Learning Tactile Representations from Vision-Based Tactile Sensors for Robotic Architectural Assembly

Lernen taktiler Repräsentationen von kamerabasierten taktilen Sensoren für robotische architektonische Montage

Master thesis by Frederik Wegner

Date of submission: September 4, 2021

1. Review: Boris Belousov
2. Review: Niklas Funk
3. Review: Dr. Roberto Calandra
4. Review: Prof. Jan Peters, Ph.D.

Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Abstract

The tactile feedback from our human sense of touch enables us to do precise contact-rich manipulation that outperforms the capabilities of state-of-the-art robotics. Advancement in the development of tactile sensors yielded visual tactile sensors that provide high-resolution tactile feedback. While other areas of robot learning can use simulations to progress rapidly, this is not the case for tactile manipulation as correct and fast modeling of contact physics remains a challenge. We present Assembly Gym, a framework that enables quick development of tactile contact-rich assembly tasks for a physical robotic setup involving a robotic manipulator with visual tactile sensors and object tracking. With Assembly Gym, a challenging assembly task is set up and a large dataset of assembly episodes is sampled. Using autoencoders we learn an unsupervised representation of the tactile sensor images of our dataset. We show that such representations are task-relevant and can be used to learn the transition function of a complex multi-modal environment. By carrying out numerous experiments, we provide insights into the key aspects and parameters that can make representation and model learning successful. Finally, the learned transition model of the environment is combined with a trajectory planner for model predictive control.

Zusammenfassung

Die taktile Rückmeldung unseres menschlichen Tastsinns ermöglicht uns präzise, berührungsintensive Manipulationen, die die Möglichkeiten der modernen Robotik übersteigen. Fortschritte in der Entwicklung von Tastsensoren haben zu visuellen Tastsensoren geführt, die hochauflösendes taktiles Feedback liefern. Während die Simulationen in anderen Bereichen des Roboterlernens rasch voranschreiten, ist dies bei der taktilen Manipulation nicht der Fall, wo die korrekte und schnelle Modellierung der Kontaktphysik eine Herausforderung bleibt. Wir entwickeln Assembly Gym, ein Framework, das die schnelle Entwicklung von taktilen, kontaktbasierten Montageaufgaben für einen physischen Roboteraufbau, bestehend aus einem Manipulator mit visuellen Berührungssensoren und Objektverfolgung, ermöglicht. Assembly Gym wird verwendet, um eine anspruchsvolle Montageaufgabe zu implementieren und einen großen Datensatz von Montageepisoden zu erfassen. Mit Hilfe von Autoencodern lernen wir unüberwacht eine Repräsentation der taktilen Sensorbilder in unserem Datensatz. Diese Repräsentation wird anschließend für Experimente verwendet, um die Zustandsübergangsfunktion unserer multimodalen Umgebung im latenten Raum zu approximieren. Durch die Durchführung zahlreicher Experimente gewinnen wir Einblicke in die Schlüsselaspekte und Parameter, die eine unüberwachte Repräsentation und das Modelllernen erfolgreich machen können. Schließlich wird das gelernte Modell unserer Umgebung mit einem Trajektorienplaner kombiniert, um die Steuerung durchzuführen.

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

Hiermit versichere ich, Frederik Wegner, 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, 4. September 2021

F. Wegner

Contents

1. Introduction

Humans are remarkable at manipulating objects with their hands and robotics is still far away from catching up. One essential property that enables our precise manipulation skills that are also robust to variation and imperfect information is our human sense of touch [1, 2]. With our skin we are able to sense the surface structure of an object, if something is slipping through our fingers, how much force we are applying with our hands or the precise location of a small object between our fingertips.

Essentially tactile sensing is a requirement for a variety of robotic tasks such as manipulation, exploration, grasping or human-robot interaction [3]. In consequence, tactile sensors were developed that equip a robot with a sense of touch. Such sensor designs vary in resolution and complexity, ranging from a one dimensional force sensors, to 6D force-torque sensors, to high resolution vision based sensors.

Vision-based tactile sensors are not a new concept but recent development has produced sensors with higher resolution, precision and in some cases even lower manufacturing costs when compared to other tactile sensors [4, 5]. These advancements provide additional possibilities of sensing touch that can be used to develop improved control policies for contact-rich manipulation.

Basic tactile sensing has been successfully used to improve control for multiple manipulation tasks [3]. But while control solutions with non-vision-based tactile sensors show a high degree of generalization for complex tasks [6, 7, 8], vision-based tactile sensor applications often still require some task-specific hand-designed features to process the raw sensor images [9].

Although the use of tactile sensor images without hand-crafted features has been demonstrated in principle [10], we argue that evaluation on more complex tasks is needed to further the widespread use of vision-based tactile sensors for manipulation. Another aspect of high resolution vision-based tactile sensors is that they are very hard to model

and to simulate. Thus, experiments on a physical systems are required where the amount of sampled data is usually highly restricted.

Assembly: A Complex Contact-Rich Manipulation Task Assembly is by its very nature a contact-rich task in an unstructured environment, as tracking of objects is impaired by occlusions, calibration errors and manufacturing tolerances.

In these scenarios tactile sensors are most relevant [11], whereas in structured environments contactless sensing may be sufficient. When assembling two components together, we need to stably grasp objects, estimate the precise relative positions between the parts and the manipulator, detect collisions between parts and detect when the part is slipping between the fingers of our gripper. Tactile sensors can provide the capabilities needed to perform such state estimation tasks [3].

Therefore, we consider tactile robotic assembly as one of the major current challenges of robotics and tactile sensors as a catalyst for advancements in this field of research [12, 13].

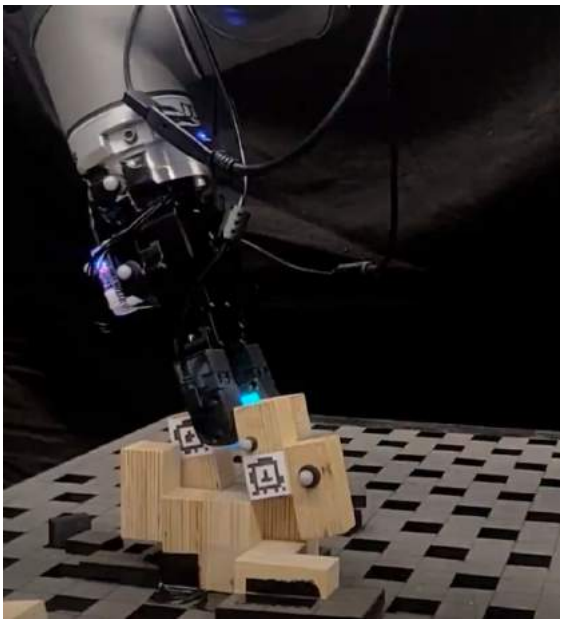


Figure 1.1.: Our contact rich tactile robotic assembly task of joining SL-Blocks. The robot is pictured during data sampling.

1.0.1. Contributions

This thesis investigates unsupervised representation learning from vision-based tactile sensors for control in a complex multi-modal assembly task. We believe that the availability of ready-to-use physical learning environments and data from physical systems is a key driver of research for tactile control. Therefore, we develop a framework that allows quick implementation of gym-style reinforcement learning environments on a UR10 robot with Robotis gripper and DIGIT tactile sensors combined with OptiTrack and/or AprilTag

tracking. The architecture and design principles of this framework are presented and a challenging contact-rich tactile robotic assembly tasks is implemented.

With a large dataset collected from our setup, we learn to extract tactile representations in an unsupervised manner by training a convolutional autoencoder to encode the sensor images. Through numerous learning experiments, we provide insights into the key aspects of successful tactile representation learning such as network architecture and hyperparameters. These representations are then used to learn the transition model of our multi-model environment. We compare different approaches to sequence prediction models for planning regarding single- and multi-step forward prediction. Finally, we combine our learned model with a CEM planner to perform model predictive control.

1.1. Related Work

We structure our review of related work by the type of tactile sensor that was employed: non-vision-based tactile sensors and vision-based tactile sensors.

1.1.1. Non-Vision-Based Tactile Control

As tactile manipulation is a large field of research, we only present research that is relevant for our range of assembly tasks. Reinforcement learning has been used to learn policies for contact-rich manipulation tasks with tactile feedback from force-torque sensors and low-resolution force sensor arrays. [6, 8, 7].

The solved tasks range from opening a door to simple peg-in-hole-assembly to plugging real-world power plugs into power sockets. The tactile sensor used for the door opening task is designed as a grid of simple one-dimensional force sensors on the fingertips of the gripper. Therefore it can be modeled reliably and a policy can be optimized in simulation with deep reinforcement learning [8].

By using self-supervision to learn a latent representation of the observations of the environment, sample efficiency can be improved, thereby enabling reinforcement learning on the real setup [6]. The learned latent representation fuses input of an RGB camera capturing the whole scene and a force-torque sensor mounted in the wrist of the gripper to solve the peg insertion tasks.

Using heavy data augmentation and reformulating the problem as simple regression, instead of using the arguably more complex reinforcement learning framework, more difficult insertion tasks have been solved with a high degree of generalization [7]. Two wrist mounted cameras and a force-torque sensor are used as feedback to the control policy.

1.1.2. Vision-Based Tactile Control

Methods of using high resolution vision-based tactile sensors for control differ in the processing of the sensor images. Markers can be embedded into the gel which can be tracked [14, 15] to estimate the deformation and forces on the surface of the sensor. By extracting contact points and contact forces from the image of the deformation of the tactile sensors gel pad, control laws for manipulation can be designed [16, 14, 9]. Contact points and forces between the robot and the object are computed from colored shading or tracked markers measuring the deformation of the sensor surface. These contact points are then used to obtain the precise location of the object under manipulation. While approaches using handcrafted tactile representations yield working control policies that accomplished their respective task, they do not involve any form of learning and are therefore only applicable to a specific setup.

Improving on this, convolutional neural networks can be used to directly process the raw sensor image without the need of hand-crafted task-specific feature extractors [10, 5]. A ConvLSTM could be trained to predict sequences of tactile sensor images from a ball manipulation task [10]. An end effector with an attached vision-based tactile sensor rolls a ball or dice over a surface towards a target location. The learned predictive model can then be used for deep model predictive control to correctly roll the objects on the table to a desired goal state.

A more complex task of manipulating a ball between two fingers of an Allegro robotic hand equipped with DIGIT tactile sensors was solved with a similar but slightly less general approach [5]. Instead of using a ConvLSTM a variational autoencoder is trained encode the image into multiple key points from which a decoder can reconstruct the image. An RNN is then trained to predict future key points according to an applied sequence of actions such that the task can again be solved with deep model predictive control. We view this approach as less general as key points are local, spacial features and may struggle to properly encode global features. While the position of a ball on the sensor gel can be represented by a set of points, the ultimately much more complex full tactile state of the DIGIT sensor, including forces, torques and texture continuously over a surface, can hardly

be mapped to a set of key points. We based our work on this general setup [5] but instead of extracting key points we use a fully unsupervised autoencoder that compress the tactile sensor images into a global unconstrained representation of the image.

We also choose deep model predictive control because it is more sample efficient while staying competitive in performance compared to model-free reinforcement learning [17]. The choice for a sample efficient method is particularly important as it is not yet possible to fully simulate vision-based tactile sensors. Although recently a new simulator for visual tactile sensors [18] has been developed which simulates the tactile sensor images realistically, it is limited in the mechanical simulation of the deformable gel on the surface on the sensors. Its usage is therefore restricted to low-contact tasks such as grasp stability prediction and can not sufficiently simulate our contact-rich assembly scenarios.

1.2. Outline

Chapter 2 introduces our Assembly Gym task and learning framework together with the physical lab setup. Chapter 3 presents our methods. Firstly, the assembly task that we implemented in our framework is introduced as a reinforcement learning environment, together with our data sampling methods. Secondly, we explain how we learned the transition of that environment, and how it can be used for control. Then chapter 4 presents the important findings of our experiments. The full details of our experiments result in the appendix. After summarizing our findings in chapter 5, we give an outlook on future research opportunities that our work enables in chapter 6.

2. Assembly Gym

Setting up the lab environment for robot learning experiments is a tedious task. A physical environment with a real robot is interfacing with multiple hardware components that each need individual drivers and setup. As a result, researchers tend to stay in simulation or only set up a minimal task that can be hard coded quickly. To overcome this issue and to enable further research in robotic assembly, we set up Assembly Gym. Assembly gym is a learning interface for robotic tactile assembly that provides a simple framework to implement complex tasks involving 6DoF manipulation with object tracking and tactile sensing. The physical setup consists of a robotic manipulator, tactile sensors and a tracking system. Assembly involves the manipulation of multiple objects to assemble them into a structure which usually requires a form of object localization. For this, our setup provides two tracking systems: an RGB camera combined with AprilTag fiducial markers as well as OptiTrack—a precise motion capturing system based on infrared cameras and reflector markers.

The Assembly Gym framework allows us to quickly define new gym-style reinforcement learning environments that we can subsequently start experimenting on for developing control solutions. To make an adoption of this setup possible we will present the architecture and key design concepts.

Interchangeable Simulation And Real Robot Interfaces When evaluating controllers that were developed in simulation on a physical robot setup, reimplementing or adapting the environment to the real robot interfaces is time consuming and tedious. We designed a generic environment interface that is implemented by both the simulation engine and the drivers for the physical setup.

This architecture makes it possible to develop the core functionality of a reinforcement learning environment such as reward function, action and observation spaces and focus solely on the design of the task. When transferring results from simulation to reality and

vice versa, only the back-end of the environment has to be replaced. As simulation of DIGITs is currently not possible we dropped support of the simulation environment for our experiments but this feature remains relevant when used with tactile sensors that can be simulated.

2.1. Hardware Components

Our setup, pictured in Figure 2.1, has three important hardware components. A six degrees of freedom (6DoF) manipulator with a gripper, tactile sensors and a precise tracking system. All of the specific hardware components are actually interchangeable, but during setup of the environment we noticed a few key features that are needed for robotic tactile assembly.

Manipulator Our manipulator is a 6DoF UR10 robotic arm with a Robotis RH-P12-RN-A gripper attach to its end-effector. Being a collaborative robot the UR10 is particularly suited for our contact rich assembly setup as it provides safety features that make safe exploration in the environment easier. We use the Robotis RH-P12-RN-A because it has a low level control API that allows actuating the gripper by current which is needed when mounting tactile sensors to it. The gripping strength needs to be set accordingly so that the silicon gripping surface of the tactile sensor is not damaged.

Tactile Sensor Many different visual tactile sensors have been developed, each of them having their own advantages and disadvantages [11]. We chose the DIGIT design [5] as it is open source and could be produced in house at moderate cost. The fact that it is manufactured by us gave us the possibility to improve and adapt it to our needs. The DIGIT tactile sensors replace the finger tips of our gripper. Figure 2.3 explains how vision-based tactile sensors work and provides example images captured by our DIGIT sensors.

Tracking system As previous work on robotic assembly used AprilTags as an object tracking system [6], we initially employed them too in our setup. However, during development, we faced multiple issues regarding precision, noise and calibration difficulties. Research using AprilTags regards noise as a key challenge that can be tackled with RL policies able to derive precise control from noisy observations. Because this challenge is

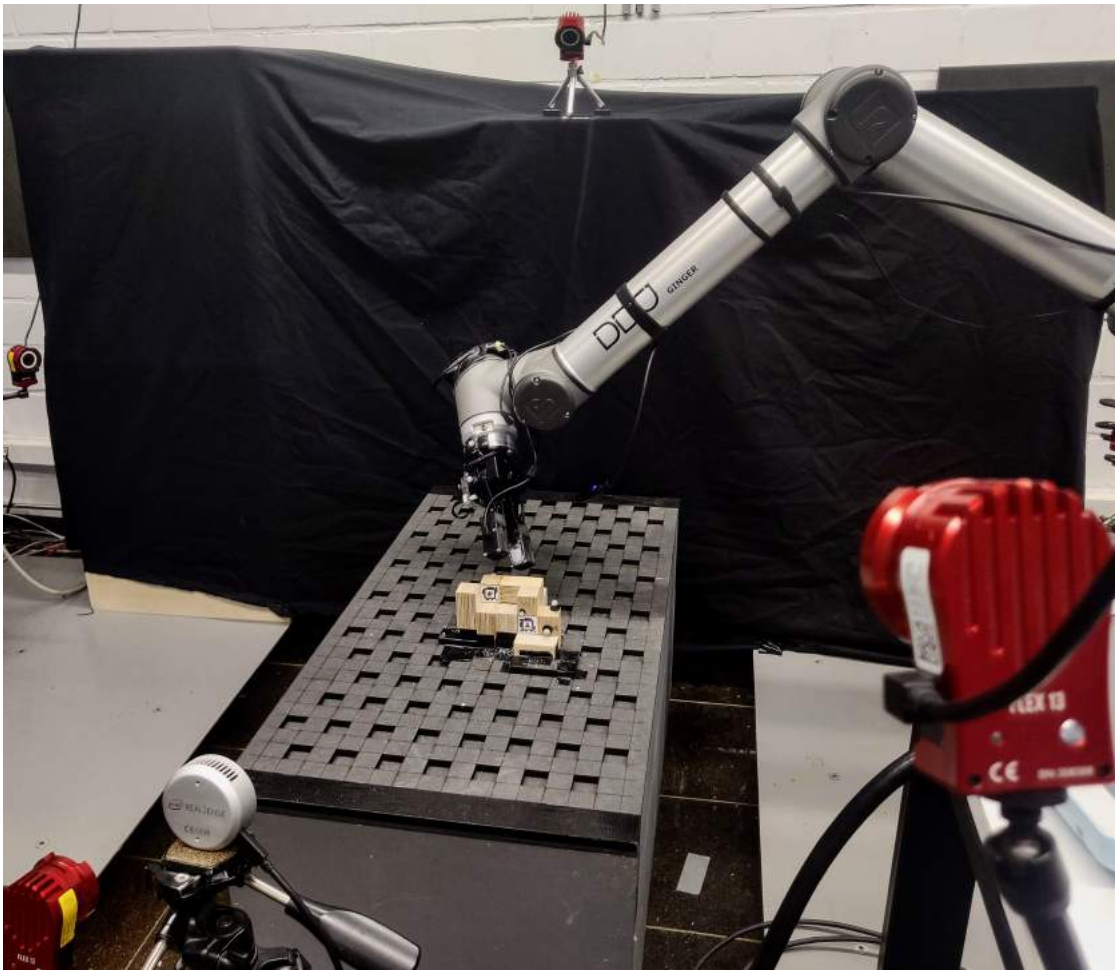


Figure 2.1.: Overview of the lab setup showing all hardware components. Four of the six used OptiTrack cameras surrounding the scene can be seen in the image (red). In the center is the UR10 with Robotis Gripper and DIGIT tactile sensors joining an SL-Block into a structure of SL-Blocks fixed to the table. In front of the table stands an RGB camera that, in combination with AprilTags, can be used alternatively to OptiTrack.

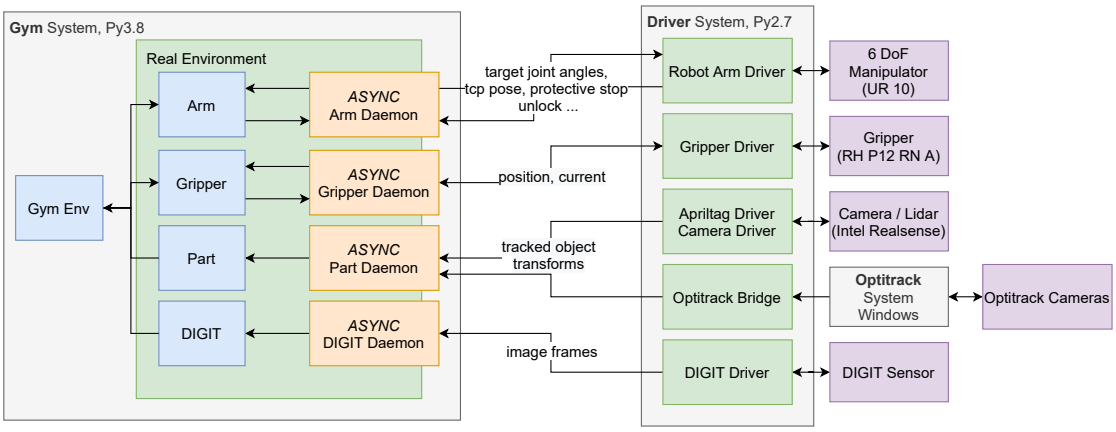


Figure 2.2.: Implementation of the RealEnvironment for our physical setup also seen in Figure 2.4. Green boxes are ROS nodes. Orange boxes are asynchronously running threads. Grey boxes are operating systems. The implementation runs in two different environments on separate machines. On the left is the machine that runs the user code in the assembly-gym framework. Each component of the Environment is implemented paired with an asynchronously running daemon that handles the communication with the drivers over ROS. These daemons subscribe and publish ROS topics while asynchronously reading and writing state from and to the component. This state is then accessed by the user through the generic component API.

(a) (b)

Figure 2.3.: (a) Schematic cross-section of a DIGIT sensor. A ring of differently colored LEDs mounted around the plexiglas base of the gripping surface illuminates the silicone laterally to its surface. This illumination is visualized by the green light ray coming from a green LED located at the bottom of the schematic. Because light coming from different directions is also differently colored, the deformation of the surface can be perceived. A camera in the back of the sensor captures a view of the inside of the illuminated gel. Various gel pads can be used that differ in surface texture or opacity. We primarily use smooth textured gels with a thin coat of white paint. (b) Two example images captured by the cameras of two DIGIT sensors. The sensors are used as the fingertips of a gripper and are seen grasping an SL-Block. Because the coating of the gel is not fully opaque the layers of the MDF wood the block is made of are visible in the image. The orange blob in the bottom is the edge of the block strongly deforming the gel.

