Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Matias Hagen Myrestrand

Recognition and Motion Tracking of 3D Objects

Master's thesis in Cybernetics and Robotics
Supervisor: Jan Tommy Gravdahl, ITK
Co-supervisor: Marialena Vagia, SINTEF, Klaus Ening, SINTEF
June 2021

NTNU
Norwegian University of Science and Technology

SINTEF

Matias Hagen Myrestrand

Recognition and Motion Tracking of 3D Objects

Master's thesis in Cybernetics and Robotics
Supervisor: Jan Tommy Gravdahl, ITK
Co-supervisor: Marialena Vagia, SINTEF, Klaus Ening, SINTEF
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Preface

The following master's thesis was done in collaboration with SINTEF Mathematics and Cybernetics, and the Department of Engineering Cybernetics at the Norwegian University of Science and Technology. The work can be seen as a continuation of the preliminary project thesis [29] with the same title. In order to give a fuller understanding of both the problem and the proposed solution, some parts from [29] have been added to this thesis. With the exception of some adjustments, this includes the introduction in section 1, along with the following subsections; 2.1-2.2, 2.4-2.5, 4.1-4.3 and 5.1.1-5.1.2.

I would very much like to thank my supervisors Marialena Vagia and Jan Tommy Gravdahl for some great guidance along the way. Your expertise have been of great value.

Also, special thanks are due to my supervisor Klaus Ening for setting up the lab, creating 3D models, and always being available for technical assistance. This work would not have been possible without your help.

Abstract

In this thesis, a highly viable framework solution is presented for both object recognition and motion tracking. Based on a literature study and experience from the preliminary project thesis, a revised solution is first proposed. This solution includes the multi-modality LineMOD detector and a subsequent translation clustering for determining the position of potential object matches. Combined with a unique strategy for finding the correct object rotation, the detector solution is able to provide initial object poses with increased precision. The revised solution furthermore proposes a novel state-of-the-art region-based Gaussian tracker (RBGT) for estimating the pose of detected objects. Despite showing some very promising results, the RBGT still struggles in some cases due to inadequate appearance models and contour ambiguity. Consequently, some additional strategies are suggested in order to improve the overall performance of the framework. This includes a sparse, yet efficient approach for utilizing depth image information, complementing the color-only RBGT. In addition, a solution for drift detection and correction is proposed, which further improves the robustness and precision of the tracker. The results of these additions are showcased in multiple experiments, demonstrating an overall improvement in tracking performance.

Contributions

All necessary equipment, including the lab setup was provided by SINTEF. The main contributions by the author are listed below.

- A literature survey on existing methods for 6-DoF object detection and tracking.
- The implementation of a complete detection and 6-DoF pose estimation pipeline, which combines existing methods with some novel techniques proposed by the author.
- A novel approach for integrating depth data utilization for the color-only Region-Based Gaussian Tracker (RBGT).
- A unique uncertainty-driven evaluation scheme for finding the best initial pose candidates during detection.
- A solution for detecting and correcting imprecise rotation estimates.

Supplementary Material

Filename	Type	Description
1. depth	mp4	Tracking run using depth information (figure 50).
2. noDepth	mp4	Tracking run not using depth information (figure 51).
3. driftCorrection	mp4	Challenging tracking run with drift correction (figure 54).
4. noDriftCorrection	mp4	Challenging tracking run without drift correction (figure 54).
5. BothParts	mp4	Tracking run including both the big and the small chair part.
6. projectThesis	pdf	The preliminary project thesis [29]

Abbreviations

ACCV Asian Conference on Computer Vision

ANN Approximate Nearest Neighbour

Assimp Open Asset Import library

DNN Deep Neural Network

DoF Degrees of Freedom

GLEW OpenGL Extension Wrangler Library

GLFW Graphics Library Framework

ICP Iterative Closest Point

IR Infrared

LiDAR Light Detection And Ranging

OpenCV Open Source Computer Vision Library

OpenGL Open Graphics Library

ORK Object Recognition Kitchen

RBGT Region Based Gaussian Tracker

RBOT Region Based Object Tracking

RGB Red, Green, Blue

RGB-D Red, Green, Blue and Depth

SDL Simple DirectMedia Layer

List of symbols

LineMOD

$D(\boldsymbol{x})$	Depth at image coordinate $\boldsymbol{x}$
ε_D,ε_G	Similarity measure for surface normals and color gradients
f_m	Similarity function
$\mathcal{I}$	Input image
$\mathcal{J}$	Binarized image of spread orientations
m	Modality
n_0	Number of quantized gradient orientations
$\mathcal{O}_m$	Template image of modality m
$\boldsymbol{ori}(\mathcal{O}, r)$	Gradient orientation for image $\mathcal{O}$ at location r
$\mathcal{P}$	List of pairs (r,m)
r	Location of discriminant feature
$\mathcal{R}(r + c)$	Pixel neighbourhood with $r + c$ as midpoint
$\mathcal{S}_i$	Response map for orientation i
$\mathcal{T}$	Template
T	Sampling step
τ_i	Precomputed lookup table for orientation similarity
$\vec{v}(\boldsymbol{x}_i)$	Vector along line of sight towards 3D point given by $\boldsymbol{x}$
$\boldsymbol{x}$	Image coordinate
$\boldsymbol{X}$	Projected 3D point

RBGT

α_b,α_f	Learning rate for background and foreground model
C	Camera reference frame
$\boldsymbol{c}_i$	Center for correspondence line i
Δc_i^+	Projected difference from correspondence line center $\boldsymbol{c}$
Δc_{si}^+	Scaled projected difference
$\Delta \tilde{c}_{si}$	Discretized projected difference
$\boldsymbol{\mathcal{D}}_i$	Color data along correspondence line i
f_x, f_y	Focal lengths
h_b,h_f	Smoothed step function for background and foreground
$\boldsymbol{I}$	Input color image
i_t	Closest template view
λ_r,λ_t	Regularization parameters for rotation and translation
M	Model reference frame
m_b,m_f	Background and foreground model
n_{cl}	Number of correspondence lines for a given template view
$\boldsymbol{n}_i$	Normal vector for correspondence line i
$\bar{n}_i$	Largest normal component
$\boldsymbol{N}_i$	Contour of object in current image
$p_j(\boldsymbol{y}_i)$	Pixel-wise posterior for model j
p_x, p_y	Principal point coordinates
$\boldsymbol{\pi}$	Pinhole camera model
$\boldsymbol{R}$	Rotation matrix $\in \mathbb{SO}(3)$
r	Distance from correspondence line center $\boldsymbol{c}$
r_s	Scaled distance from correspondence line center $\boldsymbol{c}$
s	Number of pixels combined into discretized segment (scale)
s_h	Slope parameter for the smoothed step function
$\boldsymbol{t}$	Translation vector

${}_C\boldsymbol{T}_M$	Homogeneous transformation matrix from M to C
$\boldsymbol{\theta}$	Full variation vector
$\boldsymbol{\theta}_r$	Rotation variation
$\boldsymbol{\theta}_t$	Translation variation
$\boldsymbol{x}_{cli}(r)$	Function returning the rounded image coordinate given r
$\boldsymbol{x}_i$	Image coordinate
$\boldsymbol{X}_i$	3D model point
$\tilde{\mathbf{X}}_i$	Homogeneous 3D model point
${}_C\tilde{\mathbf{X}}_i^+$	Variated model point in C caused by $\boldsymbol{\theta}$
$\boldsymbol{y}_i$	RBG color values at image coordinate $\boldsymbol{x}_i$

RBGT Additions

$\boldsymbol{\mathcal{E}}_i$	Depth data along correspondence line i
$\mu_{\mathcal{E}_i}$	Position of a detected depth discontinuity
r_d	Depth-to-color influence ratio for the combined likelihoods
$\sigma_{\mathcal{E}}$	Standard deviation for Gaussian edge likelihoods
$\boldsymbol{\mathcal{T}}_i$	Total of data along correspondence line i
t_{dd}	Lower threshold for depth discontinuities
t_{drift}	Threshold for drift correction
t_z	Upper threshold for number of zeros (missing data) in $\boldsymbol{\mathcal{E}}_i$

Contents

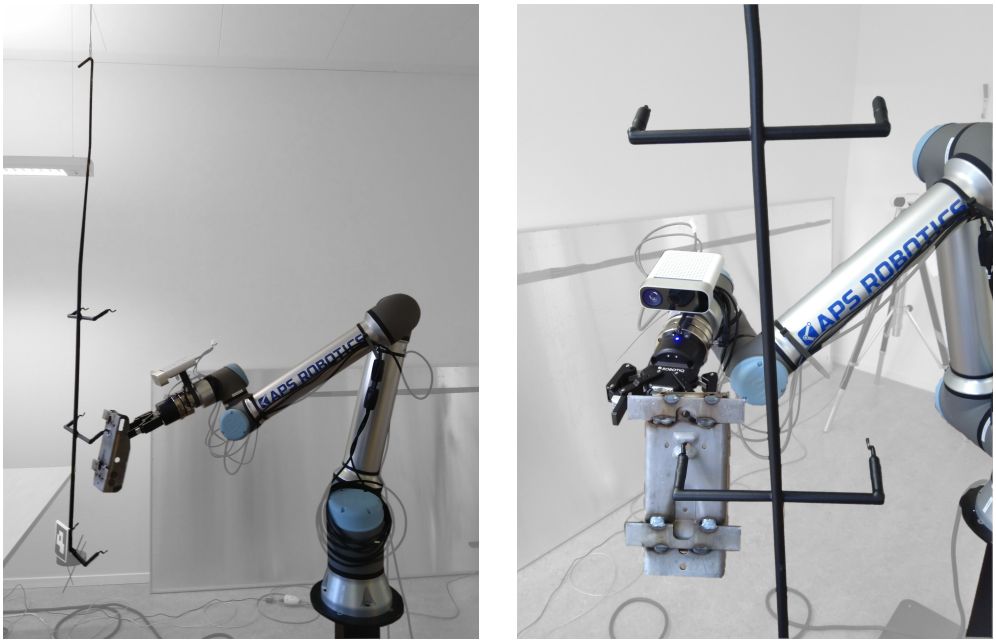
1 Introduction

Object recognition and 6-DoF pose estimation make up some of the most prominent fields in computer vision and robotics today. With the arrival of more complex use cases of robotic technology such as autonomous driving, or even medical robots performing surgery procedures [14], the systems ability to accurately perceive their surroundings is essential. Industrial assembly robots are examples of autonomous systems which require information about their environment in order to interact with it. RGB cameras, IR depth cameras and LiDAR are all examples of optical sensors that can provide the raw data needed to obtain this information. By utilizing the techniques of object recognition and pose estimation, these systems are able to discover and localize target objects or potential obstacles. Industrial- and inspecting robots, modern visual surveillance [36], augmented reality [64] and intelligent transportation [32] are just some of the applications relying on the techniques of both object recognition and motion tracking.

This thesis will mainly focus on the industrial application of these techniques. As we know, robots are widely used in manufacturing performing task such as pick and place, assembly, packaging and painting. All of these tasks require an accurate identification and pose estimate of the object to be handled. Therefore, robotic vision has great importance in automated industrial processes that otherwise would call for human intervention. Reduced costs, increased production, improved consistency and safety are some of the benefits from vision guided robotic systems [30, 47, 13]. However, recognition and motion tracking still pose some notable challenges. For instance, varying lighting conditions with shadows and glare, occlusion of objects and motion blur can all complicate the tasks of object detection and 6-DoF pose estimation [33]. Furthermore, when moving objects are added to the equation, an additional demand for real-time speed must be taken into account. This demand illustrates a crucial challenge in pose estimation as a trade-off between accuracy and computational cost is inevitable.

The problem to be solved in this thesis addresses to a chair manufacturing setting and involves a UR10 robotic manipulator responsible for loading and unloading objects off a hanger. The hanger, which is suspended from a roof mounted conveyor, is swinging freely. This manufacturing setting is recreated in a lab setup as illustrated in figure 1. The camera, an *Azure Kinect DK*, is mounted on the UR10 and provides both color and depth (RGB-D) vision. In order to enable the use of the robotic manipulator it is important to recognize and track the 3D object that we need the arm to interact with. Hence, a flexible solution for 3D tracking of different objects is needed. Figure 2 illustrates the chair parts that will be used for the object tracking. These objects have not undergone any paint job at this point in the assembly process, and will consequently have a varying surface texture.
The nature of this problem will be the basis for the literature study in section 2, where the viability of relevant existing recognition and 6-DoF estimation techniques will be discussed before proposing a feasible solution in section 3. Next, the theory behind each method is described in section 4 before discussing the initial implementation in section 5. After evaluating the implemented tracker in section 6, some solution improvements are then proposed in section 7. Thereafter, experiments are performed for both the object

detector and the tracker in section 8. Finally, some discussion and concluding remarks are given on the implemented solution in section 9 and 10 respectively.



(a) (b)

Figure 1: Lab setup for problem description **a)** Side view of the robotic manipulator UR10 and hanger. **b)** Closeup of robotic manipulator loading object onto hanger. The Azure Kinect DK is mounted on top of the robotic arm.

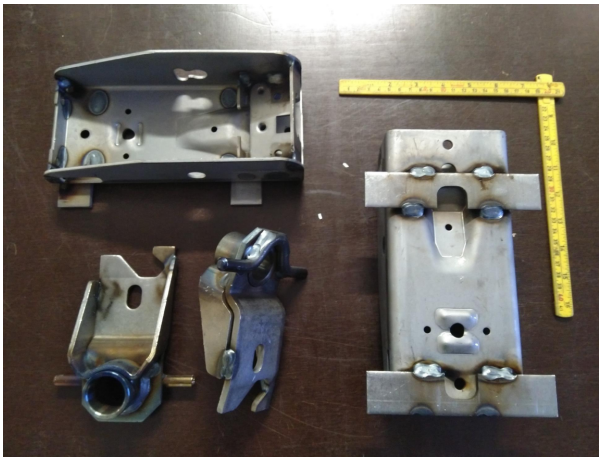


Figure 2: Objects to track.

2 Literature study

In this section we do a brief study of existing 6-DoF estimation methods, while also discussing the suitability of these methods in relation to the problem given in section 1. The study starts off by taking a look at feature based methods in 2.1, before discussing template matching in 2.2, region based methods in 2.3, learning-based methods in 2.4, and point cloud based methods in 2.5. This study, along with the experience from the preliminary project thesis [29], will later be the basis for a revised framework proposal in section 3.

2.1 Feature Based Methods

Feature based methods make up a variety of techniques utilizing local image descriptors to match keypoints between the scene and textured target objects. Using either a monocular RGB camera, or a multi-view stereo vision setup, the 2D key points in image coordinates are back-projected to 3D before retrieving the 6-DoF pose of the object based on these point-to-point correspondences. SIFT [26] and SURF [3] are both examples of feature detection algorithms which describe and detect local features in images. In short, these apply gradient magnitudes and orientations relative to the keypoint's orientation, making the descriptor invariant to rotation. The transition from n 2D-3D point correspondences to a 6-DoF pose is defined as a *Perspective-n-point* (PnP) problem. Gordon and Lowe [12] presented a 3D object pose estimation framework back in 2006, using SIFT correspondences between the scene and a 3D model of the object, and thereafter solving the following PnP problem. In order to prevent false matches the RANSAC algorithm [10] is also implemented, which removes outliers from potential 2D-3D correspondences. For further pose refinement the Levenberg-Marquardt algorithm [25] can also be applied, as was done in [12], minimizing the geometric reprojection error. In total this framework can provide a speedy 3D object detection and pose estimate for textured objects when provided corresponding 3D object coordinates for each matching 2D scene point.
As a starting point of the preliminary project thesis [29] a similar framework was implemented using ORB [43], an efficient alternative to SIFT or SURF, and *solvePnPRansac* which combines RANSAC and a PnP-solver to give an estimated pose in 6-DoF. Both of these are provided by the *Open Source Computer Vision Library* (OpenCV). The initial tests were carried out using a planar flyer as tracking object, giving a simple translation from image to 3D object coordinates. An illustration of the pose estimation result is shown in figure 3. This testing was for research purposes only, and will not be discussed in the remaining thesis. This solution did however illustrate the importance of explicit features in order to find point correspondences. As for SIFT and SURF, ORB can also be vulnerable to illumination changes such as shadows and glare. Consequently, the flyer pose estimation solution was only able to track its pose correctly in a span of ± 30 degrees in *roll*, *pitch* and *yaw*.

As stated in the problem description in section 1, the target objects will have a varying surface texture. Naturally, this does not make the ideal conditions for a local feature descriptor as each sample object will give rise to different keyponts. Self induced shadows and a general lack of explicit features would further degrate the performance of a feature

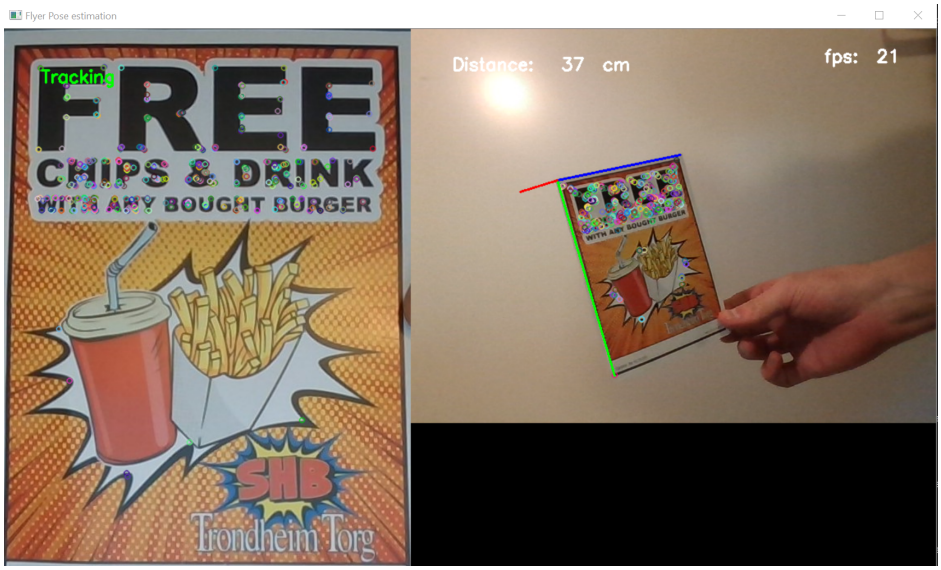


Figure 3: 6-DoF pose estimation of a textured flyer. Each colored circle represent an ORB keypoint available for matching.

based framework if applied to this problem.

2.2 Template Based Methods

Template matching is another common approach for object detection and pose estimation. In contrast to the feature based methods that use local feature descriptors, this approach uses object descriptors. An object descriptor encode the entire observed object based on a given modality, such as color gradients. The similarity score between a scene image and a set of templates will hence decide if an object is present or not. During training, these template images are obtained through sampling from different viewpoints. This way the object's pose can also be determined based on the pose of the template match with the highest similarity score. However, in order to achieve high resolution pose estimates with this approach, a huge set of template images would be required. A trade-off between pose resolution, memory consumption and search speed is inevitable. As a result, pose refinement algorithms like *Iterative Closest Point* (ICP) [4] have been used to improve the initial pose estimate. This method operates on point clouds and can only be applied when depth images are available. More details on ICP will be given in the subsection on point cloud based methods in 2.5. For pose estimation setups using RGB images only, the minimization of photometric energy functions have been used to refine the initial pose estimates, as was done in [54].

2.2.1 LineMOD

LineMOD [19] is a well known method in template based pose estimation which uses multiple modalities. The framework combines both RGB images and dense depth maps, also known as RGB-D images, to give complimentary information on an object. As demonstrated in [19], the combination of a color gradient descriptor and a surface normal descriptor makes a robust template representation. An illustration of these modalities is shown in figure 4. If the surface normals are omitted from the set of modalities, we get the more generic LINE-2D [18] method. As the figure shows, the color gradients are

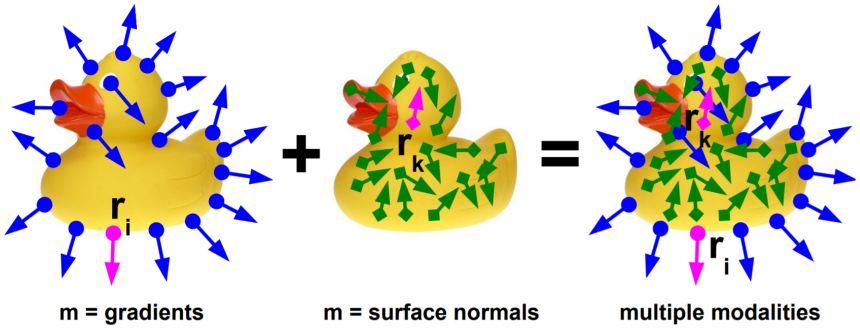


Figure 4: A rubber duck with with different modalities, m. (Source: [19])

mainly located on the contours, while the surface normals are located on the body of the object. Unlike the local feature descriptors discussed in 2.1, this template representation approach will also be able to detect texture-less 3D objects. The contours are naturally also more robust to illumination changes and noise compared to local feature keypoint found on the body. For our problem, described in section 1, a LineMOD based template representation seems to be a viable option for obtaining a decent initial pose estimate. As discussed, this solution would however still need a pose refinement step. Given that this method already makes use of depth images, the ICP-required cloud points would be easily accessible. Hence ICP could be implemented and return the final 6-DoF pose for all detected objects.

Despite claiming real-time performance for multiple object detection and pose estimation in [19], the efficiency of LineMOD has been discussed in various papers proposing improvements on the pipeline. The exhaustive nearest neighbour search used for finding the most similar template match is definetely not ideal. Shao et al. [46] recognize this, but still proclaim this method to achieve real-time speed for single object pose estimation. They also discuss *Approximate Nearest Neighbour* (ANN) techniques such as *hashing-based* and *tree-based* matching, while proposing a modified *fuzzy decision forest* framework for improved matching efficiency. Although both hashing-based and tree-based methods have sub-linear complexity for searching, these still have some drawbacks. For one, the design of an efficient hash function is often not trivial [46]. Shao et al. also points out the efficiency suffering related to *the course of dimensionality* due to backtracking for the tree structure. Regardless, [23, 20] and [46, 41] all show improved efficiency on the LineMOD ACCV12 dataset [19] by applying hashing-based and tree-based methods

respectively.

2.2.2 Template Generating

The task of generating a set of templates naturally applies to all 6-DoF object pose estimation frameworks based on template matching. As mentioned, a large number of template samples from different viewpoints are needed in order to recognize an object and give a decent initial pose estimate. In [15] a total of 12960 templates are used per object. These are rendered from 216 viewpoints uniformly distributed on a synthetic sphere around the target object. For each viewpoint the camera is rotated around the optical axis from $-60°$ to $+60°$ with a step of $10°$. Finally this is repeated for 5 different spheres with varying radii with a step of $0.1m$. A similar setup is used in [46], except they only use the upper hemisphere for sampling as shown in figure 5. The general approach to generate these sets of templates is to synthetically render a 3D mesh model of the target object. These models can for instance be obtained by scanning the object. This way we can obtain flexible template based frameworks for 6-DoF object pose estimation, as a mesh model is the only requirement for tracking a new object.

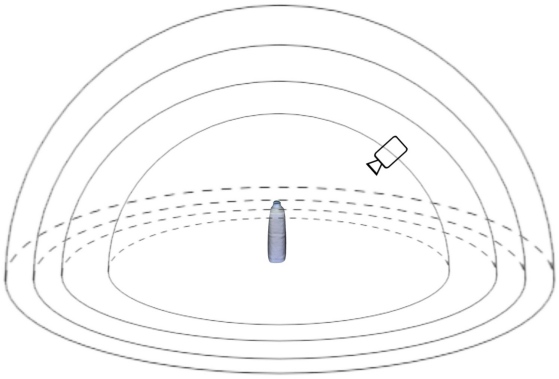


Figure 5: A synthetic rendering sample of a bottle. This setup contains four different hemispheres with varying radii. (Source: [46])

2.3 Region Based Methods

Region based methods, often using RGB images only, constitute a different subcategory of pose estimation strategies. Simplified, these apply descriptors such as color histograms to differentiate between the foreground, i.e. the object, and the background. Thereafter, the pose which best fit the object contour is found through the optimization of some kind of energy function. Just as for LineMOD, region based methods can be used for estimating the pose of texture-less 3D objects. PWP3D [34] by Prisacariu and Reid is maybe the most famous work among all region-based methods. It builds on [42, 45, 8] along with the pixel-wise color histogram posterior membership approach presented by Bibby et al. in [5]. The method, which uses a *signed distance embedding function* as

energy function, was the first region based method capable of real-time performance. Several enhancements have later been suggested to improve the efficiency and robustness of the algorithm. Among these we find [53] and [16] which both perform better with cluttered backgrounds. Inspired by [24], [16] achieves this by introducing local appearance models that are better at capturing spatial variation. In addition, [53] reduced the overall runtime by improving the optimization procedure. Based on the works of both [53] and [16], Tjaden et al. present an improved approach in [54], using *temporally consistent local color histograms* along the contours of the objects. An illustration of this technique is shown in figure 6.

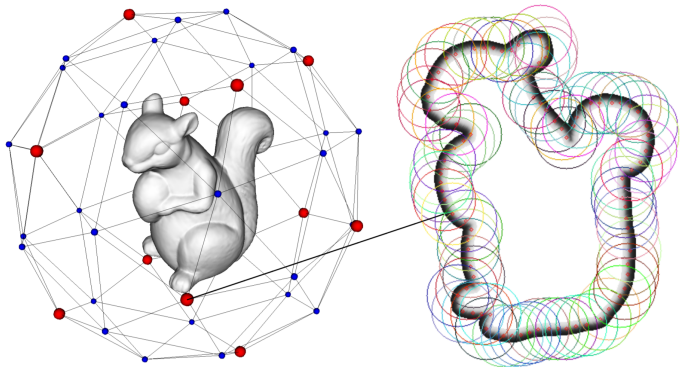


Figure 6: Projected contour of 3D object from a given template view. The local color histogram regions are illustrated by colored circles. (Source: [54])

Zhong et al. propose a different method in [63] inspired by [62]. Using *temporally consistent polar-based region partitioning* and *edge-based occlusion detection* this method clearly outperforms [54] on the RBOT dataset [55]. Another method showing some very interesting results is presented in [48]. The method was displayed at the *Asian Conference on Computer Vision* (ACCV) in late November 2020 and is the most recent work included in this literature study. While maintaining the global segmentation model of PWP3D, Stoiber et al. introduce a highly efficient and sparse Gaussian approach to region-based tracking. The method also performs better than [55], [63] and [54] on the RBOT dataset. Some other works based on the concepts of PWP3D include [22] which adds a term based on the ICP algorithm to the energy function in order to incorporate depth information, and [35] which integrate orientation information from an inertial sensor. These methods also introduce enhancements with respect to efficiency by suggesting a sparse calculation of the energy function, and a Levenberg-Marquardt based optimization approach respectively. [55] otherwise suggests a Gauss-Newton approach for speeding up the optimization.

If considering region based methods as a whole, it is not unreasonable to assume that the general lack of complementary depth descriptors can make them more prone to false matches and drifting. For instance, the pose optimization step, which might utilize something similar to a two-dimensional signed distance function as in PWP3D, will naturally be less robust in terms of ambiguity compared to the ICP algorithm which operates on 3D cloud points. To put it strongly, the shape of the object will have no importance

as long as the contours match the ones of a rendered template view. For object with less distinct silhouettes than those from [54, 6] and figure 6, it is reasonable to expect a somewhat impaired result in terms pose estimation. The intended objects to track from figure 2 would also lose a lot of their characteristics if only considering their silhouettes. Based on this brief assessment, no region based approaches were really considered during the preliminary project thesis. However, for reasons to be discussed in 3.1, along with the arrival of the very promising Region Based Gaussian Tracker (RGBT) [48], it seemed fitting to take a closer look at some of these methods. Hence, a short rundown of [54] and [48] is given in 2.3.1 and 2.3.2 respectively.

2.3.1 Temporally Consistent Local Color Histogram Pose Estimation

By incorporating the improved optimizing procedure presented in [53] and the local segmentation idea from [16], this method offers a real-time pose tracking approach which unlike the aforementioned region based methods also provides a solution for pose detection. As the unique object descriptor is used for both template matching and pose optimization, no initial object pose is required for starting the tracking process. Furthermore, the temporally consistent, local color histograms (tclc-histograms) enable the approach to recover from accidental tracking loss. Inspired by LINE-2D [18], *posterior response maps* are also introduced to speed up the pose detection approach. The binary overlap between this representation and the silhouette masks of the rendered templates are applied in order to skip image regions that are less likely to contain the object. The pose recovery detection after temporally tracking loss is especially favorable in scenes with great clutter and occlusion as shown in figure 7.

