AAMA Simulation Environment

Dogukan Altay

MICSS Lab

Agenda

- Who am I?
- What is ROS 2?
- What is Gazebo?
- AAMA-Sim
- Q&A

Who Am I?

- Worked in defence industry as a software development and R&D lead. (Selvi Technology)
- Currently working as a front-end developer in Jobilla
- Have been a Software Engineer Master’s student in UAntwerpen since 2021.
- Working on my thesis about implementing a ROS 2 framework for Multi Agent Systems (like JADE).

ROS 2

What is ROS 2

- ROS 2 (Robot Operating System 2) is an open-source, meta-operating system for your robot.
- It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.
- A set of software libraries and tools that help you build robot applications that work across a wide variety of robotic platforms.

Evolution from ROS 1

- ROS 2 is not just an upgrade from ROS 1 but a complete rewrite that retains the core concepts while improving on aspects like security, reliability, and scalability.
- Addressing the limitations of ROS 1, ROS 2 introduces changes such as support for real-time computing and a more flexible communication system.

ROS 2 Main Features

- **Multi-platform support**: Runs on Linux, macOS, and Windows.
- **Real-time capability**: Enhanced to meet the real-time requirements of robotics applications.
- **Improved security**: Incorporates tools and techniques for securing data and communications.
- **Quality of Service (QoS) settings**: Offers control over the way messages are delivered and processed.

- ROS 2 has two "sides"
    - A suite of user contributed packages that implement common robot functionality such as SLAM, planning, perception, vision, manipulation, etc.
    - Mostly useful, open-sourced libraries that helps the entry effort for such applications.
    - Creates an environment of development with the same paradigm.

ROS 2 Main Features

ROS 2 Paradigm

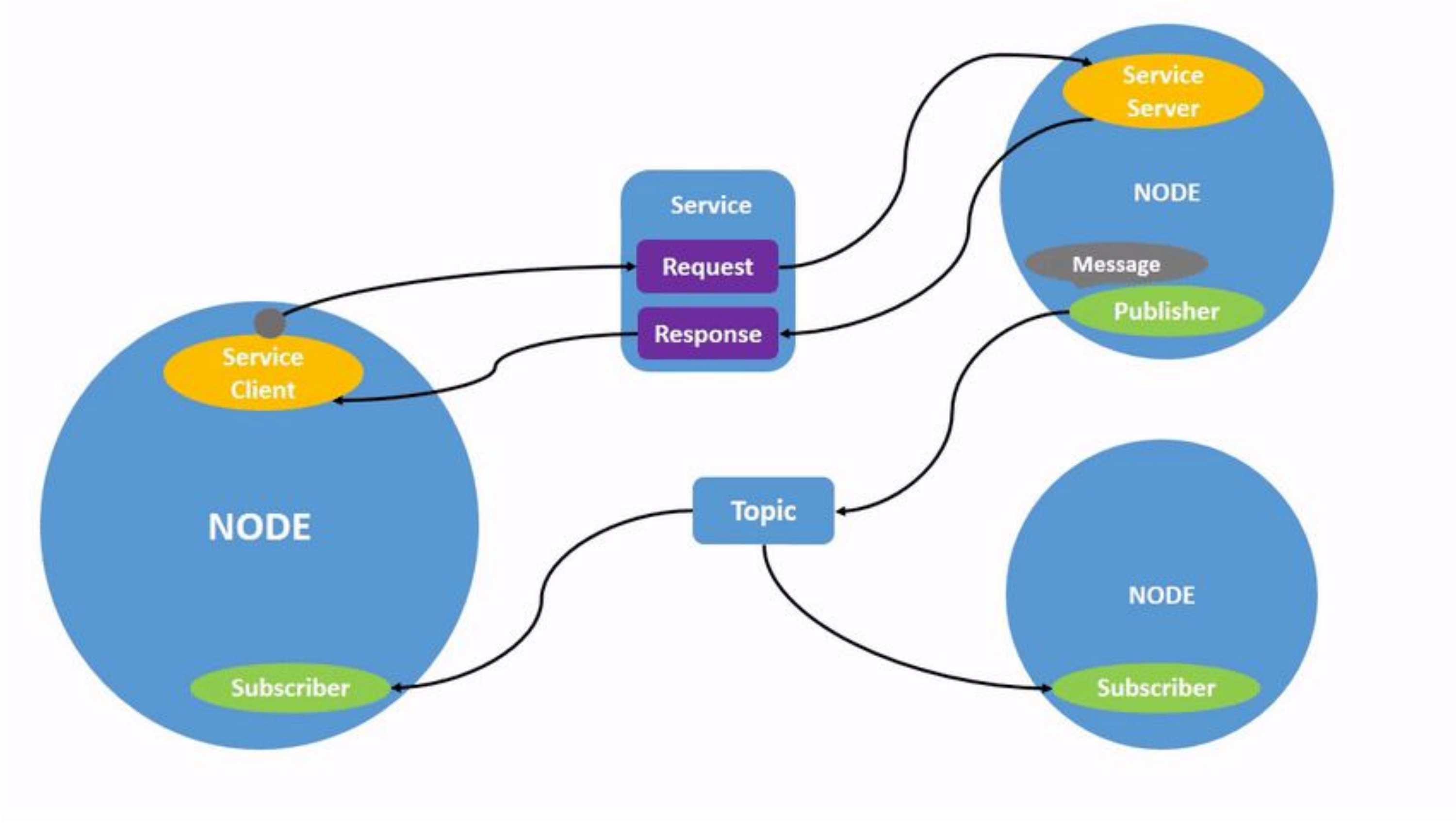
Peer to Peer

- ROS 2 systems consist of many small programs (nodes) which connect to each other and continuously exchange messages

Tools-based

- There are many small, generic programs that perform tasks such as visualization, logging, plotting data streams, etc

- Multi-Lingual
  - ROS 2 software modules can be written in any language for which a interface library has been written. Currently interface libraries exist for C++, Python, LISP, Java, JavaScript, MATLAB, Ruby, and more.
- Thin
  - The ROS 2 conventions encourage contributors to create stand-alone libraries/packages and then wrap those libraries so they send and receive messages to/from other ROS 2 modules.