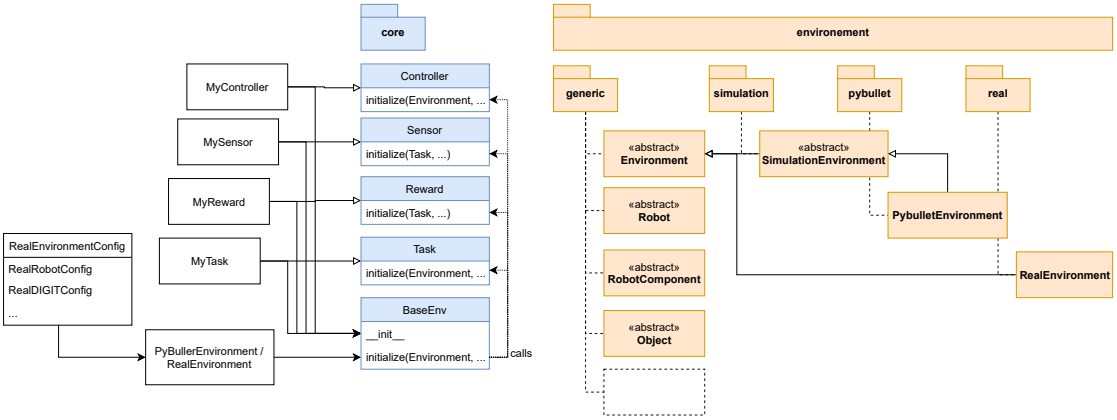


Figure 2.4.: Core components of the Assembly Gym framework. White boxes are objects instantiated in user code. These components are then composed by the BaseEnv which is the actual Gym-style Env. The orange boxes sketch out the structure of the Environment. A generic interfaces is implemented by both simulation and real Environment ensuring Interchangeability. This can be extended arbitrarily for example to additional simulations. Not all generic components are listed, visualized by the white dotted box.

not the scope of our work we prefer a tracking system as precise and reliable as possible and therefore also set up OptiTrack.

2.2. Software Architecture

This section motivates and describes important software design concepts that we considered during development.

For technical reasons, our implementation is split into two code bases that in our case also run on different machines (Figure 2.2). One code base for all Python code implementing our user interfaces and a second for all back-end, driver level code in the ROS framework.

2.2.1. Gym Env Framework

The Gym-Env Framework is the user facing part of Assembly Gym that can be used by researchers to quickly implement a new OpenAI-Gym-style environment. Gym-style means we do not strictly match the OpenAI Gym API but stick with the common *step*: *action* $\rightarrow$ (*obs*, *rew*, *done*, *info*) and *reset*: $\rightarrow$ *obs* interface. Because our framework is based on a physical setup there is no use case as a benchmark environment.

Terminology We use the terms (Gym-)**Env** and **Environment** which name the front-end and back-end of the environment. The Env provides the interfaces and behavior of a typical reinforcement learning environment while the Environment is the back-end this environment is based on. Such a back-end could be a simulator (e.g. PyBullet or Mujoco) or interfaces to an actual physical lab setup.

Composition Our software design is guided by the composition pattern. The environment is instantiated as a hierarchy of components such that the structure of components is visible on the highest level.

Using composition allows researches to define multiple components that can be exchanged with minimal effort enabling quick and agile experimentation with the environment setup. This design decision reflects the uncertainty of research projects. A researcher may be undecided whether to use position or velocity control and therefore implements two interchangeable components.

In this regard, we decompose our BaseEnv into the abstract components Controller, Sensor, Reward , Task and Environment as visualized in Figure 2.4. Controller, Sensor, Reward and Task are called from the BaseEnv and are given the Environment on which they operate. The Controller receives an action and the Environment executes that action by calling the Environment. The Sensors and the Reward receive the Task and extract and return observations and a reward. The Task is the most complex component as it defines the task-specific behavior, such as ending the current episode or keeping a state needed to compute the reward function, by implementing a step and reset function.

The Environment is decomposed into typical components of a robotic setup. In our case of tactile robotic assembly these are a Robot composed of RobotComponents such as an arm and a gripper as well as tracked Parts and DIGIT tactile sensors. These components are designed as generic interfaces that are then implemented by a concrete Environment.

Serializable Global Top-Level Configuration All important configuration parameters are defined on the highest level in the user code and then passed down through the component hierarchy. The concept of global centralized configuration is particular important in the domain of research. Often the code is a part of the experiment and parameters need to be tweaked to achieve results.

We want to ensure that the user can overview all of the parameters that have an effect on the behavior of the environment in one place. By preventing unintended misconfiguration, the parameter tweaking process can be speed up. While experimenting, one wants to log the parameters of the environment together with the experiment results to provide context and reproducibility. Our global top level configuration is therefore serializable.

Real and Simulation Environment Designed as an abstract component the Environment can have multiple implementations that can be used interchangeably. In our case we implemented a RealEnvironment based on to the real physical setup in the lab and a SimulationEnvironment based on PyBullet. Due to the simulation difficulties with the DIGIT sensors (see chapter 1) we dropped support of the SimulationEnvironment but the concept still remains relevant. While the SimulationEnvironment interfaces with the PyBullet physics engine, the RealEnvironment interfaces with the drivers of the different hardware components over ROS.

3. Learning Tactile Representations

This section describes our approach to learning tactile representations. First we give an overview of the pipeline and then proceed by describing each component in detail.

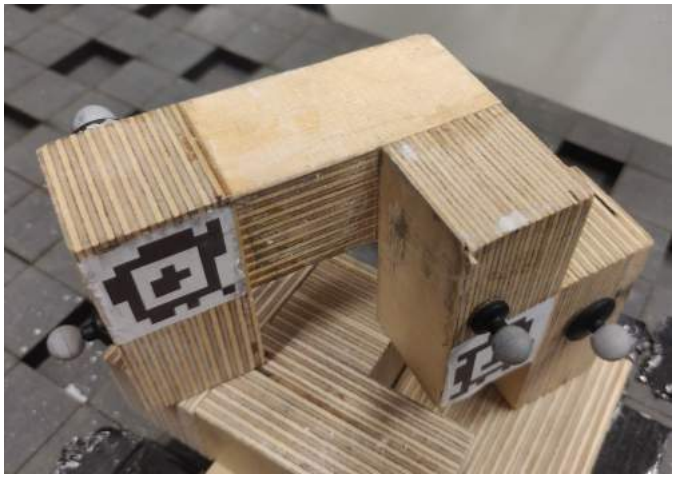
The general idea is to learn a controller that solves a task that can only be completed with tactile feedback. If we can learn such a controller, we imply that it has learned to form a tactile representation from the DIGIT images that it is using internally.

The first component that we need to learn a tactile representation is a challenging tactile task. Secondly we need a method for learning a controller and thirdly some way of validating that the control model actually extracts a representation from the tactile sensor images and does not use some other simplified heuristic.

3.1. Tactile Robotic Assembly Task

Using the Assembly Gym framework, we implement a challenging assembly task with an architectural application domain. SL-Blocks Figure 3.1a were designed by architects as universal building blocks which can be combined into arbitrary shapes. Their highly non-convex shape makes it possible to interlock the blocks such that they cannot be pulled apart easily anymore. While this property gives the possibility to build bridges or overhanging structures, as seen in Figure 3.1b, it also makes assembling them particularly difficult. When putting SL-Blocks together high friction can be created and there are several contact points.

We consider a simplified setup of the SL assembly task depicted in Figure 1.1. An already assembled structure is fixed to the table and a new SL-Block is joined into the structure by the robot. The environment provides the tracked position of the SL-Block. All poses in the environment, the pose of the gripper as well as the pose of the SL-Block, are given relative to the target, assembled pose of the SL-Block such that the task is framed as moving the



(a) Close up view of the SL-Blocks that is joined in our simplified setup. The QR-codes on the block are AprilTag fiducial markers. The grey balls are OptiTrack markers. Because the AprilTag marker tracking is not very precise and robust to changes in lighting and occlusions OptiTrack was used. But as the tracking balls make the assembly of multiple parts impossible AprilTags or even markerless object-based tracking stays relevant.

(b) Large scale structure assembled from SL-Blocks by architects. The structure is around two meters tall. Other elements that fit the joining mechanism of SL-Blocks have also been added.

SL-Block to the origin. We therefore design the reward function naturally as the pose distance of the SL-Block to the origin.

Reward The reward r is the negative distance d between the pose of the moving block p and the target pose $\hat{p}$. Because everything is transformed into the target pose frame the target is the origin $\hat{p} = (t^0, r^0)$. The distance between poses is a weighted sum of translation and rotation distance $d(p, \hat{p}) = \alpha \cdot d(p, \hat{p}) + (1 - \alpha) \cdot d(p, \hat{p})$ combining two values with different units. As translation distances are in the range of centimeters and rotations in radiants, we set $alpha = 0.9$.

Action Space A 6DoF robotic manipulator provides multiple control modes. We choose Cartesian space control over joint space control. Controlling in Cartesian space makes constraining and sampling the action space easier for us as it is easier to visualize and comprehend then joint control. We initially aimed for Cartesian force or velocity control but were unable to achieve real-time control performance with our setup and therefore had to fall back to position control making the task time-discrete. Intuitively, Cartesian force control or wrench control is the desired control mode as it seems most suited for the task. The torques and forces in Cartesian space that are applied by the manipulator directly correlate with the tactile feedback. We leave comparison of control modes to future work.

An action in the environment is therefore the next pose of the gripper in the transform of the target pose of the SL-Block given as a position and quaternion.

Observation Space The observation space is multi-modal as it consists of images of the DIGITs and poses of the gripper and SL-Block. All poses coming from the environment are 7-dimensional vectors $p_x, p_y, p_z, \omega_x, \omega_y, \omega_z, \omega_w$ The images from the DIGIT sensor are three channel RGB with resolution of a resolution of (w: 240,h: 320).

Reset A fully autonomous reset of the environment is desirable when sampling a large dataset. The reset procedure of our task involves picking up the SL-Block, either pulling it out of the structure if the last episode joined successfully or picking it from the table. As we do not have an implementation of regrasping, the human overseeing the sampling process is notified if the SL-Block rolled into an unreachable position such that it can be

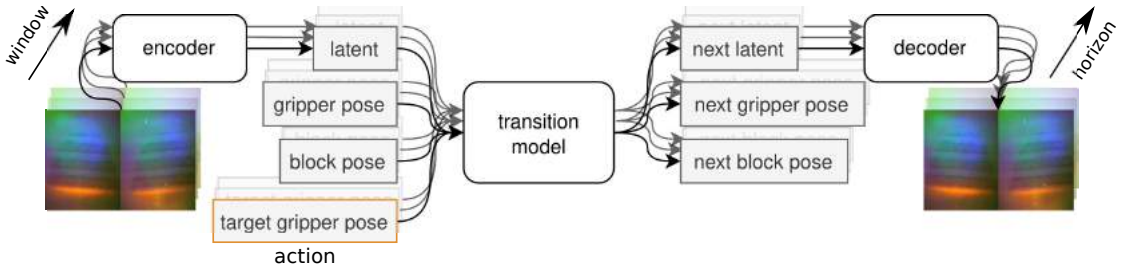


Figure 3.2.: The general architecture of the model learning setup. We have encoder and decoder networks that compress the images of the DIGIT sensors into a latent space. To this we also add the poses of the gripper and SL-Block to obtain one state as seen by the transition model. The gray shades visualize that the model takes a window of multiple past states as input and then predicts multiple steps into the future. The DIGIT state is predicted in latent space and then decoded to an image.

reoriented manually. Most of the replayed trajectories result in a successful joining of the block such that human interaction is minimal.

3.2. Deep Model Predictive Control

While model free reinforcement learning has been successfully applied to solve tactile assembly tasks [7], it has not yet been used with high resolution visual tactile sensors. On the other hand, it has been shown that model based reinforcement learning with deep model predictive control can successfully learn tactile control policies with high resolution visual tactile sensor input [5, 10]. For this reason, we choose to pursue with deep model predictive control as control method.

Model predictive control is a classical control approach. With the state transition or dynamics function of a system a trajectory of future action can be optimized according to a cost function. Deep model predictive control applies this concept to Reinforcement Learning by learning an approximation of the state transition function from data tuples (s_t, a_t, s_{t+1}) sampled from the environment. Then, using this transition model and assuming the reward function is known, the reward or cost of a trajectory of future actions $(a_t, ..., a_{t+h})$ can be estimated and optimized for, resulting in optimal control. In case the reward function is unknown it can also be learned from data from the environment.

As the learned transition model is approximate and highly non-linear, a stochastic optimization technique is used to find optimal actions. We employ the Cross Entropy Method (CEM) for planning as it was used by previous work and is easy to implement.

3.3. Learning the Transition Function

From data tuples (o_n, a_n, o_{n+1}), composed of the n-th observation o_n, action a_n and the observation of the next time step o_{n+1} sampled from the system, the transition function $t : (o_n, a_n) \rightarrow o_{n+1}$ is approximated by a neural network. As feeding a full image into a fully connected network is infeasible, the images have to be compressed to a lower dimensionality by an encoder network for which we use a convolutional neural network. The resulting network architecture and training setup is depicted in Figure 3.2.

To give a good initialization the encoder and decoder networks are pretrained with a simple reconstruction loss. During training of the whole transition function a reduced learning rate is used for the autoencoder parameters and the reconstruction loss is added as a component of the combined loss function.

Window and Horizon The most standard way of learning the transition function would be a simple supervised learning setup where the network gets the last observation and action as input and should predict the next observation. $o_{n+1} = f(o_n, a_n)$ This setup corresponds to a window of formerly seen observations of 1 and a prediction horizon of 1.

As the observation space neither includes forces nor velocities, one single observation does not give sufficient information on the state of the system.

By feeding a sequence of past observations into the system we give it the possibility to differentiate and use this additional information for better predictions. For example, suppose the gripper is holding a block in his hands and moves it towards the floor. At some point the block will collide with the floor and will therefor not move further down but the gripper will keep moving for a few time steps until it collides too. Recognizing this is only possible with knowledge of the velocities of the block and the gripper.

Considering the example above, we hypothesize that a window of multiple past observations as input can make the model more robust to noise and offsets in the observations.

A similar principle also holds for the output of the model. By training to correctly predict multiple steps into the future it has to internally predict a derivative of the future state.

Trajectory prediction instead of state prediction improves accuracy for long term predictions as well as sample efficiency during training [19].

We treat the window w and horizon h as two important hyperparameters of our model.

3.3.1. Parallel and Recursive Multi-Step Prediction

Predicting a trajectory can be achieved by either recursively calling a single step prediction model on its own output:

$$\begin{aligned}(o_{n+1}) &= f(o_{n-w}, a_{n-w}, \ldots, o_n, a_n) \\ &\vdots \\ (o_{n+h}) &= f(o_{n+h-w}, a_{n+h-w}, \ldots, o_{n+h-1}, a_{n+h-1})\end{aligned}$$

Or by simply learning a multi-step model directly as one forward pass:

$$(o_{n+1}, \ldots, o_{n+h}) = f(o_{n-w}, a_{n-w}, \ldots, o_n, a_n, \ldots, a_{n+h-1})$$

During our experiments we consider both training approaches. The transition function f is approximated by a standard multi layer feed forward neural network with ELU activation.

Predicting Deltas The observation space is continuous and future states are therefore close to previous states. Letting the model predict a delta on the previous state instead of the absolute value increases the speed of learning.

$$o_{n+1} = o_n + f(o_n, a_n)$$

As we are using Xavier Initialization,which is a commonly used, the network activations are initially mean zero. Thus, training the model to predict deltas is best suited to this initialization.

3.3.2. Multi-Part Loss Function

A system observation consists of image measurements as well as object poses which themselves are composed of a position and a rotation. Thus the function that defines the loss between a predicted observation $\hat{o}$ and the true observation o, $L(o, \hat{o})$ has multiple parts $L = L^I + L^L + L^P + L^R$ for images, latent images, positions and rotations.

Using a multi-part loss function the question arises how to correctly aggregate the losses. If one component is substantially larger than the others it will be minimized disproportionately. In our case, we are dealing with a loss on positions and quaternions which are in different ranges of values. By simply summing the components of the loss as they are, the model would mostly optimize for predicting correct orientations.

The question of how to optimally weight the components of a multi-part loss is quite difficult to answer. Our solution to the problem comes from a different direction. Inputs, targets and predictions are normalized component wise to a standard normal distribution. This way, the losses are already in a more similar range. Some additional hand-tuned weights are then added but this is rarely necessary.

As image reconstruction loss the Structural Similarity Index Measure (SSIM) is used which outperforms the more widely used l_1 and l_2 loss functions. [20]

$$L^I = 1 - \text{SSIM}(I, \hat{I}) \tag{3.1}$$

The transition model predicts images in latent space. We have to assume that the latent space is well-formed to some extent and therefore also use the Euclidean distance in that space. Since a multi-part loss function is used we need to assure that the individual losses are in a normalized range. Distances increase with the dimensionality of the space such that we would have differently scaled losses for different latent space dimensionality. This increase from dimensionality is counteracted by normalizing the distance with the square root of the dimensionality m of the latent space:

$$L^L = \frac{\sqrt{\sum_{i=1}^{m} (L_i - \hat{L}_i)^2}}{\sqrt{m}}$$

The positions are defined in 3D Euclidean space thus we naturally define L^P as the Euclidean distance:

$$L^P = \sqrt{\sum_{i=1}^{3} (P_i - \hat{P}_i)^2}$$

Because we represent orientations with quaternions it would be natural to use the angle of the difference rotation between prediction and target. However, this led to issues during training, which we attribute to the use of $\arccos$ and the fact that we are predicting deltas and therefore also comparing very small delta rotations. We therefore pivoted to simply using the mean squared error which works because quaternions are a continuous representation of orientation.

Evaluation Metrics To compare models trained under different conditions evaluation metrics are needed. Evaluation metrics are applied to unnormalized values. For positions and latent space images the Euclidean distance is used. The standard unit for positions is meters. Orientations are compared by the angle of the difference rotation in radiants.

$$d_\theta = cos^{-1}(2\langle q_1, q_2 \rangle^2 - 1.0)$$

where $\langle \cdot, \cdot \rangle$ is the scalar product.

3.3.3. Pretraining DIGIT Autoencoder CNN

The left and right DIGIT images are encoded into a latent representation with an autoencoder trained on SSIM-loss Equation 3.1. The encoder is composed of multiple convolution blocks and an MLP that aggregates and compresses the features of the last convolution block into the latent dimension. The decoder is constructed as a mirror to the encoder using transpose convolutions in place of convolutions. A convolution block is a convolution followed by a batch normalization and an activation function.

Since the two DIGIT sensor images actually represent one measurement at one point in time, it intuitively makes sense to encode them jointly. However, due to manufacturing imperfections, the images of the DIGITs look different even when measuring the same state. Therefore, learning specific parameters for left and right DIGITs is also reasonable. We consider multiple modes of separately and jointly autoencoding DIGIT images, visualized in Figure 3.3 that vary in the amount of shared parameters. We hypothesize that encoding

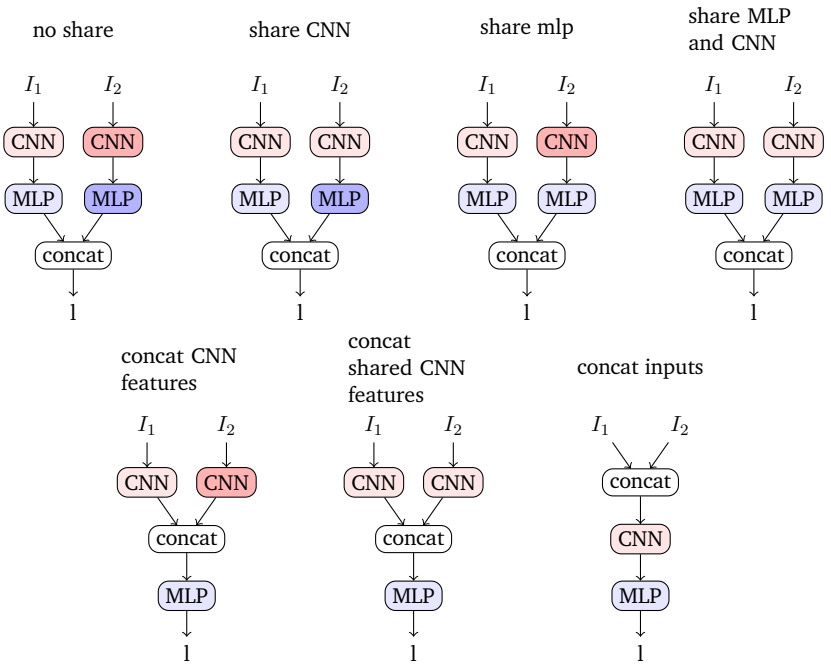


Figure 3.3.: The different architectures for the autoencoder that encodes the left and right DIGIT images. Equal color and shade mean shared parameters. share MLP and CNN is usually referred to as share-all and no share is referred to as default.

the images jointly results in a more cost efficient encoding that is also shaped more suitable to the task.

Variational Autoencoder Variational autoencoders are trained with an additional loss term that regularizes the latent space towards a gaussian normal distribution. [21] As the transition model will predict in the latent space of the autoencoder, a more well formed representation can be advantageous. Variational autoencoders have been effectively used to encode images for transition model learning by previous work [22, 23].

Similar to [24] we introduce a weighting term β balancing the reconstruction and KL-Divergence loss. Instead of finding a suitable fix β, we gradually increase beta during training such that the variational autoencoder learns to encode the variance of the data before normalizing the representation (see subsection A.1.3).

3.4. Data Sampling

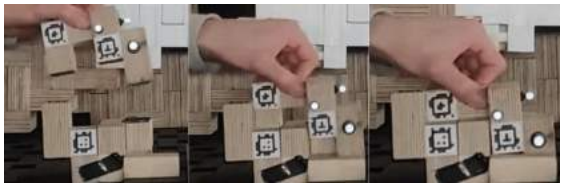


Figure 3.4.: A human performing the task repeatedly while the trajectory of the block is recorded by the tracking system. The state space is sampled close to the goal state.

In order to learn the transition function of the system we need to sample a large dataset (>10k samples) for offline learning. This data should cover as much of the relevant state space as possible. In our case such a dataset cannot be sampled from random actuations to the system because we would not get any data from the states during joining, when the block is beginning to slide into its slot. We sample the data from a set of recorded hand demonstrations (see Figure 3.4 that are then replayed by the robot. To increase the variance of the data without the need of recording additional demonstrations the recordings are noised and also interpolated to obtain new trajectories.

A hand-recorded trajectory is a sequence of SL-Block pose measurements.

3.4.1. Trajectory replay

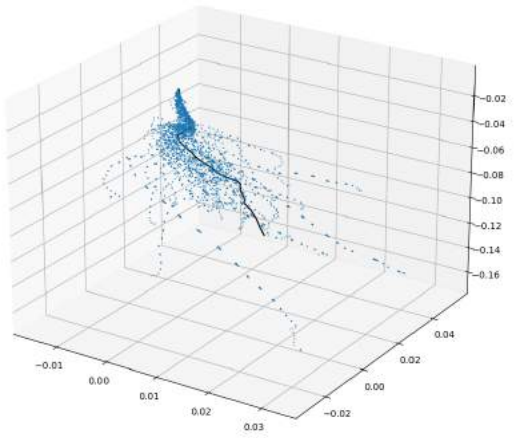


Figure 3.5.: Hand recorded trajectories of the block position, rotations are not visualized. As they are in the coordinate frame of the target position for the block they converge to zero. The blue arrows point to the next position in the trajectory. The black line is an interpolated trajectory.

