上海科技大学
ShanghaiTech University

# CS289: **Mobile Manipulation Fall 2024**

Sören Schwertfeger

ShanghaiTech University

STAR CENTER

# teb_local_planner

An optimal trajectory planner for mobile robots based on Timed-Elastic-Bands

http://wiki.ros.org/teb_local_planner

# SIMULATION

# Types of Simulators

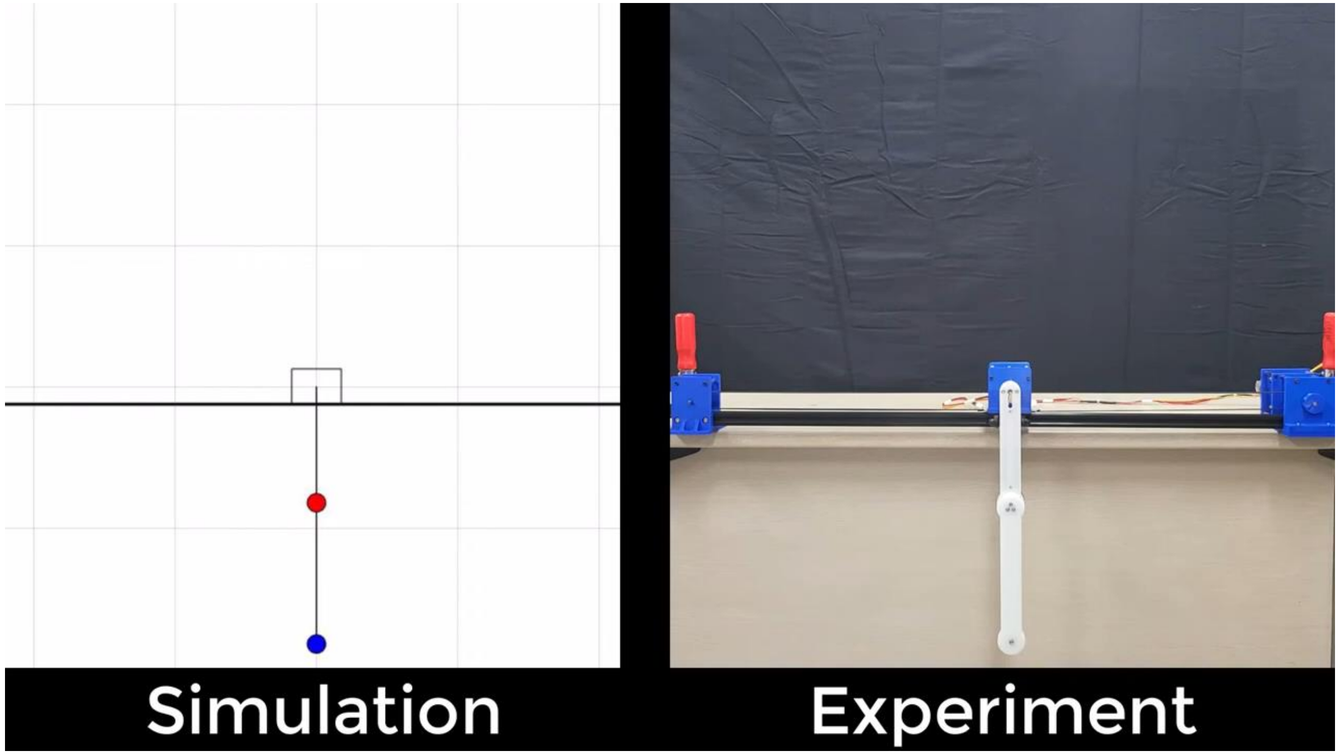
- Dynamics Simulation
- 3D Games
- Photo-realistic rendering
- Mission simulators
- 2D simulators
- 3D simulators

# Dynamics Simulation (Control)

- Model a dynamic system
  - Kinematics (joints & links)
  - Dynamics (mass, inertia)
- Often used for research on control

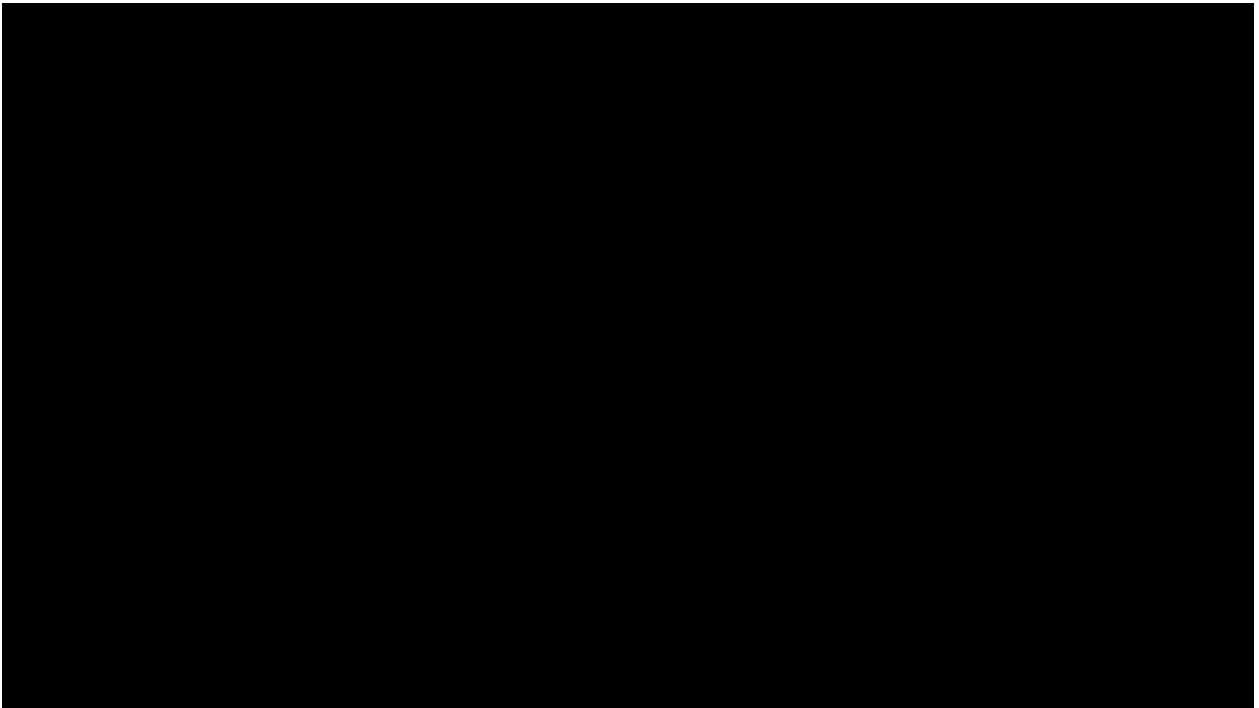
- Example:
  - Double inverted pendulum
  - E.g. via Model Predictive Control (MPC)
  - 2D simulation only

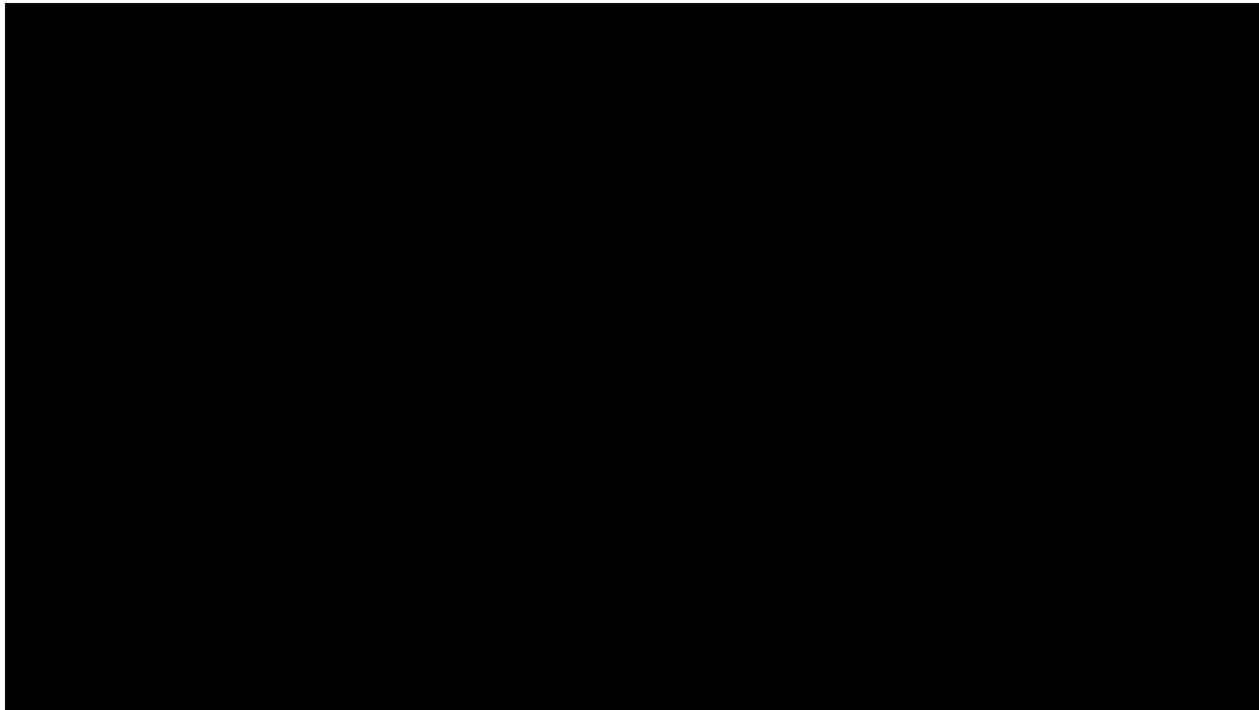
https://www.youtube.com/watch?v=e7669NPbENY



# 3D Games

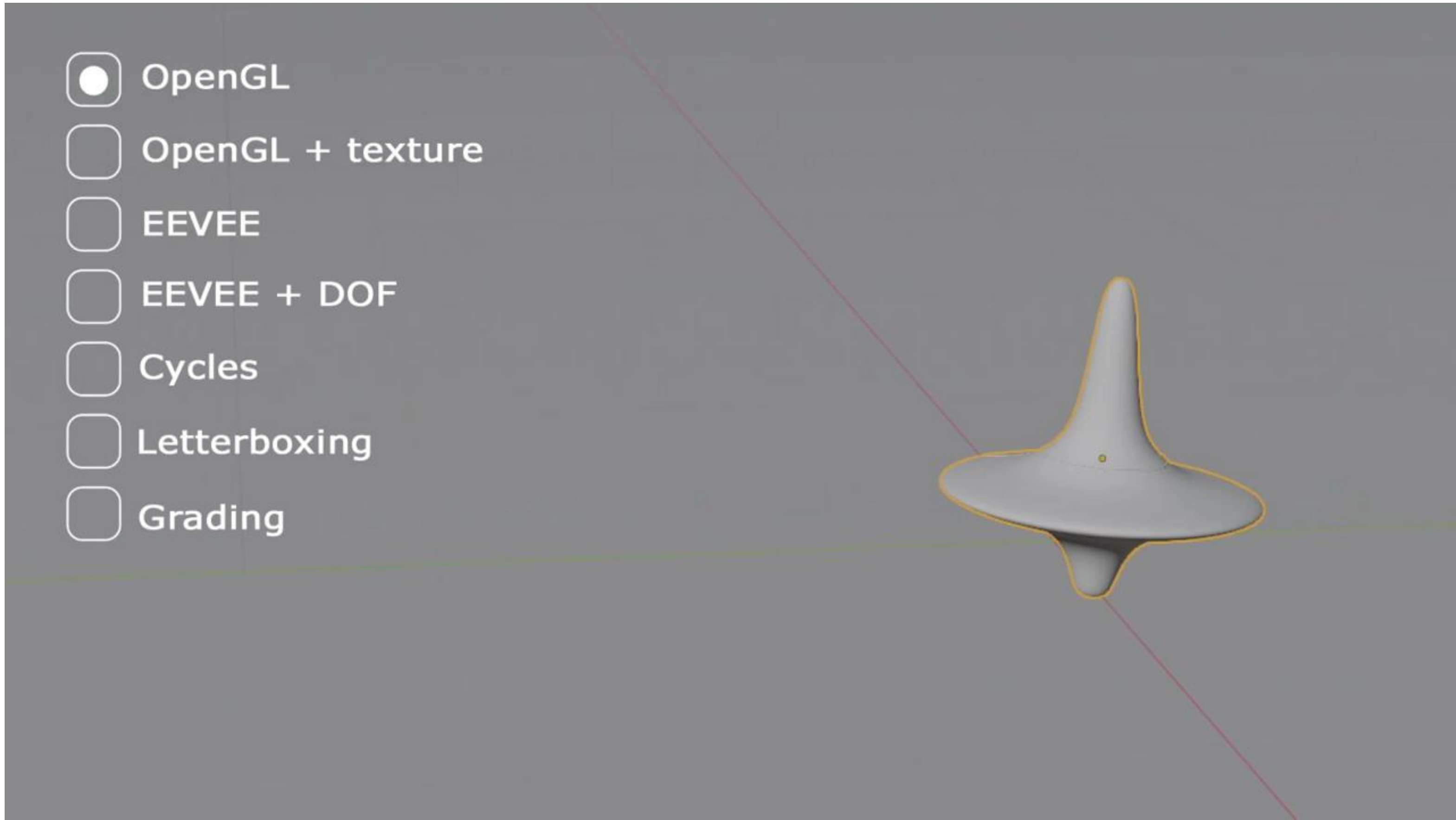
- Simulate 3D world:
  - Kinematics and Dynamics (typically very simple)
  - RGB camera view
  - Sound





