Robot Programming with Lisp

6. Introduction to ROS

Gayane Kazhoyan

Institute for Artificial Intelligence
Universität Bremen

18th November, 2014
Outline

Theory

- What is a Robot?
- ROS Overview
- ROS Communication Layer
- ROS Build System
- Programming with ROS

Organizational
Outline

Theory

What is a Robot?
- ROS Overview
- ROS Communication Layer
- ROS Build System
- Programming with ROS

Organizational
Industrial Robots

Logistics

• Extremely heavy, precise and dangerous, not really smart
• Mostly no sensors, only high-precision motor encoders
• Programmable through PLCs (using block diagrams or Pascal / Basic like languages)

Automotive

Image courtesy: BIBA

Image courtesy: Mercedes Benz Bremen

Gayane Kazhoyan
18th November, 2014
Industrial Light-weight Robots

Production: [Image of industrial robot]

Medicine: [Image of medical robot]

Automotive: [Image of automotive robot]

- Very precise, moderately dangerous, somewhat smart
- High-precision motor encoders, sometimes force sensors, cameras
- Native programming and simulation tools (C++, Java, Python, GUIs)
Service Robots

- Autonomous aircrafts
- Mobile platforms
- Manipulation platforms
- Humanoids

- Usually not very precise
- Not really dangerous
- Usually cognition-enabled
- Equipped with lots of sensors
- Usually running a Linux

Gayane Kazhoyan
18th November, 2014
Service Robots with Light-weight Arms

- Moderately precise and dangerous
- Cognition-enabled
- Equipped with lots of sensors
- Usually running a combination of a real-time and non real-time OS.

DLR Justin

TUM Rosie

Courtesy of DLR

Theory

Gayane Kazhoyan
18th November, 2014

Organizational

Robot Programming with Lisp
Outline

Theory
What is a Robot?
ROS Overview
ROS Communication Layer
ROS Build System
Programming with ROS

Organizational
Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with.

Requirements:
Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- Idea: provide a unified software framework for everyone to work with.

Requirements:
- Support for different programming languages
Motivation

- Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.
- Each lab reinventing the wheel for their platforms.
- **Idea**: provide a unified software framework for everyone to work with.

  **Requirements:**
  - Support for different programming languages
  - Different operating systems
Motivation

• Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.

• Each lab reinventing the wheel for their platforms.

• Idea: provide a unified software framework for everyone to work with.

Requirements:

– Support for different programming languages
– Different operating systems
– Distributed processing over multiple computers / robots
Motivation

• Numerous different robotics labs, each with their own robot platforms, different operating systems and programming languages but similar software and hardware modules for most of them.

• Each lab reinventing the wheel for their platforms.

• Idea: provide a unified software framework for everyone to work with.

Requirements:

– Support for different programming languages
– Different operating systems
– Distributed processing over multiple computers / robots
– Easy software sharing mechanisms
At 2007 Willow Garage, a company founded by an early Google employee Scott Hassan at 2006 in the Silicon Valley, starts working on their Personal Robotics project and ROS.
Robot Operating System [2]

ROS core components:

- Meta-Operating System for programming robotics software (configuring, starting / stopping, logging etc. software components)
- Middleware for communication of the components of a robotic system (distributed inter-process / inter-machine communication)
- A collection of packaging / build system tools with a strong focus on integration and documentation
- Language-independent architecture (C++, Python, Lisp, Java, JavaScript, ...)

ROS core software developed and maintained by OSRF and some externals.
In addition, developed by the ROS community:

- hardware drivers
- libraries (PCL, OpenCV, TF, ...)
- capabilities (navigation, manipulation, control, ...)
- applications (fetching beer, making popcorn, ...)
ROS Community

From the community report July 2014:

wiki.ros.org visitor locations:

Source: Google Analytics
Site: wiki.ros.org in August
ROS Community [2]

Some robots supporting ROS (data from November 2014):

