# Automation for camera-only 6D object detection

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Pavel Rojtberg

Fachbereich Informatik

Technische Universität Darmstadt

Dissertation zur Erlangung des Grades

*Doktor-Ingenieur*

2021

Referenten:
Prof. Dr. Arjan Kuijper
Prof. Dr. techn. Dieter W. Fellner
Prof. Dr. Didier Stricker

Darmstädter Dissertation
D 17

# Erklärung zur Dissertation

Hiermit versichere ich die vorliegende Dissertation selbständig nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 24. Februar 2021 Pavel Rojtberg

# Abstract

Today a widespread deployment of Augmented Reality (AR) systems is only possible by means of computer vision frameworks like ARKit and ARCore, which abstract from specific devices, yet restrict the set of devices to the respective vendor. This thesis therefore investigates how to allow deploying AR systems to any device with an attached camera.

One crucial part of an AR system is the detection of arbitrary objects in the camera frame and naturally accompanying the estimation of their 6D-pose. This increases the degree of scene understanding that AR applications require for placing augmentations in the real world. Currently, this is limited by a coarse segmentation of the scene into planes as provided by the aforementioned frameworks. Being able to reliably detect individual objects, allows attaching specific augmentations as required by e.g. AR maintenance applications. For this, we employ convolutional neural networks (CNNs) to estimate the 6D-pose of all visible objects from a single RGB image. Here, the addressed challenge is the automated training of the respective CNN models, given only the CAD geometry of the target object. First, we look at reconstructing the missing surface data in real-time before we turn to the more general problem of bridging the domain gap between the non-photorealistic representation and the real world appearance. To this end, we build upon generative adversarial network (GAN) models to formulate the domain gap as an unsupervised learning problem. Our evaluation shows an improvement in model performance, while providing a simplified handling compared to alternative solutions.

Furthermore, the calibration data of the used camera must be known for precise pose estimation. This data, again, is only available for the restricted set of devices, that the proprietary frameworks support. To lift this restriction, we propose a web-based camera calibration service that not only aggregates calibration data, but also guides users in the calibration of new cameras. Here, we first present a novel calibration-pose selection framework that reduces the number of required calibration images by 30% compared to existing solutions, while ensuring a repeatable and reliable calibration outcome. Then, we present an evaluation of different user-guidance strategies, which allows choosing a setting suitable for most users. This enables even novice users to perform a precise camera calibration in about 2 minutes. Finally, we propose an efficient client-server architecture to deploy the

aforementioned guidance on the web, making it available to the widest possible range of devices. This service is not restricted to AR systems, but allows the general deployment of computer vision algorithms on the web that rely on camera calibration data, which was previously not possible.

These elements combined, allow a semi-automatic deployment of AR systems with any camera to detect any object.

# Zusammenfassung

Heutzutage ist eine allgemeine Bereitstellung von Augmented Reality (AR) Systemen nur mithilfe Computer Vision Frameworks wie ARKit und ARCore möglich, welche von spezifischen Endgeräten abstrahieren, allerdings gleichzeitig die Auswahl auf den jeweiligen Hersteller einschränken. In dieser Arbeit wird daher untersucht, wie die Bereitstellung von AR-Systemen auf jedem Gerät mit angeschlossener Kamera ermöglicht werden kann.

Ein entscheidender Teil eines AR-Systems ist die Detektion von beliebigen Objekten im Kamerabild und damit einhergehend die Schätzung ihrer 6D-Pose. Dies ist notwendig, um das Verständnis der Szene zu verbessern, welches AR-Anwendungen erfordern, um Augmentierungen in der realen Welt zu platzieren. Derzeit ist dies durch eine grobe Segmentierung der Szene in Ebenen begrenzt, welche durch die oben genannten Frameworks bereitgestellt wird. Einzelne Objekte zuverlässig erkennen zu können, ermöglicht es spezifische Augmentierungen anzubringen, was z.B. bei AR-Wartungsanwendungen notwendig ist. Hierzu verwenden wir Convolutional Neural Networks (CNNs), um die 6D-Pose aller sichtbaren Objekte aus einem einzigen RGB-Bild abzuleiten. Hierbei behandeln wir das Problem des automatisierten Trainings der jeweiligen CNN-Modelle, nur ausgehend von der CAD-Geometrie des Zielobjekts. Zunächst betrachten wir die Rekonstruktion der fehlenden Oberflächendaten in Echtzeit, bevor wir uns dem allgemeineren Problem der Überbrückung der „Domänen-Diskrepanz“ zwischen der nicht fotorealistischen Darstellung und dem Erscheinungsbild in der realen Welt zuwenden. Zu diesem Zweck bauen wir auf generativen CNN-Modellen (Generative Adversarial Network) auf, um die „Domänen-Diskrepanz“ als unbeaufsichtigtes Lernproblem zu formulieren. Unsere Auswertung zeigt eine Verbesserung der Modellleistung bei vereinfachter Handhabung gegenüber vergleichbaren Lösungen.

Weiterhin müssen die Kalibrierungsdaten der verwendeten Kamera bekannt sein, um eine genaue Posenschätzung zu erzielen. Diese Daten sind aber wiederum nur für die firmeneigenen Geräten der jeweiligen Frameworks verfügbar. Um diese Einschränkung aufzuheben, schlagen wir einen webbasierten Kamerakalibrierungsdienst vor, welcher nicht nur Kalibrierungsdaten aggregiert, sondern auch Benutzer bei der

Kalibrierung neuer Kameras unterstützt. Hierfür stellen wir zunächst ein neuartiges Framework für die Auswahl von Kalibrierungsposen vor, welches die Anzahl der erforderlichen Kalibrierungsbilder im Vergleich zu vorhandenen Lösungen um 30% reduziert und gleichzeitig ein wiederholbares und zuverlässiges Kalibrierungsergebnis gewährleistet. Anschließend präsentieren wir eine Auswertung verschiedener Benutzerführungsstrategien, anhand derer eine für die meisten Benutzer geeignete Einstellung ausgewählt werden kann. Auf diese Weise können auch unerfahrene Benutzer in ca. 2 Minuten eine präzise Kamerakalibrierung durchführen. Schließlich schlagen wir eine effiziente Client-Server-Architektur vor, um die oben genannten Benutzerführung im Web bereitzustellen und sie einer möglichst breiten Palette von Geräten zur Verfügung zu stellen. Dieser Dienst ist nicht auf AR-Systeme beschränkt, sondern ermöglicht die allgemeine Bereitstellung von Computer-Vision-Algorithmen im Web, welche Kamerakalibrierungsdaten benötigen, was bisher nicht möglich war.

Diese Elemente zusammen ermöglichen eine halbautomatische Bereitstellung von AR-Systemen welche auf beliebigen Kameras, beliebige Objekte erkennen können.

# Contents

# 1 Introduction

Use the webcam attached to your PC to get an augmented-reality repair guide for your cellphone; use your cellphone camera to get instructions on how to change the oil in your car; enable a robot to manipulate and inspect objects at a production line by attaching a webcam to it — do all of this without a tedious and complicated setup phase only using the virtual 3D models of the respective objects: these are some scenarios that can benefit from the results in this work.

The goal of this dissertation is to detect known objects and their 3D position relative to the viewer only using a single image and the 3D model of the objects as provided by computer aided design (CAD) tools. Here, one should be able to focus on the task at hand, while the camera capturing the image is interchangeable. The required piece of information specific to the actual camera should be retrieved automatically as needed.

The outlined environment restricts the means that we can use for this to the camera obsucra as shown in Figure 1.1 — a drawing aid used since at least 500 BC. That is, we want to achieve the goal mentioned above by merely looking at the back-facing wall of a dark chamber. There are two main challenges to derive an object distance and orientation from that image.

- First, we must be able to separate the object from the surroundings in the image and recall the true object size. Knowing the true size is crucial as a toy car can appear to be of the size of a real car in the image, given it is much closer to the camera. This problem is generally referred to as *object detection*.

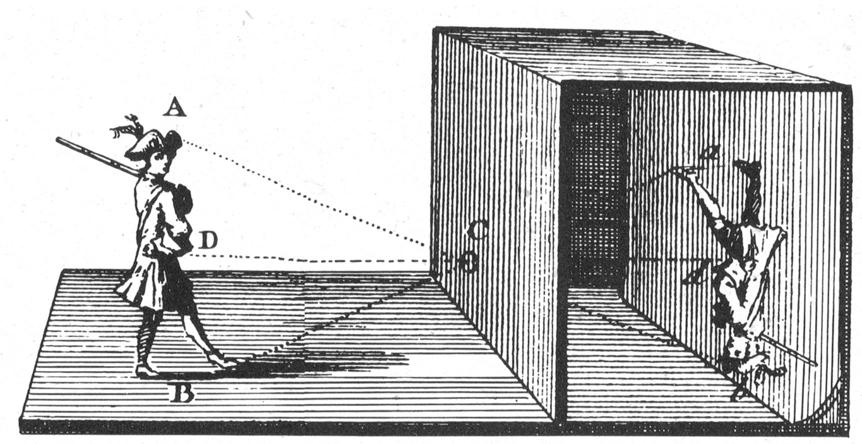


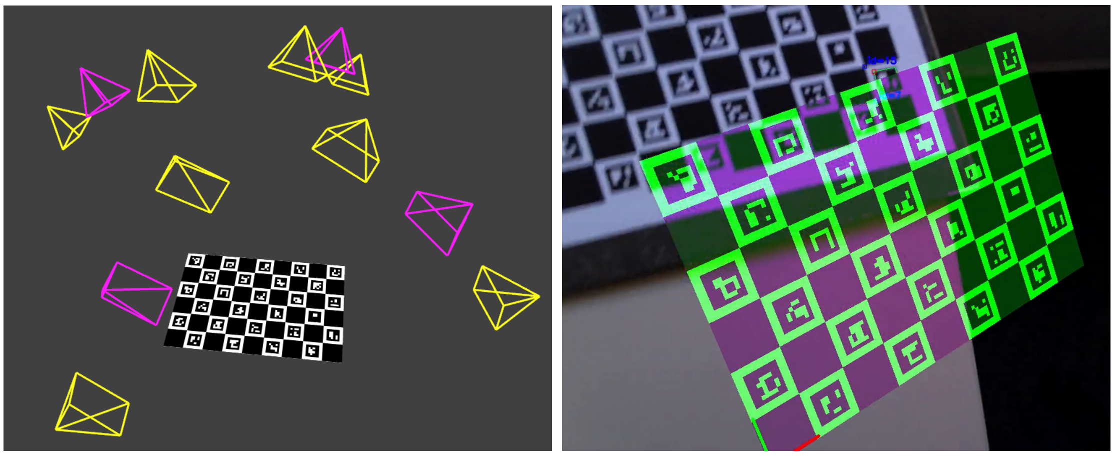
**Figure 1.1:** The geometry of a camera obscura[1]. Our goal is to recognize the soldier and estimate his distance, by only considering the image on the wall in the dark chamber.

- Second, we must know the "depth" of the dark chamber. The image size grows proportionally with the depth and the imaged size of the object together with its true size is our only means to derive the distance. Measuring, the properties of the dark chamber is generally referred to as *camera calibration*.

Object detection is a rather generic problem and there are several possible approaches. In this thesis we will focus on object detection from RGB images only. Here, we will mainly rely on Convolutional Neural Network (CNN) based models to perform this task. Only recent advances in this area make it possible to get reliable results, without the need of additional sensors. CNNs are a family of machine learning models, that are particularly well suited to analyze image data. They are characterized by a hierarchy of layers that extract and aggregate information of increasing complexity. The convolutional connectivity pattern is biologically inspired by the animal visual cortex. Furthermore, it is possible to perform the required computations in parallel on the graphics processor (GPU) leading to a very efficient implementation. CNN-based models are responsible for breakthroughs in the areas of object classification and detection. The performance of state-of-the art networks is currently limited by the error-rate of the human annotator of the training data. We are interested in methods to avoid the tedious and expensive labeling step by a human annotator. In fact, we want to get around requiring real training data at all.

To estimate an absolute 3D position and 3D orientation (6D pose) from a CNN-based detection, camera specific calibration data is required. Most notably, such calibration data describes the "zoom level" (focal length) and optical aberrations

[1]from `https://en.wikipedia.org/wiki/Camera_obscura`


**(a) Guided camera calibration:** Compute the optimal set of calibration views relative to a suitable pattern *(left)* and guide the user towards them, using an overlay *(right)*

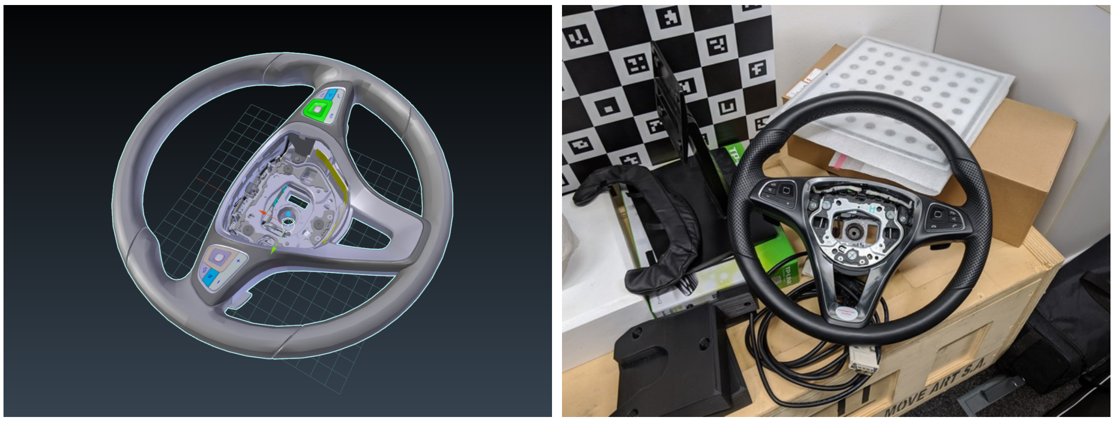

**(b) CNN training with CAD geometry:** Given the CAD geometry of an object *(left)* detect the object and its pose in real images *(right)*

**Figure 1.2:** The main tasks addressed in this work

of the used lens (distortion coefficients). Without knowing the camera calibration data, a single image is not sufficient to distinguish whether we are looking at a toy car or a real one — even if the distance to the camera is given. However, without manual calibration, such calibration data is currently only available for devices known by the ARKit and ARCore computer vision frameworks. Effectively, this limits the device range to mobile devices produced by the vendor of the respective framework. We are however interested in using any camera and more specifically, in being able to retrieve the calibration data on-the-fly.

More precisely, we are addressing the following tasks in this work, which are also illustrated in Figure 1.2:

**Reliable acquisition of calibration data** Currently, calibration data is only readily available for the very limited set of devices supported by the ARKit and

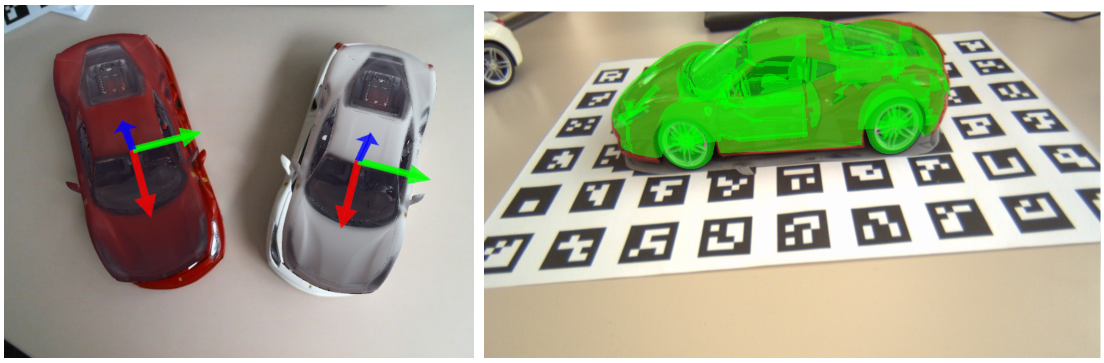
ARCore frameworks. For other devices, one has to resort to calibrating the camera manually. For this, typically a checkerboard is photographed from several views. However, special care is required to select a correct set of views to obtain a reliable calibration. Therefore, the user should be explicitly guided towards suitable views to ensure a correct calibration set. Here, the task is to measure the quality of the captured calibration data and to select additional views as appropriate.

However, ideally the system should be able to transparently retrieve the correct calibration from an open database of known devices — resembling the behavior of the aforementioned computer vision frameworks. Here, the task is to design a scalable service that covers a wide range of applications and different devices. Furthermore, the system should be extendable and allow capturing new data on-the-fly.

**CNN training from CAD geometry** Given the dependency of current CNN architectures on a large training set, the possibility of using synthetic data is alluring as it would allow generating a virtually infinite training set. Especially, assembling a training set for a specific domain is an expensive, error prone and time-consuming process that can easily take hundreds of hours [40]. Here, not only the initial capturing and labeling should be considered, but also an additional correction step of the manual annotations to ensure a high label quality. The problem is even amplified in the case of 6D object poses where the 3D data cannot be easily guessed by a human annotator and needs to be provided by custom acquisition setups e.g. by markers [38] or a turntable setup [15]. Therefore, it is desirable to use synthetic data generation to obtain annotated training samples. This is however a non-trivial task as current CNN architectures optimize exactly for the data they received during the training phase. It was shown that cross-validation is not sufficient to correctly assess CNN models[95]; their performance degrades significantly, when evaluated on a different dataset than used for training. This problem is emphasized when using synthetic, CAD-based data, which exhibits a *domain gap* to real data, a much larger difference than two real-world datasets exhibit to each other.

## 1.1 Motivation

Most products are nowadays created based on CAD models, which either serve as reference for assembly or are directly fed into a 3D printing process. Automatically aligning these models to 2D images allows transferring the information from the CAD tools, such as the geometric data, the model category or handling and assembly instructions, directly onto the images. This enables a wide range of applications


(a) Augmented Reality systems that can separate and highlight individual objects in the image

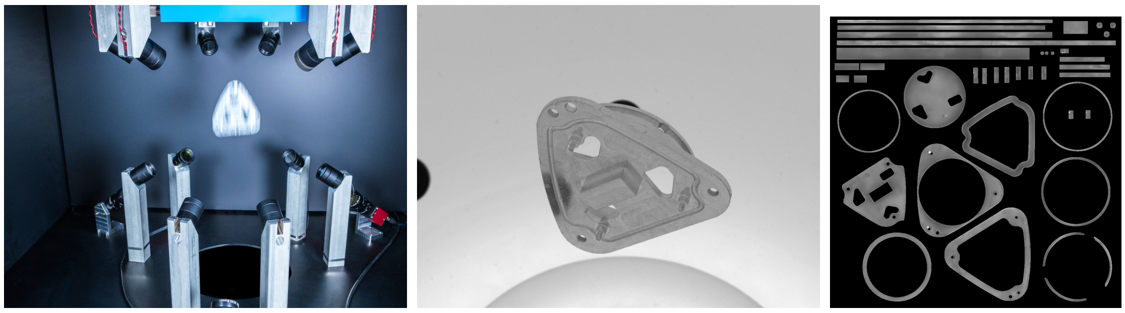

(b) Automated quality control in the production line: aligning the 3D model in multiple views allows inspecting the object surface

**Figure 1.3:** Possible applications of the methods developed in this work

in the production line and beyond by sourcing from the existing CAD product databases. Figure 1.3 shows some examples of leveraging the 3D CAD data in images, which furthermore include:

**Augmented Reality** Today Augmented Reality (AR) systems begin to be ubiquitously available through computer vision frameworks like ARKit and ARCore, which provide a precise view-pose. However, AR applications also require a certain degree of scene understanding to place the augmentations inside the real world. To this end the aforementioned frameworks provide a coarse segmentation of the scene into planes and an estimation of their size. While this allows for a certain class of applications, like placing virtual furniture, it is not possible to attach information to specific real-world objects. This is, however, a common use-case in the industrial context, where assembly and maintenance instructions aim at specific objects or object-parts.

**Product handling** During production different parts are typically grouped by their material and then produced in a batch. This batch has then to be sorted into the different parts again, which either requires the resulting parts to appear at the exact same location or a human worker to perform the task. Think of a 3D-printing process where different parts are created at once. The main objective here is to optimally exploit the printing volume, therefore the printing algorithm arranges the parts in arbitrary locations to achieve a tight packing. If the source CAD geometry could be aligned to the printed objects, the sorting could be performed automatically by a robot. This use-case poses an additional challenge as different printing materials lead to different appearances of the same CAD model. One can think of similar cases during stamping and cutting from metal.

**Quality control** After production the parts need to pass a quality control. This can happen before they are shipped to the customer or during a larger assembly. This is a crucial step as integrating a faulty part can cause the destruction of the final product which can increase the loss by an order of magnitude. Think of mounting a polluted plate into an electric motor that is afterwards sealed. Leveraging the CAD data here, one could automatically inspect the plates and remove the faulty ones from the pipeline.

## 1.2 Challenges

Solving tasks addressed in this thesis imposes the following underlying challenges:

**Reproducible, guided camera calibration** The quality of a camera calibration depends on the used set of calibration poses. Certain pose configurations can lead to unreliable results and it is possible to capture redundant views. That is, distinct viewpoints which do not add any information to the calibration set. Therefore, the challenge is to find a *set of poses* that neither includes unreliable configurations nor redundant poses. Additionally, it should be possible for an inexperienced user to perform the calibration. This means, the process should be tolerant to poses that are not matched exactly and dynamically adapt the pose sequence, given the data captured so far. Finally, the calibration results should be reproducible; i.e. repeated calibration of the same device should result in comparable output — even when repeated by a different user.

The calibration results should be aggregated by an online *calibration service*, such that no calibration needs to be performed for devices where a reliable calibration is already available. This service should scale to a wide range of scenarios — including web-based applications. These are particularly challenging, as they can be executed on virtually any device — ranging from a smartphone to an embedded Linux board with a browser and a camera. To manage the volatile set of capturing devices, the service must be dynamically extensible and fall back to *guided calibration* when encountering an unknown device. Consequently, the guidance must be executable in a web-browser environment and therefore the implementation must settle for web technologies. This imposes limitations on computation and requires and an efficient client/server separation.

**Domain Gap between synthetic and real images** There is a considerable *domain gap* between synthetic renderings and real world images which prohibits the generalization of networks trained on synthetic data to the real world. Typically, rendered images produce clear edges with only approximate shading, while real images exhibit various sources of noise, like optical aberrations, sensor noise or compression artifacts. One approach to overcome the domain gap is to generate photorealistic quality images, by a more sophisticated shading simulation and the incorporation of the aforementioned imaging artifacts. However, increasing the photo-realism requires either an artist to carefully model the specific environments in detail or a specialized acquisition setup to capture the reflectance properties. This in turn increases the cost of generating the data thus negating the primary selling point of using synthetic images in the first place.

Therefore, *surface capturing* should be straightforward, with no additional setup. Ideally, the CNN training itself should be adapted, to produce a model that is general enough to handle real images just as well as the synthetic images used for training. Here, the challenge is not to degrade the general model performance by removing essential cues. For instance, a model trained to detect cars should be able to exploit the fact that they are usually located on a ground plane. Using random backgrounds during training removes this cue and the model will be forced to handle flying cars as well.

## 1.3 Contributions and Outline

This thesis focuses on reducing the deployment overhead for 6D object detection in RGB images by increasing the automation of the associated tasks. Particularly, we consider the acquisition of camera calibration data and the automated generation of labeled training data for 6D pose estimation.

Here, we first turn to the semi-automatic acquisition and automatic distribution of camera calibration data, which is a prerequisite for 3D vision. Having the calibration data available, we then continue with training a model for the task of 6D pose estimation from RGB images only, while only relying on non photo-realistic CAD geometry. For this, we explicitly address the domain gap between real and synthetic data.

In Chapter 2, we review the basic concepts and formalize the addressed tasks, including the definition of the notation used for the remains of this work. Here, we introduce the pinhole camera model and the closely related problem of pose estimation, including suitable metrics for evaluating the estimates. We also review the general machine learning framework and give an overview over related work on architectures for object detection and object pose estimation. At this, we also discuss the domain gap as a general form of the dataset bias and review existing approaches for approaching this problem, including domain randomization and domain adaptation.

Chapter 3 presents an algorithm for efficiently selecting camera *calibration poses*, based on already captured calibration frames. This allows *interactively guiding* a user through the calibration process, while ensuring repeatability and high quality of the results. Here, we also evaluate different calibration settings and visualizations for user-guidance by performing a quantitative user survey. We then extend the system by aggregation capabilities and describe the deployment of the complete system as a *web-service*, which makes camera calibration data as well as the pose selection algorithm ubiquitously available. The results of this chapter are based on the following publications

- Rojtberg, Pavel, and Kuijper, Arjan. "Efficient pose selection for interactive camera calibration." *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR).* IEEE, 2018. — [83]

- Rojtberg, Pavel "User Guidance for Interactive Camera Calibration." *2019 Proceedings of the 21st International Conference on Human-Computer Interaction.* Springer, 2019. — [80]

- Rojtberg, Pavel, and Gorschlüter, Felix "calibDB: enabling web-based computer vision through on-the-fly camera calibration." *2019 Proceedings of the 24th International Conference on 3D Web Technology.* ACM, 2019. **Best Short Paper Award.** — [82]

In chapter 4, we present approaches to bridge the domain gap between training data generated from abstract CAD geometry and real-world images, focusing on the challenging task of 6D object pose estimation. We first approach this task by recovering the true *object appearance*, which is absent in CAD data, thus increasing the realism of the training data. Here, a real-time method is presented, that operates without the need of a sophisticated capturing setup. The captured data is then used to extend a classical object detection algorithm on-the-fly to allow object instance identification.

Then, we turn to the training step of the CNN to enforce a more general model, which is able to cover the domain gap. For this, we formulate the domain gap as a style-transfer problem. This turns the *domain gap* itself into a learn-able task and allows employing off-the shelf generative adversarial networks (GANs) to solve it. Here, we consider both supervised and unsupervised training setups and show that our formulation results in a considerable performance improvement, while requiring only little effort to set up when compared to other methods. This ultimately allows training a pose estimation network from synthetic CAD data only. The results of this chapter are based on the following publications

- Rojtberg, Pavel, and Arjan Kuijper. "Real-time texturing for 6D object instance detection from RGB Images" *2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR).* IEEE, 2019. — [84]
- Rojtberg, Pavel, and Thomas Pöllabauer "Style-transfer GANs for bridging the domain gap in synthetic pose estimator training" *Proceedings of the 25th International Conference on Pattern Recognition (AIVR2020)* IEEE, 2020. — [85]

Finally, chapter 5 concludes this thesis giving a summary of our results and discussing the limitations of the approaches. We also describe directions that seem worthwhile to follow in the future.

# 2 Background

In this chapter we present the basic concepts and the notation that will be used throughout this thesis.

We start with the pinhole camera model, which describes the fundamental relation between the 3D world points and the 2D image points that we observe and use to identify the object and its pose. We go on to the closely related task of pose estimation. In this context we discuss different evaluation metrics and the implications they have on judging the estimated pose.

Next, we review the machine learning framework that will be used for the task of object pose estimation. Specifically, we focus on deep convolution architectures. Here, we also consider the training phase where we discuss the problem of overfitting that leads to the dataset bias and ultimately to the domain gap that we are confronted with when using CAD geometry to generate the training data. In this context, we discuss related work on conditioning the training for generalization via domain randomization and domain adaptation. We also present data augmentation as a simple domain randomization technique. We then turn to specific network architectures used in the later sections of this work. Specifically, we introduce generative adversarial networks for style transfer and single-stage detector architectures. At this, we start by reviewing general learning-based detection architectures which lead to more specific 6D pose estimation models. In this context, we also discuss classical approaches for object pose estimation and their limitations to motivate our use of deep convolutional models.

## 2.1 Camera model

We will use the pinhole camera model that, given the camera orientation $\mathbf{R}$, position $\mathbf{t}$ and the parameter vector $\mathbf{C}$, maps a 3D world point $\mathbf{P} = [X, Y, Z]$ to a 2D image point $\mathbf{p} = [x, y]$:

$$\pi(\mathbf{P}; \mathbf{R}, \mathbf{t}, \mathbf{C}) = \mathbf{K}\,\Delta(\frac{1}{Z_c}[\mathbf{R}\ \mathbf{t}]\,\mathbf{P}). \tag{2.1}$$

where $[\mathbf{R}\ \mathbf{t}]$ is a $3 \times 4$ affine transformation, $Z_c$ denotes the depth of $\mathbf{P}$ after affine transformation and $\mathbf{K}$ is the camera calibration matrix:

$$\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

.

$\mathbf{K}$ encodes the focal lengths $[f_x, f_y]$ and the principal point $[c_x, c_y]$. Zhang [112] also includes a skew parameter $\gamma$ — however, for CCD-sensor based cameras it is safe to assume $\gamma$ to be zero [91, 35].

The largest limitation of the pinhole model is that optical effects introduced by the camera lens are not considered. In order to model them, the projected points are usually warped using non-linear distortion equations. The most commonly modelled effects [91] are the radial (2.3a) and tangential (2.3b) lens distortions. Following Heikkila and Silvén [36] we formulate them as

$$\Delta(\mathbf{p}) = \mathbf{p}\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \tag{2.3a}$$

$$+ \begin{pmatrix} 2p_1 xy + p_2\left(r^2 + 2x^2\right) \\ p_1\left(r^2 + 2y^2\right) + 2p_2 xy \end{pmatrix}, \tag{2.3b}$$

where $r = \sqrt{x^2 + y^2}$.

Therefore $\mathbf{C} = [f_x, f_y, c_x, c_y, k_1, k_2, k_3, p_1, p_2]$.

Additionally, Web-based computer vision should not be restricted to pinhole camera imagery, therefore we have to allow for additional camera models. Here, we consider the DNG image format [1] where eq. (2.3a) is specified as *WarpRectilinear* for processing images from interchangeable-lens cameras.

Furthermore, the DNG format includes a separate distortion model for fish-eye lenses [47], *WarpFisheye*:

$$\Delta_F(\mathbf{p}) = \mathbf{p}\frac{1}{r}\left(\theta + k_1\theta^3 + k_2\theta^5 + k_3\theta^7\right) \tag{2.4}$$

where $\theta = \text{atan}(r)$. This model is required as fisheye lenses can expose a field of view $\geq 180°$ which cannot be represented using a rectilinear projection.

The OpenCV library supports both $\Delta$ and $\Delta_F$ as well as more sophisticated models for e.g. spherical 360° cameras [32] as employed by the street-view cars or spherical video.