# Photo-realistic rendering

- 3D game engine photos not realistic (at least in the past)
  - Problem for computer vision algorithms
  - Especially for training computer vision algorithms (sim-to-real)
- Photo-realistic rendering:
  - Time consuming
  - Especially lights, shadows, transparence, hair, etc.
  - Often: non-real time
  - Blender: open source renderer:

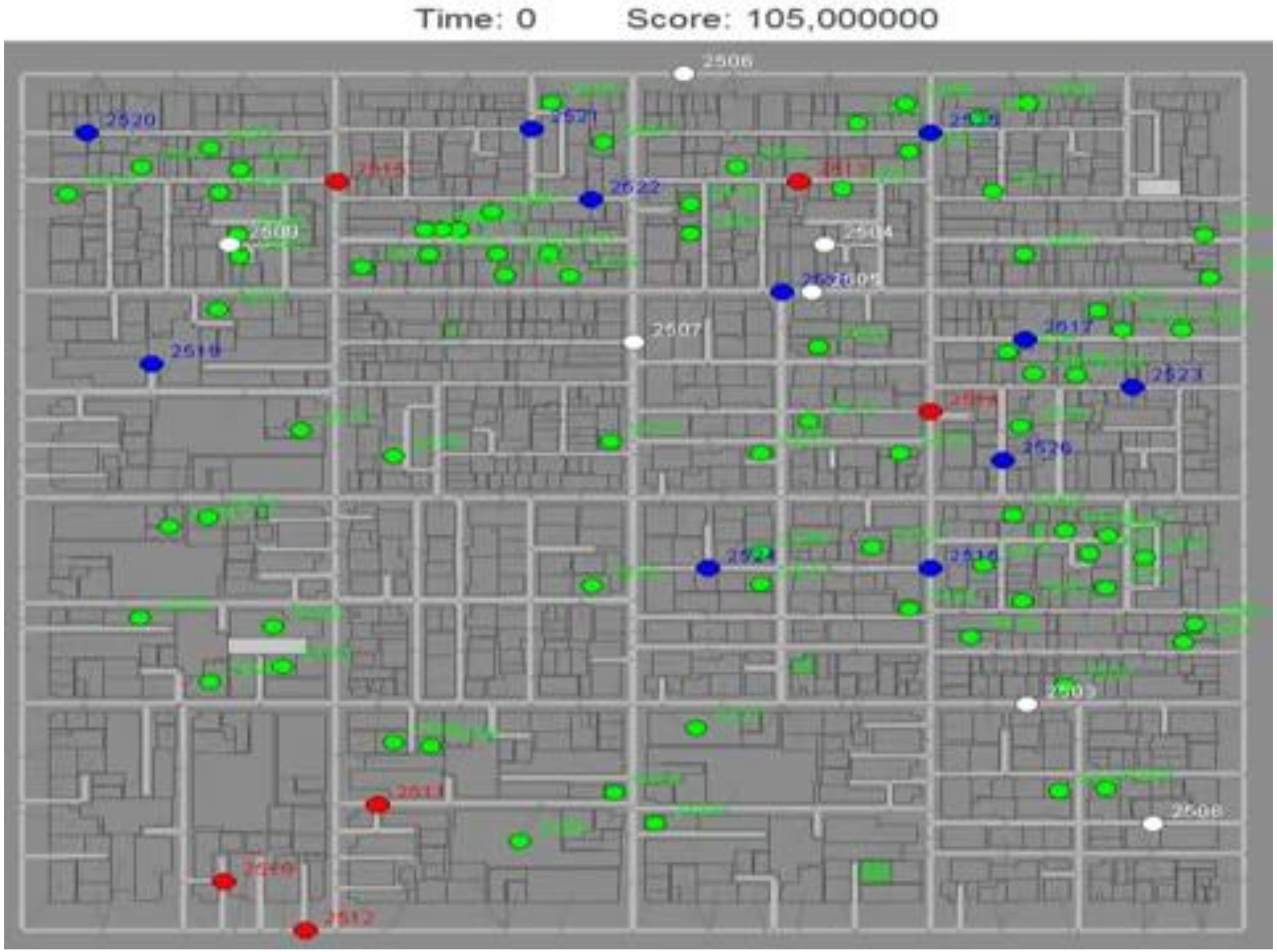



Next slide: Unrecord trailer
https://www.youtube.com/watch?v=otu_iFTivQw


DiRT

# Robot mission task simulator: RoboCup Rescue

- Simulate deployment of (robotic) units like police, fire, ambulance
- All entities are agents
- Communication between agents
- Simulate environment, e.g.:
  - Spreading of fire



# 3D Robot Simulation

## Why?

- Training & education
  - (when you don’t have a robot)
- Ease of experiments
  - Don’t need to deal with the hardware (e.g. battery charging, weather, robot breaking, bringing robot back to start, …)
- Multi-robot simulation
  - Simulate multiple (100s!?) of robots -> low cost in simulation
- Collect ground truth data
  - Know exactly where the robot is & its state
  - Know exactly where surrounding objects are
- Collect training data (see above)
- Train robot online (e.g. reinforcement learning)

## Why not?

- Extra work
  - Need to setup simulated robot & simulated world
- Simulated data differs from real data
  - Non-photo realistic rendering
  - Non-realistic noise on simulated sensor data
  - Non-realistic physics (e.g. tracked locomotion)

Activities
rviz2
7月 28 02:49
AWSIM
/home/mackie/workspace/GitHub/tier4/autoware/install/autoware_launch/share/autoware_launch/rviz/autoware....
File
Panels
Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Goal Pose
-0.4deg
2.90km/h
limited
72km/h
occlusion_spot
(d,v,m)=(-0.85 , -3.74 , 3.88 , 7.22
4d6a
82e4
Time
ROS Time: 63.06
ROS Elapsed: 53.01
Wall Time: 1658944181.38
Wall Elapsed: 53.02
Experimental
Reset
21 fps
Time Scale x 1.00
AWSIM
(prototype ver.)

Waiting for the next game…

3rd place match: 12:00


Untitled

# Simulators Overview

- Many different simulation platforms available…

Tselegkaridis, Sokratis, and Theodosios Sapounidis. "Simulators in educational robotics: A review." Education Sciences 11.1 (2021): 11.

| Name | Develop Year | Type of Robot | Users' Age | Programming Language | Scope | Used | User's Level | Based on a Real Robot | Development Platform | Operating System |
|---|---|---|---|---|---|---|---|---|---|---|
| RoSoS * | 2016 | Wheeled | 6~7 | Processing | Competition | 1, 2 | Beginner | No | Processing | W, L, M |
| RoboSim | 2017 | Modular | 10~18 | C/C++ | Competition | 1 | Intermediate | Yes | - | W, M |
| Robot One * | 2019 | Multiple | >18 | Many | Education | 2 | Expert | No | Unity3D | W, L |
| Tactode | 2019 | Multiple | 6~11 | Puzzle/Tangible | Education | - | Beginner | No | - | n/s |
| Pololu * | 2018 | Wheeled | n/s | C/C++ | Education | 2 | Beginner | Yes | - | W |
| EUROPA | 2019 | Wheeled | >12 | Python | Education | 3 | Intermediate | Yes | Gazebo | n/s |
| ADS * | 2019 | Wheeled | >18 | n/s | Competition | 1, 2 | Intermediate | Yes | Gazebo | L |
| MLPNN | 2020 | Wheeled | >18 | MATLAB | Research | - | Expert | No | MATLAB | W, L |
| Drone Simulator | 2019 | Drone | >12 | MATLAB | Research | 1, 3 | Expert | No | MATLAB/Gazebo | W, L |
| PiBot * | 2018 | Wheeled | >12 | Python | Education | 4 | Intermediate | Yes | Gazebo | W, L, M |
| AlphaBot2 * | 2019 | Wheeled | >12 | n/s | Research | - | Intermediate | Yes | Gazebo | W, L |
| MSRP | 2017 | Wheeled | >18 | n/s | Education | - | Expert | Yes | Unity3D | n/s |
| KheperaIV | 2016 | Wheeled | n/s | C/C++ | Education | - | Intermediate | Yes | V-REP | n/s |
| OBR simulator | 2018 | Wheeled | 6~18 | Flowchart | Competition | 1, 5 | Intermediate | Yes | V-REP | W, L |
| MiniSim * | 2014 | Wheeled | 6~10 | Python/Block | Education | - | Beginner | No | - | W, L |
| SRM | 2018 | Arm | >18 | MATLAB | Education | 2 | Expert | No | MATLAB/V-REP | W, L |
| LaRoCS + Unesp | 2018 | Drone | n/s | n/s | Competition | - | Intermediate | Yes | V-REP | W, L, M |

Note: 1: Competition, 2: University, 3: Secondary School, 4: Robotics workshops for teachers, 5: Technical high school; W: Windows, L: Linux, M: MacOS; *: Open access; n/s: not specified.

