Introduction to Robot Simulation (Gazebo)

Mayank Mittal

AE640A: Autonomous Navigation

January 10, 2018
Outline

● Recap
  ○ ROS Communication Layer
  ○ ROS Ecosystem
  ○ Libraries/Tools in ROS

● Robot Simulation
  ○ Why we need it?

● Elements within Simulation
  ○ Collision and Visual Geometries
  ○ Joints
  ○ Sensors
  ○ Lights

Loads of examples to come!
What is ROS?

- A “meta” operating system for robots
- A collection of packaging, software building tools
- An architecture for distributed interprocess/ inter-machine communication and configuration
- Development tools for system runtime and data analysis
- A language-independent architecture (c++, python, lisp, java, and more)

Slide Credit: Lorenz Mösenlechner, TU Munich
What is ROS not?

- An actual operating system
- A programming language
- A programming environment / IDE
- A hard real-time architecture
ROS Communication Layer : ROS Core

- **ROS Master**
  - Centralized Communication Server based on XML and RPC
  - Negotiates the communication connections
  - Registers and looks up names for ROS graph resources

- **Parameter Server**
  - Stores persistent configuration parameters and other arbitrary data.

- **`rosout`**
  - Network based `stdout` for human readable messages.

*Slide Credit: Lorenz Mösenlechner, TU Munich*
ROS Communication Layer: Graph Resources

- **Nodes**
  - Processes distributed over the network.
  - Serves as source and sink for the data sent over the network

- **Parameters**
  - Persistent data such as configuration and initialization settings, i.e., the data stored on the parameter server. E.g., camera configuration

- **Topics**
  - Asynchronous many-to-many communication stream

- **Services**
  - Synchronous one-to-many network based functions

Slide Credit: Lorenz Mösenlechner, TU Munich
ROS Communication Protocols: Connecting Nodes

- **ROS Topics**
  - Asynchronous “stream-like” communication
  - Strongly-typed (ROS .msg spec)
  - Can have one or more publishers
  - Can have one or more subscribers

- **ROS Services**
  - Synchronous “function-call-like” communication
  - Strongly-typed (ROS .srv spec)
  - Can have only one server
  - Can have one or more clients

- **Actions**
  - Built on top of topics
  - Long running processes
  - Cancellation

Slide Credit: Lorenz Mösenlechner, TU Munich
How to organize code in a ROS ecosystem?

ROS code is grouped at two different levels:

- **Packages:**
  - A named collection of software that is built and treated as an atomic dependency in the ROS build system.

- **Stacks:**
  - A named collection of packages for distribution.
How to organize code in a ROS ecosystem?

```
source code
header declarations
scripts
message definitions
service definitions
configuration files
launch files
metadata
...
```

```
package_n
...
package_two
package_one
```

“package”  “stack”
ROS Launch

- launch is a tool for launching multiple nodes (as well as setting parameters)
- Are written in XML as *.launch files
- If not yet running, launch automatically starts a roscore

Start a launch file from a package with

```
$ roslaunch package_name file_name.launch
```

More info:
http://wiki.ros.org/roslaunch

Slide Credit: Marco Hutter, ETH Zurich
ROS Parameter Server

- Nodes use the parameter server to store and retrieve parameters at runtime
- Best used for static data such as configuration parameters
- Parameters can be defined in launch files or separate YAML files

List all parameters with

```
$ rospack list
```

More info:
http://wiki.ros.org/rosparam
**ROS GUI Tools**

**rqt**: A QT based GUI developed for ROS

**rviz**: Powerful tool for 3D Visualization

(demo in today’s class)

More info: [http://wiki.ros.org/rqt](http://wiki.ros.org/rqt)
ROS Time

- Normally, ROS uses the PC’s system clock as time source (wall time)
- For simulations or playback of logged data, it is convenient to work with a simulated time (pause, slow-down etc.)
- To work with a simulated clock:
  - Set the `/use_sim_time` parameter
    ```
    $ rosparam set use_sim_time true
    ```
  - Publish the time on the topic `/clock` from
    - Gazebo (enabled by default)
    - ROS bag (use option --clock)
- To take advantage of the simulated time, you should always use the ROS Time APIs:
  - **ros::Time**
    ```
    ros::Time begin = ros::Time::now();
    double secs = begin.toSec();
    ```
  - **ros::Duration**
    ```
    ros::Duration duration(0.5); // 0.5s
    ```

