Robot Operating System

ROS Installation

Setup your sources.list

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
\$(lsb\_release -sc) main" $>$ /etc/apt/sources.list.d/
ros-latest.list'
```

Set up your keys

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.
net:80  --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

Update debian package index

```
sudo apt-get update
```

Install desktop full version

```
sudo apt-get install ros-kinetic-desktop-full
```

Initialize rosdep

```
sudo rosdep init &&
rosdep update
```

NOTE: You may need to copy by hand the commands (ctrl+c may not work)

More Info

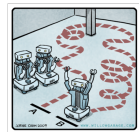
http://wiki.ros.org/kinetic/Installation/Ubuntu

What is ROS ?

ROS = Robot Operating System

+

+



+

Plumbing

- Process management
- Inter-process communication
- Device drivers

Tools

- Simulation
- Visualization
- Graphical user interface
- Data logging

Capabilities

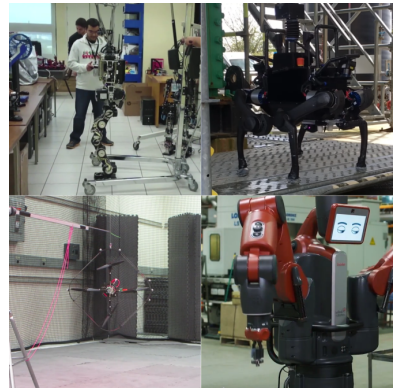
- Control
- Planning
- Perception
- Mapping
- Manipulation

Ecosystem

- Package organization
- Software distribution
- Documentation
- Tutorials

History of ROS

- Originally developed in 2007 at the Stanford Artificial Intelligence Laboratory
- Since 2013 managed by OSRF
- Today used by many robots, universities and companies
- De facto standard for robot programming



ROS Philosophy

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

ROS Workspace Environment

Defines context for the current workspace
Default workspace loaded with

```
source /opt/ros/kinetic/setup.bash
```

Create work space and initialize

```
mkdir -p ~/itesm_ws/src
cd  ~/itesm_ws/
catkin_make
```

Overlay your *catkin workspace* with

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

Check your workspace with

```
echo $ROS_PACKAGE_PATH
```

See setup with

```
cat ~/.bashrc
```

More Info
http://wiki.ros.org/kinetic/Installation/Ubuntu
http://wiki.ros.org/catkin/workspaces

ROS Master

Manages the communication between nodes
Every node registers at startup with the master

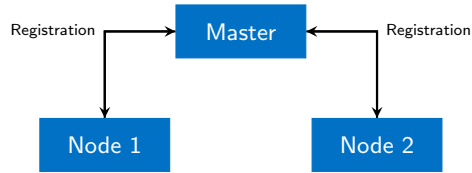
Start a master with

```
roscore
```

More Info
http://wiki.ros.org/Master

ROS Nodes

- Single-purpose, executable program
- Individually compiled, executed, and managed
- Organized in packages



Run a node with

```
rosrun package_name node_name
```

See active nodes with

```
rosnode list
```

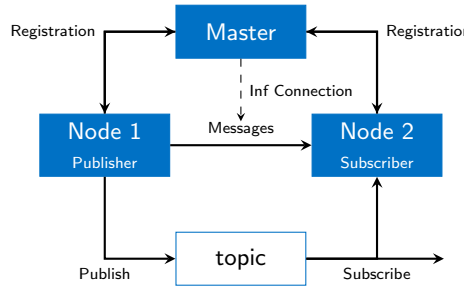
Retrieve information about a node with

```
rosnode info node_name
```

More Info
http://wiki.ros.org/rosnode

ROS Topics

- Nodes communicate over topics
  - Nodes can publish or subscribe to a topic
  - Typically, 1 publisher and n subscribers
- Topic is a name for a stream of messages



List active topics with