# Most popular professional/ research Robot Simulators

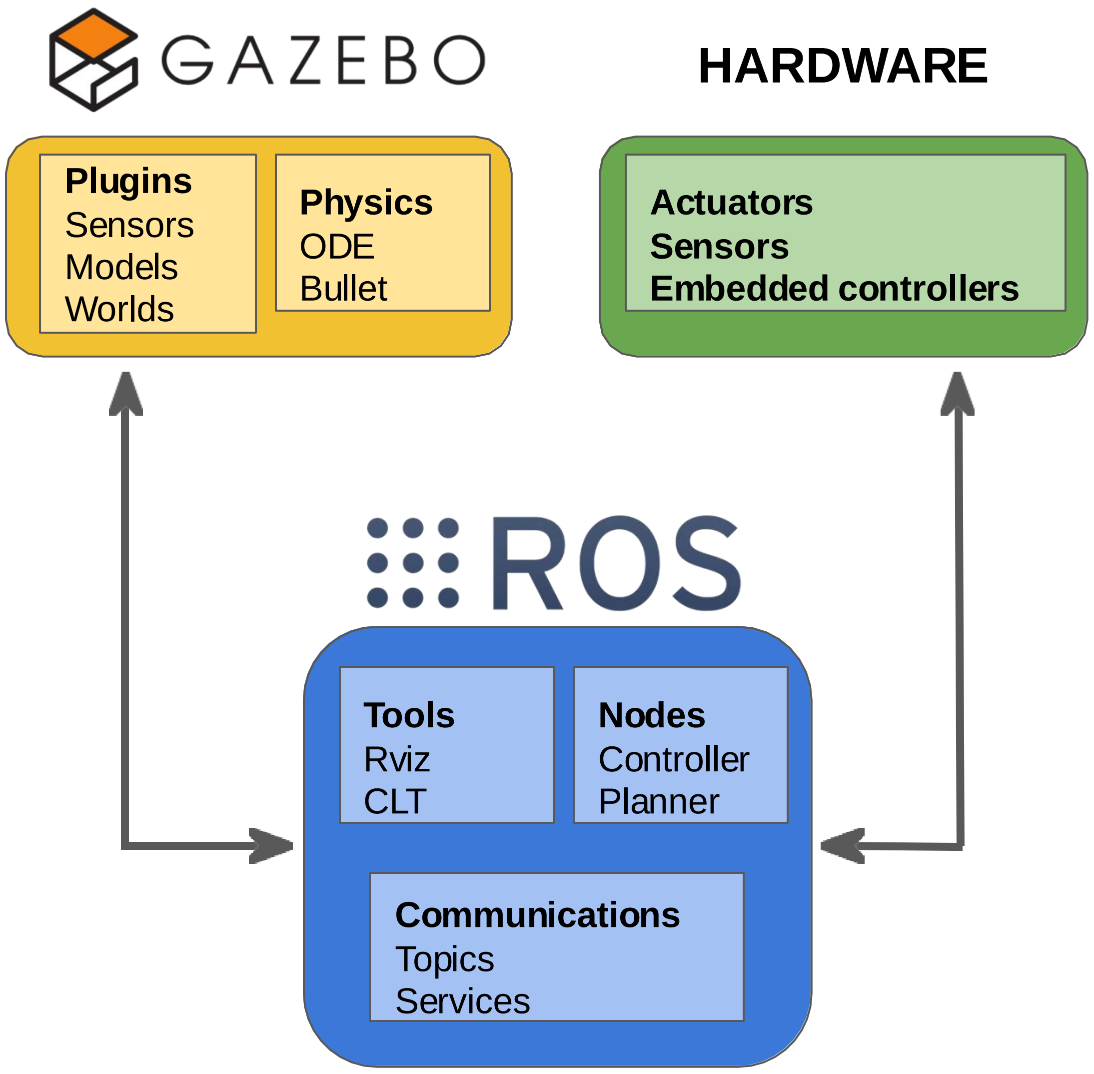
Farley, Andrew, Jie Wang, and Joshua A. Marshall. "How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion." Simulation Modelling Practice and Theory 120 (2022): 102629.

Summary of the simulator qualitative features evaluation results, after completing the setup of each simulation environment.

| Metric name | CoppeliaSim | Gazebo | MORSE | Webots |
|---|---|---|---|---|
| Free to use | True | True | True | True |
| Open source | False | True | True | True |
| ROS compatibility | A built-in plugin provided | Out of the box | Out of the box | A built-in plugin provided |
| Programming languages | C/C++, Python, Lua, MATLAB, Java, Octave | C/C++, Python | Python | C/C++, Python, Java, MATLAB |
| UI functionality | Full functionality | Full functionality | Visualization only | Full functionality |
| Model format support | URDF, SDF, Stl, Obj, Dxf, Collada | URDF, SDF, Stl, Obj, Collada | Blend | Proto Nodes |
| Physics engine support | Bullet, ODE, Vortex, Newton | Bullet, ODE, DART, Simbody | Bullet | ODE |

# Gazebo Robot Simulator

- 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



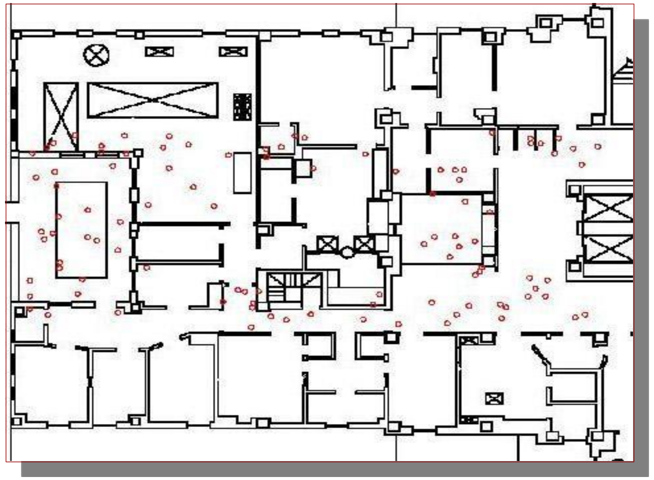
# 2D vs 3D Simulation

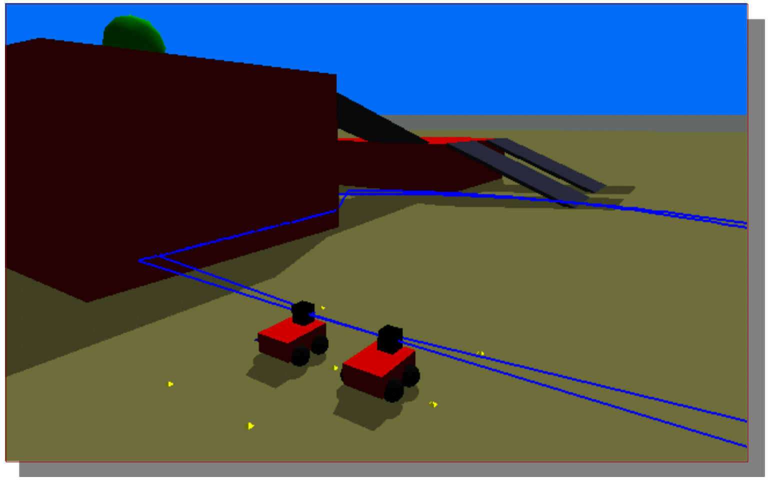
## Stage

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

## Gazebo

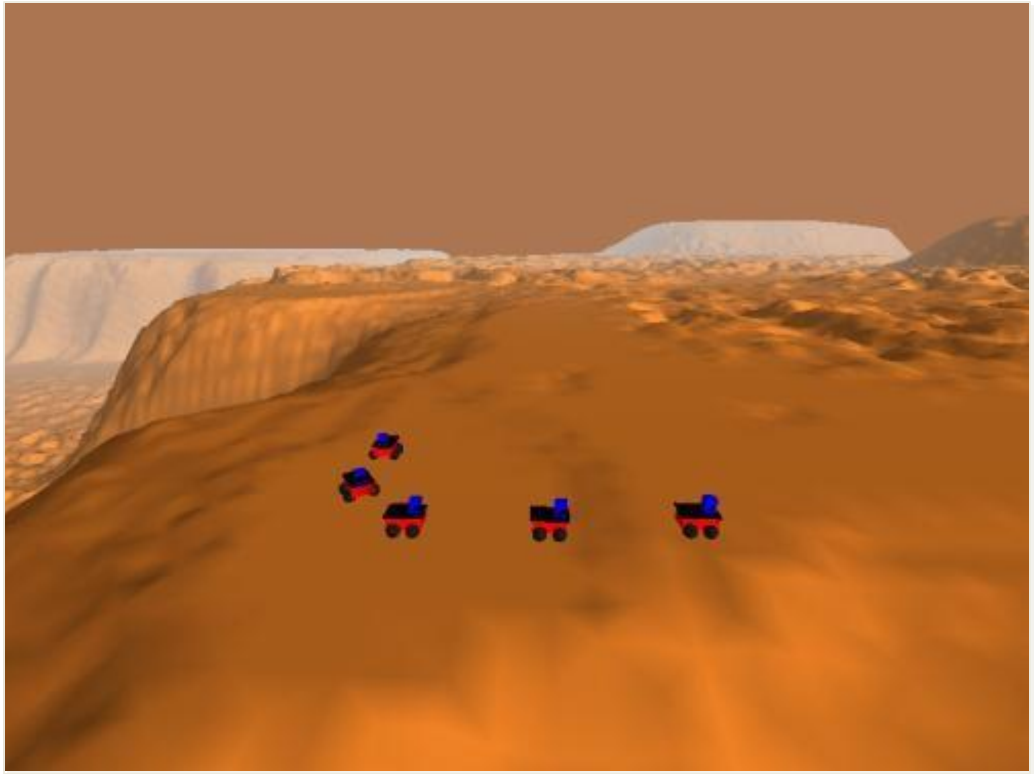
- 3D
- Sensor-based
- Player
- Dynamic
- O(n) ~ $O(n^3)$
- Small teams (10's)



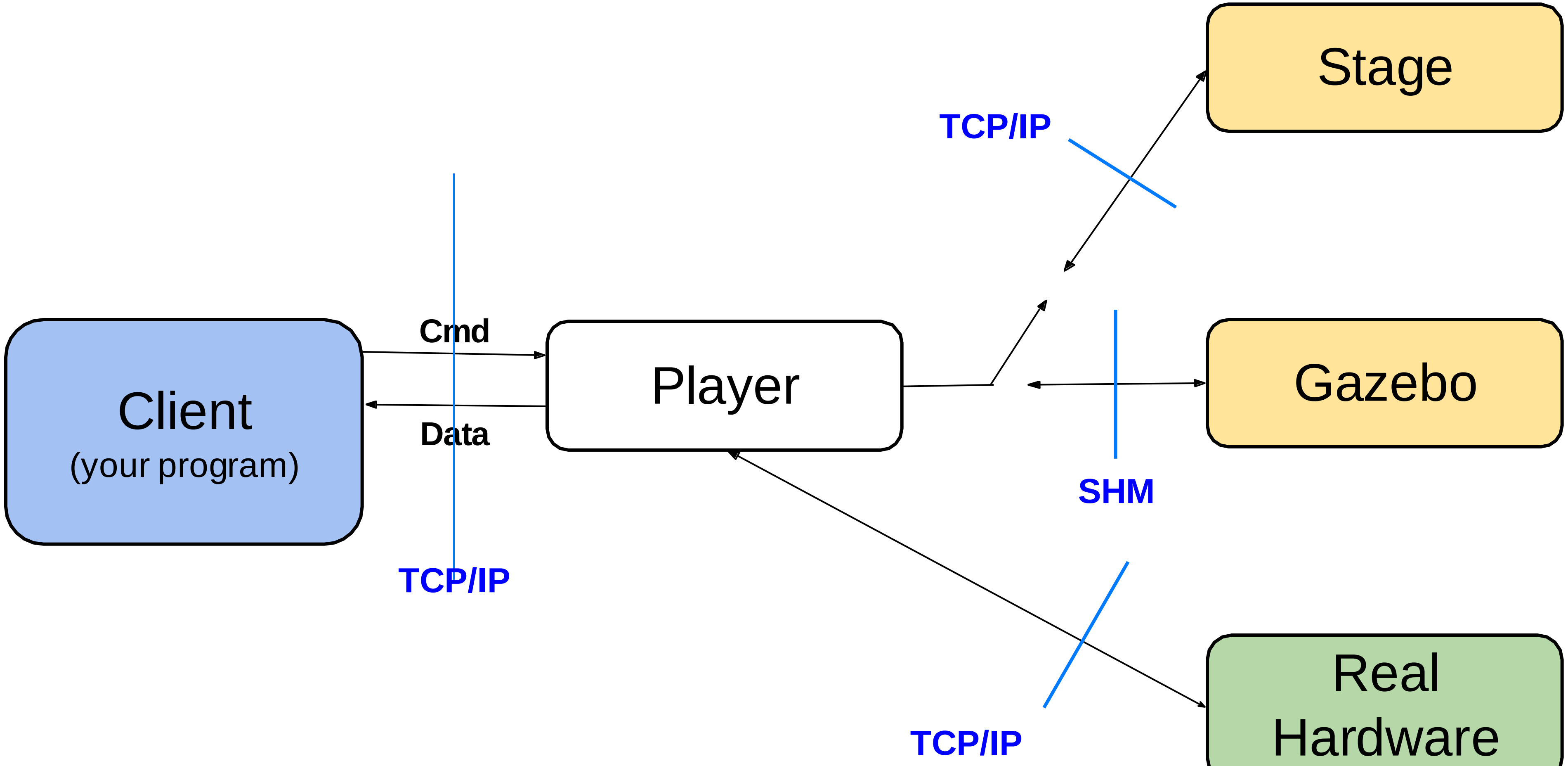


# Gazebo

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



# Simulation Architecture



# 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
```

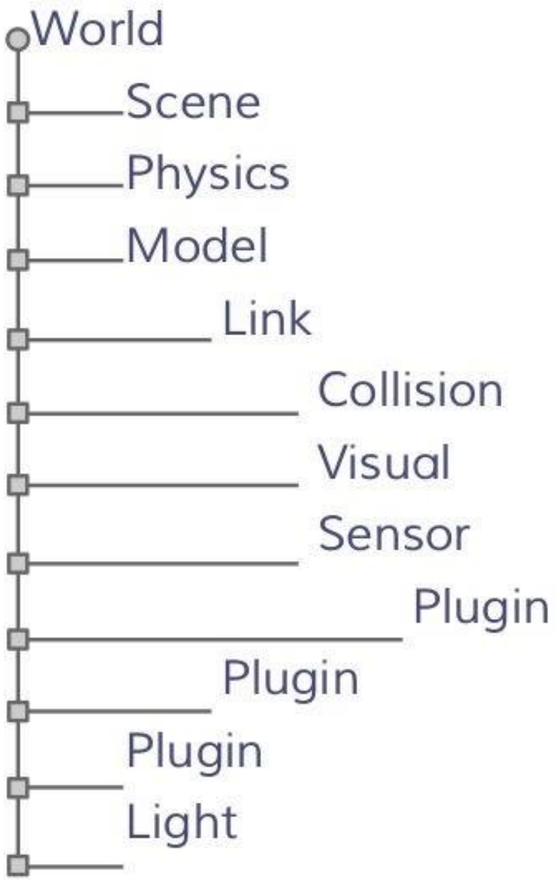
Run Gazebo server and client simultaneously:

