Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

ROS AND UNITY
A COMPREHENSIVE INTRODUCTION

Danial Bagheri

Supervisor: Sebastian Starke

Content:

- **ROS**
  - Introduction
    - What uses ROS at the moment?
    - Peripheral units
    - What make ROS outstanding?
  - Core modules
    - Standard Message Definitions
    - Robot Geometry Library
    - ROS visualizer (RVIZ)
    - Robot Description Language (URDF)
  - Side modules
    - GAZEBO
    - MoveIt
    - OpenCV
    - Arduino
  - How ROS works
    - Publisher- subscriber Topic-Message
    - Service-client
- **Unity Introduction**
- Unity with ROS
- Stand alone Unity

ROS : Introduction[1]

ROS [1]

- **ROS** stands for **R**obot **O**perating **S**ystem . Collection of tools, libraries, and conventions to simplify the task of creating robot across a wide variety of robotic platforms. [1]
- Stablishing and controlling communication between peripheral modules of a robot : sensors, cameras , physical fingers and etc. [1]
- **ROS** started at Stanford Artificial Intelligence Lab then further developed at Willow Garage. [2]
- **ROS** is fully functional on Ubuntu and partially functional on other OS like Windows or Mac[5]
- **ROS** is open source Therefore[5]:
  - It is free
  - There is a large community of contributors. You can be one of them.

  http://wiki.ros.org/

Willow Garage [3]

ubuntu [4]

ROS.org [5]

[1 - Powering the world's Robots- ROS.ORG- http://www.ros.org/]

[2 - Powering the world's Robots- ROS.ORG - History http://www.ros.org/history]

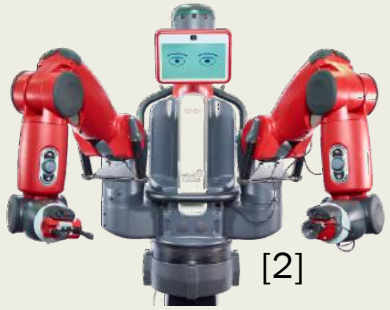
[3 - Willow Garage http://www.willowgarage.com/pages/software/ros-platform]

[4 - Ubuntu - The Ubuntu stacked logo http://design.ubuntu.com/brand/ubuntu-logo]

[5 – Wiki.ros.org - Introduction- http://wiki.ros.org/ROS/Introduction]

What uses ROS at the moment? [1]

- **Almost all robots you have seen in Academic and to some extend in industry.**
- Humanoid Robots : Nao®, GeRo®, Robonaut 2, ROBOTIS Thormang3, REEM® , …
- Manipulators: Barrett WAM ®, Baxter®, …


[2]


[3]


[4]


[5]


[6]


[7]

- Multi-fingered graspers : BarrettHand® , shadowHand, ..
- Intelligent vehicles : quadrotor helicopters, Autonomous cars , …


[8]

[1 - Powering the world's Robots- ROS.ORG – Robots - http://wiki.ros.org/Robots]

[2 – Pullman Academic – Baxter robot- http://www.pullmanacademic.com.au/Products_Robotics_Baxter.html]

[3 – Robotnic – BARRETT WAM - http://www.robotnik.eu/robotics-arms/barrett-wam/]

[4 - ROS Spotlight: Pal Robotics' REEM-C http://www.ros.org/news/2013/12/ros-spotlight-pal-robotics-reem-c.html]

[5 – German robot - Opensource humanoid robot http://www.german-robot.com/]

[6 – Generation Robotics –NAO - https://www.generationrobots.com/en/401617-programmable-humanoid-nao-evolution-robot-red.html]

[7 –Shadow Robot Company – Shadowhand -https://www.shadowrobot.com/products/dexterous-hand]

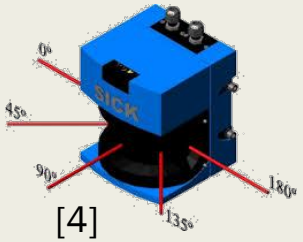
[8 – Barrett Technologies - http://www.barrett.com/products-hand.htm]

Peripheral units[1]

- 1D range finders : TeraRanger, Sharp IR range finder
- 2D range finders : SICK LMS2xx lasers, Leuze rotoScan lase
- 3D Sensors : DUO3D™ stereo camera, Kinect, PMD Camcube 3.0, …
- Cameras : USB Cameras , Ethernet camera, ….
- Force/Torque/Touch Sensors: ATI f/t sensors, Nano17 6-axis, …
- Motion Capture: OptiTrack, VICON, LEAP Motion , …
- Pose Estimation (GPS/IMU) : BOSCH® IMU, Razor's® IMU, …
- RFID : UHF RFID Reader

[2]

[3]



[4]

[6]

[5]

[7]

[8]

[1 - Powering the world's Robots- ROS.ORG – Sensors - http://wiki.ros.org/Sensors]

[2 – TeraRanger One - http://www.teraranger.com/product/teraranger-one-distance-sensor-for-drones-and-robotics/]

[3 – Drexel University- SICK LMS200 tutorial - http://www.pages.drexel.edu/~kws23/tutorials/sick/sick.html

[4 - Digital-circuitry : SICK LMS-200 / LMS-291 LIDAR LASER SCANNER RS-232 INTERFACING WITH UBUNTU & R.O.S. - http://www.digital-circuitry.com/Wordpress/sick-lms-200-lidar-laser-scanner-interfacing-with-ubuntu/]

[5 – Microsoft –Kinect for Xbox 360 - http://www.xbox.com/en-US/xbox 360/accessories/kinect]

[6 – Bosch – Mobility sensors SMI130 SMG130 SMA 130- http://www.bosch-semiconductors.de/en/automotive_electronics/news_4/ces/ces_1.html]

[7 – 9 Degrees of Freedom - Razor IMU - https://www.sparkfun.com/products/retired/10736]

[8 – ATI Industrial Automation - Multi-Axis Force / Torque Sensors- http://www.ati-ia.com/products/ft/sensors.aspx]

What make ROS outstanding?

- **ROS** is completely modular :
  - Packages : A collection of Nodes, Messages , services.
    - Nodes: a process that uses ROS framework
    - Messages: Standard definition for passing information between nodes.
  - Stack: Set of multiple package

- **ROS** is multi-language:
  - **C++ :** full functionality with **ROSCPP** library
  - **Python :** full functionality with **ROSPY** library
  - JAVA, LISP, Octave, LUA : experimental development.

- Large set of tools out of box :Standard Robot Messages, Robot Description Language, pose estimation, localization in a map, building a map, and even mobile navigation.
- Integration with other libraries for: Simulation, Image processing and etc.

Powerful ROS libraries

- **Standard Message Definitions**

For Each peripheral module or concept

code compatibility with all other part of the robotic eco system.

categorized by types in different packages.

Package : **geometry_msgs**

- Message Types available in this package:
  - **Point**
  - **Pose**
  - **Transform**
  - **...**

- Example of a message structure:
  - Package : **sensor_msgs**
  - Message Type : **Imu**

std_msgs/Header header
geometry_msgs/Quaternion orientation
float64[9] orientation_ covariance
geometry_msgs/Vector3 angular_velocity
float64[9] angular_veloci ty_covariance
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance

Powerful ROS libraries

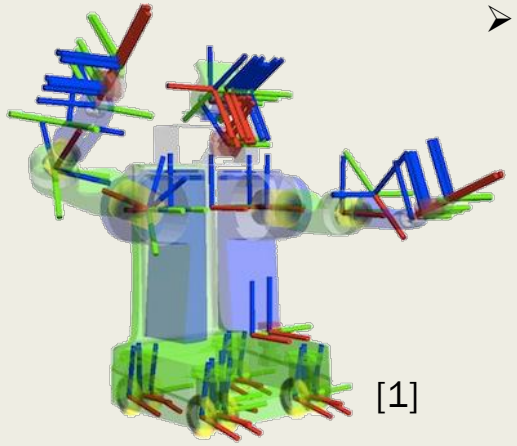
- **Robot Geometry Library**

This is essential to keep track of position of each part of robot , regarding to the other parts. **where is the hand, in respect to the head ? Where is robot1 regarding to the hand of robot2 ?**

- **Transform library (TF)** is a core library of ROS and provides a coordinate tracking system.
- TF is not a centralized library
- works base on publisher/subscriber messaging system of ROS.

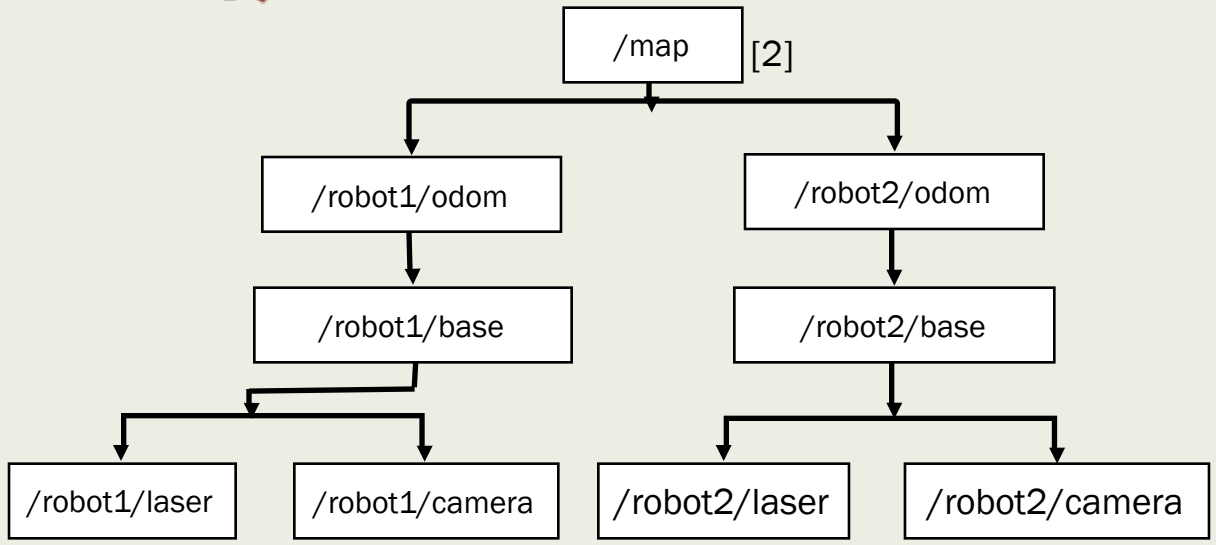
Every node has :

- Publisher (user needs to write)
- Listener (user needs to write)



[1]

- TF listeners listen to the frames and provides a **Tree** which describes how coordinate systems are related to each other.



[1 - ROS.ORG – Robot Geometry Library - http://www.ros.org/core-components/]

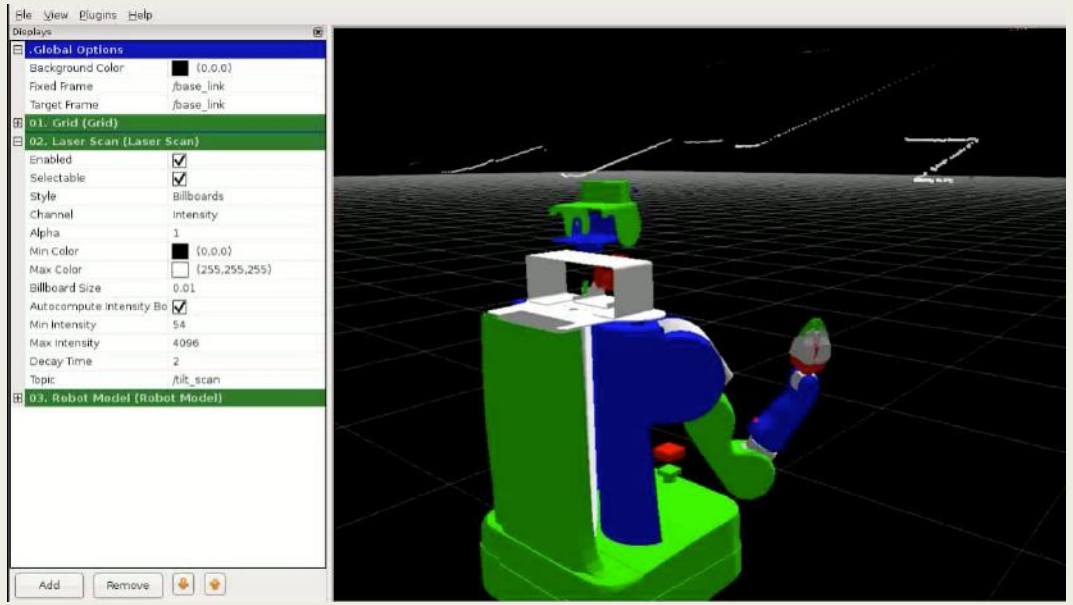
[2 – TF ROS tutorial - https://www.youtube.com/watch?v=Xf25dVrG5ks]

Powerful ROS libraries

- **ROS visualizer (RVIZ)**

- **RVIZ** is the default 3D visualization tool for.
- **RVIZ** is not a ”simulator”.
- **RVIZ** can show data that it has a plugin for displaying (DisplayTypes) and has been published by nodes:
  - **Axes** : Displays a set of Axes
  - **Camera**: Creates a new rendering window from the perspective of a camera
  - **Map** : Displays a map on the ground plane
  - **Pose** : Draws a pose as an arrow or axes.
  - .....
  - Complete set:
  http://wiki.ros.org/rviz/DisplayTypes

- Each DisplayType uses specific message.
  Axes => sensor_msgs/JointStates



[Powering the world’s Robots- ROS.ORG - RVIZ Camera type http://wiki.ros.org/rviz/DisplayTypes/Camera]

Powerful ROS libraries

- **Robot Description Language (URDF)**

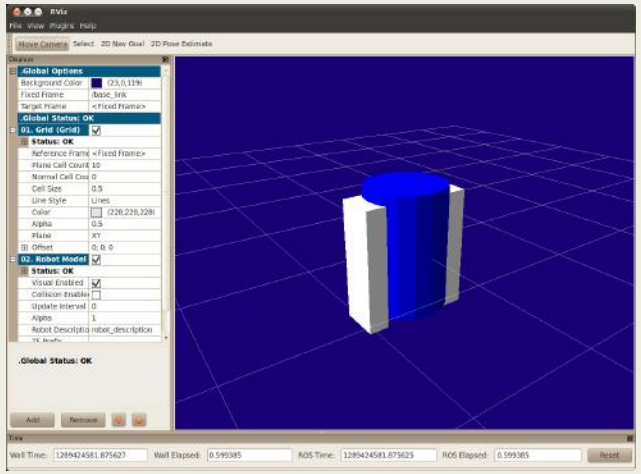
Describe a robot in a machine readable format.

URDF is an XML file describing following physical properties:

- Main parts: cylinder, box, length, radius, …
- Joints : continuous joints, prismatic joint, planar joint, Joint Dynamics (friction, damping) , Inertia

Used by different tools for simulation, visualization and motion planning:

- Rviz
- Gazebo
- Moveit
- Stage



[ROS.ORG - Building a Visual Robot Model with URDF from Scratch
http://wiki.ros.org/urdf/Tutorials/Building a Visual Robot Model with URDF from Scratch

- Example of an URDF file:

```
<?xml version="1.0"?>
    <robot name="multipleshapes">
     <link name="base_link">
      <visual>
       <geometry>
        <cylinder length="0.6" radius="0.2"/>
       </geometry>
      </visual>
     </link>
     <link name="right_leg">
      <visual>
       <geometry>
        <box size="0.6 .1 .2"/>
       </geometry>
      </visual>
     </link>
     <joint name="base_to_right_leg" type="fixed">
      <parent link="base_link"/>
      <child link="right_leg"/>
     </joint>
    </robot>
```

Powerful ROS 3rd party tools

➢ **GAZEBO**

- Simulation environment and supports many robots and sensors.
- Developing and test a **node** without a physical robot.
- Deployment of after test with minimal change.
- Start with **'gazebo'** command
- **'gzserver'** :
  - Run the physics
  - Sensor data generation
  - Can be used without any GUI
- **'gzclient':**
  - Provide a GUI for visualization of simulation

- **URDF in Gazebo :** URDF describes kinematic and dynamic properties of a robot.

- Not enough information for Gazebo for accurate simulation : pose, friction, …

- **Simulation Description Format(SDF)** invented for simulation in Gazebo.
- Stable, robust, and extensible format for describing all aspects of robots, static and dynamic objects, lighting, friction and even physics.

- **SDF** uses XML files like URDF.

Powerful ROS 3rd party tools

➢ **GAZEBO**

- **Converting URDF to SDF**
- Add tags and modify the URDF for example:

- An <inertia> element within each <link> element must be properly specified and configured.
- Add a <gazebo> element for every <link>
- Add a <gazebo> element for every <joint>
- Add a <gazebo> element for the <robot> element
- …
- The complete instruction in Gazebo website.

➢ **Part of an SDF as example**

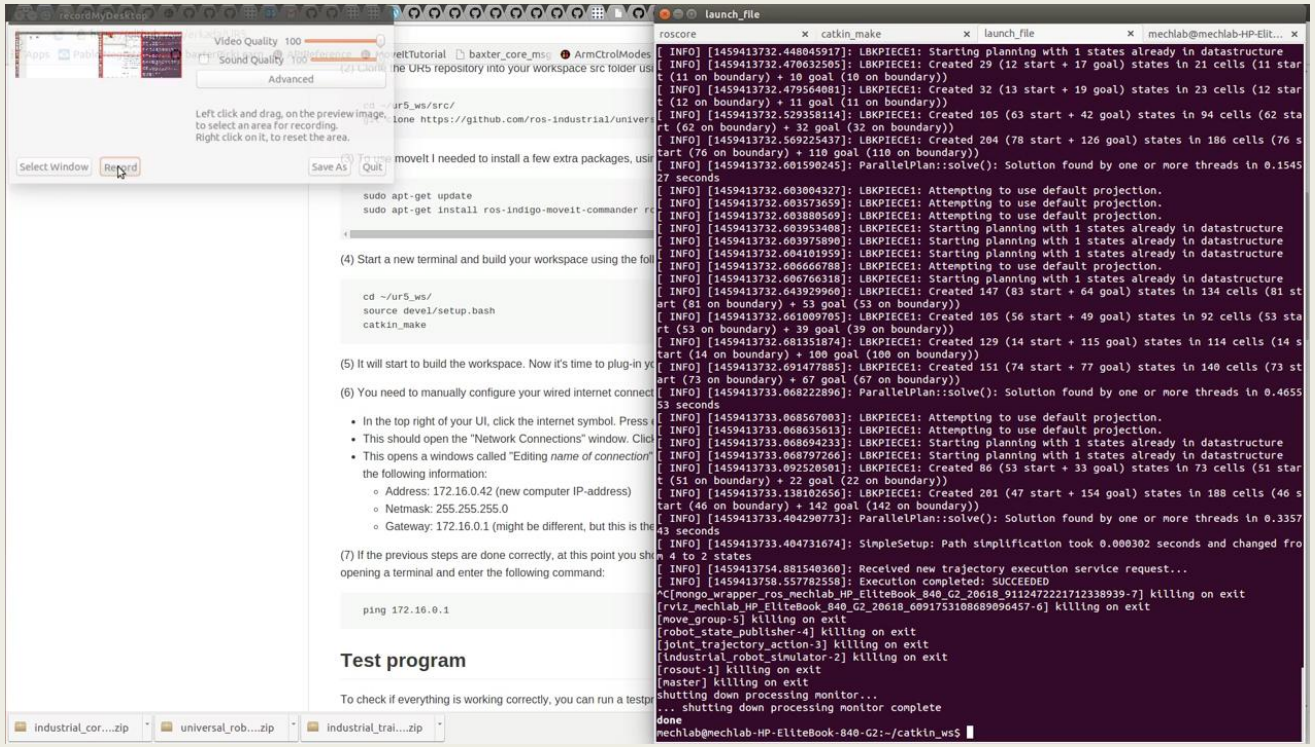
```
<camera name="head">
    <horizontal_fov>1.3962634</horizontal_fov>
    <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
    </image>
    <clip>
        <near>0.02</near>
        <far>300</far>
    </clip>
    <noise>
    <type>gaussian</type>
    <mean>0.0</mean>
    <stddev>0.007</stddev>
    </noise>
</camera>
```

Powerful ROS 3rd party tools

➢ **Moveit**

- The most widely used open-source software for manipulation, motion planning and analyzing of robot interaction with environment.

- Capabilities:
  - Collision checking
  - Integrated kinematics
  - Motion planning
  - Integrated perceptions about environment
  - Execution and monitoring
  - Interactive



MoveIt in Rviz moving the ABB robot around - https://www.youtube.com/watch?v=0hS0XUOgYXk - Pablo Negrete

Powerful ROS 3rd party tools

➢ OpenCV

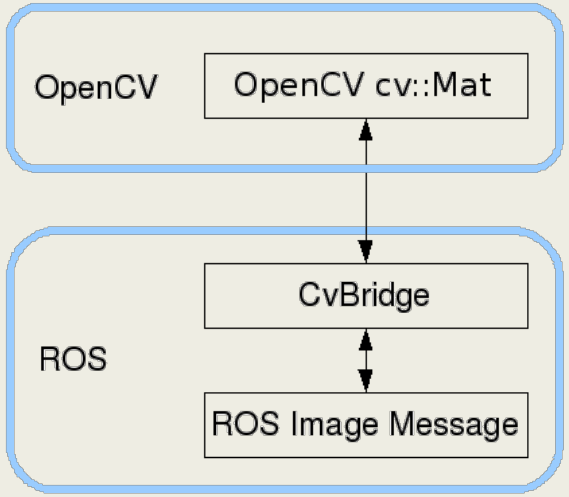
- The most powerful image processing library
- Implemented in Python and C++.
- Many functionalities out of box : Face detectio, Object tracking ,motion analysis, Feature detection and …

- **ROS** have drivers for many sort of cameras:
  - **openni_kinect** for Microsoft **kinect**
  - **gscam** for most **webcams**
  - **swissranger_camera**
  - **…**
- **ROS** uses **sensor_msgs/Image** message and OpenCV need **matrices** for images.
- **Conversion** by **cv_bridge** stack.

- **Conversion by cv_bridge : ready functions**

```
cv_ptr = cv_bridge::toCvCopy(msg,
sensor_msgs::image_encodings::BGR8);

cv::circle(cv_ptr->image, cv::Point(50, 50),
10, CV_RGB(255,0,0));
```



[ROS.ORG- vision_opencv - http://wiki.ros.org/vision_opencv]

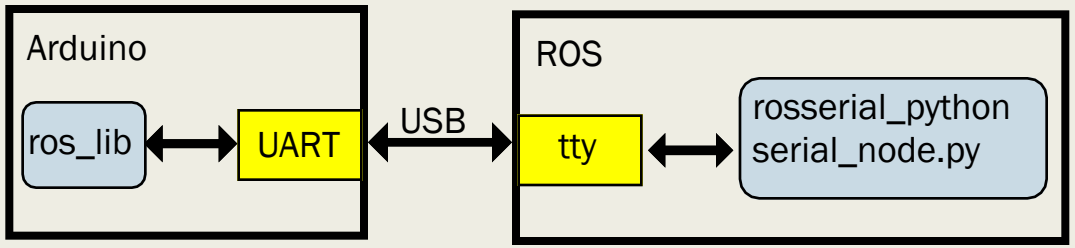
ROS and external hardware : Arduino

➢ Arduino

- A microcontroller with powerful interface library for different hardware.
- Different I/O ports : Analog and digital
- C-like language and syntax , Easy to program. Many open source projects.

[1]

[2]



[3]

[1 - Arduino Products - https://www.arduino.cc/en/Main/Products]

[2 – German robot - Opensource humanoid robot http://www.german-robot.com/]

[3 – Wiki.ros.org - rosserial_arduinoTutorials - http://wiki.ros.org/rosserial_arduino/Tutorials]

▪ Implementation

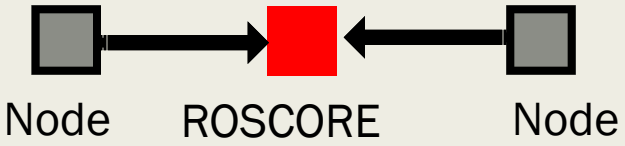
- **ROS** side : rosserial stack for serialization of message over USB [3]
- **Arduino** side: rosserial_arduino to create messages, publish, subscribe. [3]

```
#include <ros.h> <std_msgs/String.h>
ros::NodeHandle n; std_msgs::String msg;
ros::Publisher pub("/my_topic", &msg); int count = 0;
char data[100];
void setup(){
        n.initNode();
        n.advertise(pub);
        }
void loop(){
        sprintf(data, "Hello world %d", ++count);
        msg.data = data;
        pub.publish(&msg);
        n.spinOnce();
        delay(1000);
}
```

http://wiki.ros.org/rosserial_arduino

How ROS works ?

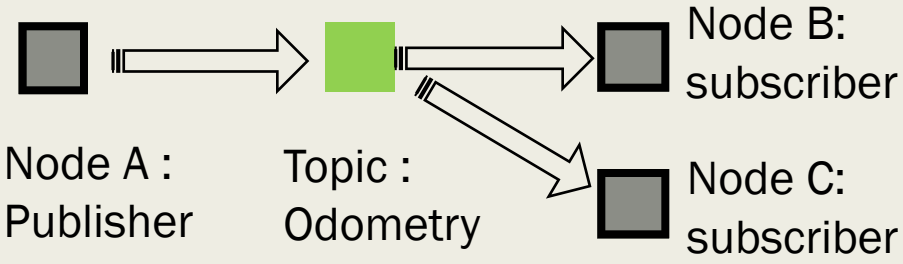
- **Nodes – Messages – Topics**
- **Node :** a process that uses ROS framework. ROSCORE connects all nodes together and provide connectivity.



- **Message:** Standard definitions for transferring data between nodes.
- **Topic:** Mechanism of transferring data between nodes.
- **Publisher:** A node which produce message and publish them.
- **Subscriber:** A node which receives the messages.

- **Workflow:**

1. Node A publish a message to a topic
2. All nodes which are subscribed to that topic, will receive the message.



- **Nodes commands:**

- rosrun package executable
- Roslaunch package_name file.launch

- **Topiccommands:**

#show list of messages inside topic

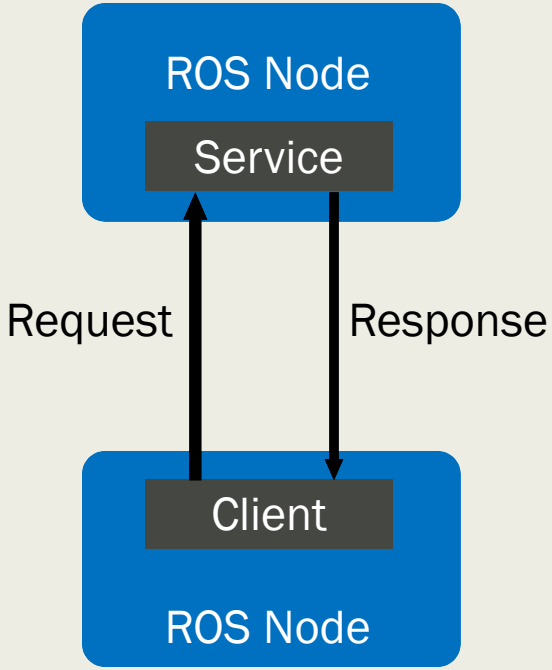
- Rostopic echo /topicName
- Rostopic list
- Rostopic info topicName

How ROS works ?

- **Service-Client**

The publish/subscribe model is very flexible but not enough for a distributed system.

- Service-Client is way to retrieve the data immediately instead of waiting for a message to be published.
- A node provides a **service** , the client node **call the service** by sending **request message.**
- **Service-client** => one-to-one
- **Topic- message** => one-to-one, one-to-many, many-to-many



[**Mathwork** - https://de.mathworks.com/help/robotics/examples/call-and-provide-ros-services.html]

Implementation example : Message-Topic

- **Subscribing to a topic**

```
# Initialize rospy
NODE_NAME = 'localization'
import roslib; roslib.load_manifest(NODE_NAME)
import rospy

# Import LaserScan message type
from nav_msgs.Odometry import *

# Scan message handler
def odom_handler(msg):
# this code is executed whenever a scan is published
[...]
# Main function
def main():
    rospy.init_node(NODE_NAME)
    rospy.Subscriber("/odom", Odometry, odom_handler)
    rospy.spin()
```

This is a callback function.
This is called whenever a message of type Odometry is received.

Topic name

Message type

Callback function

Implementation example : Message-Topic

- **Publishing to a topic**

```
# Initialize rospy
NODE_NAME = 'localization'
import roslib; roslib.load_manifest(NODE_NAME)
import rospy

# Import standard String message type
from std_msgs.msg import *

# Main function
def main():
    pub = rospy.Publisher("/scout/viewer", String)
    rospy.init_node(NODE_NAME)
    msg = "Hello world"
    pub.publish( String(msg) )
```

Topic name

Message type

Constructor call of message

Publish function

- **Main benefits of message/topic system**

  - capture messages in a file And replay them later **independently**
  - Clear communication structure between side tools and libraries. As pointed out for example in **RVIZ**

Implementation example : Service-client

- **Service**

```
# Initialize rospy
NODE_NAME = 'localization'
import roslib; roslib.load_manifest(NODE_NAME)
import rospy

# Import standard String message type
from std_msgs.msg import *

# Service handler
def handler(req):
# this code is executed whenever the service is called
return LocalizationSrvResponse()

# Main function
def main():
        rospy.init_node(NODE_NAME)
        rospy.Service("/scout/localization", LocalizationSrv, handler)
        rospy.spin()
```

Service functionality

Service name

Service type

- **client**

```
# Initialize rospy
NODE_NAME = 'viewer '
import roslib; roslib.load_manifest(NODE_NAME)
import rospy

# Import standard String message type
from std_msgs.msg import *

# Main function
def main():
        srv = rospy.ServiceProxy("/scout/localization",LocalizationSrv)
        rospy.init_node(NODE_NAME)
        response = srv(1, x, y, theta)
```

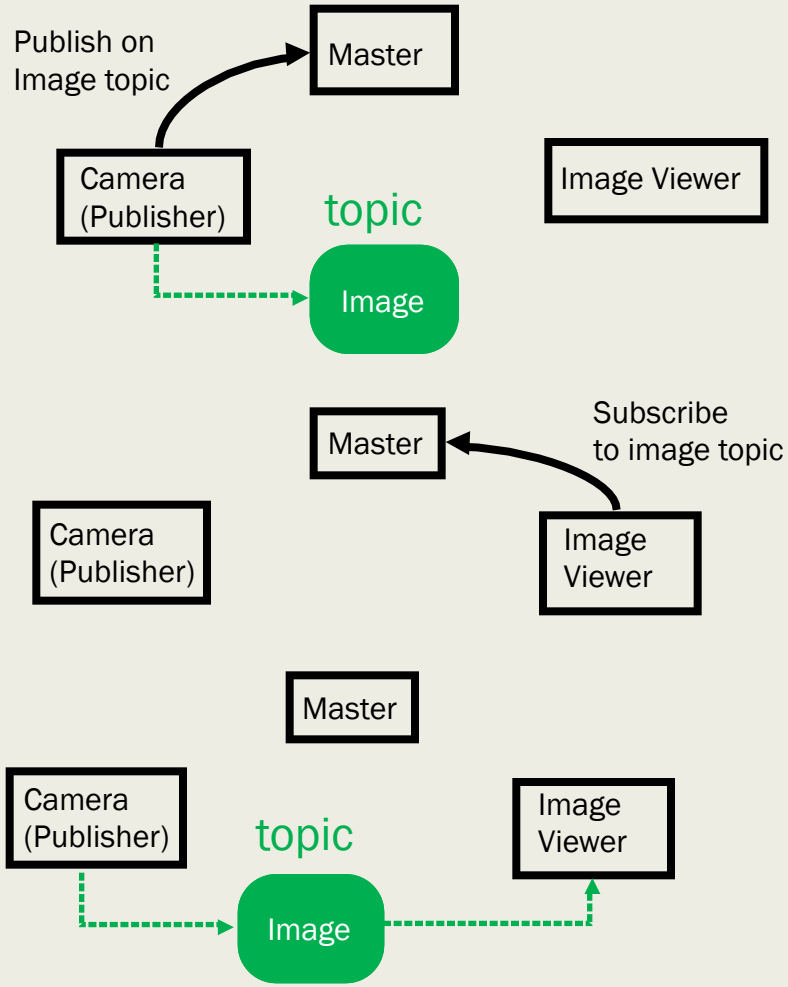
How nodes find each other : ROS Master

- One node is a ROS Master by running **roscore** command on it.
- Keep track of publishers, subscribers and topics.
- After nodes locate each other, they communicate peer-to-peer.
- Steps:

Publisher informs the ROS master about the topic and start publishing.

Subscriber informs the ROS master about the interested topics

ROS master inform Publisher that who is interested , and publisher start sending messages to them.



[ROS.ORG – ROS Master - http://wiki.ros.org/Master]

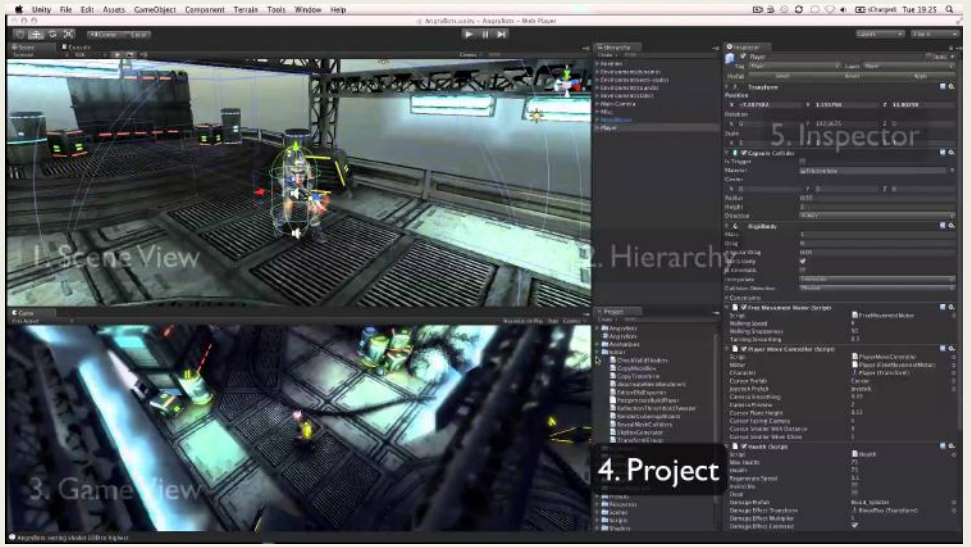
Unity: Introduction

- **Unity** is game engine used to created high qualified visual scenes.
- **Unity** is visualization tool not a simulation.
- **Unity** is widely used for virtual reality (VR) tasks because:
  - Multi-platform : OSX, Windows, MAC, Android , ….
  - Powerful physics engine : gravity and collisions
  - Very GUI lets you drag and drop elements
  - Programming languages : C# and Javascript

[1]

[2]

- **Unity interface**



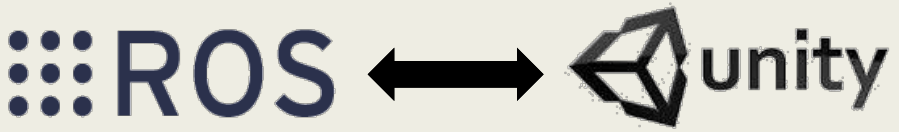
[3]

[1 – Deviant art – Angry birds logo - http://www.deviantart.com/morelikethis/421156366]

[2 – Geforde – Assassin's Creed Unity Graphics & Performance Guide - http://www.geforce.com/whats-new/guides/assassins-creed-unity-graphics-and-performance-guide]

[3- Unity Interface overview - Unity Official Tutorials - https://www.youtube.com/watch?v=5cPYpI6_yLs]

Unity with ROS



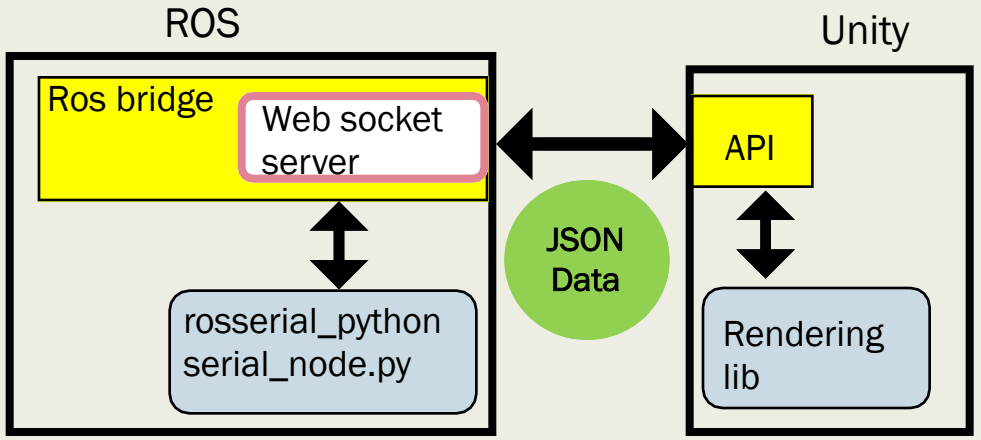
- Unity instead of RVIZ For visualization?

Not a good idea but possible.

- ROS messages => events processed by rendering loop in Unity.
- liveliness of visualization is lost because rendering should be fast.
- Method : Connection between ROS-Unity by ROS bridge.
- Rosbrige : connection to outside world by JSON API through web sockets
- roslaunch rosbridge_server rosbridge_websocket.launch

Creates a web socket server working on port 9090

- Outside software call the server/port for communication



- JSON Data examples:

{"op": "subscribe",
"topic": "/clock",
"type": "rosgraph_msgs/Clock”}.

{"op": "publish",
"topic": "/unity/joy",
"msg": msg}.

Stand alone Unity

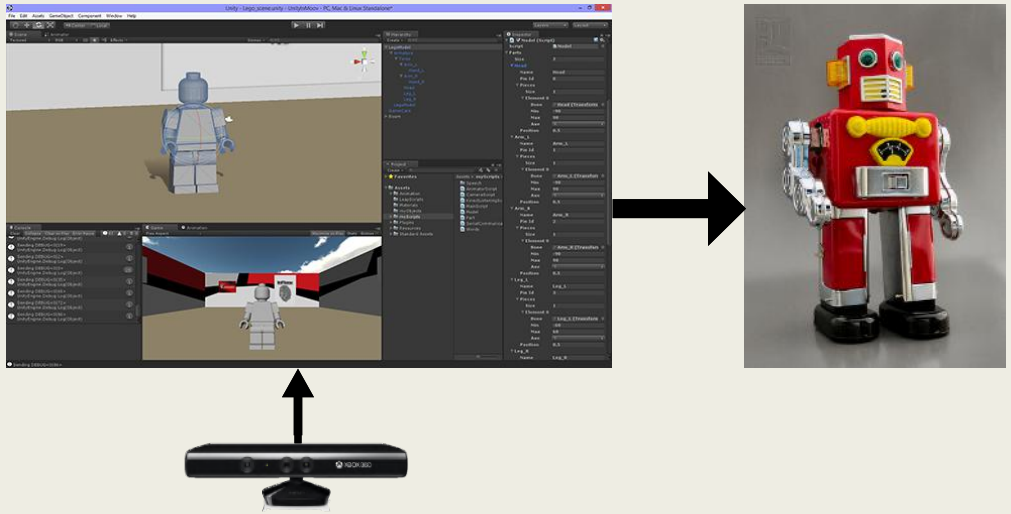
- Graphical robot controller : The reverse of previous project
- Sending move commands from graphical robot to physical robot
- Input from environment by camera, Kinect , etc to control graphical robot.

- Physical Robot => Arduino robotic frame ware

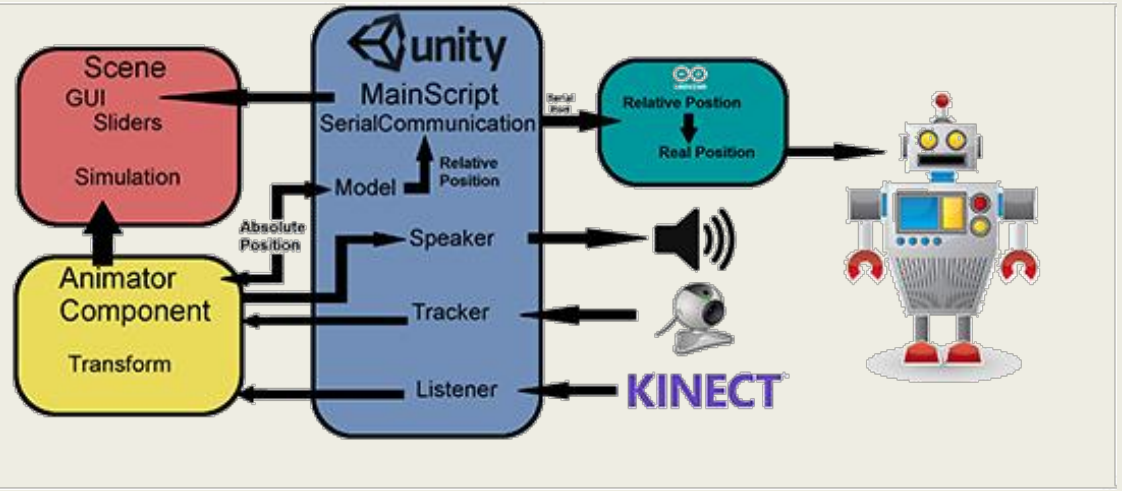
- Calculation of position, etc => Unity

- Unity to Arduino Connection => USB

- Benefit : Control robot in Real time with human interaction



[1]



[1 - The Robot Engine - Making The Unity 3D Game Engine Work For HRI
Christoph Bartneck, Marius Soucy, Kevin Fleuret, Eduardo B. Sandoval]

Conclusions

- Complete OS for Robotics
- No equivalent
- Suitable for industrial large scale robotic projects

- Powerful visualization tool
- Some equivalents: Unreal, DirectX, …
- Suitable for game, design and graphic industry
- To some extend Human Robot Interaction

- Research subject : Combining Unity3D and ROS for nice environment simulation.
- What about sensor data ?????

References:

- **ROS wiki** - http://wiki.ros.org/
- **Powering the world's Robots- ROS.ORG**- http://www.ros.org/
- **The Robot Engine - Making The Unity 3D Game Engine Work For HRI** - Christoph Bartneck, Marius Soucy, Kevin Fleuret, Eduardo B. Sandoval
- **From ROS to Unity: leveraging robot and virtual environment middleware for immersive teleoperation** - R. Codd-Downey, P. Mojiri Forooshani, A. Speers, H. Wang and M. Jenkin
- **GAZEBO - Robot simulation made easy** - http://gazebosim.org/
- **MoveIt! Motion Planning Framework** - http://moveit.ros.org/
- **Unity3D** - https://unity3d.com/
- **Mathwork** - https://de.mathworks.com/help/robotics/examples/call-and-provide-ros-services.html