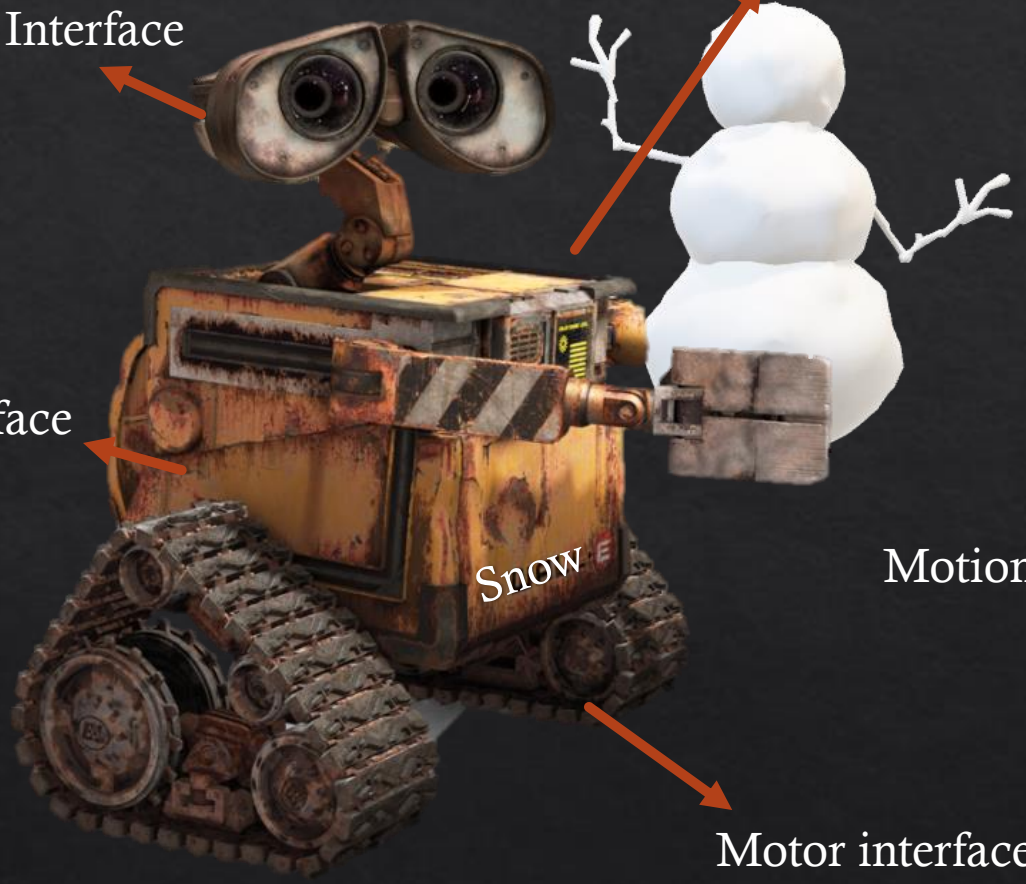
Brain teaser

- Snow-E
- #DDSMe
  - Disney don't sue me

Brain teaser



Battery interface

Camera Interface

Battery Manager

Image processing

Compression Chamber interface

Motion Planning

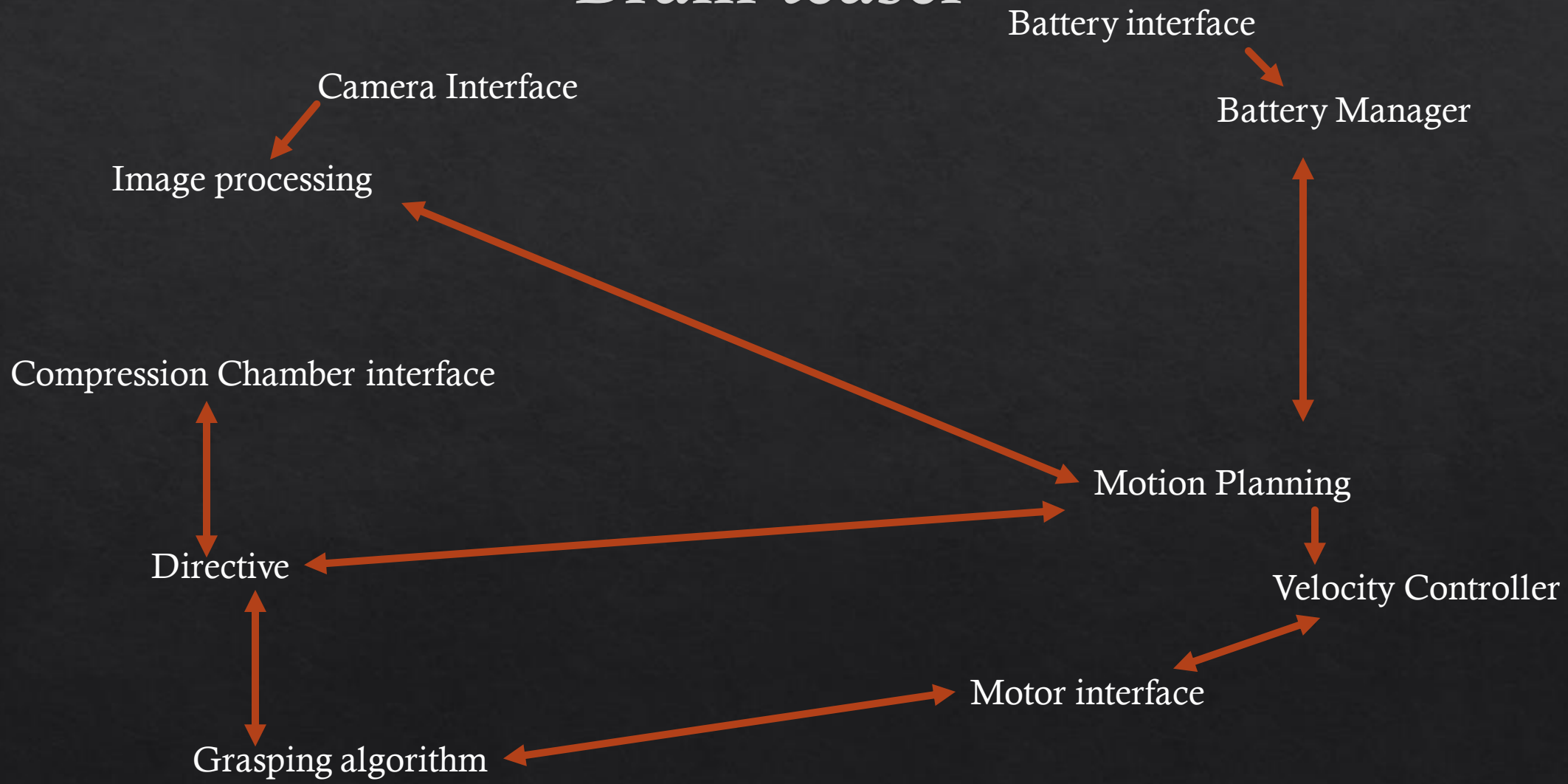
Directive

Velocity Controller

Motor interface

Grasping algorithm

Brain teaser



Outline

- Why ROS and what is it?
- Communication protocol
- How to get started
- Sample C++ code
- Sample python code
- Launch Files
- Parameters
- Services examples
- Actions examples
- Useful commands and libraries

Why ROS and what is it?

