Middleware, Software Architecture, and ROS in Duckietown

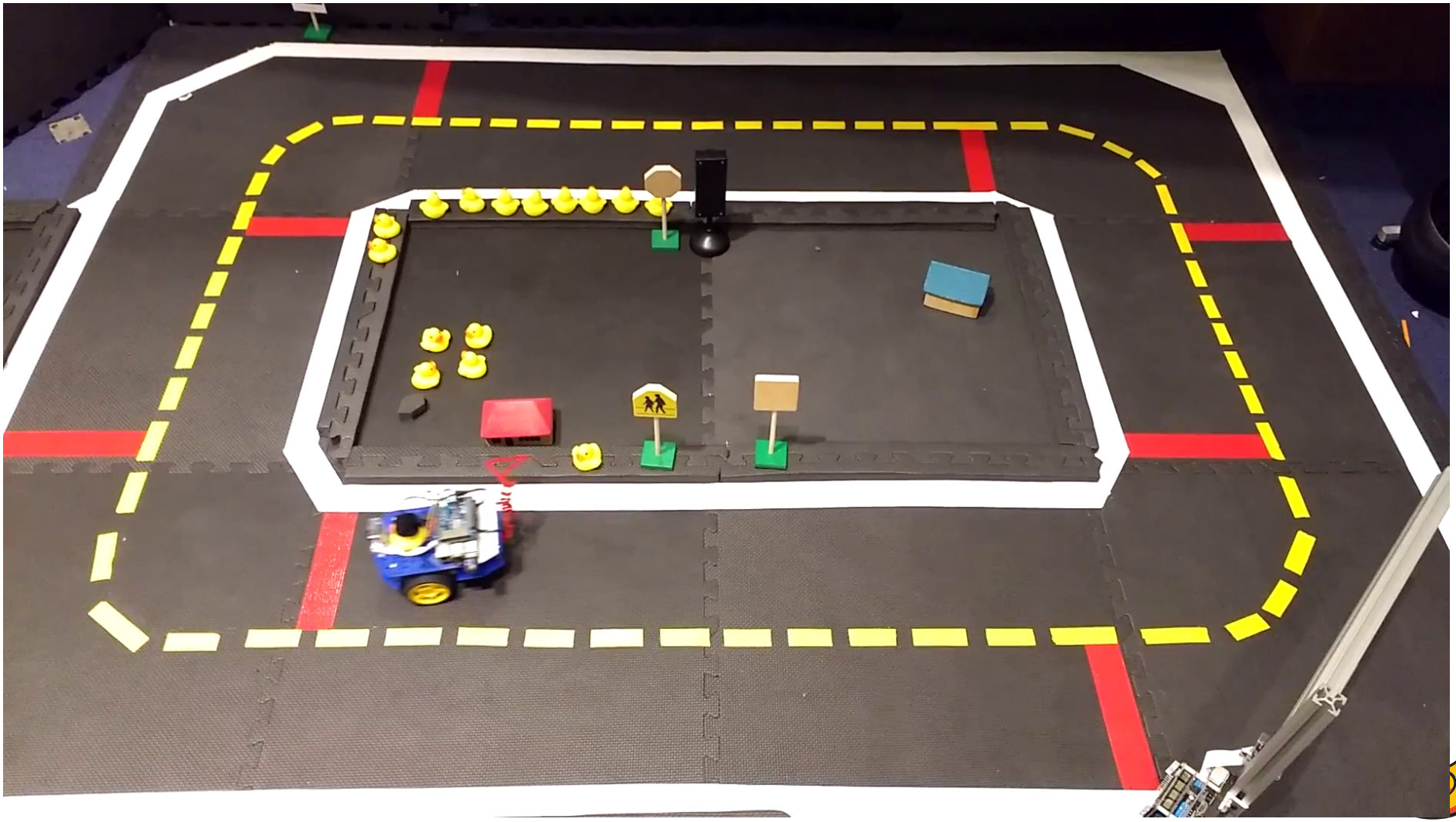
Shih-Yuan Liu and Michael "Misha" Novitzky

Duckietown, MIT

February 17th 2016

MIT 2.166
DUCKIETOWN

Duckiebot Lane Following in Action



MIT 2.166 DUCKIETOWN

Team Success

- No man is an island
- Shih-Yuan (glue)
- Team members: Liam, Andrea, Hang, Steven, Changhyun, etc

MIT 2.166 DUCKIETOWN

Historical Note on Software for Robots

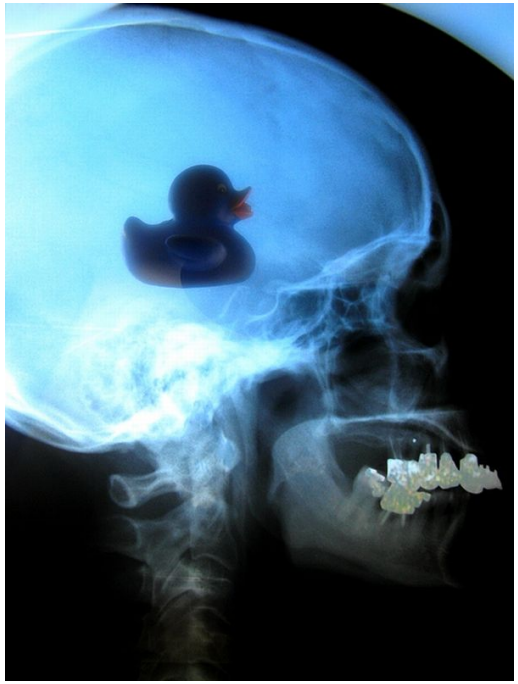
- Want to advance state of the art
- How parts talk to each other?
- Build infrastructure
- Throw away

MIT 2.166 DUCKIETOWN

What problems in software design?

Thought process:

- One giant loop?
- Many small pieces?
- Reusable?



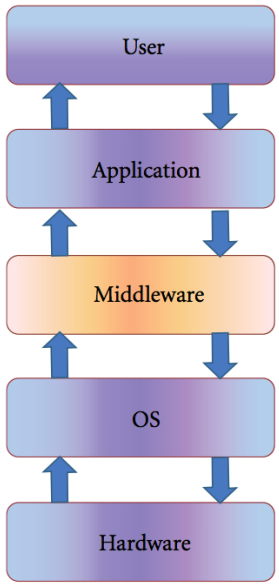
Overview

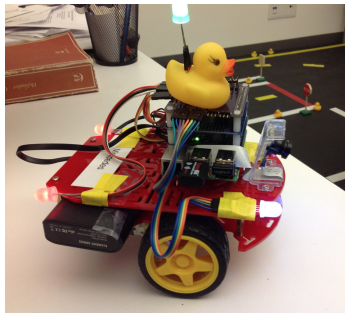
Enter the Middleware

General Definition: software that connects software components. A layer that lies between the operating system and the applications on each side of a distributed computer network.

An abstraction:

- Not the operating system
- Not the application
- Right in the middle





MIT 2.166 DUCKIETOWN

What do we gain?

Every middleware must have:

- Abstraction from hardware
- Hide communication

Good middlewares have:

- Logging/Playback tools
- Real-time analysis

Resulting benefits:

- Portable
- Reusable
- Saves time and money

MIT 2.166 DUCKIETOWN

Many Robotics Middlewares!

ROS, LCM, MOOS, JAUS, Orcos, Pyro, Player, Orca, Mira, OpenRTMaist, ASEBA, MARIE, RSCA, MRDS, OPROS, CLARAty, SmartSoft, ERSP, Webots, RoboFrame

Small Middleware Comparison

	ROS	LCM	MOOS
Communication Structure	name-/parameter server	decentralized	central database
Communication Mechanism	intra-process, TCP, UDP	UDP multicast	TCP
Data Transport	publisher / subscriber, RPC	publisher / subscriber	store / fetch
Message Types	IDL using PODs	IDL using PODs	string, double
Supported Languages	C++, Java, Python,...	C++ , Java, C#, Python, ...	C++, Java
Supported Platforms	Linux, OS X (partial), and Win (partial)	Linux, Win, OS X	Linux, OS X

Intended Learning Outcome

Learning Objectives

- Describe the value of middleware for robotics software development
- Answer the question: "What is your favorite robotics middleware and why is it ROS?"
- Draw computation graph for simple robotic system using publish/subscribe model
- Explain important ROS concepts such as node, topic, message, and parameters
- Use important ROS command-line tools
- Write a simple ROS node by following code examples

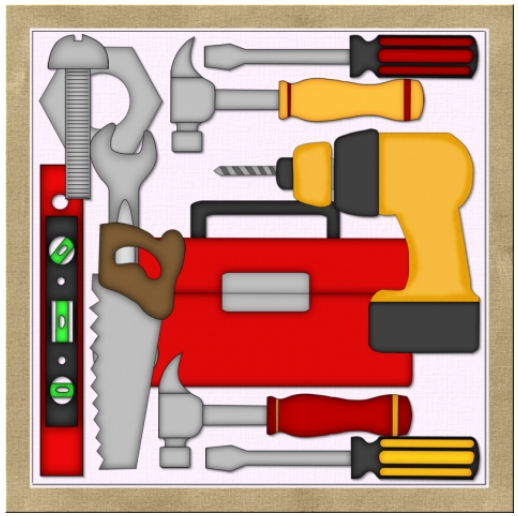
Why ROS?

- Communications infrastructure
- Recording and playback of messages
- Remote procedure calls
- Distributed parameter system
- Huge community
- Sought after skills
- Great for prototyping

MIT 2.166
DUCKIETOWN

Which Tools?

- command-line tools
- graphical (rviz, rqt)



MIT 2.166 DUCKIETOWN

Intended Learning Outcome

Learning Objectives

- Describe the value of middleware for robotics software development
- Answer the question: "What is your favorite robotics middleware and why is it ROS?"
- Draw computation graph for simple robotic system using publish/subscribe model
- Explain important ROS concepts such as node, topic, message, and parameters
- Use important ROS command-line tools
- Write a simple ROS node by following code examples

Overview

Intended Learning Outcome

Learning Objectives

- Describe the value of middleware for robotics software development
- Answer the question: "What is your favorite robotics middleware and why is it ROS?"
- Draw computation graph for simple robotic system using publish/subscribe model
- Explain important ROS concepts such as node, topic, message, and parameters
- Use important ROS command-line tools
- Write a simple ROS node by following code examples

Meet the Team: Shih-Yuan Liu

How to pronounce my name?

sh-U-when

Where am I from?

- Taiwan
- U.C. Berkeley

What am I doing here at MIT?

- Postdoc with Prof. Jonathan How (Aerospace Controls Lab)
- Control and Planning for Autonomous Vehicles

What am I doing here in Duckietown?

- ROS Man
- Control

Meet the Team: Shih-Yuan Liu

How to pronounce my name?

sh-U-when

Where am I from?

- Taiwan
- U.C. Berkeley

What am I doing here at MIT?

- Postdoc with Prof. Jonathan How (Aerospace Controls Lab)
- Control and Planning for Autonomous Vehicles

What am I doing here in Duckietown?

- ROS Man
- Control

Meet the Team: Shih-Yuan Liu

How to pronounce my name?

sh-U-when

Where am I from?

- Taiwan
- U.C. Berkeley

What am I doing here at MIT?

- Postdoc with Prof. Jonathan How (Aerospace Controls Lab)
- Control and Planning for Autonomous Vehicles

What am I doing here in Duckietown?

- ROS Man
- Control

Meet the Team: Shih-Yuan Liu

How to pronounce my name?

sh-U-when

Where am I from?

- Taiwan
- U.C. Berkeley

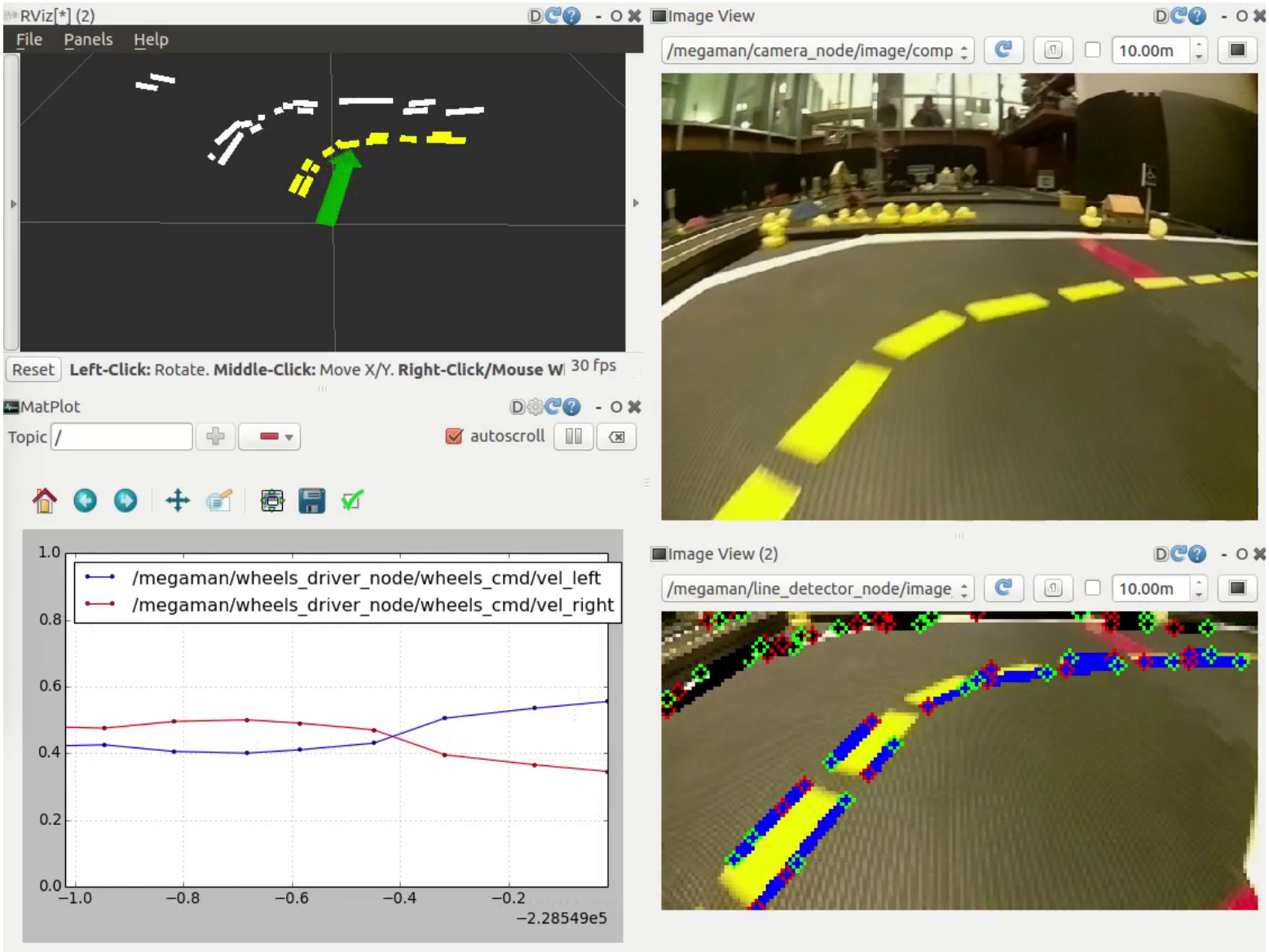
What am I doing here at MIT?

