພຽງນີ້ຈະມາເຈາະລຶກອີກທີ່



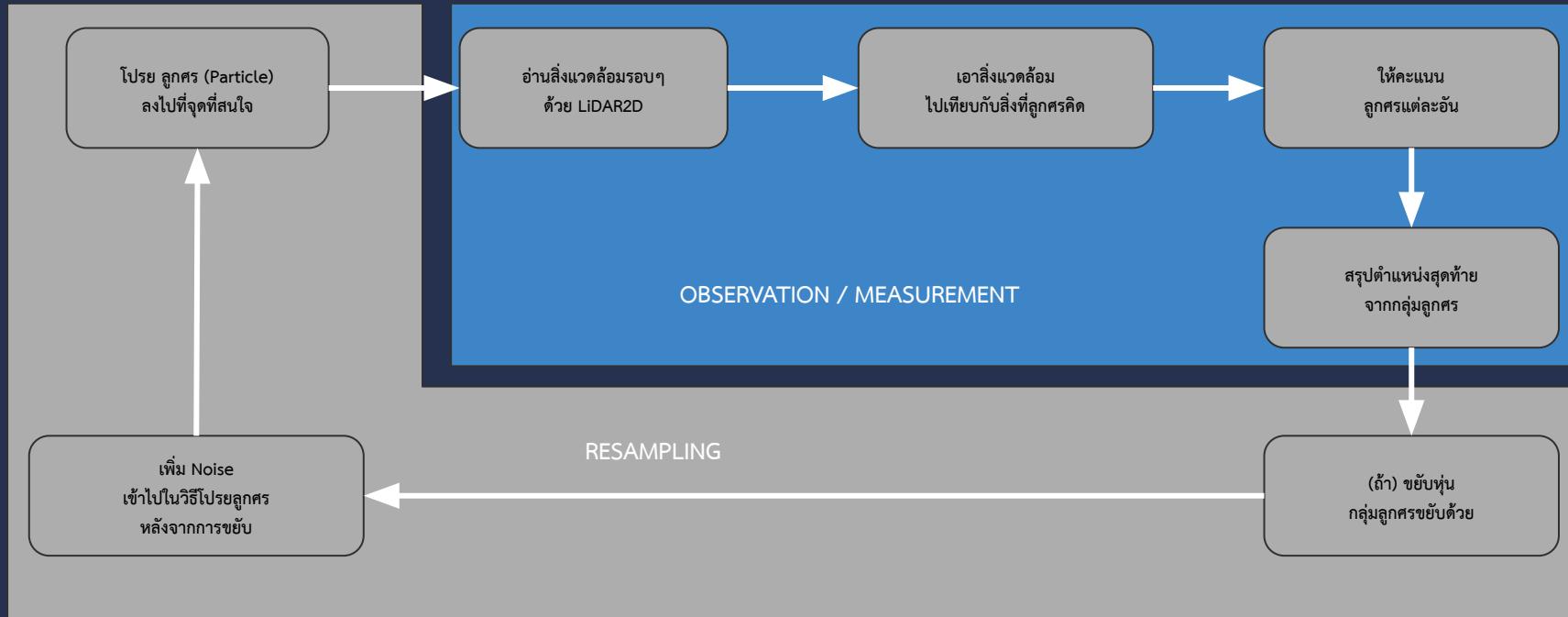
AMCL : Adaptive Monte-Carlo Localization

ໂອເຄ ສຽບໜັນຕອນຄວາມຮັບຮັດໃຫ້ດັ່ງນີ້

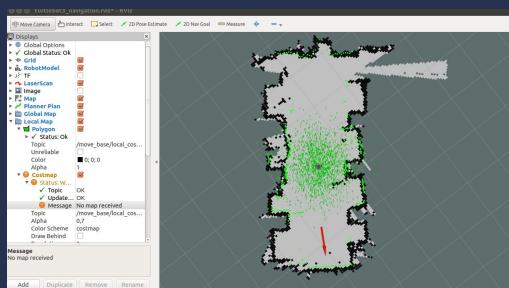


AMCL : Adaptive Monte-Carlo Localization

ໂອເຄ ສຽບໜັນຕອນຄວາມຮັບຮັດໃຫ້ດັ່ງນີ້ (ອ້າງອີງວິຊາການນິດນິ້ງ)

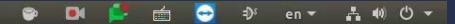


AMCL : Adaptive Monte-Carlo Localization



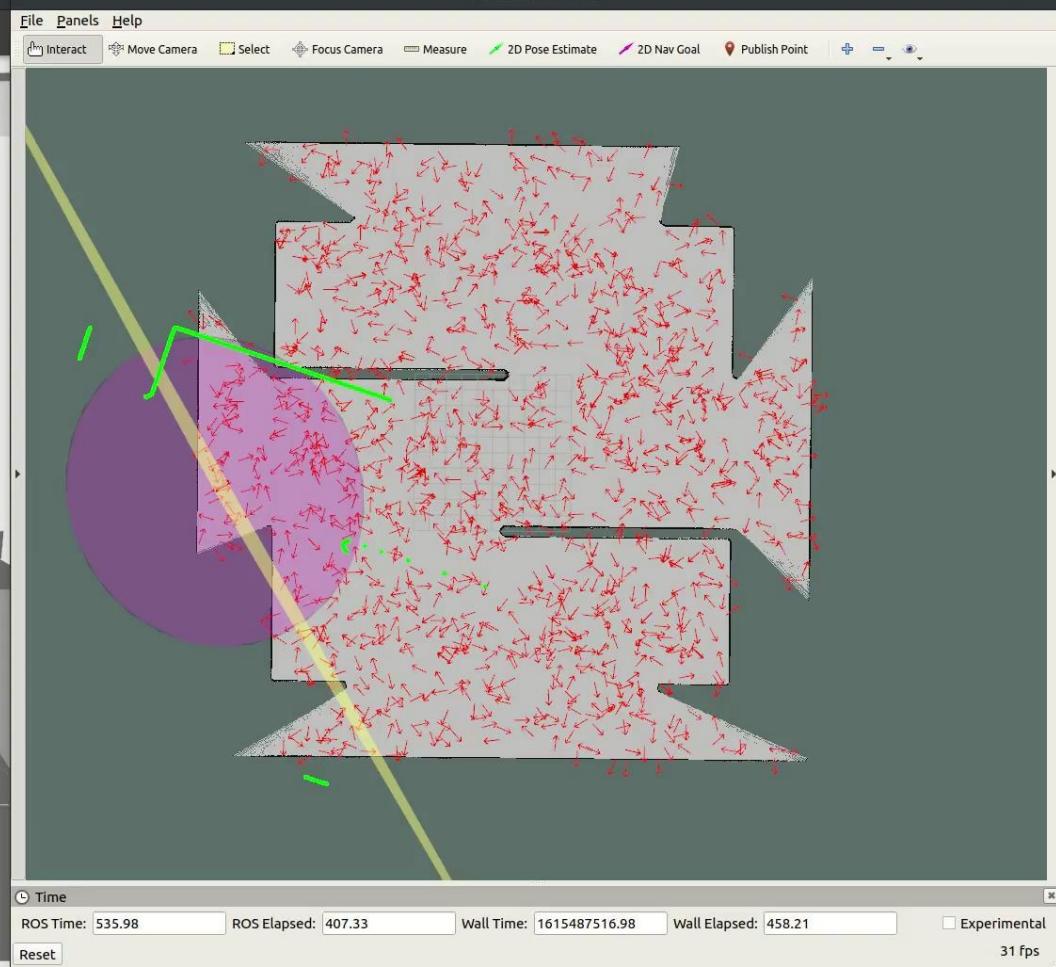
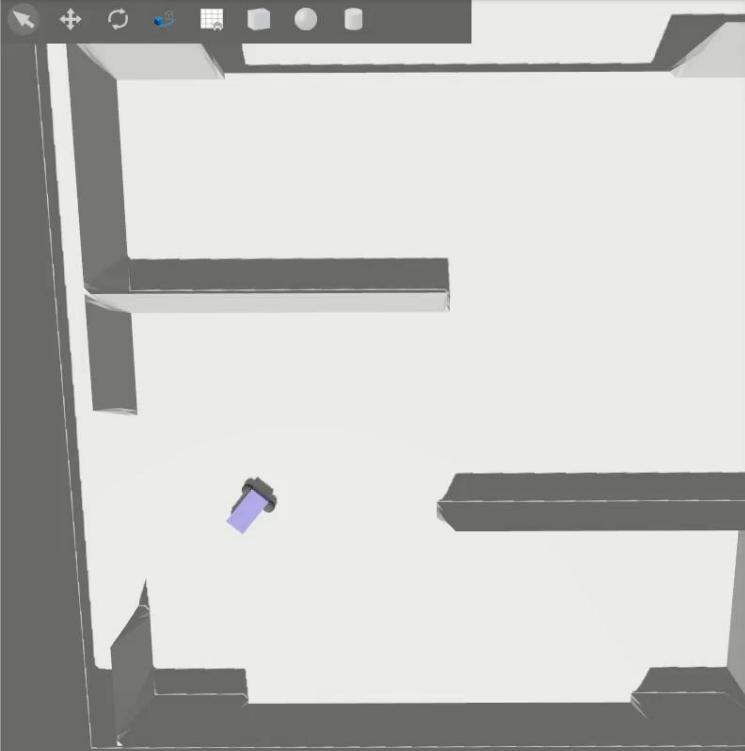
Activities Terminal

