Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

Institute for AI and Autonomous Systems

Simulation-Based Robot Software Development

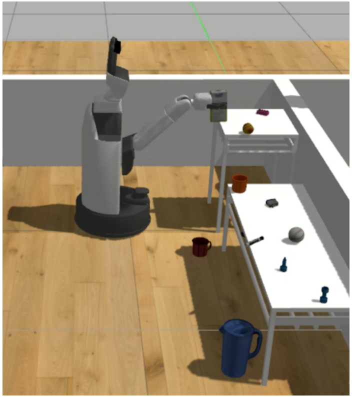
Dr. Alex Mitrevski
Master of Autonomous Systems

Structure

- Motivation for using simulations
- Physics engines
- Unified Robot Description Format (URDF)

Motivation for Using Simulations

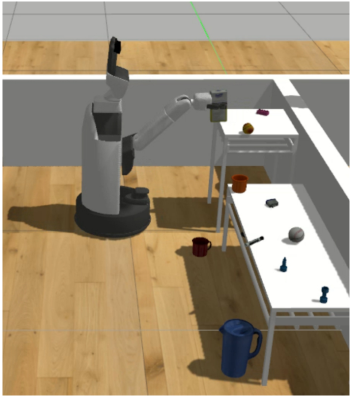
What is a Robot Simulator?



Our HSR robot performing a simulated clean up task

- A simulator in general is a **virtual environment** in which **dynamic physical processes can be modelled**

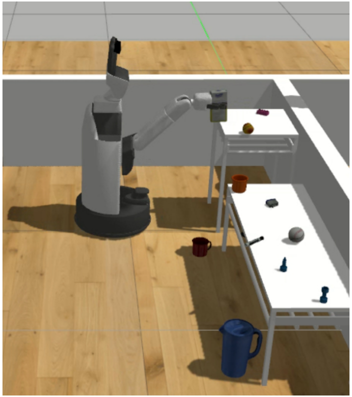
What is a Robot Simulator?



Our HSR robot performing a simulated clean up task

- A simulator in general is a **virtual environment** in which **dynamic physical processes can be modelled**
- A robot simulator is one that **simulates robotic systems and robot interaction scenarios**

What is a Robot Simulator?



Our HSR robot performing a simulated clean up task

- A simulator in general is a **virtual environment** in which **dynamic physical processes can be modelled**
- A robot simulator is one that **simulates robotic systems and robot interaction scenarios**
- Simulators are always associated with a **programming API** that makes it possible to create custom simulated worlds

Simulator Uses in Robotics

- There are two main uses of simulators in robotics:
  - To **create simulation-based testbeds** for verifying the operation of robot prototypes
  - To **collect data** for machine learning

Simulator Uses in Robotics

- There are two main uses of simulators in robotics:
  - To **create simulation-based testbeds** for verifying the operation of robot prototypes
  - To **collect data** for machine learning
- Simulators are also sometimes used for **robotics competitions** so that different algorithmic approaches for solving a given problem can be **compared under standardised conditions**

Simulations as Robot Testbeds

Simulations are primarily a **testbed for robot algorithms** — for a variety of reasons

Simulations as Robot Testbeds

Simulations are primarily a **testbed for robot algorithms** — for a variety of reasons

Rapid prototype testing

The feasibility of developed robot algorithms can be tested **before conducting a real-robot test**

Simulations as Robot Testbeds

Simulations are primarily a **testbed for robot algorithms** — for a variety of reasons

Rapid prototype testing

The feasibility of developed robot algorithms can be tested **before conducting a real-robot test**

Safe testing

Simulations enable performing algorithmic tests **without the danger of damaging a robot**

Simulations as Robot Testbeds

Simulations are primarily a **testbed for robot algorithms** — for a variety of reasons

Rapid prototype testing

The feasibility of developed robot algorithms can be tested **before conducting a real-robot test**

Safe testing

Simulations enable performing algorithmic tests **without the danger of damaging a robot**

Diverse testing

Tests in simulations can be set up so that **a robot is tested in a larger variety of scenarios** than would be possible with physical testing

Simulations as Robot Testbeds

Simulations are primarily a **testbed for robot algorithms** — for a variety of reasons

Rapid prototype testing

The feasibility of developed robot algorithms can be tested **before conducting a real-robot test**