```
rostopic list
```

Subscribe and print the contents of a topic with

```
rostopic echo /topic
```

Show information about a topic with

```
rostopic info /topic
```

More Info
http://wiki.ros.org/rostopic

ROS Messages

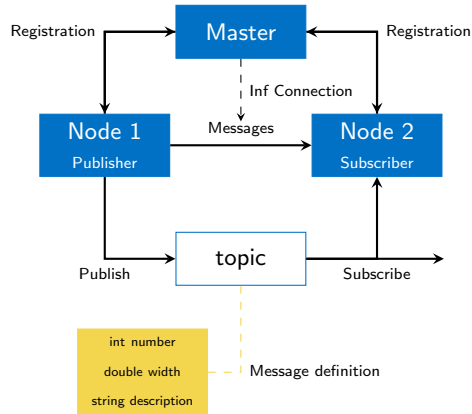
- Data structure defining the type of a topic
- Compromised of a nested structure of integers, floats, booleans, strings etc. and arrays of objects
- Defined in *.msg files

See the type of a topic

```
rostopic type /topic
```

Publish a message to a topic

```
rostopic pub /topic type args
```



More Info
http://wiki.ros.org/Messages

ROS Messages

Pose Stamped Example

geometry_msgs/Point.msg

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

geometry_msgs/PoseStamped.msg

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

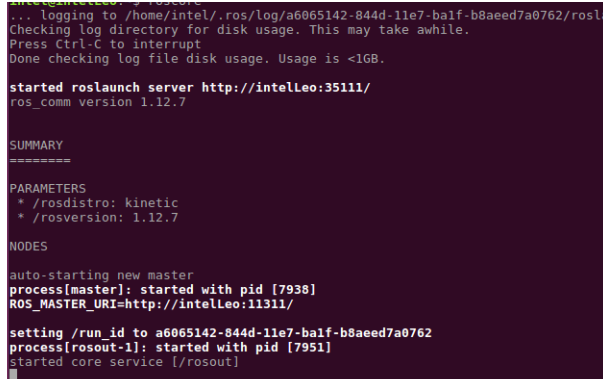
```
std_msgs/Header header
   uint32 seq time stamp
   string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

Example

Console Tab Nr. 1 – Starting a *roscore*

Start a roscore with

```
roscore
```

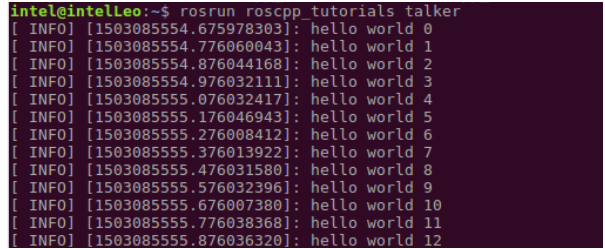


Example

Console Tab Nr. 2 – Starting a *talker* node

Run a talker demo node with

```
rosrun roscpp_tutorials talker
```



Example

Console Tab Nr. 3 – Analyze *talker* node

See the list of active nodes

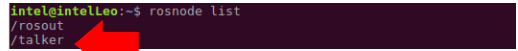
```
rosnode list
```

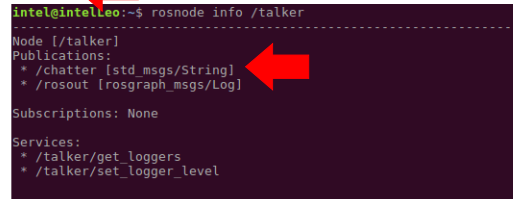
Show information about the talker node

```
rosnode info /talker
```

See information about the chatter topic

```
rostopic info /chatter
```





Example

Console Tab Nr. 3 – Analyze *chatter* topic

Check the type of the chatter topic

```
rostopic type /chatter
```

```
intel@intelLeo:~$ rostopic type /chatter
std_msgs/String
```

Show the message contents of the topic

```
rostopic echo /chatter
```

```
intel@intelLeo:~$ rostopic echo /chatter
data: hello world 807
---
data: hello world 808
---
data: hello world 809
---
data: hello world 810
```

