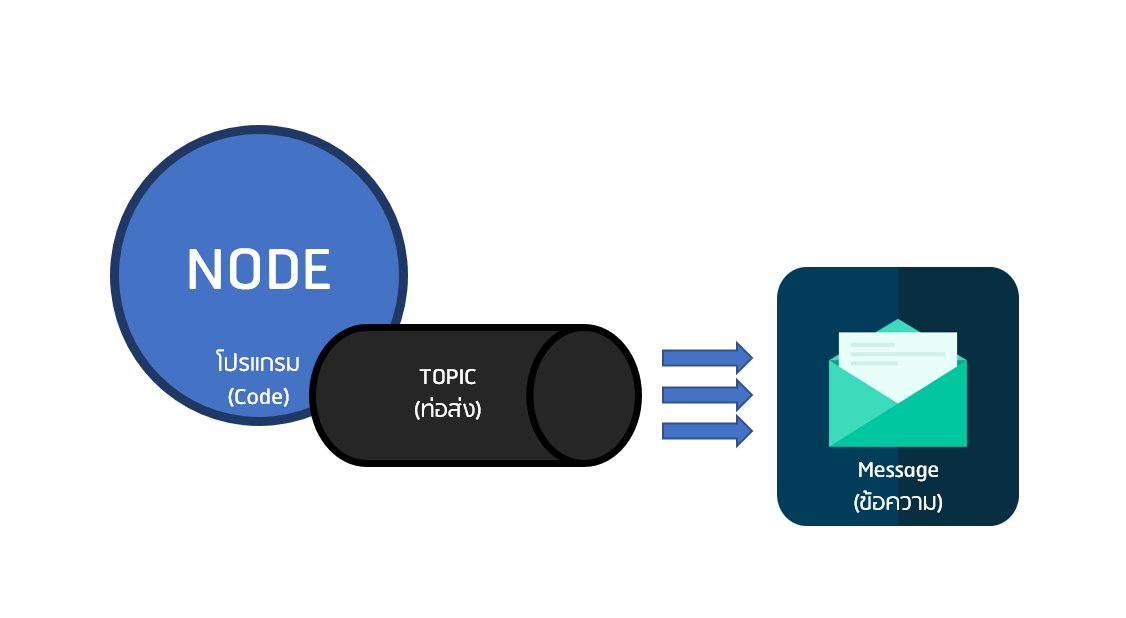
Concept of Modularity

ROS นั้นช่วยให้ผู้พัฒนาหุ่นยนต์ สามารถส่ง message ระหว่าง module ต่างๆในระบบได้ง่ายดายนั้นคือ concept หลักๆของมันโดยวันนี้เราจะมาพูดถึงโครงสร้างของ Software ที่จะเกิดขึ้นกันขอมาตั้งคำศัพท์กันหน่อยก็แล้วกัน



Node — ส่วนของโปรแกรมที่ทำหน้าที่เฉพาะ เช่น Node ของการจับหน้าคน Node ของการขับเคลื่อนมอเตอร์

Node ของการรับค่าจาก Joystickมองมันเป็น Module หนึ่งใน Software ก็ได้





Message — ข้อความที่ Node ใช้คุยกัน

คำที่ควรรู้ก่อนใช้ ROS

**ROS Terminology**

ในบทความนี้ผมจะมาเสนอคำที่เจอบ่อยเวลาใช้ ROS ซึ่งบางครั้งคนที่มาใหม่อาจจะยังไม่เข้าใจก็ยังไม่ต้องกังวล ให้อ่านพอผ่านตาสักหน่อย แล้วก็ไปต่อได้เลย ถ้าพอได้ลองทำแล้วก็จะค่อยๆคุ้นเคยกับมันเอง ต้องลองทำบ่อยๆแล้วจะเก็ตแนวคิดของมันไปเริ่มรู้จักคำศัพท์กันเลย

**\* ROS**

ROS เป็นเหมือนเครื่องมือที่ช่วยให้เราสามารถทำงานกับหุ่นยนต์ได้ง่ายขึ้น มีฟีเจอร์ต่างๆ เช่น sensing, recognizing, mapping, motion planning, message passing, package management, visualizers และไลบรารี่สำหรับการ debugging อีกมากมาย

