SATURDAY

09/05/2020

ONLINE TRAINING

ROS TRAINING

Ismail HAMROUNI

INSAT AEROBOTIX

TRAINING
PLAN

1. BASICS LINUX COMMANDS
2. WHAT'S ROS ?
3. ROS ECOSYSTEM
4. ROS NODES & TOPICS
5. ROS SERVICES
6. FILE SYSTEM TOOLS
7. WORKSPACE & PACKAGES
8. ROS TOOLS & UTILITIES
9. ROBOT MODELING

INSAT AEROBOTIX

1. BASICS LINUX COMMANDS
2. WHAT'S ROS ?
3. ROS ECOSYSTEM
4. ROS NODES & TOPICS
5. ROS SERVICES
6. FILE SYSTEM TOOLS
7. WORKSPACE & PACKAGES
8. ROS TOOLS & UTILITIES
9. ROBOT MODELING

1. BASICS LINUX COMMANDS
2. WHAT'S ROS ?
3. ROS ECOSYSTEM
4. ROS NODES & TOPICS
5. ROS SERVICES
6. FILE SYSTEM TOOLS
7. WORKSPACE & PACKAGES
8. ROS TOOLS & UTILITIES
9. ROBOT MODELING

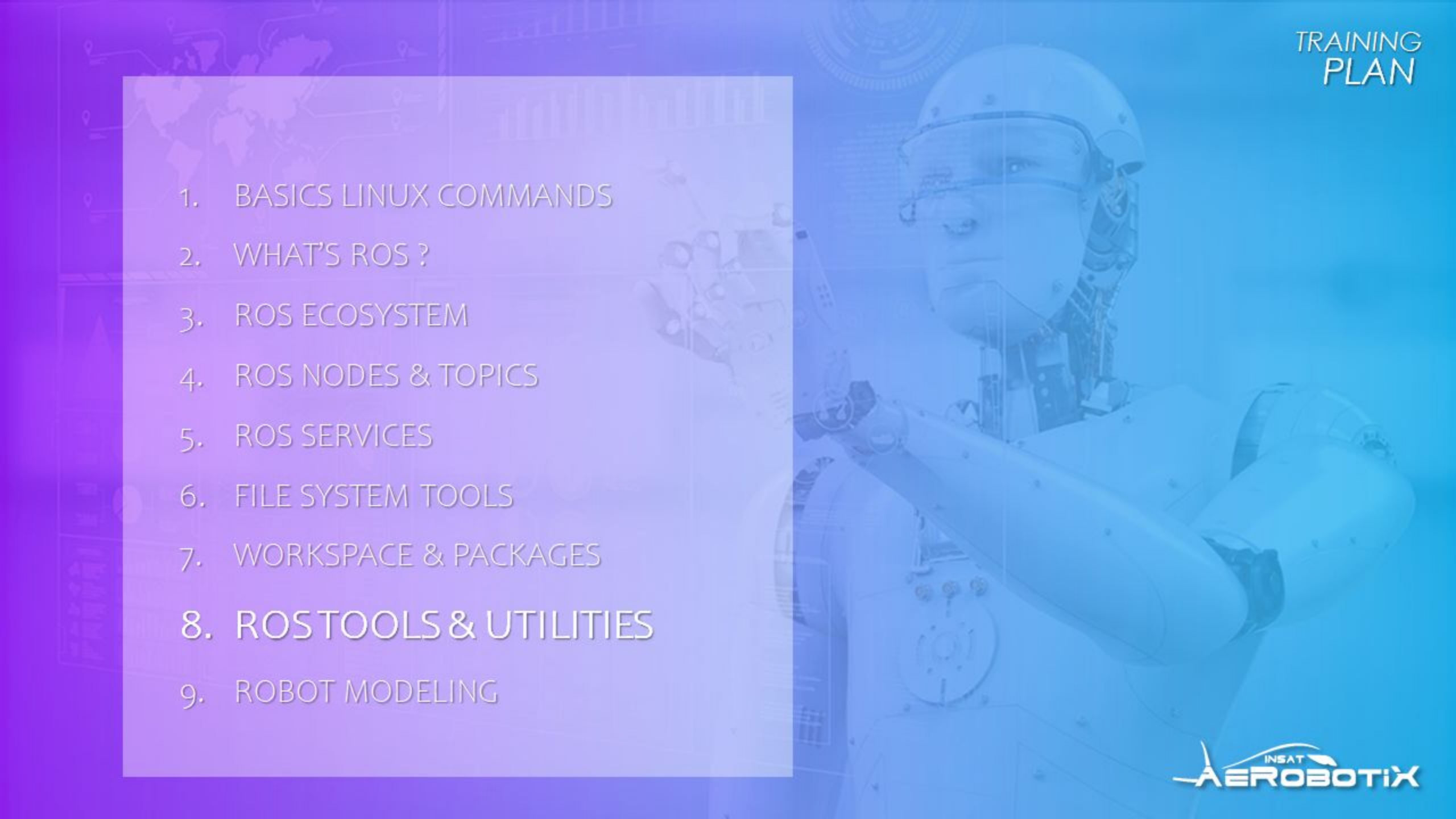
1. BASICS LINUX COMMANDS
2. WHAT'S ROS ?
3. ROS ECOSYSTEM
4. ROS NODES & TOPICS
5. ROS SERVICES
6. FILE SYSTEM TOOLS
7. WORKSPACE & PACKAGES
8. ROS TOOLS & UTILITIES
9. ROBOT MODELING

1. BASICS LINUX COMMANDS
2. WHAT'S ROS ?
3. ROS ECOSYSTEM
4. ROS NODES & TOPICS
5. ROS SERVICES
6. FILE SYSTEM TOOLS
7. WORKSPACE & PACKAGES
8. ROS TOOLS & UTILITIES
9. ROBOT MODELING

1. BASICS LINUX COMMANDS
2. WHAT'S ROS ?
3. ROS ECOSYSTEM
4. ROS NODES & TOPICS
5. ROS SERVICES
6. **FILE SYSTEM TOOLS**
7. WORKSPACE & PACKAGES
8. ROS TOOLS & UTILITIES
9. ROBOT MODELING

1. BASICS LINUX COMMANDS
2. WHAT'S ROS ?
3. ROS ECOSYSTEM
4. ROS NODES & TOPICS
5. ROS SERVICES
6. FILE SYSTEM TOOLS
7. WORKSPACE & PACKAGES
8. ROS TOOLS & UTILITIES
9. ROBOT MODELING

