Handling with AI-enhanced Robotic Technologies for flexible ManUfacturing

D2.3
Simulation infrastructure for handling component training

Deliverable ID:	D2.3
Project Acronym:	HARTU
Grant:	101092100
Call:	HORIZON-CL4-2022-TWIN-TRANSITION-01
Project Coordinator:	TEKNIKER
Work Package:	WP2
Deliverable Type:	OTHER
Responsible Partner:	TEK
Contributors:	TEK, DFKI
Edition date:	22 December 2023
Version:	03
Status:	Final
Classification:	PU

This project has received funding from the European Union's Horizon Europe - Research and Innovation program under the grant agreement No 101092100. This report reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

HARTU Consortium

HARTU "Handling with AI-enhanced Robotic Technologies for flexible manufactUring" (Contract No. 101092100) is a collaborative project within the Horizon Europe – Research and Innovation program (HORIZON-CL4-2022-TWIN-TRANSITION-01-04). The consortium members are:

1	Tekniker	FUNDACION TEKNIKER (TEK) 20600 Gipuzkoa | Spain	Contact: Iñaki Maurtua inaki.maurtua@tekniker.es
2	DFKI German Research Center for Artificial Intelligence	DEUTSCHES FORSCHUNGSZENTRUM FUER KUENSTLICHE INTELLIGENZ GMBH (DFKI) 67663 Kaiserslautern | Germany	Contact: Dennis Mronga dennis.mronga@dfki.de
3	aimen TECHNOLOGY CENTRE	ASOCIACIÓN DE INVESTIGACIÓN METALÚRGICA DEL NOROESTE (AIMEN) 36418 Pontevedra| Spain	Contact: Jawad Masood jawad.masood@aimen.es
4	ENGINEERING	ENGINEERING INGEGNERIA INFORMATICA S.P.A. (ENG) 00144 Rome| Italy	Contact: Riccardo Zanetti riccardo.zanetti@eng.it
5	TOFAŞ TÜRK OTOMOBİL FABRİKASI A.Ş.	TOFAS TURK OTOMOBIL FABRIKASI ANONIM SIRKETI (TOFAS) 34394 Istanbul | Turkey	Contact: Nuri Ertekin nuri.ertekin@tofas.com.tr
6	PHILIPS	PHILIPS CONSUMER LIFESTYLE BV (PCL) 5656 AG Eindhoven | Netherlands	Contact: Erik Koehorst erik.koehorst@philips.com
7	ULMA	ULMA MANUTENCION S. COOP. (ULMA) 20560 Gipuzkoa | Spain	Contact: Leire Zubia lzubia@ulmahandling.com
8	deepblue	DEEP BLUE Srl (DBL) 00193 ROME | Italy	Contact: Erica Vannucci erica.vannucci@dblue.it
9	FMI	FMI HTS DRACHTEN B.V. (FMI) NL-4622 RD Bergen Op Zoom, Netherlands	Contact: Floris goet floris.goet@fmi-improvia.com
10	TECNOALIMENTI	TECNOALIMENTI S.C.p.A (TCA) 20124 Milano | Italy	Contact: Marianna Faraldi m.faraldi@tecnoalimenti.com
11		POLITECNICO DI BARI (POLIBA) 70126 Bari | Italy	Contact: Giuseppe Carbone giuseppe.carbone@poliba.it
12	OMNIGRASP	OMNIGRASP S.r.l. (OMNI) 70124 Bari | Italy	Contact: Vito Cacucciolo vito.cacucciolo@omnigrasp.com
13	ITRI Industrial Technology Research Institute	INDUSTRIAL TECHNOLOGY RESEARCH INSTITUTE INCORPORATED (ITRI)	Contact: Curtis Kuan curtis.kuan@itri.org.tw
14	INFAR INFAR INDUSTRIAL CO., LTD.	INFAR INDUSTRIAL Co., Ltd (INFAR) 504 Chang-hua County | Taiwan	Contact: Simon Chen simon@infar.com.tw

Funded by the European Union

Document history

Date	Version	Status	Author	Description
30/11/2023	01	Draft	TEK	Document template
15/12/2023	02	Draft	TEK	Contributions integrated
22/12/2023	03	Final	TEK	Submitted version

Funded by the European Union

Executive Summary

HARTU has developed a set of simulation tools (SimEnv) to help system integrators configure some features of a handling application. This document describes these tools.

SimEnv includes the SyntheticImageDatasetGenerator component, which assists in the creation of a synthetic image dataset using Unity. The dataset will be used to train an object detection YOLOv5 model whose output will be part of the input to a generic object segmentation model to perform the segmentation of the object in a real scene.

The SimEnv component of the HARTU reference architecture includes also the LocalGraspPointTester component, which assists in the validation of grasping points proposed by LocalGraspModeller, and GlobalGraspPolicyTester component, which assists the GlobalGraspModeller in defining the GlobalGraspModel to decide which object in a cluttered scene is the best candidate to be picked up.

These components included in SimEnv use two well-known simulation engines: Unity as a general framework and for the generation of realistic images, and MuJoCo as a physics engine.

In addition, the **Learning from Demonstration** component also uses MuJoCO to record, refine, and pre-evaluate the assembly skills demonstrated by the user. While the recording of skills will mainly be done in real-world scenarios, the refinement of skills via Inverse Reinforcement Learning, can only be done in simulation, as it usually requires many evaluations.

This simulation environment is not intended to simulate the complete sequence of actions in a handling or assembly robotic application, but to assist the system builder in configuring it.

Funded by
the European Union

1 Table of contents

List of figures

Funded by the European Union

List of tables

Funded by the European Union

Acronyms

List of the acronyms	
HARTU	Handling with AI-enhanced Robotic Technologies for flexible manufactUring

Funded by the European Union

1 Introduction

Two key steps in materials handling are the identification of the object to be picked up and the decision on how to pick it up.

HARTU proposes to facilitate the definition of both actions through AI in such a way that human intervention is reduced to a minimum. Two components of the HARTU reference architecture will provide the models for segmenting the objects in an image and for defining the grasping points of that object.

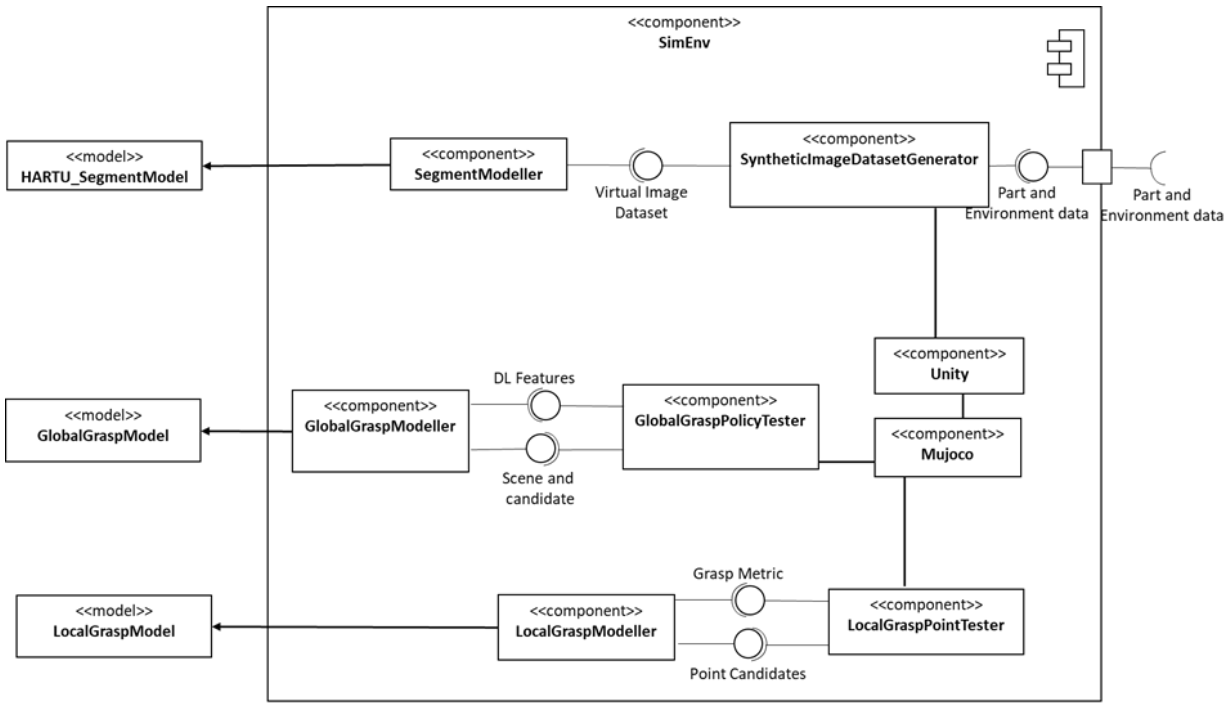

Figure 1. SimEnv component in the HARTU reference architecture

The SegmentModeller uses deep learning techniques to create models for segmenting objects in an image. Deep learning requires having many labelled images of the object, but this is not always an easy task. Image labelling is a time-consuming task. Among the various strategies to cope with this problem, one is the use of synthetic images with auto-generated labels to train a segmentation model. This is the first goal of the **SyntheticImageDatasetGenerator** component included in the HARTU's Simulation Environment (SimEnv).

The LocalGraspModeller component is responsible for defining the feasible grasping points for a product-gripper pair. Its approach starts by identifying geometrically valid grasping points (e.g., those that fit between the fingers of a parallel gripper). However, this is not enough, it is necessary to verify which of them are actually valid, i.e., that when the gripper closes and tries to move the part, it does so without the part moving in the gripper nor falling out. For this, LocalGraspModeller will use the **LocalGraspPointTester** component included in SimEnv that will test the grasping operation and return a metric of the quality of the grasping process.

These two components included in SimEnv (i.e. SyntheticImageDatasetGenerator and LocalGraspPointTester) use two well-known simulation engines: Unity as a general framework and

Funded by
the European Union

for the generation of realistic images, and MuJoCo as a physics engine (see sections **¡Error! No se encuentra el origen de la referencia.** and 1.2 for a brief introduction to these).

But identifying the grasping point of an object is not enough to decide which object should be chosen in a scene containing multiple instances of the object. HARTU will provide a GlobalGraspModel to decide which object to pick in a complex scene. To solve this sequential decision-making problem, the GlobalGraspModeller component uses Deep Reinforcement Learning (DRL).

DRL aims to create an agent that, given a number of grasping point candidates, selects the most suitable one for each scene. The agent validates the quality of its proposal by sending it to the **GlobalGraspPolicyTester** component included in the SimEnv, which will execute the action (attempt to grasp the object from the selected grasping point) and send back the result of the grasping simulation (the reward). The agent then uses the reward received from the simulation to train a neural network, and consequently optimise its behaviour for the next iterations. The long-term goal of the agent is to maximise the accumulated reward in a picking sequence.

These functionalities offered by SimEnv, which will be explained in the next sections, are accessible from the AppManager GUI, as shown in Figure 2.

Figure 2. AppManager GUI draft

In addition, the **Learning from Demonstration** component also uses MuJoCO to record, refine, and pre-evaluate the assembly skills demonstrated by the user. While the recording of skills will mainly be done in real-world scenarios, the refinement of skills via Inverse Reinforcement Learning, can only be done in simulation, as it usually requires many evaluations.

Funded by the European Union

1.1 Unity

Unity[1] is a widely used cross-platform game engine and development tool primarily utilized to create video games for various platforms, including mobile devices, consoles, computers, and the web. It was developed by Unity Technologies and first released in 2005.

Unity offers a user-friendly interface and a comprehensive set of tools that allow developers to create 2D, 3D, augmented reality (AR), and virtual reality (VR) experiences. It supports a wide range of programming languages like C#, JavaScript, and Boo.

Unity's popularity stems from its versatility, ease of use, extensive community support, and the ability to deploy projects across multiple platforms.

Key features of Unity include:

1. **Multiplatform support:** Unity allows developers to build games and applications for various platforms like iOS, Android, Windows, macOS, Linux, PlayStation, Xbox, and more.
2. **Asset Store:** A vast marketplace where developers can find and purchase various assets, including 3D models, textures, sound effects, plugins, and tools, to enhance their projects.
3. **Visual Editor:** Unity provides a robust visual editor that simplifies the process of designing and developing games and applications. It includes features for scene editing, animation, physics, lighting, and more.
4. **Scripting:** Developers can use C# (Unity's primary programming language) or other supported languages to write code and implement game logic.
5. **Physics and Animation:** Unity includes built-in physics engines and tools for creating and managing animations, making it easier to simulate realistic movements and interactions.
6. **AI and Networking:** It offers functionalities for implementing artificial intelligence (AI) behaviours and network functionalities for multiplayer games.

1.2 MuJoCo[2]

MuJoCo stands for "Multi-Joint dynamics with Contact." It is a physics engine designed primarily for simulating and controlling articulated biomechanical and robotic systems. Developed by Emo Todorov at the University of Washington, MuJoCo is widely used in research, robotics, biomechanics, and reinforcement learning.

Key features of MuJoCo include:

1. **Efficient Physics Simulation:** MuJoCo is known for its efficient simulation of rigid body dynamics, contact mechanics, and kinematics. It accurately models the behavior of articulated mechanisms and complex interactions between objects.

[1] https://www.unity.com
[2] https://www.mujoco.org

Funded by the European Union

2. **Flexible Modelling:** It allows users to model systems with multiple interconnected bodies and joints. This flexibility is particularly useful for simulating robotic systems, humanoids, and other articulated structures.
3. **Control and Simulation:** MuJoCo offers tools for controlling and simulating various types of systems. It allows users to apply control strategies, test different algorithms, and evaluate the behavior of dynamic systems in a simulated environment.
4. **Integration with Machine Learning:** MuJoCo is often used in reinforcement learning and machine learning research. Its efficient simulation capabilities make it suitable for training agents in complex environments, enabling researchers to develop and test algorithms for robotics and AI.
5. **Commercial and Academic Use:** MuJoCo offers both commercial and academic licenses. Many academic institutions and research labs utilize MuJoCo for studying biomechanics, control systems, robotics, and machine learning applications.

2 Simulation to support image segmentation

2.1 Objective

The ultimate goal is to create a dataset of self-labelled realistic images of an object from its CAD model and other human input parameters.

2.2 HARTU Solution

SimEnv includes the SyntheticImageDatasetGenerator component, which assists in the creation of a synthetic image dataset using Unity.

Each image will be a different scene, containing a variable number of objects in random positions, with different types of occlusions and different values of external parameters, such as lighting, in order to have a great variability of situations approximating real environments. The simulation provides information on the position of the objects in the scene, the occlusions present and whether or not objects are visible from a top view.

The dataset will be used to train an object detection YOLOv5 model whose output will be part of the input to a generic object segmentation model to perform the segmentation of the object in a real scene.

2.2.1 Integration in Unity

SyntheticImageDatasetGenerator is instantiated as a set of scripts in Unity Version 2020.3.48f1. This instantiation is deployed as an executable file, so no license fee is required for its use.

Funded by the European Union

2.2.2 Included functionalities

2.2.2.1 Virtual part definition

HARTU uses the OBJ CAD format, which includes information on geometry and materials. For those other standard CAD formats (e.g., STL) that do not contain information on the materials and visual appearance of the various components they represent, it will be necessary to convert to OBJ and them assign the materials manually.

Basic Mode

The user introduces a new part by defining a (unique) Name and loading the corresponding CAD file in .obj format.

The CAD file can include several components of different materials and visual properties. The graphical interface allows you to define these characteristics for each component: the material, the virtual appearance and the colour (using the RAL standard).

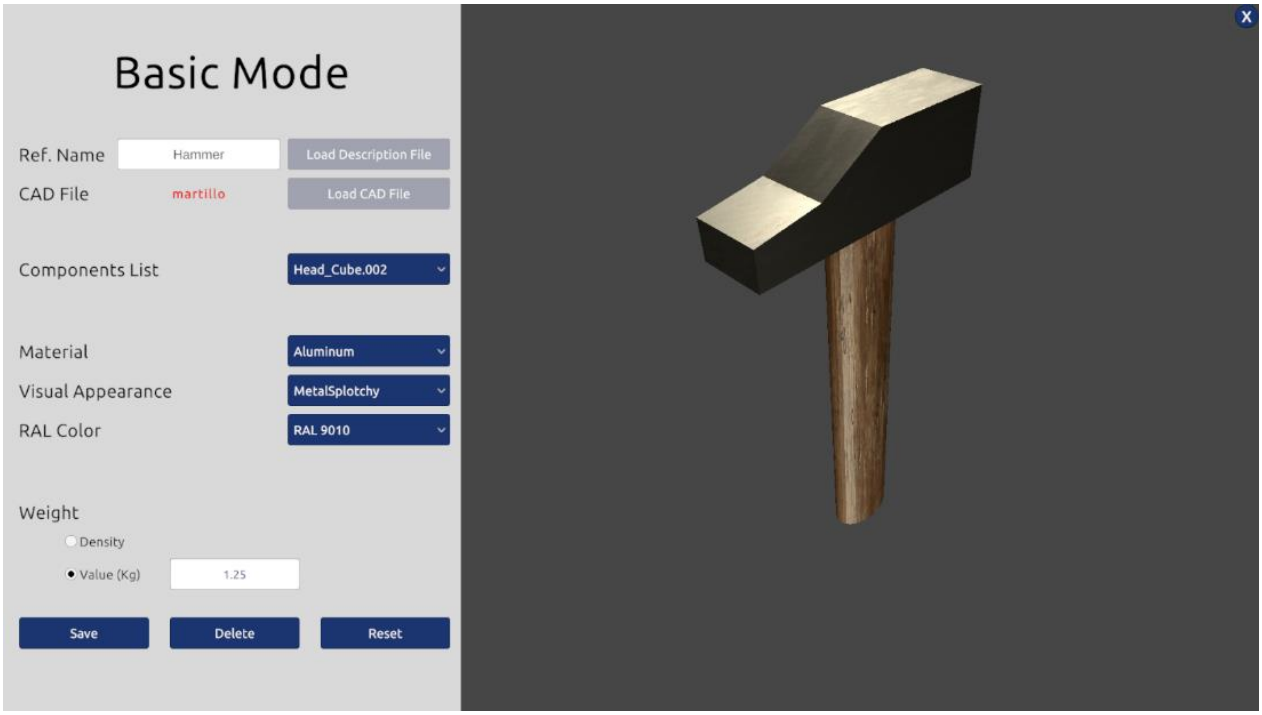

Figure 3. Definition of the materials and properties of a part. Basic Mode

The results of the configuration are stored in a folder with the following naming convention: objectName.hartudescr.

The interface provides basic materials and visual appearance templates. The user can create others directly in Unity. The images below show examples of a product of the same material (e.g., iron) with different visual textures.

Funded by the European Union

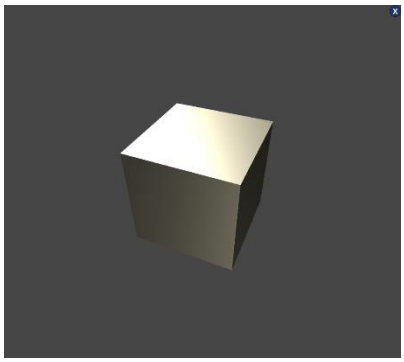

Figure 4. Rusted iron texture applied to a cube

Figure 5. Rusted iron texture applied to a cube

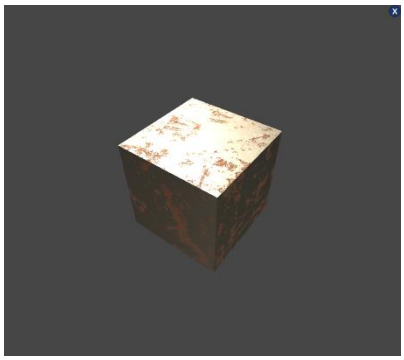

Figure 6. Rusted iron with streaks texture applied to a cube

Figure 7. Beaten up metal texture applied to a cube

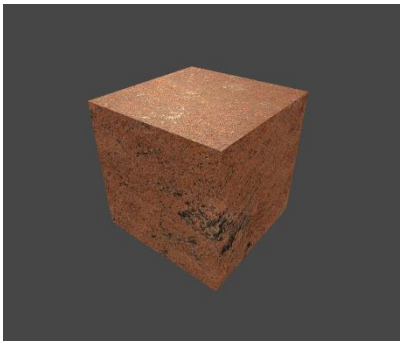

Figure 8. Metal rust coated texture applied to a cube

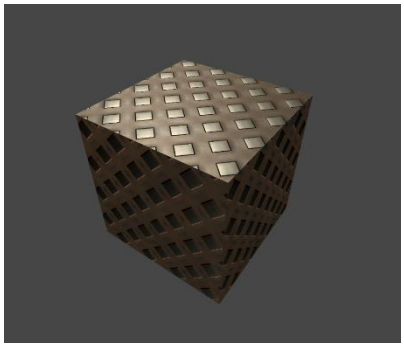

Figure 9. Metal rusting textured texture applied to a cube

Next pictures show examples of a product of the same material (scuffed iron), the same texture and different colours.

Figure 10.Rusted iron texture with RAL 9010

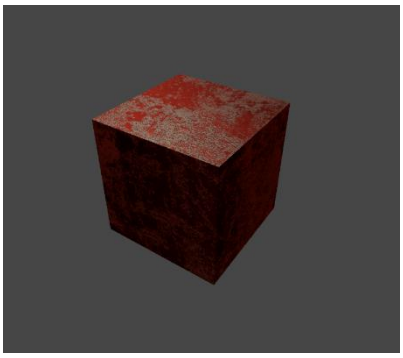

Figure 11. Rusted iron texture with RAL 3011

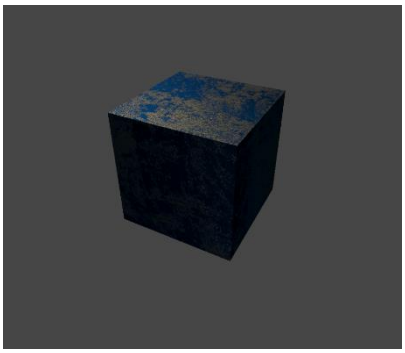

Figure 12. Rusted iron texture with RAL 5017

Advanced Mode

The advanced mode allows the user to configure more parameters that define the appearance of an object (using Unity it is possible to define many others).

As in Basic mode, the user enters a new part by defining a (unique) Name and loading the corresponding CAD file in .obj format.

The CAD file can include several components of different materials and visual properties. The graphical interface allows you to define these characteristics for each component:

Funded by the European Union

- The material
- Colour: using RAL standard and colour texture (optional)
- Normal texture (optional): it represents the behaviour of light on the surface of the object
- Metallic texture (optional): it represents the degree of metallisation of a surface. When using textures, a geometry can be partially defined as metallic while the other parts could not be metallic (e.g. corroded copper's metallic texture will be partially metallic as the copper is, and the corroded part will not behave as metallic surface).
- Ambient occlusion texture (optional): this texture defines to what extent different parts of the object are exposed to ambient lighting. Ambient occlusion textures are very common in materials that have very irregular surfaces and generate shadows on them (e.g. textures which represent corrugated plates).
- The degree of surface roughness: On a scale from 0 to 1, this parameter defines whether the surface of the object is rough or smooth.
- The degree of metallic appearance: On a scale from 0 to 1, this parameter defines the metallicity of an object. This parameter must be defined in case the user does not load a metallic texture. If so, this value will be ignored.

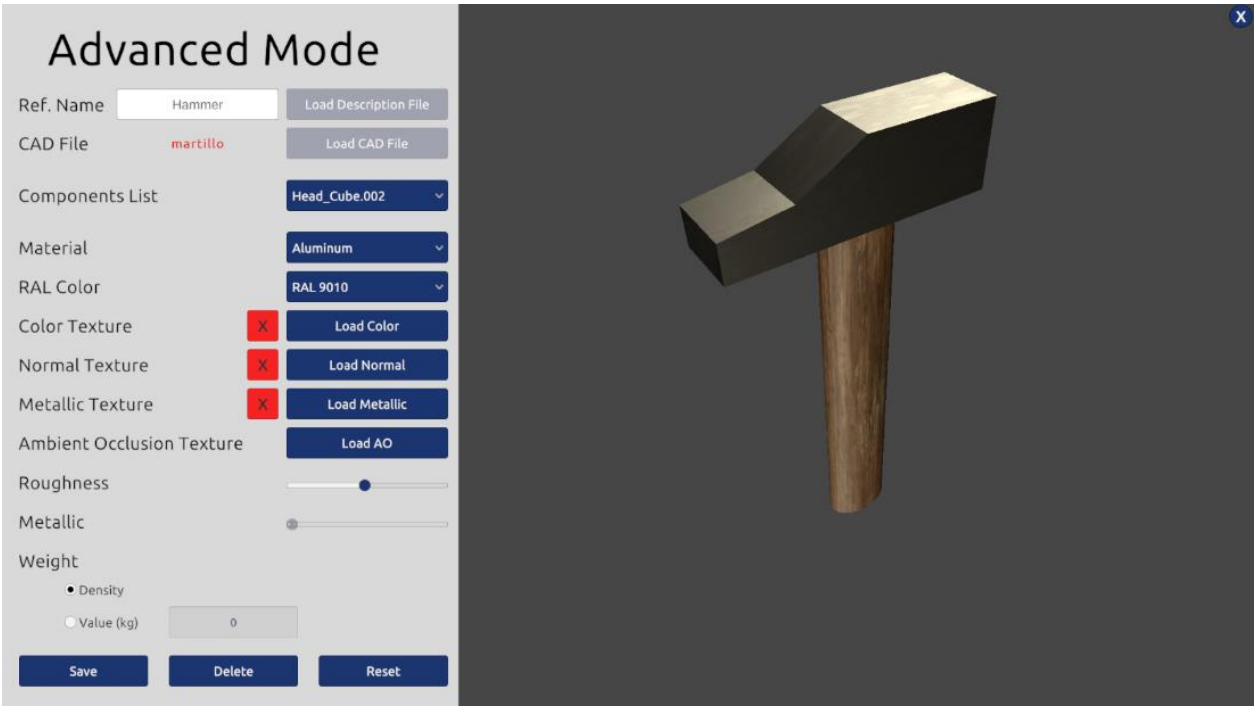

Figure 13. Advanced mode product configuration

The results of the configuration are stored in a folder with the following naming convention: objectName.hartudescr

2.2.2.2 Simulation settings for the generation of random image dataset

Once the visual appearance of the object has been defined, the user can define the parameters that will determine the generation of images for the dataset.

The user selects the part reference for the dataset and the number of images in the dataset. To understand how this number is defined, we will introduce the concept of a 'round' and how the simulation works.

Funded by the European Union

The user defines the minimum number of parts in the scene to start recording the image, and the total number of parts in each round. A round starts with the SyntheticImageDatasetGenerator throwing a part into the scene, the part will collide with the base/wall of the box or the surface (this is user-selected), and its final position is determined by the physics of the collision. A second piece is launched and its final position will depend on how it collides with the previous piece or the surface of the box, this process will continue until the total number of parts in the scene is reached.

SyntheticImageDatasetGenerator will start recording images when the user-defined minimum number of parts in the scene is reached. Then, for each new part in the scene, the systems records:

- An RGB image of the scene with all parts
- A segmented image of all parts
- A set of images, each containing one of the segmented pieces. These images are used as labels of the RGB image.

All the images are stored in PNG format in folders according to this naming convention: PartName_SceneName

Let's suppose that the user selects the CAD of a hammer and the following parameters:

- Number of Rounds: 4
- Minimum number of parts in the scene: 3
- Maximum number of parts in the scene: 5

The sequence in a round can be as follows:

The system will start recording after the 3rd sequence, for which it will generate the following 5 images:

Funded by the European Union

Figure 14.RGB Image

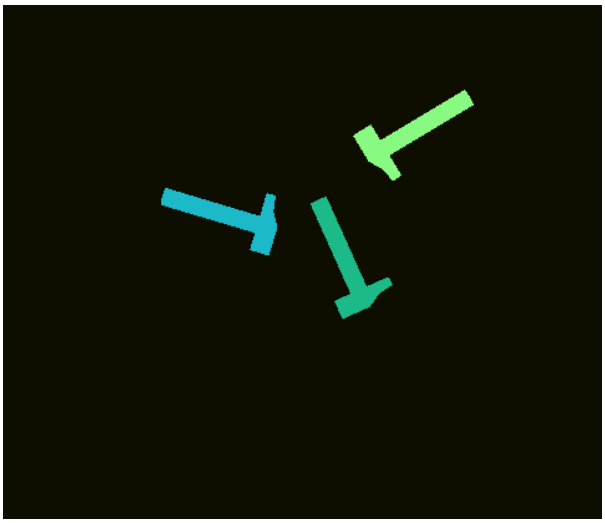

Figure 15. Segmented image with all parts

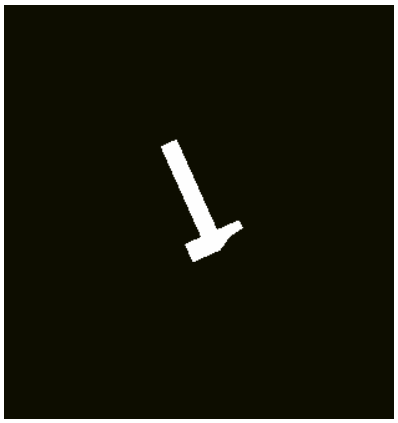

Figure 16. Part #1 segmented

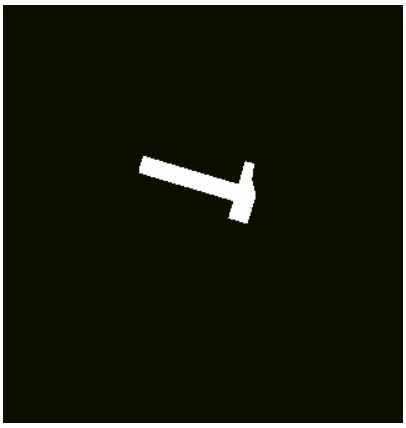

Figure 17. Part #2 segmented

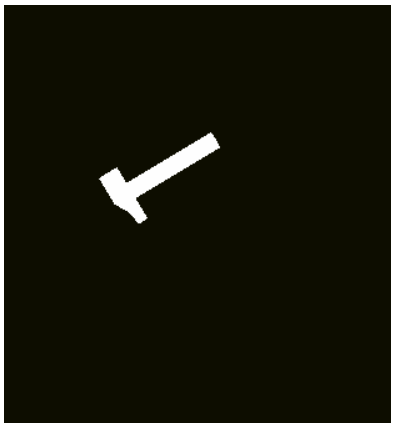

Figure 18. Part #3 segmented

At launch 4 the number of images generated is 6, at launch 5 it is 7, and in general for X number of parts it will generate X+2 images.

The definition of a picking scene is done through this interface:

Funded by the European Union

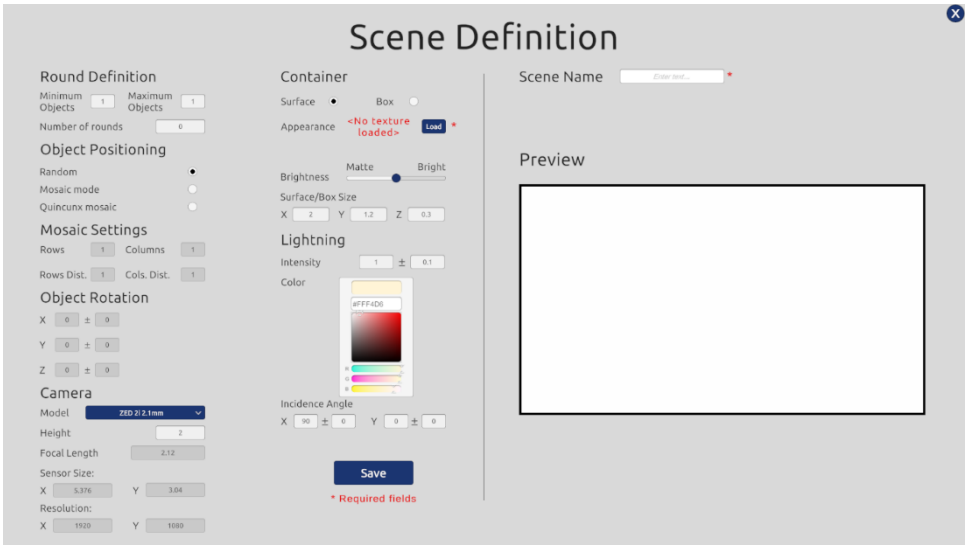

Figure 19.Settings for random generation of image dataset

The scene is stored and can be retrieve using a unique identifier (Name). The following parameters can be set:

Round:

- Number of rounds
- Minimum number of objects to start recording
- Maximum number of objects to end the round

Type of distribution:

- Regular Mosaic
- Quincunx-like mosaic
- Random distribution

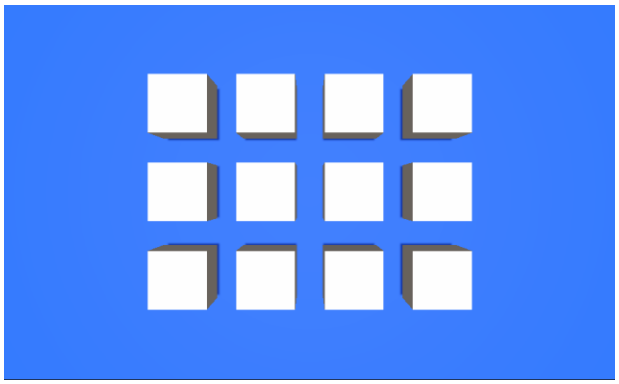

Figure 20.Example of regular mosaic

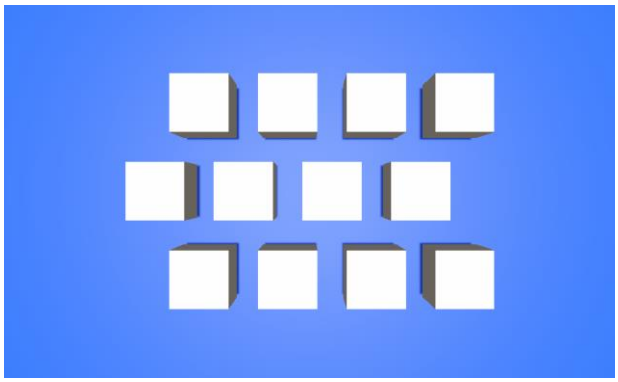

Figure 21.Example of quincunx-like mosaic

In the case of Mosaic distribution, these additional parameters:

- Number of rows
- Number of columns
- Rows distribution
- Columns distribution
- Rotation of the part in the position:

Funded by the European Union

- X, Y, Z
- The user can also define a tolerance range to be used for randomization.

Container:

It has to be defined whether the objects are to be distributed on a flat surface or inside a box. In addition, texture and brightness of the surface or the box have to be defined.

In the case of parts in a box, its size (length, width and height) must be defined.

Main Light

It defines the lighting conditions of the scene:

- Light intensity and range between 0 to 8, where 0 represents darkness (default value is 1) of valid values (for randomization)
- Colour
- Angle of incidence

Camera

It defines the main parameters of the vision system that will be used to simulate the image acquisition.

- The camera model. The system allows selection between Photoneo, ZED2i (4mm and 2.1mm focal length), Intel Realsense D435 or Custom (for manual configuration of the focal length and sensor size).
- The height position of the camera in relation to the bottom of the box/surface
- Focal length
- Sensor size (x,y)

The images generated during the dataset creation are stored in the above-mentioned folder with the following naming convention: yyyymmdd_hhmmss_imageType.png, where:

- The first and second parts of the image name correspond to the date and time the image was taken in numerical format.
- The last part of the name defines which type of image it is. There are 3 types of images:
  - defaultImage: corresponds to a render as the real camera would provide.
  - segmented: The result of the segmentation all the objects in the scene.
  - segmentedObjXXX: For each object in the scene, we export its isolated segmentation. The last part of the name corresponds to the number of the segmented object.

The preview area shows how the defined camera would view the scene with the different parameters the user has entered. As the user modifies the different configuration parameters, this scene preview updates its state showing the resulting scene. The range associated to some parameters allows to configure the randomization for that parameter.

The user can load a previously defined configuration and run it through this interface:

Funded by the European Union

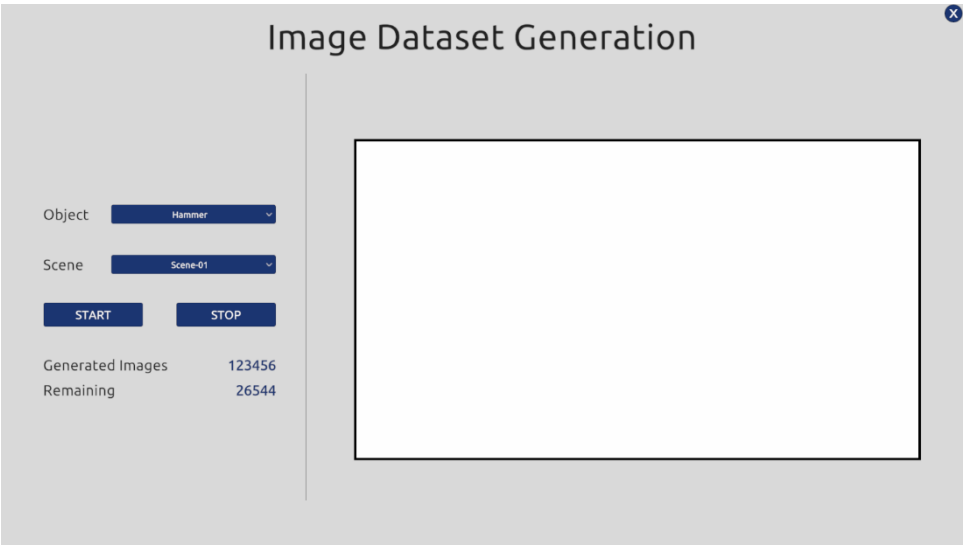

Figure 22. interface to create the image dataset

3 Simulation to support grasp point model creation

3.1 Objective

Twofold objective:

- To provide a simulation mechanism to test the grasping points proposed by the LocalGraspModeller, providing a metric of the quality of the grasp.
- To provide a simulation environment that the GlobalGraspModeller component uses in its Deep Reinforcement Learning approach to create the GlobalGraspModel.

3.2 HARTU Solution

HARTU includes (1) the LocalGraspPointTester component, which assists in the validation of grasping points proposed by LocalGraspModeller, and (2) GlobalGraspPolicyTester component, which assists the GlobalGraspModeller in defining the GlobalGraspModel.

For this purpose, the two components mentioned above use the MuJoCo physics engine, which is integrated in Unity.

The version used is MuJoCo 2.3.2 and the integration is through a plugin.

MuJoCo is primarily a physics engine designed for simulating and controlling articulated biomechanical and robotic systems. It focuses on accurately simulating the dynamics of articulated mechanisms, handling complex interactions between rigid bodies, joints, and contact forces. Unity, on the other hand, is a versatile game and application development platform used for creating a wide range of interactive experiences beyond just physics simulations.

While both MuJoCo and Unity involve simulations and physics, MuJoCo is tailored for high-precision physics modelling of articulated systems, while Unity offers a more comprehensive suite of tools and features for creating diverse interactive content.

Funded by the European Union

This is why. HARTU has chosen MuJoCo to validate in simulation the grasping points proposed by the various components.

Although MuJoCo offers some functionalities in its standalone version that are not available in the Unity plugin, this integration offers greater visual fidelity and easier and more intuitive scene modelling.

In addition, Unity allows communicate with UnityROS2Control, to control the robotic arm with MoveIt. Finally, is should be noted that through this integration of MuJoCo in Unity it is possible to generate objects dynamically, which is very complex in MuJoCo standalone.

However, there are some drawbacks in MuJoCo that have required the development of some functionalities by HARTU, as described in the following sections.

3.2.1 Non-Convex shapes management

MuJoCo only is able to handle convex shape geometries.

In geometry, a convex geometry refers to a shape, set of shapes, or a structure that possesses specific characteristics that define convexity. Convexity relates to the shape's properties with respect to its internal angles, boundaries, and the arrangement of its points.

A convex geometry or shape is defined by the following key characteristics:

1. **Convexity:** A shape is convex if, for any two points within the shape, the line segment joining those points lies completely inside the shape. In other words, any line segment drawn between any two points inside the shape remains entirely contained within the shape itself.
2. **No Interior Angles Greater Than 180 Degrees:** In a convex shape, all interior angles formed by connecting any two points within the shape are less than or equal to 180 degrees. There are no inward-facing angles (concave angles) within the shape.
3. **Boundary and Contour:** The boundary or the perimeter of a convex shape does not intersect itself, and any straight line drawn between any two points on the boundary remains within the shape or on the boundary itself.

Examples of convex shapes include:

- Circles
- Regular polygons (e.g., equilateral triangle, square, pentagon)
- Convex quadrilaterals (e.g., parallelograms, rectangles, rhombi)
- Convex polyhedrals (3D shapes where all faces are convex polygons)
- Spheres and ellipsoids (in higher dimensions)

Convex shapes have numerous applications in various fields, including mathematics, geometry, computer science, optimization, physics, and engineering. Their properties make them easier to analyse mathematically and computationally, so they are often used in algorithms, modelling and problem-solving.

Funded by the European Union

MuJoCo primarily deals with convex geometries in its simulations. It is designed to efficiently handle articulated rigid body dynamics and contact in the context of convex shapes. However, dealing with non-convex geometries or complex deformable objects (such as soft bodies) are not within MuJoCo's core capabilities. Simulating non-convex shapes or deformable objects often requires different techniques, such as finite element methods or specialized algorithms designed explicitly for non-convex geometries.

As many products in industry are non-convex, HARTU has developed a functionality that allows the behaviour of these products to be simulated.

It uses the obj2mjcf command-line interface (CLI), which uses the V-HACD library, to decompose a mesh into a set of simpler convex hulls that approximate the original shape's geometry. The convex hull or convex envelope or convex closure of a shape is the smallest convex set containing the shape.

An example of the application of this function to a non-convex geometry is presented in the following pictures (using a toroid as an example).

This decomposition process is essential in physics simulations and collision detection.

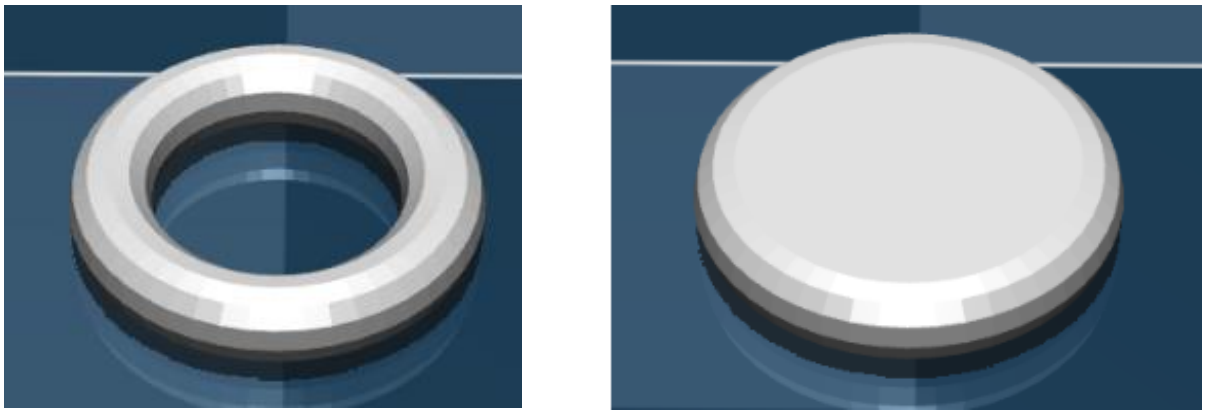

Figure 23. Left: original toroid; right: its convex hull

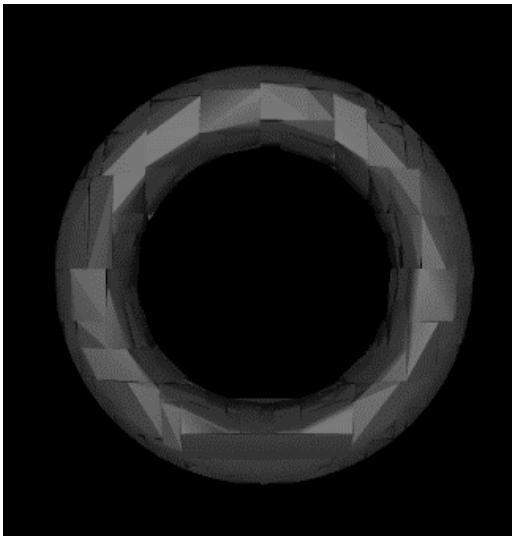

Figure 24. Toroid decomposed in 128 convex sub geometries

Funded by the European Union

In HARTU this process is done via a script that reads the CAD from a folder and generates all the convex submeshes using obj2mjcf and V-HACD. Then, the resulting MJCF generated by obj2mjcf is modified and enhanced to import the XML into the Unity-MuJoCo simulation.

3.2.2 Modelling of grasping tools

The MuJoCo simulation environment contains a set of modelled grippers. HARTU offers the capability to redefine certain key parameters that model the behaviour of these grippers, as seen below.

3.2.2.1 Parallel Jaw Grippers

These grippers have two opposing fingers or jaws that move in parallel.

The parameters that the user can define for modelling the gripper are:

- Geometry of the fingers, selected among a group of predefined finger options.
- Stroke per jaw.
- Closing force.
- The material. This parameter is used to set the proper MuJoCo parameters that will simulate the behaviour of the real material counterpart.

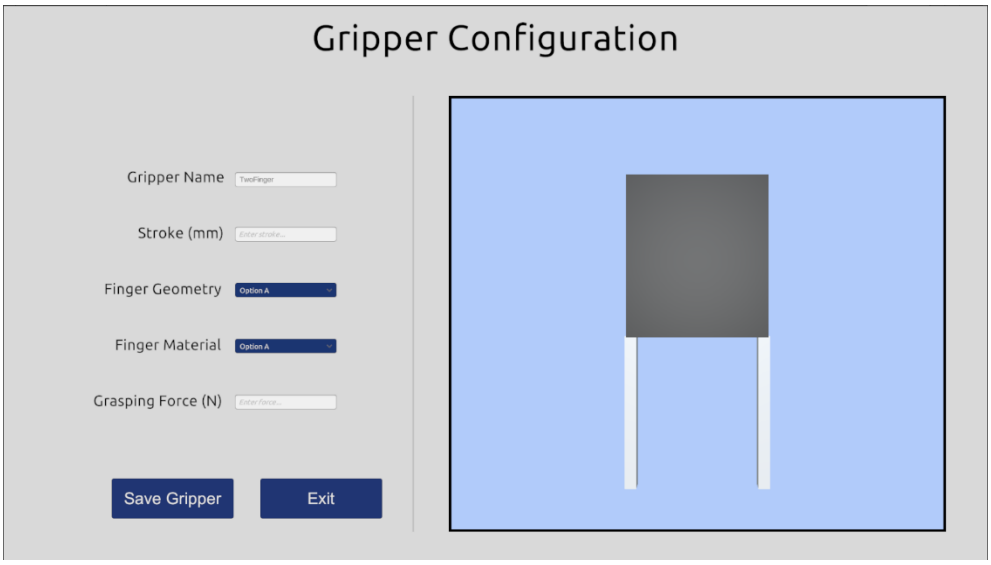

Figure 25. Interface for gripper configuration

To estimate this last parameter, HARTU has developed its own methodology. The experiments are carried out both in simulation and in reality, using the same setup, consisting of an object of the material for which we want to estimate the MuJoCo properties, a Universal Robot UR10 and a ROBOTIQ gripper. For a given set of MuJoCo parameters, an external camera measures the final position of the object after the grasping operation. The final position of the object in the simulated and real environment are compared and the parameters are iteratively adjusted until the difference is below a threshold.

This procedure should be used for each material-gripper pair.

Funded by the European Union

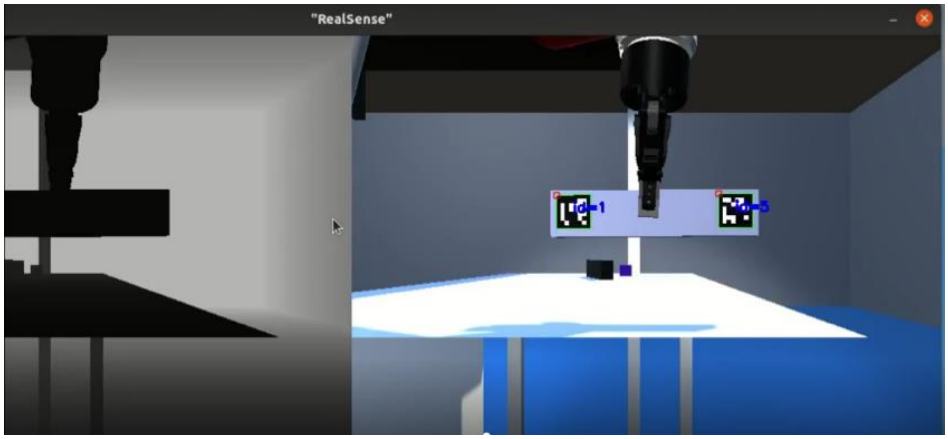

Figure 26. Simulated environment to estimate the parameters.

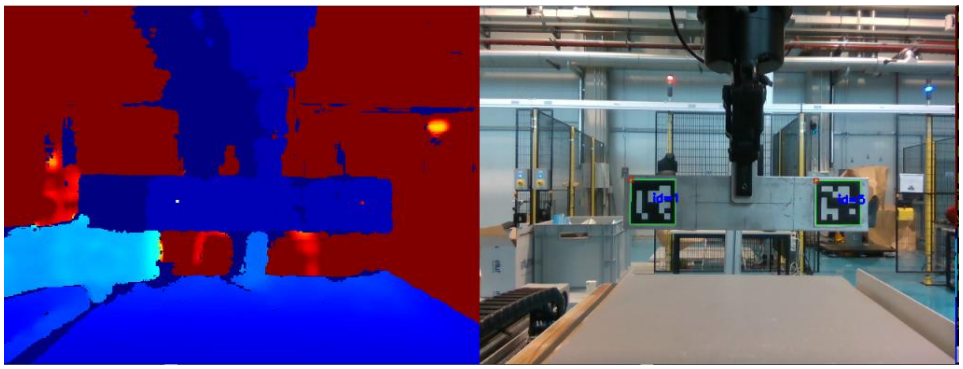

Figure 27. Real environment to estimate the parameters.

3.2.2.2 Three-Fingered Grippers

These grippers have three fingers that can move independently or in a coordinated manner to grasp objects.

The parameters that the user can define for modelling the gripper are:

- Geometry of the fingers, selected among a group of already defined fingers.
- Stroke per jaw.
- Closing force.
- The material. This parameter is used to set the proper MuJoCo parameters that will simulate the behaviour of the real material counterpart.

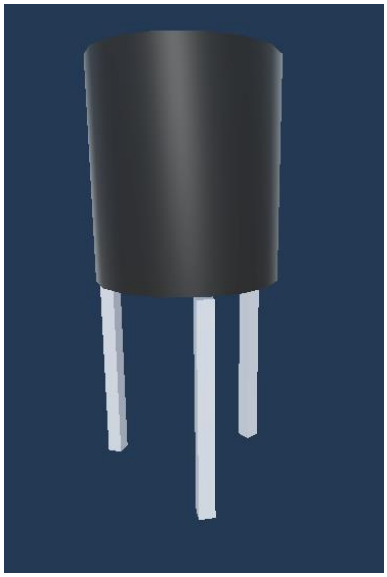

Figure 28. Three-fingered gripper model

3.2.2.3 Suction Cup Grippers

Suction cup grippers use vacuum suction to hold objects.

As mentioned above, the simulation of flexible objects (such as some suction cups) is not straightforward in MuJoCo. So, it has been necessary to develop our own suction cup modelling functionality.

Funded by the European Union

The proposed strategy is to model the suction cup as a set of small spheres linked among them by joints.

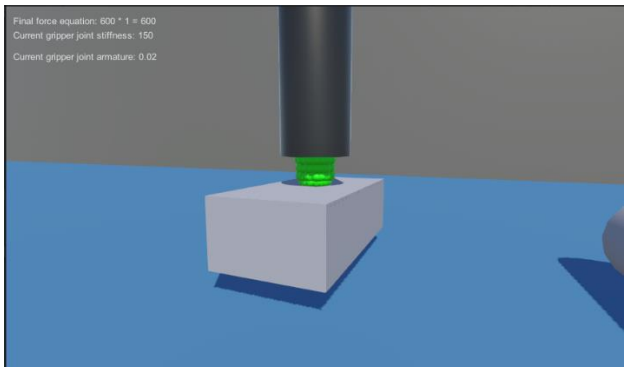

Figure 29. Simulated suction cup in contact with a flat surface

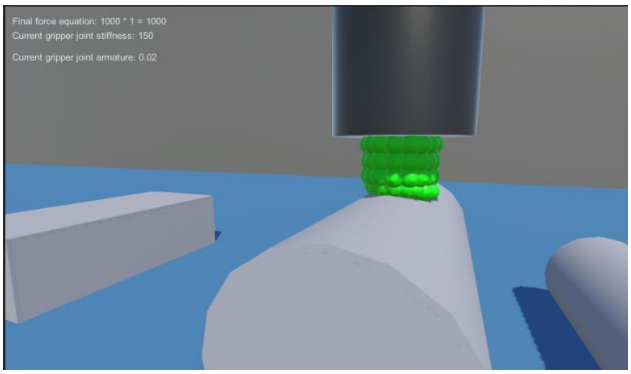

Figure 30. Simulated suction cup in contact with a curved surface

The suction force provided by the suction cup can be estimated using the formula $F=P\times A$, where P is the effective vacuum pressure inside the suction cup and A the contact area of the suction cup adhering to the surface.

In the simulated contact between the suction cup and the object, the number of spheres in contact with the object it is measured. If a sphere is not in contact, the total force is 0; otherwise, it is the theoretical F force.

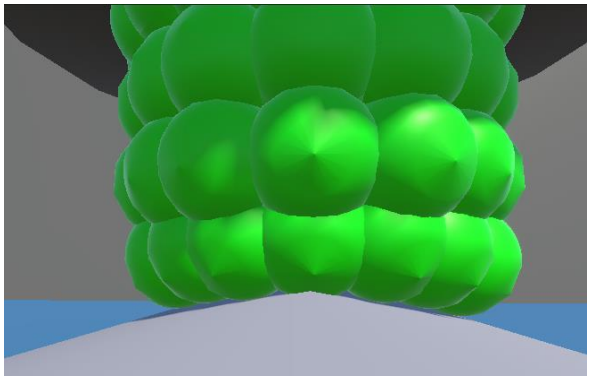

Figure 31. All spheres in contact: total force is F

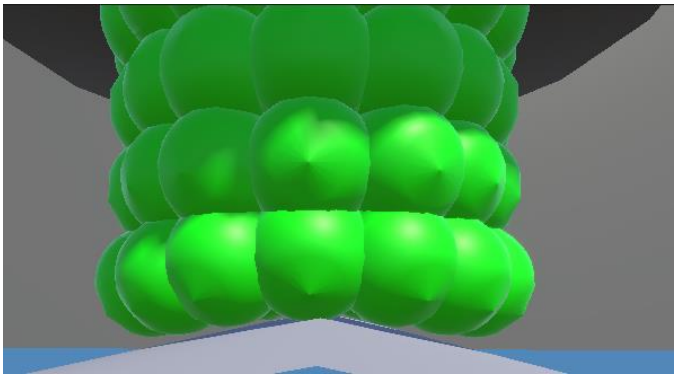

Figure 32. A sphere is not in contact: total force is 0

3.2.2.4 Magnetic Grippers

Magnetic grippers use magnetic fields to hold onto ferromagnetic objects.

The approach to modelling magnetic grippers is similar to that of the suction cup: the proposed strategy is to model the suction cup as a set of small spheres rigidly joined together (without joints in this case).

In the proposed approach, the theoretic force provided by the magnetic gripper is distributed among all the spheres in contact with the surface of the object. In the simulated contact between the magnetic gripper and the object, the number of spheres in contact with the object it is measured. If a sphere is not in contact, it contributes a value of 0 to the total Force, otherwise, it contributes with its theoretical force F/N (where N is the total number of spheres used for the magnetic gripper simulation).

Funded by the European Union

In the following example, the theoretical force of 40 N of the magnetic gripper is divided between the 25 spheres, so that each sphere can contribute 40/25 N, i.e. 1.6 N for each sphere in contact.

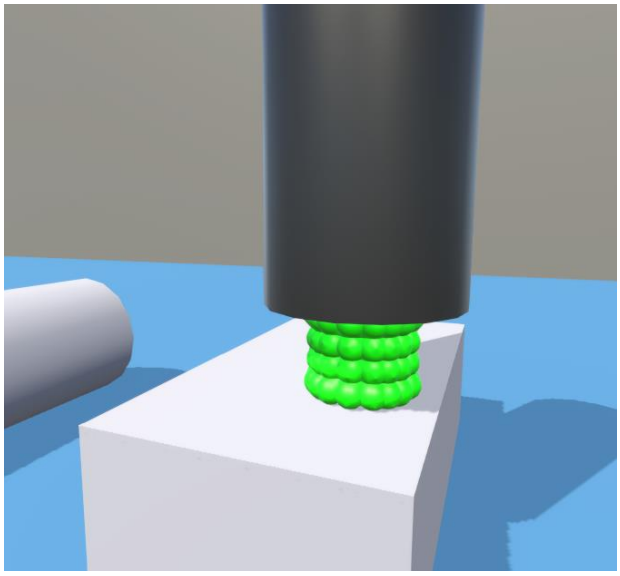

Figure 33. All spheres in contact: total force is 40 N

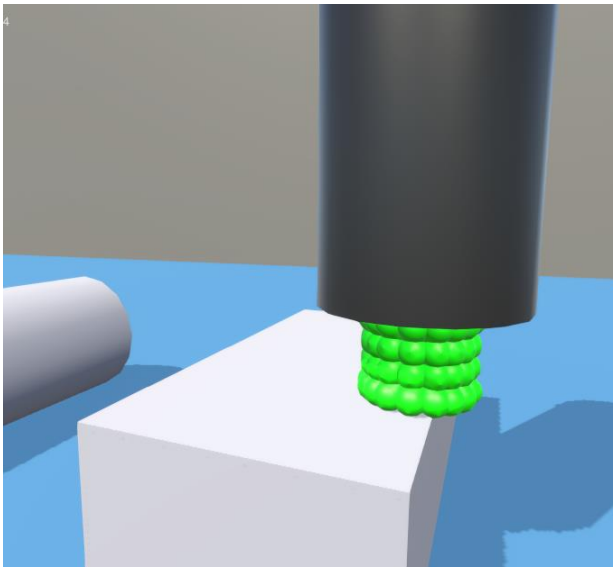

*Figure 34. 6 spheres are not in contact: total force is 30.4N (19*1.6)*

3.2.2.5 Electroadhesive

Initially, they will be assimilated to flexible suction cups.

3.2.3 Included functionalities

3.2.3.1 Isolated object grasping operation: LocalGraspPointTester component

This functionality is used by the **LocalGraspModeller** in its strategy to define the valid grasping points for an object-gripper pair.

The LocalGraspModeller passes a list of the geometrically possible grasping points to the LocalGraspPointTester component, which simulates the grasping operations for all elements in the list and returns for each of them a metric value of the quality of the grasping process.

The testing procedure is as follows: After executing the grasp for each geometrically valid grasping point the gripper holds the object with closed fingers for approximately 4s metric. The initial version of the metric considers both the initial and final pose of the object with respect to the gripper, to calculate an error in that period. A large error indicates that the object has moved and therefore indicates an unstable grasp. The total score is a weighted sum of the translation and rotation error according to the equation:

$$S_{total} = W_1 \cdot S_d + W_2 \cdot S_r$$

Where:

- S_{total} : Weighted *score*
- W_1 : Weight associated to the translation of the object

Funded by the European Union

- W_2 : Weight associated to the rotation of the object
- S_d: Translation *score*. The average score per each axis
- S_r : Rotation score. The average score per each axis

Table 1. Estimation of the score function components

Measurement	Error per axis	Score
Translation in X, Y y Z (S_d)	>1 cm	0
	< Ɛ cm, where Ɛ is near 0 cm	1
	(Ɛ,1]	Linear interpolation between 0 and 1
Rotation in X, Y y Z (S_r)	>90º	0
	<Θ , where Θ is near 0º	1
	(0º, 90º]	Linear interpolation between 0 and 1

Table 2: Estimation of the score function components

In a future version of the metric, the full trajectory of the object during the grasp process will be considered.

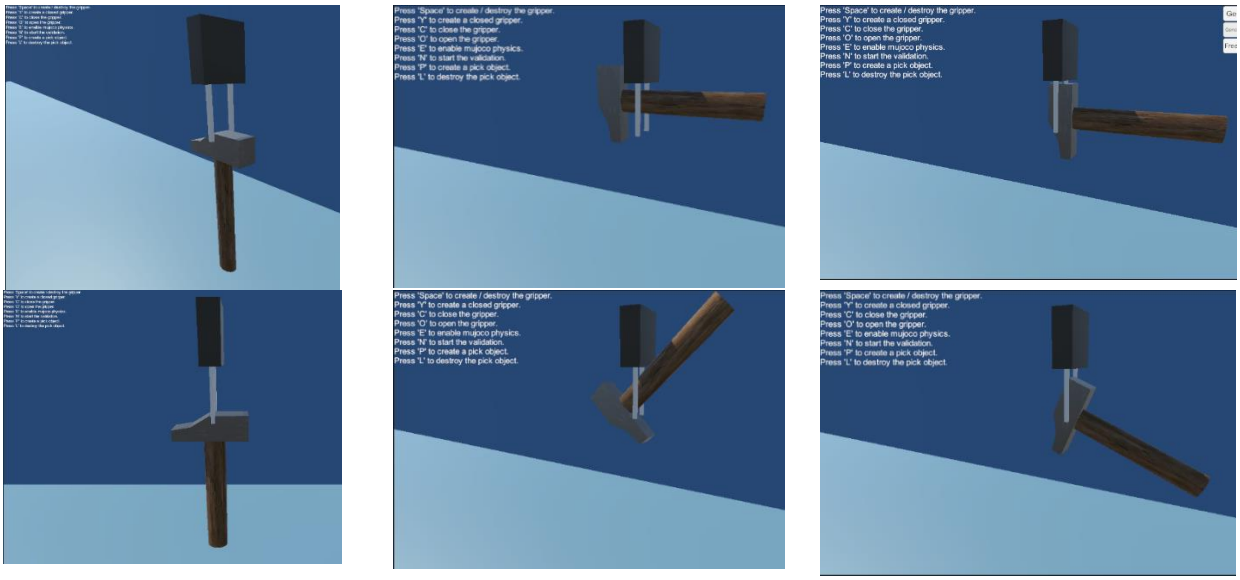

Figure 35. Testing various candidates

Funded by the European Union

3.2.3.2 Object grasping operation in scenes with multiple objects: GlobalGraspPolicyTester component

This functionality is used by the **GlobalGraspModeller** in its strategy to define the GlobalGraspModel.

The GlobalGraspModeller generates a scene with N randomly arranged objects, it segments the image and selects one of the possible grasping points. It then requests the GlobalGraspPolicyTester to execute the grasp operation, which returns the following values:

- The quality metric of the grasp operation
- A vector with the values of the parameters characterising the scene

GlobalGraspPolicyTester uses a simulated UR10 robot, for which a driver has been developed which allows using ROSControl (ROS2) from Unity.

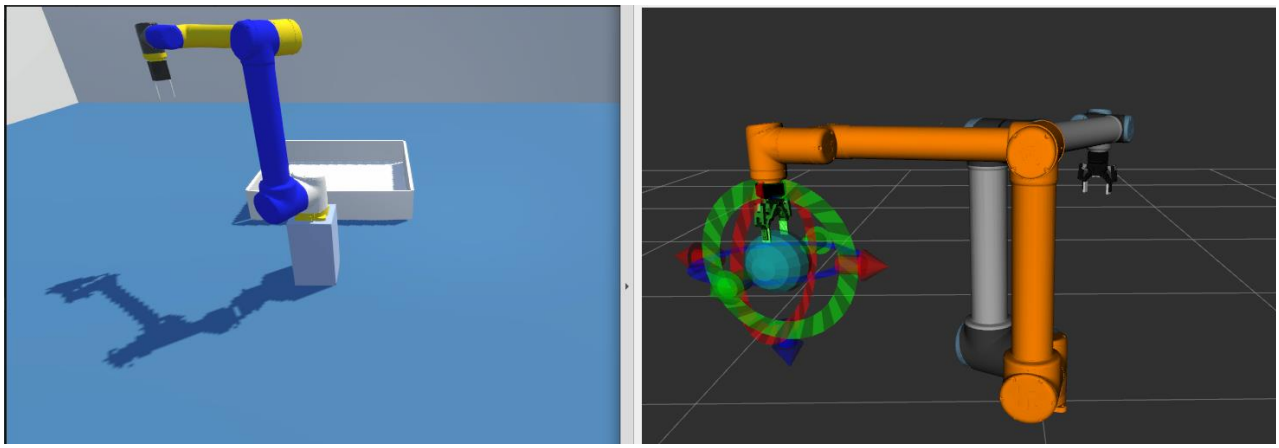

Figure 36. UR10 with integrated UnityROS2Control driver (left) and UR10 in RVIZ (right). The orange ghost of the arm represents the goal position. This is set using the MotionPlanning plugin in RVIZ.

Funded by the European Union

4 Simulation to support learning from demonstration

4.1 Objective

The objective is to record, refine, and pre-evaluate user-demonstrated assembly skills in a simulated environment.

While recording of skills will mainly take place in real-world scenarios, the refinement of skills via Inverse Reinforcement Learning can only be done in simulation, as it usually requires many evaluations. As we focus on assembly skills, physically realistic contacts are a key requirement for the simulation environment.

4.2 HARTU solution

Learning and control of assembly skills involves complex contact forces as well as real-time execution. It was decided to use MuJoCo as a simulator as it allows for multi-contact optimization and real-time performance (>= 1KHz). Other engines have been evaluated (Gazebo with ODE physics, PyBullet), but rejected for different reasons.

4.2.1 Robot Models

The downside of MuJoCo is that it is less feature-rich than other simulators. Especially the creation of complex scenarios can be tedious job due to the lack of interfaces. To ease the burden of the programmer, two extensions have been developed:

- A ROS2 interface for MuJoco which allows to access sensors and actuators in real-time.
- A converter tool to transfer robot models from URDF to the MuJoCo-native MJX format.

Up to now, two robotic systems have been modelled and tested in MuJoCo:

- Franka Emika Panda (7 DoF)
- Dual-arm KUKA iiwa robot (14 DoF)

Both systems could be simulated with 1 KHz framerate.

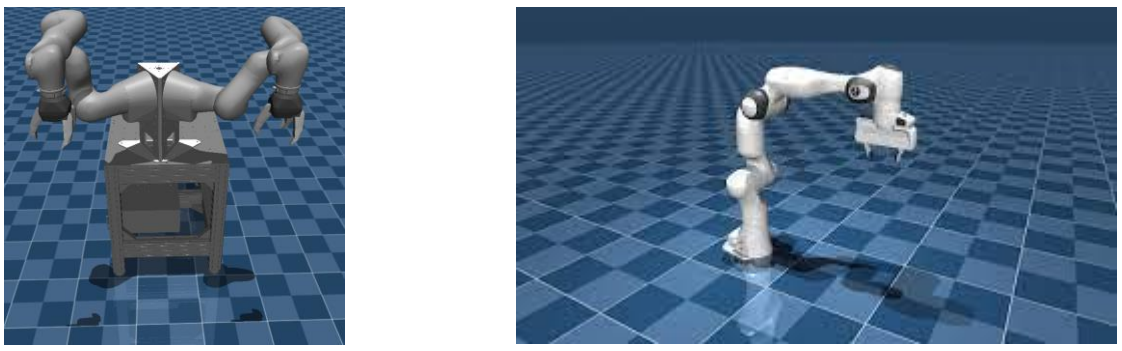

Figure 37. MuJoCo Simulator: KUKA Dual arm robot (left), Franka Emika Panda (right)

Funded by the European Union

4.2.2 Task Board

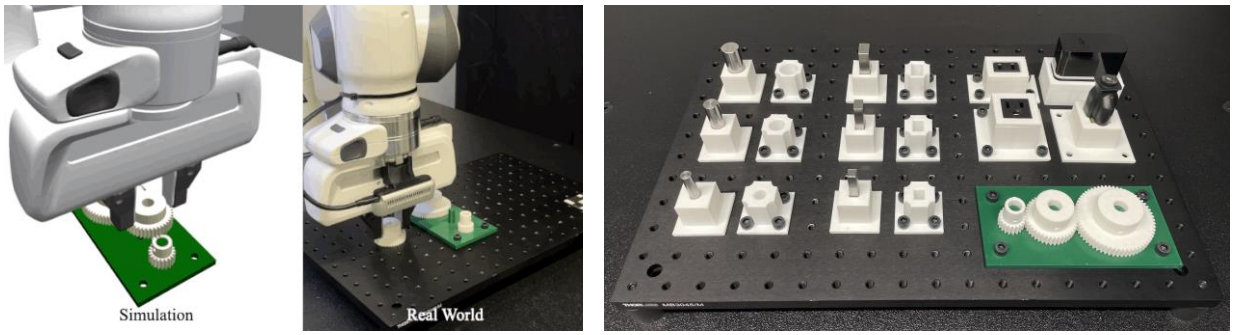

Figure 38 Nvidia Task Board (left), Task Board at DFKI (right)

For evaluation purpose, a task board (Figure 38, right) has been constructed, similar to the Nvidia Task Board[3] (Figure 38, left). It contains components of typical assembly tasks (gears, different peg-in-hole, plugs, …). The task board is available as CAD in the MuJoCo simulation. It can be used to

- Benchmark robotic assembly solutions.
- Improve physical models of robots and objects in the simulator.
- Assess methods to reduce the simulation-reality gap.

4.2.3 Interfaces

To connect with the real-time control loop, a ROS2 interface has been developed for MuJoCo. It is based on the ROS2 Control framework and allows to switch between simulated and real robot without changing any of the higher-level interfaces.

The ROS2 wrapper for MuJoCo will soon become open source on Github.

4.2.4 URDF2MJCF Converter

ROS2 uses URDF to model the kinematics and dynamics of robotic systems. For convenience a URDF2MJCF converter has been developed, where MJCF is the native XML-based modelling format in MuJoCo. After converting the URDF model to MJCF, usually some additional changes must be made manually, as a URDF model usually does not include all the MuJoCo features, e.g., surface friction.

[3] https://github.com/NVlabs/industrealkit/tree/main

Funded by the European Union