**\* Master**

master ทำหน้าที่เป็น server สำหรับให้ node หลายๆตัวเชื่อมต่อกัน และสามารถส่ง message หากันได้ คำสั่งที่ใช้รัน master คือ roscore ที่เราใช้ก่อนหน้านี้นี่เอง เมื่อเรารัน master แล้ว เราก็สามารถที่จะสร้าง node ต่างๆขึ้นมาเพื่อดึงข้อมูลได้ การเชื่อมกันระหว่าง node จะส่งข้อความหากันผ่าน topic

master ใช้ HTTP-based protocol ในการสื่อสารกัน เมื่อเรารัน ROS master เราสามารถที่จะตั้งค่า URI address และ port ได้ โดยการเปลี่ยนค่า ROS\_MASTER\_URI โดยปกติแล้วถ้าเราไม่ตั้งค่า default จะเป็น local PC และ port เป็น 11311

**\* Node**

node คิดง่ายๆก็คือโปรแกรมย่อยๆที่รันอยู่ใน ROS เวลาเขียน node ควรจะเขียนให้เป็น 1 node ทำหน้าที่ 1 อย่าง เพื่อที่จะได้ง่ายเวลาเอากลับมาใช้ใหม่ ลองจินตนาการว่าเรามีหุ่นยนต์ mobile robot ที่เคลื่อนที่ 2 ล้อ เวลาแบ่ง node ออกมาก็จะเป็นประมาณว่า มี node อ่านเซนเซอร์, node ขับเคลื่อนมอเตอร์, node อ่านเอนโค้ดเดอร์, node navigation, node obstacle recognition

การสร้าง node เราสามารถสร้างกี่ node ก็ได้ แต่ละ node จะเชื่อมกันผ่าน master, แต่ละ node จะมี name, ชนิด message, uri address และ port เป็นของตัวเอง เรานั้นสามารถที่จะทำหน้าที่เป็น publisher, subscriber, service server หรือ service client ได้ และจะคุยกับ node ตัวอื่นๆผ่าน topic และ service

**\* Package**

package คือพื้นฐานของ ROS, แอพพลิเคชั่นทั้งหมดใน ROS จะพัฒนาโดยมี package เป็นรากฐาน ใน package นั้นจะเก็บพวกไฟล์ configuration ไปจนถึงไฟล์ launch ที่สามารถไปรัน package หรือ node อื่นๆได้ ตอนนี้ ROS มี packages มากกว่า 5000 packages แล้ว

**\* Metapackage**

metapackage เป็นการรวมกันของ packages ที่ทำหน้าที่คล้ายๆกันหลายๆตัวมารวมไว้ที่เดียวเพื่อจะได้ใช้งานง่าย ตัวอย่าง Navigation metapackage ประกอบไปด้วย 10 packages เช่น [AMCL(partical filter), DWA, EKF(extended kalman filter) และ map\_server] ซึ่งเวลาใช้ก็ติดตั้ง metapackage ตัวนี้ก็จะได้มาหมดเลย

**\* Message**

node แต่ละตัวจะคุยกันผ่าน message, โดย message ก็อาจจะเป็นชนิดของตัวแปร เช่น integer, floating point, boolean หรืออาจจะเอามาประกอบกันเป็น structure เช่นประกอบเป็น array หรือประกอบรวม int, float, string เข้าด้วยกันก็ได้

**\* Topic**

topic ก็เหมือนกับหัวเรื่องที่เราไว้คุยกัน พูดง่ายๆคือ publisher node จะ publish message ของตัวเองขึ้นไปที่ topic แล้วจะมี subscriber node มารอรับ message ที่ส่งมาทาง topic ที่มีชื่อเดียวกัน

**\* Publish and Publisher**

เวลาพูดว่า publish หมายถึงการที่ node จะส่ง message ออกไปที่ topic โดยจะส่งออกไปที่ master โดยไม่สนใจว่าจะมี node รอรับอยู่รึเปล่า node 1 ตัวสามารถ publish message ออกไปได้หลายๆ topic

**\* Subscribe and Subscriber**

เวลาพูดว่า subscribe หมายถึงการที่ node จะรับ message จาก topic ที่ตัวเองรอรับอยู่ node 1 ตัวสามารถ subscribe message ที่มาจากหลายๆ topic ได้

การสื่อสารแบบใช้ topic นั้นจะเป็นแบบ asynchronous หรือก็คือทำหน้าที่แบบไม่สนใจคนอื่นเลย อยากส่งก็ส่ง อยากรับก็รับ โดยมากแล้วก็จะใช้กับ sensor ที่ต้องการส่งให้มีความถี่คงที่ แต่จะต่างกับการใช้ service ที่จะเป็นแบบ synchronization โดยจะต้องส่งไปขอแล้วก็รอตอบรับ นึกง่ายๆก็เหมือนกับการเรียก function