Fri Mar 12, 01:31



Gazebo

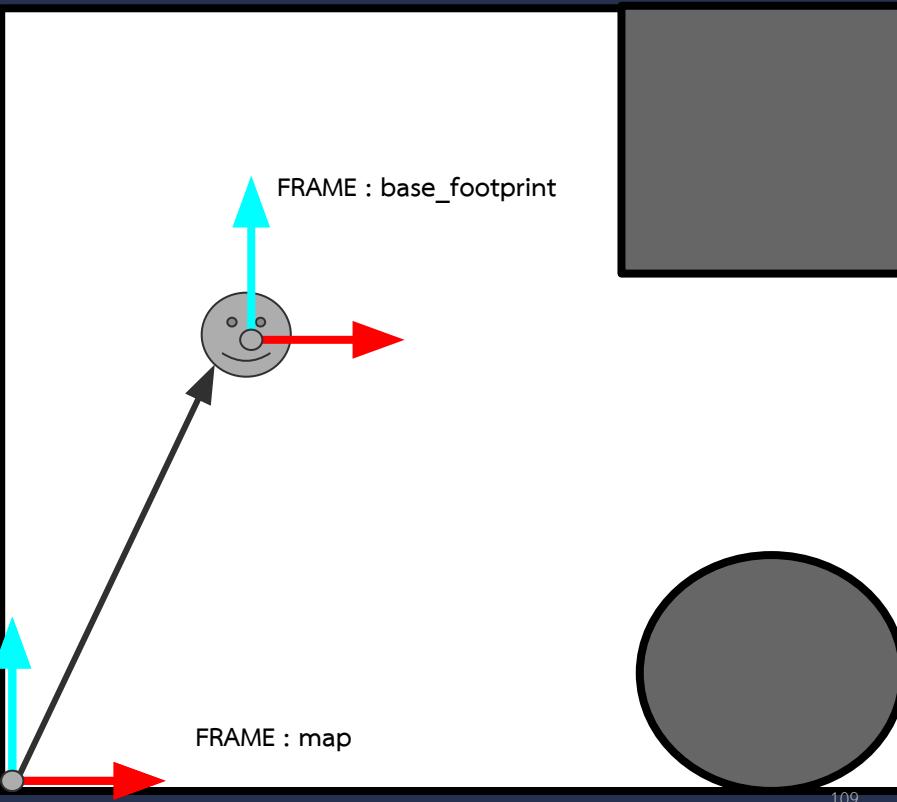
Gazebo



AMCL : Adaptive Monte-Carlo Localization

ที่น่าเชื่อถือ ผลลัพธ์ของ AMCL ตาม algorithm
จะได้ Robot Position ใน Map

ซึ่งควรจะเป็น TF : map -> base_footprint



AMCL : Adaptive Monte-Carlo Localization

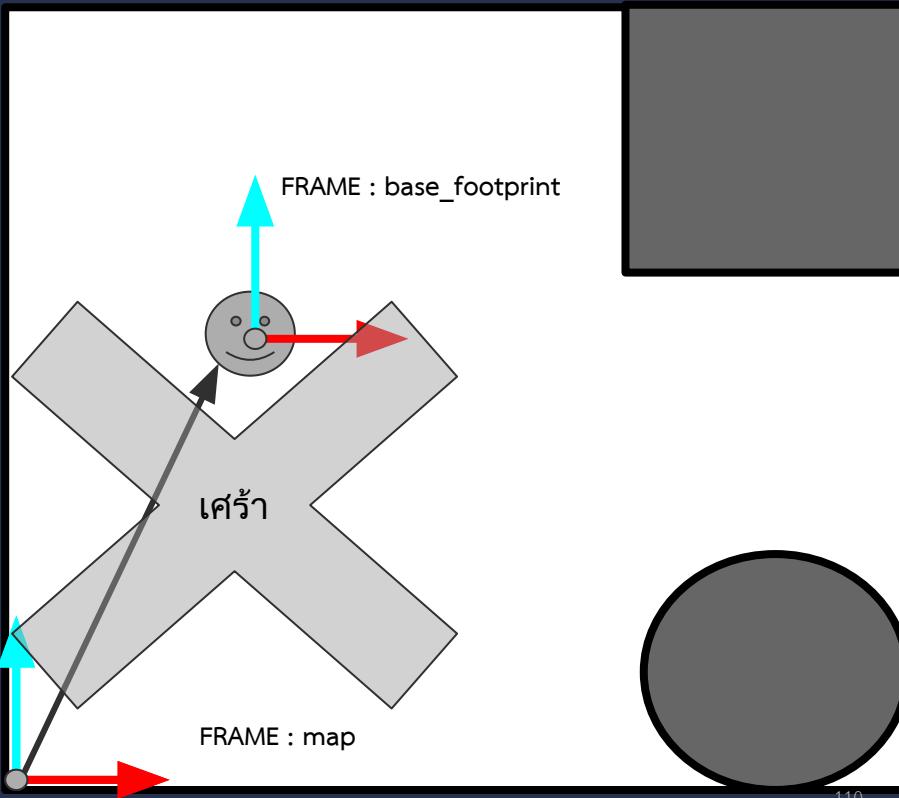
ที่น่าเชื่อถือ ผลลัพธ์ของ AMCL ตาม algorithm
จะได้ Robot Position ใน Map

ซึ่งควรจะเป็น TF : map -> base_footprint

แต่ เราทำแบบนี้ไม่ได้ (ผิดTF)

เพราะ base_footprint มี parent เป็น odom มาก่อนหน้านี้แล้ว
(Odometry)

amcl จึงขออาสาให้ผลลัพธ์แบบ
Correction Link แทน



AMCL : Adaptive Monte-Carlo Localization

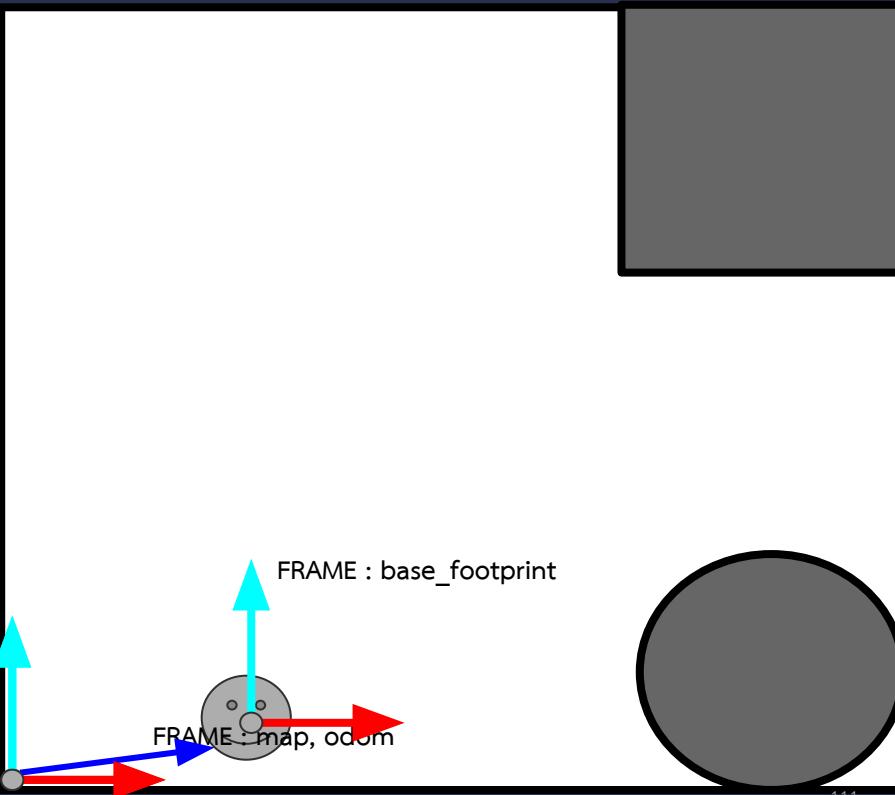
ที่น่าเชื่อถือ ผลลัพธ์ของ AMCL ตาม algorithm
จะได้ Robot Position ใน Map

ซึ่งควรเป็น TF : map -> base_footprint

แต่ เราทำแบบนี้ไม่ได้ (ผิดTF)

เพราะ base_footprint มี parent เป็น odom มาก่อนหน้านี้แล้ว
(Odometry)

amcl จึงขออาสาให้ผลลัพธ์แบบ
Correction Link แทน



AMCL : Adaptive Monte-Carlo Localization

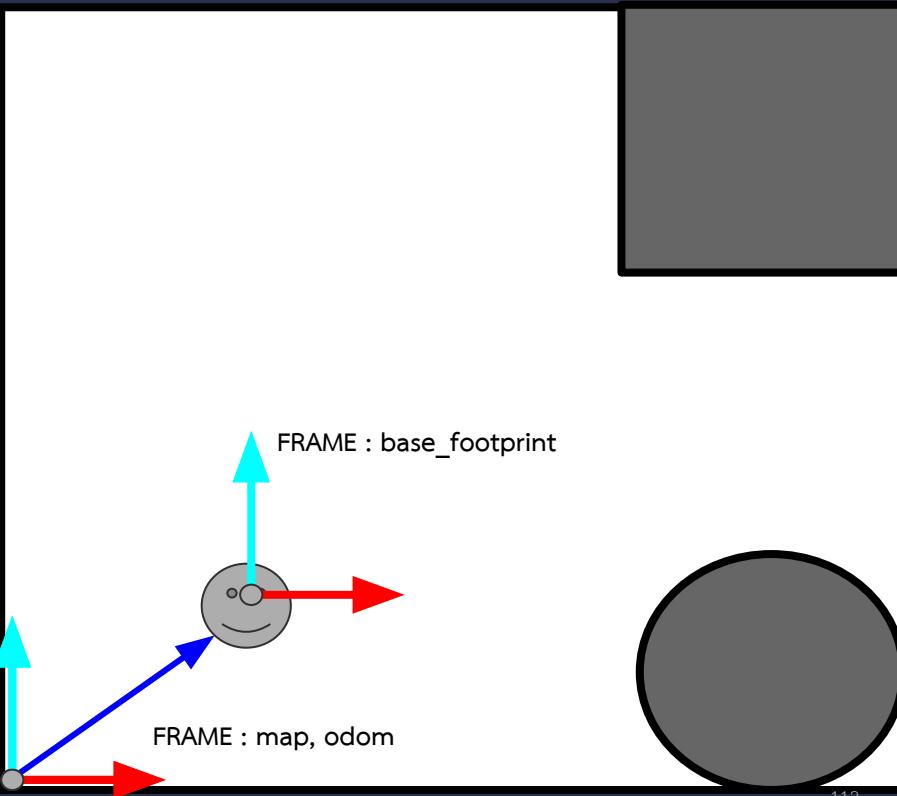
ที่น่าเชื่อถือ ผลลัพธ์ของ AMCL ตาม algorithm
จะได้ Robot Position ใน Map

ซึ่งควรเป็น TF : map -> base_footprint

แต่ เราทำแบบนี้ไม่ได้ (ผิดTF)

เพราะ base_footprint มี parent เป็น odom มาก่อนหน้านี้แล้ว
(Odometry)

amcl จึงขออาสาให้ผลลัพธ์แบบ
Correction Link แทน



AMCL : Adaptive Monte-Carlo Localization

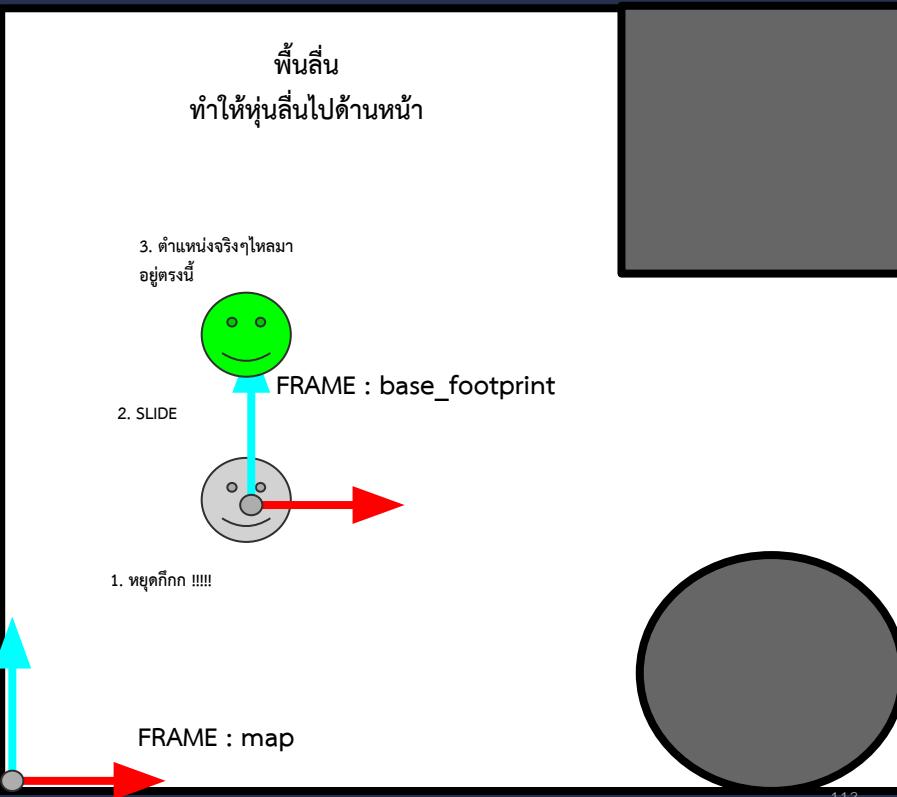
ที่น่าเศร้าคือ ผลลัพธ์ของ AMCL ตาม algoritm
จะได้ Robot Position ใน Map

ซึ่งควรจะเป็น TF : map -> base_footprint

แต่ เราทำแบบนี้ไม่ได้ (ผิดTF)

