Advanced Techniques for Mobile Robotics

ROS

Wolfram Burgard, Cyrill Stachniss,
Kai Arras, Maren Bennewitz
Today’s Lecture

- **ROS** – Robot Operating System
- **PR2** – Humanoid Robot
- **TF** – Transforms
- **PCL** – Point Clouds
- **ROS Applications**
ROS Overview Talk

- High Level Overview
  - What is ROS?
  - Who made it?
  - Why do we use it?
- Middleware Aspects
- Software Structure
- Client Libraries
High Level Overview

What is ROS?
- An Operating System like Windows, GNU/Linux?
- A Software Distribution?
- A Middleware?
- A Set of Libraries?

![Diagram showing ROS architecture with Ubuntu and Hardware layers, IPC connections between applications, and ROS middleware in the middle.]
High Level Overview

Who made ROS?

- Privatly Owned Company
- Based in Menlo Park, California
- Hardware: PR2, Texai
- Software: ROS (OpenCV, Player, PCL)
- Strong Open Source Commitment
High Level Overview

Why do we use ROS?

- Great functionality
  - Middleware (this session)
  - Development tools (2\textsuperscript{nd} Session)
  - Advanced libraries (3\textsuperscript{rd} Session)
  - Hardware drivers

- Large scientific community
  - A lot of state-of-the-art software available
  - Easy to exchange/integrate/build-upon existing projects
  - Open source (mostly BSD)
  - Actively developed by full-time staff

- To stop reinventing the wheel...
How Robotics Research Keeps...

First, someone publishes...

...a paper with a proof-of-concept robot.

This prompts another lab to try to build on this result...

...but they can't get any details on the software used to make it work...

...and all the code used by previous lab members is a mess.

But inevitably, time runs out...

...and countless sleepless nights are spent writing code from scratch.

So, a grandiose plan is formed to write a new software API...

...and they write code that barely works but lets them publish...
Robots Using ROS
ROS - a Middleware for Robots

“Middleware is a software that connects software components or applications.”

- Framework for interprocess communication
- Boosts modularization
- Enables transparent distribution of software in a network
ROS - a Middleware for Robots

Example: Communication on the Internet

- DNS Request
- DNS Server
- DNS Reply (Host IP)
- HTTP Request
- Web Browser
- Web Server
- HTTP Reply
- HTML
- DNS Information
ROS - a Middleware for Robots

Interprocess Communication Using ROS

Viewer Node

ROS Master

Hokuyo Node

Subscribe Topic

Advertise Topic

Host+Port

Topic Request

Topic Stream

LaserScan
ROS - a Middleware for Robots

Interprocess Communication Using ROS

- Viewer Node
- ROS Master
- Hokuyo Node
- Logging Node
- ROS Parameter Server

- Configuration
- Advertise Data
- Data Location
- Data
ROS - a Middleware for Robots
ROS - a Middleware for Robots
ROS - a Middleware for Robots

There is more to it...

- (De-)Serialization under the hood
- Central multi-level logging facilities
- Service calls, preemptible actions
- Central time
- Dynamic reconfiguration
ROS Overview

- High Level Overview
  - What is ROS?
  - Who made it?
  - Why do we use it?
- Middleware Aspects
- Software Structure
- Client Libraries
ROS – Software Distribution

Software Hierarchy

- **Release** – collection of stacks and packages
- **Stacks** – a full application suite
- **Package** – software (and interface definition) to solve a specific task
- **Node** – An executable with some useful functionality
- **Message/Service/Action** – Interface definitions
ROS – Software Distribution

Boxturtle
March 2010

C-Turtle
August 2010

Diamondback
March 2011

Releases

Electric Emys
November 2011
## ROS – Software Distribution

- **packages**: ~3150
- **repositories**: ~270
- **stacks**: ~500

### Stack Overview

<table>
<thead>
<tr>
<th>Name</th>
<th>Packages</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2dmapping_pr2_app</td>
<td>1</td>
<td>A 2D mapping application for the PR2 robot platform.</td>
</tr>
<tr>
<td>2dnav_pr2_app</td>
<td>1</td>
<td>A 2D navigation application for the PR2 robot platform.</td>
</tr>
<tr>
<td>arm_navigation</td>
<td>4</td>
<td>arm_navigation</td>
</tr>
<tr>
<td>articulation</td>
<td>5</td>
<td>Kinematic models for articulated objects (cabinet doors fitting, model selection and visualization).</td>
</tr>
<tr>
<td>art_vehicle</td>
<td>10</td>
<td>ART autonomous vehicle support</td>
</tr>
<tr>
<td>asctec_drivers</td>
<td>6</td>
<td>asctec_drivers</td>
</tr>
<tr>
<td>au_automow_common</td>
<td>5</td>
<td>Auburn University Autonomous Lawnmower - Common</td>
</tr>
<tr>
<td>au_automow_drivers</td>
<td>3</td>
<td>au_automow_drivers</td>
</tr>
<tr>
<td>au_automow_simulation</td>
<td>0</td>
<td>au_automow_simulation</td>
</tr>
<tr>
<td>bag_experimental</td>
<td>0</td>
<td>bag_experimental</td>
</tr>
<tr>
<td>billiards</td>
<td>17</td>
<td>billiards</td>
</tr>
</tbody>
</table>
ROS – Software Distribution