Based on Cartesian pose control and simple linear transformations, we implement a controller that moves the SL-Block along one of the recorded trajectories. The output of the controller is the next pose of the gripper. The controller traces the points of the trajectory one by one. If the next SL-Block pose in the trajectory is not reached by the last executed action the controller tries to move the gripper again such that the SL-Block reaches its next pose. This means that all poses in the trajectory have to be reached at least once. If the SL-Block is not reaching its desired next pose after executing an action then the grasp pose has changed due to some collision. If the grasp pose changes too much from its initial value the replay is aborted to prevent damage to the gels pads of the DIGITs.

Noise is applied to the next action issued by the trajetory trace controller, before it is executed by the robot, but not to the trajectory itself. That means that even with larger noise the controller will reach all points in the trajectory, if it does not lose grip of the part along the way. This sampling technique allows us to sample new states close to the the already recorded states while it does not enable us to sample entirely new trajectories.

To also sample entirely new trajectories the hand demonstrations are interpolated in

trajectory space with a k-means algorithm Figure 3.5. Starting from pose $s_0 = (p, \omega)$, k nearest neighbouring states according to l^2 distance $n_1, ..., n_k$ are taken. Then for each of these known state we obtain the vector to the next state in the same trajectory. These vectors $v_1, ..., v_k$ are then weighted by the distance of the state it belongs to the current state. $w_i = \frac{\|s_0 - n_i\|^2}{\sum_{j=1}^{k} \|s_0 - n_i\|^2}$ This weighted average is then added to the current state to obtain the next state. $s_1 = s_0 + \frac{1}{k} \sum_{i=1}^{k} v_i \cdot w_i$

Intuitively this can be described as creating a vectorfield from the recordings and subsequently integrating it.

3.5. Cross Entropy Method

Finally, we want to use the learned transition model to perform assembly by combining it with a trajectory planner. As the learned transition model is only approximate a probabilistic planning algorithm is needed. For simplicity, we use the Cross Entropy Method (CEM) to optimize for an optimal trajectory by iteratively sampling trajectories of actions and estimating their reward.

For a planning horizon h at each iteration n trajectories $\mathbf{a}_1, \ldots, \mathbf{a}_n$ each $\mathbf{a} \in \mathbb{R}^{d \times h}$, where d is the dimensionality of the action space, are sampled from a normal distribution parameterized by $\mu, \sigma \in \mathbb{R}^{d \times h}$. Propagating these with the transition model f results in observation trajectories $\mathbf{o}_i = f(\mathbf{a}_i)$ which we can score according to the observation based reward function $s_i = \sum_{j=1}^{h} \text{reward}(\mathbf{o}_{i_j})$.

Then μ, σ are updated with the mean and standard deviation of the best e trajectories according to the scores. These are called the elite. Iterating until an iteration limit or convergence is reached results in a final $\hat{\mu}, \hat{\sigma}$ as output of the optimization. The first action of $\hat{\mu}$ is then executed on in the environment.

Sampling Action Deltas The performance of CEM is very sensitive to its initialization μ and σ. A bad initialization will result in trajectories far away from the data the model was trained on resulting in larger errors in the prediction which propagate into the trajectory scores and therefore into the selection of the elites. Such a bad selection of elites can lead to an iteratively exploding variance.

As the actions in our environment are absolute positions it is really hard to find a good initialization for the variance to sample from as later actions near the horizon would need a larger variance that leads to the aforementioned instability. For this reason, we optimize a trajectory of deltas on actions instead of absolute actions. We initialize CEM with the mean and variance from the training data.

4. Experiments

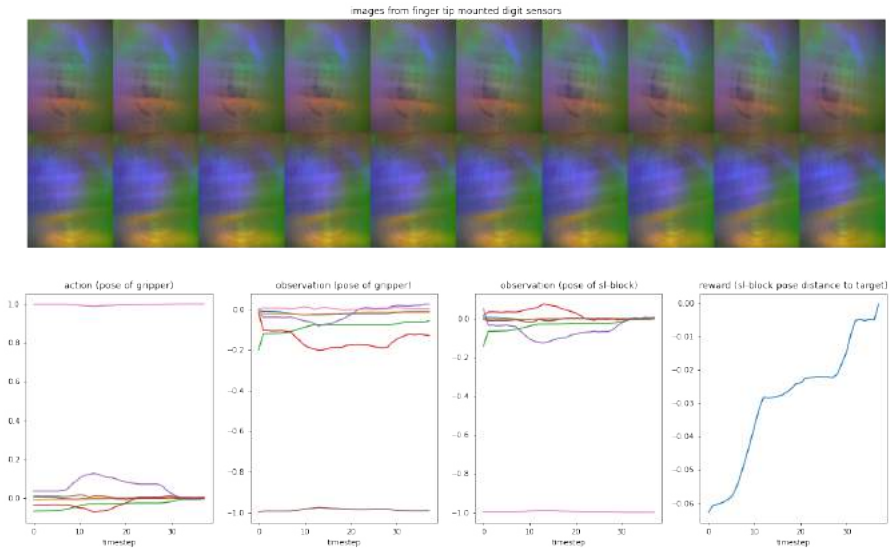
This chapter presents the performed experiments and their results. We start off by describing the data that our experiments are based on, continue with stating the questions that we want to answer with our experiments and finally presenting each experiment with its result. Our experiments are grouped into autoencoder learning, transition model learning and control experiments.

4.1. Dataset

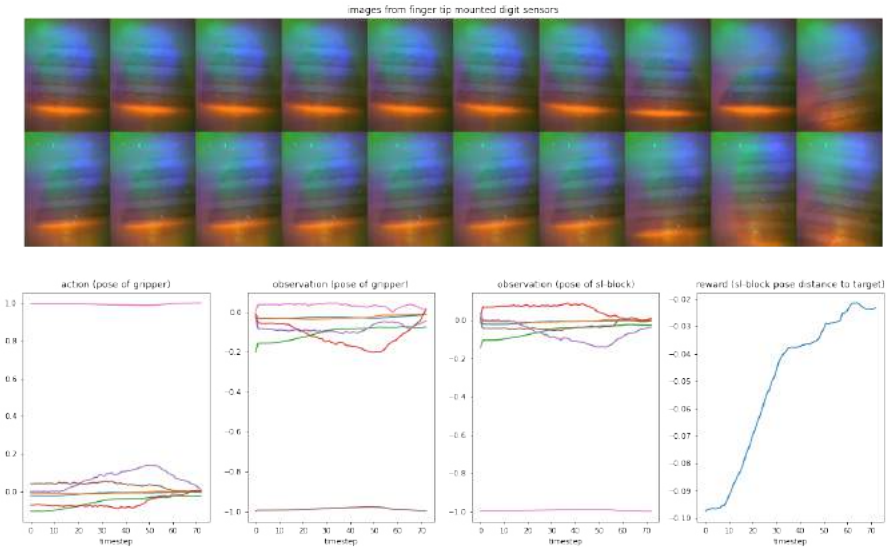
Using the methods presented in section 3.4, two datasets were created by sampling multiple episodes autonomously. One time step of an episode is regarded as a sample. One is sampled with smooth gels and the other with fingerprinted gels as seen in Figure 4.1. The smooth DIGIT dataset consists of 25,609 samples while the fingerprinted DIGIT dataset contains only 9,979, as the fingerprinted gels broke and we did not have any more replacements.

As seen in Figure 4.1, most of the contact happens at the end of the episode. Therefore, we filter our dataset and remove all data points for which the position of the SL-Block is more than 4cm away from its target, such that contact is unlikely to happen. After filtering, the datasets are reduced to 20,216 and 8,217 samples.

Breaking Gels As our data sampling procedure is explorative (see section 3.4), it happens that the robot drives the block into the structure. Even though a collaborative robot with built-in safety features that can sense collisions and trigger a safety stop is used, it happens that the gel tears. We could sample at most 10,000 time steps before a gel broke and had to be replaced. Even though the gels were produced in one batch and appear similar to the eye, there is still a significant amount of gel-specific noise, visualized in Figure 4.2.



(a) Episode with fingerprint gels. Only every fourth image is shown.



(b) Episode with smooth gels. Only every tenth image is shown.

Figure 4.1.: Two episodes sampled from our environment. During insertion the forces acting on the gripper distort the gel and consequently the fingerprint on the surface of it. The same distortion is visible on the smooth gel as the border between the sharp and the blurred. The action and observation of the gripper pose are very similar (note that the orientation part of the pose, the quaternion, is negated, but due to quaternion duality they represent a similar orientation) but not the same measurement. As jamming can occur during joining of blocks, the target pose that the robot receives as action might not always be reached.

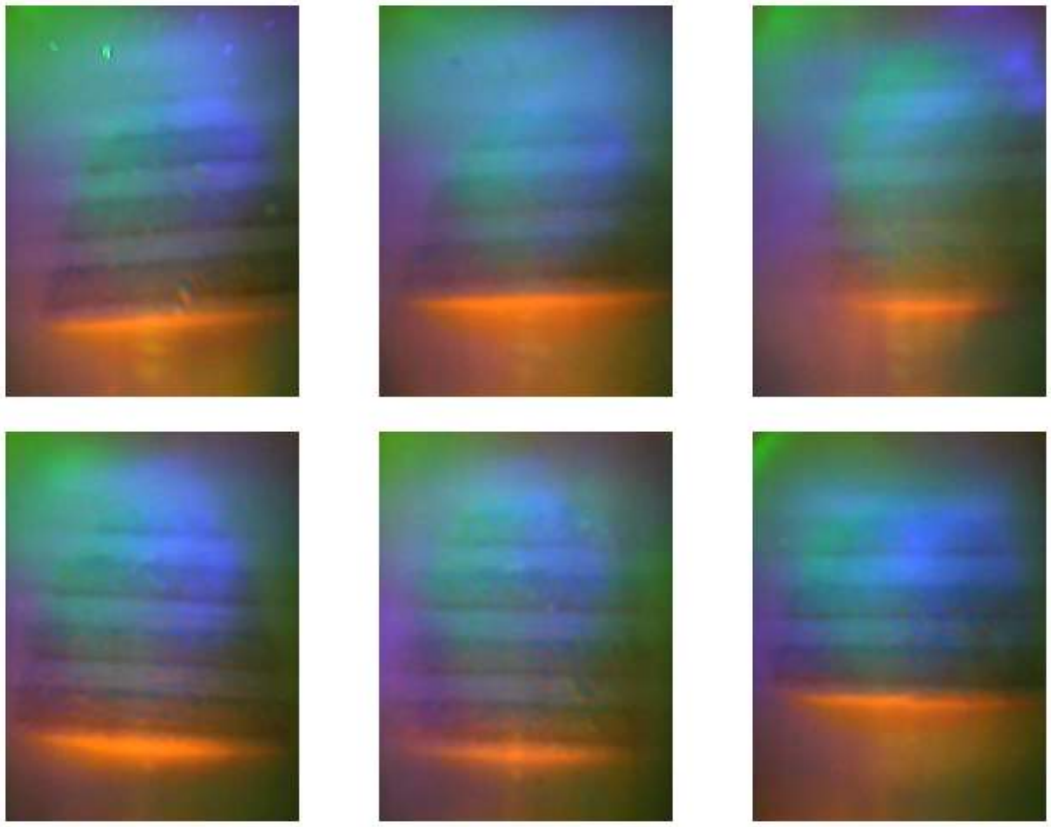


Figure 4.2.: Sensor images taken during sampling with different gel pads from the same manufacturing batch. The top row shows the right, the bottom row the left sensor images. Each column contains a different pair of images taken at different times during data sampling. On close inspection, there is quite a lot of fine-grained structure in the image that is not specific to the captured underlying tactile state, but instead to the gel pad used. This noise comes from manufacturing tolerances such as bubbles in the gel or bumps in the surface, and from abrasion and dirt on the surface of the sensor.

The models that we train on the two datasets therefore need to be able to generalize over different types of gels.

Recalibration The dataset is recorded in multiple sessions over multiple days. In between sessions it happens that external disturbances cause the OptiTrack to decalibrate. After recalibrating OptiTrack, the hand-eye calibration has to be renewed aswell. A calibration always has a calibration error that is a small unknown random deviation in rotation and translation, measured by the tracking system. The dataset is divided into chunks that each have a different calibration error on the measured poses.

4.2. Autoencoder

The questions that our experiments should answer go into two directions. Firstly, how do the parameters of the training process take effect: How large does the dimensionality of the latent space have to be? Does encoding jointly utilize the latent space more effectively? How do we scale β for the variational autoencoder? Is it even necessary to regularize the latent space with the variational autoencoder framework?

Secondly, we are interested in the meaning the learned latent representation has regarding our task. A "good" latent space is continuous and places images that represent similar measurements close to each other in latent space while spreading apart images that represent very different measurements. Ideally, identical measurements that were recorded multiple times, even with different gels, are encoded into the same representation.

To answer the questions regarding the training, we perform learning experiments with varying parameters.

We then select a subset of the models that are of interest and analyze their latent space. This is done by latent space interpolation and plotting the latent space over time.

4.2.1. Jointly Encoding

To evaluate the effectiveness of the architectures presented in Figure 3.2, we perform a grid search across the parameters latent space dimensionality (2, 4, 8, 16) and model architecture modes (Figure 4.3). When talking about the latent space dimensionality we

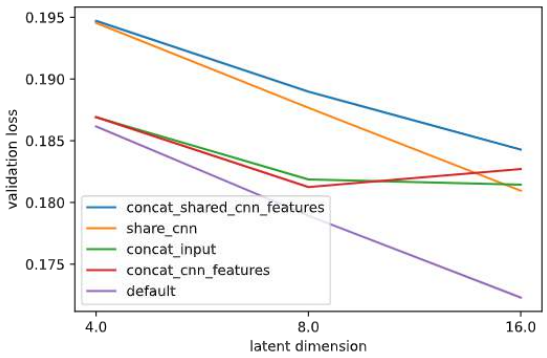


Figure 4.3.: Performance measured by validation loss of different architecture modes when compressing into differently sized latent spaces. Architectures that jointly encode (prefixed concat_) take less advantage of a larger latent space. With a smaller latent dimension the default architecture with vastly more parameters has only a diminishing performance advantage.

mean the dimensions per image in the input, thus the actual total latent space dimensionality is the double of what we usually write. It appears that having separate parameters in the MLP that can learn individual features for the left and right images is important, since share-mlp and share-all perform the worst. The default architecture learns two fully separate models for the left and right image and therefore has the largest number of parameters. Thus it is no surprise that it performs the best regarding our reconstruction metric. However, architectures that produce a joint latent representation (concat-input, concat-cnn-features, concat-shared-cnn-features) use their parameters much more efficiently. The concat-input architecture has almost half as many parameters as the default architecture, but nevertheless performs comparably. This joint encoding advantage becomes more visible when evaluating at different latent dimensions. As we reduce the size of the latent space the performance advantage of default over concat- shrinks (Figure 4.4).

4.2.2. Latent Space Analysis

Simply evaluating our autoencoders by their performance on image reconstruction is not sufficient for our case, as this gives no insights about continuity and shape. In order to be able to use such a representation for control, it should be well formed to some extent and encode features that are actually relevant for our task.

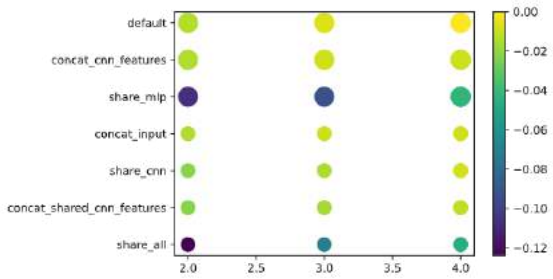


Figure 4.4.: Performance relative to the best model (default, latent 16, yellow) and worst model (share-all, latent 4, dark purple). The horizontal axis shows the $\log_2$ of the latent dimension. The size of the points is proportional to the number of parameters. The vertical axis is sorted by the model size. Two outliers, share-mlp and share-all, can be identified. It can also be seen that concat-input is the most efficient regarding performance-size trade off.

Figure A.4 shows the features of our latent representation over the progress of an episode for all trained models. Figure 4.7 gives two examples.

Latent Space Interpolation A common evaluation method is latent space interpolation as it reveals the continuity of our representation. Four images, displayed in Figure 4.5, are encoded, and then values in between them are interpolated on a square grid by taking a weighted average of the latent representation of the corners. The weights for each gridpoint are calculated from the ℓ^{max} distance to the corners of the grid. Results of these interpolations, for all models that we trained, are seen in Figure A.3. Figure 4.6 highlights two of these.

To visually evaluate the quality of the latent space interpolations, one has to define some quality measures. The ideal interpolation that we regard as good is one that interpolates the images in a smooth linear motion, so that each image looks like a sensor state which could have come from an actual measurement. Here "a smooth motion" means that when looking at the images in one row or one column, there is an incremental change from image to image. These changes should ideally be perceived as equidistant. Some examples of bad interpolations or artifacts are given in Figure 4.8.

Looking through the interpolations in Figure A.3, it can be observed that the best performing model, according to our training loss, is not the best performing model regarding the interpolations. The share-all architecture interpolates far better than its validation

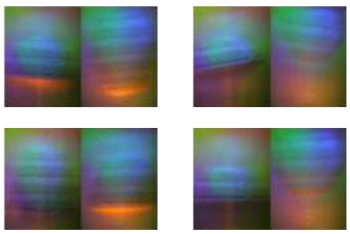
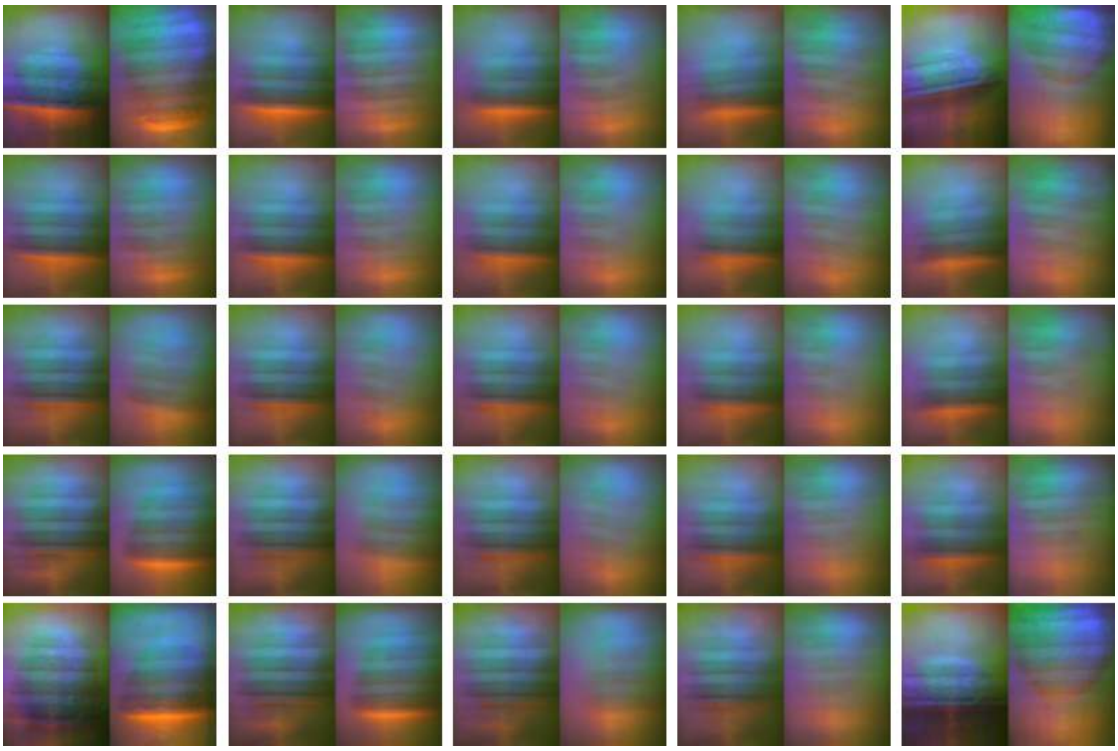


Figure 4.5.: The four sensor images of the test data between which we interpolate for latent space analysis. Images in a row come from the same episode. Images in a column come from the same time step, relative to the end of the episodes. The images in the right column are from the third last time step of their episode.

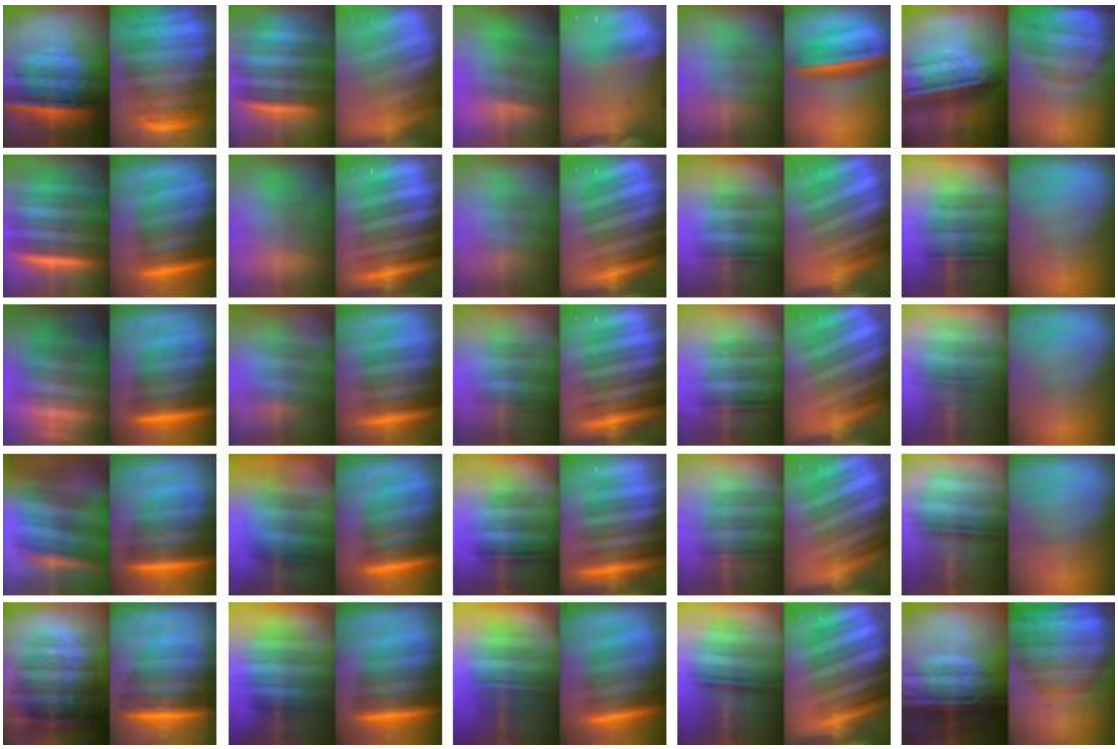
loss suggests, as it has to generalize across images from the left and the right sensor. We attribute this effect to our dataset being unbalanced, containing more images where the edge of the block (edge appears as orange blob in the images) is tilted to one side rather than the other. In fact, share-all is the only architecture that is able to generate images where edges are tilted in both directions. Data augmentation can compensate these issues, but we did not make use of this method.

