POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Pose Estimation and Semantic Meaning Extraction for Robotics Using Neural Networks

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Davide Figundio**

Student ID: 965898
Advisor: Prof. Andrea Maria Zanchettin
Co-advisors: Ing. Niccolo' Lucci, Prof. Paolo Rocco
Academic Year: 2021-22

Abstract

With the introduction of Convolutional Neural Networks (CNN), we have witnessed large advancements in accuracy and precision in the fields of object detection and 6D pose estimation from RGB images. We investigate the use of CNNs to verify their applicability for perception tasks in the fields of industrial and collaborative robotics. In particular, we devise a method for generating realistic training datasets for objects in a predetermined environment starting from photographs, greatly facilitating the usually laborious and expensive data acquisition phase that is considered to be a pre-requisite for machine learning applications. We then trained a neural network on various experimental datasets of this sort to evaluate its performance. We also devised an approach to extrapolate the semantic state of a scene from the ouputs of a pose estimation network. Finally, we demonstrated the performance of our methods in a real-world scenario by using the output of a trained neural network to plan the movement of a robotic manipulator.

Keywords: Robotics, Machine Learning, AI, Neural Networks, Semantics, Pose Estimation.

Abstract in lingua italiana

L'introduzione delle Reti Neurali Convoluzionali (CNN) ha dato il via ad enormi sviluppi nei campi dell'identificazione di oggetti e stima della posa 6D partendo da immagini a colori. In questa tesi studiamo l'utilizzo delle CNN per verificarne l'applicabilita' alla percezione nei campi della robotica industriale e collaborativa. In particolare, presentiamo un metodo per generare dataset realistici per allenare reti neurali, facilitando notevolmente il laborioso processo di acquisizione dati richiesto dal machine learning. Avendo poi allenato una rete neurale per verificarne la precisione, sviluppiamo un metodo per estrapolare lo stato semantico di una scena utilizzando le stime della posa fornite dalla rete. Finalemente, dimostriamo l'affidabilita' dei nostri metodi in uno scenario reale, utilizzandone i risultati per pianificare il movimento di un manipolatore robotico.

Parole chiave: Robotica, Machine Learning, Intelligenza Artificiale, Reti Neurali, Semantica, Stima della Posa.

Contents

Introduction

0.1. General Overview

Perception has always been a fundamental task in the field of robotics. To perform actions and modify its environment in an automated manner, a robot must be provided with proper knowledge of its surroundings via sensor data and appropriate processing methods.

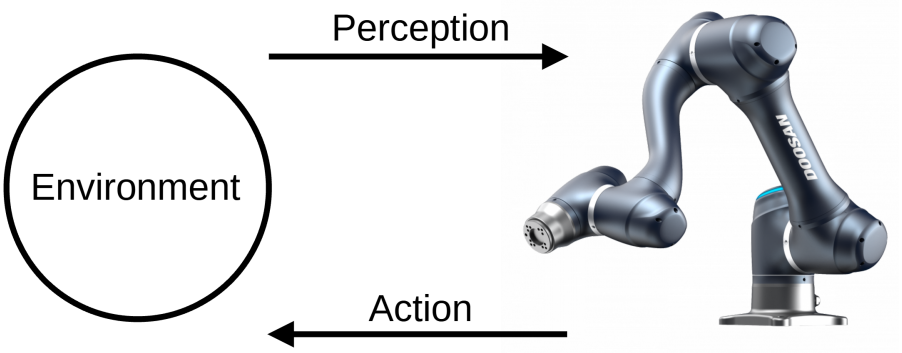


Figure 1: The main ways a robot interacts with the environment: perception and action.

One of the most readily available sensors applicable to such a scenario is the color camera; however, effectively utilizing a camera in a real-time control scenario brings about a series of difficulties and complications. While commonly used sensors, such as thermometers or encoders, usually output a single value which represents a measurement of a physical quantity, cameras generate data with an entirely different scale an meaning. A relatively low resolution 1280x720 pixel RGB sensor outputs an array of 2'774'800 values, at a rate that is anywhere between five to sixty times per second, or more. Furthermore, contrary to what occurs with traditional sensors, these values do not directly reflect a measurment of a physical quantity, but are linked through complex interactions that are difficult to express in an analytical manner. Thus devising a method to process all this data and make it available to a control algorithm is a daunting task.

Neural networks, however, are suited to face these challenges, since they are built to work with a large number of inputs in parallel, and are capable of modelling unknown

and complex functions through proper training. In particular, Convolutional Neural Networks (CNNs) have been widely applied to the field of image processing, due to their independance from prior knowledge, the relatively little amount of pre-processing they require, and a decreased tendency to overfit when compared with their fully-connected counterparts.

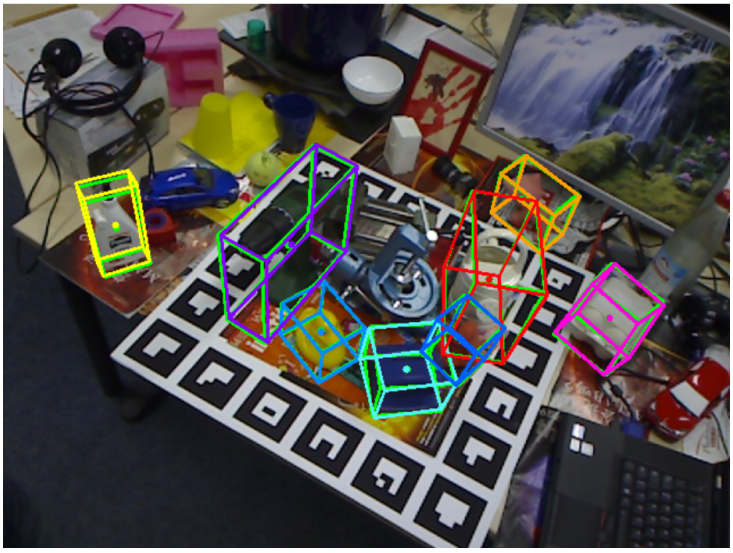


Figure 2: Multi-object inference on an image from the LINEMOD[17] dataset performed by EfficientPose[5], a 6D pose estimation CNN. Green bounding boxes visualize ground truth poses while other colors represent estimations.

Since the introduction of CNNs, most tasks related to image processing have therefore seen rapid advancements in recent years. Two particular subfields of interest for robotics that have advanced by leaps and bounds are object detection and pose estimation. Object detection tackles the problem of verifying whether an image contains a particular object, while pose estimation techniques provide the position and orientation of the detected object in relation to the camera. A combination of these tasks could cover the issues impeding RGB cameras from being effectively used in control scenarios.

However, obtaining accurate and reliable sensor data is only part of the overall perception task in autonomous systems. Another important issue often encountered is the necessity of transforming the raw outputs obatined from an array of sensors into abstract high-level information. This is ultimately essential for high-level decision making, since is often performed using semantic planners, such as PDDL planners, which rely on symbolic represenations of the state, and are ubiquitous in industrial and collaborative robotics applications.

0.2. Thesis Goals

Our objective in this thesis is therefore to verify the applicability and performance of a state-of-the-art 6D pose estimation CNN to a robotics scenario, and the difficulties involved in doing so. When working with machine learning approaches, we face the issue that neural networks in general are incapable of understanding cause and effect relationships, being only practiced in percieving and optimizing the relationship between input and output data. Therefore the only way to ensure correct behaviour is to train them with proper *training inputs*, and associated ideal outputs (the *ground truth*), which together form a *dataset*.

However, acquiring the necessary data to build a high-quality dataset is often time-consuming and expensive. Our initial goal is therefore to improve the applicability of machine learning approaches to pose estimation tasks by facilitating this initial process. For this purpose, we demonstrate an approach for quickly and easily generating large amounts of synthetic labelled data starting from a set of background images, providing realistic depictions of a small number of objects of interest in a determined environment.

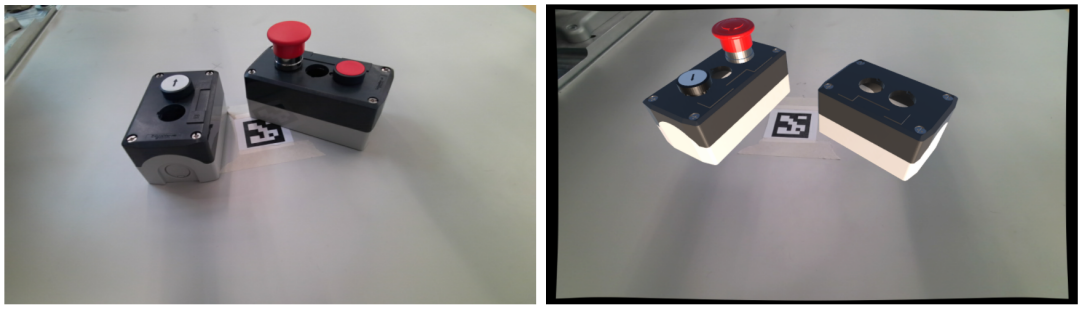


Figure 3: Side-by-side comparison of a real image and a generated training image resulting from our pipeline.

We then verify the performance of a model trained on synthetic data, and whether it is capable of effectively generalising to real world conditions.

For many applications in industrial and collaborative robotics, estimations of the pose may not be sufficient to complete a task, but more high-level knowledge of the semantic state of a scene is required. Therefore we developed a method that uses the outputs from a pose estimation network to compute this semantic state, and tested its reliability in a simple assembly scenario.

Finally, by combining the results of the previous two methods, we verified the performance of the complete system in a real-world application by using the estimations from

the trained network and from our semantic meaning extraction algorithm to plan the movement of a robotic manipulator.

0.3. Achieved Results

Our dataset generation methodology proved to be crucial when testing out different variations of traning inputs and their effect on the final performance of the network. Overall, the resulting system was able to obtain good results and generalise in a satisfactory manner to real-world conditions. Our best performing model resulted in an average error of 1.9052 mm when predicting the pose, corresponging to 98.46% correct predictions. The semantics extrapolation method also achieved remarkably good performance, reaching a top F1 score of 0.9763, which however is highly dependant on the quality of the initial pose estimations from the network. Finally, applying the combination of pose estimation and semantics extraction methods to a real-life scenario made our robot capable of planning and completing simple assembly tasks, all while relying only on an RGB camera for sensing. However, its reliability may not be sufficient for tasks that require very high precision, and is once again highly dependant on the performance of the underlying neural network.

0.4. Thesis structure

The rest of this thesis is organized as follows. We will begin by examining the current state-of-the-art for pose estimation and semantics approaches (Chapter 1), and we will give some background on the pose estimation network we chose as the basis of our approach, EfficientPose (Chapter 2). We will then go over our methodologies and underlying motivations (Chapter 3), and the metrics and experiments used to evaluate them (Chapter 4). Finally, we will lay out the results of our experiments (Chapter 5), and conclude the thesis (Chapter 6).

1 | State of the Art

In this chapter we will give an overview of 6D pose estimation and semantics methodologies in the state-of-the-art.

Pose estimation is subject to ongoing research, as it has wide applicability in a variety of fields, including but not limited to robotics, autonomous vehicles, augmented reality and computer vision. The methodologies supporting this issue can be divided into two main categories: learning-based and non-learning based, as explained hereafter.

1.1. Non-Learning-Based Methods

The first pose estimation algorithms worked through image segmentation and voting schemes. In 1972, Dula and Hart started using Hough[18] voting to detect lines and curves in images[11], and Ballard later generalised this procedure to analytically defined shapes[1], popularizing its application for computer vision. In parallel, Lamdan and Wolfson published their Geometric Hashing[22] method, which is based on the representation and matching of objects using their minimal features, such as points or lines.

Modern approaches can be divided into three sub-categories. 2D-3D correspondence methods aim to recognize features in an image and match them to known object characteristics [3], but often rely on texture information, and cannot be applied to textureless objects. Real-image-based methods[16] transform the pose estimation problem into an image-matching problem, associating the detected image to a database of previously saved templates. This requires a difficult and time consuming process to acquire these reference images. CAD image-based methods[26] aim to circumvent this by rendering the references using a 3D model. All of these approaches have issues with adapting to new situations, such as strong changes in illumination, cluttered scenes, and repeated objects.

A much easier way of identifying the 6D pose is through the usage of specialized markers. When placed on an object and photographed, these markers highlight the points on the image that correspond to the 3D location of the marker, and the pose can then be obtained by solving a Perpective-n-Points[37] (PnP) problem. For example, an ArUco

marker[13], can be easily and robustly detected by applying image thresholding and contour extraction, and its pose estimated by using its corners as keypoints[28]. The obvious downside of this method is that it requires markers to be applied to objects, which is not feasible at an industrial level. Another downside is that it also does not deal with partial or total occlusions of the marker(s).

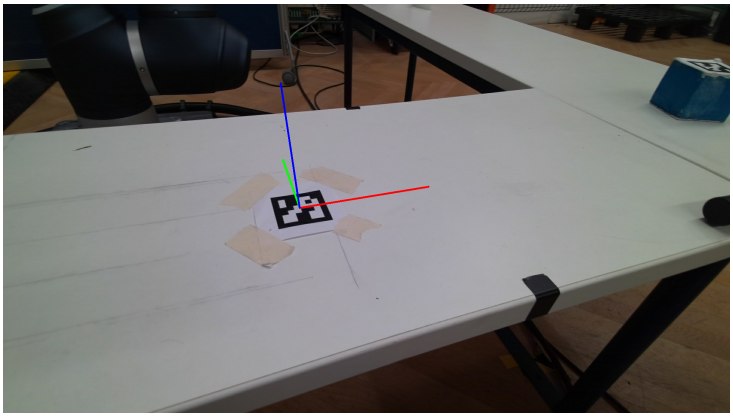


Figure 1.1: An example of pose identification using ArUco markers.

Overall, non-learning based methods, while simple and computationally efficient, often require strictly controlled enviroments and specific conditions to be functional. This greatly restricts their applicability, and therefore learning-based methods are more widely used.

1.2. Learning-Based Methods

Since the introduction of Convolutional Neural Networks (CNN), artifical intelligence and deep learning have been widely applied to the field of image processing, including its subfields of object detection and 6D pose estimation. The methods described in this section aim to train a CNN on vast quantities of data to perform a certain task. Based on this task, we can categorize these approaches into three main branches: 2D-3D correspondence, direct estimation, and pose refinement. We will give a couple of examples for each of these categories.

1.2.1. 2D-3D Correspondence

This class of methods uses a two step approach: they first implement a neural network to regress a set of 2-D points from an image, corresponding to a set of known feature points, and then use PnP to obtain the 6D pose of the object.

BB8[30] uses object segmentation to perform 2D object detection, then regresses the

8 points that form the 3D bounding box of an object, but struggles with textureless symmetric or partially occluded objects. To combat these issues, PVNet[29] uses farthest point sampling to select keypoints on the surface of the object, and then implements a dense pixel-wise voting network, where each pixel "votes" on locations for the keypoints. RANSAC[12] is then used to exclude outliers and obtain predictions with their probability distribution, which are then used for uncertainty-driven PnP.