- Binary distribution via debian package management
- Source distribution via version control systems
- Tools for dependency resolution
  - Fetch *binaries* from apt-repository
  - Fetch *source code* from version control
  - Recursively *build* dependencies
- Central documentation wiki: ros.org/wiki

Installation instructions & download mirror:
http://ros.informatik.uni-freiburg.de
ROS – Client Libraries

- What makes a program a ROS node?
  - → Usage of a client library
    - C++ and Python
    - Octave/Matlab, Lisp, Java and Lua (experimental)
    - You can implement your own
  - A client library embeds a program in the ROS interprocess communication network (i.e. attaches it to the middleware)
There is more to it...

- Tools for analysis, debugging
- and **visualization** of IPC
- Live message view
- Recording and playback
- Many more...
Today’s Lecture

- ROS – Robot Operating System
- **PR2** – Humanoid Robot
- TF – Transforms
- PCL – Point Clouds
- ROS Applications
Sensors
Joints

- r_wrist_roll_joint
- r_forearm_roll_joint
- r_upper_arm_roll_joint
- head_pan_joint
- head_tilt_joint
- laser_tilt_joint
- r_shoulder_lift_joint
- l_shoulder_pan_joint
- l_elbow_flex_joint
- l_wrist_flex_joint
- l_gripper_joint
PC Hardware

- 2x Onboard servers
  - Processors: Two Quad-Core i7 Xeon
  - Memory: 24 GB
- Internal hard drive: 500 GB
- Removable hard drive: 1.5 TB
PR2 Simulation
PR2 Simulation

- Motivation:
  - Regression testing
  - Visualization
  - Design, optimization
  - Multiple developers, single robot

- Based on Open Source project Gazebo
- Uses Open Dynamics and Physics Engine
- Uses Ogre for rendering
PR2 Simulation

- Requirements:
  - Graphics card with 3D acceleration (Nvidia, ATI)
  - Linux supported driver
  - Core2Duo processor
  - > 2GB Ram
Simulation Fidelity

- **Manipulation:**
  - Simulated real-time PR2 etherCAT node communicates actuator states (motor efforts and position)

- **Perception:**
  - Simulated sensor nodes for cameras, lasers and imu
    - Stream data as well as services provided
  - Self collision checks are disabled
  - Anti-gravity arms
  - No laser scan duration
Learning to Set a Table

- Observe example scenes of breakfast tables
- Learn a hierarchical scene model
  - Level 1: physical objects (segmented point clouds)
  - Level 2: covers (constellations of physical objects)
  - Level 3: table scenes (constellations of covers)
- Sample from the model to generate new scenes
Learning to Clean a Table

- Uncertainty about how the dirt looks like
- Idea: “Dirt is that what we can remove”

1. Cluster image into color classes
2. Observe table state
3. Try wiping each class
4. Updating belief about the expected
5. Clean the entire table
6. Observe again
Learning to Clean a Table

Jürgen Hess, Jürgen Sturm, Wolfram Burgard

Autonomous Intelligent System Lab,
University of Freiburg, Germany
Student Projects

- **Portrait Bot:**
  - Recognize faces in camera images
  - Extract edges
  - Draw edge image onto a whiteboard
  - Available in May

- **Two-handed cleaning with a broom:**
  - Objective: adapting coverage plan to using a broom (e.g. different poses/pattern for sweeping in corners or under a table)
Student Project: Portrait Bot
Student Project: Sweeping
Today’s Lecture

- ROS – Robot Operating System
- PR2 – Humanoid Robot
- **TF – Transforms**
- PCL – Point Clouds
- ROS Applications
Motivation

- Multiple sensors and actuators on robots
- Robot and sensors not static
- Multi-robot cooperation

Where is this object relative to my gripper? Where is my arm in the world?
What is tf?

A coordinate frame tracking system

- Standardized protocol for publishing transform data to a distributed system
- Helper classes and methods for:
  - Publishing coordinate frame data: `TransformBroadcaster`
  - Collecting transform data and using it to manipulate data: `Transformer, TransformListener, tf::MessageFilter, ...`
- Currently: Only tree(s) of transformations, but any robot(s) / sensor layout
- Conversion functions, mathematical operations on 3D poses
tf is Distributed!