```
$ 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



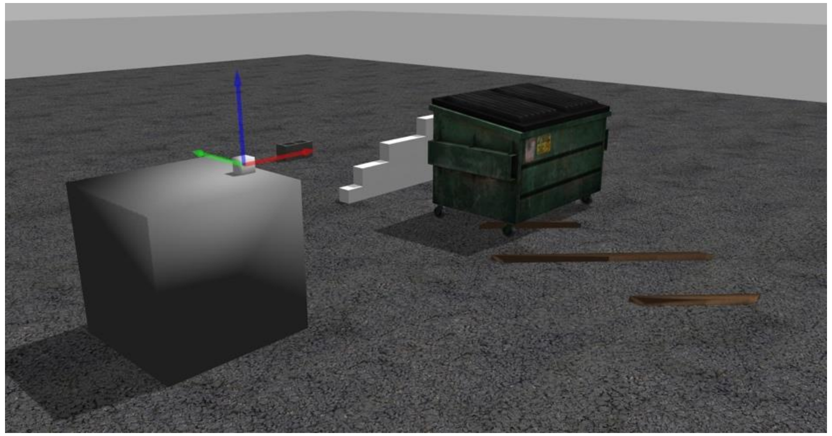

# World

- 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

Willow Garage World

# 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)

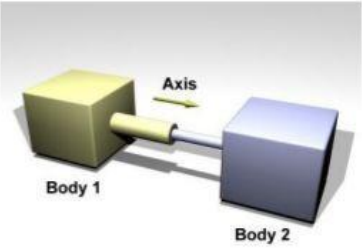


# Element Types

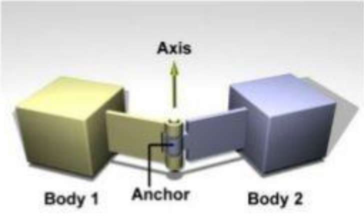
- Collision and Visual Geometries
  - Simple shapes: sphere, cylinder, box, plane
  - Complex shapes: heightmaps, meshes

# Element Types

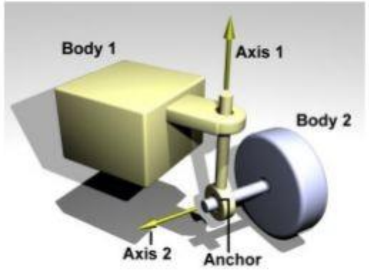
- Collision and Visual Geometries
  - Simple shapes: sphere, cylinder, box, plane
  - Complex shapes: heightmaps, meshes
- Joints
  - Prismatic: 1 DOF translational
  - Revolute: 1 DOF rotational
  - Revolute2: Two revolute joints in series
  - Ball: 3 DOF rotational
  - Universal: 2 DOF rotational
  - Screw: 1 DOF translational, 1 DOF rotational



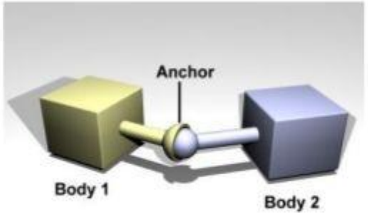
Prismatic



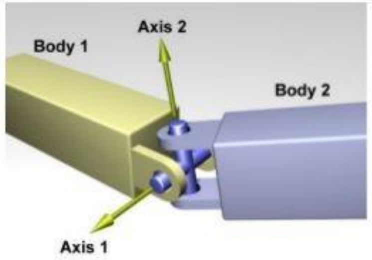
Revolute



Revolute 2



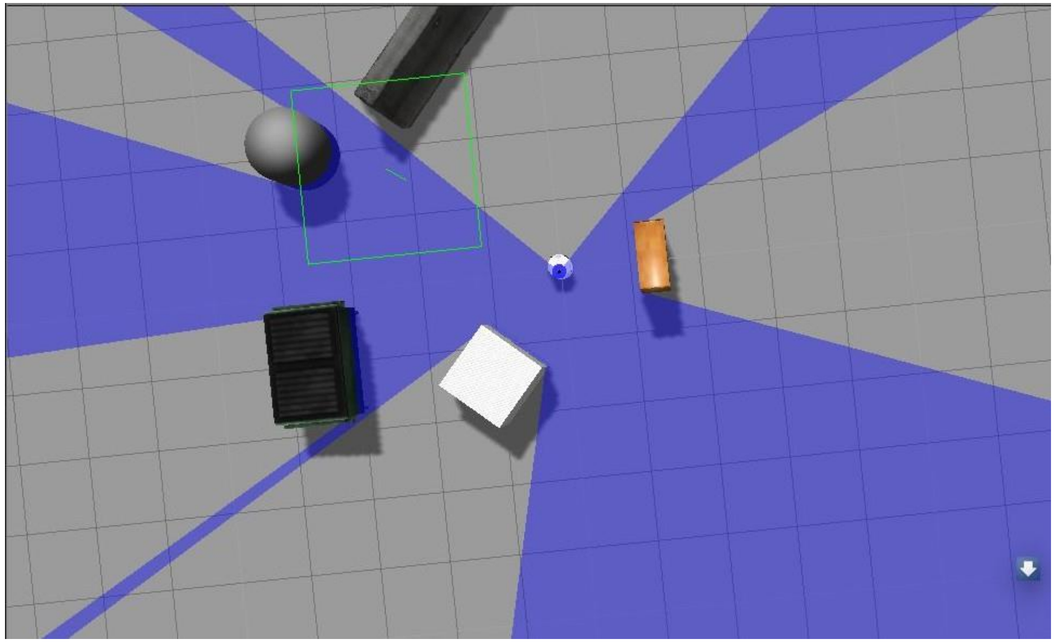
Ball



Universal

# 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

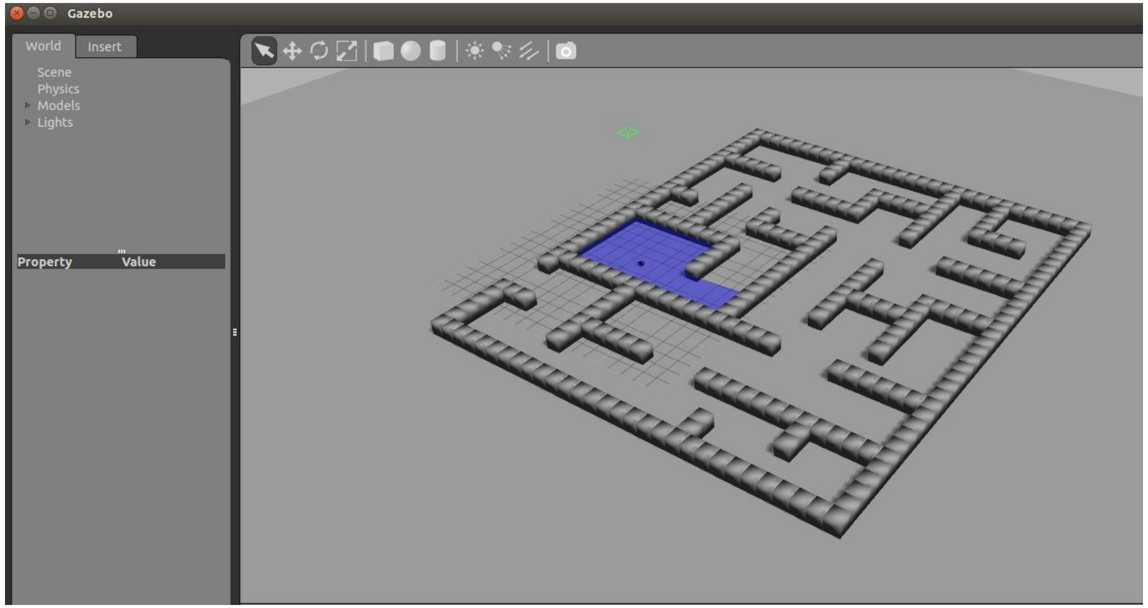


LiDAR sensor in Gazebo

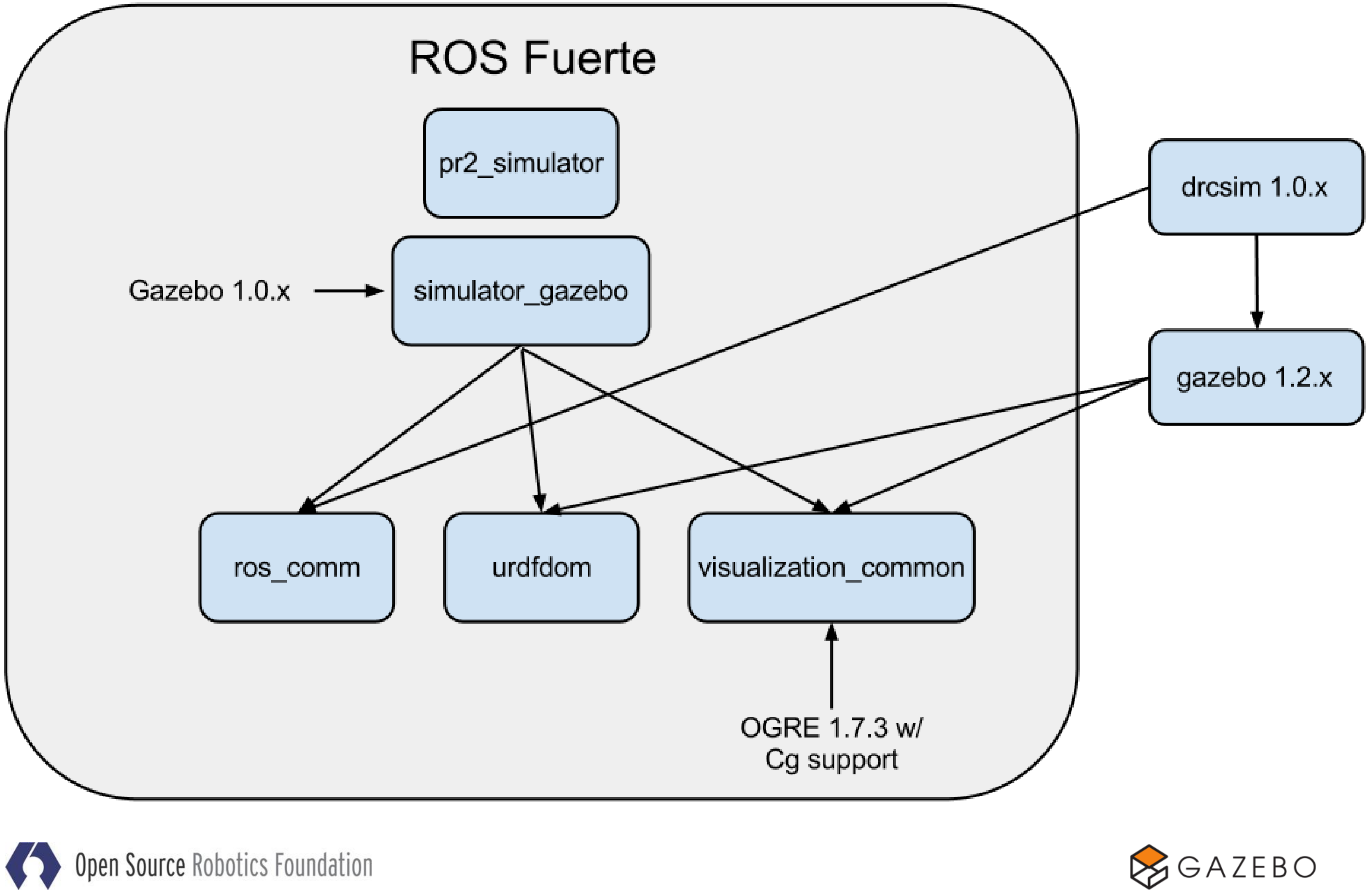
# How to use Gazebo to simulate your robot?