More info:
http://wiki.ros.org/Clock

Slide Credit: Marco Hutter, ETH Zurich
ROS Bags

- A bag is a format for storing message data
- Binary format with file extension *.bag
- Suited for logging and recording datasets for later visualization and analysis

Record all topics in a bag

```bash
$ rosbag record --all
```

Record given topics

```bash
$ rosbag record topic_1 topic_2 topic_3
```

Show information about a bag

```bash
$ rosbag info bag_name.bag
```

Record given topics

```bash
$ rosbag play [options] bag_name.bag
```

- `--rate=factor` Publish rate factor
- `--clock` Publish the clock time (set param use_sim_time to true)
- `--loop` Loop playback

More info: http://wiki.ros.org/Clock

Slide Credit: Marco Hutter, ETH Zurich
Libraries/Tools available with ROS

Image Courtesy: Open Source Robotics Foundation
What are Point Clouds?

- “Cloud”/collection of $n$-D points (usually $n=3$)
- Used to represent 3D information about the world:

$$p_i = \{x_i, y_i, z_i\} \rightarrow P = \{p_1, p_2, \ldots, p_i, \ldots, p_n\}$$

Image Courtesy: Bastian Steder, University of Freiburg
What are Point Clouds?

- besides XYZ data, each point can hold additional information like RGB colors, intensity values, distances, segmentation results, etc.
What are Point Clouds?

- besides XYZ data, each point can hold additional information like RGB colors, intensity values, distances, segmentation results, etc.

Image Courtesy: Bastian Steder, University of Freiburg
What are Point Clouds?

- besides XYZ data, each point can hold additional information like RGB colors, intensity values, distances, segmentation results, etc.
What are Point Clouds?

- besides XYZ data, each point can hold additional information like RGB colors, intensity values, distances, segmentation results, etc.
How are Point Clouds collected?

- **Laser scans** (high quality)
- **Stereo cameras** (passive & fast but dependent on texture)
- **Time of flight cameras** (fast but not as accurate/robust)
- **Simulation**
How are Point Clouds useful?

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

Grasping Objects on Table
Detection of cars in Point Cloud

More info: http://wiki.ros.org/pcl
Coordinate frames

- robots consist of many \textit{links}
- every \textit{link} describes its own coordinate system
- sensor measurements are local to the corresponding \textit{link}
- \textit{links} change their position over time
Specifying the Arrangement of Devices

- All these devices are mounted on a robot in an articulated way.
- Some devices are mounted on other devices that can move.
- In order to use all the sensors/actuators together we need to describe this configuration.
  - For each “device” specify one or more frames of interest
  - Describe how these frames are located w.r.t each other

Slide Credit: Wolfram Burgard, University of Freiburg
Defining the Structure

- Each “Link” is a reference frame of a sensor
- Each “joint” defines the transformation that maps the child link in the parent link.
- ROS does not handle closed kinematic chains, thus only a “tree” structure is allowed
- The root of the tree is usually some convenient point on the mobile base (or on its footprint)
Robot Simulation

● Simulators mimic the real world, to a certain extent
  ○ Simulates robots, sensors, and objects in a 3-D dynamic environment
  ○ Generates realistic sensor feedback and physical interactions between objects

● Why use them?
  ○ Save time and your sanity
  ○ Experimentation much less destructive
  ○ Use hardware you don’t have
  ○ Create really cool videos
Simulation Architecture

Client (your program) → Player

Player → Gazebo

Player → Real Hardware

Stage → Gazebo

TCP/IP

TCP/IP

TCP/IP

TCP/IP

SHM
Simulation Architecture

Gazebo runs two processes:

- **Server**: Runs the physics loop and generates sensor data.
  - Executable: `gzserver`
  - Libraries: Physics, Sensors, Rendering, Transport

- **Client**: Provides user interaction and visualization of a simulation.
  - Executable: `gzclient`
  - Libraries: Transport, Rendering, GUI