- ROS = Robot Operating System
- Is a Middleware framework
- Started by Stanford AI lab, later developed by Willow Garage
- Supports: Ubuntu*, Mac OS X, Windows, Raspbian, QNX.
- There are other middlewares: YARP, HOP, Player, ARIA, Carmen, iRobot Aware, Orocos, RT middleware, Webots, etc.
  - A comparison can be found in Tsardoulias, E., & Mitkas, P.A. (2017). Robotic frameworks, architectures and middleware comparison. *ArXiv, abs/1711.06842*. (https://arxiv.org/pdf/1711.06842.pdf)

Why ROS and what is it?

- Goal: “Stop reinventing the wheel.”

- How:
  - Foment code re-use and modularity
  - Opensource
  - Language agnostic
  - Easy to integrate
  - Adoption
  - Amount of available libraries
  - Very active developer community

Why ROS and what is it?

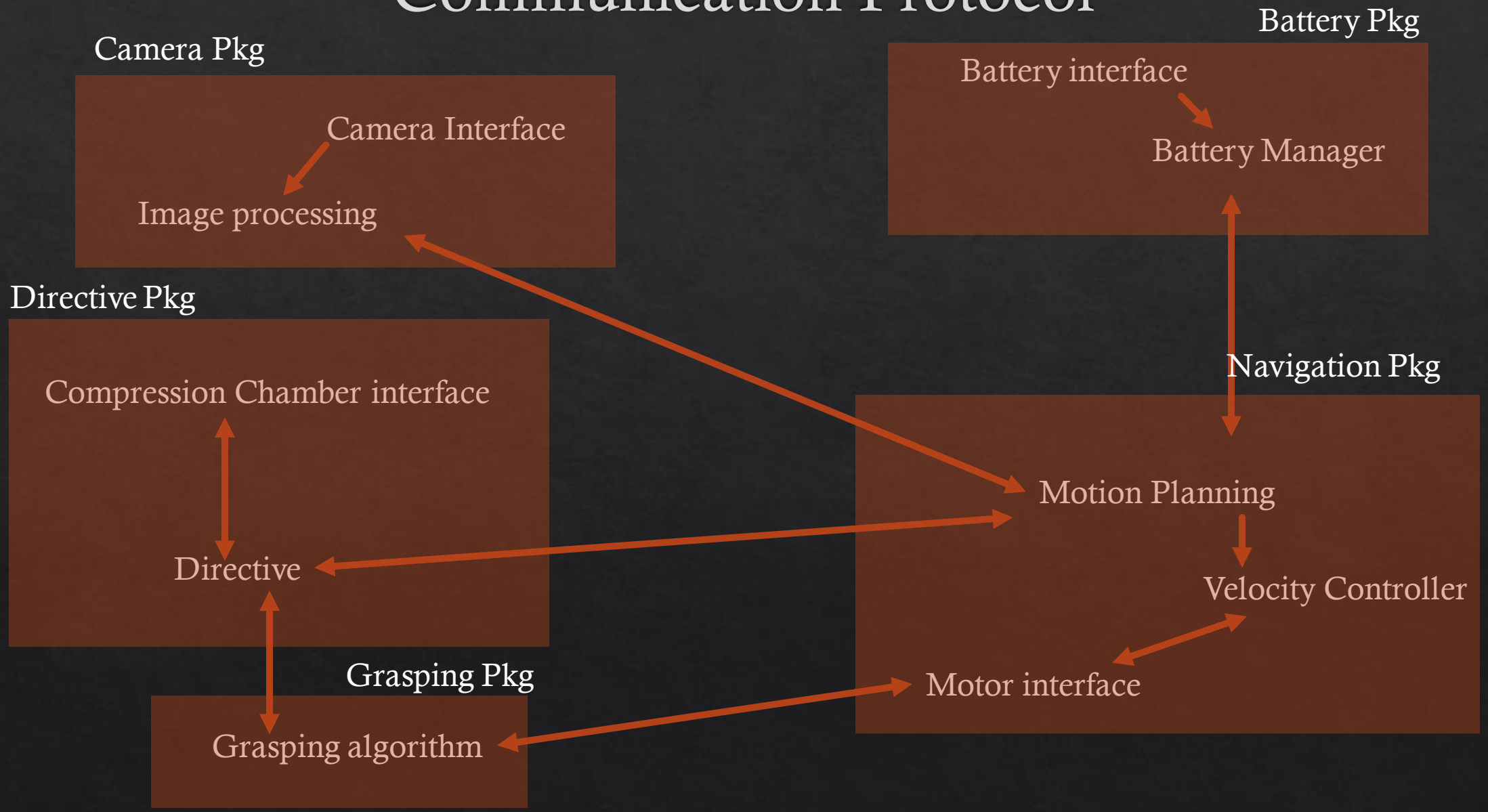
- Cons:
  - Opensource… not everything is guaranteed to work
  - Not real time guaranteed
  - Security[1]

[1] R. R. Teixeira, I. P. Maurell and P. L. J. Drews, "Security on ROS: analyzing and exploiting vulnerabilities of ROS-based systems," *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*, 2020, pp. 1-6, doi: 10.1109/LARS/SBR/WRE51543.2020.9307107.

Outline

- Why ROS and what is it?
- Communication protocol
- How to get started
- Sample C++ code
- Sample python code
- Launch Files
- Parameters
- Services examples
- Actions examples
- Useful commands and libraries

Communication Protocol



Communication Protocol

- Main node is **roscore**
- Messages: format in which information is exchanged on nodes
  - Primitive types: boolean, integer, floating-point.
  - Arrays of primitive types.
  - Nested structures and array
  - User can define custom messages

Communication Protocol

- Topics: named bus over which nodes exchange messages
  - 1-N communication
  - Publish/Subscriber
- Services
  - Client/Server
  - Synchronous
- Actions
  - Uses topics internally
  - Asynchronus

Communication Protocol

Service vs Actions in a nutshell

1. Go the Pizza shop
    1. Place your order.
    2. *Wait for the order.*
    3. Get the Pizza.
2. Order online.
    1. Place your order.
    2. Order confirmation notice.
    3. Possibly cancel the order.
    4. Check up your order status once in a while.
    5. *Do other things.*
    6. Pizza is delivered.