Jointly encoding the images (architectures named concat-...) indeed produces interpolations with valid, matching left and right images, as explained by Figure 4.8b. This issue is less present for models with more parameters such as default. It seems that a very good interpolation between individual images is likely to also produce valid combinations of images. When interpolating from an image A to another image B linearly, the abstract image features of A should change in linear increments towards B. Such an interpolation result indicates that our encoders latent space is large enough to encode individual features on individual dimensions. However, if the latent space is too small, multiple features in our image are compressed onto a line in space, resulting in disproportionally large changes in the interpolation. From observations of all latent sizes that were tried out, 8 seems sufficiently large while 4 already seem like being too small.

Latent Space Across Time Simply visualizing the latent space over the course of an episode, see Figure A.4, can reveal whether the learned encoding is related to what happens physically in our environment. For this analysis we define good by looking at the

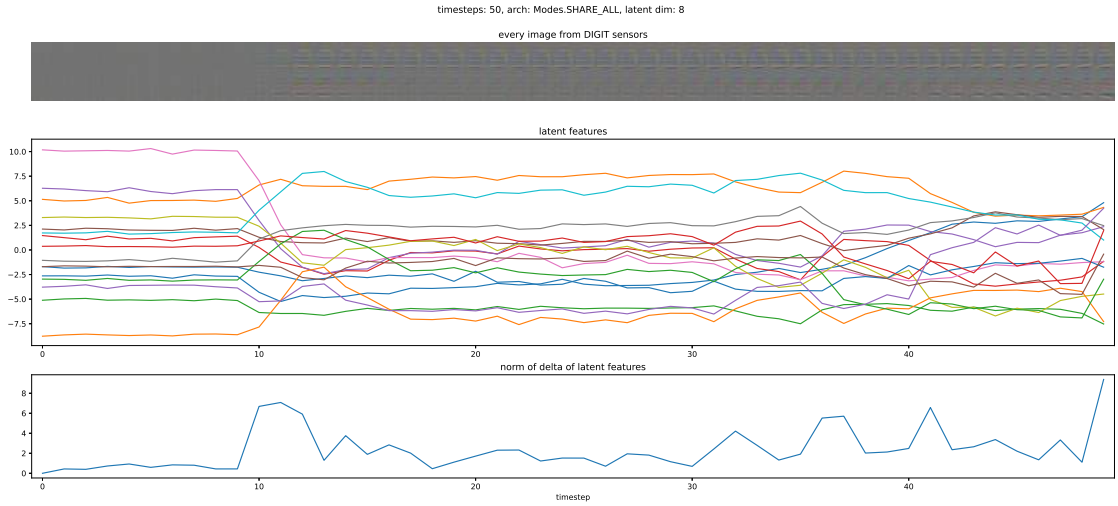


(a)

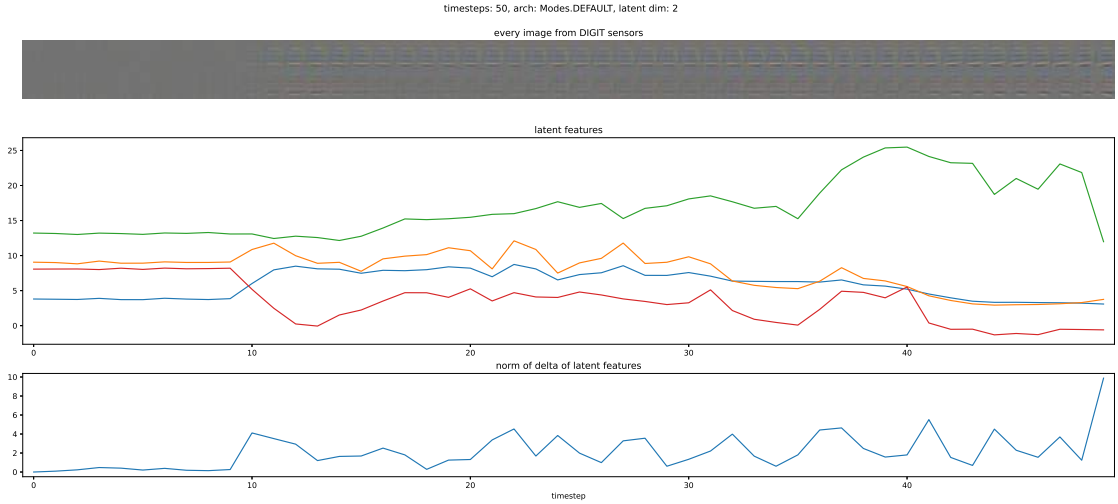


(b)

Figure 4.6.: A good (a) and a bad (b) interpolation result from two models with different parameters.



(a) Example for a good looking latent space.



(b) Example for a not so good looking latent space.

Figure 4.7.: Latent space of two different models visualized over the course of an episode. Only the last 50 time steps of the episode are visualized. The tactile sensor images are given as difference to the first image in the episode, to emphasize small changes. The bottom most plot in each subfigure shows the magnitude of change in the latent representation according to Euclidean distance. The models are the same that were also used in Figure 4.6. We call the latent space of (b) "bad" because there change in the latent representation does not correlate with what is going on with the sensor images. The large change in tactile state at time step ten is not clearly recognizable in the change in the representation.

magnitude of change in the latent space. If there is little or a lot of visual change in the sensor image we expect a small or large change in the latent representation. When there is large movement in the latent representation while there is little change in the image the latent representation must have encoded noise or irrelevant properties of the image which is not desirable. At this point, we want to highlight that such a simple encoder when combined with a threshold value can already be an effective collision detection mechanism and thus be regarded as task-relevant.

A common pattern that we observe is that some features make up most of the magnitude of change in the representation. From these visualizations one might conclude that all of the learned latent spaces are rather unsmooth as there is a large variance in the magnitude of change. But considering the underlying contact-rich environment with discrete time steps the data is coming from, it might be natural that the tactile representation does not change smoothly over time.

4.2.3. Variational Autoencoder

The motivation to learn a latent representation in the variational autoencoder framework is that it can produce a smoother more continuous latent space. This increase in smoothness and continuity should be observable in the produced latent space interpolations. However, the more well formed latent space comes at a cost of less good looking reconstructions by our decoder as it has to deal with the noise added to out latent space. subsection A.1.3 provides the full details of our training setup. Figure 4.9 compares latent space interpolations of a variational and a standard autoencoder. All parameters except an added regulization term in the loss function are equal. Comparing the interpolations side-by-side, we cannot conclude that the latent space of the variational autoencoder is more continuous. Though, looking at Figure 4.9d one could asses a smoother latent space, but compared to the standard interpolation it may actually be too smooth. Overall there seems to be no advantage of using a variational autoencoder regard the latent space interpolation.

Figure 4.10 compares the latent spaces of both standard and variational directly over the course of an episode. Except that the latent features of the variational autoencoder are in a standard normal range there are no significant advantages or differences to observe.

Competing Reconstruction and KL-Divergence Loss During training of the variational autoencoder we noticed that reconstruction loss and KL-divergence loss are directly

competing each other as seen in Figure 4.11. While β is still small the model predicts variances close to zero and there is no noise in the latent representation. As β increases the output distribution of the encoder normalizes and therefore more noise will be added to the latent representation making it more difficult to correctly decode the features to the target image. The maximum value 0.05 and scaling speed for β were selected by hand as the values from [24] cannot be used here because our reconstruction loss is at a different scale. We acknowledge that some early stopping based on the KL-loss can result in a better trade-off between reconstruction loss and KL-divergence loss.

4.2.4. Generalization Across Sensor Gel Pads

As the gel pads of our tactile sensor wear out frequently, it is important that our representation is robust to images coming from sensor with different pads. To check whether this is the case, we repeated the same motion with different gels so that we obtained multiple images from different gel pads that all measured the same underlying tactile state. Figure Figure A.6 visualizes this experiment. We chose a model with architecture share-all and latent dimensionality 8, as it gave good latent interpolations (Figure 4.6), and its latent representation looked task relevant (Figure 4.7). None of the gels that we use during comparison were used during training, but one uses the same type of gel pad, smooth with white coating on the surface, while the other two are entirely different, one translucent, the other one with a fingerprint.

4.3. Transition Model

As our final goal is to solve the control task with model predictive control, the transition model of the environment has to be learned from the data. subsection 3.3.1 introduced multiple parameters that can potentially affect the performance of our approximation. The hyperparameters of the autoencoder that is used add up to the hyperparameters of the transition learning To find out how to configure these and what their effect on performance is, we explore them combinatorially.

Our experiments compare the variants of learning the transition function that were presented in subsection 3.3.1. We hypothesize that there is a significant difference in performance between the various architecture of autoencoders that we experimented with. Therefore, different autoencoder architectures are used. The standard formulation of the

autoencoder is used, as we could not find disadvantages over the variational autoencoders (see subsection 4.2.3). Additionally, we have more pretrained standard autoencoders available because they are much easier to train.

This section highlights different findings from experimenting with learning a transition model, showing only partial results, while a fully detailed presentation of all relevant experiments resides in section A.2.

Predicting Deltas Improves Learning Predicting deltas instead of absolute values increases the learning performance significantly, as seen in Figure A.10. As a consequence of this finding we use delta prediction for all other presented experiments.

The deltas are normalized to the distribution of deltas in our data. This means that our pipeline has two normalization steps. Firstly, the absolute values are normalized, and secondly, the deltas in the normalized space of absolutes.

4.3.1. Effect Of Tactile Feedback

Comparing the mean forward prediction error on poses of transition models trained with (Figure 4.12) and without (Figure 4.13) tactile feedback, it can be seen that the model uses the additional information from the tactile representation to make better predictions for the poses of the SL-Block and TCP. However, this is only observable in the positions and orientation of the SL-Block, not in the prediction of the TCP orientation. Section 4.3.8 addresses this issue in detail.

For the predicted positions, the mean predictions without tactile state are about as bad as the worst predictions with tactile state (dotted lines in Figure 4.12). Regarding the orientation predictions, adding the tactile representation to the state can actually worsen the prediction performance. An explanation could be, that translational movements of the block between the fingers of the gripper are much more visible in the tactile image than rotations.

4.3.2. Effect Of Increased Prediction Horizon

Predicting multiple steps into the future improves the performance, as already found by previous work [19]. Looking at the full forward predictions, of all the differet models that were trained, in Figure 4.12, the advantage of a larger prediction horizon becomes

apparent. The runs underlying this plot vary in several other hyperparameters, such as learning rate and network size, and are grouped by prediction horizon. From this is can be concluded that an increased prediction horizon directly improves the learning performance.

4.3.3. Effect Of Input Window

Analogous to the previous experiment on horizon, we group the test errors of all runs that were performed by the window size used (see Figure 4.12). Only a small increase of performance with larger window size can be observed. However, this is not what we intuitively expected as described in section 3.3, paragraph Window and Horizon. Either the task could be too easy, such that the effect of a larger window cannot be measured, in the sense that a single observation already contains all necessary information, or we just need much large window sizes. In case of the task being to easy one can increase the difficulty by artificially noising and offsetting the inputs such that looking at a window of past states is required to make a good prediction. In case of large window sizes, flattening and feeding through a MLP model becomes infeasible. Instead the window can be processed with temporal convolutions or recurrent models with hidden state.

4.3.4. Parallel Vs Sequential Prediction

Parallel forward prediction of the horizon performs better (see Figure 4.12). This can be attributed to the fact that even though the sequential model is trained on a forward propagation of its own predictions, it has no hidden state that is kept across time. On the other hand, the parallel prediction network can form some hidden state, from which it predicts the horizon, similar to a recurrent model.

4.3.5. Effect Of Autoencoder Architecture

Of the seven different architectures introduced in chapter 3, sare-cnn, default, concat-input and concat-cnn-features were selected for transition model experiments, so that we have two architectures that encode a joint representation and two that encode per sensor representations. It would have been reasonable to also use share-all, due to its remarkable performance on latent space interpolation, but at that time we relied solely on the reconstruction loss as a performance metric.

The top row of Figure 4.12 compares the different architectures. Surprisingly, the default architecture remains the best performer in opposition to our intuition and analysis of the latent space.

It seems that jointly encoding the sensor images produces a latent space that makes it more difficult for the transition model to predict ahead. But as the differences between the groups are not very large we do not draw a final conclusion.

A possible explanation for this result is that when encoding jointly, data from two distributions, the left and the right sensor, is mixed. This mixed dataset contains many samples from each distribution, but as they might be arbitrarily far away from each other, and there are no samples in between them in the dataset, it could negatively affect the latent representation when the left and right images are treated as coming from a joint distribution.

4.3.6. Dimensionality Of The Representation

The second row of Figure 4.12 compares an 8 and a 16 dimensional latent space. As already suggested in subsection 4.2.2, a certain minimum dimensionality is required to separate the different abstract features of our data well. This assumption is reinforced by the observed decrease of predictive performance with a larger latent dimensionality of 16 compared to 8.

What may not be expected at first is that a larger latent space also affects the performance of the other predictions. A higher dimensional latent space is therefore not only harder to predict, but also harder to read, for the model. We have a theory on why this is the case. The larger the latent dimensionality, the more components of it are actually not necessary or meaningful - to some extent they are just noise. But during training the model computes a loss over all outputs, even these meaningless ones. It could be the case, that this loss on noisy outputs effectively adds noise to the gradients, and in this way also reduces the learning performance. This particular issue, that arises with multi-modal state spaces, could be thoroughly investigated by future work.

4.3.7. Fingerprint Gel

The latent representations that are produced by our fingerprint model (see Figure A.8b) look different to what we obtained from the smooth gels (see Figure A.9c). The represen-

tation from fingerprints is smoother and has less large spikes in features that are frequent in the representation from smooth gels.

When closely looking at the images produced by the different gels we can see that the forces that are acting on the gel can be clearly perceived by the deformation of the fingerprint structure. On the other hand, these forces are not directly visible on the smooth images.

The visual cues in the smooth images are the edge of the block pressing into the gel, the stripes of the block as they are shining through the coating of the gel and the boundary between the blurry and sharp parts of the image. The shape of this boundary relates to the forces acting on the gel, similar to the fingerprint deformation on fingerprint gels. However, this boundary is much harder to perceive and contains less information than a deformed fingerprint structure. In this sense we regard the fingerprint measurements as more continuous, as very small changes in the forces acting on the surface are clearly received in the deformation of the fingerprint pattern. We actually intended to use more fingerprint gels but were limited in supply.

4.3.8. Bad Prediction Of Robot End Effector Pose (TCP)

Looking at the forward prediction errors in Figure 4.12 one observes that the error in the TCP pose prediction is much larger than the error in the predicted pose of the SL-Block.

This observation is the opposite of what one would expect, as the future TCP poses are simply given by the trajectory of actions and are therefore the simplest thing to predict. On the other hand the pose of the SL-Block is under the influence of contact and therefore very hard to predict.

One explanation for this could be the hand-eye calibration error of the system. As the environment uses the target SL-Block pose, which is measured by the tracking system, as reference frame, the calibration error exists in the TCP poses and not in the SL1-Block poses. As a consequence the target actions do not perfectly match what is actually executed on the system and observed as TCP pose.

It suggests that learning to infer the calibration transform from the window of past observations is difficult. The hand-eye calibration of our setup is usually computed from 50 sampled poses. In relation to this a window size of 4 is indeed very small. Experiments with very large window sizes were not carried out as part of this work.

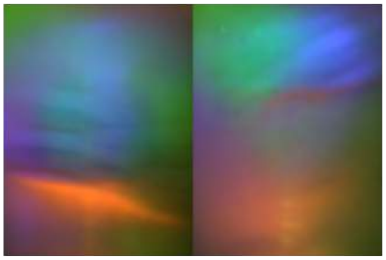
4.4. Control on Real System

Finding suitable parameters for CEM was difficult. One particular issue was the balance of the variance between position and orientation actions. We used the variances from the dataset, but still needed to reduce orientations. The problem is that the CEM optimizer outputted trajectories that would drive the state far away from the dataset. For example in our data, at the beginning of joining motions, the actions usually have small rotations and large translations. Contrastingly, at the end of the episode the changes in rotations are usually larger. Therefore using the data variance is not a good initialization because we end up with rotations that are too large in the beginning.

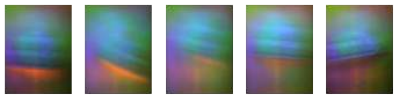
When designing the cost function on trajectories, a trajectory that ends the episode without reaching the goal should be penalized. However, our implementation simply used the summed reward of the whole trajectory, as defined by our environment, even if the trajectory could only be executed half way in reality. This way trajectories that go out of that state space that can be sampled can make it into the elite such that they are kept. The controller did not seem sensitive to the number of samples used, but the ratio to the number of elites is important.

Because we did not achieve reasonable control results with the model without feedback, no additional experiment with DIGIT feedback is presented.

(a) *fading*, instead of gradually morphing the abstract features of the image the model overlays two images from the dataset.



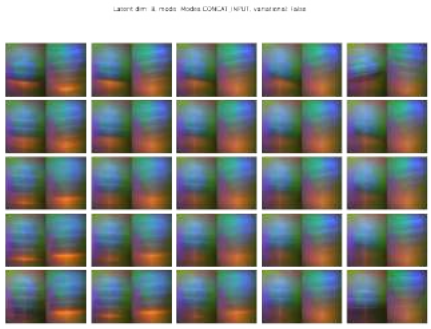
(b) *invalid combination*, left and right image do not match in the sense that such a state cannot occur in reality.



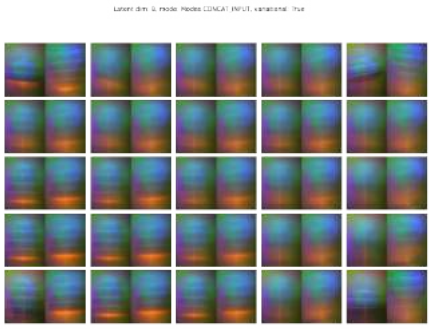
(c) *nonlinear changes*, instead of gradually morphing just the features that change from image A to B the interpolations contain new features or change features in a way that cannot be regarded as ïn betweenÄ and B.

(d) *discrete change*, most of the change happens in one step instead of a smooth transformation.

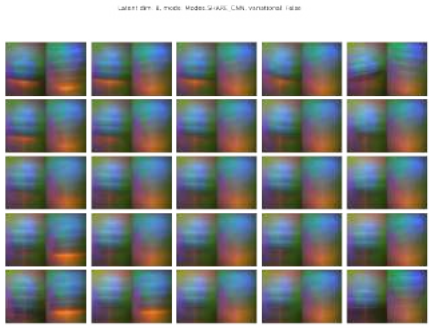
Figure 4.8.: For visual evaluation of the latent space we particularly look for four different types of artifacts in the interpolations.



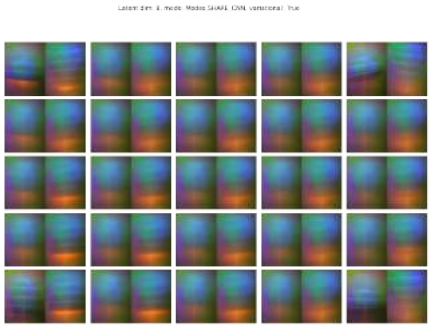
(a) standard, architecture: concat-input, latent dim: 8



(b) variaional, architecture: concat-input, latent dim: 8

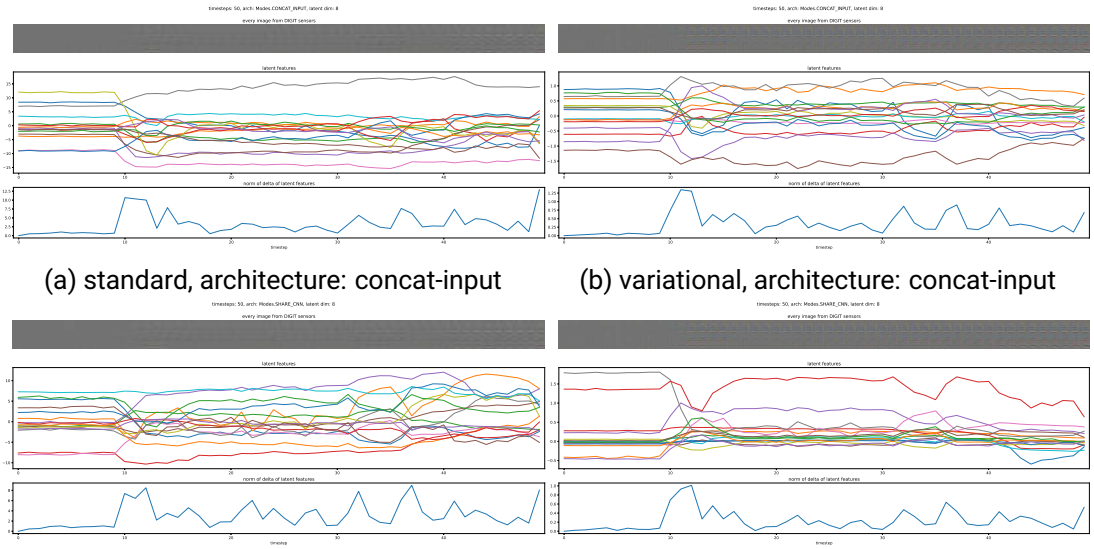


(c) standard, architecture: share-cnn, latent dim: 8



(d) variational, architecture: share-cnn, latent dim: 8

Figure 4.9.: Latent space interpolations with four different models. Left column was produced by a standard autoencoder and the right column by a variational one. The top row uses architecture mode concat-cnn while the bottom row uses sharLatent space of two different mointerpolations with four different models. Left column was produced by a standard autoencoder and the right column by a variational one. The top row uses architecture mode concat-input while the bottom row uses share-all.



