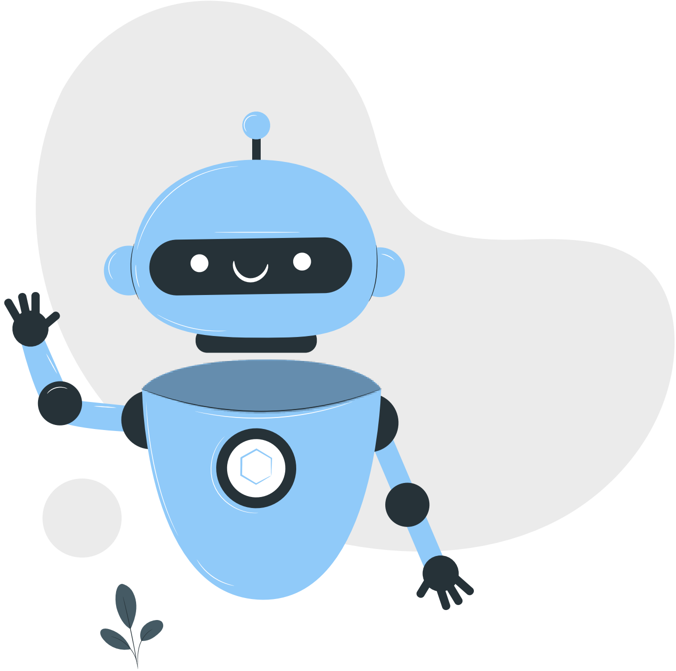
Intro to ROS

CSE574 Planning and Learning Methods in AI

Elena Oikonomou



Contents

Part I

Intro

- What is ROS?
- How to create/build your packages.

Part 2

ROS Ecosystem

- Fundamental concepts
- Basic commands
- Develop ROS nodes

Part 3

Simulation

- Rviz
- Control a robot in Gazebo.

Contents

Part I

Intro

- What is ROS?
- How to create/build your packages.

Part 2

ROS Ecosystem

- Fundamental concepts
- Basic commands
- Develop ROS nodes

Part 3

Simulation

- Rviz
- Control a robot in Gazebo.

What is ROS?

- **ROS (Robot Operating System)**
  is a set of **software libraries** and **tools** that help us build robotics applications!

Plumbing

- Process management
- Code organization
- Communication between components

Tools

- Simulation
- Visualization
- Debugging
- Plotting
- Logging
- …

Capabilities

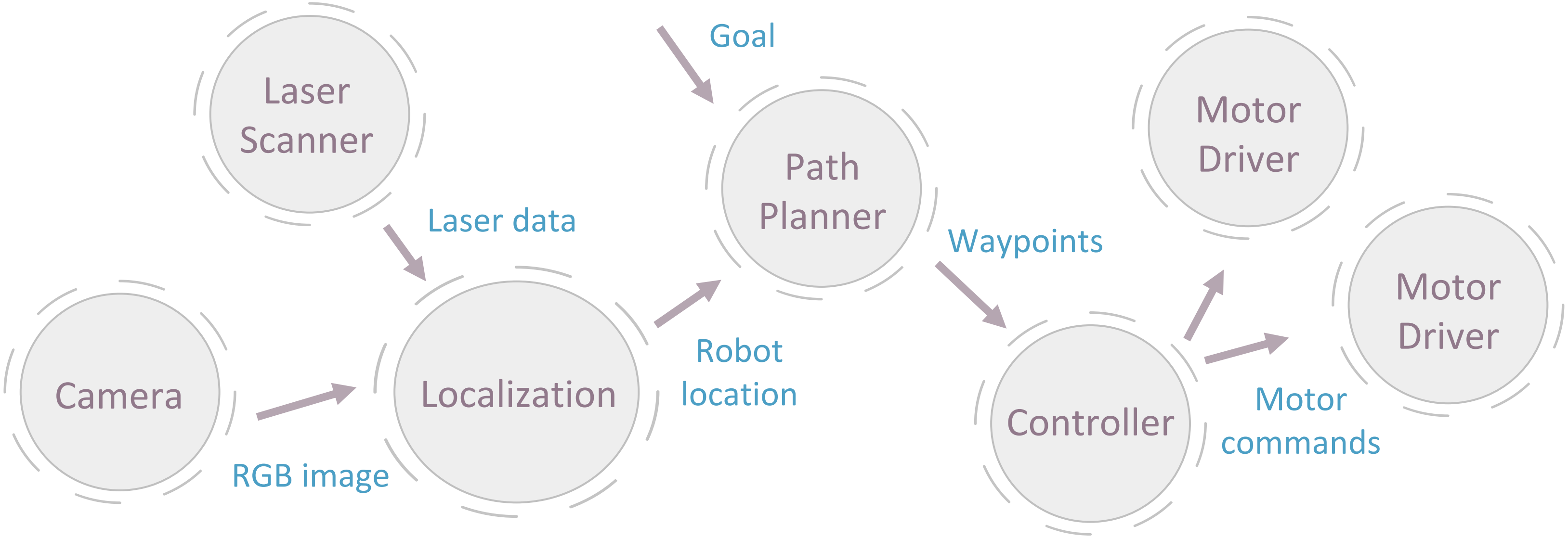
- Control
- Planning
- Manipulation
- Perception
- …

Community

- Software distribution
- Tutorials
- Support fora
- Conferences
- …

ROS

What is ROS?


Laser Scanner
Laser data
Camera
RGB image
Localization
Robot location
Goal
Path Planner
Waypoints
Controller
Motor commands
Motor Driver
Motor Driver

Features/Benefits

Distributed computation

- Divide software into small stand-alone parts.
- Programs can run on multiple computers and communicate over the network.

Communication protocol

Processes communicate over defined API.

(ROS messages, services,..)

Software reuse

Standard packages with implementations of many algorithms.

Supports multiple languages

C++, Python

Lisp, Java, Lua, MATLAB,
..

Open Source

Free to use.

De facto standard

for robotics programming.

Versions

ROS 1

ROS 2

Notes

- ROS 1 was **built for research**.
- ROS 2 aims to **address limitations** for commercial usage.
  - Security
  - Real-time Computing
  - Embedded Systems
  - …
- Core concepts still the same!

Installation

Platforms

ubuntu

debian

Distributions

Latest ROS 1 Distribution

ROS Noetic

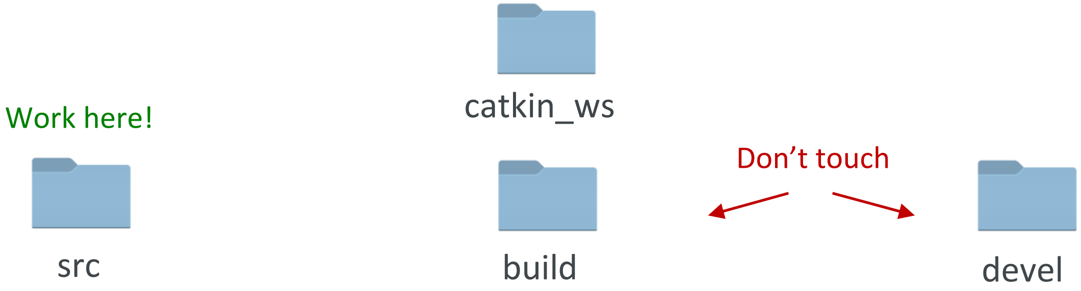
(Ubuntu 20.04)

Ways to Install

- On Ubuntu PC
- Dual-boot with Ubuntu
- Docker *
- Virtual Machine
- WSL on Windows

ROS Workspace

- catkin is the official ROS build system.
- A catkin workspace is a folder where you modify, build, and install catkin packages.



catkin_ws

Work here!

src

build

Don’t touch

devel

- Contains the source code of catkin packages.
- Where you clone, create and edit source code for the packages you want to build.

- Where CMake is invoked to build the packages in the source space.

- Where build targets are placed (prior to being installed).

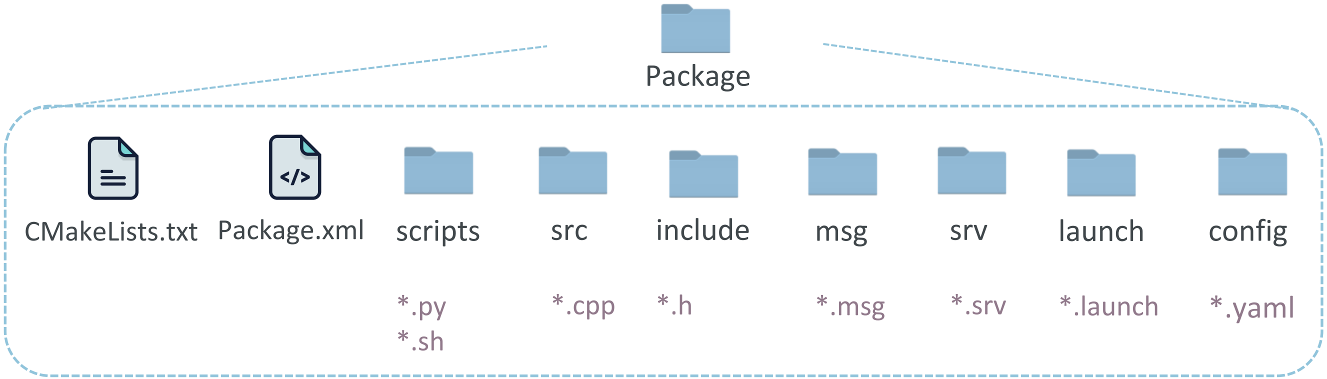
ROS Workspace

- All software is organized into (catkin) packages.

```
catkin_ws/
   build/
   devel/
   src/
      CMakeLists.txt
      Package 1/
         CMakeLists.txt
         package.xml
         …
      Package N/
         CMakeLists.txt
         package.xml
         …
```

For a package to be considered a catkin package, must contain:

- CMakeLists.txt -- info on how to build the package
- package.xml -- metadata
- Only 1 package in each folder (no nested packages)



Package

CMakeLists.txt

Package.xml

scripts
*.py
*.sh

src
*.cpp

include
*.h

msg
*.msg

srv
*.srv

launch
*.launch

config
*.yaml

Configuring Your ROS Environment

- Source your ROS environment *

```
$ source /opt/ros/noetic/setup.bash
```

- Source your catkin workspace

```
$ source ~/catkin_ws/devel/setup.bash
```

You need to run these commands on every new shell
OR
could add them to your .bashrc file.