### 2.1.1 Estimation and error analysis

Given $M$ images each containing $N$ point correspondences, the underlying calibration method [112] minimizes the geometric error

$$\epsilon_{res} = \sum_{i}^{N} \sum_{j}^{M} \parallel \mathbf{p}_{ij} - \pi\left(\mathbf{P}_i; \mathbf{R}_j, \mathbf{t}_j, \mathbf{C}\right) \parallel^2, \tag{2.5}$$

where $\mathbf{p}_{ij}$ is an observed (noisy) 2D point in image $j$ and $\mathbf{P}_i$ is the corresponding 3D object point.

Eq. (2.5) is also referred to as the reprojection error and often used to assess the quality of a calibration. Yet, it only measures the residual error and is subject to over-fitting. Particularly $\epsilon_{res} = 0$ if exactly $N = 10.5$ point correspondences are used [35, §7.1].

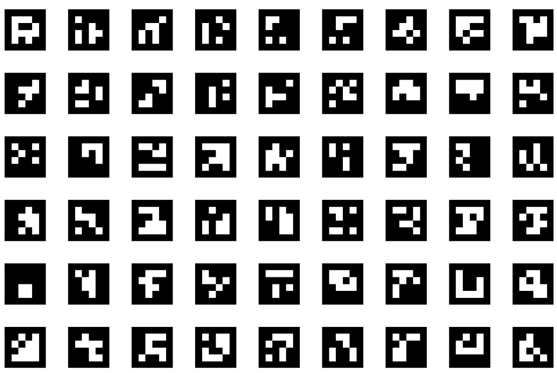
The actual objective for calibration however, is the estimation error $\epsilon_{est}$, i.e. the distance between the solution and the (unknown) ground truth. Richardson et al. [79] propose the Max ERE as an alternative metric that correlates with the estimation error and also has a similar value range (pixels). However, it requires sampling and re-projecting the current solution. Yet for user guidance and monitoring of convergence only the relative error of the parameters is needed. Therefore, we directly use the variance $\boldsymbol{\sigma}_C^2$ of the estimated parameters. More precisely, we normalize the value ranges using the index of dispersion (IOD) $\sigma_i^2/C_i$ to ensure comparability among different parameters.

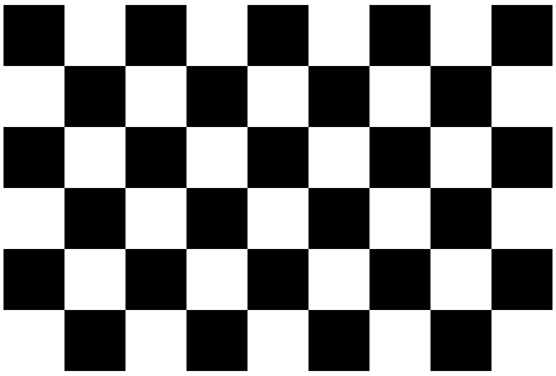
Given the covariance of the image points $\mathbf{\Sigma}_p$ the backward transport of covariance [35, §5.2.3] is used to obtain

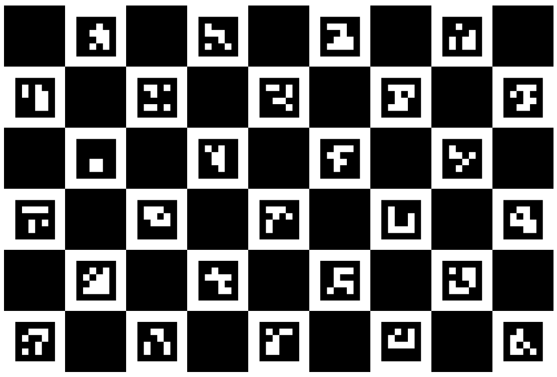
$$\begin{aligned} \mathbf{\Sigma}_v &= \left(\mathbf{J}^T \mathbf{\Sigma}_p^{-1} \mathbf{J}\right)^+ \\ \mathbf{J} &= \delta\pi/\delta\mathbf{v} \end{aligned} \tag{2.6}$$

where $\mathbf{J}$ is the Jacobian matrix, $\mathbf{v} = [\mathbf{C}, \mathbf{R}_1, \mathbf{t}_1, \ldots, \mathbf{R}_M, \mathbf{t}_M]$ is the vector of unknowns and $(\cdot)^+$ denotes the pseudo inverse. For simplicity and because of the lack of prior knowledge, we assume a standard deviation of 1px in each coordinate direction for the image points thus simplifying $\mathbf{\Sigma}_p = \mathbf{I}$.

The diagonal entries of $\mathbf{\Sigma}_v$ contain the variance of the estimated $\mathbf{C}$. Note, that $\mathbf{J}$ is readily available through the Levenberg-Marquardt step of [112].


**(a)** Fiducial marker grid "Tsai Grid"


**(b)** Chessboard


**(c)** ChArUco pattern

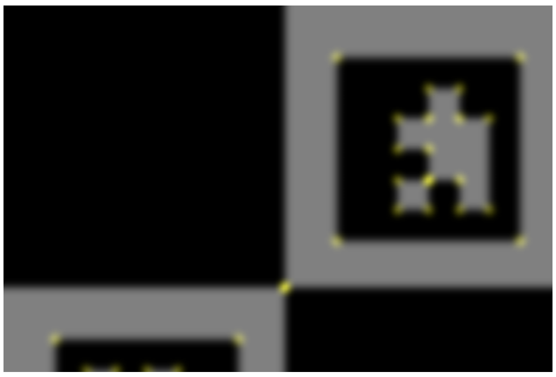

**(d)** Gradient bias

**Figure 2.1:** Planar calibration patterns

## 2.2 Calibration patterns

To perform the calibration, we need to detect points with known (local) 3D coordinates in the camera image. Basically an arbitrary, yet known, 3D geometry can be used for this — however, any imprecisions in the provided 3D coordinates result in an increased error of the calibration [91]. Therefore, it is preferable to use a planar target as it is easy to manufacture at high precision by simply printing an image and fixing it to a planar surface (e.g. glass). While one could use arbitrary images in combination with a 2D feature detector, typically squared black and white patterns are used. These have strong gradients that can be detected even under difficult lighting conditions. Additionally, the 2D points can be located with sub-pixel accuracy by searching for a maximum along the local gradient, thereby further improving the precision of the calibration.

A widely used checkerboard pattern is the "Tsai Grid" [99] that resembles Figure 2.1a, apart from the marker coding. It is used for instance in [112, 90, 35].

However, the image gradient at the corners of the used squares is biased towards the outside which impedes sub-pixel refinement. Therefore, the "Tsai Grid" was superseded by the chessboard pattern (Figure 2.1b). With the latter only points at joints of two squares are used, which do not exhibit the gradient bias. Figure 2.1d shows the top-left region of Figure 2.1c overlaid with the results of a bilateral Sobel filter. Note the maxima are outside the isolated marker but at the joint of two chessboard squares.

The main disadvantage of the chessboard pattern is that the entire board needs to be visible for the corner identification to work. Furthermore, the detection process usually involves the time-consuming task of ordering the detected rectangles to a canonical topology. This slows down board localization below 30Hz and impedes the interactivity of the method.

Approaches based on fiducial marker boards [102, 26, 79] (Figure 2.1a) overcome both of the above limitations. Marker detection is fast and one individual marker allows the direct computation of board coordinates using its unique ID. However, marker boards have the same structure as the "Tsai-Grid" and hence also suffer from corner bias.

Therefore, the ChArUco pattern [29] was recently introduced. It interleaves ArUco markers [30] with the chessboard pattern (Figure 2.1c). The markers are then used for detection and localization, while the chessboard corners provide bias-free points with sub-pixel accuracy.

Our approach works with any of the mentioned planar calibration target. However, for interactive user guidance a fast board detection is crucial. Therefore, we settle on the ChArUco pattern as implemented in OpenCV. Alternatively, one could use any of the recently developed self-identifying targets [3, 6, 26] here.

The pattern size is set to $9 \times 6$ squares resulting in up to 40 measurements at the chessboard joints per captured frame. This allows to successfully complete the initialization even if not all markers are detected as discussed in section 3.3.5.

## 2.3 Pose estimation

Assuming the pinhole camera model, we want to recover a $3 \times 4$ affine transformation $\mathbf{E} = [\mathbf{R}\ \mathbf{t}]$ between the camera and the object, given the camera calibration matrix $\mathbf{K}$ and several 2D-3D correspondences of 3D world points $\mathbf{P}_i = [X, Y, Z]$ and 2D image points $\mathbf{p}_i = [x, y]$. As in eq. (2.1), we are exploiting their relation by

$$\mathbf{p}_i = \mathbf{K} \frac{1}{Z_c} [\mathbf{R}\ \mathbf{t}]\, \mathbf{P}_i. \tag{2.7}$$

Note that compared to eq. (2.1), we just swap the known and unknowns and drop the non-linear distortion function — assuming all 2D points are undistorted before pose estimation.

The $3 \times 4$ affine transformation $[\mathbf{R}\ \mathbf{t}]$ contains only 6 degrees of freedom as the $3 \times 3$ rotation transform is part of the special orthogonal group $\mathbf{R} \in SO(3)$ and thus only has 3 degrees-of-freedom. Therefore, 3 point correspondencies are sufficient to obtain a solution. The general problem is known as "Perspective-n-point" (PnP), where $n$ refers to the number of known correspondencies. There is a large base on research for the special cases of $n = 3$ and $n = 4$ [28] as well as arbitrary values of $n$ [56, 52] providing direct and iterative solutions. Additionally, there is research on the particular configuration where all points are co-planar [19].

Generally, all cases can be handled, if a check for the degenerate configurations is performed to dispatch to an optimal implementation. For the remains of the work, we therefore do not address the degenerate configurations explicitly.

### 2.3.1 Evaluation metrics

Evaluation of 6D object pose estimates is not straightforward [41]. There are multiple popular metrics for measuring the distance between an estimated pose $\hat{\mathbf{E}}$ and a ground truth pose $\bar{\mathbf{E}}$ which we will briefly present in the following.

**Reprojection error** By generating 2D image points $\bar{\mathbf{p}}$ using the ground truth pose, one can use the re-projection error as defined by eq. (2.5) in camera calibration. This metric measures the distance in the image space which is most significant for AR applications. The disadvantage is that due to the projection, the weight of the depth error is decreased as the distance to camera increases. Therefore, the metric depends on the used camera lens. Additionally, the error must be normalized to be comparable between different image resolutions. A popular variant introduced by [10] is to accept an estimated pose if the average reprojection error is below 5px at an image resolution of $640 \times 480$.

**Intersection over Union** The IoU metric (also called Jaccard index $J$) is computed by comparing the areas $\hat{A}$, $\bar{A}$ covered by the 2D projection of an object using the estimated pose and the ground truth pose respectively, as

$$J(\hat{A}, \bar{A}) = \frac{|\hat{A} \cap \bar{A}|}{|\hat{A} \cup \bar{A}|}. \tag{2.8}$$

Measuring the distance in image space, the same advantages and disadvantages as with the reprojection error apply. The main benefit is resolution independence. A popular variant [10] is to accept an estimated pose if the IoU score is above 0.5. Some variants use the 2D bounding box of an object to approximate the true area covered by its projection.

**Average distance between model points** This metric measures the average distance between the points in 3D of the object model $\mathcal{M}$ transformed with the pose estimate $\hat{\mathbf{E}}$ and the ground truth pose $\bar{\mathbf{E}}$ respectively:

$$\frac{1}{n} \sum_{\mathbf{P} \in \mathcal{M}} \| \hat{\mathbf{E}} \mathbf{P} - \bar{\mathbf{E}} \mathbf{P} \|, \tag{2.9}$$

where $n = |\mathcal{M}|$. It was introduced by [38] and is often abbreviated as ADD. Measuring an absolute 3D distance it is favorable for robotic applications that require the full pose. However, the pivot point of the object is required to be at the center of mass for the metric to be comparable between different objects. A popular variant is to accept an estimated pose if the ADD distance is below 10% of the diameter of the 3D bounding box.

**Rotational and translational error** This metric consists of two separate measures for rotation and translation. For the translational error the 3D distance is used. For rotational error, the rotation between the estimated and the ground truth rotation $\mathbf{R}' = \hat{\mathbf{R}} \bar{\mathbf{R}}^T$ is converted into the angle-axis representation, where angle of rotation $\theta$ is used as the measure

$$\theta = \arccos\left(\frac{\mathrm{Tr}(\mathbf{R}') - 1}{2}\right). \tag{2.10}$$

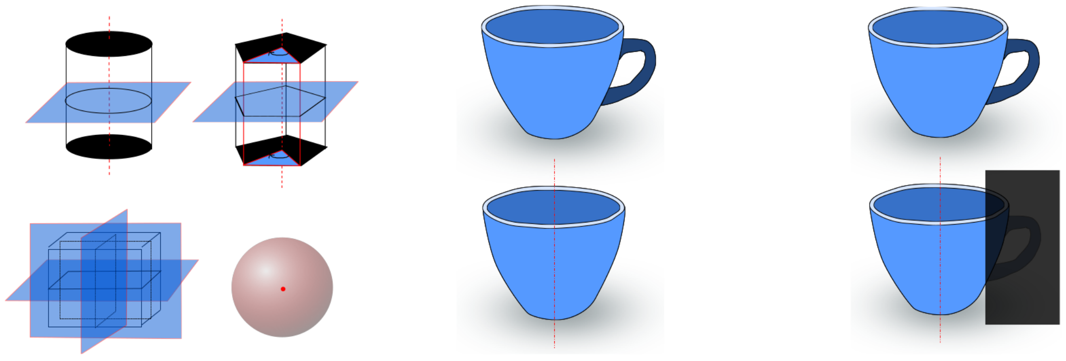
A popular variant introduced by [88] is to accept an estimated pose if the translational error is below 5cm and the rotational error is below 5°.

## 2.3.2 Indistinguishable poses

For some objects there are sets of poses that cannot be distinguished under projection. This can be due to a symmetry inherent in the object geometry. Think of a glass — rotating it around its symmetry axis results in an infinite set of poses having an identical projection. The poses can also be only indistinguishable in some views of the object. For instance, the pose of a cup can be uniquely determined as long as the handle is visible. If it is occluded by another object or not visible due to self-occlusion the poses become indistinguishable as well (see Figure 2.2).

Of the metrics presented above, only the Jaccard index is invariant to indistinguishable poses. That is, there is no penalty if $\hat{\mathbf{E}}$ and $\bar{\mathbf{E}}$ are different, yet are indistinguishable under projection. To further overcome the limitations of measuring in image space, [41] propose the Visual Surface Discrepancy (VSD) metric that computes the distance of visible surface points in 3D.

For the remains of the work, however, we assume poses to be distinguishable under projection and do not address this issue explicitly unless stated otherwise.



**(a)** Object symmetries **(b)** Self-occlusion **(c)** Occlusion

**Figure 2.2:** Causes of pose ambiguities (Image from [92])

## 2.4 Machine learning framework

In this section we will introduce the machine learning framework[1] that we will use in this work. Here, we focus on single-stage convolutional neural networks.

### 2.4.1 Machine learning

First, let us formalize a general mathematical framework for learning. We are given a set of training examples

$$\mathcal{D} = \{z_0, \dots, z_{n-1}\},$$

where each $z_i$ is a sample from an unknown distribution $P(Z)$. Additionally, we are given a loss function $L$ that takes the *decision function* $f$, a sample $z_i$ and outputs a real-valued error score

$$L(f, Z) \in \mathbb{R}^+.$$

We want to minimize the value of $L(f, Z)$ by optimizing for $f$.

**Unsupervised Learning** In the unsupervised learning setting, the function $f$ is used to characterize the distribution $P(Z)$. For instance if $f$ is gaussian, we are estimating the density of $P(Z)$. $f$ could also create a different representation of $Z$. The Principal Component Analysis would be an example for this.

**Supervised Learning** In the supervised setting, each sample is an (input, target) pair: $Z = (X, Y)$, where $X$ is the domain of $f$ and $Y$ is the co-domain of $f$.

We can now further subdivide the supervised learning setting into

[1]Based on `http://deeplearning.net/tutorial/contents.html`

**Regression** If the set $Y$ consists of continuous quantity $Y \subset \mathbb{R}^n$, the setting is called regression. A typical choice for the loss function here is the squared error

$$L(f, (X, Y)) = \parallel f(X) - Y \parallel^2 .$$

**Classification** If the set $Y$ consists of finite integers $Y \subset \mathbb{Z}$ that can be interpreted as a class index, the setting is called classification. Here, we are minimizing the negative log likelihood

$$L(f, (X, Y)) = -\log f_Y(X),$$

where $f_i(X)$ is interpreted as the likelihood that estimates the conditional probability $P(Y = i|X)$. Note that when assuming $f_i(X)$ to be Gaussian, the negative log likelihood simplifies to the squared error as in regression.

This work will focus on supervised regression and classification.

### 2.4.2 Gradient-based learning

Lets consider we want to find the parameters $\theta$ of function $f$ that minimize the loss $L$ given the training set $\mathcal{D}$. To this end we compute the average loss over $\mathcal{D}$

$$C(\theta) = \frac{1}{n} \sum_i^n L(f_\theta, z_i),$$

where $C$ is referred to as cost function or objective function.

Searching the $\theta$ that minimizes the loss $L$, can now be formalized as $arg\, min_\theta\, C(\theta)$. If we are able to compute the derivative of $C$ and solve the equation

$$\frac{dC(\theta)}{d\theta} = 0,$$

we can directly obtain the respective $\theta$. However, typically there is no closed form solution for $f$ and consequently we cannot solve the equation above.

Assuming that $C$ is locally linear and given some initial value $\theta^0$, we can resort to numerical optimization. The general idea is to iteratively update $\theta^0$ to decrease $C(\theta)$ until *convergence* i.e. until we cannot decrease $C(\theta)$ any further.

To linearize $C$ for a vector valued $\theta$ we compute its gradient, which is the row-vector Jacobian

$$\frac{\delta C(\theta)}{\delta \theta} = \left[ \frac{\delta C}{\delta \theta_0}, \ldots, \frac{\delta C}{\delta \theta_n} \right].$$

The simplest gradient-based numerical optimization method is the *gradient descent*, where we follow the objective function into the direction of its most rapid decrease at iteration $k$ to obtain $\theta^{k+1}$ from $\theta^k$ as

$$\theta^{k+1} = \theta^k - \lambda_k \frac{\delta C(\theta^k)}{\delta \theta^k},$$

where $\lambda_k$ is a scalar that controls the length of the step in gradient direction. Therefore, it is commonly referred to as *step-size.* However, in the context of neural-networks it is referred to as the *learning-rate.* If $\lambda_k$ is too large we might skip over the minimum, while if $\lambda_k$ is too small, it might take a long time until we reach convergence. There are various schedules to set $\lambda_k$, ranging from a fixed value to gradually decreasing step size with increasing iteration count.

### 2.4.3 Stochastic gradient descent

$C$ computes an average over generally independently and identically distributed samples $z_i$. Taking advantage of that, one can update $\theta$ while only using parts of $\mathcal{D}$ — in the extreme case using only one sample $z \in \mathcal{D}$. In this case the update simplifies to

$$\theta^{k+1} = \theta^k - \lambda_k \frac{\delta L(f_{\theta^k}, z)}{\delta \theta^k}. \tag{2.11}$$

This variant is called *stochastic gradient descent* (SGD). Using this formulation, the gradient direction itself is considered a random variable, whose expectation is the true gradient of the unknown distribution $P(Z)$. Notably, it allows an online learning scenario where the training set $\mathcal{D}$ is not fixed, but rather a stream of samples from the training distribution.

However, the commonly used variant of SGD is the *minibatch* stochastic gradient descent, that uses small batches of $B$ samples. This is a compromise between the ordinary (batch) gradient descent that is using the whole dataset $\mathcal{D}$ and thus results in a better estimate of the gradient and the pure SGD that uses only one sample and tends to reach convergence faster.

The main reason behind using minibatch SGD is that one can replace $B$ vector $\times$ matrix products by one matrix $\times$ matrix product which can be implemented more efficiently. The optimal choice of $B$, therefore, depends on the used hardware (memory-size, parallelism).

Nowadays, all neural-network based learning tasks are using minibatch SGD.

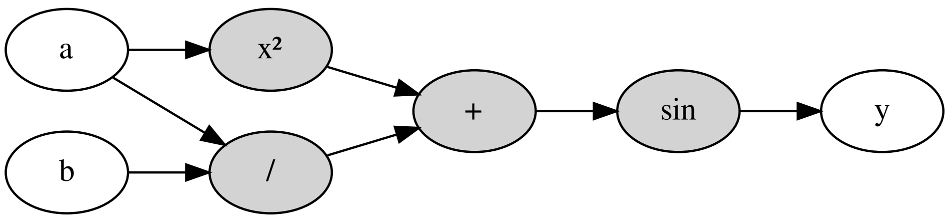


**Figure 2.3:** Flow graph for the expression $y = \sin\left(a^2 + a/b\right)$

### 2.4.4 Deep neural networks

A mathematical expression that produces an output from some inputs can be expressed as a flow graph that follows the computation. Here, each node represents a primitive operation (e.g. $+$, sin) and the resulting value.

Figure 2.3 shows the flow graph for the expression $y = \sin\left(a^2 + a/b\right)$. A key property of the flow graph is its *depth*; the longest path from any input to any output node. The depth, together with the number and type of nodes defines a family of functions. The preceding example has a depth of three. Support Vector Machines have a depth of two (one for the feature space and one for the output summing up the features). Feed-forward neural networks have a depth that corresponds to their number of layers.

Neural networks are a family of functions whose flow graph has a specific hierarchical structure. It is composed of a series of linear functions followed by non-linearities. This structure was first introduced with the perceptron algorithm [86] in 1958, which we briefly introduce in the following.

The basic single-layer perceptron algorithm can be formalized as

$$f(x) = s\left(\mathbf{W}\,x + \mathbf{b}\right), \tag{2.12}$$

where $\mathbf{W}$ is the weight matrix connecting the inputs to the output, $\mathbf{b}$ is a bias vector and $s$ is the *activation function*.

This simple model is using the input features $x$ as is and thus only has the capacity to classify linearly separable data. To make it more powerful we extend it to the multi-layer perceptron (MLP) or neural-network. To this end, we chain the perceptron (2.12) with itself as

$$f(x) = s_2\left(\mathbf{W}_2\ \left(s_1(\mathbf{W}_1\,x + \mathbf{b}_1)\right) + \mathbf{b}_2\right),$$

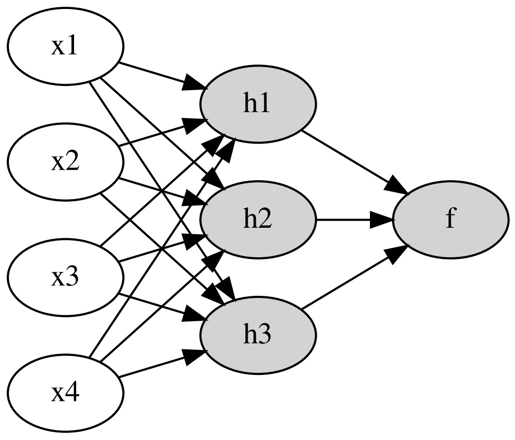


**Figure 2.4:** Example of a two-layer perceptron with 4 inputs, one hidden layer and a real valued output.

where the inner invocation $h(x) = s_1 (\mathbf{W}_1 x + \mathbf{b}_1)$ forms the *hidden* layer. See Figure 2.4 for a graphical representation.

The introduction of the hidden layer is sufficient to make the MLP an universal approximator [20, 42]. This means it can approximate any continuous function over a compact subset of $\mathbb{R}^n$, as long as a non-polynomial activation function $s$ is used. Typical choices for $s$ are $\tanh(\cdot)$ or $\mathrm{sigmoid}(\cdot)$.

We can add more hidden layers by iteratively applying eq. (2.12) to increase the depth of the neural network. Even though a single hidden-layer network is already a universal approximator, later research [5, 61] has shown that certain families of functions can be represented efficiently with $O(n)$ hidden nodes with depth $d$, where $n$ is the number of inputs. However, when limiting the depth to $d - 1$, the number of required nodes grows exponentially as $O(2^n)$. The required amount of memory and processing power to evaluate the network grows with the number of internal nodes. Therefore, it is beneficial to build deep neural network architectures.

As a convention, we will call the block of all but the last layer the *feature extractor*. In the example used above the inner invocation $h(x)$ is the feature extractor which transforms the input features $x$ to the hidden feature space. The outer invocation $f(h(x))$, that operates on that feature space is just a *linear model* that generates the final output.

Typically, neural networks are non-convex with no closed form solution. Therefore, we have to resort to gradient-based learning as described in section 2.4.2 to find optimal weights $\mathbf{W}_i$ and biases $\mathbf{b}_i$.

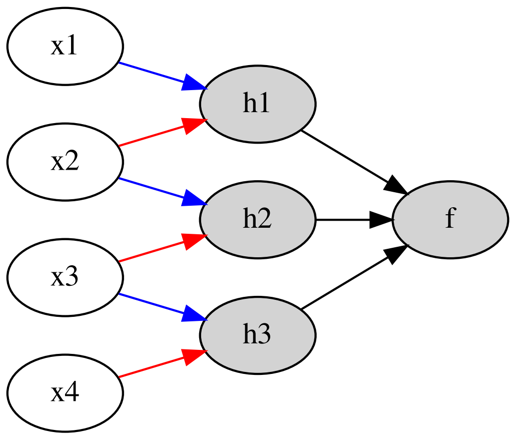


**Figure 2.5:** Example of a CNN with a receptive field of 2, operating on 1D input. Weights of the same color are shared.

One can see the neural network as a factorization of some target function. The existence of a deep and compact representation indicates some structure of this function. If there was no such structure, it would not be possible to generalize from training data.

### 2.4.5 Convolutional neural networks

In the context of image processing, typically MLP variants with limited connectivity are used which are referred to as convolutional neural networks (CNN). These are biologically inspired models that resemble the visual cortex. The visual cortex is a complex arrangement of cells, where each cell is only sensitive to a small sub-region of the visual field. This region is called the *receptive field* [44]. The cells act as a filter over the input space and exploit the spatially local correlation present in images. Furthermore, complex cells in the visual cortex have been found to be locally invariant regarding the exact position of the pattern.

In conventional neural networks all nodes in one layer are densely connected with all nodes of the following layer. For instance all input nodes $x$ in Figure 2.4 are connected to all hidden nodes $h$. This connectivity results in quadratic growth $O(n^2)$ of the weight matrix $\mathbf{W}$, which can already become a bottleneck with RGB images of the size $256 \times 256$ (38 billion connections, given a hidden layer of the same size). The dense connectivity results in giving all image regions the same influence on the output and thus not modeling any spatially local correlation between input and output.

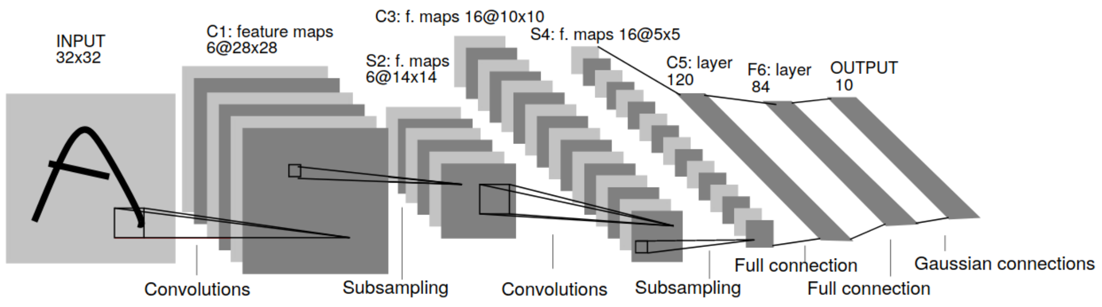


**Figure 2.6:** Architecture of the LeNet-5 CNN for digit classification (Figure from [55])

Transferring the receptive field concept to neural networks is done by the introduction of limited connectivity. This means that nodes from one layer are only connected to neighboring nodes of the previous layer. This neighborhood is then the receptive field of the node in analogy to the biological cell and acts as a signal-processing filter. Additionally, it is enforced that the weights that are connecting a node to its receptive field are the same for all nodes in a layer. This scheme is called *weight-sharing* and ensures that the learned filters are spatially invariant (see Figure 2.5).

**Convolutional layer** Mathematically this concept is modeled by convolutions, where the receptive field is equivalent to the kernel size. Typical kernel sizes are in the range $[0; 11]$. The hierarchical structure then exploits the spatially local correlation of pixels in the lower layers, while still allowing the aggregation of the extracted information at the higher levels. The output of the convolution of a preceding layer with a learned kernel produces a *feature-map* in the current layer. To allow a rich representation of the data, there can be multiple feature-maps per layer which in turn means that multiple kernels are learned. This allows a layer to spatially vary the used filter by sourcing a different feature-map, based on the node location.

**Pooling layer** Another concept that is frequently used with CNNs is pooling, which is employed for non-linear down-sampling of the data. Pooling layers partition the image into a set of non-overlapping regions and for each sub-region apply a pooling operator. Typically, pooling is implemented as the maximum over a $2 \times 2$ region. Pooling is not just useful to reduce the dimensionality of the data and thus make processing more efficient, it also provides some translation invariance. This is achieved as the pooling operator effectively discards the exact source location within the pooling region.

These two layer types together with the fully connected MLP form the LeNet [55] family of models (c.f. Figure 2.6). They are characterized by an alternation of convolutional and pooling layers at the bottom which are used for feature extraction and fully connected upper layers that are responsible for classification.

### 2.4.6 Backpropagation

The backpropagation algorithm allows computing the weight updates required in one gradient descent step as in eq. (2.11) efficiently. To this end it takes advantage of the hierarchical structure of the flow-graph (section 2.4.4) and the chain-rule to obtain a *recursive* formulation.

The algorithm proceeds as follows[2]. First, forward propagation is performed to evaluate the network of depth $d$ and compute its outputs $\mathbf{y}$. Then, the error signal $\delta^d$ at the output layer $d$ is computed, which is the gradient of the loss function $L$ with respect to the outputs

$$\delta_i^d = \frac{\delta L}{\delta y_i}.$$

Next, we can descend one layer and compute the error signal for node $i$ at layer $d-1$ using the error signal at the output layer as

$$\delta_i^{d-1} = \left( \sum_m w_{mi}^d \delta_m^d \right) s'_d,$$

where $w_{mi}^d$ is the weight connecting node $i$ to output $m$ and $s'_d$ is the derivative of the scalar activation function.