- Postdoc with Prof. Jonathan How (Aerospace Controls Lab)
- Control and Planning for Autonomous Vehicles

What am I doing here in Duckietown?

- ROS Man
- Control

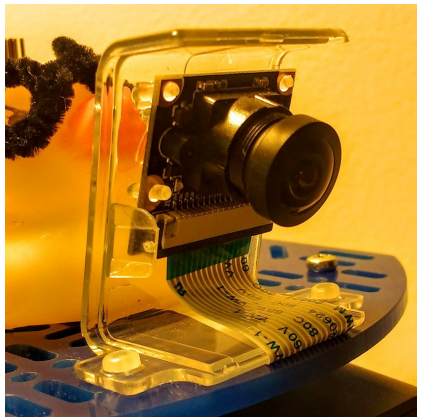
Behind the Curtain Look at Lane Following



MIT 2.166 DUCKIETOWN

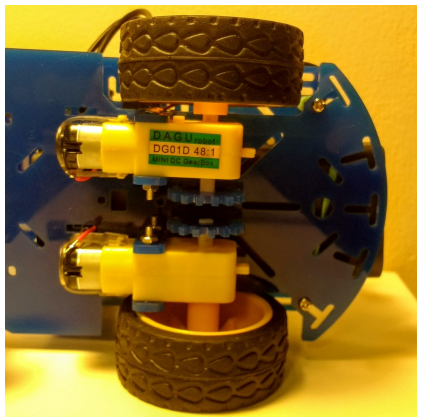
Computation Graph: Publish/Subscribe

Sensors



Sensors: Camera

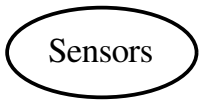
Actuators



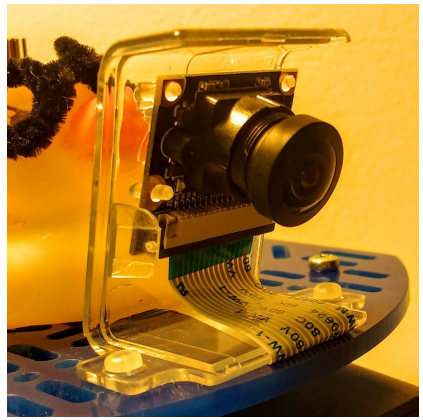
Actuators: Wheel Motors

MIT 2.166 DUCKIETOWN

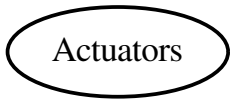
Computation Graph: Publish/Subscribe



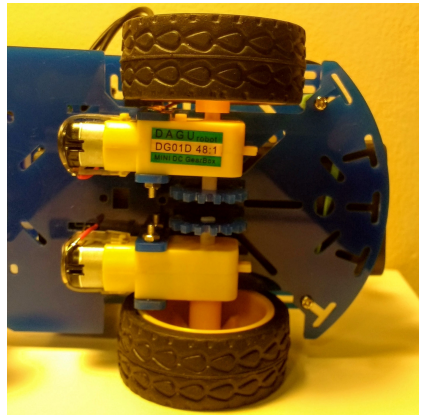
Node



Sensors: Camera



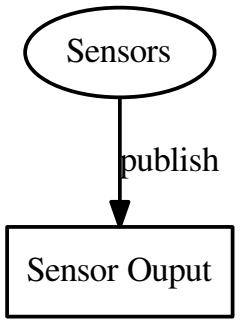
Node

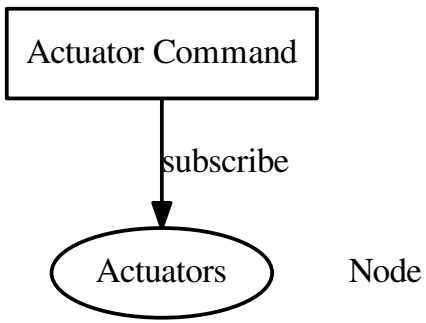


Actuators: Wheel Motors

MIT 2.166 DUCKIETOWN

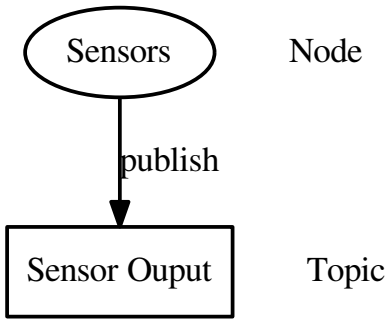
Computation Graph: Publish/Subscribe

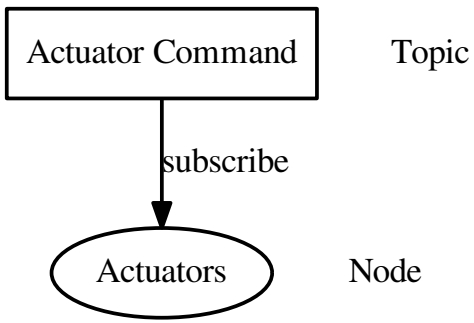




MIT 2.166 DUCKIETOWN

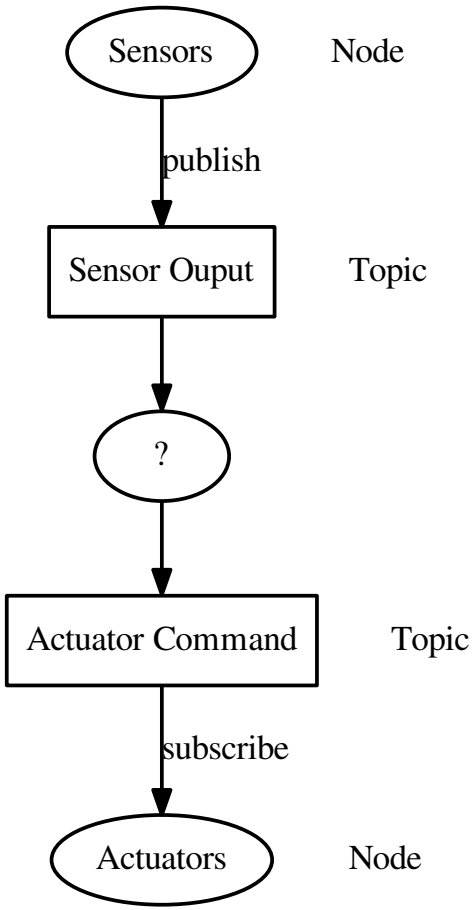
Computation Graph: Publish/Subscribe





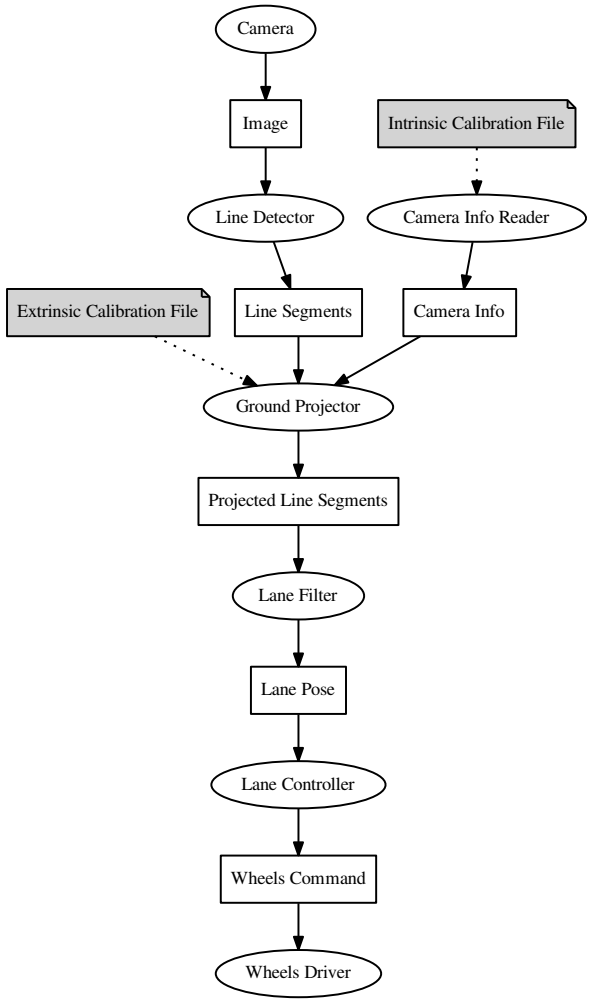
MIT 2.166 DUCKIETOWN

Computation Graph: Publish/Subscribe



MIT 2.166 DUCKIETOWN

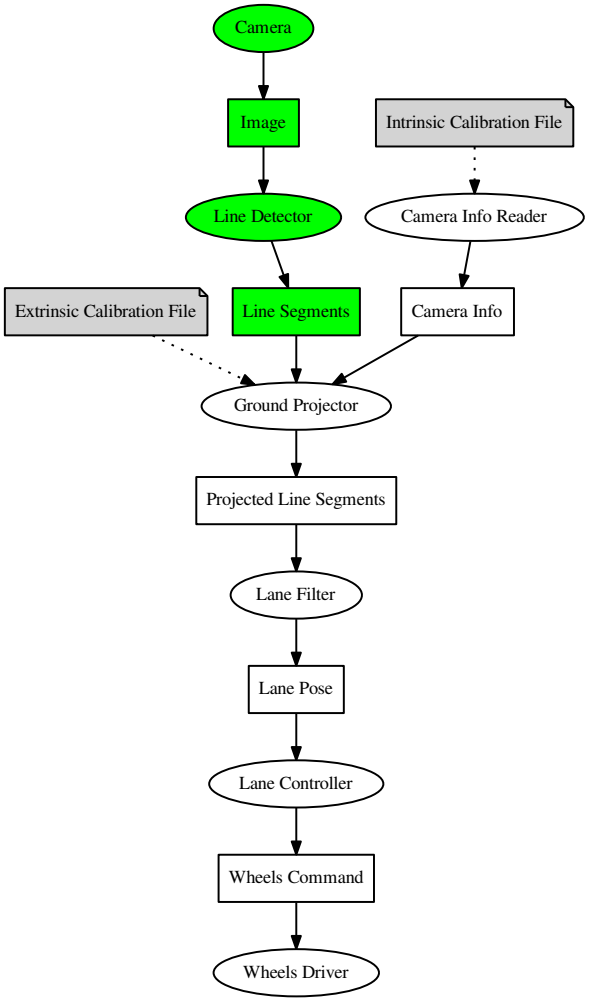
Example: Lane Following

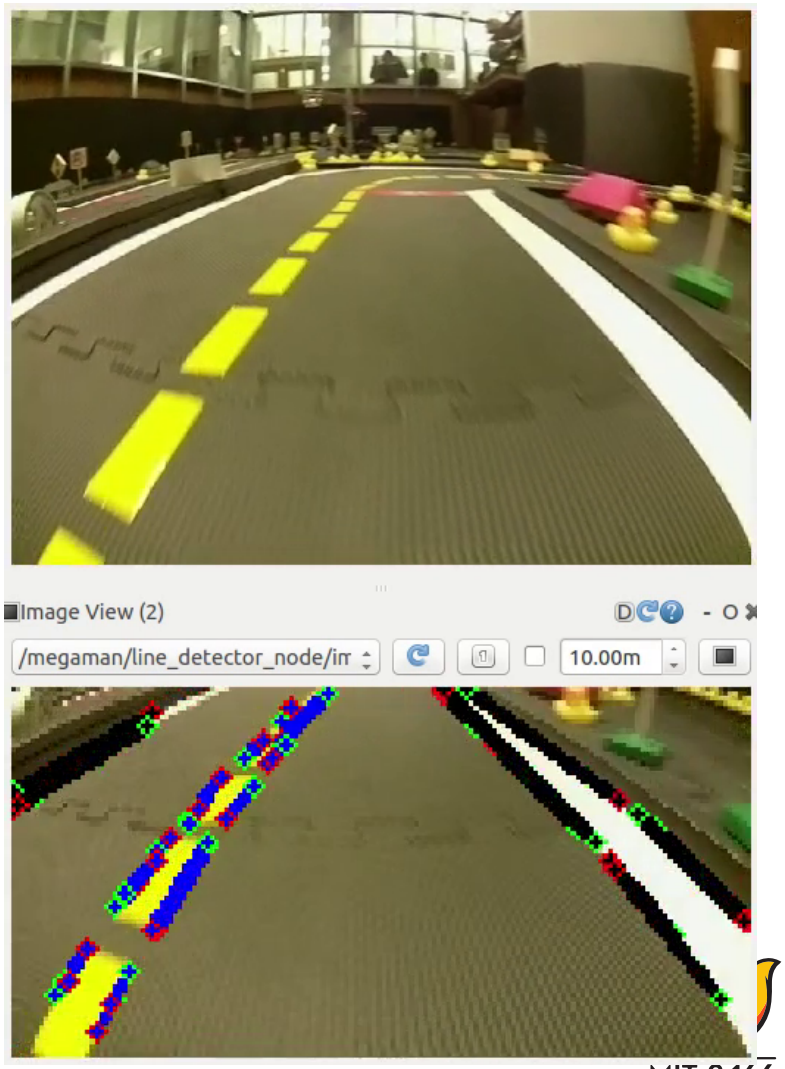


MIT 2.166 DUCKIETOWN

Line Detector

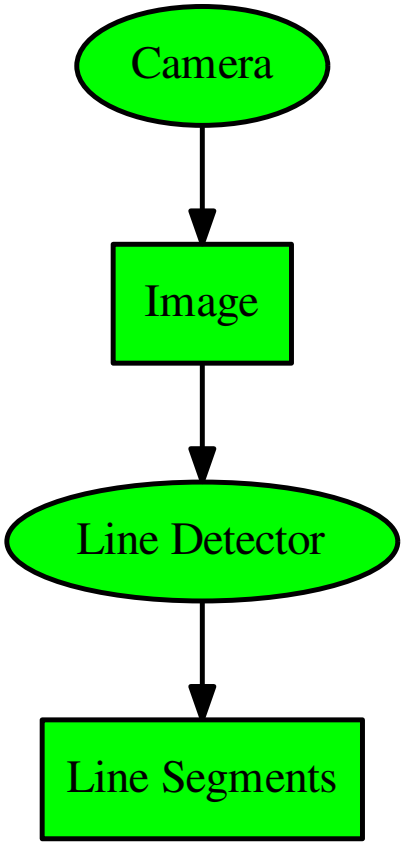
Hang Zhao

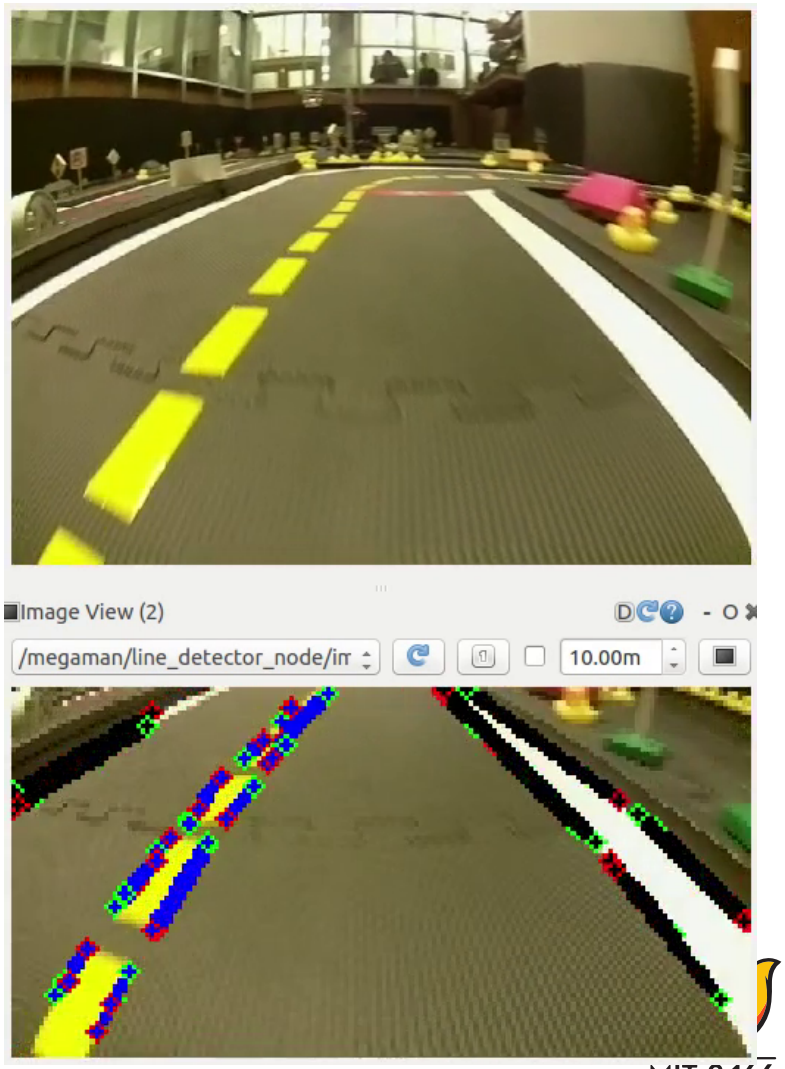




Line Detector

Hang Zhao

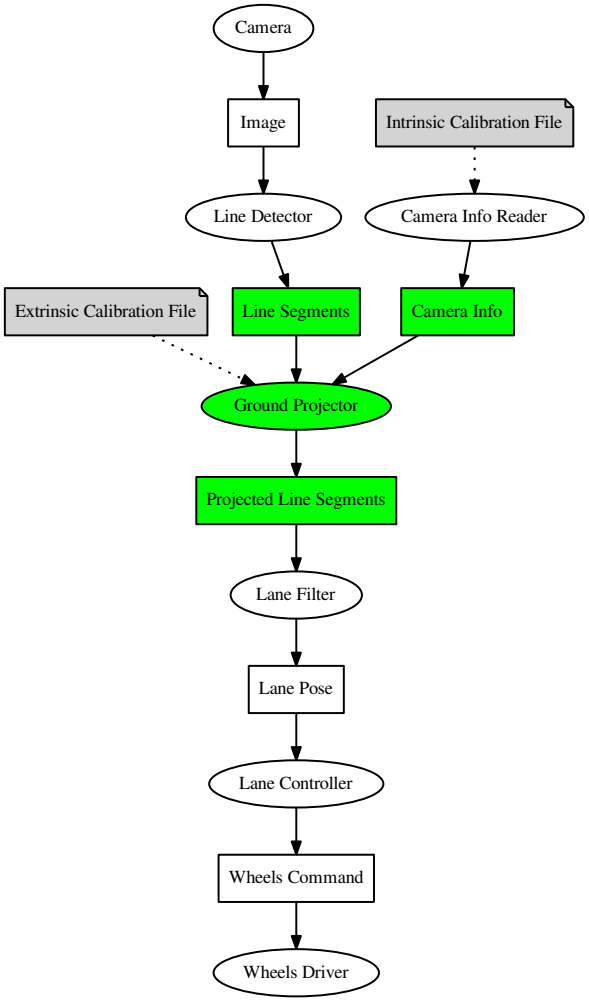


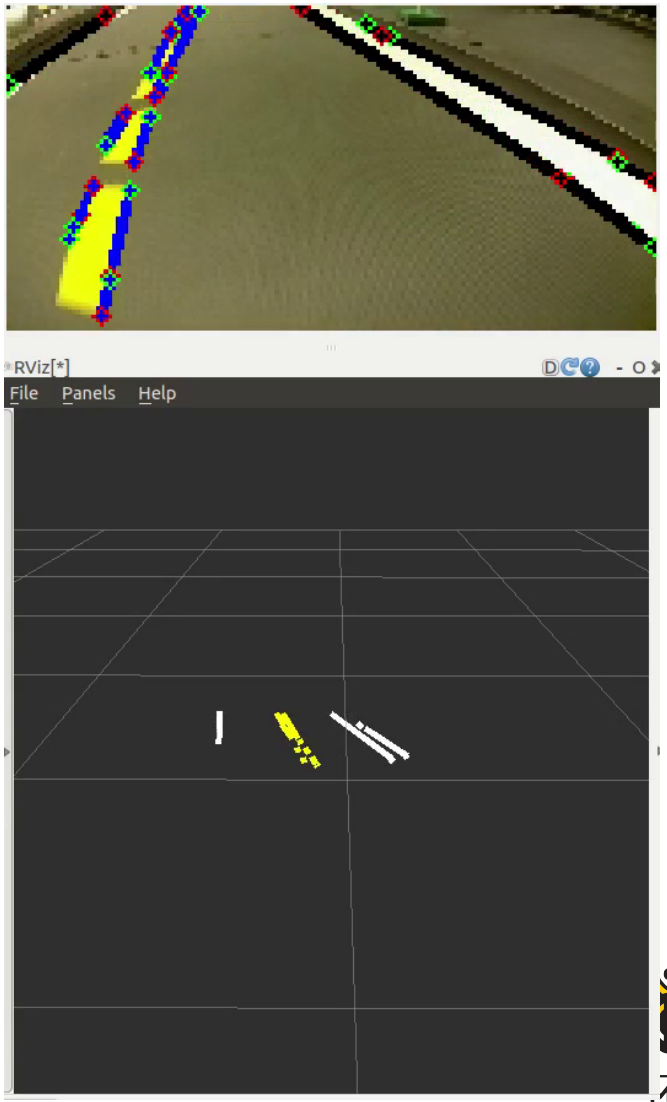


MIT 2.166 DUCKIETOWN

Ground Projector

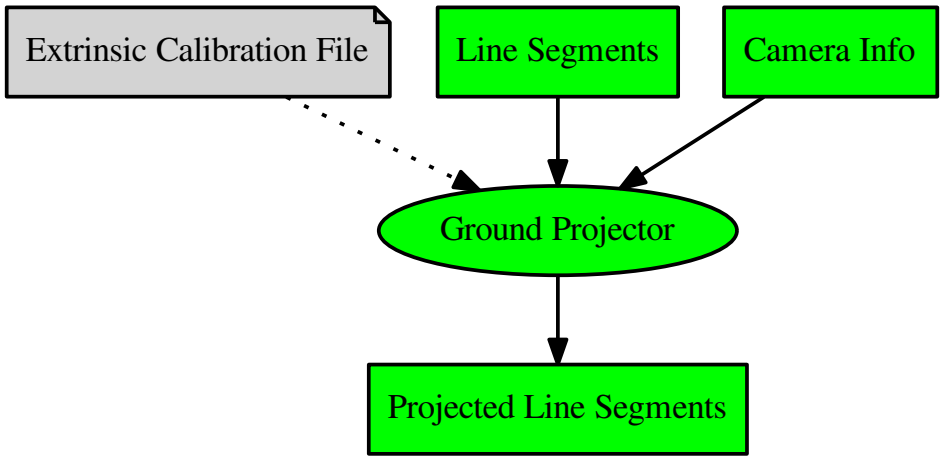
Dr. Changhyun Choi

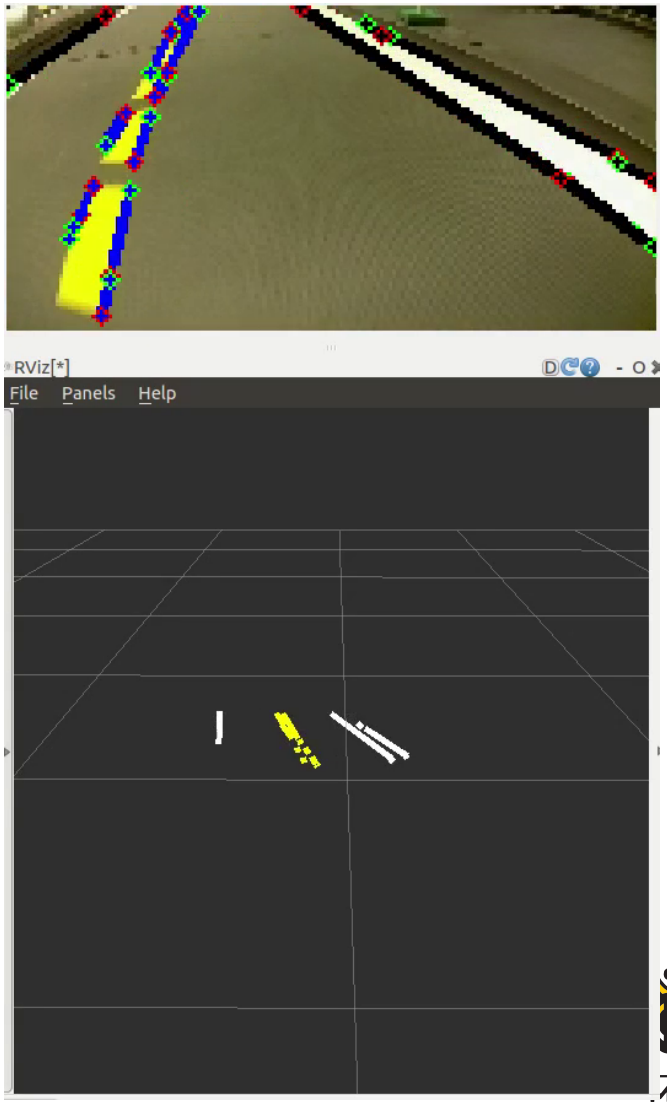




Ground Projector

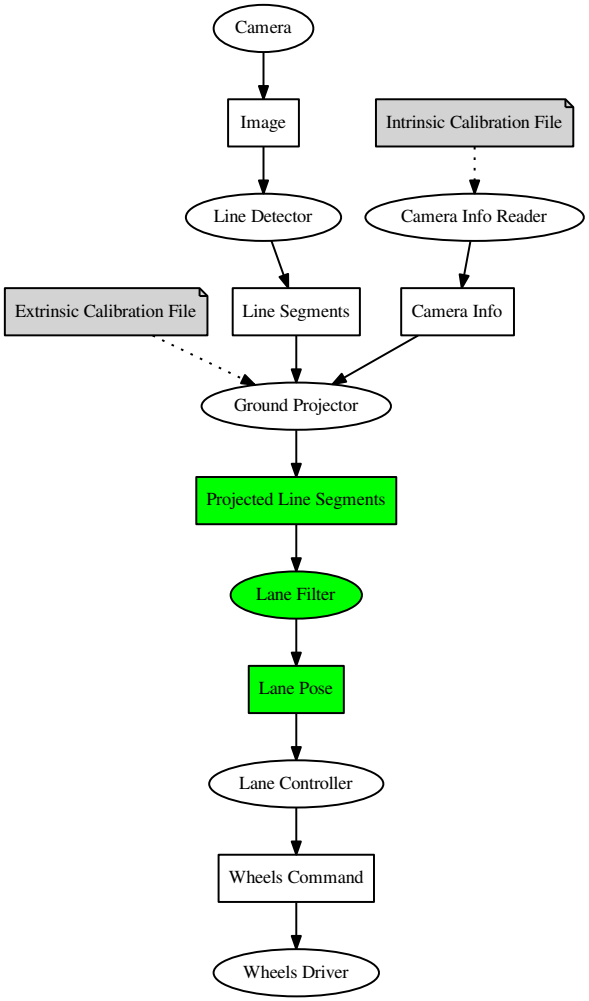
Dr. Changhyun Choi

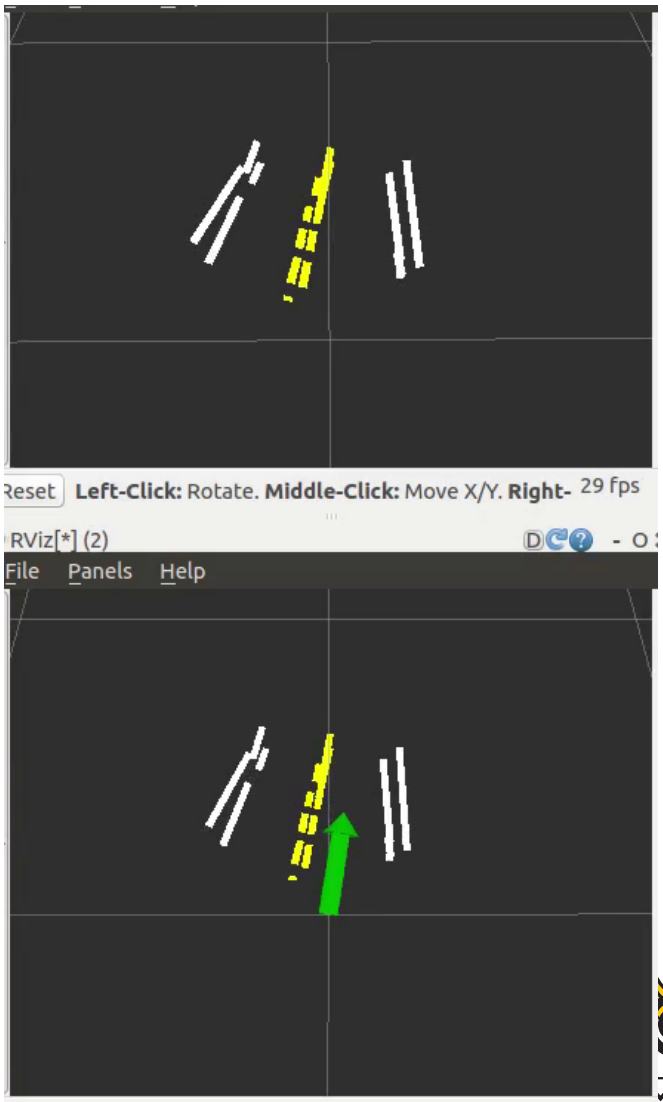




Lane Filter Node

