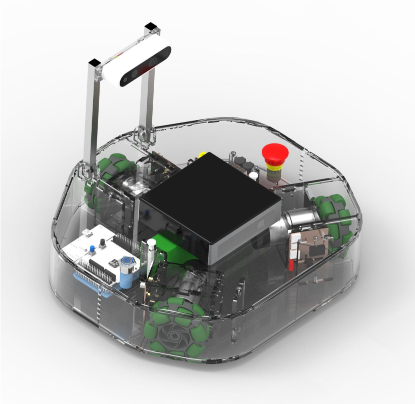
ROS programming, testing, continuous integration

ROS Training for Industry: Day 3



Robert Valner

18.09.2019

Tartu, Estonia

Agenda: Day 3 (18.09)

- 09:15 Hardware & drivers
- 10:15 Coffee Break
- 10:30 Workshop: Implementing ROS driver for Custom Hardware
  - Write driver for Arduino Sonar
  - Publish sonar range, IMU orientation, and visualize in RViz
- 12:00 Lunch Break
- 13:00 ROS Testing Tools & Continuous Integration
- 14:30 Coffee Break
- 14:45 Workshop
  - write tests and documentation for the ongoing package
  - 17:00 End of Day 3

Hardware & drivers

Overview

- Defining the Problem
- Common ROS Tools
  - ROS Control
  - Nodelets
  - ROSSerial
- Third party libraries

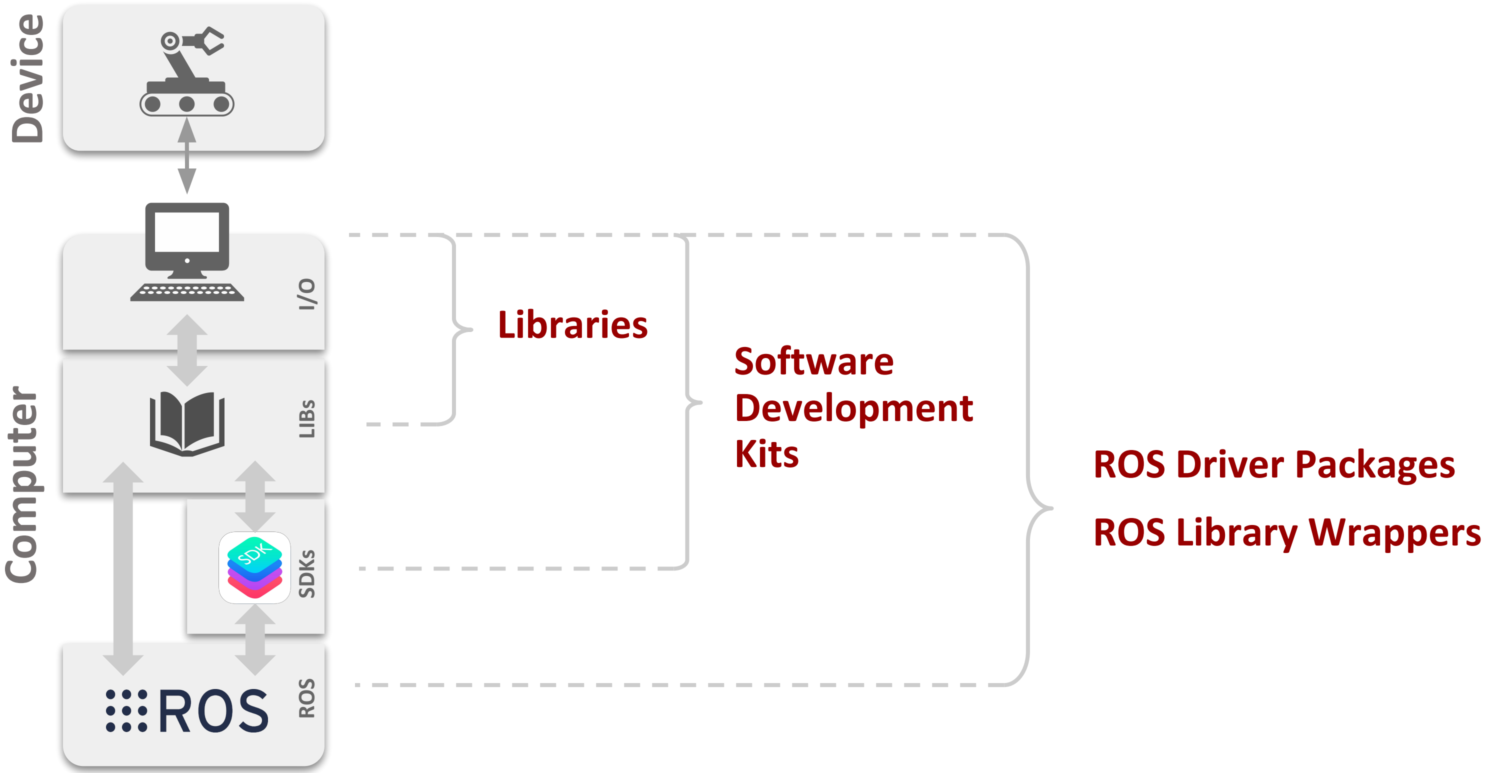
Defining the Problem

Defining the Problem

- **Type of device**
  - Sensor
  - Actuator
  - Combined
- **Type of data**
  - Protocol
  - Speed
  - Amount

- **Available tools**
  - Driver packages
  - Control packages
  - Peripheral libraries
  - Computation tools

Defining the Problem - don't reinvent the wheel



Libraries

Software Development Kits

ROS Driver Packages

ROS Library Wrappers

Defining the Problem - Architecture

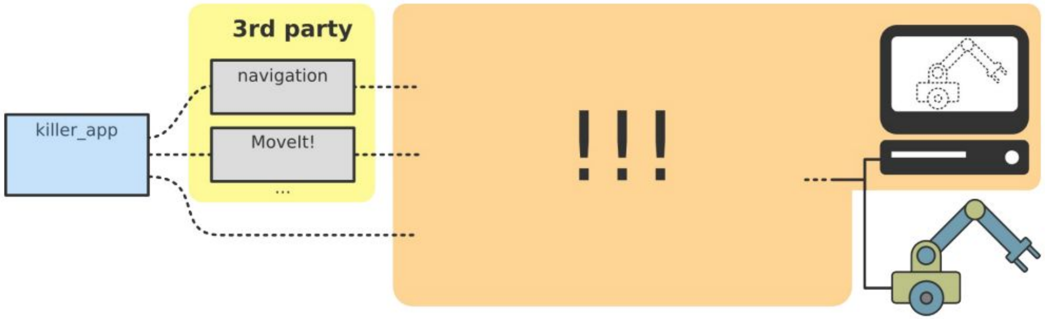
- **Asynchrony**
  - Loops → *produce/process data*
  - Callbacks → receive/*handle data*
  - Interrupts
- **Threading**
  - Multi threaded → *efficiency*
  - Multi process → *modularity*