**Steps:**

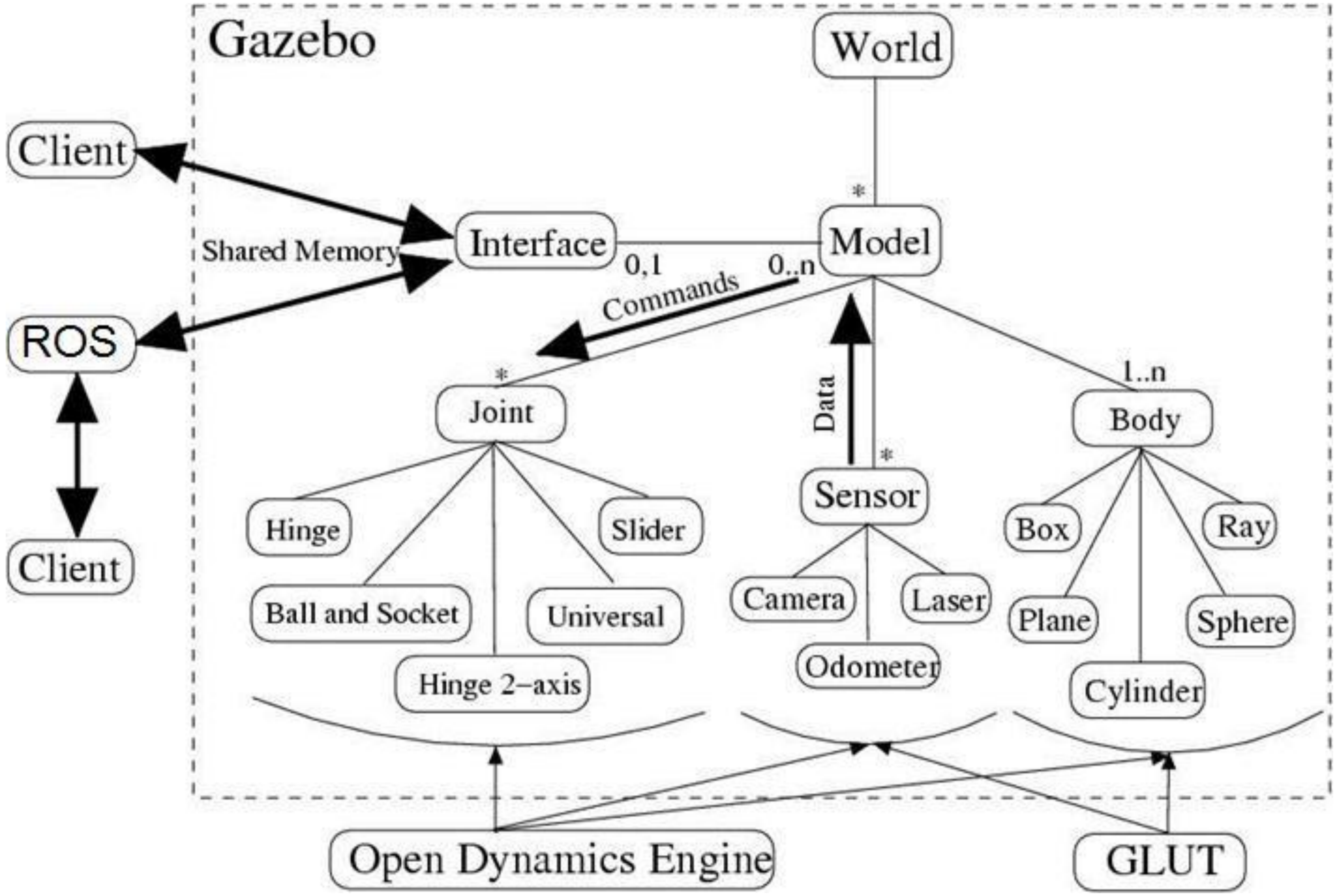
1. load a world
2. load the description of the robot
3. spawn the robot in the world
4. publish joints states
5. publish robot states
6. run rviz



# ROS Integration Overview



# Gazebo Architecture



# World File

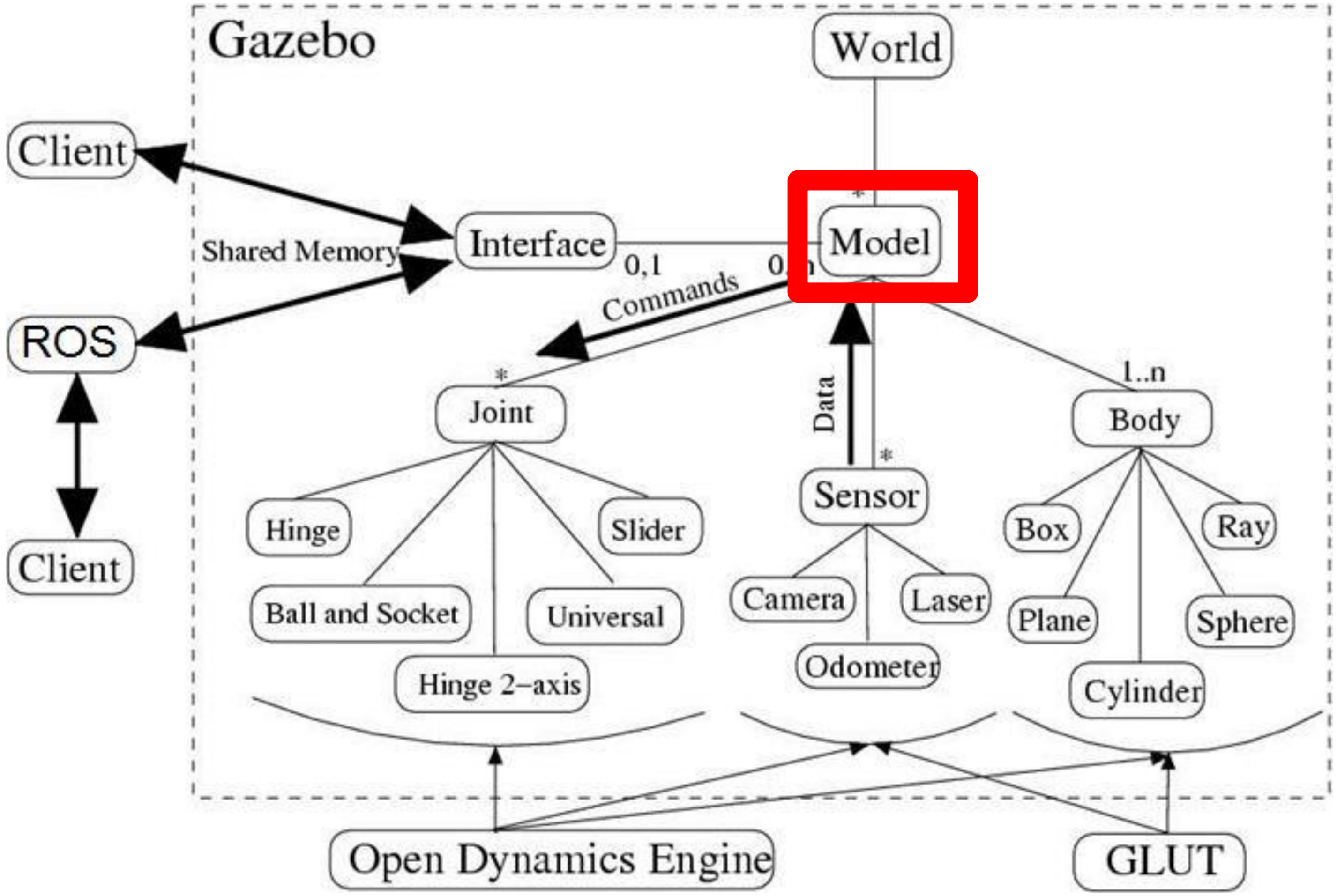
- A world is composed of a model hierarchy
  - Models define simulated devices
  - Models can be nested
    - Specifies how models are physically attached to one another
    - Think of it as “bolting” one model to another