More generally, using matrix notation, one can obtain the error signal $\delta^{l-1}$ at layer $l-1$ from its parent layer $l$ and the weight matrix $\mathbf{W}$ connecting them as

$$\delta^{l-1} = \mathbf{W}^T \, \delta^l \, s'_d.$$

Finally, we can compute the partial derivatives of the loss function $L$ at layer $l$ by multiplying the error signal by its inputs. E.g. for the input layer we have

$$\frac{\delta L}{\delta \mathbf{W}_0} = \delta^l \mathbf{x}^T.$$

This allows us updating the weights at the respective layer.

Here, the backpropagation algorithm actively re-uses the partial derivatives computed at the higher network layers to compute the partial derivative of lower layers. Furthermore, all required computations can be expressed as matrix $\times$ vector operations which allow for an efficient implementation.

[2]Based on `https://medium.com/@erikhallstrm/backpropagation-from-the-beginning-77356edf427d`

## 2.5 Training deep networks

The amount of nodes in a neural network can be related to its modeling capacity. The major problem during training a neural network is to avoid over-fitting to the training dataset, which is more likely to happen with a high modeling capacity. Over-fitting results in bad generalization and therefore bad performance on unseen data. As usual in machine-learning the complete dataset is divided into a training and testing set. The separate testing set allows drawing conclusions on how the model will perform on unseen data, as it is not used during training.

However, for training a neural-network an additional validation set becomes necessary. This set is neither part of the training nor of the testing set. Instead the validation samples are used to predict the performance of the model on a future testing set. This allows to monitor convergence and optimizing hyper-parameters of the model. These are the parameters one has to choose a priori and which are not optimized during training.

The simplest of those and one that always exists when training a neural network is the number of training *epochs*, i.e. the number of times we feed the whole dataset to the network. As with the parameters optimized during training, we want to avoid over-fitting to the training set — but we are also not allowed to set the optimal value for our testing set, so we still make valid conclusion on the generalization performance.

Therefore, during training we evaluate our model with a fixed frequency (e.g. after each epoch) on the validation set. If we see that the performance on the validation set decreases we terminate the training process as the model is likely over-fitting to the training data. This criterion is called early-stopping.

However, due to the statistic nature of stochastic gradient descent, the validation error can slightly increase only to decrease again in the next epoch. Therefore, the stopping criterion is only a heuristic. Here, we will use a patience value that increases geometrically with the iteration count when a new best validation error is found.

### 2.5.1 Dataset bias

Large, labeled datasets are the integral part of the performance achieved by contemporary CNN architectures as the deep models require massive amounts of labeled data. At the same time, datasets are means of measuring and comparing performance of different algorithms. Therefore, modern datasets [22, 25, 58] try to be a *representation* of the real world — both, to provide a representative measure of algorithm performance and a source for general algorithm models.

However, it was shown [95] that despite the best efforts, the datasets exhibit a strong built-in bias. Consequently, todays state-of-the art algorithms have poor cross-dataset generalization properties, e.g. training on ImageNet [22], but testing on PASCAL VOC [25] results in a considerably degraded performance compared to testing on ImageNet itself. The dataset bias can be attributed to different goals of the datasets: some capture more urban scenes, while others focus on rural landscapes; some use professional photographs, while others use amateur photos from the Internet; some focus on single objects, while others focus on entire scenes. Even though all modern datasets are Internet-mined, this is not a sufficient condition to remove the bias, e.g. ImageNet contains a high amount of racing-cars from canonical views. More generally, the dataset bias was attributed by [95] to the following main factors.

**Selection bias** If a dataset defines a "car" by the rear-view of a racing-car, no algorithm will generalize to a side-view of a sedan. In correspondence, keyword-based image search on the internet only returns particular types of images — especially if user specific search customization is enabled. Ideally data should be obtained from multiple search engines to alleviate the selection bias. On the other hand, the selection might be biased on purpose, e.g. when tackling the problem of detecting texture-less industrial objects.

**Capture bias** Professionally captured photographs typically have well tuned contrast and illumination. However, they almost always show the object of interest in the center — similarly to the results of keyword-based image search on the internet. Furthermore, searching for "mug" on Google Image Search shows a more subtle capture bias; most of the retrieved image will show the mug with a right-facing handle.

**Negative set bias** A dataset does not only define an object by what it is (positive samples) but also by *what it is not* (negative samples). For instance, a classifier supposed to find "boats" might not focus on the boat itself, but rather on the water below or a distant shore if the dataset exhibits this correlation. Therefore, it is important that there is a sufficient negative set including rivers, seas etc. *without boats.*

Additionally, the bias might be intrinsic due to the construction of the dataset. A notable example is when the dataset consists of synthetic images only and one wants to apply the results to real images. Exhibiting such an intrinsic difference, the dataset is considered to come from a specific domain while the real images come from another. The systematic difference between datasets is therefore called the *domain gap*.

### 2.5.2 Bridging the reality gap

It is desirable to adapt the training procedure to learn features, that not only are *discriminative* in the target domain, but also *invariant* to the change of domains [27]. At this, training should still result in a high amount of precision as required by the regression problem of 6D pose estimation.

Nowadays there are two main directions to achieve this goal, namely

**Domain randomization** Here, the parts of the domain are randomized to which the algorithm should not be sensitive to. For example [96] vary rendering parameters, like lights, object pose and object texture. This way the neural network is forced to learn the essential features — that is, the features that are not affected by randomization. More generally, the goal is to increase the domain space such that real images merely become only one of many domain instances. The core advantage of domain randomization approaches is that they do not require any data from the target domain. However, the drawback is that the amount of data grows exponentially with each parameter that is randomized and therefore extends the amount of training time. Furthermore, one has to pay attention not to randomize core cues for the task at hand to achieve the best performance possible. For instance, the camera pose should be restricted to the upper hemisphere for a tabletop detection setting instead of being fully randomized.

**Domain adaptation** When *some* data from the target domain is available, adaptation is possible. Here, *fine-tuning* is the most prominent and simple approach where a network trained on one domain is adapted to a new one by feeding according samples at a low learning rate [70]. However, this requires labeled data from the target domain (supervised adaptation) and can lead to severe overfitting if the target domain dataset is small. Conversely, the approach of [39] is to pre-train a network on real data and then to "fine-tune" on synthetic data. To avoid overfitting of the network to synthetic data they freeze the feature extraction layers.

Ganin et al. [27] use a more integrated method by extending the task network by a domain classifier that shares the deep feature extractor with the task network. During training, an additional step is introduced where the error of the classifier is

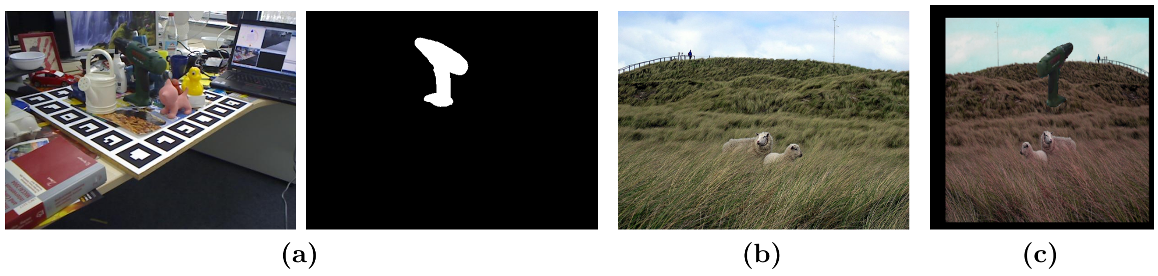


**Figure 2.7:** A standard data augmentation pipeline: given (a) an image with segmentation mask and (b) a random background, (c) a new training sample is generated by composition while also varying hue and scaling

maximized to eliminate any domain information from the feature extractor.

Recent work [111] also shows promising results in combining both approaches as *guided* domain randomization. Here, the augmentations are not chosen randomly but rather by an adversarial guide network that explicitly applies a family of pixel-level perturbations to ensure domain invariance.

### 2.5.3 Data augmentation

A simple domain randomization method to avoid overfitting is data augmentation. Here, no new data is generated, but instead the original training data is randomly perturbed to better model the expected data variation. Typical steps [50, 76, 59, 75, 93] are:

- If segmentation masks are available, replace the background with random images from the PASCAL VOC [25] dataset to avoid overfitting to the background.
- Randomly adjust the exposure and saturation in the HSV color space by a factor of 1.5 and add random Gaussian noise to generalize to different lighting conditions.
- Randomly scale and translate the image by up to a factor of 20%.

See Figure 2.7 for an example of data augmentation applied by [93].

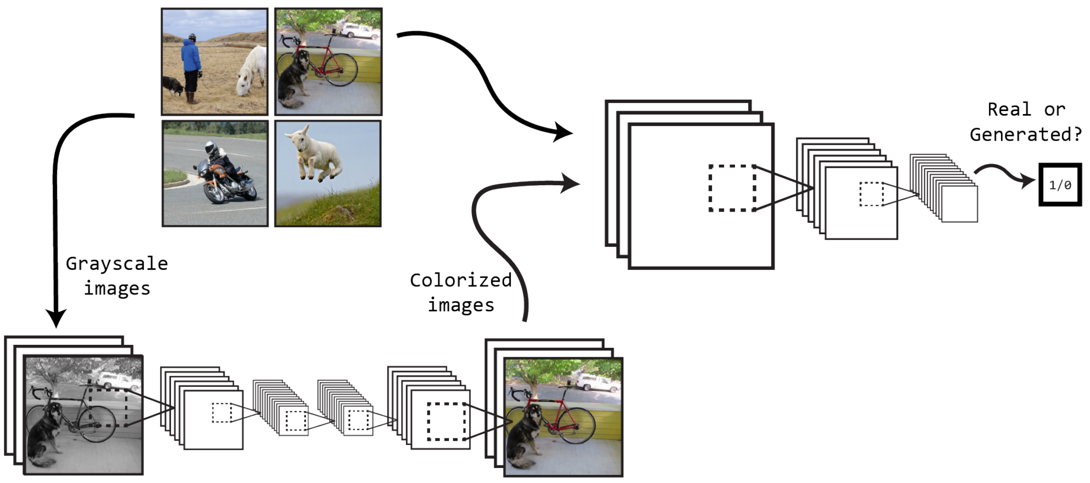


**Figure 2.8:** Conditional generative adversarial network architecture for image colorization. The discriminator (right) acts as an error function to guide the generator (bottom).

## 2.6 Generative adversarial networks

Think of the image colorization task, where a colored image has to be generated based on a gray-scale input[3]. One can easily formulate an according supervised regression problem by creating pairs of colored and gray-scale images and using the squared error loss. However, the ambiguity of the task poses a problem here; when given a gray-scale image of e.g. a red shirt, reconstructing a green shirt will produce a large loss according to the squared error. Assuming all shirt colors are equally likely, the network will then rather keep the image gray-scaled as it results on the minimal loss on average.

This means that rather than predicting the original image, we want the network to predict a plausible image i.e. one that "looks good". Therefore, we have to select a different loss function that takes human perception into account while still being differentiable. While there are loss functions that are constructed to take perception into account, we can also use an additional neural network to act as an error function. The task of this *discriminative* network is to classify whether a given image is the original or a prediction from gray-scale. By back-propagating the error signal of a predicted image, given the "original" class, it then allows us to obtain a pixel-level error that can be further back-propagated to the *generative* colorization network (see Figure 2.8).

This two-model architecture was introduced by [34] and is generally referred to as the generative adversarial network (GAN) as the generative model $G$ and the discrimative model $D$ have adversarial goals. The architecture corresponds

[3]Based on `https://pjreddie.com/courses/computer-vision/`

to a two-player minimax game and extends the machine-learning framework by making the loss function learnable as well.

More formally, we define the GAN loss as

$$\begin{aligned} L_{GAN}(G, D, Y, Z) =& \mathbb{E}_y \left[\log D(y)\right] + \\ & \mathbb{E}_z \left[\log \left(1 - D\left(G(z)\right)\right)\right], \end{aligned} \tag{2.13}$$

where $G : z \rightarrow y$ maps a random noise vector $z$ to an output image domain $Y$, while $D : y \rightarrow \mathbb{R}$ models the probability that $y$ comes from data, instead of being generated by $G$. The training objective is then

$$\arg \min_G \max_D L_{GAN}.$$

Note that this formulation does not yet consider the input image, but rather generates arbitrary output based on noise $z$.

The joint training of the two-model architecture becomes more challenging as one must avoid over-fitting of either of the models as it would stop the learning of the other. Early in learning, $G$ is poor and can easily be rejected by $D$ [34]. Conversely, as learning progresses, $G$ must not win the game either so $D$ still can provide useful gradient information [13]. Failure to stabilize the process can lead to *mode collapse* in $G$, that is all input is mapped to only one output that is accepted by $D$. In the colorization example above this would correspond to assigning the same shirt color to all input images. There is a plethora of training heuristics to ensure stability of GAN training like balancing learning rates or to only progressively grow the image size [48] such that the problem is more tractable at the start.

### 2.6.1 Image-conditional GANs

Depending on the networking architecture $G$ is not only able to modify the pixel values, but can also change the image structure — e.g. with an encoder-decoder architecture as shown in Figure 2.8. However, in many cases like colorization it is desirable to enforce the generated image to structurally resemble the input image. A straightforward solution is to extend the loss (2.13) by the L2 distance to the ground truth as

$$L_{L2} = \mathbb{E}_{y,z} \left[\| \ y - G(z) \ \|_2\right].$$

This leads to a more average grayish color as motivated above. Alternatively, one can feed the input image directly to the generator and the discriminator, leading to the extended loss function

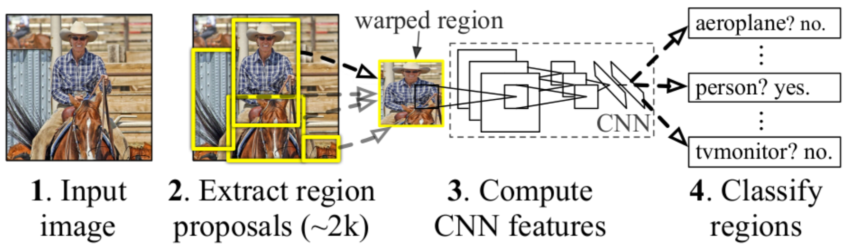


**Figure 2.9:** R-CNN based object detection pipeline: each hypothesized bounding box is resized and fed into a classification network. (Figure from [33])

$$\begin{aligned} L_{cGAN}(G, D, X, Y, Z) =& \mathbb{E}_{x,y}\left[\log D(x, y)\right] + \\ & \mathbb{E}_{x,z}\left[\log\left(1 - D\left(x, G(x, z)\right)\right)\right]. \end{aligned} \tag{2.14}$$

This variant is called *conditional* GAN as both $D$ and $G$ are given the input image $x$. Here, the discriminator $D$ can easily avoid structural deviations of samples from $G$ by comparing with $x$. Note, that the random noise vector $z$ still must be passed to $G$ as it would otherwise produce deterministic outputs and therefore fail to generalize from the actual training data.

## 2.7 Deep learning based object detection

For object detection in images, there are two established classes of methods; one sliding-window based and the other based on region proposal classification. While sliding-window based methods exhaustively search over the whole image at a fixed step size, region proposal methods first generate potential bounding boxes in the image and then run a classifier on the proposals. There are various methods to generate potential bounding boxes — e.g. a simple approach based on the color consistency heuristic would use color-based clustering. As the number of potential locations is much lower than in the sliding window approach, the cost of evaluation at each location may be higher. Before the advent of deep CNN architectures, the state of the art of both approaches had comparable performance [59].

The breakthrough of deep CNN architectures was initially achieved in the classification setting [53]. Furthermore, the deep architectures incur a high evaluation cost which makes sliding-window approaches intractable. Therefore, the first CNN-based detection architecture built upon region proposal methods and re-purposed the classifier of [53] for object detection by R-CNN [33] (see Figure 2.9).

However, the resulting pipeline is still slow, taking about 40 seconds per image, and hard to optimize as each part has to be trained separately. The original

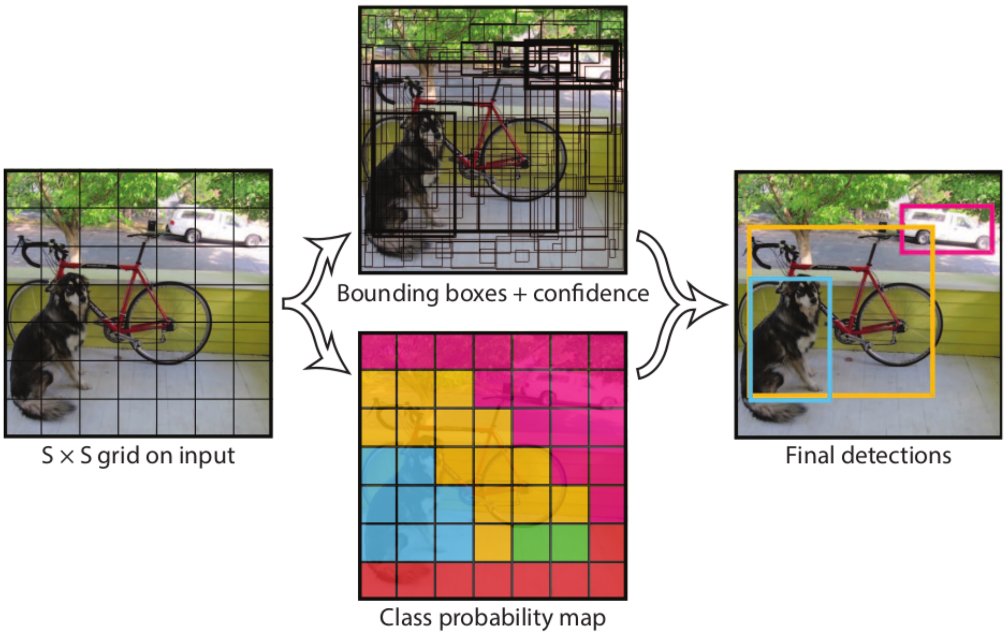


**Figure 2.10:** The YOLO output space discretization; each cell predicts $B$ bounding boxes with a confidence value and $C$ conditional class probabilities, resulting in a $S \times S \times (B \times 5 + C)$ tensor. (Figure from [76])

pipeline has been improved for both faster execution and better detection quality by replacing the region proposal step by a trained CNN that shares the feature-maps with the classifier network [78]. This results in both better region proposals and a reduced number of total computations required which could increase performance to about 142 milliseconds per image.

An alternative approach to object detection is to train a single network to directly output a set of bounding boxes and their respective class labels. With this approach one must reformulate the training objective as neural networks are only able to map a fixed size input to a fixed sized output. In the R-CNN pipeline the network operates in the classification setting; mapping fixed size image crops to a fixed number of class labels. However, to handle detection in only one step, the output must vary with the number of visible objects.

The solution introduced in YOLO [76] is to reverse the scene sampling strategy. In the R-CNN pipeline the input image is discretized by region proposals, where each proposal is associated with a continuous bounding box. Instead, YOLO discretizes the output space into a fixed number of overlapping bounding boxes, while operating densely on the whole input image. Here, the image is divided into a grid of $7 \times 7$ cells (see Figure 2.10), where each cell predicts $C$ conditional class probabilities, and 2 bounding boxes. Each bounding box consists of 5 predictions; $x$, $y$, $w$, $h$ and the probability of containing an object (confidence). For PASCAL

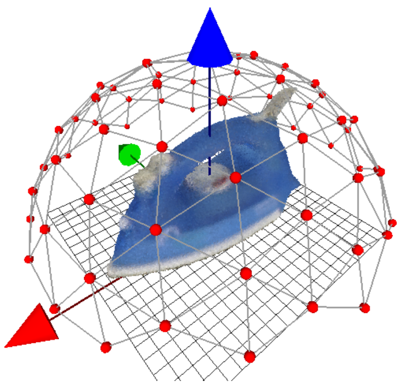
VOC, which contains $C = 20$ classes, the output of the network is consequently a $7 \times 7 \times 30$ tensor of predictions, corresponding to 98 bounding boxes. This formulation combines the classification and regression settings as it regresses the bounding box corners and simultaneously classifies the bounding box content. The per-box confidence is then used to discard predictions with low probability during inference as well as to steer training.

The single stage architecture results in a much higher performance, only requiring 22 milliseconds per image or processing a real-time video stream at 45 fps. Furthermore, the network sees the whole image during training and test time which allows it to implicitly encode context information. For instance, it can reason about spatial co-occurrence of objects instead of only using their appearance. However, the output discretization imposes spatial constraints on object proximity as each cell only predicts 2 bounding boxes and can only have one class. Therefore, the model fails for small objects that appear in groups, like a flock of birds. YOLO uses a fully-convolutional CNN architecture that is inspired by GoogLeNet, that itself is inspired by the original LeNet architecture (see Figure 2.6).

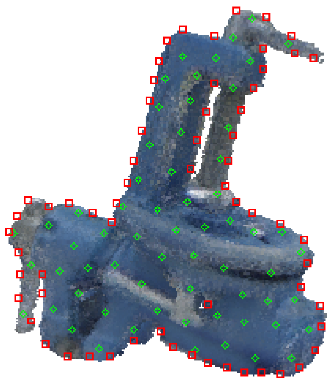
The conceptually similar SSD [59] single stage detector uses an implicit discretization of the output space. Instead of using one fully-connected layer at the end like YOLO, the network explicitly models object scale by using multiple output layers at different feature-map scales that are placed after the feature extractor. As the feature-maps differ in resolution, the bounding boxes are predicted by different $3 \times 3 \times c$ kernels, where $c$ is the depth of the respective feature-map and kernels at different scales differ in number of predicted bounding boxes. SSD uses the last 6 convolutional layers (up to $1 \times 1$) to produce predictions, which results in a total of 8732 generated boxes.

## 2.8 6D object pose estimation

For many real-world applications like augmented-reality and robot manipulation the 2D bounding box as estimated by an object detector is not enough and the estimation of the 6D pose $[\mathbf{R}\ \mathbf{t}]$ is needed. Therefore, the problem of 6D object pose estimation extends the object detection problem outlined above, to simultaneously estimate the object pose as described in section 2.3. Current 6D pose estimation methods can be separated into three classes, as follows:



**(a)** Viewpoints on the icosphere



**(b)** The LineMod template

**Figure 2.11: (a)** LineMod computes a set of templates from different viewpoints. **(b)** Each template contains a fixed number of discretized contour orientations (red) and normal orientations (green) (Figure from [38])

### 2.8.1 Sparse feature-based methods

Classic object pose estimation is based on sparse keypoint detection and feature matching. The keypoints are computed from a local pixel neighborhood, typically at multiple levels of the image pyramid. Appropriately, the feature descriptors are chosen to be scale invariant as well as robust to rotation, affine transform and illumination [60, 67]. After matching the detected keypoints to a canonical template-view of the object, the pose can be recovered with a PnP algorithm. These methods are robust to occlusion and scale to many objects that need to be separated [18]. However, the objects must be exhibiting a sufficient surface texturing for the methods to work reliably. In many practical — especially industrial — applications, texture-less objects are quite common. Therefore, research has shifted towards more general methods in recent years.

### 2.8.2 Contour orientation templates

With texture-less objects the whole object is used as a reference template. The simplest approach here is using a histogram of oriented gradients [21]. A more efficient and real-time capable approach are the dominant orientation templates (DOT) [38]. These store the discretized orientation of the image gradient along the object contour as the template.

By employing a very efficient matching scheme, a high number of templates can be used to encode individual views of the object. These are then associated with 6D poses and thus provide the object identity as well as a rough (quantized) pose estimate. DOT templates were later extended by a depth modality (LineMod) that

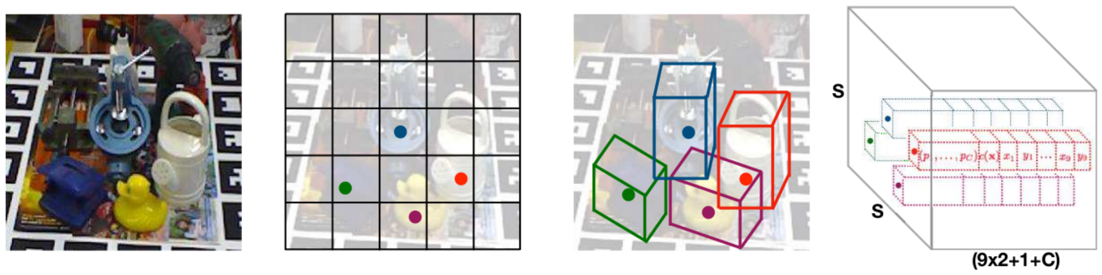


**Figure 2.12:** The YOLO6D tensor; each cell predicts $B$ 3D bounding boxes corners, the center, one confidence value and $C$ conditional class probabilities, resulting in a $S \times S \times (B \times (9 \times 2 + 1 + C))$ tensor. (Figure from [93])

allows incorporating RGBD data as provided by depth cameras. This significantly boosts the performance of the algorithm as surface-normals on the inside of the objects provide a complementary cue to the object contour as captured in RGB images. However, active depth cameras demand a higher power budget and therefore are less suitable for mobile devices. Furthermore, depth sensors typically operate in the infra-red spectrum and therefore do not work in sunlight.

### 2.8.3 CNN-based methods

Initial attempts to solve the 6D object pose estimation problem using a CNN architecture directly regressed the pose [51]. However, pose-estimation has inherent non-linearities and degenerate configurations (as discussed in section 2.3) which requires a higher network capacity and thus complicates training. Later work [9] has shown that predicting intermediate results in the 2D image and then recovering the pose using a PnP algorithm allows improving precision by an order of magnitude.

This idea was first transferred to the single stage detectors by SSD-6D [50] which extend the SSD architecture to also predict a rough estimate of the objects' orientation. Here, a classification setting similar to LineMod [38] was used, where the pose-space is discretized into individual viewpoints and the networks predict the most likely viewpoint and in-plane rotation. The translation is then estimated from the size of the measured 2D-bounding box. This only allows for a very rough pose estimate and requires an involved local refinement step to be useful, which reduces the runtime of the pipeline to 10fps.

In contrast, the YOLO6D [93] architecture (see Figure 2.12) regresses the 2D projections of the corners of the objects' 3D bounding box. From these projections the pose can be recovered using a standard PnP algorithm instead of the rather ad-hoc solution in SSD-6D. Here, the YOLO network was extended to predict the projections of the 8 bounding box corners instead of simply the 2D width and height. Furthermore, using the YOLOv2 architecture each of the class conditional probabilities are predicted per box instead of per cell which allows detecting multiple objects occluding each other. The regression setting results in a high precision of the predicted pose without the need of an additional local refinement step. Therefore, the YOLO6D pipeline is capable of running at 50fps.

# 3
# Camera calibration

In this chapter we present an algorithm for efficiently selecting camera calibration poses, based on already captured calibration frames. We use this property to interactively guide a user through the calibration process and aggregate the resulting calibration data by deploying the whole system on the web.

## 3.1 Introduction

Camera calibration in the context of 3D computer vision is the process of determining the internal camera geometric and optical characteristics (intrinsic parameters) and optionally the position and orientation of the camera frame in the world coordinate system (extrinsic parameters) [99]. The performance of many 3D computer vision and image processing algorithms directly depends on the quality of this calibration. For instance, camera localization as provided by AR systems requires the intrinsic parameters to be known for determining the extrinsic parameters from a single image. Similarly, the intrinsic parameters are required for panoramic stitching to remove any optical distortions present in the input.

Furthermore, calibration is a recurring task that has to be performed each time the setup is changed. Even if a camera is replaced by an equivalent from the same series, the intrinsic parameters may vary due to build inaccuracies. The prevalent approach to camera calibration [112] is based on acquiring multiple images of a planar pattern of known size. It is implemented in many popular computer vision toolboxes like Matlab [7] and OpenCV [11].

However, there are degenerate pose configurations [90] that lead to unreliable solutions. Therefore, the task of calibration cannot be performed by inexperienced users — even researchers working in the field often struggle to quantify what constitutes good calibration images.

In this chapter we use the pinhole camera model with radial and tangential optical distortions as described in section 2.1 and determine the according parameters using the ChArUco calibration pattern presented in section 2.2. The chapter is structured as follows: in Section 3.2 existing methods related to pose selection for camera calibration and their shortcomings are presented.

Section 3.3 introduces a *pose selection* method that finds a compact and robust set of calibration poses and is suitable for interactive calibration. Consequently, singular poses that would lead to an unreliable solution are avoided explicitly, while poses reducing the uncertainty of the calibration are favoured. For this, we use uncertainty propagation. Our method takes advantage of a self-identifying calibration pattern to track the camera pose in real-time. This allows to iteratively guide the user to target poses, until the desired quality level is reached. Therefore, only a sparse set of key-frames is needed for calibration.

In Section 3.4, we then discuss different visualization methods for *user guidance* and evaluate them by performing a user survey. By combining our pose selection method with a suitable visualization, we enable even novel users to perform a precise camera calibration in about 2 minutes.

Next, we propose a web-based *calibration service* in Section 3.5, that not only aggregates calibration data, but also allows calibrating new cameras on-the-fly. This allows general deployment of computer vision algorithms on the web, which was previously not possible due to lack of calibration data.

We conclude with Section 3.6 by giving a summary of our results and discussing the limitations.

## 3.2 Related Work

Camera calibration is influenced most by the choice of camera-to-pattern poses. Here, most existing works only consider individual pinhole camera parameters and do not address the optical distortion parameters. Specifically, there exists research adressing the effect of the angle between image plane and pattern on the estimation error.

- Triggs [98] related the angular spread to the error in focal length. He found a spread of more than $5°$ necessary.

- Sturm and Maybank [90] further investiagated this effect, while evaluating the principal point and focal length indifidually. More importantly, they discussed possible singularities when using one and two planes for calibration and related them to the individual pinhole parameters; e.g. if the pattern is parallel to the image plane in every frame, the focal length cannot be determined.

The results of both were later replicated by [112]. However, the effect of poses on the estimation of the distortion parameters or general camera-to-board poses have not been considered so far.

Another aspect is the quality and quantity of calibration data. Sun and Cooperstock [91] evaluated the sensitivity of camera models to noise, training data quantity and the calibration accuracy in respect to model complexity. However, they only measured the reprojection error on the respective training set, which is subject to over-fitting as was discussed in section 2.1. To overcome this, Richardson et al. [79] introduce the Max Expected Reprojection Error (Max ERE) metric that instead correlates with the testing error and thus allows a meaningful test for convergence.

