UNIVERSITY OF GENOVA

PHD PROGRAM IN BIOENGINEERING AND ROBOTICS

Robotic Perception and Manipulation: Leveraging Deep Learning Methods for Efficient Instance Segmentation and Multi-fingered Grasping

by

Federico Ceola

Thesis submitted for the degree of *Doctor of Philosophy* (36° cycle)

March 2024

Dr. Lorenzo Natale — Supervisor

Prof. Lorenzo Rosasco — Supervisor

Prof. Paolo Massobrio — Head of the PhD program

Thesis Reviewers:

Prof. Georgia Chalvatzaki, *Technische Universität Darmstadt* — External examiner

Prof. Marcello Restelli, *Politecnico di Milano* — External examiner

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

I would like to dedicate this Thesis to the people who supported me during the Ph.D.
Your kind words have been the best relief during hard times.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Federico Ceola
March 2024

Acknowledgements

First and foremost, I would like to thank Dr. Lorenzo Natale for giving me the opportunity to pursue the Ph.D. in the Humanoid Sensing and Perception group at the Italian Institute of Technology. I am deeply grateful for the guidance and for the freedom to pursue my research ideas over the last three years.

A special thanks to Prof. Lorenzo Rosasco for giving me the opportunity to be part of the MaLGa center at the University of Genova. The exposure to the theoretical aspects of Machine Learning and the research discussions enhanced the level of the work done during my Ph.D.

I would like to thank Prof. Niko Sünderhauf and Dr. Krishan Rana for hosting me at the QUT Centre for Robotics. The months spent at the Queensland University of Technology enriched me both personally and professionally, giving me different perspectives about my research.

I am grateful to Dr. Elisa Maiettini and Dr. Giulia Pasquale for mentoring me. Thanks for guiding and supporting me throughout the challenges of the Ph.D. journey.

Finally, I would like to thank all the colleagues and friends that I met in the HSP, MaLGa and QCR groups during the Ph.D. Thank you for all the chats, the shared experiences, and the kind moments spent together.

Abstract

The ability to adapt to perceive and manipulate novel objects is an important requirement for robots operating in unstructured dynamically-changing environments like the ones we live in. Autonomous perception and manipulation of objects in the environment surrounding the robot requires processing sensor data, including images, depth information and tactile feedback. Extracting meaningful semantic and geometric information from such data is per se a challenging open problem, which becomes even more pronounced in the considered scenario. In this setting, the target task of the robot may be not known in advance, requiring continuous adaptation of the robot perception system and control policies.

The Deep Learning breakthrough provided great improvements both in the Computer Vision literature and on some open problems in robotics. While these approaches have shown huge potential to be applied in robotic tasks and to overcome some problems related to the use of classical methods, their application is constrained by some limitations which are intrinsic in the Deep Learning based approaches, such as the requirement of huge amounts of training data and the need of long training sessions to optimize such models. The aim of this Thesis is to overcome these limitations, allowing robots to perform tasks that would not be achievable leveraging only classical methods.

The proposed methods aim at making Deep Learning approaches for the visual task of instance segmentation and for multi-fingered grasping suitable for training on real robotic platforms. In this perspective, I firstly proposed a hybrid method that leverages a pre-trained Convolutional Neural Network for feature extraction and Kernel-based classifiers for fast adaptation of an instance segmentation model in the presence of novel objects or different visual domains. Secondly, I proposed a Residual Reinforcement Learning method with the purpose of learning multi-fingered grasping of novel objects on the real robot. This relies on a policy pre-trained in simulation with a Deep Reinforcement Learning from Demonstration approach which has also been presented in this Thesis. Furthermore, I contributed to a community-driven effort aimed at providing a generalist policy for robotic manipulation by collecting a dataset for language-guided long-horizon manipulation tasks.

List of Publications

Included Publications

This Thesis is based on the work presented in the following papers:

1. **F. Ceola**, E. Maiettini, G. Pasquale, L. Rosasco, and L. Natale. *Fast Object Segmentation Learning with Kernel-based Methods for Robotics*. In 2021 IEEE International Conference on Robotics and Automation (ICRA), pages 13581–13588, 2021.

2. **F. Ceola**, E. Maiettini, G. Pasquale, G. Meanti, L. Rosasco, and L. Natale. *Learn Fast, Segment Well: Fast Object Segmentation Learning on the iCub Robot*. IEEE Transactions on Robotics, 38(5):3154–3172, 2022.

3. **F. Ceola**, E. Maiettini, L. Rosasco, and L. Natale. *A Grasp Pose is All You Need: Learning Multi-fingered Grasping with Deep Reinforcement Learning from Vision and Touch*. In 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2023.

4. **F. Ceola**, L. Rosasco, and L. Natale. *RESPRECT: Speeding-up Multi-fingered Grasping with Residual Reinforcement Learning*. In IEEE Robotics and Automation Letters (RA-L), 2024.

5. **F. Ceola**, L. Natale, and N. Sünderhauf, K. Rana. *LHManip: A Dataset for Long-Horizon Language-Grounded Manipulation Tasks in Cluttered Tabletop Environments*. Submitted to International Journal of Robotics Research (IJRR), 2023.

Excluded Publication

During the Ph.D., I also contributed to the following paper, although it is not included in the Thesis:

- Open X-Embodiment Collaboration, A. Padalkar, A. Pooley, A. Jain, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Singh, A. Brohan, A. Raffin, A. Wahid, B. Burgess-Limerick, B. Kim, B. Schölkopf, B. Ichter, C. Lu, C. Xu, C. Finn, C. Xu, C. Chi, C. Huang, C. Chan, C. Pan, C. Fu, C. Devin, D. Driess, D. Pathak, D. Shah, D. Büchler, D. Kalashnikov, D. Sadigh, E. Johns, **F. Ceola**, F. Xia, F. Stulp, G. Zhou, G. S. Sukhatme, G. Salhotra, G. Yan, G. Schiavi, H. Su, H. S. Fang, H. Shi, H.B. Amor, H. I. Christensen, H. Furuta, H. Walke, H. Fang, I. Mordatch, I. Radosavovic, I. Leal, J. Liang, J. Kim, J. Schneider, J. Hsu, J. Bohg, J. Bingham, J. Wu, J. Wu, J. Luo, J. Gu, J. Tan, J. Oh, J. Malik, J. Tompson, J. Yang, J. J. Lim, J. Silvério, J. Han, K. Rao, K. Pertsch, K. Hausman, K. Go, K. Gopalakrishnan, K. Goldberg, K. Byrne, K. Oslund, K. Kawaharazuka, K. Zhang, K. Majd, K. Rana, K. Srinivasan, L. Y. Chen, L. Pinto, L. Tan, L. Ott, L. Lee, M. Tomizuka, M. Du, M. Ahn, M. Zhang, M. Ding, M. K. Srirama, M. Sharma, M. J. Kim, N. Kanazawa, N. Hansen, N. Heess, N. J. Joshi, N. Suenderhauf, N. D. Palo, N. M. M. Shafiullah, O. Mees, O. Kroemer, P. R. Sanketi, P. Wohlhart, P. Xu, P. Sermanet, P. Sundaresan, Q. Vuong, R. Rafailov, R. Tian, R. Doshi, R. Martín-Martín, R. Mendonca, R. Shah, R. Hoque, R. Julian, S. Bustamante, S. Kirmani, S. Levine, S. Moore, S. Bahl, S. Dass, S. Song, S. Xu, S. Haldar, S. Adebola, S. Guist, S. Nasiriany, S. Schaal, S. Welker, S. Tian, S. Dasari, S. Belkhale, T. Osa, T. Harada, T. Matsushima, T. Xiao, T. Yu, T. Ding, T. Davchev, T. Z. Zhao, T. Armstrong, T. Darrell, V. Jain, V. Vanhoucke, W. Zhan, W. Zhou, W. Burgard, X. Chen, X. Wang, X. Zhu, X. Li, Y. Lu, Y. Chebotar, Y. Zhou, Y. Zhu, Y. Xu, Y. Wang, Y. Bisk, Y. Cho, Y. Lee, Y. Cui, Y. H. Wu, Y. Tang, Y. Zhu, Y. Li, Y. Iwasawa, Y. Matsuo, Z. Xu, and Z. J. Cui. *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*. In 2024 IEEE International Conference on Robotics and Automation (ICRA), 2024.

Table of contents

List of figures

List of tables

List of acronyms

BC	*Behavior Cloning*
CNN	*Convolutional Neural Network*
DMP	*Dynamic Movement Primitive*
DoF	*Degree of Freedom*
DRL	*Deep Reinforcement Learning*
FCN	*Fully Convolutional Network*
FPN	*Feature Pyramid Network*
G-PAYN	*A Grasp Pose is All You Need*
HRI	*Human-Robot Interaction*
IK	*Inverse Kinematics*
IoU	*Intersection over Union*
KL	*Kullback-Leibler*
KRR	*Kernel Ridge Regression*
LHManip	*Long-Horizon Manipulation dataset*
LLM	*Large Language Models*
MAE	*Masked Autoencoder*
mAP	*mean Average Precision*
MDP	*Markov Decision Process*
MetaRL	*Meta Reinforcement Learning*
MoCap	*Motion Capture*
MSO	*MuJoCo Scanned Objects*
O-OD	*On-line Object Detection*
O-OS	*On-line Object Segmentation*
RESPRECT	*RESidual learning with PREtrained CriTics*
RL	*Reinforcement Learning*
RLfD	*Reinforcement Learning from Demonstration*

RLS	*Regularized Least Squares*
RoI	*Region of Interest*
RPL	*Residual Policy Learning*
RPN	*Region Proposal Network*
RRL	*Residual Reinforcement Learning*
SAC	*Soft Actor-Critic*
TAMP	*Task and Motion Planning*

Part I

Introduction

Chapter 1

Motivation

> *"Encoded in the large, highly evolved sensory and motor portions of the human brain is a billion years of experience about the nature of the world and how to survive in it. The deliberate process we call reasoning is, I believe, the thinnest veneer of human thought, effective only because it is supported by this much older and much more powerful, though usually unconscious, sensorimotor knowledge. We are all prodigious olympians in perceptual and motor areas, so good that we make the difficult look easy. Abstract thought, though, is a new trick, perhaps less than 100 thousand years old. We have not yet mastered it. It is not all that intrinsically difficult; it just seems so when we do it."*
>
> Hans Moravec, *Mind Children: The Future of Robot and Human Intelligence*, 1998

Extraction of meaningful information and manipulation of objects in unstructured and dynamically changing environments are key components to enable robots to autonomously operate in cooperation with humans, interact with unknown objects, or perform tasks in new environmental conditions.

Research in classical control in constrained environments allowed the deployment of robotic industrial and manufacturing systems [13]. In these settings, robots are programmed to execute sets of pre-defined tasks with a high level of precision. The deployment of these approaches in environments like those in which we live, however, is hampered by the difficulty of extracting precise visual and physical information of unseen objects and environment conditions that change over time [205].

The latest research on *Deep Learning* methods for *Computer Vision* and *robot control* problems exhibits remarkable performance. They solve visual perception tasks, such as image classification [216, 83, 140], object detection [165, 18] or instance segmentation [66] when trained on large datasets composed of thousands or millions of images. Similarly, *Imitation Learning* or *Reinforcement Learning* approaches have been deployed for example to control robotic arms used to solve manipulation tasks [77, 231], in whole-body control

problems such as legged locomotion [33, 119], or to navigate in the environment [215]. However, the deployment of these approaches on real robotic platforms is hindered by the requirement of huge amounts of training data and interactions with the environment, and by the necessity of ground-truth information [138] or dense rewards [195], which are difficult to obtain.

Motivated by the success of *Deep Learning* methods in off-line settings and robotic simulators, and by evidence that their on-line adaptation can improve performance [64, 217], with this Thesis I aim at addressing the problem of deploying such methods on real robotic platforms. Extracting information from models pre-trained on huge amounts of data offers the opportunity to speed-up the training on novel tasks [143], either leveraging on features extracted with a pre-trained model, or using pre-trained policies as base sub-optimal controllers. A widely used approach for adaptation of pre-trained models is fine-tuning them on the novel tasks. However, this approach comes with some limitations that hinder its application in the tasks considered in this Thesis.

I will focus on the tasks of instance segmentation and multi-fingered grasping, investigating the limitations of current state-of-the-art methods for deployment of these tasks on the iCub [123] humanoid. I will leverage multi-modal sensory information to reduce the burden deriving from data labeling or to gather precise descriptions of the environment, without considering information that would be available only in simulated environments. Starting from this information, I will propose novel methods that can be trained on the real robot, being much faster than existing methods for the tasks at hand.

Finally, I will consider an additional limitation of current learning-based approaches for robotic manipulation tasks. Typically, these methods solve short-horizon tasks that require few steps to be completed [92, 14]. These tasks, however, are not representative of the those that are typically expected to be performed by a robot in everyday environments. While approaches based on *Hierarchical Reinforcement Learning* [130, 218] have been extensively studied to solve long-horizon tasks, their deployment on real robots is limited by the lack of real-world datasets. To address this limitation and motivated by the long-term reasoning capabilities of *Large Language Models* [1], I will present a new dataset for long-horizon language-guided manipulation tasks.

Chapter 2

Research Objectives

In this Thesis, I pursued several research objectives to provide methods for the tasks of instance segmentation and multi-fingered grasping trainable as fast as possible on the iCub [123] humanoid.

I started my Ph.D. project working on the task of instance segmentation. The first objective was to design a new method trainable on novel classes, keeping the performance of state-of-the-art *Deep Learning* approaches, but reducing the training time as much as possible. I then considered the problem of rapidly adapting the segmentation model also to novel visual scenarios. After achieving these objectives by designing a training pipeline which was suitable for training on off-line datasets, the deployment of this pipeline on the robot became the target of the project. To this aim, the first objective was to design a training protocol that could perform some operations while acquiring data from the robot and then to adapt it to an incremental setting, since the robot may be required to learn to segment novel objects at different times.

Then, I moved to the task of multi-fingered grasping with *Deep Reinforcement Learning*. I started the project with the aim of learning grasping policies in a simulated environment, but using only information available also on the real robot. Relying on policies trained with this approach, I finally focused on the goal of speeding-up the training to learn to grasp new objects on the real robot.

As an additional contribution to the Thesis, I acquired a dataset for real world long-horizon robotic manipulation tasks from a single language instruction. The development of learning-based methods for such tasks may be the key component for the deployment of robot in everyday environments.

Chapter 3

State-of-the-art

3.1 Robotic Visual Object Perception Learning

Perception of the environment is a long-standing problem in robotics. While being strictly related to the *Computer Vision* literature, methods for robotic visual perception have different requirements, depending on the target applications.

Computer Vision methods that require long training sessions and huge amounts of labeled training data for the tasks of object detection and instance segmentation [165, 66, 18, 201] are suited for applications that either involve a pre-defined set of objects or do not require specific knowledge of the unknown objects. For example, [39] presents an approach to grasp unknown objects, after segmenting them with a modified Mask R-CNN [66] trained on synthetic depth data. The approach in [198], instead, jointly learns instance and semantic segmentation for pick-and-place of 40 objects.

The latest research on *Open World* learning of visual perception tasks addresses the problem of detecting [74, 211, 232] or segmenting [203, 208] objects belonging to classes unseen during training. These methods first detect unknown objects from the background, and then incrementally learn the discovered ones. While showing promising performance, they do not always focus on accelerating the training process or reducing the number of training images containing objects belonging to the unknown classes, which may be a critical requirement in robotics.

Few-shot incremental learning methods learn to detect [150, 47] and segment [57, 78] new objects, reducing the number of training images as much as possible. Some of these approaches have been deployed to solve robotic tasks. For example, [44] relies on a few-shot incremental object detector to learn to grasp new objects. These methods are closely related to the ones proposed in the instance segmentation project described in this Thesis, in that they

focus on reducing the number of training images. However, few-shot incremental learning methods usually do not specifically aim at accelerating the training procedure, which may be critical in robotic applications.

Open Vocabulary [209] vision-language models trained on large datasets have shown remarkable generalization properties to unknown object instances. These approaches for object detection or instance segmentation first predict visual class embeddings for each bounding box or mask in an image. Subsequently, the visual embeddings are compared to a set of language embeddings representing different classes computed with a language model, such as CLIP [156]. Representative *Open Vocabulary* methods for the task of detection are OWLv2 [127] and BARON [210]. OpenSD [96] and Semantic-SAM [94], instead, simultaneously detect and segment objects. While overcoming the limitations of models trained on a closed set of classes, these approaches are not suited for robotic applications requiring to learn specific objects during robot operations.

The two approaches for fast instance segmentation and region proposal learning presented in this Thesis (see Chapter 7) target robotic applications aimed at learning unknown objects as fast as possible. Therefore, they differ from the state-of-the-art, in that they focus on reducing the time required by a robot to learn new objects in possibly different visual domains.

3.2 Learning for Multi-fingered Grasping

Multi-fingered grasping is an intrinsically challenging task that requires controlling tens of degrees of freedom and taking into account object-finger interactions that may occur during grasp execution. It represents a key task in robotics, not only for pick-and-place, but also to enable robots to perform more sophisticated tasks, such as object re-orientation [4, 29], food preparation, or clothes folding [133].

The problem of grasping with anthropomorphic hands has been historically addressed with analytical methods that consider the geometrical and physical properties of both the hand and the object [7]. Due to the difficulty of estimating such properties in practice, and thanks to the development of reliable depth sensors such as the Microsoft Kinect, data-driven approaches have found widespread application for the estimation of grasp poses, and grasp execution [8]. In the last decade, *Deep Learning* methods have become the dominant trend for grasp synthesis [133]. However, these mainly focus on generating grasp poses for two-fingered grippers [11, 28]. Methods proposed in [230, 95], instead, start from pointcloud information to generate multi-fingered grasp candidates by computing hand-object contact points. In practice, these methods are difficult to apply because they do not take into account

the hand-object interactions occurring during grasp execution and are constrained to the hardware used for training.

The work in [101] and DexPoint [154] propose to overcome these limitations by learning grasping policies with *Deep Reinforcement Learning*. [101] computes the grasp pose for a Shadow hand with an external algorithm and relies on synergies to reduce the number of degrees of freedom to control the fingers during grasp execution. The policy is trained using tactile information, joint angles and torques as input. While moving toward the development of policies for grasp execution, rather than focusing on grasp synthesis, this approach is limited by the use of torque information as input, that is not always available in other robotic hands. Moreover, the approach in [101] does not consider information about the object during policy execution, which may prevent it to grasp the object if the initial grasp pose is unfeasible. DexPoint [154], instead, trains policies in simulation to grasp objects and open doors from pointclouds. While demonstrating that the policies can be transferred to the real robot without fine-tuning, this transfer can be strongly affected by the quality of the sensors.

The latest research on robot learning of large models via *Imitation Learning* [14, 231, 138, 137, 226, 54, 10] has shown remarkable performance on several manipulation tasks using two-fingered grippers. The deployment of these methods for multi-fingered tasks is, to the best of my knowledge, still unexplored and can represent an interesting future research direction, but is hindered by the requirement of huge amounts of training data. These data may be difficult to obtain, and their collection can require expensive hardware and be time consuming. Moreover, *Behavior Cloning* approaches are known to suffer from the mismatch between the training data and the estimated policy used in practice. This leads to compounding errors that are challenging to address [167].

To reduce the effort required for collecting data for robot policy learning, approaches such as those presented in [155, 178, 202] leverage human demonstrations. This represents an interesting research direction for scaling-up data collection, but these approaches usually require either a simulated environment to address the human-to-robot visual gap or to fine-tune a policy pre-trained on human data with robot demonstrations collected via teleoperation. Furthermore, they are poorly suited for collection of different data modalities, such as tactile data, which may be crucial for tasks like multi-fingered grasping.

In this Thesis, I present two methods for learning multi-fingered grasping with *Deep Reinforcement Learning* (see Chapter 8). The former, *G-PAYN*, leverages automatically collected demonstrations to learn grasping policies in simulation from visual, tactile and proprioceptive information. The latter, *RESPRECT*, learns to grasp novel objects both in simulation and on the real robot with a *Residual Reinforcement Learning* approach, leveraging

a *G-PAYN* policy pre-trained on a large dataset of different objects. Differently from the state-of-the art, they have been designed with the final goal of learning the task on the real robot as fast as possible with *Deep Reinforcement Learning* to avoid compounding errors and the requirement of real grasping demonstrations typical of *Imitation Learning* methods.

3.3 Robotic Manipulation Datasets

The availability of diverse and heterogeneous datasets has been the key to achieve the latest success in *Natural Language Processing* [1] and *Computer Vision* [156, 82] both for large-scale training on such datasets and for fine-tuning pre-trained models on small-scale datasets. For example, image classification models pre-trained on large and heterogeneous data from ImageNet [43] and fine-tuned on different and smaller datasets, overall perform better than the same models trained from scratch on the target data [84].

Collecting datasets that demonstrate robots performing real manipulation tasks to train the equivalent for robotics of a *Computer Vision* model pre-trained on ImageNet has been a long-standing open problem due to the complexity of collecting real robotic data. The problem has recently been tackled by the *Open X-Embodiment* project [138]. This collaboration between 21 institutions aimed to collect a dataset showing 22 different robots demonstrating 527 skills, corresponding to 160266 tasks. The experimental evaluation in [138] assesses performance on different small-scale datasets. This compares the original methods presented together with the datasets, RT-1 [14] models trained on each dataset, and an RT-1 model, named RT-1-X, which was trained on a mixture of datasets that includes the small-scale datasets considered to evaluate performance. Results show that, overall, RT-1-X outperforms the considered baselines on tasks with limited data, benefiting from co-training on the mixture of datasets.

The *Open X-Embodiment* dataset comprises a large number of state-of-the-art datasets for robotic manipulation with two-fingered grippers. Among others, it includes datasets such as Jaco Play [41] and BridgeData V2 [199] showing robots performing tasks in kitchen environments, data used to train VINN [142] to open cabinet doors, and a dataset for cable routing [111]. I contributed to the project with the *LHManip* dataset (see Chapter 9). I collected the dataset via teleoperation, showing the robot performing long-horizon tabletop tasks. *LHManip* differs from the other datasets in *Open X-Embodiment* in that they mostly show robots performing tasks that require few steps to be completed. It also advances the state-of-the-art of long-horizon datasets, which are usually collected either in simulation [224] or simplified environments [68].

One common limitation of the datatsets in the *Open X-Embodiment* collaboration is that they generally consider two-fingered grippers, and tasks are performed with a single arm and limited sensor multi-modality (visual and proprioceptive data). While recent work provides data for bi-manual tasks [226, 54] or datasets considering visual, tactile and language data [53], large-scale heterogeneous datasets with similar features are still not available.

Chapter 4

Thesis Outline

The Thesis is organized as follows:

- Part II provides an overview of the background necessary to understand the key concepts and methodologies relevant to the research work presented in this Thesis. Specifically, in Chapter 5, I describe the background for the fast instance segmentation learning project. In Chapter 6, I provide an overview of the *Reinforcement Learning* algorithms and methodologies underlying the proposed approaches for multi-fingered grasping.

- In Part III, I summarize the scientific contributions provided with the work done in this Thesis. Specifically, in Chapter 7, I report the contributions for the fast instance segmentation learning project. In Chapter 8, I describe the work carried out for the multi-fingered grasping with *Deep Reinforcement Learning* project. Finally, in Chapter 9, I overview the key features of the dataset that I collected for language-guided long-horizon manipulation tasks.

- Chapters in Part IV correspond to the five publications included in this Thesis. Each Chapter is self-contained and includes all the components of the corresponding paper.

- In Part V, I conclude the thesis, summarizing the results achieved with my Ph.D. project and discussing directions for future work.

Part II

Background

Chapter 5

Instance Segmentation

Figure 5.1 Exemplar instance segmentation input and output[1].

Instance Segmentation is a *Computer Vision* task that consists in classifying every pixel of an image, taking into account different instances of the same class, as for example the people in Fig. 5.1. The problem can be formalized as follows:

- Given an input image I represented as a matrix of pixels, where $I(x,y)$ represents the pixel value at coordinates (x,y).
- Supposing that there are N objects in the image, and that each object is associated with a unique label l_i with $i \in \{1,2,\ldots,N\}$.
- The objective is to assign every pixel (x,y) in the image I to one of the object instances or to the background. Formally, this corresponds to find a map S such that $S : \{(x,y)\} \rightarrow \{l_1, l_2, \ldots, l_N, \text{background}\}$.

In the literature, this problem has been addressed with three different kinds of approaches:

[1] Source: https://github.com/facebookresearch/Detectron. Accessed on: 23/12/2023.

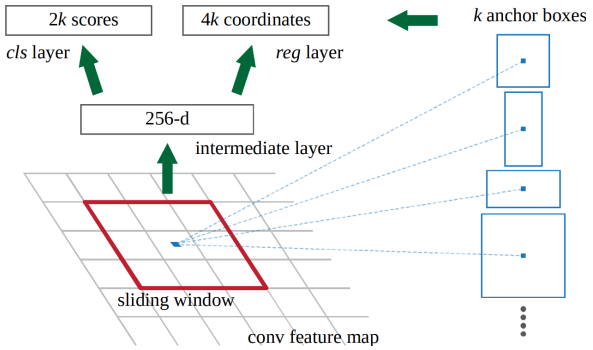


Figure 5.2 **Region Proposal Network (RPN).** For each location in the feature map, the RPN computes $2k$ classification values and $4k$ regression values, where k is the number of anchors representing bounding boxes with fixed aspect ratio. The classification values represent whether, for a given anchor in the given location, the feature map represents an object. The regression values are used to refine the bounding boxes, starting from the fixed bounding boxes represented by each anchor[2].

- **Detection-based Methods**: these approaches build on top of approaches for object detection, by adding a branch for mask prediction within the bounding boxes proposed by the detector. Three exemplar methods are Mask R-CNN [66] that extend the two-stage object detector Faster R-CNN [165], YOLACT [9] that builds on top of a single-stage RetinaNet-like [103] detector, and the extended version of the transformer-based DETR [18].

- **Semantic Segmentation-based Methods**: these approaches cluster into instances pixels of the same class as predicted by a semantic segmentation method. SSAP [58] and InstanceCut [81] are two approaches in this class.

- **Dense Sliding Window Methods**: approaches as DeepMask [153] and Instance-FCN [36] simultaneously predict mask instances and their class-agnostic or class-specific scores.

In Sec. 5.1, I will describe in details Mask R-CNN. At the beginning of the project aimed at accelerating instance segmentation learning, it represented the state-of-the-art for the task, and I built my methods on top of it.

5.1 Mask R-CNN

[2]Source: [165].

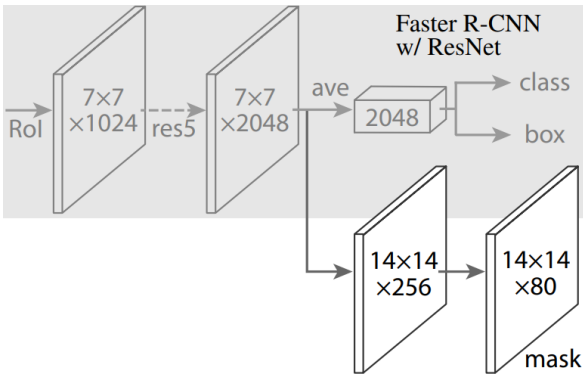


Figure 5.3 ***Detection* and *segmentation* heads in Mask R-CNN with ResNet backbone.** For each *RoI*, Mask R-CNN computes a mask of size 14×14 for each of the considered classes. In the figure, masks are computed for each of the 80 classes in the MS COCO [105] dataset[3].

Mask R-CNN extends Faster R-CNN by adding, in parallel with the branches for classification and bounding box regression, a branch for predicting segmentation masks on each *Region of Interest* (*RoI*). Given an input image, Mask R-CNN detects objects and computes masks as follows:

- The image is processed by a backbone ResNet [67] or ResNeX [216] convolutional neural network (CNN) for feature extraction.
- The feature map computed by the backbone is then processed by the *Region Proposal Network* (RPN) to generate a set of *RoIs*, representing bounding boxes which are candidate to contain on object. Fig. 5.2 shows how the RPN works.
- For each *RoI*, the feature map is cropped and resized with the *RoI Align* to a smaller feature map with a fixed spatial extent. Each of these per-*RoI* feature maps is further processed by other convolutional layers.
- Two fully connected layers in the *detection* head classify the per-*RoI* feature maps and refine their bounding box.
- Finally, in parallel to the *detection* head, the *segmentation* head processes the feature maps with a small CNN and predicts the segmentation mask for each object. Fig. 5.3 shows the architecture of Mask R-CNN's *detection* and *segmentation* heads.

[3] Source: figure adapted from [66].

While Mask R-CNN represents the state-of-the-art for the task of instance segmentation, its adoption in robotics is hampered by the fact that the end-to-end training requires long time, particularly when it is trained from scratch. This limitation hinders its use in robotic applications requiring fast adaptation to novel objects and visual scenarios. To overcome this limitation, in my works, I proposed to rely on Mask R-CNN for feature extraction while replacing the final layers of the RPN and of the *segmentation* head with FALKON [169] classifiers, and to rely on the method in [114] for adaptation of the *detection* head. I will describe FALKON and the method for *On-line Object Detection* (O-OD) [114, 115] in the following sections.

5.2 FALKON

FALKON is a kernel method that solves *Kernel Ridge Regression* (KRR) [173] for large-scale problems. It combines Nyström subsampling [206] to approximate the KRR problem and efficiently computes a preconditioning for optimization via conjugate gradient [171].

Kernel methods consider a space $\mathcal{F}$ of functions

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i), \tag{5.1}$$

where n is the number of points in the dataset $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ and K is a positive definite kernel. Coefficients $\alpha_1, \ldots \alpha_n$ are typically derived by solving a convex optimization problem. If this solved using the square loss, the KRR estimator is defined as

$$\hat{f}_{n,\lambda} = \arg\min_{f \in \mathcal{F}} \left(\frac{1}{n} \sum_{i=1}^{n} (f(x_i) - y_i)^2 + \lambda \|f\|_{\mathcal{F}}^2 \right), \tag{5.2}$$

where λ is the regularization parameter. With this formulation, the solution of the problem is reduced to the solution of the linear system

$$(\mathbf{K}_{nn} + \lambda \mathbf{I})\alpha = \hat{\mathbf{y}}, \tag{5.3}$$

where $\mathbf{K}_{nn}$ is an $n \times n$ matrix defined by $(\mathbf{K}_{nn})_{ij} = K(x_i, x_j)$ and $\hat{\mathbf{y}} = (y_1, \ldots, y_n)$. Solving Eq. 5.3 requires $O(n^2 c_K + n^3)$ in time (with c_K, assumed constant, the time for kernel evaluation), which may is not suitable for problems that require fast learning on large-scale datasets.

To overcome this problem, FALKON approximates the problem in Eq. 5.1 as

$$\tilde{f}_{\lambda,M}(x) = \sum_{i=1}^{M} \tilde{\alpha}_i K(x, \hat{x}_i), \text{ with } \{\tilde{x}_1, \ldots, \tilde{x}_M\} \subseteq \{x_1, \ldots, x_n\}. \tag{5.4}$$

This approximation considers a subset of M training points (Nyström centers) uniformly sampled from the dataset. By solving the problem using the square loss as in Eq. 5.2, and choosing $M = \tilde{O}(\sqrt{n})$ to preserve the optimal statistical guarantees of the exact KRR [168], the computational time for kernel evaluations is reduced to $O(n\sqrt{n})$ and the time complexity to solve the problem to $O(n^2)$, which still may not be suitable for fast learning on large-scale datasets.

By preconditioning the linear system in Eq. 5.3 and solving the preconditioned problem via conjugate gradient [171], a fast iterative gradient method that does not require to set the step-size, FALKON reduces the time complexity to $O(nMt + M^3)$, where t is the number of training iterations. Approximately $\log n$ iterations are sufficient for preserving optimal statistical properties, decreasing the computational time requirements for optimal accuracy to $\tilde{O}(n\sqrt{n})$. This results in a training time reduction of a factor $\tilde{O}(n\sqrt{n})$ with respect to standard kernel-based classifiers, and of a factor $\tilde{O}(\sqrt{n})$ with respect to other Nyström approaches.

5.3 On-line Object Detection

The approach for O-OD proposed in [114, 115] leverages a pre-trained Faster R-CNN [165] for feature extraction and FALKON [169] for on-line learning of object detection on new classes. O-OD extracts per-*RoI* features from the penultimate layer of the Faster R-CNN *detection* head (see Fig. 5.3) and trains N, with N the number of the novel classes, FALKON binary classifiers to replace the *class* layer in Fig. 5.3, and $4N$ Regularized Least Squares (RLS) for bounding box refinement (*box* layer in the figure).

The key component to train the FALKON classifiers is the proposed *Minibootstrap* algorithm. This is an iterative training procedure that addresses the problem of hard negatives mining [187, 59] in object detection. *Minibootstrap* selects a subset of hard negative samples to balance the training sets associated to each of the N classes. At the beginning of the training, *Minibootstrap* initializes the set P of positive examples and n_B batches of negative samples N_i with $i = 1, \ldots, n_B$ for each class. Then, at each training iteration i, *Minibootstrap* performs the following steps for each of the N classifiers:

- It selects hard negatives (i.e. negatives whose classification score is above a threshold t_H according to a given classifier) N_i^H from N_i using the classifier M_{i-1} trained at iteration $i-1$ and adds them to the train set D_i. This is now composed of P, N_i^H and the hard negatives chosen at iteration $i-1$ $N_{chosen,i-1}$ (see the last point).

- It trains the new classifier M_i using the dataset D_i. M_i is a FALKON classifier, but the Nyström centers are not uniformly sampled from the whole training dataset as in the original FALKON sampling procedure. In the the *Minibootstrap*, M Nyström centers are chosen such that they are composed of a set of $P' = \min(|P|, \frac{M}{2})$ positive examples uniformly sampled from P, and of a set of $M - P'$ negatives uniformly sampled from the negative examples in D_i.

- It computes $N_{chosen,i}$ by removing easy negatives (i.e. negatives whose classification score is below a t_E threshold according to a given classifier) from D_i using M_i.

The full *Minibootstrap* procedure is reported in [115] and in Sec. 11.14.

O-OD allows to train detection models in few seconds. This makes it suitable for robotic applications that require fast adaptation in dynamically changing environments.

Chapter 6

Reinforcement Learning

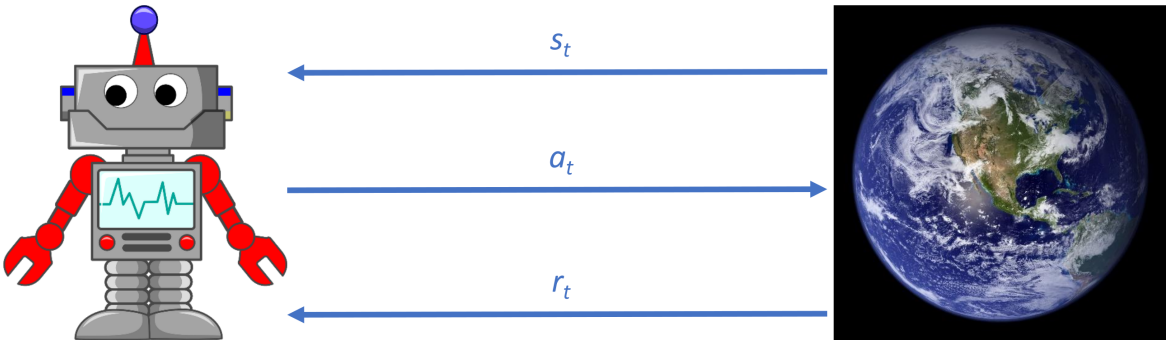


Figure 6.1 Reinforcement Learning *state-action-reward* loop.

Reinforcement Learning (RL) is a learning paradigm where an agent learns by interacting with the environment. Differently from the supervised learning framework, where the agent is trained to mimic actions to be executed in a given state from a labeled dataset, the agent is trained to maximize the cumulative long-term reward obtained throughout an episode of the task at hand.

An RL problem involves an *agent*, the robot in Fig.6.1, interacting with the *environment*, the world in the figure. In this framework, the *agent* performs a sequence of actions to solve a task. At a given timestep, the *agent* executes the action a_t predicted by its current policy, given the observed state s_t as input. After the execution of the action, the *environment* transitions to a new state s_{t+1}, and the *agent* receives a reward r_t, which evaluates the action.

An RL problem can be formalized as a Markov Decision Process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, P_{sa}, R)$, where:

- $\mathcal{S}$ is a set of states.

- $\mathcal{A}$ is a set of actions.
- $P_{sa} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty)$ is the transition probability that defines the probability $P(s_{t+1}|s_t, a_t)$ of transitioning from state s_t to state s_{t+1} after taking action a_t.
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, where $R(s_t, a_t) = r_t$ is the immediate reward received after transitioning from state s_t with action a_t.

The transition probability P_{sa} can be either known (*model-based* RL) or unknown (*model-free*). In this Thesis, I will always consider P_{sa} as an unknown probability distribution that can be accessed only by sampling.

A key feature of an MDP is the Markov property. This asserts that the probability of transitioning to the state s_{t+1} depends only on the current state s_t and the taken action a_t, and not on the sequence of the previous states and actions. Formally, the Markov property is given by the following conditional probability, defined for any sequence of timesteps $t_1 < t_2 < \ldots < t_n$:

$$P(s_{t_n}|s_{t_{n-1}}, a_{t_{n-1}}) = P(s_{t_n}|s_{t_{n-1}}, a_{t_{n-1}}, \ldots, s_{t_1}, a_{t_1}). \tag{6.1}$$

This property allows the agent to make decisions based only on the current state, without the need to remember the past state-action pairs, significantly simplifying the learning process.

The objective in an MDP is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected cumulative reward obtained transitioning the environment from timestep 0 to timestep T. This is expressed as the maximization of the discounted expected return given by

$$R_t = \sum_{t=0}^{T-1} \gamma^t r_t, \tag{6.2}$$

where $\gamma \in [0, 1]$ is the discount factor, which weights future and immediate rewards in the discounted return. In infinite-horizon tasks (i.e. $T = \infty$) $\gamma < 1$ is necessary to make the sum in the equation finite.

The goal of RL is to find a policy π^* that maximize the discounted expected return in Eq. 6.2. This can be formally expressed as:

$$\pi^* = \arg\max_{\pi} \sum_t \mathbb{E}_{(s_t, a_t) \sim \rho_\pi}[\gamma^t r_t], \tag{6.3}$$

where ρ_π denotes the state-action marginal of the trajectory distribution induced by a policy π.

In this Thesis, I will consider only *Deep Reinforcement Learning* (DRL) algorithms. These algorithms define the policy π as a neural network parameterized by a set of weights θ. Hereinafter, I will refer to a generic DRL policy as π_θ. Consequently, if we define a trajectory $\tau = ((s_0, a_0, r_0), \ldots, (s_t, a_t, r_t), \ldots, (s_{T-1}, a_{T-1}, r_{T-1}))$ as the sequence of state-action-reward tuples obtained during execution of a policy π_θ, the probability of starting the trajectory from state s_0 as $p(s_0)$, the probability distribution over trajectories induced by a policy π_θ as

$$P(\tau|\pi_\theta) = p(s_0)\prod_t P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t), \tag{6.4}$$

and we express the cumulative reward defined in Eq.6.2 as

$$R(\tau) = \sum_t \gamma^t r_t, \tag{6.5}$$

we can define the expected discounted return over trajectories induced by a policy π_θ as:

$$J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]. \tag{6.6}$$

Therefore, the goal of a DRL policy can be expressed as follows:

$$\pi_\theta^* = \arg\max_{\pi_\theta} J(\pi_\theta). \tag{6.7}$$

6.1 Model-Free Deep Reinforcement Learning Algorithms

Model-free DRL algorithms can be divided into three categories: *Policy-Gradient*, *Value-Based* and *Actor-Critic* methods. These methods differ on the optimization target to learn the DRL policy.

6.1.1 Policy-Gradient Methods

Policy-gradient methods are a class of RL algorithms that directly optimize the policy. These methods adapt the parameters of the policy θ to maximize the expected cumulative reward. Eq. 6.7 can be equivalently formulated as follows:

$$\theta^* = \arg\max_{\theta} J(\pi_\theta). \tag{6.8}$$

The optimization of the parameters θ to improve the policy is usually performed via gradient ascent:

$$\theta_{\text{new}} = \theta_{\text{old}} + \alpha \nabla_\theta J(\theta_{\text{old}}), \tag{6.9}$$

where α denotes the learning rate. The gradient of the objective function $\nabla_\theta J(\theta)$ can be computed as described in the REINFORCE algorithm [207, 188]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \gamma^t R_t(\tau) \right], \tag{6.10}$$

where $R_t(\tau)$ is the return received after timestep t in a trajectory τ and is defined as:

$$R_t(\tau) = \sum_{k=0}^{T-1-t} \gamma^k r_{t+k}. \tag{6.11}$$

Policy-gradient methods directly optimize the policy. In principle this can lead to stable learning, but this class of algorithms suffers from high variance for policy-gradient estimation, thus requiring several policy evaluations for an effective estimate of such value. This process, often involving multiple interactions with the environment, can result in significant computational overhead and sample-inefficiency. Algorithms like TRPO [174] and PPO [175] limit this issue by constraining policy updates.

