Politecnico di Torino

Politecnico di Torino

Master's Degree in Computer Engineering – Artificial Intelligence and Data Analytics

Deep Learning-Based Depalletizer

Object Localization with Real and Synthetic Data

Supervisor:

Prof. Marina Indri

PhD. Enrico Civitelli

Candidate:

Nicolò Bonincontro

December 2025

Abstract

This thesis investigates the use of **NVIDIA Isaac Sim**, a robotics simulation platform built on **NVIDIA Omniverse**, designed to simulate and validate AI-driven robotic solutions within physically realistic virtual environments. The main objective is to assess the effectiveness of synthetic datasets generated through Isaac Sim and their applicability in industrial scenarios such as depalletization and bin-picking.

Synthetic data—artificially produced information obtained from virtual simulations—represents a strategic asset when real data is costly, difficult to acquire, or prone to bias. To this end, a test environment faithfully reproducing real operating conditions was developed in Isaac Sim, introducing variability in position, orientation, lighting, textures, materials, and in the physical characteristics of boxes/objects and containers. The simulations generate multiple types of outputs (RGB images, instance/semantic segmentation maps, depth maps, and surface normals), which are subsequently employed to train modern segmentation networks (e.g., *Mask2Former*).

In addition, two robotic hand-simulations: vacuum and parallel-jaw, were implemented to produce labeled results and gripper information, which also included unsuccessful attempts on non-graspable objects. Different methods were used to train the Mask2Former model: supervised on real, synthetic, and mixed datasets; domain adaptation; and synthetic-only (zero-shot sim-to-real). A GUI, API, and AI-assisted annotation tool based on SAM are developed to improve the efficiency of data generation and labeling. The findings demonstrate the potential of synthetic data and domain adaptation to reduce the sim-to-real gap for industrial vision tasks.

Acknowledgements

Completing this journey has not been an achievement of mine alone, but the result of many people who, in different ways, gave me support, trust, and motivation along the way. I would like to thank Professor Marina Indri for the availability and care with which she supervised my thesis work, and for the suggestions that helped me give this project a clear direction. A truly special thank-you goes to my tutor at Comau, Enrico Civitelli, who has followed me throughout these six months of thesis work with great expertise and patience. His help was essential not only from a technical standpoint, but also in the way I learned to approach problems: with method, practicality, and critical thinking. Thank you for the time you dedicated, for your advice, and for helping me grow throughout this experience. I thank my family. Thank you to my parents for their constant support, for the sacrifices they made without ever making me feel their weight, and for always giving me the freedom to build my own path. I also thank my brother for his everyday presence and for that kind of support that often doesn't make noise, but holds a lot. A big thank-you goes to my friends and university classmates, to those who shared with me lectures, projects, nights before exams, and all that organized chaos that makes university a real experience. With you, this journey was lighter and, above all, more enjoyable. And then I thank my girlfriend. We met in high school, we truly found each other at university, and in between we've come a long way together: shared time, adventures, simple moments and tougher ones. You became an important part of my life naturally, without big speeches, but through your presence and your actions. Thank you for being there through these years, for your patience when I was absorbed in the thesis, and for your ability to keep me balanced without ever letting me lose sight of what really mattered. This thesis closes an important chapter and opens another, and I'm happy to do it carrying with me everything I've gained thanks to you all.

Thank you, everyone.

List of Figures

Table of Contents

Chapter 1

Introduction

In the past few years, the rapid development of artificial intelligence and computer vision had a profound influence on the realm of industrial robotics. The current paradigm relies on an increasingly stronger linkage of physical and digital systems, in which intelligent automation is a fundamental pillar. Difficult tasks, such as bin-picking (retrieving objects from bulk containers), depalletizing, and autonomous manipulation, require extremely capable robotic perception systems with the ability to identify and localize objects precisely in unstructured, cluttered, and highly variable environments.

The functionality of such systems depends heavily on the deep learning models powering them, which require enormous quantities of high-quality data for training. The traditional approach, based only on the collection and annotation of real data, however,shows severe limitations in both its effectiveness and scalability. These include:

- **High Cost and Time**: Gathering real-world data is an expensive process. It entails building physical scenarios, using specialized hardware (robots, sensors), and, above all, a very time-consuming and costly manual annotation process, which is the real bottleneck of the development process.
- **Poor Scalability**: Generating millions of data samples, as often required for the generalization of newer neural models, is logistically difficult and sometimes impossible, especially if it requires disrupting normal production operations.
- **Insufficient Diversity and Coverage of Edge Cases**: Real data collected, however numerous in quantity, are only a subset of the possible configurations a system would encounter in the production environment. It is difficult to reproduce severe conditions (edge cases) in a systematic and controlled manner, such as adverse lighting conditions, complex obstacles, or rare but potential object occurrences.

- **Quality and Accuracy of Annotation**: Manual annotation is sensitive to human errors, anomalies, and impurities, and therefore it is difficult to achieve accurate ground truth, especially for examples such as partitions that require pixel-level outlines.

To deal with these shortcomings, synthetic data has emerged as a strategic and promising solution. Produced in controlled, photorealistic and physically realistic simulated worlds, synthetic datasets allow to reduce the cost of traditional methods and to generate large scale, fully annotated and diversity-rich data in a fraction of the time. Yet, this strategy's success is thwarted by the so-called *sim-to-real gap*: the disparity between the distribution of simulated data and real-world data. The gap, which is mainly due to less than realistic lighting, texture lack, physical simulation errors, and unmodeled sensor noise, is the main hindrance to directly transferring models trained solely in simulation.

Here, NVIDIA Isaac Sim, developed on the NVIDIA Omniverse platform, is a cutting-edge simulation tool, tailored particularly for AI and robotics research. It enables the creation of photorealistic virtual worlds, as well as the simulation of physics-based interactions, and the key data such as RGB images, depth maps, segmentation masks, are automatically rendered, without which computer vision models cannot be trained. Being such a tool, it is the perfect one on which one can explore the potential of synthetic data towards industrial application. Developed entirely at **Comau**, a global leader in automation, this thesis aims to bridge the gap between simulation innovation and industrial practice. Guided by real-world manufacturing challenges, the research focused on delivering a practical, validated methodology for robotic perception that moves beyond purely theoretical work.

1.1 Motivation and Objectives

The main goal of this thesis is the construction and validation of a systematic method that exploits the use of synthetic and real-world data to learn high-performing perception models for industrial robots, directly tackling the sim-to-real gap problem. It attempts to show that a systematic solution, mixing simulation, real world data, and latest AI architecture, can speed up the construction time and enhance the efficiency of autonomous automation systems in a very significant way.

The aim of the present thesis is thus two-fold:

1. **Generation and Analysis of Synthetic Datasets**: Develop a large-scale simulation environment in Isaac Sim to generate synthetic datasets (RGB images, segmentation maps, depth maps, surface normals) with systematic and controlled changes in appearance of objects, environmental conditions, and simulation of grasp attempt simulations.

2. **Closing the Sim-to-Real Gap**: Learn and transfer the latest approaches to integrate real and synthetic data. It encompasses the considerations of domain adaptation, fine-tuning, and semi-supervised approaches.

1.2 Thesis Contributions

The thesis work offers a number of novel contributions in the robotic simulation and computer vision domains:

- **Synthetic Data Generation Pipeline**: Creation of a modular, extendable simulation pipeline in NVIDIA Isaac Sim that enables the stochastic construction of scenes, asset variability (objects, containers, textures,lights), and automated collection of multimodal data.
- **Simulation of Robot Grasps**: Integrating and testing two grasp modalities, parallel-jaw gripper and vacuum grip, and diligent logging of results (success, failure, drop, collision), and handling ungraspable objects as part of simulating the conditions in the real world.
- **Integration of the User Interface and the APIs**: Creation of a Graphical User Interface (GUI) and REST API endpoints that can handle outgoing requests for dataset creation and user-friendly interaction with the simulation environment.
- **Hybrid Workflow with SAM and Human-in-the-Loop**: Developing a hybrid annotation workflow that incorporates box-to-mask conversion through the employment of SAM along with a light correction tool by means of a custom-built web app that dramatically reduces the expense of marking up real-world datasets.
- **Experimental Comparison with Mask2Former**: Training and testing segmentation models on solely synthetic, solely real, and hybrid data, along with comprehensive training paradigm exploration like zero-shot sim-to-real transfer, fine-tuning, and semi-supervised learning.
- **Evaluation of Domain Adaptation Efficacy**: Qualitative review of techniques used to thwart the sim-to-real gap and quantitative evaluation of their impact on performance in industrial robotics tasks.

1.3 Thesis Structure

The thesis is organized as follows:

- **Chapter 2 - Background and Related Works** introduces the theoretical background and the state of the art, such as synthetic data, the NVIDIA Isaac Sim platform, the sim-to-real gap problem, and the architectures of the principal models used: Mask2Former and SAM.
- **Chapter 3 - Dataset Creation** details the process for the creation of the synthetic dataset and the refining of real data, introducing the simulation system architecture, scene randomization methods, data acquisition and user interface implementation.
- **Chapter 4 - Experimental setup and implementation details** shows the setting of experiments, specifies dataset, training methods, assessment matrix and hardware/software setup.
- **Chapter 5 - Experimental results** presents and analyzes quantitative and qualitative results of the experiments.
- **Chapter 6 - Discussion** analyzes the experimental results, including their implications, effectiveness of planned approaches and research limits.
- **Chapter 7 - Conclusion and future work** provides conclusions and discusses possible directions for future research.

Chapter 2

Related Work and Background

This chapter sets out the key ideas and background research that underpin the thesis. It deals with some themes: contribution of synthetic data for computer vision and robotics, industrial applications of the NVIDIA Omniverse/Isaac Sim [1] platform in practice , problems with the sim-to-real gaps and adaptation solutions, and key models for the project, the Segmentation Anything Model (SAM) [2] and the Mask2Former [3].

2.1 Synthetic data in computer vision and robotics

Synthetic data became more important in computer vision as an annotation cost-reducing and development time-saving tool. It is best applicable to perception robotics problems, such as object detection, segmentations, and manipulation planning, where large expert-annotated datasets do not exist.

Various synthetic datasets have provided evidence for the potential of simulation for training vision-based robotic perception systems. SceneNet RGB-D [4] provided millions of photo-realistic indoor scenes with dense labels, making synthetic data perspective on the tedious work of hand-labelling millions of Samples. Falling Things [5] took the idea one step further by applying the concept to 3D object pose prediction for cluttered, physically-realistic scenes, aiming squarely at manipulation and bin-picking issues. Broadly speaking, the idea behind domain randomization [6] was to show that adding visual variability during simulation-based training would dramatically narrow the sim-to-real gap, so that synthetics-only trained policies are capable of generalising to the physical environment. Those techniques are well-suited for industrial robots, where synthetics are capable of replacing cluttered bins, dynamically changing product poses, infrequent failure modes too

complicated or too expensive to implement in the physical environment.

Despite all this benefit, though, synthetic sets are necessarily limited: their value will depend not only on the diversity and realism they achieve, but also on their potential for mixture with real data by means of domain adaptation techniques.

2.2 NVIDIA Omniverse and Isaac Sim

NVIDIA Omniverse is an interoperable real-time rendering and simulation platform based on the USD, or Universal Scene Description, standard. It facilitates collaboration, reuse of content, and domain-overspanning integration in robotics, simulation, and CAD, among other applications [7]. On top of that, Omniverse is extended by Isaac Sim with robotics-oriented tools, for example, physics engines, sensor simulation, domain randomization, and robot control APIs. Among the most prominent features are:

- integration of physically based rendering with physics simulation;
- programmable scene randomness for closing the sim-to-real gap;
- the Replicator framework for large-scale dataset generation;
- native ROS/ROS2 and industrial robot simulations support;
- GPU acceleration of parallel simulations.