เพราะ base_footprint มี parent เป็น odom มาก่อนหน้านี้แล้ว
(Odometry)

amcl จึงขออาสาให้ผลลัพธ์แบบ
Correction Link แทน



AMCL : Adaptive Monte-Carlo Localization

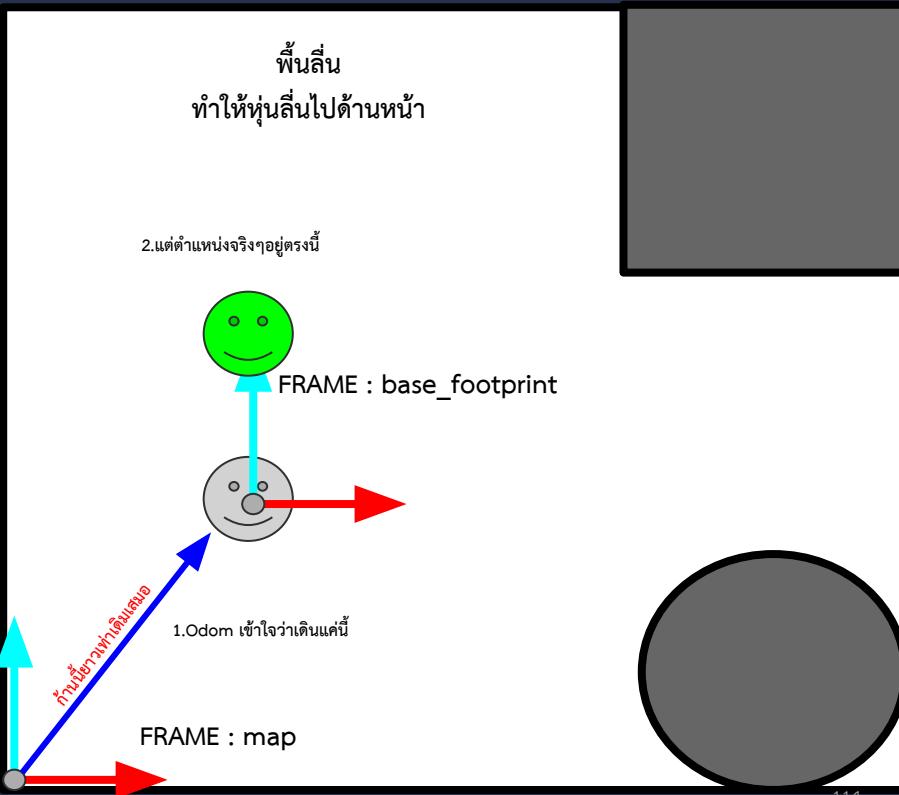
ที่นำเคร้าคือ ผลลัพธ์ของ AMCL ตาม algorithm
จะได้ Robot Position ใน Map

ซึ่งควรจะเป็น TF : map -> base footprint

แต่ เราทำแบบนั้นไม่ได้ (ผิดTF)

เพรา base_footprint มี parent เป็น odom มา ก่อนหน้านี้แล้ว
(Odometry)

amcl จึงขออาสาให้ผลลัพธ์แบบ Correction Link แทน



AMCL : Adaptive Monte-Carlo Localization

ที่น่าเชื่อถือ ผลลัพธ์ของ AMCL ตาม algorithm
จะได้ Robot Position ใน Map

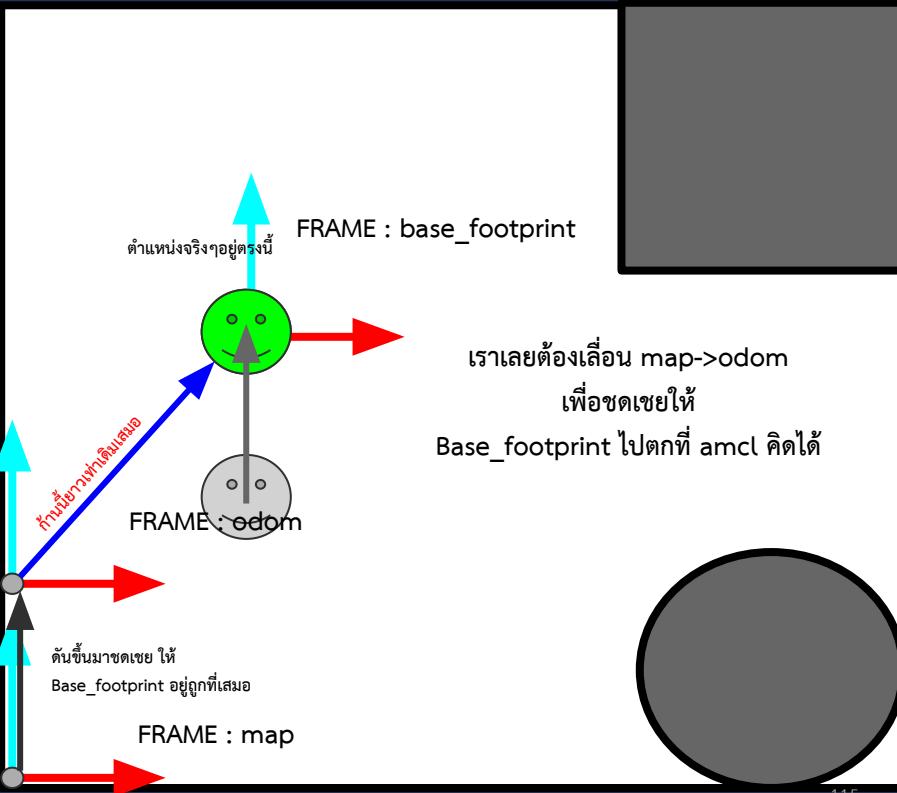
ซึ่งควรจะเป็น TF : map -> base_footprint

แต่ เราทำแบบนั้นไม่ได้ (ผิด TF)

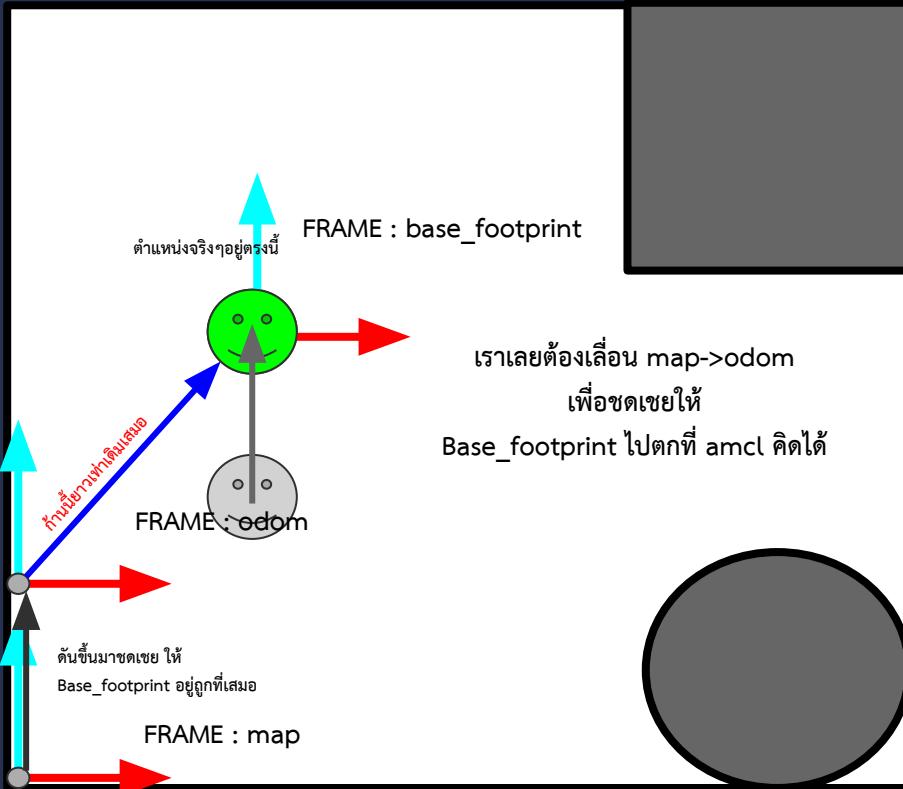
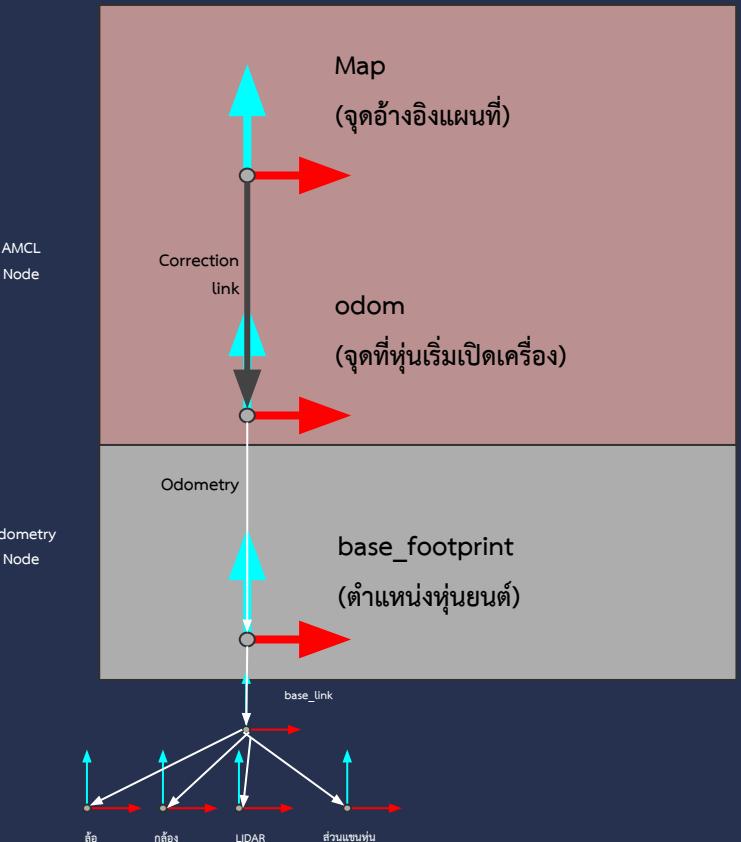
เพราะ base_footprint มี parent เป็น odom มาก่อนหน้านี้แล้ว
(Odometry)

amcl จึงขออาสาให้ผลลัพธ์แบบ
Correction Link แทน----->

= map->odom



AMCL : Adaptive Monte-Carlo Localization



ROS Navigation Stack : Components (Nodes)

รู้ว่าตัวเองอยู่ตรงไหน

AMCL

เดินไปจุดหมายด้วยตัวเอง

MOVE_BASE

MAP_SERVER

โหลดไฟล์รูป -> แผนที่

move_base : พระเอกของงาน Node ที่ทำให้หุ่นเดินได้อ่อง !

การทำงานจะประกอบด้วย 2 ส่วนคือ

1. วางแผนการเดิน แบบ GLOBAL (Global Planning)

- จากแผนที่โลก เราจะเดินไปทางไหนดี ที่สั้นที่สุด
- บัน Global costmap

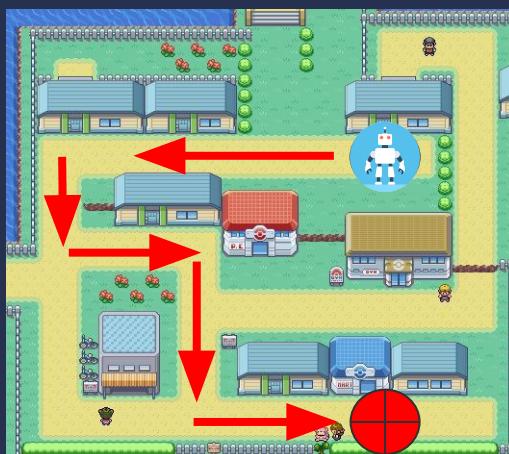
2. วางแผนการเดิน แบบ LOCAL (Local Planning)

- จะรีบตาม Global ยังไงดีไม่ให้ชนข้าวของ
- บัน Local costmap

วางแผนการเดินบนไหน ? - บัน COSTMAP ! - แยกกันทำไม่

เดินไปจุดหมายด้วยตัวเอง

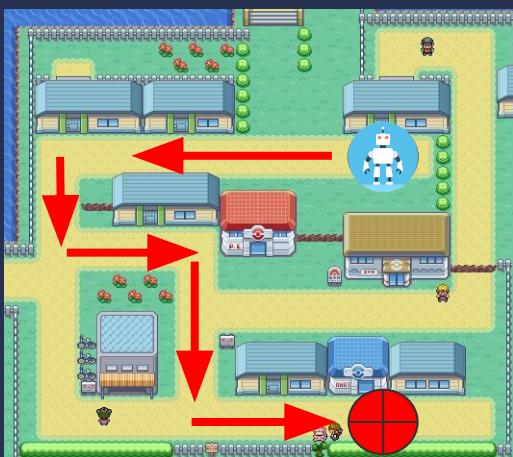
MOVE_BASE



move_base : พระเอกของงาน Node ที่ทำให้หุ่นเดินได้อ่อง !

