ROS tutorial for DRC-Hubo Driving

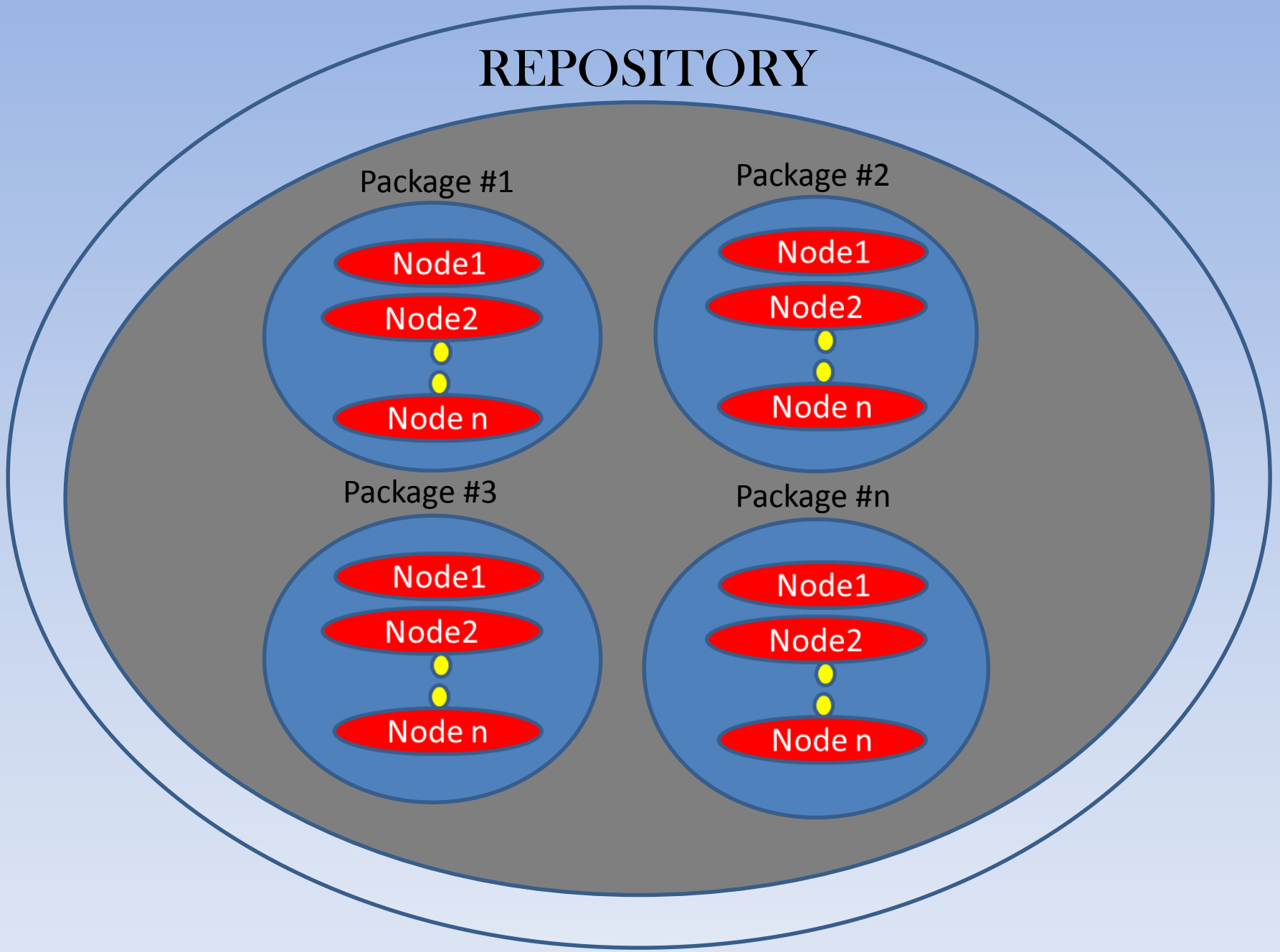
Karthikeyan Yuvaraj, MS EE

Note

- The ROS code developed for the driving purpose of DRC are c++ based and supported by ROS-Groovy.
- The slides general overview of what ROS is and gets into details relevant to ROS nodes involved in Driving task of DRC-Hubo team.
- The purpose of the nodes are to receive the encoder value(from the drc-hubo) and play it on a virtual model. Hence providing a means to monitor the robot.
- Here is the link to access those nodes.
- For more detailed tutorials and other materials on ROS, please click here.
- Also I would suggest you to read this paper to gain strong basic knowledge in ROS.

