Introduction to Robotics

Course Introduction & Robotic Operating System

Johns Hopkins
Whiting School of Engineering

About the Instructor

Dr. Galen Mullins

- **Research Interests:**
  Artificial Intelligence, Machine Learning, Robotics, Autonomous Vehicles, Testing and Evaluation
- **Education**:
  - Ph.D. Mechanical Engineering, UMD
  - M.S. Physics, JHU
  - B.S. Mathematics, CMU
  - B.S Mechanical Engineering, CMU

Course Objectives

By the end of this course, the student will be able to:

- Compute coordinate transforms in 3 dimensions and describe a robotic system from different reference frames
- Describe how publish and subscribe architectures for robotic systems work
- Use kinematic equations of motion for wheeled robots to compute its trajectory
- Solve inverse kinematics for a robotic arm to determine how to move the end-effector to a specific point in space
- Select sensors for a robotic system and explain their function
- Explain how to use readings from an encoder and range sensor to determine how far a robot has traveled.
- Apply a Kalman filter to estimate the state of a simulated robot
- Explain the differences between different path planning algorithms and in what situations each one should be utilized
- Implement a collision avoidance algorithm for a wheeled robot
- Understand the principles behind robotic localization and how it can be applied to mapping environments.
- Write a reinforcement learning algorithm for a robotic navigation environment.

Prerequisites

- Basic understanding of Linear Algebra
- Ability to program in Python 3.5+
- Familiarity with Linux and terminal commands

Module Topics

1. Intro & Robotic Operating System
2. Coordinate Systems
3. Vehicle Kinematics
4. Controls
5. Kinematic Chains & Manipulators
6. Sensors
7. Obstacle Avoidance [**Mid-Term Project Due**]
8. Path Planning – Discrete Space
9. Path Planning - Continuous Space
10. State Estimation
11. Localization
12. Simultaneous Localization and Mapping
13. Robotic Learning
14. Class Project Presentations [**Final Project Due**]

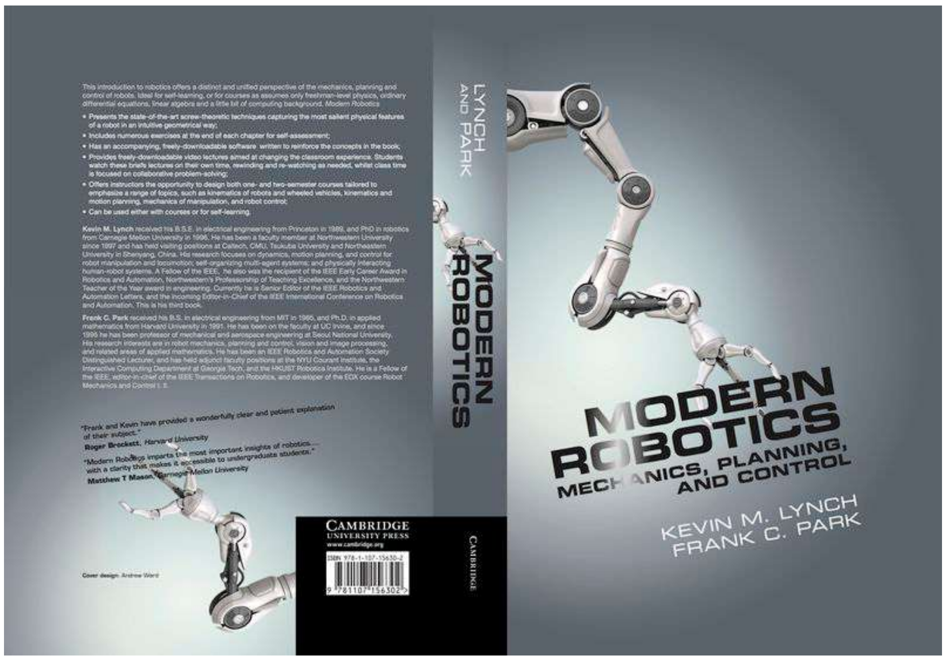
Course Policies

- **Collaborators** –
  - **Group Projects**
    - **OK –** Sharing code and assets between team members to create the final product
  - **Individual Projects**
    - **OK** – Asking why a package won't install or you are getting a strange segfault
    - **Not OK** – Programming simulation together, sharing code with other students, etc.
    - When in doubt ask instructor
- **Internet** – **Yes**
  - Modern programming often requires copious use of online resources and open source packages
  - If you use a library or 3rd party resource it must be cited. **Cannot** replace the core functionality being asked for in the assignment
    - If it seems questionable ask instructor
- **Copying** (from anywhere, including emails)
  - Without proper attribution is known as plagiarism
  - Forgetting to provide attribution is not an excuse
  - Plagiarism will earn an immediate 0 for the associated assignment
- **Late work**
  - Late assignments will be deducted 1 letter grade the first week it was missed. Will receive a zero afterward.
  - The final project **cannot** be turned in late without exceptional circumstances

Coursework & Grading

- **Participation – 10% of grade**
  - Join in class discussions
  - Come to class office hours
  - Make discussion posts on blackboard
- **5 Assignments – 50% of grade**
  - Each assignment will be to implement a technique learned in that week's lecture in Python 3.9
  - Some assignments will utilize programming a ROS node for a simulated robot.
  - Assignments may depend on code from previous assignments!
- **3 Group Assignments – 20% of grade**
  - Work as a team to extend concepts taught in class into code.
- **Mid-Term Project – 10% of grade**
  - Develop a ROS based robot to travel to user defined waypoints
- **Final Project – 10% of grade**
  - Develop a ROS based robot to complete a mission in Gazebo. 10%
  - Present design process and robot during final class. 10%

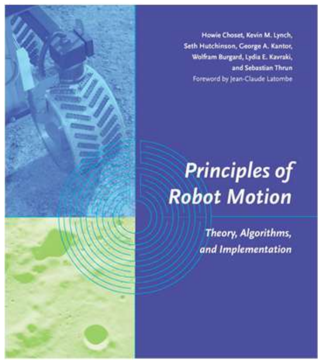
Course Book



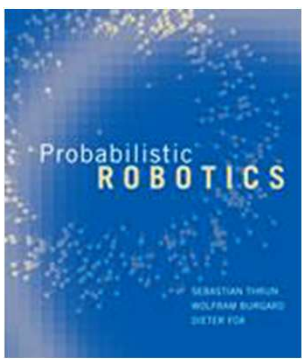
Northwestern University Center for Robotics and Biosystems Mechapedia

- Lynch, K. M., & Park, F. C. (2017). *Modern robotics: Mechanics, planning and control*. Cambridge University Press.

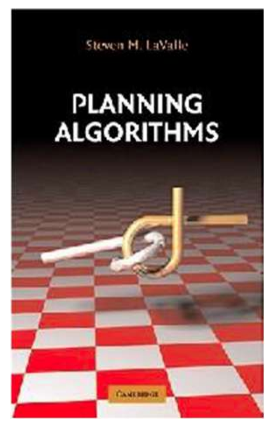
Other Books



Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G. A., & Burgard, W. (2005). *Principles of robot motion: Theory, algorithms, and implementations*. MIT Press.