Run Gazebo server and client separately:

```
$ gzserver
$ gzclient
```

Run Gazebo server and client simultaneously:

```
$ gazebo
```
Elements within Simulation

- **World**
  - Collection of models, lights, plugins and global properties

- **Models**
  - Collection of links, joints, sensors, and plugins

- **Links**
  - Collection of collision and visual objects

- **Collision Objects**
  - Geometry that defines a colliding surface

- **Visual Objects**
  - Geometry that defines visual representation

- **Joints**
  - Constraints between links

- **Sensors**
  - Collect, process, and output data

- **Plugins**
  - Code attached to a World, Model, Sensor, or the simulator itself
Element Hierarchy

World
  - Scene
  - Physics
  - Model
    - Link
      - Collision
      - Visual
      - Sensor
    - Plugin
      - Plugin
      - Light
World

- A world is composed of a model hierarchy
- The Gazebo server (gzserver) reads the world file to generate and populate a world
  - This file is formatted using SDF (Simulation Description format) or URDF (Unified Robot Description Format)
  - Has a “.world” extension
  - Contains all the elements in a simulation, including robots, lights, sensors, and static objects
Models

- Each model contains a few key properties:
  - **Physical presence** (optional):
    - Body: sphere, box, composite shapes
    - Kinematics: joints, velocities
    - Dynamics: mass, friction, forces
    - Appearance: color, texture
  - **Interface** (optional):
    - Control and feedback interface (libgazebo)
Element Types

- Collision and Visual Geometries
  - Simple shapes: sphere, cylinder, box, plane
  - Complex shapes: heightmaps, meshes
Element Types

- **Collision and Visual Geometries**
  - Simple shapes: sphere, cylinder, box, plane
  - Complex shapes: heightmaps, meshes

- **Joints**
  - Prismatic: 1 DOF translational
  - Revolute: 1 DOF rotational
  - Revolute2: Two revolute joints in series
  - Ball: 3 DOF rotational
  - Universal: 2 DOF rotational
  - Screw: 1 DOF translational, 1 DOF rotational
Element Types

- **Sensors**
  - Ray: produces range data
  - Camera (2D and 3D): produces image and/or depth data
  - Contact: produces collision data
  - RFID: detects RFID tags

- **Lights**
  - Point: omni-directional light source, a light bulb
  - Spot: directional cone light, a spot light
  - Directional: parallel directional light, sun

LiDAR sensor in Gazebo
How to use Gazebo to simulate your robot?

Steps:

1. load a world
2. load the description of the robot
3. spawn the robot in the world
4. publish joints states
5. publish robot states
6. run rviz
Meet Robot “Alpha”

- Two-wheeled differential drive robot
- **Sensors:**
  - Rotary Encoders
  - IMU
  - Camera
  - Kinect 360
  - Hokuyo URG-04
- **Actuator**
  - Brushed DC Motor
Meet Robot “Alpha”

- How to design and create your own robot?

Motor Driver  Micro-controller  Voltage Regulator  IMU  Switch
Real-Time Appearance-Based (RTAB) Mapping

- RGB-D Graph-Based SLAM approach based on an incremental appearance-based loop closure detector
- Can be used alone with a hand-held Kinect or stereo camera for 3D RGB-D mapping

More info:
http://wiki.ros.org/rtabmap
Rao-Blackwellized Particle Filter SLAM (GMapping)

- Uses a particle filter in which each particle carries an individual map of the environment
- Optimized for long-range laser scanners like SICK LMS or PLS scanner

More info:
https://www.openslam.org/gmapping.html
Meet Robot “Alpha”

- All source code available online, feel free to test them out and contribute!

https://github.com/Mayankm96/Phase-VII
Homework

- Install **Ubuntu 16.04** and **ROS Kinetic** on laptop
  - Software setup scripts [here](#)
- Checkout ROS Wiki and Tutorials
  - Available Packages ([http://www.ros.org/browse/list.php](http://www.ros.org/browse/list.php))
- Go through the lecture videos on ‘**Programming for Robotics**’ by ETH Zurich *(optional)*
References

- Gazebo Website (http://gazebosim.org/)