ROS

Formal definition:
"*ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license".*


REPOSITORY
Package #1
Node1
Node2
Node n
Package #2
Node1
Node2
Node n
Package #3
Node1
Node2
Node n
Package #n
Node1
Node2
Node n

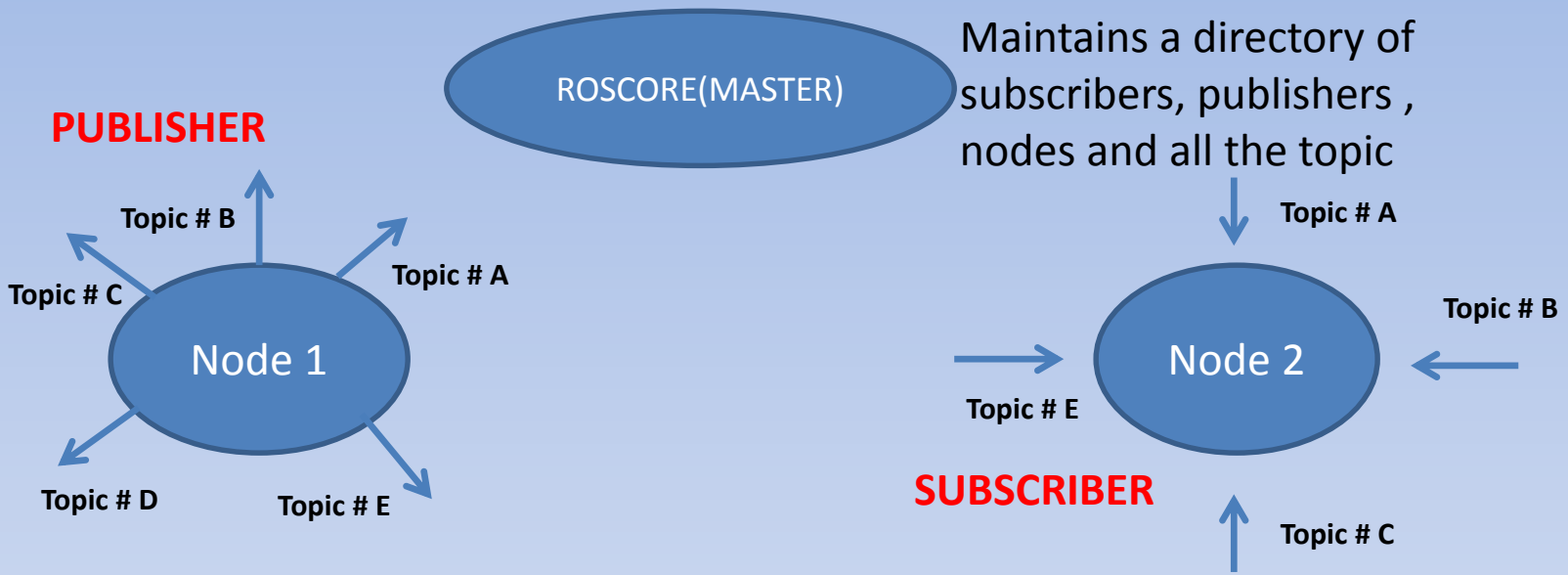
Nodes

- Executable which performs actual computation.
- Smallest possible elements of a ROS program.
- In DRC,
  - Node 1 was tilting the Hokuyo LIDAR
  - Node 2 was acquiring data from the Hokuyo LIDAR and plotting it in RVIZ
  - Node 3 was sending UDP packets to hubo
  - Node 4 was listening for UDP packets from hubo
- A node can be written in Python or C++

Node to node communication

- Nodes communicate by passing messages between them
- A message could be an integer, array of integers, float/array of float values or a combination of them(like string, float, double, timestamps).
- There are prebuilt messages like Standard messages(floats, string, integers, the basic ones), Sensor messages(Point cloud, range, IMU, laser scan, image. Etc), Geometry messages(quaternion, Odometry, velocity. Etc).
- Node communication happens on TCP/IP protocol by default.

Skeleton of a ROS Program



C++, C or Python

An example

Publisher node

Node_1

Sensor msgs/Joint state msgs

Current Network

UDP Packets
(LWP%LWY%LSR….)

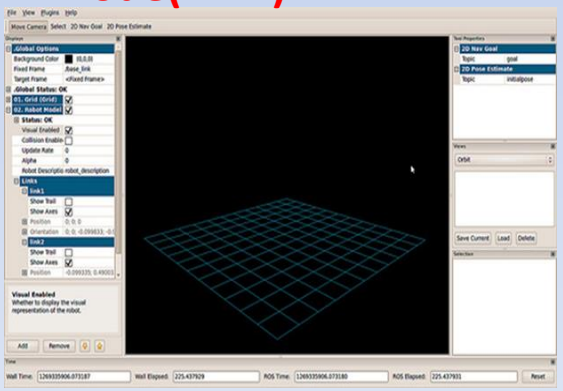
/joint_states(name given to the message)

Publisher/Subscriber node

Subscriber node(RVIZ)

Robot state Publisher(a ROS program)

Assigns the joint values to the URDF in the form of topics



Subscribing to a topic-1

- Syntax for subscribing to a topic (**c++ based**):

ros::NodeHandle <name_node>;

ros::Subcriber <subscriber_name> = <name_node>.subscriber("<topic_name>", <rate_name>, <callback_function_name>);

Example:

ros::NodeHandle n;

ros::Subscriber sub=n.subscribe("hubo_encoder",100,hubo_callback);

Subscribing to a topic-2

- Callback functions:

1) The callback functions receives the information of the corresponding topic subscribed.(It is a like an interrupt call to a function, when a topic is being published)

2) Each topic requires a separate callback function.

3)The syntax of a callback function changes according to type of topic subscribed.

Subscribing to a topic-3

- Syntax for a callback function:

```
void function_name(const <message_type_class_name>::sub_class_name::ConstPtr&
<pointer_name>)
{
//accessing the data structure name
<pointer_name>-><datastructure_name_1>;
<pointer_name>-><datastructure_name_2>;


.
.
.
}
```

Subscribing to a topic-4

Example-1:

```
int main(int argc, char** argv)
{
ros::NodeHandle n;
ros::Subscriber sub=n.subscribe("hubo_encoder",100,hubo_callback);
..}
void hubo_callback(const hubo_joint_push::hubo_msg::ConstPtr& hubo)
{
hubo->enco_d[0];
hubo->enco_d[1];
.
.
}
```

Subscribing to a topic-5

- Example-2:

```
int main(int argc, char** argv)
{
ros::NodeHandle n;
ros::Subscriber scan_sub_=n.subscribe<sensor_msgs::LaserScan> (“/scan”, 100, scancallback);
..}
void scancallback(const sensor_msgs::ConstPtr& scan)
{
Laser_value_1=scan->ranges[0];
Laser_value_1=scan->ranges[1];
}
```

For More Info on c++ based subscriber/publisher

- Click here.

Basic ROS Commands

- Getting into ROS Workspace:
- $ roscd
- $ cd ..
- $ cd src
- This takes you to workspace for ROS programs.

All programs developed for ROS usage should be done in this directory only.

Basic ROS Commands

- Running a ROS node

1) $ roscore (must be always started incase you are starting a node)

2) $ rosrun <package_name> <node_name>

$ rosrun rviz rviz (Note: rviz itself is a package)

- What is a launch file?

A launch file comprises the name of a collection of ROS nodes.

This is useful to run several nodes in a single command

- Running a launch file:

$ roslaunch <package_name> <launch_file_name>

(Note: One does not need to run roscore while starting a launch file)

$ roslaunch hubo_state_publisher hubo_state.launch

Basic ROS Commands

- To see the current active nodes:

$ rosnode list

- To see the current active topics:

$ rostopic list

- To see if a topic has any messages:

$ rostopic echo <topic_name>

$ rostopic echo \hubo_encoder

- To check the bandwidth taken by a topic

$ rostopic bw <topic_name>

$ rostopic bw \hubo_encoder

Commands to run the robot monitoring program in rviz for DRC-Hubo

- In the rainbow side(Also please check whether the IP address is directed to the destination computer):

  1)Turn on UDP socket(Button name: “UDP ON”)

  2)Click “On Visual” button.

- (Open up a new terminal and enter)

$ roslaunch hubo_state_pubisher hubo_state.launch

- In rviz (which loads up as you execute the launch file)

1)Click the fixed frame tab in the upper right corner and choose “\body_torso”.

2)Click the “add” panel(in the lower left) and select “robotmodel”. Hopefully DRC-Hubo should show up.

3)Click the “add” panel again and choose the “tf” icon. This gives transformation between links on the robot.

ROSBAG-1

What?:

- ROSBAG is a handy tool in ros to record sensor data(which are in the form of ROS topics) like Pointclouds(from depth sensors and LIDARS), image(from monocular to stereo cameras), encoder, IMU data of the robot.

Why?:

- This allows you to work on the sensor data anywhere! without the necessity of the sensor.

ROSBAG-2

How?:

Method_1

- $ rosbag record topic_1 topic_2.. topic_n

Examples:

1) $ rosbag record /tf /joint_states

(**Note**: Always record “tf”! When you record “join_states”)

2) $ rosbag record /sonar_x /sonar_y /image_raw /depth_points

(recording sensor data)

ROSBAG-3

Method_2(I prefer this method, since you can give a name to the rosbag file, which will easier for future access)

- $ rosbag record –o name_given topic_1 topic_2

Examples:

1) $ rosbag record –o driving_trial_1 /tf /joint_states /ft_hand /ft_leg

2) $ rosbag record –o image_and_pointclouds_test_1 /image_raw /depth_points

ROSBAG-4

- Playing a ROS bag file:

$ rosbag play <filename>

- Example: (**Note**: Always run roscore when you run a bag file)

$ rosbag play driving_trial_1(wait for it to start playing)

$ rostopic list (view the available topics)

- For more info on rosbag click here.

Visualizing the robot state with sample data

1. $ roslaunch hubo_state_publisher hubo_state.launch
2. $ roscd
3. $ cd ..
4. $ cd src
5. $ cd rosbag_encoder
6. $ rosbag play <filename>(hit space bar after 3-4 seconds, hence pausing the topic published)
7. In rviz(upper left corner), in fixed frame choose “Body_torso” and in lower left corner, click the “add “ button.
8. In the “add” window choose “robotmodel” icon. This will load up the hubo.
9. To view the tf links between the joints of the robot, click the “add” button again and choose “tf” icon.