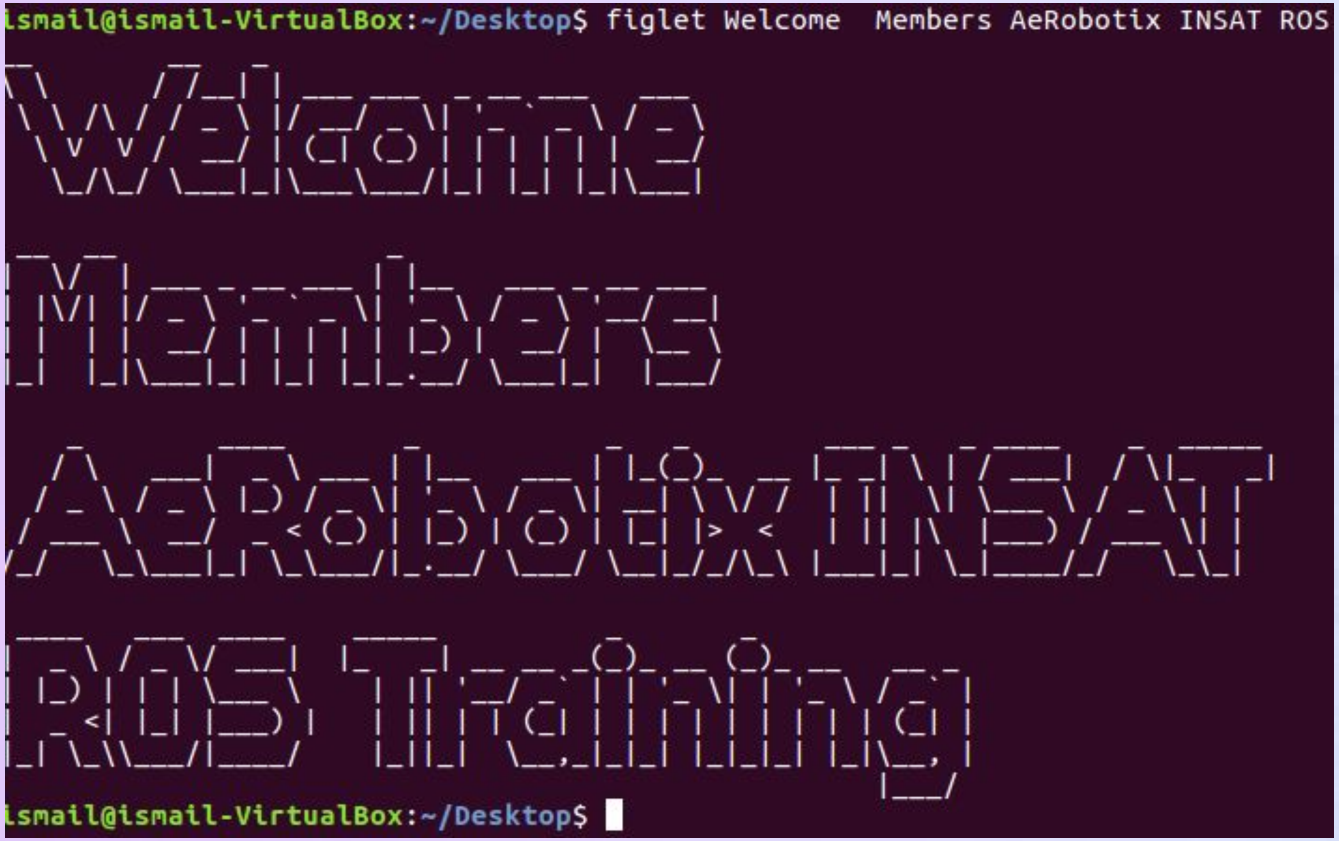
1. BASICS LINUX COMMANDS
2. WHAT'S ROS ?
3. ROS ECOSYSTEM
4. ROS NODES & TOPICS
5. ROS SERVICES
6. FILE SYSTEM TOOLS
7. WORKSPACE & PACKAGES
8. **ROS TOOLS & UTILITIES**
9. ROBOT MODELING



1. BASICS LINUX COMMANDS
2. WHAT'S ROS ?
3. ROS ECOSYSTEM
4. ROS NODES & TOPICS
5. ROS SERVICES
6. FILE SYSTEM TOOLS
7. WORKSPACE & PACKAGES
8. ROS TOOLS & UTILITIES
9. ROBOT MODELING

Basics linux Commands

In Linux, Copying, moving, deleting files, Creating directory, Execution program could not be only launched with graphical user interface like Windows but also and usually though ***Terminal Command lines***



Terminal will be our best friend in this ROS training
ENJOY

Basics linux Commands

File Commands	Fonction
$ ls	Directory listing
$ ls -al	Directory listing with hidden files
$ cd <dir>	Change directory to dir
$ pwd	Show current directory
$ mkdir <dir>	Create directory dir
$ rm <file>	Delete file
$ rm –r <dir>	Delete directory dir
$ rm –f <file>	Force remove file
$ rm -rf <dir>	Remove directory dir
$ cp <file1> <file2>	Copy file1 to file2
$ mv <file1> <file2>	Move/rename file1 to file2
$ touch <file>	Create or update file
$ head <file>	Output first 10 line of file
$ tail <file>	Output last 10 line of file

Network Commands	Fonction
$ ping host	Ping host to test connectivity

Ssh Commands	Fonction
$ ssh user@host	Connect to host as user

File permission	Fonction
$ chmod +x <file>	Make file executable

Compression	Fonction
$ gzip <file>	Compress file and rename to file.gz
$ gzip –d file.gz	Decompress file.gz

INSAT AEROBOTIX

What is ROS ?

ROS is an **open-source, meta-operating system** for your robot.

What is Meta-operating system ?

Meta operating system is built **on top** of **the operating system** and allows different processes (nodes) to communicate with each other at runtime.

The Robot Operating System (ROS) is a set of **software libraries** and **tools** that help you build robot applications.
From drivers to state-of-the-art algorithms, and with powerful developer tools,
ROS has what you need for your next robotics project. And it's all **open source**.