วางแผนการเดิน แบบ GLOBAL (Global Planning)

- จากแผนที่โลก เราจะเดินไปทางไหนดี ที่สั้นที่สุด
- บัน Global costmap



วางแผนการเดิน แบบ LOCAL (Local Planning)

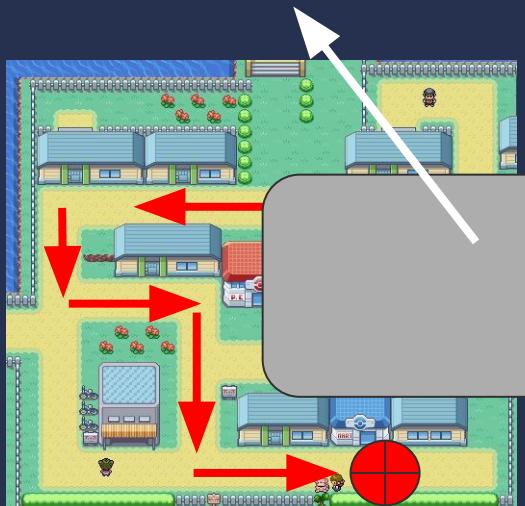
- จะวิ่งตาม Global ยังไงดีไม่ให้ชนข้าวของ
- บัน Local costmap



move_base : พระเอกของงาน Node ที่ทำให้หุ่นเดินได้อ่อง !

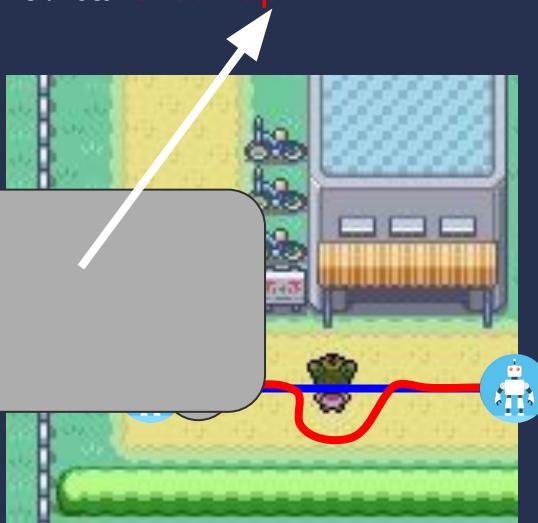
วางแผนการเดิน แบบ GLOBAL (Global Planning)

- จากแผนที่โลก เราจะเดินไปทางไหนดี ที่สั้นที่สุด
- บน Global **costmap**



วางแผนการเดิน แบบ LOCAL (Local Planning)

- จะวิ่งตาม Global ยังไงดีไม่ให้ชนข้าวของ
- บน Local **costmap**



COSTMAP คืออะไร ?

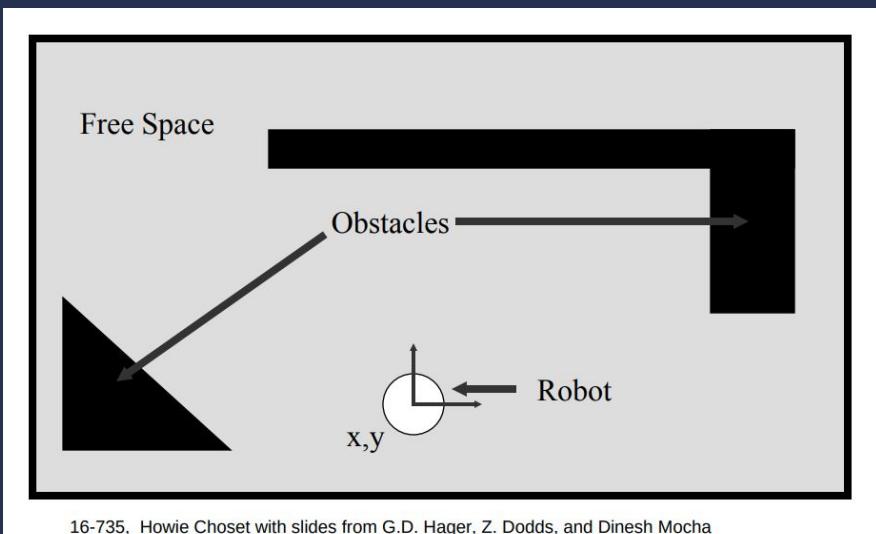
move_base : วางแผนการเดินทาง

Configuration Space

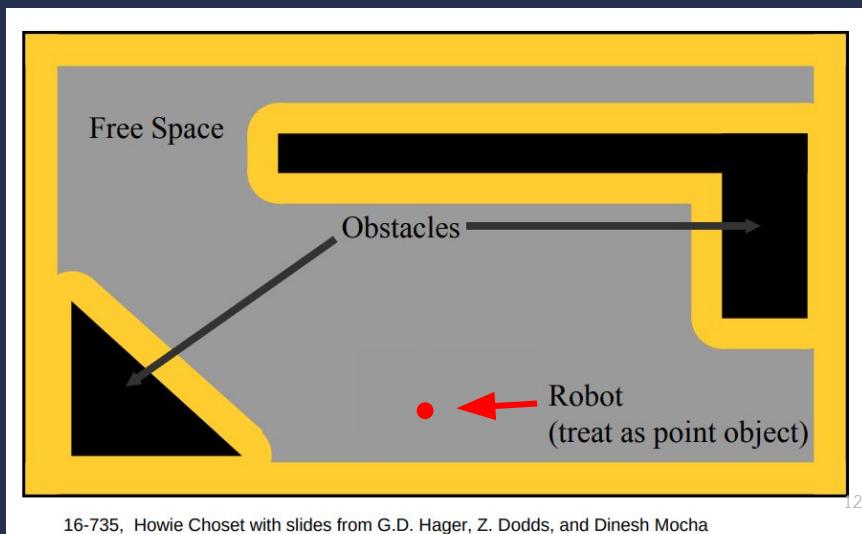
เราengกับการคำนวณจุดมากๆ

เพราะฉะนั้น เราจะเปลี่ยนโลกของหุ่นยนต์ให้กลายเป็นจุด

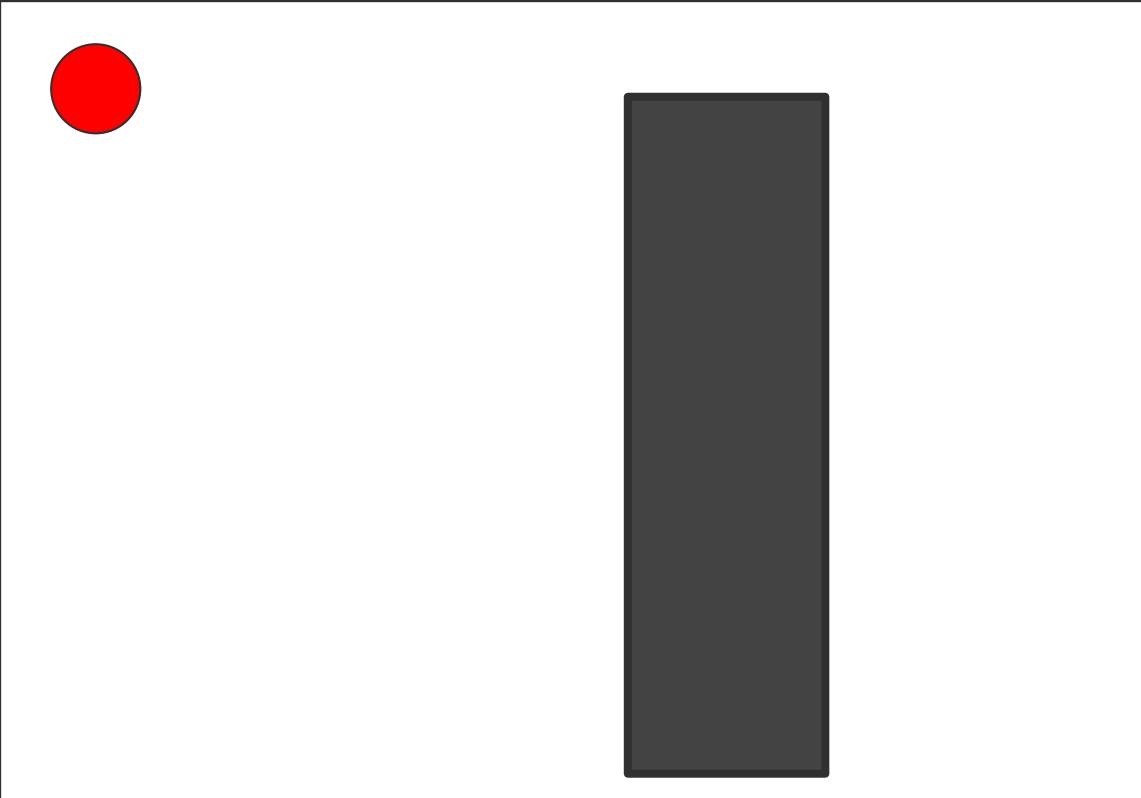
และมองสิ่งกีดขวางให้ใหญ่ขึ้น จะได้คำนวณง่ายๆ

