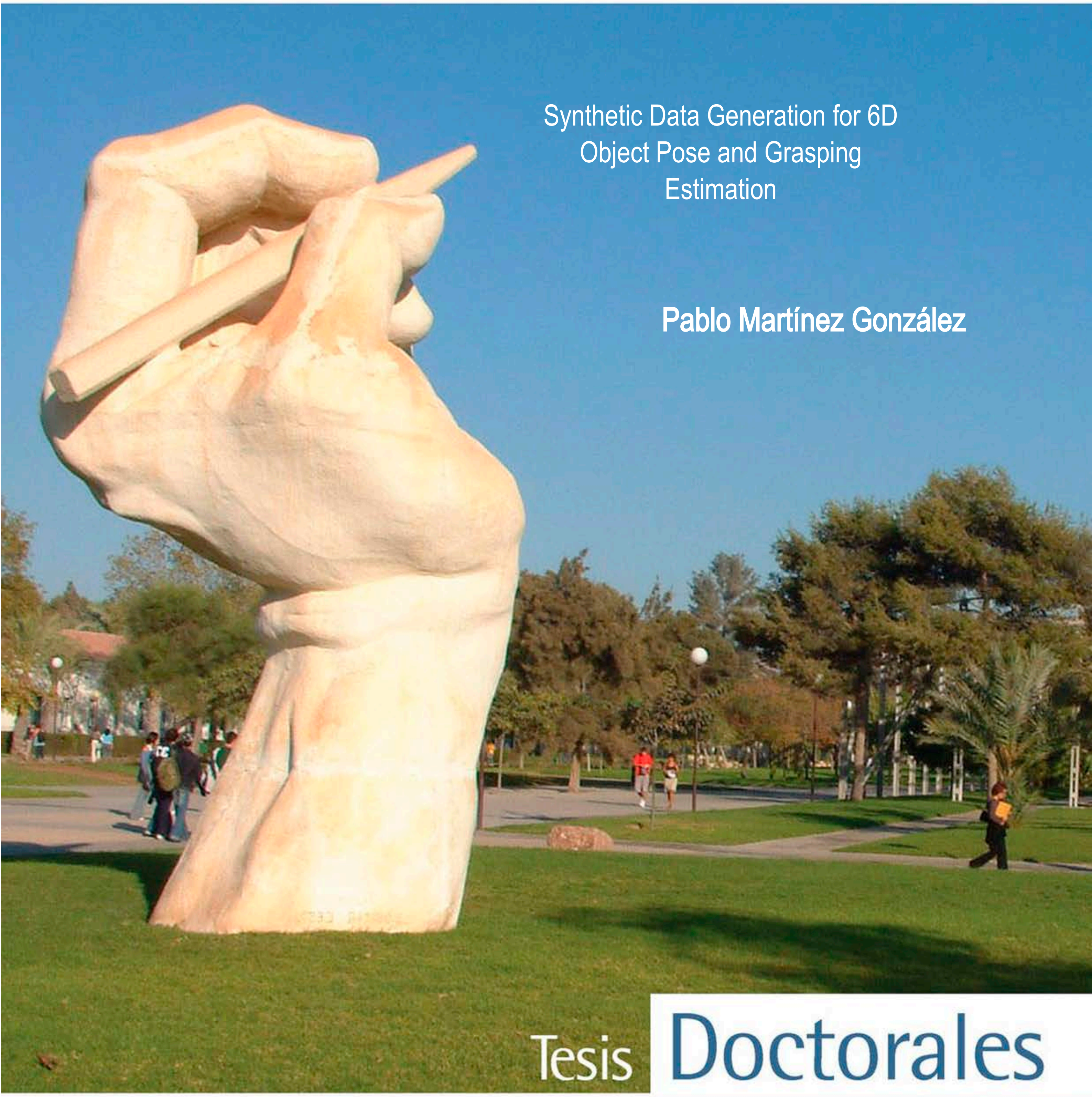


Synthetic Data Generation for 6D Object Pose and Grasping Estimation

Pablo Martínez González

Universitat d'Alacant
Universidad de Alicante

DEPARTAMENTO DE TECNOLOGÍA INFORMÁTICA Y COMPUTACIÓN

ESCUELA POLITÉCNICA SUPERIOR

Synthetic Data Generation for 6D Object Pose and Grasping Estimation

Pablo MARTÍNEZ GONZÁLEZ

Tesis presentada para aspirar al grado de

DOCTOR POR LA UNIVERSIDAD DE ALICANTE

MENCIÓN DE DOCTOR INTERNACIONAL

DOCTORADO EN INFORMÁTICA

Dirigida por

Dr. José GARCÍA RODRÍGUEZ

Dr. Sergio ORTS ESCOLANO

La realización de esta tesis ha sido financiada por el Ministerio de Educación y Formación Profesional [FPU17/00166]

Universitat d'Alacant
Universidad de Alicante

DEPARTAMENT OF COMPUTER TECHNOLOGY

HIGHER POLYTECHNIC SCHOOL

Synthetic Data Generation for 6D Object Pose and Grasping Estimation

Pablo MARTÍNEZ GONZÁLEZ

A thesis submitted in fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY BY THE UNIVERSITY OF ALICANTE

INTERNATIONAL MENTION

COMPUTER SCIENCE

Advised by

Dr. José GARCÍA RODRÍGUEZ

Dr. Sergio ORTS ESCOLANO

This thesis has been funded by the Spanish Ministry of Education [FPU17/00166]

This thesis is dedicated

To all the people who has been with me through this process at the university. In the bachelor degree: Javier, Alex, Manu, Brayan, Cayetano and Ginés. In the masters degree: Rosa and Héctor. And specially to those in the PhD program: Albert, John, Fran, Alejandro, Sid, Kooshan, Elena, Marcelo, Álvaro, Manuel, David and Joan. Also in Vienna: Negar, Matthias, Stefan and Elena. And of course, to Sergiu and Victor. Also, there is a special place to Andrea, for being there whenever I need you.

To all my teachers and references that I had since I started to study, and specially to my advisors during this period: Jose García and Sergio Orts, also Markus Vincze in Vienna.

And finally, to my family: my aunt Inma, my uncle Javi, my grandparents, my brother Victor, my Dad, and specially to my Mom.

Pablo Martínez González
February 17, 2023
Alicante, Spain.

Contents

List of Figures

List of Tables

Glossary

AAL	Ambient Assisted Living
ADR	Active Domain Randomization
API	Application Program Interface
AUC	Area Under the Curve
BVH	Bounding Volume Hierarchy
CHALET	Cornell House Agent Learning Environment
CNN	Convolutional Neural Network
DLSS	Deep Learning Super Sampling
DR	Domain Randomization
EPnP	Efficient PnP
FoV	Field of View
FPS	frames per second
GAN	Generative Adversarial Network
GPU	Graphics Processor Unit
HoME	Household Multimodal Environment
HUD	Head-Up Display
ICP	Iterative Closest Point
KDF	Key Distance Field
MINOS	Multimodal Indoor Simulator
MLP	Multi-layer Peceptron
MSAA	Multi-Sample Anti-Aliasing
NDDS	NVIDIA's Deep Learning Dataset Synthesizer
PBR	Physically Based Rendering
PnP	Perspective-n-Point
RANSAC	Random Sample Consensus
RGB	Red, Green and Blue
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SDR	Structured Domain Randomization
SVPG	Stein Variational Policy Gradient
THOR	THe House of inteRactions
UE4	Unreal Engine 4
URDF	Unified Robot Description File
VKITTI	Virtual KITTI
VR	Virtual Reality

Abstract

Teaching a robot how to behave so it becomes completely autonomous is not a simple task. When robotic systems become truly intelligent, interactions with them will feel natural and easy, but nothing could be further from truth. Make a robot understand its surroundings is a huge task that the computer vision field tries to address, and deep learning techniques are bringing us closer. But at the cost of the data.

Synthetic data generation is the process of generating artificial data that is used to train machine learning models. This data is generated using computer algorithms and simulations, and is designed to resemble real-world data as closely as possible. The use of synthetic data has become increasingly popular in recent years, particularly in the field of deep learning, due to the shortage of high-quality annotated real-world data and the high cost of collecting it. For that reason, in this thesis we are addressing the task of facilitating the generation of synthetic data with the creation of a framework which leverages advances in modern rendering engines.

In this context, the generated synthetic data can be used to train models for tasks such as 6D object pose estimation or grasp estimation. 6D object pose estimation refers to the problem of determining the position and orientation of an object in 3D space, while grasp estimation involves predicting the position and orientation of a robotic hand or gripper that can be used to pick up and manipulate the object. These are important tasks in robotics and computer vision, as they enable robots to perform complex manipulation and grasping tasks. In this work we propose a way of extracting grasping information from hand-object interactions in virtual reality, so that synthetic data can also boost research in that area. Finally, we use this synthetically generated data to test the proposal of applying 6D object pose estimation architectures to grasping region estimation. This idea is based on both problems sharing several underlying concepts such as object detection and orientation.

Resumen

Enseñar a un robot a ser completamente autónomo no es tarea fácil. Cuando los sistemas robóticos sean realmente inteligentes, las interacciones con ellos parecerán naturales y fáciles, pero nada más lejos de la realidad. Hacer que un robot comprenda y asimile su entorno es una difícil cruzada que el campo de la visión por ordenador intenta abordar, y las técnicas de aprendizaje profundo nos están acercando al objetivo. Pero el precio son los datos.

La generación de datos sintéticos es el proceso de generar datos artificiales que se utilizan para entrenar modelos de aprendizaje automático. Estos datos se generan mediante algoritmos informáticos y simulaciones, y están diseñados para parecerse lo más posible a los datos del mundo real. El uso de datos sintéticos se ha vuelto cada vez más popular en los últimos años, especialmente en el campo del aprendizaje profundo, debido a la escasez de datos reales anotados de alta calidad y al alto coste de su recopilación. Por ello, en esta tesis abordamos la tarea de facilitar la generación de datos sintéticos con la creación de una herramienta que aprovecha los avances de los motores modernos de renderizado.

En este contexto, los datos sintéticos generados pueden utilizarse para entrenar modelos para tareas como la estimación de la pose 6D de objetos o la estimación de agarres. La estimación de la pose 6D de objetos se refiere al problema de determinar la posición y orientación de un objeto en el espacio 3D, mientras que la estimación del agarre implica predecir la posición y orientación de una mano robótica o pinza que pueda utilizarse para coger y manipular el objeto. Se trata de tareas importantes en robótica y visión por computador, ya que permiten a los robots realizar tareas complejas de manipulación y agarre. En este trabajo proponemos una forma de extraer información de agarres a partir de interacciones mano-objeto en realidad virtual, de modo que los datos sintéticos también puedan impulsar la investigación en esa área. Por último, utilizamos estos datos generados sintéticamente para poner a prueba la propuesta de aplicar arquitecturas de estimación de pose 6D de objetos a la estimación de regiones de agarre. Esta propuesta se basa en que ambos problemas comparten varios conceptos subyacentes, como la detección y orientación de objetos.

Chapter

1

Introduction

This first chapter presents the main topic of this thesis. First, the Section 1.1 establishes a framework for the research conducted in this thesis. Then, we define the main proposal and goals in Section 1.2. We enumerate the contributions of this thesis in Section 1.3, and finally, in Section 1.4 we list the published papers as a result of the research conducted in this thesis.

1.1 Motivation

The importance of assistance robotics is likely to increase as demographics evolve and the population ages. As people grow older, they may require assistance with daily tasks such as bathing, dressing, and cooking, which can place a strain on caregivers, including family members and healthcare workers. This demand for assistance is only expected to increase in the coming years, with the aging of the Baby Boom generation and the increasing life expectancy of individuals in many European countries [Eur20] [Fer10].

In this context, assistance robotics has the potential to play a crucial role in providing support to older individuals and enhancing their quality of life. By automating many of the tasks that would otherwise need to be performed by a human caregiver, assistance robots can help to alleviate the burden on care-giving staff and provide older individuals with a greater degree of independence. Additionally, the use of robots in care-giving can provide a higher standard of care and improve safety, as robots can be equipped with sensors and other technology that can help to detect and respond to potential hazards.

The development of this technology is particularly useful for patients with cognitive disorders. Diagnosing this kind of disorders in the elderly is challenging and in over 40% of cases, physicians are unaware of the condition [Cho14]. In the US, more than half of patients with dementia have not received a cognitive evaluation [Kot14]. This can result in missed or delayed diagnosis of dementia [Bra09], which may complicate the treatment of the underlying disease and pose safety concerns for patients. To address these issues, two Spanish national projects, COMBAHO[*] and MoDeAsS[†], aim to provide individualized care, monitoring, and assistance to improve the quality of life and well-being of the elderly, particularly those with cognitive impairment. This aligns with the European Horizon initiatives, which propose an assistance system to support and monitor the elderly. This thesis aims to contribute to the early prediction of behavioural abnormalities in the elderly through the analysis of their interaction with the environment, helping to prevent undesirable outcomes.

The field of computer vision has seen tremendous growth in recent years, thanks in large part to advances in deep learning algorithms. These algorithms have enabled computer vision systems to perform tasks such as object recognition and image classification with impressive accuracy. However, there is still much work to be done in order to make computer vision systems more widely applicable in real-world scenarios, particularly in the field of assistance robotics. The use of deep learning in computer vision has the potential to revolutionize the way that robots interact with their environments. With the ability to accurately recognize and classify objects, robots can be equipped with the knowledge necessary to perform a wide range of tasks, from assisting with household chores to providing support in medical and industrial settings. However, in order for these systems to be effective, they need to be trained on large amounts of data. This is where synthetic data generation comes into play. Synthetic data generation is the process of creating artificial data samples that can be used to train computer vision algorithms. This data can be generated to simulate real-world scenarios and can be used to augment existing data sets, making it possible to train computer vision systems even in the absence of real-world data.

In this PhD thesis, we will explore the application of deep learning to computer vision problems in the context of assistance robotics. We will focus on developing novel techniques for synthetic data generation and using these techniques to train deep learning algorithms for various computer vision tasks. Our goal is to

[*]**COMBAHO: Com**e **Ba**ck **Ho**me System for Enhancing Autonomy of People with Acquired Brain Injury and Dependent to Integrate them Back into Society. This work has been partially funded by the Spanish Government for the COMBAHO project (TIN2016-76515-R grant), supported by the European Regional Development Fund (ERDF).

[†]**MoDeAsS: Mo**nitoring and **De**tection of human behaviours for personalized **Ass**istance and early disease detection. I would like to thank "A way of making Europe" European Regional Development Fund (ERDF) and the Spanish Government (MCIN/AEI/10.13039/501100011033) for supporting this work under the MoDeAsS project (grant PID2019-104818RB-I00).

demonstrate the effectiveness of these techniques in improving the performance of computer vision systems in real-world scenarios and to lay the groundwork for future research in the field. Overall, this research will contribute to the advancement of computer vision and its applications in the field of assistance robotics. By leveraging the power of deep learning and synthetic data generation, we hope to bring us closer to a future where robots are capable of providing valuable support to humans in a wide range of tasks and environments.

1.2 This Thesis

This thesis is structured in two main blocks:

1. **Synthetic Data Generation:** Presents a complete framework for the generation of fully-annotated synthetic data from virtual environments in UE4, with a novel data-acquisition pipeline. Originally, it implied a sequence-only workflow, but we extended the tool with a more flexible one. A lot of different kinds of data can be generated, and it is useful for a wide variety of machine learning models applied to computer vision problems. In this thesis we will treat the problem of grasping region estimation from RGB images, for which we implement a way of obtaining contact maps as textures from virtual interactions with objects in VR. With this, we generate a synthetic contact map dataset.
2. **6D Object Pose and Grasping Point Estimation:** Presents a review on 6D object pose estimation models. It analyses the most relevant deep learning-based architectures, with the objective of modifying some of them to leverage their structures with a different but related problem: the estimation of grasping regions on objects. Three different models are proposed.

In the first part of this thesis, Chapter 2 proposes a complete framework for generating synthetic data from virtual environments. Given how data-hungry deep learning algorithms have become, and how rendering engines have developed to resemble reality, synthetic data generation has been a trend to alleviate the tedious task of labelling huge amounts of image data. This framework seeks to take advantage of the development and community of one of the most widespread video game engines available: Unreal Engine 4. The tool is built on top of UE4, providing all the data acquisition system on the one hand (including a record-playback pipeline that allows to store big amounts of high-quality images with a high frame rate), and all the logic for controlling an agent through VR in real-time, allowing hand-object interactions. An extension is also proposed for enabling the use of such a powerful tool with more ease inside Unreal Engine projects, and with a wider kind of applications. One of these applications is the grasping region estimation on objects, topic which is treated in Chapter 3. In this chapter, a system for generating synthetic contact maps as textures from hand-objects interactions in VR is proposed. It is implemented as a shader in UE4 for generating a texture which represents a heat map of contacts.

This system is used, along with the rest of the framework, for generating a grasping region dataset which is useful for the rest of the thesis.

In the second part of this thesis, Chapter 4 presents a literature review of the datasets and methods in the 6D object pose estimation field. We will focus on methods based on deep learning techniques, since we built our synthetic data generation tools for focusing and training these kind of approaches. This analysis is useful for better understanding the foundations of these models, and being able in Chapter 5 to propose three models for predicting grasping regions on objects, based on 6D object pose estimation architectures. The motivation behind this proposal is the underlying relation between both problems, regarding the detection of position and orientation of objects. We modify three existing architectures for comparing results with state-of-the-art methods for estimating grasping regions.

1.3 Contributions

Each chapter of this thesis supports contributions in multiple fields such as synthetic data generation, interactions in virtual reality, 6D object pose estimation, and grasping region synthetic generation and estimation. These contributions are supported by relevant publications in the literature, which are listed in Section 1.4. In addition, most of the source code implemented in the context of this thesis is publicly available on GitHub. We invite the reader to check the links provided in each chapter.

In a nutshell, the contributions of this thesis are the following:

- We propose UnrealROX, a framework for generating synthetic data from virtual environments. It features a novel pipeline of image sequence generation which allows storing big amounts of high-quality images at a high frame rate, among other type of data. Moreover, UnrealROX includes the logic needed for enabling real-time interactions in VR environments.
- Closely related with previous contribution, we packed up and released UnrealROX, with all its extended features, as an open-source UE4 plug-in, so that it can be used with ease in any research project. It can boost deep learning research through synthetic data generation in a wide variety of fields: Reinforcement Learning, Semantic Segmentation, Relighting, 6D Object Pose Estimation, Video Prediction, among others.
- We present a way of generating synthetic contact maps from hand-object interactions in VR environments, extending the possibility of leveraging synthetic data in the field of grasping region estimation. Moreover, we generate and release a synthetic contact map dataset.
- We propose the idea of leveraging 6D object pose estimation deep-learning architectures for grasping region estimation on objects. We propose three models and test them in order to evaluate their viability.

1.4 Publications

This thesis is the result of continuous effort throughout the last few years. Such efforts have sometimes crystallized in the form of conference and journal publications. A significant part of this thesis consists of extracts from the following co-authored publications.

Chapter 2: A Framework for generating Synthetic Data from Unreal Engine Environments

- [Mar19] **P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, A. Jover-Alvarez, S. Orts-Escolano, and J. Garcia-Rodriguez**. «UnrealROX: An eXtremely Photorealistic Virtual Reality Environment for Robotics Simulations and Synthetic Data Generation». *Virtual Reality*, 2019[‡]

- [Mar21] **P. Martinez-Gonzalez, S. Oprea, J. A. Castro-Vargas, A. Garcia-Garcia, S. Orts-Escolano, J. Garcia-Rodriguez, and M. Vincze**. «UnrealROX+: An Improved Tool for Acquiring Synthetic Data from Virtual 3D Environments». *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021[§]

Chapter 3: Generating Synthetic Hand-object Contact Maps for Grasping Region Prediction

- [Mar22] **P. Martinez-Gonzalez, D. Mulero-Perez, S. Oprea, M. Benavent-Lledo, S. Orts-Escolano, and J. Garcia-Rodriguez**. «Synthetic contact maps to predict grasp regions on objects». *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, 2022[¶]

Chapter 5: Estimating Grasping Regions on Objects based on 6D Pose Estimation Models

- [Mar23] **P. Martinez-Gonzalez, S. Thalhammer, D. Mulero-Perez, S. Oprea, S. Orts-Escolano, and J. Garcia-Rodriguez**. «Estimation of Grasping Regions on Objects leveraging 6D Object Pose Estimation Models». *Yet to be published*, 2023

Additional publications resulting from fruitful collaborations in the context of this thesis

- [Gar18b] **A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez,**

[‡]Code `https://github.com/3dperceptionlab/unrealrox`
[§]Code `https://github.com/3dperceptionlab/unrealrox-plus`
[¶]Code `https://github.com/3dperceptionlab/unrealhandgrasp`

P. Martinez-Gonzalez, and J. Garcia-Rodriguez. «A survey on deep learning techniques for image and video semantic segmentation». *Applied Soft Computing*, Vol. 70, pp. 41–65, Sep. 2018

- [Gar18a] **A. Garcia-Garcia, P. Martinez-Gonzalez, S. Oprea, J. A. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Jover-Alvarez**. «The RobotriX: An eXtremely Photorealistic and Very-Large-Scale Indoor Dataset of Sequences with Robot Trajectories and Interactions». *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6790–6797, IEEE, Oct. 2018
- [Opr19] **S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. A. Castro-Vargas, S. Orts-Escolano, and J. Garcia-Rodriguez**. «A Visually Realistic Grasping System for Object Manipulation and Interaction in Virtual Reality Environments». *Computers and Graphics*, Vol. 83, pp. 77 – 86, 2019
- [Gar19] **A. Garcia-Garcia, P. Martinez-Gonzalez, S. Oprea, J. A. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Jover-Alvarez**. «The RobotriX: A Large-scale Dataset of Embodied Robots in Virtual Reality». *Workshop on 3D Scene Generation*, 2019
- [Cas19] **J. A. Castro-Vargas, A. Garcia-Garcia, S. Oprea, P. Martinez-Gonzalez, and J. Garcia-Rodriguez**. «3D Hand Joints Position Estimation with Graph Convolutional Networks: A GraphHands Baseline». *Advances in Intelligent Systems and Computing*, pp. 551–562, Springer International Publishing, Nov. 2019
- [Opr20] **S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. A. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Argyros**. «A Review on Deep Learning Techniques for Video Prediction». *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020
- [Cas20] **J.-A. Castro-Vargas, A. Garcia-Garcia, S. Oprea, P. Martinez-Gonzalez, and J. Garcia-Rodriguez**. «3D Hand Joints Position Estimation with Graph Convolutional Networks: A GraphHands Baseline». M. F. Silva, J. Luís Lima, L. P. Reis, A. Sanfeliu, and D. Tardioli, Eds., *Robot 2019: Fourth Iberian Robotics Conference*, pp. 551–562, 2020
- [Gar20] **J. Garcia-Rodriguez, F. Gomez-Donoso, S. Oprea, A. Garcia-Garcia, M. Cazorla, S. Orts-Escolano, Z. Bauer, J. Castro-Vargas, F. Escalona, D. Ivorra-Piqueres, P. Martinez-Gonzalez, E. Aguirre, M. Garcia-Silviente, M. Garcia-Perez, J. M. Cañas, F. Martin-Rico, J. Gines, and F. Rivas-Montero**. «COMBAHO: A deep learning system for integrating brain injury patients in society». *Pattern Recognition Letters*, Vol. 137, pp. 80–90, Sep. 2020
- [Opr21] **S. Oprea, G. Karvounas, P. Martinez-Gonzalez, N. Kyriazis, S. Orts-Escolano, I. Oikonomidis, A. Garcia-Garcia, A. Tsoli, J. Garcia-Rodriguez, and A. Argyros**. «H-GAN: the power of GANs in your Hands». *International*

Joint Conference on Neural Networks (IJCNN), pp. 1–8, 2021

- [Cas21] **J. A. Castro-Vargas, P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, S. Orts-Escolano, and J. Garcia-Rodriguez**. «Graph Convolutional Neural Networks-based 3D Hand Pose Estimation over Point Clouds». *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, 2021
- [Ben21] **M. Benavent-Lledó, S. Oprea, J. A. Castro-Vargas, P. Martinez-Gonzalez, and J. Garcia-Rodriguez**. «Interaction Estimation in Egocentric Videos via Simultaneous Hand-Object Recognition». *16th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2021)*, pp. 439–448, Springer International Publishing, Sep. 2021

Part I

Synthetic Data Generation

Chapter

2

A Framework for generating Synthetic Data from Unreal Engine Environments

Data-driven algorithms have surpassed traditional techniques in almost every aspect in robotic vision problems. Such algorithms need vast amounts of quality data to be able to work properly after their training process. Gathering and annotating that sheer amount of data in the real world is a time-consuming and error-prone task. These problems limit scale and quality. Synthetic data generation has become increasingly popular since it is faster to generate and automatic to annotate. However, most of the current datasets and environments lack realism, interactions, and details from the real world. UnrealROX is an environment built over Unreal Engine 4 which aims to reduce that reality gap by leveraging hyperrealistic indoor scenes that are explored by robot agents which also interact with objects in a visually realistic manner in that simulated world. Photorealistic scenes and robots are rendered by Unreal Engine into a virtual reality headset which captures gaze so that a human operator

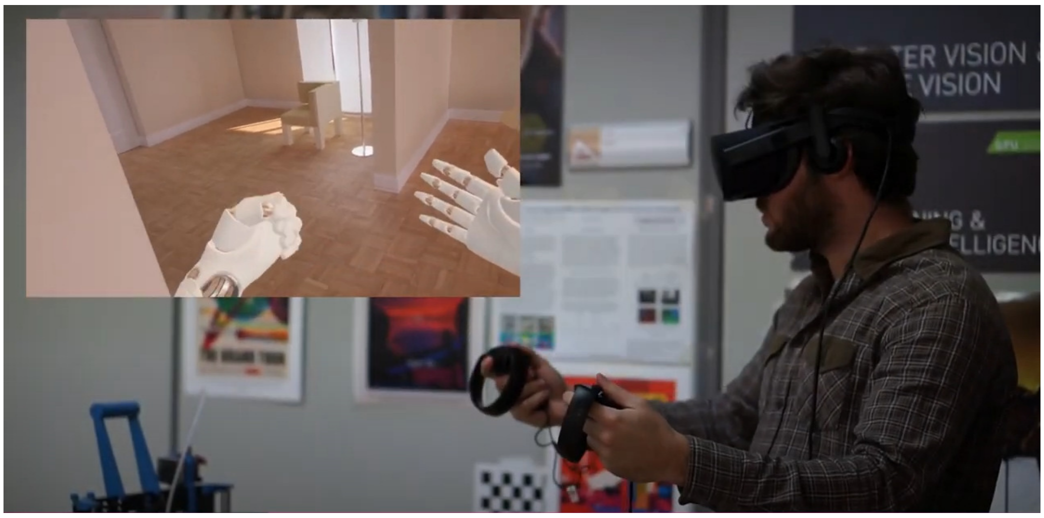


Figure 2.1: A human operator embodied in VR recording a sequence with the framework.

can move the robot and use controllers for the robotic hands; scene information is dumped on a per-frame basis so that it can be reproduced offline to generate raw data and ground truth annotations. This virtual reality environment enables robotic vision researchers to generate realistic and visually plausible data with full ground truth for a wide variety of problems such as class and instance semantic segmentation, object detection, depth estimation, visual grasping, and navigation.

2.1 Introduction

Vision-based robotics tasks have made a huge leap forward mainly due to the development of machine learning techniques (e.g. deep architectures [LeC15] such as CNN or RNN) which are continuously rising the performance bar for various problems such as semantic segmentation [Lon15][He17], depth estimation [Eig14][Umm17], and visual grasping [Len15][Lev18] among others. Those data-driven methods are in need of vast amounts of annotated samples to achieve those exceptional results. Gathering that sheer quantity of images with ground truth is a tedious, expensive, and sometimes nearly impossible task in the real world. On the contrary, synthetic environments streamline the data generation process and are usually able to automatically provide annotations for various tasks. Because of this, simulated environments are becoming increasingly popular and widely used to train those models.

Learning on virtual or simulated worlds allows faster, low-cost, and more scalable data collection. However, synthetic environments face a huge obstacle to be actually useful despite their inherent advantages: models trained in that simulated domain must also be able to perform properly on real-world test scenarios which often feature numerous discrepancies between them and their synthetic counterparts. That set of differences is widely known as the reality gap. In most cases, this gap is big enough so that transferring knowledge from one domain to another is an extremely difficult task, either because renderers are not able to produce images like real-world sensors (due to the implicit noise or the richness of the scene) or either the physical behaviour of the scene elements and sensors is not as accurate as it should be.

In order to address this reality gap, two methods have been proven to be effective: extreme realism and domain randomization. On the one hand, extreme realism refers to the process of making the simulation as similar as the real-world environment in which the robot will be deployed as possible [McC16][Gai16]. That can be achieved through a combination of various techniques, e.g., photorealistic rendering (which implies realistic geometry, textures, lighting and also simulating camera-specific noise, distortion and other parameters) and accurate physics (complex collisions with high-fidelity calculations). On the other hand, domain randomization is a kind of domain adaptation technique that aims for exposing the model to a huge range of simulated environments at training time instead of just to a single synthetic one [Bou17][Tob17c][Tob17a]. By doing that, and if the variability is enough, the model will be able to identify the real world as just another variation thus being able to generalize to it [Tre18a]. **Another remarkable line of research which intertwines both approaches is learning to augment realistic synthetic images with sequences of random transformations [Pas19].**

Purpose: Developing an extremely photorealistic virtual reality environment for generating synthetic data for various robotic vision tasks. In such environment, **we propose a novel way to generate motions and grasps**: a human operator can be embodied, in virtual reality, as a robot agent inside a scene to freely navigate and interact with objects as if it was a real-world robot (Figure 2.1). In addition, releasing the tool as an open-source tool so that any researcher can easily use it for their own purpose.

Our work: Our environment is built on top of UE4 to take advantage of its advanced VR, rendering, and physics capabilities. We developed a tool which provides the following features: (1) a visually plausible grasping system for robot manipulation which is modular enough to be applied to various finger configurations, (2) routines for controlling robotic hands and bodies with commercial VR setups such as Oculus Rift, Oculus Quest and HTC Vive Pro, (3) a sequence recorder component to store all the information about the scene, robot, and cameras while the human operator is embodied as a robot, (4) a sequence playback component to reproduce the previously recorded sequence offline to generate raw data such as RGB, depth, normals, or instance segmentation images, (5) a multi-camera component to ease the camera placement process and enable the user to attach them to specific robot joints and configure their parameters (resolution, noise model, field of view), and (6) open-source code, assets, and tutorials for all those components and other subsystems that tie them together. At last, an extension of the tool is proposed to be able to leverage its main features of acquiring data from virtual environments in UE4 for a wider range of computer vision problems.

Contribution: A whole easy-to-use freely-available open-source framework for generating a wide variety of synthetic data from one of the most widespread graphics engines, Unreal Engine. Moreover, we implement a novel record-playback pipeline which allows: (1) overcome the disk write overhead when storing several images per frame, (2) ensure high quality and high frame-rate sequences by decoupling real-time interaction from final rendering, (3) flexible use of the same recording by generating different data modifying backgrounds, lighting or textures.

This chapter is organized as follows. Section 2.2 analyses already existing environments for synthetic data generation and puts our proposal in context. Next, Section 2.4 describes our proposal and provides in-depth details for each one of its components. After that, we briefly discuss application scenarios for our environment in Section 2.5 **and we also carry out a set of experiments in Section 2.6 to prove the usefulness of our simulator in some of those applications**. At last, in Section 2.7, we draw conclusions about this work and we go over current limitations of our work and propose future works to improve it. After analysing those limitations, we describe in Section 2.8 a set of improvements that we made over the system to make it more flexible and useful for other Deep Learning scenarios, described in Section 2.5. We also perform some experiments with this improved version of the system in Section 2.10, and we draw some final conclusion and limitations in 2.11.

2.2 Related Works

Synthetic environments have been used for a long time to benchmark vision and robotic algorithms [But12], but recently, their importance has been highlighted for training and evaluating machine learning models for robotic vision problems [Bro17] [Ros16] [Mah17]. Due to the increasing need for samples to train such data-driven architectures, there exists an increasing number of synthetic datasets, environments, and simulation platforms to generate data for indoor robotic tasks and evaluate those learned models. In this section, we briefly review the most relevant ones according to the scope of our proposal. We describe both the most important features and main flaws for the following works: CHALET, HoME, AI2-THOR, MINOS, Gibson, VirtualHome, Habitat and ElderSim. In addition, we also describe other related tools such as UnrealCV, Gazebo, **NDDS**, and NVIDIA's Isaac Sim which are not strictly similar but relevant enough to be mentioned. At last, we put our proposal in context taking into account all the analysed strong points and weaknesses.

CHALET (Cornell House Agent Learning Environment) [Yan18] is a 3D house simulator for manipulation and navigation learning. It is built upon Unity 3D so it supports physics and interactions with objects and the scene itself thanks to its built-in physics engine. CHALET features three modes of operation: stand-alone (navigate with keyboard and mouse input), replay (reproduce the trajectory generated on stand-alone mode), and client (use the framework's API to control the agent and obtain information, **making it useful for reinforcement learning tasks**). On the other hand, CHALET presents various weak points such as its lack of realism, the absence of a robot's body or mesh, and the limitation in the number of cameras.

HoME (Household Multimodal Environment) [Bro17] is a multimodal household environment for AI learning from visual, auditive, and physical information within realistic synthetic environments sourced from the SUNCG dataset [Son17]. HoME provides RGB, depth, and semantic maps based on 3D renderings produced by Panda3D. **It also provides some uncommon features such as** acoustic renderings based on EVERT, language descriptions of objects, and physics simulations. It also provides a Python framework compatible with Open AI gym. However, HoME is not anywhere close to photorealism, there is no physical representation of the robot itself, and interactions are discrete.

AI2-THOR (THe House of inteRactions) [Kol17] is a **fully-fledged framework for visual AI research** which consists of near-photorealistic synthetic 3D indoor scenes in which agents can navigate and change the state of actionable objects. It is built over Unity so it also integrates a physics engine which enables modelling complex physical interactions. The framework also provides a Python interface to communicate with the engine through HTTP commands to control the agent and obtain visual information and annotations **(so it is suitable for reinforcement learning approaches)**. Some of the weaknesses of this environment are the lack of a 3D robot model and hands, only a first-person view camera, and the discrete nature of its actions with binary states.

MINOS (Multimodal Indoor Simulator) [Sav17] is a simulator **which stands out of the crowd** for navigation in complex indoor environments. An agent, represented by a cylinder proxy geometry, is able to navigate (in a discrete or continuous way) on scenes sourced from existing synthetic and reconstructed datasets of indoor scenes such as SUNCG and Matterport respectively. Such agent can obtain information from multimodal sensory inputs: RGB, depth, surface normals, contact forces, semantic segmentation, and various egocentric measurements such as velocity and acceleration. The simulator provides both Python and web client APIs to control the agent and set the parameters of the scene. However, this simulator lacks some features such as a fully 3D robot model instead of a geometry proxy, photorealism, configurable cameras and points of view, and interactions with the scene.

Gibson [Xia18a] is a learning environment in which an agent is embodied and made subject to constraints of space and physics (using Bullet physics) and spawned in a virtualized real space (coming from real-world datasets such as Matterport or Stanford 2D-3D). Neural network-based rendering is used to fix artefacts and generate realistic looking images. Another baked-in mechanism is used for transferring to real world, namely Goggles. The approach is extremely useful for online learning of complex navigation and locomotion; however, it lacks dynamic scenes and object interaction.

VirtualHome [Pui18] is a multi-agent platform to simulate activities in a household. Agents are represented as humanoid avatars, which can move and interact with the environment through high-level instructions. It can be used to render videos of human activities, or train agents to perform complex tasks. Its strongest point is being able to generate whole sequences from just some high-level instructions, and its weakest one is probably the render and animation realism.

Habitat [Sav19] enables the training of embodied agents (virtual robots) in a highly efficient photorealistic 3D simulation. It uses its own fast and optimized 3D simulator, and also offers an API for end-to-end development of embodied AI algorithms: defining tasks (e.g. navigation, instruction following, question answering), configuring, training, and benchmarking embodied agents.

ElderSim [Hwa20] is a synthetic action simulation platform that can generate synthetic data on elders' daily activities. It can generate realistic motions of synthetic characters for 55 kinds of activities, with several customizable data-generating options and output modalities. It is based on Unreal Engine, and provides an user interface for making data generation easy. ElderSim is focused on generating data from multiple and fixed points of view, and oriented to action and human pose detection.

2.2.1 Other Tools and Environments

Although not strictly related, we would like to remark some other tools from which we drew inspiration to shape our proposal: UnrealCV, Gazebo, NVIDIA's Isaac Sim and SPEAR.

UnrealCV [Qiu16][Qiu17a] is a project that extends UE4 to create virtual worlds and ease communication with computer vision applications. UnrealCV consists of two parts: server and client. The server is a plug-in that runs embedded into an UE4 game. It uses sockets to listen to high-level UnrealCV commands issued by a client and communicates with UE4 through its C++ API to provide advanced functionality for each command, e.g., rendering per-instance segmentation masks. The client is a Python API which communicates with the server using plain text protocol.

Related to UnrealCV and our work in the sense that all of them make use of UE4, we can find NDDS [To18a], another UE4 plug-in which enables researchers to export images with annotations (RGB images, segmentation masks, depth, object pose, and bounding boxes). The main difference with UnrealCV is its all-UE4 approach without outside communication. The plug-in itself provides a graphical interface within UE4 for configuration and command execution. It is important to remark that, in addition to the image generator, it also features a randomization component to automatically vary lighting, objects, poses, and textures to easily create randomized scenes.

Another framework which helped us design our environment was Gazebo[*], a well-known robot simulator that enables accurate and efficient simulation of robots in indoor and outdoor environments. It integrates a robust physics engine (Bullet, ODE, Simbody, and DART), advanced 3D graphics (using OGRE), and sensors and noise modelling.

NVIDIA's Isaac Sim[†] is a virtual simulator for robotics that lets developers train and test their software using highly realistic virtual simulation environments. When our tool was presented, this project was in an early development phase, and ran under UE4. Currently, the project has been integrated with *NVIDIA Omniverse*, an open platform built for virtual collaboration and real-time photorealistic simulation. More recently, a framework for training RL models, Isaac Gym [Mak21] has been released.

SPEAR (A Simulator for Photorealistic Embodied AI Research) [Rob22] is a simulated environment for training and testing AI agents in photorealistic, 3D environments. It allows researchers to train and test AI agents in a controlled, realistic setting, with the ability to manipulate various aspects of the environment, such as lighting and object placement. It was developed in UE4, and an OpenAI Gym interface is provided for interacting with the environments via Python. The goal of SPEAR is to provide an efficient and effective tool for researchers in the field of embodied AI to develop, test, evaluate and deploy their models, and to advance the state-of-the-art in this field.

2.3 Sim-To-Real

One of the main problems when training a semantic segmentation model is the dataset. Specific large-scale sets of images are needed to solve different problems since we need to include a considerable amount of variability and complete ground truth for each one of them. Collecting data that satisfies these requirements is hard due to the main difficulties of capturing and processing this data manually.

Traditional data collection techniques consist on capturing the environment with the right device. For instance, if we want to capture 2D data, an RGB camera is needed. Following next we have to process the data based on the problem requirements, this means that we need to define a set of classes to represent every entity we want to classify and assign an identifier to each class. In the case of 2D Semantic Segmentation, this codification can be done with colours so we would paint every pixel from each captured image with the chosen colour for that class. This is a complicated and costly process since it requires human operators to paint the images, which can introduce errors in the dataset due to the resolution and the quantity of classes some of these datasets handle. Figure 2.2 showcases a manually labelled

[*] `http://http://gazebosim.org/`

[†] https://developer.nvidia.com/isaac-sim

image from the Mapillary Vistas Dataset[‡], where each colour represents a different entity type.

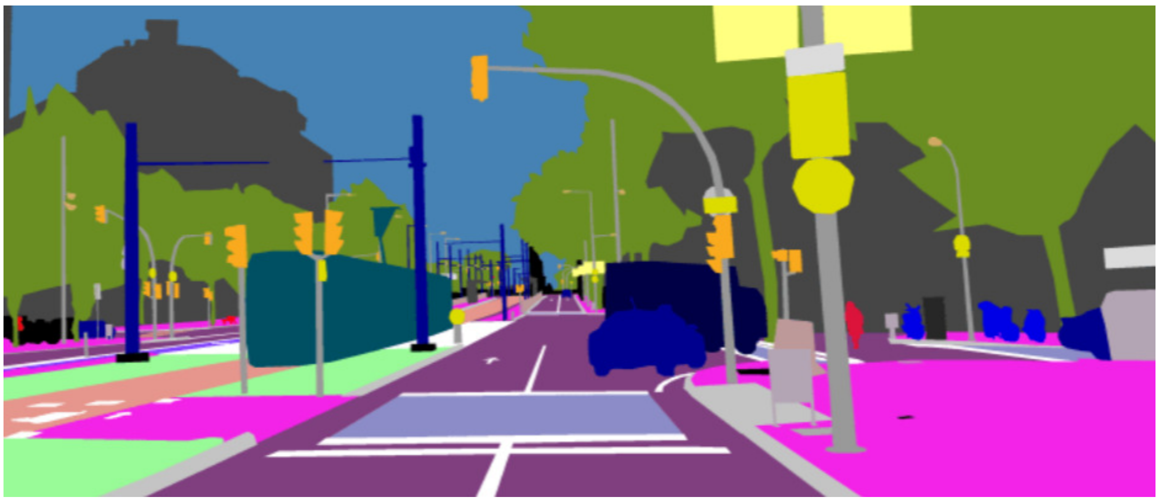


Figure 2.2: Manually labelled scene from the Mapillary Dataset.

The importance of large-scale data when working with data-hungry learning algorithms is critical in order to achieve proper results and generalization. As we saw before, collecting and labelling real-world data is a tedious and costly process partly due to the main complications explained above.

This problem is partially solved thanks to synthetic datasets, which are created without the need for real data. However, for a synthetic dataset to be useful, it must resemble reality as much as possible. Physics simulators like Bullet[§], FleX[¶] or PhysX[||], and 3D rendering engines in Unity, UE4, and PBR engines have played a substantial role in this process.

To recapitulate, synthetic datasets are sets of computer generated or enhanced images for which ground truth can be extracted automatically without the need for human intervention, which alleviates the laborious and complex task of collecting real world data.

In this section, we will follow our previous work [Jov19], we will document sim-to-real and see the different influxes that dominate the state of art. In section 2.3.1, we explain how photorealism has been a very important factor when generating synthetic datasets, as well as how some modern tools, such as ray-tracing, have improved the rendering pipeline to achieve more photorealistic results. In section 2.3.2 we describe the alternative to photorealism, DR, and we will expose some related works. Section 2.3.3 describes different simulators that have helped on this

[‡]https://www.mapillary.com/dataset/vistas?pKey=aFWuj_m4nGoq3-tDz5KAqQ&lat=20&lng=0&z=1.5
[§]https://pybullet.org/wordpress/
[¶]https://developer.nvidia.com/flex
[||]https://www.geforce.com/hardware/technology/physx

matter. Then, Section 2.3.4 showcases UnrealCV and NDDS, two generators able to generate ground truth given a dataset. And finally, Section **??** will put UnrealROX, our sim-to-real simulator, in context.

2.3.1 Photorealism

As we commented previously, a synthetic dataset is useful when it is capable of significantly resembling reality in a considerable manner. In this regard, the use of photorealistic rendering solutions is a necessity.

The state of the art in relation to photorealism is evolving vertiginously. One of the biggest advancements today is the inclusion of real-time ray-tracing technology in NVIDIA GeForce RTX[**] graphics cards. Ray-tracing allows to simulate the physical behaviour of light to provide a cinematographic quality rendering in real time. This implies a great advance in the field of real-time rendering, since real-time ray-tracing has not been possible until the current date. NVIDIA RTX technology introduces some new features to the graphics pipeline:

- **Ray-Triangle Intersection**: This technique lets us decide what to do when a ray is shot into a scene. It is up to the user to determine if it intersects with geometry and what to do with it regarding reflections, occlusion, shadows and things of that nature.
- **BVH**: A BVH is a tree-based data structure that contains multiple hierarchically-arranged bounding boxes that surround different amounts of scene primitives (see Figure 2.3). In this type of data structure each ray only needs to be tested against the BVH using a depth-first tree traversal process instead of against every primitive in the scene, this makes doing high-performance ray tracing into a scene in real time possible.
- **Denoising Filtering**: Ray tracing results in a grainy image for real time solutions. Denoising filtering can produce high fidelity images from ray tracers that appear visually noiseless, as we can see in Figure 2.4. Denoisers need to interpret and merge these pixels in the most appropriate way, that is why many approaches do not work quite well in motion, since pixel caching is not a possibility. Some denoising implementations choose to blur the final result by design to eliminate any kind of artefact, this is the case of Metro Exodus Global Illumination Ray Tracing approach, which does not take into account normal maps, just stores the scene data in a voxel grid as spherical harmonics in world space which encodes some colour and directional properties[‡‡].