Analyze the frequency

```
rostopic hz /chatter
```

```
^Cintel@intelLeo:~$ rostopic hz /chatter
subscribed to [/chatter]
average rate: 10.000
        min: 0.100s max: 0.100s std dev: 0.00003s window: 10
average rate: 9.999
        min: 0.100s max: 0.100s std dev: 0.00005s window: 20
average rate: 10.000
        min: 0.100s max: 0.100s std dev: 0.00011s window: 30
```

Example

Console Tab Nr. 4 – Starting a *listener* node

Run a listener demo node with

```
rosrun roscpp_tutorials listener
```

Example

Console Tab Nr. 3 – Analyze

See the new listener node with

```
rosnode list
```

Show the connection of the nodes over the chatter topic with

```
rostopic info /chatter
```

Example

Console Tab Nr. 3 – Publish Message from Console

Close the talker node in console nr. 2 with
Ctrl + C

Publish your own message with

```
rostopic pub /chatter std_msgs/String
"data: 'ITESM ROS Course'"
```

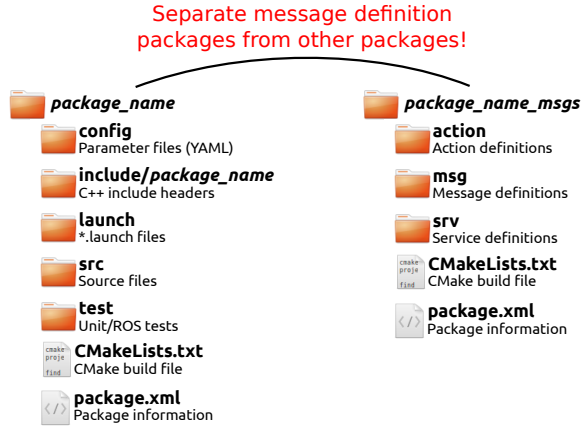
Check the output of the listener in console nr. 4

```
intel@intelLeo:~$ rostopic pub /chatter std_msgs/String "data: 'ITESM ROS Course'"
publishing and latching message. Press ctrl-C to terminate
[ INFO] [1503085731.025308570]: I heard: [hello world 1593]
[ INFO] [1503085731.125018721]: I heard: [hello world 1594]
[ INFO] [1503085731.225000965]: I heard: [hello world 1595]
[ INFO] [1503085731.324976646]: I heard: [hello world 1596]
[ INFO] [1503085783.288099483]: I heard: [ITESM ROS Course]
```

ROS Packages

- ROS software is organized into packages, which can contain source code, launch files, configuration files, message definitions, data, and documentation
- A package that builds up on/requires other packages (e.g. message definitions), declares these as dependencies
  To create a new package, use

```
catkin_create_pkg package_name
{dependencies}
```



More Info
`http://wiki.ros.org/Packages`

ROS Packages

package.xml

- The package.xml file defines the properties of the package
  - Package name
  - Version number
  - Authors
  - **Dependencies on other packages**
  - ...

```
<?xml version="1.0"?>
<package format="2">
    <name>ros_package_template</name>
    <version>0.1.0</version>
    <description>A template for ROS packages.</description>
    <maintainer email="lecamp@g...">Leo Campos</maintainer>
    <license>BSD</license>
    <url type="website">lecampos.github.io</url>
    <author email="lecamp@g...">Leo Campos</author>

    <buildtool_depend>catkin</buildtool_depend>

    <depend>roscpp</depend>
    <depend>sensor_msgs</depend>
</package>
```

More Info
http://wiki.ros.org/roslaunch/XML/include

ROS Packages

CMakeLists.txt

The CMakeLists.txt is the input to the CMakebuild system

1. Required CMake Version (cmake_minimum_required)
2. Package Name (project())
3. Find other CMake/Catkin packages needed for build (find_package())
4. Message/Service/Action Generators (add_message_files(), add_service_files(), add_action_files())
5. Invoke message/service/action generation (generate_messages())
6. Specify package build info export (catkin_package())
7. Libraries/Executable to build (add_library()/add_executable()/target_link_libraries())
8. Tests to build (catkin_add_gtest())
9. Install rules (install())