Thrun, S., Burgard, W., & Fox, D. (2006). *Probabilistic robotics*. MIT Press.



LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press.

Free download available

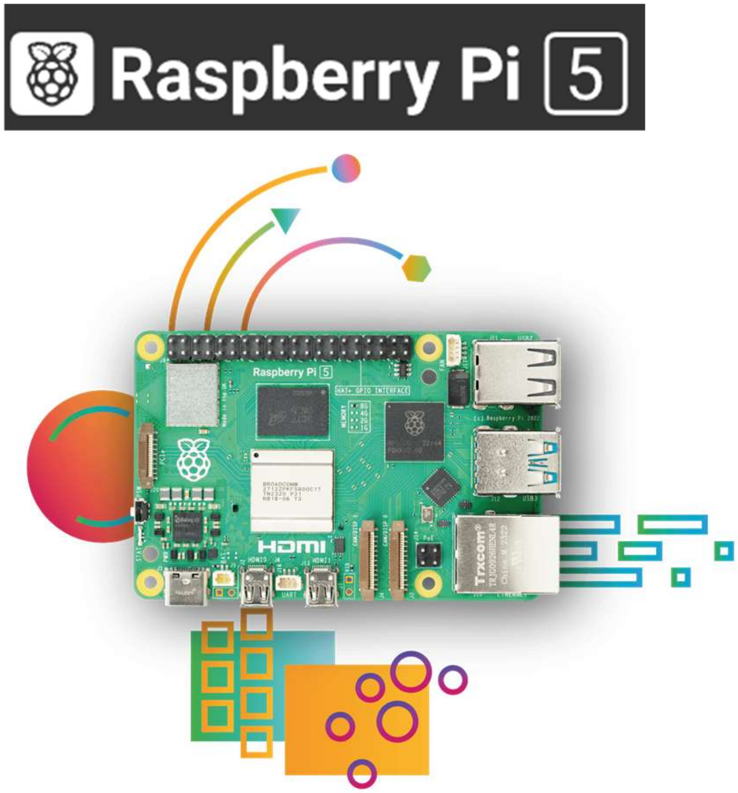
Course Hardware - Beta

This semester we will be experimenting with using the Raspberry Pi 5 as the "official" hardware for the course.

- Powerful enough to run ROS2 and Gazebo
- Relatively inexpensive (cheaper than many textbooks)
- Potential for integrating with hardware robot in the future

Using one is **not required** and instructions have not yet been posted.

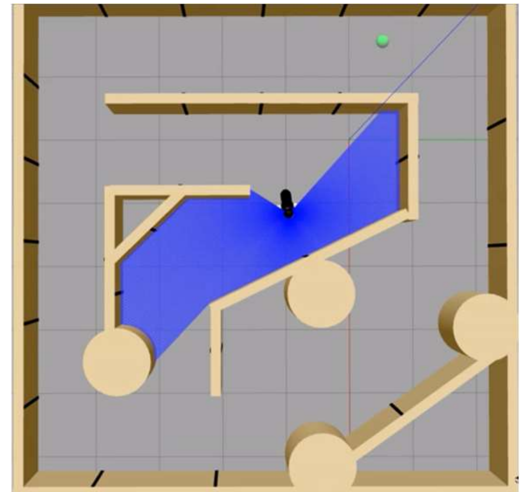
Documentation will be continuously updated throughout the semester.

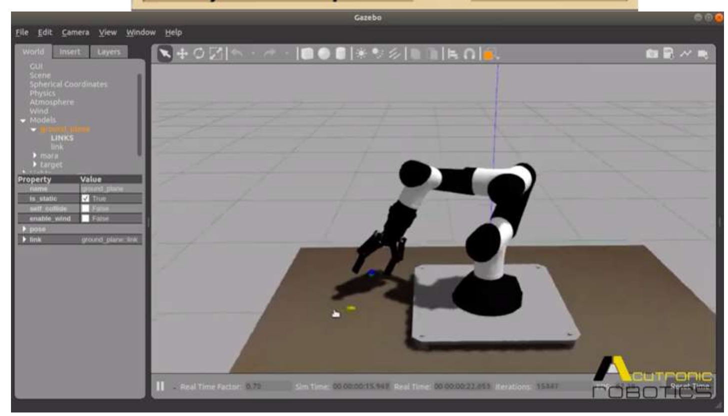


Course Project

Two Options

- Program a ROS robot to complete a mission in a simulated environment (Gazebo) designed by the instructor
  - Will leverage code developed throughout semester
- Design your own project
  - Proposals due by the 5th week of class
  - Pick a planning algorithm
  - Pick a robot or simulation
    - Needs to be one you can program in C/C++ or python
  - Must document all 3rd party code used

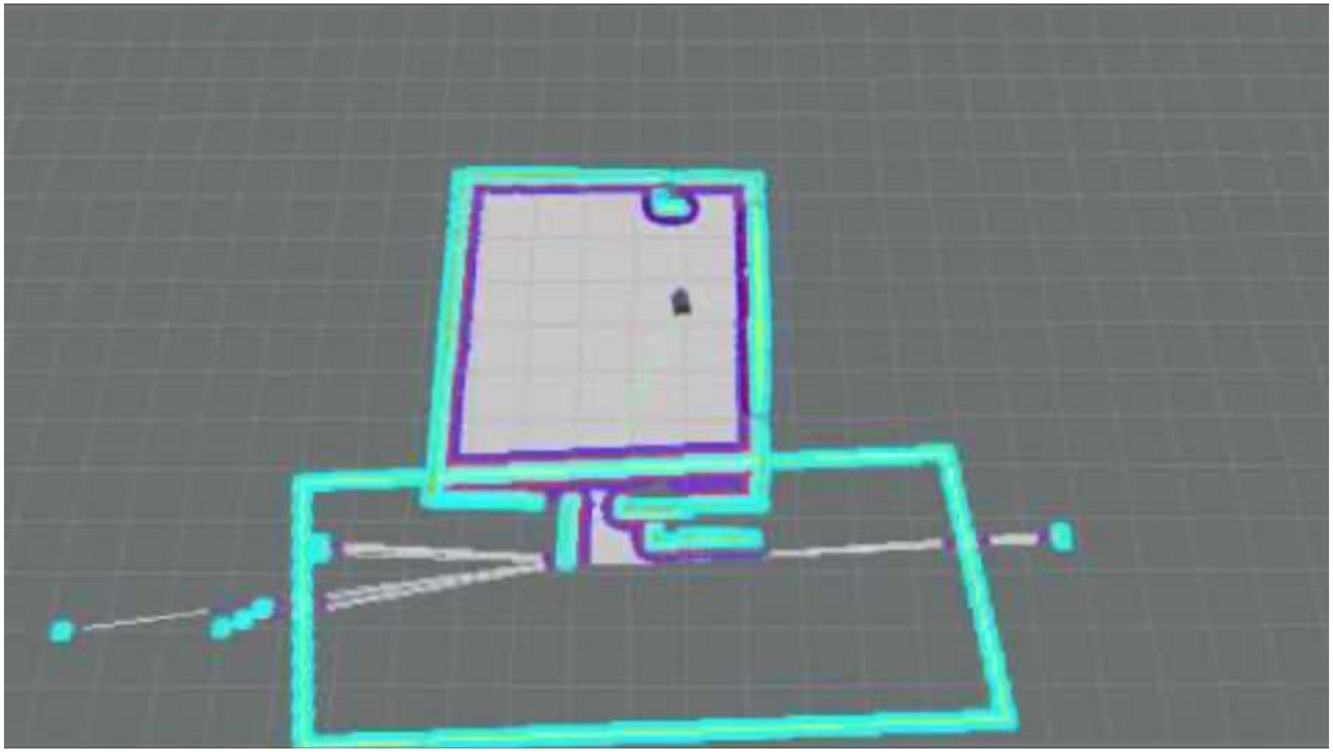




Some Project Ideas

- Program an arm to stack cubes while avoiding obstacles in the workspace
- Avoid moving obstacles (pedestrians, other vehicles)
- Explore an unknown environment to discover goal