Communication Protocol

Topics example

- Getting to know a topic:
  - *rostopic list*
  - *rqt_graph*
  - *rostopic info /walking_controller/cmd_vel*
  - *rosmsg show geometry_msgs/Twist*
- Using a topic:
  - *rostopic pub /walking_controller/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 0]'*
  - *rostopic pub /walking_controller/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 0]'*

Communication Protocol

Services :

- Getting to know a service:
  - *rosservice info /walking_controller/do_step*
  - *rosservice call /walking_controller/do_step "step: pose: x: 0.3 y: -0.25 theta: 0.0 leg: 0“*
  - *rosservice call /walking_controller/do_step "step: pose: x: 0.3 y: 0.25 theta: 0.0 leg: 1"*

- Data before --- are request, after is the response

**Tip:* You can double tap after the name of the service to get a skeleton of the message. Copy and paste in a normal text editor for easier filling of variables

Communication Protocol

Actions :

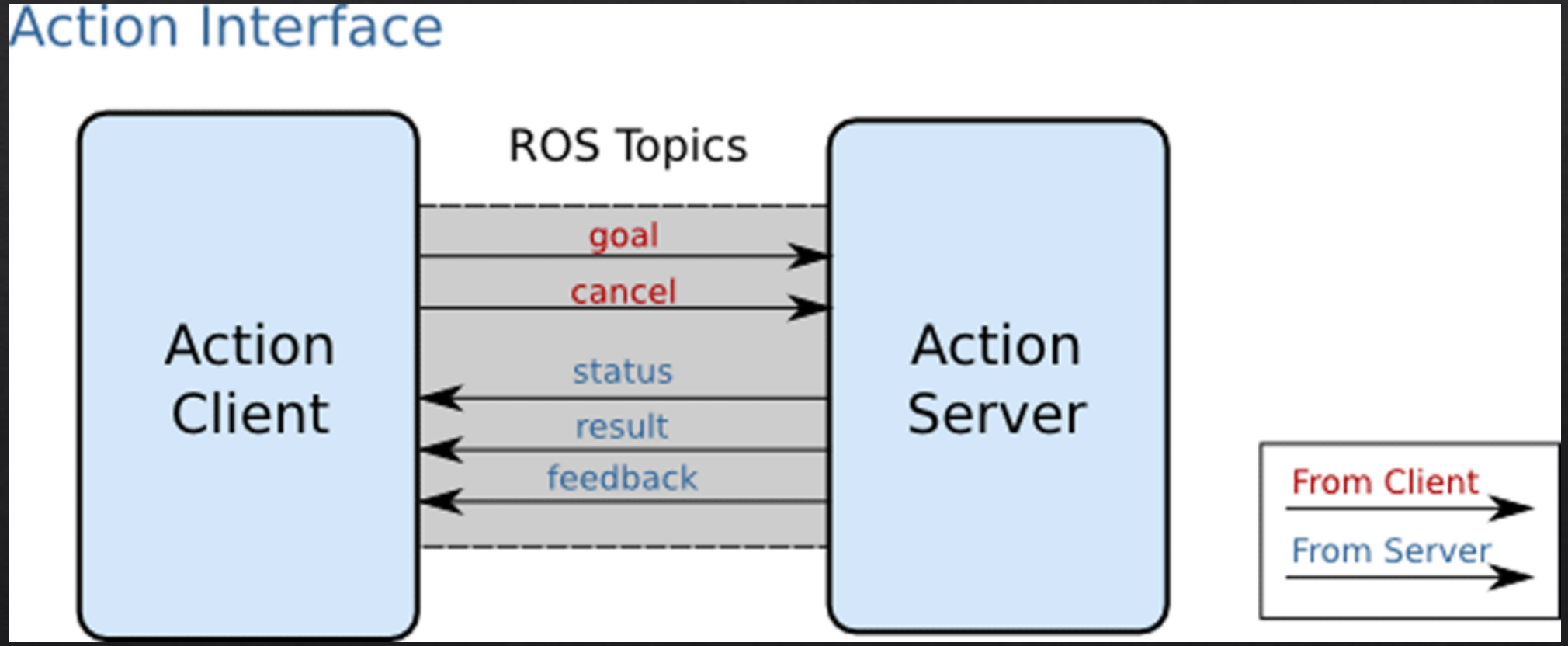
- Actions have multiple topics: */cancel*, */feedback*, */goal*, */result* and */status*

- Getting to know an action:
  - *rostopic info /play_motion/goal*
  - *rosmsg show play_motion_msgs/PlayMotionActionGoal*
  - *rostopic pub /play_motion/goal play_motion_msgs/PlayMotionActionGoal "header: seq: 0 stamp: secs: 0 nsecs: 0 frame_id:"goal_id: stamp: secs: 0 nsecs: 0 id: "goal: motion_name: 'yes' skip_planning:false priority: 0"*
  - Other acceptable motion names: alive_waiting_4, prepare_for_dance, point_lasers, point_head, strong_arms

**Tip:* You can double tap after the name of the action to get a skeleton of the message. Copy and paste in a normal text editor for easier filling of variables

Communication Protocol

Actions :



Communication Protocol

- Parameter Server
  - Shared dictionary accessible via network APIs
  - Used to store/retrieve parameters at runtime.
  - Suited for static data
  - Parameter types: 32 bit integers, Booleans, Strings, …
  - There are global parameters and private parameters, specific to a node.

