Principles of Safe Autonomy
ECE 498 SMA
Lecture 4: ROS

Professors: Joohyung Kim and Sayan Mitra

Graduate Teaching Assistants: Tianqi (Ted) Liu and Yangge Li

Undergraduate Assistants: Qichao Gao and Hebron Taylor

Plan

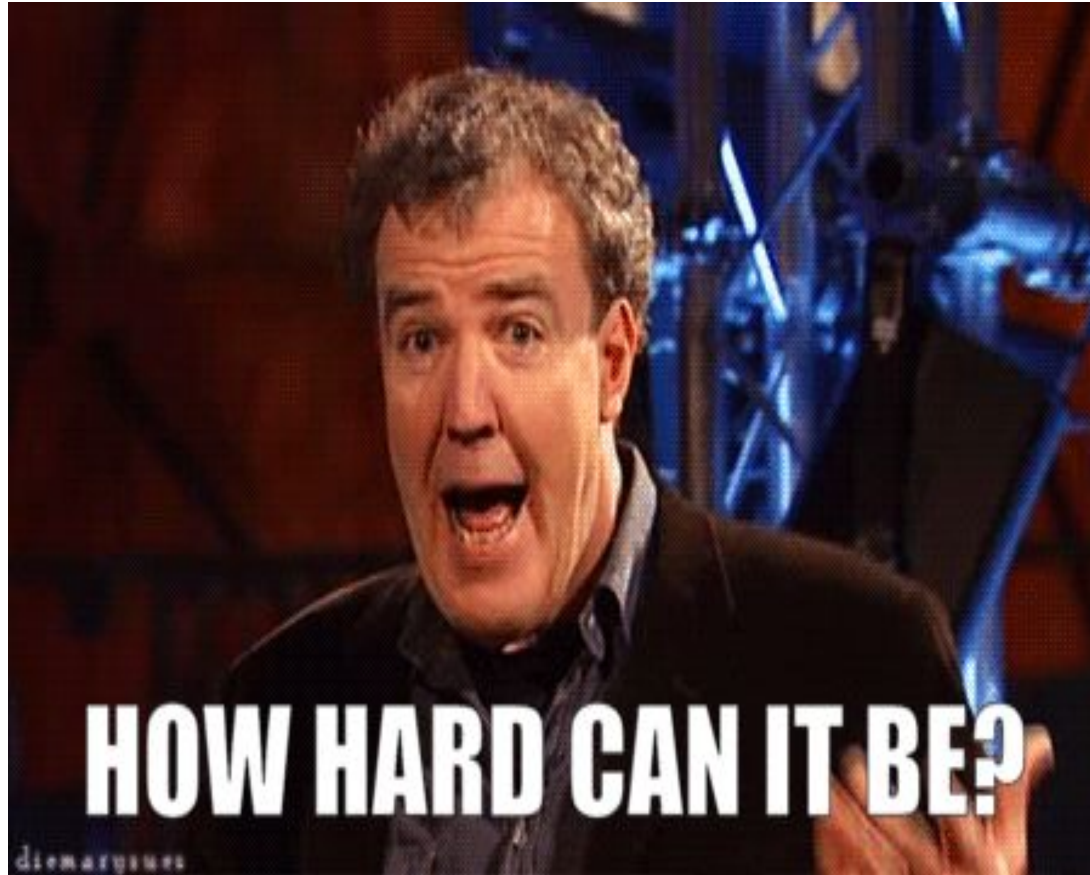
- Background
- Brief Introduction on ROS
- ROS Node, Messaging
- Code

Let’s build a house cleaning robot

- 1. Camera to observe environment
- 2. 2 motors to power the wheels
- 3. Microcontroller (Arduino, RosberryPi) to control the robot