**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)

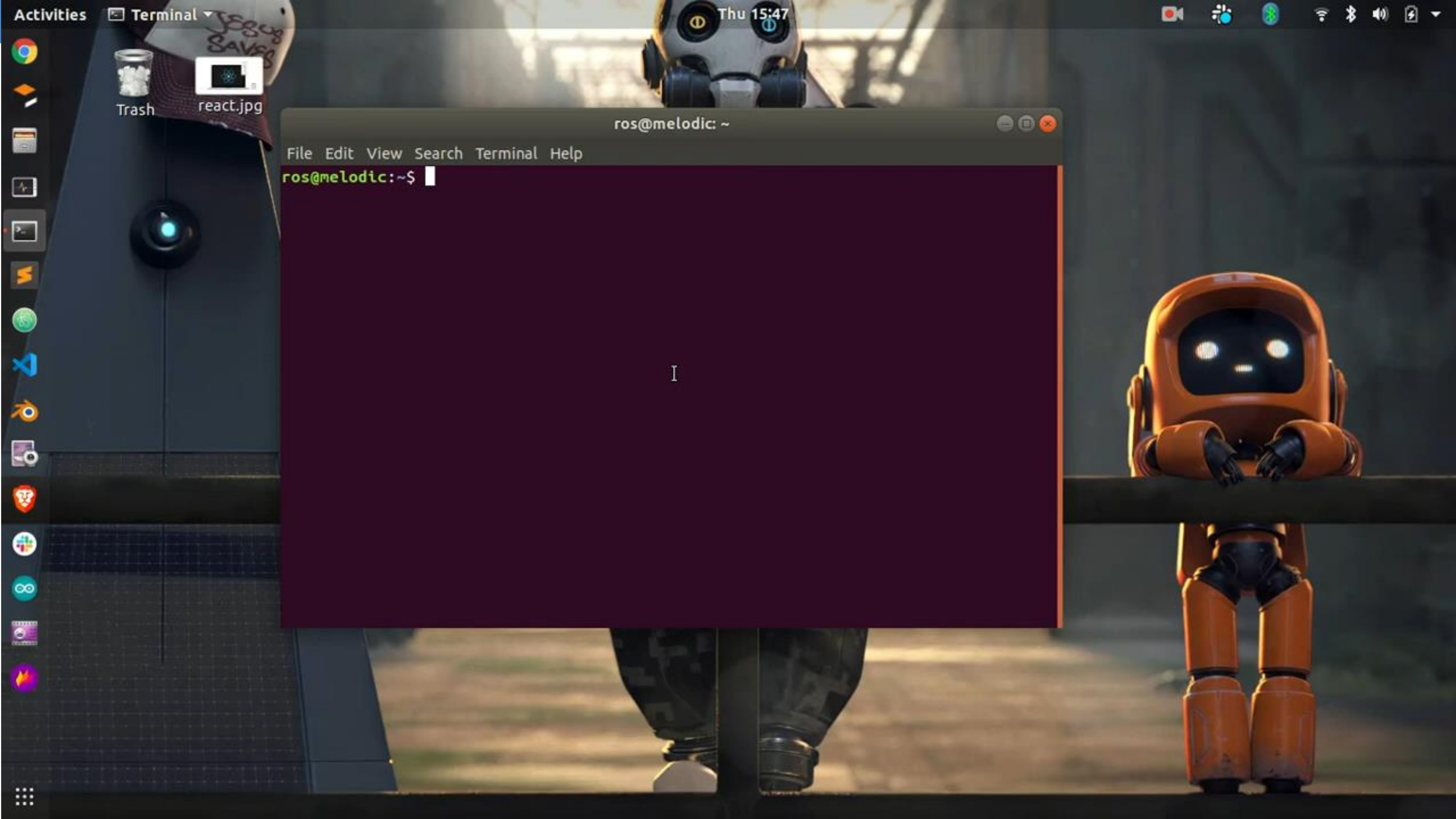
# Components of a Model

- **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

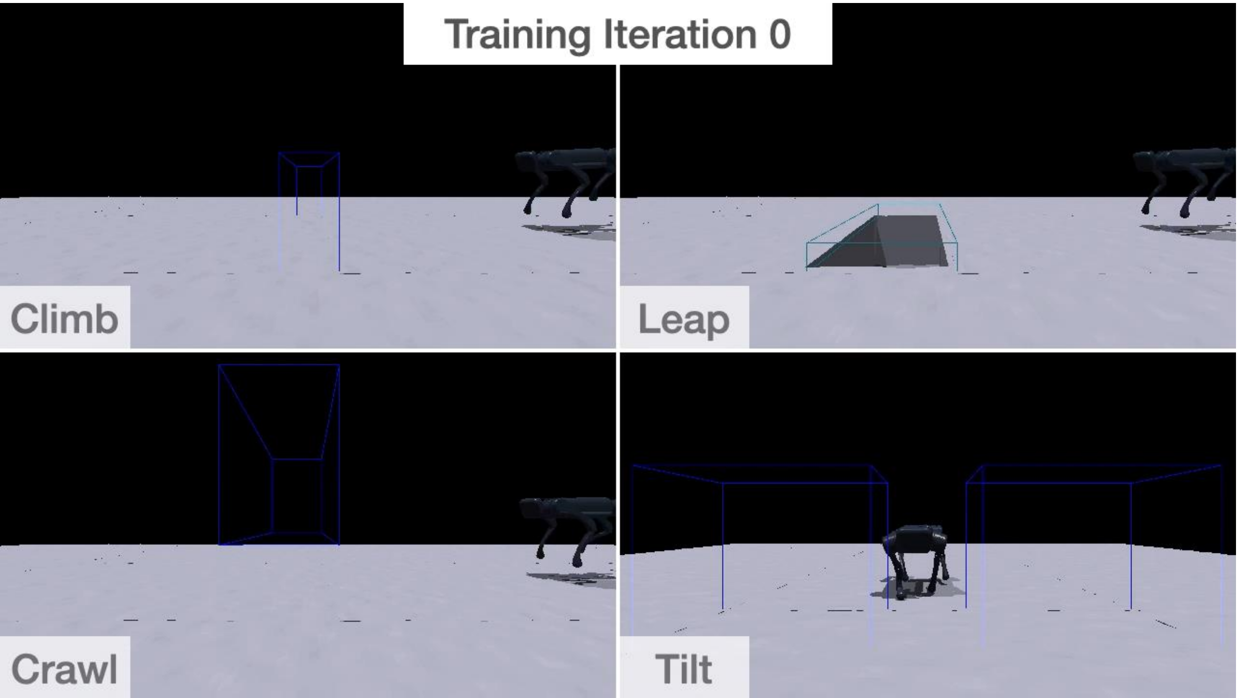
```
<model name="my_model">
  <pose>0 0 0.5 0 0 0</pose>
  <static>true</static>
    <link name="link">
      <inertial>
        <mass>1.0</mass>
          <ixx>0.083</ixx>      <!-- for a box: ixx = 0.083 * mass * (y*y + z*z) -->
          <ixy>0.0</ixy>        <!-- for a box: ixy = 0 -->
          <ixz>0.0</ixz>        <!-- for a box: ixz = 0 -->
          <iyy>0.083</iyy>      <!-- for a box: iyy = 0.083 * mass * (x*x + z*z) -->
          <iyz>0.0</iyz>        <!-- for a box: iyz = 0 -->
          <izz>0.083</izz>      <!-- for a box: izz = 0.083 * mass * (x*x + y*y) -->
        </inertia>
      </inertial>
.......
```

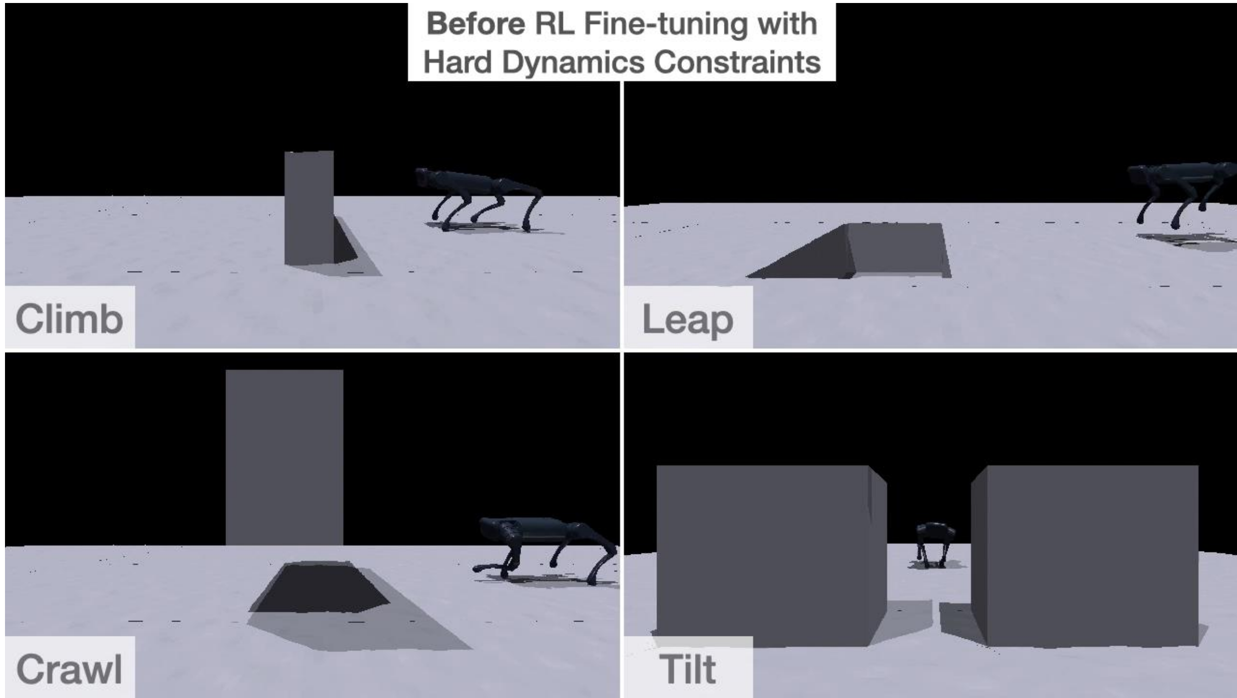
```
...............
 <collision name="collision">
      <geometry>
        <box>
          <size>1 1 1</size>
        </box>
      </geometry>
  </collision>
  <visual name="visual">
      <geometry>
        <box>
          <size>1 1 1</size>
        </box>
      </geometry>
   </visual>
  </link>
</model>
```


Activities
Terminal
Thu 15:47
Trash
react.jpg
ros@melodic: ~
File Edit View Search Terminal Help
ros@melodic:~$

# Reinforcement learning example: Robot parkour learning

- Zhuang, Z., Fu, Z., Wang, J., Atkeson, C., Schwertfeger, S., Finn, C., & Zhao, H. (2023). Robot parkour learning. arXiv preprint arXiv:2309.05665.



# Robot Parkour Learning

twitter: @ziwenzhuang_leo
@zipengfu

# PERCEPTION: VISION

# Image filtering

- Filer: frequency domain processing where “filtering” refers to the process of accepting or rejecting certain frequency components. E.g.:
  - Lowpass filer: pass only low frequencies => blur (smooth) an image
  - **spatial filters** (also called masks or kernels): same effect

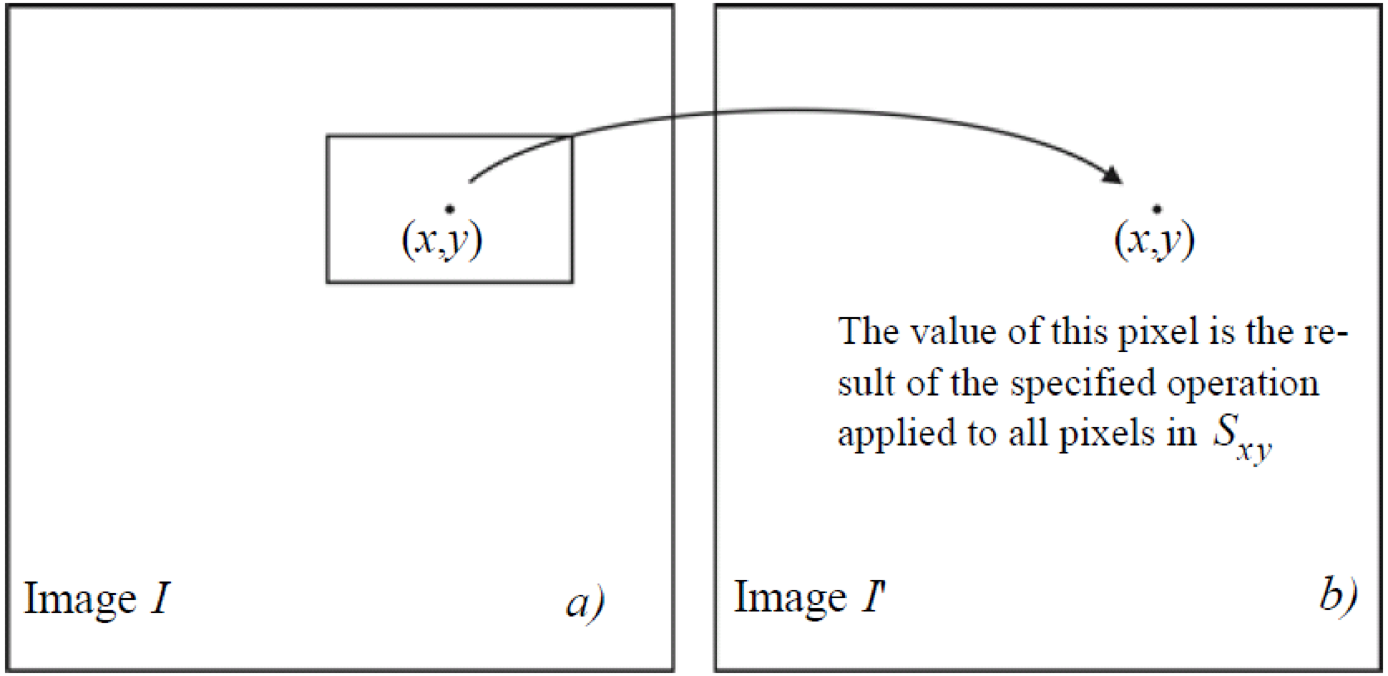
Lowpass filtered image

Highpass filtered image

Lena: Image processing standard test picture (512x512) since 1972

# Spatial filters

- Let *Sxy* denote the set of coordinates of a neighborhood centered on an arbitrary point (x,y) in an image *I*
- Spatial filtering generates a corresponding pixel at the same coordinates in an output image *I'* where the value of that pixel is determined by a specified operation on the pixels in *Sxy*




- For example, an averaging filter is:

$$I' = \frac{1}{mn} \sum_{(r,c) \in S_{xy}} I(r,c)$$

# Linear filters

- Filter *w* of size *m* x *n* :

$$I'(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) \cdot I(x+s, y+t)$$

  where *m=2a+1* and *n=2b+1* usually assumed odd integers.
- *w* is also called **kernel**, **mask**, or **window**.

