

# Introduction to a Simulation Environment – Gazebo

**Welcome**

Dr. Ahmad Kamal Nasir

# Today's Objectives

- Introduction to Gazebo
- Building a robot model in Gazebo
- Populating robot environment with simulated objects
- Writing plugins
- Sensors
- Interface with ROS

# Working with Simulators

- What
  - Mimic the real world, to a certain extent
- When
  - Always!!
- Why
  - Save time and your sanity
  - Experimentation much less destructive
  - Use hardware you don't have
  - Create really cool videos
- How
  - Someone has probably already done it, so use it

# Which Simulator?

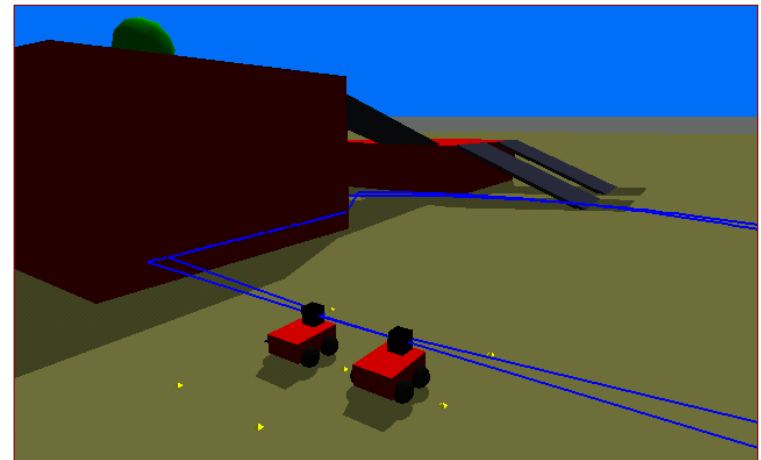
## Stage

- 2D
- Sensor-based
- Player interface
- Kinematic
- $O(1) \sim O(n)$
- Large teams (100's)



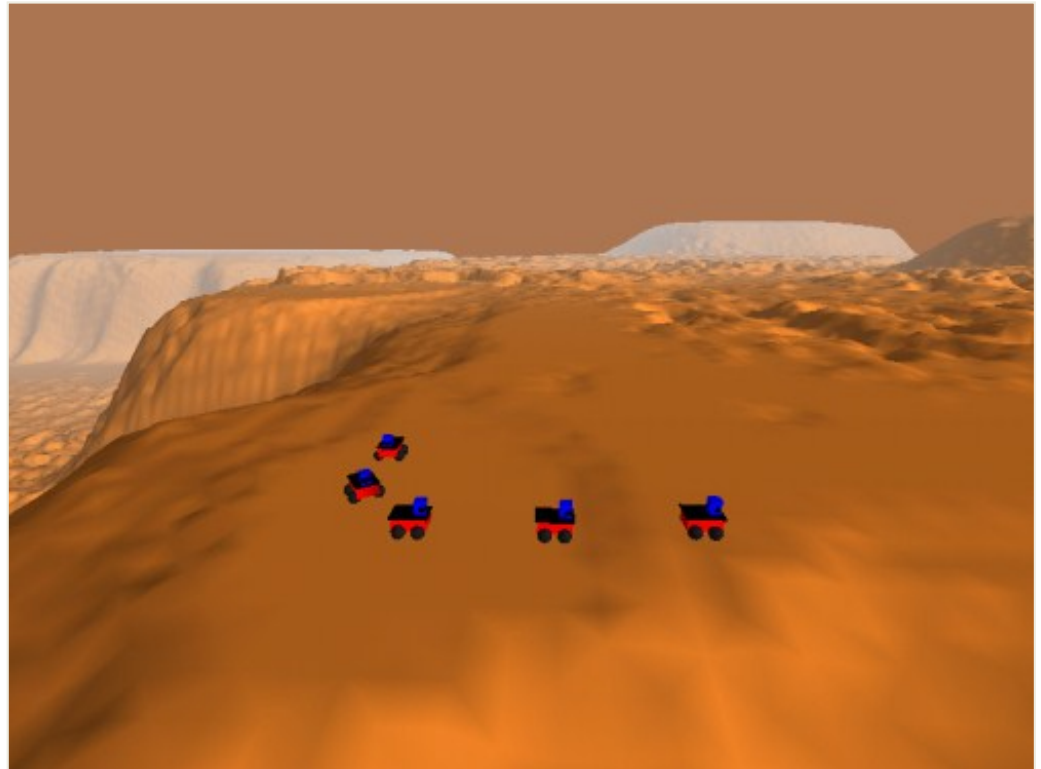
## Gazebo

- 3D
- Sensor-based
- Player
- Dynamic
- $O(n) \sim O(n^3)$
- Small teams (10's)