6.1.2 Value-Based Methods

Value-based RL algorithms focus on estimating the value function of a policy - a measure of how good it is to be in a given state or to perform a certain action in a state. These algorithms do not explicitly maintain a policy, which is implicitly derived from the value function.

If the value function is computed to evaluate the expected return by executing the action a when the environment is in state s, we define the action-value function for a policy π as:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a \right]. \tag{6.12}$$

Instead, the state-value function representing the expected return when the environment is in state s is defined as:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s \right]. \tag{6.13}$$

An important property of the value functions is that they can be expressed in a recursive way. Eq. 6.14 and Eq. 6.15 are known as *Bellman* equations [6] of Q^π and V^π.

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a\right] \tag{6.14}$$

$$V^\pi(s) = \mathbb{E}_\pi\left[r_t + \gamma V^\pi(s_{t+1}) \mid s_t = s\right] \tag{6.15}$$

From the definitions of Q^π and V^π derives also the definition of the advantage function A^π. This is used to estimate the advantage of taking an action a over the expected return from state s. A^π is defined as:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). \tag{6.16}$$

For value-based methods, the agent selects the action that has the highest estimated value in the current state. Therefore, the resulting optimal policy can be expressed as:

$$\pi^*(s) = \arg\max_a Q^{\pi^*}(s,a). \tag{6.17}$$

DRL value functions are defined as neural networks parameterized by a set of parameters ϕ. These algorithms are usually *off-policy* (i.e. they can learn from state-action-reward tuples that are not derived from the current policy) and optimize ϕ on batches of transitions sampled from the *replay buffer*. The latter stores transitions (s_t, a_t, r_t, s_{t+1}) accumulated during training.

An exemplar value-based DRL algorithm is Deep Q-learning [128, 129]. This algorithm samples a set of transitions (s_j, a_j, r_j, s_{j+1}) from the *replay buffer* and defines the target for Q^π_ϕ, y_j, as:

$$y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_a Q^\pi_\phi(s_{j+1}, a) & \text{otherwise} \end{cases} \tag{6.18}$$

Then, it minimizes the loss $(y_j - Q^\pi_\phi(s_j, a_j))^2$, also known as Bellman error minimization, to find the optimal weights ϕ for Q^{π^*}. Algorithms like Double DQN [194] and Dueling DQN [204] improve Deep Q-learning limitations. Despite their simplicity, it is impractical to scale these algorithms to environments with large or continuous action and state spaces.

6.1.3 Actor-Critic Methods

Actor-critic algorithms combine the advantages of policy-gradient and value-based methods. These algorithms use two models: the *actor* representing the policy and the *critic* to estimate

the value function. The training for these methods is an alternating procedure where the *critic* is updated to improve the prediction of the expected returns, while the *actor* is updated relying on the *critic*'s value estimates.

A representative DRL actor-critic method is DDPG [102]. Under the assumption that the policy π_θ is deterministic, i.e. $a_t = \pi_\theta(s_t)$, and that the expectation in Eq. 6.14 depends only on the environment E, which may be stochastic, DDPG estimates the action-value function as:

$$Q_\phi^{\pi_\theta}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r_t + \gamma Q_\phi^{\pi_\theta}(s_{t+1}, \pi_\theta(s_{t+1}))\right]. \tag{6.19}$$

Since the expectation depends only on the environment, it is possible to learn Q^{π_θ} *off-policy*, i.e. using transitions which are generated from a different stochastic behavior policy π. Optimization of $Q_\phi^{\pi_\theta}$ is performed as described in Sec. 6.1.2 for the Deep Q-Learning algorithm. $Q_\phi^{\pi_\theta}$ is then used to update the *actor* weights θ of the policy π_θ via gradient ascent, as defined in Eq. 6.9. The gradient of the expected cumulative reward with respect to the parameters ϕ of the policy is estimated as:

$$\begin{aligned} \nabla_\theta J &\approx \mathbb{E}_{s_t \sim \rho^\pi}\left[\nabla_\theta Q_\phi^{\pi_\theta}(s,a)|_{s=s_t, a=\pi_\theta(s_t)}\right] \\ &= \mathbb{E}_{s_t \sim \rho^\pi}\left[\nabla_a Q_\phi^{\pi_\theta}(s,a)|_{s=s_t, a=\pi_\theta(s_t)} \nabla_\theta \pi_\theta(s)|_{s=s_t}\right], \end{aligned} \tag{6.20}$$

where $s_t \sim \rho^\pi$ represents a state s_t sampled from the state visitation distribution ρ^π for a generic policy π. Optimization of parameters ϕ and θ is alternated and repeated until convergence.

Actor-critic DRL algorithms are more sample-efficient than policy-gradient methods, often providing faster convergence. Furthermore, they can handle continuous action spaces effectively, making them more suitable for robotic applications than value-based methods. TD3 [55] and Soft Actor-Critic (SAC) [61, 62] (which I will describe in details in Sec. 6.2) are two additional actor-critics algorithms largely used in DRL applications.

6.2 Soft Actor-Critic

The interplay between the deterministic *actor* network and the deep Q-function makes DDPG highly sensitive to hyperparameters and difficult to stabilize. It is therefore difficult to to use in practice for high dimensional tasks. SAC overcomes this limitation by combining off-policy *actor-critic* training with a stochastic *actor*, and maximizing the entropy $\mathcal{H}$ of the *actor* during training. To this end, SAC extends the RL problem formulation in Eq. 6.3 to

find the policy π^* that maximizes the target $J(\pi)$ defined as:

$$J(\pi) = \sum_{t=0}^{\infty} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[\sum_{k=t}^{\infty} \gamma^{k-t} \mathbb{E}_{s_k\sim P_{sa}, a_k\sim\pi} \left[r_k + \alpha \mathcal{H}(\pi(\cdot|s_k)) \right] \right] \tag{6.21}$$

where ρ_π denotes the state-action marginal of the trajectory distribution induced by a policy π as in Eq. 6.3 and α is a learned temperature parameter. The description of the optimization procedure for this parameter is out of the scope of this Thesis. I refer the reader to [62] for its derivation. To compute the action-value of a policy π, SAC computes the soft Q-value iteratively, starting from any function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and repeatedly applying a modified Bellman backup operator T^π given by:

$$T^\pi Q(s_t, a_t) \triangleq r_t + \gamma \mathbb{E}_{s_{t+1}\sim P_{sa}} \left[V(s_{t+1}) \right], \tag{6.22}$$

where

$$V(s_t) = \mathbb{E}_{a_t\sim\pi} \left[Q(s_t, a_t) - \alpha \log \pi(a_t|s_t) \right] \tag{6.23}$$

is the soft state-value function. The soft Q-function for any policy can be obtained by repeatedly applying T^π for any policy π.

In SAC, the soft Q-function Q_ϕ is parameterized by a neural network with weights ϕ. These can be optimized to minimize the soft Bellman residual. This is computed for state-action tuples sampled from the *replay buffer* $\mathcal{D}$ and defined as:

$$J_Q(\phi) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}} \left[\frac{1}{2} \left(Q_\phi(s_t, a_t) - \left(r_t + \gamma \mathbb{E}_{s_{t+1}\sim p} \left[V_{\bar{\phi}}(s_{t+1}) \right] \right) \right)^2 \right], \tag{6.24}$$

where the value function V is implicitly parameterized by the parameters of the soft Q-function, as described in Eq. 6.23. Computation of the value function makes use of a soft Q-function parameterized by a set of parameters $\bar{\phi}$. These weights are obtained as an exponentially moving average of ϕ. $J_Q(\phi)$ can be optimized with stochastic gradient descent. The gradient of $J_Q(\phi)$ can be estimated as:

$$\hat{\nabla}_\phi J_Q(\phi) = \nabla_\phi Q_\phi(a_t, s_t)(Q_\phi(s_t, a_t) - (r_t + \gamma(Q_{\bar{\phi}}(s_{t+1}, a_{t+1}) - \alpha \log(\pi_\theta(a_{t+1}|s_{t+1}))))). \tag{6.25}$$

In practice, SAC makes use of two soft Q-functions parameterized by weights ϕ_i, with $i \in (1,2)$, and trains them independently to optimize $J_Q(\phi_i)$. The minimum of the the soft Q-functions is then used for the gradient estimations in Eq. 6.25 and Eq. 6.29.

To improve the policy, SAC updates π_θ towards the exponential of the new Q-function in terms of *Kullback-Leibler* (KL) divergence. This choice guarantees to obtain an improved policy in terms of its soft value. The objective function $J_\pi(\theta)$ can be obtained by directly minimizing the expected KL-divergence as

$$J_\pi(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\mathbb{E}_{a_t \sim \pi_\theta} \left[\alpha \log(\pi_\theta(a_t|s_t)) - Q_\phi(s_t, a_t) \right] \right]. \tag{6.26}$$

Policy optimization is then performed applying the reparameterization trick. The policy is reparameterized as:

$$a_t = f_\theta(\varepsilon_t; s_t), \tag{6.27}$$

where ε_t is an input noise vector sampled from a fixed distribution $\mathcal{N}$. Eq. 6.26 can be rewritten with π_θ implicitly defined in terms of f_θ as:

$$J_\pi(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}, \varepsilon_t \sim \mathcal{N}} \left[\alpha \log \pi_\theta(f_\theta(\varepsilon_t; s_t)|s_t) - Q_\phi(s_t, f_\theta(\varepsilon_t; s_t)) \right], \tag{6.28}$$

and the gradient can be approximated as:

$$\hat{\nabla}_\theta J_\pi(\theta) = \nabla_\theta \alpha \log(\pi_\theta(a_t|s_t)) + (\nabla_{a_t} \alpha \log(\pi_\theta(a_t|s_t)) - \nabla_{a_t} Q(s_t, a_t)) \nabla_\theta f_\theta(\varepsilon_t; s_t), \tag{6.29}$$

where a_t is evaluated at $f_\theta(\varepsilon_t; s_t)$. SAC outperforms state-of-the-art model-free algorithms as PPO [175], DDPG [102] and TD3 [55] in terms of training stability, number of training timesteps and average return. However, the timesteps required for training are still unfeasible for the target multi-fingered grasping task presented in this Thesis. I will address this problem by proposing two methods based on SAC: the first leverages on grasping demonstrations, the second is a *Residual Reinforcement Learning* (RRL) approach.

6.3 Reinforcement Learning from Demonstration

Reinforcement Learning from Demonstration (RLfD) accelerates the learning process for a DRL policy by providing the agent with demonstrations of the task at hand. Demonstrations are typically used in RLfD either to:

- Pre-train the agent to imitate the demonstrations. This model is then fine-tuned through a traditional DRL method.

- Guide the exploration process of the agent during training. This is usually implemented by modifying the loss functions of traditional DRL algorithms to imitate the behavior of the expert demonstrator.

6.3.1 Pre-training from Demonstration

Pre-training of a DRL policy on demonstrations can be performed either via *Behavior Cloning* (BC) or with *off-line* RL approaches. An exemplar method of the first class of algorithms is DAPG [159], which after the pre-training with BC, fine-tunes the policy with an augmented loss for NPG [76] to stay close to the demonstrations. This loss mitigates the distribution shift between the demonstrations and the data acquired during training. AWAC [131], instead, is a state-of-the-art actor-critic *off-line* RL algorithm. The optimization of the *critic* network parameterized by ϕ is performed by minimizing the Bellman error presented in Sec. 6.1.2. Instead, the *actor* parameters θ are updated, both during off-line and on-line training, as follows:

$$\theta_{k+1} = \arg\max_{\theta_k} \mathbb{E}_{(s,a)\sim\mathcal{D}} \left[\log \pi_{\theta_k}(a|s) \exp\left(\frac{1}{\lambda} A^{\pi_k}(s,a)\right)\right], \tag{6.30}$$

where A^{π_k} is the advantage function for the policy after k training iterations and λ a constant Lagrangian multiplier. In the first part of the training, performed off-line, the *replay buffer* $\mathcal{D}$ is composed of the off-line demonstrations. In the last stage of the training, transitions acquired during on-line training are added to $\mathcal{D}$. This allows to keep the policy during on-line training close to the off-line demonstrations, guiding the policy to perform better action if the action-value estimate is informative, cloning the behavior of the demonstrations otherwise.

6.3.2 Learning to Imitate Demonstrations

RL methods that imitate the demonstrations during training usually load transitions in the *replay buffer* at the beginning of the training, and keep them throughout all the optimization procedure. Two exemplar methods are DDPGfD [195] and the one presented in [132]. The latter optimizes the *actor* parameters by maximizing the expected return and minimizing a BC loss component. This is a modified version of the standard BC loss. It considers only state-action pairs s_i and a_i sampled from the set of demonstrations whose Q-value is greater than the Q-value estimated for the same state, but with the action predicted by the policy. This is done to avoid cloning suboptimal actions. The modified BC loss is defined as:

$$L_{BC} = \sum_{i=1}^{N_D} \|\pi_\theta(s_i) - a_i\|^2 \mathbf{1}_{Q_\phi(s_i,a_i) > Q_\phi(s_i,\pi_\theta(s_i))}. \tag{6.31}$$

The policy is updated with a gradient that combines the gradient of the BC loss with the gradient used to update the *actor* parameters θ, as described in Sec. 6.1.3. This can be expressed as:

$$\lambda_1 \nabla_\theta J - \lambda_2 \nabla_\theta L_{BC}. \tag{6.32}$$

In Eq. 6.32, $\nabla_\theta J$ is the gradient with respect to θ of the DDPG *actor* loss as described in Eq. 6.20, and λ_1 and λ_2 are scalar weighting coefficients. This gradient formulation allows to maximize the expected return, while also minimizing the BC loss.

6.4 Residual Reinforcement Learning

RRL is a class of RL methods that aim at improving base policies that may be non-differentiable by adding a residual policy trained with RL on top of them. The RRL paradigm has been introduced with *Residual Policy Learning* (RPL) [185] to improve imperfect controllers in simulated manipulation tasks. RPL starts from an initial policy $\pi : \mathcal{S} \to \mathcal{A}$ and learns a residual function f_θ to obtain an improved final policy $\pi_\theta(s) = \pi(s) + f_\theta(s)$. Since $\pi(s)$ does not depend on θ, the gradient of $\pi_\theta(s)$ can be defined as

$$\nabla_\theta \pi_\theta(s) = \nabla_\theta f_\theta(s), \tag{6.33}$$

and $\pi_\theta(s)$ can be learned with policy-gradient methods even if π is not differentiable.

The initial policy π together with an MDP $M = (\mathcal{S}, \mathcal{A}, P_{sa}, R)$ induces a residual MDP $M(\pi) = (\mathcal{S}, \mathcal{A}, P^\pi_{sa}, R)$, where:

$$P^\pi_{sa}(s, a, s') = P_{sa}(s, \pi(s) + a, s'). \tag{6.34}$$

$M(\pi)$ can be treated as a standard MDP, and the residual policy f_θ is a policy within this MDP that can be learned with standard RL techniques.

RRL approaches have proven effective to improve on base policies and to be more data-efficient than learning policies from scratch. Extensions of RPL have been proposed to solve both manipulation [73, 172] and navigation [161] tasks in robotics. However, they share the common limitation of relying on a classical controller as base policy π.

Part III

Contributions

Chapter 7

Fast Instance Segmentation

Contributions for the fast instance segmentation learning project are presented in the papers *Fast Object Segmentation Learning with Kernel-based Methods for Robotics* [21] (Chapter 10) and *Learn Fast, Segment Well: Fast Object Segmentation Learning on the iCub Robot* [19] (Chapter 11).

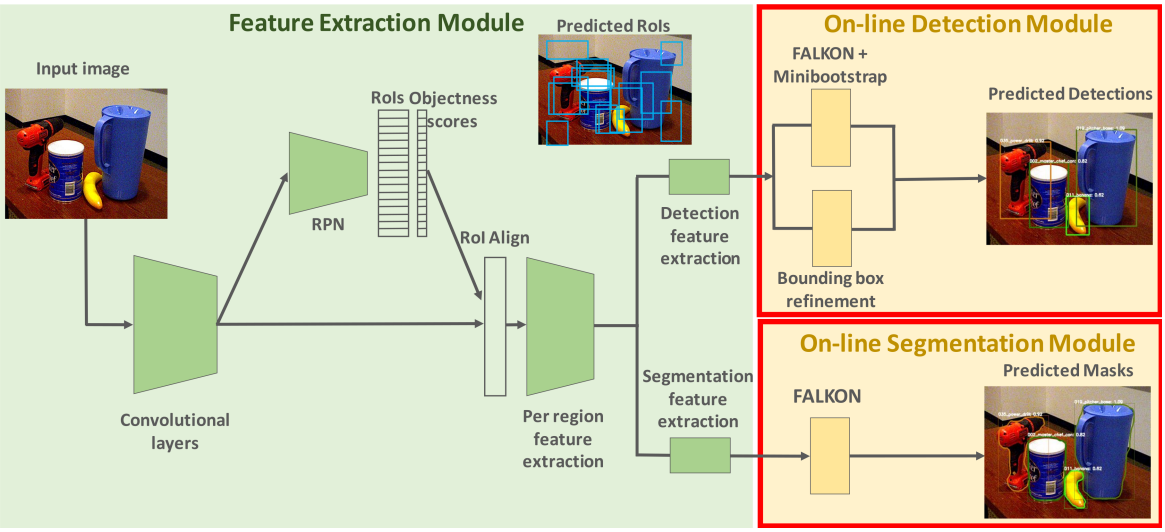


Figure 7.1 *On-line Object Segmentation* pipeline.

In *Fast Object Segmentation Learning with Kernel-based Methods for Robotics*, I proposed a method for fast learning of the task of instance segmentation on novel classes. This approach for *On-line Object Segmentation* (O-OS) is composed of three main components:

- The module for feature extraction, represented in green in Fig. 7.1. This is composed of the first layers of a pre-trained Mask R-CNN [66] architecture. This is used to extract convolutional feature maps for a set of *RoI*s.

- The module for on-line object detection, trained on the new classes, uses the per-region features computed by the feature extractor to predict their class and to refine their bounding box with O-OD [114, 115].
- The proposed *On-line Segmentation Module* predicts masks of the new objects. This is composed of N FALKON [169] binary classifiers, being N the number of the new classes. The classifiers are trained on the per-pixel convolutional features computed by the first layers of the Mask R-CNN segmentation head for each *RoI*. Each classifier predicts whether the pixel in the considered location represents the corresponding class or the background.

O-OS has been validated on the YCB-Video [212] dataset. It achieves similar performance to the fine-tuning of Mask R-CNN, with a significant reduction ($\sim 6\times$) of the training time. The pipeline has also been implemented on the R1 [144] robot[1], showing the feasibility of the approach for on-line learning of new objects on a real robot.

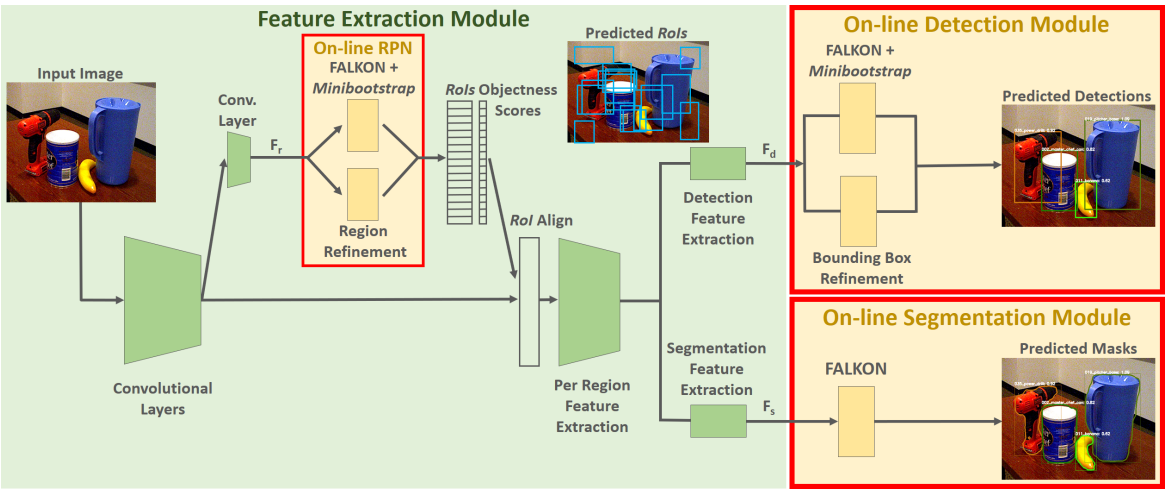


Figure 7.2 *On-line Object Segmentation and Region Proposal learning* pipeline.

I then extended the O-OS pipeline for on-line adaptation to novel visual domains in *Learn Fast, Segment Well: Fast Object Segmentation Learning on the iCub Robot*. This is done by replacing the last layers of the RPN in Mask R-CNN with the proposed *On-line RPN*. This is composed of k FALKON classifiers, being k the number of anchors in the RPN, and $4k$ RLS regressors for classification of bounding boxes that may contain an object of interest and *RoI*s refinement. Regions proposed by the *On-line RPN* are then classified and refined by O-OD [114, 115]. Within the regions classified as one of the N objects of interest, the

[1]https://youtu.be/Q0g9k8taLHc

proposed approach computes the mask of the object with the *On-line Segmentation Module* presented above. The full pipeline is shown in Fig. 7.2.

Adaptation of the *On-line RPN* improves performance, but leads to a more complex and longer training pipeline if addressed naïvely. This would require two feature extraction steps: the first to train the *On-line RPN* and the second for *On-line Detection Module* and *On-line Segmentation Module* training. This would prevent the deployment of the pipeline on the robot, where data are received in stream. To overcome this limitation, I also proposed an approximated training protocol for adaptation of the three on-line modules in few seconds. This simultaneously extracts features for the *On-line RPN*, and for the *On-line Detection Module* and the *On-line Segmentation Module*. Features for training the last two modules are computed considering *RoI*s from the pre-trained RPN of Mask R-CNN. This approximation in the training protocol led to a moderate drop in performance, but played a central role in the deployment of the pipeline on the real robot.

Finally, I adapted the training pipeline to an incremental setting where the robot is required to learn objects at different times. The incremental training pipeline differs from the off-line training protocol in the computation of the *Minibootstrap* batches for *On-line RPN* and *On-line Detection Module*. However, I showed that each tensor of features has the same probability of being considered for training with the two protocols. This adaptation allowed to reduce the amount of false positive predictions at inference[2].

[2]https://youtu.be/eLatoDWY4OI

Chapter 8

Multi-fingered Grasping with Deep Reinforcement Learning

Work done in the project for learning multi-fingered grasping with DRL resulted in the proposal of two methods for such task: *G-PAYN*, presented in the paper *A Grasp Pose is All You Need: Learning Multi-fingered Grasping with Deep Reinforcement Learning from Vision and Touch* [22] (Chapter 12) and *RESPRECT*, described in the paper *RESPRECT: Speeding-up Multi-fingered Grasping with Residual Reinforcement Learning* [24] (Chapter 13).

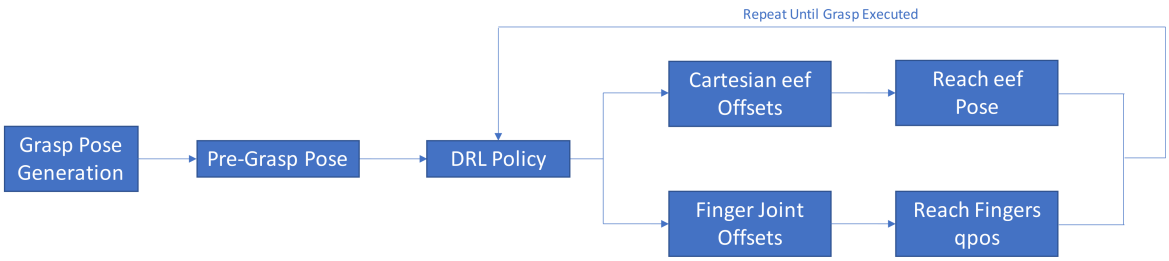


Figure 8.1 *G-PAYN* and *RESPRECT* grasping pipeline.

G-PAYN and *RESPRECT* are two methods to train DRL-based closed-loop grasping policies. DRL policies trained with the two methods form part of the pipeline presented in Fig. 8.1 which has been deployed on the iCub [123] humanoid for grasping objects. This is composed of two stages:

- It starts by computing a grasp pose for the object of interest with an external algorithm for grasp pose synthesis (*Superquadrics* [196] or *VGN* [11]). The end-effector is then moved to a pre-grasp pose close to this latter.

- Starting from the pre-grasp pose, the DRL policy predicts cartesian end-effector displacements and finger joint offsets until the grasp is executed.

The two approaches have been designed to be deployed on the real robot without adapting state and action spaces with respect to the policies trained in simulation.

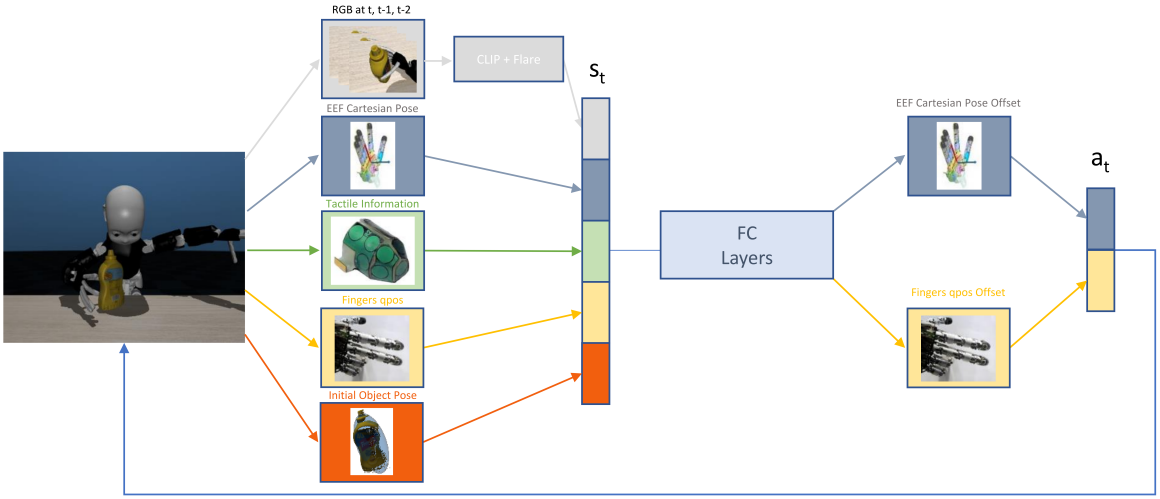


Figure 8.2 *G-PAYN* DRL policy.

As shown in Fig. 8.2, *G-PAYN* learns grasping policies from RGB images (processed through CLIP [156] for visual feature extraction), proprioceptive information from the robot (the cartesian pose of the end-effector and the position of finger joints), binary tactile information from sensors mounted on the fingertips of the robot, and an estimate of the pose of the object to grasp (this is computed at the beginning of the grasping episode and kept constant during grasp execution). Starting from this information, *G-PAYN* learns the weights of two fully-connected layers to predict cartesian and finger joint displacements. This is done in two steps:

- It firstly fills a *replay buffer* of grasping demonstrations acquired with an automatic procedure for collection of such data. This procedure starts from the same pre-grasp pose used for the DRL policy, approaches the object on a straight line, and finally closes the fingers to grasp the object. This addresses the problem of collecting grasping demonstrations for example via teleoperation.

- Starting from this *replay buffer*, it trains grasping policies with SAC [62].

For *G-PAYN* training and evaluation, I deployed a MuJoCo [193] simulated environment for the iCub, which has been made publicly available as a further contribution of the paper.

G-PAYN outperforms all the DRL baselines and achieves a similar success rate to the demonstrations acquired for training initialization, outperforming them in half of the experiments. Furthermore, policy trained in simulation with *G-PAYN* can be deployed on the real robot without requiring any adaptation of action and state spaces[1].

Training policies with *G-PAYN* is unfeasible on the real robot due to the requirement of filling a *replay buffer* with real grasping demonstrations and for the amount of training timesteps necessary for policies optimization ($\sim 5M$).

To overcome these limitations, I proposed *RESPRECT*, an RRL method to learn a residual grasping policy for a new object on top of a policy pre-trained in simulation with *G-PAYN* on a large dataset of different objects.

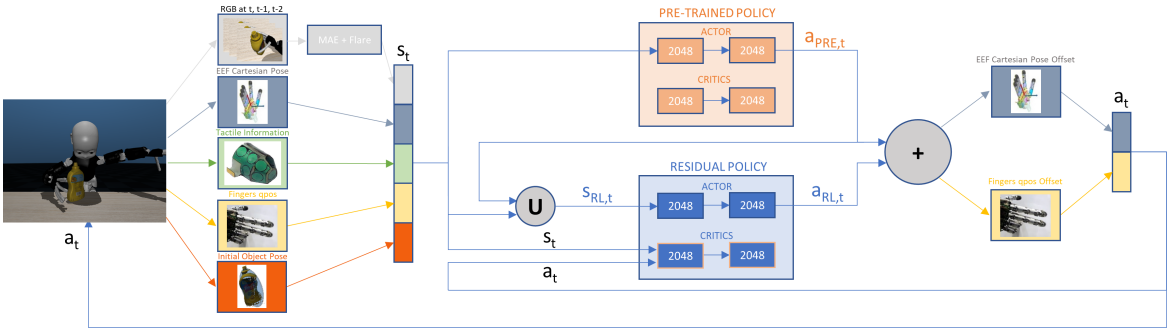


Figure 8.3 *RESPRECT* DRL policy.

Training a residual policy on top of another policy trained with DRL allows to learn residual policies for tasks where a classical base controller is not available, but also to further speed-up the training by leveraging some components of the base policy. The DRL policy proposed in *RESPRECT* is presented in Fig. 8.3. *RESPRECT*'s state and action spaces are the same as in *G-PAYN*, with the exception of the visual feature extractor.

While the weights of the pre-trained policy remain fixed throughout the training of the residual policies, *RESPRECT*'s residual policies are trained with a modified version of SAC [62].

- For the residual *Actor*, I consider as input the concatenation between the state s_t as shown in Fig. 8.3 and the action produced by the pre-trained policy. This allows the policy to compute the residual action conditioned not only to the current state of the system, but also to the action produced by the pre-trained component.

- *Critics* in the residual policy, instead, are fed with the state s_t and the sum of the actions predicted by the pre-trained and residual policies. This allows to train the residual

[1] https://youtu.be/qc6gksKH3Mo

Critics starting with the weights of the pre-trained counterpart and to speed-up the initial stage of the training.

I experimentally validated *RESPRECT* both in simulation and on the real iCub. In simulation, it trains policies five times faster than *G-PAYN*, without using any grasping demonstrations, outperforming *Meta Reinforcement Learning* [160] and fine-tuning baselines. Finally, I showed that *RESPRECT* can be effectively used to learn a multi-fingered grasping policy on the real iCub robot, starting from the same base policy pre-trained in simulation used for the simulated evaluation, and using only visual, tactile and proprioceptive information from the robot[2].

[2]https://youtu.be/JRsBLVclhpg

Chapter 9

Toward Long-Horizon Manipulation Tasks

In the perspective of proposing new methods for learning long-horizon manipulation tasks, I collected a dataset, *LHManip*, for such tasks. This has been presented in the paper *LHManip: A Dataset for Long-Horizon Language-Grounded Manipulation Tasks in Cluttered Tabletop Environments* [23].

Figure 9.1 Exemplar *LHManip* episode.

I collected *LHManip* via teleoperation, tracking human fingers and wrist motion with a *Motion Capture* system to guide the end-effector and the gripper opening of a *Franka Panda* [63] manipulator. The main features of *LHManip* are the following:

- It comprises 200 episodes, demonstrating 20 different manipulation tasks of everyday objects performed in a cluttered tabletop environment. Each of these tasks is described through a single natural language instruction, and different episodes of the same task either present differences in the manipulated objects, or in the environment. Fig. 9.1 shows some images from an episode in *LHManip*.
- Each observation in the dataset is composed of RGB-D images from a wrist-mounted and two static cameras, and proprioceptive information of the robot. Each action

describes the cartesian displacement of the end-effector of the robot and the position offset applied to the gripper opening.

LHManip is part of the *Open X-Embodiment* [138] dataset. This has been collected in a collaboration between 21 institutions with the aim of learning a generalist policy for robotic manipulation tasks.

Part IV

Included Publications

Chapter 10

Fast Object Segmentation Learning with Kernel-based Methods for Robotics

Federico Ceola, Elisa Maiettini, Giulia Pasquale, Lorenzo Rosasco and Lorenzo Natale

ABSTRACT

Object segmentation is a key component in the visual system of a robot that performs tasks like grasping and object manipulation, especially in presence of occlusions. Like many other computer vision tasks, the adoption of deep architectures has made available algorithms that perform this task with remarkable performance. However, adoption of such algorithms in robotics is hampered by the fact that training requires large amount of computing time and it cannot be performed on-line.

In this work, we propose a novel architecture for object segmentation, that overcomes this problem and provides comparable performance in a fraction of the time required by the state-of-the-art methods. Our approach is based on a pre-trained Mask R-CNN, in which various layers have been replaced with a set of classifiers and regressors that are re-trained for a new task. We employ an efficient Kernel-based method that allows for fast training on large scale problems.

Our approach is validated on the YCB-Video dataset which is widely adopted in the computer vision and robotics community, demonstrating that we can achieve and even surpass performance of the state-of-the-art, with a significant reduction ($\sim 6\times$) of the training time.

The code to reproduce the experiments is publicly available on GitHub[1].

[1] https://github.com/robotology/online-detection

10.1 Introduction

Object segmentation is a key task in computer vision and robotics. A precise localization of the object on the image plane is key to solve problems like pose estimation [212, 219, 200], refinement [100, 118] and grasping or manipulation [45].

Like many other computer vision tasks (e.g., image classification and object detection), for decades this was mostly addressed with feature extraction (e.g., sparse coding [72], Fisher vectors [151]) followed by Kernel methods [56] or structured prediction approaches [85, 109]. In recent years, instead, end-to-end training of deep architectures became mainstream since it showed to provide superior performance [108, 66, 126].

However, deep learning is known to require large training data to optimize complex architectures with expensive and long training procedures. While the problem of training data has recently been alleviated by synthetic image generation [164, 46], the computational burden remains.

Lately, the literature of one-shot and few-shot object segmentation in video [15] presented methods based on the idea of fine-tuning only part of the network on-line [197, 134]. This approach can learn to segment novel objects in shorter time than standard end-to-end approaches, while keeping performance. However, the procedure of fine-tuning a deep network in general does not guarantee generalization properties, since it involves a non convex optimization and several hyper-parameters, while it can be still prohibitive for robotic settings which require learning to happen very fast (e.g., in seconds or minutes).

Recently, it was shown that pre-trained deep features can be efficiently re-used also to train Kernel-based methods for tasks like image classification [25] object detection and even segmentation [51, 59]. The advantage of Kernel-based methods, among others, is that the number of hyper-parameters to tune is smaller (e.g. in [169], one regularization parameter, one Kernel parameter) and convergence and generalization properties are guaranteed [169]. This approach was recently adopted to achieve high performance and remarkably faster training times in interactive object learning applications for object recognition [145] and detection [115] in robotics. By relying on the above pipelines, a robot can observe a short image sequence of the objects of interest and seamlessly train to detect them right afterwards[2,3].

In this paper, we build on this prior work and present a method to re-use general-purpose deep features (e.g., pre-trained on COCO [105]) to train Kernel methods for the *dense* task of image segmentation. As evidenced from prior work [115, 59, 51], the main challenge lies in

[2]https://youtu.be/HdmDYIL48H4
[3]https://youtu.be/eT-2v6-xoSs

the difficulty of training a *non-linear* classifier over the *large* training sets that characterize object detection and segmentation problems (where each sample is a region or a single pixel in an image).

To this end, we feed deep features to FALKON [169], a recently proposed Nyström-based Kernel method optimized for large-scale problems. We instantiate one per-pixel FALKON classifier for every class and train it on pixel-wise features extracted from the second-last layer of the network.

We first solve the detection task by applying the same approach, which was presented in [115]. Then, we feed the dense features extracted from the detected regions to FALKON classifiers to perform figure-ground segmentation within each region. To tackle the well-known problem of dataset (positive-negative) imbalance which characterizes the task of object detection, we use the bootstrapping technique from [115]. Instead, to manage the great quantity of data in the figure-ground segmentation task, we apply data subsampling in our training procedure.

As network, we use the region-based Mask R-CNN [66], to which we basically replace every output layer with a set of FALKON classifiers or regressors. The resulting architecture is clean and flexible, allowing to easily train accurate instance segmentation on small datasets in a few minutes, but possibly also on large ones.

We benchmark our method on the YCB-Video dataset [212], a state-of-the-art benchmark for pose estimation and tracking in challenging conditions for robotics, featuring clutter and occlusion across varied objects. In fact, while pose estimation results are widespread over this latter, but also other robotics datasets (see, e.g., the BOP challenge [69]), it is less common for robotic papers to present the partial results over the segmentation task –which, however, is almost always part of their pipelines. Hence, we also hope that this work could pave the way to other works in robotics, to present not only pose estimation but also intermediate segmentation results.

The remaining of this paper is organized as follows. In Sec. 10.2 we review the state-of-the-art in the fields of instance segmentation and efficient learning strategies for visual tasks in robotics. Then in Sec. 10.3, we describe the proposed approach and in Sec. 10.4 we report on the performed empirical validation. Finally, in Sec. 10.5, we summarize the main contributions and identify directions for future works.

10.2 Related Work

In this section, we overview the main state-of-the-art approaches for object segmentation (Sec. 10.2.1), then we present the literature that tackles the problem of fast adaptation of robotic vision systems to novel tasks (Sec. 10.2.2).

10.2.1 Object Instance Segmentation

The task of *instance segmentation* [126] aims at classifying all the pixels in an image, while identifying different instances of the categories of interest. Note that, this task is different from *semantic segmentation*, which has the similar objective of classifying every pixel of an image but not being aware of the different instances. While the field of semantic segmentation is mainly dominated by approaches based on Fully Convolutional Networks (FCNs) [108, 179], the literature in instance segmentation can be divided in the following three groups.

Region-based approaches. Methods in this group split the segmentation task into four main stages: (i) a convolutional feature map is generated by a FCN applied to the whole image, (ii) a list of Regions of Interest (RoIs) is generated by a region proposal method, (iii) a RoI pooling layer [165] warps each RoI and extracts per-RoI features from the convolutional map and, finally, (iv) two sibling shallow Neural Networks predict detections and masks for each feature map. The main representative example of this group is Mask R-CNN [66] which extends the object detection architecture Faster R-CNN [165] by adding, in parallel to the final detection layers, a further output layer for binary mask prediction. Note that, this is in contrast to latest approaches [37, 99, 152], where the classification depends on the prediction of the masks. Mask R-CNN is devised as a monolithic architecture which trains end-to-end, via stochastic gradient descent and back-propagation, all the components for region proposals and feature extraction, object detection and segmentation. The network optimization is carried out by minimizing a “multi-task loss" [165, 66], which accounts for the contribution of classification, bounding box and mask precision (see [165, 66] for details).

Pixel-based approaches. Methods of this group predict a so-called “auxiliary" information for each pixel (like, e.g, energy levels [5]) and then a clustering algorithm groups pixels into object instances based on such information. An example of this approach is presented in [5], where the watershed transform is combined with a deep convolutional network to produce an energy map of the image where object instances are represented as energy basins. Then, they

perform a cut at a single energy level to yield connected components corresponding to object instances.

Snake approaches. Methods of this group deform an initial coarse contour to the object boundary by optimizing a handcrafted or learned energy measure with respect to the contour coordinates. In [106], for instance, the presented pipeline uses a graph convolutional network to predict vertex-wise offsets for contour deformation. In [149], instead, the object's contour is not treated as a general graph, but it leverages the cycle graph topology and uses the circular convolution for efficient feature learning on a contour.