- linear filtering: the process of moving a filter mask over the entire image and computing the sum of products at each location.
- Also called:
  - *correlation with the kernel w*
  - *convolution with the kernel w*

- Expressed in compact way as

$$I'(x,y) = w(x,y) * I(x,y)$$

  where * denotes the operator of convolution

# Smoothing filters

- Constant averaging:

$$w = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- Gaussian averaging:

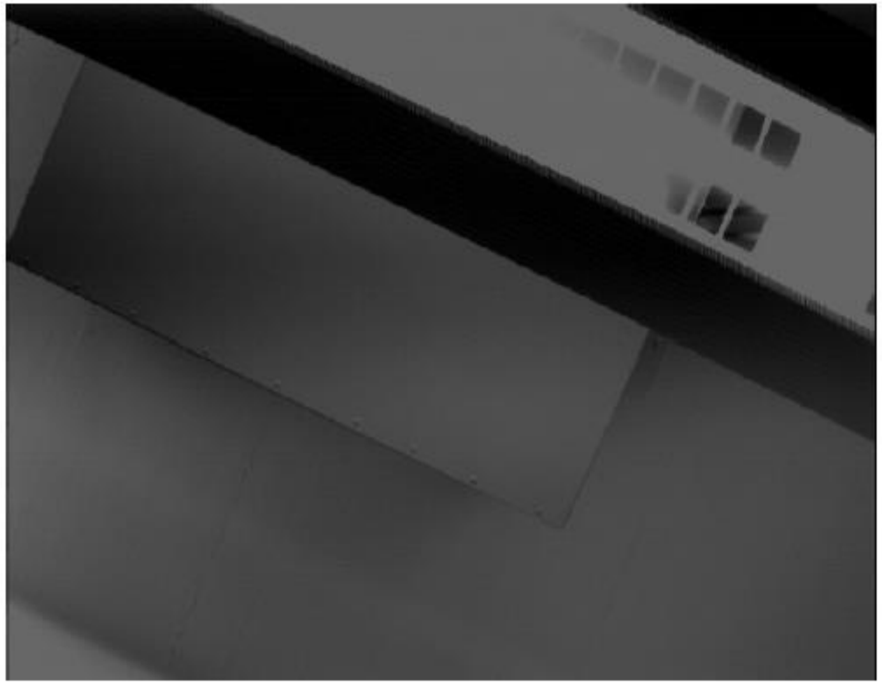
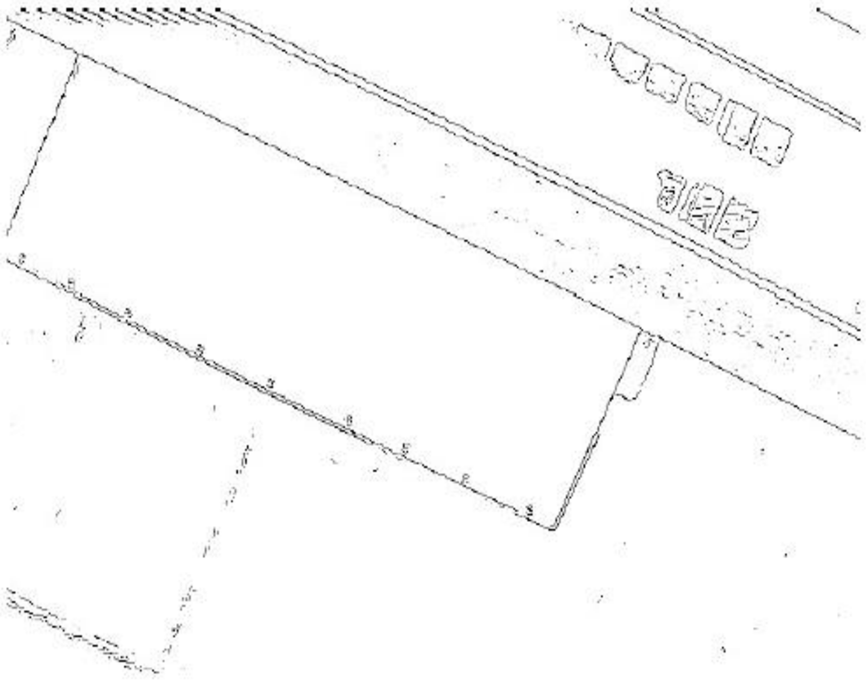
$$G_\sigma(x, y) = e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

$$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

This example was generated with a 21x21 mask

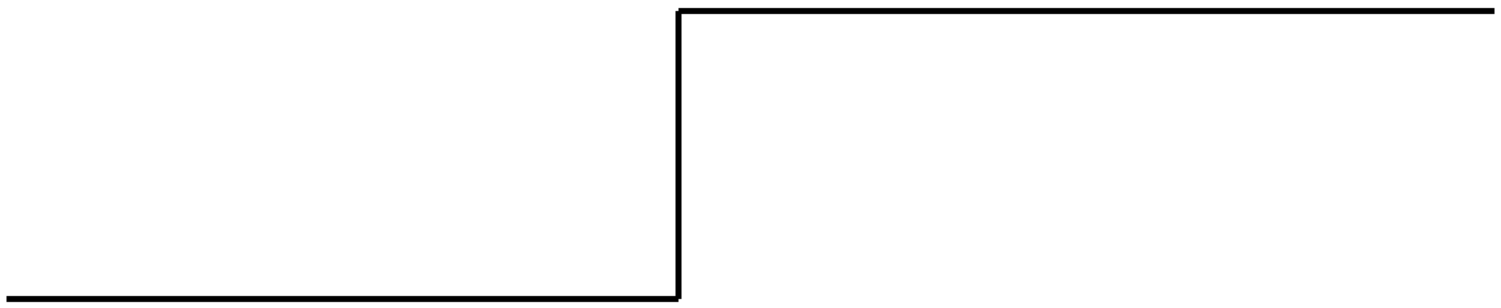
# Edge Detection

- Ultimate goal of edge detection
  - an idealized line drawing.
- Edge contours in the image correspond to important scene contours.

# Edge is Where Change Occurs

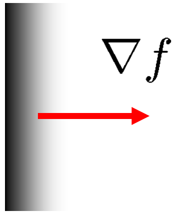
- Edges correspond to sharp changes of intensity
- Change is measured by 1st derivative in 1D
- Biggest change, derivative has maximum magnitude
- Or 2nd derivative is zero.



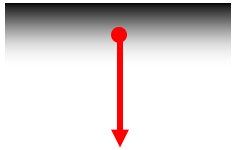
# Image gradient

- The gradient of an image: $$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

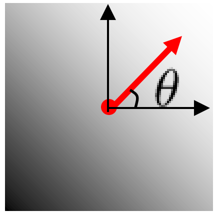
- The gradient points in the direction of most rapid change in intensity



$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$



$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$




$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

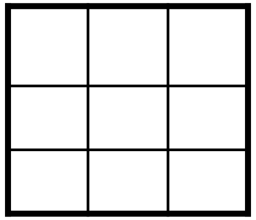
- The gradient direction is: $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$
- The gradient magnitude is: $\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

# The discrete gradient

- How can we differentiate a *digital* image f[x,y]?
  - Option 1: reconstruct a continuous image, then take gradient
  - Option 2: take discrete derivative (finite difference)

$$\frac{\partial f}{\partial x}[x, y] \approx f[x+1, y] - f[x, y]$$

- How to implement this as a spatial filter?



*w*

# Gradient Edge Detectors

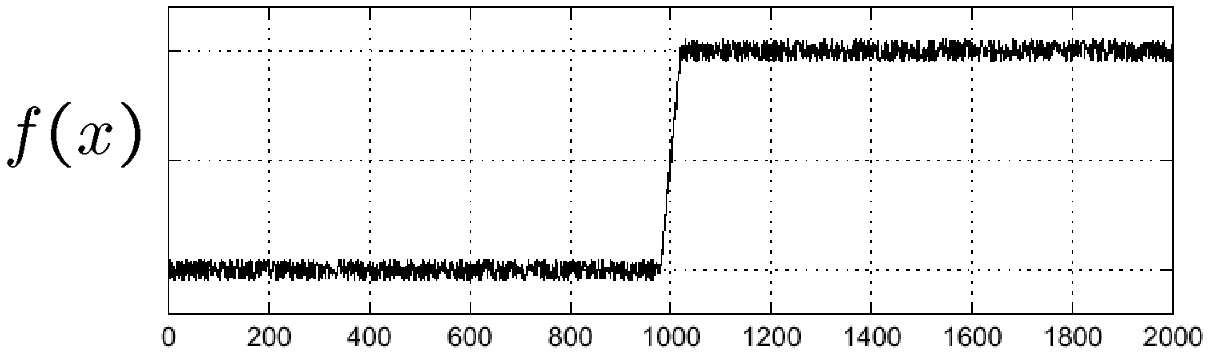
- **Roberts** $|G| \cong \sqrt{r_1^2 + r_2^2}$ ; $r_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ ; $r_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$

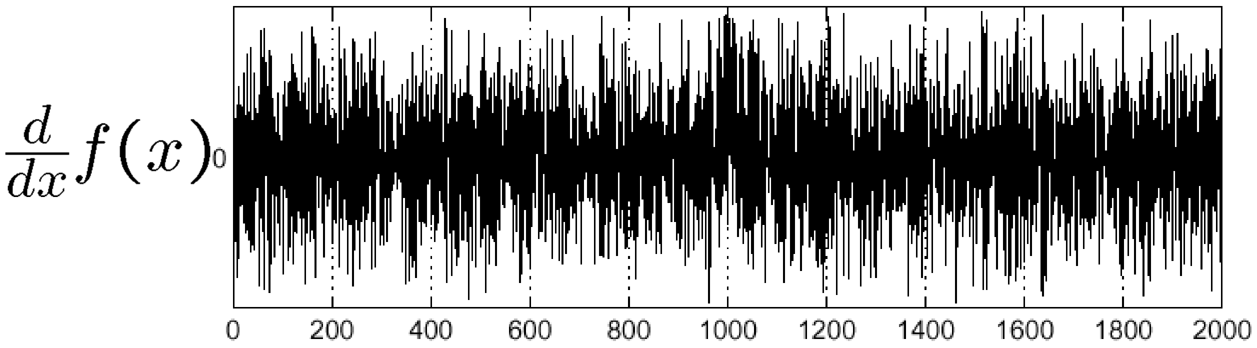
- **Prewitt** $|G| \cong \sqrt{p_1^2 + p_2^2}$ ; $\theta \cong \operatorname{atan}\left(\frac{p_1}{p_2}\right)$ ; $p_1 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ ; $p_2 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

- **Sobel** $|G| \cong \sqrt{s_1^2 + s_2^2}$ ; $\theta \cong \operatorname{atan}\left(\frac{s_1}{s_2}\right)$ ; $s_1 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$ ; $s_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

# Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal




- Where is the edge?