Most approaches in this class share two common weaknesses. First, they are very perfomance-intensive when estimating the pose of multiple objects, since keypoint regression and PnP have to be computed for each object individually[5]. Second, they are not end-to-end trainable, as the loss functions implemented do not reflect the final perfomance on the pose estimation task[19]. However, recent approaches have faced this issue by implementing learned or differentiable PnP algorithms, so as to enable end-to-end training[6].

1.2.2. Direct Estimation

The approaches in this category exploit convolutional neural networks to directly regress the pose of an object in a single step. They are end-to-end trainable and boast better run times than the previously seen 2D-3D methods.

PoseNet[20] was one of the first implementations of this concept, and was originally conceptualized for obtaining the camera pose from outdoor or indoor enviroments, and not for object pose estimation. Deep-6DPose[10] works by extending the Mask R-CNN[15] instance segmentator, which in turn extends the Faster R-CNN[31] object detector, and introduced a key technical feature by decoupling rotation and translation parameters, so as to make the pose regression loss differential. PoseCNN[40] expanded on this idea, and introduced a novel loss function that enabled it to properly handle symmetrical objects.

Most networks in this category are fast and computationally efficient, but struggle in situations with partial occlusions.

1.2.3. Pose Refinement

The previously mentioned algorithms may only give a rough estimate of the object pose. If greater accuracy is required, it is often necessary to use pose refinement algorithms. Approaches in this category start from an inital estimate, and then use various methods to refine it, obtaining a more accurate prediction.

DeepIM[23] employs an iterative approach, by repeatedly rendering a 3D model of the object and matching it against the observed image. To ensure successive iteraterations

Rank	Model Name	Mean ADD	Method	Year
1	RNNPose	97.37	Refinement	2022
2	EfficientPose	97.35	Direct + Refinement	2020
3	RePOSE	96.1	Refinement	2021
4	EPro-PnP-6DoF v1	95.8	2D-3D	2022
5	ROPE	95.61	2D-3D	2021
6	DPOD	95.2	2D-3D + Refinement	2019
7	HRNet	93.3	2D-3D	2019
8	HybridPose	91.3	2D-3D	2020
9	CDPN	89.86	2D-3D + Direct	2019
10	PoseCNN + DeepIM	88.6	Direct + Refinement	2018

Table 1.1: Top ten performing models on the LINEMOD dataset[17] as of November 2022, ranked by their ADD metric (see section 4.1.2).

gain in precision, it is trained not only on an annotated dataset, but also on previous outputs of the network. RNNPose[41], while also starting from a rendering and the observed image, formulates the task as a nonlinear optimization problem: it minimizes differences between correspondence fields by leveraging recent discoveries in the field of optical flow estimation, while recurrently calling itself. RNNPose currently boasts the best performance on the widely used LINEMOD[17] dataset by a narrow margin, as highlighted by table 1.1.

While refinement methods achieve remarkable performance, they have two main downsides. First, they rely on an inital estimate of the pose, so they cannot be applied independently, and second, they are computationally intensive, especially when one considers that they must be run in parallel with another estimation method to generate the initial poses.

1.2.4. Conclusions on Learning Approaches

Data-driven models based on deep learning are more widely applied than non-learning-based models, as a large variety of approaches exists, and each approach brings its own distinct advantages. When choosing a model, special attention must be given to the application the model is destined for. For single object pose estimation, especially in highly occluded enviroments, 2D-3D methods are the best choice. For multi-object pose estimation, direct estimation methods provide greater computational efficency. Whenever greater accuracy is required, pose refinement algorithms offer the best results.

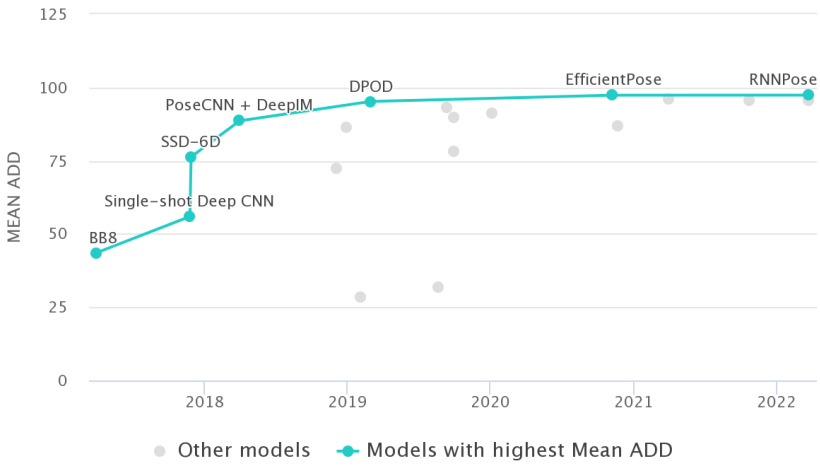


Figure 1.2: Performance on LINEMOD of recent pose estimation algorithms by year. Graphic originates from paperswithcode.com/sota/6d-pose-estimation-on-linemod.

1.3. Semantic Description Approaches

The problem of building abstract descriptions from low level data in robotics has persisted for as long as high-level planners have been in use. Robots percieve the world through sensor inputs, and act upon these perceptions by supplying their motors and drives with appropriate control targets. However, it is very difficult to plan actions while purely relying on a low-level representation: therefore high-level descriptions simplify planning for complicated tasks considerably.

Initial approaches face this issue by abstracting the lower details of control (*procedural abstraction*). Researchers at Stanford in 1984, while developing their mobile robot "Shakey"[27], developed an approach that would combine individual low-level actions, into progressively higher hierarchies of subroutines, each representing the combination of a sequence of actions to perform a task. A plan then consists in series of high-level skills necessary in order to reach the goal, a concept that has been re-applied numerous times [2][32].

Koindaris et al. [21], however, show that performance can be further improved through *state abstraction*: by defining a set of questions that an abstract representation will be used to answer, we can construct a representation that has the capability of answering them. This further simplifies planning, bypassing the inherent complexity of the robot's sensing space by providing a simplified representation of the state to the decision-making process. Koindaris also shows how this representation can be converted into commonly used planning formats, such as Planning Domain Definition Language[14].

2 | EfficientPose Background

In this chapter, we will be providing a brief insight into the EfficientPose 6D pose estimation neural network. EfficientPose was chosen as the starting point for this thesis, as it boasts state-of-the-art results while maintaining relative simplicity and low computational costs. It is designed with multi-object estimation in mind, which is especially significant for robotics applications, as other approaches do not scale well with the number of total detections.

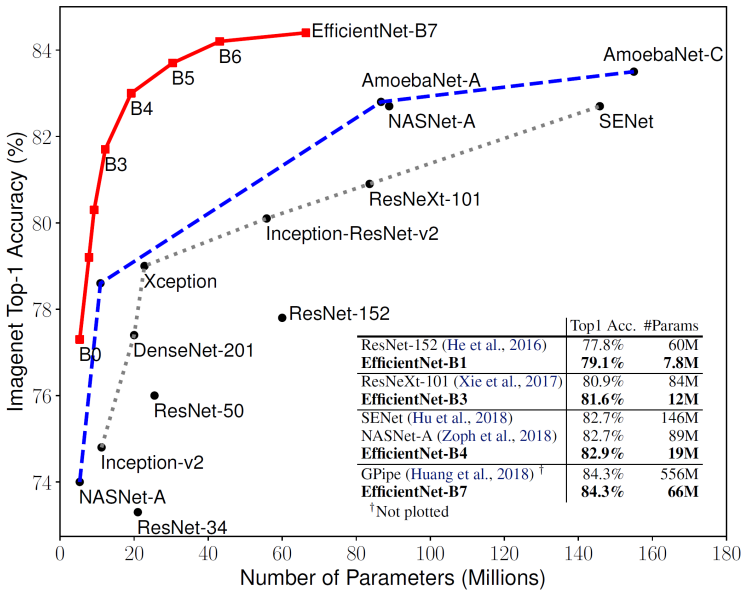
2.1. Parent Networks

Similarly to Deep-6DPose (previously mentioned in section 1.2.2), EfficientPose is an end-to-end direct 6D pose estimation approach that extends the functionality of a 2D network. While Deep-6D extends the segmentation network Mask-R-CNN, EfficientPose extends Google's object classification network EfficientDet[34], which in turn builds on Google's backbone network EfficientNet[33].

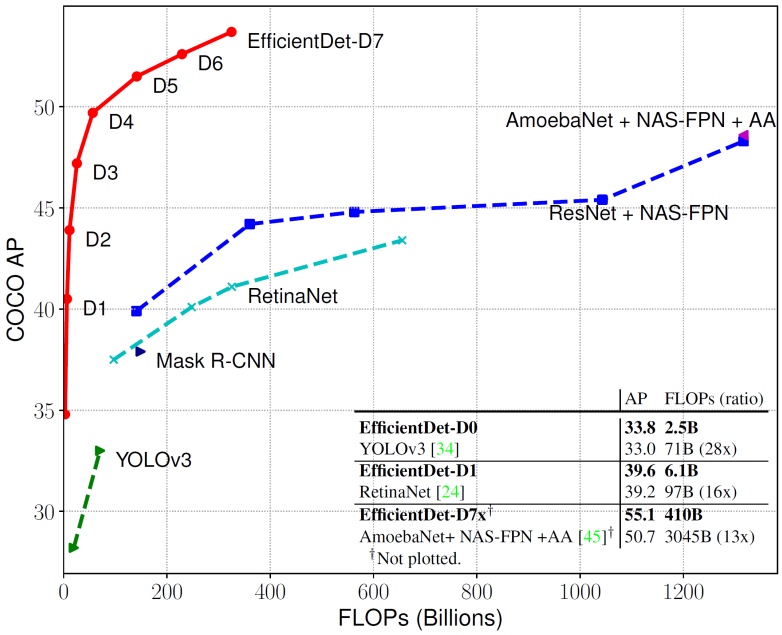
Both of these parent networks are based on the concept of scalability: the possibility of increasing network dimensions to achieve better performance in exchange for greater computational cost. These approaches scale using a single hyperparameter, ϕ. For $\phi = 0$, we have a base network with minimum depth, width and resolution in an optimal ratio. By increasing the value of ϕ, we can scale up these dimensions while mainining the ratio, thus obtaining better performance than if we had just increased the depth, width or resolution of the network individually.

The backbone of this approach is EfficientNet: a network, or more precisely a family of networks, that was designed using Network Architecture Search (NAS)[42] to provide an optimal ratio of depth, width and resolution. This allows it to obtain a performance that is similar or better than other backbone networks while requiring much fewer parameters, thus greatly lowering computational costs.

EfficientDet then expands on EfficientNet by adding a Feature Pyramid Network (FPN)[25], which performs multi-scale feature fusion, combining data from low-resolution, semanti-



(a) EfficientNet: X values are the number of parameters used in the network, Y values are the percentage of correct answers on the Imagenet dataset[9].



(b) EfficientDet: X values are the number of Floating Point Operations per second (FLOPs), Y values are the Average Precision (AP) tested on the Microsoft Commmon Objects in Context dataset[24].

Figure 2.1: Performance of the EfficientNet and EfficientDet families compared to other approaches.

cally strong layers with data from high-resolution, semantically weak layers. In particular, EfficientDet uses a new form of bi-directional feature pyramids, providing multiple top-down and bottom-up aggregation paths with learnable weights. The outputs of this feature network are then fed into multiple subnetworks that use them to perform single tasks, such as object class and 2-D bounding box estimations. This structure is depicted in figure 2.2.

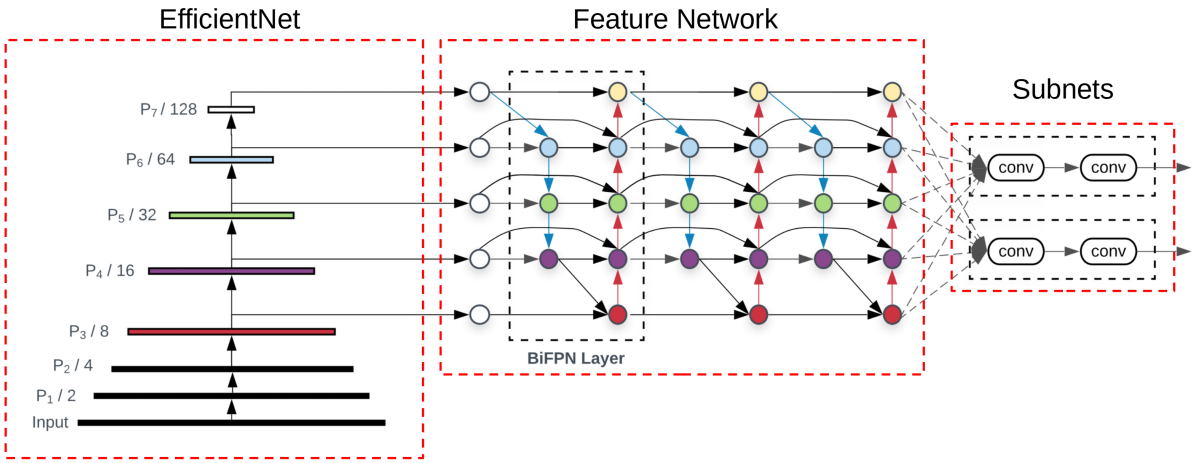


Figure 2.2: Overview of the EfficientDet architecture. BiFPN layers and subnet layers may be repeated multiple times according to resource constraints.

One advantage EfficientDet has over many other 2-D object detectors is that it is single-shot: while other methods require an intermediate region proposal step, EfficientDet performs inference directly on the input image. This means that it requires significantly lower computational costs, while maintaining state-of-the-art performance.

2.2. Pose Estimation Methodology

EfficientPose expands on EfficientDet's architecture with the addition of two additional subnets, which predict translation and rotation for each object class. Since these networks are relatively small, the additional computational costs are minimal.

The rotation network outputs a vector $R \in \mathbb{R}^3$ containing a minimal representation of the rotation in Rodrigues angles, and then employs an additional iterative refinement strategy similar to what has been utilised in other pose refinement methods. Both the network size and the number of iterations are controlled by the hyperparameter ϕ.

The translation network instead splits the task of predicting the position $p = [x, y, z]^T$ of the object into separate predictions of the 2D center $c = [c_x, c_y]^T$ and of the depth z,

analogously to what is done in other direct estimation networks such as PoseCNN [40]. The final position p can be computed using the camera intrinsic parameters by inverting the relationship:

$$\begin{bmatrix} c_x \\ c_y \\ 1 \end{bmatrix} = \frac{1}{z} \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

where f_x, f_y are the focal lengths and (p_x, p_y) is the principal point. Thus we obtain:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = z \begin{bmatrix} \frac{1}{f_x} & 0 & \frac{p_x}{f_x} \\ 0 & \frac{1}{f_y} & \frac{p_y}{f_y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_x \\ c_y \\ 1 \end{bmatrix}$$

This also means that EfficientPose is trained on a single set of camera parameters, since the estimation of the depth would be thrown off by changes in the focal distance.

An advantage of EfficientPose's approach is that multiple class, box, rotation and translation networks for different object instances can share the same backbone and feature network. This minimizes additional computation costs for multi-object pose estimation and training, with the downside of losing accuracy compared to a network trained on a single object.

