InsertionNet for Autonomous Assembly of Self-Interlocking Blocks

Bachelor thesis by Dominik Horstkötter
Date of submission: May 20, 2024

1. Review: Prof. Jan Peters, Ph.D.
2. Review: M.Sc. Tim Schneider
3. Review: Dr.-Ing. Boris Belousov
4. Review: Alap Kshirsagar, Ph.D.
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Abstract

With people growing more aware of environmental pollution, the resource efficiency and waste reduction become more important. Both of these can be improved by building structures reusable, so they can be disassembled without producing waste. Self-interlocking building blocks, like the SL-block, are designed for that purpose. The name comes from the S and L shapes, that one SL-block is made of. But their contact-rich assembly, that keeps the construct stable, makes the task of autonomous assembly difficult.

We approach this problem, by using a method with minimal contact, namely the InsertionNet 2.0. It is based on a neural network architecture designed to insert electrical plugs into their sockets, while having as few contact to the surroundings of the socket as possible. We adapt the InsertionNet 2.0 to a robot and cameras better suited for our task, which has more occlusion and contact due to the size and shape of the SL-block. Instead of an UR5e, we use the Franka Emika Panda robot arm as one of the most widely used research robots. The Intel RealSense D435 cameras are replaced by D405 with better depth vision in our working distance. We collect the data for the SL-Block assembly, train the network and test the insertion on the real system.

To address local optima during the training process, we changed the representations and can show empirically that the new representation can escape local optima much better, resulting in a ten times lower error in distance.

Even though the training converges, the system is not able to complete the task. Our approach seems to be prone to small environment changes. Our attempts to reduce the irrelevant information in the image, through cropping it, did result in a small improvement.

Despite our improvements, we found InsertionNet 2.0 to be unable to solve the SL-block insertion task. We believe that the challenging nature of the task can be overcome by further modifications, such as the collection of a larger dataset or more sophisticated neural network architectures. We plan to address these challenges in future work.

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

Hiermit erkläre ich, Dominik Horstkötter, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle 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.

Darmstadt, May 20, 2024

Dominik Horstkötter

D. Horstkötter

Contents

Algorithms, Figures and Tables

List of Algorithms

List of Figures

Abbreviations

Abbreviations

Notation	Description	Page List
DoF	Degrees of Freedom	6
EE	end effector	6, 13, 14, 26
Franka	The Franka Robot used for this work	13, 16
NN	neural network	6, 8, 10, 12

1 Introduction

As the importance of waste reduction and therefore reuse of building materials grows, so does the need for reusable structures. But most common structures are built with adhesives like mortar or glue, which naturally makes the disassembly and thus reuse hard. To avoid the problem of either having structures that are not robust or that can not be reused easily, engineers and architects are searching for self-interlocking building blocks, that have no need for adhesives to hold together.

An example of such self-interlocking blocks is the SL-block [1, 2]. The name comes from the S and L shaped tetracubes, that one SL-block is made of. But the assembly of such self-interlocking blocks is a very contact rich task, due to the structure of the blocks. This makes it hard for robots to assemble structures with these blocks. If we want to make the assembly process of SL-blocks more efficient by using autonomous assembly robots, we need the ability to handle these contact rich situations.

Not only the assembly of SL-blocks is contact rich, also the peg-in-hole tasks and the insertion of plugs in their socket have this problem [3, 4]. To decrease the contact in insertion tasks and thus make them easier, InsertionNet2.0 [5], uses two wrist-mounted cameras, that implicitly obtain depth information, while always tracking the plug to insert.

The goal of this thesis is to adapt the method of InsertionNet2.0 from the small electrical plugs to SL-blocks, which are much larger and thus not only have even more contact, but also more occlusion. The problem statement is to insert an SL-block into a socket consisting of two preassembled SL-blocks.

The rest of this thesis is structured as follows: Chapter 2 discusses existing work on self interlocking blocks and autonomous assembly, to give an overview of the current state of the art. Chapter 3 gives the background knowledge needed to understand our setup and methods. We explain the work of Shih [1] on the structure of SL-blocks. This chapter also contains how InsertionNet2.0 by Spector, Tchuiev, and Di Castro [5] works and why the continuous representation for 3-dimensional rotations that Zhou et al. [6] propose is better than the Euler angles that InsertionNet2.0 uses. Our own work and methods are described in Chapter 4, where we present the differences in task and setup, before illustrating the resulting changes we made to the original InsertionNet2.0. We show our results in Chapter 5, where we evaluate the loss curves and examine the errors in the insertion process. In Chapter 6, we discuss our results and remaining problems, propose possible approaches to solve them. We finish this work with an outlook on further work, that we can implement after we solved the remaining problems.

2 Related Work

In this section, we describe prior research on self-interlocking blocks (Section 2.1) and automated assembly (Section 2.2), we continue with previous work on autonomous insertions and assembly, to situate this thesis in the existing literature.

2.1 Self Interlocking Blocks

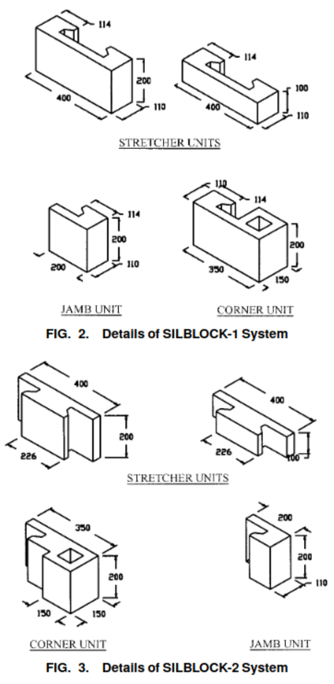
Shih [1] introduced a self interlocking building block, the SL-block, in 2016. It is an octocube, consisting of two tetracubes, shaped like an S and an L, attached to each other (see Fig. 3.1). The blocks are not only self-interlocking: due to their topological properties, they can be assembled to various structures. In [2], Shih introduced a context-free grammar, describing the different ways in which the SL-Blocks can engage and be assembled.

Liu, Liu, and Charkhab [7] have already developed an algorithm to generate assembly sequences, such that larger structures can be assembled autonomously. But the transfer of their simulation result to the actual working environment had a lot of problems, due to instabilities of half-finished structures.

There exist different approaches for self interlocking blocks: Shikalgar and Kate [8] propose a system with small hooks at the ends of each block to connect to the next one, and two elevations on top to lock with the notches on the bottom of the upper block. However, they had not only problems with the vertical interlocking parts, but also noticed that the blocks are easily damaged on the interlocking parts, since they are a lot thinner than the block itself. Another self interlocking block is the SILBLOCK [9], which uses different blocks with hooks (Fig. 2.1). But this system can only build walls, despite being more complex than SL-blocks, considering the number of block types. There are eight block types needed, and no structures more complex than a T-joint can be built.

2.2 Automated Assembly

Robots currently struggle with the autonomous assembly of self-interlocking blocks, since it is a very contact rich and occlusion heavy task. There are many works on peg-in-hole tasks. Xu et al. [3] surveyed two of the most used strategies, namely model based and model free learning. They conclude that model based algorithms are safer, but model free methods have better generalizing capabilities. But since both these strategies rely on contact to the surroundings of the hole, they are not suited for more complex assembly tasks.



STRETCHER UNITS
JAMB UNIT
CORNER UNIT
FIG. 2. Details of SILBLOCK-1 System
STRETCHER UNITS
CORNER UNIT
JAMB UNIT
FIG. 3. Details of SILBLOCK-2 System

Figure 2.1: The SILBLOCK system uses four block types per hook direction (inwards/outwards), resulting in eight types total. Image by Anand and Ramamurthy [9]

There are other works that manage to insert gears or electrical plugs [4, 10], but most of them still maintain contact to the surroundings of the hole. Spector, Tchuiev, and Di Castro [5] avoids the contact with the use of two wrist mounted cameras. It is explained in detail in Section 3.2.

Yamada, Collins, and Posner [11] try to solve the problem of obstacles in the working area of the robot, and manage to get satisfying results by using one fixed camera to locate the obstacles, as well as another, wrist mounted camera for the actual insertion task.

There is also work on the use of probabilistic movement primitives to assemble blocks. Carvalho et al. [12] manage to assemble blocks from the Ubongo3D game. Apolinarska et al. [13] assemble timber structures with their reinforcement learning approach, but struggle with multi-engagement tasks.

3 Background

For our work, we mainly need three different parts: the building blocks, the architecture to train for the assembly, and a better loss function. As building blocks, we use the SL-blocks, which are described in detail in Section 3.1. The architecture we show in Section 3.2 is called InsertionNet2.0, and was mainly used for electrical plugs. As we show in Chapter 5, the rotation representation used in InsertionNet2.0 is vulnerable to local optima. We replace it with a better representation, which is explained in Section 3.3.

3.1 SL Blocks

The SL block was first proposed by Shih [1]. It consists of two tetracubes, shaped such as an S and an L, as shown in Fig. 3.1.

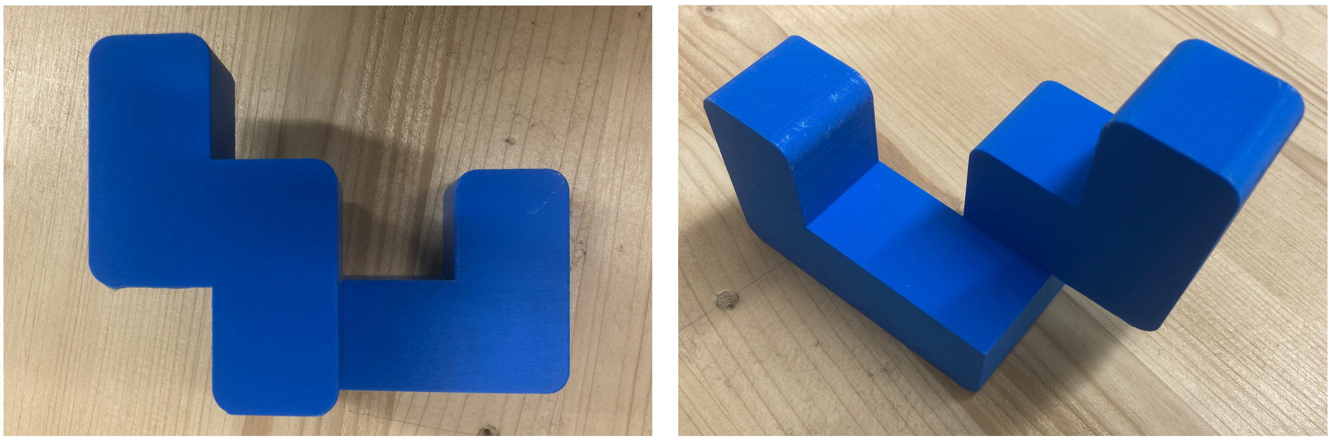


Figure 3.1: The 3D printed SL-block. The left image shows it from the side, such that the S and L is clearly visible. The right image shows the block from an angle to show the 3D structure of the block.

The SL blocks can be assembled in various ways, as Shih show in the extension of their work [2]. There are ten different types of engagements (see Fig. 3.2), proposed in [1], which are reduced to six with the introduction of conjugate pairs in [2].

The assembly of complex structures has challenges that exceed the extent of this work, like needing to assemble groups of SL-blocks, which are afterwards assembled as one [7]. To avoid this multi-block assembly, we print one h-engagement as one block (see Fig. 3.3) and use it as a base for further assembly in Chapter 4.

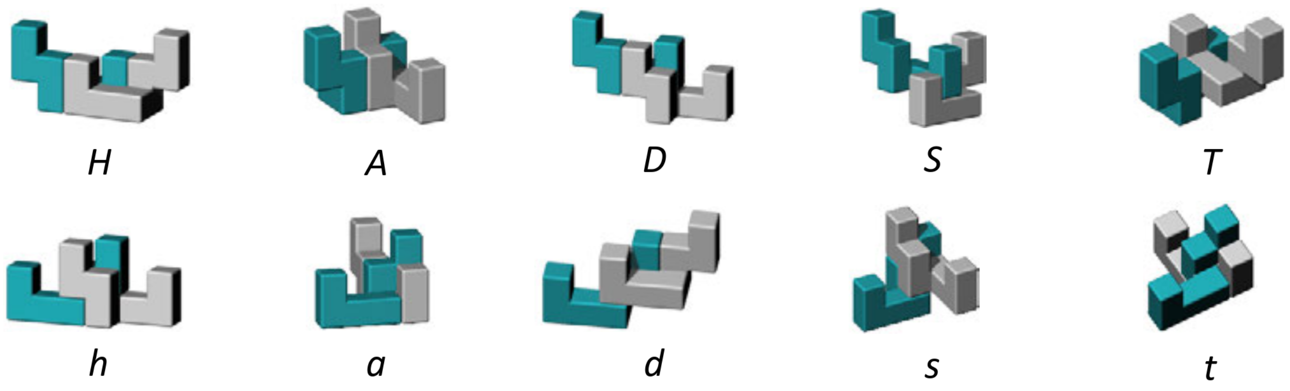


Figure 3.2: The 10 types of engagements. We focus on the h-engagement. (Image by Shih [1])

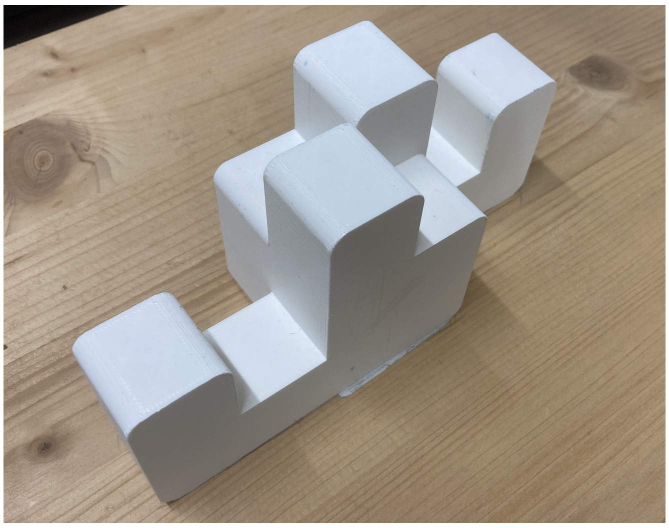


Figure 3.3: The 3D printed SL h-engagement. It was printed at once instead of assembling two SL-blocks. We use it as a socket for our SL-block.

3.2 InsertionNet

This thesis is mainly based on InsertionNet 2.0 by Spector, Tchuiev, and Di Castro [5], a neural network (NN) to insert different plugs with minimal contact to the surface around the hole. InsertionNet2.0 is the improved version of InsertionNet 1.0 [10]. The first version tries to solve the same task, but with only one camera and simpler network architecture. Because of this, it relies more on contact between the plug and the surroundings of the hole, to measure the force and its direction. Version 2.0 is not only faster and more accurate than version 1, it is better suited for contact rich task, since it reduces the contact to a minimum, by utilizing a second camera.

3.2.1 Definitions and Setup

Spector, Tchuiev, and Di Castro [5] use a UR5e robot arm [14] with six Degrees of Freedoms (DoFs). As sensors, they use two Intel RealSense D435 cameras [15] mounted on the wrist of the arm. The cameras are tilted in a $45°$ angle to focus on the fingers of the end effector (EE).

The two resulting RGB images are stacked in channels, resulting in a single $\text{Img} \in \mathbb{R}^{H\times W\times 6}$. Additionally, Spector, Tchuiev, and Di Castro [5] use the force $F = (f_x, f_y, f_z) \in \mathbb{R}^3$ and momentum $M = (m_x, m_y, m_z) \in \mathbb{R}^3$. A complete observation is then defined as $O = (\text{Img}, F, M)$.

The EE is a gripper, with the pose $L = [L_{\text{pos}}, L_{\text{ang}}]$ where $L_{\text{pos}} = (x_0, y_0, z_0)$ describes its location and $L_{\text{ang}} = (\theta_0^x, \theta_0^y, \theta_0^z)$ describes its orientation. An action $\Delta = (\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y, \Delta\theta_z)$ is the correction of the pose needed, w.r.t. the current position.

The wrist-mounted cameras help to avoid occlusions and can use symmetries, opposed to fixed cameras, as they already showed in their first version [10]. The second camera not only enables them to extract depth information, it also solves the ambiguity of a single image and helps against occlusions.

They use 16 different insertion and threading tasks, including different electrical plugs, to train and test the method.

3.2.2 Data Collection

For the data collection, Spector, Tchuiev, and Di Castro [5] use Algorithm 3.1. First, each plug is inserted in the final position L_{final}, which is recorded. Then, for each iteration, two poses are randomly selected: one high, above the hole, and one low, at the height of the hole. The position and orientation is sampled from a normal distribution, the height is constant for the low poses and uniform sampled for the high poses. Next, the robot moves to the high pose, records the observation and the action needed to reach L_{final}. This action is simply the difference $\Delta_{\text{correction}} = L_{\text{final}} - T_{random}$ where T_{random} is the current random pose. Then the robot repeats this with the low pose, before moving to the next iteration.

This results in more data points close to the hole and less far away. Also, due to half the points being high, we can predict the action, without touching the hole. The collected data is augmented with random visual augmentations before training, to improve the robustness of the model. The augmentations used are resize, random crop, color jitter, translation, rotation, erase and random convolution. The Force and Momentum are normalized, such that only the direction is stored.

Algorithm 3.1: Data Collection, by InsertionNet2.0 [5].

```
Input: Maximum Randomized Location $r_0 = 10mm$
       Maximum Randomized Angle $\theta_0 = 10^\circ$
       Maximum Height $z_{max} = 50mm$
       Force threshold $F_{th}$
       Momentum theshold $M_{th}$
       Number of Datapoints $N$

Reaching time $t_{max}$
Buffer of Dataset $D$
Init: Insert plug into the hole, register final pose $L_{final}$
1  $(\overbrace{x_0, y_0}^{X_0}, z_0, \overbrace{\theta_0^x, \theta_0^y, \theta_0^z}^{\Theta_0}) = L_{final}$
2  for $i \leftarrow 1, \dots, N$ do
3      $p_X^0, p_X^1 \sim \mathcal{N}(X_0, r_0)$
4      $p_\Theta^0, p_\Theta^1 \sim \mathcal{N}(\Theta_0, \Theta_0)$
5      $z_1 \sim$ Uniform$[z_0, z_{max}]$
6      $T_{high}^i = (p_X^0, z_1, P_\Theta^0)$
7      $T_{low}^i = (p_X^1, z_0, P_\Theta^1)$
8      for $T \leftarrow [T_{high}^i, T_{low}^i]$ do
9          Move robot to $T$ using an impedance controller
10         while Robot did not reach $T$ do
11             if $timemod(t_{max}/P_{rc})$ equals $0$ then
12                 $E = (x, y, z, \theta_x, \theta_y, \theta_z)$
13                 $Img = cams.get\_images()$
14                 $F = robot.get\_force()$
15                 $M = robot.get\_momentum()$
16                 $Obs = (Img, F, M)$
17                 $Action = L_{final} - E$
18                 $Datapoint = (Obs, Action)$
19                 $D.append(Datapoint)$
20 return $D$
```

3.2.3 Network Architecture and Losses

The Insertion Net 2.0 is split into three parts: a ResNet18, written as ϕ, and two smaller NNs, the Delta Net and the Relation Net. They compose three models: Contrastive, Delta and Relation model (Fig. 3.4). The ResNet18 ϕ is trained on all three architectures at the same time, by adding the separate losses.

The Contrastive architecture is composed of ϕ and a second ResNet18 ϕ', which uses the same weights. It uses a Polyak average $\phi' = m\phi' + (1-m)\phi$ with $m = 0.999$ to update the weights. By taking the same image with different augmentations to let ϕ and ϕ' extract the features and using the InfoNCE loss [16]. The Contrastive loss works in the same way, as it does in CURL [17], to shrink the variance of the network and thus creating a more robust model.

The Delta architecture uses the same encoder ϕ to encode the images. It is extended through the Delta Net, a fully connected NN, which takes the encoded image together with the observation of force and momentum as an input, and learns to output the action $\Delta_{\text{correction}}$, but without the Δz (see Section 3.2.4). We note that in the original work, this $\Delta = (\Delta x, \Delta y, \Delta\theta_x, \Delta\theta_y, \Delta\theta_z)$ still includes the Euler angles.

The Relation architecture is built with two times the same encoder ϕ, which are both trained at the same time. Each one encodes its own image and puts the result into the Relation Net, which is similar to the Delta Net. It is fully connected and outputs an action without Δz, but since it takes two encoded images as input, the action is not $\Delta_{\text{correction}}$. Instead, it is the action to get from the first image to the second one: $\Delta_{rel} = \Delta_{\text{correction},2} - \Delta_{\text{correction},1}$. The Relation loss is used to help with one-shot learning and multistep insertions [18]. The images need to be from the same insertion task and have consistent augmentations. We note, like before, that it uses Euler angles to represent the Rotation.

3.2.4 Experiments and Results

InsertionNet 2.0 is tested by performing 16 different insertion tasks. They mainly consist of electrical plugs, like a power cord, being inserted into their corresponding socket. For this, the plug is placed in the gripper, like for the data collection. The gripper is placed at a random position above the desired position. They test both, the Delta or Relation policy, one at a time. The trained policy predicts the action to the final position. The Δz which is not predicted is filled in with a fixed value, such that the robot moves down the same distance each step. This way, they can correct for errors in the next step. The robot moves to the predicted position (with added Δz) before taking the next picture. This is repeated till the robot reaches the final height and the plug should be inserted.

Each task is tested 200 times, with near perfect results. The lowest success rate of all 16 tasks was 97.5%. Especially HDMI and VGA, two of the hardest tasks for InsertionNet 1.0 were drastically improved due to the low contact needed, the second camera and a more accurate controller for the robot.

They also used the AC 2-pin plug task to test the sample efficiency with policies trained on different sized datasets. They conclude that 150 samples are sufficient, since the success rate already reached 100% after 40 samples and the completing time almost stops decreasing at 150 samples, close to $0 sec$. They also test the impact of the Contrastive loss, confirming its usefulness for the Relation policy, which has an approximately 20% higher success rate with than without the contrastive loss, and less variance in the completion time. They tested the robustness to obstacles, strong grasping errors and illumination changes, showing that the Delta policy is a lot more robust than the Relation policy, while still needing training data with strong grasping errors to solve these test. The Relation policy on the other hand does not need additional data to solve a new task, as shown by their tests on transferability. After being trained

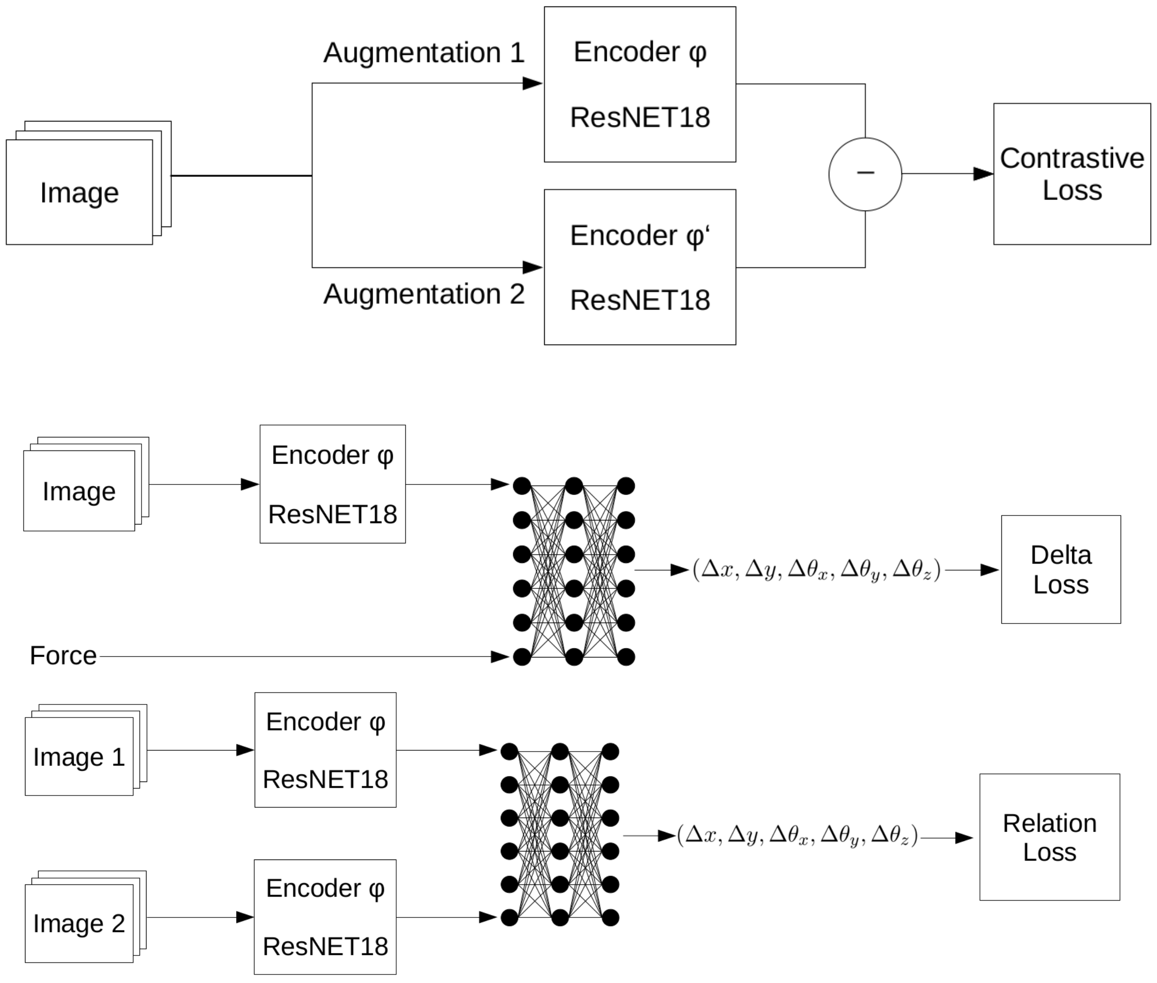


Figure 3.4: The three architectures: Contrastive, Delta and Relation. The Contrastive Model compares the encodings of one image with two different augmentations, the Delta Model predicts the action needed to reach the hole and the Relation Model predicts the difference in distance and rotation between two images.

on 15 of the 16 tasks, the Delta policy needs 20 samples to learn the last one, the Relation policy can solve it without additional data. But considering the 150 samples initially needed to learn the task, it is still shown, that the Delta policy generalizes and shares information between tasks. Additionally, the Relation policy managed a multistep insertion task, consisting of inserting a key into its lock, turning the lock, and turning back.

3.3 Continuous Rotation Representation

Like we mentioned before, InsertionNet2.0 uses Euler angles for their representation of rotations. In the following, we explain the problems that Zhou et al. [6] found with this representation.

3.3.1 Motivation and Definition of Continuous Representations

3D rotations can be represented in multiple ways, for example Euler angles or quaternions. Zhou et al. [6] investigate the impact of discontinuities in rotation representations on the learning of NNs, and propose a continuous representation [6].

With $SO(n)$ being the space of n dimensional rotations, defined as the group of $n \times n$ matrices M with $MM^T = M^T M = I$ and $\det(M) = 1$, they first look into $SO(2)$.

Any 2D rotation $M \in SO(2)$ can be represented with a single angle θ in the representation space $R = [0, 2\pi]$ (Or any equivalent set of angles). But any mapping $g : SO(2) \rightarrow R$ is not continuous: The upper and lower directional limits of the identity matrix give 0 and 2π respectively. Such discontinuous representations can be difficult for NNs to learn, according to Zhou et al. [6].

They therefore define a representation as a tuple of two mappings (f, g), where f maps from the representation space R to the original space X (for example $SO(2)$), while g is the inverse of f. Such a representation is said to be continuous, if g is continuous.

They then take a NN, that predicts values in the representation space. These outputs are passed through the mapping f, both in training and testing of the network. Therefore, if g would not be continuous, a connected set C in the original space could result in a disconnected set in the representation space and a discontinuous training signal for the NN. This cannot happen, as long as g is continuous.

3.3.2 Discontinuity Proof for other Representations

Before proposing a continuous representation, Zhou et al. [6] prove that not only Euler angle representations and Quaternions are discontinuous, but that every representation with four or fewer dimensions for 3D rotations is discontinuous. Thus, their proposed 5D representation, which we explain soon, is as low dimensional as any continuous representation for 3D rotations.

In topology, a homeomorphism is a continuous bijection with a continuous inverse. This is given for a continuous representation (f, g), since g is a continuous bijection from a compact topological space to a Hausdorff space, as we can restrict the codomain of g to $g(X)$ [19]. The inverse is f restricted to the domain $g(X)$. This implies, on the other hand, that if no subset of the representation space R is homeomorphic to the original space X, there is no possible continuous representation (f, g) on these spaces.

3D rotations $SO(3)$ are homeomorphic to $\mathbb{R}P^3$, the quotient space of $\mathbb{R}^4 \setminus \{0\}$ under the equivalence relation $x \sim \lambda x$ for all $\lambda \neq 0$. It is already known in topology, that $\mathbb{R}P^3$ embeds in $\mathbb{R}^5$, but not in $\mathbb{R}^d$ for any $d < 5$ [20]. By definition of embedding, there is no homeomorphism from $\mathbb{R}P^3$ (and therefore $SO(3)$) to any subset of $\mathbb{R}^d$ for any $d < 5$. Thus, there also is no such continuous representation.

3.3.3 Proposed Representation

The representation proposed by Zhou et al. [6] reduces a rotation matrix in its dimensionality and performs the orthogonalization. For $X = SO(n)$ as the original space, they choose $R = \mathbb{R}^{n\times(n-1)} \setminus D$ as the representation space (with D being the set that does not map back to $SO(n)$). The mapping g_{GS} simply drops the last column:

$$g_{GS}\left(\begin{bmatrix} | & & | \\ a_1 & \dots & a_n \\ | & & | \end{bmatrix}\right) = \begin{bmatrix} | & & | \\ a_1 & \dots & a_{n-1} \\ | & & | \end{bmatrix}$$

while f_{GS} uses a Gram-Schmidt-like process to recreate it:

$$f_{GS}\left(\begin{bmatrix} | & & | \\ a_1 & \dots & a_{n-1} \\ | & & | \end{bmatrix}\right) = \begin{bmatrix} | & & | \\ b_1 & \dots & b_n \\ | & & | \end{bmatrix}$$

$$b_i = \left[\begin{cases} N(a_1) & \text{if } i = 1 \\ N\left(a_i - \sum_{j=1}^{i-1} (b_j \cdot a_i)\, b_j\right) & \text{if } 2 \leq i < n \\ \det \begin{bmatrix} | & & | & e_1 \\ b_1 & \dots & b_{n-1} & \dots \\ | & & | & e_n \end{bmatrix} & \text{if } i = n \end{cases}\right]^T$$

where a_i and b_i are the column vectors and e_i are the canonical basis vectors of the Euclidean space, $i = 1, 2, \dots, n$. $N(\cdot)$ is a normalization function.

For $n = 3$, this results in an 6D representation, which they reduce further by using a normalized projection $P : \mathbb{R}^m \to \mathbb{R}^{m-1}$:

$$P(u) = \begin{bmatrix} \frac{v_2}{1 - v_1} & \frac{v_3}{1 - v_1} & \dots & \frac{v_m}{1 - v_1} \end{bmatrix}^T$$
$$v = \frac{u}{\|u\|}$$

and the stereographic un-projection $Q : \mathbb{R}^{m-1} \to \mathbb{R}^m$:

$$Q(u) = \frac{1}{\|u\|} \begin{bmatrix} \frac{1}{2}\left(\|u\|^2 - 1\right) & u_1 & \dots & u_{m-1} \end{bmatrix}^T$$

The matrix has to be reshaped to a vector for this. With $m = 6$, this gives us a continuous 5D representation for 3D rotations. Up to $n - 2$ normalized projections can be done to reduce dimensionality while keeping continuity, which is 1 for 3D rotations.

Zhou et al. [6] tested the representations empirically by training a NN over a fixed number of iterations and comparing the mean and max error and the standard derivation. The tests included a sanity test, a pose estimation for 3D point clouds and with inverse kinematics for human poses. Their 6D and 5D representations outperformed quaternions, axis angles and Euler angles on almost every category, except for the maximum error in the point cloud pose estimation, where every representation had a maximum error close to $180°$.

4 Methodology

Since we are handling SL-blocks instead of electrical plugs like in InsertionNet, we encountered different problems, that did not appear in the original work. To solve these problems, we needed to adapt the setup and training accordingly. In the following sections, we describe the problems and the changes we made to InsertionNet to solve them. In Section 4.1, we describe the differences in the task and the main problems resulting from them. Section 4.2 explains the changes in the setup we made to counter these problems, while Section 4.3 focuses on the changes in data collection and training.

4.1 Task Differences and Problems

Our main task is similar to the task of InsertionNet, but instead of inserting a plug into its socket, we try to assemble two SL-blocks. For this, we take a h-engagement (see Section 3.1) as the socket and fix it on the table. We then try to insert a single SL-block into the socket with the ends of the Ls interlocking (see Fig. 4.1). This configuration is not considered an engagement by Shih [1], because it opposes conjugates and cannot be used to create strands. But it serves as a proof of concept for us, so that we can investigate the different engagements as a future work. The insertion starts with the block already grasped and at a randomized position above the socket. Then it follows the same algorithm as in Section 3.2 to insert the block.

The SL-block is not only a lot bigger than the electrical plugs, resulting in much more occlusion for our task. Also, as described in Section 3.1, SL-blocks are self-interlocking. This naturally results in our task being extremely contact rich, which makes the insertion harder.

The combination of the size of the block and contact rich task causes another problem. The block in the gripper is subject to relatively big grasping errors, as soon as it touches the socket. Due to the size of the SL-block, touching the socket happens quite often during data collection and insertion.

4.2 Experimental Setup

We use a Franka Emika Panda robot arm (Franka) [21] instead of an UR5e robot arm. The Franka is one of the most widely used research robots, and it supports an impedance controller. Since the EE can be removed, we can hold the cameras with a 3D printed mount, which we install above the EE (Fig. 4.2).

Like in Spector, Tchuiev, and Di Castro [5], we use two cameras to get images of the scene. We change the angle from the original $45°$ to $55°$, since our EE is different. By changing the angle, we don't see the EE anymore, which was in the field of view before. They are fixed with a 3D printed mount (Fig. 4.3). This way, the SL-block in the gripper and its surroundings are always in the field of view, as shown in Fig. 4.4.

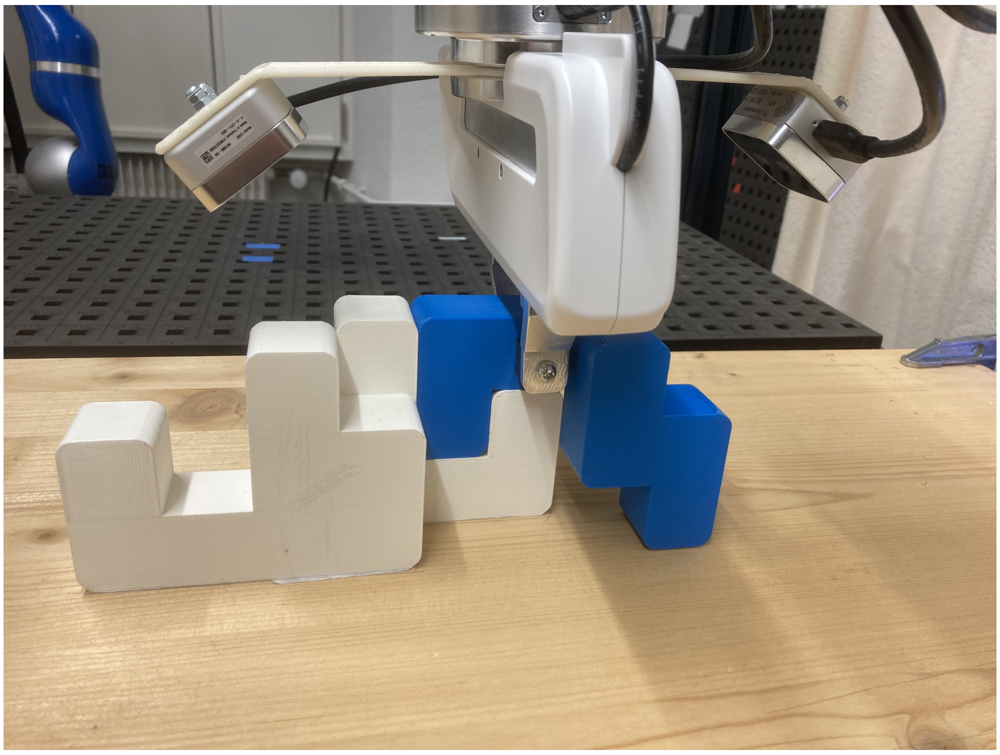


Figure 4.1: The SL socket (white) and the inserted SL-block (blue). The blue SL-block is still held by the robot. The L-part of the blue block connects to the white socket.

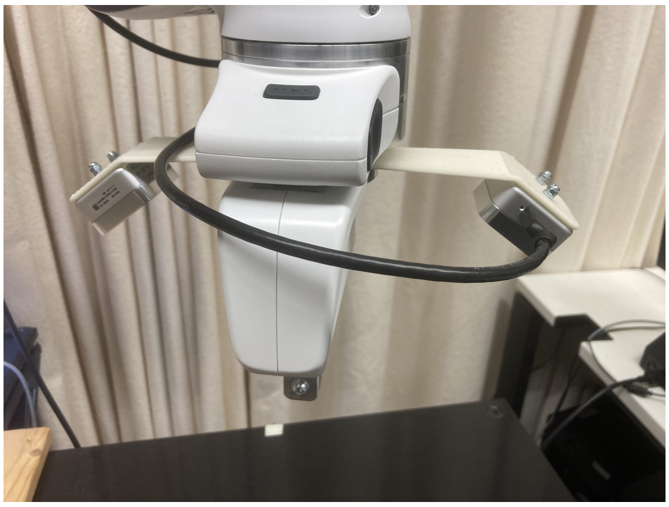


Figure 4.2: The D405 cameras mounted above the EE, looking between the fingers. The mount is 3D printed. The angle is set at $55°$, so that only the block and the fingers of the gripper are in view.

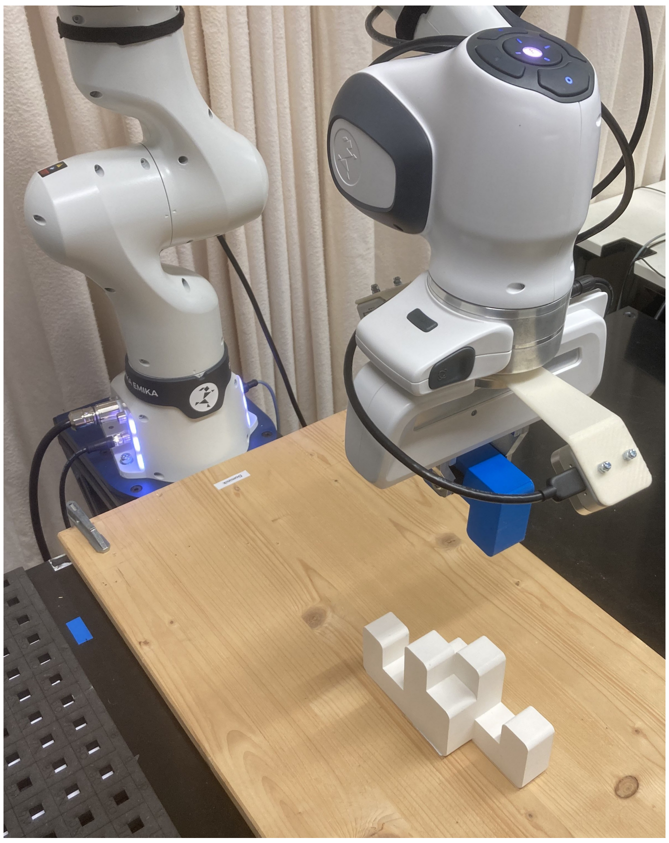


Figure 4.3: The complete setup: The Franka arm, the blue SL-block in the gripper, two cameras looking at it, and the white socket below. The second camera is partly hidden behind the gripper.

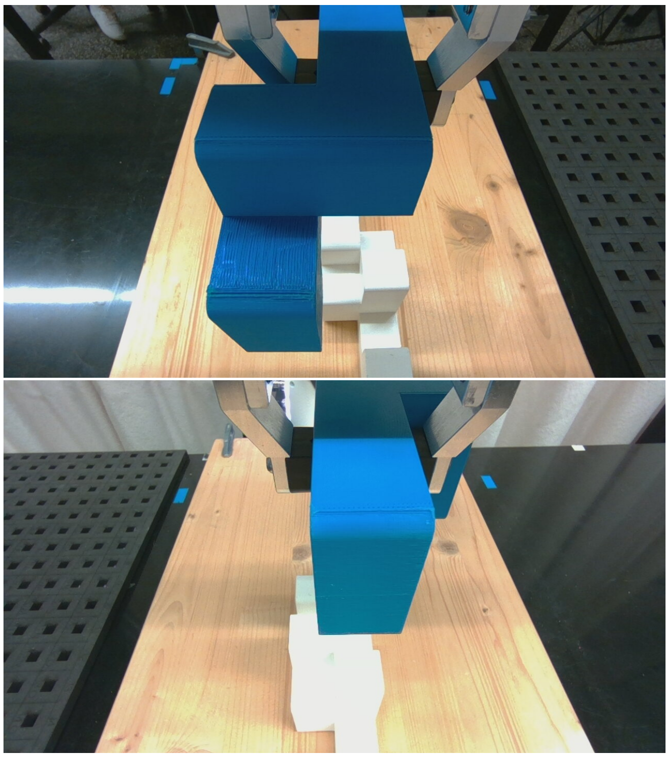


Figure 4.4: Field of view of the cameras. The blue and the white block are both visible, but not the gripper itself.

Instead of two RealSense D435 cameras [15], we use two D405 cameras [22]. The minimum distance for depth images is only 7 cm, opposed to the 28 cm of the D435, so we can use the built-in depth cameras. In InsertionNet 2.0, depth information was only indirectly extracted from the two images. Due to the increased occlusion in our task, this method might not be reliable enough. By using the more accurate depth camera of the D405, instead of relying on the network to extract the depth information, we simplify the task to learn and provide better training data. The image quality is also better than with a D435, since we mostly operate in distances from 10 cm to 50 cm from the camera to the socket. This is within the optimal range of the D405 (7-50 cm). The optimal range of the D435 on the other hand is 30-300 cm, which does not include the most important images, which are close to the socket.

4.3 Data Collection and Training

The data that we use for training is different in shape than the InsertionNet data. We use a different camera with additional depth perception, resulting in an image being $Img \in \mathbb{R}^{H \times W \times 8}$, with height $H = 720$ and width $W = 1280$. The 4 channels of each image stay together when the images are stacked, so the resulting stack has the channels RGBD RGBD. The force and moment are from $\mathbb{R}^3$ like before. They are measured with the built-in force estimation of the Franka.

The data collection algorithm stays mostly the same, but we needed to implement it from scratch, since there is no code published. We first put the SL block into the desired pose, while still grasped, and save the pose of the robot. Then, we move it to alternating high and low poses, like explained in Section 3.2, taking a picture each with both cameras and recording the current force and momentum. We also record the action needed to reach the final pose from each high and low pose.

Since SL-blocks are very large compared to the plugs used in InsertionNet, the SL-blocks collide often during this process. This results in large, accumulating grasping errors. To correct them, we stop after every 10 points, to start again. To correct the grasping error, we let the robot grasp the SL-block again, before inserting it and recording the new starting position. By repeating this, we create 150 data points $Datapoint = (Obs, Action)$.

The data points are shuffled and random 10% are used as test set. We use the test set to evaluate the model and its ability to generalize during training, to assure that we are learning our task and not overfitting. The data points are augmented with only resize and random crop, since color augmentations are not defined for 4 dimensional images. The implementation of extended versions of the other augmentations is left for future work.

InsertionNet 2.0 does not explicitly state the calculation of the Delta loss, and we have no access to the source code. Therefore, we assume the Delta loss to be an RMSE over the prediction and actual $\Delta_{\text{correction}}$.

Since we use the Delta architecture to actually insert the SL-block, we use a weighted sum of the three losses instead of a regular sum, i.e. $loss = 100l_{Delta} + 10l_{Contrastive} + 1l_{Relation}$. This helps the Delta network to learn faster, compared to the Contrastive and Relation networks, that are less important for our use case, since the Delta architecture is the one used for the final insertion.

4.4 Continuous Rotation Representation

Like explained in Section 3.2, InsertionNet2.0 uses Euler angles to represent the rotation. But like explained in Section 3.3, Zhou et al. [6] showed in their work that this is not a continuous representation, which makes it difficult for the network to predict the correct rotation. In our case, this results in local optima becoming a problem, as we explain in detail in Chapter 5. To counter this, we use the continuous loss representation from Zhou et al. [6].

The Contrastive architecture and loss stay the same, since the Euler angles are only used in the other two architectures. In both Delta and Relation architectures, we change the dimension of the output: instead of predicting $\Delta x \in \mathbb{R}^5$ (without Δz like explained in Section 3.2), we predict $\Delta x_{cont} \in \mathbb{R}^7$, still without Δz, but with 5 instead of 3 Dimensions for the rotation. We use the L2 distance as the loss, like Zhou et al. [6] proposed, and use it to train the Delta and Relation networks.

This helps the network to learn the desired function and helps to avoid local optima. This is shown empirically in Chapter 5, where we present the training losses of the different representations.

5 Empirical Evaluation and Results

In this chapter, we present the results and discuss their implications. Section 5.1 focuses on the losses and the results during training, before trying to perform an insertion. It shows the training losses of the different representations in different training runs. Each training run has the same color through all figures. Only Fig. 5.6, which shows train and test loss of the same run, uses brighter and darker versions of the same color. All graphs are smoothed for better visualization. We use the graphs to show empirically that our continuous representation improves the average error in distance by a factor of ten, due to the ability to escape local optima.

Section 5.2 provides the results of the insertion task with the real robot, described in Section 4.1. We also look at possible reasons and solving approaches for the problems occurring.

5.1 Training Results

With the initial Euler angle representation, as proposed by InsertionNet2.0 [5], we sometimes see an increase in the Delta loss after only around 500 epochs, followed by a continuously high loss (see red line in Fig. 5.1). This increase happens in about 40% of the training runs, the other 60% show a normal decrease (see blue line in Fig. 5.1), although it also stops learning after a short time. This behavior can be caused by a local optimum, in which our model gets stuck.

But if we change the rotation representation to be continuous, like the one proposed by Zhou et al. [6], we don't see the increase anymore. We still get into local optima sometimes, but we manage to leave them, opposed to when using the Euler angle representation (see Fig. 5.2). In some runs, we get into a second and sometimes even third local optimum, after leaving the first one (see orange loss). But they are not permanent, and we manage to leave them. The training ends after 8000 Epochs, when the difference in local optima gets very small.

Since we changed the representation, the units and dimensions of the loss change, too. Therefore, we plotted the losses separately and did not add labels to the y-axis. But the changed dimension prevents a quantitative comparison between the losses trained on different representations. We need to ensure that the model with the continuous representation is not just escaping local optima that it caused itself and that are worse than the local optimum in which the Euler angle representation gets stuck. To do this, we recorded the average Euclidean distance between predicted and correct pose during training (see Fig. 5.3).

When comparing the two rotation representations, we notice that the first local optimum of the continuous representation has almost twice the error of the local optimum of the Euler representation. But since the continuous representation can leave the local optimum, the second one has almost the same error as the local optimum of the Euler representation, at around 2 cm. However, the continuous representation

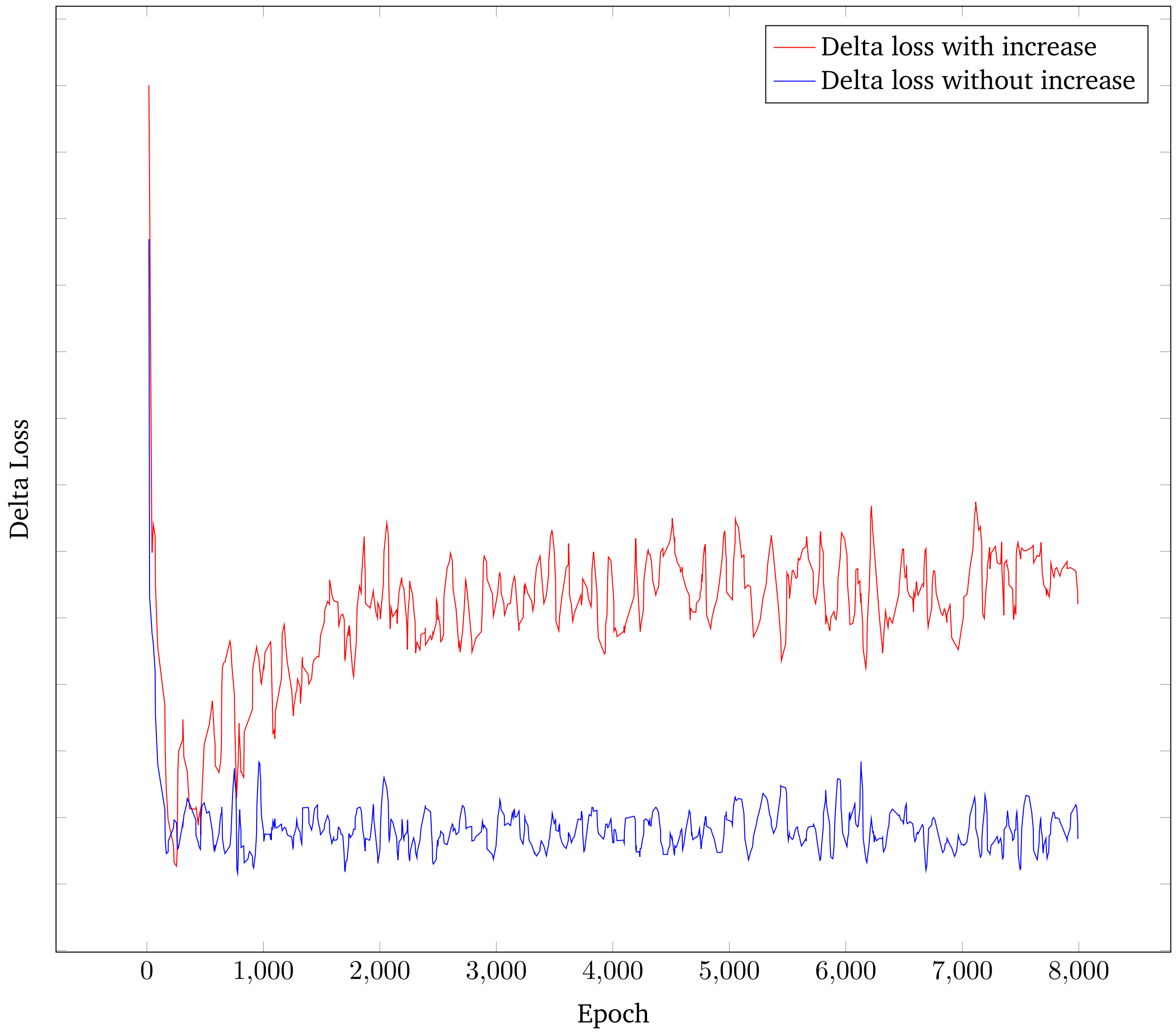


Figure 5.1: The Delta loss when using Euler angle representation. In the red case, where the loss increases again before converging, we expect it to be a local optimum. The blue run does not increase, but it stops decreasing fast.

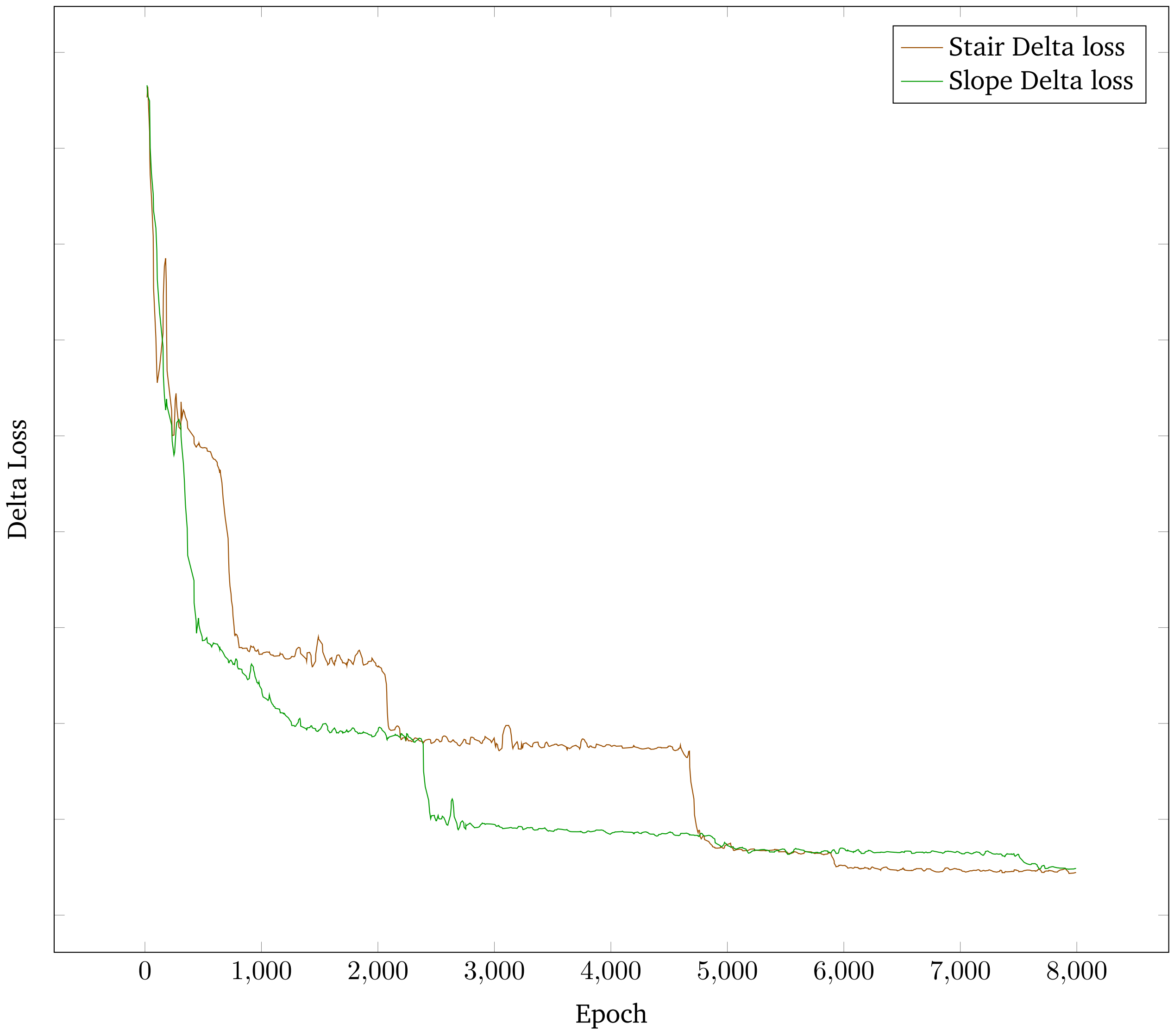


Figure 5.2: The Delta loss when using continuous representation. The orange loss shows multiple local optima, each better than the one before. Through the steep decline after escaping a local optimum, it shows a stair-like decrease. The green loss manages to avoid the first local optimum, in which the orange one gets stuck, and continues to decrease in a slope. Although it gets stuck in the second local optimum of the orange loss for a short time, both losses manage to leave it and continue to decrease (though mostly in steps) till the end of training after 8000 epochs.

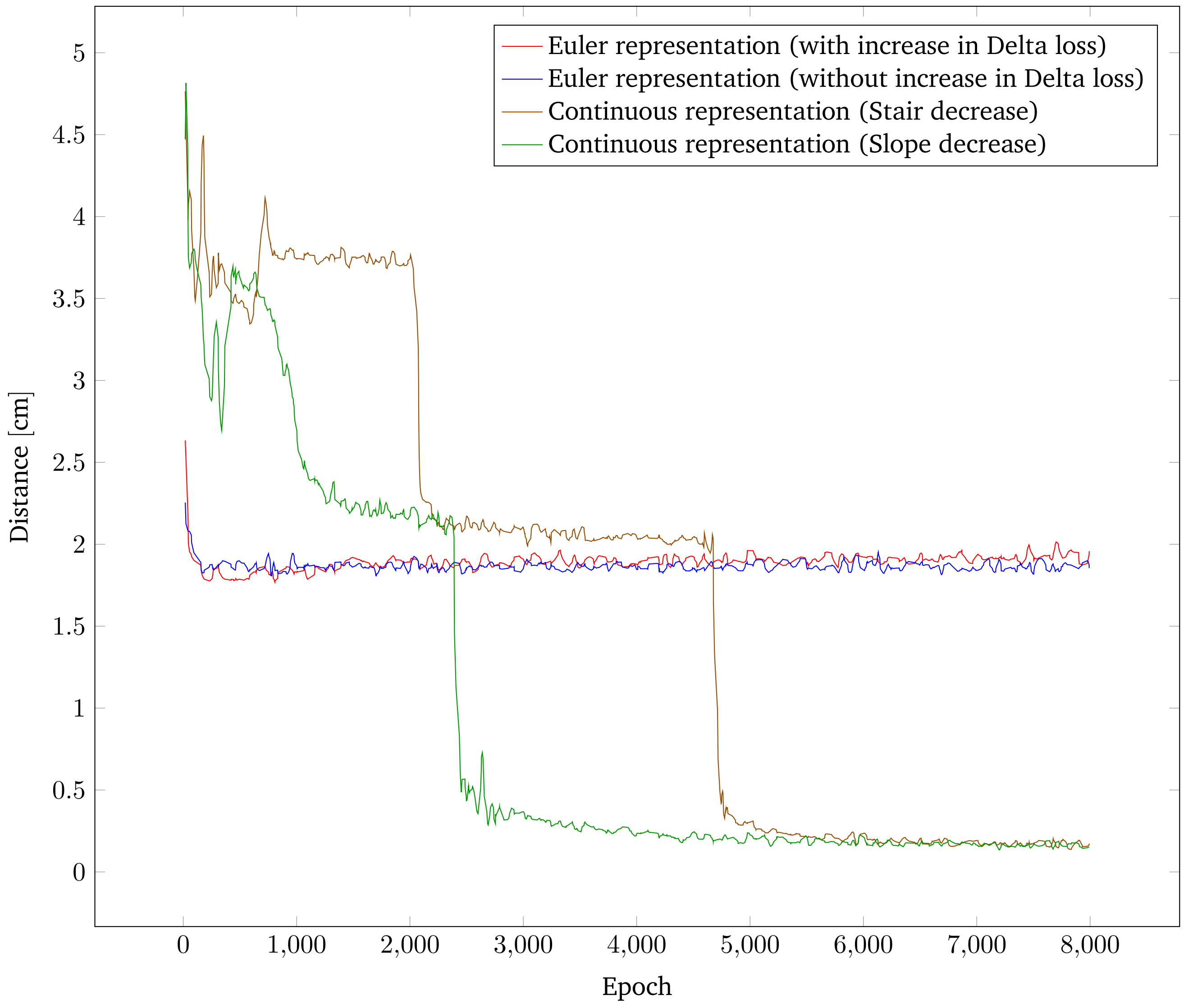


Figure 5.3: The error in distance during training. The two runs of the Euler representation show almost no difference. The orange run of the continuous representation gets stuck at in a local optimum at first, that is worse than the one of the Euler representation. But after it leaves this local optimum, it manages to reach the same and later a smaller distance error than the Euler representation. The green run, also with the continuous representation, manages to avoid the first local optimum, like it did in the Delta loss (compare Fig. 5.2). Both reach the same value.

also manages to escape this local optimum, resulting in an improvement to below 2 mm. The Euler representation loss that did not increase in Fig. 5.1 is only slightly lower than the one that increased. The difference in error is less than 2 mm, which implies that the increase probably happened mainly in the rotation prediction (since the loss only contains information about position and rotation). Since we changed the representation of that prediction, it would be difficult to compare these errors.

When using the Euler representation, the Delta loss shows no signs of decrease till 8000 epochs after the local optimum is reached. In opposite, it even shows a slight increase. The continuous representation on the other hand leaves the local optimum after around 5000 epochs (considering that the second local optimum has the same distance error as the Euler representation).

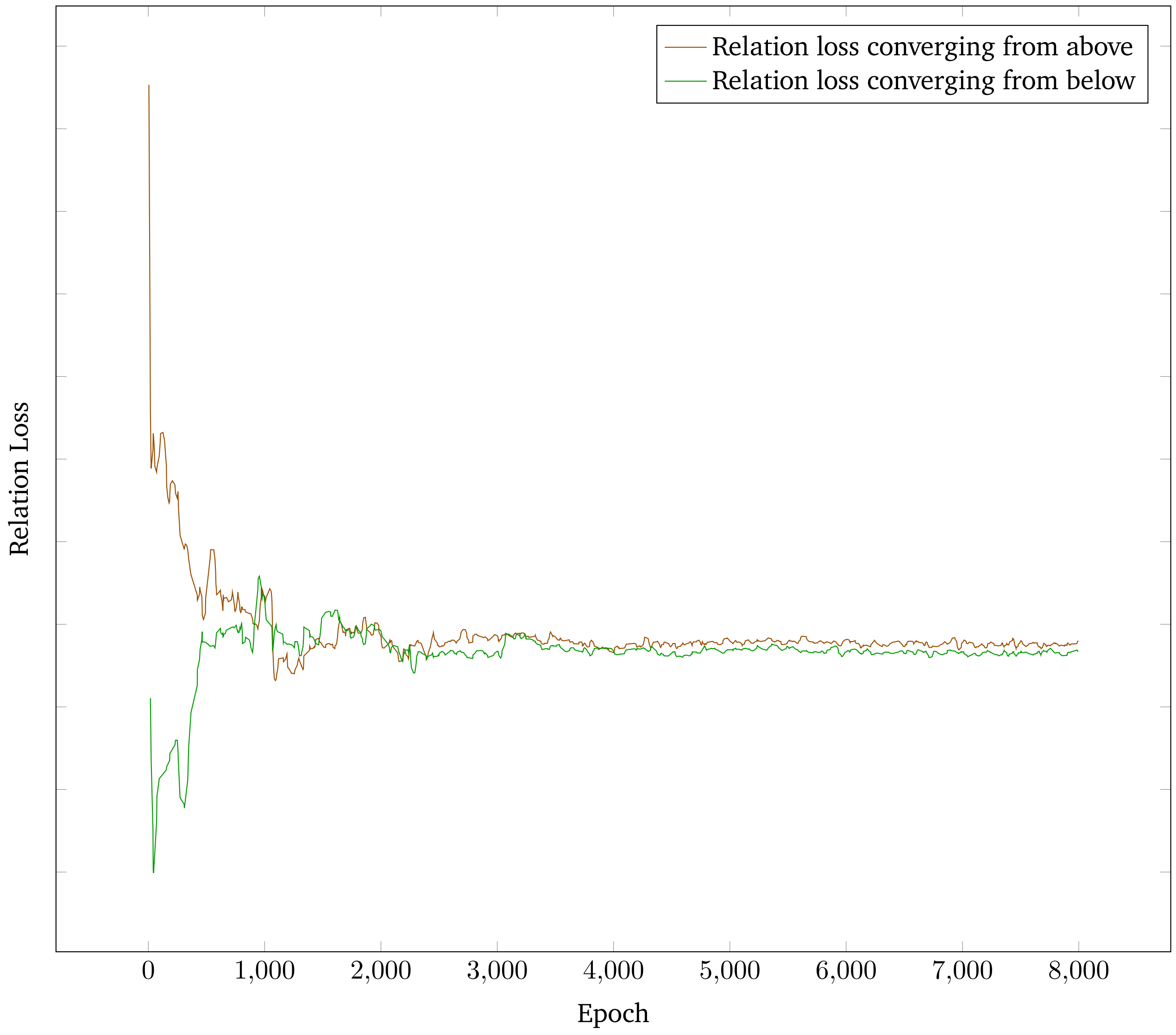


Figure 5.4: The Relation loss converging towards the same value in different runs.

The Relation loss shows an interesting behavior. Instead of decreasing, it seems to converge towards a fixed value (see Fig. 5.4). Since it always converges to the same value, the behavior is unlikely to be the result of a local optimum, since only starting parameters close to the optimum would get stuck. This behavior could be due to the low influence we give to the Relation loss, compared to the Delta or Contrast loss. It is possible that the optimal solution for the Delta loss is not optimal for the Relation loss, and thus increases the Relation loss to decrease the Delta loss. But it would result in the observed behavior, that the model converges towards a solution with a small Delta loss and a fixed, higher Relation loss. We plan to investigate this behavior further in future work and test our theory.

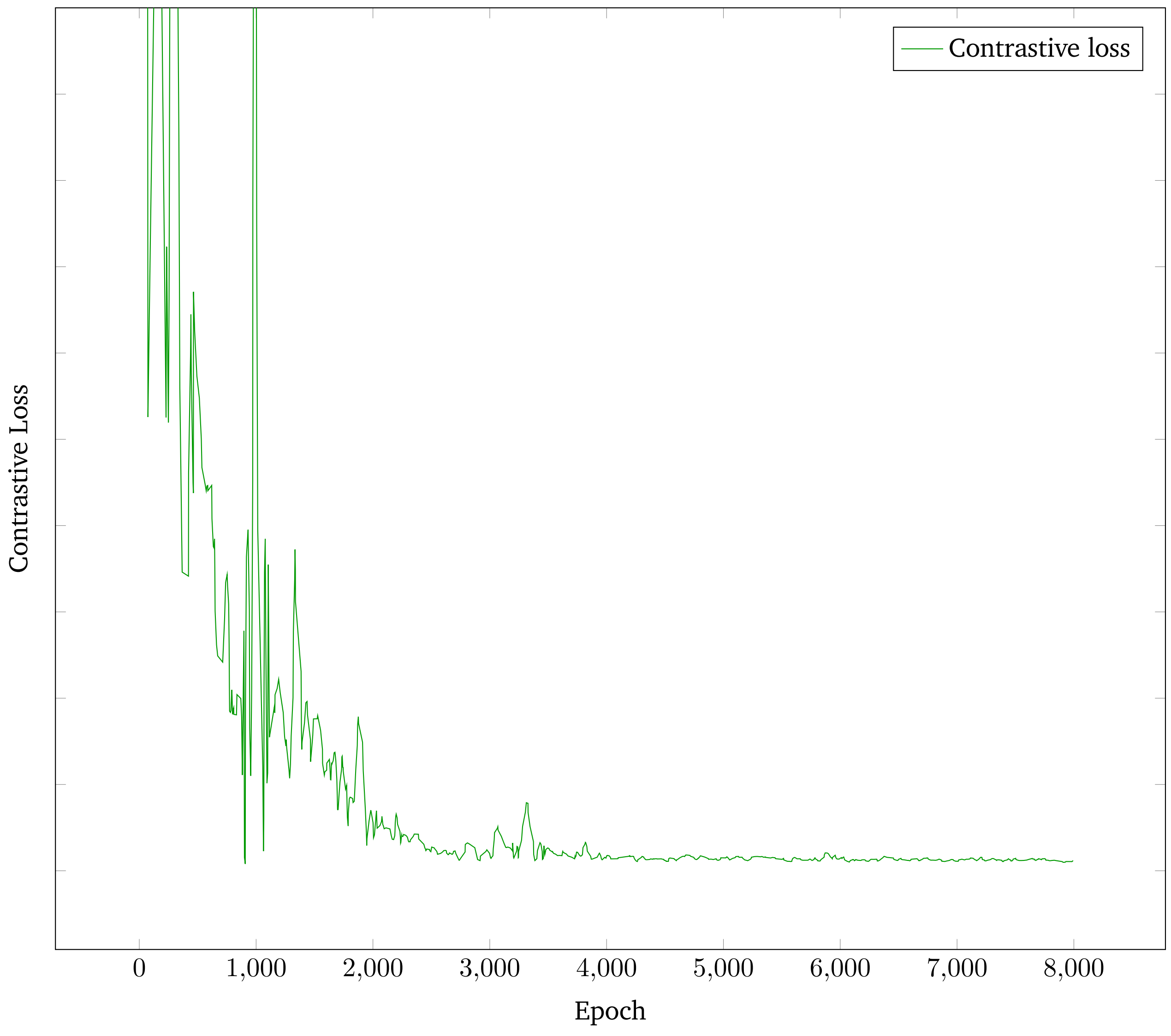


Figure 5.5: The Contrastive loss shows only a steep decrease in all runs.

The Contrastive loss shows just a steep decrease without anything noteworthy (see Fig. 5.5). The weighted average over the three losses, as explained in Section 4.3, is decreasing on both training and test set, though naturally the training set has a stronger decrease (see Fig. 5.6). The test loss also has more uncertainty than the training loss. Since the test loss also decreases, the model is not overfitting much and has good generalization abilities.

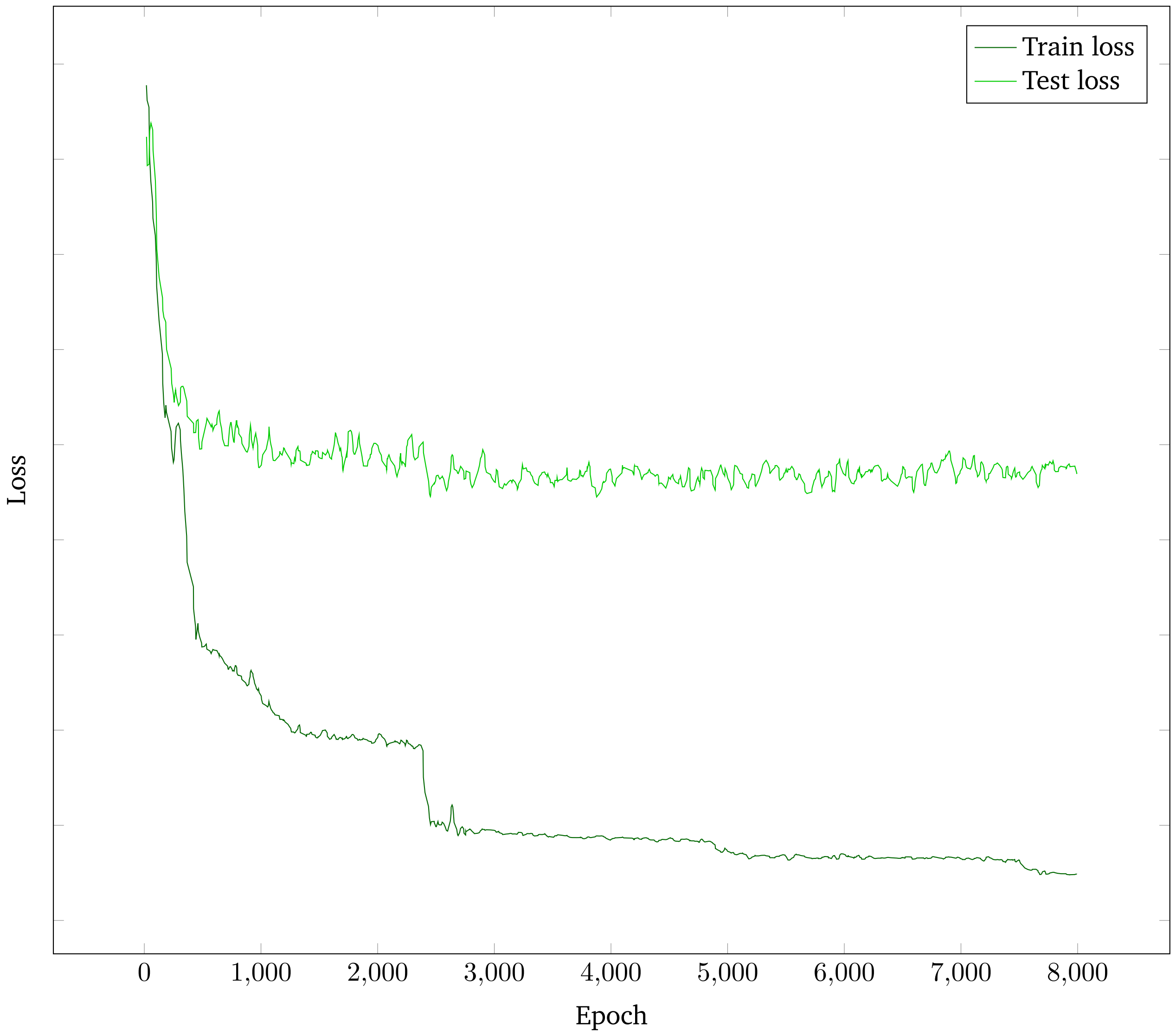


Figure 5.6: The training loss is a lower than the testing loss. The model is still generalizing, as can be seen by the sharp decrease in both losses at around 2500 epochs. The test loss shows less decrease, but it is still noticeable.

5.2 Insertion Results

We test the insertion with different trained models. The models are trained with the continuous Delta loss. The insertion failed in all tests. We investigated the errors, and tried to improve our method.

First, we noticed that most models have a bias in the same direction. In that direction, there was a sheet of paper with a checkers board printed on, attached next to the robot. This paper was put on the table after the data collection process and does not show up in the training data. It is in view of the cameras and thus influences the output of the network. This change in the environment was first unnoticed. The error not only moves the robot in that direction, but also moves the robot exactly over the checkers board, as soon as the white socket is out of view. Therefore, we assume, that the policy confused the white parts of the checkers board with the socket of the same color. As a countermeasure, we removed the checkers board to restore the initial environment. This reduced the error drastically, down to a few centimeters. If the error gets so large that the socket leaves the field of view, the error still increases drastically. This shows that our model is very susceptible to environment changes and occlusions of the socket. But it also shows that the socket is recognized, and the network tries to stay close to it.

To reduce the influence of environment changes, we trained again, with cropped images (see Fig. 5.7). Everything left and right of the wooden board would be cropped and not in view anymore. But this did not have the desired effect. The models trained on the cropped images would instead move to positions far from the socket and produce large errors, even though their loss implied otherwise. This happens, like before, whenever the socket leaves the field of view — which is more often with cropping, since the field of view is reduced. This indicates that we have a bias in our data collection process, since we keep the socket always in view of the cameras, which in not ensured in the insertion test.

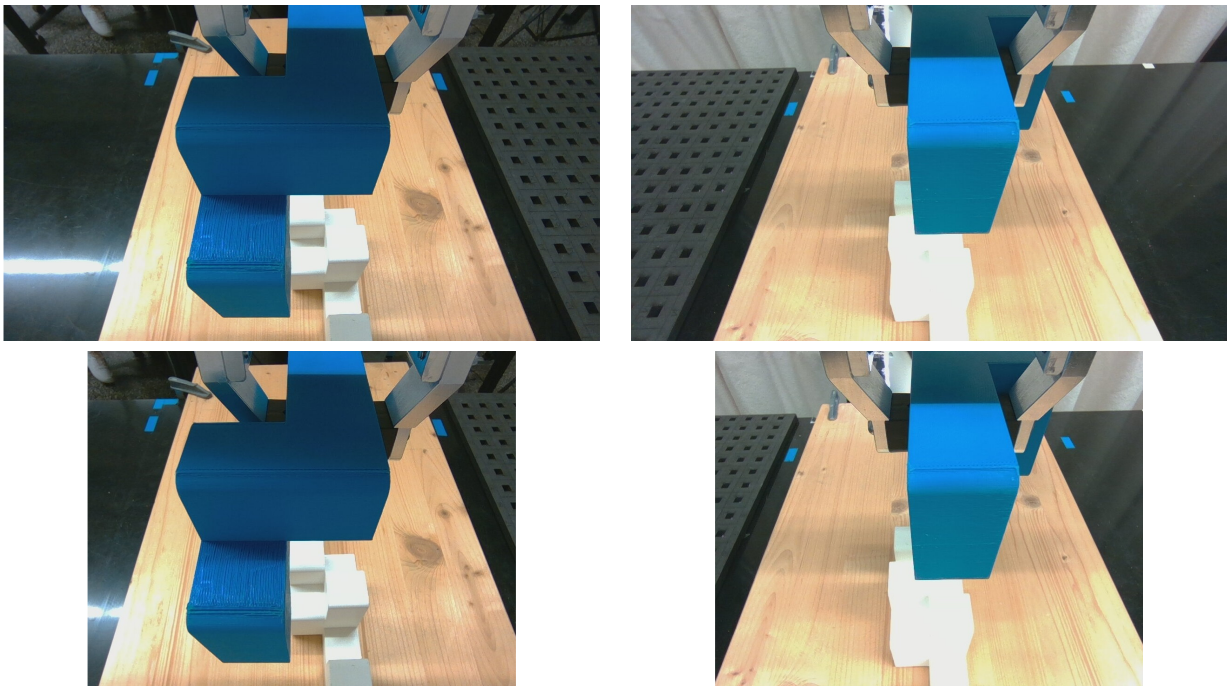


Figure 5.7: The field of view before and after cropping. The cropped image contains less unnecessary information. Further cropping would result in removing the information about the socket, if the EE is not centered above the socket but tilted.

6 Discussion

Even though we did not succeed in our task to insert the SL-block, we got valuable insights as to how we can improve our work in the future. Section 6.1 summarizes our contributions and improvements. The remaining problems on the other hand are summarized in Section 6.2, where we also propose approaches to solve them, which we will try to implement in the future work. Possible extensions of this project are described in Section 6.3.

6.1 Contributions

We managed to adapt InsertionNet2.0 to insertion of SL-blocks, which is more difficult than the original insertion task used in InsertionNet2.0. This does not only include the complete implementation of data collection, training and insertion tests, but also changing the hardware (i.e., the robot and cameras) to better fit our needs, and changing the model accordingly. Our implementation is learning the task, even though not accurately enough.

We improved the initial InsertionNet2.0, by changing the representation used for the 3D rotations. Using a continuous representation instead of Euler angles helps us to escape local optima during training. Due to this, we get far better training and testing losses. Our training also gets more reliable, since no training runs get stuck in local optima.

6.2 Remaining Problems and Possible Solutions

Even though we made considerable improvements on the initial InsertionNet2.0, we did not manage to perform a successful insertion. There are mainly two remaining problems.

First, the model is not accurate enough. We did not manage to get the final error below 3 cm in most test runs. If the error gets too large, the socket moves out of the view of the camera. As soon as this happens, the error increases drastically, since the network is trained to find the socket, and an image without a socket is unseen data. To reduce the error, we could increase the number of data points compared to InsertionNet2.0, since they had more than one task, for which they collected the data points. So if we only have one task, we still have significantly less data if we still use the same number of data points as they do. Also, we could shrink the variance of the poses where we take photos, which would add some bias, but would increase the accuracy due to the training data being closer to the actual target. This could make the second problem worse, but if we manage to solve that problem, the effect could be negligible.

Second, the model is vulnerable to environment changes. Small changes in the background, especially white rectangular things, can be confused with the socket. This results in larger errors in that direction, until the socket is out of view and the background object is targeted as the socket. To avoid this, one could define 4D versions of the augmentations that Spector, Tchuiev, and Di Castro [5] used, so that we augment not only RGB images, but also the depth channel. If we implement the 4D versions and use them to augment our RGBD images, we can increase the robustness of our policy to environment changes.

6.3 Outlook

When the remaining problems are solved, we would like to extend our work to more complex assembly tasks.

The first step would be training a policy to assemble all different SL-engagements. This could either be done by modeling them as different insertion tasks, or by adding a parameter for the desired engagement. The first approach has the problem that all engagements use the same SL-blocks, so it would be difficult or impossible to differentiate them. The second approach has the problem that we would have to train a model for every engagement, which results in longer training time. The longer training time could possibly be reduced by using the same ResNet18 for all models, similar to the core idea of InsertionNet2.0, but with more than 3 models.

With the ability to assemble different engagements, we plan to try to assemble larger structures autonomously, using the work of Liu, Liu, and Charkhab [7] to create assembly sequences. We would still have to solve the problems they mention, for example the support for parts of the structure that are not stable till the construction is complete.

Bibliography

[1] Shen-Guan Shih. “On the hierarchical construction of SL blocks.” In: *Proc. Advances in Architectural Geometry* (2016), pp. 124–136.

[2] Shen-Guan Shih. “The art and mathematics of self-interlocking SL blocks.” In: *Proceedings of Bridges 2018: Mathematics, Art, Music, Architecture, Education, Culture*. 2018, pp. 107–114.

[3] Jing Xu et al. “Compare contact model-based control and contact model-free learning: A survey of robotic peg-in-hole assembly strategies.” In: *arXiv preprint arXiv:1904.05240* (2019).

[4] Gerrit Schoettler et al. “Meta-reinforcement learning for robotic industrial insertion tasks.” In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 9728–9735.

[5] Oren Spector, Vladimir Tchuiev, and Dotan Di Castro. “InsertionNet 2.0: Minimal Contact Multi-Step Insertion Using Multimodal Multiview Sensory Input.” In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 6330–6336. DOI: `10.1109/ICRA46639.2022.9811798`.

[6] Yi Zhou et al. “On the continuity of rotation representations in neural networks.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5745–5753.

[7] Binqun Liu, Yuxi Liu, and Mehrzad Esmaeili Charkhab. “A Digital Framework for Interlocking SL-Blocks Assembly with Robots.” In: ().

[8] Junaid N Shikalgar and Rahul G Kate. “Experimental Study and Analysis of Self-Interlocking Blocks.” In: *International Research Journal of Engineering and Technology* 7 (2020), pp. 5445–5449.

[9] KB Anand and K Ramamurthy. “Development and performance evaluation of interlocking-block masonry.” In: *Journal of Architectural Engineering* 6.2 (2000), pp. 45–51.

[10] Oren Spector and Dotan Di Castro. “InsertionNet - A Scalable Solution for Insertion.” In: *IEEE ROBOTICS AND AUTOMATION LETTERS* 6.3 (2021), pp. 5509–5516.

[11] Jun Yamada, Jack Collins, and Ingmar Posner. “Efficient skill acquisition for complex manipulation tasks in obstructed environments.” In: *arXiv preprint arXiv:2303.03365* (2023).

[12] Joao Carvalho et al. “Residual robot learning for object-centric probabilistic movement primitives.” In: *arXiv preprint arXiv:2203.03918* (2022).

[13] Aleksandra Anna Apolinarska et al. “Robotic assembly of timber joints using reinforcement learning.” In: *Automation in Construction* 125 (2021), p. 103569.

[14] Teradyne Robotics (Germany) GmbH. *UR5e - der flexible Leichtbauroboter von Universal Robots*. 2024. URL: `https://www.universal-robots.com/de/produkte/ur5-roboter/` (visited on 04/04/2024).

[15] Intel Corporation. *Depth CameraD435*. 2024. URL: `https://www.intelrealsense.com/depth-camera-d435/` (visited on 04/04/2024).

[16] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation learning with contrastive predictive coding.” In: *arXiv preprint arXiv:1807.03748* (2018).

[17] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. “CURL: Contrastive Unsupervised Representations for Reinforcement Learning.” In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 5639–5650. URL: `https://proceedings.mlr.press/v119/laskin20a.html`.

[18] Flood Sung et al. “Learning to Compare: Relation Network for Few-Shot Learning.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[19] Czes Kosniowski. *A first course in algebraic topology*. Cambridge University Press, 1980.

[20] Donald M Davis. “Embeddings of real projective spaces.” In: *Bol. Soc. Mat. Mexicana (3)* 4 (1998), pp. 115–122.

[21] Franka Robotics GmbH. *Franka Research 3*. 2024. URL: `https://franka.de/research` (visited on 03/17/2024).

[22] Intel Corporation. *Depth CameraD405*. 2024. URL: `https://www.intelrealsense.com/depth-camera-d405/` (visited on 04/04/2024).