Robotics Programming Laboratory

Bertrand Meyer
Jiwon Shin

Lecture 2: ROS and Roboscooop
Robots of today

- Many sensors and actuators
- Able to operate in familiar or expected environments
- Able to perform specialized tasks
Robots of the future

C-3PO
- Provides etiquette, customs, and translation assistance
- Has own thoughts and feelings

R2-D2
- Rescues people and robots
- Repairs other robots and complex hardware and software

Advanced robots must be able to operate and perform tasks in complex, unknown environments.

As robotics advances, we must be aware that robots can be both helpful and harmful.
Concurrency in robotics

Advanced robotic systems have many hardware components that can operate concurrently.

- Sensors and actuators can run in parallel.
- Locomotion and manipulators can run concurrently.
Concurrency in robotics
Multiprocessing, parallelism

- **Multiprocessing**: the use of more than one processing unit in a system
- **Parallel execution**: processes running at the same time
Multitasking, concurrency

- **Interleaving**: several tasks active, running one at a time
- **Multitasking**: the OS runs interleaved executions
- **Concurrency**: multiprocessing and/or multitasking

P1: Go to goal  
P2: Avoid obstacle
Concurrency

Benefits of introducing concurrency into programs:

- **Efficiency**: time (load sharing), cost (resource sharing)
- **Availability**: multiple access
- **Convenience**: perform several tasks at once
- **Modeling power**: describe systems that are inherently parallel
Concurrency framework for robotics
<table>
<thead>
<tr>
<th>Roboscoop</th>
<th>SCOOP</th>
<th>ROS</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Library (set of primitives and tools for their coordination)</td>
<td>• O-O Structure</td>
<td>• Communication</td>
</tr>
<tr>
<td>• Integration with other robotics frameworks</td>
<td>• Coordination</td>
<td>• Navigation, image processing, coordinate transforms, visualization, ...</td>
</tr>
<tr>
<td>• External calls</td>
<td>• Concurrency</td>
<td></td>
</tr>
</tbody>
</table>
ROS: Robot Operating System

**ROS**: Open-source, meta-operating system for robots

ROS provides the services of an operating system, including

- hardware abstraction,
- low-level device control,
- implementation of commonly-used functionality,
- message-passing between processes, and
- package management


http://www.ros.org
Goals of ROS

- Support code reuse in robotics research and development.
- Enable executables to be individually designed and loosely coupled at runtime through its distributed framework of processes.
- Group processes for easy sharing and distribution.
- Enable the distribution of collaboration through its repositories.

Properties of ROS

- Thin
- Peer-to-Peer
- Multi-lingual: C++, Python, Lisp
ROS node

Node

- A process that performs computation
- Interchangeable with a software module
- Can generate data for and receive data from other nodes

A system is typically comprised of many nodes: robot control node, localization node, path planning node, perception node, etc.

Benefits of using nodes

- Fault-tolerance: crashes are isolated to individual nodes
- Reduction of code complexity
ROS topic

**Topic**

- Named bus over which nodes exchange messages
- Has anonymous publish/subscribe semantics.

*A node can publish and/or subscribe to multiple topics.*

*A topic supports multiple publishers and subscribers.*

![Diagram of ROS topic with Object Detection, UI, Viewer, Path Planner, and Goal Pose (Topic) nodes connected with Publication and Subscription arrows.](image-url)
**ROS message**

**Message**: Strictly typed data structure used for communication between nodes

**Message description specification**

- Build-in types
- Names of Messages defined on their own
- Fixed- or variable-length arrays:
- Header type: `std_msgs/Header`:
  - `uint32 seq, time stamp, string frame_id`
- Constants

Messages can be arbitrarily nested structures and arrays.

```
int16 x
uint32 y
sensor_msgs/LaserScan s
uint8[] data
float32[10] a
Header header
int32 z=123
string s=foo
```
**common_msgs**

- Messages that are widely used by other ROS packages
- Provide a shared dependency to multiple stacks, eliminating a circular dependency

**Types of common_msgs**

- *geometry_msgs*: Point, Pose, Transform, Vector, Quaternion, etc.
- *nav_msgs*: MapMetaData, Odometry, Path, etc.
- *sensor_msgs*: LaserScan, PointCloud, Range, etc.
ROS service

Service: A pair of strictly typed messages for synchronous transactions

Service description specification

- Request messages
- Response messages

Two messages are concatenated together with a ‘---’.

A service cannot be embedded inside another service.

Only one node can advertise a service of any particular name.