2.3. Network Performance

EfficientPose is trained and evaluated on the LINEMOD and Occulsion-LINEMOD datasets, which are standard benchmarks for pose estimation methods.

For single-object estimation on LINEMOD, EfficientPose obtains 97.35% ADD, placing it at the very top of the state-of-the-art, while for multi-object-estimation on Occlusion-LINEMOD, it obtains 79.04% ADD with $\phi = 0$ and 83.90% ADD with $\phi = 3$.

The computational performance also deserves a comment: while running on an Nvidia RTX 2080 Ti graphics card, with $\phi = 0$ it maintained 27.45 FPS for single-object estimation and 26.22 FPS for multi-object estimation.

3 | Methodology

In this chapter we will go over the methods used to train our own pose estimation model. These are structured into three sections, depending on the strategies implemented and their objective.

Our starting point is the EfficientPose pose estimation network, as described in the previous chapter. The first section describes our attempt to train this network using purely synthetic data, with the objective of verifying whether it is possible to facilitate the laborious real-world dataset acquisition phase that is considered to be a prerequisite for deep learning.

The second section takes from the first and introduces a novel dataset generation method, that makes use of "augmented reality" to create arbitrarily large datasets for specific objects in specific environments.

Finally, the third section exploits this method to generate a dataset for new objects, and evaluates the possibility and strategies required for extracting additional semantic information from the output of the pose estimation network.

3.1. Fully Rendered Datasets

3.1.1. Motivations and Objective

Since we are starting with a given pose estimation network, we must train it in order to fit our own needs. In the vast majority of situations, the objects we would like to identify will not be present in any available dataset: therefore the first essential step to develop our model is the creation of our own datasets for training and evaluation.

These datasets consist in a collection of images containing the object we wish to track, with associated ground truths encoding the pose of the tracked object for each image. Collecting this data in the real world is tedious and difficult, considering both the number of samples required for deep learning, and that any errors or biases will strongly affect the perfomance of the trained model.

Various methods have been attempted to perform this step. LINEMOD[17], one of the most popular pose estimation datasets, was generated by setting the dataset objects onto a board lined by ArUco markers. The pose of the objects relative to the board is then computed with good accuracy using depth information; thus by photographing the board from different angles it is possible to obtain a complete and accurate set of poses for each object by first computing the pose of the reference board. Similarly, the YCB-Video dataset[40], another popular choice, also uses depth information to accurately obtain an initial estimate for the first frame in a video, and then tracks the camera trajectory to obtain poses for the successive frames, concluding with a final optimization step that minimizes the average error of the ground truths.

Both of these methods require costly equipment and a significant time investment even before training. We would like to investigate whether it is feasible to instead use rendering software, which can generate potentially infinite quantites of training images with associated, perfectly accurate ground truths at low cost.

The largest issue with using synthetic datasets is that, while a model trained in this manner could function perfectly in a simulation, we have no guarantee whether it would also perform similarly in real world conditions. This is because a simulated sensor and simulated enviroment are unable to reproduce unmodeled physical effects and noise in the same way a real sensor would with a real environment, an issue commonly dubbed as the "reality gap"[38].

Domain Randomization[35] is one of the most utilised methods for solving this problem. Its principle states that the introduction of sufficient variability in the simulated domain will allow the model to generalise to the real world with no additional training. This would allow us to entirely skip the laborious data collection step and instead rely on a 3D model of the object we wish to track, which is usually readily available and accurate.

3.1.2. Dataset Generation

To render the images for our dataset, we used the Unity Perception package[36], which integrates domain randomization features into its pipeline. Unity Perception works by simulating a scene, and then rendering each simulated frame from the perspective of a virtual camera.

When setting up the simulation, we specify the number of iterations to simulate and the number of frames to render for each iteration. At the beginning of each iteration, we call a set of randomizer scripts: each one sets one of the domain variables for the iteration, such as the pose of an object or the colour of the light source, by extracting its value

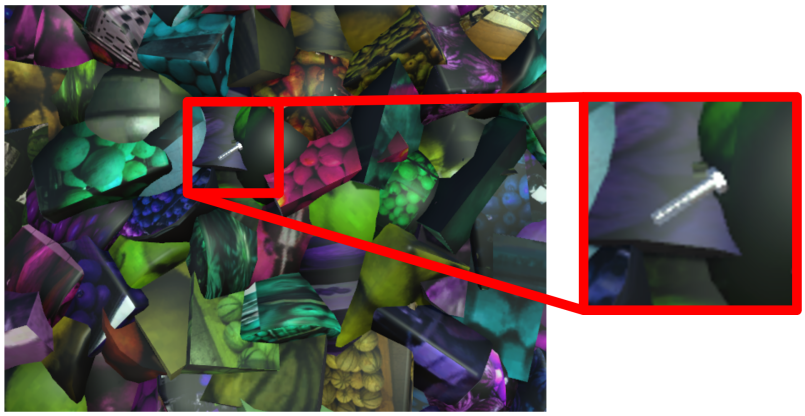


Figure 3.1: One of the images generated with Unity's Perception package for training our model.

from a pre-defined probability distribution. The scene is then updated according to these variables, rendered, and the associated ground truth saved.

As a test case, we decided to generate a dataset for a standard M6x30 hexagonal head screw. This is a very challenging object for pose estimation, as it is small and symmetrical. Symmetrical objects have always been difficult for pose estimation algorithms, due to reasons explained in depth in section 4.1.2. The model of the M6x30 screw was obtained from the FreeCAD Fasteners workbench[4] and colored with a metallic texture.

The domain for this dataset consists of images of the screw placed inside of a scene: thus the primary domain variables are the pose, the background, and the lighting. We used a custom randomizer to set position and rotation for each iteration, and default randomizers provided as part of the Perception package to generate a background, composed by random 3D shapes placed with random positions, orientations and textures. Finally, we used a custom randomizer to set the lighting color, intensity and origin. A sample image from this dataset appears in figure 3.1.

We can then interface the output of this procedure with EfficientPose using a conversion script, which performs the necessary tasks to make the dataset compatible with the network. In this manner we can quickly and easily generate arbitrarily large datasets for training, by first running the Unity scenario, and then running the conversion script for EfficientPose.

3.1.3. Network Training

The original version of EfficientPose is trained on LINEMOD. However, the specifics of LINEMOD and of our own dataset are widely different: LINEMOD has around 1200 images per object, and only about 200 of these are used for training, while our dataset has 10000 images, 9000 of which are used for training. This means that we must set proper training parameters for our own dataset.

First, we reduced the number of epochs from 5000 to 100. Since our dataset contains 45 times more images, these two values represent a similar training time. EfficientPose by default evaluates the model every 10 epochs due to the small epoch size; we change this value and evaluate the netowrk at the end of every epoch.

EfficientPose implements Keras' ReduceLRonPlateau callback to dynamically set the learning rate during training. This is standard practice: large learning rates quickly adjust the model but can lead to fluctuations, local minima and divergence; smaller learning rates avoid these issues but take an excessive amount of time to improve the model[39]. This method instead starts with a large learning rate, and then automatically reduces the learning rate whenever training stagnates, thus maintaining a value closer to the ideal. By default, EfficientPose halves the learning rate every time the accuracy does not improve for 25 epochs; we changed this to an 80% reduction every 5 epochs, to account for the increased number of samples per epoch.

The initial and minimum learning rates are mantained identical to EfficientPose's, set at 10^{-4} and 10^{-7} respectively. For all purposes in this thesis, we will be using the networks scaled to their lowest hyperparameter $\phi = 0$, as going any higher requires excessive amounts of time to train.

The results of this training are presented in section 5.1.

3.2. Augmented Reality Datasets

3.2.1. Motivations and Objective

In the previous section, we explored the option of generating fully synthetic training images using domain randomization. This represents a generic approach: by representing a wide variety of environments and conditions in the dataset, we hope that the model learns to generalise to more situations, including eventually our usecase. However, this is only one way of bridging the reality gap. Another approach could be to "reduce" the gap between simulation and reality, by making the training images as similar as possible

to the real-world environment the model will then be tested in.

This approach has the disadvantage of fewer guarantees on performance outside of the selected environment, which makes it better for usecases which are stationary or limited to fewer settings. However, this is often not an issue for applications in industrial scenarios.

To generate realistic images, we considered two options: either re-creating the environment inside the simulator, or using an "augmented reality" approach by rendering the 3D models of the dataset objects on top of real photographed images taken from the testing environment. We decided on the second option, since it is both faster and gives more realistic final results compared to a fully simulated environment. Furthermore, changing the environment is also as simple as capturing a new set of background images.

Thus our objective is to create a method to easily and quickly generate a realistic training dataset for our testing environment, which is a simple table with a set of objects placed on its surface.

3.2.2. Dataset Generation

We again used Unity Perception to generate our training dataset, as described in 3.1.2. We used models for 5 objects: the same M6x30 screw used previously, a M8x16 round head screw, a M8x25 and M8x50 socket head screws, and a M4x40 countersunk screw. As previously stated, these are small, symmetric objects that are generally challenging to identify, with the additional complication that they all have similar shapes and sizes. Only the first four are annotated, while the M4x40 is included as a "decoy" to reduce the number of false positives (a concept further explained in subsection 5.1.2).

The key issue now becomes the positioning of the objects in the image. Since in real-life conditions, the pose of an object is almost always influenced to some degree by its environment, we believe that simply placing the item freely in 6D space as we did in the previous approach would lose information compared to a realistic placement. In an example setting where the objects are placed on a horizontal surface, this would constrain three degrees of freedom for each object: the vertical position relative to the surface, and the two rotations around the axises that determine the surface itself. Thus our objective is, for each background image, to start from the pose of the surface relative to the camera, and from there generate a realistic pose for each dataset object.

This pose is generated using the composition in sequence of three roto-translations:

1. An initial transformation (t_s, R_s) from the camera frame to the surface's reference frame.

2. A second transformation (t_r, R_r) that shifts the object from the surface frame to a random position and rotation.
3. A final correcting transformation (t_c, R_c) that takes into consideration the object's geometry to obtain a realistic placement.

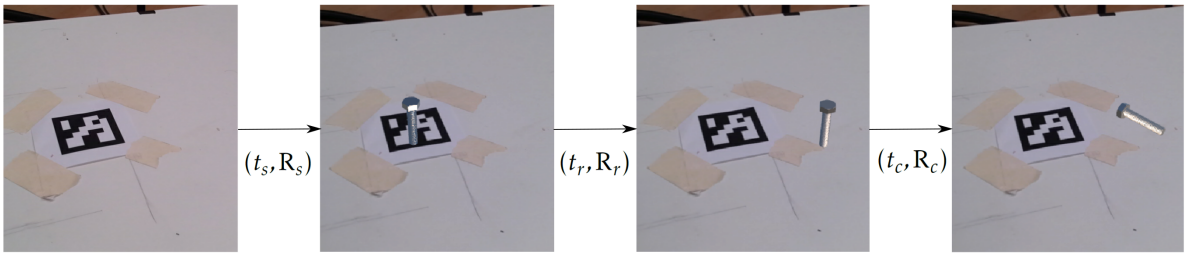


Figure 3.2: Visualization of the three roto-translations used to obtain a realistic placement.

We can obtain the initial transformation (t_s, R_s) by preparing the testing surface with an ArUco marker. In this manner, by capturing a video of the surface, eliminating off-center and blurry frames, and undistorting the resulting images, we obtain backgrounds for our scene that are then associated with the pose of the marker, as previously described in section 1.1.

We then compute values of t_r and R_r considering the degrees of freedom afforded to each object by the surface: that it is free to translate along the surface's x and y axes, and to rotate around its z axis.

$$t_r = \begin{bmatrix} x_r \\ y_r \\ 0 \end{bmatrix}, \quad \mathrm{R}_r = \begin{bmatrix} \cos\theta_r & -\sin\theta_r & 0 \\ \sin\theta_r & cos\theta_r & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

x_r, y_r, and θ_r can be extracted from pre-defined probability distributions; in our case three uniform distributions $U(x_{min}, x_{max})$, $U(y_{min}, y_{max})$, and $U(\theta_{min}, \theta_{max})$.

The final correction transformation (t_c, R_c) differs based on the object geometry, thus must be computed individually for each object. For example, if we consider the M6x30 screw, (t_c, R_c) is given by a translation z_c along the z-axis and a rotation by θ_c around the y-axis:

$$t_c = \begin{bmatrix} 0 \\ 0 \\ z_c \end{bmatrix}, \quad \mathrm{R}_c = \begin{bmatrix} \cos\theta_c & 0 & -\sin\theta_c \\ 0 & 1 & 0 \\ \sin\theta_c & 0 & \cos\theta_c \end{bmatrix}$$

The resulting transformation is shown in figure 3.3, while z_c and θ_c depend on the dimensions of the screw as follows:

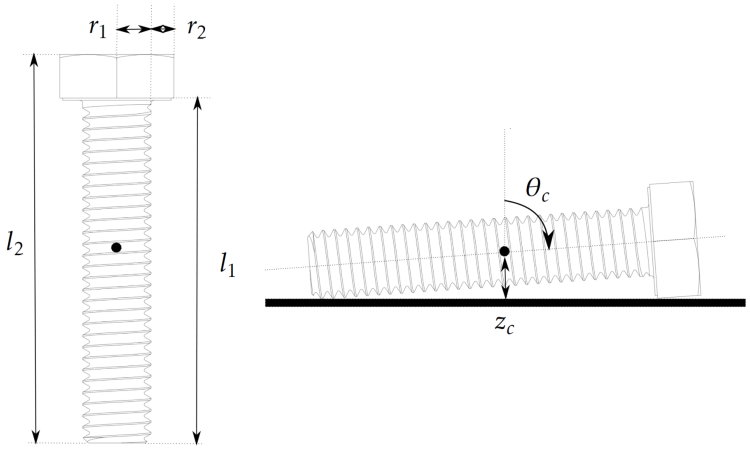


Figure 3.3: Dimensions and pose corrections for the M6x30 hexagonal head screw.

$$\theta_c = \frac{\pi}{2} - \arctan\frac{r_2}{l_1}$$
$$z_c = r_1 \sin\theta_c + \frac{1}{2} l_2 \cos\theta_c$$

One thing to note is that if an object can have multiple positions on the surface, we consequently have multiple correction transformations to choose from. For example, each screw could be on its side or on its head, which implies a choice between two sets of t_c, R_c.

Once we have the three transformations (t_s, R_s), (t_r, R_r) and (t_c, R_c), the final pose (t, R) in camera reference is computed as:

$$t = t_s + \mathrm{R}_s t_r + \mathrm{R}_s \mathrm{R}_r t_c$$
$$\mathrm{R} = \mathrm{R}_s \mathrm{R}_r \mathrm{R}_c$$

Eventual intersections between objects resulting from successive placements are resolved using a simple brute-force approach: whenever a placement would generate a collision,

the process is re-attempted from the second step, with a limit on the maximum number of attempts allowed.