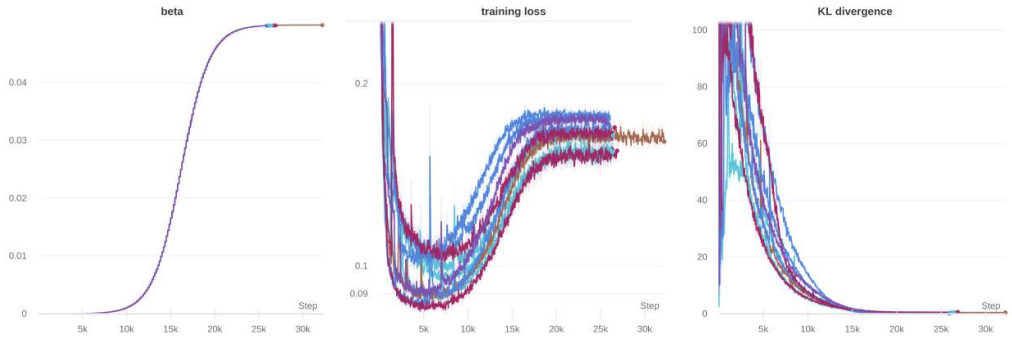
(a) standard, architecture: concat-input

(b) variational, architecture: concat-input

(c) standard, architecture: share-cnn

(d) variational, architecture: share-cnn

Figure 4.10.: Comparison of the latent representation of standard and variational autoencoders with latent dimensionality 8 over the course of an episode. Left and right column compare standard and variational while top and bottom compare concat-input architecture with share-cnn.



(a) Weight of the KL-Divergence loss β

(b) Overall training loss: reconstruction loss plus KL-Divergence loss weighted by β.

(c) Unweighted KL-Divergence of the output of the model to a standard normal distribution.

Figure 4.11.: Normality of the latent space and reconstruction performance seem to be a trade-off. The plots visualize the competing reconstruction and KL-Divergence loss during training. As β is increased gradually the KL-Divergence reduces but the reconstruction loss increases such that the overall performance is worse. Multiple runs with different hyperparameters are visualized.

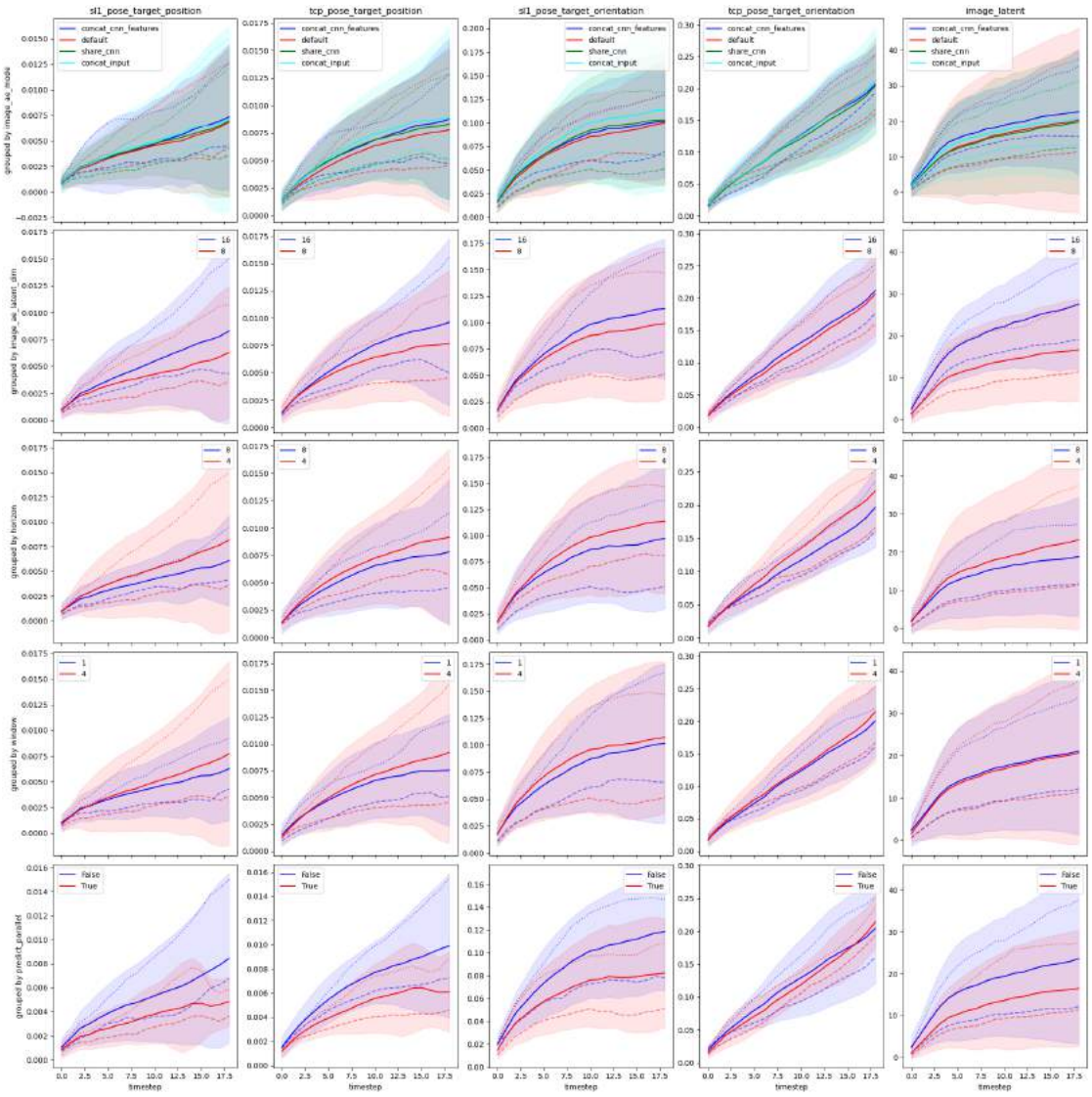


Figure 4.12.: Visualizes the error in the individual components that the transition model is predicting. Each row groups the learning runs by a different hyperparameter. The solid line and the area around it are the mean and 3 standard deviation of the group for each time step. The dashed and dotted line are the forward prediction error of the best and worst run in the group according to the accumulated total error of each run. The full details of this experiment are given in subsection A.1.2

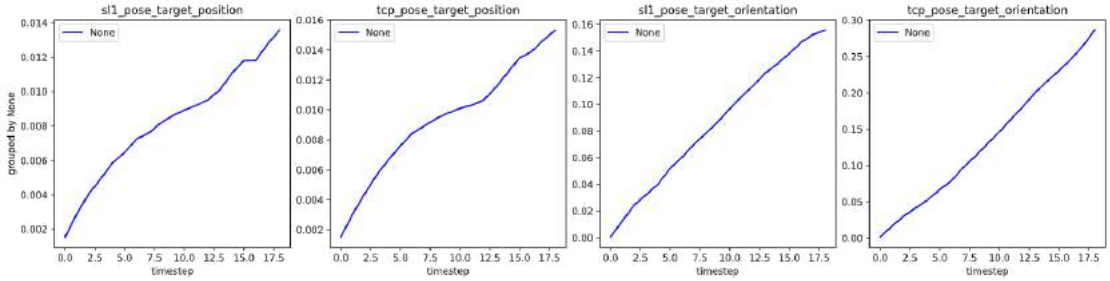


Figure 4.13.: Error in the individual components the transition model without tactile feedback predicted on the test data. The prediction is a full forward prediction of the last 20 time steps. The full details of this experiment are given in subsection A.2.3.

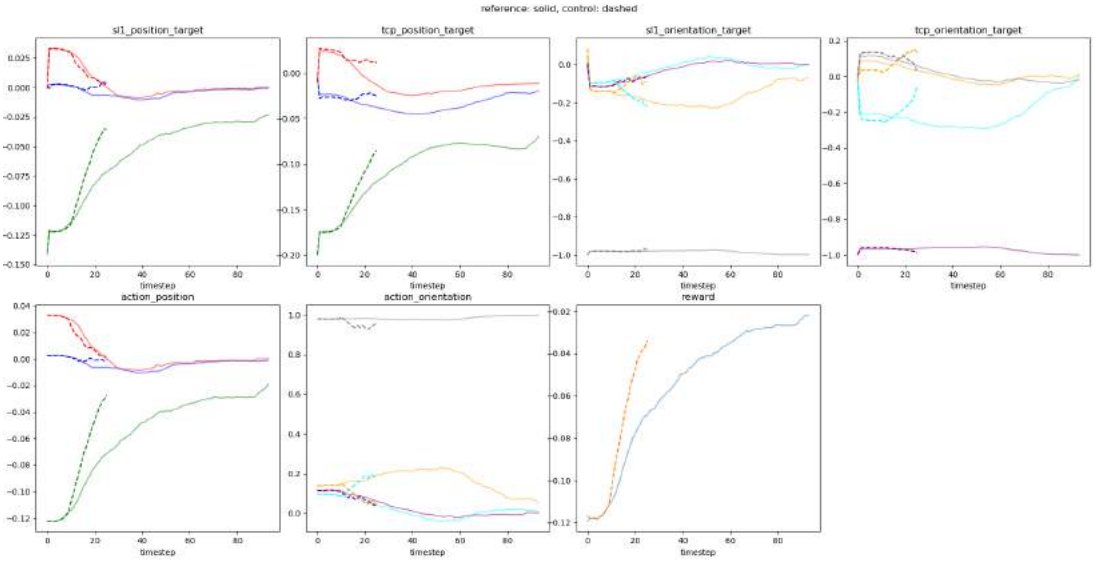


Figure 4.14.: Trajectory produced by our controller. The first 10 executed actions are taken from the reference trajectory. From time step 10, the controller takes much larger actions than the reference.

5. Conclusion

With Assembly Gym, a physical contact-rich tactile manipulation environment for assembly challenges was set up. To our knowledge, such a complex multi-modal setup, involving vision-based tactile sensors, a 6-DoF manipulator and object tracking, is unprecedented.

From this environment, many episodes of tactile assembly data were sampled autonomously and compiled into a large dataset. Using autoencoders, a representation of tactile sensor images was learned in a fully unsupervised manner. We showed that these representations can generalize to some extent across different tactile sensors and encode relevant information for tactile control. Our contribution paves the way for further research into highly generalizable tactile representations that can overcome the barriers to widespread application of vision-based tactile sensors.

Our experiments on learning to autoencode tactile sensor images resulted in several key discoveries concerning the quality of the learned representation.

- The dimensionality of the latent space has a large impact on the quality of the representation. A larger dimensionality tended to produce less usable latent representations for latent space interpolation and for approximation of the transition function.
- Against our intuition, separately encoding the images resulted in better performance than encoding a latent representation jointly. When dealing with images from multiple tactile sensors, the architecture of the autoencoder network has to be designed with care. But this may not always be the case. As the generalization of tactile representations progresses, and more varied datasets become available, this conclusion needs reevaluation.
- Variational autoencoders cannot be regarded as superior over a standard autoencoder. We experienced that they are more difficult to train, as the regularization needs to be tuned carefully.

- The learned representation showed a certain generalizability to different gels used.

By training a transition model of our environment, we showed that a fully unsupervisedly learned representation can be task-relevant. As we added the encoded tactile state to the state space of our transition model, its predictive performance improved. However, making predictions for future tactile states in latent space was difficult for the trained models.

During experimentation with transition model learning we found that:

- Predicting multiple time steps ahead with a single network pass significantly improved learning.
- Passing a window of past observations into the model did not show large improvements in performance, but the used window size of four could have been too small. We suggest that either very large input windows, or alternatively recurrent models trained on whole trajectories, are needed to observe notable effects of an observation window over the past.
- A significant challenge of learning the transition function came from our multimodal state space with different types of sensors. It seemed that simply defining the state as a concatenation of all sensors of the environment did not result in a good environment state for transition model learning. For example, using a tactile representation of larger dimension decreased the predictive performance over all sensors in the environment even though this larger dimensional space, in theory, should include at least the same amount of information as a smaller one. As already suggested by related research, using a model that fuses the different sensors modes into a joint distribution can be beneficial [6].

Lastly, we attempted to solve our assembly task by combining the learned transition model with a CEM trajectory planner. Model predictive control with the basic formulation of CEM does not perform well in our high-dimension long horizon scenario. Even after including several minor task-specific improvements to the algorithm and parameter tuning the controller did not solve the task. CEM is one of the simplest methods for probabilistic trajectory planning. Using a more sophisticated planner will likely yield much better control performance.

6. Outlook

Attempting to solve a challenging tactile manipulation task, we uncover multiple opportunities for future research. These opportunities range from small but significant improvements to our method to entirely different approaches.

6.1. Towards Robust Representations

The regular replacement of gel pads of visual tactile sensors due to wear and tear is an obstacle for experimenting with on-policy reinforcement learning methods. A gel exchange is a large discrete change in the environment that would likely disturb on-policy learning. Current tactile representations are not yet robust enough to the variations between different gel pads. We propose some directions that could lead to the needed robust tactile representations.

Data Augmentation During our analysis in subsection 4.2.2, we suggested that data augmentation can highly improve representation learning. This suggestion is also backed up by related work [7]. While a readily available toolbox of data augmentation methods for natural RGB images exists, it makes sense to review their applicability and suitability to the more restricted domain of tactile sensor images.

Sampling Larger More Varied Datasets We think that data augmentation alone might not suffice to overcome this barrier and the creation of larger more varied datasets of tactile images is needed. As the DIGIT sensor is relatively cheap in production, it is feasible to produce a larger batch of varied designs including many different types of gels. This variation in the hardware, combined with multiple tasks that are designed to cover up as much of the measurement space of the sensor as possible, can make learning highly

generalizable representations of tactile sensor images possible. Representations that are robust to variations in gel could make an otherwise non-stationary tactile manipulation environment stationary.

Learning To Separate Semantics Directly Methods from the field of image generation that are able to separate image semantics from style could be applied to tactile representation learning. Regarding the specific gel and sensor used as style and the underlying tactile measurements as semantics, these methods appear promising to produce more robust representations.

The pose estimations coming from our tracking system are subject to non-stationary calibration errors and therefore augmenting them with random decalibrations would make sense too.

6.2. Sampling-Based Trajectory Planning

We envision multiple improvements on the usage of sampling-based trajectory optimization methods with an approximate transition model. One improvement would be to initialize the planning with an interpolation obtained with our method introduced in subsection 3.4.1 or with a collision-aware motion planner in simulation.

A very important component of a sampling base optimizer is the design of the cost function used to rank the samples. If invalid trajectories are correctly penalized, sampling steps will converge into regions of the state space that yield trajectories close to our data that our model can also correctly model. In contrast, if this is not the case and invalid trajectories, far away from where the model is able to make good predictions, are not penalized correctly, the variance in the elite can explode.

A common regularization method is using a probabilistic transition model that outputs an uncertainty for its predictions, which can be incorporated in the cost function. An alternative approach, that also works with a deterministic model, could be to compute the likelihood of a trajectory under the data distribution. In a more generalized way this could be the KL-divergence of the distribution the planner is sampling from to the distribution of the training data.

6.3. Model Free Learning

As our results suggested, learning tactile representations that generalize, over the variance introduced by the specific gel pads used, is possible. By engineering more durable gel pads and constraining the environment to be more gentle, sampling data in another order of magnitude becomes possible. We envision that with a highly automated environment, where the only human intervention is a gel pad replacement, application of less sample efficient on-policy learning methods become feasible. Scaling up, policies that are able to assemble complete SL-Block structures could be learned.

Bibliography

[1] R. S. Johansson and J. R. Flanagan, "Coding and use of tactile signals from the fingertips in object manipulation tasks," *Nature Reviews Neuroscience*, vol. 10, pp. 345–359, May 2009. Bandiera_abtest: a Cg_type: Nature Research Journals Number: 5 Primary_atype: Reviews Publisher: Nature Publishing Group.

[2] R. Calandra, A. Owens, M. Upadhyaya, W. Yuan, J. Lin, E. H. Adelson, and S. Levine, "The Feeling of Success: Does Touch Sensing Help Predict Grasp Outcomes?," *arXiv:1710.05512 [cs, stat]*, Oct. 2017. arXiv: 1710.05512.

[3] Q. Li, O. Kroemer, Z. Su, F. F. Veiga, M. Kaboli, and H. J. Ritter, "A Review of Tactile Information: Perception and Action Through Touch," *IEEE Transactions on Robotics*, vol. 36, pp. 1619–1634, Dec. 2020. Conference Name: IEEE Transactions on Robotics.

[4] A. Yamaguchi and C. G. Atkeson, "Recent progress in tactile sensing and sensors for robotic manipulation: can we turn tactile sensing into vision?," *Advanced Robotics*, vol. 33, pp. 661–673, July 2019.

[5] M. Lambeta, P. Chou, S. Tian, B. Yang, B. Maloon, V. R. Most, D. Stroud, R. Santos, A. Byagowi, G. Kammerer, D. Jayaraman, and R. Calandra, "DIGIT: A Novel Design for a Low-Cost Compact High-Resolution Tactile Sensor With Application to In-Hand Manipulation," *IEEE Robotics and Automation Letters*, vol. 5, pp. 3838–3845, July 2020. Conference Name: IEEE Robotics and Automation Letters.

[6] M. A. Lee, Y. Zhu, P. Zachares, M. Tan, K. Srinivasan, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg, "Making Sense of Vision and Touch: Learning Multimodal Representations for Contact-Rich Tasks," *arXiv:1907.13098 [cs]*, July 2019. arXiv: 1907.13098.

[7] O. Spector and D. Di Castro, "InsertionNet – A Scalable Solution for Insertion," *arXiv:2104.14223 [cs]*, Apr. 2021. arXiv: 2104.14223.

[8] Z. Ding, Y.-Y. Tsai, W. W. Lee, and B. Huang, "Sim-to-Real Transfer for Robotic Manipulation with Tactile Sensory," *arXiv:2103.00410 [cs]*, Feb. 2021. arXiv: 2103.00410.

[9] Y. She, S. Wang, S. Dong, N. Sunil, A. Rodriguez, and E. Adelson, "Cable Manipulation with a Tactile-Reactive Gripper," *arXiv:1910.02860 [cs, eess]*, June 2020. arXiv: 1910.02860.

[10] S. Tian, F. Ebert, D. Jayaraman, M. Mudigonda, C. Finn, R. Calandra, and S. Levine, "Manipulation by Feel: Touch-Based Control with Deep Predictive Models," *arXiv:1903.04128 [cs]*, Mar. 2019. arXiv: 1903.04128.

[11] M. I. Tiwana, S. J. Redmond, and N. H. Lovell, "A Review of Tactile Sensing Technologies with Applications in Biomedical Engineering," June 2012.

[12] O. Kroemer, S. Niekum, and G. Konidaris, *A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms*. July 2019.

[13] K. Nottensteiner, A. Sachtler, and A. Albu-Schäffer, "Towards Autonomous Robotic Assembly: Using Combined Visual and Tactile Sensing for Adaptive Task Execution," *Journal of Intelligent & Robotic Systems*, vol. 101, p. 49, Mar. 2021.

[14] L. Cramphorn, B. Ward-Cherrier, and N. F. Lepora, "Tactile manipulation with biomimetic active touch," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 123–129, May 2016.

[15] W. Yuan, S. Dong, and E. H. Adelson, "GelSight: High-Resolution Robot Tactile Sensors for Estimating Geometry and Force," *Sensors*, vol. 17, p. 2762, Dec. 2017. Number: 12 Publisher: Multidisciplinary Digital Publishing Institute.

[16] F. R. Hogan, J. Ballester, S. Dong, and A. Rodriguez, "Tactile Dexterity: Manipulation Primitives with Tactile Feedback," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8863–8869, May 2020. ISSN: 2577-087X.

[17] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models," *arXiv:1805.12114 [cs, stat]*, Nov. 2018. arXiv: 1805.12114.

[18] S. Wang, M. Lambeta, P.-W. Chou, and R. Calandra, "TACTO: A Fast, Flexible and Open-source Simulator for High-Resolution Vision-based Tactile Sensors," *arXiv:2012.08456 [cs, stat]*, Dec. 2020. arXiv: 2012.08456.

[19] N. O. Lambert, A. Wilcox, H. Zhang, K. S. J. Pister, and R. Calandra, "Learning Accurate Long-term Dynamics for Model-based Reinforcement Learning," *arXiv:2012.09156 [cs]*, Dec. 2020. arXiv: 2012.09156.

[20] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss Functions for Neural Networks for Image Processing," *arXiv:1511.08861 [cs]*, Apr. 2018. arXiv: 1511.08861.

[21] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv:1312.6114 [cs, stat]*, May 2014. arXiv: 1312.6114.

[22] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning Latent Dynamics for Planning from Pixels," *arXiv:1811.04551 [cs, stat]*, June 2019. arXiv: 1811.04551.

[23] D. Ha and J. Schmidhuber, "World Models," *arXiv:1803.10122 [cs, stat]*, Mar. 2018. arXiv: 1803.10122.

[24] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework," Nov. 2016.

A. Appendix

All the parameters and details of carried our experiments are documented in this chapter. It is structured into two section one for autoencoder experiments and one for transition model learning experiments. We provide a description, the hyperparameters and visualizations of the result for each experiment.

For all learning experiments the optimizer Adam is used.

A.1. Autoencoder Experiments

When strided convolutions are used for up or down sampling, they are grouped by the number of channels to reduce the amount of parameters. Experiments, that are not documented here, showed that this grouping does not have a strong negative effect on the learning performance. We always use SSIM loss as described in subsection 3.3.2.

The experiments in this section are grouped by the used dataset.

A.1.1. Fingerprint Dataset

Many experiments were carried out to find suitable parameters for most of the hyperparameters. The size of latent dimension was kept fixed at 16. At that time only the default architecture mode was used training for left and right DIGITs separately.

Hyperparameters The hyperparameters were mostly identical for the left and right DIGIT model. If different then they are shown separated by a forward slash: left model/right model. During this experiment, the batch size equals the number of images a loss is computed for, as the training for the left and right DIGIT model is separated completely.

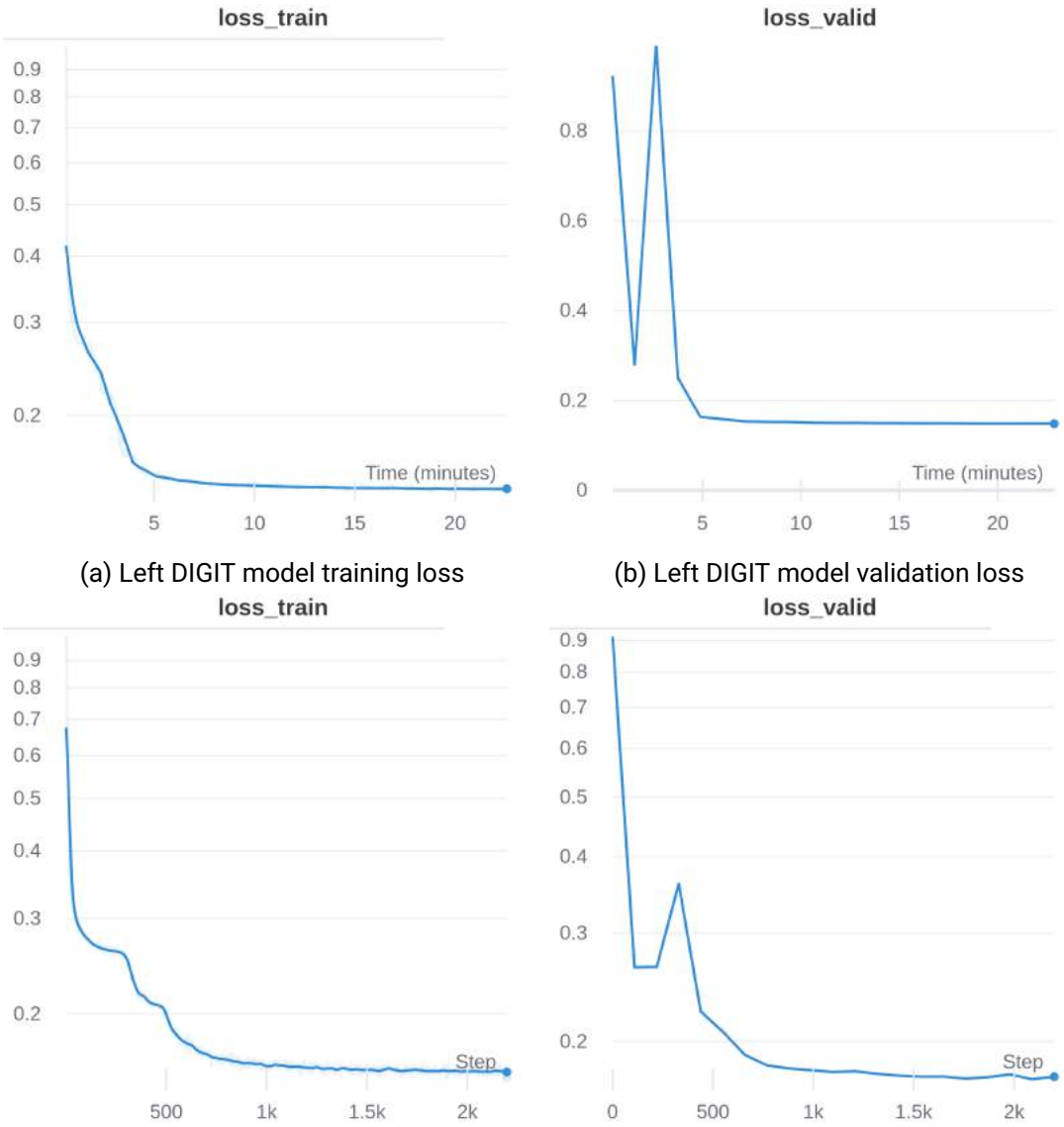
latent dimension	16
kernel size	5/3
input image size	[320,240]
up-/downsampling	grouped conv
activation	ELU
batch norm	True
cnn block channels	[[32],[32,64],[64,128],[128,256],[256,512],[512,8]]
mlp hidden channels	[]
variational	false
learning rate	0.004643/0.00495
batch size	64
epochs	20

Performance Figure Figure A.1 shows the learning performance of the autoencoders during training and validation.

A.1.2. Smooth Dataset

A series of experiments was carried out, to determine good values for the hyperparameters on this dataset, that is not documented here. Then, with these hyperparameters, a model for each architecture as defined in Figure 3.3 and different latent dimensions is trained.

Hyperparameters The hyperparameters that vary for each training run are given in brackets as a set $(\text{variation } 1, \text{variation } 2, \dots)$. The reduced input image size means that the image is sampled down before being passed to the model, and later the output is



(a) Left DIGIT model training loss

(b) Left DIGIT model validation loss

(c) Right DIGIT model training loss

(d) Right DIGIT model validation loss

Figure A.1.: Learning performance of autoencoders on fingerprint dataset. Training and validation loss converge. As the images of the left and right DIGITs are slightly different and are subject to very different tactile influences during sampling, the model for the right DIGIT images is more difficult to learn.

sampled up again. The number of images that is passed through the model is actually the double of the batch size, as we are passing a left and a right image simultaneously, even for the default architecture. The parameter *filter block height* filters that dataset down to data points for which the SL-Block is at least 4cm close to its target location on the z-axis (pointing upwards in our case).

latent dimension	(2,4,8,16)
arch	(default, share-cnn, share-mlp, share-all, concat-input concat-shared-cnn-features, concat-cnn-features)
kernel size	5
input image size	[160,120]
up-/downsampling	grouped conv
activation	LeakyReLU
batch norm	True
cnn block channels	[[16, 32],[32,64],[64,128],[128,256],[256,8]]
mlp hidden channels	[128]
variational	false
learning rate	0.005
batch size	64
epochs	100
filter block height	0.04

Performance We evaluate three performance measures: training and validation loss grouped by varied parameter (Figure A.2), latent space interpolation (Figure A.3) and latent space over time (Figure A.4).

A.1.3. Variational

Training variationally introduces a few additional hyperparameters. As described in section 3.3.3, paragraph Variational Autoencoder, the parameter β is scaled up during training. This scaling is implemented as a function of the epoch that is parameterized by α and δ

$$\beta(x) = \frac{\delta}{(1 + e^{-(\alpha x - 10)})}$$

which is just a scaled and shifted sigmoid function.

The variaional autoencoder learned slower than the standard autoencoder. The learning speed was increased by scheduling the learning rate with an exponential learning rate scheduler $f(x) = cd^x$ where c is the initial learning rate and d the decay factor.

Hyperparameters

latent dimension	(2,4,8,16)
h arch	(share-cnn, concat-input)
kernel size	5
input image size	[160,120]
up-/downsampling	grouped conv
activation	LeakyReLU
batch norm	True
cnn block channels	[[16, 32],[32,64],[64,128],[128,256],[256,16]]
mlp hidden channels	[256]
variational	false
initial learning rate	0.02
batch size	90
epochs	150
filter block height	0.04
kl lossweightmax	0.05
γ	0.98

Performance We did not evaluate all the variational autoencoders on the testdata. Figure 4.9 shows that it does not perform as well as the standard autoencoders that we learned.

A.1.4. Different Gels

The same sequence of actions is executed on the environment, but three different gel pads are used in the DIGIT sensor: one with a fingerprint patter and surface coating (gel 0), one with smooth surface but without coating (gel 1) and one with smooth surface and coating (gel 2). Non of the gel pad was used during collection of the datasets described in

section 4.1, but the gel with smooth surface and coating was manufatured in the same batch as our training gels.

Then the model from section A.2.2, paragraph Hyperparameters with latent dimension 8 and architecture share-all is used to encode all three trajectories into latent space. Figure A.6a, A.6b, Figure A.6c visualize the encoded latent space together with the rest of the observations of the episode.

A.2. Transition Model Experiments

The transition model experiments use pretrained autoencoders to learn the full transition model on one of the datasets.

Important hyperparameters are usually window, horizon, parallel prediction, autoencoder latent dimension and architecture, delta, layers, features per layer.

A.2.1. Fingerprint Dataset

The left and right autoencoder models documented in subsection A.1.1 are used to encode the images and being fine-tuned during training of the transition model.

Hyperparameters One learning experiment with following hyperparameters was carried out:

autoencoder architecture	default
autoencoder latent dimension	16
batch size	16
filter block height	0.04
delta	true
horizon	3
window	7
activation	ReLU
batch norm	true
hidden channels	6x500
learning rate	0.00085
autoencoder learning rate	0.0002

Performance We evaluate performance by looking at the validation, training losses and test forward predction error.

A.2.2. Smooth Dataset

An additional parameter, latent space loss weight, was added to increase the learning on the pose predictions. Sets of values in parantheses mean that one experiment was carried out for each of them.

Hyperparameters Sets of values in braces mean that we tried all combinations of these values. The hyperparameter configuration given below, when expanded combinatorially, results in 64 training runs.

autoencoder architecture	(concat-cnn-features, default, share-cnn, concat-input)
autoencoder latent dimension	(8, 16)
batch size	64
filter block height	0.04
delta	true
horizon	(1, 4)
window	(4, 8)
activation	ELU
batch norm	false
hidden channels	6x500
learning rate	0.001
autoencoder learning rate	0.0001
latent space loss weight	0.2
predict parallel	(true, false)
epochs	100

Forward Prediction Performance For each training run the last 20 time steps of all episodes in the test set are forward predicted. Figure 4.12 visualized the effect of the varied hyperparameters.

For each predicted component, we select the best model and visualize its prediction.

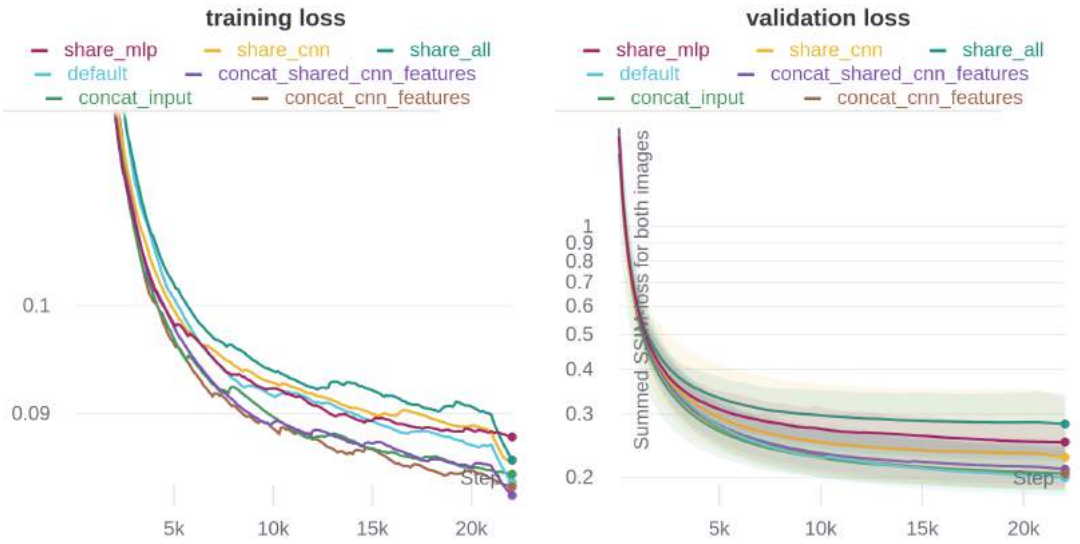
Best Models:

best in component	mode	latent dim	window	horizon	parallel
sl1 pose target position	share-cnn	8	4	4	true
sl1 pose target orientation	share-cnn	8	4	4	true
tcp pose target position	default	8	4	8	true
tcp pose target orientation	default	8	4	8	true
image latent	default	8	4	4	true

A.2.3. w/o Tactile Sensor

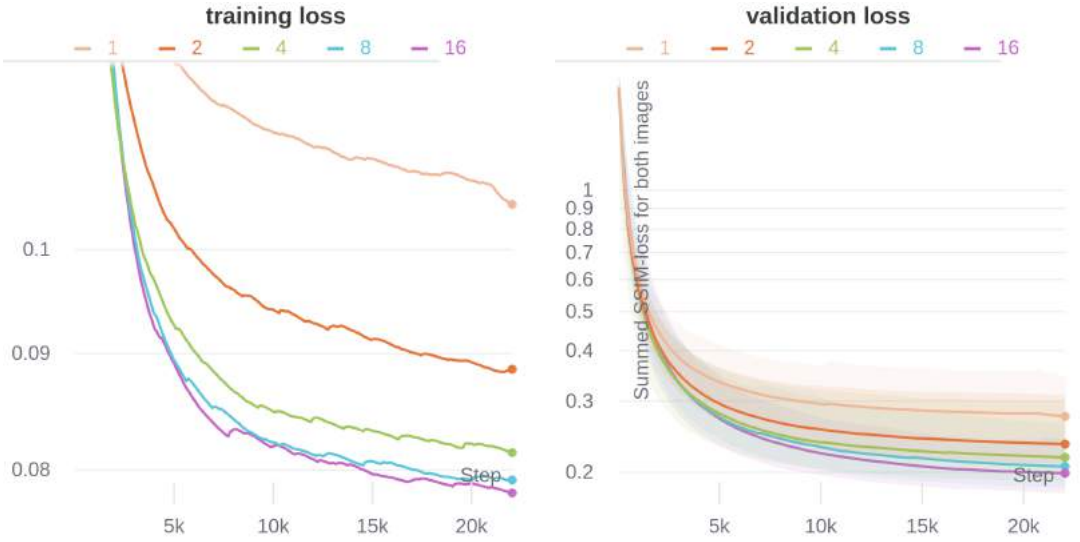
Comparison of delta and absolute prediction. When predicting absolute values the loss on the predicted quaternions has two summed components $l = c_1 + c_2$. The quaterion distance defined as $c_1(q, \hat{q}) = 1 - \langle q1, q2 \rangle^2$ and a normalization loss $c_2(q) = (1 - \|q\|)^2$

batch size	128
delta	(true, false)
horizon	8
window	4
activation	ELU
batch norm	false
hidden channels	5x300
learning rate	0.001
predict parallel	true
epochs	100



(a) grouped by architecture, line: group mean

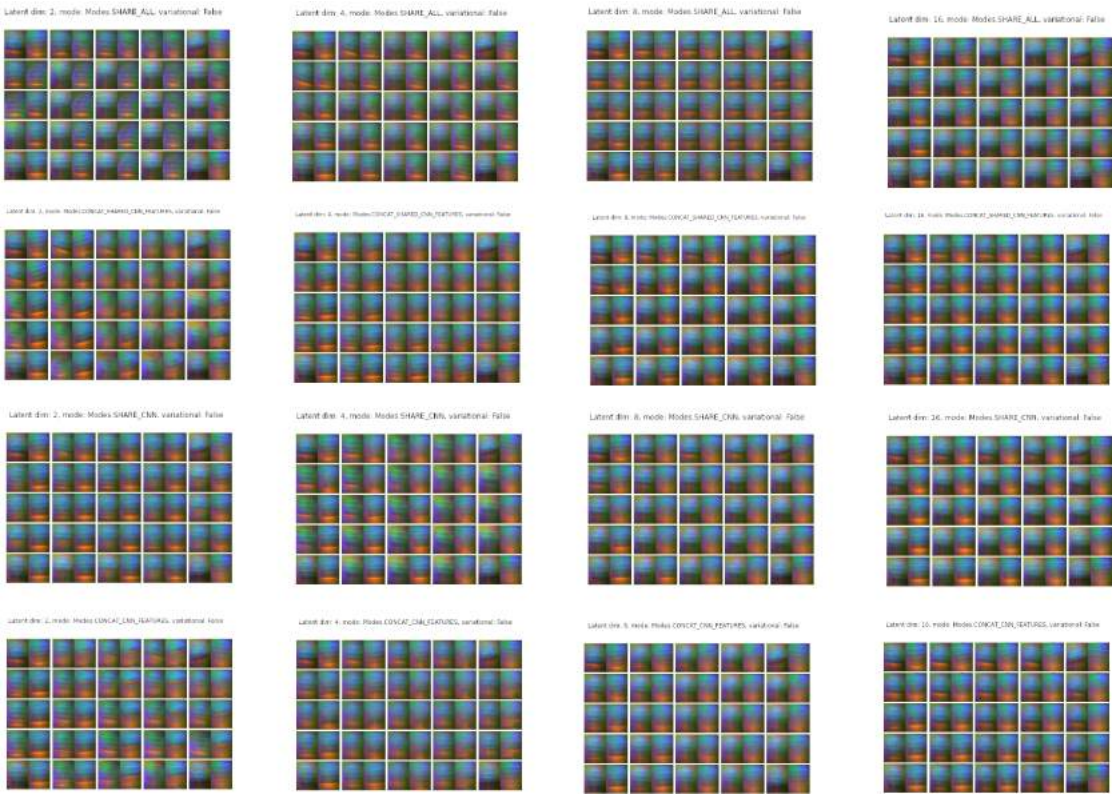
(b) grouped by architecture, line: group mean, area: group min/max



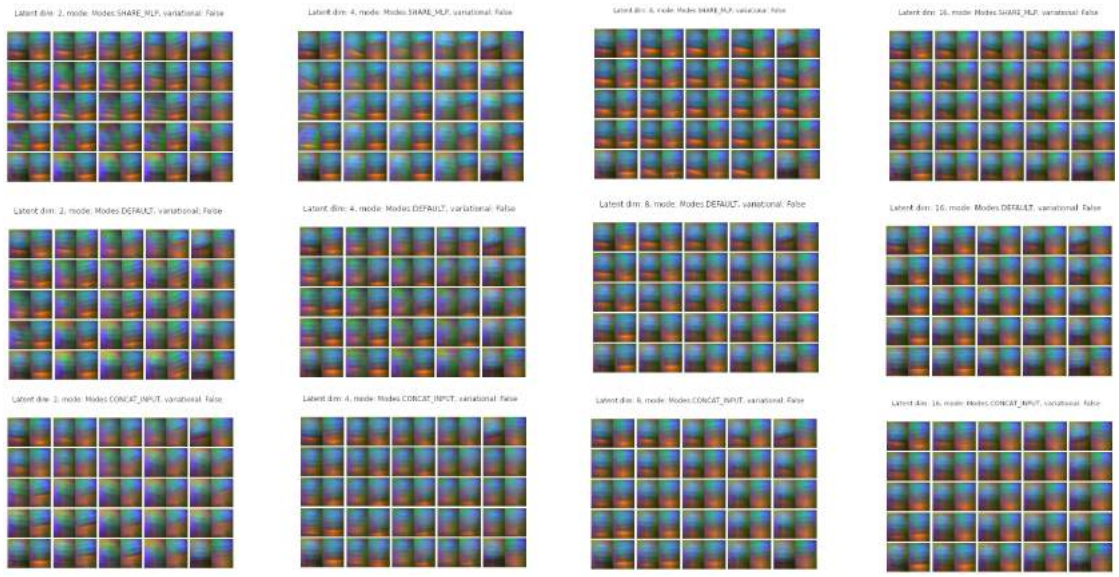
(c) grouped by architecture, line: group mean

(d) grouped by architecture, line: group mean, area: group min/max

Figure A.2.: Learning performance of autoencoders with different architectures and latent space dimensions on smooth dataset. The training loss is the average of the SSIM loss for both images. The validation loss sums the SSIM-losses of both images. A strong correlation between the latent space size and performance can be observed, while the effects of the model architecture is less significant.

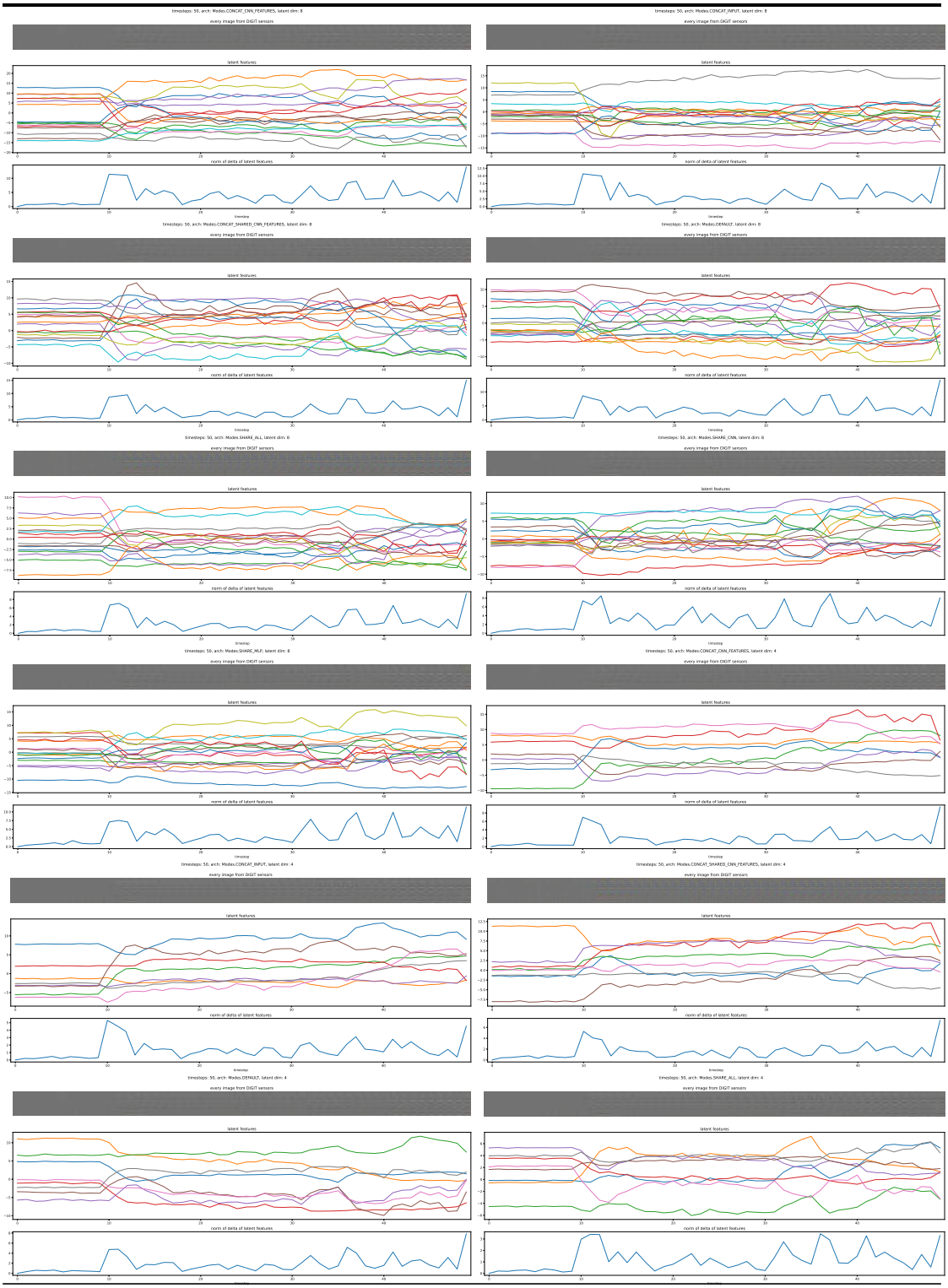


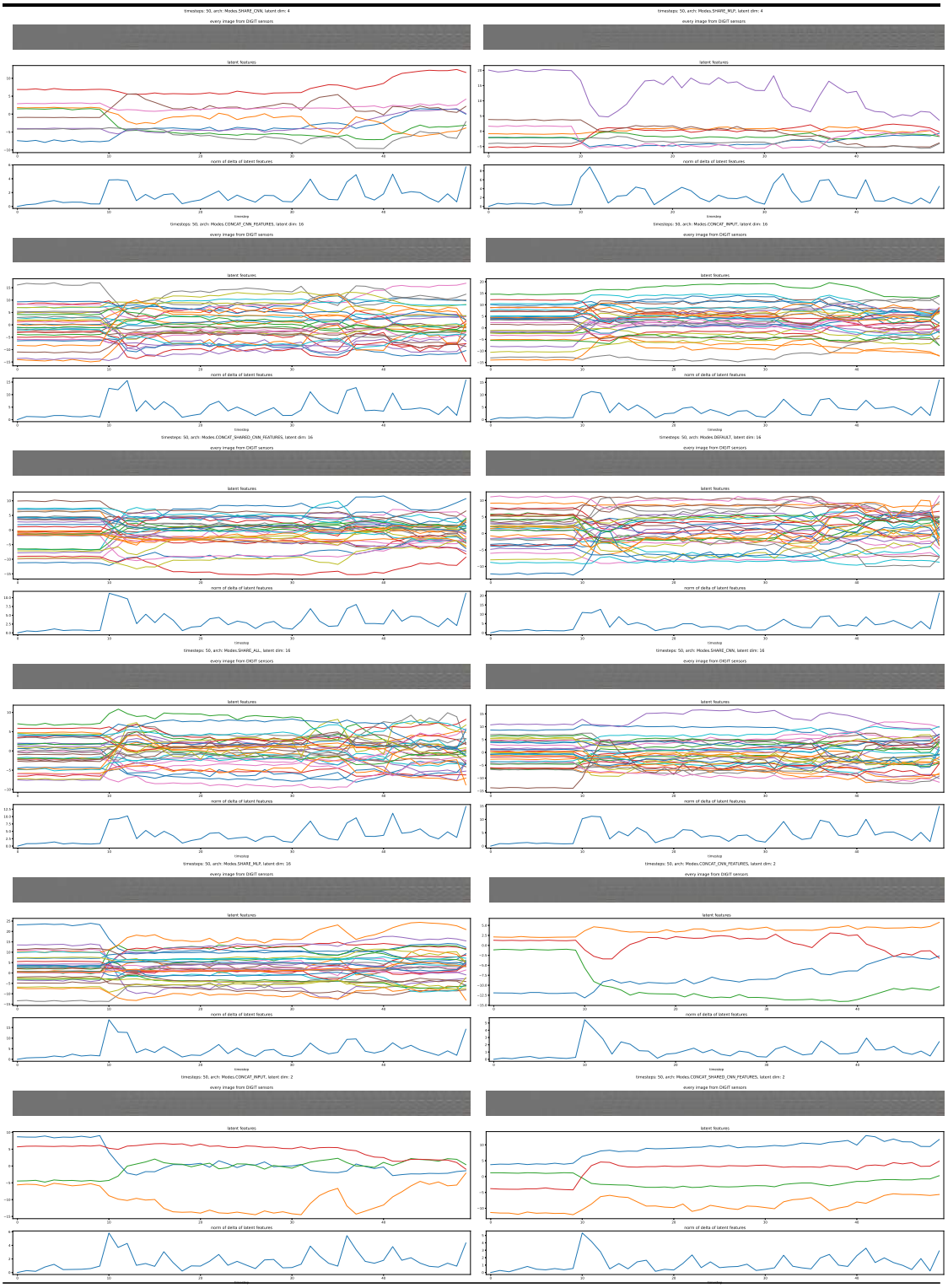
(a)