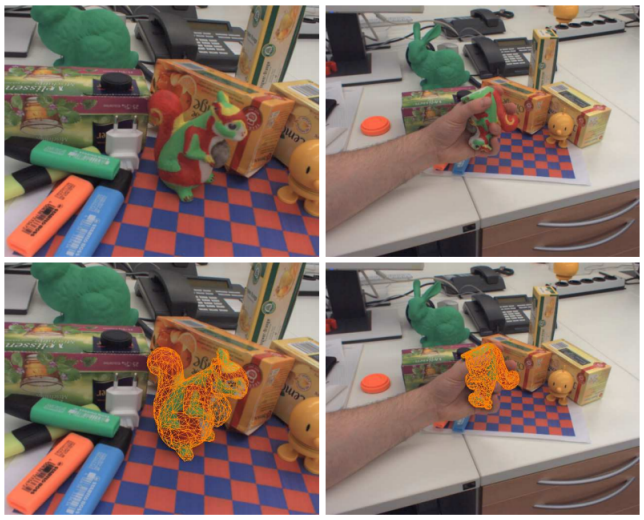


Figure 7: Pose estimation results in a cluttered scene with significant occlusion. (Source: [54])

A drawback of this method is that it requires background knowledge, and must hence be trained in the intended scene to outperform the more generic LINE-2D approach. If operating in known scenes, as suggested in the initial problem description, this could be acceptable. However, Gaussian based methods like [6] and [48] still perform better on the ACCV dataset [54] and the RBOT dataset respectively compared to their approach with scene knowledge. Consequently, methods such as the RBGT [48] might seem like a more viable option.

2.3.2 RBGT

The Region Based Gaussian Tracker (RBGT) from [48] is a highly efficient sparse tracker that like most other region based methods only requires a monocular RGB camera and a 3D object model to start the tracking. The main novelty of the method is a probabilistic model that considers pixel color information sparsely along *correspondence lines* as illustrated in figure 8.

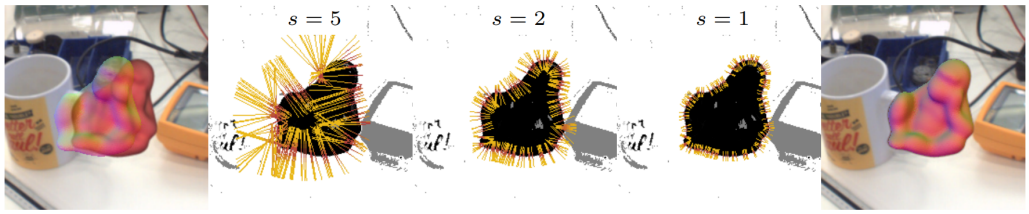


Figure 8: Example of the optimization process while tracking the *ape* object from the RBOT dataset. The leftmost image displays a rendered overlay before the optimization while the rightmost image displays the same overlay updated with the optimized 6DoF pose. The images in the middle visualize the pixel-wise posteriors describing the probability of a pixel belonging to the background. White pixels indicate $p_b = 1$ while black pixels indicate $p_b = 0$. The orange lines illustrate the correspondence lines converging towards the final pose with decreasing scale s. Line segments with high contour probabilities are illustrated in red. (Source: [48])

In addition to providing a discrete scale-space formulation for improved computational efficiency, they derive a mathematical proof that shows that the proposed likelihood function follows a Gaussian distribution. Based on this information, robust approximations for the derivatives of the log-likelihood are presented. Using a regularized Newton optimization this approach outperforms state-of-the-art region based methods in terms of tracking success while also being about one order of magnitude faster [48]. Although this approach does not provide object detection like [54], its performance and efficiency still make it very appealing. Having the slight doubt for region based methods in mind, it would be interesting to see how it handles challenges like contour ambiguity during difficult rotations.

2.4 Learning Based Methods

Learning based methods make up a different approach to 6-DoF pose estimation which generalize better to variations in viewpoint and slight shape deformations. [41] from 2.2, which extended LineMOD by introducing an efficient tree-based search for template matching, is an example of such a method, as the templates are learned in a discriminative fashion. In general, learning based methods often evoke less false positives than nearest neighbour approaches such as the exhaustive LineMOD search from [19]. However, as stated in [46], their efficiency often depends on the quality of negative training samples. If trained for one specific scene, the performance may not be transferable to others. This should also be considered when being presented with results from a learning based methods. As with the temporally consistent local color histogram approach from [54], these have often been trained on that particular dataset to achieve the best performance for that scene.
Some of the latest and most prominent related works include PointNet [37], which directly uses point clouds for object classification and segmentation, and [56] which propose a method for human pose estimation called DeepPose. Both are based on *Deep Neural Network* (DNN) architecture. Gao et al. also present a method for 6-DoF object pose estimation in [11] based on both PointNet and ICP for pose refinement. As for most template and learning based methods, the trade-off between efficiency and accuracy may hurt the performance of real-time systems with moving objects. For instance, [11] demonstrate an average processing time of $0.41s$ for a single object image when running on a Nvidea Titan X GPU. This naturally would not be sufficient for a real-time system unless we were dealing with stationary objects. DOPE [57] and PoseCNN [60], both using DNN architecture, also fail to meet the real-time requirements for dynamic tracking. However, [57] present a novel synthetic data generation procedure which enables a more flexible training setup with pre-labeled data. This way, the cumbersome process of image assembling and labeling is avoided.

To summarize, learning based methods make up a variety of approaches. While typically generalizing better to variations such as shape deformations, their performance often rely on scene specific training and the quality of negative training samples. Furthermore, the popular DNN based architectures introduce high computational complexity, making these methods unfit for the dynamic tracking problem described in section 1. On the other side, approaches such as the tree-based LineMOD extension [46] can actually result in increased efficiency.

2.5 Point Cloud Based Methods

The entering of low-cost 3D cameras in the market has resulted in increased focus on approaches that operate directly on 3D point clouds. The 3D object classifier PointNet [37] from 2.4, and the LineMOD surface normal descriptor from 2.2.1 both utilize 3D point data, making these invariant to object texture and illumination changes. Methods employing depth data exclusively are mainly used for pose refinement or template matching. The previously mentioned ICP algorithm [4] is an example of the former.

2.5.1 ICP

ICP, or Iterative Closest Point, differs from most object pose estimation methods as it does not detect the object. Instead, ICP uses an iterative scheme to align two cloud points. This geometric optimization will hence find the translation and rotation that minimize the distances between corresponding object points in 3D. However, a decent initial guess or estimate of the objects pose is required. The algorithm is sensitive to both the initial pose and sensor noise, which in turn can result in convergence to local optima. In order to reduce sensor noise Ruotao He et al. [15] use a moving least squares algorithm [2] for smoothing scene points before applying ICP. For template based 3D object detection and pose estimation frameworks, such as [15, 19, 46, 23], ICP is often applied as the template matching alone won't give a sufficiently accurate object pose estimate.

2.5.2 Oriented Point Pairs

A different category of point cloud based approaches is presented by Drost et al. in [9]. Using oriented point pair features, they create global model descriptions of the objects, which are later matched using a voting scheme. The features describe relative position and orientation of two point normals, as illustrated in figure 9.

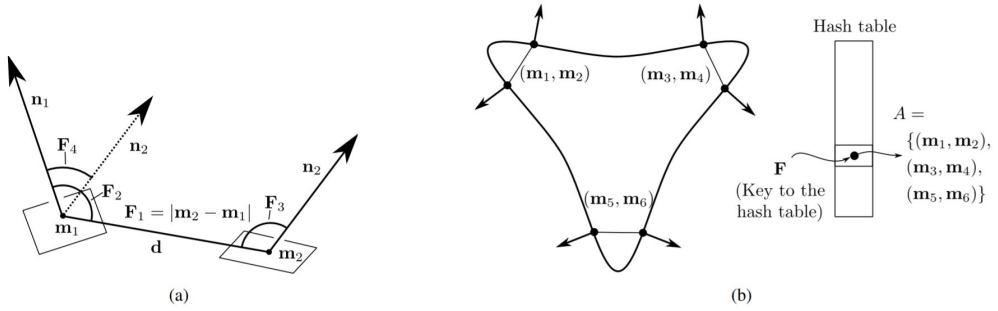


Figure 9: **a)** Point pair feature $\mathbf{F}$ of two oriented points. $\mathbf{F_1}$ is set to the distance between the points, $\mathbf{F_2}$ and $\mathbf{F_3}$ equals the angle between the normals and the vector defined by the two points. Finally, $\mathbf{F_4}$ is set to the angle between the two normals. **b)** The global model description. Point pairs with similar vector $\mathbf{F}$ are stored in the same slot in the hash table. (Source: [9])

By analysing point pair features from the object scene, this method can detect the target objects while simultaneously output probable poses in 6-DoF. For increased stability Drost et al. also use pose clustering which removes isolated poses with low scores. Similar techniques are applied in template based methods such as [15]. Unlike this LineMOD based approach however, [9] is not refined by methods such as ICP. Nevertheless, when requesting high precision, this framework will pay the price in terms of high processing time. For our intended real-time application this naturally won't be desirable. On the other hand, it offer a somewhat flexible solution, as a 3D model is the only thing required for tracking a new object.

3 Proposed Method

In this section the experience from the preliminary thesis project [29] will be summarized briefly in 3.1 before providing a revised framework proposal on the basis of this experience and the preceding literature study in section 2.

3.1 Preliminary Proposal

For the preliminary thesis project [29] the LineMOD approach [19] was evaluated. This template based method had been proven to work for texture-less objects while also being robust to illumination changes due to the utilization of depth data. In addition, works like [46] and [23] demonstrated that it can be made more efficient by proposing new template matching strategies. Last but not least, the synthetic template generating approach from 2.2.2 also seemed practical, as it provides a flexible solution for multiple object tracking. Consequently, combining a LineMOD detector with template match clustering and subsequent ICP pose refinement looked to make a reasonable object detection and pose estimation pipeline. A rough sketch of this pipeline is shown in figure 10.

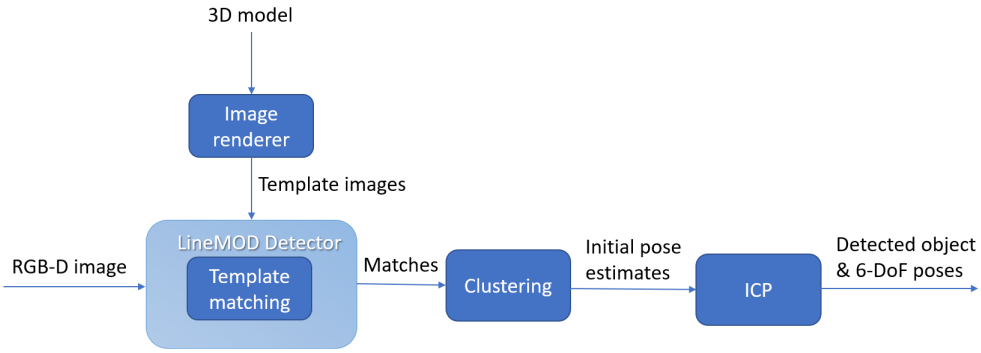


Figure 10: Rough sketch of the preliminary 6-DoF pose estimation pipeline. The template generation and loading to detector, i.e. training, is done offline.

Overall, the LineMOD detector clearly showed some potential. The detector usually found at least one approximate match for the target object, whilst not obtaining too many false matches. However, as discussed in [29] it would be preferable to attain a higher share of matches on the objects in order to make the validation step a bit simpler. In addition, the precision of the pose estimates were often inadequate, as the rotations did not match the ones of the true objects. It seemed like the shape of the object, namely the big chair part from figure 2, made the detection more challenging as there are no easily recognizable curved surfaces. The discontinuity of flat surfaces on the object also seemed to make it more prone to false matches, primarily from flat surfaces in the scene. As a consequence of the insufficient pose estimates from the LineMOD detector the match evaluating *"Clustering"* and pose refining *"ICP"* modules were not implemented. Judging from the results, the ICP would not be able to deliver proper pose estimates as

these modules require a decent selection of approximate matches. If the pose estimation pipeline from figure 10 should remain, modifications to the detector were considered necessary. Another drawback of this object detection and pose estimation approach was the computational time. Although hashing-based and tree-based methods can speed up the matching process, the total would still be rather significant when including clustering and ICP. This concern gave rise to the idea of combining the implemented detector with a more efficient motion tracker, only requesting detections when initializing the tracker, and for some occasional checkups. The RBGT [48] was thus mentioned as it seemed like an interesting candidate if having to look for alternate solutions.

3.2 Revised Proposal

Some more modifications were done to the LineMOD detector at the beginning of this thesis. Changing the sampling step and the number of template images seemed to improve the performance slightly. However, the implementation of ICP demonstrated that the initial pose estimates were very often inadequate as the algorithm was unable to converge. As a result, focus was shifted to the aforementioned RBGT [48]. This tracker was considered to be a good candidate as it has demonstrated some very promising results in terms of both tracking performance and efficiency. Furthermore, the publicly available source code [49] only requires a 3D model of the objects, and their maximum body diameter in order to get the tracker up and running. As the Azure Kinect is already set as the default camera model, there is no need to create a custom camera class either. Naturally, as the RBGT does not provide any solution for object detection, initial poses are required for all objects in order to start the tracking. Nevertheless, further testing with clustering of LineMOD matches showed that the position of objects can be estimated rather accurately by clustering in terms of translation, and ignoring the rotation of the matches. Having some rough assumptions for the object rotations based on the use case would hence simplify the pose detection task. This gave rise to an idea of combining the clustered translation with a rough rotation estimate and the RBGT in order to find a sufficient initial pose and start the tracking. This approach will be explained further in 7.2. A rough sketch of the new proposed pipeline is shown in figure 11.

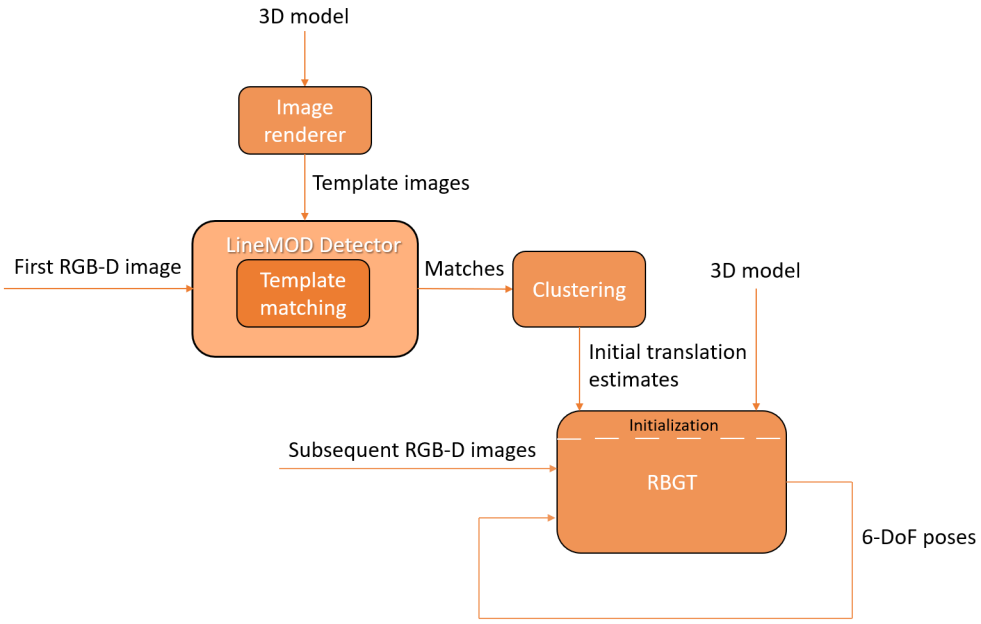


Figure 11: Rough sketch of the revised detection and 6-DoF pose estimation pipeline. The template generation and loading to the detector, and the corresponding *model generating* for the RBGT is done offline. Note that the RBGT [48] only utilizes the RGB images.

4 Theoretical Background

In this section a more detailed description of the proposed framework methods will be provided. First, the LineMOD [19] descriptors and similarity measures are presented in 4.1, before additional information will be given on clustering and the ICP algorithm in 4.2 and 4.3 respectively. Finally, the main concepts of the RBGT method [48] are described in 4.4. For easier referencing, the original notations are used for both the LineMOD detector and the RBGT. Additional description for these notations can be found in the *list of symbols* at the beginning of the thesis.

4.1 LineMOD

The following subsection is based on [19] and [17] by Hinterstoisser et al. Both papers cover the LineMOD image representation and template matching strategy.

4.1.1 Similarity Measure

In order to find potential matching objects in an input image, a similarity measure is required. The generalized LineMOD variant can be formalized as:

$$\varepsilon(\mathcal{I}, \mathcal{T}, c) = \sum_{(r,m)\in\mathcal{P}} \max_{t\in\mathcal{R}(c+r)} f_m(\mathcal{O}_m(r), \mathcal{I}_m(t)), \tag{1}$$

where $\mathcal{I}$ is the input image and $\mathcal{T}$ is a given template. This template is defined as $\mathcal{T} = (\{\mathcal{O}_m\}_{m\in M}, \mathcal{P})$, where $\mathcal{O}_m$ is the template image of modality m, and $\mathcal{P}$ is a list of pairs (r, m), where r is the location of a discriminant feature of modality m. The comparing between a scene image and a template is done through a sliding window approach, where c is the location in $\mathcal{I}$ to be evaluated. By summarizing the similarity scores over the discriminant features in $\mathcal{P}$, through a similarity function f_m, a total similarity score is provided. A template is matched if the score is higher than an applied threshold. Furthermore, the separate feature scores corresponding to $(r, m) \in \mathcal{P}$ are set to equal the maximum similarity score in a neighbourhood $\mathcal{R}(c + r)$ of size $N \times N$ with $r + c$ as the midpoint. This way the similarity measure in equation 1 archives robustness to small translations and deformations.

4.1.2 Modalities

As previously mentioned, LineMOD combines the modalities of both color gradients and surface normals. For the case of **color gradients**, these are obtained by in inspecting each of the three color channels (R,G,B) separately. This naturally increases robustness as the different channels provide a greater option of gradients than what would be the case in grayscale images. Figure 12 also illustrates the difference in contour visibility using both methods. In addition, this method considers only the orientation of the gradients and not their norms, which increases robustness to contrast changes. For each image location the gradient orientation of the channel $\mathcal{C}$ whose magnitude is largest will be selected. This can be expressed as:

$$\mathcal{I}_G(\boldsymbol{x}) = \mathbf{ori}(\hat{\mathcal{C}}(\boldsymbol{x})), \tag{2}$$

where $\mathcal{I}_G(\boldsymbol{x})$ is the orientation of the most prominent color gradient:

$$\hat{\mathcal{C}}(\boldsymbol{x}) = \underset{\mathcal{C}\in\{R,G,B\}}{\arg\max} \left\| \frac{\partial \mathcal{C}}{\partial \boldsymbol{x}} \right\|, \tag{3}$$

at location $\boldsymbol{x} \in \mathbb{R}^2$ in the input image. As the normalized gradient map only considers the gradients orientation, and not their direction, the orientation space of the map is divided into n_o equal spacings as shown in figure 12. This will prevent the detection from being affected if the background changes from bright to dark. The similarity measure for the gradient orientation can accordingly be stated as:

$$\varepsilon_G(\mathcal{I}, \mathcal{T}, c) = \sum_{r\in\mathcal{P}} \max_{t\in\mathcal{R}(c+r)} |cos(\mathbf{ori}(\mathcal{O}, r) - \mathbf{ori}(\mathcal{I}, t))|. \tag{4}$$

In addition, to make the quantization of orientations more robust to noise, each location will be assigned the quantized orientation which occurs most often in a 3 x 3 neighborhood.

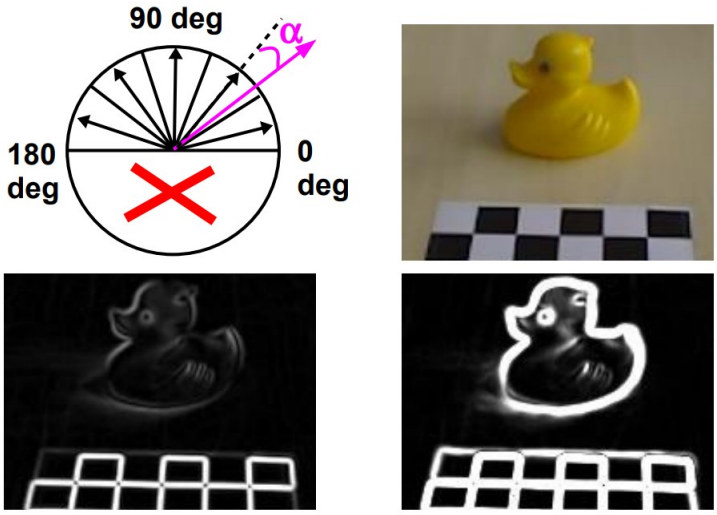


Figure 12: **Upper Left:** Quantization of the gradient orientations: The pink orientation is closest to the second bin. **Upper Right:** A toy duck with a calibration pattern. **Lower Left:** The gradient image computed on a grayscale image. **Lower Right:** The gradient image computed using maximum magnitude from the separate color channels. (Source: [17])

The second modality, **surface normals**, are computed from a dense depth field provided by the 3D camera. The method apply the first order Taylor expansion of the depth function $D(\boldsymbol{x})$:

$$D(\boldsymbol{x} + d\boldsymbol{x}) - D(\boldsymbol{x}) = d\boldsymbol{x}^\top \nabla D + h.o.t. \tag{5}$$

For each pixel location $\boldsymbol{x}$, an optimal depth gradient estimate $\hat{\nabla D}$ can be found given some pixel offset vectors $d\boldsymbol{x}$. This gradient can accordingly be expressed as a 3D plane going through three points $\boldsymbol{X}_1$, $\boldsymbol{X}_2$ and $\boldsymbol{X}_3 \in \mathbb{R}^3$:

$$\boldsymbol{X}_1 = \vec{v}(\boldsymbol{x})D(\boldsymbol{x}), \tag{6}$$

$$\boldsymbol{X}_2 = \vec{v}(\boldsymbol{x} + [1,0]^\top)(D(\boldsymbol{x}) + [1,0]\hat{\nabla D}), \tag{7}$$

$$\boldsymbol{X}_3 = \vec{v}(\boldsymbol{x} + [0,1]^\top)(D(\boldsymbol{x}) + [0,1]\hat{\nabla D}), \tag{8}$$

where $\vec{v}(\boldsymbol{x})$ is the vector along the line of sight pointing towards the 3D point given by pixel $\boldsymbol{x}$. This vector can be seen as a projective element provided by the internal parameters of the depth sensor. Finally, the surface normal at the 3D point can be estimated as the

normalized cross-product of $\boldsymbol{X}_2 - \boldsymbol{X}_1$ and $\boldsymbol{X}_3 - \boldsymbol{X}_1$. The similarity function for these surface normals is defined as the dot product of the normalized surface normals. Hence, the similarity measure for the depth image can be expressed as:

$$\varepsilon_D(\mathcal{I}, \mathcal{T}, c) = \sum_{r \in \mathcal{P}} \max_{t \in \mathcal{R}(c+r)} \mathcal{O}_D(r)^\top \mathcal{I}_D(t), \tag{9}$$

where $\mathcal{O}_D(r)$ is the normalized surface normal at location r from the reference image, and $\mathcal{I}_D(t)$ is the normalized surface normal at location t in the input image. As for the color gradients, the surface normals are also quantized into n_0 bins. These are spread out in a right circular cone as shown in figure 13. In order to reduce the quantization noise on the surfaces, the pixels, or 3D points with substantial depth differences will be ignored. This primarily increases robustness for areas with depth discontinuity. In addition, to further increase the robustness to noise, each location in the normalized surface normal map will be assigned the quantized orientation which occurs most often in a 5 x 5 neighborhood.

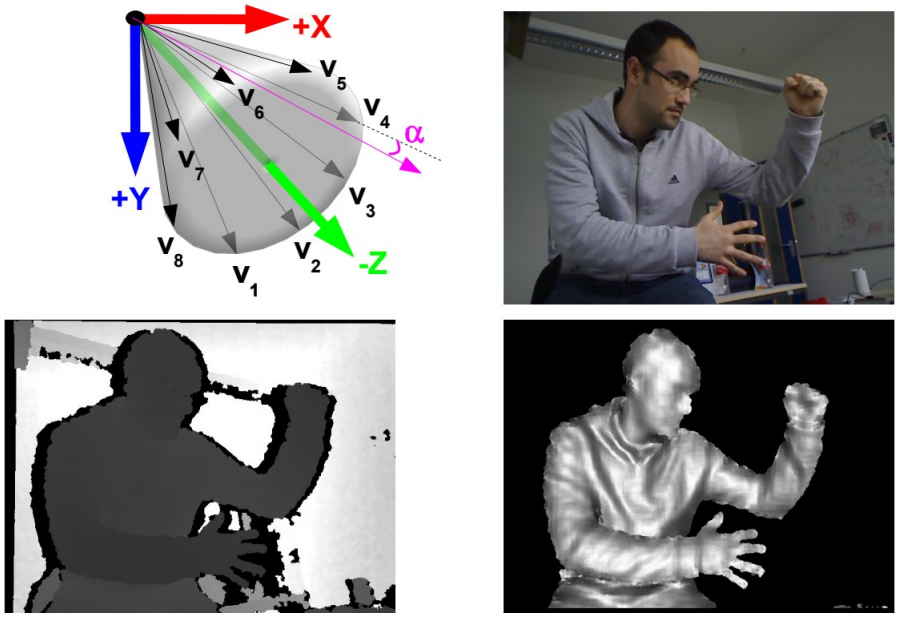


Figure 13: **Upper Left:** Quantization of the surface normals: The pink orientation is closest to the precomputed normal v_4. **Upper Right:** A person standing in an office. **Lower Left:** The corresponding depth image. **Lower Right:** Computed surface normals. The background was removed for visibility reasons. (Source: [17])

4.1.3 Precomputed Response Maps

For efficient computation of similarity scores, the method introduces a binary representation of *spread orientations*, and a lookup table for fast computation of the similarity measures found in precomputed response maps. The spreading of orientations and its simple representation prevents us from having to evaluate the *max* operator in equation 1. For each image location, a binary string indicates the presence of a quantized orientation by setting the corresponding bit to 1. Similar representation is used for the surface normal modality. However, for simplicity, only the color gradient representation and response map computation will be illustrated in this subsection. This representation, and the process of orientation spreading is shown in figure 14. The encoding of this spreading is performed by OR'ing the concerning binary strings resulting in the more robust gradient representaion, denoted by $\jmath_m$.

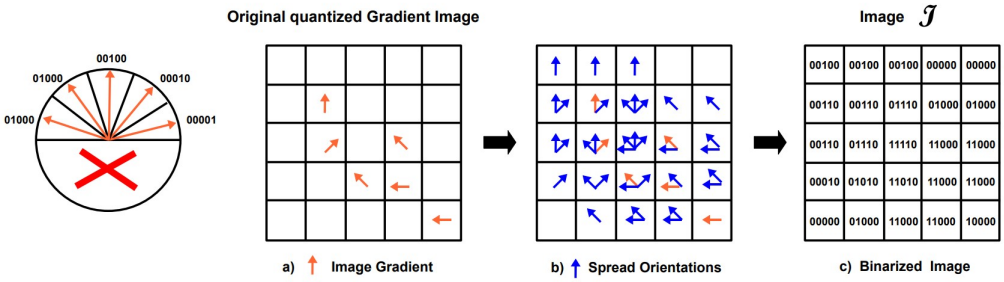


Figure 14: Spreading the gradient orientations. **Left:** The n_0 gradient orientations and their binary code. **a)** The gradient orientations in the input image, shown in orange. **b)** The gradient orientations are spread to a neighbourhood of size T, as shown in blue. **c)** The binary representation of the spread orientations. For this figure $T = 3$ and $n_0 = 5$. In practice, the method uses $T = 8$ and $n_0 = 8$. (Source: [17])

When assessing the similarity for each discriminant feature i of modality m in $\mathcal{P}$, a precomputed lookup table τ_i is utilized, where the integer value of $\jmath_m$ is used as an index to the corresponding similarity score:

$$\tau_{i,m}[\jmath_m] = \max_{l \in \jmath_m} |f_m(i,l)|. \tag{10}$$

For the case of color gradients, i is the index of the quantized gradient orientation of the template feature from $\mathcal{P}$, while l is the individual gradient orientations in a location as shown in figure 14 b). Accordingly, the similarity score between each discriminant gradient feature i, and corresponding spread input image orientations $\jmath$ can be stated as:

$$\tau_i[\jmath] = \max_{l \in \jmath} |cos(\mathbf{ori}(i) - \mathbf{ori}(l))|. \tag{11}$$

As $\jmath$ in principle represent all gradient orientations present in a neighbourhood, the method achieves robustness to small translations and deformations without having to iterate through $t \in \mathcal{R}(c + r)$, as done in equation 1 and 4. By defining $\mathcal{J}$ as an image of

$\mathcal{J}$-pixels, the values of the response map $\mathcal{S}_i$ can be precomputed as:

$$\mathcal{S}_i(c+r) = \tau_i[\mathcal{J}(c+r)]. \tag{12}$$

Finally, the similarity measure from equation 4 becomes:

$$\varepsilon_G(\mathcal{I}, \mathcal{T}, c) = \sum_{r \in P} \mathcal{S}_{ori(\mathcal{O},r)}(c+r), \tag{13}$$

where the different response map variants are chosen as specified by the orientation i of the current reference image feature in location r. The process of precomputing the response maps is illustrated in figure 15.

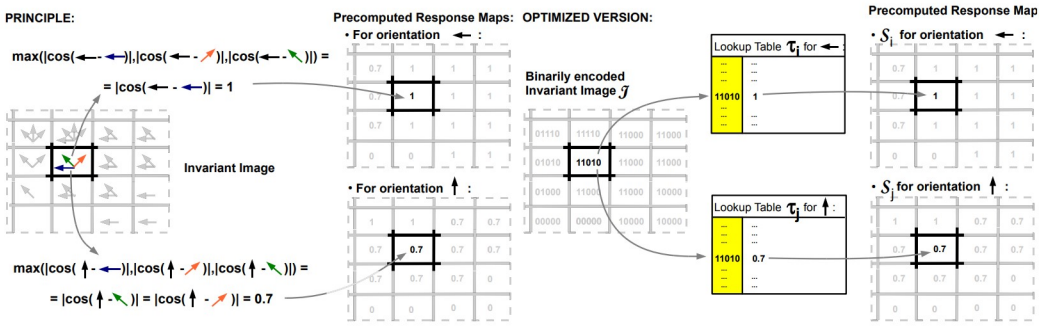


Figure 15: Precomputation of the response maps $\mathcal{S}_i$. **Left:** There is one response map for each quantized orientation. These store the maximal similarity between this quantized orientation, and the corresponding combinations of orientations i the input image. **Right:** This process is done efficiently by using the binary representation in $\mathcal{J}$ as index to lookup tables of maximum similarity. (Source: [17])

4.2 Clustering

Clustering or cluster analysis is the task of classifying objects into distinct groups or classes based on their available data [27]. Objects with similar attributes will hence be clustered together, while more dissimilar objects will be placed in different classes. This multi-objective optimization problem can be solved by numerous different clustering methods, such as *connectivity-based clustering*, also known as *hierarchical clustering*, *distribution-based clustering* and *centroid-based clustering* [28]. *k-means clustering* is an example of the latter and is illustrated in figure 16. Given a fixed number of clusters k, this method minimizes the total Euclidean distance between all objects and their nearest cluster center by iteratively changing the centroid positions. The objective function can be expressed as:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n_j} ||x_{i,j} - c_j||^2, \tag{14}$$

where k is the number of clusters, n_j is the number of objects in cluster j and $x_{i,j}$ is object of index i in cluster j. The center position of cluster j is given by c_j. As the optimization problem itself is NP-hard [58] the iterative approach will search for approximate solutions. Lloyd's algorithm [44], also known as *k-means algorithm*, does this by repeatedly assigning each object to its nearest centroid c_j before computing the new centroid positions as the mean of these objects. The closest cluster center for a given object x_i is found as:

$$arg\min_{j} ||x_i - c_j||^2, \tag{15}$$

while the the new centroid positions are calculated as:

$$c_j = \frac{1}{n_j} \sum_{i=1}^{n_j} x_{i,j}. \tag{16}$$

The algorithm stops when the object distribution converges, i.e. no objects are assigned to new clusters.

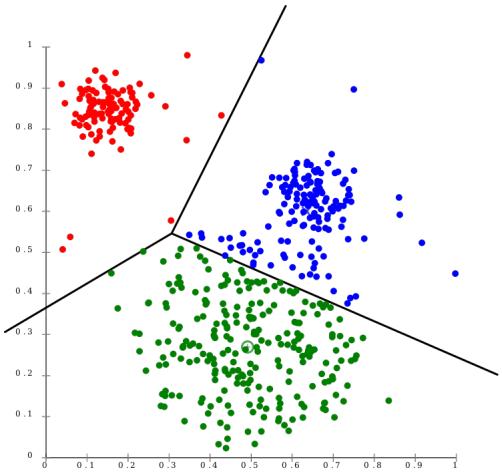


Figure 16: Two dimensional k-means clustering for $k = 3$. The different colors illustrate to which cluster the objects have been assigned. (Source: [7])

4.3 ICP

Iterative Closest Point (ICP) [4] is an algorithm used to minimize the difference between two clouds of points. As discussed in section 2.5.1, ICP is often applied when given an approximate initial pose estimate from a template based framework. Through an iterative scheme, the geometric distances are minimized, resulting in a translation and rotation which aligns the two point clouds as illustrated in figure 17.

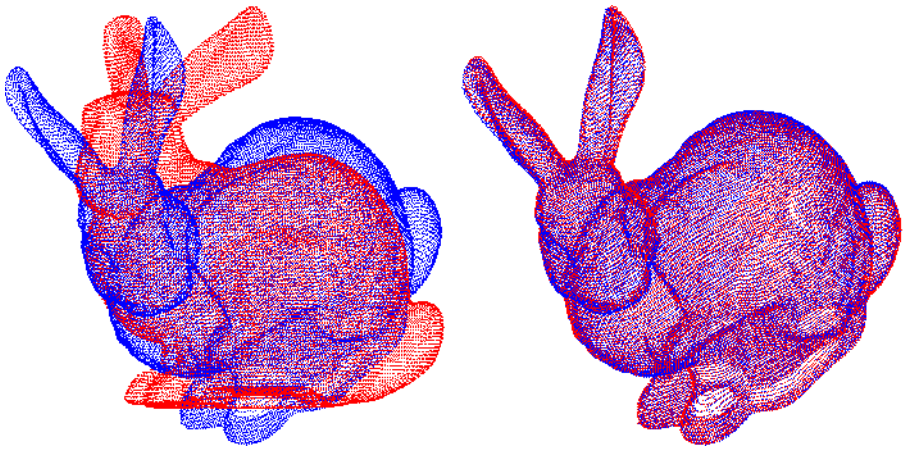


Figure 17: Alingning of point clouds using ICP. **Left:** Two clouds of points (blue and red) given as input for the ICP algorithm. **Right:** The red point cloud has been translated and rotated in order to minimize geometric difference to the blue point cloud. (Source: [31])

In contrast to *Kabsch algorithm* [21] and other solutions, ICP needs no correspondences between the two sets of points. Instead, for each iteration, every point in the source point cloud will be matched with the closest point in the reference point cloud. These matches will then be the basis for the subsequent geometric difference minimization, providing the transformation applied in the next iteration. Zhang [61] also proposes a modified *k-d tree* algorithm for efficient computation of the closest point matches. A statistical method furthermore takes care of outliers and variations in the presence and absence of corresponding object points.

4.4 RBGT

The following subsections are based on [48] by Stoiber et al. and give an introduction to the basic mathematical concepts and notations in 4.4.1 before introducing the probabilistic model in 4.4.2 - 4.4.4, and the discrete scale-space formulation in 4.4.5. In 4.4.6 the Gaussian equivalence of the proposed likelihood function is demonstrated before the optimization method is presented in 4.4.7. Lastly, the gradient and Hessian approximations are given in 4.4.8.

4.4.1 Preliminaries

For the remaining theory sections concerning the RBGT, the 3D model points are defined by ${}_M\boldsymbol{X}_i = \boldsymbol{X}_i = [X_i\ Y_i\ Z_i]^\top \in \mathbb{R}^3$, or the corresponding homogeneous form ${}_M\tilde{\mathbf{X}}_i = \tilde{\mathbf{X}}_i = [X_i\ Y_i\ Z_i\ 1]^\top$. The color images are denoted by $\boldsymbol{I} : \Omega \to \{0, ..., 255\}^3 \subset \mathbb{R}^2$, while the RGB values at image coordinate $\boldsymbol{x}_i = [x_i\ y_i] \in \mathbb{R}^2$ are expressed as $\boldsymbol{y}_i = \boldsymbol{I}(\boldsymbol{x}_i)$. If given a 3D model point represented in the camera reference frame C, ${}_C\boldsymbol{X}_i$ is projected into an undistorted image using the pinhole camera model $\boldsymbol{\pi}$:

$$\boldsymbol{x}_i = \boldsymbol{\pi}({}_C\mathbf{X}_i) = \begin{bmatrix} \frac{X_i}{Z_i} f_x + p_x \\ \frac{Y_i}{Z_i} f_y + p_y \end{bmatrix}, \tag{17}$$

with f_x and f_y being the focal lengths, and p_x and p_y being the principal point coordinates. In order to describe the relative translation $\boldsymbol{t} \in \mathbb{R}^3$ and rotation $\boldsymbol{R} \in \mathbb{SO}(3)$ between the model reference frame M and the camera reference frame C, the homogeneous matrix ${}_C\boldsymbol{T}_M$ is used:

$${}_C\tilde{\mathbf{X}}_i = {}_C\boldsymbol{T}_M\, {}_M\tilde{\mathbf{X}}_i = \begin{bmatrix} {}_C\boldsymbol{R}_M & {}_C\boldsymbol{t}_M \\ \mathbf{0} & 1 \end{bmatrix} {}_M\tilde{\mathbf{X}}_i, \tag{18}$$

where ${}_M\tilde{\mathbf{X}}_i$ is a 3D model point expressed in the model reference frame M, and ${}_C\tilde{\mathbf{X}}_i$ is the same point expressed in the camera reference frame C.

For small rotations, the RBGT [48] uses the minimal angle-axis representation. By neglecting the higher order terms of the exponential map series expansion:

$$\boldsymbol{R} = \exp([\boldsymbol{r}]_\times) = \boldsymbol{I} + [\boldsymbol{r}]_\times + \frac{1}{2!}[\boldsymbol{r}]_\times^2 + \frac{1}{3!}[\boldsymbol{r}]_\times^3 + ..., \tag{19}$$

where $[\boldsymbol{r}]_\times$ represent the skew-symmetric matrix of $\boldsymbol{r}$, the linear variation of a 3D model point represented in the camera reference frame C can be described as:

$${}_C\tilde{\mathbf{X}}_i^+ = \begin{bmatrix} {}_C\boldsymbol{R}_M & {}_C\boldsymbol{t}_M \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{I} + [\boldsymbol{\theta}_r]_\times & \boldsymbol{\theta_t} \\ \mathbf{0} & 1 \end{bmatrix} {}_M\tilde{\mathbf{X}}_i, \tag{20}$$

where $\boldsymbol{\theta_r} \in \mathbb{R}^3$ is the rotation variation, $\boldsymbol{\theta_t} \in \mathbb{R}^3$ is the translational variation and ${}_C\tilde{\mathbf{X}}_i^+$ is the variated model point. The plus operator symbol indicates that this variated variable depends on the full variation vector $\boldsymbol{\theta}^\top = [\boldsymbol{\theta}_r\ \boldsymbol{\theta}_t]$.

4.4.2 Appearance Models

Just as PWP3D [34], RBGT [48] also utilizes a pixel-wise color histogram posterior membership approach inspired by [5]. Using a color histogram representation, global appearance models for the foreground $p(\boldsymbol{y}|m_f)$ and background $p(\boldsymbol{y}|m_b)$ are created, which in turn are used to calculate the pixel-wise posteriors:

$$p_j(\boldsymbol{y}_i) = \frac{p(\boldsymbol{y}_i|m_j)}{p(\boldsymbol{y}_i|m_f) + p(\boldsymbol{y}_i|m_b)}, \quad j \in \{f, b\}. \tag{21}$$

An illustration of such membership posteriors is given in figure 18. The color histograms, which are discretized with 32 equidistant bins in each RGB dimension, are updated every tracking iteration as the final pose is estimated. The affiliated probability distributions $p(\boldsymbol{y}|m_f)$ and $p(\boldsymbol{y}|m_b)$ are thus updated accordingly to account for a changing background or variation in the foreground region due to a diverse surface or illumination changes. When updating the color histograms, the correspondence lines, as displayed in figure 18a are put to use. With an offset of two pixels at the center, the first 10 pixels are used in both directions along the correspondence lines. Pixels along the inner segment are assigned to the foreground model, while the pixels along the outer segment are assigned to the background model.

(a) (b)

Figure 18: Sample of posterior membership probabilities given a RGB image and color histogram models $p(\boldsymbol{y}|m_f)$ and $p(\boldsymbol{y}|m_b)$. **a)** Pixel-wise posteriors describing the probability of a pixel belonging to the background. White pixels indicate $p_b = 1$, while black pixels indicate $p_b = 0$. When updating the color histograms, the method uses RGB values along the orange correspondence lines. The line segments with high contour probability are indicated by red. **b)** Corresponding RGB input image.

Based on [5], the statistical models are adapted online using:

$$p_t(\boldsymbol{y}|m_i) = \alpha_i p_t(\boldsymbol{y}|m_i) + (1 - \alpha_i)p_{t-1}(\boldsymbol{y}|m_i), \quad j \in \{f, b\}, \tag{22}$$

where $\alpha_f = 0.1$ and $\alpha_b = 0.2$ are the learning rates for the foreground and background, respectively. When initializing the histograms, no previous models exist, and $\alpha_f = \alpha_b = 1$.

4.4.3 Correspondence Lines

As mentioned in 2.3.2, the sparse use of pixel color information along correspondence lines is a key novelty of the RBGT approach [48]. Motivated by the term *correspondence points* used in ICP, the correspondence lines are also defined before being the subject of optimization. Described by a center $\boldsymbol{c}_i = [c_{xi}\ c_{yi}]^\top \in \mathbb{R}^2$ and a normal vector $\boldsymbol{n}_i = [n_{xi}\ n_{yi}]^\top \in \mathbb{R}^2$, with $||\boldsymbol{n}_i||_2 = 1$, these are defined by projecting a 3D contour point $\boldsymbol{X}_i$ into

the image along with an associated vector normal to the contour $\boldsymbol{N}_i$. Similar to [51], the required information is precomputed and stored by rendering a 3D model of the associated object. A total of 2562 different viewpoints are used, each placed on the vertices of a geodesic grid 0.8 m from the object center. For each of these template viewpoints $n_{cl} = 200$ points are randomly sampled from the object contour. The corresponding 3D model points and normal vectors, i.e. correspondence lines, are thus stored in separate data structures connected to their respective template view.

For each template view, the direction vector ${}_M\boldsymbol{v}_i \in \mathbb{R}^3$ pointing from the camera to the model center is also stored. When provided the previous pose from the tracker, or an initial pose from an object detection pipeline, the closest template view i_t is retrieved using:

$$i_t = \underset{i \in \{1,\dots,2562\}}{\arg\min} \; ({}_M\boldsymbol{v}_i^\top \, {}_M\boldsymbol{R}_C \, {}_C\boldsymbol{t}_M), \tag{23}$$

where ${}_C\boldsymbol{t}_M$ is a normalized translation vector given by the most recent pose update.

After finding the closest template view and associated correspondence lines, the pixels on the lines are described by rounding as follows:

$$\boldsymbol{x}_{cli}(r) = \lfloor \boldsymbol{c}_i + r\boldsymbol{n}_i + \mathbf{0.5} \rfloor, \tag{24}$$

where $r \in \mathbb{R}$ is the distance from the center $\boldsymbol{c}_i$ and $\boldsymbol{n}_i$ is the normal vector for correspondence line i. As the correspondence lines remain fixed during the 6-DoF pose variation, the projected difference Δc_i^+ from the center $\boldsymbol{c}_i$ can be calculated as:

$$\Delta c_i^+ = \boldsymbol{n}_i^\top (\boldsymbol{\pi}({}_C\boldsymbol{X}_i^+) - \boldsymbol{c}_i), \tag{25}$$

where ${}_C\boldsymbol{X}_i^+$ is the variated model point in the camera reference frame C. An illustration of a correspondence line with a projected difference Δc_i^+ is shown in figure 19.

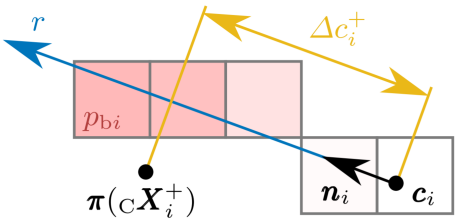


Figure 19: Correspondence line defined by a center $\boldsymbol{c}_i$ and a normal vector $\boldsymbol{n}_i$, along with evaluated pixels and the projected difference Δc_i^+ from $\boldsymbol{c}_i$ to $\boldsymbol{\pi}({}_C\boldsymbol{X}_i^+)$. The color intensity in red indicates the magnitude of the pixel-wise posterior p_{bi} for each pixel. (Source: [48])

In order to describe how well a variated model point ${}_C\boldsymbol{X}_i^+$ or projected difference Δc_i^+

explains the pixel colors along a given correspondence line, the pixel-wise posteriors from equation 21 must first be calculated for each correspondence line. Using $\boldsymbol{y}_i(r) = \boldsymbol{I}(\boldsymbol{x}_{cli}(r))$, the posterior probabilities now become:

$$p_{ij}(r) = \frac{p(\boldsymbol{y}_i(r)|m_j)}{p(\boldsymbol{y}_i(r)|m_f) + p(\boldsymbol{y}_i(r)|m_b)}, \quad j \in \{f, b\}. \tag{26}$$

4.4.4 Probabilistic Formulation

Inspired by PWP3D [34] the probabilistic formulation is finally stated as:

$$p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta}) \propto \prod_{r \in R_i} (h_f(r - \Delta c_i^+)p_{fi}(r) + h_b(r - \Delta c_i^+)p_{bi}(r)), \tag{27}$$

where h_f and h_b are smoothed step functions for foreground and background used to model uncertainty in the contour location. These will be specified in 4.4.6. The expression describes how well the pose dependent contour model explains the data $\boldsymbol{\mathcal{D}}_i$ for an associated correspondence line. R_i is a set of distances r from the line center to pixel centers that ensures that every pixel along the line appears exactly once. The full likelihood, assuming n_{cl} correspondence lines, can thus be calculated as:

$$p(\boldsymbol{\mathcal{D}}|\boldsymbol{\theta}) \propto \prod_{i=1}^{n_{cl}} p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta}). \tag{28}$$

4.4.5 Discrete Scale-Space Formulation

In order to improve computational efficiency, a discrete scale-space formulation is provided. This allows multiple pixels to be combined into one segment, while also having precomputed values for h_f and h_b available for each associated segment. These features are illustrated in figure 20. Accordingly, real-numbered values such as $r \in R_i$ are projected into a discrete space that is scaled as follows:

$$r_s = (r - \Delta r_i)\frac{\bar{n}_i}{s}, \qquad \Delta c_{si}^+ = (\Delta c_i^+ - \Delta r_i)\frac{\bar{n}_i}{s}, \tag{29}$$

where r_s and Δc_{si}^+ are the scaled versions of r and Δc_i^+ respectively, $s \in \mathbb{N}^+$ is the scale describing the number of pixels combined into one segment, and $\bar{n}_i = max(|n_{xi}|, |n_{yi}|)$ is the largest normal component. $\bar{n}_i$ is introduced as tilted correspondence lines in fact are longer than those that are completely horizontal or vertical. Finally, $\Delta r_i \in \mathbb{R}$ is the distance from the correspondence line center $\boldsymbol{c}_i$ to the closest border of a segment. Using the scaled values from equation 29, the likelihood function from equation 27 can thus be stated as:

$$p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si}) \propto \prod_{r_s \in R_s} (h_f(r_s - \Delta\tilde{c}_{si})p_{sfi}(r_s) + h_b(r_s - \Delta\tilde{c}_{si})p_{sbi}(r_s)), \tag{30}$$

where $\Delta\tilde{c}_{si}$ is a discretized projected difference value ensuring that the segment-wise posteriors p_{sfi} and p_{sbi}, marked by blue and yellow dots on the correspondence line in figure 20, are aligned with the precomputed values of h_f and h_b. R_s is a set segment distances making sure that every segment appears exactly once. By assuming pixel-wise independence, which is a well-established approximation, the segment-wise posteriors are defined as:

$$p_{sij}(r_s) = \frac{\prod\limits_{r \in S(r_s)} p(\boldsymbol{y}_i(r)|m_j)}{\prod\limits_{r \in S(r_s)} p(\boldsymbol{y}_i(r)|m_f) + \prod\limits_{r \in S(r_s)} p(\boldsymbol{y}_i(r)|m_b)}, \quad j \in \{f, b\}, \tag{31}$$

where S is a function mapping r_s to a set of values r that describe the s pixel centers belonging to a segment.

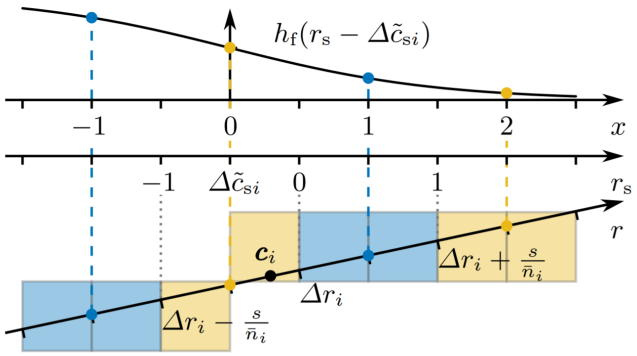


Figure 20: Example of the relation between the unscaled space r along the correspondence line, the scale space r_s, and the smoothed step function h_f given a $\Delta\tilde{c}_{si}$. The segments are visualized by same color pixels, while the blue and yellow dots indicate segment centers and precalculated values of h_f. Dashed vertical lines connecting the dots are simply emphasizing that $\Delta\tilde{c}_{si}$ must be chosen such that the precalculated values of h_f are aligned with the segment centers. Δr_i was chosen such that the center $r_s = 0$ lies on a defined location between two segments. (Source: [48])

The distribution from equation 30 is calculated for 11 discrete values of $\Delta\tilde{c}_{si} \in \{-5, -4, ..., 5\}$, while a minimal Δr_i ensures that the distribution center at $\Delta\tilde{c}^+_{si} = 0$ is closest to the correspondence center $\boldsymbol{c}_i$. For the smoothed step functions, 10 precalculated values corresponding to $x \in \{-4.5, -3.5, ..., 4.5\}$ are utilized. If using the h_f and h_b functions specified in 4.4.6, this means that the smallest considered value is $h_f(4.5) = h_b(-4.5) = 0.03$. Smaller values are thus neglected, as they do not contribute significantly to the overall distribution. As the likelihood function from equation 30 can only be calculated for a limited number of discrete values $\Delta\tilde{c}_{si}$, an approximation is needed in order to evaluate

the likelihood for arbitrary $\boldsymbol{\theta}$ and corresponding Δc_{si}^{+}. Using linear interpolation, the approximation can be stated as:

$$p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta}) \propto (\Delta\tilde{c}_{si}^{+} - \Delta c_{si}^{+})p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si}^{-}) + (\Delta c_{si}^{+} - \Delta\tilde{c}_{si}^{-})p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si}^{+}), \tag{32}$$

where $\Delta\tilde{c}_{si}^{+}$ and $\Delta\tilde{c}_{si}^{-}$ are the upper and lower neighboring discretized values.

4.4.6 Gaussian Equivalence

As Newton optimization in general produces fine results for Gaussian distributions, [48] present smoothed step functions h_f and h_b that ensures that the likelihood function from equation 30 follows a normal distribution. For practical reasons, only the main concepts of the mathematical proof are reproduced in this section. A more detailed description can be found in the *supplementary material* in [49]. By assuming a contour at the center of the correspondence line, and a perfect segmentation, the simple unit step function for pixel-wise posteriors, illustrated by $p_{fi}(r)$ in figure 21 is provided. Apart from that, h_f and h_b are first of all restricted to be symmetric and sum to one. Accordingly, these are defined as:

$$h_f(x) = 0.5 - f(x), \quad h_b(x) = 0.5 + f(x), \tag{33}$$

where $f(x) \in [-0.5, 0.5]$ is an odd function fulfilling $\lim_{x\to\infty} f(x) = 0.5$ and $\lim_{x\to-\infty} f(x) = -0.5$.

By furthermore assuming infinitesimally small pixels, the likelihood from equation 27

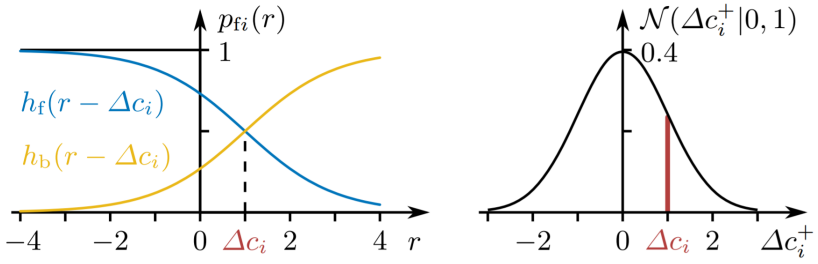


Figure 21: Example demonstrating the relation between the smoothed step functions h_f and h_b and the normal distribution of Δc_i^+ for $s_h = 1$. The graph on the left shows perfect posterior probabilities p_{fi} for the foreground, i.e. a perfect segmentation, including the smoothed step functions for a specific projected difference Δc_i. The graph on the right displays the corresponding normal distribution for all values of Δc_i^+. The probability value for Δc_i and associated smoothed step functions from the left graph is marked in red. (Source: [48])

can be stated in continuous form as:

$$p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta}) \propto \exp\Big(\int_{r=-\infty}^{\infty} \ln\big(h_f(r - \Delta c_i^+)p_{fi}(r) + h_b(r - \Delta c_i^+)p_{bi}(r)\big)\big)dr\Big). \tag{34}$$

Using the assumption of perfect pixel-wise posteriors for $p_{fi}(r)$ and $p_{bi}(r)$, this is simplified into:

$$p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta}) \propto \exp\Big(\int_{r=-\infty}^{-\Delta c_i^+} \ln\big(h_f(x)\big)dx + \int_{-\Delta c_i^+}^{\infty} \ln\big(h_b(x)\big)dx\Big). \tag{35}$$

In order to get rid of constant scaling terms and the integral, the logarithm and Leibniz's rule for differentiation are applied to calculate the first order derivative with respect to Δc_i^+:

$$\frac{\partial \ln\big(p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta})\big)}{\partial \Delta c_i^+} = -\ln\big(h_f(-\Delta c_i^+)\big) + \ln\big(h_b(-\Delta c_i^+)\big). \tag{36}$$

By factorizing this expression using equation 33, this can be written as:

$$\frac{\partial \ln\big(p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta})\big)}{\partial \Delta c_i^+} = 2\tanh^{-1}\big(2f(-\Delta c_i^+)\big). \tag{37}$$

Finally, the result in equation 37 is set equal to the first order derivative of $\ln\big(\mathcal{N}(\Delta c_i^+|0, s_h)\big)$. Knowing that this should give $-s_h^{-1}\Delta c_i^+$, the equality is solved for f using equation 33, resulting in the following smoothed step functions:

$$h_f(x) = \frac{1}{2} - \frac{1}{2}\tanh\big(\frac{x}{2s_h}\big), \quad h_b(x) = \frac{1}{2} + \frac{1}{2}\tanh\big(\frac{x}{2s_h}\big), \tag{38}$$

where s_h is the slope parameter for the smoothed step functions in addition to being the variance of the associated likelihood function.

Given that the equality is enforced for the first order derivatives of the logarithms, $p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta})$ and $\mathcal{N}(\Delta c_i^+|0, s_h)$ can only differ by a constant scaling factor. Accordingly, the likelihood can be stated as:

$$p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta}) \propto \mathcal{N}(\Delta c_i^+|0, s_h). \tag{39}$$

As visualized in figure 21, the derived smoothed step functions from equation 38 ensures that the probabilistic model follows an approximate Gaussian distribution. Although using assumptions, [48] claim to achieve excellent convergence for the regularized Newton optimization.

4.4.7 Regularized Newton Optimization

As already mentioned, the RBGT [48] uses a regularized Newton approach in order to maximize the likelihoods. Following the calculation of the distributions, the variation vector $\boldsymbol{\theta}^\top = [\boldsymbol{\theta}_r \boldsymbol{\theta}_t]$ is thus estimated using Newton optimization with Tikhonov regularization as follows:

$$\hat{\boldsymbol{\theta}} = \Big(-\boldsymbol{H} + \begin{bmatrix} \lambda_r \boldsymbol{I}_3 & \mathbf{0} \\ \mathbf{0} & \lambda_t \boldsymbol{I}_3 \end{bmatrix}\Big)^{-1} \boldsymbol{g}, \tag{40}$$

where $\boldsymbol{g}$ is the gradient vector, $\boldsymbol{H}$ is the Hessian matrix, and λ_r and λ_t are the regularization parameters for rotation and translation, respectively. The gradient vector and Hessian matrix are defined as:

$$\boldsymbol{g}^\top = \frac{\partial}{\partial \boldsymbol{\theta}} \ln\big(p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta})\big)\Big|_{\boldsymbol{\theta}=\mathbf{0}}, \quad \boldsymbol{H} = \frac{\partial^2}{\partial \boldsymbol{\theta}^2} \ln\big(p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta})\big)\Big|_{\boldsymbol{\theta}=\mathbf{0}}, \tag{41}$$

while $\lambda_r = 5000$ and $\lambda_t = 5000000$. The log-likelihood is utilized as this removes scaling terms and turn products into summation, and thus simplifies the calculations of $\boldsymbol{g}$ and $\boldsymbol{H}$. After having estimated the variation vector, the pose is updated according to:

$${}_C\boldsymbol{T}_M = {}_C\boldsymbol{T}_M \begin{bmatrix} \exp\big([\hat{\boldsymbol{\theta}}_r]_\times\big) & \hat{\boldsymbol{\theta}}_t \\ \mathbf{0} & 1 \end{bmatrix}. \tag{42}$$

By iteratively repeating the process of calculating the pose dependent gradient vector and Hessian matrix, and updating the pose according to the estimated variation from equation 40, an optimal pose can be estimated based on the data $\boldsymbol{\mathcal{D}}$. The RBGT [48] performs only 2 iteration of the regularized Newton method. However, after the 2 iterations of optimization, an updated closest template view is retrieved based on the optimized pose. This gives rise to new data $\boldsymbol{\mathcal{D}}$ and a corresponding likelihood $p(\boldsymbol{\mathcal{D}}|\boldsymbol{\theta})$ to optimize. In order to find the final pose, the process of calculating and optimizing the likelihood is repeated 7 times, each time starting with the retrieval of a new template view. While the first and second iterations uses a scale of $s = 5$ and $s = 2$, respectively, all remaining iterations have scales of $s = 1$. Consequently, larger areas with lower resolution are considered initially, while short lines with high resolution are used as the pose estimate converges. In total, no more than approximately 2 ms are required for completing all 7 iterations.

4.4.8 Gradient and Hessian Approximation

Using the chain rule, the gradient vector and Hessian matrix can be defined as:

$$\boldsymbol{g}^\top = \sum_{i=1}^{n_{cl}} \frac{\partial \ln\big(p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta})\big)}{\partial \Delta c_{si}^+} \frac{\partial \Delta c_{si}^+}{\partial {}_C\boldsymbol{X}_i^+} \frac{\partial {}_C\boldsymbol{X}_i^+}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\mathbf{0}}, \tag{43}$$

$$\boldsymbol{H} \approx \sum_{i=1}^{n_{cl}} \frac{\partial^2 \ln\big(p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta})\big)}{\partial {\Delta c_{si}^+}^2} \left(\frac{\partial \Delta c_{si}^+}{\partial {}_C\boldsymbol{X}_i^+} \frac{\partial {}_C\boldsymbol{X}_i^+}{\partial \boldsymbol{\theta}}\right)^\top \left(\frac{\partial \Delta c_{si}^+}{\partial {}_C\boldsymbol{X}_i^+} \frac{\partial {}_C\boldsymbol{X}_i^+}{\partial \boldsymbol{\theta}}\right)\Big|_{\boldsymbol{\theta}=\mathbf{0}}. \tag{44}$$

where n_{cl} is the total number of correspondence lines. The second order partial derivatives for Δc_{si}^+ and ${}_C\boldsymbol{X}_i^+$ are neglected as the first order partial derivatives of the log-likelihood (equation 47) becomes zero when the optimization reaches the maximum. By omitting the plus operator for variables evaluated at $\boldsymbol{\theta} = \mathbf{0}$ and using equation 20, 25, 29 and 32, the following first order partial derivatives are obtained:

$$\frac{\partial {}_C\boldsymbol{X}_i^+}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\mathbf{0}} = {}_C\boldsymbol{R}_M\big[-[{}_M\boldsymbol{X}_i]_\times \boldsymbol{I}\big], \tag{45}$$

$$\left.\frac{\partial \Delta c_{si}^{+}}{\partial_C \boldsymbol{X}_i^{+}}\right|_{\boldsymbol{\theta}=\mathbf{0}} = \frac{\bar{n}_i}{s}\frac{1}{{}_C Z_i^2}\begin{bmatrix} n_{xi} f_{xC} Z_i & n_{yi} f_{yC} Z_i & -n_{xi} f_{xC} X_i & -n_{yi} f_{yC} Y_i \end{bmatrix}, \tag{46}$$

$$\left.\frac{\partial \ln\left(p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta})\right)}{\partial \Delta c_{si}^{+}}\right|_{\boldsymbol{\theta}=\mathbf{0}} \approx \frac{p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si}^{+}) - p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si}^{-})}{(\Delta\tilde{c}_{si}^{+} - \Delta c_{si})p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si}^{-}) + (\Delta c_{si} - \Delta\tilde{c}_{si}^{-})p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si}^{+})}. \tag{47}$$

For the first order partial derivative of the log-likelihood, a different approximation is used if the normalized values of $p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si}^{+})$ or $p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si}^{-})$ are below a given threshold. If this is the case, the probabilities are regarded as unreliable and the derivatives of an approximated normal distribution as illustrated in figure 22 are utilized instead:

$$\left.\frac{\partial \ln\left(p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta})\right)}{\partial \Delta c_{si}^{+}}\right|_{\boldsymbol{\theta}=\mathbf{0}} \approx -\frac{1}{{\sigma_{\Delta\tilde{c}_{si}}}^2}(\Delta c_{si} - \mu_{\Delta\tilde{c}_{si}}), \tag{48}$$

where $\mu_{\Delta\tilde{c}_{si}}$ is the calculated mean for the distribution, while $\sigma_{\Delta\tilde{c}_{si}}$ is the standard deviation. [48] uses a threshold of 0.01. For the second order partial derivative, the approximation:

$$\left.\frac{\partial^2 \ln\left(p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta})\right)}{\partial \Delta c_{si}^{+}}\right|_{\boldsymbol{\theta}=\mathbf{0}} \approx -\frac{1}{{\sigma_{\Delta\tilde{c}_{si}}}^2} \tag{49}$$

is always used. These approximations are employed in order to ensure that the partial derivatives maintain a global view of the distribution, and increase the robustness in the presence of image noise and inaccurate probability values.

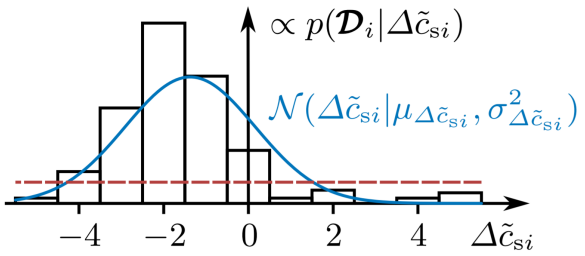


Figure 22: Example showing normalized values of a noisy discrete likelihood $p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si})$ and the normal distribution $\mathcal{N}(\Delta\tilde{c}_{si}, \mu_{\Delta\tilde{c}_{si}}, \sigma^2_{\Delta\tilde{c}_{si}})$ that approximates the likelihood. The red line indicates a lower threshold for the probability values. The normal distribution in blue is used for calculating partial derivatives of the log-likelihood for all values below the given threshold. (Source: [48])

5 Initial Implementation

This section will focus on the initial implementation of the proposed pipeline as described in section 3.2. First, the implementation of the detector, including the template generation and clustering is described in 5.1. Thereafter, the initial implementation of the RBGT [48], i.e. the tracker, is outlined in 5.2. All solutions and implementations are written in C++ and tested on a personal laptop with an Intel Core i5-8265U CPU and 8 GB RAM.

5.1 Detector

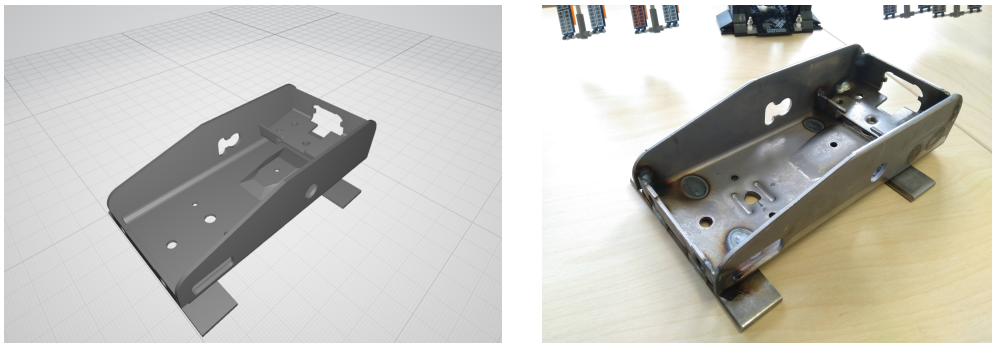
The following subsections describe the implementation of the template generation in 5.1.1 before discussing the implementation of the LineMOD [19] detector and the clustering approach in 5.1.2 and 5.1.3 respectively. Notice however that the template generation and most parts of the LineMOD detector were already implemented as a part of the preliminary project thesis.

5.1.1 Template Generation

In order to obtain the required set of template images for the LineMOD detector, a synthetic image renderer was utilized. Similar to the approach described in section 2.2.2, a 3D mesh model of the target object is given as input. The template images are then acquired by visiting a number of uniformly distributed viewpoints on synthetic spheres of different radii, while also rotating around the optical axis. The source code for this image rendering was provided by the *Object Recognition Kitchen* (ORK) [38]. For 3D model operations and scene creation, *Open Asset Import library* (Assimp) and *Simple Direct-Media Layer* (SDL) functionality is employed. In addition, the *Open Graphics Library* (OpenGL) yields the more elemental graphics operations.

The implementation of this image renderer proved to be a rather demanding process. Although the source code was easily accessible, integrating the different libraries was quite challenging. For instance, not having the correct extension or version of different software caused a lot of time-consuming troubleshooting. This was particularly the case when trying to make use of some additional required OpenGL functions. Various OpenGL extensions, such as *GLFW*, *FreeGLUT* and *GLEXT* were all tested before finally making it work by directly adding the source code from *OpenGL Extension Wrangler Library* (GLEW) into the project. Otherwise, the binary 64 bit versions of SDL 2.0 (SDL2) and Assimp 4.1.0 were both installed before linking these to the project by adding their respective *Dynamic-link library* (DLL) files to the repository. The same was done for FreeImage 3.18.0, which provides image reading and converting functionality for the *model* module in the 3D renderer implementation.

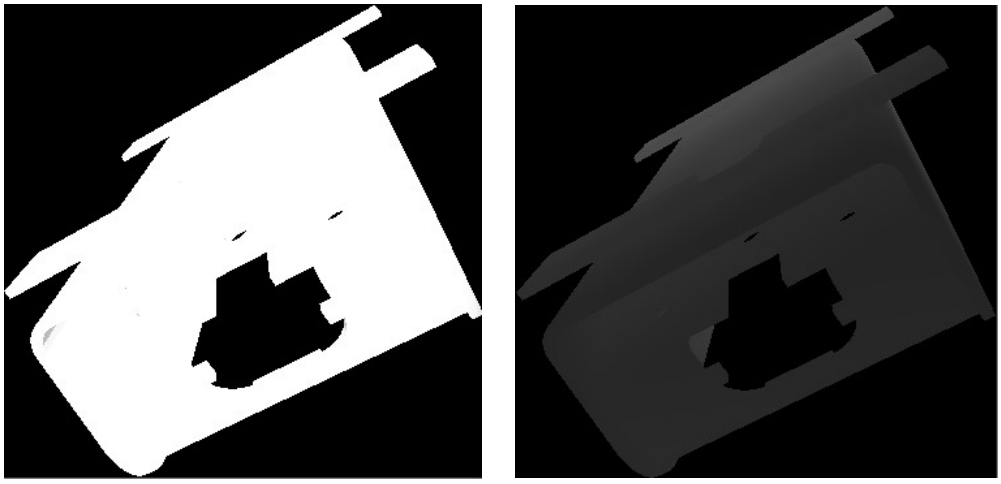
Seeing that a 3D mesh model is required for this template generating approach, models of the objects from figure 2 would be necessary for performing the intended object detection. A 3D model of the large chair part was therefore created using the free and open-source software *FreeCAD* [40]. An illustration of both the 3D model and the actual object is shown in figure 23. By using this model, and the image renderer from ORK, template images such as the ones from figure 24 were produced.



(a) (b)

Figure 23: 3D mesh model and real image of the object to track. **a)** 3D mesh model created in *FreeCAD* by project supervisor Klaus Ening. **b)** Illustrative image of object.

As mentioned in section 2.2.1 on LineMOD, the color gradients for texture-less objects will mainly be located on the contours of the object. Consequently, the 3D models require no data concerning the object texture. The binary image representation, as illustrated in figure 24a, will furthermore make the detection more robust to the varying surface texture of target objects in the pre-paint stage as addressed in the problem description in section 1.



(a) (b)

Figure 24: Sample of produced template images. **a)** Template color image. **b)** Template depth image. This image was edited in order to increase visibility.

5.1.2 LineMOD Detector

Given a set of template images, the work of producing efficient modality descriptors and performing template matching still remains. As suggested by the revised detection and 6-DoF pose estimation pipeline in section 3.2, this is still done using a LineMOD detector. This detector implementation was provided by OpenCV, or more precise, their additional repository for unreleased modules called *opencv_contrib* [1]. Despite having to do some extra installs, and linking these to the repository, the process of applying this detector was not too challenging. It did however require a rebuild of the original OpenCV repository in order to add the extra module, *rgbd*. This build was executed using CMake GUI.

The offline template loading, i.e. the response map computation described in section 4.1.3, is performed for one template image pair at a time, as the 3D model is being rendered. When the rendering is done and all templates have been added to the detector, all information is written to an *Extensible Markup Language* (XML) file. In addition, an XML file is created containing rendering parameters like the synthetic camera incintrics and the rotation matrices associated with the id of the corresponding template instance. This approach was inspired by [59]. After loading a given detector and the related parameters from the rendering, the approximate rotation of a target object can be found through the id of the matched template. As explained in 3.1, these rotations showed to be quite unreliable during the preliminary project thesis, and are thus ignored for the revised detection framework. The implementation from the project thesis also included the retrieval of 3D point maps for each potential match. As the ICP algorithm is no longer a part of the detection pipeline, this is also omitted in the revised solution. Instead, only the image coordinates, and the 3D point at the match center is utilized in the subsequent clustering. The 3D point, or translation of the object, is found by first retrieving the

depth Z_i at the center of the matching frame. Using the pinhole camera model, and the camera intrinsics, the translation at the surface $\boldsymbol{t}_{i,surface}$ is calculated as:

$$\boldsymbol{t}_{i,surface} = \begin{bmatrix} X_i & Y_i & Z_i \end{bmatrix}^\top = \begin{bmatrix} (u_i - p_x)\frac{Z_i}{f_x} & (v_i - p_y)\frac{Z_i}{f_z} & Z_i \end{bmatrix}^\top, \tag{50}$$

where f_x and f_y are the focal lengths, p_x and p_y are the principal point coordinates, and u_i and v_i are the image coordinates at the center of the matching frame for match i. Examples of such matching frames are illustrated in figure 25. The final step of finding the initial translation estimate t_i is to add the distance from the object surface to the object center d_i for the given template match. This distance is saved among the rendering parameters and is added to the to the z-axis (camera depth) of the translation:

$$\boldsymbol{t}_i = \boldsymbol{t}_{i,surface} + \begin{bmatrix} 0 & 0 & d_i \end{bmatrix}^\top. \tag{51}$$

Although d_i is dependent on the rotation of the template, this distance still seem to improve the translation accuracy, as related variations in rotation often have a very similar d_i. If the detection is flipped the other way around for instance, the defined object center ensures that the distance d_i is pretty much the same. Accordingly, an initial translation estimate can be provided for all potential matches attaining a similarity score above a given threshold.

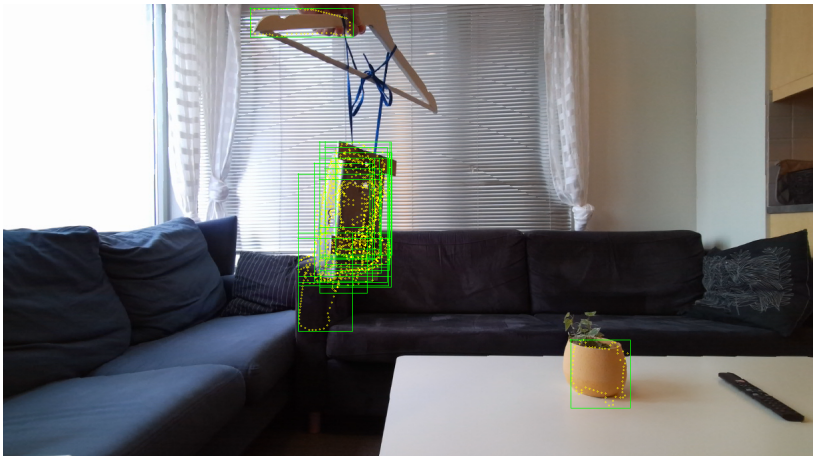


Figure 25: Examples of matches attained from the LineMOD detector. The green rectangles illustrate the matching frame, while the yellow dots illustrate the modality feature locations corresponding to the color gradient orientations.

The presented solution uses a total of 8250 templates, uniformly distributed on seven spheres with radii between 0.4 m and 1.0 m. In addition the implementation utilizes two different sampling steps, with $T_1 = 8$ and $T_2 = 10$, as this seemed to give the best results during the project thesis. Using these settings, the detection can be completed in approximately 0.2 seconds. Furthermore, the initialization, i.e. the loading of the detector,

only takes a few seconds.

For handling the RGB-D sensor data, *Azure Kinect Sensor Software Development Kit* [50] is used. In order to reduce computational time, the lowest resolution (1280 x 720) is utilized. This toolkit also provides functionality for aligning the depth image into the format of the color camera image, which is required for the lineMOD detector approach.

5.1.3 Match Clustering

As described in 3.2, initial translation estimates are required for the RBGT to start tracking an object. In order to eliminate false matches and pass on the n_b best object translations, clustering is applied to the LineMOD matches. Based on these translations, the RBGT will later find approximate rotations for the true objects, using the approach to be described in 7.2. The clustering itself is performed using a function called *kmeans* from OpenCV. By utilizing the *k-means algorithm*, all matches are first clustered solely based on their image coordinates (u_i, v_i). This initial clustering uses:

$$k_1 = \left\lceil \frac{n_{matches}}{2} \right\rceil \tag{52}$$

cluster centers, where $n_{matches}$ is the total number of matches. Next, the n_b biggest clusters are evaluated one by one. For each of these clusters, the match translations from equation 50 and 51 are retrieved. By performing a second round of clustering, this time based on the translations of the remaining matches, the single best translation is found for each of the n_b initial clusters:

$$\boldsymbol{t}_j = \boldsymbol{c}_{j,biggest} \qquad j \in \{1, ..., n_b\}, \tag{53}$$

where $\boldsymbol{c}_{j,biggest}$ is the center of the biggest cluster among the matches from initial cluster j. For the second round of clustering, the number of centers are calculated as:

$$k_{2,j} = \left\lceil \frac{n_{matches,j}}{2} \right\rceil \qquad j \in \{1, ..., n_b\}, \tag{54}$$

where $n_{matches,j}$ is the number of matches in initial cluster j. Depending on n_b and the number of total matches, the clustering can be completed in approximately $0.1 - 0.3s$.

5.2 RBGT

In order to set up the Region-Based Gaussian Tracker (RBGT), the required dependencies were first added to the project setup in *Microsoft Visual Studio*. Similar to the template generation method for the LineMOD detector, the RBGT also uses graphical libraries and extensions such as OpenGL, GLFW and GLEW. While version 3.1.2 or newer is required for the *Graphics Library Framework* (GLFW), version 2.1.0 of GLEW was linked to the project without having to explicitly add the source code. Otherwise, version 4.0.0 or newer of OpenCV is required for vector operations and image processing, while version

3.3.2 or newer of *Eigen3* is required for various matrix decompositions and geometry features. As mentioned in 3.2, the Azure Kinect is already set as the default camera in the RBGT framework. Accordingly, no customization is necessary for tracking objects with the provided camera. All camera functionality are thus obtained by simply adding *k4a* (1.3.0) from the *Azure Kinect Sensor Software Development Kit* to the project. In order to speed up the visualization features such as the rendering of object overlays (See figure 8), the utilization of a dedicated GPU was first considered. However, when running the optimized solution in Microsoft Visual Studio, the updating of the viewer is completed in just $11 - 13ms$ when tracking a single object. Although this is a fair amount of time compared to the time spent calculating and optimizing the likelihoods, it does not impact the real time performance of the tracker. Furthermore, as the visualization is not part of the actual tracking, this can easily be disabled to reduce the computational costs of the tracker in a real-time application.

Just as for the LineMOD detector, the RBGT tracker also requires 3D models of the objects to track. The 3D mesh model, illustrated in figure 23, was thus imported as a *Wavefront .obj* file as requested in the RBGT Github repository [49]. The repository also provides instructions for how to set up the tracker and required objects. After adding the path for the respective *obj* file and specifying the maximum body diameter, a model of the object is generated for you. This model includes the 2562 template views, along with 200 data points, or correspondence lines, for each associated view. The generated model is then saved for later use, before starting the tracker. [49] also includes code examples demonstrating how to run the tracker on either a prerecorded sequence, or a real-time camera sequence. As previously discussed, the tracker relies on being provided the initial 6-DoF poses for all objects to track. During the introductory testing, these were assigned manually by keeping the starting pose unchanged, or by refining the initial pose on a prerecorded sequence. The rendered object overlays were very helpful for this task.

6 RBGT Evaluation

In this section the first impressions of the RBGT method [48] will be presented in 6.1, before discussing what seems to be the biggest challenges in relation to our pose estimation problem in 6.2. These challenges will later be the basis for the presented solution improvements in section 7.

6.1 Initial Conclusions

After having implemented the RBGT, the method immediately demonstrated some very impressive results. Whether the tracker was running on live camera images, or prerecorded sequences, it performed even better than expected. In particular, the efficiency of the approach stood out, achieving a tracking frequency of $20Hz$ and $30Hz$ with and without visualization respectively. Furthermore, the precision of the pose estimates, as indicated by the object overlays, were often impeccable. One of the very first tracking runs, using live camera images, is illustrated in figure 26. As shown in figure 26a - 26d, the RBGT had no trouble keeping track of the moving objects. In addition, 26e and 26f

demonstrate the method's ability to handle occlusion. By rejecting occluded correspondence lines, the approach avoids losing track, even when significant portions of the object is no longer visible.

One of the natural limitation of the RBGT is that the frame to frame motion can not exceed the area covered by the correspondence lines. If exposed to significant translations perpendicular to the camera axis, the tracking is easily lost. In addition, notable rotations between frames can result in incorrect pose estimates as the object contours of the new frame no longer fit the correspondence lines associated with the previous pose. However, the high frequency of the method ensures that this most likely will not be a problem. For pure translations especially, the performance is very robust. As the object contour remains very much the same, the set of correspondence lines is more likely to contain the updated object contour. The object colors and the corresponding appearance models m_f and m_b from 4.4.2 will furthermore remain fairly unchanged as the surfaces visible for the camera will be the same. Other limitations such as contour ambiguity, insufficient color dissimilarity between object and background, and pose dependent color variation will be discussed in 6.2.

6.2 Challenges

As the RBGT is based on color values along correspondence lines and computed color histograms, having a prominently and uniformly colored object would undoubtedly make the tracking easier. During testing, incorrect likelihoods, illustrated by red segments along the yellow correspondence lines, were often responsible for deviating pose estimates. Figure 27 demonstrate how a slightly incorrect initial pose, along with color similarity between foreground and background can make the method lose track of the object. The dark surface on the table is clearly preferred over the far side of the object as illustrated in figure 27a and 27b. Finally, the incorrect pixel-wise posteriors and likelihoods cause the RBGT to lose track of the object in figure 27c and 27d. Naturally, it would be preferable if the tracker was able to recover from the incorrect pose estimate illustrated in figure 27a as the statistical models $p_t(\boldsymbol{y}|m_i)$ are iteratively updated. For objects with more distinct colors, such as the ape object from the RBOT dataset [55] (see figure 8), a corresponding pose recovery would most likely not be a problem. Although actions such as changing the background color would help, the tracker's ability to correctly identify the object contour could still be slightly better when exposed to less prominent object colors.

A somewhat similar case is illustrated in figure 28. While having a decent pose estimate initially, the deviation grows as the tracker is unable to produce correct contour likelihoods during the rotation of the object. Just as for the example in figure 27, the far side of the object seems to be rejected by the statistical foreground model. Pose dependent color variation and glare on the surface can arguably complicate the task of generating correct likelihoods. The pixel-wise posteriors in figure 28d also demonstrate this, as the marginally darker far side is considered too dissimilar for being part of the foreground. Again, the showcased tracking runs clearly do not present the ideal conditions for the RBGT. A different background would probably solve this by increasing the difference between $p(\boldsymbol{y}_i|m_f)$ and $p(\boldsymbol{y}_i|m_b)$. One could also argue that the metallic

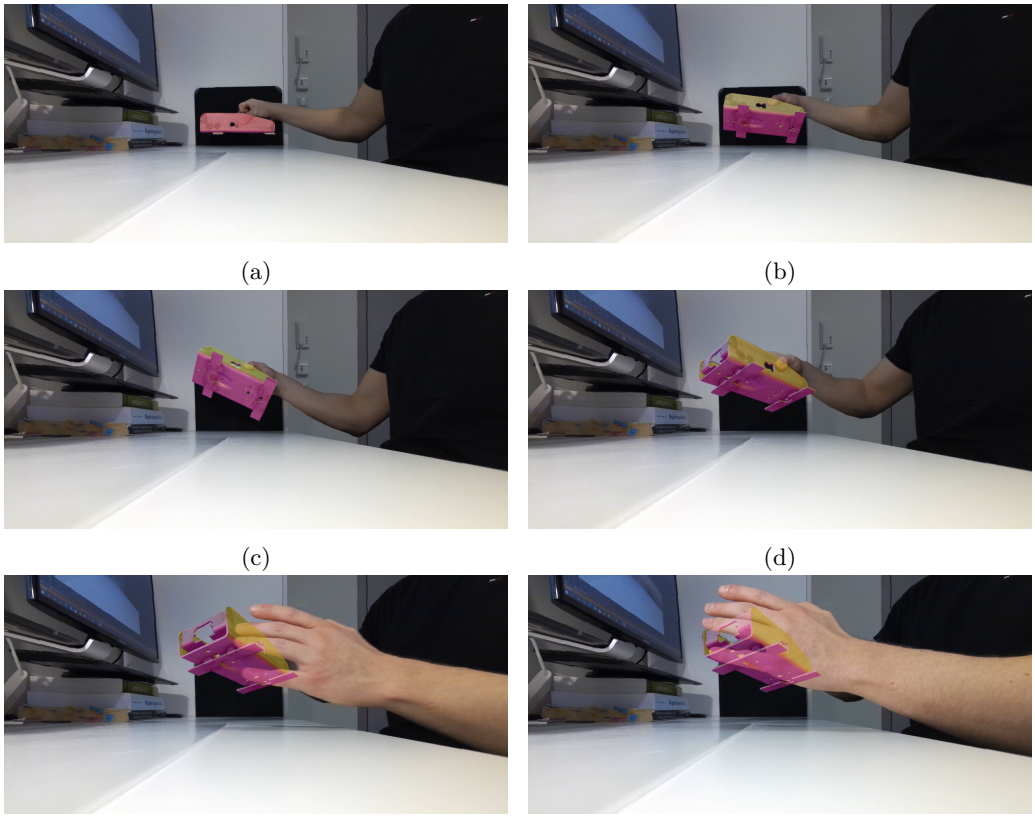


Figure 26: Images from a successful tracking run using live camera images. The precise 6-DoF pose estimates are visualized by the rendered object overlays. **e)** and **f)** also demonstrate the method's ability to handle occlusion.

object might not be the ideal target object for this tracker. However, as the RBGT still demonstrates some very promising results, even for these non-ideal conditions, it would be interesting to see how this approach can be modified to further improve the tracking performance. Naturally, it would be preferable if the implemented tracker solution was able to handle these sub-optimal conditions.

As discussed in 2.3 regarding region based methods in general, contour ambiguity can also introduce some challenges for the RBGT. In short, this challenge concerns the trackers ability to correctly estimate the object's pose when the contour likelihoods can be explained by multiple pose variations. In particular, the rotation illustrated in figure 29 can cause incorrect pose estimates, as the opposed pose variations give rise to some rather similar contour likelihoods. In contrast to the previously presented challenges, contour ambiguity can still cause problems despite having decent statistical models for foreground and background. However, having accurate likelihoods can undoubtedly reduce the prob-

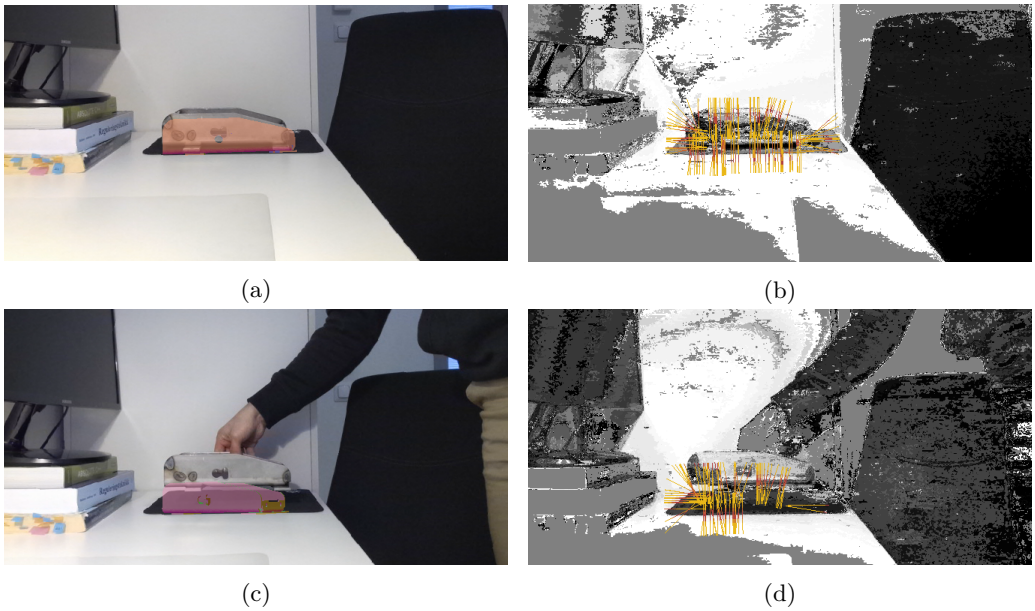
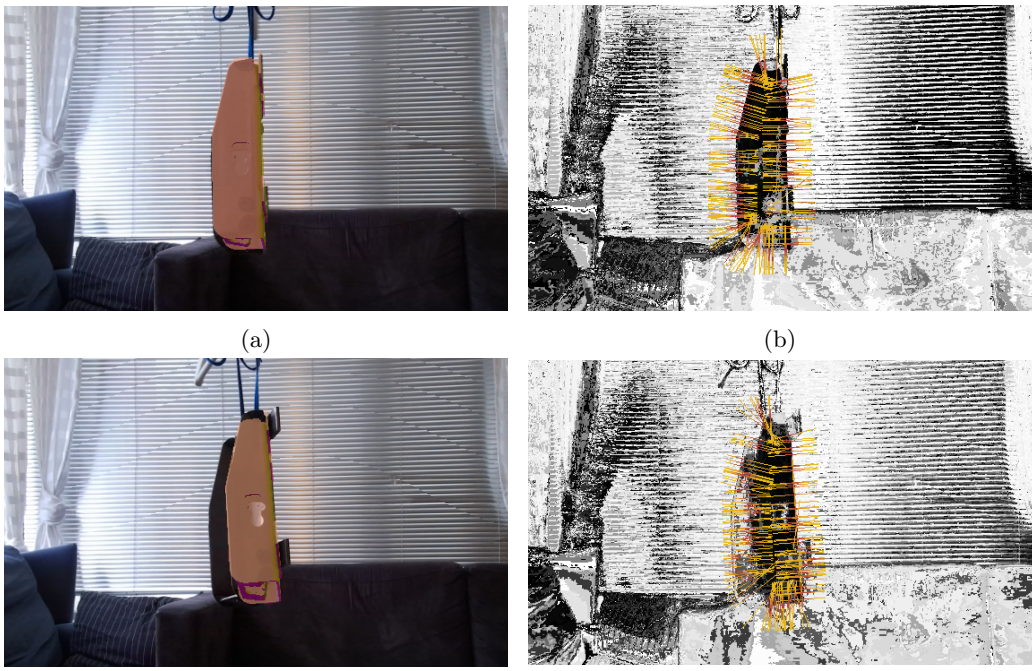


Figure 27: Example illustrating tracking loss. **a)** and **c)** on the left show how the tracking is lost as their corresponding contour likelihoods in **b)** and **d)** don't agree with the actual object.

ability for experiencing this problem. Rotations prone to contour ambiguity, such as the one illustrated in figure 29 become more challenging with increased rotation velocity. If first exposed, the estimation can not be corrected by accurate likelihoods. Accordingly, the pose estimate is likely to remain incorrect if the object rotation is not reversed to it's previous state.

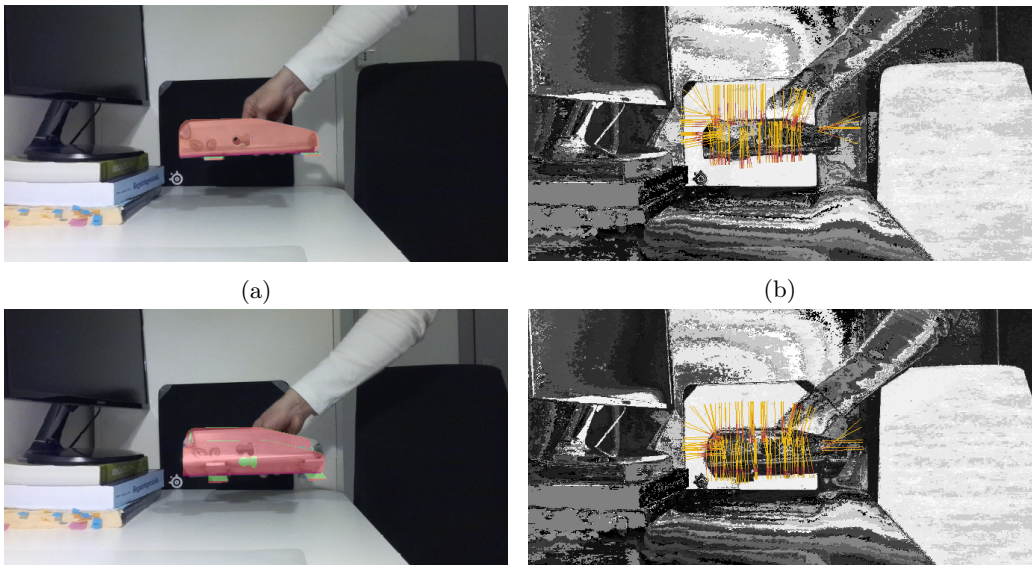
A final, and natural limitation of the RBGT is that it requires a decent initial pose estimate in order to start the tracking. This is required for computing the associated correspondence lines and initializing the statistical models for the foreground and background. As discussed in 3.2, the detection part of the proposed pipeline is only capable of providing the translation of the detected object, not the rotation. This is not sufficient for the RBGT. Although small deviations can be tolerated, rotation knowledge is vital for being able to track the object. By utilizing an assumption for initial rotation, and the RBGT method itself, a solution for full pose detection is presented in section 7. In addition, the integration of depth data is proposed in order to improve the performance of the tracker. In particular, the use of depth data should help the tracker when exposed to insufficient color dissimilarity between objects and their background. Finally, a solution for detecting and correcting imprecise pose estimates, such as the ones in figure 28c and 29c is presented.



(a) (b)

(c) (d)

Figure 28: Example demonstrating pose estimation drift as the tracking object rotates. **a)** and **c)** show the estimated poses, while **b)** and **d)** show their corresponding pixel-wise posteriors and contour likelihoods. The method clearly struggles to identify the true object contour.



(a) (b)

(c) (d)

Figure 29: Example illustrating the effect of pose ambiguity. The real pose variation from **a)** to **c)** is misinterpreted as the real pose and the opposed estimated pose have quite similar contours. **b)** and **d)** show that the tracking fails despite having decent contour likelihoods.

7 Solution Improvements

In this section several additions and improvements are proposed for the implemented solution. First, the integration of depth data into the RBGT [48] is proposed in 7.1. Next, a solution for deciding initial object poses is presented in 7.2, before finally suggesting an approach for detecting and correcting inaccurate pose estimates in 7.3.

7.1 Integration of Depth Information

Following a general formulation of the depth based *edge probabilities* in 7.1.1, an approach for processing the depth data is described in 7.1.2. A Gaussian approximation for the edge probability is then proposed in 7.1.3, before discussing the combination of color based contour probabilities and depth based edge probabilities in 7.1.4. Finally, a brief evaluation of the modified RBGT is presented in 7.1.5.

7.1.1 General Formulation

Ren et. al [39] and Kehl et. al [22] both propose strategies for incorporating depth information into region based tracking methods. While [39] adds pixel depth to the generative model for deciding whether or not a pixel is part of the foreground, an ICP based term is added to the energy function in [22]. Although both demonstrate improved performance compared to their color-only counterpart, a different strategy would seem more fitting for this solution. Hence, inspired by the sparse nature of RBGT [48], a strategy utilizing depth data along the correspondence lines is proposed.

By analysing depth information along the correspondence lines, the idea is to find distinct edge probability distributions derived from depth discontinuities. The discrete scale-space formulation from 4.4.5 can thus be used to find $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$ by varying the distance between each depth evaluation according to scale s. $\boldsymbol{\mathcal{E}}_i$ is defined as the depth data along correspondence line i. The total probability for a correspondence line can accordingly be stated as the normalised sum of $p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si})$ and $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$. Furthermore, if assuming n_{cl} correspondence lines, the full likelihood can roughly be calculated as:

$$p(\boldsymbol{\mathcal{T}}|\boldsymbol{\theta}) \propto \prod_{i=1}^{n_{cl}} \big(p(\boldsymbol{\mathcal{D}}_i|\boldsymbol{\theta}) + p(\boldsymbol{\mathcal{E}}_i|\boldsymbol{\theta})\big). \tag{55}$$

More details on how the probabilities are combined is provided in 7.1.4. In order to approximate the likelihood for arbitrary $\boldsymbol{\theta}$ and corresponding $\Delta\tilde{c}_{si}$, the linear interpolation from equation 32 is used. The concept of combining both depth and color information along the same correspondence lines is illustrated in figure 30. The data along the correspondence lines in 30a and 30b give rise to the probabilities shown in 30c.

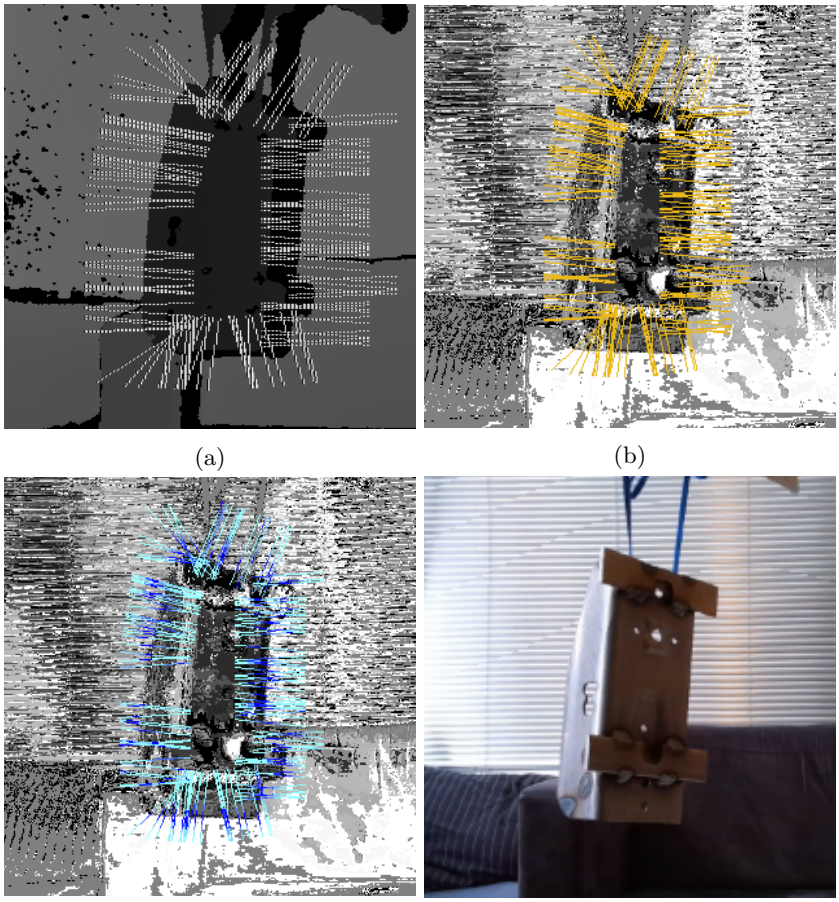


Figure 30: Example illustrating the concept of combining depth and color information. **a)** Depth image with correspondence lines, along which 12 pixel values are evaluated. **b)** Pixel-wise color posteriors with correspondence lines, along which 20-100 pixel values are evaluated depending on scale s. **c)** The resulting combined likelihoods corresponding to $\Delta\tilde{c}_{si} \in \{-5, ..., 5\}$. High probabilities are indicated by dark blue line segments. **d)** Original color image.

7.1.2 Depth Data Processing

In order to derive the edge probability likelihoods $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$, a data processing procedure must first be decided. For this, a total of 12 depth image pixels are evaluated along each correspondence line. The depth data from an arbitrary correspondence line in figure 30a is presented in figure 31a. Each depth measurement is connected to the discretized projected difference value $\Delta\tilde{c}_{si} \in \{-5.5, ..., 5.5\}$, ranging from the innermost to the outermost line tip.

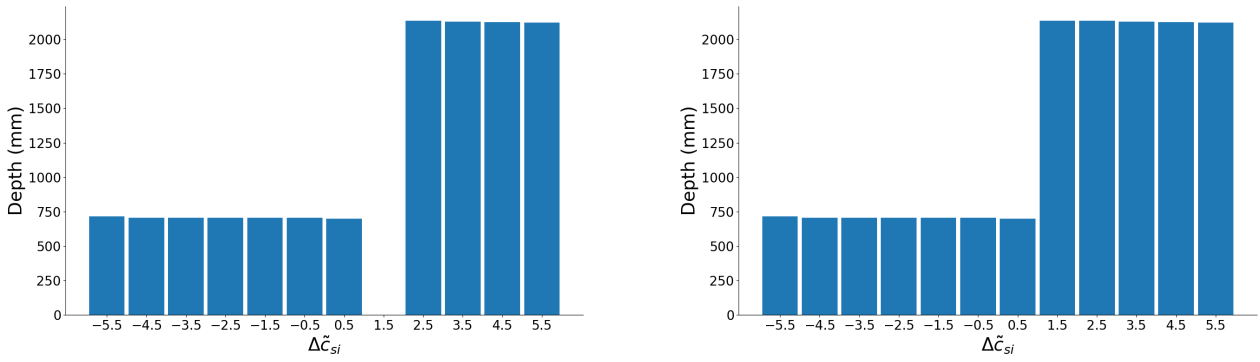


Figure 31: Example of depth data along an arbitrary correspondence line for $\Delta\tilde{c}_{si} \in \{-5.5, ..., 5.5\}$. **a)** Raw depth data. **b)** Edited depth data where the missing values have been replaced by the neighboring measurement at the right hand side.

Figure 31a demonstrate a common limitation of depth sensing cameras. By casting modulated near-IR (NIR) illumination into the scene, the Azure Kinect DK indirectly measures depth by implementing the Time-of-Flight (ToF) principle. Missing depth data, illustrated by zero depth, can thus emerge at surfaces distracting infrared light, or at occluded surfaces due to the baseline between the NIR emitters and the depth camera. Missing data caused by occlusion can be observed as "shadow" on the right side of the object in figure 30a. In addition, pixels at object edges are often invalidated as these contain the mixed signal from foreground and background [52]. The missing depth data from $\Delta\tilde{c}_{si} = 1.5$ in 31a is probably caused by the latter. Another limitation concerning the use of depth images is the accuracy of image alignment. While assuming perfect alignment between color and depth image, small deviation do occur. Consequently, some additional uncertainty must be accounted for when utilizing the depth based edge likelihoods $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$.

In order to handle missing depth data along correspondence lines, these are set equal to their neighboring measurement, one step further away from the object center. This procedure is illustrated in figure 31b, where the value from $\Delta\tilde{c}_{si} = 2.5$ is asserted at $\Delta\tilde{c}_{si} = 1.5$. If depth data is missing on either far side of the correspondence line, all depth information is neglected and $p(\boldsymbol{\mathcal{T}}_i|\Delta\tilde{c}_{si}) = p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si})$. Testing shows that including these data segments would only add noise to the likelihoods. Furthermore, having zeros at the outermost depth measurements will hinder the zero-filling from figure 31. The depth data $\boldsymbol{\mathcal{E}}_i$ is also invalidated if the number of zeros exceeds a defined threshold t_z. For the implementation, $t_z = 4$ as this seem to allow for an appropriate number of reliable depth data lines. By including data lines with $t_z > 4$, testing shows reduced performance as less accurate edge probabilities will mislead the tracker. The remaining selection of depth data is illustrated by blue correspondence lines in figure 32.

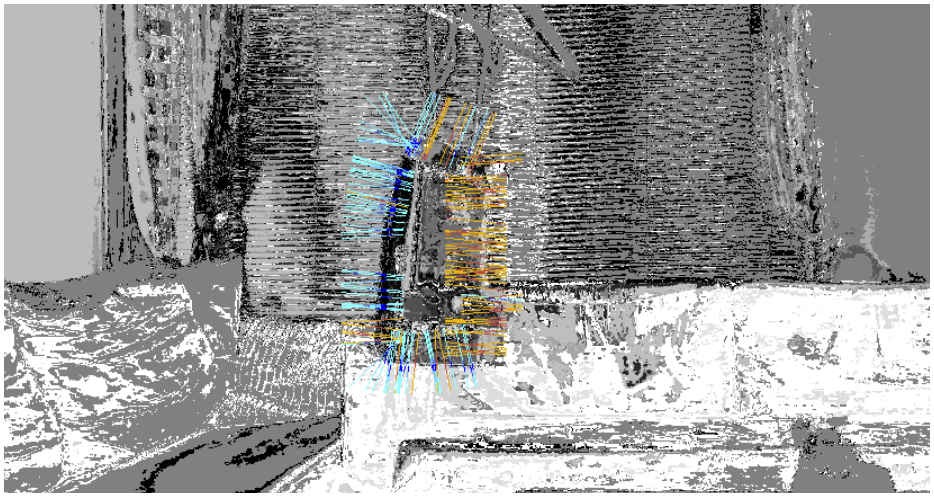


Figure 32: Example showing the remaining selection of depth data lines (indicated by blue correspondence lines) after neglecting those missing depth data at $\Delta\tilde{c}_{si} = -5.5$ or $\Delta\tilde{c}_{si} = 5.5$, and those missing a total of $n_z > t_z = 4$ depth values. Note that the posteriors at this frame have not yet been properly initialized, as this is the first iteration of the tracking run. However, the utilization of depth data clearly assist the initialization by accurately pointing out the object contour.

In order to derive the associated edge likelihoods $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$ for these correspondence lines, the depth difference between each depth value is first calculated. This way, the 12 depth values for $\Delta\tilde{c}_{si} \in \{-5.5, ..., 5.5\}$ are now turned into 11 depth differences $\Delta\tilde{c}_{si} \in \{-5, ..., 5\}$. If non of the calculated depth differences exceeds a given threshold t_{dd}, the depth data for this line is neglected. The calculated depth differences from figure 31 are illustrated in figure 33, along with a depth difference threshold of $t_{dd} = 100$ mm. By utilizing this threshold, internal depth discontinuities on the object itself are ignored. This threshold should naturally be adjusted according to the intended tracking object. Although lower thresholds can be applied to capture smaller depth discontinuities for some use cases, such as when tracking objects on flat surfaces, this can easily have an adverse effect as local depth differences withing the object overshadow the depth differences at the object contour. By making sure that no internal depth discontinuities are considered, an additional tactic can ensure that depth differences in the background are also disregarded. This *second edge suppression* is achieved by simply ignoring all depth differences on the far side of the fist depth difference bigger than t_{dd}. For the example in figure 33, an even bigger discontinuity at $\Delta\tilde{c}_{si} \in \{2, 3, 4, 5\}$ would accordingly not be evaluated. This addition improved the tracking performance significantly when exposed to cluttered scenes with notable depth variation in the background. As negative depth differences indicate object occlusion rather than object contours, these are neglected entirely. The position of the single remaining depth difference $\mu_{\mathcal{E}_i} \in \{-5, ..., 5\}$ is now the only value of interest. The depth difference value has no influence on the likelihood $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$.

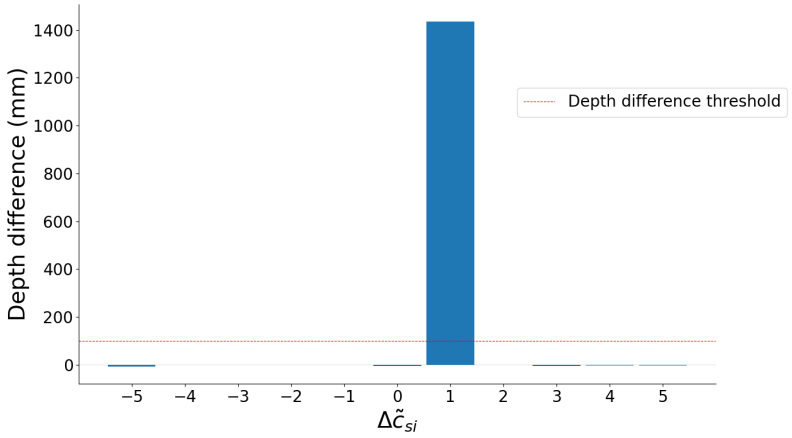


Figure 33: Example showing the depth differences corresponding to the depths in figure 31. If no depth differences exceed the depth difference threshold t_{dd}, the affiliated depth data line is neglected. Through *second edge suppression*, only the first "edge" exceeding t_{dd} is used for designing the edge probabilities $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$.

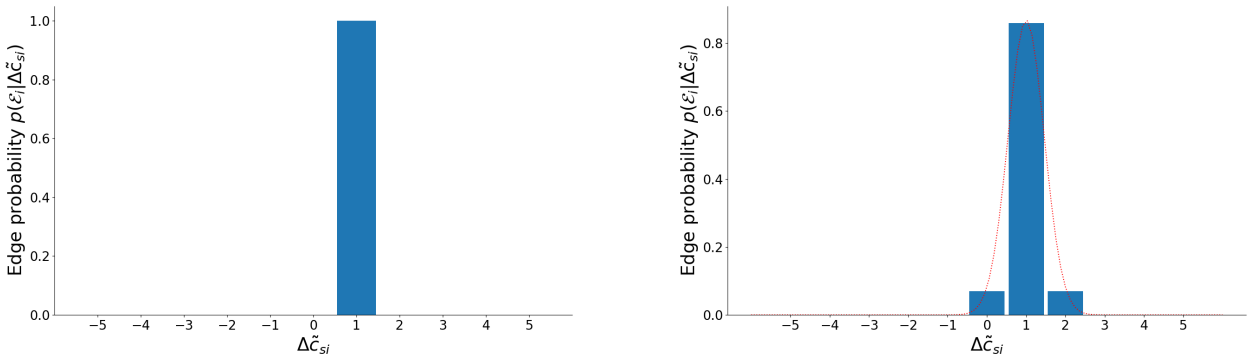
7.1.3 Gaussian Approximation

As pointed out in [48], Gaussian distributions are preferred when using a Newton-based optimization scheme. Consequently, similar to the contour likelihood function $p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si})$, the edge likelihoods $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$ should also have Gaussian characteristics. Rather than using a binary probability distribution with $p(\boldsymbol{\mathcal{E}}_i|\mu_{\mathcal{E}_i}) = 1$ as illustrated in figure 34a, the probability for each discretized projected difference is calculated as:

$$p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si}) \propto \exp\frac{(\Delta\tilde{c}_{si} - \mu_{\mathcal{E}_i})^2}{2\sigma_{\mathcal{E}}}, \tag{56}$$

where $\mu_{\mathcal{E}_i}$ is the position of the respective depth discontinuity, and $\sigma_{\mathcal{E}}$ is the desired standard deviation for the Gaussian approximation. Using the depth data example from earlier, and $\sigma_{\mathcal{E}} = 0.2$, the edge likelihood from figure 34b is attained.

For the implemented solution $\sigma_{\mathcal{E}} = 0.1$ is utilized as this value seems to give the best performance. For difficult rotations, which require very high accuracy in order to deal with contour ambiguity, $\sigma_{\mathcal{E}} > 0.1$ will make the likelihoods too imprecise. On the other hand, standard deviations lower than 0.1 might be a hindrance to finding the global optimum. However, testing shows that using a Gaussian approximation with $\sigma_{\mathcal{E}} < 0.1$ still improves the tracking performance compared to the binary counterpart from figure 34a.



(a) (b)

Figure 34: Example illustrating edge probability distributions. **a)** Binary representation with $p(\boldsymbol{\mathcal{E}}_i|\mu_{\mathcal{E}_i}) = 1$. **b)** Gaussian approximation based on the position of the depth discontinuity $\mu_{\mathcal{E}_i}$, and a predetermined standard deviation $\sigma_{\mathcal{E}}$. Here $\mu_{\mathcal{E}_i} = 1$ and $\sigma_{\mathcal{E}} = 0.2$

7.1.4 Combining the Probabilities

When having computed the edge probability distributions $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$, the total combined likelihoods $p(\boldsymbol{\mathcal{T}}_i|\Delta\tilde{c}_{si})$ can be decided. The simple summation from equation 57 provides a feasible solution by giving equal influence for both the color- and depth based distributions. This strategy is illustrated in figure 35. By adding the two Gaussians from 35a and 35b, the total probability distribution in 35c clearly preserve the essential qualities of both likelihoods. Furthermore, the low variance edge likelihood has the ability to determine the optimum of the total distribution, as $p(\boldsymbol{\mathcal{E}}_i|\mu_{\mathcal{E}_i}) > p(\boldsymbol{\mathcal{D}}_i|\mu_{\mathcal{D}_i})$. Figure 35 also demonstrate this, as the distribution optimum is shifted from $\Delta\tilde{c}_{si} = 0$ to $\Delta\tilde{c}_{si} = 1$ when adding the depth based edge likelihood. For challenging tracking scenarios including low foreground-background disparity and risk of contour ambiguity, this can be very beneficial as the edge likelihoods appear to be more precise and sturdy. Due to this observation, an increased edge probability impact was also applied for the total likelihood composition. Using a depth ratio r_d of 2.0 seems to result in the best tracking performance. The combined probabilities are thus calculated as:

$$p(\boldsymbol{\mathcal{T}}|\Delta\tilde{c}_{si}) \propto \prod_{i=1}^{n_{cl}} \big(p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si}) + r_d p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})\big). \tag{57}$$

If utilizing a lower depth ratio r_d, conflict can easily occur between incorrect contour likelihoods and the more sturdy edge likelihoods. This is also the case for the failing tracking run illustrated in figure 36. For the combined likelihoods, an equally divided influence can cause the incorrect color probabilities to win, as some of the depth segments are omitted due to noise. For the subsequent pose refining iteration with reduced scale, the left contour from figure 36a is no longer part of the correspondence lines. Accordingly, no depth data can recover the true pose. In contrast, $r_d = 2.0$ can ensure that the validated

depth segments are prioritized over conflicting color data, which again can evade drifting of the histogram based appearance models. Conflicting contour- and edge likelihoods, as illustrated in figure 37, are also the reason for why no single Gaussian can be derived for the total probability distribution. As these offer some very different qualities, there are no simple way of combining the two. By instead emphasising the validated depth data, the robustness of the tracker is undoubtedly improved.

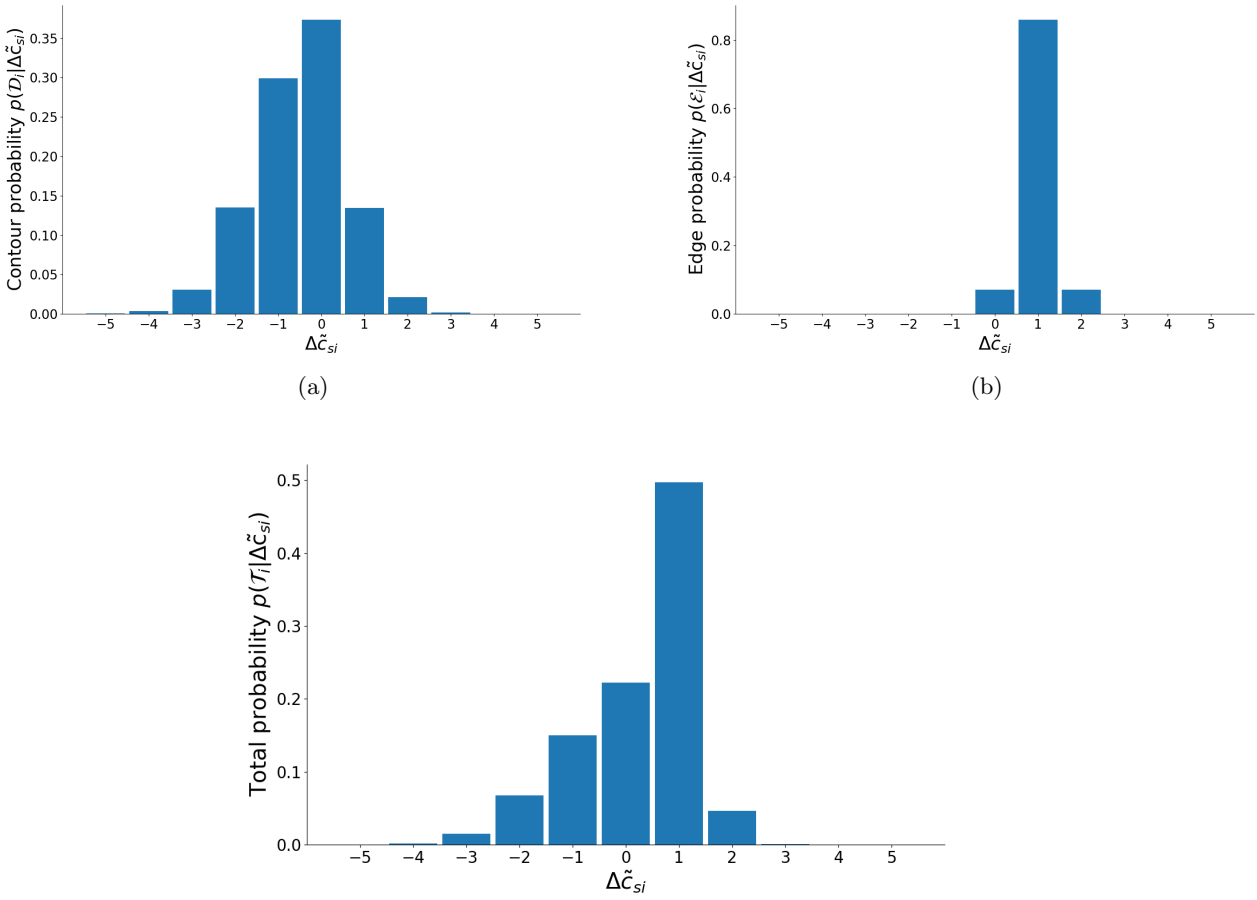
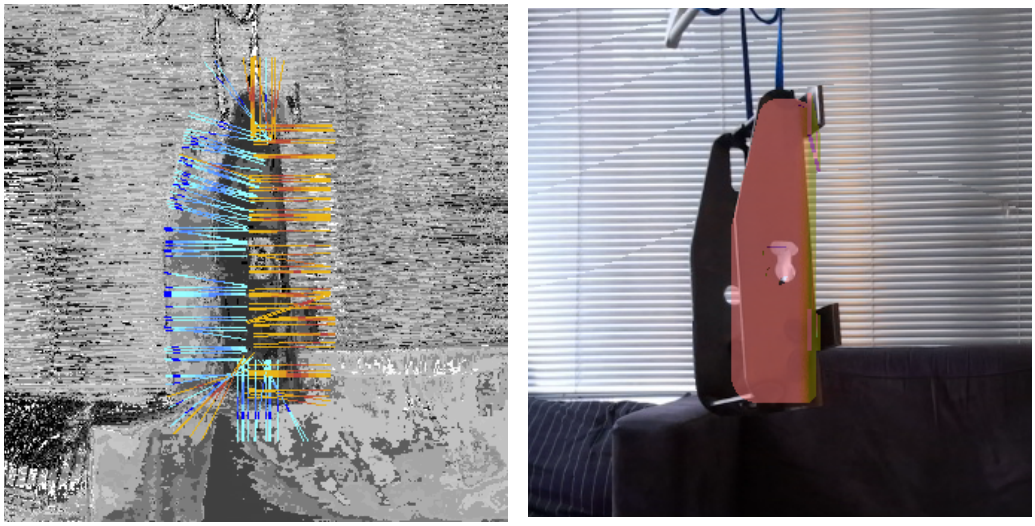


Figure 35: Example illustrating the summation of contour- and edge likelihoods ($r_d = 1$). **a)** The contour probability distribution $p(\boldsymbol{\mathcal{D}}_i|\Delta\tilde{c}_{si})$. **b)** The edge probability distribution $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$ **c)** The total combined probability distribution $p(\boldsymbol{\mathcal{T}}_i|\Delta\tilde{c}_{si})$.

While higher depth ratios can be beneficial for some difficult rotations with contour ambiguity, using $r_d > 2$ will also make the tracker more prone to noisy and inaccurate



(a) (b)

Figure 36: Example illustrating conflict between color and depth data when using $r_d = 1$. In order to recover the true pose, the depth based likelihoods must be given more influence on the total likelihoods. **a)** An illustration of the calculated likelihoods at the initial and largest scale. **b)** The corresponding estimated pose.

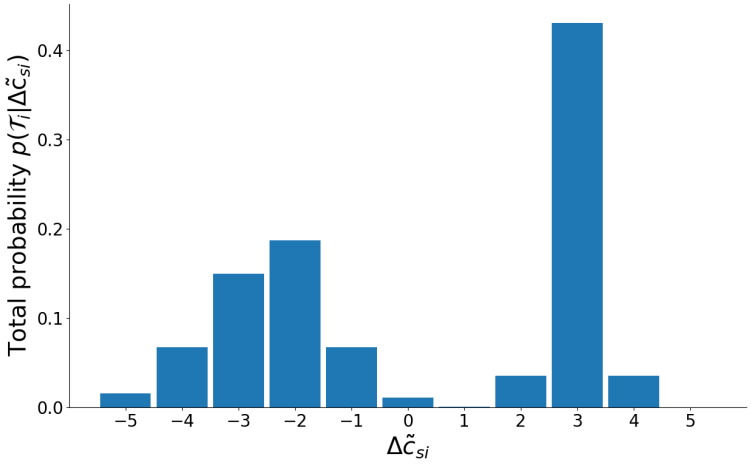


Figure 37: Example illustrating conflicting contour- and edge likelihoods. Naturally, no joint Gaussian approximation would not be able to describe their distinct features.

depth measurements. Whether the inaccuracies are caused by missing data, or by incorrect image alignment, these can still result in small deviations between the true and

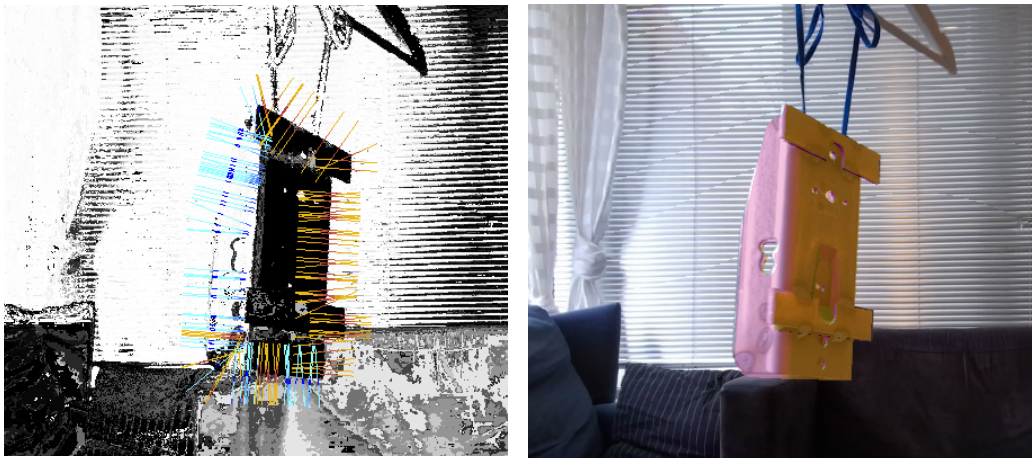
estimated pose. The occurrence of imprecise depth based edge likelihoods is particularly noticeable at lowest scale, where a pixel perfect edge location is requested. Due to the binary nature of the edge data, these deviations are inevitable for the depth based likelihoods. However, increasing the variance from $\sigma_{\mathcal{E}}$ to $2\sigma_{\mathcal{E}}$ for the smallest scale appears to increase performance slightly as the tracker account for reduced precision at pixel level. Initially, it was considered to only utilize depth data at the largest scale. This was clearly not good enough as potentially conflicting and incorrect contour likelihoods causes the pose estimate to diverge during the remaining iterations. On the other hand, utilizing depth data for all 7 iterations can lead to small pose estimate deviations due to the reduced precision at the lowest scale. This can furthermore result in incorrect rotation estimates when having to deal with contour ambiguity. Consequently, the edge probabilities $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$ are calculated and utilized for the first 6 iterations, leaving the final low scale iteration with color based likelihoods only. The idea is that the depth information then has guided the final set of correspondence lines to the true contour of the object.

7.1.5 Evaluation

All in all, the implementation of depth information has improved the tracker greatly. By utilizing the very sparse and effective depth data evaluation strategy, the modified RBGT clearly outperforms the original tracker [48] while practically keeping the computational costs unchanged. In fact, using the laptop described in the intro of section 5, all 7 iterations are still completed in just $2-3\ ms$. The few extra milliseconds spent collecting the depth image from the camera capture is also negligible compared to the total, and the previously mentioned tracking frequencies of $20Hz$ and $30Hz$, with and without visualization, are still attained. An example illustrating the great advantage of adding the supplementary depth based likelihoods is shown in figure 38. As the color based appearance models clearly disregard the leftmost part of the object, there would be no way to keep track without the additional depth information.

Figure 39 and 40 furthermore demonstrate the difference in tracking performance before and after utilizing the described depth integration solution. Each row of images in figure 39 illustrate a situation in which the original RBGT [48] fails to correctly estimate the object pose. Like the presented cases in 6.2, these samples also portray faulty appearance models along with contour ambiguity when exposed to difficult rotations. The faulty appearance models, and associated contour likelihoods, are particularly evident in figure 39a - 39d. The corresponding tracking samples from figure 40a - 40d show great improvement. Even though the appearance models still struggle to correctly define the object contour, the low variance edge probabilities $p(\boldsymbol{\mathcal{E}}_i|\Delta\tilde{c}_{si})$ manage to guide the tracker towards the correct pose estimates. Finally, figure 40e and 40f show how the increased precision from the depth based likelihoods can prevent incorrect pose estimates when exposed to contour ambiguity.

Although additional testing should be done in the intended lab setup described in section 1, there is no doubt that the modified tracker shows some excellent potential. After applying features like the depth data filtering, along with a suitable depth difference threshold, and second edge suppression, there has been no situations where the modified

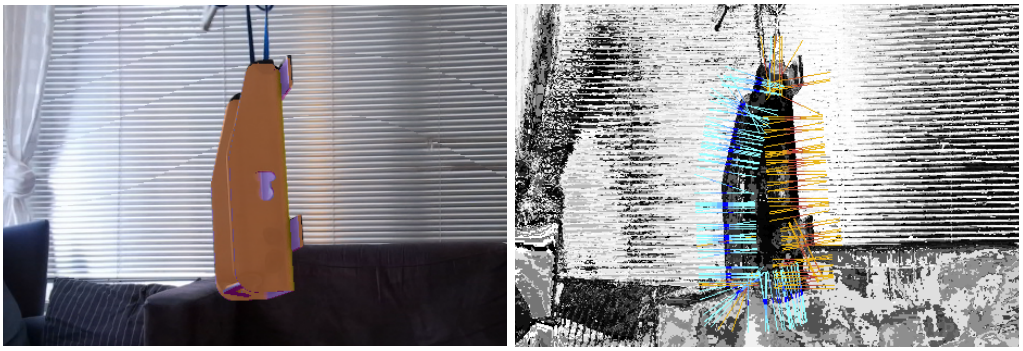


(a) (b)

Figure 38: Example demonstrating the impact of utilizing depth information. **a)** The pixel-wise posteriors indicate $p_b \approx 1$ for the leftmost part of the object. However, the depth based likelihoods make sure that the tracker maintains the correct pose estimate. **b)** The resulting pose estimate.

solution performs worse than the original RBGT [48]. In fact, the main source for tracking error with the modified solution appears to be missing depth data. Due to occlusion, light distracting surfaces, and ambiguous depth measurements nearby edges, the number of viable depth data segments might not always be sufficient for the most challenging rotations.

Figure 39: Samples from a tracking run using the original RBGT method [48]. Incorrect appearance models and corresponding posteriors causes the tracker to misinterpret the object's rotation in **a)** - **d)**. In **e)** and **f)** the imprecise nature of the contour likelihoods lead the tracker to pick out the opposed pose variation.



(a) (b)

(c) (d)

(e) (f)

Figure 40: Tracking samples corresponding to those in figure 39 when using the modified tracker solution with depth data utilization. All samples show great improvement as the incorrect contour likelihoods are overcome in **a)** - **d)**, and pose ambiguity is averted in **e)** and **f)**.

7.2 Initial Pose Detection

In this subsection the motivation for developing a new pose detection solution is first discussed in 7.2.1 before presenting the solution itself in 7.2.2. Finally, a brief evaluation of the implemented solution is given in 7.2.3.

7.2.1 Motivation

As discussed in the revised pipeline proposal in 3.2, a solution for finding the object's pose is required for initializing the RBGT [48]. Rather than having to set these manually, it would be better if all initial poses could be determined through an automated scheme. For an actual real world application this is an essential quality. The pose estimation pipeline should be able to operate in a changing environment with multiple tracking objects. If considering an assembly process, a robotic manipulator should thus be able to detect new objects as these arrive the workstation. For the preliminary manufacturing setting described in section 1, this can for instance correspond to detecting chair parts hanging from the roof mounted conveyor. In order to apply the modified RBGT to these parts, a new strategy is required in order to convert the attained LineMOD matches into initial object poses. While the rotations for the matches can be somewhat unreliable, the translations in general seem a lot more trustworthy. Consequently, the clustered translations should make a decent starting point for finding the initial 6-DoF poses.

7.2.2 Solution

As described in 5.1.3, the matches from the LineMOD [19] detector are first clustered based on their image coordinates. The n_b best clusters are then going through a secondary clustering process based on the translation of the matches. Here, the single most promising cluster center is chosen as the translation candidate for each initial cluster. An example illustrating the matches belonging to the $n_b = 3$ biggest clusters is shown in figure 41. For this example the final translation candidates become:

$$\begin{bmatrix} -0.104 & -0.039 & 0.700 \end{bmatrix}, \begin{bmatrix} -0.109 & 0.047 & 0693 \end{bmatrix}, \begin{bmatrix} -0.090 & -0.044 & 0.715 \end{bmatrix}, \quad (58)$$

where the coordinates describe the position in meters relative to the camera. In order to start the tracker, the initial object pose was first set equal to the translation of the biggest cluster center, while using the default object rotation (see figure 42a).

As long as the initial rotation did not differ too much from the fixed configuration, the RBGT seemingly had no trouble initializing the tracking process. However, the tracker should be able to track objects with varying initial rotations. Furthermore, the pose detection solution should also be able to disregard false detections. This gave rise to an approach which involves carrying out trials for a selection of potential initial object poses. Although the fixed-rotation strategy is a bit far fetched, it is not unreasonable to have some assumptions regarding the rotation of the object. In particular, the frequently displayed hanging scenario clearly restrict the variation in object rotation. Except for variation about the up-down axis, the rotation remains more or less the same. As this setting is quite similar to the intended hanger use case from the lab setup, this restricted

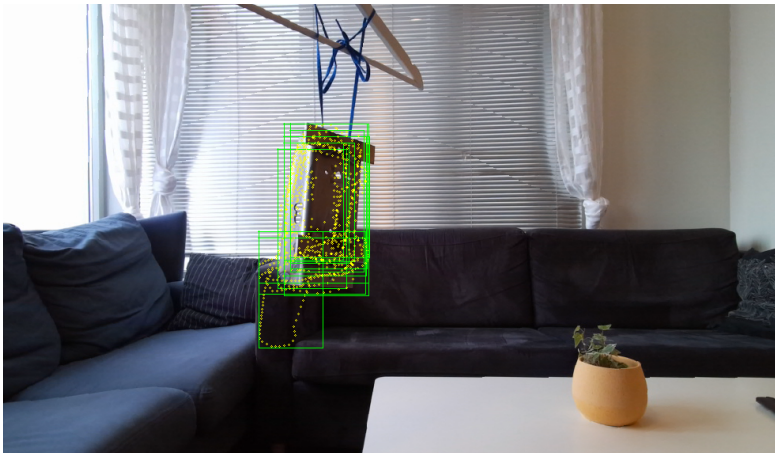
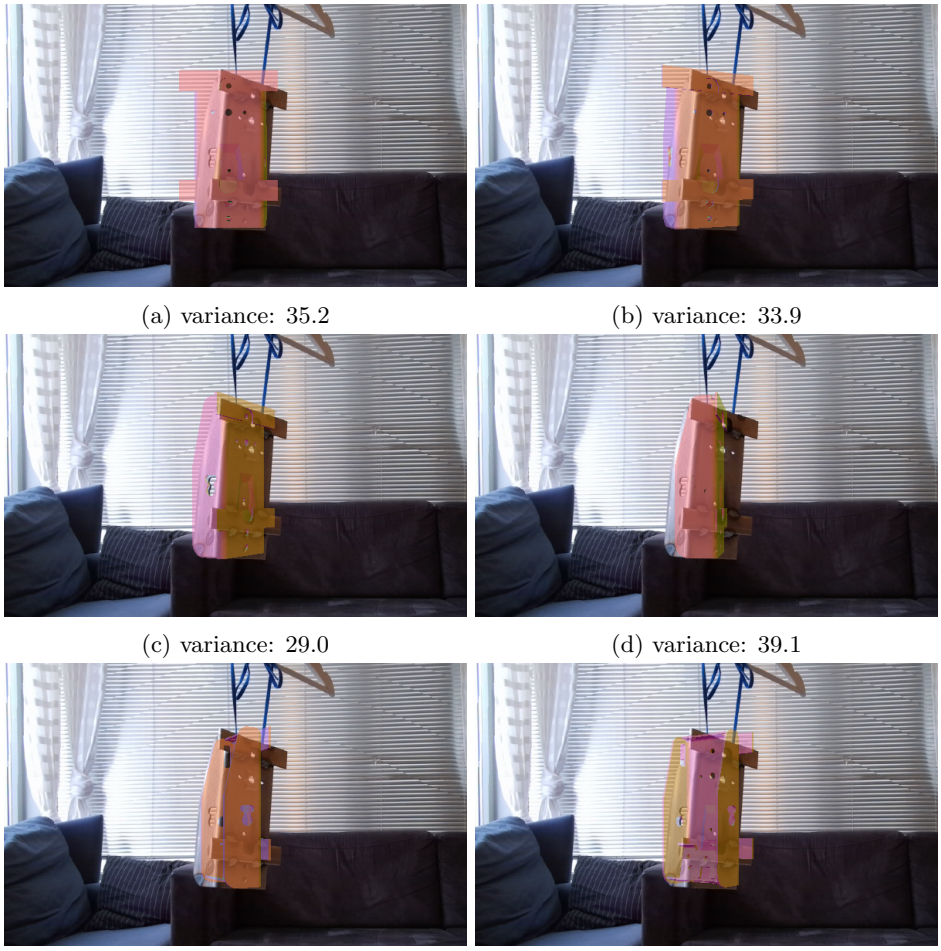


Figure 41: Example illustrating the LineMOD matches belonging to the $n_b = 3$ biggest clusters.

rotation space arguably make a sensible assumption for the initial rotation. By utilizing a performance measure for the different rotation candidates, the pose detection approach ensures that the tracker is given the rotation which best fit the real initial pose. If non of the candidates yield an acceptable score, their associated translation is neglected, and the same procedure is repeated for the translation next in line. This approach furthermore allows for detection of multiple objects. However, this requires an additional step checking whether or not the euclidean distances between objects are large enough. If not, one object can give rise to multiple object detections.

In order to evaluate the selection of initial rotations, one single tracking cycle is performed for each corresponding pose candidate. The average likelihood variance is then summarised over all 7 pose correcting iterations. As the likelihood variance is already calculated for the optimization, this can be done rather seamlessly. These variance scores give a good indication regarding the suitability of the rotation. More accurate initial poses will result in more accurate appearance models, as parts of the background will not contribute to the foreground model, and vice versa. Accordingly, the variance will be higher for incorrect rotation candidates. In addition, the low variance depth based edge likelihoods will increase the variance of the total likelihood when diverging from the color based contour likelihoods. An example illustrating this approach is shown in figure 42. By utilizing an incremental step of 30° around the up-down axis, a total of 12 pose candidates are tested. Half of these are shown in the figure. This example clearly demonstrate the value of the variance evaluation, as the candidate from 42c is correctly chosen as the best fit for the initialization. The efficiency of this approach is also a bonus, as 12 different rotations can be evaluated in just 32 ms. The implemented pose detecting solution also allows for angular variation along multiple axes. For a more extreme case, all axes could have an incremental angle step of 45°, giving $8^3 = 512$ different rotation candidates. However, it would take the pose detector approximately 1.1 s to evaluate all of them.



(a) variance: 35.2 (b) variance: 33.9

(c) variance: 29.0 (d) variance: 39.1

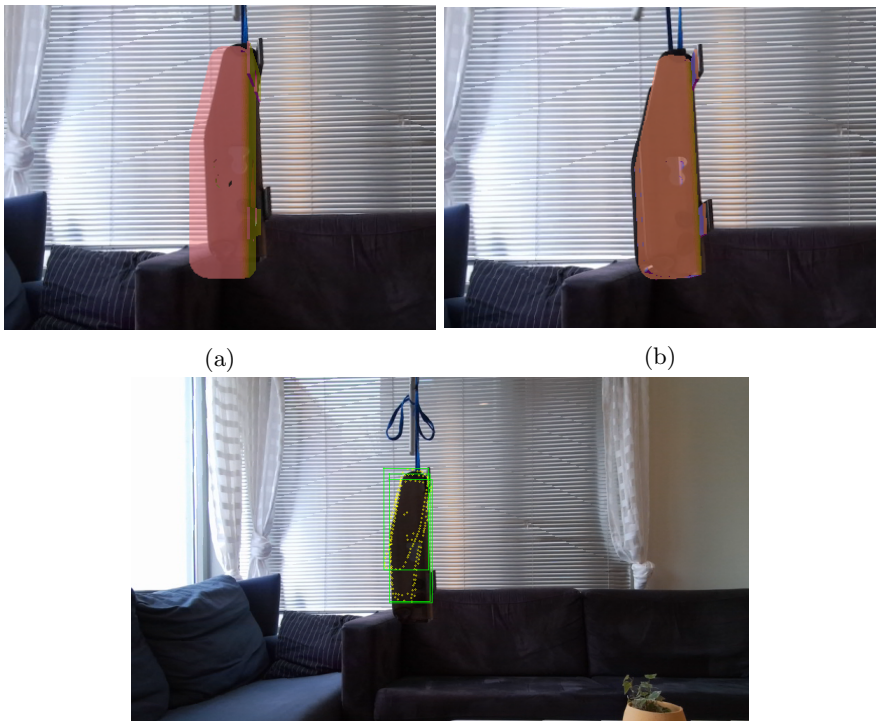
(e) variance: 39.3 (f) variance: 36.6

Figure 42: Illustration of the pose detection approach used for finding the initial rotation which best fit the true object. The total average variances, indicating the accuracy of the suggested poses, are also disclosed for the displayed rotation candidates.

If this procedure furthermore has to be repeated for multiple translation candidates, the computational time would naturally be far too high for achieving accurate tracking initializations when dealing with moving objects. As the implemented LineMOD detector and subsequent clustering alone spend approximately 0.4 s finding the best translation candidates, the number of rotation candidates should not be kept higher than necessary. Any justified assumptions reducing the potential rotation space are therefore of great value. For the single-axis angle variation case, as illustrated in figure 42, an angle step of 30° appears to make a good fit.

7.2.3 Evaluation

In general, the presented pose detection solution has showed some great results. The testing of different initializations appears to give a good indication on the quality of the respective pose candidates. If provided some viable potential translations from the LineMOD detector and subsequent clustering, the pose detector should have no trouble finding suitable object poses for the tracker initialization. The approach is also very efficient, which furthermore enables the pose detector to evaluate 12 different rotation candidates faster than the average frame duration of the tracker. Even though this would not be possible without the very practical reduced rotation space assumption, the presented use case arguably allow this simplification. By adding customised incremental variation steps about multiple axes, the solution would however be able to correctly detect additional object poses while still keeping the number of rotation candidates fairly low. Another great aspect of this pose detecting solution is that it does not require accurate LineMOD detections in order to produce satisfactory initial poses for the tracker.

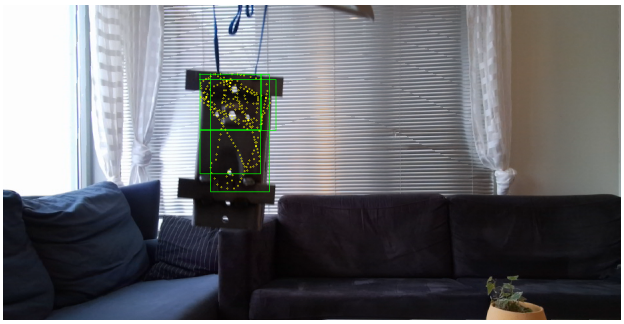


(a) (b)

(c)

Figure 43: Example showcasing a successful tracking initialization. The clustered translations from **c)** give rise to all pose candidates evaluated by the pose detector. Just two tracking iterations after correctly picking out the pose candidate from **a)**, the refined pose estimate from **b)** is obtained.

(a) (b)



(c)

Figure 44: Another example showcasing a successful tracking initialization. The clustered translations from **c)** give rise to all pose candidates evaluated by the pose detector. Just two tracking iterations after correctly picking out the pose candidate from **a)**, the refined pose estimate from **b)** is obtained.

As discussed in 3.1, the quality of the attained LineMOD matches can be rather inaccurate. The approach for calculating match translations in 5.1.2 do however provide decent translation coordinates for each detection. Consequently, seemingly inadequate LineMOD matches can still give rise to successful object initializations for the tracker. This is also clearly demonstrated in figure 43 and 44. While the clustering result in 43c and 44c show significant deviation from the true object, the correct pose is still found by the implemented pose detector in 43a and 44a. In just two tracking iterations, the refined pose estimates from 43b and 44b are obtained.

7.3 Drift Detection and Correction

In this subsection the motivation for a drift detecting and correcting solution is first given in 7.3.1. After presenting the implementation in 7.3.2, an evaluation of this solution is provided in 7.3.3.

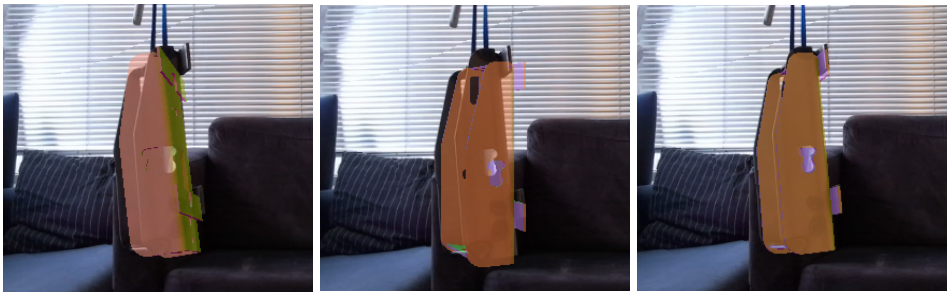
7.3.1 Motivation

Even though the implemented pipeline for object detection and pose estimation has showed some very good results, tracking errors can still occur. Whether this is caused by an incorrect initial object pose detection, or by a challenging rotation during tracking, it would naturally be preferable if the tracker was able to recover the true object pose. Consequently, the idea of an additional mechanism emerged, capable of detecting and correcting faulty pose estimates, i.e. *drifting*. In the event of a tracking error, the rotation of the pose estimate is almost exclusively at fault. While the translation estimate tend to always follow some part of the object, imprecise appearance models and pose ambiguity, along with high-speed rotations and missing depth data can result in deviating rotation estimates. Accordingly, the implemented solution will only focus on finding the correct rotation of the associated object, not its translation. If having to recover from a faulty translation, a reinitialization using the approach described in 7.2.2, including LineMOD matching and clustering would be required. For correcting rotational drift however, a faster approach is preferable. By using the current translation estimate as the true translation candidate, the task of drift correction is reduced to finding the rotation candidate which best fit the most recent RGB-D images.

7.3.2 Solution

In order to detect faulty rotation and pose estimates, the summarised average variances from 7.2.2 are utilized. Just as for the initial pose candidates, this score can express the uncertainty for a current pose estimate. For instance, similar low variance color and depth based likelihoods will result in low scores, indicating accurate pose estimates. On the other hand, inaccurate appearance models and conflicting contour and edge probabilities will give rise to higher variance scores, indicating less accurate pose estimates. By applying an upper threshold for the summarised average variance t_{upper}, tracking iterations with less accurate pose estimates are easily recognised. If the variance score remains too high for $n > t_{drift}$ consecutive tracking iterations, the drift correction is initialized. The best rotation candidate is then found by performing a single tracking cycle for each candidate, just as in 7.2.2. The respective object pose estimate is always updated according to the lowest scoring rotation candidate. Some additional n_{reset} iterations are also added to t_{drift}, allowing the pose estimate to stabilize after performing a drift correction. An example illustrating the result of this solution is presented in figure 45.

Several values for the upper variance threshold were tested. Naturally it is beneficial to keep the threshold low in order to detect drifting as fast as possible. While some pose estimates clearly deviate from the true object pose, the variance scores do not always match the degree of inaccuracy for these estimates. This is particularly the case when provided few trustworthy depth data segments, and when exposed to contour ambiguity.



(a) (b) (c)

Figure 45: Three consecutive tracking iterations illustrating the solution for drift detection and correction. Note that the utilization of depth data was disabled for this tracking run in order to induce more incorrect pose estimates. **a)** The tracking iteration in which the drift is finally detected. **b)** The lowest scoring rotation candidate is chosen as the updated pose estimate. **c)** The refined pose estimate after an additional tracking cycle.

The lack of conflicting color and depth based likelihoods, along with barely noticeable contour mismatches, keep the variance fairly low. Some examples of deviating pose estimates with low variance scores are shown in figure 46.

Figure 46: Examples of inaccurate pose estimates with low variance scores due to contour ambiguity and a lack of trustworthy depth based likelihoods.

Although reducing the variance threshold could enable the tracker to detect these inaccuracies, this would not be good for the performance of the tracker during challenging rotations. Due to high-speed rotations and imprecise color models, accurate pose estimates can sometimes give rise to higher variance scores. If trying to detect all inaccurate pose estimates from figure 46, this would cause the tracker to initialize numerous unnecessary drift corrections, instead of just keep tracking the object. In order to avoid this problem, an additional *drift property* was added to the total average variance score. For variance scores above a given threshold t_{lower}, which includes the examples of figure 46, the drift property is calculated. This value is also depending on the change in translation and rotation summarised from all 7 pose correcting iterations during the tracking iteration. The total variance score V_T is calculated as follows:

$$V_T = \begin{cases} V_{SAV} + 2 \times \frac{\left(V_{SAV} - (t_{lower} - 2)\right)}{100 \times \|\boldsymbol{\theta}\|_2 + 1} & \text{if } V_{SAV} > t_{lower} \\ V_{SAV} & \text{if } V_{SAV} \leq t_{lower} \end{cases}, \tag{59}$$

where V_{SAV} is the summarised average variance, and $\boldsymbol{\theta}$ is the summarised full variation vector. The additional drift property justifies challenging rotations to a greater extent, as increasing pose variation reduce the total variance score. If the tracker on the other hand is stuck with an inaccurate pose estimate for a stationary object, a bigger drift property should enable the tracker to detect these inaccuracies.

7.3.3 Evaluation

All in all, the solution for drift detection and correction seems to be a valuable extension to the implemented tracker. When running the RBGT [48] without depth utilization, the advantages are especially evident. As the lack of depth data in general result in more deviations for the pose estimates, a frequent reevaluation of the current rotation definitely increase the overall accuracy of the tracker. The modified tracker with depth utilization also appears to benefit from the drift handling. However, it can be challenging to find appropriate variance thresholds for the drift detection. Although the additional drift property from equation 59 can help in cases such as those illustrated in figure 46, it can still be difficult to avoid unnecessary drift corrections. Due to the additional depth based likelihoods, the modified tracker is able to maintain accurate pose estimates despite having imprecise appearance models. Hence, the high variance of the color based contour likelihoods can cause the initialization of drift corrections while still having satisfactory pose estimates. For these cases, the subsequent pose estimates are often worse as the new appearance models and pose estimates need to be refined. Other approaches for drift detection were also tested. This includes an assessment of the total number of approved depth segments, indicating the presence of an edge along the respective correspondence lines. For this case in particular, contour ambiguity made it difficult to detect incorrect pose estimates. Consequently, the variance based detection approach from equation 59 is still utilized despite showing some weaknesses when enabling depth data. Further testing in the lab setup could however improve the drift detection by fine-tuning the variance score calculation, and the drift thresholds.

8 Experiments

In this section the complete solution for object recognition and motion tracking is tested in the intended lab setup from section 1. Using the camera mounted on the robotic manipulator, results from the object detection solution are first displayed in 8.1, before results from the modified tracker are presented in 8.2.

8.1 Detection Results

In order to give a reasonable assessment of the object detection approach, the solution was tested while having two exemplars of the big chair part attached on the hanger. A representative selection of the attained results are presented in figure 47 and 48. As the detection part of the pipeline only concerns the initial pose estimates, only the clustered LineMOD matches and the final pose candidates are displayed. Consequently no refined posed estimates from the tracker are included.

As illustrated in the aforementioned figures, the implemented solution delivers some fairly good results for the object detection. For the three test runs showcased in figure 47, both target objects attain a fine amount of LineMOD matches. Accordingly, the clustering procedure has no problem finding the $n_b = 3$ best translation candidates. The final pose candidates also demonstrate this, as the translation of the rendered overlays clearly coincide with the true object translations. For the pose detections in figure 47b and 47d, the rotations are also precisely determined by the variance-based approach from 7.2. Using these initial pose estimates, the tracker can smoothly start tracking the detected objects. For the pose detection presented in figure 47f, this is not necessarily the case. While the translation of the detections seem rather accurate, only one of the objects attain a decent rotation estimate. The rotation mismatch for the topmost object is most likely caused by contour ambiguity, as the true object contour clearly resembles the contour of the estimated object pose. As this initial pose estimate deviates quite a bit from the true object, a drift correction would be necessary for achieving an accurate pose estimate. However, the precise translation estimate would prevent tracking loss, as the object would simply be tracked with a faulty rotation estimate.

The drift correction could also help in the pose detection case displayed in figure 48b. While being provided with two very promising translation candidates from the clustered LineMOD matches in figure 48a, the rotation is misinterpreted for the leftmost object. The fact that both objects are recognised by the detector is nevertheless very positive as it enables the tracker to initialize with both objects included. Figure 48d and 48f both display a different scenario. Despite having clustered LineMOD matches on both target objects in figure 48c, only one of the objects is passed on to the tracker, as illustrated in figure 48d. Due to a slightly inaccurate clustered translation for the leftmost object, non of the associated rotation candidates obtain a sufficiently low variance score, causing the detector to discard the affiliated object. While the detection displayed in figure 48f is also missing the leftmost object, this exclusion is caused by a lack of clustered LineMOD matches on the target object. With few to none true matches for an object, the object is not likely to be included in the following tracking. Overall, this seem to

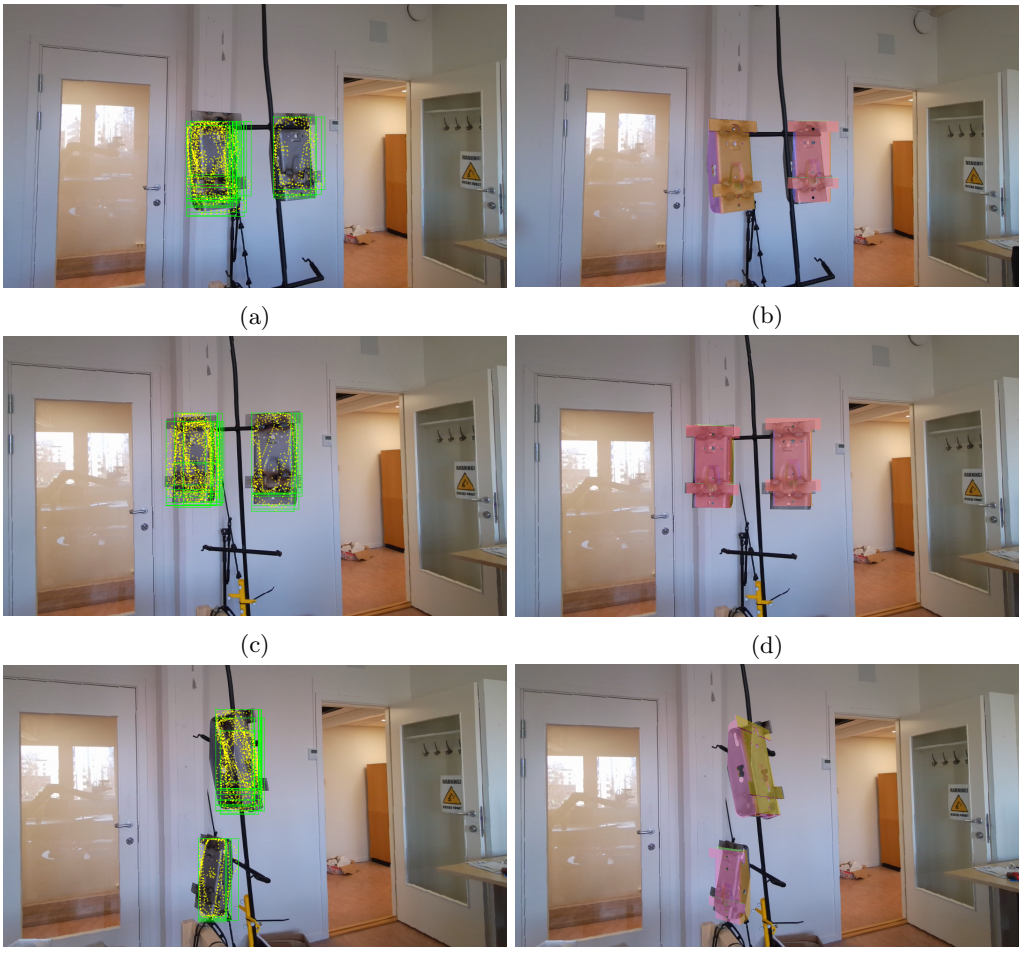


Figure 47: Detection results obtained from the implemented object detection solution. **Left:** The clustered LineMOD matches giving rise to the n_b best translation candidates. **Right:** The associated final pose candidates being passed on to the tracker.

be the biggest issue for the implemented detection solution. Although this solution do not demand very accurate initial pose estimates from the LineMOD detector, a lack of matches can still cause problems. In an effort to improve the selection of object translation candidates, lower matching thresholds and an increased number of clusters were tested without appreciable results.

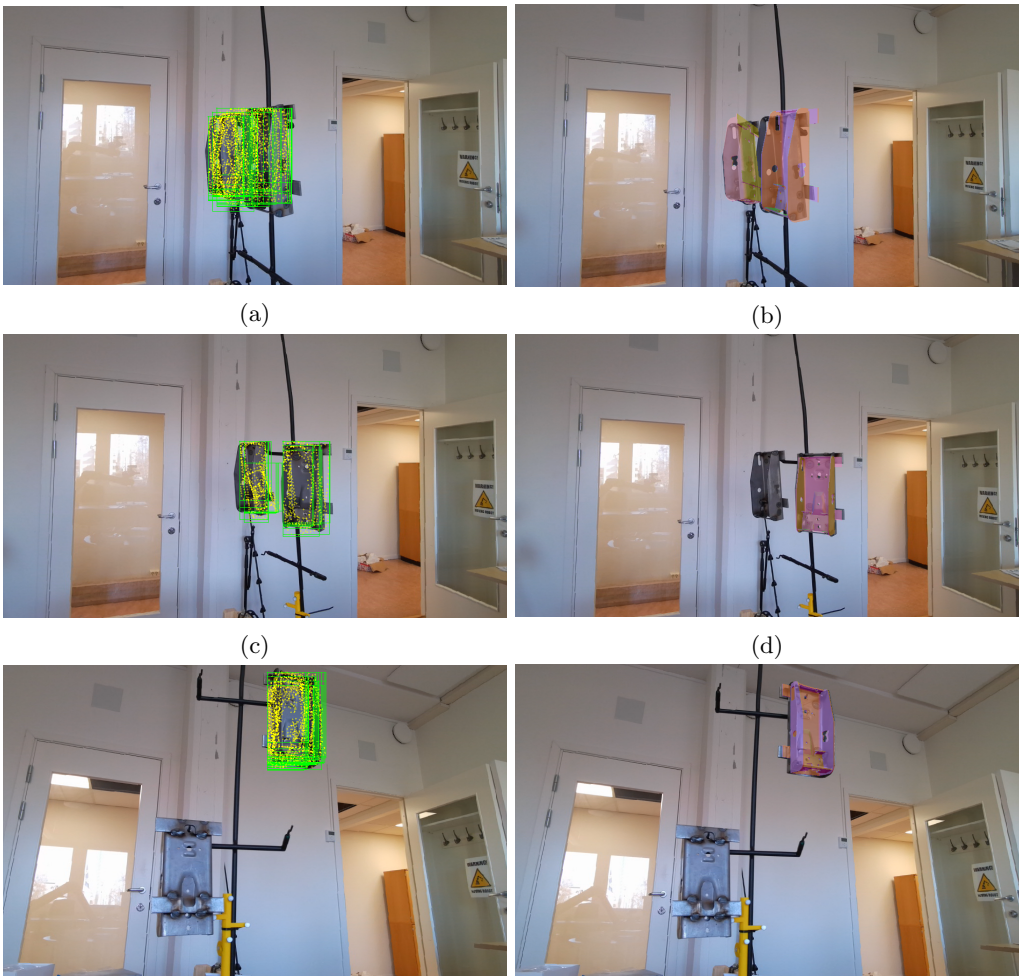


Figure 48: Detection results obtained from the implemented object detection solution. **Left:** The clustered LineMOD matches giving rise to the n_b best translation candidates. **Right:** The associated final pose candidates being passed on to the tracker.

8.2 Tracking Results

In this subsection the modified tracker with depth data utilization is first evaluated in 8.2.1. Next, the solution for drift detection and correction is put to the test in 8.2.2. Finally, some multi object tracking results are briefly presented in 8.2.3.

8.2.1 Depth Data Utilization

For evaluating the modified tracker, a prerecorded image sequence from the lab is utilized. This image sequence shows one object, the big chair part, being exposed to substantial changes in rotation while attached on the hanger. When using the modified tracker, the Euler angle estimates from figure 50 are obtained. For better readability, the values are adjusted to exceed the defined limits of $-\pi < ex \leq \pi$, with ex being the Euler angle estimate for the x-axis in the object reference frame. When attached to the hanger, this axis is pointing upwards throughout the object. As discussed previously, this axis will exhibit most of the changes in object rotation for this hanging scenario. The remaining Euler angle axes are defined as shown in figure 49.

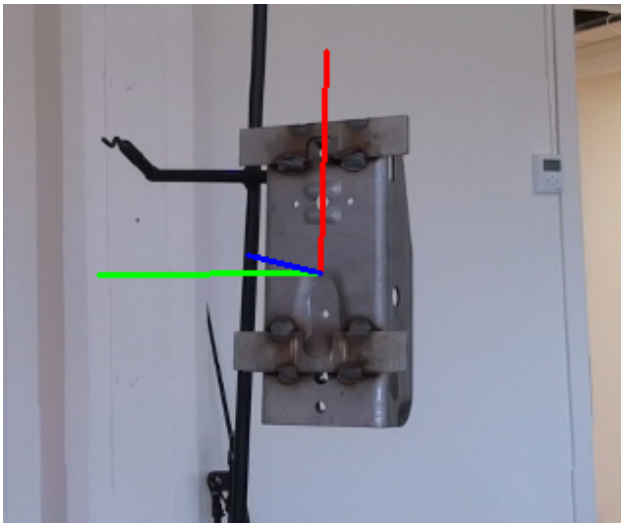


Figure 49: Illustration of the object reference frame axes. The x-axis is displayed in red, while the y-axis and z-axis are shown as green and blue lines respectively.

Using the original RBGT [48] on the same sequence of images, the Euler angles from figure 51 are attained. In order to better compare the two trackers, no drift correction is utilized.

While some variations between the trackers can be observed for both ey and ez, the estimates for ex clearly differentiate the presented tracking runs. The differences for these estimates are also demonstrated in figure 52 by directly comparing the ex value from both tracking runs. As illustrated, the two estimates are more or less identical during the first half of the tracking run. However, after approximately 155 tracking iterations, the estimate from the color-only RBGT starts to deviate from the estimate provided by the modified tracker. After producing some opposing Euler angle estimates, the respective trackers once again obtain some very similar estimates after approximately 220 tracking iterations. This continues for roughly 50 iterations, before another series of mismatching ex estimates are produced for the remaining tracking run.

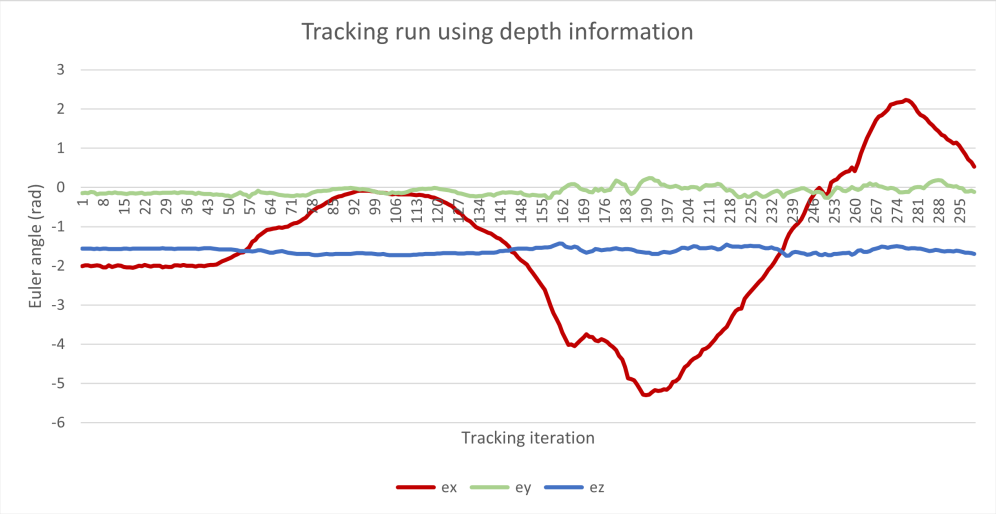


Figure 50: Euler angle estimates for the modified RBGT.

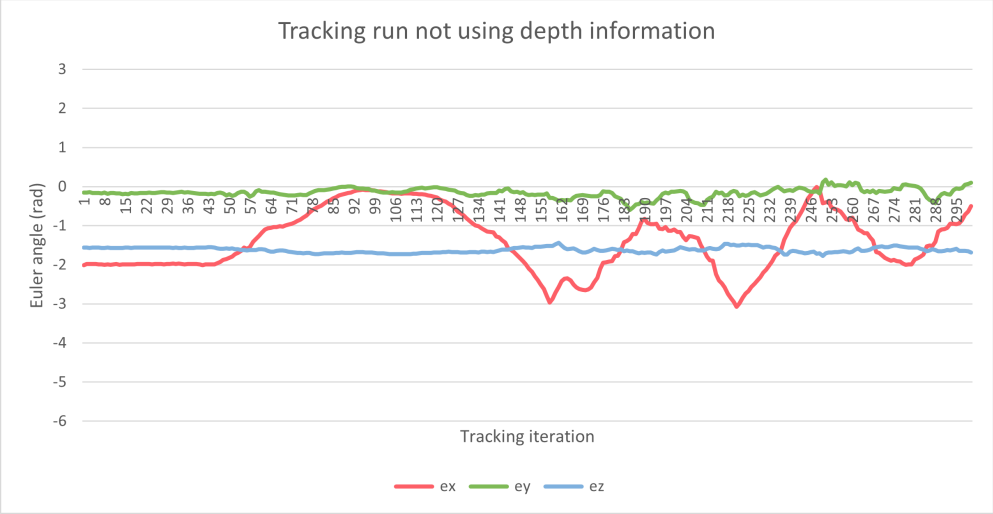


Figure 51: Euler angle estimates for the original RBGT.

By making use of the pose estimate visualization for each of the presented tracking runs, information on the actual tracking performance is also provided. Accordingly, samples from both tracking runs are showcased in figure 53. The rendered object overlays show that the modified tracker with depth data utilization clearly outperforms the color-only RBGT during the intervals with mismatching pose estimates. The tracking samples from figure 53 also suggest that the deviating pose estimates are caused by con-

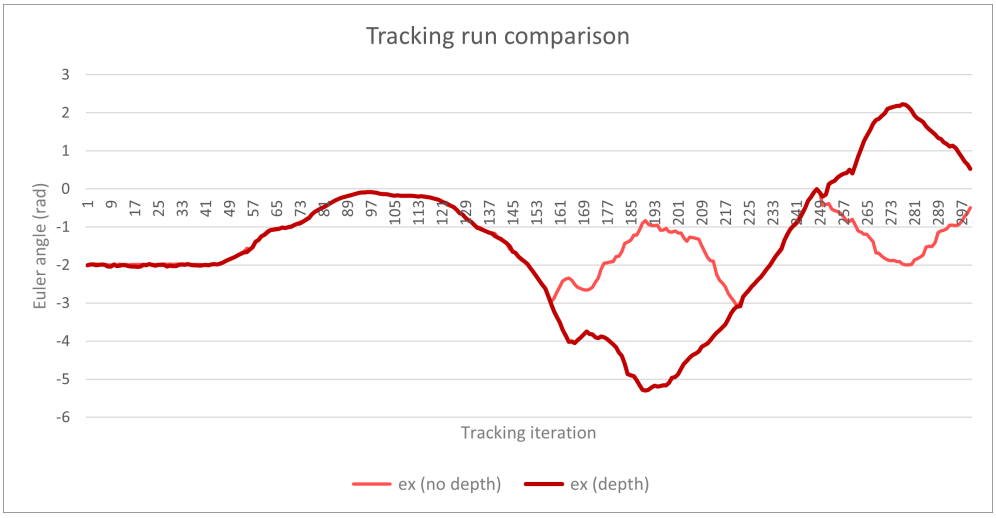


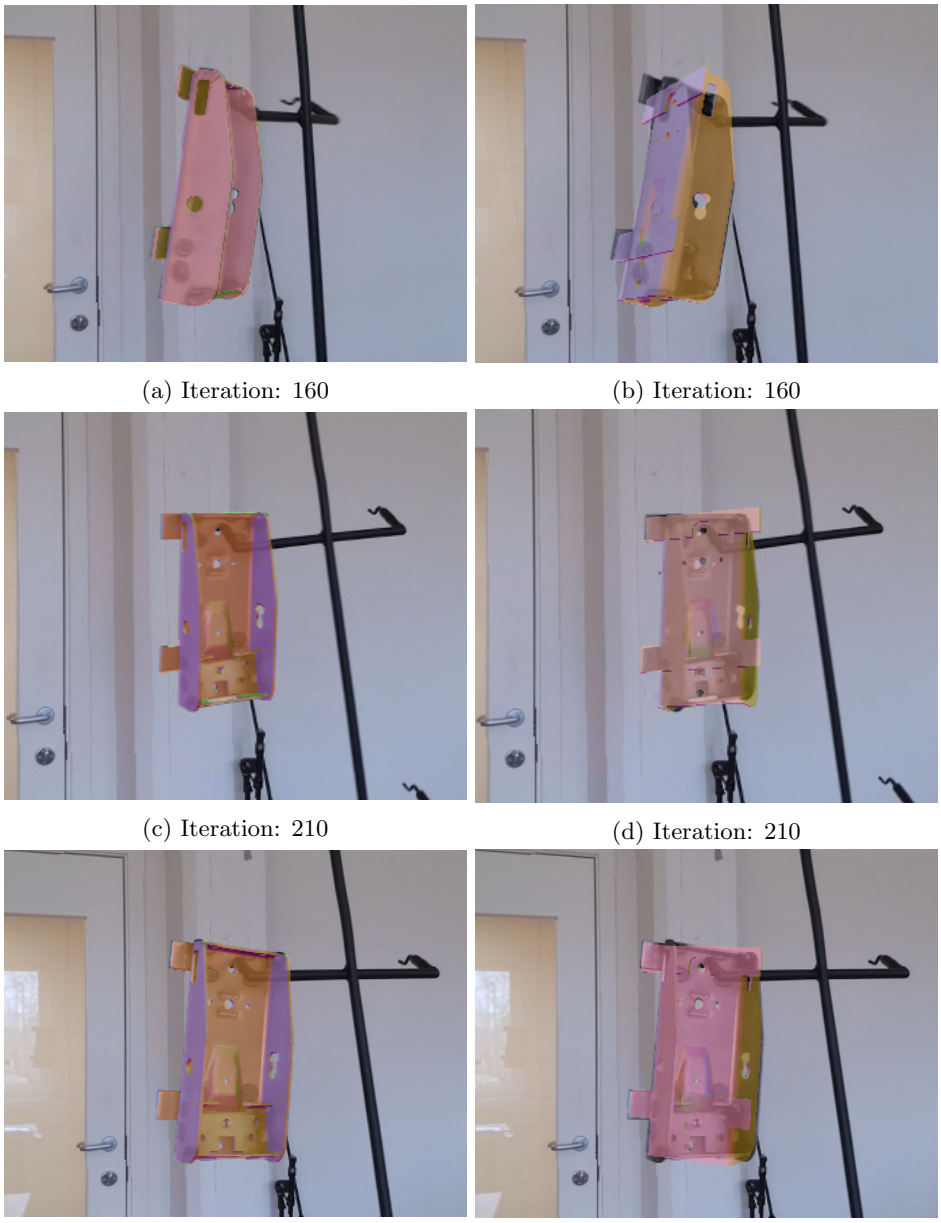
Figure 52: Side by side comparison of the original and the modified RBGT.

tour ambiguity, which is why the opposite ex estimates arise for $ex \approx 0$ and $ex \approx -\pi$. Both tracking runs, with and without depth data utilization, can be found in the supplementary material as video number 1 and 2 respectively. Despite showing some minor deviations during the tracking run, the modified tracking solution arguably demonstrate some very impressive results.

8.2.2 Drift Correction

For a different sequence of images, the tracking results are not as impressive for the modified tracker. As the aforementioned tracking case, this sequence of images also displays the big chair part attached to a rotating hanger. In contrast to the already presented tracking runs however, the beam of the hanger, along with contour ambiguity, causes the modified RBGT to track the object with incorrect rotation estimates. Despite producing some vastly inaccurate angle estimates for multiple object axes, the translation estimates seem to remain fairly accurate. The proposed solution for drift detection and correction can thus be utilized. The rather descriptive ex estimates for this tracking case are presented in figure 54, comparing the tracking performance with and without drift correction. Just as for the previous tracking runs, the values of ex have been adjusted to exceed the defined limits of $-\pi < ex \leq \pi$.

Note that due to lab access difficulties, no ground truth was attained for any of the presented tracking runs. Accordingly, the dotted grey lines in figure 54 only showcase an estimated correct rotation when the tracking estimates seem to deviate from the true object pose. Anyhow, judging by the rendered overlays, the initial pose detection clearly succeeds in picking out the correct translation and rotation of the object. However, the ex estimates reveal how the rotation starts to deviate from the presumed true object



(a) Iteration: 160 (b) Iteration: 160

(c) Iteration: 210 (d) Iteration: 210

(e) Iteration: 275 (f) Iteration: 275

Figure 53: Tracking samples from the modified and the original RBGT during the intervals with mismatching pose estimates. **Left:** Pose estimates obtained by the modified RBGT. **Right:** Pose estimates obtained by the original RBGT.

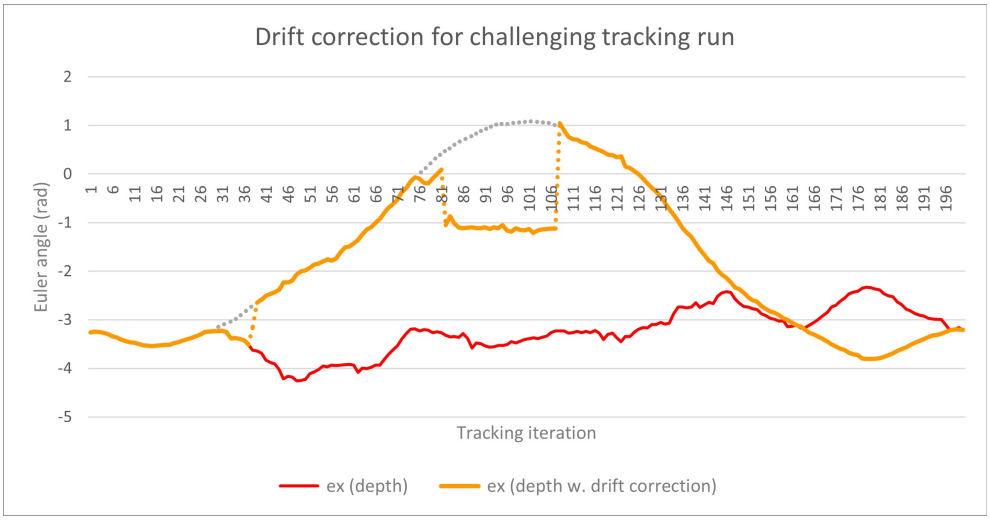


Figure 54: Side by side comparison of the modified RBGT with and without drift correction. The dotted orange lines illustrate drift corrections while the dotted grey lines represent the presumed true rotation estimates.

pose after approximately 30 tracking iterations. From this point, the tracker with no drift correction struggles to recover the correct rotation estimate. In contrast, the tracker with drift correction overall demonstrate a clearly improved tracking result. When the Euler angle estimates start to drift after approximately 30 iterations, this is quickly detected and corrected using the approach from 7.3. Having retrieved the correct rotation, the tracker can continue to produce accurate pose estimates. For the next challenging rotation, with $ex \approx 0$ and the hanger itself misleading the contour likelihoods, the rotation estimates start to diverge once again. The drift detector immediately sees this and initialize another drift correction. However, this time contour ambiguity leads the tracker to select an incorrect rotation candidate. As illustrated in figure 54, the object is thereafter tracked with an incorrect rotation estimate for approximately 25 tracking iterations, or 1.0 seconds, before yet another drift correction successfully recovers the true object pose. For the remaining tracking run the object pose estimates appear to be very precise. Although the drift correction fails to recover the correct rotation candidate after roughly 80 tracking iterations, the tracker with drift correction still clearly outperforms the tracker with no drift correction. Both tracking runs presented in figure 54 can be found as video number 3 and 4 in the supplementary material.

8.2.3 Multiple Objects

In addition to the aforementioned single object tracking runs, the supplementary material also include a fifth tracking run with both the big and the small chair part, illustrating the methods ability to track multiple different objects simultaneously. A sample of this tracking run is shown in figure 55. Note that the *modeled occlusion* functionality was

not enabled during this sequence. As both objects have more or less identical foreground models, the *unmodeled occlusion* is not able to disregard those correspondence lines occluded by the other object. Nevertheless, the tracker still produces some very impressive pose estimates throughout the tracking run.

Figure 55

9 Discussion

In this section the main challenges of the implemented detection and tracking solutions are first discussed in 9.1 and 9.2 respectively, before the suitability of the complete framework is discussed in 9.3.

9.1 Detection Challenges

Starting with the LineMOD detector, the occasional lack of positive template matches is an obvious obstacle for the detection solution. As discussed in the preliminary project thesis [29], there is no guarantee that all objects are detected. The presented result in 8.1 also verify this. If provided close to none LineMOD matches for an object, these matches are not likely to survive the clustering. This is especially the case if there is a high total number of matches in the scene. Having a simple and uniform background seems to improve the clustered result by reducing the number of false matches and making the object contour more explicit. However, the quality of the matches is still far from perfect. Naturally, lowering the matching threshold and increasing the number of clusters can be helpful in some cases by providing more translation candidates for the rotation deciding pose detection. Having said that, the quality of the LineMOD detector still appears to be an underlying issue for the implemented detection solution. As discussed in the preliminary project thesis, the shape of the object, i.e. the big chair part, might not be ideal

for this type of detector due to the general lack of easily recognizable curved surfaces on the object. Luckily, the detector is still capable of attaining LineMOD matches with fairly precise image coordinates for most target object. As the subsequent pose detection only requests the translation of these objects, no further information is required from the LineMOD detector to initialize the tracking of these objects. This way, the implemented detection approach makes up for some of the limitations of the LineMOD detector.

As discussed previously, the rotation deciding pose detection works fairly good considering the quality of most LineMOD matches. However, pose ambiguity and imprecise translation candidates can still cause the detector to fail, by either picking out an incorrect rotation candidate, or by failing to find a suitable low scoring candidate all together. The latter does not appear to occur very often. If provided an approximate object translation, this object is typically always accepted by the pose detection step. However, as shown in 8.1, the estimated rotation can sometimes be incorrect. Although a faulty initial estimate is likely to be corrected later, it would still be beneficial to start off the tracking with the correct rotations. Another minor drawback of the implemented detection approach is the need for a rotation assumption. Just as for the drift correction, the initial pose detection also depends on this assumption in order to control the number of rotation candidates. If considering angular variation along all reference frame axes, while keeping a decent rotation resolution, the computational cost would be far too great for a real-time application. For the presented object recognition and pose estimating case, the utilized rotation assumption has been rather unproblematic. For other use-cases however, corresponding restrictions might not be applicable.

9.2 Tracking Challenges

As described in 2.3, contour ambiguity was thought to be the biggest challenge for the RBGT [48]. Despite showing some excellent results for a great number of tracking runs, contour ambiguity can still cause problems for the implemented tracker solution. Although the adding of depth based edge likelihoods undoubtedly helped out, the shape of the big chair part, along with missing depth data continue to complicate the pose estimation in some cases when exposed to contour ambiguity. As illustrated in 7.1.2, a significant portion of all depth data segments are rejected for having too many invalid measurements. If provided higher quality depth images with less invalid measurements, the additional edge likelihoods would surely improve the overall performance of the tracker. A potential fix could be to add a second camera for noise removal. If not, the utilization of a depth image filter might also give some viable results. Another approach for avoiding incorrect pose estimates when facing pose ambiguity could be to incorporate motion prediction to the RBGT. By making use of prior information on angular velocities, multiple cases of drifting rotation estimates could probably be averted.

In addition to contour ambiguity, the hanger beam also seems to cause some occasional difficulties for the tracker. When close to the object, the color based likelihoods can sometimes confuse the hanger with the object contour. Furthermore, the depth based likelihoods can be steered away from the true contour if the beam pass in front of the object. For this to happen however, the hanger would have to be rotated all the way

around. Under normal circumstances this should not be an issue. Moreover, the utilization of depth data appears to counterbalance the color based beam confusion. Although the drift correction in a way works as a safety net for the aforementioned tracking challenges, the drift correction also has some challenges. As presented in 8.2.2, the approach does not always find the correct object rotation. In addition, it can be hard to find a suitable drifting measure and threshold providing rapid drift detections for difficult cases with contour ambiguity, without initializing excessive corrections. Having a simpler background definitively helped by provoking less false drift detections. Nevertheless, the drift property from equation 59 is still unable to expose the most challenging cases of contour ambiguity. Just as for the general performance of the tracker, the detection of drifting would probably also benefit from having more reliable depth data. Otherwise, a periodically checkup could help for cases where a stationary object is being tracked with an incorrect rotation.

9.3 Suitability of Implemented Framework

All things considered, the implemented object recognition and motion tracking system has shown some great results for both detection and tracking. Despite having to deal with some rather imprecise LineMOD matches, the detector solution is still able to find and initialize most target objects by applying translation clustering and testing different rotation candidates. Furthermore, the system almost never loses track. By maintaining an adequate translation estimate, a faulty rotation estimate can normally be corrected without difficulties. The adding of depth based likelihoods to the RBGT has also improved the general performance of the tracker. Under normal circumstances for the intended use case, without substantial changes in rotation, contour ambiguity would also be far less of a problem than what has been the case for the experiments in section 8. For differently shaped objects, such as the small chair part, the problems with contour ambiguity can even be avoided entirely.

The solution's efficiency is also a great advantage. While the object detection and tracker initialization can be completed in less than 0.5 seconds, tracking frequencies above $30Hz$ can be achieved even when tracking multiple object. As the setup only requires a 3D model and a rotation assumption for detection and tracking, the solution moreover makes a very flexible framework. For the presented manufacturing setting, a new chair part can accordingly be added to the production line without having to make any customised changes. Although the demand for a rotation assumption might seem inhibitory, this has showed to be a convenient solution when dealing with the roof mounted hanger. Naturally, these assumptions can also be changed to deal with different use cases. Another essential quality of the implemented framework is that it requires no prior information regarding texture or color. By using the object shape as the only basis for the initial detection, no color information is utilized before the initialization of the RBGT color histograms. This way, objects with varying surface texture should obtain the same level of performance for both detection and motion tracking. The online learning of appearance models also adds to this by continuously updating the color histograms as the visible parts of the objects changes. Naturally, the depth based edge likelihoods can provide further robustness in these cases by precisely picking out the location of the object contour. This combination

of online learning and depth data utilization can be particularly advantageous for tracking objects while for instance undergoing a paint job.

10 Concluding Remarks

In this work, a feasible solution for both object recognition and motion tracking is proposed. This solution is primarily based on the combination of the multi-modality LineMOD detector and RBGT, a novel region-based 6-DoF object tracker. Starting off with these, several additions have been introduced in order to improve the overall performance of both the object detection and the tracker. These include translation clustering for the attained LineMOD matches, along with a subsequent rotation candidate evaluation scheme used for finding the most promising initial object poses. Furthermore, the RBGT has been modified to utilize depth information using a very sparse, yet efficient approach. Along with a presented solution for drift detection and correction, the use of depth data has unquestionably added precision and robustness to the original RBGT. All in all, the total system has proven itself to be a good fit for the intended recognition and motion tracking problem.

In order to further improve the implemented solution, it would be interesting to see if additions such as motion prediction could increase precision when exposed to difficult cases of contour ambiguity. In addition, future work could also include experimenting with depth image filtering. If provided higher quality depth data, with less invalid measurements, their corresponding edge likelihoods would probably boost the tracker's general performance. Finally, it might also be interesting to check out different object detectors. Although the LineMOD detector is capable of detecting most objects under the right circumstances, it would still be preferable if the detector could correctly estimate their rotation. If not, a 2D detector might also replace it, as the current implementation only requires the image coordinates for the associated matches.

References

[1] Alexander Alekhin, Vadim Pisarevsky, Pavel Rojtberg, and Rostislav Vasilikhin. linemod. *Open Source Computer Vision Library (OpenCV), Github repository: opencv/opencv_contrib/modules/rgbd, https://github.com/opencv/opencv_contrib/blob/master/modules/rgbd/src/linemod.cpp*, 2019.

[2] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *In the IEEE Transactions on visualization and computer graphics*, volume 9:pp. 3–15, 2003.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *In the proceedings of European Conference on Computer Vision (ECCV)*, pages 404–417, 2006.

[4] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. *Sensor fusion IV: control paradigms and data structures, International Society for Optics and Photonics*, volume 1611:pp. 586–606.

[5] Charles Bibby and Ian Reid. Robust real-time visual tracking using pixel-wise posteriors. In *the European Conference on Computer Vision*, pages 831–844. Springer, 2008.

[6] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold, and C. Rother. Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image. *In the proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3364–3372, 2016.

[7] Chire. Kmeans-gaussian-data.svg. *https://en.wikipedia.org/wiki/Cluster_analysis/media/File:KMeans-Gaussian-data.svg*, 2011.

[8] Samuel Dambreville, Romeil Sandhu, Anthony Yezzi, and Allen Tannenbaum. Robust 3d pose estimation and efficient 2d region-based segmentation from a 3d shape prior. In *European Conference on Computer Vision*, pages pp. 169–182. Springer, 2008.

[9] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3d object recognition. *In the IEEE computer society conference on computer vision and pattern recognition*, pages 998–1005, 2010.

[10] Martin A Fischler and Robert C Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *In the Communications of the ACM*, volume 24(6):pp. 381–395, 1981.

[11] Ge Gao, Mikko Lauri, Yulong Wang, Xiaolin Hu, Jianwei Zhang, and Simone Frintrop. 6d object pose regression via supervised learning on point clouds. *In the proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3643–3649, 2020.

[12] Iryna Gordon and David G. Lowe. What and where: 3d object recognition with accurate pose. *Toward category-level object recognition*, pages 67—-82, Springer, 2006.

[13] Gina Gould. How workplace robotics improve safety and health. *Enablon, Wolters Kluwer, https://enablon.com/blog/the-impact-of-robotics-on-safety-and-health/, 13.10.2020*, 2019.

[14] Carlton Gyles. Robots in medicine. *The Canadian Veterinary Journal*, volume 60(8):pp. 819, Canadian Veterinary Medical Association, 2019.

[15] Ruotao He, Juan Rojas, and Yisheng Guan. A 3d object detection and pose estimation pipeline using rgb-d images. *In the proceedings of European Conference on Computer Vision (ECCV)*, 2017.

[16] Jonathan Hexner and Rami R Hagege. 2d-3d pose estimation of heterogeneous objects using a region based approach. *In the International Journal of Computer Vision*, volume 118(1):pp. 95–112, Springer, 2016.

[17] Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. Gradient response maps for real-time detection of texture-less objects. *In the IEEE transactions on pattern analysis and machine intelligence*, volume 34(5):pp. 876–888, 2011.

[18] Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. Gradient response maps for real-time detection of textureless objects. *In the IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, volume 34:pp. 876—-888, 2012.

[19] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. *In the proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 858–865, 2011.

[20] Tomas Hodan, Xenophon Zabulis, Manolis Lourakis, Stepan Obdrzalek, and Jiri Matas. Detection and fine 3d pose estimation of texture-less objects in rgb-d images. *In the proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4421–4428, 2015.

[21] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, volume 32:pp. 922–923, 1976.

[22] Wadim Kehl, Federico Tombari, Slobodan Ilic, and Nassir Navab. Real-time 3d model tracking in color and depth on a single cpu core. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 745–753, 2017.

[23] Wadim Kehl, Federico Tombari, Nassir Navab, Slobodan Ilic, and Vincent Lepetit. Hashmod: A hashing method for scalable 3d object detection. *In the proceedings of the British Machine Vision Conference (BMVC)*, 2015.

[24] Shawn Lankton and Allen Tannenbaum. Localizing region-based active contours. *In the IEEE transactions on image processing*, volume 17(11):pp. 2029–2039, 2008.

[25] Manolis I. A. Lourakis. A brief description of the levenberg-marquardt algorithm implemented by levmar. *Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH), http://users.ics.forth.gr/lourakis/levmar/levmar.pdf*, 2005.

[26] David G. Lowe. Distinctive image features from scale-invariant keypoints. *In the International Journal of Computer Vision*, volume 60:pp.91–110, 2004.

[27] David MacKay. An example inference task: Clustering. *Information theory, inference and learning algorithms*, volume 20:pp. 284–292, 2003.

[28] David MacKay. Clustering methods. *Data mining and knowledge discovery handbook*, pages 321–352, 2005.

[29] Matias Hagen Myrestrand. Recognition and motion tracking of 3d objects. *Department of Engineering Cybernetics at the Norwegian University of Science and Technology, TTK4550*, 2020.

[30] The International Federation of Robotics Frankfurt. The impact of robots on productivity, employment and jobs. *https://ifr.org/img/office/IFR_The_Impact_of_Robots_on_Employment.pdf*, 2018.

[31] Maks Ovsjanikov. Ps 1 - rigid shape registration. *Informatics laboratory of l'École polytechnique (LIX), Ovsjanikov notes, http://www.lix.polytechnique.fr/∼maks/Verona_MPAM/TD/TD1/*, 21.10.2020.

[32] Anna Petrovskaya and Sebastian Thrun. Model based vehicle tracking for autonomous driving in urban environments. *In the proceedings of robotics: science and systems IV, Zurich, Switzerland*, volume 34, 2008.

[33] Nicolas Pinto, David D. Cox, and James J. DiCarlo. Why is real-world visual object recognition hard? *PLoS Comput Biol*, volume 4:pp. 151–156, 2008.

[34] Victor A Prisacariu and Ian D Reid. Pwp3d: Real-time segmentation and tracking of 3d objects. *In the International journal of computer vision*, volume 98(3):pp. 335–354, 2012.

[35] Victor Adrian Prisacariu, Olaf Kähler, David W Murray, and Ian D Reid. Real-time 3d tracking and reconstruction on mobile phones. *In the IEEE transactions on visualization and computer graphics*, volume 21(5):pp. 557–570, 2014.

[36] Neelam V. Puri and P. R. Devale. Human tracking and pose estimation in video surveillance system. *In the proceedings of International Journal of Computer Science Engineering and Information Technology Research (IJCSEITR)*, volume 3:pp. 257–274, 2013.

[37] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *In the proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[38] Vincent Rabaud, Carlos M. Mateo, Scott K. Logan, Jimmy Da Silva, Natalia Lyubova, and Lincoln Lorenz. ork_renderer. *Willow Garage - Object Recognition Kitchen, Github repository: wg-perception/ork_renderer, https://github.com/wg-perception/ork_renderer*, 2017.

[39] Carl Yuheng Ren, Victor Adrian Prisacariu, Olaf Kähler, Ian D Reid, and David William Murray. Real-time tracking of single and multiple objects from depth-colour imagery using 3d signed distance functions. *In the International Journal of Computer Vision*, volume 124(1):pp. 80–95, 2017.

[40] Jürgen Riegel, Werner Mayer, and Yorik van Havre. Freecad (software). 2015.

[41] R. Rios-Cabrera and T. Tuytelaars. Discriminatively trained templates for 3d object detection: A real time scalable approach. *In the proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2048–2055, 2013.

[42] Bodo Rosenhahn, Thomas Brox, and Joachim Weickert. Three-dimensional shape knowledge for joint image segmentation and pose tracking. *In the International Journal of Computer Vision*, volume 73(3):pp. 243–262, 2007.

[43] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. *In the proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2564–2571, 2011.

[44] Lloid S. Least squares quantization in pcm. *In the IEEE Transactions on Information Theory*, volume 28:pp. 129—-137, 1982.

[45] Christian Schmaltz, Bodo Rosenhahn, Thomas Brox, and Joachim Weickert. Region-based pose tracking with occlusions using 3d models. *Machine vision and applications*, volume 23(3):pp. 557–577, 2012.

[46] Mang Shao, Danhang Tang, and Tae-Kyun Kim. Real-time background-aware 3d textureless object pose estimation. *arXiv preprint arXiv:1907.09128*, 2019.

[47] Amit Shukla and Hamad Karki. Application of robotics in offshore oil and gas industry — a review part ii. *Robotics and Autonomous Systems*, volume 75:pp. 508–524, 2016.

[48] Manuel Stoiber, Martin Pfanne, Klaus H. Strobl, Rudolph Triebel, and Alin Albu-Schaeffer. A sparse gaussian approach to region-based 6dof object tracking. *In the proceedings of the Asian Conference on Computer Vision (ACCV)*, 2020.

[49] Manuel Stoiber, Martin Pfanne, Klaus H. Strobl, Rudolph Triebel, and Alin Albu-Schäffer. A sparse gaussian approach to region-based 6dof object tracking. *German Aerospace Center (DLR) - Institute of Robotics and Mechatronics (RM), Github repository: DLR-RM/RBGT, https://github.com/DLR-RM/RBGT*, 2020.

[50] Tetyana Sych et al. Azure kinect sensor sdk (software). *Microsoft*, 2019.

[51] David Joseph Tan, Nassir Navab, and Federico Tombari. Looking beyond the simple scenarios: Combining learners and optimizers in 3d temporal tracking. *In the IEEE transactions on visualization and computer graphics*, volume 23(11):pp. 2399–2409, 2017.

[52] Sych Tetyana, Phil Meadows, and Brent Allen. Azure kinect dk depth camera. *https://docs.microsoft.com/en-us/azure/kinect-dk/depth-camera, 8.02.2021*, 2019.

[53] Henning Tjaden, Ulrich Schwanecke, and Elmar Schömer. Real-time monocular segmentation and pose tracking of multiple objects. In *the European conference on computer vision*, pages 423–438. Springer, 2016.

[54] Henning Tjaden, Ulrich Schwanecke, and Elmar Schomer. Real-time monocular pose estimation of 3d objects using temporally consistent local color histograms. *In the proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 124–132, 2017.

[55] Henning Tjaden, Ulrich Schwanecke, Elmar Schömer, and Daniel Cremers. A region-based gauss-newton approach to real-time monocular multiple object tracking. *In the IEEE transactions on pattern analysis and machine intelligence*, volume 41(8):pp. 1797–1812, 2018.

[56] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. *In the proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1653–1660, 2014.

[57] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. *In the proceedings of the Conference on Robot Learning (CoRL), arXiv preprint arXiv:1809.10790*, 2018.

[58] Andrea Vattani. The hardness of k-means clustering in the plane. *University of California, San Diego, Vattani papers, https://cseweb.ucsd.edu/∼avattani/papers/kmeans_hardness.pdf*, 2010.

[59] Hongmin Wu. linemod_pose_estimation. *Biomimetics Robotics lab at Guangdong University of Technology, Github repository: birlrobotics, https://github.com/birlrobotics/linemod_pose_estimation*, 2018.

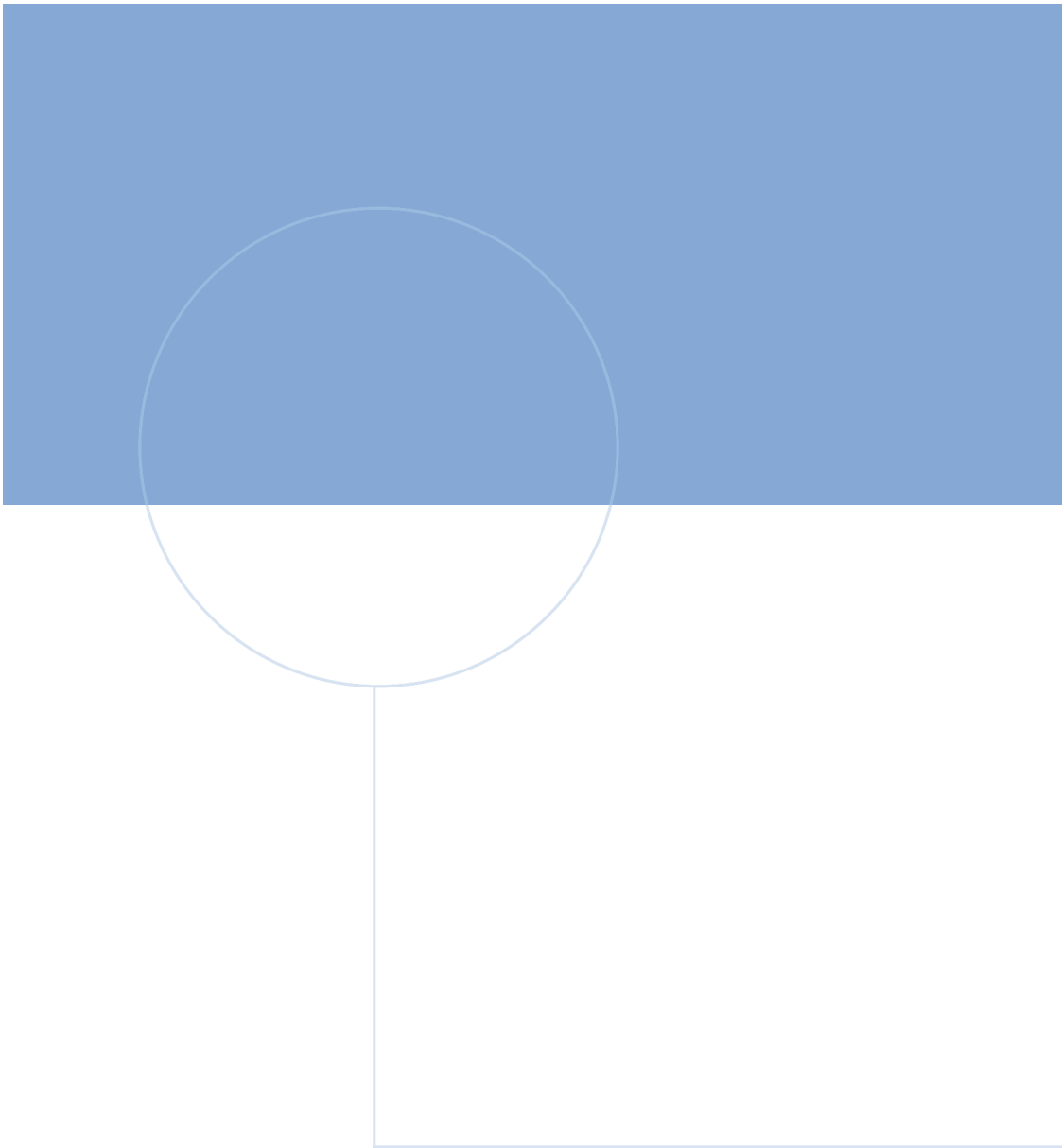
[60] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *In the proceedings of the conference: Robotics: Science and Systems, arXiv preprint arXiv:1711.00199*, 2017.

[61] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *In the International Journal of Computer Vision*, volume 13:pp. 119–152, 1994.

[62] Leisheng Zhong and Li Zhang. A robust monocular 3d object tracking method combining statistical and photometric constraints. *In the International Journal of Computer Vision*, volume 127:pp. 973–992, 2019.

[63] Leisheng Zhong, Xiaolin Zhao, Yu Zhang, Shunli Zhang, and Li Zhang. Occlusion-aware region-based 3d pose tracking of objects with temporally consistent polar-based local partitioning. *In the IEEE Transactions on Image Processing*, volume 29:pp. 5065–5078, 2020.

[64] Feng Zhou, Henry Been-Lirn Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. *In the IEEE/ACM International Symposium on Mixed and Augmented Reality*, volume 7:pp. 193–202, 2008.



NTNU
Norwegian University of
Science and Technology

SINTEF