The RTX graphics cards also feature a new technology called DLSS that uses

[**]`https://www.nvidia.com/en-us/geforce/20-series/rtx/`
[††]`https://devblogs.nvidia.com/thinking-parallel-part-ii-tree-traversal-gpu/`
[‡‡]`https://www.eurogamer.net/articles/digitalfoundry-2019-metro-exodus-tech-interview`
[§§]`https://www.youtube.com/watch?v=476N4KX8shA`

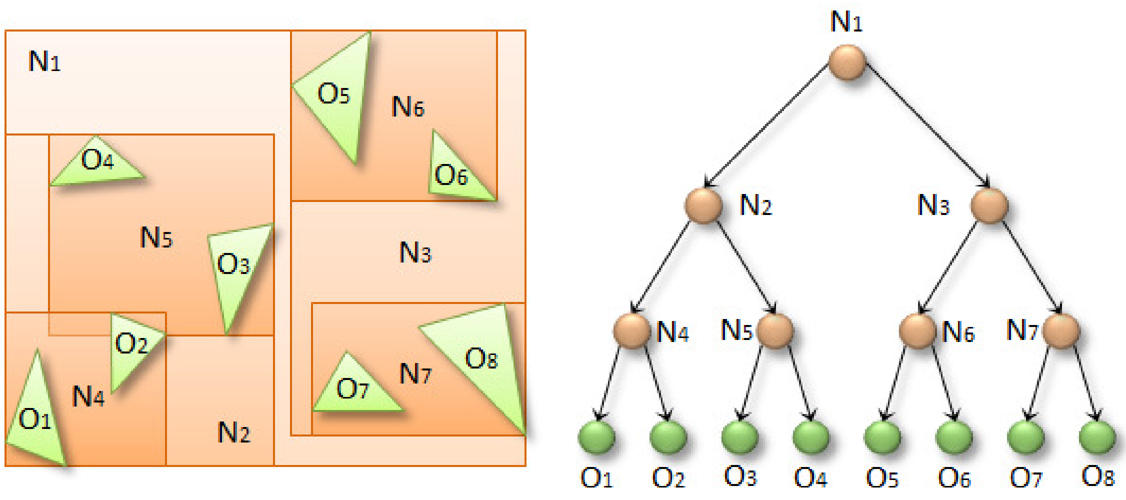


Figure 2.3: Visual representation of a BVH data structure. Source: *NVIDIA Devblogs*[††].

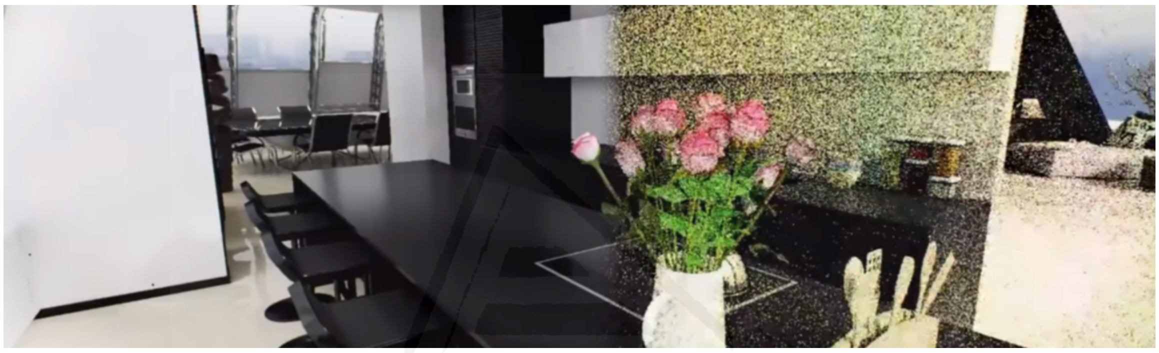


Figure 2.4: Denoise filter applied over a real time ray-traced image. Image from: *What's the Latest? DirectX and the New Rise of Ray Tracing*[§§].

artificial intelligence to generate crisp images while running up to 2 times faster than previous generation GPUs using conventional anti-aliasing techniques. In short, DLSS render pass, renders the scene at a lower resolution and then uses an AI algorithm to make it look as rendered at a higher one, but without the overhead of rendering it at that resolution. That's where the performance enhancement comes from, and ideally, where the maintained visual quality does too. Figure 2.5 displays a comparison between DLSS enabled and disabled.

However, the final output image that DLSS produces is far from perfect, since the super-sampling algorithm generates blur in some cases, which is not very ideal for a photorealistic setup. Added to the fact that DLSS is an NVIDIA exclusive technology that requires new hardware and compatible software as of today.

Even with all these inconveniences, real-time ray tracing is usable to generate photorealistic synthetic datasets, since the quality of the output images is decent enough. Selective ray tracing is a technique that DICE has developed and demonstrated on Battlefield 5[¶¶]. This approach allows them to know which surfaces have

[¶¶]https://www.ea.com/es-es/news/battlefield-5-real-time-ray-tracing

(a) Metro with DLSS disabled. (b) Metro with DLSS enabled.

Figure 2.5: DLSS provides better images without using as much processing power of the graphics card as traditional anti-aliasing techniques. Images are courtesy of Patrick Glynn from Bubble Pony INC.

reflective properties, so they trace more rays from the most reflective surfaces, and less rays from the less reflective ones. Which allows them to have ray tracing reflections where it matters, which lets them deliver a really great result, while still delivering a good frame rate.

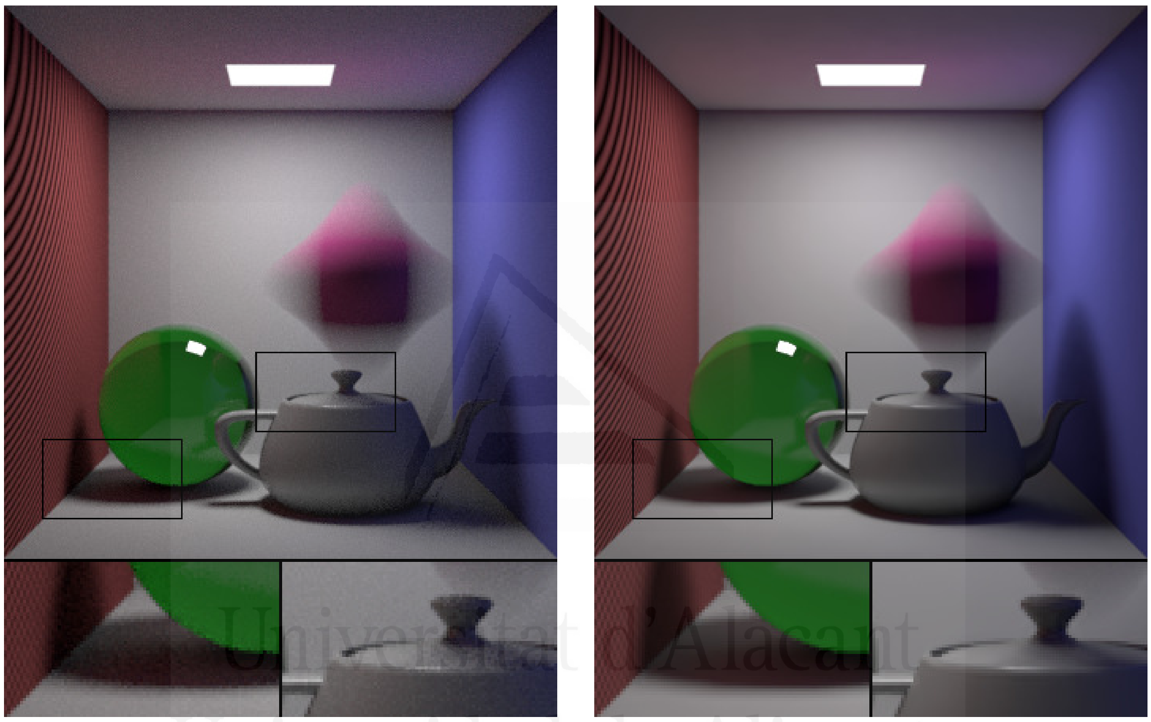
To sum up, if it is desired to use real time ray tracing in its greater potential as of today, it is necessary to make an intelligent use of the technology. Nowadays, our hardware resources do not allow us to trace an infinite number of rays in real time, that is why hybrid solutions must be considered.

We have talked about hyper-realistic rendering in real time and how for the present time, these solutions are limited to high end devices even using advanced optimization techniques. However, offline solutions are more affordable for the average consumer since the main trade-off is rendering time. These systems take longer to compute because they are looking for the most accurate result. In this case, instead of having a hybrid ray-tracing approach in real time to get high resolution and good frame-rate, we focus on achieving the same but offline. This means that we can multiply significantly the amount of rays we trace, meaning that each frame will take a considerable amount of time to render.

Although we increase the number of rays per pixel, it is necessary to apply a denoising filter over the scene to hide minimum artefacts that have been produced. Obviously this denoising will not have to assume as much information as an aggressive denoiser and will be able to return more precise results as we can see in Figure 2.6 taken from the RenderMan[***] render system.

Offline rendering is used in animation films, where they can afford to trace billions of rays per frame. However, rendering a single frame using this technique is very slow, that is why film studios, such as PIXAR or LucasFilms, employ rendering farms where they can distribute the work-load and speed up the production of the film. It is clear that there is a relationship between visual quality and computational time, where the user utilizing the rendering engine is the one who decides how many

[***]https://renderman.pixar.com



(a) 96 rays per pixel, before denoising. (b) After denoising.

Figure 2.6: RenderMan denoising algorithm created by Walt Disney Animation Studios and Disney Research.

traces make a specific scene look right for the imposed quality requirements.

As we can see, the rendering cost is equivalent to the amount of rays we want to trace. The more rays we add, the more accurate and expensive to render the scene will be. The less rays we add, a more aggressive denoising filter will be needed, and as we said previously, an aggressive denoiser decreases the overall quality of the output image.

One of the main problems when it comes to render hyper-realistic scenes is resolution. If we take a modern high-budget animated ray-traced film as an example, we could speak of an average of 1584 rays per pixel. That would be 3 284 582 400 rays for a single 1920 x 1080 image, which makes 13 138 329 600 rays for a 4K frame, without having any lights that add more complexity to it (this data has been extracted from Arnold render system[†††] on production quality).

The problem is evident, the computational cost increases for the same amount of rays per pixel the higher the desired resolution is. We spoke earlier about NVIDIA's DLSS technology, which upscales the input image using Artificial Intelligence. This technique suggests to render first at a lower resolution, which means that we trace less rays (maintaining the same amount of rays per pixel), to then post-process the image upscaling it. NVIDIA has demonstrated that this technique is applicable in real time with DLSS [NVI18]; however, the algorithm is not available to the public as of today.

On the other hand, a paper presented in 2017, proposes the use of GAN [Goo14] to solve this problem. In this paper they discuss and compare the performance of SRResNet and SRGAN to NN, bicubic interpolation, and four state-of-the-art methods [Led17]. The idea is similar: rendering the image at a lower resolution (off-line or on-line), and then post-process it to achieve the desired resolution. Figure 2.7 represents the distinction between different upscaling methods in a practical way.

The main objective of that work is to improve the overall quality of the target images rather than computational efficiency. However, they prove that shallower networks have the potential to provide very efficient alternatives at a small reduction of qualitative performance, nonetheless, this comes at the cost of longer training and testing times.

This type of architecture consists of two neural networks that compete with each other in a zero-sum game framework. The generative network generates candidates while the discriminative network evaluates them. The generative network training objective is to increase the error rate of the discriminative network [Sal16]; this is why GANs deliver a powerful environment for generating realistic looking natural images with high perceptual fidelity.

In conclusion, the best rendering technique in terms of photorealism out of the

[†††]https://docs.arnoldrenderer.com

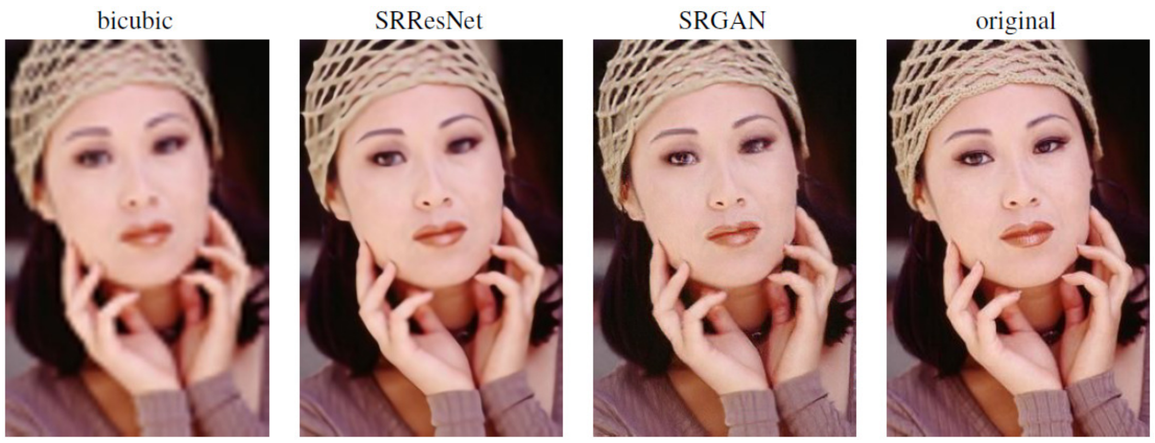


Figure 2.7: Results of a sample from Set5 using bicubic interpolation, SRResNet and SRGAN [4× upscaling] [Led17]

traditional ones[‡‡‡] [Wor95] is non-hybrid ray-tracing; however, this method comes with the disadvantage of a very great performance cost if it is desired to render at a high resolution with a great sample-per-pixel rate. In order to circumvent this issue, the current state of the art is trying to optimize the process using upscaling techniques; however, these approaches affect the quality of the final image. Nonetheless, they can provide accurate enough output images to represent a close-to photorealistic setup. In conclusion, it is conceptually impossible to get a lossless output using these methods, but thanks to this problem, upscaling techniques have evolved to a point where the upscaled result is very close to the original image, hence we could consider that getting close to a photo-realist environment should be the main consideration when creating a photorealistic synthetic dataset.

2.3.2 Domain randomization

As we said in the introduction of this section, domain randomization is the main alternative to photorealism. Domain randomization is a technique for training models with synthetic data that gets generated by randomizing input in a simulator (meshes, materials, lighting...). With enough variability, the real world may appear to the model as just another variation [Tob17b]. There are a lot of approaches to generate synthetic data to construct a domain randomized dataset. The simplest approach is object randomization. If we take NVIDIA's *Isaac Sim*[§§§] domain randomization plugin as an example, displayed in Figure 2.8, we can study simple randomization techniques:

- **Randomize Meshes**: Each class has an array of possible meshes. The selected mesh gets determined thanks to the seed mechanism included in the plugin used to initialize a pseudorandom number generator.

[‡‡‡]https://en.wikipedia.org/wiki/Rendering_(computer_graphics)#Techniques
[§§§]https://docs.nvidia.com/isaac/isaac_sim/index.html

- **Randomize Lighting**: Randomizing how lighting behaves on a specific scene is crucial if it's desired to achieve a data-set able to perform great at different lighting conditions. This Plugin allows the user to tweak how the light looks.
- **Randomized Materials**: Aside randomizing meshes and lighting, we can take existing meshes and modify their materials to increase the samples of a single class without the need of modeling a new object of the same type.

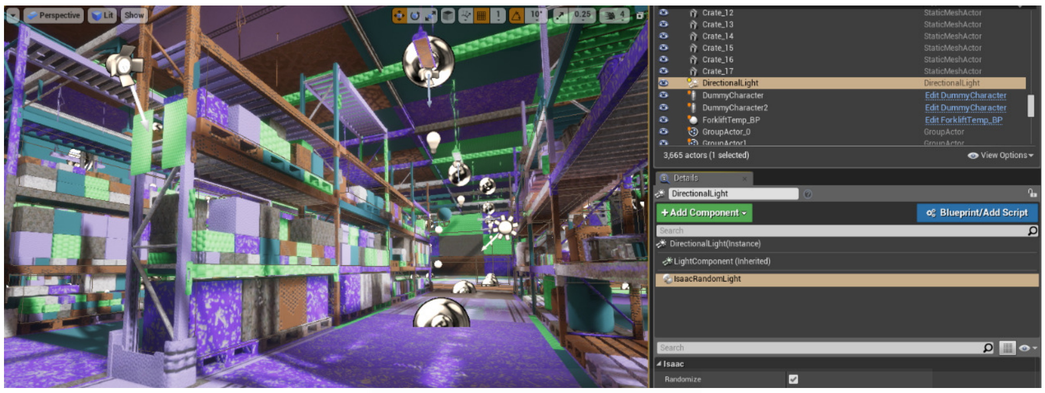


Figure 2.8: Isaac Sim Randomized scene.

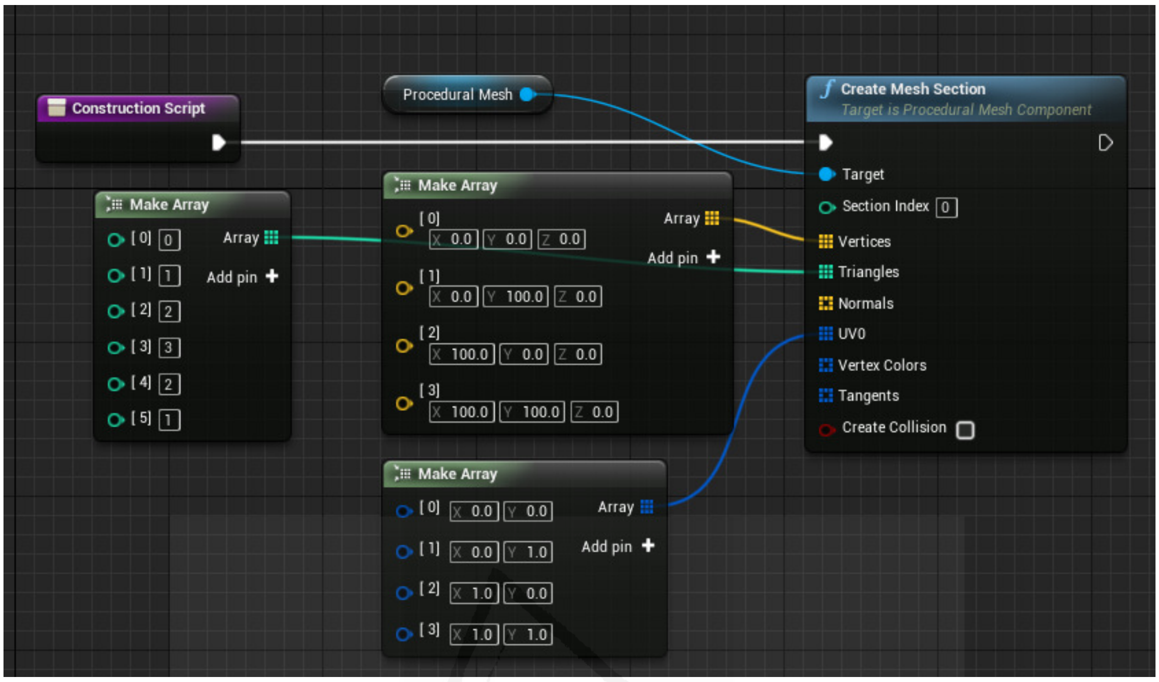
Another interesting approach not observed on this project, is generating and deforming meshes in runtime. UE4 provides a powerful set of tools to generate and edit meshes in real time[¶¶¶]; however, the main complication when using these tools is that we need to work with fully procedural content, meaning that we have to parametrize everything to create objects that make sense and belong to our domain. Figure 2.9 shows a mesh created using these tools.

Defining a mesh in runtime is costly and complicated, that is why it is not considered on most of the cases. However, deforming a mesh could help for future non-rigid objects deformations, which is one of the current limitations of most of the grasping simulators, like UnrealGrasp [Opr19], included on UnrealROX.

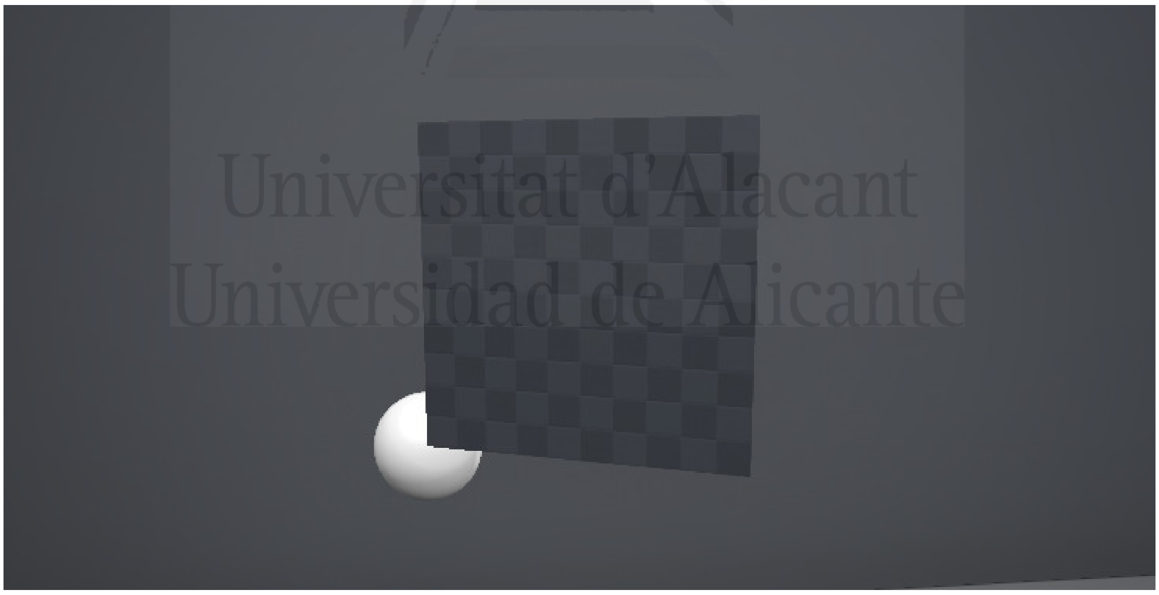
On October of 2018, NVIDIA's research group published a paper that presents SDR as an alternative of DR. In contrast to DR, which places objects and distractors randomly according to a uniform probability distribution, SDR places objects and distractors randomly according to probability distributions specific to the presented problem [Pra18]. The main strength of SDR is that takes into account the context and structure of the scene; hence the parameters exposed for the random system to modify are constraint to real world possible data. Meaning that we won't have nonsensical randomizations as the one exposed in Figure 2.10 for the VKITTI dataset [Gai16].

SDR strikes a balance between photorealism and domain randomization, producing images that are realistic in many respects but nevertheless exhibit large variety.

¶¶¶https://api.unrealengine.com/INT/BlueprintAPI/Components/ProceduralMesh/index.html



(a) Blueprint graph to generate a simple squared plane with a procedural mesh component.



(b) Result of the parameters applied on Figure 2.9a.

Figure 2.9: Using UE4's procedural mesh component to generate a mesh in real time.

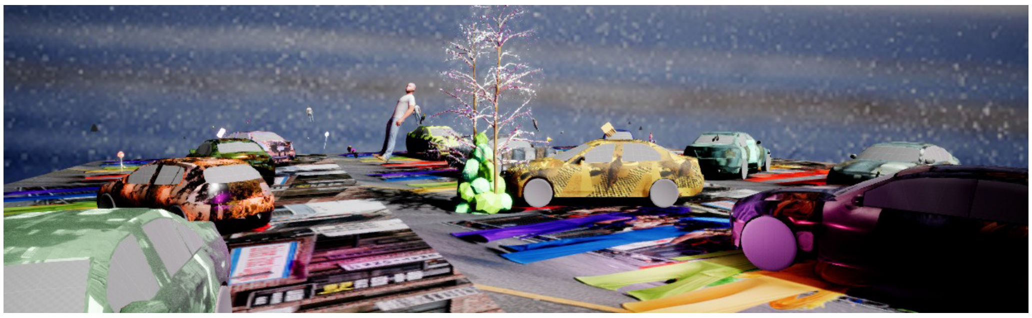


Figure 2.10: Domain randomization intentionally avoids photorealism for variety, which leads to weird generated synthetic data [Pra18].

SDR focuses its attention on improving the VKITTI dataset with sensical domain randomization; for that, a scenario is chosen at random, then global parameters (road curvature, lighting, etc.), which cause context splines (road lanes, sidewalks, etc.) to be generated, upon which objects (cars, pedestrians, cyclists, houses, buildings, etc.) are placed. Figure 2.11, shows a comparison between different synthetic datasets used for training object detection models.

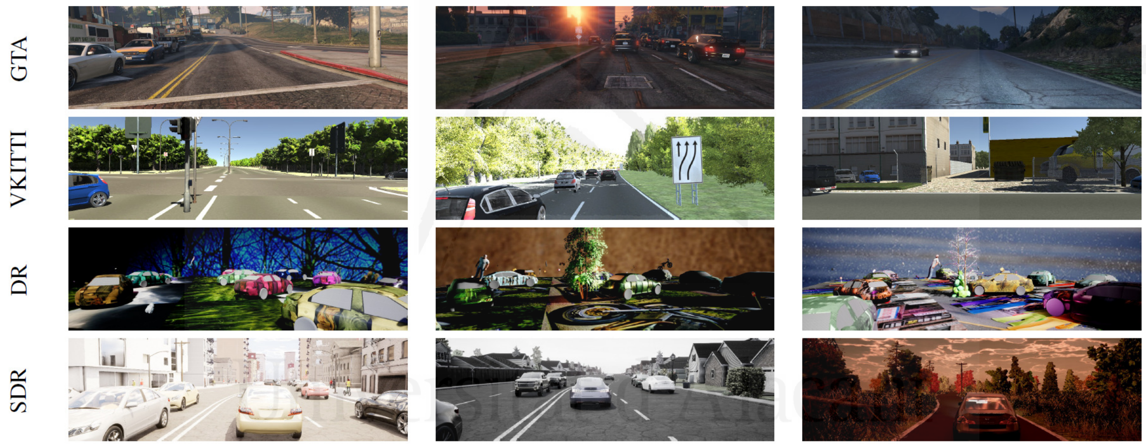


Figure 2.11: Synthetic data coming out from similar datasets [Pra18].

SDR outperforms the GTA-based synthetic data of Sim 200k [Joh17], because SDR provides more variability in the geometry of the scenes. VKITTI is a replica of the KITTI dataset [Gei12a] so it is highly correlated with KITTI, which means not enough variability. Also, DR generates nonsensical data as commented above. According to the authors, synthetic SDR data combined with real KITTI data outperforms real KITTI data alone, thanks to the context based generation.

As we mentioned before, domain randomization sacrifices realism to include a lot of variety in its samples. SDR partially solves this problem by manually controlling the generation parameters with a predefined set of scenarios.

Bhairav Mehta et al. proposed on April of 2019 ADR [Meh19], a type of DR that looks for randomized environments that maximize utility for the agent policy within a given randomization range. The task of searching randomized environments is casted as a RL problem [Sut18], where the samples get parametrized using SVPG, which balances exploitation and exploration [Liu17]. ADR method learns an adaptive randomization strategy that finds problematic environments within the given randomization ranges. They found that training on these instances led to better agent generalization. Also, according to the authors, ADR can provide insight into which dimensions and parameter ranges are most influential, which can aid the tuning of randomization ranges before expensive experiments are undertaken. Figure 2.12, displays the main functioning scheme of Active Domain Randomization.

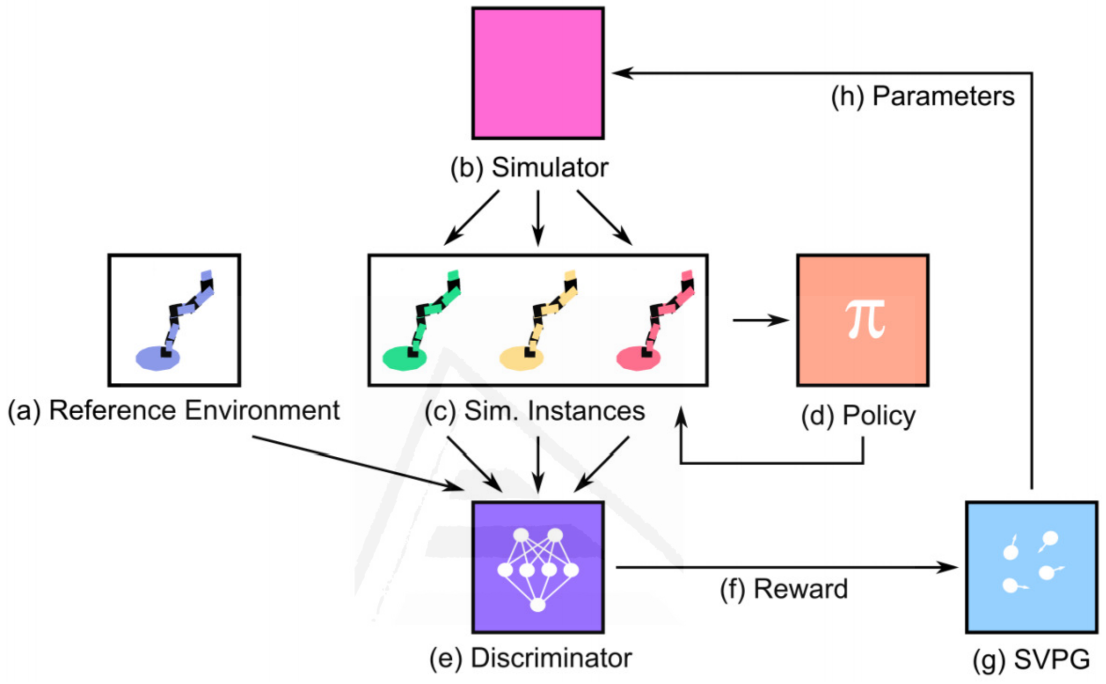


Figure 2.12: "ADR proposes randomized environments (c) or simulation instances from a simulator (b) and rolls out an agent policy (d) in those instances. The discriminator (e) learns a reward (f) as a proxy for environment difficulty by distinguishing between rollouts in the reference environment (a) and randomized instances, which is used to train SVPG particles (g). Enforced through the SVPG formulation, the particles propose a diverse set of environment dynamics, and try to find the environment parameters (h) that are currently causing the agent the most difficulty" [Meh19].

The main issue of ADR is that the reward calculation and discount factor need to be defined manually for each proposed environment due to the RL approach. However, this solution expedites the overall process of obtaining a randomized set that works for a specific problem by automating the parameter lookup.

2.3.3 Simulators

Traditionally, computer vision problems are approached using real world datasets. However, most of the current vision problems need to develop perception models for agents that are physically active in the world, like robots. Commercial datasets are passive and cannot achieve this task, since the content or camera location cannot change according to agents actions. This issue can be solved by placing a physical agent in a world with a series of on-board cameras. The main trade-off of this approach is that learning speed is bounded to real time; also, important critical events, like a car crash, cannot be freely reproduced; plus the fact that working with hardware can become very tedious and expensive. An alternative approach is learning in simulators; however, we encounter two new challenges about generalization:

- **Photorealism**: In Section 2.3.1 we documented some of the main issues photorealism has as of today. However, these problems can be alleviated as hardware and computer graphics get better followed along the improvement of domain adaptation methods.
- **Semantic distribution mismatch**: The models used in simulators are usually hand designed and artificial so they do not really reflect the semantic complexity of real-world. This issue can be improved with domain randomization.

In this section we review some of the most popular simulators that alleviate the commented issues. First, we describe Gibson, followed by AI2-THOR. Next, we discuss how Minos and House3D have been relevant on the current state of the art.

On August 2018, Fei Xia et al. proposed the Gibson Environment [Xia18a], a real-world perception simulator for embodied agents. Figure 2.13 displays and describes the Gibson Environment at a glance.

Gibson provides a dataset that counts with 572 full buildings, which consist on real spaces scanned with 3D scanners, that can be fully explored on the simulator. Gibson also gives the possibility to import arbitrary agents using the URDF[17] as we can see in Figure 2.14.

In Gibson, the physics constraints are enforced by integrating PyBullet3D[18] engine, which encompasses all the collision and gravitational data of the simulator.

In order to provide RGB frames from arbitrary viewpoints, they developed a neural view synthesizer called *Goggles*, that has a baked-in adaption mechanism for transferring to real world. This approach geometrically renders a base image for the target view, which is resorted to a neural network to correct artefacts. Said neural network fills the dis-occluded areas, along with jointly training a backward function for mapping real images onto the synthesized ones, as we can see on Figure 2.15.

One of the main issues of Gibson is the lack of interaction with the scene, AI2-THOR [Kol17] proposes an environment centred mainly in interaction, as we can see in Figure 2.16. AI2-THOR is an open-source visual AI platform that provides a rich actionable 3D environment controlled by a Python interface to communicate with the engine through HTTP commands.

[17]`http://gazebosim.org/tutorials/?tut=ros_urdf`
[18]`https://pybullet.org/wordpress/`

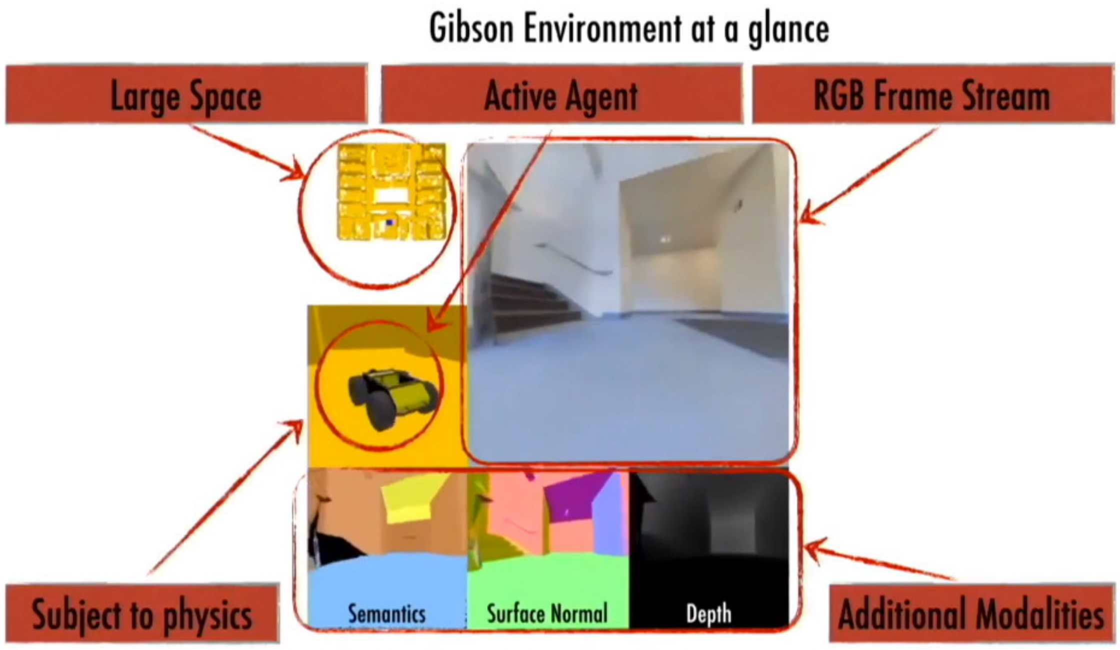


Figure 2.13: On this representation, there is an active agent subject to physics constraints simulated by PyBullet (gravity and collision) that can move around in a large space. Said agent receives a stream of RGB frames, captured with a virtual on-board camera, plus some additional modalities like semantics or depth [Xia18a].

AI2-THOR v1.0 consists of 120 near photorealistic scenes covering four different categories: bedrooms, bathrooms, living rooms and kitchens, within which an agent can interact. Interaction takes various shapes in THOR: the agent can open and close different objects, pick up and place them in various locations, turn on and off lights...

Photorealism is one of the main features of THOR, since it allows better transfer of learned models to the real world. THOR is built on top of the Unity game engine, which allows for observing pixel level results to actions performed by the agents, as well as support on a wide variety of platforms. Some of the drawbacks of this environment are the absence of a controllable 3D agent, the lack of multiple points of view, and the binary categorization for interactables, e.g., a door can be only open or closed. However, the fact that THOR os a Unity project, makes easier its extension.

MINOS [Sav17] is a simulator for navigation in complex indoor environments. The framework provides access to a large number of realistic synthetic scenes and reconstructed indoor spaces such as SUNCG [Son17] and Matterport3D [Cha17] respectively. MINOS provides a 3D agent represented by a cylinder proxy geometry with parametrized radius, height and ground offset; able to navigate through the scenes, obtaining (if desired) information from multimodal sensory inputs, including vision, depth, surface normals, contact forces, and semantic segmentation. The simulator provides two client APIs: a Python wrapper designed to support efficient

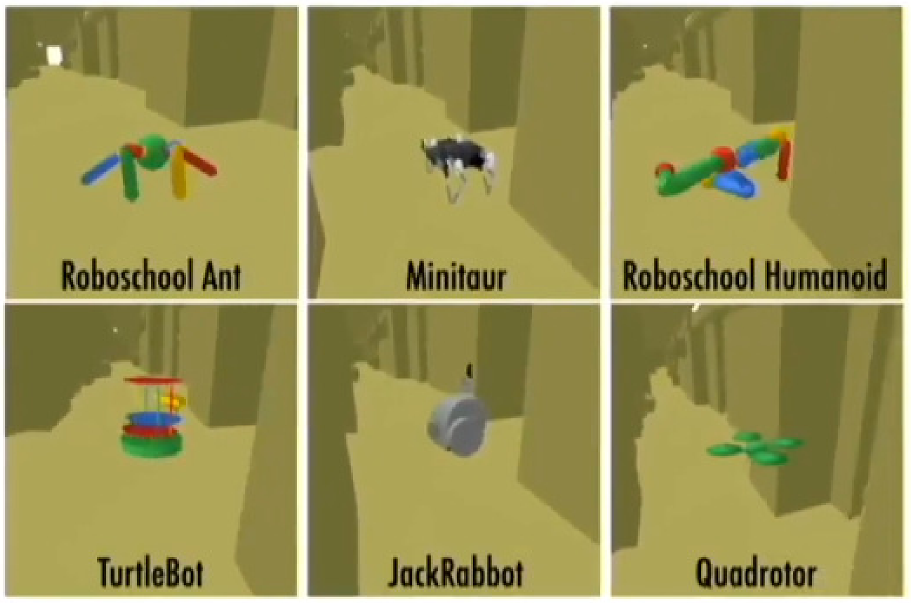


Figure 2.14: Multiple agents on Gibson thanks to the import feature [Xia18a].

RL, and a web client for interactive exploration and data collection. Figure 2.17 showcases an overview of the MINOS framework and APIs.

However, this simulator lacks some features such as photorealism, a more complex geometry model instead of a cylindrical proxy, configurable points of view, and scene interactions.

House3D [Wu18] is a rich, extensible and efficient environment that contains over 45,000 synthetic 3D scenes of visually realistic houses, equipped with a diverse set of fully labelled 3D objects, textures and scene layouts, sourced from the SUNCG dataset. Each scene in SUNCG is fully annotated containing 3D coordinates, object type and the room these objects are in.

There is a simplistic agent able to access the following data: the visual RGB signal of its current first person view, semantic/instance segmentation masks for all the objects visible in the same view, and depth information (see Figure 2.18).

To render SUNCG scenes they have employed OpenGL, which can run on both Linux and MacOS, and provides RGB images, semantic segmentation masks, instance segmentation masks and depth maps.

Nevertheless, House3D has some disadvantages, such as the lack of a realistic 3D agent, the absence of interactive elements and the non-existence of many perspectives to capture the data.

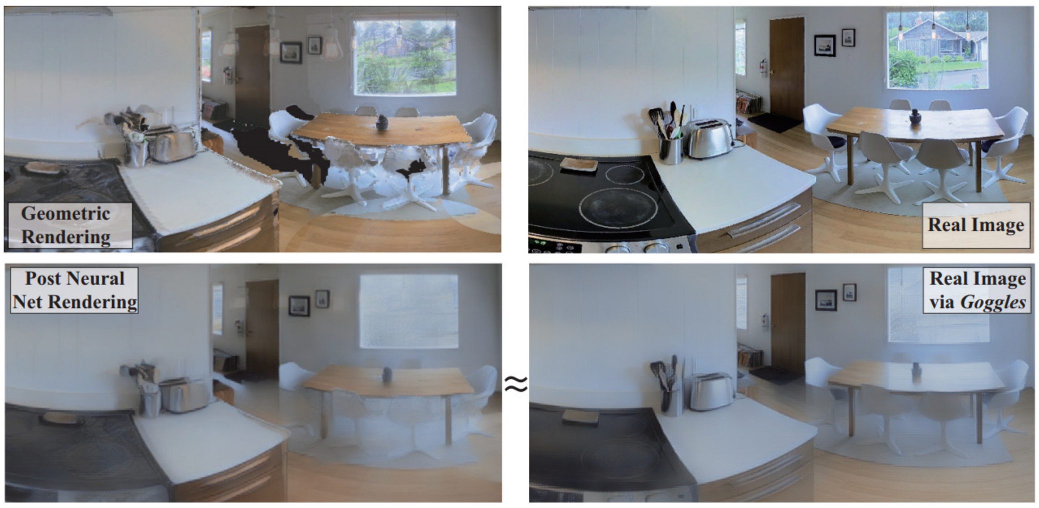


Figure 2.15: Goggles can be seen as corrective glasses of the agent [Xia18a].

2.3.4 Generators

When it comes to generate a dataset, not only simulators have been relevant for this task. Generators decouple the ground truth creation from the problem paradigm. A generator consists of a decoupled interface to generate ground truth given synthetic data. These generators work with environments able to hold said samples, such as Unity or UE4. The main difference between a generator and a simulator that generates, is that the generator is agnostic to the problem paradigm, while the simulator can induce data into their generation process, to solve problems such as the semantic interaction proposal mentioned in Section **??**.

Although generators are agnostic to the problem to resolve and can only be used in the environment for which they were created, they have the great advantage of being decoupled entities. This implies that we will be able to use the generator in any project of the environment for which they have been created without major complications.

In this section we are going to review two generators that have inspired the creation of our simulator, UnrealROX. Concretely, we are going to describe chronologically UnrealCV and NDDS, both working under the UE4 environment.

UnrealCV [Qiu16] [Qiu17b] is a generator for UE4 that extends the engine to create virtual worlds and facilitate communication with computer vision applications. UnrealCV is comprised of two components: server plug-in and client application.

- **Server plugin:** The server consists of a UE4 plug-in that runs embedded into a UE4 project. It uses the built-in socket system of the engine to listen to UnrealCV commands sent by a client, executing them using UE4's C++ API[19]. For example: change the current view mode.

[19]`https://docs.unrealengine.com/en-us/Programming`

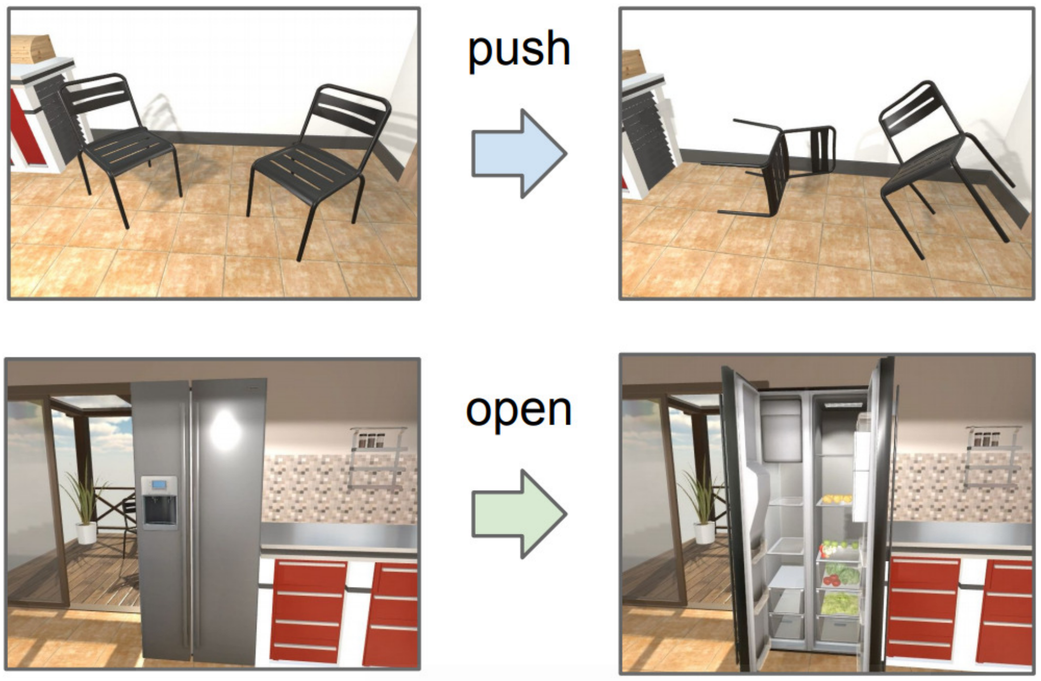


Figure 2.16: Interaction example of AI2-THOR [Kol17].

