AAU SUMMER SCHOOL

PROGRAMMING SOCIAL ROBOTS FOR HUMAN INTERACTION

INTRODUCTION TO ROS (ROBOT OPERATING SYSTEM)
What is ROS (Robot Operating System)?
RIP (ROS Installation Party 😊)
Key Elements:
- Package
- Node
- Message
- Topic
- Service
- Rosrun and Roslaunch
Simple example (We demonstrate)
Master/client configuration
Getting your hands dirty
Questions
WHAT IS ROS?

• First released in 2010
• Open-source 😊
• Supports Python and C++
• Middleware that handles communication between OS (e.g., Ubuntu) and the application
• Robotic systems are complex, but everything needs to communicate in (near) real-time
• Modular – interfaces are forced to be nice 😊
• Towards OS-independency (currently only Ubuntu stable)
KEY ELEMENTS - PACKAGE

• A software module that fulfills certain requirements
• Often a functionality or application, e.g., face recognition, sound source localization
• A package consists of:
  • Src: cpp source files (e.g., Different algorithms/methods)
  • Scripts: python scripts (e.g., Different algorithms/methods)
  • Msg: definitions of custom messages
    • Ex: containing probabilities for the known faces
  • Launch: launch files
    • Nice when several algorithms are available
  • Srv: definitions of custom services
KEY ELEMENTS - NODE

“A node really isn’t much more than an executable file within a ros package” – ros.org

- Uniquely identified by their given name, e.g., Face_Recognition
- Nodes can communicate with each other
  - Nodes can publish data (msg) on topics (face recognition results)
  - Nodes can acquire data (msg) by subscribing to topics (getting image to perform face recognition on)
- Useful command line tool: rosnodes <option>
  - list: list the current nodes
  - info <node_name>: prints information
KEY ELEMENTS - MESSAGE

- Description of data being published
- Std_msgs contains all the basics: Bool, Float, Integer etc. -> so no need to define these
- Possible to nest custom messages, e.g., define a message for face recognition results to contain many messages of for each known face
- Good practice to use Header as first field (for debugging and logging)
- Can be used across packages
- Example (Face Detection)

**Face_Msg.msg:**
- Header header
- Int32 x_coor
- Int32 y_coor
- Int32 width
- Int32 height
- Float32 angle

**Face_Array_Msg.msg:**
- Header header
- Int32 N_faces
- Face_Msg[] faces
KEY ELEMENTS – TOPICS

“Topics are named buses over which nodes exchange messages”
- ros.org

- Uni-directional
- Typed, i.e. only nodes that matches the type can receive message
- “Nodes are not aware of who they are communicating with”
- Useful command line tool: rostopic <option>
  - list: lists the current topics
  - info <topic_name>: shows information about a specific topic
KEY ELEMENTS – SERVICE

• Publish/Subscribe model is one-way communication, thus it does not allow for request/reply
• ... that is where services come into the picture
• It is often obvious which of the two communication models to use
  • Topic: continuous data flow, e.g., sensor data
  • Service: on-demand specific tasks, e.g., move 1 meter forward (we want to know if this happened or not)
• Typed, like topics
• Example (Rotate/turn robot):

  Rotation.srv:
  Float32 angle
  ___
  Bool succes
KEY ELEMENTS – ROSRUN/ROSLAUNCH

• So how to run a node?
• Rosrun: `rosrun <package_name> <node_name> (_param:=value)`
  • Starts a single node

• Roslaunch: `roslaunch <package_name> <launch_file>`
  • Can start many node (running a specific application might require many nodes to run)

```xml
<launch>
  <machine name="Robot1" address="192.168.1.150" env-loader="/home/sociobot/catkin_ws/src/env.sh"/>
  <machine name="Robot2" address="192.168.1.192" env-loader="/home/sociobot/catkin_ws/src/env.sh"/>

  <node ns="Robot2" name="usb_cam_node" pkg="usb_cam" type="usb_cam_node" machine="Robot2">
    <param name="image_width" value="1920"/>
    <param name="image_height" value="1080"/>
    <param name="video_device" value="/dev/HDCamera"/>
  </node>
</launch>
```
SIMPLE EXAMPLE – SOUND SOURCE LOCALIZATION

- DEMO -
Download and install VirtualBox and Extension Pack
  • [https://www.virtualbox.org/wiki/Downloads](https://www.virtualbox.org/wiki/Downloads)
  • [http://download.virtualbox.org/virtualbox/5.0.0/Oracle_VM_VirtualBox_Extension_Pack-5.0.0-101573.vbox-extpack](http://download.virtualbox.org/virtualbox/5.0.0/Oracle_VM_VirtualBox_Extension_Pack-5.0.0-101573.vbox-extpack)

Install virtual machine
  • Run the file ‘RobotSummer.ova’
  • Please select Reinitialize the MAC address of all network cards

User: sociobot Password: sociobot
Distributed processing could be a desire:
  - Multiple robots
  - Low-resource robots -> computation on server
  - …
Ros allows for this in an elegant way

Create launch file and use machine tag

```xml
<launch>
  <machine name="Robot1" address="192.168.1.150" env-loader="/home/sociobot/catkin_ws/src/env.sh"/>
  <machine name="Robot2" address="192.168.1.192" env-loader="/home/sociobot/catkin_ws/src/env.sh"/>

  <node ns="Robot2" name="usb_cam_node" pkg="usb_cam" type="usb_cam_node" machine="Robot2"/>
</node>
</launch>
```
GETTING YOUR HANDS DIRTY

Complete the following steps in VM
• Connect to “ROBOT_SUMMER_5G” OR “ROBOT_SUMMER_2.4G”
  • Password: isociobot
• Open terminal
• Open /home/sociobot/catkin_ws/src/summer_env.sh
• Set “ROS_IP” and “ROS_CLIENT” to match your device
• Open
  /home/sociobot/catkin_ws/src/robot_summer/launch/start.launch
• Set “client” ip-address to match your device
• Open new terminal window
• Run:
  Roslaunch robot_summer start.launch

• What do you see (if any)??
HOW TO MAKE PACKAGE FOR YOUR PROJECT

• We cannot go through all the code and how to interface -> don’t hesitate to ask! 😊

• Demo