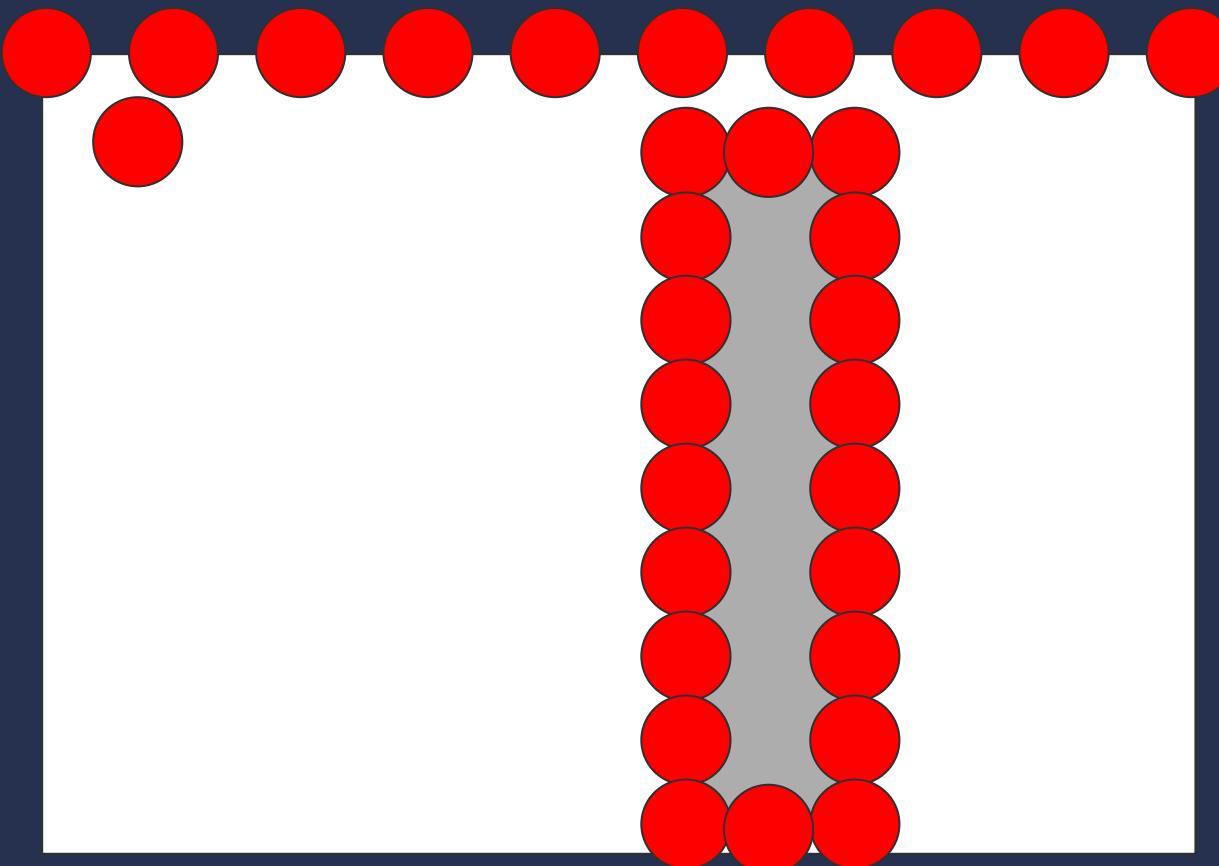


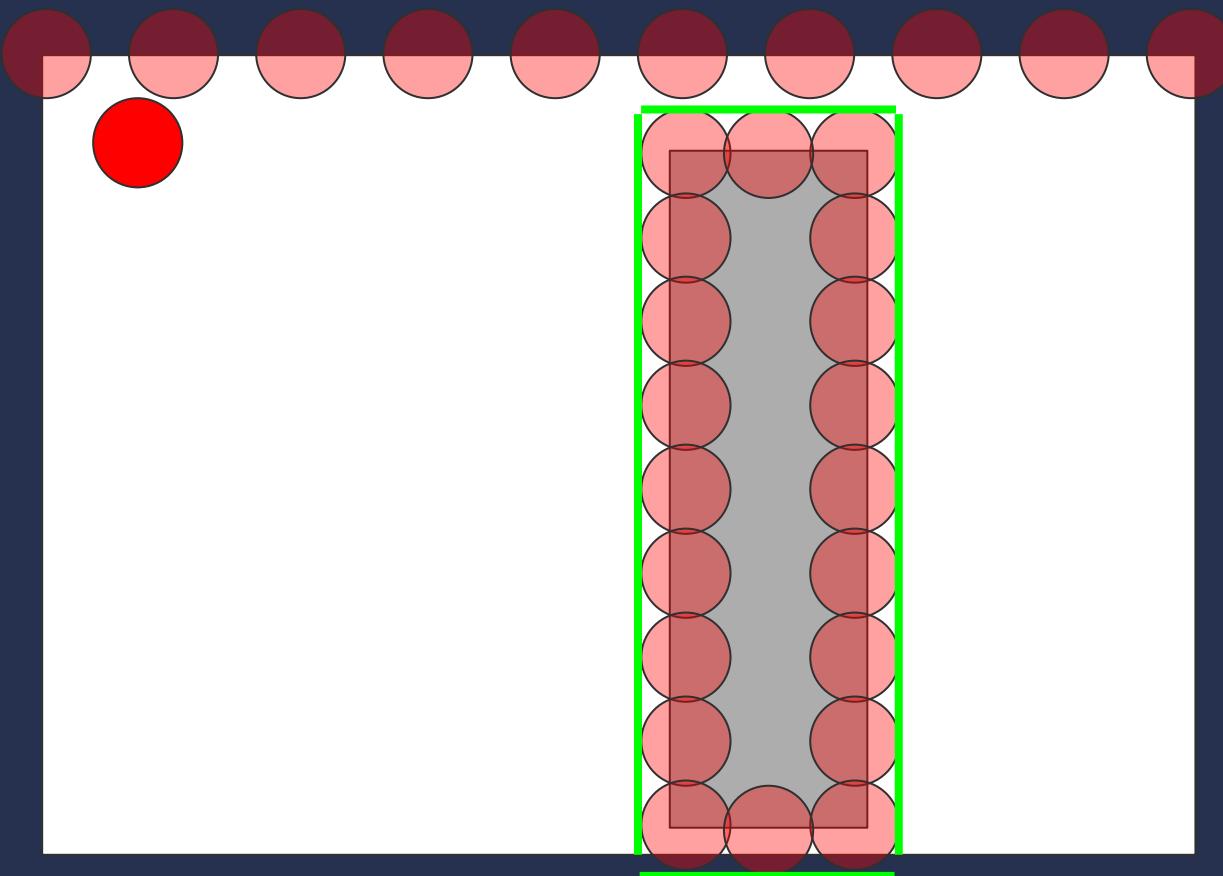
16-735, Howie Choset with slides from G.D. Hager, Z. Dodds, and Dinesh Mocha

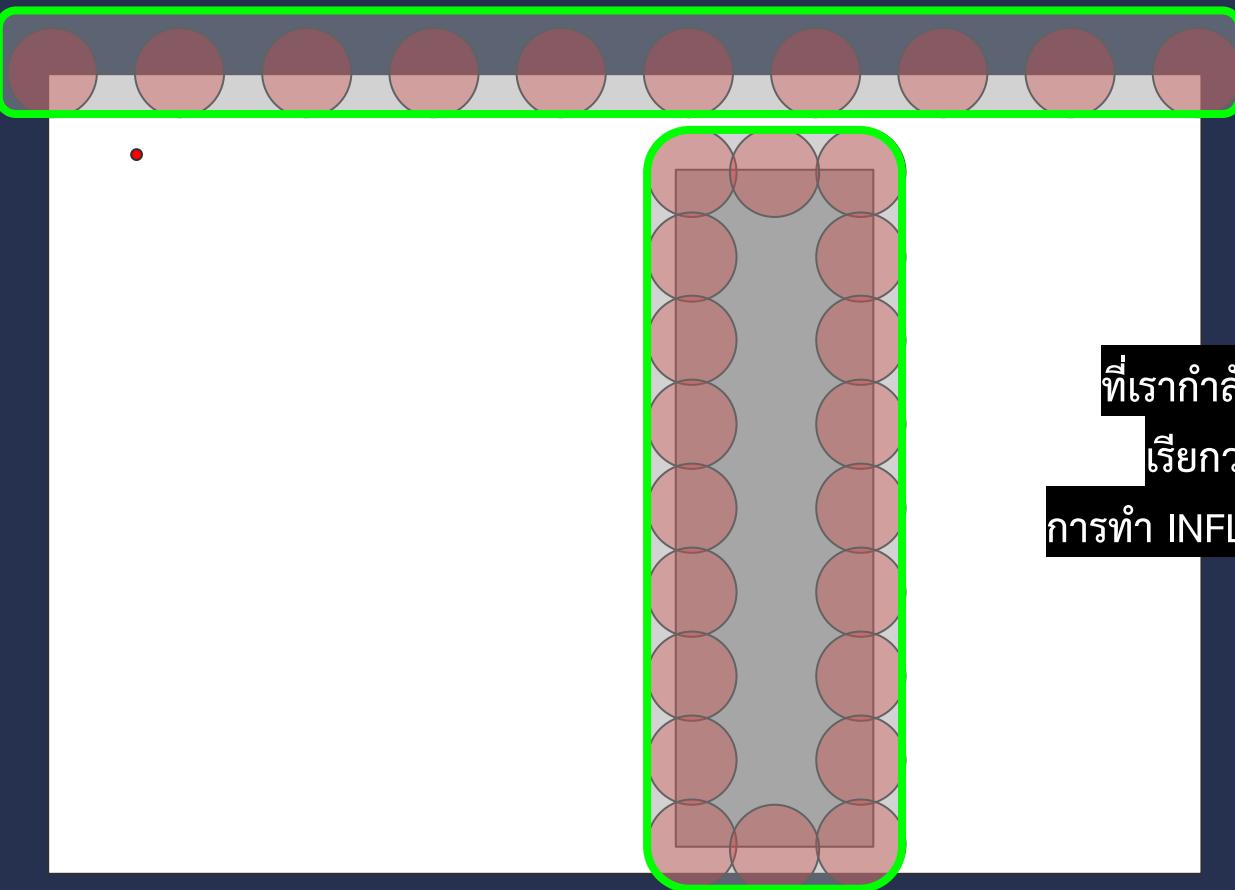


16-735, Howie Choset with slides from G.D. Hager, Z. Dodds, and Dinesh Mocha

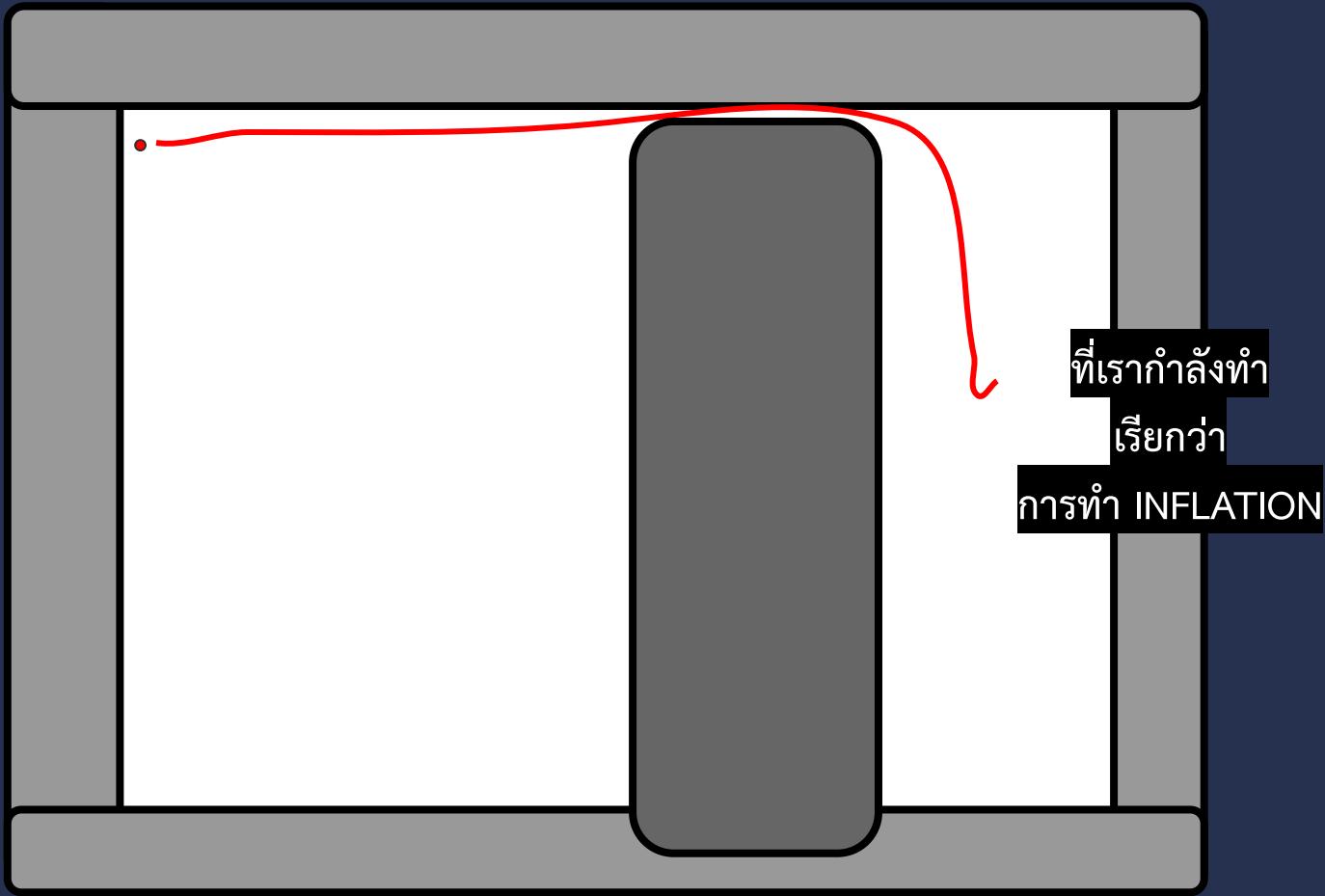








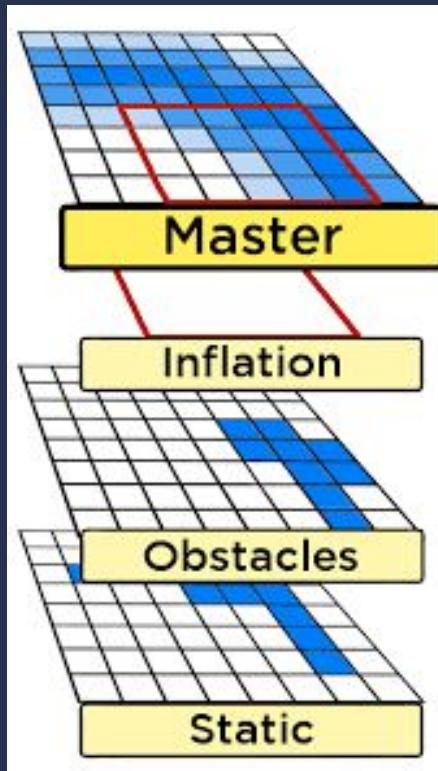
ที่เรากำลังทำ
เรียกว่า
การทำ INFLATION



move_base : costmap

COSTMAP ถือว่าเป็นที่สำคัญของ move_base

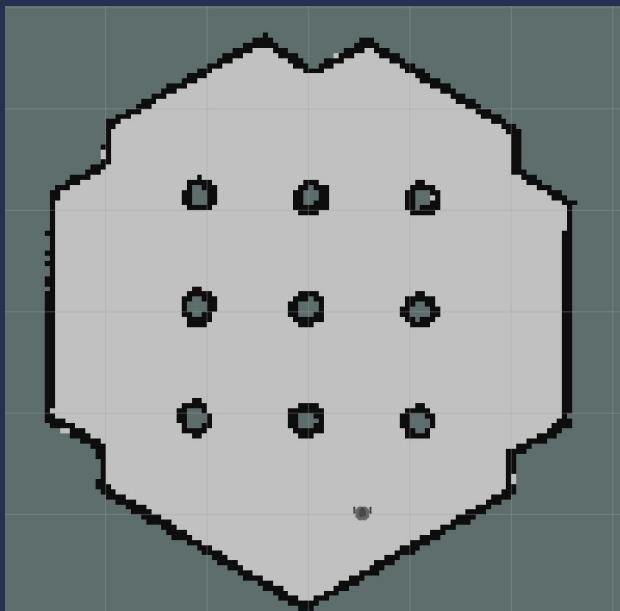
- แผนการเดิน และ การระวังอุปสรรค (Obstacle) จะคำนวณจากข้อมูลตรงนี้เป็นหลัก
- COSTMAP คือ Occupancy Grid
 - รวบรวม Obstacle ต่างๆ
 - เพิ่มข้อมูล อุปสรรคต่างๆเข้าไปบน COSTMAP ได้
 - สร้างระยะระหว่างของได้
- ใช้การทำงานแบบ Layered วางชั้nonๆกัน
 - ผลลัพธ์คือ Master Grid
 - จะประกอบด้วยชั้นย่อยๆ ค่อยๆ EDIT Grid ให้ดีขึ้น