With this method, for each background we can quickly generate a series of training images with associated ground truths. While we considered the particular situation of a set of objects placed on a flat surface, the three steps of this approach can be applied to other conditions. In general, these steps are:

1. The identification of the area of interest, and primary placement inside that area.
2. A random transformation within the area of interest, dictated by the degrees of freedom present.
3. A final transformation dictated by the specifics of the object to be placed.

3.2.3. Data Augmentation and Network Training

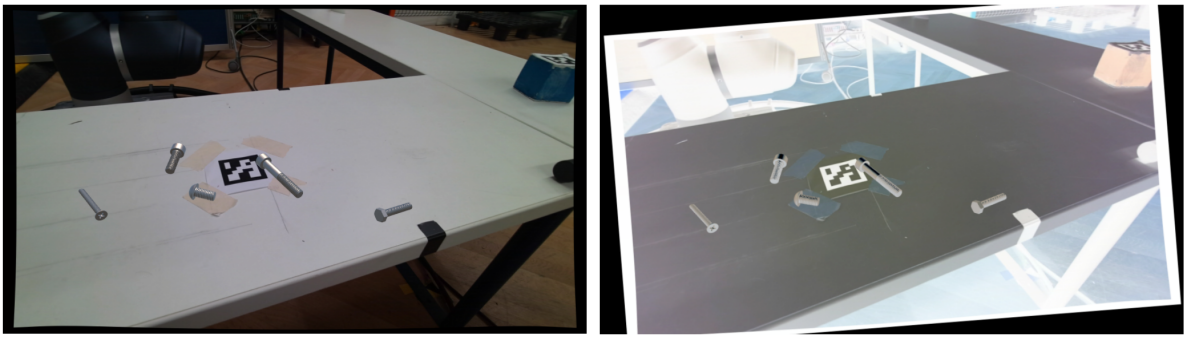


Figure 3.4: An example of a training image before and after data augmentation.

The dataset generated with the previous methodology has two major weaknesses. First, there is a limited number of background images, which means that the model has a finite number of camera positions to learn from. This may lead to overfitting and unreliable results for positions that don't have sufficient representation. Second, it is difficult to randomize light intensity and color for the background images inside Unity, as it is decoupled from the 3D models.

We can remedy these issues using data augmentation. This is a technique that involves applying random changes to data during training, similarly to how domain randomization would apply them during dataset generation. EfficientPose already provides two data augmentation methods: 6 Degree-of-Freedom augmentation and color augmentation. 6 Degree-of-Freedom augmentation involves randomly rescaling and rotating the input image and consequently adjusting the ground truth, so as to greatly increase the

number of possible poses each image can provide. Color augumentation instead implements RandAugment[8] to change the color and grain for the entire image. Applying both these methods results in images such as the one depicted in figure 3.4, conveniently fixing the issues of our dataset.

Other training parameters are identical to the previous attempt with the fully synthetic dataset: 100 epochs, with an 80% learning rate reduction if the model stagnates for 5 epochs. The results of training a network on this dataset are presented in section 5.1.

3.3. Semantics Understanding

3.3.1. Motivations and Objective

In many applications, raw sensor data on its own is not sufficient to perform a task. For example, when using high-level planning algorithms, it is often necessary to provide information on the overall semantic state of a scene. The issue then becomes how to deduce this state using the available data, which in our case is the output of a pose estimation network.

The example situation we will be considering is the assembly of a workpiece starting from its components. By tracking the pose of each individual component, we attempt to track the state of the overall assembly task. This could be necessary information for planning out a sequence of actions that would then lead to the task's completion.

3.3.2. Dataset Generation and Training

The first step to work with the neural network is again the generation of the datasets required for training and evaluation. For our task, we are considering the assembly of a set of modular button boards. We have two boards, one with two slots, and one with three slots. These slots can be filled in any order with one of three buttons: a larger safety button, and two smaller buttons with different designs on their faces, but identical shape. The CAD models for these objects were available online from the supplier's website, and we used Blender to color them appropriately. Renders for these objects are depicted in figure 3.5.

To generate the dataset images we exploited the previously mentioned Augmented Reality method, however we have a few noticeable differences that require consideration. The first obvious one is that we would like to represent the buttons not only while they are freely placed on the table surface, but also when they are slotted into a board, as can be see in

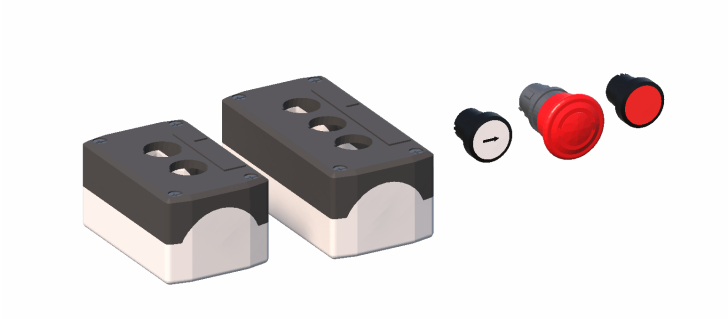


Figure 3.5: Orthographic rendering of dataset objects: two modular button boards and three buttons.

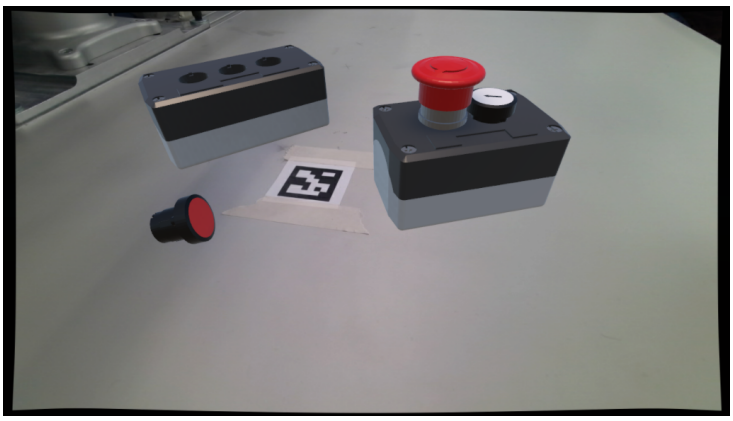


Figure 3.6: Sample image generated for the set of modular buttons and boards in figure 3.5.

figure 3.6. We compute these transformations in advance and simply apply them when necessary. One important thing to note is that these roto-translations depend on the shape of the button: thus the larger safety button will require different values compared to the other two.

The second issue deals with the two smaller buttons with identical shape. These buttons are distinguishable only by the different designs on their faces, but when placing them randomly, there is a good chance that these faces are not visible. This leads to a situations where it is impossible to differentiate between the two, causing a drastic drop in performance, as during training the network will percieve a large number of false positives.

To solve this issue, we add a fictional object to the dataset, the "unidentified button". In particular, we categorize buttons without their unique face visible as a new object class, since it is impossible for the network to directly identify them as one type or the other.

We can use a simple geometric method to determine if a button's face is visible during generation, explained in figure 3.7. Considering the origin of the camera reference frame

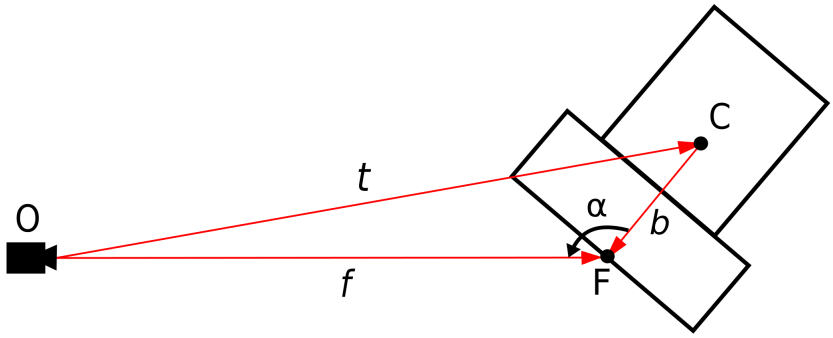


Figure 3.7: Schematic representation of the button relative to the camera, and of the variables used when evaluating occlusion of the button face.

$O = [0, 0, 0]^T$, the button's pose is given by the translation vector t and the rotation matrix R. t also indicates the position of the center of the button C, due to it being the origin of the button's 3D model. If we then consider the vector $b = [b_x, b_y, b_z]^T$ indicating the position of the center of the button's face F in the button's reference frame, the position of this point in the camera frame is given by:

$$F = t + \mathrm{R}b = f$$

We can then compute the amplitude of the angle $\alpha = C\hat{F}O$ by applying the law of cosines:

$$t^T t = b^T b + f^T f - 2||b|| \cdot ||f|| \cdot \cos\alpha$$

By making α explicit in this equation we obtain:

$$\alpha = \cos^{-1}\left(\frac{t^T t - b^T b - f^T f}{2||b|| \cdot ||f||}\right)$$

It is simple to verify geometrically that the button's face is occluded whenever $\alpha < 90°$, and viceversa. To reduce edge cases where the button's face is barely visible in the proximity of $\alpha = 90°$, we introduce a small buffer angle β, usually around $5°$. Thus we categorize a button as unrecognizeable whenever $\alpha < 90° + \beta$, overwriting its class in the ground truth with the "unknown button" class. The complete method for placing an object is therefore described with a flowchart in figure 3.8.

We decided to generate 20'000 images in this manner, where 18'000 are used for training

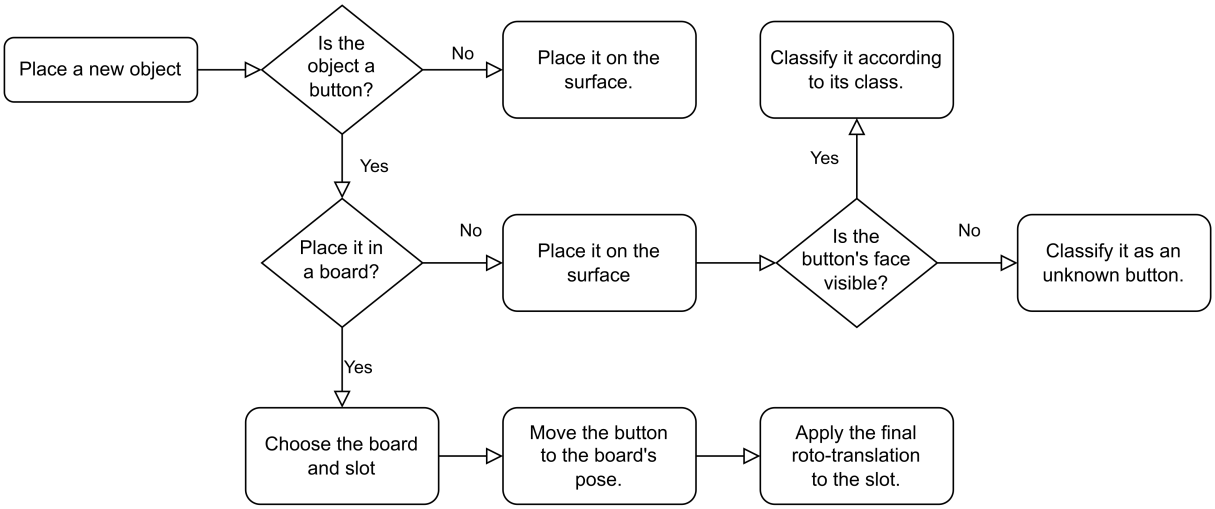


Figure 3.8: Flowchart representing the algorithm for placement of dataset objects, when placing buttons inside of slots and checking for face occlusion.

and 2'000 for verification. The increased dataset size is due to us giving more representation to each of the unique positions each object can have. We subsequently reduced the number of epochs to 40, and the patience for learning rate reduction to 3 epochs. All other parameters are identical to our previous attempts.

Results for training a model on this dataset can be found in section 5.1.

3.3.3. Semantic Meaning Extraction

Now that we have a dataset and a model trained on it, we can begin the task of creating a method to identify the state of the overall scene. There are two pieces in this assembly: the buttons and the boards they are to be slotted in. The problem now becomes: given the position of a button and the position of a board, how does one determine whether the button is slotted into the board, and in which slot?

We handle this using a *threshold comparison* approach, described by the following three steps:

1. For each button and slot, we compute a *distance metric* that represents the button's "distance" from that slot.
2. We compare this metric with a *distance threshold*: if it is less than this value, the button is considered a viable candidate for filling the slot.
3. We resolve conflicts between multiple buttons competing for the same slot and vice-versa.

For *distance metrics*, we considered two possible candidates: Center-to-Center distance and Average Symmetric Distance.

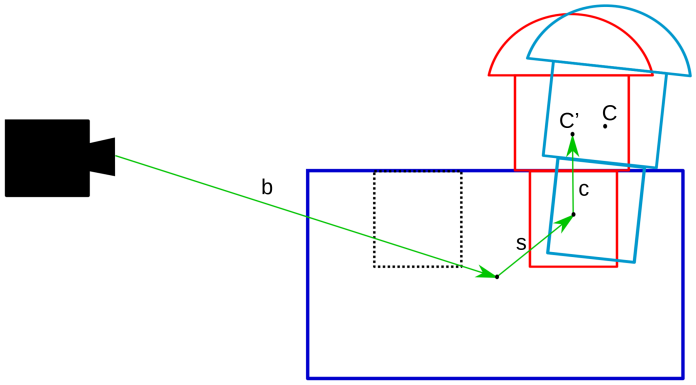


Figure 3.9: Depiction of the method for obtaining a button's *reference position*. The network's estimation of the position of the button and board is in blue, while the *reference position* is in red.

Center-to-Center is the distance between the estimated position of the center of the button C, and a *reference position*, representing an estimation of the position this center would have if the button was already in the slot, C'. As depicted in figure 3.9, while the first position is a direct output of our network, the second one must be obtained from the succession of three roto-translations: one from the camera reference to the estimated position of the board, b, which is a direct output of the network; a second to the position of the slot relative to the board, s, which is known for each slot; and the last one from the position of the slot to the position of the hypothetical button center, c, which varies based on the geometry of the button. The Center-to-Center distance is then given by $||CC'||_2$.

The Average Symmetric Distance is instead identical to the AD-S computed as a metric during training and evaluation. Considering if the estimated pose of the button is given by (R, t) and the *reference position* is given by (R', t'), AD-S is computed as:

$$\text{AD-S} = \frac{1}{n} \sum_{x_1 \in M} \min_{x_2 \in M} ||(\mathrm{R}x_2 + \mathrm{t}) - (\mathrm{R}'x_1 + \mathrm{t}')||_2$$

where M is the set of the points composing the button's 3D model, and n is the number of points considered. This metric is further explained in section 4.1.2. While this metric has the advantage of considering differences in rotation, which Center-to-Center is unable to do, it is also much more computationally demanding, depending on the number of points considered.

Once we have a *distance metric*, it is necessary to develop a conflict resolution algorithm. This is of fundamental importance for higher values of the threshold, since multiple slots will be within range of the same button or multiple buttons will be in range of the same slot, as depicted in figure 3.10.

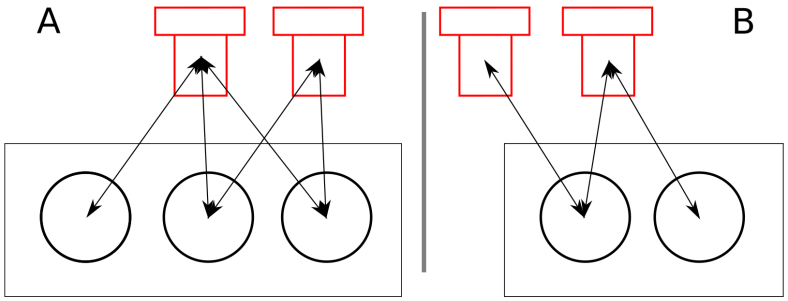


Figure 3.10: Schematic depiction of two hypothetical situations that would generate conflicts by placing multiple buttons within the threshold of a single slot or vice-versa.

For this purpose we introduce a "double check" method that provides excellent results with minimal complexity. This method is composed of three steps:

1. For each button, we assign it to the closest slot within the threshold, if there is one.
2. For each slot, we assign it to the closest button within the threshold, if there is one.
3. For each assignment, it is confirmed only if it is reciprocated, and otherwise it is ignored.

An assignment from a slot to a button is considrered reciprocated only if the button is also assigned to the slot, and vice-versa.

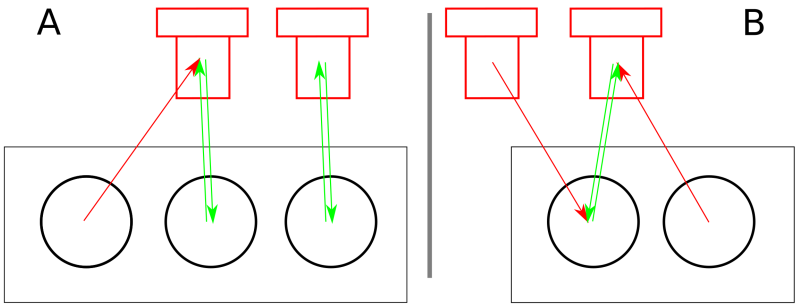


Figure 3.11: Schematic depiction of the resolution of the conflicts previously depicted in figure 3.10. Green arrows represent reciprocating assignments, which are confirmed, while red arrows represent non-reciprocating assignments, which are ignored.

In this way we can resolve most possible ambiguites that may result during evaluation of the semantic state.

4 | Experiments and Evaluation Metrics

In this chapter we will go over the various experiments and metrics used to evaluate the performance of our methods. This chapter is subsequently divided into three main sections.

In the first section we will show the metrics used to describe the performance of the pose estimation network. In the second section we will then show how we evaluated the performance of our semantic meaning extraction strategy. Finally, in the third section we will discuss the experimental setup we used to test the performance of our complete model in a real-life robotics application.

4.1. Evaluation Metrics for Pose Esitmation and Object Detection

Most pose estimation and object detection methods share a common set of metrics on which their performance is evaluated. These are namely Average Precision for object detection strategies, and Average Distance and ADD for pose estimation strategies. In this section we will describe each metric, its meaning and how it is computed.

4.1.1. Average Precision

The performance of object detectors and 2D bounding box regressors is usually evaluated using Average Precision (AP), which is a descriptor of the reliability of a method's predictions. It exploits the intersection over union (IoU), computed as:

$$\text{IoU} = \frac{B_{GT} \cap B_P}{B_{GT} \cup B_P}$$

where B_{GT} is the area of the ground truth bounding box and B_P is the area of the

network's predicion. A prediction is considered true if its IoU is greater than a *threshold*; based on this, we can generate the model's confusion matrix, as described in table 4.1.

	Actual Positives	Actual Negatives
Predicted Positives	True Positives (TP)	False Positives (FP)
Predicted Negatives	False Negatives (FN)	True Negatives (TN)

Table 4.1: Generation of the confusion matrix.

This matrix is the basis for the definition of the precision and recall metrics. Precision is an indicator of how well the model avoids false positives, while recall is an indicator of how well a model avoids false negatives. They are computed as follows:

$$\text{Precision (P)} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
$$\text{Recall (R)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Precision and recall both depend on the value of the IoU *threshold*: larger values will result in a more restrictive model, thus less false positives and more false negatives, high precision and low recall, while smaller values will result in the opposite: more false positives, less false negatives, low precision and high recall.

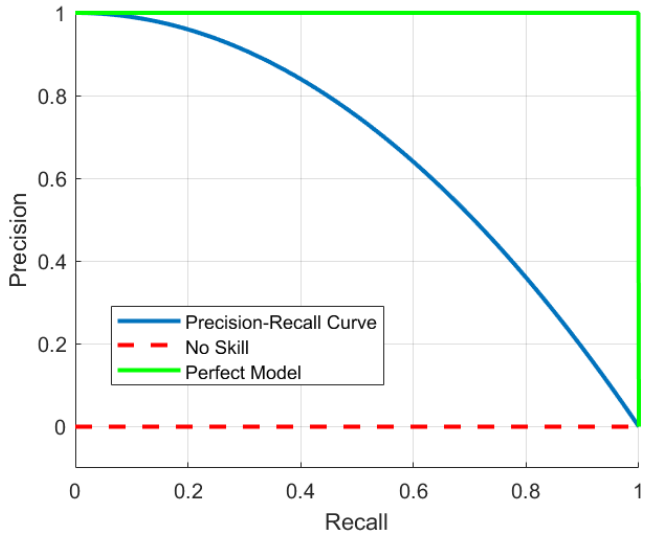


Figure 4.1: Example of precision recall curves. The red line indicates a model with zero precision; the blue line a typical model; the green line is the ideal behavior with maximum precision and recall at all times.

It is common practice to plot the precision as a function of the recall in what is called

a precision-recall curve, $P = f(R)$. Each point in the curve represents a value of the *threshold*, corresponding to its own confusion matrix and subsequent precision and recall metrics. An example plot is shown in figure 4.1.

At this point we can describe the Average Precision (AP) as the mean value of the precision, corresponding to the area under the precision-recall curve:

$$\text{AP} = \int_0^1 f(R)dR$$

Therefore it is a real value between 0 and 1, with 0 representing the model with zero precision, and 1 representing the ideal behavior.

4.1.2. Average Distance and ADD

Evaluation of pose estimation methods is almost exclusively done using the ADD metric, and by extension the Average Distance (AD). While the first is a ratio of correct estimations against total estimations, the second is instead a representation of the average error the network commits when estimating the pose of an object.

Given n points belonging to the 3D model M of an object, the AD represents the average of the distance between these points transformed according to the ground truth (R, t) and according to the prediction $(\hat{\text{R}}, \hat{\text{t}})$:

$$\text{AD} = \frac{1}{n} \sum_{x \in M} ||(\text{R}x + \text{t}) - (\hat{\text{R}}x + \hat{\text{t}})||_2$$

The ADD is then given by the percentage of correct poses given by the model. A pose is generally considered to be correct if its AD metric is less than 10% of the 3D model's largest dimension.

This metric however has serious issues when dealing with objects that have rotational symmetries, as these present no visual differences for multiple different orientations. For example, one image of the M6x30 screw we use for inferencing could correspond to six different poses, each varying 60°from the previous one. This means that the model will eventually stabilize at a value that minimizes the average error, which is ususally large.

To combat this issue, we use the Symmetric Average Distance (AD-S)[40] metric, defined as the average minimum distance between points in the predicted pose and the ground truth:

$$\text{AD-S} = \frac{1}{n} \sum_{x_1 \in M} \min_{x_2 \in M} ||(\text{R}x_2 + \text{t}) - (\hat{\text{R}}x_1 + \hat{\text{t}})||_2$$

This considers the distance from each point to its closest correspodent in the ground truth. Analogously to ADD, we then implement ADD-S as the percentage of correct poses.

We would like our model to obtain the highest possible ADD-S, however in an industrial environment it is important to also evaluate the AD-S. This is because for larger objects the ADD will tolerate greater estimation errors, as it is based on the diameter of the objects; these errors however may not be compatible with the precision required for a determined task.

4.2. Semantics Evaluation Methodology

To evaluate the semantic meaning extraction method, we must compare ground truth values for the semantic state of each scene, associated with its own image, with the outputs of our method.

Our strategy therefore is to save the semantic state for each image during dataset generation as the ground truth. We then run the trained model on the test dataset to obtain pose and class predictions, use these results to run our semantic meaning extraction method, and evaluate the results against the ground truth. In particular, we select a button-slot *distance metric* and *distance threshold* for each experiment, and compare the estimate of the state of each slot, for each board in each scene, against its true value.

However, an issue arises from this application: due to the symmetry of the boards, it is impossible to determine visually which slot is which. For example in figure 4.2, the ground truth is that there is a button in the first slot and the second slot is empty, however, the model may output that the button is in the second slot, while the first slot is empty. Visually, both of these interpretations are correct, since the board is symmetrical, but directly comparing the prediction with the ground truth results in two "false" values: a false negative for the first slot and a false positive for the second, resulting in an erroneous evaluation.

To counter this issue, for each board we take into account both the model's prediction, and its symmetrical, obtained by simply reversing the order of its slots. For the example in figure 4.2, the prediction is [empty, button], therefore its symmetrical is [button, empty]. We then only consider configuration with the greatest number of "true" values when compared with the ground truth. In the example shown in figure 4.2, since the symmetrical

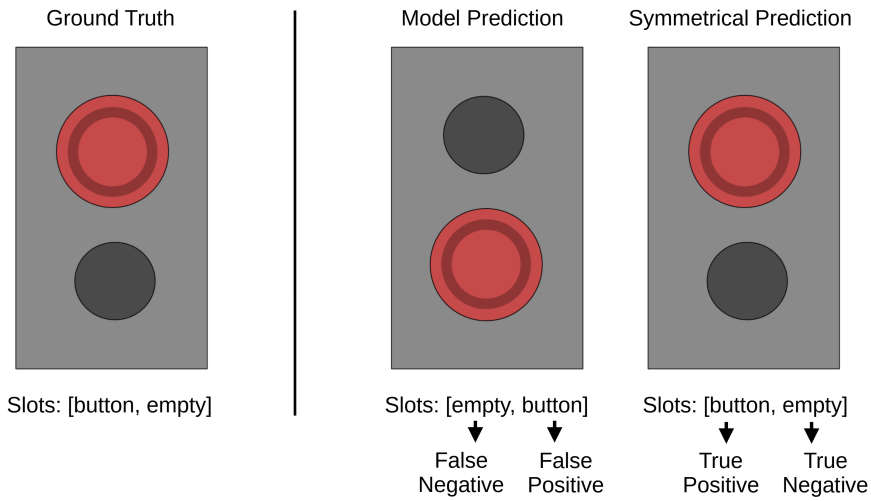


Figure 4.2: Schematic depiction of a board with a button slotted into the first slot. The model runs into issues because the prediction and its symmetrical are visually undistinguishable and give different results.

results in two "true" values, the final output of the comparison is two "true" values: a true positive and a true negative. These values are then summed for all estimations computed using a determined *distance metric* and *distance threshold*, and the sums are used to build a precision-recall curve, as described in the previous section.

The selection of the optimal *distance threshold* is done by considering the F1 score: a balanced function of precision and recall, computed as:

$$\mathrm{F1} = 2 \times \frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

We consider the optimal threshold to be the one that maximises this value.

4.3. Real-Life Experimental Setup

To test the effectiveness of our vision and semantics model, we implemented it in a real robotic setup where we have to complete a simple assembly task. We used a Doosan A0509s robotic manipulator equipped with a pneumatic gripper, and an Azure Kinect camera. We then re-implemented previously developed code for this system that generates behavior trees [7] to drive the robot based on previous demostrations of actions and their effects on the scene.

Our contribution is mostly in the perception phase of this architecture, where we perform three tasks:

1. From the RGB image provided by the camera, detecting objects of interest and estimating their position.
2. Understanding the state of the interactions between the detected objects.
3. Building a list of predicates that describes the scene.

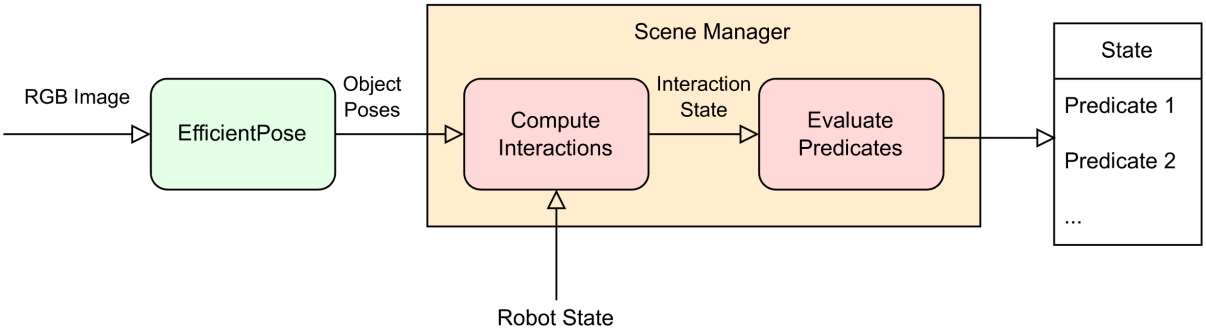


Figure 4.3: Schematic representation of the overall vision system.

While the first step is performed by the network, the final two are performed by what is known as the Scene Manager (see figure 4.3). The ultimate objective of the Scene Manager is to describe a scene using a set of predicates: first-order logic functions that can be either true or false. Their value is updated for each object whenever we obtain new information on the scene, with the ones resulting true then being compiled into the *state*. For our application, this is accomplished using the predicates in table 4.2.

Predicate	Description
IsGripperEmpty(gripper)	True when no objects are in the gripper.
IsGrasped(button, gripper)	True when the button is grasped by the gripper.
IsButtonInSlot(button, slot)	True when the button is inserted in the slot.
IsSlotEmpty(slot)	True when no buttons are in the slot.

Table 4.2: List of predicates used to describe the state in our application.

The experiments with the robot consist of two phases: *teaching* and *evaluation*.

In the *teaching phase*, we show the robot how to perform actions through kinesthetic demostrations. During these demostrations, the robot, moved by a human operator, will modify the environment and thus change the scene. The actions performed during this phase, can be divided into move actions and interactions, which in our case are limited to opening and closing the gripper. Move actions reference a destination position, which we save relative to a manually defined reference object. For example, if we want to pick up a button, we set the reference to the button itself: this way when we require repeating the same actions but the button is in a different position, we can compute the new position by changing the initial reference pose, so as to not lose in generality.

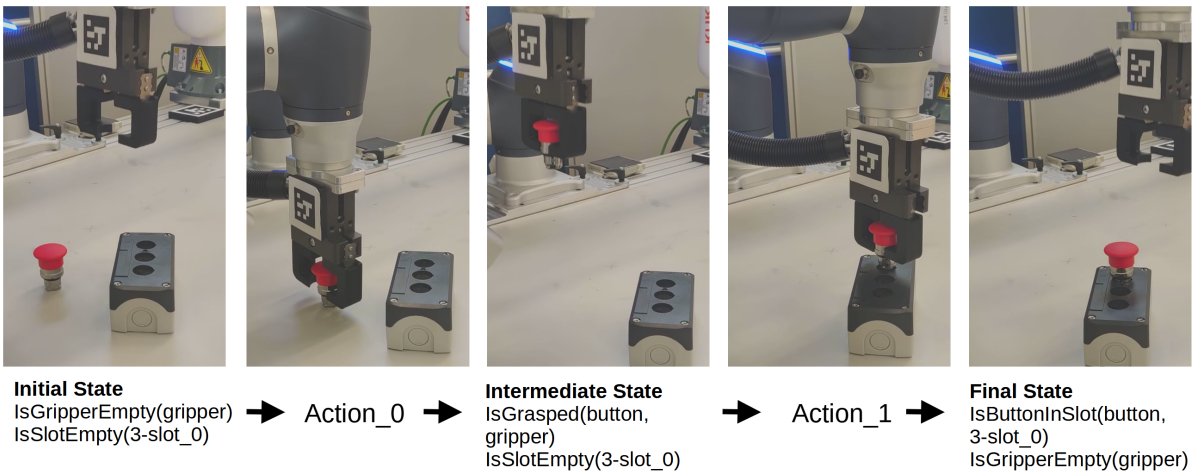


Figure 4.4: Evolution of the state and actions for the example task of picking up a button and inserting it into a slot.

By examining the differences between the state before and after the demonstration, we can use Planning Domain Definition Language (PDDL) [14] to generate a set of *preconditions* and *effects* expressed in predicate form, and combine them with the low level robotic actions to obtain a *skill*. For our example case shown in figure 4.4, we define two actions as shown below. The *domain* for this task then consists of the set of all defined objects, predicates and skills, as shown in figure 4.5.

```
(:action action_0
    :parameters(
        ?Gripper1 - Gripper
        ?SmallButton1 - SmallButton
    );end_of_parameter
    :precondition (and
        (is_gripper_empty ?Gripper1)
    );end_of_precondition
    :effect (and
        (is_grasped ?SmallButton1 ?Gripper1)
        (not (is_gripper_empty ?Gripper1))
        (increase (total-cost) 50)
    );end_of_effect
);end_of_action
```

```
(:action action_1
    :parameters(
        ?SmallButton1 - SmallButton
        ?Gripper1 - Gripper
        ?Slot1 - Slot
    );end_of_parameter
    :precondition (and
        (is_grasped ?SmallButton1 ?Gripper1)
        (is_slot_empty ?Slot1)
    );end_of_precondition
    :effect (and
        (is_gripper_empty ?Gripper1)
        (is_button_in_slot ?SmallButton1 ?
    Slot1)
        (not (is_grasped ?SmallButton1 ?
    Gripper1))
        (not (is_slot_empty ?Slot1))
        (increase (total-cost) 50)
    );end_of_effect
);end_of_action
```

In *evaluation mode* we instead specify what is known as the *problem*. This consists of

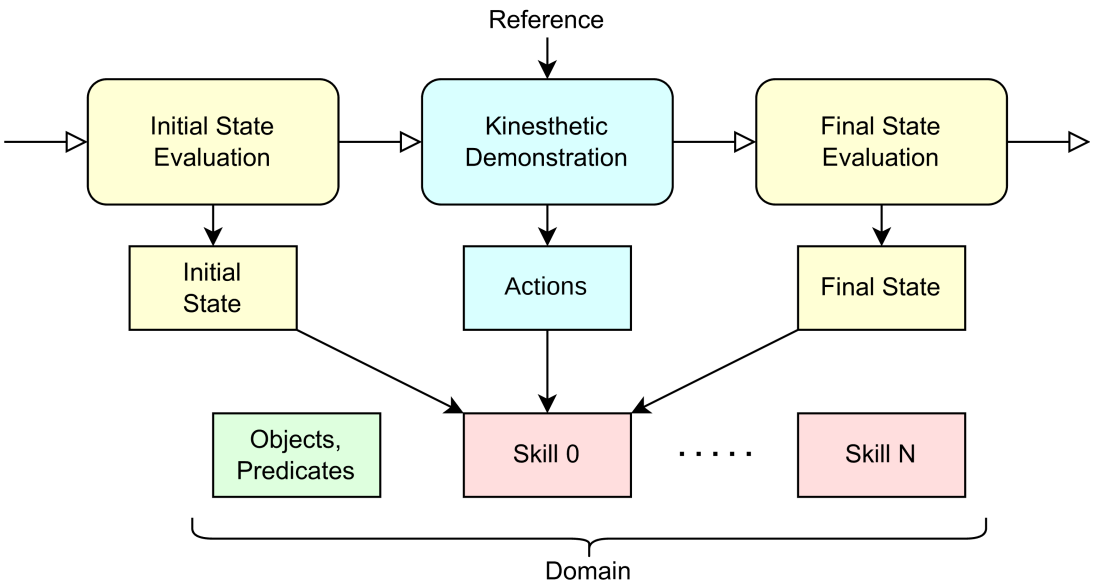


Figure 4.5: Construction of the domain in the teaching phase, through evaluation of the state before and after the kinesthetic demonstration.

a combination of the semantic meaning of the scene's initial state (*init*) and the final state we would like to achieve (*goal*). By providing a PDDL planner with a *domain* and *problem*, it will compute the ordered set of actions necessary to pass from the initial to the final states described in the problem.

A behavior tree is then built by combining the individual trees for each action, in the order defined by the planner. This process is demonstrated in figure 4.6. These trees, combined with our prior modelling of the task in the form of the domain and the problem, form a *knowledge base* that is fundamental for the task's execution. The robot's actions are then performed by evaluating the tree.

Apart from the introduction of the new vision system and predicates, the other main modification applied to the pre-existing code was the decision to evaluate the semantic state only at specific instants. Therefore instead of constantly computing the state of the scene, we evaluate it only at the beginning and at the end of each action. The reason for this is the susceptibility of the vision model to false positives for untrained objects. More information on this can be found in section 5.1.2, but essentially in this manner we ensure that the state is evaluated only when the gripper and human operator are outside of the camera's field of view.

Finally, we tested this setup by having it perform the simple assembly task of picking up a button in various positions and inserting it into various different slots on the two button boards.

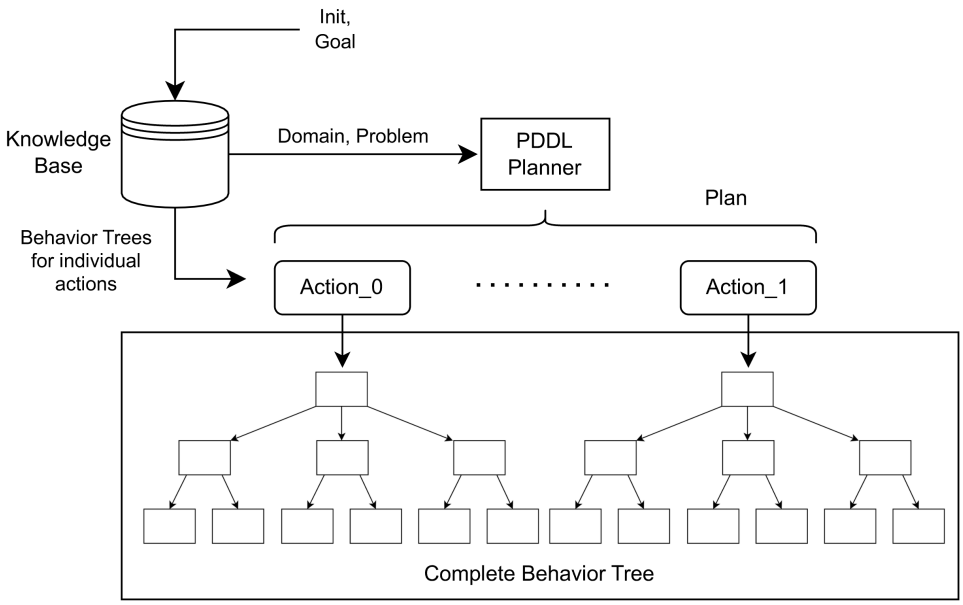


Figure 4.6: Construction of the behavior tree using the PDDL planner during the evaluation phase.

5 | Results

In this chapter we will go over the results we obtained by testing our methods.

In the first section we will show the results of training the EfficientPose network on our datasets, and subsequent observations. In the second section we test the performance of our semantic meaning extraction strategy. Finally, in the third section we show how effective our overall system is in the real-world robotics application.

5.1. Model Training Results

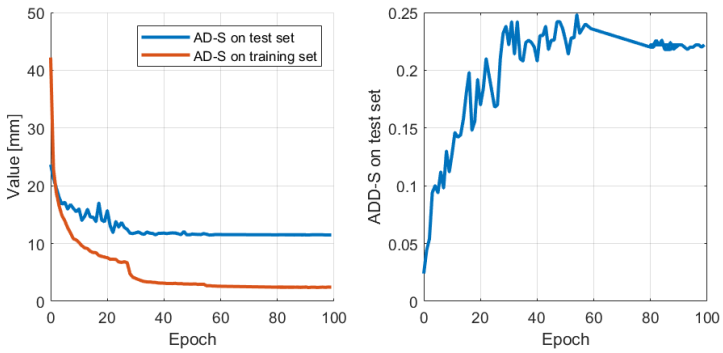
In this section we will show the results of training the EfficientPose network on the three datasets presented in the previous chapter: the fully rendered dataset representing an M6x30 screw, henceforth referenced as "ScrewDataset", the augmented reality dataset representing a set of screws, henceforth "ScrewPose", and the augmented reality dataset representing the set of buttons and boards, henceforth "ButtonPose".

We will then be comparing these results with those obtained by EfficientPose on other datasets, namely LINEMOD for single object estimation and Occlusion-LINEMOD for multi-object estimation.

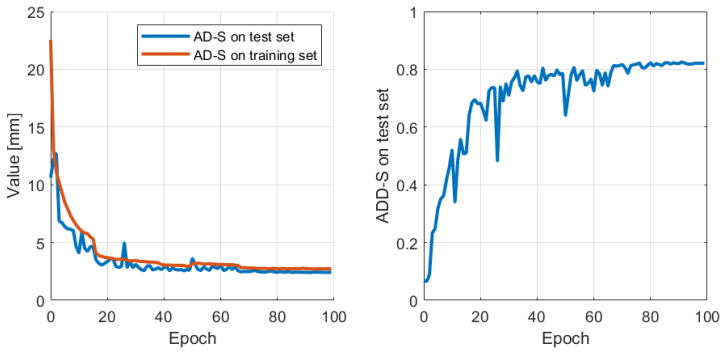
The evolution of the loss and ADD-S metrics during training for these three datasets is shown in figure 5.1. As we can see, the model's performance gradually improves over the training period, eventually stabilizing at a plateau for all three datasets. Also visible is the effect of the learning rate reduction, which has visible results when appiled.

As for training results, visible in figure 5.2, we will examine them independantly for each dataset below.

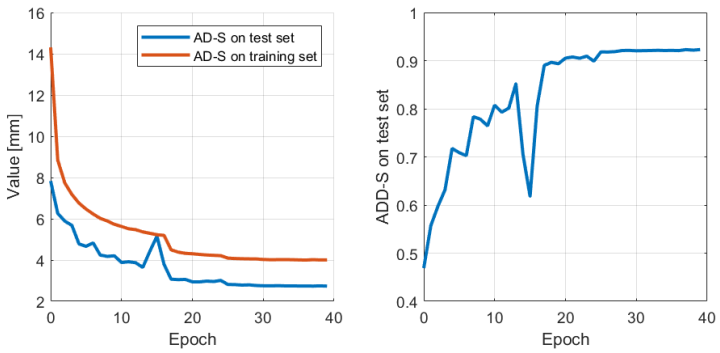
On ScrewDataset, after 100 epochs of training the model has a final ADD of 22.20%, with a peak value obtained during training of 24.8%, much lower than the 97.35% with $\phi = 0$ reported by EfficientPose on LINEMOD. We can hypothesize that the reason for this performance gap is that the rendered dataset is much more difficult than LINEMOD, since we are dealing with a very small, symmetric object hidden inside a chaotic, colorful



(a) Evolution using ScrewDataset for training.



(b) Evolution using ScrewPose for training.



(c) Evolution using ButtonPose for training.

Figure 5.1: Training progress for EfficientPose on the ScrewDatset, ScrewPose and ButtonPose datsets, represented as the evolution of the AD-S and ADD-S metrics in evaluation for each epoch.

background with widely differing light conditions. Another serious issue with this dataset is that the model is not able to bridge the reality gap: while testing in real-life scenarios, it failed to identify the screw in most conditions, let alone produce accurate estimations. This means that it generalises poorly outside of the simulated environment, making it essentially unuseable in real world applications.

On the flip side, the ScrewPose datset obtainined an average ADD-S of 82.05%, which is better than EfficientPose's 79.04% with $\phi = 0$ on Occlusion-LINEMOD, and comparable to its 83.98% with $\phi = 3$. This is a good result considering that the objects for our dataset are smaller, symmetric and all visually similar. Even though the Occlusion dataset is notoriously challenging, this anyways demonstrates the good performance of our own dataset.

Object	AP	AD-S [mm]	ADD-S
M6x30	0.9675	11.4921	22.20%

(a) ScrewDataset.

Object	AP	AD-S [mm]	ADD-S
M6x30	0.9399	2.1434	82.30%
M8x16	0.9538	1.9988	67.54%
M8x25	0.9645	2.1179	85.07%
M8x50	0.9880	3.4482	93.30%
Average	0.9615	2.4271	82.05%

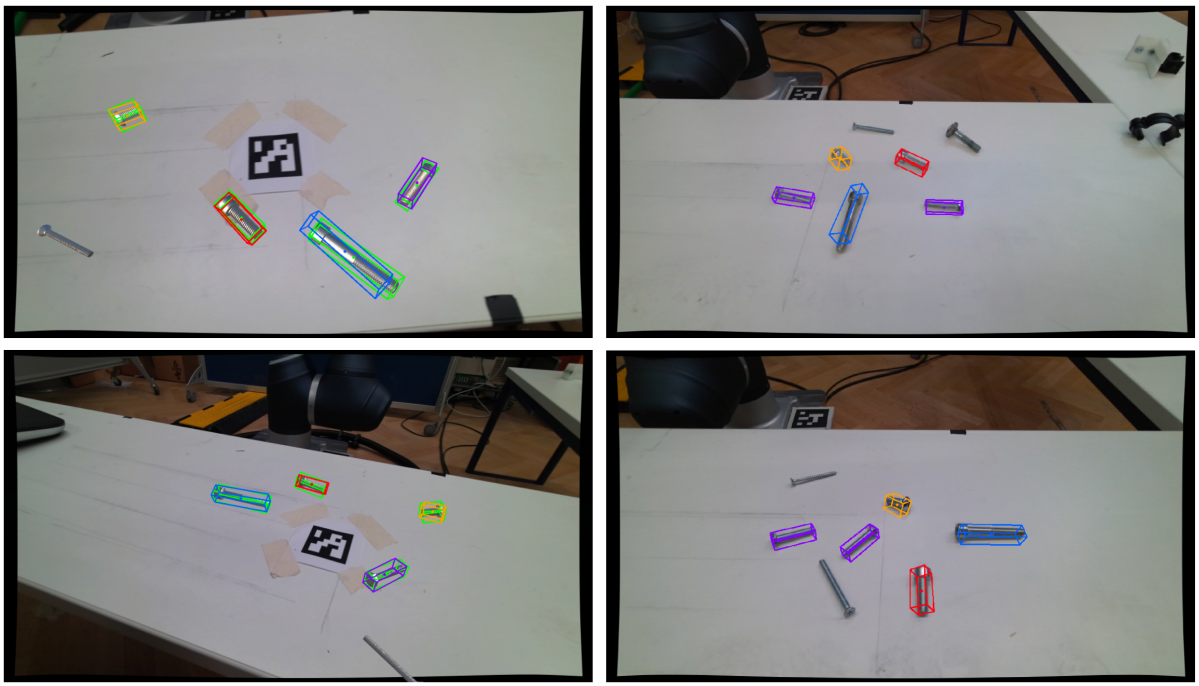
(b) ScrewPose.

Object	AP	AD-S [mm]	ADD-S
2-slot	0.9990	3.5420	99.90%
3-slot	0.9985	3.9304	99.85%
red button	0.9260	1.9825	86.01%
arrow button	0.9349	2.0497	86.05%
safety button	0.9962	2.6053	98.01%
unknown button	0.9561	2.4757	82.95%
Average	0.9685	2.76	92.13%

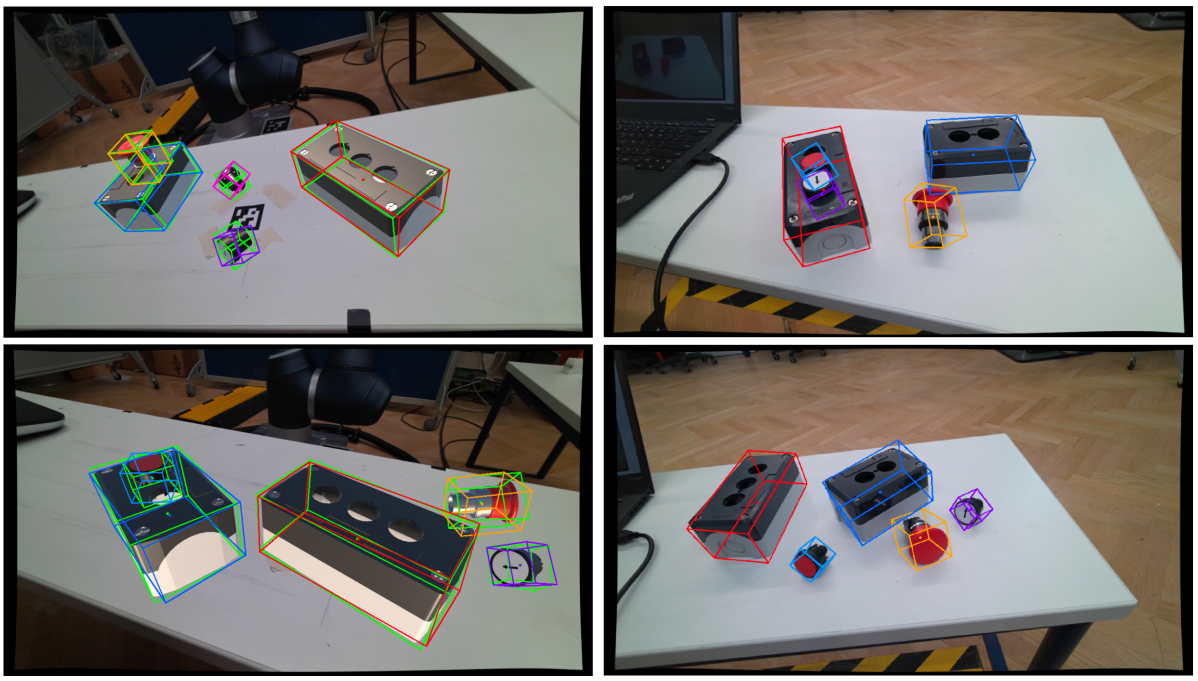
(c) ButtonPose.

Figure 5.2: Evaluation of the Average Precision, Average Symmetric Distance, and ADD-S metrics on the ScrewDataset, ScrewPose and ButtonPose datasets after training.

Finally, training on the ButtonPose dataset resulted in optimal performance for the boards, reaching over 99% ADD-S and AP for both. The larger safety button also obtained great results, with a 98% ADD-S, while the other buttons achieved more middling performances, but still better than the Occlusion-LINEMOD benchmark, showing that



(a) ScrewPose.



(b) ButtonPose.

Figure 5.3: Images displaying pose estimations from the network for the ScrewPose and ButtonPose datasets. The left images are part of evaluation and display ground truths with green bounding boxes, the right ones are captures from a real camera in the testing environment.

our approach is valid for more object sets.

The ScrewPose and ButtonPose datasets were also able to generalise to real-life conditions without noticeable losses in performance, as can be observed in figure 5.3.

5.1.1. Impact of object dimensions and distance

One noticeable result we observed after training was the impact that physical dimensions have on the final performance of the model for each object. Namely, larger and closer objects have much better performance than smaller and further objects. This is immediately noticeable in the ScrewPose dataset, where the larger M8x50 screw obtained the best results and the M8x16 obtained the worst.

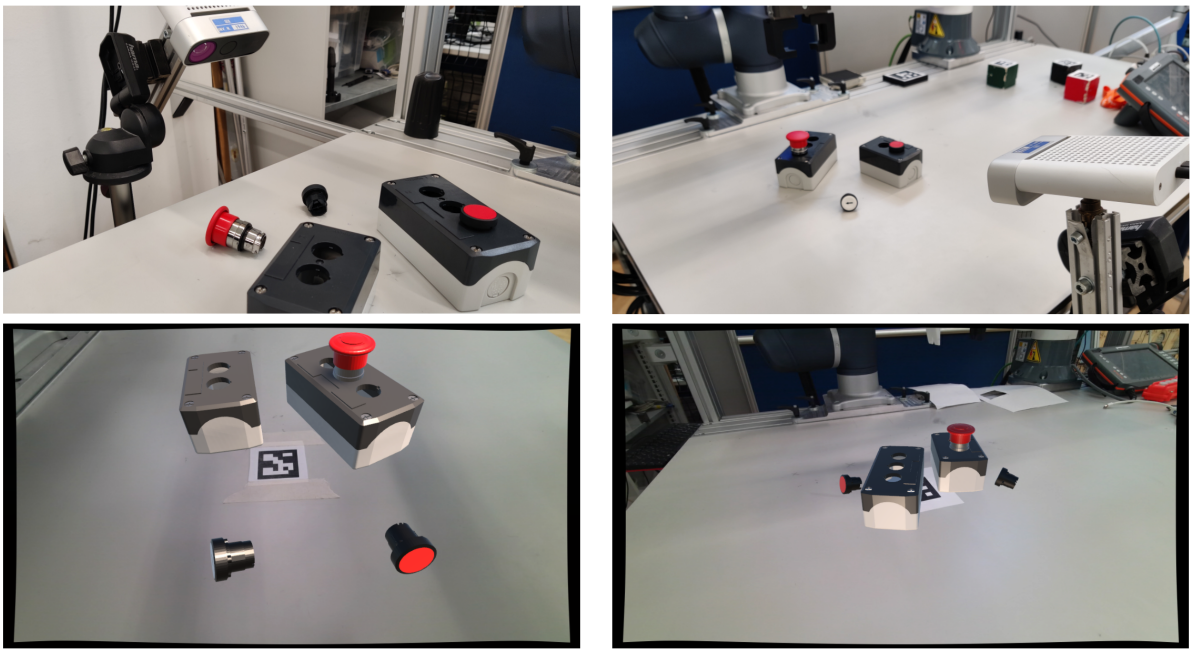


Figure 5.4: Photographs of the different capture positions for the ButtonPose-near (left) and ButtonPose-far (right) datasets, with a sample image presented for each.

To show how extreme these differences can be, we trained two additional models using new datasets based on the ButtonPose dataset. These two differ solely based on the background images: for the first one the backgrounds were captured from a further distance, while for the second one the backgrounds were captured from close up. For each of these conditions, we captured 50 backgrounds from a variety of positions, and fed them through our dataset generation pipeline. Sample images for these datasets and example positions are shown in figure 5.4. The close-up dataset (ButtonPose-near hereafter) ended up having an average camera-marker distance of 27.64 cm while the further away dataset

(ButtonPose-far hereafter) had an average distance of 49.33 cm.

We can then compare the performance of the models trained on the two different datasets, shown in figure 5.5. As can be seen, the near dataset obtained significantly better results than both the far dataset and the regular dataset.

Object	AP	AD-S [mm]	ADD-S
2-slot	0.9994	2.6850	99.94%
3-slot	1.0	2.8913	99.95%
red button	0.9663	1.3550	95.84%
arrow button	0.9729	1.4384	96.47%
safety button	1.0	1.7229	99.84%
unknown button	0.9948	1.3384	98.74%
Average	0.9889	1.9052	98.46%

(a) ButtonPose-near.

Object	AP	AD-S [mm]	ADD-S
2-slot	1.0	2.9985	99.90%
3-slot	1.0	3.1377	99.95%
red button	0.5477	4.3679	29.57%
arrow button	0.4902	4.4586	22.06%
safety button	0.9381	4.3261	79.49%
unknown button	0.7489	3.7920	44.68%
Average	0.7875	3.8468	62.61%

(b) ButtonPose-far.

Figure 5.5: Evaluation of the Average Precision, Average Symmetric Distance, and ADD-S metrics on the ButtonPose-near and ButtonPose-far datasets after training.

We can hypothesize that the reason for such a large gap between the two models lies in the input resolution of the network, which for $\phi = 0$ is set at 512 pixels. This makes it much more difficult for the network to make out fine details at a distance; in particular it would struggle to distinguish between the arrow button and the red button, since their faces would appear similar. This issue could therefore probably be alleviated by using higher values of ϕ.

5.1.2. False Positive Issues

A noticeable issue we encountered when testing our models in real world conditions was the difficulties they expressed in dealing with objects that are not present in the original dataset. In almost all cases, introduction of a never-before-seen object results in multiple false positives, as shown in figure 5.6. This is because the model is not trained to "ignore"

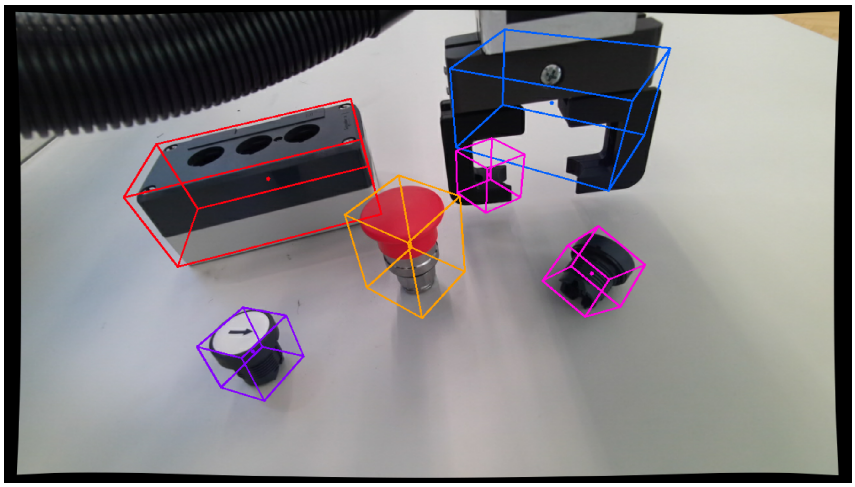


Figure 5.6: An example of how introducing untrained objects results in false positives.

these objects, and thus attempts to classify them according to what it effectively "knows".

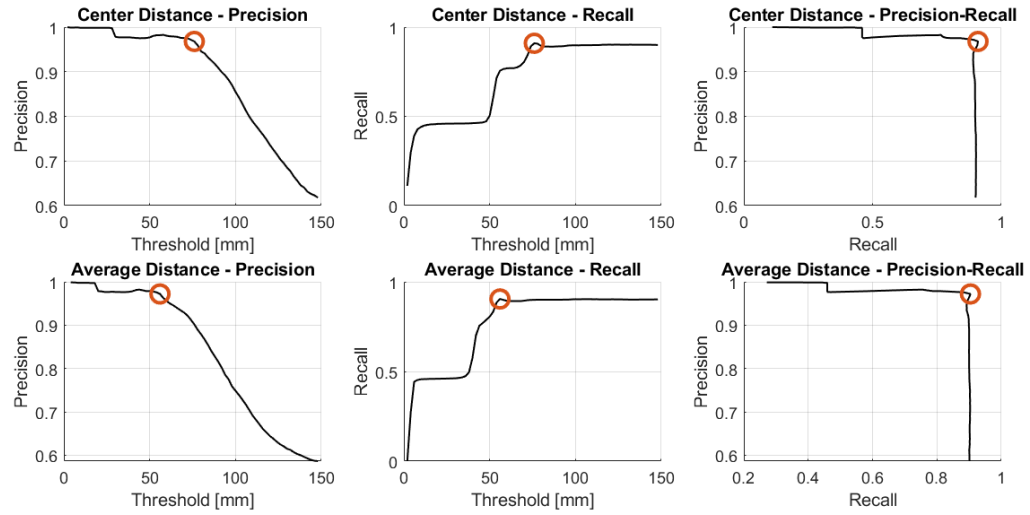
There are ways to approach and mitigate this issue. For example, if there is a high probability that a certain object that should not be tracked by the model will appear frequently in the scene, one could include that object within the dataset generation method without labelling it. In this manner, since the object would appear with a certain frequency during training, eventual false positives resulting from its presence would be recognized as such, and the model's behavior corrected.

We tested this method with the ScrewPose dataset: since all of its objects have a similar shape, a model trained on ScrewPose would tend to categorize any long and thin object as a screw, making it especially susceptible to false positives. However, introducing a decoy screw into most images, without labelling it as a dataset object, led to the model subsequently ignoring its appearance in most scenes. It was also able to generalise this behavior to a certain extent, ignoring other never-before-seen screws when they were introduced into the scene (see figure 5.3). Therefore in a case such as the one represented in figure 5.6, if we were certain of the appearance of the gripper in many scenes, it would be worthwile to include its 3D model in some training images, so that the network could effectively learn to ignore it.

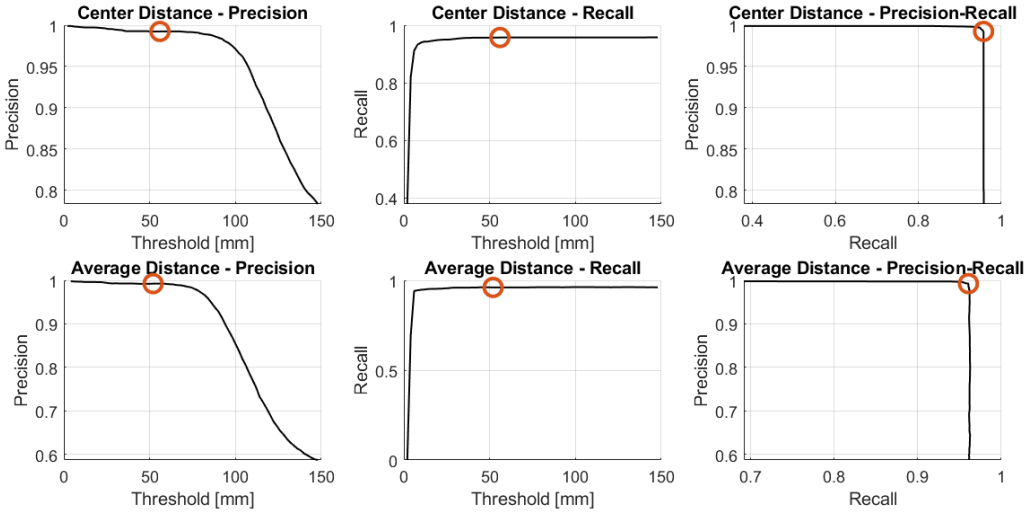
We hypothesize that by including a general enough set of decoy items in the training images, the model could then generalize this behavior to a wider variety of previously unseen objects. However, due to the nature of black-box methods, more research is necessary to verify whether this is feasible.

5.2. Semantic Meaning Extraction Results

In this section we will evaluate the performance of our semantic meaning extraction method. The precision-recall curves in figure 5.7 were computed using the methodology described in section 4.2 on the ButtonPose and ButtonPose-near datasets. The optimal thresholds and F1 scores are laid out in figure 5.8.



(a) ButtonPose



(b) ButtonPose-near

Figure 5.7: Precision and recall for both distance metrics on the ButtonPose and ButtonPose-near datasets. The point with the best F1 score is highlighted with a red circle in both cases.

Overall our method shows promising results, with precision-recall curves that are similar

Metric	F1	Threshold
AD	0.9369	56mm
CC	0.9255	78mm

(a) ButtonPose

Metric	F1	Threshold
AD	0.9763	52mm
CC	0.9746	56mm

(b) ButtonPose-near

Figure 5.8: Optimal F1 scores and thresholds for both the Average Distance (AD) and Center-to-Center (CC) metrics on the ButtonPose and ButtonPose-near datasets.

to the ideal, and high F1 scores.

We observe that our application obtained no great advantage in using Average Distance over Center-to-Center distance. Given the additional complexity and computation time, the increase in performance is not significant: for reference, computing the values for figure 5.7 took 23.67 seconds using Center-to-Center, and 2650.34 seconds using Average Distance, making the second approximately 112 times slower.

We also obtained high values for the optimal *distance thresholds*, above 5 cm in all testing conditions. We consider this value as the threshold to determine whether a button is in a slot at all, while the slot itself is selected using the conflict resolution strategy previously described.

We hypothesize that the reason for such a high value for the threshold lies in the dataset generation algorithm: by stating that the buttons can exclusively be either inside a slot or placed on the surface, we are in fact considering an ideal situation where a theoretical manipulator does not commit any errors in picking up and inserting the buttons. If failed robotic attempts in performing this task were considered in the dataset, it is likely that the optimal threshold would be lower, and the Average Distance method, being more sensible to situations with different rotations, would likely give better results than Center-to-Center.

Finally, the value of the threshold is also influenced by the probability distribution used for generating the poses for the button dataset, which in our case being a uniform distribution, resulted in a more spread-out placement, thus a higher optimal threshold. This is further exasperated by the brute-force collision avoidance strategy implemented within the placement algorithm: if a placement attempt would generate an intersection with an already placed object, the placement is simply re-attempted from scratch. This naturally results in less conditions where dataset objects are in close proximity, and therefore in a higher threshold.

Button Type	Small	Large
Total Attempts	10	10
Successful Pick-Ups	7	10
Successful Insertions	5	7

5.3. Real-World Application Results

As previously described in section 4.2, we tested the performance of our vision method in a real-world application where the robot had to pick up a button from the ButtonPose dataset and insert it into a slot on one of the button boards.

In our testing, the robot was able to pick up the smaller buttons seven times out of ten, while it was able to pick up the larger button in all test cases. Out of the times it was able to pick up a button, the following insertion was performed correctly in 71% of cases.

The main issue encountered during testing was the prevalence of small errors in the rotation of the gripper relative to the button. These errors are usually in the neighborhood of $\pm 5°$, but they are noticeable and can lead to mistakes, as the button may be gripped in an awkward manner.

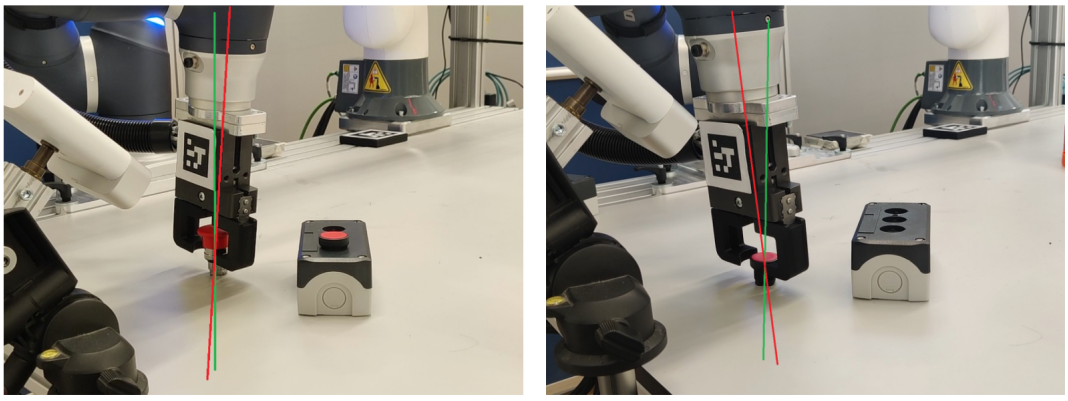


Figure 5.9: Examples of small rotation errors when attempting to pick up a button with the robotic arm. The green line indicates the button's central axis, while the red line is the robot's.

Positioning relative to the boards was instead very accurate, and errors is the insertion phase were mostly due to mistakes in grasping the necessary buttons. Overall the robot performed much better when dealing with the larger buttons and boards than when dealing with the smaller buttons, making its reliability very dependant on the accuracy of the pose estimation network for the specific object.

6 | Conclusions and Future Developments

In the field of robotics, and automation in general, the interactions between the robot and the environment are of fundamental importance. The perception of the robot's surroundings in particular requires the system to have a certain number of sensors that provide it with data from the environment, which it then processes to obtain the information necessary to perform its purposed task.

RGB Color cameras, while being relatively cheap and easily obtainable sensors, introduce a series of difficulties that limit their applicability in real-time control applications: they produce a huge amount of data that does not directly reflect any physical measurments, instead being linked together through complex interactions that are difficult to express analyitically. However, neural networks, and convolutional neural networks in particular, are perfectly suited to cover these issues, since they work efficiently on large amounts of inputs in parallel, and can model unknown and complex functions with proper training.

Due to these advantages, machine learning approaches have been widely applied to image processing tasks. In particular, the fields of object identification and pose estimation have gone through rapid development, and there has been an increased interest in applying these techniques to tasks for industrial and collaborative robotics. However, machine learning approaches introduce a series of additional challenges, including but not limited to: the necessity of acquiring vast amounts of labelled data for training, the opaqueness of trained networks to conventional analysis, and the necessity of abstracting low-level data into high-level information for use in planning algorithms.

To tackle these issues, we began by developing a dataset generation algorithm to simplify the data acquisition phase. While unfortunately a more general approach involving fully simulated training images failed to produce acceptable results, we developed a more specific technique that involves realistically rendering objects of interest on top of photographed backgrounds of the testing environment. These "Augmented Reality" datasets resulted in satisfactory performance, that did not decline in a noticeable manner when

tested in a real-world environment. However, our approach struggles when presented with objects it has not seen in training, which lead to large amounts of false positives. Furthermore, training benefited greatly from better testing conditions, with objects being placed closer to the camera resulting in much greater accuracy.

We then developed a method to extract the high-level semantic state of a scene, starting from low-level pose estimations performed by a trained network. We applied a thresholding technique, while devising methods to deal with several issues, such as the occlusion of identifying features and conflicts resulting from higher values of the distance threshold. The performance of the resulting method was excellent, though it is highly dependant on the performance of the underlying network.

Finally, we applied the complete network and semantics method to a real-world application. By "teaching" a robotic manipulator how to perform basic actions using kinesthetic demonstrations, the overall system was able to plan and complete simple assembly tasks. However, these experiments highlighted how the system struggled with smaller objects, with consistent small errors in rotation estimations that detracted from its overall reliability.

In conclusion, object detection and pose estimation approaches have remarkable performance in robotics applications, but may be insufficient for tasks that require high precision and reliability, or tasks involving small objects, or objects that are difficult to identify in other ways.

Furthermore, the black-box nature of neural networks means that the performance of our own approach and datasets may be compromised when applied to a radically different environment than the one used for dataset generation and training, thus requiring a new dataset and training for the new environment. This is the main disadvantage of our approach, that it is specific on one environment.

In the future, it could be possible to progress and improve upon our work. Possible developments include:

- Comparing the performance of a network trained on one of our generated datasets with a network trained on real-world labelled data, with the same objects and in the same environment.
- Verifying the possibility of avoiding false positives by training a model to ignore a wider variety of objects.
- When generating a dataset, verifying whether including multiple different environments in the background improves generalisiation, or whether this is unnecessary.

- If future pose estimation approaches increase performance in a significant manner, verifying whether they obtain the accuracy required for high precision robotics applications.

Bibliography

[1] D. H. Ballard. Generalising the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2), 1981.

[2] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13:41–77, 2003.

[3] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In H. Leonardis, Alešand Bischof and A. Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[4] U. Brammer and S. Seger. Freecad fasteners workbench 0.4.19, 2022. URL `https://github.com/shaise/FreeCAD_FastenersWB`.

[5] Y. Bukschat and M. Vetter. Efficientpose: An efficient, accurate and scalable end-to-end 6d multi object pose estimation approach, 2020.

[6] H. Chen, P. Wang, F. Wang, W. Tian, L. Xiong, and H. Li. Epro-pnp: Generalized end-to-end probabilistic perspective-n-points for monocular object pose estimation, 2022. URL `https://arxiv.org/abs/2203.13254`.

[7] M. Colledanchise and P. Ögren. *Behavior Trees in Robotics and AI*. CRC Press, jul 2018. doi: 10.1201/9780429489105. URL `https://doi.org/10.1201%2F9780429489105`.

[8] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. Randaugment: Practical automated data augmentation with a reduced search space, 2019. URL `https://arxiv.org/abs/1909.13719`.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[10] T.-T. Do, M. Cai, T. Pham, and I. Reid. Deep-6dpose: Recovering 6d object pose from a single rgb image, 2018. URL `https://arxiv.org/abs/1802.10367`.

[11] R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, 1 1972.

[12] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 6 1981. ISSN 0001-0782. doi: 10.1145/358669.358692. URL `https://doi.org/10.1145/358669.358692`.

[13] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014. ISSN 0031-3203.

[14] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language, 1998. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212`.

[15] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn, 2017. URL `https://arxiv.org/abs/1703.06870`.

[16] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):876–888, 2012. doi: 10.1109/TPAMI.2011.206.

[17] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Proc. Asian Conf. Computer Vision*, volume 7724, 10 2012. ISBN 978-3-642-37330-5. doi: 10.1007/978-3-642-33885-4_60.

[18] P. V. C. Hough. Method and means for recognizing complex patterns, U. S. Patent 3069654, Dec. 18, 1962.

[19] Y. Hu, P. Fua, W. Wang, and M. Salzmann. Single-stage 6d object pose estimation, 2019. URL `https://arxiv.org/abs/1911.08324`.

[20] A. Kendall, M. Grimes, and R. Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization, 2015. URL `https://arxiv.org/abs/1505.07427`.

[21] G. D. Konidaris, L. P. Kaelbling, and T. Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *J. Artif. Intell. Res.*, 61:215–289, 2018.

[22] Y. Lamdan and H. Wolfson. Geometric hashing: A general and efficient model-based

recognition scheme. In *Second International Conference on Computer Vision*, pages 238–249, 1988.

[23] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox. DeepIM: Deep iterative matching for 6d pose estimation. *International Journal of Computer Vision*, 128(3):657–678, 11 2019. doi: 10.1007/s11263-019-01250-9. URL https://doi.org/10.1007%2Fs11263-019-01250-9.

[24] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll'a r, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL http://arxiv.org/abs/1405.0312.

[25] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection, 2016. URL https://arxiv.org/abs/1612.03144.

[26] E. Muñoz, Y. Konishi, C. Beltran, V. Murino, and A. D. Bue. Fast 6d pose from a single rgb image using cascaded forests templates. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4062–4069, 2016. doi: 10.1109/IROS.2016.7759598.

[27] N. J. Nilsson. Shakey the robot. 1984.

[28] OpenCV. Detection of aruco markers, accessed 24 October 2022. URL docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.

[29] S. Peng, Y. Liu, Q. Huang, H. Bao, and X. Zhou. Pvnet: Pixel-wise voting network for 6dof pose estimation, 2018. URL https://arxiv.org/abs/1812.11788.

[30] M. Rad and V. Lepetit. Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth, 2017. URL https://arxiv.org/abs/1703.10896.

[31] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015. URL https://arxiv.org/abs/1506.01497.

[32] R. S. Sutton, D. Precup, and S. P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999. URL http://webdocs.cs.ualberta.ca/~sutton/papers/SPS-aij.pdf.

[33] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2019. URL https://arxiv.org/abs/1905.11946.

[34] M. Tan, R. Pang, and Q. V. Le. Efficientdet: Scalable and efficient object detection, 2019. URL https://arxiv.org/abs/1911.09070.

[35] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017. URL https://arxiv.org/abs/1703.06907.

[36] Unity Technologies. Unity Perception package. https://github.com/Unity-Technologies/com.unity.perception, 2020.

[37] P. F. V. Lepetit, M. Moreno-Noguer. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision.*, 2(81), 2009.

[38] L. Weng. Domain randomization for sim2real transfer. *lilianweng.github.io*, 2019. URL https://lilianweng.github.io/posts/2019-05-05-domain-randomization/.

[39] Y. Wu, L. Liu, J. Bae, K.-H. Chow, A. Iyengar, C. Pu, W. Wei, L. Yu, and Q. Zhang. Demystifying learning rate policies for high accuracy training of deep neural networks, 2019. URL https://arxiv.org/abs/1908.06477.

[40] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes, 2017. URL https://arxiv.org/abs/1711.00199.

[41] Y. Xu, K.-Y. Lin, G. Zhang, X. Wang, and H. Li. Rnnpose: Recurrent 6-dof object pose refinement with robust correspondence field estimation and pose optimization, 2022. URL https://arxiv.org/abs/2203.12870.

[42] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning, 2016. URL https://arxiv.org/abs/1611.01578.

List of Figures

List of Tables