How to edit your .bashrc file

- Open .bashrc file to edit

```
$ gedit ~/.bashrc
```

- Paste the following at the end of the file & save

```
source /opt/ros/noetic/setup.bash
source ~/catkin_ws/devel/setup.bash
echo "ROS Noetic & catkin_ws sourced!"
```

→ optional message

- Source .bashrc for changes to take effect

```
$ source ~/.bashrc
```

* Sets all the path variables to use the ROS built-in packages.

Building Your ROS Packages

- Step 1: Navigate to your catkin workspace

```
$ cd ~/catkin_ws/
```

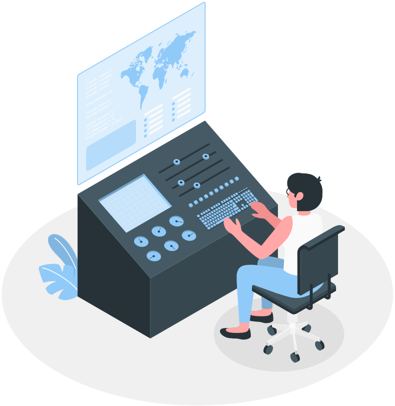
- Step 2: Build your packages

```
$ catkin build
```

- Step 3: Make the workspace visible to the file system

```
$ source devel/setup.bash
```

Use *catkin build* instead of *catkin_make*!
Don’t mix the two!



Installing Existing Packages

- Install Debian packages

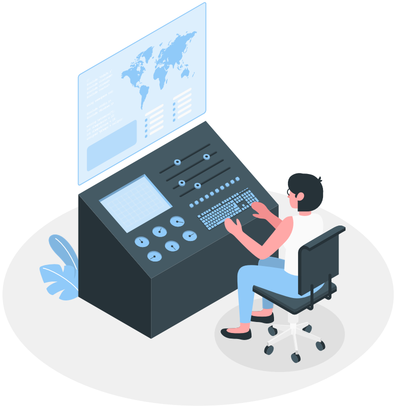
```
$ sudo apt update
$ sudo apt install ros-noetic-<package_name>
```

<ROS_distro>

- Install packages from GitHub

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/<username>/<repo>.git
```

  - Build your packages & source the workspace!



Creating a ROS Package

- Step 1: Navigate to source space dir of your catkin workspace

```
$ cd ~/catkin_ws/src
```

- Step 2: Create your packages with optional dependencies

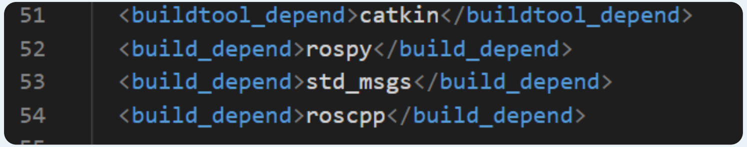
```
$ catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

Example:

```
$ catkin_create_pkg my_package std_msgs rospy roscpp
```

- Step 3: Build your packages & source the workspace!

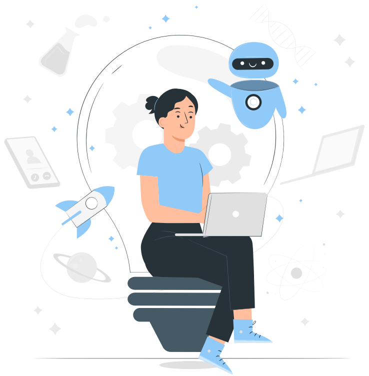
These first-order dependencies are stored in the package.xml file.



```
<buildtool_depend>catkin</buildtool_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>roscpp</build_depend>
```

Info on ROS packages

- *rospack* is the ROS package management tool.



Common uses:

- Find the absolute path to a package

```
$ rospack find <package_name>
```

- Get a list of all the package’s dependencies

```
$ rospack depends <package_name>
```

- Get a list of packages that depend on the given package

```
$ rospack depends-on <package_name>
```

Contents

Part I

Intro

- What is ROS?
- How to create/build your packages.

Part 2

ROS Ecosystem

- Fundamental concepts
- Basic commands
- Develop ROS nodes

Part 3

Simulation

- Rviz
- Control a robot in Gazebo.

ROS Nodes

- A node is a program that **performs some computation**.
- An **executable file** within a ROS package.
- Single-purpose.

.py or .cpp

- Run a node:

```
$ rosrun <package_name> <executable_name>
```

- Get a list of running nodes:

```
$ rosnode list
```

- Get information about a node:

```
$ rosnode info <node_name>
```

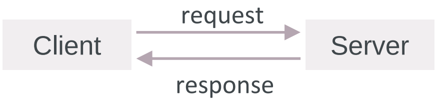
ROS Forms of Communication

Topics

- Message exchange
- For continuous data stream

Services

- Request-response type
- Blocks program execution
- For quick computations

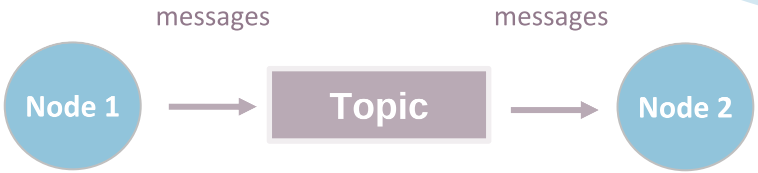


Actions

- Non-blocking
- Sends progress feedback to the client
- For goal-oriented tasks

ROS Topics

- ROS topics transport information between nodes.
- Nodes can **publish** and/or **subscribe** to a topic.
  - There can be multiple publishers and subscribers to a topic.
- Each topic has a specific ROS **message type**.



- List active topics:

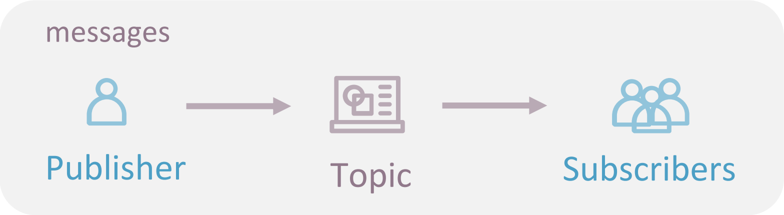
```
$ rostopic list
```

- Show information about a topic:

```
$ rostopic info /topic_name
```

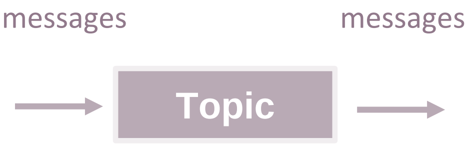
- Show current contents of a topic:

```
$ rostopic echo /topic_name
```



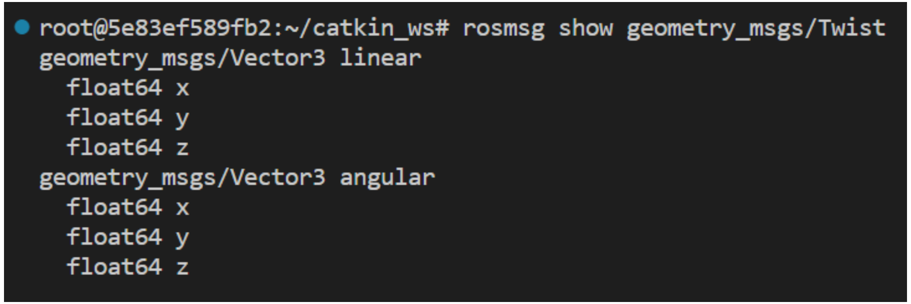
ROS Messages

- Each topic has a specific ROS **message type**.
- Data structures used to exchange data between nodes.



- Display the fields in a ROS message type:

```
$ rosmsg show <message_type>
```



```
root@5e83ef589fb2:~/catkin_ws# rosmsg show geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

To express velocity:

geometry_msgs/Twist.msg

Vector3 linear

Vector3 angular

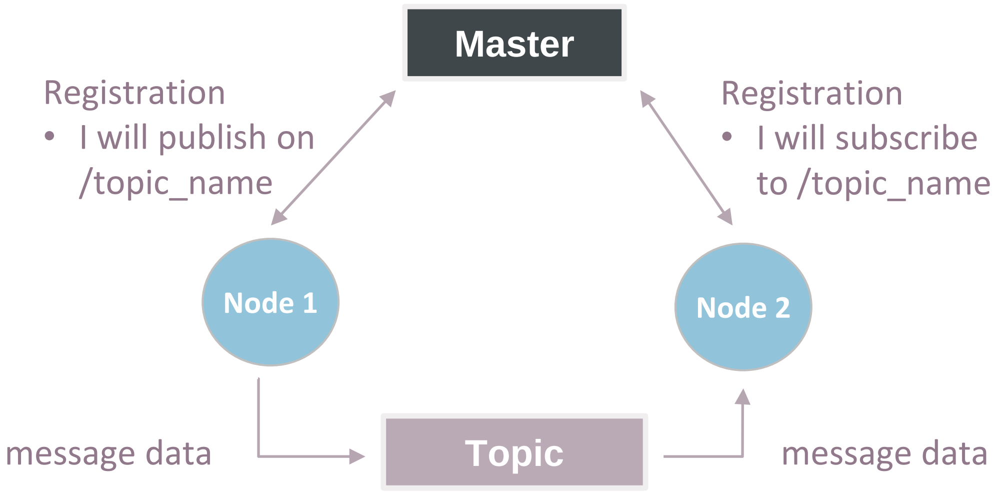
field type name

ROS Master

- Enable nodes to communicate with each other.
- All nodes need to register to Master at startup.
- Provides the Parameter Server.

- To start the ROS Master:

```
$ roscore
```



Fun Quiz

Question I

What is a ROS node?

A

A graphical tool to visualize the communication between ROS topics.

B

A computational process that performs a task.

C

A physical robot component (like a sensor or actuator).

Question 2

ROS Nodes can use any of the fundamental types of communication (Publisher, Subscriber, Services and Actions).

Often called: "Publisher Node", "Subscriber Node“, “Server”, etc.

Can a ROS node use a combination of these types?

A

Yes

B

No

Question 3

How many message types can be published to a topic?

A

An arbitrary amount.

B

1

Question 4

How many nodes can publish to a single topic?

A

Only one at a time.

B

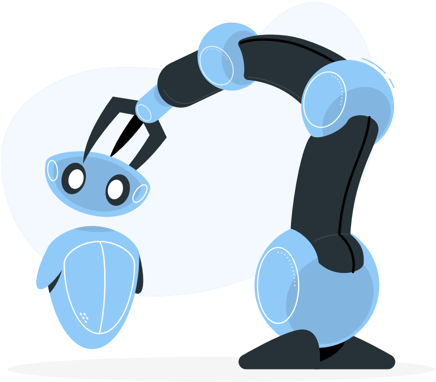
The amount is defined by the topic.

C

Any number of nodes can publish, if the message has the right type.

Example

turtlesim



Example - turtlesim

- Start ROS Master