# Gazebo

- Simulates robots, sensors, and objects in a 3-D dynamic environment
- Generates realistic sensor feedback and physical interactions between objects



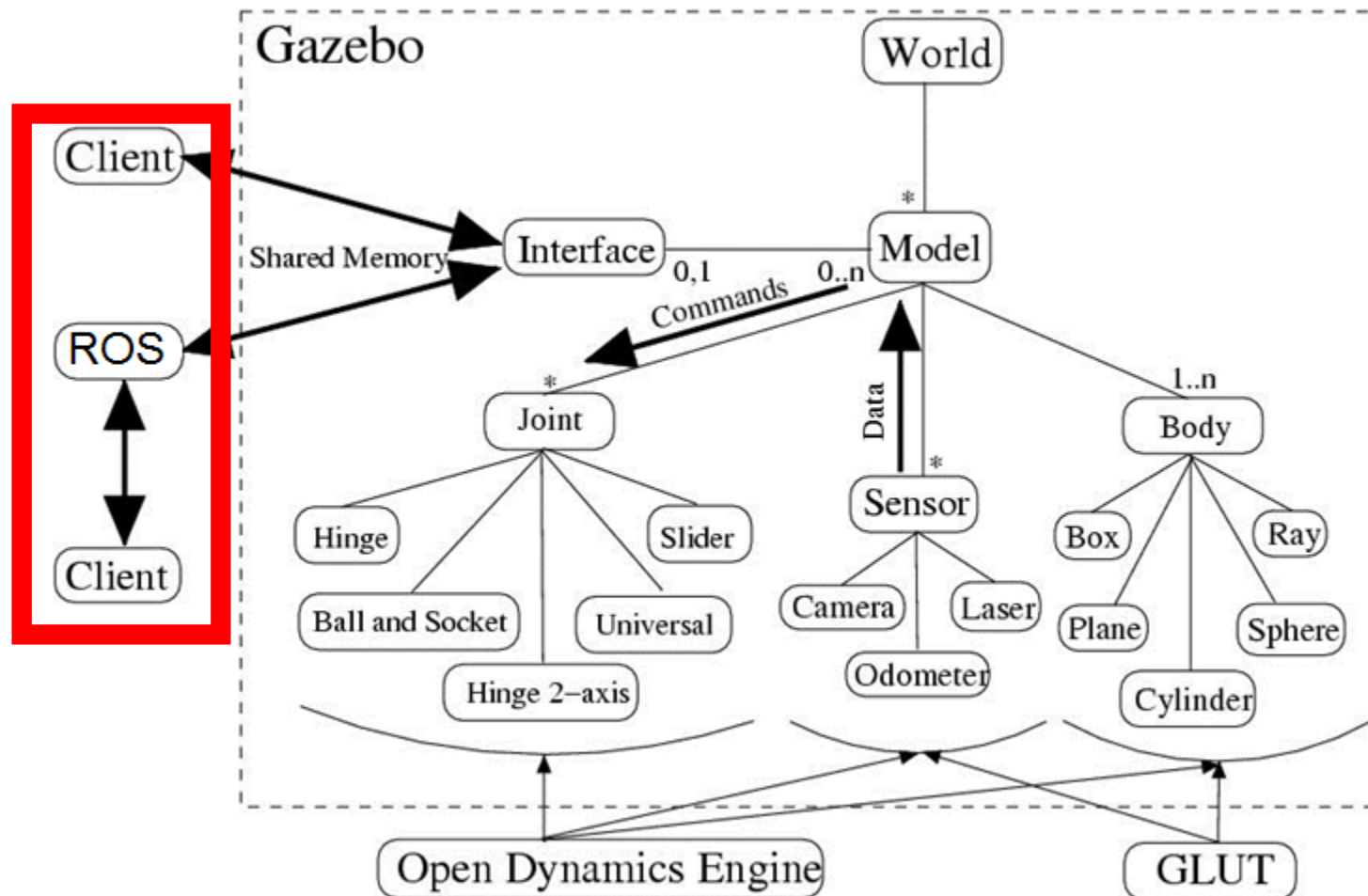
# Gazebo (Cont.)

- gzserver
  - executable runs the physics update-loop and sensor data generation
  - This is core of Gazebo, and can be used independently of any graphical interface
- Gzclient
  - executable runs the QT based user interface
  - provides a nice visualization of simulation, and convenient controls over various simulation properties

# Gazebo Components

- **World File:** Contains all the elements in a simulation, including robots, lights, sensors, and static objects. This file is formatted using SDF (Simulation Description Format), and typically has a .world extension.
- **Model File:** A SDF file used to describe a single model.
- **Environment Variables:** For storing environment, communication settings
- **Gazebo Server+Client :** The two main components of a simulation.
- **Plugins :** A simple mechanism to interface with the simulation world

# Gazebo Architecture

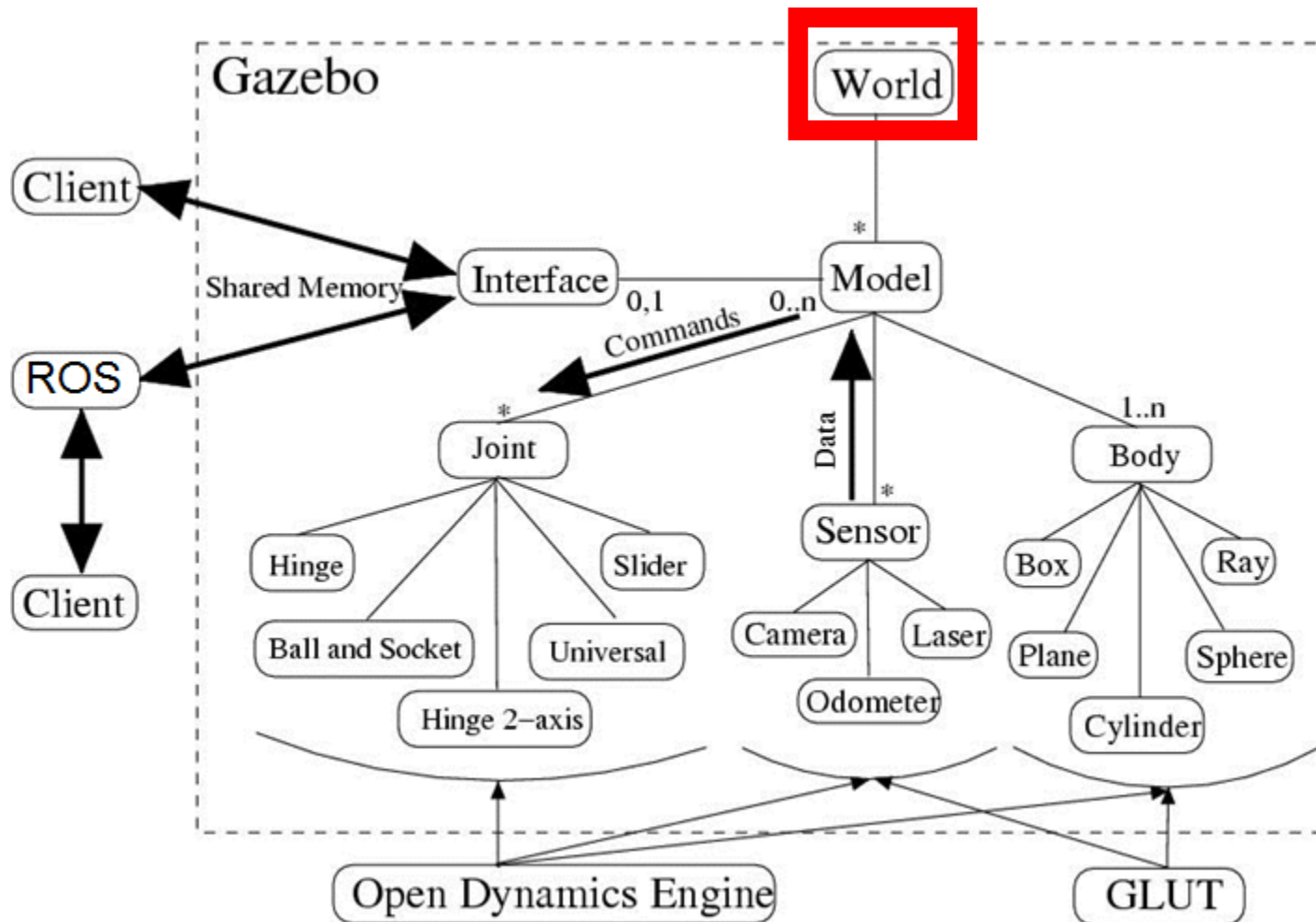




# Gazebo Client Code

- Client code (your program), can interface to Gazebo in two ways
  - libgazebo
    - Shared Memory, direct interface
    - Fast, but requires more work
  - ROS
    - Simulation transparency
    - Get all of ROS's goodies
    - Recommended for most cases
    - Gazebo was part of ROS

# Gazebo Architecture



# World File

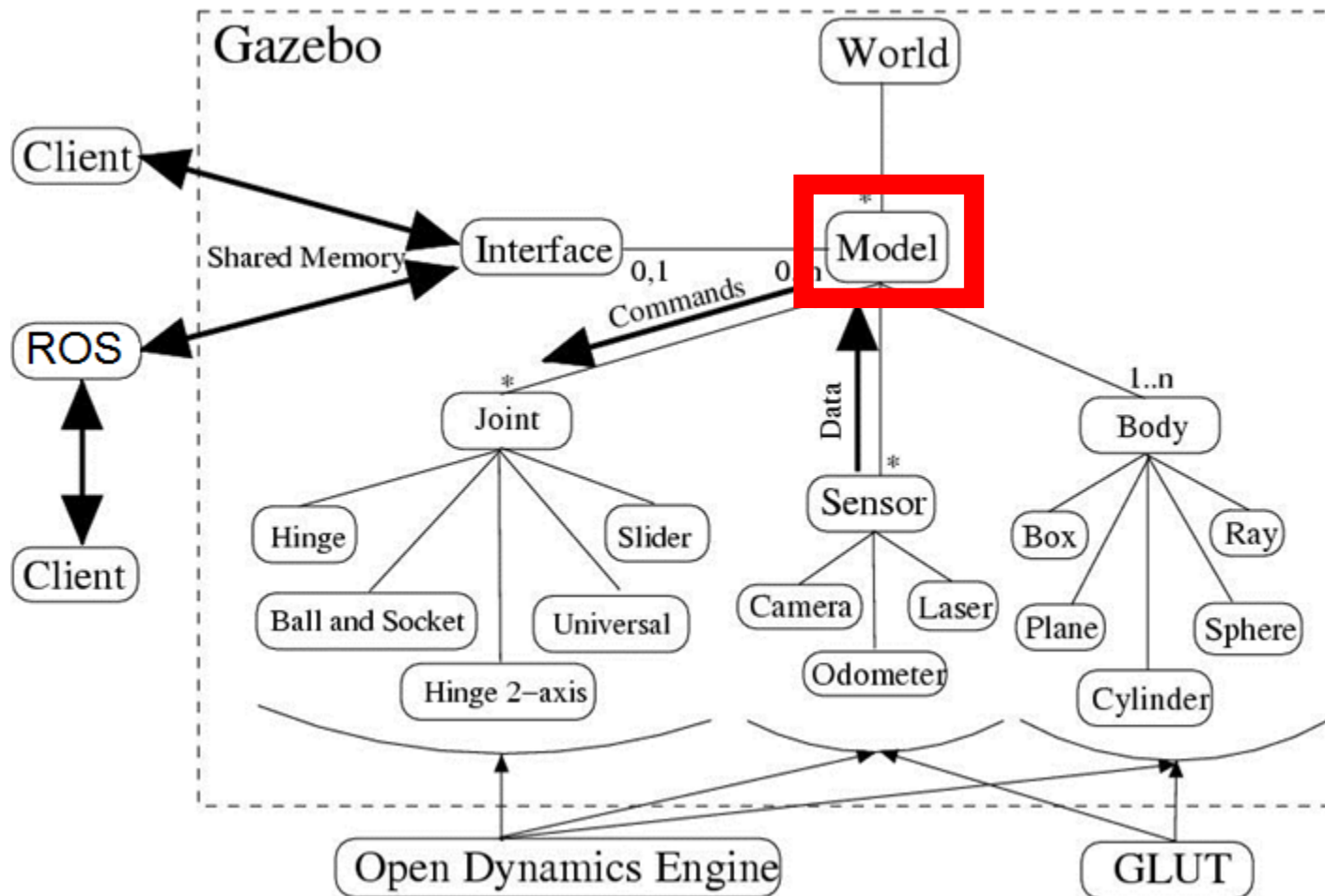
- A world is composed of a model hierarchy
  - Models define simulated devices
  - Models can be nested
    - Specifies how models are physically attached to one another
    - Think of it as “bolting” one model to another