Course Project Concept: Search and Rescue



Note: This is a working concept that will be updated as we proceed through the class. Project details and rubric will be finalized by the 5th class

Office Hours

Office Hours will be held virtually via Zoom.
Survey for best time is available on Canvas.

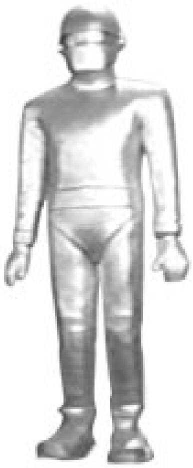

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

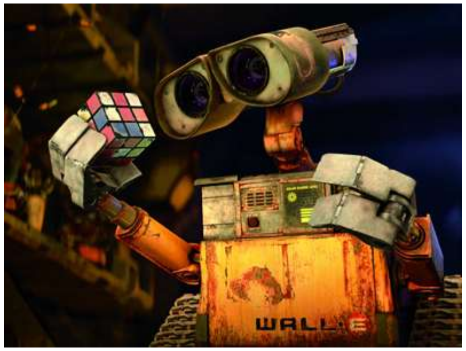
Introduction To Robotics

What is a Robot?

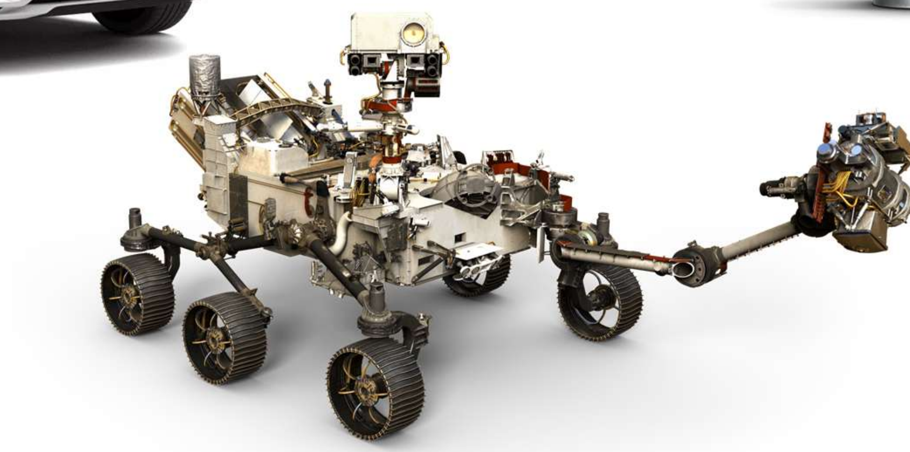
Johns Hopkins
Whiting School of Engineering

Robots!!! (in Fiction)

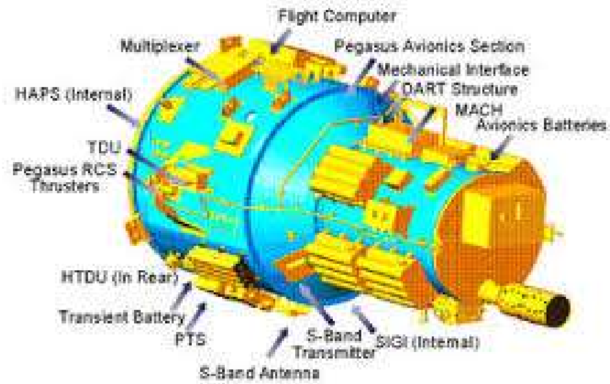




Robots in Reality



What is a Robot?



What is a Robot

- A physical machine that ….
  - Senses its environment,
  - Makes decisions,
  - Takes actions that affect its state and the state of the world around it

Origin of the Term

- Autonomous, automaton
  - Self-willed (Greek, auto+matos)

- Robot
  - Karel Capek in 192 play RUR (Rossum's Universal Robots)
    - Labor (Czech or Polish, robota)
    - Workman (Czech or Polish, Robotnik)

Three Laws of Robotics

1. A **robot** shall not harm a human, or by inaction allow a human to come to harm.

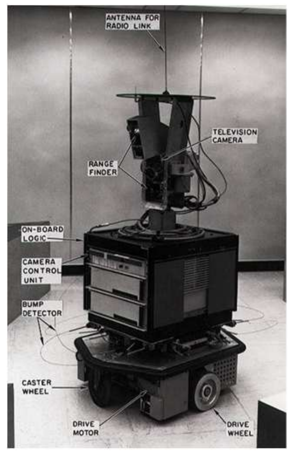
2. A **robot** shall obey any instruction given to it by a human. Except when such orders would conflict with the first law.

3. A **robot** must protect its own existence as long as such protection does not conflict with the first or second law.

Asimov, I. (1942). *Runaround*.

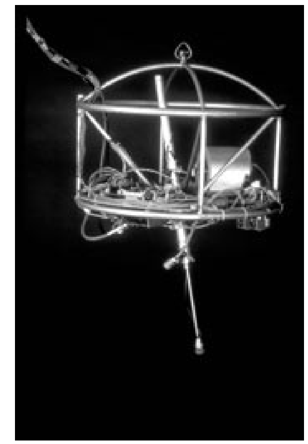
Robot Hall of Fame (1)

1961
Unimate
The first industrial robot.
General Motors



1966
Shakey
The first mobile robot with
onboard decision making

1978
SCARA
Revolutionary pick-and-place
Robot arm

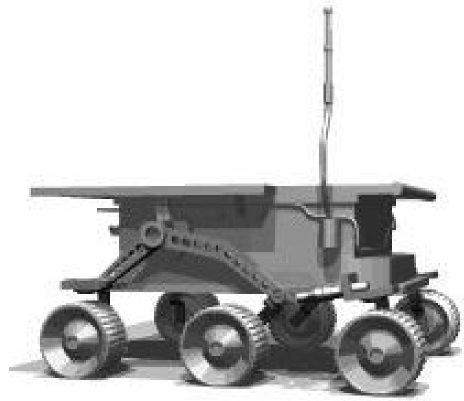


1980
Raibert Hopper
Breakthrough in dynamic
locomotion

Hopping Robots

Robot Hall of Fame (2)

1995
NavLab 5
First cross-country trip
In an autonomous vehicle



1997
Sojourner
The first wheeled
vehicle on mars

1998
AIBO
The first robot pet.
Robocup star!

1990s
PACKBOT
First widely deployed
Bomb defusing robot

Robot Hall of Fame (3)

1998
Lego Mindstorms
Robot platform for all ages

2000
Da Vinci Surgery System
First robot approved for
General laproscopic surgery*

2002
Roomba
First commercially available
robot vacuum

2005
Big Dog
The gold standard for
quadrapedal motion

Possible New Nominees?