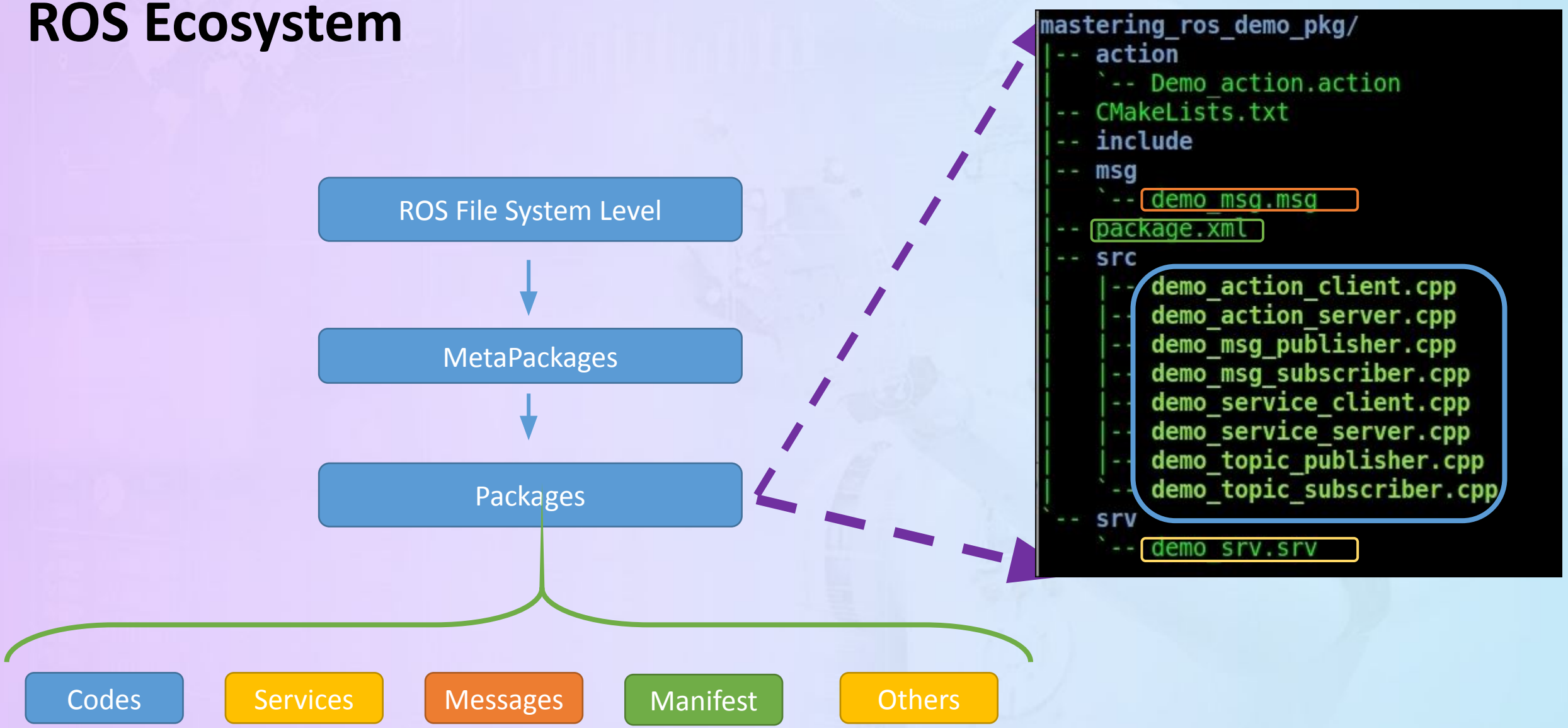
INSAT AEROBOTIX

Where is used ?

ROS is implemented on this ROBOT

INSAT AEROBOTIX

ROS Ecosystem



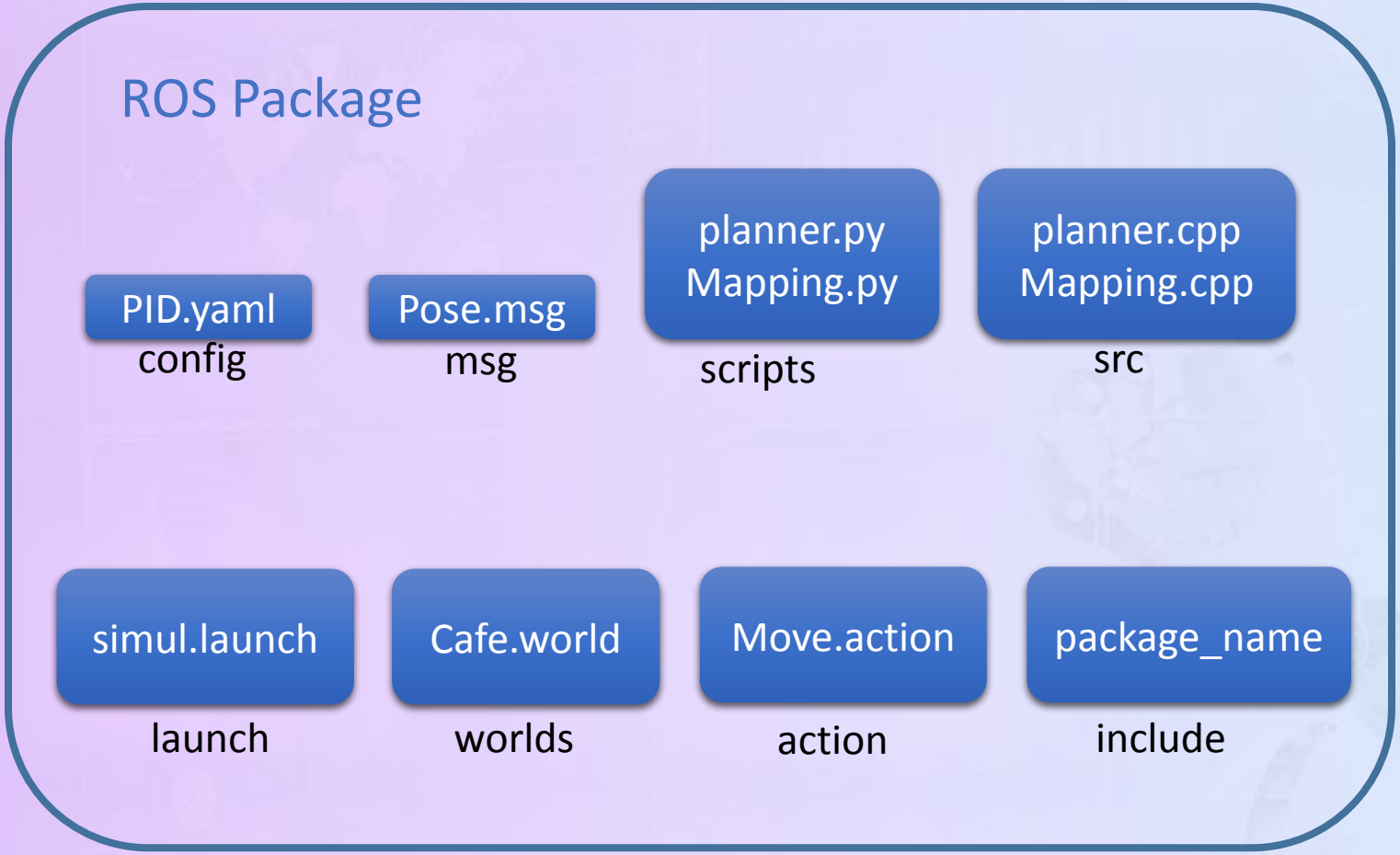
Package.XML

```xml
<?xml version="1.0"?>
<package format="2">
  <name>beginner_tutorials</name>
  <version>0.1.0</version>
  <description>The beginner_tutorials package</description>

  <maintainer email="you@yourdomain.tld">Your Name</maintainer>
  <license>BSD</license>
  <url type="website">http://wiki.ros.org/beginner_tutorials</url>
  <author email="you@yourdomain.tld">Jane Doe</author>

  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>

  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>

</package>
```

CMakeLists.txt

```
# Get the information about this package's buildtime dependencies
find_package(catkin REQUIRED
  COMPONENTS message_generation std_msgs sensor_msgs)

# Declare the message files to be built
add_message_files(FILES
  MyMessage1.msg
  MyMessage2.msg
)

# Declare the service files to be built
add_service_files(FILES
  MyService.srv
)

# Actually generate the language-specific message and service files
generate_messages(DEPENDENCIES std_msgs sensor_msgs)

# Declare that this catkin package's runtime dependencies
catkin_package(
 CATKIN_DEPENDS message_runtime std_msgs sensor_msgs
)

# define executable using MyMessage1 etc.
add_executable(message_program src/main.cpp)
add_dependencies(message_program ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

# define executable not using any messages/services provided by this package
add_executable(does_not_use_local_messages_program src/main.cpp)
add_dependencies(does_not_use_local_messages_program ${catkin_EXPORTED_TARGETS})
```

INSAT AEROBOTIX



config: All configuration files that are used in this ROS package are kept in this folder.