  4. A vacuum cleaner

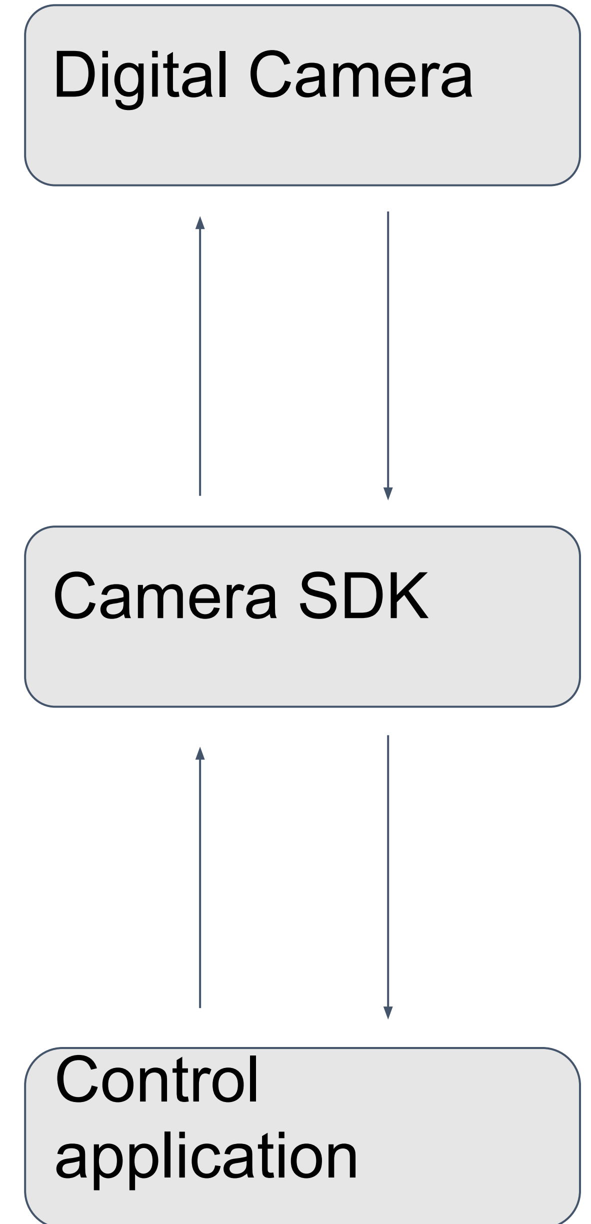
How Hard can it be?



Camera is easy to use, right?

- Need Software Development Kit (SDK) between our application and camera to interpret the signal from camera to RGB image and send command to camera
- "The realization of intelligent solutions requires additional software that is suitable to run with the camera."

--[Mako Camera User Manual]



Simple Roomba

Digital Camera

Wheel Motor

Vacuum Motor

2

3

7

10

Camera SDK

Wheel Motor SDK

Vacuum Motor SDK

1

4

6

9

Lane Detection Module

5

Motor Controller Module

8

Vacuum Controller Module

1.Can we do multiple tasks at the same time?

- Case 1. The motor controller module and vacuum controller module have no dependence on each other. They should be able to run in parallel.
- Case 2. The lane detection module needs image data from camera module. So they should run in sequence.

2.What if things become a bit more complicated?

- Imagine we have a lane detection module and a face recognition module, both needs to read image data from camera. It would be a tedious process if they individually communicate with camera hardware. We want some centralized system to set up those connections automatically through some carefully designed messaging interfaces.

3.Can we reuse the modules?

- If we developed a lane detection module, we might want to use it on both a self-driving car and a Roomba. The software needs to be independent of hardware.

What do we want?

- 1. Asynchronous System. Unrelated code can run in parallel to improve efficiency.
- 2. Encapsulation. Standardize the interface. So that we can build complex systems
- 3. Abstraction. Separate different parts of applications. So we can reuse code for different systems

ROS can give us all of them (plus more)

What is ROS?

- It is a open source framework with a set of tools, libraries and interfaces that helps users to set up a robot quickly.
- Founded by Willow Garage(a robotics incubator) at Stanford University in 2007
- ROS 1 and ROS 2 are different . We are using ROS 1 Kinetic in this course because the GEM car uses it.

What is ROS?

- It is NOT an operating system
- It is a “meta-operating system” for robots
- It is a middleware between operating system and applications