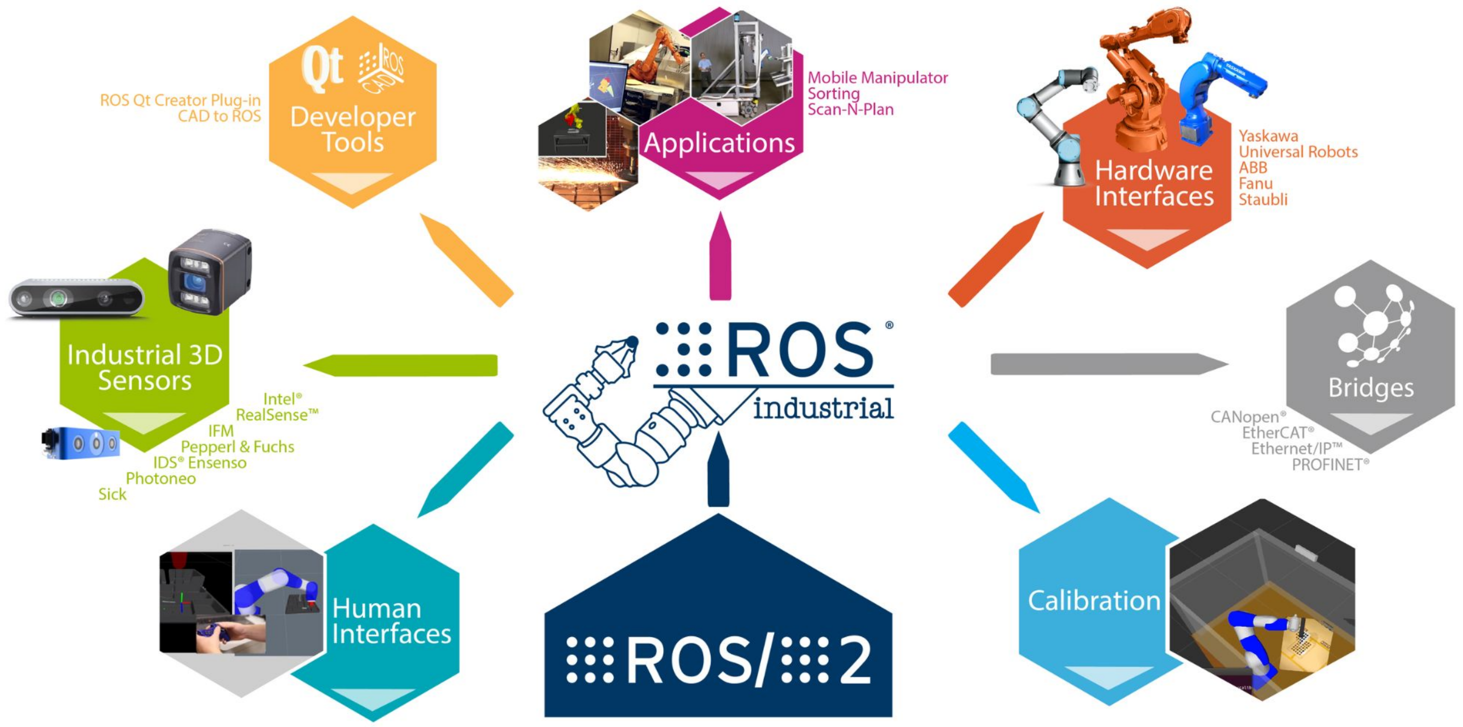
ROS Core Concepts

- Nodes
- Messages and Topics
- Services
- Actions
- Packages and Stacks

ROS Topics and ROS Messages

- Topic: named stream of messages with a defined type
    - Data from a range-finder might be sent on a topic called scan, with a message of type LaserScan
- Nodes communicate with each other by publishing messages to topics
- Publish/Subscribe model: 1-to-N broadcasting
- Messages: Strictly-typed data structures for internode communication

ROS Ecosystem



Limitations of ROS

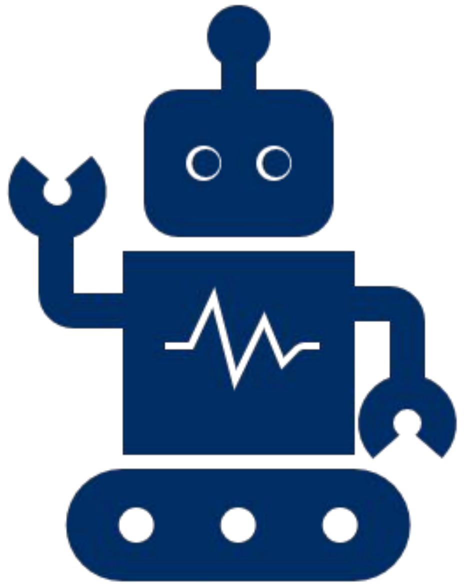
- **Partial Real-Time Support**: Despite improvements over ROS 1, ROS 2 still may not fully meet the real-time requirements of some advanced robotics applications.
- **Dependence on Underlying Middleware**: Real-time capabilities are largely dependent on the DDS implementation, which varies among providers.
- **Third-Party Integrations**: Integration with some third-party tools and libraries might require additional effort due to changes in the ROS 2 architecture
- **System Resources**: ROS 2 might require more computational resources compared to ROS 1, which could be a constraint for low-power or embedded systems.

Limitations of ROS

- **Network Bandwidth**: The use of DDS for communication can lead to higher bandwidth usage, which might be a limitation in bandwidth-constrained environments.
- **Ecosystem Growth**: While growing, the ROS 2 ecosystem is still catching up to the extensive range of packages and tools available in ROS 1.
- **Community Transition**: The transition from ROS 1 to ROS 2 in the community is ongoing, which means that some resources and expertise are still centered around ROS 1.

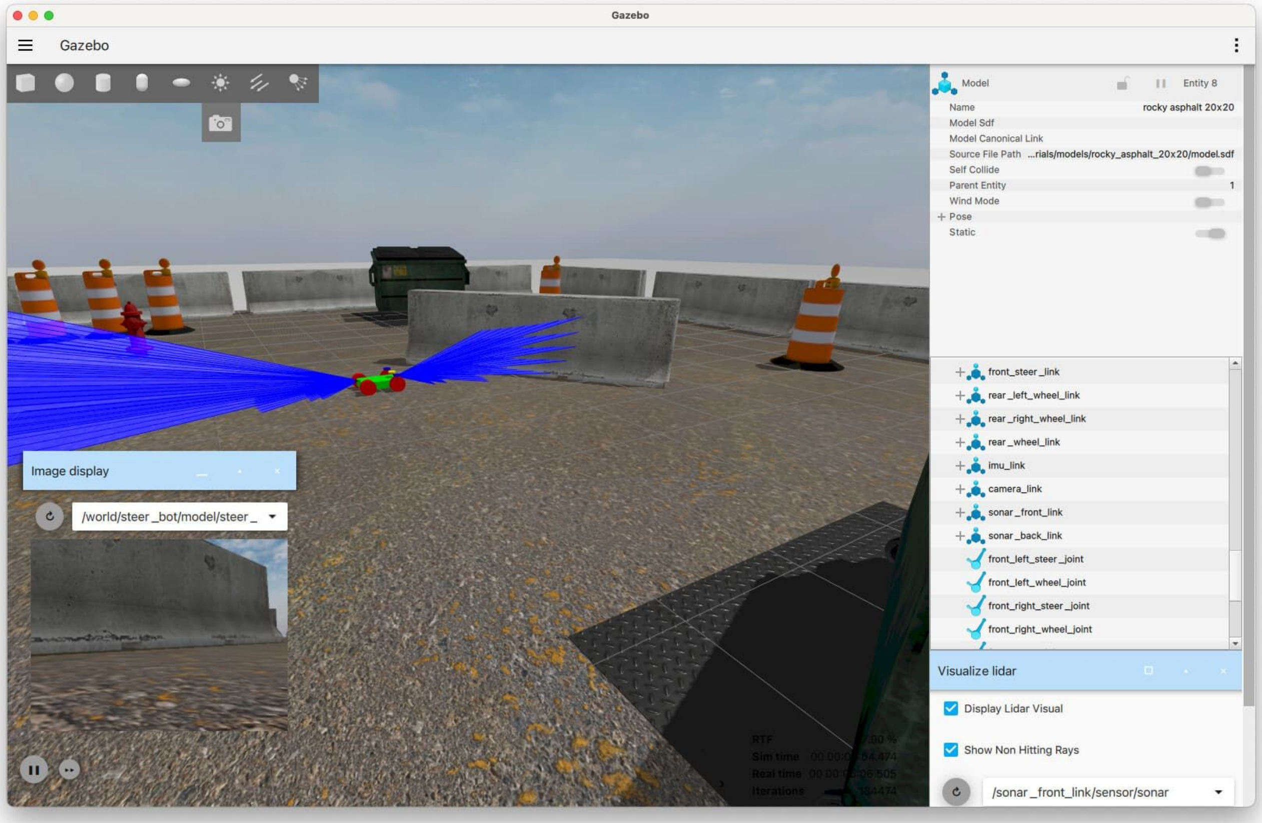
Gazebo

What is Gazebo



- Gazebosim is a robotics simulation software.
- It is open source.
- Provides a realistic physics engine, high-quality graphics, and convenient programmatic and graphical interfaces.
- Gazebo is commonly used in robotics research and education to help robots learn new tasks, evaluate their performance, and test new algorithms.