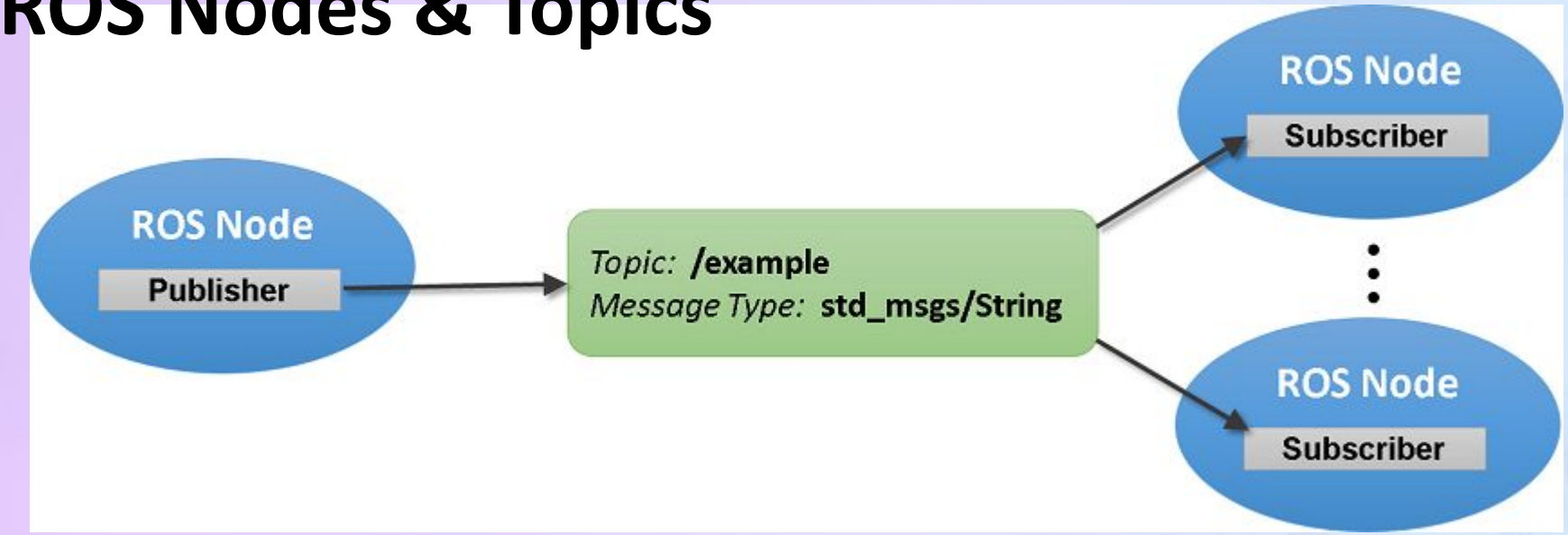
include/package_name: This folder consists of headers and libraries that we need to use inside the package.

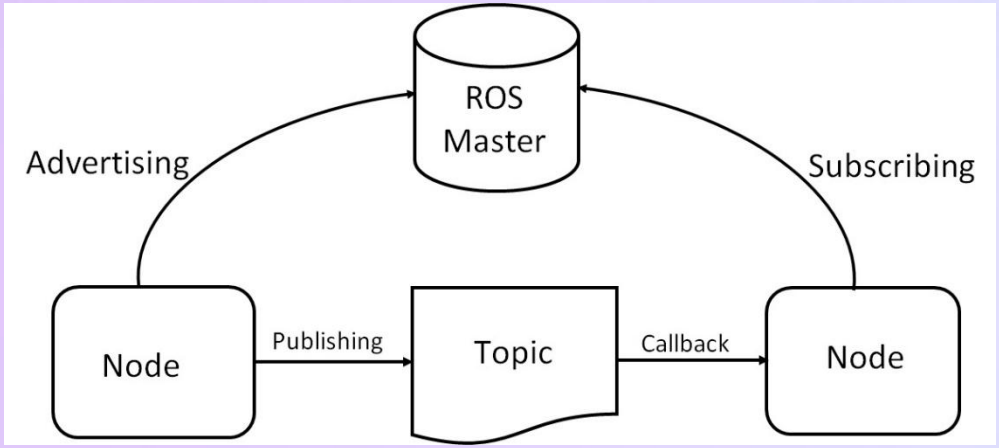
scripts: This folder keeps executable Python scripts. In the block diagram,

src: This folder stores the C++ source codes.

launch: This folder keeps the launch files that are used to launch one or more ROS nodes.
msg: This folder contains custom message definitions.
srv: This folder contains the service definitions.
package.xml: This is the package manifest file of this package.
CMakeLists.txt: This is the CMake build file of this package.

ROS Nodes & Topics



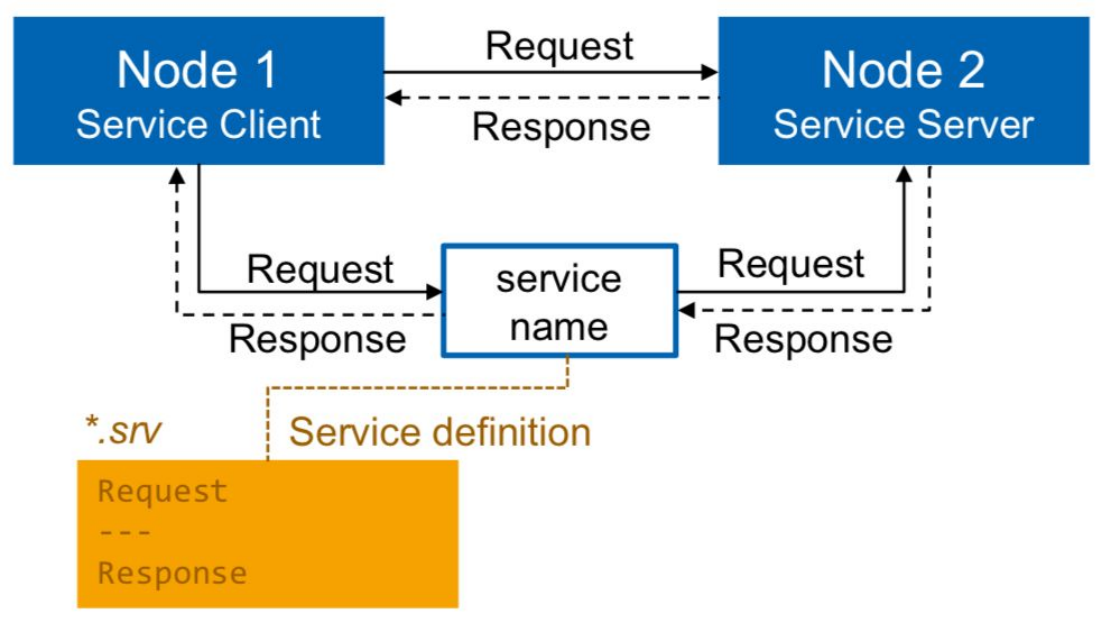


Node, is basically a process that performs computation. It is an executable program running inside your application

Topics are named buses over which nodes exchange messages

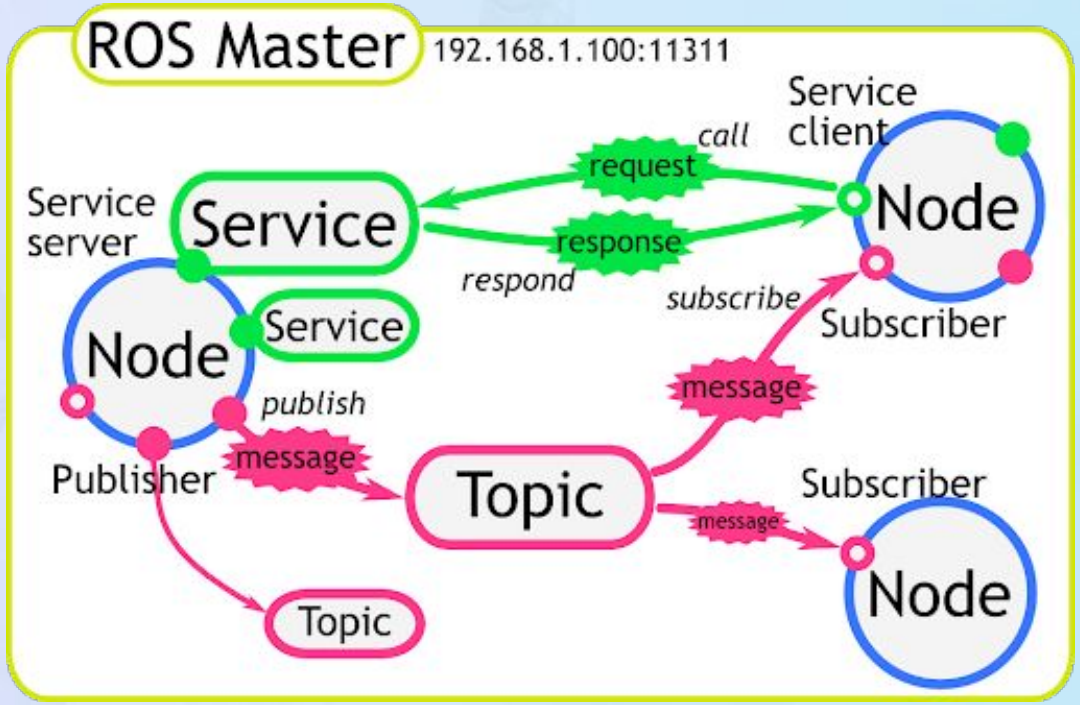
Nodes are not aware of who they are communicating with. Instead, nodes that are interested in data *subscribe* to the relevant topic.

ROS Services



Request / reply is done via a **Service**, which is defined by a pair of messages: one for the request and one for the reply

Overall architecture diagram



Filesystem Tools

Command line	Function	Example
roscore	This starts the Master	$ roscore
rosrun	This runs an executable program and creates nodes	$ rosrun [package name] [executable name]
rosnode	This shows information about nodes and lists the active nodes	$ rosnode <subcommand> [node name] ***Subcommand***: list, info
rostopic	This shows information about ROS topics	$ rostopic <subcommand> <topic name> ***Subcommands: echo***, ***info***, and ***type***
rosmsg	This shows information about the message types	$ rosmsg <subcommand> [package name]/ [message type] ***Subcommands: show***, ***type***, and ***list***
rosservice	This displays the runtime information about various services and allows the display of messages being sent to a topic	$ rosservice <subcommand> [service name] ***Subcommands: args***, ***call***, ***find***, ***info***, ***list***, and ***type***
rosparam	This is used to get and set parameters (data) used by nodes	$ rosparam <subcommand> [parameter] ***Subcommands: get***, ***set***, ***list***, and ***delete***

INSAT AEROBOTIX

Filesystem Tools

Commands

- `rospack find [package_name]`
- `roscd [locationname[/subdir]]`
- `rosls [locationname[/subdir]]`

Exemples

- `rospack find roscpp`
- `roscd roscpp`
- `rosls roscpp_tutorials`

`Find packages`

`Go to package location`

`Printing the containing files`

Link to others commands

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

Create Workspace

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

Creating Workspace

- Create directories (catkin_ws and src)
- Go to the /catkin_ws folder
- Building the workspace

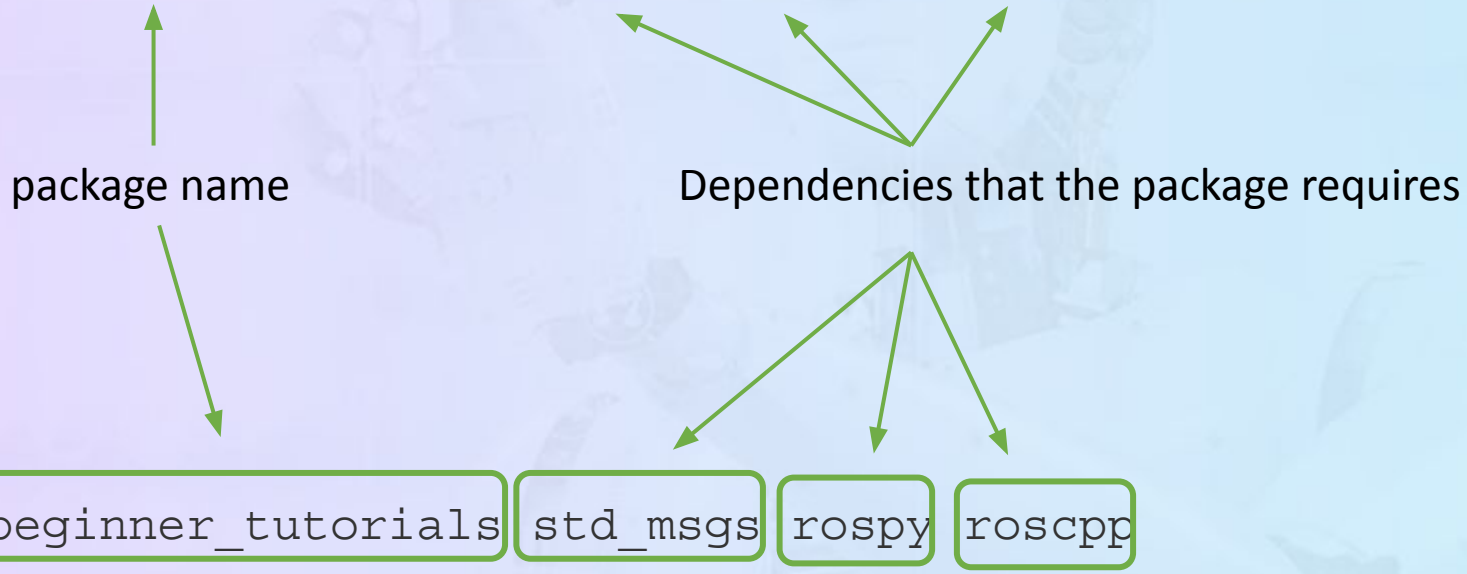
- `source devel/setup.bash`
- `echo $ROS_PACKAGE_PATH`

Sourcing the Workspace and printing the ROS_PACKAGE_PATH

INSAT AEROBOTIX

Create Package

- `cd ~/catkin_ws/src`
- `catkin_create_pkg <package_name> [depend1] [depend2] [depend3]`



Example

- `catkin_create_pkg beginner_tutorials std_msgs rospy roscpp`

Don't Forget to rebuild the Workspace each time you create a new package

- `cd ~/catkin_ws`
- `catkin_make`

Create Nodes

Steps :

1. Choose language C++ or python
2. Create folder for script « /src » for c++ and « /script » for python)
3. Write your code
4. Build you code

ROS support 2 Programming Languages

INSAT AEROBOTIX

Publisher:

Subscriber:

```cpp
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "talker");

  ros::NodeHandle n;

  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

  ros::Rate loop_rate(10);


  int count = 0;
  while (ros::ok())
  {

    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str());

    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

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


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

int main(int argc, char **argv)
{

  ros::init(argc, argv, "listener");

  ros::NodeHandle n;

  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);

  ros::spin();

  return 0;
}
```

INSAT AEROBOTIX

Publisher:

Subscriber:

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

import rospy
from std_msgs.msg import String

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

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

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

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

def listener():


    rospy.init_node('listener', anonymous=True)

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

    rospy.spin()

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

from std_msgs.msg import String

- String is the type of the message
- Std_msgs.msg package that contain String message

INSAT AEROBOTIX

Create Message:

Step 1: Create srv file

- `$ roscd beginner_tutorials`
- `$ mkdir msg`

Step 2: Create .msg file

```
string first_name
string last_name
uint8 age
uint32 score
```

Step 3: Edit package.XML file

- `<build_depend>message_generation</build_depend>`
- `<exec_depend>message_runtime</exec_depend>`

Step 4: Edit CMakeLists.txt file

```
find_package(catkin REQUIRED COMPONENTS
roscpp
rospy
std_msgs
message_generation
)


# add_message_files(
# FILES
# Message1.msg
# Message2.msg
# )
```

Create Services :

Step1: Create srv file

- `$ roscd beginner_tutorials`
- `$ mkdir srv`

Step2: Create .srv file

```
int64 A
int64 B
---
int64 Sum
```

Step3: Edit package.XML file

- `<build_depend>message_generation</build_depend>`
- `<exec_depend>message_runtime</exec_depend>`

Step4: Edit CMakeLists.txt file

```
find_package(catkin REQUIRED COMPONENTS
roscpp
rospy
std_msgs
message_generation
)


# add_service_files(
# FILES
# Service1.srv
# Service2.srv
# )
```

Server Script :

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

from beginner_tutorials.srv import AddTwoInts,AddTwoIntsResponse
import rospy

def handle_add_two_ints(req):
    print "Returning [%s + %s = %s]"%(req.a, req.b, (req.a + req.b))
    return AddTwoIntsResponse(req.a + req.b)

def add_two_ints_server():
    rospy.init_node('add_two_ints_server')
    s = rospy.Service('add_two_ints', AddTwoInts, handle_add_two_ints)
    print "Ready to add two ints."
    rospy.spin()

if __name__ == "__main__":
    add_two_ints_server()
```

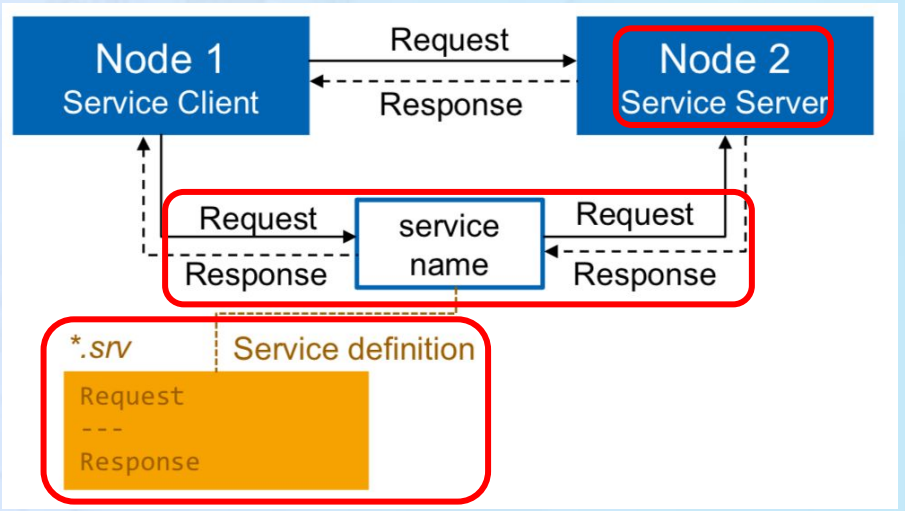
Importing code generated

Send respond

Initialize Server

Don't Forget to make your script executable:

- `$ chmod +x scripts/add_two_ints_server.py`



Ros Tools & utilities :

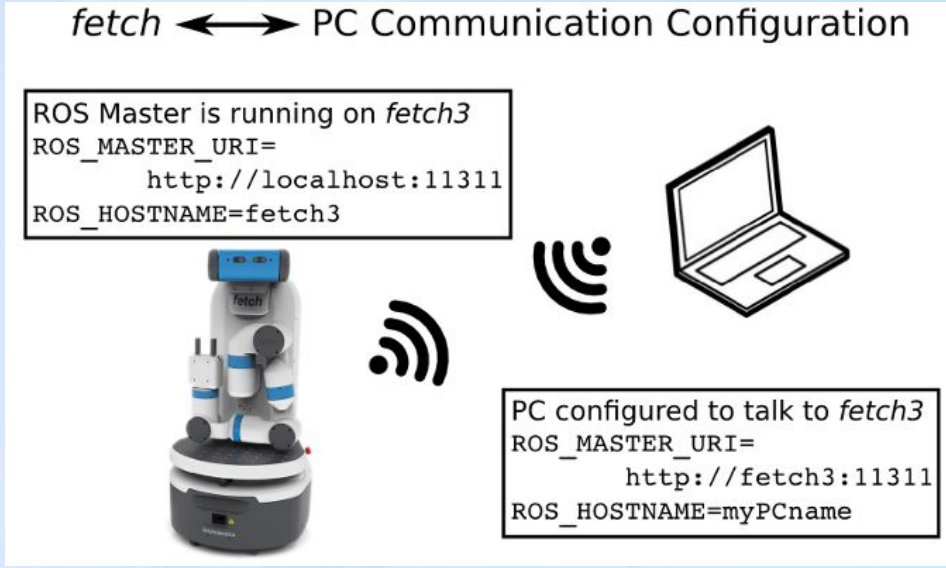
SSH:

Robot Machine

- `$ export ROS_MASTER_URI=http://localhost:11311`
- `$ export ROS_HOSTNAME=10.0.0.2`
- `$ export ROS_IP=10.0.0.2`

Workstation Machine

- `$ export ROS_IP=10.0.0.6`
- `$ export ROS_HOSTNAME=10.0.0.6`
- `$ export ROS_MASTER_URI=http://10.0.0.2:11311`



Launch files:

The problem :

We saw that every node run on a separate terminal
So imagine that we have 10 nodes to run

➡ So we must open 10 terminal and execute nodes one by one

So here came our live saver ***: LAUNCH FILE***

What's a launch file

An XML document with .launch extension in which we specify :

- Which nodes to execute
- Their parameters
- What others launch files

➡ The launch file is executed with **roslaunch** command

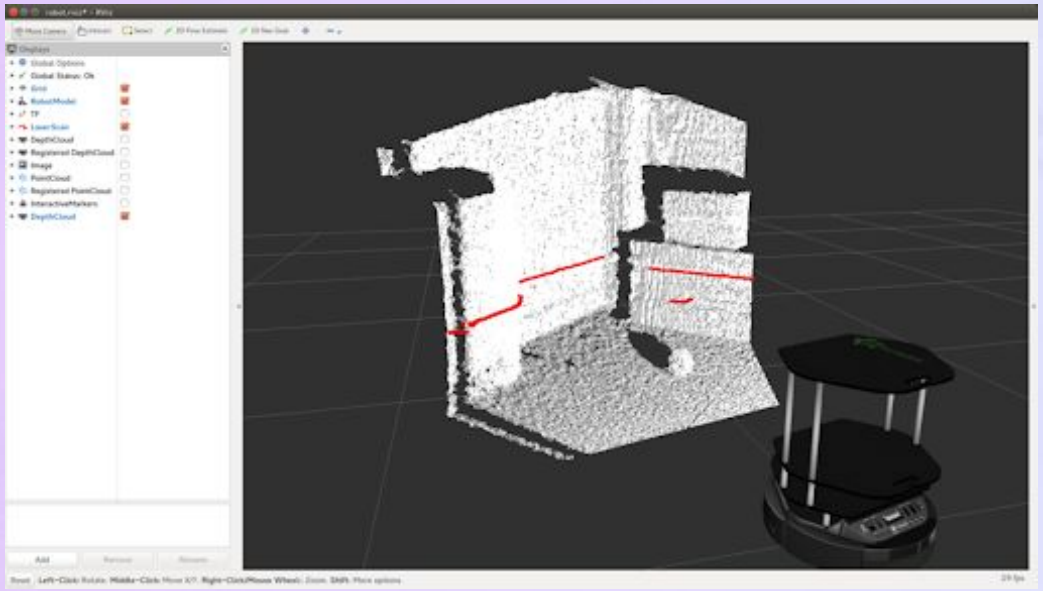
```
<launch>
  <!-- ros_args.launch -->
  <arg name="foo" default="true" doc="I pity the foo'."/>
  <arg name="bar" doc="Someone walks into this."/>
  <arg name="baz" default="false"/>
  <arg name="nop"/>
  <arg name="fix" value="true"/>

  <node pkg="pakage_name1" type="node_name1" name="rename_node"/>
  <node pkg="pakage_name2" type="node_name2" name="rename_node"/>
  <node pkg="pakage_name3" type="node_name2" name="rename_node"/>

</launch>
```

INSAT AEROBOTIX

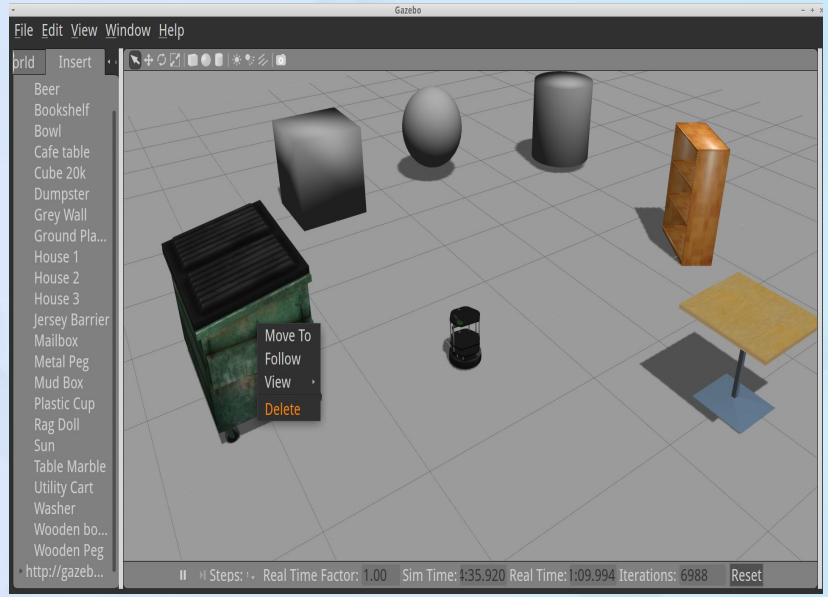
Robot Modeling :

To simulate robot we could use 2 tools:

RVIZ :



GAZEBO:



How we do it ? :

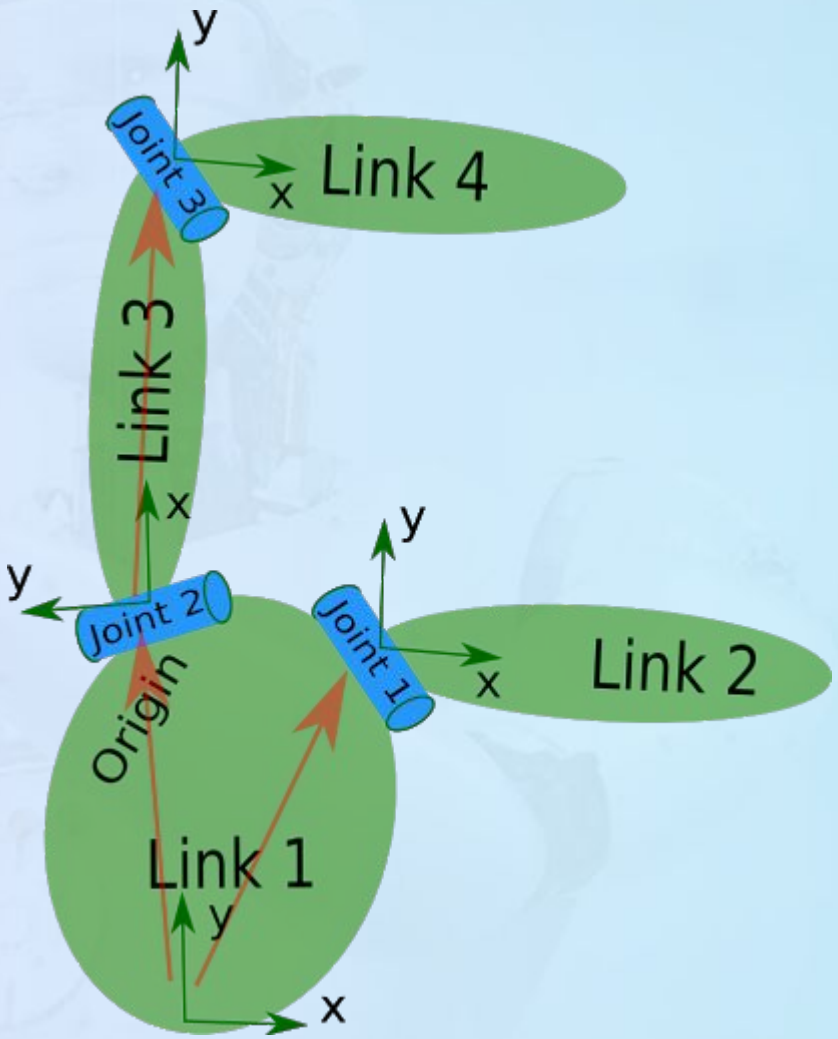
- First Steps is writing URDF file for the ROBOT

INSAT AEROBOTIX

What URDF does for us ?:

URDF: “The Unified Robot Description Format” is a representation of the robot in a format close to XML to describe a global model of the robot in the ROS system, which can be used by different programs.

the robot is defined as a set of “**link**” and “**joint**” as the following figure indicates:



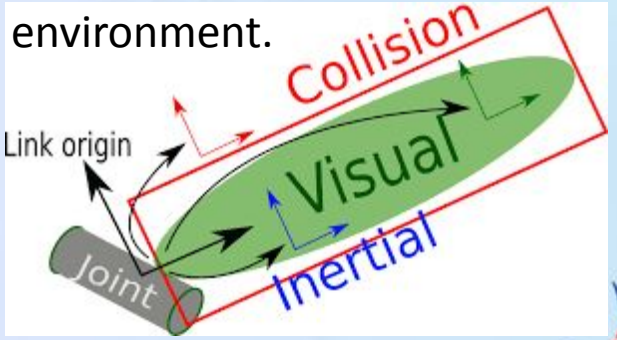
Link :