```
$ roscore
```

- On a new terminal, run the turtlesim_node

```
$ rosrun turtlesim turtlesim_node
```

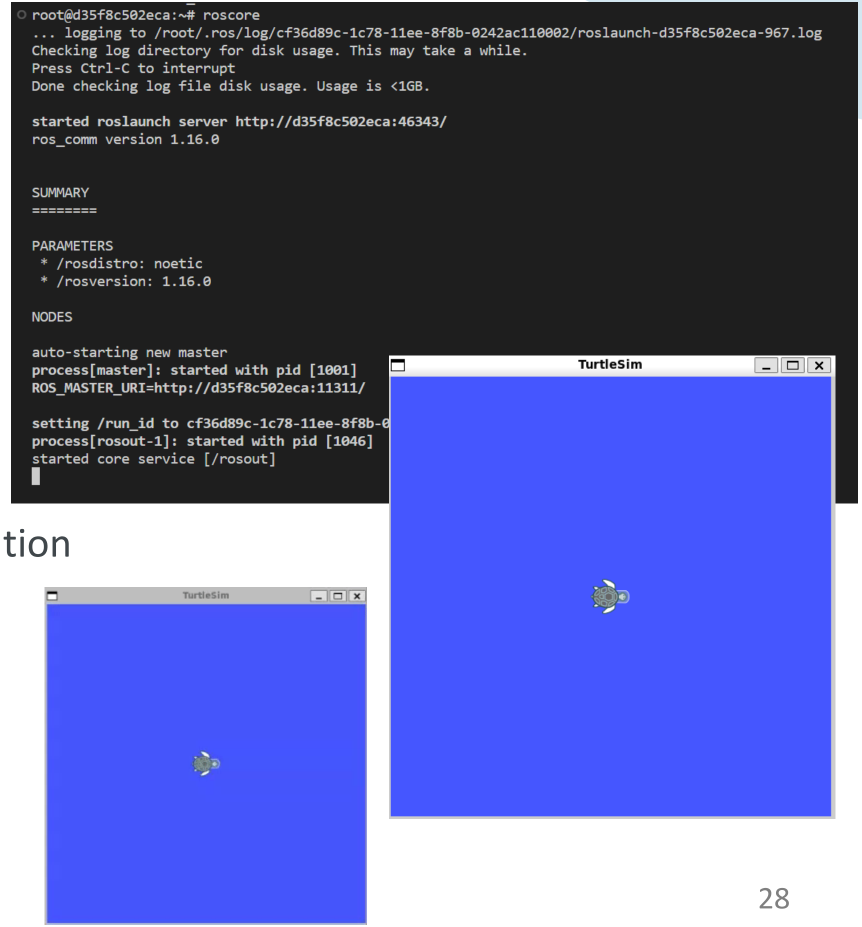
package name executable name

- On a new terminal, run the keyboard teleoperation

```
$ rosrun turtlesim turtle_teleop_key
```

- Press the arrow keys to move the turtle.

(Ensure the terminal with the teleoperation is in focus.)

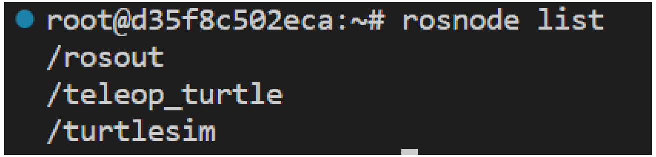


```
root@d35f8c502eca:~# roscore
... logging to /root/.ros/log/cf36d89c-1c78-11ee-8f8b-0242ac110002/roslaunch-d35f8c502eca-967.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://d35f8c502eca:46343/
ros_comm version 1.16.0


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.16.0

NODES

auto-starting new master
process[master]: started with pid [1001]
ROS_MASTER_URI=http://d35f8c502eca:11311/

setting /run_id to cf36d89c-1c78-11ee-8f8b-0
process[rosout-1]: started with pid [1046]
started core service [/rosout]
```

Example - turtlesim – Node Info

- Show list of running nodes

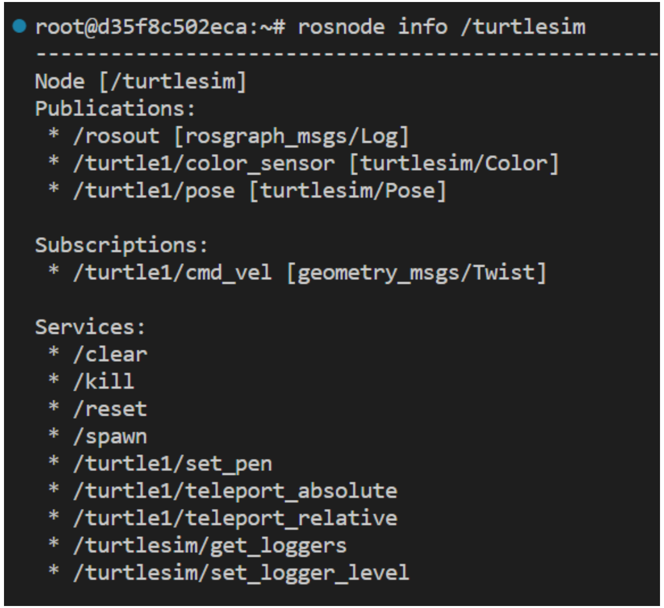
```
$ rosnode list
```



```
root@d35f8c502eca:~# rosnode list
/rosout
/teleop_turtle
/turtlesim
```

- Get information about the *turtlesim* node

```
$ rosnode info /turtlesim
```



```
root@d35f8c502eca:~# rosnode info /turtlesim
--------------------------------------------------
Node [/turtlesim]
Publications:
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/color_sensor [turtlesim/Color]
 * /turtle1/pose [turtlesim/Pose]

Subscriptions:
 * /turtle1/cmd_vel [geometry_msgs/Twist]

Services:
 * /clear
 * /kill
 * /reset
 * /spawn
 * /turtle1/set_pen
 * /turtle1/teleport_absolute
 * /turtle1/teleport_relative
 * /turtlesim/get_loggers
 * /turtlesim/set_logger_level
```

turtlesim **publishes** on these topics

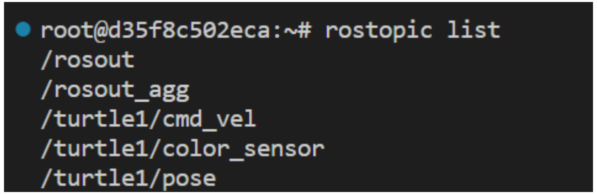
turtlesim **subscribes** to these topics

turtlesim can be configured using these services

Example - turtlesim – Topic Info

- List active topics

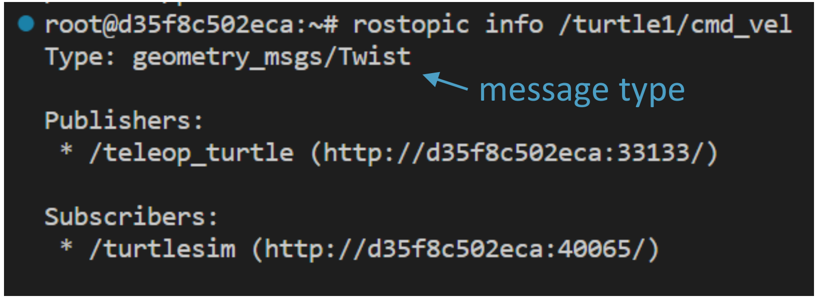
```
$ rostopic list
```



```
root@d35f8c502eca:~# rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

- Show info about the /turtle1/cmd_vel topic:

```
$ rostopic info /turtle1/cmd_vel
```

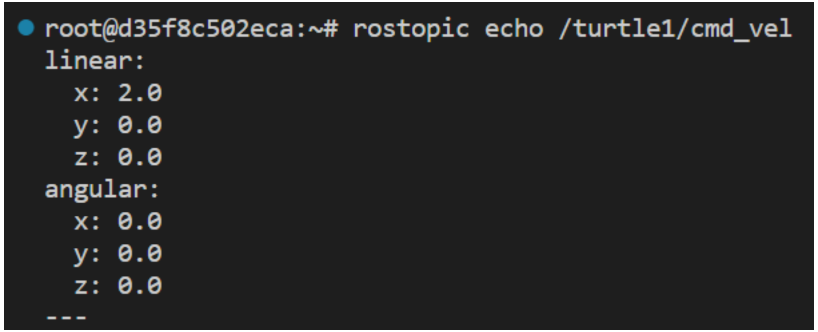


```
root@d35f8c502eca:~# rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_turtle (http://d35f8c502eca:33133/)

Subscribers:
 * /turtlesim (http://d35f8c502eca:40065/)
```

message type

- Show contents of /turtle1/cmd_vel topic:

```
$ rostopic echo /turtle1/cmd_vel
```

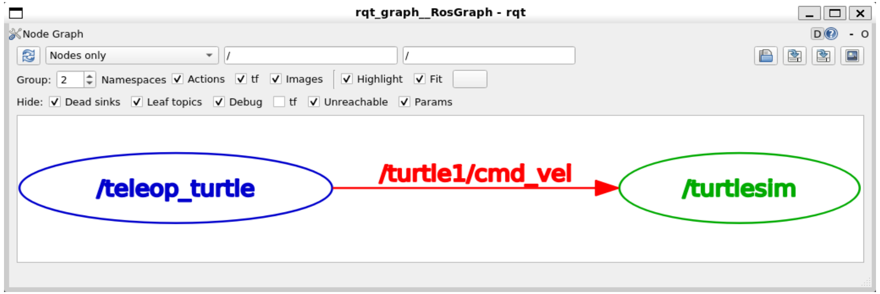


```
root@d35f8c502eca:~# rostopic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

Example - turtlesim – rqt

- Show computation graph

```
$ rosrun rqt_graph rqt_graph
```



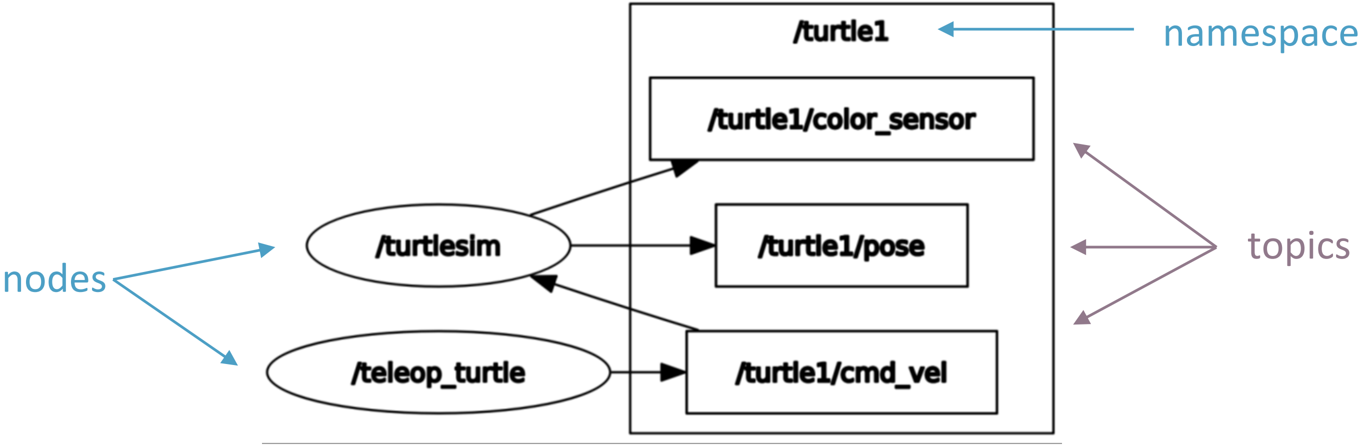
rqt_graph__RosGraph - rqt

Node Graph

Nodes only / /

Group: 2 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

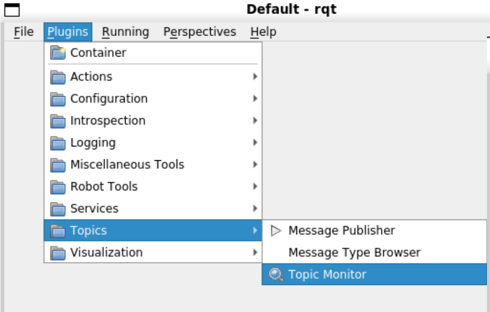
Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug ☐ tf ☑ Unreachable ☑ Params

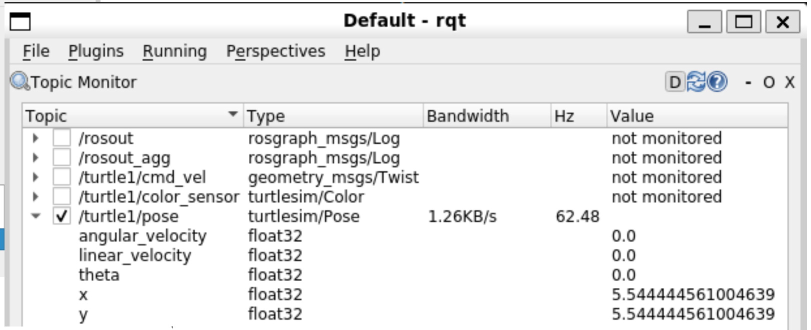


Example - turtlesim – rqt

- Run rqt tools

```
$ rqt
```

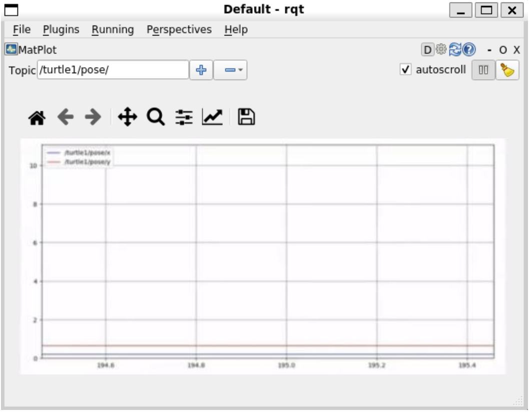




Topic	Type	Bandwidth	Hz	Value
☐ /rosout	rosgraph_msgs/Log			not monitored
☐ /rosout_agg	rosgraph_msgs/Log			not monitored
☐ /turtle1/cmd_vel	geometry_msgs/Twist			not monitored
☐ /turtle1/color_sensor	turtlesim/Color			not monitored
☑ /turtle1/pose	turtlesim/Pose	1.26KB/s	62.48	
angular_velocity	float32			0.0
linear_velocity	float32			0.0
theta	float32			0.0
x	float32			5.544444561004639
y	float32			5.544444561004639

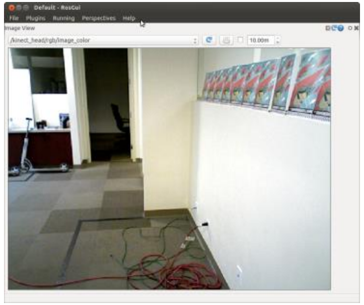
- Topic Monitor Plugin

  Plugins → Topics → Topic Monitor



- Plot Plugin

  Plugins → Visualization → Plot



- Image View Plugin

  Plugins → Visualization → Image View

Example - turtlesim – Publish message

- **Publish messages** to a given topic from terminal

```
$ rostopic pub [topic] [msg_type] [args]
```

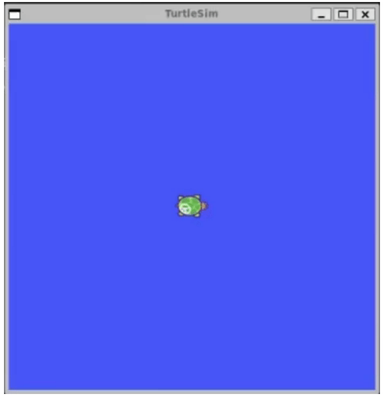
```
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]'
```

topic name

message type

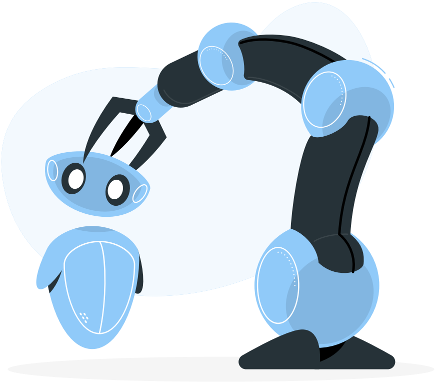
rate 1Hz
(optional)

data



Example

Simple Publisher & Subscriber



ROS Publisher & Subscriber (Python)

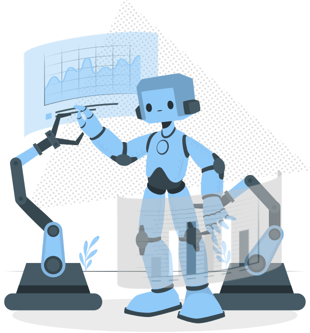
- Publishing to a topic (write messages)

```
pub = rospy.Publisher(‘topic_name’, message_type, queue_size)
```

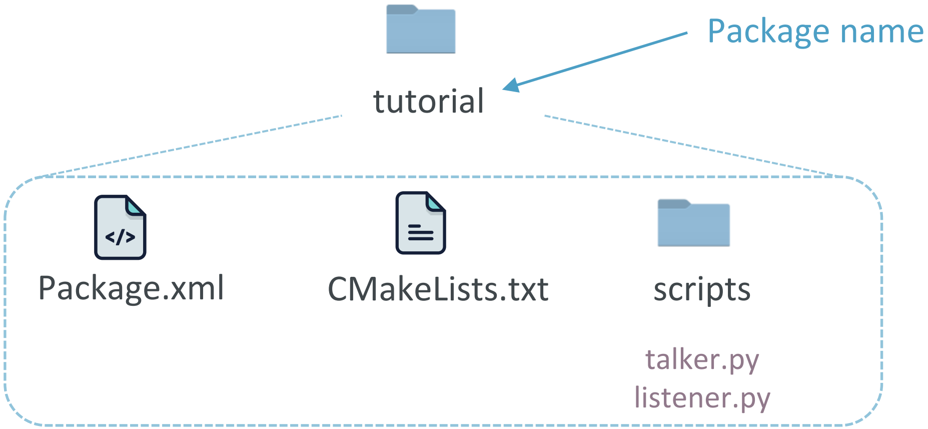
```
pub.publish(message)
```

- Subscribing to a topic (read messages)

```
sub = rospy.Subscriber(‘topic_name’, message_type, callback_function)
```



Example - Simple Publisher & Subscriber



Package name

tutorial

Package.xml

CMakeLists.txt

scripts

talker.py
listener.py

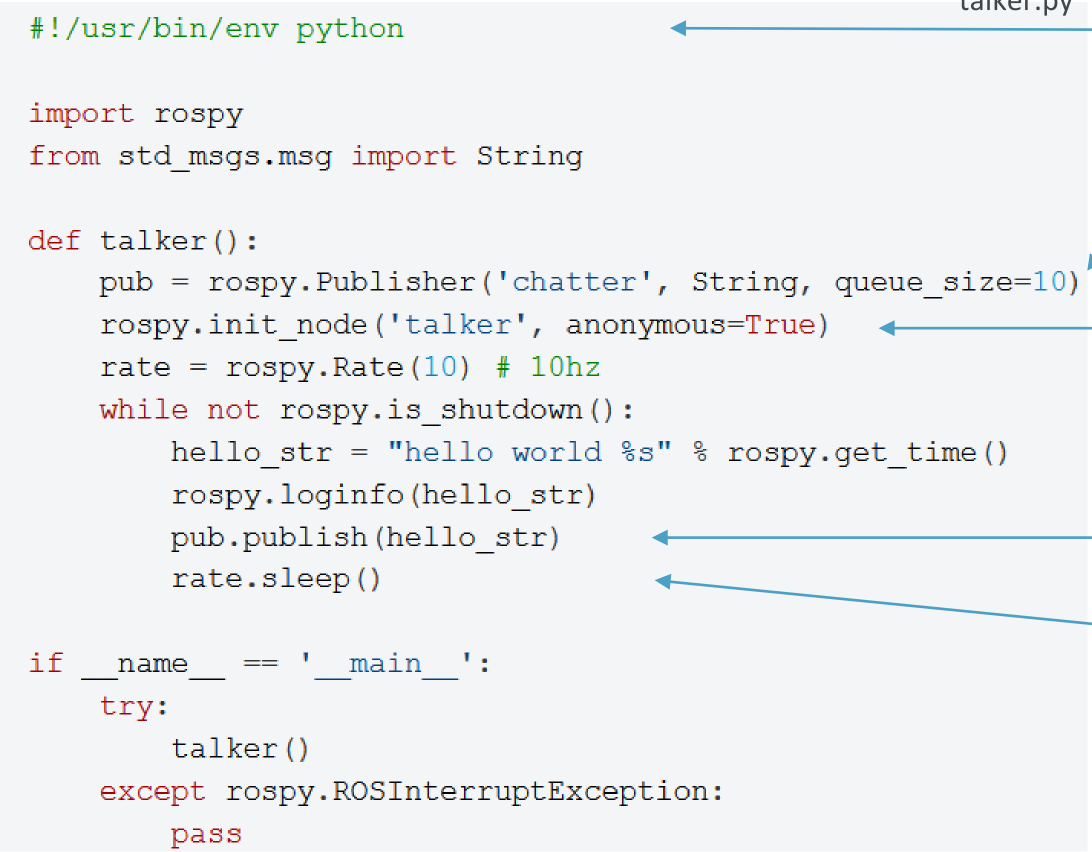
- Important Note: You need to make Python scripts executable!