Common ROS Tools

Standard solutions for standard problems

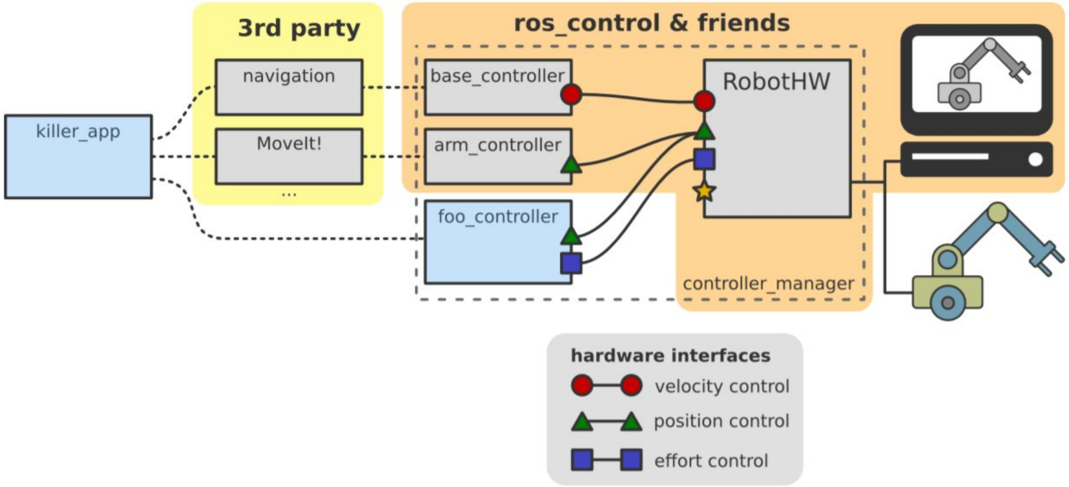
Common ROS Tools - ROS Control

- Standardizes controller infrastructure
- Offers base implementations for common controller types
  - joint _state_controller
  - diff_drive_controller



Common ROS Tools - ROS Control

- Existing controllers
- Custom controllers
- Custom hardware backend
- Abstract interfaces



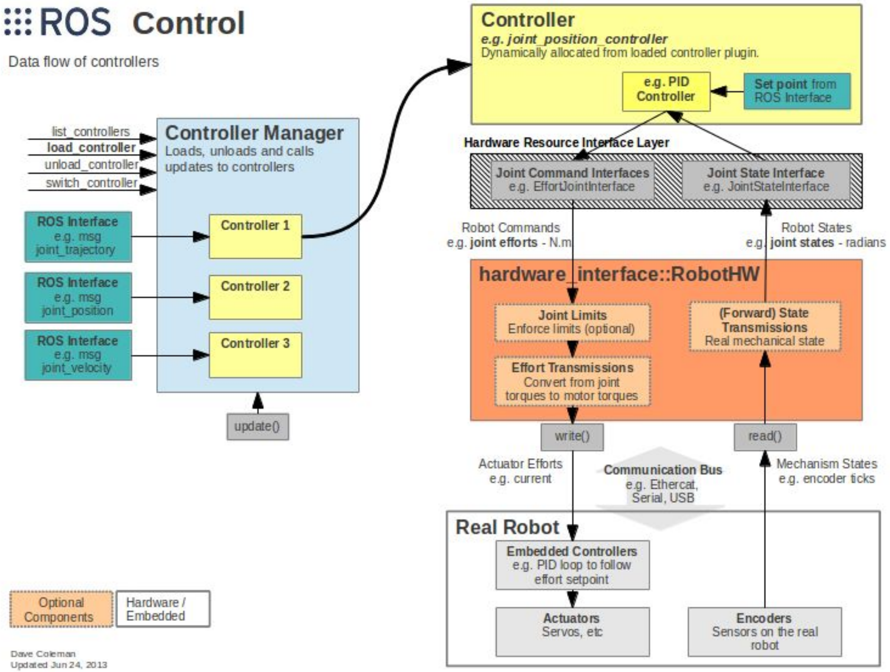
Common ROS Tools - ROS Control

ros_control handles two things from that process:

- receiving the goals (effort, position, velocity, trajectory, etc.)
- running the PID controllers

ros_control doesn't know or handle:

- implementing hardware control (sending current to motors)
- reading hardware state

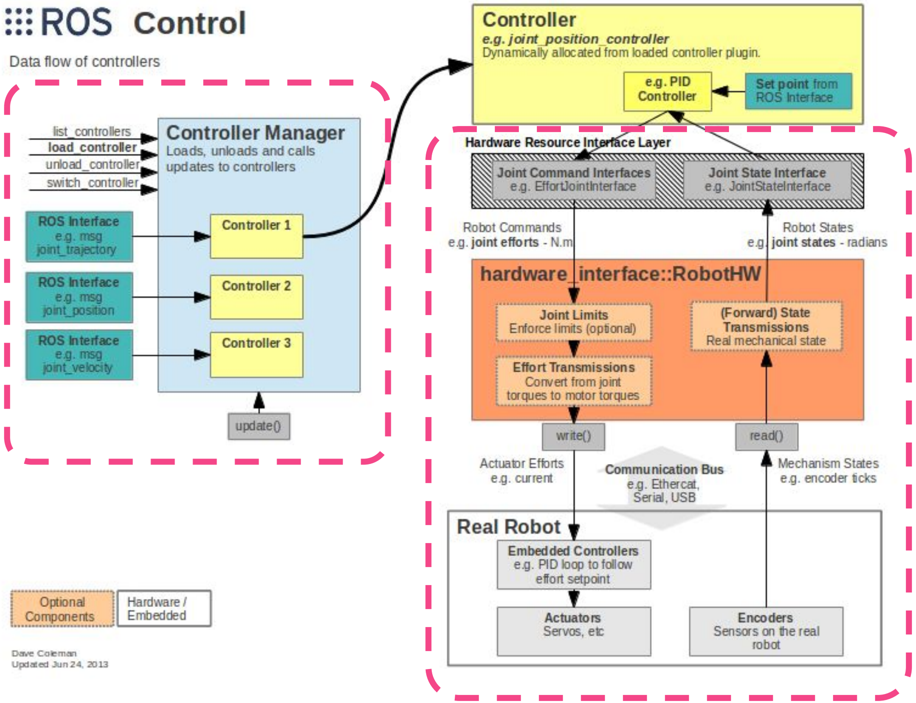


ROS Control
Data flow of controllers
list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()
Controller
e.g. joint_position_controller
Dynamically allocated from loaded controller plugin.
e.g. PID Controller
Set point from ROS Interface
Hardware Resource Interface Layer
Joint Command Interfaces e.g. EffortJointInterface
Joint State Interface e.g. JointStateInterface
Robot Commands e.g. joint efforts - N.m
Robot States e.g. joint states - radians
hardware_interface::RobotHW
Joint Limits Enforce limits (optional)
(Forward) State Transmissions Real mechanical state
Effort Transmissions Convert from joint torques to motor torques
write()
read()
Actuator Efforts e.g. current
Communication Bus e.g. Ethercat, Serial, USB
Mechanism States e.g. encoder ticks
Real Robot
Embedded Controllers e.g. PID loop to follow effort setpoint
Actuators Servos, etc
Encoders Sensors on the real robot
Optional Components
Hardware / Embedded