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_package_template)

## Use C++11
add_definitions(--std=c++11)

## Find catkin macros and libraries
find_package(catkin REQUIRED
    COMPONENTS
    roscpp
    sensor_msgs
)
```

More Info

http://wiki.ros.org/catkin/CMakeLists.txt

ROS Packages

CMakeLists.txt Example

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_arduino_car)
add_definitions(--std=c++11)
find_package(catkin REQUIRED
    COMPONENTS roscpp sensor_msgs )
catkin_package(
    INCLUDE_DIRS include
    # LIBRARIES
    CATKIN_DEPENDS roscpp sensor_msgs
    # DEPENDS
)
include_directories(include ${catkin_INCLUDE_DIRS})
add_executable(${PROJECT_NAME}
src/${PROJECT_NAME}_node.cpp
src/comm_interface.cpp )
target_link_libraries(${PROJECT_NAME}
${catkin_LIBRARIES})
```

Use the same name as in the package.xml

We use C++11 by default

List the packages that your package requires to build (have to be listed in package.xml)

Specify build export information

- INCLUDE_DIRS: Directories with header files
- LIBRARIES: Libraries created in this project
- CATKIN_DEPENDS: Packages dependent projects also need
- DEPENDS: System dependencies dependent projects also need (have to be listed in package.xml)

Specify locations of of header files

Declare a C++ executable

Specify libraries to link the executable against

ROS C++ Client Library (roscpp)

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

int main(int argc, char** argv)
{
    ros::init(argc, argv, "hello_world");
    ros::NodeHandle nodeHandle;
    ros::Rate loopRate(10);

    unsigned int count = 0;
    while(ros::ok()) {
        ROS_INFO_STREAM("Hello_World"
                        << count);
        ros::spinOnce();
        loopRate.sleep();
        count++;
    }

    return 0;
}
```

More Info
http://wiki.ros.org/roscpp
http://wiki.ros.org/roscpp/Overview

ROS main header file include

ros::init(...) has to be called before calling other ROS functions

The node handle is the access point for communications with the ROS system (topics, services, parameters)

ros::Rate is a helper class to run loops at a desired frequency ros::ok() checks if a node should

continue running

Returns false if SIGINT is received (Ctrl + C) or ros::shutdown() has been called

ROS_INFO() logs messages to the filesystem

ros::spinOnce() processes incoming messages via callbacks

ROS C++ Client Library (roscpp)

Node Handle

- There are four main types of node handles

1. Default (public) node handle:
   `nh_ = ros::NodeHandle();`
2. Private node handle:
   `nh_private_ = ros::NodeHandle("~");`
3. Namespaced node handle:
   `nh_itesm_ = ros::NodeHandle("itesm");`
4. Global node handle:
   `nh_global_ = ros::NodeHandle("/");`

Recommended

Not recommended

For a node in *namespace* looking up *topic*, these will resolve to:

`/namespace/topic`

`/namespace/node/topic`

`/namespace/itesm/topic`

`/topic`

More Info
`http://wiki.ros.org/roscpp/Overview/NodeHandles`

ROS C++ Client Library (roscpp)

Subscriber

- Start listening to a topic by calling the method subscribe() of the node handle

```
ros::Subscriber subscriber =
nodeHandle.subscribe(topic, queue_size, callback_function);
```

- When a message is received, callback function is called with the contents of the message as argument
- Hold on to the subscriber object until you want to unsubscribe

ros::spin() processes callbacks and will not return until the node has been shutdown

More Info
http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers

```
#include "ros/ros.h"
#include "std_msgs/String.h"

void chatterCallback(const std_msgs::String& msg)
{
    ROS_INFO("I_heard:_[%s]", msg.data.c_str());
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "listener");
    ros::NodeHandle nodeHandle;

    ros::Subscriber subscriber =
        nodeHandle.subscribe("chatter",10,
        chatterCallback);
    ros::spin();
    return 0;
}
```

ROS C++ Client Library (roscpp)

Publisher

- Create a publisher with help of the node handle

```
ros::Publisher publisher =
nodeHandle.advertise<message_type>(topic, queue_size);
```

- Create the message contents
- Publish the contents with

```
publisher.publish(message);
```

More Info
http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers

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

int main(int argc, char **argv) {
    ros::init(argc, argv, "talker");
    ros::NodeHandle nh;
    ros::Publisher chatterPublisher =
        nh.advertise<std_msgs::String>("chatter", 1);
    ros::Rate loopRate(10);

    unsigned int count = 0;
    while (ros::ok()) {
        std_msgs::String message;
        message.data = "hello_world" +
            std::to_string(count);
        ROS_INFO_STREAM(message.data);
        chatterPublisher.publish(message);
        ros::spinOnce();
        loopRate.sleep();
        count++;
    }
    return 0;
}
```

ROS C++ Client Library (roscpp)

Object Oriented Programming

```cpp
#include <ros/ros.h>
#include "my_package/MyPackage.hpp"

int main(int argc, char** argv)
{
    ros::init(argc, argv, "my_package");
    ros::NodeHandle nodeHandle("~");
    my_package::MyPackage myPackage(nodeHandle);
    ros::spin();
    return 0;
}
```

class MyPackage

Main node class providing ROS interface (subscribers, parameters, timers etc.)

class Algorithm

Class implementing the algorithmic part of the node

Note: The algorithmic part of the code could be separated in a (ROS-independent) library

Specify a function handler to a method from within the class as

```cpp
subscriber_ = nodeHandle_.subscribe(topic, queue_size, &ClassName::methodName, this);
```

More Info

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

ROS + OpenCV

```cpp
#include <ros/ros.h>
#include "cv_bridge/cv_bridge.h"
#include <image_transport/image_transport.h>
#include <sensor_msgs/image_encodings.h>
#include <sensor_msgs/Image.h>
//OpenCV
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"

using namespace std;
using namespace cv;

// Video vars
static const int         camera_id       = 0;
static const int         width           = 640;
static const int         height          = 480;
static const int         fps_camera      = 30;

int main( int argc, char** argv )
{
        //Initialize ROS node
        ros::init(argc, argv, "camera_stream");
        //Initialize Cameras variables
        ros::NodeHandle nh;
        cv::VideoCapture input_video;
        input_video.open(camera_id);
        input_video.set(CV_CAP_PROP_FRAME_WIDTH,         width);
        input_video.set(CV_CAP_PROP_FRAME_HEIGHT,         height);
        input_video.set(CV_CAP_PROP_FPS, fps_camera);
        if(!input_video.isOpened())
        {
                ROS_ERROR("Couldn't Open The Camera");
                ros::shutdown();
        }
    // Image container
        cv_bridge::CvImage cvi;
        cvi.header.frame_id = "RGB";
        cvi.encoding = sensor_msgs::image_encodings::BGR8;
        //Initialize ros pubublishers
        image_transport::ImageTransport *it =
            new image_transport::ImageTransport(nh);
        image_transport::Publisher pub_rgb =
            it->advertise("/camera/image_raw",1);

        //Ros rate as fps
        ros::Rate loop_rate(fps_camera);
        ROS_INFO("Oppened id %i camera correctly", camera_id);

        while (ros::ok())
        {
                input_video.read(cvi.image);
                cvi.header.stamp = ros::Time::now();
                pub_rgb.publish( cvi.toImageMsg() );
                loop_rate.sleep();
        }
        input_video.release();
        return 0;
}
```

ROS + OpenCV

```cpp
#include <ros/ros.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>

static constexpr char RGB_IMAGE_PUBLISHER_NAME[] = "/camera/image_raw";
static constexpr char MONO_IMAGE_PUBLISHER_NAME[] = "/camera/image_mono";
void imageCallback(const sensor_msgs::ImageConstPtr& msg);
sensor_msgs::ImagePtr mono_msg;

int main(int argc, char **argv)
{
        /* Initialize ros Node*/
        ros::init(argc,argv,"rgb_to_mono");
        /* Initialize ros handler*/
        ros::NodeHandle nh;
        /* Initialize image transport handler*/
    image_transport::ImageTransport it(nh);
        /* Initialize image transport*/
        image_transport::Publisher mono_publisher =
            it.advertise(MONO_IMAGE_PUBLISHER_NAME,1);
        /*subs to image*/
        image_transport::Subscriber sub =
            it.subscribe(RGB_IMAGE_PUBLISHER_NAME, 1, imageCallback);
        ros::Rate rate = ros::Rate(30);
        while(ros::ok())
        {
                ros::spinOnce();
                /* publish new message */
                mono_publisher.publish(mono_msg);
                rate.sleep();
        }
        return 0;
}

void imageCallback(const sensor_msgs::ImageConstPtr& msg)
{
    try
    {
        cv::Mat img_rgb=cv_bridge::toCvShare(msg, "bgr8")->image;
        cv::Mat img(msg->height,msg->width, CV_8UC1);
        cvtColor(img_rgb, img,CV_RGB2GRAY);
        /*Frame processing*/
        mono_msg = cv_bridge::CvImage(std_msgs::Header(),
            sensor_msgs::image_encodings::MONO8, img).toImageMsg();
        mono_msg->header.frame_id = "image_mono";
        mono_msg->width = img_rgb.cols;
        mono_msg->height = img_rgb.rows;
        mono_msg->is_bigendian = false;
        mono_msg->step = sizeof(unsigned char) * img_rgb.cols;
        mono_msg->header.stamp = ros::Time::now();
    }
    catch (cv_bridge::Exception& e)
    {
        ROS_ERROR("Could not convert from '%s' to 'bgr8'.",
            msg->encoding.c_str());
    }
}
```

ROS + OpenCV

```
PROJECT(camera)
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
FIND_PACKAGE(catkin_simple REQUIRED)
CATKIN_SIMPLE(ALL_DEPS_REQUIRED)
ADD_DEFINITIONS(-std=c++14 -Wunused -Wunreachable-code -Wunused-value)
SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++14
-Wno-deprecated-declarations -O3 -Wunused-value")

## Find packages macros and libraries
find_package (OpenCV REQUIRED)
INCLUDE(FindPkgConfig)
ADD_DEFINITIONS("-D${PROJECT_NAME}_VERSION=\"${${PROJECT_NAME}_VERSION}\"")
SET(ROS_NODE_SOURCES
  src/opencv_ros.cpp
        )
CS_ADD_EXECUTABLE(open_cameras ${ROS_NODE_SOURCES})
TARGET_LINK_LIBRARIES(open_cameras ${catkin_LIBRARIES} ${OpenCV_LIBS} )
SET(ROS_NODE_SOURCES
  src/listener.cpp
        )
CS_ADD_EXECUTABLE(listener ${ROS_NODE_SOURCES})
TARGET_LINK_LIBRARIES(listener ${catkin_LIBRARIES} ${OpenCV_LIBS} )
CS_EXPORT()
CS_INSTALL()
```

```
<?xml version="1.0"?>
<package format="2">
  <name>camera</name>
  <version>0.0.1</version>
  <description>a package for stream a camera cameras</description>

  <author email="leo.campos@tec.mx">Leo Campos</author>

  <maintainer email="leo.campos@tec.mx">Leo Campos</maintainer>

  <license>Closed</license>

  <buildtool_depend>catkin</buildtool_depend>
  <buildtool_depend>catkin_simple</buildtool_depend>
  <depend>cv_bridge</depend>
  <depend>sensor_msgs</depend>
  <depend>image_transport</depend>
  <depend>visualization_msgs</depend>
  <depend>roscpp</depend>
  <depend>roslib</depend>


</package>
```

PC to Robot

At Master (PC) run:

```
export ROS_MASTER_URI=http:// { IP_MASTER }:11311/
export ROS_IP= { IP_MASTER }
```

At Slave (robot) run:

```
export ROS_MASTER_URI=http:// { IP_MASTER }:11311/
```