```
$ cd ~/catkin_ws/src/tutorial/scripts
```

```
$ chmod +x *.py
```

Example - A Simple Publisher Node

talker.py



```python
#!/usr/bin/env python

import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

ensures it is executed as a Python script

declares that the node will publish to the *chatter* **topic,** using the **message type** *String*

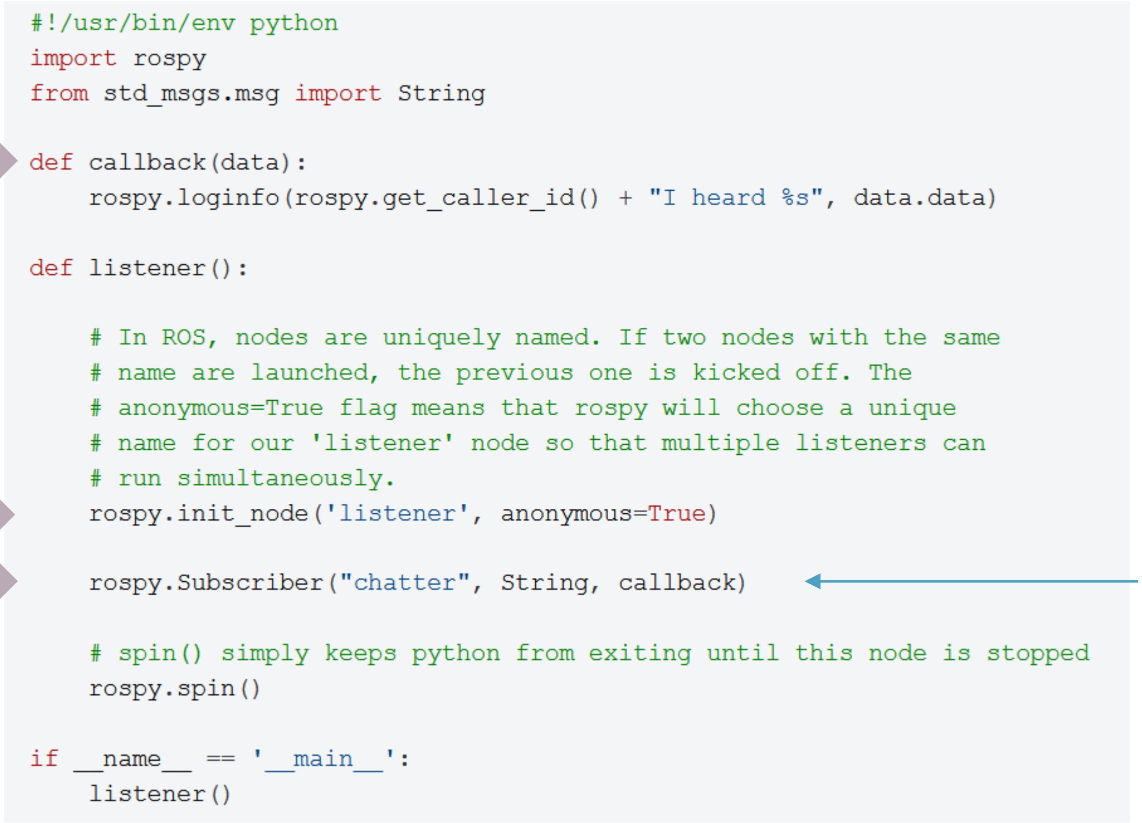
registers with Master

publishes a string to the *chatter* **topic**

to loop at specified frequency

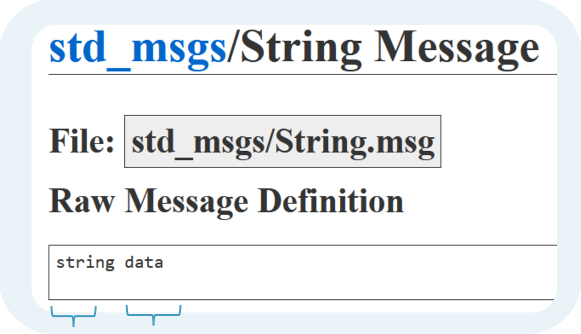
Example - A Simple Subscriber Node

listener.py



```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

declares that the node will subscribe to the *chatter* **topic** of **message type** *String*



std_msgs/String Message

File: **std_msgs/String.msg**

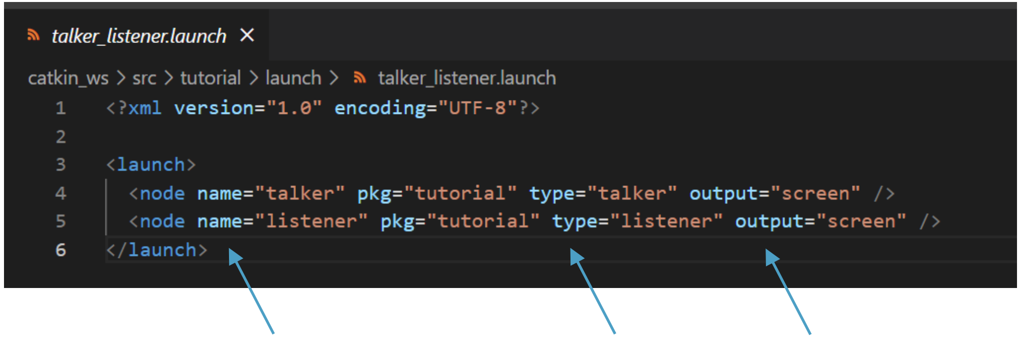
Raw Message Definition

```
string data
```

field type name

ROS Launch

- Starts multiple ROS nodes
- Sets parameters
- Written in XML
- Starts ***roscore***, if not already running


talker_listener.launch
catkin_ws > src > tutorial > launch > talker_listener.launch

```xml
<?xml version="1.0" encoding="UTF-8"?>

<launch>
  <node name="talker" pkg="tutorial" type="talker" output="screen" />
  <node name="listener" pkg="tutorial" type="listener" output="screen" />
</launch>
```

node name

(overrides name that node assigns to itself in its call to rospy.init_node)

executable name

where to output log messages (screen or log file)

- Run a launch file

```
$ roslaunch <package_name> <filename>.launch
```

- or navigate to the folder and run