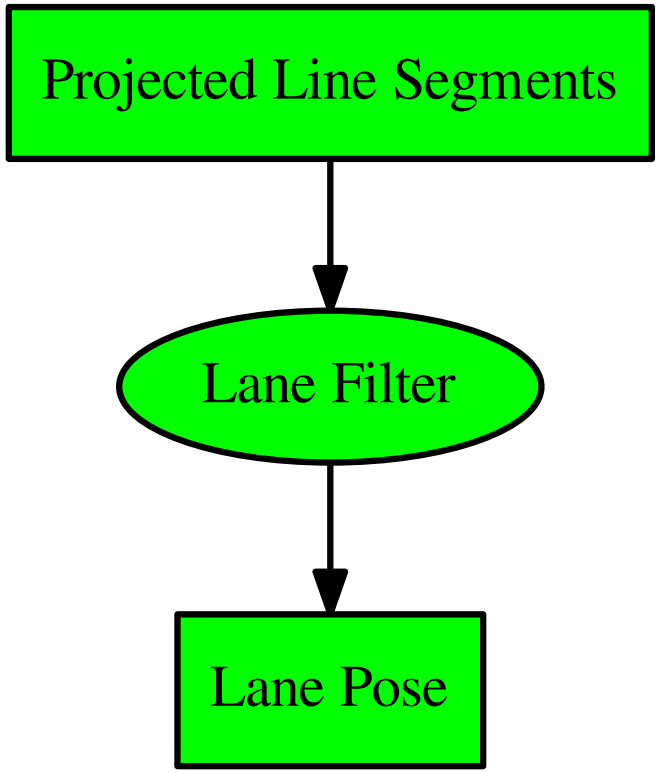
Dr. Liam Paull

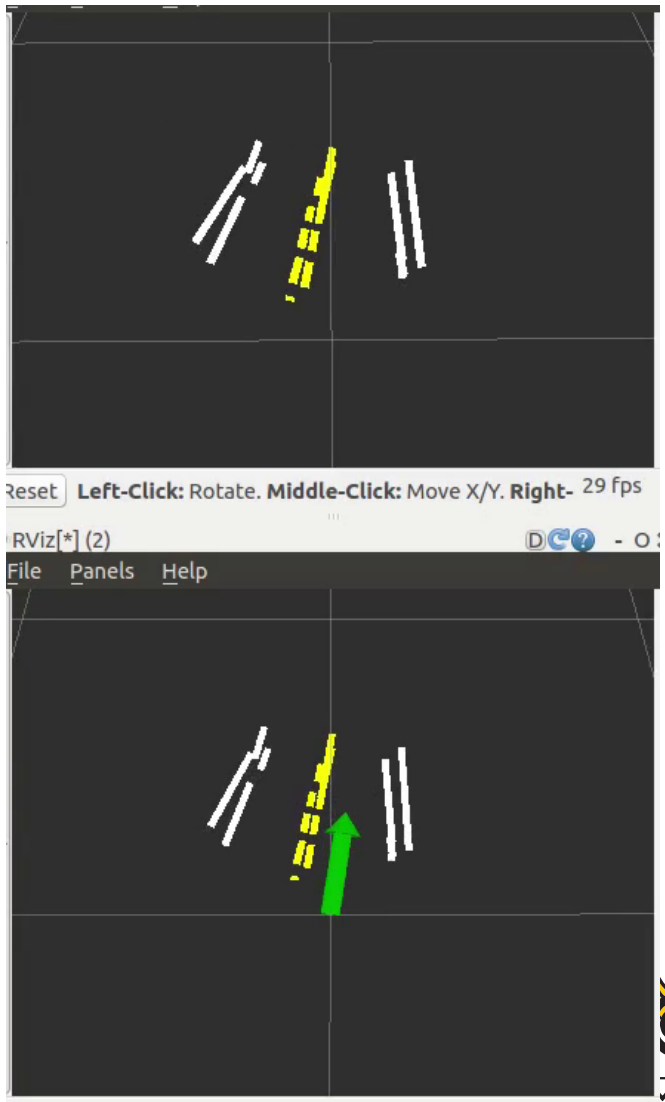




Lane Filter Node

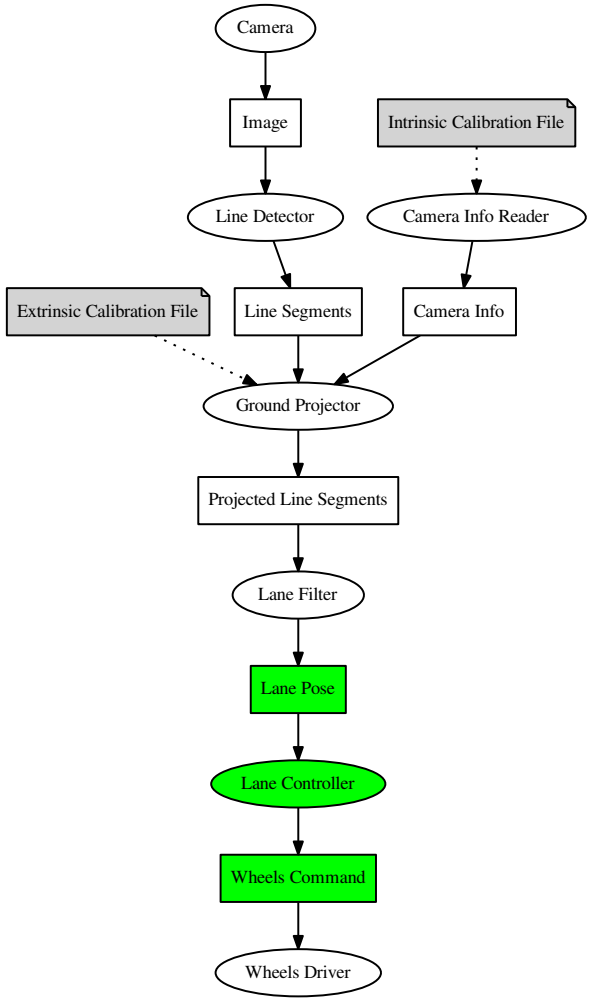
Dr. Liam Paull

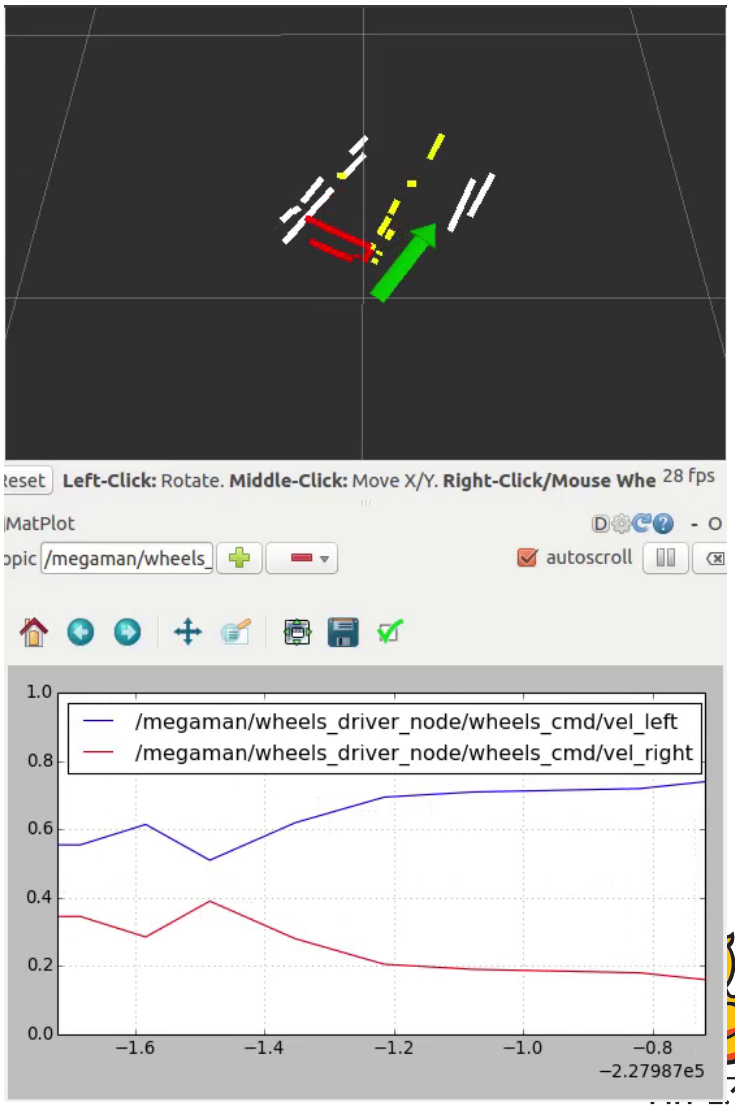




Lane Controller

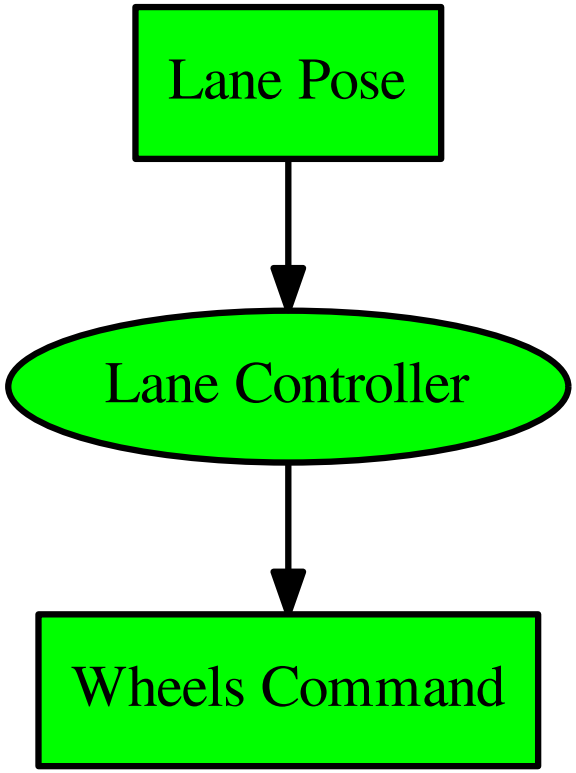
Steven Chen

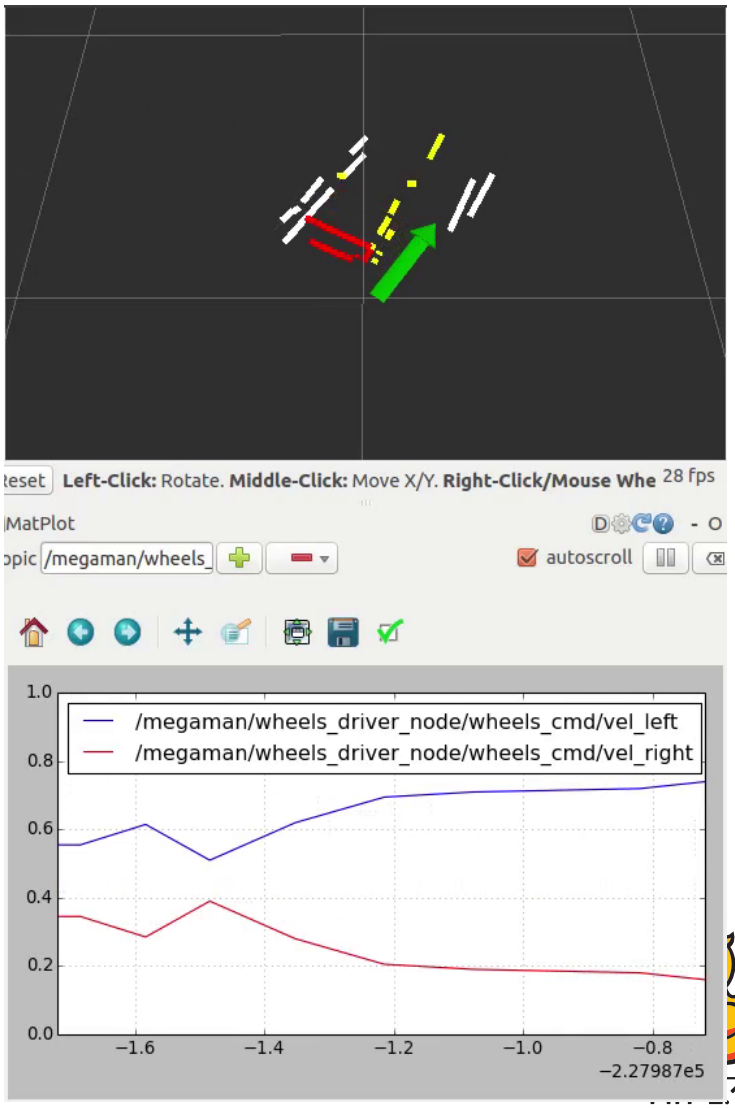




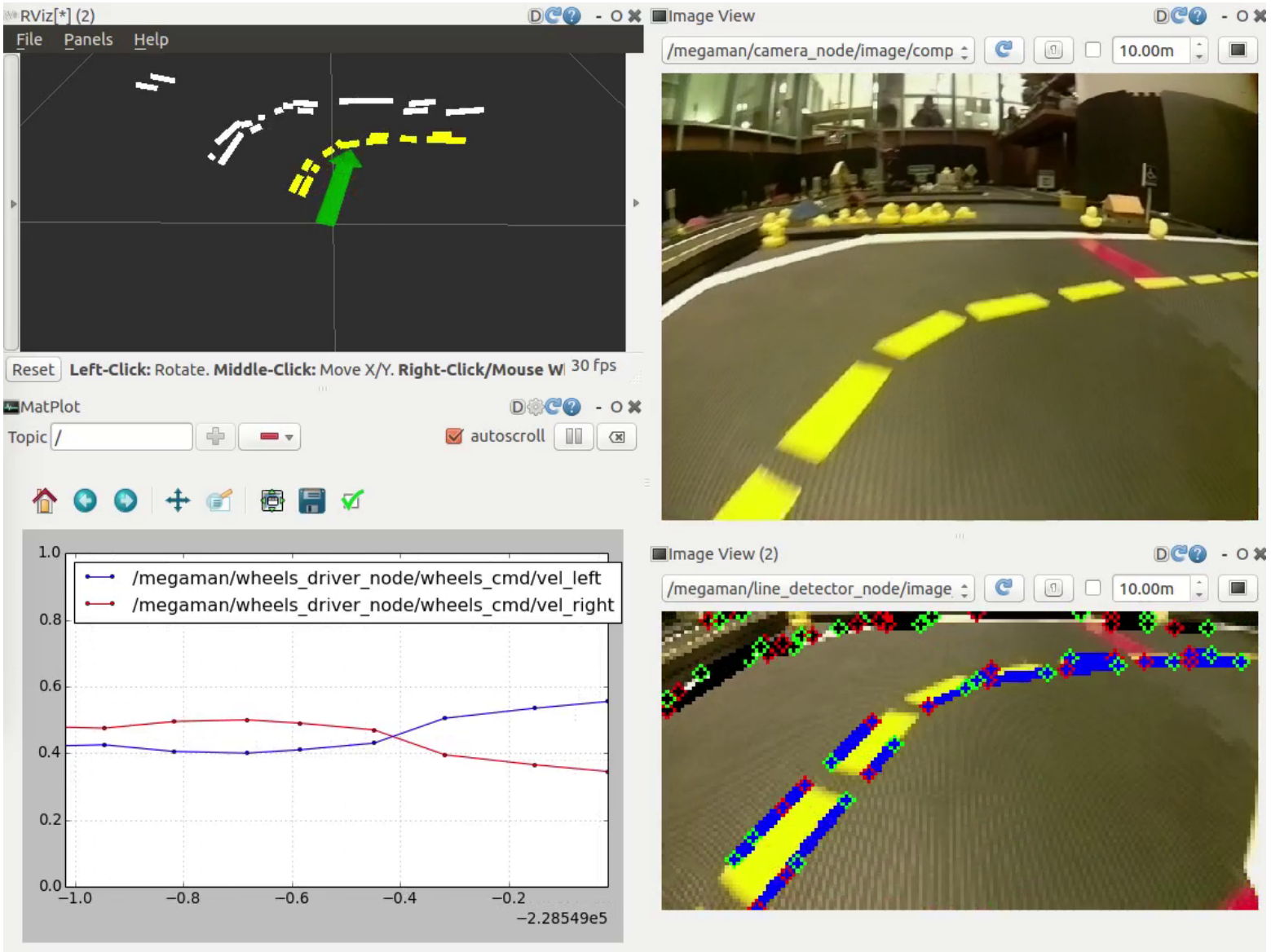
Lane Controller

Steven Chen





Lane Folloiwng Summary



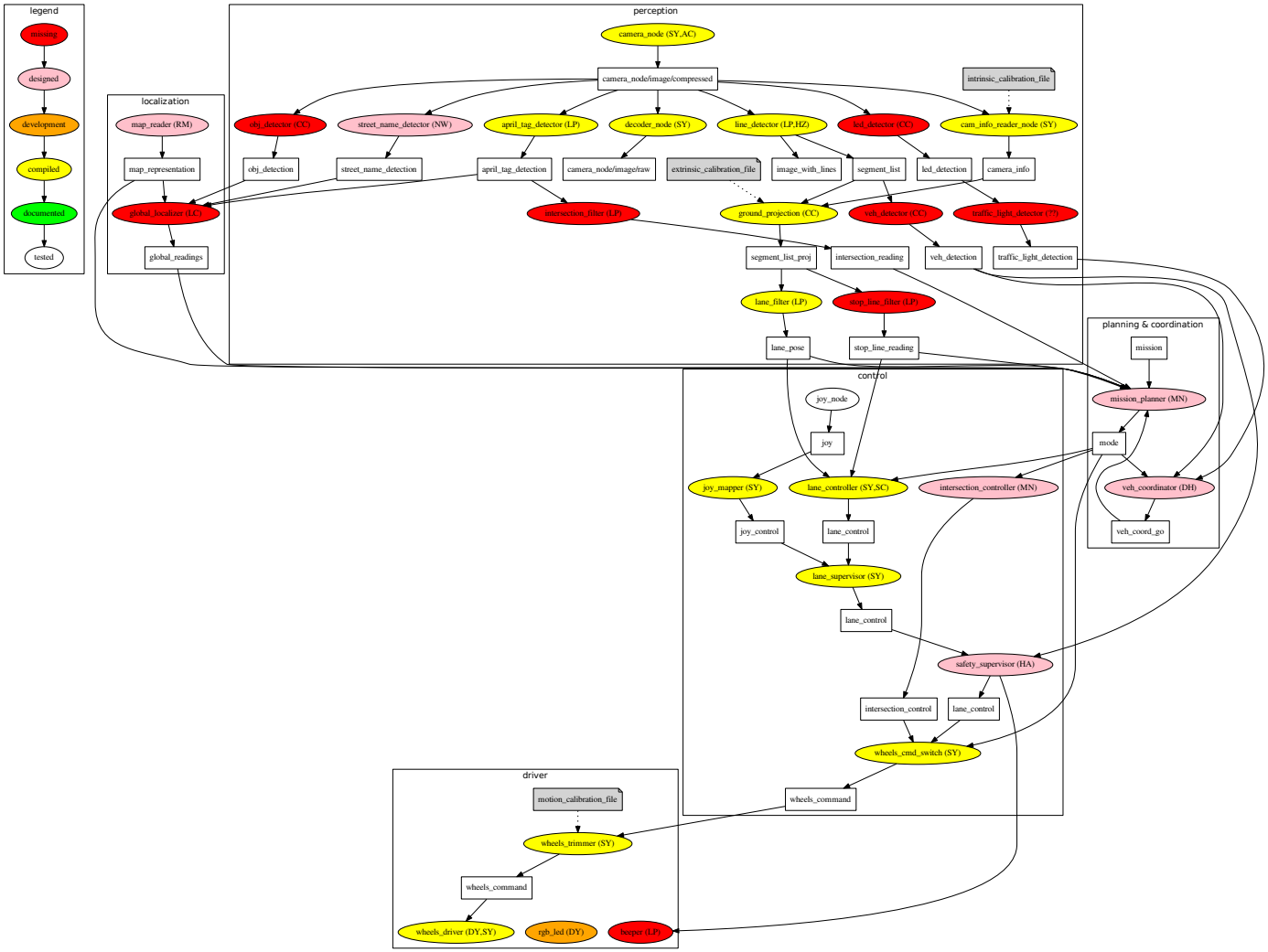
Intended Learning Outcome

Learning Objectives

- Describe the value of middleware for robotics software development
- Answer the question: "What is your favorite robotics middleware and why is it ROS?"
- Draw computation graph for simple robotic system using publish/subscribe model
- Explain important ROS concepts such as node, topic, message, and parameters
- Use important ROS command-line tools