- Two types of tf nodes: Publishers & Listeners
  - Listeners: Listen to /tf and cache all data heard up to cache limit (10 sec default)
  - Publishers: Publish transforms between coordinate frames on /tf
- No central source of tf information, or history before a node was started
Debugging Tools

- **Command Line Tools**
  - `tf_echo`: Print a specific transform to the screen
  - `tf_monitor`: Display statistics about transforms
  - `roswtf`: Debug common tf configuration errors

- **Visualizations**
  - Rviz tf plugin
  - `view_frames`
Coordinate Frames on the PR2

- Coordinate frames for every link and sensor of the robot
- Each sensor publishes data in its own coordinate frame
Common Setup for Mobile Robots

- $\text{odom} \rightarrow \text{base}_\text{link}$ provided by robot odometry
- $\text{map} \rightarrow \text{odom}$ provided by localization method (e.g. amcl in ROS)
  - How to transform $\text{odom}$ link in map frame so that $\text{base}_\text{link}$ is "correct"?
Sources of Transformations

- URDF file defines robot “body layout”
  - Joints, links, sensor positions, visualization

- tf for all robot links automatically published
- Static transforms published from command line
- In your own node (e.g. localization)
Today’s Lecture

- ROS – Robot Operating System
- PR2 – Humanoid Robot
- TF – Transforms
- PCL – Point Clouds
- ROS Applications
What are Point Clouds

- Point Cloud = a “cloud” (i.e., collection) of $nD$ points (usually $n = 3$)
- $p_i = \{x_i, y_i, z_i\} \rightarrow P = \{p_1, p_2, \ldots, p_i, \ldots, p_n\}$
- used to represent 3D information about the world
What are Point Clouds

- besides XYZ data, each point can hold additional information
- examples include: RGB colors, intensity values, distances, segmentation results, etc...
What are Point Clouds
What are Point Clouds
Where do they come from?

- Laser scans (high quality)
- Stereo cameras (passive & fast but dependent on texture)
- Time of flight cameras (fast but not as accurate/robust)
- Kinect-Style Sensors
- Simulation
- ...
Tilting Laser Scanner

Current laser beams and resulting point cloud
Tilting Scanner on Moving Robot

Press "r" to reset the point cloud
For what!?

- Spatial information of the environment has many important applications
  - Navigation / Obstacle avoidance
  - Object recognition
  - Grasping
  - ...

Obstacle & Terrain Detection
Object Recognition

Point Clouds can complement and supersede images when they are ambiguous.
What is PCL?

**PCL**
- is a fully templated modern C++ library for 3D point cloud processing
- uses SSE optimizations (Eigen backend) for fast computations on modern CPUs
- uses OpenMP and Intel TBB for parallelization
- passes data between modules (e.g., algorithms) using Boost shared pointers
- will be made independent from ROS in one of the next releases
## PCL Modules

<table>
<thead>
<tr>
<th>Filters</th>
<th>Features</th>
<th>Keypoints</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Filters" /></td>
<td><img src="image2.png" alt="Features" /></td>
<td><img src="image3.png" alt="Keypoints" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Registration</th>
<th>KdTree</th>
<th>Octree</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image4.png" alt="Registration" /></td>
<td><img src="image5.png" alt="KdTree" /></td>
<td><img src="image6.png" alt="Octree" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Segmentation</th>
<th>Sample Consensus</th>
<th>Surface</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image7.png" alt="Segmentation" /></td>
<td><img src="image8.png" alt="Sample Consensus" /></td>
<td><img src="image9.png" alt="Surface" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Range Image</th>
<th>Io</th>
<th>Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image10.png" alt="Range Image" /></td>
<td><img src="image11.png" alt="Io" /></td>
<td><img src="image12.png" alt="Visualization" /></td>
</tr>
</tbody>
</table>
Data Representation

- PointCloud class (templated over the point type):
  ```cpp
  template <typename PointT>
  class PointCloud;
  ```

- Important members:
  ```cpp
  std::vector<PointT> points; // the data
  uint32_t width, height; // scan structure?
  ```
Point Types

- Examples of PointT:
  ```
  struct PointXYZ
  {
    float x;
    float y;
    float z;
  }
  ```
  or
  ```
  struct Normal
  {
    float normal[3];
    float curvature;
  }
  ```

See pcl/include/pcl/point_types.h for more examples.
PointCloud2 Message

- We distinguish between two data formats for the point clouds:
  - PointCloud<PointType> with a specific data type (for actual usage in the code)
  - PointCloud2 as a general representation containing a header defining the point cloud structure (for loading, saving or sending as a ROS message)

Conversion between the two is easy:

pcl::fromROSMsg and pcl::toROSMsg
Basic Interface

Filters, Features, Segmentation all use the same basic usage interface:

