Machine Learning and Robotics Lab

Masterarbeit

Deep 6-DoF Tracking of Unknown Objects during Robotic Manipulation

Marc Tuscher

Course of Study:	Computer Science
Examiner:	Prof. Dr. Marc Toussaint
Supervisor:	Prof. Dr. Marc Toussaint Danny Driess, M.Sc.
Commenced:	October 14, 2018
Completed:	May 27, 2020

Abstract

Robotic manipulation of unknown objects is an important field of research. Practical applications occur in many real-world settings where robots need to interact with an unknown environment. Deep learning methods seem promising to tackle such environments, due to their ability to generalize to unseen data. This work combines Siamese Networks for visual tracking with an Iterative Closest Point approach for pointcloud registration to propose a conceptually simple, but powerful method for 6-DoF unknown object tracking. The method does not require further training and is robust to noise and occlusion. Integrating this work in a system for dynamic grasping in changing environments, yields a robotic manipulation system, which is able to grasp formerly unseen objects and is robust against object perturbations and inferior grasping points.

Kurzfassung

Robotergestütze Manipulation von unbekannten Objekten ist ein relevantes Forschungsgebiet. Praktische Anwendungen finden sich in vielen Bereichen der realen Welt, in denen Roboter mit einer unbekannten Umgebung interagieren müssen. Aufgrund ihrer Fähigkeit, auf unbekannte Daten generalisieren zu können, sind Methoden des Deep Learnings eine vielversprechende Technik um solche Umgebungen zu addressieren. Diese Arbeit kombiniert Siamese Networks für visuelles Objekt Tracking mit einer ICP Methode, zu einer konzeptionell einfachen, aber performanten Methode um unbekannte Objekte in sechs Freiheitsgraden zu tracken. Die Technik ist robust gegen Verdeckung und benötigt kein weiteres Training. Durch die Integration dieser Arbeit in ein System für dynamisches Greifen, entsteht ein Manipulationssystem, das in der Lage ist, vorher unbekannte Objekte zu greifen.

Contents

List of Figures

List of Algorithms

1 Introduction

The need for robotic automation is ubiquitous, from domestic [1] to industrial settings where traditional automation technology is limited [2]. Robots can further play a key role in general medical environments [3], as well as in fighting global infectious diseases [4].

Aiming towards a wide-spread application of robots in natural and unstructured industrial environments, methods for robotic perception need to incorporate more general approaches. Robots manipulating in noisy and cluttered real-world scenarios, e.g. in medical, or domestic settings, require the ability to interact with arbitrary, unknown objects. Interacting with objects in the real world inherently requires the ability to efficiently perceive the pose of the target objects. Target objects need to be tracked in real-time. Versatile and robust tracking algorithms are required.

Object tracking methods are often trained on specific classes of objects [5], [6] or require a 3D model of the target object a priori [7]–[10]. This heavily limits their usability for practical applications. Existing tracking methods for unknown object tracking either target coarse grained tracking [11]–[14], are prone to pose drifting [15] or limited to a specific environment [16].

Harnessing the properties of Convolutional Neural Networks seems promising, however, datasets containing labels for tracking in six degrees of freedom (6-DoF) are rare [5]. Siamese Networks dominate the 2D object tracking benchmarks [17], mostly due to transfer learning [18] of very deep neural networks, pretrained on ImageNet [19]. Transfer learning RGB pretrained Siamese Networks to 6-DoF object tracking on RGBD data is not straightforward. The depth channel provides key information for tracking, yet, incorporating depth in RGB pretrained CNNs is not trivial.

In order to leverage the full potential of RGB based deep learning methods, the proposed approach tracks objects in 2D using RGB input and uses the segmentation mask of the target object as starting point for an Iterative Closest Point (ICP) method on depth data. ICP based methods are generally heavily affected by outliers and occlusions [20], [21]. Having access to a segmentation mask of the target object, depth pixels, only corresponding to the target object can be select. Thus, no occluding objects are contained in the input to ICP, so the ICP algorithm does not need to cope with occlusions. Further, this segmentation mask also reduces the number of points in the observation pointcloud, therefore reducing the possibility of outliers and simplifying correspondence search.

This work proposes a method for 6-DoF tracking based on Siamese Networks for 2D tracking and ICP for pointcloud registration. Specifically, SiamMask [17] is used in conjunction with THOR [22] as a template memory to track and segment the target object in the current frame. The segmentation mask is used to generate a pointcloud from the depth image, containing only points corresponding to the target object. Generalized-ICP (G-ICP) [23] is then used to fit a template pointcloud of the object to this observation, starting from the last known object pose as initial guess.

While the method is able to run in a standalone fashion, priors on the object pose are incorporated, when available, e.g. during robotic manipulation, an estimate can be computed using forward kinematics. A 3D bounding box containing the object is projected onto the image plane, and the estimated object pose is used as an initial guess for G-ICP. This approach yields a real-time system, named THOR + G-ICP, able to track arbitrary unknown objects without any prior knowledge.

Every object tracking method needs a method for object detection. Either in every frame, such methods can be called "tracking by detection"-methods. Or when the object first appears, and after that a temporal state is kept by the tracking method [24]. Such methods are called "temporal tracking" methods. The method proposed in this work does not rely on a complicated state to be propagated from the past to the current prediction, but is only able to work efficiently, when given position and size of the target object in the last RGB frame and an estimate of the cartesian pose, thus, the approach can be classified as a temporal tracking method. For object detection, this work relies on background subtraction in depth images. However, THOR + G-ICP does not depend on a specific method for object detection.

Experiments show that this approach to object tracking proves useful in various tracking tasks and object manipulation in the real-world, while keeping the advantage to be model-free and class-agnostic. A limitation of the approach is that under heavy change of the target objects orientation, tracking using the initial pointcloud as template does not work. Additionaly, to overcome the short-comings of a model-free approach, a method to accumulate a 3D pointcloud model of the target object during tracking is layed out.

Interacting with objects often requires grasping. Most grasping applications aim towards predicting the best possible grasping point given a single RGBD image or pointcloud of a scene [25]–[28]. However, little effort is put into the natural approach of having a second try, if grasping fails, e.g. due to object movements, scene changes, noise or poor grasp points. Combining THOR + G-ICP with a method for dynamic grasp planning and collision avoidance, yields a manipulation system, robust against object perturbations or inferior grasp points.

This work contributes in the following ways:

1. To the best of knowledge, it is the first approach combining a neural network for RGB tracking with ICP to realize a conceptually simple, but powerful 6-DoF tracking method: this method is able to track arbitrary unknown objects under occlusion in real-time, while being class-agnostic and model-free.
2. The method supports incorporating priors on the object pose from robot kinematics.
3. Extensive quantitative comparison of the proposed method to a state-of-the-art object tracking method.
4. Performance of THOR + G-ICP is shown in combination with an approach to reactive grasping. This leads to a real-world robot application able to robustly grasp arbitrary dynamic objects.

The work is structured as follows: in chapter 2 related work is presented, chapter 3 takes a deep dive into the underlying methods used in this work and chapter 4 explains the object detection method. THOR + G-ICP is described in 5. In chapter 6 quantitative experiments conducted in simulation are layed out, as well as a qualitative analysis on real-world robotic grasping. Further, chapter 7 draws a conclusion on the method and its effectiveness.

Figure 1.1: THOR + G-ICP tracking a formerly unseen object, while a robot tries to grasp it. Tracking is done from the side camera seen in the figure. While a human moves the object, major parts of it are occluded by the gripper finger of the robot. Grasping is successfull either way.

2 Related Work

This chapter provides a scientific discussion to state-of-the-art approaches for 6-DoF object tracking, mainly but not exclusively, with focus on robotic manipulation. However, due to the wide field of research on object tracking, the reader should not expect completeness. First, related approaches to model-based tracking, pose estimation and unknown object tracking are discussed. Second, methods which serve as a foundation for THOR + G-ICP are outlined.

Model-based tracking To this date, the most popular paradigm for 6-DoF object tracking is model-based object tracking. That is, a 3 dimensional description of the target object is available a priori. A large body of work has been evaluated on this topic. Particle Filtering [29] proves to be useful in model-based object tracking [30]. Particle filter based tracking models often operate directly on pointclouds. Incorporating control inputs from the robot, to give a good estimate on the current object is also successfully applied to particle filter trackers [7]. Other appraoches extend particle filter tracking to multi-target tracking using a mixture in particles [30] and learning the correct correspondence to a certain target object with the AdaBoost algorithm [31]. [32] uses a particle filter algorithm operating on RGB-pointclouds, thus, integrating texture information into tracking. GPU implementations of particle filters boost their real-time capabilities [33]. Extending filter based tracking to Gaussian filters yields a higher accuracy in object tracking and more robustness against occlusions [34].

Fully integrating the robot kinematics in object tracking helps to reduce the degrees of freedom of the target object and serves as a good prior for vision based object tracking in robotic manipulation [9], [20]. However, such approaches rely on both, an object model and a robot model being present beforehand, and reduce the general applicability of the system. Schulman et. al [10] track deformable objects using a probabilisitc model in conjunction with a physical model for pointcloud registration.

Rendering multiple simulated views from the 3D model of the target object can be used to leverage deep transfer learning for pose estimation [35]. Garon et. al [8] use convolutional neural networks to directly operate on RGBD data, by training from scratch for each individual object.

Using visual template-matching with ICP pointcloud registration is also applied successfully to model-based object tracking [36]. The ICP point-to-plane error in conjunction with the contour error from RGB images can be used to formulate an update step on the current object pose [37].

Pose estimation Pose estimation and Object Detection are related fields of research: in [38] a deep learning method for object detection is extended by an object database, yielding an object tracking system. Many approaches extend deep learning models for visual object detection to 6-DoF pose estimation from RGB data [39]–[41]. Applying deep learning models directly

on pointcloud data seems promising since the rise of PointNet feature extractors [42], [43]. VoteNet [44] combines deep learning with Hough voting [45] to predict 3D bounding boxes on pointcloud input data.

However, most approaches to pose estimation lack the ability to run in real-time, and therefore render impractical for robotic manipulation. Tremblay et. al [5] train a deep neural network for pose estimation on synthetic RGB data using domain randomization. The method is suited as a real-time system for real-world robotic grasping of known objects. The major weakness of all approaches presented in this section, is not being class-agnostic. That is, most of these methods only estimate poses of objects belonging to classes contained in the training dataset.

Unknown objects Despite the practical importance, only little research is done on tracking of unknown objects in robotics applications. Early work is done using optical flow tracking and an integrated eye-in-hand vision system to grasp arbitrary objects with a visual servoing appraoch [46]. Experiments show that the approach only works in a specific environment. Other approaches rely on detecting and segmenting unkown objects in a known environment [16] or do not take the shape of the object into account [47]. Methods based on a large number of different algorithms, each tuned for a specific application, are usually brittle and sensitive to changing environments. A more recent approach to unknown object tracking applies multiple deep learning models to first segment the scene into model segments, then predict the position and orientation relative to the last frame [15]. Therefore, the method bootstraps the current object pose on earlier estimations. Experiments suggest that this approach is particularly prone to tracking drift, stable tracking of real-world objects is only possible for approximately one second.

Siamese Networks Visual tracking is a fundamental computer vision problem and focuses on finding a 2D bounding box in the image plane. Template-matching methods for visual tracking aim to find an object contained in a template image in a novel frame. Siamese Networks, originally developed for One-Shot Learning, form a great foundation for template-matching methods. Combined with a powerful pretrained feature extractor, approaches based on Siamese Networks as a matching function, constantly achieve a new state-of-the-art performance in various benchmarks for visual object tracking.

Siamese Instance Search for Tracking (SINT) [48] is the first visual object tracker to apply Siamese Networks for template-matching to visual object tracking. Despite setting a new state-of-the-art on OTB [49] the matching function is learned from scratch on the ALOV dataset [50] without any pretraining on ImageNet. Due to the use of fully convolutional networks the approach proposed in [51] is able to evaluate a siamese matching function on multiple areas of the image at once and therefore runs an order of magnitude faster (approximately 80 fps) while achieving similar results. Even faster inference times (approximately 160 fps) are achieved using Region Proposal Networks [52]. Using ResNets as backbone for feature extraction in Siamese Networks, introduces problems due to no strict translation invariance of the features. Mitigating these problems and using features from different stages for template-matching yields an even higher accuracy [53]. However, SiamMask [17] proposes an extension, which not only returns a bounding box containing the target object, but also a segmentation mask segmenting the object from the scene. THOR (Tracking Holistic Object Representations) [54] is a template memory module, storing multiple templates, which can be used on top of any template-matching method. However, it is particularly

useful when used with siamese networks for tracking, since these models inherently rely on an inner product in feature space for similarity comparison. Using multiple templates for tracking proves useful for robotic applications: during manipulation, objects are likely to change their visual appearence due to rotation and deformation. [55] extends the Siamese Network architecture for template-matching to long-term tracking.

Iterative Closest Point One of the most famous methods for pointcloud registration is the Iterative-Closest-Point (ICP) algorithm [56]. Over the years this field of research has seen many variations of ICP [57]–[60]. Generalized-ICP (G-ICP) [23] improves the ICP algorithm by formulating the minimization step as MLE on a probability distribution over distances of corresponding points. This makes the method more robust against noise and outliers.

The approach presented in this work combines SiamMask with THOR for template-matching in RGB frames, and uses G-ICP for pointcloud registration in depth observations to track the 3D pose of unknown objects. All classes of objects can be tracked, no further training is needed. Compared to existing methods for unknown object tracking, THOR + G-ICP is fast, accurate, not susceptible to noise and occlusions or cluttered environments, and can track the object for an arbitrary amount of time. Integrating priors on the object pose from the robot kinematics is also simple and straightforward.

3 Background

The following section contains general definitions of problems which are considered subproblems of 6-DoF tracking, and algorithms this work builds upon to solve the problem of 6-DoF tracking.

3.1 Deep Learning

This section gives an overview on the theoretical foundations of the methods used in this work for RGB tracking. If not stated otherwise, the sections are based on lectures on Machine Learning[1], the book by Ian Goodfellow et. al. [61] or the `fastai`-online courses[2]. Code for the practical examples used in this section can be found on github [3]. Implementations are based on the `fastai`-library [62] or `detectron2` [63]. Both libraries use the PyTorch [64] deep learning library. Practical examples are run on a machine with an AMD Ryzen 1700x CPU, 48 GB RAM and a NVIDIA GeForce GTX 1080 Ti GPU.

3.1.1 Neural Networks

Neural networks are functions $f_\theta : \mathbb{R}^d \to \mathbb{R}^m$ parametrized by θ, consisting of a set of weight matrices W and a set of biases b. A two layer feed-forward, or fully-connected neural network is defined by the function

$$f_\theta(x) = W_2\sigma(W_1 x + b_1) + b_2 \tag{3.1}$$

where $W = \{W_1, W_2\}$ and $b = \{b_1, b_2\}$ and σ is some activation function. Activation functions are typically non-linear and applied element-wise. In modern deep learning models, the most common acitvation function is the Rectified Linear Unit (ReLU) function

$$\sigma(z) = \max\{0, z\}.$$

This function in particular has the practical property of having a gradient which is either 0, or 1 if the *neuron* z_i is active. Therefore, not contributing heavily to the problems of vanishing or exploding gradients. Given a dataset $D = \{(x_i, y_i)\}_{i=1}^n$ and a differentiable loss function ℓ, an optimal set of parameters θ^* is found by minimizing the loss

$$\theta^* = \arg\min_\theta \sum_i \ell(f_\theta(x_i), y_i).$$

[1] https://github.com/MarcToussaint/AI-lectures/releases/download/v0.1/script.pdf
[2] https://www.fast.ai/
[3] https://github.com/marctuscher/cnn

The procedure of finding such a θ^* is called training and is done via stochastic gradient descent: a subset $\hat{D} \subset D$, called mini-batch, is sampled from D at random. The loss function is evaluated and the gradient with respect to each weight in θ is computed

$$\mathcal{L}(\hat{D},\ \theta) = \sum_{x,y \in \hat{D}} \ell(f_\theta(x), y)$$
$$\nabla_\theta \mathcal{L}(\hat{D},\ \theta) = \sum_{x,y \in \hat{D}} \nabla_\theta \ell(f_\theta(x), y).$$

Being a good estimate for the direction of steppest descent, the negative gradient is used as an update on the current neural network weights

$$\theta \leftarrow \theta_{old} - \eta \nabla_\theta \mathcal{L}(\hat{D},\ \theta). \tag{3.2}$$

This procedure is iterated over the complete dataset and repeated for a certain number of epochs, or until convergence.

For an arbitrary function $g : \mathbb{R}^k \to \mathbb{R}^l$, the direction of steepest descent is defined as the direction, where, when making a step of length 1, the largest decrease of g in its linear approximation is encountered:

$$\arg\min_{\delta} \nabla g(x)^T \delta, \quad \text{s.t.} ||\delta|| = 1.$$

The negative gradient used in equation 3.2, is therefore only an estimate on the direction of steepest descent, because in vector spaces with non-euclidean inner product $\langle v, w \rangle_G$ and non-euclidean metric tensor G the direction of steepest descent is

$$\delta \sim -G^{-1} \nabla g.$$

Further, the magnitute $|\nabla_\theta f|$ can not be trusted to supply a meaningful stepsize. Relying on the magnitude of the gradient would end in taking small steps in plateaus and large steps at steep slopes.

Since equation 3.2 adds θ, which is a vector with contravariant coordinates, and the gradient $\nabla_\theta f$, which is, by definition, the transpose of the 1-form called derivative, with covariant coordinates, the result is not invariant under coordinate transforms. Having access to the Hessian $\nabla^2_\theta f$ and using newton steps as an update

$$\theta \leftarrow \theta - \nabla^2_\theta f(x)^{-1} \nabla_\theta f(x)$$

would mitigate major problems with gradient descent. Newton steps give a very robust step direction since the Hessian $\nabla^2_\theta f$ acts like local a metric. Further, taking a full newton step results in jumping to the minimum of the local 2nd-order Taylor aproximation, and therefore yielding a meaningful stepsize.

The caveat is, that in useful deep learning models, θ is in general too large to efficiently compute $\nabla^2_\theta f$ and therefore algorithms for stochastic gradient descent rely on setting a good learning rate, or stepsize, η.

The learning rate is arguably the most important parameter for training neural networks. Algorithms like Adam [65] successfully incorporate information from past iterations, like the momentum of the optimization process, to compute a more useful stepsize. Often, η is also

manually adapted, e.g. by continously decaying the learning rate after a certain number of epochs. A relatively new approach to setting stepsizes is learning rate cycling [66]. The learning rate is cycled between a lower and an upper bound, which helps getting out of saddle points. The One-Cycle Policy [67] trains with different learning rates during a single cycle, e.g. 10k iterations. Starting near the lower bound, increasing the learning rate to arrive at the upper bound in the middle of the cycle, then decreasing to the lower bound (cf. fig. 3.8b). The higher learning rate in the middle works as regularization, and keeps the model from overfitting.

In traditional machine learning models, a feature function $\phi : \mathbb{R}^k \to \mathbb{R}^d$ maps the raw data

$$x \mapsto \phi(x)$$

to some arbitrary feature space. ϕ is often chosen manually, and machine learning models operate only in the feature space, e.g. in linear regression, the model is linear in features and defined by the vector $\beta \in \mathbb{R}^d$:

$$f_{lreg} : \mathbb{R}^d \to \mathbb{R}^m$$
$$f_{lreg} : x \mapsto \phi(x)^T \beta.$$

Neural networks operate on raw data, the mapping function ϕ is also learned by the model. Deep learning models are especially good in learning a meaningful low dimensional representation of potentially very high dimensional data such as images. Often, this representation is used in conjunction with a simple, easy to train linear model [68].

The layered structure of neural networks advocates the use of computation graphs, or function networks, for realizing deep learning models. In a computation graph, the chain rules can be applied efficiently to compute gradients w.r.t. every input. Since functions realized by neural networks typically have only a few outputs, but many inputs (the weights are also inputs) the backward chain rule

$$\frac{df}{dx} = \sum_{g:x \in \pi(g)} \frac{df}{dg} \frac{\partial g}{\partial x}$$

is used.

Frameworks like TensorFlow [69] and PyTorch [64] apply the paradigm of computing using computational graphs efficiently. With such frameworks, deep learning models and training procedures can be implemented with little programmatical overhead and training can be run easily on multiple GPUs.

3.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are neural networks with a special architecture, originally introduced by LeCun et al. in 1989 [70]. Convolutional neural networks are particularly useful when dealing with image data, since their spatial structure is very similar to the structure for image recognition of the human brain [71]. This section will go into detail on convolutional neural networks. Theoretical considerations are demonstrated on the problem of classifiying the dog or cat breed of a pet shown in an image. The Oxford-IIIT Pet dataset [72], featuring 12 cat breeds and 25 dog breeds, is used. Most CNNs for image classification share roughly the same basic architecture. The input is usually fed through a few convolutional layers to do

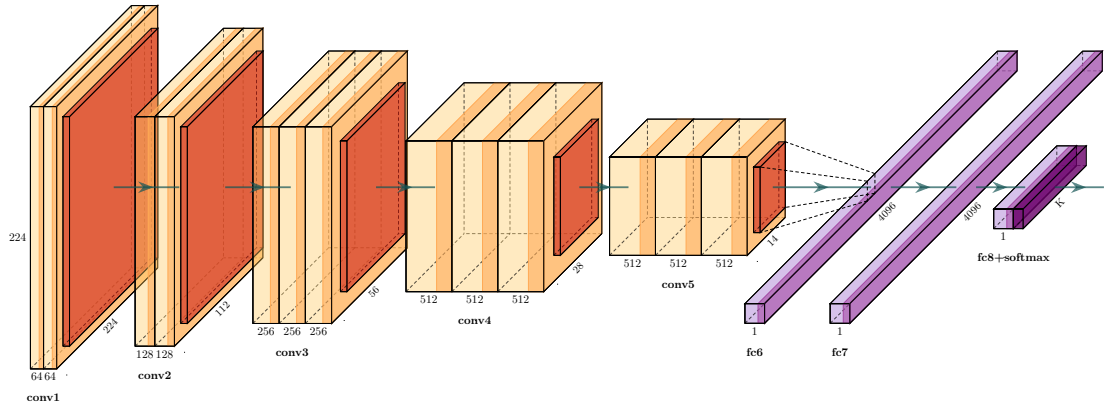


Figure 3.1: VGG-16 architecture shows the a common architecture for designing CNNs for image classification [73].

feature extraction. The output of the last convolutional layers is then flattened and connected to one or more fully-connected layers performing classification on the extracted features. Such an architecture is seen in figure 3.1.

In computer vision, typically 2D convolutional layers are used. A convolutional layer applies the convolution operation to its input and a trainable kernel. Such a kernel $k \in \mathbb{R}^{k \times k}$ is a matrix of size $k \times k$. A convolution is a mathematical operation on two functions f and g and defined as

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

in its general continous form. In CNNs, the discrete version

$$(f \star g)(n) = \sum_{m=-\infty}^{\infty} f(m) g(n - m)$$

is used. Figure 3.2 shows a convolution operation, as computed by a convolutional layer. The kernel $K \in \mathbb{R}^{3 \times 3}$ is placed on the input matrix $X \in \mathbb{R}^{4 \times 4}$ and the dot product

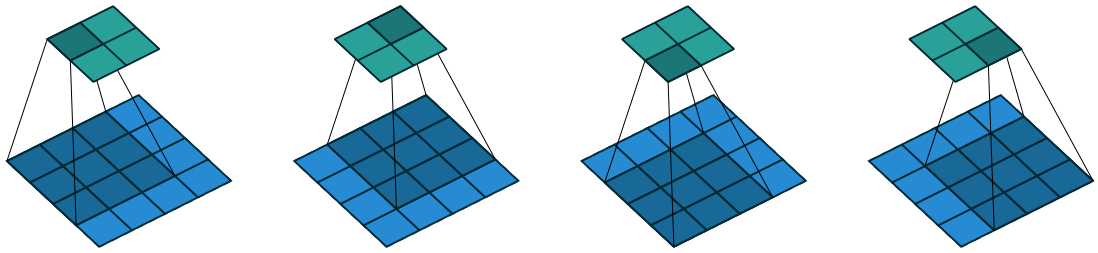
$$y_{lm} = \sum_{i=1}^{3} \sum_{j=1}^{3} x_{ij} k_{ij}$$

between the kernel and the corresponding part of the input image is computed. Then the kernel is shifted one step further. This is repeated until the kernel has evaluated the complete input image. The complete operation can be written in terms of a discrete two dimensional convolution (or more specifically: a cross-correlation operation, since the kernel is not flipped here)

$$Y(l, m) = (K \star X)(l, m) = \sum_{i} \sum_{j} X(l + i, m + j) K(i, j),$$

or, complying with the notation of equation 3.1

$$z = K \star X.$$



(a) (b) (c) (d)

Figure 3.2: Convolutional kernel operating on an image without padding and without strides. The kernel calculates an inner product on its weights and the receptive field on the input image [74].

The resulting matrix $Y \in \mathbb{R}^{2\times2}$, the so-called feature map, contains the results of each individual inner product, where each neuron preserves the spatial relationship to the corresponding entries in the input image, this is also called the receptive field of the neuron. In this specific example, no input padding is used and the stride $s = 1$, meaning the kernel is shifted 1 step further, and importantly the depth of the resulting is 1. Often multiple neurons share the same receptive field, thus the resulting tensor is of rank 3, each depth dimension representing a different filter. These operations can be completely parallelized and optimized for GPU processing. Since different regions of the input image share the same kernel, convolutional layers are invariant under translation. While the weight matrices $W \in \mathbb{R}^{d\times o}$ of fully connected layers have $d \cdot o$ parameters, convolutional kernels only have k^2 parameters, making them more lightweight to train, since in general

$$k \ll d,\ k \ll o.$$

The resulting feature maps are in general multi-dimensional, with each dimension encoding a specific feature. An example of different feature maps in a CNN can be seen in figure 3.3.

Another important technique in CNNs is called pooling. A pooling layer is usually used for dimensionality reduction and extracts a certain value from a region on an input filter. E.g. max pooling extracts the highest value from the given input region, while average pooling computes the average in that region. The max pooling operation can be seen in figure 3.4.

For classification, a softmax activation function

$$\sigma(z)_i = \frac{\exp^{z_i}}{\sum_{j=1}^{K} \exp^{z_j}}$$

is commonly used on the last layer. The number of neurons in the last layer typically corresponds to the number of output classes. Therefore, applying softmax acitivation to this layer yields a vector where the value of element i corresponds to the probabilty of the input image belonging to class i. The categorical cross-entropy loss function

$$\ell(x, y) = -\log f_\theta(x)_i[c = i]$$

where $c \in C$ is the correct output class, can then be used as a suitable loss function.

Deep learning models generally tend to overfit to the training data, since such models have a high capacity for just memorizing the training data, instead of learning the underlying patterns and therefore generalizing to unseen data. Regularization in different forms is used to reduce

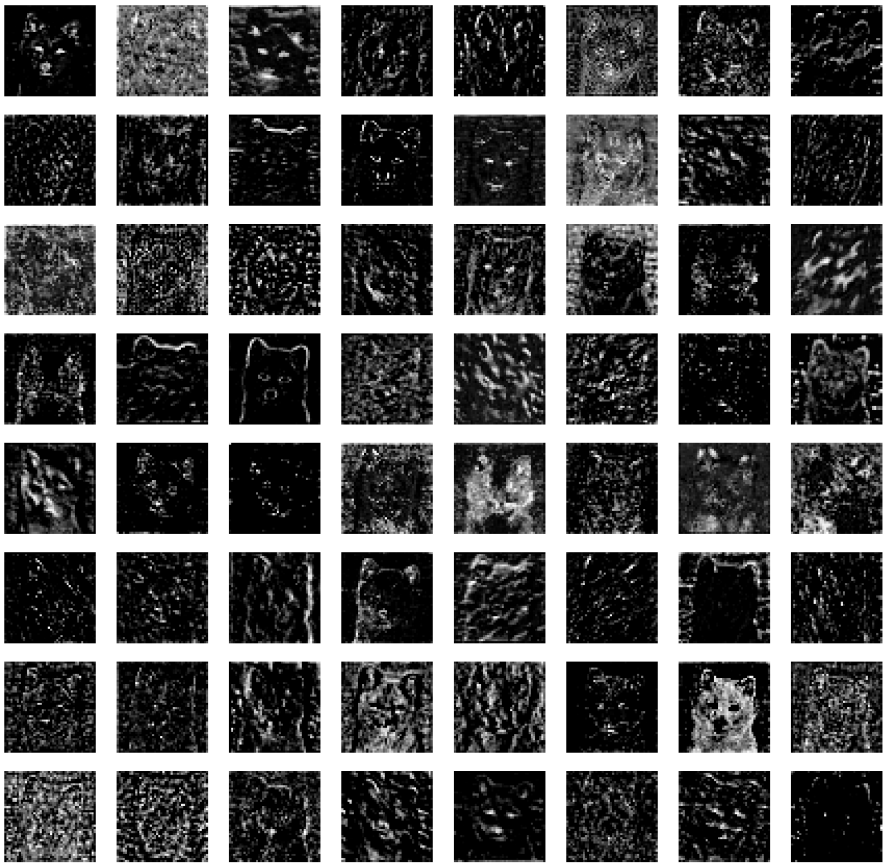


Figure 3.3: Feature maps of a CNN.

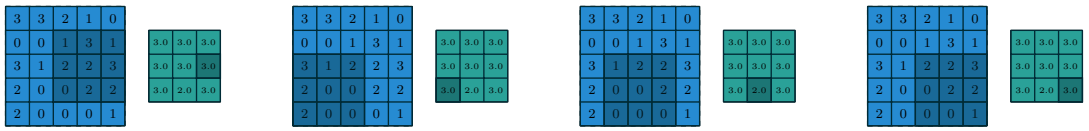


Figure 3.4: Max pooling operation for dimensionality reduction on an input filter. The input filter has a shape of 5×5 while the output filter shape is 3×3 [74].

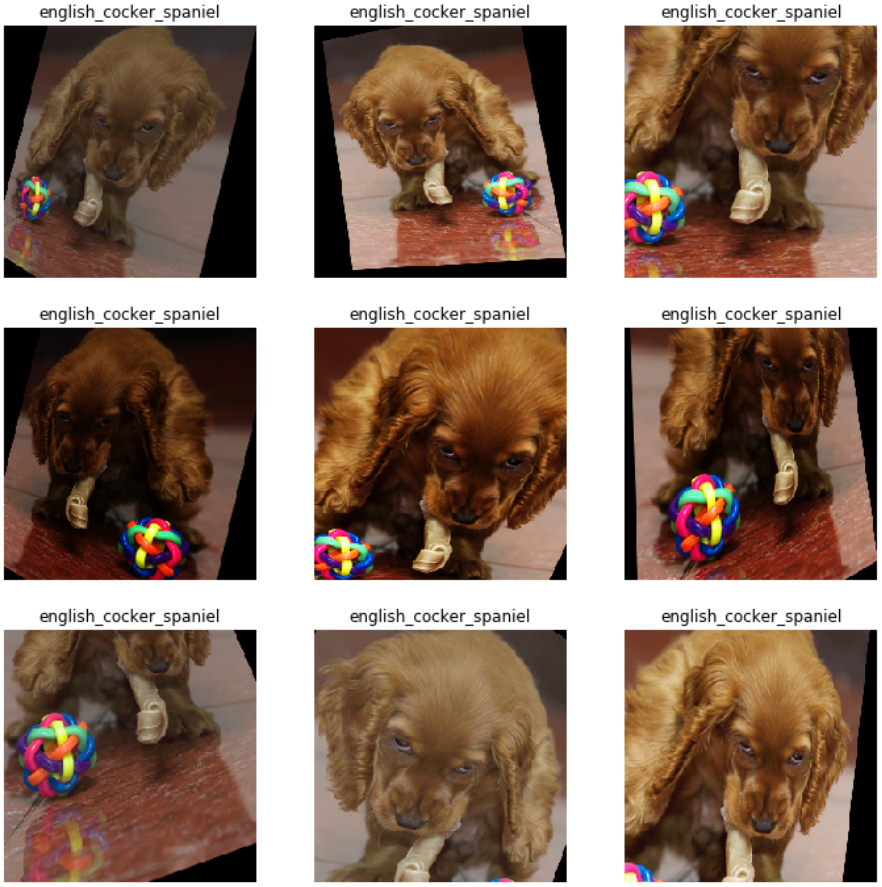


Figure 3.5: Regularization by data augmentation. Each individual image is randomly transformed.

overfitting. An important parameter for regularization is the network size. Although in practice, most models are trained to overfit to the data, balancing the model to be rich enough to explain the underlying structure in data, and simple enough to avoid overfitting, is desirable [75].

Notably, another interesting implementation of regularization in image models is data augmentation: changing the input images by rotating, flipping, skewing or changing lighting conditions. Figure 3.5 shows an example of applying augmentating transforms to an input image. Normally, such transforms are applied automatically at random when a new image is loaded from the disk into a batch.

Convolutional neural networks trained on ImageNet are biased towards texture [76] and instead of really generalizing to shapes of unseen objects, they tend to just memorize images very well. There is the possibility that k-nearest neighbors would perform equally well on the ImageNet

classification dataset. However, evaluating k-nearest neighbors on ImageNet would be too computationally expensive, while inference with CNNs has become really fast, due to optimized operations and optimized hardware.

3.1.3 ResNet

Optimizing functions given by neural networks with stochastic gradient descent generally only converges to a local minimum. However, the propability of ending in a ”bad local minimum”, that is, the function value of this local minimum is substantially higher than the function value of the global minimum, decreases with the number of layers [77]. Realizing very deep networks with standard CNN architectures like VGG [73], as seen in figure 3.1 leads to a degradation in accuracy. Even, if it should theoretically be possible for the deeper layers to just learn the identity mapping, deeper models performe worse than shallower ones. Residual deep learning [78] solves this problems by using residual blocks. Such a residual block realizes the function

$$\mathcal{F}(x) + x$$

by introducing a shortcut connection in the neural network. An illustration of this is seen in figure 3.6. Briefly, the idea behind deep residual learning is: it is easier to optimize a neural network with such shortcut connections than without. If an identity mapping was optimal, the activation of the nonlinear layers is easier pushed to zero than learning an identity mapping with non-linear layers. Figure 3.7 shows the results of using ResNets compared to using plain CNNs. The plot depicts the absolute improvement in performance of using deeper ResNet models on the ImageNet dataset, as well as the possibility to train deeper models. Plain CNNs perform worse when using a deeper architecture. Due to their possibility to realize really deep networks, ResNets are widely used as a standard architecture for feature extraction.

However, training a ResNet from scratch, that is, starting at random weights, is a time-consuming task, especially on consumer hardware. A ResNet-50 for example has 23 million trainable parameters. Fortunately, deep learning models, which are previously trained on another dataset, can be used as a starting point for training on a novel dataset. This procedure is referred to as *transfer learning*. Using transfer learning, a ResNet, pretrained on the ImageNet dataset, is trained on a new image classification problem in only a few minutes, even with a very small dataset. The idea behind is that the features, the model has learned on the large ImageNet dataset, are also useful for the new problem. The pretrained ResNet is used as a backbone and only the fully connected layers, the head of the network, are trained.

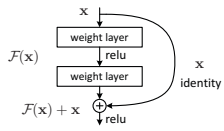


Figure 3.6: A building block of a ResNet [78].

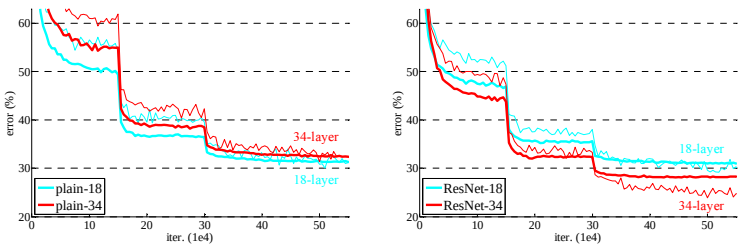
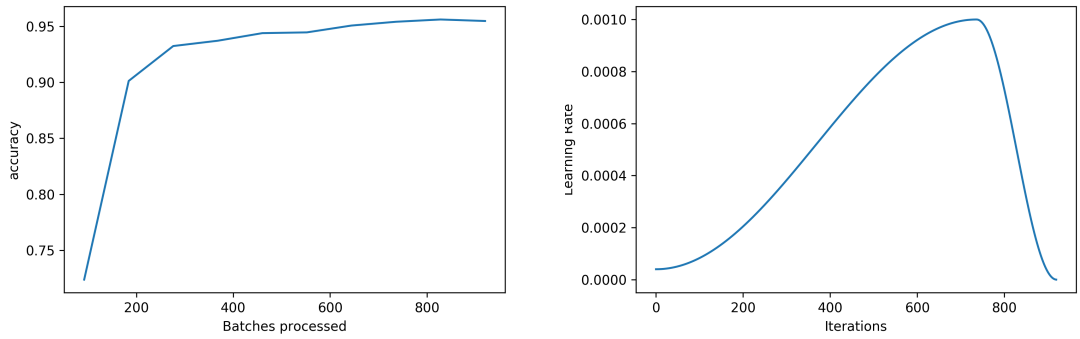


Figure 3.7: Error on the imagenet dataset when using ResNets [78].



(a) **(b)**

Figure 3.8: Training of a deep neural network on a small dataset for about 10 minutes. a) shows the accuracy over training time, b) shows the learning rates used. The One-Cycle Policy first increases the learning rate to a maximum and then decreases again. This keeps the network from overfitting.

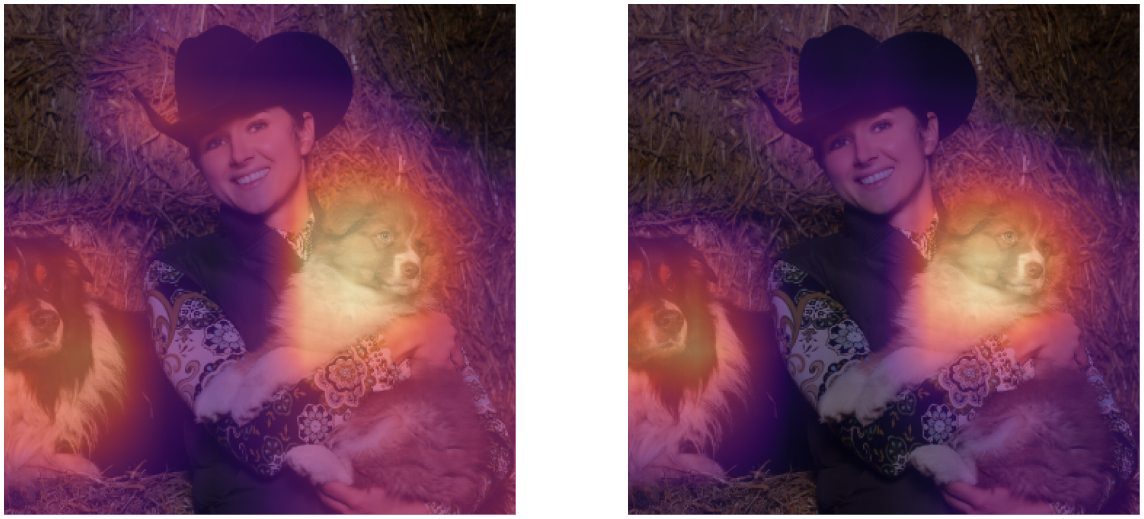
In deep learning models, feature extraction is done in a hierarchical manner. Deeper layers extract low-level features, while shallower layers extract high-level features. In transfer learning it is sometimes useful to also train the feature extractor on the new problem. However, low level features are highly likely to be useful, hence should not be destroyed by the new training process. This can be realized by setting a different learning rate for each layer, starting at very small learning rates on the deep layers and increasing the learning rate with reducing depth of the layer in the network.

Training a ImageNet-pretrained ResNet-50 on the Oxford-IIIT Pet dataset [72] takes around 10 minutes and results in a model with a state-of-the-art accuracy of 0.95. This exceeds the results of the models presented in the original paper of the dataset and shows that deep learning models can be trained quickly and on a relatively small dataset (approx. 5900 images). However, the original results were reported in 2012 and deep learning models have since come a long way. The NN is trained for one cycle of 10 epochs with a maximum learning rate of 10^{-6} for the deepest layer and 10^{-3} for the most shallow layer. The learning rate for layers in between is linearly interpolated between these two numbers. Code for training this model is found on github[4]. Learning rate scheduling and training process can be seen in figure 3.8.

Figure 3.9 shows the activations of the ResNet feature extractor, before and after training on the new dataset, plotted onto an unseen input image. Before training, the features focus the dogs as well as the human. After training, the focus is mainly on the dogs, as expected by training on a pet dataset.

Recent research suggests that overall performance gains not only relies on scaling up the depth of the model, but balancing network depth, resolution and width [79].

[4] https://github.com/marctuscher/cnn/blob/master/pets.ipynb



(a) (b)

Figure 3.9: Activation heatmap on an unseen input image before, and after training. a) shows the activation heatmap before training on the specialized dataset. The ImageNet features tend to focus the dog in the foreground, but the human is highlighted, too. b) shows that training on the specific dataset leads the network to focus on the pets.

3.1.4 CNNs for Object Detection

Object detection extends the problem of image classification. In this setting, the problem can be defined as: Given an image, what objects does it contain and what are their locations? Naturally, models pretrained for classification, can be used for transfer learning to object detection. Even if kNN would perform equally well on Image classification, using kNN for object detection would be much more complicated. CNNs shine in object detection due to their spatial structure of processing images. In the simplest form, a location of an object is a 2D bounding box as seen in figure 3.11. However, more advanced approaches also tackle the problem of instance segmentation. These approaches not only return a class score and a bounding box, but also a binary pixel mask, describing all pixels of the object in the image (also seen in fig. 3.11). Object detection methods relying on Convolutional Neural Networks have made great progress in advancing the state-of-the-art in object detection. Since the CNN used for tracking in this work is based on architectures for object detection, this section gives a brief overview on object detection methods.

Region-based CNN

Object detection requires localization of (potentially many) objects within an image. Natural approaches to this are to either construct this problem as a regression problem [80] or use a sliding-window approach [81]. However, the object tracking model used in this work relies on region-based CNNs. The first approach to introduce region-based neural networks is called R-CNN [82]. At inference time, R-CNN generates around 2000 region proposals, extracts a feature vector from each proposal using a CNN and then classifies each region with one linear SVM per class. Region proposals are also called regions of interest (RoI). R-CNN is agnostic to the method used for region proposal. Each region proposal is preprocessed using affine image warping to

compute a fixed size input for the CNN. Since inference and training is tedious with a model that is a mix of ConvNets and SVM, Fast R-CNN [83] proposes an architecture consisting only of neural network layers and is therefore faster in training and inference. A max-pooling layer extracts a small size feature map from each RoI, this is also called RoI pooling. Each RoI is fed into a series of fully-connected layers with two heads, one of length $|C|$ predicting class scores and another fully-connected layer of length $4 \cdot |C|$ predicting a bounding box for each class. Bounding boxes on images are commonly parametrized by the coordinates x, y of either the top left corner or the center, combined with the width and height w, h of the bounding box.

Fully Convolutional Networks

Methodologically Fully Convolutional Networks [84] are CNNs built only with convolutional layers. Given an input image, instead of predicting a vector, these models predict a tensor of rank 3, typically the output of a convolutional layer, corresponding to the desired spatial result. Single shot detectors like SSD [85] and YOLO [86] use fully convolutional networks to predict class scores for a predefined set of anchor boxes and regress the bilinear interpolation between the anchor boxes. These operations are performed by the neurons sharing receptive fields. Regression and classification can then be done for each anchor box separately without the necessity to have a fully connected layer for each individual anchor box. Anchor boxes are defined on multiple stages of the downsampling path of the network, to find objects on many different scales.

Region Proposal Networks

The efficiency of region-based object detection networks is mainly bottlenecked by the computation of region proposals. Fast R-CNN utilizes the GPU for proposal evaluation, but most region proposal strategies are executed on the CPU. Region Proposal Networks (RPN) are fully convolutional networks that take an image as input and output a set of rectangular regions along with a score, which indicates if the region contains an object or background. They are first introduced in Faster R-CNN [87]. Similar to single shot detectors, Faster R-CNN defines anchor boxes with different scales. A Region Proposal Network outputs a fixed number of proposals. Each proposal is defined by a set of anchor box offsets and an objectness score, stating the probability of an object being contained in the corresponding bounding box. Such a bounding box is retrieved by adding the anchor box offsets to the base coordinates of the corresponding anchor boxes. Figure 3.10 shows the top 50 of 2000 region proposals by an RPN for a single image. The main difference between models relying on RPN and single shot detectors is that single shot detectors classify the output class and regress the final location directly on the region proposals. R-CNN models combine multiple proposed regions by non-maxima suppresion and a further neural network head classifies and extracts the location of the object contained in the region. Features are shared for both operations, region proposal and final classification and bounding box regression. While single shot detectors are faster in general, R-CNN models tend to be more accurate [88].

MaskRCNN

Mask R-CNN [89] is an extension to the general R-CNN architecture, able to not only predict a bounding box for an object, but also a segmentation mask. Conceptually, the method extends the output of Faster R-CNN for object classification and localization by a third branch that generates a binary mask for the object. The output of the mask branch encodes $|C|$ binary masks of resolution $m \times m$ for each RoI. Since the original RoIPooling layer subdivides each RoI into spatial bins, and therefore introdues misalignments between the RoI and the extracted features, making it hard to find exact pixel-wise binary maps. Mask R-CNN introduces a RoIAlign layer, where exact values of the input features are computed using bilinear interpolation. Figure 3.11 shows an example prediction of a Mask R-CNN model. For this example, a pretrained model contained in the detectron2-library is used. The code is also found on github[5]. Combining Mask R-CNN with a Feature Pyramid Network (FPN) [90] instead of a baseline ResNet for feature extraction leads to even better performance. Early approaches to object detection use a pyramid of images to detect differently scaled objects [91]–[93]. Similarily a FPN extracts a pyramid of features from different levels in the downsampling path of the backbone. Note the difference between different sized anchor boxes in RPNs and features from different levels in FPNs. Bringing both together results in the highest accuracy. All feature maps of the pyramid are then used for region proposal generation as well as bounding box regression and object classification. In fact, the results in figure 3.11

Figure 3.10: Top 30 region proposals generated by a Region Proposal Network.

[5] https://github.com/marctuscher/cnn/blob/master/object_detection.ipynb

Figure 3.11: Predictions of a Mask R-CNN model. Note particularly how the saddle is left out in both masks, of the rider and the horse.

are generated using a FPN as backbone. The Faster/Mask R-CNN architecture can be seen as a generalized architecture for object detection which is also extended to meta-learning [94], thus enabling to learn a powerful model with only a few images.

3.1.5 Siamese Networks for One-Shot Learning

Learning classifiers for visual classification of objects usually requires a vast amount of training examples from each class. One-Shot Learning models are able to correctly classify an image relying on only a single training example [95]. Addressing One-Shot learning by developing domain-specific features can be promising for specific instances of the problem, but fails in generalizing across multiple domains. The deep learning framework provides a possibility to learn features, enabling models to generalize across a wide area of domains. Furthermore, similar to transfer learning, Convolutional Neural Networks are trained to classify images in a supervised setting using a large dataset and directly reused for One-Shot Learning [96]. While the initial training of such classifiers imposes considerable computationally costs, no further training is needed to tackle One-Shot Learning with the framework deep learning provides. Siamese Networks, originally invented by Jane Bromley and Yann LeCun et. al. [97] apply deep learning to the problem of One-Shot Learning. In general, the architecture of a Siamese Network (or Twin network) consists of two similar feature extractors sharing exactly the same weights, followed by a loss function computing any distance metric on the extracted features. When presented by two extremely similar inputs, the feature extractors cannot possibly map to different locations in feature space

because both networks compute the same function. Further, the network architecture is symmetric, that is, swapping the inputs has no impact on the result [96]. Applications of Siamese Networks include face verification [98], [99], geo image matching [100] and stereo matching [101].

3.1.6 SiamMask

This section explains the SiamMask [102] architecture and its characteristics. SiamMask is a template-matching method based on siamese networks and an important part of the tracking method proposed in this work. It combines object tracking with visual object segmentation. Object tracking is done using a fully-convolutional siamese network. SiamMask is agnostic to the choice of specific architecture of siamese network, in fact, in [102] training is run using SiamFC [51] and SiamRPN [52], which advances SiamFC by using a Region Proposal Network. However, this section focuses on SiamRPN as underlying architecture, due to its high performance. Similar to most template-matching algorithms for object tracking, siamese networks search an object contained in a template image in the current frame of a video sequence. The template image is retrieved by drawing a bounding box around the target object in the first frame of a video sequence. Since searching on the complete frame would be too expensive at inference time, the frame is often cropped, centering at the location of the target object in the previous frame. Specifically siamese networks use the same CNN f_θ to process both images. Let z be a small template image and x be a crop of the current frame. Both inputs are fed into the same CNN f_θ, yielding two cross-corellated feature maps

$$g_\theta(z,\ x) = f_\theta(z) \star f_\theta(x). \tag{3.3}$$

Note that this function is vector-valued, that is,

$$g^n_\theta(z,\ x)$$

encodes a set of k anchor box proposals and corresponding object/background scores, the common output of RPNs. Unlike Faster/Mask R-CNN where object detection is a two stage process, SiamMask directly regresses localization of the target object in the RPN, hence, also exposing a relationship to SSDs. $g^n_\theta(z,\ x)$ is called a response of a candidate window (**RoW**). Instead of directly computing cross-correlation (eq. 3.3) between input features and template features, another convolutional block is used to reduce the number of features from each residual block. This is called depthwise cross correlation [53]. SiamMask extends SiamRPN by producing a pixel-wise binary mask: an extra branch is added to the existing architecture, this can be seen in figure 3.12.

For each **RoW** , a $w \times h$ binary mask is predicted by a two layer network e_ϕ

$$m_n = e_\phi \left(g^n_\theta(z, x)\right),$$

this way z is used to guide the segmentation process. Segmentation is further improved by multiple refinement modules. These modules merge low and high resolution features from the ResNet-50 backbone, as to be seen in figure 3.13.

Training SiamMask is trained using a ResNet-50 until the final convolutional layer of the 4-th stage, pretrained on ImageNet, as backbone for feature extraction. Using an additional convolutional layer, the output stride is reduced from 8 to 1. This results in a higher spatial resolution. In order to increase the receptive field of the neurons, dilated convolutions (fig. 3.14) are used. During training, each **RoW** is labelled with a binary label $l^n \in \{\pm 1\}$ and associated with a mask c^n of size $w \times h$. Categorical cross entropy

$$\ell_{cls}(z,\ x) = -\log g^n_\theta(z,\ x) l^n. \tag{3.4}$$

is used as a loss function for object/background classification. A **RoW** is considered positive, $y_n = 1$ if one of its anchor boxes has an IoU with the ground truth box of at least 0.6, $y_n = -1$ otherwise. Let A_x, A_y, A_w, A_h be the center and size of the proposal boxes and T_x, T_y, T_w, T_h be the corrdinates of the ground truth bounding boxes. The normalized distance is retrieved by

$$\delta_1 = \frac{T_x - A_x}{A_w},\ \delta_2 = \frac{T_y - A_y}{A_h}$$
$$\delta_3 = \ln \frac{T_w}{A_w}, \delta_4 = \ln \frac{T_h}{A_h}.$$

The robust L1 loss can be written as

$$\ell_{smooth}(x,\ \sigma) = \begin{cases} 0.5\sigma^2 x^2 & \text{if } |x| < \frac{1}{\sigma^2} \\ |x| - \frac{1}{2\sigma^2} & \text{otherwise,} \end{cases}$$

which is less sensitive to outliers than the MSE loss. The loss function for bounding box regression is then defined as

$$\ell_{reg}(A) = \sum_{i=1}^{4} \ell_{smooth}(\delta_i, \sigma). \tag{3.5}$$

The mask loss function

$$\ell_{mask}(z,\ x) = \sum_n \left[\frac{1 + l^n}{2wh} \sum_{ij} \log(1 + e^{-c^n_{ij} m^n_{ij}}) \right] \tag{3.6}$$

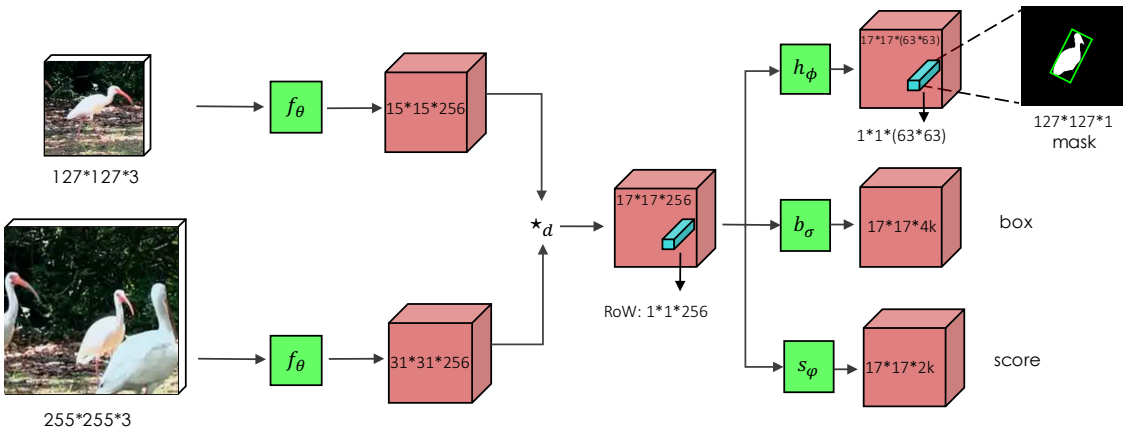


Figure 3.12: SiamMask architecture. Template and observation frame are fed through the same feature extractor, followd by a convolutional block, and the cross-correlation operation [102].

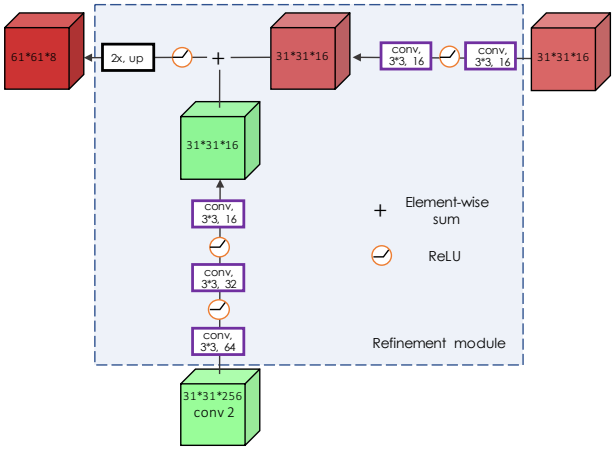


Figure 3.13: Mask refinement modules [102].

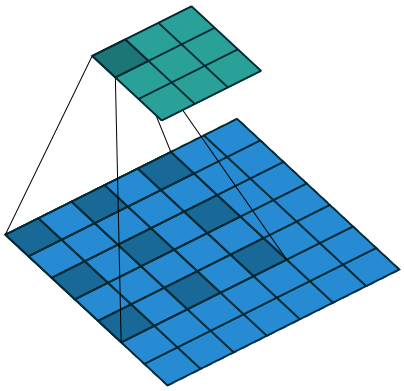


Figure 3.14: Dilated convolution increases the recpetive field [74].

defines a loss function for the pixel-wise prediction and the ground truth mask $c_{ij}^{n} \in \{\pm 1\}$.

Training is done via the multi-task loss function

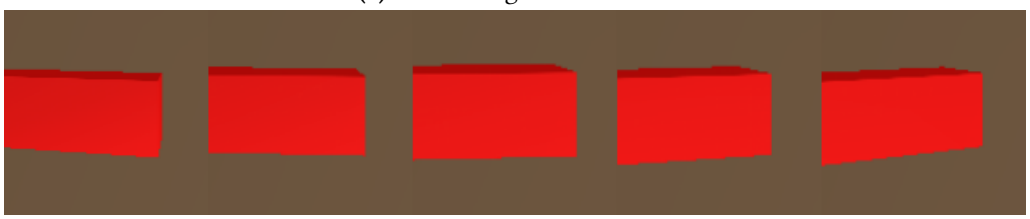
$$\mathcal{L}_{siam} = \lambda_1 \cdot \mathcal{L}_{mask} + \lambda_2 \cdot \mathcal{L}_{cls} + \lambda_3 \cdot \mathcal{L}_{reg}$$

where $\lambda_1 = 32$, $\lambda_2 = \lambda_3 = 1$.

3.1.7 Tracking Holistic Object Representations (THOR)

Briefly, THOR [22] is a template memory consisting of short-term templates and long-term templates which can be used in conjuction with any template-matching tracking method. The template memory is composed of a **long-term module (LTM)** and a **short-term module (STM)**. The LTM represents the target object in the most diverse conditions encountered during the current tracking process, as seen in figure 5.6d.

(a) THOR long-term module.



(b) THOR short-term module.

Long-Term Module For the LTM, the goal is to find templates which maximize the information about the target object. However, only a limited number K_{lt} of templates can be kept, due to memory limitations. Since a 2D image of an object is only a projection of a 3D object into 2D space, only limited object descriptions are possible. Therefore describing objects with visual features, is a practical alternative. THOR works with every template-matching method describing visual features, which uses an inner product for similartiy computation. In this work, THOR is used alongside a siamese network for object tracking. The n-dimensional visual feature space, together with the convolution operator, form an inner product space. Maximizing the volume $\Gamma(z_1, \ldots, z_m)$ of the parallelotope formed by the feature vectors z_i of template T_i leads to maximizing the distance in similarity of all feature vectors. The template kernel z_i is applied to the input image during tracking, so measuring how similar two templates T_i and T_j are, boils down to calculating $z_i \star z_j$. For kernels $z_1, \ldots, z_m$ the Gram matrix

$$G(z_1, \cdots, z_m) = \begin{bmatrix} z_1 \star z_1 & z_1 \star z_2 & \cdots & z_1 \star z_m \\ \vdots & \vdots & \ddots & \vdots \\ z_m \star z_1 & z_m \star z_2 & \cdots & z_m \star z_m \end{bmatrix}$$

is constructed. Since G is symmetric, the determinant of G is the squared volumed Γ of the parallelotope constructed by $z_1, \ldots, z_m$ and therefore maximizing $|G|$ is proportional to maximzing Γ. Hence, the objective becomes

$$\max_{z_1, z_2, \ldots, z_m} \Gamma(z_1, \ldots, z_m) \propto \max_{z_1, z_2, \ldots, z_m} |G(z_1, z_2, \ldots, z_m)|.$$

Note that T_1 is the ground truth template seen when tracking is initialized. This template is always kept in memory to reduce tracking drift, that is, the tracker gradually moving away from the target object to other objects in the scene. Tracking drift can also lead to adding templates containing other objects than the target object to the LTM, since these are likely to increase Γ. Setting an upper bound on $|G|$ would solve this problem, yet, such an upper bound is not easy to find. Therefore a new template T_c is only added to the LTM if storing it maximizes G and its features z_c satisfy

$$z_c \star f_1 > l \cdot G_{11}, \tag{3.7}$$

which can be seen as a lower bound on the similarity to the ground truth template T_1. The parameter l can be used to trade-off tracking performance against drifting robustness. [54] describe two methods for calculating a lower bound, if the static lower bound in eq. 3.7 is to conservative: a *dynamic* lower bound and an *ensemble* lower bound. The dynamic lower bound takes short-term appearance of the object into account by subtracting a diversity measure γ given by the short-term memory. The lower bound criteria becomes

$$z_c \star f_1 > l \cdot G_{11} - \gamma. \tag{3.8}$$

Computing the lower bound using the ensemble lower bound is done by using all templates in the LT module:

$$z_c \star f_1 > l \cdot \operatorname{diag}(G_{11}) \tag{3.9}$$

which is satisfied by a new template T_c if it satisfies all inequalities in eq. 3.9.

Short-Term Module Since templates created in situations of partial occlusion or abrupt movements are too dissimilar to the base template and therefore do not satisfy a lower bound condition, these situations are handled by the short-term module. Templates are processed by the short-term module in a FIFO-manner, and the STM has a fixed number K_{st} of placeholders for templates. The STM also stores a Gram-Matrix G_{st} similar to the long-term Gram-Matrix G. It is used to compute a diversity measure

$$\gamma = 1 - \frac{2}{N(N+1)G_{st,max}} \sum_{i<j}^{N} G_{st,ij}$$

of all templates in the STM. The closer γ is to 1, the more diverse are templates in the STM.

Inference This section describes how inference is done using the THOR template modules. Two strategies are used during inference: *modulation* and the *ST-LT switch*. For every frame, activation maps of every template in both STM and LTM are computed in a batch operation manner. Predictions of all templates are used by computing a weighted spatial average over all activation maps: if a template is more certain, the weight, depending on the maximum score is higher and it contributes more to the final activation map. The ST-LT switch helps to avoid tracking drift compared to using only the STM for prediction. If the IoU between a prediction bounding box in the LTM and a prediction in the STM is lower than a threshold t_{iou}, the prediction of the LTM is used and the STM is reinitiallized by being filled with the specific template of the LTM.

3.2 Iterative Closest Point

This section follows [23] in explaining the iterative closest point algorithm for matching two scans of a scene. Such a scan is often retrieved in form of a pointcloud. A pointcloud

$$\mathcal{P} = \{x_i\}_{i=1}^{n}$$

is a set of n points x_i, each characterizing a set of features of that point. These features can be position, color, normals or curvature of the pointcloud at this specific point. For simplicity only points describing a position are considered in this section, such that

$$x_i \in \mathbb{R}^3.$$

In short, given two pointclouds the ICP algorithm iterates two steps:

1. compute corresponding points between the two pointclouds
2. compute a transformation which minimizes the distance between corresponding points.

These steps are repeated until the difference between consecutive transformations is sufficiently small. During this process only corresponding points within a certain range d_{max} are taken into account. Algorithm 3.1 shows the ICP procedure. In line **??** point correspondences are computed using the euclidean norm. This simple and fast distance metric allows the use of kd-trees to speed up the look up of the closest point.

Algorithm 3.1 Standard Iterative Closest Point

Input: Two pointclouds: $A = \{a_i\}_{i=1}^{n}$, $B = \{b_i\}_{i=1}^{n}$, an initial transformation: T_0
Output: The transformation T, which aligns A and B

1: $T = T_0$ // initial transform as guess
2: **while** not converged **do**
3: **for** $i \leftarrow 1$ **to** n **do**
4: $m_i \leftarrow$ `findClosestPointInA`$(T\dot{b}_i)$
5: **if** $\|m_i - T\dot{b}_i\| \leq d_{max}$ **then**
6: $w_i \leftarrow 1$
7: **else**
8: $w_i \leftarrow 0$
9: **end if**
10: **end for**
11: $T \leftarrow \arg\min_T \sum_i w_i \left\|T\dot{b}_i - m_i\right\|^2$
12: **end while**

Point-to-plane ICP A more robust and accurate extension of this algorithm is the so-called point-to-plane ICP [57]. Instead of minimizing the sum of distances of each corresponding pair of points, as it is done in line 11, the point-to-plane ICP minimizes the error of the projection of $(T\dot{b}_i - m_i)$ onto the sub-space spanned by the surface normal η_i at m_i. Line 11 is changed to

$$T \leftarrow \arg\min_T \sum_i w_i \left\|\eta_i(T\dot{b}_i - m_i)\right\|^2 .$$

Generalized-ICP The Generalized-ICP [23] also only changes this exact line to make use of a probabilistic model in the minimization step, while leaving the rest of the algorithm unchanged. For to purpose of better readability it is assumed that all points of pointclouds $A = \{a_i\}_{i=1,\ldots,n}$ and $B = \{b_i\}_{i=1,\ldots,n}$ with correspondences with $\|m_i - T\dot{b}_i\| > d_{max}$ are removed. Furthermore, the pointclouds are ordered in the sense that $\{a_i\}$ is the point in A that corresnponds to $\{b_i\}$ in B. Probabilistc modelling of the minimization step of the ICP is done by assuming that A and B are generated according to $a_i \sim \mathcal{N}(\hat{a}_i, C_i^A)$ and $b_i \sim \mathcal{N}(\hat{b}_i, C_i^B)$ with underlying points $\hat{A} = \{\hat{a}_i\}$ and $\hat{B} = \{\hat{b}_i\}$. Here $\{C_i^A\}$ and $\{C_i^B\}$ are the covariance matrices associated with the measured ground truth points. The correct transformation T^* to be found by ICP gives

$$\hat{b}_i = \mathrm{T}^* \hat{a}_i \tag{3.10}$$

assuming perfect point correspondences. This also yields the distance between two corresponding points under an abitrary rigid transformation T

$$d_i^{(T)} = b_i - \mathrm{T}a_i .$$

Since a_i and b_i are assumed to be drawn from independent Gaussians the distribution from which $d_i^{\mathrm{T}^*}$ is drawn can be written as

$$\begin{aligned} d_i^{\mathrm{T}^*} &\sim \mathcal{N}\left(\hat{b}_i - (\mathrm{T}^*)\hat{a}_i, C_i^B + (\mathrm{T}^*)C_i^A(\mathrm{T}^*)^T\right) \\ &= \mathcal{N}\left(0, C_i^B + (\mathrm{T}^*)C_i^A(\mathrm{T}^*)^T\right) \end{aligned}$$

by applying eq. 3.10.

T is computed by using Maximum Likelihood Estimation by setting

$$T = \arg\max_T \prod_i p\left(d_i^{(\mathrm{T})}\right) = \arg\max_T \sum_i \log\left(p(d_i^{(\mathrm{T})})\right)$$

which can be simplified to

$$T = \arg\min_T \sum_i d_i^{(\mathrm{T})^T} (C_i^B + \mathrm{T} C_i^A \mathrm{T}^T)^{-1} d_i^{(\mathrm{T})}. \tag{3.11}$$

Setting

$$C_i^B = I$$
$$C_i^A = 0$$

yields the standard ICP update rule, which is a special case of equation 3.11. Relationship to the point-to-plane ICP can be shown by also modelling the point-to-plane ICP probabilistically. The update step in the point-to-plane ICP can be defined as

$$T = \arg\min_T \sum_i ||P_i \cdot d_i||^2,$$

where P_i is the projection onto the span of the surfance normal at b_i. The part $||P_i \cdot d_i||^2$ can be refomulated as quadratic form

$$||P_i \cdot d_i||^2 = d_i^T P_i d_i$$

since P_i is an orthogonal projection matrix. Therefore the point-to-plane ICP is the case of the Generalized-ICP where

$$C_i^B = P_i^{-1}$$
$$C_i^A = 0.$$

Every pointcloud sampled from a range-measuring sensor is a 2-manifold in 3-space, and therefore, has more structure than an arbitrary pointcloud. The point set can be assumed to be locally planar, since real-world surfaces are piece-wise differentiable. Sampling from two different perspectives, the correspondences of points will never be exact, each should only provide a constraint along its surface normal. This can be modelled by assuming that each point is distributed with high covariance along its local plane, and low covariance along its surface normal direction. A point with e_1 as surface normal then has the covariance matrix

$$C_\epsilon = \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

where ϵ is a small constant. Given μ_i and ν_i, the normal vectors at b_i and $a_i - C_i^B$ and C_i^A the can be conputed by rotating the covariance matrix that the ϵ term corresponds to the uncertainty aloung the surface normal. The covarraince matrices can then be computed by

$$C_i^B = R_{\mu_i} C_\epsilon R_{\mu_i}^T$$
$$C_i^A = R_{\nu_i} C_\epsilon R_{\nu_i}^T,$$

where each R_x represents one of the rotations which transforms the bases vector $e_1 \rightarrow x$.

4 Object Detection

Initialization of the tracking method requires an initial pose in world coordinates and a 2D bounding box in the image plane of the camera. For this purpose some of the properties of the specific targeted application are harnessed, as described in the following sections. Note that most of this initialization method was developed by Prof. Marc Toussaint and M.Sc. Danny Driess and was previously used in various projects [103].

Since the objective of this work is to manipulate objects, initially lying on a table in front of the robot, one of the cameras is installed directly above the table. Detection of objects is done via simple background subtraction on depth images. The construction is illustrated in figure 4.1.

Two tensors $B \in \mathbb{R}^{w \times h}$, $L \in \mathbb{N}^{w \times h}$, where w and h are width and height of an image, are saved. B contains information about the current background, that is, each element $b_{i,j}$ contains the last stably observed depth value in the depth images. L contains a label l_{ij} for each pixel in depth image. These labels $l_{ij} \in [0, 255]$ describe whether a pixel belongs to the background, to an object, to the robot (by comparing the real-world view to the simulation view) or is unexplained. Note that in this work no information on the location of the robot is used. Initialization of the

Figure 4.1: Setup for experiments. The RealSense camera observes the scene from the side, the Asus camera is placed directly above the table.

background subtraction is done by saving the depth of the table, without any object, into B. After this initialization, new objects on the table are detected by comparing each pixel d_{ij} of the novel depth image to the corresponding $b_{ij} \in B$ and setting the corresponding label

$$l_{ij} = \begin{cases} l_{\text{background}}, & \text{if } d_{ij} > b_{ij} - \epsilon \\ l_{\text{unexplained}}, & \text{otherwise} \end{cases}.$$

If the new value is smaller than the previous background $b_{ij} - \epsilon$, this pixel corresponds to the surface of a novel object and is labelled as an unexplained pixel. The threshold ϵ usually is set to $\epsilon = 0.01$, that is, objects with a height smaller than 1 cm are not detected. If a pixel d_{ij} is farther than the background b_{ij},

$$d_{ij} > b_{ij}$$

multiple times in a row, the new value d_{ij} is considered background

$$b_{ij} = d_{ij}.$$

Retrieving objects lying on the table is done by finding contours on the pixel mask $M \in \{0, 1\}^{w \times h}$ where

$$m_{ij} = [\![l_{ij} = l_{\text{unexplained}}]\!].$$

Finding the 2D bounding boxes, enclosing circles and centers of such an object in 2D depth image is now simple. The 2D bounding box is also used to compute a 3D bounding box, position and orientation of the object. Retrieving the object's size is done by using the 2D bounding box' width and height, and the difference between the topmost point of the object's surface and the background as depth. The center of the 2D bounding box combined with half the height of the object is directly used as position. The orientation of the object is retrieved by applying the rotation angle of the bounding box to the object's frame. Such a 3D bounding box is exact for box-like objects and approximate for other objects. Therefore, this object detection method only works correct for boxes. However, note that the method for 6-DoF tracking presented in the following sections works for arbitrary objects.

5 THOR + G-ICP

The approach, this work takes on solving the problem of tracking arbitrary objects in 6 degrees of freedom is described in the following sections. That is, given an initial description and initial pose of the target object, what are the subsequent poses as time advances. Since this approach is model-free, only information from RGBD sensors at start time is used to describe the target object. Briefly, this method works as follows: 1. In an initialization phase, 3D bounding boxes and cartesian poses of arbitrary objects on a table are detected using background subtraction in depth images from top down view (cp. chapter 4). 2. SiamMask + THOR is started to track the target object in RGB frames, an initial pointcloud is saved for 3D pose estimation via pointcloud registration. 3. SiamMask tracks the target object in new frames in real-time, a pointcloud of this observation is segmented from the depth image. The initial pointcloud is fit to the new observation to get the 6-DoF transform. Figure 5.1 shows an overview of the complete system. The system maps RGBD input alongside an initial guess (gained from the previous iteration or a kinematic map) of the target object pose to an estimate of the current object pose.

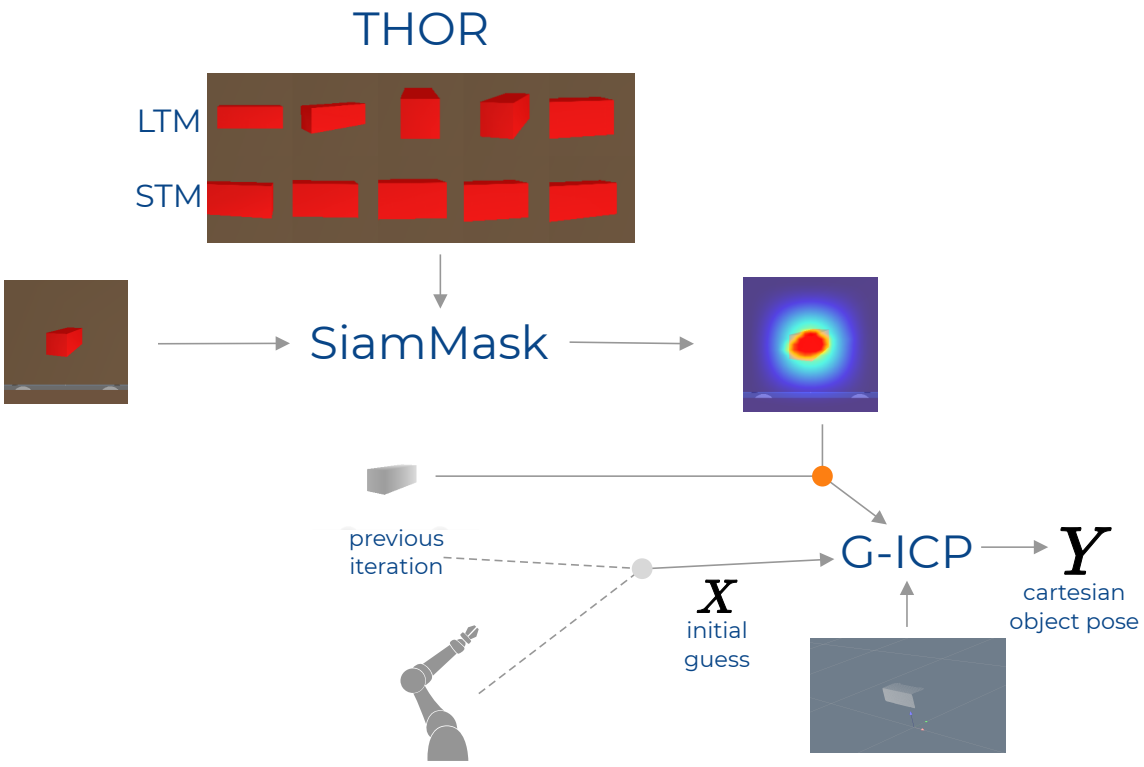


Figure 5.1: Overview of the complete system.

5.1 Tracker initialization

Once the initial size, pose and 2D bounding box of the target object is found by object detection or pose estimation, tracking is started as described in the following sections. Briefly, the goal is to gather information on the visual appearance, for RGB template-matching, and the geometric shape of the target object, for pointcloud registration.

RGB Initialization Starting RGB tracking is done by supplying an initial frame and a bounding box containing the target object to the tracker. An initial frame with a bounding box can be seen in figure 5.2 The bounding box is used to crop the frame to an initial RGB template T_{init}. The THOR long-term module (LTM)

$$T_{\text{l1}},\ \ldots,\ T_{\text{l5}} \leftarrow T_{\text{init}}$$

and short-term module (STM)

$$T_{\text{s1}},\ \ldots,\ T_{\text{s5}} \leftarrow T_{\text{init}}$$

are filled with this initial template of the object.

Pointcloud Initialization In order to find a template pointcloud of the target object, the segmentation mask from the SiamMask tracker in the first iteration of RGB tracking is used to retrieve the pixels $d_{ij} \in \mathbb{R}^{w \times h}$ in the depth image, which correspond to points on the target object's surface. Projecting the depth values d_{ij} to points $x^k \in \mathbb{R}^3$ in world coordinates using the inverse camera projection $\hat{P} \in \mathbb{R}^{3\times4}$ is done via

$$x^k = \hat{P} \cdot \begin{pmatrix} i \cdot d_{ij} & j \cdot d_{ij} & d_{ij} & 1 \end{pmatrix}^T . \tag{5.1}$$

$\hat{P}$ can be retrieved by camera calibration. The set of these points

$$\mathcal{P}_{\text{template}} = \{x^k\}_{k=1}^N$$

is the template pointcloud, which is registered to a novel observation to retrieve the transformation of the target object in world coordinates. Pointcloud registration in cartesian coordinates requires this template pointcloud to be centered around 0, therefore each individual point x^k is projected using the inverse transform of the initial object pose Y

$$\bar{x}^k \leftarrow Y^{-1} \cdot \bar{x}^k,$$

where

$$\bar{x}^k = (x_1^k,\ x_2^k,\ x_3^k,\ 1) \in \mathbb{R}^4.$$

Such a pointset (or pointcloud) belonging to the target object from figure 5.2 can be seen in figure 5.3.

5.2 Object Tracking

Object Tracking is performed as follows: first, the target object is found in the RGB frame, then a segmentation mask containing the object in RGB frame is used to find points in the depth frame corresponding to the surface of the target, these points are fit in 3D using G-ICP. Algorithm 5.1 gives an overview on the method, including the initial object detection scheme.

Figure 5.2: Initialization of RGB tracking. A frame and a rectangle, containing the target object, is supplied to the siamese network in order to start tracking.

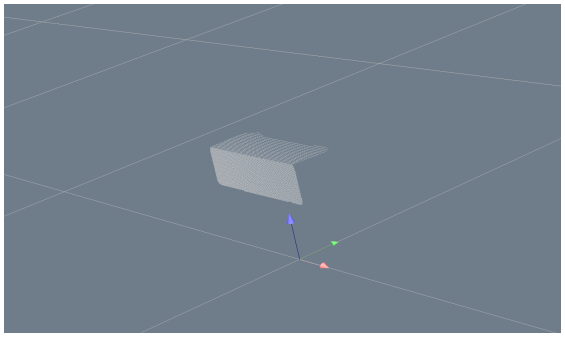


Figure 5.3: Initial pointcloud template retrieved by projecting the pixels in the depth image, belonging to the target object, to points in world coordinates.

Algorithm 5.1 Object tracking using THOR + G-ICP

```
1:  T ← ∅
2:  Find new objects O                                   // Object detection method.
3:  for o in O do                                        // For every object detected.
4:      t.pose ← o.pose
5:      t.templates ← generateThorTemplates(o)
6:      t.pointcloud ← generateTemplatePointcloud(o)
7:      T ← T ∪ t
8:  end for
9:  while not stoppped do
10:     rgb, d = cam.frames()
11:     for t in T do                                    // For every object that being tracked.
12:         boundingBox, mask ←SiamMaskTrack(rgb, t.templates, t.imgPos)
13:         observation ←getPointCloudMask(d, mask, cam.Pinv)
14:         t.pose ←icpAlign(observation, t.pointcloud, t.pose)
15:     end for
16: end while
```

RGB Tracking RGB tracking is performed according to SiamMask with THOR as template module. In every step, SiamMask relies on cropping the correct subwindow from the image to perform template matching on a smaller region of the image. Cropping the correct subwindow, depends on prior information on the target object's location and its size in the image frame.

If the robot is not moving the object, the position and size of the object in the previous frame is used. If the robot is moving the object, the kinematic map $\phi : \mathbb{R}^n \to \mathbb{R}^d$ of the robot is used to compute an estimate of the current object pose and a 3D bounding box containing the object.

A kinematic map is a function that maps joint states $q \in \mathbb{R}^n$ to any feature vector $y \in \mathbb{R}^d$

$$\phi : q \mapsto y.$$

In this setting, the maps of interest are

$$\phi^{\text{pos}}_{\text{target},v}(q) = \text{T}_{\text{W}\to\text{target}}(q)\, v \quad \in \mathbb{R}^3,$$

mapping the position of the target object $v \in \mathbb{R}^3$ to world coordinates, and

$$\phi^{\text{quat}}_{\text{target},v}(q) = \text{R}_{\text{W}\to\text{target}}(q) \qquad \in \mathbb{R}^4,$$

mapping the quaternion orientation of the target object to world coordinates.

$$\text{T}_{\text{W}\to\text{target}}(q) : \ \mathbb{R}^n \to \mathbb{R}^3$$

and

$$\text{R}_{\text{W}\to\text{target}}(q) : \ \mathbb{R}^n \to \mathbb{R}^4$$

are computed by forward chaining each individual transformation of the joints. Combining the results of both mappings yields a good initial guess on the current target object pose in cartesian coordinates.

Position and size of the target object in the current RGB frame, is fed into the RGB tracker in form of a 2D bounding box. The problem of finding such a bounding box, while the robot is manipulating the target, can be stated as: given a 3D bounding box of an object in world coordinates, what is the 2D projection on the image plane? Using the camera projection matrix P allows to project all eight corners x^W of the 3D bounding box to points p^P to the image plane

$$\begin{aligned} x^P &= P \cdot (x_1^W \ x_2^W \ x_3^W \ 1)^T \\ p^P &= \begin{pmatrix} \frac{x_1^P}{x_3^P} & \frac{x_2^P}{x_3^P} \end{pmatrix}^T \end{aligned} \qquad , \tag{5.2}$$

where $x^W \in \mathbb{R}^3$ and $p^P \in [0, w] \times [0, h]$. The corners of the target object, projected to the image plane, can be seen in blue in figure 5.4, where also the position of the target object is shown in green. A 2D bounding box containing the complete target object in image dimension, and therefore yielding the position and size in pixels, is then easily found by computing a bounding rectangle containing the polygon formed by the projected corners $\{p_i^P\}_{i=1}^8$. Such a 2D bounding box is shown in figure 5.5. Note that only the two corners with maximum distance, as seen from the camera persepective, contribute to the bounding rectangle and thus selecting the right pair of corners would bring down the number evaluations of equation 5.2 to two, and would also eliminate the need to compute a bounding rectangle on the polygon formed by the corners. However, finding the pair of corners with maximum distance in the image is not a trivial task and therefore yields a higher computational complexity.

The following paragraphs go into detail on how inference is done using SiamMask and THOR. Once the estimated position and size of the target object in the RGB frame is found, the next iteration of RGB tracking is done according to SiamMask and THOR.

Given a template T and an input x, the SiamMask model computes a cross-correlated feature map

$$g_\theta(T, \ x) = f_\theta(T) \star f_\theta(x), \tag{5.3}$$

where f_θ denotes the feature extraction network which is used to process both, the novel input and the template. Equation 5.3 yields a bounding box in image coordinates and a tracking score. Computing a segmentation mask of the target object in the RGB observation is done using a separate branch in the neural network for mask refinement. The output of this branch is a pixel-wise binary mask indicating whether a pixel belongs to the target object or not. For more details on Siamese Networks and SiamMask inference refer to chapter 3.1.6.

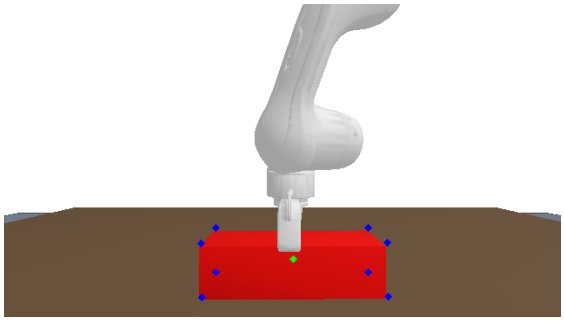


Figure 5.4: Corners and center of the target object projected into the image plane, to get the size and position of the target object in image dimensions.

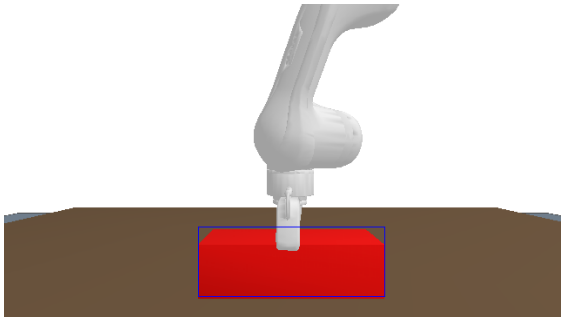


Figure 5.5: Bounding rectangle in image frame containing all corners of the target object.

In each iteration of tracking, a subregion x, corresponding to the estimated position and size, is cropped from the current frame and fed through the SiamMask model alongside each individual RGB template from the THOR long-term module $T_{\mathrm{l}1}, \ldots, T_{\mathrm{l}5}$ and short-term module $T_{\mathrm{s}1}, \ldots, T_{\mathrm{s}5}$

$$g_\theta(T_{\mathrm{l}1}, \ \ldots, \ T_{\mathrm{l}5}; T_{\mathrm{s}1}, \ \ldots, \ T_{\mathrm{s}5}; x). \tag{5.4}$$

The execution is done in batched mode, thus not increasing the computational cost, in terms of computation time, of SiamMask inference. Both template modules, the subregion x and the original frame are shown in figure 5.6.

The template modules are constantly updated during tracking. Templates in the STM are updated in a first-in first-out manner. Every 10 iterations a new template is generated from the current crop and appended to the STM, while the oldest template in the STM is removed. $T_{\mathrm{l}1}$, the ground truth template, always stays in the LTM.

Let z denote the features extracted from a template T

$$z = f_\theta(T).$$

A Gram Matrix

$$G(z_1, \cdots, z_m) = \begin{bmatrix} z_1 \star z_1 & z_1 \star z_2 & \cdots & z_1 \star z_m \\ \vdots & \vdots & \ddots & \vdots \\ z_m \star z_1 & z_m \star z_2 & \cdots & z_m \star z_m \end{bmatrix}$$

is constructed, where each entry G_{ij} expresses the similarity of templates T_i and T_j by cross-correlating the features z_i and z_j. If a novel template T_{c} is similar enough to the ground-truth template $T_{\mathrm{l}1}$, and incorporating it in the LTM, by replacing another one, increases the volume of the parallelotope

$$\Gamma\,(z_1, \ \ldots, z_5)$$

spanned by the feature vectors of each template in feature space, it is added to the LTM. A new template T_{c} is similar enough, if its features z_c satisfy the inequality

$$z_c \star z_1 > l \cdot G_{11} - \gamma,$$

where l is a hyperparameter to trade-off tracking performance against drifting robustness and γ takes the short-term appearence of the target object into account. γ is computed using the templates in the STM

$$\gamma = 1 - \frac{2}{N(N+1)G_{st,max}} \sum_{i<j}^{N} G_{st,ij}$$

where G_{st} is a Gram Matrix for the STM similar to G.

For inference, an activation map is computed using the weighted features from all templates. This activation map can be seen in figure 5.6c. The weights correspond to the maximum score of each individual template. Templates in the STM are prone to tracking drift, since no template stays in the STM permanently. Addressing this issue is done by computing the intersection over union (IoU) between the two bounding boxes $B_{max,st}$ and $B_{max,lt}$ of the templates with the highest score from the STM and the LTM

$$s_{\text{IoU}} = \frac{A_{B_{max,st} \cap B_{max,lt}}}{A_{B_{max,st} \cup B_{max,lt}}}.$$

If s_{IoU} is smaller than an IoU threshold t_{IoU}

$$s_{\text{IoU}} < t_{\text{IoU}},$$

the prediction of the LTM is used and the STM is reinitialized using the template from the LTM with highest score. More details on the template collection strategy, alternatives and inference considerations are found in chapter 3.1.7.

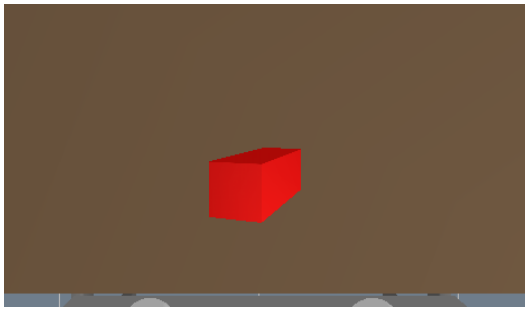
Pointcloud registration After successfully retrieving the current prosition and segmentation of the target object in RGB frame, the corresponding pixels in the depth image are selected. These pixels are then projected to points on the object's surface in world coordinates using equation 5.1. The resulting pointcloud in world coordinates can be seen in figure 5.7. This set of points $\mathcal{P}_{\text{obs}}$ is used as an observation to fit the template pointcloud $\mathcal{P}_{\text{tem}}$ (seen in figure 5.3) to.

To reduce the computational cost of pointcloud registration, both pointclouds are preprocessed via a voxel grid filter, reducing the dimesionality of the pointcloud, while keeping the geometrical properties.

Pointcloud fitting is done using the Generalized-ICP algorithm, setting either the last known pose of the object or the pose computed using forward kinematics, in case the robot is handling the target object, as an initial guess. Formally, this can be seen as a function mapping from a set of pointclouds and an initial guess X to final transformation Y

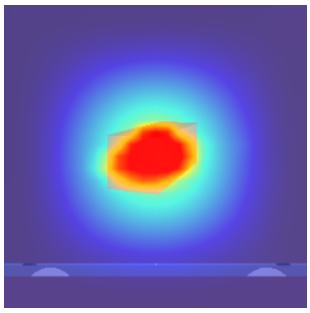
$$h_{\text{G-ICP}} : \mathbb{R}^{n\times 3} \times \mathbb{R}^{m\times 3} \times \mathbb{R}^{4\times 4} \to \mathbb{R}^{4\times 4}$$
$$h_{\text{G-ICP}} : (\mathcal{P}_{\text{observation}}, \mathcal{P}_{\text{template}}, X) \mapsto Y.$$

Chapter 3.2 goes into further detail on the G-ICP algorithm.



(a) Original input RGB frame.

(b) Cropped subwindow of the original frame as input for SiamMask.



(c) Modulated activation maps from inference with all templates.

(d) THOR long-term module.

(e) THOR short-term module.

Figure 5.6: Inference with SiamMask and THOR. a) shows the original input RGB frame. The original frame is cropped to b) centering at an approximate location of the target object. c) shows the weighted activation map from inference with all templates in the LTM d) and the STM e).

5.3 Accumulating a 3D model during tracking

The method for 6-DoF tracking presented so far works for any translation of the target object, but has some theoretical and practical limitations when it comes to heavy changes in orientation (e.g. more than $45°$ on some axis). This is due to the model-free approach of the method. This section, however, explains a method to constantly update the model of the object in an online manner. Briefly, the idea is to accumulate a memory of n template pointclouds

$$\mathcal{T}_{\text{mem}} = \{(\mathcal{P}_{\text{t1}}, R_{\text{t1}}), \ \ldots \ , (\mathcal{P}_{\text{tn}}, R_{\text{tn}})\}$$

while tracking the object, similar to what THOR does in RGB images. $\mathcal{P}_{\text{ti}}$ denotes the set of points $x \in \mathbb{R}^3$ from surface of the target object captured by the depth sensor. $R_{\text{ti}} \in \mathbb{R}^{3\times3}$ denotes the orientation of the target object at the time $\mathcal{P}_{\text{ti}}$ is captured.

The strategy for accumulating $\mathcal{T}_{\text{mem}}$ needs to obey two objectives: 1. add a new template pointcloud soon before pointcloud registration with the current set of templates becomes innacurate due to a heavy change in orientation, and 2. keep the number of templates small to reduce tracking drift.

At each iteration of object tracking, the target object's current orientation $\hat{R}$ is used to compute the geodesic loss

$$\ell_{geo}(\hat{R},\ R) = \cos^{-1}\left[\frac{tr(\hat{R}^T R) - 1}{2}\right] \tag{5.5}$$

between itself and the orientations R_{ti} of all n template pointclouds added so far:

$$l_{\text{min}} = \min_i \left(\ell_{geo}(\hat{R}, R_{\text{ti}})\right)_{i=1}^{n} . \tag{5.6}$$

If l_{min} is greater than the maximum orientation difference treshold t_{ori}

$$l_{\text{min}} > t_{\text{ori}} ,$$

a new template pointcloud is generated from the current observation $\mathcal{P}_{\text{obs}}$ and the current cartesian pose Y of the object by applying Y^{-1} to each individual point $x^i \in \mathcal{P}_{\text{obs}}$

$$\bar{x}^i \leftarrow Y^{-1} \cdot \bar{x}^i,$$

where

$$\bar{x}^i = (x_1^i,\ x_2^i,\ x_3^i,\ 1) \in \mathbb{R}^4.$$

$\mathcal{P}_{\text{obs}}$ is saved along the current cartesian object orientation R_Y and added to the set of templates

$$\mathcal{T}_{\text{mem}} \leftarrow \mathcal{T}_{\text{mem}} \cup (\mathcal{P}_{\text{obs}},\ R_Y)$$

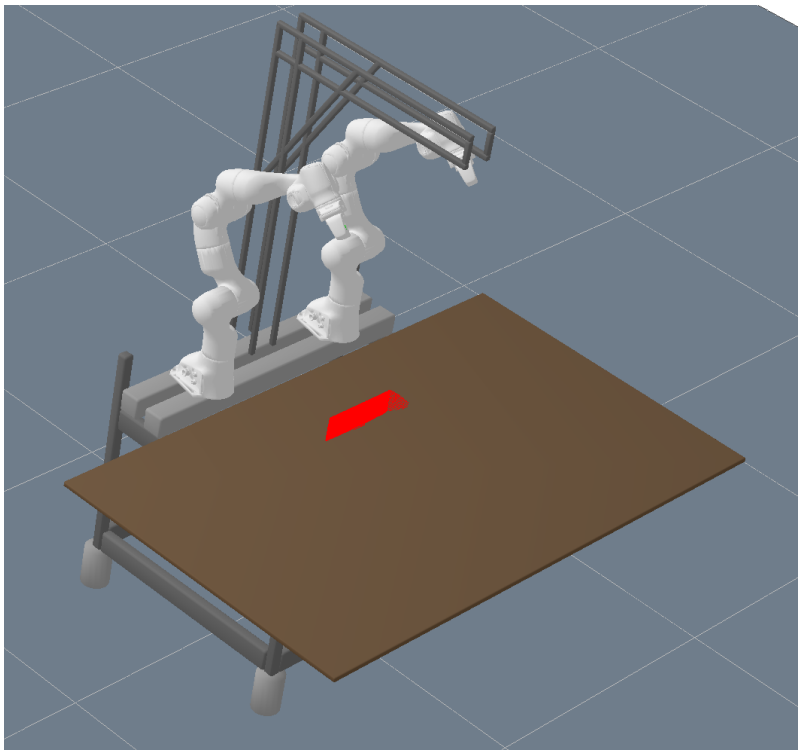


Figure 5.7: Segmented pointcloud observation projected from depth image to world coordinates. The template pointcloud is fit to this observation, which yields the target object's current pose.

At inference time, the template pointcloud with minimal orientation difference l_{min} to the current object pose is used to be fit to the current observation. In order to keep the tracking drift low, a Kalman filter similar to [104] is used on the object's poses.

In contrast to the THOR method for accumulating a template memory, where the set of all templates maximizes the volume of a parallelotope in the space of visual features, deciding on templates to add, can here be done solely by taking the 3D orientation of the object into account. Adding new templates in conjunction with the THOR mechanism would likely be too often, since there is a limited set of templates in different object orientations needed in order to track rigid objects.

Further, following THOR's inference strategy is particularly error prone, since symmetrical objects often "look" the same from both sides in RGB image, and therefore matching with a RGB template from either side does not imply any errors. However, fitting a template pointcloud observed from the wrong side of the object implies a heavy error in rotation, from which recovering is complicated.

6 Experiments

Following sections describe experiments conducted in order to validate effectiveness of the proposed approach. Quantitive evaluation of the method is done in simulation only, otherwise no ground truth signal would be available. Altough to quantify this is complicated, in general THOR + G-ICP performed better on real life images than in simulation. This might be due to the bias, training the underlying deep learning model on real-world images introduces.

6.1 6-DoF Object Tracking

Experiments on 6-DoF tracking in this section are conducted in simulation on a Ubuntu 18.04 machine with an AMD Ryzen 1700x CPU, 48 GB RAM and a NVIDIA GeForce 1080 Ti GPU. The method proposed in this work is evaluated against an implementation of a particle-filtering based object tracker [32] (named particle in plots). All experiments measure two important metrics, the translational error ℓ_t and the rotational error ℓ_{geo} of the estimated pose $\hat{x}$ compared to the ground truth pose x. Translational errors are calculated using the euclidean norm

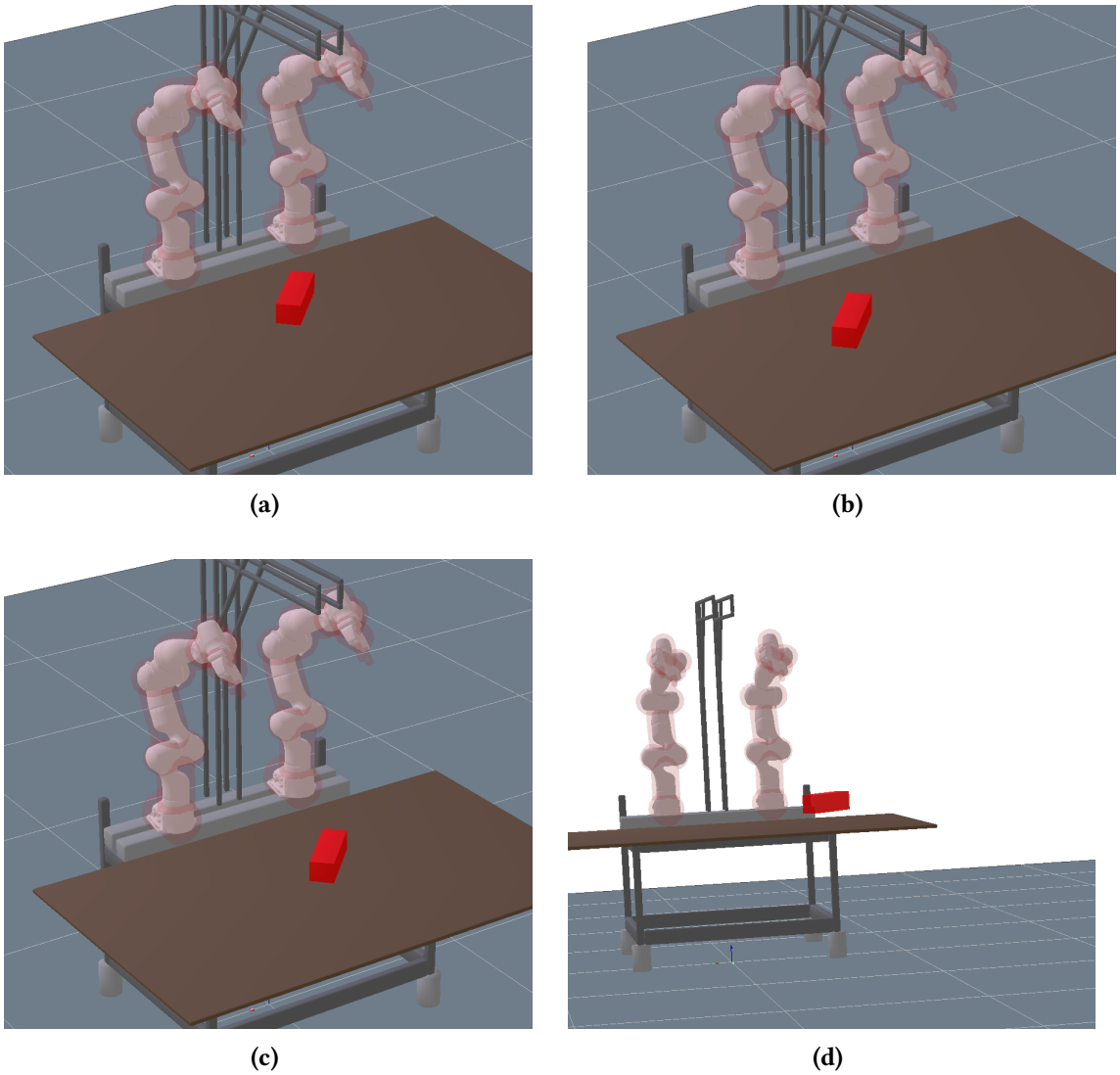
$$\ell_t(\hat{p},\ p) = ||\hat{p} - p||_2 \tag{6.1}$$

where $\hat{p} \in \mathbb{R}^3$ is the translational component of $\hat{x}$ and $p \in \mathbb{R}^3$ is the translational component of x. Reporting rotational errors is done using the geodesic loss (equation 5.5).

6.1.1 Translation

The following section covers experiments on plain object tracking. No occlusions or object manipulations occur in the test scenarios within this section.

Free movement In this section the target object is tracked while being freely moved along a trajectory. This is a general benchmark case for object tracking. Figure 6.1 shows the target object (a red box) and the trajectory. As seen in figure 6.2 results show that THOR + G-ICP outperforms the particle tracker both, in terms of translational and rotational error. Specifically, THOR + G-ICP



(a) **(b)**

(c) **(d)**

Figure 6.1: Trajectory of the object during tracking experiment without any robot interaction. First, the object is translated slowly for a small step on the table, then the translation direction is changed rapidly and speed is increased. Last, the object is lifted from the table, hovering 10 cm above the table in the last configuration.

imposes nearly no error in the objects rotation and a slight error in the position, which stays stable during tracking. This is particularly important for robotic manipulation, since computing a good grasping position is only possible if the estimated pose of the tracked object stays stable.

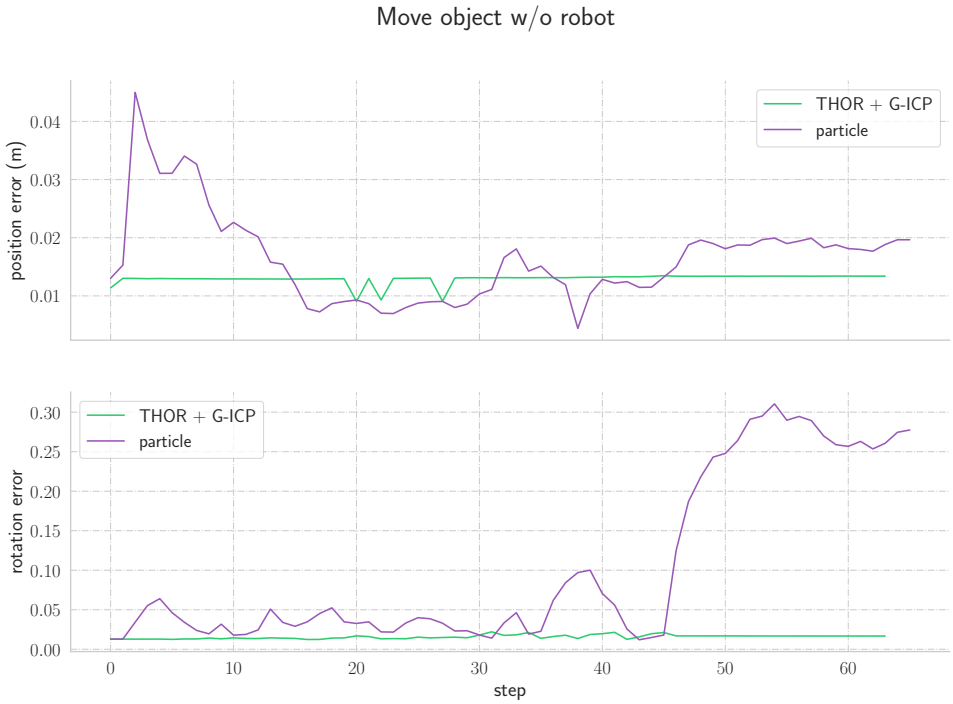


Figure 6.2: Results of object tracking under a free trajectory. THOR + G-ICP imposes almost no rotational error, and a slight error in position. Notably, tracking stays stable for the complete trajectory.

Velocity Experiments on the maximum object velociy, the tracking method is able to cope with, are described in this section. The object trajectory is the same as in the experiment before. The trajectory is executed at different velocities, namely $0.5\frac{m}{s}$, $1\frac{m}{s}$, $2\frac{m}{s}$, $4\frac{m}{s}$. Object tracking should work under various velocities, and be stable even under high velocities. However, velocities like $4\frac{m}{s}$ are unlikely to occur in real-world robotics manipulation tasks. Results can be seen in figure 6.3 and clearly show an advantage of THOR + G-ICP over the particle-filter based tracker: at no point the tracked object is completely lost. The particle filter tracker, however, can only deal with an average velocity of $0.5\frac{m}{s}$, and even here, the rotational error gets significantly higher towards the end of the trajectory.

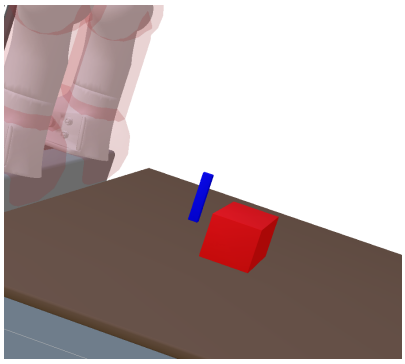
Figure 6.3: Results of tracking under various object velocities. While the particle tracker can only deal with a velocity of $0.5\frac{m}{s}$, THOR + G-ICP does not lose the object at any given velocity.

No movement The experiment conducted in this section targets the accuracy of tracking while the object is not moving at all. Hence, the desired behavior is that tracking remains stable at the same location. This is necessary for robotic manipulation since predicting whether an object will move is hard, and therefore it needs to be tracked even during times when no movements occur. If tracking is not stable enough, robot motion planning will infer temporary collisions and therefore grasping will not be possible. Results of this experiment are depicted in figure 6.4. The plot shows that THOR + G-ICP imposes a slight translational error on tracking which stays stable, however, the rotational error is minimal. The particle filter tracker on the other hand shows fluctuating error in both, rotational and translational terms. Real-world experiments on this topic show that using the particle filter tracker, stably grasping the object is nearly impossible. This is also due to occlusions by the gripper finger, which is discussed in the next section.

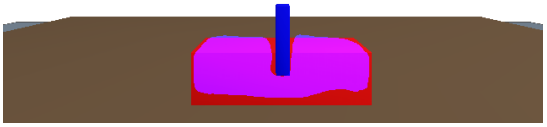
Figure 6.4: Results of tracking an object while no object movements occur. THOR + G-ICP has a constant translational error which stays stable, and is slightly above the error of the particle filter. Yet, the particle filter shows a rotational error with fluctuations which can introduce heavy complications for object grasping and manipulation.

6.1.2 Occlusions

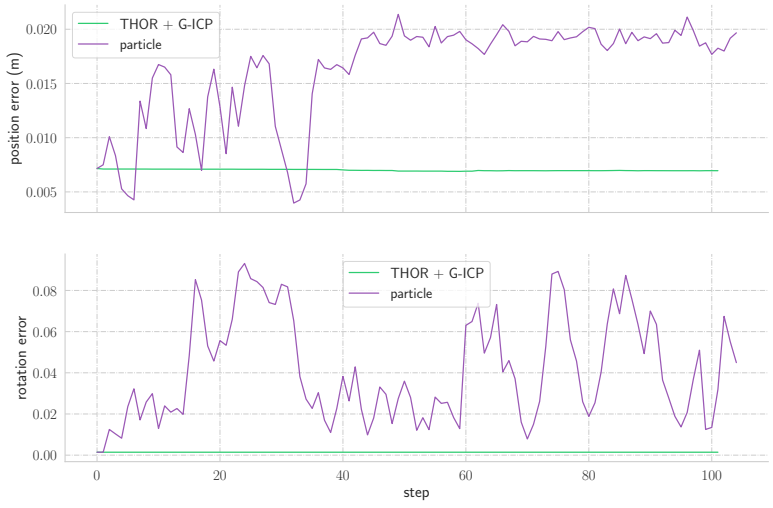
This section demonstrates how well the approach deals with occlusions. Occlusions occur frequently during robotic manipulation. For example when grasping an object with a parallel jaw gripper, the gripper fingers already occlude parts of the object. Therefore, the approach should be able to deal with occlusions to a certain degree. In this experiment a small box which moves into the scene from the top occludes the target object, as seen in figure 6.5a. Results, seen in figure 6.5c, show that occlusion has almost no impact on THOR + G-ICP. The accuracy of the particle tracker is reduced proportional to the amount of occlusion. The stability of THOR + G-ICP only relies on the fact that the RGB tracker SiamMask is able to accurately segment the tracked object from the occluding object. As seen in figure 6.5b, the segmentation mask of SiamMask solely contains part of the target object. Therefore G-ICP has no problems to fit the template pointcloud to the observations, because the observation pointcloud omits the occluded part.



(a) A small rectangle occludes the target object.



(b) Camera view on the scene. The object, which occludes the target object, is not contained in the segmentation mask.



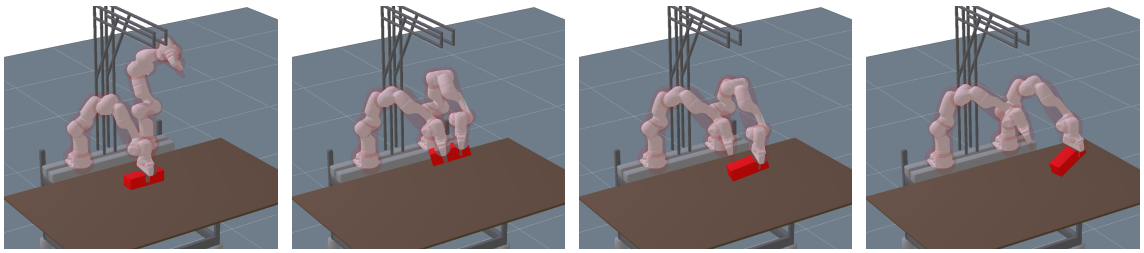
(c) Results of object tracking during occlusion. THOR + G-ICP stays stable during the complete tracking sequence.

Figure 6.5: Object tracking during occlusion.

6.1.3 Manipulation

This section describes experiments on object tracking while the object is manipulated by two robots. Supporting robotic manipulation is the main purpose of this object tracking method and therefore an important experiment. It incorporates many different possibilites which can occur during object tracking, e.g. translational and orientational movements as well as partial occlusion due to the gripper fingers or the second robot. Figure 6.6 shows the manipulation process and results of object tracking during manipulation. First, one of two robots picks the object and hands it over to the other robot. The second robot keeps the object held and executes a trajectory to a predefined goal position. Results of object tracking are depicted in figure 6.7. These show that both methods are able to track the target object during manipulation. However, the particle filter tracker loses the object once completely. This corresponds to the moment of the handover (figure 6.6b) and is due to the occlusion both gripper fingers generate on the object. Tracking with THOR + G-ICP is stable during the whole manipulation process and also generates almost no rotational error. All experiments conducted so far show that object rotations are captured well by THOR + G-ICP.

Robot information While in figure 6.7 no information from the robots kinematic tree is used, figure 6.8 shows the improvement when incorporating information from the robot kinematics on the object pose. However, since the particle-filtering tracking method keeps a complicated particle state between each transition, such information is not easily integrated into the method. This states a major advantage of THOR + G-ICP. Results show that object tracking clearly improves over the complete trajectory when incorporating robot information. Unfortunately, such prior information is not available when humans move the object.



(a) **(b)** **(c)** **(d)**

Figure 6.6: Object tracking during manipulation. In figure a) the first robot picks up the target object and hands it over to the second robot in b). The second robot moves the object to an intermediate position in c) and finally moves to the goal position d) while keeping the object in hand.

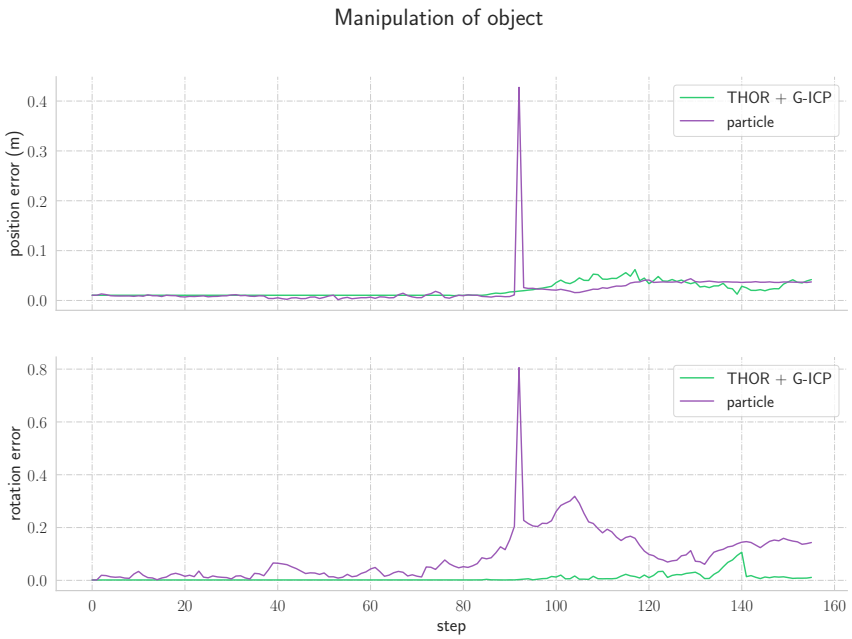


Figure 6.7: Results of tracking during manipulation. No information from the robot kinematics is used in this experiment.

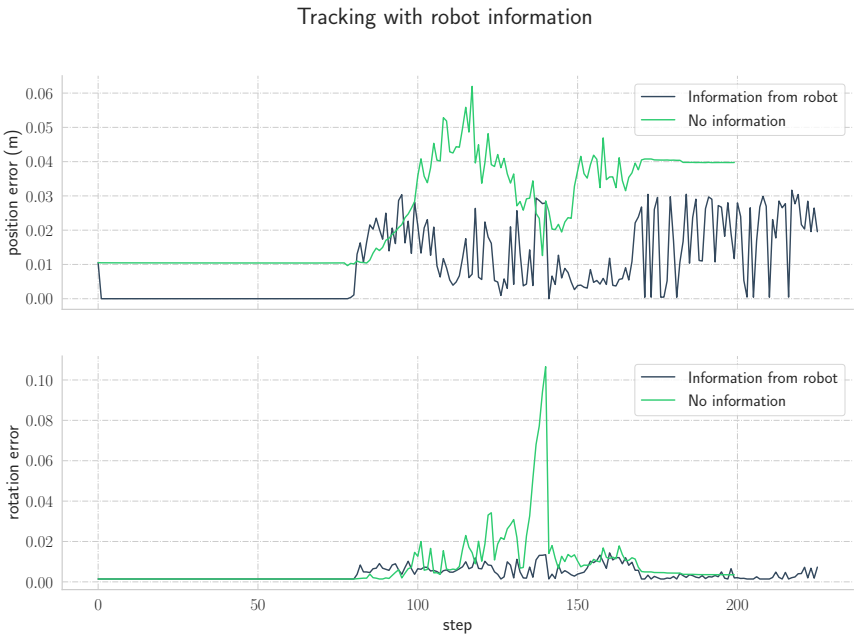
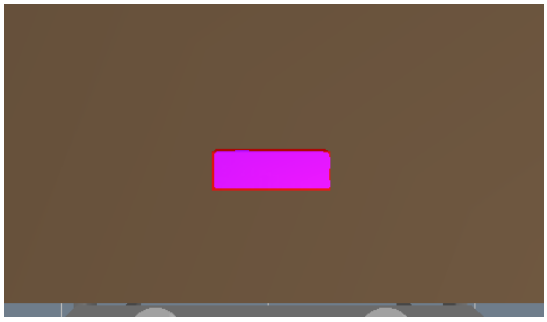


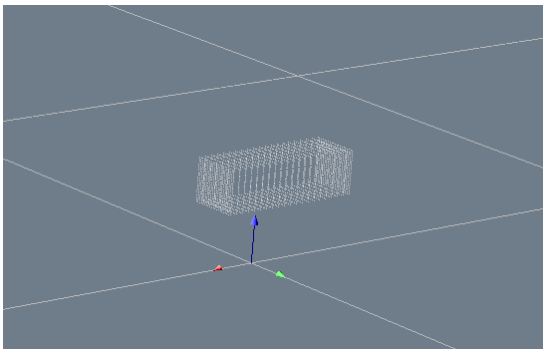
Figure 6.8: Result of tracking when incorporating robot information into the tracking process. Object tracking clearly improves.

6.1.4 Full model

Experiments in this section are conducted using a pointcloud template sampled from the complete surface of the object. Such a pointcloud can be seen in figure 6.9b. In general this data is not available for arbitary unknown objects, but it can be sampled from a CAD file describing the target object. Specfically, in this case the pointcloud is generated using the knowledge that only boxes are manipulated in this experiment. Tracking is done in top-down camera view, as seen in figure 6.9a. Since the particle-filter based tracker is a model-based algorithm, it should perform better in this section. Figure 6.9 shows results of tracking the target object while rotating it for 2π around the x-axis and figure 6.10 shows results of tracking while rotating the object around the y-axis. THOR + G-ICP outperforms the other algorithm in both experiments, showing its usability even in settings, where 3D models are available. Rotating the the object around the z axis however generates a failure, which can be mitigated by using a different pointcloud sub-sampling method. This is addressed in chapter 6.3.



(a) Camera view of table and target object from top-down camera.



(b) Full template pointcloud of the target object.

(c)

Figure 6.9: Rotating the object for 2π around the x-axis. The plot shows the translational and rotational error of THOR + G-ICP and the particle filter tracker. THOR + G-ICP outperforms the particle filter based tracker in this experiment.

Figure 6.10: Rotating the object for 2π around the y-axis. The plot shows the translational and rotational error of THOR + G-ICP and the particle filter tracker.

6.1.5 Generating new template pointclouds

In this section experiments are conducted on generating a full object model on-the-fly during tracking, as described in chapter 5.3. The target object is tracked from top camera view. First, it is rotated about 180 degree around the x-axis, then the object is translated about 20cm. Figure 6.11 shows the results of this experiment. In the first experiment, the target object is tracked while constructing an object model by storing multiple pointcloud templates from different object orientations. These pointcloud templates are then used for pointcloud registration, according to the current object pose. Using multiple pointclouds, the THOR + G-ICP is able to estimate the correct orientation, yet, the performance in estimating the correct position suffers. Tracking with only the first pointcloud observed during initialization, fails as expected, since after a rotation around 180 degree, the initial view of the object can not be seen anymore. However, only the rotational error increases, while the positional error stays in about the range of the other experiments. Usage of the Kalman filter has no major impact on tracking performance.

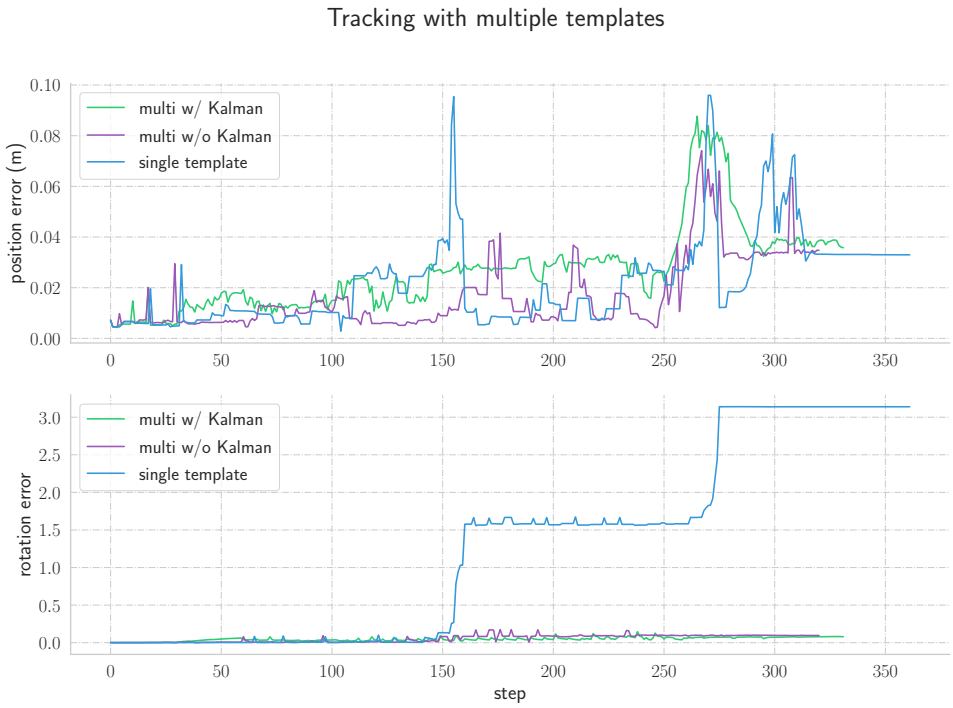


Figure 6.11: Result of tracking while constructing an object model. Multiple pointcloud templates of different object orientations are stored . Tracking is done using multiple templates with or without a Kalman filter on the object poses and with only a single pointcloud template.

6.2 Dynamic grasping of objects

The object tracking method proposed in this work is used to realize dynamic grasping of arbitrary ojects using a parallel jaw gripper and a Franka Emika Panda robot. This experiment shows qualitative results of using THOR + G-ICP for object tracking in real-world robotic manipulation and is conducted in conjunction with the Master's Thesis of Julian Hörz. His Master's Thesis answers the question: How can grasp planning react to a dynamically changing environment? In order to perceive the environment, the methods for object detection and object tracking presented in chapter 5 are used.

We use two cameras, an ASUS Xtion PRO LIVE RGBD camera as the top camera and an Intel RealSense D415 RGBD as side camera. The system is controlled by two computers: one for communicating with the robot via the Franka realtime interface [1], the second one is used for object tracking. Both machines communicate via ROS. The ASUS camera is mounted above the table and the RealSense is mounted to view the scene from the side, to be seen in figure 6.13. This is needed since the robot completely occludes objects in top view when manipulating them.

A target object is placed on the table in front of the robot. As soon as the object is detected, the initial pose and size of the 3D bounding box containing the target object is sent to the robot control and tracking starts. At each iteration of tracking the pose of the target object is sent to the robot control. The robot control computes the best possible grasp for the 3D bounding box containing the object and approaches the object. As soon as the objects orientation or position is changed, a new best grasp is computed and used as a target.

Despite grasping points only being computed for the 3D bounding box containing the object, the robot is able to grasp arbitrary rigid objects, since object detection and tracking works for any object. This can be seen in figure 6.12, where a can of pringles is manipulated, and in figure 6.13 where a packaged lashing belt is manipulated. This object is reflecting and partly transparent, however, this had no impact on tracking quality. If approaching the grasp was successful and the object did not move, the robot grasps the object and heads toward a homing position. Yet, if someone takes the object from the robot, the robot starts to grasp it again.

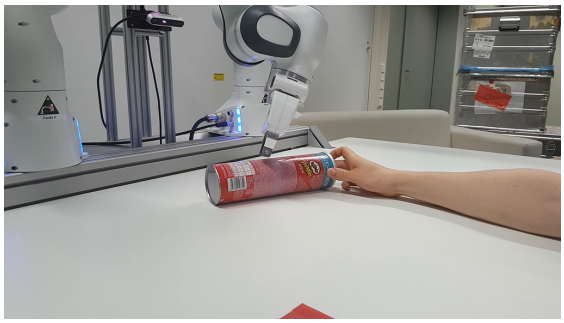


Figure 6.12: Grasping a Pringles can.

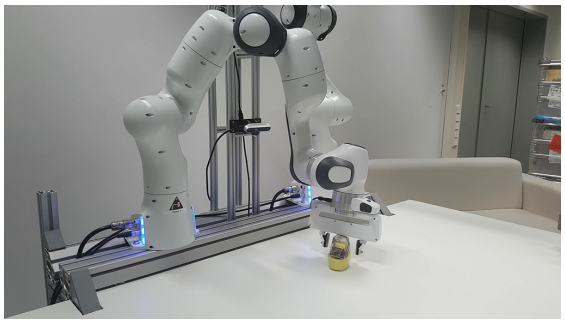


Figure 6.13: Grasping a packaged lashing belt. This object is reflecting and partly transparent.

[1] https://github.com/frankaemika/libfranka

Figure 6.14: Grasping a box while it is dynamically moved by a human. Note the orientation of the object.

6.3 Failure cases

This section addresses known problems where tracking errors might occur.

Rotation Z-Axis Figure 6.15 shows the results of tracking with the full object model pointcloud (see fig. 6.9) as template, while the target object is rotated around the z-axis. The top camera is used for this experiment. After a short period of accurate tracking, THOR + G-ICP loses the object. The kd-tree algorithm used for finding correspondences fails to find the correct corresponding points in both pointclouds and classifies important points as outliers. Therefore only a subset of all points is used for pointcloud registration. The transformation between the individual correspondences in this subset does not impose a meaningful error, making the ICP optimization step more difficult. As seen in the plot, the number of correspondences found in each step is inversely proportional to the translational error.

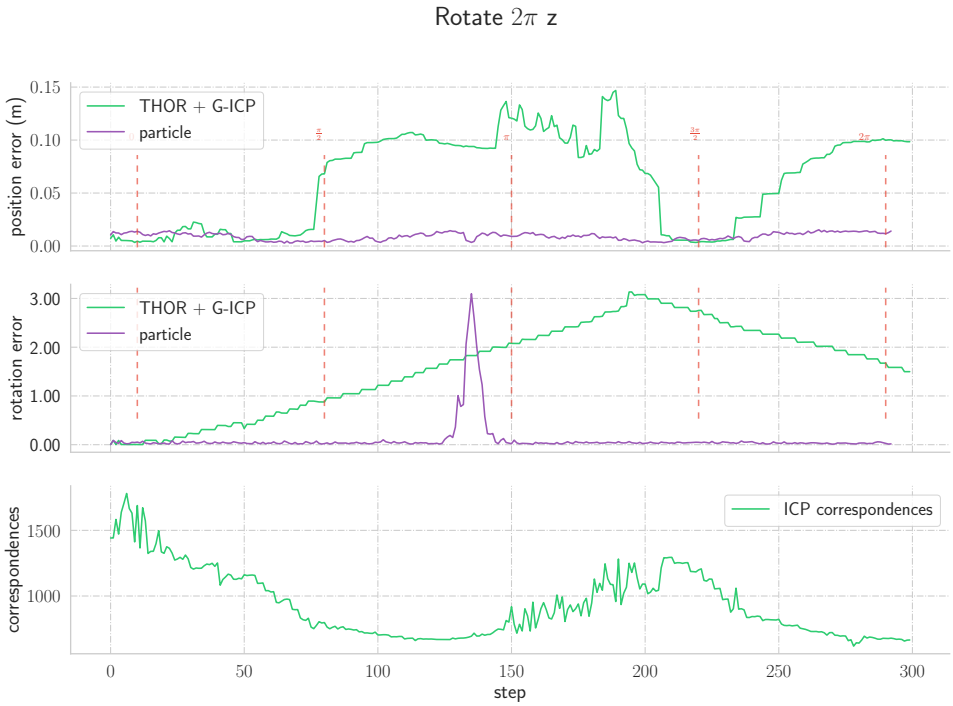


Figure 6.15: Results of tracking during rotation around z using the full object model as template pointcloud. THOR + G-ICP fails in this experiment, since G-ICP classifies important points as outliers.

Mitigation This problem however, only occurs when tracking box-like objects and can be mitigated by using normal-space sampling instead of voxel grid sampling as sub-sampling method. Similar behavior is also observed in [105]. Normal-space sampling groups points according to the position of their normals in angular space and samples points uniformly accross all groups of points. Figure 6.16 shows the results of tracking using different sub-sampling method. With normal-space sampling THOR + G-ICP is stable and accurate, even for rotations around z.

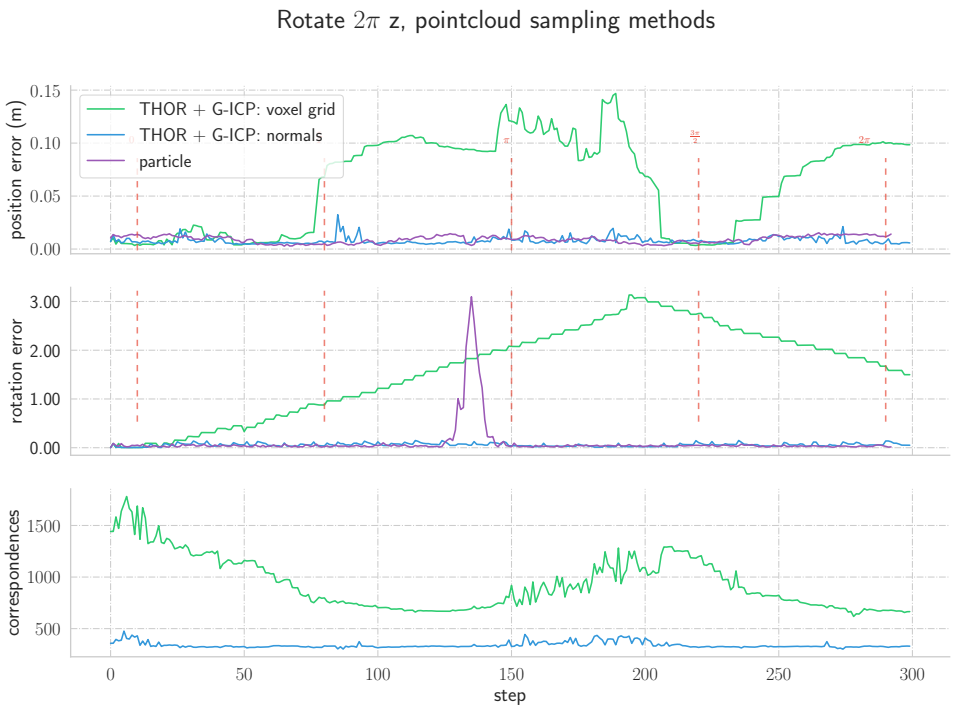
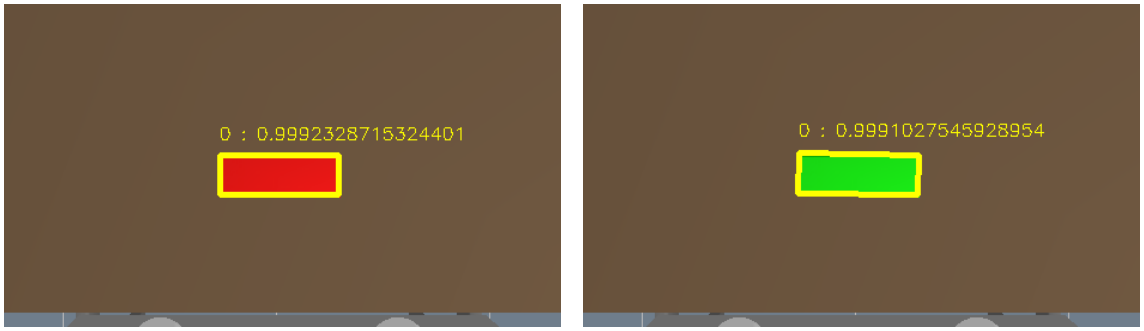


Figure 6.16: Using normal-space sampling as preprocession method mitgates the problem when rotating around z. No tracking errors occur, and THOR + G-ICP outperforms the particle filtering tracker.

Recovery Another weak spot of THOR + G-ICP can further be seen in figure 6.15: while the particle filter is able to recover from a rotation error of nearly π, THOR + G-ICP is not able to recover, since G-ICP is not able to find the correct pose, when the initial guess is far off.

Changing colors This experiment focuses on the tracking score. That is, the confidence $s \in [0, 1]$ of SiamMask on tracking the correct object. If tracking drift occurs and suddenly an other object, not similar to the target object, is tracked, one would expect the confidence score to degrade heavily. Even more so, if the other object has a completely different color. Figure 6.17 shows the confidence score before and after an object swap occurs. The red box is the correct target object. After swapping this object to a green box, the confidence score stays extremely similar, rendering the confidence score an impractical metric to decide, whether tracking still works correctly. While slighlty less severe, this behavior is also encountered in real-world experiments. Here, the confidence score drops shortly from $s = 0.99$ to $s = 0.985$, then increases again, also not serving as a useful metric.



(a) (b)

Figure 6.17: Swapping the object during tracking. In a) the original target object is seen with the corresponding tracking score. b) shows tracking after swapping the target object to an object with a different color. The tracking score is very similar to tracking with the correct object. This behavior also occurs in real-world experiments.

7 Conclusion

While being a simple approach, THOR + G-ICP is stable, fast and accurate enough to support robotic manipulation. Experiments show that it outperforms state-of-the-art object trackers in multiple settings. Using it for object tracking leads to a more robust approach to real-world grasping.

In general, all parts of the approach can be exchanged for other methods: it boils down to having a function which takes an RGB input image and outputs a segmentation mask, containing the target object, and another function using the observation from depth image for pointcloud registration. Thus, other approaches for both problems are evaluated in this work.

RGB Tracking Although neural networks can be executed in batched mode, doing inference using ResNet backbones has heavy implications on the compute power that has to be available for real-time tracking. Executing such operations in CPU-only mode does not even come close to real-time performance. Further, since THOR compares multiple templates to the current observation, these comparisons need to be executed in batched mode to keep the computational overhead of THOR low. Unfortunately, batch inference directly leads to an increased GPU memory utilization, so at inference time, tracking a single object in RGB needs about 2.5 GB of GPU memory. Parallel tracking of multiple objects is therefore limited. Experimenting with object tracking methods contained in the OpenCV library [106], most of which work in real-time on CPU only, shows that these were highly susceptible to occlusions and visual object pertubation. Further, not a single approach was able to segment the target object in the observation frame, therefore the starting point for the pointcloud registration method would be worse. So the deep learning approach to template-matching has proven to be best suited for the task, aligning with the results that benchmarks suggest [52].

Although siamese networks perform really well on the problem of RGB tracking, this does not mean that the problem of visual tracking should be considered solved. There are many ways, tracking can go horribly wrong and get lost without any other possibility to recover, than having an additional method watching the tracking process and recovering if needed [55]. Examples of failure situations include very fast object movements, full occlusion and crossing paths with objects that are very similar. Oddly, only the presence of human skin in the current frame degrades tracking performance heavily, even more so when human faces are present. This clearly demonstrates the effect of training dataset bias: the DAVIS [107] dataset contains many video sequences where humans are the target object.

Further, if tracking gets lost and a completely different object is tracked, the similarity score of this object to the template is very high, suggesting that the features learned for discriminating objects are only partially helpful. Size and position of the target object in the last frame are two parameters absolutely crucial to the performance of tracking. Getting these parameters wrong has a huge impact on tracking performance. Humans always use a prior based on the phyiscs,

how the object can acutally be moved, while following an object visually. Such vital information should not be underestimated: solving the computer vision problem of object tracking might depend on having such a prior.

Pointcloud registration As for pointcloud registration, a few methods other than the Generalized-ICP are also evaluated. Of course, the first one being the standard point-to-point ICP, which does in no way lead to any convergence, not even in the first iteration. Point-to-plane ICP is slightly better, but loses the target object after a few iterations, therefore also proving not very useful. A RANSAC based method was too slow and unstable.

Since the current progress in deep learning methods for pointcloud processing seems promising, I also tested two approaches to pointcloud registration reyling on PointNet [42] as feature extractor. Namely PCRNet [108] and PointNetLK [109], which had to be trained on ModelNet40 [110], since there was no pretrained model available. Deep learning methods seem especially appealing since they can be executed in batched mode on a GPU, leading to a performance boost (e.g. multiple objects or multiple starting points would be possible). Unfortunately, neither of both methods is able to produce useful results for the application in this work. Due to the fact that, unlike dense images, pointclouds are sparse and unstructured data and the weight matrices of neural networks being of fixed size, these models also need an input of a fixed length in order to be executed. Therefore, a fixed number of points has to be sampled from the pointcloud. Research suggests that sampling the right points for the specific task is crucial [111], [112]. So despite the theoretical possibility to interchange the pointcloud registration method with any other method, Generalized-ICP is the only usable method in this particular setting. Collecting a complete model of the object during tracking, as explained in chapter 5.3, is a difficult task. Accumulating errors when saving a new template pointcloud with an estimated transformation will break the system.

Remarks While the proposed method is simple, and works for arbitrary objects with no further training, it has a few problems. Designing it as a model-free method, reduces the assumptions made for tracking, but comes with the limitation that the method is only able to track objects if they only do slight changes in orientaion. To mitigate this flaw, designing a more holistic tracking method, inherently dealing with the multi-modality of RGBD data seems promising. It is worth, looking into creating a 6-DoF tracking system, similar to SiamMask with THOR, but directly operating on pointcloud data. Using PointNet as a feature extractor, it is also possible to incorporate RGB data in the pointcloud [42]. Siamese Networks have been successfully applied to pointcloud registration [113], and should therefore be able to do template-matching in cartesian space. A memory of templates helps to observe the target object from either side.

Bibliography

[1] J. Forlizzi, C. DiSalvo, "Service robots in the domestic environment: A study of the roomba vacuum in the home", in *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction*, 2006, pp. 258–265 (cit. on p. 11).

[2] J. Liang, J. Mahler, M. Laskey, P. Li, K. Goldberg, "Using dVRK teleoperation to facilitate deep learning of automation tasks for an industrial robot", en, in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, Xi'an: IEEE, Aug. 2017, pp. 1–8, ISBN: 978-1-5090-6781-7. DOI: 10.1109/COASE.2017.8256067 (cit. on p. 11).

[3] R. A. Beasley, "Medical Robots: Current Systems and Research Directions", en, *Journal of Robotics*, vol. 2012, pp. 1–14, 2012, ISSN: 1687-9600, 1687-9619. DOI: 10.1155/2012/401613 (cit. on p. 11).

[4] G.-Z. Yang, B. J. Nelson, R. R. Murphy, H. Choset, H. Christensen, S. H. Collins, P. Dario, K. Goldberg, K. Ikuta, N. Jacobstein, D. Kragic, R. H. Taylor, M. McNutt, "Combating COVID-19—The role of robotics in managing public health and infectious diseases", en, *Science Robotics*, vol. 5, no. 40, eabb5589, Mar. 2020, ISSN: 2470-9476. DOI: 10.1126/scirobotics.abb5589 (cit. on p. 11).

[5] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, S. Birchfield, "Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects", en, *arXiv:1809.10790 [cs]*, Sep. 2018. arXiv: 1809.10790 [cs] (cit. on pp. 11, 16).

[6] S. Giancola, J. Zarzar, B. Ghanem, "Leveraging Shape Completion for 3D Siamese Tracking", en, *arXiv:1903.01784 [cs]*, Mar. 2019. arXiv: 1903.01784 [cs] (cit. on p. 11).

[7] M. Wüthrich, P. Pastor, M. Kalakrishnan, J. Bohg, S. Schaal, "Probabilistic Object Tracking using a Range Camera", *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3195–3202, Nov. 2013. DOI: 10.1109/IROS.2013.6696810. arXiv: 1505.00241 (cit. on pp. 11, 15).

[8] M. Garon, J.-F. Lalonde, "Deep 6-DOF Tracking", en, *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 11, pp. 2410–2418, Nov. 2017, ISSN: 1077-2626. DOI: 10.1109/TVCG.2017.2734599. arXiv: 1703.09771 (cit. on pp. 11, 15).

[9] T. Schmidt, R. Newcombe, D. Fox, "DART: Dense Articulated Real-Time Tracking", en, in *Robotics: Science and Systems X*, Robotics: Science and Systems Foundation, Jul. 2014, ISBN: 978-0-9923747-0-9. DOI: 10.15607/RSS.2014.X.030 (cit. on pp. 11, 15).

[10] J. Schulman, A. Lee, J. Ho, P. Abbeel, "Tracking deformable objects with point clouds", en, in *2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany: IEEE, May 2013, pp. 1130–1137, ISBN: 978-1-4673-5643-5 978-1-4673-5641-1. DOI: 10.1109/ICRA.2013.6630714 (cit. on pp. 11, 15).

[11] A. Feldman, T. Balch, M. Hybinette, R. Cavallaro, "The Multi-ICP Tracker: An Online Algorithm for Tracking Multiple Interacting Targets", en, p. 22, (cit. on p. 11).

[12] J. Cho, S. Jin, X. Pham, J. Jeon, J. Byun, H. Kang, "A Real-Time Object Tracking System Using a Particle Filter", en, in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China: IEEE, Oct. 2006, pp. 2822–2827, ISBN: 978-1-4244-0258-8 978-1-4244-0259-5. DOI: 10.1109/IROS.2006.282066 (cit. on p. 11).

[13] A. Almeida, J. Almeida, Rui Araujo, "Real-Time Tracking of Moving Objects Using Particle Filters", en, in *Proceedings of the IEEE International Symposium on Industrial Electronics, 2005. ISIE 2005.*, Dubrovnik, Croatia: IEEE, 2005, pp. 1327–1332, ISBN: 978-0-7803-8738-6. DOI: 10.1109/ISIE.2005.1529124 (cit. on p. 11).

[14] M. Fleder, S. Pillai, "3D Object Tracking Using the Kinect", en, p. 8, (cit. on p. 11).

[15] F. Leeb, A. Byravan, D. Fox, "Motion-Nets: 6D Tracking of Unknown Objects in Unseen Environments using RGB", en, *arXiv:1910.13942 [cs]*, Oct. 2019. arXiv: 1910.13942 [cs] (cit. on pp. 11, 16).

[16] D. Kragic, M. Björkman, H. I. Christensen, J.-O. Eklundh, "Vision for robotic object manipulation in domestic settings", en, *Robotics and Autonomous Systems*, vol. 52, no. 1, pp. 85–100, Jul. 2005, ISSN: 09218890. DOI: 10.1016/j.robot.2005.03.011 (cit. on pp. 11, 16).

[17] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, P. H. S. Torr, "Fast Online Object Tracking and Segmentation: A Unifying Approach", en, *arXiv:1812.05050 [cs]*, Dec. 2018. arXiv: 1812.05050 [cs] (cit. on pp. 11, 16).

[18] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, C. Liu, "A Survey on Deep Transfer Learning", *arXiv:1808.01974 [cs, stat]*, Aug. 2018. arXiv: 1808.01974 [cs, stat] (cit. on p. 11).

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", en, *arXiv:1409.0575 [cs]*, Jan. 2015. arXiv: 1409.0575 [cs] (cit. on p. 11).

[20] W. Gao, R. Tedrake, "FilterReg: Robust and Efficient Probabilistic Point-Set Registration using Gaussian Filter and Twist Parameterization", *arXiv:1811.10136 [cs]*, Jul. 2019. arXiv: 1811.10136 [cs] (cit. on pp. 11, 15).

[21] B. Eckart, K. Kim, J. Kautz, "HGMR: Hierarchical Gaussian Mixtures for Adaptive 3D Registration", en, in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, Y. Weiss, Eds., vol. 11219, Cham: Springer International Publishing, 2018, pp. 730–746, ISBN: 978-3-030-01266-3 978-3-030-01267-0. DOI: 10.1007/978-3-030-01267-0_43 (cit. on p. 11).

[22] A. Sauer, E. Aljalbout, S. Haddadin, "Tracking Holistic Object Representations", en, *arXiv:1907.12920 [cs, stat]*, Aug. 2019. arXiv: 1907.12920 [cs, stat] (cit. on pp. 11, 34).

[23] A. Segal, D. Haehnel, S. Thrun, "Generalized-icp.", in *Robotics: Science and Systems*, Seattle, WA, vol. 2, 2009, p. 435 (cit. on pp. 11, 17, 36, 37).

[24] A. Yilmaz, O. Javed, M. Shah, "Object tracking: A survey", en, *ACM Computing Surveys*, vol. 38, no. 4, 13–es, Dec. 2006, ISSN: 03600300. DOI: 10.1145/1177352.1177355 (cit. on p. 12).

[25] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, K. Goldberg, "Learning ambidextrous robot grasping policies", *Science Robotics*, vol. 4, no. 26, eaau4984, 2019 (cit. on p. 12).

[26] D. Morrison, P. Corke, J. Leitner, "Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach", in *Proc. of Robotics: Science and Systems (RSS)*, 2018 (cit. on p. 12).

[27] H. Liang, X. Ma, S. Li, M. Görner, S. Tang, B. Fang, F. Sun, J. Zhang, "PointNetGPD: Detecting Grasp Configurations from Point Sets", en, *arXiv:1809.06267 [cs]*, Feb. 2019. arXiv: 1809.06267 [cs] (cit. on p. 12).

[28] A. ten Pas, M. Gualtieri, K. Saenko, R. Platt, "Grasp Pose Detection in Point Clouds", en, *arXiv:1706.09911 [cs]*, Jun. 2017. arXiv: 1706.09911 [cs] (cit. on p. 12).

[29] A. Doucet, N. de Freitas, N. Gordon, "Sequential Monte Carlo Methods in Practice", en, p. 12, (cit. on p. 15).

[30] Vermaak, Doucet, Perez, "Maintaining multimodality through mixture tracking", en, in *Proceedings Ninth IEEE International Conference on Computer Vision*, Nice, France: IEEE, 2003, 1110–1116 vol.2, ISBN: 978-0-7695-1950-0. DOI: 10.1109/ICCV.2003.1238473 (cit. on p. 15).

[31] K. Okuma, A. Taleghani, N. de Freitas, J. J. Little, D. G. Lowe, "A Boosted Particle Filter: Multitarget Detection and Tracking", en, in *Computer Vision - ECCV 2004*, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, T. Pajdla, J. Matas, Eds., vol. 3021, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 28–39, ISBN: 978-3-540-21984-2 978-3-540-24670-1. DOI: 10.1007/978-3-540-24670-1_3 (cit. on p. 15).

[32] R. B. Rusu, *Tracking 3D objects with Point Cloud Library*, Jan. 2012 (cit. on pp. 15, 51).

[33] C. Choi, H. I. Christensen, "RGB-D object tracking: A particle filter approach on GPU", in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 1084–1091 (cit. on p. 15).

[34] J. Issac, M. Wüthrich, C. G. Cifuentes, J. Bohg, S. Trimpe, S. Schaal, "Depth-Based Object Tracking Using a Robust Gaussian Filter", *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 608–615, May 2016. DOI: 10.1109/ICRA.2016.7487184. arXiv: 1602.06157 (cit. on p. 15).

[35] Y. Xiao, X. Qiu, P.-A. Langlois, M. Aubry, R. Marlet, "Pose from Shape: Deep Pose Estimation for Arbitrary 3D Objects", *arXiv:1906.05105 [cs]*, Aug. 2019. arXiv: 1906.05105 [cs] (cit. on p. 15).

[36] J. A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard, M. Pivtoraiko, J.-S. Valois, R. Zhu, "An integrated system for autonomous robotics manipulation", en, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal: IEEE, Oct. 2012, pp. 2955–2962, ISBN: 978-1-4673-1736-8 978-1-4673-1737-5 978-1-4673-1735-1. DOI: 10.1109/IROS.2012.6385888 (cit. on p. 15).

[37] W. Kehl, F. Tombari, S. Ilic, N. Navab, "Real-Time 3D Model Tracking in Color and Depth on a Single CPU Core", en, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI: IEEE, Jul. 2017, pp. 465–473, ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.57 (cit. on p. 15).

[38] B. A. Erol, A. Majumdar, J. Lwowski, P. Benavidez, P. Rad, M. Jamshidi, "Improved Deep Neural Network Object Tracking System for Applications in Home Robotics", en, in *Computational Intelligence for Pattern Recognition*, W. Pedrycz, S.-M. Chen, Eds., vol. 777, Cham: Springer International Publishing, 2018, pp. 369–395, ISBN: 978-3-319-89628-1 978-3-319-89629-8. DOI: 10.1007/978-3-319-89629-8_14 (cit. on p. 15).

[39] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, K. Daniilidis, "6-DoF Object Pose from Semantic Keypoints", *arXiv:1703.04670 [cs]*, Mar. 2017. arXiv: 1703.04670 [cs] (cit. on p. 15).

[40] S. Song, J. Xiao, "Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images", en, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 808–816, ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.94 (cit. on p. 15).

[41] J. Hou, A. Dai, M. Nießner, "3D-SIS: 3D Semantic Instance Segmentation of RGB-D Scans", en, *arXiv:1812.07003 [cs]*, Apr. 2019. arXiv: 1812.07003 [cs] (cit. on p. 15).

[42] C. R. Qi, H. Su, K. Mo, L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation", en, *arXiv:1612.00593 [cs]*, Apr. 2017. arXiv: 1612.00593 [cs] (cit. on pp. 16, 70).

[43] C. R. Qi, L. Yi, H. Su, L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", en, *arXiv:1706.02413 [cs]*, Jun. 2017. arXiv: 1706.02413 [cs] (cit. on p. 16).

[44] C. R. Qi, O. Litany, K. He, L. J. Guibas, "Deep Hough Voting for 3D Object Detection in Point Clouds", en, *arXiv:1904.09664 [cs]*, Aug. 2019. arXiv: 1904.09664 [cs] (cit. on p. 16).

[45] B. Leibe, A. Leonardis, B. Schiele, "Combined Object Categorization and Segmentation with an Implicit Shape Model", en, p. 16, (cit. on p. 16).

[46] R. Luo, R. Mullen, D. Wessell, "An adaptive robotic tracking system using optical flow", en, in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, USA: IEEE Comput. Soc. Press, 1988, pp. 568–573, ISBN: 978-0-8186-0852-0. DOI: 10.1109/ROBOT.1988.12112 (cit. on p. 16).

[47] A. Pieropan, N. Bergstrom, M. Ishikawa, H. Kjellstrom, "Robust 3D tracking of unknown objects", en, in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA: IEEE, May 2015, pp. 2410–2417, ISBN: 978-1-4799-6923-4. DOI: 10.1109/ICRA.2015.7139520 (cit. on p. 16).

[48] R. Tao, E. Gavves, A. W. M. Smeulders, "Siamese Instance Search for Tracking", en, May 2016 (cit. on p. 16).

[49] Y. Wu, J. Lim, M.-H. Yang, "Online Object Tracking: A Benchmark", en, p. 8, (cit. on p. 16).

[50] A. W. M. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, M. Shah, "Visual Tracking: An Experimental Survey", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1442–1468, Jul. 2014, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2013.230 (cit. on p. 16).

[51] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, P. H. S. Torr, "Fully-Convolutional Siamese Networks for Object Tracking", *arXiv:1606.09549 [cs]*, Sep. 2016. arXiv: 1606.09549 [cs] (cit. on pp. 16, 32).

[52] B. Li, J. Yan, W. Wu, Z. Zhu, X. Hu, "High Performance Visual Tracking with Siamese Region Proposal Network", en, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT: IEEE, Jun. 2018, pp. 8971–8980, ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00935 (cit. on pp. 16, 32, 69).

[53] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, J. Yan, "SiamRPN++: Evolution of Siamese Visual Tracking with Very Deep Networks", en, *arXiv:1812.11703 [cs]*, Dec. 2018. arXiv: 1812.11703 [cs] (cit. on pp. 16, 32).

[54] A. Sauer, "Tracking Holistic Object Representations", en, p. 12, (cit. on pp. 16, 35).

[55] B. Yan, H. Zhao, D. Wang, H. Lu, X. Yang, "'Skimming-Perusal' Tracking: A Framework for Real-Time and Robust Long-term Tracking", en, *arXiv:1909.01840 [cs]*, Sep. 2019. arXiv: 1909.01840 [cs] (cit. on pp. 17, 69).

[56] P. Besl, H. McKay, "A method for registration of 3-D shapes. IEEE Trans Pattern Anal Mach Intell", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 14, pp. 239–256, Mar. 1992. DOI: 10.1109/34.121791 (cit. on p. 17).

[57] Y. Chen, G. G. Medioni, "Object modeling by registration of multiple range images.", *Image and Vision Computing*, vol. 10, no. 3, pp. 145–155, 1992 (cit. on pp. 17, 37).

[58] Z. Zhang, "Iterative point matching for registration of free-form curves", p. 46, (cit. on p. 17).

[59] F. Lu, E. Milios, "Robot Pose EMstaimtcahtiinogn 2inDURnaknngoewSncaEnnsvironments by", en, p. 38, (cit. on p. 17).

[60] J. Minguez, F. Lamiraux, L. Montesano, "Metric-Based Scan Matching Algorithms for Mobile Robot Displacement Estimation", en, in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain: IEEE, 2005, pp. 3557–3563, ISBN: 978-0-7803-8914-4. DOI: 10.1109/ROBOT.2005.1570661 (cit. on p. 17).

[61] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, ser. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: The MIT Press, 2016, ISBN: 978-0-262-03561-3 (cit. on p. 19).

[62] J. Howard, S. Gugger, "Fastai: A Layered API for Deep Learning", en, *Information*, vol. 11, no. 2, p. 108, Feb. 2020, ISSN: 2078-2489. DOI: 10.3390/info11020108. arXiv: 2002.04688 (cit. on p. 19).

[63] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, R. Girshick, "Detectron2", 2019 (cit. on p. 19).

[64] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library", en, *arXiv:1912.01703 [cs, stat]*, Dec. 2019. arXiv: 1912.01703 [cs, stat] (cit. on pp. 19, 21).

[65] D. P. Kingma, J. Ba, "Adam: A Method for Stochastic Optimization", en, *arXiv:1412.6980 [cs]*, Jan. 2017. arXiv: 1412.6980 [cs] (cit. on p. 20).

[66] L. N. Smith, "Cyclical Learning Rates for Training Neural Networks", en, *arXiv:1506.01186 [cs]*, Apr. 2017. arXiv: 1506.01186 [cs] (cit. on p. 21).

[67] ——, "A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay", en, *arXiv:1803.09820 [cs, stat]*, Apr. 2018. arXiv: 1803.09820 [cs, stat] (cit. on p. 21).

[68] F. Schroff, D. Kalenichenko, J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering", *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, Jun. 2015. DOI: 10.1109/CVPR.2015.7298682. arXiv: 1503.03832 (cit. on p. 21).

[69] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, "TensorFlow: A system for large-scale machine learning", en, p. 21, (cit. on p. 21).

[70] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition", *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989, ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541 (cit. on p. 21).

[71] R. M. Cichy, A. Khosla, D. Pantazis, A. Torralba, A. Oliva, "Comparison of deep neural networks to spatio-temporal cortical dynamics of human visual object recognition reveals hierarchical correspondence", en, *Scientific Reports*, vol. 6, no. 1, pp. 1–13, Jun. 2016, ISSN: 2045-2322. DOI: 10.1038/srep27755 (cit. on p. 21).

[72] O. Parkhi, A. Vedaldi, A. Zisserman, C. Jawahar, "Cats and dogs", ser. Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Jun. 2012, pp. 3498–3505, ISBN: 978-1-4673-1226-4. DOI: 10.1109/CVPR.2012.6248092 (cit. on pp. 21, 27).

[73] K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", en, *arXiv:1409.1556 [cs]*, Apr. 2015. arXiv: 1409.1556 [cs] (cit. on pp. 22, 26).

[74] V. Dumoulin, F. Visin, "A guide to convolution arithmetic for deep learning", en, *arXiv:1603.07285 [cs, stat]*, Jan. 2018. arXiv: 1603.07285 [cs, stat] (cit. on pp. 23, 24, 34).

[75] M. Belkin, D. Hsu, S. Ma, S. Mandal, "Reconciling modern machine learning practice and the bias-variance trade-off", en, *arXiv:1812.11118 [cs, stat]*, Sep. 2019. arXiv: 1812.11118 [cs, stat] (cit. on p. 25).

[76] R. Geirhos, C. Michaelis, F. A. Wichmann, P. Rubisch, M. Bethge, W. Brendel, "IMAGENET-TRAINED CNNS ARE BIASED TOWARDS TEXTURE; INCREASING SHAPE BIAS IMPROVES ACCURACY AND ROBUSTNESS", en, p. 22, 2019 (cit. on p. 25).

[77] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, Y. LeCun, "The Loss Surfaces of Multilayer Networks", en, *arXiv:1412.0233 [cs]*, Jan. 2015. arXiv: 1412.0233 [cs] (cit. on p. 26).

[78] K. He, X. Zhang, S. Ren, J. Sun, "Deep Residual Learning for Image Recognition", *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385 [cs] (cit. on p. 26).

[79] M. Tan, Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", en, *arXiv:1905.11946 [cs, stat]*, Nov. 2019. arXiv: 1905.11946 [cs, stat] (cit. on p. 27).

[80] C. Szegedy, A. Toshev, D. Erhan, "Deep Neural Networks for Object Detection", en, p. 9, (cit. on p. 28).

[81] H. A. Rowley, S. Baluja, T. Kanade, "Neural Network-Based Face Detection", en, p. 6, (cit. on p. 28).

[82] R. Girshick, J. Donahue, T. Darrell, J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, en, *arXiv:1311.2524 [cs]*, Oct. 2014. arXiv: 1311.2524 [cs] (cit. on p. 28).

[83] R. Girshick, “Fast R-CNN”, en, *arXiv:1504.08083 [cs]*, Sep. 2015. arXiv: 1504.08083 [cs] (cit. on p. 29).

[84] J. Long, E. Shelhamer, T. Darrell, “Fully Convolutional Networks for Semantic Segmentation”, en, p. 10, (cit. on p. 29).

[85] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, A. C. Berg, “SSD: Single Shot MultiBox Detector”, en, *arXiv:1512.02325 [cs]*, vol. 9905, pp. 21–37, 2016. DOI: 10.1007/978-3-319-46448-0_2. arXiv: 1512.02325 [cs] (cit. on p. 29).

[86] J. Redmon, A. Farhadi, “YOLOv3: An Incremental Improvement”, en, *arXiv:1804.02767 [cs]*, Apr. 2018. arXiv: 1804.02767 [cs] (cit. on p. 29).

[87] S. Ren, K. He, R. Girshick, J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, en, *arXiv:1506.01497 [cs]*, Jan. 2016. arXiv: 1506.01497 [cs] (cit. on p. 29).

[88] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, K. Murphy, “Speed/accuracy trade-offs for modern convolutional object detectors”, en, *arXiv:1611.10012 [cs]*, Apr. 2017. arXiv: 1611.10012 [cs] (cit. on p. 29).

[89] K. He, G. Gkioxari, P. Dollár, R. Girshick, “Mask R-CNN”, *arXiv:1703.06870 [cs]*, Jan. 2018. arXiv: 1703.06870 [cs] (cit. on p. 30).

[90] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, S. Belongie, “Feature Pyramid Networks for Object Detection”, en, *arXiv:1612.03144 [cs]*, Apr. 2017. arXiv: 1612.03144 [cs] (cit. on p. 30).

[91] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, D. Ramanan, “Object Detection with Discriminatively Trained Part Based Models”, en, p. 20, (cit. on p. 30).

[92] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun, “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks”, en, *arXiv:1312.6229 [cs]*, Feb. 2014. arXiv: 1312.6229 [cs] (cit. on p. 30).

[93] K. He, X. Zhang, S. Ren, J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition”, en, *arXiv:1406.4729 [cs]*, vol. 8691, pp. 346–361, 2014. DOI: 10.1007/978-3-319-10578-9_23. arXiv: 1406.4729 [cs] (cit. on p. 30).

[94] X. Yan, Z. Chen, A. Xu, X. Wang, X. Liang, L. Lin, “Meta R-CNN : Towards General Solver for Instance-level Few-shot Learning”, en, *arXiv:1909.13032 [cs]*, Mar. 2020. arXiv: 1909.13032 [cs] (cit. on p. 31).

[95] Li Fei-Fei, R. Fergus, P. Perona, “One-shot learning of object categories”, en, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 594–611, Apr. 2006, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2006.79 (cit. on p. 31).

[96] G. Koch, R. Zemel, R. Salakhutdinov, “Siamese Neural Networks for One-shot Image Recognition”, en, p. 8, (cit. on pp. 31, 32).

[97] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, R. Shah, “Signature Verification using a ßiamese”time delay neural network”, en, p. 8, (cit. on p. 31).

[98] S. Chopra, R. Hadsell, Y. LeCun, "Learning a Similarity Metric Discriminatively, with Application to Face Verification", en, in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, San Diego, CA, USA: IEEE, 2005, pp. 539–546, ISBN: 978-0-7695-2372-9. DOI: 10.1109/CVPR.2005.202 (cit. on p. 32).

[99] Y. Taigman, M. Yang, M. Ranzato, L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification", en, in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA: IEEE, Jun. 2014, pp. 1701–1708, ISBN: 978-1-4799-5118-5. DOI: 10.1109/CVPR.2014.220 (cit. on p. 32).

[100] T.-Y. Lin, Yin Cui, S. Belongie, J. Hays, "Learning deep representations for ground-to-aerial geolocalization", en, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA: IEEE, Jun. 2015, pp. 5007–5015, ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7299135 (cit. on p. 32).

[101] J. Žbontar, Y. LeCun, "Computing the Stereo Matching Cost with a Convolutional Neural Network", en, *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1592–1599, Jun. 2015. DOI: 10.1109/CVPR.2015.7298767. arXiv: 1409.4326 (cit. on p. 32).

[102] B. X. Chen, J. K. Tsotsos, "Fast Visual Object Tracking with Rotated Bounding Boxes", en, *arXiv:1907.03892 [cs]*, Sep. 2019. arXiv: 1907.03892 [cs] (cit. on pp. 32–34).

[103] D. Driess, O. Oguz, J.-S. Ha, M. Toussaint, "Deep Visual Heuristics: Learning Feasibility of Mixed-Integer Programs for Manipulation Planning", en, p. 7, (cit. on p. 39).

[104] Buckl, Katharina, *Sensor Fusion using the Kalman Filter*, Mar. 2004 (cit. on p. 49).

[105] S. Rusinkiewicz, M. Levoy, "Efficient variants of the ICP algorithm", en, in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, Quebec City, Que., Canada: IEEE Comput. Soc, 2001, pp. 145–152, ISBN: 978-0-7695-0984-6. DOI: 10.1109/IM.2001.924423 (cit. on p. 65).

[106] G. Bradski, "The OpenCV library", *Dr. Dobb's Journal of Software Tools*, 2000 (cit. on p. 69).

[107] S. Caelles, J. Pont-Tuset, F. Perazzi, A. Montes, K.-K. Maninis, L. Van Gool, "The 2019 DAVIS Challenge on VOS: Unsupervised Multi-Object Segmentation", en, *arXiv:1905.00737 [cs]*, May 2019. arXiv: 1905.00737 [cs] (cit. on p. 69).

[108] V. Sarode, X. Li, H. Goforth, Y. Aoki, R. A. Srivatsan, S. Lucey, H. Choset, "PCRNet: Point Cloud Registration Network using PointNet Encoding", en, *arXiv:1908.07906 [cs]*, Nov. 2019. arXiv: 1908.07906 [cs] (cit. on p. 70).

[109] Y. Aoki, H. Goforth, R. A. Srivatsan, S. Lucey, "PointNetLK: Robust & Efficient Point Cloud Registration using PointNet", en, *arXiv:1903.05711 [cs]*, Apr. 2019. arXiv: 1903.05711 [cs] (cit. on p. 70).

[110] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes", en, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA: IEEE, Jun. 2015, pp. 1912–1920, ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298801 (cit. on p. 70).

[111] O. Dovrat, I. Lang, S. Avidan, "Learning to Sample", en, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA: IEEE, Jun. 2019, pp. 2755–2764, ISBN: 978-1-72813-293-8. DOI: 10.1109/CVPR.2019.00287 (cit. on p. 70).

[112] I. Lang, A. Manor, S. Avidan, "SampleNet: Differentiable Point Cloud Sampling", *arXiv:1912.03663 [cs]*, Dec. 2019. arXiv: 1912.03663 [cs] (cit. on p. 70).

[113] Y. Zhou, C. Barnes, J. Lu, J. Yang, H. Li, “On the Continuity of Rotation Representations in Neural Networks”, en, *arXiv:1812.07035 [cs, stat]*, Apr. 2019. arXiv: 1812.07035 [cs, stat] (cit. on p. 70).

All links were last followed on April 30, 2020.

Acknowledgements

Herein, I want to express my gratitude to people who made this work possible. First and foremost, I want to thank Prof. Marc Toussaint for his guidance during this work. I am thankful for his farsight throughout my whole course of study in teaching material which serves as a great foundation for various fields in modern machine learning and robotics. Further, I want to thank M. Sc. Danny Driess for his feedback and his valuable ideas through the process of researching and writing this thesis. I would also like to thank Julian Hörz for the constructive and fun cooperation.

I want to thank M. Sc. Ralf Gulde for his support and insightful discussions. Finally, I want to express mit gratitude to my family for supporting me throughout my entire life. I am especially grateful for the consistent support and inspiring motivation of my beloved wife Angela.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature