Simulation, Optimization and Mathematical Libraries

Course: Robotic Programming Environments

Michał Drwięga
michal.drwiega@pwr.edu.pl
www.mdrwiega.com/edu/rpe

Department of Cybernetics and Robotics
Wrocław Univeristy of Science and Technology

Wrocław University
of Science and Technology

Content

- Simulation
- ODE Solving
- Optimization
- Linear Algebra

Introduction to Robotics Simulation

What is simulation?

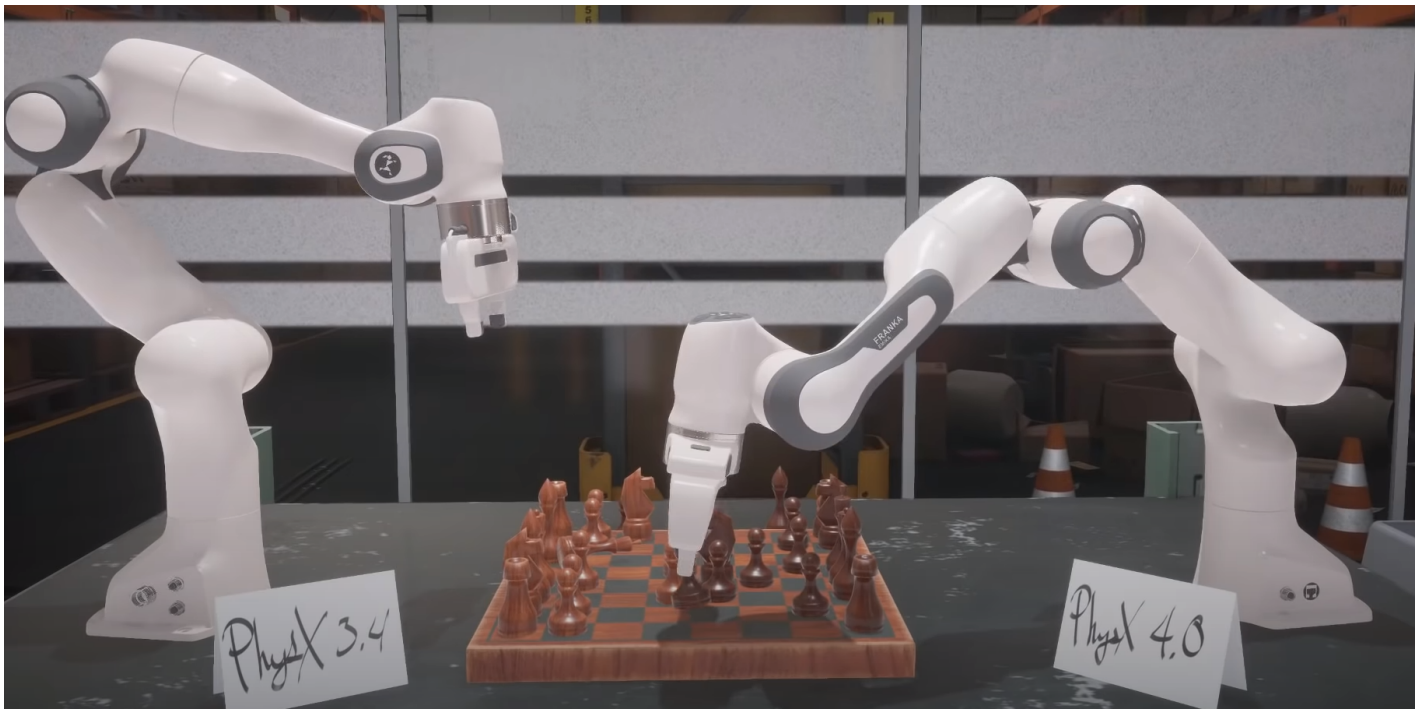
- Simulation imitates the behavior of real systems or processes using models.
- A model represents the behavior and characteristics of a specific process or system.
- Simulation shows how the model changes under various conditions over time.

Why systems are simulated?

- Reduces financial risk.
- Allows for repeated testing under identical conditions.
- Enables generation of synthetic data.
- Examines long-term impacts.

Introduction to Simulation

- **Robotics Simulation:** The process of mimicking the behavior of a robot in a virtual environment.
- **Purpose:** To model, analyze, and test robot systems before physical implementation.



1

[1] https://blogs.nvidia.com/

Applications of Robotics Simulation

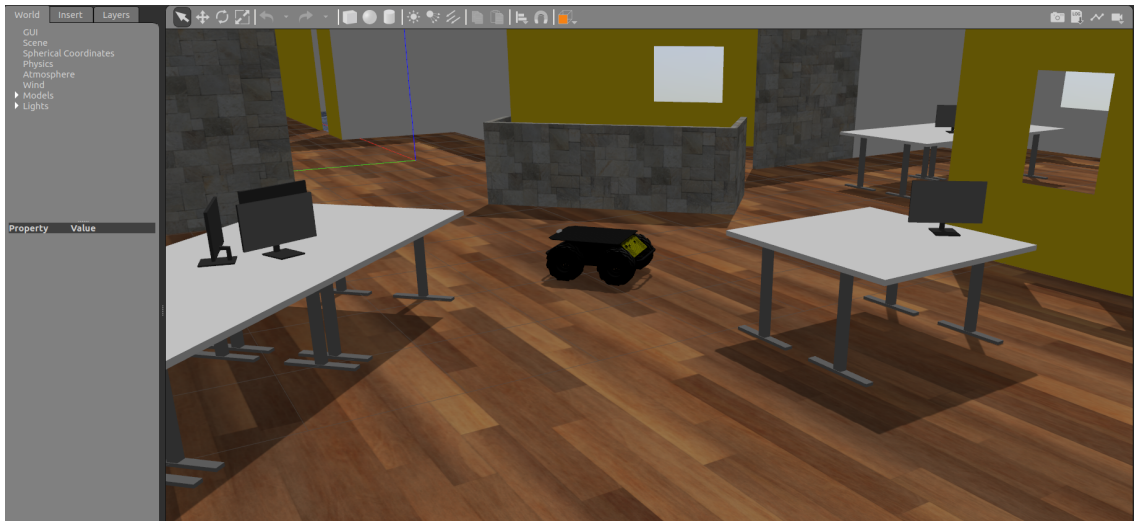
- **Algorithm Development:** Testing control algorithms in a safe and controlled environment.
- **Prototyping:** Designing and iterating robot models before physical construction.
- **Training:** Training machine learning models for perception and decision-making.
- **Verification and Validation:** Validating the performance of robotic systems.

Advantages of Robotics Simulation

- **Cost-Efficiency:** Reduces the cost of physical prototyping and testing.
- **Safety:** Allows testing in a controlled environment without risk to humans or equipment.
- **Iterative Design:** Facilitates rapid iteration and refinement of robot designs.
- **Scenario Testing:** Enables testing in various scenarios and environmental conditions.

Key Components of Robotics Simulation

- **Robot model:** Mathematical representation of the robot.
- **Sensor models:** Emulation of sensors like cameras, lidars, IMUs, etc.
- **Algorithms:** Perception, planning, control algorithms.
- **Environment:** Virtual representation of the robot's surroundings.
- **Physics engine:** Simulation software that calculates the interactions between objects in the virtual world.



2

[2] https://www.clearpathrobotics.com/assets/guides/kinetic/husky/additional_sim_worlds

Physics engines

- Simulates physical phenomena in a virtual environment.
- Used for realistic simulations in gaming, engineering, and animation.
- **Components:**
  - Rigid Body Dynamics
  - Collision Detection
  - Integration Methods
- **Popular Engines:** Bullet Physics[3], Nvidia PhysX[4], ODE (Open Dynamics Engine)[5]
- **Challenges:** Overcoming computational complexity and accuracy issues.
- **Future Developments:** Trends include improved real-time simulations and AI integration.

[3] https://github.com/bulletphysics/bullet3
[4] https://developer.nvidia.com/physx-sdk
[5] https://www.ode.org/

Popular Robotics Simulation Tools

- **Gazebo:** An open-source simulation tool widely used in the robotics community.
- **Nvidia Isaac Sim**: A simulation environment focused on the physics simulation and acceleration based on GPU computations.
- **O3DE (Open 3D Engine):** General purpose robotics simulator.
- **V-REP (CoppeliaSim):** Versatile Robot Experimentation Platform with extensive features.
- **Webots:** A development environment used for modeling, simulation, and control of mobile robots.
- **CARLA:** Mostly for autonomous vehicles purposes.
- **Unity:** A game development engine increasingly used for robot simulation.
- **MuJoCo (Multi-Joint dynamics with Contact)** is a physics engine used for simulating highly detailed and dynamic robotics and biomechanical systems.

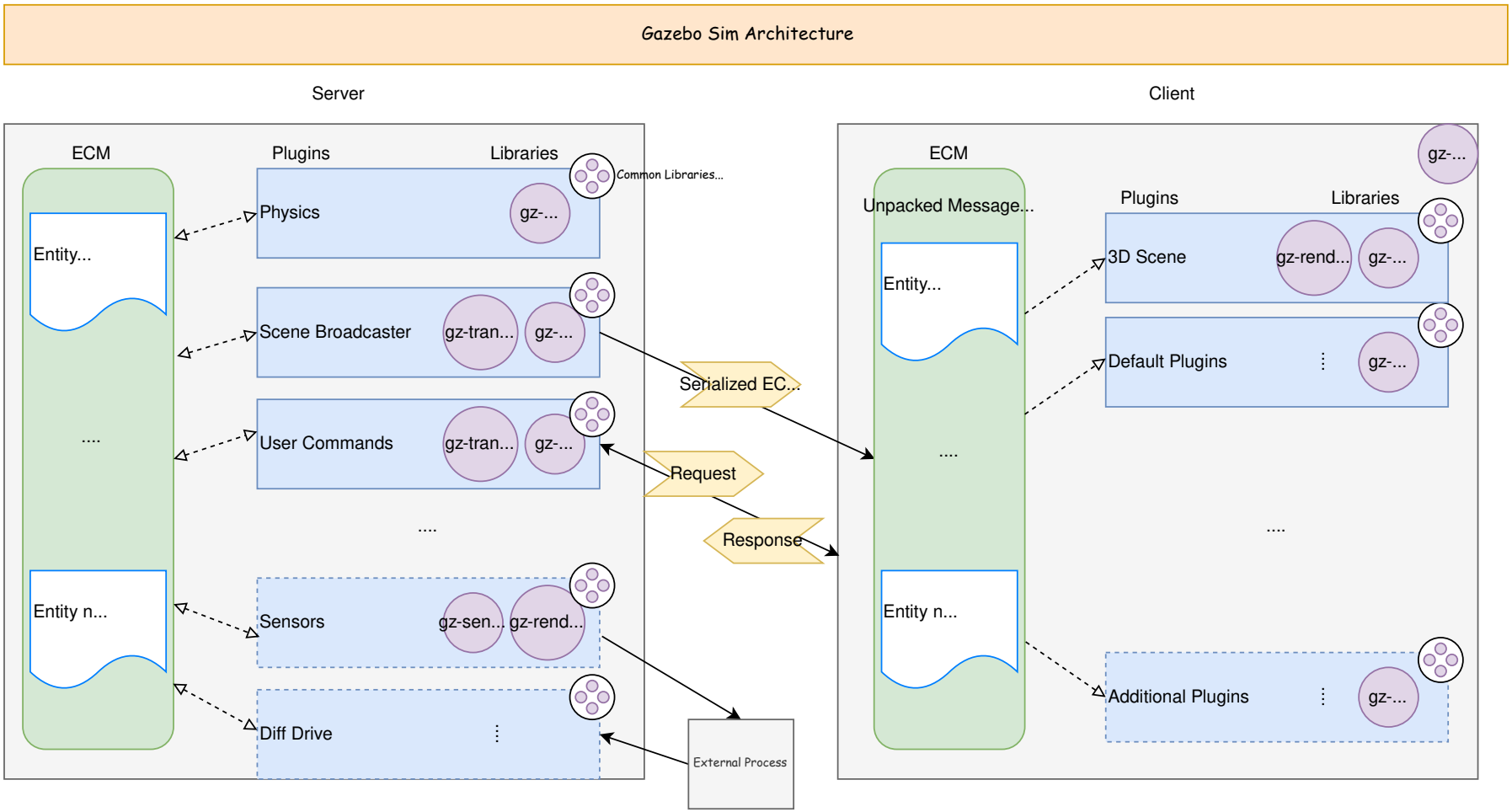
Gazebo

- **Introduction**
  - Gazebo is an open-source robotics simulation software.
  - Developed by the Open Source Robotics Foundation (OSRF).
- **Features**
  - Dynamics simulation: works with multiple high-performance physics engines.
  - 3D visualization for modeling and testing robot designs.
  - Extensive library of sensors (cameras, lidars, RGB-D sensors, IMU, GPS, etc.), actuators models, and environments.
  - TCP/IP Transport: Run simulation on remote servers.

6

[6] https://github.com/gazebosim/gz-sim

Gazebo Architecture



[7]

ECM = Entity Component Manager

[7] https://gazebosim.org/docs/latest/architecture/

Introduction to URDF

What is URDF?

- **Definition:** URDF (Unified Robot Description Format) is a standard XML format for describing a robot's structure, kinematics, and dynamics.
- **Purpose:** Used for modeling and visualizing robots, aiding in simulation and control.

Features

- **Standardization:** Provides a standardized format for robot descriptions.
- **Interoperability:** URDF is supported by different ROS tools.
- **Simulation:** Facilitates accurate simulation of robot behavior (URDF is supported by multiple simulation environments).

URDF: Capabilities and Limitations

Capabilities

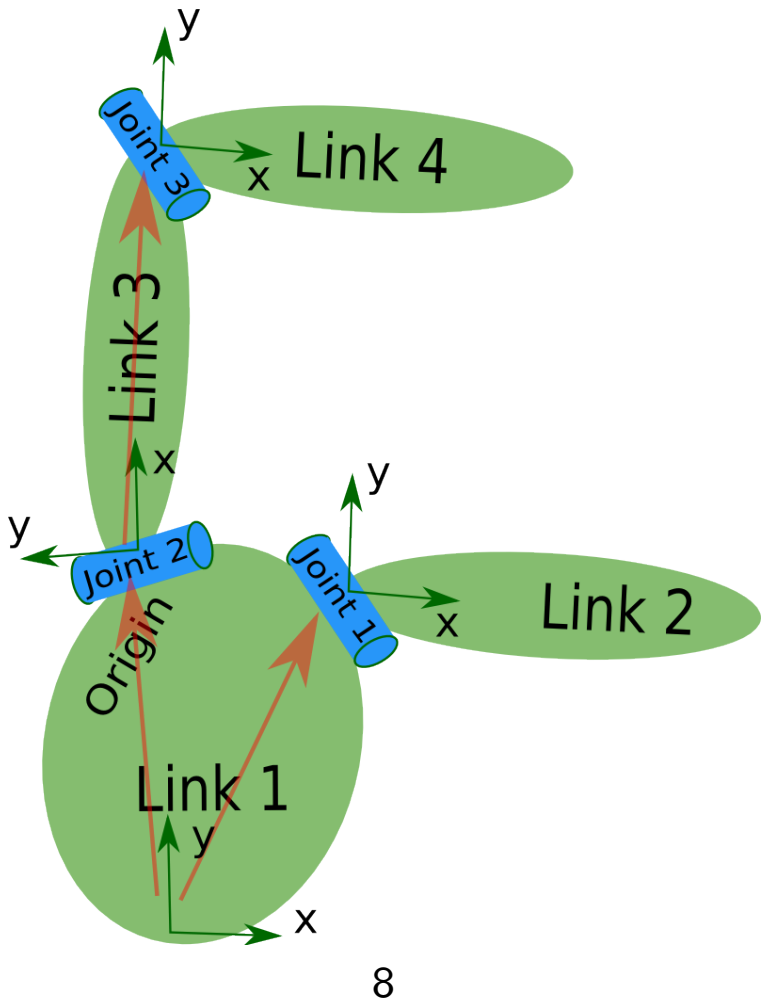
- **Kinematic and dynamic description**: Defines the robot's physical properties, such as mass, center of mass, and inertia for each link.
- **Visual representation**: Specifies the robot's visual appearance using 3D meshes and materials for rendering.
- **Collision model**: Simplifies geometry to define the robot's collision boundaries, useful in simulation and planning.

Limitations

- **Tree structure only**: It cannot represent closed kinematic chains or parallel robots.
- **No support for flexible elements**: Does not support flexible or deformable parts.
- **Limited dynamics**: Lacking advanced dynamics like damping or friction coefficients.
- **Static visual model**: Visual properties cannot change dynamically, limiting use in scenarios where the robot appearance may vary.

Key Components of URDF

- **Link:** Represents a rigid body in the robot.
- **Joint:** Defines the connection between two links and their relative motion.
- **Material:** Specifies visual and collision properties.
- **Transmission:** Describes how joint efforts result in link movements.



8

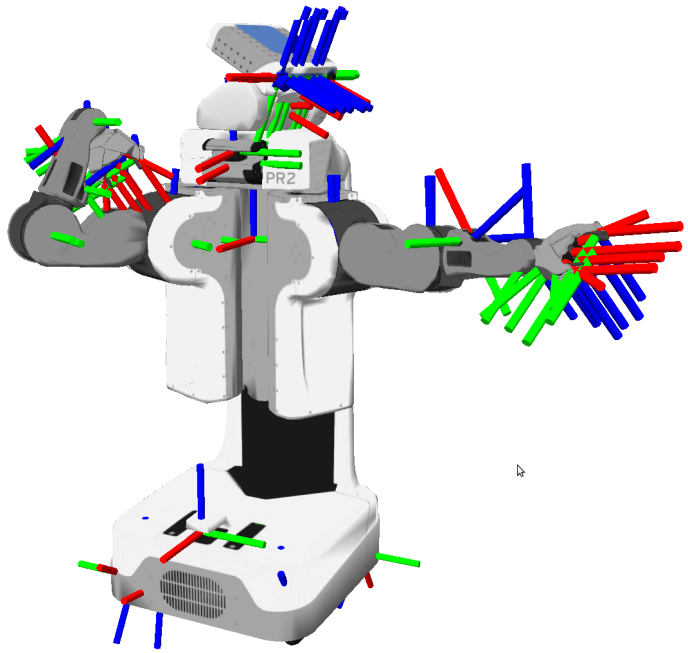
[8] https://wiki.ros.org/urdf/XML/model

URDF example

```
<robot name="simple_arm">
  <link name="base_link"/>

  <link name="arm_link">
    <visual>
      <geometry>
        <box size="0.1 0.1 0.5"/>
      </geometry>
    </visual>
  </link>

  <joint name="arm_joint" type="revolute">
    <parent link="base_link"/>
    <child link="arm_link"/>
    <origin xyz="0 0 0.25" rpy="0 0 0"/>
    <axis xyz="0 0 1"/>
    <limit lower="-1.57" upper="1.57" effort="10" velocity="
        1.0"/>
  </joint>
</robot>
```

ROS 2 Robot Description Publisher

- `robot_state_publisher` is a ROS 2 package used to publish the state of a robot's URDF model to **tf2**.
- It publishes the URDF description and joint states to enable visualization and interaction.
- `robot_state_publisher` uses the URDF specified by the parameter robot_description and the joint positions from the topic joint_states.



9

[9] https://wiki.ros.org/robot_state_publisher

Introduction to ODEs

Definition

An Ordinary Differential Equation (ODE) is an equation that involves one or more derivatives of an unknown function with respect to one independent variable. It describes how a system changes over time.

Example

The general form of a first-order ODE is:

$$\frac{dy}{dx} = f(x, y)$$

Importance

ODEs are widely used to model various real-world phenomena such as population dynamics, mechanical systems, and of course robotics systems.

Application of ODE in robotics - simulation

The dynamics model of the underwater unmanned vehicle (UUV)

$$M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) = \tau$$

where:

M : inertia matrix,

$C(\nu)$: matrix of Coriolis and centripetal terms

$D(\nu)$: damping matrix

$g(\eta)$: vector of gravitational forces and moments

ν : velocities in local frame

η : generalized pose in global frame

τ : vector of control inputs

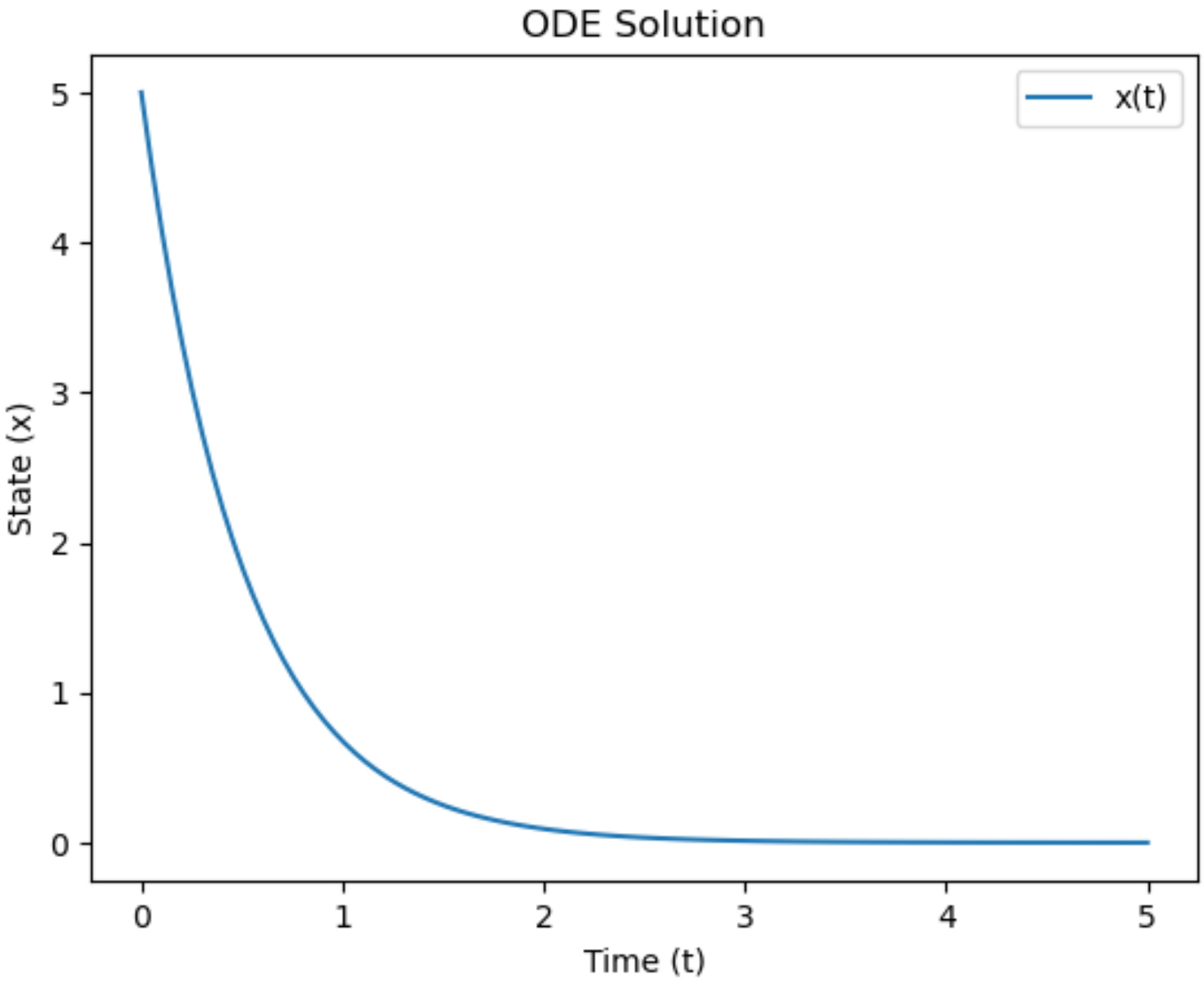
ODE Solving with Python SciPy library

- **Features**
  - Open-source library for mathematics, science, and engineering.
  - Provides the *solve_ivp* function for ODE integration.
  - Built on the 'ODEPACK' (Fortran) library for efficient and accurate integration.
- **Applications**
  - Widely used in physics, biology, engineering, and various scientific fields.
  - Essential for simulating and analyzing dynamic systems.

Example - ODE solving with SciPy

```python
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# ODE dot(x) = f(t, x, p), where
# t -- time, x -- state, p -- parameter
def f(t, x, p):
    return -p * x

T = [0, 5] # Time horizon
x0 = [5] # Initial state
p = 2.0 # Parameter

# Solve
sol = solve_ivp(f, T, x0, args=[p])

# Plot the solution
plt.plot(sol.t, sol.y[0], label='x(t)')
plt.title('ODE Solution')
plt.xlabel('Time (t)'); plt.ylabel('State (x)')
plt.legend(); plt.show()
```

Example - ODE solving with SciPy



Optimization

Definition

Optimization - The act of making something as good as possible[10]

- **Types:** Unconstrained, Constrained, Global, Local.
- **Algorithms:** Gradient Descent, Genetic Algorithms, Simulated Annealing, Particle Swarm, etc.
- **Applications:** ML, Operations Research, Engineering, Finance.
- **Challenges:** Non-convexity, High-dimensionality.
- Popular optimization tools
  - Ceres Solver
  - g2o
  - CasADI
  - ...

[10] https://dictionary.cambridge.org/dictionary/english/optimization

Ceres Solver: A large scale non-linear optimization

- **Introduction**
  - Ceres Solver[11] is an open-source C++ library for modeling and solving large, complex optimization problems.
  - Developed by Google and named after the dwarf planet Ceres.
- **Features**
  - Handles a wide range of optimization problems (unconstrained and constrained).
  - It's robust - it handles outliers and noisy data effectively.
  - Utilizes advanced optimization algorithms for fast convergence.
  - Open source - released under the BSD license.
- **Applications**
  - **Robotics:** Sensor fusion, SLAM.
  - **Computer Vision:** Image registration and feature matching.
  - **Autonomous Vehicles:** Path planning and sensor calibration.

[11] http://ceres-solver.org/

Ceres - Example usage

Problem definiton

Non-linear least squares (NLLS) problem

$$\frac{1}{2}\sum_{i}||f_i(x_{i_1},\ldots,x_{i_k})||^2$$

Find the minimum of the function

$$\frac{1}{2}(10-x)^2$$

Ceres - Example usage

Write a functor in the form $f(x) = 10 - x$

```
1  struct CostFunctor {
2      template <typename T>
3      bool operator()(const T* const x, T* residual) const {
4          residual[0] = T(10.0) - x[0];
5          return true;
6      }
7  };
```

12

[12] http://ceres-solver.org/nnls_tutorial.html

Ceres - Example usage

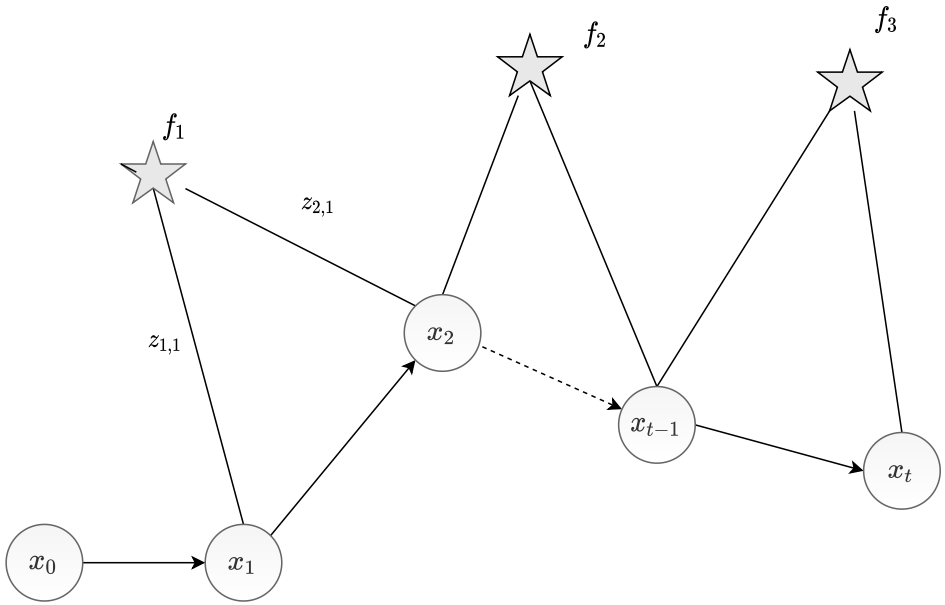
```
#include <ceres/ceres.h>

int main() {
  double initial_x = 5.0;
  double x = initial_x;

  ceres::Problem problem;

  CostFunction* cost_function = new AutoDiffCostFunction<
      CostFunctor, 1, 1>(new CostFunctor);
  problem.AddResidualBlock(cost_function, nullptr, &x);

  ceres::Solver::Options options;
  ceres::Solver::Summary summary;
  ceres::Solve(options, &problem, &summary);

  std::cout << summary.BriefReport() << "\n";
  std::cout << "x : " << initial_x << " -> " << x << "\n";

  return 0;
}
```

Optimization - g2o framework

- **g2o (general graph optimization):**[14] Open-source C++ framework for optimizing graph-based nonlinear error functions.
- **Graph Optimization:** Represents optimization problems as a graph of nodes and edges.

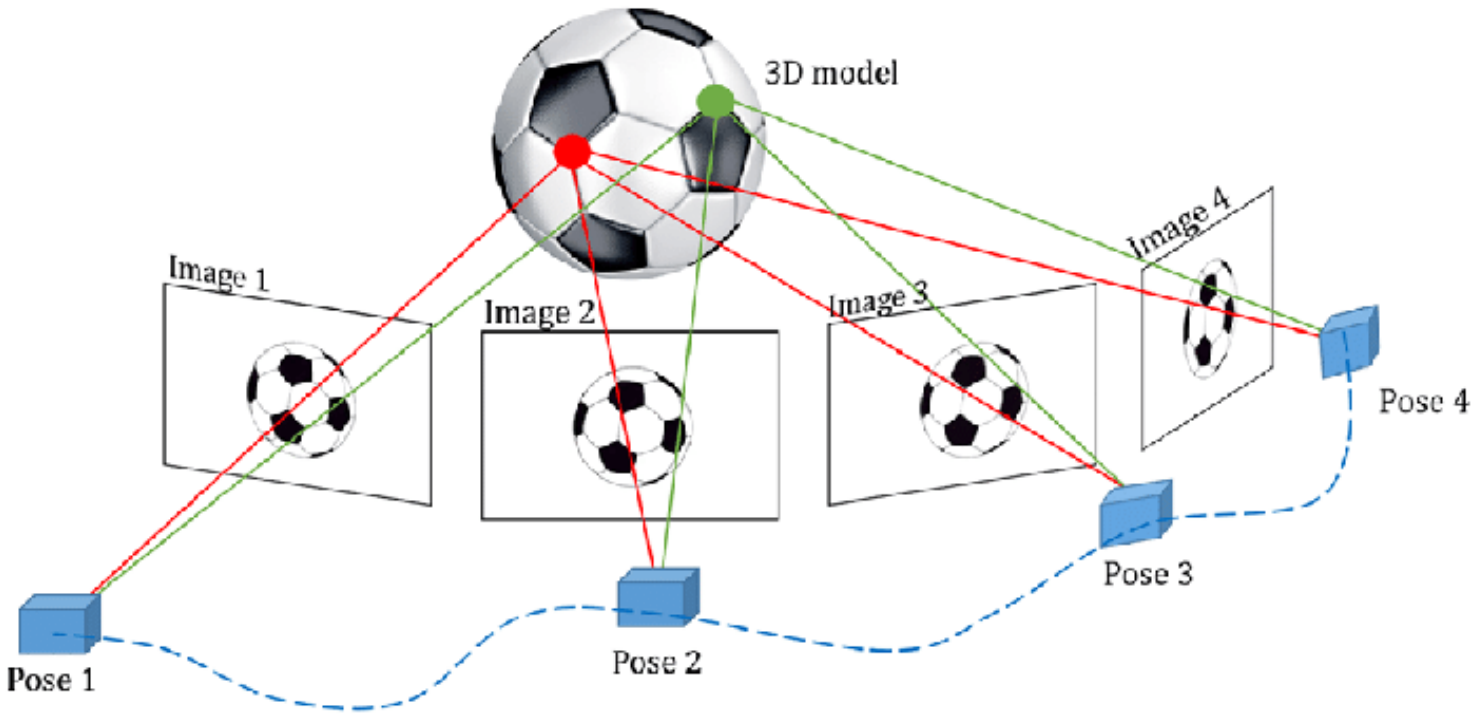
[14] http://ais.informatik.uni-freiburg.de/publications/papers/kuemmerle11icra.pdf

Applications of g2o



- **Simultaneous Localization and Mapping (SLAM):** Optimization of robot poses and map.
- **Structure from Motion:** Estimation of 3D scene structure from 2D images.
- **Robotics:** Trajectory optimization and sensor fusion.
- **Bundle Adjustment:** Refinement of camera poses and 3D structure in computer vision.

Boundle adjustment



15

[15] R. Azzam et al. Feature-based visual simultaneous localization and mapping: a survey

g2o - example usage - step 1 - optimizer initialization

```
1  int main() {
2    // Create a sparse optimizer
3    g2o::SparseOptimizer optimizer;
4
5    // Set up the solver and algorithm
6    typedef g2o::BlockSolver<g2o::BlockSolverTraits<3, 1>>
         BlockSolverType;
7    typedef g2o::LinearSolverDense<BlockSolverType::
         PoseMatrixType> LinearSolverType;
8
9    auto linearSolver = g2o::make_unique<LinearSolverType>();
10   auto solverPtr = g2o::make_unique<BlockSolverType>(std::
         move(linearSolver));
11   auto algorithm = new g2o::OptimizationAlgorithmLevenberg(
         std::move(solverPtr));
12
13   optimizer.setAlgorithm(algorithm);
```

g2o - example usage - step 2 - add vertices and edges

```
1  // Add a 3D point vertex
2  auto pointVertex = new g2o::VertexPointXYZ();
3  pointVertex->setId(0);
4  pointVertex->setEstimate(Eigen::Vector3d(1.0, 2.0, 3.0));
5  optimizer.addVertex(pointVertex);
6  // Add a SE3 (Pose) vertex
7  auto poseVertex = new g2o::VertexSE3();
8  poseVertex->setId(1);
9  poseVertex->setEstimate(g2o::SE3Quat());
10 optimizer.addVertex(poseVertex);
11
12 // Add an edge
13 auto edge = new g2o::EdgeSE3PointXYZ();
14 edge->setId(0);
15 edge->setVertex(0, poseVertex);
16 edge->setVertex(1, pointVertex);
17 edge->setMeasurement(Eigen::Vector3d(1.0, 2.0, 3.0));
18 edge->setInformation(Eigen::Matrix3d::Identity());
19 optimizer.addEdge(edge);
20
21 // Optimize
```

g2o - example usage - step 3 - optimize

```
1   // Optimize
2   optimizer.initializeOptimization();
3   optimizer.optimize(10);
4
5   // Print optimized point position
6   std::cout << "Optimized point position: " << pointVertex->
        estimate().transpose() << std::endl;
7
8   return 0;
9 }
```

Introduction to CasADi

CasADi:[16] An open-source symbolic framework for nonlinear optimization and algorithmic differentiation.

Features:

- **Symbolic expressions:** Define mathematical models symbolically.
- **Algorithmic differentiation:** Efficient computation of derivatives.
- **Interfaces:** Integrates with various numerical libraries and solvers.

Applications:

- **Optimal control:** Design and control of dynamic systems.
- **Robotics:** Trajectory optimization and kinematics.
- **System identification:** Model validation and parameter identification.

[16] https://web.casadi.org/

CasADi - example - optimization with constraints

Function:

$$f(x, y, z) = x^2 + 100z^2 + y$$

Constraint:

$$g(x, y, z) = z + (1 - x)^2 - y = 0$$

CasADi - example - optimization with constraints

```
1  # Symbols/expressions
2  x = MX.sym('x')
3  y = MX.sym('y')
4  z = MX.sym('z')
5  f = x**2+100*z**2
6  g = z+(1-x)**2-y
7
8  nlp = {}                 # NLP declaration
9  nlp['x']= vertcat(x,y,z) # decision vars
10 nlp['f'] = f             # objective
11 nlp['g'] = g             # constraints
12
13 # Create solver instance
14 F = nlpsol('F','ipopt',nlp);
15
16 # Solve the problem using a guess
17 F(x0=[2.5,3.0,0.75],ubg=0,lbg=0)
```

Linear algebra - Eigen library

What is Eigen?

- C++ template library for linear algebra.
- Designed for high-performance matrix and vector operations.

Key features

- **Expressiveness:** Intuitive syntax for linear algebra operations.
- **Performance:** Emphasis on efficient and fast computations.
- **Versatility:** Supports various matrix and vector types.
- **Ease of integration:** Header-only library.
- **Open Source:** Released under the MPL2 license.

Applications of Eigen

- **Robotics:** Used for kinematics, dynamics, and control computations.
- **Computer Vision:** Efficient matrix operations for image processing.
- **Machine Learning:** Supporting linear algebra in data analysis.
- **Scientific Computing:** Widely employed in numerical simulations.

Linear algebra - Eigen example

```cpp
#include <iostream>
#include <Eigen/Dense>

using namespace std;

int main() {
    Eigen::MatrixXd A(2,2);
    A << 1, 2,
         3, 4;

    cout << "Matrix A:\n" << A << endl;
    cout << "Determinant: " << A.determinant() << endl;
    cout << "Inverse:\n" << A.inverse() << endl;

    cout << "A*A: " << A*A << endl;

    return 0;
  }
```

Introduction to `numpy.linalg`

Overview

`numpy.linalg` is a subpackage in NumPy that provides a collection of linear algebra functions.

Key features

- Linear equation solving
- Eigenvalue problems
- Singular value decomposition
- Matrix and vector operations

numpy.linalg - example: solving linear equations

Solve the system of linear equations $Ax = B$ where

$$A = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 \\ 7 \end{bmatrix}$$

```
1  A = np.array([[2, 1], [3, 4]])
2  B = np.array([5, 7])
3  x = np.linalg.solve(A, B)
```

Summary

- Simulation: Gazebo, URDF
- ODE solving: scipy
- Optimization: ceres, g2o, CasADi
- Linear algebra: Eigen, numpy.linalg

The end

Thank you for your attention!