- Fraunhofer IPA Care-O-bot
- Aldebaran Naö
- Willow Garage PR2
- Merlin mrabotPro
- Clearpath Robotics Hasky
- Gostai Jazz
- Neobotix mpo-700
- Neobotix mpo-500
- ROS-Industrial
- Robotnik Summit
- Adept Mobile Robots (Pioneer family, P3DX, P3AT, PioneerBot, PowerBot, AdeptBot, Patroller, GuaBot)
- Adept Mobile Robots Seekur family (Seekur, Seekur Jr.)
- Adept Mobile Robots Pioneer LX
- Cytos-Gamma
- Robonaut 2
- Otto Bock SensorHand Speed
- Fiesta Didactic Rebotina
- Robotnik Agus
- Robotnik Arm
- PAL Robotics REEM-C
- Denso VS000
- Kinova JACO
- Lili
- Nav2
- Kandoa Nostage / Hiro
- Dr. Robot Jaguar
- Allegro Hand
- SimLab
- RBBM
- TuLip
- Robotnik Guardian
- Shadow Hand
- Riddlerbot
- Robotnik SX
- Robotnik SUMMITXL
- AMIGO
- AMIKO
- Actic Rex Quadrant
- CoroWare Corobot
- CippedRobin
- WheeledRobin
- KemoDo
Theory

What is a Robot?
ROS Overview
ROS Communication Layer
ROS Build System
Programming with ROS

Organizational
Robotic software components

- Knowledge Base
- Planning System
- Camera
- Navigation System
- Motor Drivers
- Controllers
- Image Processing

AI PC
Robot PCs
Vision PC

Gayane Kazhoyan
18th November, 2014
Robotic software components

→ Processes distributed all over the place.

Gayane Kazhoyan
18th November, 2014
Connecting Pieces Together

- Knowledge Base
- Planning System
- Camera
- Navigation System
- Motor Drivers
- Controllers
- Image Processing

Theory

Gayane Kazhoyan
18th November, 2014
roscore

- ROS master
  - A centralized XML-RPC server
  - Negotiates communication connections
  - Registers and looks up names of participant components
- Parameter Server
  - Stores persistent configuration parameters and other arbitrary data
- rosout
  - Distributed stdout
Distributed Hosts

Robot

Network Bus (TCP/IP)

Host B
- Image Processing

Host A
- Knowledge Base
- Planning System

Controllers
Camera
Motor Drivers
Navigation System

droscore
Terminology

• **Nodes** are processes that produce and consume data
• **Parameters** are persistent data stored on parameter server, e.g. configuration and initialization settings

Node communication means:

• **Topics**: asynchronous many-to-many “streams-like”
  - Strongly-typed (ROS .msg spec)
  - Can have one or more *publishers*
  - Can have one or more *subscribers*

• **Services**: synchronous blocking one-to-many “function-call-like”
  - Strongly-typed (ROS .srv spec)
  - Can have only one *server*
  - Can have one or more *clients*

• **Actions**: asynchronous non-blocking one-to-many “function-call-like”
  - Built on top of topics but can be canceled
Establishing Communication

```
ros "master"
camera
viewer
```
establishing communication

advertise("images")

- `camera`
- `ros "master"`
- `viewer`
Establishing Communication

```
ros
"master"

topic:images

camera    viewer
```
Establishing Communication

```
ros "master"
```

topic:images

subscribe("images")

camera

viewer
Establishing Communication

```
subscribe("images")
ros "master"
topic:images

camera

viewer

```
Establishing Communication

```
ros "master"
topic:images
```

```
camera ─── images(tcp) ─── viewer
```
Establishing Communication

```
ros
"master"
topic:images

images(tcp)
camera
publish(img)

viewer
```
Establishing Communication

```
ros "master"
topic:images
```

- camera
- viewer
- images(tcp)
- publish(img)
Establishing Communication

```
ros
"master"
topic:images
```

**camera**

images(tcp)

**viewer**

publish(img)

**viewer_too**
Establishing Communication

```
ros "master"

subscribe("images")

camera
```

publish(img)

```
images(tcp)

viewer

viewer_too
```
Establishing Communication

```
ros
"master"
topic:images
```

```
camera
publish(img)
images(tcp)
```

```
viewer
images(tcp)
viewer_too
```
Establishing Communication [2]
Establishing Communication [2]

- Knowledge Base
- Planning System
- Navigation System
- Controllers
- Image Processing
- Motor Drivers
- Direct Connection
- Camera
- roscore

Diagram:

- Camera
- Direct Connection
- Motor Drivers
- Image Processing
- roscore
- Planning System
- Navigation System
- Controllers
Establishing Communication [2]
Artificial Intelligence

ROS Graph

- Starting the core:
  
  
  $ roscore$

- Starting a node:
  
  $ rosrun turtlesim turtlesim_node$

- Starting another node:
  
  $ rosrun turtlesim turtle_teleop_key$

- Examining the ROS Graph:
  