Furthermore, they automatically compute a "best next pose" and use it for user guidance as an overlaid projection of the pattern. The poses are selected by performing an exhaustive search in a fixed set of about 60 candidate poses. For each pose a hypothetical calibration including this pose is performed and the pose that minimizes the Max ERE is selected. Therefore, searching for the next candidate pose becomes increasingly involved as the size of the calibration set grows, significantly slowing down the calibration process. Additionally, the candidate poses are uniformly distributed in the field of view and do not explicitly consider the angular spread and degenerate cases [90].

In the broader context of user assistance for AR calibration tasks, intrinsic camera calibration was not specifically considered yet. Most closely related is the work of [71], which only describes a guidance system for camera-extrinsic and hand-eye calibration.

## 3.3 Efficient pose selection

Our key idea is to analytically generate optimal pattern poses while explicitly avoiding the degenerate pose configurations. For this, we relate constraints on individual parameters to specific camera to board poses while considering the data captured so far. This allows us to generate an dynamically adapting sequence of poses, that constrains all intrinsic parameters and ensures an accurate calibration. The analytic approach reduces the computation time from seconds to milliseconds compared to the exhaustive search of [79].

On a high level, our approach works as follows. We assess the uncertainty of the calibration parameters using the covariance of the current solution. Here, we establish key pose configurations that reduce variance of parameters groups. For selecting the next pose, we determine the group which the most uncertain parameter belongs to and generate a pose that adds constraints specific to this group. This successively constrains all parameters, resulting in a reliable calibration. Here, we show that the parameter covariance can be used as a convergence criterion, as it correlates with the testing error. Our method explicitly specifies individual key-frames which are used with the calibration method of Zhang [112]. This is opposed to using consecutive frames of a video sequence which leads to many similar poses and thus can bias the calibration.

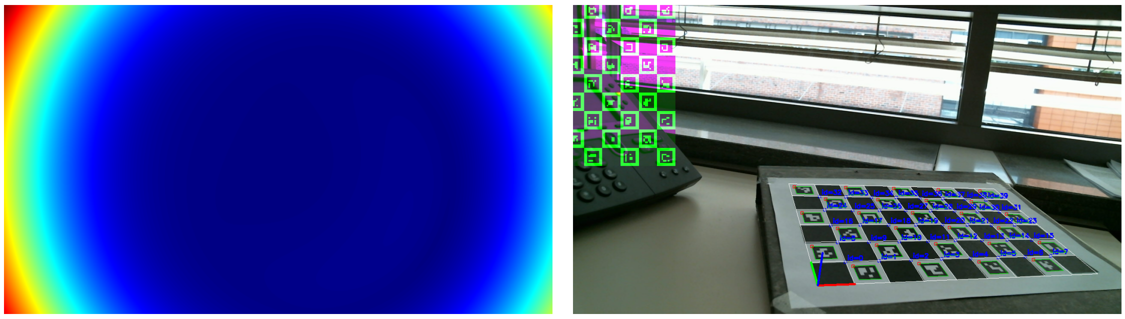
Based on the above, our key contributions are;

1. Empirical evidence for the need of two distinct pose selection strategies and

2. an efficient pose selection scheme for implementing both of them.

In the following, first the relation of intrinsic parameters and board poses is discussed to motivate our split of the parameter vector into two groups of pinhole and distortion parameters. For each parameter group we then establish a set of rules to generate an optimal pose while explicitly avoiding degenerate configurations.

Subsections 3.3.1 - 3.3.3 motivate and describe our novel pose selection method, while Subsection 3.3.5 describes the full calibration pipeline. In Subsections 3.3.6 and 3.3.7 the method is evaluated on real and synthetic data and compared with OpenCV [11] and AprilCal [79] calibration methods. Furthermore, the compactness of the resulting calibration is analyzed.



**(a)** Estimated distortion map **(b)** Target pose at max. distortion

**Figure 3.1:** Distortion map showing the magnitude of $\Delta(\mathbf{p})$ for each pixel. To find the target pose we apply thresholding and fit an axis aligned bounding box. (see supplemental material at `https://youtu.be/aDzUNgVihdQ`)

### 3.3.1 Splitting pinhole and distortion parameters

Looking at eq. (2.1), which we repeat here for clarity, we see that both $\mathbf{K}$ and $\Delta(\cdot)$ are applied at post-projection and thus merely represent 2D-to-2D mappings;

$$\mathbf{p} = \mathbf{K}\,\Delta(\frac{1}{Z_c}\,[\mathbf{R}\;\mathbf{t}]\,\mathbf{P}).$$

Therefore, one might consider estimating $\mathbf{C}$ just from one board pose that uniformly samples the image. However, both intrinsic and extrinsic parameters are estimated simultaneously by [112], which results in ambiguities.

For instance, assume $\mathbf{R} = \mathbf{I}$ and the distortion parameters to be zero. By multiplying out (2.1) we get

$$\mathbf{p} = \begin{bmatrix} \dfrac{f_x(X+t_x)}{Z+t_z} + c_x \\ \dfrac{f_y(Y+t_y)}{Z+t_z} + c_y \end{bmatrix} \tag{3.1}$$

for all pattern points $\mathbf{P}$. In this case there are two ambiguities between

1. the focal length $f$ and the distance to camera $t_z$ and
2. the in-plane translation $[t_x, t_y]$ and principal point $[c_x, c_y]$.

These ambiguities can be resolved by requiring the pattern to be tilted towards the image plane such that there is only one $\mathbf{t}$ that satisfies eq. (2.1) for all pattern points.

Considering the distortion parameters of $\Delta(\cdot)$ on the other hand, there are no similar ambiguities due to the non-linearity of the mapping. The parameters are rather determined by the maximal distortion strength evident in the image. Here, it is more important to accurately measure the distortion in the corresponding image regions (see Figure 3.1a). Note that when only considering radial distortion, one could take advantage of the radial symmetry around the principal point. However, in our case this is not applicable as we are estimating both radial and tangential distortion simultaneously.

To account for the different pose requirements, we split the parameter vector $\mathbf{C}$ into $\mathbf{C}_K = [f_x, f_y, c_x, c_y]$ and $\mathbf{C}_\Delta = [k_1, k_2, k_3, p_1, p_2]$ and consider each group separately.

### 3.3.2 Avoiding pinhole singularities

While optimizing parameters in $\mathbf{C}_K$, singular poses must be avoided. In addition to the case discussed above, we incorporate the cases identified in [90]. Particularly, we restrict the 3D configuration of the calibration pattern as follows:

- The pattern must not be parallel to the image plane.
- The pattern must not be parallel to one of the image axes.
- Given two patterns, the "reflection constraint" must be fulfilled. This means that the vanishing lines of the two planes are not reflections of each other along both a horizontal and a vertical line in the image.

These restrictions ensure that each pose adds information that further constrains the pinhole parameters.

### 3.3.3 Pose generation

As described in Section 3.3.1, each parameter group requires a different strategy to generate an optimal calibration pose.

For the intrinsic parameters $\mathbf{C}_K$ we follow [98, 112] and aim at maximizing the angular spread between image plane and calibration pattern. Accordingly, poses are generated as follows:

1. We choose a distance such that the whole pattern is visible, maximizing the amount of observed 2D points.
2. Depending on the principal axis (e.g. $x$ for $f_y$) the pattern is tilted in the range of $(-70°; 70°)$ around that axis. The actual angle is interpolated using the sequence $[0.25, 0.75, 0.125, 0.375, \ldots]$ which corresponds to the binary subdivision of the $(0; 1)$ range (see Figure 3.2). This strategy, as desired, maximizes the angular spread.

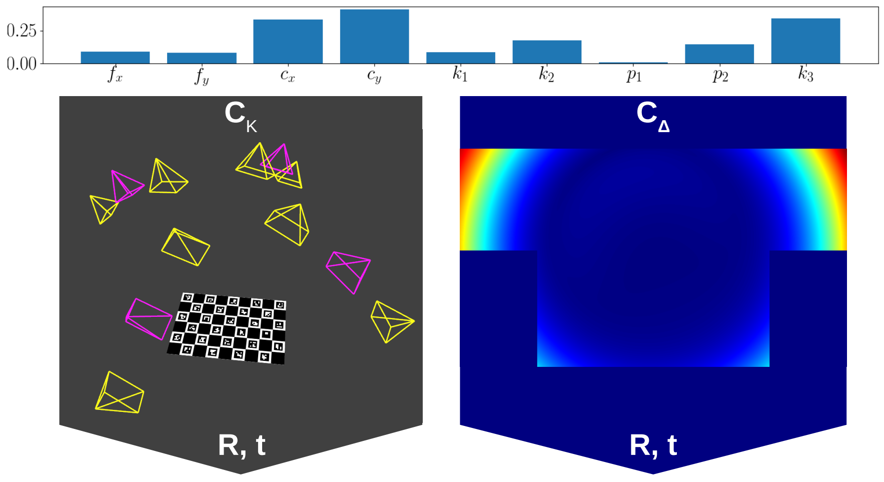


**Figure 3.2:** Exemplary pose selection state. *Top:* Index of dispersion. *Left:* Intrinsic calibration position candidates after one (magenta) and two (yellow) subdivision steps . *Right:* Distortion map with already visited regions masked out.

3. The resulting pose would still be parallel to one of the image axes which prevents the estimation of the principal point along that axis [90]. Therefore, the resulting view is rotated by 22.5° which implements this requirement while keeping the principal orientation.

4. When determining $[c_x, c_y]$ the view is further shifted along the respective image axis by 5% of the image size. This increases the spread along that axis and reduced the number of frames needed for convergence in our experiments.

For the distortion parameters $\mathbf{C}_\Delta$ the goal is to increase sampling accuracy in image regions exhibiting strong distortions. For this, we generate a distortion map based on the current calibration estimate that encodes the displacement for each pixel. Using this map, we search for the distorted regions as follows:

1. Threshold the distortion map (Figure 3.1a) to find the region with the strongest distortion.

2. Given the threshold image, an axis aligned bounding box (AABB) is fitted to the region, corresponding to a parallel view on the pattern. Note that the constraints for $\mathbf{C}_K$ do not apply here.

3. The area covered by the AABB is excluded from subsequent searches (see Figure 3.2). Effectively, the distorted regions are visited in order of distortion strength.

4. The pattern is aligned with the top-left corner of the AABB and positioned at a depth s.t. its projection covers 33% of the image width.

The angular range and width limits mentioned above were set such that the calibration pattern could be reliably detected using the Logitech C525 camera.

### 3.3.4 Initialization

The underlying calibration method [112] requires at least two views of the pattern for an initial solution which we select as follows:

- For the parameters $\mathbf{C}_K$ a pose tilted by 45° around $x$ is selected (see Section 3.3.3). This particular angle was suggested by [112] and lies in between the extrema of 0° where the focal length cannot be determined and 90° where the aspect ratio and principal point cannot be determined.
- Without any prior knowledge we aim at an uniform sampling for estimating $\mathbf{C}_\Delta$. To this end we compute a pose such that the pattern is parallel to the image plane and covers the whole view. While this violates the axis alignment requirements for $\mathbf{C}_K$ poses, it still provides extra information as it is not co-planar to the first pose [112]. Furthermore, the reflection constraint is fulfilled.

To render an accurate overlay for the first pose without prior knowledge of the used camera, we employ a bootstrapping strategy similar to [79]; if the pattern can be detected, we perform a single frame calibration estimating the focal length only — the principal point is fixed at the center and $\mathbf{C}_\Delta$ is set to zero.

### 3.3.5 Calibration process

In the following we present the parameter refinement and user guidance parts as well as any employed heuristics. This completes the calibration pipeline as used for the real data experiments.

**Parameter refinement** After obtaining an initial solution using two key-frames, the goal is to minimize the cumulated variance $\sum_i \sigma_i^2 \mid \sigma_i^2 \in \boldsymbol{\sigma}_C^2$ of the estimated parameters $\mathbf{C}$. We approach this problem by targeting the variance of a single parameter $C_i \in \mathbf{C}$ at a time. Here we pick the parameter with the highest index of dispersion (MaxIOD) $\sigma_i^2/C_i$ ($\sigma_i^2$ iff $C_i = 0$). Depending on the parameter group, a pose is then generated as described in Section 3.3.3.

**Convergence** For determining convergence, we use a ratio test of the parameter variance $r = \sigma^2_{i,n+1}/\sigma^2_{i,n}$. If the reduction $1 - r$ is below a given threshold, we assume the parameter to be converged and exclude it from further refinement. Here, we only consider parameters from the same group as there is typically only little reduction in the complementary group. The calibration terminates once all parameters $\mathbf{C}$ have converged.

**Heuristics** Throughout the process, we enforce the common heuristic [35, §7.2] that the number of constraints should exceed the number of unknowns by a factor of five. The used calibration method [112] not only estimates the intrinsic parameters $\mathbf{C}$, but also the relative pose of model plane and image plane i.e. the parameters $\mathbf{R}$, a 3D rotation, and $\mathbf{t}$, a 3D translation. When using $M$ calibration images we thus have $d = 9 + 6M$ unknowns and each point correspondence provides two constraints. For initialization ($M = 2$) we thus have 21 unknowns, meaning 52.5 point correspondences are needed in total or 27 correspondences per frame. For any subsequent frame only 15 points are required.

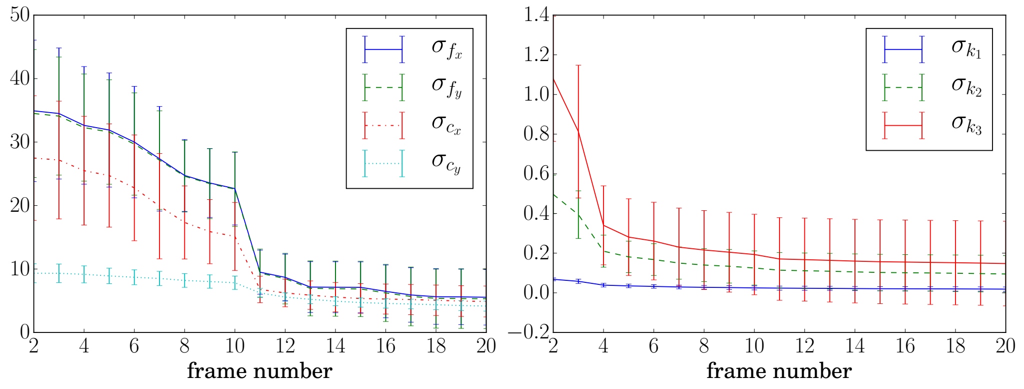
To prevent inaccurate measurements due to motion blur and rolling shutter artifacts the pattern should be still. To ensure this we require all points to be re-detected in the consecutive frame and the mean motion of the points to be smaller than 1.5px (determined empirically).

### 3.3.6 Evaluation

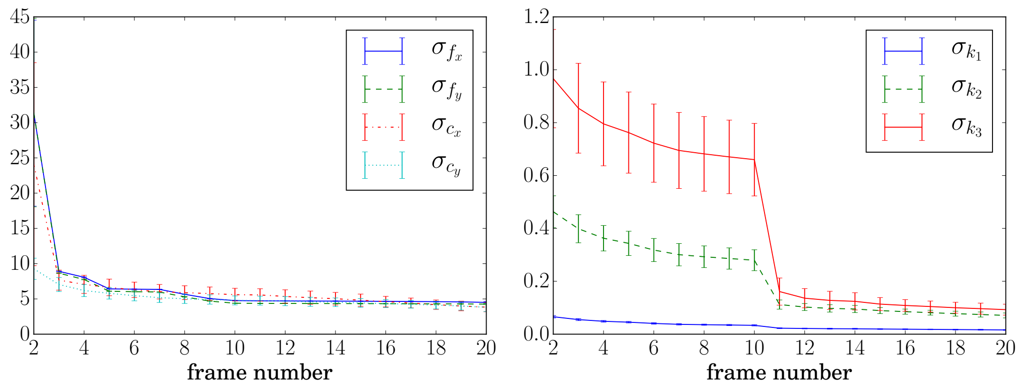
The presented method was evaluated on both synthetic and real data. The synthetic experiments aimed at validating the parameter splitting and pose generation rules presented in Section 3.3, while the real data was used for comparison with other methods. Furthermore, the compactness of the results with real data was estimated. For this, we compared the number of frames selected by our algorithm to an offline method that tried to find a subset with an equivalent testing error.

**Synthetic data** We performed multiple calibrations, each using 20 synthetic images. The first two camera poses were chosen as described in section 3.3.4 to allow a rough initial solution. The next 8 poses were chosen to optimize $\mathbf{C}_\Delta$ while the last 10 poses were optimizing $\mathbf{C}_K$ (and vice versa).

The camera parameters were based on the calibration parameters of a Logitech C525 camera $\mathbf{C}_{real}$. However, the actual parameters were sampled around $\mathbf{C}_{real}$



**(a)** Poses 11-20 are optimizing $\mathbf{C}_K$.



**(b)** Poses 11-20 are optimizing $\mathbf{C}_\Delta$.

**Figure 3.3:** Correlation of pose selection strategies and calibration parameter uncertainty expressed using the standard deviation $\sigma$ (thus the error bars mean "variance of $\sigma$"). The first two poses are selected according to the initialization method. Poses 2-10 and 11-20 are selected by complementary strategies. Evaluated with synthetic images on 20 camera models sampled around the estimate of the Logitech C525 camera.

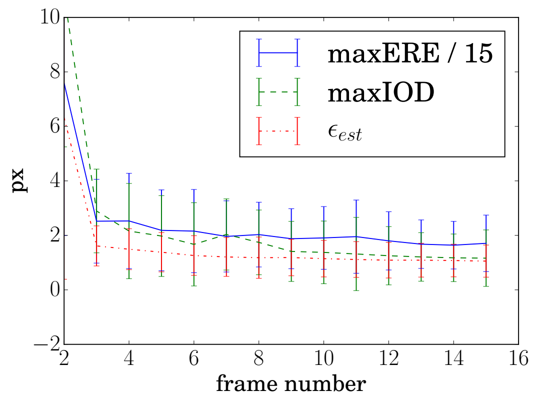
using a covariance matrix that allowed 10% deviation for each of the parameters $\Sigma = diag(0.1 \cdot \mathbf{C}_{real})$ as

$$\mathbf{C} \sim \mathcal{N}(\mathbf{C}_{real}, \Sigma). \tag{3.2}$$

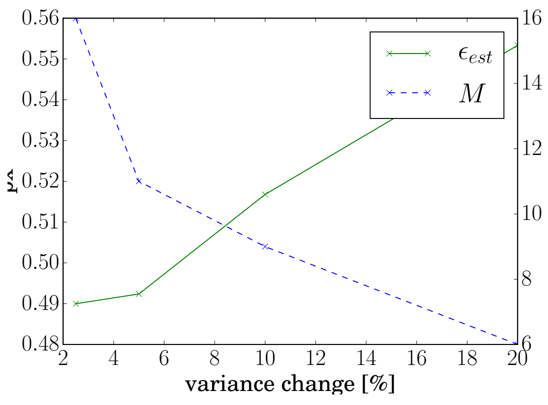
Therefore, each synthetic calibration corresponds to using a different camera $\mathbf{C}$ with known ground truth parameters. To allow generalization to different camera models, we kept the above pose generation sequence, but used 20 different cameras $\mathbf{C}$.

Figure 3.3 shows the mean standard deviation $\boldsymbol{\sigma}_C$ of the parameters. Notably, there is a significant drop in $\sigma$ iff a pose matching the parameter group is used.

We also evaluated the usage of MaxIOD as an error metric by comparing it to MaxERE [79] and a known estimation error $\epsilon_{est}$. Just as the MaxERE, the MaxIOD correlates with $\epsilon_{est}$ (see Figure 3.4a). Additionally, as Figure 3.4b indicates,



**(a)**



**(b)**

**Figure 3.4:** *(a)* Comparing error metrics on synthetic data: both MaxERE and the proposed Max IOD correlate with estimation error $\epsilon_{est}$. (standard deviation over 20 samples) *(b)* Required number of frames $M$ and $\epsilon_{est}$ in respect to the variance reduction threshold

| Method | mean $\epsilon_{est}$ | frames used | mean $\epsilon_{res}$ |
|---|---|---|---|
| Pose selection | 0.518 | 9.4 | 0.470 |
| OpenCV [11] | 1.465 | 10 | 0.345 |
| AprilCal [79] | 0.815 | 13.4 | 1.540 |
| Compactness test | 0.514 | 7 | 0.476 |

**Table 3.1:** Our method compared to AprilCal and OpenCV on real data. Showing the average over five runs. Training on the testing set results in $\epsilon_{est} = 0.479$.

the IOD reduction is suitable for balancing calibration quality and the number of required calibration frames.

**Real data** For evaluating our method with real images, we recorded a separate testing set consisting of 50 images at various distances and angles covering the whole field of view. All images were captured using a Logitech C525 webcam at a resolution of 1280x720px. The autofocus was fixed throughout the whole evaluation, while exposure was fixed per sequence. Our method was compared to AprilCal [79] and calibrating without any pose restrictions using OpenCV.

We used the pattern described in section 2.2 that provides 40 measurements per frame for OpenCV as well as for our method. With AprilCal, we used the 5x7 AprilTag target that generates approximately the same amount of measurements.

The convergence threshold was set to 10% for our method and the stopping accuracy parameter of AprilCal was set to 2.0. As the OpenCV method does not provide convergence monitoring, we stopped calibration after 10 frames here.

Table 3.1 shows the mean results over 5 calibration runs for each method, measuring the required number of frames, $\epsilon_{est}$ and $\epsilon_{res}$. Here, our method requires only 70% of the frames required by AprilCal while arriving at a 36% lower $\epsilon_{est}$ (64% compared to OpenCV).

### 3.3.7 Analyzing the calibration compactness

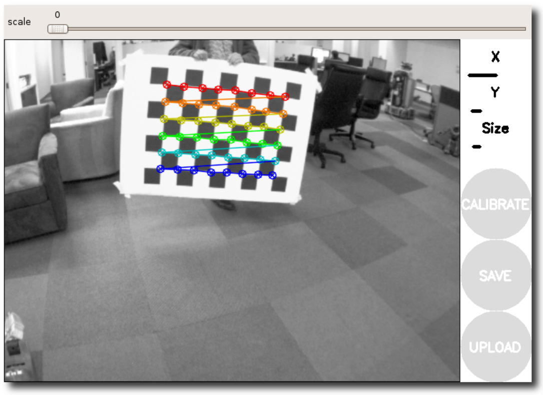
The results in the previous section show that our method is able to provide the lowest calibration error $\epsilon_{est}$ while using fewer calibration frames then comparable approaches. However, it is not clear whether the solution is using the minimal amount of frames or whether it is possible to use a subset of frames while arriving at the same calibration error.

Therefore, we further tested the compactness of our calibration result. We used a greedy algorithm that, given a set of frames captured by our method, tries to find a smaller subset. It optimizes for the testing set, directly minimizing the estimation error.

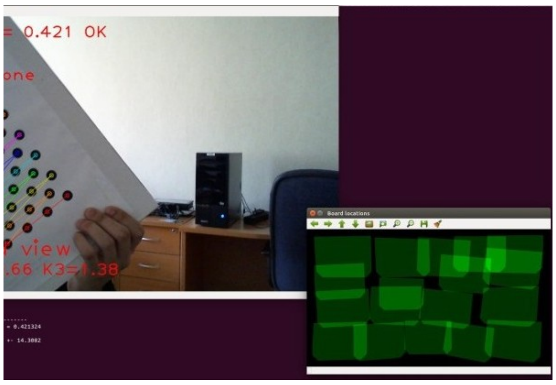
The algorithm is computed as follows; given a set of training images (the calibration sequence)

1. the initialization frames as described in Section 3.3.4 are added unconditionally;
2. each of the remaining frames is now *individually* added to the key-frame set and a calibration is computed.
3. For each calibration the estimation error $\epsilon_{est}$ is computed using the testing frames.
4. The frame that minimizes $\epsilon_{est}$ is incorporated into the key-frame set. Continue at step 2.
5. Terminate if $\epsilon_{est}$ cannot be further reduced or all frames have been used.

The greedy optimal solution requires 75% of the frames compared to the proposed method while keeping the same estimation error (see Table 3.1). This indicates that, while a significant improvement over [79], our method is not yet optimal in the compactness sense. The greedy algorithm requires an a priori recorded testing set and only finds a minimal subset of an existing calibration sequence, but cannot generate any calibration poses.

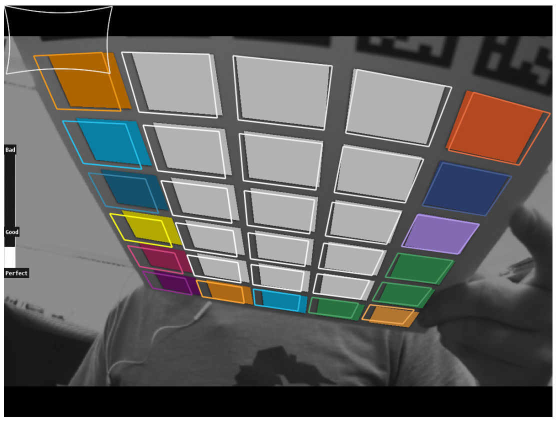


**(a)** The ROS toolbox showing the variance in position and size.

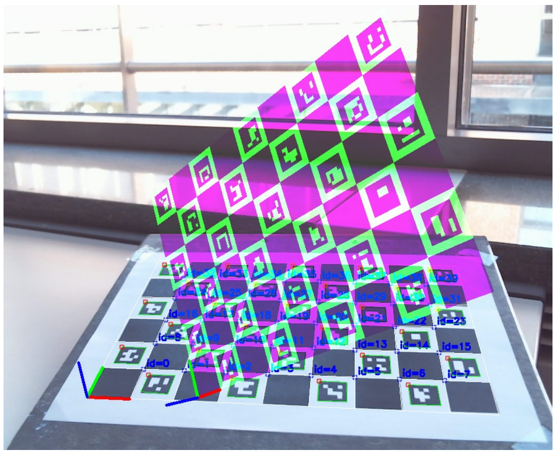


**(b)** OpenCV showing the current screen coverage

**Figure 3.5:** User interfaces of popular, non-interactive, systems.



**(a)** Target pose wireframe and real-board over-paint as used by [79]



**(b)** Target pose projection as used by [83]

**Figure 3.6:** User guidance overlays used by interactive calibration systems

## 3.4 User guidance

Popular calibration toolboxes like ROS[1] or OpenCV [2] impose some heuristics on pose variance or screen space coverage to alleviate the problem (see figure 3.5).

As these systems are not capable of generating pose suggestions, their user interfaces only visualize statistics about the data captured so far. The user is responsible to reason about an optimal next pose that would improve on the imposed heuristics. Furthermore the unreliable pose configurations are not explicitly addressed — therefore degraded performance is still possible.

[1]http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration

[2]https://docs.opencv.org/master/d7/d21/tutorial_interactive_calibration.html

In contrast, new calibration systems [79, 83] are capable to guide users to specific target poses by displaying an overlay (see figure 3.6). This explicitly avoids unreliable configurations and reduces intrinsic cognitive load [17]. While both [79, 83] performed user surveys, they merely showed operability of their methods by novice users.

To guide the user, the targeted camera pose is projected using the current estimate of the intrinsic parameters. This projection is then displayed as an overlay on top of the live video stream.

To verify whether the user is sufficiently close to the target pose we use the Jaccard index (2.8) computed from the area covered by the projection of pattern from the target pose $T$ and the area covered by the projection from the current pose estimate $E$. We assume that the user has reached the desired pose if $J(T, E) > 0.8$.

Comparing the projection overlap instead of using the estimated pose directly is more robust since the pose estimate is often unreliable — especially during initialization.

Yet the user interfaces implemented by each method are very different. [83] only display highlighted projection of the real pattern to tag the target pose, while [79] display an abstractly colored, wireframe of the board at the target pose and additionally overpaint the real board with squares of matching color (see Figure 3.6).

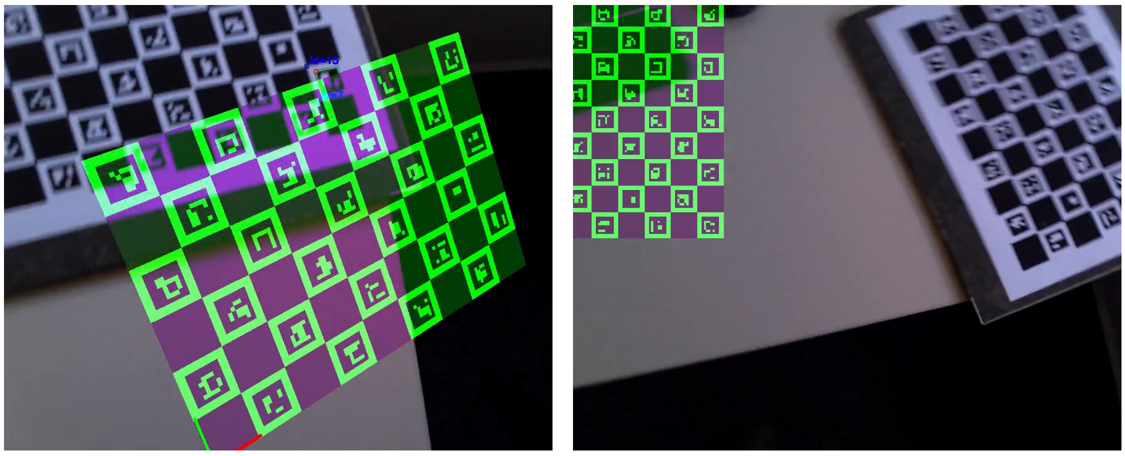
Therefore, this work focuses on the question which user interface is best suited to guide users to specific calibration poses. At this we take the specific geometric properties of the calibration problem into account, namely:

- Only the relative pose between camera and pattern matters
- The pattern can be arbitrarily flipped horizontally and vertically.

Indeed, these properties make the calibration guidance significantly different from typical AR guidance use-cases where a pose needs to be matched exactly.

### 3.4.1 Calibration poses

In general a rigid pose has six degrees of freedom (DOF); yaw, pitch, roll for the orientation and the three-dimensional position. However, the underlying algorithm [83] generates more restricted poses, based on the calibration objective. These fall in the following two categories:



**(a)** Intrinsic calibration pose **(b)** Distortion calibration pose

**Figure 3.7:** Exemplary view from the two pose categories

**Intrinsic calibration pose** To estimate the intrinsic camera parameters, the goal is to maximize the angular spread of the measurement points. Here the pattern is placed in the central image region and tilted along one primary axis. Additionally, the board needs to be tilted and rotated along the remaining axes to avoid ambiguous configurations (see Figure 3.7a). Therefore there are only three rotational DOF.

**Distortion calibration pose** To estimate the lens distortion parameters, the pattern must be placed in regions with highest distortion which are typically the corners. Here a parallel view is used and the distance and relative position changes (see Figure 3.7b). Therefore, there are only three positional DOF.

Therefore, a user only ever has to change 3 DOF when starting from a central, parallel view on the pattern.

## 3.4.2 Method

To evaluate different user guidance options, we performed two user surveys, measuring the time the users required to match a series of target poses. The participants were students and co-workers at our lab. Most of them had never performed a camera calibration before and all users were using the tool for the first time. The pose sequence was given by our system [83].

The only instruction given was that the calibration pattern should be matched with the displayed overlay.

We triggered the time measurement only after the first target pose was reached. This explicitly discards the time the users needed to accommodate to the calibration scenario and the system setup.

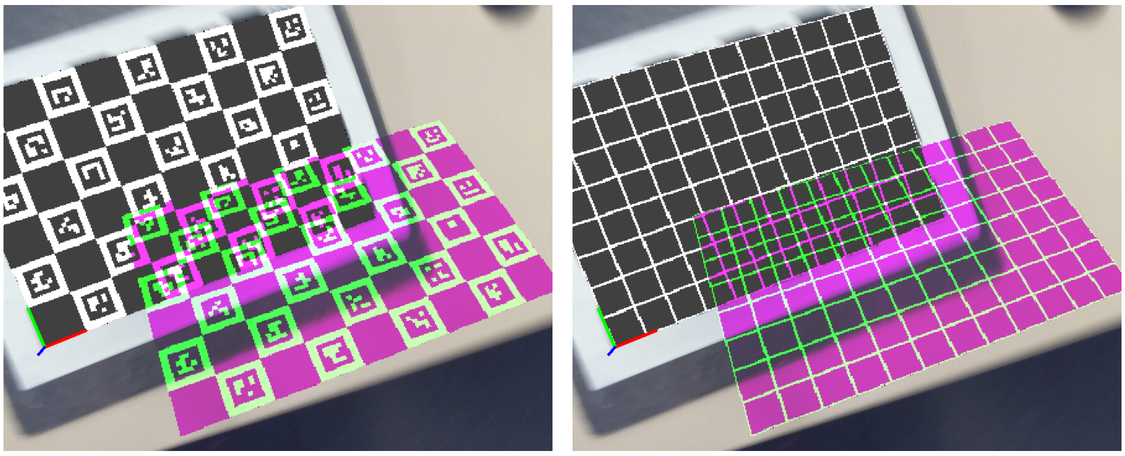

(a) The default "chessboard" pattern (b) The "quadrille paper" pattern

**Figure 3.8:** The two overlay patterns we evaluated in our second user survey. Note that we now do overpaint the real board in the video stream.

For each question a separate survey was performed. The surveys were several months apart time-wise. Hence, there is no overlap of participants and the pose setup varies slightly.

### 3.4.3 Relative motion survey

The goal of the first survey was to determine whether moving the camera or moving the calibration pattern is preferable. This takes advantage of the fact that only the relative orientation and translation between camera and pattern matters. Therefore, we evaluated the following two scenarios:

1. Fixing the camera position at the screen and let the user move the pattern like in front of a virtual mirror.

2. Fixing the pattern position and let the user move the camera in a first-person-view like fashion.

There were 5 participants in this survey which successively tried both options. To exclude the effect of familiarization we randomized the order of the options. The user guidance consisted only of the target pose overlay as shown in Figure 3.7. There were 9 target poses that had to be matched.

| | t (first-person view) | t (virtual mirror) |
|---|---|---|
| User 1 | 1:39 min | 1:44 min |
| User 2 | 3:17 min | 1:08 min |
| User 3 | 2:46 min | 1:55 min |
| User 4 | 7:22 min | 1:36 min |
| User 5 | 2:22 min | 1:25 min |
| Mean | 3:29 min | 1:33 min |

**Table 3.2:** time the users required to reach 9 given target poses.

### 3.4.4 Pattern appearance survey

Complementing the first survey, the second survey determined whether one can take advantage of the geometric property that the pattern can be flipped horizontally and vertically. To this end we chosen two different visualization of the calibration pattern as follows:

1. The asymmetric chessboard as in used for the preceding survey.
2. A quadrille paper visualization, which is fully symmetric yet still contains the necessary perspective cues.

To keep the connection between the target pose overlay and the physical calibration board when using the new visualization, we overpaint the actual calibration target in the video stream - similarly to [79]. We also apply the over-painting to the first option (see figure 3.8) to exclude the effect of tracking imprecision from the survey.

There were 7 Participants in this survey which had to reach 10 target poses. As with the preceding survey the order of the options was randomized so 3 participants started with option 1 and 3 participants started with option 2.

We only used the "virtual mirror" setup based on the results from the first survey.

### 3.4.5 Results

In the following the results of our user surveys are shown. First we discuss the quantitative timings of each experiment. Then we also present some qualitative observations made during the trials.

| | t (chessboard) | t (quadrille paper) |
|---|---|---|
| User 1 | 2:14 min | 1:00 min |
| User 2 | 2:07 min | 1:20 min |
| User 3 | 3:06 min | 2:11 min |
| User 4 | 3:43 min | 3:20 min |
| User 5 | 1:21 min | 1:44 min |
| User 6 | 1:52 min | 2:00 min |
| Mean | 2:24 min | 1:56 min |

**Table 3.3:** time the users required to reach 10 given target poses with different visualizations

**Quantitative results** Table 3.2 shows the quantitative results of the first survey, giving the per-user times as well as the overall average.

The average calibration time of 1:33 min to complete the calibration show a strong advantage of the virtual mirror scenario over the first-person view approach with an average time of 3:29 min. Looking at the individual results we see that only User 1 is slightly faster using the first-person view, while all other Users were considerably faster using the virtual mirror approach. User 4 even struggles to complete the calibration using in the first-person view. Therefore, we conclude that the virtual mirror approach is preferable.

Table 3.3 shows the results of the second survey, again giving the average as well as the per user times. There are only 6 results given as one participant failed to match the first intrinsic pose within 3 min with any method. Therefore, we aborted the trial and no results are given.

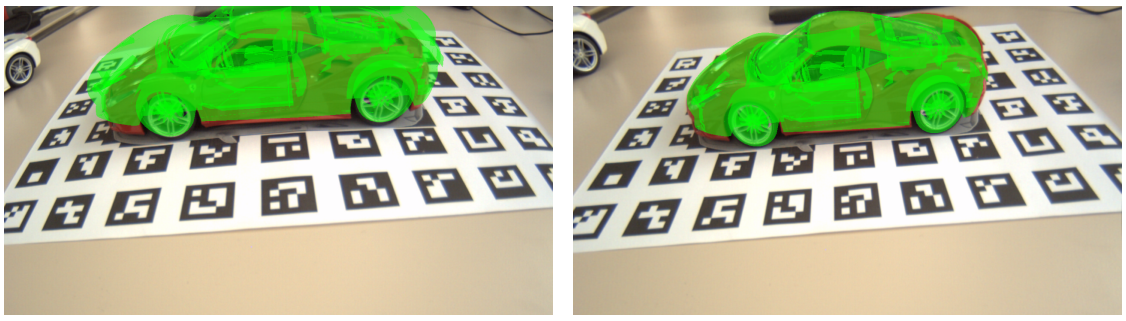
The average time of 1:56 min to complete the calibration using the quadrille paper visualization shows a slight advantage over the chessboard visualization with 2:24 min. However, looking at the individual results there are 2 participants being faster using the chessboard visualization. Furthermore, there is strong variation between the individual users. Consequently, a larger number of participants will be needed to draw a clear conclusion here.

**Qualitative results** Additionally, to the times presented above we made the following qualitative observations:

- It took the participants much longer to match the intrinsic pose then the distortion pose.
- With the "quadrille paper" pattern, some users did not rotate the pattern to match the distortion calibration pose, but rather moved it out of view.

- The users reached a target pose faster if it was from the same category as the previous one; e.g. if a distortion pose followed a distortion pose. Conversely, the needed to re-orient if e.g. a distortion pose followed a intrinsic pose.
- When asked about the experience users preferred the "quadrille paper" visualization - even if their calibration time was higher in this mode.

Here, the time it took the participants to match the intrinsic pose was the determining factor in overall calibration time.



**(a)** Calibrated camera matrix only **(b)** Fully calibrated camera

**Figure 3.9:** Effect of camera calibration on an augmented reality scene: Although a calibrated camera matrix is used in (a), the misalignment is clearly visible. Using a complete distortion model allows rectifying the image. Together with an adapted camera matrix, this results in a fully aligned augmentation (b).

## 3.5 Building a calibration database

In this section we describe our calibration service "calibDB" in detail. First we discuss the high-level architecture and internal protocol of the service. Then we describe the external API and data format used for calibration data retrieval and acquisition. Finally, we discuss how the current WebXR API should be extended to seamlessly provide calibration data to computer vision applications. A live version of the presented service is available at `https://www.calibdb.net/`.

Lens distortion is currently not handled by the WebXR API, which only exposes the camera matrix. While the effect of lens distortion can be neglected on simple webcams which resemble the pinhole optics, this does not hold generally.

Figure 3.9a shows an image captured with the Computar E3Z4518CS lens with an AR-overlay rendered with a matching camera matrix. As can be seen the AR-overlay diverges from the image towards the image edges. Rectifying the image by inverting eq. (3.3) and adapting $\mathbf{K}$ accordingly, we can make the overlay fit the image as can be seen in Figure 3.9b.

### 3.5.1 Common lens distortion models

The camera parameters recovered during calibration (see section 2.1) are typically the camera matrix $\mathbf{K} \in \mathbb{R}^{3x3}$ and a set of lens distortion coefficients $\mathbf{d} = [k_0, \dots, k_n]$.

The lens distortion function $\Delta(\cdot)$ then typically models a radial distortion as

$$\Delta_R(\mathbf{p}) = \mathbf{p}\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right). \tag{3.3}$$

Web-based computer vision should not be restricted to webcam imagery, therefore we have to expect all kinds of cameras. Eq. (3.3) is also specified in the DNG image format [1] as *WarpRectilinear* for processing images from interchangeable-lens cameras.

Additionally, the DNG format includes a specialized distortion model for fisheye lenses [47], *WarpFisheye*:

$$\Delta_F(\mathbf{p}) = \mathbf{p}\frac{1}{r}\left(\theta + k_1\theta^3 + k_2\theta^5 + k_3\theta^7\right) \tag{3.4}$$

where $\theta = \operatorname{atan}(r)$. This model is required as fisheye lenses can expose a field of view $\geq 180°$ which cannot be represented using a rectilinear projection.

The OpenCV library supports both $\Delta_R$ and $\Delta_F$ as well as more sophisticated models for e.g. spherical 360° cameras [32] as employed by the street-view cars or spherical video.

To accommodate for the different calibration models our database therefore not only stores the distortion coefficients $\mathbf{d}$, but the full calibration data to be able to fit a new camera model on demand — without requiring a user to capture new calibration data.

### 3.5.2 Efficient client/server separation

To bring our existing OpenCV based implementation to the Web, we utilize the OpenCV.js bindings, that wrap the C++ code with Emscripten [110] into a WebAssembly library. Here, we do not fully port our existing code to javascript to be executed in the browser. Instead, we introduce a client/server split as the captured 2D measurements, and the final calibration parameters will be transferred to the server anyway. Our architecture is split as follows:

- A web-based acquisition client, that captures video using WebRTC [14] and performs low-level image processing directly on the device. This reduces latency and offloads the computation heavy image processing from the server.
- The calibDB server component that receives the captured key-points and provides new target poses to the clients. This allows re-using most of our control logic and keeps the architecture extendable for multiple clients, as is useful with e.g stereo camera calibration.

Figure 3.10 shows a sequence diagram of the REST based communication between browser and calibDB. As we want to provide our calibration service publicly on the internet we employ API tokens to prevent abuse. After the client was authorized by calibDB, a session ID is returned that is used to track the calibration session and for further authentication. The client then asks for a new target pose which

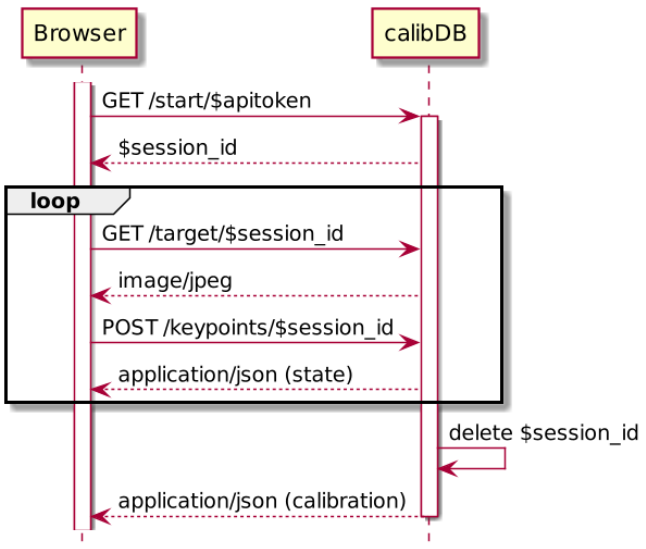


**Figure 3.10:** The REST protocol of our web-based camera calibration system

is returned as a jpeg image that is composited with the video stream using the "color" blend mode. Our underlying method compares the projected pattern images to check whether the user is sufficiently close to the target pose, therefore we can just use the non-black pixels of the overlay image to extract this information. Once the target pose was reached the client sends the acquired 2D keypoint positions to calibDB, which returns a JSON-message [12] containing the calibration results or a state indicating that further measurements are needed.

Our client was tested with Google Chrome and Mozilla Firefox. Here, Chrome is preferable as it also provides the USB-ID of the device, which allows differentiating devices of one series that use different hardware (same name, but different sensor).

### 3.5.3 Calibration database

The service can be queried for calibration data using a combination of *userAgent*, *MediaStreamTrack* and *MediaTrackSettings* [107] as the key:

**Listing 3.1:** Example calibration-data request

```
{
    "camera": "C922␣Pro␣Stream␣Webcam␣(046d:085c)",
    "host": "Linux␣x86_64",
    "image_width": 1280,
    "image_height": 720,
    "zoom": 0
}
```

Here the *camera* property is used for differentiating multiple cameras attached to the PC or the front and back camera on mobile devices. The *host* property is mainly used to differentiate mobile devices where *camera* would only contain "front" or "back". The "zoom" property translates to the currently set focal length of the camera or zero if the focal length cannot be determined.

If no reliable calibration data is available the server responds with the HTTP/307 status code, redirecting to the calibration-guidance landing page as described in Section 3.5.1.

To verify whether calibration data is reliable, we collect at least 5 different calibrations and compute the variance of the intrinsic parameters. Only if the variance is small compared to the parameter values, we consider the calibration data reliable. Here, we aim to enforce re-calibration for interchangeable lens cameras. These identify using the same name, but have largely varying intrinsic properties. Notably, this also covers the use of manually operated lenses where the "zoom" property cannot be read automatically.

If reliable calibration data is available it is returned in JSON encoding as:

**Listing 3.2:** Example calibration-data response

```
{
    "image_width": 1280,
    "image_height": 720,
    "camera_matrix": [[1.43e+03, 0.0, 9.52e+02],
                      [0.0, 1.43e+03, 5.05e+02],
                      [0.0, 0.0, 1.0]],
    "distortion_coefficients": [ ... ],
    "distortion_model": "rectilinear",
    "avg_reprojection_error": 0.72
}
```

The message contains the parameters $\mathbf{K}$ and $\mathbf{d}$ as discussed in Section 3.5.1. Additionally, it provides the resolution at which the calibrated was performed. This is useful when the exact requested resolution is not available. In this case the calibration for closest resolution is returned. The client is now able to either adapt the capturing or redirect to the guidance page, if a specific resolution is crucial.

The client is also able to explicitly specify the desired *distortion_model*, by adding it to the request (Listing 3.1), if only a specific model is supported. In case no calibration using the requested model is available for the specified camera, the server can transparently perform a new parameter fitting on-the-fly. This is made possible by storing the 2D key-points alongside the calibration results. For instance if $\Delta_R$ is requested, but only calibrations for $\Delta_F$ are available, the server can repeat the parameter fitting using the existing data. However, this is not always valid. In the example above the *rectilinear* model is not capable of explaining all measurements as produced by a fisheye lens. Therefore, the response also includes the *avg_reprojection_error*, which is the residual error on the measurements. The client is now again able to redirect to the guidance page to force a more precise calibration.

Our prototype implementation supports the "rectilinear" and "fisheye" distortion models and stores the calibration results as well as the key-points in a schema-less database [66]. This allows to easily extend the system to new distortion models as needed.

### 3.5.4 Extending the WebXR API

To provide the relevant calibration information through the WebXR API, it needs to be extended in several ways. We propose to extend the *XRView* interface, as it already contains the related *projectionMatrix* attribute. To this end, we suggest extending the WebXR matrix notion to 9 element 3x3 matrices to accommodate the $\mathbf{K}$ matrix. Although it duplicates some information, it can be passed to computer vision algorithms without conversion — similarly to how *projectionMatrix* can be directly passed to WebGL. Furthermore, an attribute storing $\mathbf{d}$ and the distortion model must be added.

The distortion model attribute should also be added to *XRRenderState* for allowing applications to request a specific model as discussed in the section above — similarly to how developers request a specific *depthNear*.

This would enable browsers to transparently provide calibration data as provided by our service through the WebXR API. Alternatively browser vendors could opt to bundle a set of calibrations for popular cameras directly with the browser.

However, additional support is be needed to allow matching AR visualization. One possibility is to support image remapping through the WebXR API to allow rectification as shown in Figure 3.9. Alternatively, the WebGL API could be extended to support the reverse direction, namely distorted rendering. For this, actual usage patterns should be analyzed to decide whether this is beneficial or whether it is sufficient to offload these tasks to client libraries like OpenCV.js.

## 3.6 Conclusion

In this chapter, we have presented improvements to state of the art regarding camera calibration in the following areas:

- We have introduced a calibration method, that generates a compact set of calibration frames and is suitable for interactive user guidance. Singular pose configurations are avoided such that capturing about 9 key-frames is sufficient for a precise calibration. This is 30% less than comparable solutions. Calibration precision can be weighted against the required calibration time using the convergence threshold. The camera parameter uncertainty is monitored throughout the processes, ensuring that a given confidence level can be reached repeatedly. Combined with the provided user guidance, this allows even inexperienced users to perform the calibration in less than 2 minutes.
- We have evaluated different user guidance methods for camera calibration. This allows us to give a recommendation that the "virtual mirror" setup is preferable for camera calibration. However, the results of our second survey only hint that using the simplified "quadrille paper" overlay is of advantage. While the user feedback was generally positive, and we measured a slight advantage in the average calibration time, there was a strong variation between the individual participants. Therefore, a larger scale survey is necessary to give a definitive answer here.
- The presented calibration aggregation service allows the general deployment of web-based computer vision algorithms. Previously these would have been limited to systems where WebXR back-ends like ARKit or ARCore were available. The service also guides end-users through the task of calibration, enabling them to use cameras that were not considered by the developers of a particular computer vision algorithm. This property is beneficial for both users and developers of computer vision on the web. We have evaluated the shortcomings of the current WebXR API draft end suggested extensions that can make the whole process transparent for the end-user. Here, it needs to be evaluated whether our calibration key is sufficient to identify the various cameras and devices or if we have to use more sophisticated fingerprinting.

# 4 Synthetic training from CAD geometry

In this chapter we present methods for training a CNN model from abstract CAD geometry. Here, we focus on domain-adaptation. First, we rapidly acquire the true object appearance by extracting a 2D texture map from camera images, which we then use to improve the realism of sythetic renderings. Then we approach the problem more generally, by employing image-conditional GANs to turn the task of bridging the domain gap into a learnable problem.

## 4.1 Introduction

A common belief is that CNN architectures promote the aggregation of information from lower levels, where e.g. edges are detected, to higher levels where those are combined to semantic higher-order arrangements, like houses or windows, which are of high expressiveness. Therefore, the last layer, which is responsible for generating the outputs, receives the most semantically meaningful feature-maps and thus should be be invariant against slight variations in the underlying data.

However, as discussed in Section 2.5.1, current CNN architectures are biased towards the dataset they are trained on, which results in degraded performance when training and testing data exhibit systematic differences. More specifically, Geirhos et al. [31] have shown, that even when using large datesets like ImageNet [22] the resulting CNN models are biased towards texture (See Figure 4.1), which is a rather low-level feature. Still, the models are able to reach top detection performance on these datasets. This means that a large amount of data alone, is not sufficient to force architectures to learn an expressive object representation, which in turn can explain the bias towards the source dataset.

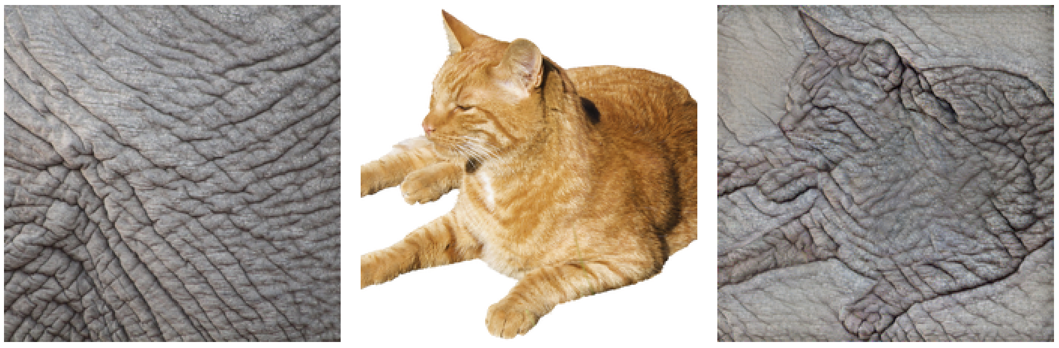


**Figure 4.1:** Predictions of a classifier trained on IMAGENET in presence of cue-conflicts: **(a)** indian elephant (81.4%) **(b)** tabby cat (71.1%) and **(c)** indian elephant (63.9%). Human observers, however, would still classify (c) as a cat due to a bias towards shape. (Figure from [31])

In their analysis, [31] artificially limited the receptive-field of the model, such that even on the higher layers the model "received" only the local pixel neighborhood i.e. the texture. However, the model performance degraded only marginally. This means that even though the dataset consists of more than 14 millions different images of over 20 thousand categories, it seems sufficient to merely learn the texture of the target objects to solve it i.e. to reach a high detection performance.

Furthermore, they performed a user study to find out whether texture is similarly relevant for human observers. To this end, they generated images with cue-conflicts, that is images with the shape of one category and the texture of another (See Figure 4.1). The results showed that, contrary to neural networks, humans are biased towards shape. This is not an issue per se, as neural networks are able to outperform human observers on IMAGENET.

Therefore, they applied data augmentation to remove most of the texture cues from IMAGENET thus forcing the network to focus on the object shape as the main remaining cue. While this slightly degraded performance when doing cross-validation on IMAGENET, it significantly increased performance when applying the learned model to Pascal VOC [25] without any adaptation. This shows that the human bias towards object shape is indeed beneficial for high generalization performance. Returning to our task of training a model on abstract CAD data and generalizing to real images, it leads us to the conclusion that biasing the CNN model towards shape is crucial.

As the industrial objects, that we focus on in this thesis, are intrinsically textureless one might believe that the problem of texture-bias is alleviated. However, there is still a surface structure specific to certain objects, which the model can adapt to. Indeed, we will show in section 4.3 that knowing the true surface properties significantly improves detection rates when compared to a rough color approximation. Furthermore, when only using CAD geometry for synthetic training, not even approximate surface information is available. The parts are typically colored by semantics (e.g. engine, wheel) instead of material appearance (e.g. metal, rubber). Therefore, one cannot rely on surface color or texture. In the context of 3D printing, this is even true for 3D-scanned models, as a manifold of different materials can be used to represent the same CAD geometry. To a certain extent, this also applies to more traditional production lines, where an object runs through multiple processing stages. These can involve polish or coating, which severely alters the object appearance.

The shape information on the other hand, exhibits very high precision. Therefore, it is crucial to prime the network for shape in this scenario, as the shape is the only cue that transfers to real images.

To evaluate the trained CNN models, we focus on the task of object pose estimation as described in Section 2.3, which not only requires correctly detecting and classifying an object in the 2D image, but also involves estimating its 6D-pose. For this, we employ single-stage CNN models as introduced in Section 2.8 for real-time operation. Here, we focus on the domain-adaptation aspect to close the domain gap between training and testing stages. First, we present a straightforward method to *explicitly* capture the true object surface in real time, which in turn improves the realism of the rendered data. In this context, we also show that this is beneficial with classical, contour-based methods. Then, we turn to *implicitly* modeling the domain-gap by employing off-the-shelf style-transfer GAN models.

The chapter is structured as follows: Section 4.2 reviews existing methods for bridging the domain gap, including domain adaptation and domain randomization.

In Section 4.3, we present a method for capturing the *object surface* as a texture-map from image sequences in real-time. The method relies on 6 degree-of-freedom poses and a 3D-model being available. In contrast to previous works this allows interleaving detection and texturing on-the-fly for upgrading the detector. Our evaluation shows that the acquired texture-map significantly improves detection rates using the LineMod [38] detector on RGB images only. Additionally, we use the texture-map to differentiate instances of the same object by surface color.

Section 4.4 formulates the *domain gap* as a style-transfer problem between real and synthetic images. This allows adopting general-purpose GAN models for pixel-level image translation, thus making domain adaptation itself a learn-able task. Here, we consider both supervised and unsupervised training setups and show that our formulation results in a considerable performance improvement, while requiring only little effort to set up when compared to other methods. For evaluation, we focus on training the single-stage YOLO6D [93] object pose estimator on synthetic, CAD-based, geometry. When employing supervised GAN models, we use an edge-based intermediate domain and introduce different mappings to represent the unknown surface properties. Our results show a considerable improvement in model performance when compared to a model trained with the same degree of domain randomization, while requiring only very little additional effort.

## 4.2 Related work

Recent advances with deep convolutional models such as SSD-6D [50], PoseCNN [108] and YOLO6D [93] allow solving the pose estimation problem in real-time. Combined with a successive local refinement method [65, 87], it is now possible to obtain a precise object pose from a single RGB image only.

However, to achieve state-of-the-art performance these models require a large amount of labeled training data. The assembly of such a training-set is an expensive, error-prone and time-consuming process [40], making it cumbersome and often times inapplicable for use with custom applications. In some cases, it is possible to use high-fidelity 3D scans of the target objects to generate training data. The 3D scans not only provide geometry but also surface information. Although scans are reasonably easy to acquire [68], the need to perform a 3D scan for each object does not scale to many objects.

In industrial environments, models created with computer aided design (CAD) software are often available. Here, one can resort to synthetically generated training data by rendering, which allows to create a virtually infinite training set in an automated fashion. However, it was shown that deep CNN models, even when applying cross-validation, tend to over-fit to the specific data-set [95] and show a significantly degraded performance when presented with data from a different domain [27]. Particularly, there is a strong *domain gap* between real and synthesized images, which typically prevents the use of synthetic images for training. In these cases depth-only variants of some algorithms [10, 37] can be used — however at the cost of degraded performance.

To overcome this limitation, existing approaches apply *domain randomization* (DR) to enforce domain invariance by saturating the model with variation [96, 94], requiring the network to learn deeper, more abstract, features that are invariant across domains. An alternative direction is to reduce the gap by employing photo-realistic rendering [100] and structurally correct context generation [73]. Notably, Tremblay et al. [97] apply both for the task of object pose estimation. However, designing a randomization method for a specific domain requires a domain expert to define which parts must stay invariant. Conversely, increasing photo-realism requires an artist to carefully model the specific environments in detail. This in turn increases the cost of generating the data thus negating the primary selling point of using synthetic images in the first place.

When *some* real images are available, transfer learning can be exploited, e.g. by fine-tuning a synthetically trained model with real images [70]. Alternatively, [39] propose to pre-train a network on real data and fine-tune on synthetic data. Furthermore, it is possible to use the real data to enforce domain invariance during training [27]. Recent work [111] extend this approach to guide domain-randomization, thus introducing some benefits of learning-based domain adaptation. However, it is still required to correctly select and design randomization modules.

Recent advances on generative adversarial networks (GANs) [34, 48, 13] have shown great improvements regarding image quality and plausibility, training stability and variation of output. Of particular interest in the task of closing the domain gap are conditional GANs [45]. These are networks that, unlike traditional GANs, pass additional input to both, the generator and the discriminator, to condition the generator output. Here, the image-conditional GANs form a general-purpose framework for image-to-image translation problems, like semantic segmentation, colorization and other style transfer tasks. Existing solutions can be split into paired models [45, 103] and unpaired models [114]. The former are trained to adapt source to target images, paired in a supervised fashion, while the latter do not require supervision and instead directly learn to transfer the distribution of image features between two unstructured data-sets.

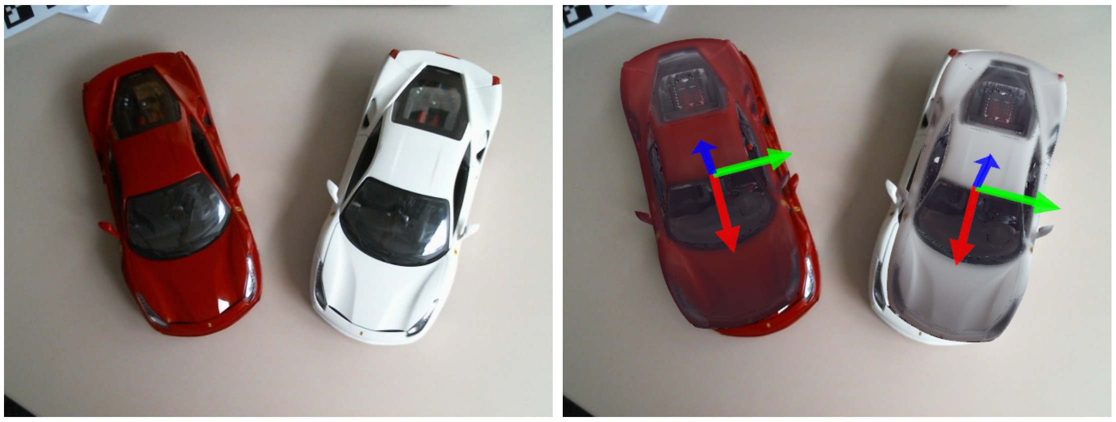


**Figure 4.2:** Our extension of LineMod [38] is able to correctly detect multiple instances of the same object based on surface color. The corresponding 6D poses are visualized using the textured mesh. (see supplemental material at `https://youtu.be/IB19rTXUOt8`)

## 4.3 Real-time texturing for domain adaptation

This section focuses on the real-time acquisition of surface texture data and dynamic detection-model augmentation. This allows capturing and using the surface color information on-the-fly.

In the context of real-time surface reconstruction there is notably the work by Whelan et al. [105], who extend the KinectFusion [68] algorithm to colors. For this, an additional 3D color volume of the same size as the geometry voxel grid is used. This means that the color resolution is tied to the geometry resolution and the memory consumption has cubic complexity. Furthermore, RGB-D data is required. The mentioned 3D scans are typically acquired by variations of [68] and consequently store surface information as vertex colors. Here, [113] improve surface resolution by subdividing the scanned geometry. This is similar to using texture maps but results in inhomogeneous sampling and inefficient storage.

On the other hand, in the context of 3D scanning [16], 2D textures are often used which only have quadratic storage complexity and allow decoupling surface resolution from geometric resolution. More specifically, structure-from-motion based methods [101] only require RGB images to reconstruct both color and geometry.

However, these methods operate on a-priori recorded data-sets which prevents real-time operation. Notably, the global optimization step alone, as employed by those methods, takes up to several hours. Our method in contrast operates in real-time while only requiring RGB frames to incrementally generate a 2D texture. To this end, we assume all geometry as fixed and given and only optimize locally for color consistency. The closest method to our work is by Magnenat et al. [63], who also map a 2D camera image to texture in real-time. However, their work specifically only addresses a single view and focuses on the in-painting aspect.

To employ our texturing method for object detection, we build upon the LineMod detection framework by [38]. They employ a two-stage, handcrafted feature descriptor, specifically tuned for texture-less object detection. In the first step gradient templates (DOT), which capture the contour of an object, are matched to the input image in a sliding window fashion. In a successive outlier-rejection step, surface color is used to filter implausible matches, based on the interior color.

Even though this no longer provides state-of-the-art detection performance [93, 50], the internal separation allows computing the DOT features on CAD geometry only and add the surface color at run-time. This is generally not possible with deep learning based approaches, which rely on surface color being available during training. Efforts to train on an abstract representation [74] to allow for different object appearances, typically result in a degraded performance compared to training on real images. In contrast, our extension of [38] improves its performance, while allowing to differentiate several instances of the same geometric object by their surface properties.

Based on the above, our key contributions are;

1. an incremental, real-time texture-map extraction pipeline and
2. efficient integration of texture-maps for object instance recognition.

This section is structured as follows: in Sections 4.3.1-4.3.4 we present our texture extraction algorithm in detail. Section 4.3.5 then describes the application of the texture-maps for object instance recognition, while in Section 4.3.6 the use of texture-maps for object detection is evaluated using the LineMod dataset [38].

### 4.3.1 Texture extraction

In rendering, the process of "texture mapping" consists of the following two steps

1. creating a mapping from a texture to the surface of a 3D model and
2. projecting the model and simultaneously mapping the texture into a 2D image.

The first step is also called "texture atlas creation" and is typically performed by an artist during mesh creation. In contrast, our focus lies on the reverse direction, namely mapping from a 2D image of a projected 3D model back to the surface image as specified by the texture atlas (see Figure 4.3).

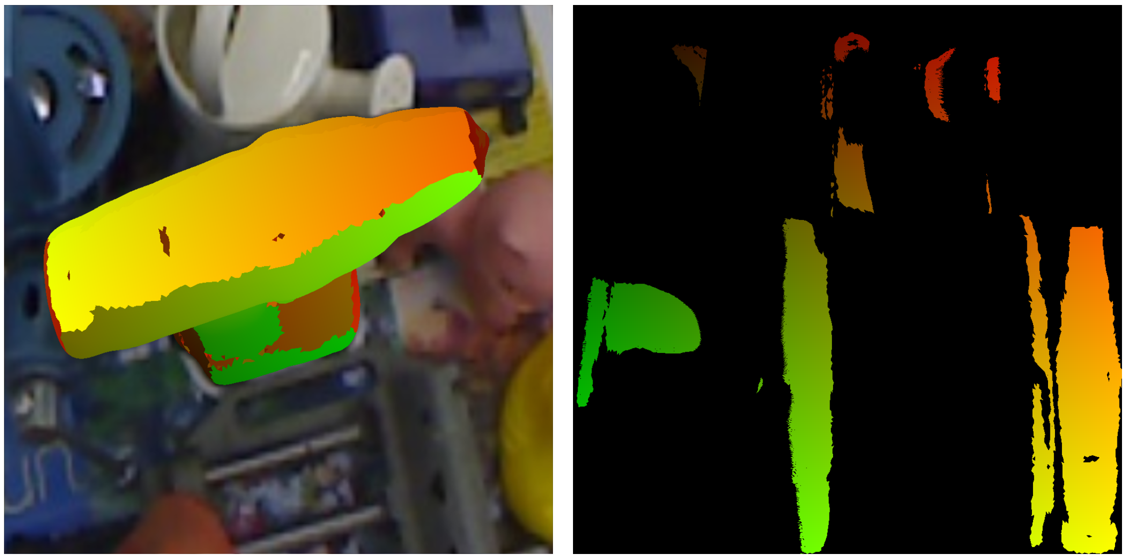

(a) Image space (b) Texture space

**Figure 4.3:** We are mapping from image to texture space, which is the reverse direction compared to rendering. Texture coordinates are encoded as red-green.

Generally texture atlas coordinates are not included in CAD data and therefore have to be generated. However, automatic texture atlas generation is still an active area of research [57] and outside the scope of this work. Here, we just use the angle-based "Smart UV Project" algorithm implemented in the Blender toolset (v2.79b[1]) to generate the texture atlas and instead focus on the second step of texture-mapping.

In the remainder of this section we first discuss a simple exposure normalization scheme, before we present our texture extraction method in detail and finally turn to merging multiple views into one texture. The full pipeline is illustrated in Figure 4.4.

## 4.3.2 Exposure normalization

As our method does not explicitly compensate for different exposure times and lighting conditions we preprocess the image stream to homogenize the brightness. For this we use the first captured frame as reference and modify the successive frames to match its brightness and contrast levels.

Here we follow the idea of Reinhard et al. [77] of adapting an input image $\mathbf{I}$ to match a reference image as

$$\mathbf{I}_n = \frac{\sigma_{ref}}{\sigma_I} \cdot (\mathbf{I} - \mu_I) + \mu_{ref} \tag{4.1}$$

[1]`https://www.blender.org/download/releases/2-79/`

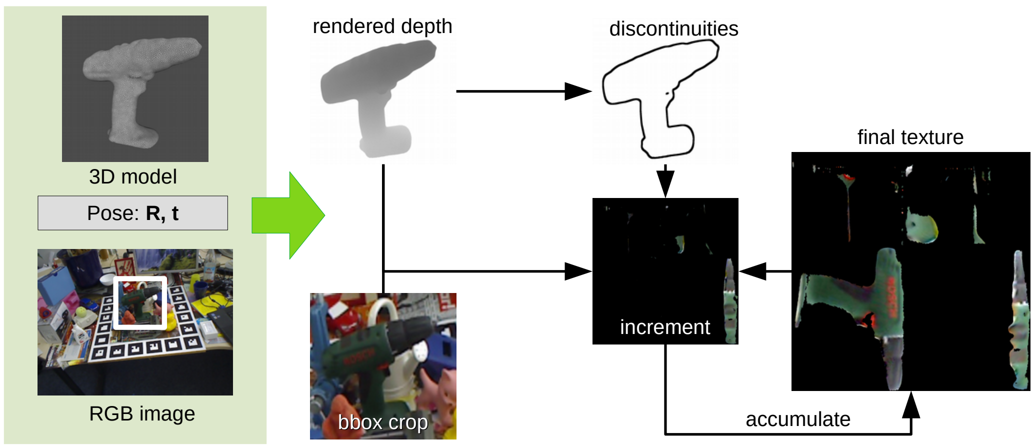


**Figure 4.4:** Our texture extraction pipeline. Given a 3D model and its pose in a RGB frame, we first render the depth to determine visibility. Image regions around depth discontinuities are discarded as they are unreliable. Next a texture-increment is extracted and a per-pixel score is computed to decide whether to merge the visible pixels into the final texture. Only the following buffers are required on the GPU; "final texture", "increment" and "discontinuities".

where $\mu_I, \sigma_I$ and $\mu_{ref}, \sigma_{ref}$ are the mean and variance of the input image and the reference image, respectively.

However, whereas [77] apply the transfer for all channels in the Lab color space, we only apply it to the luma component Y in the YUV color space as we explicitly want to preserve the chrominance information.

This step is omitted if the exposure can be fixed during capturing.

## 4.3.3 Texture-space to image-space mapping

Texture mapping can be formalized as follows: given a triangulated mesh, each vertex $\mathbf{v}_i = [X, Y, Z, 1]$ with an associated texture coordinate $\mathbf{t}_i = [u, v]$ is projected into the current view by a world-to-image transform $\mathbf{P}$ as $\mathbf{p}_i = \mathbf{P} \cdot \mathbf{v}_i$. Here $\mathbf{p} = [x, y, 1]$ is a normalized pixel location in the image $\mathbf{I}$.

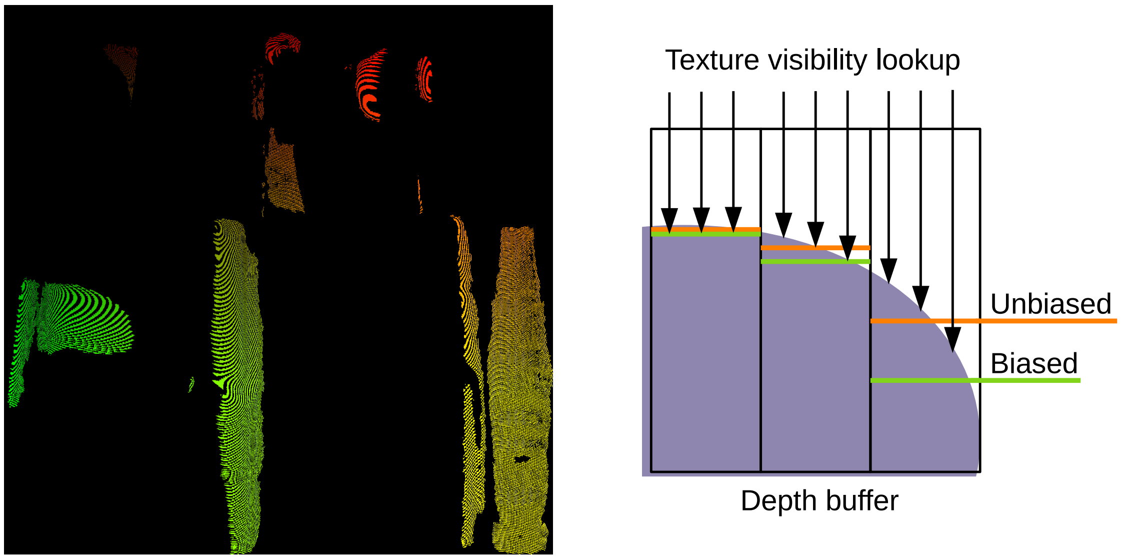
On the interior of the triangle formed by $(\mathbf{t}_i, \mathbf{t}_j, \mathbf{t}_k)$, a texture coordinate $\hat{\mathbf{t}}$ is interpolated and used for lookup in texture $\mathbf{T}$ as

$$\mathbf{I}(\mathbf{p}) = \mathbf{T}(\hat{\mathbf{t}}). \tag{4.2}$$

This mapping is continuous in texture space and therefore allows for bi-linear interpolation to avoid aliasing artifacts.

For texture extraction however we are interested in the reverse mapping, namely

$$\mathbf{T}(\mathbf{t}) = \mathbf{I}(\hat{\mathbf{p}}). \tag{4.3}$$



**(a)** Depth test aliasing **(b)** Slope biased depth

**Figure 4.5:** We store slope-scaled biased depth values to avoid aliasing errors during the visibility test.

Instead of iterating over the mesh topology as defined by $\mathbf{v}_i$ in 3D, we now iterate over $\mathbf{t}_i$ as defined by the texture-atlas in 2D. Conversely, we now require a continuous value of $\hat{\mathbf{p}}$ in image space for lookup. This is computed by interpolating in the triangle formed by $(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k)$, of which each point is obtained as above by $\mathbf{p}_i = \mathbf{P} \cdot \mathbf{v}_i$.

Here, visibility must be explicitly computed; with equation (4.2), we implicitly assumed overlapping points to be resolved by a depth-test, only retaining the points closest to the camera. This can no longer be exploited, as points do not overlap in the texture space.

To handle visibility we therefore introduce an additional depth buffer and render depth from the camera view. This allows comparing the depth of an interpolated coordinate $\hat{\mathbf{p}}$ to the actually visible depth value. However, this leads to aliasing; with non-planar objects the view resolution cannot be adapted to match the texture space resolution.

To remedy the aliasing artifacts we apply techniques from the shadow mapping domain [8] where the same problem occurs when a scene is rendered from a shadow camera and an observer camera view. Particularly, we

1. focus the camera on the object bounding box to increase the sampling rate in image space and

2. apply a slope-scale depth-bias to account for the remaining differences in sampling rates during visibility testing.

The latter is especially important; as the texture atlas has a higher sampling rate than the depth buffer, several points $\hat{\mathbf{p}}$ , interpolated in the texture space, map to the same point $\mathbf{p}$ in the image depth-buffer. At steep angles $\hat{\mathbf{p}}$ has a strong depth variation and thus neighboring points alternatively fail and pass the visibility test when compared to a single reference value (see Figure 4.5b).

To account for this we store a biased depth that allows for a sampling offset of 1px in image space. The bias $b$ depends on the depth slope $dz$ per pixel $dx$ and the minimal depth buffer resolution $r$ as:

$$b = \frac{dz}{dx} + r. \tag{4.4}$$

The bias is large in steep regions while minimal for faces parallel to the camera. This computation can be implemented efficiently on the GPU by using e.g. *glPolygonOffset*. The effect can be observed by comparing Figure 4.5a and Figure 4.3b. This allows us to map each visible pixel from a single image into the texture to record the object surface. The resulting reconstruction can already be applied for detecting the object in similar views (see Section 4.3.5).

### 4.3.4 Merging multiple views

Generally, the object surface is only partially visible from a single view and therefore multiple images are needed to reconstruct the full texture.

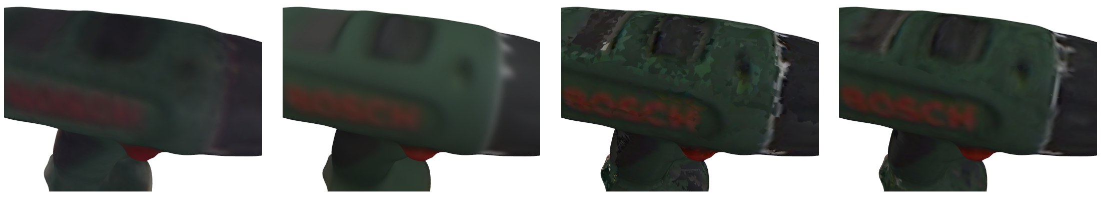
Assuming that the same texture point $\mathbf{t}$ will be observed in different images as $\mathbf{c_0}, \dots, \mathbf{c_N}$ where $\mathbf{c_i} = \mathbf{I}_i(\hat{\mathbf{p}}_i)$, we discard edge-pixels at object boundaries or strong depth discontinuities. These measurements are unreliable as they might come from different surfaces due to pose imprecisions and limited camera resolution. Instead, we aim for a view where $\mathbf{t}$ is not at an observed edge. A pixel is considered to be part of an edge if the depth change is larger than 10% of the object diameter. All points in a 5px neighborhood of an edge-pixel are discarded as well (see Figure 4.7a).

To combine multiple valid observations $\mathbf{c_i}$ of $\mathbf{t}$, we define score $s$ that, inspired by [16], weighs each observation by the distance to camera $d$ and the angle $\alpha$ between surface normal and view direction as

$$s = \cos\alpha \cdot (1 - d), \tag{4.5}$$

where $d$ is assumed in normalized device coordinates ranged $[0; 1]$ and $\alpha$ is computed based on the interpolated surface normal, which can be defined per vertex $\mathbf{v}_i$ (e.g. for a sphere) and therefore is not required to be constant for a single face.

Using $s$ we implemented two merging strategies; a weighted arithmetic mean



**(a)** Vertex colors **(b)** Weighted mean as in Eq. (4.6) **(c)** Best score as in Eq. (4.7) **(d)** Best score with blending

**Figure 4.6:** Exemplary surface color reconstructions of the "Driller" object Texture merging strategies using (a) KinectFusion and (b, c, d) variations of our algorithm.

$$\mathbf{t} = \frac{s_0 \cdot \mathbf{c_0} + \ldots + s_n \cdot \mathbf{c_n}}{s_0 + \ldots + s_n}, \tag{4.6}$$

and only retaining the best view

$$\mathbf{t} = \arg\max_{\mathbf{c_i}} \left\{ s_0, \ldots, s_n \right\}. \tag{4.7}$$

Both equations can be efficiently implemented on the GPU using a single RGBA buffer for accumulation, as $RGBA = [\mathbf{c}, s]$.

Figure 4.6 shows exemplary results. Eq. (4.6) produces a smooth surface, while retaining more detail than vertex coloring. However, the averaging over slightly inaccurate object poses results in a loss of fine detail when compared to Eq. (4.7).

Using Eq. (4.7) on the other hand retains all details, but emphasizes inconsistencies in exposure or object pose as seams between neighboring increment texture-patches.

To alleviate this problem we blend increment-patches at their boundaries into the existing texture during accumulation. Instead of simply overwriting the texture content with the new maximum, we compute the distance transform to the patch boundaries over a 5x5px support using the L2 norm. Using the distance we then linearly interpolate between the old and the new color value $\mathbf{c}$ and pixel score $s$.

Figure 4.7b shows an increment-patch for Figure 4.7a, projected onto the object. Note the gradient at the edges, which is linear in texture space.

The blending not only produces visually more pleasing results (compare Figures 4.6c and 4.6d), but is crucial for computing the LineMod descriptor which relies on local gradient orientation.

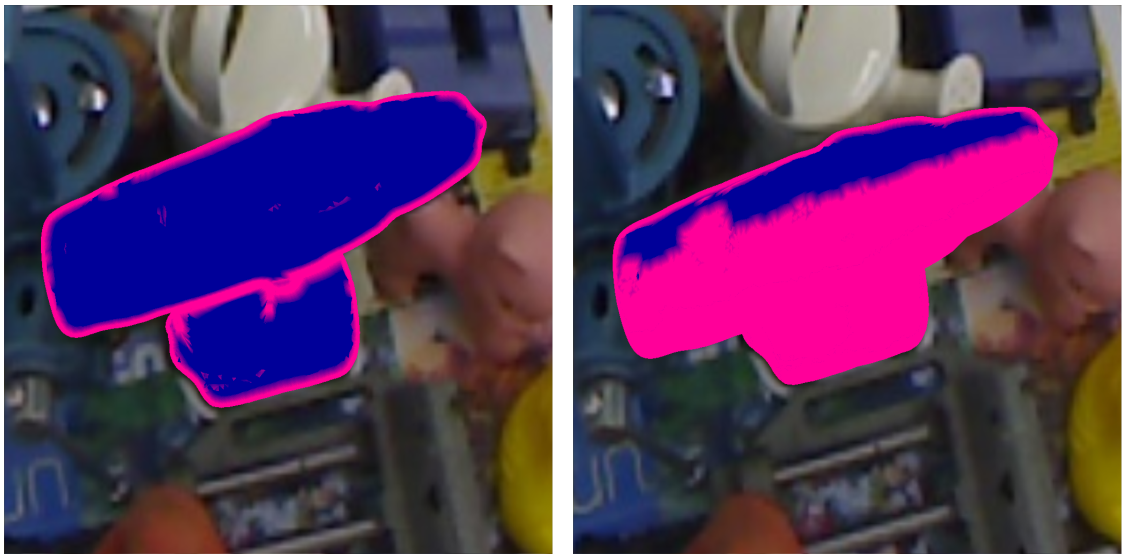

(a) Initial view (b) argmax-based update with blending

**Figure 4.7:** Merge-maps of two successive frames when using eq. (4.7). Valid pixels are colored blue.

### 4.3.5 Object instance detection

In this section we describe how to employ the extracted textures for object instance detection i.e. differentiating multiple instances of the same object. Here, we extend the color-based outlier rejection of [38] to multiple color hypotheses to simultaneously perform classification.

The idea of color-based outlier rejection in [38] is to store the expected color of the object projection alongside the LineMod template and at run-time count how many pixels in the camera frame have the expected color.

To make the check robust against lighting variations, they convert the images to the HSV colour space and compare only the hue component. However, hue does not cover the colors black ($V = 0$) and white ($V = 1, S = 0$). Therefore, these are mapped to blue and yellow respectively, which completes the color-based descriptor (see Figure 4.8a).

To extend this scheme for object instance detection as well as for on-the-fly recorded textures, we separate the expected color from the expected surface visibility. To this end, we store the texture coordinates of the object projection (compare Figure 4.3a) instead of storing the expected color directly. The template surface-texture is stored separately. At runtime, we now use the texture coordinates to perform a lookup into the template-texture to retrieve the expected color, which gives us the same information as in [38].

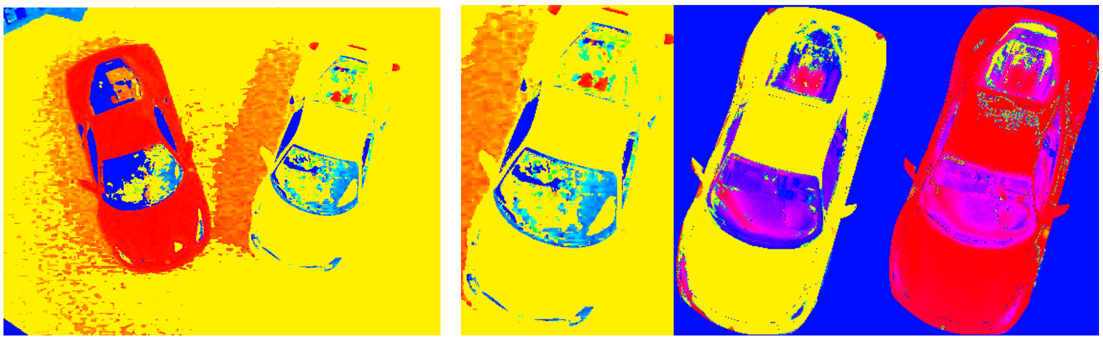

(a) input image (b) candidate / white template / red template

**Figure 4.8:** Hue-based instance detection. The input image (Figure 4.2) is cropped based on the template bounding box and compared to a set of hue templates.

However, it is now possible to easily swap the surface-texture to globally change the expected colors. Here a live-reconstructed texture can provide more accurate template colors and notably multiple template-textures can be used for object instance detection (see Figure 4.8b).

Finally, the outlier rejection scheme needs a slight modification for classification. Instead of returning the first inlier based on the expected color, it needs to allow multiple matches without repetition. For this, after finding an inlier, only the corresponding texture-template is removed and the remaining candidates are checked until all template-textures are found or all candidates are rejected.

While this is an integral part of the LineMod pipeline, it can be optionally integrated as a post-processing step to a CNN-based architecture that is capable of abstracting the object appearance to some degree. E.g. it can be executed after non-maximum-suppression in [93] to compute agreement with the color template.

### 4.3.6 Evaluation

The presented method is evaluated in the context of object detection. To this end we train the Line2D variant of the LineMod detector on the corresponding dataset [38]. The dataset does not contain views specifically intended for surface reconstruction and thus represents reconstruction during detection well. We use the publicly available LineMod implementation of OpenCV.

There are 15 sequences for different objects, consisting of RGB-D frames with ground-truth poses and recorded at distances of 65cm-115cm. We select a subset of 8 objects for which a 3D mesh is available and that are large enough to provide a reasonable texture resolution. The meshes included in the dataset were recorded using a variation of KinectFusion [68] and thus encode surface information as vertex-colors.

We apply our texturing algorithm on each sequence using the ground-truth poses to merely simulate a tracking algorithm for better reproducibility. Then we train LineMod on synthetic renderings using the generated textures as well as included vertex-colors as a baseline. We parametrize training and testing as in [38], particularly;

- We use 89 views on the upper hemisphere around the object, derived by subdividing an icosahedron recursively twice.
- For each view there are 7 in-plane rotations with roll angles between $-45°$ and $45°$.
- Furthermore 6 distances, with 10cm increments, between 65cm and 115cm are used.
- During color-based outlier rejection we discard candidates where less than 70% of the pixels have the expected color. The threshold on per-pixel hue difference is set to $54°$.

This results in a total of 3738 templates per object for training. However, in contrast to [38], we are only using RGB data without depth — therefore we do not restrict the color gradient features to the contour, but compute them on the interior as well.

For testing, we measure the true positive rate on the sequences. As in [37] we consider an object successfully detected when it is within a fixed radius $r$ around the ground truth position. We globally set $r = 11\text{cm}$ in our experiments to allow for depth mis-classification by one step.

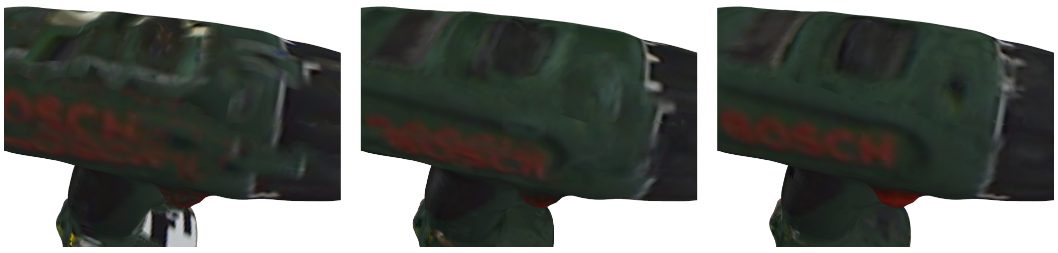
In order to keep interactive performance, we only consider the first 30 LineMod candidates for matching and outlier rejection.

To simulate the CAD data use-case without any surface information available, we additionally perform training using a white diffuse material for all objects. For generating gradient features on the interior of the object, we use ambient occlusion (AO) [4] as a lighting approximation. Ambient occlusion is a purely geometrical method that is independent of actual light and surface properties. We skip the outlier-rejection step as no color information is available.

Table 4.1 shows the true positive rates for the variants mentioned above — as can be seen the texture-based variants outperform the vertex-color baseline of [38] by a margin of 10% on average. However, there are strong variations between

| Object | AO | vertexcolor | texture (4.7) | texture (4.6) |
|---|---|---|---|---|
| benchvise | 0.54 | 0.75 | **0.82** | 0.82 |
| driller | 0.15 | 0.43 | **0.63** | 0.54 |
| iron | 0.53 | **0.71** | 0.67 | 0.68 |
| can | 0.38 | 0.67 | 0.78 | **0.83** |
| glue | 0.07 | **0.21** | 0.17 | 0.17 |
| cam | 0.1 | 0.28 | **0.62** | 0.55 |
| eggbox | 0.47 | 0.6 | **0.79** | 0.79 |
| holepuncher | 0.2 | 0.62 | 0.59 | **0.65** |
| average | 0.3 | 0.53 | **0.64** | 0.63 |

**Table 4.1:** True positive rates on the LineMod dataset with different training data. The ambient occlusion (AO) variant does not include any outlier rejection.



**(a)** LineMod [38] **(b)** "Single shot pose" [93] **(c)** Ground truth

**Figure 4.9:** Qualitative results for on-the-fly surface color reconstructions of the "driller" object in relation to different pose detection methods.

the individual objects, therefore it remains inconclusive whether variant (4.6) or variant (4.7) of our algorithm is preferable.

Notably the AO variant cannot reach the performance of the other methods. With some objects where it even becomes unusable (e.g. driller, cam). This emphasizes the need of surface information for object detection.

**Using noisy pose data** To evaluate the applicability of our method for on-the-fly texturing with noisy pose data, we additionally used the state-of-the-art "single shot pose" (SSP) detector [93] instead of relying on ground-truth poses.

Figure 4.9 shows qualitative results of texturing using ground-truth, SSP and LineMod poses. While the LineMod results only allow for a rough color-based outlier rejection, the results using SSP poses are very similar to using the ground truth. To further quantify this, we repeated the training of SSP using synthetic renderings of the "driller" object instead of using cross-validation as in the original paper. At this, we measured the true positive rate (TPR) using the 5cm, 5deg metric. Here, training with textured renderings (Fig. 4.9c) resulted in a TPR of 0.37. Using the imperfectly textured objects (Fig. 4.9b) resulted in a TPR of 0.34, which supports the qualitative impression. When training with vertex colored renderings only, the performance was significantly degraded, resulting in a TPR of 0.14.

**Speed** The evaluation was performed on a notebook with an Intel i7-7700HQ CPU at 2.80GHz and an Intel HD 630 iGPU. The average time to accumulate one video frame into a 1024x1024 px sized texture is 2.69 ms. This allows running the texturing algorithm in parallel to tracking to reconstruct a texture on-the-fly.

The average time to perform a texture lookup as described in Section 4.3.5 is 0.82 ms using the software remap implementation in OpenCV. This step can be therefore applied generally without requiring GPU usage.

**Multi instance detection** For the multi-instance detection we performed a qualitative analysis using a separate sequence where two toy cars are alternately and simultaneously visible. The surface colors are white and red which are adjacent in HSV space (white is mapped to yellow as described in section 4.3.5). Furthermore, the surface exhibits specular reflection which is not filtered during texturing.

Nevertheless, our method was able to robustly discriminate both objects (see Figure 4.2).

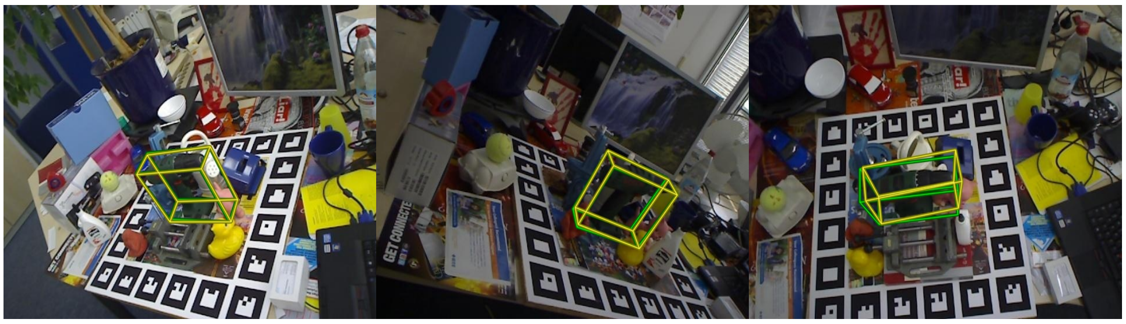


**Figure 4.10:** Example of YOLO6D [93] on the LineMod data-set when trained *solely* on synthetic images. The green bounding box represents the ground truth pose and the yellow one the predicted pose.

## 4.4 Style-transfer GANs for bridging the domain gap

This section focuses on employing conditional GAN models to formulate the domain gap between real and synthetic images as a learning problem. At this, we propose training pipelines incorporating both paired and unpaired style-transfer and evaluate the results on the task of object pose estimation — a particularly challenging scenario which requires a high fidelity of the object contours, which was, to the best of our knowledge, not previously addressed with GAN based domain adaptation. In the context of paired style-transfer, we propose the use of the intermediate edge domain to do away with the need of real images for supervised training. Here, we evaluate different representation for mapping CAD geometry with unknown surface properties into the edge domain.

Most closely related to our work is [2] which employ GANs for data augmentation. In contrast, we use GANs to specifically address the domain gap and employ synthetic data generation instead of data augmentation. [74] introduce the "pencil filter" as a domain with reduced expressiveness to tackle the domain gap and train a pose estimation network in this domain. However, the pose estimation network takes a strong performance hit as the "pencil domain" does not retain enough relevant features. We are able to avoid this hit by learning a mapping from the reduced domain to real images to reconstruct appropriate features. In the medical domain, [64] employ a GAN architecture for domain adaptation. However, they only consider the use of an unsupervised GAN model for reverse domain adaptation by making real images more synthetic, while we consider both paired and unpaired architectures and also consider the forward domain adaptation in the unsupervised case.

Based on the above, our key contributions are;

1. formulating the domain gap as a learning problem using off-the-shelf image-conditional GANs,
2. introduction of the intermediate edge domain for training paired translation networks purely from synthetic data and
3. evaluation of paired and unpaired models regarding pose estimation performance.

This Section is structured as follows: in Section 4.4.1 the general approach is introduced, Section 4.4.2 defines baseline methods and Section 4.4.3 discusses the choice of suitable GAN models. We then continue to describe our training pipeline employing an intermediate domain for paired domain translation in Section 4.4.4 and unpaired direct domain translation in Section 4.4.5. In Section 4.4.6 the method is evaluated in terms of pose detection performance of the YOLO6D [93] model on the LineMod [38] data-set.

### 4.4.1 Approach

The core idea of our approach is to formulate the domain gap as a learning problem that is addressed with generative CNN models. Here, we use the generative adversarial framework to train a conditional generator that is able to augment images such that the pose estimation network becomes invariant to the source domain. For this, the statistical distribution of image features found in both the real world and the synthetic domain must be matched, allowing the alignment of one domain to the other.

In this section we first discuss applicable GAN models and show qualitative results on the LineMod data-set to motivate the choice of specific image-conditional GAN models. Then, we present our pipeline for fully synthetic training based on supervised image translation, leveraging Pix2PixHD [103]. Next, we turn to unsupervised image translation and introduce an alternative pipeline, that replaces the GAN model with CycleGAN [114], which simplifies data acquisition by lifting the requirement of pairing images from both domains.

### 4.4.2 Baseline methods

We define two baseline methods for synthetic training. Both follow the data augmentation scheme of [93] by using random backgrounds from a set of real images, apply random image scaling and randomly adjust exposure and saturation adjustment. However, instead of using crops from real images, we render the object on top of the background (See Figure 4.11) with the following methods:

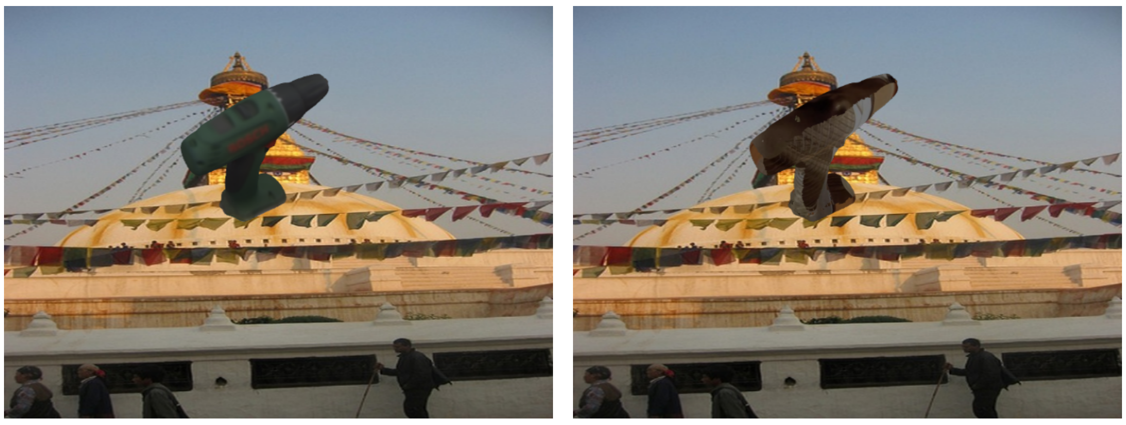

**(a)** real texture **(b)** random texture

**Figure 4.11:** Baseline data augmentation schemes for object "driller" using *(a)* realistic object texturing and *(b)* randomly selected image for object texturing

a) Realistic texturing, by applying the true texture extracted from the data-set (see Section 4.3) and

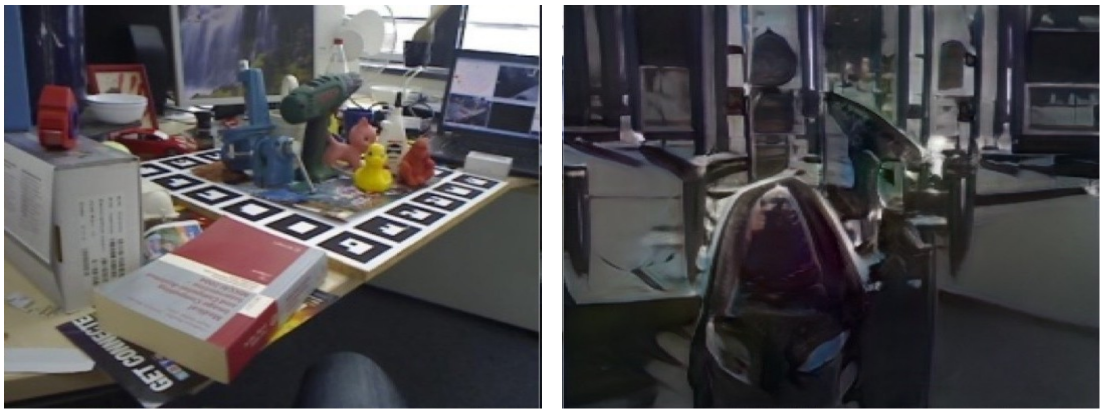
b) randomly texturing the object during rendering.

The first option depends on having the surface properties of the physical object available, but allows generating an arbitrary amount of realistic data by rendering. The second scheme applies blind DR by randomizing both the object and the background appearance. Note that this scheme neither requires plausible object placement nor plausible object appearance and thus is far easier to set up, compared to other DR solutions.

### 4.4.3 Suitable GAN models

The main requirement on the generator is that the resulting images have a high correlation with a given pose, such that the pose estimation model can be trained in a supervised fashion. In theory any GAN model can be used for this if images can be mapped into the latent space of the generator. Such a mapping allows to condition the generator to create images, that resemble the input inside the *latent space*.

If the latent space is constructed in a way that allows for interpolation by e.g. employing Kullback-Leibler loss [23], this approach would also allow synthesizing novel samples not seen during training. In the case of pose estimation, this is required to generate images for views not seen by the adaptation network during training. However, the resulting image must retain enough fidelity for the pose estimation network to predict the correct pose. Not all GAN architectures are



**(a)** input **(b)** sample

**Figure 4.12:** Exemplary results of using StyleGAN for conditional sampling. Here, the input image is mapped into the latent space of the generator and is used as the mean for sampling.

practical for this use case. For a preliminary experiment, we use the StyleGAN [49] model, which offers state-of-the art generation performance and allows interpolation in the latent space of the generator.

Here, we map a real image into the latent space and use it as the mean for the random vector to sample new images, which would ideally retain most of the original image content; particularly the pose of the target object. Figure 4.12 shows an exemplary result; while the images seem plausible (considering the reduced training time), it is obvious that the model is not able to sufficiently capture the locality of the object. This results in a significantly different pose in the sampled image, compared to the input image. This precludes this approach in being used for pose estimation. Therefore, we focus on conditional GAN models in the following.

For this experiment, StyleGAN was trained at a reduced resolution of $256 \times 256$ px for three days using approximately 110.000 renderings composed on top of real backgrounds. The model was initialized using the weights for the LSUN Cat data-set [109] of the original publication. This allowed operating with the reduced training period, compared to training the model from scratch which required 13 days according to the original publication.

### 4.4.4 Paired intermediate domain translation

In this section we introduce a training pipeline based on the Pix2PixHD [103] paired image translation model. This is a supervised approach, which requires aligned image pairs to learn the domain translation. Pairing synthetic images with real images requires an according training set of real images, which defies the goal

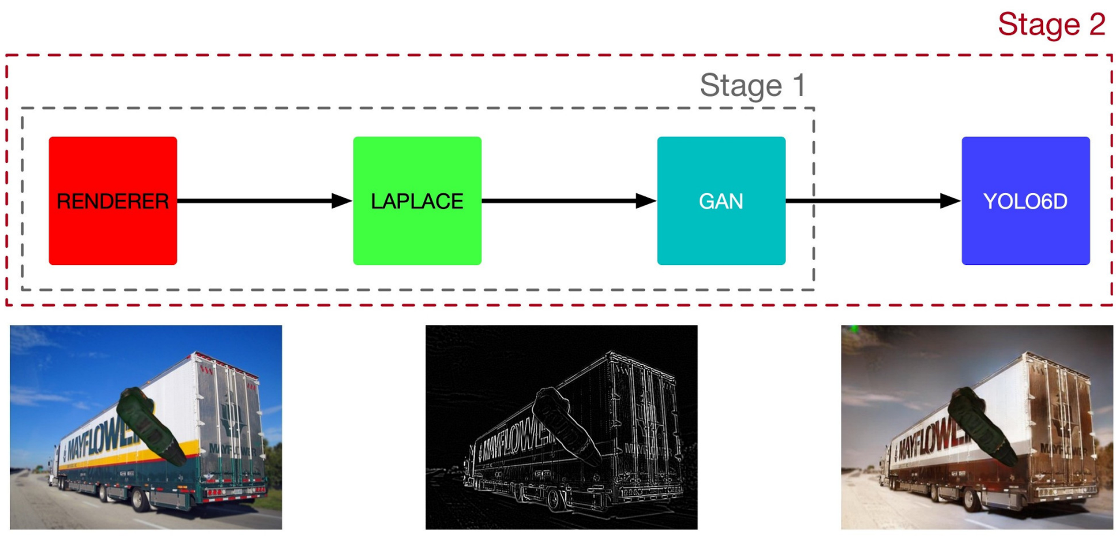
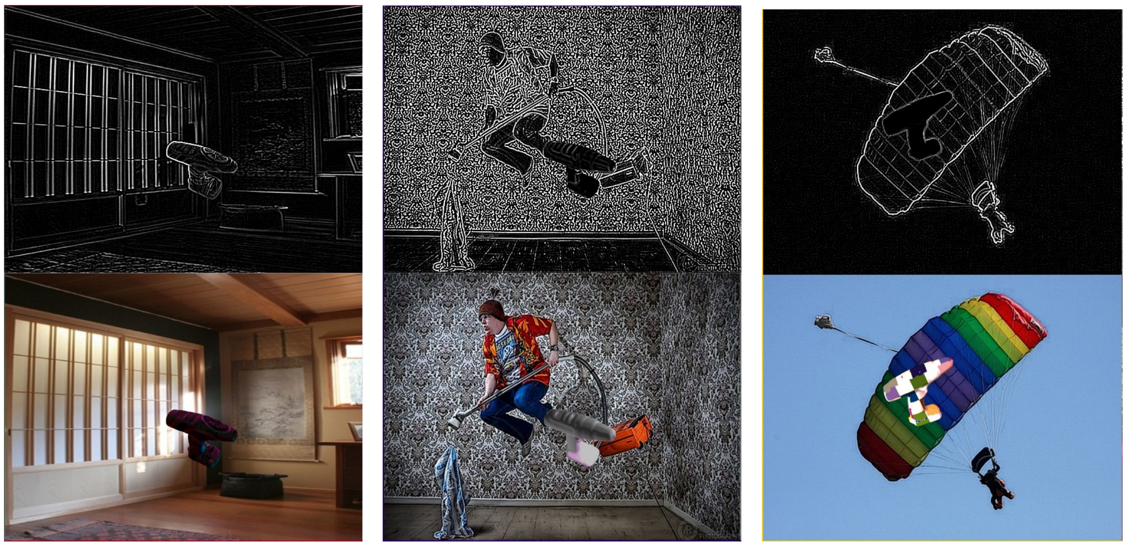


**Figure 4.13:** Our synthetic training pipeline using the laplace filtered intermediate domain and incorporating the Pix2PixHD GAN for domain adaptation as presented in Section 4.4.4.

of synthetic training. Therefore, we introduce a deterministic transform into an intermediate domain with reduced expressiveness, making real-world and rendered images less distinguishable. Here, we use the Laplace filter, which approximates the second order image derivative that is continuous and directly translates to edge strength without requiring a thinning step. The Pix2PixHD model is then employed to reconstruct real images from their Laplace filtered variants.

The pipeline now consists of two trainable models; the Pix2PixHD model for domain adaptation followed by the pose-estimation task network (see Figure 4.13). As the performance of the second model depends on the first model, the pipeline has to be trained in two stages. First, the domain adaptation network is trained until convergence on rendered images with random backgrounds and their Laplace-filtered variants. Here, the network must simultaneously reconstruct the rendering as well as the real background which forces the network towards realistic reconstructions. Next, the pose estimation model is trained on the reconstructed images. Assuming the adaptation network was able to generalize from the training data, we now can create a virtually infinite amount of realistic views.

The remaining question is how to model the object surface before converting it to the Laplace domain, given that we do not want to impose any restrictions on possible object appearances. Here, one needs to balance the learning problem between the domain adaptation and the pose estimation network — e.g. enforcing discriminative features makes the problem for the adaptation network more challenging, yet it reduces the difficulty for the pose estimation network. Specifically, we opted for the following methods covering different work distributions of the involved networks:



**(a)** random texture **(b)** solid gray **(c)** checkerboard

**Figure 4.14:** Examples of the reconstruction tasks for Pix2PixHD with different rendering methods. *Top row:* source image in the Laplace domain *Bottom row:* target images to be reconstructed.

1. Use the real world texture (see Figure 4.13). This should result in the best model performance as it is the easiest task. However, to obtain the texture real images are needed, which dissents with our goal of fully synthetic training. This option was therefore mainly included to assess the performance loss of mapping into the Laplace domain as well as the loss induced by the following options. It corresponds to baseline variant a) with domain adaptation.

2. Use a random texture (see Figure 4.14a). This prevents the networks from learning any surface related information of the object to be detected. While this should not affect the domain adaptation network which is already faced with the task of reconstructing arbitrary background images, it makes the task of pose estimation significantly more challenging. The important shading and contour cues can be arbitrarily degraded by the used texture. This corresponds to baseline variant b) with domain adaptation.

3. Use a uniform color for the object (see Figure 4.14b). Instead of using a random texture, we assume the object to be of one uniform, yet arbitrary color. Given the Laplace intermediate domain, the adaptation network cannot learn a correct object colorization. Therefore, we set the target color to gray, which is the most likely guess the adaptation network can take, given this task. Keeping the surface properties fixed allows to apply a consistent shading

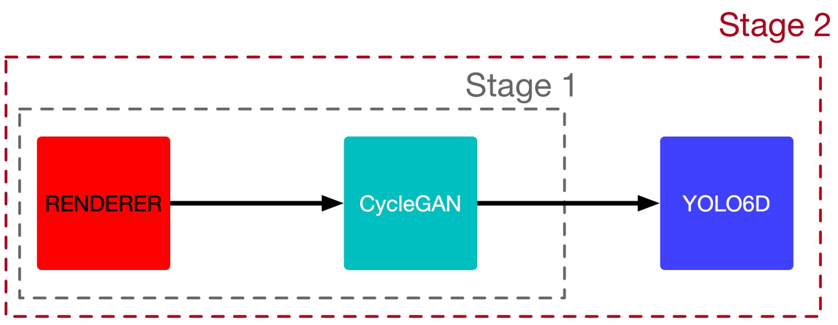


**Figure 4.15:** Our synthetic training pipeline using direct domain adaptation as presented in Section 4.4.5

to the object, which in turn can be exploited by the pose estimation network. However, the reconstruction of a plausible surface shading in turn makes the task of the domain adaptation network more challenging, but is a deliberate choice for balancing the work. To prevent the adaptation network to over-fit to a specific lighting position, we place several point lights randomly around the object.

4. Use a fixed checkerboard pattern for the object surface (see Figure 4.14c). Here, the Laplace images are generated from a uniformly colored object as above, while the reconstruction target is rendered with a fixed checkerboard texture. This makes the pose estimation task easier as the network can rely on stable cues on the object surface. However, this is particularly challenging for the adaptation network as it must encode the object geometry to reconstruct a correct appearance, while being confronted with merely an edge image generated from an uniformly colored object.

Note that the translation of the object appearance to a different representation in the last two methods, requires the translation network to be executed at inference time as well.

### 4.4.5 Direct image domain translation

In this section we introduce a CycleGAN [114] based training pipeline for unsupervised domain adaptation. As this model does not require matching image pairs, there is no need for an explicit intermediate representation, and the model can directly learn the mapping between the domains of synthetic and real images.

For the real domain we use the same generation scheme as in baseline b) of rendering randomly textured objects on random backgrounds. However, for the synthetic domain we cannot use real backgrounds to generate samples, as the different

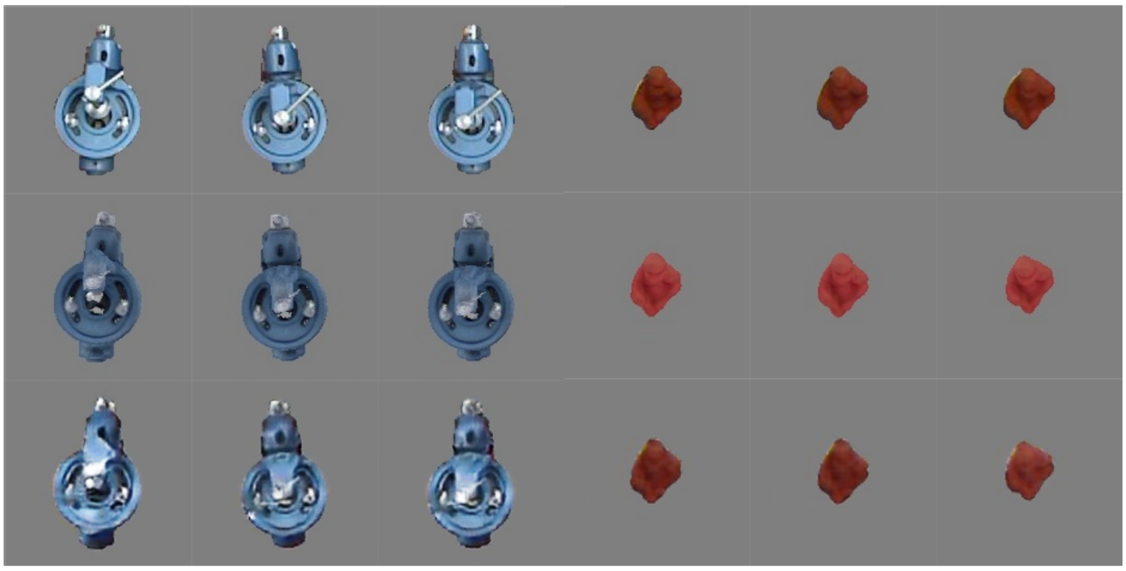
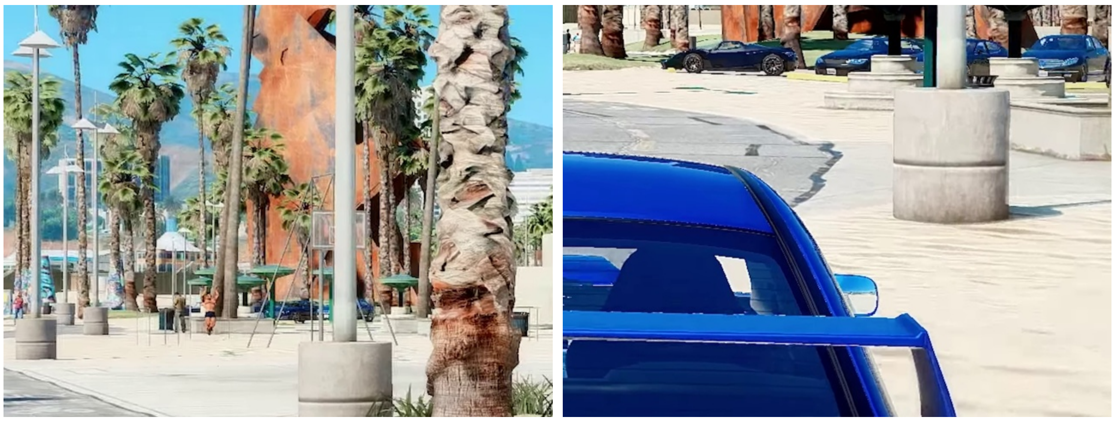


**Figure 4.16:** Domain translation capabilites of CycleGAN. *Top row:* masked crops from real images. *Middle row:* synthetic rendering without lighting and using vertex colors only. *Bottom row:* output of CycleGAN applied on the middle row, adding lighting effects and generally improving image fidelity.

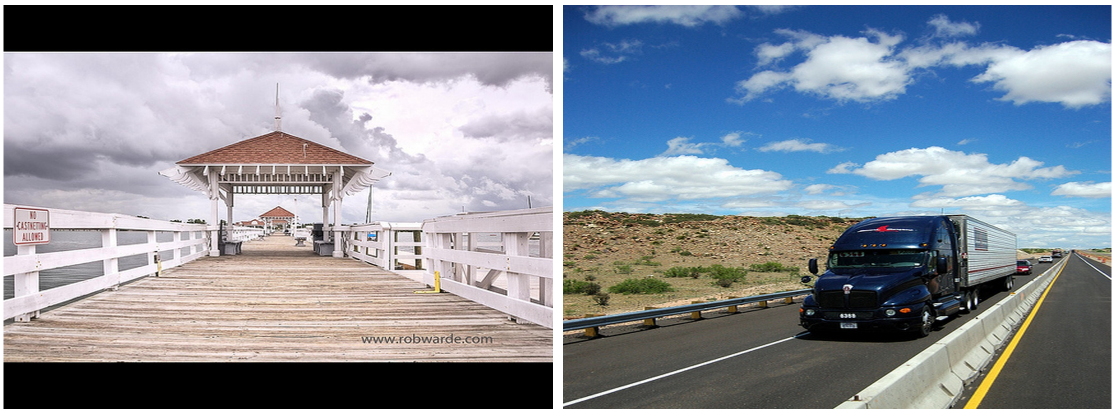
statistics would give away the synthetic object and bias the GAN towards object segmentation. Instead, we collect a separate data-set of synthetic background images from 3D-game footage on Youtube. Here, we select 50.000 random crops of randomly selected frames (see Figure 4.17) from a total of about 5 hours of video footage.

We train CycleGAN on the custom 3D-game and the ImageNet datsets to obtain a mapping between the real and the synthethic domain. To train the YOLO6D detector, we generate synthetic renderings and then map them into the real domain with the learned CycleGAN model. Furthermore, the CycleGAN architecture is capable of translating images in both directions. This allows reversing the pipeline; instead of adapting synthetic images to the real domain at training time, it is also possible to adapt real images to the synthetic domain at run-time. While this eases training, it comes with the cost of having to execute the translation network for inference.

We train CycleGAN on the custom 3D-game and the ImageNet datsets to obtain a mapping between the real and the synthethic domain. To train the YOLO6D detector, we generate synthetic renderings and then map them into the real domain with the learned CycleGAN model. Furthermore, the CycleGAN architecture is capable of translating images in both directions. This allows reversing the pipeline; instead of adapting synthetic images to the real domain at training time, it is also possible to adapt real images to the synthetic domain at run-time.



**(a)** 3D-game crops



**(b)** ImageNet

**Figure 4.17:** Examples of the datasets, that we use as backgrounds and for training CycleGAN

As a limitation, the original CycleGAN model is tuned to produce images at a resolution of $256 \times 256$ px, which is only a fraction of the YOLO6D receptive field of $416 \times 416$ px. Scaling the GAN up would require fine-tuning its hyper-parameters like the size of the hidden layers, which is out of the scope of this work. Therefore, we perform our experiments with the limited resolution, which allows to judge the feasibility of the method. However, one should keep in mind that pose precision can be improved by scaling the network output to match YOLO6D.

### 4.4.6 Evaluation

In this section we quantitatively and qualitatively evaluate, whether our pipeline allows the training of the demanding pose estimation task network from synthetic, randomized and non-photorealistic renderings only. For comparability with related work, we use the LineMod data-set for evaluation. Here, we focus on the "Driller" object as it exhibits a characteristic, non-uniform surface and is the most challenging object [93] for the pose estimation model.

In the following, we first present the results of the baseline approaches as introduced in section 4.4.1. We then turn to the paired image translation via an intermediate, reduced domain and finally present the results for direct image domain translation.

**Implementation details** For training the pipeline, we follow the procedure outlined by [93] in initially dropping the confidence loss, when training YOLO6D on a domain different from real images. This proved essential to allow the pose-estimator to adapt, as samples from the GAN exhibit significantly different colors and details than the ImageNet data-set, which the model was initialized on. After 500.000 samples the estimator was able to reach 85% recall which improved to 95% after 1.000.000 samples. Only after such initialization, we proceeded with training with the complete loss function.

If not specified otherwise, each benchmark used the following training parameters:

- stochastic gradient descent with momentum (0.9)
- 2700 epochs
- weight decay of 0.0005 and learning rate of 0.001
- batch size of 32.

When not explicitly aiming for convergence, the learning rate was kept constant, otherwise it was reduced gradually with advancing training.

**Quantitative results** We measure the domain translation performance of the presented methods in terms of pose estimation error of the task network [93]. Here, we employ two different metrics; the 3D translation and 3D rotation as well as the 2D corner re-projection error. The latter measures the error in screen-space and therefore is well suited for augmented-reality, while the former is more meaningful for robotic applications.

Comparing the baseline methods as introduced in Section 4.4.2 (see Figure 4.18), we see that the performance of baseline method b) is significantly reduced, although we ensured model convergence in both cases. This shows, that our task network for pose-estimation, YOLO6D, is not sufficiently conditioned to overcome the domain gap on its own, even when presented with a virtually unlimited amount of images from the synthetic domain.

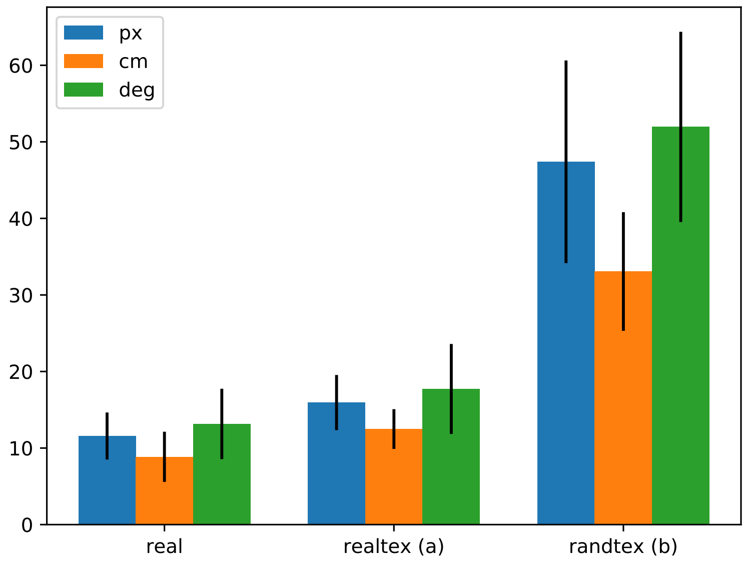


**Figure 4.18:** Pose estimation error of using real images as in [93] and the baseline methods a) and b) respectively.

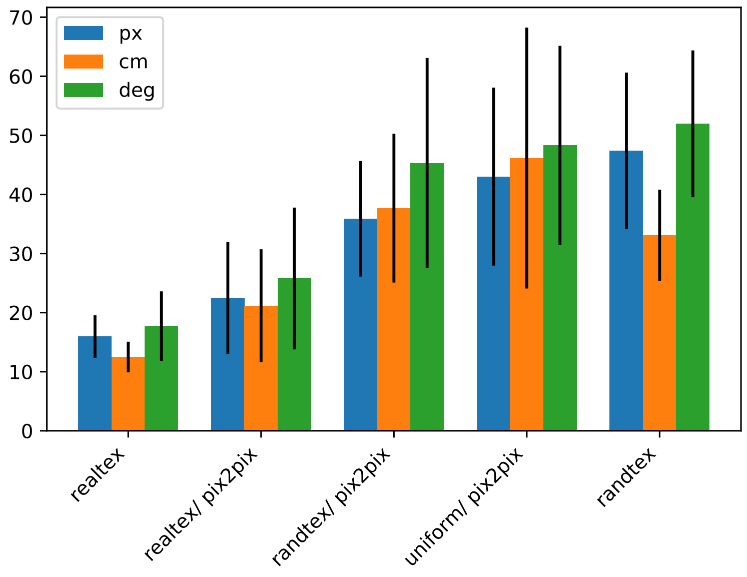


**Figure 4.19:** Pose estimation error of paired domain translation as introduced in Section 4.4.4

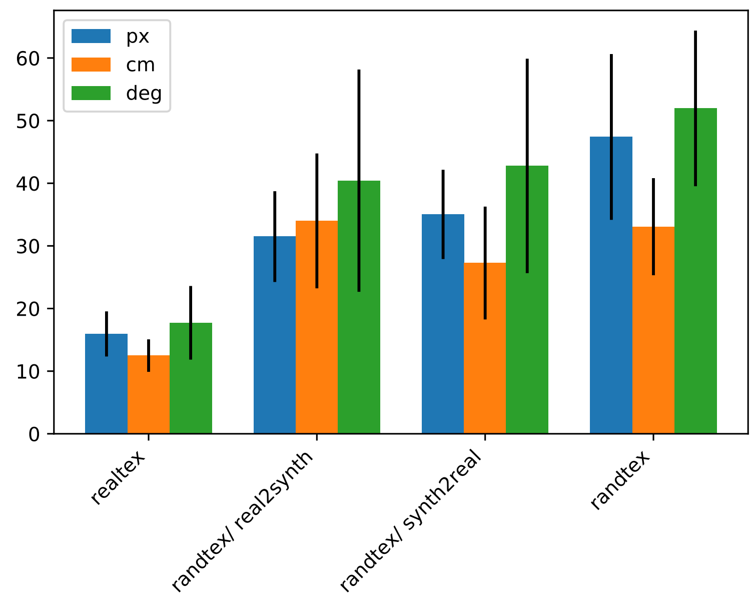


**Figure 4.20:** Pose estimation error of direct domain translation as introduced in Section 4.4.5, as well as translation reversal (real2synth).

Looking at the domain translation results using the paired model as introduced in Section 4.4.4 in Figure 4.19, we see a reduction of re-projection error by about 23% when using random texturing as in 2). This indicates that the edge-based intermediate representation leads to a more robust representation with YOLO6D. Likely, because we are able to reduce the texture bias [31] of the model. In contrast, rendering the object using a solid color as in 3) results in degraded results compared to using random texturing. Probably the pose-estimator overfits to the uniform colour present in the reconstruction, thus still suffering from the domain-gap. Training the pose estimation network was not possible when applying rendering-method 4).

While the results improve over baseline, the margin is only moderate. Likely, the reason for this is that the Laplace filtering does not sufficiently reduce the expressiveness of the image and the domain gap is still present in the intermediate representation.

Turning to the direct domain translation using CycleGAN as introduced in Section 4.4.5 in Figure 4.20, the results of synthetic to real translation are consistent with paired translation, even though the network produces images only at quarter the resolution. This leads us to believe that unsupervised models are generally sufficient for domain adaptation.

| Rendering method / Domain translation | Mean error | | |
|---|---|---|---|
| | re-projection | translation | angle |
| real images [93] | 12 px | 8.9 cm | 13.2° |
| realtex / none | 16 px | 12.5 cm | 17.7° |
| randtex / none | 47 px | 33 cm | 52.0° |
| realtex / laplace | 22.5 px | 21.2 cm | 25.8° |
| randtex / laplace | 36 px | 37.6 cm | 45.3° |
| uniform / laplace | 43 px | 46.1 cm | 48.3° |
| randtex / real2synth | 35 px | 27.2 cm | 42.8° |
| randtex / synth2real | 32 px | 34 cm | 40.4° |

**Table 4.2:** Raw data for Figure 4.18, Figure 4.19, Figure 4.20

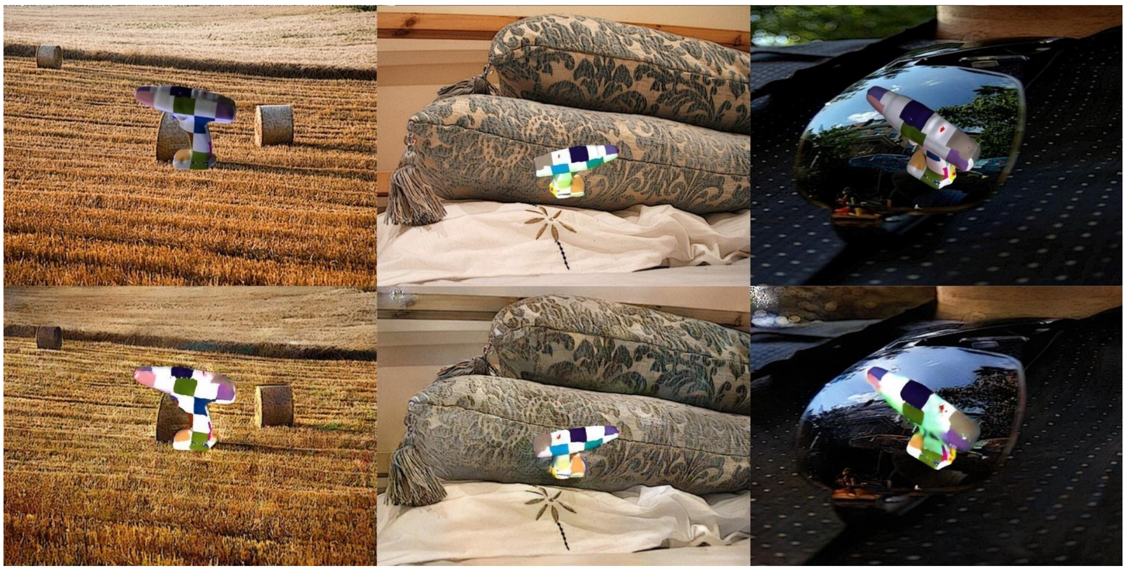


**Figure 4.21:** Checkerboard surface reconstruction results as required by surface rendering method 4). *Top row:* target image. The input to the network is a Laplace filtered version (confer Figure 4.14c) *Bottom row:* output of the PIX2PIXHD model trained to reconstruct the checkerboard pattern and applied to the whole image.

When reversing the translation from real to synthetic, we see a reduction of re-projection error by 31% compared to random texturing (b). However, the real-textured baseline (a) has a 50% lower error.

**Qualitative results** As shown in Figure 4.21, PIX2PIXHD cannot predict a consistent checkerboard texture. While the results are plausible, they do not exhibit enough detail to help the pose estimation model. Still, this is a remarkable result as the translation network is confronted with a significantly more demanding challenge; to correctly apply the texture pattern it not only has to infer the object pose, but also the object geometry.

Figure 4.10 shows some predictions of the pipeline trained with CycleGAN for domain adaptation. While the model is able to reliably detect the object, the accuracy of the object pose is lacking. This is likely due to the limited resolution of the adaptation network and can be further improved by scaling it up to the pose estimation input size.

Figure 4.16 shows the effect on image quality after applying CycleGAN on the objects "ape" and "benchvise". The model is able to improve color reproduction as well as applying details like specular highlights.

## 4.5 Conclusion

In this chapter, we have presented improvements to the state-of-the-art regarding transfer learning for CAD geometry in the following areas:

- We have described a method for real-time texturing that can be used to improve an object detection model on-the-fly. At this we have shown that texturing itself is crucial for detection of CAD data, where no surface information is available. However, even with meshes where vertex-colors were previously available, our approach improves detection performance significantly. Furthermore, we successfully applied the resulting textures to extend LineMod for object-instance recognition. By interleaving detection and texture extraction it now becomes possible to extend detection algorithms by color cues on-the-fly.

- We have shown that employing a paired translation GAN for domain adaptation during training generally improves model robustness and hence the performance of the target network. Turning to unpaired translation GANs, we have shown that training solely on CAD geometry with neither knowing the surface properties nor the environment is possible and is in the same error range as using paired GANs — with a slight advantage, when using real2synth translation. These results indicate that image-conditional GANs are indeed an effective measure to close the domain gap between real and synthetic images. However, when only relying on the CAD geometry with both the object background and object surface being randomized, there is still a considerable performance degradation.

  On the other hand, the introduced training is much simpler compared to existing solutions relying on domain randomization. The latter require a faithful setup of randomization modules — even when employing guided domain randomization. In contrast, the presented style transfer pipelines only require collecting unstructured images from the target domain, which are fed into the adaptation network in an unsupervised fashion. This allows focusing on training the task network. At this, we have shown that it is possible to train a pose-estimation model with satisfactory precision.

# 5 Discussion

The availability of scene understanding on an object-pose level is crucial for many computer vision applications. In this thesis we have outlined a pipeline to achieve this understanding, while only relying on a consumer-grade camera. First, we introduced a system which made reliable calibration data ubiquitously available, before we continued with the task of object detection and pose estimation. While there are many ways to perform object detection, we focused on approaches using deep CNN models. This choice imposes the least amount of restrictions on the application environment. Conversely, setting up a respective system is considerably more involved compared to classical algorithms. Here, we presented an approach that allows fully automating the training of a CNN model, if the geometry of the object is known. Particularly, the presented method is able to work with CAD geometry, where the actual surface properties are unknown.

In the following we summarize the contributions presented in this thesis and discuss possible perspectives for future work.

## 5.1 Contributions

In Chapter 3, we have addressed the task of acquiring reliable calibration data for a wide range of devices. The general approach can be summarized as follows:

- In chapter 3.3 we have presented a *pose selection* algorithm, that generates a compact set of calibration poses and is suitable for interactive user guidance. By monitoring the uncertainty of individual parameters it can adapt to inexact pose matching by the user and dynamically generate new poses ensuring that a given confidence level is reached. This allows even inexperienced users to perform a precise camera calibration in about 2 min.

- In chapter 3.4 we have further evaluated different camera setups and visualizations to be used for *user-guidance*. The conducted user survey, resulted in recommendations for an improved calibration setup, that makes the task more accessible for users.
- In chapter 3.5 we presented a *calibration aggregation service*, that allows the general deployment of web-based computer vision algorithms by providing a database of known calibrations. In case an unknown camera is encountered, it transparently falls back to the user-guidance method based on pose selection. Here, we presented an efficient client-server architecture, which allows for a purely web-based implementation of the guidance part. As only a web-browser environment is required, this enables the calibration of a wide range of devices including smartphones and AR glasses.

In Chapter 4, we have addressed the task of training a CNN model for object detection from synthetic, non-photorealistic, data only. Specifically, we followed the following approaches:

- In chapter 4.3, we have presented a real time texturing method that can be used to reconstruct the missing *surface information* of CAD data, which significantly improved the color fidelity compared to the vertex-color based methods used previously.
- In chapter 4.4, we have shown that off-the-shelf GAN architectures can be used to formulate the *domain gap* as a style transfer problem. Particularly, we have presented pipelines for both, supervised and unsupervised image translation architectures. Here, we introduced the intermediate edge domain for the supervised case.

## 5.2 Future work

This thesis has shown how a pipeline for object pose detection can be constructed in a semi-automated way. Nevertheless, the presented approaches have certain limitations that can motivate further research in this area. We will address particularly relevant aspects below.

### 5.2.1 Extending pose selection to complex lens models

In chapter 3.3, we have presented a pose selection algorithm, that efficiently generates a set of poses suitable for precise camera calibration. Here, we focused on the widespread radial and tangential distortion model, which covers many kinds of lenses. However, specialized camera setups or more severe construction flaws require more complex distortion models like thin prism [104], rational [62] and tilted sensor. Eventually, one could incorporate a detection of unused parameters. This would allow starting with the most complex distortion model which could be gradually reduced during calibration. Furthermore, the applicability of the proposed method to non-pinhole camera models like omnidirectional cameras [32] should be evaluated. Finally, the general method needs adaptation to special cases like;

- microscopy, where the depth of field limits the possible calibration angles and
- calibration for a large distance, which requires calibrating at out of focus distances, as scaling the pattern to distance is not applicable.

### 5.2.2 Adapting pose selection to improve guidance

In chapter 3.4, we have evaluated different camera setups and visualizations for guidance during calibration. While the surveys indicate benefits from using a specific visualization, our qualitative observations indicate that larger gains are to be expected from adapting the pose sequence then from modifying the pattern visualization. Particularly the arbitrary switching between the pose categories requires physical and mental switching on the user side. Additionally, we observed that matching 3 arbitrary rotations of the pattern to the target pose took considerably longer than matching the position. Therefore, the pose sequence should be adapted to further improvements on user guidance. Currently, the poses optimize the algorithmic constraints while neglecting the user. One option would be to look for a better compromise between these two objectives. Alternatively, one could introduce "guidance only" poses that are placed between the current pattern position and the target pose. Those would not be used for calibration, but rather give the user more hints on how to reach the target pose. Trivially one could insert the neutral pose between two calibration poses such that only 3 DOF change between each two displayed targets. As our compactness evaluation shows that the amount of required frames can be further reduced, the introduction of neutral transition poses would not necessarily increase calibration time.

### 5.2.3 Enable capturing complex surfaces

In chapter 4.3, we have presented a method that makes it possible to extract the texture of an object on the fly. However, the method relies on a fixed camera exposure. To account for exposure compensation as employed by many cameras, we presented a global exposure normalization approach, which, however, is error-prone. Here, reading the actual camera exposure could be used for accurate exposure fusion of the images. The surface specularity could be explicitly considered during merging [46]. Currently, we assume diffuse reflection, which systematically over-brightens specular surfaces. At this a plausibility test during merging could be used to reject implausible colors as caused by e.g. occlusion. As the LineMod detector is no longer state-of-the-art and further investigation is needed to similarly integrate our approach into an existing CNN-based method. This would require breaking up the end-to-end trained "black-box" to make the color information explicit.

### 5.2.4 Multi-modal image translation

In chapter 4.4, we have introduced GAN models to bridge the domain gap between real images and synthetic images. To consider the multitude of possible surface materials, the method relies on blind randomization during rendering. Instead, it seems beneficial to employ a generator that is aware of the multi-modal data and therefore is capable of producing different surface materials based on a noise vector. To this end, one could replace the image-conditional CycleGAN by a more recent variant like MUNIT [43]. Alternatively, the unconditional StyleGAN [49] architecture is capable of generating multiple styles in a similar fashion, as it is able to disentangle content from style. However, it needs further regularization to force it to closer resemble the input image.

## 5.3 Final thoughts

In this thesis, we have shown how to automate the setup of a camera for the task of object pose detection. Particularly, we introduced a camera calibration service and an easy to handle method for synthetic training of deep CNN models. This enables scene understanding for many internet-connected cameras today. Notably inside industrial production lines, which can now be dynamically set up and re-configured. However, similarly any current inference-capable smartphone can be used. This enables a wide range of new applications, as the choice of a device is no longer dictated by a vendor-specific framework. Indeed, the actual device can be considered as a black box thus allowing to fully focus on the task at hand.

The techniques developed in this thesis are however only the first step towards a wide deployment of the computer vision algorithms that are available today. At least partly inspired by the approaches in this work, researchers have started applying similar ideas to stereo-camera setups, depth sensors and beyond [54, 89, 72].

# A
# Appendix

## A.1 Supervising activities

The following bachelor and master thesis were supervised by the author.

1. Pöllabauer, Thomas; *Supervisors:* Rojtberg, Pavel; Kuijper, Arjan "STYLE: Style Transfer for Synthetic Training of a YoLo6D Pose Estimator". *TU Darmstadt.* Master Thesis, 2020.

2. Matthiesen, Moritz; *Supervisors:* Rojtberg, Pavel; Kuijper, Arjan "Interpolation von Kalibrier daten für Zoom und Autofokus Kameras". *TU Darmstadt.* Bachelor Thesis, 2019.

3. Huertas Celis, Lizeth Andrea; "Evaluation and design of camera calibration patterns". *hda Darmstadt.* Bachelor Thesis, 2018.

4. Bergmann, Tim Alexander; *Supervisors:* Kuijper, Arjan; Rojtberg, Pavel "Interaktive Echtzeit-Kalibrierung". *TU Darmstadt.* Bachelor Thesis, 2016.

5. Gorschlueter, Felix; *Supervisors:* Goesele, Prof. Dr. Michael; Rojtberg, Pavel "Eine quantitative Evaluation von Farbtransferverfahren für Augmented Reality". *TU Darmstadt.* Master Thesis, 2015.

In the years from 2016 until 2020 the author contributed to the lecture *Virtual and Augmented Reality* at the TU Darmstadt by supervising student projects and holding lectures on the topics of "Camera Calibration", "3D Visualization" and "Object Detection".

## A.2 Further publications

The following publications are related to the field of CAD-based object detection and have been co-authored. As they are not core to this work, they are mainly mentioned for completeness.

- Rojtberg, Pavel, and Audenrith, Benjamin. "x3ogre: connecting X3D to a state of the art rendering engine." *Proceedings of the 22nd International Conference on 3D Web Technology.* 2017. — [81]
- Engelke, Timo, Jens Keil, Pavel Rojtberg, Folker Wientapper, Michael Schmitt, and Ulrich Bockholt "Content first: a concept for industrial augmented reality maintenance applications using mobile devices." *Proceedings of the 6th ACM Multimedia Systems Conference.* 2015. — [24]
- Olbrich, Manuel, Tobias Franke, and Pavel Rojtberg. "Remote visual tracking for the (mobile) web." *Proceedings of the 19th International ACM Conference on 3D Web Technologies.* 2014. — [69]
- Wientapper, Folker, Harald Wuest, Pavel Rojtberg, and Dieter Fellner "A camera-based calibration for automotive augmented reality head-up-displays." *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR).* IEEE, 2013. — [106]

## A.3 Software contributions

During the course of this thesis, several contributions to open-source projects have been made. Particularly, I became the core maintainer of the Ogre3D rendering engine. The following list shows some notable mentions:

- Honorable mention for contributing to the OpenCV 3.2 release
  `https://opencv.org/opencv-3-2/`
- Maintainer of the Ogre3D rendering engine
  `https://github.com/OGRECave/ogre`
- Author of the *ovis* 3D rendering module in OpenCV
  `https://github.com/opencv/opencv_contrib`
- General availability of the calibDB calibration service
  `https://www.calibdb.net/`
- Source code release of posecalib
  `https://github.com/paroj/pose_calib`

# Bibliography

[1] Adobe Systems Inc. Digital negative specification, 2012. URL `https://www.adobe.com/products/dng/pdfs/dng_spec_1_3_0_0.pdf`.

[2] A. Antoniou, A. Storkey, and H. Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

[3] B. Atcheson, F. Heide, and W. Heidrich. Caltag: High precision fiducial markers for camera calibration. In *VMV*, volume 10, pages 41–48, 2010.

[4] L. Bavoil and M. Sainz. Screen space ambient occlusion. *NVIDIA developer information: http://developers. nvidia. com*, 6, 2008.

[5] Y. Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[6] T. Birdal, I. Dobryden, and S. Ilic. X-tag: A fiducial tag for flexible and accurate bundle adjustment. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 556–564. IEEE, 2016.

[7] J.-Y. Bouguet. Camera calibration toolbox for matlab. 2004.

[8] S. Brabec, T. Annen, and H.-P. Seidel. Practical shadow mapping. *Journal of Graphics Tools*, 7(4):9–18, 2002.

[9] E. Brachmann and C. Rother. Learning less is more-6d camera localization via 3d surface regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4654–4662, 2018.

[10] E. Brachmann, F. Michel, A. Krull, M. Ying Yang, S. Gumhold, et al. Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3364–3372, 2016.

[11] G. Bradski, A. Kaehler, and V. Pisarevsky. Learning-based computer vision with intel's open source computer vision library. *Intel Technology Journal*, 9 (2), 2005.

[12] T. Bray. The javascript object notation (json) data interchange format. Technical report, 2017.

[13] A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

[14] D. C. Burnett and A. Narayanan. getusermedia: Getting access to local devices that can generate multimedia streams. *W3C Editor's Draft*, 2011.

[15] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics & Automation Magazine*, 22(3):36–52, 2015.

[16] M. Callieri, P. Cignoni, M. Corsini, and R. Scopigno. Masked photo blending: Mapping dense photographic data set on high-resolution sampled 3d models. *Computers & Graphics*, 32(4):464–473, 2008.

[17] P. Chandler and J. Sweller. Cognitive load theory and the format of instruction. *Cognition and instruction*, 8(4):293–332, 1991.

[18] A. Collet, M. Martinez, and S. S. Srinivasa. The moped framework: Object recognition and pose estimation for manipulation. *The International Journal of Robotics Research*, 30(10):1284–1306, 2011.

[19] T. Collins and A. Bartoli. Infinitesimal plane-based pose estimation. *International Journal of Computer Vision*, 109(3):252–286, 2014.

[20] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[21] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[22] 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.

[23] P. K. Diederik, M. Welling, et al. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, volume 1, 2014.

[24] T. Engelke, J. Keil, P. Rojtberg, F. Wientapper, M. Schmitt, and U. Bockholt. Content first: a concept for industrial augmented reality maintenance applications using mobile devices. In *Proceedings of the 6th ACM Multimedia Systems Conference*, pages 105–111, 2015.

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

[26] M. Fiala and C. Shu. Self-identifying patterns for plane-based camera calibration. *Machine Vision and Applications*, 19(4):209–216, 2008.

[27] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[28] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng. Complete solution classification for the perspective-three-point problem. *IEEE transactions on pattern analysis and machine intelligence*, 25(8):930–943, 2003.

[29] S. Garrido-Jurado. Detection of ChArUco Corners, 2017. URL `http://docs.opencv.org/3.2.0/df/d4a/tutorial_charuco_detection.html`. [Online; accessed 11-February-2017].

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

[31] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, 2019.

[32] C. Geyer and K. Daniilidis. A unifying theory for central panoramic systems and practical implications. In *Computer Vision—ECCV*, pages 445–461. Springer, 2000.

[33] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[34] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[35] R. Hartley and A. Zisserman. Multiple view geometry in computer vision. *Robotica*, 23(2):271–271, 2005.

[36] J. Heikkila and O. Silvén. A four-step camera calibration procedure with implicit image correction. In *Computer Vision and Pattern Recognition, Proceedings., 1997 IEEE Computer Society Conference on*, pages 1106–1112. IEEE, 1997.

[37] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 858–865. IEEE, 2011.

[38] 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 *Asian conference on computer vision*, pages 548–562. Springer, 2012.

[39] S. Hinterstoisser, V. Lepetit, P. Wohlhart, and K. Konolige. On pre-trained image features and synthetic images for deep learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[40] S. Hinterstoisser, O. Pauly, H. Heibel, M. Martina, and M. Bokeloh. An annotation saved is an annotation earned: Using fully synthetic training for object detection. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[41] T. Hodaň, J. Matas, and Š. Obdržálek. On evaluation of 6d object pose estimation. *European Conference on Computer Vision Workshops (ECCVW)*, 2016.

[42] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[43] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz. Multimodal unsupervised image-to-image translation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 172–189, 2018.

[44] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[45] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[46] J. Jachnik, R. A. Newcombe, and A. J. Davison. Real-time surface light-field capture for augmentation of planar specular surfaces. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 91–97. IEEE, 2012.

[47] J. Kannala and S. S. Brandt. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE transactions on pattern analysis and machine intelligence*, 28(8):1335–1340, 2006.

[48] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

[49] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[50] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *Proceedings of the International Conference on Computer Vision (ICCV 2017), Venice, Italy*, pages 22–29, 2017.

[51] A. Kendall, M. Grimes, and R. Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[52] L. Kneip, H. Li, and Y. Seo. Upnp: An optimal o (n) solution to the absolute pose problem with universal applicability. In *European Conference on Computer Vision*, pages 127–142. Springer, 2014.

[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[54] A. Langbein, D. A. Plecher, F. Pankratz, C. Eghtebas, F. Palmas, and G. Klinker. Using gamification for stereo camera calibration in terms of augmented reality. 2020.

[55] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[56] V. Lepetit, F. Moreno-Noguer, and P. Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009.

[57] M. Li, D. M. Kaufman, V. G. Kim, J. Solomon, and A. Sheffer. Optcuts: joint optimization of surface cuts and parameterization. In *SIGGRAPH Asia 2018 Technical Papers*, page 247. ACM, 2018.

[58] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[59] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[60] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[61] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. In *Advances in neural information processing systems*, pages 6231–6239, 2017.

[62] L. Ma, Y. Chen, and K. L. Moore. Rational radial distortion models of camera lenses with analytical solution for distortion correction. *International Journal of Information Acquisition*, 1(02):135–147, 2004.

[63] S. Magnenat, D. T. Ngo, F. Zünd, M. Ryffel, G. Noris, G. Rothlin, A. Marra, M. Nitti, P. Fua, M. Gross, et al. Live texturing of augmented reality characters from colored drawings. *IEEE transactions on visualization and computer graphics*, 21(11):1201–1210, 2015.

[64] F. Mahmood, R. Chen, and N. J. Durr. Unsupervised reverse domain adaptation for synthetic medical images via adversarial training. *IEEE transactions on medical imaging*, 37(12):2572–2581, 2018.

[65] F. Manhardt, W. Kehl, N. Navab, and F. Tombari. Deep model-based 6d pose refinement in rgb. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 800–815, 2018.

[66] I. MongoDB. Mongodb. *URL https://www. mongodb. com/. Cited on*, page 9, 2019.

[67] J.-M. Morel and G. Yu. Asift: A new framework for fully affine invariant image comparison. *SIAM journal on imaging sciences*, 2(2):438–469, 2009.

[68] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[69] M. Olbrich, T. Franke, and P. Rojtberg. Remote visual tracking for the (mobile) web. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies*, pages 27–33, 2014.

[70] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[71] F. Pankratz and G. Klinker. [poster] ar4ar: Using augmented reality for guidance in augmented reality systems setup. In *Mixed and Augmented Reality (ISMAR), 2015 IEEE International Symposium on*, pages 140–143. IEEE, 2015.

[72] S. Peng and P. Sturm. Calibration wizard: A guidance system for camera calibration based on modelling geometric and corner uncertainty. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1497–1505, 2019.

[73] A. Prakash, S. Boochoon, M. Brophy, D. Acuna, E. Cameracci, G. State, O. Shapira, and S. Birchfield. Structured domain randomization: Bridging the reality gap by context-aware synthetic data. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7249–7255. IEEE, 2019.

[74] J. Rambach, C. Deng, A. Pagani, and D. Stricker. Learning 6dof object poses from synthetic single channel images. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 164–169. IEEE, 2018.

[75] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[76] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[77] E. Reinhard, M. Adhikhmin, B. Gooch, and P. Shirley. Color transfer between images. *IEEE Computer graphics and applications*, 21(5):34–41, 2001.

[78] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[79] A. Richardson, J. Strom, and E. Olson. AprilCal: Assisted and repeatable camera calibration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 2013.

[80] P. Rojtberg. User guidance for interactive camera calibration. In *International Conference on Human-Computer Interaction*, pages 268–276. Springer, 2019.

[81] P. Rojtberg and B. Audenrith. x3ogre: connecting x3d to a state of the art rendering engine. In *Proceedings of the 22nd International Conference on 3D Web Technology*, page 2. ACM, 2017.

[82] P. Rojtberg and F. Gorschlüter. calibdb: enabling web based computer vision through on-the-fly camera calibration. In *The 24th International Conference on 3D Web Technology*, pages 1–4, 2019.

[83] P. Rojtberg and A. Kuijper. Efficient pose selection for interactive camera calibration. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 31–36. IEEE, 2018.

[84] P. Rojtberg and A. Kuijper. Real-time texturing for 6d object instance detection from rgb images. In *2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 295–300. IEEE, 2019.

[85] P. Rojtberg, T. Pöllabauer, and A. Kuijper. Style-transfer gans for bridging the domain gap in synthetic pose estimator training. In *2020 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pages 188–195. IEEE, 2020.

[86] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[87] B. Seo, H. Park, J. Park, S. Hinterstoisser, and S. Ilic. Optimal Local Searching for Fast and Robust Textureless 3D Object Tracking in Highly Cluttered Backgrounds. *IEEE Transactions on visualization and computer graphics (TVCG)*, 2013.

[88] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2930–2937, 2013.

[89] V. Sterzentsenko, A. Doumanoglou, S. Thermos, N. Zioulis, D. Zarpalas, and P. Daras. Deep soft procrustes for markerless volumetric sensor alignment. *arXiv preprint arXiv:2003.10176*, 2020.

[90] P. F. Sturm and S. J. Maybank. On plane-based camera calibration: A general algorithm, singularities, applications. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on.*, volume 1. IEEE, 1999.

[91] W. Sun and J. R. Cooperstock. Requirements for camera calibration: Must accuracy come with a high price? In *Application of Computer Vision, WACV/MOTIONS'05 Volume 1. Seventh IEEE Workshops on*, volume 1, pages 356–361. IEEE, 2005.

[92] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel. Implicit 3d orientation learning for 6d object detection from rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 699–715, 2018.

[93] B. Tekin, S. N. Sinha, and P. Fua. Real-time seamless single shot 6d object pose prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 292–301, 2018.

[94] 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. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[95] A. Torralba and A. A. Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528. IEEE, 2011.

[96] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 969–977, 2018.

[97] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In *Conference on Robot Learning*, pages 306–316, 2018.

[98] B. Triggs. Autocalibration from planar scenes. In *Computer Vision—ECCV'98*, pages 89–105. Springer, 1998.

[99] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *Robotics and Automation, IEEE Journal of*, 3(4):323–344, 1987.

[100] A. Tsirikoglou, J. Kronander, M. Wrenninge, and J. Unger. Procedural modeling and physically based rendering for synthetic data generation in automotive applications. *arXiv preprint arXiv:1710.06270*, 2017.

[101] M. Waechter, N. Moehrle, and M. Goesele. Let there be color! large-scale texturing of 3d reconstructions. In *European Conference on Computer Vision*, pages 836–850. Springer, 2014.

[102] J. Wang and E. Olson. AprilTag 2: Efficient and robust fiducial detection. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016.

[103] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018.

[104] J. Weng, P. Cohen, and M. Herniou. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (10):965–980, 1992.

[105] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald. Robust real-time visual odometry for dense rgb-d mapping. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5724–5731. IEEE, 2013.

[106] F. Wientapper, H. Wuest, P. Rojtberg, and D. Fellner. A camera-based calibration for automotive augmented reality head-up-displays. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 189–197. IEEE, 2013.

[107] World Wide Web Consortium. Media Capture and Streams, 2017. URL `https://www.w3.org/TR/mediacapture-streams/`. Candidate Recommendation, 3 October 2017.

[108] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.019.

[109] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[110] A. Zakai. Emscripten: an llvm-to-javascript compiler. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 301–312. ACM, 2011.

[111] S. Zakharov, W. Kehl, and S. Ilic. Deceptionnet: Network-driven domain randomization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 532–541, 2019.

[112] Z. Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, 2000.

[113] Q.-Y. Zhou and V. Koltun. Color map optimization for 3d reconstruction with consumer depth cameras. *ACM Transactions on Graphics (TOG)*, 33 (4):155, 2014.

[114] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.