move_base : costmap

COSTMAP ทำงานเป็น LAYER ที่ ซ้อนๆกัน แบบ Filter ใน IG

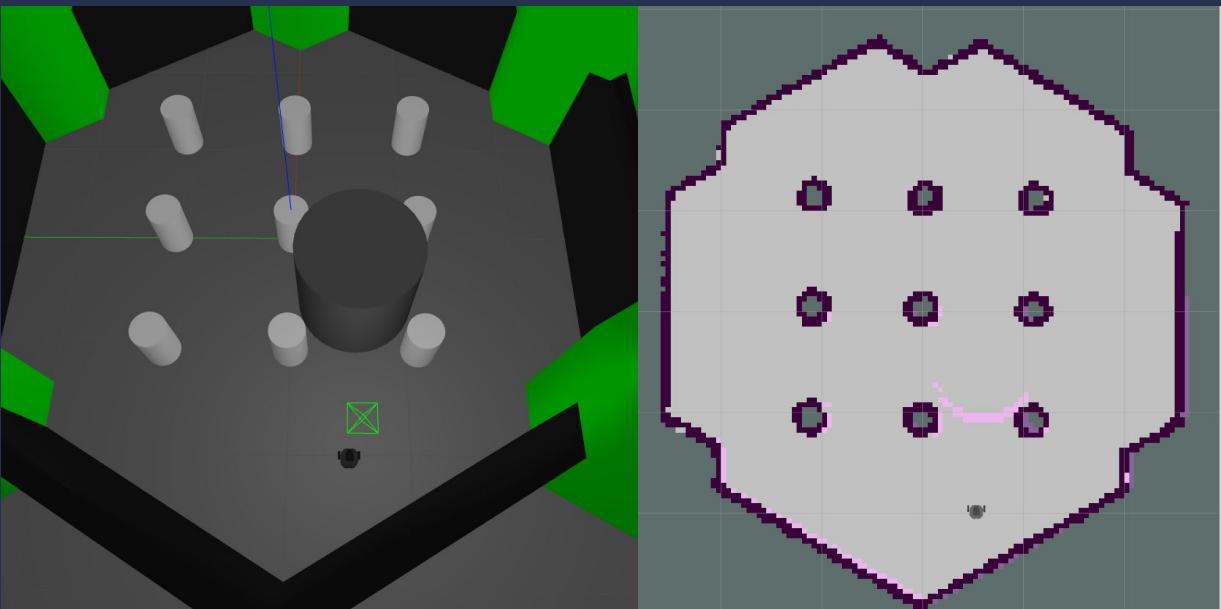
- Static Layer
 - รับ Occupancy Grid มาแปะใน Master Grid
- Obstacle Layer
 - รับข้อมูล Sensor มาแปลงไปใน Master Grid
- Inflation Layer
 - สร้างระยะระวังให้กับ จุดสีดำ ใน Master Grid



move_base : costmap

COSTMAP ทำงานเป็น LAYER ที่ ซ้อนๆกัน แบบ Filter ใน IG

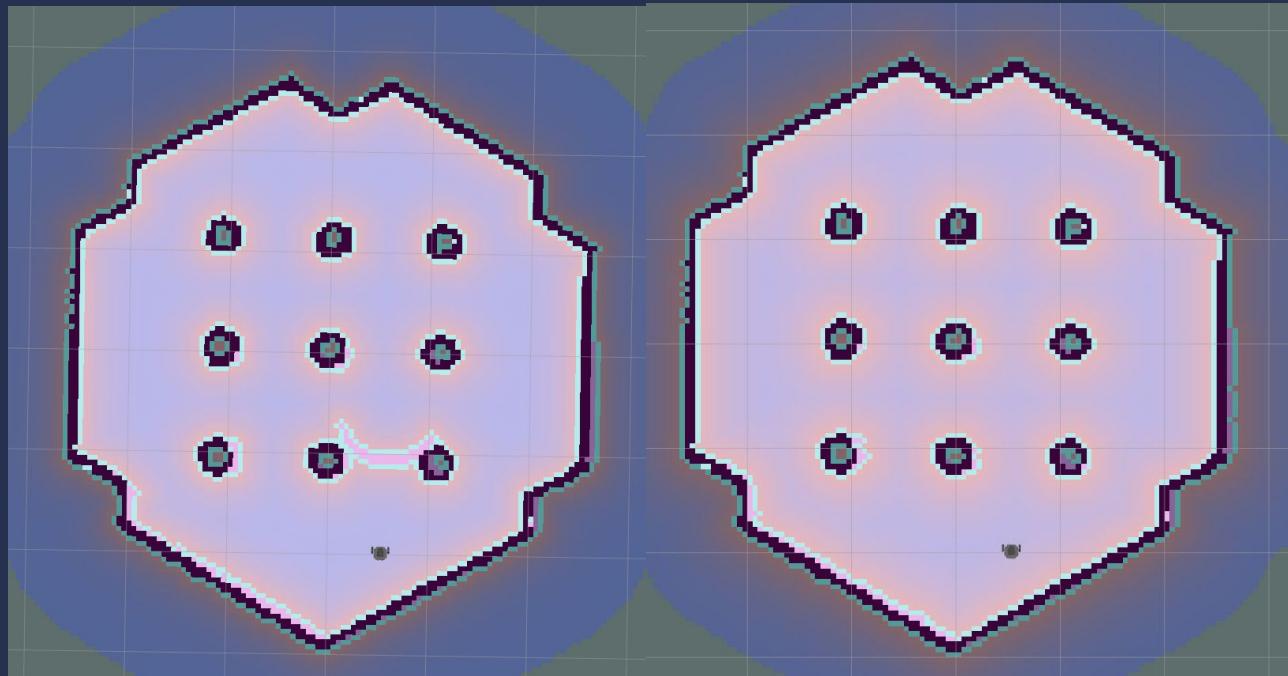
- Static Layer
- Obstacle Layer
- Inflation Layer



move_base : costmap

COSTMAP ทำงานเป็น LAYER ที่ ซ้อนๆกัน แบบ Filter ใน IG

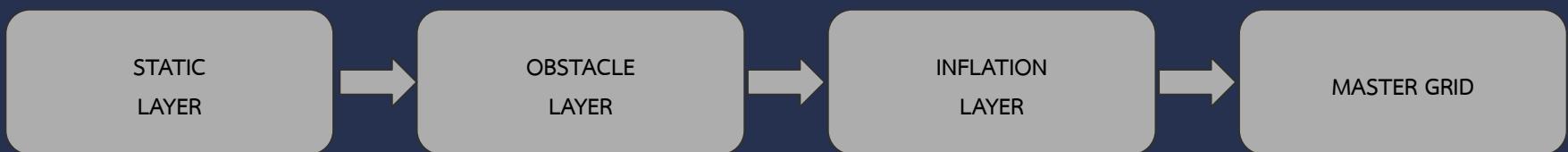
- Static Layer
- Obstacle Layer
- Inflation Layer



แบบละเอียด
ໄລ່ທີ່ລະ Step ເລຍລະກົນ

move_base : costmap

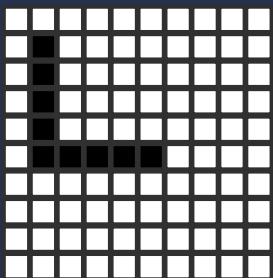
ยกตัวอย่าง การใช้งาน Costmap พื้นฐาน (จริงๆ จะเรียกว่า “Costmap”)



move_base : costmap

ยกตัวอย่าง การใช้งาน Costmap พื้นฐาน (จริงๆ จะเรียกว่า Costmap ได้นะ)

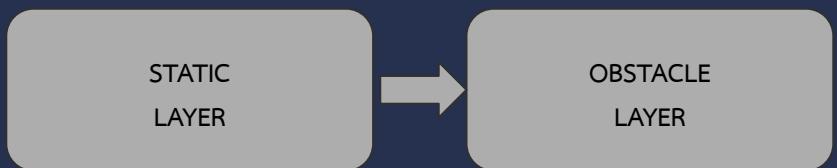
STATIC
LAYER



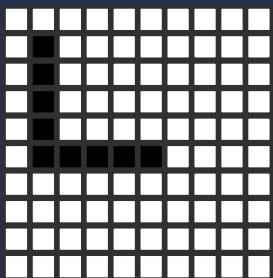
เป็นแผนที่เปล่าๆ ที่ได้จาก
map_server

move_base : costmap

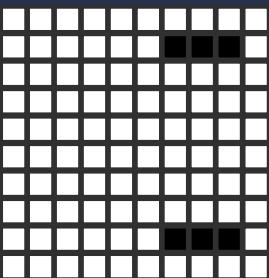
ยกตัวอย่าง การใช้งาน Costmap พื้นฐาน (จริงๆ จะเรียกว่า Costmap ได้นะ)



ผลลัพธ์ของ
Static + Obstacle Layer

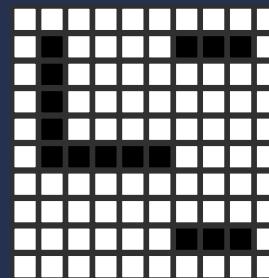


เป็นแผนที่เปล่าๆ ที่ได้จาก
map_server



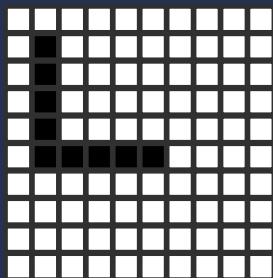
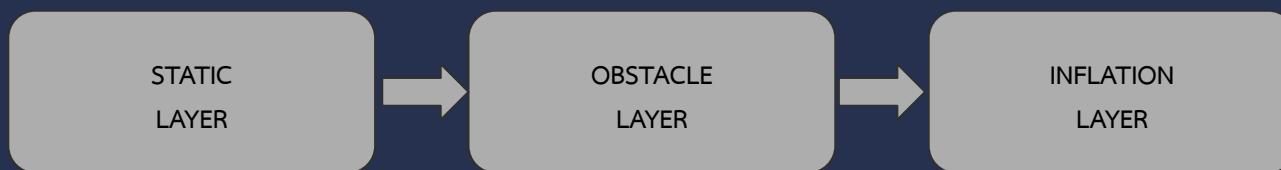
หุ่นยนต์ใช้ Sensor Detect
และ หาด Obstacle

ถ้ามีแค่ 2 Layers นี้มารวมกัน
จะได้แบบนี้

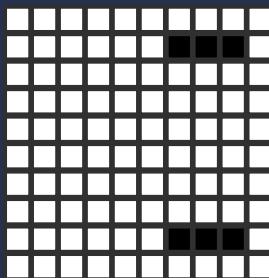


move_base : costmap

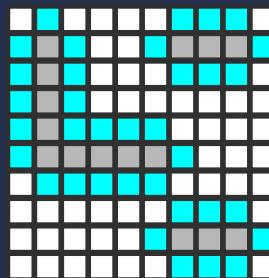
ยกตัวอย่าง การใช้งาน Costmap พื้นฐาน (จริงๆ จะเรียกว่า Costmap ได้นะ)



เป็นแผนที่เปล่าๆ ที่ได้จาก
map_server



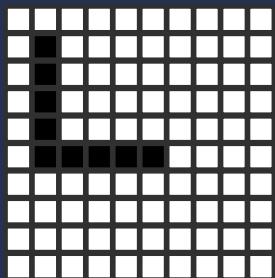
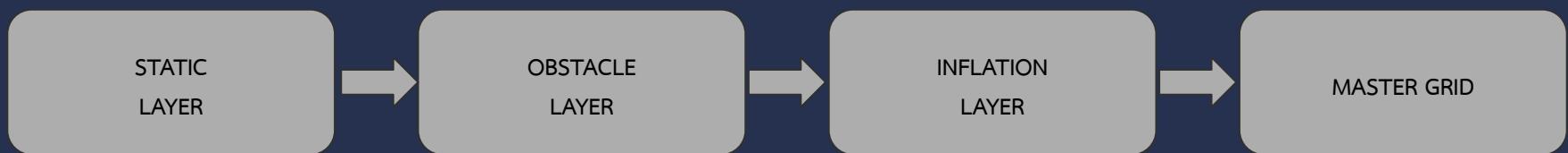
หุ่นยนต์ใช้ Sensor Detect
และ หาด Obstacle



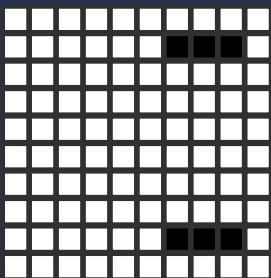
สร้างระยะรั้ง
จาก ช่องสีดำที่มีในแผนที่

move_base : costmap

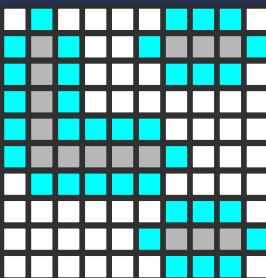
ยกตัวอย่าง การใช้งาน Costmap พื้นฐาน (จริงๆ จะเรียกว่า Costmap ได้นะ)



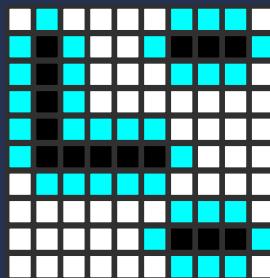
เป็นแผนที่เปล่าๆ ที่ได้จาก
map_server



ทุนยนต์ใช้ Sensor Detect
และ วาด Obstacle



สร้างระยะระหว่าง
จาก ช่องสีดำที่มีในแผนที่



ผลลัพธ์สุดท้าย
ก่อนเอ้าไปใช้งาน

move_base : Global Planning

การวางแผนการเดินแบบ Global Planning จะมีการทำงานตามนี้

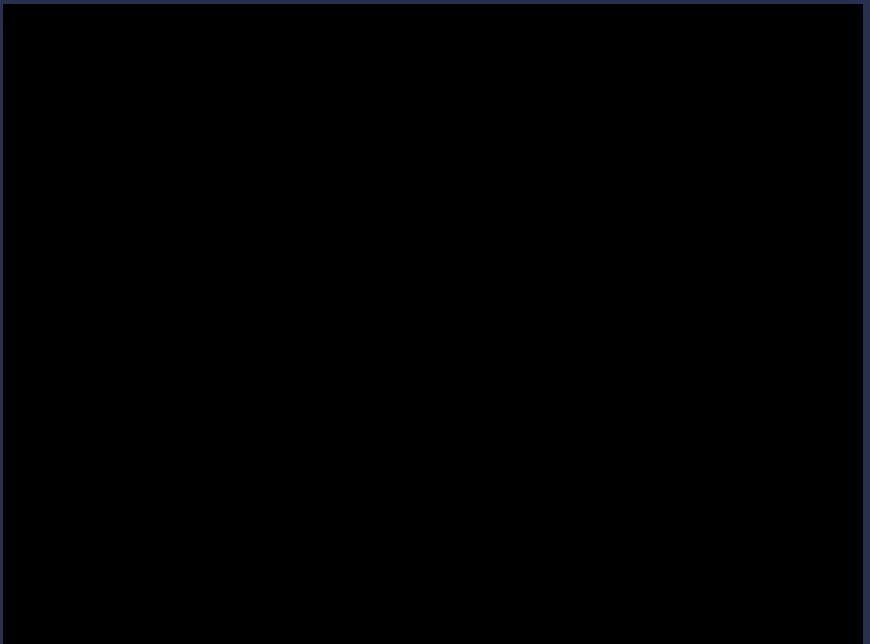
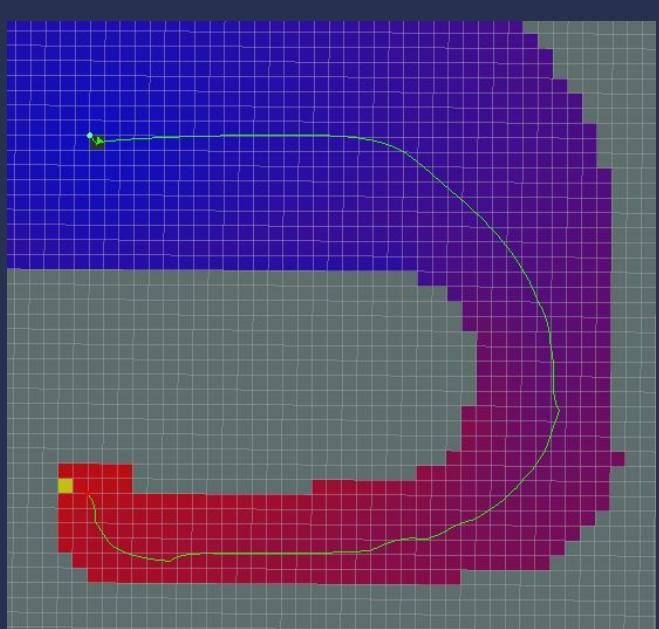
1. สร้างและ Update Global Costmap
2. Global Planner วางแผนการเดิน จาก Global Costmap
3. ส่ง Global Plan ให้ Local Planner

ถ้าสร้างไม่ได้ move_base จะตอบว่า สร้างไม่ได้ และจะทำ Recovery Behavior (ถ้าใส่ไว้)

ถ้ากู้ไม่ได้อีก move_base actionlib จะตอบว่า “ABORTED” เพราะไม่มีทางเดิน (T_T)

move_base : Global Planning

การทำงานในส่วน Global Planning จะเป็นแนวๆ ด้วยการใช้ Algorithm Dijkstra เป็นเบื้องหลัง



move_base : Local Planning

การวางแผนการเดินแบบ Local Planning จะมีการทำงานตามนี้

1. รับ Global Path จาก Global Planner เข้ามา ถ้ามีของใหม่ก็เอาอันใหม่
2. สร้างและ Update Local Costmap Size เล็กๆ รอบๆหุ่น
3. Local Planner วางแผนการเดินจาก Global Path และ Costmap รอบๆหุ่น
ด้วย Algorithm DWA
4. Local Planner สั่งให้หุ่นเดินอย่างปลอดภัย (?)

ถ้าสร้าง Local Plan

ไม่ได้ move_base จะตอบว่า สร้างไม่ได้ และจะขอให้ Global Planner Plan ใหม่

ถ้าเดินไม่ได้ move_base actionlib จะตอบว่า “ABORTED” เพราะหุ่นติด

move_base : Local Planning

การวางแผนใน Local Frame

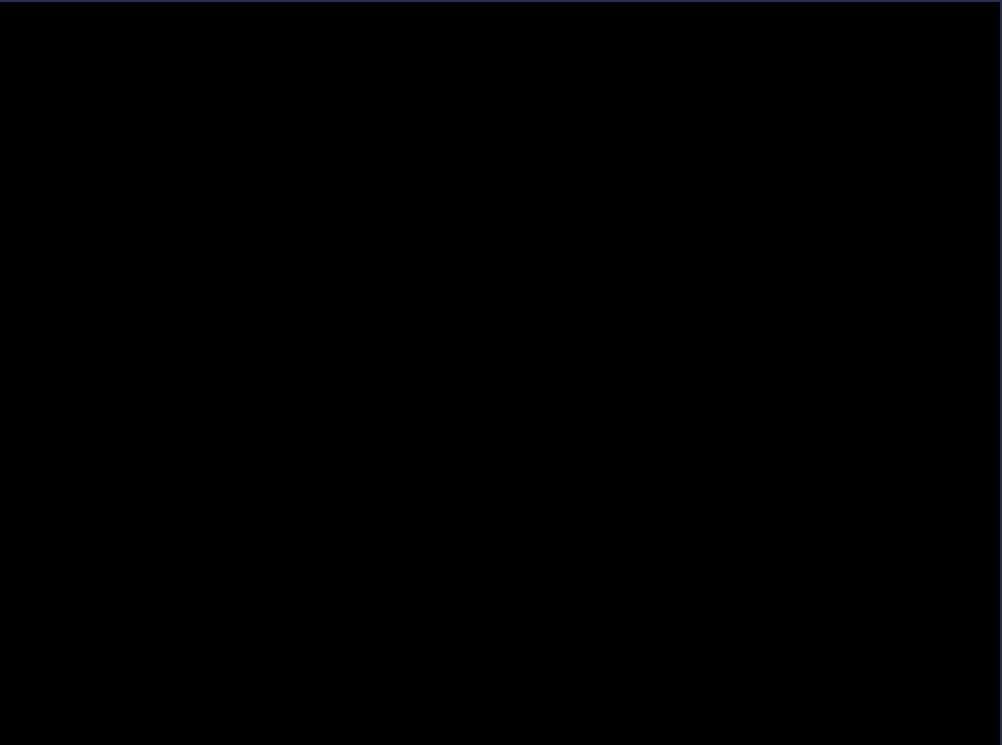
สิ่งที่เราสนใจจะมีดังนี้

- หุ่นยนต์จะรีบความเร็วได้มาก
- หุ่นยนต์จะต้องวิ่งไปทางไหน
- ทางไหนดี ที่จะไม่ชนสิ่งของ

ในการ Plan Local Path ส่วนใหญ่

Algorithm ที่ใช้ จะซึ่งว่า

Dynamic Windows Approach



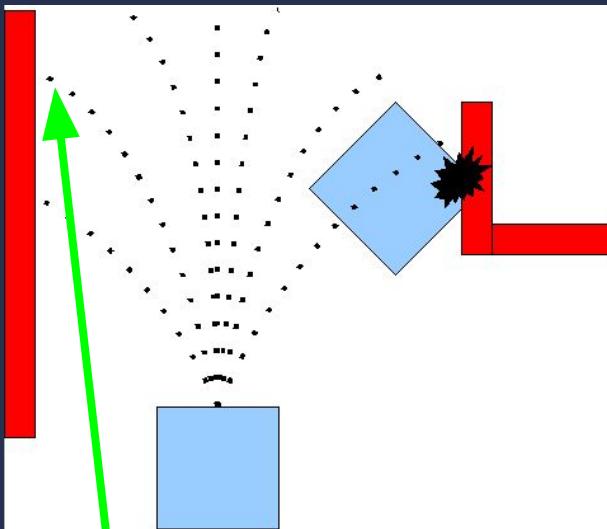
move_base : Local Planning

DWA จะทำการ Simulate ความเร็วที่หุ่นยนต์ทำได้
ทุกความเร็ว ในช่วงเวลาสั้นๆ

ถ้าหุ่นยนต์เป็น Non-holonomic Drive จะได้ความเร็วดังรูป

- แกนนอน V_w (ความเร็วโค้ง)
- แกนตั้ง V_x (ความเร็วหน้าหลัง)

จะพบว่าเราเลือกความถี่ของการจำลองได้
หลังจากนั้นเอาจุดปลาย ($V_r * dt$) มาเช็ค
ว่าโดนสิ่งกีดขวางไหม ?