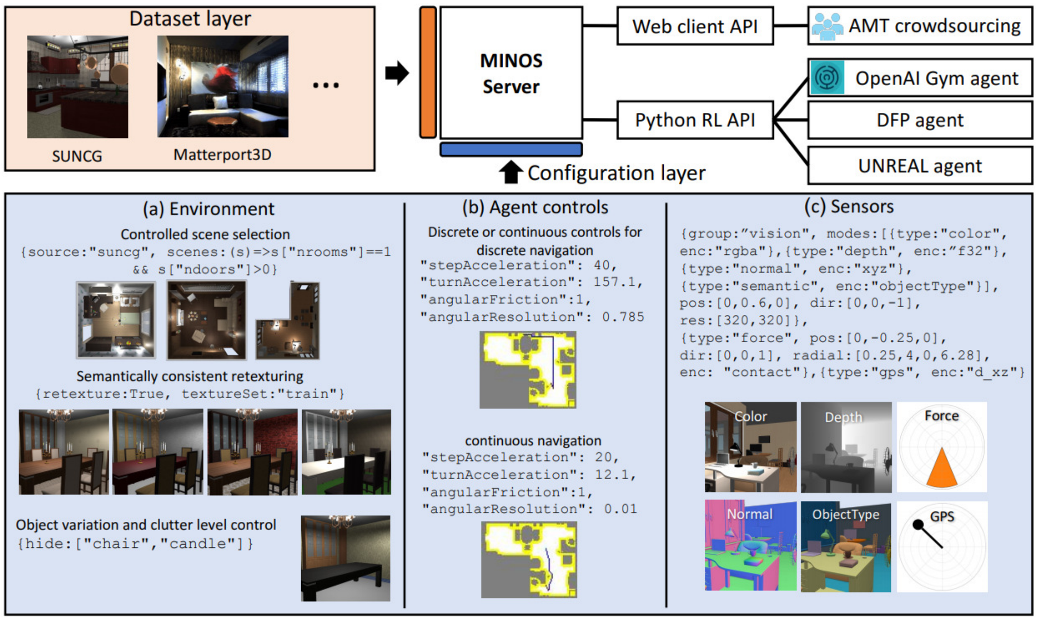
- **Client application:** The client is a Python application which communicates with the server. It sends commands to the server and waits for a response. All the commands can be consulted in the official documentation of the generator.

UnrealCV generates synthetic images and its ground truth as we can see in Figure 2.19. The generator can produce RGB, object instance mask, depth and surface normal images.

Besides the fact that UnrealCV is an open-source tool decoupled and agnostic to the problem to resolve, it has some inconveniences. The main disadvantage of UnrealCV is the delay induced by the client-server communication. This delay slows down the dataset generation process in spite of gaining a good flexibility thanks to the proposed paradigm.

Thang To et al. released on 2018 NDDS [To18b], a UE4 based simulator which solves the problematic issue of generating hand-labelled data. This is troublesome when the task demands expert understanding or not-so-obvious notations (e.g., 3D bounding box vertices). To solve these limitations, using simulators is one of the most recurrent state of the art strategies.

NDDS is an NVIDIA UE4 plug-in that allows computer vision researchers to export high-quality labelled synthetic images. NDDS supports images, segmentation, depth, object pose, bounding box, key points and custom templates. In addition to the exporter, the plug-in includes different components to generate random images. This randomization includes lighting, objects, camera position, poses, textures and distractors, as well as camera path tracking, and so on. Together, these components allow researchers to create random scenes to train deep neural networks.



```
{source:"suncg", scenes:(s)=>s["nrooms"]==1
&& s["ndoors"]>0}
```

```
{retexture:True, textureSet:"train"}
```

```
{hide:["chair","candle"]}
```

```
"stepAcceleration": 40,
"turnAcceleration": 157.1,
"angularFriction":1,
"angularResolution": 0.785
```

```
"stepAcceleration": 20,
"turnAcceleration": 12.1,
"angularFriction":1,
"angularResolution": 0.01
```

```
{group:"vision", modes:[{type:"color",
enc:"rgba"},{type:"depth", enc:"f32"},
{type:"normal", enc:"xyz"},
{type:"semantic", enc:"objectType"}],
pos:[0,0.6,0], dir:[0,0,-1],
res:[320,320]},
{type:"force", pos:[0,-0.25,0],
dir:[0,0,1], radial:[0.25,4,0,6.28],
enc: "contact"},{type:"gps", enc:"d_xz"}
```

Figure 2.17: Overview of the MINOS framework and APIs [Sav17].

This simulator has been used to generate the *Falling Things* dataset: *A Synthetic Dataset for 3D Object Detection and Pose Estimation* [Tre18b]. Which has been successfully tested and used to solve a deep object pose estimation problem [Tre18c], so this proves that networks trained on synthetic data operate correctly when exposed to real-world data.

We are proving slowly that these networks perform well in the real world when trained with synthetic samples, this means that these environments are effectively helping to close the gap between synthetic and real data. Continuous research on simulators depend on how rendering solutions and hardware evolve in the future. Domain randomization has proven to work if the data is sanitized, however, photorealism is as well part of the formula in which simulators have yet to improve.

2.3.5 Our Proposal in Context

After analysing the strong points and weaknesses of the most popular indoor robotic environments, we aimed to combine the strengths of all of them while addressing their weaknesses and introducing new features. In this regard, our work focuses on simulating a wide range of common indoor robot actions, both in terms of poses and object interactions, by leveraging a human operator to generate plausible trajectories and grasps in virtual reality. To the best of our knowledge, this is the first extremely photorealistic environment for robotic vision in which interactions and movements can be realistically generated in virtual reality **by an embodied human agent**. Furthermore, we make possible the generation of raw data (RGB-D/3D/Stereo) and ground truth (2D/3D class and instance segmentation, 6D poses, and 2D/3D bounding boxes) for many vision problems. Although UnrealCV is fairly similar to our

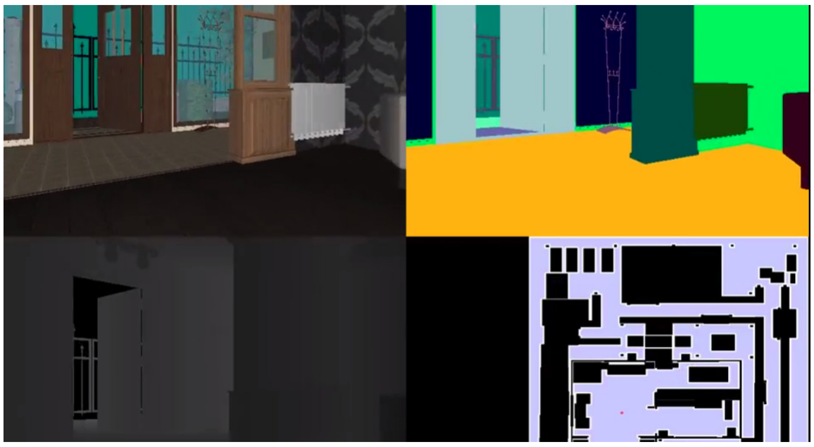


Figure 2.18: House3D agent ground truth. Figure from: *House3D : A Rich and Realistic 3D Environment* [Wu18].

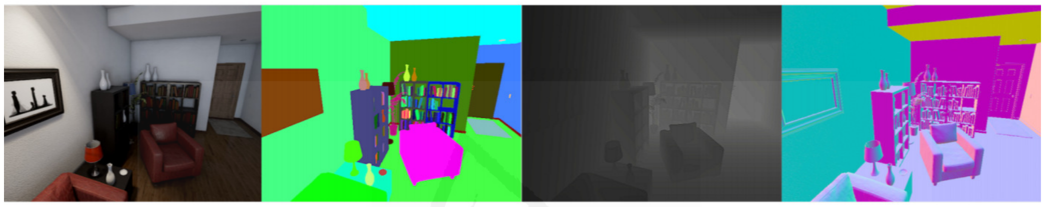


Figure 2.19: Room from the demo RealisticRendering, built by Epic Games. From left to right are the synthetic RGB image, object instance mask, depth, surface normal [Qiu16].

work, since both aim to connect Unreal Engine and computer vision/robotics, we took radically different design decisions: while its architecture is a Python client/server, ours is contained entirely inside UE4 in C++. That architecture allows us to place objects, cameras, and skeletons, and generate images in a more efficient way than other frameworks. Finally, the whole pipeline and tools are released as open-source software with extensive documentation[20].

2.4 System

The rendering engine we chose to generate photorealistic RGB images and immerse the agent in VR is UE4 (Unreal Engine 4). The reasons for this choice are the following ones: (1) it is arguably one of the best game engines able to produce extremely realistic renderings, (2) beyond gaming, it has become widely adopted by Virtual Reality developers and indoor/architectural visualization experts so a whole lot of tools, examples, documentation, and assets are available; (3) due to its impact across various communities, many hardware solutions offer plug-ins for UE4 that make them work out-of-the-box; and (4) Epic Games provides the full C++ source

[20] `https://github.com/3dperceptionlab/unrealrox` and `https://github.com/3dperceptionlab/unrealrox-plus`

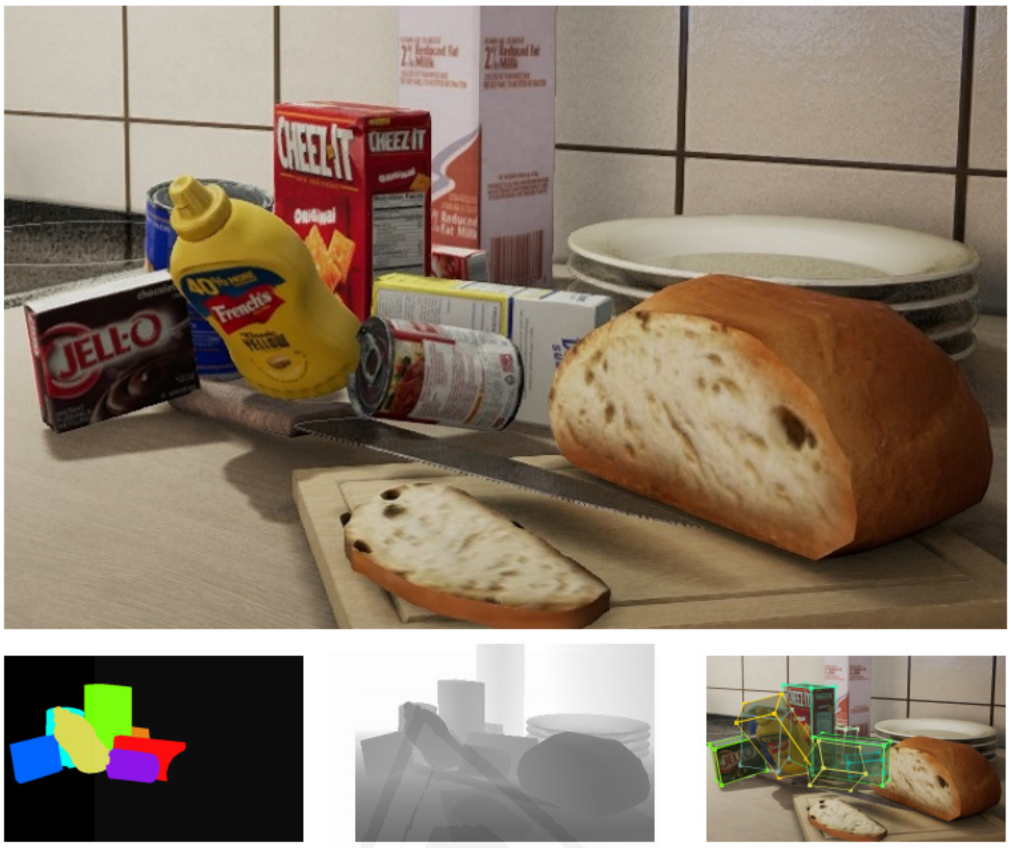


Figure 2.20: Example of an image generated using NDDS, along with ground truth segmentation, depth, and object poses. From the referenced paper.

code and updates to it so the full suite can be used and easily modified for free. Arguably, the most attractive feature of UE4 that made us take that decision is its capability to render photorealistic scenes like the one shown in Figure 2.21. Some UE4 features that enable this realism are: physically-based materials, pre-calculated bounce light via Lightmass, stationary lights, post-processing, and reflections.

It is also important to remark that we do have strict real-time constraints for rendering since we need to immerse a human agent in virtual reality, i.e., we require extremely realistic and complex scenes rendered at very high frame rates (usually more than 80 FPS). By design, UE4 is engineered for virtual reality so it provides a specific rendering solution for it named Forward Renderer. That renderer is able to generate images that meet our quality standards at 90 FPS thanks to high-quality lighting features, MSAA, and instanced stereo rendering.

The whole system is built over UE4 taking advantage of various existing features, extending certain ones with to suit our specific needs, and implementing others from scratch to devise a more efficient and cleaner project that adheres to software design principles. The project was named **UnrealROX**, so half of its name is obviously based on UE4. The other half corresponds to the dataset for which the tool was created: The RobotriX [Gar18a]. The RobotriX is a synthetic image sequence dataset featuring

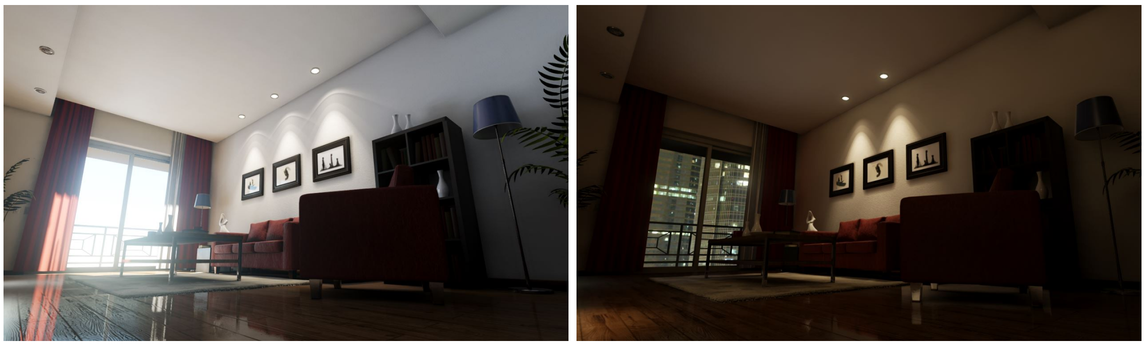


Figure 2.21: Snapshots of the daylight and night room setup for the *Realistic Rendering* released by Epic Games to showcase the realistic rendering capabilities UE4.

hyperrealistic indoor scenes which are explored by robot agents which also interact with objects in a visually realistic manner in that simulated world. It is composed of 38 semantic classes totalling 8M frames recorded at +60 frames per second with full HD resolution. For each frame, RGB-D and 3D information is provided with full annotations in both spaces. For being able to generate such dataset, UnrealROX was created on UE4.

A general overview of our proposal is shown in Figure 2.22. In this section we describe each one of the subsystems that our proposal is composed of: robotic pawns, controller, HUD, grasping, multi-camera, recording, and playback.

2.4.1 Robotic Pawns

One of the most important parts of the system is the representation of the robots in the virtual environment. Robots are represented by the mesh that models them, the control and movement logic, the animations that it triggers, and the grasping system (explained later in its corresponding section). To encapsulate all this, we have created a base class that contains all the common behaviour that any robot would have in our system, which can then be extended by child classes that implement specific things such as the mesh or the configuration of the fingers for the grasping system. Using that encapsulation, we introduced two sample robots in our environment: UE4's mannequin and Aldebaran's Pepper (see Figure 2.23).

In UE4 there is a hierarchy of predefined classes ready to work together that should be used properly in order to take advantage of the facilities offered by the engine. For example, any element that we want to place in a scene must extend the *Actor* class, and at the same time, an *Actor* that is supposed to receive inputs from the user must extend the *Pawn* class (and optionally can have a *Controller* class to abstract input events, as we will see in the next section). This means that our base class that represents the common behaviour of robots must extend *Pawn* class.

The meshes that model characters with joints like our robots are called *SkeletalMesh* in UE4. In addition to the mesh that defines their geometry, they incorporate

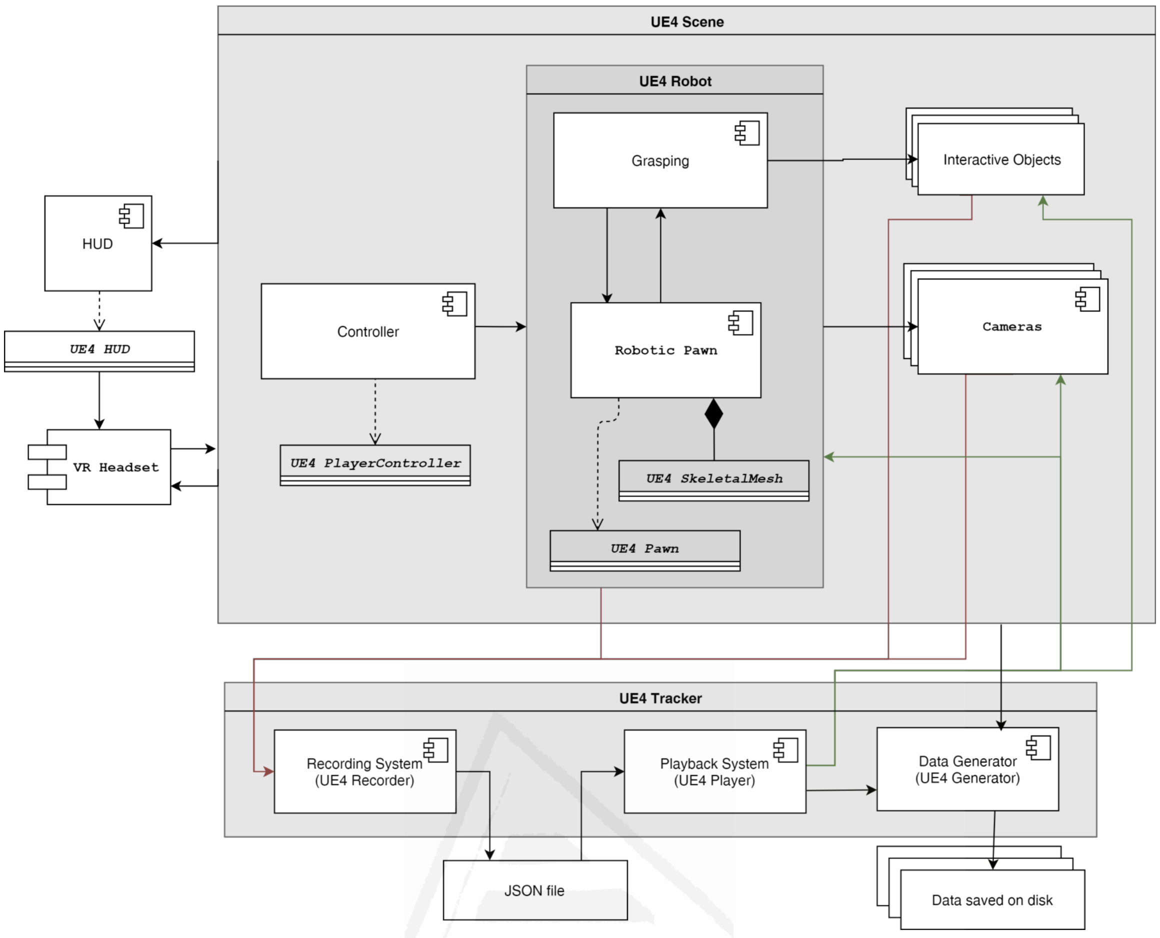


Figure 2.22: System diagram showing the various subsystems and their abstract relationships: Robotic Pawn, Controller, HUD, Multi-camera, Grasping, Recording, and Playback. Gray containers represent the scope in which the various systems act. For instance, the recording, playback, and data generation subsystems compose the UE4 Tracker component in which the recording system takes the poses of the joints of the pawn, the camera poses, and all object poses and then dumps all that information on a per frame basis. That information is later converted into a JSON file which is fed to the playback system, which in turn sets the poses for the pawn's joints, the cameras, and the objects to capture data (RGB, depth, and instance masks). The data generated by the playback subsystem is processed by the generator to produce additional modalities and ground truth (e.g., point clouds, class masks, and bounding boxes). The other container is the UE4 Scene itself and everything inside it is an integral part of the very UE4's map or scene. It contains the controller subsystem, which handles the input coming from the VR controllers and translates it to robotic pawn movements. It also contains the objects (both static and dynamic) and the cameras in the scene. Besides, the UE4 Robot is also a part of it. The UE4 Robot contains the grasping subsystem, which makes use of the robot's finger joints movement and poses to interact with the dynamic objects in order to grab or move them, and the robotic pawn itself, which encapsulates all the assets (mesh and textures) and logic (constraints and animations) of the embodied agent. It is important to notice that the HUD subsystem does not belong (logically) to any of the aforementioned containers since it operates independently just taking important information coming from the scene and rendering it to the VR headset.

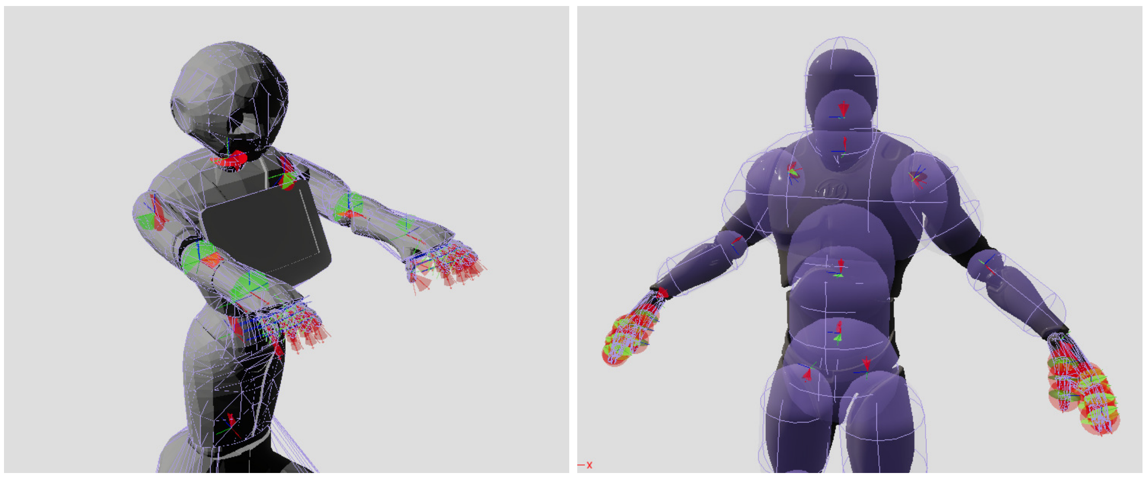


Figure 2.23: Pepper and Mannequin integrated with colliders and constraints.

a skeleton that defines how that geometry will be deformed according to the relative position and rotation of its bones. A *SkeletalMesh* is added as a component to our class (actually, an instance of *SkeletalMeshComponent*).

There are two types of inputs to which our robots must react to, those that come from pressing buttons or axes, and those that come from moving the VR motion controllers. The latter is managed by an UE4 component that must be added to our *Pawn* class and that will modify its position according to the real-world motion controllers movement. We will be able to access the position of these components from the animation class, associate it with the hand bones of the robot *SkeletalMesh*, and move the whole arm by inverse kinematics.

The animation class is created from the *SkeletalMesh*, so it is separate from the *Pawn* class, although the first has to access information from the second. Specifically, our animation classes handles the hand closing animation for the grasping system, and, in the case of robots with legs, it also takes control of the displacement speed to execute the walking animation at different speeds. Finally, the animation class is also used by the playback system (described below) to recover the *SkeletalMesh* pose for a single frame, since it is from where the position and rotation of each joint of the *SkeletalMesh* is accessible for modification.

2.4.2 Controller Subsystem

We would like our system to seamlessly support a wide range of Virtual Reality setups to reach a potentially higher number of users. In this regard, it is important to decouple the controller system from the rest of the environment so that we can use any device (such as the Oculus Rift and the HTC Vive Pro shown in Figure 2.24) without excessive effort. To that end, it is common to have a class that handles all the inputs from the user (in an event-driven way) and then distributes the execution to other classes depending on that input. The very same UE4 provides the base class for this purpose, namely *PlayerController*. Many of these user inputs are focused on controlling the movement and behaviour of a character in the scene, usually represented in Unreal Engine 4 as a *Pawn* class. This means that the *PlayerController* class is closely related to the *Pawn* one. Decoupling input management from functionality is useful as it allows us switching among different controllers for the same *Pawn* (different control types for example), or use the same controller for several ones (if they have the same behaviour for inputs).

Our controller system extends the base class *PlayerController* and handles all kind of user inputs, both from keyboard and VR controllers (we have tested our system with Oculus Rift and HTC Vive). This is configured in the UE4 editor, more specifically in the *Input Project Preferences* panel, where several keys, buttons, or axes can be associated with an event name, which is later bound to a handler function in the custom *PlayerController* class. The controller calls the movement and grasping functionalities from the pawn, and also global system functions as toggling the recording system, restarting the scene, and resetting the VR headset position. It also controls the HUD system for showing input debugging feedback.

Figure 2.24: Seamlessly supported VR headsets thanks to the decoupled controller subsystem: HTC Vive Pro (a) and Oculus Rift (b).

2.4.3 HUD Subsystem

It is convenient for any system to feature a debug system that provides feedback about the application state to the user. In UnrealROX, we offer an interface to show information at various levels to the user if requested. This information is presented in a HUD which can be turned off or on to the user's will. It can even be completely decoupled from the system as a whole for maximum performance. The main information modalities provided by the HUD are the following ones:

- **Recording state:** A line of text with the recording state is always shown in the HUD in order to let the user know if his movements through the scene are being recorded.

- **States:** Notifies the user with a message on the screen of the relevant buttons pressed, the joints in contact with an object, the profiling being activated, etc. The amount of seconds these messages last on screen can be established independently. Most of them are printed for 5 seconds.
- **Error:** Prints a red message indicating an error that lasts in screen for 30 seconds (or until another error occurs). An example of this would be trying to record without the tracker on the scene.
- **Scene Capture:** It allows us to establish a debugging point of view so that we can see our robot from a different point of view than the first person camera.

We have implemented this functionality extending the HUD class that UE4 provides, and we also made it fully decoupled from the rest of the system in a simple way by implementing an interface[21]. Classes that inherit from HUD class have a canvas and a debug canvas on which primitive shapes can be drawn. It provides some simple methods for rendering text, textures, rectangles, and materials which can also be accessed from blueprints. An example of texture drawing in practice in our project is the Scene Capture, which consists in drawing a texture in the viewport captured from an arbitrary camera (as shown in Figure 2.25). This will be useful for the user to see if the animations are being played correctly in a Virtual Reality environment.

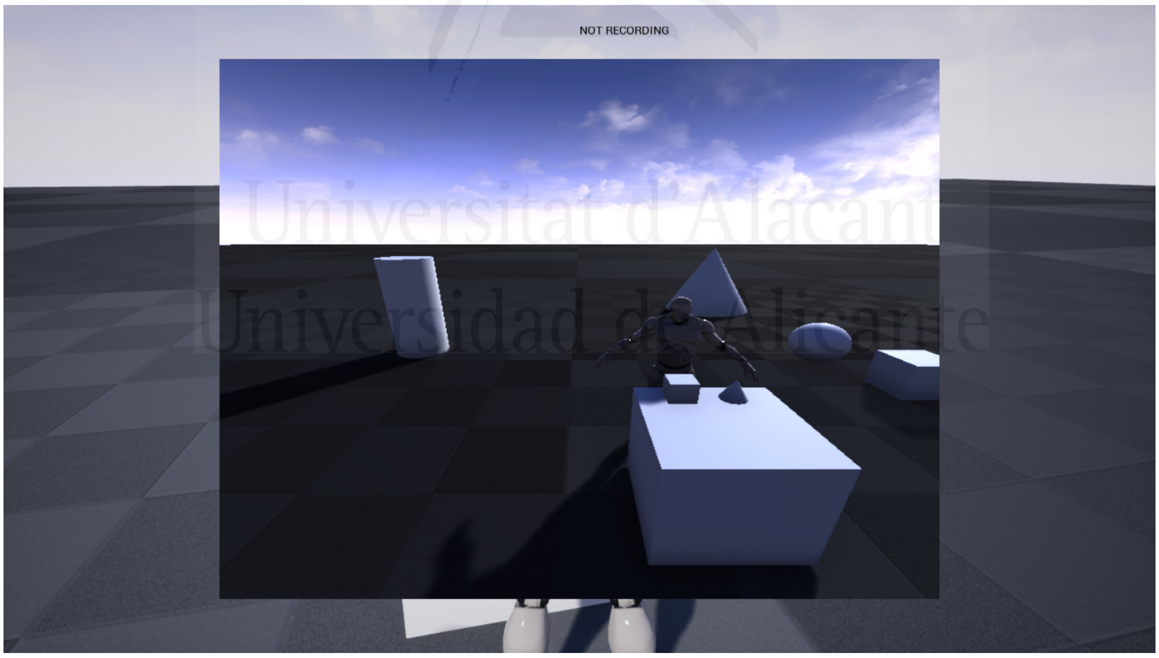


Figure 2.25: Scene Capture drawn in the viewport.

[21]https://docs.unrealengine.com/en-US/Programming/UnrealArchitecture/Reference/Interfaces

2.4.4 Grasping Subsystem

The grasping subsystem is considered one of the core components of UnrealROX. We have focused on providing a realistic grasping, both in the way the robot grasp an object and in the movements it makes. When grasping an object we need to simulate a real robot behaviour, thus smooth and plausible movements are needed. The grasping action is fully controlled by the user through the controls, naturally limited to the degrees of freedom of the human body. In this way, we achieve a good representation of a humanoid robot interacting in a realistic home environment, also known as assistive robots which are the current trend in the field of social robotics.

Current approaches for grasping in VR environments are animation-driven, and based on predefined movements [Ocu17][Loo][Ocu16]. This will restrict the system to only a few pre-defined object geometries hindering user's interaction with the environment resulting also in a unrealistic grasping. In contrast with these approaches, the main idea of our grasping subsystem consists in manipulating and interacting with different objects, regardless of their geometry and pose. In this way, the user can freely decide which object to interact with without restrictions. The robot can manipulate an object with each hand, and change an object from one hand to the other. It can also manipulate two different objects at the same time, drop them freely or throw them around the scene.

At the implementation level of this subsystem, we make use of UE4's *trigger volumes* placed on each one of the finger phalanges as we can see in Figure 2.26. These *triggers* act as sensors that will determine if we are manipulating an object in order to grasp it. With the controllers we are able to close the robot's hands limiting individually each finger according to the *triggers*. We also implement a logic to determine when to grasp or release an object based on the *triggers* state. Fingers' positions change smoothly in order to replicate a real robot hand behaviour and to avoid passing through objects.

A sequence example grasping two objects with our custom system is shown in Figure 2.27.

The most common issues of existing grasping systems on virtual reality environments are:

- Animation-driven grasping with predefined movements
- Impossibility to deal with new object geometries
- Unrealistic grasping

Through our grasping subsystem we dealt with above limitations by providing a realistic grasping, independent of the object geometries and without predefined movements. This will allow us to provide infinite grasping situations depending only on the user's ability of grasping an object or interact with the scene.

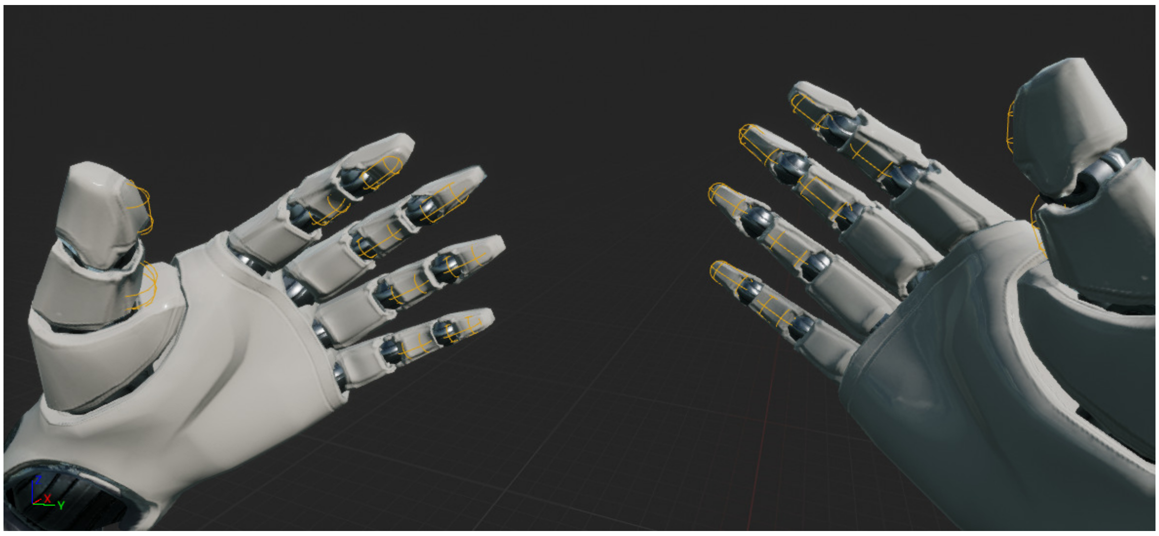


Figure 2.26: *Sphere trigger volumes* placed on finger phalanges of both hands represented in yellow.

2.4.5 Multi-camera Subsystem

Most robots in the public market (such as Pepper or Baxter) integrate multiple cameras in different parts of their bodies. In addition, external cameras are usually added to the system to provide data from different points of view, e.g., ambient assisted living environments tend to feature various camera feeds for different rooms to provide the robot with information that it is not able to perceive directly. In UnrealROX, we want to simulate the ability to add multiple cameras in a synthetic environment with the goal in mind of having the same or more amount of data that we would have in a real environment. For instance, in order to train a data-driven grasping algorithm it would be needed to generate synthetic images from a certain point of view: the wrist of the robot. To simulate this situation in our synthetic scenario, we give the user the ability to place cameras attached to sockets in the robot's body, e.g., the wrist itself or the end-effector (eye-in-hand). Furthermore, we also provide the functionality to add static cameras over the scene.

To implement this subsystem, we make use of UE4's *CameraActor* as the camera class and the *Pawn* class as the entity to which we will attach them. By default, UE4 does not allow us to precisely attach components in the editor so it is necessary to define a socket-camera relationship in the *Pawn* class. This is due to the fact that it has direct access to the skeleton to which we will be attaching specific cameras.

The objective of the *CameraActor* class is to render any scene from a specific point of view. This actor can be placed and rotated at the user's discretion in the viewport, which makes them ideal for recording any type of scene from any desired point of view. The *CameraActor* is represented in UE4 by a 3D camera model and like

[21]`https://www.softbankrobotics.com/emea/en/robots/pepper`
[21]`https://www.rethinkrobotics.com/baxter/`

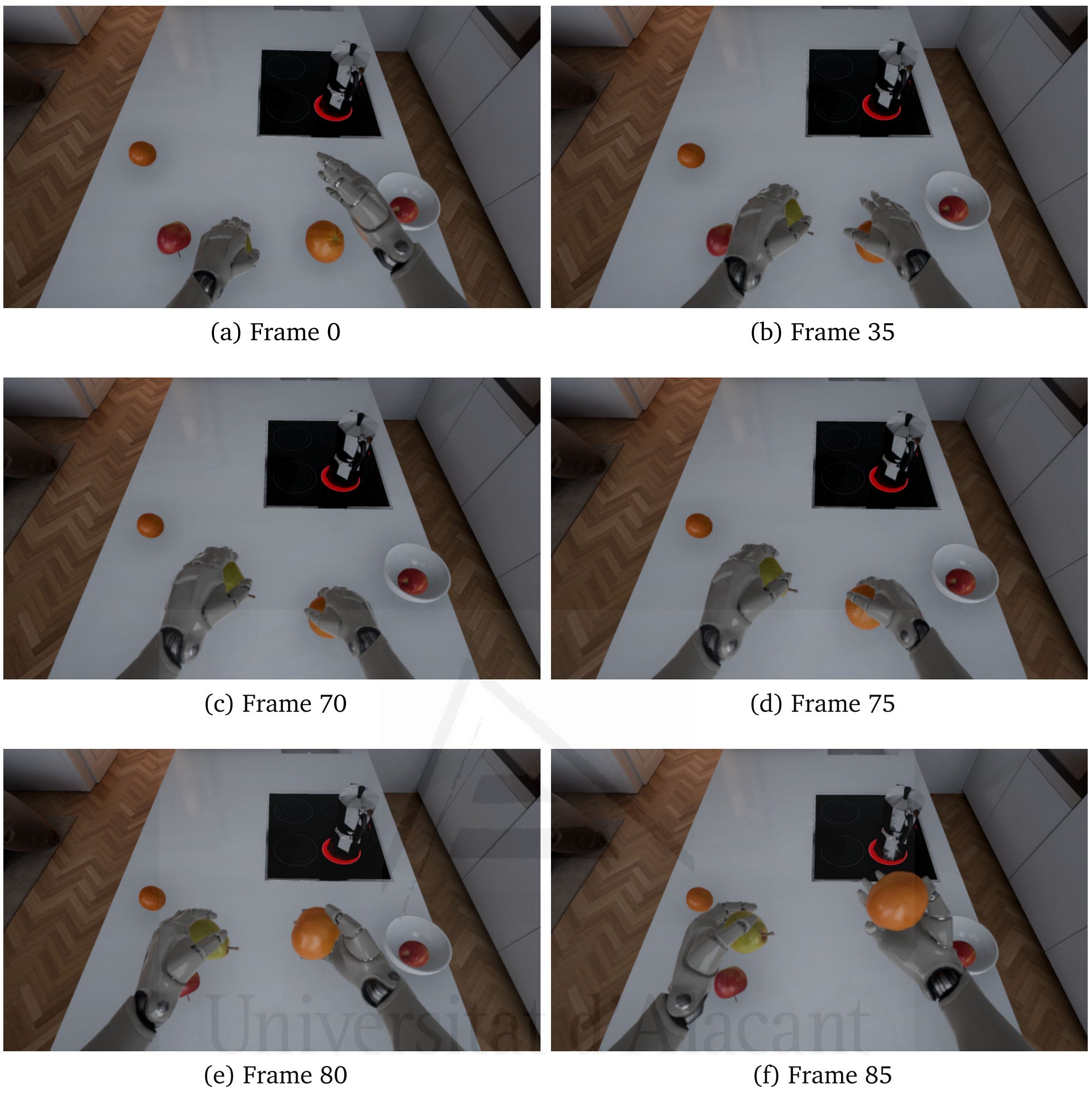
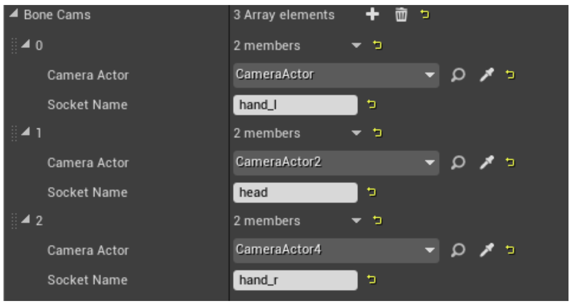


Figure 2.27: Sequence of 6 frames ($Seq = F_0, F_{35}, F_{70}, F_{75}, F_{80}, F_{85}$) representing a grasping action with right hand meanwhile holding a fruit with the left hand. Frames are left-right and top-down ordered

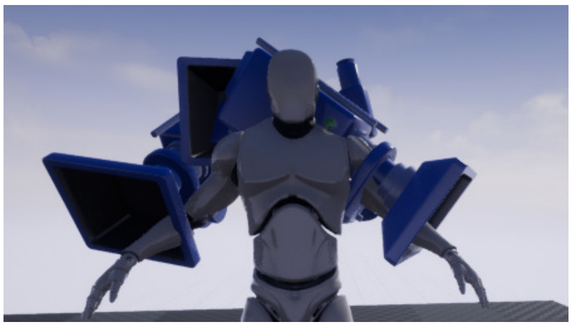
any other actor, it can be moved and rotated in the viewport. Apart from handling attached and static cameras, UnrealROX exposes the most demanded camera settings through its interface (projection mode, FoV, colour grading, tone mapping, lens, and various rendering effects), as well as providing additional features such as creating stereo-vision setups.

To implement the camera attachment functionality we make extensive use of the *AttachToActor* function provided by UE4, which is in charge of parenting one actor with another following some attachment rules. We can specify the socket to which we want to attach the object. This means that when the selected socket changes its transform, the attached object will change it too according to the specified *At-*



Bone Cams — 3 Array elements
0 — 2 members
Camera Actor — CameraActor
Socket Name — hand_l
1 — 2 members
Camera Actor — CameraActor2
Socket Name — head
2 — 2 members
Camera Actor — CameraActor4
Socket Name — hand_r

(a) Representation of the array.



(b) Pawn Actor with cameras.

Figure 2.28: In-engine representation of the array of structs. In (2.28a) we can see the representation of the array struct in the instance of the object, while in (2.28b) we see its visual representation in the engine.

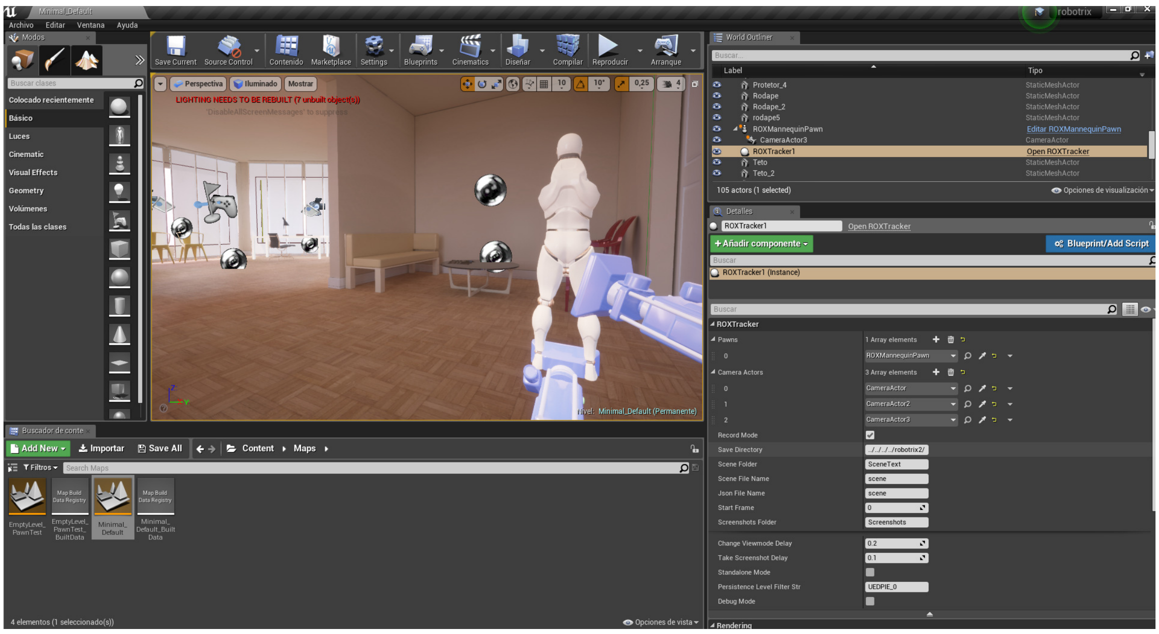
tachmentRules. **The *AttachmentRules* can be defined separately for location, rotation, and scale so we can define a fixed transform of the camera relative to the socket.** This lead us to define an implicit relationship between the *CameraActor* and the socket it is attached to. To make the attachment process easier, we provide a friendly user interface inside the editor (see Figure 2.28).

2.4.6 Recording Subsystem

UnrealROX decouples the recording and data generation processes so that we can achieve high frame rates when gathering data in VR (Virtual Reality) without decreasing performance due to extra processing tasks such as changing rendering modes, cameras, and writing images to disk. In this regard, the recording subsystem only acts while the agent is embodied as the robot in the virtual environment. When enabled, this mode gathers and dumps, on a per-frame basis, all the information that will be needed to replay and reconstruct the whole sequence, its data, and its ground truth. That information will be later used as input for the playback system to reproduce the sequence and generate all the requested data.

In order to implement such behaviour we created a new UE4 *Actor*, namely *ROXTracker*, which overrides the *Tick* function. This new invisible actor is included in the scene we want to record and executes its tick code for each rendered frame. That tick function loops over all cameras, objects, and robots (skeletons) in the scene and writes all the needed information to a text file in an asynchronous way. For each frame, the actor dumps the following information: recorded frame number, timestamp in milliseconds since the start of the game, the position and rotation for each camera, the position, rotation, and bounding box minimum and maximum world-coordinates for each object, and the position and rotation of each joint of the robot's skeleton.

The information is dumped in raw text format for efficiency, after the sequence is



ROXTracker

Pawns	1 Array elements
0	ROXMannequinPawn
Camera Actors	3 Array elements
0	CameraActor
1	CameraActor2
2	CameraActor3
Record Mode	✓
Save Directory	../../../../robotrix2/
Scene Folder	SceneText
Scene File Name	scene
Json File Name	scene
Start Frame	0
Screenshots Folder	Screenshots
Change Viewmode Delay	0.2
Take Screenshot Delay	0.1
Standalone Mode	
Persistence Level Filter Str	UEDPIE_0
Debug Mode	

Figure 2.29: *ROXTracker* custom interface showing the robot mannequin, multiple cameras, and various parameters to configure which pawns and cameras are tracked and other sequence details.

fully recorded, the raw text file is processed and converted into a more structured and readable JSON file so that it can be easily interpreted by the playback system.

2.4.7 Playback Subsystem

Once the scene has been recorded, we can use the custom user interface in UE4 to provide the needed data for the playback mode: the sequence description file in JSON format and an output directory. Other parameters such as frame skipping (to skip a certain amount of frames at the beginning of the sequence) and dropping (keep only a certain amount of frames) can also be customized (see Figure 2.29).