Outline

- Why ROS and what is it?
- Communication protocol
- How to get started
- Sample C++ code
- Sample python code
- Launch Files
- Parameters
- Services examples
- Actions examples
- Useful commands and libraries

How to get started

- Install ROS
- Which version:
  - Ubuntu 16 → Kinetic
  - Ubuntu 18 → Melodic*
  - Ubuntu 20 → Noetic
- Most maintained libraries will have compatibility with the latest two
- Need a specific version, don’t want to change OS→ Docker

How to get started

Create a package

- The default way is to use Catkin Workspaces.
- Catking is the official build system of ROS
- Combines CMake macros and Python scripts to provide some functionality on top of Cmake
- Catkin Workspace can have multiple packages
- Using *catkin_create_pkg <package_name> [depend1] … [dependN]* helps initialize a package
- It generates and initial package.xml and CMakeLists.txt

How to get started

- Package.xml file

Package name

```
<?xml v e r s i on="1.0"?>
<package>
<name>my_package_name</name>
<ve r s ion >0.0.1</ ve r s ion>
<d e s c r i p t i o n>
Examples from A Gent le Introduction to ROS
</d e s c r i p t i o n>
<maintaine r emai l="jokane@cse.sc.edu">
Jason O' Kane
</maintainer>
<l i c e n s e >TODO</l i c e n s e >
<bui ldtool_depend>catkin </bui ldtool_depend>
<bui ld_depend>geometry_msgs</bui ld_depend>
<run_depend>geometry_msgs</run_depend>
<bui ld_depend>tur t l e s im </bui ld_depend>
<run_depend>tur t l e s im </run_depend>
</package>
```

Package dependencies

How to get started

Typical workspace and package structure

- Catkin_workspace
  - src
    - Package_folder
      - src
      - include
      - launch
      - scripts
      - msg
      - srv
      - action
      - config

How to get started

- C++ example:

```
// This is a ROS version of the standard "hello , world"
// program.

// This header defines the standard ROS classes .
#include <ros/ros.h >

int main ( int argc , char ** argv ) {
// Initialize the ROS system.
ros::init ( argc , argv , " hello_ros " ) ;

// Establish this program as a ROS node .
ros::NodeHandle nh ;

// Send some output as a log message .
ROS_INFO_STREAM( "Hello , ROS! " ) ;
}
```

Required for standard ROS classes

Initializes ROS client library

Object Handler for Ros nodes

ROS version of cout (messages to diff locations)

How to get started

- CMakeLists.txt example:

```
# What version of CMake is needed ?
cmake_minimum_required (VERSION 2.8.3 )

# Name o f this package .
project ( my_package_name )

# Find the catkin build system , and any other packages on
# which we depend .
find_package ( catkin REQUIRED COMPONENTS roscpp )

# Declare our catkin package .
catkin_package ( )

# Specify locations of header files .
include_directories ( include ${catkin_INCLUDE_DIRS})
```

Required for compiling ROS through catkin, add other package names after COMPONENTS

Other packages should match the ones declared in package.xml

To find the headers

How to get started

- CMakeLists.txt example:

Add the name of the executable file and source files

```
# Declare the executable , along with its sourcefiles . If
# there are multiple executables , use multiple copies o f
# this line .
add_executable ( hello hello.cpp )

# Specify libraries against which to link . Again , this
# line should be copied for each distinct executable in
# the package .
target_link_libraries ( hello ${catkin_LIBRARIES })
```

To use appropriate library flags

How to get started

Compile your program

- Go back to the catkin workspace folder
- Do either *catkin_make* or *catkin build**
  - *catkin_make* builds the whole workspace as a whole. It shares dependencies
  - *catkin build* builds each package in isolation with its own dependencies

Run program using *rosrun my_package_name executable_name*

Outline

- Why ROS and what is it?
- Communication protocol
- How to get started
- Sample C++ code
- Sample python code
- Launch Files
- Parameters
- Services examples
- Actions examples
- Useful commands and libraries

C++ code publisher

- Include ros/rosh.h and any type of message you are planning to use
- Init ros and declare a node handler
- Create a publisher object

```
ros::Publisher pub = node_handle.advertise<message_type>(topic_name, queue_size);
```

- One publisher per topic

C++ code publisher

- Fill a msg. Consider it an object, fill each field obtained through *rosmsg show*

"linear:
x: 0.0
y: 0.0
z: 0.0
Angular:
x: 0.0
y: 0.0
z: 0.0"

```
geometry_msgs::Twist msg;
msg.linear.x = double(rand())/double(RAND_MAX);
msg.angular.z = 2*double(rand())/double(RAND_MAX) - 1;
```

- Publish
  - *pub.publish(msg);*

C++ code publisher

- Publish loop
  - Set the rate *ros::Rate rate(Hz);* This checks for node shutdown
  - Loop condition *while (ros::ok ( ))*
  - Wait command *rate.sleep()*

- To compile make sure you add the dependency to the msg
  - in *find_package* in CMakeLists.txt
  - in package.xml *build_depend* and *run_depend* sections

C++ code subscriber

- Subscriber does not know when messages arrive
- Code corresponding to incoming messages should go into a callback function

```
void function_name(const package_name::type_name &msg) {
. . .
}

void chatterCallback(const std_msgs::String::ConstPtr& msg) {
ROS_INFO("I heard: [%s]", msg->data.c_str());
 }
```

- Create subscriber object

```
ros::Subscriber sub = node_handle.subscribe(topic_name, queue_size, pointer_to_callback_function);
```

- Let ROS do its work

ros::spinOnce()
Or
ros::spin();

Outline

- Why ROS and what is it?
- Communication protocol
- How to get started
- Sample C++ code
- Sample python code
- Launch Files
- Parameters
- Services examples
- Actions examples
- Useful commands and libraries

Python code publisher

- Python code goes into a scripts folder

```
#!/usr/bin/env python
# license removed for brevity
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
```

Required for standard ROS classes

Adds a message type

Initialize publisher object

Initializes ROS client library

Node frequency

Node frequency

ROS loop

Catch errors

Python code subscriber

- Python code goes into a scripts folder

```
#!/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():

    # 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()
```