(b)

Figure A.3.: Grid of latent space interpolations, between the four images presented in Figure 4.5, for models trained under different parameters on the smooth dataset. The corners of each interpolation grid are the original images. Each row shows interpolations from models with the same architecture mode and each column uses the same latent dimension. One can see how a larger latent space helps the model to disentangle the variances in the dataset into individual components of the representation. Only very few models are able to represent the image in the top right corner well. This is due to an imbalance of the dataset. Some tactile states occur only rarely at the end of an episode.


timesteps: 50, arch: Modes.CONCAT_CNN_FEATURES, latent dim: 8
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.CONCAT_INPUT, latent dim: 8
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.SHARED_CNN_FEATURES, latent dim: 8
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.DEFAULT, latent dim: 8
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.SHARE_ALL, latent dim: 8
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.SHARE_CNN, latent dim: 8
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.SHARE_MLP, latent dim: 8
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.CONCAT_CNN_FEATURES, latent dim: 4
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.CONCAT_INPUT, latent dim: 4
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.SHARED_CNN_FEATURES, latent dim: 4
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.DEFAULT, latent dim: 4
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.SHARE_ALL, latent dim: 4
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep


timesteps: 50, arch: Modes.SHARE_CNN, latent dim: 4
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.SHARE_MLP, latent dim: 4
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.CONCAT_CNN_FEATURES, latent dim: 16
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.CONCAT_INPUT, latent dim: 16
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.CONCAT_SHARED_CNN_FEATURES, latent dim: 16
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.DEFAULT, latent dim: 16
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.SHARE_ALL, latent dim: 16
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.SHARE_CNN, latent dim: 16
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.SHARE_MLP, latent dim: 16
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.CONCAT_CNN_FEATURES, latent dim: 2
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.CONCAT_INPUT, latent dim: 2
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep
timesteps: 50, arch: Modes.CONCAT_SHARED_CNN_FEATURES, latent dim: 2
every image from DIGIT sensors
latent features
norm of delta of latent features
timestep

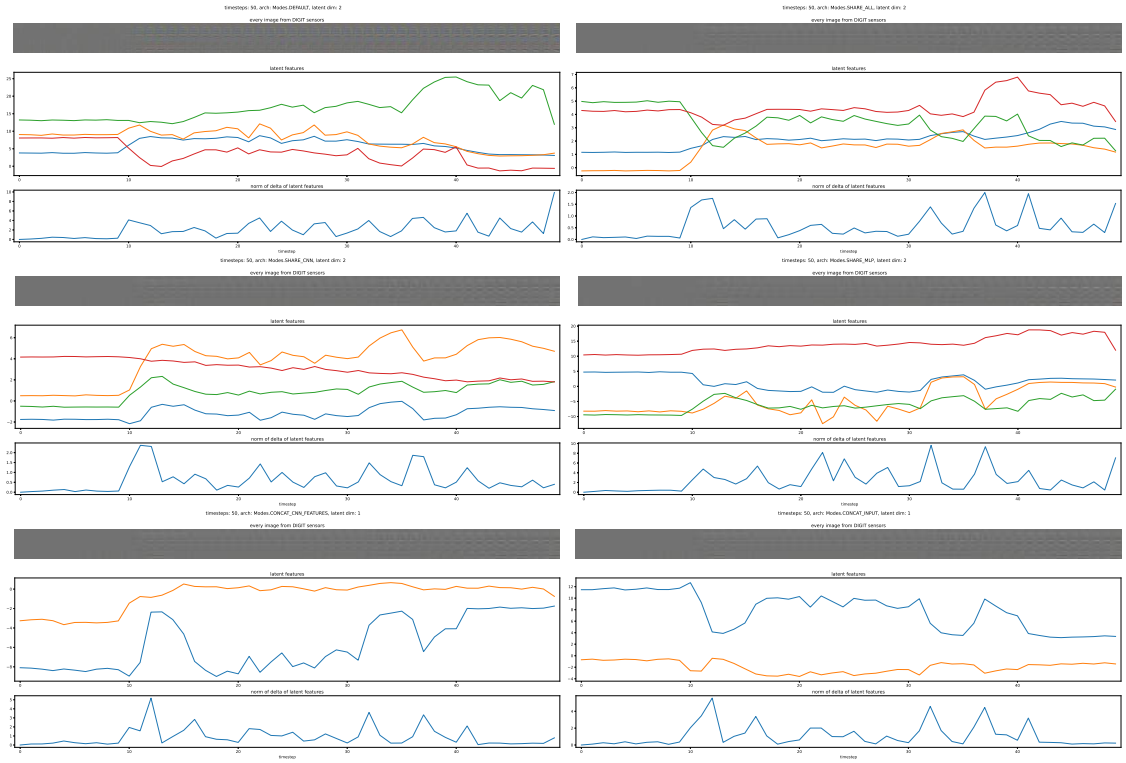
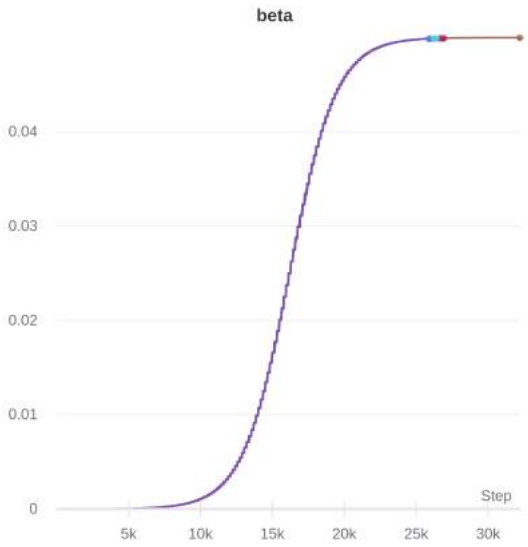
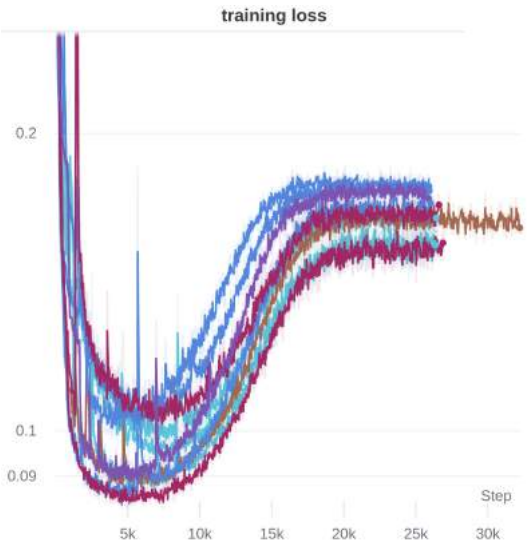


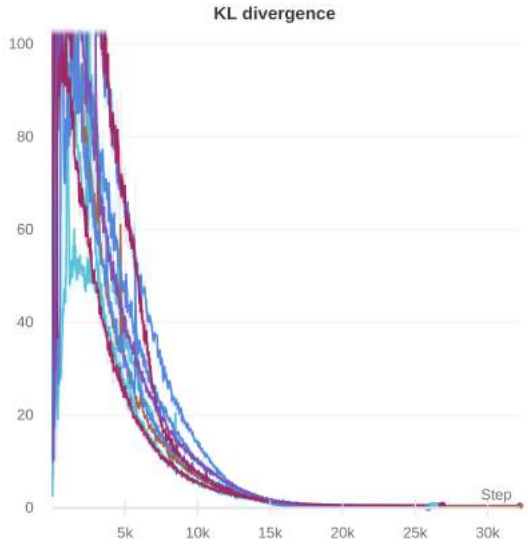
Figure A.4.: Latent space of an episode for all models of the experiment described in this section. Each figure shows the difference of the sensor image to the first image of the episode, the individual components of the representation and the change in the vector of the representation measured by the Euclidean distance. Some patterns can be observed. There are often a few components in the latent representation that cause most of the delta in whole representation. Observing the delta images closely it can be seen that the magnitude of change in the latent representation roughly correlates with the change in tactile measurement seen in the images.



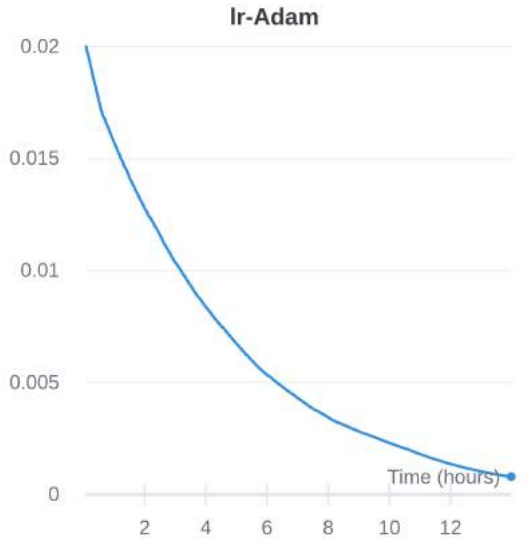
(a) Weight of the KL-Divergence-loss β



(b) Overall training loss: reconstruction loss plus kl-loss weighted by β



(c) Unweighted KL-Divergence of the output of out model to a standard normal distribution.



(d) Learning rate

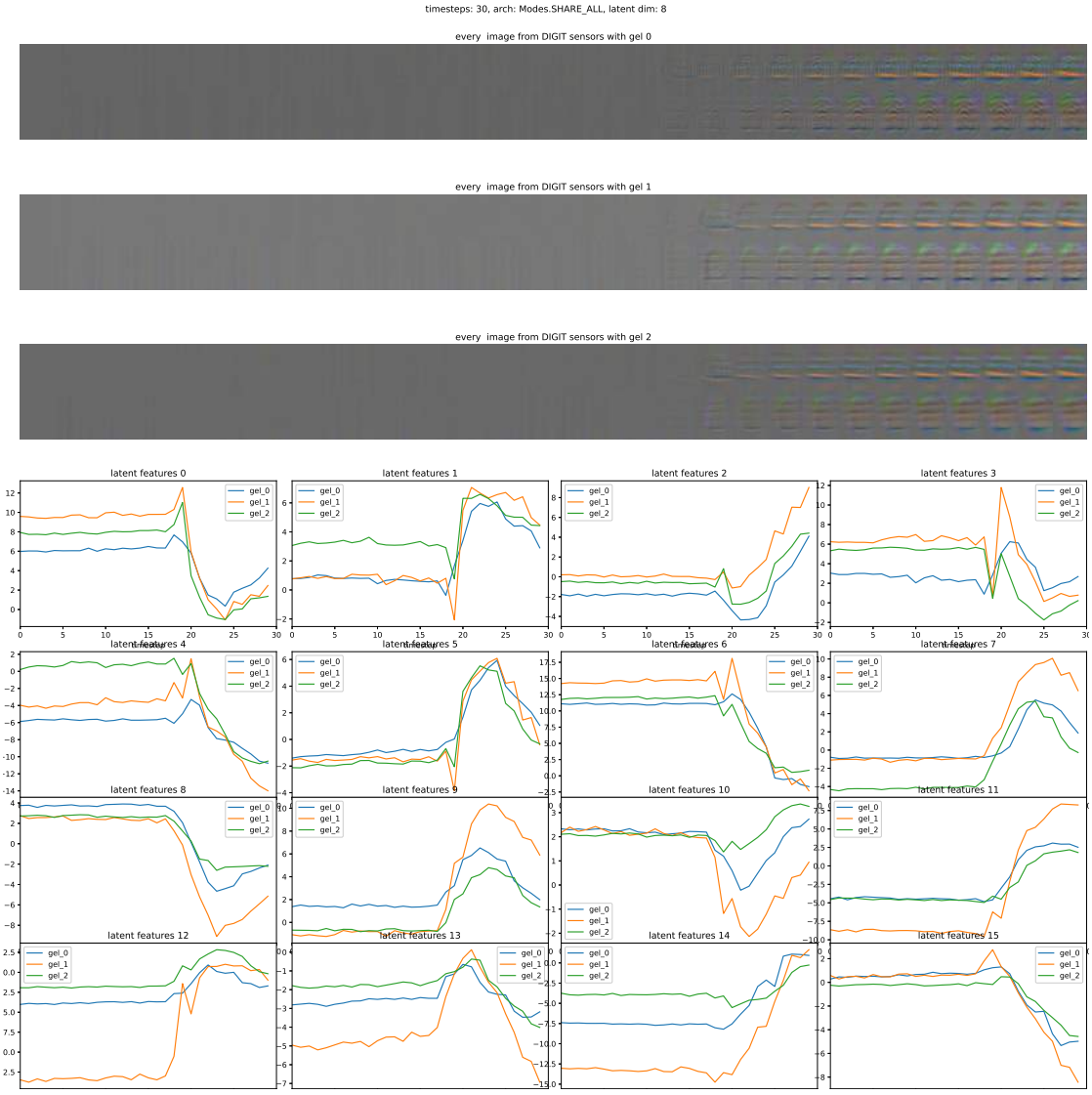
Figure A.5.: Logs of training variational autoencoders.


timesteps: 30, arch: Modes.SHARE_ALL, latent dim: 8
every image from DIGIT sensors with gel 0
every image from DIGIT sensors with gel 1
every image from DIGIT sensors with gel 2
latent features 0
latent features 1
latent features 2
latent features 3
latent features 4
latent features 5
latent features 6
latent features 7
latent features 8
latent features 9
latent features 10
latent features 11
latent features 12
latent features 13
latent features 14
latent features 15
gel_0
gel_1
gel_2

(a) Motion 1


timesteps: 30, arch: Modes.SHARE_ALL, latent dim: 8
every image from DIGIT sensors with gel 0
every image from DIGIT sensors with gel 1
every image from DIGIT sensors with gel 2
latent features 0
latent features 1
latent features 2
latent features 3
latent features 4
latent features 5
latent features 6
latent features 7
latent features 8
latent features 9
latent features 10
latent features 11
latent features 12
latent features 13
latent features 14
latent features 15
gel_0
gel_1
gel_2

(b) Motion 2



(c) Motion 3

Figure A.6.: Three different motions are executed three times with a different type of gel pads mounted in the DIGIT sensors. Each component of the latent space is plotted in an individual plot together with the representations coming from different gels. The sensor images are visualized as difference to the first image of the episode.

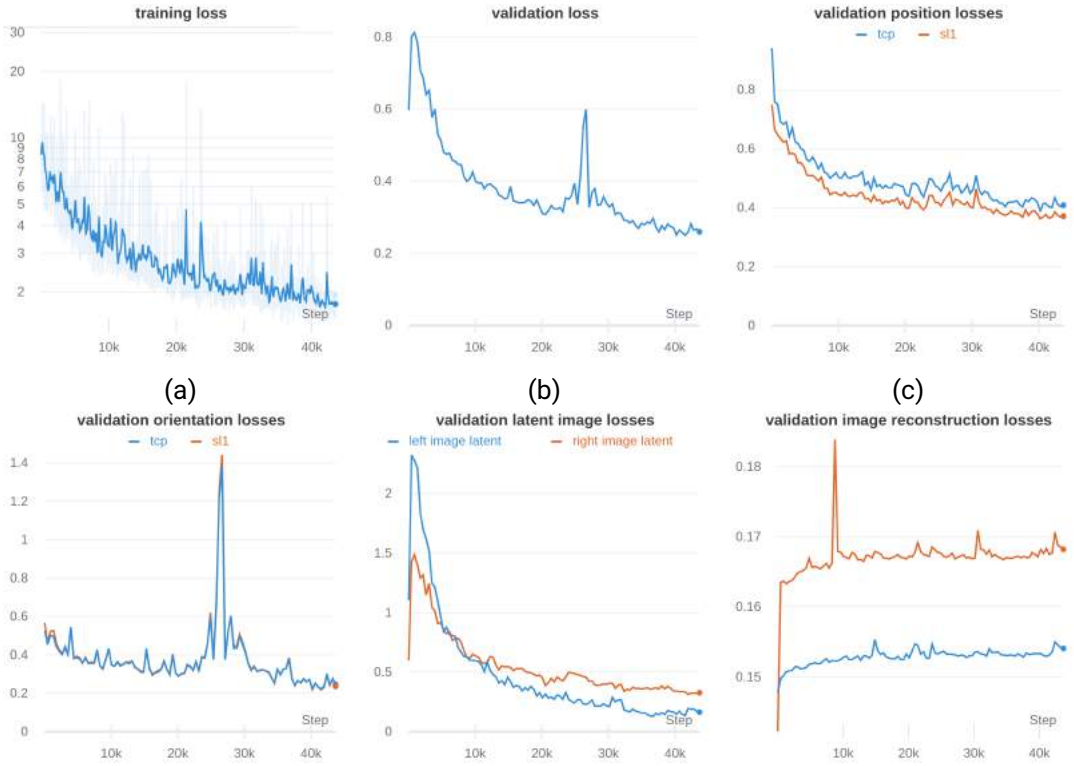
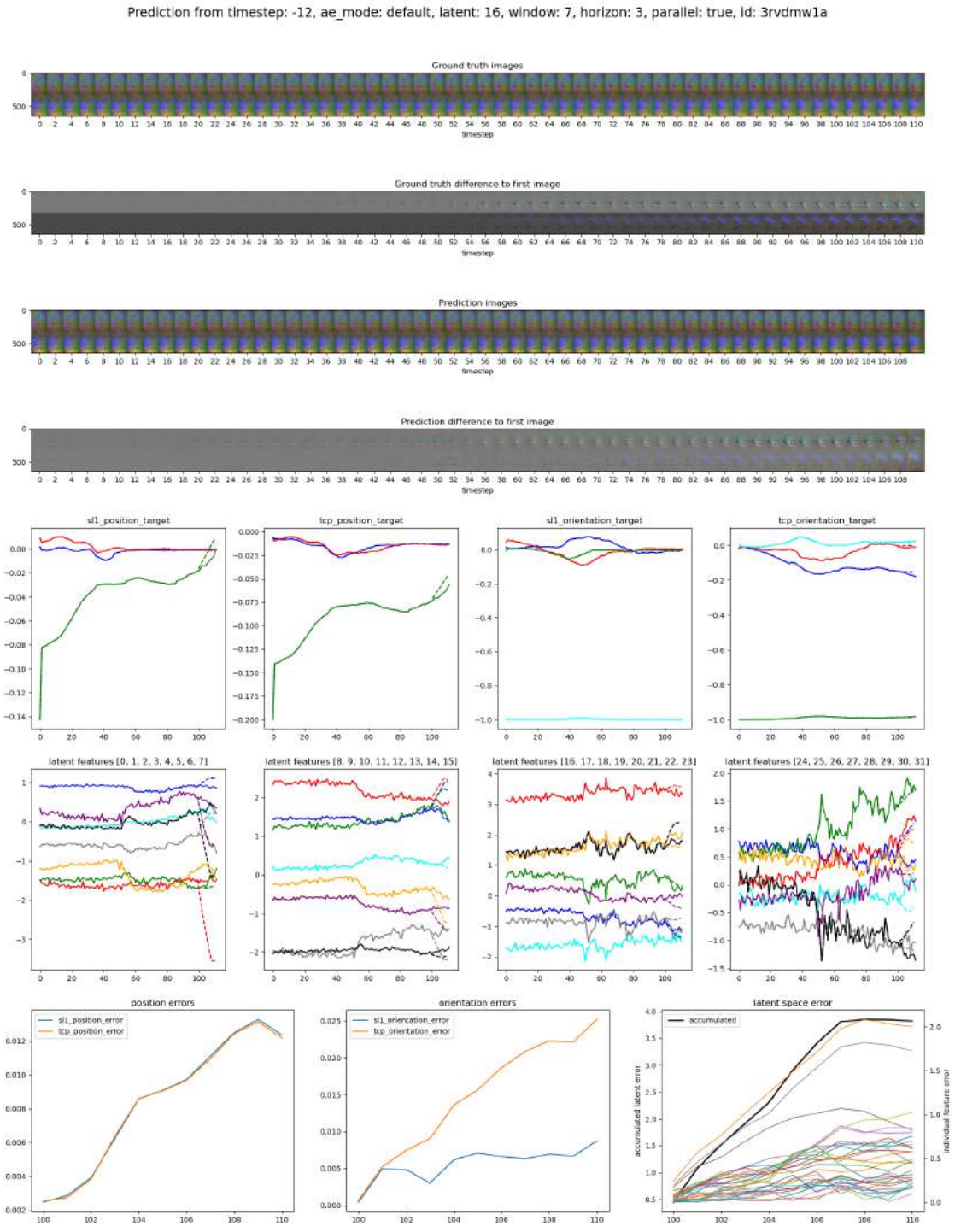
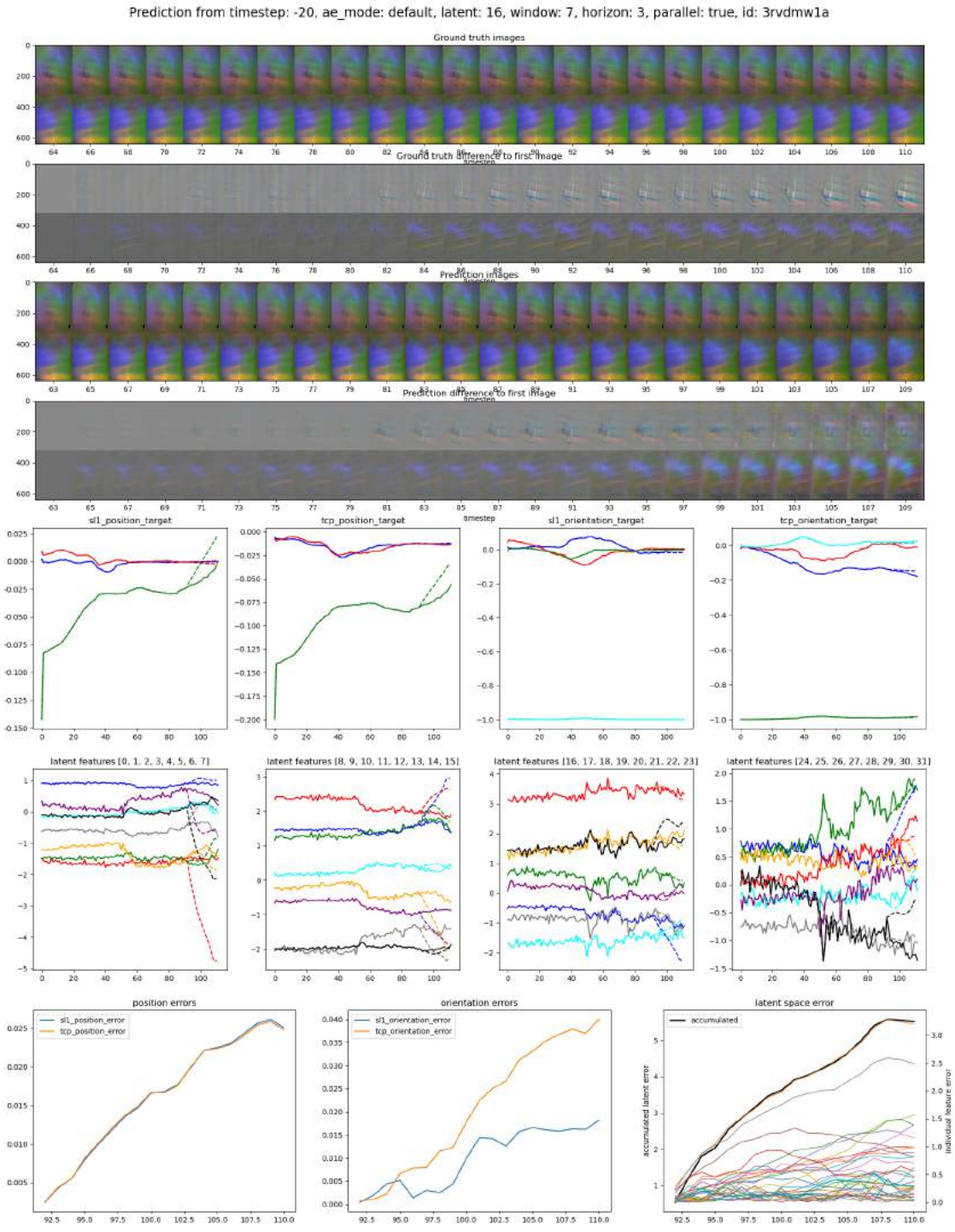