Due to these characteristics, Isaac Sim is also extensively utilized in robotics research as well as in industrial automation applications (e.g., bin picking, pallet transport, assembling), where it would be expensive or inconvenient to acquire annotated real data. Its success is also validated in recent works, including large-scale synthetic dataset creation for perception [8] , industrial manipulation benchmark datasets, and systems, which utilize Isaac Sim both for training robots and for generating datasets, such as the NVIDIA Isaac Manipulation Dataset [9].

2.3 The sim-to-real gap & domain adaptation

One of the most critical problems in computer vision and robotics is the sim-to-real gap, i.e., the mismatch between models trained in simulations and their performances in reality. The sim-to-real gap occurs due to disparity in data distributions and is normally put down to three factors:

- Appearance gap: variations in texture, lighting, shadow, and material that simulations typically could not emulate with full realism;

- Physics gap: errors in simulation of dynamic contacts, frictions, or deformations;
- Perception gap: the influence of actual sensor imperfections, such as noise, calibration errors, and distortions, that it is hard to reproduce informatively in virtual surroundings.

The research community has also advanced some ways of bridging these mismatches:

Domain Randomization – Introducing large variations in simulation (e.g., randomizing colors, textures, lighting, and camera positions) so that models learn invariances and generalize more reliably to unobserved real-world scenarios.

Domain Adaptation – While diversifying simulation, these techniques aim at aligning synthetic and real data distributions. One of the most influential works in such a direction is Unsupervised Domain Adaptation by Backpropagation by Ganin et al. (2015) [10], who pioneered the Domain-Adversarial Neural Network (DANN). It utilizes adversarial training by a gradient reversal layer such that the feature extractor is compelled to learn domain-invariant representations (see Figure 2.1). As a part of my thesis work, I have used the DANN methodology, thereby directly evaluating the performance of adversarial domain adaptation in tackling the sim-to-real transition in robotics perception problems.

Hybrid Techniques – Combining synthetic and real-world data. Typical approaches are pretraining with large synthetic sets and then fine-tuning on small annotated real sets, or by applying unsupervised or semi-supervised schemes with unlabeled real data. It is typical to use image-to-image translation systems like CycleGAN [11] to make simulated scenes more real prior to training.

In this thesis, we decided to go with DANN from the various adaptation techniques, which gives us the possibility to maintain the scalability of synthetic data while achieving a good real-world performance.

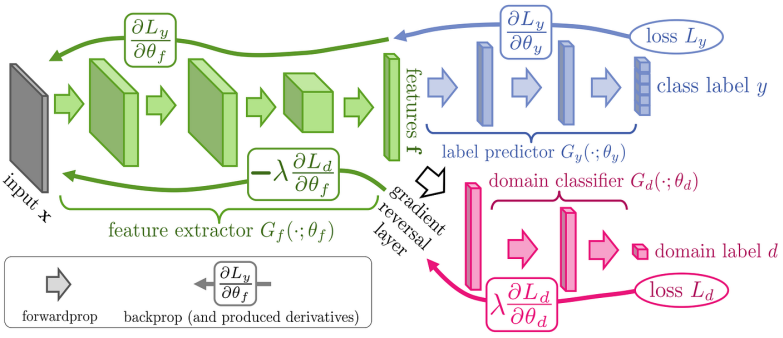


Figure 2.1: Domain-Adversarial Neural Network

2.4 Mask2Former: core architecture & applications

Mask2Former is an integrated transformer architecture for semantic, instance, and panoptic segmentation, put forth by Cheng et al. [3] (see Figure 2.2).

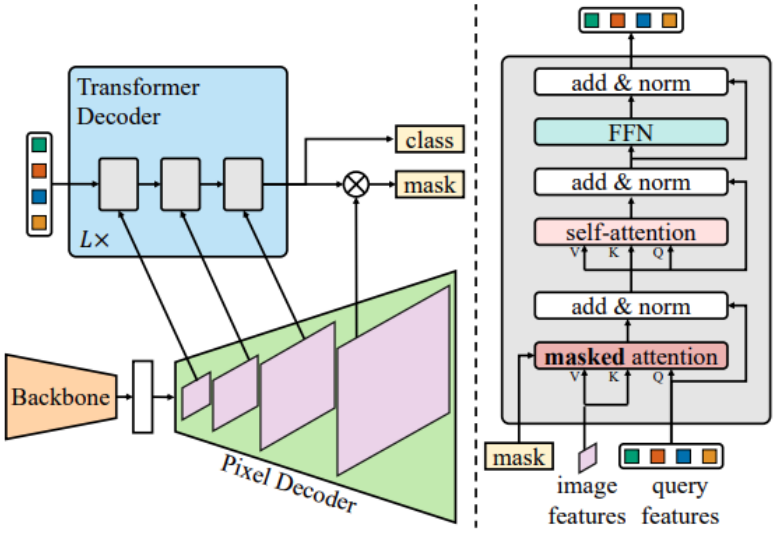


Figure 2.2: Mask2Former architecture diagram [3]

Its main contribution is in terms of the masked attention, through which learnable queries can iteratively attend to relevant regions in the images and region-level predictions refine sequentially. It deviates from pixel-wise classification but rather utilizes a set-based formulation, thus enabling the model to scale better and in a more flexible manner on varying segmentation tasks.

State-of-the-art results on large benchmark datasets are attained by Mask2Former and it is particularly successful in cluttered and unstructured scenes, in which many objects could overlap or hide one another. Its one-size-fits-all framework is also a virtue, as the identical architecture is applicable in principle to semantic, instance, or panoptic segmentation, and hence experimentation is minimized as well as task-specialized design decisions. Within the purview of this thesis, I used Mask2Former in its "small" form in particular for instance segmentation task. It was used in preference to larger variants due to their balance of accuracy and computational needs: the compact model retains the strengths of the full architecture but is agile enough for practical training as well as deployment in resource-limited experimental contexts.

2.5 Segment Anything Model

Segment Anything Model (SAM) is a large foundation model for image segmentation that was introduced in 2023 by Kirillov et al. [2]. It was trained with billions of mask annotations, and it is a promptable model that can create high-quality segmentation masks from highly varying inputs such as points, bounding boxes, as well as from text. Its prowess is in its exceptional few-shot generalization, with which it can excel in varying domains with absolutely no task-specific retraining. Among annotation cost and time saving solutions for dataset prep pipelines, SAM stands out as an exceptionally successful choice. The data preparation process flow of the thesis utilized the Segment Anything Model (SAM) to make the process efficient. The process utilized bounding box annotations as the source of the ground truth, automatically translated by the SAM into segmentation masks.

Chapter 3

Dataset Creation: Synthetic Generation and Real Data Refinement

This chapter is focused on the development of the pipeline for creating the synthetic dataset using Isaac Sim and on annotations and corrections of the real dataset.

3.1 Synthetic Dataset Generation

3.1.1 Overall system architecture

The overall organization of the synthetic dataset generation project is split into three functional modules related to three phases of the process: scene generation, data acquisition, and grasp simulation. This functional decomposition of the modules into the phases of the process confers flexibility, reutilizability of the software, and the potential for extension of the framework to other industrial applications.

- **Scene Generation**
  It Surfaces, objects, and containers are used in this phase to create the world in NVIDIA Isaac Sim in an automated mode. Random but physically plausible variability in material, texture, lighting, and object pose is used by the system to create a new scene in every generation cycle. The PhysX dynamics engine provided with Isaac Sim uphold physical coherence by the management of gravity, collision, and body stabilization.

- **Data Acquisition**

After the scene is stabilized the automated data acquisition module is launched through the Omni.Replicator framework. This is the module that defines the virtual sensors (RGB, depth, normals, segmentation) and the synchronization of the acquisition of the different types of output.

- **Grasp Simulation**
  The final module handles the robotic simulation of grasping actions, testing two different end-effectors, including a parallel gripper and a suction gripper. The simulations allow for the recording of the outcome of each attempt (success, failure, collision, premature release). This information enriches the dataset, making it suitable not only for perception tasks, but also for studies in learning-based manipulation and automatic planning.

3.1.2 Assets management

The objects utilised within the simulation are items with mixed shape and size, boxes and industrial bins, chosen on the basis of the type required dataset. The system has three various asset management modes that serve to allow the system to be flexible and accommodate various usage settings. These are the different methods you can use:

- Procedural generation, to automatically generate parallelepipeds with variable shape and size within some configurable range. With this method, one can quickly obtain a high diversity of samples that can be used to train strong computer vision models to generalize to geometric variations.

- Predefined assets chosen from an internal catalog of models available by default within Isaac Sim. Each asset is uniquely represented by link and materials and physical properties set to match the simulator's needs.

- Import of external models, possible through .obj files inserted within an appropriate directory. One can thus introduce custom objects within the data set and make the pipeline tractable to specific industrial case studies or real-world applications.

When, however, you wish to create a dataset filled with generic items, procedural generation is not implemented. For the boxes and containers, the system also allows the user to choose freely among the models available by Isaac Sim and these consist of crates and pallets and other common objects within warehouse and logistical settings. The full generation configuration — asset data, dimensional intervals, and physical parameters — is controlled by an input YAML configuration file.

3.1.3 Scene variability

In synthetic data generation, variability serves a critical function: it produces datasets with greater realism and diversity, which in turn directly improves the generalization performance of computer vision models. For this purpose, I included a domain randomization system that randomly alters various aspects of the scene within each cycle of generation. All the parameters can be configured via a YAML config file that allows to set the range over which the different attributes can vary. The things to be randomized primarily are:

- **Lighting:** Light sources differ by quantity, kind, location, strength, and hue to simulate various working conditions (i.e., direct, reflected, and diffuse lighting). Three base lights are employed—Dome Light, Sphere Light, and Distant Light—which are randomly chosen and parameterized within the ranges set out by the config file. This randomness supports the simulation of inhomogeneous scenes and renders the model more insensitive to the lighting variations common to real-world scenes.

- **Materials and Textures:** Object surfaces, container surfaces, and floor surfaces are altered through randomly mixed combinations of textures. These textures can be edited by the user through the simple addition of .png files to the special folder (textures/). Materials and properties are generated and set on the fly at each generation with parameters involving metallicity, roughness, and reflection moving within the range set out in the config file. This enables the copying of the immense diversity of surfaces and materials within the real world.

- **Pose and Orientations:** Objects are randomly dropped at various positions and orientations in the workspace. Then Isaac Sim's physics simulation automatically computes the final state of equilibrium by making the generated configurations plausible.

- **Camera Parameters:** the optical parameters of the virtual camera, i.e., its focal length and its aperture, are held fixed to ensure consistency between exposures. But the height of the camera with respect to the scene plane is varied within a range configurable by the user (minimum and maximum heights defined within the YAML file). In this manner, different view points of acquisition are retrieved to simulate the sensor positioning variations typical of real industrial scenes.

The combination of these randomizations enables the generation of large, photo-realistic, and highly diverse datasets, enhancing the robustness of deep learning models during the training phase.

3.1.4 Automatic scene generation pipeline

There are steps that must be followed in the automated scene creation process. The first step is to make the floor and the container (or pallet) and give them a random texture from the script's folder. You can also use the asset's default texture. Next, you put in one to three different types of light sources, like a Dome Light, Sphere Light, or Distant Light. They are picked at random to make different lighting conditions. The system can make invisible walls around the spawn area to make it more likely that things will fall correctly into the container or onto the pallet. These parts keep things from spreading outside the camera's view during the release phase. After the environment is set up, the system creates the objects. Each object is created at a certain height, which is chosen based on the settings in the configuration file. Each object gets a random texture, and then it is let go so that the physics simulation can figure out how it will fall and where it will end up in the scene. The Omni.Replicator module turns on at the end of this phase. This module takes care of automatic data collection. Replicator records and saves the different versions of the scene.