```
$ roslaunch <filename>.launch
```

Contents

Part I

Intro

- What is ROS?
- How to create/build your packages.

Part 2

ROS Ecosystem

- Fundamental concepts
- Basic commands
- Develop ROS nodes

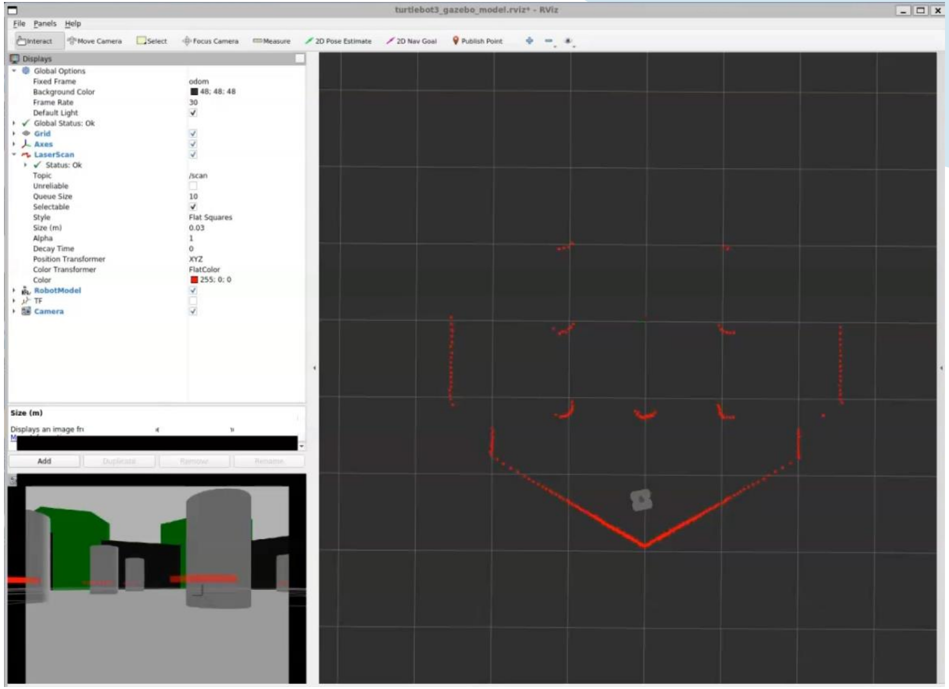
Part 3

Simulation

- Rviz
- Control a robot in Gazebo.

Rviz

- 3D visualizer for ROS
- Visualizes **sensor** and **state** information
- Visualization markers



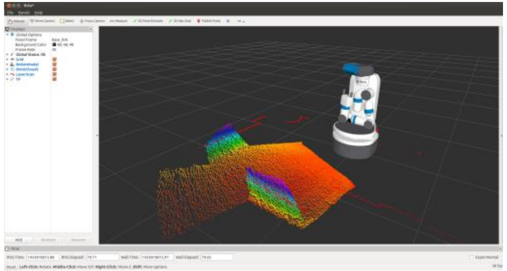
Global Options
Fixed Frame odom
Background Color 48; 48; 48
Frame Rate 30
Default Light ✓
✓ Global Status: Ok
Grid ✓
Axes ✓
LaserScan ✓
✓ Status: Ok
Topic /scan
Unreliable □
Queue Size 10
Selectable ✓
Style Flat Squares
Size (m) 0.03
Alpha 1
Decay Time 0
Position Transformer XYZ
Color Transformer FlatColor
Color 255; 0; 0
RobotModel ✓
TF □
Camera ✓

Size (m)

Displays an image fr

- Run Rviz

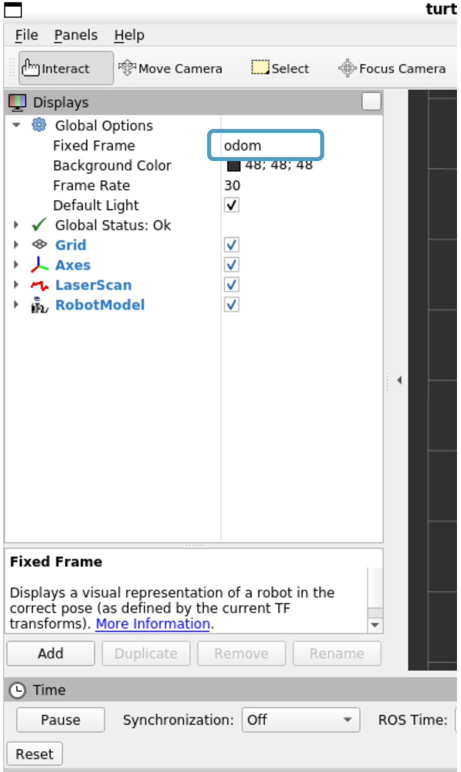
```
$ rosrun rviz rviz
```



Rviz

Frame in which data are displayed

Add display plugins



- Global Options
  - Fixed Frame: odom
  - Background Color: ■ 48; 48; 48
  - Frame Rate: 30
  - Default Light: ☑
- ✓ Global Status: Ok
- Grid ☑
- Axes ☑
- LaserScan ☑
- RobotModel ☑

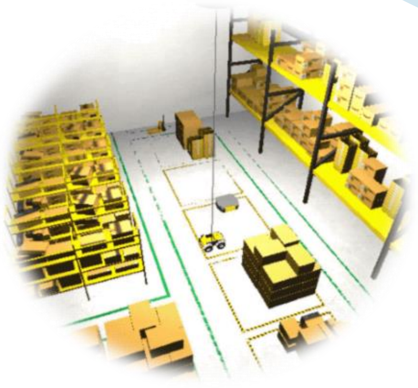
Fixed Frame

Displays a visual representation of a robot in the correct pose (as defined by the current TF transforms). More Information.

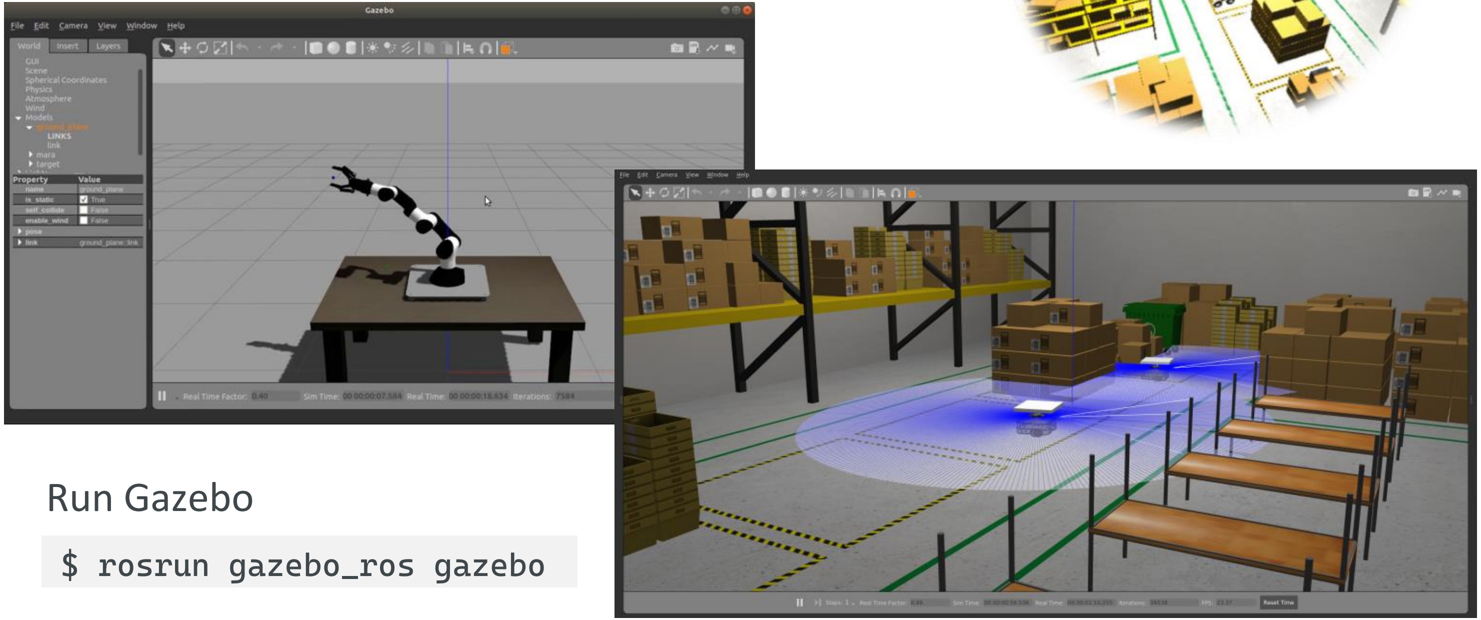
Synchronization: Off

- AccelStamped
- Axes
- Camera
- DepthCloud
- Effort
- FluidPressure
- Grid
- GridCells
- Group
- Illuminance
- Image
- InteractiveMarkers
- LaserScan
- Map
- Marker
- MarkerArray
- Odometry
- Path
- PointCloud
- PointCloud2
- PointStamped
- Polygon
- Pose
- PoseArray
- PoseWithCovariance
- Range
- RelativeHumidity
- RobotModel
- TF
- Temperature
- TwistStamped
- WrenchStamped

Gazebo Simulator



- 3D Physics-based simulator



Property	Value
name	ground_plane
is_static	☑ True
self_collide	□ False
enable_wind	□ False
pose	
link	ground_plane::link

Real Time Factor: 0.40 Sim Time: 00 00:00:07.584 Real Time: 00 00:00:18.634 Iterations: 7584

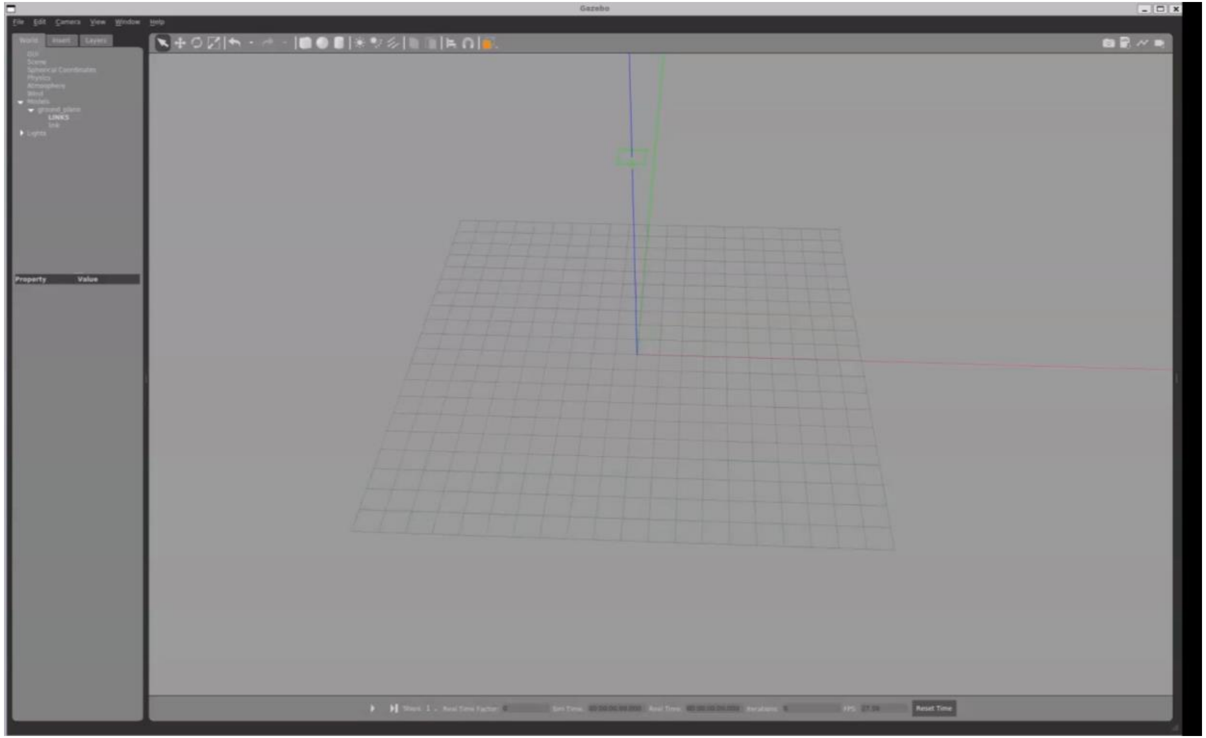
- Run Gazebo