Node Service Invocation Response Node

int16 x
uint32 y
string s
**Master**

- Provides naming and registration services to nodes
- Tracks publishers and subscribers to topics and services
- Enables individual nodes to locate one another
Master and Slave APIs

- Manage information about the availability of topics and services
- Negotiate the connection transport
- **XML-RPC** (remote procedure call protocol using XML)
  - HTTP-based protocol
  - Stateless
  - Lightweight
  - Available in many programming languages

**Return value format**

- Status code: -1 (ERROR), 0 (FAILURE), 1 (SUCCESS)
- Status message: human readable string
- Value: defined by individual API calls
Master API

Register/unregister methods

- registerService(caller_id, service, service_URI, caller_URI)
- unregisterService(caller_id, service, service_URI)
- registerSubscriber(caller_id, topic, topic_type, caller_URI)
- unregisterSubscriber(caller_id, topic, caller_URI)
- registerPublisher(caller_id, topic, topic_type, caller_URI)
- unregisterPublisher(caller_id, topic, caller_URI)

Name service and system state

- lookupNode(caller_id, node_name)
- lookupService(caller_id, service)
- getPublishedTopics(caller_id, subgraph)
- getTopicTypes(caller_id)
- getSystemState(caller_id)
- getUri(caller_id)
Slave API

Receive callbacks from the Master
- `publisherUpdate(caller_id, topic, publishers)`
- `paramUpdate(caller_id, parameter_key, parameter_value)`

Negotiate connections with other nodes
- `requestTopic(caller_id, topic, protocols)`
- `shutdown(caller_id, msg='')`

System state
- `getBusStats(caller_id)`
- `getBusInfo(caller_id)`
- `getMasterUri(caller_id)`
- `getPid(caller_id)`
- `getSubscriptions(caller_id)`
- `getPublications(caller_id)`
**ROS topic connection**

**Subscriber**

- `registerSubscriber(subscriber_ID, topic, topic_type, subscriber_URI)`

- `requestTopic(subscriber_ID, topic, [[Protocol1Name, P1Param1, P1Pram2, ...], [Protocol2Name, ...], ...])`

**Master**

- `[1, “no subscriber”, []]`

- `[1, “publisher”, [[publisher1_URI, publisher2_URI]]]`

**Publisher**

- `registerPublisher(publisher_ID, topic, topic_type, publisher_URI)`

- `[1, “initialize communication”, [[Protocol1Name, P1Param1, P1Pram2, ...]]]`

**Acknowledgement**

**Connection Header**

**Data**

- `- XMLRPC - TCPROS`

- `Request` — ` Reply`
ROS topic connection

**Subscriber**

```
registerSubscriber(
    subscriber_ID, topic, topic_type, subscriber_URI)
```

```
[1, “received”, -]
```

```
requestTopic(
    subscriber_ID, topic, [[Protocol1Name, P1Param1, P1Param2, ...], [Protocol2Name, ...], ...])
```

**Master**

```
[1, “no publisher”, []]
```

```
[1, “subscriber”, [subscriber1_URI, subscriber2_URI, ...]]
```

```
publisherUpdate(
    publisher_ID, topic, [publisher1_URI, publisher2_URI, ...])
```

**Publisher**

```
registerPublisher(
    publisher_ID, topic, topic_type, publisher_URI)
```

```
[1, “initialize communication”, [Protocol1Name, P1Param1, P1Pram2, ...]]
```

**Data**

- XMLRPC - UDPROS

**Request**

- Request

- Reply
**ROS topic connection example**

```python
Master

registerPublisher("camera", "image", "sensor_msgs/Image", "pub:123")

[1, "initialize communication", [TCPROS, "pub:234"]]

requestTopic("image_viewer", "image", [[TCPROS, "sub:567"]])

[1, "no subscriber", []]

[1, "camera", [pub:123]]

registerSubscriber("image_viewer", "image", "sensor_msgs/Image", "sub:456")

Camera

Image Viewer

Image data message

- XMLRPC - TCPROS
**TCPROS**

- Provides a simple, reliable communication stream
- TCP packets always arrive in order
- Lost packets are resent until they arrive.

**UDPROS**

- Packets can be lost, contain errors, or be duplicated.
- Is useful when multiple subscribers are grouped on a single subnet
- Is useful when latency is more important than reliability, e.g., teleoperation, audio streaming
- Suited for a lossy WiFi or cell modem connection.
ROS service connection

Service

```
registerService(
    service_ID, service, 
    service_TCP_URI, 
    service_URI)
```

Master

```
[1, "registered", -]
```

```
[1, "service", 
    service_TCP_URI]
```

Client

```
lookupService(
    client_ID, service)
```

Acknowledgement

Connection Header

Response

Request - XMLRPC - TCPOROS

Initiate Connection

Acknowledgement

Request - Reply
ROS package

Package

- A software unit with useful functionality
- Aims to provide enough functionality to be useful but still lightweight and reusable in other software.
- Can contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, etc.

Useful packages for the class

TF: coordinate transformation

RViz: 3D visualization
TF: Coordinate Transformation

```c++
static tf::TransformBroadcaster br;
tf::Transform transform;
transform.setOrigin( tf::Vector3(x, y, 0.0) );
transform.setRotation( tf::Quaternion(theta, 0, 0) );
br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "world", "robot1"));
```
ROS coordinate frame conventions

Axis orientation

- **x**: forward, **y**: left, **z**: up

Rotation representation

- Quaternion: \( x, y, z, w \)
  - **Compact representation**
  - No singularities
- Rotation matrix
  - No singularities
- **roll**: \( x \), **pitch**: \( y \), **yaw**: \( z \)
  - No ambiguity in order
  - Used for angular velocities
# Standard SI units

<table>
<thead>
<tr>
<th>Base Units</th>
<th>Derived Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quantity</strong></td>
<td><strong>Unit</strong></td>
</tr>
<tr>
<td>Length</td>
<td>Meter</td>
</tr>
<tr>
<td>Mass</td>
<td>Kilogram</td>
</tr>
<tr>
<td>Time</td>
<td>Second</td>
</tr>
<tr>
<td>Current</td>
<td>Ampere</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Build system: CMake

Build system

- A software tool for automating program compilation, testing, etc.
- Maps a set of source code (files) to a target (executable program, library, generated script, exported interface)
- Must fully understand the build dependencies

CMake

- Cross-platform build system
- Controls the build process using a CMakeLists.txt file
- Creates native makefile in the target environment

```bash
cmake_minimum_required(VERSION 2.8.3)
project(ProjectName)
add_executable(ExecutableName file.cpp)
```
ROS build system: catkin

catkin

- Official build system of ROS
- CMake with some custom CMake macros and Python scripts
- Supports for automatic 'find package' infrastructure and building multiple, dependent projects at the same time
- Simplifies the build process of ROS’s large, complex, and highly heterogeneous code ecosystem

Advantages of using catkin

- Portability through Python and pure CMake
- Independent of ROS and usable on non-ROS projects
- Out-of-source builds: can build targets to any folder

http://wiki.ros.org/catkin/Tutorials
Dependency management: package.xml

```
<package>
  <name>foo</name>
  <version>1.2.3</version>
  <description>
    This package provides foo capability.
  </description>
  <maintainer email="me@ethz.ch">Me</maintainer>
  <license>BSD</license>
  <url>http://www.ethz.ch/foo</url>
  <author>Me</author>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <run_depend>roscpp</run_depend>
  <test_depend>python-mock</test_depend>
</package>
```

Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Package's build system tools
Dependency management: CMakeLists.txt

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmake_minimum_required(VERSION 2.8.3)</td>
<td>Minimum Cmake version</td>
</tr>
<tr>
<td>project(foo)</td>
<td>Project name</td>
</tr>
<tr>
<td>find_package(catkin REQUIRED COMPONENTS roscpp)</td>
<td>Dependent packages</td>
</tr>
<tr>
<td>catkin_package(</td>
<td>Installs package.xml and generates code for find_package</td>
</tr>
<tr>
<td>INCLUDE_DIRS include</td>
<td>Include paths for the package</td>
</tr>
<tr>
<td>LIBRARIES ${PROJECT_NAME}</td>
<td>Exported libraries from the project</td>
</tr>
<tr>
<td>CATKIN_DEPENDS roscpp</td>
<td>Other catkin projects this project depends on</td>
</tr>
<tr>
<td>DEPENDS opencv</td>
<td>Non-catkin CMake projects this project depends on</td>
</tr>
<tr>
<td>include_directories(include ${catkin_INCLUDE_DIRS})</td>
<td>Location of header files</td>
</tr>
<tr>
<td>add_executable(foo src/foo.cpp)</td>
<td>An executable target to be built</td>
</tr>
<tr>
<td>add_library(moo src/moo.cpp)</td>
<td>Libraries to be built</td>
</tr>
<tr>
<td>target_link_libraries(foo moo)</td>
<td>Libraries the executable target links against</td>
</tr>
</tbody>
</table>

http://wiki.ros.org/catkin/CMakeLists.txt
## Roboscoop software architecture

<table>
<thead>
<tr>
<th>Roboscoop</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Library (set of primitives and tools for their coordination)</td>
<td></td>
</tr>
<tr>
<td>• Integration with other robotics frameworks</td>
<td></td>
</tr>
<tr>
<td>• External calls</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SCOOP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• O-O Structure</td>
<td></td>
</tr>
<tr>
<td>• Coordination</td>
<td></td>
</tr>
<tr>
<td>• Concurrency</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Communication</td>
<td></td>
</tr>
<tr>
<td>• Navigation, image processing, coordinate transforms, visualization, ...</td>
<td></td>
</tr>
</tbody>
</table>
SCOOP: a brief introduction

Simple Concurrent Object Oriented Programming

- Easy parallelization
- One more keyword in Eiffel (separate)
- Natural addition to O-O framework
- Retains natural modes of reasoning about programs
- Coordination is easy to express: close correspondence with behavioral specification\(^1\)

---

Object and processor architecture

- **PRIMITIVE_BEHAVIOR**
- **ROBOT_CONTROL**
- **STOP_SIGNALER**
- **VELOCITY_PUBLISHER**
- **ROBOT_STATE_SIGNALER**

Connections:
- **a** from **PRIMITIVE_BEHAVIOR** to **ROBOT_CONTROL**
- **b** from **PRIMITIVE_BEHAVIOR** to **ROBOT_CONTROL**
- **stop** from **ROBOT_CONTROL** to **STOP_SIGNALER**
- **vel** from **ROBOT_CONTROL** to **VELOCITY_PUBLISHER**
- **robot** from **ROBOT_CONTROL** to **ROBOT_STATE_SIGNALER**
To go straight, to avoid obstacles ...

Get the state of the robot

- Location and orientation
- Linear and angular velocity
- Sensory information

Control the velocity

Stop if there is a request for stopping (e.g., emergency stop)

\[ \text{separate: objects are potentially on a different processor} \]

\[ \text{r: separate ROBOT\_STATE\_SIGNALER} \]
\[ \text{v: separate VELOCITY\_PUBLISHER} \]
\[ \text{s: separate STOP\_SIGNALER} \]

P1: Go straight  P2: Avoid obstacle

Obstacle
separate calls

feature

robot: separate ROBOT_STATE_SIGNALER  -- Current robot's state
vel: separate VELOCITY_PUBLISHER  -- Control robot's velocity
stop: separate STOP_SIGNALALER  -- Whether stop requested

start  -- Start the control

local

a, b: separate PRIMITIVE_BEHAVIOR

do

create a.make_with_attributes (robot, vel, stop)
create b.make_with_attributes (robot, vel, stop)

start_robot_behaviors (a, b)

end

start_robot_behaviors (a, b: separate PRIMITIVE_BEHAVIOR)
do

a.avoid_obstacle_repeatedly
b.go_straight_repeatedly

end
Synchronization through preconditions