Dave Coleman
Updated Jun 24, 2013

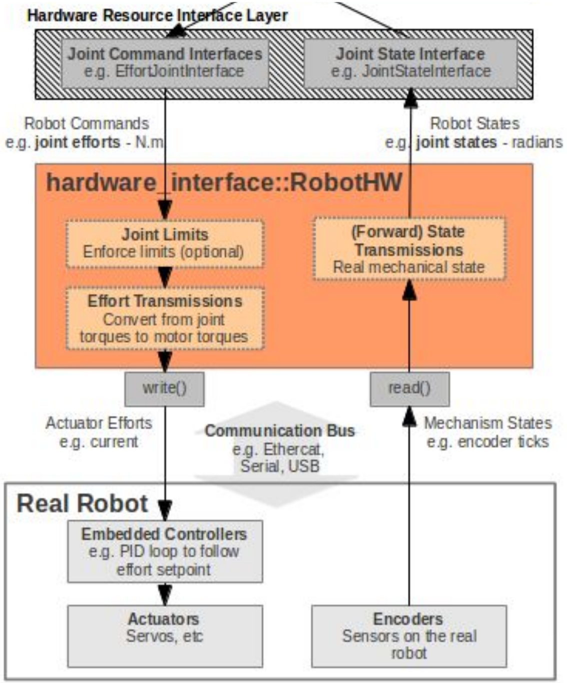
Common ROS Tools - ROS Control

Configured


ROS Control
Data flow of controllers
list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()
Controller
e.g. joint_position_controller
Dynamically allocated from loaded controller plugin.
e.g. PID Controller
Set point from ROS Interface
Hardware Resource Interface Layer
Joint Command Interfaces e.g. EffortJointInterface
Joint State Interface e.g. JointStateInterface
Robot Commands e.g. joint efforts - N.m
Robot States e.g. joint states - radians
hardware_interface::RobotHW
Joint Limits Enforce limits (optional)
Effort Transmissions Convert from joint torques to motor torques
(Forward) State Transmissions Real mechanical state
write()
read()
Actuator Efforts e.g. current
Communication Bus e.g. Ethercat, Serial, USB
Mechanism States e.g. encoder ticks
Real Robot
Embedded Controllers e.g. PID loop to follow effort setpoint
Actuators Servos, etc
Encoders Sensors on the real robot
Optional Components
Hardware / Embedded
Dave Coleman
Updated Jun 24, 2013

Implemented/ coded

Common ROS Tools - ROS Control



Hardware Resource Interface Layer

Joint Command Interfaces
e.g. EffortJointInterface

Joint State Interface
e.g. JointStateInterface

Robot Commands
e.g. **joint efforts** - N.m

Robot States
e.g. **joint states** - radians

hardware_interface::RobotHW

Joint Limits
Enforce limits (optional)

(Forward) State Transmissions
Real mechanical state

Effort Transmissions
Convert from joint torques to motor torques

write()

read()

Actuator Efforts
e.g. current

Communication Bus
e.g. Ethercat, Serial, USB

Mechanism States
e.g. encoder ticks

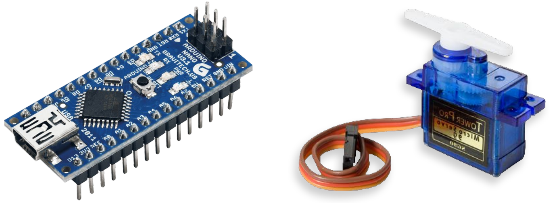
Real Robot

Embedded Controllers
e.g. PID loop to follow effort setpoint

Actuators
Servos, etc

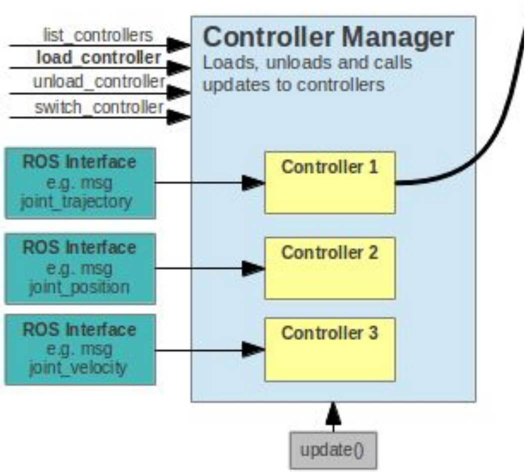
Encoders
Sensors on the real robot

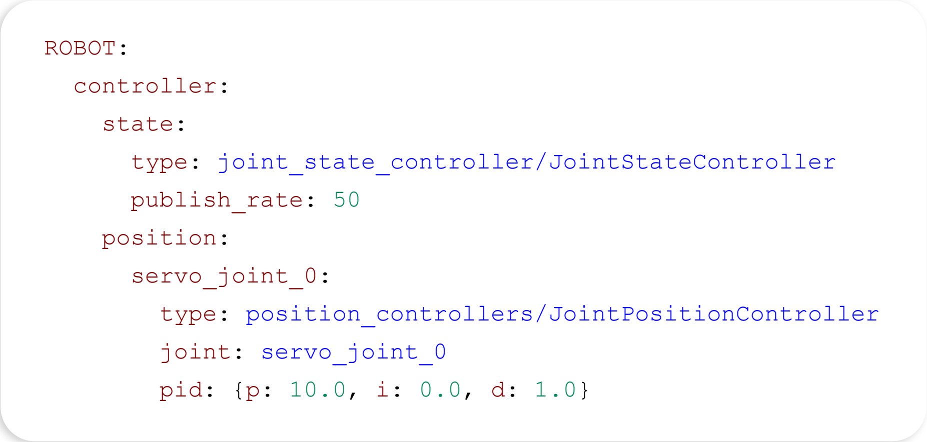
```
class ServoHardwareInterface: public hardware_interface::RobotHW
{
Public:
  JointStateInterface joint_state_interface_;
  PositionJointInterface joint_position_interface_;
  ServoHardwareInterface();
  void init();
  void update(const ros::TimerEvent& e);
  void read();
  void write(ros::Duration elapsed_time);
};
```



Common ROS Tools - ROS Control

controllers YAML

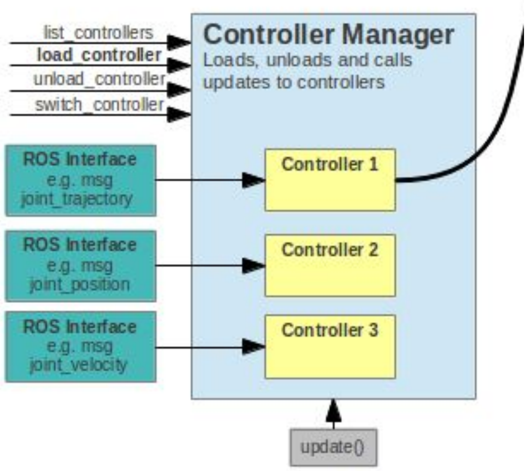

list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()



```
ROBOT:
  controller:
    state:
      type: joint_state_controller/JointStateController
      publish_rate: 50
    position:
      servo_joint_0:
        type: position_controllers/JointPositionController
        joint: servo_joint_0
        pid: {p: 10.0, i: 0.0, d: 1.0}
```

Common ROS Tools - ROS Control

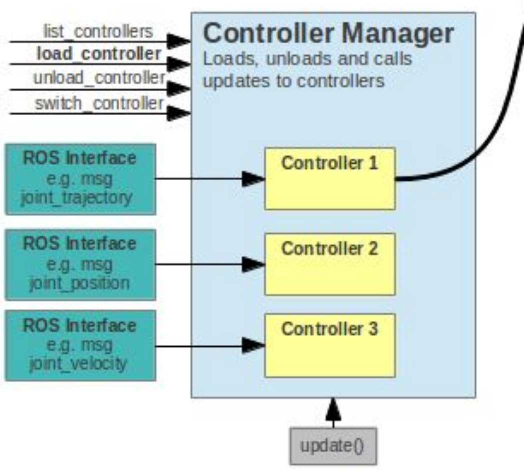
hardware YAML


list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()

```
ROBOT:
  hardware_interface:
     loop_hz: 50 # hz
     joints:
       - servo_joint_0
```

Common ROS Tools - ROS Control

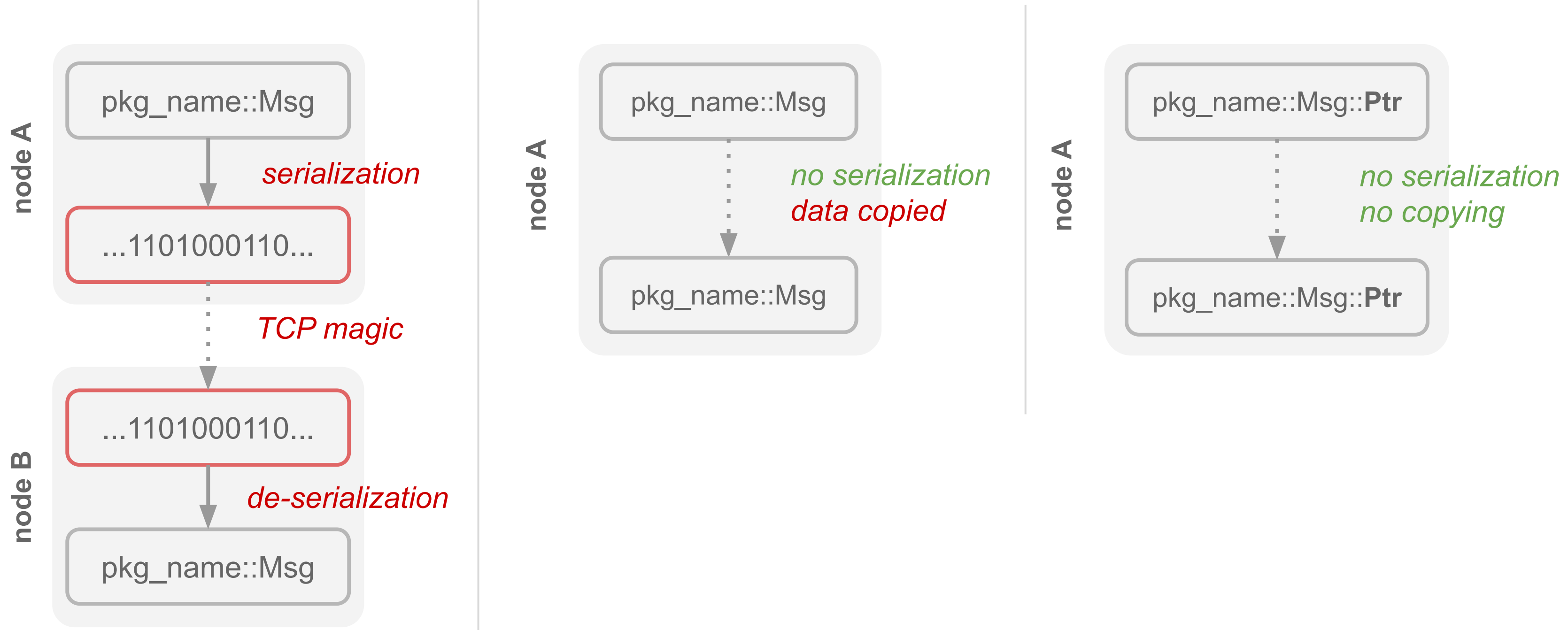
joint_limits YAML


list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()

```yaml
joint_limits:
  servo_joint_0:
    has_position_limits: true
    min_position: 0.02
    max_position: 3.14
    has_velocity_limits: true
    max_velocity: 2.0
    ...
```

Common ROS Tools - Nodelets

First … a bit about **sending messages in ROS**



node A

pkg_name::Msg

serialization

...1101000110...

TCP magic

node B

...1101000110...

de-serialization

pkg_name::Msg

node A

pkg_name::Msg

no serialization

data copied

pkg_name::Msg

node A

pkg_name::Msg::**Ptr**

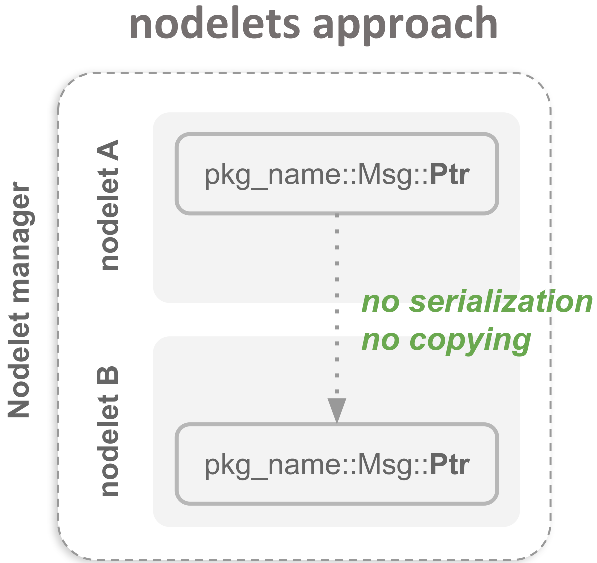
no serialization

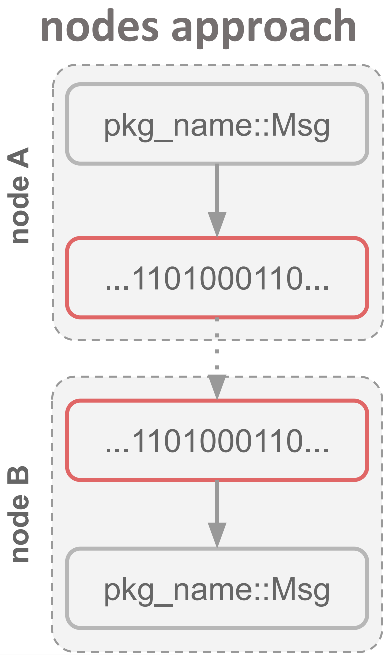
no copying

pkg_name::Msg::**Ptr**

Common ROS Tools - Nodelets

Nodelets help to maintain **modularity without losing efficiency**





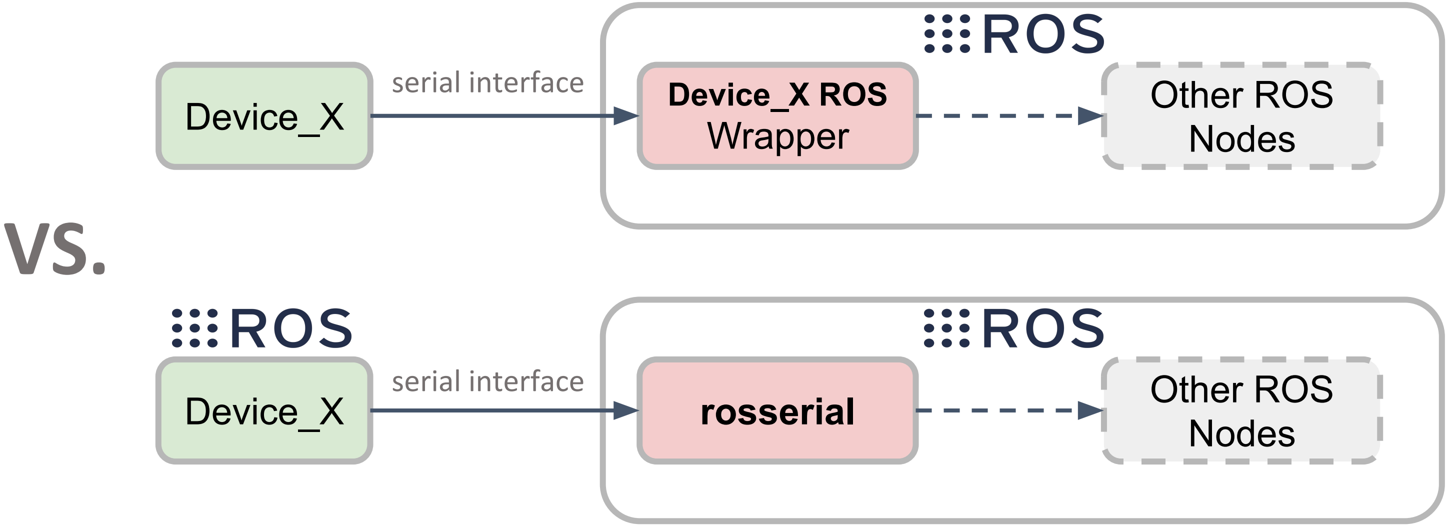
Common ROS Tools - Nodelets

What is it good for?

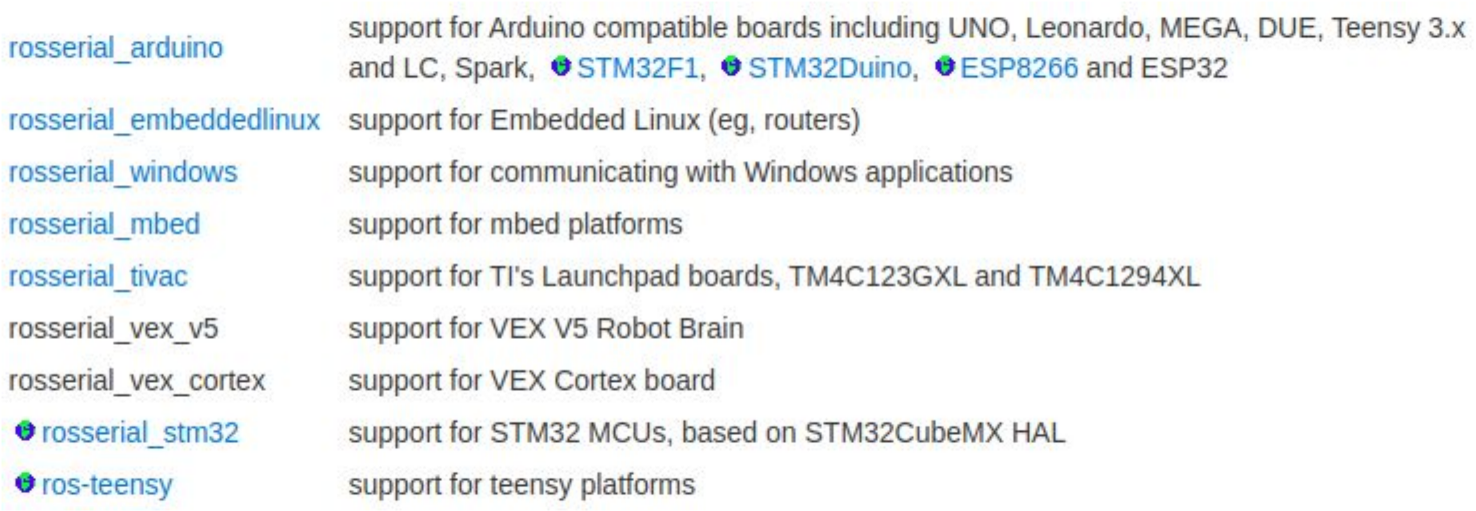
- Image processing pipelines
- Point cloud processing pipelines

Rosserial - Concept

Wrapperless (almost) ROS interfacing



Rosserial - Platforms



rosserial_arduino	support for Arduino compatible boards including UNO, Leonardo, MEGA, DUE, Teensy 3.x and LC, Spark, STM32F1, STM32Duino, ESP8266 and ESP32
rosserial_embeddedlinux	support for Embedded Linux (eg, routers)
rosserial_windows	support for communicating with Windows applications
rosserial_mbed	support for mbed platforms
rosserial_tivac	support for TI's Launchpad boards, TM4C123GXL and TM4C1294XL
rosserial_vex_v5	support for VEX V5 Robot Brain
rosserial_vex_cortex	support for VEX Cortex board
rosserial_stm32	support for STM32 MCUs, based on STM32CubeMX HAL
ros-teensy	support for teensy platforms

Third Party Libraries

How to decently integrate external libraries

Third Party Libraries

Library coming:

- … in its own **ROS package** (simplest case)
  - from package manager
  - from source
- … as a **non-ROS package**
- … **from non-ROS source** (trickier case)

Third Party Libraries - as bin ROS pkg

- sudo apt install ros-<distro>-**name-of-lib-package**
- In **CMakeLists.txt**
  - `find_package( catkin REQUIRED COMPONENTS <name_of_lib_package> …`
  - `CATKIN_DEPENDS <name_of_lib_package> …`
  - `include_directories( ${catkin_INCLUDE_DIRS} )`
  - `target_link_libraries( my_node ${catkin_LIBRARIES} )`
- In **package.xml**
  - `<build_depend>name_of_lib_package</build_depend>`
  - `<exec_depend>name_of_lib_package</exec_depend>`
- In **my_node.cpp**
  - `#include "name_of_lib_package/some_header_file.h"`

Third Party Libraries - as src ROS pkg

- git clone https://github.com/xyz/**name-of-lib-package.git**
- In **CMakeLists.txt**
  - find_package(catkin REQUIRED COMPONENTS **<name_of_lib_package>** …
  - CATKIN_DEPENDS **<name_of_lib_package>** …
  - include_directories(**${catkin_INCLUDE_DIRS}**)
  - target_link_libraries(my_node **${catkin_LIBRARIES}**)
- In **package.xml**
  - <build_depend>**name_of_lib_package**</build_depend>
  - <exec_depend>**name_of_lib_package**</exec_depend>
- In **my_node.cpp**
  - #include "**name_of_lib_package**/some_header_file.h"

Third Party Libraries - as non-ROS pkg

- sudo apt install **name-of-lib-package**
  - Pray that it has a **Config.cmake** file
- In **CMakeLists.txt**
  - `find_package( <name_of_lib_package> )`
  - `CATKIN_DEPENDS <name_of_lib_package> …`
  - `include_directories( ${catkin_INCLUDE_DIRS} )`
  - `target_link_libraries( my_node ${catkin_LIBRARIES} )`
- In **package.xml**
  - `<build_depend>name_of_lib_package</build_depend>`
  - `<exec_depend>name_of_lib_package</exec_depend>`
- In **my_node.cpp**
  - `#include "name_of_lib_package/some_header_file.h"`

Third Party Libraries - as non-ROS src

1. Either follow the instructions (if any)

   -
     ```
     $ make && make install
     ```
   - … and then continue as outlined in "**as non-ROS pkg**"

2. Or add as a **git submodule**

   -
     ```
     $ git submodule add https://github.com/xyz/name-of-lib-package.git
     custom libs/name-of-lib-package
     ```

   -
     ```
     include_directories( ${catkin_INCLUDE_DIRS} custom_libs/name-of-lib-package/include)
     ```

   -
     ```
     target_link_libraries(my_node ${catkin_LIBRARIES}
     custom_libs/name-of-lib-package/lib/libblah.so)
     ```

Workshop

Overview

- Documentation via **rosdoc**
  - Doxygen
  - Sphinx
- Unit & Integration tests via **rostest**
  - Googletest
  - rostest features
  - building tests
- Continuous Integration via **Travis CI**
  - Building packages
  - Running tests
  - Running custom build scripts

Documentation

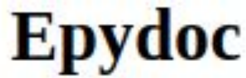
Documentation - rosdoc

A convenience package that

- Combines common documentation tools such as
  - **doxygen**
  - **sphinx**
- Manages documentation build process via **rosdoc.yaml**

Documentation - doc generators

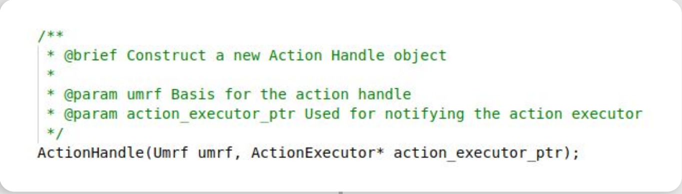
- Scan the workspace for source files
- Look for code comments in a **specific** format
- Extract as much info as indicated in the “**format-file**”
- Generate neat looking documents (html, pdf, latex, …)



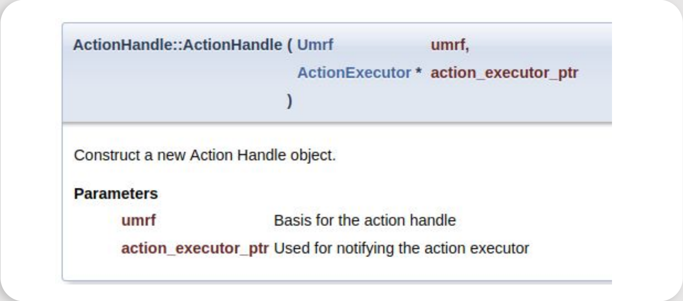
Documentation - doc generators

C++

Doxygen



```
/**
 * @brief Construct a new Action Handle object
 *
 * @param umrf Basis for the action handle
 * @param action_executor_ptr Used for notifying the action executor
 */
ActionHandle(Umrf umrf, ActionExecutor* action_executor_ptr);
```



ActionHandle::ActionHandle (Umrf umrf,
ActionExecutor * action_executor_ptr
)

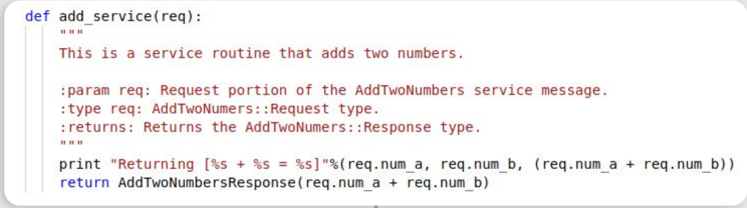
Construct a new Action Handle object.

Parameters

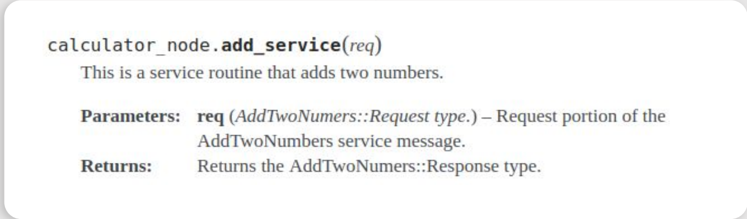
umrf	Basis for the action handle
action_executor_ptr	Used for notifying the action executor

python

SPHINX



```
def add_service(req):
    """
    This is a service routine that adds two numbers.

    :param req: Request portion of the AddTwoNumbers service message.
    :type req: AddTwoNumers::Request type.
    :returns: Returns the AddTwoNumers::Response type.
    """
    print "Returning [%s + %s = %s]"%(req.num_a, req.num_b, (req.num_a + req.num_b))
    return AddTwoNumbersResponse(req.num_a + req.num_b)
```



`calculator_node.`**`add_service`**(*req*)

This is a service routine that adds two numbers.

Parameters: **req** (*AddTwoNumers::Request type.*) – Request portion of the AddTwoNumbers service message.

Returns: Returns the AddTwoNumers::Response type.

Documentation - doxygen

- Document the classes/functions/variables …

```
/**
  * @brief Adds two numbers
  * @param num_a first input number
  * @return double result of adding the two input numbers
 …
```

- Specify the format in the **rosdoc.yaml**

```
- builder: doxygen
  output_dir: doc_cpp
  file_patterns: '*.c *.cpp *.h *.cc *.hh'
   …
```

- Run

```
$ cd <your_package> && rosdoc_lite .
```

- Demo

Documentation - sphinx

- Document the classes/functions/variables …

```
"""
   This is a service routine that adds two numbers.
   :param req: Request portion of the AddTwoNumbers service message.
   :type req: AddTwoNumers::Request type.
   :returns: Returns the AddTwoNumers::Response type.
   ...
```

- Specify the format in the **rosdoc.yaml**

```
- builder: sphinx
  sphinx_root_dir: ./pydoc
  ...
```

- Run

```
$ cd <your_package> && sphinx-quickstart
$ sphinx-apidoc -o ./pydoc ./scripts
```

  - *Modify conf.py file to look for scripts*

```
$ rosdoc_lite .
```

http://wiki.ros.org/Sphinx

Testing

Testing - rostest

rostest allows you to do full integration testing across multiple nodes.

- **Test nodes**
  - **C++:** Gtest
  - **Python:** unittest
- **Reusable test nodes**
  - **hztest:** tests the publishing rate of a node
  - **paramtest:** tests if certain parameters are registered at the Parameter Server
  - **publishtest**: tests if specified topics are published at least once
- **Test launch files**
  - Compatible with *launch* file format
  - “.test” or “.launch” extension

```
<launch>
  <node pkg="mypkg" type="mynode" name="mynode" />
  <test test-name="test_mynode" pkg="mypkg" type="test_mynode" />
</launch>
```

Testing - rostest - gtest test.cpp

Declare the test via “TEST” macro

```
TEST(TestSuite, add_test)
{
  /*
   * Set up your test
   */

  /*
   * Use assertion macros to evaluate your results
   */
  EXPECT EQ( ... ,  ... );// Continues the test
  ASSERT EQ( ... ,  ... );// Terminates the test
  EXPECT TRUE( ... );
  ASSERT_TRUE( ... );
}
```

Run the test from “main”

```
int main(int argc, char **argv)
{
  // Initialize the gtest
  testing::InitGoogleTest(&argc, argv);

  // Run the tests
  return RUN_ALL_TESTS();
}
```

Testing - rostest - gtest test.cpp

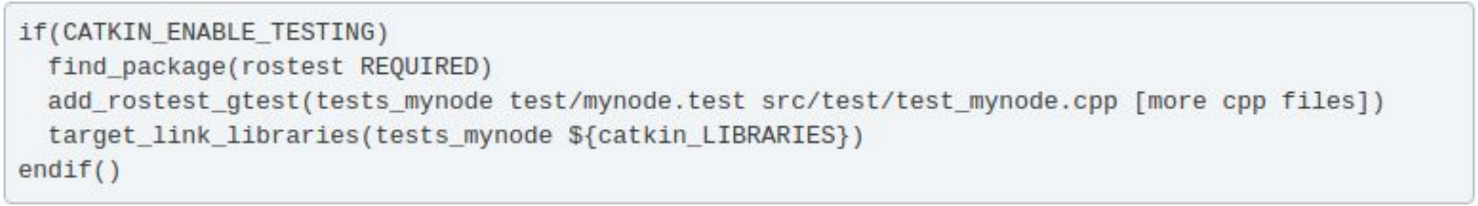
```cpp
/*
 * Tests the adding functionality of the calculator node
 */
TEST(TestSuite, add_test )
{
  ros::NodeHandle nh;
  ros::ServiceClient scl = nh.serviceClient<robert_v_sandbox::AddTwoNumbers>(  "add_two_numbers" );

  // Compose the service message
  robert_v_sandbox::AddTwoNumbers srv_msg;
  srv_msg.request.num_a =  34;
  srv_msg.request.num_b =  5;

  // The result we are expecting to receive from the service call
  double expected_result = srv_msg.request.num_a + srv_msg.request.num_b;

  // Invoke the service call
  if (scl.call(srv_msg))
  {
    // Check if the response is equal to the expected result
    EXPECT_EQ(srv_msg.response.result, expected_result ) << "Expected " << expected_result <<  " but got " << srv_msg.response.result;
  }
  else
  {
    // Assert false if the client was not able to reach the server
    ASSERT_TRUE(false) << "Could not reach the 'add_two_numbers' server"  ;
  }
}
```

Testing - rostest - test launch

```xml
<?xml version="1.0"?>
<launch>
  <node name="calculator_node" pkg="robert_v_sandbox" type="calculator_node"/>
  <test test-name="calculator_test_node" pkg="robert_v_sandbox" type="calculator_test_node"/>
</launch>
```

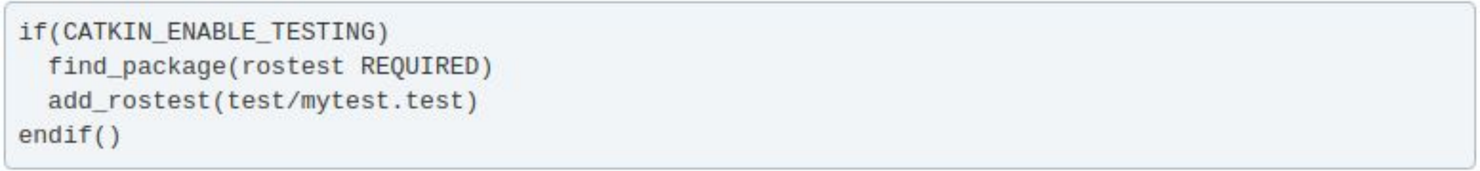
Testing - rostest - CMakeLists.txt

gtest:



```
if(CATKIN_ENABLE_TESTING)
  find_package(rostest REQUIRED)
  add_rostest_gtest(tests_mynode test/mynode.test src/test/test_mynode.cpp [more cpp files])
  target_link_libraries(tests_mynode ${catkin_LIBRARIES})
endif()
```

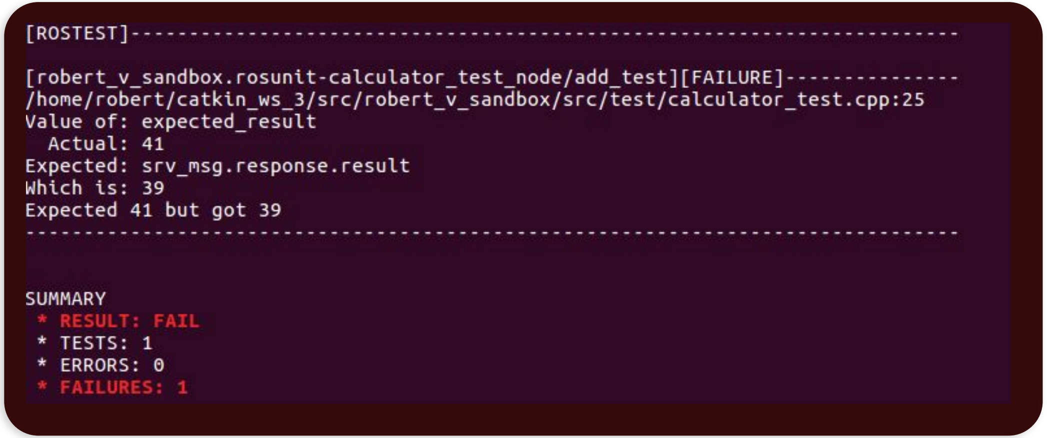
python:



```
if(CATKIN_ENABLE_TESTING)
  find_package(rostest REQUIRED)
  add_rostest(test/mytest.test)
endif()
```

Testing - rostest - run

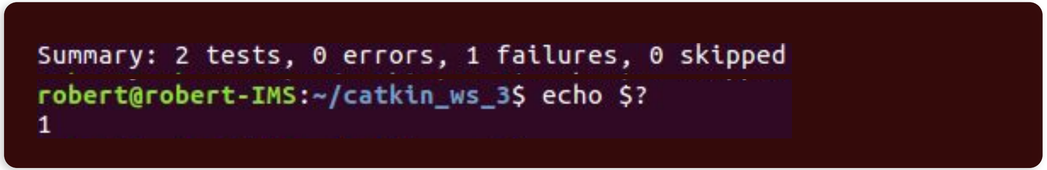
```
$ catkin run_tests
```



```
[ROSTEST]-----------------------------------------------------------------------

[robert_v_sandbox.rosunit-calculator_test_node/add_test][FAILURE]---------------
/home/robert/catkin_ws_3/src/robert_v_sandbox/src/test/calculator_test.cpp:25
Value of: expected_result
  Actual: 41
Expected: srv_msg.response.result
Which is: 39
Expected 41 but got 39
--------------------------------------------------------------------------------


SUMMARY
 * RESULT: FAIL
 * TESTS: 1
 * ERRORS: 0
 * FAILURES: 1
```

Testing - rostest - test return code

```
$ cd catkin_ws
$ catkin_test_results build/<your_pkg>
```



```
Summary: 2 tests, 0 errors, 1 failures, 0 skipped
robert@robert-IMS:~/catkin_ws_3$ echo $?
1
```

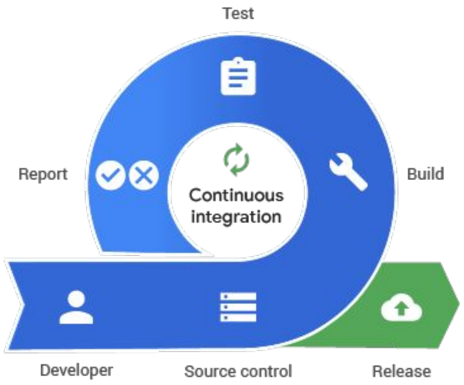
0 = success
1 = fail

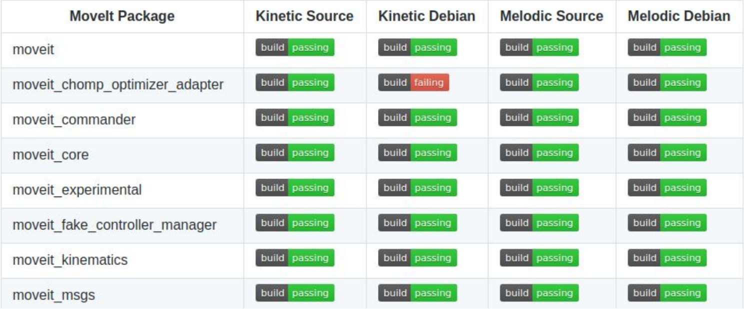
Continuous Integration

Continuous Integration - overview

Automated routines that

- Are performed on pull request basis or periodically
- Validate the code
- Build the code
- Test the code
- Indicate the results via
  - Email
  - Status messages: build passing





MoveIt Package	Kinetic Source	Kinetic Debian	Melodic Source	Melodic Debian
moveit	build passing	build passing	build passing	build passing
moveit_chomp_optimizer_adapter	build passing	build failing	build passing	build passing
moveit_commander	build passing	build passing	build passing	build passing
moveit_core	build passing	build passing	build passing	build passing
moveit_experimental	build passing	build passing	build passing	build passing
moveit_fake_controller_manager	build passing	build passing	build passing	build passing
moveit_kinematics	build passing	build passing	build passing	build passing
moveit_msgs	build passing	build passing	build passing	build passing

https://cloud.google.com/solutions/continuous-integration/

Continuous Integration - options

- Cloud services

Travis CI
https://travis-ci.org/

GitLab
https://gitlab.com

- In-house/corporate servers

Jenkins
https://jenkins.io/

GitLab
https://gitlab.com

- CI jobs on ros build farm (hosted by OSRF)

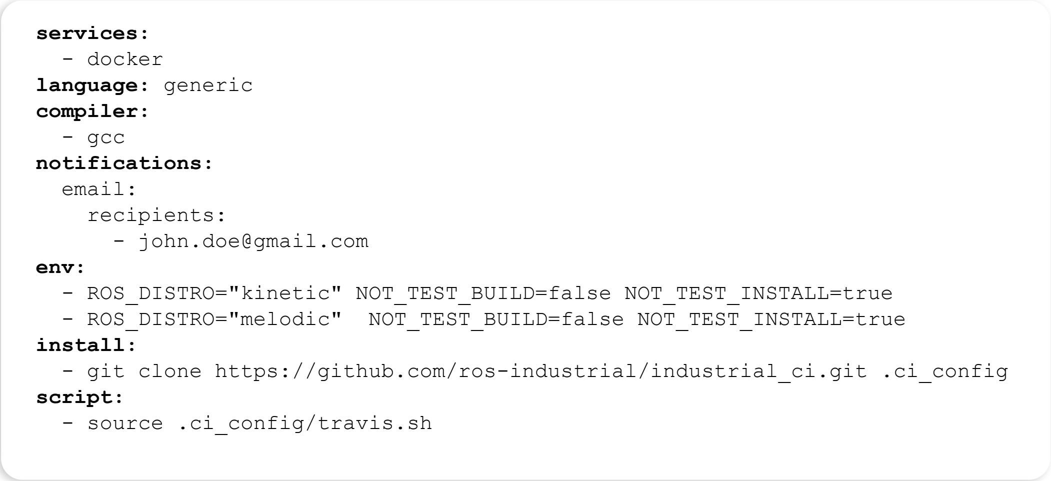
http://build.ros.org/ http://wiki.ros.org/buildfarm

Continuous Integration - set-up via industrial_ci

- https://github.com/ros-industrial/industrial_ci
- Set of Docker based scripts for automating common tests:
  - ROS compilation and ROS testing scripts
  - catkin_lint
  - ABI compliance tests
- Supported platforms
  - Travis and Gitlab
  - Custom CI servers, e.g. Jenkins

Continuous Integration - Travis CI

- Add .travis.yml to your GitHub repository



```
services:
  - docker
language: generic
compiler:
  - gcc
notifications:
  email:
    recipients:
      - john.doe@gmail.com
env:
  - ROS_DISTRO="kinetic" NOT_TEST_BUILD=false NOT_TEST_INSTALL=true
  - ROS_DISTRO="melodic"  NOT_TEST_BUILD=false NOT_TEST_INSTALL=true
install:
  - git clone https://github.com/ros-industrial/industrial_ci.git .ci_config
script:
  - source .ci_config/travis.sh
```

- Enable CI for your repository
  https://docs.travis-ci.com/user/getting-started

Workshop