All the aforementioned methods of the state-of-art are "monolithic" deep architectures, trained end-to-end via back-propagation and stochastic gradient descent, with long training times that are not suited for on-line robotic applications. In this work, we propose a region-based method for object segmentation that, building on previous work [115], addresses this issue, allowing for fast model learning.

10.2.2 Fast Object Detection Methods in Robotics

Fast and efficient adaptation to novel tasks is critical for robots that need to operate in unconstrained environments. Previous works in robotics analyze this problem and propose solutions for object recognition [145] and object detection [114, 115]. In both cases, the proposed pipelines rely on hybrid architectures that integrate deep Convolutional Neural Network's (CNN) feature descriptors with efficient "shallow" Kernel-based methods (like, e.g., FALKON [169]) for respectively classification or detection. In these works, the key aspect to achieve on-line training time is to decouple the learning of the features and region proposals extractor from the optimization of the Kernel-based methods. Moreover, in [115] an approximated bootstrapping procedure for selecting hard negative samples has been proposed, to address the well-known foreground-background imbalance problem in object detection.

In this work, we extend the pipeline proposed in [115], to also perform object segmentation. Following the same principle as in [115], in our proposed approach, the training of the features and region proposals extractor is decoupled by the optimization of the detection and segmentation methods, allowing for a fast model re-adaptation.

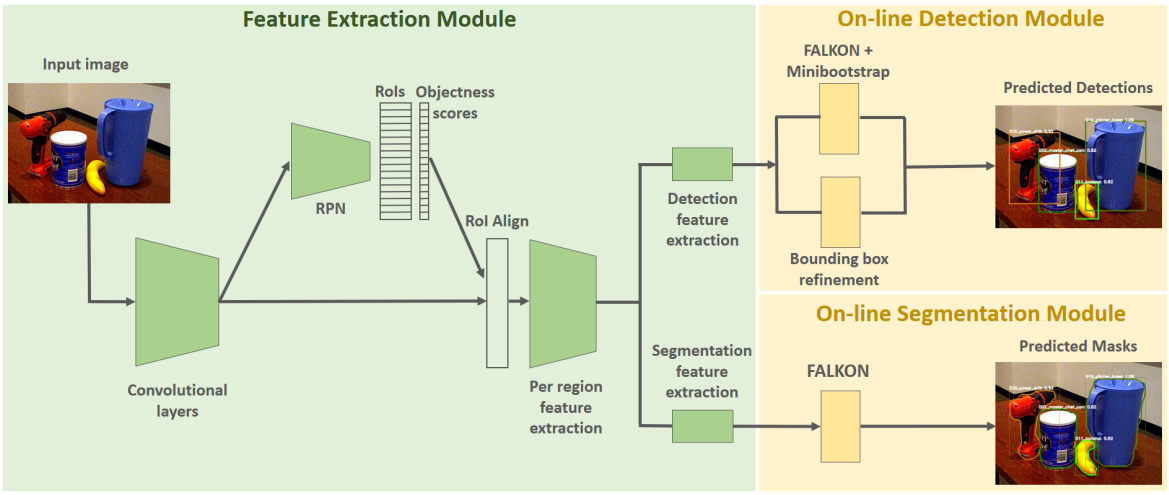


Figure 10.1 Overview of the proposed pipeline. The *Feature Extraction Module* is composed of Mask R-CNN's first layers trained off-line on the FEATURE-TASK. For each input image, it extracts per RoI features to train on-line on the TARGET-TASK the *On-line Detection Module* and the *On-line Segmentation Module*. The *On-line Detection Module* classifies the RoIs and predicts the corresponding detections. The *On-line Segmentation Module* predicts masks for each detection. Note that the picture represents the pipeline at training time. At test time, the detection output is fed as input to the RoI Align.

10.3 Methods

In this work, we present an instance segmentation method, which allows to learn to predict masks of previously unseen objects (referred to as TARGET-TASK) in a fraction of the time required by state-of-the-art approaches. To do this, it relies on some components of a deep learning based instance segmentation network trained once and off-line on the available data, depicting a different set of objects (referred to as FEATURE-TASK).

In this section, we describe the proposed approach. Sec. 10.3.1 provides an overview of the pipeline, while in Sec. 10.3.2 the proposed on-line learning strategy for instance segmentation is described.

10.3.1 Overview of the Pipeline

The pipeline proposed in this work is mainly composed of three modules, as depicted in Fig. 10.1:

- The **Feature Extraction Module** employs the first layers of the Mask R-CNN architecture, trained on the FEATURE-TASK, to extract two sets of convolutional features for a set of RoIs, one for **On-line Detection** and one for **On-line Segmentation**.

- The **On-line Detection Module** uses the per-region features computed by the feature extractor to predict their class and to refine their bounding box with the approach described in [115].
- The **On-line Segmentation Module**, instead, predicts the masks of the instances contained in the RoIs with the approach proposed in this work and described in Sec. 10.3.2. As the previous module, also this module is trained on the TARGET-TASK.

The main contribution relies on the latter module. Indeed, we propose a fast and efficient learning strategy that allows to update the instance segmentation model in few minutes. Moreover, we integrate it in the on-line detection pipeline presented in [115], extending it and proposing a novel approach. This permits fast adaptation time as new data is available for both object detection and segmentation.

Learning protocol. Our learning procedure is composed of two stages. Firstly, Mask R-CNN is trained off-line on the FEATURE-TASK via backpropagation, following the training protocol proposed in [66]. Then, the on-line training of both the detector and the mask predictor on the TARGET-TASK are performed in three sub-steps. First, some of the layers of the Mask R-CNN architecture learned in the first phase on the FEATURE-TASK (specifically the backbone, the RPN and the convolutional layers following the Roi Align, depicted as green blocks in Fig. 10.1), are used to jointly extract the two sets of features needed to train the *On-line Detection Module* and the *On-line Segmentation Module* (represented by yellow blocks in Fig. 10.1). These two modules are then sequentially trained with the on-line learning strategy described in Sec. 10.3.2.

10.3.2 On-line Learning Strategy

As mentioned in Sec. 10.3.1, the trainings of the *On-line Detection Module* and of the *On-line Segmentation Module* are performed by employing two different sets of features computed by the *Feature Extraction Module*. These are extracted in two different points of the Mask R-CNN architecture. Specifically, features necessary to train the *On-line Detection Module* are extracted from the set of one-dimensional tensors, which in Mask R-CNN are given as input to the last fully connected layers of the detection head. Features necessary to train the *On-line Segmentation Module*, instead, are contained in the convolutional feature maps produced as output by the penultimate convolutional layer of the mask head in Mask R-CNN. Once the sets of features are extracted for all the training images, the *On-line Detection*

Module and the *On-line Segmentation Module* can be trained with the procedures described below.

On-line Detection. For the training of the *On-line Detection Module*, we adopt the method described in [115], but we use the improved implementation of the *Feature Extraction Module* proposed in [20]. Indeed, such improvement allows to increase the quality of the features, due to the substitution of the first layers of Faster R-CNN [165], with the ones from Mask R-CNN [66]. As in [115], the last fully connected layers of the detection head in Mask R-CNN are substituted with N FALKON [169, 120] binary classifiers (N being the number of classes of the TARGET-TASK) for class prediction, and by $N \times 4$ Regularized Least Squares (RLS) regressors for the refinement [59] of the RoIs proposed by the RPN. To deal with the well-known problem of foreground-background imbalance and with the high redundancy among the negative examples in object detection problems [103], the N FALKON classifiers are trained with the *Minibootstrap* proposed in [115]. This procedure is an adaptation of the Hard Negatives Mining strategy [59] and it allows to train the classifiers in a few seconds by visiting a random subset of negative examples.

On-line Segmentation. In Mask R-CNN [66], the mask head is a shallow Fully Convolutional Network (FCN), which for every RoI computes N squared masks (one for each of the N classes) of fixed size $s \times s$. In particular, the last layer of the segmentation branch (i.e., the per-pixel mask predictor) is a convolutional layer with N channels, kernel size 1 and stride 1 that receives as input a convolutional feature map of size $s \times s \times f$ (where f is the number of channels). Therefore, the output tensor of the segmentation head has dimension $s \times s \times N$ and the ijn^{th} element of such tensor represents the probability of the pixel in the location ij of belonging to an instance of the n^{th} class. In this work, we substitute the N kernels for per-pixel classification with N FALKON binary classifiers. We flatten the $s \times s \times f$ feature map into a list of $s \times s$ feature tensors of size f associated to specific locations in the RoIs and use them as samples to train the FALKON classifiers. At training time, we consider the ground-truth bounding boxes as RoIs. The ij^{th} element of the input feature map of size $s \times s \times f$ is considered as a positive training sample for the n^{th} classifier if the ij^{th} pixel of the associated ground-truth mask belongs to the foreground and the ground-truth class is n. Otherwise, if it is a background pixel within a ground-truth bounding box of class n, it is considered as a negative sample for the n^{th} classifier. To speed-up the training procedure, both the positive and the negative instances are subsampled by a fixed factor r. We employ FALKON to address the per-pixel classification problem since, as shown in [115] (specifically, refer to Tab. 6 in [115]), among other equivalent choices, it allows to achieve the best time-accuracy trade-off (refer to [169] for details).

Hyper-parameters. For both the detection and segmentation modules, we use FALKON with a Gaussian Kernel. The main hyper-parameters of the proposed method are:

- The standard deviation σ of the Gaussian Kernel, the regularization parameter λ and the number of Nyström centers M of the FALKON classifiers of both the *On-line Detection Module* and the *On-line Segmentation Module*.
- The *Minibootstrap* parameters to train the *On-line Detection Module*, i.e., the number of batches n_B and the size of such batches BS.
- The sampling factor r of the training examples in the *On-line Segmentation Module*.

For our experiments, we rely on the criterion proposed in [115] to set the *Minibootstrap*'s parameters, we cross-validate the FALKON's parameters as explained in Sec 10.4.1 and we report on an empirical analysis of the impact of the sampling factor r.

10.4 Experiments

In this section, we report the experiments performed to validate the proposed approach. In Sec. 10.4.1, we describe the experimental setup. In Sec. 10.4.2, we provide the results obtained on the YCB-Video benchmark. Finally, in Sec. 10.4.3, we report on ablation studies to analyze our approach.

10.4.1 Experimental Setup

In this work, we consider Mask R-CNN as baseline for comparison with the proposed pipeline (**Ours**). We start with Mask R-CNN trained on the FEATURE-TASK and we fine-tune the output layers on the TARGET-TASK (i.e., the fully connected layers for detection and the last convolutional layer for masks prediction). We refer to this architecture and training method as **Mask R-CNN (output layers)**. We use ResNet50 [67] as backbone of Mask R-CNN for both the **Mask R-CNN (output layers)** baseline and the *Feature Extraction Module* of the proposed approach (**Ours**). To set the training hyper-parameters, for both methods we perform a cross-validation on a validation set. Specifically, the model chosen for the **Mask R-CNN (output layers)** baseline has been trained for the minimum number of epochs at which the highest segmentation accuracy is achieved on the validation set. In **Ours**, instead, the validation set is used to select σs and λs of the FALKON classifiers, both in the *On-line Detection Module* and in the *On-line Segmentation Module*. To compare the performance

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (output layers)	**76.87**	74.12	68.56	64.80	1h 57m 1s
Ours	76.46	**74.66**	**68.71**	**64.96**	**19m 32s**

Table 10.1 Benchmark on the YCB-VIDEO dataset. We compare the accuracy achieved by (**Mask R-CNN (output layers)**) (first row) and by the proposed approach (**Ours**) (second row).

between the two approaches, we consider three metrics: the mean Average Precision (mAP), as defined in [50] of the predicted detections (**mAP bbox(%)**) and of the predicted masks (**mAP segm(%)**) and the training time. Concerning the bounding boxes and masks IoU to be considered as positives matches in the mAP computations, we set the thresholds to 50% and 70%. As regards the training time[4], for **Mask R-CNN (output layers)** it is the time necessary to fine-tune Mask R-CNN via backpropagation. For **Ours**, instead, it is composed of the feature extraction time and of the training time of the on-line modules. The former one in practical robotic applications can happen during the data acquisition phase[2,3], which can be performed, for instance, as in [146, 113, 213, 186], where the ground-truth is collected with automatic procedure.

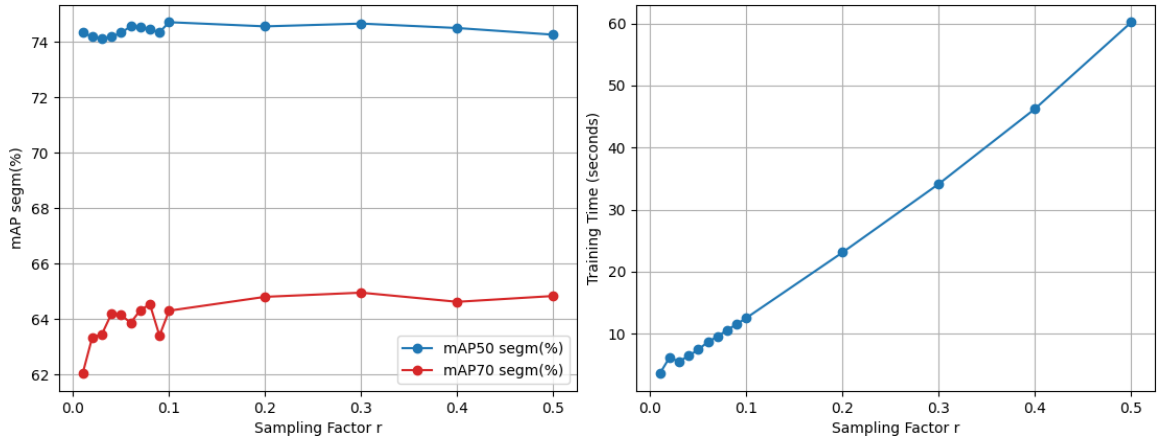
Datasets. In our experiments we consider the YCB-Video [212] dataset. Specifically, we use as training set a sample of the 80 training sequences, by selecting one every ten frames, resulting into a set of 11320 images. For testing, instead, we use the 2949 images included in the *keyframe* set. To select the training hyper-parameters, according to the training protocol described in Sec.10.4.1, we use a set of 1000 images, randomly sampled from the 12 test sequences, which are not included in the *keyframe* set. The 80 categories in the MS COCO [105] dataset, instead, compose the FEATURE-TASK in some of our experiments.

10.4.2 Benchmark on the YCB-Video Dataset

We validate the proposed on-line segmentation approach by considering the 21 objects from the YCB-Video dataset as TARGET-TASK and MS COCO as FEATURE-TASK. As described in Sec. 10.4.1, we compare the performance obtained by our approach (**Ours**), with the **Mask R-CNN (output layers)** baseline. As in [115], to train the *On-line Detection Module* we set the *Minibootstrap*'s batch size *BS* to 2000, but, given the higher amount of training images with respect to the benchmark reported in [115] (Table 2), we increase the number of batches n_B to 15. The number of Nyström centers M of the FALKON classifiers of both the *On-line Detection Module* and the *On-line Segmentation Module* is set to 2000.

[4]All the experiments reported in this paper have been performed on a machine equipped with Intel(R) Xeon(R) E5-2690 v4 CPUs @2.60GHz, and a single NVIDIA(R) Tesla P100 GPU.

Figure 10.2 Randomly chosen predictions of **Ours** on the YCB-Video *keyframe* set.



(a) (b)

Figure 10.3 Segmentation mAPs and training time for increasing values of the sampling factor r.

The sampling factor r of the segmentation training samples is set to 0.3 (refer to Sec. 10.4.3 for an analysis of the impact of this parameter).

Results reported in Tab. 10.1 show that our approach, while being $\sim$6 times faster than the baseline, outperforms **Mask R-CNN (output layers)** in the segmentation accuracy computed with both IoU thresholds, while achieving comparable performance in terms of detection mAP. Train time of **Ours** is composed of 18min:23s for feature extraction, 35s to train the *On-line Detection Module* and 34s to train the *On-line Segmentation Module*. This shows that, once the features are extracted, the TARGET-TASK can be learned extremely fast. We report in Fig. 10.2, some random images from the test set with the overlaid predicted masks.

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)
Mask R-CNN (full)	89.76	91.29	86.05	**80.41**
Ours	**90.98**	**92.28**	**86.90**	80.17

Table 10.2 Ablation study with YCB-VIDEO dataset used as both FEATURE-TASK and TARGET-TASK. We compare the results of the full training of Mask R-CNN (**Mask R-CNN (full)**) (first row) with the proposed approach (**Ours**) (second row) considering as feature extractor the weights learned with the training in the first row.

10.4.3 Ablation Studies

In this section, we provide three ablation studies of the proposed pipeline to further validate our approach. Firstly, we evaluate segmentation accuracy by decoupling such task from the detection one. Then, we analyze the impact on accuracy and training time of the sampling factor r (see Sec. 10.3.2). Finally, to compare the accuracy of our method with the one of Mask R-CNN fully trained on the TARGET-TASK, we benchmark our approach in a scenario in which FEATURE-TASK and TARGET-TASK correspond.

Segmentation of ground-truth bounding boxes. In Mask R-CNN [66] and in our pipeline, the segmentation performance depends on the detection. To separately evaluate the segmentation errors, we consider the ideal case in which the bounding boxes proposed by the detection branch correspond to the ground-truths. FEATURE-TASK and TARGET-TASK are the same as the ones considered in Sec. 10.4.2. The achieved segmentation mAP is **94.66** and **81.78** when the IoU thresholds are set respectively to 50% and 70%. We compare the obtained results with **Mask R-CNN (last layers)** as in Sec. 10.4.2 where, as for **Ours**, we consider the ground-truth boxes as proposed detections. For the baseline, the segmentation mAP is **93.70** and **79.46** when the IoU thresholds are set to 50% and 70%, showing the effectiveness of the proposed *On-line Segmentation Module*.

Training samples vs. training time and mAP. Since the time required to train the *On-line Segmentation Module* is critical in robotic applications, we provide an analysis of how the choice of the sampling factor r affects both the training time and the segmentation accuracy of such module. As it can be observed in Fig. 10.3 (a), the sampling factor r can be pushed to the extreme while preserving the segmentation accuracy in terms of both the considered IoU thresholds. By using a small fraction of training examples, the training times can be reduced until they become in the order of a few seconds, as it can be noticed in Fig. 10.3 (b). When the sampling factor r is 0.01, the number of per class positives and negatives training samples is ranging from $\sim 1k$ to $\sim 4k$.

YCB-Video for both FEATURE-TASK and TARGET-TASK. The goal of this work is to quickly learn to segment previously unseen objects. To this aim, in Sec. 10.4.2 we

showed the efficiency of the proposed approach when the TARGET-TASK is different form the FEATURE-TASK. Nevertheless, to further validate our pipeline, we evaluate the performance of our method when the two tasks correspond. In particular, we set them by considering the 21 object identification task in the YCB-Video dataset (the TARGET-TASK defined in Sec. 10.4.1). In this case, we train, as baseline, the entire Mask R-CNN network (starting from the weights pre-trained on the COCO dataset) on the TARGET-TASK (**Mask R-CNN (full)**). Then, we use such weights to extract the features needed to train the *On-line Detection Module* and the *On-line Segmentation Module* with the approach described in Sec. 10.3.2 (**Ours**). The obtained accuracy are reported in Tab. 10.2. Since the full training of Mask R-CNN on the TARGET-TASK is necessary both to obtain a baseline for our comparison and subsequently to train our pipeline, for this experiment the training times cannot be compared. Results reported in Tab. 10.2 show that, as in previous experiments, our approach provides comparable performance with respect to the baseline in terms of detection and segmentation mAP with both the IoU thresholds.

10.5 Conclusions

Fast learning and efficient adaptation are fundamental for robots that need to quickly adjust their vision systems to ever-changing environments. In this perspective, this work presents a system for instance segmentation that extends and improves previous on-line learning approaches for object detection [115]. The proposed method combines Mask R-CNN for feature extraction and two sets of FALKON classifiers to efficiently perform object detection and segmentation. The resulting pipeline allows for fast adaptation to novel tasks in a fraction of the time of the deep learning based counterparts, representing a step toward the implementation of more adaptive robotic vision systems.

In Sec. 10.4.3, we showed that we can achieve high per-pixel prediction accuracy with few training samples. We believe that this can be pushed to the extreme by integrating few-shots learning techniques in the proposed pipeline. Moreover, given the importance of instance segmentation for other more complex tasks, such as 6D pose estimation and object grasping, the proposed approach may represent a starting point towards the on-line adaptation of such tasks.

Chapter 11

Learn Fast, Segment Well: Fast Object Segmentation Learning on the iCub Robot

Federico Ceola, Elisa Maiettini, Giulia Pasquale, Giacomo Meanti, Lorenzo Rosasco and Lorenzo Natale

ABSTRACT

The visual system of a robot has different requirements depending on the application: it may require high accuracy or reliability, be constrained by limited resources or need fast adaptation to dynamically changing environments. In this work, we focus on the instance segmentation task and provide a comprehensive study of different techniques that allow adapting an object segmentation model in presence of novel objects or different domains.

We propose a pipeline for fast instance segmentation learning designed for robotic applications where data come in stream. It is based on an hybrid method leveraging on a pre-trained CNN for feature extraction and fast-to-train Kernel-based classifiers. We also propose a training protocol that allows to shorten the training time by performing feature extraction during the data acquisition. We benchmark the proposed pipeline on two robotics datasets and we deploy it on a real robot, i.e. the iCub humanoid. To this aim, we adapt our method to an incremental setting in which novel objects are learned on-line by the robot.

The code to reproduce the experiments is publicly available on GitHub[1].

[1] https://github.com/hsp-iit/online-detection

11.1 Introduction

Perceiving the environment is the first step for a robot to interact with it. Robots may be required to solve different tasks, as for instance grasping an object, interacting with a human or navigate in the environment avoiding obstacles.

Different applications have different requirements for the robot vision system. For example, for an application in which a robot interacts with a predefined set of objects, fast learning is not the primary requirement. On the other hand, when a robot is operating in a dynamic environment (for instance a service robot operating in a hospital, a supermarket or a domestic environment), fast adaptation is fundamental.

The *computer vision* literature is progressing at fast pace providing algorithms for object detection and segmentation that are remarkably powerful. These methods, however, are mostly based on deep neural networks and quite demanding in terms of training samples and optimization time. For this reason, they are badly suited for applications in robotics that require fast adaptation. Because the dominant trend in computer vision is to push performance as much as possible, comparably little effort is spent to propose methods that are designed to reduce training time. To fill this gap, in this work, we propose a comprehensive analysis in which we study various techniques for adaptation on a novel task. In particular, we consider approaches based on deep neural networks and on a combination of deep neural networks and Kernel methods, focusing on the trade-off between training time and accuracy.

We target the instance segmentation problem which consists in classifying every pixel of an image as belonging to an instance of a known object or to the background. In particular, we consider the scenario in which the robot encounters new objects during its operation and it is required to adapt its vision system so that it is able to segment them after a learning session that is as short as possible. We observe that this scenario offers opportunities to shorten the training time, for example if we are able to perform some of the training steps (i.e., feature extraction) already during data acquisition, and we propose a new method that is specifically optimized to reduce training time without compromising performance.

Specifically, we propose an instance segmentation pipeline which extends and improves our previous work [21]. In [21], we proposed a fast learning method for instance segmentation of novel objects. One limitation of that method was to rely on a pre-trained region proposal network. In this work, we address this by making the region proposal learning on-line too. While this improves performance, it leads to a more complex and longer training pipeline if addressed naïvely as it is done in [20]. To this aim, we propose an approximated training protocol which can be separated in two steps: *(i)* feature extraction and *(ii)* fast

and simultaneous training of the proposed approaches for region proposal, object detection and mask prediction. We show that this allows to further reduce the training time in the aforementioned robotic scenario.

In addition, we provide an extensive experimental analysis to investigate the training time/accuracy trade-off on two public datasets (i.e., YCB-Video [212] and HO-3D [65]). In particular, we show that our method is much more accurate than [21], while requiring a comparable training time. Moreover, the proposed method allows to obtain accuracy similar to conventional fine-tuning approaches, while being trained much faster.

In summary, the contributions of this work are:

- We propose a new pipeline and training protocol for instance based object segmentation, which is specifically designed for fast, on-line training.
- We benchmark the obtained results on two robotics datasets, namely YCB-Video [212] and HO-3D [65].
- We provide an extensive study to compare our pipeline against conventional fine-tuning techniques, with an in-depth analysis of the trade-off between the required training time and the achieved accuracy.
- We deploy and demonstrate the proposed training pipeline on the iCub [123] humanoid robot, adapting the algorithm for an incremental setting where target classes are not known a-priori.

This paper is organized as follows. In Sec. 11.2, we review state-of-the-art approaches for instance segmentation, focusing on methods designed for robotics. Then, in Sec. 11.3, we describe the proposed training pipeline for fast learning of instance segmentation. In Sec. 11.4, we report on the experimental setup used to validate our approach. We then benchmark our approach on the two considered robotics datasets in Sec. 11.5. In Sec. 11.6, we specifically quantify the benefit of the adaptation of the region proposal. In Sec. 11.7, we simulate the robotic scenario in which data come into stream and we discuss various performance trade-offs. Then, in Sec. 11.8, we describe an incremental version of the proposed pipeline and we deploy it on a robotic platform. Finally, in Sec. 11.9 we draw conclusions.

11.2 Related Work

In this section, we provide an overview of state-of-the-art methods for instance segmentation (Sec. 11.2.1), focusing on their application in robotics (Sec. 11.2.2).

11.2.1 Instance Segmentation

Approaches proposed in the literature to address instance segmentation can be classified in the following three groups.

Detection-based instance segmentation. Methods in this category extend approaches for object detection, by adding a branch for mask prediction within the bounding boxes proposed by the detector. Therefore, as for object detection methods, they can be grouped in *(i) multi-stage* (also known as *region-based*) and *(ii) one-stage*. Methods from the first group rely on detectors that firstly predict a set of candidate regions and then classify and refine each of them (e.g. Faster R-CNN [165] or R-FCN [38]). *One-stage* detectors, instead, solve the object detection task in one forward pass of the network. Differently from *multi-stage* approaches, they do not perform any per-region operation, like e.g. per-region feature extraction and classification (see for instance, EfficientDet [190] and YOLOv3 [163]).

The representative method among the *multi-stage* approaches is Mask R-CNN [66] that builds on top of the detection method Faster R-CNN [165], by adding a branch for mask prediction (segmentation branch) in parallel to the one for bounding box classification and refinement (detection branch). In Mask R-CNN, input images are initially processed by a convolutional backbone to extract a feature map. This is then used by the Region Proposal Network (RPN) to propose a set of *Regions of Interest* (*RoIs*) that are candidate to contain an object, by associating a class-agnostic objectness score to each region. Then, the *RoI Align* layer associates a convolutional feature map to each *RoI* by *warping* and *cropping* the output of the backbone. These features are finally used for *RoIs* classification, refinement and, subsequently, for mask prediction. In the literature, many other state-of-the-art *multi-stage* approaches for instance segmentation build on top of Mask R-CNN, like Mask Scoring R-CNN [70] or PANet [107].

YOLACT [9] and BlendMask [26] are representative of *one-stage* methods. YOLACT [9] extends a backbone RetinaNet-like [103] detector with a segmentation branch. BlendMask [26], instead, extends FCOS [192] for mask predictions. An alternative paradigm for instance segmentation based on the *one-stage* detector CenterNet [228] is Deep Snake [149]. Differently from the methods mentioned above that predict per-pixel confidence within the proposed

bounding boxes, it exploits the circular convolution [149] to predict an offset for each mask vertex point, starting from an initial coarse contour.

Labelling pixels followed by clustering. Approaches in this group build on methods for semantic segmentation, which is the task of classifying each pixel of an image according to its category (being thus agnostic to different object instances). Building on these methods, approaches in the literature separate the different instances by clustering the predicted pixels. As an example, SSAP [58] uses the so-called *affinity pyramid* in parallel with a branch for semantic segmentation to predict the probability that two pixels belong to the same instance in a hierarchical manner. This is done with the aim of grouping pixels of the same instance. InstanceCut [81], instead, exploits an instance-agnostic segmentation and an instance-aware edge predictor to compute the instance-aware segmentation of an image. Finally, the method proposed in [5] learns the watershed transform with a convolutional neural network, the *Deep Watershed Transform*, given an image and a semantic segmentation. This is done to predict an energy map of the image, where the energy basins represent the object instances. This information is then used, with a cut at a single energy level, to produce connected components corresponding to different object instances.

Dense sliding window. These approaches simultaneously predict mask instances and their class-agnostic or class-specific scores. For instance, DeepMask [153] predicts in parallel a class-agnostic mask and an objectness score for each patch of an input image with a shallow convolutional neural network. InstanceFCN [36], alternatively, predicts an instance sensitive score map for each window of the considered input image. This method exploits local coherence for class-agnostic masks prediction, and, as DeepMask, per-window class-agnostic scores. Similarly, TensorMask [30] predicts class-agnostic instance masks, but it leverages on the proposed mask representation as a 4D tensor to preserve the spatial information among pixels. Moreover, the classification branch of the proposed approach outputs a class-specific score, thus improving the class-agnostic predictions provided by DeepMask and InstanceFCN.

11.2.2 Instance Segmentation in Robotics

The instance segmentation task plays a central role in robotics, not only for providing an accurate 2D scene description for a robot, but also to support other tasks like 6D object pose estimation [212] or computation of grasp candidates [183]. In the literature, the problem is tackled in different ways, depending on the target application. In [198] and [93] the problem is addressed in cluttered scenarios, while [97] and [39] propose adopting synthetic data (both

images and depth information) for training. In this work, instead, we focus on learning to segment previously unseen objects. In the following paragraphs, we will cover the main literature on this topic.

Some works propose to generalize to unseen objects in a class-agnostic fashion. However, these methods either focus on particular environments, such as tabletop settings, as in [213] and [214], or require some post-processing [89] which may be unfeasible during the robot operation.

Approaches as the ones proposed in [147] and [49] learn to segment new objects instances by interacting with them. Nevertheless, similarly to the class-agnostic approaches, they are constrained to tabletop settings.

The latest literature on *Video Object Segmentation* provides some methods for learning to segment a set of previously unseen objects in videos. They deal with the problem either in a semi-supervised way [141], leveraging on the ground-truth masks of the objects in the first frame of the video, or in an unsupervised fashion [170]. They allow to learn to segment new object instances in a shorter time than that required by the fully supervised approaches presented in Sec. 11.2.1. They typically rely on pre-training a network for instance segmentation and on the subsequent fine-tuning on the target video sequence frames [197]. Some of these approaches have been targeted for robotic scenarios. For instance, the method in [184] proposes to learn to segment novel objects in a *Human-Robot Interaction (HRI)* application, leveraging only on objects motion cues. Nevertheless, these approaches are known to suffer from changes of the objects appearance through the video sequence and error drifts [141].

We instead focus on learning to segment novel objects in a class-specific fashion, keeping the performance provided by the state-of-the-art but reducing the required training time. All the approaches mentioned in Sec. 11.2.1 rely on convolutional neural networks that require to be trained end-to-end via backpropagation and stochastic gradient descent. Despite providing impressive performance, they require long training time and large amounts of labeled images to be optimized. These constraints make the adoption of such approaches in robotics difficult, especially for robots operating in unconstrained environments, that require fast adaptation to new objects.

Incremental learning aims at learning new objects instances without degrading performance on the previously known classes. Nevertheless, these approaches rarely focus on speeding up the training of the models, which may be crucial in robotic applications. Moreover, the current literature in this field mainly focuses on object recognition [17, 116], object detection [180, 150] or semantic segmentation problems [124], while we target an instance

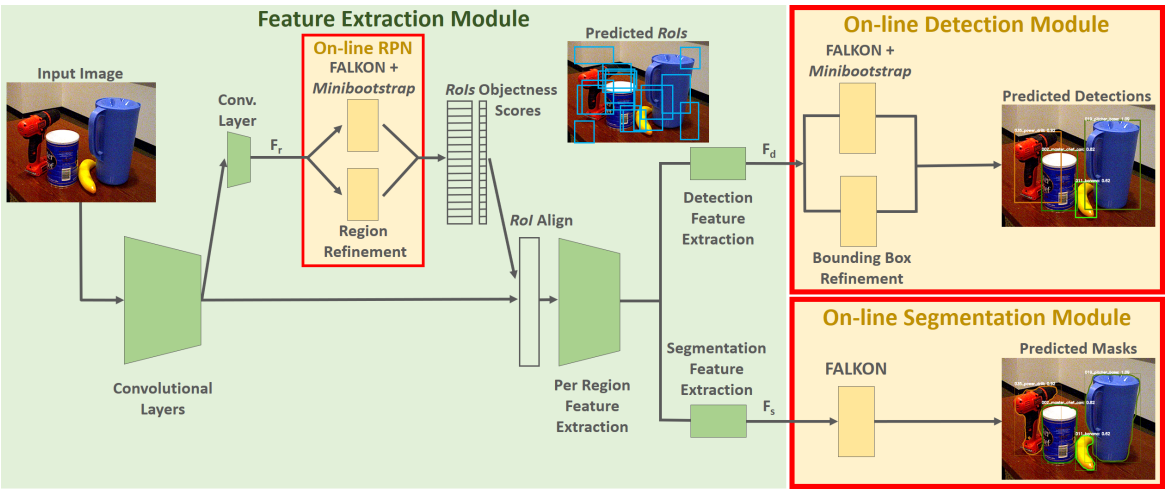


Figure 11.1 **Overview of the proposed pipeline**. The *Feature Extraction Module* is composed of Mask R-CNN's first layers trained off-line on the FEATURE-TASK. The three sets of features for *(i)* region proposal (F_r), *(ii)* object detection (F_d) and *(iii)* instance segmentation (F_s) are fed to *(i)* the *On-line RPN*, *(ii)* the *On-line Detection Module* and *(iii)* the *On-line Segmentation Module*. At inference time, we substitute the final layers of the Mask R-CNN's RPN with the *On-line RPN* trained on the TARGET-TASK and, as in Mask R-CNN, the output of the *On-line Detection Module* is fed as input to the *RoI Align* to compute the objects masks within the proposed bounding boxes.

segmentation application. As we show in Sec. 11.8, we deploy the proposed pipeline on the iCub humanoid robot, adapting it to an incremental setting, where the target classes are not known a-priori.

In this work, we propose a pipeline and a training protocol for instance segmentation which is specifically designed to reduce training time, while preserving performance as much as possible. This approach is based on Mask R-CNN [66], in which the final layers of the RPN and of the detection and segmentation branches have been replaced with "shallow" classifiers based on a fast Kernel-based method optimized for large scale problems [169, 120]. The backbone of the network is trained off-line, while the Kernel-based classifiers are adapted on-line. In this paper, we build on our previous work [21], in that we include the adaptation of the region proposal network and a novel training protocol which allows to further reduce the training time. This makes the pipeline suitable for on-line implementation.

11.3 Methods

The proposed hybrid pipeline allows to quickly learn to predict masks of previously unseen objects (TARGET-TASK). We rely on convolutional weights pre-trained on a different set of

objects (FEATURE-TASK) and we rapidly adapt three modules for region proposal, object detection and mask prediction on the new task. This allows to achieve on-line adaptation on novel objects and visual scenarios.

11.3.1 Overview of the Pipeline

The proposed pipeline is composed of four modules, which are depicted in Fig. 11.1. They are:

- **Feature Extraction Module.** This is composed of the first layers of Mask R-CNN, which has been pre-trained off-line on the FEATURE-TASK. We use it to extract the convolutional features to train the three on-line modules on the TARGET-TASK. In particular, we use it to extract the features F_r, F_d and F_s from the penultimate layers of the RPN, and of the detection and segmentation branches, respectively.
- **On-line RPN.** This replaces the last layers of the Mask R-CNN's RPN to predict a set of regions that likely contain an object in an image, given a feature map F_r. We describe the training procedure in Sec. 11.3.2.
- **On-line Detection Module.** This is composed of classifiers and regressors that, starting from a set of feature tensors F_d, classify and refine the regions proposed by the *On-line RPN*. See Sec. 11.3.2 for the description of the training procedure.
- **On-line Segmentation Module.** Given a feature map F_s, this module predicts the masks of the objects within the detections proposed by the *On-line Detection Module*. We describe the training procedure in Sec. 11.3.3.

In the three on-line modules described above, we use FALKON for classification. This is a Kernel-based method optimized for large-scale problems [169]. In particular, we use the implementation available in [120].

11.3.2 Bounding Box Learning

The prediction of region proposal candidates and object detection are problems that share similarities. In both cases, input bounding boxes are classified and then refined, starting from associated feature tensors as input. In our pipeline, these problems are carried out by the *On-line RPN* and the *On-line Detection Module*, which are implemented by N_c FALKON binary classifiers and $4N_c$ Regularized Least Squares (RLS) regressors [59]. Specifically, N_c

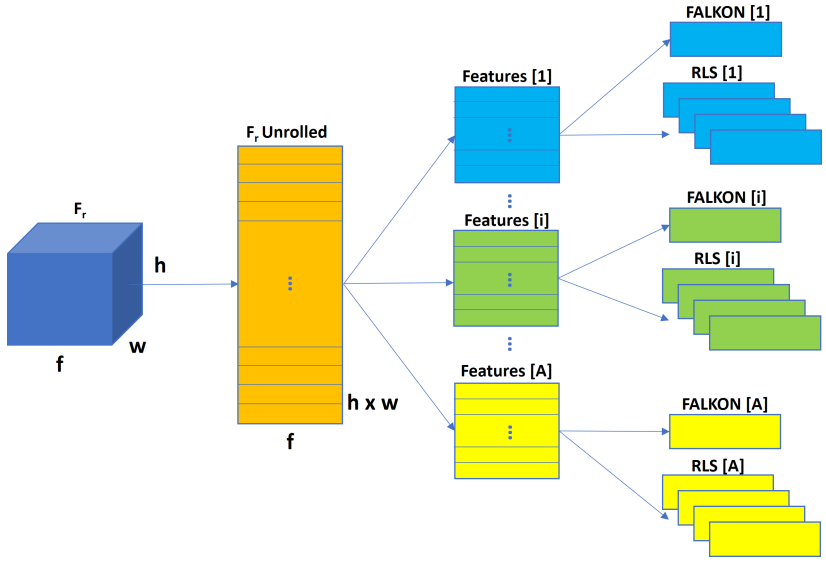


Figure 11.2 **On-line RPN.** Given the feature map F_r, this is unrolled into $h \times w$ tensors of features of size f (F_r *Unrolled*). A subset of these features is chosen to train a FALKON classifier and four RLS regressors for each anchor.

represents: *(i)* the number of anchors for the *On-line RPN* (see the following paragraphs) or *(ii)* the number of semantic classes of the TARGET-TASK for the *On-line Detection Module*. In both the on-line modules, we tackle the well known problem of foreground-background imbalance of training samples in object detection [103] by adopting the *Minibootstrap* strategy proposed in [114, 115] for FALKON training. The *Minibootstrap* is an approximated procedure for hard negatives mining [187, 59] that allows to iteratively select a subset of hard negative samples to balance the training sets associated to each of the N_c classes. We report the pseudo-code of the *Minibootstrap* procedure in App. E in Sec. 11.14. The RLS regressors for boxes refinement, instead, are trained on a set of positive (foreground) instances.

In the *On-line RPN*, the classifiers are trained on a binary task to discriminate anchors representing the background from those representing *RoIs*, i.e., containing an instance of any of the TARGET-TASK classes. An anchor [165] is a bounding box of a predefined size and aspect ratio centered on an image pixel. For each pixel, there are a fixed number of anchors of different form factors and one classifier is instantiated for each of them. In the *On-line Detection*, instead, a binary classifier is instantiated for each class. Each classifier is trained to discriminate regions proposed by the *On-line RPN* depicting an object of its class from other classes or background.

On-line RPN. In Mask R-CNN's RPN, the classification is performed on a set of anchors A (i.e., $N_c = A$). Given the input feature map computed by the backbone of height h, width

w and with f channels, the RPN firstly processes it with a convolutional layer to obtain a feature map F_r of the same size ($h \times w \times f$). Then, F_r is processed by two convolutional layers. One is composed of A convolutional kernels which compute the objectness scores of each considered anchor. This layer computes an output tensor of size $h \times w \times A$, in which the ija^{th} element is the objectness score of the a^{th} anchor in the location (i, j). The other output layer, instead, is composed of $4A$ kernels for the refinement of such bounding boxes. It computes $h \times w \times 4A$ values for the refinement of the regions associated to the anchors at each location (i, j). Both the output convolutional kernels have size 1 and stride 1.

As shown in Fig. 11.2, we replace the A convolutional kernels for the computation of the objectness scores with A FALKON binary classifiers and we train them with the *Miniboot-strap*. We use the $h \times w$ tensors of size f resulting from the flattening of the feature maps F_r as training features. The considered positive features are those associated to a specific location of an anchor whose *Intersection over Union* (*IoU*) with at least a ground-truth bounding box is greater than 0.7[2] (in case there are no anchors overlapping the ground-truth bounding boxes with $IoU > 0.7$, the ones with the highest *IoU* are chosen as positives). The feature tensors for the background, instead, are those whose *IoU* with the ground-truths is smaller than 0.3. Similarly, we replace the $4A$ convolutional kernels for the refinement of the proposed regions with $4A$ RLS regressors (4 RLS for each anchor). We consider as training samples for the 4 regressors associated to each anchor a set of features chosen as the positive samples for the FALKON classifiers, but setting the *IoU* threshold to 0.6[3].

On-line object detection. We train the *On-line Detection Module* with the strategy illustrated above, considering the N classes of the TARGET-TASK (i.e., $N_c = N$). As training samples, we consider the tensors of features produced by the penultimate layer of the Mask R-CNN's detection branch (F_d) associated to each *RoI* proposed by the region proposal method. In particular, we consider as positive samples for the n^{th} FALKON classifier, those *RoIs* with $IoU > 0.6$[4] with a ground-truth box of an instance of class n ($n \in [1, \ldots, N]$). The same positive samples are also used for training the n^{th} RLS regressor[3]. Then, as negative samples, we consider the *RoIs* with $IoU < 0.3$[5] with the ground-truths of class n.

[2]For the *On-line RPN*, we set the positive and negative thresholds for the classifiers as in Mask R-CNN's RPN [66].

[3]We set the value of the *IoU* threshold for the RLS regressors as in R-CNN [59].

[4]We consider as positive samples for the classifiers in the *On-line Detection Module* the training features for region refinement as in [66].

[5]For the classifiers in the *On-line Detection Module* we define the negative samples as in [59].

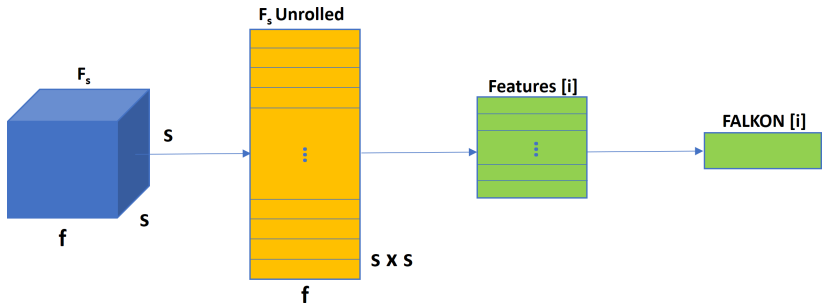


Figure 11.3 **On-line Segmentation.** Given the feature map F_s associated to a *RoI* of class i, this is unrolled into $s \times s$ tensors of features of size f (F_s *Unrolled*) from which positive and negative features are sampled to train the i^{th} FALKON per-pixel classifier. Note that this procedure is performed for each *RoI* of the N classes.

11.3.3 On-line Segmentation

In Mask R-CNN, in the configuration that does not use the *Feature Pyramid Network* (FPN) [104] in the backbone, the segmentation branch is a shallow fully convolutional network (FCN) composed of two layers that takes as input a feature map of size $s \times s \times f$ associated to each *RoI*. The first layer processes the input feature map into another feature map F_s of the same size. The last convolutional layer, instead, has N channels (one for each class of the TARGET-TASK) and kernel size 1 and stride 1. Therefore, the output of the Mask R-CNN's segmentation branch is a tensor of size $s \times s \times N$, where the ijn^{th} value of such tensor represents the confidence that the pixel in the location (i, j) of the *RoI* corresponds to the n^{th} class.

For the fast learning of the *On-line Segmentation Module*, we rely on the first layer of the segmentation branch for feature extraction, but we substitute the last convolutional layer for per-pixel prediction with N FALKON binary classifiers. To train such classifiers, we consider the ground-truth boxes of each training image, we compute the feature map of size $s \times s \times f$ for each of them and we flatten each of such feature maps into $s \times s$ tensors of size f, as shown in Fig. 11.3. Among these tensors, we consider as positive samples for the n^{th} classifier the features associated to the pixels in the ground-truth masks of class n. Instead, we consider as negative samples the features associated to the background pixels contained in ground-truth bounding boxes of class n. Given the great amount of training samples, to speed-up the training procedure, we randomly subsample both the positive and the negative features by a factor r. According to the analysis provided in [21], we set r to 0.3.

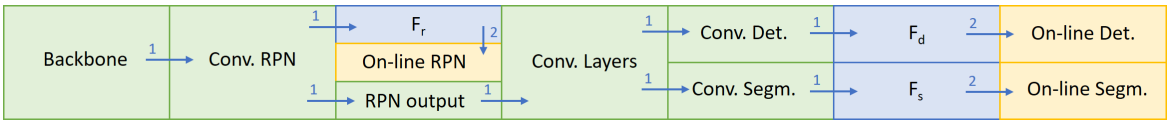


Figure 11.4 **Ours** training protocol. We rely on the feature extraction layers of Mask R-CNN pre-trained on the FEATURE-TASK to simultaneously extract F_r, F_d and F_s. We then use these features to train the three on-line modules on the TARGET-TASK. The values on the arrows correspond to the training steps in Sec. 11.3.4.

11.3.4 Training Protocol

In this work, we propose a training protocol that allows to quickly update the *On-line RPN*, the *On-line Detection Module* and the *On-line Segmentation Module*. The proposed method (referred to as **Ours**) starts with the weights of Mask R-CNN pre-trained on the FEATURE-TASK and adapts the on-line modules on the TARGET-TASK. This is composed of the two steps depicted in Fig. 11.4:

1. *Feature extraction.* This is done with a forward pass of the pre-trained Mask R-CNN feature extractor to compute F_r, F_d and F_s.

2. *On-line training.* The set of features F_r, F_d and F_s are used to respectively train *(i)* the *On-line RPN*, *(ii)* the *On-line Detection Module* and *(iii)* the *On-line Segmentation Module* on the TARGET-TASK.

The training features fed to the *On-line Detection Module* are those associated to the regions proposed by the Mask R-CNN's RPN pre-trained on the FEATURE-TASK. These regions are different (and therefore sub-optimal) with respect to the ones that would be proposed by a region proposal method that has been adapted on the TARGET-TASK. Training the *On-line Detection Module* using features extracted from the *On-line RPN* after its adaptation is possible. This, however, would require two feature extraction steps (one for training the *On-line RPN* and the other to train the *On-line Detection Module* and the *On-line Segmentation Module*), which is computationally expensive. For this reason, we consider the *On-line Detection Module* obtained with **Ours** as an approximation of the one that would be provided by the serial training (see Sec. 11.6.2). Instead, the adaptation of the *On-line Segmentation Module* is not affected by this approximation, since we sample the training features for this module from the ground-truth bounding boxes.

While this approximation is a key component of the proposed training protocol, at inference time features fed to the *On-line Detection Module* and to the *On-line Segmentation*

Module are those associated to the regions proposed by the *On-line RPN* trained on the TARGET-TASK, as depicted in Fig. 11.1.

In this work, we show that a single feature extraction step can be performed paying a small price in terms of accuracy (see Sec. 11.6.2), allowing to further improve the training in the on-line implementation (see Sec. 11.7 and Sec. 11.8).

11.4 Experimental Setup

In this section, we report on the experimental settings that we employ for validating the proposed approach. We first evaluate our approach in an off-line setting (Sec. 11.5, 11.6 and 11.7), analyzing performance on two different robotics datasets. Then, we validate it in a real robotic application (Sec. 11.8), i.e., in an on-line setting. Therefore, in this section, we firstly report on the off-line experimental setup (Sec. 11.4.1) and the datasets (Sec. 11.4.2) that we use in our experiments. Then, in Sec. 11.4.3, we describe the settings considered for the deployment in a real robotic scenario.

11.4.1 Off-line Experiments

For our experiments, we compare the proposed method, **Ours**, with two Mask R-CNN [66] baselines. In particular, we consider:

- **Mask R-CNN (output layers)**: starting from the Mask R-CNN weights pre-trained on the FEATURE-TASK, we re-initialize the output layers of the RPN and of the detection and segmentation branches, and we fine-tune them on the TARGET-TASK, freezing all the other weights of the Mask R-CNN network.
- **Mask R-CNN (full)**: we use the weights of Mask R-CNN pre-trained on the FEATURE-TASK as a warm-restart to train Mask R-CNN on the TARGET-TASK.

Specifically, we rely on Mask R-CNN [66], using ResNet-50 [67] as backbone, for the feature extraction of **Ours** and for the baselines. In App. A in Sec. 11.10, we report a summary of the training protocols used in this work.

In all cases, we choose hyper-parameters providing the highest performance on a validation set. Specifically, for **Ours** we cross-validate the standard deviation of FALKON's Gaussian kernels (namely, σ) and FALKON's regularization parameter (namely, λ) for the *On-line RPN*, the *On-line Detection Module* and the *On-line Segmentation Module*. Regarding the baselines, instead, for the experiments in Sec. 11.5 and 11.6, we train **Mask R-CNN**

(output layers) and **Mask R-CNN (full)** for the number of epochs that provides the highest segmentation accuracy on the validation set.

For **Ours**, we set the number of Nyström centers M of the FALKON classifiers composing the *On-line RPN*, the *On-line Detection Module* and the *On-line Segmentation Module* to 1000, 1000 and 500, respectively. Moreover, to train both the *On-line RPN* and the *On-line Detection Module*, we set to 2000 the size of the batches, BS, considered in the *Minibootstrap*.

Evaluation metrics. We consider the *mean Average Precision (mAP)* as defined in [50] for both object detection and segmentation. Specifically, the accuracy of the predicted bounding boxes will be referred to as **mAP bbox(%)** and the accuracy of the mask instances as **mAP segm(%)**. For each of them, we consider as positive matches the bounding boxes and the masks whose IoU with the ground-truths is greater or equal to a threshold. In our experiments we consider two different thresholds to evaluate different levels of accuracy, namely, 50% (**mAP50**) and 70% (**mAP70**). In App. A in Sec. 11.10 we overview the acronyms considered for *mAP* computation. We also evaluate the methods on the time required for training[6]. For the Mask R-CNN baselines, the training time is the time needed for their optimization via backpropagation and stochastic gradient descent. As regards **Ours**, instead, except where differently specified, it is the time necessary for extracting the features and training the on-line modules. For each experiment, we run three trials for each method. We report results in terms of average and standard deviation of the accuracy and of average training time.

11.4.2 Datasets

In our experiments we consider three datasets. Specifically, we use MS COCO [105] as FEATURE-TASK and the two datasets YCB-Video [212] and HO-3D [65] as TARGET-TASKs to validate our approach. We opted to validate our system on these datasets, which are composed of streams of frames in tabletop and hand-held settings, to be close to our target application. These datasets are usually considered for the task of 6D object pose estimation, however, they are annotated also with object masks. Specifically:

- **MS COCO** [105] is a general-purpose dataset, which contains 80 objects categories, for object detection and segmentation.
- **YCB-Video** [212] is a dataset for 6D pose estimation in which 21 objects from the YCB [16] dataset are arranged in cluttered tabletop scenarios, therefore presenting strong occlusions. It is composed of video sequences where the tabletop scenes are

[6]All the *off-line experiments* have been performed on a machine equipped with Intel(R) Xeon(R) W-2295 CPU @ 3.00GHz, and a single NVIDIA Quadro RTX 6000.

recorded under different viewpoints. We use as training images a set of 11320 images, obtained by extracting one image every ten from the total 80 training video sequences. As test set, instead, we consider the 2949 *keyframe* [212] images chosen from the remaining 12 sequences. For hyper-parameters cross-validation, we randomly select a subset of 1000 images from the 12 test sequences, excluding the *keyframe* set.

- **HO-3D** [65] is a dataset for hand-object pose estimation, in which objects are a subset of the ones in **YCB-Video**. It is composed of video sequences, in which a moving hand-held object is shown to a fixed camera. For choosing the training and test sets, we split the available annotated sequences in HO-3D[7], such that, we gather one and at most four sequences for testing and training, respectively. In particular, we use 20156 images as training set, which result from the selection of one every two images from 34 sequences. Instead, we consider as test set 2020 images chosen one every five frames taken from other 9 sequences. For hyper-parameters cross-validation, we consider 2160 frames chosen one every five images, from a subset of 9 sequences taken from the training set (see App. B in Sec. 11.11 for further details).

11.4.3 Robotic Setup

We deploy the proposed pipeline for on-line instance segmentation on the humanoid robot iCub[8] [123]. It is equipped with a *Intel(R) RealSense D415* on a headset for the acquisition of RGB images and depth information. We rely on the YARP [122] middleware for the implementation and the communication between the different modules (see Sec. 11.8). With the exception of the proposed one, we rely on publicly available modules[9]. We set all the training hyper-parameters as described in Sec. 11.4.1.

11.5 Results

In this section, we benchmark the proposed approach on YCB-Video (Sec. 11.5.1) and HO-3D (Sec. 11.5.2).

[7]Note that, in HO-3D, the annotations for instance segmentation are not provided for the test set. Therefore, we extract training and test sequences from the original HO-3D training set.

[8]We run the module with the proposed method on a machine equipped with Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, and a single NVIDIA RTX 2080 Ti.

[9]https://github.com/robotology

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (full)	89.66 ± 0.47	91.26 ± 0.56	84.67 ± 0.81	80.26 ± 0.59	1h 35m 42s
Mask R-CNN (output layers)	84.51 ± 0.40	81.70 ± 0.17	75.81 ± 0.30	70.46 ± 0.24	2h 57m 12s
Ours	83.66 ± 0.84	83.06 ± 0.92	72.97 ± 1.02	68.11 ± 0.29	13m 53s

Table 11.1 Benchmark on the YCB-Video dataset. We compare the proposed approach **Ours** to the baseline **Mask R-CNN (output layers)** and to the upper bound **Mask R-CNN (full)**.

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (full)	92.21 ± 0.88	90.70 ± 0.17	86.73 ± 0.71	77.25 ± 0.62	38m 38s
Mask R-CNN (output layers)	88.05 ± 0.32	86.11 ± 0.29	74.75 ± 0.19	65.04 ± 0.62	1h 50m 33s
Ours	83.63 ± 1.64	84.50 ± 1.63	63.33 ± 1.65	61.54 ± 0.33	16m 51s

Table 11.2 Benchmark on the HO-3D dataset. We report on the performance obtained with **Ours** and we compare it to **Mask R-CNN (output layers)** and **Mask R-CNN (full)** for the analysis in Sec. 11.5.2.

11.5.1 Benchmark on YCB-Video

We consider the 21 objects from YCB-Video as TARGET-TASK and we compare the performance of **Ours** against the baseline **Mask R-CNN (output layers)**. We also report the performance of **Mask R-CNN (full)**, which can be considered as an upper-bound because, differently from the proposed method, it updates both the feature extraction layers and the output layers (i.e., the backbone, the RPN and the detection and segmentation branches) fitting more the visual domain of the TARGET-TASK. In **Ours**, we empirically set the number of batches in the *Minibootstrap* to 10, to achieve the best training time/accuracy trade-off (see Fig. 11.6 for details).

Results in Tab. 11.1 show that **Ours** achieves similar performance to **Mask R-CNN (output layers)** in a fraction ($\sim$**12.8**$\times$ smaller) of the training time. In comparison with the upper bound, **Ours** is not as accurate as **Mask R-CNN (full)** ($\sim$**9.0**% less precise if we consider the **mAP50 segm(%)**), but is trained $\sim$**6.9**$\times$ faster.

11.5.2 Benchmark on HO-3D

We evaluate the proposed approach on the HO-3D dataset. As in Sec. 11.5.1, we compare **Ours** with **Mask R-CNN (output layers)** and we consider **Mask R-CNN (full)** as upper bound. For this experiment, we empirically set the number of the *Minibootstrap* batches of the *On-line RPN* and of the *On-line Detection Module* in **Ours** to 12 and we report the obtained results in Tab. 11.2.

Similarly to the experiment on YCB-Video, **Ours** can be trained $\sim$**2.3**$\times$ and $\sim$**6.6**$\times$ faster than **Mask R-CNN (full)** and **Mask R-CNN (output layers)**, respectively. Models obtained with **Ours** are slightly less precise than those provided by **Mask R-CNN (output**

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (full)	89.66 ± 0.47	91.26 ± 0.56	84.67 ± 0.81	80.26 ± 0.59	1h 35m 42s
O-OS	76.15 ± 0.31	74.44 ± 0.11	68.06 ± 0.34	63.90 ± 0.36	11m 14s
Ours	83.66 ± 0.84	83.06 ± 0.92	72.97 ± 1.02	68.11 ± 0.29	13m 53s

Table 11.3 Comparison between **Ours**, **Mask R-CNN (full)** and **O-OS** trained on YCB-Video. For **O-OS**, we reproduce the experiment of Tab. I in [21], but we run the experiment three times (reporting mean and standard deviation of the obtained results) on the hardware used for this work and we set the training hyper-parameters as described in Sec. 11.6.1.

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (full)	92.21 ± 0.88	90.70 ± 0.17	86.73 ± 0.71	77.25 ± 0.62	38m 38s
O-OS	75.27 ± 0.26	77.42 ± 0.45	57.89 ± 0.24	57.86 ± 0.21	13m 31s
Ours	83.63 ± 1.64	84.50 ± 1.63	63.33 ± 1.65	61.54 ± 0.33	16m 51s

Table 11.4 We report on the performance obtained on HO-3D with **Ours** and we compare it to **Mask R-CNN (full)** and **O-OS** for the analysis in Sec. 11.6.1.

layers) for the task of instance segmentation, while they are ∼**15.3**% less accurate if we consider the **mAP70 bbox(%)**. We will show in Sec. 11.6.2 that this gap can be recovered with a different training protocol (**Ours Serial** in Sec. 11.6.2). However, **Ours**, achieves the best training time with an accuracy that is close to the state-of-the-art.

11.6 Fast Region Proposal Adaptation

In this section, we investigate the impact of region proposal adaptation on the overall performance. In particular, in Sec. 11.6.1, we show that, with respect to our previous work [21], updating the RPN provides a significant gain in accuracy, maintaining a comparable training time. Then, in Sec. 11.6.2 we analyze the speed/accuracy trade-off achieved with the proposed approximated training protocol.

11.6.1 Is Region Proposal Adaptation Key to Performance?

The adaptation of the region proposal on a new task provides a significant gain in accuracy for object detection (in this paper we report some evidence while additional experiments can be found in [20]). In particular, adapting the region proposal is especially effective when FEATURE-TASK and TARGET-TASK present a significant domain shift (which represents a common scenario in robotics). In this section, we show that better region proposals improves also the downstream mask estimation.

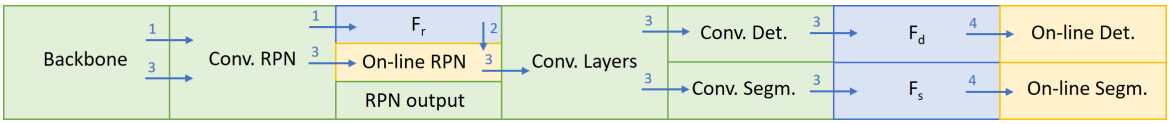


Figure 11.5 **Ours Serial** training protocol. We rely on the feature extraction layers of Mask R-CNN pre-trained on the FEATURE-TASK to extract F_r and we train the *On-line RPN* on the TARGET-TASK. Then, we rely on the feature extraction layers of Mask R-CNN and on the *On-line RPN* trained on the TARGET-TASK to extract F_d and F_s. Finally, we train the *On-line Detection Module* and the *On-line Segmentation Module* on the TARGET-TASK. The values on the arrows correspond to the training steps in Sec. 11.6.2.

For testing performance under domain shift, we consider as FEATURE-TASK the categorization task of the general-purpose dataset MS COCO. Instead, we consider as TARGET-TASKs the identification tasks of the YCB-Video and HO-3D datasets, which depict tabletop and in-hand scenarios, respectively.

We consider **Mask R-CNN (full)** as the upper bound of the experiment, since it updates the entire network on the new task. We compare **Ours** with the method proposed in [21] (Sec. III), namely **O-OS**[10], in which the RPN remains constant during training on the TARGET-TASK. For a fair comparison, we set **O-OS** training hyper-parameters according to Sec. 11.4.1 (i.e., changing the number of Nyström centers of FALKON in the *On-line Detection Module* and in the *On-line Segmentation Module* with respect to [21]).

Results in Tab. 11.3 and in Tab. 11.4 show that, as expected, there is an accuracy gap between **Mask R-CNN (full)** and all the other considered methods (**Ours** and **O-OS**). However, notably, the adaptation of the region proposal on the TARGET-TASK in **Ours** allows to significantly reduce the accuracy gap between **Mask R-CNN (full)** and **O-OS**. Moreover, **Ours** outperforms the accuracy of **O-OS** with a comparable training time. For instance, in the HO-3D experiment (see Tab. 11.4), the segmentation **mAP50** obtained with **Ours** is, on average, $\sim$**7.1** points greater than **O-OS**, with a difference in training time of only *3m 20s*.

11.6.2 Approximated On-line Training: Speed/Accuracy Trade-off

In this section, we evaluate the impact of the approximation in **Ours**. To do this, we compare it with a different training protocol (referred to as **Ours Serial**). This relies on the same on-line modules as **Ours**. However, **Ours Serial** performs two steps of feature extraction, one to train the *On-line RPN* and the other for the *On-line Detection Module* and the *On-line*

[10] In [21], **O-OS** is referred to as **Ours**.

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (output layers)	84.51 ± 0.40	81.70 ± 0.17	75.81 ± 0.30	70.46 ± 0.24	2h 57m 12s
Ours Serial	83.97 ± 0.59	83.00 ± 0.78	75.06 ± 0.88	69.12 ± 0.56	24m 42s
Ours	83.66 ± 0.84	83.06 ± 0.92	72.97 ± 1.02	68.11 ± 0.29	13m 53s

Table 11.5 Comparison between the proposed approach **Ours**, the baseline **Mask R-CNN (output layers)** and **Ours Serial** trained on YCB-Video. Refer to Sec. 11.6.2 for further details.

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (output layers)	88.05 ± 0.32	86.11 ± 0.29	74.75 ± 0.19	65.04 ± 0.62	1h 50m 33s
Ours Serial	88.70 ± 0.43	87.87 ± 0.37	71.65 ± 0.93	64.76 ± 0.70	37m 18s
Ours	83.63 ± 1.64	84.50 ± 1.63	63.33 ± 1.65	61.54 ± 0.33	16m 51s

Table 11.6 We report on the performance obtained on HO-3D with **Ours** and we compare it to **Mask R-CNN (output layers)** and **Ours Serial** for the analysis in Sec. 11.6.2.

Segmentation Module. This latter is done after region proposal adaptation and allows to use better *RoIs* to train the module for on-line detection, improving the overall performance of the pipeline. In details, **Ours Serial** is composed of the four steps depicted in Fig. 11.5:

1. Feature extraction for region proposal. This is done to extract F_r (see Sec. 11.3.1) on the images of the TARGET-TASK.

2. These features are then used to train the *On-line RPN* on the TARGET-TASK, as described in Sec. 11.3.2.

3. The new *On-line RPN* is used to extract more precise regions and the corresponding features for detection and segmentation (respectively, F_d and F_s).

4. F_d and F_s are used to train the *On-line Detection Module* and the *On-line Segmentation Module* on the TARGET-TASK, as described in Sec. 11.3.2 and Sec. 11.3.3, respectively.

We evaluate **Ours** and **Ours Serial** in the same setting used for previous experiments (Sec. 11.5 and Sec. 11.6.1). In **Ours Serial**, we set the Nyström centers of the FALKON classifiers and the batch size *BS* considered in the *Minibootstrap* to train the *On-line RPN* and the *On-line Detection Module* as described in Sec. 11.4.1. Moreover, we empirically set the number of *Minibootstrap* iterations n_B to 8 and 7 in the experiments on YCB-Video and HO-3D, respectively.

We report results in Tab. 11.5 (YCB-Video) and in Tab. 11.6 (HO-3D). Specifically, the first one shows that the accuracy of **Ours** in the YCB-Video experiment is comparable to the one of **Ours Serial**, demonstrating that the approximated training procedure substantially

does not affect performance in this case. Instead, in the HO-3D experiment (see Tab. 11.6), **Ours** is slightly less precise than **Ours Serial** for the task of instance segmentation, while being $\sim$**11.6**% less accurate if we consider the **mAP70 bbox(%)**. However, **Ours** is trained $\sim$**1.8**$\times$ and $\sim$**2.2**$\times$ faster than **Ours Serial** in the YCB-Video and in the HO-3D experiments, respectively.

Still, with respect to **Mask R-CNN (output layers)**, **Ours Serial** achieves comparable performance, but with training time that is much shorter. However, the approximated training protocol proposed in this paper allows further optimization which is discussed in the next section.

11.7 Stream-based Instance Segmentation

We now consider a robotic application, in which the robot is tasked to learn new objects on-line, while automatically acquiring training samples. In this case, training data arrive continuously in stream, and the robot is forced to either use them immediately or store them for later use. We investigate to what extent it is possible to reduce the training time and how this affects segmentation performance.

Because data acquisition takes a considerable amount of time, there is the opportunity to perform, in parallel, some of the processing required for training. In the proposed pipeline, for example, the training protocol **Ours** has been designed to separate feature extraction and the training of the Kernel-based components. In this case, feature extraction can be performed while images and ground-truth labels are received by the robot. In this section, we investigate to what extent this possibility can be exploited also with the conventional Mask R-CNN architecture.

We compare the proposed **Ours** with three different Mask R-CNN baselines. Specifically, we consider **Mask R-CNN (full)** and two variations of **Mask R-CNN (output layers)** as presented in Sec. 11.4.1.

Because images arrive in a stream, similar views of the same objects are represented in subsequent frames. In App. C in Sec. 11.12 we show that a proper training of **Mask R-CNN (full)** and **Mask R-CNN (output layers)** requires that the images are shuffled randomly. This requires storing all images and waiting until the end of the data acquisition process, before starting the training. We hence consider an additional baseline, **Mask R-CNN (store features)**, in which, similarly to **Mask R-CNN (output layers)**, we fine-tune the output layers of the RPN and of the detection and segmentation branches. In this case, however, we compute and store the backbone feature maps for each input image during data acquisition to

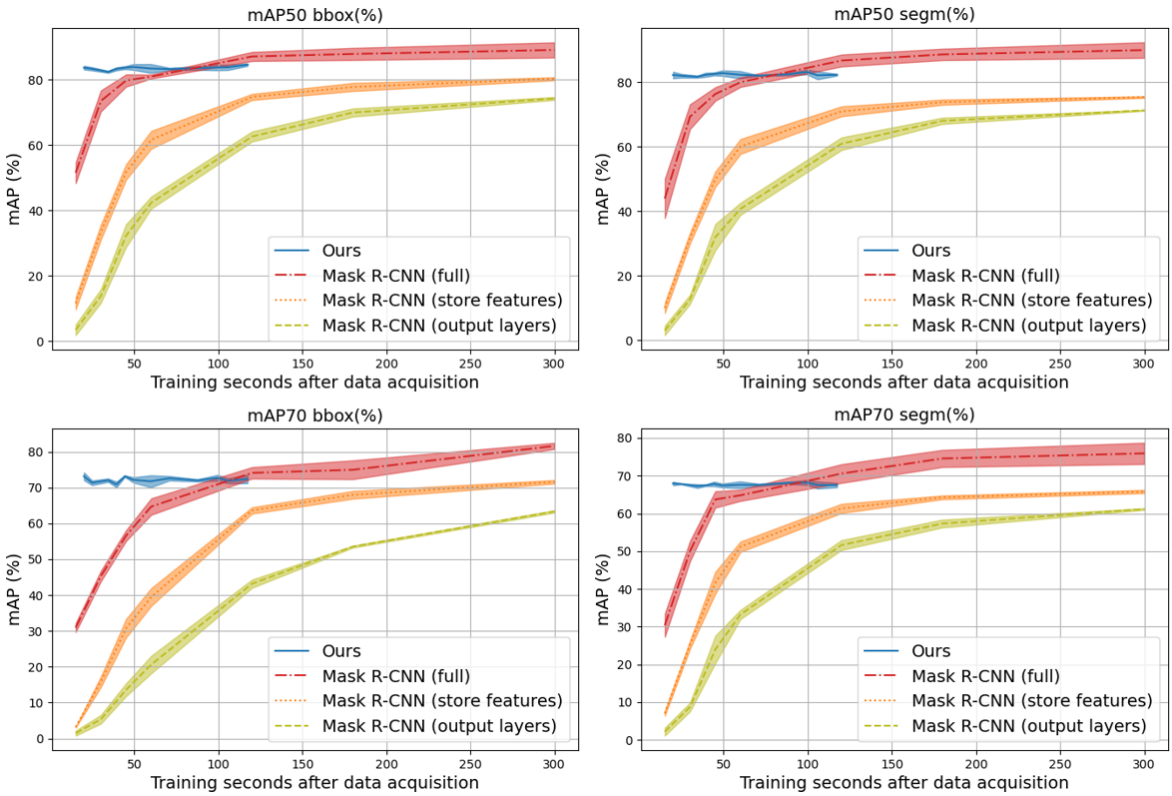


Figure 11.6 Detection and segmentation *mAPs* for increasing number of *Minibootstrap* iterations for **Ours** and for increasing training time of the Mask R-CNN baselines, considering YCB-Video as TARGET-TASK. The plots show the average and the standard deviation of the accuracy obtained over three training sessions with the same parameters.

save time. This can be done because, during the fine-tuning, the weights of the backbone remain unaltered.

Both **Ours** and **Mask R-CNN (store features)** can perform the feature extraction while receiving the stream of images: this allows to further reduce the training time. This is possible because the frame rate for feature extraction in both cases is greater than the frame rate of the stream of incoming data. For instance, with **Ours**, we extract features at **14.7** FPS for YCB-Video while the stream of images that is used for training has a frame rate of 3 FPS (note that the dataset has been collected at 30 FPS, but we use one image over ten to avoid data redundancy). This allows to completely absorb the time for feature extraction in the time for data acquisition for both approaches. Since the time required for the data acquisition is the same for the two compared methods, we remove it from the training time computation, therefore comparing only the processing time that follows this phase. This represents the time to wait for a model to be ready in the target robotic application. As explained above, the

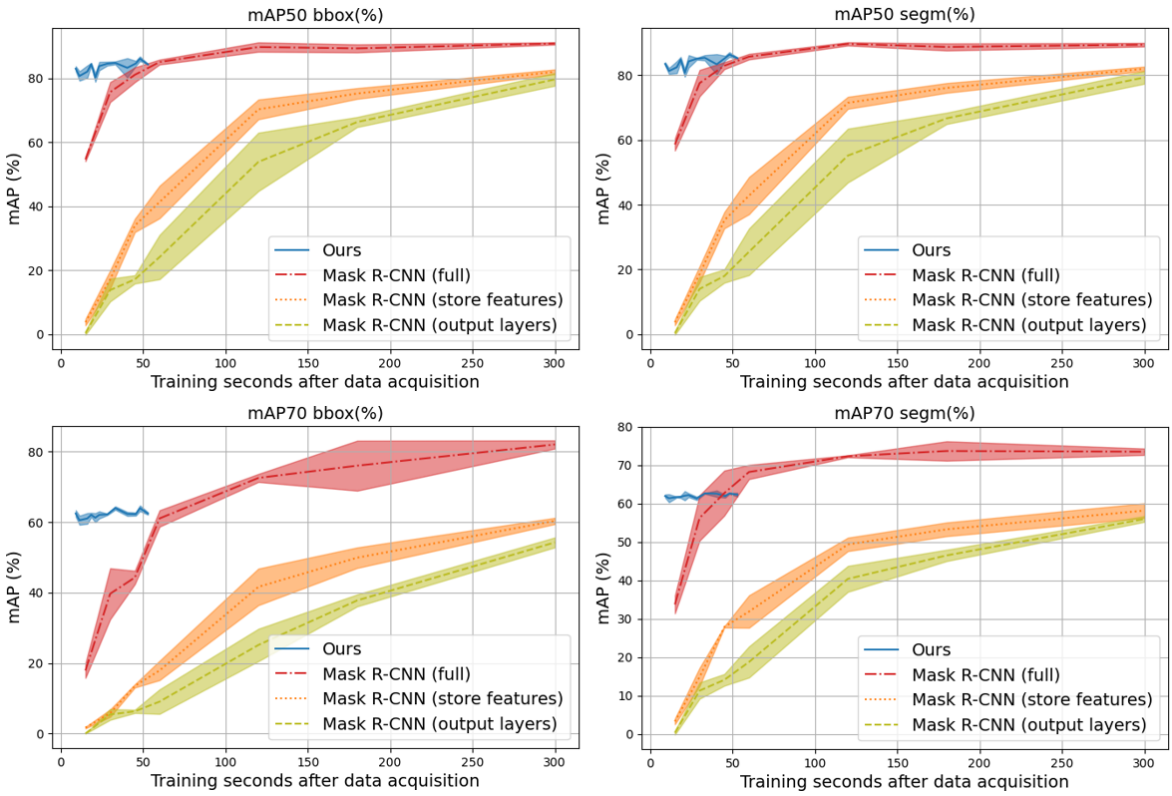


Figure 11.7 We consider HO-3D as TARGET-TASK and we report the average and the standard deviation of the *mAPs* over three training sessions with the same parameters for increasing number of *Minibootstrap* iterations for **Ours**, and for increasing training time of **Mask R-CNN (full)**, **Mask R-CNN (output layers)** and **Mask R-CNN (store features)**.

time required for feature extraction cannot be removed in the case of **Mask R-CNN (full)** and **Mask R-CNN (output layers)**.

We present results for this experiment for YCB-Video and HO-3D in Fig. 11.6 and in Fig. 11.7, respectively. Specifically, we compare the performance of the four considered methods for increasing training time. For the Mask R-CNN baselines, we take the accuracy in different moments of the fine-tuning, while, for **Ours**, we increase the number of iterations of the *Minibootstrap* from 2 to 15. In both Fig. 11.6 and Fig. 11.7, we report in the first row the *mAP* trends at *IoU* 50% while in the second row we report the results for *IoU* 70%, for both detection and segmentation.

As it can be noticed, **Ours** achieves the best accuracy for short training time. For instance, in the YCB-Video experiment, if we consider a training time of $\sim$**20s**, which is the necessary training time if we set the minimum number of *Minibootstrap* iterations n_B=2, **Ours** achieves a **mAP** for instance segmentation of *(i)* $\sim$**82.2** and *(ii)* $\sim$**67.9** for the *IoU* thresholds set to *(i)* 50% and *(ii)* 70%. With a similar optimization time, the Mask R-CNN baselines perform

quite poorly. For example, **Mask R-CNN (full)** (which is the best among the baselines) reaches a **mAP** of *(i)* $\sim$**53.9** and *(ii)* $\sim$**39.8** for the *IoU* thresholds set to *(i)* 50% and *(ii)* 70%.

Moreover, the plots show that, for all the experiments, **Mask R-CNN (output layers)** achieves the worst performance, while **Mask R-CNN (store features)** has a steeper slope. This is due to the fact that this method does not perform the forward pass of the Mask R-CNN backbone for feature extraction. On the contrary, **Mask R-CNN (full)** presents a better trend than **Mask R-CNN (output layers)** and **Mask R-CNN (store features)**. This might be due to the following reasons. Firstly, **Mask R-CNN (full)** optimizes more parameters of the network. While requiring more time for each training step, this allows to speed up the optimization process, requiring less iterations on the dataset to achieve comparable accuracy. Secondly, **Mask R-CNN (full)** performs a warm restart of the the output layers of the RPN, while in the other baselines they are re-initialized from scratch. However, to achieve a similar performance to **Ours**, **Mask R-CNN (full)** requires $\sim$**75s** for the YCB-Video experiment and $\sim$**50s** on HO-3D.

Finally, as it can be noticed, the standard deviations of most of the Mask R-CNN baselines are greater than the ones of **Ours**. This derives from the fact that while **Ours** samples features from all the training images, the Mask R-CNN baselines are optimized only on a subset of them due to time constraints (e.g. in the YCB-Video experiment **Mask R-CNN (full)** processes images at $\sim$**8.0** FPS when trained for **1** minute). Reducing the number of training images increases the variability of the results.

In the video attached as supplementary material to the manuscript[11], we show qualitative results to compare **Ours** to **Mask R-CNN (full)** when trained for the same time.

11.8 Robotic Application

In this section, we describe the pipeline based on the proposed method, that we developed for the iCub [123] robot. We set our application in a teacher-learner scenario, in which the robot learns to segment novel objects shown by a human. The proposed application depicts a similar setting to the experiments on HO-3D showing the effectiveness of the approach to learn new objects also in presence of domain shift.

While in the off-line experiments all the input images and the object instances are fixed beforehand, in the application this information is not known in advance. New objects may appear in the scene and, while learning to segment them, the robot has to keep and integrate

[11] https://youtu.be/eLatoDWY4OI

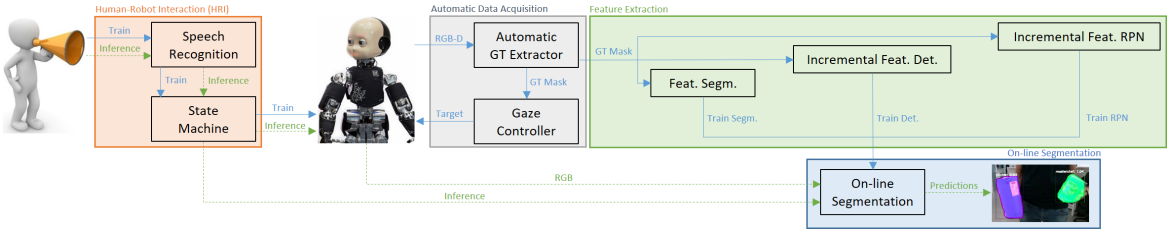


Figure 11.8 **Overview of the proposed robotic pipeline** for on-line instance segmentation. At training time (solid arrows), a human teacher shows a new object to the robot, which automatically acquires the ground-truth annotations exploiting the depth information. Then, it extracts the features to train the on-line modules. At inference time (dashed arrows), the robot employs such modules to predict the masks of the images acquired by the camera.

the knowledge of the classes that are already known. We therefore propose a strategy to process the incoming images and extract corresponding features such that, for each new class, a detection model is trained with **Ours**, integrating the knowledge of old and new objects. This is done by first training new classifiers on the new classes, considering also the information from the objects already known. Then, the classifiers previously trained on the old classes are updated using features of the new classes.

The proposed application consists of four main modules (the blocks depicted in Fig. 11.8). It allows to train and update an instance segmentation model by: *(i)* automatically collecting ground-truth for instance segmentation with an interactive pipeline for incoming training images, *(ii)* extracting corresponding features and aggregating them such that the information of old and new objects are integrated in the *Minibootstrap* and *(iii)* updating the *On-line RPN*, the *On-line Detection Module* and the *On-line Segmentation Module*. In the next paragraphs, we provide further details for each of the main blocks.

Human-Robot Interaction (HRI). This block allows the human to give commands to the robot with a module for speech recognition (*Speech Recognition* in Fig. 11.8), triggering different states of the system. This allows the user to either teach the robot a new object, by presenting and rotating it in front of the camera (*train*) or to perform *inference*, i.e., to segment objects already known in the scene.

Automatic Data Acquisition. When the state of the system is set to *train*, this block extracts a blob of pixels representing the closest object to the robot [146]. This is used as ground-truth annotation for the new object that is presented by the human. This blob is computed by exploiting the depth information to segment the object from the background (*Automatic GT Extractor*). Moreover, in order to enhance the background variability in the training images, the extracted blob is also used by the robot to follow the object with the gaze (*Gaze Controller*). To deal with noise in the depth image, we post-process the masks to ensure

spatio-temporal coherence between consecutive frames. Specifically, we consider as valid ground-truth masks those overlapping over a certain threshold with the ones of previous and subsequent frames.

Feature Extraction. It is used to extract the features to train the three on-line modules. It relies on the ground-truth masks provided by the *Automatic GT Extractor* and on the corresponding image collected by the robot. This block implements the *Feature Extraction Module* as described in Sec. 11.3.1, with some modifications introduced to adapt it to the interactive setting of the demonstration. We describe the major differences in Sec. 11.8.1.

On-line Segmentation. This block is trained with the proposed approach **Ours** (see Sec. 11.3.4) relying on the features extracted by the *Feature Extraction* block. At inference time, it predicts objects masks on a given image. To do this, similarly to **Ours**, it relies on Mask R-CNN pre-trained on the MS COCO dataset for feature extraction and on the proposed on-line modules as described in Sec. 11.3.1.

11.8.1 Incremental Instance Segmentation Learning

When a new object has to be learned, the three on-line modules need to be updated. However, only for the *On-line RPN* and the *On-line Detection Module* specific operations are required to integrate the knowledge of the old classes with the new one and to re-train the two modules with the updated information. Instead, for the *On-line Segmentation Module* only the classifier of the novel class must be trained. This is due to the fact that, for each class, the latter extracts masks labels from the ground-truth bounding boxes for that class, while the other modules use all the images in the dataset (see Sec. 11.3.2 and Sec. 11.3.3 for details).

To this end, in the following paragraphs we describe how we adapt the feature extraction procedures for the *On-line RPN* and the *On-line Detection Module* reported in Sec. 11.3.2 such that past and novel classes can be properly integrated for the training. We refer the reader to App. D in Sec. 11.13 for the probabilistic equivalence between the feature sampling procedures of the two on-line modules described in Sec. 11.3.2 and the ones presented in this section. Please note that, for each module, we consider training features sampled independently from each training image.

For the sake of simplicity, in the following analysis, we consider a single incremental task scenario where a sequence of images representing an instance of a new class is shown to the robot, which is required to learn the new object at the end of the demonstration. Specifically, the robot has already been trained on $N-1$ classes and must learn the N^{th} object. However, Alg. 2 and Alg. 3 describe in detail the general procedures, which are suitable also

if multiple objects must be learned simultaneously. Once the training features for these two modules have been computed with the described approaches, they can be trained with the same procedure described in Sec. 11.3.2, namely with the steps 2 and 3 of the *Minibootstrap* procedure (see App. E in Sec. 11.14).

Positive samples for the N^{th} object, as defined in Sec. 11.3.2 for FALKON classifiers and RLS regressors in the *On-line RPN* and in the *On-line Detection Module*, are taken from the current image stream and are not affected by previous sequences. Therefore, we stick to the method described in Sec. 11.3.2 for their extraction. Instead, for the computation of the negative features we design two different procedures that we describe in the following paragraphs.

On-line RPN. When learning the N^{th} class, we first collect features for *On-line RPN* training for that class. We do this by extracting convolutional features from the images of the associated sequence and sub-sample them as described in Sec. 11.3.2. Then, we need to integrate the extracted features with those from the previous $N-1$ classes for each anchor, such that *(i)* the number and size of *Minibootstrap* batches are kept fixed and *(ii)* the number of negative samples per-image is kept balanced. To this end, we randomly remove a fraction of the samples collected for the *Minibootstrap* batches of the previous $N-1$ classes and substitute them with those for the N^{th} class. More details about this procedure can be found in App. F in Sec. 11.15.

On-line Detection. Similarly to what is done for the *On-line RPN*, when the N^{th} class arrives, we extract convolutional features from the images of the associated sequence and sub-sample them as described in Sec. 11.3.2. Then, the extracted features have to be integrated with those from the previous $N-1$ classes. This is done in a twofold way: *(i)* we create the N^{th} dataset to train a classifier for the new object integrating the extracted features with a subset of the previous $N-1$ ones and *(ii)* we update the $N-1$ datasets for training the previous classes with features from the N^{th}. As for the *On-line RPN*, the number and size of *Minibootstrap* batches are kept fixed and we balance the number of per-image negative samples. More details about this procedure can be found in App. G in Sec. 11.16.

11.8.2 Discussion and Qualitative Results

We design the incremental feature extraction procedures for the *On-line RPN* and the *On-line Detection Module* to be analogous to the ones used in the off-line experiments (batch procedures), such that, training the on-line modules with *Minibootstrap* batches obtained with the former, provides comparable models to the ones obtained with the batch procedures

Figure 11.9 Predictions on test images from the incremental application deployed on the iCub.

Figure 11.10 **Dealing with False Positives**. *Left image*: an unknown object (a *glass*) is misclassified (as a *masterchef*). *Center*: training. The robot is provided with the correct label and a demonstration of the object. *Right*: after training the new object is correctly classified.

(and therefore, comparable accuracy). This is due to the fact that a set of negative samples has the same probability to end up in the *Minibootstrap* batches of a classifier (either of the *On-line RPN* or of the *On-line Detection Module*), either using the batch pipelines as described in Sec. 11.3.2 or the incremental procedures presented in the previous section. This is demonstrated in App. D in Sec. 11.13. Specifically, we demonstrate that the per-image negative selection probabilities with the two procedures are equivalent, because each image is considered independently in both cases. This proves their equivalence.

We qualitatively show the effectiveness of the incremental pipeline by deploying it on the iCub robot. We train ten object instances and we report the results of the inference on some exemplar frames in Fig. 11.9. A video of the complete demonstration, comprising the training of all the considered objects and the inference of the trained models, is attached as supplementary material to the manuscript[11]. In Fig. 11.10, we show how the proposed incremental approach allows to deal with false positive predictions. Key to achieve this is the re-training of the $N-1$ classifiers for previous classes when the N^{th} object arrives. Indeed, integrating data from the N^{th} object when updating the previous $N-1$ allows to strongly reduce the amount of false predictions at inference time.

11.9 Conclusions

The ability of rapidly adapting their visual system to novel tasks is an important requirement for robots operating in dynamic environments. While state-of-the art approaches for visual tasks mainly focus on boosting performance, a relatively small amount of methods are designed to reduce training time. In this perspective, we presented a novel pipeline for fast training of the instance segmentation task. The proposed approach allows to quickly learn to segment novel objects also in presence of domain shifts. We designed a two-stage hybrid pipeline to operate in the typical robotic scenario where streams of data are acquired by the camera of the robot. Indeed, our pipeline allows to shorten the total training time by extracting a set of convolutional features during the data acquisition and to use them in a second step to rapidly train a set of Kernel-based classifiers.

We benchmarked our results on two robotics datasets, namely YCB-Video and HO-3D. On these datasets, we provided an extensive empirical evaluation of the proposed approach to evaluate different training time/accuracy trade-offs, comparing results against previous work [21] and several Mask R-CNN baselines.

Finally, we demonstrated the application of this work on a real humanoid robot. At this aim, we adapted the fast training pipeline for incremental region proposal adaptation and instance segmentation, showing that the robot is able to learn new objects following a short interactive training session with a human teacher.

11.10 Appendix A

Method	Training Protocol	Section
Ours	This is the proposed approach. It is composed of two steps. One for *feature extraction* and one for simultaneous *on-line training* of the proposed methods for region proposal, object detection and mask prediction.	11.3.4
Ours Serial	It is composed of the same modules as **Ours**, but it differs on the training protocol. This is composed of two steps for *feature extraction*, one for the *on-line training* of the proposed method for region proposal and one to *train* the modules for on-line detection and segmentation.	11.6.2
O-OS	This is the approach proposed in [21]. It is composed of two steps. One for *feature extraction* and one for *on-line training* of the modules for object detection and mask prediction.	11.6.1
Mask R-CNN (full)	This protocol relies on Mask R-CNN pre-trained on the FEATURE-TASK as a warm-restart to train Mask R-CNN on the TARGET-TASK.	11.4.1
Mask R-CNN (output layers)	Starting from the Mask R-CNN weights pre-trained on the FEATURE-TASK, it fine-tunes the output layers of the RPN and of the detection and segmentation branches on the TARGET-TASK.	11.4.1
Mask R-CNN (store features)	Similarly to **Mask R-CNN (output layers)**, it fine-tunes the output layers of the RPN and of the detection and segmentation branches. However, since the weights of the backbone remain unaltered, it computes and stores the backbone feature maps for each input image during data acquisition.	11.7

Table 11.7 Training protocols overview.

***Intersection over Union (IoU)* with ground-truth**	**Object Detection**	**Instance Segmentation**
50%	mAP50 bbox(%)	mAP50 segm(%)
70%	mAP70 bbox(%)	mAP70 segm(%)

Table 11.8 Object detection and segmentation metrics taxonomy.

In Tab. 11.7 and in Tab. 11.8, we overview the training protocols and the acronyms for *mAP* evaluation used in this work.

11.11 Appendix B

In this appendix, we report the sequences considered in the experiments on the HO-3D dataset.

- **Training** sequences: *ABF10*, *ABF11*, *ABF12*, *ABF13*, *BB10*, *BB11*, *BB12*, *BB13*, *GPMF10*, *GPMF11*, *GPMF12*, *GPMF13*, *GSF10*, *GSF11*, *GSF12*, *GSF13*, *MC1*, *MC2*, *MC4*, *MC5*, *MDF10*, *MDF11*, *MDF12*, *MDF13*, *ShSu10*, *ShSu12*, *ShSu13*, *ShSu14*, *SM2*, *SM3*, *SM4*, *SMu1*, *SMu40*, *SMu41*.
- **Validation** sequences: *ABF13*, *BB13*, *GPMF13*, *GSF13*, *MC5*, *MDF13*, *ShSu14*, *SM4*, *SMu41*.
- **Test** sequences: *ABF14*, *BB14*, *GPMF14*, *GSF14*, *MC6*, *MDF14*, *SiS1*, *SM5*, *SMu42*.

11.12 Appendix C

Method		mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
1 Epoch	Mask R-CNN (full)	89.21 ± 0.77	90.75 ± 0.80	83.27 ± 0.54	78.83 ± 1.70	35m 8s
	Mask R-CNN (output layers)	82.78 ± 0.50	79.38 ± 0.40	75.04 ± 0.97	68.42 ± 0.55	26m 48s
1 Epoch No Shuffling	Mask R-CNN (full)	46.29 ± 1.26	43.93 ± 0.84	28.66 ± 1.89	32.83 ± 1.37	33m 41s
	Mask R-CNN (output layers)	69.63 ± 0.48	66.03 ± 0.68	38.48 ± 2.88	54.70 ± 0.46	26m 39s
Ours		83.66 ± 0.84	83.06 ± 0.92	72.97 ± 1.02	68.11 ± 0.29	13m 53s

Table 11.9 Comparison between the training of **Ours** and the Mask R-CNN baselines trained for one epoch and for one epoch without shuffling the input images, considering YCB-Video as TARGET-TASK.

Method		mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
1 Epoch	Mask R-CNN (full)	92.21 ± 0.88	90.70 ± 0.17	86.73 ± 0.71	77.25 ± 0.62	38m 38s
	Mask R-CNN (output layers)	86.77 ± 0.68	85.45 ± 0.64	70.57 ± 0.41	63.57 ± 0.43	17m 53s
1 Epoch No Shuffling	Mask R-CNN (full)	6.72 ± 2.35	6.53 ± 2.33	6.24 ± 2.21	6.01 ± 2.18	37m 30s
	Mask R-CNN (output layers)	19.28 ± 2.33	21.01 ± 0.75	11.03 ± 1.70	16.74 ± 1.26	17m 52s
Ours		83.63 ± 1.64	84.50 ± 1.63	63.33 ± 1.65	61.54 ± 0.33	16m 51s

Table 11.10 Comparison between the training of **Ours** and the Mask R-CNN baselines trained for one epoch and for one epoch without shuffling the input images, considering HO-3D as TARGET-TASK.

In the stream-based scenario, data is used as soon as it is received (Sec. 11.7). In this case, the Mask R-CNN baselines can be trained only for one epoch and their weights can be updated only when a new image is received. Therefore, we compare the performance achieved by **Ours** against the Mask R-CNN baselines, training **Mask R-CNN (output layers)** and **Mask R-CNN (full)** for one epoch and without shuffling the input images.

Result are reported in Tab. 11.9 for YCB-Video and in Tab. 11.10 for HO-3D. As it can be noticed, while training the Mask R-CNN baselines for just one epoch allows to achieve a performance similar to the one provided in the benchmarks (Sec. 11.5), shuffling the input images turns out to be crucial. Indeed, in both cases, the accuracy provided by the Mask R-CNN baselines drops when the input images are not shuffled, while **Ours** is not affected by this constraint. Moreover, shuffling is particularly critical for the Mask R-CNN baselines on the HO-3D dataset because training objects are shown subsequently, one by one, as in the target teacher-learner setting (Sec. 11.8). Therefore, in the stream-based scenario, such baselines cannot be trained in practice.

11.13 Appendix D

In this appendix, we prove the probabilistic equivalence of the two sampling procedures which are used in the *Minibootstrap* and in the incremental feature extraction pipelines for both the *On-line RPN* and for the *On-line Detection Module*.

We consider a pool of tensors S_0 and a set of tensors $\hat{S} \subseteq S_0$. We compute the probability of sampling $\hat{S}$ from S_0 with the two following procedures:

- We sample $|\hat{S}|$ tensors from S_0. We refer to the probability that the sampled tensors are equal to $\hat{S}$ as $P(\hat{S} \sim S_0)$.
- We recursively sample m sets from S_0 and we obtain the pools $S_1, .., S_m$ such that $|S_0| \geq |S_1| \geq ... \geq |S_m| \geq |\hat{S}|$. Namely, for each S_i, with i in $0, ..., m-1$, S_{i+1} is a random sample of size $|S_{i+1}|$ of S_i. Finally, we sample $|\hat{S}|$ tensors from S_m. We refer to the probability that the sampled tensors are equal to $\hat{S}$ as $P(\hat{S} \sim S_m)$.

We note that, for the *On-line RPN* and for the *On-line Detection Module*, the pool of tensors S_0 represents the whole set of features associated to an image. $S_1, .., S_m$, instead, represent consecutive sub-samples of S_0 in the incremental feature extraction pipelines. Finally, $\hat{S}$ correspond to the final per-image set of features chosen for training the on-line modules either with the *Minibootstrap* (as presented in 11.3.2) or with the incremental pipelines.

We prove that $P(\hat{S} \sim S_0)$ is equal to $P(\hat{S} \sim S_m)$.

Proof. $P(\hat{S} \sim S_0)$ can be computed as follows:

$$P(\hat{S} \sim S_0) = \frac{1}{\binom{|S_0|}{|\hat{S}|}} \tag{11.1}$$

Instead, due to the law of total probability, we can decompose $P(\hat{S} \sim S_m)$ as follows (note that if $\hat{S}$ is not a subset of S_m, $P(\hat{S} \sim S_m | \hat{S} \not\subseteq S_m) = 0$):

$$P(\hat{S} \sim S_m) = P(\hat{S} \sim S_m | \hat{S} \subseteq S_m) \times P(\hat{S} \subseteq S_m) \tag{11.2}$$

Again, due to the law of total probability, we can decompose $P(\hat{S} \subseteq S_m)$ from equation 11.2 as follows (note that, for each i in $0, ..., m-1$, $P(\hat{S} \subseteq S_i | \hat{S} \not\subseteq S_{i-1}) = 0$):

$$\begin{aligned} P(\hat{S} \subseteq S_m) &= P(\hat{S} \subseteq S_m | \hat{S} \subseteq S_{m-1}) \times P(\hat{S} \subseteq S_{m-1}) \\ &= \prod_{k=m}^{1} P(\hat{S} \subseteq S_k | \hat{S} \subseteq S_{k-1}) \times P(\hat{S} \subseteq S_0) \end{aligned} \tag{11.3}$$

Note that $P(\hat{S} \subseteq S_0) = 1$ by definition (i.e., $\hat{S}$ is always in S_0). Therefore:

$$P(\hat{S} \subseteq S_m) = \prod_{k=m}^{1} P(\hat{S} \subseteq S_k | \hat{S} \subseteq S_{k-1}) \tag{11.4}$$

We note that $\binom{|S_{k-1}|-|\hat{S}|}{|S_k|-|\hat{S}|}$ is the total number of feasible samples s.t. $\hat{S} \subseteq S_k$ given that $\hat{S} \subseteq S_{k-1}$. Namely, we fix $\hat{S}$ in S_k and we compute the number of possible combinations of the remaining $|S_k| - |\hat{S}|$ tensors that can be in S_k sampled from the remaining pool of size $|S_{k-1}| - |\hat{S}|$. Therefore:

$$P(\hat{S} \subseteq S_k | \hat{S} \subseteq S_{k-1}) = \frac{\binom{|S_{k-1}|-|\hat{S}|}{|S_k|-|\hat{S}|}}{\binom{|S_{k-1}|}{|S_k|}} \tag{11.5}$$

We can decompose the component in the product as follows:

$$\frac{\binom{|S_{k-1}|-|\hat{S}|}{|S_k|-|\hat{S}|}}{\binom{|S_{k-1}|}{|S_k|}} = \frac{(|S_{k-1}| - |\hat{S}|)!}{(|S_k| - |\hat{S}|)!} \times \frac{|S_k|!}{|S_{k-1}|!} \tag{11.6}$$

Note that, for each of these elements with $k \neq 1$ and $k \neq m$, if we multiply it by the element at $k-1$ and by the element at $k+1$, all the elements are simplified. Therefore, we can rewrite $P(\hat{S} \sim S_m)$ from equation 11.2 as:

$$
\begin{aligned}
P(\hat{S} \sim S_m) &= P(\hat{S} \sim S_m | \hat{S} \subseteq S_m) \times P(\hat{S} \subseteq S_m) \\
&= \frac{1}{\binom{|S_m|}{|\hat{S}|}} \times \prod_{k=m}^{1} \frac{\binom{|S_{k-1}|-|\hat{S}|}{|S_k|-|\hat{S}|}}{\binom{|S_{k-1}|}{|S_k|}} \\
&= \frac{(|S_m|-|\hat{S}|)! \times |\hat{S}|!}{|S_m|!} \times \frac{|S_m|! \times (|S_0|-|\hat{S}|)!}{(|S_m|-|\hat{S}|)! \times S_0!} \\
&= \frac{|\hat{S}|! \times (|S_0|-|\hat{S}|)!}{|S_0|!} \\
&= \frac{1}{\binom{|S_0|}{|\hat{S}|}}
\end{aligned}
\tag{11.7}
$$

This concludes our proof, since:

$$
P(\hat{S} \sim S_0) = P(\hat{S} \sim S_m) \tag{11.8}
$$

□

11.14 Appendix E

Algorithm 1 Minibootstrap Pseudo-code for the *Minibootstrap* in the off-line experiments. See [115] for further details.

Input: BS, n_B: size and number of *Minibootstrap* batches
I: set of training images
N: number of classes
Output: M: N trained classifiers
***Stage 1**: Feature extraction*
for $n = 1$ to N **do** ▷ *Initialize N empty sets of per class* ***positives*** *and* ***negatives*** *features*
$Pos[n] \leftarrow \emptyset$; $Neg[n] \leftarrow \emptyset$
end for
for $i = 1$ to $|I|$ **do**
for $n = 1$ to N **do**
if ($I[i]$ has positives of class n $Pos_{i,n}$) **then**
$Pos[n] \leftarrow Pos[n] \bigcup Pos_{i,n}$ ▷ ***Add positives*** *for class n from the* i^{th} *image*
end if
$Neg[n] \leftarrow Neg[n] \bigcup \text{Sample}(Neg_{i,n}, \lceil \frac{n_B \times BS}{|I|} \rceil)$ ▷ ***Add negatives*** *for class n from the* i^{th} *image*
end for
end for
***Stage 2**: Shuffling and batches creation*
for $n = 1$ to N **do**
$Neg[n] \leftarrow \text{Sample}(Neg[n], n_B \times BS)$
$Neg[n] \leftarrow \text{Split}(Neg[n], n_B, BS)$ ▷ ***Split*** $Neg[n]$ *in* n_B *batches of size BS*
end for
***Stage 3**: Classifiers training*
for $n = 1$ to N **do**
$F[n] \leftarrow Pos[n] \bigcup Neg[n][1]$
$M[n] \leftarrow \text{TrainClassifier}(F[n])$ ▷ ***Train*** *classifier using the first batch*
$Neg_{chosen}[n] \leftarrow \text{PruneEasy}(M[n], Neg[n][1])$ ▷ ***Prune easy negatives*** *from* $Neg[n][1]$ *using* $M[n]$
for $j = 2$ to n_B **do**
$N^H \leftarrow \text{SelectHard}(M[n], Neg[n][j])$ ▷ ***Select hard negatives*** *from* $Neg[n][j]$ *using* $M[n]$
$F[n] \leftarrow Pos[n] \bigcup Neg_{chosen}[n] \bigcup N^H$ ▷ ***Add hard negatives*** *from* $Neg[n][j]$ *to the training set*
$M[n] \leftarrow \text{TrainClassifier}(F[n])$ ▷ ***Train*** *classifier using the new dataset*
$Neg_{chosen}[n] \leftarrow Neg_{chosen}[n] \bigcup N^H$ ▷ ***Update*** *the chosen* ***negatives***
$Neg_{chosen}[n] \leftarrow \text{PruneEasy}(M[n], Neg_{chosen}[n])$ ▷ ***Prune easy negatives*** *from* $Neg_{chosen}[n]$ *using* $M[n]$
end for
end for
Return M ▷ ***Return*** *the final classifiers*

In Alg. 1, we report the pseudo-code of the *Minibootstrap* [115] procedure. Note that, we use the Sample(*Set, Sample size*) function to extract *Sample size* random tensors from the given *Set*. We will use this function also in Alg. 2 and Alg. 3.

11.15 Appendix F

Algorithm 2 Incremental RPN Pseudo-code of the *incremental* feature extraction procedure for the *On-line RPN*.

Input: BS, n_B: size and number of *Minibootstrap* batches
Pos^{t-1}, Neg^{t-1}: **positive** and per-image **negative** features at iteration $t-1$ for a generic anchor a
I^t: t^{th} sequence of training images
$\#IMG^t = \#IMG^{t-1} + |I^t|$: total number of training images seen in the previous iterations and in this sequence
Output: Pos^t, Neg^t: **positive** and per-image **negative** features at iteration t for anchor a
Stage 1*: Sample negative features from the ones at iteration* $t-1$
for $j = 1$ to $\#IMG^{t-1}$ **do**
$Neg^t[j] \leftarrow \text{Sample}(Neg^{t-1}[j], \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$
end for
Stage 2*: Append new features from sequence* I^t
$Pos^t \leftarrow Pos^{t-1}$
for $i = 1$ to $|I^t|$ **do**
if $I^t[i]$ has positives for anchor a $Pos_{I^t[i]}$ **then**
$Pos^t \leftarrow Pos^t \bigcup Pos_{I^t[i]}$ ▷ ***Add positives*** *for anchor a from the image* $I^t[i]$
end if
$Neg^t \leftarrow Neg^t \bigcup \text{Sample}(Neg_{I^t[i]}, \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$ ▷ ***Add negatives*** *for anchor a from the image* $I^t[i]$
end for
Return Pos^t, Neg^t ▷ ***Return*** *positive and negative features at iteration t*

We report the pseudo-code for the incremental feature extraction pipeline for the *On-line RPN*. Since the procedure is equal for all the considered anchors, in Alg. 2 we report the algorithm for a generic anchor a.

11.16 Appendix G

Algorithm 3 Incremental On-line Detection *Incremental* feature extraction pseudo-code for the *On-line Detection Module*.

Input: BS, n_B: size and number of the *Minibootstrap* batches
N^t: number of classes. It comprises the N^{t-1} classes known at iteration $t-1$ and the novel classes at iteration t
Pos^{t-1}, Img^{t-1}_{Neg}: sets of N^{t-1} (one for each class) **positive** and per-image **negative** features at iteration $t-1$
Img^{t-1}_{Buf}: set of per-image **buffer** features at iteration $t-1$
I^t: t^{th} sequence of training images
$\#IMG^t = \#IMG^{t-1} + |I^t|$: total number of training images in the previous iterations and in this sequence
Output: Pos^t, Neg^t: N^t sets of **positive** and **negative** training features at iteration t
Stage 1*: Sample negative per-image features from the previous sequences*
for $i = 1$ to $\#IMG^{t-1}$ **do**
  for $n = 1$ to N^t **do**
    if $n \leq N^{t-1}$ **then**
      $Img^t_{Neg}[n][i] \leftarrow \text{Sample}(Img^{t-1}_{Neg}[n][i], \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$
    else
      $Img^t_{Neg}[n][i] \leftarrow \emptyset$ ▷ *Set per-image negatives of the old sequences to an* ***empty set*** *for the* ***new classes***
    end if
  end for
  $Img^t_{Buf}[i] \leftarrow \text{Sample}(Img^{t-1}_{Buf}[i], \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$
end for
$Pos^t \leftarrow Pos^{t-1} \bigcup \bigcup_{i=0}^{N^t - N^{t-1}} \emptyset$ ▷ ***Compute*** Pos^t ***from*** Pos^{t-1} *adding an* ***empty set*** *for each* ***new class***
Stage 2*: Sample features from the* t^{th} *sequence of images*
for $i = 1$ to $|I^t|$ **do**
  for $n = 1$ to N^t **do**
    if $I^t[i]$ has positives of class n $Pos_{I^t[i],n}$ **then**
      $Pos^t[n] \leftarrow Pos^t[n] \bigcup Pos_{I^t[i],n}$ ▷ ***Add positives*** *for class n from* $I^t[i]$
      $Img^t_{Neg}[n] \leftarrow Img^t_{Neg}[n] \bigcup \text{Sample}(Neg_{I^t[i],n}, \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$ ▷ ***Add*** *per-image* ***negatives*** *for class n*
    else
      $Img^t_{Neg}[n] \leftarrow Img^t_{Neg}[n] \bigcup \emptyset$
    end if
  end for
  $Img^t_{Buf} \leftarrow Img^t_{Buf} \bigcup \text{Sample}(All\_feat_{I^t[i]}, \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$ ▷ ***Add*** *per-image* ***buffer negatives*** *from* $I^t[i]$
end for
Stage 3*: Fill* ***negative*** *batches*
for $n = 1$ to N^t **do**
  $Neg^t[n] \leftarrow \emptyset$
  for $i = 1$ to $\#IMG^t$ **do**
    if $(Img_{Neg}[n][i] \neq \emptyset)$ **then** ▷ *If any, add to the* ***negatives*** *of class n, per-image* ***negatives*** *for class n*
      $Neg^t[n] \leftarrow Neg^t[n] \bigcup Img^t_{Neg}[n][i]$
    else ▷ *Otherwise, add to the* ***negatives*** *of class n, per-image* ***buffer negatives***
      $Neg^t[n] \leftarrow Neg^t[n] \bigcup Img^t_{Buf}[i]$
    end if
  end for
end for
Return Pos^t,Neg^t ▷ ***Return*** *positive and negative features at iteration t*

In Alg. 3, we report the pseudo-code for the incremental feature extraction pipeline for the *On-line Detection Module*.

Chapter 12

A Grasp Pose is All You Need: Learning Multi-fingered Grasping with Deep Reinforcement Learning from Vision and Touch

Federico Ceola, Elisa Maiettini, Lorenzo Rosasco and Lorenzo Natale

ABSTRACT

Multi-fingered robotic hands have potential to enable robots to perform sophisticated manipulation tasks. However, teaching a robot to grasp objects with an anthropomorphic hand is an arduous problem due to the high dimensionality of state and action spaces. Deep Reinforcement Learning (DRL) offers techniques to design control policies for this kind of problems without explicit environment or hand modeling. However, state-of-the-art model-free algorithms have proven inefficient for learning such policies. The main problem is that the exploration of the environment is unfeasible for such high-dimensional problems, thus hampering the initial phases of policy optimization. One possibility to address this is to rely on off-line task demonstrations, but, oftentimes, this is too demanding in terms of time and computational resources. To address these problems, we propose the *A Grasp Pose is All You Need* (G-PAYN) method for the anthropomorphic hand of the iCub humanoid. We develop an approach to automatically collect task demonstrations to initialize the training of the policy. The proposed grasping pipeline starts from a grasp pose generated by an external algorithm, used to initiate the movement. Then a control policy (previously

trained with the proposed G-PAYN) is used to reach and grab the object. We deployed the iCub into the MuJoCo simulator and use it to test our approach with objects from the YCB-Video dataset. Results show that G-PAYN outperforms current DRL techniques in the considered setting in terms of success rate and execution time with respect to the baselines.
The code to reproduce the experiments is released together with the paper with an open source license[1].

12.1 Introduction

Robotic grasping is one of the most important manipulation tasks, due to its importance for other downstream tasks such as pick-and-place [35], in-hand manipulation [29], or objects stacking [90].

While two-fingered grasping has been extensively studied in the literature [112, 191, 92, 75], grasping with multi-fingered robotic hands is still an open problem. Although grasping with two-fingered grippers is easier to plan and execute, anthropomorphic hands offer the opportunity to perform dexterous tasks such as objects re-orientation [139], and enable robots to use tools such as hammers [159]. However, due to the intrinsic difficulty of the task, which requires controlling tens of degrees of freedom (DoFs) finding suitable manipulation strategies is challenging.

The latest advancements in the DRL literature provide tools to design high-dimensional control policies without requiring specific environment and hand modeling. State-of-the-art model-free algorithms such as SAC [62] or PPO [175], have proven inefficient to learn policies on multi-fingered manipulation tasks. This is due to the fact that, in these cases, an efficient exploration of the environment at the beginning of policies optimization is unfeasible due to the high dimensionality of the problem. Some recent methods propose to address this problem leveraging on data acquired from off-line task demonstrations, and to combine them with data acquired on-line during policy training. While these approaches have shown promising results, collection of demonstrations is a non-trivial procedure which requires appropriate tools, such as MoCap [189] or Virtual Reality [88] systems.

Another problem with state-of-the-art methods is that they typically use information such as poses and velocities of joints and objects that are difficult to obtain, or that can be noisy in practice [159, 40, 87].

[1] https://github.com/hsp-iit/rl-icub-dexterous-manipulation

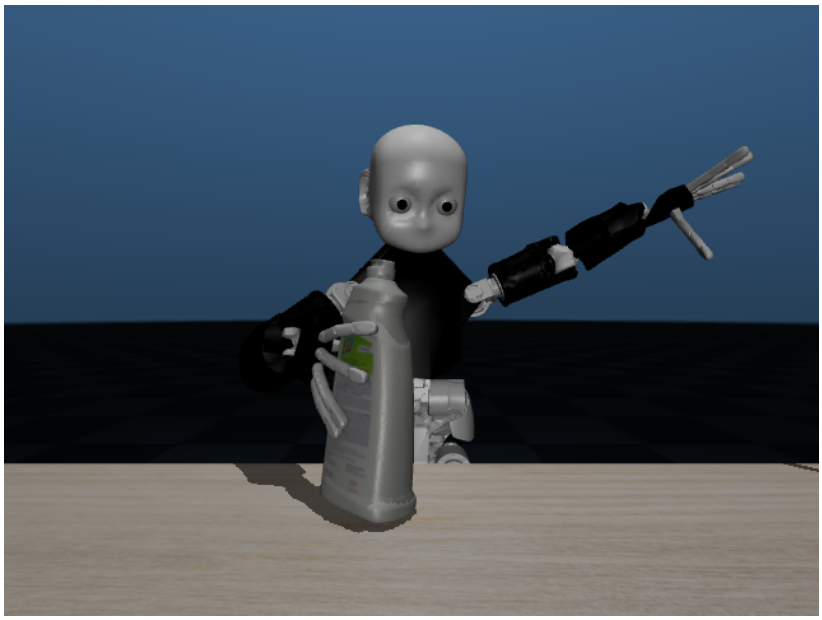


Figure 12.1 **iCub simulated environment**.

In this work we we aim at overcoming these limitations by proposing a grasping method for the iCub [123] humanoid robot based on DRL that leverages on automatically collected demonstrations. To our knowledge, this is the first method that learns this task from RGB data, tactile and proprioceptive information. We start from a grasp pose generated by an external algorithm, using it as a prior information for our task. We assume that this initial pose is inaccurate and should be refined given the specific object and grasping hand. For this reason, the robot first moves the end-effector close to this pose to start the grasping movement, and then uses a separate policy to approach and grasp the object. We train the policy with the proposed **G-PAYN**. The method firstly automatically acquires a set of demonstrations leveraging on a given grasp planner, and then it trains a policy starting from the data originating from the execution of such demonstrations. We design a reward function for the training process that uses a measure of grasp success or failure, but also takes into consideration intermediate steps of the grasping movement. For example, we use information from the tactile sensors, and a positive reward for those hands configurations that increase the number of contacts to achieve a more stable grasp.

We test our approach on five objects from the YCB-Video [212] dataset and we consider two different grasp pose generators to evaluate how their choice affects performance. We benchmark our method against three DRL baselines, outperforming them in all the experiments. Moreover, we compare the success rate of the proposed grasping pipeline against the baseline used to generate the task demonstrations in the proposed **G-PAYN**. Experiments

show that the learned policies surpass the baseline in half of the cases and perform comparably well in the remaining instances. This demonstrates that the proposed **G-PAYN** does not just imitate the behavior of the off-line demonstrations but it refines the movements, adapting them to the specific object.

We run our experiments in a simulated tabletop setting. To this aim, we deployed a MuJoCo [193] model for the iCub humanoid, which, as a further contribution of this paper, we also make publicly available together with the code to reproduce the experiments. In Fig. 12.1, we show a snapshot of the simulated environment.

Finally, in the video attached as supplementary material to the manuscript[2], we show a grasping demonstration on the real iCub humanoid, relying on a policy trained in simulation.

12.2 Related Work

We propose a DRL-based application for multi-fingered grasping, leveraging on automatically collected demonstrations to train our policy. In the following subsections, we cover the main literature on multi-fingered grasping and on DRL from demonstrations.

12.2.1 Multi-fingered Grasping

Multi-fingered grasping is a challenging task due to the high number of DoFs involved and complex hand-object interactions. Some recent works [230, 95] propose methods for multi-fingered grasp synthesis starting from pointcloud information. These methods are difficult to apply because they are constrained to the hardware that is used for training, and they do not take into account the hand-object interactions during grasp execution. The approach described in [101] deals with the high number of DoFs in the Shadow hand with a PCA-based hand synergy. Then, similarly to our solution, it trains a DRL policy to grasp an object starting from a grasp pose given by an external algorithm. However, this method uses as input to the policy binary tactile information, joint torques (which might not be available in all the robots) and hand joint positions, without considering any information of the object (e.g. object position or visual feedback) that would allow grasp recovery if the grasp pose is not suitable. Other approaches, such as the one in [31], instead learn grasping policies using data collected with a MoCap system, with the aim of reducing the amount of training data, since the data collection procedure for multi-fingered grasping is challenging. In our

[2]https://youtu.be/qc6gksKH3Mo

approach, we propose a method to overcome this issue by automatically collecting off-line demonstrations.

12.2.2 Deep Reinforcement Learning from Demonstrations

Methods that learn DRL policies leveraging on demonstrations can be grouped in two categories. The first is composed of methods that use demonstrations throughout the training. Two exemplar methods are DDPGfD [195] and the approach proposed in [132]. Both methods modify the DDPG [102] algorithm to leverage on the demonstrations included in the replay buffer. The second class of methods uses the demonstrations for pre-training a policy, either with behavior cloning or with DRL, and then fine-tunes such policy on data acquired on-line. Two exemplar approaches are DAPG [159] and AWAC [131]. The former learns several dexterous manipulation tasks, leveraging on pre-training from demonstrations with behavior cloning and then fine-tuning the policy with an augmented loss to stay close to the demonstrations. The latter mitigates the distribution shift between the off-line demonstrations and the data acquired on-line during training. In this work, we consider an approach from each of the above classes as baselines for our experiments, namely [132], adapted to the considered setting, and AWAC [131].

12.3 Methodology

12.3.1 Grasping Pipeline

We propose a modular pipeline for grasping an object with the iCub humanoid (see Fig. 12.2). Our approach is composed of two stages. We firstly compute a suitable grasp pose with an external algorithm and we move the end-effector of the robot in a pre-grasp pose spaced $5cm$ from the one given by the algorithm. Then, starting from this configuration, we rely on a DRL policy to move the end-effector in the cartesian space and to control the positions of the finger joints in order to accomplish the grasp and lift the object.

Grasp Pose Computation

We rely on two different algorithms for grasp pose computation:

- The approach based on superquadric models proposed in [196]. This is specifically designed to compute a grasp pose for the iCub humanoid.

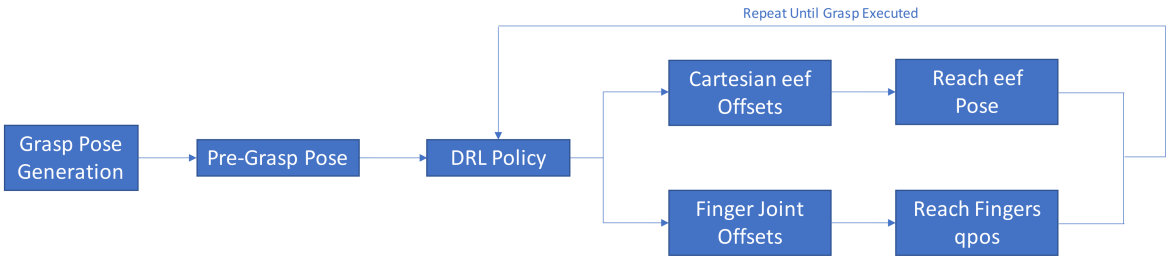


Figure 12.2 **Overview of the proposed grasping pipeline**. We rely on a *Grasp Pose Generator* to compute a suitable grasp pose for the considered object. Then, we move the end-effector of the robot to a *Pre-Grasp Pose* close to the previously generated grasp pose. Finally, we use a *DRL Policy* to predict cartesian offsets to move the end-effector toward the object and offsets in the joint space of the fingers to grasp it. We repeat this procedure until the grasp is executed.

- The state-of-the-art grasp pose generator VGN [11]. This approach computes grasp poses for a two-fingered gripper mounted on a Franka Emika Panda robot. To compute a feasible grasp pose for the iCub, we post-process the grasping candidates proposed by VGN by rotating the original grasp pose of 45°, see Fig. 12.3 *(b)*. We then analyze the reachability of the rotated grasp poses in decreasing order of confidence until a suitable candidate is found.

Grasp Execution

We model the grasp execution task as a Markov Decision Process (MDP) $\{S, A, T, r\}$. At each timestep t, the robot observes the state $s_t \in S$ of the system. This has five components:

- A visual information of the environment. This is computed starting from the RGB image acquired by the camera mounted on the iCub's head at a given timestep t. We process this with the *ViT-B/32* model[3] pre-trained with CLIP [156] to extract a latent representation of the image. Finally, to encode temporal information, we combine this latter with the latent representations obtained at the timesteps $t-1$ and $t-2$ with the *Flare* architecture [177].
- The cartesian pose of the end-effector.
- Finger joint positions (*qpos*).
- A binary tactile value for each of the fingertips of the iCub hand, encoding the information of the contact with the object to grasp.

[3]We rely on the pre-trained models available at https://github.com/openai/CLIP.

- The center of the superquadric that fits the object to be grasped, or the median point of object's pointcloud for VGN. This value is taken at the beginning of the episode and remains unchanged throughout its duration. We estimate the center of the object in this way because on the real robot the actual position of the object is not available. This estimate may be error prone and non-deterministic (e.g. due to the random sampling of the pointcloud for superquadrics estimation), and therefore it may affect the trained policies. However, we show that our approach is able to learn grasping policies using the considered, possibly imprecise, estimation of the center of the object.

At each timestep t, given the current state of the system s_t, the robot acts in the environment with unknown dynamics T, according to a policy $\pi(a_t|s_t)$. Specifically, the action $a_t \in A$ has two components:

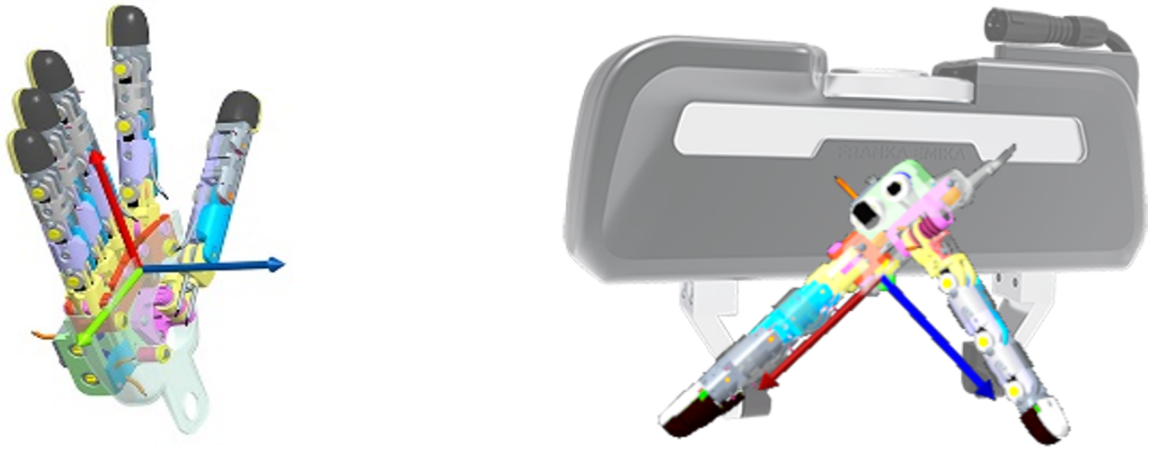
- A cartesian offset that represents the movement that the end-effector of the robot needs to accomplish w.r.t. its current cartesian pose. This is used to reach the object in a suitable configuration, starting from the pre-grasp pose.
- The movement that the finger joints have to perform w.r.t. the current *qpos*. This is needed to close the fingers and grasp the object.

We learn the policy π^* with the SAC [62] algorithm by maximizing the maximum entropy objective:

$$\pi^* = \arg\max_{\pi} \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))], \tag{12.1}$$

This adds a weighted entropy component $\alpha\mathcal{H}(\pi(\cdot|s_t))$ to the standard action-value function $\arg\max_\pi \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t)]$, where ρ_π denotes the state-action marginal of the trajectory distribution induced by a policy π, with the aim of encouraging exploration during training. Specifically, for our task, we design the reward function $r(s_t,a_t)$ in eq. (12.1), which evaluates the outcome of action a_t when the state of the environment is s_t, as the sum of the following components:

- $r_{fingers} = f(t+1) - f(t)$, where f denotes the number of fingers in contact with the object at the given timestep.
- $r_{dist\_object\_center}$ guides the end-effector to approach the object. It considers the distance d, measured in *cm*, of the object position estimated at the beginning of the episode



(a) *(b)*

Figure 12.3 *(a)* **iCub hand reference frame.** *(b)* **VGN grasp transformation for the iCub hand.** We rotate the grasp pose generated for the Franka Emika Panda gripper by 45° to obtain the corresponding grasp pose for the iCub hand.

w.r.t. the x and y axes of the iCub hand reference frame (red and green axes in Fig. 12.3 *(a)*). Specifically, for the superquadrics-based approach, we consider the center of the superquadric, while we use the median point of the object's pointcloud, when considering VGN. The value of this component is equal to $d(t+1) - d(t)$ if $f(i) < 2$ for all the timesteps $i \in 0, ..., t+1$, otherwise it is equal to 0.

- r_{object_height}: this component rewards the difference of the object height h measured in mm with respect to the table between two consecutive timesteps, $t+1$ and t, therefore evaluating whether the object has been lifted from the table in that time frame. We consider this term only from the moment when at least two fingers touch the object. Specifically, if $f(t+1) \geq 2$ and $h(t+1) - h(t) > 0$, or $f(i) \geq 2$ for at least one timestep $i \in 0, ..., t+1$ and $h(t+1) - h(t) < 0$, $r_{object\_height} = f(t+1) * (h(t+1) - h(t))$, otherwise $r_{object\_height} = 0$.

- $r_{end\_of\_episode}$ evaluates the termination of the episode. An episode might end in one of the following cases: *(i)* the object is grasped and uplifted by $10cm$, *(ii)* the object is moved too far from the initial position, *(iii)* the inverse kinematic solver cannot find a solution for the required configuration of the end-effector, or *(iv)* the current timestep is equal to the maximum number of allowed timesteps per episode. In the first case, the robot has successfully grasped the object and the term $r_{end\_of\_episode}$ is equal to 1. In all other cases, the episode is considered a failure and thus the term is equal to -1.

Algorithm 4 G-PAYN Pseudo-code for policy training.

1: Let $a_t \in \mathbb{R}^{15} = a_{EEF\_POS\_t} \in \mathbb{R}^3 \bigcup a_{EEF\_RPY\_t} \in \mathbb{R}^3 \bigcup a_{FINGERS\_t} \in \mathbb{R}^9$
2: Let $qpos_{OPEN} \in \mathbb{R}^9$ and $qpos_{CLOSE} \in \mathbb{R}^9$ be the finger joint positions with open and close hand, respectively.
3: $\#Steps = 0$
4: $RB = \emptyset$ ▷ **Initialize** an empty replay buffer.
5: **while** $\#Steps < RB\_size$ **do** ▷ **Collect** demonstrations.
6: Compute grasp $\bar{x}_{GP\_POSE} = \bar{x}_{GP\_POS} \bigcup \bar{x}_{GP\_RPY}$ and pre-grasp $\bar{x}_{PGP\_POSE} = \bar{x}_{PGP\_POS} \bigcup \bar{x}_{PGP\_RPY}$ poses.
7: Move the end-effector to $\bar{x}_{PGP\_POSE}$.
8: $\#ep\_steps = 0$
9: $tmp\_a_{FINGERS} = [0, ..., 0]$
10: **while** not done **do**
11: **if** $\#ep\_steps < 100$ **then** ▷ **Approach** the object.
12: $a_{EEF\_POS\_t} = \frac{\bar{x}_{GP\_POS} - \bar{x}_{PGP\_POS}}{100}$
13: $a_{EEF\_RPY\_t} = [0, 0, 0]$
14: $a_{FINGERS\_t} = [0, ..., 0]$
15: **else if** $\#ep\_steps < 600$ **then** ▷ **Close** fingers.
16: $a_{EEF\_POS\_t} = [0, 0, 0]$
17: $a_{EEF\_RPY\_t} = [0, 0, 0]$
18: Let $qpos_t \in \mathbb{R}^9$ be the finger joint positions at timestep t.
19: $a_{FINGERS\_t\_1} = \frac{qpos_{CLOSE} - qpos_{OPEN}}{250}$
20: $a_{FINGERS\_t\_2} = qpos_{OPEN} + \frac{(\#ep\_steps - 100)(qpos_{CLOSE} - qpos_{OPEN})}{500} - qpos_t$
21: $a_{FINGERS\_t} = \bigcup_{i=1}^{9} \min(a_{FINGERS\_t\_1}[i], a_{FINGERS\_t\_2}[i])$
22: $tmp\_a_{FINGERS} = a_{FINGERS\_t}$
23: **else** ▷ **Lift** the object.
24: $a_{EEF\_POS\_t} = [0, 0, 0.002]$
25: $a_{EEF\_RPY\_t} = [0, 0, 0]$
26: $a_{FINGERS\_t} = tmp\_a_{FINGERS}$
27: **end if**
28: Execute action a_t from the current state s_t, observe s_{t+1}, and compute reward $r(s_t, a_t)$.
29: $\#ep\_steps = \#ep\_steps + 1$
30: $\#Steps = \#Steps + 1$
31: $RB = RB \bigcup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
32: **end while**
33: **end while**
34: Initialize *SAC*'s replay buffer with *RB*.
35: Train *SAC* for $5M$ timesteps, initializing episodes with the procedure in lines 6 and 7.

12.3.2 Policy Training

We propose a two-stage algorithm to learn the grasping policy described in Sec. 12.3.1: we first design an automatic procedure for the acquisition of grasping demonstrations, and then we learn the policy with the SAC [62] algorithm, leveraging on the previously acquired data.

The proposed pipeline for demonstrations collection is composed of three steps. Firstly, once the end-effector has reached the considered pre-grasp pose (as described in Sec. 12.3.1), we move it toward the grasp pose on a straight line by splitting the trajectory in 100 waypoints,

keeping the fingers of the hand open. Then, we adaptively close the fingers in 500 steps, maintaining the end-effector in the grasp pose. Finally, we uplift the object from the table by increasing the height of the end-effector by $2mm$ for each step, until the object is uplifted by $10cm$.

We then train our grasping policy with a modified version of the SAC [62] algorithm. Instead of starting with an empty replay buffer, we fill the initial replay buffer with demonstrations collected with the pipeline described above. We found this strategy particularly effective and easy to implement. It can be potentially applied to all the off-policy DRL algorithms without requiring any adaptation of the loss function to manage the distribution shift between the off-line demonstrations and the transitions acquired during training. The pseudo-code of the whole training procedure is reported in Alg. 4.

12.4 Experimental Setup

12.4.1 Simulated Environment

To learn our task, we deployed a simulated environment for manipulation tasks with the iCub humanoid using the MuJoCo [193] simulator (see Fig. 12.1). We implemented our environment as a Gym [12] interface to train policies with the main libraries implementing state-of-the-art DRL algorithms (e.g. Stable-Baselines3 [158]).

We designed the model of the simulated iCub to be as similar as possible to the real robot, to reduce the sim-to-real gap. For example, in our environment, we simulate the *Intel(R) RealSense D415* headset mounted on the real iCub robot that acquires RGB and depth images. Moreover, we implemented fingers actuation as in the real robot, e.g. using a tendon to simulate the actuator that controls the six joints in the little and ring fingers.

Finally, we integrated the iKin [148] and iDynTree [136] libraries in our environment for inverse kinematics computation, and we adapted the models of the YCB-Video [212] objects to perform manipulation tasks and benchmark our results.

12.4.2 Training Hyperparameters

We train our policy with the implementation of the SAC [62] algorithm available in the Stable-Baselines3 [158] library. We report the considered training hyperparameters in Tab. 12.1. While most of the parameters are the same as the ones in the original SAC [62] implementation, we increase the number of hidden layers units to 1024 to deal with the

Parameter	Value
Optimizer	Adam [80]
Learning Rate	$3 \cdot 10^{-4}$
Discount (γ)	0.99
Replay Buffer size	10^6
Number of Hidden Layers (all networks)	2
Number of Hidden Units per Layer	1024
Number of Samples per Minibatch	256
Entropy Target	-15
Non-linearity	ReLU
Target Smoothing Coefficient (τ)	0.005
Target Update Interval	1
Gradient Steps	1
Training Frequency	10 Timesteps
Total Environment Timesteps	$5 \cdot 10^6$

Table 12.1 G-PAYN Training Hyperparameters.

high-dimensionality of our state space and we increase the training frequency to 10 timesteps to avoid implicit underparameterization [86].

12.5 Results

We benchmark our approach on five objects from the YCB-Video [212] dataset. Specifically, we consider the *004_sugar_box*, the *006_mustard_bottle*, the *010_potted_meat_can*, the *021_bleach_cleanser*, and the *035_power_drill*. We chose these objects, that are graspable with the iCub hand, to evaluate our approach on different types of grasps. For example, when considering the superquadrics-based approach for grasp pose computation, the *004_sugar_box* is grasped from a top-down direction, while the *021_bleach_cleanser* is grasped from a lateral configuration. We show results in Fig. 12.4, where we evaluate the grasping success rate for increasing environment timesteps (see Tab. 12.1) for different objects and grasp pose generators. For each experiment, we set the object in a random initial position that is reachable by the iCub, and we randomly rotate it around the axis perpendicular to the table.

12.5.1 Baselines

We compare the proposed **G-PAYN** (blue line in Fig. 12.4) to four different baselines:

- **Demonstrations Pipeline** (orange line in Fig. 12.4): this is the approach described in Sec. 12.3.2 that is used to automatically collect the demonstrations for the training. Since this method does not require training, it has a constant success rate. We consider the success rate obtained while acquiring demonstrations to fill the replay buffer for the corresponding experiment.

- **SAC** (green line in Fig. 12.4): this is the standard SAC [62] algorithm trained with the hyperparameters in Tab. 12.1. Differently from **G-PAYN**, it starts with an empty replay buffer.

- **OERLD** (red line in Fig. 12.4): we train a grasping policy with the loss function proposed in [132]. For a fair comparison with the other methods, differently from the approach in [132], we combine the behavior cloning loss with the actor loss of the SAC [62] algorithm. Moreover, since we designed our task with a dense reward function with a well defined goal state, we do not apply the procedure to overcome the problem of sparse rewards described in [132], and called *resets to demonstration states* by the authors. As for the other approaches, we consider as the demonstrations replay buffer the same replay buffer used at the beginning of the **G-PAYN** training. At every training step, we sample 256 transitions from the replay buffer and 32 transitions from the demonstrations.

- **AWAC** (purple line in Fig. 12.4): this is the approach proposed in [131]. We rely on the implementation in [176]. For training, we consider the default hyperparameters, setting the batch size to 256 and the number of per layer hidden units to 1024. During off-line training, we use as demonstrations data the same replay buffer used for **G-PAYN**'s warm start. Fig. 12.4 reports the success rate for increasing timesteps of the on-line training stage.

12.5.2 Discussion

Results in Fig. 12.4 show that **G-PAYN** achieves at least a comparable success rate to the **Demonstrations Pipeline**, with the exception of the *021_bleach_cleanser* experiment when using VGN as grasp pose generator. More importantly, in half of the experiments **G-PAYN** surpasses **Demonstrations Pipeline**, and in some cases, e.g. in the *010_potted_meat_can+VGN* or in the *021_bleach_cleanser+Superquadrics* experiments, its success rate outperforms the baseline by a large margin (~ 0.3 and ~ 0.15 gap in the success rate for

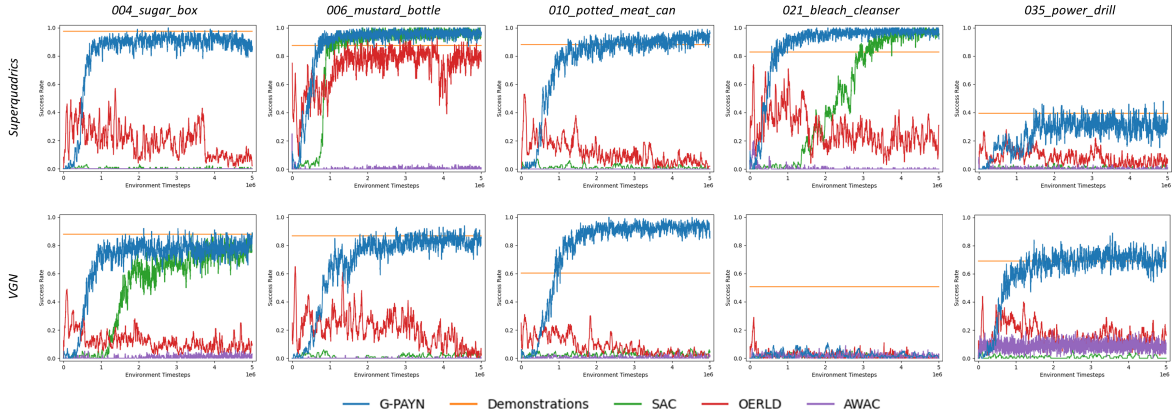


Figure 12.4 **Results**. We compare the considered methods for grasping execution on different objects and different grasp pose generators. In the first row we consider the approach based on superquadric modeling proposed in [196] for grasp pose generation. In the second row, instead, we use VGN. In each column, we report results for different YCB-Video objects.

the two experiments, respectively). Notably, **G-PAYN** outplays all the DRL baselines. **SAC**, due to the high dimensionality of the problem, suffers from the lack of initial demonstrations in the replay buffer and achieves a comparable success rate to **G-PAYN** in only three experiments. Moreover, in these three cases, **SAC** requires a significantly higher number of training timesteps to achieve the same success rate as **G-PAYN**. **OERLD**, instead, achieves an acceptable success rate only in the *006_mustard_bottle+Superquadrics* experiment. In all the other cases, despite the initial highest success rate obtained thanks to the behavior cloning component in the loss function, this method is not able to improve the performance throughout the training. Finally, **AWAC** achieves the worst results among the considered methods. In preliminary experiments, we noticed that training a policy on the task at hand with SAC [62] or behavior cloning on off-line data is impractical. This motivates the low success rate achieved by **AWAC**, since at the beginning of the optimization it is trained off-line on the demonstrations.

Results in Fig. 12.4 show also the importance of the initial grasp pose used as a prior information by all the considered methods. Experiments with the *021_bleach_cleanser* provide evidence of this. In fact, the *021_bleach_cleanser+Superquadrics* policy achieves a success rate close to 1, while the success rate of the *021_bleach_cleanser+VGN* experiment is close to 0. This is due to the fact that, for the *021_bleach_cleanser*, grasp poses generated by the superquadrics-based algorithm are always feasible for the iCub (see also the qualitative evaluation in Fig. 12.5), while VGN predicts top-down grasp poses which, especially for tall objects, are difficult to reach by the iCub, and add a degree of complexity to the task, requiring a much more precise grasping procedure. This aspect further motivates the need

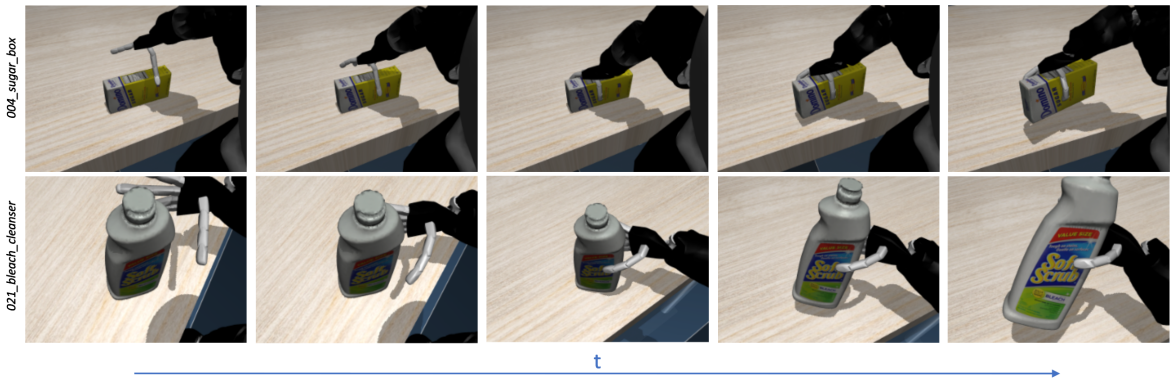


Figure 12.5 **Qualitative evaluation**. We show examples of our method on two of the five objects considered in the experiments (the *004_sugar_box* and the *021_bleach_cleanser*) using the approach based on superquadrics for grasp pose generation. We show that the learned policies manage to successfully approach the objects, grasp and uplift them.

for the robotics community to develop methods that strive for both accurate grasp poses synthesis and grasp execution, as we aim to do with the proposed **G-PAYN**.

Qualitative results reported in Fig. 12.5 show that policies obtained with **G-PAYN** behave differently from the ones collected with the **Demonstrations Pipeline**. Indeed, instead of splitting the grasp execution in three phases as explained in Alg. 4, **G-PAYN** policies perform a continuous movement that closes the fingers already while approaching the object. Moreover, the trained policies manage to solve the task in fewer steps ($\sim$ 100 steps for **G-PAYN** vs. $\sim$ 650 steps for the **Demonstrations Pipeline**). This indicates that **G-PAYN**, while benefiting from the demonstrations as a warm start for the training, is able to learn policies that effectively optimize the task-specific reward function, instead of just imitating the off-line data. This further supports our choice to use demonstrations only as a warm start for policy training in contrast to those methods that are instead constrained to off-line data such as **OERLD** and **AWAC**. Indeed, the former attempts at imitating the off-line data throughout the entire training session with the loss component for behavior cloning. The latter, instead, learns a policy only from the demonstrations during the off-line training phase. This aspect leads to poor results for both methods, compared to the performance of our approach.

Finally, we qualitatively evaluate the *006_mustard_bottle +Superquadrics* policy trained in simulation on the real iCub[11], without any fine-tuning. While the deployment of a sim-to-real method for transferring policies on the real robot is out of the scope of this paper and left as future work, we show that our policies can be deployed on the real robot without requiring any adaptation of action and state spaces.

12.6 Conclusions

Multi-fingered grasping is an important task for robots that need to perform dexterous manipulation tasks. However, due to the difficulty of designing grasping strategies to control robotic hands with tens of DoFs, solving this task is still an open problem.

To fill this gap, we propose **G-PAYN**, a DRL approach that leverages on automatically collected demonstrations and on an initial grasp pose generated by an external algorithm for grasp synthesis. We learn the task using visual, tactile and proprioceptive information as inputs, and we show that our approach outperforms all the considered DRL baselines. Our approach also outperforms the success rate achieved by the pipeline for collection of off-line demonstrations in half of the experiments, achieving a comparable performance in almost all the remaining instances.

With our experiments, we highlight the importance of a suitable initial grasp pose to effectively learn the task. As a future work, we plan to improve our approach by developing a learning method that integrates different approaches to generate initial grasp poses and then selects the best one based on past experience.

While our approach has shown its effectiveness to learn the multi-fingered grasping task, we plan to further improve our method by speeding-up the training procedure and making it feasible on the real robot. At this aim it could be beneficial to incorporate a component for behavior cloning in the loss function during the initial learning steps, to take advantage of the steep learning curve demonstrated by **OERLD**. Finally, we plan to extend our approach to deal with distractors in the environment and with external forces.

Chapter 13

RESPRECT: Speeding-up Multi-fingered Grasping with Residual Reinforcement Learning

Federico Ceola, Lorenzo Rosasco and Lorenzo Natale

ABSTRACT

Deep Reinforcement Learning (DRL) has proven effective in learning control policies using robotic grippers, but much less practical for solving the problem of grasping with dexterous hands – especially on real robotic platforms – due to the high dimensionality of the problem.

In this work, we focus on the multi-fingered grasping task with the anthropomorphic hand of the iCub humanoid. We propose the *RESidual learning with PREtrained CriTics* (RESPRECT) method that, starting from a policy pre-trained on a large set of objects, can learn a residual policy to grasp a novel object in a fraction ($\sim 5\times$ faster) of the timesteps required to train a policy from scratch, without requiring any task demonstration. To our knowledge, this is the first Residual Reinforcement Learning (RRL) approach that learns a residual policy on top of another policy pre-trained with DRL. We exploit some components of the pre-trained policy during residual learning that further speed-up the training. We benchmark our results in the iCub simulated environment, and we show that RESPRECT can be effectively used to learn a multi-fingered grasping policy on the real iCub robot.

The code to reproduce the experiments is released together with the paper with an open source license[1].

[1] https://github.com/hsp-iit/rl-icub-dexterous-manipulation

13.1 Introduction

Learning dexterous manipulation tasks is an open challenge in robotics [4]. These tasks require controlling tens of degrees of freedom (DoFs), to deal with possibly imprecise perception of the environment, and to manage hand-object interactions. Grasping is the key task to enable the execution of other dexterous manipulation tasks such as object re-orientation [4, 29].

The latest model-free DRL approaches, such as SAC [62] or PPO [175], are feasible options to learn problems with high-dimensionality. However, their deployment on real robotic platforms is difficult due to the huge amount of timesteps required to explore the environment at the beginning of the training. A major trend in the literature to overcome this problem is to train a policy in simulation and transfer it on the real robot [154]. However, the success of these approaches depends on the sim-to-real gap between the simulated and the real environments. With our work, we aim at overcoming this limitation by proposing a method for fast learning of the multi-fingered grasping task.

The contribution of this paper is a method that we call RESPRECT. The core of this algorithm is an RRL method that leverages on a pre-trained base policy to learn a residual additive policy on a novel object. In contrast to existing residual learning methods, which usually rely on classical closed-loop controllers as base policies, we show that it is possible to learn a residual policy also if a hand-tuned closed-loop base controller is unavailable, as for the considered multi-fingered grasping task.

As a further contribution, we propose to exploit pre-trained components to speed-up the residual training. We initialize the SAC *Critics* components in the residual policy using the weights of the pre-trained policy. In the experimental section we show that this leads to a significant reduction of the training time, making it suitable for learning policies directly in the real world.

We benchmark our results against conventional fine-tuning and Meta Reinforcement Learning (MetaRL) approaches in a simulated environment with the 9-DoFs hand of the iCub humanoid [123], surpassing all the considered baselines in most of the experiments. Furthermore, the experimental results show that RESPRECT achieves the same success rate as the state-of-the-art method for multi-fingered grasping G-PAYN [22], while requiring only a fraction of the training timesteps ($1M$ compared to $5M$). Notably, differently from G-PAYN [22], RESPRECT accomplishes this without the need for grasping demonstrations to initialize the training process. While these requirements prevent G-PAYN [22] from

learning grasping policies on the real robot, we deploy RESPRECT on the real iCub, showing that it can learn a grasping policy for two new objects in ~ 2.5 hours and ~ 30 minutes.

To the best of our knowledge, this is the first DRL algorithm that has been successfully used to solve the problem of grasping with an articulated hand with several DoFs directly on the real robot from visual, tactile and proprioceptive data, and without the need for task demonstrations.

13.2 Related Work

Multi-fingered Grasping The literature mainly focuses on grasp poses generation [196] or detection of finger-object contact points [229, 95], whereas the deployment of efficient control strategies to perform the grasp has received less attention. Recent work proposes to solve the problem with DRL-based approaches. The work in [101] starts from a grasp pose given by an external algorithm, and relies on synergies to reduce the number of DoFs to control the fingers of a Shadow hand. The policy is trained on a multi-modal input comprising tactile information, joint angles and torques, which are not always available in other robotic hands. Moreover, it does not take into account information about the object (e.g., visual feedback of the environment or object pose) during grasp execution to allow grasp recovery if the initial grasp pose is unfeasible. G-PAYN [22], instead, considers as proprioceptive information the cartesian pose of the end-effector and finger joint positions, relying on visual feedback from the head-mounted camera of the robot. However, training is performed in simulation with long training sessions relying on huge amounts of grasping demonstrations, and therefore it cannot be performed on the real robot. DexPoint [154], instead, learns to grasp and open a door from pointclouds, showing that sim-to-real transfer can be obtained without fine-tuning. While being an interesting research direction, this transfer is strongly affected by sensors quality. This is particularly evident with objects in proximity to depth sensors. We overcome these limitations providing a method that can be deployed on a real robot for fast learning of multi-fingered grasping from visual, tactile, and proprioceptive data. Notably, differently from the state-of-the-art G-PAYN [22], RESPRECT does not require any task demonstration, while being trained much faster.

Fast DRL State-of-the-art DRL algorithms, e.g. SAC [62] or PPO [175], are sample-inefficient and require huge amounts of training episodes to be optimized, hindering their application on tasks that require to be trained on real robots. Some methods adapt standard DRL algorithms leveraging off-line task demonstrations to overcome this limitation. [195, 132] adapt DDPG [102] to exploit task demonstrations: DDPGfD [195] leverages on a

prioritized replay mechanism to sample transitions between demonstrations and agent data, while [132] adds a Behavior Cloning (BC) loss component to the one of DDPG [102] to mimic demonstrations. DAPG [159], instead, fine-tunes a DRL policy pre-trained on demonstrations with imitation learning. Off-line DRL approaches as AWAC [131], instead, pre-train a policy with DRL on demonstrations and then adapt it on-line on the robot. Results in [22] show that these approaches are not suited for multi-fingered grasping from visual, tactile, and proprioceptive data. A different approach to adapt a DRL policy is fine-tuning. [225] compares fine-tuning to several MetaRL approaches. While being promising for fast adaptation of DRL tasks, MetaRL algorithms are often difficult to use in practice due to their complexity. For example, PEARL [160] requires learning an additional network for task context inference, which is used to condition the trained policy. In some cases, MetaRL methods can be employed only with on-policy DRL algorithms [52]. We benchmark our method against fine-tuning and MetaRL baselines.

RRL RRL methods aim at speeding-up policy learning. They are designed to predict a residual action, that is added to the output of an existing controller. The first RRL approach was introduced in [185] to improve imperfect controllers in simulated manipulation tasks, such as object pushing or pick-and-place. [73, 172] propose RRL methods for insertion tasks on real robots, either from observable and measurable states, or from raw pixels. [162] proposes to modify also the signal to a base feedback controller to avoid the feedback distribution shift caused by the residual policy, which the base controller tries to resist. These approaches share a common limitation, in that they rely on manually designed conventional controllers, which are difficult to design for a multi-fingered grasping task. [42] overcomes this limitation learning a residual policy to solve insertion tasks on top of Dynamic Movement Primitive (DMP) base policies extracted through BC. However, learning DMPs from visual, tactile and proprioceptive data is impractical. [3] proposes an RRL method to improve a policy trained with BC by superimposing a residual policy trained with DRL on seven simulated manipulation tasks (the same tasks used for BC training). This removes the dependency on hand-engineered base controllers, but requires task demonstrations which are difficult to obtain on the real robot. This challenge becomes even more pronounced in the context of this work, where we aim at learning multi-fingered grasping policies on objects unseen during base policy training. Furthermore, evidence from previous work [22] shows that training policies on the task at hand with BC is impractical. Residual learning has also found application on different robotic tasks, such as object throwing [222], where the residual throwing velocity is regressed and superimposed on the velocity predicted by an

ideal physics controller, or to learn navigation strategies [161], where a classical controller serves as the base policy.
We overcome the limitations of the state-of-the-art, which either depend on classical base controllers (unavailable for the considered multi-fingered grasping task), or rely on a policy trained with BC on the same task. The goal of the proposed algorithm is to allow the robot to quickly learn to grasp unseen objects, starting from a DRL policy that is pre-trained on a set of generic objects. To achieve this, we use visual features obtained from a backbone pre-trained on a dataset including egocentric images from everyday tasks, without making any assumption on the target object to be grasped. We leverage the pre-trained DRL policy even further: the residual *Critics* component is initialized using the pre-trained weights of the base policy, significantly speeding-up the training. These enhancements are inherently unattainable using the base policies employed in existing literature.

13.3 Methodology

13.3.1 Grasping Pipeline

We rely on the grasping pipeline introduced in [22], and consider the right hand of the iCub humanoid robot. This is actuated by 9 motors and is equipped with tactile sensors on the fingertips. The pipeline includes two main phases. In the initial phase, the end-effector is moved in a pose spaced $5cm$ from a grasp pose generated by an object agnostic algorithm. This is either an algorithm based on superquadrics [196], hereinafter referred to as *Superquadrics*, or VGN [11]. Subsequently, we employ a DRL policy, trained with the proposed RESPRECT, to approach and lift the object. The DRL policy controls the 6 DoFs of the end-effector's pose and the 9 finger joints to approach the object and finally lift it.

We consider five elements as the state of the Markov Decision Process (MDP) underlying the DRL policy at hand. We use the visual Flare [177] features extracted with the *ViT-Large* model from the Masked Autoencoder presented in [157] (MAE). This model was trained on several real-world visual datasets, comprising Ego4D [60] that well represents the type of visual feedback acquired by the robot head-mounted camera. We also consider a binary tactile value for each fingertip, the cartesian pose of the end-effector, finger joint positions and an estimate of the initial pose of the object to grasp (the latter computed as the center of the superquadric, or the center of the segmented pointcloud, when considering VGN for grasp pose synthesis). The DRL policy outputs a 15-dimensional vector that represents offsets for moving the end-effector and finger joints. The policy is trained with a reward function that

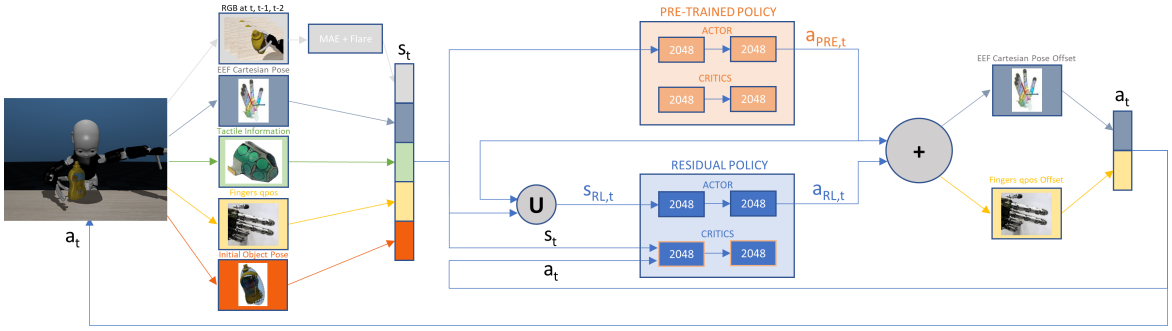


Figure 13.1 RESPRECT overview. We compute state s_t from RGB images at timesteps t, $t-1$ and $t-2$ (processed through the MAE in [157] and combined with Flare [177]), end-effector cartesian pose, tactile information and finger joint poses. We compute action a_t (composed of cartesian offsets for the end-effector and finger joint offsets) combining the outputs $a_{PRE,t}$ of the pre-trained policy and $a_{RL,t}$ of the residual policy. Note that $a_{RL,t}$ is the output of the residual policy *Actor*, given the concatenation of s_t and $a_{PRE,t}$ into $s_{RL,t}$. We train only the two 2048-dimensional fully connected layers in the residual *Actor* and *Critics*. For the latter, we start from the *Critics* weigths of the pre-trained policy (orange outline). For the sake of clarity, we do not report the input of the *Critics* in the pre-trained policy, and the output of both the *Critics* being the same as the ones in SAC [62].

considers: the pose of the hand with respect to the estimated initial position of the object, the number of fingers in contact with the object (detected in the case of tips-object meshes contact in simulation, or when tactile sensors on the real robot are triggered), the position of the object when lifted, and the terminal condition of the episode. An episode is regarded as successfully terminated if the object is grasped and uplifted by $10cm$. Conversely, an episode is deemed a failure if the object is moved too far from the initial position, the inverse kinematics (IK) solver cannot find a solution for the specified configuration, or the number of timesteps exceeds the designated maximum threshold. For further details, we refer the reader to the description in [22].

13.3.2 Residual Policy Training

To train the grasping policy we propose a novel RRL method. This leverages on two components: a policy that is pre-trained with G-PAYN [22] in simulation, and a residual, object-specific, policy trained with a modified version of SAC [62]. The former is trained for two million timesteps using the MuJoCo models of the *Scanned Objects Dataset* (MSO) [48, 220]. In the simulated experiments, the latter is trained on seven YCB-Video [212] objects, while in the real world experiment it is trained on the *006_mustard_bottle* and on the *021_bleach_cleanser* (also taken from YCB-Video). Given the state of the system s_t at time

Parameter	Value
Optimizer	Adam
Learning Rate	$3 \cdot 10^{-4}$
Discount (γ)	0.99
Replay Buffer size	10^6
Number of Hidden Layers (all networks)	2
Number of Hidden Units per Layer	1024
Number of Samples per Minibatch	256
Entropy Target	-15
Non-linearity	ReLU
Target Smoothing Coefficient (τ)	0.005
Target Update Interval	1
Gradient Steps	10
Training Frequency	10 Timesteps
Total Environment Timesteps	$1 \cdot 10^6$
Entropy Coefficient	0.01

Table 13.1 **RESPRECT** Training Hyperparameters.

t, our approach outputs an action a_t for the robot. This is the sum of two components: the action $a_{PRE,t}$ predicted by the pre-trained policy, and the action $a_{RL,t}$ predicted by the residual policy. While training the residual policy, the weights of the pre-trained policy remain fixed, and this is used only to predict $a_{PRE,t}$ which is needed for the optimization of the residual policy. As shown in Fig. 13.1, we modify the standard SAC [62] inputs for the *Actor* and *Critics* components in the residual policy. Specifically, for the *Actor*, we compute the state $s_{RL,t}$ as the concatenation between the state s_t and the action produced by the pre-trained policy $a_{PRE,t}$. This allows the policy to compute the residual action $a_{RL,t}$ conditioned not only to the current state of the system, but also to the action produced by the pre-trained component. The *Critics* are fed with the state s_t and the action a_t, instead of $a_{RL,t}$ and $s_{RL,t}$ as in the standard SAC [62] algorithm. This allows training the residual *Critics* starting with the weights of the pre-trained counterparts and to speed-up the initial stage of the training as demonstrated by the experiments.

13.4 Experimental Setup

We validate our approach in the MuJoCo [193] simulated environment customized for the iCub robot in [22]. We train all the policies by adapting the SAC [62] implementation in the Stable-Baselines3 [158] library. For RESPRECT, we consider training hyperparameters as

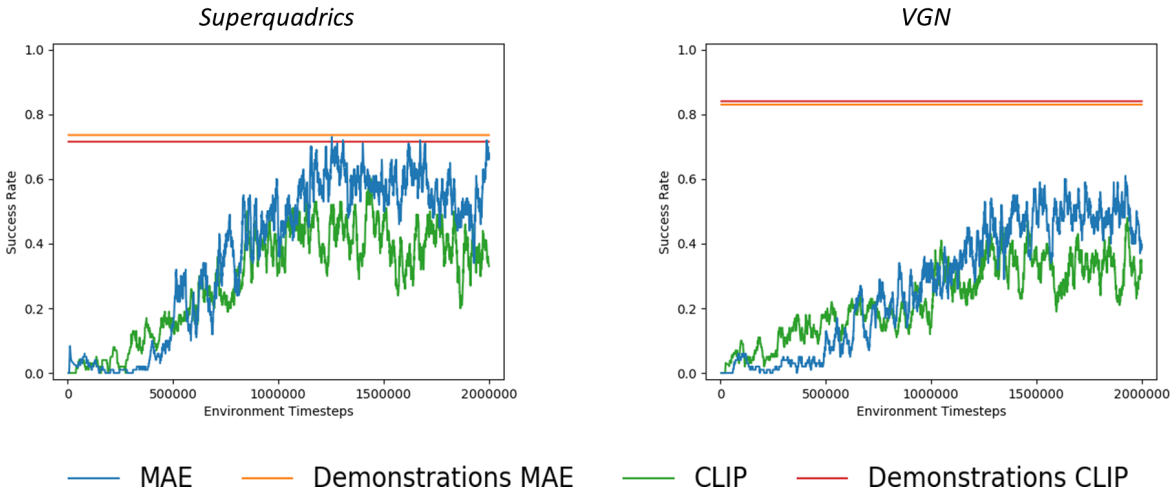


Figure 13.2 We compare the success rate obtained with different visual backbones (MAE and CLIP) when learning with G-PAYN [22] on the MSO dataset for $2M$ timesteps. We report in separate plots the cases in which we use *Superquadrics* and VGN for the initial grasp pose synthesis. We also report the average success rate of the *Demonstrations* used to initialize the G-PAYN replay buffer. Note that they slightly differ for CLIP and MAE due to the random initialization of each episode to collect demonstrations.

the ones in [22], but we increase the number of gradient steps to 10 and we set the initial entropy coefficient for SAC [62] to 0.01. We provide a complete overview of the training hyperparameters in Tab. 13.1.

Both for RESPRECT and the baselines (see Sec. 13.5.1), we improve the visual feature extractor with respect to [22] by replacing the pre-trained *ViT-B/32* CLIP [156] model with the pre-trained *ViT-Large* model of the MAE in [157] and we increase the number of hidden units in the 2 layers of the SAC [62] *Actor* and *Critics* to 2048. The reason for this change is that the *ViT-Large* model of the MAE led to higher success rate of the pre-trained policy. In Fig. 13.2 we compare G-PAYN trained on MSO with the two different feature extractors.

13.4.1 Real Robot Setup

We deploy our method on the real iCub[2] [123] humanoid. The robot is equipped with an *Intel(R) RealSense D405* on a headset for the acquisition of RGB images and depth information (the latter is not used by the DRL algorithm but only during the approach phase to compute the initial grasp pose). We rely on the YARP [122] middleware for the

[2]We run the module for training the grasping policy on a machine equipped with an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, and a single NVIDIA RTX 2080 Ti.

implementation and the communication between the different modules. For policy training, we adapt some components of the simulated training pipeline:

- We use as input RGB image to the MAE the central crop of size 320×240 of the image acquired by the camera of the robot to match the real and simulated visual fields of view.

- We adapt some components of the reward function and some of the terminal conditions, since the precise position of the target object is difficult to obtain on the real robot. Specifically, to reward the position of the object along the axis perpendicular to the table (r_{object_height} in [22]), we consider the z-component of the position of the end-effector of the robot, once it has touched the object with at least two fingers. While this is sufficient to evaluate whether an episode ends positively (i.e. the object has been grasped), it does not allow to determine failure cases when the object falls off the table or moves away from its initial condition. We overcome this problem by manually sending a notification to the learning module.

- We assume that a finger is in contact with the object when the tactile sensors mounted on the fingertip is triggered. If a fingertip is in contact with other parts of the environment (e.g. the table or other fingers in possibly dangerous configurations) we consider this as a possibly unsafe state for the robot, and we manually terminate the execution of the grasping episode.

- We move the end-effector of the robot both to initialize the grasping (i.e. in a pose spaced $5cm$ from the one estimated by the *Superquadrics*) and during policy execution via a cartesian controller that performs IK and trajectory computation [148]. We set the fixation point of the gaze of the robot to the center of the object, which is randomly placed on the table in a graspable configuration at the beginning of the grasping episode, as it is estimated by the *Superquadrics*.

We refer the reader to the code released together with the paper for the implementation details of RESPRECT on the real iCub.

13.5 Results

To evaluate the effectiveness of our approach, we benchmark our results in the scenarios proposed in [22], using seven objects from the YCB-Video dataset chosen to repre-

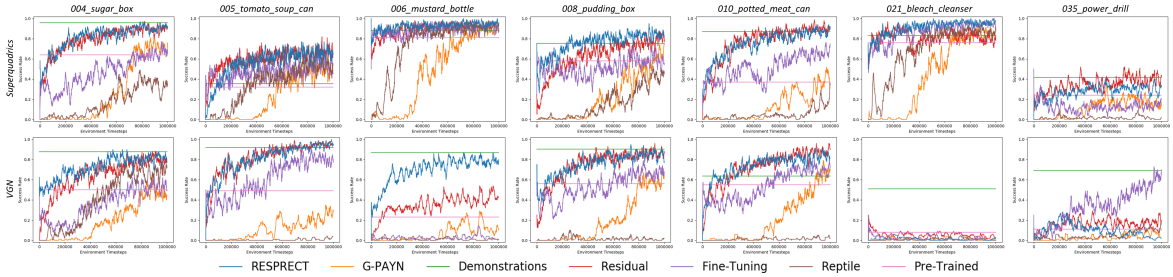


Figure 13.3 **Results**. We compare the success rate achieved by **RESPRECT** to the baselines for 1M environment timesteps. We benchmark the performance over seven YCB-Video objects (on different columns) starting from grasp poses generated either by *Superquadrics* or *VGN* (on different rows).

sent various grasp types (the *004_sugar_box*, the *005_tomato_soup_can*, the *006_mustard_bottle*, the *008_pudding_box*, the *010_potted_meat_can*, the *021_bleach_cleanser*, and the *035_power_drill*). As in [22], for each object, we employ two different methods for the computation of the initial grasp pose: *Superquadrics* and *VGN*. We then evaluate our approach training a policy to grasp a *006_mustard_bottle* and a *021_bleach_cleanser* on the real iCub robot.

13.5.1 Baselines

We benchmark **RESPRECT** (blue line in Fig. 13.3), comparing the success rate for increasing environment timesteps against the following:

- **G-PAYN** (orange line in Fig. 13.3): this is the approach proposed in [22]. For a fair comparison, we train from scratch **G-PAYN** on the considered objects using the pre-trained MAE [157] as feature extractor, and we increase the dimension of the two fully connected layers in the SAC *Actor* and *Critics* networks to 2048.

- **Demonstrations** (green line in Fig. 13.3): this is the pipeline proposed in [22] that is used to collect the demonstrations for the training of **G-PAYN**. We report the success rate of the demonstrations collected to fill the initial replay buffer in the **G-PAYN** experiment.

- **Residual** (red line in Fig. 13.3): this is a similar approach to **RESPRECT**, but we do not initialize the SAC *Critics* of the residual policy with the weights of the SAC instance pre-trained on MSO. We provide an overview of this approach in App. IV in Sec. 13.11. Note that **Residual** is a contribution itself over pre-existing methods that

rely on classical base controllers, which, to the best of our knowledge are unavailable for the considered multi-fingered grasping task.

- **Fine-Tuning** (violet line in Fig. 13.3): we start from the policy pre-trained with **G-PAYN** on the MSO dataset and we fine-tune the fully connected layers in the *Actor* and *Critics* on the considered YCB-Video object. Differently from **RESPRECT** and **Residual**, we perform one gradient step at each training iteration. In App. I in Sec. 13.8, we compare the obtained success rate to the one achieved when performing 10 gradient steps.

- **Reptile** (brown line in Fig. 13.3): this is an adaptation for the SAC [62] algorithm of the meta learning approach proposed in [135]. We modify the original *Reptile* algorithm as in [225], filling the initial replay buffers for the pre-training on MSO as in **G-PAYN**. We chose this method because, according to the results shown in [225], it is the most competitive among the considered MetaRL baselines on the robotic benchmark RLBench [71]. As for **Fine-Tuning**, we perform one gradient step at each training iteration (see App. II in Sec. 13.9).

- **Pre-Trained** (pink line in Fig. 13.3): we compute the success rate achieved by the policy pre-trained on MSO over 100 randomly initialized episodes.

13.5.2 Simulation Results

Results in Fig. 13.3 show that **RESPRECT** and **Residual** constantly outperform the success rate obtained with **Pre-Trained**. This demonstrates the effectiveness of the proposed RRL approach.

RESPRECT manages to achieve in one million environment timesteps a comparable success rate as **G-PAYN** when the latter is trained for five million timesteps (we refer the reader to the results in App. III in Sec. 13.10 for a comparison with the full training of **G-PAYN**), and outperforms it when the training is stopped after one million timesteps. We also show that our approach manages to achieve a comparable success rate as the **Demonstrations** baseline in twelve experiments out of fourteen, outperforming it in seven task instances.

Compared to fine-tuning and MetaRL based approaches for fast task adaptation of a pre-trained policy, overall, **RESPRECT** performs much better than **Fine-Tuning** and **Reptile**. The only exception is represented by the *035_power_drill+VGN* experiment, where **Fine-Tuning** outperforms all the other considered methods, and the full training of **G-PAYN**. In those cases in which the baselines achieve a similar success rate to **RESPRECT**, they

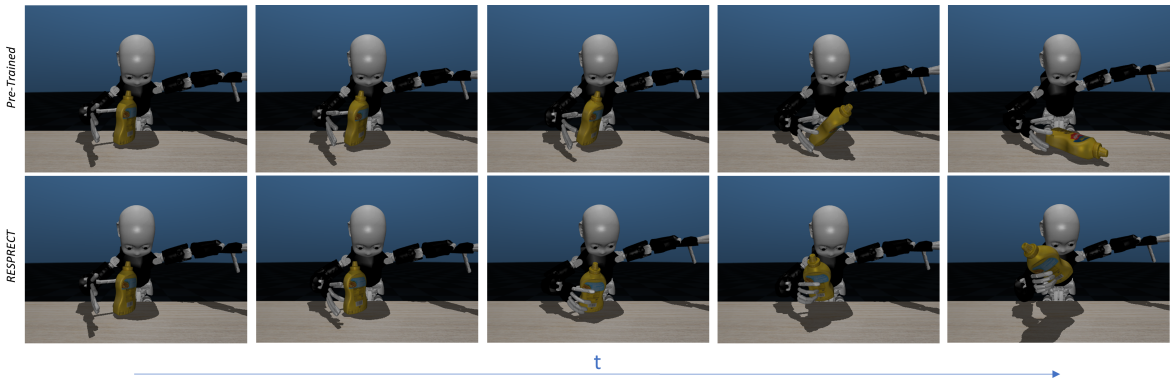


Figure 13.4 Qualitative evaluation of the proposed RESPRECT. We compare it to the pre-trained policy in the same experiment. We show how the residual output of RESPRECT allows to solve the task.

require a larger number of timesteps, see for instance the *010_potted_meat_can+VGN* and the *004_sugar_box+VGN* experiments.

In most of the experiments, for example the *004_sugar_box+VGN*, the success rate of **RESPRECT** in the initial training timesteps has a steeper slope than **Residual**, which is crucial to speed-up the training procedure. Moreover, we noticed that **Residual** tends to have higher *Critics* losses, which may lead to training instability. This occurs, for example in the *021_bleach_cleanser+Superquadrics* experiment, where there is a performance drop after $\sim 300k$ timesteps.

In Fig. 13.4, we show a qualitative evaluation of RESPRECT in the iCub simulated environment. We report an exemplar sequence in which the residual policy successfully grasps the target object, while the pre-trained policy fails starting from the same object configuration.

13.5.3 Real Robot Results

We train RESPRECT to grasp the *006_mustard_bottle* and the *021_bleach_cleanser* starting from grasp poses given by *Superquadrics* on the real iCub humanoid. For these experiments, we consider as base policy the same policy pre-trained on the simulated MSO dataset used for the experiments in simulation. In Fig. 13.5, we report the success rate for increasing robot training time. Results show that after ~ 2.5 hours ($\sim 10k$ timesteps) and ~ 30 minutes ($\sim 1.5k$ timesteps), the robot manages to successfully (with success rate ~ 0.9 and ~ 0.8) grasp the *006_mustard_bottle* and the *021_bleach_cleanser*. In Fig. 13.6, we show a successful grasping sequence during the *006_mustard_bottle* training. We refer the reader to the

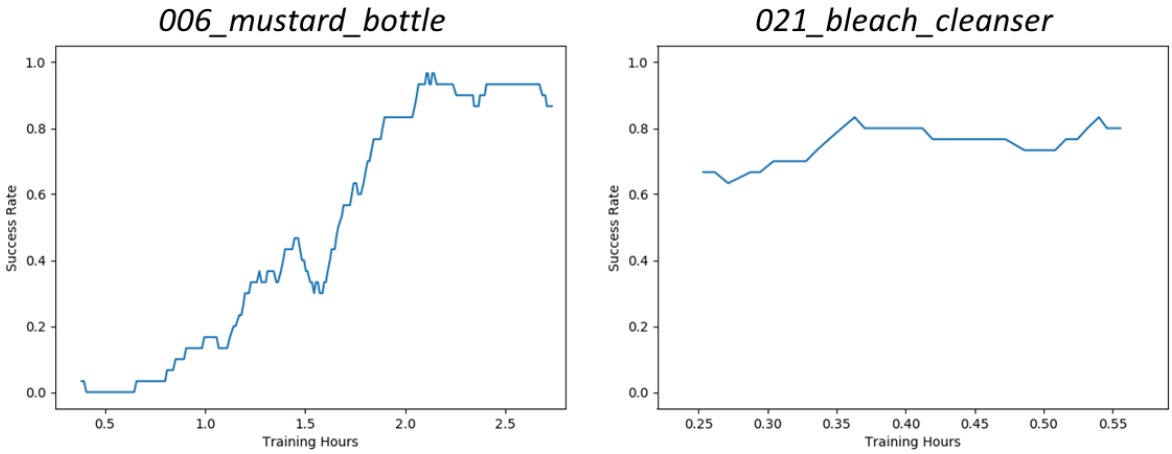


Figure 13.5 RESPRECT success rate (averaged over the last 30 training episodes) for increasing training time on the real iCub robot.

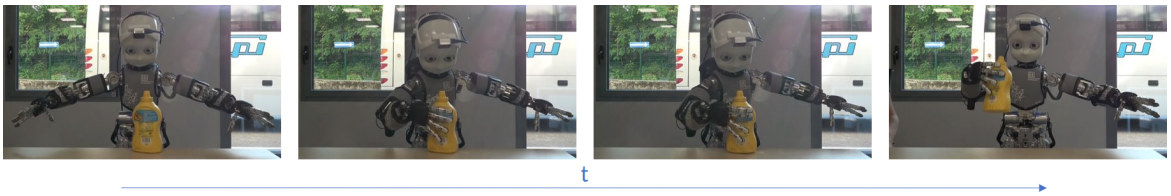


Figure 13.6 Object grasping with the iCub humanoid robot. In this exemplar sequence the policy was trained to grasp the *006_mustard_bottle* with RESPRECT on the real robot.

video submitted as supplementary material[3] which shows two successful grasps on the two considered objects and illustrates the whole training process for the *006_mustard_bottle*.

13.6 Limitations

With the proposed method, we managed to speed-up the training for a multi-fingered grasping task by a factor of 5, while also removing the need for task demonstrations. However, training a model for grasping a single object for $1M$ timesteps still requires a considerable amount of time, especially if these must be executed on the real robot.

From a qualitative analysis of the residual policies, we noticed that they struggle to react to failures during grasp execution. We believe that this is the reason why their overall success rate is comparable to the open-loop pipeline **Demonstrations**. We plan to further improve the reward function, adding components for adapting the position of the fingers once grasp failure is detected. Moreover, in the current problem setting, the observation of

[3] https://youtu.be/JRsBLVclhpg

the environment comprises an estimate of the initial position of the object, which is kept constant throughout the grasping episode. We plan to integrate a class-agnostic tracker to keep updating the estimated position of the object. This can help obtaining more reactive grasping policies.

Finally, while we show that RESPRECT does not need a hand-tuned controller, and that this has some advantages with respect to the state-of-the-art, it is fair to say that our method requires a suitable base policy pre-trained with an *Actor-Critic* DRL algorithm, such as SAC [62] or G-PAYN [22].

13.7 Conclusion

Grasping with multi-fingered robotic hands is an important task for dexterous manipulation. State-of-the art DRL approaches that tackle this problem suffer from data-inefficiency and are difficult to deploy on real robots. Some recent works propose to train a policy only with simulated data, and to transfer the trained policy on the real robot without any adaptation. However, these approaches are highly dependent on the quality of the simulated environment, and may not be suitable for those cases in which there is a large sim-to-real gap. In addition, these approaches are intrinsically off-line and do not allow the robot to adapt after its deployment. In this perspective, we propose RESPRECT, with the aim of speeding-up the training of DRL policies to grasp novel objects. The proposed approach learns a residual policy for the object at hand on top of a policy pre-trained on a different set of objects. In contrast to existing RRL methods that leverage model-based controllers, we employ a pre-trained policy. This allows to use the weights of the latter to warm start some components of the residual policy to significantly speed-up the training.

We show that RESPRECT achieves a comparable success rate as G-PAYN [22] in a fraction of the training timesteps and without using task demonstrations. Moreover, RESPRECT outperforms two fine-tuning and MetaRL baselines for adaptation of a pre-trained policy on a new target object both in terms of success rate and training steps required to achieve comparable performance.

Finally, we deploy the proposed RESPRECT on the real iCub [123] humanoid, showing that it is possible to obtain a policy that is trained directly on the real robot.

As a future work, we plan to improve the grasping pipeline to obtain policies which are more reactive to failures.

13.8 Appendix I

In Fig. 13.7, we compare the success rate achieved by **Fine-Tuning** updating the policies for one and ten gradient steps at each training iteration.

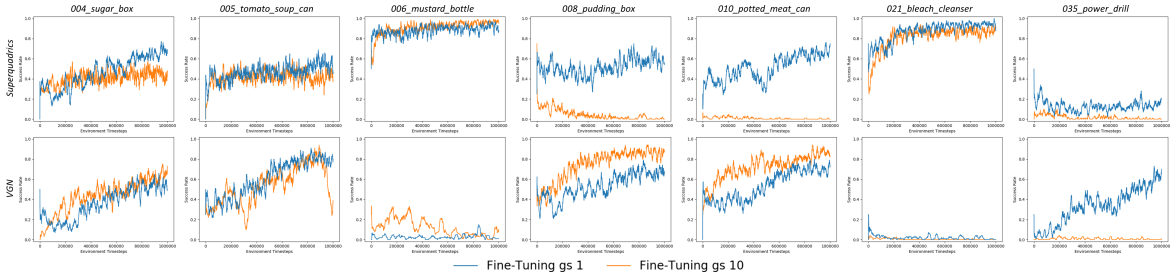


Figure 13.7 We evaluate **Fine-Tuning** considering one and ten gradient steps at each training timestep.

13.9 Appendix II

In Fig. 13.8, we evaluate **Reptile** updating the policies for one and ten gradient steps at each training iteration.

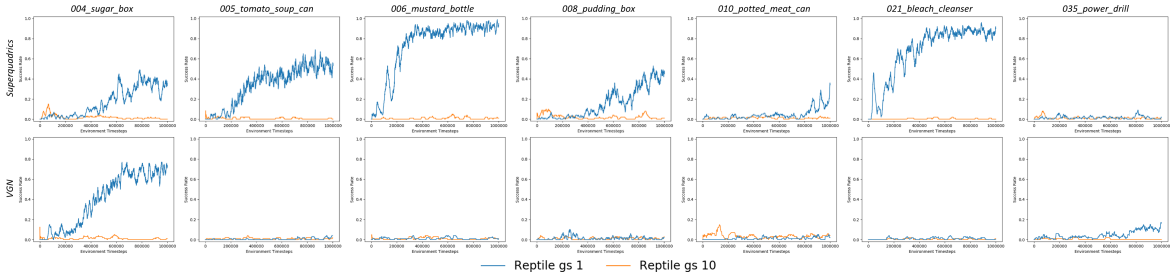


Figure 13.8 We evaluate **Reptile** considering one and ten gradient steps at each training timestep.

13.10 Appendix III

In Fig. 13.9, we compare the success rate achieved by **G-PAYN** trained with different visual backbones for $5M$ environment timesteps.

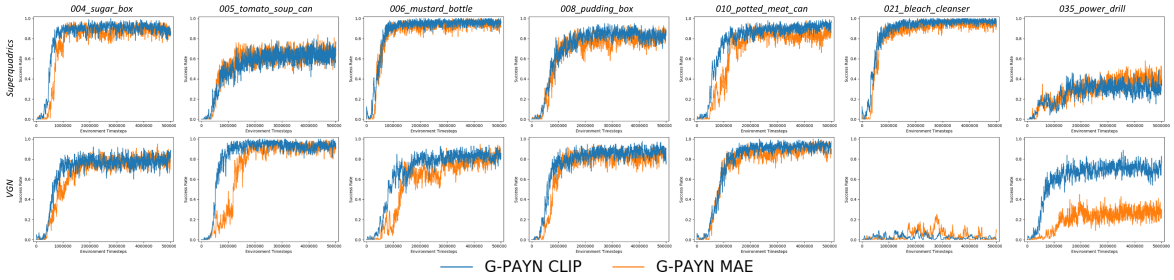


Figure 13.9 We evaluate **G-PAYN** trained with different visual backbones (MAE and CLIP) for $5M$ environment timesteps.

13.11 Appendix IV

In Fig. 13.10, we overview the **Residual** approach used to compare results obtained with **RESPRECT** in Sec. 13.5.

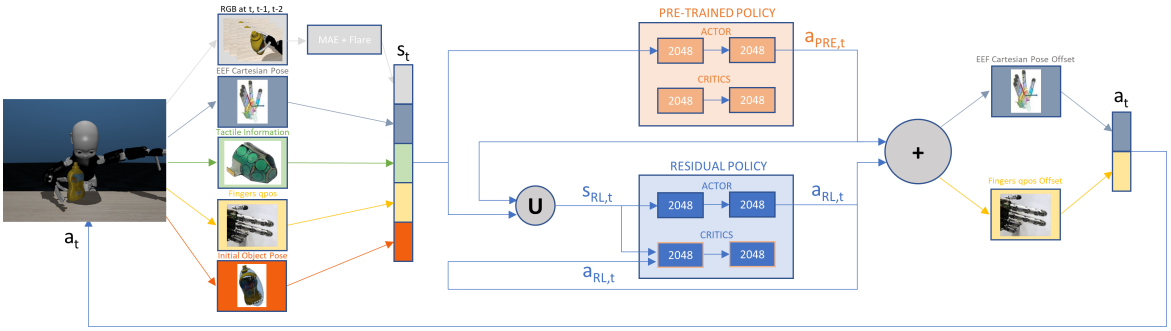


Figure 13.10 **Residual** overview. We compute state s_t from RGB images at timesteps t, $t-1$ and $t-2$, end-effector cartesian pose, tactile information and finger joint poses. We compute action a_t (composed of cartesian offsets for the end-effector and finger joint offsets) combining the outputs $a_{PRE,t}$ of the pre-trained policy and $a_{RL,t}$ of the residual policy. Note that $a_{RL,t}$ is the output of the residual policy *Actor*, given the concatenation of s_t and $a_{PRE,t}$ into $s_{RL,t}$. We train only the two 2048-dimensional fully connected layers in the residual *Actor* and *Critics*. For the sake of clarity, we do not report the input of the *Critics* in the pre-trained policy, and the output of both the *Critics* being the same as the ones in SAC [62].

Chapter 14

LHManip: A Dataset for Long-Horizon Language-Grounded Manipulation Tasks in Cluttered Tabletop Environments

Federico Ceola, Lorenzo Natale, Niko Sünderhauf and Krishan Rana

ABSTRACT

Instructing a robot to complete an everyday task within our homes has been a long-standing challenge for robotics. While recent progress in language-conditioned imitation learning and offline reinforcement learning has demonstrated impressive performance across a wide range of tasks, they are typically limited to short-horizon tasks – not reflective of those a home robot would be expected to complete. While existing architectures have the potential to learn these desired behaviours, the lack of the necessary long-horizon, multi-step datasets for real robotic systems poses a significant challenge. To this end, we present the *Long-Horizon Manipulation* (*LHManip*) dataset comprising 200 episodes, demonstrating 20 different manipulation tasks via real robot teleoperation. The tasks entail multiple sub-tasks, including grasping, pushing, stacking and throwing objects in highly cluttered environments. Each task is paired with a natural language instruction and multi-camera viewpoints for point-cloud or NeRF reconstruction. In total, the dataset comprises 176,278 observation-action pairs which form part of the Open X-Embodiment dataset. The full *LHManip* dataset is made publicly available here.

Figure 14.1 Robot and environment setup used for data collection.

14.1 Introduction

Solving long-horizon manipulation tasks is crucial for addressing real-world applicability of robotic problems. Many practical tasks and activities, such as meal preparation, room cleaning, or workspace organization, involve a sequence of actions performed over an extended period. These tasks are inherently more complex than the short-term ones, as they require robots not only to manipulate objects but also to plan and execute actions across multiple steps. Long-horizon manipulation datasets have potential to allow the development of algorithms capable of generalizing across diverse scenarios, adapting to new settings, and addressing the challenges posed by tasks that require several steps to be executed.

Long-horizon manipulation tasks bring forth the importance of perceptual skills in robotic systems. While requiring to execute low-level control policies to solve the sub-tasks in which they are decomposed, these tasks also require high-level planning and reasoning capabilities. Furthermore, as robots become increasingly integrated into human-centered settings, they will be required to understand and follow natural language instructions for such tasks. Understanding natural language instructions offers the opportunity to develop robotic systems that are capable of autonomously learning, generalizing and adapting to novel settings and environments, rather than being explicitly programmed for each task [181, 231].

The wide availability of datasets for short-horizon manipulation tasks fostered the development of effective learning-based approaches for such tasks [92, 77]. In contrast, the

robotics literature lacks real-world datasets for long-horizon tasks. With *LHManip*, our objective is to address this existing gap in the robotics literature, providing data to develop novel approaches to solve real-world long-horizon manipulation tasks, benchmark existing methods for such tasks evaluated only in simulation [218] on real data, and evaluate generalization properties of state-of-the-art approaches [27]. Our dataset consists of 20 tabletop manipulation tasks involving 33 everyday objects. For each of these tasks, we provide a natural language description and 10 different demonstrations collected via teleoperation. Different demonstrations of the same tasks either involve manipulation of different object instances (e.g. objects with different texture or size) or consider different environment conditions (e.g. different distractors on the table or different initial configurations of the objects involved in the tasks). For each demonstration, we provide visual *RGB* and *depth* observations from a wrist-mounted and two static cameras, and robot proprioceptive information. Furthermore, for each timestep of the episode, we provide the cartesian displacement of the end-effector of the robot and the position offset applied to the gripper. Fig. 14.1 shows a visual observation in *LHManip* from one of the two external static cameras. This dataset forms part of the larger effort by the Open X-Embodiment collaboration project [138], and this dataset paper serves to provide full details of our contribution.

14.2 Related Work

Long-horizon manipulation tasks are challenging robotics problems because they require both high-level reasoning capabilities to decompose the tasks in sequences of sub-tasks and low-level reasoning to solve the short-horizon subtasks. Existing approaches based on *Hierarchical Reinforcement Learning* [130, 218] have been extensively studied to solve long-horizon tasks, but their application in real robotic tasks is hampered by the need of huge amounts of real training data. Other approaches solve long-horizon tasks with methods based on *Imitation Learning* [117], combinations of *Task and Motion Planning* (*TAMP*) and *Reinforcement Learning* (*RL*) [32], or skills learning with *Large Language Models* (*LLM*) [223], and would benefit from the availability of real-world data for training.

While it has been shown in the literature the benefit of using real robotic datasets to learn short-horizon manipulation tasks such as grasping [92], pick and place [14], pouring and scooping [227], object pushing [34], or combinations of different tasks [199], datasets for long-horizon tasks are mainly provided in simulated environments. For example, the IKEA furniture assembly environment [91] provides simulated environments for assembly tasks that require long-horizon manipulation skills. CALVIN [121], instead, provides a dataset

and a benchmark for language-guided long-horizon tasks with the aim of evaluating robot capabilities of learning new skills. This differs from the ALFRED [182] benchmark, that combines seven predefined skills described through language instructions for navigation and manipulation tasks. LoHoRavens [224], instead, is a simulated benchmark composed of ten long-horizon language-conditioned tasks that require at least five pick-and-place steps to be achieved.

To overcome the limitations of learning robotic tasks in simulated environments, FurnitureBench [68] proposes a benchmark for long-horizon assembly tasks, such as lamp screwing and assembling table legs. While providing a huge amount of real teleoperated data (5100 demonstrations), the considered environments are constrained to tabletop settings with 3D-printed objects with multiple markers attached on them. The dataset used to benchmark results in [166], instead, composes short-horizon language-guided tasks demonstrations into long-horizon tasks in constrained tabletop environments. With our dataset, we aim at overcoming these limitations and to provide a language-grounded dataset to perform cluttered tabletop manipulation tasks on everyday objects, relying only on a single language description for task description, multiple RGB-D observations of the environment and the proprioceptive state of the robot.

14.3 LHManip

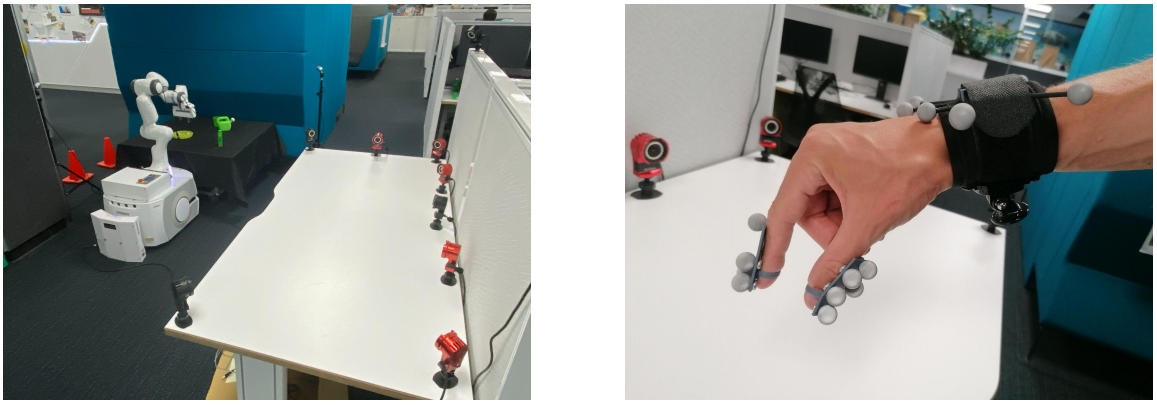
14.3.1 Experimental Set-Up and Data Collection

We collect data via teleoperation, tracking the movements of a human operator via 10 OptiTrack *Motion Capture* (*MoCap*) cameras. Fig. 14.2 (a) shows the set-up. We equip the operator with three different sets of markers to track their movement in the cartesian space of the wrist, used to move the robot end-effector, and to measure the distance between the human thumb and index fingers, used to measure the robot gripper aperture. Fig. 14.2 (b) shows the sets of markers used to capture the motion of the operator.

We perform the tasks on a Franka Panda 7-DoF arm [63] mounted on a LD-60 Omron mobile base[1], keeping the mobile base fixed throughout data collection.

We teleoperate the end-effector of the Panda arm relying on the *servo* implementation of the *armer*[2] library. Specifically, we move the robot to mimic the human movement in the x, y, z position coordinates and the *yaw* orientation of the end-effector, i.e. the rotation around

[1]https://www.ia.omron.com/products/family/3664/dimension.html
[2]https://github.com/qcr/armer



(a) (b)

Figure 14.2 (a) Motion capture and robot setup. (b) The robot was teleoperated by a human operator equipped with a motion capture system for hand gestures and movements detection in the 3D space.

Figure 14.3 Sub-tasks decomposition of a sequence from the *Place the bowl on the plate and the cup in the bowl matching the color* task.

Figure 14.4 Tasks variations: we consider different plate-bowl colors for the *Place the bowls on the appropriate plates* task (left) and different plates for the *Dry the plate* task (right).

the axis perpendicular to the floor. Furthermore, we perform position control of the gripper joints, to mimic the fingers aperture performed by the operator.

We perform all the tasks in the dataset in the cluttered tabletop setting shown in Figs. 14.1, 14.3, and 14.4. We record proprioceptive information from the robot, and acquire visual and depth information from the environment. Specifically, we acquire RGB and depth information from an *Intel(R) RealSense D435* mounted on the wrist of the robot and from two external *Intel(R) RealSense D455*. We refer the reader to the *Observation and Action Space* section for a detailed description of the information provided with the dataset.

Task	Success Condition
Clean the pan.	The robot picks the sponge and performs a movement on the pan.
Cook the capsicum and place it on a plate.	The robot picks the capsicum, puts it in the pan, and then places it on a plate.
Cook the vegetables.	All the vegetables are in the pan.
Dry the plate.	The robot grasps a tissue and performs a movement on the plate.
Hide the teddy bear in the red bowl.	The teddy bear is in the red bowl, and there is an object on it.
Match the cups with the appropriate bowls.	The cups are in the bowls of the same color.
Place the bowl on the plate and the cup in the bowl matching the color.	The bowl is on the plate of the same color, and the cup is in the bowl of the same color.
Place the bowls on the appropriate plates.	The bowls are on the plates of the same color.
Prepare two cups of tea.	The two tea bags are placed in two different cups.
Put a highlighter on each book.	There is a highlighter on each book.
Put the ball in the red pot.	The ball is in the red pot.
Roll the dices in the bowl.	The dices are thrown in the bowl.
Serve the vegetables in different plates.	The vegetables in the pan are moved to different plates.
Set the table.	The fork and the spoon are to the right of the plate, and the cup is beyond the plate from the robot's point of view.
Sort the balls from left to right in order of size.	The balls are sorted in decreasing order from left to right in the robot's point of view.
Stack green blocks.	The three green blocks are stacked.
Stack the bowls.	The three bowls on the table are stacked.
Stack the cups.	The three cups on the table are stacked.
Throw away the rubbish paper.	The rubbish paper is in the trash bin.
Water the potted plant and put the can on the plate.	The robot performs a movement with the can close to the potted plant and then puts the can on the plate.

Table 14.1 Tasks Overview. We provide the list of tasks in the dataset, specifying the conditions that must be met to consider the task execution successful.

14.3.2 Dataset

The proposed dataset consists of 20 long-horizon tabletop manipulation tasks. In the following we describe the tasks and the information that we make available to the users.

***LHManip* Items**		
Ball	Block	Book
Bowl	Capsicum	Cob
Cup	Dice	Eggplant
Fennel	Fork	Highlighter
Ladle	Lizard	Marker
Mug	Pan	Plate
Plug Adapter	Plush Dog	Pot
Potted Plant	Rubbish Paper	Sippy Cup
Spatula	Sponge	Spoon
Tea Bag	Teddy Bear	Tissue Box
Trash Bin	Watering Can	Zucchini

Table 14.2 Items considered in the *LHManip* dataset.

Tasks

We report in Tab. 14.1 an overview of the 20 tasks considered in our dataset. The tasks are performed in a highly-cluttered tabletop environment and require the robot to manipulate everyday objects. We provide the list of objects used in the dataset in Tab. 14.2. The considered tasks are composed by at least two sub-tasks that the robot must complete to successfully achieve the task. We report in Fig. 14.3 an exemplar sequence from the *Place the bowl on the plate and the cup in the bowl matching the color* task. As it can be noticed, to achieve the task, the robot needs to perform several sub-tasks: it firstly grasps the orange bowl and places it on the orange plate. Then it picks the orange cup and places it in the orange bowl.

Furthermore, with our dataset we aim at providing data to solve tasks that require high-level reasoning capabilities. In that we specify the task at hand via a natural language instruction, without providing any other low-level environment descriptions (i.e. object details like shape, spatial information, or color). We believe that the robot must be able to infer this information autonomously from the environment. For these reasons, for each task, we provide different variations of either the initial placement of the objects in the environments or variations of the objects involved to achieve the task. Fig. 14.4 shows two exemplar task variations for two different tasks, where the robot is required to solve the same task with different objects.

Observation	**Description**	**Details**
Cameras	Main static *RGB* camera	$640 \times 480 \times 3$
	Main static *Depth* camera	$848 \times 480 \times 1$
	Secondary static *RGB* camera	$640 \times 480 \times 3$
	Secondary static *Depth* camera	$848 \times 480 \times 1$
	Wrist-mounted *RGB* camera	$640 \times 480 \times 3$
	Wrist-mounted *Depth* camera	$848 \times 480 \times 1$
Proprioceptive	End-effector position	(x, y, z) w.r.t. root_frame
	End-effector orientation	(x, y, z, w) quaternion w.r.t. root_frame
	Robot joint angles	7 values in *rad*
	Gripper position	2 values in [0, 0.0404]
	Robot joint velocities	7 values in *rad/s*
Instruction	Natural language instruction	*String*
Action	**Description**	**Details**
Robot Action	End-effector position displacement	(x, y, z) w.r.t. root_frame
	End-effector orientation displacement	(x, y, z, w) quaternion w.r.t. root_frame
	Gripper opening displacement	1 value in [-0.0808, 0.0808]

Table 14.3 Observations and Actions provided in *LHManip*.

Observation and Action Space

LHManip provides a set of visual and proprioceptive information that captures the robot movement and the action that the robot is required to perform via teleoperation. In the dataset, we provide for each timestep observations and actions as reported in Tab. 14.3. We control the robot in a non-blocking mode at 30 *Hz*.

While we control the cartesian end-effector of the robot and the aperture of the gripper, in the dataset, we also provide joint-level information such as joint positions and velocities.

Please note also that, while we provide the full quaternion displacements at each timestep, we control only the rotation of the end-effector around the axis perpendicular to the floor (z quaternion value). Values different from zero in the other coordinates, therefore, aim at correct the error between the real orientation of the end-effector measured via *Forward Kinematics* and the desired null orientation around the x and y axes.

Dataset Access

The *LHManip* dataset is publicly available[3] and can be downloaded as a single *.zip* file. We provide data in *.png* format for *RGB* and *Depth* images, and in *.pkl* files for numerical and

[3]https://www.dropbox.com/scl/fi/6t717h5mo5kyhq521qavb/long_horizon_manipulation_dataset.zip?rlkey=rk4wsxp464x5bt4a8tgz563ne&dl=0

textual information. We provide a *Python* snippet code and the instructions to parse the dataset on *GitHub*[4]. Furthermore, *LHManip* is included as part of the larger dataset released in the *Open X-Embodiment* project [138]. We release additional depth and the unprocessed sensory data together with this white paper, as well as the code to preprocess this dataset[5] back into the desired *RLDS*[6] data format as required by the *Open X-Embodiment* project.

14.4 Conclusion

The resolution of long-horizon tasks is crucial for integrating robots performing everyday tasks in our homes. Motivated by the success of learning-based approaches from short-term manipulation datasets, we presented *LHManip*, a dataset for long-horizon robotic manipulation tasks, with the aim of addressing the current gap in the literature where such datasets are lacking.

In the perspective of developing robots that can interact with humans in everyday environments, we believe that they must possess the ability to address long-horizon tasks based on a single high-level natural instruction, relying solely on visual and proprioceptive feedback. Existing real datasets are either constrained to simplified environments or have long-horizon instructions composed of short-horizon task descriptions. *LHManip*, instead, presents several challenges, such as natural language instruction understanding, visual perception of the environment in the presence of changing challenging conditions, and learning of low-level control policies for sub-tasks execution.

We hope that our work will motivate the need for more datasets, benchmarks and methodologies to learn long-horizon manipulation tasks, thereby taking a significant stride toward the integration of robots into human-centered environments.

[4]https://github.com/fedeceola/LHManip
[5]https://github.com/fedeceola/rlds_dataset_builder
[6]https://github.com/google-research/rlds

Part V

Conclusion

Chapter 15

Conclusion

In this Thesis, I presented new methods for the visual problem of instance segmentation and multi-fingered grasping, and a dataset for long-horizon language-guided manipulation tasks. These have been proposed with the aim of enabling robots to autonomously operate in everyday environments. To satisfy this requirement, robots need to rapidly adapt to dynamically changing unconstrained environments and to solve long-horizon tasks with minimal human supervision, such as a single high-level language instruction.

I started my project by proposing two methods for fast learning of instance segmentation also in presence of domain shifts. They rely on a state-of-the-art *CNN*-based architecture for feature extraction and train kernel-based classifiers for learning novel classes and for adaptation to different visual domains. The proposed approaches achieve similar accuracy to the state-of-the-art model used for feature extraction, requiring a fraction of the training time, and making the deployment of such methods feasible for robotic applications that require fast adaptation. In this part of the project, I focused on speeding-up the training, without reducing the amount of data required for optimization. The proposal of a few-shot approach [98] could further accelerate the training. Another interesting future direction is the extension of the proposed approaches by integrating them with the latest transformer-based methods for image segmentation. These latter, e.g. SAM [82] or approaches based on *Visual Language Models* like CLIPSeg [110], provide impressive zero-shot performance. However, learning to segment specific objects may still be a requirement for several robotic applications. Extension of the proposed methods building on top of models for zero-shot segmentation is a research direction toward the deployment of robotic systems that are both general and adaptable to new environment conditions.

Then, I worked on multi-fingered grasping proposing *G-PAYN*, a DRL from demonstration method, and *RESPRECT*, an algorithm based on RRL, to learn the task. *G-PAYN* learns to

grasp objects from visual, tactile and proprioceptive information in simulation. It achieves a higher success rate than all the considered DRL baselines, and can be deployed on the real robot without adapting state and action spaces. *RESPRECT* learns a residual policy on top of a base policy pre-trained with *G-PAYN*. It leverages some components of the base policy to further speed-up the residual training, and can be used for training multi-fingered grasping policies on the real robot. Experimental evaluation of the two approaches show the importance of starting the grasping from a suitable pose for the end-effector of the robot and the development of new algorithms for generation of grasping candidates could improve the performance of the proposed methods. Moreover, improving the reward function considered for training, could make the policies more reactive to failures. In the perspective of learning downstream tasks, such as in-hand manipulation [29], or to perform tasks in collaboration with humans, the integration of the proposed approaches with affordance models [2] could further improve *G-PAYN* and *RESPRECT*. Finally, the deployment of large-scale feature extractor to compute spatial information from visual observations of the environment could help improving the grasping policies. The deployment of such methods is currently limited [221], and extracting features from NeRF [125] reconstructions of the environment, or methods that embed semantic features into NeRF models, e.g. LERF [79], could be an interesting research direction.

Finally, I collected *LHManip*, with the aim of providing a dataset composed of real data for long-horizon manipulation tasks that a robot would be required to accomplish in everyday environments. Data have been collected via teleoperation and each task is described with a natural level instruction. *LHManip* is part of the *Open X-Embodiment* [138] dataset and can be used for co-training or fine-tuning *Behavior Cloning* policies as in [138] for a large number of manipulation tasks. *LHManip* can also be used to validate existing approaches for long-horizon manipulation tasks benchmarked only in simulation and to evaluate long-horizon reasoning capabilities of *Large Language Models*. An interesting research direction that can be pursued leveraging *LHManip* as training set is the development of an *off-line RL* approach for long-horizon tasks to overcome the limitations of the models trained with *Behavior Cloning*.

In conclusion, the methodologies and the dataset presented in this Thesis explored different aspects of the problem of enabling robot to operate in everyday environments. Approaches proposed for the tasks of instance segmentation and multi-fingered grasping have been designed to speed-up training of specific objects, but can benefit from the deployment of methods trained on large and general datasets. Integration of pre-trained general models

and methods for fast adaptation of such models could be the key for the deployment of autonomous, flexible and efficient robotic systems in real-world environments.

References

[1] Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

[2] Agarwal, A., Uppal, S., Shaw, K., and Pathak, D. (2023). Dexterous functional grasping. In Tan, J., Toussaint, M., and Darvish, K., editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 3453–3467. PMLR.

[3] Alakuijala, M., Dulac-Arnold, G., Mairal, J., Ponce, J., and Schmid, C. (2021). Residual reinforcement learning from demonstrations. *arXiv preprint arXiv:2106.08050*.

[4] Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2020). Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20.

[5] Bai, M. and Urtasun, R. (2017). Deep watershed transform for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5221–5229.

[6] Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684.

[7] Bicchi, A. and Kumar, V. (2000). Robotic grasping and contact: A review. In *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, volume 1, pages 348–353. IEEE.

[8] Bohg, J., Morales, A., Asfour, T., and Kragic, D. (2014). Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309.

[9] Bolya, D., Zhou, C., Xiao, F., and Lee, Y. J. (2019). YOLACT: Real-time instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9157–9166.

[10] Bousmalis, K., Vezzani, G., Rao, D., Devin, C. M., Lee, A. X., Villalonga, M. B., Davchev, T., Zhou, Y., Gupta, A., Raju, A., Laurens, A., Fantacci, C., Dalibard, V., Zambelli, M., Martins, M. F., Pevceviciute, R., Blokzijl, M., Denil, M., Batchelor, N., Lampe, T., Parisotto, E., Zolna, K., Reed, S., Colmenarejo, S. G., Scholz, J., Abdolmaleki, A., Groth, O., Regli, J.-B., Sushkov, O., Rothörl, T., Chen, J. E., Aytar, Y., Barker, D., Ortiz, J., Riedmiller, M., Springenberg, J. T., Hadsell, R., Nori, F., and Heess, N. (2024).

Robocat: A self-improving generalist agent for robotic manipulation. *Transactions on Machine Learning Research*.

[11] Breyer, M., Chung, J. J., Ott, L., Roland, S., and Juan, N. (2020). Volumetric grasping network: Real-time 6 dof grasp detection in clutter. In *Conference on Robot Learning*.

[12] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

[13] Brogårdh, T. (2007). Present and future robot control development—an industrial perspective. *Annual Reviews in Control*, 31(1):69–79.

[14] Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jackson, T., Jesmonth, S., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Leal, I., Lee, K.-H., Levine, S., Lu, Y., Malla, U., Manjunath, D., Mordatch, I., Nachum, O., Parada, C., Peralta, J., Perez, E., Pertsch, K., Quiambao, J., Rao, K., Ryoo, M., Salazar, G., Sanketi, P., Sayed, K., Singh, J., Sontakke, S., Stone, A., Tan, C., Tran, H., Vanhoucke, V., Vega, S., Vuong, Q., Xia, F., Xiao, T., Xu, P., Xu, S., Yu, T., and Zitkovich, B. (2022). Rt-1: Robotics transformer for real-world control at scale. In *arXiv preprint arXiv:2212.06817*.

[15] Caelles, S., Pont-Tuset, J., Perazzi, F., Montes, A., Maninis, K.-K., and Van Gool, L. (2019). The 2019 davis challenge on vos: Unsupervised multi-object segmentation. *arXiv:1905.00737*.

[16] Calli, B., Singh, A., Walsman, A., Srinivasa, S., Abbeel, P., and Dollar, A. M. (2015). The YCB object and model set: Towards common benchmarks for manipulation research. In *2015 international conference on advanced robotics (ICAR)*, pages 510–517. IEEE.

[17] Camoriano, R., Pasquale, G., Ciliberto, C., Natale, L., Rosasco, L., and Metta, G. (2017). Incremental robot learning of new objects with fixed update time. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3207–3214.

[18] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer.

[19] Ceola, F., Maiettini, E., Pasquale, G., Meanti, G., Rosasco, L., and Natale, L. (2022). Learn fast, segment well: Fast object segmentation learning on the icub robot. *IEEE Transactions on Robotics*, 38(5):3154–3172.

[20] Ceola, F., Maiettini, E., Pasquale, G., Rosasco, L., and Natale, L. (2020). Fast region proposal learning for object detection for robotics. *arXiv preprint arXiv:2011.12790*.

[21] Ceola, F., Maiettini, E., Pasquale, G., Rosasco, L., and Natale, L. (2021). Fast object segmentation learning with kernel-based methods for robotics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13581–13588.

[22] Ceola, F., Maiettini, E., Rosasco, L., and Natale, L. (2023a). A grasp pose is all you need: Learning multi-fingered grasping with deep reinforcement learning from vision and touch. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Sysems (IROS)*.

[23] Ceola, F., Natale, L., Sünderhauf, N., and Rana, K. (2023b). Lhmanip: A dataset for long-horizon language-grounded manipulation tasks in cluttered tabletop environments. *arXiv preprint arXiv:2312.12036*.

[24] Ceola, F., Rosasco, L., and Natale, L. (2024). Resprect: Speeding-up multi-fingered grasping with residual reinforcement learning. *IEEE Robotics and Automation Letters*, pages 1–8.

[25] Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*.

[26] Chen, H., Sun, K., Tian, Z., Shen, C., Huang, Y., and Yan, Y. (2020). BlendMask: Top-down meets bottom-up for instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8573–8581.

[27] Chen, L., Bahl, S., and Pathak, D. (2023a). Playfusion: Skill acquisition via diffusion from language-annotated play. In *Conference on Robot Learning*, pages 2012–2029. PMLR.

[28] Chen, S., Tang, W., Xie, P., Yang, W., and Wang, G. (2023b). Efficient heatmap-guided 6-dof grasp detection in cluttered scenes. *IEEE Robotics and Automation Letters*, 8(8):4895–4902.

[29] Chen, T., Xu, J., and Agrawal, P. (2022a). A system for general in-hand object re-orientation. In *Conference on Robot Learning*, pages 297–307. PMLR.

[30] Chen, X., Girshick, R., He, K., and Dollár, P. (2019). TensorMask: A foundation for dense object segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2061–2069.

[31] Chen, Z. Q., Van Wyk, K., Chao, Y.-W., Yang, W., Mousavian, A., Gupta, A., and Fox, D. (2022b). Dextransfer: Real world multi-fingered dexterous grasping with minimal human demonstrations. *arXiv preprint arXiv:2209.14284*.

[32] Cheng, S. and Xu, D. (2023). League: Guided skill learning and abstraction for long-horizon manipulation. *IEEE Robotics and Automation Letters*, 8(10):6451–6458.

[33] Cheng, X., Shi, K., Agarwal, A., and Pathak, D. (2023). Extreme parkour with legged robots. In *Towards Generalist Robots: Learning Paradigms for Scalable Skill Acquisition @ CoRL2023*.

[34] Chi, C., Feng, S., Du, Y., Xu, Z., Cousineau, E., Burchfiel, B., and Song, S. (2023). Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*.

[35] Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., and Wurman, P. R. (2018). Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188.

[36] Dai, J., He, K., Li, Y., Ren, S., and Sun, J. (2016a). Instance-sensitive fully convolutional networks. In *European Conference on Computer Vision*, pages 534–549. Springer.

[37] Dai, J., He, K., and Sun, J. (2016b). Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3150–3158.

[38] Dai, J., Li, Y., He, K., and Sun, J. (2016c). R-FCN: Object detection via region-based fully convolutional networks. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 379–387. Curran Associates, Inc.

[39] Danielczuk, M., Matl, M., Gupta, S., Li, A., Lee, A., Mahler, J., and Goldberg, K. (2019). Segmenting unknown 3d objects from real depth images using Mask R-CNN trained on synthetic data. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7283–7290. IEEE.

[40] Dasari, S., Gupta, A., and Kumar, V. (2022). Learning dexterous manipulation from exemplar object trajectories and pre-grasps. *arXiv preprint arXiv:2209.11221*.

[41] Dass, S., Yapeter, J., Zhang, J., Zhang, J., Pertsch, K., Nikolaidis, S., and Lim, J. J. (2023). Clvr jaco play dataset.

[42] Davchev, T., Luck, K. S., Burke, M., Meier, F., Schaal, S., and Ramamoorthy, S. (2022). Residual learning from demonstration: Adapting dmps for contact-rich manipulation. *IEEE Robotics and Automation Letters*, 7(2):4488–4495.

[43] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[44] Deng, J., Zhang, H., Hu, J., Zhang, X., and Wang, Y. (2023). Class incremental robotic pick-and-place via incremental few-shot object detection. *IEEE Robotics and Automation Letters*, 8(9):5974–5981.

[45] Deng, X., Xiang, Y., Mousavian, A., Eppner, C., Bretl, T., and Fox, D. (2020). Self-supervised 6d object pose estimation for robot manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3665–3671.

[46] Denninger, M., Sundermeyer, M., Winkelbauer, D., Olefir, D., Hodan, T., Youssef Zidan and, M. E., Knauer, M., Katam, H., and Ahsan, L. (2020). Blenderproc: Reducing the reality gap with photorealistic rendering. In *RSS 2020*.

[47] Dong, N., Zhang, Y., Ding, M., and Lee, G. H. (2023). Incremental-detr: Incremental few-shot object detection via self-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 543–551.

[48] Downs, L., Francis, A., Koenig, N., Kinman, B., Hickman, R., Reymann, K., McHugh, T. B., and Vanhoucke, V. (2022). Google scanned objects: A high-quality dataset of 3d scanned household items.

[49] Eitel, A., Hauff, N., and Burgard, W. (2019). Self-supervised transfer learning for instance segmentation through physical interaction. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4020–4026. IEEE.

[50] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338.

[51] Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2013). Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929.

[52] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

[53] Fu, L., Datta, G., Huang, H., Panitch, W. C.-H., Drake, J., Ortiz, J., Mukadam, M., Lambeta, M., Calandra, R., and Goldberg, K. (2024a). A touch, vision, and language dataset for multimodal alignment. *arXiv preprint arXiv:2402.13232*.

[54] Fu, Z., Zhao, T. Z., and Finn, C. (2024b). Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *arXiv*.

[55] Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR.

[56] Fulkerson, B., Vedaldi, A., and Soatto, S. (2009). Class segmentation and object localization with superpixel neighborhoods. In *2009 IEEE 12th International Conference on Computer Vision*, pages 670–677.

[57] Ganea, D. A., Boom, B., and Poppe, R. (2021). Incremental few-shot instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1185–1194.

[58] Gao, N., Shan, Y., Wang, Y., Zhao, X., Yu, Y., Yang, M., and Huang, K. (2019). SSAP: Single-shot instance segmentation with affinity pyramid. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 642–651.

[59] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[60] Grauman, K., Westbury, A., Byrne, E., Chavis, Z., Furnari, A., Girdhar, R., Hamburger, J., Jiang, H., Liu, M., Liu, X., et al. (2022). Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18995–19012.

[61] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018a). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

[62] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018b). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

[63] Haddadin, S., Parusel, S., Johannsmeier, L., Golz, S., Gabl, S., Walch, F., Sabaghian, M., Jähne, C., Hausperger, L., and Haddadin, S. (2022). The franka emika robot: A reference platform for robotics research and education. *IEEE Robotics & Automation Magazine*, 29(2):46–64.

[64] Haldar, S., Pari, J., Rai, A., and Pinto, L. (2023). Teach a Robot to FISH: Versatile Imitation from One Minute of Demonstrations. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea.

[65] Hampali, S., Rad, M., Oberweger, M., and Lepetit, V. (2020). HOnnotate: A method for 3D annotation of hand and object poses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3196–3206.

[66] He, K., Gkioxari, G., Dollár, P., and Girshick, R. B. (2017). Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988.

[67] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[68] Heo, M., Lee, Y., Lee, D., and Lim, J. J. (2023). Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation. *arXiv preprint arXiv:2305.12821*.

[69] Hodaň, T., Sundermeyer, M., Drost, B., Labbé, Y., Brachmann, E., Michel, F., Rother, C., and Matas, J. (2020). BOP challenge 2020 on 6D object localization. *European Conference on Computer Vision Workshops (ECCVW)*.

[70] Huang, Z., Huang, L., Gong, Y., Huang, C., and Wang, X. (2019). Mask Scoring R-CNN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6409–6418.

[71] James, S., Ma, Z., Arrojo, D. R., and Davison, A. J. (2020). Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026.

[72] Jianchao Yang, Kai Yu, Yihong Gong, and Huang, T. (2009). Linear spatial pyramid matching using sparse coding for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1794–1801.

[73] Johannink, T., Bahl, S., Nair, A., Luo, J., Kumar, A., Loskyll, M., Ojea, J. A., Solowjow, E., and Levine, S. (2019). Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE.

[74] Joseph, K. J., Khan, S., Khan, F. S., and Balasubramanian, V. N. (2021). Towards open world object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2021)*.

[75] Joshi, S., Kumra, S., and Sahin, F. (2020). Robotic grasping using deep reinforcement learning. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 1461–1466.

[76] Kakade, S. M. (2001). A natural policy gradient. *Advances in neural information processing systems*, 14.

[77] Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., and Levine, S. (2018). Scalable deep reinforcement learning for vision-based robotic manipulation. In Billard, A., Dragan, A., Peters, J., and Morimoto, J., editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 651–673. PMLR.

[78] Kang, D. and Cho, M. (2022). Integrative few-shot learning for classification and segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9979–9990.

[79] Kerr, J., Kim, C. M., Goldberg, K., Kanazawa, A., and Tancik, M. (2023). Lerf: Language embedded radiance fields. In *International Conference on Computer Vision (ICCV)*.

[80] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

[81] Kirillov, A., Levinkov, E., Andres, B., Savchynskyy, B., and Rother, C. (2017). InstanceCut: from edges to instances with multicut. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5008–5017.

[82] Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., and Girshick, R. (2023). Segment anything. *arXiv:2304.02643*.

[83] Kolesnikov, A., Dosovitskiy, A., Weissenborn, D., Heigold, G., Uszkoreit, J., Beyer, L., Minderer, M., Dehghani, M., Houlsby, N., Gelly, S., Unterthiner, T., and Zhai, X. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*.

[84] Kornblith, S., Shlens, J., and Le, Q. V. (2019). Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[85] Krähenbühl, P. and Koltun, V. (2011). Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in neural information processing systems*, pages 109–117.

[86] Kumar, A., Agarwal, R., Ghosh, D., and Levine, S. (2021). Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*.

[87] Kumar, V., Hermans, T., Fox, D., Birchfield, S., and Tremblay, J. (2019). Contextual reinforcement learning of visuo-tactile multi-fingered grasping policies. *arXiv preprint arXiv:1911.09233*.

[88] Kumar, V. and Todorov, E. (2015). Mujoco haptix: A virtual reality system for hand manipulation. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 657–663.

[89] Kuo, W., Angelova, A., Malik, J., and Lin, T.-Y. (2019). ShapeMask: Learning to segment novel objects by refining shape priors. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9207–9216.

[90] Lee, A. X., Devin, C. M., Zhou, Y., Lampe, T., Bousmalis, K., Springenberg, J. T., Byravan, A., Abdolmaleki, A., Gileadi, N., Khosid, D., Fantacci, C., Chen, J. E., Raju, A., Jeong, R., Neunert, M., Laurens, A., Saliceti, S., Casarini, F., Riedmiller, M., Hadsell, R., and Nori, F. (2021a). Beyond pick-and-place: Tackling robotic stacking of diverse shapes. In *5th Annual Conference on Robot Learning*.

[91] Lee, Y., Hu, E. S., and Lim, J. J. (2021b). Ikea furniture assembly environment for long-horizon complex manipulation tasks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6343–6349.

[92] Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research*, 37(4-5):421–436.

[93] Li, A., Danielczuk, M., and Goldberg, K. (2020a). One-shot shape-based amodal-to-modal instance segmentation. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 1375–1382. IEEE.

[94] Li, F., Zhang, H., Sun, P., Zou, X., Liu, S., Yang, J., Li, C., Zhang, L., and Gao, J. (2023a). Semantic-sam: Segment and recognize anything at any granularity. *arXiv preprint arXiv:2307.04767*.

[95] Li, K., Baron, N., Zhang, X., and Rojas, N. (2022). Efficientgrasp: A unified data-efficient learning to grasp method for multi-fingered robot hands. *IEEE Robotics and Automation Letters*, 7(4):8619–8626.

[96] Li, S., Li, M., Wang, P., and Zhang, L. (2023b). Opensd: Unified open-vocabulary segmentation and detection. *arXiv preprint arXiv:2312.06703*.

[97] Li, S., Zhou, J., Jia, Z., Yeung, D.-Y., and Mason, M. T. (2020b). Learning accurate objectness instance segmentation from photorealistic rendering for robotic manipulation. In Xiao, J., Kröger, T., and Khatib, O., editors, *Proceedings of the 2018 International Symposium on Experimental Robotics*, pages 245–255, Cham. Springer International Publishing.

[98] Li, W., Wang, Z., Yang, X., Dong, C., Tian, P., Qin, T., Huo, J., Shi, Y., Wang, L., Gao, Y., and Luo, J. (2023c). Libfewshot: A comprehensive library for few-shot learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(12):14938–14955.

[99] Li, Y., Qi, H., Dai, J., Ji, X., and Wei, Y. (2017). Fully convolutional instance-aware semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2359–2367.

[100] Li, Y., Wang, G., Ji, X., Xiang, Y., and Fox, D. (2018). Deepim: Deep iterative matching for 6d pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[101] Liang, H., Cong, L., Hendrich, N., Li, S., Sun, F., and Zhang, J. (2021). Multifingered grasping based on multimodal reinforcement learning. *IEEE Robotics and Automation Letters*, 7(2):1174–1181.

[102] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning.

[103] Lin, T., Goyal, P., Girshick, R. B., He, K., and Dollár, P. (2017a). Focal loss for dense object detection. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2999–3007.

[104] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017b). Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125.

[105] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, Zürich. Oral.

[106] Ling, H., Gao, J., Kar, A., Chen, W., and Fidler, S. (2019). Fast interactive object annotation with curve-gcn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5257–5266.

[107] Liu, S., Qi, L., Qin, H., Shi, J., and Jia, J. (2018). Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768.

[108] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

[109] Lucchi, A., Li, Y., Smith, K., and Fua, P. (2012). Structured image segmentation using kernelized features. In Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., and Schmid, C., editors, *Computer Vision – ECCV 2012*, pages 400–413, Berlin, Heidelberg. Springer Berlin Heidelberg.

[110] Lüddecke, T. and Ecker, A. (2022). Image segmentation using text and image prompts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7086–7096.

[111] Luo, J., Xu, C., Geng, X., Feng, G., Fang, K., Tan, L., Schaal, S., and Levine, S. (2024). Multistage cable routing through hierarchical imitation learning. *IEEE Transactions on Robotics*, 40:1476–1491.

[112] Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Aparicio, J., and Goldberg, K. (2017). Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts.

[113] Maiettini, E., Pasquale, G., Rosasco, L., and Natale, L. (2017). Interactive data collection for deep learning object detectors on humanoid robots. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 862–868.

[114] Maiettini, E., Pasquale, G., Rosasco, L., and Natale, L. (2018). Speeding-up object detection training for robotics with falkon. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[115] Maiettini, E., Pasquale, G., Rosasco, L., and Natale, L. (2019). On-line object detection: a robotics challenge. *Autonomous Robots*.

[116] Maltoni, D. and Lomonaco, V. (2019). Continuous learning in single-incremental-task scenarios. *Neural Networks*, 116:56–73.

[117] Mandlekar, A., Xu, D., Martín-Martín, R., Savarese, S., and Fei-Fei, L. (2020). GTI: Learning to Generalize across Long-Horizon Tasks from Human Demonstrations. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA.

[118] Manhardt, F., Kehl, W., Navab, N., and Tombari, F. (2018). Deep model-based 6d pose refinement in rgb. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[119] Margolis, G. B., Fu, X., Ji, Y., and Agrawal, P. (2023). Learning to see physical properties with active sensing motor policies. In Tan, J., Toussaint, M., and Darvish, K., editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2537–2548. PMLR.

[120] Meanti, G., Carratino, L., Rosasco, L., and Rudi, A. (2020). Kernel methods through the roof: handling billions of points efficiently. *arXiv preprint arXiv:2006.10350v1*.

[121] Mees, O., Hermann, L., Rosete-Beas, E., and Burgard, W. (2022). Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334.

[122] Metta, G., Fitzpatrick, P., and Natale, L. (2006). YARP: Yet another robot platform. *International Journal of Advanced Robotics Systems*, 3(1):43–48.

[123] Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., Bernardino, A., and Montesano, L. (2010). The iCub humanoid robot: an open-systems platform for research in cognitive development. *Neural networks : the official journal of the International Neural Network Society*, 23(8-9):1125–34.

[124] Michieli, U. and Zanuttigh, P. (2019). Incremental learning techniques for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*.

[125] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*.

[126] Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., and Terzopoulos, D. (2020). Image segmentation using deep learning: A survey. *arXiv preprint arXiv:2001.05566*.

[127] Minderer, M., Gritsenko, A., and Houlsby, N. (2023). Scaling open-vocabulary object detection. *Advances in Neural Information Processing Systems*, 36.

[128] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *NIPS Deep Learning Workshop*.

[129] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

[130] Nachum, O., Gu, S. S., Lee, H., and Levine, S. (2018). Data-efficient hierarchical reinforcement learning. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[131] Nair, A., Gupta, A., Dalal, M., and Levine, S. (2020). Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*.

[132] Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6292–6299. IEEE.

[133] Newbury, R., Gu, M., Chumbley, L., Mousavian, A., Eppner, C., Leitner, J., Bohg, J., Morales, A., Asfour, T., Kragic, D., Fox, D., and Cosgun, A. (2023). Deep learning approaches to grasp synthesis: A review. *IEEE Transactions on Robotics*, 39(5):3994–4015.

[134] Newswanger, A. and Xu, C. (2017). One-shot video object segmentation with iterative online fine-tuning. *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops*.

[135] Nichol, A., Achiam, J., and Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

[136] Nori, F., Traversaro, S., Eljaik, J., Romano, F., Del Prete, A., and Pucci, D. (2015). icub whole-body control through force regulation on rigid noncoplanar contacts. *Frontiers in Robotics and AI*, 2(6).

[137] Octo Model Team, Ghosh, D., Walke, H., Pertsch, K., Black, K., Mees, O., Dasari, S., Hejna, J., Xu, C., Luo, J., Kreiman, T., Tan, Y., Sadigh, D., Finn, C., and Levine, S. (2023). Octo: An open-source generalist robot policy. https://octo-models.github.io.

[138] Open X-Embodiment Collaboration, Padalkar, A., Pooley, A., Jain, A., Bewley, A., Herzog, A., Irpan, A., Khazatsky, A., Rai, A., Singh, A., Brohan, A., Raffin, A., Wahid, A., Burgess-Limerick, B., Kim, B., Schölkopf, B., Ichter, B., Lu, C., Xu, C., Finn, C., Xu, C., Chi, C., Huang, C., Chan, C., Pan, C., Fu, C., Devin, C., Driess, D., Pathak, D., Shah, D., Büchler, D., Kalashnikov, D., Sadigh, D., Johns, E., Ceola, F., Xia, F., Stulp, F., Zhou, G., Sukhatme, G. S., Salhotra, G., Yan, G., Schiavi, G., Su, H., Fang, H.-S., Shi, H., Amor, H. B., Christensen, H. I., Furuta, H., Walke, H., Fang, H., Mordatch, I., Radosavovic, I., Leal, I., Liang, J., Kim, J., Schneider, J., Hsu, J., Bohg, J., Bingham, J., Wu, J., Wu, J.,

Luo, J., Gu, J., Tan, J., Oh, J., Malik, J., Tompson, J., Yang, J., Lim, J. J., Silvério, J., Han, J., Rao, K., Pertsch, K., Hausman, K., Go, K., Gopalakrishnan, K., Goldberg, K., Byrne, K., Oslund, K., Kawaharazuka, K., Zhang, K., Majd, K., Rana, K., Srinivasan, K., Chen, L. Y., Pinto, L., Tan, L., Ott, L., Lee, L., Tomizuka, M., Du, M., Ahn, M., Zhang, M., Ding, M., Srirama, M. K., Sharma, M., Kim, M. J., Kanazawa, N., Hansen, N., Heess, N., Joshi, N. J., Suenderhauf, N., Palo, N. D., Shafiullah, N. M. M., Mees, O., Kroemer, O., Sanketi, P. R., Wohlhart, P., Xu, P., Sermanet, P., Sundaresan, P., Vuong, Q., Rafailov, R., Tian, R., Doshi, R., Martín-Martín, R., Mendonca, R., Shah, R., Hoque, R., Julian, R., Bustamante, S., Kirmani, S., Levine, S., Moore, S., Bahl, S., Dass, S., Song, S., Xu, S., Haldar, S., Adebola, S., Guist, S., Nasiriany, S., Schaal, S., Welker, S., Tian, S., Dasari, S., Belkhale, S., Osa, T., Harada, T., Matsushima, T., Xiao, T., Yu, T., Ding, T., Davchev, T., Zhao, T. Z., Armstrong, T., Darrell, T., Jain, V., Vanhoucke, V., Zhan, W., Zhou, W., Burgard, W., Chen, X., Wang, X., Zhu, X., Li, X., Lu, Y., Chebotar, Y., Zhou, Y., Zhu, Y., Xu, Y., Wang, Y., Bisk, Y., Cho, Y., Lee, Y., Cui, Y., hua Wu, Y., Tang, Y., Zhu, Y., Li, Y., Iwasawa, Y., Matsuo, Y., Xu, Z., and Cui, Z. J. (2024). Open X-Embodiment: Robotic learning datasets and RT-X models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*.

[139] OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. (2018). Learning dexterous in-hand manipulation. *CoRR*.

[140] Oquab, M., Darcet, T., Moutakanni, T., Vo, H. V., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., Howes, R., Huang, P.-Y., Xu, H., Sharma, V., Li, S.-W., Galuba, W., Rabbat, M., Assran, M., Ballas, N., Synnaeve, G., Misra, I., Jegou, H., Mairal, J., Labatut, P., Joulin, A., and Bojanowski, P. (2024). Dinov2: Learning robust visual features without supervision.

[141] P. Zhang, L. Hu, B. Z. and Pan, P. (2020). Spatial constrained memory network for semi-supervised video object segmentation. *The 2020 DAVIS Challenge on Video Object Segmentation - CVPR Workshops*.

[142] Pari, J., Shafiullah, N. M., Arunachalam, S. P., and Pinto, L. (2021). The surprising effectiveness of representation learning for visual imitation.

[143] Parisi, S., Rajeswaran, A., Purushwalkam, S., and Gupta, A. (2022). The unsurprising effectiveness of pre-trained vision models for control. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 17359–17371. PMLR.

[144] Parmiggiani, A., Fiorio, L., Scalzo, A., Sureshbabu, A. V., Randazzo, M., Maggiali, M., Pattacini, U., Lehmann, H., Tikhanoff, V., Domenichelli, D., Cardellino, A., Congiu, P., Pagnin, A., Cingolani, R., Natale, L., and Metta, G. (2017). The design and validation of the r1 personal humanoid. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 674–680.

[145] Pasquale, G., Ciliberto, C., Odone, F., Rosasco, L., and Natale, L. (2019). Are we done with object recognition? the icub robot's perspective. *Robotics and Autonomous Systems*, 112:260 – 281.

[146] Pasquale, G., Mar, T., Ciliberto, C., Rosasco, L., and Natale, L. (2016). Enabling depth-driven visual attention on the icub humanoid robot: Instructions for use and new perspectives. *Frontiers in Robotics and AI*, 3:35.

[147] Pathak, D., Shentu, Y., Chen, D., Agrawal, P., Darrell, T., Levine, S., and Malik, J. (2018). Learning instance segmentation by interaction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2042–2045.

[148] Pattacini, U., Nori, F., Natale, L., Metta, G., and Sandini, G. (2010). An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots. In *2010 IEEE/RSJ international conference on intelligent robots and systems*, pages 1668–1674. IEEE.

[149] Peng, S., Jiang, W., Pi, H., Li, X., Bao, H., and Zhou, X. (2020). Deep snake for real-time instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8533–8542.

[150] Perez-Rua, J.-M., Zhu, X., Hospedales, T. M., and Xiang, T. (2020). Incremental few-shot object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13846–13855.

[151] Perronnin, F., Sánchez, J., and Mensink, T. (2010). Improving the fisher kernel for large-scale image classification. In Daniilidis, K., Maragos, P., and Paragios, N., editors, *Computer Vision – ECCV 2010*, pages 143–156, Berlin, Heidelberg. Springer Berlin Heidelberg.

[152] Pinheiro, P. O., Collobert, R., and Dollar, P. (2015). Learning to segment object candidates. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 1990–1998. Curran Associates, Inc.

[153] Pinheiro, P. O., Lin, T.-Y., Collobert, R., and Dollár, P. (2016). Learning to refine object segments. In *European conference on computer vision*, pages 75–91. Springer.

[154] Qin, Y., Huang, B., Yin, Z.-H., Su, H., and Wang, X. (2023). Dexpoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation. In *Conference on Robot Learning*, pages 594–605. PMLR.

[155] Qin, Y., Wu, Y.-H., Liu, S., Jiang, H., Yang, R., Fu, Y., and Wang, X. (2022). Dexmv: Imitation learning for dexterous manipulation from human videos. In *European Conference on Computer Vision*, pages 570–587. Springer.

[156] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR.

[157] Radosavovic, I., Xiao, T., James, S., Abbeel, P., Malik, J., and Darrell, T. (2023). Real-world robot learning with masked visual pre-training. In *Conference on Robot Learning*, pages 416–426. PMLR.

[158] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

[159] Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2018). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania.

[160] Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. (2019). Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR.

[161] Rana, K., Talbot, B., Dasagi, V., Milford, M., and Sünderhauf, N. (2020). Residual reactive navigation: Combining classical and learned navigation strategies for deployment in unknown environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11493–11499.

[162] Ranjbar, A., Vien, N. A., Ziesche, H., Boedecker, J., and Neumann, G. (2021). Residual feedback learning for contact-rich manipulation tasks with uncertainty. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2383–2390. IEEE.

[163] Redmon, J. and Farhadi, A. (2018). YOLOv3: An incremental improvement. *CoRR*, abs/1804.02767.

[164] Remez, T., Huang, J., and Brown, M. (2018). Learning to segment via cut-and-paste. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[165] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NIPS)*.

[166] Rosete-Beas, E., Mees, O., Kalweit, G., Boedecker, J., and Burgard, W. (2022). Latent plans for task agnostic offline reinforcement learning.

[167] Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings.

[168] Rudi, A., Camoriano, R., and Rosasco, L. (2015). Less is more: Nyström computational regularization. *Advances in Neural Information Processing Systems*, 28.

[169] Rudi, A., Carratino, L., and Rosasco, L. (2017). Falkon: An optimal large scale kernel method. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 3888–3898. Curran Associates, Inc.

[170] S. Garg, V. G. and Kumar, S. (2020). Unsupervised video object segmentation using online mask selection and space-time memory networks. *The 2020 DAVIS Challenge on Video Object Segmentation - CVPR Workshops*.

[171] Saad, Y. (2003). *Iterative methods for sparse linear systems*. SIAM.

[172] Schoettler, G., Nair, A., Luo, J., Bahl, S., Ojea, J. A., Solowjow, E., and Levine, S. (2020). Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5548–5555. IEEE.

[173] Schölkopf, B. and Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

[174] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France. PMLR.

[175] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[176] Seno, T. and Imai, M. (2022). d3rlpy: An offline deep reinforcement learning library. *Journal of Machine Learning Research*, 23(315):1–20.

[177] Shang, W., Wang, X., Srinivas, A., Rajeswaran, A., Gao, Y., Abbeel, P., and Laskin, M. (2021). Reinforcement learning with latent flow. *Advances in Neural Information Processing Systems*, 34:22171–22183.

[178] Shaw, K., Bahl, S., and Pathak, D. (2023). Videodex: Learning dexterity from internet videos. In *Conference on Robot Learning*, pages 654–665. PMLR.

[179] Shelhamer, E., Long, J., and Darrell, T. (2017). Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651.

[180] Shmelkov, K., Schmid, C., and Alahari, K. (2017). Incremental learning of object detectors without catastrophic forgetting. In *Proceedings of the IEEE international conference on computer vision*, pages 3400–3409.

[181] Shridhar, M., Manuelli, L., and Fox, D. (2023). Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pages 785–799. PMLR.

[182] Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., Zettlemoyer, L., and Fox, D. (2020). Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[183] Shu, X., Liu, C., Li, T., Wang, C., and Chi, C. (2018). A self-supervised learning manipulator grasping approach based on instance segmentation. *IEEE Access*, 6:65055–65064.

[184] Siam, M., Jiang, C., Lu, S., Petrich, L., Gamal, M., Elhoseiny, M., and Jagersand, M. (2019). Video object segmentation using teacher-student adaptation in a human robot interaction (hri) setting. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 50–56. IEEE.

[185] Silver, T., Allen, K., Tenenbaum, J., and Kaelbling, L. (2018). Residual policy learning. *arXiv preprint arXiv:1812.06298*.

[186] Suchi, M., Patten, T., Fischinger, D., and Vincze, M. (2019). Easylabel: a semi-automatic pixel-wise object annotation tool for creating robotic rgb-d datasets. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6678–6684. IEEE.

[187] Sung, K. K. (1996). *Learning and Example Selection for Object and Pattern Detection*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA. AAI0800657.

[188] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction. Second Edition.* MIT press.

[189] Taheri, O., Ghorbani, N., Black, M. J., and Tzionas, D. (2020). Grab: A dataset of whole-body human grasping of objects. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, pages 581–600. Springer.

[190] Tan, M., Pang, R., and Le, Q. V. (2020). EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790.

[191] ten Pas, A., Gualtieri, M., Saenko, K., and Platt, R. (2017). Grasp pose detection in point clouds. *The International Journal of Robotics Research*, 36(13-14):1455–1473.

[192] Tian, Z., Shen, C., Chen, H., and He, T. (2019). FCOS: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9627–9636.

[193] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE.

[194] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

[195] Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*.

[196] Vezzani, G., Pattacini, U., and Natale, L. (2017). A grasping approach based on superquadric models. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1579–1586.

[197] Voigtlaender, P. and Leibe, B. (2017). Online adaptation of convolutional neural networks for the 2017 davis challenge on video object segmentation. *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops*.

[198] Wada, K., Okada, K., and Inaba, M. (2019). Joint learning of instance and semantic segmentation for robotic pick-and-place with heavy occlusions in clutter. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9558–9564. IEEE.

[199] Walke, H., Black, K., Lee, A., Kim, M. J., Du, M., Zheng, C., Zhao, T., Hansen-Estruch, P., Vuong, Q., He, A., Myers, V., Fang, K., Finn, C., and Levine, S. (2023). Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning (CoRL)*.

[200] Wang, C., Xu, D., Zhu, Y., Martin-Martin, R., Lu, C., Fei-Fei, L., and Savarese, S. (2019). Densefusion: 6d object pose estimation by iterative dense fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[201] Wang, C.-Y. and Liao, H.-Y. M. (2024). YOLOv9: Learning what you want to learn using programmable gradient information.

[202] Wang, J., Qin, Y., Kuang, K., Korkmaz, Y., Gurumoorthy, A., Su, H., and Wang, X. (2024). CyberDemo: Augmenting Simulated Human Demonstration for Real-World Dexterous Manipulation. *arXiv preprint arXiv: 2402.14795*.

[203] Wang, W., Feiszli, M., Wang, H., Malik, J., and Tran, D. (2022). Open-world instance segmentation: Exploiting pseudo ground truth from learned pairwise affinity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4422–4432.

[204] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR.

[205] Wen, B., Tremblay, J., Blukis, V., Tyree, S., Müller, T., Evans, A., Fox, D., Kautz, J., and Birchfield, S. (2023). Bundlesdf: Neural 6-dof tracking and 3d reconstruction of unknown objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 606–617.

[206] Williams, C. and Seeger, M. (2000). Using the nyström method to speed up kernel machines. *Advances in neural information processing systems*, 13.

[207] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

[208] Wu, J., Jiang, Y., Yan, B., Lu, H., Yuan, Z., and Luo, P. (2023a). Exploring transformers for open-world instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6611–6621.

[209] Wu, J., Li, X., Xu, S., Yuan, H., Ding, H., Yang, Y., Li, X., Zhang, J., Tong, Y., Jiang, X., et al. (2024). Towards open vocabulary learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[210] Wu, S., Zhang, W., Jin, S., Liu, W., and Loy, C. C. (2023b). Aligning bag of regions for open-vocabulary object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15254–15264.

[211] Wu, Z., Lu, Y., Chen, X., Wu, Z., Kang, L., and Yu, J. (2022). Uc-owod: Unknown-classified open world object detection. In *European Conference on Computer Vision*, pages 193–210. Springer.

[212] Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2018). Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *Robotics: Science and Systems (RSS)*.

[213] Xie, C., Xiang, Y., Mousavian, A., and Fox, D. (2020). The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation. In *Conference on robot learning*, pages 1369–1378. PMLR.

[214] Xie, C., Xiang, Y., Mousavian, A., and Fox, D. (2021). Unseen object instance segmentation for robotic environments. *IEEE Transactions on Robotics*.

[215] Xie, L., Wang, S., Rosa, S., Markham, A., and Trigoni, N. (2018). Learning with training wheels: speeding up training with a simple controller for deep reinforcement learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6276–6283. IEEE.

[216] Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017). Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500.

[217] Xiong, H., Mendonca, R., Shaw, K., and Pathak, D. (2024). Adaptive mobile manipulation for articulated objects in the open world. *arXiv preprint arXiv:2401.14403*.

[218] Yang, X., Ji, Z., Wu, J., Lai, Y.-K., Wei, C., Liu, G., and Setchi, R. (2022). Hierarchical reinforcement learning with universal policies for multistep robotic manipulation. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):4727–4741.

[219] Zakharov, S., Shugurov, I., and Ilic, S. (2019). Dpod: 6d pose object detector and refiner. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[220] Zakka, K. (2022). Scanned Objects MuJoCo Models.

[221] Zeng, A., Florence, P., Tompson, J., Welker, S., Chien, J., Attarian, M., Armstrong, T., Krasin, I., Duong, D., Sindhwani, V., and Lee, J. (2021). Transporter networks: Rearranging the visual world for robotic manipulation. In Kober, J., Ramos, F., and Tomlin, C., editors, *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 726–747. PMLR.

[222] Zeng, A., Song, S., Lee, J., Rodriguez, A., and Funkhouser, T. (2020). Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319.

[223] Zhang, J., Zhang, J., Pertsch, K., Liu, Z., Ren, X., Chang, M., Sun, S.-H., and Lim, J. J. (2023a). Bootstrap your own skills: Learning to solve new tasks with large language model guidance. In *7th Annual Conference on Robot Learning*.

[224] Zhang, S., Wicke, P., Şenel, L. K., Figueredo, L., Naceri, A., Haddadin, S., Plank, B., and Schütze, H. (2023b). Lohoravens: A long-horizon language-conditioned benchmark for robotic tabletop manipulation. *arXiv preprint arXiv:2310.12020*.

[225] Zhao, M., Abbeel, P., and James, S. (2022). On the effectiveness of fine-tuning versus meta-reinforcement learning. *Advances in Neural Information Processing Systems*, 35:26519–26531.

[226] Zhao, T. Z., Kumar, V., Levine, S., and Finn, C. (2023). Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea.

[227] Zhou, G., Dean, V., Srirama, M. K., Rajeswaran, A., Pari, J., Hatch, K., Jain, A., Yu, T., Abbeel, P., Pinto, L., Finn, C., and Gupta, A. (2023). Train offline, test online: A real robot learning benchmark. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*.

[228] Zhou, X., Wang, D., and Krähenbühl, P. (2019). Objects as points. *arXiv preprint arXiv:1904.07850*.

[229] Zhu, T., Wu, R., Hang, J., Lin, X., and Sun, Y. (2023). Toward human-like grasp: Functional grasp by dexterous robotic hand via object-hand semantic representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–14.

[230] Zhu, T., Wu, R., Lin, X., and Sun, Y. (2021). Toward human-like grasp: Dexterous grasping via semantic representation of object-hand. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15741–15751.

[231] Zitkovich, B., Yu, T., Xu, S., Xu, P., Xiao, T., Xia, F., Wu, J., Wohlhart, P., Welker, S., Wahid, A., Vuong, Q., Vanhoucke, V., Tran, H., Soricut, R., Singh, A., Singh, J., Sermanet, P., Sanketi, P. R., Salazar, G., Ryoo, M. S., Reymann, K., Rao, K., Pertsch, K., Mordatch, I., Michalewski, H., Lu, Y., Levine, S., Lee, L., Lee, T.-W. E., Leal, I., Kuang, Y., Kalashnikov, D., Julian, R., Joshi, N. J., Irpan, A., Ichter, B., Hsu, J., Herzog, A., Hausman, K., Gopalakrishnan, K., Fu, C., Florence, P., Finn, C., Dubey, K. A., Driess, D., Ding, T., Choromanski, K. M., Chen, X., Chebotar, Y., Carbajal, J., Brown, N., Brohan, A., Arenas, M. G., and Han, K. (2023). Rt-2: Vision-language-action models transfer web knowledge to robotic control. In Tan, J., Toussaint, M., and Darvish, K., editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR.

[232] Zohar, O., Lozano, A., Goel, S., Yeung, S., and Wang, K.-C. (2023). Open world object detection in the era of foundation models. *arXiv preprint arXiv:2312.05745*.