- Write a simple ROS node by following code examples

Why do we need this?

Duckietown Architecture



MIT 2.166 DUCKIETOWN

Intended Learning Outcome

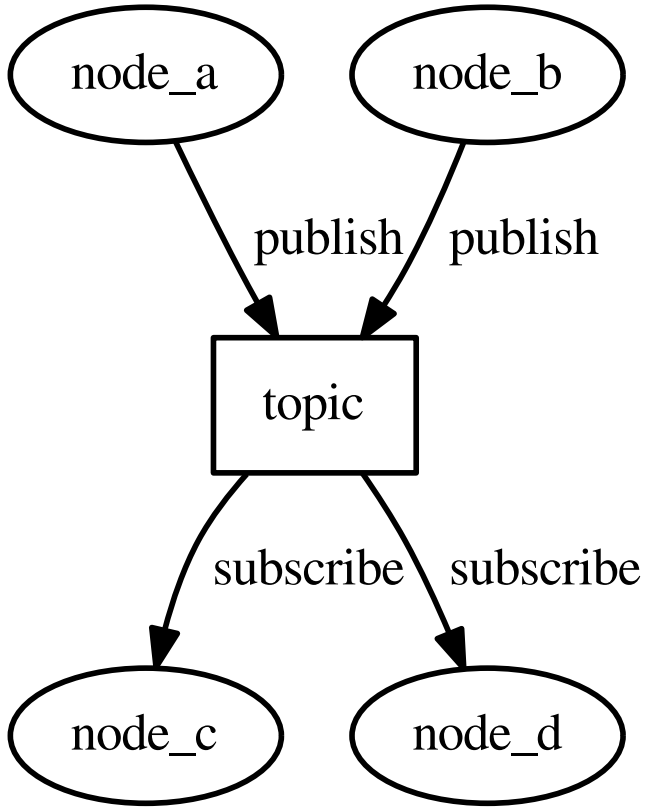
Learning Objectives

- Describe the value of middleware for robotics software development
- Answer the question: "What is your favorite robotics middleware and why is it ROS?"
- Draw computation graph for simple robotic system using publish/subscribe model
- Explain important ROS concepts such as node, topic, message, and parameters
- Use important ROS command-line tools
- Write a simple ROS node by following code examples

Overview

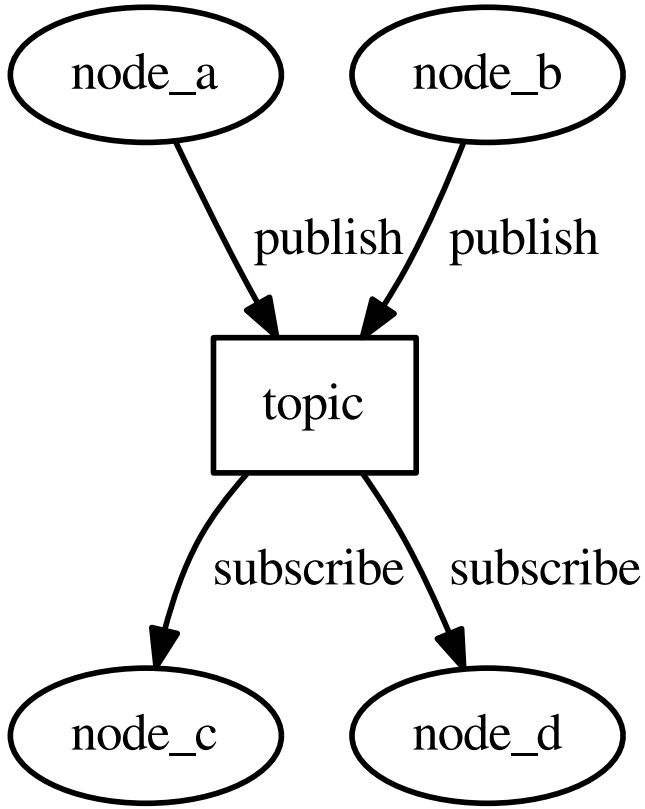
Nodes and Topics

- Nodes:
  - Executables (`python` or `C++`)
  - Each node is a process
  - Publishes/Subscribes to Topics
- Topics:
  - Passes information between nodes
  - Topic type defined by messages
  - Support many-to-many communication



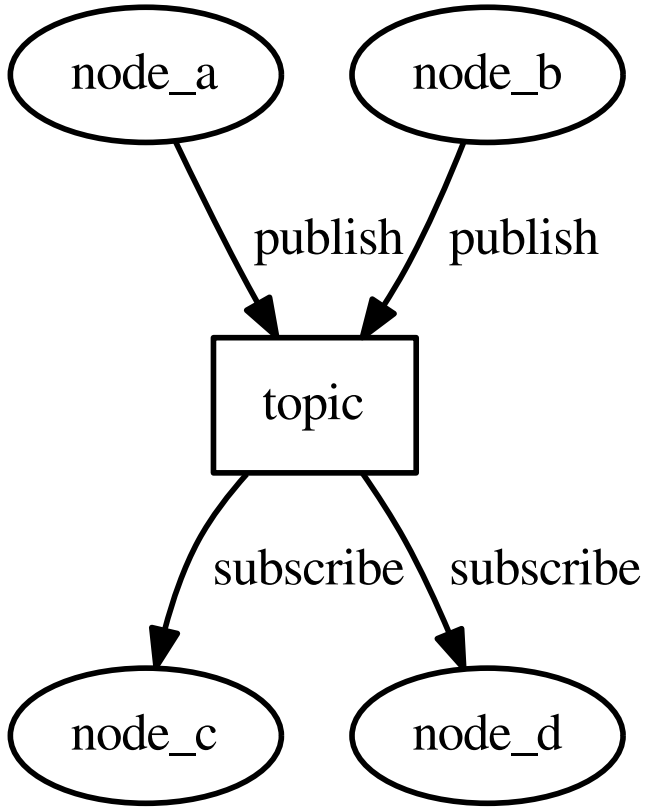
Nodes and Topics

- Nodes:
  - Executables (`python` or `C++`)
  - Each node is a process
  - Publishes/Subscribes to Topics
- Topics:
  - Passes information between nodes
  - Topic type defined by messages
  - Support many-to-many communication



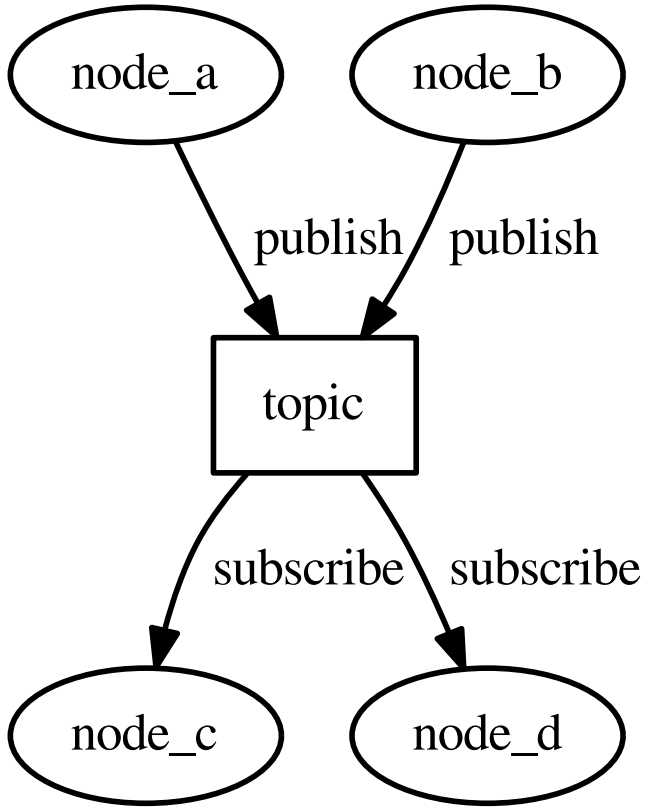
Nodes and Topics

- Nodes:
  - Executables (`python` or `C++`)
  - Each node is a process
  - Publishes/Subscribes to Topics
- Topics:
  - Passes information between nodes
  - Topic type defined by messages
  - Support many-to-many communication



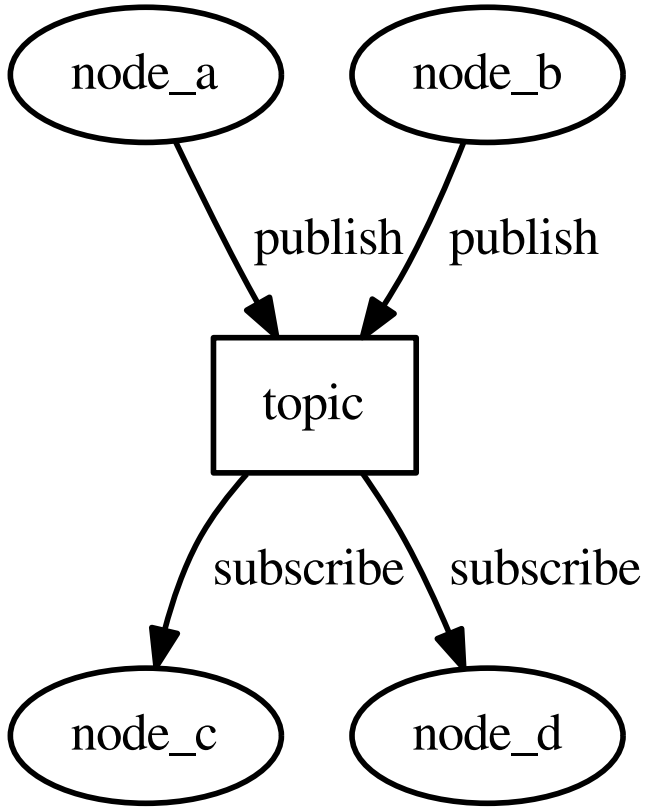
Nodes and Topics

- Nodes:
  - Executables (`python` or `C++`)
  - Each node is a process
  - Publishes/Subscribes to Topics
- Topics:
  - Passes information between nodes
  - Topic type defined by messages
  - Support many-to-many communication



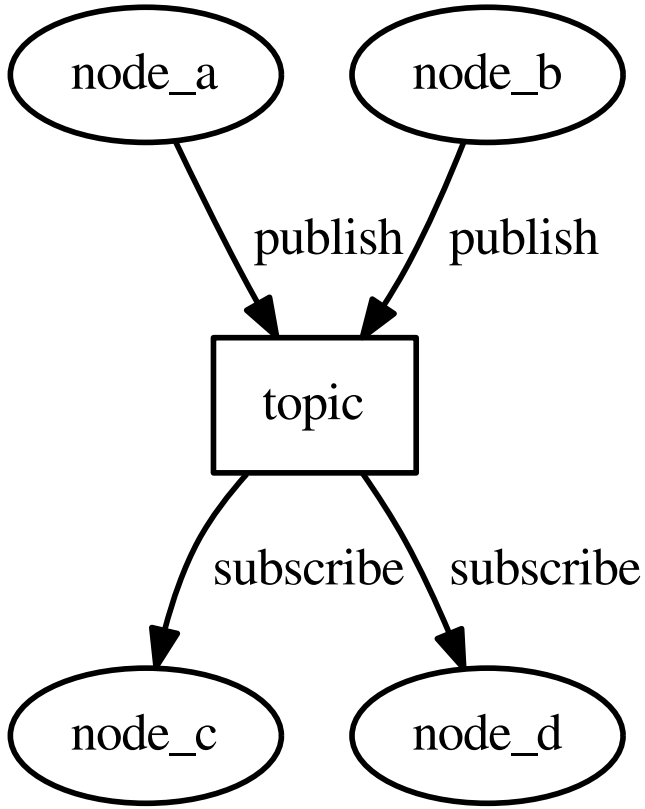
Nodes and Topics

- Nodes:
  - Executables (`python` or `C++`)
  - Each node is a process
  - Publishes/Subscribes to Topics
- Topics:
  - Passes information between nodes
  - Topic type defined by messages
  - Support many-to-many communication



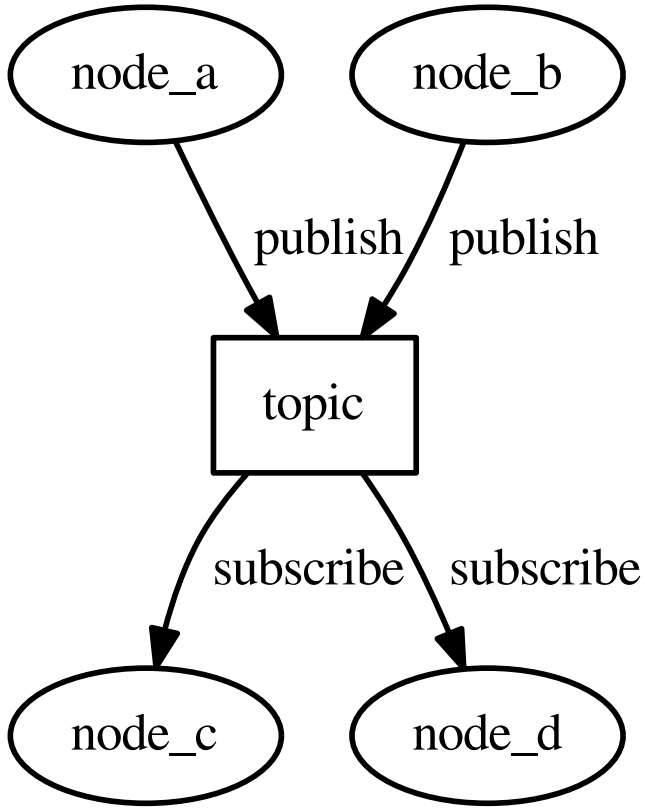
Nodes and Topics

- Nodes:
  - Executables (`python` or `C++`)
  - Each node is a process
  - Publishes/Subscribes to Topics
- Topics:
  - Passes information between nodes
  - Topic type defined by messages
  - Support many-to-many communication



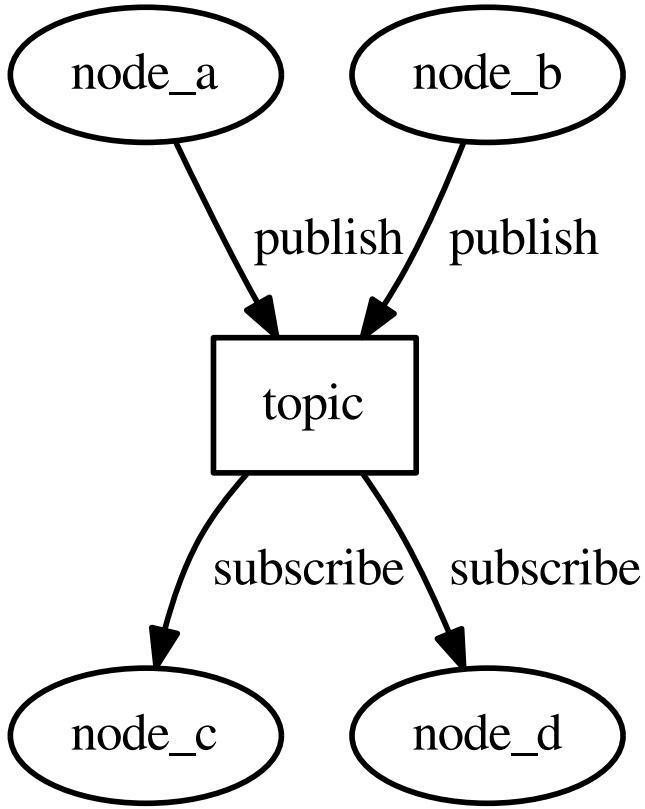
Nodes and Topics

- Nodes:
  - Executables (`python` or `C++`)
  - Each node is a process
  - Publishes/Subscribes to Topics
- Topics:
  - Passes information between nodes
  - Topic type defined by messages
  - Support many-to-many communication



Nodes and Topics

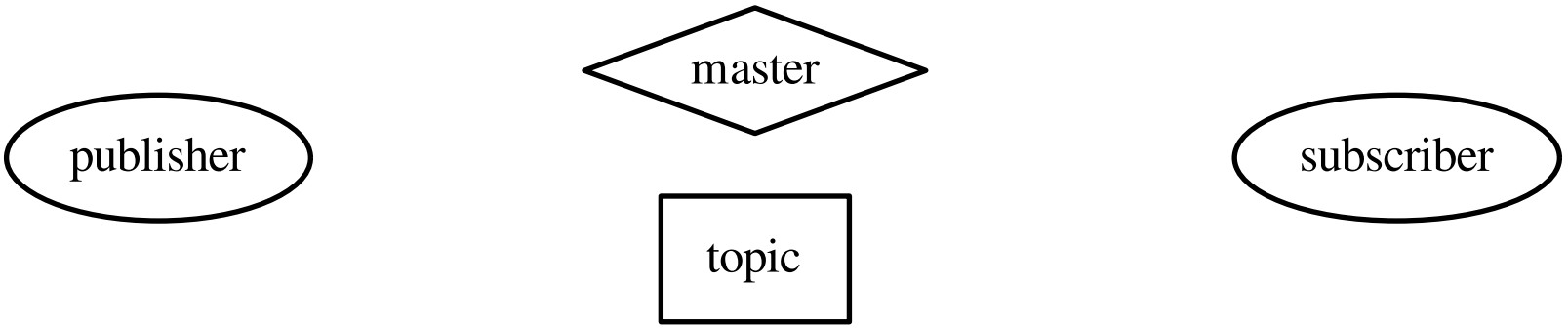
- Nodes:
  - Executables (`python` or `C++`)
  - Each node is a process
  - Publishes/Subscribes to Topics
- Topics:
  - Passes information between nodes
  - Topic type defined by messages
  - Support many-to-many communication



ROS Master

The Telephone Operator

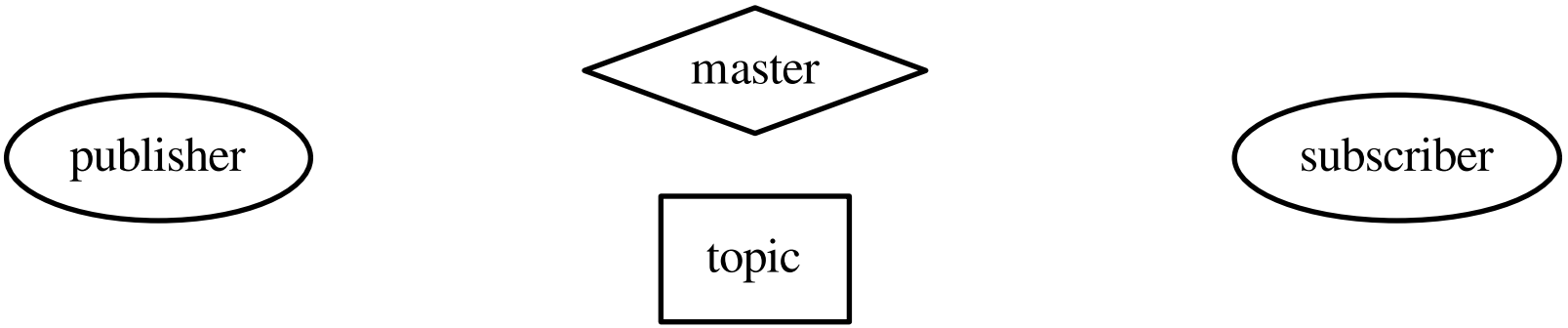
- Handles communication between nodes
- Connects publisher and subscriber through topics
- Traffic does not go through the master once connected
- TCP/IP connection



ROS Master

The Telephone Operator

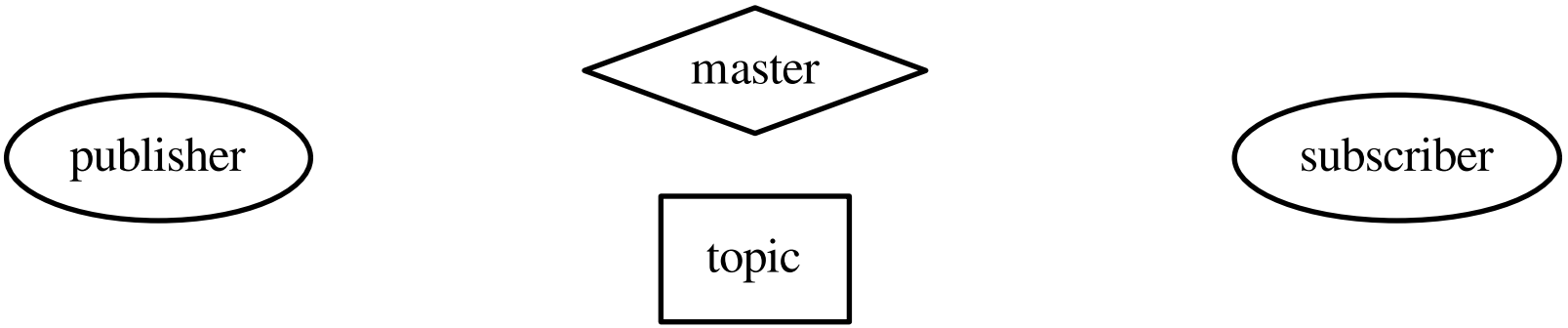
- Handles communication between nodes
- Connects publisher and subscriber through topics
- Traffic does not go through the master once connected
- TCP/IP connection



ROS Master

The Telephone Operator

- Handles communication between nodes
- Connects publisher and subscriber through topics
- Traffic does not go through the master once connected
- TCP/IP connection

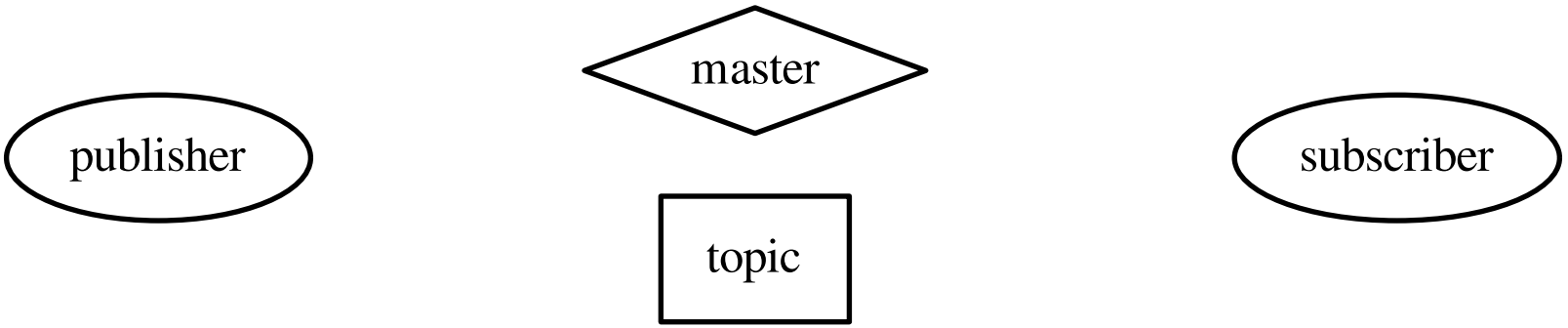


MIT 2.166 DUCKIETOWN

ROS Master

The Telephone Operator

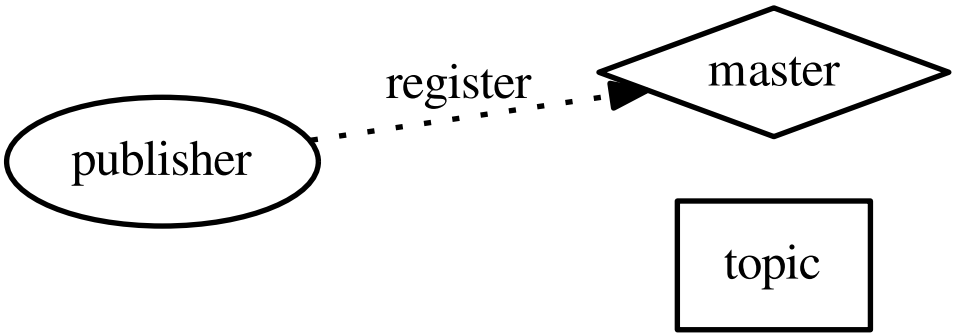
- Handles communication between nodes
- Connects publisher and subscriber through topics
- Traffic does not go through the master once connected
- TCP/IP connection



ROS Master

The Telephone Operator

- Handles communication between nodes
- Connects publisher and subscriber through topics
- Traffic does not go through the master once connected
- TCP/IP connection

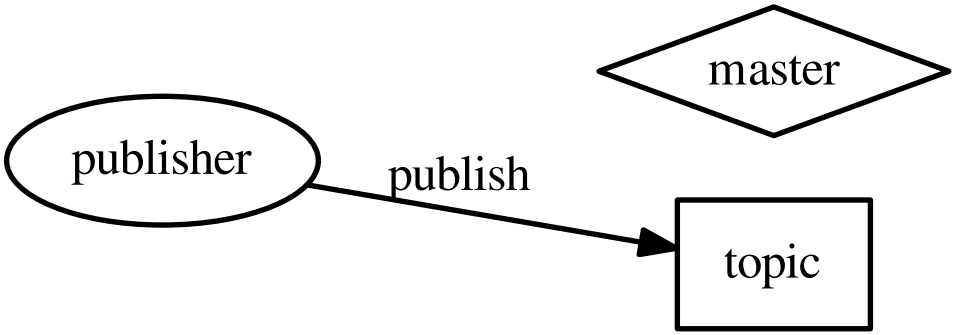


MIT 2.166 DUCKIETOWN

ROS Master

The Telephone Operator

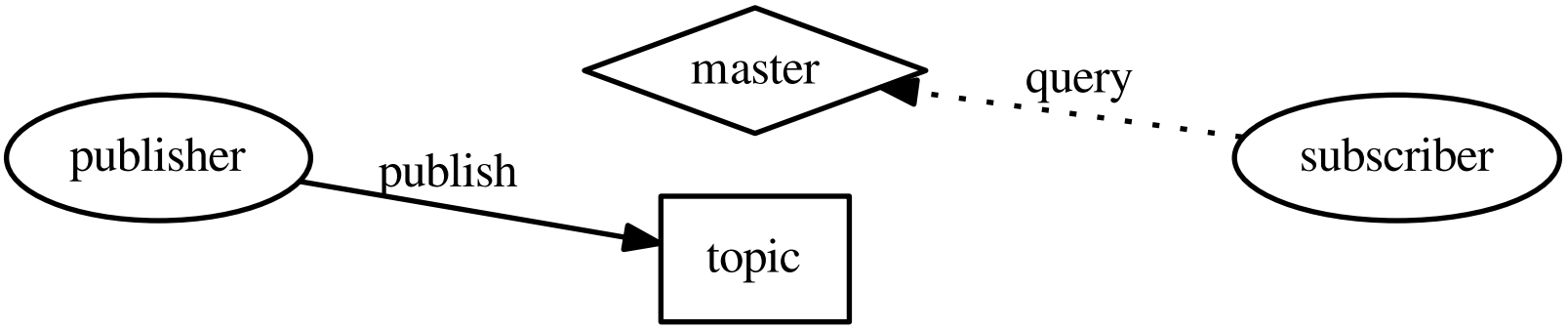
- Handles communication between nodes
- Connects publisher and subscriber through topics
- Traffic does not go through the master once connected
- TCP/IP connection



ROS Master

The Telephone Operator

- Handles communication between nodes
- Connects publisher and subscriber through topics
- Traffic does not go through the master once connected
- TCP/IP connection

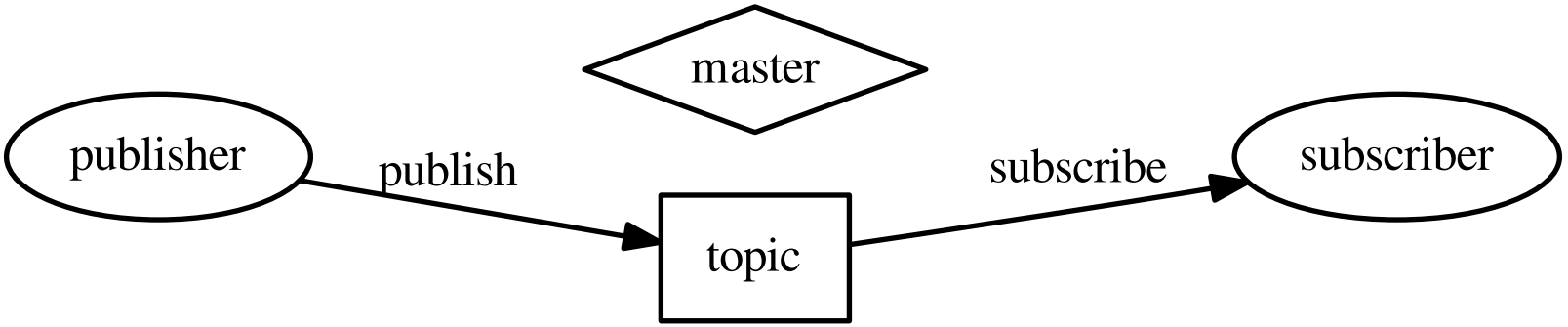


MIT 2.166 DUCKIETOWN

ROS Master

The Telephone Operator

- Handles communication between nodes
- Connects publisher and subscriber through topics
- Traffic does not go through the master once connected
- TCP/IP connection



MIT 2.166 DUCKIETOWN

Commandline Tools and Demo

- Start a ROS Master:

  `roscore`
- Launch a launch file:

  `roslaunch pkg_name launch_file`
- View communication graph:

  `rqt_graph`
- List all nodes:

  `rosnode list`
- Info of a node:

  `rosnode info node_name`

Live demo

Commandline Tools and Demo

- List all topics:

  `rostopic list`
- Info of a topic:

  `rostopic info topic_name`
- Listen to a topic:

  `rostopic echo topic_name`
- Check frequency of a topic:

  `rostopic hz topic_name`
- Plot a topic:

  `rqt_plot`
- 3D Visualizer:

  `rviz`

Live demo

Messages

- The content and structure of a topic is defined by a message
- Application Programming Interface (API) for nodes
- Defined in `.msg` files
- Primitive Type: `bool`, `string`, `float32`, `int32`, ... etc
  - Single value field: `field_type field_name`
  - Fixed-length field: `field_type[n] field_name`
  - Variable-length field: `field_type[] field_name`

Messages

- The content and structure of a topic is defined by a message
- Application Programming Interface (API) for nodes
- Defined in `.msg` files
- Primitive Type: `bool`, `string`, `float32`, `int32`, ... etc
  - Single value field: `field_type field_name`
  - Fixed-length field: `field_type[n] field_name`
  - Variable-length field: `field_type[] field_name`

Messages

- The content and structure of a topic is defined by a message
- Application Programming Interface (API) for nodes
- Defined in `.msg` files
- Primitive Type: `bool`, `string`, `float32`, `int32`, ... etc
  - Single value field: `field_type field_name`
  - Fixed-length field: `field_type[n] field_name`
  - Variable-length field: `field_type[] field_name`

Messages

- The content and structure of a topic is defined by a message
- Application Programming Interface (API) for nodes
- Defined in `.msg` files
- Primitive Type: `bool`, `string`, `float32`, `int32`, ... etc
  - Single value field: `field_type field_name`
  - Fixed-length field: `field_type[n] field_name`
  - Variable-length field: `field_type[] field_name`

Messages

- The content and structure of a topic is defined by a message
- Application Programming Interface (API) for nodes
- Defined in `.msg` files
- Primitive Type: `bool`, `string`, `float32`, `int32`, ... etc
  - Single value field: `field_type field_name`
  - Fixed-length field: `field_type[n] field_name`
  - Variable-length field: `field_type[] field_name`

Messages

- The content and structure of a topic is defined by a message
- Application Programming Interface (API) for nodes
- Defined in `.msg` files
- Primitive Type: `bool`, `string`, `float32`, `int32`, ... etc
  - Single value field: `field_type field_name`
  - Fixed-length field: `field_type[n] field_name`
  - Variable-length field: `field_type[] field_name`

Messages

- The content and structure of a topic is defined by a message
- Application Programming Interface (API) for nodes
- Defined in `.msg` files
- Primitive Type: `bool`, `string`, `float32`, `int32`, ... etc
  - Single value field: `field_type field_name`
  - Fixed-length field: `field_type[n] field_name`
  - Variable-length field: `field_type[] field_name`