At this point, two algorithms are run :

1. **Calculating the stereo depth map:** The first algorithm uses the stereo images that were made to make a more accurate depth map. It uses the Stereo Semi-Global Block Matching (SGBM) algorithm, which is part of the OpenCV library and is called by the `cv2.StereoSGBM_create` function. This method finds matches and calculates the disparity by comparing small pixel blocks (`blockSize`) between the left and right images. From the disparity, depth is calculated. The `P1` and `P2` parameters are used to punish changes in disparity between neighboring pixels.This way we can create a more realistic depth map.

2. **Enhancement of Instance Segmentation Masks:** The segmentation masks are processed by second algorithm. It makes use of the connected-component analysis implemented by the `cv2.connectedComponents` function. This algorithm finds all connected pixel areas (also known as pixel "groups" or "islands") by analyzing the mask of a single object. An object's mask may break up into several non-contiguous pieces if it is partially obscured by another. By separating and labeling each of these elements separately, this technique produces masks that make segmentation network training easier.

The system starts the grasp simulation after the post-processing stage, where various end-effectors and grasping techniques are evaluated. The generation cycle is finished with this phase, which is covered in more detail in the following section. The scene is entirely reset at the conclusion of every operation, enabling the creation of a new one.

3.1.5 Data acquisition with Replicator

The acquisition component uses Isaac Sim's native Omni.Replicator framework to create annotated datasets. Replicator enables the definition of virtual sensors, synchronization of rendering with the physics of the scene, and automatic generation of the ground truth needed for neural network training. Among the data types produced are:

- PBR-rendered photorealistic RGB images (see Figure 3.1);
- and automatically generated instance and semantic segmentation annotations through object tags (see Figure 3.2);
- surface normals and depth maps (see Figure 3.3);

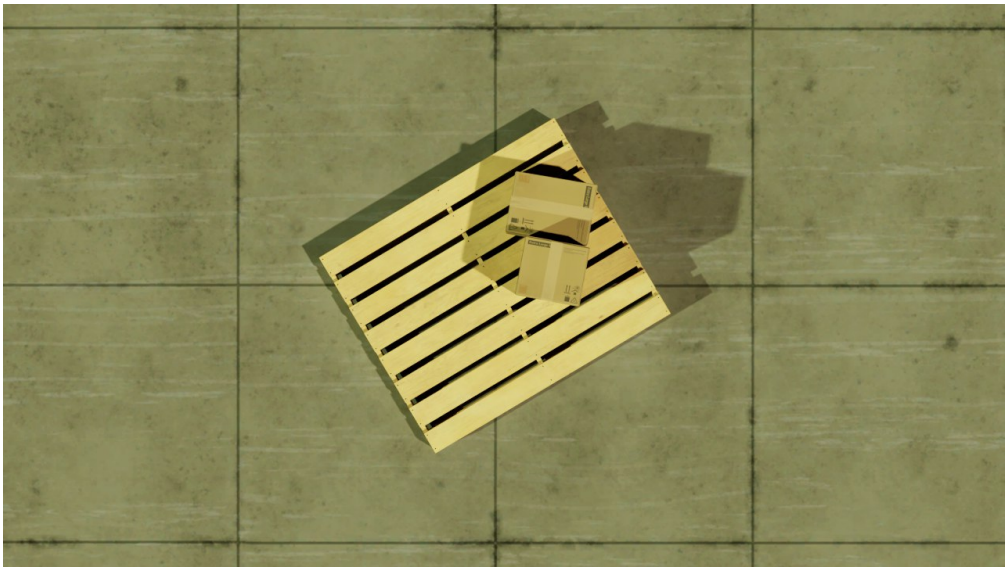


Figure 3.1: RGB Synth image

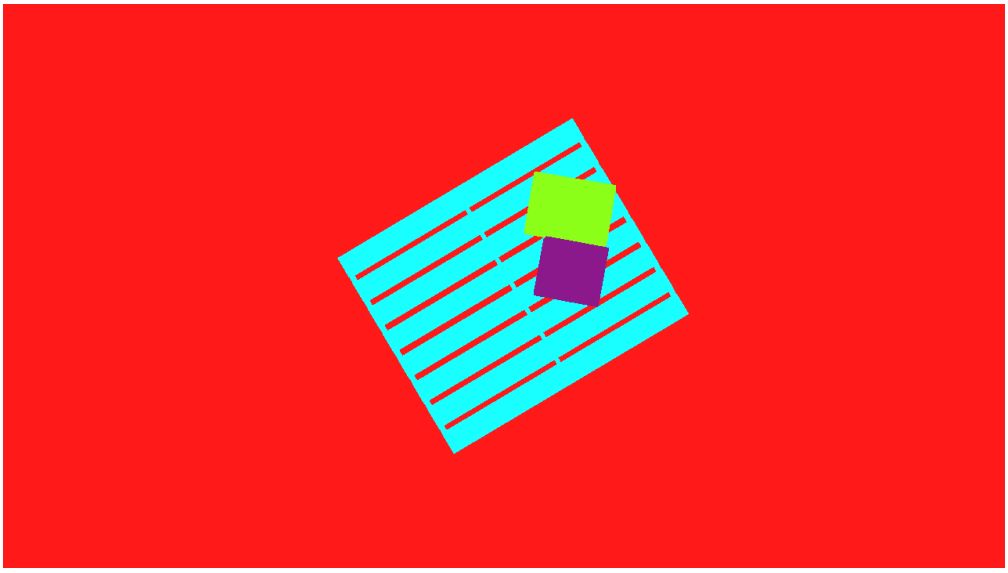


Figure 3.2: Segmentation Synth image

Figure 3.3: Depth Synth image

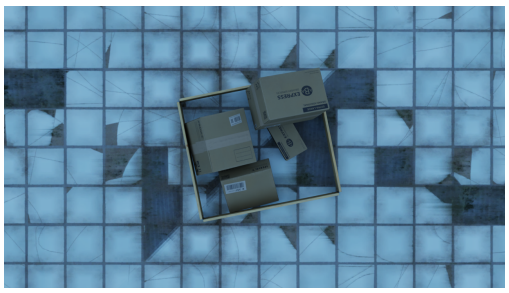
Here are some generated RGB images (See Figure 3.4).

Synth image 1

Synth image 2

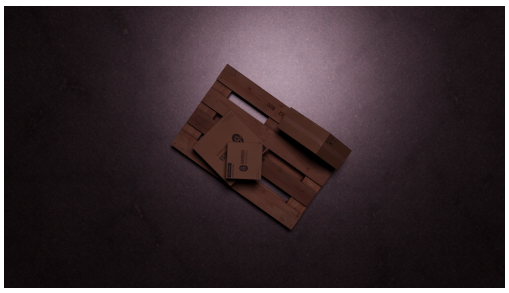
Synth image 3

Synth image 4



Synth image 5

Synth image 6



Synth image 7

Synth image 8

Figure 3.4: Set of synth images.

3.1.6 Grasp simulation

Grasping simulation is the last stage in the data generation pipeline. Its function is to determine which grasps are successful with the objects that are in the scene. Consequently, the data for perception as well as for training robotic manipulation policies can be created.

We have implemented the two main kinds of end-effectors:

- Suction Gripper
- Parallel Gripper

Suction Gripper

Suction gripper is an extensively used grasping device in industrial environments; it works great with objects that have flat surfaces. As for the simulation, the geometric cone (`GripperCone`) is a representation of the end-effector.

Operating Principle. The adhesion mechanism is simulated with the help of the `Surface_Gripper`, which is a special feature in Isaac Sim that creates a high-adhesion physical joint between two bodies. The method enables one to have the fine control over the physical parameters of the grasp through the configuration file, where one can set:

- **Maximum Force (`absolute_grip_force`):** The maximum force suction cup can apply in order to keep the adhesion.
- **Maximum Torque (`torqueLimit`):** Resistance to twisting.
- **Stiffness and Damping (`stiffness, damping`):** Parameters that characterize the joint's reaction to forces coming from the outside.

Phases of testing. The testing process for each object in the scene is divided into three main phases.

Phase 1: Systematic Sampling of Grasp Points. Instead of attempting a single random grasp, the system exhaustively identifies all possible valid grasp points on the object's surface, using the `sample_grasp_grid` algorithm.

1. **Geometry Extraction:** The tool inspects the USD prim of the target object and extracts the entire geometry in the form of a triangle list.
2. **Tilt Filtering:** For each piece of the geometry, it computes the normal (**n**) of the triangle and eliminates surfaces that have an angle with the vertical axis greater than a specified threshold (`max_tilt_deg`).

Mathematical Formulation: The geometry checking criterion, given by (3.1), is done by calculating the dot product between the unit normal vector of the triangle ($\mathbf{n}$) and the world's "up" vector ($\mathbf{v}_{\text{up}} = [0, 0, 1]^T$).

$$\cos(\theta) = \mathbf{n} \cdot \mathbf{v}_{\text{up}} \tag{3.1}$$

A grasp is defined as feasible only when $\cos(\theta) > \cos(\theta_{\max})$.

3. **Grid and Ray-Casting:** Firstly, the system outlines a 2D grid (in the XY plane) over the target object. Next, it fires a vertical ray downward from each vertex of this grid to figure out a possible grasp point on the surface of the object.

   The grid density—which consequently dictates the grasp granularity—is a very important configuration parameter. In particular, the `grid_step_m` parameter specifies the distance (in meters) between the vertices: a lower value results in more detailed and finer sampling, whereas a higher value speeds up the process but makes the sampling less detailed.

   Mathematical Formulation: The exact point of intersection is found by calculating t from the geometrical definitions of the ray and the plane of the object's triangle. The ray is given parametrically by $\mathbf{P}(t) = \mathbf{O} + t\mathbf{D}$, and the plane of the triangle by $(\mathbf{P} - \mathbf{P}_0) \cdot \mathbf{n} = 0$.

   O represents the origin of the ray (a point on the 2D grid).

   D is the direction vector of the ray (e.g., the vertical down vector).

   t is the scalar parameter that defines the distance along the ray to the intersection point.

   $\mathbf{P}_0$ is a known point belonging to the plane of the triangle.

   n is the unit normal vector of the triangle's surface .

   The value of t that defines the intersection is computed as:

$$t = \frac{(\mathbf{P}_0 - \mathbf{O}) \cdot \mathbf{n}}{\mathbf{D} \cdot \mathbf{n}} \tag{3.2}$$

   The precise point of intersection on the object, $\mathbf{P}_{\text{intersect}}$, is then calculated using the ray definition: $\mathbf{P}_{\text{intersect}} = \mathbf{O} + t\mathbf{D}$.

4. **Grasp Pose Calculation:** From a surface ray-casting operation, a candidate point on the surface is given. The pose (position and orientation) of the gripper's suction cone is finally determined. The main concern here is to find a rotation that makes the gripper's axis be completely perpendicular to the object's surface at the calculated grasp point.

Mathematical Formulation: The orientation is determined by computing the rotation $\mathbf{R}$ which aligns the standard approach vector of the gripper ($\mathbf{v}_{\text{gripper}}$) with the unit normal vector ($\mathbf{n}$) of the grasped surface. This rotation is represented in terms of the axis-angle representation ($\mathbf{a}$, θ).

The geometric variables that participate are:

$\mathbf{n}$ The unit normal vector of the triangle surface at the point of intersection.

$\mathbf{v}_{\mathbf{gripper}}$ The initial, predetermined approach direction of the gripper (e.g., generally the negative Z-axis: $[0, 0, -1]^T$).

$\mathbf{a}$ The vector that denotes the rotation axis which transforms $\mathbf{v}_{\text{gripper}}$ into $\mathbf{n}$.

θ The angle of rotation about $\mathbf{a}$.

The axis and angle are found by means of vector algebra, as detailed below. The rotation quaternion is consequently obtained from the axis-angle pair:

$$\mathbf{a} = \mathbf{v}_{\text{gripper}} \times \mathbf{n} \tag{3.3}$$

$$\theta = \arccos(\mathbf{v}_{\text{gripper}} \cdot \mathbf{n}) \tag{3.4}$$

Phase 2: Grasp Simulation Execution. A full physics simulation for each candidate pose is run, controlled by a state machine, after the scene has been reset to guarantee that each test is independent. The operations conducted are:

- Kinematic positioning,
- Grasp attempt,
- Dynamic physics activation,
- Execution of a validation trajectory (lifting, moving, and lowering).

Phase 3: Results Analysis and Grasp Map Generation. Each simulation ended with the recording of a status code (`grip_status_code`) that categorizes the result:

- **Successful Grasp (Code 3)**
- **Object Dropped (Code 2)**
- **Failed Grasp (Code 1)**
- **Collision (Code -1)**

The experiments conclude with the **visual representation of the grasp map** based on the accumulated data.

Mathematical Formulation: Each 3D grasp point ($\mathbf{P}_{\text{world}}$) is mapped to 2D pixel coordinates ($\mathbf{p}_{\text{image}}$) by the camera's intrinsic matrix $\mathbf{K}$ and extrinsic matrix $[\mathbf{R}|\mathbf{t}]$ as per the pinhole camera projection formula (3.5).

$$\mathbf{p}_{\text{image}} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{P}_{\text{world}} \tag{3.5}$$

The final output is a single composite image where each grasp attempt is displayed as a colored circle based on its outcome, thus enabling the immediate visual assessment of the grasp strategies. See Figure 3.5 and Figure 3.6.

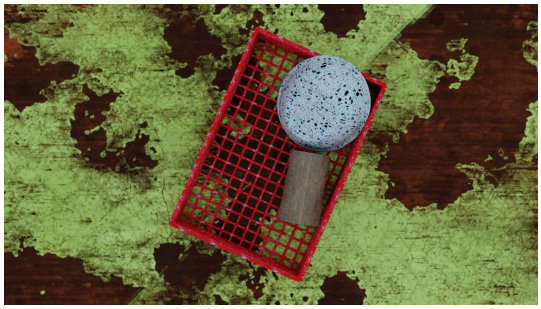


Figure 3.5: RGB Picking Image

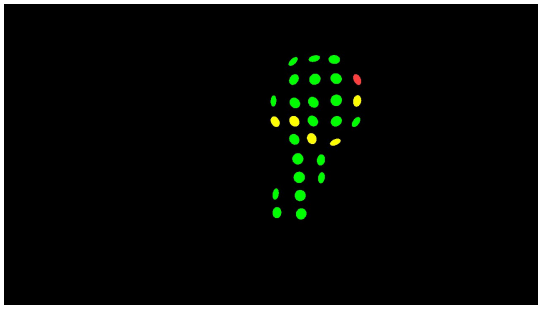


Figure 3.6: Grasp Map Image

Parallel Gripper

While the suction gripper is suitable for flat surfaces only, the parallel gripper becomes a more versatile tool, since it can grab almost any object by simply squeezing it from the sides. For this simulation, a standard industrial gripper model, the **Robotiq 2F-85**, was employed, and it was dynamically loaded into the simulation environment.

The functional verification of a gripping device is something radically different from the set of operations that we usually perform on the suction gripper, and is based on a **Finite State Machine (FSM)** that logically leads the gripper through the consecutive steps of the operation from positioning to lifting the object.

Operating Principle. The grasping mechanism does not use a special joint, but relies on Isaac Sim's standard physics. The stability of the grasp is guaranteed by the two most important factors:

1. **Closing Force**: Control of Grip is achieved by setting target positions for the three actuation joints of the gripper.

2. **Friction**: A material with extremely high static and dynamic friction coefficients has been assigned to the inner surfaces of the gripper's fingers. This is done to completely eliminate the slipping of the object once it is grasped.

The testing phases are divided into three main stages through which the whole process is orchestrated.

Phase 1: Spherical Grasp Point Sampling. This sampling strategy tests approach directions originating from multiple points around the object, thus simulating a varied set of possible real-world scenarios.

1. **Object Information via OBB**: The system first computes the object's Oriented Bounding Box (OBB). The OBB, defined as the smallest oriented bounding box that encloses the object, provides essential geometric properties: the geometric center ($\mathbf{c}$), the size ($\mathbf{s}$), and the object's orientation matrix ($\mathbf{A} = \{\mathbf{a}_x, \mathbf{a}_y, \mathbf{a}_z\}$) corresponding to the OBB axes.

2. **Sampling on a Virtual Sphere**: The `generate_grasp_poses` function generates candidate approach points ($\mathbf{p}$) on a virtual sphere surrounding the OBB.

   Mathematical Formulation: A point $\mathbf{p}$ on a sphere centered at $\mathbf{c}$ with radius r is produced using spherical coordinates, as shown in equation (3.6), where θ and ϕ are random variables used for uniform sampling of the sphere's surface.

$$\mathbf{p} = \mathbf{c} + r \cdot \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix} \tag{3.6}$$

   The radius r is computed as $r = \frac{1}{2}||\mathbf{s}|| + \delta$, where:

   $\mathbf{c}$ is the geometric center of the OBB (vector).

   $\mathbf{s}$ is the vector defining the dimensions of the OBB.

   δ is a safety offset specified by the parameter `GRIPPER_APPROACH_OFFSET`.

   θ, ϕ are the random angular variables used for spherical coordinate generation.

3. **Approach Orientation Calculation**: For each sampled point $\mathbf{p}$, a calculation is performed to define the final 6D gripper pose (position and orientation) necessary for a correct approach.

   Mathematical Formulation: An orthonormal basis for the gripper's rotation $\mathbf{R} = [\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z]^T$ is constructed. The primary approach direction, $\mathbf{v}_x$ (the

unit vector pointing from $\mathbf{p}$ toward the object's center $\mathbf{c}$), is obtained from formula (3.7):

$$\mathbf{v}_x = \frac{\mathbf{c} - \mathbf{p}}{||\mathbf{c} - \mathbf{p}||} \tag{3.7}$$

The components of this formulation are:

$\mathbf{c}$ The geometric center of the OBB.

$\mathbf{p}$ The candidate point on the virtual sphere.

$\mathbf{v}_x$ The resulting unit vector representing the desired approach direction of the gripper.

The other two perpendicular vectors, $\mathbf{v}_y$ and $\mathbf{v}_z$, are sequentially derived via cross products using the longest axis of the OBB ($\mathbf{a}_{\text{long}}$) to guarantee a collision-free and stable lateral orientation. This resulting rotation matrix is then converted to quaternion format for simulation execution.

4. **Safety Filter**: Candidate poses located below a minimum required height from the ground (`MIN_SAFE_HEIGHT_Z`) are systematically removed from the set of valid grasps.

Phase 2: Runtime Execution through a State Machine (FSM) Given a candidate pose, the grasping FSM conducts a simulation by moving through the following states one after another:

1. **`NEXT_POSE` State**: This serves as the initialization and reset state. After saving the output of the last run, it resets the gripper to its fully open configuration and moves it kinematically to the next candidate pose while suppressing any residual velocity.

2. **`APPROACH` State**: Manages the controlled linear movement of the gripper towards the object along the approach vector ($\mathbf{v}_x$). The system, employing a ray-OBB intersection test (Slab method), halts the motion at a specified pre-grasp distance (`STOP_DISTANCE`).

3. **`GRASP` State**: The command to close the fingers is sent to initiate grasping. Grasp success in this state is inferred if the fingers cease closing *before* reaching their fully closed position. This intermediate halt confirms that the fingers have achieved physical contact and are securely squeezing the object.

4. **`LIFT` State**: After the previous state if the gripper has made contact, it will try to lift the object vertically. The distance between the object and the gripper is always monitored to make sure that the object is kept within

a certain tolerance boundary. The operation is classified as successful for this state if the object maintains the grasp and reaches the target height (`LIFT_HEIGHT_Z`).

Phase 3: Results Analysis Following the termination of each FSM cycle, a boolean result is registered to formally classify the outcome of the grasping attempt:

- **Success (`GraspResult.SUCCESS`)**: The operation was successful, meaning the object was grasped, lifted, and maintained stable until reaching the specified target height.
- **Failure (`GraspResult.FAILURE`)**: Indicates a failure, which may include the grasp being empty, the object being released or dropped during the lift trajectory, or the state machine sequence failing to complete successfully.

Unlike the evaluation results for the suction gripper, the data generated by the parallel gripper tests are typically saved to a JSON file for detailed post-analysis.

3.1.7 User interfaces: GUI and REST API endpoints

To control the complexity of the process of creating the data and to offer an efficient means by which to display the results, an overall interface system has been constructed. This has a client-server architecture.

System Architecture

The infrastructure is broken down into two major sections: a backend that performs scene generating and a client that has a graphical user interface (GUI) for visualization and control.

Backend (Server)

The backend is the computational heart of the system. It performs all operations that are resource-intensive, like starting Isaac Sim, handling physics, photorealistic rendering, and running grasp simulations. In order to enable remote control, it provides a sequence of endpoints by means of a REST API. By means of such decoupling, long and complex generation procedures can be started without needing continuous user intervention.

Frontend (Client GUI)

It is a desktop application that can be run in standalone mode to provide an easy-to-use interface. It is intended to provide an interface to initiate and set up

generation jobs in the backend as well as display the generated synthetic data. The server communication is carried out by standard HTTP requests to the REST API.

Technology Stack

The GUI in the application was designed in Python using the CustomTkinter library, which is an author's reimplementation of the standard Tkinter library, seeking to give the widgets a more modern appearance and feel and extreme customizability. Other principal libraries are:

- **Requests:** For communications over http with the API in the backend.
- **Pillow (PIL):** For raster image processing and display.
- **Multiprocessing and Threading:** To keep the interface responsive at all times by threading the 3D visualization and network tasks onto individual threads and processes.
- **Open3D:** As an external viewer for the point cloud data (`.pcd`, `.npy`).

Interface Structure

The main window is made up of two vertically arranged panels (See Figure 3.7).

Control Panel (Left):

It is responsible for the configuration and the initiation of the processes. It holds:

- **Generation Section:** There is a set of checkboxes (`replicator`, `grip`, `gripper`) to choose which of the pipeline modules to run. Two buttons, "Generate Scene" and "Regenerate Data," initiate the respective actions throughout the server.
- **Configuration Editor:** There is a special button that pops up a modal dialog (`YamlEditorWindow`) whereby the `config.yaml` file can be edited in real-time. The file is transmitted to the backend during generation, making it possible to configure simulation parameters in great detail in a permanent way without the necessity of editing the source code.
- **Remote File Browser:** There exists a tree view of the server's file and folder structure that's loaded dynamically. The user can choose the files they wish, download, and show them.

Visualization Panel (Right):

The panel functions as an analysis workspace of the retrieved information.

- **Integrated Viewer:** The panel shows the file's contents when a file is chosen. The file's type is detected by the system, and the display is customized accordingly: images are shown in place, and text files (such as JSON) are presented in a formatted text area.
- **External 3D Visualization:** In the case of complex file formats such as the point clouds (`.npy`, `.pcd`), the client initiates an external process that employs the Open3D library. This design decision (multiprocessing) prevents blocking the main GUI thread, allowing for a seamless user experience.

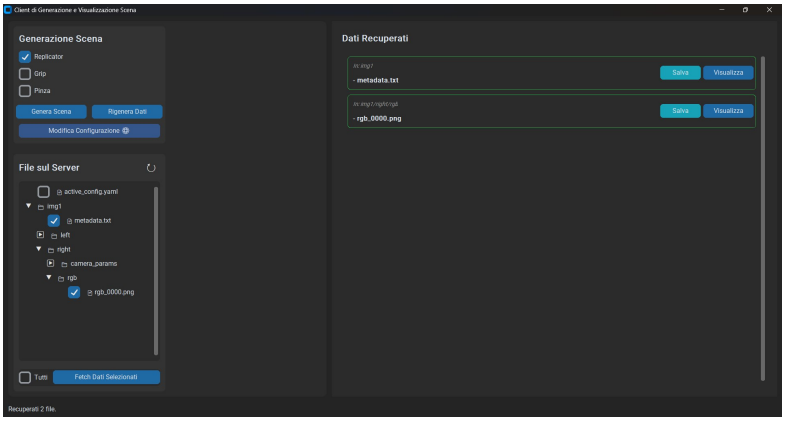


Figure 3.7: Client interface

REST API Endpoints (Backend)

The backend provides a streamlined yet flexible REST API so that the client can communicate with the generation pipeline. The most essential endpoints are shown as follows.

`GET /list_files`

- Lists all output files that exist in the server.
- **Payload:** None.
- **Response:** A JSON object with the list of file paths. E.g., `{"status": "success", "files": ["img0/rgb.png", . }`.

`GET /get_document`

- Downloads a file directly from the server. The path can include subdirectories.
- **URL Parameters:** The complete path to the file being searched for.
- **Response:** The raw binary content of the file (e.g., `image/png`).

`POST /generate_scene`

- Starts a new scene generation session at the beginning.
- **Payload:** A JSON object with the chosen options (`{"options": [.]}`) or ideally a `multipart/form-data` payload containing the options as well as the user-defined `config.yaml` file.
- **Response:** A JSON confirmation response. Eg, `{"status": "success", "message": "Generation started."}`.

`POST /regenerate`

- Initiates the regeneration process, typically using an active scene to generate new data (e.g., re-running the simulation of the grasp by itself).
- **Payload:** Similar to `/generate_scene`.
- **Response:** A JSON confirmation message.

3.2 Real Dataset Correction and Integration

3.2.1 Box-to-Mask conversion with SAM

The available real-world dataset had a limitation in that the annotations were only in the form of the bounding box coordinates. However, the problem to be solved was instance segmentation. Thus, the masks for the whole dataset would have had to be created from scratch, which is a very time-consuming and labor-intensive process that practically blocks the development.

We implemented a semi-automatic conversion method based on the Segment Anything Model (SAM) to get through the limitation. SAM has different input modalities (prompts) that help the segmentation process:

- **Point Input:** The user may indicate one or more points on the image to represent the object of interest (foreground points) or the background to be excluded (background points). Usually, a single point is enough to segment unambiguous objects.

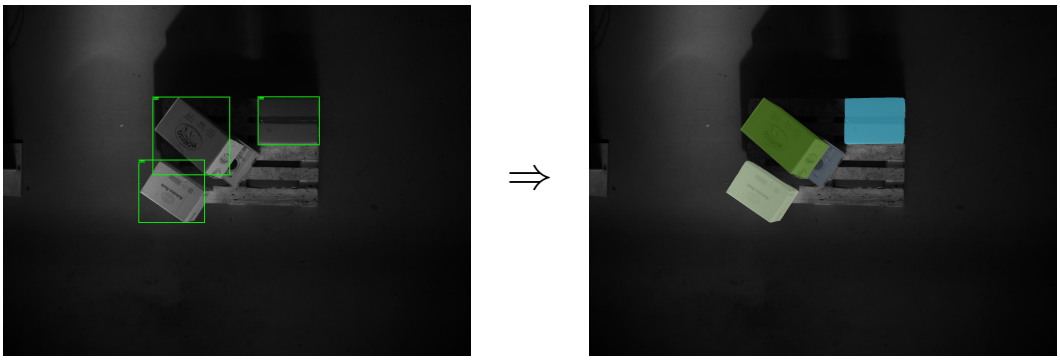
- **Bounding Box:** When a bounding box is given around an object, SAM is limited to find and segment the most obvious object within that box. This prompt is very effective, because it gives the model a clear spatial context.

- **Approximate Mask:** One may give SAM a low-resolution or inaccurate mask (e.g., one that is quickly hand-drawn). The model treats this as a zero point to eventually come up with a highly detailed and more precise version, thus a very strong tool for refinement.

- **Text:** Although it wasn't a core feature of the original model, subsequent extensions and systems have incorporated text prompts to segment objects referred to by words (e.g., "segment the blue chair").

Adopted Strategy: "Box-to-Mask" Conversion

Even though our real-world dataset was annotated with bounding boxes, and thus it seemed straightforward to choose this input mode for segmentation, the preliminary analysis of the data revealed a major limitation: the original annotations were incomplete in many cases. For a large number of instances, the bounding boxes did not cover the whole object but only one visible face or a partial segment. If these partial bounding boxes were used as input prompts for SAM, the segmentation masks would have been correspondingly incomplete, so the whole process would have been ineffective. In order to get past this obstacle and perform complete object segmentation, a very important step in data preprocessing was set, deciding not to feed the original annotations directly to SAM but to enlarge each bounding box programmatically. This was done by increasing the size of the box by a certain percentage in each direction. The expansion of the bounding box turned out to be a crucial point in the strategy. When given a larger initial bounding box, SAM had to look at a larger visual context. As a result, the model no longer segmented the portion that had only been annotated previously. Instead, the most visually salient and complete object instance contained within the new, enlarged box was accurately recognized and fully delineated. In this way, the enlarged bounding box serves as an effective "hint" for the segmentation model to isolate the entire desired object instance. This approach not only made it possible to use the existing imperfect annotations that had been made, but also facilitated their automatic correction, thus, increasing the robustness and effectiveness of the segmentation process to the maximum extent (see Figure 3.8).



box label segmentation label

Figure 3.8: Conversion dataset

3.2.2 Human-in-the-loop annotation correction

Even though SAM is precise, the masks that are generated can be less precise in scenarios that are complicated, such as partial occlusions or objects that have ill-defined edges. In order to produce a dataset of the highest quality, it was necessary to implement a vital human-in-the-loop validation process via a custom-built annotation application.

The platform is a web application developed in React (see Figure 3.9). The interface allows a human operator to load an image with the masks automatically generated by SAM and to make accurate and efficient corrections.

The significant features of the application are:

- An interactive canvas that allows smooth navigation through zoom and pan, which is very important for working on high-resolution details.
- Direct polygon editing tools, which enable users to refine the edges of the masks just by dragging the existing vertices.
- The possibility to create new masks from scratch to annotate objects that SAM may not have seen.
- A side panel for managing individual annotations (rename, hide, delete).

After validation, the corrected masks are saved in JSON format that can be easily integrated into the training pipeline.

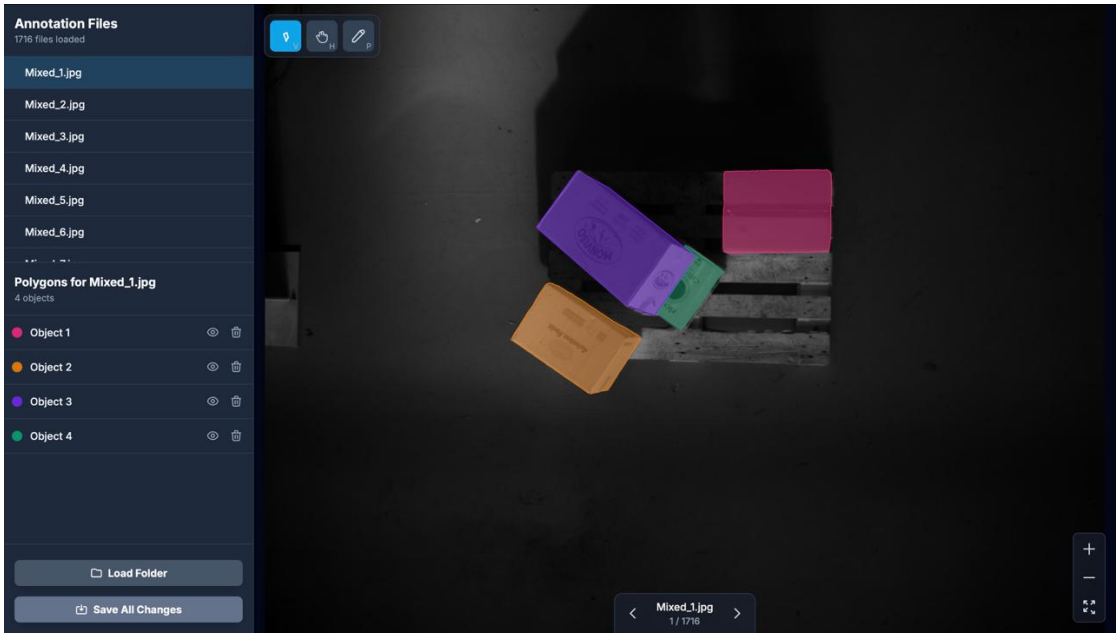


Figure 3.9: React app used for fixing annotation

Chapter 4

Experimental Setup and Implementation Details

The aim of this chapter is to first identify the datasets that were used, the training configurations, the experimental approaches, the evaluation metrics, and the hardware and software environment

4.1 Datasets and splits

Our experimental strategy hinges on the use of three separate datasets, which have been intentionally designed to facilitate thorough model training and extensive model testing against the sim-to-real challenge.

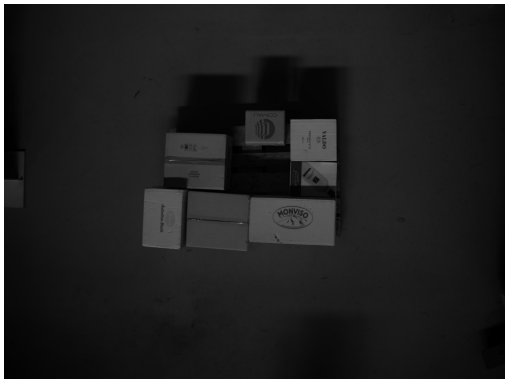
4.1.1 Dataset Composition

Synthetic Dataset (Source Domain): With the help of the modular pipeline described in Chapter 3, the synthetic dataset is produced; it is the main source for data augmentation and domain randomization techniques. The dataset includes about 1,800 visually different images in total, and for each image, multimodal ground-truth annotations (RGB, depth, normals, and instance segmentation masks) are generated automatically.

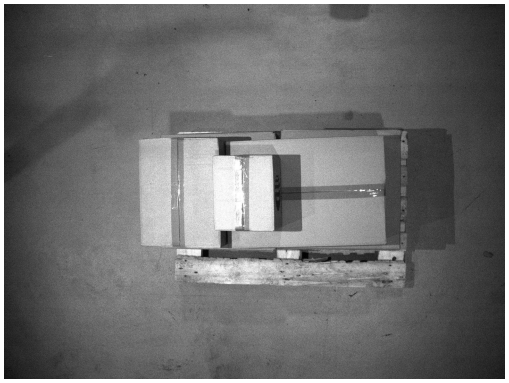
Annotated Real Dataset (Target Domain): The dataset is a representation of our target domain and thus reflects the real environment for the operations. The collection consists of 1,700 pictures of the real world taken in an industrial warehouse. Moreover, these pictures were corrected and carefully annotated through a hybrid Segment Anything Model (SAM) and Human-in-the-Loop process, as described in the previous chapter.

Here some real image (see Figure 4.1):

Real image 1



Real image 2

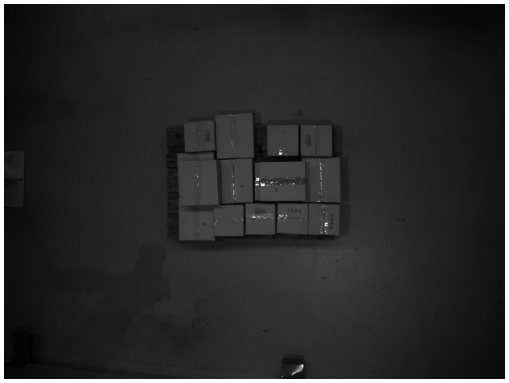


Real image 3

Real image 4

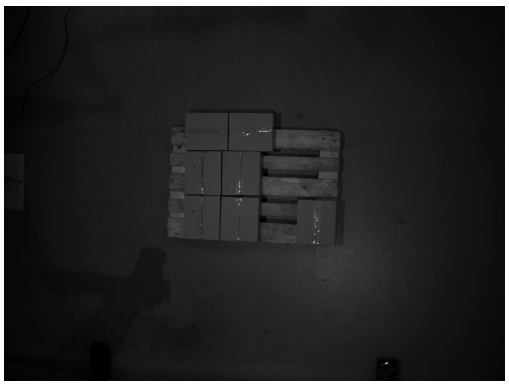
Figure 4.1: Set of real images annotated.

Unannotated Challenge Dataset (External Evaluation): Here we only have unannotated real images depicting the most challenging kinds of arrangements of objects, i.e., tightly packed boxes that are strictly aligned. The main goal was to perform a thorough qualitative examination of the model's ability for zero-shot generalization and the capability to accurately differentiate instances in tightly packed, out-of-distribution scenarios, which were not explicitly present in the training data. This collection made it possible to visually examine model robustness in an especially demanding operational scenario where the maximum level of segmentation ambiguities is reached. Here some images (see Figure 4.2) :



Real image not annotated 1

Real image not annotated 2



Real image not annotated 3

Real image not annotated 4

Figure 4.2: Set of real images not annotated.

4.1.2 Data Splits

The annotated real dataset was divided in a way to allow for a thorough and unbiased evaluation of model performance:

- **Training Set:** 70% of the real annotated data.
- **Validation Set:** 20% of the real annotated data.
- **Test Set:** 10% of the real annotated data.

The artificial data was solely intended to serve as the source domain for training purposes.

In order to keep the experiments fair, the splits were made in a stratified manner to ensure that the subsets are representative of the total data distribution variability. It is also of great importance that the Validation and Test Sets consist solely of

images from the Real Dataset; hence, they provide a genuine measurement of the model's performance in the target domain.

4.2 Training configurations

In the instance segmentation experiments, the small variant of the Mask2Former architecture, a cutting-edge Transformer-based model for complex segmentation tasks, was applied. The major configurations and hyperparameters used are enumerated in Table 4.1 .

Table 4.1: Hyperparameters and Configurations for Domain Adaptation Training.

Parameter	**Value / Description**
Base Architecture	Mask2Former (Small)
Backbone	Swin Transformer-Small, pre-trained on ImageNet-22K and fine-tuned on the COCO dataset for Instance Segmentation.
Optimizer	AdamW (Adam with Weight Decay).
Learning Rate (Initial)	5×10^{-5}.
Scheduler	Cosine Annealing.
Weight Decay	10^{-2}.
Batch Size (Per GPU)	4 total images per training step (2 from Source domain + 2 from Target domain).
Training Duration	50 Epochs.
DA Technique	Domain-Adversarial Neural Network (DANN)
Adversarial trade-off (λ_{ent})	gradually increased from 0 to 1.
Data Augmentation	Applied to both domains (Source and Target), including standard techniques such as random flipping, scaling variations, and cropping.

4.3 Experimental Methodology and Setups

In order to effectively assess the efficiency of various integration techniques and how much the model relies on the amount of real data, Baseline 1 (Real-Only), and Experimental Setups 1 and 2 with a scaled budget of real images were carried out. This method allows to measure the added value of synthetic data when the availability of target data is very limited.

Hence, their comparative results were based on the following subsets of annotated real data, which were used for training or fine-tuning:

- Full Real Dataset
- 250 images
- 50 images
- 30 images
- 20 images
- 10 images

For this purpose, the setups that were compared are described in the following subsections:

4.3.1 Baseline 1: Real-Only

The model was developed only by an annotated real dataset. This trial sets the benchmark performance that can be achieved with a limited amount of high-quality real data and acts as a reference for all other configurations.

4.3.2 Baseline 2: Synthetic-Only (Zero-Shot Sim-to-Real)

The model was built only by using the synthetic dataset and later was tested directly on the real test set. The main target of the setting here is to measure the inherent "sim-to-real gap" and to assess the model's ability to generalize zero-shot from the simulated to the real domain.

4.3.3 Experimental Setup 1: Mixed-Data Training

The model used the combined dataset (real + synthetic) from the very beginning to learn. This approach i.e. "direct mixing" allows the model to be exposed to both the domains during the learning process.

4.3.4 Experimental Setup 2: Synthetic Pre-training and Real Fine-tuning

This is how we adapt our main domain strategy. The model was pre-trained on a the synthetic dataset to learn features that are both general and robust, and then it was further trained (fine-tuned) on a small but high-quality real dataset in order to specialize in the target domain's features.

4.3.5 Experimental Setup 3: Semi-Supervised Learning (SSL)

In order to prove the effectiveness of utilizing unlabeled data and to expand the range of potential applications, the model was trained using also a semi-supervised methodology.

The training dataset was a composite one and it consisted of:

- A portion of the real image set (with labels).
- The rest of the real images (without labels).
- The entire synthetic dataset (with labels).

The goal of this arrangement was to gauge how far the addition of unlabeled data (both real and synthetic) would lead to the performance being better than in the case of pure supervised training.

4.3.6 Experimental Setup 4: Domain Adaptation and Comparison with Baseline

The aim of this experiment was to compare the results of the training that was done only on the synthetic data with those of the domain adaptation techniques.

Phase 1: Synthetic Baseline (Unsupervised)

The model was initially developed only with the use of the synthetic data set (without any real images see The Baseline 2 used at 4.3.2) to figure out the performance baseline when the target data is missing.

Phase 2: Unsupervised Domain Adaptation (UDA)

At the second stage, an Unsupervised Domain Adaptation (UDA) plan was carried out. Training was done with:

- All synthetic images (as the source domain with labels).
- All real images (as the target domain without labels).

The goal of this comparison was to find out if the adoption of the Domain Adaptation technique would result in a substantial increase in the performance of the real domain compared to the model that was only trained on the synthetic set.

4.4 Evaluation metrics

The applications of machine learning models have been assessed by typical metrics of the COCO benchmark. A special emphasis has been made on the mean Average Precision at an IoU threshold of 0.5 (mAP_{50}) that served as the main comparison metric.

4.4.1 Mean Average Precision (mAP)

Mean Average Precision (mAP) is an aggregated measure of the model's capability to not only detect the objects correctly (classification precision) but also locate them accurately (spatial precision).

First of all, for each class and for different Intersection over Union (IoU) thresholds, the Average Precision (AP) is defined as the area under the precision-recall curve. After this, the mAP stands for the average of the APs over all the classes taken into account.

The main variants are:

- **mAP**: The mean of the APs is computed for IoU thresholds ranging from 0.5 to 0.95 (step 0.05), according to the COCO protocol.
- **mAP_{50}**: AP is calculated with a fixed IoU threshold of 0.5, meaning that a prediction is considered correct if the predicted mask and the ground truth mask have an overlap of at least 50%.
- **mAP_{75}**: AP is calculated with a fixed IoU threshold of 0.75, which is much more strict than the previous one.

In simpler terms, for a given IoU threshold (e.g., 0.5):

$$\text{AP} = \int_0^1 p(r)\, dr \tag{4.1}$$

where $p(r)$ is the precision as a function of the recall (r). mAP_{50} is thus the average of the APs calculated at an IoU threshold of 0.5 for all classes.

4.5 Hardware and software environment

The hardware and software platform used have a significant impact on the reproducibility and efficiency of Machine Learning experiments. To ensure consistency, all the training, validation, and synthetic data generation steps were performed on the technical configuration specified hereafter.

4.5.1 Hardware

Component	**Specifications**
GPU	RTX A1000
CPU	Intel Core i7-13700H
RAM	32 GB DDR5

4.5.2 Software and Development Tools

Component	**Description**
Operating System	Windows 11
Package Management	Conda
Simulation Environment	NVIDIA Isaac Sim 4.5
Control Interface	CustomTkinter (GUI)
Annotation Application	React and Flask (Web App)

Chapter 5

Experimental Results

This chapter describes and investigates the results achieved from the live experiments, which were planned to check the performance of synthetic data and domain adaptation methods for the training of instance segmentation models. The main metrics that has been used for the assessment is Mean Average Precision.

5.1 Quantitative evaluation

The quantitative assessment was carried out by analyzing the performance of Mask2Former on the real validation set; four major training strategies have been used for the experiments, as detailed in Table 5.1. The study mainly discusses how artificially generated data influenced the results in the presence of a large amount of real data and, most importantly, in the case of a small number of real data.

The numbers in brackets indicate the change in absolute values with respect to the “Real Only” baseline training.

Detailed Analysis of Results

1. **Comparison with Simple Data Mixing** The column "Real + Synthetic (Simple Mix)" points out the fundamental challenge of the sim-to-real gap. It is clearly shown in the Table 5.1 that mixing synthetic data with real data without any alignment techniques leads to a very significant drop of the performance in the data-scarce scenarios (e.g. -0.026 mAP@50 for 10 real images) missions.

   The Negative Impact of Domain Shift: This decrease points to the fact that the distributional mismatch (the domain shift) between simulation and reality causes the model to pick up patterns that are peculiar to the synthetic

domain and that do not generalize well to the real one, thus the gain with the additional volume of data is annulled.

2. **Effectiveness of Domain Adaptation (DA)** The evidence is conclusive that the Domain Adaptation (**DANN**) approach is the most effective method for the incorporation of synthetic data.

   - **Full Real Dataset (1200 Images):** Hybrid training with DA reached a mAP@50 of 0.924 which is 1.5 percentage points higher than the real-only baseline (0.909). This is the point where synthetic data, if the distributions are matched, not only fills in the gaps but also increases the model's robustness and generalization capability even in case of having a large amount of labeled real data.
   - **Low-Data Regime (50 Real Images):** In this very important case, the benefit of Domain Adaptation is dramatically increased, showing an increase of +0.026 points (from 0.762 to 0.788), which corresponds to a relative change of 3.4%. The reduction in overfitting to the few real samples through feature alignment makes it possible to completely exploit the synthetic domain's diversity.

3. **Zero-Shot Sim-to-Real and Fine-Tuning** The unsupervised training (Zero-Shot Sim-to-Real), which involved synthetic data only without real labels, achieved a mAP@50 of around 0.578. This is a performance that can hardly be considered for industrial deployment, but it is still quite significant and shows a good transfer learning capability, thanks to the **Domain Randomization** used in Isaac Sim.

   The Fine-Tuning strategy (first training on the entire synthetic set, then fine-tuning on real data) was able to achieve great results, especially in the low-data regime.

   50 Real Images Case: Fine-Tuning (0.789) and DA (0.788) were able to achieve almost the same and very competitive performance levels, which implies that strong pre-training on synthetic data could be considered as a viable and effective alternative to DANN-based training for limited datasets.

4. **Semi-Supervised Evaluation** The Semi-Supervised method (labels for 50 real images and the rest of the real data unlabeled) scored the highest absolute number of the low-data regime (0.794). This finding confirms that the most promising way to maximize performance when labeling is costly and limited is to make the most efficient use of unlabeled real data, at the same time taking advantage of the structural knowledge derived from the synthetic domain.

Table 5.1: Experimental Results (mAP@50) for Instance Segmentation

Real Training Set	**Real Only**	**Real+Synth (Simple Mix)**	**Domain Adapt. (DANN)**	**Fine-Tuning (from Synth.)**	**Semi-Supervised**
Full Real (1200)	0.909	0.907 (-0.002)	**0.924** (+0.015)	0.913 (+0.004)	—
250 Real	0.849	0.850 (+0.001)	**0.854** (+0.005)	—	—
50 Real	0.762	0.764 (+0.002)	0.788 (+0.026)	0.789 (+0.027)	**0.794** (+0.032)
30 Real	0.758	0.747 (-0.011)	**0.783** (+0.025)	—	—
20 Real	0.734	0.711 (-0.023)	**0.740** (+0.006)	—	—
10 Real	0.698	0.672 (-0.026)	**0.707** (+0.009)	—	—
Unsupervised (0)	—	**0.578**	0.576	—	—

5.2 Qualitative evaluation

5.2.1 Inference results

Our qualitative analysis confirms the model's ability to segment objects with high precision even in complex and dense depalletization scenarios. The model demonstrates robustness to lighting variations, including strong contrasts between illuminated and shadowed areas, as well as diverse textures and surface patterns. Segmentation quality remains stable even when additional background elements (such as the workbench or surrounding structures) introduce additional geometric complexity. Furthermore, the model preserves accuracy across different object densities, from scenes with only a few boxes to highly cluttered environments where occlusions are frequent. Very fine edges or extremely small details may occasionally be missed. This is expected due to their minimal physical footprint and the use of the "small" variant of the segmentation architecture. Below are representative inference examples (Figures 5.1–5.8).

Figure 5.1: Inference 1

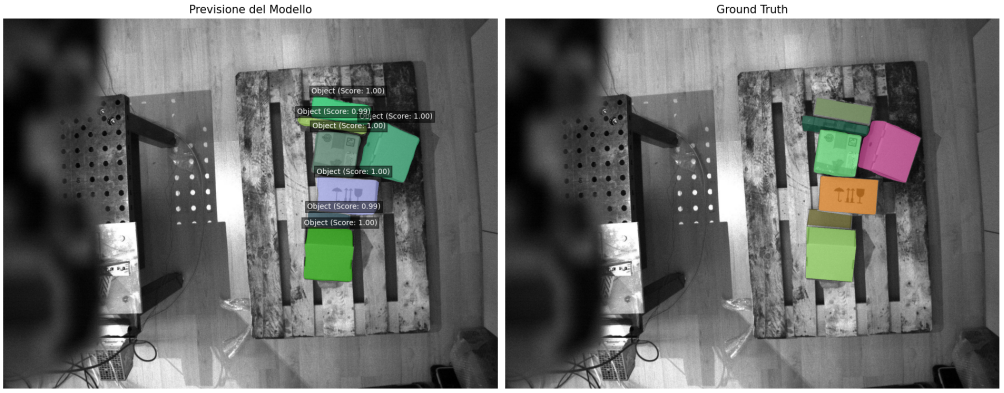


Figure 5.2: Inference 2

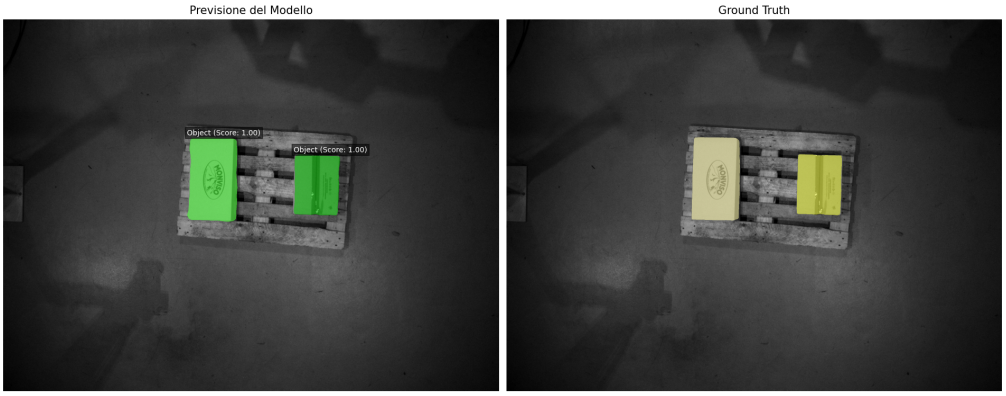


Figure 5.3: Inference 3

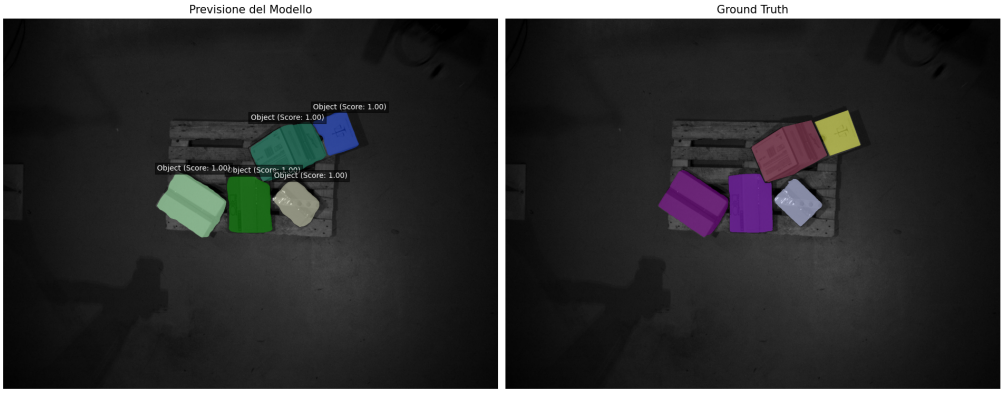


Figure 5.4: Inference 4

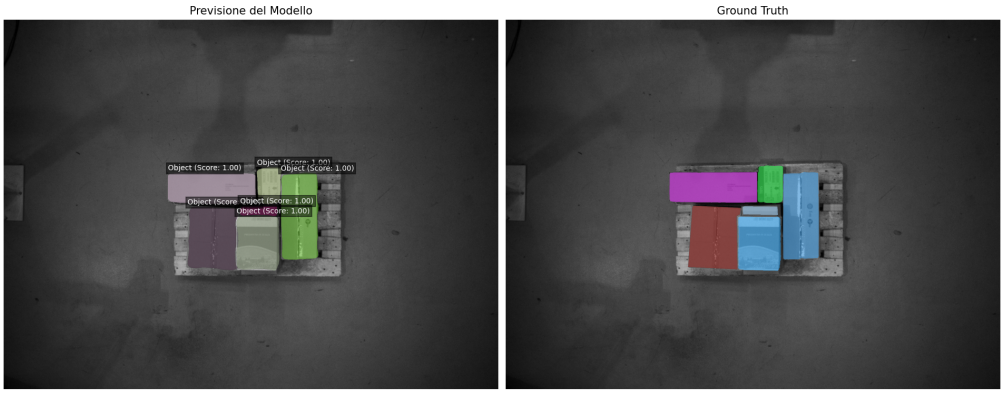


Figure 5.5: Inference 5

Figure 5.6: Inference 6

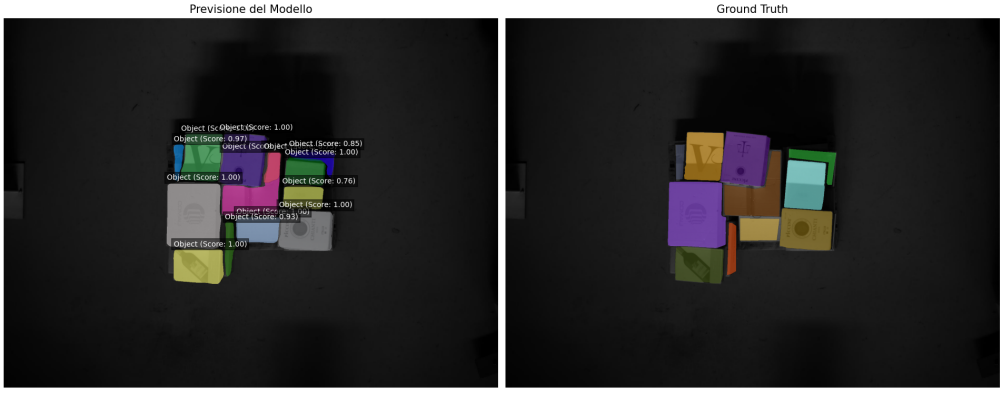


Figure 5.7: Inference 7

Figure 5.8: Inference 8

5.3 Analysis of model performance across datasets

5.3.1 Challenge Dataset evaluation

During qualitative evaluation on the Unannotated Challenge Dataset, the model was able to generalize a remarkable level of complexity in unseen object arrangements. Although the boxes were tightly packed and perfectly aligned—conditions that were not part of the training data—the model managed to correctly identify and segment most instances. This is indicative of high transfer ability and robustness of the learned features (see Figure 5.9). Nevertheless, the model in some instances merged the adjacent boxes into one, particularly when the edges of the boxes were barely distinguishable. This slight over-segmentation, which is in line with our expectations, is due to the absence of examples of such strictly aligned and densely arranged configurations in the training set (see Figure 5.10) . As a result of qualitative analysis, the model is confirmed to be able to sustain detection performance at a high level even when faced with out-of-distribution scenarios, thereby revealing not only its capabilities but also the limitations of its generalization power.

Figure 5.9: Correct inference on Challenge Dataset

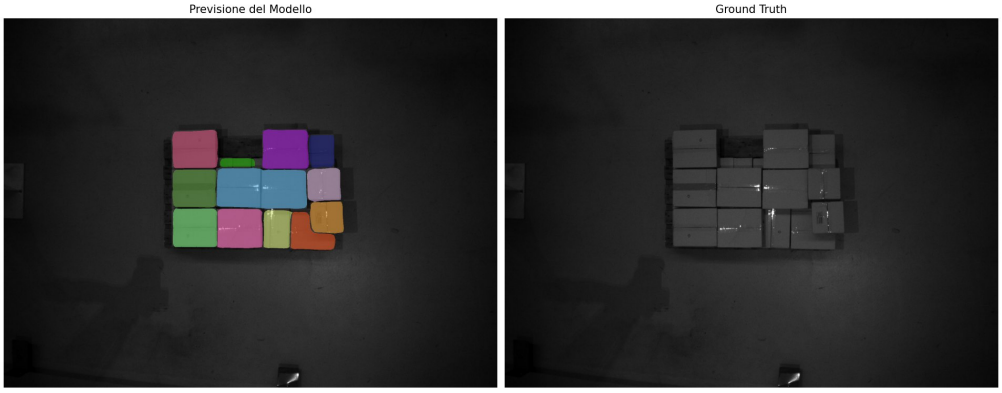


Figure 5.10: Merge inference on Challenge Dataset

Chapter 6

Discussion

6.1 Interpretation of Results and Role of Synthetic Data

The experimental results clearly show the effectiveness of synthetic data and domain adaptation techniques in improving the generalization ability of instance segmentation models for robot perception. In particular, the Domain-Adversarial Neural Network (DANN) consistently increased the mean Average Precision (mAP) across both low-data and high-data regimes. The enhancement in value was especially notable in limited annotated data settings, thus synthetic data becomes a very viable solution to the problem of scarce real-world samples, provided that it is properly aligned with the target domain. On the other hand, the models have exhibited a qualitatively strong segmentation even in the cases of out-of-distribution scenarios, e.g., the Challenge Dataset, which contains tightly aligned and densely packed boxes. The inference quality remained solid, although some cases of over-segmentation and merging between adjacent instances were observed. These results were expected, as the model was not explicitly trained on configurations exhibiting such extreme regularity. On the contrary, the model enhanced its capability to trace the object boundaries and keep the semantic coherence under the changing of the light and texture, which means the representations learned are deep and can be transferred to another domain. Moreover, the experiments emphasize the pivotal insight that simply blending synthetic and real data without domain alignment will lead to a drop in performance. The result here is consistent with the theory of the sim-to-real gap, which refers to the differences in the visual and statistical properties of simulated and real-world images. In this respect, domain adaptation is the only way to feature invariant representations and generalization that is stable across different domains. Beyond numerical performance, synthetic data was the main ingredient that made the scalability and efficiency aspects of the work possible besides just

the numbers. The NVIDIA Isaac Sim pipeline developed in this work enabled the automatic generation of photo-realistic and physically consistent datasets enriched with domain randomization, which introduced controlled variability in lighting, materials, and object configuration. The range of variations introduced through this randomization not only increased the robustness of the model, but also allowed for fewer manual annotation processes, which are both expensive and time-consuming. When synthetic data is used together with semi-automatic labeling tools, like the Segment Anything Model (SAM) and human-in-the-loop correction mechanisms, the result is a quicker and more sustainable dataset creation workflow that is also appropriate for industrial settings. Moreover, the experimental comparison serves as an evidence that hybrid training strategies, i.e. the use of synthetic and real data through fine-tuning or domain adaptation, are the most advantageous in terms of data efficiency and performance. These methods were able to achieve a significantly better performance than the real-data baselines that were only used in low-data regimes, thus demonstrating that synthetic data can be more than just a substitute, but rather a complementary and stabilizing component in model training.

6.2 Limitations and validity threats

Several limitations, however, have to be taken into account along with the promising results. First, the synthetic scenes produced were photorealistic, but they still lacked the ability to fully replicate the physical and optical nuances of the real world—such as subtle lighting diffusion, lens distortions, or sensor noise. These differences could be the reasons for the remaining performance gap between models trained with synthetic and real data. Secondly, it took quite a long time to create synthetic datasets. Although the pipeline offered automation and reproducibility, the rendering of physically consistent scenes and the simulation of grasping interactions were very demanding in terms of computational resources. Therefore, a limited number of synthetic images were generated and used for training, which means that the dataset is smaller than standard large-scale computer vision benchmarks. This limitation may have led to a reduction in the variability and statistical completeness of the synthetic domain. Thirdly, the limitation of the depalletization experiments with rigid box-like objects only, in terms of the scope, the findings' generalizability is restricted. It will be possible that extending the method to different industrial tasks, such as handling deformable objects or detecting transparent materials, will require further changes and validation. Fourthly, this evaluation mainly depended on mAP and qualitative analysis. Although these figures are proper in measuring segmentation accuracy, there is a chance that they ignore those aspects which are very important to a robot and have to do with its performance like the robustness

to partial occlusions, consistency over time, or computational latency during a real-time operation. The subsequent research should really take these additional factors into account in order to have a complete picture of the system's capabilities in the field. Finally, the domain adaptation method employed—DANN—represents only one possible approach. Alternative or complementary techniques, such as adversarial image translation (e.g., CycleGAN-based domain stylization), self-supervised adaptation, or multi-modal feature alignment, could further narrow the sim-to-real gap. Nevertheless, the stability and robustness of the results obtained confirm the practical effectiveness and methodological soundness of the proposed approach.

Another limitation encountered during the development process concerns the parallel gripper simulation. The Robotiq 2F-85 gripper asset is loaded directly from the Omniverse cloud; a recent update modified the internal structure of the asset (link hierarchy, joint naming, and physical parameters), making the previously implemented control logic and state machine incompatible. As a result, the parallel gripper simulation is currently non-functional unless the code is updated accordingly. This highlights a critical issue related to the dependency on cloud-distributed assets and the importance of maintaining stable local versions to ensure reproducibility and continuity of the system.

Chapter 7

Conclusions and Future Work

7.1 Summary of Contributions

The main objective of the thesis was to investigate the utility of synthetic data and domain adaptation methods to improve the perception capabilities of industrial robotic systems, with a focus on depalletization tasks. Through the integration of NVIDIA Isaac Sim, modern deep learning architectures such as Mask2Former, and advanced annotation methods including the Segment Anything Model (SAM), a complete framework for data generation, training, and evaluation was designed and implemented.

The main contributions are as follows:

1. **Synthetic Data Generation Pipeline:** A modular and extensible pipeline was developed within NVIDIA Isaac Sim to automatically generate photo-realistic synthetic datasets with high variability in texture, lighting, object geometry, and arrangement. The method combines physically based rendering and physics simulations to ensure realistic and diverse data.

2. **Robotic Grasp Simulation:** The simulation of two different gripper types: vacuum and parallel-jaw, was performed to generate successful and failed grasp samples, providing rich information useful for perception and manipulation learning.

3. **Real Dataset Enhancement and Annotation:** A semi-automatic annotation workflow was implemented using SAM for box-to-mask conversion, complemented by a custom human-in-the-loop correction web interface, significantly reducing annotation time while maintaining label quality.

4. **Hybrid Training and Domain Adaptation:** Various experimental setups were tested, mixing real and synthetic data under different training paradigms. The integration of Domain-Adversarial Neural Network (DANN) adaptation demonstrated measurable improvements in segmentation accuracy, particularly in low-data regimes.

5. **Comprehensive Evaluation:** Quantitative results, expressed in terms of mean Average Precision (mAP), and qualitative assessments across multiple datasets confirmed the robustness of the proposed approach and its ability to generalize to unseen configurations.

Overall, this work provides a concrete demonstration of how simulation-based data generation, combined with advanced deep learning techniques, can substantially reduce data acquisition costs, while maintaining competitive performance in real-world robotic vision tasks.

7.2 Main Findings

The experiments conducted reflect the following key points:

- **Effectiveness of Synthetic Data:** Synthetic datasets can lead to model generalization and stability if they are made with enough variability and realism, even when only a few real samples are available.

- **Necessity of Domain Adaptation:** Directly mixing real and synthetic data without alignment causes performance to decrease due to the sim-to-real gap. Domain adaptation, as implemented through DANN, successfully mitigates this issue and allows models to learn domain-invariant features.

- **Feasibility of a Hybrid Workflow:** Combining synthetic data with real-world annotations and semi-automated labeling methods offers an efficient, scalable strategy for training high-performance perception systems suitable for industrial applications.

- **Strong Zero-Shot Transfer Capabilities:** Although zero-shot sim-to-real performance remains below supervised baselines, it nonetheless demonstrates promising transfer potential—an encouraging step toward fully simulation-trained perception systems.

- **Industrial Applicability:** The entire workflow—spanning synthetic data generation, model training, and deployment—was conceived with real manufacturing constraints in mind, providing a replicable methodology for similar industrial contexts.

7.3 Future Directions

While the presented work achieves promising results, several issues remain open for future exploration and enhancement:

1. **Expansion of the Synthetic Dataset:** Increasing the scale and diversity of the synthetic dataset will improve statistical coverage and domain variability. Future pipelines could leverage distributed rendering or cloud-based GPU clusters to accelerate data generation.
2. **Parallelization of Image Generation:** The current dataset generation process is computationally intensive and sequential. Future developments could introduce multi-threaded or distributed processing to parallelize scene rendering, drastically reducing generation time and increasing throughput.
3. **Improved Domain Bridging Techniques:** Incorporating more advanced domain adaptation methods—such as image translation via CycleGAN, contrastive domain alignment, or self-supervised feature learning—could further reduce the visual and statistical discrepancies between synthetic and real domains.
4. **Integration with Real-Time Robotic Control:** A natural continuation involves connecting the perception module directly with robotic control loops, assessing latency, inference stability, and adaptability under online feedback conditions.
5. **Enhanced Evaluation Metrics:** Beyond mAP, future work should incorporate task-specific metrics, such as grasp success rate, inference latency, and temporal consistency, providing a more comprehensive view of the system's operational effectiveness.
6. **Introduction and Segmentation of Object Defects:** A potential extension involves augmenting the synthetic data with realistic defects (e.g., scratches, dents, deformations, or contamination) and training the model to detect and segment such anomalies. This addition would greatly expand the practical utility of the perception system in quality control and inspection applications.

In conclusion, this thesis demonstrates that the combination of synthetic data, domain adaptation, and modern segmentation architectures represents a powerful and pragmatic framework for advancing industrial robotic perception. By bridging simulation and reality, the proposed methodology provides both a scientific and practical contribution toward scalable, adaptive, and cost-efficient AI-driven automation systems.

Bibliography

[1] *NVIDIA Isaac Sim*. https://developer.nvidia.com/isaac-sim. 2021 (cit. on p. 5).

[2] Alexander Kirillov et al. «Segment Anything». In: *arXiv preprint arXiv:2304.02643* (2023). URL: https://segment-anything.com/ (cit. on pp. 5, 9).

[3] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. «Mask2Former: Masked-Attention Mask Transformer for Universal Image Segmentation». In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 1290–1299. URL: https://github.com/facebookresearch/Mask2Former (cit. on pp. 5, 8).

[4] John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J. Davison. «SceneNet RGB-D: 5M Photorealistic Images of Synthetic Indoor Trajectories with Ground Truth». In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2678–2687. URL: https://arxiv.org/abs/1612.05079 (cit. on p. 5).

[5] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. «Falling Things: A Synthetic Dataset for 3D Object Detection and Pose Estimation». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2018, pp. 2038–2041. URL: https://arxiv.org/abs/1804.06534 (cit. on p. 5).

[6] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. «Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World». In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 23–30. URL: https://arxiv.org/abs/1703.06907 (cit. on p. 5).

[7] *NVIDIA Omniverse Documentation*. https://docs.omniverse.nvidia.com/ (cit. on p. 6).

[8] NVIDIA Corporation. *Perception Data Generation (Replicator) — Isaac Sim Documentation*. Accessed: 2025-10-11. 2025. URL: https://docs.isaacsim.omniverse.nvidia.com/latest/replicator_tutorials/index.html (cit. on p. 6).

[9] NVIDIA Corporation. *PhysicalAI Robotics Manipulation (Synthetic) Dataset*. Generated in Isaac Lab/Isaac Sim; Accessed: 2025-10-11. 2025. URL: https://huggingface.co/datasets/nvidia/PhysicalAI-Robotics-Manipulation-Augmented (cit. on p. 6).

[10] Yaroslav Ganin and Victor Lempitsky. «Unsupervised Domain Adaptation by Backpropagation». In: *arXiv preprint arXiv:1409.7495* (2015). URL: https://arxiv.org/abs/1409.7495 (cit. on p. 7).

[11] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. «Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks». In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2017. URL: https://junyanz.github.io/CycleGAN/ (cit. on p. 7).