## Worldfile snippet:

- Pioneer with a sick laser attached
- Sick's <xyz> relative location to Pioneer

```
<model:Pioneer2AT>  
  <id>robot1</id>  
  <model:SickLMS200>  
    <id>laser1</id>  
    <xyz>0.10.0 0.2</xyz>  
  </model:SickLMS200>  
</model:Pioneer2AT>
```

# Gazebo Architecture



# 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)

# Components of a Model

- **Links:** an object may consist of multiple links and can define following properties, e.g. wheel
  - **Visual:** For visualization
  - **Collision:** Encapsulate geometry for collision checking
  - **Inertial:** Dynamic properties of a link e.g. mass, inertia
  - **Sensors:** To collect data from world for plugins
- **Joints:** connect links using a parent-child relationship
- **Plugins:** Library to control model

# Example Model File

<pre> &lt;model name="my_model"&gt;   &lt;pose&gt;0 0 0.5 0 0 0&lt;/pose&gt;   &lt;static&gt;true&lt;/static&gt;   &lt;link name="link"&gt;     &lt;inertial&gt;       &lt;mass&gt;1.0&lt;/mass&gt;       &lt;ixx&gt;0.083&lt;/ixx&gt;    &lt;!-- for a box: <math>ixx = 0.083 * mass * (y*y + z*z)</math> --&gt;       &lt;ixy&gt;0.0&lt;/ixy&gt;    &lt;!-- for a box: <math>ixy = 0</math> --&gt;       &lt;ixz&gt;0.0&lt;/ixz&gt;    &lt;!-- for a box: <math>ixz = 0</math> --&gt;       &lt;iyy&gt;0.083&lt;/iyy&gt;   &lt;!-- for a box: <math>iyy = 0.083 * mass * (x*x + z*z)</math> --&gt;       &lt;iyz&gt;0.0&lt;/iyz&gt;   &lt;!-- for a box: <math>iyz = 0</math> --&gt;       &lt;izz&gt;0.083&lt;/izz&gt;   &lt;!-- for a box: <math>izz = 0.083 * mass * (x*x + y*y)</math> --&gt;     &lt;/inertial&gt;   &lt;/link&gt;   ..... </pre>	<pre> ..... &lt;collision name="collision"&gt;   &lt;geometry&gt;     &lt;box&gt;       &lt;size&gt;1 1 1&lt;/size&gt;     &lt;/box&gt;   &lt;/geometry&gt; &lt;/collision&gt; &lt;visual name="visual"&gt;   &lt;geometry&gt;     &lt;box&gt;       &lt;size&gt;1 1 1&lt;/size&gt;     &lt;/box&gt;   &lt;/geometry&gt; &lt;/visual&gt; &lt;/link&gt; &lt;/model&gt; </pre>
--	--

# Task 1: Perform the following Gazebo Tutorials

- Build a Robot
  - Model structure and requirements
  - Make a model
  - Make a Mobile Robot
  - Import Meshes
  - Attach Meshes
  - Add a Sensor to a Robot



# Task 2: Perform the following Gazebo Tutorials

- Build a World
  - Building a world
- Write a plugin
  - Plugins 101
  - Model plugins
  - World plugins
- Sensors.
  - Sensor Noise Model (Ray Laser noise)

# Task 3: Perform the following Gazebo Tutorials

- Connect to ROS
  - Installing gazebo\_ros\_pkgs
  - Using roslaunch
  - Gazebo plugins in ROS
    - Adding Plugins
    - Differential Drive
  - ROS communication
  - ROS plugins

# Lab Assignment

1. Create a wheeled mobile robot with a Hokuyo laser scanner attached on it.
2. Use an existing plugin for the mobile robot drive system.
3. **Bonus:** create your own plugin for the robot drive system
4. Use Rviz to visualize odometry and laser scan topics.
5. Create a ROS node to communicate with robot odometry and laser range scanner data. Use the robot wheel odometry to estimate the wheels velocity (Hint: inverse kinematics). To navigate the robot use existing teleop node.