```
$ rosrun gazebo_ros gazebo
```

or standalone:

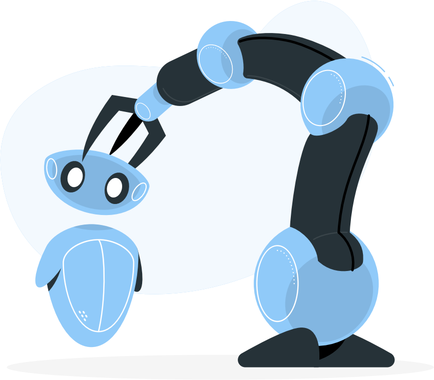
```
$ gazebo
```

Gazebo Simulator



Example

Go-to-Goal Control

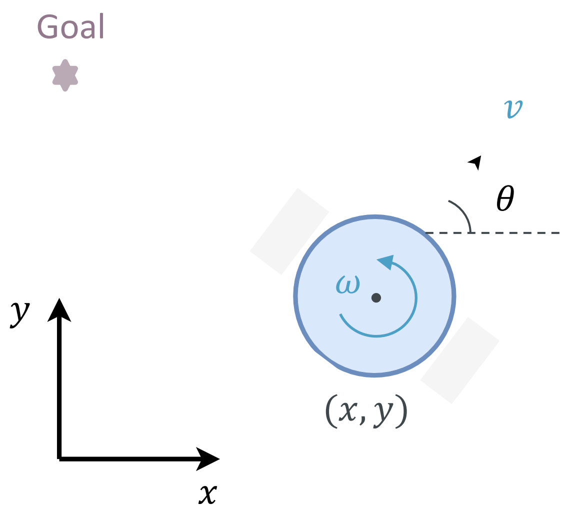


Example – Go-to-Goal Control

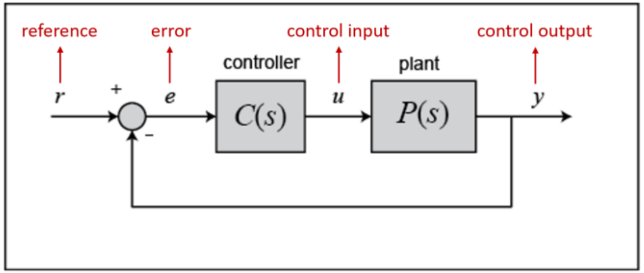
- “Unicycle” model
- Robot state: $\boldsymbol{x} = [x \quad y \quad \theta\ ]^T$
- Control inputs: $\boldsymbol{u} = [v \quad \omega]^T$

Linear velocity

Angular velocity

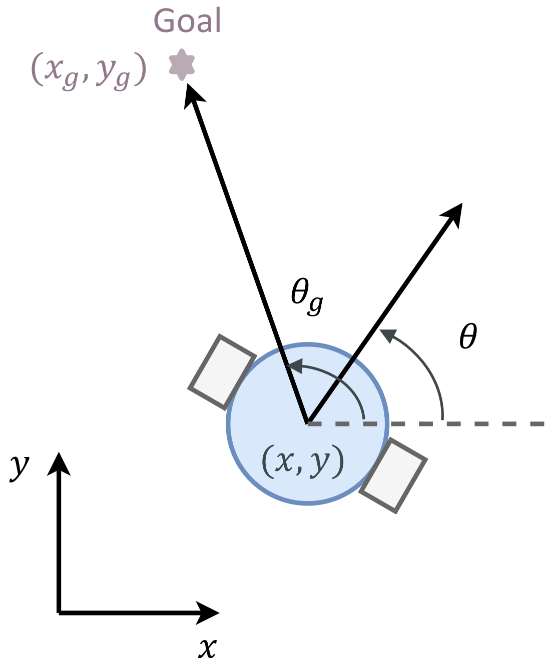


PID Control



$$u(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt}$$

Example – Go-to-Goal Control



Angular velocity

1. Heading angle to the goal:

$$\theta_g = atan2(y_g - y, x_g - x)$$

2. Heading error:

$$error = (\theta_g - \theta) = atan2(\sin(\theta_g - \theta), \cos(\theta_g - \theta))$$

3. Compute angular velocity:

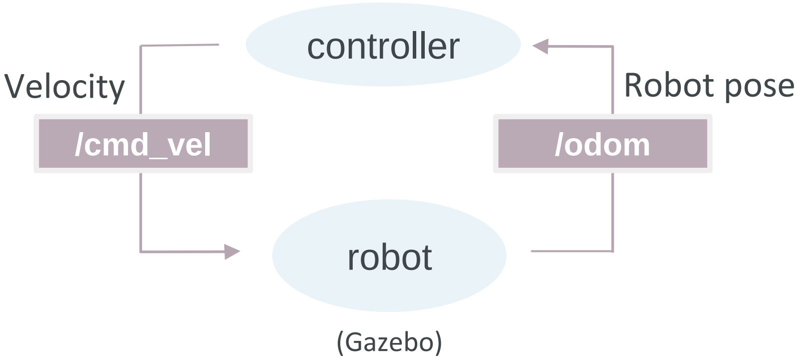
$$\omega = K_p \cdot error, \quad K_p > 0$$

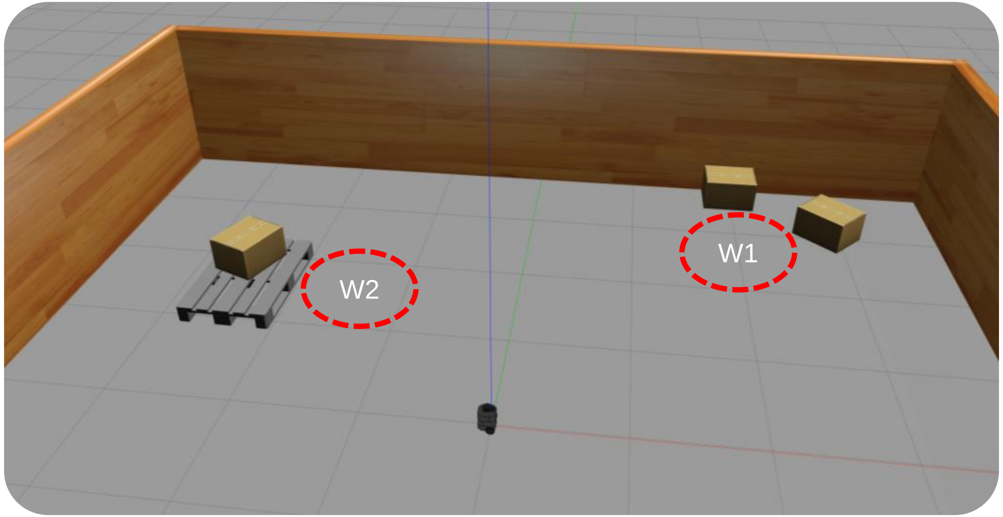
Linear velocity

$$\mathrm{v} = K_v \sqrt{(x_g - x)^2 + (y_g - y)^2} = K_v \cdot distance$$

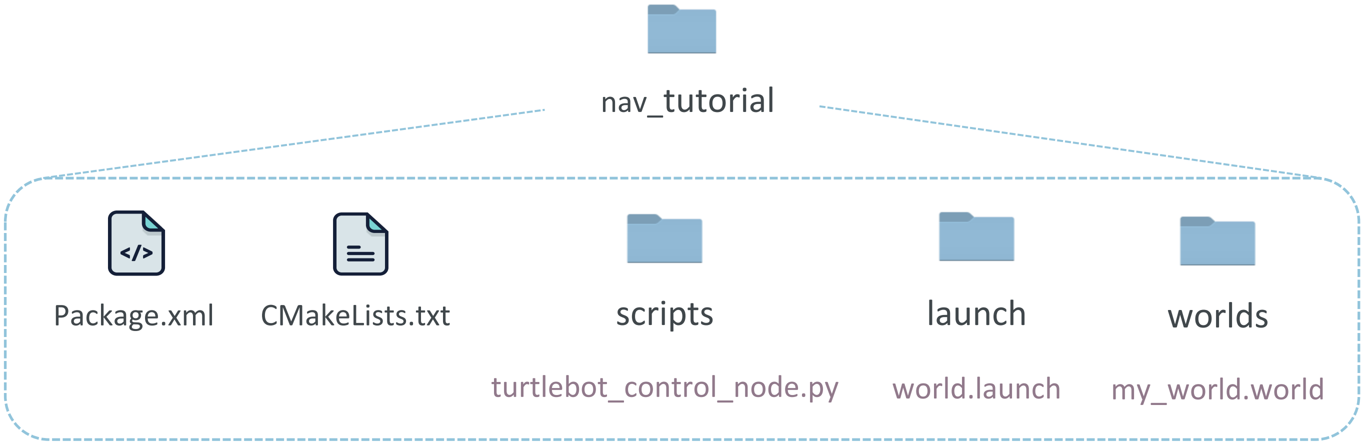
Example – Go-to-Goal Control

- Navigate to 2 waypoints
- TurtleBot3 robot





Example – Go-to-Goal Control



nav_tutorial

Package.xml

CMakeLists.txt

scripts

turtlebot_control_node.py

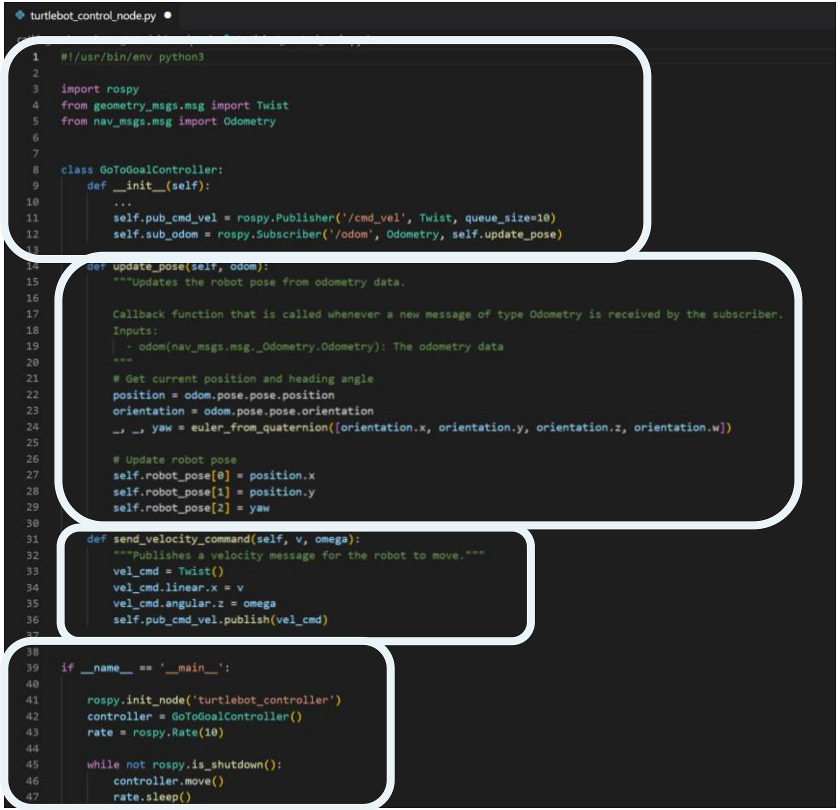
launch

world.launch

worlds

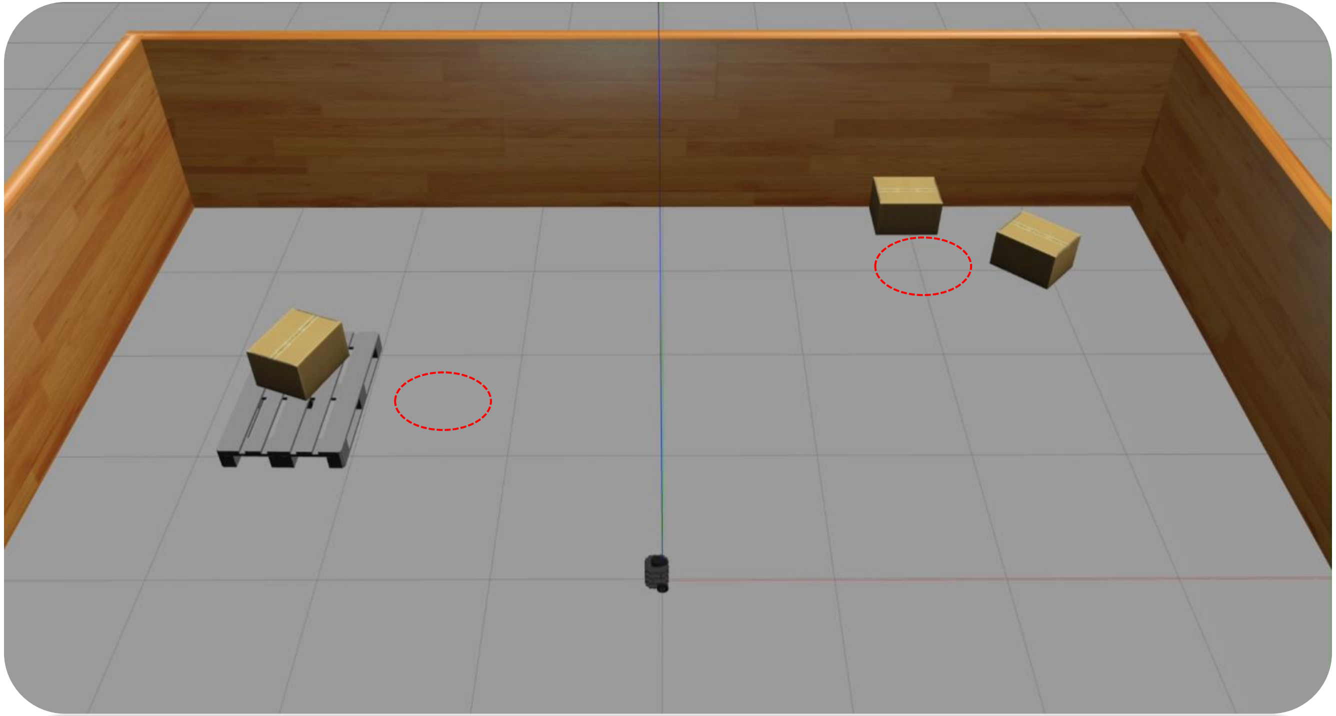
my_world.world

Example – Go-to-Goal Control