  $ rosrun turtlesim turtle_teleop_key$

Theory Organizational

Gayane Kazhoyan
18th November, 2014

Robot Programming with Lisp

42
Tools

• **rosnode**: gives the user information about a node
  
  $ rosnode -help
  cleanup, info, kill, list, machine, ping

• **rostopic**: gives publishers, subscribes to the topic, datarate, the actual data
  bw, echo, find, hz, info, list, pub, type

• **rosservice**: enables a user to call a ROS Service from the command line
  call, find, list, type, uri

• **rosmsg**: gives information about message types
  list, md5, package, packages, show

• **rossrv**: same as above for service types
  list, md5, package, packages, show

• **roswtf**: diagnoses problems with a ROS network
Launch Files
Automated Starting, Stopping and Configuring the Nodes

XML files for launching nodes:

- automatically set parameters and start nodes with a single file
- hierarchically compose collections of launch files
- automatically re-spawn nodes if they crash
- change node names, namespaces, topics, and other resource names
- without recompiling
- easily distribute nodes across multiple machines
Launch Files [2]
Automated Starting, Stopping and Configuring the Nodes

Example

```xml
<launch>
  <!-- Starting nodes-->
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
  <node pkg="turtlesim" type="turtle_teleop_key" name="teleop"
        output="screen"/>

  <!-- Setting parameters -->
  <param name="some_value" type="double" value="2.0"/>
</launch>
```

Using the launch file:

```
$ roslaunch package_name launch_file_name
```

Gayane Kazhoyan
18th November, 2014
Outline

Theory
- What is a Robot?
- ROS Overview
- ROS Communication Layer
- ROS Build System
- Programming with ROS

Organizational
Packages and Metapackages

- **Packages** are a named collection of software that is built and treated as an atomic dependency in the ROS build system.
- **Metapackages** are dummy “virtual” packages that reference one or more related packages which are loosely grouped together.

Similar to Debian packages.
Actually released through the Debian packaging system.
ROS Workspace

Packages are stored in ROS workspaces:

```
$ roscd
```

Workspaces have a specific structure

- build
- devel
- install
- src
  - CMakeLists.txt
  - rosc_package_1
  - metapack_repo_1
    - metapackage_1
    - rosc_package_2
    - rosc_package_3
  - CMakeLists.txt
  - package.xml
  - asdf-system.asd
  - src

Theory Organizational

Gayane Kazhoyan
18th November, 2014

Robot Programming with Lisp 48
Managing Packages

• Creating a package:
  
  $ ros cd && cd src/lisp_course_material
  
  $ catkin_create_pkg assignment_6 roslisp turtlesim geometry_msgs

• Compiling a package:
  
  $ ros cd && catkin_make

• Moving through ROS workspaces:
  
  $ ros cd assignment_6

Naming convention: underscores (no CamelCase, no-dashes)!

All the packages in your workspace are one huge CMake project.

→ Multiple workspaces chained together.
<?xml version="1.0"?>
<package>
  <name>assignment_6</name>
  <version>0.0.0</version>
  <description>The assignment_6 package</description>
  <maintainer email="kazhoyan@cs.uni-bremen.de">Gaya</maintainer>
  <license>Public domain</license>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>geometry_msgs</build_depend>
  <build_depend>roslisp</build_depend>
  <build_depend>turtlesim</build_depend>
  <run_depend>geometry_msgs</run_depend>
  <run_depend>roslisp</run_depend>
  <run_depend>turtlesim</run_depend>
</package>
cmake_minimum_required(VERSION 2.8.3)
project(assignment_6)
find_package(catkin REQUIRED COMPONENTS
  roslisp
  geometry_msgs
)
catkin_package(
  CATKIN_DEPENDS roslisp geometry_msgs
)
Outline

Theory

What is a Robot?
ROS Overview
ROS Communication Layer
ROS Build System
Programming with ROS

Organizational
ROS API

ROS API provides the programmer with means to

- start ROS node processes
- generate messages
- publish and subscribe to topics
- start service servers
- send service requests
- provide and query action services
- find ROS packages
- ...

ROS APIs: roscpp, rospython, rosjava, rosjs, roslisp
Links

- ROS documentation
  http://wiki.ros.org/

- ROS community support
  http://answers.ros.org/questions/
Outline

Theory
- What is a Robot?
- ROS Overview
- ROS Communication Layer
- ROS Build System
- Programming with ROS

Organizational
Organizational Info

- Assignment: roslisp tutorial not graded
- Next class: 25.11, 14:15, TAB 1.58
Thanks for your attention!

Special thanks to Lorenz Mösenlechner and Jan Winkler for providing illustrations!