Gazebo GUI



/world/steer _bot/model/steer _

Model — Entity 8

Name: rocky asphalt 20x20
Model Sdf
Model Canonical Link
Source File Path: ...rials/models/rocky_asphalt_20x20/model.sdf
Self Collide
Parent Entity: 1
Wind Mode
+ Pose
Static

```
+ front_steer_link
+ rear_left_wheel_link
+ rear_right_wheel_link
+ rear_wheel_link
+ imu_link
+ camera_link
+ sonar_front_link
+ sonar_back_link
  front_left_steer_joint
  front_left_wheel_joint
  front_right_steer_joint
  front_right_wheel_joint
```

[✓] Display Lidar Visual
[✓] Show Non Hitting Rays
/sonar _front_link/sensor/sonar

Gazebo Main Features

Gazebo has native support for ROS 2.

Gazebo has been implemented with pure CPP. As a result this creates a natural relationship with ROS 2 environment.

Gazebo provides vast range of ready-to-use sensors for robotic applications. Such as, LiDAR, camera sensors, ultrasonic sensors, IMUs, etc.

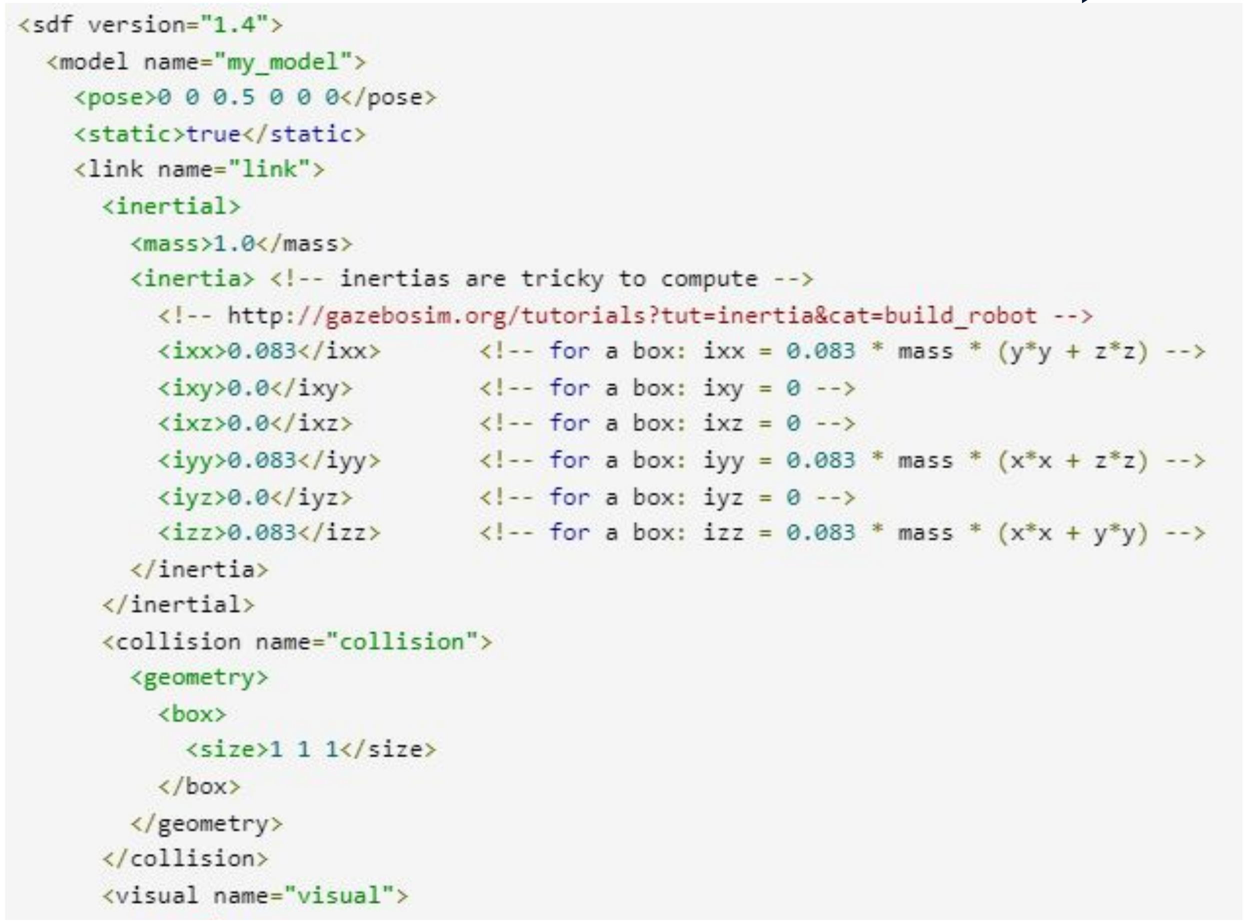
Gazebo also allows developers to implement custom plugins for its Model, Sensor and World entities. This feature allows us to customize the 3D engine for developers' different use cases.

Gazebo Model Definition

Gazebo models consists of:

- Links
- Joints
- Visuals
- Sensors
- Plugins

Detailed breakdown of a model specification can be found here.



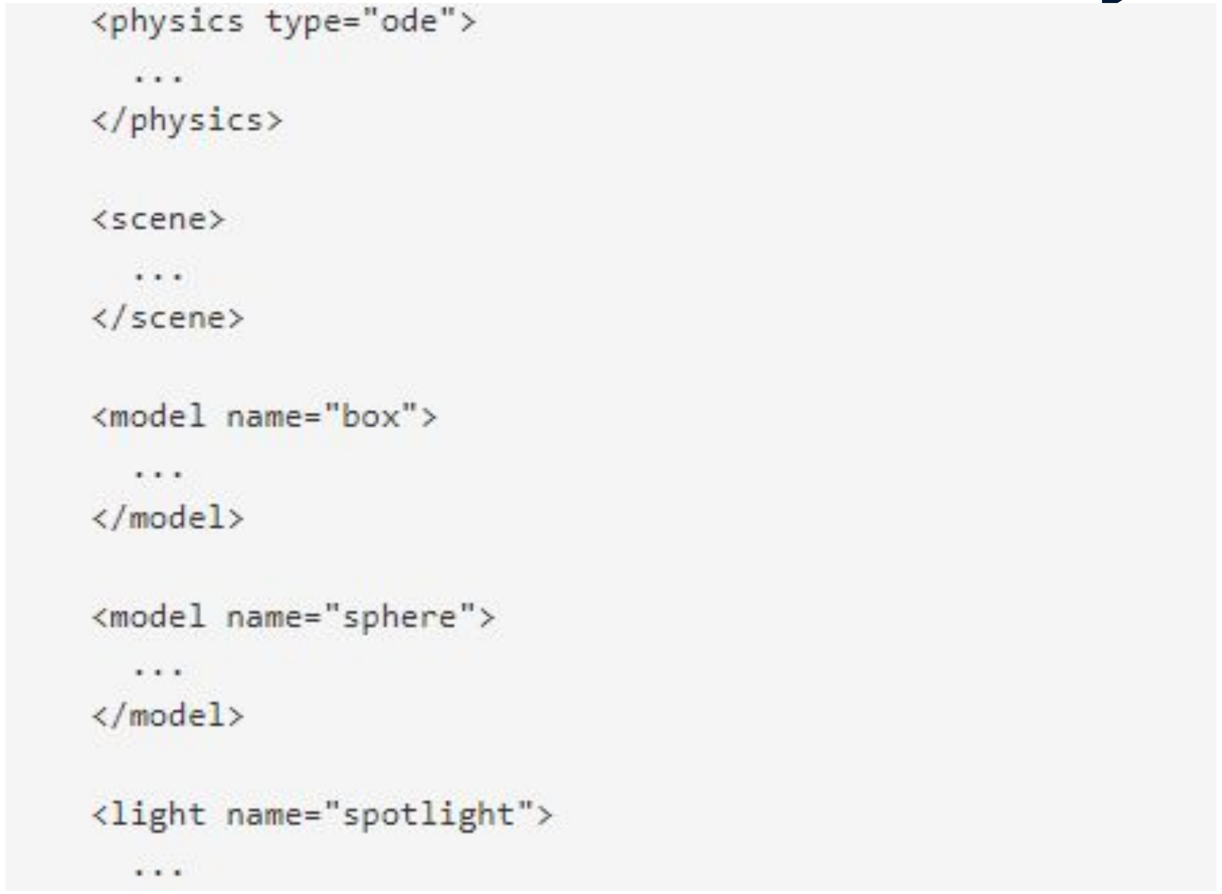
```
<sdf version="1.4">
  <model name="my_model">
    <pose>0 0 0.5 0 0 0</pose>
    <static>true</static>
    <link name="link">
      <inertial>
        <mass>1.0</mass>
        <inertia> <!-- inertias are tricky to compute -->
          <!-- http://gazebosim.org/tutorials?tut=inertia&cat=build_robot -->
          <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">
```

Gazebo World Definition

Gazebo World consists of:

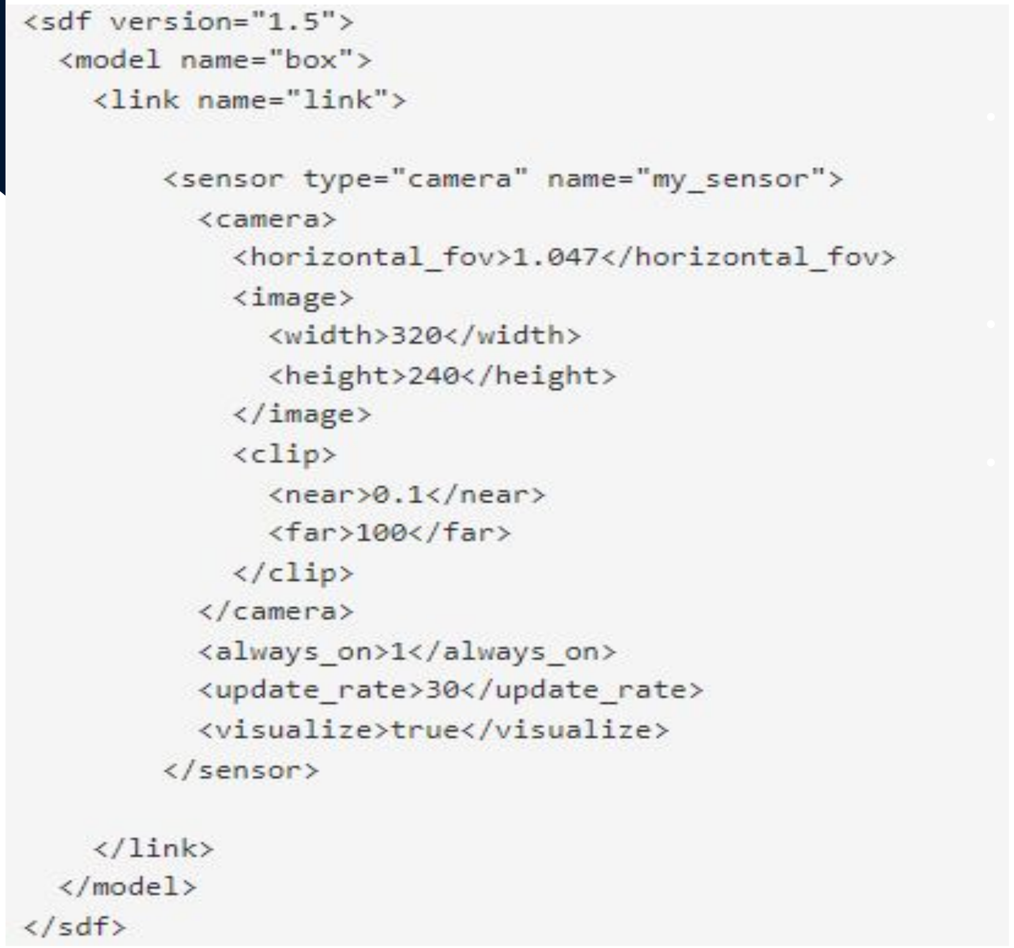
- Models
- Lights
- Physics
- Plugins

Detailed breakdown of a world specification can be found here.



```
<physics type="ode">
  ...
</physics>

<scene>
  ...
</scene>

<model name="box">
  ...
</model>

<model name="sphere">
  ...
</model>

<light name="spotlight">
  ...
```

Gazebo Sensor Definition



```
<sdf version="1.5">
  <model name="box">
    <link name="link">

        <sensor type="camera" name="my_sensor">
          <camera>
            <horizontal_fov>1.047</horizontal_fov>
            <image>
              <width>320</width>
              <height>240</height>
            </image>
            <clip>
              <near>0.1</near>
              <far>100</far>
            </clip>
          </camera>
          <always_on>1</always_on>
          <update_rate>30</update_rate>
          <visualize>true</visualize>
        </sensor>

    </link>
  </model>
</sdf>
```

- Gazebo Sensor can include vast range of sub definitions depending on the sensor type.
- An example camera sensor can be as such.
- Some built-in sensor can be found here but not limited.

How Gazebo and ROS Communicates

As we mentioned before, ROS 2 and Gazebo natively supports C++.

All custom plugins developed for Gazebo therefore can easily interface with the ROS 2 ecosystem with ease and creates a natural coupling between each tool.

Competitions with Gazebo ROS

- DARPA Robotics Challenge. Link
- Toyota Prius Challenge. Link
- DARPA Subterranean Challenge. Link

AAMA-Sim

What is AAMA-Sim

- AAMA-Sim is a development and testing environment that utilizes already existing tools within the ROS 2 and Gazebo ecosystems.
- The main purpose of this tool is to enable developers to develop Multi-Agent systems without the restrictions of real-life equipment.
- The simulation environment is also a testing suite for the upcoming AAMA-ROS framework for Multi-Agent applications.

AAMA-Sim Main Features

Simulation environment uses a basic robot with differential drive capabilities for robot entity. This decision allows us to benefit from an already existing and proven robot technology and its Gazebo simulation.

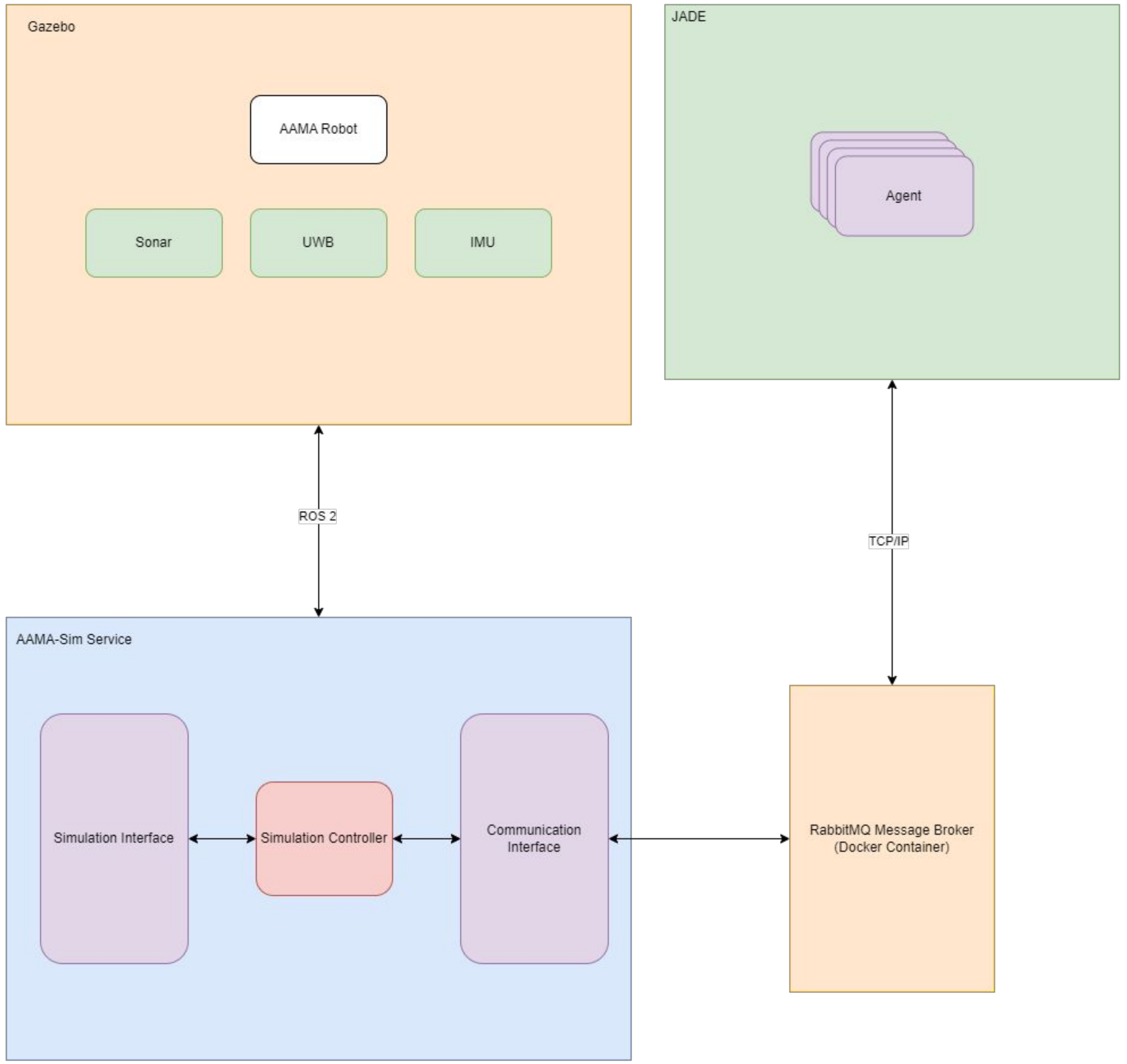
AAMA-Sim allows developers to customize their robots without digging deeper in Turtlebot4 like projects.

Gazebo used as 3D Engine and Ogre as its physics engine.

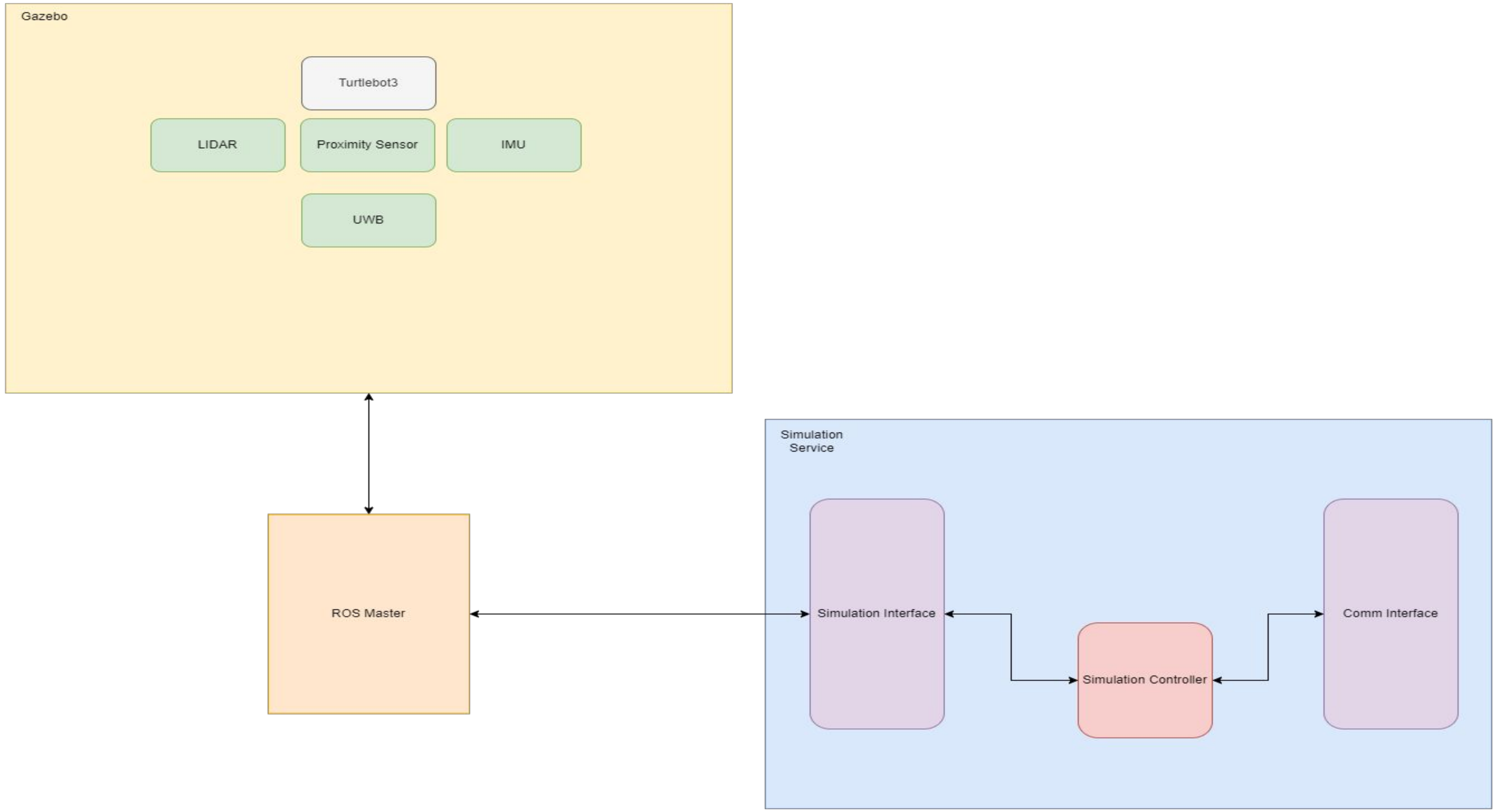
ROS 2 is used as main middleware for robotic applications.

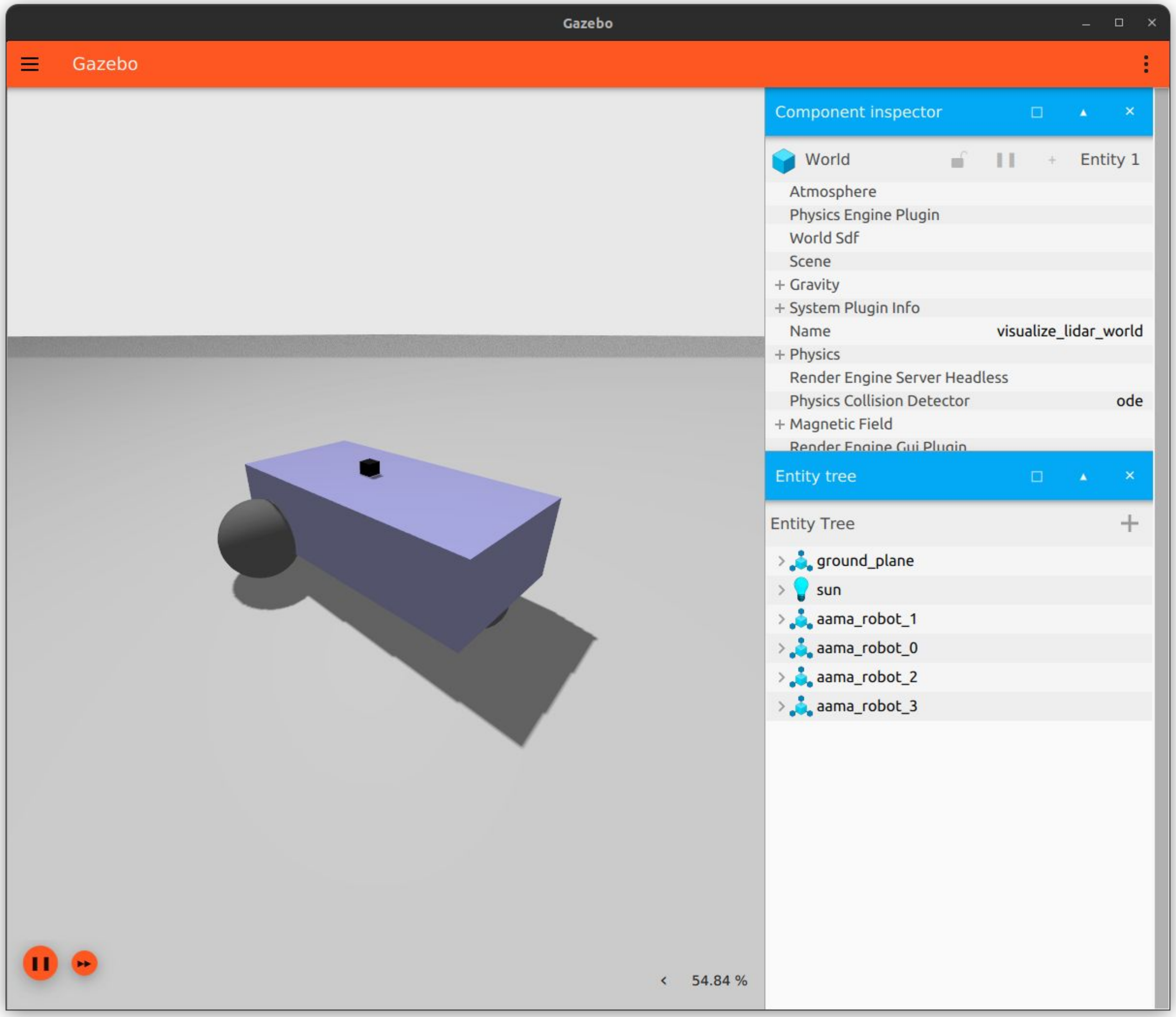
Provides interfacing with different MAS frameworks (such as JADE) through RabbitMQ servers.

AAMA-Sim Design



AAMA-Sim Design Gazebo




Gazebo

Component inspector

World — Entity 1

Atmosphere	
Physics Engine Plugin	
World Sdf	
Scene	
+ Gravity	
+ System Plugin Info	
Name	visualize_lidar_world
+ Physics	
Render Engine Server Headless	
Physics Collision Detector	ode
+ Magnetic Field	
Render Engine Gui Plugin	

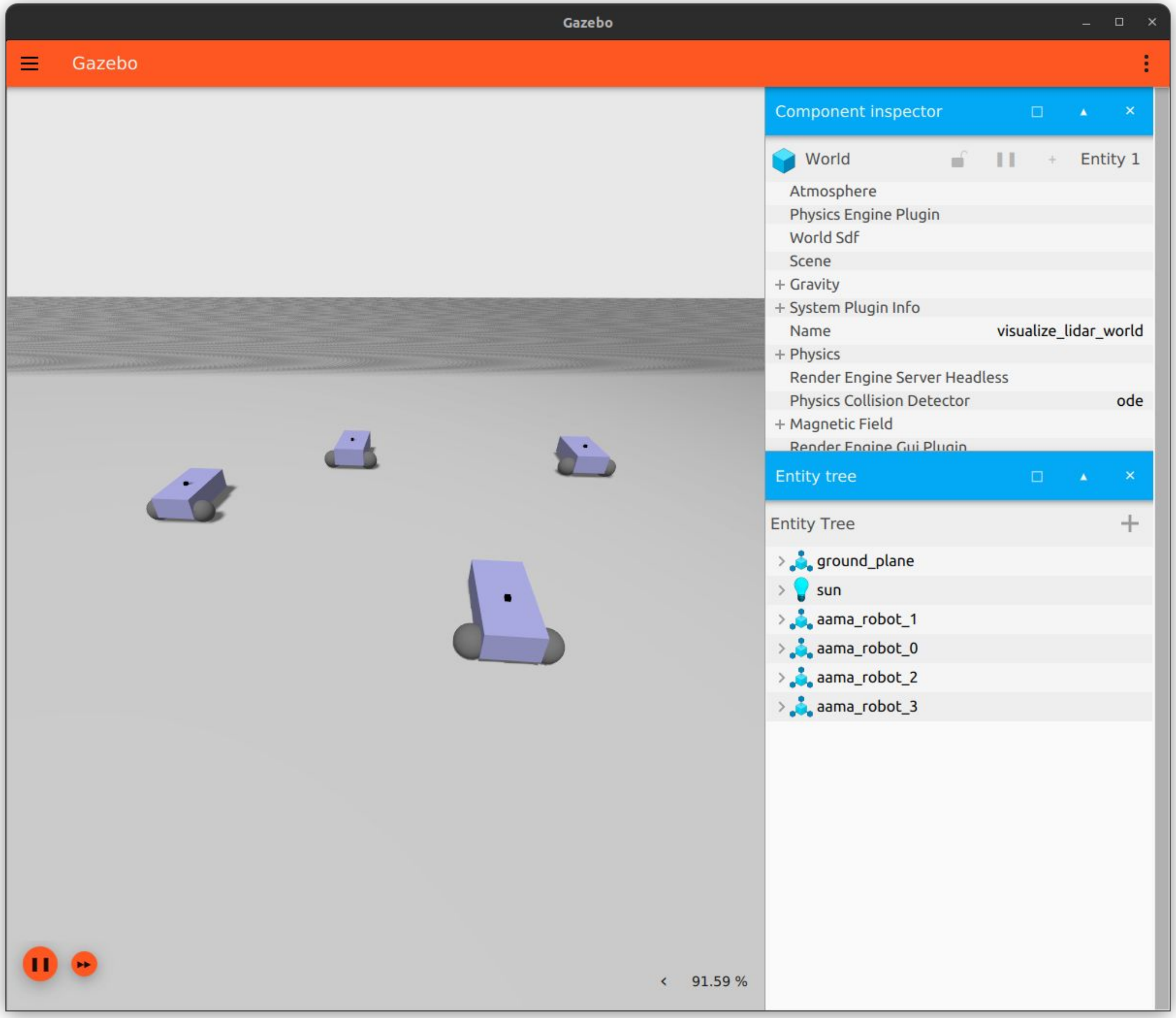
Entity tree

Entity Tree

- ground_plane
- sun
- aama_robot_1
- aama_robot_0
- aama_robot_2
- aama_robot_3

54.84 %

Gazebo: AAMA-Robot



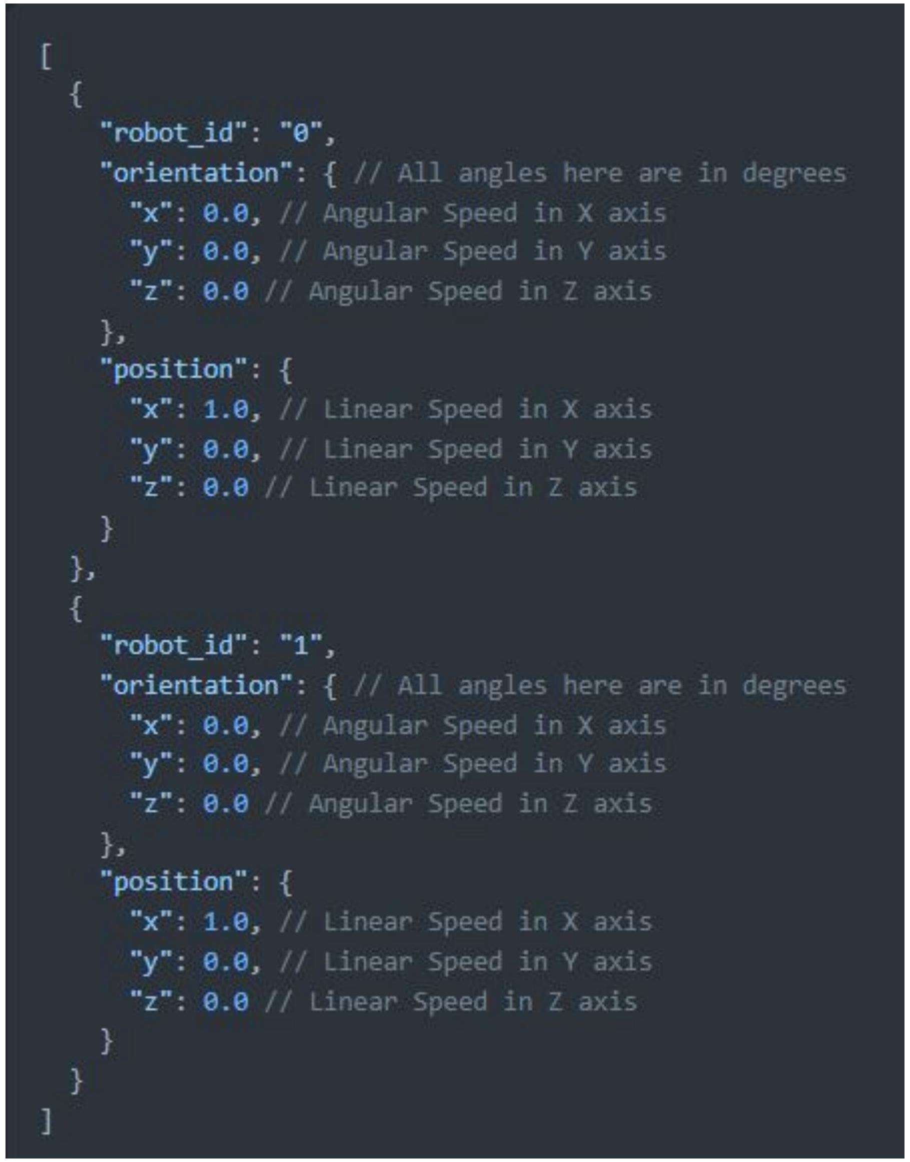
Atmosphere
Physics Engine Plugin
World Sdf
Scene
+ Gravity
+ System Plugin Info
Name visualize_lidar_world
+ Physics
Render Engine Server Headless
Physics Collision Detector ode
+ Magnetic Field
Render Engine Gui Plugin

ground_plane
sun
aama_robot_1
aama_robot_0
aama_robot_2
aama_robot_3

Multi Robot

Message Type: RobotControl
Message Format:

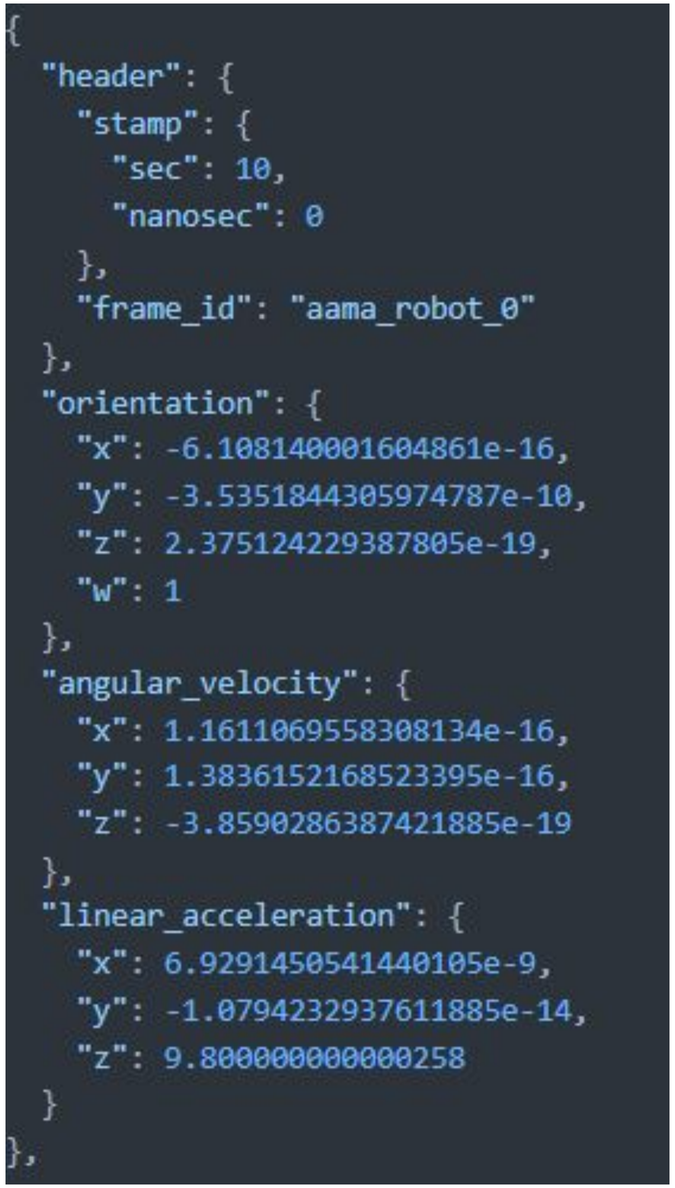
Robot Control Messages: RobotControl



```
[
  {
    "robot_id": "0",
    "orientation": { // All angles here are in degrees
      "x": 0.0, // Angular Speed in X axis
      "y": 0.0, // Angular Speed in Y axis
      "z": 0.0 // Angular Speed in Z axis
    },
    "position": {
      "x": 1.0, // Linear Speed in X axis
      "y": 0.0, // Linear Speed in Y axis
      "z": 0.0 // Linear Speed in Z axis
    }
  },
  {
    "robot_id": "1",
    "orientation": { // All angles here are in degrees
      "x": 0.0, // Angular Speed in X axis
      "y": 0.0, // Angular Speed in Y axis
      "z": 0.0 // Angular Speed in Z axis
    },
    "position": {
      "x": 1.0, // Linear Speed in X axis
      "y": 0.0, // Linear Speed in Y axis
      "z": 0.0 // Linear Speed in Z axis
    }
  }
]
```

Robot Control Messages: IMU

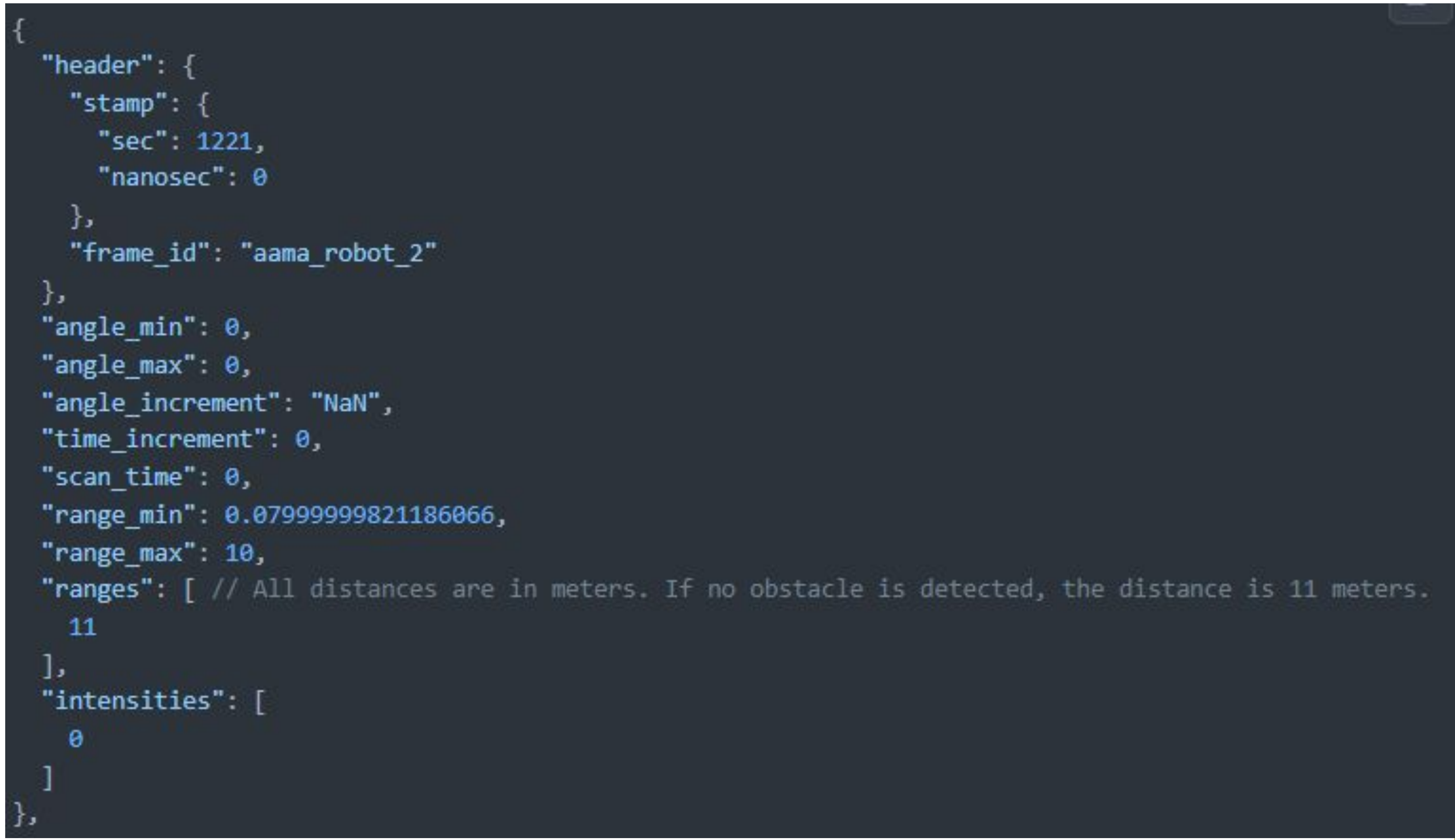
Message Type: IMU
Message Format:



```
{
  "header": {
    "stamp": {
      "sec": 10,
      "nanosec": 0
    },
    "frame_id": "aama_robot_0"
  },
  "orientation": {
    "x": -6.108140001604861e-16,
    "y": -3.5351844305974787e-10,
    "z": 2.375124229387805e-19,
    "w": 1
  },
  "angular_velocity": {
    "x": 1.1611069558308134e-16,
    "y": 1.3836152168523395e-16,
    "z": -3.8590286387421885e-19
  },
  "linear_acceleration": {
    "x": 6.9291450541440105e-9,
    "y": -1.0794232937611885e-14,
    "z": 9.800000000000258
  }
},
```

Robot Control Messages: Sonar

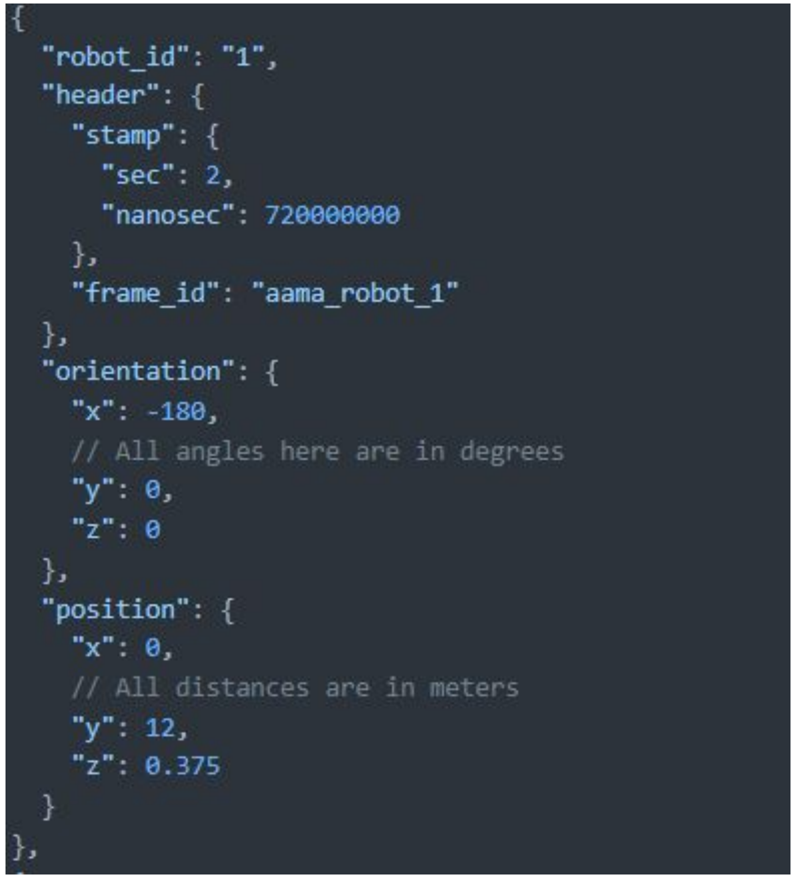
Message Type: Sonar
Message Format:



```
{
  "header": {
    "stamp": {
      "sec": 1221,
      "nanosec": 0
    },
    "frame_id": "aama_robot_2"
  },
  "angle_min": 0,
  "angle_max": 0,
  "angle_increment": "NaN",
  "time_increment": 0,
  "scan_time": 0,
  "range_min": 0.07999999821186066,
  "range_max": 10,
  "ranges": [ // All distances are in meters. If no obstacle is detected, the distance is 11 meters.
    11
  ],
  "intensities": [
    0
  ]
},
```

Robot Control Messages: UWB

Message Type: UWB
Message Format:



```
{
  "robot_id": "1",
  "header": {
    "stamp": {
      "sec": 2,
      "nanosec": 720000000
    },
    "frame_id": "aama_robot_1"
  },
  "orientation": {
    "x": -180,
    // All angles here are in degrees
    "y": 0,
    "z": 0
  },
  "position": {
    "x": 0,
    // All distances are in meters
    "y": 12,
    "z": 0.375
  }
},
```

Future of AAMA-Sim

Initially AAMA-Sim designed for enabling developers to interface JADE with ROS 2 ecosystem. However, in future with the development of AAMA-ROS this simulation environment will support full native ROS 2 applications for MASs.

Some QoL improvements:

- Better config file to defining simulation parameters easier.
- Increased support for more sensors.
- Support for more robots. Like Kobuki, etc.
- Better documentation with tutorials and how to contribute to the source code.

Some non-functional improvements:

- Lower latency with RabbitMQ interface to make the control systems easier.
- A well defined testing suite
- Github CI definitions to control and support open source contribution.

Useful Links

- https://docs.ros.org/en/humble/index.html
- https://www.ros.org/
- https://gazebosim.org/home
- AAMA-Sim: https://github.com/micss-lab/AAMA-sim
- AAMA-Sim Example Agents: https://github.com/micss-lab/AAMA-example-agents

Q&A

Thanks!

MICSS Lab