Application

ROS

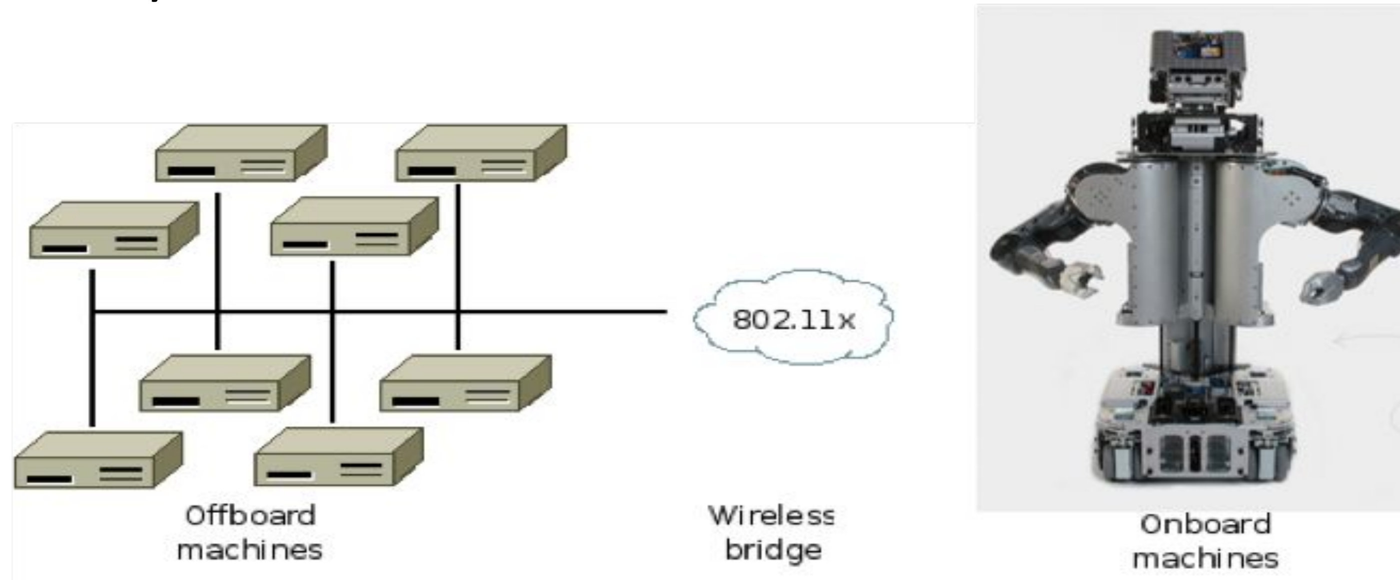
UNIX Operating System

Why ROS?

- Support modular and distributed design. (Pick whatever you want)
  ROS separates different parts of your code into different nodes, manages their builds and dependencies. So we can develop and compile them separately, thus minimize compatibility issues. When adding new modules developed by others, no matter the hardwares or software, we can just “plug in” to the existing project

Why ROS?

- Enable communication between different systems (TCP and UDP under the hood)



Why ROS?

- Support and documentation

(1) Multiple Languages: C++, Python and more
(2) Include existing tools: OpenCV, Point Cloud Library, OpenRAVE
(3) Huge Community (>3000 packages)

Restrictions on ROS

- ROS1 only works on Linux Systems(officially). MacOS and Windows users can install Ubuntu Virtual Machines.
- ROS is based on UNIX platforms. Hence it will NOT work on a real time operating system (RTOS) or real time applications. No guarantee on how much time it will take to finish processes.
- Some autonomous driving companies modified ROS to RTOS

Let's dive a bit deeper into the details….

ROS Node

- A node is an executable that can communicate with each other in ROS
- Each node is a process. You can run multiple nodes in one machine.
- Your system (roomba, self driving car...) is a peer to peer network of ROS nodes. No central server. Data is passed between nodes.
- Master:
  1. register the node and find other nodes. Name service like DNS server
  2. contains parameter server, a shared, multi-variate dictionary that is accessible via network APIs. Nodes use this server to store and retrieve parameters at runtime.

ROS Topic

- A topic is a named bus through which nodes can exchange data.
- ROS topic is for unidirectional streaming
- The node that sends out data is called Publisher
- The node that receives data is called Subscriber
- One topic can have multiple publishers and multiple subscribers
- Use *rostopic list* to see all the topics that are alive

ROS Message

- ROS Nodes communicate through messages
- A message is a data structure, with typed fields
- The type of each field of the message is strictly defined (double, array of int32...)