```
<link name="my_link">

  <inertial>
    <origin xyz="0 0 0.5" rpy="0 0 0"/>
    <mass value="1"/>
    <inertia ixx="100"  ixy="0"  ixz="0" iyy="100" iyz="0" izz="100" />
  </inertial>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <box size="1 1 1" />
    </geometry>
    <material name="Cyan">
      <color rgba="0 1.0 1.0 1.0"/>
    </material>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="1" length="0.5"/>
    </geometry>
  </collision>

</link>
```

1. Inertial part: to define the inertia of the part to be added.

2. Visual part: to define the shape of the part to add (circle, cylinder, etc…) or you can put a 3D file in .STL form

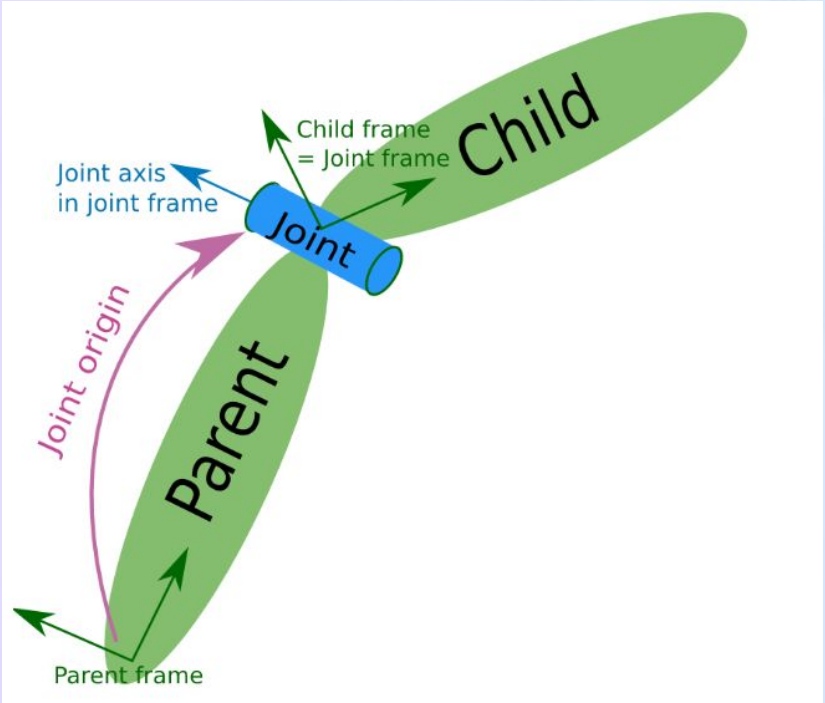
3. Collision part: to define the form of collision of the part to be added (circle, cylinder, etc.) in order to simulate the collision of the part within the environment.



INSAT AEROBOTIX

Joint:

```
<joint name="my_joint" type="floating">
   <origin xyz="0 0 1" rpy="0 0 3.1416"/>
   <parent link="link1"/>
   <child link="link2"/>

   <calibration rising="0.0"/>
   <dynamics damping="0.0" friction="0.0"/>
   <limit effort="30" velocity="1.0" lower="-2.2" upper="0.7" />
   <safety_controller k_velocity="10" k_position="15" soft_lower_limit="-2.0" soft_upper_limit="0.5" />
</joint>
```



Launching RVIZ:

```
<launch>

<arg name="model" default="$(find urdf_tutorial)/urdf/01-myfirst.urdf"/>
<arg name="gui" default="true" />
<arg name="rvizconfig" default="$(find urdf_tutorial)/rviz/urdf.rviz" />

<param name="robot_description" command="$(find xacro)/xacro $(arg model)" />

<node if="$(arg gui)" name="joint_state_publisher" pkg="joint_state_publisher_gui" type="joint_state_publisher_gui" />
<node unless="$(arg gui)" name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />
<node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)" required="true" />

</launch>
```

Launched Nodes:

- RVIZ
- Joint_state_publisher
- Robot_state_publisher

Arguments :

- Model
- gui
- Robot_description


Thank You
For Your
Attention
INSAT
AEROBOTIX
STAY
HOME

Thank you
for your attention

INSAT AEROBOTIX

STAY HOME