- Create the object
- use `setInputCloud` to give the input
- set some parameters
- call `compute` to get the output
Filter Example 1

```cpp
pcl::PassThrough<T> p;
p.setInputCloud (data);
p.FilterLimits (0.0, 0.5);
p.SetFilterFieldName ("z");

filter_field_name = "x"; | filter_field_name = "xz";
```
Filter Example 2

```cpp
pcl::VoxelGrid<T> p;
p.setInputCloud (data);
p.FilterLimits (0.0, 0.5);
p.SetFilterFieldName ("z");
p.setLeafSize (0.01, 0.01, 0.01);
```
Filter Example 3

```cpp
pcl::StatisticalOutlierRemoval<T> p;
p.setInputCloud (data);
p.setMeanK (50);
p.setStddevMulThresh (1.0);
```
Features Example 1

```cpp
pcl::NormalEstimation<T> p;
p.setInputCloud (data);
p.SetRadiusSearch (0.01);
```
Features Example 2

```cpp
pcl::BoundaryEstimation<T,N> p;
p.setInputCloud(data);
p.setInputNormals(normals);
p.SetRadiusSearch(0.01);
```
Features Example 3

NarfDescriptor narf_descriptor(&range_image);
narf_descriptor.getParameters().support_size = 0.3;
narf_descriptor.getParameters().rotation_invariant = false;
PointCloud<Narf36> narf_descriptors;
narf_descriptor.compute(narf_descriptors);
Segmentation Example 1

```cpp
pcl::SACSegmentation<T> p;
p.setInputCloud (data);
p.setModelType (pcl::SACMODEL_PLANE);
p.setMethodType (pcl::SAC_RANSAC);
p.setDistanceThreshold (0.01);
```
Segmentation Example 2

```cpp
cpl::EuclideanClusterExtraction<T> p;
p.setInputCloud (data);
p.setClusterTolerance (0.05);
p.setMinClusterSize (1);
```
Segmentation Example 3

pcl::SegmentDifferences<T> p;
p.setInputCloud (source);
p.setTargetCloud (target);
p.setDistanceThreshold (0.001);
Higher level example

How to extract a table plane and the objects on it?

Diagram:

- PointCloud2
- VoxelGrid
- NormalEstimation
- ProjectInliers
- SACSegmentationFromNormals (planar segmentation)
- ExtractInliers
- ConvexHull2D
- ExtractPolygonalPrismData (get all points lying on the table)
- EuclideanClusterExtraction (split the points into N object clusters)
- TablePlane
- ObjectClusters
More details

Today's Lecture

- ROS – Robot Operating System
- PR2 – Humanoid Robot
- TF – Transforms
- PCL – Point Clouds
- ROS Applications
SLAM on Dense Colored Clouds

- Our goal: SLAM systems for RGB-D sensors
  - 3D environment representation
  - 6DOF trajectory estimation
  - Online Operation
  - No dependency on further sensors (e.g. Odometry, Laser)

- New kinect-style sensors provide:
  - Dense RGB-D data
  - High framerate (30 Hz)
  - Low weight (440g)
  - Low-cost (~100€)
Approach Overview

- RGBD Sensor
  - Depth Images
  - RGB Images
    - Pairwise 6D Transformation Estimation (RANSAC)
    - Pairwise Feature Matching (SURF, SIFT,...)
  - Global Pose Graph Optimization (g2o)
  - Registered 3D Point Clouds
    - Voxelization (OctoMap)
    - 3D Occupancy Grid Map
Map Representation

- Point clouds are inefficient for applications such as collision detection and navigation
- We use the OctoMap framework
  - Octree-based data structure
  - Recursive subdivision of space into octants
  - Volumes allocated as needed

→ Smart 3D grid
OctoMap Framework

Probabilistic Update of Voxels

- Full 3D model
- Probabilistic
- Multi-resolution
- Memory efficient

Available from octomap.sf.net
Octomap with Per-Voxel Colors

Probabilistic 3D mapping using OctoMap and RGBDSLAM

Kai M. Wurm, Felix Endres
Autonomous Intelligent Systems Lab
University of Freiburg, Germany
Kinematic Models

A Probabilistic Framework for Learning Kinematic Models of Articulated Objects

Jürgen Sturm, Cyrill Stachniss,
Wolfram Burgard
University of Freiburg, Germany
Kinematic Models

- Fit and select appropriate models for observed motion trajectories of articulated objects
- Estimate structure and DOFs of articulated objects with \( n > 2 \) object parts

1-DOF closed chain

3-DOF open chain
Kinematic Models

- Marker-less perception:
  Detect and track articulated objects in depth images and learn their kinematic models

- Approach:
  - Segment planes
  - Fit pose candidates and filter
  - Learn kinematic models
A Probabilistic Framework for Learning Kinematic Models of Articulated Objects

Jürgen Sturm, Cyrill Stachniss,
Wolfram Burgard
University of Freiburg, Germany
Thank You...

...for your attention