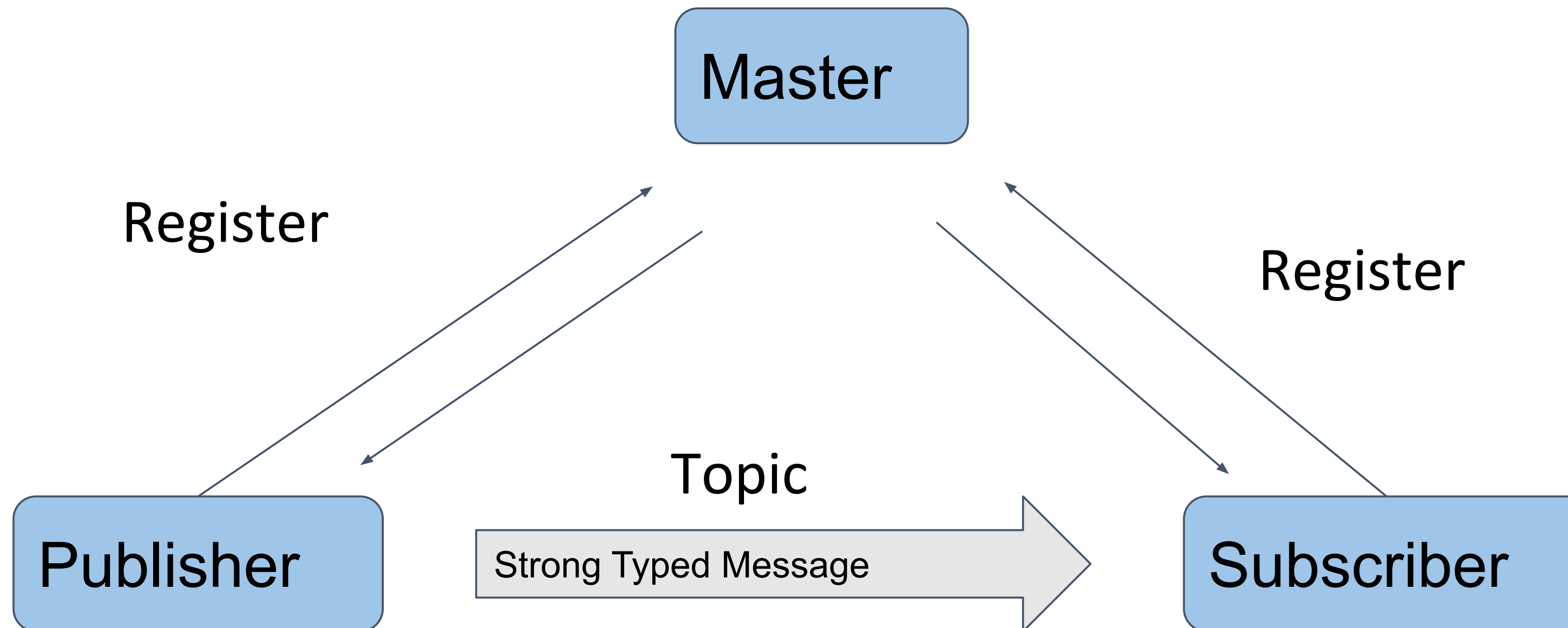
sensor_msgs/Image.msg

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

ROS Messaging Procedure :

1. Publisher node register at Master node, saying it wants to publish some data through the topic
2. Subscriber node register at Master node, saying it wants to subscribe to the topic
4. Master send back information to publisher and subscriber. Let them know the existence of each other
5. Publisher sends Message to all Subscribers
6. Subscribers receive Message, trigger some callback functions

ROS Messaging



Remember what we want? We got them all.

Example(from MP1 studentVision.py):

```
class lanenet_detector():
    def __init__(self):

        self.bridge = CvBridge()
        self.sub_image = rospy.Subscriber('camera/image_raw', Image, self.img_callback, queue_size=1)
        self.pub_image = rospy.Publisher("lane_detection/annotate", Image, queue_size=1)
        self.pub_bird = rospy.Publisher("lane_detection/birdseye", Image, queue_size=1)
```

Example:

```
def img_callback(self, data):

    try:
        # Convert a ROS image message into an OpenCV image
        cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
    except CvBridgeError as e:
        print(e)

    raw_img = cv_image.copy()
    mask_image, bird_image = self.detection(raw_img)

    # Convert an OpenCV image into a ROS image message
    out_img_msg = self.bridge.cv2_to_imgmsg(mask_image, 'bgr8')
    out_bird_msg = self.bridge.cv2_to_imgmsg(bird_image, 'bgr8')

    # Publish image message in ROS
    self.pub_image.publish(out_img_msg)
    self.pub_bird.publish(out_bird_msg)
```

ROS Bags

- .bag is a file type that stores message data. It can be recorded or played by rosbag tool.
- Record: rosbag subscribe to one or multiple rostopics, store all data in the topics into .bag file.
- Replay: rosbag play the .bag file, send out data through the topics(or new topics), just like the original node.

How to debug ROS?

- Shut down one node, change code and recompile while the rest of nodes are running
- Use rosbag to record data streams and replay them
- Use “Rostopic echo” to livestream the data in command terminal, or use Rviz to visualize the data

A lot more to explore...

- We have only discussed ROS in computational graph level, you can explore more on File system level and Community level.
- If you want to define the type of message by yourself, you need to use Catkin workspace to create your own package. Then use roslaunch to run it. We will see it in MP2.

What we just learned is only a tip of the iceberg. Please checkout the ROS Wiki and go over the tutorial for beginner’s level.