Commandline Tools and Demo

- Look up definition of a message:

  `rosmsg show pkg_name msg_name`

- Common message packages:
  - `std_msgs`
  - `geometry_msgs`
  - `sensor_msgs`
  - `duckietown_msgs`

Live Demo

Parameters

- Publish/Subscribe does not suit everything
- Nodes often requires parameterization
  - Camera resolution
  - Threshold
  - Controller gain
- Kept on the parameter server:
  - Initialized with master
  - Parameters persist until master is killed
- Can be set and changed at run time

Parameters

- Publish/Subscribe does not suit everything
- Nodes often requires parameterization
  - Camera resolution
  - Threshold
  - Controller gain
- Kept on the parameter server:
  - Initialized with master
  - Parameters persist until master is killed
- Can be set and changed at run time

Parameters

- Publish/Subscribe does not suit everything
- Nodes often requires parameterization
  - Camera resolution
  - Threshold
  - Controller gain
- Kept on the parameter server:
  - Initialized with master
  - Parameters persist until master is killed
- Can be set and changed at run time

Parameters

- Publish/Subscribe does not suit everything
- Nodes often requires parameterization
  - Camera resolution
  - Threshold
  - Controller gain
- Kept on the parameter server:
  - Initialized with master
  - Parameters persist until master is killed
- Can be set and changed at run time

Parameters

- Publish/Subscribe does not suit everything
- Nodes often requires parameterization
  - Camera resolution
  - Threshold
  - Controller gain
- Kept on the parameter server:
  - Initialized with master
  - Parameters persist until master is killed
- Can be set and changed at run time

Parameters

- Publish/Subscribe does not suit everything
- Nodes often requires parameterization
  - Camera resolution
  - Threshold
  - Controller gain
- Kept on the parameter server:
  - Initialized with master
  - Parameters persist until master is killed
- Can be set and changed at run time

Parameters

- Publish/Subscribe does not suit everything
- Nodes often requires parameterization
  - Camera resolution
  - Threshold
  - Controller gain
- Kept on the parameter server:
  - Initialized with master
  - Parameters persist until master is killed
- Can be set and changed at run time

Parameters

- Publish/Subscribe does not suit everything
- Nodes often requires parameterization
  - Camera resolution
  - Threshold
  - Controller gain
- Kept on the parameter server:
  - Initialized with master
  - Parameters persist until master is killed
- Can be set and changed at run time

Parameters

- Publish/Subscribe does not suit everything
- Nodes often requires parameterization
  - Camera resolution
  - Threshold
  - Controller gain
- Kept on the parameter server:
  - Initialized with master
  - Parameters persist until master is killed
- Can be set and changed at run time

Commandline Tools and Demo for Parameters

- List parameters:

```
rosparam list
```

- Get parameter:

```
rosparam get param_name
```

- Set parameter:

```
rosparam set param_name
```

- Dump parameters into file:

```
rosparam dump file_name [namespace]
```

- Read parameters from file:

```
rosparam load file_name [namespace]
```

Live Demo

Launch File

- XML format file can specify almost all aspects/actions of ROS
  - Launch nodes, set/load parameters, remap topics, include other launch files, pass commandline arguments
- Syntax:

```
<tag attribute1="value1" attribute2="value2">
        <element_tag_1 attribute3="value3">
                <!-- Other nesting tags -->
        </element_tag_1>
</tag>
```

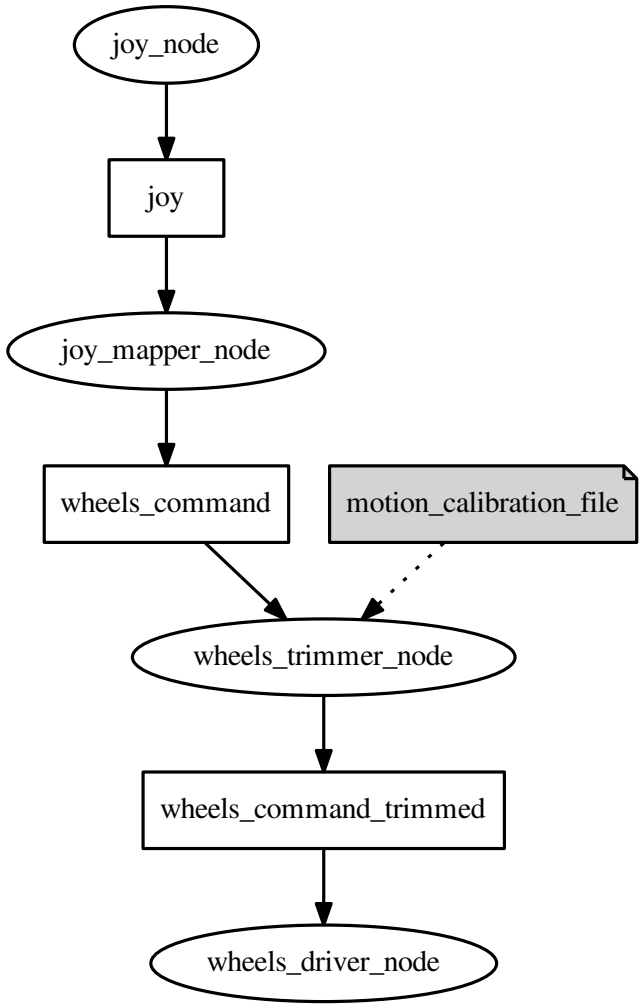
- Tags:
  - `<launch>` : Container of launch files.
  - `<group>` : Group nodes together.
  - `<arg>` : I/O of launch files.
  - `<node>` : Run a node.
  - `<include>` : Invoke other launch files.

Attribute and Elements for <node>

- Key attributes:
  - `pkg` : Name of package
  - `type` : Executable file name
  - `name` : Name of the node
  - `args` : Commandline arguments for the exec
  - `machine` : Machine to run the node
  - `ns` : Namespace of the node
  - `output` : Outputs of the node
  - `if/unless` : Conditional
- Key elements:
  - `<remap>` : Remap topics
  - `<rosparam>` : Set parameters
  - `<param>` : Set parameters

Joystick (Remote Control)

- `joy_node`
- `joy_mapper_node`
- `wheels_trimmer_node`
- `wheels_driver_node`



MIT 2.166 DUCKIETOWN

Joystick.launch

```
<launch>
    <arg name="veh"/>
    <arg name="local" default="false"/>
    <arg name="config" default="baseline"/>
    <arg name="param_file_name" default="default"/>
    <arg name="trim" default="false"/>

    <include file="$(find duckietown)/machines"/>
    <!-- joy -->
    <node ns="$(arg veh)" if="$(arg local)" pkg="joy" type="joy_node" name="joy" o
        <rosparam command="load" file="$(find duckietown)/config/$(arg config)/joy
    </node>
    <node ns="$(arg veh)" unless="$(arg local)" machine="$(arg veh)" pkg="joy" typ
        <rosparam command="load" file="$(find duckietown)/config/$(arg config)/joy
    </node>

    <!-- joy_mapper -->
    <remap unless="$(arg trim)" from="joy_mapper_node/wheels_cmd" to="wheels_drive
    <include file="$(find joy_mapper)/launch/joy_mapper_node.launch">
        <arg name="veh" value="$(arg veh)"/>
        <arg name="local" value="$(arg local)"/>
        <arg name="config" value="$(arg config)"/>
        <arg name="param_file_name" value="$(arg param_file_name)"/>
    </include>

```

Intended Learning Outcome

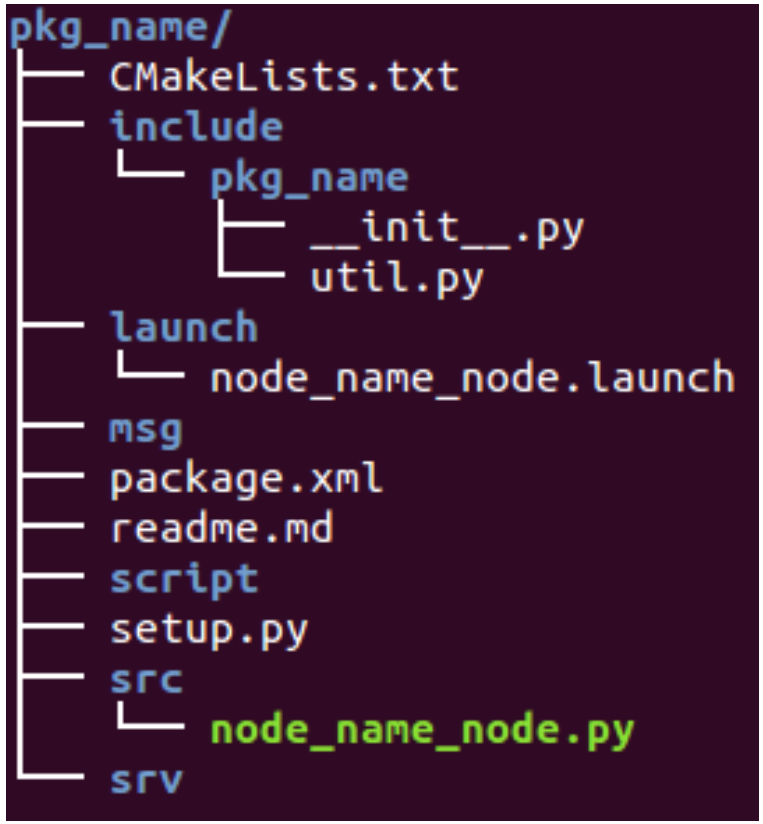
Learning Objectives

- Describe the value of middleware for robotics software development
- Answer the question: "What is your favorite robotics middleware and why is it ROS?"
- Draw computation graph for simple robotic system using publish/subscribe model
- Explain important ROS concepts such as node, topic, message, and parameters
- Use important ROS command-line tools
- Write a simple ROS node by following code examples

Overview

Packages

- Packages are organization tools for:
  - Functionality
  - Build and runtime dependencies
  - Namespaces
- Key Files
  - CMakeLists.txt
  - package.xml
  - setup.py
- Tools
  - `roscd`
  - `rospack profile`
  - `rospack depends`



MIT 2.166 DUCKIETOWN

Publisher Node Code Example with Timer

publisher_node_timer.py

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String #Imports msg
# Initialize the node with rospy
rospy.init_node('publisher_node')
# Create publisher
publisher = rospy.Publisher("topic",String,queue_size=1)
# Define Timer callback
def callback(event):
    msg = String()
    msg.data = "Hello Duckietown!"
    publisher.publish(msg)
# Create timer
rospy.Timer(rospy.Duration.from_sec(1.0),callback)
# spin to keep the script for exiting
rospy.spin()
```

- Create a publisher: `rospy.Publisher(topic_name,msg_type,queue_size)`
- Start a timer: `rospy.Timer(duration,callback)`

Subscriber Node Code Example

subscriber_node.py

```
1  #!/usr/bin/env python
2  import rospy
3  from std_msgs.msg import String #Imports msg
4
5  # Define callback functoin
6  def callback(msg):
7          rospy.loginfo("I heard: %s" %(msg.data))
8
9  # Initialize the node with rospy
10 rospy.init_node("subscriber_node")
11 # Create subscriber
12 subscriber = rospy.Subscriber("topic", String, callback)
13
14 rospy.spin() #Keeps the script for exiting
```

- Create a subscriber: `rospy.Subscriber(topic_name,callback)`

Parameter Code Example

publisher_node_param.py

```
1  #!/usr/bin/env python
2  import rospy
3  from std_msgs.msg import String #Imports msg
4  # Initialize the node with rospy
5  rospy.init_node('publisher_node')
6  # Create publisher
7  publisher = rospy.Publisher("topic",String,queue_size=1)
8  # Set parameter if doesn't exist
9  if not rospy.has_param("~place"):
10     rospy.set_param("~place","Duckietown")
11 # Define Timer callback
12 def callback(event):
13     # Read parameter
14     place = rospy.get_param("~place")
15     msg = String()
16     msg.data = "Hello %s!" %(place)
17     publisher.publish(msg)
18 # Create timer
19 rospy.Timer(rospy.Duration.from_sec(1.0),callback)
20 # spin to keep the script for exiting
21 rospy.spin()
```

Message Definition Examples

- `Duckie.msg` defines the name, state, and pose of a duckie.

```
#Duckie
Header header
string name
string state
geometry_msgs/Vector3 pos
# Enums
string STATE_HAPPY="HAPPY"
string STATE_NORMAL="NORMAL"
string STATE_SUPER="SUPER"
```

- `DuckieList.msg` contains a list of duckies.

```
#DuckieList
Header header
pkg_name_msgs/Duckie[] duckie_list
```

- Look up msg definition:

```
rosmsg show pkg_name/msg_name
```

Message Usage in Publisher Code Example

publisher_node_msg.py

```
1  #!/usr/bin/env python
2  import rospy
3  from pkg_name_msgs.msg import Duckie
4  import random
5  # Initialize the node with rospy
6  rospy.init_node("publisher_node")
7  # Create publisher
8  publisher = rospy.Publisher("topic",Duckie,queue_size=1)
9  # Define Timer callback
10 def callback(event):
11     msg = Duckie()
12     msg.header.stamp = rospy.Time.now()
13     msg.name = random.choice(["Shih-Yuan","Liam","Misha"])
14     msg.state = random.choice([Duckie.STATE_HAPPY,Duckie.STATE_SUPER])
15     publisher.publish(msg)
16 # Create timer
17 rospy.Timer(rospy.Duration.from_sec(1.0),callback)
18 # spin to keep the script for exiting
19 rospy.spin()
```

Message Usage in Subscriber Code Example

subscriber_node_msg.py

```
1  #!/usr/bin/env python
2  import rospy
3  from pkg_name_msgs.msg import Duckie
4  # Define callback functoin
5  def callback(msg):
6          rospy.loginfo("%s is %s." %(msg.name,msg.state))
7  # Initialize the node with rospy
8  rospy.init_node("subscriber_node")
9  # Create subscriber
10 subscriber = rospy.Subscriber("topic", Duckie, callback)
11
12 rospy.spin() #Keeps the script for exiting
```

Resources

`http://wiki.ros.org/`