Callback function

Initializes ROS client library

Create subscriber object

Let ROS take over

Python code

- Remember to do *chmod +x* command for the scripts.
- To compile python code add into CMakeLists.txt

```
catkin_install_python(PROGRAMS scripts/python_file_name.py
DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

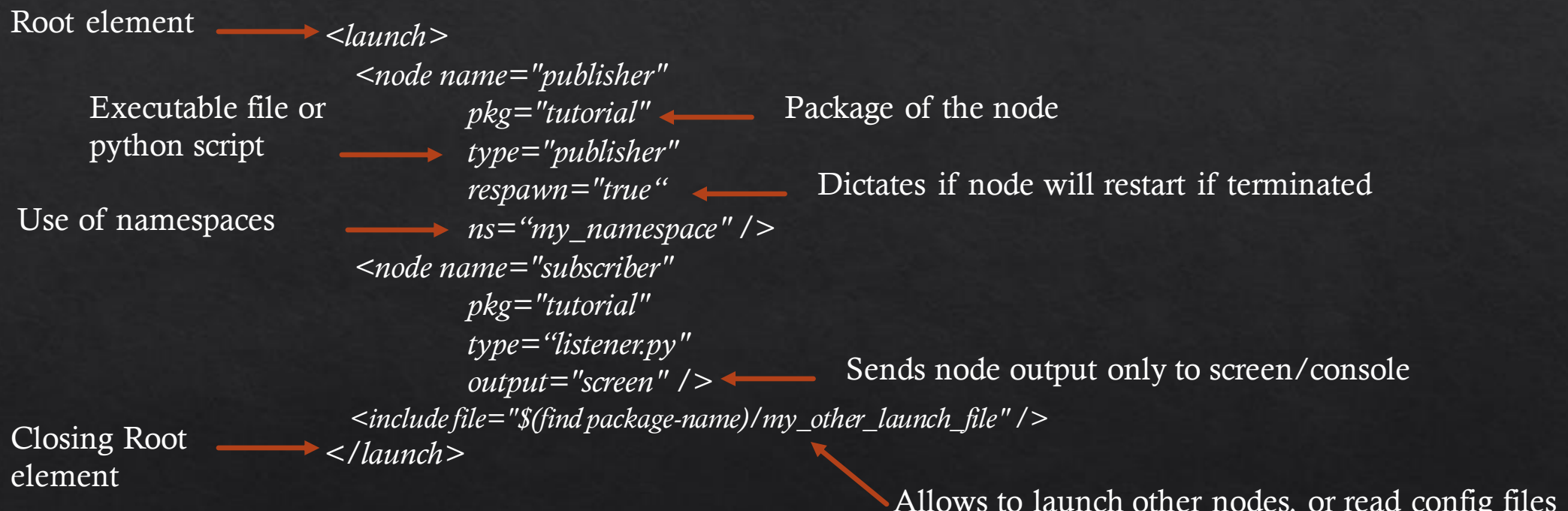
- Do *catkin build*

Outline

- Why ROS and what is it?
- Communication protocol
- How to get started
- Sample C++ code
- Sample python code
- Launch Files
- Parameters
- Services examples
- Actions examples
- Useful commands and libraries

Launch files

- A launch file is a powerful tool for us to launch multiple files in one single file written in xml format



Launch files

- Nodes will be launched almost simultaneously.
- The command is *roslaunch package-name launch-file-name*
- It checks if there is a **roscore** running, it starts it otherwise
- Using the *$(find package-name)* is a powerful way to launch nodes in another package
- Rqt_launchtree is a good tool to help navigate and understand what a launch file is doing.

Outline

- Why ROS and what is it?
- Communication protocol
- How to get started
- Sample C++ code
- Sample python code
- Launch Files
- Parameters
- Services examples
- Actions examples
- Useful commands and libraries

Parameters

- Parameters are values that do not change often
- Can be set in multiple ways
- From command line:
  - To check the available parameters do *rosparam list*
  - To get a value *rosparam get parameter_name*
  - To set a value *rosparam set parameter_name parmeter_value*
  - To send parameters of a namespace to a YAML file by *rosparam dump filename namespace*
  - To load from a YAML file *rosparam load filename namespace*

Parameters

- In C++ interface:
  - *void ros::param::set(parameter_name, input_value);*
  - *bool ros::param::get(parameter_name, output_value);*
- *In python interface :*
  - *rospy.get_param(param_name)*
  - *rospy.set_param(param_name, param_value)*
- In launch files you set it as a param element:
  - *<param name+"param_name" value="param_value" />*
- You can also load from a YAML file from launch files:
  - *<rosparam command="load" file="path_to_param_file" />*

Outline

- Why ROS and what is it?
- Communication protocol
- How to get started
- Sample C++ code
- Sample python code
- Launch Files
- Parameters
- Services examples
- Actions examples
- Useful commands and libraries

Service code client example

```
#include<ros/ros.h>

#include<turtlesim/Spawn.h>

int main(int argc,char argv){
ros::init(argc,argv,"spawn_turtle");
ros::NodeHandle nh;

ros::ServiceClient spawnClient=nh.serviceClient<turtlesim::Spawn>("spawn");

turtlesim::Spawn::Request req;
turtlesim::Spawn::Response resp;

req.x=2;
req.y=3;
req.theta=M_PI/2;
req.name="Leo";

bool success=spawnClient.call(req,resp);

if(success){
ROS_INFO_STREAM("Spawned a turtle named "<<resp.name);
}else{
ROS_ERROR_STREAM("Failed to spawn.");
}
```

srv class for the service

Create client object

Create the request and response objects.

Fill the request data

Call the service

Check for success

Use the response

Service code server example

srv class for the service

```
#include<ros/ros.h>
#include<std_srvs/Empty.h>
#include<geometry_msgs/Twist.h>

bool forward=true;
bool toggleForward(
                std_srvs::Empty::Request &req,
                std_srvs::Empty::Response &resp

){
                forward=!forward;
                ROS_INFO_STREAM("Now sending "<<(forward?
                 "forward":"rotate")<<" commands.");
                return true;
}
```

Create the request and response objects.

Call back function

Service code server example

```
int main(int argc,char argv){
        ros::init(argc,argv,"pubvel_toggle");
        ros::NodeHandle nh;

        ros::ServiceServer server=nh.advertiseService(
          "toggle_forward",&toggleForward);

        ros::Publisher pub=nh.advertise<geometry_msgs::Twist>(
          "turtle1/cmd_vel",1000);
        ros::Raterate(2);
        while(ros::ok()){
                geometry_msgs::Twistmsg;
                msg.linear.x=forward?1.0:0.0;
                msg.angular.z=forward?0.0:1.0;
                pub.publish(msg);
                ros::spinOnce();
                rate.sleep();
        }
}
```

Create client object

Service name

Pointer to Callback function

ROS loop

Let's ROS process things, this will call the callback if needed

Is this program efficient?

Outline

- Why ROS and what is it?
- Communication protocol
- How to get started
- Sample C++ code
- Sample python code
- Launch Files
- Parameters
- Services examples
- Actions examples
- Useful commands and libraries

Action message example

- Similar to srv, action messages use --- to divide sections into: goal, result, feedback.
- Example:

```
#goal definition
int32 order
---
#result definition
Int32[ ] sequence
---
#feedback
Int32[ ] sequence
```

- Need to add *actionlib* and *actionlib_msgs* to both package.xml and CMakeLists.txt
- When compiling package with action messages, headers and other msgs will be created automatically

Action server example

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

import rospy

import actionlib

import actionlib_tutorials.msg

class FibonacciAction(object):
    _feedback = actionlib_tutorials.msg.FibonacciFeedback()
    _result = actionlib_tutorials.msg.FibonacciResult()

    def __init__(self, name):
        self._action_name = name
        self._as = actionlib.SimpleActionServer(self._action_name,
actionlib_tutorials.msg.FibonacciAction, execute_cb=self.execute_cb, auto_start = False)
        self._as.start()
```

required

Add the action msgs package

Action class

prepare action messages

Create the action server variable

Function that will be exacuted when called

Action server example

Start definition of the actual callback function

```
def execute_cb(self, goal):
        # helper variables
        r = rospy.Rate(1)
        success = True
        self._feedback.sequence = []
        self._feedback.sequence.append(0)
        self._feedback.sequence.append(1)

        rospy.loginfo('%s: Executing, creating fibonacci sequence of order %i with seeds %i, %i' %
(self._action_name, goal.order, self._feedback.sequence[0], self._feedback.sequence[1]))
```

Add initial values of sequence to feedback

Check if termination has been requested

```
        for i in range(1, goal.order):
            if self._as.is_preempt_requested():
                rospy.loginfo('%s: Preempted' % self._action_name)
                self._as.set_preempted()
                success = False
                break
            self._feedback.sequence.append(self._feedback.sequence[i] + self._feedback.sequence[i-
1])
            self._as.publish_feedback(self._feedback)

            r.sleep()
```

Public the feedback message

Action server example

Fill result message

```
if success:
    self._result.sequence = self._feedback.sequence
    rospy.loginfo('%s: Succeeded' % self._action_name)
    self._as.set_succeeded(self._result)
```

Set status as success

Main function

```
if __name__ == '__main__':
    rospy.init_node('fibonacci')
    server = FibonacciAction(rospy.get_name())
    rospy.spin()
```

Create action server

Action client example

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

from __future__ import print_function
import rospy

import actionlib

import actionlib_tutorials.msg

def fibonacci_client():
    client = actionlib.SimpleActionClient('fibonacci', actionlib_tutorials.msg.FibonacciAction)

    client.wait_for_server()

    goal = actionlib_tutorials.msg.FibonacciGoal(order=20)

    client.send_goal(goal)

    client.wait_for_result()

    return client.get_result()  # A FibonacciResult
```

required

Add the action msgs package

Client object

Type of action message

Wait for server to start

Creates a goal

Send goal to server

Wait for result (optional)

Return result of action

Action client example

Action client

```
if __name__ == '__main__':
    try:
        rospy.init_node('fibonacci_client_py')
        result = fibonacci_client()
        print("Result:", ', '.join([str(n) for n in result.sequence]))
    except rospy.ROSInterruptException:
        print("program interrupted before completion",
file=sys.stderr)
```

Do something with the result

Outline

- Why ROS and what is it?
- Communication protocol
- How to get started
- Sample C++ code
- Sample python code
- Launch Files
- Parameters
- Services examples
- Actions examples
- Useful commands and libraries

Rosbash

- Package that allows certain useful commands.
- Allows autocompletion of certain commands.
- *roscd*: allows you to cd directly to a package
- *rosed*: allows to directly edit a file within a package by package name

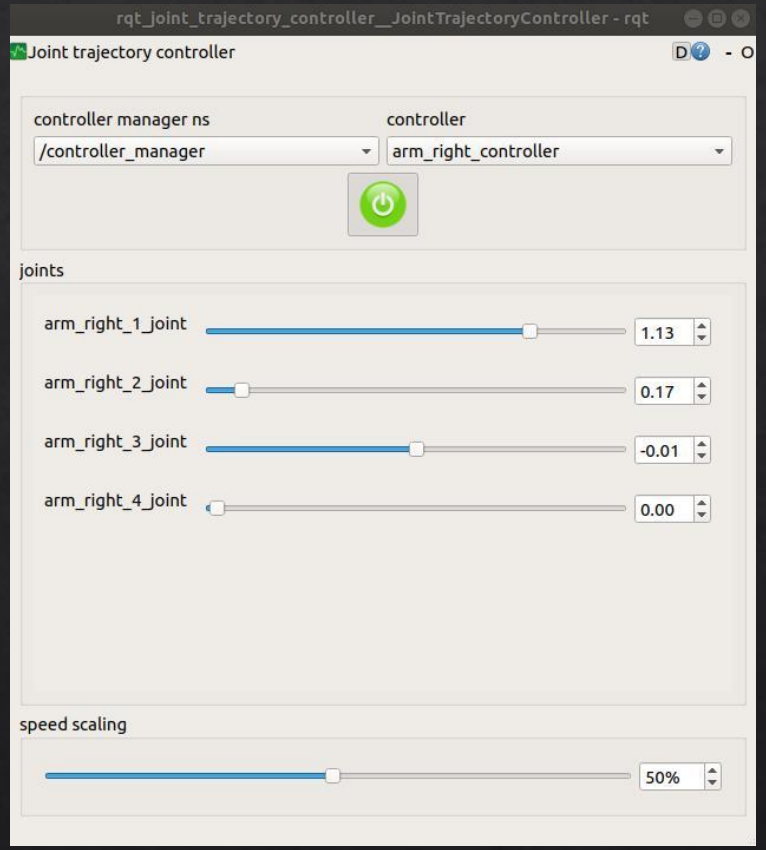
Rosbag

- Allows saving of information published in topics.
- *rosbag record*: allows to record topics
- Example
  - *rosbag record –o my_rosbag –d=15 my_topic_name1 my_topic_name2*

- *rosbag info :* human-readable summary of the contents of the bag
- *rosbag play* : allows to replay the contents of a bag in a time-synchronized fashion

Rqt

- Qt-based framework for GUI development for ROS
- Running *rqt* alone opens a gui from where you can load plugins
- *rqt_bag* : viewing data in ROS bag files
- *rqt_graph* : interactive graph of ROS nodes and topics
- *rqt_plot* : plots numerical data of a topic over time
- *rosrun rqt_joint_trajectory_controller rqt_joint_trajectory_controller :* graphical interface to move joints



Transform server

- Management of coordinate frames
- Answers questions like:
  - Where was the head frame relative to the world frame, 5 seconds ago?
  - What is the pose of the object in my gripper relative to my base?

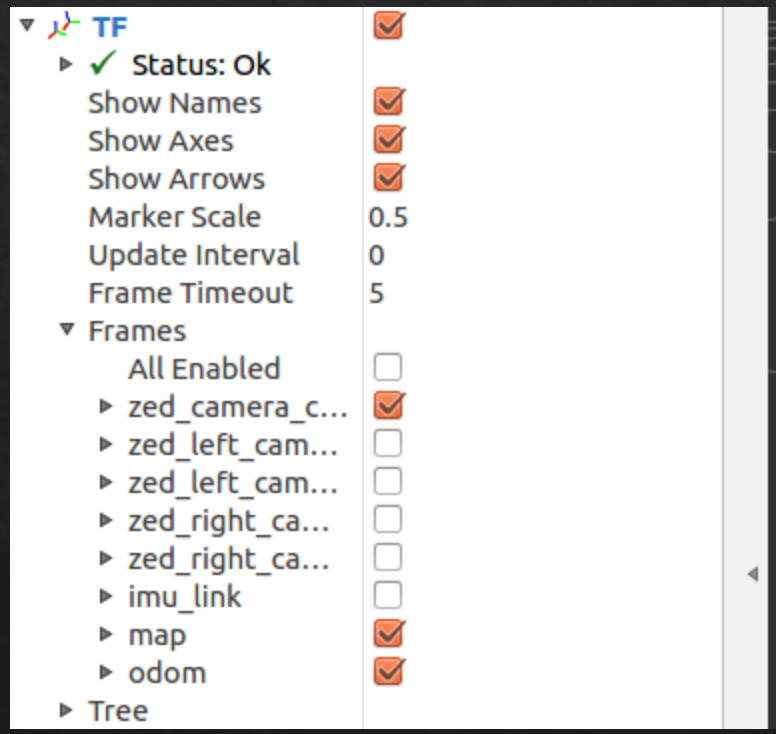
- *rqt_tf_tree* : allows to visualize a tree of frames broadcasted over ROS

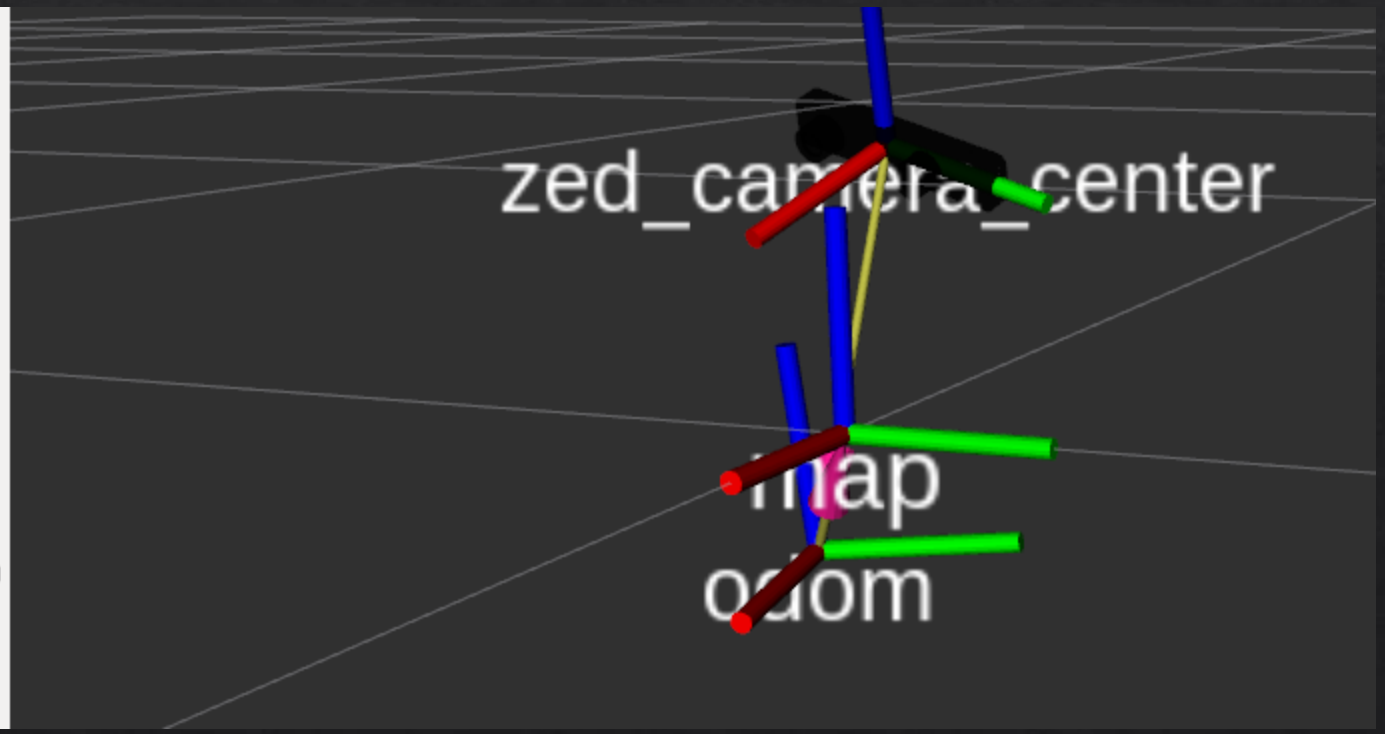
- *rosrun tf tf_echo [reference_frame] [target_frame] :* reports transform between frames