2013
ATLAS Robot
Increasingly agile humanoid robot

2010
Parrot AR Drone

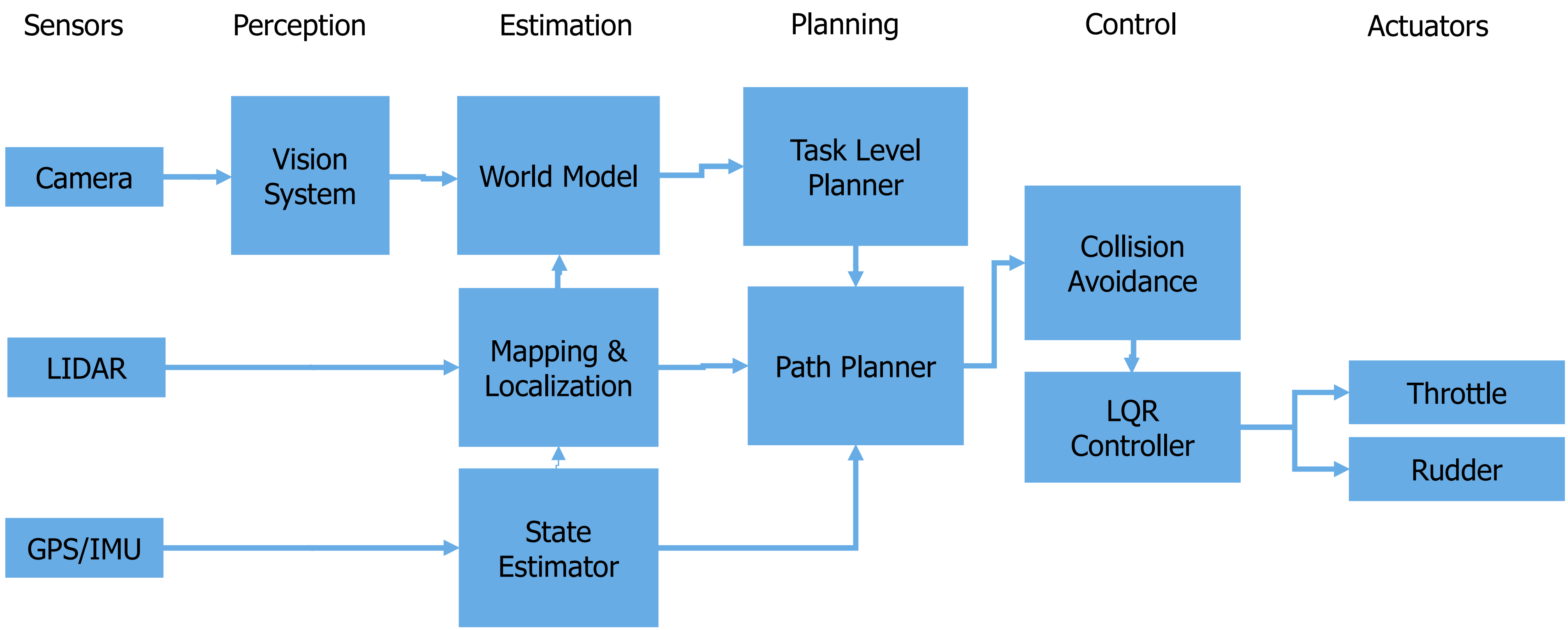
2013
Phantom Quadcopter

Commercial quadcopters for consumers

Ferdinand – JHUAPL 1964

Classical / Hierarchical Paradigm

Example Robotics Architecture




JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

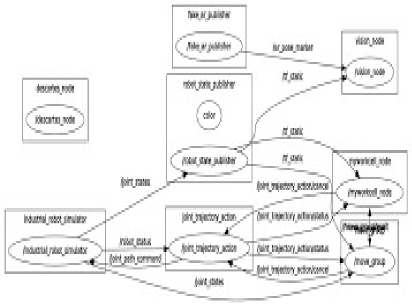
Introduction To Robotics

Robotic Operating System

Slides adapted from: ETH Zurich – Programming for Robotics – Fankhauser, et. al

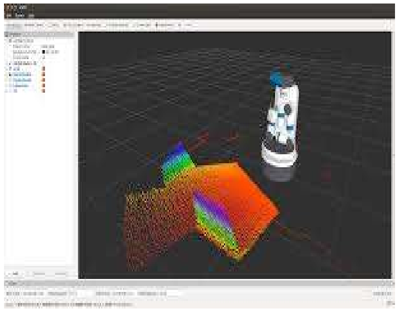
What is ROS?

ROS = Robotic Operating System



Plumbing

Process Management
Inter-process communication
Device Drivers

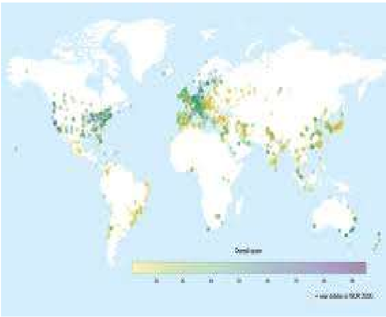


Tools

Simulation
Visualization
Graphical User Interface

Capabilities

Control
Planning
Perception
Mapping
Manipulation



Ecosystem

Package organization
Software distribution
Documentation
Tutorials

Slides adapted from: ETH Zurich – Programming for Robotics – Fankhauser, et. al

History of ROS

- Originally during the early 2000's at the Stanford Artificial Intelligence Laboratory
- Group went onto form Willow Garage in 2007
- Since 2013 managed by OSRF (Open Source Robotics Foundation)
- Today used by many robots, universities and companies
- De facto standard for robot programming

ros.org

Slides adapted from: ETH Zurich – Programming for Robotics – Fankhauser, et. al

ROS Philosophy

- **Peer to peer**
  - Individual programs communicate over defined API (ROS *messages*, *services*, etc.).
- **Distributed**
  - Programs can be run on multiple computers and communicate over the network.
- **Multi-lingual**
  - ROS modules can be written in any language for which a client library exists (C++, Python, MATLAB, Java, etc.).
- **Light-weight**
  - Stand-alone libraries are wrapped around with a thin ROS layer.
- **Free and open-source**
  - Most ROS software is open-source and free to use.

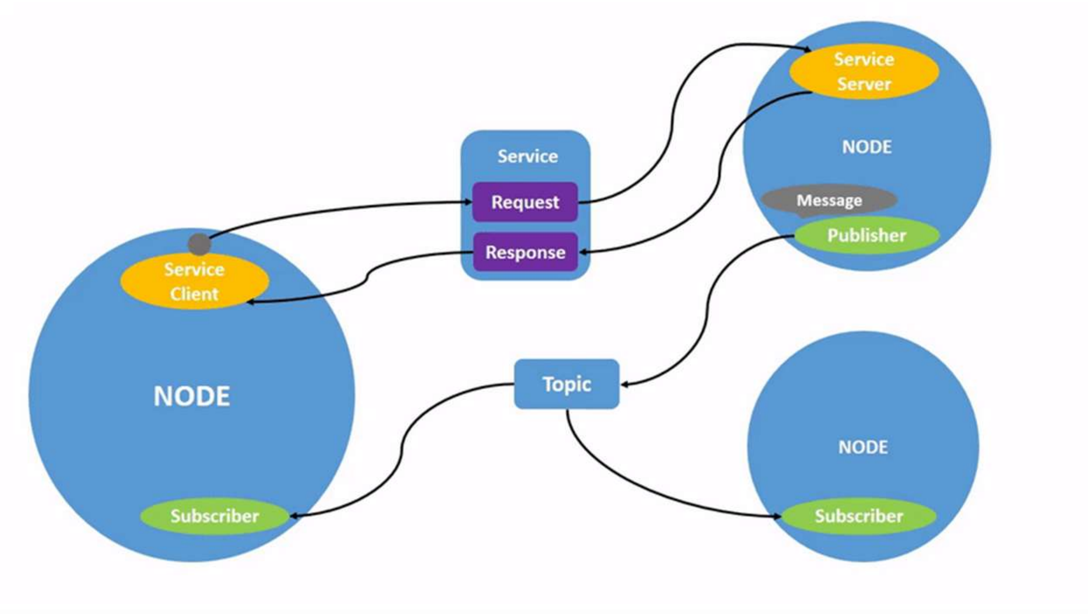
Slides adapted from: ETH Zurich – Programming for Robotics – Fankhauser, et. al

ROS Versus ROS2

Features	ROS	ROS2
Platforms	Tested on Ubuntu Maintained on other Linux flavors as well as OS X	ROS2 is currently being CI tested and supported on Ubuntu Xenial, OS X El Capitan as well as Windows 10
C++	C++03 // don't useC++11 features in its API	Mainly uses C++11 Start and plan to use C++14 & C++17
Python	Target Python 2	>= Python 3.5
Middleware	Custom serialization format (transport protocol + central discovery mechanism)	Currently all implementations of this interface are based on the DDS standard.
Unify duration and time types	The duration and time types are defined in the client libraries, they are in C++ and Python	In ROS2 these types are defined as messages and therefore are consistent across languages.
Components with life cycle	In ROS every node usually has its own main function.	The life cycle can be used by tools like roslaunch to start a system composed of many components in a deterministic way.
Threading model	In ROS the developer can only choose between single-threaded execution or multi-threaded execution.	In ROS2 more granular execution models are available and custom executors can be implemented easily.
Multiple nodes	In ROS it is not possible to create more than one node in a process.	In ROS2 it is possible to create multiple nodes in a process.
roslaunch	In ROS roslaunch files are defined in XML with very limited capabilities.	In ROS2 launch files are written in Python which enables to use more complex logic like conditionals etc.

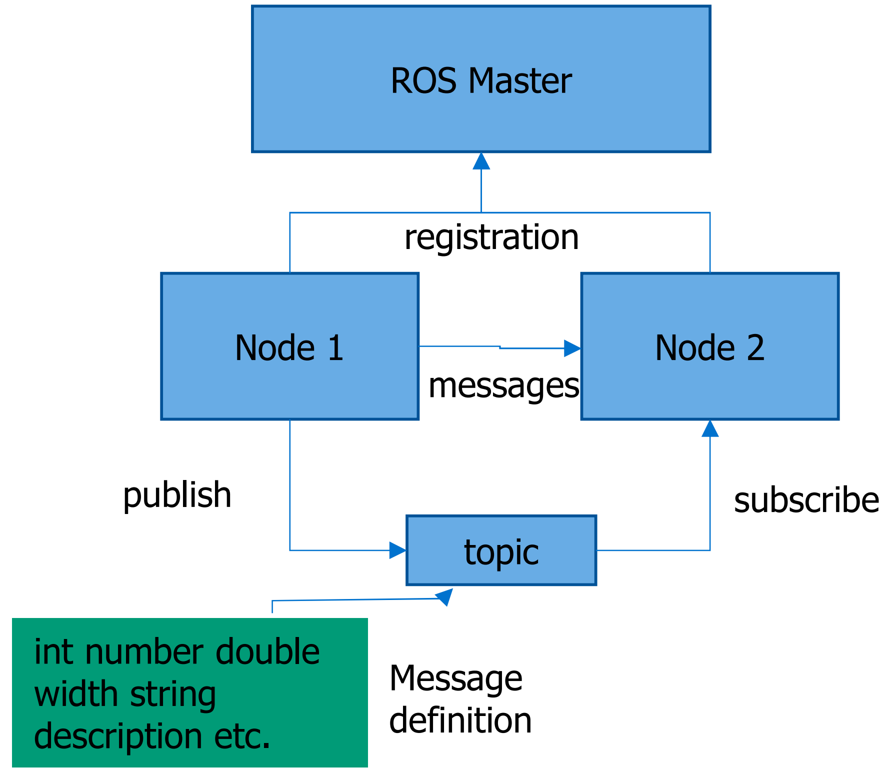
ROS2 Architecture

- Each node devoted to a single purpose
  - Laser range finder
  - Wheel motors
  - Mapping
  - State Estimation
- Each node can send and receive data to other nodes via topics, services, actions, or parameters.



ROS 1 Architecture

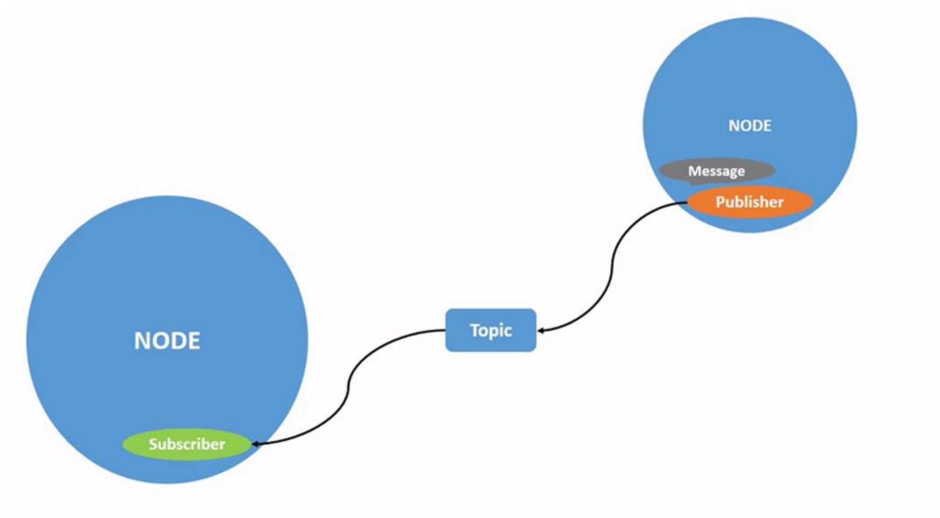
- Manages the communication between nodes (processes)
- Every node registers at startup with the master
- Master sets up peer-to-peer communications between nodes who subscribe to each others topics
- Intra-robot communication only



Slides adapted from: ETH Zurich – Programming for Robotics – Fankhauser, et. al

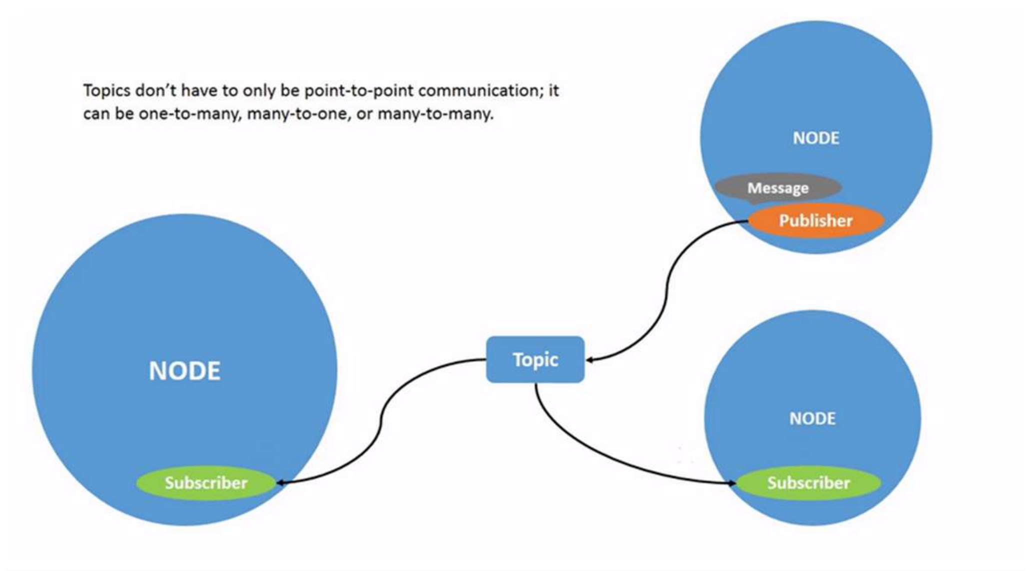
ROS2 Topics

ROS 2 breaks complex systems down into many modular nodes.
Topics are a vital element of the ROS graph that act as a bus for nodes to exchange messages.



ROS2 Topics (cont.)

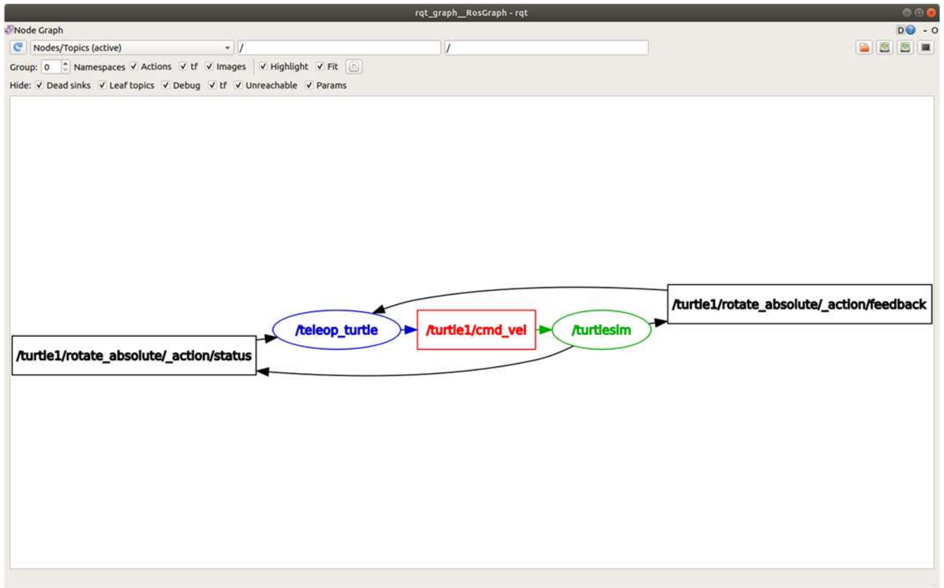
A node may publish data to any number of topics and simultaneously have subscriptions to any number of topics.



RQT Graph

rqt_graph is a useful tool for understanding how your ROS nodes are communicating with one another

```
> ros2 run turtlesim turtlesim_node
> ros2 run turtlesim turtle_teleop_key
> rqt_graph
```



Nodes in ROS2

Node commands have changed in ROS2

- View topics with

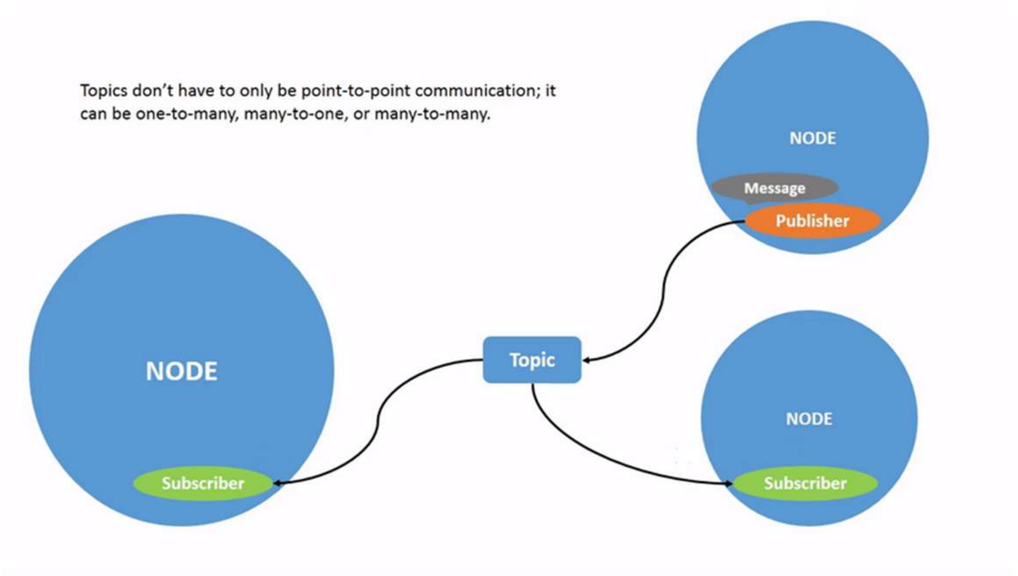
```
> ros2 run node_name
```

- See active nodes with

```
> ros2 node list
```

- Retrieve information about a node with

```
> ros2 node info node_name
```



Topic Commands in ROS2

- View available topics

  > ros2 topic list

- Get information about the topic

  > ros2 topic info */topic/name*

- See the rate at which a topic is being published

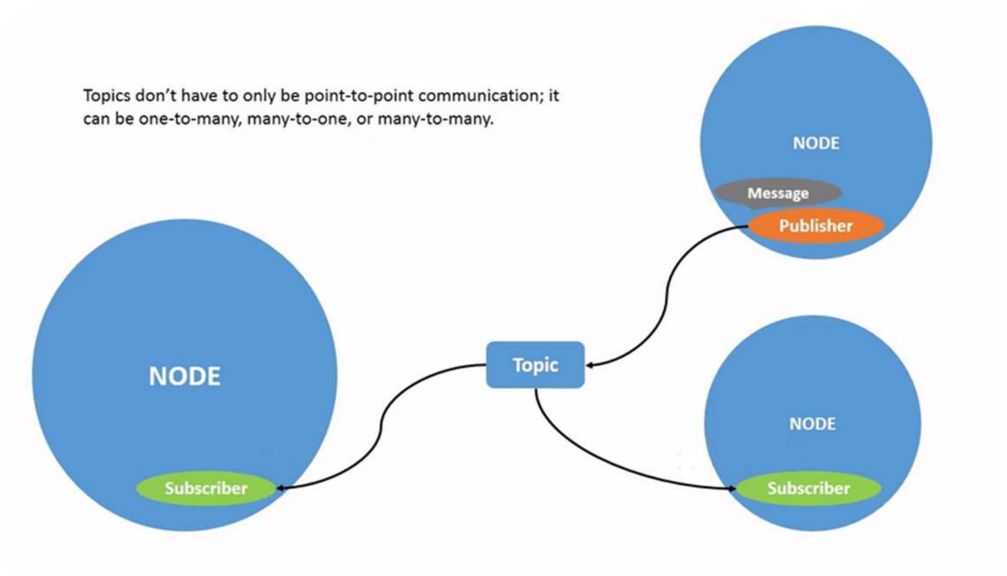
  > ros2 topic hz */topic/name*

- View the data being published

  > ros2 topic echo */topic/name*

- Publish data to a topic

  > ros2 topic pub –-rate <hz> */topic/name msg_type "{json format data}"*



ROS Messages
Pose Stamped Example

geometry_msgs/Point.msg

```
float64 x
float64 y
float64 z
```

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
```

geometry_msgs/PoseStamped.msg

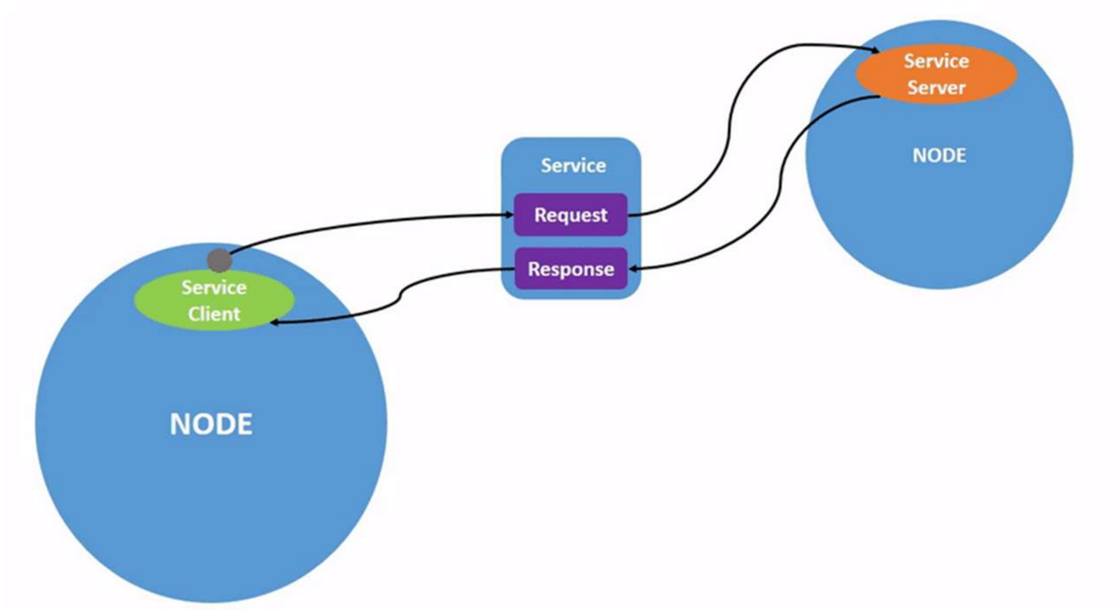
```
std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
geometry_msgs/Pose pose
geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
geometry_msgs/Quaternion
orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

Slides adapted from: ETH Zurich – Programming for Robotics – Fankhauser, et. al

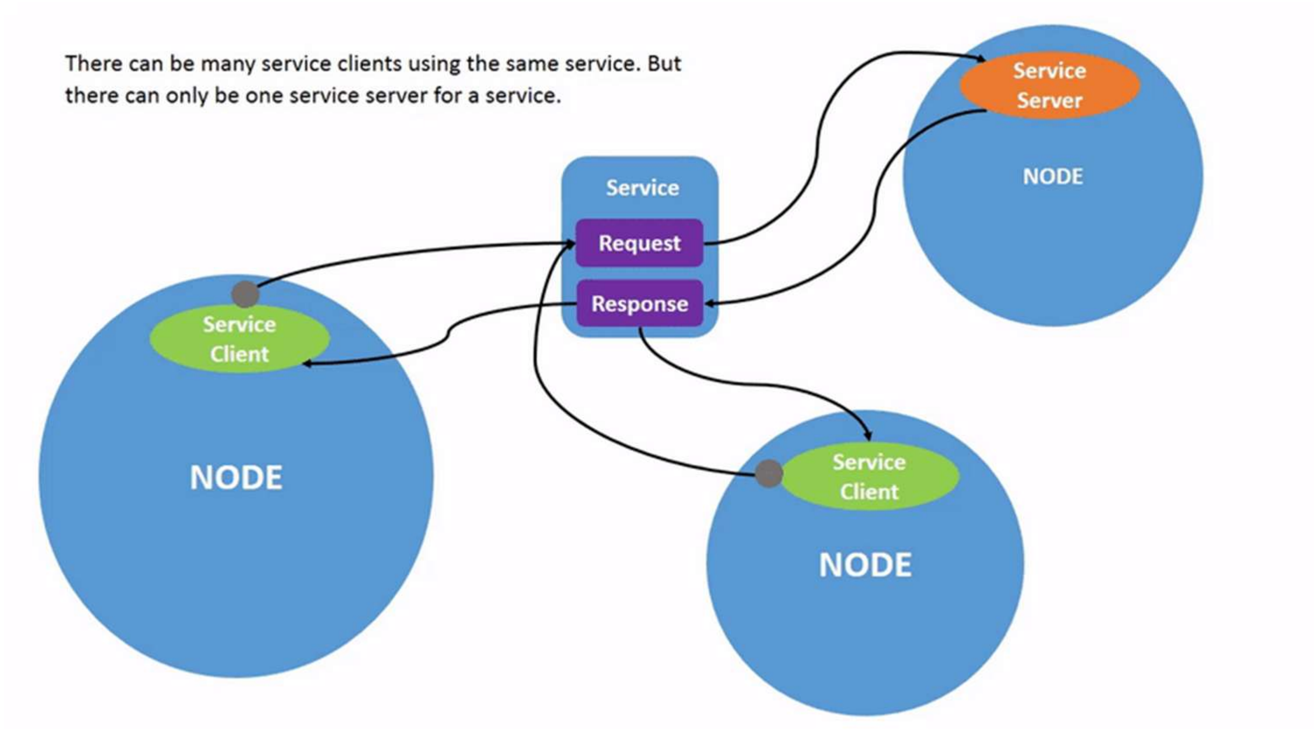
ROS2 Services

Services are based on a call-and-response model, versus topics' publisher-subscriber model.

While topics allow nodes to subscribe to data streams and get continual updates, services only provide data when they are specifically called by a client.



ROS2 Services (cont.)



Service Commands in ROS2

- View available services

> > ros2 service list

- Get information about the service

> > ros2 service type */topic/name*

- Find services with a particular type

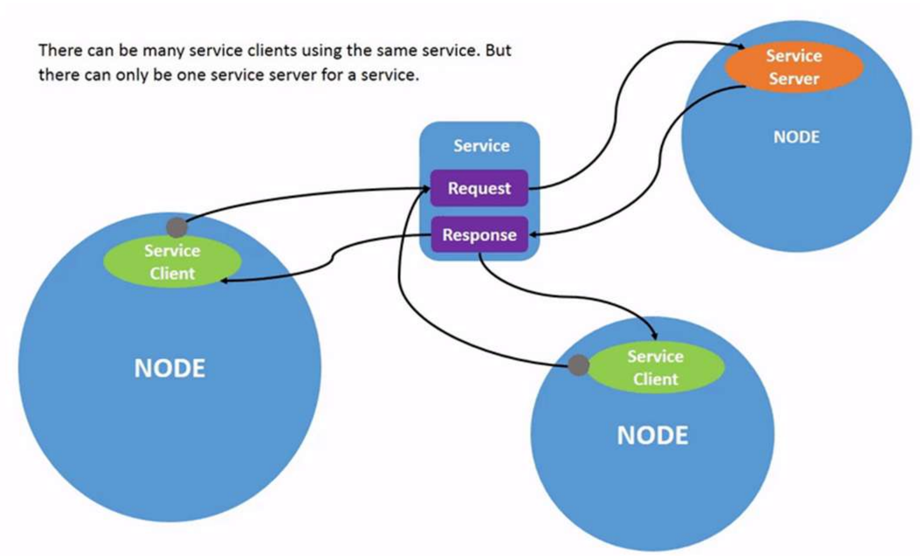
> > ros2 service find *<type_name>*

- See the arguments for a service

> > ros2 interface show <type_name>.srv

- Call a service

> > ros2 service call <service_name> <service_type> <arguments>



ROS2 Discovery Process

- When a node is started, it advertises its presence to other nodes on the network with the same ROS domain (set with the ROS_DOMAIN_ID environment variable).
- Other Nodes respond to this advertisement with information about themselves so that the appropriate connections can be made and the nodes can communicate.
- Nodes periodically advertise their presence so that connections can be made with new-found entities, even after the initial discovery period.
- Nodes advertise to other nodes when they go offline.
- Nodes will only establish connections with other nodes if they have compatible Quality of Service settings.

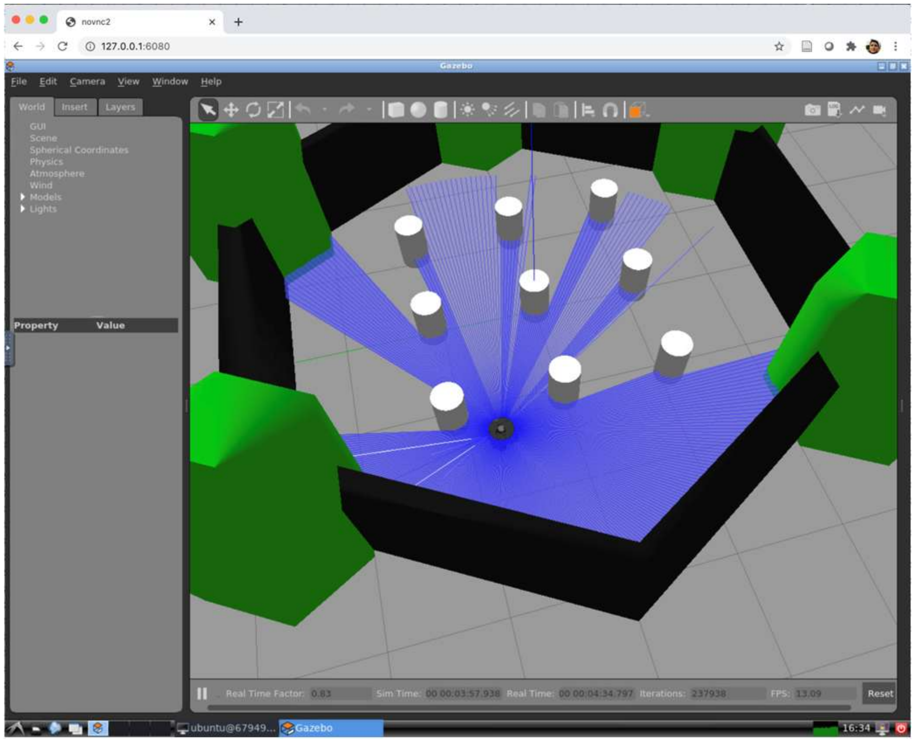
Setting Up Your ROS2 Environment

- Local Install
  - Recommend Ubuntu 20.04
  - Instructions here https://index.ros.org/doc/ros2/Installation/Foxy/
  - Latest ubuntu instructions also good: https://ubuntu.com/blog/simulate-the-turtlebot3
- Virtual Machine
  - Same as local install
  - VirtualBox (free) , VMWare (Paid)
- Docker Image (Will be using this example in class)
  - https://github.com/Tiryoh/docker_ros2-desktop-vnc
  - Installation instructions in Class handout.

Notes for Using Docker

- Gazebo for ROS 2 Humble has issues with Arm64 chips (which includes Mac M1/2/3 chips) if you have one of these you will need to use Foxy instead BUT the latest Foxy image has not been stable since it was deprecated. Please use a tagged image such as
  - foxy-20230327T0226

Simulating TurtleBot3



Assignment 1

- Set up a ROS environment for use in the class
- Run the Assignment 1 demo
- Fill out information sheet
    - Your level of experience with different programming languages & mathematics
    - What you are most interested in getting out of the class
    - When would office hours work best for you?
    - Your current ROS setup (e.g. Local Install, Virtual Machine, Docker Image, or Cloud)
    - Screenshot of the final state of the demo.

References

- LaVallе, S. M. (2006). *Planning algorithms*. Cambridge University Press. https://lavalle.pl/planning/
- Northwestern University Center for Robotics and Biosystems. (n.d.). *Modern robotics* [image]. Mechapedia. https://hades.mech.northwestern.edu/index.php/Modern_Robotics


JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING