6DCenterPose: Multi-object RGB-D 6D pose tracking with synthetic training data

6DCenterPose: Multi-object RGB-D 6D pose tracking mit synthetischen Trainingsdaten
Master thesis by Pascal Herrmann
Date of submission: April 23, 2023

1. Review: João Carvalho
2. Review: Jan Peters
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Pascal Herrmann, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 23. April 2023

P. Herrmann

Abstract

Tracking the 6D poses of objects is essential in robotics applications. Known poses can be used in behavioural cloning to teach the robot a desired policy, and they are crucial in manipulation tasks. Current methods either estimate the poses of all objects in a scene given a single image or track the pose of a single object in a sequence of images. However, to the best of our knowledge, there is currently no online method which tracks the poses of multiple objects in a sequence of images.

This thesis aims to close this research gap. To this end, a fully convolutional neural network, called 6DCenterPose, is proposed. Given the current RGB-D observation and a synthetic image showing the previous pose predictions, the method aims to predict the changes of poses to the current frame. A greedy matching algorithm based on sparse optical flow is used to associate the pose changes to the respective objects.

Disentanglement of the input feature encodings allows training 6DCenterPose exclusively on synthetic images. A synthetic data generation pipeline is proposed to generate photo-realistic and physically plausible images, which are used to train and evaluate the method. Experiments on real and synthetic videos of the board game *Ubongo3D* are conducted and show that the reality gap is successfully bridged. 6DCenterPose is online and runs at around 12.8 Hz, making it a valuable tool for multi-object 6D pose tracking tasks in robotic applications.

Zusammenfassung

Die 6D Posen von Objekten zu tracken ist essentiell für Robotik-Anwendungen. Sie können genutzt werden, um einem Roboter mittels Behavioural Cloning ein Verhalten beizubringen, oder Objekte mit einem Roboter zu bewegen. Aktuelle Methoden bestimmen entweder die Posen aller Objekte in einem einzelnen Bild oder tracken die Pose eines einzelnen Objekts in einer Reihe von Bildern. Nach unserem Kenntnisstand gibt es derzeit keine Methode, welche die Posen mehrerer Objekte in einer Reihe von Bildern online tracken.

Ziel dieser Arbeit ist es, diese Forschungslücke zu schließen. Es wird die Methode 6DCenterPose, ein fully convolutional Neuronales Netzwerk vorgestellt. Für ein RGB-D-Bild des aktuellen Zeitpuktes und ein synthetisches Bild, welches die Objektposen des vorherigen Zeitpunktes darstellt, wird die Veränderung der Posen zum aktuellen Zeitpunkt bestimmt. Ein greedy Zuordnungsalgorithmus, basierend auf sparse optical flow, wird genutzt, um die Posenänderungen den Objekten zuzuordnen.

Die Feature-Repräsentationen der Bilder sind entkoppelt, was es erlaubt, 6DCenterPose ausschließlich mit synthetischen Bildern zu trainieren. Hierfür wird eine Pipeline zur Erstellung fotorealistischer und physikalisch plausibler Bilder vorgestellt, welche zum Trainieren und Evaluieren der Methode genutzt werden.
Experimente mit echten und synthetischen Videos des Brettspiels *Ubongo3D* zeigen, dass die Realitätslücke erfolgreich überbrückt wird. 6DCenterPose ist online und erreicht eine Laufzeit von etwa 12.8 Hz, was es zu einem hilfreichen Werkzeug für multi-object 6D pose tracking Anwendungen in der Robotik macht.

Contents

1. Introduction

Access to the 6D poses of objects is essential in many applications like autonomous driving [1] and augmented reality [2]. In robotic applications, the 6D poses of objects can be used to teach the robot a policy via behavioural cloning [3, 4]. Additionally, accurately tracking the pose of objects is essential in manipulation tasks like pick-and-place and insertion. A straightforward approach to tracking the pose of the manipulated object is to assume a rigid connection between the robot's gripper and the object. Using forward kinematics, the pose of the object can be obtained. However, this assumption can fail in practical scenarios due to slippage of the object [5], or may not be precise enough due to inaccuracies in joint angle measurements [6]. Furthermore, this approach only allows obtaining the pose of the manipulated object. Especially in insertion or assembly tasks, it can be essential to know the poses of other objects in the scene, like the pose of the receptacle for insertion tasks or the pose of already placed objects in assembly tasks.

Vision-based approaches aim to accurately determine the 6D pose of objects in the scene from RGB(-D) images. These approaches can be divided into two lines of work. 6D pose estimation [7, 8, 9] determines the 6D poses of all objects in a scene given a single image. These methods are highly accurate, but they are also computationally expensive. Furthermore, given a sequence of images as common in robot manipulation, the poses are estimated from scratch for every frame without exploiting temporal information. This is redundant and can lead to less coherent pose estimates [10].
6D pose tracking [10, 11, 12], on the other hand, focuses on practical aspects like achieving higher framerates necessary for robot manipulation while focusing on accuracy in the local regime. Specifically, 6D pose tracking approaches usually operate on two adjacent frames and, given the object's pose in the former frame, predict the change of the pose to the latter frame. Iterating this procedure allows the propagation of an initial pose estimate over the whole sequence of images. While methods following this approach achieve impressive performances, they can suffer from the focus on the local regime. These methods can not recover from tracking loss and need to be reinitialised if the track is lost.

Additionally, current 6D pose tracking methods only focus on a single object and must be retrained when used for a different object [12].

Multi-object tracking [13] is a well-known problem in the computer vision community. Methods for this task can track multiple objects in a video, exploiting temporal information, but they primarily focus on tracking the 2D bounding boxes of the given objects [14, 15, 16]. While some of these approaches also track the 3D bounding boxes of objects [17, 18], this information is not enough for accurate robot manipulation. Approaches that track the full 6D poses of multiple objects in a unified way are still missing.

This thesis aims to combine the task of multi-object tracking with the tasks of 6D pose estimation and tracking. To this extent, we present 6DCenterPose, a fully convolutional neural network which can track the 6D pose of multiple objects over a sequence of images. Following the local approach of 6D pose tracking methods [10], 6DCenterPose operates on pairs of images, the previous and current frames, and predicts the pose changes of the objects in the scene. Given the initial poses of all objects, the pose changes are used to propagate the poses from the previous to the current frame, and the procedure is iterated. Following recent multi-object tracking approaches [14, 19], we represent objects as points by their 2D centre location, which are predicted by a heatmap. All other quantities, like the change of pose, are also predicted using a heatmap, while the prediction is obtained at the location of the object centre. Additionally, a local offset is predicted, which a simple greedy matching algorithm uses to link centre predictions between the previous and current frames. To further guide the prediction process for the current frame, the heatmap prediction of the previous frame is used as input to the method.

Training deep neural networks requires large amounts of annotated training data. Annotating images for 3D tasks like 6D pose estimation is inherently more difficult than providing annotations for 2D tasks. Annotations have to be provided in 3D metric coordinates relative to the camera instead of providing annotations in pixel coordinates. Due to this difficulty, recent approaches [8, 10] use synthetic training data, which already comes with annotations. But this creates the new problem of bridging the reality gap [20] as simulated images do not fully resemble real images.

We follow the approach of using photo-realistic, physically plausible synthetic images to train 6DCenterPose. Following *se(3)*-TrackNet [10], 6DCenterPose uses separate input branches for the previous and current frames to decouple their feature representation. During training, both images are synthetic, while during testing, the current image is obtained from a real sensor, while the previous image is a rendering of the last pose prediction. This pushes the reality gap to the input branch of the current frame while the branch for the previous frame can be effortlessly aligned between training and testing.

Accurately and consistently tracking the poses of all objects in a scene is important for robot manipulation tasks. As a surrogate task, we evaluate the proposed method on the board game *Ubongo3D* as it exhibits similarities to many real-world tasks. The pose of the manipulated object needs to be known to accurately grab it, while the poses of the other poses need to be known to avoid collisions with them and, in the case of *Ubongo3D*, place objects precisely on top of each other.

In summary, this thesis makes the following contributions:

- 6DCenterPose, a fully convolutional neural network, is proposed, which tracks the 6D poses of multiple objects in a sequence of RGB-D images.
- A synthetic data generation pipeline is proposed, which generates photo-realistic, physically plausible images. These images are used to train and evaluate the method.
- A disentanglement of the feature encodings of the inputs allows training the method exclusively on synthetic images. Experiments on synthetic and real trajectories for the task of the board game *Ubongo3D* show that the reality gap is successfully bridged.

The remainder of this thesis is structured as follows: Chapter 2 explains recent advances in 6D pose estimation and tracking, multi-object tracking, and the generation of synthetic data for training deep neural networks. Chapter 3 explains two related methods, CenterTrack [14] (Chapter 3.1) and *se(3)*-TrackNet [10] (Chapter 3.2), in detail as they build the basis of the proposed method of this thesis. Next, Chapter 4 explains 6DCenterPose, which combines 6D pose tracking with multi-object tracking and proposes, to the best of our knowledge, the first method for online multi-object 6D pose tracking while only requiring synthetic RGB-D images for training. The data generation pipeline, which generates synthetic images used for training and testing, is detailed in Chapter 5. Chapter 6 explains how we obtain real annotated trajectories, which are subsequently used to evaluate and compare the proposed approach to a baseline method. Finally, Chapter 7 concludes this thesis and provides directions for future research.

2. Related Work

This chapter gives an overview of related literature relevant to this thesis. Chapter 2.1 explains approaches for 6D pose estimation, which is the task of predicting the 6D poses of all objects in a single image. In contrast, Chapter 2.2 explains approaches for 6D pose tracking. These approaches are concerned with tracking the 6D pose of a single object in a sequence of images. Similarly, Chapter 2.3 presents recent approaches for the task of multi-object tracking. These approaches track multiple objects in a sequence of images, but they focus only on 2D or 3D bounding boxes. Finally, Chapter 2.4 discusses how recent approaches use synthetic images to train neural network-based methods.

2.1. 6D pose estimation

6D pose estimation is the task of determining the translation and orientation, represented by a 6D pose, of all objects in a scene from visual data [7]. The pose is expressed relative to the camera. Common approaches use either RGB images [7, 8, 21] or RGB-D images [22, 23] as visual information. However, the RGB-based methods can usually be extended to using depth information as well. Most state-of-the-art approaches use convolutional neural networks (CNNs) as the basis of their approaches [7, 22, 23], but following the recent trend of the computer vision literature, Transformer-based [24] methods grow in popularity [25, 26, 27].

SSD-6D [9] treats the task of 6D pose estimation as a classification problem. In an offline stage, they use the 3D CAD models of the objects to build dictionaries of different viewpoints, one dictionary for every object. The CNN-based method then predicts bounding boxes and scores for viewpoints and in-plane rotation given an RGB image. These scores are used to generate pose hypotheses which are matched with the viewpoint directory to determine the most probable pose hypothesis. Finally, a pose refinement step is used, either RGB-based or using the Iterative Closest Point (ICP) [28] algorithm if additional

depth images are available, to generate the final pose output. SSD-6D is a template-based approach, which has advantages for texture-less objects, but they usually struggle with occlusion [7]. Furthermore, the discrete output space of the poses further limits accuracy [27].

To improve the shortcomings of template-based methods, recent approaches treat the 6D pose estimation problem as a regression problem rather than a classification problem [7, 8]. PoseCNN [7] uses a CNN-based architecture to predict a segmentation mask given an RGB image. The segmentation mask is used in a Hough voting scheme to determine the 2D centre of the objects in the scene. Together with a depth prediction for the centre locations, the 3D location is obtained. Finally, a Region of Interest (RoI) pooling [29] is performed on the feature maps obtained from the CNN to regress to a quaternion representation of the orientation of each object. If depth images are available, the pose is refined using ICP [28].

DOPE [8] uses a CNN-based model to predict two types of outputs given an RGB image. The first output is nine belief maps which predict the eight projected corners of the 3D bounding box and the projected centre locations. The second output is eight vector fields predicting the direction from each projected corner to the corresponding centre. During inference, they are used to determine which corner belongs to which object, similar to PoseCNN [7]. Once the corners are assigned to the centres, a PnP algorithm [30] is used to retrieve the pose of the objects.

Another line of work directly uses depth images in addition to RGB images to make pose predictions [22, 23] instead of only using depth images for pose refinement. DenseFusion [22] starts by predicting segmentation masks given the RGB image. The predicted mask is then used to crop the RGB image around the object and obtain a partial point cloud given the depth image. A CNN-based architecture is used to compute pixel-wise colour embeddings using the cropped RGB image, and a PointNet [31] is used to compute pixel-wise geometric embeddings using the predicted point cloud. A dense fusing procedure is used to fuse both embedding types, which is used to make pixel-wise pose predictions together with a confidence score. They further propose a pose refinement step to predict a pose residual used to improve the pose prediction. The final pose output is the pixel-wise pose prediction with the highest confidence score.

MoreFusion [23] is a multi-view approach, which means that multiple images from the scene with different camera poses are used to make the 6D pose prediction. This is done by predicting instance maps from the RGB images, which are fused with camera tracking obtained from the depth images to obtain volumetric maps from the scene. These volumetric maps contain information on which areas of the 3D scene are free and which

are occupied by an object or obstacle. The volumetric map and the RGB-D data are used to make initial pose predictions of the objects, which are subsequently used, together with the 3D CAD models of the objects and the volumetric maps, in a collision-based refinement step to ensure the physical plausibility of the predicted poses.

So far, all the discussed methods are based on CNNs. Recently, Transformer-based [24] methods have grown in popularity [32, 33]. This trend is also visible in the 6D pose estimation literature.
T6D-Direct [21] builds on the object detection method DETR [33], which treats object detection as a set prediction problem. A conventional CNN backbone is used to extract image features from RGB images which, together with a positional encoding, forms the input to the transformer encoder. The transformer decoder then attends to the encoder output and object queries, which are learned positional embeddings. Finally, the output of the decoder is passed to a shared feedforward network to produce bounding boxes and class probabilities. T6D-Direct extends this approach by adding two additional output heads to independently predict the translation and orientation component of the 6D pose. In the experiments, problems of the method are shown to predict the orientation of objects. To alleviate this problem, YOLOPose [25] is proposed. The same architecture as in T6D-Direct is used, but instead of predicting the orientation of the objects directly, they predict keypoints of the objects, specifically their interpolated bounding boxes [34]. These keypoints are then used as input to a separate rotation estimation network to predict the orientation of the objects.

All discussed methods so far follow the approach of instance-level pose estimation, meaning they are trained to estimate the pose of one specific instance of a category of objects, e.g. one specific cup. Category-level pose estimation, on the other hand, is the task of learning to predict the poses of objects from a category, even though the specific instance of the object was not seen during training. This is more challenging since, during inference, usually, no 3D CAD models of the objects are available.

6D-Vit [26] is a Transformer-based approach for category-level 6D pose estimation using RGB-D images as input. 6D-ViT processes objects separately, so if more than one instance is present in the scene, an instance segmentation pre-processing step is necessary. Given the predicted segmentation map, the RGB images are cropped around the object, and a partial point cloud of the object is extracted from the depth image. 6D-ViT then processes the two modalities separately. The cropped RGB image is used as input to the sub-network Pixelformer, which uses a Transformer encoder and a multilayer perceptron decoder to produce a pixel-wise instance representation. The partial point cloud is used as input to the sub-network Pointformer which also uses a Transformer encoder and a multilayer

perceptron decoder to produce a pixel-wise instance representation. Both instance representations are fused with a category shape prior using a multisource aggregation network to produce a joint and dense object representation. This representation is used together with the obtained point cloud and the category shape prior to obtain a pose estimation using the Umeyama algorithm [35].

Perpendicular to the task of pose estimation is the task of pose refinement. Given a pose estimate of any of the presented methods, DeepIM [36] refines the pose to improve the prediction. For this, the 3D CAD model of the object is required, which is used to generate a rendering of the initial pose estimate. DeepIM then uses this rendering and the image used to produce the initial pose as input to a CNN-based architecture to predict a residual between the poses shown in the rendering and the real image. The residual pose is then used to improve the pose estimate iteratively.

The approaches discussed in this chapter produce accurate pose estimates of all the objects given a single RGB(-D) image. However, especially in robotics applications, we are interested in the poses of the objects in a sequence of images which is necessary, for example, in manipulation tasks. One approach is using the discussed methods to estimate the poses of the objects from scratch for every frame of the video. However, the discussed methods usually trade off accuracy for speed, which limits the use of this approach. Furthermore, the strong correlation between frames of a video is ignored, leading to less coherent pose estimates [10]. One possible solution to these problems is discussed in the next chapter.

2.2. 6D pose tracking

6D pose tracking is a task very similar to 6D pose estimation. Again, the 6D pose of an object relative to the camera is estimated, but in contrast to 6D pose estimation, the pose should be determined in a sequence of images instead of estimating the pose in a single image, allowing for the exploitation of temporal information [25]. However, these approaches are usually specific to an object and require retraining if used for a different object [12].

Tracking is a well-researched problem in robotics using Kalman filters or particle filters [37, 38]. MaskUKF [39] follows this line of work while still leveraging the advantages of deep neural networks. Given an RGB-D image, MaskUKF uses an image segmentation network to predict a segmentation map from the RGB image. The segmentation mask, together

with the depth image, is used to obtain a partial point cloud. This point cloud is then used as a measurement for an Unscented Kalman Filter to update the belief about the object's pose and velocity using the 3D CAD model of the object.

PoseRBPF [11] estimates the full posterior distribution over 6D poses by factorising the posterior into the 3D location and 3D orientation component. This factorisation leads to an efficient sampling scheme for a Rao-Blackwellized particle filter [40]. Before applying the method, a 3D CAD model of the object is used to train an auto-encoder to build a dictionary of different object orientations. The tracking is initialised by a 2D bounding box prediction of the object. An initial 3D location can be obtained by back projecting the 2D centre and sampling different distances. This sampling initialises the particles of the particle filter. For each subsequent frame, the particles are propagated using a motion model. Next, the particles are updated using the current observation. A bounding box for the translation estimation of each particle is obtained, which is subsequently used to compute a region of interest for that estimation. The region of interest is passed through the auto-encoder to compute a feature embedding. Using the cosine distance between this embedding and all the embeddings of the precomputed dictionary, the most likely orientation is obtained, which is used to update the rotation distribution of the particle. Finally, the distribution of the translation is updated, and the process can be iterated for the next frame.
The big advantage of filtering-based approaches is that the pose prediction contains information about the uncertainty of the estimate. This information is usually unavailable when using deep learning-based approaches, even though it can be critical in robotics applications such as grasp planning [41]. Their drawbacks, especially in the case of particle filters, are their low runtimes [11].

Another line of research views 6D pose tracking as a learning-based problem which is solved using neural networks [10, 12, 27]. This has the advantage that no occlusions [37] or likelihoods [11] have to be explicitly modelled, but they are instead learned from large amounts of data. Furthermore, state-of-the-art learning-based approaches exhibit fast runtimes [10], making them well-suited for robotics applications.

Garon and Lalonde [12] propose a simple CNN-based model to solve the 6D pose tracking problem. The input to the method is the RGB-D image of the current timestamp and a rendering of the pose prediction of the previous timestamp. Therefore, their method requires a CAD model of the object. The method has two independent convolutional layers, one for each image, for the input. The outputs of these layers are then concatenated before being passed to the rest of the network. The method's output is the change of pose between the previous frame and the current frame in a joint representation. This change

in pose is then used to obtain the pose prediction of the current frame, after which the process is iterated to obtain poses for all images in the sequence.

DeepIM [36], originally proposed as a pose refinement method (as discussed in Chapter 2.1), is shown to be extended to 6D pose tracking. Instead of estimating the pose residual between a rendering of the pose estimation of the current frame and the current frame, the residual between a rendering of the previous pose estimation and the current frame is estimated. Again, this residual is used to propagate the pose estimation of the previous frame to the current frame. Contrary to Garon and Lalonde [12], the RGB-D image of the previous and current frames are concatenated before being passed to a FlowNetSimple [42], instead of being passed to separate input branches. Further, they explain the importance of using a decoupled pose prediction by using separate output branches for the translation and orientation component of the pose.

se(3)-TrackNet [10] combines the approaches of Garon and Lalonde [12] and DeepIM [36]. Two separate input branches are used for the rendering of the previous pose and the current pose observation to decouple the feature representations while using separate output branches for the change in orientation and change in translation. The proposed method of this thesis takes inspiration from *se(3)*-TrackNet, so the approach is explained in detail in Chapter 3.2.

All discussed methods so far rely on access to a 3D CAD model of the object. In practice, this dependency can be limiting, especially for novel objects unknown during training. BundleTrack [43] proposes an approach to alleviate this problem by not requiring any instance-level or category-level 3D CAD models. Given the previous instance segmentation mask and the current RGB-D image, the instance segmentation mask for the current frame is computed. Both segmentation masks are used to crop the respective image around the target object, after which a keypoint detection network is utilised to compute keypoints and feature descriptors for the cropped images. The keypoints are then used to predict a first estimate for the change of pose between the two views. To obtain the final pose estimate, pose graph optimisation is used. The nodes of the pose graph are obtained from a memory pool of previous keypoint estimates. The current frame is added to the memory pool if it corresponds to a novel view. The pose graph optimisation is performed online and results in the pose prediction of the current frame.

All discussed methods so far rely on CNN-based architectures. But similar to Chapter 2.1, Transformer-based methods are also used for 6D pose tracking. VideoPose [27] predicts absolute poses, in contrast to the other, discussed approaches which predict changes in poses, and exploits temporal information by attending to features of previous frames. Starting point of VideoPose is a generic object detection method to obtain bounding boxes

of the objects in the scene from an RGB-D image. The image is then cropped around the object and used as input to a Transformer encoder. To make a pose prediction, a Transformer decoder attends to the encoder features of the current frame and the encoder features of previous frames. The decoder output is then used as input to a 6D pose regression network, which outputs values for the orientation, the distance to the camera, and the 2D image location in three separate branches.

All discussed methods in this chapter can track the 6D pose of a single object in a sequence of videos. One drawback of these methods is that they are specific to the object they are trained on, and they can not be readily applied to a different object [12]. In contrast, the methods discussed in Chapter 2.1 can estimate the 6D poses of all objects in the scene, but only for a single frame. While it is possible to use a separate tracker for every object in the image sequence or estimate all poses from scratch for every image using a pose estimation method and link them in a separate step, both approaches are not ideal.
Schmauser et al. [44] propose an approach for joint 6D pose estimation, 3D reconstruction, and data association, resulting in 6D multi-object tracking. However, their approach is offline and optimises the object trajectories over a whole RGB-D image sequence. Therefore, it is not applicable for online tasks like robot manipulation.
In contrast, this thesis aims to combine the approaches of 6D pose estimation and tracking. A method is proposed that consistently tracks the 6D poses of all objects in the scene using a single approach in an online fashion.

2.3. Multi-object tracking

Multi-object tracking (MOT) is the task of tracking all object instances, usually pedestrians, in a sequence of images obtained from a single camera [13]. A trajectory of an object is represented by a time-ordered set of object detections, which are usually represented as bounding boxes [45]. The classical MOT approach follows the tracking-by-detection paradigm, which treats MOT as a two-step process. The first step consists of frame-by-frame object detection to obtain bounding boxes for all objects in the scene and for all frames. The second step links bounding boxes corresponding to the same object instance across frames, formulated as a data association problem, to obtain trajectories. Since the bounding boxes can be obtained by any object detection method, approaches following this paradigm focus on solving the data association problem.

To prevent the linking of false positive detections to trajectories, common MOT approaches apply a threshold on the bounding boxes obtained from the object detection method and

only link bounding boxes with a high confidence score. Zhang et al. [46] argue that this procedure can harm the robustness of MOT approaches since low-score bounding boxes not only correspond to background predictions but also to occluded objects. To increase robustness and to prevent the loss of true positive bounding box predictions, they propose the data association algorithm BYTE [46]. Similar to common MOT methods, the first step is to apply a threshold to the bounding boxes received from the object detector. The high-score bounding boxes are matched to the existing tracks in a motion-based fashion by predicting the locations of the previous bounding boxes in the current frame using a Kalman filter [47] and computing the similarity to the bounding boxes of the current frame using the intersection over union (IoU) or Re-ID features. In a second matching step, unmatched tracks are matched to low-score bounding boxes using the same similarity measures as before. In addition to the data association method, they propose a MOT approach, ByteTrack [46], by combining object detections obtained from the object detector YOLOX [48] with their data association method BYTE.

Instead of solving the data association problem using motion, a common approach is to formulate the data association as a graph partitioning problem [49, 50, 51], where graph nodes represent object detections and edges represent whether two detections belong to the same trajectory. Finding an optimal graph partition determines which objects belong to the same trajectory and, by this, links them to tracks. Brasó and Leal-Taixé [45] propose a message passing network which is used to learn the partitioning of the graph. The first step is to build the graph. Nodes correspond to appearance feature embeddings obtained from a CNN by passing the cropped bounding boxes through it. Edges are obtained for bounding boxes of different frames by computing a feature vector containing relative bounding box sizes, locations, and temporal distances, and then feeding it through a multilayer perceptron to obtain geometry embeddings. Next, several message passing steps are performed to update the edge and node embedding and, by this, propagate higher-order information through the graph. Finally, the updated edge embeddings are used to perform binary classification to determine which edges are active. Here, an active edge represents that the two corresponding nodes should be linked to a track. While this approach allows obtaining globally consistent trajectories of all the objects in the scene, it requires object detection of all frames to build the graph. Therefore, the method is not online and, by this, not fitting for a robot manipulation task.

Zhang et al. propose mmMOT [52], another approach using graph optimisation, to solve the data association problem. But instead of relying only on colour images obtained from a camera, a framework is proposed that fuses data from different sensors, e.g. cameras, LiDAR, or radar, in their experiments, to increase the robustness and accuracy of their approach. Following the tracking-by-detection paradigm, the first step is to obtain

bounding boxes of the objects using an object detection method. Next, the detections are used to separately extract features from each sensor modality. These features are used in a robust fusion module to obtain a joint feature representation. The robust fusion module also preserves the individual feature representations to increase robustness. If one of the sensors malfunctions, the method can simply use the other sensors. mmMOT is an online method and operates on pairs of detections. Following the described procedure, a joint feature representation of each image is obtained, which is then used in an adjacency estimator to predict the scores necessary for graph optimisation. The graph optimisation is solved using an out-of-the-box solver to obtain the final association of the detections.

3D MOT is a task similar to 2D MOT, but instead of associating 2D detections across frames, the goal is to associate 3D detections across frames [17]. A 3D detection is usually represented by a 3D bounding box, which consists of a 3D location, the 3D size of the bounding box, a confidence score, and a heading angle. Therefore, 3D object tracking is also very similar to 6D pose tracking, as both representations share the same 3D location. But the heading angle of the 3D bounding box is not enough to recover the full 3D orientation of the 6D pose. Furthermore, 3D MOT only uses 3D measurements, like point clouds obtained from LiDAR sensors, while 6D pose tracking operates on RGB(-D) images.

Weng et al. [17] propose an approach for 3D MOT using classic approaches, namely a Kalman filter [47] and the Hungarian method [53]. Following the tracking-by-detection paradigm, they use a 3D detection module to obtain 3D bounding boxes for the current frame given a LiDAR point cloud. A 3D Kalman filter using a constant velocity model is used to predict the state of the previous detections for the current frame. To match the predicted trajectories to the current detections, an affinity matrix is constructed by computing the 3D IoU or negative centre distance. This results in a bipartite graph matching problem which is solved using the Hungarian algorithm. Each trajectory is then updated using the matched detection following the state update of the Kalman filter.

CenterPoint [18] represents objects as points instead of a full 3D bounding box. A standard LiDAR-based backbone is used to extract a feature representation of the input point cloud. The feature representation is then flattened into an overhead map view, which is used as input to a standard image-based keypoint detector to find the object centres. All other object properties, like the 3D size and orientation, are predicted from the point feature at the centre location. Furthermore, given the map views of the current frame and the previous frame, the velocity of the objects is estimated. The velocity is used to greedily match previous detections to current ones and, by this, track the objects in the scene. Due to the sparse nature of point clouds [54], the point feature of the centre location might not contain enough information to accurately regress to all properties of the 3D bounding

box. Because of this, CenterPoint proposes a second stage for pose refinement. Given the current 3D bounding box prediction, point features at the centres of each bounding box face are extracted and, together with the centre point feature, passed through a multilayer perceptron, which outputs a confidence score and a box refinement.

While tracking-by-detection successfully solves the MOT problem, approaches following this paradigm are becoming increasingly complicated [16] and mainly focus on increasing the accuracy while neglecting practical aspects like runtime [17], which is important for downstream tasks like robot manipulation. Another paradigm to solve the MOT problem is tracking-by-regression, where object detection and tracking are solved in a joined fashion by regressing past detections to the current frame. These methods are, by definition, local and online, making reinitialisation of lost long-range tracks impossible. Instead, they are simple, fast, and more accurate in the local regime [14], making them well-suited for applications like robot manipulation.

D&T [16] is an approach which uses correlation maps of adjacent frames to learn the tasks of detection and tracking simultaneously. The method builds upon the fully convolutional object detector R-FCN [55]. Given two adjacent input images, the first step is to compute convolutional feature maps for each image using a backbone network. Based on these feature maps, a region proposal network is used to propose candidate regions based on anchor boxes [56]. These candidate regions are used in RoI pooling [29] to aggregate position-sensitive score and regression maps from intermediate convolutional layers to classify boxes and refine their coordinates. This way, bounding boxes are predicted independently for the images. To link the detections to trajectories, intermediate position-sensitive regression maps from both frames are used to compute correlation maps. They are used, together with the regression maps of the individual frames, in an RoI tracking operation to output the box transformation from one frame to the other. These box transformations are then used to link the detections of the adjacent frames to create tracks.

Tracktor [15] uses a standard two-stage object detection method, in this case Faster R-CNN [56], and exploits its bounding box regressor to track objects without the need to specifically train the method on the tracking task. Faster R-CNN extracts features from an image using a backbone. Then, a region proposal network is used to extract bounding box proposals for the objects in the scene. RoI pooling [29] is used to extract feature maps for each proposal which are then used in classification and bounding box regression heads to predict the bounding boxes. Tracktor uses the bounding box regression mechanism to propagate the bounding boxes of the previous frame to the current frame. Specifically, RoI pooling is performed on the features of the current frame but with the bounding box

coordinates of the previous frame, assuming small object motion between the two frames. This way, bounding boxes are obtained in the current frame while preserving the object identity and, therefore, creating tracks. Furthermore, they propose extensions to Tracktor to alleviate problems arising due to the local nature of the approach. A motion model is used if the assumption of small object movements can not be satisfied. If the camera is moving in the image sequence, camera motion compensation [57] is used, and for low framerates, a constant velocity assumption [58, 59] is used.

CenterTrack [14] also follows the tracking-by-regression paradigm and represents objects as their centre location instead of their full bounding box. Other object properties, like bounding box size, are regressed from the features at the centre location. Furthermore, given two adjacent input images and the previous detections represented by a heatmap, CenterTrack predicts the offset of each object from the previous frame to the current frame, linking them to tracks. The method proposed in this thesis builds upon CenterTrack. Therefore, it is explained in detail in Chapter 3.1.

Following the general trend of the recent computer vision literature, Transformer-based [24] methods are used for MOT in a tracking-by-attention paradigm [60, 61, 62].
Trackformer [60] formulates MOT as a set prediction problem and jointly performs tracking and detection of the objects in the scene. Features from each frame are extracted using a common CNN-based backbone. These features are then encoded using a standard Transformer encoder with self-attention. Trackformer uses two types of queries in a standard Transformer decoder to reason about the objects in the scene. Static and learned object queries [33] are used to initialise new tracks. Each object query learns to predict bounding box position and size while avoiding duplicate detections using self-attention. Track queries follow objects through the image sequence while preserving their identity and adapting to changes in the locations of the objects. Both query types and the image feature encoding serve as input to a standard Transformer decoder that produces output embeddings. The output embeddings are mapped to the bounding boxes and classification scores using a multilayer perceptron. Furthermore, the output embeddings are used as track queries as input to the Transformer decoder for the next frame. This way, Trackformer represents objects as queries and uses them to link detections across time using self- and encoder-decoder attention.

All the discussed approaches successfully track multiple objects across sequences of images, but only as 2D or 3D bounding boxes. In this thesis, we also work on the task of MOT, but instead of representing the objects as bounding boxes, we track them using their 6D pose.

2.4. Synthetic data generation

Due to the difficulty of the tasks of 6D pose estimation and tracking, deep learning-based methods require a large amount of training data. However, annotating the 6D poses of objects in images is inherently more complex than providing annotations for tasks like object detection or image classification, as annotations need to be provided in 3D metric space instead of in 2D image coordinates [8]. This challenge is reflected in the number of images and annotations of commonly used data sets. The ImageNet [63] classification dataset contains approximately 1.3 million images with the same number of annotations, divided into 1000 classes. The 2014 release of the object detection dataset MS COCO [64] contains approximately 165k images with 886k annotated instances divided into 80 categories. In contrast, the 6D pose estimation dataset LINEMOD [65] contains around 1000 images with objects from 15 categories, for which annotations are provided, resulting in 15k annotations.

To alleviate this problem, several approaches explore the use of synthetic training data [8, 10, 11]. Using synthetic images has the significant advantage that highly accurate pose annotations are readily available as they are needed to render the synthetic images in the first place. However, using synthetic images shifts the problem of the laborious task of providing pose annotations for real images to another problem. The simulated environment used to render the synthetic images will not precisely match the real world due to errors in system identification, unmodeled physical effects, and low-fidelity simulated sensors. These phenomenons combined are known as the reality gap [20]. Bridging the reality gap (called sim-to-real transfer) is the challenge methods trained on synthetic images need to face.

One approach to bridge the reality gap is the idea of domain adaptation [66, 67]. The idea is to train a model on a source domain, here on synthetic images, and then transfer to a target domain, which in this case are real images. Ganin and Lempitsky [66] propose an approach which combines deep feature learning with domain adaptation. Their proposed method consists of a backbone that extracts features on top of which a head is used for the task at hand. During training, annotated samples from the source domain are used in a supervised learning setting. In addition, unlabeled data from the target domain is used in an unsupervised manner. A domain classification head is used on top of the backbone, which predicts whether the input image is from the source or the target domain. During training, the parameters of the backbone are trained to maximise the loss of the domain classifier, while the parameters of the domain classification branch and task-specific branch are trained to minimise their respective objectives. This training approach encourages the

backbone to learn domain invariant features, which allows the learned method to be used in the target domain, i.e. in this case for real images.

Shrivastava et al. [67] propose an approach similar to Generative Adversarial Networks (GANs) [68] to improve the realism of synthetic images. They train a refinement network that takes a synthetic image as input and outputs an image more closely resembling a real image. The refinement network is trained in a way to preserve the annotations of the synthetic image so it can be used in subsequent training of neural networks. Furthermore, the development of artefacts found in GAN-based approaches is discouraged. To inform the refinement network what real images look like, a discriminator is used, which takes unlabeled real images and the refined synthetic images as input and is trained to determine if the input is synthetic.
Both approaches assume that the source and target domains are sufficiently close, which is not trivially satisfied in practice [10]. Furthermore, both approaches require real images of the target domain, even though only unlabelled ones, which further hinders their applicability in unseen scenarios.

To use synthetic images for training a deep neural network, they must be somehow generated. The most straightforward approach used by some methods [8, 9, 11, 12, 36] is to sample random poses of the objects and render them on top of images obtained from commonly used datasets [64, 69, 70, 71]. While this approach produces images with 6D pose annotations, the generated images look unrealistic. Additionally, image contexts, like interactions of the rendered objects with the background and shadows, are ignored. This approach is mainly used because of the lower computational cost for generating the images compared to rendering full scenes [72].

Research regarding how transferable features learned in neural networks are [73] suggests that the rendered images should be as similar to the real world as possible. This idea is further supported by Movshovitz-Attias et al. [72], who investigate the influence of the realism of model textures when training neural networks using synthetic images. This research suggests using full synthetic 3D scenes instead of overlaying static images with rendered models. Many synthetic datasets using full synthetic scenes exist [74, 75, 76, 77, 78], but they are expensive to generate as they require artists to carefully model specific environments [79], negating the big advantage of reducing human involvement when using synthetic data.

Instead of modelling scenes photo-realistically, Tobin et al. [20] propose domain randomisation (DR). In contrast to training a model on a single photo-realistic environment, the idea is to expose the model to a wide range of simulated environments by randomising the settings of the simulator, like the number and shape of distractor objects, positions of all

objects, light settings, camera settings, among others. In particular, they also randomise all textures by using random RGB values, gradients between two RGB values, and checker patterns of two RGB values. They explicitly use non-photo-realistic and not physically plausible images and, instead, provide a lot of variability in the synthetic scenes. Hence, the real world is simply another configuration, allowing the model to generalise to it. Tremblay et al. [79] build on domain randomisation, but instead of using random textures, they randomly sample them from a set of photo-realistic textures. The randomised objects are then rendered on top of background images from a real dataset [71] instead of placing them in full 3D scenes.

So far, all discussed approaches randomly sample object poses. However, random sampling can lead to unnatural object penetration, which does not occur in the real world. Especially for depth images, object penetrations can introduce an undesired bias to the data distribution [10]. While aligning some physical properties between the real world and the simulator may be challenging, gravity and collision can be easier aligned [80]. Tremblay et al. [81] place objects in a way to avoid object penetration and then let them fall under the influence of gravity while taking images from different viewpoints of the falling objects. Wen et al. [10] propose the idea of physically plausible domain randomisation, where most simulator properties are randomised as in domain randomisation [20], including object poses allowing object penetration. Several physics simulation steps follow the placement to separate colliding objects and let them fall onto a table. Once all objects reach stable poses, an image using the simulated camera is obtained. These approaches combine the benefits of domain randomisation with the closer alignment of the photo-realistic synthetic images to the real world while being physically plausible.

This thesis follows the idea of physically plausible domain randomisation and proposes a data generation pipeline in Chapter 5, which produces photo-realistic, physically plausible images while randomising most simulator properties. Some approaches [7, 12, 72, 79, 80] highlight the benefits of using real data in combination with synthetic images. But the task solved in this thesis is not used in an existing dataset, and creating a dataset is beyond the scope of this thesis, so we only rely on synthetic training images.

3. Preliminaries

This chapter discusses two related works in detail as they build the foundation for the proposed method of this thesis. Chapter 3.1 explains CenterTrack [14], a multi-object tracking method. It represents tracked objects as points and uses a greedy matching algorithm based on sparse optical flow to associate detections between frames. Chapter [10] explains *se(3)*-TrackNet, a single-object 6D pose tracking method. They disentangle the input features, which allows training the method exclusively on synthetic images. Furthermore, they represent poses using a Lie Algebra representation which helps to learn pose residuals effectively. These ideas are combined to design the proposed approach of this thesis.

3.1. CenterNet & CenterTrack

CenterNet

CenterNet [19] is an object detection method representing objects as a single point. This representation is in contrast to other current object detection methods [29, 56, 82], which represent objects by a tightly-fitting axis-aligned bounding box. Given an input image $\boldsymbol{I} \in \mathbb{R}^{W \times H \times 3}$ with width W, height H and three colour channels, the aim of CenterNet is to produce a low-resolution heatmap $\hat{\boldsymbol{Y}} \in [0,1]^{W/R \times H/R \times C}$ with a downsampling factor $R = 4$ and C classes. A prediction $\hat{\boldsymbol{Y}}_{x,y,c} = 1$ corresponds to a detected object of class c at location (x, y), while $\hat{\boldsymbol{Y}}_{x,y,c} = 0$ is background. In practice, the values of $\hat{\boldsymbol{Y}}$ are between 0 and 1 and can be interpreted as the confidence of the prediction.

To learn the heatmap $\hat{\boldsymbol{Y}}$, the centre locations $\boldsymbol{p}_i$ of the objects in the training images need to be known. If only bounding box annotations are given, represented by the coordinates of the four corners of the bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$, the centre coordinate can be obtained as $\boldsymbol{p} = ((x_{min} + x_{max})/2, (y_{min} + y_{max})/2) \in \mathbb{R}^2$. A ground truth heatmap

$\boldsymbol{Y} \in [0,1]^{W/R \times H/R \times C}$ can be constructed using these centre locations. Since the heatmap is a downsampled version of the input image, the centre locations need to be downsampled as well and are obtained by $\tilde{\boldsymbol{p}} = \lfloor \boldsymbol{p}/R \rfloor = (\tilde{p}_x, \tilde{p}_y)$. Predicted centre locations near the ground truth centre location can also produce sufficiently good bounding boxes. Instead of penalising all incorrect location predictions equally, a region around the actual location is considered correct. This assumption is realised by an unnormalised 2D Gaussian around the centre location

$$\boldsymbol{Y}_{x,y,c} = \exp\left(-\frac{(x-\tilde{p}_x)^2 + (y-\tilde{y}_y)^2}{2\sigma_p^2}\right), \tag{3.1}$$

where the standard deviation σ_p, i.e. the spread of the kernel, is dependent on the object size [83]. These Gaussians are used for all the classes' centre points to obtain the full ground truth heatmap. If two Gaussians of the same class overlap, the maximum value at that location is used. The maximum is preferable over taking the average of the Gaussians at that location to ensure that the close-by peaks remain distinct instead of smoothing them out [84].
The objective to train the heatmap output of CenterNet is given by a variant of the focal loss [83, 85]:

$$L_{\mathrm{k}} = \begin{cases} (1-\hat{\boldsymbol{Y}}_{x,y,c})^{\alpha} \log(\hat{\boldsymbol{Y}}_{x,y,c}) & \text{if } \boldsymbol{Y}_{x,y,c} = 1 \\ (1-\boldsymbol{Y}_{x,y,c})^{\beta} (\hat{\boldsymbol{Y}}_{x,y,c})^{\alpha} \log(1-\hat{\boldsymbol{Y}}_{x,y,c}) & \text{otherwise,} \end{cases}$$

where N is the number of objects in the image and $\alpha = 2$ and $\beta = 4$ are hyper-parameters. The fact that the input is downsampled by a factor R introduces a discretisation error since the ground truth location could lie between two pixels in the downsampled location output. To recover the actual location, CenterNet outputs a local location regression map $\hat{\boldsymbol{O}} \in \mathbb{R}^{W/R \times H/R \times 2}$, containing an offset in x and y directions for all location outputs. To save computation and memory costs, a single location regression map is used for all classes c. It is trained using an $L1$ loss

$$L_{\text{off}} = \frac{1}{N} \sum_{\boldsymbol{p}} \left| \hat{\boldsymbol{O}}_{\tilde{\boldsymbol{p}}} - \left(\frac{\boldsymbol{p}}{R} - \tilde{\boldsymbol{p}} \right) \right|,$$

where $\hat{\boldsymbol{O}}_{\tilde{\boldsymbol{p}}}$ is the value of the location regression map at location $\tilde{\boldsymbol{p}}$. Supervision is only provided at the centre locations $\tilde{\boldsymbol{p}}$, while all other regions are ignored.

So far, the method only predicts downsampled centre locations and refines them to the original input resolution. To get full bounding box predictions, CenterNet additionally outputs a size map $\hat{\boldsymbol{S}} \in \mathbb{R}^{W/R \times H/R \times 2}$, which regresses to the height and width of the bounding box given the centre location. Again, one size map is used for all classes c. The size map is again learned using an $L1$ loss

$$L_{\text{size}} = \frac{1}{N} \sum_{k=1}^{N} |\hat{\boldsymbol{S}}_{\boldsymbol{p}_k} - \boldsymbol{s}_k|,$$

where $\hat{\boldsymbol{S}}_{\boldsymbol{p}_k}$ is the values of the size map at centre location $\boldsymbol{p}_k$ and $\boldsymbol{s}_k = (x_{max} - x_{min}, y_{max} - y_{min})$ is the ground truth size of the bounding box, which is obtained from the bounding box annotations. Supervision is again only provided at centre locations.
The overall training loss is the weighted sum of these losses

$$L_{\text{det}} = L_{\text{k}} + \lambda_{\text{size}} L_{\text{size}} + \lambda_{\text{off}} L_{\text{off}}.$$

The weights are set to be $\lambda_{\text{size}} = 0.1$ and $\lambda_{\text{off}} = 1$.
The architecture of CenterNet consists of a fully convolutional encoder-decoder backbone network to generate image features, which are then passed to separate heads to output the heatmap, the size map and the location regression map.

CenterTrack

CenterTrack [14] extends CenterNet to perform tracking of objects through a sequence of images. It is a purely local approach and only associates objects in adjacent frames. If the tracking for one object is lost and later detected again, the tracks will not be linked; rather, a new track is initialised.
More precisely, CenterTrack follows the idea of tracking-conditioned detection [86, 87, 88]. Instead of only providing the current frame to detect objects in it, previous detections and frames are provided to the network as well.
CenterTrack takes as input the current frame $\boldsymbol{I}^t \in \mathbb{R}^{W \times H \times 3}$ at time t, the previous frame $\boldsymbol{I}^{t-1} \in \mathbb{R}^{W \times H \times 3}$ at time $t-1$, and the detections of the tracked objects in the previous frame $B^{t-1} = \{b_0^{t-1}, b_1^{t-1}, ...\}$. Each object $b = (\boldsymbol{p}, \boldsymbol{s}, w, id)$ is represented by its centre $\boldsymbol{p} \in \mathbb{R}^2$, its size $\boldsymbol{s} \in \mathbb{R}^2$, its detection confidence $w \in [0, 1] \subset \mathbb{R}$, and an unique identity $id \in \mathbb{I}$. This representation follows the representation of CenterNet, with the addition of the identity id.

The detections of the previous frame need to be pre-processed to be a suitable input to the method. CenterTrack exploits the point-based nature of the tracks to achieve a suitable input. Similar to the heatmap output of CenterNet, all detections of the previous frame B^{t-1} are displayed using a 2D Gaussian at their respective centre location (see Equation 3.1) in a class-agnostic heatmap $\boldsymbol{H}^{t-1} \in \mathbb{R}^{H \times W \times 1}$. Only detections with a confidence w greater than a threshold τ are displayed to avoid the propagation of false positive detections.
The resulting heatmap $\boldsymbol{H}^{t-1}$ is concatenated with the previous frame $\boldsymbol{I}^{t-1}$ and the current frame $\boldsymbol{I}^{t}$, resulting in seven input channels. Besides this change, the rest of the backbone architecture of CenterNet remains unchanged.

CenterTrack aims to find all objects in the current frame, similar to CenterNet, but also link these detections across time by assigning objects that appeared in the previous frame a consistent id. To achieve this association, CenterTrack adds a 2D displacement map $\hat{\boldsymbol{D}}^{t} \in \mathbb{R}^{W/R \times H/R \times 2}$ to the existing outputs of CenterNet, where $R = 4$ is the same downsampling factor as for CenterNet. Given a detected object location $\hat{\boldsymbol{p}}^{t}$, the displacement $\hat{\boldsymbol{d}} = \hat{\boldsymbol{D}}_{\hat{\boldsymbol{p}}^{t}}$, where $\hat{\boldsymbol{D}}_{\hat{\boldsymbol{p}}^{t}}$ is the entry of the displacement map at location $\hat{\boldsymbol{p}}^{t}$, captures the difference in location of the object between the current frame and the previous frame $\hat{\boldsymbol{d}} = \hat{\boldsymbol{p}}^{t} - \hat{\boldsymbol{p}}^{t-1}$, similar to a sparse optical flow. The displacement map is learned using an $L1$ loss

$$L_{\text{dis}} = \frac{1}{N} \sum_{i=1}^{N} |\hat{\boldsymbol{D}}_{\boldsymbol{p}_i^t} - (\boldsymbol{p}_i^{t-1} - \boldsymbol{p}_i^t)|,$$

where $\boldsymbol{p}_i^{t-1}$ and $\boldsymbol{p}_i^t$ are the ground truth locations of the object. Supervision is only provided at the ground truth centre locations. The final training loss is

$$L_{\text{det}} = L_{\text{k}} + \lambda_{\text{size}} L_{\text{size}} + \lambda_{\text{off}} L_{\text{off}} + \lambda_{\text{dis}} L_{\text{dis}},$$

where $\lambda_{\text{size}} = 0.1$, $\lambda_{\text{off}} = 1$, and $\lambda_{\text{dis}} = 1$.
Data association is achieved using a simple greedy matching algorithm. Each detection in the current frame at position $\hat{\boldsymbol{p}}$ is greedily associated with the closest unmatched detection at position $\hat{\boldsymbol{p}} - \hat{\boldsymbol{D}}_{\hat{\boldsymbol{p}}}$ in descending order of confidence. If no unmatched prior detection within a radius κ is found, a new track is initialised. κ is defined as the geometric mean of the width and height of the predicted bounding box of each track.

CenterTrack uses two types of data augmentations, which are essential to successfully training the method. The first type of data augmentation concerns the class-agnostic

input heatmaps of the previous detections $\boldsymbol{H}^{t-1}$. During inference, this heatmap can contain wrongly localised objects, false positive detections, and false negative detections. These errors are not present during training but represent important failure cases of object tracking. To alleviate this problem, data augmentations are used. First, each track location $\boldsymbol{p}^{t-1}$ is locally perturbed by adding Gaussian noise to simulate localisation errors. Next, to simulate false positives, spurious locations $\boldsymbol{p}'$ are randomly added near ground truth locations and displayed in the heatmap using a 2D Gaussian as before. Finally, to simulate false negatives, ground truth detections are randomly removed.
The second type of augmentation concerns the output of the method. Object locations in the previous and current frames are highly correlated. Due to this correlation, CenterTrack could simply output the previous heatmap $\boldsymbol{H}^{t-1}$ and a displacement map $\hat{\boldsymbol{D}}^{t}$ full of zeros without resulting in a large training error. This output is not desirable, as CenterTrack would refuse to track. Therefore, an aggressive type of data augmentation is used to alleviate this problem. Specifically, both input frames $\boldsymbol{I}^{t-1}$ and $\boldsymbol{I}^{t}$ are randomly scaled and translated. Applying these transformations results in a larger object displacement between two frames, discouraging CenterTrack from repeating the previous detection. Hence, it is important to sample a separate scaling and translation for both frames since using the same transformation would leave the object's displacement unchanged. Of course, the translation and scaling have to be applied to the ground truth centre and size annotations as well.

CenterTrack is online and real-time capable with a reported runtime between 15 FPS and 28 FPS, making it well-suited for robotics applications. We extend CenterTrack to the task of multi-object 6D pose tracking while training it exclusively on synthetically generated data, contrary to CenterTrack, which relies on real images.

3.2. se(3)-TrackNet

se(3)-TrackNet [10] is an RGB-D-based method for 6D pose tracking of a single object. It is a local and online approach with a reported runtime of 90.9 Hz, which makes it ideal for robotics applications. *se(3)*-TrackNet is exclusively trained on synthetic image data, which shifts the problem from the laborious task of providing 6D pose annotations for training images to the sim-to-real problem. This chapter explains how *se(3)*-TrackNet works and how they achieve the sim-to-real transfer.
Being a local approach, *se(3)*-TrackNet operates on pairs of images, the previous frame and the current frame, instead of optimising the trajectory over the whole video at once.

Given a 3D CAD model of an object which should be tracked, the initial pose of the object, and a sequence of RGB-D images, the goal of *se(3)*-TrackNet is to predict the object's pose for all frames in the sequence. This goal is achieved by predicting the pose change between the previous frame and the current frame. Since the initial pose is given as input, the predicted pose change can be used to propagate the pose over the whole sequence of images iteratively and, this way, obtain the trajectory of the object in the video.
se(3)-TrackNet represents poses using Lie Algebra. Specifically, a change in pose $\Delta\boldsymbol{\xi}$ is locally parameterised in the tangent space as $\Delta\boldsymbol{\xi} = (\boldsymbol{t}, \boldsymbol{w})^T \in$ *se(3)*, with $\boldsymbol{t} \in \mathbb{R}^3$ and $\boldsymbol{w} \in$ *so(3)*, in a way that its exponential mapping lies in the Lie Group

$$\Delta\boldsymbol{T} = \exp(\Delta\boldsymbol{\xi}) = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \in \text{SE(3)},$$

where the rotation matrix $\boldsymbol{R}$ is obtained from $\boldsymbol{w}$ by

$$\boldsymbol{R} = \boldsymbol{I}_{3\times3} + \frac{[\boldsymbol{w}]_x}{|\boldsymbol{w}|}\sin|\boldsymbol{w}| + \frac{[\boldsymbol{w}]_x^2}{|\boldsymbol{w}|^2}(1 - \cos|\boldsymbol{w}|), \tag{3.2}$$

where $\boldsymbol{I}_{3x3}$ is the identity matrix in $\mathbb{R}^3$ and $[\boldsymbol{w}]_x$ is the skew-symmetric matrix. Given a rendering of the previous pose $\boldsymbol{T}^{t-1}$ and an image of the current frame, *se(3)*-TrackNet predicts $\boldsymbol{t}$ and $\boldsymbol{w}$. Using the exponential mapping, the change in pose $\Delta\boldsymbol{T}$ is obtained and used to propagate the previous pose to the current pose as $\boldsymbol{T}^t = \Delta\boldsymbol{T} \cdot \boldsymbol{T}^{t-1} = \exp(\Delta\boldsymbol{\xi}) \cdot \boldsymbol{T}^{t-1}$.

A data generation pipeline is proposed to train the method using only synthetic images. The authors follow the idea of domain randomisation [20] and randomise the number of objects, poses, textures, lighting, etc., to provide a high variability in the training data so the model can generalise to the real world. Previous approaches [7, 20] sample the poses randomly, which can lead to unnatural object penetration. Object penetrations are especially problematic for the depth modality because these penetrations introduce an undesired bias to the depth data distribution. Physically Plausible Domain Randomisation (PPDR) is proposed to alleviate this problem. Here, the object poses are still randomly sampled, but several physics simulation steps follow the sampling. The simulation ensures that colliding objects are separated and all objects have fallen onto the table. This procedure is used to generate images of individual, random scenes. Since *se(3)*-TrackNet operates on pairs of images, a second image, the previous frame, must be generated for every scene. They are obtained by randomly sampling a pose perturbation, where the

magnitude of $\boldsymbol{t}$ and $\boldsymbol{w}$ are separately sampled from zero-mean Gaussian distributions, and their direction is uniformly sampled. A second pose is obtained using this pose perturbation, which is then used to generate the second image. For the second image, only the tracked object is rendered without a background or other objects. Finally, both images are cropped around the tracked object and resized to a fixed resolution of 176×176 to provide a consistent input size to the neural network.

Instead of concatenating the current frame and the previous frame before feeding the data to the network, *se(3)*-TrackNet uses two input branches with separate weights. They disentangle the feature encodings of the previous frame and the current frame. Both images are synthetically generated during training, but during testing, the current frame comes from a real sensor, while the previous frame is a rendering of the previous pose estimate. This fact explains why only the tracked object is rendered in the data generation pipeline because, during inference, we usually can not reconstruct the entire scene. This way, the data distribution of the previous frame can be effortlessly aligned between training and testing without having to think about the reality gap. It also pushes the domain gap to be only present at the input branch of the current frame.
The networks' output consists of two separate output branches, one for $\boldsymbol{t}$ and one for $\boldsymbol{w}$, representing the change in pose between the previous and current frames. *se(3)*-TrackNet is trained in an end-to-end manner using an $L2$ loss

$$L = \lambda_1 ||\boldsymbol{w} - \bar{\boldsymbol{w}}||_2 + \lambda_2 ||\boldsymbol{t} - \bar{\boldsymbol{t}}||_2,$$

where $\bar{\boldsymbol{t}}$ and $\bar{\boldsymbol{w}}$ are the ground truth values and λ_1 and λ_2 are weights. Both weights are set to 1 during their experiments.

A bidirectional alignment is used to help further bridge the domain gap of the depth data. During training, two augmentation steps are applied to the depth data of the current frame. First, random Gaussian noise is added to the depth data. Second, a depth missing procedure is applied by randomly changing valid depth pixels to invalid ones. These augmentations simulate sensor noise and corrupted pixels commonly found in real depth cameras. During testing, a bilateral filter is applied to the depth image of the current frame to smooth sensor noise and fill in missing pixels. These augmentations are only applied to the images of the current frame since the domains of the depth data of the previous frame are already aligned by design.

We use the same pose parametrisation as *se(3)*-TrackNet in this thesis and follow the local approach of providing the initial pose, predicting the change to the next frame, and use the pose change to propagate the initial pose over the whole image sequence. Furthermore,

we also train the approach of this thesis exclusively on synthetic data and follow the same ideas of bridging the sim-to-real gap. But instead of only tracking a single object, we extend the approach to multi-object 6D pose tracking.

3.3. Camera calibration

Many parts of the proposed method in this thesis rely on a calibrated camera, like the data generation pipeline in Chapter 5, the data augmentation in Chapter 4.2, and the rendering of the previous frame in Chapter 4.2. The procedure of calibrating a camera and obtaining the intrinsics matrix $\boldsymbol{K}$ is explained in the following.
The intrinsics matrix $\boldsymbol{K} \in \mathbb{R}^{3\times3}$ can be obtained from the projection matrix $\boldsymbol{P} \in \mathbb{R}^{3\times4}$, which expresses the relation between 3D coordinates in the world to 2D coordinates in the image. Therefore, n points are needed for which both the 3D location $\boldsymbol{X}_i$ as well as the 2D location $\boldsymbol{x}_i$, both expressed in homogeneous coordinates, are known. Each pair of points lies on a ray since the 3D point is projected along this ray to obtain the 2D image coordinate. Therefore, the projected 3D point $\boldsymbol{P}\boldsymbol{X}_i$ and the image point $\boldsymbol{x}_i$ are collinear, which also means that the cross product between the two is zero

$$\boldsymbol{x}_i \times \boldsymbol{P}\boldsymbol{X}_i = \begin{pmatrix} y_i\boldsymbol{p}_3^T\boldsymbol{X}_i - \boldsymbol{p}_2^T\boldsymbol{X}_i \\ \boldsymbol{p}_1^T\boldsymbol{X}_i - x_i\boldsymbol{p}_3^T\boldsymbol{X}_i \\ x_i\boldsymbol{p}_2^T\boldsymbol{X}_i - y_i\boldsymbol{p}_1^T\boldsymbol{X}_i \end{pmatrix} = 0,$$

where $\boldsymbol{p}_j^T$ are the rows of $\boldsymbol{P}$ and x_i and y_i are the entries of $\boldsymbol{x}$. Rewriting this equation yields a system of linear equations

$$\begin{bmatrix} \boldsymbol{0} & -\boldsymbol{X}_i^T & y_i\boldsymbol{X}_i^T \\ \boldsymbol{X}_i^T & \boldsymbol{0} & -x_i\boldsymbol{X}_i^T \\ -y_i\boldsymbol{X}_i^T & x_i\boldsymbol{X}_i^T & \boldsymbol{0} \end{bmatrix} \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \end{pmatrix} = 0,$$

with two linearly independent equations. $\boldsymbol{P}$ has 11 degrees of freedom; therefore, $n \geq 6$ point correspondences are needed, where $n = 6$ is needed for a minimal solution while more point correspondences lead to a least-squares estimation problem. Stacking the

linearly independent equations of all the point correspondences yields a matrix $\boldsymbol{A} \in \mathbb{R}^{2n \times 9}$, resulting in the equation system $\boldsymbol{Ap} = \boldsymbol{0}$, which needs to be solved.
The trivial solution $\boldsymbol{p} = \boldsymbol{0}$ is eliminated by adding a constraint on the norm of $\boldsymbol{p}$, which results in a homogeneous least squares optimisation problem

$$\boldsymbol{Ap} = \boldsymbol{0} \quad \text{s.t.} \quad ||\boldsymbol{p}|| = 1.$$

In practice, it is solved by decomposing $\boldsymbol{A}$ using a singular value decomposition $\boldsymbol{A} = \boldsymbol{USV}^T$. $\boldsymbol{P}$ is then given by the last right singular vector $\boldsymbol{p} = \boldsymbol{v}_{12}$.
For this thesis, we are only interested in the intrinsics matrix $\boldsymbol{K}$, which needs to be extracted from the projection matrix $\boldsymbol{P}$. To extract $\boldsymbol{K}$, the projection matrix is split into a 3×3 matrix $\boldsymbol{M}$ and a 3×1 vector $\boldsymbol{m}$: $\boldsymbol{P} = [\boldsymbol{M}|\boldsymbol{m}]$. Next, $\boldsymbol{M}$ is decomposed using RQ-decomposition into an upper triangle part and an orthonormal part. The upper triangle part is the desired intrinsics matrix

$$\boldsymbol{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

which we can now use for the proposed method. f_x and f_y are the focal length along the x and y axis, respectively, and (c_x, c_y) is the principal point. We assume that no skew is present.

4. 6DCenterPose

This work aims to predict the 6D pose of all objects in a scene $\boldsymbol{T}_i^t \in \text{SE}(3), i = 1, 2, ..., N$, for N objects, at any time $t > 0$. For this, 3D CAD models of all the objects, their initial poses $\boldsymbol{T}_i^0 \in \text{SE}(3)$, and a sequence of RGB-D images $\boldsymbol{I}^\tau \in \mathbb{R}^{W \times H \times 4}, \tau = 0, 1, ..., t$, where $\boldsymbol{I}^t$ is the current observation, are required. To achieve this goal, we propose a local and online method called 6DCenterPose. As a local method, it predicts the change of pose given the observation of some timestamp t and the previous observation at timestamp $t - 1$. Since the initial pose is given, the change of pose can be used to propagate the initial pose to the next frame. This procedure is iterated for all the frames, which yields the trajectories of the objects in the image sequence.
The change of the poses is predicted by a fully convolutional neural network, whose architecture is inspired by both CenterTrack [14] and *se(3)*-TrackNet [10]. This combination allows for training exclusively on synthetic images while generalising to real images. Furthermore, a simple data association procedure based on sparse optical flow allows linking detections across frames which is necessary to apply the changes of poses to the correct corresponding objects.

4.1. Network architecture

The starting point for 6DCenterPose is a regular ResNet-18 [89], similar to *se(3)*-TrackNet, and one of the backbones used for CenterTrack. We follow CenterTrack and use heatmap-based outputs instead of directly regressing to the desired values. Using heatmaps has the advantage that the architecture of the method does not need to be changed when changing the number of tracked objects. The architecture of ResNet consists of a 7×7 convolutional layer, followed by a 3×3 max pooling layer, both with stride 2. These layers are followed by four stages of residual building blocks, where the size of the stages determines the architecture (ResNet-18, ResNet-34, etc.). Each stage halves the size of the feature map

while doubling the number of feature channels. However, since 6DCenterPose should output heatmaps with a similar size as the input images, the original image extends need to be recovered from the low-resolution feature map of the ResNet. The image extends are recovered by adding deconvolutional layers to the ResNet [90]. We follow CenterTrack and use an output stride of $R = 4$. Therefore, we add four deconvolutional layers, each with stride 2, to the architecture to recover the desired output resolution. Every convolutional layer and transposed convolutional layer is followed by a Batch Normalisation [91] layer and a ReLU activation function.

On top of this backbone structure, separate heads are added to predict the desired output maps. Each output head consists of one 3×3 convolutional layer, a ReLU activation function, and a 1×1 convolutional layer. We adopt the same output maps as CenterTrack: the centre heatmap, a bounding box size map, a location refinement regression map, and an offset map. Additionally, we add two more output maps. To predict the change of pose, we follow the parametrisation of *se(3)*-TrackNet. To recap, the change of pose $\Delta\boldsymbol{T}$ is locally parameterised in the tangent space $\Delta\boldsymbol{\xi} = (\boldsymbol{t}, \boldsymbol{w}) \in$ *se(3)*, so that its exponential mapping lies in the Lie Group $\Delta\boldsymbol{T} = \exp(\Delta\boldsymbol{\xi}) \in \mathrm{SE}(3)$. To predict the change in pose, we add a change in 3D location map $\hat{\boldsymbol{T}} \in \mathbb{R}^{W/R \times H/R \times 3}$ which is used to predict $\boldsymbol{t}$ and a change in 3D orientation map $\hat{\boldsymbol{W}}^{W/R \times H/R \times 3}$ which is used to predict $\boldsymbol{w}$. Both maps are trained using a mean squared error loss

$$L_{\text{loc}} = \frac{1}{N} \sum_{k=1}^{N} (\hat{\boldsymbol{T}}_{\boldsymbol{p}_k} - \bar{\boldsymbol{t}}_k)^2$$

and

$$L_{\text{ori}} = \frac{1}{N} \sum_{k=1}^{N} (\hat{\boldsymbol{W}}_{\boldsymbol{p}_k} - \bar{\boldsymbol{w}}_k)^2,$$

for N objects with centre locations $\boldsymbol{p}_k$, while $\hat{\boldsymbol{T}}_{\boldsymbol{p}_k}$ and $\hat{\boldsymbol{W}}_{\boldsymbol{p}_k}$ are the predicted values of the location and orientation map at location $\boldsymbol{p}_k$, respectively, and $\bar{\boldsymbol{t}}_k$ and $\bar{\boldsymbol{w}}_k$ constitute the ground truth pose changes. Following CenterTrack, supervision is only provided at the centre locations.
Together with the other losses of CenterTrack, we get the following overall training loss:

$$L = L_{\text{k}} + \lambda_{\text{size}} L_{\text{size}} + \lambda_{\text{off}} L_{\text{off}} + \lambda_{\text{dis}} L_{\text{dis}} + \lambda_{\text{loc}} L_{\text{loc}} + \lambda_{\text{ori}} L_{\text{ori}}.$$

One goal of 6DCenterPose is to be trained exclusively on synthetic images and still be able to generalise to real images. To achieve this sim-to-real transfer, we follow the approach of *se(3)*-TrackNet. Specifically, we use separate input branches for all the inputs to disentangle the respective feature encodings. As already mentioned, ResNet consists of four stages of residual building blocks. We duplicate the first of these stages and use it as the separate input branches of the method. Then, the feature representations of the input branches are concatenated and used as input to the remaining three stages.
During training, both the current and previous frames are synthetic images. During testing, the current frame is obtained from a real sensor, while the previous frame is a rendering of the previous pose predictions. This way, the domain gap reduces to the input branch of the current frame, while the input branches for the heatmap and the previous frame are effortlessly aligned.
An overview of 6DCenterPose is shown in Figure 4.1.

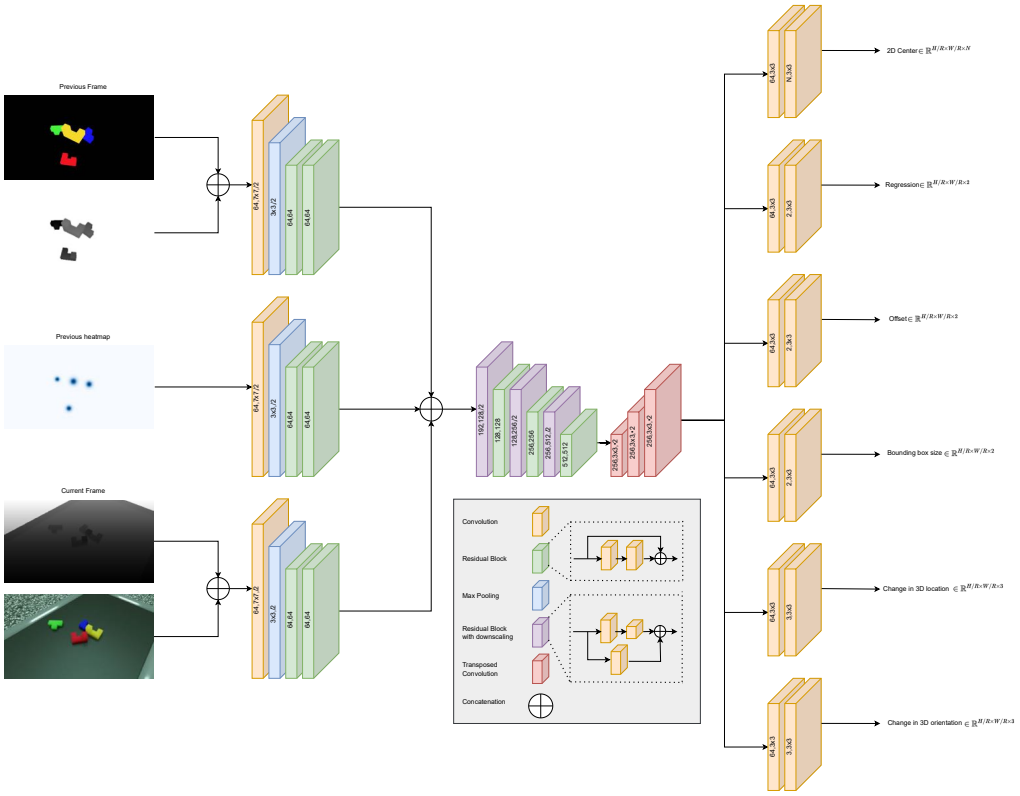


Figure 4.1.: The proposed 6DCenterPose architecture: Inputs to the method are an RGB-D image corresponding to the current observation, a rendering of all objects corresponding to the pose predictions of the previous timestamp, and a class-agnostic heatmap of the object centres of the previous timestamp. Each input has a separate feature encoder to disentangle the feature representations. The encoders' outputs are then concatenated and used to predict the outputs, i.e. a heatmap of the current centre locations, the bounding box size map, the location refinement map, the offset map, the change in 3D location map and the change in 3D orientation map. During training, both images are synthetic, while during testing, the image of the current observation is a real image.

4.2. Data augmentations

As discussed in Chapter 3.1 and Chapter 3.2, data augmentations play an essential role in training CenterTrack [14] and *se(3)*-TrackNet [10]. Since 6DCenterPose builds closely upon these methods, we also adapt their data augmentations.

Bidirectional alignment of depth data

To bridge the sim-to-real gap of the depth modality, we follow *se(3)*-TrackNet [10] and adopt a bidirectional alignment of the depth data. During training, random Gaussian noise is added to the pixels with a valid depth value of the depth map of the current frame. This augmentation is followed by applying a depth missing procedure to the depth map of the current frame by randomly changing pixels with valid values to pixels with invalid values. This augmentation simulates sensor noise and corrupted pixels usually found in commercially-available depth sensors.
During testing, a bilateral filter is applied to the depth map of the current frame to smooth sensor noise and fill in missing values of the depth map. This augmentation helps to align the real measurements with the synthetic domain.
These augmentations are only applied to the current frame's depth map since the previous frame's depth maps are aligned by design, as the values are synthetic both during training and testing.

Affine transformations

As discussed in Chapter 3.1, scaling and translating the input images independently is essential to train CenterTrack to discourage the method from simply repeating the previous detection. Since 6DCenterPose builds upon CenterTrack, we also need to apply these transformations. Furthermore, 6DCenterPose predicts a rotation component, the change of orientation of the objects, in addition to the other translation-based outputs (including the change of 3D location). Because of this rotation component, we add a rotation augmentation to encourage 6DCenterPose to make rotation predictions other than zero.
When applying affine transformations to an image, like scaling, translating, and rotating, it is also important to transform the annotations of the image. Transforming 2D annotations like the object centres and bounding box sizes is quite straightforward, but it is a bit more involved for 3D annotations like the 6D poses. The procedure to apply an affine

transformation to a 6D pose is explained in the following.
A general affine transformation of scaling, translating and rotating in homogeneous coordinates is given by

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix}.$$

Here, a_3 and a_6 correspond to a translation in x and y direction, respectively. The norm along the first and second column corresponds to the scaling along the x and y direction, i.e. $s_x = \sqrt{a_1^2 + a_4^2}$ and $s_y = \sqrt{a_2^2 + a_5^2}$. We assume equal scaling along both image axes, i.e. $s := s_x = s_y$. The rotation $\boldsymbol{R}$ of the image is recovered by the following expression

$$\boldsymbol{R} = \begin{pmatrix} \frac{a_1}{s_x} & \frac{a_2}{s_y} \\ \frac{a_4}{s_x} & \frac{a_5}{s_y} \end{pmatrix} = \frac{1}{s} \begin{pmatrix} a_1 & a_2 \\ a_4 & a_5 \end{pmatrix}.$$

Given a pose annotation $\boldsymbol{T}$ with an orientation part $\boldsymbol{R_T} \in \mathbb{R}^{3\times3}$ and a translation part $\boldsymbol{t} \in \mathbb{R}^3$, the orientation part and the translation part are treated separately.
The rotation of the affine transformation $\boldsymbol{R}$ is applied in image coordinates, but the object's orientation is given in the 3D coordinate frame of the camera. The z-axis of the camera coordinate system corresponds to the principal axis, which is the axis around which the rotation of the affine transformation is applied. The rotation of the affine transformation, therefore, corresponds to a rotation around the z-axis in the camera coordinate system

$$\boldsymbol{R}_z = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{pmatrix} = \begin{pmatrix} \frac{a_1}{s_x} & \frac{a_2}{s_y} & 0 \\ \frac{a_4}{s_x} & \frac{a_5}{s_y} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The orientation part of the pose after the affine transformation is then $\boldsymbol{R'_T} = \boldsymbol{R}_z \cdot \boldsymbol{R_T}$
To apply the affine transformation to the translation part $\boldsymbol{t}$ of the pose, the intrinsics matrix of the camera $\boldsymbol{K}$ is needed. The first step is to project the 3D location of the object to image coordinates using the intrinsics matrix $\boldsymbol{K}$

$$\boldsymbol{t}_{\text{img}} = \boldsymbol{K} \cdot \boldsymbol{t} = \begin{pmatrix} x_{\text{img}} \\ y_{\text{img}} \end{pmatrix}.$$

In order to apply the affine transformation $\boldsymbol{A}$, $\boldsymbol{t}_{\text{img}}$ needs to be transformed to homogeneous coordinates first resulting in $\hat{\boldsymbol{t}}_{\text{img}} = (x_{\text{img}}\ y_{\text{img}} 1)^T$. Then, the affine transformation is applied

$$\hat{\boldsymbol{t}}'_{\text{img}} = \boldsymbol{A} \cdot \hat{\boldsymbol{t}}_{\text{img}} = \begin{pmatrix} x'_{\text{img}} \\ y'_{\text{img}} \\ 1 \end{pmatrix}$$

Finally, the new image point needs to be projected back to the 3D camera coordinate system to obtain the new 3D location of the object. Since all points along one ray project to the same point on the image, i.e. the depth is lost, the distance of the new 3D point to the camera is required. We obtain it based on the distance z of the original pose.
Given the scaling factor s of the affine transformation $\boldsymbol{A}$, we define the distance of the new 3D point as $z' = z/s$. With this value and the inverse of the camera intrinsics matrix $\boldsymbol{K}^{-1}$, we obtain the new location of the object relative to the camera

$$\boldsymbol{t}' = \boldsymbol{K}^{-1} \cdot (z' \cdot \hat{\boldsymbol{t}}'_{\text{img}}).$$

The resulting pose after the affine transformation is then given as

$$\boldsymbol{T}' = \begin{pmatrix} \boldsymbol{R}' & \boldsymbol{t}' \\ \boldsymbol{0} & 1 \end{pmatrix},$$

which is used as the new ground truth pose for the given object. This process is repeated for all objects in the scene, resulting in the augmented pose annotations for the given image.

5. Synthetic data generation

Annotating training images for the 3D domain is inherently more challenging than annotating images with 2D labels since, in the latter case, annotations only have to be provided in pixel coordinates. In contrast, in the former case, the annotations have to be provided in metric coordinates relative to the camera. To alleviate this problem, approaches like *se(3)*-TrackNet [10] train their method exclusively on synthetically generated images. This way, the 6D poses are readily available since they need to be known to render the images. Using synthetic images shifts the problem of annotating training images to the sim-to-real problem of aligning the synthetic domain to the real world.
We follow this approach and use synthetic images to train the method in this thesis. To this end, we created a synthetic data generation pipeline using BlenderProc [92], which serves as an interface to the 3D modelling software *Blender*[1]. Chapter 5.1 describes the process of generating the synthetic training images. In addition to generating random images, we generate random trajectories and render image sequences of objects following these trajectories. These sequences can then be used as test data to evaluate the in-distribution performance of the proposed method. This procedure is described in Chapter 5.2. CAD models for all the rendered objects are created using *Blender*.

5.1. Training data generation

The data generation pipeline aims to generate realistic-looking scenes. Therefore, it is not enough to place the tracked objects in an empty space. The basis for the synthetic scene is a table which is modelled in a way to resemble the real table used in the experiments (see Chapter 6). The tabletop size for our experiments is 66 cm by 100 cm. A cube is placed around the table to resemble a room for several purposes. First, the upper part of the cube, i.e. the ceiling, is set to emit light and illuminate the scene. The emission strength, as well

[1] `https://www.blender.org/`

as the colour of the light, are randomised for the individual images. Second, the room's walls and floor are used to display textures, which are randomised for individual images. Finally, adding a room is very important to render depth images. Without the room, the depth values besides the objects and the table are infinite, or rather the maximal value of the data type used to save the depth image, while in real scenes, especially in indoor scenarios, the depth values rarely reach the maximal values of the used data type.
Now that the scene is set up, the objects can be placed in it. For this purpose, random locations are uniformly sampled. In x and y directions, the sample space is limited by the tabletop. The z direction, i.e. the height, is sampled between 0 cm and 25 cm above the tabletop. In addition to sampling the poses, a collision check is performed to ensure unrealistic object penetration is avoided. Avoiding object penetrations is especially important for the depth modality since they can not happen in the real world and would introduce undesired bias to the depth data [10]. Finally, a physics simulation is used to ensure object separation further and let the objects fall onto the table.
Finally, the 3D scenes have to be converted to 2D images. For this reason, a camera is placed in the scene. The camera's location is uniformly sampled in a range of -75 cm to 75 cm in x and y direction and 6 cm to 100 cm above the tabletop in z direction. After placing the camera, it is oriented to look at the centre of the tabletop. Using this setup, we can generate an RGB-D image for every scene and get the pose annotations for all the objects.

We now have synthetic images of random scenes, but the proposed approach is trained on pairs of images. So what remains to be done is to generate a second image for every synthetic image. To generate them, a pose perturbation for every object in the scene is sampled. Let $\boldsymbol{T}^{\tau}$ be the pose annotation for one object in the current frame τ. The pose perturbation $\boldsymbol{T}_{\tau}^{\tau-1}$ consists of a translation component $\boldsymbol{t}$ and a rotation component $\boldsymbol{R}$. The direction of the translation is uniformly sampled while the distance of the translation, i.e. its norm, follows a Gaussian distribution $|\boldsymbol{t}| \sim |\mathcal{N}(0, \sigma_t)|$. For the rotation, we follow *se(3)*-TrackNet [10] and use a Lie Algebra representation $\boldsymbol{w} \in$ *so(3)*. The direction of $\boldsymbol{w}$ is again uniformly sampled and the norm of $\boldsymbol{w}$ follows a Gaussian distribution $|\boldsymbol{w}| \sim |\mathcal{N}(0, \sigma_w)|$. The rotation matrix $\boldsymbol{R}$ is then obtained from $\boldsymbol{w}$ using Equation 3.2. In our case, we set σ_t to 2 cm and σ_w to 0.5236 rad ($\widehat{=}\ 30°$). Using $\boldsymbol{R}$ and $\boldsymbol{t}$, we can construct the pose perturbation $\boldsymbol{T}_{\tau}^{\tau-1}$, with which we can determine the previous pose $\boldsymbol{T}^{\tau-1} = \boldsymbol{T}_{\tau}^{\tau-1}\boldsymbol{T}^{\tau}$.

To generate the second image of the previous frame, a perturbed pose for all the objects in the scene is sampled. Again, a collision check is performed to ensure no object penetrations occur. If a collision is detected, a new perturbed pose is sampled, and this process is repeated until all objects are placed. In contrast to the current frame, the previous frame

is rendered using the python package *pyrender*[2]. It is used because, during inference, the poses for the previous frame must also be rendered. While *Blender* generally produces higher-quality images, the rendering time is also usually higher. Since a low runtime during inference is crucial, the lightweight package *pyrender* is used. Another difference between the current frame and the previous frame is that only the objects that the proposed method should track are rendered for the previous frame. For the training data, we could easily render the whole scene as well, but we are usually unable to reconstruct the entire scene during inference. By only rendering the tracked objects, the domain of the previous frame is exactly the same during inference and training, pushing the domain gap to be only present for the current frame [10].

After generating a second frame for every scene, the synthetic training dataset is complete and is used to train the proposed method. Figure 5.1 shows an example of a generated image pair.

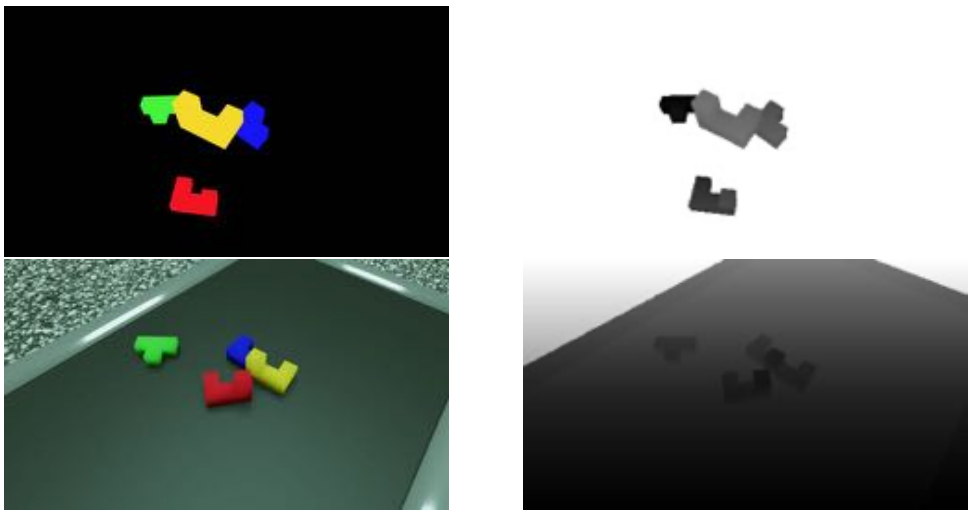


Figure 5.1.: An example training image pair generated using the data generation pipeline. The lower row shows the RGB and depth images which are generated using BlenderProc. Based on these poses, perturbed poses are sampled, which are used to render the RGB and depth image in the upper row using *pyrender*.

[2] `https://pyrender.readthedocs.io/`

5.2. Test data generation

After training the proposed method, its performance needs to be evaluated. In addition to testing on real trajectories (see Chapter 6.2), we use the data generation pipeline to render synthetic trajectories. The benefit of using synthetic trajectories is that annotating real trajectories is labour-intensive and prone to errors. For synthetic trajectories, the pose annotations are readily available and also very precise. While the performance on real data is more important for real-world applications, the performance on synthetic data is a good indicator if the method works in the first place.

The general setup is very similar to the training data. Again, we use a table as the base of the scene and place a cube around it to simulate a room. The textures of the room are randomised, but instead of using a different texture for every frame as for the training data, the textures are kept for the whole trajectory. The illumination conditions are also randomised and kept for the entire trajectory. A fixed camera pose is used to ensure that the whole table is visible and the trajectories of the objects can be captured.
After setting up the scene, the objects need to be placed in it. For every synthetic test sequence, the number of moving objects is specified. If the number of moving objects is lower than the total number of objects in the scene, the rest of the objects will be stationary (except in the case of collisions, see below). The objects that are moving are chosen randomly.

For each of the moving objects, a trajectory needs to be generated. Here, a trajectory is a smooth, continuous curve through 3D space combined with a smooth, continuous rotation of the object. Videos are a discretisation of continuous visual information; therefore, we also need a discrete trajectory. A discrete trajectory $\mathcal{T}$ is a series of poses $\mathcal{T} = [\boldsymbol{T}^0, \boldsymbol{T}^1, ..., \boldsymbol{T}^{t-1}]$ for t time steps. Our first idea was to sample two random poses $\boldsymbol{T}^0$ and $\boldsymbol{T}^{t-1}$ and linearly interpolate between them. This does generate a smooth, continuous curve, but the difference between the individual poses is, by definition, always the same. Since the proposed method predicts these differences in poses, having the same difference for every frame pair is not ideal, as the ground truth value for all the frame pairs would be the same. Instead, we use Bézier curves, which are also smooth and continuous curves, and allow for different changes between poses throughout the trajectory.

The basis of Bézier curves are Bernstein polynomials. A Bernstein polynomial of degree n is defined as

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i},$$

where $i = 0, 1, ..., n$, $n \in \mathbb{N}$, and $t \in [0, 1] \subset \mathbb{R}$. The Bernstein polynomials are then used to define a basis of the vector space of polynomials Π_n^m of at most degree n in $\mathbb{R}^m$. The polynomials using the Bernstein basis are called Bézier curves and are defined as

$$\boldsymbol{p}(t) = \sum_{i=0}^{n} \boldsymbol{b}_i B_i^n(t), \tag{5.1}$$

where again $i = 0, 1, ..., n$, $n \in \mathbb{N}$ and $t \in [0, 1] \subset \mathbb{R}$. $\boldsymbol{b}_i \in \mathbb{R}^m$ are called Bézier points or control points. The start point and end point of the Bézier curve coincide with the first and last control point, i.e. $\boldsymbol{p}(0) = \boldsymbol{b}_0$ and $\boldsymbol{p}(1) = \boldsymbol{b}_n$, while the rest of the curve lies in the convex hull of the control points.
We can now use the Bézier curves to define a trajectory. First of all, we need to decide how many control points are used. Using two control points results in a linear interpolation between the two points, while the trajectory becomes more intricate by using more control points. For our experiments, we use four control points, i.e. $n = 3$. Next, we have to decide on the number of frames N_{frames} the synthetic trajectory should have. This choice does not influence the shape of the trajectory, but it influences how much the pose changes between frames. For our experiments, we set $N_{frames} = 150$. To generate the trajectories, we treat the location component and the rotation component separately. For the location component, we uniformly sample $n + 1$ locations $\boldsymbol{l}_i \in \mathbb{R}^3, i = 0, 1, ..., n$. In x and y direction, the sample space is limited by the tabletop size, similar to the training data generation. The range of the z direction is set to be between 0 cm and 25 cm above the tabletop. The location part of the trajectory is then given by Equation 5.1 with the sampled locations $\boldsymbol{l}_i$ as control points

$$\boldsymbol{l}(t) = \sum_{i=0}^{n} \boldsymbol{l}_i B_i^n(t),$$

with $t = j/(N_{\text{frames}} - 1)$, $j = 0, 1, ..., N_{\text{frames}} - 2$.
We decided to use a quaternion representation for the rotation component due to their favourable interpolation properties [93]. We sample $n+1$ unit quaternions $\boldsymbol{q}_i, i = 0, 1, ..., n$, where each quaternion is represented with four real numbers $\boldsymbol{q}_i = (q_{0,i}, q_{1,i}, q_{2,i}, q_{3,i})$

and the norm of the quaternion is $|\boldsymbol{q}_i| = 1$. The rotation matrix for a unit quaternion $\boldsymbol{q} = (q_0, q_1, q_2, q_3)$ can be recovered using

$$\boldsymbol{R}_{\boldsymbol{q}} = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & -2q_0q_3 + 2q_1q_2 & 2q_0q_2 + 2q_1q_3 \\ 2q_0q_3 + 2q_1q_2 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & -2q_0q_1 + 2q_2q_3 \\ -2q_0q_2 + 2q_1q_3 & 2q_0q_1 + 2q_2q_3 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix}$$

Again, using Equation 5.1 with the sampled quaternions $\boldsymbol{q}_i$ as control points, we get the rotation part of the trajectory

$$\boldsymbol{q}(t) = \sum_{i=0}^{n} \boldsymbol{q}_i B_i^n(t),$$

with $t = j/(N_{\text{frames}} - 1)$, $j = 0, 1, ..., N_{\text{frames}} - 2$.
The full pose for every frame t is then given by

$$\boldsymbol{T}^t = \begin{pmatrix} \boldsymbol{R}_{\boldsymbol{q}(t)} & \boldsymbol{l}(t) \\ \boldsymbol{0} & 1 \end{pmatrix} \in \text{SE(3)},$$

resulting in a complete trajectory for the moving objects. For the stationary objects, a random pose is sampled similarly to the moving objects and kept unchanged for all the frames.
So far, the trajectories and stationary poses are generated completely independently. As already discussed, these poses may lead to unnatural object penetrations, which must be resolved. But contrary to the training data generation, we can not simply sample a new pose in the event of a collision as this would violate the continuity and smoothness requirements of the trajectories and sampling a completely new trajectory would be very inefficient. Instead, we calculate the distance vectors between all pose pairs for every frame. The orientation component does not matter here, and we only focus on the location part. Given two locations $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$, the distance vector between both locations is given as $\boldsymbol{d} = \boldsymbol{l}_2 - \boldsymbol{l}_1$. The norm of this vector is used for the collision check: If $||\boldsymbol{d}|| < d_{\text{coll}}$ for some collision distance d_{coll}, a collision is detected and needs to be resolved. Else, no collision is detected, and the poses can be kept. To determine the collision distance, we calculate the diameter of all the objects in the scene and take the biggest one d_{obj}. To allow for a smoother collision resolve, we define the collision distance as $d_{\text{coll}} = 2.5 d_{\text{obj}}$.

To resolve the collision, we calculate how far the two poses overlap, which is given by $r = d_{coll} - ||\boldsymbol{d}||$. Then, we move both locations by half of this distance along the distance vector:

$$\boldsymbol{l}_{1,\text{resolved}} = \boldsymbol{l}_1 - r\frac{\boldsymbol{d}}{||d||}$$
$$\boldsymbol{l}_{2,\text{resolved}} = \boldsymbol{l}_2 + r\frac{\boldsymbol{d}}{||d||}$$

This shift resolves the collision between the two poses, but the collision resolve can lead to collisions between the two objects and other objects in the scene (including the table). Therefore, the collision check has to be iterated until all collisions are resolved.
We now have collision-free, smooth, and continuous trajectories for all the objects in the scene. They are rendered using BlenderProc and are used to evaluate the proposed method in Chapter 6. Contrary to the training data, there is no need to generate a second, previous frame for every generated frame as they are generated during inference based on the pose predictions of the proposed method.

6. Experiments

This chapter explains the experimental setup used in this thesis. Chapter 6.1 explains how to calibrate the camera used for the experiments. Chapter 6.2 explains the specific task which is solved in this thesis. That chapter also lays out the metrics used to evaluate the method and how real trajectories are annotated for the purpose of evaluating the proposed method. Chapter 6.3 explains implementation details. Finally, Chapter 6.4 compares 6DCenterPose with a baseline and discusses the results of the experiments.

6.1. Camera calibration

Both the data generation (see Chapter 5) and parts of the proposed method (see Chapter 4) require the intrinsics matrix $\boldsymbol{K}$ of the camera used for the experiments. To follow the described procedure to calibrate a camera in Chapter 3.3, point correspondences between 3D and 2D are needed. To obtain them, we build a camera calibration board shown in Figure 6.1, which contains several black dots. To obtain the 3D coordinates, a coordinate origin is defined on the calibration board and the locations of all the black dots relative to the origin are carefully measured.

Next, the 2D coordinates of the black points are needed. We use an *Azure Kinect* RGB-D camera for all our experiments and use it to capture the image shown in Figure 6.1. From it, the pixel coordinates of the black dots are determined. The resolution of the recorded image is 1920×1080, but we use a resolution of 960×540 to allow for a faster generation of images in the data generation pipeline and to follow the input resolutions used by CenterTrack [14]. Therefore, the obtained 2D locations are divided by 2 to account for this downscaling.

Now that we have 2D-3D point correspondences, we use the procedure described in Chapter 3.3 to obtain the following intrinsics matrix:

$$\boldsymbol{K} = \begin{pmatrix} 524.79512479 & -2.04110125 & 520.71537408 \\ 0 & 541.88587573 & 242.56187974 \\ 0 & 0 & 1 \end{pmatrix}.$$

Due to inaccuracies in the measurements and properties of the camera, we receive a non-zero skew component, which we set to zero for the experiments.

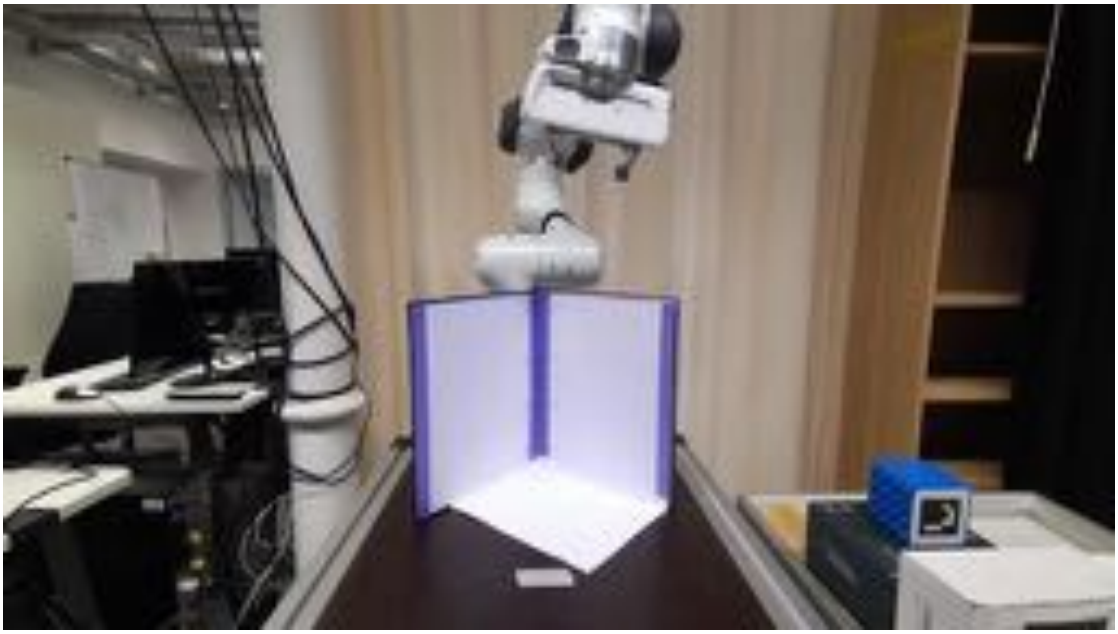


Figure 6.1.: The image used for calibrating the camera. The locations of the black dots on the calibration board are measured in 3D metric coordinates relative to a corner of the calibration board and in 2D pixel coordinates to establish a 2D-3D correspondence. These correspondences are used to obtain the projection matrix $\boldsymbol{P}$ from which the intrinsics matrix $\boldsymbol{K}$ is obtained.

6.2. Datasets

The task we want to solve during this thesis is the board game *Ubongo3D* [1]. The game consists of 16 differently shaped blocks in the four colours red, green, blue, and yellow. Additionally, there are cards displaying an outline. The objective of the game is to stack some given blocks on the outline so that two layers are completely covered. The fastest player wins the round, and the same procedure is repeated with new outline cards.
We decided to work with this task because it is an excellent surrogate task for robot manipulation. Placing the first block represents a pick-and-place task, while placing the subsequent blocks represents an insertion task. The insertion task has the additional challenge that the current block has to be placed so that the blocks already placed are not toppled over, undoing the current progress.

To train 6DCenterPose, we use the data generation pipeline described in Chapter 5.1 to generate 200k image pairs used for training. Additionally, 20k image pairs are generated for validation during training. Instead of training on all 16 Ubongo3D shapes, we use four shapes, one of each colour, to train and evaluate our method. The generation of the 220k images took around one day.

After training the method, its performance needs to be evaluated. We follow the evaluation procedure used in previous 6D pose estimation methods [7, 10] based on the metrics ADD, which performs exact model matching and ADD-S, which is designed to evaluate symmetric models. They are defined as [65]

$$\text{ADD} = \frac{1}{m} \sum_{\boldsymbol{x} \in M} ||\boldsymbol{R}\boldsymbol{x} + \boldsymbol{t} - (\hat{\boldsymbol{R}}\boldsymbol{x} + \hat{\boldsymbol{t}})||$$

$$\text{ADD-S} = \frac{1}{m} \sum_{\boldsymbol{x}_1 \in M} \min_{\boldsymbol{x}_2 \in M} ||\boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{t} - (\hat{\boldsymbol{R}}\boldsymbol{x}_2 + \hat{\boldsymbol{t}})||,$$

where M denotes the set of 3D model points and m is the number of points. $\boldsymbol{t}$ and $\boldsymbol{R}$ are the ground truth translation and rotation, respectively, while $\hat{\boldsymbol{t}}$ and $\hat{\boldsymbol{R}}$ are the estimated translation and rotation, respectively. A predicted pose is considered correct if the value of the metric is smaller than some threshold. The threshold is varied between 0 cm and a maximal threshold, 10 cm for our experiments, following previous approaches [7, 10].

[1] `https://www.kosmos.de/spielware/spiele/familienspiele/7333/ubongo-3-d`

This change of threshold results in a threshold-accuracy curve of which the area under the curve is reported. This approach allows to represent the performance of the whole method using a single value (per metric).

To use these metrics to evaluate the accuracy of 6DCenterPose, we need annotated test videos on which predictions are made. To evaluate the in-distribution performance, we use the proposed data generation pipeline to generate synthetic test videos where the annotations of all the objects in the scene are given (see Chapter 5.2). In total, we generate 100 test videos. Each video contains all four Ubongo shapes the method was trained on. One random object is moving in 50 videos, two random objects are moving in 25 videos, three random objects are moving in 15 videos, and all four objects are moving in 10 videos.

While the evaluation on synthetic trajectories is important to prove the method is working in the first place, the main objective of 6DCenterPose is to be used in real scenarios. To evaluate this out-of-distribution performance and the sim-to-real transfer, we record real videos using an *Azure Kinect* RGB-D camera statically mounted to the table of the experimental setup. A 3D printer is used to print three of the four Ubongo3D shapes used for training so they can be placed in the real scene. Four videos are recorded - three videos where only one of the three shapes is moved and one video where two of the three shapes are moved. The movement of the shapes is done by a human demonstrator.

To obtain the 6D poses of the objects, we place ArUco markers [94] on the shapes. After recording the video, the markers' pose is obtained for every frame independently. However, several post-processing steps are needed to obtain the poses of the objects.
First of all, filters are applied to smooth the recorded trajectories. Furthermore, the ArUco markers may not be detected in every frame. To still get a pose annotation for every frame, we interpolate the poses for the frames that do not have an annotation to fill these holes.

These post-processing steps result in pose annotations for all the markers in the scene. However, the origin of the marker usually does not coincide with the origin of the 3D CAD model of the object the marker is placed on. Since we are interested in the trajectory of the object and not the trajectory of the markers, we determine the offset between the position of the marker on the object and the origin of the object. This offset is added to the recorded trajectories to obtain the trajectories of the objects.

Thus, we now have real annotated videos to evaluate the sim-to-real performance of 6DCenterPose. Figure 6.2 shows the experimental setup used to record the videos. Figure 6.7 and Appendix A show poses obtained from the ArUco markers.

Figure 6.2.: The experimental setup used for the recording of real videos. The Ubongo3D shapes are used, and an ArUco marker is placed on each of them to recover the ground truth pose.

6.3. Implementation details

The implementation of 6DCenterPose is based on CenterTrack [14]. We train 6DCenterPose using the Adam [95] optimiser with a learning rate of $1.25e-4$ and a batch size of 32. All weights are randomly initialised, and we use no pre-training to evaluate the sim-to-real performance. Pre-training on a real image dataset would implicitly inform the method about the data domain of the real world. The weights of the loss function are set to $\lambda_{\text{size}} = 0.1$, $\lambda_{\text{off}} = 1$, $\lambda_{\text{dis}} = 1$, $\lambda_{\text{loc}} = 1$, and $\lambda_{\text{ori}} = 10$ for all our experiments. The method is trained for 100 epochs, and the learning rate is divided by a factor of 10 after 60 epochs. The training is performed on a *Nvidia Quadro RTX 8000* GPU and a *AMD Ryzen Threadripper 3990X* CPU and took around 12 days.

We follow CenterTrack [14] and use an input resolution of 544×960. The synthetic images are rendered in a resolution of 540×960 and then resized to fit the input resolution. The videos from the RGB-D camera are recorded in a resolution of 3840×2160 at 30 FPS, so

the ArUco markers are well visible. After recording, they are downsampled and resized to fit the input resolution of the 6DCenterPose.

Data augmentations include the already discussed affine transformations and the depth missing procedure. The affine transformation consists of a scaling factor which is randomly sampled between 0.6 and 1.4, a translation randomly sampled in a way that the new image centre is somewhere in the image extends, leaving a border of 128 pixels to the image edges, and a rotation randomly sampled following a normal distribution with zero mean and a standard deviation of 7. The rotation is then clipped to a range of $-14°$ to $14°$. The depth missing procedure is applied with a probability of 30%, and up to 20% of pixels are turned from valid to invalid. Depth values in the range of 1 cm to 200 cm are considered valid, while values outside this range are considered invalid.

Additionally, we apply HSV jitter, Gaussian blur and Gaussian noise to the RGB image of the current frame. Again, these augmentations are not applied to the images of the precious frame as they are rendered both during training and testing and, by this, follow the same data distribution.

Centre locations are perturbed in x and y direction following a normal distribution with zero mean and a standard deviation of 5% of the width and height of the corresponding bounding box, respectively. A ground truth centre location is removed with a probability of 40% (false negative rate), while a new centre near an existing one is added with a probability of 10% (false positive rate).

Following *se(3)*-TrackNet, we normalise the ground truth labels of the change in location and of the change in orientation by dividing them by the maximal translation (2 cm) and by the maximal rotation ($0.5236 \mathrel{\widehat{=}} 30°$) so the values lie in the range of -1 and 1. Note that these are only the maximal values used to render the previous image given a synthetic image. Due to the affine transformation, values with an absolute value bigger than 1 are possible.

During inference, only detections with a confidence score bigger than 0.1 are added to the heatmap used as input to 6DCenterTrack. Only detections with a confidence score bigger than 0.1 are returned as output. In case the tracking of an object is lost, the corresponding track is kept alive for up to 3 frames before it is terminated. During this time, the last prediction for the changes in pose is used to obtain a new pose. Currently, no initialisation of new tracks during the video is supported since 6DCenterPose can only predict changes in pose, not absolute poses. Therefore, the initialisation of a new track requires the absolute pose of the object for the frame in question, which can be either obtained from ground truth values or a 6D pose estimation method.

To initialise the tracking pipeline, the initial poses of all objects are needed. For the synthetic trajectories, the ground truth poses used to render the images are used. For the real trajectories, the ground truth poses obtained from the ArUco markers are used.

6.4. Results

6.4.1. Baseline

To assess if 6DCenterPose works well, it needs to be compared to another approach. As a baseline, we use *se(3)*-TrackNet [10] since it inspired the proposed approach and the source code is publicly available. Since *se(3)*-TrackNet is only designed as a single-object tracking method, we need to extend it to the task of multi-object tracking. This extension is achieved by training four separate *se(3)*-TrackNets, one for each object also used to train 6DCenterPose. For every frame of the test videos, we iterate over all the *se(3)*-TrackNets to get a pose prediction of all the objects in the scenes. This gives *se(3)*-TrackNet an advantage, as each method is specifically trained for each object.
The *se(3)*-TrackNets are trained using the same training images used to train 6DCenterPose, and the training follows the procedure outlined in the corresponding publication for *se(3)*-TrackNet. To initialise the tracking pipelines, *se(3)*-TrackNet uses the same ground truth poses as 6DCenterPose.

6.4.2. Synthetic trajectories

Table 6.1 and Figure 6.5 display the results of *se(3)*-TrackNet and 6DCenterPose, evaluated on the synthetic trajectories.
6DCenterPose achieves 32.1% on the ADD metric and 47.7% on the ADD-S metric. The higher score on the ADD-S metric is because the Ubongo shapes consist of symmetric cubes. This difference between ADD and ADD-S suggests that 6DCenterTrack can predict the location part of the 6D pose while struggling with accurately predicting the orientation.
In contrast, *se(3)*-TrackNet performs excellently on the synthetic trajectories, achieving 96.6% on the ADD metric and 97.0% on the ADD-S metric, which is in line with the results reported in the publication. There is no notable difference between the ADD and ADD-S metrics, suggesting that *se(3)*-TrackNet can accurately predict the location and orientation of the objects. However, as already mentioned, *se(3)*-TrackNet predicts the poses for each object separately, using specifically trained methods for the individual objects. In contrast,

	se(3)-TrackNet [10]		6DCenterPose	
Objects	ADD	ADD-S	ADD	ADD-S
02_ubongo_yellow	96.5	96.8	30.5	48.1
08_ubongo_blue	97.7	98.1	33.0	45.8
11_ubongo_red	96.2	96.4	33.5	50.7
15_ubongo_green	96.0	96.6	31.1	46.3
ALL	96.6	97.0	32.1	47.7
Speed (FPS)	8.1		12.8	

Table 6.1.: Results of 6DCenterPose compared to *se(3)*-TrackNet evaluated on the synthetically generated trajectories. *se(3)*-TrackNet tracks each object separately while 6DCenterPose tracks all objects simultaneously. Therefore, the speed of *se(3)*-TrackNet is obtained by adding the individual runtimes and dividing the number of frames by it.

6DCenterPose predicts all object poses with a single pass through the networks. Using specified approaches for each object results in a big advantage for *se(3)*-TrackNet, which explains the gap in performance between the two approaches.

To analyse the predictions made by the approaches in more detail, Figure 6.3 displays the absolute differences between the predictions and the ground truth values. The graphs display the mean and standard deviation over all 100 synthetic trajectories and all 4 Ubongo shapes, separated into the three spatial dimensions and the four dimensions of the quaternion representations of the orientations.

These results further confirm that 6DCenterPose struggles with accurately predicting the orientation of the objects, indicated by the large means and standard deviations of the four quaternion components, compared to *se(3)*-TrackNet. Furthermore, all three spatial dimensions display a drift during the trajectory, indicated by the increasing means and standard deviations for increasing frame numbers. Finally, Figure 6.3 indicates that 6DCenterPose has problems accurately predicting the object's depth, displayed by the absolute error in z, compared to the absolute error in x and y. Note the larger scale of the y-axis in the plot for the z direction compared to the x and y direction.

se(3)-TrackNet displays a slight drift in x-direction, but, all in all, the absolute errors for all dimensions are small, as reflected by the metrics in Table 6.1.

Table 6.2 reports the tracking losses of 6DCenterPose, detailing which object is lost in which frame. 6DCenterPose currently does not allow the reinitialisation of tracks. Therefore, a lost track results in lower values in the metrics, as a lost track can be interpreted as a value higher than the maximal threshold. The main reason for tracking loss in the synthetic trajectories is occlusions, which regularly occur due to the random nature of the synthetic trajectories. *se(3)*-TrackNet does not experience any tracking loss and is robust to occlusions.

Object	Frame of track loss
02_ubongo_yellow	3
11_ubongo_red	3
08_ubongo_blue	4
15_ubongo_green	4
15_ubongo_green	22
11_ubongo_red	50
02_ubongo_yellow	94
02_ubongo_yellow	126
15_ubongo_green	144

Table 6.2.: Tracking loss of 6DCenterPose during the synthetic trajectories.

Table 6.1 displays the inference speed of the evaluated methods. The value for *se(3)*-TrackNet is obtained by adding the runtime of the individual methods for all objects in the scene and dividing the number of frames by it. In practice, given enough computational power, the individual *se(3)*-TrackNets can be executed in parallel to increase the speed, which was beyond the scope of this thesis. The runtime for both approaches includes all steps of the pipeline, including the rendering of the pose prediction of the previous frame, the loading of the current frame from memory, as well as the pass of the data through the networks. 6DCenterPose achieves a competitive runtime to *se(3)*-TrackNet, being near real-time, which makes it applicable for robotics applications.

Figure 6.6 shows qualitative results of 6DCenterPose and *se(3)*-TrackNet for one of the synthetic trajectories. Further qualitative results, as well as absolute error graphs for the individual objects, are shown in Appendix A.

6.4.3. Real trajectories

Table 6.3 and Figure 6.5 present the results of 6DCenterPose and *se(3)*-TrackNet, evaluated on the real trajectories.

	se(3)-TrackNet [10]		6DCenterPose	
Objects	ADD	ADD-S	ADD	ADD-S
08_ubongo_blue	61.3	73.3	40.2	60.5
11_ubongo_red	49.3	64.8	19.5	30.7
15_ubongo_green	72.3	80.1	19.3	41.5
ALL	61.0	72.7	26.3	44.2

Table 6.3.: Performance of 6DCenterPose compared to *se(3)*-TrackNet evaluated on the real trajectories.

6DCenterPose achieves 26.3% on the ADD metric and 44.2% on the ADD-S metric. Again, the score for ADD-S is higher, indicating that 6DCenterPose is better at predicting the location than predicting the orientation of the objects. In contrast to the synthetic trajectories, the scores of the individual objects are not as uniform. Compared to the other shapes, the score for the blue shape is almost twice as high. This is most likely because the blue shape is the simplest and smallest among the evaluated shapes. One reason for the low score of the red shape is the fact that the track for it is lost in two of the four trajectories, as shown in Table 6.4. However, the green shape does not experience tracking loss, but the scores are similar to the red shape. This indicates that 6DCenterPose struggles with more complex shapes.
se(3)-TrackNet achieves 61.0% on the ADD metric and 72.7% on the ADD-S metric, again outperforming 6DCenterPose. For the real trajectories, the difference between the ADD and ADD-S metrics is higher for *se(3)*-TrackNet as well, indicating problems with the orientation prediction.

Another reason for the lower metric scores might be the occlusion due to the ArUco markers. Large parts of the object are occluded by placing the markers on the objects, making predictions more difficult. However, the markers do not influence the depth measurement as they have no significant thickness, which is probably why the methods can still make reasonably accurate predictions.

Object	Frame of track loss
11_ubongo_red	75
11_ubongo_red	482

Table 6.4.: Tracking loss of 6DCenterPose during the real trajectories.

To further analyse the results of the methods, Figure 6.4 displays the absolute differences between the pose predictions of the methods and the ground truth poses. Again, the mean and standard deviation over all trajectories and all objects are displayed.
These results further confirm the observations made for the synthetic trajectories. 6DCenterPose, again, is quite accurate in the prediction made in x and y direction but struggles with the orientation prediction as well as the depth prediction, as shown in the errors in qw, qx, qy, qz, and z.
The high mean and standard deviation in y and z direction around frame 450 are due to the tracking loss of the red shape as shown in Table 6.4. This results indicates that the errors in depth prediction eventually lead to tracking loss.
As no occlusions occur in the real trajectories, the main reason for tracking loss are the inaccuracies of the pose predictions and problems in bridging the reality gap.

Figure 6.7 shows qualitative results of 6DCenterPose and *se(3)*-TrackNet for one of the real trajectories, as well as ground truth annotations obtained by the ArUco markers. Frame 328 shows the problems of 6DCenterTrack to accurately predict the depth and orientation of the red shape, which eventually leads to the tracking loss in frame 482 as shown in Table 6.4. Furthermore, the movement of one of the objects leads to changes in pose for the other objects, even the stationary ones. This result indicates that the pose predictions of the individual objects are highly correlated. In contrast, the pose predictions of *se(3)*-TrackNet are not correlated because the input image is cropped to a tight region around the tracked object. This way, the movement of one object cannot influence the pose prediction for another object.

Further qualitative results of 6DCenterPose and *se(3)*-TrackNet on real trajectories and absolute difference plots for the individual objects are shown in Appendix A.

6.4.4. Sim to real

Figure 6.5, Table 6.1, and Table 6.3 show the results of 6DCenterPose and *se(3)*-TrackNet on synthetic and real trajectories.
The values of the ADD and ADD-S metrics for 6DCenterPose decrease only slightly when evaluating on real trajectories instead of evaluating on synthetic trajectories. This result indicates that 6DCenterPose successfully bridges the reality gap, allowing training the method on synthetic images and transferring to real images.
In contrast, values of the ADD and ADD-S metrics decrease drastically for *se(3)*-TrackNet. However, *se(3)*-TrackNet still achieves high values for both metrics, outperforming 6DCenterTrack.

The main reason why *se(3)*-TrackNet struggles to bridge the reality gap is probably the synthetic training data. 6DCenterPose uses the whole images as input, which allows the method to benefit from the photo-realistic rendering of the entire scene during training. *se(3)*-TrackNet only uses a tight crop around the target image as input, making the photo-realistic rendering of the whole scene obsolete. Furthermore, due to the crop of the image, all the information provided to *se(3)*-TrackNet concerns the tracked object. This way, minor errors in the 3D models of the object can negatively influence the predictions. In contrast, 6DCenterPose uses the whole image as input. This way, minor errors in the 3D model only constitute a small portion of the input data, making the method less dependent on it.

For a more detailed analysis, accuracy-threshold curves for the individual objects are shown in Appendix A.

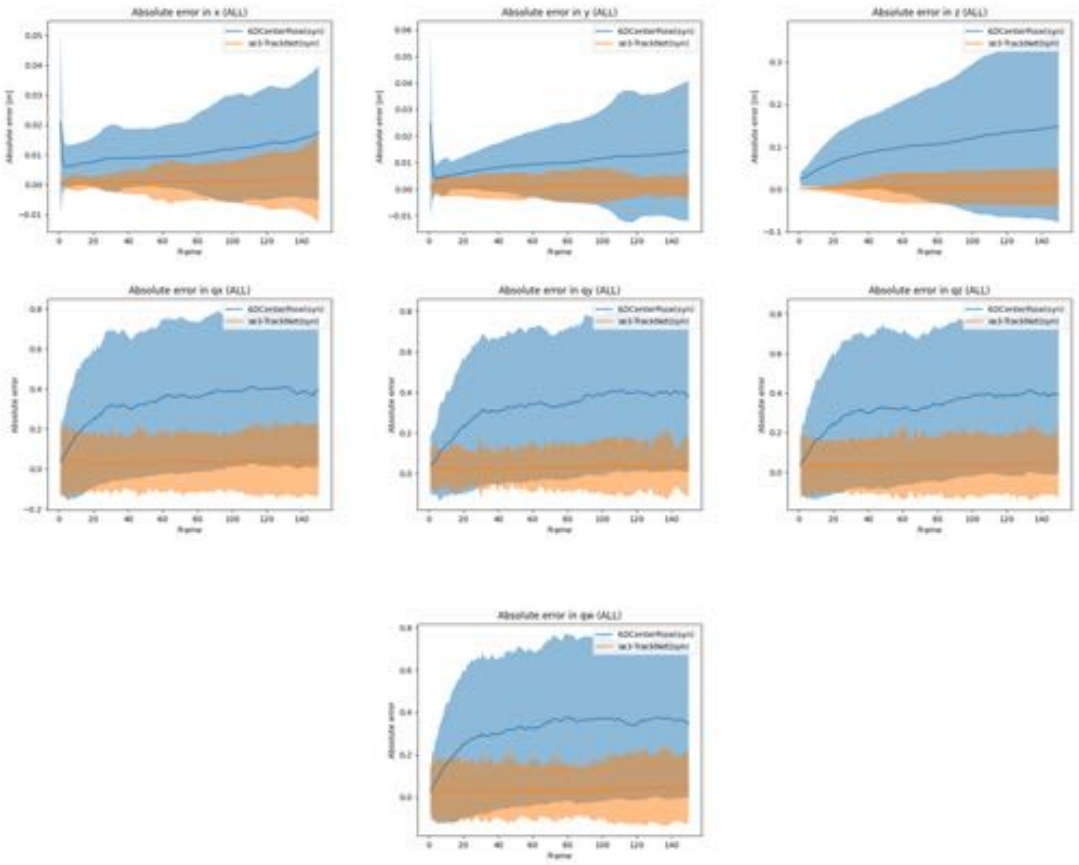


Figure 6.3.: Absolute errors of *se(3)*-TrackNet and 6DCenterPose for the synthetic trajectories, divided in the location coordinates and the quaternion representation of the orientation. The graphs show the mean of the absolute error over all trajectories and objects, and the shaded area shows the standard deviation around the mean.

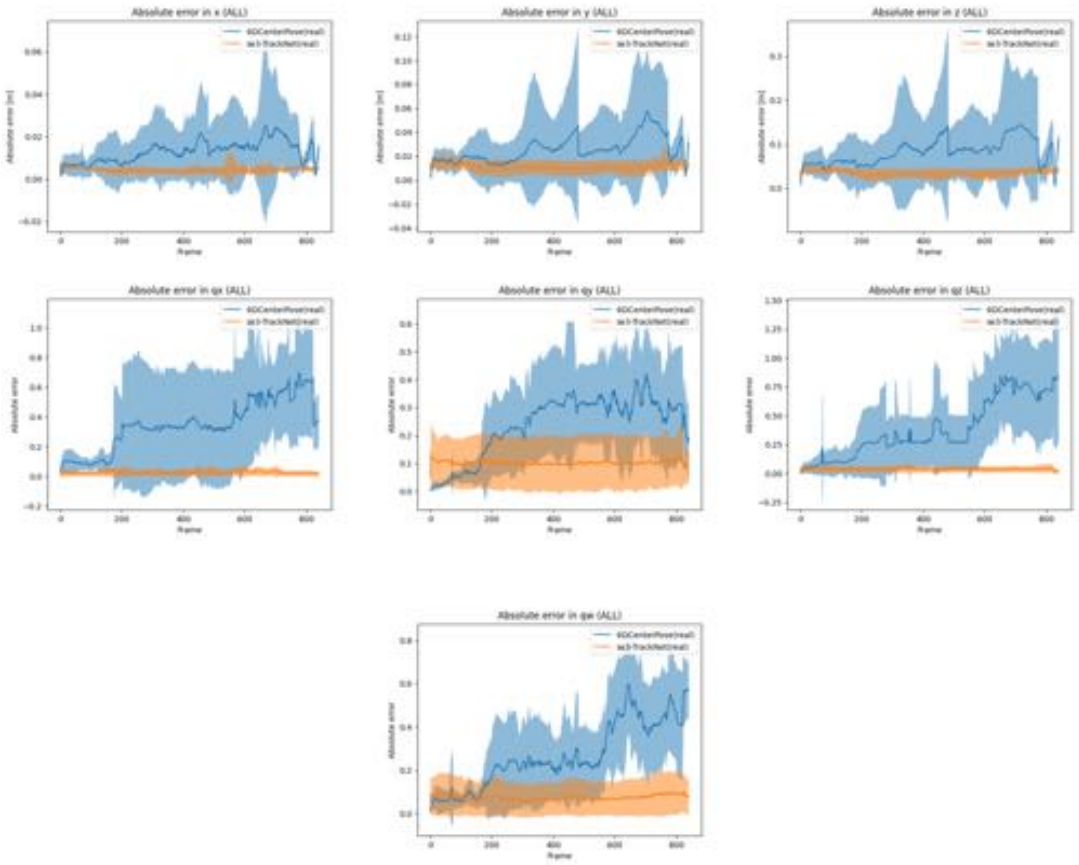


Figure 6.4.: Absolute errors of *se(3)*-TrackNet and 6DCenterPose for the real trajectories, divided in the location coordinates and the quaternion representation of the orientation. The graphs show the mean of the absolute error over all trajectories and objects, and the shaded area shows the standard deviation around the mean.

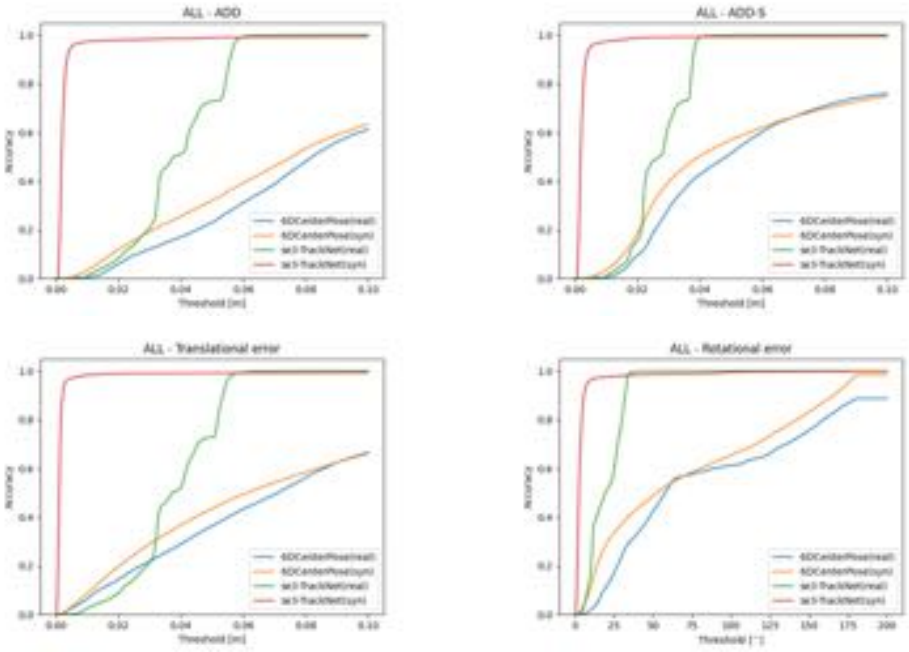
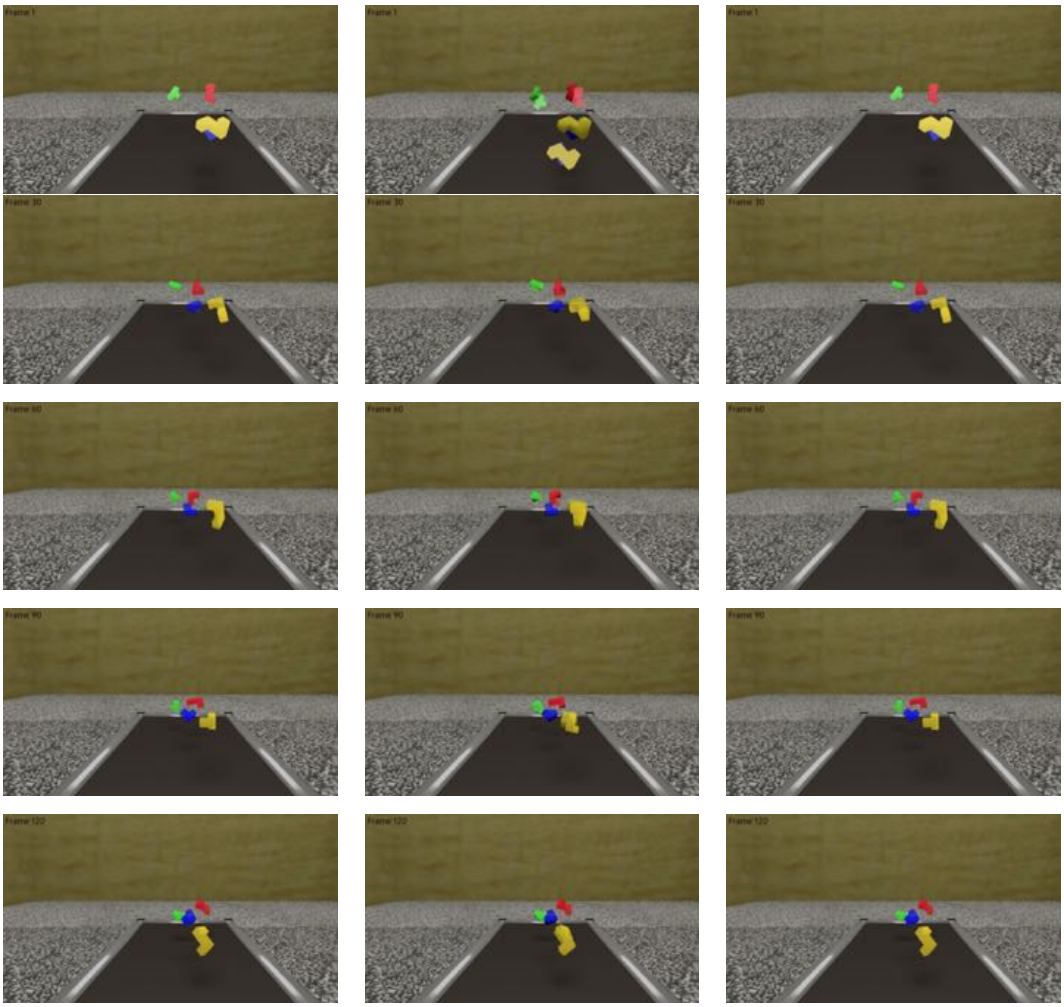
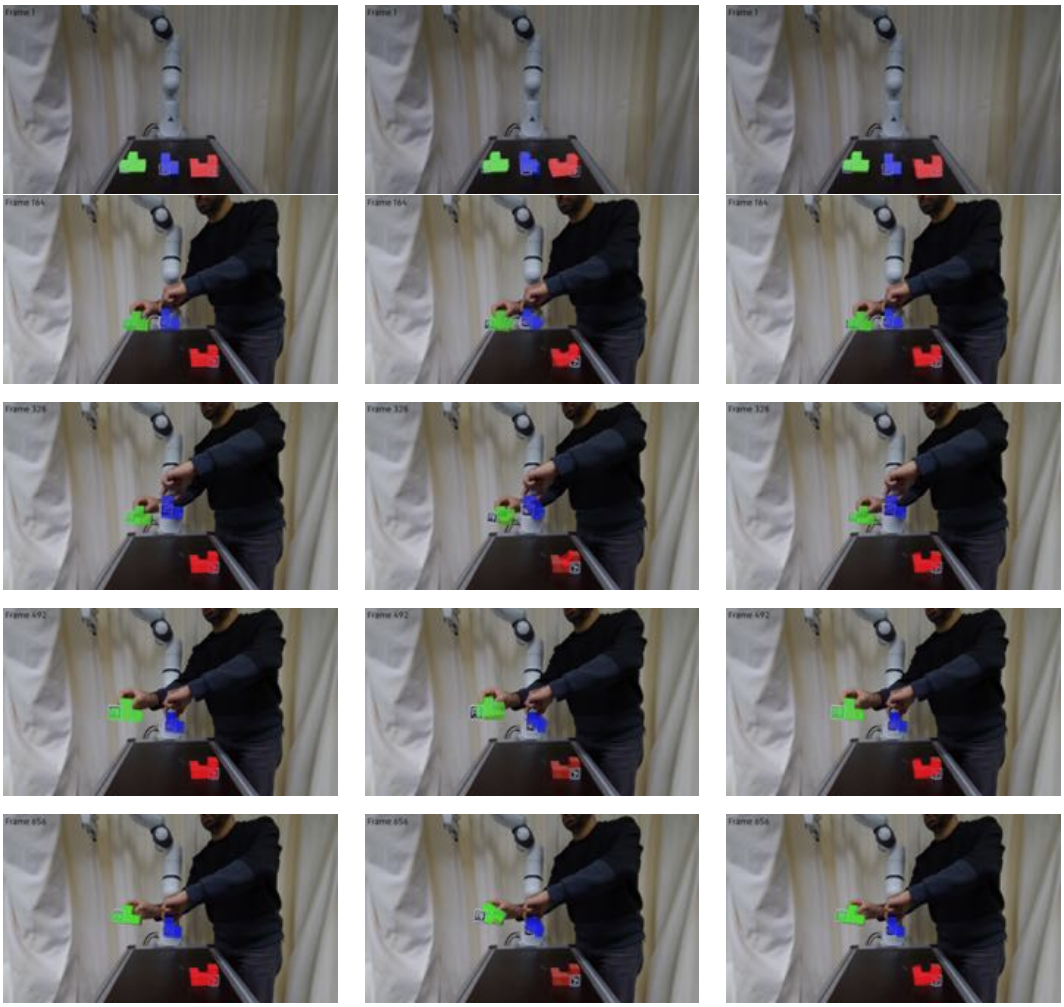


Figure 6.5.: Accuracy-threshold curves for *se(3)*-TrackNet and 6DCenterPose, evaluated on the real and synthetic test trajectories, aggregated over all tested shapes.



Ground truth | 6DCenterPose | *se(3)*-TrackNet [10]

Figure 6.6.: Qualitative results for one of the synthetic trajectories. The images of each row correspond to one method. The left row contains the ground truth poses used to render the images, the middle row contains predictions from the proposed method 6DCenterPose, and the right column contains predictions from *se(3)*-TrackNet [10].



Ground truth | 6DCenterPose | *se*(3)-TrackNet [10]

Figure 6.7.: Qualitative results for one of the real trajectories. The images of each row correspond to one method. The left row contains the ground truth poses obtained using ArUco markers [94], the middle row contains predictions from the proposed method 6DCenterPose, and the right column contains predictions from *se*(3)-TrackNet [10].

6.4.5. Discussion of results

To conclude the evaluation, this chapter analysis the problems observed during the experiments.

Sim-to-real transfer of *se(3)*-TrackNet

The first observation is the problem of *se(3)*-TrackNet to bridge the reality gap, even though, in the corresponding publication, they succeed at this task. This thesis uses the publicly available implementation of *se(3)*-TrackNet, which is assumed to be correct. The only difference to the publication is, therefore, the synthetic data used to train the method and the task used for evaluation.

The main difference between the proposed data generation pipeline in this thesis and the data generation of *se(3)*-TrackNet is that *se(3)*-TrackNet randomises more textures, e.g. the texture of the table. In contrast, we only randomise the wall textures. Additionally, *se(3)*-TrackNet uses a random number of point light sources while we only use the ceiling object as an ambient light source. Finally, the synthetic data of *se(3)*-TrackNet contains more objects which can occlude the tracked object, while occlusion is rare in our synthetic data, as only tracked objects are placed on the table. However, occlusion by other objects is not present in our real trajectories, so this fact should not influence the sim-to-real transfer.
Another problem might be the placement of the ArUco marker on the object as it introduces substantial occlusion of the object. However, *se(3)*-TrackNet is trained keeping occlusion in mind, so it should be robust to the marker placement.

Another source of error is the calibration of the intrinsics matrix $\boldsymbol{K}$. Errors in the calibration matrix can lead to inconsistencies in the perspective projection between the rendered images and the images obtained from the camera. The intrinsics matrix $\boldsymbol{K}$ is used in many parts of *se(3)*-TrackNet and 6DCenterPose, therefore, an accurately calibrated camera is essential.

Finally, the ground truth poses obtained from the ArUco markers could explain the performance degradation of *se(3)*-TrackNet. Having erroneous ground truth labels can result in a high error, even though the method produces valid pose predictions. However, the qualitative results shown in Figure 6.7 suggest that the poses obtained from the ArUco markers are reasonable.

6DCenterPose successfully bridges the reality gap. This indicates that 6DCenterPose is robust to potential errors in camera calibration, occlusions introduced by ArUco markers, and less variance in textures in the synthetic training data.

Problems in depth and orientation prediction

As shown in the experiments, the main problems of 6DCenterPose are to accurately predict the depth and the orientation of the object while being reasonably accurate in predicting the x and y coordinates of the object. The reason for this behaviour is that a change in x and y direction of the object pose results in a different location of the object in the image. In contrast, a change in z direction or a change in orientation changes the appearance of the object in the image while keeping the location in the image unchanged.

6DCenterPose operates on the whole image to predict the changes of pose. Therefore, most of the provided information does not concern a specific object, making it challenging to predict changes in appearance accurately. In contrast, *se(3)*-TrackNet uses a tight crop around the tracked object as input. Therefore, basically all the information provided as input concerns the object, making *se(3)*-TrackNet more sensitive to changes in appearance.

Another observation made from the qualitative results is that the predicted poses are highly correlated. A change in orientation and depth for one object frequently leads to changes in orientation and depth of other objects, even if those objects are not moving. This behaviour is also a result of using the whole image as input, which results in correlated pose predictions. Furthermore, all objects share the same change in location and change in orientation maps $\hat{\boldsymbol{T}}$ and $\hat{\boldsymbol{W}}$, respectively, leading to a further correlation of pose predictions. This behaviour can not occur for *se(3)*-TrackNet because it only operates on cropped parts of the image. This way, other objects are not visible to *se(3)*-TrackNet, and they can not influence the pose predictions.

7. Conclusion

This thesis presents 6DCenterPose, a fully convolutional neural network for RGB-D-based multi-object 6D pose tracking. Separate input branches are used to disentangle the feature representations of the inputs. This disentanglement allows training the method exclusively on synthetic training images while being robust to the domain change to the real world. To this end, a synthetic data generation pipeline is proposed to generate the training images. The output of the method consists of heatmaps, making 6DCenterPose invariant to the number of tracked objects. Given the initial poses of all objects, 6DCenterPose predicts a Lie Algebra representation of the pose changes, which are used to propagate previous pose predictions to the current frame. Offset predictions of object locations between the previous and current frames are used to greedily assign the pose changes to the corresponding objects.

Experiments on synthetic and real trajectories are conducted on the task of the board game *Ubongo3D*. They show that the reality gap is successfully bridged, achieving 26.3% on the ADD metric and 44.2% on the ADD-S metric on the real trajectories and 32.1% on the ADD metric and 47.7% on the ADD-S metric on the synthetic trajectories, resulting only in a slight drop in performance between the domains.

6DCenterPose runs at 12.8 Hz and is an online method, which makes it applicable in robot manipulation tasks. The advantage of 6DCenterPose is that all objects are tracked simultaneously, resulting in a runtime independent of the number of objects. Previous methods for 6D pose tracking are trained for a specific object, therefore, the runtime decreases with an increase of tracked objects.

Even though the reality gap is successfully bridged, the pose predictions of 6DCenterPose are not as precise as the predictions of the baseline method. A detailed analysis is presented, highlighting the weaknesses of 6DCenterPose, which indicate directions for future research.

Future research

The main shortcomings of 6DCenterPose are the predictions of depth and orientation as shown in Chapter 6.4. Future research should address these problems. Zhou et al. [14] highlight problems in depth predictions and show how to solve them using an output transformation. Instead of predicting a change in location map $\hat{\boldsymbol{T}}$, future research could predict the depth of the object as shown by Zhou et al. Together with the predicted object location in 2D and the camera intrinsics matrix $\boldsymbol{K}$, the 2D location can be projected back to 3D as shown in Chapter 4.2, obtaining the location part of the object poses.

To obtain more accurate orientation predictions, future research could follow PoseCNN [7]. They predict bounding boxes of the objects in the scene and use them in RoI pooling [29] to regress to quaternion representations of the objects separately. Ideally, this will improve the precision of the orientation predictions. Furthermore, this would disentangle the orientation predictions, which was shown to be problematic in the experiments.

Implementing these changes would result in absolute pose predictions for all objects instead of predicting pose changes. Predicting absolute poses has the additional advantage that no initial pose predictions are necessary to start the tracking pipeline. Furthermore, the tracking pipeline can easily recover from tracking loss, which is currently not the case.

The problems in bridging the reality gap observed for *se(3)*-TrackNet suggest that the data generation pipeline can be improved. Potential changes include adding point light sources to generate more realistic lighting and randomising object and table texture, following the idea of domain randomisation [20]. Additionally, following Falling Things [81], synthetic images could be obtained during the physics simulation to obtain more diverse object poses instead of only capturing stable poses. Furthermore, distractor objects can be added to the scene to incorporate occlusion to the training data and make it more challenging, ideally improving the robustness of the trained methods.

Finally, 6DCenterPose should be trained and evaluated on a standard 6D pose estimation dataset like YCB-Video [7] to better compare it to previous methods and eliminate potential problems of the pose predictions obtained from the ArUco markers.

Bibliography

[1] J. Zheng, X. Shi, A. Gorban, J. Mao, Y. Song, C. R. Qi, T. Liu, V. Chari, A. Cornman, Y. Zhou, *et al.*, "Multi-modal 3d human pose estimation with 2d weak supervision in autonomous driving," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2022.

[2] E. Marchand, H. Uchiyama, and F. Spindler, "Pose estimation for augmented reality: a hands-on survey," *IEEE transactions on visualization and computer graphics (TVCG)*, 2015.

[3] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, *et al.*, "An algorithmic perspective on imitation learning," *Foundations and trends® in robotics*, 2018.

[4] B. Wen, W. Lian, K. Bekris, and S. Schaal, "You only demonstrate once: Category-level manipulation from single visual demonstration," in *Robotics: science and systems (RSS)*, 2022.

[5] B. Wen, C. Mitash, S. Soorian, A. Kimmel, A. Sintov, and K. E. Bekris, "Robust, occlusion-aware pose estimation for objects grasped by adaptive hands," in *IEEE international conference on robotics and automation (ICRA)*, 2020.

[6] D. Kappler, F. Meier, J. Issac, J. Mainprice, C. G. Cifuentes, M. Wüthrich, V. Berenz, S. Schaal, N. Ratliff, and J. Bohg, "Real-time perception meets reactive motion generation," *IEEE robotics and automation letters (RA-L)*, 2018.

[7] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," in *Robotics: science and systems (RSS)*, 2018.

[8] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," in *Conference on robot learning (CoRL)*, 2018.

[9] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again," in *Proceedings of the IEEE international conference on computer vision (ICCV)*, 2017.

[10] B. Wen, C. Mitash, B. Ren, and K. E. Bekris, "se (3)-tracknet: Data-driven 6d pose tracking by calibrating image residuals in synthetic domains," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2020.

[11] X. Deng, A. Mousavian, Y. Xiang, F. Xia, T. Bretl, and D. Fox, "Poserbpf: A rao–blackwellized particle filter for 6-d object pose tracking," *IEEE transactions on robotics (T-RO)*, 2021.

[12] M. Garon and J.-F. Lalonde, "Deep 6-dof tracking," *IEEE transactions on visualization and computer graphics (TVCG)*, 2017.

[13] P. Dendorfer, A. Osep, A. Milan, K. Schindler, D. Cremers, I. Reid, S. Roth, and L. Leal-Taixé, "Motchallenge: A benchmark for single-camera multiple target tracking," *International journal of computer vision (IJCV)*, 2021.

[14] X. Zhou, V. Koltun, and P. Krähenbühl, "Tracking objects as points," in *Proceedings of the European conference on computer vision (ECCV)*, 2020.

[15] P. Bergmann, T. Meinhardt, and L. Leal-Taixe, "Tracking without bells and whistles," in *Proceedings of the IEEE international conference on computer vision (ICCV)*, 2019.

[16] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Detect to track and track to detect," in *Proceedings of the IEEE international conference on computer vision (ICCV)*, 2017.

[17] X. Weng, J. Wang, D. Held, and K. Kitani, "3d multi-object tracking: A baseline and new evaluation metrics," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2020.

[18] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3d object detection and tracking," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2021.

[19] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," *arXiv preprint arXiv:1904.07850*, 2019.

[20] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2017.

[21] A. Amini, A. S. Periyasamy, and S. Behnke, "T6d-direct: Transformers for multi-object 6d pose direct regression," in *DAGM German conference on pattern recognition (GCPR)*, 2022.

[22] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese, "Densefusion: 6d object pose estimation by iterative dense fusion," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2019.

[23] K. Wada, E. Sucar, S. James, D. Lenton, and A. J. Davison, "Morefusion: Multi-object reasoning for 6d pose estimation from volumetric fusion," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2020.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems (NeurIPS)*, 2017.

[25] A. Amini, A. Selvam Periyasamy, and S. Behnke, "Yolopose: Transformer-based multi-object 6d pose estimation using keypoint regression," in *International conference on intelligent autonomous systems (IAS)*, 2023.

[26] L. Zou, Z. Huang, N. Gu, and G. Wang, "6d-vit: Category-level 6d object pose estimation via transformer-based instance representation learning," *IEEE transactions on image processing (TIP)*, 2022.

[27] A. Beedu, H. Alamri, and I. Essa, "Video based object 6d pose estimation using transformers," *arXiv preprint arXiv:2210.13540*, 2022.

[28] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, 1992.

[29] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision (ICCV)*, 2015.

[30] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Ep n p: An accurate o (n) solution to the p n p problem," *International journal of computer vision (IJCV)*, 2009.

[31] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

[32] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable detr: Deformable transformers for end-to-end object detection," *arXiv preprint arXiv:2010.04159*, 2020.

[33] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Proceedings of the European conference on computer vision (ECCV)*, 2020.

[34] S. Li, Z. Yan, H. Li, and K.-T. Cheng, "Exploring intermediate representation for monocular vehicle pose estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2021.

[35] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE transactions on pattern analysis and machine intelligence (PAMI)*, 1991.

[36] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "Deepim: Deep iterative matching for 6d pose estimation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018.

[37] M. Wüthrich, P. Pastor, M. Kalakrishnan, J. Bohg, and S. Schaal, "Probabilistic object tracking using a range camera," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2013.

[38] J. Issac, M. Wüthrich, C. G. Cifuentes, J. Bohg, S. Trimpe, and S. Schaal, "Depth-based object tracking using a robust gaussian filter," in *IEEE international conference on robotics and automation (ICRA)*, 2016.

[39] N. A. Piga, F. Bottarel, C. Fantacci, G. Vezzani, U. Pattacini, and L. Natale, "Maskukf: An instance segmentation aided unscented kalman filter for 6d object pose and velocity tracking," *Frontiers in robotics and AI*, 2021.

[40] A. Doucet, N. d. Freitas, K. P. Murphy, and S. J. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," in *Conference on uncertainty in artificial intelligence (UAI)*, 2000.

[41] F. Stulp, E. Theodorou, J. Buchli, and S. Schaal, "Learning to grasp under uncertainty," in *IEEE international conference on robotics and automation (ICRA)*, 2011.

[42] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE international conference on computer vision (ICCV)*, 2015.

[43] B. Wen and K. Bekris, "Bundletrack: 6d pose tracking for novel objects without instance or category-level 3d models," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2021.

[44] D. Schmauser, Z. Qiu, N. Müller, and M. Nießner, "3d multi-object tracking with differentiable pose estimation," *arXiv preprint arXiv:2206.13785*, 2022.

[45] G. Brasó and L. Leal-Taixé, "Learning a neural solver for multiple object tracking," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2020.

[46] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, "Bytetrack: Multi-object tracking by associating every detection box," in *Proceedings of the European conference on computer vision (ECCV)*, 2022.

[47] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic engineering*, 1960.

[48] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430*, 2021.

[49] S. Tang, M. Andriluka, B. Andres, and B. Schiele, "Multiple people tracking by lifted multicut and person re-identification," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

[50] L. Zhang, Y. Li, and R. Nevatia, "Global data association for multi-object tracking using network flows," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2008.

[51] L. Leal-Taixé, G. Pons-Moll, and B. Rosenhahn, "Everybody needs somebody: Modeling social and grouping behavior on a linear programming multiple people tracker," in *Proceedings of the IEEE international conference on computer vision (ICCV workshops)*, 2011.

[52] W. Zhang, H. Zhou, S. Sun, Z. Wang, J. Shi, and C. C. Loy, "Robust multi-modality multi-object tracking," in *Proceedings of the IEEE international conference on computer vision (ICCV)*, 2019.

[53] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, 1955.

[54] P. Hu, J. Ziglar, D. Held, and D. Ramanan, "What you see is what you get: Exploiting visibility for 3d object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2020.

[55] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," *Advances in neural information processing systems (NeurIPS)*, 2016.

[56] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems (NeurIPS)*, 2015.

[57] G. D. Evangelidis and E. Z. Psarakis, "Parametric image alignment using enhanced correlation coefficient maximization," *IEEE transactions on pattern analysis and machine intelligence (PAMI)*, 2008.

[58] W. Choi and S. Savarese, "Multiple target tracking in world coordinate with single, minimally calibrated camera," in *Proceedings of the European conference on computer vision (ECCV)*, 2010.

[59] A. Andriyenko and K. Schindler, "Multi-target tracking by continuous energy minimization," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2011.

[60] T. Meinhardt, A. Kirillov, L. Leal-Taixe, and C. Feichtenhofer, "Trackformer: Multi-object tracking with transformers," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2022.

[61] F. Zeng, B. Dong, Y. Zhang, T. Wang, X. Zhang, and Y. Wei, "Motr: End-to-end multiple-object tracking with transformer," in *Proceedings of the European conference on computer vision (ECCV)*, 2022.

[62] P. Sun, J. Cao, Y. Jiang, R. Zhang, E. Xie, Z. Yuan, C. Wang, and P. Luo, "Transtrack: Multiple object tracking with transformer," *arXiv preprint arXiv:2012.15460*, 2020.

[63] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International journal of computer vision (IJCV)*, 2015.

[64] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Proceedings of the European conference on computer vision (ECCV)*, 2014.

[65] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes," in *Proceedings of the Asian conference on computer vision (ACCV)*, 2013.

[66] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *International conference on machine learning (ICML)*, 2015.

[67] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

[68] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems (NeurIPS)*, 2014.

[69] J. Xiao, A. Owens, and A. Torralba, "Sun3d: A database of big spaces reconstructed using sfm and object labels," in *Proceedings of the IEEE international conference on computer vision (ICCV)*, 2013.

[70] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision (IJCV)*, 2010.

[71] M. Hodosh, P. Young, and J. Hockenmaier, "Framing image description as a ranking task: Data, models and evaluation metrics," *Journal of artificial intelligence research (JAIR)*, 2013.

[72] Y. Movshovitz-Attias, T. Kanade, and Y. Sheikh, "How useful is photo-realistic rendering for visual learning?," in *Proceedings of the European conference on computer vision workshops (ECCV workshops)*, 2016.

[73] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *Advances in neural information processing systems (NeurIPS)*, 2014.

[74] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *Proceedings of the European conference on computer vision (ECCV)*, 2012.

[75] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla, "Understanding real world indoor scenes with synthetic data," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[76] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[77] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *Proceedings of the European conference on computer vision (ECCV)*, 2016.

[78] W. Qiu and A. Yuille, "Unrealcv: Connecting computer vision to unreal engine," in *Proceedings of the European conference on computer vision workshops (ECCV workshops)*, 2016.

[79] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops (CVPR workshops)*, 2018.

[80] C. Mitash, K. E. Bekris, and A. Boularias, "A self-supervised learning system for object detection using physics simulation and multi-view pose estimation," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2017.

[81] J. Tremblay, T. To, and S. Birchfield, "Falling things: A synthetic dataset for 3d object detection and pose estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops (CVPR workshops)*, 2018.

[82] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[83] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *Proceedings of the European conference on computer vision (ECCV)*, 2018.

[84] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

[85] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision (ICCV)*, 2017.

[86] J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik, "Human pose estimation with iterative error feedback," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[87] M. Fieraru, A. Khoreva, L. Pishchulin, and B. Schiele, "Learning to refine human pose estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops (CVPR workshops)*, 2018.

[88] G. Moon, J. Y. Chang, and K. M. Lee, "Posefix: Model-agnostic general human pose refinement network," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2019.

[89] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[90] B. Xiao, H. Wu, and Y. Wei, "Simple baselines for human pose estimation and tracking," in *Proceedings of the European conference on computer vision (ECCV)*, 2018.

[91] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning (ICML)*, 2015.

[92] M. Denninger, D. Winkelbauer, M. Sundermeyer, W. Boerdijk, M. Knauer, K. H. Strobl, M. Humt, and R. Triebel, "Blenderproc2: A procedural pipeline for photorealistic rendering," *Journal of open source software (JOSS)*, 2023.

[93] K. Shoemake, "Animating rotation with quaternion curves," in *International conference on computer graphics and interactive techniques (SIGGRAPH)*, 1985.

[94] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern recognition*, 2014.

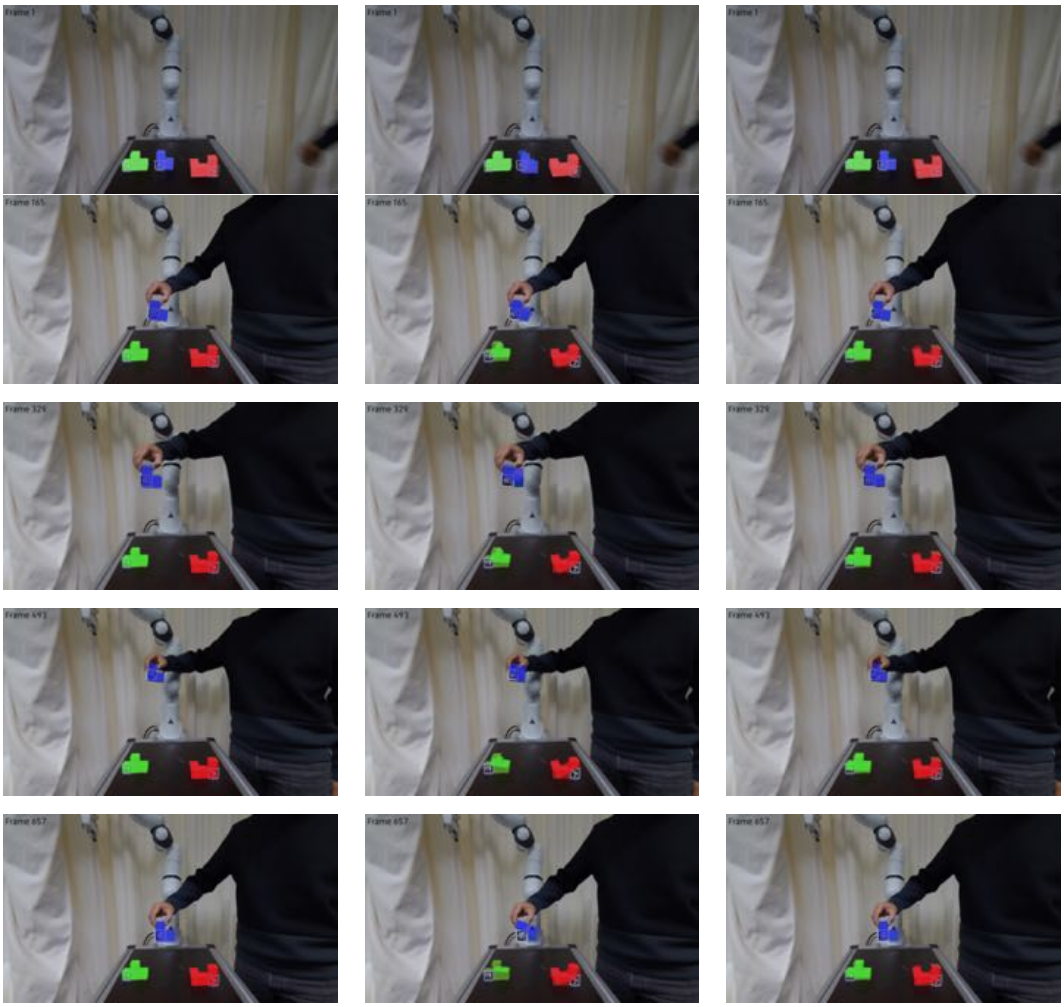
[95] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International conference on learning representations (ICLR)*, 2015.

A. Further results

This chapter provides more detailed results for the discussed methods. Figure A.1, Figure A.2, and Figure A.3 provide more qualitative results on real trajectories. Figure A.4, Figure A.5, and Figure A.6 provide more qualitative results on synthetic trajectories.

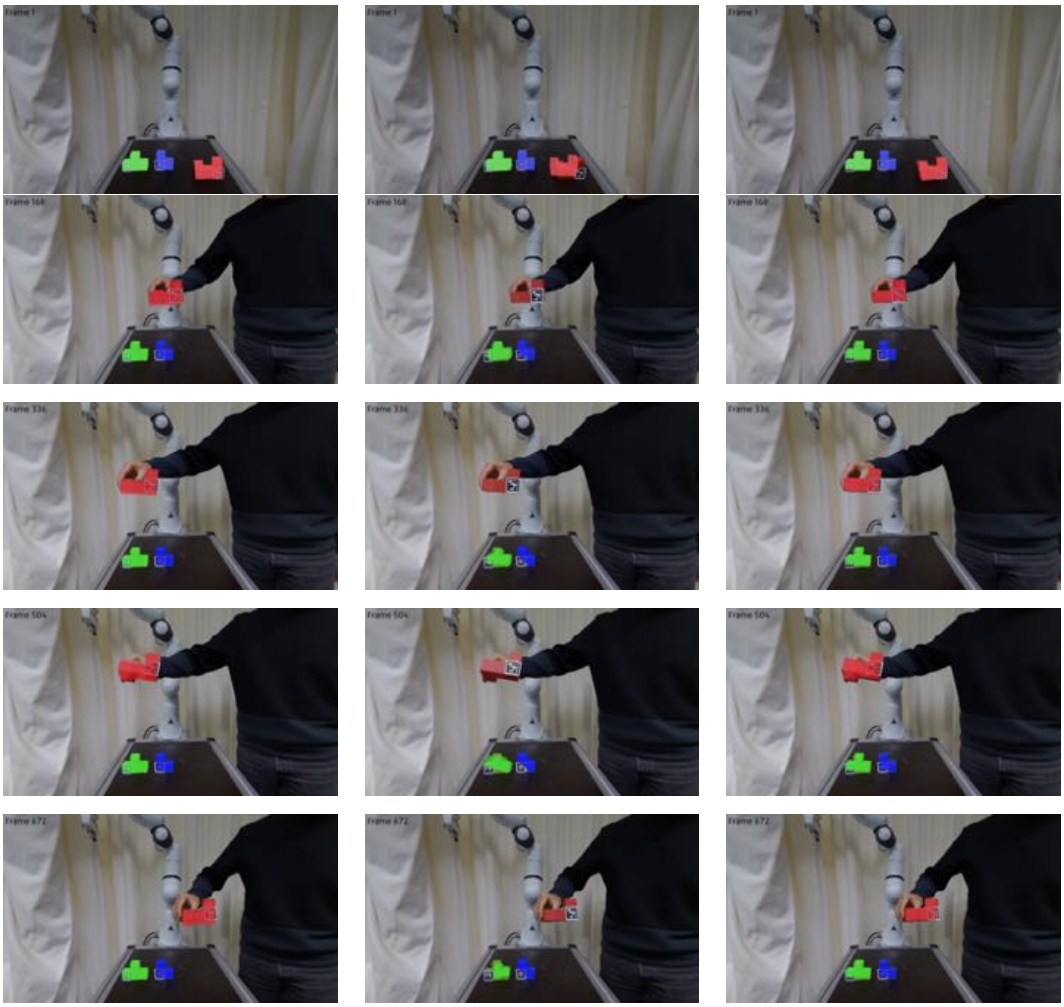
Figure A.7, Figure A.8, and Figure A.9 display absolute error curves for the real trajectories, separated into the individual objects. Figure A.10, Figure A.11, Figure A.12, and Figure A.13 display absolute error curves for the synthetic trajectories, separated into the individual objects.

Finally, Figure A.14, Figure A.15, Figure A.16, and Figure A.17 show accuracy-threshold curves for the evaluated methods, separated into the individual objects.



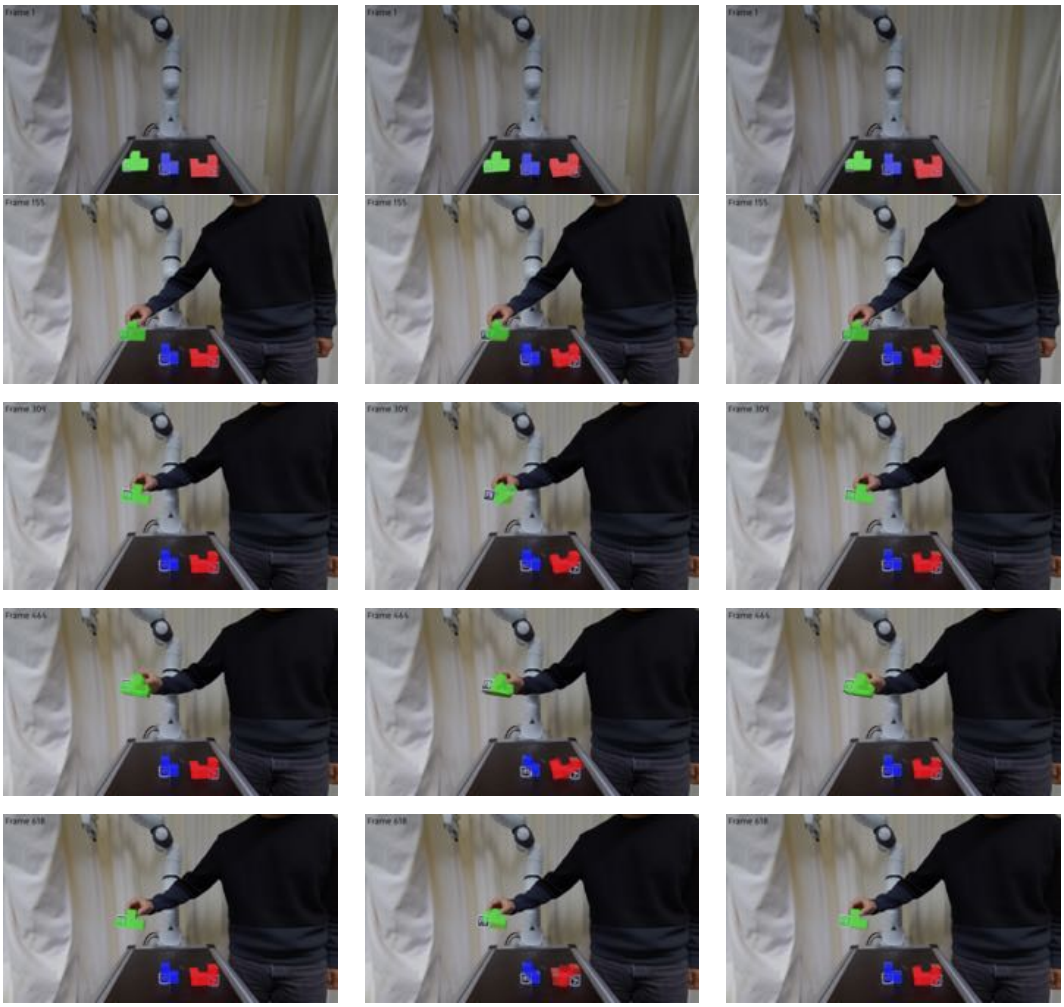
Ground truth | 6DCenterPose | *se*(3)-TrackNet [10]

Figure A.1.: Qualitative results for one of the real trajectories. The images of each row correspond to one method. The left row contains the ground truth poses obtained using ArUco markers [94], the middle row contains predictions from the proposed method 6DCenterPose, and the right column contains predictions from *se*(3)-TrackNet [10].



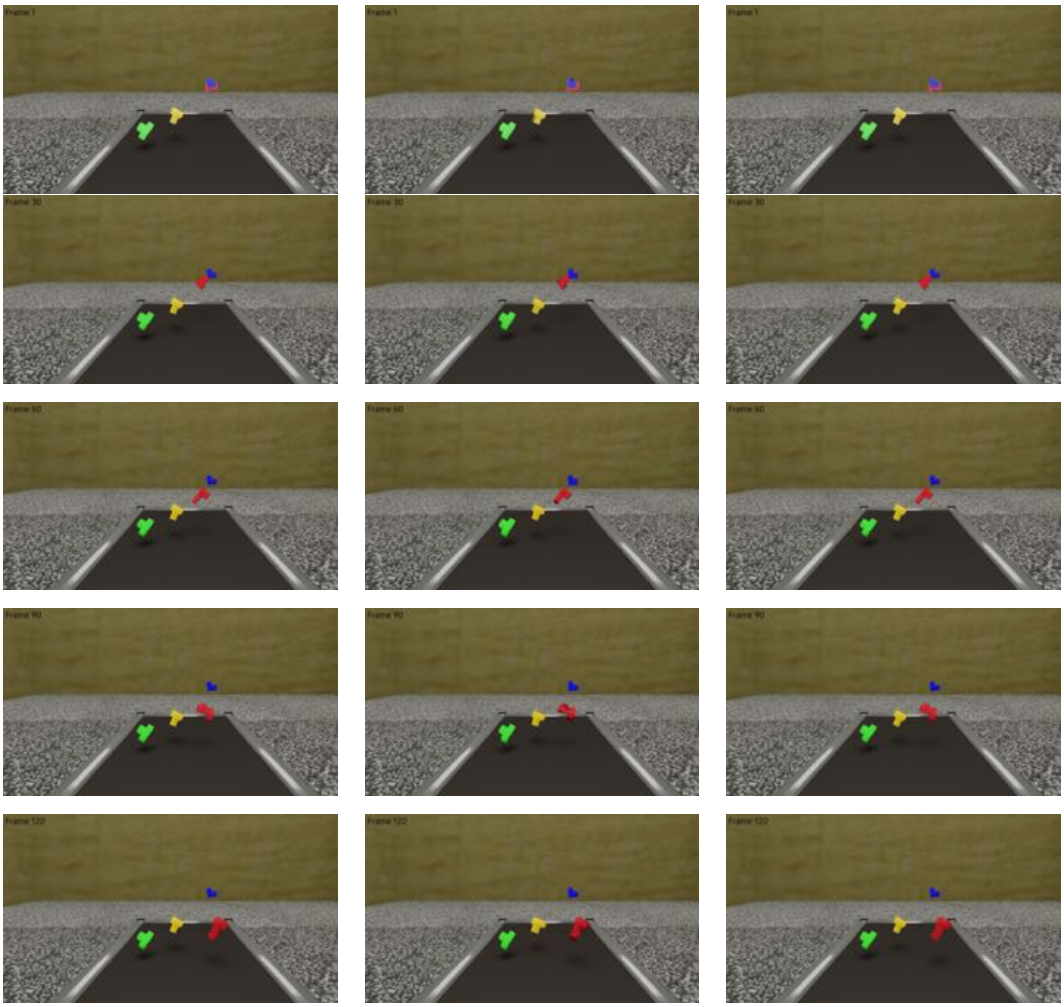
Ground truth | 6DCenterPose | *se*(3)-TrackNet [10]

Figure A.2.: Qualitative results for one of the real trajectories. The images of each row correspond to one method. The left row contains the ground truth poses obtained using ArUco markers [94], the middle row contains predictions from the proposed method 6DCenterPose, and the right column contains predictions from *se*(3)-TrackNet [10].



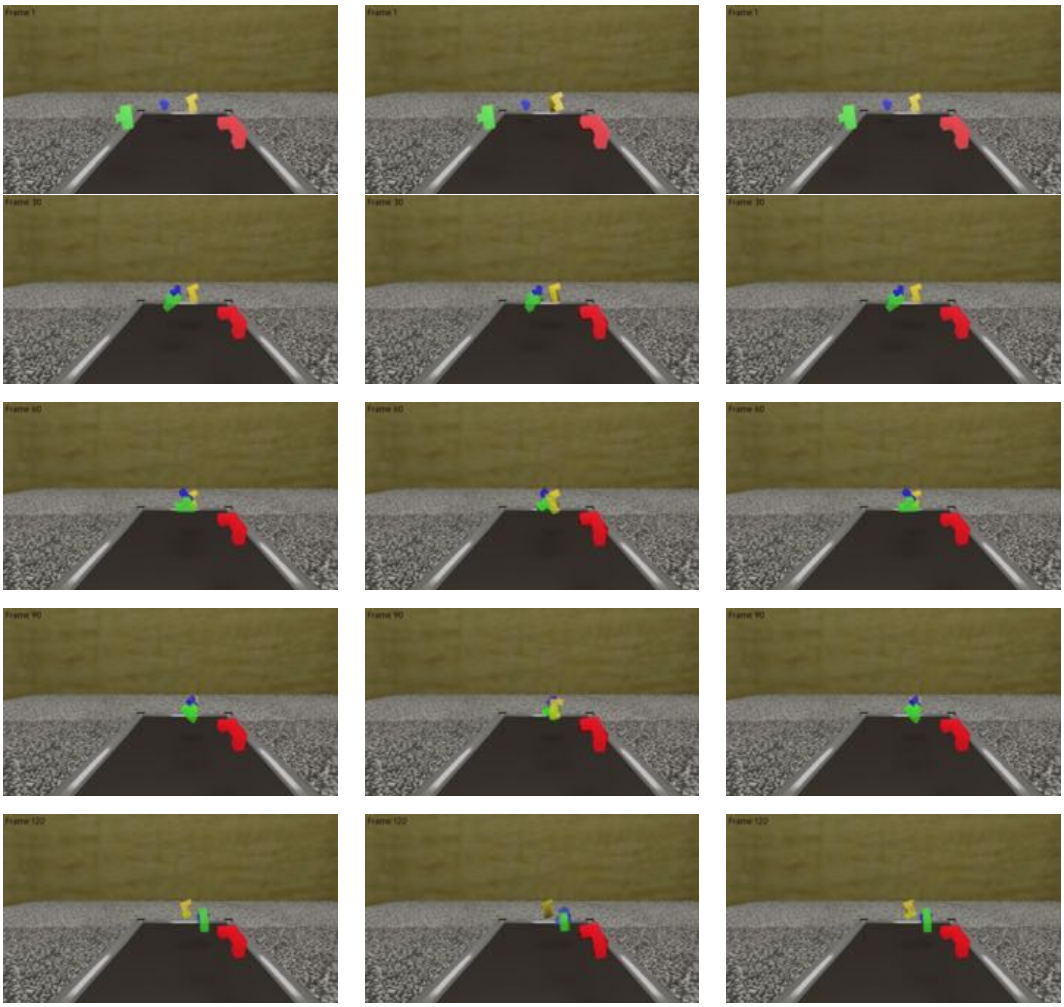
Ground truth | 6DCenterPose | *se(3)*-TrackNet [10]

Figure A.3.: Qualitative results for one of the real trajectories. The images of each row correspond to one method. The left row contains the ground truth poses obtained using ArUco markers [94], the middle row contains predictions from the proposed method 6DCenterPose, and the right column contains predictions from *se(3)*-TrackNet [10].



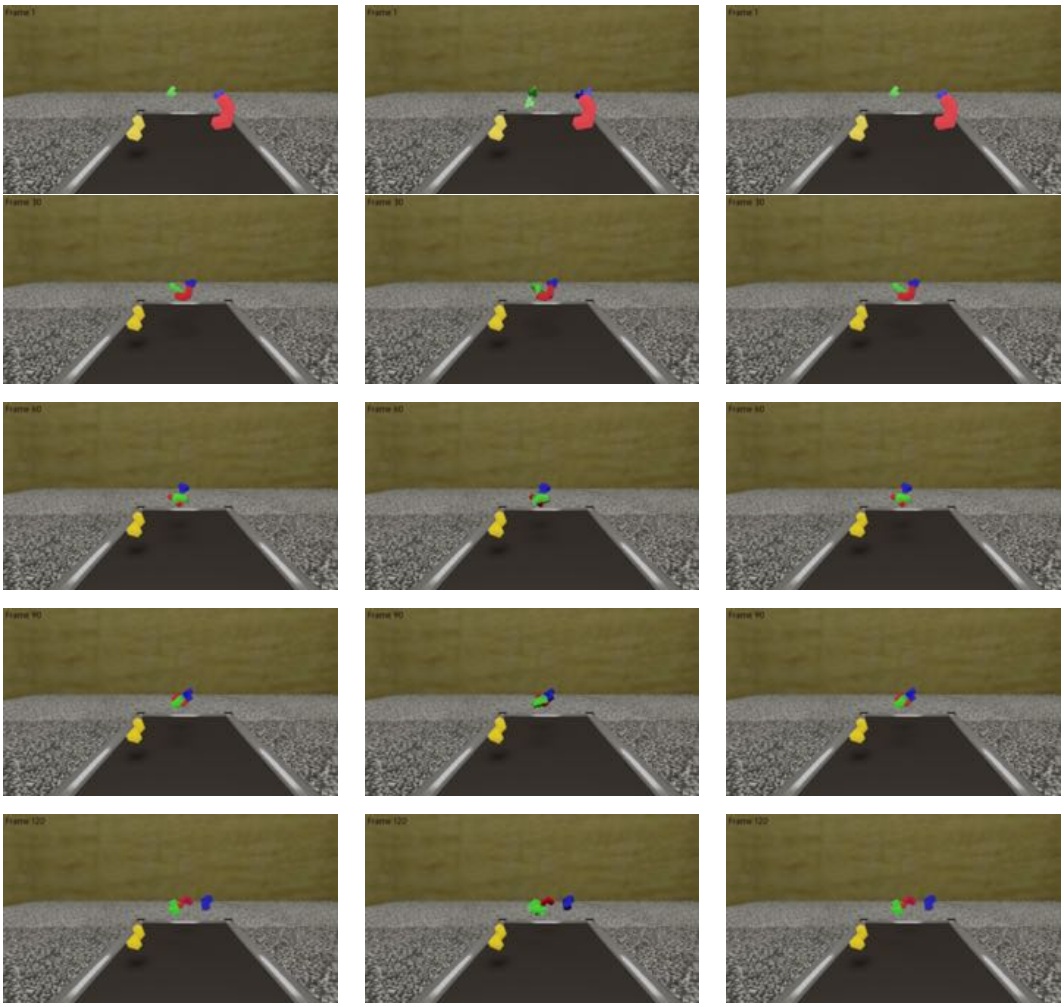
Ground truth | 6DCenterPose | *se*(3)-TrackNet [10]

Figure A.4.: Qualitative results for one of the synthetic trajectories. The images of each row correspond to one method. The left row contains the ground truth poses used to render the images, the middle row contains predictions from the proposed method 6DCenterPose, and the right column contains predictions from *se*(3)-TrackNet [10].



Ground truth | 6DCenterPose | *se*(3)-TrackNet [10]

Figure A.5.: Qualitative results for one of the synthetic trajectories. The images of each row correspond to one method. The left row contains the ground truth poses used to render the images, the middle row contains predictions from the proposed method 6DCenterPose, and the right column contains predictions from *se*(3)-TrackNet [10].



Ground truth | 6DCenterPose | *se*(3)-TrackNet [10]

Figure A.6.: Qualitative results for one of the synthetic trajectories. The images of each row correspond to one method. The left row contains the ground truth poses used to render the images, the middle row contains predictions from the proposed method 6DCenterPose, and the right column contains predictions from *se*(3)-TrackNet [10].

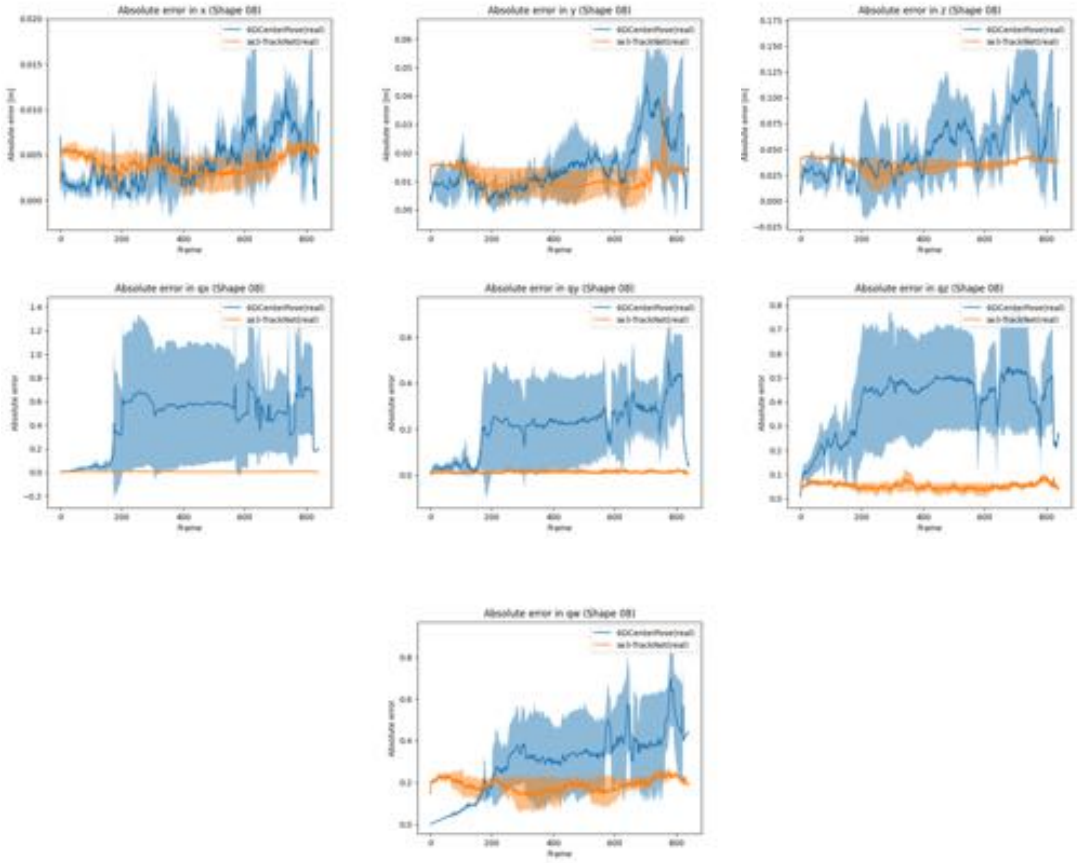


Figure A.7.: Absolute errors of *se(3)*-TrackNet and 6DCenterPose for the real trajectories, divided in the location coordinates and the quaternion representation of the orientation. The graphs show the mean of the absolute error over all trajectories for shape 08, and the shaded area shows the standard deviation around the mean.

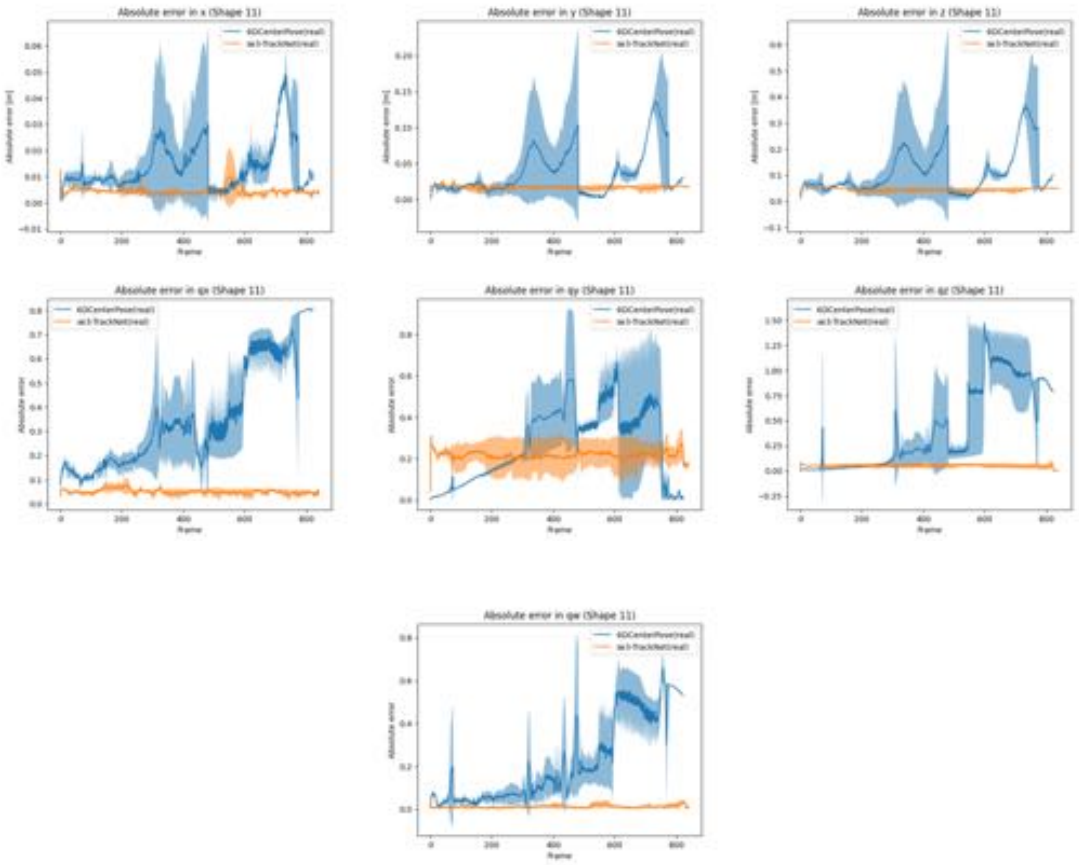


Figure A.8.: Absolute errors of *se(3)*-TrackNet and 6DCenterPose for the real trajectories, divided in the location coordinates and the quaternion representation of the orientation. The graphs show the mean of the absolute error over all trajectories for shape 11, and the shaded area shows the standard deviation around the mean.

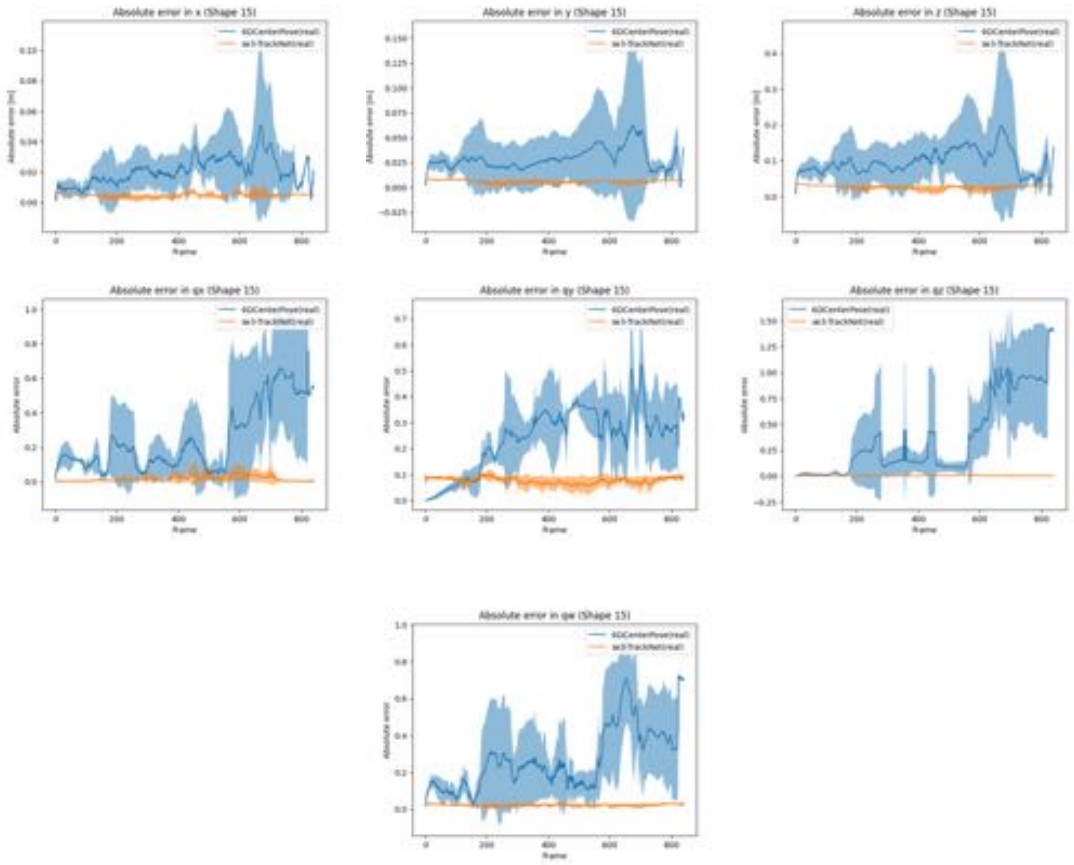


Figure A.9.: Absolute errors of *se(3)*-TrackNet and 6DCenterPose for the real trajectories, divided in the location coordinates and the quaternion representation of the orientation. The graphs show the mean of the absolute error over all trajectories for shape 15, and the shaded area shows the standard deviation around the mean.

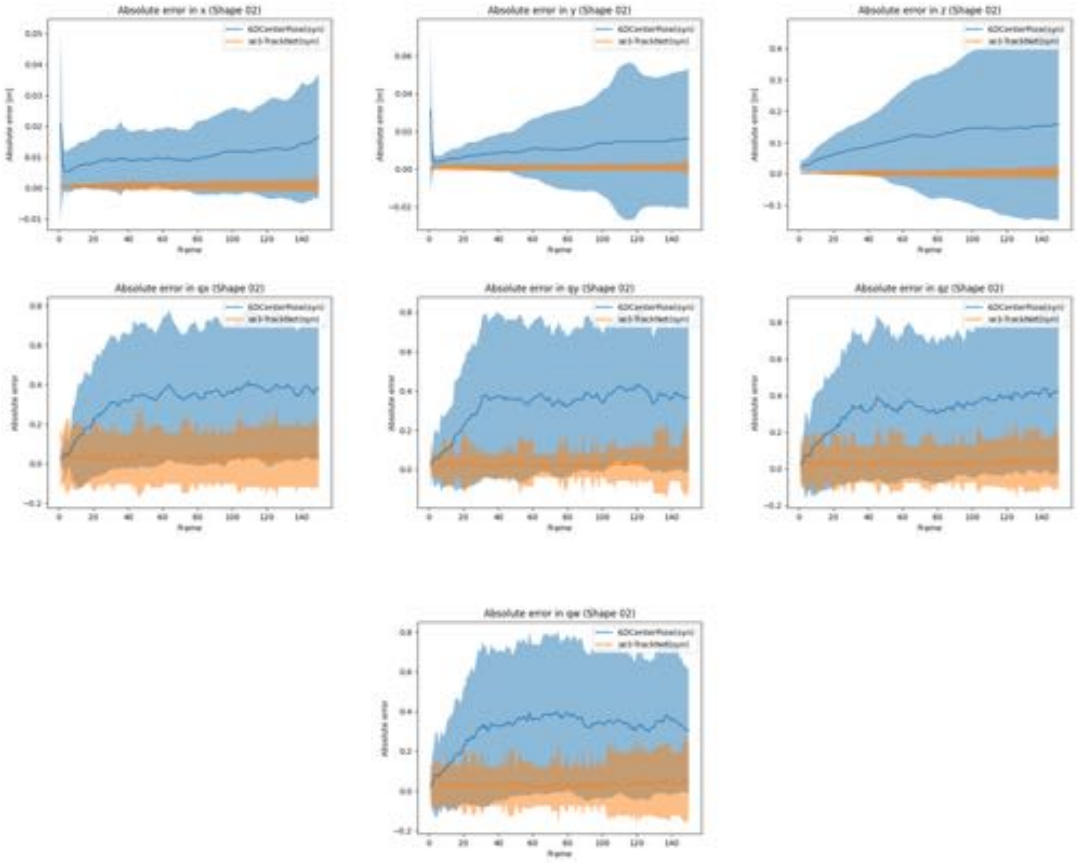


Figure A.10.: Absolute errors of *se(3)*-TrackNet and 6DCenterPose for the syn trajectories, divided in the location coordinates and the quaternion representation of the orientation. The graphs show the mean of the absolute error over all trajectories for shape 02, and the shaded area shows the standard deviation around the mean.

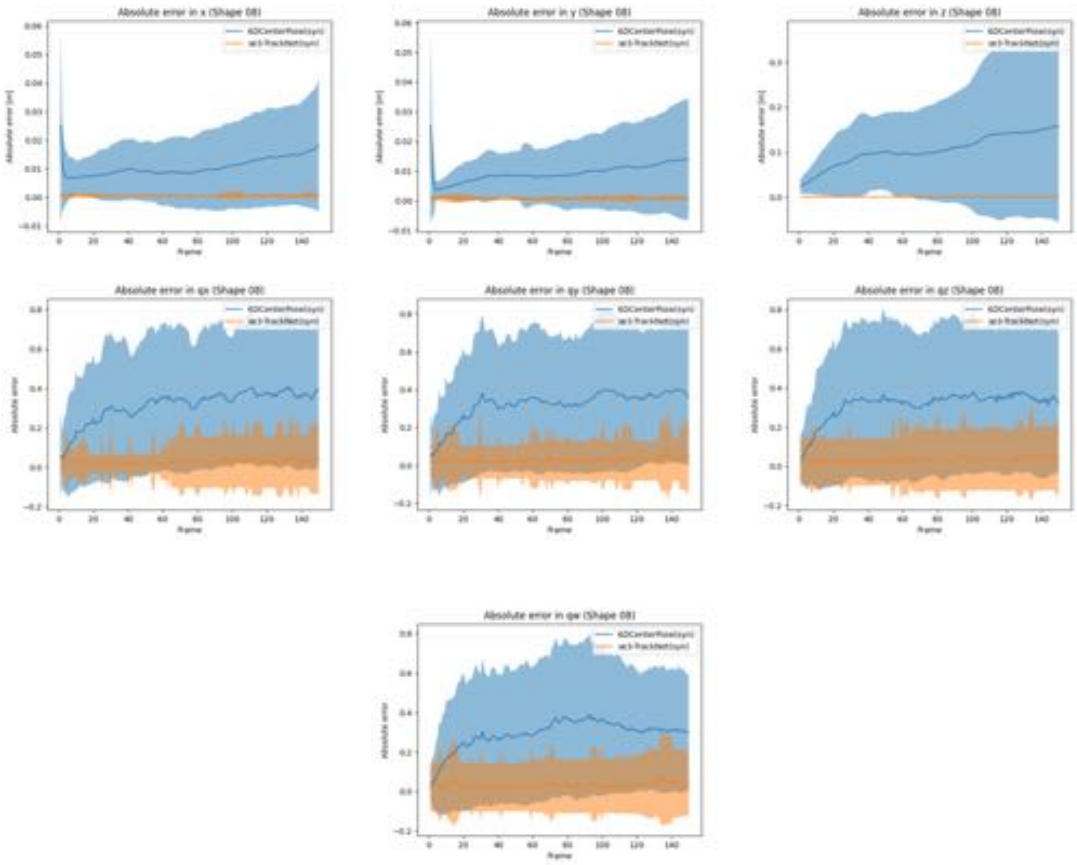


Figure A.11.: Absolute errors of *se*(3)-TrackNet and 6DCenterPose for the synthetic trajectories, divided in the location coordinates and the quaternion representation of the orientation. The graphs show the mean of the absolute error over all trajectories for shape 08, and the shaded area shows the standard deviation around the mean.

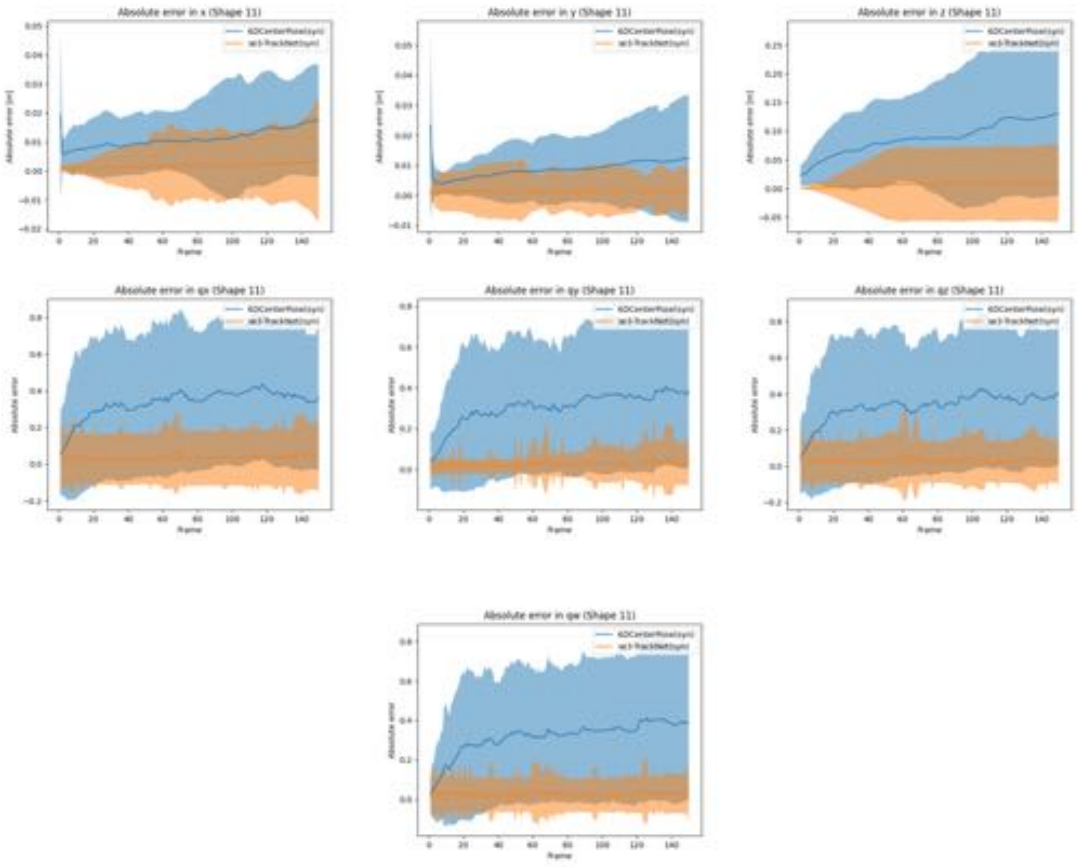


Figure A.12.: Absolute errors of *se(3)*-TrackNet and 6DCenterPose for the synthetic trajectories, divided in the location coordinates and the quaternion representation of the orientation. The graphs show the mean of the absolute error over all trajectories for shape 11, and the shaded area shows the standard deviation around the mean.

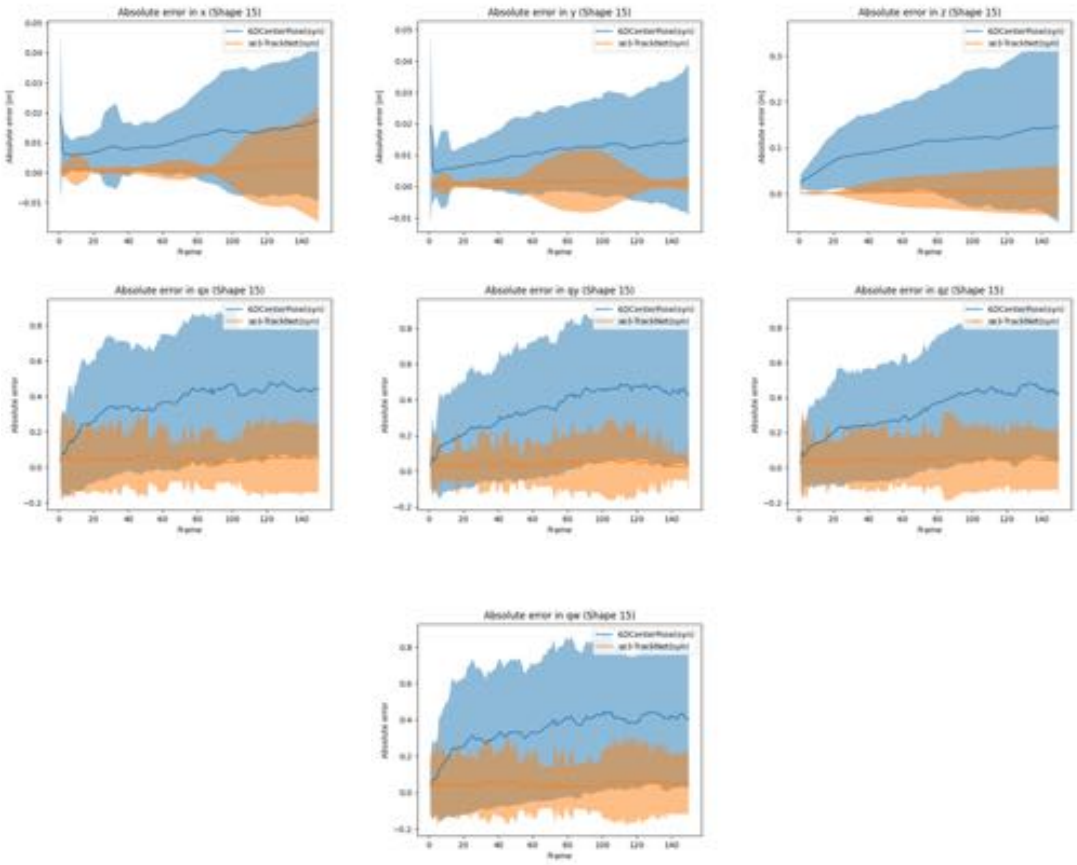


Figure A.13.: Absolute errors of *se(3)*-TrackNet and 6DCenterPose for the synthetic trajectories, divided in the location coordinates and the quaternion representation of the orientation. The graphs show the mean of the absolute error over all trajectories for shape 15, and the shaded area shows the standard deviation around the mean.

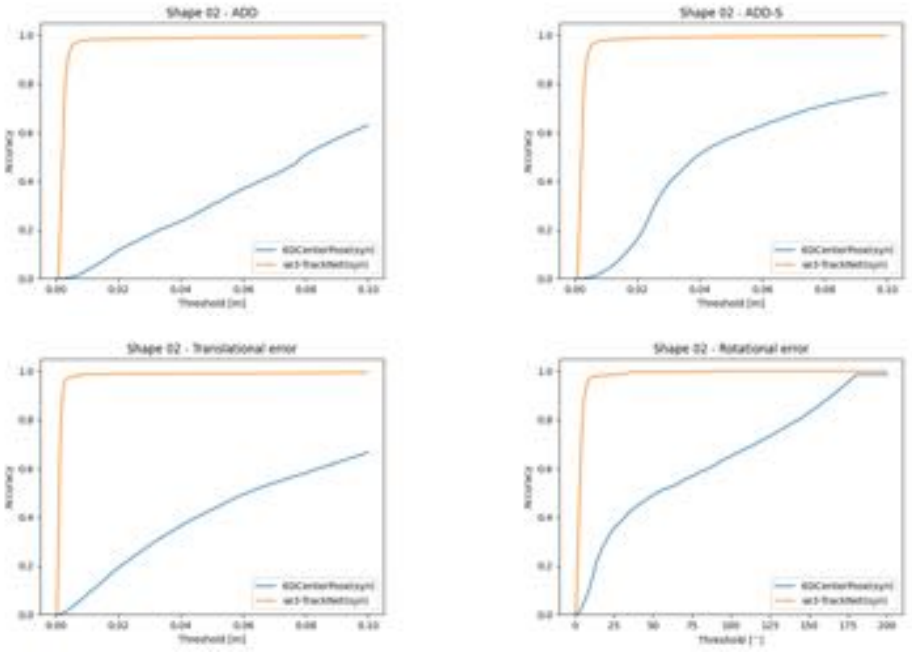


Figure A.14.: Accuracy-threshold curves for *se(3)*-TrackNet and 6DCenterPose, evaluated on the real and synthetic test trajectories for shape 02.

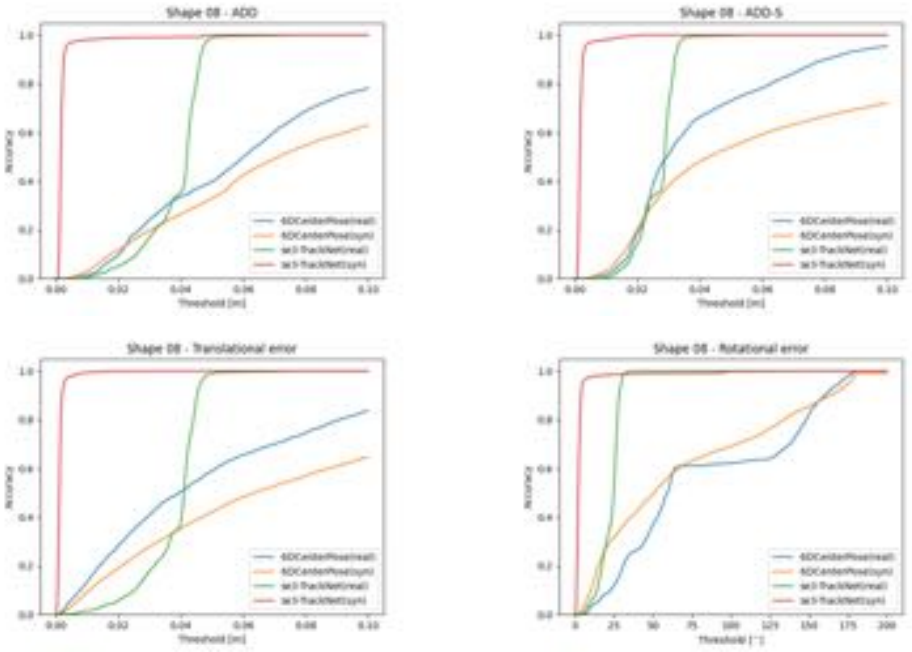


Figure A.15.: Accuracy-threshold curves for *se(3)*-TrackNet and 6DCenterPose, evaluated on the real and synthetic test trajectories for shape 08.

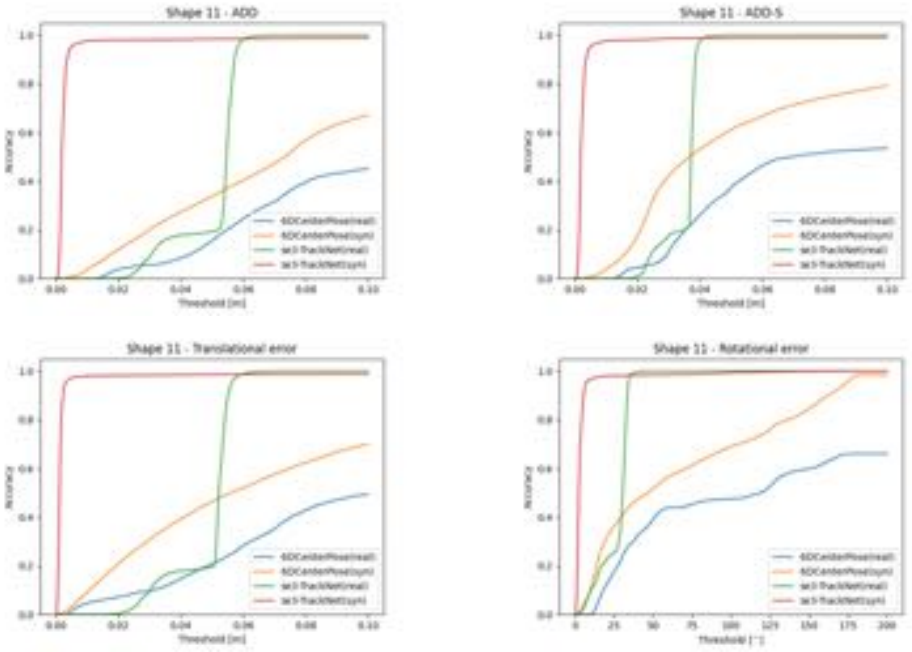


Figure A.16.: Accuracy-threshold curves for *se(3)*-TrackNet and 6DCenterPose, evaluated on the real and synthetic test trajectories for shape 11.

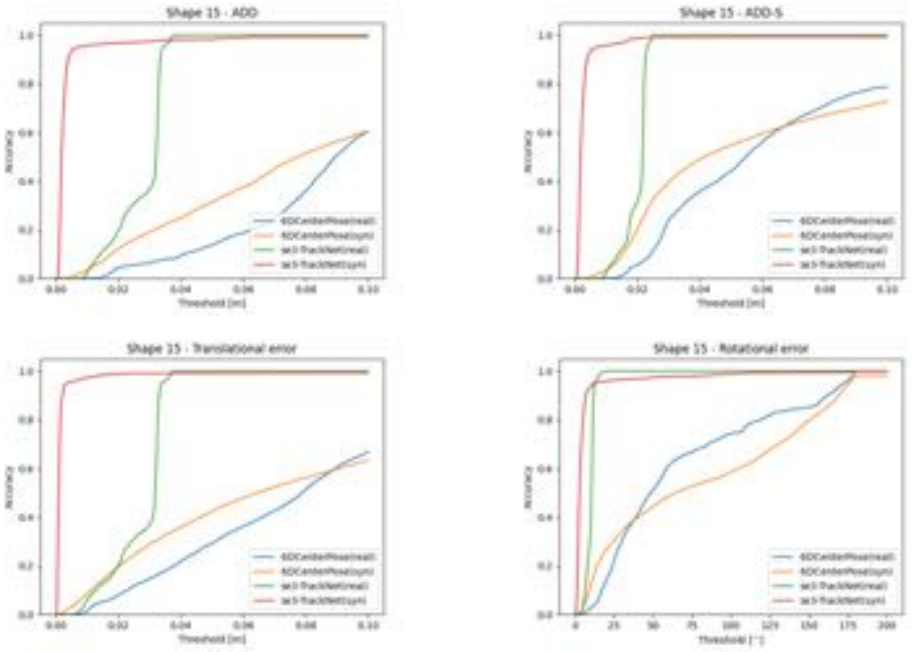


Figure A.17.: Accuracy-threshold curves for *se(3)*-TrackNet and 6DCenterPose, evaluated on the real and synthetic test trajectories for shape 15.