This mode disables any physics simulation and interactions (since object and skeleton poses will be hard-coded by the sequence description itself) and then interprets the sequence file to generate all the raw data from it: RGB images, depth maps, instance segmentation masks, and normals. For each frame, the playback mode moves every object and every robot joint to the previously recorded position and sets their rotation. Once everything is positioned, it loops through each camera. For each one of them, the aforementioned rendering modes (RGB, depth, instance, and normals) are switched and the corresponding images are generated as shown in Figure 2.30.

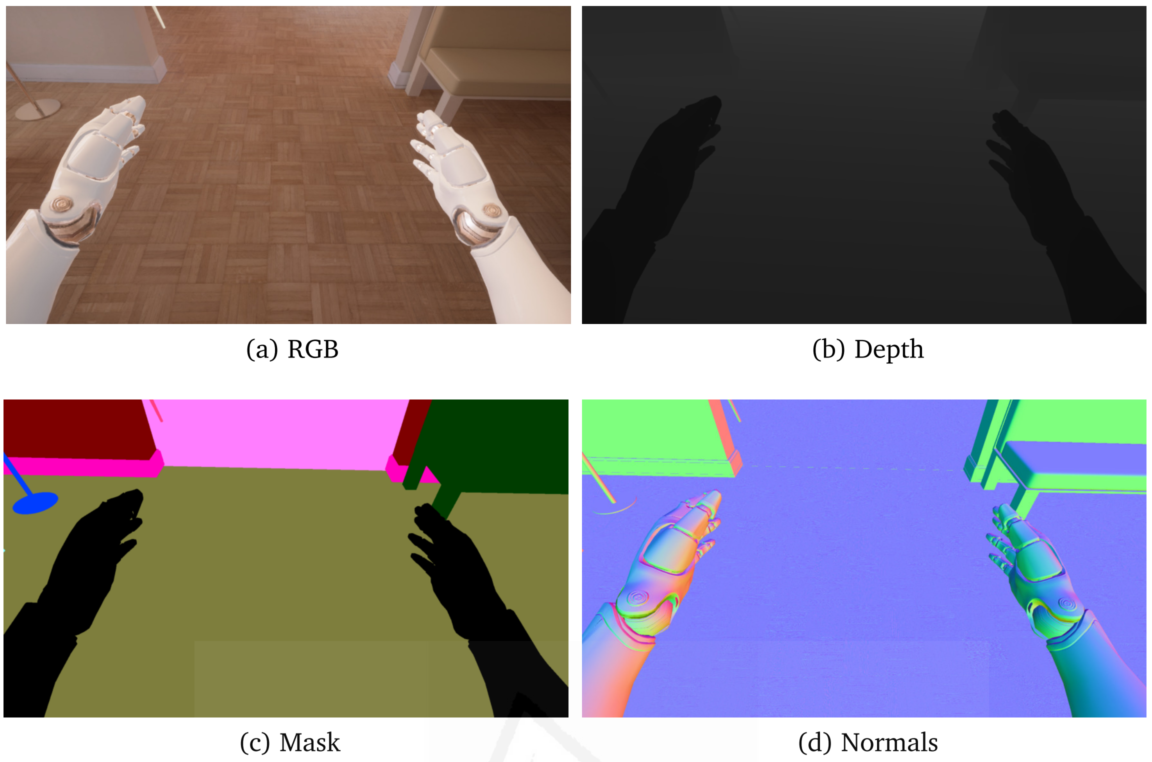


Figure 2.30: Rendering modes cycled by the playback mode.

2.5 Applications

UnrealROX environment has multiple potential application scenarios to generate data for various robotic vision tasks. Traditional algorithms for solving such tasks can take advantage of the data but the main purpose of this environment is providing the ability to generate large-scale datasets. Having the possibility of generating vast amounts of high-quality annotated data, data-driven algorithms such as deep learning models can especially benefit from it to increase their performance, in terms of accuracy, and improve their generalization capabilities in unseen situations during training. The set of tasks and problems that can be addressed using such data ranges from low to high-level ones, covering the whole spectrum of indoor robotics. Some of the most relevant low-level tasks include:

- Stereo Depth Estimation: One of the typical ways of obtaining 3D information for robotics is using a pair of displaced cameras used to obtain two different views from the same scene at the same time frame. By comparing both images, a disparity map can be obtained whose values are inversely proportional to the scene depth. Our multi-camera system allows the placement of stereo pairs at configurable baselines so that the environment is able to generate pairs of RGB images, and the corresponding depth, from calibrated cameras.
- Monocular Depth Estimation: Another trending way of obtaining 3D information consists of using machine learning methods to infer depth from a single

RGB image instead of a stereo pair. From a practical standpoint, it is specially interesting since it requires far less hardware and avoids the need for calibration strategies. Our multi-camera system generates by default depth information for each RGB frame (see Figure 2.31).

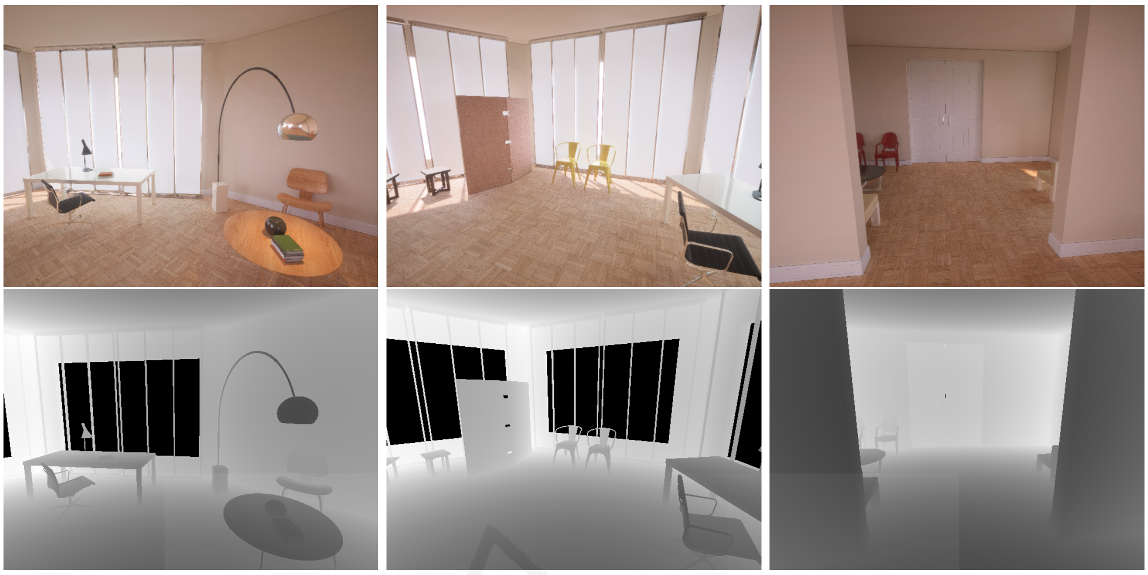


Figure 2.31: Sample RGB sequence for monocular depth estimation.

- Object Detection and Pose Estimation: Being able not only to identify which objects are in a given scene frame but also their estimated pose and bounding box is of utmost importance for an indoor robot. Our environment is able to produce 2D and 3D bounding boxes for each frame as ground truth. Furthermore, for each frame of a sequence, the full 6D pose of the objects is annotated too (see Figure 2.32).

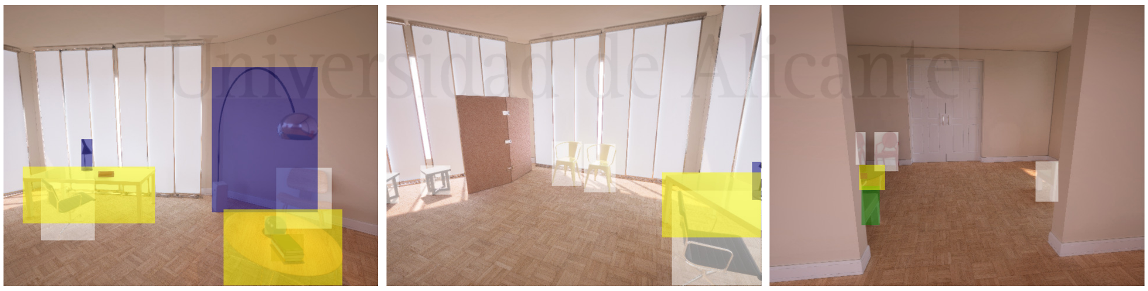


Figure 2.32: Sample bounding box annotations for the RGB sequence shown in Figure 2.31.

- Instance/Class Segmentation: For certain applications, detecting a bounding box for each object is not enough so we need to be able to pinpoint the exact boundaries of the objects. Semantic segmentation of frames provides per-pixel labels that indicate to which instance or class does a particular pixel belong. Our environment generates 2D (per-pixel) and 3D (per-point) labels for instance and class segmentation (see Figure 2.33).

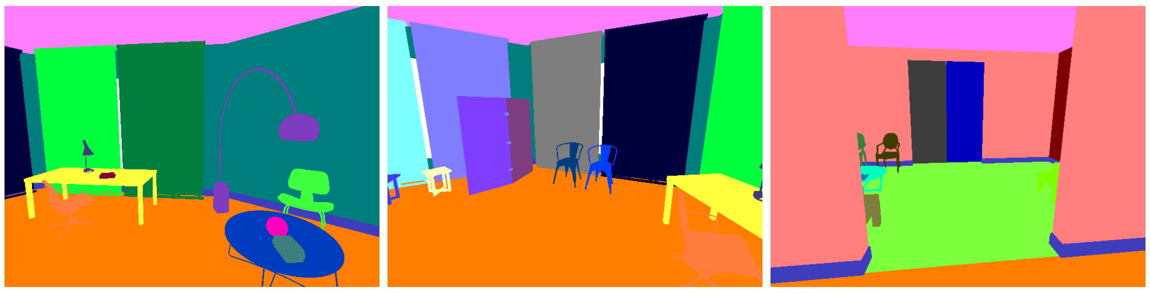


Figure 2.33: Sample instance segmentation sequence for the RGB images shown in Figure 2.31.

- Normal Estimation: Estimating the normals of a given surface is an important previous step for many other tasks. For instance, certain algorithms require normal information in a point cloud to extract grasping points for a robot. UnrealROX provides per-pixel normal information.

That low-level data enables other higher-level tasks that either make use of the output of those systems or take the low-level data as input or even both possibilities:

- Hand Pose Estimation: Estimating the 6D pose of each joint of the hands provides useful information for various higher-level tasks such as gesture detection, grasping or collaboration with other robots. We provide per-frame 6D pose annotations for each joint of the robot's hands.
- Visual Grasping and Dexterous Manipulation: Grasping objects and manipulating them while grasped with one or both hands is a high-level task which can be solved using information from various sources (RGB images, depth maps, segmentation masks, normal maps, and joint estimates to name a few). In our case, we provide sequences in which the robot interacts with objects to displace, grab, and manipulate them so that grasping algorithms can take advantage of such sequences recorded from various points of view (see Figure 2.34).
- Robot Pose Estimation: As well as providing 6D pose for hand joints, our environment also provides such information for all the joints of a robot on a per-frame basis. This allows training and testing body pose estimation algorithms which can be extremely useful in indoor environments to analyse behaviours and even collaborate with other robots too. To that end, we equipped our multi-camera system with the capability of adding room cameras that capture full bodies typical from assisted indoor living (see Figure 2.35).
- Obstacle Avoidance and Navigation: By leveraging various types of low-level information such as RGB images, depth maps, bounding boxes, and semantic segmentation, robots can learn to avoid obstacles (by detecting objects and estimating their distance) and even navigate in indoor environments (by building a map to localize themselves in the indoor scene while avoiding objects and walls and being able to reason semantically to move intelligently). **Fur-**

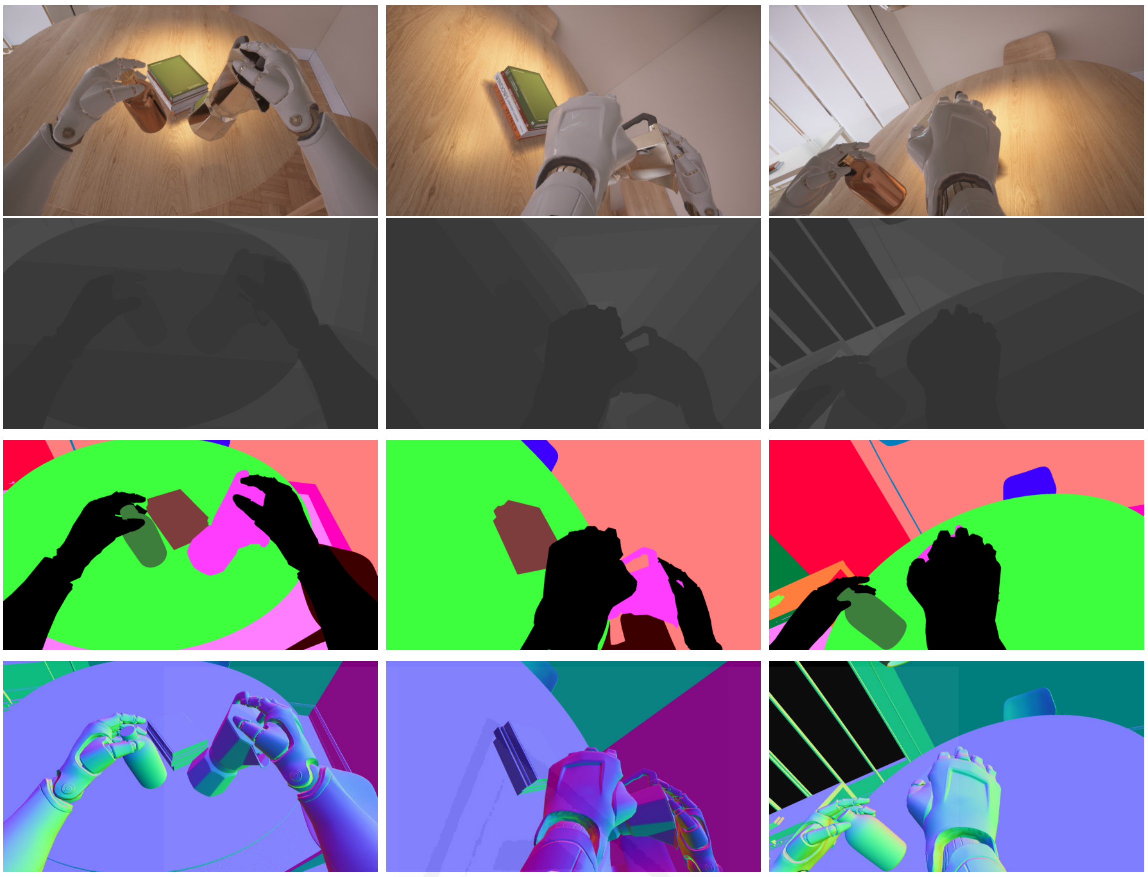


Figure 2.34: Sample hands interaction sequence and its associated data (from top to bottom: RGB, depth, instance masks, and normals).

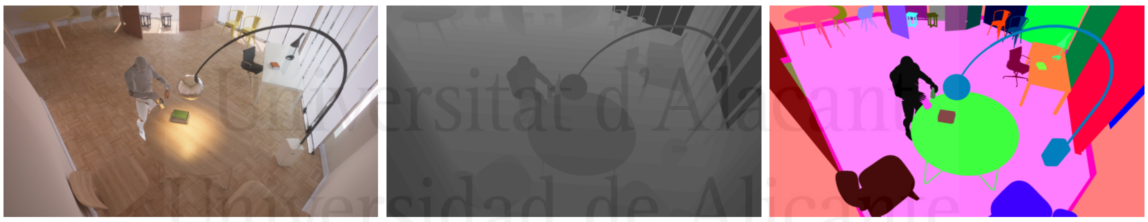


Figure 2.35: Sample data from an external point of view in the room with the corresponding images (RGB, depth, and instance masks).

thermore, since the movements are being performed by human agents who carry out smoother motions and obstacle avoid in a more natural way, the environment presents an interesting opportunity for learning such natural movements from demonstration.

As we can observe, UnrealROX is able to generate data for a significantly wide range of robotic vision applications. Most of them orbit around indoor robotics, although some of them might as well be applied to outdoor situations. In general, their purpose can be grouped into the more general application of AAL due to the inherent goal of achieving a robotic system able to operate intelligently in an indoor scenario in an autonomous way to provide support at various social tasks such as in-house rehabilitation, elder care, or even disabled assistance.

2.6 Experiments

In the previous section we showed multiple potential applications to which our data generator and the corresponding ground truth could be applied to train machine learning systems. In this section, we selected two of those applications to experiment with them in order to prove the effectiveness of our approach. Those two representative problems are: monocular depth estimation from RGB images and 6D object pose estimation.

2.6.1 Monocular Depth Estimation

As we already mentioned, estimating depth from 2D RGB images is an useful technique for many other higher-level applications such as scene reconstruction, object detection, and semantic segmentation. The problem can be formulated as follows: given a coloured RGB image from any camera, the goal is to predict a dense depth map for each pixel as accurately as possible [Bho19].

The current trend for monocular depth estimation takes advantage of deep architectures, more concretely deep CNN, with or without additional post-processing techniques for further refinement [Eig14] [Eig15] [Xu18]. Arguably, one of the most successful architecture is the Fully Convolutional Residual Network proposed by Laina *et al.* [Lai16]. To prove the usefulness of our simulator, we have trained Laina's method using a set of samples coming from our simulator (Figure 2.36 shows a random subset of the training images) and then we have tested it on a real-world dataset such as NYUDv2 [Nat12] (Figure 2.37 shows a random subset of testing samples for qualitative visualization).

As shown in this qualitative evaluation, knowledge learned using our simulated data can be seamlessly transferred to real-world data with adequate results in terms of accuracy and mean error.

2.6.2 6D Object Pose Estimation

Another widely used technique for which data generated with our tool can be helpful is 6D pose estimation of objects from 2D RGB images. This approach takes the object location problem one step further since it infers 3D rotation of the detected objects besides its location in an image (traditionally represented with a 2D bounding box). As a result, this estimation gives back a 3D bounding box that will estimate both 3D location (centroid) and rotation of the object.

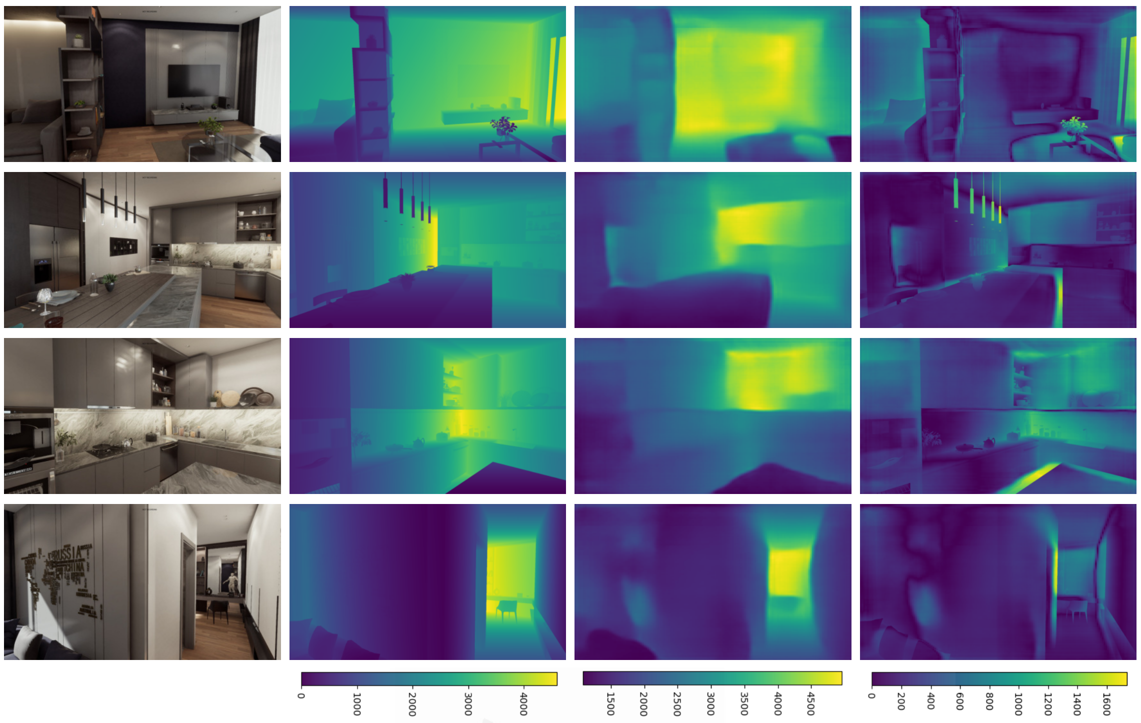


Figure 2.36: Qualitative visualization of Fully Convolutional Residual Networks for monocular depth estimation on data generated by our simulator. First column shows the RGB images, second column is the depth ground truth, third column shows the corresponding depth predictions, and the last column is the error map between the predicted and the ground truth depth.

This estimation was usually done through multi-stage algorithms that generated a coarse initial estimation that needed to be refined later. However, newer approaches like the one from Tekin *et al.* [Tek18] generate fine estimations which are accurate enough without requiring multiples stages thus making it possible to perform 6D object pose estimation in real time. It is inspired by the YOLO network for semantic segmentation [Red16] [Red17] to estimate projected 3D bounding boxes that, later, will be converted to a 6D pose by leveraging PnP (Perspective-n-Point, perspective from N points) algorithms.

To prove the usefulness of our generator in this problem, the network by Tekin *et al.* [Tek18] has been trained with our simulated data, and then tested with both synthetic and real images in order to see if it can transfer the knowledge to real-world data. First of all, in Figure 2.38 we can observe the pose estimation (through its 3D bounding box) of a banana in a sequence of synthetic images from our simulator.

Later, Figure 2.39 shows the same previously trained network trying to estimate the pose of a real banana on a live sequence captured by a camera.

As shown in this evaluation, it follows the same trend as the depth estimation experiments: knowledge learned from our synthetic data generator can be transferred to real-world data with success.

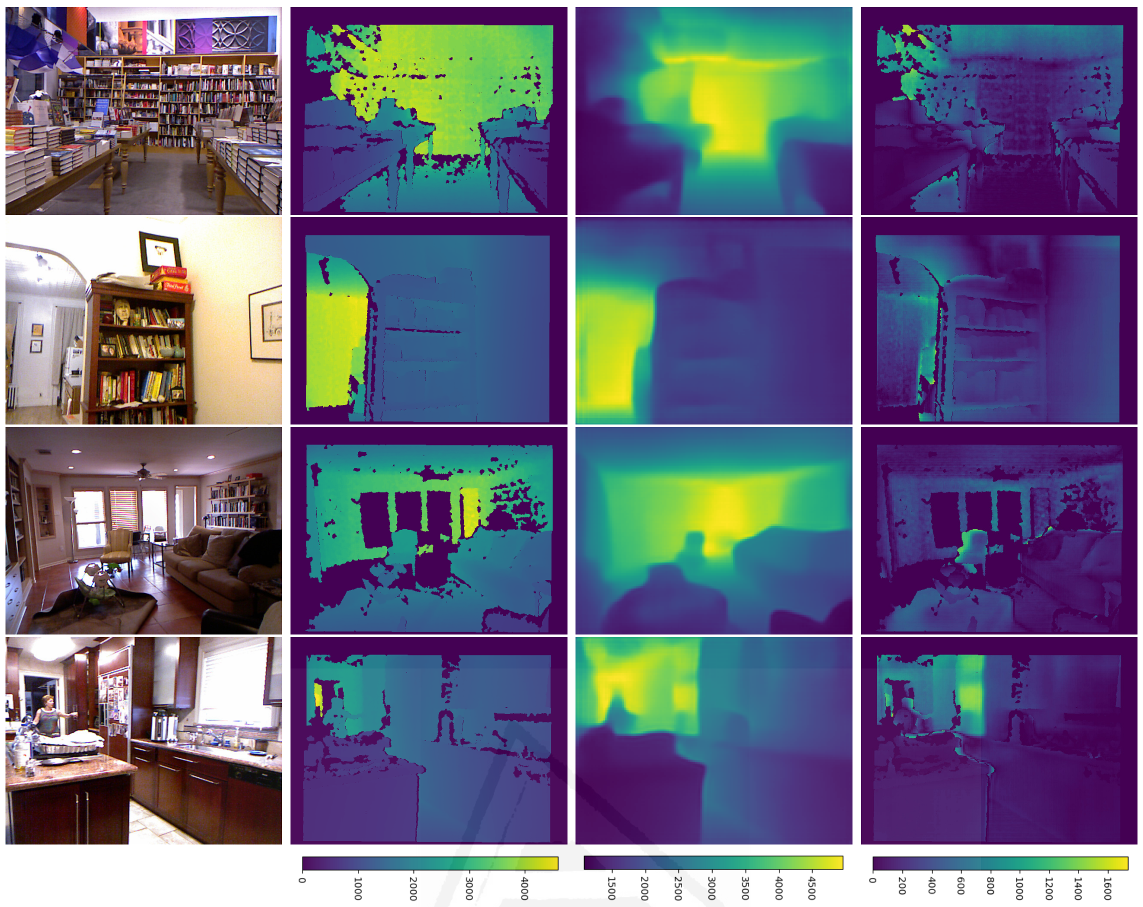


Figure 2.37: Qualitative evaluation of Fully Convolutional Residual Networks for monocular depth estimation on test data coming from NYUDv2 dataset [Nat12]. First column shows the RGB images, second column is the depth ground truth, third column shows the corresponding depth predictions, and the last column is the error map between the predicted and the ground truth depth.

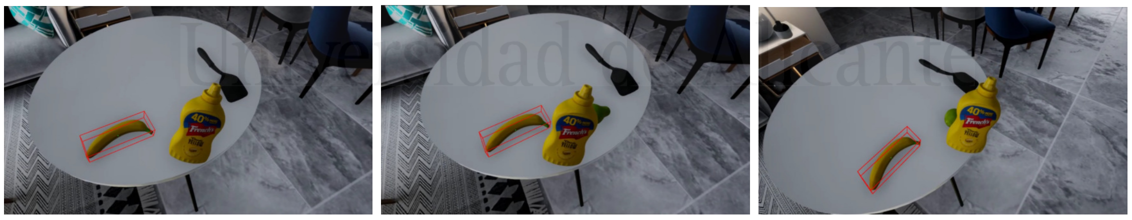


Figure 2.38: Qualitative evaluation of 6D pose estimation over synthetic data with single shot 6D object pose network [Tek18] trained with synthetic data from our simulator.

2.7 Preliminary conclusions and limitations

So far, this chapter presented a virtual reality system, in which a human operator is embodied as a robotic agent using VR setups such as Oculus Rift, Oculus Quest or HTC Vive Pro, for generating automatically annotated synthetic data for various robotic vision tasks. This environment leverages photorealism for bridging the reality

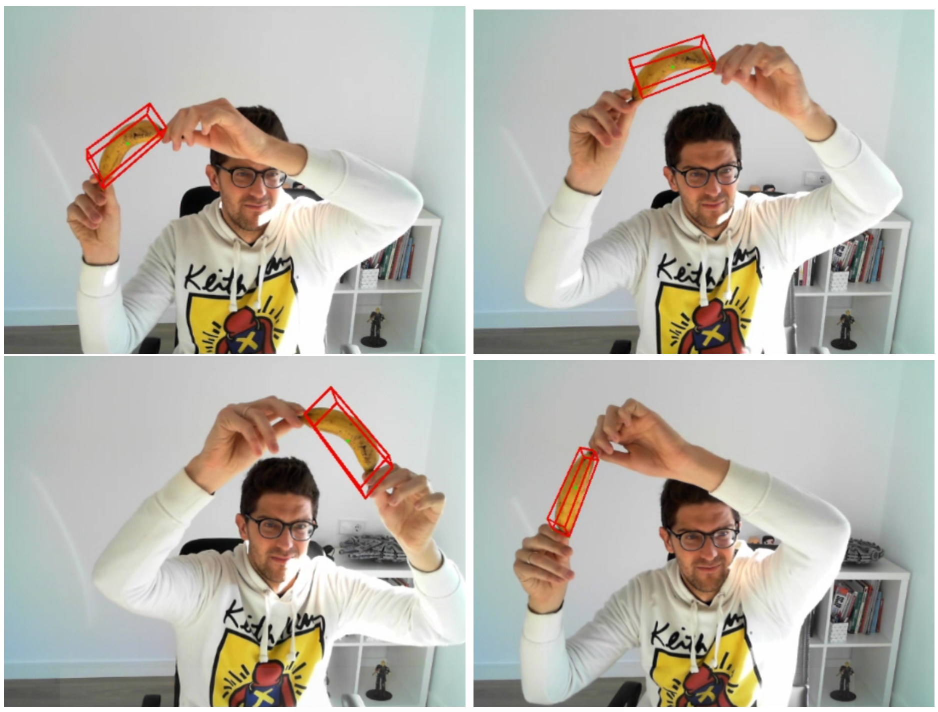


Figure 2.39: Qualitative evaluation of 6D pose estimation over real data with single shot 6D object pose network [Tek18] trained with synthetic data from our simulator.

gap so that models trained on its simulated data can be transferred to a real-world domain while still generalizing properly. The whole project, with all the aforementioned components (recording/playback, multi-camera, HUD, controller, and robotic pawns) is freely available[22] with an open-source license and detailed documentation so that any researcher can use to generate custom data or even extend it to suit their particular needs. That data generation process was engineered and designed with efficiency and easiness in mind and it outperforms other existing solutions at object, robot, and camera repositioning, and image generation.

The outcome of this work demonstrates the potential of using **embodied agents in** VR for simulating robotic interactions and generating synthetic data that facilitates training data-driven methods for various applications such as semantic segmentation, depth estimation, or object recognition.

Nevertheless, the environment still has certain limitations that must be addressed in order to make it applicable to a wider range of robotic vision tasks. The most important one: the system was designed only for retrieving image sequences from the virtual environment, so the recording-playback pipeline is the only way to go. Many more case scenarios can leverage UnrealROX features with a more flexible workflow.

Among the other limitations we found the simulation of non-rigid objects and deformations when grasping such kind of objects. We have limited ourselves to

[22]https://github.com/3dperceptionlab/unrealrox

manipulate non-deformable objects in order not to affect realism, since this is a different approach with a non-haptic manipulation and deformations need to be modelled at the object level. The system would need to implement a custom physics engine in order to model the transformations on deformable objects and be able to record and playback them. Another important shortcoming is the absence of tactile information when grasping objects. We plan to include simulated tactile sensors to provide force data when fingers collide with objects and grasp them instead of providing only visual information. Furthermore, although not strictly a limitation, we are working on making the system able to process URDF to automatically import robots, including their constraints, kinematics, and colliders, in the environment instead of doing that manually for each robot model.

2.8 Extension

In 2018, the RobotriX dataset[Gar18a] was presented as a synthetically generated multi-purpose dataset for robotics. The main contribution of this dataset was the quality and variety of the data that it presented, including RGB images, depth and normal maps, and instance and semantic segmentation masks at 1080p resolution and 60 FPS. The tool that was developed to generate it, UnrealROX [Mar19], is based on UE4 (Unreal Engine 4), one of the most widespread video game engines and, therefore, it is one of the most cutting-edge and evolving platforms for realistic 3D rendering. As we have said, the tool was developed specifically for generating that robotic-oriented dataset, and due to that, it was not engineered for getting data in some other ways that could be useful for different purposes, such us reinforcement learning, or simply for random or scripted data gathering. In this way, in order to make UnrealROX a truly data generator for a wider range of applications, we decoupled the main functionalities from it, so that it would be easy to isolate the data acquisition tool, and to add any custom behaviour that any developer could need for generating its own data. In this part of the thesis, we present this redesigned and more flexible workflow, and all the new functionalities that we added to the tool, as well as several qualitative experiments that prove the usefulness of synthetic data from UE4 in a variety of vision-based deep learning architectures.

2.8.1 Overview

The original version of UnrealROX focuses on simulating a wide range of common indoor robot actions, both in terms of poses and object interactions, by leveraging a human operator to generate plausible trajectories and grasps in virtual reality. Now, we propose to take advantage of all this data-generation potential, making it easier to use it for a wider variety of goals. UE4 (Unreal Engine 4) is a widespread platform in continuous development, and any new hardware, such as a virtual or augmented reality headsets, motion capture devices, or similar, will presumably provide support for it. So, we find highly interesting to have the possibility of generating any kind of data, in a fast, easy and customizable way, with this versatile and powerful engine.

Along with the generation of raw data (RGB-D/3D/Stereo) and ground truth (2D/3D class and instance segmentation, 6D poses, and 2D/3D bounding boxes), we now aim to provide albedo, shading maps, interaction information, and a Python API. Moreover, the pose of virtual agents with skeleton (human meshes, robots or hands, for example) can also be projected over RGB or mask images, since joint 6D pose is provided in a frame-by-frame basis.

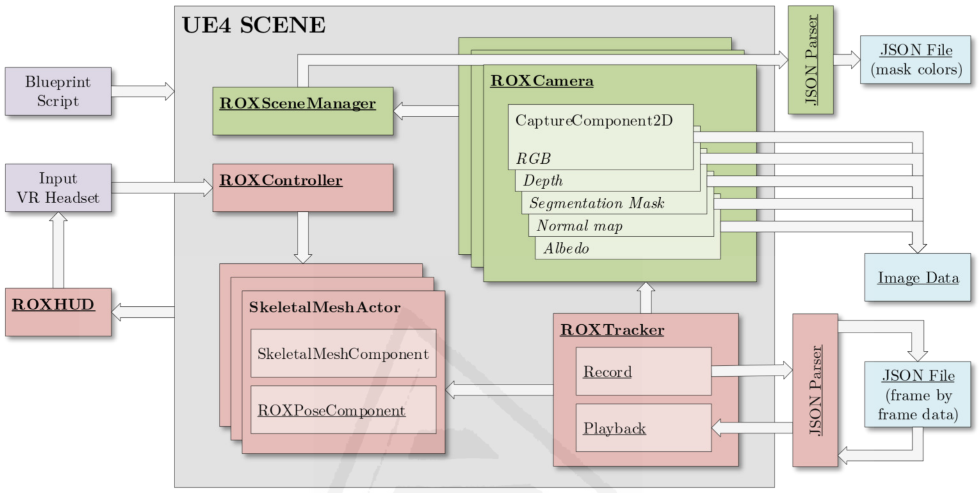


Figure 2.40: UnrealROX+ system diagram. In green, decoupled data acquiring system that can be used with more flexibility. In red, adapted logic from UnrealROX for using new decoupled data acquiring system, not essential for generating data. Custom classes (actors or components) are underlined.

As we already stated when we presented the original version of our tool in Section 2.4, the rendering engine we chose to generate photorealistic RGB images was UE4. A big advantage of this decision is its constant development. An example of how we can leverage this evolution is the introduction of real-time ray tracing rendering as beta feature in Unreal Engine's 4.22 version on April 2019 [Can], and nowadays it is completely available [Cow]. This allowed us to generate ray-tracing-rendered images, as the one showed in Figure 2.41, both in real time and offline, without modifying the tool.

UnrealROX was initially developed as a tool for generating synthetic data from virtual photorealistic 3D environments, as we already know. However, its workflow was totally focused on immersing a human agent in these environments through virtual reality, *recording* all its movements and interactions, and later *rebuilding* the sequence frame by frame in order to have the time needed to retrieve all kind of image data, at high frame rates, high resolutions, and high realism. After releasing RobotriX [Gar18a] and UnrealROX, we continued using the tool for generating data for different kinds of problems, and several modifications were introduced to adapt

Figure 2.41: Real time ray-tracing snapshot on UE4 from the Archviz Interior Rendering sample project.

its workflow. We end up deciding to decouple the data acquisition feature of the tool from the main workflow that was used for generating RobotriX (*record*, *rebuild* and *acquire*). In this way, the tool now allows to acquire image and other data from the virtual environment very easily, directly from the UE4 visual scripting language, which is also known as *blueprints*. That means that preparing scripts for generating images of objects from different point of views, or moving a camera (randomly or not) across the scene is truly fast. We also kept the possibility of generating data with the original workflow, but adapting the code for calling the new decoupled data acquiring system.

The other big improvement for the tool was packaging it as a Unreal Engine plug-in, which is remarkable because it allows to incorporate all these interesting features to any existing project very easily. These two main changes in the tool implied many other little ones that we are going to review in this chapter, as well as other additions that the tool received during this period. For example, along with RGB, depth maps, normal maps, and instance segmentation masks, we also provide albedo directly retrieved from UE4, and its corresponding shading maps that can be post processed later. We can see the new system diagram for UnrealROX+ in Figure 2.40. The tool itself, as well as more extensive documentation is available as open-source software[23].

2.8.2 Data acquiring subsystem

The motivation for this tool revision was decoupling the ground-truth image acquisition in order to make it faster and more flexible to use for any scenario. We decided to encapsulate all these functionalities under a custom class inheriting from the basic Unreal class *Camera*, that we called *ROXCamera*. In this way, each *ROXCamera* in the scene has the possibility to be configured and requested to retrieve some concrete data, separately. Automating this process for more than one camera is as simple as keeping a data structure with their references and using them when

[23]https://github.com/3dperceptionlab/unrealrox-plus

needed. Moreover, these cameras can be referenced and called from Blueprints very easily.

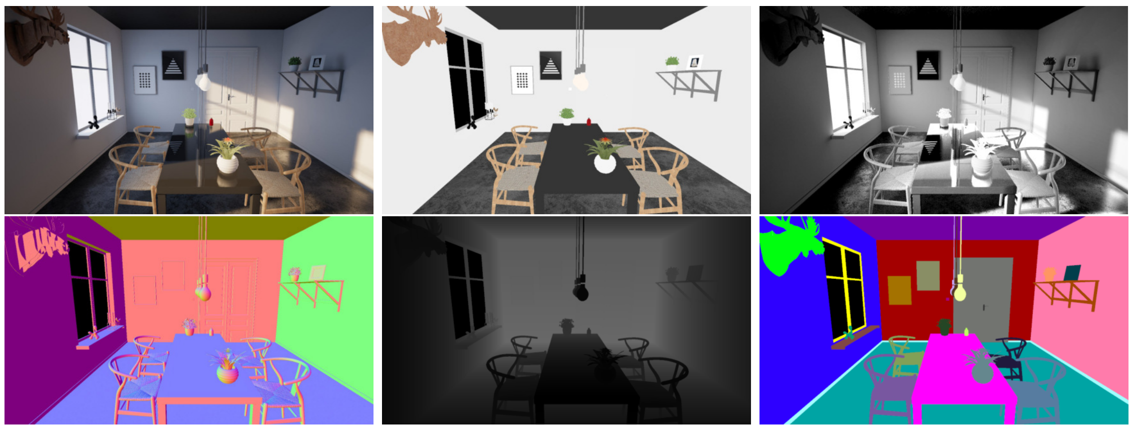


Figure 2.42: UnrealROX+ image data. From left to right and top down: RGB, albedo, shading map, normal map, depth map and instance segmentation map. Shading map is not generated directly from UE4, it is computed later from RGB and albedo.

This encapsulation simplifies usage, but we also reinterpreted the data-acquiring logic itself to make it more elegant. Original UnrealROX retrieved all image data directly from the main viewport, which is the point of view that is being rendered in UE4 user's screen. Using main viewport when generating data from several cameras implies continuously switching the viewport source camera, as well as the rendering mode of the whole scene (for generating RGB, depth, normal maps, etc). This strategy was rough, especially knowing that UE4 provides a dedicated and flexible entity for capturing image data from the scene without having to use a standard camera and the viewport: the *SceneCapture2D* actor, or more precisely, the *CaptureComponent2D*, which is the component that contains its functionality. It can be used for rendering (with its own custom render mode) a concrete point of view to a *RenderTarget*, instead of using the viewport. *RenderTarget* is another entity, which encapsulates the data that is rendered by *CaptureComponent2D* and can dump it to a *Texture*, or to a file, for example.

Our *ROXCamera* class is designed to create one *CaptureComponent2D* entity for each type of data that must be generated from that point of view, and everything is configured automatically. As we have one *CaptureComponent2D* for each render mode, we don't have to worry about changing the general render mode. The file saving logic can be easily managed from the *RenderTarget*, where we can select the colour codification or the file format (which differs among different image data).

2.8.3 Segmentation masks

Semantic segmentation masks are one of the most expensive data to provide in any real dataset. For any real-world image, manually discriminating at pixel level for generating a reliable mask (for semantic or instance segmentation) is a very tedious task. So, obtaining this kind of information in a pixel-perfect and automatic way is one of the most relevant reasons to use synthetic generators for creating huge image datasets.

In our first version of UnrealROX, inspired by a preliminary implementation of UnrealCV [Qiu16], we modified the engine's source code (simply changing a flag) in order to be able to modify at will the vertex colour as a post process operation, and thus being able to render and capture these kind of images. The fact of recompiling the whole engine shouldn't be a problem, since UE4 is open source, but both the development and the use by other teams become harder and slower in this way, so we developed alternatives for generating these data. At the beginning of each generating execution, every mesh on the scene is gathered, and a reference to it is stored. Then, we developed two approaches:

- A plain-colour material instance is automatically created and associated with each of those meshes. A distinct colour is assigned to each mesh, in such way that, when rendering the scene in *base colour* mode (i.e. *unlit*, without lighting computations) with these new materials applied, we get pixel-perfect segmentation masks. Obviously, we have to keep a data structure with references to both original and plain-colour materials in order to be able to swap between these two rendering modes at any time.
- A *post process material* is applied depending on the value stored at the *custom stencil buffer*. This is a special GPU buffer which we have the possibility to modify. For each mesh in the scene, we can assign a custom stencil value, resulting in the buffer storing that value for each pixel that belongs to that mesh. Then, it can be used in a post process material, which act as shaders in UE4, performing operations at pixel level. Post process materials can be applied directly to a *CaptureComponent2D* entity, which is great for our workflow.

Acquiring and storing these mask images is a delicate task, since their stored pixel values must match exactly the ones that we set numerically in the engine. In order to avoid little colour variances inside the same region, tone mapping is disabled (or at least it is applied before the post process), and linear colour read from *RenderTarget* must be transformed to *sRGB* colour space with gamma equal to 1.

It is worth mentioning that the plain-colour material approach has a remarkable disadvantage. Swapping materials for every mesh on the scene is way more computationally demanding than the *render mode* switch that we had in the first UnrealROX version, or the post process material alternative. Nevertheless, we still consider that it is viable option because we can still generate segmentation masks separately from the rest of ground truth information (as long as we are generating data offline). In this way, we will only have to perform the material swap once at the beginning of the generation, and once more at the end. On the other hand, the post process material approach has the limitation of just being able to codify stencil 256 different values, so it only should be used if the the scene has less than that number of different meshes.

2.8.4 Reflectance and Shading maps

Reflectance is another image data which is interesting for several problems, such as Intrinsic Image Decomposition [Bel14] or Inverse rendering [Yu19]. It represents the inherent colour from surfaces, the colour they naturally reflect, without any lightning influence or computation (shadows, specularity, etc). In computer graphics research area is common to use the term *albedo*, which technically represents the percentage of light that is reflected by a surface [Coa03], but can be used as well as *reflectance* for referring this kind of images with only the diffuse reflection component from an illumination model [Sat97]. We will use reflectance or albedo indistinctly. This data can be directly obtained in UE4 through the render mode *base colour*, that can be chosen in *CaptureComponent2D* structure. As we already verified when working on generating segmentation masks, pixel colour information obtained through images captured in base colour are perfectly accurate when retrieved as described.

Shading maps, on the other side, provide us information about lightning for each pixel, mainly how much it is darkened or lightened starting from the base colour provided by reflectance. Image decomposition is defined as follows [Bel14]:

$$I = R \odot S$$

being I the composed image, R the reflectance layer, and S the shading layer. So I is defined as the element-wise multiplication ($\odot$) between reflectance and shading. In this case, we don't acquire this data directly from the engine. Instead, we compute it from RGB and albedo images. Starting from the equation defined previously, we can infer that:

$$S = I \oslash (R + \epsilon)$$

which means that shading map can be computed as the matrix element-wise division ($\oslash$) between the image and its correspondent albedo. Since a division must be performed, we can avoid dividing by zero adding a very little value (ϵ) to albedo pixel values. For the vast majority of scenarios, white lighting can be assumed, so shading maps are widely represented as greyscale. That means that light affects to all three RGB channels equally. An example of albedo and shading map images, along with its correspondent RGB and other images, can be seen at Figure 2.42.

2.8.5 Skeletal Meshes

One of the key parts in the development of the original UnrealROX was storing every joint (bone) 6D transformation of the controlled pawn, which is essentially an skeleton (i.e. *SkeletalMeshActor* in UE4). We needed to keep all that information frame by frame in order to later be able to retrieve the exact pose of the skeleton for acquiring data offline. This goal was reached by creating our own class (*ROXBasePawn*) that inherits from the basic *Pawn* class, and adding to it all the logic needed to save and later recover joint transformations. However, this approach had a huge disadvantage: it was mandatory to use our *ROXBasePawn* class in order to be able to use the offline data-acquiring workflow with an skeletal mesh. This may imply migrate code and assets from an already existing class, to ours. The alternative we developed to better align this with UE4 programming standards and the fact of delivering a plug-in, consisted of encapsulating this joint-retrieving logic as a component, so that it can be added to any existing actor with a *SkeletalMeshComponent*.

2.8.6 Python communication for Reinforcement Learning

One of the biggest future works mentioned on the original UnrealROX work was a *Python API* in order to be able to modify Unreal Engine's scene from deep learning frameworks. This workflow is specially useful for reinforcement learning models, since they are systems that successively improve the actions they take in an environment by maximizing a reward function. So, these models need to modify the scene, see what happens (retrieving data from it) and check how the reward function behaves. In order to make our tool useful for these applications, and therefore, make UE4 usable for reinforcement learning, we developed several commands that can be thrown from Python programming language, and caught by Unreal Engine, performing the correspondent actions in the virtual scene. The system is defined as a client-server architecture, where Python API acts as client, and our UE4 plug-in as server, and it works both locally and remotely. We defined the following groups of commands (further documentation is available at GitHub repository):

- **Lists**: In order to interact with actors and meshes in the scene, we must know which they are, so we implemented commands to list them depending of their type, such as `actor_list`, `object_list`, `camera_list`, `skeletal_list`, among others.
- **Transformations**: To apply transformations over actors in the scene we have `move`, `rotate` and `scale`, as well as their correspondent commands for retrieving the current location, rotation and scale.
- **Cameras**: We can, for example, spawn and orientate cameras with `spawn_camera` and `camera_look_at`, respectively.
- **Data acquiring**: We can acquire image data from a concrete camera with

get_rgb, get_depth, get_normal, get_instance_mask and get_albedo. Other commands can configure this operations, and also acquire other kind of data, such as bounding boxes with get_3d_bounding_box.

2.8.7 Grasping subsystem and interaction information

The grasping subsystem that we used for the robot-object interactions in RobotriX, which was included on the original version of UnrealROX, was completely decoupled and evolved on its own project with *UnrealGrasp* [Opr19], which is still being developed in parallel. It will also be released as a plug-in, completely compatible with UnrealROX+. This also decoupled interaction information that was added to the rest of frame-by-frame data stored for movable *StaticMeshActor* actors. Data gathered for each frame was: if that object was being touched (overlapped), grasped, or none, by the controlled actor. We left this information to be given separately by UnrealGrasp, so UnrealROX+ just provide a list of overlapping actors. This is, for each actor, the ones that are being overlapped in that concrete frame, not distinguishing static from skeletal meshes.

Contact points extraction

Additionally, we worked on generating some kind of heat map showing the contact points that a hand (or a robot clamp) are most likely to use for getting a stable grasping. This data can help a robotic system to identify a suitable grasp given an object. These heat maps are generated after repeating several times suitable grasps over the same object.

In the same way as previous interaction information, we developed this to work with the grasping system that we decoupled, so it was left out from the base code of UnrealROX+. We developed it using a material in the interacted object that detects and projects on itself the overlapping section between the "grasp" actor and the own object. Then, this material is exported as a texture, and stored as an image. This aspect will be detailed in Chapter 3, where a full work regarding the generation of a heat map dataset is developed.

2.9 New applications

UnrealROX+ provides a great variety of data. The set of tasks and problems that can be addressed using such data ranges from low to high-level ones. Some of the most relevant low-level tasks include:

- Depth Estimation: Both monocular (deep learning models estimating depth from RGB) and stereo (estimate 3D information from a pair of displaced cameras) are enabled by our system. Stereo may be reached manually by placing two cameras, but we implemented it natively so that it was more comfortable to work with camera pairs.

- Object detection and pose estimation: We provide rich information about objects in the scene (instance and class segmentation masks, 2D and 3D bounding boxes, 6D position) to work in these problems.
- Instance/Class segmentation: Instance masks are directly provided, and they can be post-processed for enabling class segmentation as well.
- Normal estimation: Estimating the normal map of a given surface is an important previous step for many other tasks. For instance, certain algorithms require normal information in a point cloud to extract possible grasping points. UnrealROX+ provides per-pixel normal information.
- Intrinsic image decomposition: Image decomposition in its intrinsic parts, reflectance and shading.

That low-level data enables other higher-level tasks that either make use of the output of those systems, or take the low-level data as input, or both:

- Hand pose estimation: UnrealROX+ provides the 6D pose of every skeleton joint, so hand pose can be estimated. It is useful for gesture detection, for example.
- Human pose estimation: Again, 6D pose of every skeleton joint is provided, so skeleton pose estimation can be trained with data generated by our tool.
- Obstacle avoidance and navigation: By leveraging various types of low-level information such as RGB images, depth maps, bounding boxes, and semantic segmentation, robots can learn to avoid obstacles (by detecting objects and estimating their distance) and even navigate in indoor environments.

Data generated by UnrealROX+ can be useful for other applications, although in this case they may need further development or combination with other tools:

- Reinforcement learning: Python API enables using the tool for generating data on demand directly from a reinforcement learning model in training process.
- Visual grasping: In combination with *UnrealGrasp*, agents in the virtual scene can perform plausible grasps and record RGB and hand pose data.
- Action detection and prediction: RGB and skeleton pose data are provided directly, and further labelling for actions at multiple granularity levels can be made afterwards with proper tools. Even some kind of automatic labelling can be developed if pre-designed animations are being used for performing those actions.

2.10 New experiments

In the previous section we showed multiple potential applications to which our data generator and the corresponding ground truth could be applied to train machine learning systems. Besides tasks such us depth estimation and visual grasping, which relied on data generated via the previous workflow [Mar19], we selected two more applications in which the new UnrealROX+ is exploited. We show the usefulness of the generated data in the aforementioned tasks, complementing the already available datasets from the real domain.

2.10.1 Hand joint position estimation

3D hand joint estimation is a computer vision challenge that have been addressed through deep learning models in the last years [Asa17]. However, datasets that provide hand pose or joints precise position in 3D are not very common and extensive. The problem consists of estimating hand joint positions in 3D from an RGB image, so data such as segmentation masks, depth (to compute point clouds) and skeleton pose are needed. In this context, UnrealROX+ is pretty useful for generating this kind of data easily from UE4. That has been done in [Cas19], where a hand pose dataset has been developed using an early version of UnrealROX+ (see Figure 2.43). Moreover, a graph convolutional network fed with point cloud data, GraphHands, is presented to validate the generated data.

2.10.2 6D pose estimation

Another widely used technique for which data generated with our tool can be helpful is 6D pose estimation of objects from 2D RGB images. This approach takes the object location problem one step further since it infers 3D rotation of the detected objects besides its location in an image (traditionally represented with a 2D bounding box). As a result, this estimation gives back a 3D bounding box that will estimate both 3D location (centroid) and rotation of the object.

Pix2Pose [Par19] is a recent model that addresses this problem. It implements an auto-encoder architecture designed to estimate 3D position of each pixel from a 2D RGB image. In addition, it includes some interesting particularities, such its robustness to occlusions by leveraging recent achievements in generative adversarial training to precisely recover occluded parts. To prove the usefulness of our generator in this problem, we trained Pix2Pose with our simulated data for single object pose estimation. The objects chosen for the experimentation have been taken from YCB model set [Cal15], since it provides both real objects, that can be ordered for physically having them, and high-precise registered models from that objects, which can be imported to UE4. The real data we are going to use is the one provided from

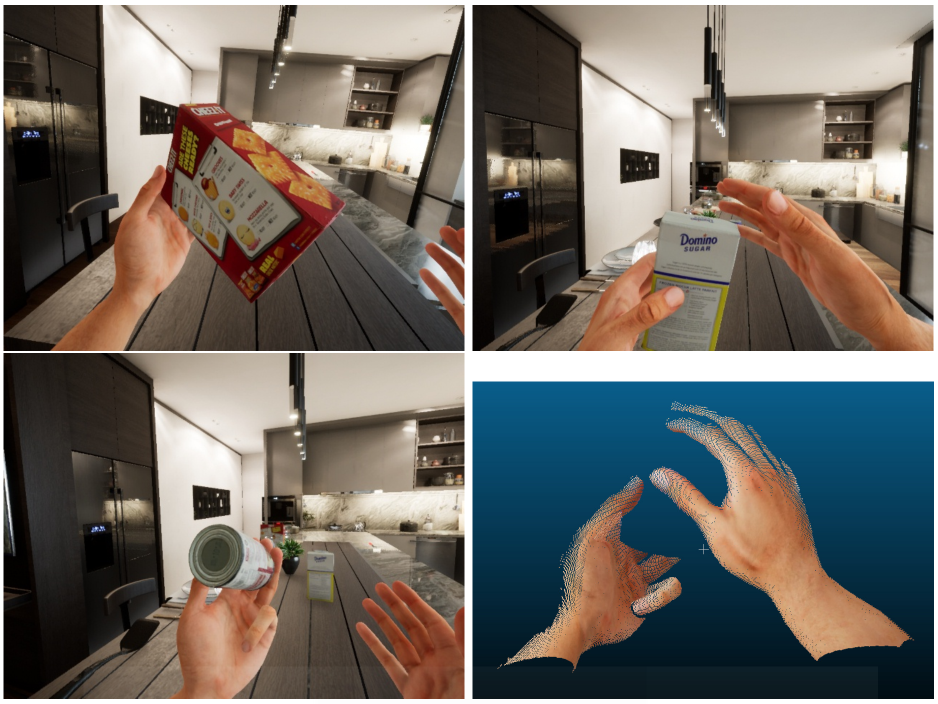


Figure 2.43: This figure shows a subset of samples images with 3 of the objects available on the UnrealROX-generated dataset presented in [Cas19], as well as a point-cloud example. Point clouds are computed from the dataset and used for feeding GraphHands.

the YCBv dataset, from the BOP 6D pose Benchmark [Hod18], which contains image sequences with real objects from YCB model set. So, we can generate fully annotated synthetic data to train the network, and real data to test it. Some qualitative results can be appreciated in Figure 2.44, where we can see how 3D bounding boxes for this two objects are pretty well estimated over real data.

2.11 Final conclusions and limitations

Most of the limitations and future works previously presented in Section 2.7 have been addressed in this work, and many others are left outside for having decoupled grasping from the data-acquiring tool. One unsolved limitation is the possibility of managing non-rigid objects and deformations offline, i.e. being able to recreate these behaviours in a frame-by-frame basis for acquiring data at a slower and more precise regime. Nevertheless, it remained slightly out of scope since it would apply to deformations when interacting or grasping objects, or to pure physics simulations, aspects we rely on UE4 or on any other physics engine which works with it. We also didn't address possibility of processing URDF files to automatically import robots, including their constraints, kinematics, and colliders, in the environment instead of doing that manually for each robot model.

Another possible improvement is providing an additional segmentation layer for

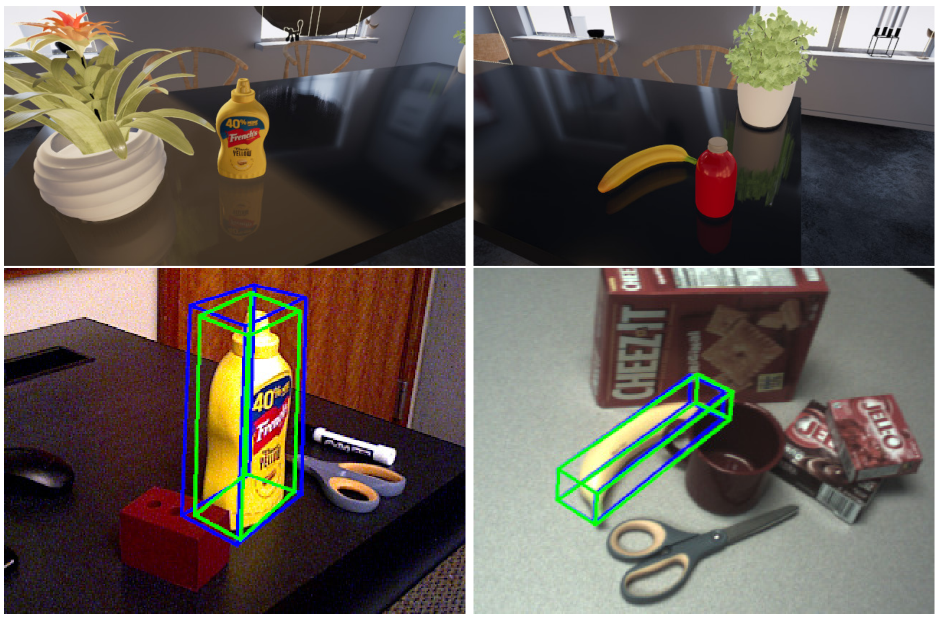


Figure 2.44: Above, examples of training synthetic data with two different objects generated with UnrealROX+. Below, qualitative evaluation of 6D pose estimation with Pix2Pose [Par19] over real data from YCBv dataset [Hod18] (in blue estimated pose, in green ground truth).

objects behind transparent materials. Camera distortion and noise are aspects that may distinguish real from synthetic images. They can be addressed as a post-process, but it could be interesting to parametrize in the tool itself. Other phenomenons, such as camera shake, very common on robotic applications, are not provided or guided by our tool, although they could be simulated through UE4 programming. Talking about new functionalities, the presented Python API for Reinforcement Learning is somewhat preliminary and many more commands and more efficient or comfortable ways of working may be added. Following the OpenAI Gym interface would be ideal to make it compatible with community standards.

Lastly, the main future work is definitely the improved and encapsulated version of *UnrealGrasp*, including detailed interaction information. We will further discuss this in Chapter 3.

At the end, this chapter presented major improvements on the original UnrealROX, our tool for generating synthetic data from realistic 3D environments in Unreal Engine 4. It already demonstrated its potential for simulating robotic interactions and generating huge amounts of data that facilitates training data-driven methods for various applications such as semantic segmentation, depth estimation, or object recognition. However, it lacked from the flexibility and modularity for making the tool profitable on a wider range of scenarios, and for a bigger number of researches. So, we worked on decoupling the main data-retrieval system from the main workflow in order to make it much easier and faster to set up and script custom behaviours with Unreal Engine's visual scripting language. Moreover, new kinds of data, such as albedo or shading maps, or more efficient ways of acquire and managing data, such as segmentation masks or skeleton pose retrieval, were also added, along with useful features such as the Python API, useful for Reinforcement Learning.

All UnrealROX original features (multi-camera, bounding boxes, ...), and also the original workflow that was used for generating RobotriX, are included or migrated to this new version of the tool, this time as an easy-to-use Unreal Engine plug-in. Grasping system, that has been moved to a separate project that will have its own plug-in, which will be completely compatible with UnrealROX+.

Chapter

3

Generating Synthetic Hand-object Contact Maps for Grasping Region Prediction

From a human point of view, adapting our grasping to different shaped and sized objects feels natural, since we are continuously learning from environment interaction. We have mastered object manipulation, grabbing a hammer by its handle is all but unconscious. However, performing the same task from a machine's perspective is particularly challenging, as it requires an in-depth understanding of the physical properties of objects (e.g. weights, geometry, texture). We could approximate such an understanding by capturing the hand-object contact resulting from human grasps, i.e., contact maps. This has already been done using cumbersome capture systems, and not on a large scale. In this chapter, we simplify and accelerate such a process by generating contact maps from our interaction with household objects in photorealistic virtual reality environments. To the best of our knowledge, this is the first attempt to generate contact maps at scale from natural interactions in a virtual scenario. We train an image-to-image translation method to predict grasp regions on objects, demonstrating the usefulness of our generated contact maps. Our purpose with this work is to foster applications where hand-object understanding is necessary, so all the developed tools, code and generated datasets are provided.

3.1 Introduction

We know how to naturally grasp previously unseen objects from our experience interacting with similar objects in shape, size, and texture. However, object grasping and interaction is not straightforward from a machine's perspective. Robots rely on on-board cameras and sensors to model the environment, identify objects and extract information from their appearance and shape. In the last decade, computer vision has undergone a massive shift due to the rise of machine learning. Classical vision algorithms have been outperformed by deep learning-based models in addressing problems such as semantic segmentation and object detection. Identifying grasping regions on an object is another related problem where we can take advantage of machine learning.

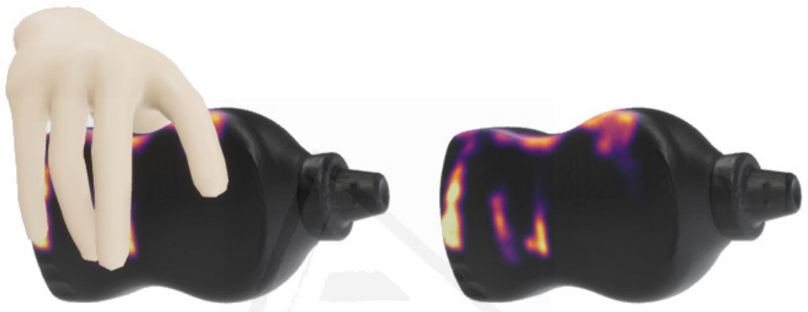


Figure 3.1: At left, a virtual hand grabbing the mustard bottle from YCB object dataset; at right, a generated contact map on the mustard bottle.

There are many tasks where identifying an object and determine how to grasp it is useful for an autonomous robot: assistance, industry, automation, and space exploration, among others. The end effector typically used in these systems is a grip, way simpler than a human hand. However, in some scenarios predicting human grasps on objects can be interesting for robots: holding and offering an object for a human to grasp it, supervision, or just because the robot has a hand-like end effector. This last case scenario is specially useful for robots that operate in the same environment as humans, such as social robots for assistance.

For addressing this computer vision problem from a deep learning approach, a lot of data will be needed. In this case, if we want a system to be able to recognize object regions where a stable grasp can be performed, we will need to label those areas in a big amount of objects to generate the required training data. This task can be done in several ways, such as thermal camera data collection after grasping objects, or manually arrange a virtual hand over an object on a simulator. This last approach generates the grasping region synthetically, but it still requires a lot of work and criteria to define the proper hand pose for each grasp. Synthetic data can be useful in other ways in this context to minimize the requirements, the set up, and the work needed to label grasping regions in big amounts of objects. And that is the proposal for this work, leveraging VR and simulated interactions to that end.

Purpose: Training a learning-based approach to predict grasp areas on objects requires huge amounts of labelled data. In this context, we use contact maps to describe hand-object interaction. Contact maps are obtained from the repeated interaction of a hand with a given object. This is a tedious process and usually requires specific hardware settings such as 3D-printed objects and thermal cameras [Bra19a]. Here, synthetic data generation plays an important role and accelerates this process. Modern rendering engines allow us to generate visually realistic images with full ground-truth annotations. By leveraging advances in synthetic-data generation and real-time hand tracking, we capture how we grasp household objects [Cal15] from our interaction in VR environments.

Our work: First, we modify an existing hand-object interaction system [Opr19] to use Oculus Quest's on-board hand tracking [Han20] instead of the handheld devices, to naturally grab objects in VR settings. This allows for a more natural object grasping leveraging bare hand interaction. Next, we adapt a framework for synthetic data generation [Mar19, Mar21], to capture heat maps, i.e., contact maps, representing the contact of the virtual hand with household objects from the YCB dataset [Cal15] (see Figure 3.1). Similarly to ContactDB [Bra19a], we finally use the resulting dataset to train an image-to-image translation model to predict contact maps from RGB images. We show the usefulness of our data to train such a model. We also developed tools for extending the YCB-Video dataset [Xia18b] with the synthetic grasps to make it useful for grasping prediction problems as well.

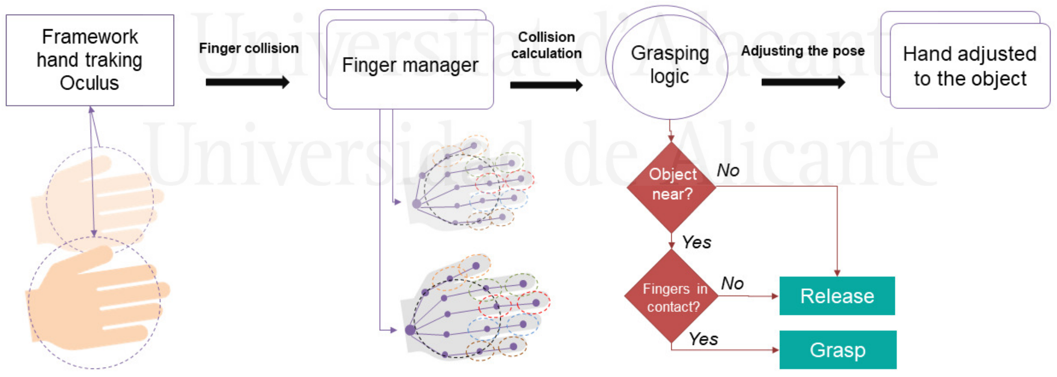


Figure 3.2: Grasping system pipeline. In the first place, we use the Oculus hand tracking framework to move the virtual hands. Second, the finger manager detects individual finger collision with the object we are manipulating. Then, the grasping logic decides whether to grab an object based on previously computed collisions. As a result, the virtual hand naturally fits the object geometry.

Contributions: First, we achieved a natural hand-object interaction in virtual environments, adapting UnrealGrasp [Opr19] to use Oculus Quest hand tracking instead of the hand held controllers. Second, from the VR interaction, we created a dataset made up of heat maps representing object grasps, and the hand and object pose which produced that heat map. This required additional development in our previously presented framework for synthetic data generation [Mar19, Mar21][*]. Furthermore, we trained an image-to-image translation model on our dataset to predict contact maps from RGB images of objects. We showed the predicted contact maps are consistent with object shape, also proving the usefulness of our data. Finally, the dataset, the model and the tool generated during the current study are available in the GitHub repository [†]. It also includes the scripts for extending YCB Video dataset.

This chapter is organized as follows. In Section 3.2 we analyse existing contact map datasets, and models for predicting grasping regions or hand poses. In Section 3.3 we present our grasping system and we describe how we compute and retrieve the contacted object surface as a heat map. Section 3.4 describes the procedure we followed to generate and organized our heat map dataset, SynContact. In Section 3.5 we describe how we leverage the generated data for training a Pix2Pix-based network, ContactNet, for predicting heat maps from RGB images. Finally, in Sections 3.6 and 3.7 we draw some conclusions about the work and we analyse its limitations, besides some future work proposals.

3.2 Related Works

In this section, we provide a brief introduction to contact/heat maps datasets, and models for predicting grasping regions or hand poses.

In the literature, we mostly distinguish two different grasping types: robotic (grip) and human-like (hand). In the same way, we can find very different datasets depending on the task. The standard dataset for robotic grasping is the Cornell Grasping Dataset, which consist of a set of objects labelled with plausible parallel jaw grips. The Jacquard dataset [Dep18] also contains that information, and its data was labelled through simulations. It is very common to label valid robotic grips through simulators. GraspIt! [Mil04] is probably the most famous one as it was originally proposed in 2004. It also simulates hand grasps.

Indeed, ObMan dataset [Has19] was created with the purpose of training a model for reconstructing hand and object meshes from images during interaction. It is a synthetic dataset, and body poses were recorded via motion capture, while hand poses were generated using GraspIt.

If we move to non-synthetic datasets, we can find GRAB [Tah20], a very complete dataset of full-body sequences interacting with daily objects, which includes heat

[*]https://github.com/3dperceptionlab/unrealrox-plus
[†]https://github.com/3dperceptionlab/unrealhandgrasp

maps and annotations of hand-object interactions. It served for training a model that generates hand poses to grasp unseen objects. Jiang *et al.* [Jia21] proposed an architecture for that same task. In this case, they divided the problem in: (1) generating a contact map given the model of an object, and (2) generating a hand pose given the object with the contact map.

Corona *et al.* [Cor20] proposed a model for predicting how a human would grab an object (i.e. the hand pose) given just an image. For that purpose, they used GraspIt to manually annotate the YCB objects with some realistic grasps, and they generated an enormous amount of grasping-annotated images just transferring those grasps to the frames of the YCB-Video dataset [Xia18b].

ContactDB [Bra19a] dataset contains grasping information recorded from thermal cameras after a human hand interacting with an object. They established two kinds of grasping purpose: use and hand-off. That subtle difference meant very different grasps in some objects, such as scissors or gamepad. The same authors presented an algorithm [Bra19b] to generate grasps given an object model and its contact map. It consists of refining randomly generated hand poses with GraspIt, trying to match them with the human-generated contact maps from their previous work. The same authors recorded human hand poses with RGB-D cameras for the same set of objects in [Bra20].

ContactOpt [Gra21] proposes an algorithm where, given estimated hand and object poses from a hand-object grasp RGB image, the hand pose is refined in order to match the object pose and its expected grasping areas. To do so, they propose two models: DeepContact and DiffContact. DeepContact estimates contact maps in both the object and the hand from their estimated poses as point clouds using PointNet++ [Qi17]. DiffContact computes contact maps based on vertex distances between current hand and object poses. Then, the hand mesh is iteratively optimized to minimize the difference between the current contact map computed through DiffContact, and the one estimated by DeepContact.

3.3 Grasping system

To the best of our knowledge, This is the first attempt to generate contact maps from natural interactions with objects in VR environments. In other words, we synthetically generate grasping information from the hand-object interaction in VR environments. Users, embodied in the VR environment using the Oculus Quest headset, can interact with the objects using their hands, resulting in visually realistic grasps (pipeline represented in Figure 3.2). We chose the Oculus Quest headset because it incorporates on-board hand tracking [Han20]. This is done in real time and allows for bare hand interaction in VR settings. As a result, natural finger mobility helps the hand to adapt to the object shape. The grasping system is capable of both grasping and releasing objects without any use of handheld controllers, i.e., there is no need to press a button to grab or release an object.

3.3.1 Main features

The grasping system has to be simple, flexible, visually realistic and robust. We summarize these features as follows:

- **Simple:** Interactions have to be performed in real time and on different types of devices to ensure a pleasant user experience. This points to the need for a simple grasping logic and efficient pipeline that runs seamlessly cross-device.
- **Multi-purpose and flexible:** One of the most remarkable aspects of the grasping algorithm is how flexible it is when grasping objects of different shapes and sizes.
- **Visually realistic:** Fingers are individually positioned on the object surface to precisely fit the hand to its geometry. We correct finger position with inverse kinematics. Although this small modification has a higher computational cost, it enables a realistic looking grasping. This step can be performed offline, to ensure contact map accuracy.

3.3.2 Overview

Our grasping system improves UnrealGrasp [Opr19] with a simpler grasping logic, and using hand tracking as input instead of the handheld controllers. This required modifications in the grasping pipeline which are discussed below. At its roots, the system detects individual finger collisions with the object we are manipulating. To fit the hand to the object shape, each finger is blocked independently. This strategy relied on experimentally placing a capsule collider (figure 3.3) in every fingertip and distal phalanx of the hand model to detect collisions early. That allows to block the position of the fingers over the object surface before the hand-object collision, which usually means unpredictable behaviour. The grasping system is depicted in Figure 3.2 and consists of the following modules:

- **Tracking and Object selection:** Two skeletons compose the system. The first one is invisible and one-to-one matches the movement of our hands using hand tracking, and without considering collisions. The second one follows the movement of the first one, joint by joint, but it physically interacts with the environment, i.e. considering collisions. We have programmatic control over the second skeleton, and that is how we can block the movement of the visible skeleton when we need it. Moreover, to speed up the grasping logic, only the closest object to the palm is considered as a potential target to be grasped. This is detected by a sphere collider placed on the palm and represented in purple in Figure 3.3.

- **Finger manager:** Manages hand-object collisions and determines finger state (free or blocked). It checks the collision capsules placed on finger phalanges. If a collision is detected, the finger movement is blocked to prevent it from penetrating the object surface. It also sets the finger free when the tracked one has moved far enough in the opposite direction where the object is. When a finger is released, this module interpolates the hand pose from the one it had when the hand was blocked to the current pose provided by hand tracking. This is done to avoid drift and ensure a smooth hand movement. Finally, it exposes all finger data to the next component, the Grasping Logic
- **Grasping logic.** This component decides when to grab or release an object. The decision is made based on the phalanx state provided by the finger manager component and the object data. To detect grasps, we have developed an algorithm outputting two possible states: grasped and released. In order to be grasped, the object must be colliding with the collision sphere (so that is reachable) along with either thumb and index, or one of the other fingers plus thumb or index. In that moment, the object is attached to the hand, so that it follows all its movements. When these conditions are not met any more, the object is released by detaching it from the hand.
- **Fingertip adjustment to the object geometry:** When a new grasp occurs, the closest point to the object surface from each finger is computed. Then, fingers are placed on the object surface using inverse kinematics. Resulted grasps are visually improved, and we also increase the accuracy of the contact maps.

Further system description can be found in [Opr19], along deeper explanations on design decisions.

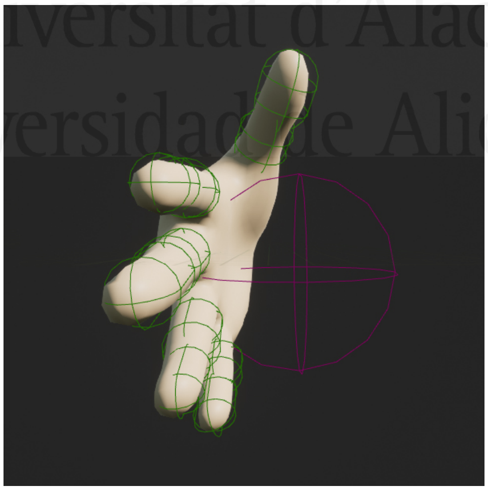


Figure 3.3: Virtual hand with capsule triggers placed on the finger phalanges represented in green. Notice the purple sphere trigger on the palm used to detect graspable objects based.

3.3.3 Limitations

Some of the limitations present in UnrealGrasp [Opr19] have been addressed with this work, such as not being able to move and control each finger separately, or being forced to use motion controllers. These problems were solved in this work as a consequence of the implementation of the hand tracking system.

However, other important limitations are still present and keep the tool far from a truly realistic grasping simulation. The most remarkable one is given by the main simplifications done in the grasping system: blocking completely the fingers when colliding with the object, and attaching rigidly the object to the hand. In this way, we can not properly model the subtle movements in the hand that every object suffers when grasping and holding it. Not having into account the object mass is also a limitation.

Lastly, we are limited by the accuracy of the hand tracking system provided by the Oculus Quest API. A more precise and natural hand tracking system may improve the results, but this is not part of our contribution, as we leverage the built-in hand tracking in the Oculus Quest, making it affordable and easily replicable.

3.4 Grasping data generation and dataset

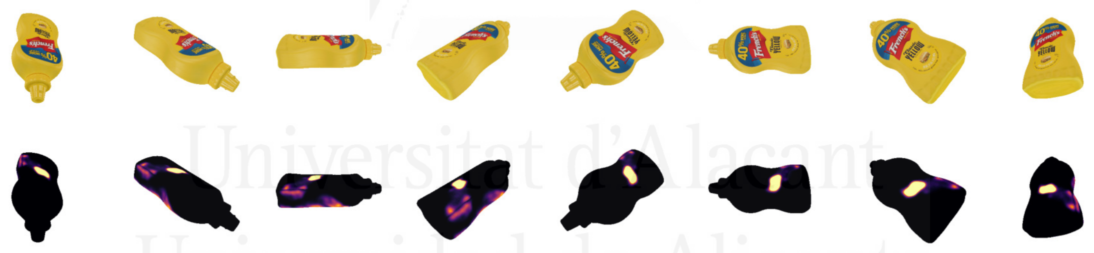


Figure 3.4: Example of the generated views of the same object.

In this section we describe what kind of data can be generated with the tool that has been just presented in section 3.3. Afterwards, the resulting dataset is also described.

3.4.1 Data generation

Using our grasping system in Unreal Engine, we can extract information from object grasps. Different types of data can be generated, from simple grasping contact points to more concrete data such as sequences with per-finger labels. Moreover, our proposed method for extracting data from hand-object interactions in VR has been used along with UnrealROX+ [Mar21] to generate a wider range of data, mainly image and skeleton data.

We can differentiate between extraction tools, developed in Unreal Engine for obtaining and storing data directly from the virtual environment, and post-process scripts, which help us to obtain more useful data from the primary data. Extraction tools are:

- **Surface detector.** Once an object is grasped, this module computes the points over its surface that are in contact with the hand. We have two different strategies for doing so: online and offline. Online is much less computationally hungry, and allows storing the data while interaction is happening between the user and virtual environment, but its accuracy is lower. It consists on tracing a ray between capsule colliders in the fingertips in the direction of the surface normal, detecting the point in the object surface. This strategy focuses on the fingertips and ignore the rest of the hand. The offline method is much more accurate, but it requires rebuilding the relative pose of the hand on the object since its computation would freeze the interaction for a couple of seconds. Once the grasping is rebuilt offline, a ray is traced from every vertex of the object model in the direction of its surface normal, obtaining distances to the hand. These distances are formatted and exported as an UV texture, so that they can be post-processed. We can also define contact points and areas directly in Unreal Engine establishing a threshold distance for considering a contact.
- **Contact map generator.** We also propose two approaches here. The first one is more suitable for online detection, although it also can be used with the offline one. Given the contact points on the object surface, we compute a UV texture that represents the contact map using a custom shader in Unreal, which is computationally fast. They can be exported in HDR format, containing just information about contact intensity. The other approach consist on exporting a map containing the distance of every vertex of the object to the hand, as it is computed in the offline surface detector explained before. Then, we should post-process that map for generating the heat map. That is done through a script which is later described.

On the other hand, we developed several post-process scripts for formatting, refine or compute data from primary data extracted from the rendering engine:

- **Offline contact map generator**. It computes contact maps following the same strategy than the Unreal Engine shader approach, colouring a texture depending on the distance of vertices to the object surface. It can be preferred in offline contact map computation because this step can be easily integrated in the rest of the data-processing pipeline, while HDR textures need to be manage manually in Unreal Engine.
- **Heat map recolouring**. Contact maps can be post-processed to obtain images

coloured as heat maps, as can be seen in Figure 3.5.

- **Automatic object rendering**. Renders of objects with different textures and points of view can be automatically generated through a script using the Pytorch3D python library. Useful for generating all the data for our experiment.
- **3D model overlay**. Renders of 3D models of objects with RGB or heat map textures over an real RGB image with 6D object pose information. Useful for extending the YCB Video dataset [Xia18b], as we will see later in this chapter.

All these tools are available in the *dataset* branch of the public repository, and they can be used by researchers for generating their own data. The different types of data that can be extracted using them is listed below:

- **Contact maps.** Areas that were in contact with the virtual hand during the interaction, provided as UV textures. Contact maps can be generated on individual textures for each finger. In that case, they can also be merged in a single heat map afterwards. Both in image and text.
- **Contact points.** Contact points can be both stored as coordinates in world space or coloured in an individual UV texture.
- **Hand poses and object 6D poses.** Every hand joint location and rotation is stored in a text file along with the 6D pose of the grasped object, as well as the hand used for the grasp (left or right). This allows to rebuild the grasping pose offline and even in other rendering engines (Figure 3.1).

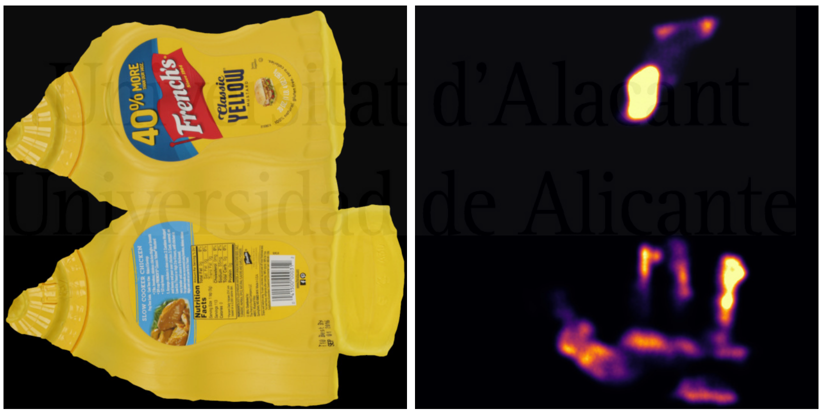


Figure 3.5: At left, original texture of the mustard bottle; at right, generated heat map texture.

Once this data is generated and organized, it can be used for different tasks: conducting a study on the grasps obtained, using this data along with other data to train deep learning networks to predict grasping areas, and even reconstructing and predicting future grasping poses.

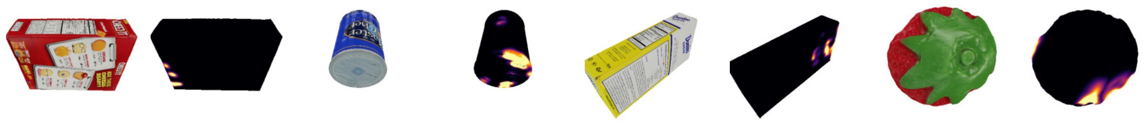


Figure 3.6: Example of data in the dataset. From left to right: cracker box, master chef can, sugar box and strawberry, all from YCB object dataset. In each pair of images, at left, we have the objects rendered with their normal texture; at right we see a contact map applied.

3.4.2 The Dataset: SynContact

We generated a set of data which will be used to train a model for predicting grasp regions. For that purpose, we interacted with all the objects from the YCB dataset [Cal15], and performed several grasps with each one, varying the orientation of the hand and the object. Grasps were performed by three different people. After generating all the heat map textures, everything is post-processed ending up with a set of eight different views of the object with the heat map and original RGB textures applied (figure 3.4). Our dataset, which we named *SynContact*, is composed by:

- 79 different objects from YCB dataset.
- 8 different grasps per object (each one with a unique heat map texture).
- 8 different views per grasp of the object with the heat map and RGB textures applied, as well as the segmentation mask and depth maps.
- 6D Hand Pose and dexterity per grasp.
- 6D Object pose per grasp

Example objects and heat maps from our generated dataset can be seen in Figure 3.6. The dataset can be accessed through a link that can be found in the already mentioned project repository.

3.4.3 YCB Video dataset application

One of the main advantages of working with one of the most extended object datasets, as it is the YCB Object dataset [Cal15], is the synergies that can emerge with other works. In this case, we focus on the YCB Video dataset [Xia18b], which consist on 92 image sequences with a total of 133827 frames, where 21 objects from the YCB dataset appear in different poses and relative locations. Each sequence is composed by 3-6 objects over a table or similar, and they often overlap depending on the frame and point of view. It includes precise and independent annotations for each object in each frame, such as 6D pose and complete and visible masks as well. All these features have made it an standard dataset for 6D pose prediction, and it is part of the BOP Challenge [Hod18] benchmark for this problem.

As the YCB Video dataset has been recorded with the real counterparts of the YCB object dataset, and the 6D pose annotation of each object is provided, we can easily overlay our virtually generated 3D-model grasps over them, as can be seen in the Figure 3.7. This allow us to generate 133827 frames with even more possible grasps combinations, since each frame has several objects, and we provide 8 grasps per object. Tools for generating all this frames are also provided in the project repository.

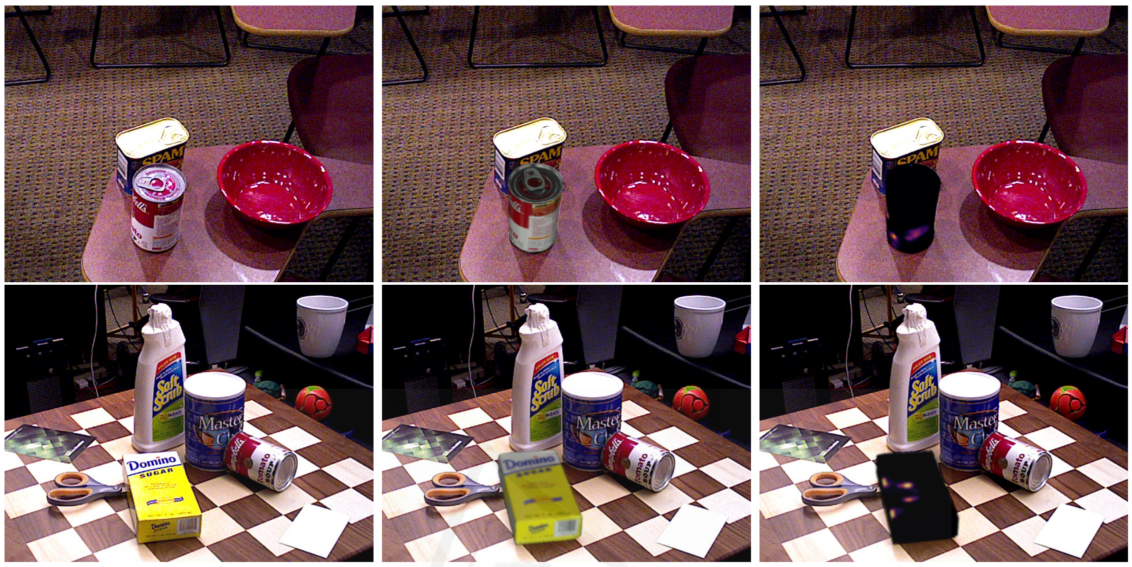


Figure 3.7: Two sample frames from the YCB Video dataset, on the top we focus on the *tomato soup can*, and on the bottom we focus on the *sugar box*. On left, the RGB images. In the middle, the synthetic 3D model of the object projected in the image with the same 6D pose. On right, the synthetic 3D model with the SynContact heat map texture.

3.5 Predicting Contact Maps

In this section, we propose and implement an image-to-image translation model to predict contact maps from RGB images of objects. We trained our model on our generated dataset of contact maps to prove its usefulness at the application level. Specifically, we implemented a Pix2Pix network [Iso17] we call ContactNet from now on. We choose Pix2Pix as base model because it was successfully trained on ContactDB [Bra19a] dataset, proving its usefulness for the task at hand. However, our ContactNet does not use depth information. The model input is a single RGB image of an object, and the output is a three-channel contact map.

ContactNet: Implements vanilla Pix2Pix with no architectural changes apart from the number of input and output channels. It uses a U-Net [Ron15] as a generator and a PatchGAN classifier as a discriminator. On the one side, PatchGAN is a convolutional network implementing four downsampling blocks and a final 2D convolution.

Each downsampling block consists of a 4 × 4 2D convolution, batch normalization, and leaky ReLu as activation function. Different from a regular discriminator, Patch-GAN evaluates the inputs as a grid of patches, where each patch may be "real" or "fake". Mathematically, this is equivalent to slicing the image into a grid of patches, input each patch to a regular discriminator, and as a result average the results of all patches. On the other side, the U-Net generator is a simple encoder-decoder like architecture. For more details, we recommend the reader to check the original paper [Ron15].

Training procedure: We trained the ContactNet on objects centred in 256 × 256 RGB images. Both RGB images and contact maps have white backgrounds. Thus, we make the model focus only on the object in the image by masking the generated images. ContactNet was trained for 200 epochs with the same learning rate (0.0002) for the generator and discriminator. We used a batch size of 8, and sample shuffling in the training set.

Validation: We have validated the model by feeding images of unseen objects or objects from a different perspective. The idea is to verify the model produces suitable contact maps. This is, check the model can relate the 2D shape of an object to the grasping areas that a hand can reach. For instance, in the case of a cereal box, we expect to generate potential grasping areas in the borders. Some predictions are depicted in Figure 3.8.

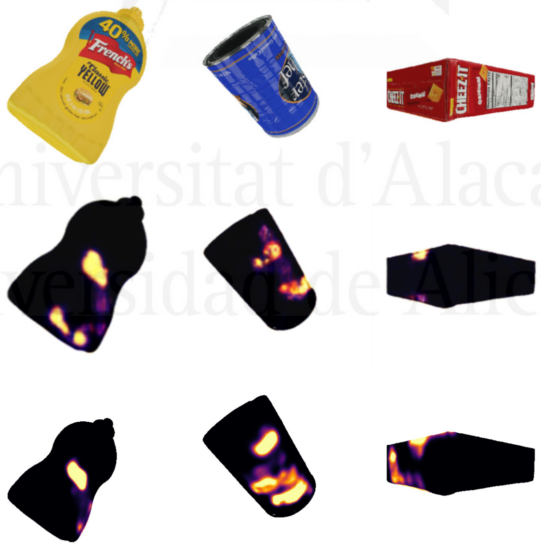


Figure 3.8: At the top row, input images to the ContactNet. At the middle, predicted contact maps. At the bottom row, ground truth contact maps. Regarding the mustard bottle, notice the generated contact map shares similarities with those represented in the ground truth. Moreover, look at the contact map generated for the big cracker box in the borders. Boxes often present contact areas at the edges, as we naturally grip them from the sides.

3.6 Conclusion

In this chapter of the thesis, we improved the UnrealGrasp [Opr19] grasping system with a novel grasping logic and using hand tracking instead of Oculus handheld controllers. Using the grasping system, we were able to interact with objects in VR environments. We captured hand-object interactions in forms of contact maps to enable grasp region prediction on objects using learning-based approaches. To ensure our synthetically generated dataset of contact maps is useful, we implemented a Pix2Pix to translate RGB images of objects to heat maps indicating potential grasping areas on the objects. We showed the model can generate plausible grasping spots on unseen objects.

3.7 Limitations and Future Works

As we said at beginning, the purpose of our grasping system is being able to naturally interact with objects in virtual environments, so that we can automatically extract accurate, realistic and labelled data which help us to train machine learning models for predicting grasping regions. In that sense, we don't really need the grasp to be completely precise in real time. We could just detect the grasp, store the object and hand poses, rebuild the scene offline and refine the grasp as precisely as we need. With that strategy we should be able to keep the grasping logic as simple as possible and boost interaction fluency. That is something that can be improved in the future.

Moreover, a strong limitation of the system is the fact of not being physically rigorous. We trust in the rendering engine physics and focus on being visually realistic. Also the interaction through a VR headset is far from being completely natural, and the way of grabbing an object is affected by that. We try to maximize the success when grasping things in the virtual environment, but maybe that is not the way we would have grasped the object.

Finally, in the test we carried out with our dataset, we only have used heat map information. Hand and object pose information is also available, so an interesting future work may be training some grasping and hand pose estimation method with synthetic data in order to see how good it performs when moving to the real domain.

Part II

6D Object Pose and Grasping Point Estimation

Chapter

4

The 6D Object Pose Estimation Problem

Deep learning has recently emerged as a promising approach for 6D object pose estimation, which involves determining the position and orientation of an object in 3D space from a RGB image. This task has numerous applications in fields such as robotics, augmented reality, and computer vision. Deep learning-based methods leverage the ability of neural networks to learn complex representations of objects and scenes, allowing for accurate and efficient pose estimation.

In this brief review, we will explore various methods based on deep learning that have been proposed for 6D object pose estimation. We will examine their strengths and limitations, as well as their performance on benchmark datasets. Additionally, we will discuss in more depth two methods whose architecture will be modified in chapter 5 to the problem of grasping estimation. Lastly, we will briefly analyse the challenges and future directions in this field, and how deep learning can continue to improve the accuracy and efficiency of 6D object pose estimation.

4.1 Introduction

In recent years, the field of autonomous robots has seen a significant growth in its ability to grasp objects. For a robot being able to successfully grasp an object, it needs to intelligently understand its environment. We can disaggregate this huge task in three smaller (but also complex) ones: object localization, object pose estimation, and grasp estimation. However, many object pose estimation methods do not require object localization and instead perform both tasks jointly.

In this chapter we will focus on methods for estimating the 6D pose of objects based on deep learning techniques. With the advancement of the latest, the accuracy and reliability of 6D object pose estimation has improved significantly. This chapter will provide an overview of the state-of-the-art methods in this field and provide insights into how these methods can be used to improve the grasp and manipulation capabilities of autonomous robots.

This chapter is organized as follows. Section 4.2 describes the problem from a general point of view and defines it mathematically, along with some taxonomy for different approaches. After that, in Section 4.3, deep learning-based techniques are addressed, both those which take just a RGB image as input and those which also take a depth map (RGB-D). Later, two concrete architecture are analysed in detail in Section 4.4, as they will be the chosen ones in the following chapter for the proposal of several modifications. Finally, in Section 4.5 we close the chapter with some concluding words and an introduction for the next chapter.

4.2 The problem

6D object pose estimation is a task in computer vision and robotics that involves determining the position and orientation of an object in 3D space. This is an important problem with many practical applications, such as robotic grasping and manipulation, augmented reality, and scene understanding. The goal of 6D object pose estimation is to determine the 6D pose, which consists of the 3D translation and 3D rotation of the object relative to a fixed reference frame. This information is crucial for many robotic tasks, such as grasping an object or placing it in a specific location.

The 6D object pose estimation problem is challenging for several reasons. First, objects in the real world can appear in a wide variety of poses and scales, making it difficult for algorithms to generalize to new situations. Second, the appearance of objects can change significantly based on lighting, occlusion, and viewpoint, making it difficult to detect and recognize objects from all angles. Finally, the 3D structure of objects and scenes can be complex and ambiguous, making it difficult to estimate the 6D pose accurately and efficiently.

To overcome these challenges, various approaches have been proposed, ranging from traditional computer vision methods based on feature detection and matching, to more recent deep learning-based methods that leverage the power of neural networks to learn rich representations of objects and scenes.

4.2.1 6D Pose Definition

The 6D pose of an object is defined as the combination of a 3D rotation matrix $R \in SO(3)$ and a 3D translation vector $T = [t_x, t_y, t_z]^T$ that transform each point $X_o = [x_o, y_o, z_o]^T$ in the object frame O to the same point $X_c = [x_c, y_c, z_c]^T$ in the camera frame C [Nej22a]. The pose parameters represent the orientation and position of the object in the camera coordinate frame. The translation vector $T = [t_x, t_y, t_z]^T$ indicates the location of the origin of the object frame in the camera frame, and the rotation matrix $R = R_z(\alpha)R_y(\beta)R_x(\gamma)$ represents the angles between each axis of the object frame and the corresponding axis of the camera frame when the origin of the object frame is transformed to the origin of the camera frame. Any point X_o in O has its corresponding point X_c in C following the expression:

$$X_c = [R|T]X_o \tag{4.1}$$

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[RT] \begin{bmatrix} x_o \\ y_o \\ x_o \\ 1 \end{bmatrix} \tag{4.2}$$

Where R is the rotation matrix. For 3D bounding box estimation, the size of the object D is also considered in addition to R and T.

4.2.2 General approaches

The problem of pose estimation can be approached using different strategies. In [Nej22a] four categories are differentiated: direct classification, regression, 2D-3D correspondences, or 3D-3D correspondences. In the following subsection we will briefly describe each approach.

- **Classification**: Classification-based approaches divide the space of 3D rotations into discrete bins, turning the 3D rotation estimation into a classification problem [Sun18a, Nej22b]. The discretization results in a rough estimate, which requires further refinement to obtain accurate 6D pose, typically through the use of ICP algorithm or by regressing the offset [Mou17]. These models usually consist of Convolution layers for feature extraction, followed by a MLP at the end to classify the output into a number of bins (Figure 4.1). To obtain the full 6D pose, the classification results are combined with offset regression. The offset represents the residual rotation correction required to be applied to the centre of the bin. This offset can be expressed as two numbers, the sine and cosine of the angle, which are typically predicted using a separate branch of the network. These models generally perform better than approaches that directly regress the 6D pose. The output for each bin i includes the class of the pose, the cosine of the offset, and the sine of the offset. The loss function for these methods combines the classification and regression losses.

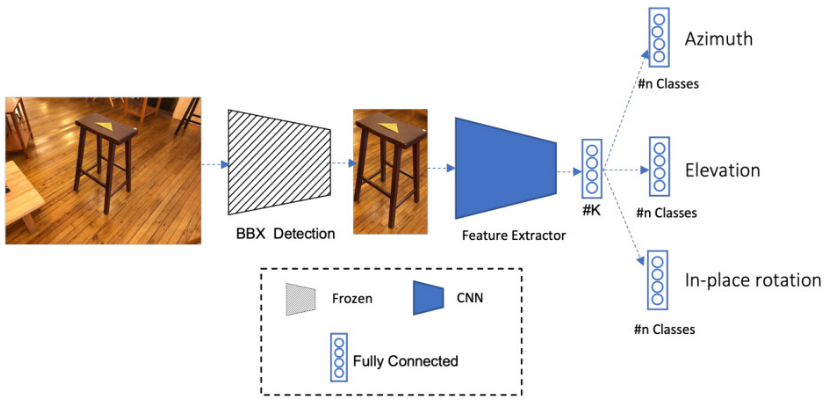


Figure 4.1: Overall 6D pose classification model (from [Nej22a]).

- **Regression**: The pose estimation problem can be approached as a regression task, with simple methods using a CNN to directly predict the 3D rotation and translation [Ami21]. While these regression approaches are simpler, they tend to perform worse than more complex classification methods. Figure 4.2 illustrates a basic pipeline.

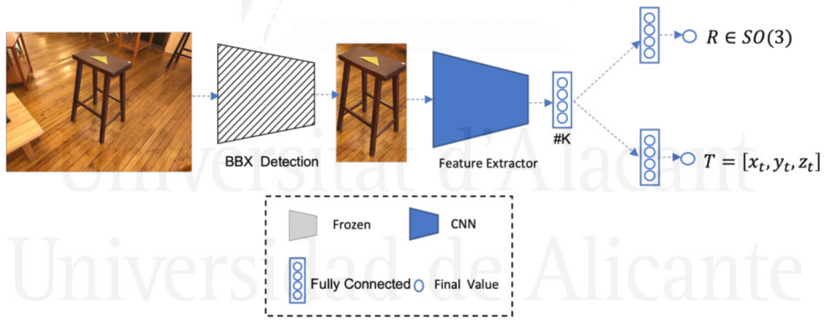


Figure 4.2: Overall 6D pose classification model (from [Nej22a]).

- **2D to 3D**: The object pose estimation problem can be solved through a two-stage process. The first stage involves finding correspondences between the object in its own coordinate frame and the image captured by the camera. These correspondences can be found using traditional feature descriptors like SIFT for images with rich textures. However, these methods are not effective for texture-less objects or low-resolution images. The second stage involves using the correspondences and algorithms like PnP and RANSAC to calculate the 6D pose of the object. PnP uses a set of 3D points in the world coordinate frame, their corresponding 2D projections in the image, and the calibrated camera to solve for the pose. The algorithm can be solved with three points correspondences, but additional ones can be used to remove ambiguity. A more recent method, EPnP, is able to solve the general problem of PnP for more than four correspondences. The PnP algorithm can be affected by outliers in the set of corresponding points. To overcome this issue, RANSAC can be utilized in combination with PnP. RANSAC is an iterative method that selects a random subset of the correspondences, and then PnP is applied to that subset. The other points are then evaluated against the computed pose, and points that fit the estimated model well are included in the consensus set. The final model is then re-estimated using all members of the consensus set. This approach is more robust to occlusions and truncations compared to the simple classification and regression methods, as it utilizes information from all points that fit the estimated model well.

- **3D to 3D**: The 6D pose estimation with the availability of 3D data can be tackled by finding correspondences between the object's 3D representation and the CAD model. These correspondences are then used to solve for the translation and rotation. To find the translation, the average of the points in one coordinate system A is subtracted from the average of their corresponding points in the other coordinate system B. We won't focus on these methods since we will address only images as inputs.

4.3 Approaches using Deep Learning Techniques

In this work we want to focus on deep learning models for estimating the 6D pose of an object. Even if they have some drawbacks, e.g. the poor interpretability of the decisions made by these models, they have been outperforming classical approaches, especially with the application of CNN [Kri12]. Based on the work of [Zhu22], we are going to review some of the state-of-the-art methods, dividing them depending on whether they use just RGB images as input, or whether they also use a depth map (RGB-D).

4.3.1 Algorithms based on RGB images

An RGB image is formed by the modulation and superposition of the red, green, and blue colour channels. Several researchers have attempted to utilize deep learning models to comprehend the pose of objects in RGB images. However, these RGB image-based deep learning algorithms are often susceptible to the impact of illumination and pose variations.

One approach for object pose estimation is to treat it as either a regression or classification task. This involves directly predicting the parameter representation associated with the object's pose from the input image. For instance, the SSD-6D model [Keh17] extends the SSD paradigm and takes a single-frame RGB image as input. Using the Inception-V4 network [Sze17], it estimates the 2D bounding boxes and scores for all views and in-plane rotations. The model then establishes a pose pool using projective geometric characteristics and further refines the results using ICP.

A network called DeepIM has been proposed [Li18] to estimate the pose of objects. While VGG16 [Sim15] was originally considered as the basic network, it was found to perform poorly. Instead, the FlowNetSimple [Dos15] was used to predict the optical flow between images. In the pose estimation branch, the input consisted of the characteristic map obtained after passing through 10 convolution layers. This input was processed through two successive 256-dimensional fully connected layers. The 3D rotation and 3D translation were predicted using two fully connected layers, respectively. During training, DeepIM also utilized two auxiliary branches. One branch was employed to predict the optical flow between the rendered image and the observed image, while the other was used to predict the foreground mask of the object in the observed image. Through iterative pose optimization of images, this algorithm realizes an automatic learning internal optimization mechanism.

PVNet [Pen19] has been developed to predict vector fields and perform semantic segmentation. Similar to reference [Bra14a], the RANSAC algorithm is utilized to vote and cast candidate points of 2D key points, which possess a position and distribution. Finally, an improved PnP algorithm is used to estimate their pose. However, the key point-based algorithm requires two critical assumptions. Firstly, the position of 2D key points must be accurately estimated. Secondly, sufficient constraints are needed to regress the potential pose from these estimates. Nevertheless, these two assumptions are often challenging to fulfil in practical applications. Due to occlusion, network representation, and other factors, accurately predicting the location of 2D key points from RGB images alone is a significant challenge.

Pix2Pose [Par19] predicts the 3D position of each object pixel without relying on textured models. It uses an auto-encoder architecture to calculate the 3D coordinates and expected errors per pixel, and these predictions are then utilized in multiple steps to establish 2D-3D relationships, ultimately leading to the calculation of poses through the PnP algorithm with RANSAC. This system is highly resistant to occlusion, due to its use of generative adversarial training to accurately recover obscured parts.

HybridPose [Son20] stands out as a more stable algorithm than PVNet and other single-representation-based algorithms. The pose estimation performance of this algorithm remains unaffected even if one of the intermediate representations is not accurately estimated. The hybrid intermediate representation used in HybridPose represents different geometric information in the input image, such as key points, edge vectors, and symmetrical features. These various intermediate representations can be predicted by a simple neural network. To ensure accuracy, the robust regression module filters out any abnormal values in the intermediate representation.

Many of the above-mentioned pose estimation algorithms rely on CNN networks, which are supervised learning algorithms. However, the limited availability of annotated data for a target object poses a significant challenge for training such models. To address this challenge, self-supervised learning-based algorithms have been proposed. For instance, the Self6D [Wan20] model uses synthetic RGB data and unannotated RGB-D data to learn an auto-correlation model between real images. Similarly, the DSC-PoseNet model [Yan21] generates pseudo mask real images, annotated only with a bounding box, and performs cross-scale self-supervision to estimate the mask of the object, key point offset of each pixel, and attention map. The final pose estimation is achieved by weighted averaging pixel by pixel in the foreground. These algorithms do not require 3D pose labels and can be trained using easily available 2D frame annotations. The DSC-PoseNet network realizes self-supervised training and pose estimation based on the self-supervised loss of contour alignment and bi-scale consistency, and its performance is comparable to that of supervised learning-based algorithms.

The geometry-guided direct regression network (GDR-Net)[Wan21] utilizes both a CNN and a learnable Patch-PnP method to predict the pose of the target object directly from a single RGB image (Figure 4.3). This is achieved by incorporating geometric guidance through the use of an intermediate representation of dense correspondence. SO-Pose [Di21] employs a shared encoder and two separate decoders to generate 2D-3D correspondence and self-occlusion information. The outputs are combined to directly output the 6-degree-of-freedom (6dof) pose parameters. KDF [Liu21] is a continuous representation technique that projects 2D key positions to address occlusion in pose estimation.

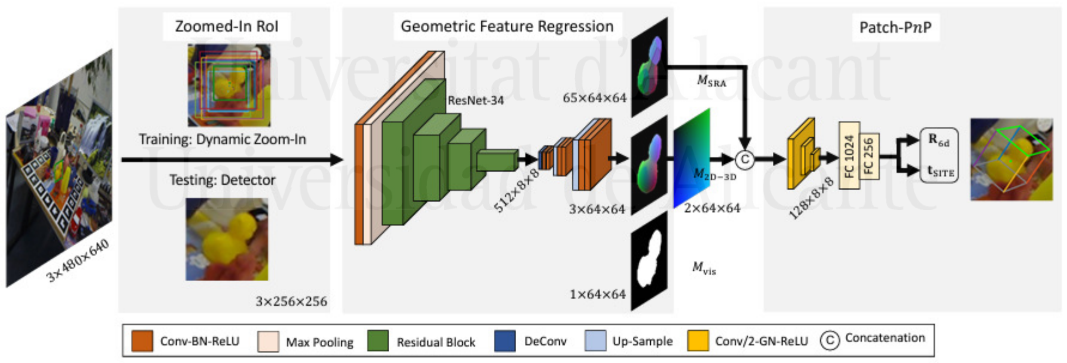


Figure 4.3: Framework of GDR-Net. Given an RGB image I, our GDR-Net takes the zoomed-in region of interest (Dynamic Zoom-In for training, off-the-shelf detections for testing) as input and predicts several intermediate geometric features. Then the PatchPnP directly regresses the 6D object pose from Dense Correspondences (M2D-3D) and Surface Region Attention (MSRA) [Wan21].

Various evaluation measures are used for estimating the position of objects, such as ADD, ADD(-S), ADD(-S) AUC, among others [Hin12a, Hod16]. ADD measures whether the average deviation of all points between the predicted and actual poses is less than 10% of the object's diameter. This is useful for asymmetric objects, but for symmetrical objects we can use ADD(-S), which calculates the deviation by measuring the distance from each predicted point to the nearest actual point. ADD(-S) AUC involves plotting the ADD(-S) curve using different thresholds and measuring the area under the curve as the evaluation index, with a threshold of 10cm. In the Table 4.1 a performance comparison between the treated methods can be seen.

Method	Year	Linemod ADD(-S)	Linemod Occlusion ADD(-S)	YCB-Video ADD(-S)	YCB-Video ADD(-S) AUC
SSD-6D [Keh17]	2017	76.30	-	-	-
DeepIM [Li18]	2018	88.60	55.50	-	81.90
PVNet [Pen19]	2019	86.27	40.77	-	73.40
Pix2Pose [Par19]	2019	72.40	32.00	-	-
HybridPose [Son20]	2020	91.30	47.50	-	-
Self6D [Wan20]	2020	58.90	32.10	-	-
DSC-PoseNet [Yan21]	2021	58.60	24.80	-	-
GDR-Net [Wan21]	2021	93.70	62.20	60.10	84.40
SO-Pose [Di21]	2021	62.30	-	-	-
KDFNet [Liu21]	2021	-	50.30	-	-

Table 4.1: RGB image-based methods for deep learning 6D object pose estimation [Zhu22].

4.3.2 Algorithms based on RGB-D images

RGB-D image-based deep learning algorithms have gained attention due to the limitations of using only RGB images for object detection and pose estimation. These algorithms utilize both RGB and depth information, where an RGB-D image is composed of an RGB map and a depth map. The depth map contains information about the distance from the sensor to the object and corresponds to the pixels on the RGB map. This additional information provided by the depth map allows for more accurate object localization and pose estimation.

Traditional methods for feature fusion extract global features from both RGB and depth features, but their performance can be easily affected by occlusion, non-target object features, and background features. In reference [Wan19], it is proposed that even though RGB and depth information in RGB-D data have the same format, they have different internal correlation structures. Therefore, if the information is represented in different feature spaces, RGB and depth information should not be processed in the same way. A heterogeneous network called DenseFusion is proposed to process the heterogeneous data of RGB and depth data in different feature spaces, which effectively retains their own structure. The network is integrated in a dense pixel level fusion method and is continuously studied and trained through a differentiable iterative refinement module, optimizing the results of pose prediction. Due to the limitations of ICP in the implementation process, a differentiable iterative refinement module is designed to refine the prediction. This module is an iterative refinement module based on a neural network. It takes the prediction in the previous iteration as part of the input of the next iteration and transforms the input point cloud into a standard frame. The transformed point cloud is implicitly encoded and fed back to the network, which predicts the current pose based on the previously estimated pose.

The G2L-Net [Che20] is a framework for real-time object pose estimation that uses two-dimensional detection to extract coarse-grained object point clouds from RGB-D images. The segmentation and translation information is predicted through the migration and positioning network using these coarse-grained point clouds. The point cloud of the fine-grained target object is transformed into local regular coordinates, and the initial target rotation is predicted using a trained rotation positioning network. The rotation result is refined by a rotation residual estimator. In PVN3D [He20], a deep Hough voting network is used to detect the three-dimensional key points of the target object, and pose parameters are estimated by least square fitting, which takes advantage of the geometric constraint information of rigid objects. The hough transform used in the algorithm votes in the parameter space to determine the shape of the object, relying on the local maximum in the accumulation space.

In contrast to colour information, depth information has received less attention despite its rich geometric information on object shape. Since synthesizing colour images can be costly, it is often easier to estimate pose based solely on depth information. Some researchers have explored the use of depth information only for pose estimation [Gao20], such as in a proposal to generate a point cloud of a target object using depth and semantic information. The resulting point cloud segmentation is downsampled and the rotation and translation of the target objects are regressed through two separate networks.

The CloudAAE method [Gao21] uses an improved auto-encoder specifically designed for point cloud data to learn a latent code that encodes the pose information of the target object. Compared to existing pose estimation algorithms that rely on synthesizing RGB data, CloudAAE has a lower computational cost and requires less hardware storage.

The StablePose model [Shi21], goes beyond determining stable patch combinations for pose estimation and also calculates the correspondence between the observed image patch and 3D model patch, which is used to learn the relationship for pose estimation. In addition, a differentiable geometric attitude estimator is proposed, which allows for attitude error back propagation and the simultaneous regression of candidate results and confidence, leading to better robustness in the prediction of abnormal key points [Hua21]. BundleTrack [Wen21], optimizes pose estimation results by leveraging the continuity between key frame images through pose map optimization. All these RGB-D based methods can be compared in Table 4.2.

Method	Year	Input	Linemod	Linemod Occlusion	YCB-Video
G2L-Net [Che20]	2020	RGB-D	98.7	-	92.4
PVN3D [He20]	2020	RGB-D	95.5	-	92.3
CloudAAE [Gao21]	2021	Depth	82.1	58.9	-
CloudAAE [Gao21] + ICP	2021	Depth	92.5	66.1	93.5
StablePose [Shi21]	2021	RGB-D	-	63	-
REDE [Shi21]	2021	RGB-D	98.9	65.4	-

Table 4.2: RGB-D image-based methods for deep learning 6D object pose estimation [Zhu22].

4.3.3 Datasets

Object pose estimation datasets can be categorized into two formats: RGB and RGB-D. RGB images are created by combining red, green, and blue colour channels. In contrast, RGB-D images provide additional depth information. KITTI [Gei12b] is an example of an RGB dataset that contains real-world images captured in various settings, including urban, rural, and highway environments. The dataset annotations include 7 degrees of freedom, which consist of three for object translation, three for object size, and one for object rotation. The dataset consists of eight object categories, and the images have a resolution of 384x1280. Objectron [Ahm21] is another RGB dataset, which comprises annotated video clips with a resolution of 1920x1440. The data were collected from ten countries on five continents to ensure diversity in geographic locations.

The most widely used data set for RGB-D object pose estimation is Linemod [Hin12b], which provides annotations for 6 degrees of freedom, including three for rotation and three for translation, and has an image resolution of 640×480. To investigate the problem of occlusion, the Linemod Occlusion data set [Bra14b] was created based on Linemod and includes annotations for 6 degrees of freedom for 8 objects under severe occlusion. The YCB-Video [Xia18b] data set contains annotations for 21 objects. All these data sets and their features are collected in Table 4.3.

Data sets	Format	Categories	DOF	Resolution
KITTI [Gei12b]	RGB	8	7	384x1280
Objectron [Ahm21]	RGB	9	9	1020x1440
Linemod [Hin12b]	RGB-D	13	6	640x480
Linemod Occlusion [Bra14b]	RGB-D	8	6	640x480
YCB-Video [Xia18b]	RGB-D	21	6	640x480

Table 4.3: Data sets for 6D object pose estimation [Zhu22].

4.4 Cases of study

For our experiments in the Chapter 5, we are going to focus mainly in two models. These are PVNet [Pen19] and Pix2Pose [Par19]. In this section we are going to describe each of them in detail.

4.4.1 PVNet

PVNet (Pixel-wise Voting Network) is a deep learning-based method for 6D object pose estimation from RGB images. It consists of two main stages: feature extraction (2D keypoints detection) and then compute 6D pose parameters using the PnP algorithm. The proposal is a novel representation of the 2D object keypoints which is robust against occlusion, as well as a modified version of PnP to include uncertainty. Figure 4.4 shows the main pipeline of the method.

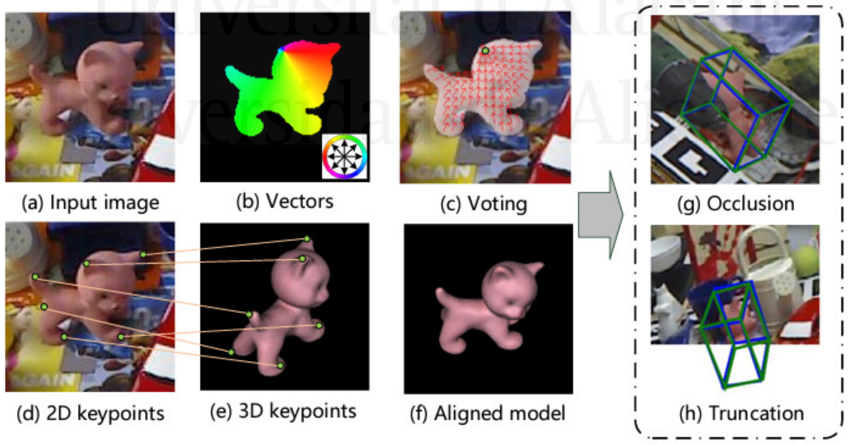


Figure 4.4: The 6D pose estimation problem is formulated as a PnP problem [Pen19].

The proposed keypoint localization pipeline is illustrated in Figure 4.5. The process starts with an RGB image, which PVNet uses to predict object labels for each pixel, along with unit vectors indicating the direction from each pixel to each keypoint. These directions are used to generate 2D location hypotheses and confidence scores for each keypoint via RANSAC-based voting. Finally, the mean and covariance of the spatial probability distribution for each keypoint are calculated based on these hypotheses. The approach of predicting pixel-wise directions instead of directly regressing keypoint locations from an image patch has several benefits. By focusing on local features of objects, it helps to reduce the impact of cluttered backgrounds. Additionally, this approach makes it possible to locate keypoints that are occluded or outside the image. Even if a keypoint is not visible, it can still be accurately positioned based on directions estimated from other visible parts of the object. This is in contrast to other methods that rely solely on regressing keypoint locations from an image patch. More specifically, PVNet performs two tasks: semantic segmentation and vector-field prediction.

In the RANSAC-based voting scheme, keypoint hypotheses are generated using the semantic labels and unit vectors. The first step is to identify the pixels belonging to the target object using the semantic labels.

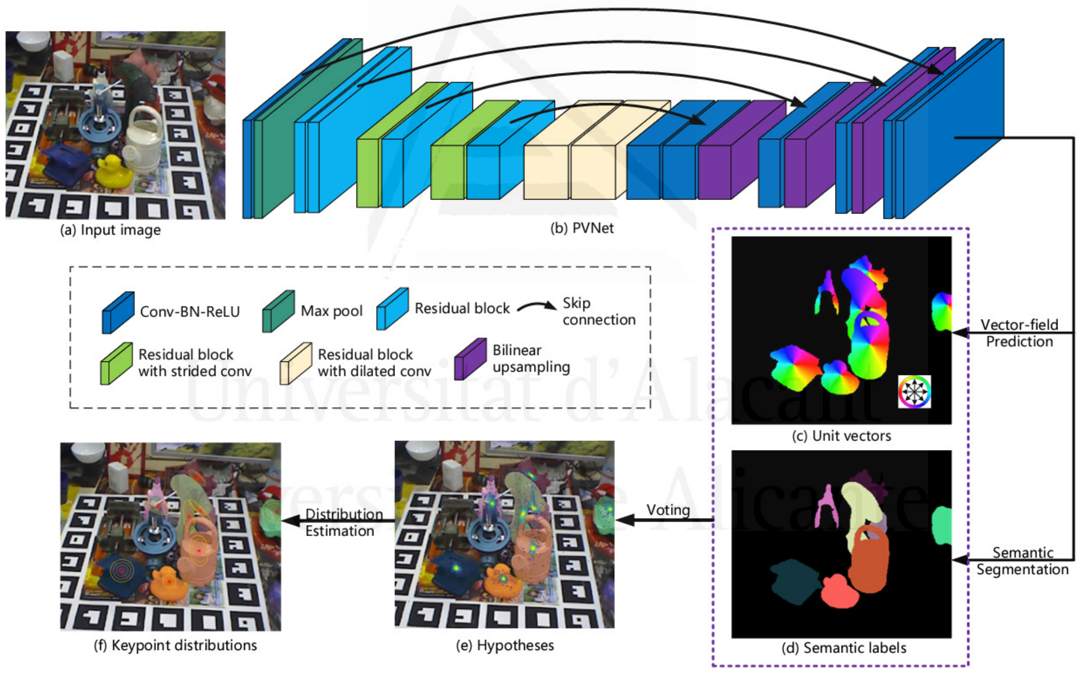


Figure 4.5: Overview of the keypoint localization [Pen19].

The choice of keypoints is crucial and must be based on the 3D object model. Many recent methods choose to use the eight corners of the 3D bounding box of the object as keypoints. However, this approach can result in larger localization errors because the keypoint hypotheses are generated using vectors that originate from object pixels, and the bounding box corners are farther away from these pixels. Figures 3(b) and (c) demonstrate the hypotheses generated by PVNet for a bounding box corner and a keypoint on the object surface, respectively. Keypoints on the object surface typically have a smaller variance in localization.

To achieve more accurate localization, it is important to choose keypoints on the object surface. Additionally, these keypoints should be distributed evenly across the object to ensure stability of the PnP algorithm. To meet these requirements, we use the farthest point sampling (FPS) algorithm to select K keypoints. The keypoint set is initialized by adding the object centre, and the process then involves repeatedly finding the point on the object surface that is farthest from the current keypoint set and adding it to the set until K keypoints have been selected. Conducted experiments show that this strategy produces better results than using the bounding box corners. We also evaluate the results for different numbers of keypoints and suggest $K = 8$ as a good balance between accuracy and efficiency.

Once the 2D locations of keypoints for each object have been determined, its 6D pose can be computed by solving the PnP problem using a standard PnP solver, such as the EPnP algorithm used in many previous methods. However, many of these methods fail to account for the fact that different keypoints may have varying levels of confidence and uncertainty, which should be taken into consideration when solving the PnP problem. As explained before, our voting-based method calculates a spatial probability distribution for each keypoint.

4.4.2 Pix2Pose

Pix2Pose is a computer vision model designed to estimate the 6D pose of objects in an image. It is a deep learning model that uses CNN to predict the object's position and orientation in the image. Pix2Pose predicts the 3D position of each object pixel without relying on textured models. It uses an auto-encoder architecture to calculate the 3D coordinates and expected errors per pixel, and these predictions are then utilized in multiple steps to establish 2D-3D relationships, ultimately leading to the calculation of poses through the PnP algorithm with RANSAC. This system is highly resistant to occlusion, due to its use of generative adversarial training to accurately recover obscured parts. Additionally, a new loss function called the "transformer loss" has been introduced to handle symmetrical objects, guiding predictions towards the closest symmetrical pose. Evaluations on three benchmark datasets featuring both symmetrical and occluded objects demonstrate that our method outperforms existing solutions and uses only RGB images.

The design of the Pix2Pose network is depicted in Figure 4.6. The network takes as input a cropped image of an object class that has been detected using a bounding box. The outputs of the network are the normalized 3D coordinates (I3D) of each pixel in the object coordinate system, as well as the estimated errors (Ie) for each prediction, computed through the function G(Is). The target output includes predictions of the 3D coordinates of occluded parts, which enhances the network's resistance to partial occlusion. The output, I3D, is similar to the RGB values in an image and can be

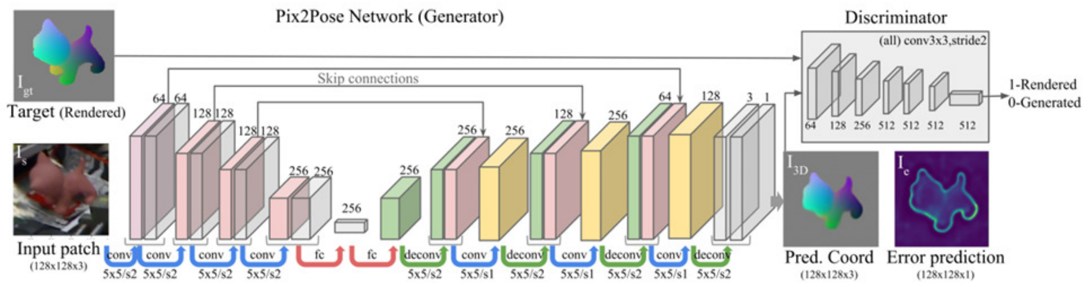


Figure 4.6: An overview of the architecture of Pix2Pose and the training pipeline [Par19].

visualized as a colour image, as shown in Figure 4.7. The ground truth output can be easily obtained by rendering the coloured coordinate model in the ground truth pose. The error prediction, Ie, serves as a confidence score for each pixel, which is used to distinguish outliers from inliers prior to pose computation.

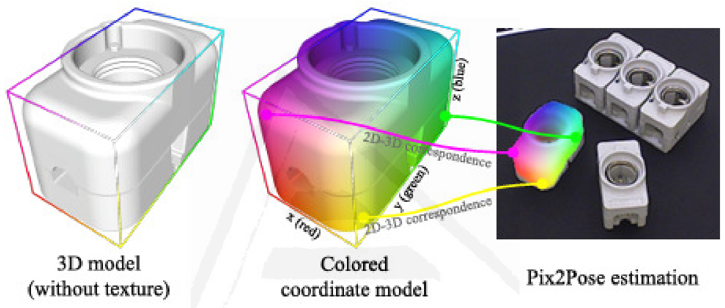


Figure 4.7: An example of converting a 3D model to a coloured coordinate model [Par19].

The input image patch is resized to 128x128 pixels and has three channels to represent RGB values. The first four layers of the network, which form the encoder, have the same filter and channel sizes as described in [Sun18b]. To preserve low-level details, skip connections [Ron15] are added by copying half of the outputs from the first three layers to the corresponding layers in the decoder. This solution provides more accurate predictions for pixels near geometric boundaries. The filter size for all convolutional and deconvolutional layers is fixed at 5x5 with a stride of either 1 (denoted as "s1") or 2 (denoted as "s2") in Figure 4.6. Two fully-connected layers are used for the bottleneck, with 256 dimensions, between the encoder and decoder. Batch normalization [Iof15] and the *LeakyReLU* activation function are applied to all intermediate layer outputs, except for the last layer. The final layer has three channels and uses the *tanh* activation function to produce the 3D coordinate image (I3D), and one channel with the sigmoid activation function to estimate the expected errors (Ie).

The process of computing a pose using the output of the Pix2Pose network is depicted in Figure 4.8. To estimate the pose, the region of interest is cropped based on the centre, width, and height of each bounding box, and then resized to 128x128 pixels. The aspect ratio is maintained by taking the larger value of the width and height, which are then multiplied by a factor of 1.5 to potentially include occluded parts. The pose prediction is performed in two stages, using the same network in both stages. The first stage aligns the input bounding box to the centre of the object and removes any unnecessary pixels that are not preferred by the network. The second stage refines the input from the first stage to make a final estimation and compute the final pose.

1. **Stage 1: Mask prediction and Bounding Box Adjustment**. In the first stage of the pose computation process, the predicted 3D coordinate image I3D is used to identify the pixels that belong to the object, including the occluded parts, by selecting pixels with non-zero values. The error prediction is also used to eliminate uncertain pixels if the error for a given pixel is greater than the outlier threshold θ_o. The valid object mask is created by combining pixels with non-zero values and pixels with errors lower than θ_o. The centre of the bounding box is then recalculated using the centroid of the valid mask. This results in a refined input that only contains the pixels within the valid mask and crops the input to a new bounding box. The refined input may include occluded parts if the error prediction for these pixels is below the outlier threshold, indicating that their 3D coordinates can still be predicted accurately despite the occlusions. Figure 4.8 shows examples of the output from the first stage.
2. **Stage 2: Pixel-wise 3D Coordinate Regression with Errors**. In the second stage of the process, the refined input from stage 1 is used to predict a 3D coordinate image and its associated error values, as shown in Figure 4.8. The black pixels in the 3D coordinate samples indicate points that are ignored if the error prediction is higher than the inlier threshold θ_i, even if the points have non-zero coordinate values. The pixels with non-zero coordinate values and error predictions lower than θ_i are used to form 2D-3D correspondences, where the 2D image coordinates and predicted 3D coordinates directly correspond to each other. The final pose is then calculated by applying the PnP algorithm with RANSAC iteration, maximizing the number of inliers that have re-projection errors lower than the threshold θ_re. It's worth noting that Pix2Pose doesn't require textured 3D models and that the pose estimation process is fast since there's no rendering involved.

4.5 Final Discussion

In this chapter, we have explored the 6D object pose estimation problem in general, and we focused on deep learning-based methods. It is an important and challenging problem in computer vision, key to enable robots to be fully autonomous in real

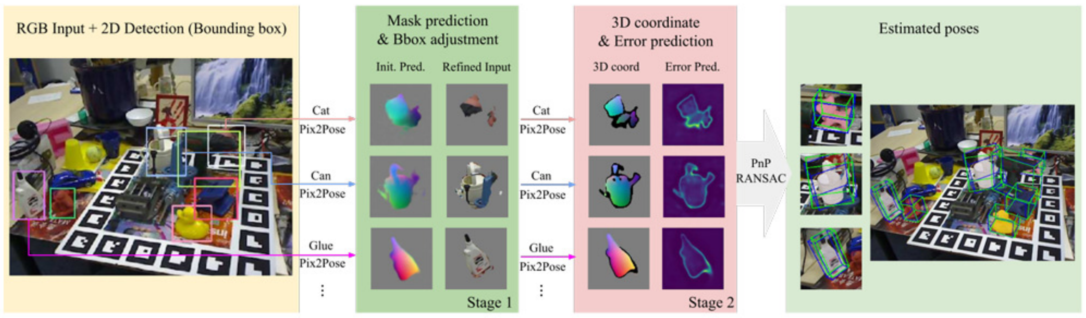


Figure 4.8: An example of the pose estimation process [Par19].

human environments, interacting with objects. The methods we have reviewed are designed to predict the position and orientation of objects in 3D space, given a 2D image (with or without a depth map).

Deep learning-based methods utilize large amounts of training data. However, we have previously discussed the limitations and challenges of these methods, including their sensitivity to lighting, occlusions, and noise, and their dependence on large amounts of labelled training data. Some potential solutions to these challenges are data augmentation, transfer learning, unsupervised learning, and synthetic data as we proposed in the first part of this thesis. All the datasets reviewed in section 4.3.3 are based on real data, obtained through cameras. In this field, 6D pose of objects in a video sequence can be labelled by just labelling the first frame, and then transform it according to the camera movement between frames. However, it is often required to also have the segmentation mask for each object, which is much more tedious to label. One approach if the 3D model of the object is available is to leverage it to estimate the segmentation mask by projecting it over the image given the 6D object pose. In any case, synthetic data generation makes easier generating all these types of data at scale

In the last chapter we will try to leverage all the geometric and environment awareness that deep learning architectures assimilate for the 6D object pose estimation problem, in a different but closely related problem: the grasping region estimation on objects from an image.

Chapter

5

Estimating Grasping Regions on Objects based on 6D Pose Estimation Models

As we have seen throughout this thesis, a truly autonomous robot which is able to move and interact in human environments is not something easy to achieve. We have already treated the synthetic data generation as a way of facilitating the application of deep learning-based approaches on computer vision problems, as they are outperforming classical ones. After that, we analysed the 6D object pose estimation problem as a key problem in our target on making a robot able to autonomously interact with objects. Object detection is commonly addressed at the same stage as 6D pose, either directly in its architecture, or through the use of an state-of-the-art detector. So, our next step once the object is located in the space, is to be able to know how to properly grasp it. For that purpose, we will treat in this chapter the problem of the grasping region estimation in objects. After that, only grasp planning would be left to make a robot autonomously able to interact with objects. However, it will remain out of the scope of our work.

5.1 Introduction

The problem of predicting grasping regions on objects given a RGB image is a task in the field of robotics and computer vision that aims to automatically identify the regions on an object that are suitable for a robot to grasp. This is a crucial task for robotic manipulation and grasping, as it enables robots to interact with objects in their environment and perform tasks with them. The goal is to develop algorithms that can analyse an RGB image of an object and predict the most suitable regions for grasping, based on factors such as the shape, texture, and size of the object. This requires a deep understanding of image processing, computer vision, and machine learning techniques, as well as a robust evaluation system to assess the accuracy of the predictions.

This chapter is organized as follows. First, in Section 5.2 some previous works in the field are presented, along with some relevant datasets. After that, our proposal is described in detail in Section 5.3, where modified architectures are also described. Later, in Section 5.4 some experiments are presented to validate the proposal. Finally, in Section 5.5 we draw some conclusions and future works are discussed.

5.2 Related works

In the literature, we mostly distinguish two different grasping types: robotic (grip) and human-like (hand). In the same way, we can find very different datasets depending on the task. The standard dataset for robotic grasping is the Cornell Grasping Dataset, which consist of a set of objects labelled with plausible parallel jaw grips. The Jacquard dataset [Dep18] also contains that information, and its data was labelled through simulations. It is very common to label valid robotic grips through simulators. GraspIt! [Mil04] is probably the most famous one as it was originally proposed in 2004. It also simulates hand grasps.

Indeed, ObMan dataset [Has19] was created with the purpose of training a model for reconstructing hand and object meshes from images during interaction. It is a synthetic dataset, and body poses were recorded via motion capture, while hand poses were generated using GraspIt.

If we move to non-synthetic datasets, we can find GRAB [Tah20], a very complete dataset of full-body sequences interacting with daily objects, which includes heat maps and annotations of hand-object interactions. It served for training a model that generates hand poses to grasp unseen objects. Jiang *et al.* [Jia21] proposed an architecture for that same task. In this case, they divided the problem in: (1) generating a contact map given the model of an object, and (2) generating a hand pose given the object with the contact map.

Corona *et al.* [Cor20] proposed a model for predicting how a human would grab an object (i.e. the hand pose) given just an image. For that purpose, they used GraspIt to manually annotate the YCB objects with some realistic grasps, and they generated an enormous amount of grasping-annotated images just transferring those grasps to the frames of the YCB-Video dataset [Xia18b].

ContactDB [Bra19a] dataset contains grasping information recorded from thermal cameras after a human hand interacting with an object. They established two kinds of grasping purpose: use and hand-off. That subtle difference meant very different grasps in some objects, such as scissors or gamepad. The same authors presented an algorithm [Bra19b] to generate grasps given an object model and its contact map. It consists of refining randomly generated hand poses with GraspIt, trying to match them with the human-generated contact maps from their previous work. The same authors recorded human hand poses with RGB-D cameras for the same set of objects in [Bra20].

ContactOpt [Gra21] proposes an algorithm where, given estimated hand and object poses from a hand-object grasp RGB image, the hand pose is refined in order to match the object pose and its expected grasping areas. To do so, they propose two models: DeepContact and DiffContact. DeepContact estimates contact maps in both the object and the hand from their estimated poses as point clouds using PointNet++ [Qi17]. DiffContact computes contact maps based on vertex distances between current hand and object poses. Then, the hand mesh is iteratively optimized to minimize the difference between the current contact map computed through DiffContact, and the one estimated by DeepContact.

5.3 Proposal

Given the development and success of 6D object pose estimation methods, our proposal consist on leveraging the architectures of these models for a closely related job, as grasping region estimation is. For that purpose, we first propose a naive approximation, which is not based on a 6D pose estimation method. That will help us to establish a base line. In addition to this, we modify two 6D pose methods with completely different approaches.

5.3.1 ContactNet

ContactNet is the network we developed in Chapter 3 for validating our grasping dataset on YCB objects, SynContact. This is a Pix2Pix-based [Iso17] network which will serve us as a naive baseline. Its purpose is performing a domain adaptation between the segmented RGB input image and the segmented grasp region output image (in form of a heat map).

It uses a U-Net [Ron15] as a generator and a PatchGAN classifier as a discriminator. On the one side, PatchGAN is a convolutional network implementing four downsampling blocks and a final 2D convolution. Each downsampling block consists of a 4×4 2D convolution, batch normalization, and leaky ReLu as activation function. Different from a regular discriminator, PatchGAN evaluates the inputs as a grid of patches, where each patch may be “real” or “fake”. Mathematically, this is equivalent to slicing the image into a grid of patches, input each patch to a regular discriminator, and as a result average the results of all patches. On the other side, the U-Net generator is a simple encoder-decoder like architecture.

5.3.2 PVNet

PVNet [Pen19] is one of our choices for trying to leverage 6D pose architectures on grasping estimation. As we saw in the Chapter 4, PVNet learns to regress a certain set of points from known or similar objects through a voting system that makes it remarkably robust against occlusions. Those points have been experimentally set to 8, and they are previously calculated for the 3D model (represented as a point cloud) of the object using the FPS (frames per second) algorithm.

This algorithm selects the set of points that maximizes the distance between them. At implementation level, it starts from the point which represents the average (or centre) of the object, and chooses the furthest point from there. After that, it selects the point which is the furthest with respect to that first point. Then, the following selected point will be the one that maximizes the aggregated distance with respect the ones which have been already selected. We will repeat this process until having the set of points that we need, in this case, 8. In Figure 5.1, the FPS algorithm can be seen applied on several point clouds. The reason why this algorithm is used for this task is that it chooses outstanding points of the object, which tend to be more representative. If we look at the cat in Figure 5.1, we can easily see how legs and ears are selected, which are a good representation for that object.

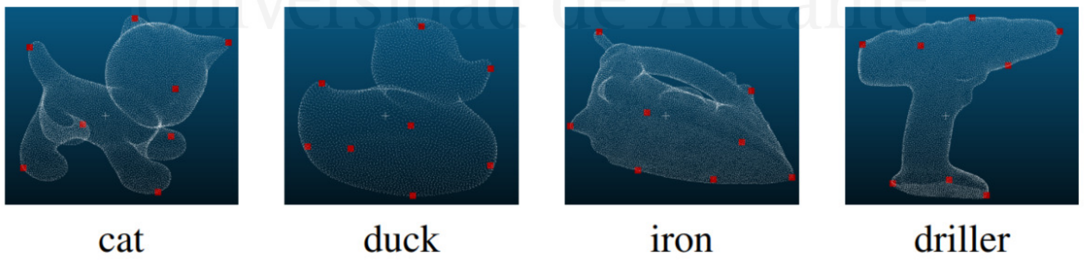


Figure 5.1: FPS algorithm applied on several object models to select 8 points [Pen19].

The modification we propose for this network seeks to leverage the fact that PVNet estimates the 6D pose of the object by regressing keypoints. From the relative position of those keypoints, the 6D pose is computed. So, we can manually choose two or more points that represent our ground-truth grasping region, and provide them to the network as keypoints in the training stage. We can use the FPS algorithm to select the remaining keypoints. PVNet, without further modification, will regress our grasping points, since they are included in the keypoint set. In Figure 5.2 we can see several examples of how we can select our desired grasping points as keypoints (in green), and compute the rest with FPS (in red).

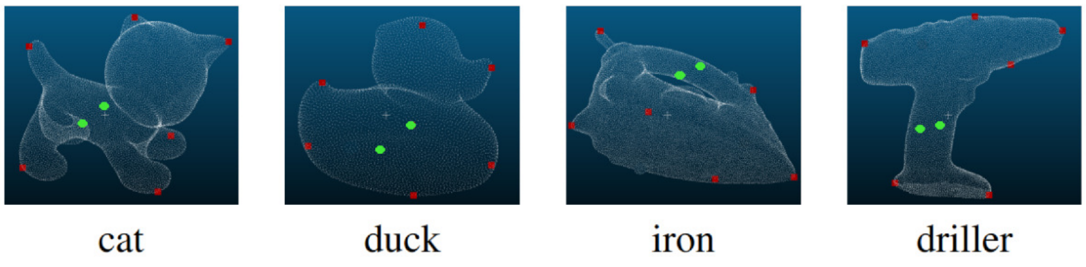


Figure 5.2: Our custom proposal for choosing the keypoints: in green the two points that are used as start, representing contact points of the object.

5.3.3 Pix2Pose

Pix2Pose [Par19] is the other choice to propose a modification for grasping region estimation. This network pipeline, as we saw in the previous chapter, is composed by two passes over the same architecture: the first one is specialised in segmenting objects, and the second one in generating a representation where each pixel of the object is coloured according to its 3D position respect to the camera. After that, this 3D coordinate representation is used with PnP and RANSAC algorithms to obtain the 6D pose of the object. This can be seen in Figure 5.3.

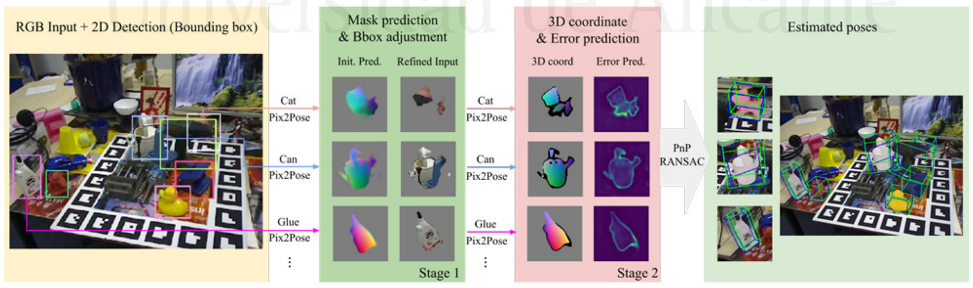


Figure 5.3: Pix2Pose pipeline.

The modification we propose consist on training the second pass of the pipeline to learn a heat map representation of the object, which represents the grasping regions of the object. This proposal is presented in Figure 5.4.

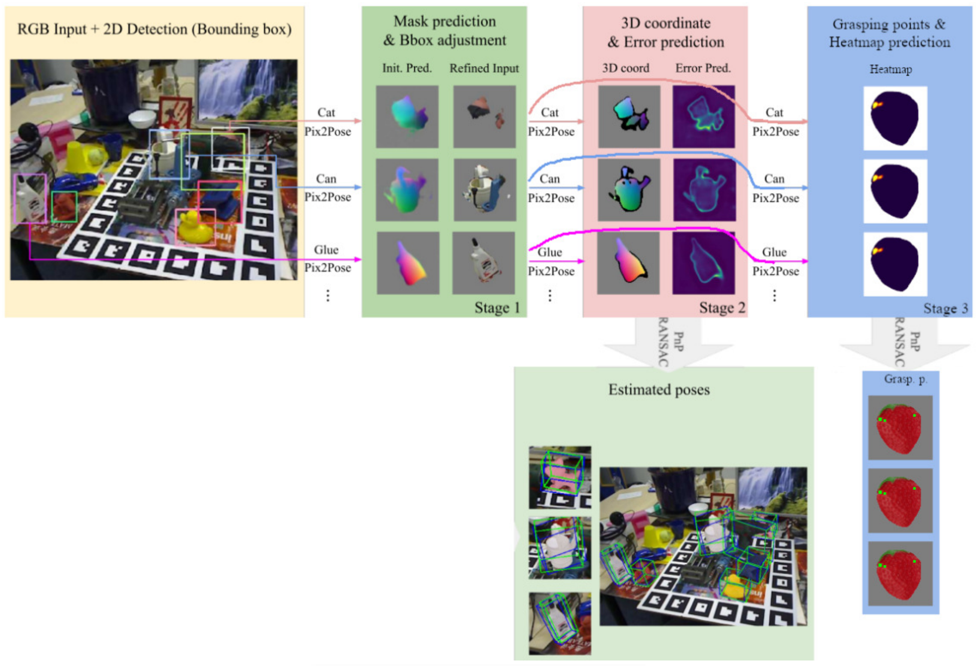


Figure 5.4: Our proposal for Pix2Pose: second pass specialized in obtaining heat maps instead of 3D colour representation.

5.4 Experiments

First, we propose ContactNet as our baseline. Its naive approach will help to establish a comparison point for validating if our hypothesis is promising. We trained the ContactNet with the synthetic generator and the dataset that we introduced in Chapter 3. It is composed of objects centred in 256×256 RGB images. Both RGB images and contact maps have white backgrounds. Thus, we make the model focus only on the object in the image by masking the generated images. ContactNet was trained for 200 epochs with the same learning rate (0.0002) for the generator and discriminator. We used a batch size of 8, and sample shuffling in the training set. In Figure 5.5 we can see heat map predictions (middle row) and ground truth (bottom row) on our SynContact dataset.

We plan to use our SynContact dataset to also test and validate PVNet (with our custom implementation of FPS) and Pix2Pose. In order to be able to compare results, we need to translate the grasping points predicted by PVNet, to a heat map. Moreover, for training Pix2Pose more effectively, transfer learning is proposed so we can keep the first pass of the architecture untrained, as it is given by the authors.

More experiments are currently being developed, but unfortunately we cannot provide more results for the time being.

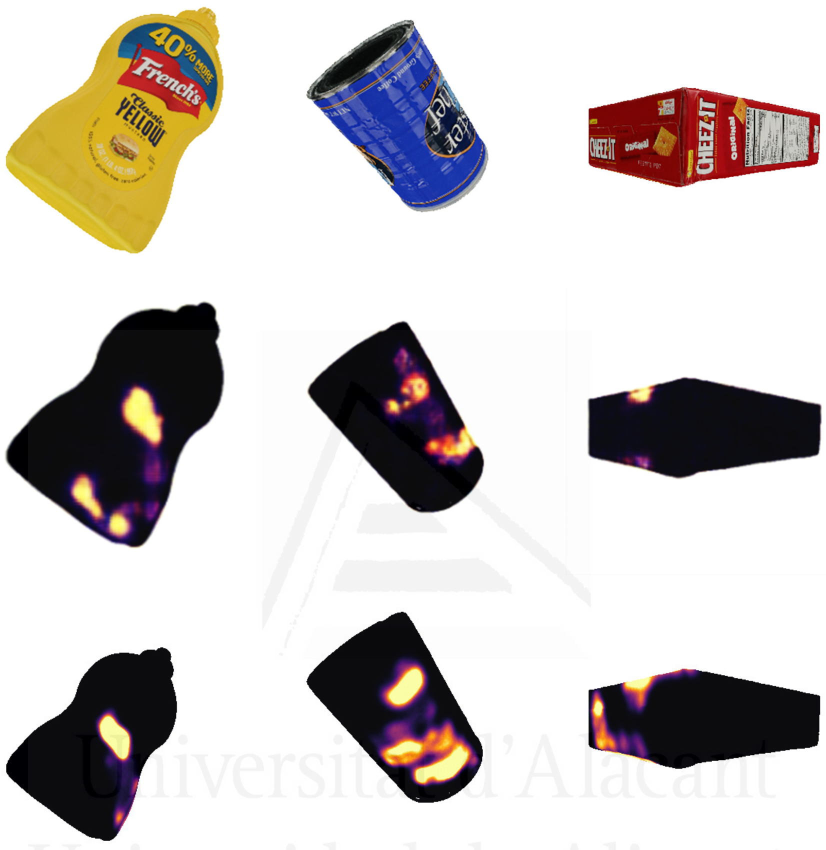


Figure 5.5: At the top row, input images to the ContactNet. At the middle, predicted contact maps. At the bottom row, ground truth contact maps. Regarding the mustard bottle, notice the generated contact map shares similarities with those represented in the ground truth. Moreover, look at the contact map generated for the big cracker box in the borders. Boxes often present contact areas at the edges, as we naturally grip them from the sides.

5.5 Conclusions

We propose the hypothesis of 6D object pose estimation architectures being useful for the grasping region estimation problem. We present a baseline (ContactNet, a naive Pix2pix-based network to perform a domain adaptation between RGB and heat maps) and two modifications to 6D pose architectures: a PVNet version for regressing grasping points instead of keypoints, and a Pix2Pose version for estimating heat maps instead of 3D colour coordinates. We also present some experiments to be performed in order to validate the hypothesis. Nevertheless, we cannot draw any conclusions from the proposal at this stage, since experiments and being developed.

Future work obviously includes finishing these experiments to validate or discard the hypothesis.

Part III

Concluding Remarks, Appendices and Bibliography

Chapter

6

Concluding Remarks

In this thesis, we have treated 4 problems: synthetic data generation for computer vision problems in general, synthetic contact maps generation, 6D Object Pose Estimation and Grasping Regions Estimation. All these problems and their contributions are closely related to computer vision, machine learning, and even computer graphics and virtual reality fields.

First, we presented a virtual reality system, in which a human operator is embodied as a robotic agent using VR setups such as Oculus Rift, Oculus Quest or HTC Vive Pro, for generating automatically annotated synthetic data for various robotic vision tasks. This environment leverages photorealism for bridging the reality gap so that models trained on its simulated data can be transferred to a real-world domain while still generalizing properly. The whole project, with all the aforementioned components (recording/playback, multi-camera, HUD, controller, and robotic pawns) is freely available with an open-source license and detailed documentation so that any researcher can use to generate custom data or even extend it to suit their particular needs. That data generation process was engineered and designed with efficiency and easiness in mind and it outperforms other existing solutions at object, robot, and camera repositioning, and image generation. Furthermore, we presented major improvements to the original tool. It already demonstrated its potential for simulating robotic interactions and generating huge amounts of data that facilitates training data-driven methods for various applications such as semantic segmentation, depth estimation, or object recognition. However, it lacked from the flexibility and modularity for making the tool profitable on a wider range of scenarios, and for a bigger number of researches. So, we worked on decoupling the main data-retrieval system from the main workflow in order to make it much easier and faster to set up and script custom behaviours with Unreal Engine's visual scripting language. Moreover, new kinds of data, such as albedo or shading maps, or more efficient ways of acquire and managing data, such as segmentation masks or skeleton pose retrieval, were also added, along with useful features such as the Python API, useful for Reinforcement Learning.

Second, we improved our original grasping system with a novel grasping logic and using hand tracking instead of Oculus handheld controllers. Using the grasping system, we were able to interact with objects in VR environments. We captured hand-object interactions in forms of contact maps to enable grasp region prediction on objects using learning-based approaches. To ensure our synthetically generated dataset of contact maps is useful, we implemented a Pix2Pix to translate RGB images of objects to heat maps indicating potential grasping areas on the objects. We showed the model can generate plausible grasping spots on unseen objects.

Third, we presented a brief review on the 6D object pose estimation problem, in order to be able to better understand the complexity of this problem from the computer vision point of view. We focused on deep learning approaches, since it is the topic we are addressing in this work, and we analysed in more depth the PVNet and Pix2Pose methods. These methods are important to understand because they were our vehicle for proposing modifications on their architectures to make them able to estimate grasping regions on objects, which is a related and similar problem. We propose a series of experiments and baselines in order to validate our hypothesis.

6.1 Limitations and future works

In each chapter, we found several obstacles which limit the applications of our proposals, or simply side applications or extensions we didn't develop. In this section we collect these aspects for each of them at a glance.

6.1.1 Synthetic Data Generation

As one of the main limitations, we found the simulation of non-rigid objects and deformations when grasping such kind of objects. We have limited ourselves to manipulate non-deformable objects in order not to affect realism, since this is a different approach with a non-haptic manipulation and deformations need to be modelled at the object level. The system would need to implement a custom physics engine in order to model the transformations on deformable objects and be able to record and playback them. Another important shortcoming is the absence of tactile information when grasping objects. We plan to include simulated tactile sensors to provide force data when fingers collide with objects and grasp them instead of providing only visual information. Furthermore, although not strictly a limitation, we are working on making the system able to process URDF to automatically import robots, including their constraints, kinematics, and colliders, in the environment instead of doing that manually for each robot model.

Another possible improvement is providing an additional segmentation layer for objects behind transparent materials. Camera distortion and noise are aspects that may distinguish real from synthetic images. They can be addressed as a post-process, but it could be interesting to parametrize in the tool itself. Other phenomenons, such as camera shake, very common on robotic applications, are not provided or guided by our tool, although they could be simulated through UE4 programming. Talking about new functionalities, the presented Python API for Reinforcement Learning is somewhat preliminary and many more commands and more efficient or comfortable ways of working may be added. Following the OpenAI Gym interface would be ideal to make it compatible with community standards.

6.1.2 Generation of Synthetic Hand-Object Contact Maps

The purpose of our grasping system is being able to naturally interact with objects in virtual environments, so that we can extract accurate, realistic and labelled data which help us to train machine learning models for predicting grasping regions. In that sense, we don't really need the grasp to completely precise in real time. We could just detect the grasp, store the object and hand poses, rebuild the scene offline and refine the grasp as precisely as we need. With that strategy we should be able to keep the grasping logic as simple as possible and boost interaction fluency. That is something that can be improved in the future.

Moreover, a strong limitation of the system is the fact of not being physically rigorous. We trust in the rendering engine physics and focus on being visually realistic. Also the interaction through a VR headset is far from being completely natural, and the way of grabbing an object is affected by that. We try to maximize the success when grasping things in the virtual environment, but maybe that is not the way we would have grasped the object.

6.1.3 6D Object Pose and Grasping Estimation

Results from the presented experiments are pending before we can draw some conclusions and limitations about our hypothesis. Nevertheless, for the future, other 6D object pose estimation architectures may be proposed for grasping estimation, once we have the feedback from our experiments. Transfer learning is another important approach that could be very interesting in order to leverage the knowledge assimilated by a network in a different (but related) task. Some other modifications can be proposed based on these principles.

Appendix

A

Introducción y Conclusión

A.1 Introducción

Este primer capítulo presenta el tema principal de esta tesis. En primer lugar, la Sección A.1.1 establece un marco para la investigación realizada en esta tesis. A continuación, definimos la propuesta principal y los objetivos en la Sección A.1.2. Enumeramos las contribuciones de esta tesis en la Sección A.1.3, y finalmente, en la Sección A.1.4 enumeramos los artículos publicados como resultado de la investigación llevada a cabo en esta tesis.

A.1.1 Motivación

La importancia de la robótica asistencial está llamada a aumentar a medida que evolucione la demografía y envejezca la población. A medida que las personas envejecen, pueden necesitar ayuda para realizar tareas cotidianas como bañarse, vestirse y cocinar, lo que puede suponer una carga para los cuidadores, incluidos los familiares y el personal sanitario. Se prevé que esta demanda de asistencia aumente en los próximos años, con el envejecimiento de la generación del “Baby boom” y el aumento de la esperanza de vida de las personas en muchos países europeos [Eur20] [Fer10].

En este contexto, la robótica asistencial puede desempeñar un papel crucial para ayudar a las personas mayores y mejorar su calidad de vida. Al automatizar muchas de las tareas que de otro modo tendría que realizar un cuidador humano, los robots de asistencia pueden ayudar a aliviar la carga del personal asistencial y proporcionar a las personas mayores un mayor grado de independencia. Además, el uso de robots en la asistencia puede proporcionar un mayor nivel de atención y mejorar la seguridad, ya que los robots pueden estar equipados con sensores y otras tecnologías que pueden ayudar a detectar y responder a posibles peligros.

El desarrollo de esta tecnología es especialmente útil para los pacientes con trastornos cognitivos. Diagnosticar este tipo de trastornos en los ancianos es un reto y en más del 40% de los casos, los médicos desconocen la afección [Cho14]. En EE.UU., más de la mitad de los pacientes con demencia no han recibido una evaluación cognitiva [Kot14]. Esto puede hacer que no se diagnostique la demencia o que se haga con retraso [Bra09], lo que puede complicar el tratamiento de la enfermedad subyacente y plantear problemas de seguridad para los pacientes. Para abordar estas cuestiones, dos proyectos nacionales españoles, RETOGAR[*] y A2HUMPA[†], tienen como objetivo proporcionar atención, seguimiento y asistencia individualizados para mejorar la calidad de vida y el bienestar de las personas mayores, en particular de aquellas con deterioro cognitivo. Esto se alinea con las iniciativas European Horizon, que proponen un sistema de asistencia para apoyar y monitorizar a las personas mayores. Esta tesis pretende contribuir a la predicción precoz de anomalías de comportamiento en las personas mayores mediante el análisis de su interacción con el entorno, ayudando a prevenir resultados no deseados.

El campo de la visión por ordenador ha experimentado un enorme crecimiento en los últimos años, gracias en gran parte a los avances en algoritmos de aprendizaje profundo. Estos algoritmos han permitido a los sistemas de visión por ordenador realizar tareas como el reconocimiento de objetos y la clasificación de imágenes con una precisión impresionante. Sin embargo, aún queda mucho trabajo por hacer para que los sistemas de visión por ordenador sean más ampliamente aplicables en escenarios del mundo real, en particular en el campo de la robótica de asistencia. El uso del aprendizaje profundo en la visión por ordenador tiene el potencial de revolucionar la forma en que los robots interactúan con sus entornos. Con la capacidad de reconocer y clasificar objetos con precisión, los robots pueden estar equipados con los conocimientos necesarios para realizar una amplia gama de tareas, desde la asistencia en las tareas domésticas hasta la prestación de apoyo en entornos médicos e industriales. Sin embargo, para que estos sistemas sean eficaces, deben entrenarse con grandes cantidades de datos. Aquí es donde entra en juego la generación de datos sintéticos. La generación de datos sintéticos es el proceso de creación de muestras de datos artificiales que pueden utilizarse para entrenar algoritmos de visión por ordenador. Estos datos pueden generarse para simular situaciones del mundo real y utilizarse para aumentar los conjuntos de datos existentes, lo que permite entrenar sistemas de visión por ordenador incluso en ausencia de datos del mundo

[*]**RETOGAR: Reto**rno al Ho**gar**. Sistema de mejora de la autonomía de personas con daño cerebral adquirido y dependientes en su integración en la sociedad. Este trabajo ha sido financiado parcialmente por el Gobierno de España a través del proyecto RETOGAR (TIN2016-76515-R), subvencionado por el Fondo Europeo de Desarrollo Regional (FEDER).

[†]**A2HUMPA: A**prendizaje y **A**nálisis de **C**omportamientos **Hu**manos para **M**onitorización, Asistencia **P**ersonalizad**a** y Detección Temprana de Dolencias. Agradecimientos a "A way of making Europe" del Fondo Europeo de Desarrollo Regional (FEDER) y al Ministerio de Ciencia e Innovación (MCIN/AEI/10.13039/501100011033) por apoyar este trabajo bajo el proyecto A2HUMPA (PID2019-104818RB-I00).

real.

En esta tesis doctoral exploraremos la aplicación del aprendizaje profundo a problemas de visión por ordenador en el contexto de la robótica asistencial. Nos centraremos en el desarrollo de técnicas novedosas para la generación de datos sintéticos y en el uso de estas técnicas para entrenar algoritmos de aprendizaje profundo para diversas tareas de visión por computador. Nuestro objetivo es demostrar la eficacia de estas técnicas para mejorar el rendimiento de los sistemas de visión por ordenador en escenarios reales y sentar las bases para futuras investigaciones en este campo. En general, esta investigación contribuirá al avance de la visión por computador y sus aplicaciones en el campo de la robótica asistencial. Aprovechando la potencia del aprendizaje profundo y la generación de datos sintéticos, esperamos acercarnos a un futuro en el que los robots sean capaces de proporcionar una valiosa ayuda a los humanos en una amplia gama de tareas y entornos.

A.1.2 Esta Tesis

Esta tesis está estructurada en dos bloques principales:

1. **Generación de datos sintéticos:** Presenta un entorno para la generación de datos sintéticos completamente anotados a partir de entornos virtuales en UE4, con un novedoso *pipeline* de adquisición de datos. Originalmente, implicaba un flujo de trabajo únicamente para secuencias, pero la herramienta fue ampliada para un uso más flexible. Se pueden generar muchos tipos diferentes de datos, y es útil para una amplia variedad de modelos de aprendizaje automático aplicados a problemas de visión por computador. En esta tesis trataremos el problema de la estimación de regiones de agarre a partir de imágenes RGB, para lo cual implementamos una forma de obtener mapas de contacto como texturas a partir de interacciones virtuales con objetos en VR. Con ello, generamos un *dataset* sintético de mapas de contacto.
2. **Estimación de pose 6D y de regiones de agarre de objetos:** Presenta una *review* sobre modelos de estimación de pose 6D de objetos . Analiza las arquitecturas más relevantes basadas en *deep learning*, con el objetivo de modificar algunas de ellas para aprovechar sus arquitecturas con un problema diferente pero relacionado: la estimación de regiones de agarre sobre objetos. Se proponen tres modelos diferentes.

En la primera parte de esta tesis, el capítulo 2 propone una herramienta completa para generar datos sintéticos a partir de entornos virtuales. Dada la creciente voracidad de datos de los algoritmos de *deep learning*, y cómo se han desarrollado los motores de renderizado para parecerse a la realidad, el uso de datos sintéticos ha sido común en los últimos tiempos para aliviar la tediosa tarea de etiquetar enormes cantidades de datos de imágenes. Esta herramienta pretende aprovechar el desarrollo y la comunidad de uno de los motores de videojuegos más extendidos que existen: *Unreal Engine 4*. La herramienta está construida sobre UE4, proporcionando por un lado todo el sistema de adquisición de datos (incluyendo un *pipeline* de grabación-reproducción que permite almacenar grandes cantidades de imágenes de alta calidad con una alta tasa de *frames* por segundo), y por otro toda la lógica para controlar un agente a través de VR en tiempo real, permitiendo interacciones mano-objeto. También se propone una extensión para permitir el uso de una herramienta tan potente con mayor facilidad dentro de cualquier proyecto de *Unreal Engine*, y con una mayor variedad de aplicaciones. Una de estas aplicaciones es la estimación de la región de agarre sobre objetos, tema que se trata en el capítulo 3. En este capítulo se propone un sistema para generar mapas sintéticos de contacto como texturas a partir de interacciones mano-objeto en VR. Se implementa como un *shader* en UE4 para generar una textura que representa un mapa de calor de contactos. Este sistema se utiliza, junto con el resto de la herramienta, para generar un conjunto de datos de regiones de agarre que es útil para el resto de la tesis.

En la segunda parte de esta tesis, el capítulo 4 presenta una revisión bibliográfica de los *datasets* y métodos en el campo de la estimación de la pose 6D de objetos. Nos centraremos en los métodos basados en técnicas de aprendizaje profundo, ya que construimos nuestras herramientas de generación de datos sintéticos para centrarnos en este tipo de enfoques. Este análisis es útil para comprender mejor los fundamentos de estos modelos y poder, en el capítulo 5, proponer tres modelos para predecir regiones de agarre sobre objetos basados en arquitecturas de estimación de pose de objetos 6D. La motivación detrás de esta propuesta es la relación subyacente entre ambos problemas, en cuanto a la detección de posición y orientación de objetos. Modificamos tres arquitecturas existentes para comparar los resultados con los métodos más avanzados de estimación de regiones de agarre.

A.1.3 Contribuciones

Cada capítulo de esta tesis aporta contribuciones en múltiples campos como la generación de datos sintéticos, interacciones en realidad virtual, estimación de pose 6D de objetos, y generación sintética y estimación de regiones de agarre. Estas contribuciones están respaldadas por publicaciones relevantes en la literatura, que se enumeran en la sección A.1.4. Además, la mayor parte del código fuente implementado en el contexto de esta tesis está disponible públicamente en *GitHub*. Invitamos al lector a consultar los enlaces proporcionados en cada capítulo.

En resumen, las contribuciones de esta tesis son las siguientes:

- Proponemos UnrealROX, una herramienta para generar datos sintéticos a partir de entornos virtuales. Cuenta con un novedoso pipeline de generación de secuencias de imágenes que permite almacenar grandes cantidades de imágenes de alta calidad a una alta velocidad de fotogramas, entre otro tipo de datos. Además, UnrealROX incluye la lógica necesaria para permitir interacciones en tiempo real en entornos de realidad virtual.

- En estrecha relación con la contribución anterior, hemos empaquetado y publicado UnrealROX, con todas sus funciones extendidas, como un *plug-in* de Unreal Engine de código abierto, para que pueda utilizarse fácilmente en cualquier proyecto de investigación. Puede servir para impulsar la investigación en aprendizaje profundo mediante la generación de datos sintéticos en una amplia variedad de campos: Aprendizaje por Refuerzo, Segmentación Semántica, *Relighting*, Estimación de Pose 6D de objetos, predicción de vídeo, entre otros.
- Presentamos una forma de generar mapas de contacto sintéticos a partir de interacciones mano-objeto en realidad virtual, ampliando la posibilidad de aprovechar datos sintéticos en el campo de la estimación de regiones de agarre. Además, generamos y publicamos un conjunto de datos de mapas de contacto sintéticos.
- Proponemos la idea de aprovechar arquitecturas de aprendizaje profundo de estimación de pose 6D de objetos para la estimación de regiones de agarre en objetos. Proponemos tres modelos y los probamos para evaluar su viabilidad.

A.1.4 Publicaciones

Esta tesis es el resultado de un trabajo continuo a lo largo de los últimos años. Dicho trabajo ha ido cristalizándose en forma de publicaciones en congresos y revistas. Gran parte de esta tesis está formada por extractos de las siguientes publicaciones:

Capítulo B: Una herramienta para generar datos sintéticos a partir de entornos virtuales en Unreal Engine

- [Mar19] **P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, A. Jover-Alvarez, S. Orts-Escolano, and J. Garcia-Rodriguez.** «UnrealROX: An eXtremely Photorealistic Virtual Reality Environment for Robotics Simulations and Synthetic Data Generation». *Virtual Reality*, 2019[‡]

- [Mar21] **P. Martinez-Gonzalez, S. Oprea, J. A. Castro-Vargas, A. Garcia-Garcia, S. Orts-Escolano, J. Garcia-Rodriguez, and M. Vincze.** «UnrealROX+: An Improved Tool for Acquiring Synthetic Data from Virtual 3D Environments». *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021[§]

Capítulo C: Generación de mapas de contacto mano-objeto para la predicción

[‡]Código: `https://github.com/3dperceptionlab/unrealrox`
[§]Código: `https://github.com/3dperceptionlab/unrealrox-plus`

de regiones de agarre

- [Mar22] **P. Martinez-Gonzalez, D. Mulero-Perez, S. Oprea, M. Benavent-Lledo, S. Orts-Escolano, and J. Garcia-Rodriguez**. «Synthetic contact maps to predict grasp regions on objects». *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, 2022[¶]

Capítulo E: Estimación de regiones de agarre en objetos basado en modelos de estimación de pose 6D

- [Mar23] **P. Martinez-Gonzalez, S. Thalhammer, D. Mulero-Perez, S. Oprea, S. Orts-Escolano, and J. Garcia-Rodriguez**. «Estimation of Grasping Regions on Objects leveraging 6D Object Pose Estimation Models». *Yet to be published*, 2023

Participación en otras publicaciones fruto de colaboraciones en el contexto de esta tesis

- [Gar18b] **A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez**. «A survey on deep learning techniques for image and video semantic segmentation». *Applied Soft Computing*, Vol. 70, pp. 41–65, Sep. 2018
- [Gar18a] **A. Garcia-Garcia, P. Martinez-Gonzalez, S. Oprea, J. A. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Jover-Alvarez**. «The RobotriX: An eXtremely Photorealistic and Very-Large-Scale Indoor Dataset of Sequences with Robot Trajectories and Interactions». *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6790–6797, IEEE, Oct. 2018
- [Opr19] **S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. A. Castro-Vargas, S. Orts-Escolano, and J. Garcia-Rodriguez**. «A Visually Realistic Grasping System for Object Manipulation and Interaction in Virtual Reality Environments». *Computers and Graphics*, Vol. 83, pp. 77 – 86, 2019
- [Gar19] **A. Garcia-Garcia, P. Martinez-Gonzalez, S. Oprea, J. A. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Jover-Alvarez**. «The RobotriX: A Large-scale Dataset of Embodied Robots in Virtual Reality». *Workshop on 3D Scene Generation*, 2019
- [Cas19] **J. A. Castro-Vargas, A. Garcia-Garcia, S. Oprea, P. Martinez-Gonzalez, and J. Garcia-Rodriguez**. «3D Hand Joints Position Estimation with Graph Convolutional Networks: A GraphHands Baseline». *Advances in Intelligent Systems and Computing*, pp. 551–562, Springer International Publishing, Nov. 2019

[¶]Código: `https://github.com/3dperceptionlab/unrealhandgrasp`

- [Opr20] **S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. A. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Argyros**. «A Review on Deep Learning Techniques for Video Prediction». *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020
- [Cas20] **J.-A. Castro-Vargas, A. Garcia-Garcia, S. Oprea, P. Martinez-Gonzalez, and J. Garcia-Rodriguez**. «3D Hand Joints Position Estimation with Graph Convolutional Networks: A GraphHands Baseline». M. F. Silva, J. Luís Lima, L. P. Reis, A. Sanfeliu, and D. Tardioli, Eds., *Robot 2019: Fourth Iberian Robotics Conference*, pp. 551–562, 2020
- [Gar20] **J. Garcia-Rodriguez, F. Gomez-Donoso, S. Oprea, A. Garcia-Garcia, M. Cazorla, S. Orts-Escolano, Z. Bauer, J. Castro-Vargas, F. Escalona, D. Ivorra-Piqueres, P. Martinez-Gonzalez, E. Aguirre, M. Garcia-Silviente, M. Garcia-Perez, J. M. Cañas, F. Martin-Rico, J. Gines, and F. Rivas-Montero**. «COMBAHO: A deep learning system for integrating brain injury patients in society». *Pattern Recognition Letters*, Vol. 137, pp. 80–90, Sep. 2020
- [Opr21] **S. Oprea, G. Karvounas, P. Martinez-Gonzalez, N. Kyriazis, S. Orts-Escolano, I. Oikonomidis, A. Garcia-Garcia, A. Tsoli, J. Garcia-Rodriguez, and A. Argyros**. «H-GAN: the power of GANs in your Hands». *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021
- [Cas21] **J. A. Castro-Vargas, P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, S. Orts-Escolano, and J. Garcia-Rodriguez**. «Graph Convolutional Neural Networks-based 3D Hand Pose Estimation over Point Clouds». *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, 2021
- [Ben21] **M. Benavent-Lledó, S. Oprea, J. A. Castro-Vargas, P. Martinez-Gonzalez, and J. Garcia-Rodriguez**. «Interaction Estimation in Egocentric Videos via Simultaneous Hand-Object Recognition». *16th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2021)*, pp. 439–448, Springer International Publishing, Sep. 2021

A.2 Conclusiones

En esta tesis, hemos tratado 4 problemas: generación de datos sintéticos para problemas de visión por computador en general, generación de mapas de contacto sintéticos, estimación de pose 6D de objetos, y estimación de regiones de agarre en objetos. Todos estos problemas y sus contribuciones están estrechamente relacionados con la visión por computador, el aprendizaje automático e incluso los gráficos por computador y la realidad virtual.

En primer lugar, presentamos un sistema de realidad virtual, en el que un operador humano se encarna como un agente robótico utilizando configuraciones como Oculus Rift, Oculus Quest o HTC Vive Pro, para generar automáticamente datos sintéticos anotados para diversas tareas de visión por computador. Este entorno aprovecha el fotorrealismo para salvar la brecha entre la realidad y lo artificial, de modo que los modelos entrenados con sus datos simulados puedan transferirse a un dominio del mundo real sin dejar de generalizar adecuadamente. Todo el proyecto, con todos los componentes mencionados (grabación/reproducción, multicámara, HUD, controlador y peones robóticos) está disponible gratuitamente con una licencia de código abierto y documentación detallada para que cualquier investigador pueda utilizarlo para generar sus propios datos, o incluso ampliarlo para adaptarlo a sus necesidades particulares. Este proceso de generación de datos se diseñó pensando en la eficiencia y la facilidad, y supera a otras soluciones existentes en el posicionamiento de objetos, robots y cámaras, así como en la generación de imágenes. Además, presentamos importantes mejoras a la herramienta original. Ya había demostrado su potencial para simular interacciones robóticas y generar enormes cantidades de datos que facilitan el entrenamiento de métodos basados en datos para diversas aplicaciones, como la segmentación semántica, la estimación de la profundidad o el reconocimiento de objetos. Sin embargo, el entorno carecía de la flexibilidad y modularidad necesarias para aprovechar la herramienta en un abanico más amplio de escenarios, y para un mayor número de proyectos de investigación. Por ello, hemos trabajado para desacoplar el sistema principal de recuperación de datos del flujo de trabajo principal, con el fin de que sea mucho más fácil y rápido configurar y programar comportamientos personalizados con el lenguaje de programación visual de Unreal Engine. Además, se han añadido nuevos tipos de datos, como mapas de albedo o de sombreado, o formas más eficientes de adquirir y gestionar datos, como máscaras de segmentación o recuperación de la pose del esqueleto, junto con funciones útiles como la API de Python, útil para el aprendizaje por refuerzo.

En segundo lugar, mejoramos nuestro sistema de agarre original con una lógica de agarre novedosa y utilizando el seguimiento de la mano en lugar de los controladores Oculus. Con el sistema de agarre, hemos podido interactuar con objetos en entornos de realidad virtual. Capturamos las interacciones mano-objeto en forma de mapas de contacto para permitir la predicción de la región de agarre de los objetos mediante enfoques basados en el aprendizaje. Para garantizar la utilidad de nuestro conjunto de datos de mapas de contacto generados sintéticamente, implementamos una red basad en Pix2Pix para traducir las imágenes RGB de los objetos en mapas de calor que indican las posibles zonas de agarre de los objetos. Demostramos que el modelo puede generar puntos de agarre plausibles en objetos desconocidos.

En tercer lugar, presentamos una breve revisión sobre el problema de estimación de pose de objetos en 6D, para poder entender mejor la complejidad de este problema desde el punto de vista de la visión por computador. Nos centramos en los enfoques de aprendizaje profundo, ya que es el tema que abordamos en este trabajo, y analizamos en mayor profundidad los métodos PVNet y Pix2Pose. Estos métodos son importantes de entender porque fueron nuestro vehículo para proponer modificaciones en sus arquitecturas para hacerlos capaces de estimar regiones de agarre sobre objetos, que es un problema relacionado y similar. Proponemos una serie de experimentos y líneas de base para validar nuestra hipótesis.

A.3 Limitaciones y trabajos futuros

En cada capítulo hemos encontrado varios obstáculos que limitan las aplicaciones de nuestras propuestas, o simplemente aplicaciones secundarias o extensiones que no hemos podido desarrollar. En esta sección recogemos de un vistazo estos aspectos para cada uno de ellos.

A.3.1 Generación de datos sintéticos

Como una de las principales limitaciones, encontramos la simulación de objetos no rígidos y las deformaciones al agarrar este tipo de objetos. Nos hemos limitado a manipular objetos no deformables para no afectar al realismo, ya que se trata de un enfoque diferente con una manipulación no háptica, y las deformaciones deberían modelarse a nivel de objeto. El sistema tendría que implementar un motor de físicas personalizado para modelar las transformaciones en objetos deformables y poder grabarlas y reproducirlas. Otra carencia importante es la ausencia de información táctil al agarrar objetos. Tenemos previsto incluir sensores táctiles simulados que proporcionen datos de fuerza cuando los dedos choquen con los objetos y los agarren, en lugar de proporcionar únicamente información visual. Además, aunque no es estrictamente una limitación, estamos trabajando para que el sistema pueda importar robots automáticamente al entorno utilizando ficheros URDF, que representan sus restricciones, cinemática y colisionadores, en lugar de hacerlo manualmente para cada modelo de robot.

Otra posible mejora consistiría en proporcionar una capa de segmentación adicional para los objetos situados detrás de materiales transparentes. Por otro lado, la ausencia de distorsión y ruido en las imágenes obtenidas por nuestras cámaras virtuales son aspectos que pueden alejarlas de la imágenes obtenidas con cámaras reales. Añadir estos factores puede abordarse como un post-proceso, pero podría ser interesante parametrizarlos en la propia herramienta. Otros fenómenos, como la vibración de la cámara, muy común en aplicaciones robóticas, no son proporcionados ni guiados por nuestra herramienta, aunque podrían simularse mediante programación en UE4. Hablando de nuevas funcionalidades, la API de Python para Aprendizaje por Refuerzo presentada en este trabajo es algo preliminar y podrían añadirse muchos más comandos y formas de trabajo más eficientes o cómodas. Seguir la interfaz de OpenAI Gym sería lo ideal para hacerlo compatible con los estándares de la comunidad.

A.3.2 Generación de mapas de contacto sintéticos a partir de interacciones mano-objeto virtuales

El objetivo de nuestro sistema de agarre es poder interactuar de forma natural con objetos en entornos virtuales, de modo que podamos extraer datos precisos, realistas y etiquetados que nos ayuden a entrenar modelos de aprendizaje automático para predecir regiones de agarre. En este sentido, no necesitamos que el agarre sea totalmente preciso en tiempo real. Bastaría con detectar el agarre, almacenar las poses del objeto y la mano, reconstruir la escena fuera de línea y perfeccionar el agarre con la precisión que necesitemos. Con esta estrategia deberíamos ser capaces de mantener la lógica de agarre lo más simple posible y aumentar la fluidez de la interacción. Es algo que puede mejorarse en el futuro.

Además, una fuerte limitación del sistema es el hecho de no ser físicamente riguroso. Confiamos en las físicas del motor de renderizado y nos centramos en que sea visualmente realista. Además, la interacción a través de un casco de realidad virtual dista mucho de ser completamente natural, y la forma de agarrar un objeto se ve afectada por ello. Intentamos maximizar el éxito a la hora de agarrar cosas en el entorno virtual, haciendo que la forma en que cogemos el objeto en el entorno virtual no coincida con cómo lo agarraríamos en la vida real.

A.3.3 Estimación de la pose 6D y regiones de agarre en objetos

Quedan pendientes los resultados de los experimentos para extraer algunas conclusiones y limitaciones sobre nuestra hipótesis. No obstante, en el futuro podrían proponerse otras arquitecturas de estimación de pose 6D de objetos para ser modificadas para estimación de agarre, una vez podamos analizar los resultado obtenidos de nuestros experimentos. El aprendizaje por transferencia es otro enfoque que podría ser muy interesante para aprovechar el conocimiento asimilado por una red en una tarea diferente (pero relacionada). Se pueden proponer otras modificaciones basadas en estos principios.

Appendix

B

Una herramienta para generar datos sintéticos a partir de entornos virtuales en Unreal Engine

Los algoritmos basados en datos han superado a las técnicas tradicionales en casi todos los aspectos de los problemas de visión robótica. Estos algoritmos necesitan grandes cantidades de datos de calidad para poder funcionar correctamente tras su proceso de entrenamiento. Recopilar y anotar esa ingente cantidad de datos en el mundo real es una tarea que requiere mucho tiempo y es propensa a errores. Estos problemas limitan la escala y la calidad. La generación de datos sintéticos es cada vez más popular, ya que su generación es más rápida y su anotación automática. Sin embargo, la mayoría de los conjuntos de datos y entornos actuales carecen de realismo, interacciones y detalles del mundo real. UnrealROX es un entorno construido sobre Unreal Engine 4 que pretende reducir esa brecha de realidad aprovechando escenas interiores hiperrealistas que son exploradas por agentes robóticos que también interactúan con objetos de forma visualmente realista en ese mundo simulado. Unreal Engine renderiza escenas y robots fotorrealistas en un casco de realidad virtual que capta la mirada para que un operador humano pueda mover el robot y utilizar los controladores de las manos robóticas; la información de la escena se vuelca por fotogramas para que pueda reproducirse sin conexión y generar datos brutos y anotaciones de la verdad sobre el terreno. Este entorno de realidad virtual permite a los investigadores de la visión robótica generar datos realistas y visualmente plausibles con la verdad básica completa para una amplia variedad de problemas como la segmentación semántica de clases e instancias, la detección de objetos, la estimación de la profundidad, el agarre visual y la navegación.

B.1 Introducción

Las tareas de robótica basadas en la visión han dado un gran salto adelante debido principalmente al desarrollo de técnicas de aprendizaje automático (p. ej. las arquitecturas profundas [LeC15] como CNN o RNN) que están subiendo continuamente el listón del rendimiento para diversos problemas como la segmentación semántica [Lon15][He17], la estimación de la profundidad [Eig14][Umm17], y el agarre visual [Len15][Lev18] entre otros. Estos métodos basados en datos necesitan grandes cantidades de muestras anotadas para lograr resultados excepcionales. Reunir esa enorme cantidad de imágenes con la verdad sobre el terreno es una tarea tediosa, costosa y, a veces, casi imposible en el mundo real. Por el contrario, los entornos sintéticos agilizan el proceso de generación de datos y suelen ser capaces de proporcionar automáticamente anotaciones para diversas tareas. Por eso, los entornos simulados son cada vez más populares y se utilizan ampliamente para entrenar esos modelos.

El aprendizaje en mundos virtuales o simulados permite una recopilación de datos más rápida, barata y escalable. Sin embargo, los entornos sintéticos se enfrentan a un enorme obstáculo para ser realmente útiles a pesar de sus ventajas inherentes: los modelos entrenados en ese dominio simulado también deben ser capaces de rendir adecuadamente en escenarios de prueba del mundo real, que a menudo presentan numerosas discrepancias entre ellos y sus homólogos sintéticos. Este conjunto de diferencias se conoce como brecha de realidad. En la mayoría de los casos, esta brecha es lo suficientemente grande como para que transferir conocimientos de un dominio a otro sea una tarea extremadamente difícil, bien porque los motores de renderizado no son capaces de producir imágenes como las de los sensores del mundo real (debido al ruido implícito o a la riqueza de la escena), bien porque el comportamiento físico de los elementos de la escena y los sensores no es todo lo preciso que debería.

Para abordar esta brecha entre la realidad y lo generado, se ha demostrado la eficacia de dos métodos: el realismo extremo y la aleatorización de dominios (*domain randomization*). Por un lado, el realismo extremo se refiere al proceso de hacer que la simulación sea lo más parecida posible al entorno real en el que se desplegará el robot [McC16][Gai16]. Eso puede lograrse mediante una combinación de varias técnicas, por ejemplo, renderizado fotorrealista (que implica geometría, texturas e iluminación realistas y también simular el ruido, la distorsión y otros parámetros específicos de la cámara) y física precisa (colisiones complejas con cálculos de alta fidelidad). Por otro lado, la aleatorización de dominios es un tipo de técnica de adaptación de dominios que tiene como objetivo exponer el modelo a una gran variedad de entornos simulados en el momento del entrenamiento, en lugar de a uno solo sintético [Bou17][Tob17c][Tob17a]. Al hacer esto, y si la variabilidad es suficiente, el modelo será capaz de identificar el mundo real como una variación más, siendo así capaz de generalizar a él [Tre18a]. Otra línea de investigación destacable que entrelaza ambos enfoques es el aprendizaje para aumentar imágenes sintéticas realistas con secuencias de transformaciones aleatorias [Pas19].

Propósito: Desarrollar un entorno de realidad virtual extremadamente fotorrealista para generar datos sintéticos para diversas tareas de visión robótica. En dicho entorno, **proponemos una novedosa forma de generar movimientos y agarres**: un operador humano puede encarnarse, en realidad virtual, como un agente robótico dentro de una escena para navegar libremente e interactuar con objetos como si de un robot del mundo real se tratase. Además, liberar la herramienta como una herramienta de código abierto para que cualquier investigador pueda utilizarla fácilmente para sus propios fines.

Nuestro trabajo: Nuestro entorno está construido sobre UE4 para aprovechar sus avanzadas capacidades de VR, renderizado, y físicas. Hemos desarrollado una herramienta que ofrece las siguientes características (1) un sistema de agarre visualmente plausible para la manipulación de robots que es lo suficientemente modular como para ser aplicado a diversas configuraciones de los dedos, (2) rutinas para el control de las manos y los cuerpos robóticos con configuraciones comerciales VR como Oculus Rift, Oculus Quest y HTC Vive Pro, (3) un componente grabador de secuencias para almacenar toda la información sobre la escena, el robot, y las cámaras, mientras que el operador humano está encarnado como un robot, (4) un componente de reproducción de secuencias para reproducir fuera de línea la secuencia previamente grabada y generar datos en bruto como imágenes RGB, de profundidad, normales o de segmentación de instancias, (5) un componente multicámara para facilitar el proceso de colocación de las cámaras y permitir al usuario acoplarlas a articulaciones específicas del robot y configurar sus parámetros (resolución, modelo de ruido, campo de visión), y (6) código abierto, recursos y tutoriales para todos estos componentes y otros subsistemas que los unen. Por último, se propone una extensión de la herramienta para poder aprovechar sus principales características de adquisición de datos de entornos virtuales en UE4 para una gama más amplia de problemas de visión por computador.

Contribución: Una completa herramienta de código abierto de fácil uso para generar una amplia variedad de datos sintéticos a partir de uno de los motores gráficos más extendidos, Unreal Engine. Además, implementamos un novedoso pipeline de grabación-reproducción que permite: (1) superar la sobrecarga de escritura en disco cuando se almacenan varias imágenes por fotograma, (2) garantizar secuencias de alta calidad y alta tasa de fotogramas desacoplando la interacción en tiempo real del renderizado final, (3) utilizar de forma flexible la misma grabación generando diferentes datos modificando fondos, iluminación o texturas.

Appendix

C

Generación de mapas de contacto mano-objeto para la predicción de regiones de agarre

Desde el punto de vista humano, adaptar nuestro agarre a objetos de formas y tamaños diferentes nos parece natural, ya que aprendemos continuamente de la interacción con el entorno. Dominamos la manipulación de objetos, coger un martillo por el mango es casi inconsciente. Sin embargo, realizar la misma tarea desde la perspectiva de una máquina es especialmente difícil, ya que requiere un conocimiento profundo de los objetos y su manipulación. Podríamos aproximarnos a esa comprensión capturando el contacto mano-objeto resultante de los agarres humanos, es decir, mapas de contacto. Esto ya se ha hecho utilizando engorrosos sistemas de captura, y no a gran escala. En este capítulo, simplificamos y aceleramos dicho proceso generando mapas de contacto a partir de nuestra interacción con objetos domésticos en entornos de realidad virtual fotorrealistas. Hasta donde sabemos, somos los primeros en generar mapas de contacto a escala a partir de nuestra interacción en un escenario virtual. Entrenamos un método de traslación de imagen a imagen para predecir las regiones de agarre de los objetos, demostrando la utilidad de nuestros mapas de contacto generados. Nuestro propósito con este trabajo es fomentar aplicaciones en las que sea necesaria la comprensión mano-objeto, por lo que se proporcionan todas las herramientas desarrolladas, el código y los datos generados.

C.1 Introducción

Sabemos cómo agarrar objetos nunca vistos de manera orgánica gracias a nuestra experiencia interactuando con objetos similares en forma, tamaño y textura. Sin embargo, el agarre y la interacción con objetos no son sencillos desde el punto de vista de una máquina. Los robots utilizan cámaras y sensores a bordo para modelar el entorno, identificar objetos y extraer información de su aspecto y forma. En la última década, la visión por ordenador ha experimentado un enorme cambio debido al auge del aprendizaje automático. Los algoritmos de visión clásicos han sido superados en rendimiento y eficiencia por modelos basados en aprendizaje profundo a la hora de abordar problemas como la segmentación semántica y la detección de objetos. La identificación de regiones de agarre en un objeto es otro problema relacionado en el que podemos aprovechar el aprendizaje automático.

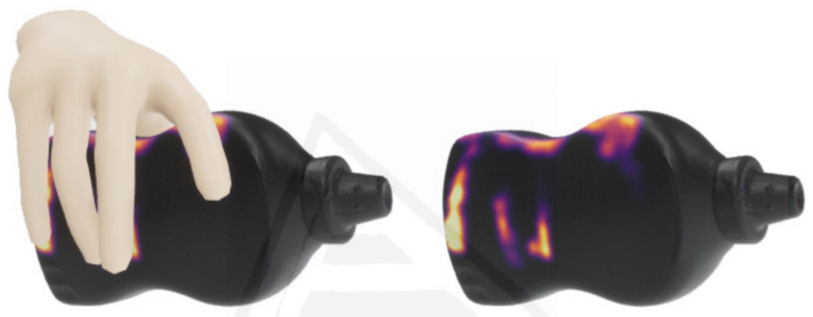


Figure C.1: A la izquierda, una mano virtual agarrando la botella de mostaza del conjunto de datos de objetos YCB; a la derecha, un mapa de contacto generado en la botella de mostaza.

Hay muchas tareas en las que identificar un objeto y determinar cómo agarrarlo resulta útil para un robot autónomo: asistencia, industria, automatización y exploración espacial, entre otras. El efector final habitual que puede verse en estos sistemas es un agarre, mucho más simple que una mano humana. Sin embargo, en algunos escenarios predecir el agarre humano de objetos puede ser interesante para los robots: sujetar y ofrecer un objeto para que un humano lo agarre, supervisión, o simplemente porque el robot tiene un efector final similar a una mano. Este último caso es especialmente útil para robots que operan en el mismo entorno que los humanos.

Propósito: Entrenar un enfoque basado en el aprendizaje para predecir las zonas de agarre de los objetos requiere enormes cantidades de datos etiquetados. En este contexto, utilizamos mapas de contacto para describir la interacción mano-objeto. Los mapas de contacto se obtienen a partir de la interacción repetida de una mano con un objeto determinado. Este es un proceso tedioso y suele requerir configuraciones de *hardware* específicas, como objetos impresos en 3D y cámaras térmicas [Bra19a].

Aquí, la generación de datos sintéticos juega un papel importante y acelera este proceso. Los motores de renderizado modernos nos permiten generar imágenes visualmente realistas con anotaciones completas sobre el terreno. Aprovechando los avances en la generación de datos sintéticos y el seguimiento de las manos en tiempo real, captamos cómo agarramos los objetos domésticos [Cal15] a partir de nuestra interacción en entornos de realidad virtual.

Nuestro trabajo: En primer lugar, modificamos un sistema de interacción mano-objeto existente [Opr19] para utilizar el seguimiento de la mano a bordo de Oculus Quest [Han20] en lugar de los dispositivos de mano, para agarrar objetos de forma natural en entornos de realidad virtual. Esto permite un agarre de objetos más natural aprovechando la interacción con la mano desnuda. A continuación, adaptamos un marco para la generación de datos sintéticos [Mar19, Mar21], para capturar mapas de calor, es decir, mapas de contacto, que representan el contacto de la mano virtual con objetos domésticos del conjunto de datos YCB [Cal15] (véase la Figura 3.1). De forma similar a ContactDB [Bra19a], finalmente utilizamos el conjunto de datos resultante para entrenar un modelo de traducción de imagen a imagen para predecir mapas de contacto a partir de imágenes RGB. Demostramos la utilidad de nuestros datos para entrenar dicho modelo. También desarrollamos herramientas para ampliar el conjunto de datos YCB-Video (YCBv) [Xia18b] con los agarres sintéticos para que también sea útil para los problemas de predicción de agarre.

Contribución: En primer lugar, conseguimos una interacción mano-objeto natural en entornos virtuales, adaptando UnrealGrasp [Opr19] para utilizar el seguimiento de manos de Oculus Quest en lugar de los controladores de mano. En segundo lugar, a partir de la interacción VR, creamos un conjunto de datos compuesto por mapas de calor que representan agarres de objetos y poses de manos. Esto requirió un desarrollo adicional en un marco existente para la generación de datos sintéticos [Mar19, Mar21][*]. Además, entrenamos un modelo de traducción de imagen a imagen en nuestro conjunto de datos para predecir mapas de contacto a partir de imágenes RGB de objetos. Demostramos que los mapas de contacto predichos son coherentes con la forma del objeto, lo que también demuestra la utilidad de nuestros datos. Por último, el conjunto de datos, el modelo y la herramienta generados durante el presente estudio están disponibles en el repositorio *GitHub* [†]. También incluye los *scripts* para ampliar el conjunto de datos YCB Video.

[*]https://github.com/3dperceptionlab/unrealrox-plus
[†]https://github.com/3dperceptionlab/unrealhandgrasp

Appendix D

El problema de la estimación de la pose 6D de objetos

El aprendizaje profundo ha surgido recientemente como un enfoque prometedor para la estimación de la pose de objetos 6D, que implica determinar la posición y orientación de un objeto en el espacio 3D. Esta tarea tiene numerosas aplicaciones en campos como la robótica, la realidad aumentada y la visión por ordenador. Los métodos basados en aprendizaje profundo aprovechan la capacidad de las redes neuronales para aprender representaciones complejas de objetos y escenas, lo que permite una estimación de pose precisa y eficiente.

En esta breve revisión, exploraremos varios métodos basados en el aprendizaje profundo que se han propuesto para la estimación de la pose de objetos 6D. Examinaremos sus fortalezas y limitaciones, así como su rendimiento en conjuntos de datos de referencia. Además, discutiremos los desafíos y las direcciones futuras en este campo, y cómo el aprendizaje profundo puede seguir mejorando la precisión y la eficiencia de la estimación de la pose de objetos 6D.

D.1 Introducción

En los últimos años, el campo de los robots autónomos ha experimentado un crecimiento significativo en su capacidad para agarrar objetos. Para que un robot pueda agarrar con éxito un objeto, necesita comprender de forma inteligente su entorno. Podemos dividir esta enorme tarea en tres más pequeñas (pero también complejas): localización del objeto, estimación de la pose del objeto y estimación del agarre. Sin embargo, muchos métodos de estimación de la pose del objeto no requieren la localización del objeto, sino que realizan ambas tareas conjuntamente.

En este capítulo nos centraremos en los métodos de estimación de la pose 6D de objetos basados en técnicas de aprendizaje profundo. Con el avance de estas técnicas, la precisión y fiabilidad de la estimación de la pose de objetos en 6D ha mejorado significativamente. Este capítulo proporcionará una visión general de los métodos más avanzados en este campo y proporcionará ideas sobre cómo estos métodos se pueden utilizar para mejorar las capacidades de agarre y manipulación de los robots autónomos.

D.2 Conclusión

En este capítulo, hemos explorado el problema de la estimación de la pose 6D de objetos en general, y nos hemos centrado en los métodos basados en el aprendizaje profundo. Se trata de un problema importante y desafiante en visión por computador, clave para que los robots sean totalmente autónomos en entornos humanos reales, interactuando con objetos. Los métodos que hemos revisado están diseñados para predecir la posición y orientación de objetos en el espacio 3D, dada una imagen 2D (con o sin mapa de profundidad).

Los métodos basados en el aprendizaje profundo utilizan grandes cantidades de datos de entrenamiento. Sin embargo, ya hemos hablado de las limitaciones y dificultades de estos métodos, como su sensibilidad a la iluminación, las oclusiones y el ruido, y su dependencia de grandes cantidades de datos de entrenamiento etiquetados. Algunas posibles soluciones a estos retos son el aumentado de datos, el aprendizaje por transferencia, el aprendizaje no supervisado y los datos sintéticos, como propusimos en la primera parte de esta tesis. Todos los conjuntos de datos revisados en la sección 4.3.3 se basan en datos reales, obtenidos a través de cámaras. En este campo, la pose 6D de los objetos en una secuencia de vídeo puede etiquetarse simplemente con el primer fotograma, y luego transformar la pose según el movimiento de la cámara entre fotogramas. Sin embargo, a menudo se requiere disponer también de la máscara de segmentación para cada objeto, lo que resulta mucho más tedioso de etiquetar. Si se dispone del modelo 3D del objeto, se puede aprovechar para estimar la máscara de segmentación proyectando el modelo sobre la imagen dada la pose 6D del objeto. En cualquier caso, la generación de datos sintéticos facilita la generación de todos estos tipos de datos a grandes escalas.

En el último capítulo trataremos de aprovechar todo el conocimiento geométrico y del entorno que las arquitecturas de aprendizaje profundo asimilan para el problema de estimación de la pose del objeto 6D, en un problema diferente pero estrechamente relacionado: la estimación de regiones de agarre en objetos a partir de una imagen.

Appendix

E

Estimación de regiones de agarre en objetos basado en modelos de estimación de pose 6D

Como hemos visto a lo largo de esta tesis, un robot realmente autónomo que sea capaz de moverse e interactuar en entornos humanos no es algo fácil de conseguir. Ya hemos tratado la generación de datos sintéticos como una forma de facilitar la aplicación de enfoques basados en aprendizaje profundo en problemas de visión por computador, ya que estas están superando a las aproximaciones clásicas. Posteriormente, analizamos el problema de la estimación de la pose de objetos en 6D como un problema clave en nuestro objetivo de conseguir un robot capaz de interactuar de forma autónoma con los objetos. La detección de objetos suele abordarse en la misma fase que la pose 6D, ya sea directamente en su arquitectura o mediante el uso previo de un detector. Así, nuestro siguiente paso una vez localizado el objeto en el espacio, es ser capaces de saber cómo agarrarlo adecuadamente. Para ello, trataremos en este capítulo el problema de la estimación de la región de agarre en los objetos. Tras este paso, solo quedaría pendiente la planificación del agarre para que un robot fuera capaz de interactuar de manera autónoma con objetos. Sin embargo, este aspecto no será abordado por quedar fuera de nuestro ámbito de trabajo.

E.1 Introducción

El problema de predecir regiones de agarre en objetos dada una imagen RGB es una tarea en el campo de la robótica y la visión por computador que pretende identificar automáticamente las regiones de un objeto que son adecuadas para que un robot realice un agarre estable. Se trata de una tarea crucial para la manipulación y el agarre robóticos, ya que permite a los robots interactuar con objetos de su entorno y realizar tareas con ellos. El objetivo es desarrollar algoritmos que puedan analizar una imagen RGB de un objeto y predecir las regiones más adecuadas para el agarre, basándose en factores como la forma, la textura y el tamaño del objeto. Para ello se requiere un profundo conocimiento de las técnicas de procesamiento de imágenes, visión por ordenador y aprendizaje automático, así como un sólido sistema de evaluación que permita valorar la exactitud de las predicciones.

En este capítulo se propone la utilización y modificación de arquitecturas diseñadas para la estimación de pose 6D de objetos, y aprovecharlas para el problema de la estimación de regiones de agarre en objetos. La base de esta propuesta radica en la similitud de ambos problemas a la hora de entender el entorno e identificar la forma, geometría y textura de los objetos. Por tanto, se buscará aprovechar arquitecturas exitosas en el campo de la estimación de pose 6D en objetos y adaptarlas para ver si también se comportan adecuadamente para otro problema relacionado.

Se proponen modificaciones sobre dos arquitecturas descritas en el capítulo anterior, junto a una serie de experimentos para validar la propuesta.

Bibliography

[Ahm21] **A. Ahmadyan, L. Zhang, A. Ablavatski, J. Wei, and M. Grundmann**. «Objectron: A Large Scale Dataset of Object-Centric Videos in the Wild With Pose Annotations». *CVPR*, pp. 7822–7831, Computer Vision Foundation / IEEE, 2021. 94, 95

[Ami21] **A. Amini, A. S. Periyasamy, and S. Behnke**. «T6D-Direct: Transformers for Multi-object 6D Pose Direct Regression». C. Bauckhage, J. Gall, and A. G. Schwing, Eds., *Pattern Recognition - 43rd DAGM German Conference, DAGM GCPR 2021, Bonn, Germany, September 28 - October 1, 2021, Proceedings*, pp. 530–544, Springer, 2021. 88

[Asa17] **M. Asadi-Aghbolaghi, A. Clapés, M. Bellantonio, H. J. Escalante, V. Ponce-López, X. Baró, I. Guyon, S. Kasaei, and S. Escalera**. «A Survey on Deep Learning Based Approaches for Action and Gesture Recognition in Image Sequences». *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*, pp. 476–483, 2017. 64

[Bel14] **S. Bell, K. Bala, and N. Snavely**. «Intrinsic images in the wild». *ACM Trans. Graph.*, Vol. 33, No. 4, pp. 159:1–159:12, 2014. 60

[Ben21] **M. Benavent-Lledó, S. Oprea, J. A. Castro-Vargas, P. Martinez-Gonzalez, and J. Garcia-Rodriguez**. «Interaction Estimation in Egocentric Videos via Simultaneous Hand-Object Recognition». *16th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2021)*, pp. 439–448, Springer International Publishing, Sep. 2021. 7, 121

[Bho19] **A. Bhoi**. «Monocular Depth Estimation: A Survey». *arXiv preprint arXiv:1901-09402*, 2019. 51

[Bou17] **K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, *et al.*** «Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping». *arXiv preprint arXiv:1709.07857*, 2017. 13, 126

[Bra09] **A. Bradford, M. E. Kunik, P. Schulz, S. P. Williams, and H. Singh**. «Missed and Delayed Diagnosis of Dementia in Primary Care». *Alzheimer Disease & Associated Disorders*, Vol. 23, No. 4, pp. 306–314, Oct. 2009. 2, 116

[Bra14a] **E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother**. «Learning 6D Object Pose Estimation Using 3D Object Coordinates». D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II*, pp. 536–551, Springer, 2014. 90

[Bra14b] **E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother**.

«Learning 6D Object Pose Estimation Using 3D Object Coordinates». *ECCV (2)*, pp. 536–551, Springer, 2014. 94, 95

[Bra19a] **S. Brahmbhatt, C. Ham, C. C. Kemp, and J. Hays**. «ContactDB: Analyzing and Predicting Grasp Contact via Thermal Imaging». *CVPR*, 2019. 71, 73, 80, 103, 130, 131

[Bra19b] **S. Brahmbhatt, A. Handa, J. Hays, and D. Fox**. «ContactGrasp: Functional Multi-finger Grasp Synthesis from Contact». *IROS*, 2019. 73, 103

[Bra20] **S. Brahmbhatt, C. Tang, C. D. Twigg, C. C. Kemp, and J. Hays**. «ContactPose: A Dataset of Grasps with Object Contact and Hand Pose». *ECCV*, August 2020. 73, 103

[Bro17] **S. Brodeur, E. Perez, A. Anand, F. Golemo, L. Celotti, F. Strub, J. Rouat, H. Larochelle, and A. Courville**. «HoME: A household multimodal environment». *arXiv preprint arXiv:1711.11017*, 2017. 14, 15

[But12] **D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black**. «A naturalistic open source movie for optical flow evaluation». A. Fitzgibbon et al. (Eds.), Ed., *European Conference on Computer Vision (ECCV)*, pp. 611–625, Springer-Verlag, Oct. 2012. 14

[Cal15] **B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar**. «Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set». *IEEE Robotics Automation Magazine*, Vol. 22, No. 3, pp. 36–52, 2015. 64, 71, 79, 131

[Can] **J. Canada**. «Real-time ray tracing in Unreal Engine». 56

[Cas19] **J. A. Castro-Vargas, A. Garcia-Garcia, S. Oprea, P. Martinez-Gonzalez, and J. Garcia-Rodriguez**. «3D Hand Joints Position Estimation with Graph Convolutional Networks: A GraphHands Baseline». *Advances in Intelligent Systems and Computing*, pp. 551–562, Springer International Publishing, Nov. 2019. xiii, 6, 64, 65, 120

[Cas20] **J.-A. Castro-Vargas, A. Garcia-Garcia, S. Oprea, P. Martinez-Gonzalez, and J. Garcia-Rodriguez**. «3D Hand Joints Position Estimation with Graph Convolutional Networks: A GraphHands Baseline». M. F. Silva, J. Luís Lima, L. P. Reis, A. Sanfeliu, and D. Tardioli, Eds., *Robot 2019: Fourth Iberian Robotics Conference*, pp. 551–562, 2020. 6, 121

[Cas21] **J. A. Castro-Vargas, P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, S. Orts-Escolano, and J. Garcia-Rodriguez**. «Graph Convolutional Neural Networks-based 3D Hand Pose Estimation over Point Clouds». *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, 2021. 7, 121

[Cha17] **A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang**. «Matterport3D: Learning from RGB-D Data in Indoor Environments». *International Conference on 3D Vision (3DV)*, 2017. 30

[Che20] **W. Chen, X. Jia, H. J. Chang, J. Duan, and A. Leonardis**. «G2L-Net: Global to Local Network for Real-Time 6D Pose Estimation With Embedding Vector Features». *CVPR*, pp. 4232–4241, Computer Vision Foundation / IEEE, 2020. 93, 94

[Cho14] **K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio**. «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation». *EMNLP*, pp. 1724–1734, 2014. 2, 116

[Coa03] **J. Coakley**. «Reflectance and albedo, surface». *Encyclopedia of the Atmosphere*, pp. 1914–1923, 2003. 60

[Cor20] **E. Corona, A. Pumarola, G. Alenya, F. Moreno-Noguer, and G. Rogez**. «GanHand: Predicting Human Grasp Affordances in Multi-Object Scenes». *CVPR*, June 2020. 73, 102

[Cow] **R. Cowgill**. «Introducing Ray Tracing in UE4». 56

[Dep18] **A. Depierre, E. Dellandrea, and L. Chen**. «Jacquard: A Large Scale Dataset for Robotic Grasp Detection». *IROS*, 2018. 72, 102

[Di21] **Y. Di, F. Manhardt, G. Wang, X. Ji, N. Navab, and F. Tombari**. «SO-Pose: Exploiting Self-Occlusion for Direct 6D Pose Estimation». *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pp. 12376–12385, IEEE, 2021. 91, 92

[Dos15] **A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox**. «FlowNet: Learning Optical Flow with Convolutional Networks». *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 2758–2766, IEEE Computer Society, 2015. 90

[Eig14] **D. Eigen, C. Puhrsch, and R. Fergus**. «Depth Map Prediction From a Single Image using a Multi-scale Deep Network». *NIPS*, pp. 2366–2374, 2014. 12, 51, 126

[Eig15] **D. Eigen and R. Fergus**. «Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture». *ICCV*, pp. 2650–2658, 2015. 51

[Eur20] **Eurostat**. «Population projections in the EU». Apr. 2020. 1, 115

[Fer10] **J.-L. Fernández, C. Parapar, and M. Ruíz**. «Population Ageing: an FGCSIC strategic line». Sep. 2010. 1, 115

[Gai16] **A. Gaidon, Q. Wang, Y. Cabon, and E. Vig**. «Virtual Worlds as Proxy for Multi-object Tracking Analysis». *CVPR*, pp. 4340–4349, 2016. 13, 25, 126

[Gao20] **G. Gao, M. Lauri, Y. Wang, X. Hu, J. Zhang, and S. Frintrop**. «6D Object Pose Regression via Supervised Learning on Point Clouds». *ICRA*, pp. 3643–3649, IEEE, 2020. 93

[Gao21] **G. Gao, M. Lauri, X. Hu, J. Zhang, and S. Frintrop**. «CloudAAE: Learning 6D Object Pose Regression with On-line Data Synthesis on Point Clouds». *ICRA*, pp. 11081–11087, IEEE, 2021. 93, 94

[Gar18a] **A. Garcia-Garcia, P. Martinez-Gonzalez, S. Oprea, J. A. Castro-Vargas, S. Orts-**

Escolano, J. Garcia-Rodriguez, and A. Jover-Alvarez. «The RobotriX: An eXtremely Photorealistic and Very-Large-Scale Indoor Dataset of Sequences with Robot Trajectories and Interactions». *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6790–6797, IEEE, Oct. 2018. 6, 36, 55, 56, 120

[Gar18b] **A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez**. «A survey on deep learning techniques for image and video semantic segmentation». *Applied Soft Computing*, Vol. 70, pp. 41–65, Sep. 2018. 5, 120

[Gar19] **A. Garcia-Garcia, P. Martinez-Gonzalez, S. Oprea, J. A. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Jover-Alvarez**. «The RobotriX: A Large-scale Dataset of Embodied Robots in Virtual Reality». *Workshop on 3D Scene Generation*, 2019. 6, 120

[Gar20] **J. Garcia-Rodriguez, F. Gomez-Donoso, S. Oprea, A. Garcia-Garcia, M. Cazorla, S. Orts-Escolano, Z. Bauer, J. Castro-Vargas, F. Escalona, D. Ivorra-Piqueres, P. Martinez-Gonzalez, E. Aguirre, M. Garcia-Silviente, M. Garcia-Perez, J. M. Cañas, F. Martin-Rico, J. Gines, and F. Rivas-Montero**. «COMBAHO: A deep learning system for integrating brain injury patients in society». *Pattern Recognition Letters*, Vol. 137, pp. 80–90, Sep. 2020. 6, 121

[Gei12a] **A. Geiger, P. Lenz, and R. Urtasun**. «Are we ready for autonomous driving? The KITTI vision benchmark suite». *2012 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, jun 2012. 27

[Gei12b] **A. Geiger, P. Lenz, and R. Urtasun**. «Are we ready for autonomous driving? The KITTI vision benchmark suite». *CVPR*, pp. 3354–3361, IEEE Computer Society, 2012. 94, 95

[Goo14] **I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio**. «Generative Adversarial Nets». *NeurIPS*, pp. 2672–2680, 2014. 23

[Gra21] **P. Grady, C. Tang, C. D. Twigg, M. Vo, S. Brahmbhatt, and C. C. Kemp**. «ContactOpt: Optimizing Contact to Improve Grasps». *CVPR*, 2021. 73, 103

[Han20] **S. Han, B. Liu, R. Cabezas, C. D. Twigg, P. Zhang, J. Petkau, T.-H. Yu, C.-J. Tai, M. Akbay, Z. Wang, A. Nitzan, G. Dong, Y. Ye, L. Tao, C. Wan, and R. Wang**. «MEgATrack: Monochrome Egocentric Articulated Hand-Tracking for Virtual Reality». *ACM Transactions on Graphics (TOG), SIGGRAPH*, Vol. 39, No. 4, July 2020. 71, 73, 131

[Has19] **Y. Hasson, G. Varol, D. Tzionas, I. Kalevatykh, M. J. Black, I. Laptev, and C. Schmid**. «Learning joint reconstruction of hands and manipulated objects». *CVPR*, 2019. 72, 102

[He17] **K. He, G. Gkioxari, P. Dollár, and R. Girshick**. «Mask R-CNN». *CVPR*, pp. 2961–2969, 2017. 12, 126

[He20] **Y. He, W. Sun, H. Huang, J. Liu, H. Fan, and J. Sun**. «PVN3D: A Deep Point-Wise 3D Keypoints Voting Network for 6DoF Pose Estimation». *CVPR*, pp. 11629–11638, Computer Vision Foundation / IEEE, 2020. 93, 94

[Hin12a] **S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. R. Bradski, K. Konolige, and N. Navab.** «Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes». *ACCV (1)*, pp. 548–562, Springer, 2012. 91

[Hin12b] **S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. R. Bradski, K. Konolige, and N. Navab.** «Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes». *ACCV (1)*, pp. 548–562, Springer, 2012. 94, 95

[Hod16] **T. Hodan, J. Matas, and S. Obdrzálek.** «On Evaluation of 6D Object Pose Estimation». *ECCV Workshops (3)*, pp. 606–619, 2016. 91

[Hod18] **T. Hodaň, F. Michel, E. Brachmann, W. Kehl, A. Glent Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, C. Sahin, F. Manhardt, F. Tombari, T.-K. Kim, J. Matas, and C. Rother.** «BOP: Benchmark for 6D Object Pose Estimation». *ECCV*, 2018. 65, 66, 79

[Hua21] **W. Hua, Z. Zhou, J. Wu, H. Huang, Y. Wang, and R. Xiong.** «REDE: End-to-End Object 6D Pose Robust Estimation Using Differentiable Outliers Elimination». *IEEE Robotics Autom. Lett.*, Vol. 6, No. 2, pp. 2886–2893, 2021. 94

[Hwa20] **H. Hwang, C. Jang, G. Park, J. Cho, and I. Kim.** «ElderSim: A Synthetic Data Generation Platform for Human Action Recognition in Eldercare Applications». *CoRR*, Vol. abs/2010.14742, 2020. 16

[Iof15] **S. Ioffe and C. Szegedy.** «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift». *ICML*, pp. 448–456, JMLR.org, 2015. 98

[Iso17] **P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros.** «Image-to-Image Translation with Conditional Adversarial Networks». *CVPR*, 2017. 80, 103

[Jia21] **H. Jiang, S. Liu, J. Wang, and X. Wang.** «Hand-Object Contact Consistency Reasoning for Human Grasps Generation». *ICCV*, pp. 11107–11116, October 2021. 73, 102

[Joh17] **M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan.** «Driving in the Matrix: Can virtual worlds replace human-generated annotations for real world tasks?». *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017. 27

[Jov19] **A. Jover-Alvarez and P. Martinez-Gonzalez.** «Synthetic Data Generation for Deep Learning-based Semantic Segmentation». 2019. 18

[Keh17] **W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab.** «SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again». *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 1530–1538, IEEE Computer Society, 2017. 90, 92

[Kol17] **E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi.** «AI2-THOR: An interactive 3d environment for visual AI». *arXiv preprint arXiv:1712.05474*,

2017. xi, 15, 29, 33

[Kot14] **V. Kotagal, K. M. Langa, B. L. Plassman, G. G. Fisher, B. J. Giordani, R. B. Wallace, J. R. Burke, D. C. Steffens, M. Kabeto, R. L. Albin, and N. L. Foster**. «Factors associated with cognitive evaluations in the United States». *Neurology*, Vol. 84, No. 1, pp. 64–71, Nov. 2014. 2, 116

[Kri12] **A. Krizhevsky, I. Sutskever, and G. E. Hinton**. «ImageNet Classification with Deep Convolutional Neural Networks». *NIPS*, pp. 1106–1114, 2012. 89

[Lai16] **I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab**. «Deeper depth prediction with fully convolutional residual networks». *3DV*, pp. 239–248, 2016. 51

[LeC15] **Y. LeCun, Y. Bengio, and G. E. Hinton**. «Deep learning». *Nature*, Vol. 521, No. 7553, pp. 436–444, 2015. 12, 126

[Led17] **C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi**. «Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network». 5 2017. xi, 23, 24

[Len15] **I. Lenz, H. Lee, and A. Saxena**. «Deep Learning for Detecting Robotic Grasps». *IJRR*, Vol. 34, No. 4-5, pp. 705–724, 2015. 12, 126

[Lev18] **S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen**. «Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection». *The International Journal of Robotics Research*, Vol. 37, No. 4-5, pp. 421–436, 2018. 12, 126

[Li18] **Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox**. «DeepIM: Deep Iterative Matching for 6D Pose Estimation». V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VI*, pp. 695–711, Springer, 2018. 90, 92

[Liu17] **Y. Liu, P. Ramachandran, Q. Liu, and J. Peng**. «Stein Variational Policy Gradient». 2017. 28

[Liu21] **X. Liu, S. Iwase, and K. M. Kitani**. «KDFNet: Learning Keypoint Distance Field for 6D Object Pose Estimation». *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021*, pp. 4631–4638, IEEE, 2021. 91, 92

[Lon15] **J. Long, E. Shelhamer, and T. Darrell**. «Fully Convolutional Networks for Semantic Segmentation». *CVPR*, pp. 3431–3440, 2015. 12, 126

[Loo] **T. Looman**. «VR Template». 42

[Mah17] **J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg**. «Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics». *Robotics: Science and Systems (RSS)*, 2017. 14

[Mak21] **V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State**. «Isaac Gym:

High Performance GPU-Based Physics Simulation For Robot Learning». *CoRR*, Vol. abs/2108.10470, 2021. 17

[Mar19] **P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, A. Jover-Alvarez, S. Orts-Escolano, and J. Garcia-Rodriguez**. «UnrealROX: An eXtremely Photorealistic Virtual Reality Environment for Robotics Simulations and Synthetic Data Generation». *Virtual Reality*, 2019. 5, 55, 64, 71, 72, 119, 131

[Mar21] **P. Martinez-Gonzalez, S. Oprea, J. A. Castro-Vargas, A. Garcia-Garcia, S. Orts-Escolano, J. Garcia-Rodriguez, and M. Vincze**. «UnrealROX+: An Improved Tool for Acquiring Synthetic Data from Virtual 3D Environments». *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021. 5, 71, 72, 77, 119, 131

[Mar22] **P. Martinez-Gonzalez, D. Mulero-Perez, S. Oprea, M. Benavent-Lledo, S. Orts-Escolano, and J. Garcia-Rodriguez**. «Synthetic contact maps to predict grasp regions on objects». *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, 2022. 5, 120

[Mar23] **P. Martinez-Gonzalez, S. Thalhammer, D. Mulero-Perez, S. Oprea, S. Orts-Escolano, and J. Garcia-Rodriguez**. «Estimation of Grasping Regions on Objects leveraging 6D Object Pose Estimation Models». *Yet to be published*, 2023. 5, 120

[McC16] **J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison**. «Scenenet RGB-D: 5M Photorealistic Images of Synthetic Indoor Trajectories with Ground Truth». *arXiv preprint arXiv:1612.05079*, 2016. 13, 126

[Meh19] **B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull**. «Active Domain Randomization». 2019. 27, 28

[Mil04] **A. Miller and P. Allen**. «Graspit! A versatile simulator for robotic grasping». *IEEE Robotics Automation Magazine*, 2004. 72, 102

[Mou17] **A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka**. «3D Bounding Box Estimation Using Deep Learning and Geometry». *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 5632–5640, IEEE Computer Society, 2017. 87

[Nat12] **P. K. Nathan Silberman, Derek Hoiem and R. Fergus**. «Indoor Segmentation and Support Inference from RGBD Images». *ECCV*, pp. 746–760, 2012. 51, 53

[Nej22a] **N. Nejatishahidin and P. Fayyazsanavi**. «Review on 6D Object Pose Estimation With the Focus on Indoor Scene Understanding». *Adv. Artif. Intell. Mach. Learn.*, Vol. 2, No. 4, pp. 588–613, 2022. xiii, 87, 88

[Nej22b] **N. Nejatishahidin, P. Fayyazsanavi, and J. Kosecka**. «Object Pose Estimation using Mid-level Visual Representations». *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2022, Kyoto, Japan, October 23-27, 2022*, pp. 13105–13111, IEEE, 2022. 87

[NVI18] **NVIDIA**. «NVIDIA TURING GPU ARCHITECTURE». 9 2018. 23

[Ocu16] **Oculus**. «Oculus First Contact». `https://www.oculus.com/experiences/rift/1217155751659625/`. Dec. 2016. Accessed: 2019-04-11. 42

[Ocu17] **Oculus**. «Distance Grab Sample Now Available in Oculus Unity Sample Framework». `https://developer.oculus.com/blog/distance-grab-sample-now-available-in-oculus-unity-sample-framework/`. 2017. Accessed: 2019-04-11. 42

[Opr19] **S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. A. Castro-Vargas, S. Orts-Escolano, and J. Garcia-Rodriguez**. «A Visually Realistic Grasping System for Object Manipulation and Interaction in Virtual Reality Environments». *Computers and Graphics*, Vol. 83, pp. 77 – 86, 2019. 6, 25, 62, 71, 72, 74, 75, 76, 82, 120, 131

[Opr20] **S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. A. Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and A. Argyros**. «A Review on Deep Learning Techniques for Video Prediction». *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020. 6, 120

[Opr21] **S. Oprea, G. Karvounas, P. Martinez-Gonzalez, N. Kyriazis, S. Orts-Escolano, I. Oikonomidis, A. Garcia-Garcia, A. Tsoli, J. Garcia-Rodriguez, and A. Argyros**. «H-GAN: the power of GANs in your Hands». *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021. 6, 121

[Par19] **K. Park, T. Patten, and M. Vincze**. «Pix2Pose: Pix2Pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation». *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019. xiii, 64, 66, 90, 92, 95, 98, 100, 105

[Pas19] **A. Pashevich, R. Strudel, I. Kalevatykh, I. Laptev, and C. Schmid**. «Learning to Augment Synthetic Images for Sim2Real Policy Transfer». *arXiv preprint arXiv:1903.07740*, 2019. 13, 127

[Pen19] **S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao**. «PVNet: Pixel-Wise Voting Network for 6DoF Pose Estimation». *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 4561–4570, Computer Vision Foundation / IEEE, 2019. xiii, 90, 92, 95, 96, 104

[Pra18] **A. Prakash, S. Boochoon, M. Brophy, D. Acuna, E. Cameracci, G. State, O. Shapira, and S. Birchfield**. «Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data». 2018. xi, 25, 27

[Pui18] **X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba**. «Virtual-Home: Simulating Household Activities via Programs». *CVPR*, pp. 8494–8502, IEEE Computer Society, 2018. 16

[Qi17] **C. R. Qi, L. Yi, H. Su, and L. J. Guibas**. «PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space». *NIPS*, p. 5105–5114, 2017. 73, 103

[Qiu16] **W. Qiu and A. Yuille**. «UnrealCV: Connecting Computer Vision to Unreal Engine». *ECCV*, pp. 909–916, Springer, 2016. 16, 32, 35, 59

[Qiu17a] **W. Qiu, F. Zhong, Y. Zhang, S. Qiao, Z. Xiao, T. S. Kim, and Y. Wang**. «UnrealCV: Virtual Worlds for Computer Vision». *ACM Multimedia Open Source Software Competition*, pp. 1221–1224, ACM, 2017. 16

[Qiu17b] **W. Qiu, F. Zhong, Y. Zhang, S. Qiao, Z. Xiao, T. Soo Kim, and Y. Wang**. «UnrealCV: Virtual Worlds for Computer Vision». pp. 1221–1224, 10 2017. 32

[Red16] **J. Redmon, S. Divvala, R. Girshick, and A. Farhadi.** «You only look once: Unified, real-time object detection». *CVPR*, 2016. 52

[Red17] **J. Redmon and A. Farhadi.** «YOLO9000: Better, faster, stronger». *CVPR*, Vol. 2017-Janua, pp. 6517–6525, 2017. 52

[Rob22] **M. Roberts, Q. Leboutet, R. Prakash, R. Wang, H. Zhang, R. Tang, M. Ferragut, S. Leutenegger, S. R. Richter, V. Koltun, M. Müller, and G. Ros.** «SPEAR: A Simulator for Photorealistic Embodied AI Research». `http://github.com/isl-org/spear`. 2022. 17

[Ron15] **O. Ronneberger, P. Fischer, and T. Brox.** «U-Net: Convolutional Networks for Biomedical Image Segmentation». *MICCAI*, 2015. 80, 81, 98, 103

[Ros16] **G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez.** «The Synthia Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes». *CVPR*, pp. 3234–3243, 2016. 14

[Sal16] **T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen.** «Improved Techniques for Training GANs». 6 2016. 23

[Sat97] **Y. Sato, M. D. Wheeler, and K. Ikeuchi.** «Object shape and reflectance modeling from observation». *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 379–387, 1997. 60

[Sav17] **M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun.** «MINOS: Multimodal indoor simulator for navigation in complex environments». *arXiv preprint arXiv:1712.03931*, 2017. xi, 15, 30, 34

[Sav19] **M. Savva, J. Malik, D. Parikh, D. Batra, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, and V. Koltun.** «Habitat: A Platform for Embodied AI Research». *ICCV*, pp. 9338–9346, IEEE, 2019. 16

[Shi21] **Y. Shi, J. Huang, X. Xu, Y. Zhang, and K. Xu.** «StablePose: Learning 6D Object Poses From Geometrically Stable Patches». *CVPR*, pp. 15222–15231, Computer Vision Foundation / IEEE, 2021. 93, 94

[Sim15] **K. Simonyan and A. Zisserman.** «Very Deep Convolutional Networks for Large-Scale Image Recognition». Y. Bengio and Y. LeCun, Eds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 90

[Son17] **S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. A. Funkhouser.** «Semantic Scene Completion from a Single Depth Image». *CVPR*, pp. 190–198, 2017. 15, 30

[Son20] **C. Song, J. Song, and Q. Huang.** «HybridPose: 6D Object Pose Estimation Under Hybrid Representations». *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pp. 428–437, Computer Vision Foundation / IEEE, 2020. 90, 92

[Sun18a] **X. Sun, J. Wu, X. Zhang, Z. Zhang, C. Zhang, T. Xue, J. B. Tenenbaum, and W. T. Freeman.** «Pix3D: Dataset and Methods for Single-Image 3D Shape Modeling». *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 2974–2983, Computer Vision Foundation / IEEE Computer Society, 2018. 87

[Sun18b] **M. Sundermeyer, Z. Marton, M. Durner, M. Brucker, and R. Triebel**. «Implicit 3D Orientation Learning for 6D Object Detection from RGB Images». *ECCV (6)*, pp. 712–729, Springer, 2018. 98

[Sut18] **R. S. Sutton and A. G. Barto**. «Reinforcement Learning: An Introduction». The MIT Press, second Ed., 2018. 28

[Sze17] **C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi**. «Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning». S. Singh and S. Markovitch, Eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 4278–4284, AAAI Press, 2017. 90

[Tah20] **O. Taheri, N. Ghorbani, M. J. Black, and D. Tzionas**. «GRAB: A Dataset of Whole-Body Human Grasping of Objects». *ECCV*, 2020. 72, 102

[Tek18] **B. Tekin, S. N. Sinha, and P. Fua**. «Real-Time Seamless Single Shot 6D Object Pose Prediction». *CVPR*, pp. 292–301, June 2018. 52, 53, 54

[Tob17a] **J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel**. «Domain randomization for transferring deep neural networks from simulation to the real world». *IROS*, pp. 23–30, 2017. 13, 126

[Tob17b] **J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel**. «Domain randomization for transferring deep neural networks from simulation to the real world». *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep 2017. 24

[Tob17c] **J. Tobin, W. Zaremba, and P. Abbeel**. «Domain Randomization and Generative Models for Robotic Grasping». *arXiv preprint arXiv:1710.06425*, 2017. 13, 126

[To18a] **T. To, J. Tremblay, D. McKay, Y. Yamaguchi, K. Leung, A. Balanon, J. Cheng, and S. Birchfield**. «NDDS: NVIDIA Deep Learning Dataset Synthesizer». 2018. `https://github.com/NVIDIA/Dataset_Synthesizer`. 16

[To18b] **T. To, J. Tremblay, D. McKay, Y. Yamaguchi, K. Leung, A. Balanon, J. Cheng, and S. Birchfield**. «NDDS: NVIDIA Deep Learning Dataset Synthesizer». 2018. `https://github.com/NVIDIA/Dataset_Synthesizer`. 33

[Tre18a] **J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield**. «Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization». *arXiv preprint arXiv:1804.06516*, 2018. 13, 126

[Tre18b] **J. Tremblay, T. To, and S. Birchfield**. «Falling Things: A Synthetic Dataset for 3D Object Detection and Pose Estimation». *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Jun 2018. 34

[Tre18c] **J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield**. «Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects».

2018. 34

[Umm17] **B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox**. «Demon: Depth and Motion Network for Learning Monocular Stereo». *CVPR*, pp. 5038–5047, 2017. 12, 126

[Wan19] **C. Wang, D. Xu, Y. Zhu, R. M. Martin, C. Lu, L. Fei-Fei, and S. Savarese**. «DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion». *CVPR*, pp. 3343–3352, Computer Vision Foundation / IEEE, 2019. 92

[Wan20] **G. Wang, F. Manhardt, J. Shao, X. Ji, N. Navab, and F. Tombari**. «Self6D: Self-supervised Monocular 6D Object Pose Estimation». A. Vedaldi, H. Bischof, T. Brox, and J. Frahm, Eds., *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, pp. 108–125, Springer, 2020. 91, 92

[Wan21] **G. Wang, F. Manhardt, F. Tombari, and X. Ji**. «GDR-Net: Geometry-Guided Direct Regression Network for Monocular 6D Object Pose Estimation». *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pp. 16611–16621, Computer Vision Foundation / IEEE, 2021. xiii, 91, 92

[Wen21] **B. Wen and K. E. Bekris**. «BundleTrack: 6D Pose Tracking for Novel Objects without Instance or Category-Level 3D Models». *IROS*, pp. 8067–8074, IEEE, 2021. 94

[Wor95] **M. F. Worboys**. «GIS: A Computer Science Perspective». p. 232, 10 1995. 24

[Wu18] **Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian**. «Building Generalizable Agents with a Realistic and Rich 3D Environment». 2018. xi, 31, 35

[Xia18a] **F. Xia, A. R. Zamir, Z.-Y. He, A. Sax, J. Malik, and S. Savarese**. «Gibson Env: Real-world Perception for Embodied Agents». *CVPR*, 2018. xi, 15, 29, 30, 31, 32

[Xia18b] **Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox**. «PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes». *Robotics: Science and Systems (RSS)*, 2018. 71, 73, 78, 79, 94, 95, 103, 131

[Xu18] **D. Xu, W. Wang, H. Tang, H. Liu, N. Sebe, and E. Ricci**. «Structured attention guided convolutional neural fields for monocular depth estimation». *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3917–3925, 2018. 51

[Yan18] **C. Yan, D. Misra, A. Bennnett, A. Walsman, Y. Bisk, and Y. Artzi**. «CHALET: Cornell house agent learning environment». *arXiv preprint arXiv:1801.07357*, 2018. 14

[Yan21] **Z. Yang, X. Yu, and Y. Yang**. «DSC-PoseNet: Learning 6DoF Object Pose Estimation via Dual-Scale Consistency». *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pp. 3907–3916, Computer Vision Foundation / IEEE, 2021. 91, 92

[Yu19] **Y. Yu and W. A. P. Smith**. «InverseRenderNet: Learning Single Image Inverse Rendering». *CVPR*, pp. 3155–3164, Computer Vision Foundation / IEEE, 2019. 60

[Zhu22] **Y. Zhu, M. Li, W. Yao, and C. Chen**. «A Review of 6D Object Pose Estimation». *2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, pp. 1647–1655, 2022. xv, 89, 92, 94, 95