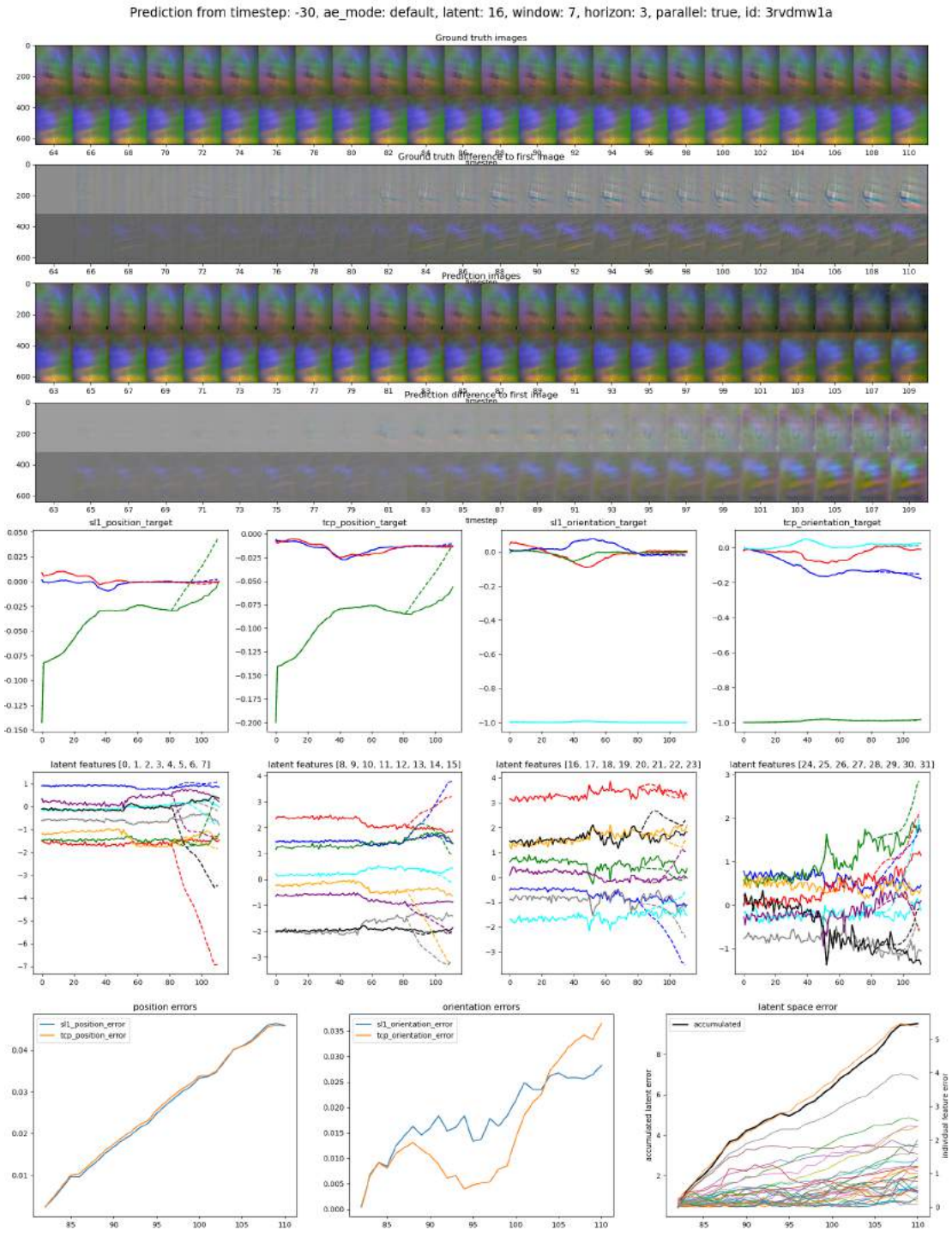
Figure A.7.: Losses logged during the training of a transition model on the fingerprint dataset. All losses converge except the image reconstruction. The additional information of the other sensors could not be utilized to forward predict images well.



(a) Predicting last 12 time steps of an episode.

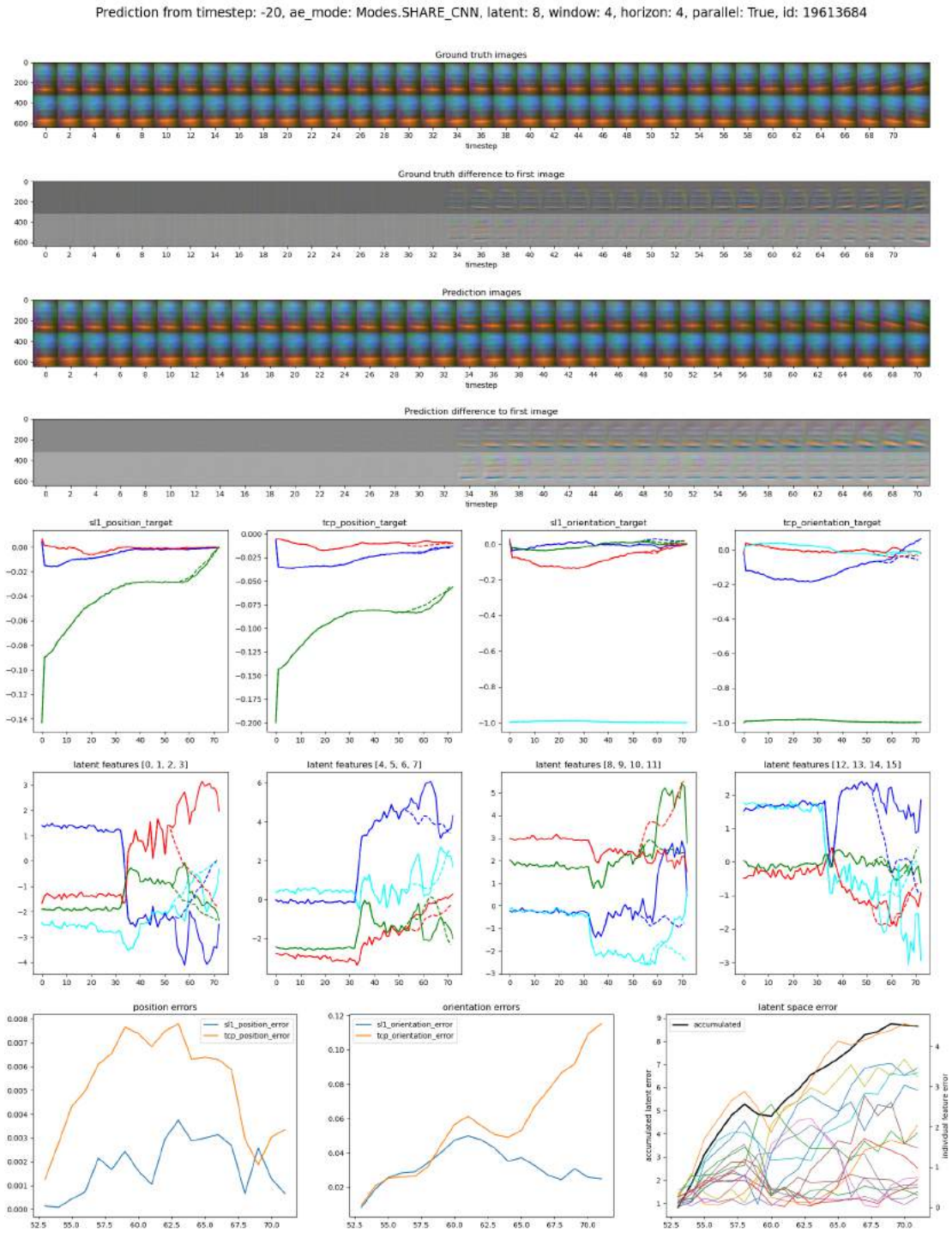


(b) Predicting last 20 time steps of an episode.

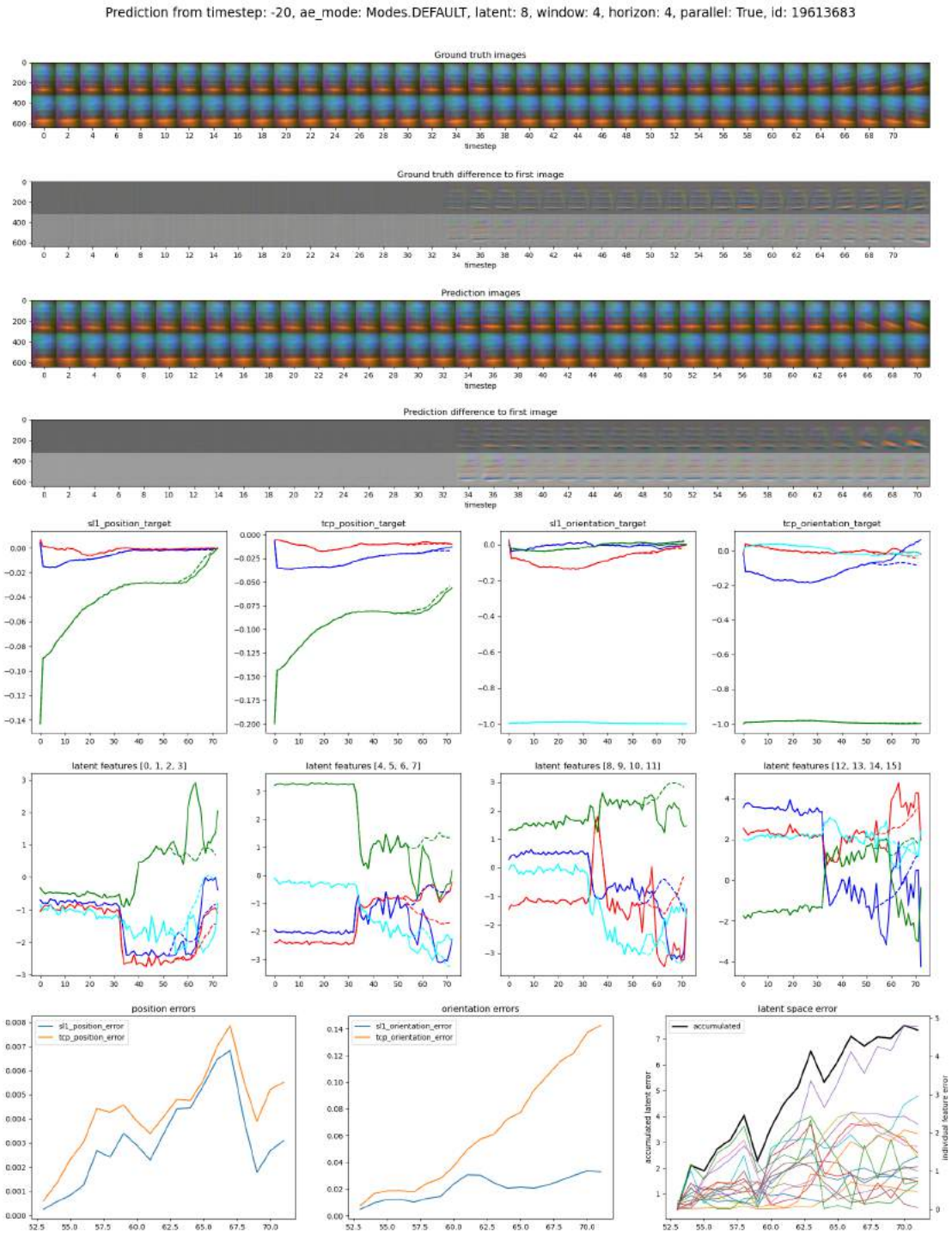


(c) Predicting last 30 time steps of an episode.

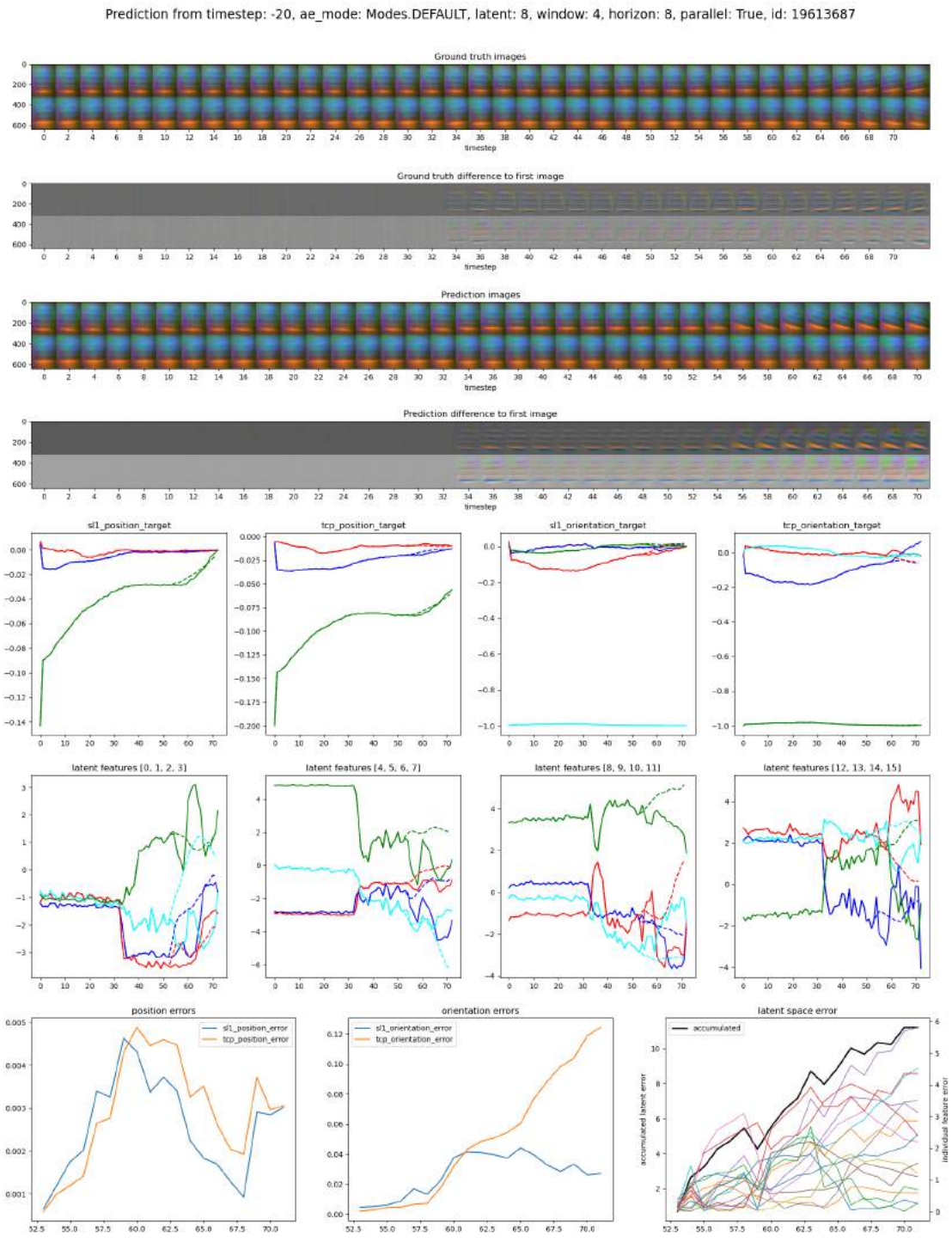
Figure A.8.: Forward prediction on test data with fingerprint sensor images. The 3rd and 4th row of images show the images decoded from latent space. For the last n time steps of the episode, the latent space is forward predicted by the transition model. While the predictions on the poses are just linear extrapolations and not very good, the prediction in the latent space is okay, when compared to the smooth dataset models. This should not be attributed to a better transition model, but to the more well-formed latent space of the fingerprint images.



(a) best in sl1 pose prediction

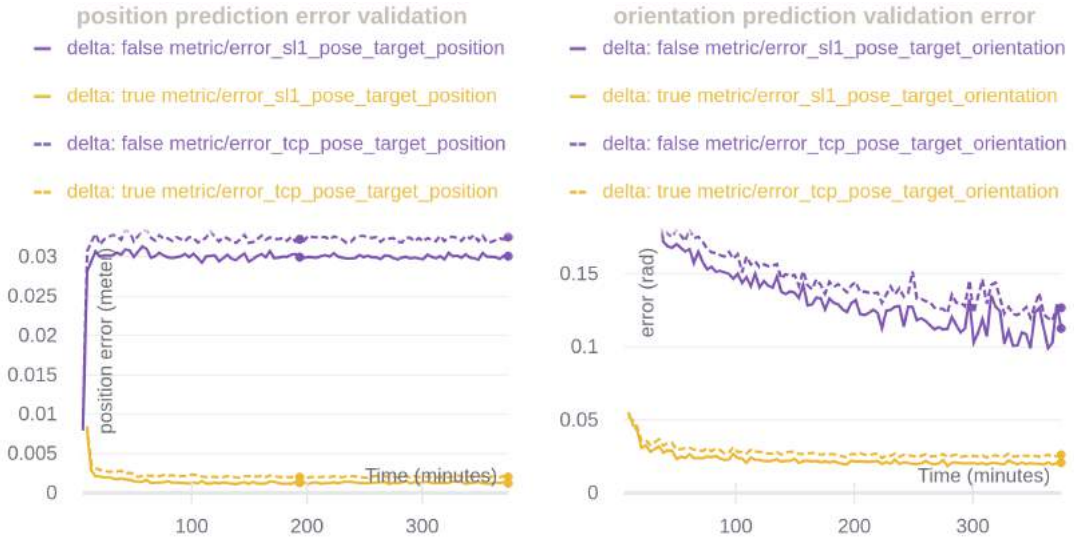


(b) best in tcp pose prediction



(c) best in latent space prediction

Figure A.9.: Visualization of predictions of best models on an episode from test data. DIGIT images are visualized as is and as difference to the first image of episode.



(a) position error

(b) orientation error

Figure A.10.: Comparison of training with delta predictions and training with absolute predictions. Position and orientation error on validation data computed with one pass forward prediction. The model prediction "horizon" steps ahead for each time step but does not predict recursively on its own output. Tactile sensor images were ignored in this experiment.