```
At time 1416409795.450
- Translation: [0.000, 0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.914, 0.405]
            in RPY [0.000, -0.000, 2.308]
```

RVIZ

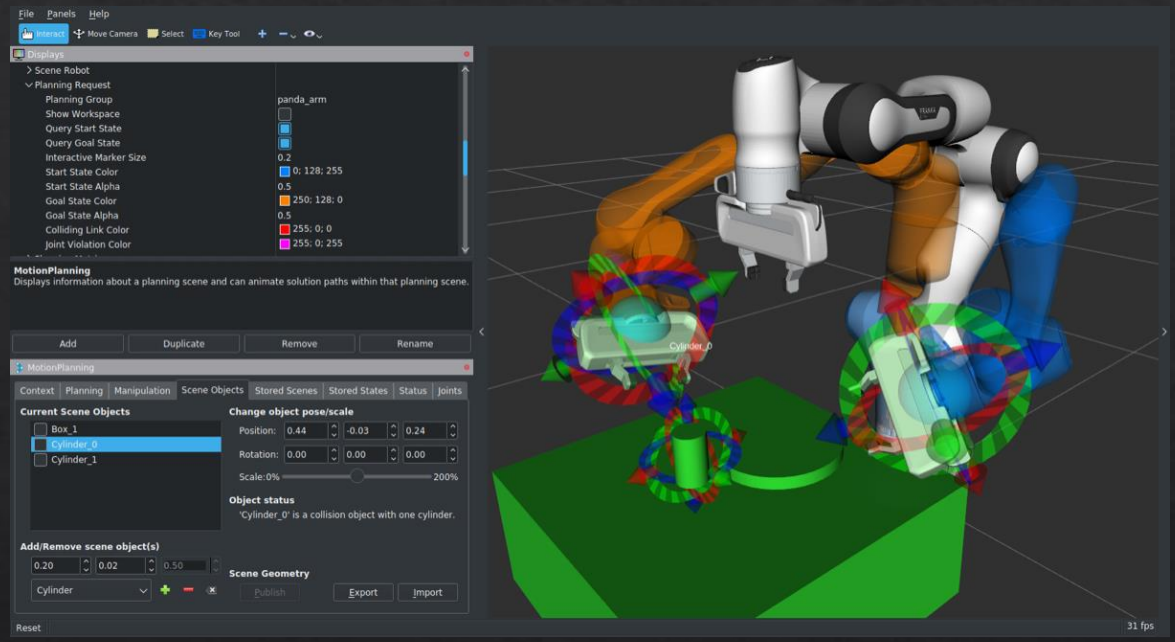
- Is a 3d visualization tool.
- It allows for configuration of a visualization setup, also saving and loading of the configuration.





Move-it

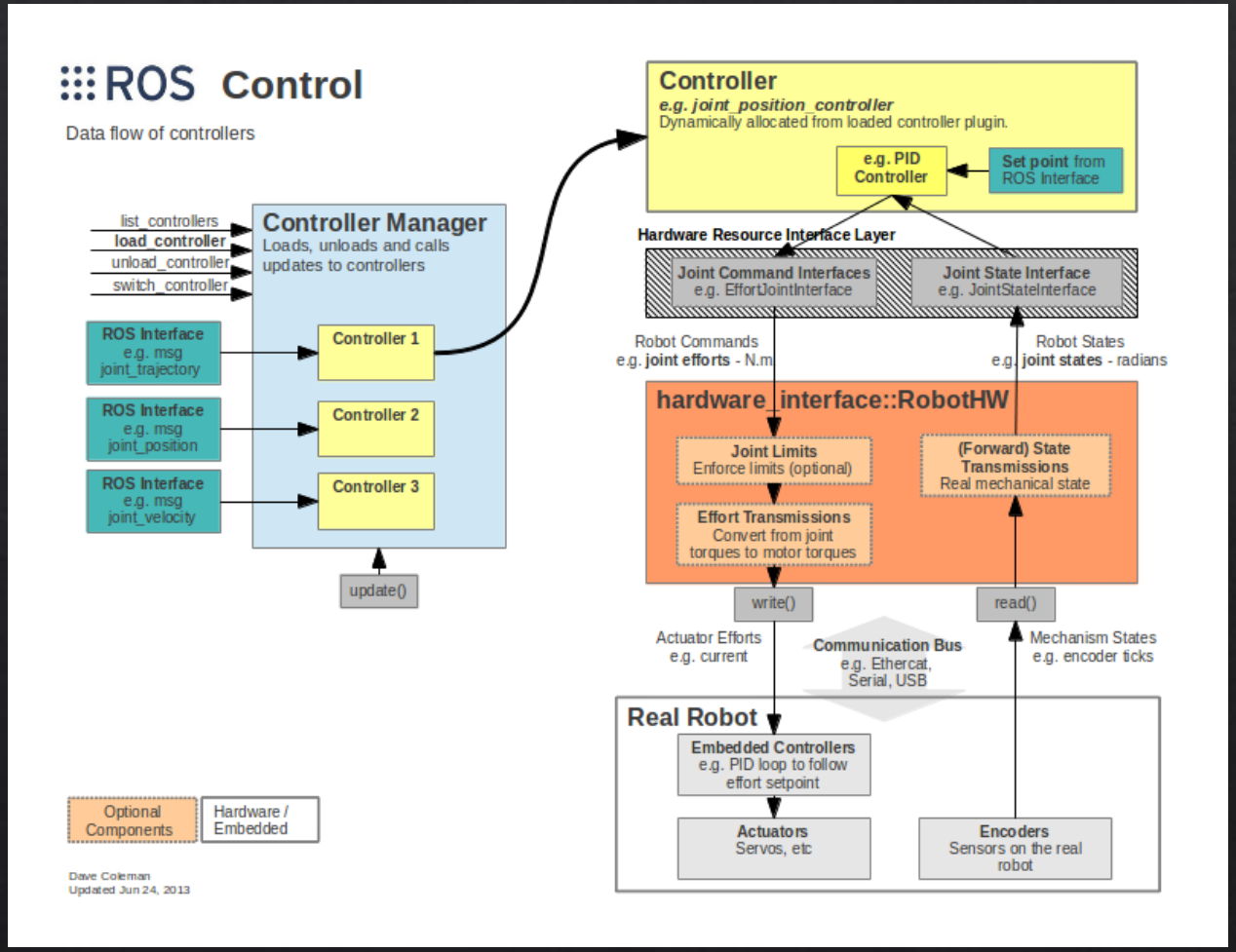
- Created originally for manipulation
- It has algorithms for:
  - Motion planning
  - manipulation
  - 3d perception
  - Kinematics
  - Control
  - Navigation



- Allows for a relatively easy interaction with the robot

ROS control

- Capability to implement and manage robot controllers
- Motivation: lack of real-time communication layer on ROS
- Allows for abstraction between hardware and control
- Controller Manager allows: load, unload and switch between controllers



ROS control

- Available controllers:
  - Effort_controller : to control using position, velocity or effort (torque/current)
  - Position_controllers
  - Velocity_controllers
  - Joint_trajectory_controllers: set entire trajectory
  - Joint_state_controller : define publish joint states
- Usually defined with yaml files in the config folder

Gazebo

- Is a robotics simulator.
- It interfaces with ROS
- Allows to test algorithms in a safe environment

- Remark: Simulation results do not guarantee same results on th real robot but they might be close

Other useful material

- ROS main page (http://wiki.ros.org/)
- A gentle introduction to ROS (https://www.cse.sc.edu/~jokane/agitr/)
- The construct (https://www.theconstructsim.com/)
- Introduction to ROS online course (https://sir.upc.edu/projects/rostutorials/index.html)
- Git repo with tutorials code by Jan Rosell (https://gitioc.upc.edu/rostutorials)
- Blog post on ways to get started with ROS (https://www.theconstructsim.com/ros-for-beginners-how-to-learn-ros/) . Note they promote their product. Their product is nice but not free.
- Robitcs Back-End tutorials (https://roboticsbackend.com/category/ros/page/5/)
- Example of action lib server and client (https://github.com/MariaC27/actionlib_tutorials)

Other useful material

- Packages for action lib tutorial (http://library.isr.ist.utl.pt/docs/roswiki/actionlib(2f)Tutorials.html)
- Ros controllers repo (https://github.com/ros-controls/ros_controllers)
- Transform server tutorials (http://wiki.ros.org/tf/Tutorials)
- Rviz tutorials (http://wiki.ros.org/rviz/Tutorials)
- Moveit documentation (https://moveit.picknik.ai/galactic/index.html)
- Gazebo tutorials (http://gazebosim.org/tutorials)
- Other robot's urdfs (https://wiki.ros.org/urdf/Examples)

Thank you for your attention