Safe testing

Simulations enable performing algorithmic tests **without the danger of damaging a robot**

Diverse testing

Tests in simulations can be set up so that **a robot is tested in a larger variety of scenarios** than would be possible with physical testing

Continuous testing

Simulations can be **integrated into continuous integration workflows** so that program changes are automatically verified

Simulations for Robot Learning

Simulations are also **frequently used during robot learning**, also for multiple reasons

Simulations for Robot Learning

Simulations are also **frequently used during robot learning**, also for multiple reasons

Simple data collection

Simulations are a relatively **cheap source of data** compared to real-world data, which need a careful setup and are time-consuming to collect

Simulations for Robot Learning

Simulations are also **frequently used during robot learning**, also for multiple reasons

Simple data collection

Simulations are a relatively **cheap source of data** compared to real-world data, which need a careful setup and are time-consuming to collect

Diverse data collection

Due to the simplicity of making changes to simulated worlds, **datasets exposing a robot to a variety of scenarios can be collected**

Simulations for Robot Learning

Simulations are also **frequently used during robot learning**, also for multiple reasons

Simple data collection

Simulations are a relatively **cheap source of data** compared to real-world data, which need a careful setup and are time-consuming to collect

Diverse data collection

Due to the simplicity of making changes to simulated worlds, **datasets exposing a robot to a variety of scenarios can be collected**

Safe learning

Simulations **enable a robot to try out potentially dangerous actions**, which are inevitable for some learning algorithms

Many Uses of Simulations in Robotics

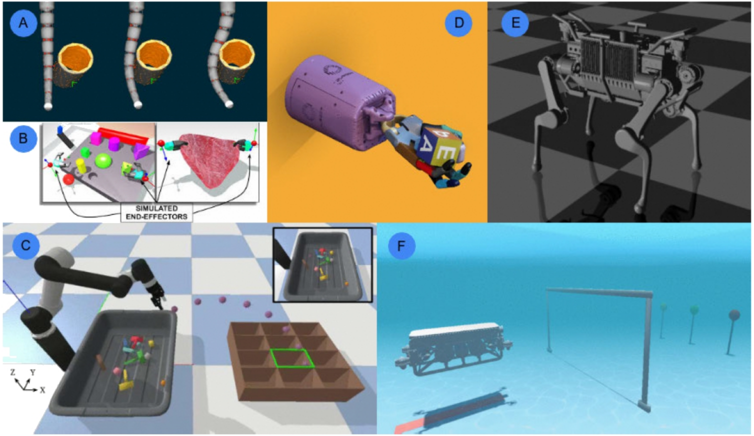
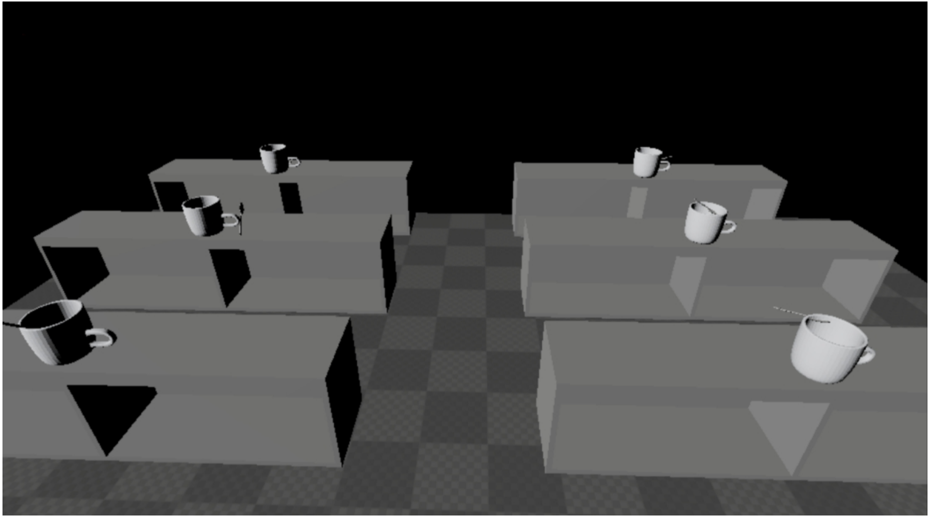