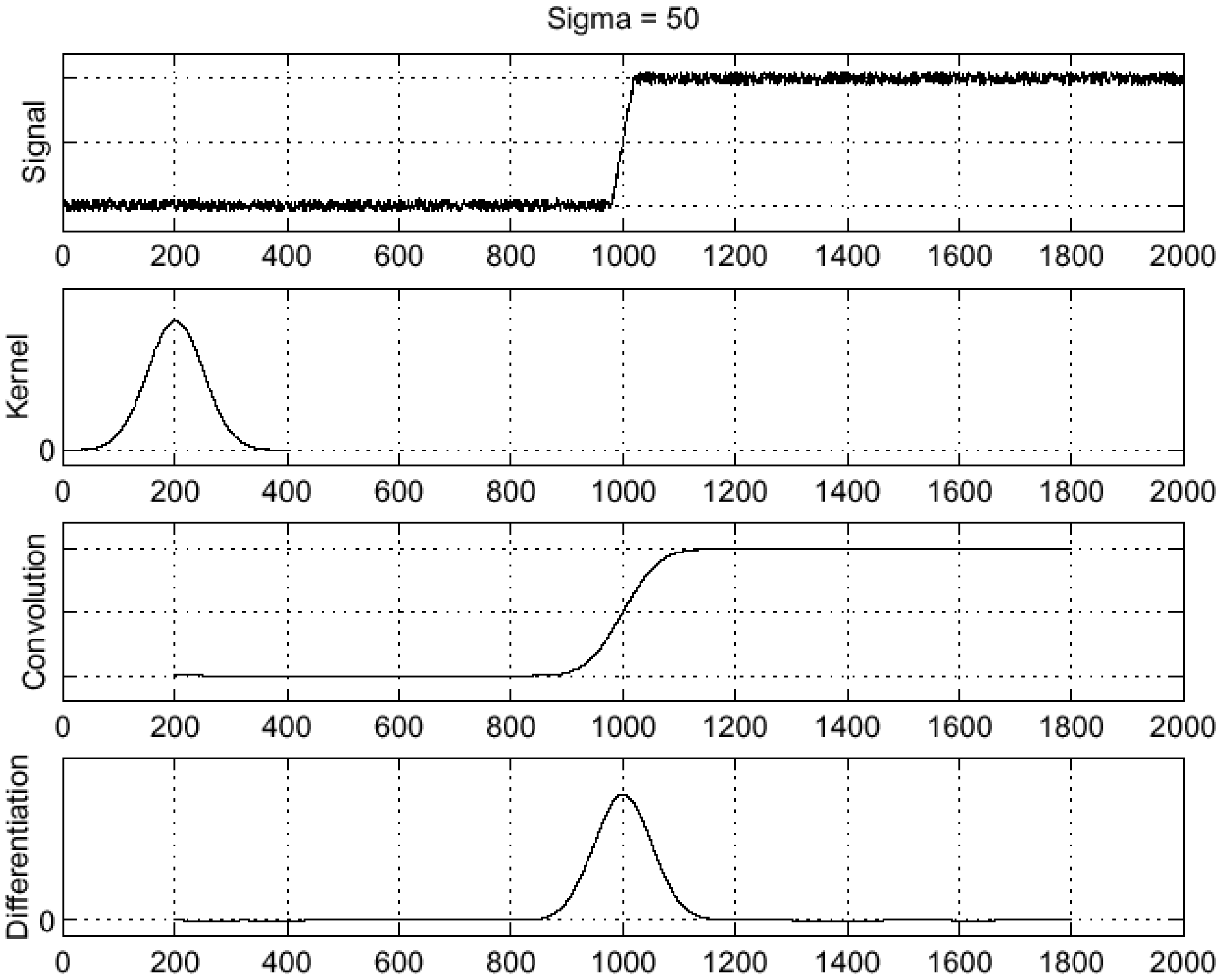
# Solution: smooth first

$f$

$h$

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$




- Where is the edge?
- Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

# Derivative theorem of convolution

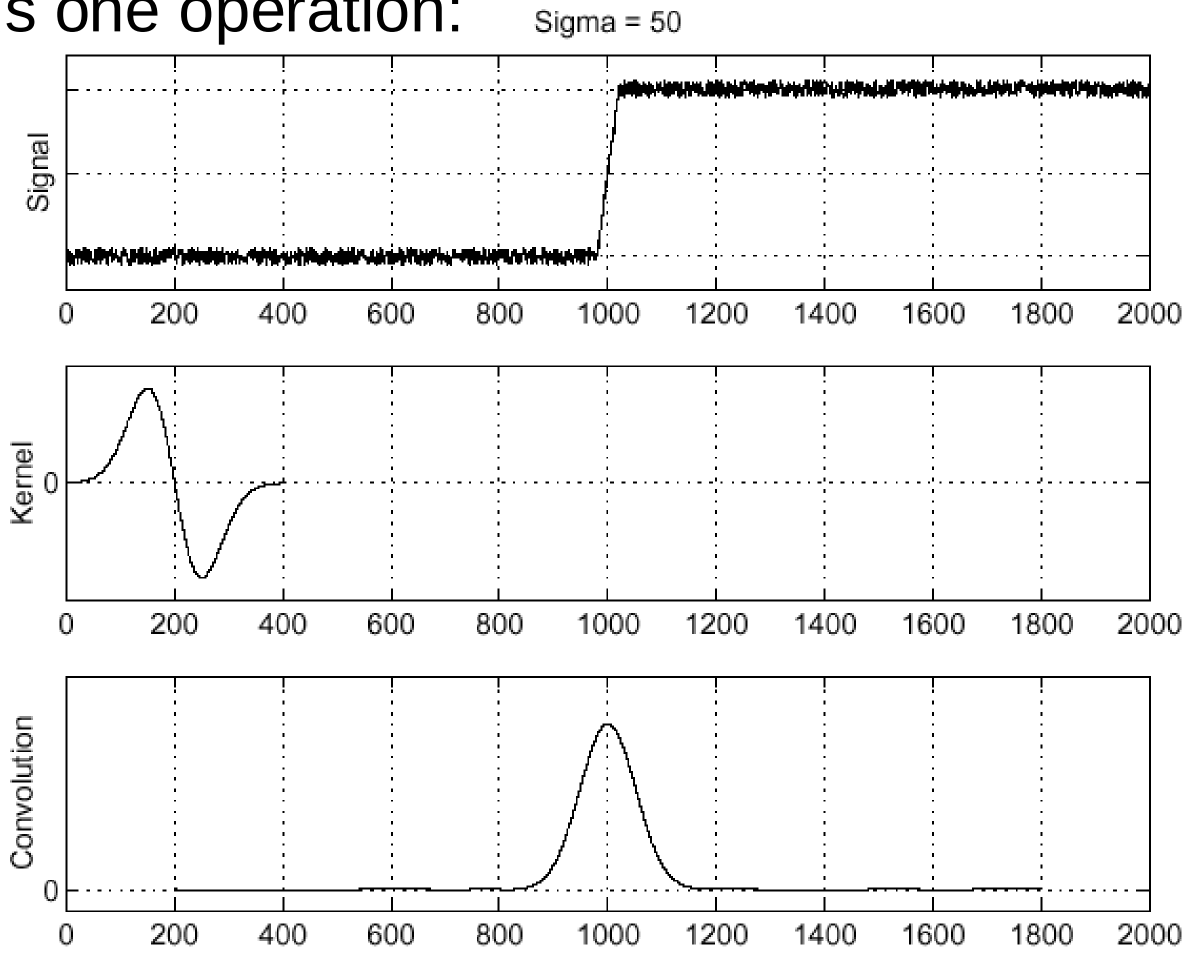
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

- This saves us one operation:

$f$

$\frac{\partial}{\partial x}h$

$(\frac{\partial}{\partial x}h) \star f$



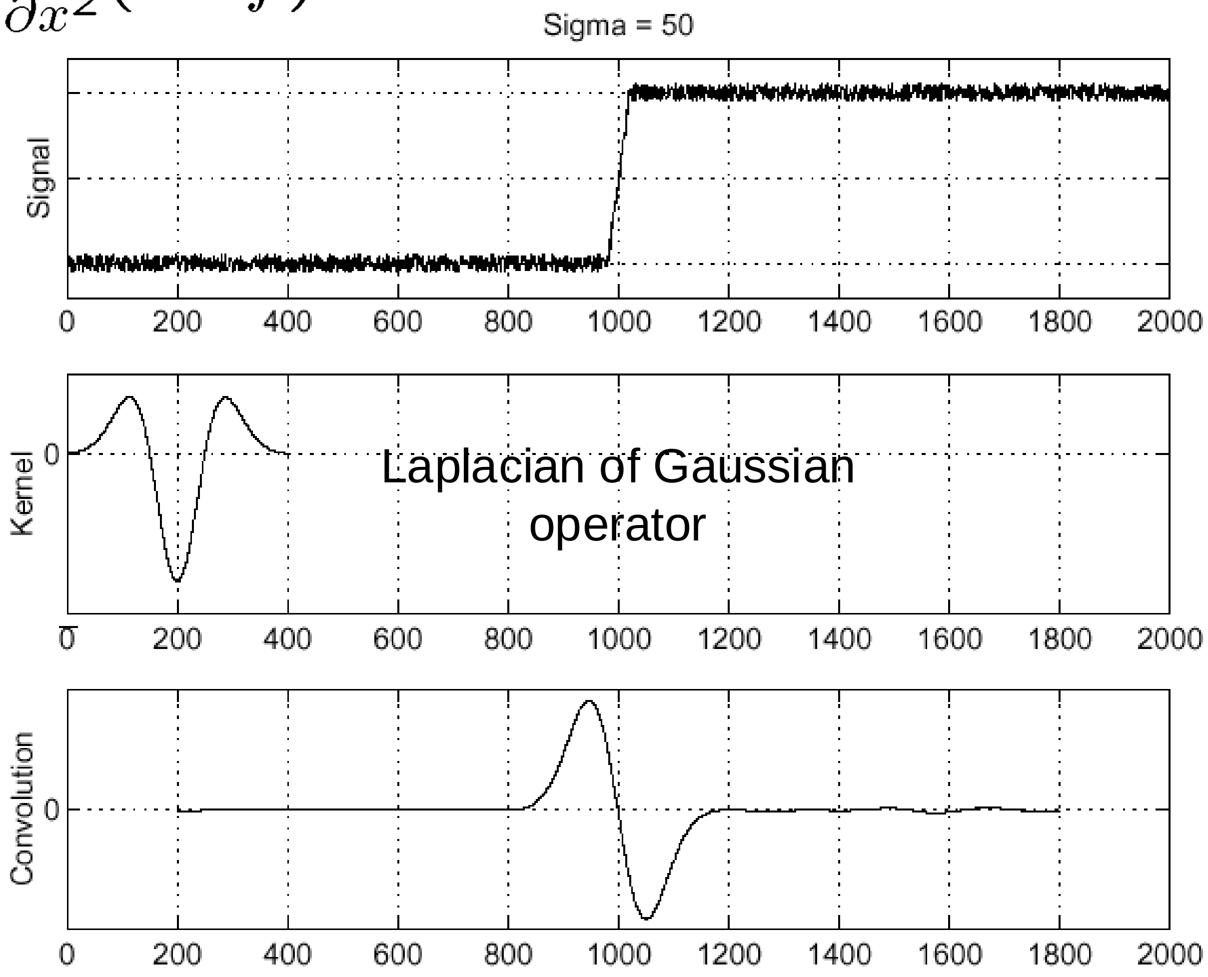
# The Canny Edge Detector

- Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

$f$

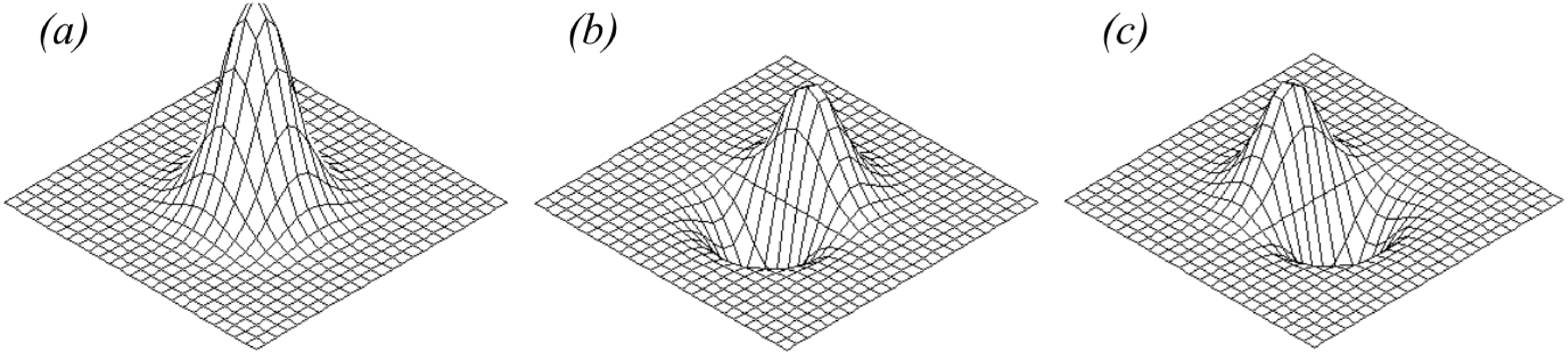
$\frac{\partial^2}{\partial x^2}h$

$(\frac{\partial^2}{\partial x^2}h) \star f$




- Where is the edge?
- Zero-crossings of bottom graph

# 2D Canny edge detector




$$G_\sigma(x, y) = G_\sigma(x)G_\sigma(y) \qquad f_V(x, y) = G'_\sigma(x)G_\sigma(y) \qquad f_H(x, y) = G'_\sigma(y)G_\sigma(x)$$

- Two perpendicular filters:
  - Convolve imge $I(x, y)$ with $f_V(x, y)$ and $f_H(x, y)$ – obtaining $R_V(x, y)$ and $R_H(x, y)$
  - Use square of gradient magnitude: $R(x, y) = R_V^2(x, y) + R_H^2(x, y)$
  - Mark peaks in $R(x, y)$ above a threshold

# The Sobel edge detector

original image (Lena image)

# The Sobel edge detector

norm of the gradient

# The Sobel edge detector

thresholding

# The Sobel edge detector

thinning

(non-maxima suppression)