```python
#!/usr/bin/env python3

import rospy
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry


class GoToGoalController:
    def __init__(self):
        ...
        self.pub_cmd_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        self.sub_odom = rospy.Subscriber('/odom', Odometry, self.update_pose)

    def update_pose(self, odom):
        """Updates the robot pose from odometry data.

        Callback function that is called whenever a new message of type Odometry is received by the subscriber.
        Inputs:
          - odom(nav_msgs.msg._Odometry.Odometry): The odometry data
        """
        # Get current position and heading angle
        position = odom.pose.pose.position
        orientation = odom.pose.pose.orientation
        _, _, yaw = euler_from_quaternion([orientation.x, orientation.y, orientation.z, orientation.w])

        # Update robot pose
        self.robot_pose[0] = position.x
        self.robot_pose[1] = position.y
        self.robot_pose[2] = yaw

    def send_velocity_command(self, v, omega):
        """Publishes a velocity message for the robot to move."""
        vel_cmd = Twist()
        vel_cmd.linear.x = v
        vel_cmd.angular.z = omega
        self.pub_cmd_vel.publish(vel_cmd)


if __name__ == '__main__':

    rospy.init_node('turtlebot_controller')
    controller = GoToGoalController()
    rate = rospy.Rate(10)

    while not rospy.is_shutdown():
        controller.move()
        rate.sleep()
```

Example – Go-to-Goal Control



MoveIt

Motion Planning

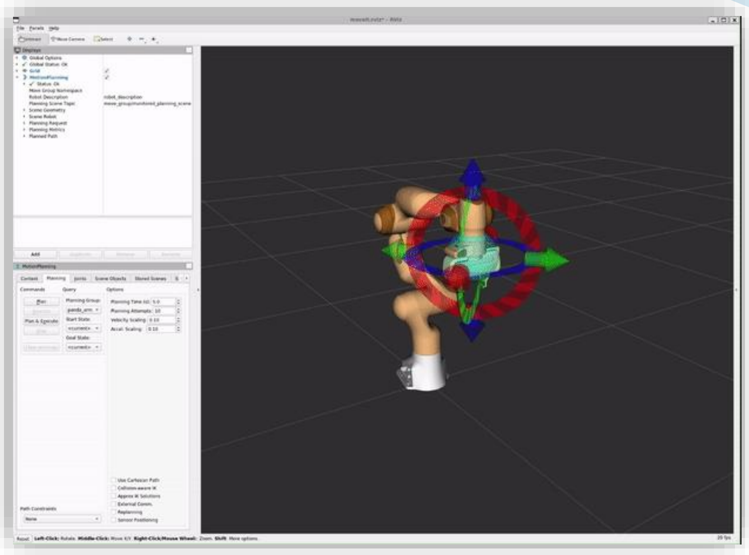
Inverse Kinematics

Control

Manipulation

Collision Checking

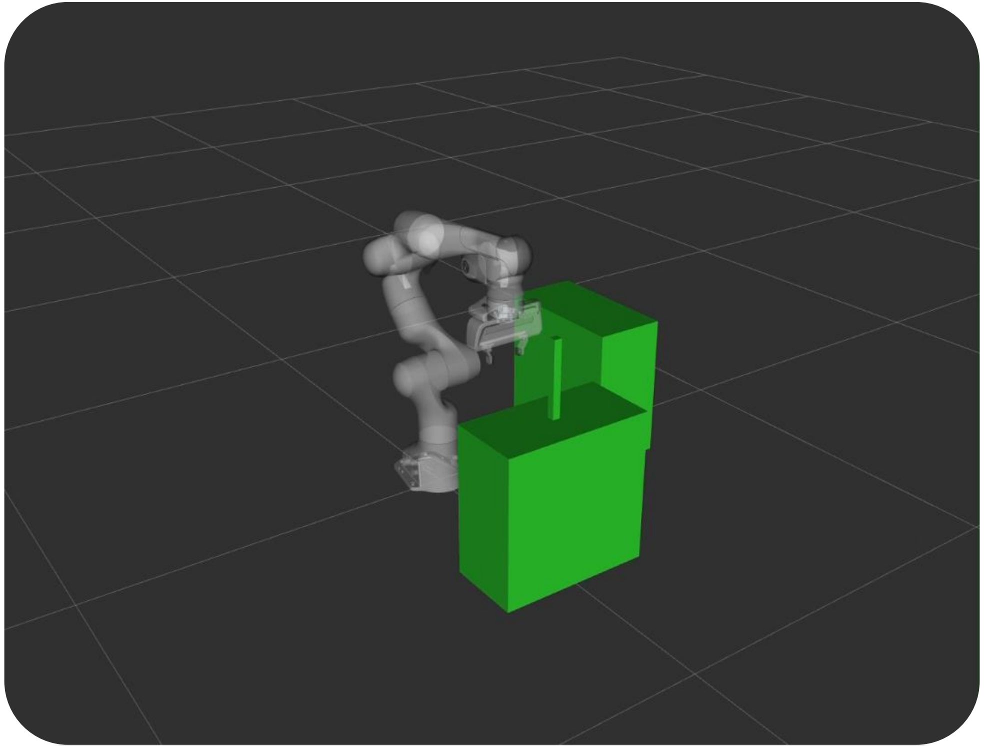
3D Perception



Why MoveIt?

By incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation, MoveIt is state of the art software for mobile manipulation.

MoveIt



ROS Resources

Wiki

http://wiki.ros.org/

Installation

http://wiki.ros.org/ROS/Installation

Launch files

http://wiki.ros.org/roslaunch/XML

Tutorials

http://wiki.ros.org/ROS/Tutorials

Recommended: Beginner Level 1-6, 11-14

Transforms

http://wiki.ros.org/tf2

Support forum

https://answers.ros.org/
https://robotics.stackexchange.com/

Ask ROS related questions here!

TurtleBot 3

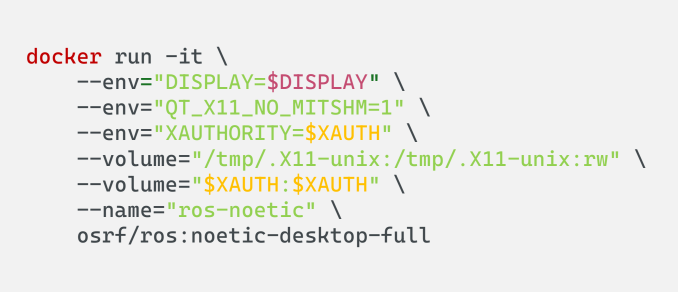
https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

MoveIt

https://moveit.ros.org/

ROS on Docker

- ROS image:
  osrf/ros:noetic-desktop-full
- Docker command for graphics support on Windows:



```
docker run -it \
    --env="DISPLAY=$DISPLAY" \
    --env="QT_X11_NO_MITSHM=1" \
    --env="XAUTHORITY=$XAUTH" \
    --volume="/tmp/.X11-unix:/tmp/.X11-unix:rw" \
    --volume="$XAUTH:$XAUTH" \
    --name="ros-noetic" \
    osrf/ros:noetic-desktop-full
```

How To Attach Visual Studio Code To A Running Docker Container

- Install the Docker extension on VS Code

Once you have the container running:

- Select the docker extension in VS Code (left pane)
- Right-click on your container
- Select *"Attach Visual Studio Code"*

Useful VS Code extensions:

- Python, C/C++
- CMake
- Docker
- ROS

Thank you!

Any questions?

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon** and illustrations by **Storyset**.