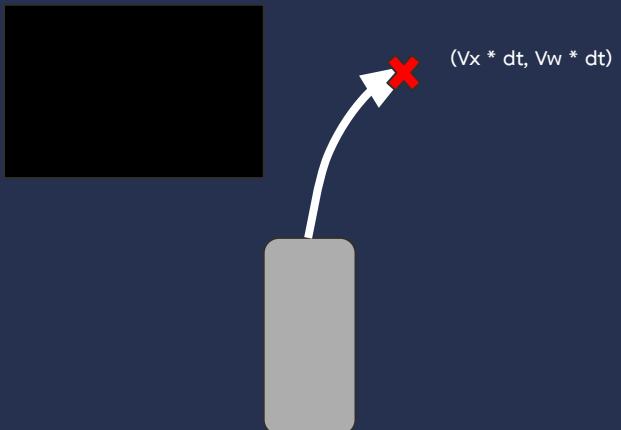
move_base : Local Planning

DWA จะทำการ Simulate ความเร็วที่หุ่นยนต์ทำได้
ทุกความเร็ว ในช่วงเวลาสั้นๆ

ถ้าหุ่นยนต์เป็น Non-holonomic Drive จะได้ความเร็วดังรูป

- แกนนอน Vw (ความเร็วโค้ง)
- แกนตั้ง Vx (ความเร็วหน้าหลัง)

จะพบว่าเราเลือกความถี่ของการจำลองได้
หลังจากนั้นอาจจุดปลาย ($Vr * dt$) มาเช็ค
ว่าโดนสิ่งกีดขวางไหม ?



move_base : Local Planning

พอเลือกจุดมาแล้ว Local Planner จะต้องดูว่า

“มันเป็นแผน ... ที่ดีไหม ?”

แผนที่ดี = ตามเส้นทาง + ไม่ชนของ + ใกล้จุดหมาย

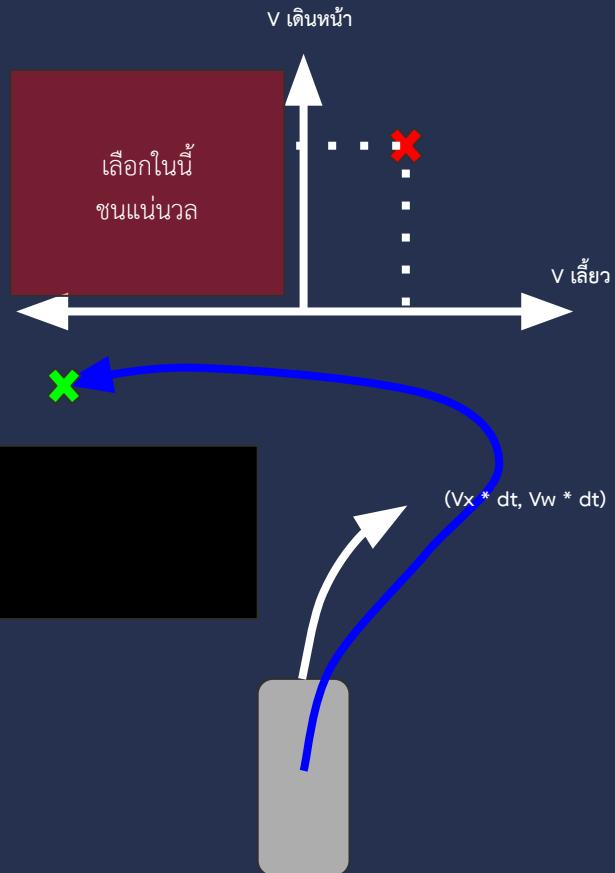
เจ้าความเร็วที่เดินไปหาว่าถ้าเดินไปแบบนั้น

ตำแหน่งต่อนับ ผลลัพธ์ดีไหม ?

อาจมีปัจจัยอื่นๆ ที่ ปรับเพิ่มได้

- หน้าหันตรงกับเส้นทางเดินไหม ?
- อย่างให้ห่างสิ่งกีดขวางมากๆ

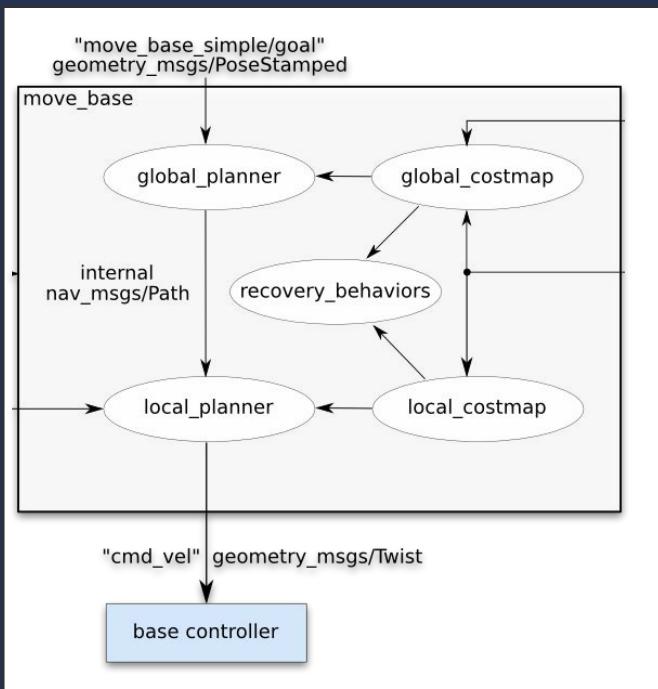
“คิดหาอันที่ดีที่สุดมา”



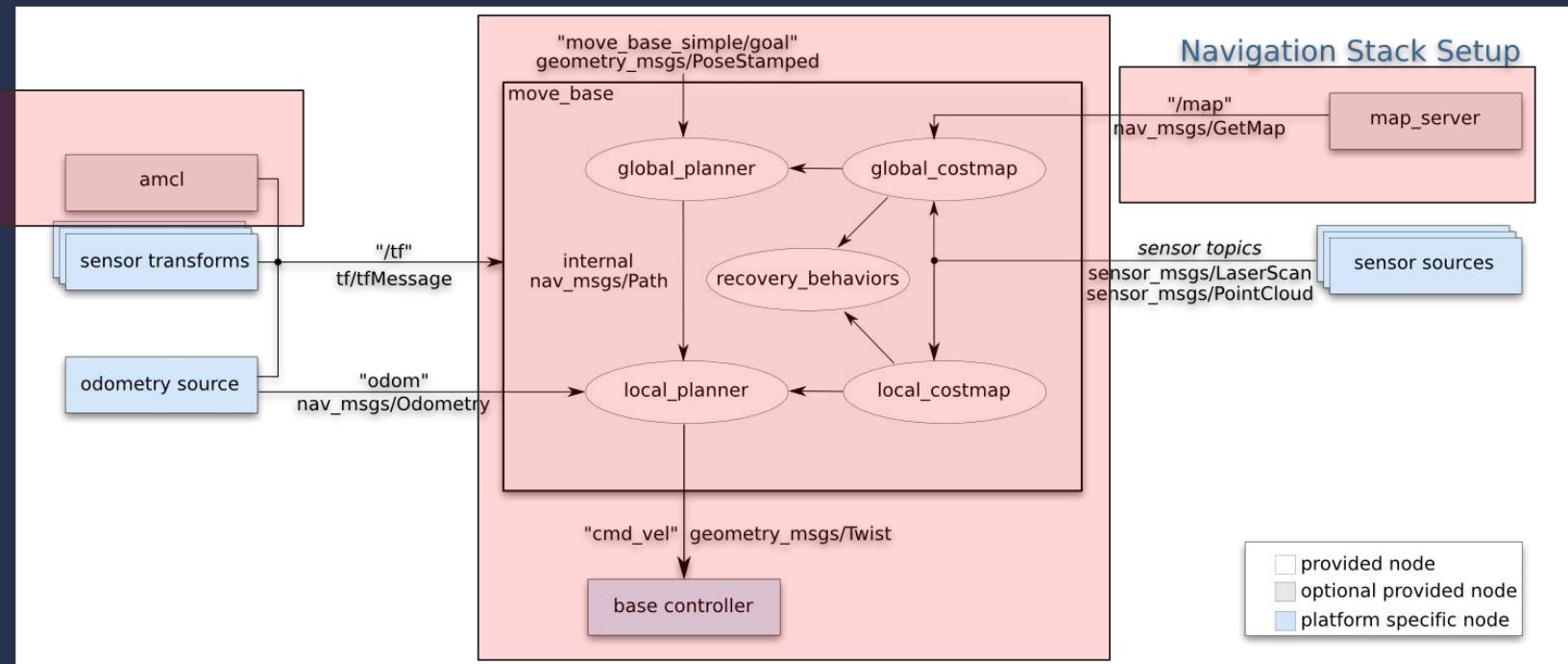
move_base

ສຸດທ້າຍແລ້ວ ຄວາມເຮົາທີ່ຄິດໄດ້ ໃປ້ໄໝ ?

- Move base ຈະ Publish TOPIC : /cmd_vel
 - TYPE : geometry_msgs/Twist (ເກີບຄວາມເຮົາຫຸ່ນ)
- ຂຶ້ງພວກຈູານຫຸ່ນຍືນຕໍ່ 99%
ຈະຊອບຮັບ (Subscribe) Type ນີ້
- ທຳໃຫ້ສຽບໄດ້ວ່າ move_base
 - ຮັບ ຈຸດໝາຍປາຍທາງເຂົ້າໄປ (goal)
 - ຮັບ ແຜນທີ່ (map) ເຂົ້າໄປ
 - ຮັບ Sensor ແວດລື້ອມ (LaserScan) ເຂົ້າໄປ
 - ຮັບ ຫຼຸ່ມໆ Odometer ຂອງຫຸ່ນ (Odometry)
 - ພ່ນ ຄວາມເຮົາຫຸ່ນຍືນຕໍ່ (cmd_vel) ອອກນາ



และทั้งหมดที่เราเรียนวันนี้ ก็คือทั้งหมดนี่เอง !!!!



ลองเล่นกันเถอะ !

อีกละ 5555

ทดลองเล่นตัว ROS Navigation Stack

1. [Terminal 1] roscore

2. [Terminal 2] Simulation
 - > roslaunch **ignition_playground** block.launch

3. [Terminal 3] ROS Simulation Bridge
 - > roslaunch **ignition_playground** offline_team.launch

4. [Terminal 4] ROS Navigation Stack
 - จะประกอบด้วย map_server, amcl , move_base
 - แออิรุบมาให้แล้วແລະ
 - > roslaunch **ignition_navigation** bringup.launch

ทดลองเล่นตัว ROS Navigation Stack

โปรแกรมที่จำเป็น

- RViz
 - เป็นโปรแกรมช่วยแสดงผล ว่าหุ่นยนต์ได้ข้อมูลอะไรบ้าง (ข้อมูลที่หุ่นยนต์ถือไว้)
 - > rosrun rviz rviz
 - หรือใช้ > rviz เลยก็ได้
- RQT TF Tree
 - เอาไว้ดูว่า TF Tree ของเราหน้าตาเป็นยังไง
 - > rosrun rqt_tf_tree rqt_tf_tree
- RQT Reconfigure Parameter
 - เอาไว้ปรับจุนค่าหุ่นยนต์
 - จนได้เฉพาะ ตัวแปรแบบ Dynamic Reconfigure
 - ใน navigation stack มีหลายตัวที่ทำ Dynamic Reconfigure ได้

ทดลองเล่นตัว ROS Navigation Stack

โปรแกรมที่จำเป็น

- RQT ROBOT STEERING
 - โปรแกรมเอาไว้ขับหุ่นยนต์ (แบบไม่จริงจัง 55)
 - > rosrun rqt_robot_steering rqt_robot_steering
- หรือถ้าอยากรทำ Dashboard ของตัวเอง ลองใช้
 - > rqt
 - จะได้หน้า Dashboard เป็นๆ เราสามารถ Add Plugin ต่างๆ
 - และกดเชป ไว้เดิม ** อย่าลืมเชฟงาน ณณ

ແລ້ວເຮົາຈະປັບຄ່າອະໄຫຼວດໃບໆບ້າງ ?

ຕິດຕາມຕໍ່ອິນ DAY 2

:)