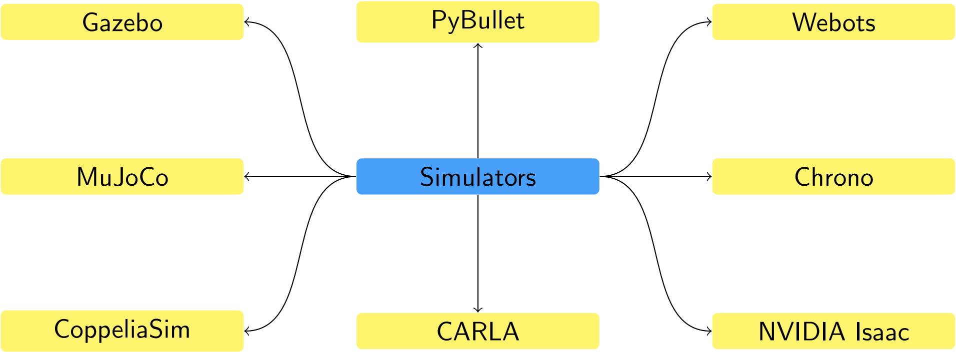
FIGURE 1. **Diversity of simulation scenes and environments throughout robotics (a) soft robotics in Simulation Open Framework Architecture [3], (b) medical robotics in Asynchronous Multi-Body Framework [4], (c) manipulation in PyBullet [5], (d) dexterous manipulation in MuJoCo [6], (e) legged locomotion in RaiSim [7] and (f) underwater vehicles in URSim [8].**

J. Collins et al., "A Review of Physics Simulators for Robotic Applications," in *IEEE Access*, vol. 9, pp. 51416–51431, 2021.

Simulations are Good for Parallel Scenario Execution

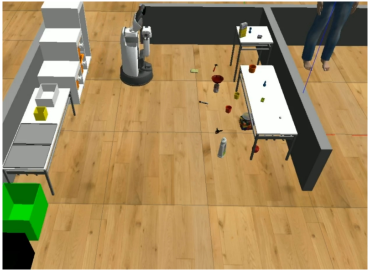


Common Robot Simulators



Robot Simulators and Game Engines

Environment in the CARLA simulator (based on Unreal Engine): https://carla.readthedocs.io/en/latest/map_town03/

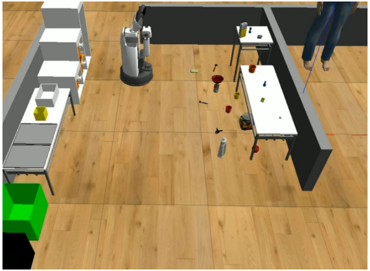


Environment in Gazebo

- Robot simulators are **typically developed independently of game engines** and are **specifically tailored to fit the needs of robotics developers**
  - For instance, Gazebo has many robot models as well as plugins for a variety of commonly used robot sensors
  - Most robotics simulators also support standard robotics development frameworks, such as ROS

Robot Simulators and Game Engines

Environment in the CARLA simulator (based on Unreal Engine): https://carla.readthedocs.io/en/latest/map_town03/



Environment in Gazebo

- Robot simulators are **typically developed independently of game engines** and are **specifically tailored to fit the needs of robotics developers**
  - For instance, Gazebo has many robot models as well as plugins for a variety of commonly used robot sensors
  - Most robotics simulators also support standard robotics development frameworks, such as ROS
- **Some robot simulators are, however, based on game engines**, such as Unity or Unreal Engine
  - Game engines can usually render more photorealistic worlds, which is particularly important if a robot needs to collect visual data in simulation

Physics Engines

What is a Physics Engine?

- In the background, **simulators apply a model of physical laws** — this model is provided by a physics engine

What is a Physics Engine?

- In the background, **simulators apply a model of physical laws** — this model is provided by a physics engine
- Most physical phenomena are governed by (ordinary or partial) differential equations — **physics engines thus need to perform numerical integration**
  - The algorithm that implements a solution to a numerical problem is referred to as a **solver**

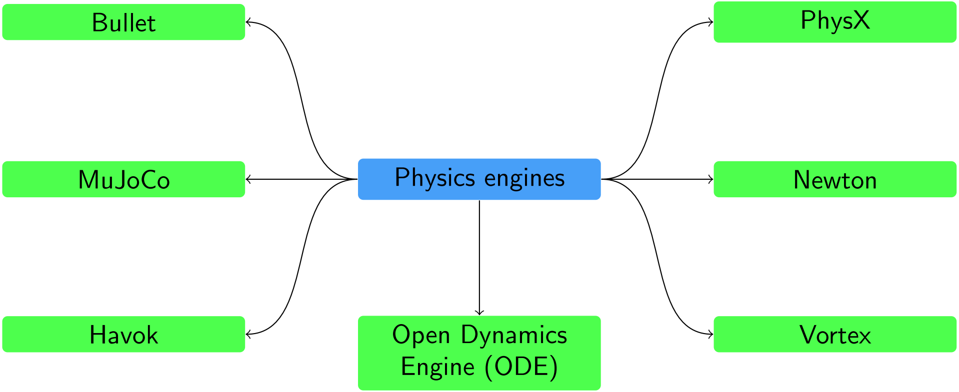
What is a Physics Engine?

- In the background, **simulators apply a model of physical laws** — this model is provided by a physics engine
- Most physical phenomena are governed by (ordinary or partial) differential equations — **physics engines thus need to perform numerical integration**
  - The algorithm that implements a solution to a numerical problem is referred to as a **solver**
- Physics engines **are faced with a trade-off between accuracy and speed** — accurate estimation of physical phenomena requires more solver iterations, but this increases the time to reach a result
  - Engines primarily differ in the solvers they implement and the way they set up the problem

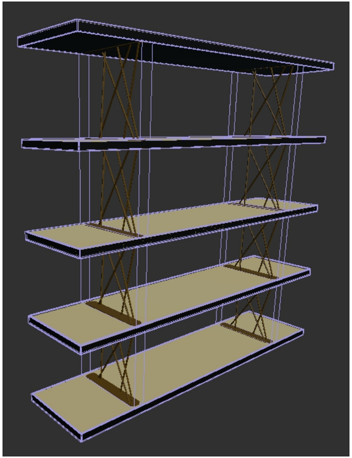
What is a Physics Engine?

- In the background, **simulators apply a model of physical laws** — this model is provided by a physics engine
- Most physical phenomena are governed by (ordinary or partial) differential equations — **physics engines thus need to perform numerical integration**
  - The algorithm that implements a solution to a numerical problem is referred to as a **solver**
- Physics engines **are faced with a trade-off between accuracy and speed** — accurate estimation of physical phenomena requires more solver iterations, but this increases the time to reach a result
  - Engines primarily differ in the solvers they implement and the way they set up the problem
- Most simulators **support multiple physics engines**
  - The engine can be exchanged so that higher accuracy or higher efficiency is achieved, depending on the use case

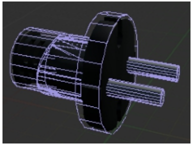
A Multitude of Physics Engines



Body Meshes



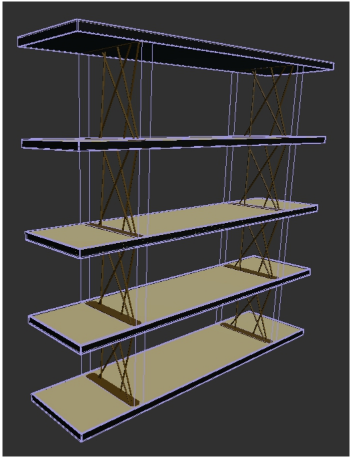
The collision model of the complex side grid is a simple box around it



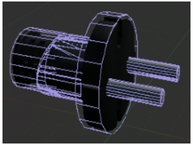
This power plug has a detailed collision model

- Simulators represent objects by **polygon meshes**

Body Meshes



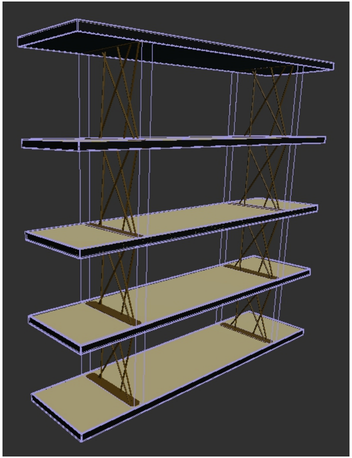
The collision model of the complex side grid is a simple box around it



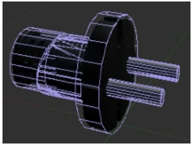
This power plug has a detailed collision model

- Simulators represent objects by **polygon meshes**
- Objects are actually represented by two different models:

Body Meshes



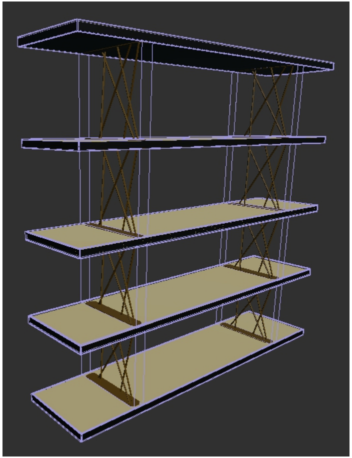
The collision model of the complex side grid is a simple box around it



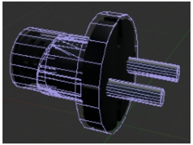
This power plug has a detailed collision model

- Simulators represent objects by **polygon meshes**
- Objects are actually represented by two different models:
  - **Visual model**: A model of the object that is rendered by a simulator, including all object segments, materials, and element colours

Body Meshes



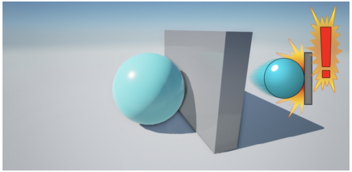
The collision model of the complex side grid is a simple box around it



This power plug has a detailed collision model

- Simulators represent objects by **polygon meshes**
- Objects are actually represented by two different models:
  - **Visual model**: A model of the object that is rendered by a simulator, including all object segments, materials, and element colours
  - **Collision model**: A model used for identifying body collisions and computing collision impacts; this is typically simpler than the visual model so that efficient collision detection procedures can be used

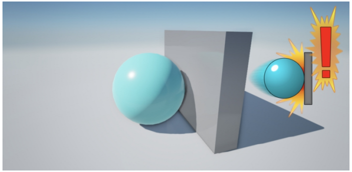
Colliding Bodies



https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Physics/Collision/Overview/

- The core of all physics engines is the **handling of interactions between bodies**

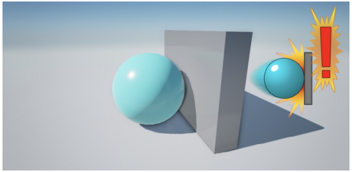
Colliding Bodies



https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Physics/Collision/Overview/

- The core of all physics engines is the **handling of interactions between bodies**
- The interaction handling process is a sequence of two steps:

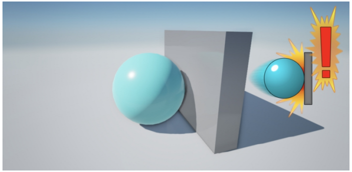
Colliding Bodies



https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Physics/Collision/Overview/

- The core of all physics engines is the **handling of interactions between bodies**
- The interaction handling process is a sequence of two steps:
  - **Collision detection**: This is a continuously running process that checks whether there is an interaction between any parts of a body
  - **Collision effect handling**: When two bodies collide, the impact on the bodies needs to be determined and applied

Colliding Bodies



https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Physics/Collision/Overview/

- The core of all physics engines is the **handling of interactions between bodies**
- The interaction handling process is a sequence of two steps:
  - **Collision detection**: This is a continuously running process that checks whether there is an interaction between any parts of a body
  - **Collision effect handling**: When two bodies collide, the impact on the bodies needs to be determined and applied
- If collisions are not handled or the handling is too slow compared to the motion of the bodies of interest, **the bodies will overlap**

Soft-Body Dynamics

- The dynamic behaviour of deformable objects under external forces is referred to as **soft-body dynamics**; this is in contrast to **rigid-body dynamics**, which, as the name says, is concerned with the dynamic behaviour of rigid bodies

Soft-Body Dynamics

- The dynamic behaviour of deformable objects under external forces is referred to as **soft-body dynamics**; this is in contrast to **rigid-body dynamics**, which, as the name says, is concerned with the dynamic behaviour of rigid bodies
- Soft-body dynamics can be modelled in different ways, for instance using a **spring-mass system** between different points on a body

Soft-Body Dynamics

- The dynamic behaviour of deformable objects under external forces is referred to as **soft-body dynamics**; this is in contrast to **rigid-body dynamics**, which, as the name says, is concerned with the dynamic behaviour of rigid bodies
- Soft-body dynamics can be modelled in different ways, for instance using a **spring-mass system** between different points on a body
- All physics engines are able to simulate rigid-body dynamics; soft-body dynamics is, on the other hand, less commonly supported
  - Soft-body dynamics solvers are more computationally demanding than rigid-body solvers, so some physics engines do not implement them at all and opt for simpler models

Unified Robot Description Format (URDF)

What is URDF?

- The Unified Robot Description Format (URDF) is **an XML-based language for describing robots and simulated worlds**
- URDF robot models can be found for **most commonly used robots**
  - URDF models are typically developed and provided by the robot manufacturers themselves
- **Some simulators have their own description formats**, but most either support URDF directly or provide tools to convert URDF to their custom formats

URDF Elements

- In URDF, complex bodies, such as that of a robot, are defined **through composing elements and connections between them**, namely through **links and joints**:
  - **Links** are used to define body components
  - **Joints** define connections between links

URDF Elements

- In URDF, complex bodies, such as that of a robot, are defined **through composing elements and connections between them**, namely through **links and joints**:
  - **Links** are used to define body components
  - **Joints** define connections between links
- URDF **imposes a tree structure** on bodies:
  - The structure has **a single root**
  - Each joint can only have **one parent link**

URDF Elements

- In URDF, complex bodies, such as that of a robot, are defined **through composing elements and connections between them**, namely through **links and joints**:
  - **Links** are used to define body components
  - **Joints** define connections between links
- URDF **imposes a tree structure** on bodies:
  - The structure has **a single root**
  - Each joint can only have **one parent link**

Based on the YouBot's URDF:
https://github.com/a2s-institute/youbot_description

```xml
<?xml version="1.0"?>
<robot name="youbot">
    <link name="base_footprint"/>
    <link name="base_link">
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://youbot_description/
                meshes/youbot_base/base.dae"/>
        </geometry>
        <material name="youBot/DarkGrey"/>
      </visual>
      <inertial>
        <mass value="22.0"/>
        <origin xyz="0 0 0"/>
        <inertia ixx="5.7" ixy="-0.01" ixz="1.3" iyy="5.7"
                iyz="-0.007" izz="3.7"/>
      </inertial>
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <box size="0.57 0.36 0.1" />
        </geometry>
      </collision>
    </link>

    <joint name="base_footprint_joint" type="fixed">
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <child link="base_link"/>
      <parent link="base_footprint"/>
    </joint>
</robot>
```

Link

Based on the YouBot's URDF:
https://github.com/a2s-institute/youbot_description

```
<link name="wheel_link_fl">
  <inertial>
    <mass value="1.4"/>
    <origin xyz="0 0 0"/>
    <inertia ixx="0.01" ixy="-0.0007" ixz="0.0005" iyy="
          0.02" iyz="-0.000004" izz="0.01"/>
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <sphere radius="0.0475"/>
    </geometry>
    <material name="youBot/Orange"/>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="1.57 0 0"/>
    <geometry>
      <cylinder radius="0.0475" length="0.0475"/>
    </geometry>
  </collision>
</link>
```

- A link defines an element of a body, **positioned at a given origin with respect to its parent** — links are thus nodes in the body's tree structure

Link

Based on the YouBot's URDF:
https://github.com/a2s-institute/youbot_description

```
<link name="wheel_link_fl">
  <inertial>
    <mass value="1.4"/>
    <origin xyz="0 0 0"/>
    <inertia ixx="0.01" ixy="-0.0007" ixz="0.0005" iyy="
          0.02" iyz="-0.000004" izz="0.01"/>
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <sphere radius="0.0475"/>
    </geometry>
    <material name="youBot/Orange"/>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="1.57 0 0"/>
    <geometry>
      <cylinder radius="0.0475" length="0.0475"/>
    </geometry>
  </collision>
</link>
```

- A link defines an element of a body, **positioned at a given origin with respect to its parent** — links are thus nodes in the body's tree structure
- Links have two elements:
  - A **visual element**, which defines how the link looks when rendered
  - A **collision element**, which defines the link's collision model

Link

Based on the YouBot's URDF:
https://github.com/a2s-institute/youbot_description

```
<link name="wheel_link_fl">
  <inertial>
    <mass value="1.4"/>
    <origin xyz="0 0 0"/>
    <inertia ixx="0.01" ixy="-0.0007" ixz="0.0005" iyy="
          0.02" iyz="-0.000004" izz="0.01"/>
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <sphere radius="0.0475"/>
    </geometry>
    <material name="youBot/Orange"/>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="1.57 0 0"/>
    <geometry>
      <cylinder radius="0.0475" length="0.0475"/>
    </geometry>
  </collision>
</link>
```

- A link defines an element of a body, **positioned at a given origin with respect to its parent** — links are thus nodes in the body's tree structure
- Links have two elements:
  - A **visual element**, which defines how the link looks when rendered
  - A **collision element**, which defines the link's collision model
- Both visual and collision elements can be defined using either **simple geometric shapes** or **meshes**

Link

Based on the YouBot's URDF:
https://github.com/a2s-institute/youbot_description

```
<link name="wheel_link_fl">
  <inertial>
    <mass value="1.4"/>
    <origin xyz="0 0 0"/>
    <inertia ixx="0.01" ixy="-0.0007" ixz="0.0005" iyy="
          0.02" iyz="-0.000004" izz="0.01"/>
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <sphere radius="0.0475"/>
    </geometry>
    <material name="youBot/Orange"/>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="1.57 0 0"/>
    <geometry>
      <cylinder radius="0.0475" length="0.0475"/>
    </geometry>
  </collision>
</link>
```

- A link defines an element of a body, **positioned at a given origin with respect to its parent** — links are thus nodes in the body's tree structure
- Links have two elements:
  - A **visual element**, which defines how the link looks when rendered
  - A **collision element**, which defines the link's collision model
- Both visual and collision elements can be defined using either **simple geometric shapes** or **meshes**
- **Physical properties**, such as the link's mass and inertia, can be specified as well

Joints

Based on the YouBot's URDF:
https://github.com/a2s-institute/youbot_description

```
<joint name="wheel_joint_fl" type="continuous">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <parent link="caster_link_fl"/>
  <child link="wheel_link_fl"/>
  <dynamics damping="1.0" friction="1.0"/>
</joint>
```

```
<joint name="base_footprint_joint" type="fixed">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <child link="base_link"/>
  <parent link="base_footprint"/>
</joint>
```

```
<joint name="wheel_joint_fl" type="continuous">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <parent link="caster_link_fl"/>
  <child link="wheel_link_fl"/>
  <limit effort="30" velocity="10"/>
  <dynamics damping="1.0" friction="1.0"/>
  <safety_controller k_velocity="10.0"/>
</joint>
```

- Joints specify connections between links — a joint thus adds an edge to the body's tree structure

Joints

Based on the YouBot's URDF:
https://github.com/a2s-institute/youbot_description

```
<joint name="wheel_joint_fl" type="continuous">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <parent link="caster_link_fl"/>
  <child link="wheel_link_fl"/>
  <dynamics damping="1.0" friction="1.0"/>
</joint>
```

```
<joint name="base_footprint_joint" type="fixed">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <child link="base_link"/>
  <parent link="base_footprint"/>
</joint>
```

```
<joint name="wheel_joint_fl" type="continuous">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <parent link="caster_link_fl"/>
  <child link="wheel_link_fl"/>
  <limit effort="30" velocity="10"/>
  <dynamics damping="1.0" friction="1.0"/>
  <safety_controller k_velocity="10.0"/>
</joint>
```

- Joints specify connections between links — a joint thus adds an edge to the body's tree structure
- URDF supports multiple types of joints:
  - **Fixed**: Joints that cannot move
  - **Continuous**: Rotate freely around a given axis without any limits
  - **Revolute**: Rotate around a given axis, but within specified limits
  - **Prismatic**: Translate along a given axis
  - **Planar**: Translate along two axes
  - **Floating**: Able to move freely in 3D

Joints

Based on the YouBot's URDF:
https://github.com/a2s-institute/youbot_description

```
<joint name="wheel_joint_fl" type="continuous">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <parent link="caster_link_fl"/>
  <child link="wheel_link_fl"/>
  <dynamics damping="1.0" friction="1.0"/>
</joint>
```

```
<joint name="base_footprint_joint" type="fixed">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <child link="base_link"/>
  <parent link="base_footprint"/>
</joint>
```

```
<joint name="wheel_joint_fl" type="continuous">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <parent link="caster_link_fl"/>
  <child link="wheel_link_fl"/>
  <limit effort="30" velocity="10"/>
  <dynamics damping="1.0" friction="1.0"/>
  <safety_controller k_velocity="10.0"/>
</joint>
```

- Joints specify connections between links — a joint thus adds an edge to the body's tree structure
- URDF supports multiple types of joints:
  - **Fixed**: Joints that cannot move
  - **Continuous**: Rotate freely around a given axis without any limits
  - **Revolute**: Rotate around a given axis, but within specified limits
  - **Prismatic**: Translate along a given axis
  - **Planar**: Translate along two axes
  - **Floating**: Able to move freely in 3D
- For some joint types, additional parameters can also be specified, such as **dynamics parameters** or **controller coefficients**

Xacro

- Mostly due to the XML nature, URDF descriptions have several practical limitations:

Xacro

- Mostly due to the XML nature, URDF descriptions have several practical limitations:
  - There may be **significant repetition of blocks** that only differ in a few parameters

Xacro

- Mostly due to the XML nature, URDF descriptions have several practical limitations:
  - There may be **significant repetition of blocks** that only differ in a few parameters
  - **Numerical parameters need to be hard-coded**, which reduces the understandability of a description

Xacro

- Mostly due to the XML nature, URDF descriptions have several practical limitations:
  - There may be **significant repetition of blocks** that only differ in a few parameters
  - **Numerical parameters need to be hard-coded**, which reduces the understandability of a description
  - The descriptions **can get rather large** and thus **hard to read**

Xacro

- Mostly due to the XML nature, URDF descriptions have several practical limitations:
  - There may be **significant repetition of blocks** that only differ in a few parameters
  - **Numerical parameters need to be hard-coded**, which reduces the understandability of a description
  - The descriptions **can get rather large** and thus **hard to read**

- A better approach is to **define reusable URDF macros** that can be **parameterised** and **reused** — this is what xacro enables

Xacro

- Mostly due to the XML nature, URDF descriptions have several practical limitations:
  - There may be **significant repetition of blocks** that only differ in a few parameters
  - **Numerical parameters need to be hard-coded**, which reduces the understandability of a description
  - The descriptions **can get rather large** and thus **hard to read**
- A better approach is to **define reusable URDF macros** that can be **parameterised** and **reused** — this is what xacro enables
- In practice, URDF robot descriptions are typically written using xacro and not with pure URDF

Xacro

- Mostly due to the XML nature, URDF descriptions have several practical limitations:
  - There may be **significant repetition of blocks** that only differ in a few parameters
  - **Numerical parameters need to be hard-coded**, which reduces the understandability of a description
  - The descriptions **can get rather large** and thus **hard to read**
- A better approach is to **define reusable URDF macros** that can be **parameterised** and **reused** — this is what xacro enables
- In practice, URDF robot descriptions are typically written using xacro and not with pure URDF

Adapted from Freddy's URDF: https://github.com/a2s-institute/freddy_description

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
    <xacro:property name="wheel_radius" value="0.05" />
    <xacro:property name="wheel_mass" value="1.0" />
    <xacro:property name="inertia_wheels" value="0.001" />
    <xacro:include filename="$(find freddy_base_description)/urdf
          /common.xacro" />

    <xacro:macro name="kelo_wheel" params="name parent *origin
          material_name">
        <joint name="${name}_joint" type="fixed">
            <xacro:insert_block name="origin" />
            <parent link="${parent}"/>
            <child link="${name}_link"/>
        </joint>
        <link name="${name}_link">
            <visual>
                <geometry>
                    <sphere radius="${wheel_radius}" />
                </geometry>
                <material name="${material_name}"/>
            </visual>
            <collision>
                <geometry>
                    <sphere radius="${wheel_radius}" />
                </geometry>
            </collision>
            <xacro:sphere_inertia mass="${wheel_mass}" radius="${
                  wheel_radius}" />
        </link>
    </xacro:macro>
</robot>
```

Summary

- Robot development is often supported by the use of simulators, which can be used both for algorithmic testing as well as in the context of machine learning
- All simulators use physics engines, which model physical phenomena and provide solvers for approximating those
- URDF and its improvement xacro are commonly used for describing robots and specifying various parameters associated with robot models