**\* Service**

service เป็น synchronous โดยจะมี service client กับ service server การทำงานก็คือ node ที่เป็น service client จะส่งไปหา node service server แล้ว node server ก็จะตอบคำตอบกลับไปที่ node client

**\* Service Server**

service server จะเป็นเหมือน server ที่จะคอยรับ request เป็น input แล้วก็ response เป็น output, โดยทั้งรับและส่งจะเป็น message

**\* Service Client**

service client เวลาใช้ก็จะเหมือนกับส่ง message ไปหา service server แล้วก็รอให้ server ตอบกลับมาแล้วเอาไปประมวลผลต่อ

**\* Action**

action ก็เป็นการเชื่อมต่อกันอีกรูปแบบ โดย action จะคล้ายๆกับ service ต่างกันตรงที่เวลา call service จะต้องรอให้ service ทำเสร็จก่อน แต่ call action จะไม่ต้องรอสามารถทำงานอย่างอื่นได้เลย หรือพูดง่ายๆก็คือ action คือการเรียก function ที่เป็นแบบ asynchronous

**\* Action Server**

action server จะรอรับ goal จาก client แล้วก็ response ด้วย feedback, ตัว server จะรับ goal จาก client ครั้งเดียวแล้วมันจะทำงานตาม process จนเสร็จเอง

**\* Action Client**

action client จะทำหน้าที่ส่ง goal ไปหา server แล้วก็รับ feedback กลับมาจาก server, เราสามารถส่ง instruction ให้ cancel การทำงานได้

**\* Parameter**

parameter ในที่นี้จะใช้กับ node ให้นึกถึง \*.ini ของโปรแกรมใน windows โดยปกติแล้ว paremeter จะมีค่า default และสามารถที่จะอ่านหรือเขียนได้ ส่วนใหญ่จะใช้กับพวกค่าที่ต้องเปลี่ยนบ่อยๆ เช่น usb port number, camera calibration parameter, maximum and minimum value of motor speed

**\* Parameter Server**

parameter server จะใช้ตอนตอนเรียก package โดยค่าพวกนี้จะถูก load ไปให้ master

**\* Catkin**

catkin เป็นตัวที่เราใช้ build system ของ ROS, การ build จะใช้ CMake (Cross Platform Make) และจะ build environment ตามที่กำหนดใน ‘CMakeLists.txt’ ที่อยู่ในโฟลเดอ package

**\* roscore**

roscore เป็นคำสั่งสำหรับใช้รัน ROS master ถ้าใช้คอมหลายๆเครื่องเชื่อมต่อกันสามารถใช้ roscore ตัวเดียวกันได้โดยการไปตั้งค่า ROS\_MASTER\_URI ก็จะทำให้เชื่อมต่อหากันได้

**\* rosrun**

rosrun เป็นคำสั่งพื้นฐานสำหรับรัน node ใดๆใน package แต่จะใช้ rosrun ได้ก็จะต้องรัน roscore ก่อน

**\* roslaunch**

rosrun ใช้สำหรับรันเพียง 1 node แต่ roslaunch จะสามารถรันหลายๆ node พร้อมกันได้

roslaunch จะใช้มีนามสกุลเป็น \*.launch มีลักษณะเหมือนภาษา XML (Extensible Markup Language) และก็มี tag เป็นรูปแบบให้อยู่แล้ว

**\* rosbag**

ข้อมูลต่างๆใน ROS ที่มีการส่งหากันสามารถเก็บเอาไว้ได้โดยใช้ rosbag ซึ่งจะเก็บเป็นไฟล์ \*.bag แล้วไฟล์ตัวนี้นอกจากจะเก็บไว้แล้ว ยังเอามา playback ได้อีกด้วย ตัวอย่างการใช้งานก็เช่น ถ้าเรามีหุ่นยนต์ที่อ่านค่าเซนเซอร์มาเก็บไว้ เมื่อเราเก็บไว้แล้ว เราสามารถเอาค่านั้นมาจำลองการทำงานของหุ่นยนต์ได้โดยไม่ต้องรันหุ่นจริงๆ

**\* Graph**

ความสัมพันธ์กันระหว่าง nodes, topics, publisher, subscribers สามารถแสดงออกมาเป็น grpah ได้ โดยใช้คำสั่ง rqt\_graph