```plaintext
go_straight (a_robot: separate ROBOT_STATE_SIGNALER;
    a_vel: separate VELOCITY_PUBLISHER;
    a_stop: separate STOP_SIGNALER)
    -- Move robot unless stopped or an obstacle observed.

require
  (not a_robot.is_moving and not a_robot.has_obstacle)
or a_stop.is_stop_requested

do
  if a_stop.is_stop_requested then
    a_vel.send_stop
  else
    a_vel.send_velocity (0.03, 0.0) -- 3cm/sec, no spinning
  end
end
```
How do we cancel all processors?

STOP_SIGNALER

is_stop_requested: BOOLEAN
set_stop (val: BOOLEAN)

GO 直线 (BEHAVIOR 1)

AVOID_OBSTACLE (BEHAVIOR 2)

APPLICATION

stop.is_stop_requested

stop.set_stop(FALSE)

stop.is_stop_requested
Roboscoop

Coordination layer above SCOOP

Three-layer architecture

Synchronization: wait conditions

Interoperability through ROS (external calls)
Roboscoop repository structure

roboscoop_app

- application.e
- _cpp

roboscoop_lib

- controller
- sequencer
- sensor
- common
- utils
- signaler
- ros
- msgs
- communication

roboscoop_ros

- msg
- src
Communication with ROS nodes: publication

**THYMIO_SOUND_PUBLISHER**

**inherit**

**ASEBA_PUBLISHER**

**SOUND_PUBLISHER**

**Topic name:**
/aseba/events/sound_cmd

**Message type:**
asebaros/AsebaEvent

**time stamp**
**uint16** source
**int16[]** data
Communication with ROS nodes: subscription

Edit `topics.xml` (in `roboscoop_app/_cpp/callbacks_gen`)

Defines topics that the application subscribes to

Generate C++ subscriber (run script)

Write your custom subscriber class in Eiffel

Create an object of the generated class and pass it to your subscriber
Communication with ROS nodes: application

subscriber: THYMIO_SUBSCRIBER

some_feature
local
  sub_name: C_STRING
  gen_subscriber_ptr: POINTER
  do
    create sub_name.make ("pregenerated_subscriber")
    gen_subscriber_ptr := c_new_gen_subscriber (1, sub_name.item)
  end
create subscriber.make_with_ptr (gen_subscriber)
end

c_new_gen_subscriber (a_id: INTEGER; a_c_name: POINTER): POINTER
external
  "C++ inline use %"gen_subscriber.h%"
alias
  "return new GenSubscriber($a_id, $a_c_name);"
end