Diffusion Policy for Collision Avoidance in a Two-Arm Robot Setup

Master Thesis

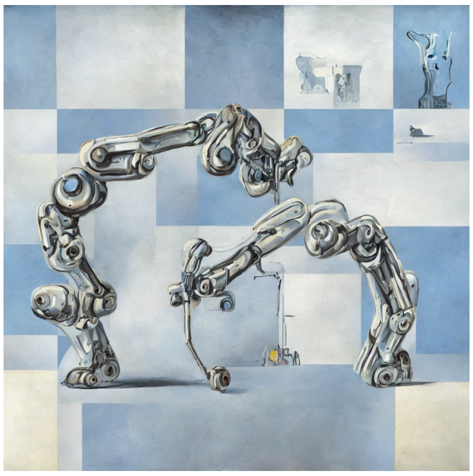
Author(s):
Spieler, Jonas

Publication date:
2024

Permanent link:
https://doi.org/10.3929/ethz-b-000675013

Rights / license:
In Copyright - Non-Commercial Use Permitted

Diffusion Policy for Collision Avoidance in a Two-Arm Robot Setup



Jonas Spieler

Master's Thesis

Computational Robotics Lab

ETH Zürich

Supervisors:
Miguel Zamora
Prof. Dr. Stelian Coros

May 25, 2024

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computational Robotics Lab

Abstract

Diffusion models in robotics have shown great potential with a wide range of applicability. Especially their capability to model multi-modal data distribution has many benefits when learning a task such as collision-free trajectory optimisation with dynamic moving objects. Collision avoidance is a central problem in robotics where classical approaches still suffer from non-optimal solutions or high computational costs.

In this work, we present four diffusion models, capable of predicting collision-free trajectories in a pick and place setup of two moving robot arms. The best model achieves a success rate of 88.1% in producing collision-free trajectories, while the worst one succeeds in 76.2% of the episodes. Furthermore, we analyse the models in terms of their accuracy in reaching the target pose, their capability of predicting smooth trajectories, and their success rate in generating collision-free trajectories.

Contents

Introduction — Chapter 1

Robotics is a strongly growing field with many opportunities to further improve the state of the art with machine learning methods. Trajectory optimisation is a core element for all sorts of robots. Solving the constrained version of collision-free trajectory planning is still an open problem despite the many promising approaches. Classical methods tend to suffer from finding only local optima, being computationally too complex to run in real time, or failing to scale up to many robots. Learned trajectory planning has the benefit of leveraging task-specific data to optimise the model for these requirements.

The goal of this work is to test the adaptability of diffusion models to generate collision-free trajectories on a pick and place task that involves two robot arms. The efficiency of diffusion models to learn from few expert samples is demonstrated in [1], which is a promising path for sample-efficient learning. Furthermore we want to assess the accuracy of diffusion models in reaching the target pose along with their capability to generate low jerk trajectories like the underlying training data.

While many learning-based approaches have been applied to collision-free trajectory planning, there is still room for improvements. Narrowing the gap and getting closer to optimal collision-avoidant trajectory planning is a central aspect in current machine learning research. The low availability of high quality data in the domain further hinders progress in the field. However this also gives opportunities for innovative ways to sample data and to train models with sparse data supply.

We introduce our collision avoidance diffusion policy, consisting of four models, two operating in joint-space and two in Cartesian-space. The models are conditioned on their trajectory history that includes among others the collision spheres of the two robots. Furthermore we present our dataset that contains jerk- and acceleration-minimised trajectories. Finally, we simulate the robot on a held-out evaluation dataset for the same task. We assess its performance regarding the accuracy of reaching to the target pose, the capability to produce jerk- and acceleration-minimised trajectories and on its ability to avoid collisions.

Related Work   Chapter 2

2.1 Motion Planning and Trajectory Optimisation

In robotics the task of planning motion and generating trajectories is a well-studied topic with broadly two categories where the approaches fall into: classical, non-parameterised methods and machine learning-based, parameterised models [2].

Classical algorithms include for example sampling-based methods like RRT [3] and PRM [4] along with their improved versions RRT* and PRM* [5]. Other approaches include search-based algorithms that utilise a given graph to find the shortest paths like A* [6] and Dijkstra's algorithm [7]. However these methods only provide path planning which still needs further refinement to get a smooth and executable trajectory.

The second category consists of learned methods that optimise a parameterised model such as a deep neural network. Examples are among others imitation learning (IL) and inverse reinforcement learning (IRL) [8]. Imitation learning covers approaches such as behavior cloning (BC) [9], that utilises supervised expert data to learn the underlying data distribution. Other learning methods comprise of Reinforcement Learning algorithms that learn policies through indirect feedback signals from reward functions [2].

In this paper we focus on a behavior cloning approach for trajectory optimisation with collision avoidance based on a limited amount of expert data.

2.1.1 Learned Collision-Avoidant Trajectory Planning

The Motion Policy Network from [10] shows that it is possible to learn dynamic obstacle avoidance given the current robot state and a point cloud of the environment. Another work from [11] presents a decentralized multi-arm motion planner, trained in a Multi-Agent Reinforcement Learning setup. It shows the capability of a learned motion planning system to predict collision-free trajectories for a setup with up to 10 robot arms. However both of the two approaches need large quantities of data in order to learn collision-free trajectory planning.

2.1.2 Diffusion Models in Robotics

With the introduction of Denoising Diffusion Probabilistic Models (DDPM) in [12], a new class of latent variable models has been shown to produce high-quality images.

In [1] the strengths and adaptability of diffusion models for robotics tasks were shown. Unlike in other machine learning domains, in robotics, data is generally not available in large quantities, and efficient learning methods are required. The paper shows that diffusion models are applicable to a wide range of tasks allowing for effective learning of multi-modal data distributions.

The closest work to ours is [13], where the capability of diffusion models to generate collision-free trajectories was shown and also compared against classical sampling-based trajectory optimisation methods such as RRT-connect [14] and conditional Variational AutoEncoders [15]. However this method only uses diffusion models for collision avoidance in a static environment with one robot actuator.

Method — Chapter 3

Our setup involves a 7 DoF robot arm that performs a pick and place task with the additional constraint of dynamic collision avoidance. The dataset consists of jerk- and acceleration-minimised trajectories computed with the cuRobo library [16] and is sampled in two steps. First we collected the data for the obstacles and then we generated trajectories for the actuated robot. The obstacles consist of collision sphere recordings from the same type of robot arm. Along with our dataset we highlight the architecture of the diffusion policy models and then briefly introduce the training setup for diffusion models.

Our hypothesis follows the approach from Chi et al. [1] that utilises expert data to train a diffusion policy. The goal is to learn to imitate the behavior of the data, similar to the original behavior cloning paper from [9]. The difference is that we have a continuous action space and our model has a probabilistic nature and refines the action in an iterative way.

3.1 Dataset Sampling

Similar to [13] we created a synthetic dataset for our diffusion models. However we took use of cuRobo [16] and Isaac Sim [17] to record the data as opposed to the sampling-based approach with RRT-connect. CuRobo already provides jerk- and acceleration-minimised trajectories, which removes the work needed to post process the data to get smooth trajectories.

In [13] they focus on trajectory optimisation with static collision avoidance. Our approach focuses on dynamic collision avoidance where the objects come from the collision mesh from another robot arm.

The dataset consists of continuous position-control actions recorded from the Emika Franka Panda robot arm [18]. Using position-control trajectories also follows the suggestion from [1], which claims that diffusion policies based on position-control outperform velocity-control action space models. As collision mesh for the Panda robot we utilised the set of 61 collision spheres that came from cuRobo, which is visualised in figure 3.2 on the right.

The cuRobo library allows for dynamic and static collision checking. For our setup the dynamic version is more interesting but the problem with it is that it only works in a model predictive control fashion. This does not guarantee convergence in a fixed amount of time steps and has the possibility of getting stuck in a local minima. This can lead to non-optimal trajectories with slow progress toward the target.

In order to avoid adding further convergence issues when leveraging the trajectory optimisation to a neural network prediction, we decided to use the static collision checking from cuRobo. This ensures convergence in a fixed amount of time steps but on the other hand introduces the potential that the sampled trajectory is not collision-free. CuRobo only avoids a collision with the initial positions of all the obstacles and therefore collisions can happen when the obstacles move in the meantime. To solve this issue, we sampled trajectories and only kept them if they were collision-free.

3.1.1 Pick and Place Dataset Generation

We first sampled the trajectories of one robot alone performing pick and place trajectories. The dataset is called *Ghost Robot Dataset* and it served as the dynamic collision objects for the second dataset. We will also call the robot from this first dataset the ghost robot, since it is just a recording of the collision spheres and its trajectories are not directly used to train the policies. In a second step we sampled the final dataset called *Robot Collision Avoidance Dataset*.

Ghost Robot Dataset

The setup in the simulation is designed such that the Panda robot is located at the origin $(0, 0, 0)$ and oriented with the identity $\mathbb{I} \in \mathbb{R}^3$ rotation matrix. In order to avoid learning all possible trajectory mappings and to facilitate the learning, we focused on sampling targets within a predefined area for the pick and place locations.

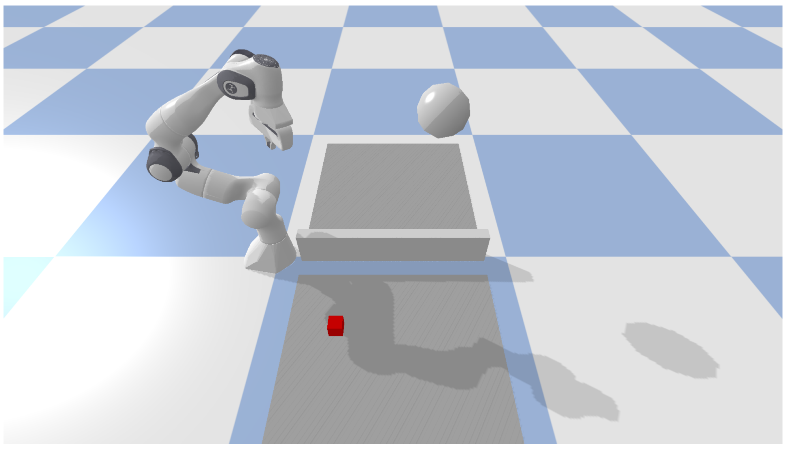


Figure 3.1: This is the setup from the recording of the ghost robot. A sphere and a bar represent the obstacles that help in generating a more diverse set of trajectories. The sphere position is drawn uniformly at random in the area above the bar for each new episode.

To get a diverse set of trajectories that avoid moving always in the most direct way to the target, we introduced two static objects for this setup. The first is a small bar between the pick and place field and the second is a sphere that was randomly placed in the area above the bar. This resulted in trajectories generated by cuRobo that were more diverse regarding to which paths the robot took to get to the targets. Figure 3.1 shows the setup marking the pick and place area along with the two collision objects in light grey.

Specifications The *Ghost Robot Dataset* consists of a total of $2'000$ different episodes. Each episode has three different phases: a pick, a place and a retract phase. The robot always starts from the same initial pose and first performs a pick task that consists of moving the end-effector to the pick target location. Then the robot moves from the pick target location to the place target location and finally moves back to the retract end-effector pose. Each phase has 34 trajectory steps, computed by cuRobo. This makes a total of $2'000 \cdot 3 \cdot 34 = 204'000$ data points.

Each row represents one step and contains 282 entries. Table 3.1 shows which columns correspond to which data entries. A pose is always represented with the three Cartesian coordinates x, y, z and a quaternion as w, i, j, k.

Column Index	Data Type
0	episode $\in \{0, ..., 1999\}$
1	step $\in \{0, ..., 101\}$
2:9	target pose (x, y, z, w, i, j, k)
9:16	end-effector pose (x, y, z, w, i, j, k)
16:25	Panda joint pose (radians)
25:269	Panda collision spheres $(x, y, z, radius)$
269:273	pick area $([x_{min}, x_{max}], [y_{min}, y_{max}])$
273:277	place area $([x_{min}, x_{max}], [y_{min}, y_{max}])$
277	phase $\in \{1, 2, 3\}$
278:282	collision sphere $\in \{x, y, z, radius\}$

Table 3.1: Ghost robot dataset entry structure, index range: (incl:excl)

Robot Collision Avoidance Dataset

The next step was to sample trajectories in the same pick and place setup as before, but now we replaced the two collision objects (the bar and the sphere) with the prerecorded collision spheres of the Panda robot from the first dataset. We flipped the positions of the recorded robot spheres along the x-axis and added an offset of one meter in the x direction. This way we placed it opposite to the new robot for which we compute the collision-free trajectories. Furthermore, we sampled some random offset in the time step for the ghost robot to ensure that the two robots are not synced in their execution of the trajectories.

Figure 3.2 shows the setup during an evaluation episode of the diffusion models. Only the collision spheres of the ghost Panda are shown in this figure. For this setup we reduced the pick and place areas in order to have more sample points per area and to minimise the locations that are not reachable for the robot. This should help the neural network to focus on just a small set of target locations.

As mentioned above, cuRobo only computes collision-free trajectories with static collision checking at the beginning of the computation. This gives the risk that a collision can happen when the obstacles move from their initial positions. To mitigate this problem, we first tried to resample the trajectory multiple times during the execution to avoid a collision with the new state of the ghost robot. This however turned out to be infeasible to sample any data point. The possibility that cuRobo fails to converge to a solution in each resampling stage increases the more often a resampling is performed. Since we already have three possible convergence failure one for each of the pick, place and retract phase, this would only further increase the failure rate of sampling an episode. The reason cuRobo can fail to converge is that an object might not be reachable in a given state of the collision objects or due the sampling-based nature of the motion generation algorithm that did not converge.

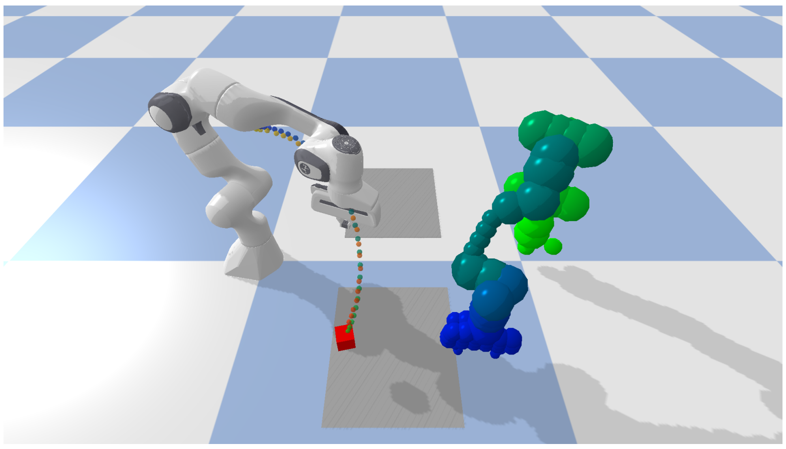


Figure 3.2: On the left is the robot actuated by the neural network and on the right is the recording of the collision spheres of the Panda robot. The blue/green dots indicate the ground truth end-effector trajectory and the red/yellow ones represent the predicted trajectory. The grey areas mark the pick and place fields.

However, by resampling the trajectory on the way, we would also loose the smoothness property since cuRobo's static collision avoidance optimisation does not take into account past trajectory steps.

Therefore we only sampled a trajectory once at the beginning of a phase with static collision avoidance and then tested if it contained a collision. This also ensured that the full trajectory of a phase has low jerk and acceleration values.

3.1.2 Specifications

The final dataset consists of 501 episodes with a total of 51'102 rows and 522 columns. Again each episode contains 102 steps, 34 steps for each of the three phases. Table 3.2 shows the data types of each of the 522 columns.

3.2 Multi-Arm Robot Systems

In this section we describe our training pipeline of the diffusion models, which includes the input and output shape of the models, the used neural network architectures and also a brief overview on diffusion models.

Column Index	Data Type
0	episode $\in \{0, ..., 500\}$
1	step $\in \{0, ..., 33\}$
2:9	target pose (x, y, z, w, i, j, k)
9:16	end-effector pose (x, y, z, w, i, j, k)
16:25	Panda joint pose (radians)
25:269	Panda collision spheres $(x, y, z, radius)$
269:273	pick area $([x_{min}, x_{max}], [y_{min}, y_{max}])$
273:277	place area $([x_{min}, x_{max}], [y_{min}, y_{max}])$
277:521	ghost robot collision spheres $(x, y, z, radius)$
521	phase $\in \{1, 2, 3\}$

Table 3.2: collision avoidance dataset entry structure, index range: (incl:excl)

Like in the Diffusion Policy paper from [1] and the Motion Planning Diffusion paper [13] we train the neural networks as diffusion models. This way the model does not predict the output directly but it predicts the noise gradient used to iteratively refine the output, starting from Gaussian noise. The pipeline also follows closely the approach from [1] with just some small adaptations for our use case. We used the code of the neural network implementations and the training scripts from [1] as basis for our four diffusion models. In the next section we briefly highlight the changes made to the setup.

3.2.1 Model Input and Output

The setup from [1] is designed in a model-predictive control (MPC) style, with two sequences defining the input to the neural networks.

The first sequence is the observation horizon, which is the conditioning parameter that contains all the observations the model gets from its environment. This consists of a total of n observations from a past time step $t - n + 1$ up to the current time step t. The second is the prediction horizon, which consists of m trajectory actions. These can range from a past time step up to a number of time steps into the future. Furthermore, an action sequence is defined, which specifies the number of executed action steps from the predicted ones until a replanning is performed. Table 3.3 shows an example with an observation horizon of four steps, a prediction horizon of eight steps and an action horizon of two steps.

This setup is designed such that new observations are appended to the list of observations and the oldest observation gets discarded, like a moving window of n observations.

During the experimental phase, we found that the tested diffusion models suffered from achieving sub-centimeter level accuracy in reaching to the target pose. An open horizon from the MPC style models introduces the need to define a distance threshold that indicates an end of the trajectory.

Time step	$t-3$	$t-2$	$t-1$	t	$t+1$	$t+2$	$t+3$	$t+4$
Observation	o_{t-3}	o_{t-2}	o_{t-1}	o_t				
Action				a_t	a_{t+1}			
Prediction	p_{t-3}	p_{t-2}	p_{t-1}	p_t	p_{t+1}	p_{t+2}	p_{t+3}	p_{t+4}

Table 3.3: An example of a MPC style model input with an observation horizon consisting of the last 4 steps and an action horizon of 2 steps. The prediction horizon is 8 steps, with 3 of them consisting of past actions.

In order to avoid convergence issues coming from the low accuracy, we fixed the total number of time steps the model can use to 34. This number is chosen to match the steps provided by the cuRobo library. The model gets the full observation over the trajectory history from the initial pose until the end pose.

At time step t observations from timstep $0, ..., t$ consist of the executed joint poses along with all world state vectors. All future observations with $t_i > t$ are zero vectors, except for the entries of the end-effector target pose which are known from the beginning. Finally the model always predicts the whole trajectory towards its target, regardless of the number of time steps left to execute.

In this setup we do not move the window but only increment the index of the action step to take. This guarantees a finite amount of steps until the trajectory is executed.

The action horizon is chosen to be 8 steps, which follows [1].

Observation Dimension

Our observation serves as conditional input and consists of a sequence of 34 observations. Each observation is a vector in $o_t \in \mathbb{R}^{509}$. The 509 dimensions come from stacking all observations of the state from time step t together: joint pose (7 dimensions), end-effector position and rotation (7 dimensions), target position and rotation (7 dimensions), ghost robot collision spheres (244 dimensions), and panda robot collision spheres (244 dimensions). Observations larger than the current time step are all set to zero, except for the entries that represent the target position and orientation.

Action Dimension

As action domain we used two different model types. The first model predicts the end-effector pose in Cartesian coordinates, and the second model directly predicts joint positions for each degree of freedom of the robot.

Cartesian-Space Model This model predicts the Cartesian position and orientation directly, but needs an inverse kinematics solver to compute the next joint state.

Solving inverse kinematics with seven degrees freedom is not a uniquely determined problem. This means that an infinite number of poses can satisfy a given end-effector position and orientation. In order to get collision-free trajectories we need to know and control the positions of all of the robot joints in order to avoid collisions coming from any of the robot links.

Our assumption is that given a starting pose, the predicted update is only minimally different such that the inverse kinematics solver finds a feasible solution that is close to the first pose and therefore giving the model the ability to control the joint positions by changing the end-effector pose in a way to avoid collisions with any of the robots links.

An end-effector pose is represented as a Cartesian coordinate in $\mathbb{R}^3$ combined with the end-effector orientation as a unit quaternion. This results in an action space of one trajectory step $a_t \in \mathbb{R}^7$. Since the model produces a sequence of 34 trajectory steps the output of the model is in $\mathbb{R}^{34\times 7}$

Joint-Space Model In order to avoid the problem with the Cartesian model, we also tested directly predicting the joint angles. This way the model learns the inverse kinematic mapping from joint angles to the end-effector and directly controls the state of the robot.

This has the advantage that the neural network can control the robot state explicitly in order to avoid joint poses that may lead to collisions. Another problem that can be mitigated is when the inverse kinematics solver does not find a feasible solution or when a kinematic singularity locks two or more joints. This can lead to problems in executing the trajectories.

For this model type we also get an action space $a_t \in \mathbb{R}^7$ for each step. This comes from the 7 degrees of freedom from the Panda robot, excluding the fingers. The whole model output is therefore also in $\mathbb{R}^{34\times 7}$, consisting of 34 joint poses that represent the full trajectory.

Quaternion Convention

A problem with learning quaternion rotations by gradient descent, presented in [19], is double cover. This means that a given rotation in SO(3) can be represented by the two quaternions q and $-q$. This leads to discontinuities in the learned mapping from the neural network. However we can not utilise the proposed $\mathbb{R}^9$ + SVD solution due to the indirect nature of learning diffusion models. We would need to have access to the final predicted rotation in order to compute the loss on the SVD-projection of the rotation.

To avoid learning the double cover of quaternions, we restricted the space of orientations by manually mapping all quaternions to the half-space where the w component is positive.

3.2.2 Model Architecture

In the following, we will closely highlight the two different types of neural network architectures that were used for training the diffusion models. The first is an adaptation of the U-Net and the second is a transformer based model.

U-Net

Introduced in [20] the U-Net was first used for medical image segmentation. Over the years the architecture got adapted to include many newer neural network features, among others is a conditioned U-Net, presented in [21]. Here an added feature-wise linear modulation layer (FiLM) from [22] is used to condition the input of the neural network. The idea behind the conditional U-Net was to guide the encoding process by introducing a learnable affine transform that gets applied in each down-sampling layer. Finally a universal decoder is used to recreate the conditioned latent representation. The conditioning consists of two learnable matrices $\gamma(z)$ and $\beta(z)$ which are applied to our propagated input variable x as

$$FiLM(x) = \gamma(z) \cdot x + \beta(z) \tag{3.1}$$

where z is our conditioning variable.

In the Diffusion Policy paper [1] they used FiLM in the encoder and decoder step of the U-Net to guide the whole denoising process. Our model takes a sequence as input and generates a sequence of outputs, therefore we need to flatten the sequence into a vector before passing it to the conditioning layers.

All together we used three down sampling layers which gives a total of 472 M parameters for the U-Net.

Transformer

In [23] the transformer architecture was presented that led to large improvements in the natural language processing field. The second type of model that was used for the diffusion models in [1], is a transformer-based decoder combined with a dense multi-layer perceptron as an encoder. The basic building block of the transformer is the attention mechanism that is computed by equation 3.2

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.2}$$

It uses to three different values extracted from the input sequence: the query matrix Q, the key matrix K and the value matrix V. The softmax together with the cross-correlation term QK^T computes a distribution specifying which entries attend to each other. The value matrix is then used to extract the values corresponding to the attended entries.

The encoder from the diffusion transformer extracts information from the sequence of observations. The decoder is then responsible for predicting the noise level in the current diffusion step conditioned on the encoded observation sequence. This is done by 12 stacked modules of the decoder block. One decoder block first performs self-attention on the noise level of the action sequence and then further propagates the input to 12 parallel cross-attention blocks. In the end the predicted noise sequence is fed through two dense layers.

The resulting transformer architecture is with 119 M parameters much smaller compared to the U-Net.

3.2.3 Training

In this section we first give an overview of how training a diffusion model works and then we list our most important key parameters for the training of the models.

Denoising Diffusion Probabilistic Models (DDPM)

Diffusion models, introduced in [12], learn to remove noise from the input sequence in an iterative way. First, we need to sample an input tensor from a Gaussian normal distribution $x_T \sim \mathcal{N}(0, \mathbb{I}) \in \mathbb{R}^{34\times7}$, then use the neural network to predict the noise level of the current step and remove it from the input. By doing this in an iterative way, we get our diffusion Markov chain. In the denoising-process, the chain starts at the sampled noise x_T and ends at x_0. The process uses a fixed noise variance schedule $\beta_1, ..., \beta_T$. For our task we used a cosine schedule proposed in [24], which also follows the approach from the Diffusion Policy paper [1]. The reason for a cosine schedule is to avoid weak learning signals coming from the first quarter of a linear diffusion schedule.

An important feature of the forward diffusion process is that it can be computed in closed-form given by the formula in 3.3. During training we can now sample a time step uniform at random as $t \sim \mathcal{U}(0, T-1) \in \mathbb{N}$ and add Gaussian noise $\epsilon \sim \mathcal{N}(0, \mathbb{I}) \in \mathbb{R}^{34\times7}$ according to equation 3.3.

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon \tag{3.3}$$

where x_0 is our data point and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ with $\alpha_t = 1 - \beta_t$ coming from the predefined variance schedule $\beta_1, ..., \beta_T$.

Finally we get a gradient update step as defined in equation 3.4 for training the diffusion model

$$\nabla_\theta \left\| \epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t \right) \right\|^2 \tag{3.4}$$

where ϵ_θ is our neural network that learns to predict the noise level in each diffusion step. During inference we sample Gaussian noise and perform a full reverse diffusion process by starting at time step T and iteratively remove noise until we reach x_0.

Training

In total we trained four models, the U-Net once as a Cartesian-space model and once as joint-space model and the same two types with the transformer architecture. In order to improve training stability we normalised the data input to the range $[-1, 1]$ and denormalised the output again from $[-1, 1]$.

To measure the performance of the models on unseen data, we split the 501 episodes from the dataset into 80% training data (400 episodes) and 20% evaluation data (101 episodes). The models were all trained for 1'500 epochs with the loss plots visualised in figure 4.1. An evaluation step was performed every 20 epochs. In each evaluation phase the same MSE loss as in the training loop was computed and in addition to that, we also performed a full reverse diffusion process to evaluate the quality of the predicted trajectories.

Inference

During training we used a β schedule of 100 steps, but in order to increase performance at inference time, we reduced the steps to 10. This also follows the approach in [1]. We found that the performance did not change when evaluating with 100 or 10 diffusion steps.

All together the U-Net takes with 10 diffusion steps $\approx 0.099s$ on an Nvidia RTX A4000, while the transformer is more efficient with $\approx 0.07s$. This is the average value of 100 samples from a Pyhton script that uses PyTorch on the GPU. It includes normalisation of the data, the sampling of the noise, the 10 denoising steps and the denormalisation of the predicted action.

Results — Chapter 4

To simplify the setup for the evaluation process, we took use of PyBullet [25] due to its plug-and-play nature and faster startup time compared to Isaac Sim. We will briefly touch upon the graphs from the training and evaluation steps of our four models. Then we will discuss and analyse the results of the simulated evaluations. Finally we highlight a potential workaround for the problem with the smoothness during recomputation.

4.1 Evaluation on the Dataset

The first evaluation of the diffusion policies consists of measuring their performance on unseen data. Figure 4.1 shows on the right the evaluation loss. Interpreting this on itself could lead to the conclusion that the model does not perform well since the loss is more or less constant for the joint-space models and it is increasing for the Cartesian-space models. However, if we also consider the metrics of the generated trajectories from the full reverse diffusion process shown in figure 4.3 and 4.2, we see that the models learn and make progress over the train time. For the evaluation metrics of the full reverse diffusion process, we only calculated the model's accuracy in reaching the target position and its capability to predict smooth trajectories.

On the left of figure 4.3 is the mean distance between the end-effector and the current target visualised and on the right the same but with the mean angular distance. In order to evaluate this metrics for the joint-space model, we performed forward kinematics on the last predicted joint angles of the trajectories.

The smoothness plot in 4.2 shows that the models' ability to predict smooth trajectories increases over training time. This shows that the models learn to produce data similar to the underlying training data that originates from cuRobo.

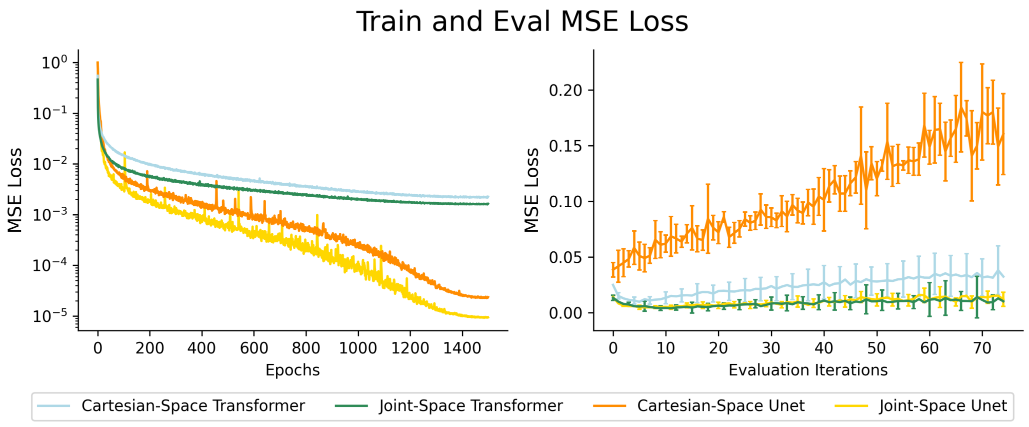


Figure 4.1: On the left is the train mean squared error and on the right is the evaluation mean squared error of all four models.

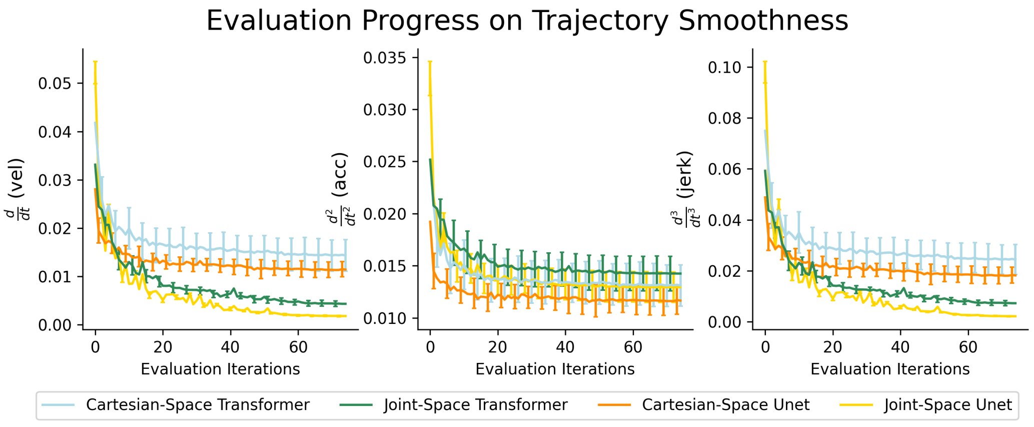


Figure 4.2: This plot shows the progress on the trajectory smoothness of all four models during train time. The smoothness values are computed by taking the difference of two consecutive values: $a_t - a_{t-1}$. For the Cartesian models the values are calculated on the end-effector trajectory and for the joint-space models the value is calculated on the joint angles.

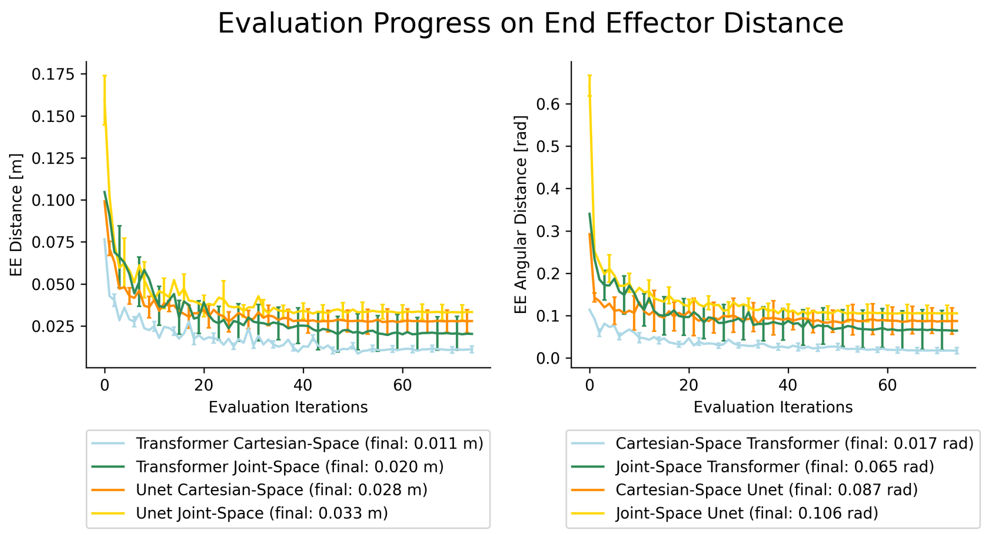


Figure 4.3: This plot shows the progress on the final end-effector distance (angular distance) of all four models during train time.

We have to note that we can not apply normal metrics to the smoothness values in figure 4.2. Normally the values would represent velocity, acceleration and jerk but we do not have a time step duration and these values are averaged over the total predicted trajectory. Another important point is that the smoothness of the Caretsian-space models were not evaluated in the same way as the joint-space models since they predict different types of trajectories. The smoothness metrics are calculated as the mean of the difference of two consecutive trajectory points. This does not guarantee jerk-minimised trajectories for the Cartesian models, however it is a good indicator.

The downsides of these evaluations is that they do not simulate the predicted actions. The model gets the ground truth observation and predicts the trajectory based on clean data. By simulating the predictions, inaccuracies accumulate, which in turn can lead to reduced model performance. This effect is also known as covariate shift in imitation learning [26], where the probability distribution changes from the train data points $x \sim P_{train}$ to the test data points $x \sim P_{test}$.

The next evaluation therefore consists of a simulated pass through the evaluation data.

4.2 Evaluation using Simulation

In order to simulate the predictions we used PyBullet [25]. The setup is visualised in 3.2, where we have the actuated Panda robot at the origin and the ghost robot is presented by its collision spheres.

The final trajectories are in joint-space, either directly predicted from the joint-space models or indirectly from the Cartesian-space models. Therefore, we need a position controller to execute the trajectory. We also need a time step duration to specify how long the robot can take to execute one step of the trajectory. We could theoretically recompute the trajectory in every step of our trajectory and therefore the simulation time in between two consecutive trajectory steps should be large enough to allow for a recomputation. As stated above, we have $\approx 0.099s$ prediction time for the U-Net and $\approx 0.07s$ for the transformer. A trajectory step is further divided into multiple steps for the motor controller to fully converge to the new position of the robot arm. Therefore, we set the time step duration of PyBullet to $0.099/20 = 0.00495s$ for the U-Net and to $0.07/20 = 0.0035s$ for the transformer. This way we perform 20 simulation steps for each trajectory step.

For this evaluation we have three important metrics that specify the performance of the models: the accuracy in reaching the target positions and orientations, the smoothness of the trajectories and the number of collisions occurred in the evaluation process.

4.2.1 Accuracy

Table 4.1 shows the accuracy of all four models. It is clearly visible that the transformer models outperform the U-Net models in this evaluation with the best average accuracy of 1.5cm and 2.22° on all three phases coming from the Cartesian-space transformer. The log-distance plot of this model is shown in figure 4.4. It visualises the end-effector distance and the angluar distance to the target over the progress of the episode on the evaluation dataset.

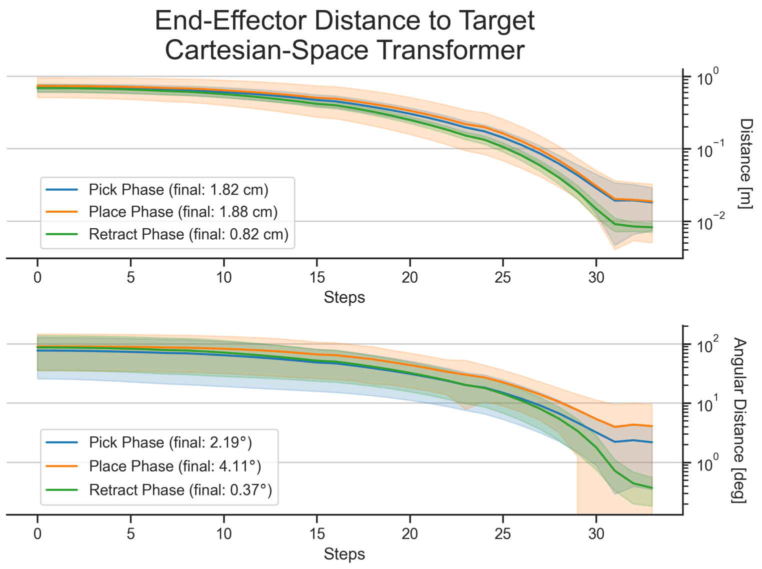


Figure 4.4: This graph shows the log distance (linear and angular) between the end-effector and its target for the Cartesian-space transformer model.

Furthermore, it is visible from table 4.1 that the retract phase has the highest accuracy, where all four models are in the sub-centimeter range and also have a low angular error.

Model	Pick	Place	Retract	Avg.
Joint-Space U-Net	3.4cm (3.28°)	6.4cm (9.31°)	0.5cm (0.97°)	3.4cm (4.52°)
Cartesian-Space U-Net	4.1cm (3.45°)	6.3cm (9.86°)	0.8cm (0.38°)	3.7cm (4.56°)
Joint-Space Transformer	2.2cm (2.58°)	3.9cm (10.18°)	0.5cm (1.43°)	2.2cm (4.73°)
Cartesian-Space Transformer	1.8cm (2.19°)	1.9cm (4.11°)	0.8cm (0.37°)	1.5cm (2.22°)

Table 4.1: This table shows the accuracy as distance (angular distance) to the target of each model on the last step of the trajectory.

Figure 4.5 shows a visual version of the evaluation accuracy of all four models. The yellow to red colored points indicate the position of the targets on the pick and place areas. The closer the point is to the color yellow, the higher is the accuracy of the model on this evaluation data point. The green ticks represent the final z-axis orientation of end-effector and the blue ticks are the target z-axis orientation. The white dots represent the 400 training data points. It is visible that the transformer models have a more uniform distribution of the accuracy on both pick and place areas compared to the U-Net models.

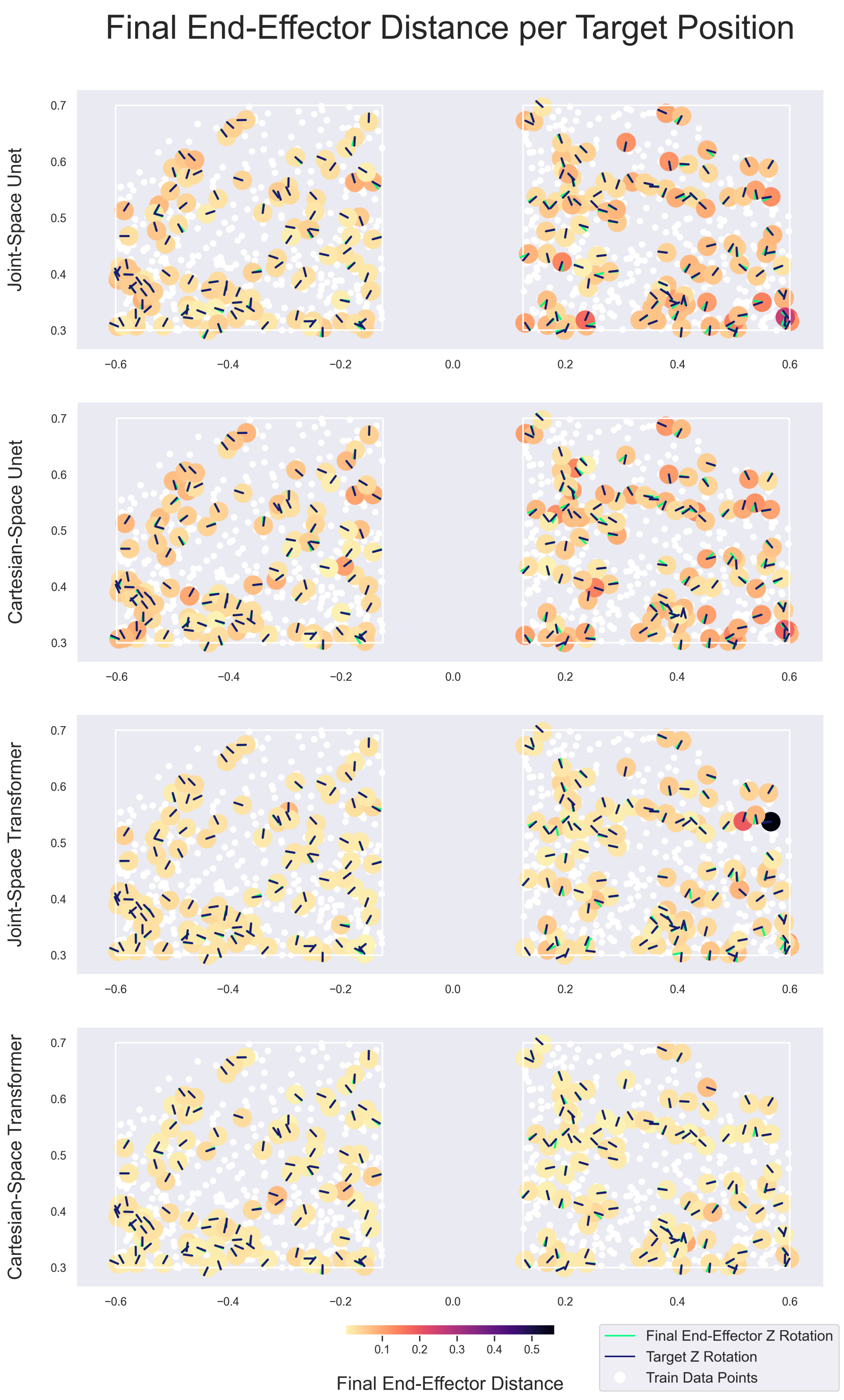


Figure 4.5: On the left is the pick area and on the right the place area. The colored points indicate the position of the targets in the evaluation data. A more purple/black point symbolises a larger distance error on that data point. The white points indicate the positions of the training data.

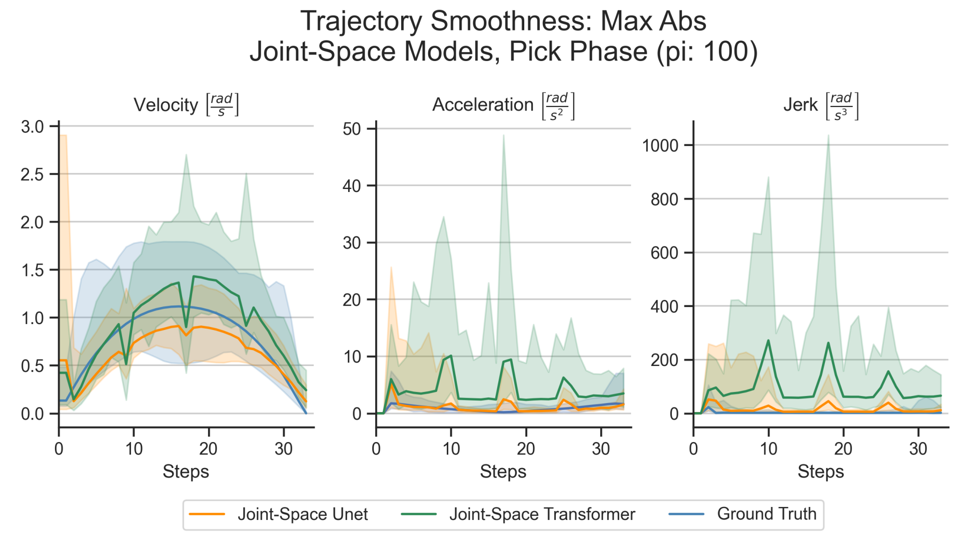


Figure 4.6: This plot shows the maximum absolute value of the velocity, acceleration and jerk from the 7 robot joints. The trajectories are from the two joint-space models during the pick phase. The lines represent the mean value over all evaluation episodes and the colored area represents all of the data points from the episodes (100 percentile).

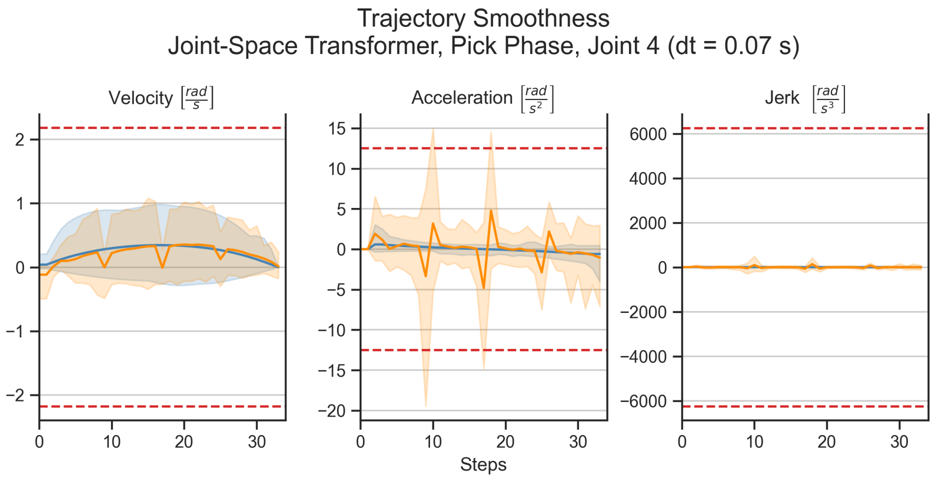


Figure 4.7: This plot shows the velocity, acceleration and jerk values of the Panda joint 4 during the pick phase from the joint-space transformer. The graph represents the mean value over all evaluation episodes and the area shows the complete range of the evaluation data (100 percentile). The red lines mark the limits of the joint.

4.2.2 Smoothness

The next metric to consider is the smoothness of the simulated trajectories. Here the picture changes such that the U-Net is generally better capable of producing smooth trajectories. This effect can be seen in the plot 4.6. The graphs show the per-step maximum absolute values across all joints and evaluation episodes. The orange line represents the U-Net trajectory, which is much closer to the blue cuRobo trajectory compared to the green line representing the transformer trajectory. Here we only show the pick phase of the joint-space models for better visibility. The plots for the other two phases look similar and are therefore omitted here.

The values are in a good range since the mean is most of the time close to the blue low-jerk trajectory line. For a more precise picture, we have visualised all smoothness values for each joint with its specific limits. An example can be seen in figure 4.7, which shows joint 4 of the Panda robot from the joint-space transformer. The mean value over all episodes is always below the red limit marks and only a few times the values exceed the limits. All figures for all joints, phases and models can be seen in the appendix in chapter A.3.

However, figure 4.6 and 4.7 also show three noticeable spikes, equally distributed around time steps 8, 16, and 24, which surpass the joint limits for some of the evaluation episodes. This effect is caused by the recomputation of the trajectory, and the interval also matches our chosen action horizon of 8 time steps until we predict the trajectory again.

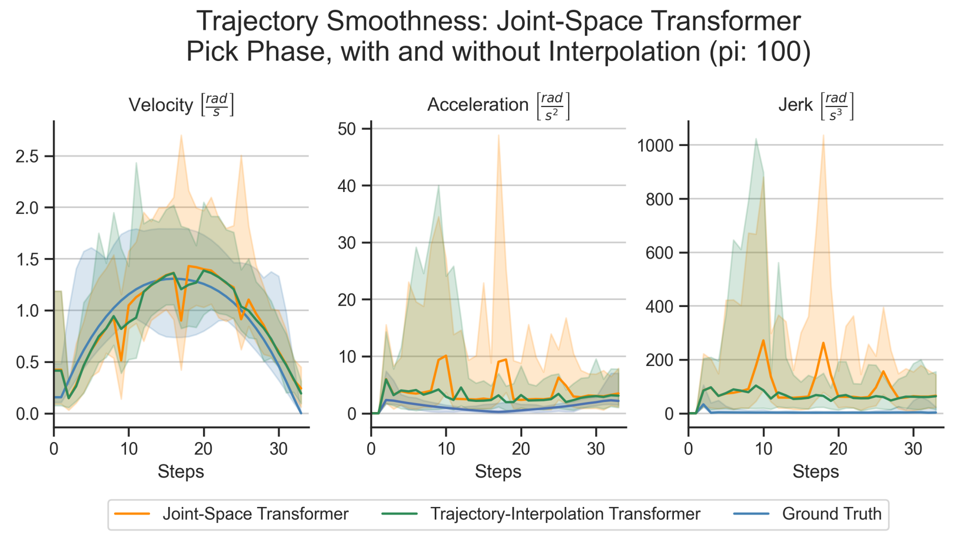


Figure 4.8: This plot shows the maximum absolute value of the velocity, acceleration and jerk for the joint-space transformer model with and without interpolation of the old and new trajectories when a new prediction is performed.

One way to mitigate this problem is to specify a transition interval and interpolate between the old and the new trajectory. Figure 4.8 shows that this effect reduces the spikes, while still being able to recompute the trajectory every 8 steps. The interpolation interval here is set to 2 time steps. In the first time step the action is defined as

$$a_i = \frac{1}{3} \cdot a_{new,i} + \frac{2}{3} \cdot a_{old,i}$$

and the second time step is chosen as

$$a_{i+1} = \frac{2}{3} \cdot a_{new,i+1} + \frac{1}{3} \cdot a_{old,i+1}$$

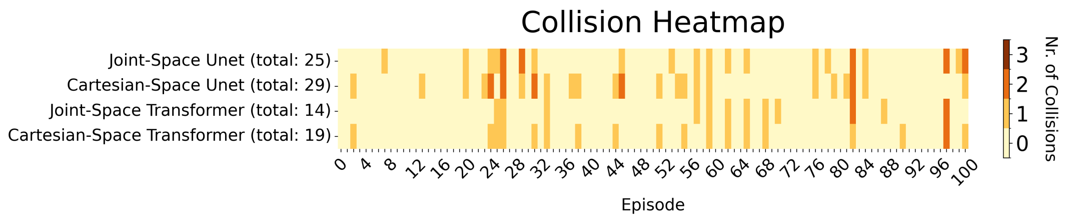


Figure 4.9: The figure shows the number of collisions occurred during the evaluation simulation with the four models.

4.2.3 Collision Avoidance

The final metric on the simulated evaluation is to count how many collisions happened in the 101 evaluation episodes. Figure 4.9 shows the heatmap of all collisions that occurred for each of the models. We defined a phase as collision-free if none of the 61 spheres of the two robots intersected and otherwise counted at most one collision per phase.

Here we have similar results as for the accuracy of the models, where the transformer models perform much better with 14 collisions for the joint-space transformer and 19 collisions for the Cartesian-space transformer, compared to the 25 collisions for the joint-space U-Net and the 29 collisions for the Cartesian-space U-Net. We also see that the joint-space transformer is better capable of avoiding collisions compared to its Cartesian-space counterpart. Table 4.2 show the success rate for a collision-free episode.

Model	Success rate for a collision-free episode
Joint-Space U-Net	$1.0 - (20/101) = 80.2\%$
Cartesian-Space U-Net	$1.0 - (24/101) = 76.2\%$
Joint-Space Transformer	$1.0 - (12/101) = 88.1\%$
Cartesian-Space Transformer	$1.0 - (18/101) = 82.2\%$

Table 4.2: This table shows probability of a collision-free episode.

Discussion — Chapter 5

Next, we will draw conclusions for each of the metrics from the previous chapters, compare the results to those in the Diffusion Policy paper [1], briefly highlight the shortcomings and how to improve them, and finally sum up all the evaluations.

5.1 Metrics

5.1.1 Accuracy

There are two main conclusions from the presented metrics in table 4.1. The first is the jump in accuracy from the U-Net to the transformer architecture. This hints to the fact that a more capable and efficient architecture can get a higher accuracy. Currently, the transformer is the state of the art deep learning architecture for sequence-to-sequence predictions, but future improvements may change this.

The other conclusion is that the models are capable of learning high accuracy when only considering the results for the retract phase. Here all models have sub-centimeter accuracy.

The reason might be that the models need much more training data on each target position. If $\frac{1}{3}$ of the data consists of the same retract target pose, then the models train more frequently on this target and are therefore biased toward it. This large imbalance in the dataset explains why the models get higher accuracy in the retract phase.

5.1.2 Smoothness

The next conclusion focuses on the smoothness of the trajectories. Here we see that the U-Nets are generally better capable of producing smooth trajectories. This is also a finding in [1], where they state that the U-Net has an over-smoothing effect on the predictions. For the task of trajectory prediction, this is a desirable property that comes from the convolutional nature of the U-Net. Overall, the acceleration and jerk values from all four models were most of the time within the limits of the robot with only a few exceptions.

From our empirical results we found that by training longer, the trajectories tended to get smoother. But it is unclear whether training the transformer model for much longer than $1'500$ epochs helps in this case, since the evaluation plots in figure 4.2 show that the smoothness values converged.

One cause for non-smooth trajectories, discussed in the results chapter and visualised in figure 4.8, is that the current models have higher jerk and acceleration values during the replanning phase. This can be resolved with interpolation between the old and the new trajectory but it would be desirable to have a model that does not need a manual fix.

5.1.3 Collision Avoidance

Regrading the capability to avoid collisions, we observe an increase in the success rate from the transformer architecture over the U-Net, as shown in table 4.2. The reason is most likely the higher accuracy combined with the more powerful architecture of the transformer.

The results also show that the Cartesian-space models suffer from the indirect way they predict the trajectories compared to the joint-space models. For both architectures the number collisions decrease when we switch to the joint-space equivalent of the model. Even though the Cartesian-space transformer model has the highest accuracy, this does not guarantee that the model also gets the lowest number of collisions. The Cartesian-space models have no control over the exact pose that the inverse kinematic solver produces. Therefore they are limited in steering the link positions solely from the end-effector state. Furthermore, in this setup, kinematic singularities can occur more easily due to the lack of direct control over the joints. By trading off accuracy versus collision avoidance, it is clear that the joint-space transformer model has the most potential. Also when considering that the accuracy issue can potentially be solved with a larger, balanced dataset and an improved architecture. Overall, our best success rate in predicting a collision-free trajectory is 88.1% and comes from the joint-space transformer.

5.2 Comparison to the Diffusion Policy Paper

We verified several assertions presented in the Diffusion Policy paper [1], including the adaptability of diffusion models to new tasks, the performance increase coming from the transformer compared to the U-Net architecture, the over-smoothing effect of the U-Net predictions, the performance decline from too large models and the training stability of diffusion models.

On the other hand we were not able to achieve the same accuracy on our own task. This might be due to the change in the setup with the larger prediction horizon (34 compared to 16), too few training data points or the imbalance in our dataset. However, the results from the accuracy in the retract phase shows that with enough data on a particular target point, higher accuracy can be achieved with all four models.

5.3 Limitations

A disadvantage of the current setup is that it does not allow for scaling to many robots. The collision objects are passed to the network as a fixed set of sphere coordinates and do not allow for more robots in the scene. Utilising an approach like in [10] where the collision objects are passed as a point cloud to an encoding network would certainly improve the setup.

Furthermore, the models in their current form do not allow for a simulation with two actuated robots controlled by the diffusion models. The problem lies in the setup of the dataset, where the ghost robot is mirrored along the x-axis but not along the y-axis. If mirrored along both axes, this would represent a 180°-rotation, allowing us to place another Panda robot opposite to the first one during the evaluation. Then, the setup would be the same as the one the model was trained on. The current models are not capable of producing usable results when the ghost robot is replaced with a real Panda robot. However, one conclusion from this setup was that the transformer models are better capable of trying to avoid collisions, while the U-Nets produced smooth trajectories that collided most of the time with the other robot arm.

Another problem of our dataset is that it only consists of collision-free trajectories and that it has no negative samples, that contain collisions. This problem is also mentioned in [1, 11] as an inherent problem of behavior cloning. Utilising negative samples to properly learn the boundaries between collision-occurring and collision-free trajectories could be an important step towards generating better collision-avoidant trajectories.

5.4 Future Work

First of all, it would be interesting to address the drawbacks from the previous section. The first one would need an adaptation to the architecture but could generally be done in a similar way with a PointNet++ [27] like in the Motion Policy Networks paper [10]. The second problem could simply be addressed by resampling the dataset and retraining the models.

To address the last point from section 5.3 and to increase the models' success rate in handling collisions, one could utilise diffusion models within a reward-based Reinforcement Learning setup. A summary of different approaches is shown in [28].

Another approach cloud include training on two datasets. The first contains collision-free trajectories (the positive dataset) and the second only has negative samples that contain collisions. When learning on the positive dataset, we perform gradient descent, and when training on the negative dataset we use gradient ascent. This approach, proposed in [29], is utilised to fine-tune a Language Model to unlearn undesired next-token predictions.

Furthermore, it would be interesting to test the models accuracy by using a weighted sampling method during train time to reduce the imbalance of the dataset. This way we could sample data points from the pick and place phases much more often compared to the ones from the retract phase. Along with the rebalancing, utilising more training data would probably also increase the accuracy and the generalisation capability of the models.

To address the smoothness issues of the transformer architecture, a longer training duration beyond $1'500$ epochs can help solve the issue. Otherwise, adding several convolution layers at the end of the transformer could lead to a learned smoothing effect on the trajectories. This way, we could combine the strengths of both architectures, where the attention mechanism contributes to higher accuracy and collision-free trajectories, and the convolutional output layers refine the trajectory to be smoother.

5.5 Conclusion

To conclude, we presented four different models, two of them utilised a U-Net architecture and the other two a transformer architecture. We have shown that it is possible to adapt diffusion models, as presented in [1], to a dynamic collision avoidance environment.

This work also demonstrates the improvement achieved in collision avoidance by directly predicting joint-space actions compared to the Cartesian-space models. The latter still require an inverse kinematics solver and therefore lack the ability to fully control the robot arm. The joint-space models need to learn the forward kinematics of the robot but have therefore a more precise control. When utilised with the more powerful transformer architecture, this model type achieves the best results regarding collision avoidance and accuracy combined.

Overall, we still have the limitations of behavior cloning, as mentioned in [11]. We would have to actively learn the boundaries between collision-free and collision-occurring trajectories in order to truly get collision-avoidant trajectories. However, the potential of diffusion models within the behavior cloning domain is large, especially when considering the scalability that was shown from Large Language Models in the natural language processing domain.

We hope that this work will contribute to further improvements on the combination of diffusion models and collision-free trajectory optimisation. Furthermore, we are optimistic that the current shortcomings can be solved and that transformer-based diffusion models play a key role in the future of motion planning and trajectory optimisation for robots.

Bibliography

[1] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," 2024.

[2] C. Zhou, B. Huang, and P. Fränti, "A review of motion planning algorithms for intelligent robotics," 2021.

[3] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998. [Online]. Available: https://api.semanticscholar.org/CorpusID:14744621

[4] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," 2011.

[6] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. Available: https://doi.org/10.1109/tssc.1968.300136

[7] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[8] M. Zare, P. M. Kebria, A. Khosravi, and S. Nahavandi, "A survey of imitation learning: Algorithms, recent developments, and challenges," 2023.

[9] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 1. Morgan-Kaufmann, 1988. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf

[10] A. Fishman, A. Murali, C. Eppner, B. Peele, B. Boots, and D. Fox, "Motion policy networks," 2022.

[11] H. Ha, J. Xu, and S. Song, "Learning a decentralized multi-arm motion planner," 2020.

[12] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.

[13] J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters, "Motion planning diffusion: Learning and planning of robot motions with diffusion models," 2024.

[14] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001 vol.2.

[15] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf

[16] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. V. Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, N. Ratliff, and D. Fox, "curobo: Parallelized collision-free minimum-jerk robot motion generation," 2023.

[17] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," 2021.

[18] F. R. GmbH. (2024) Franka robotics webpage. Accessed: 12.04.2024. [Online]. Available: https://franka.de/de/

[19] A. R. Geist, J. Frey, M. Zobro, A. Levina, and G. Martius, "Learning with 3d rotations, a hitchhiker's guide to so(3)," 2024.

[20] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[21] G. M. Brocal and G. Peeters, "Conditioned-u-net: Introducing a control mechanism in the u-net for multiple source separations," 2019. [Online]. Available: https://zenodo.org/record/3527766

[22] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," 2017.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[24] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8162–8171. [Online]. Available: https://proceedings.mlr.press/v139/nichol21a.html

[25] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning,” http://pybullet.org, 2016–2021.

[26] M. Sugiyama, M. Krauledat, and K.-R. Müller, “Covariate shift adaptation by importance weighted cross validation,” *J. Mach. Learn. Res.*, vol. 8, p. 985–1005, dec 2007.

[27] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” 2017.

[28] Z. Zhu, H. Zhao, H. He, Y. Zhong, S. Zhang, H. Guo, T. Chen, and W. Zhang, “Diffusion models for reinforcement learning: A survey,” 2024.

[29] D. Yoon, J. Jang, S. Kim, and M. Seo, “Gradient ascent post-training enhances language model generalization,” 2023.

[30] OpenAI, “Chatgpt 3.5,” 2024, https://www.openai.com. [Online]. Available: https://chatgpt.com

[31] GitHub, “Github copilot,” 2024, aI-powered code completion tool. [Online]. Available: https://github.com/features/copilot

[32] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 10 684–10 695.

[33] L. Zhang and M. Agrawala, “Adding conditional control to text-to-image diffusion models,” 2023.

Appendix

Appendix A

A.1 Generative AI

In this work, we used generative AI, such as ChatGPT [30] and Github Copilot [31], for repetitive tasks like plotting and visualising data, finding bugs in the code and for brainstorming.

The title image of this thesis was generate using Stable Diffusion [32], in combination with ControlNet [33]. The diffusion process is guided by an image-to-image process, along with a prompt. As input image we used a screenshot from our evaluation setup with the two Panda robots in PyBullet. The prompt used to condition the diffusion process was:

"A painting of two robot arms that perform collision
avoidance with a pick and place task,
Salvador Dali style, surreal, abstract,
warm cold contrast, sun is shining in the top right."

A.2 Log Distance of End-Effector to Target

The plots A.3, A.4, A.1, and A.2 show the log distance between the end-effector and the target for each model.

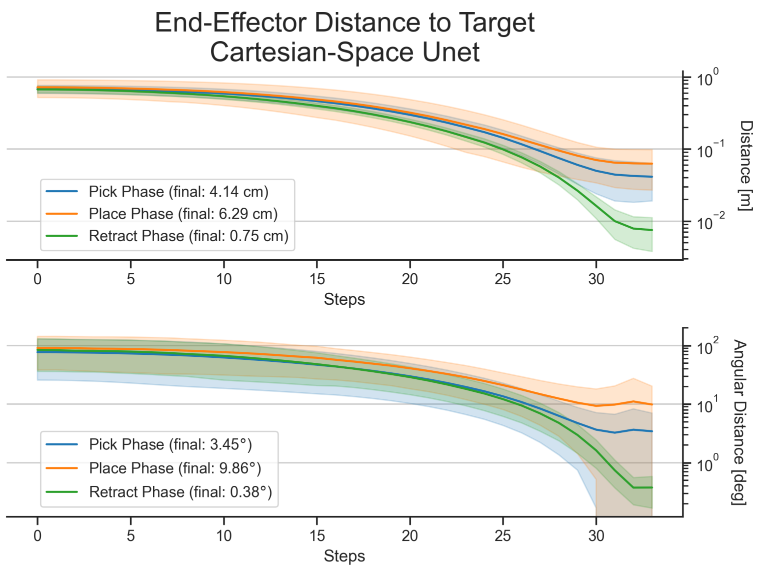


Figure A.1

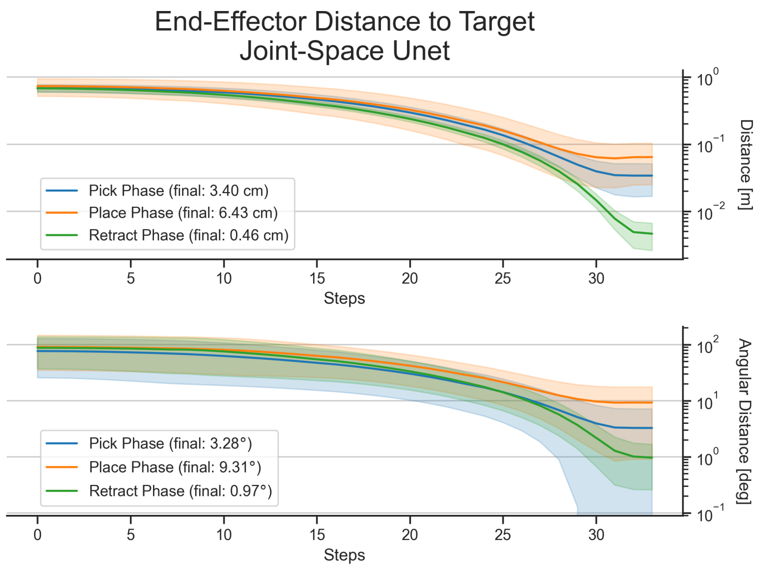


Figure A.2

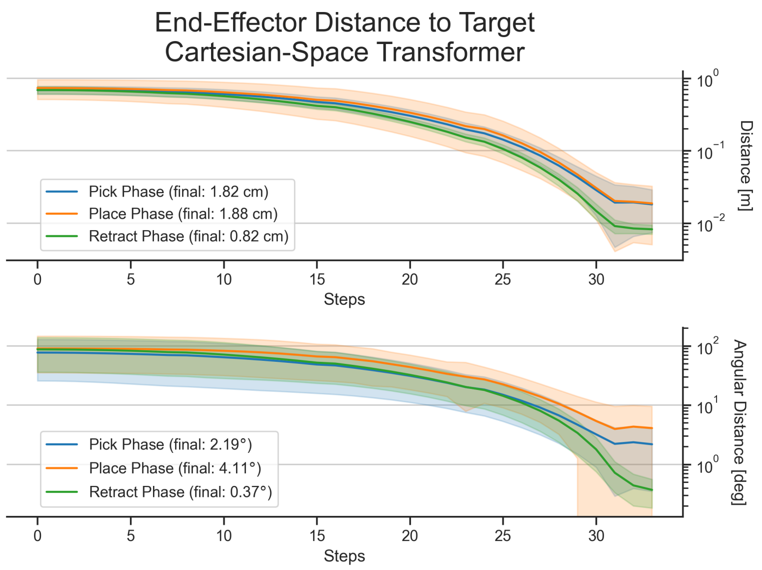


Figure A.3

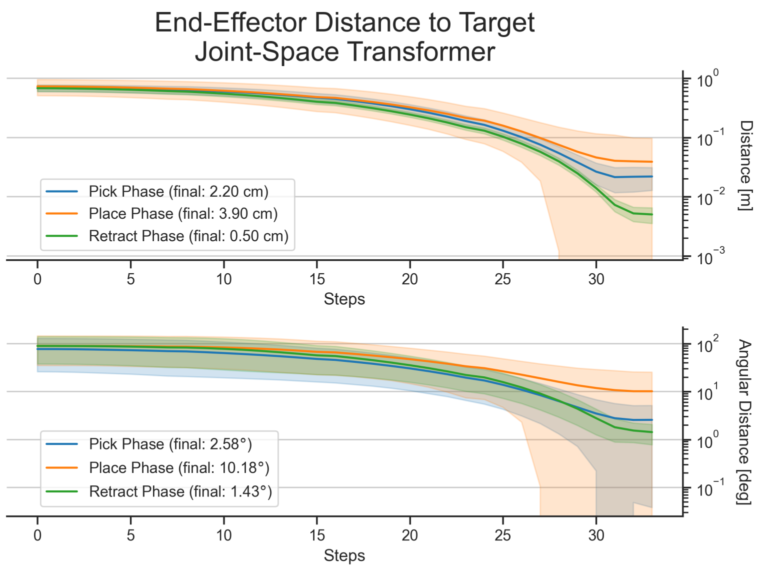


Figure A.4

A.3 Trajectory Smoothness Plots

The plot A.5 shows the comparison of the Cartesian-space models on trajectory smoothness, figure A.7 shows the same for the joint-space models and figure A.9 visualises the comparison with and without trajectory interpolation during recomputation of the trajectory.

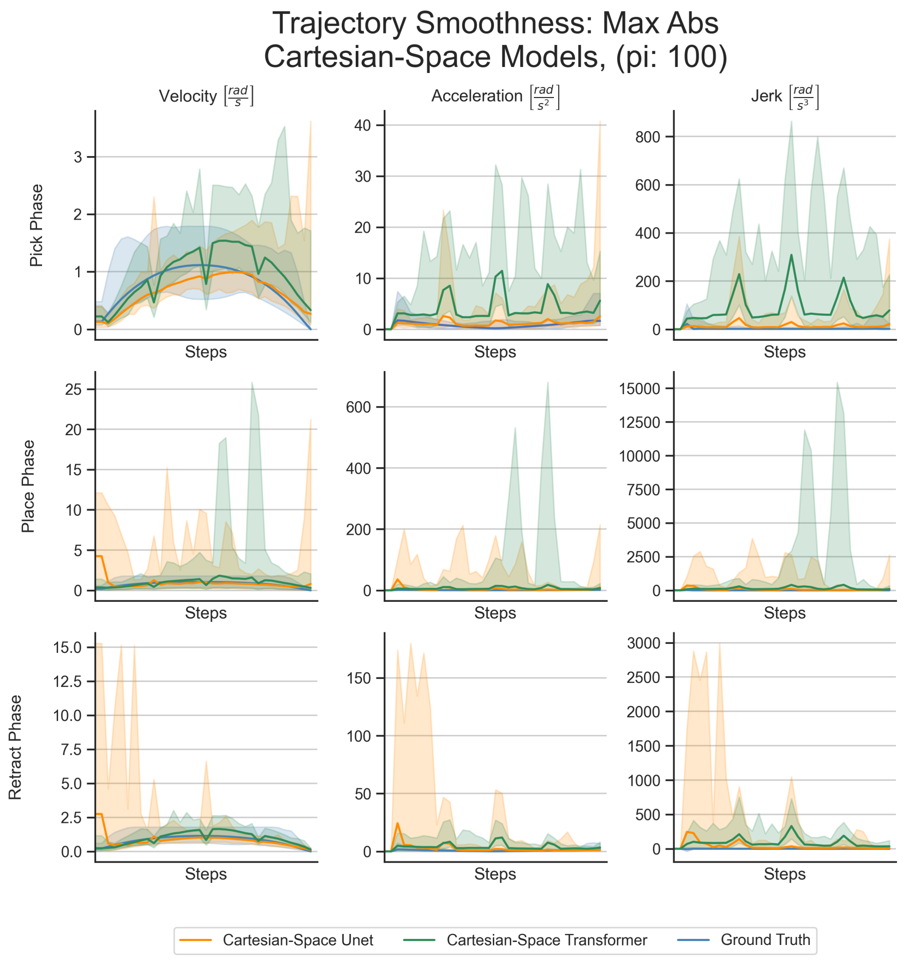


Figure A.5

The plots A.6, A.8 and A.10 show the same smoothness values but for the middle 75 percentile of the data around the mean.

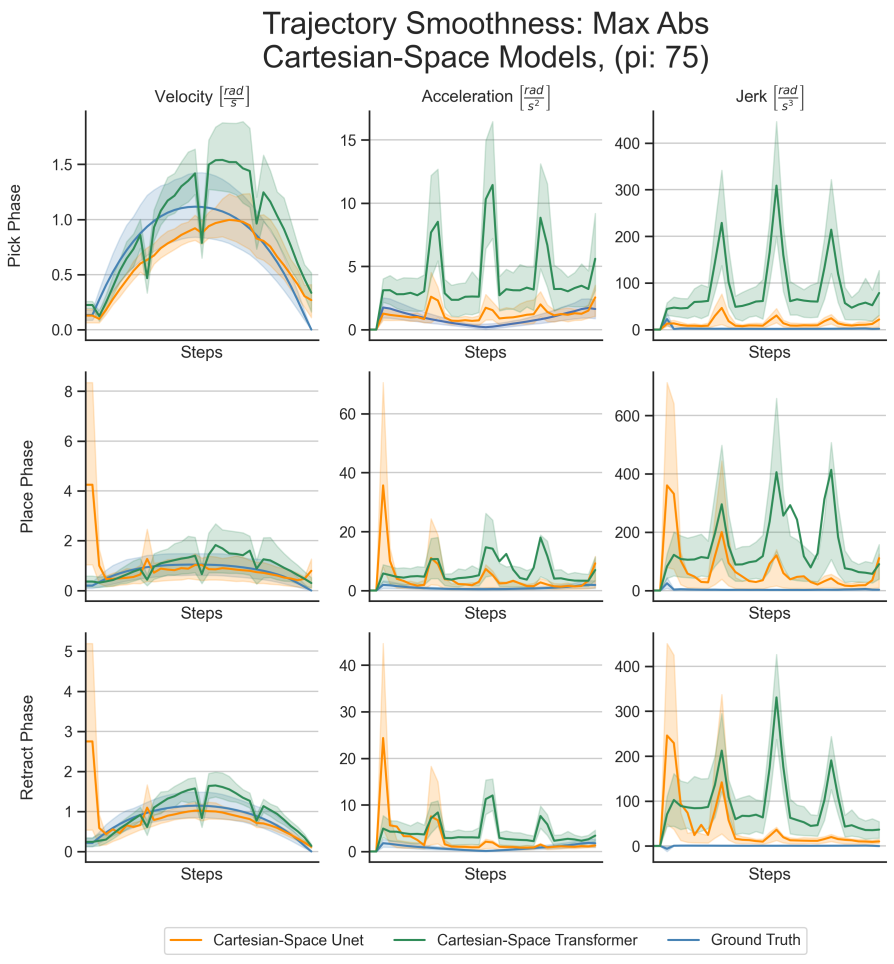

Trajectory Smoothness: Max Abs
Cartesian-Space Models, (pi: 75)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Cartesian-Space Unet
Cartesian-Space Transformer
Ground Truth

Figure A.6

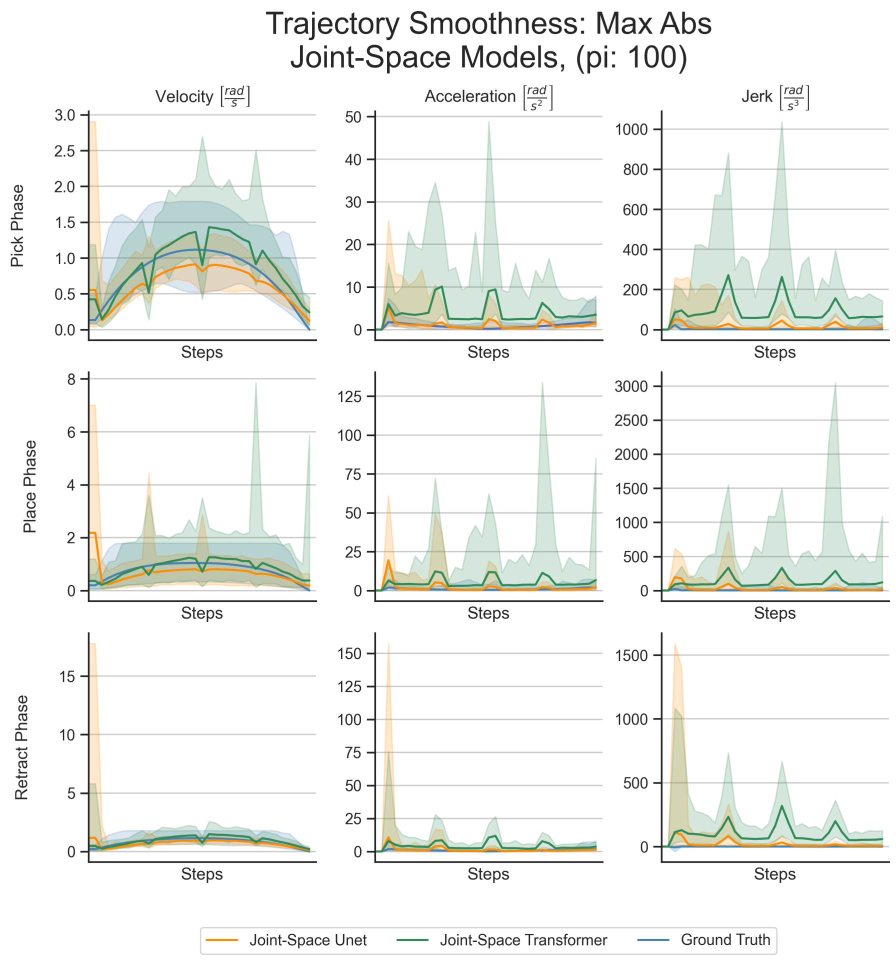
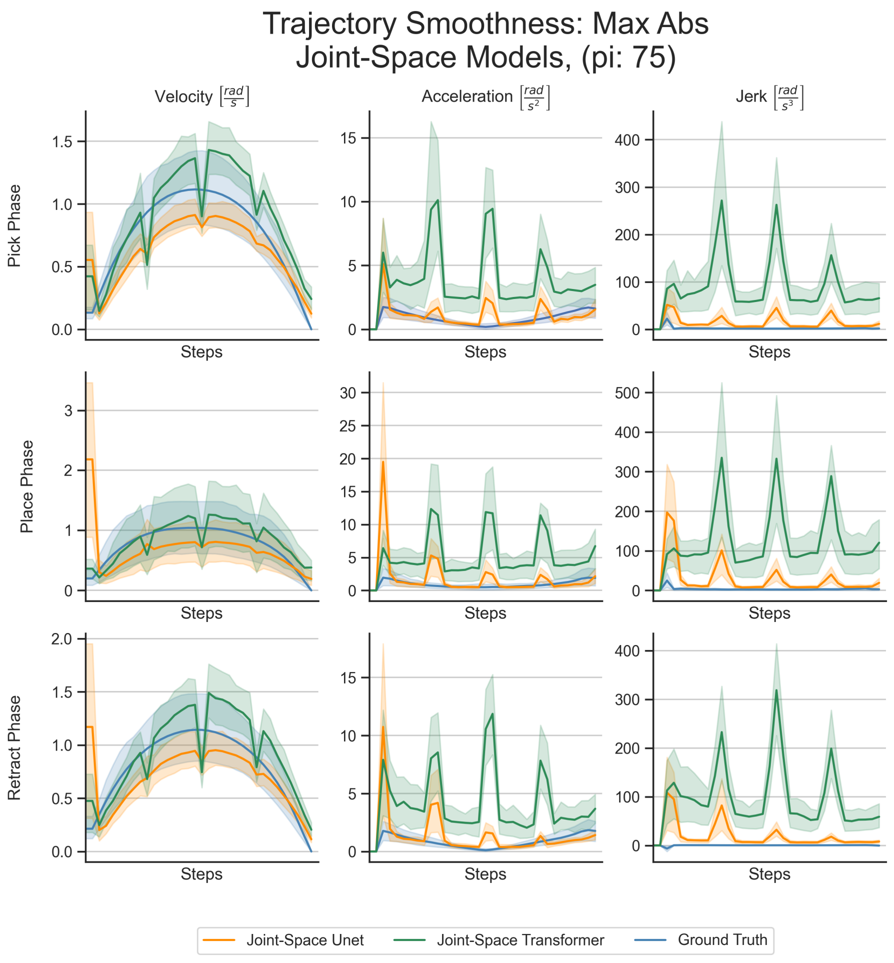

Trajectory Smoothness: Max Abs
Joint-Space Models, (pi: 100)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Joint-Space Unet
Joint-Space Transformer
Ground Truth

Figure A.7



Trajectory Smoothness: Max Abs
Joint-Space Models, (pi: 75)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Joint-Space Unet
Joint-Space Transformer
Ground Truth

Figure A.8

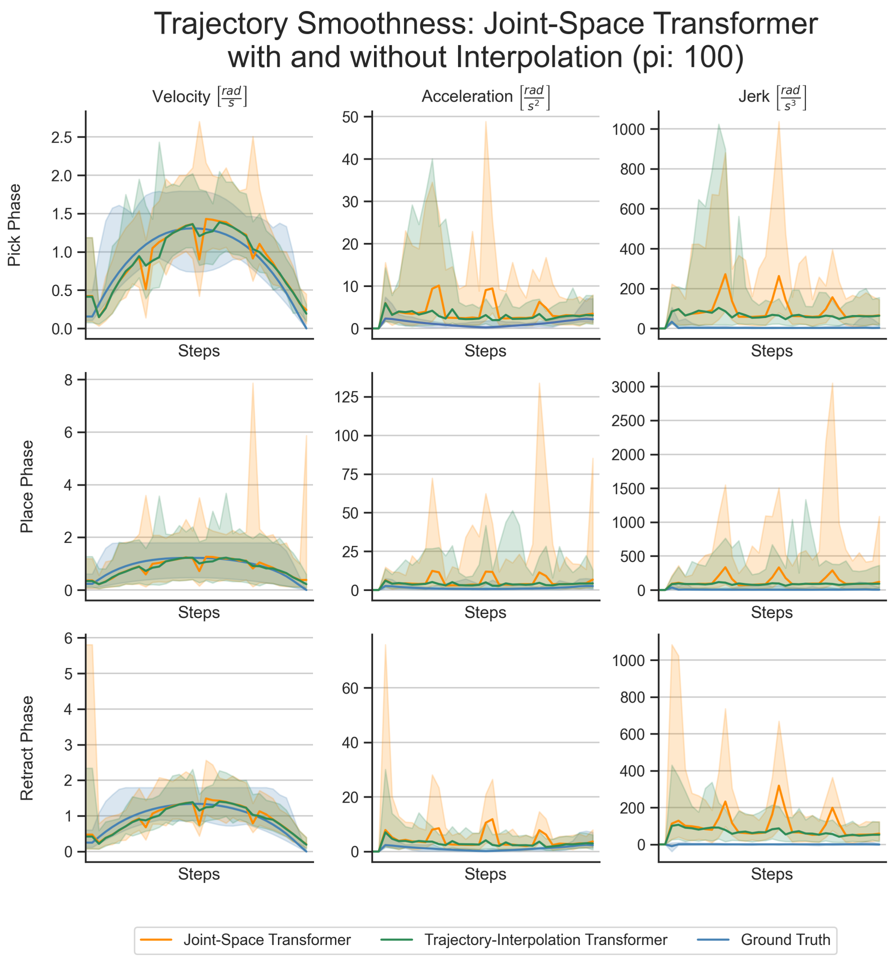

Trajectory Smoothness: Joint-Space Transformer with and without Interpolation (pi: 100)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Joint-Space Transformer
Trajectory-Interpolation Transformer
Ground Truth

Figure A.9

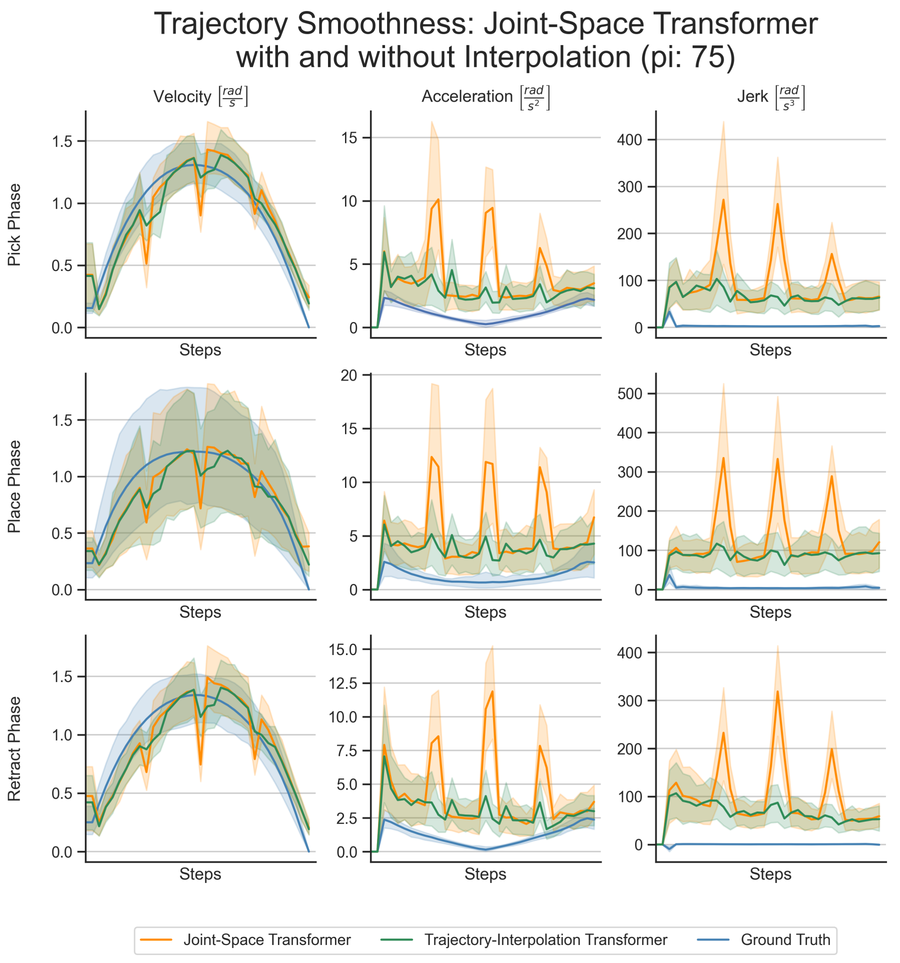

Trajectory Smoothness: Joint-Space Transformer with and without Interpolation (pi: 75)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Joint-Space Transformer
Trajectory-Interpolation Transformer
Ground Truth

Figure A.10

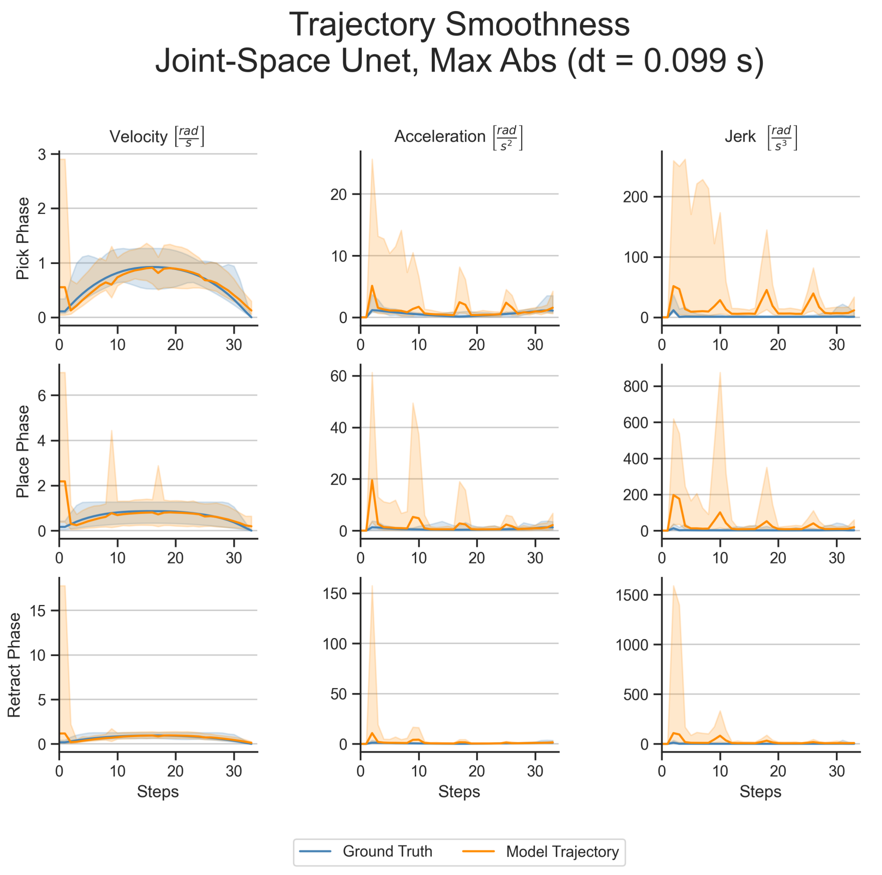


Figure A.11

The plots (A.11 , A.12 , A.13 , A.14) show the maximum absolute value of the velocity, acceleration and jerk for each model separately.

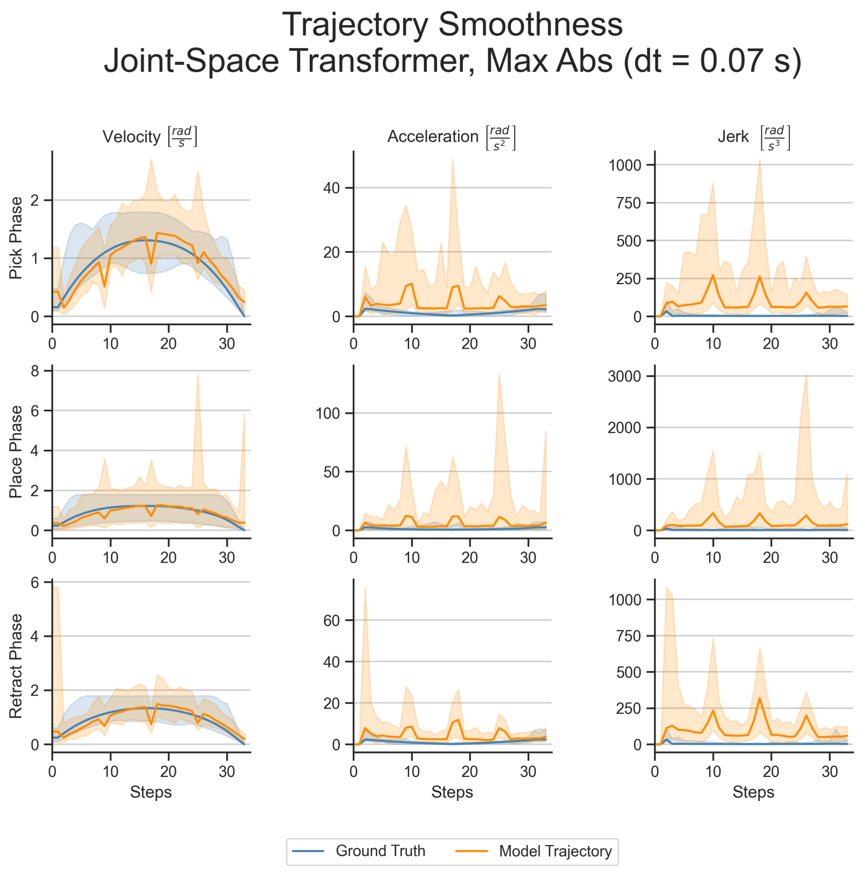

Trajectory Smoothness
Joint-Space Transformer, Max Abs (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.12

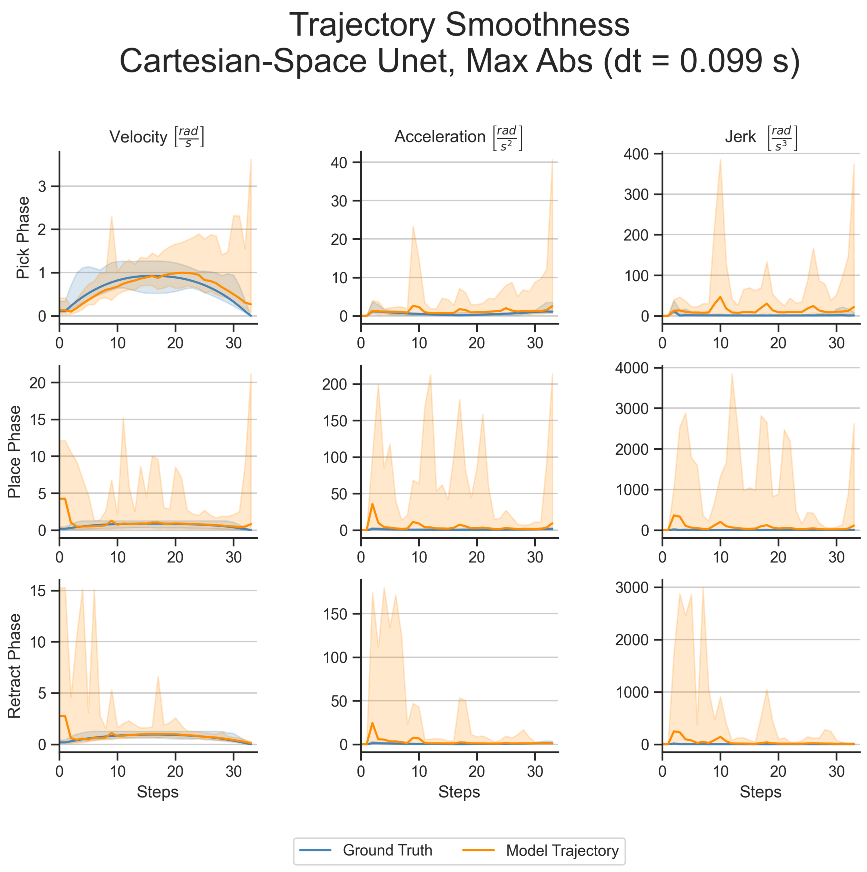

Trajectory Smoothness
Cartesian-Space Unet, Max Abs (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.13

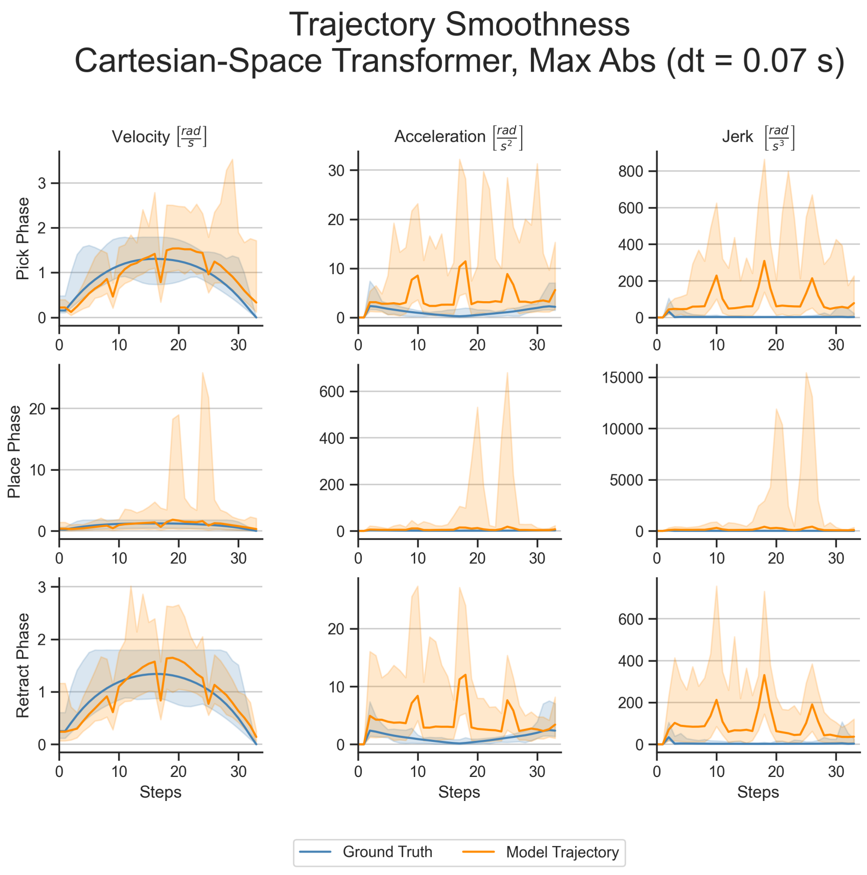

Trajectory Smoothness
Cartesian-Space Transformer, Max Abs (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.14

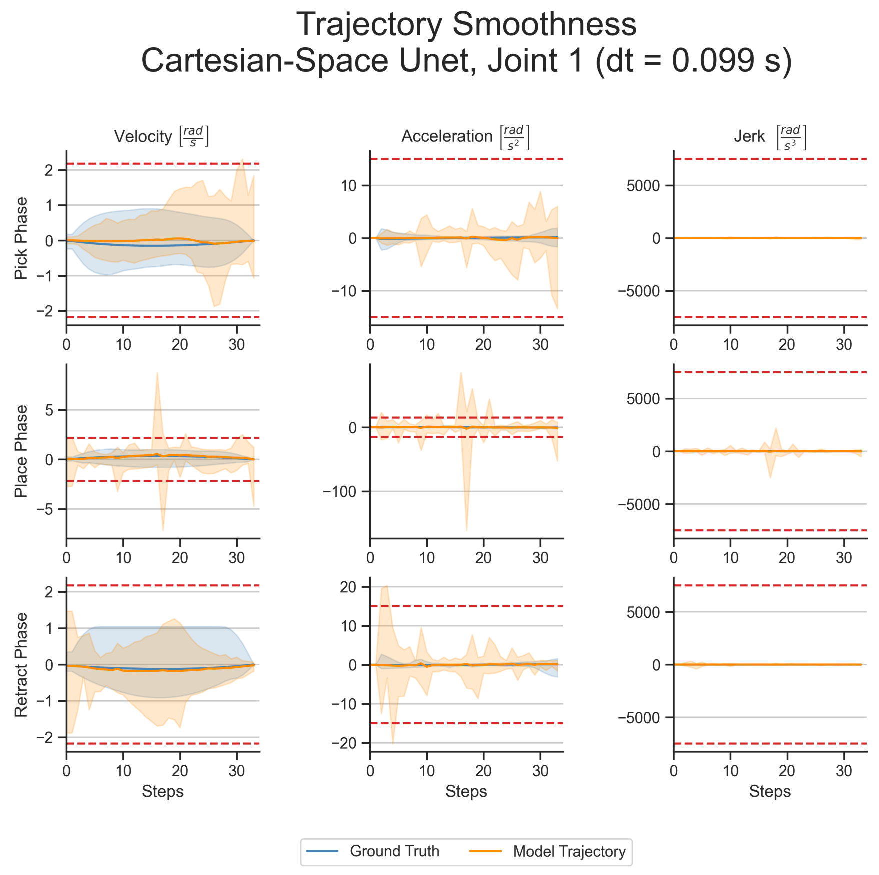


Figure A.15

A.3.1 Cartesian-Space U-Net

The plots A.15, A.16, A.17, A.18, A.19, A.20 and, A.21 show the per joint values of the velocity, acceleration and jerk for the Cartesian-space U-Net model.

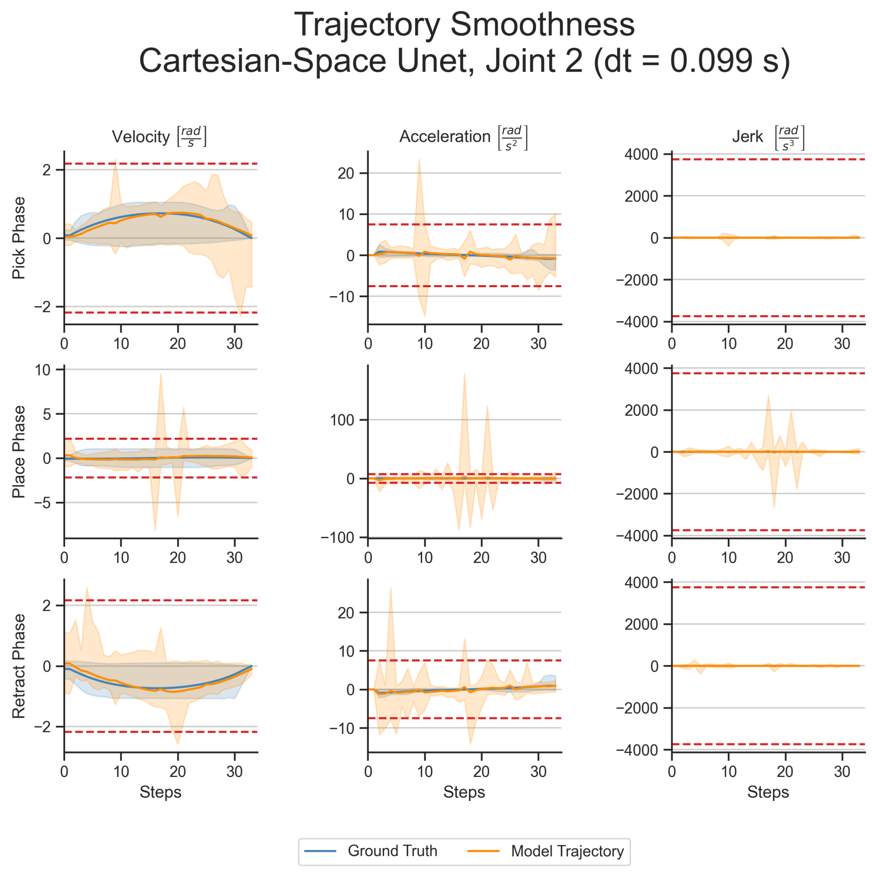

Trajectory Smoothness
Cartesian-Space Unet, Joint 2 (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.16

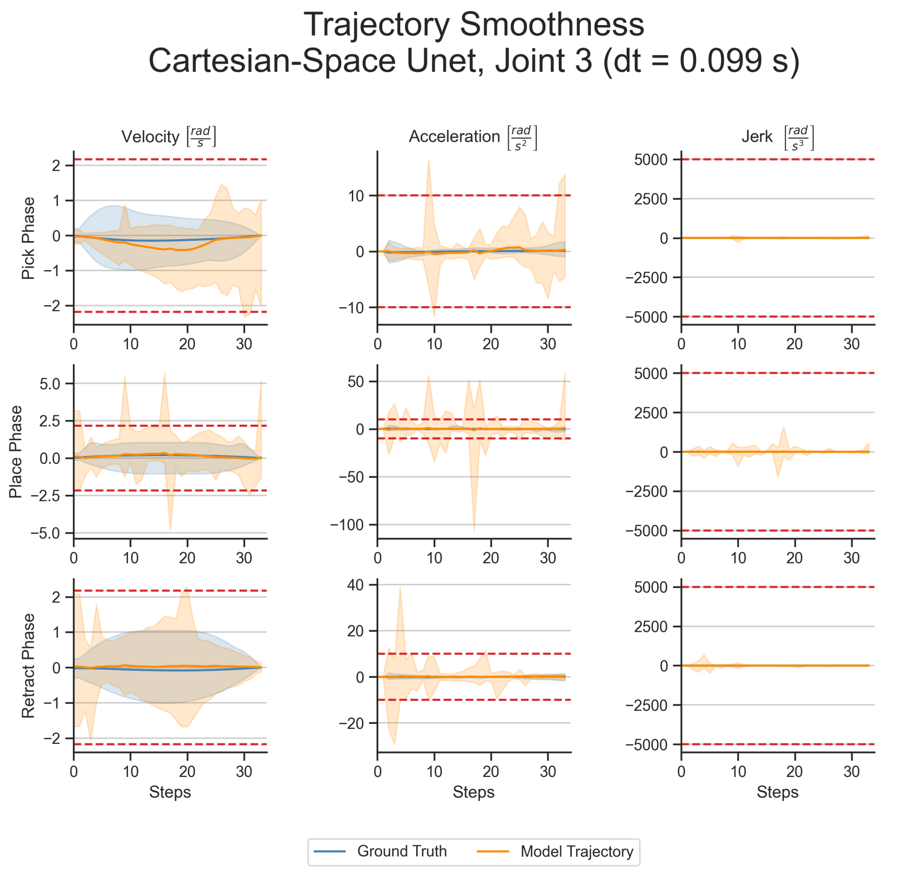

Trajectory Smoothness
Cartesian-Space Unet, Joint 3 (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.17

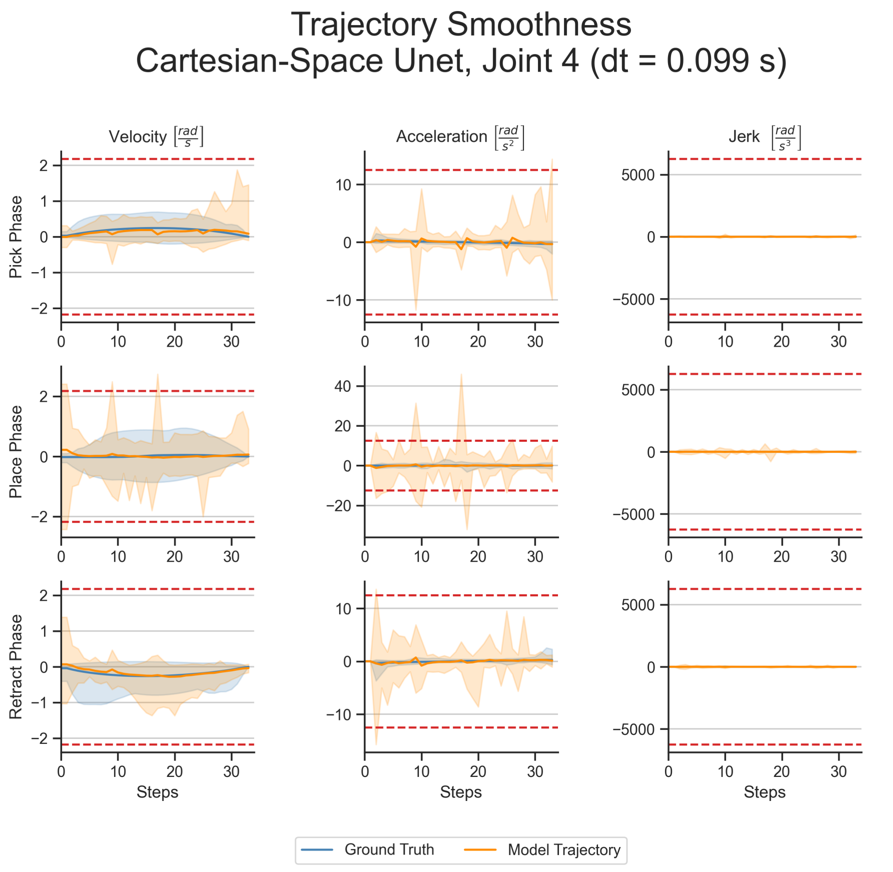

Trajectory Smoothness
Cartesian-Space Unet, Joint 4 (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.18

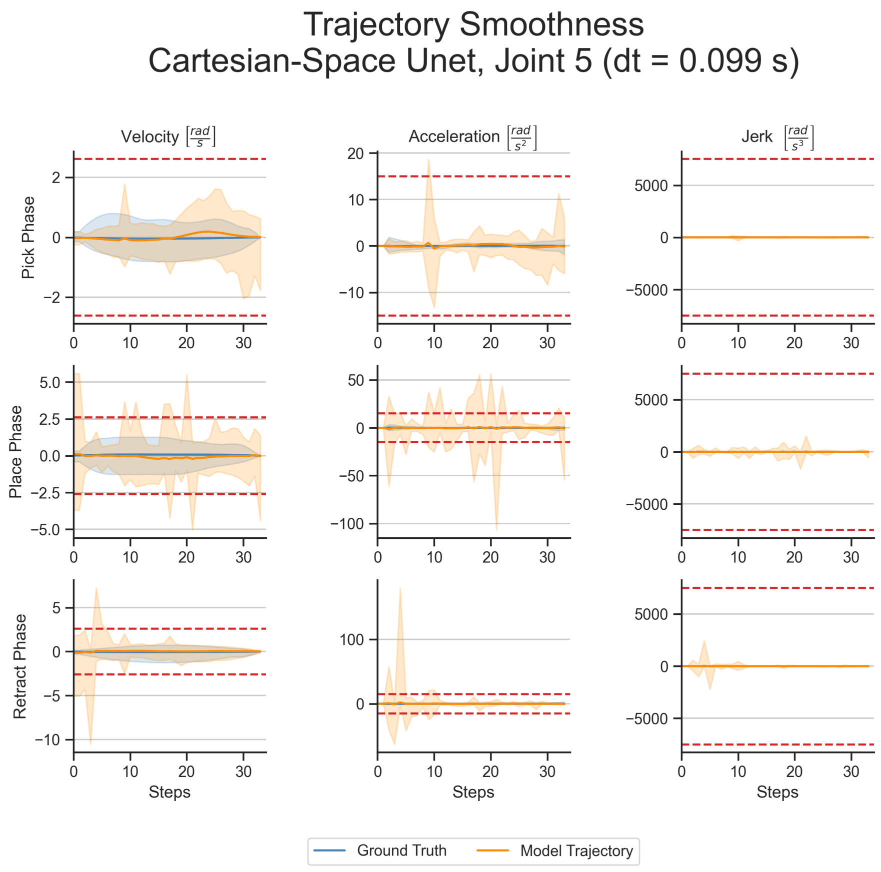

Trajectory Smoothness
Cartesian-Space Unet, Joint 5 (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.19

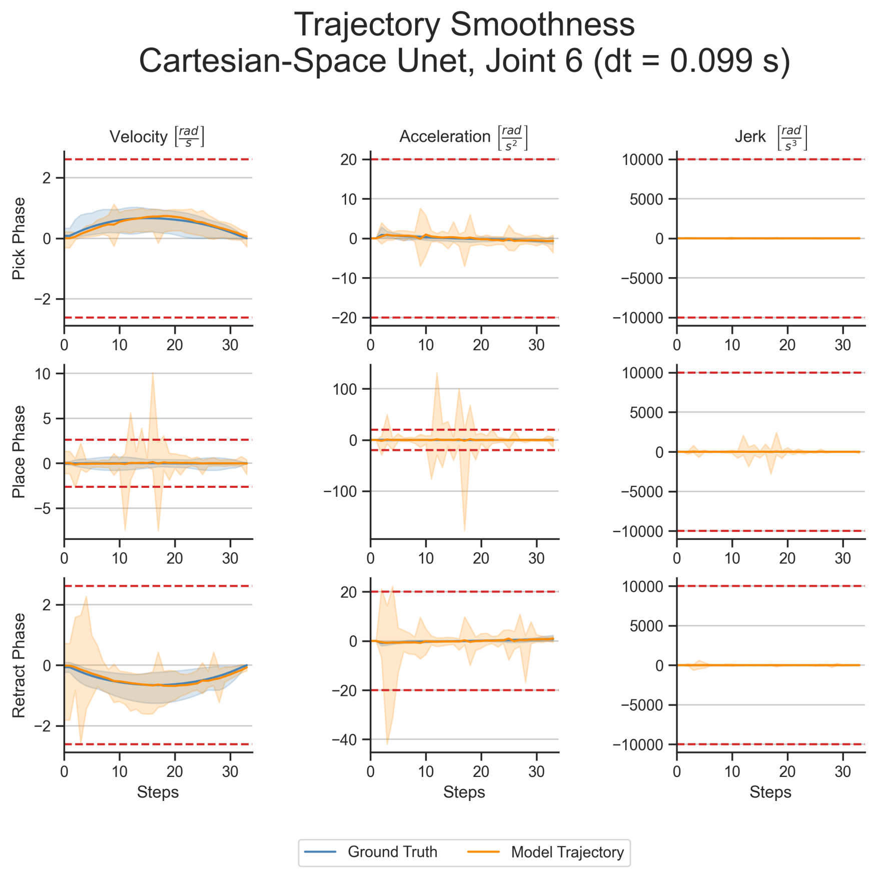
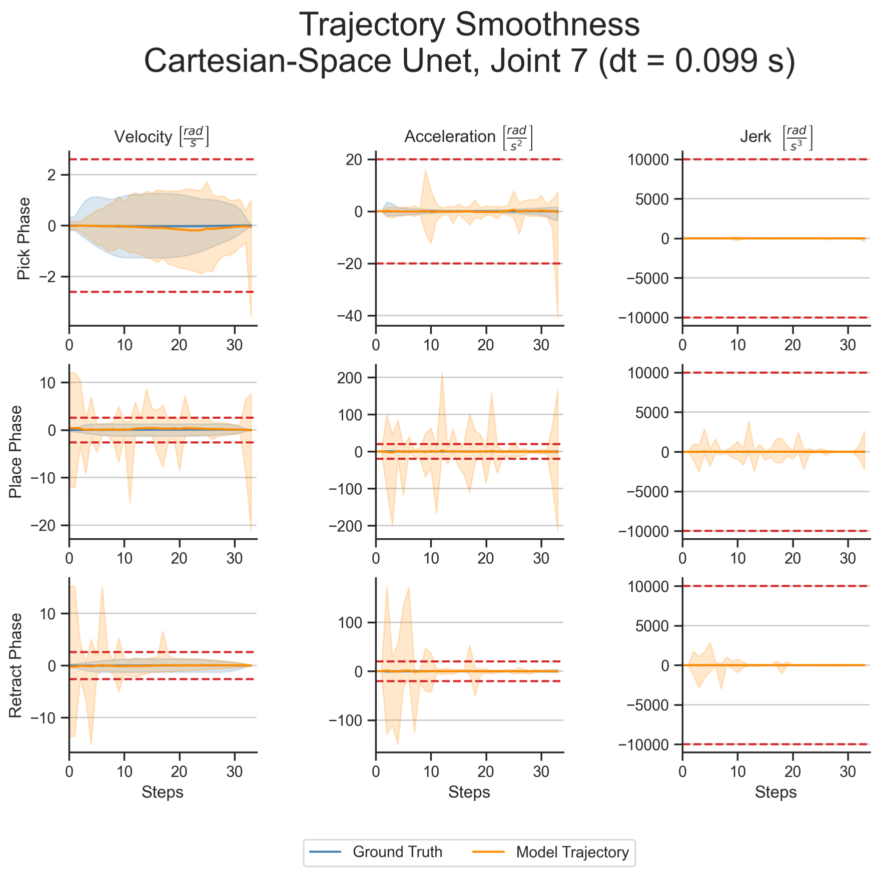

Trajectory Smoothness
Cartesian-Space Unet, Joint 6 (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.20



Trajectory Smoothness
Cartesian-Space Unet, Joint 7 (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.21

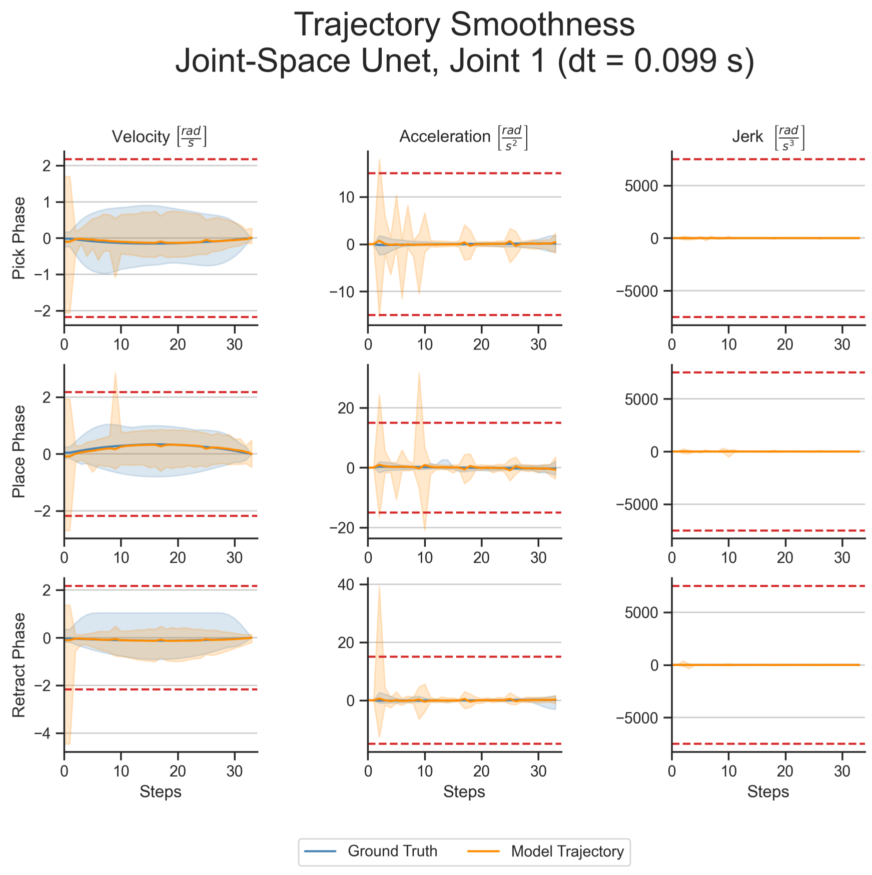


Figure A.22

A.3.2 Joint-Space U-Net

The plots A.22, A.23, A.24, A.25, A.26, A.27, and A.28 show the per joint values of the velocity, acceleration and jerk for the joint-space U-Net model.

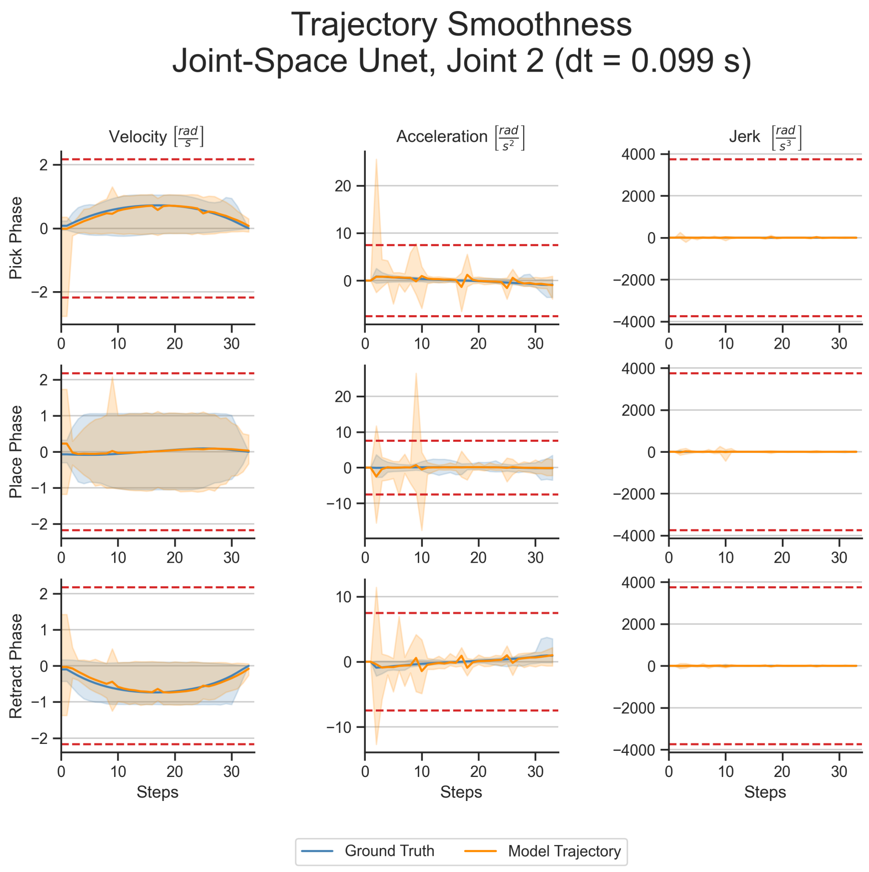

Trajectory Smoothness
Joint-Space Unet, Joint 2 (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.23

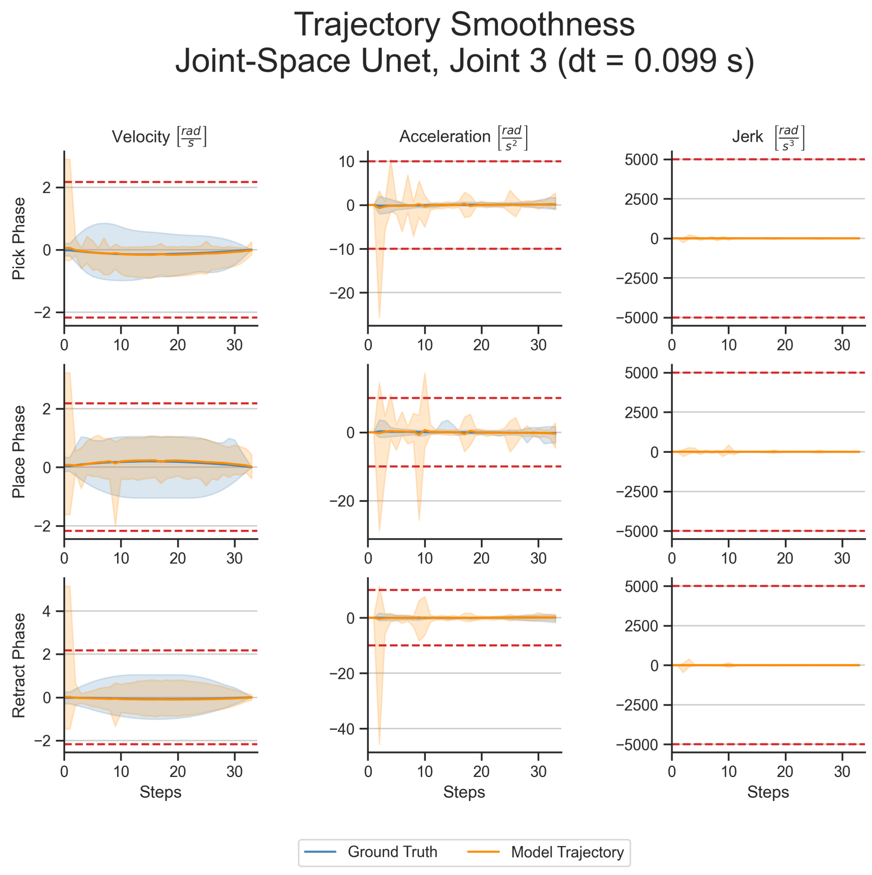
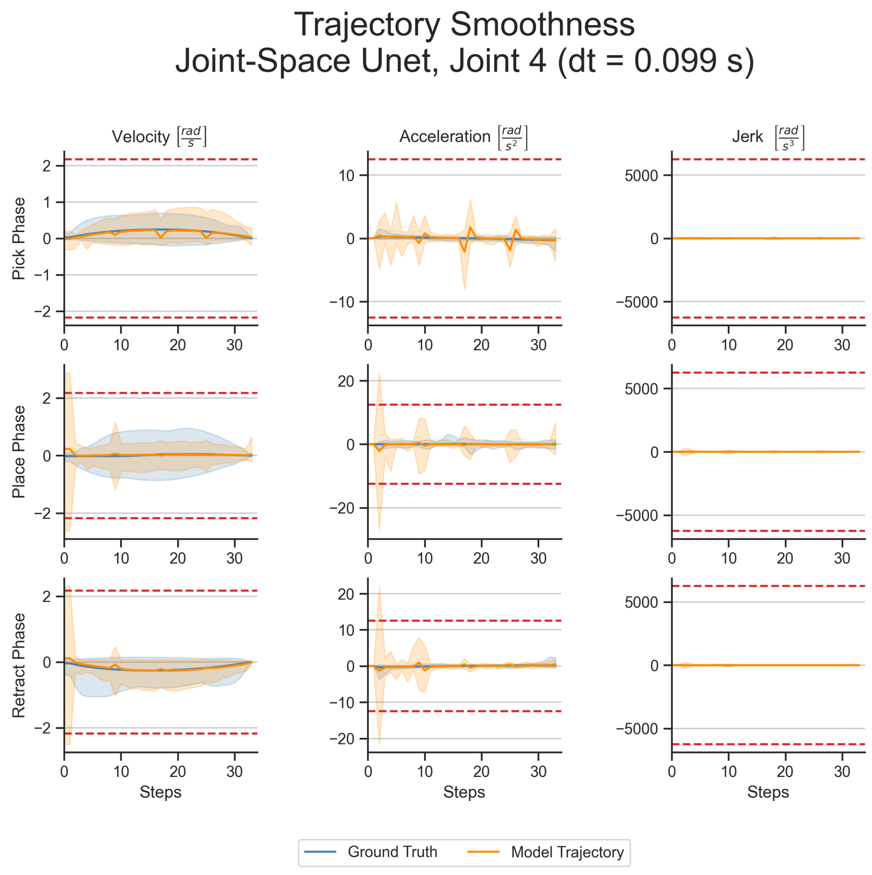

Trajectory Smoothness
Joint-Space Unet, Joint 3 (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.24



Trajectory Smoothness
Joint-Space Unet, Joint 4 (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.25

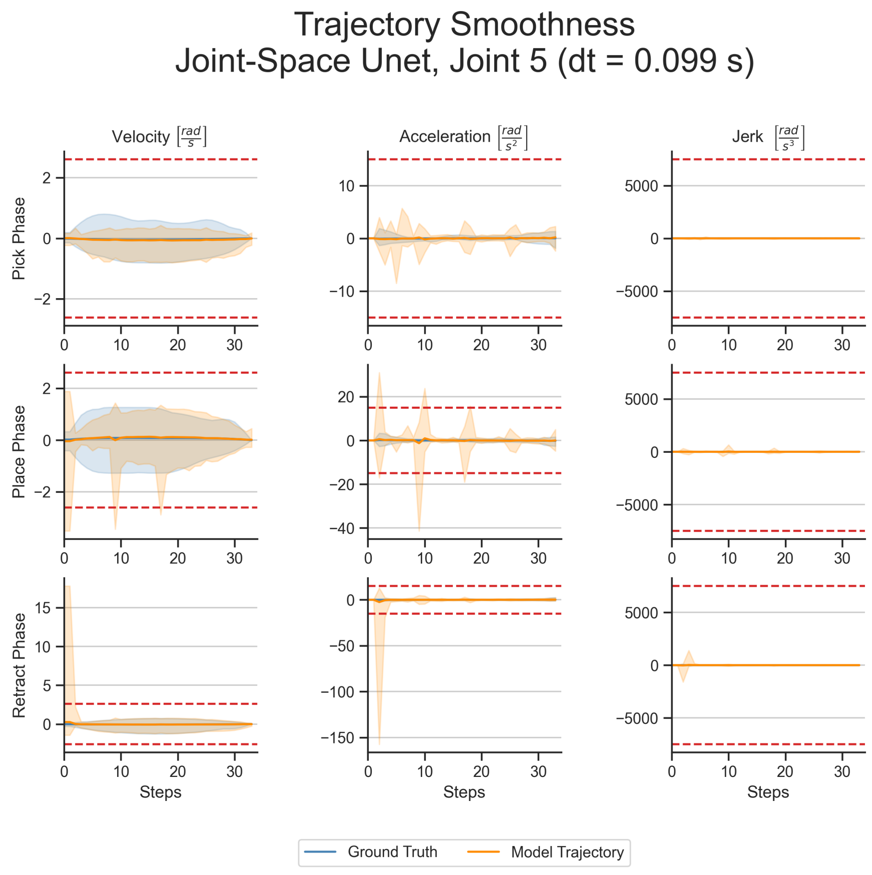

Trajectory Smoothness
Joint-Space Unet, Joint 5 (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.26

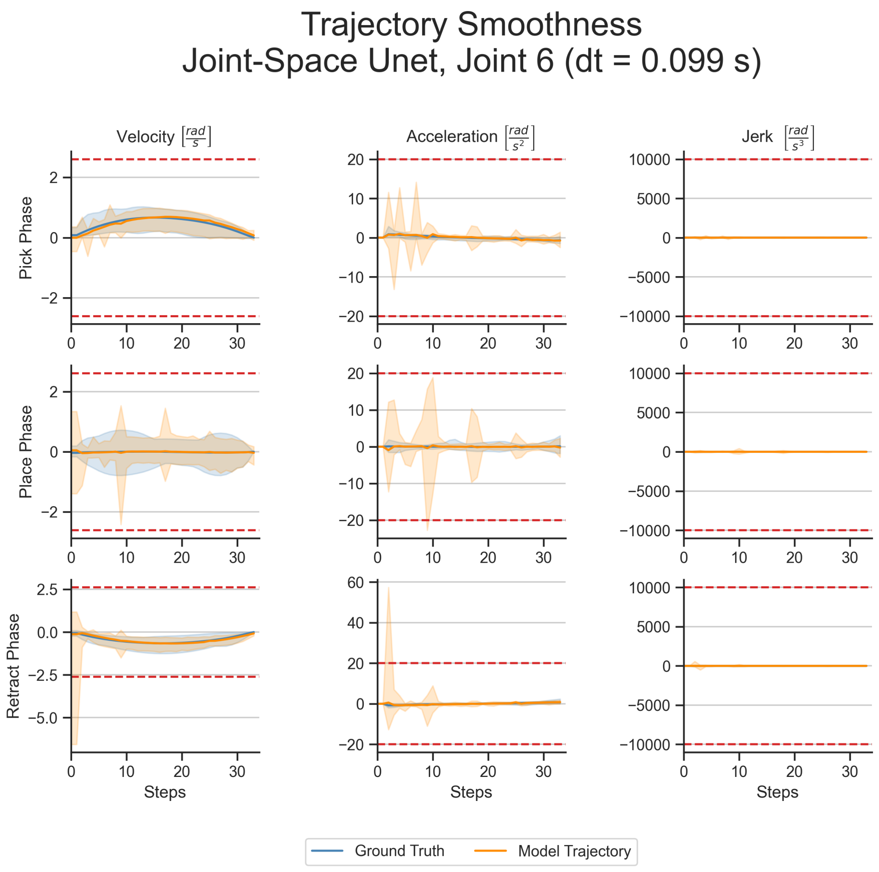

Trajectory Smoothness
Joint-Space Unet, Joint 6 (dt = 0.099 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.27

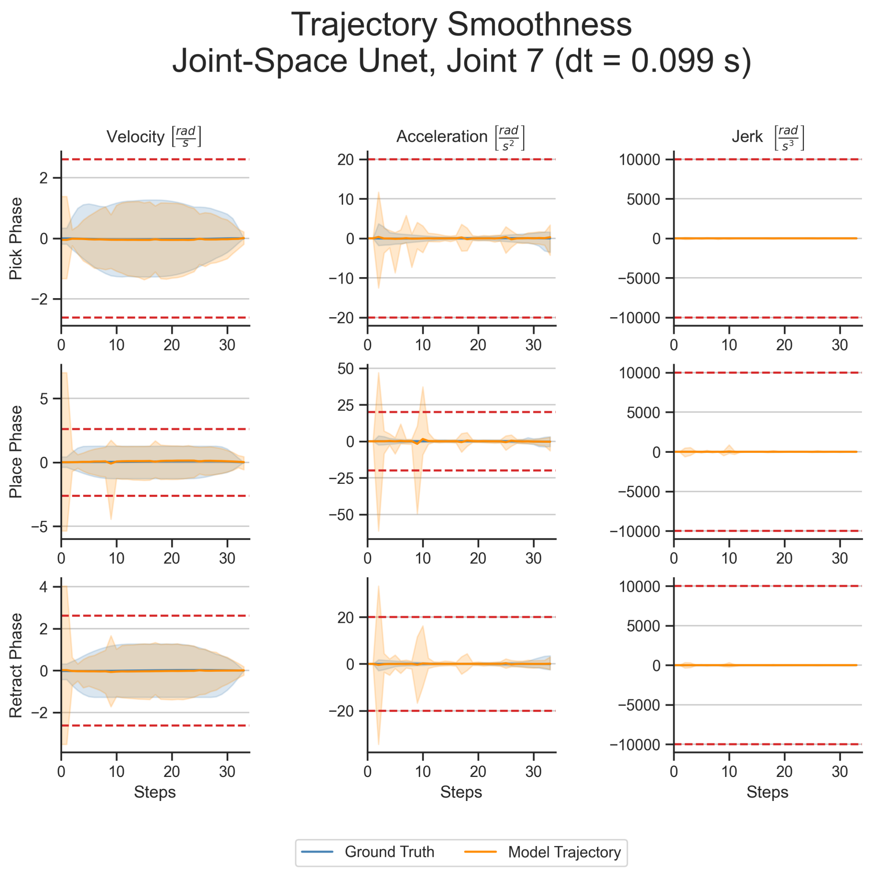

Trajectory Smoothness
Joint-Space Unet, Joint 7 (dt = 0.099 s)
Velocity [rad/s]
Acceleration [rad/s²]
Jerk [rad/s³]
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.28

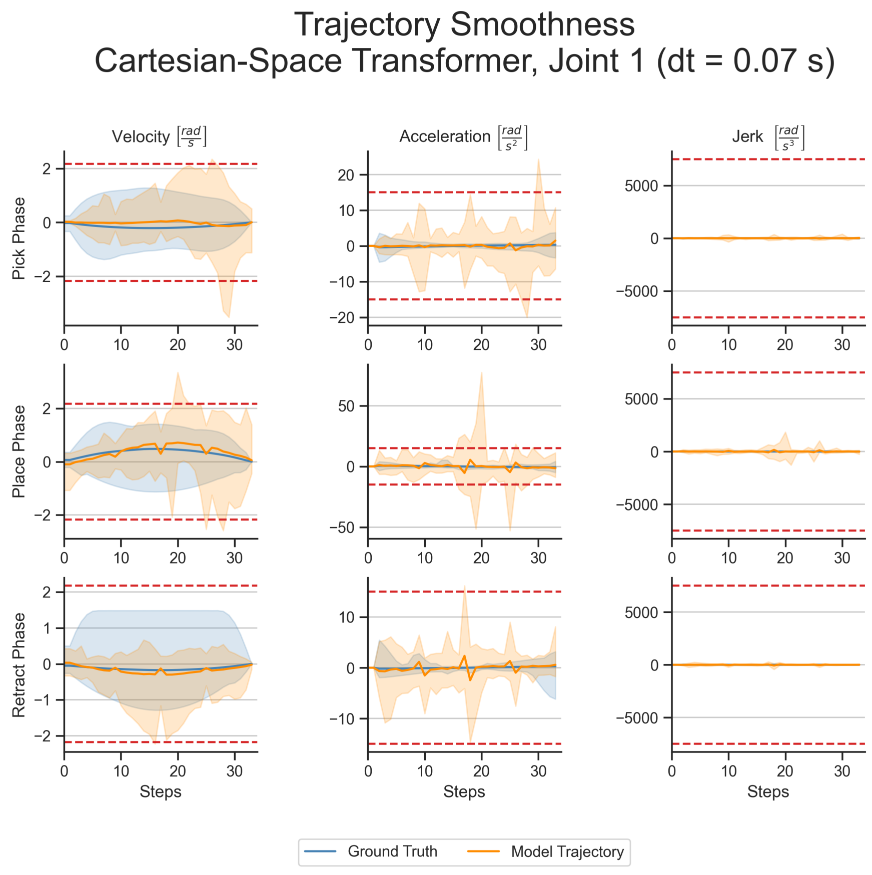


Figure A.29

A.3.3 Cartesian-Space Transformer

The plots A.29, A.30, A.31, A.32, A.33, A.34, and A.35 show the per joint values of the velocity, acceleration and jerk for the Cartesian-space transformer model.

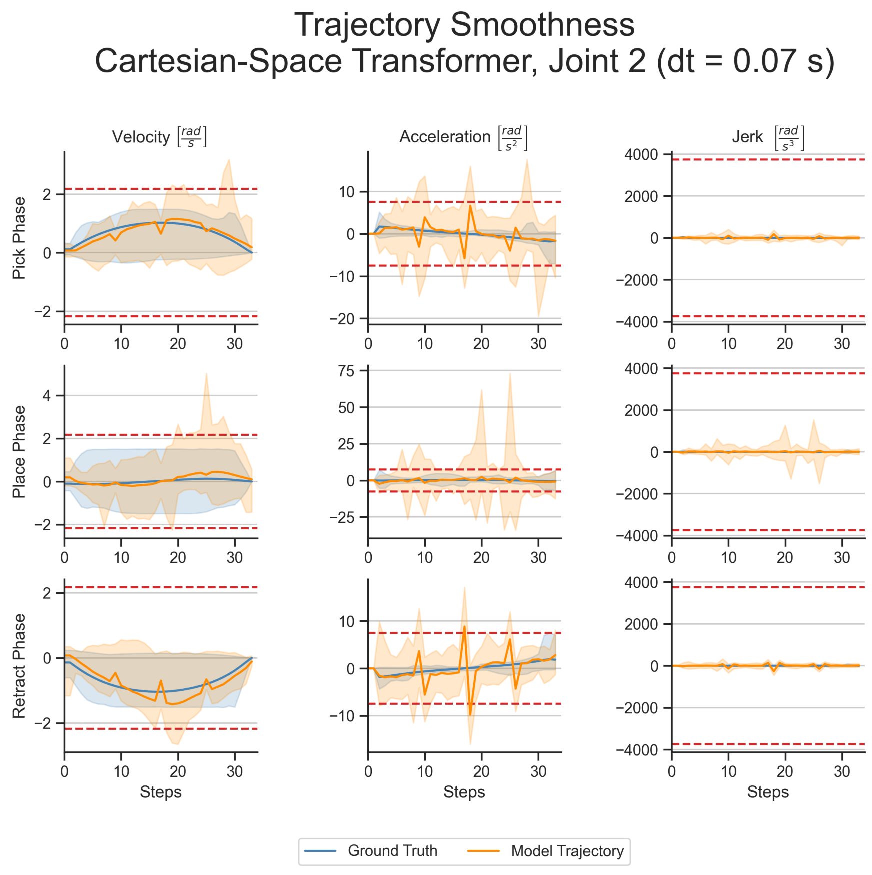

Trajectory Smoothness
Cartesian-Space Transformer, Joint 2 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.30

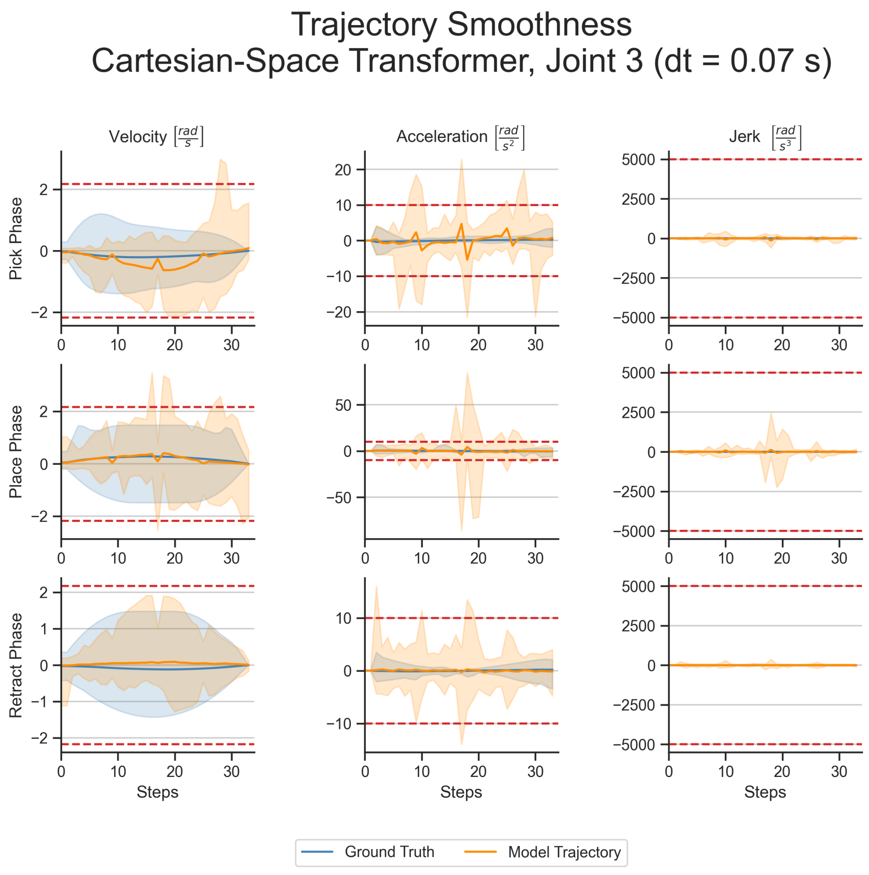

Trajectory Smoothness
Cartesian-Space Transformer, Joint 3 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.31

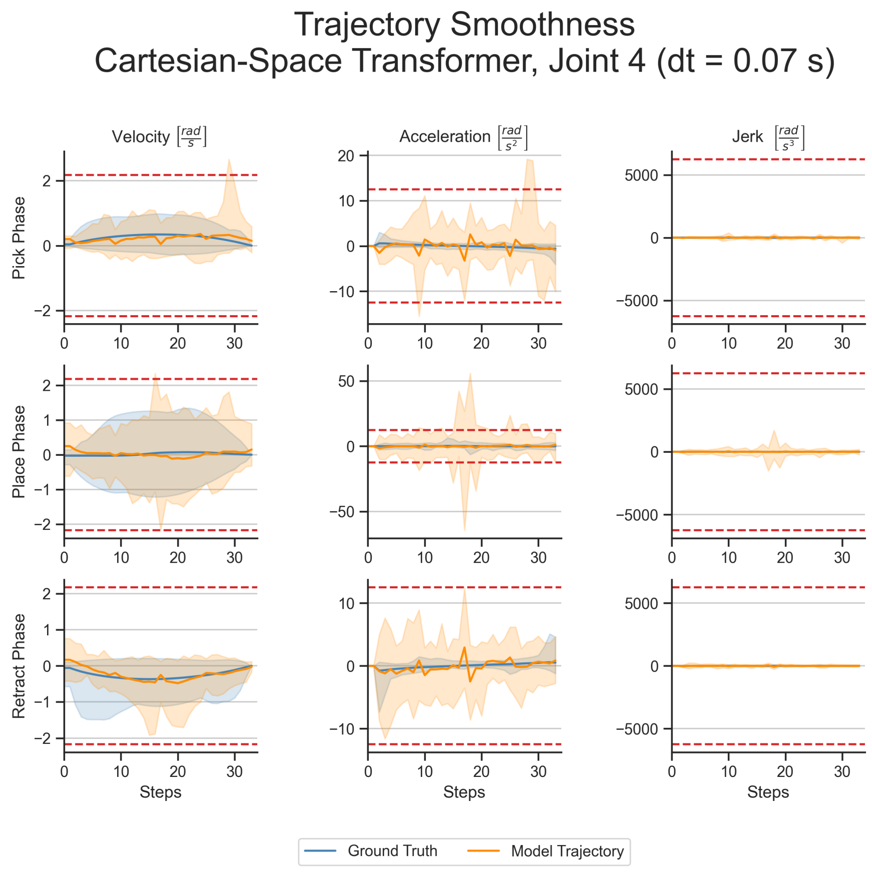

Trajectory Smoothness
Cartesian-Space Transformer, Joint 4 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.32

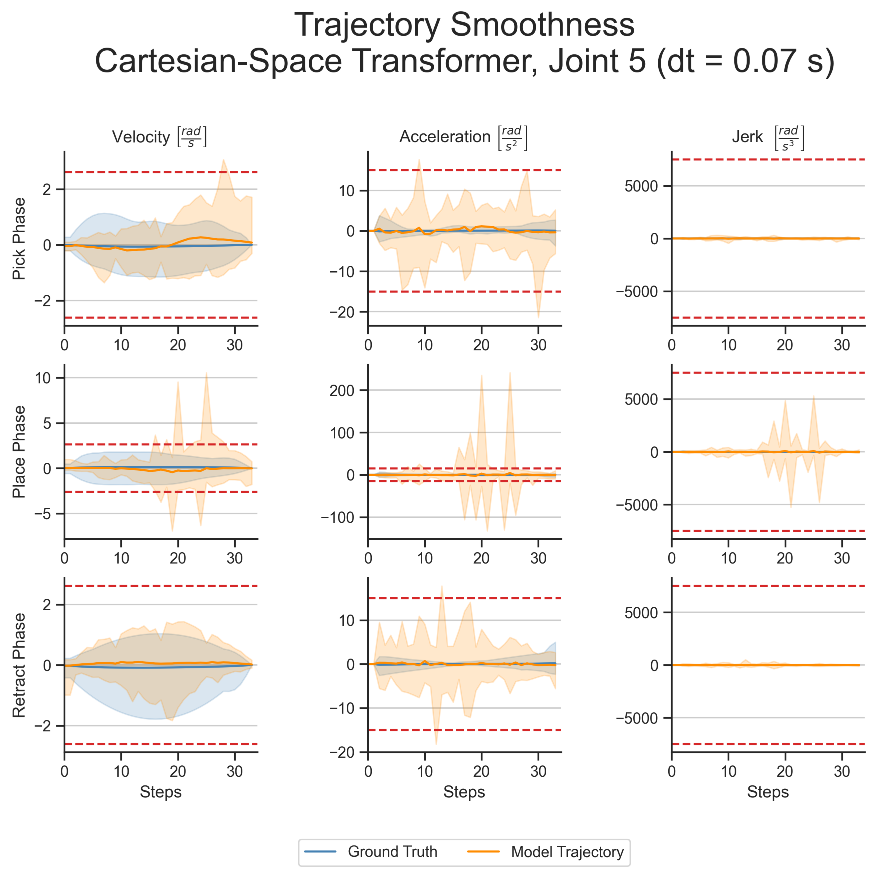

Trajectory Smoothness
Cartesian-Space Transformer, Joint 5 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.33

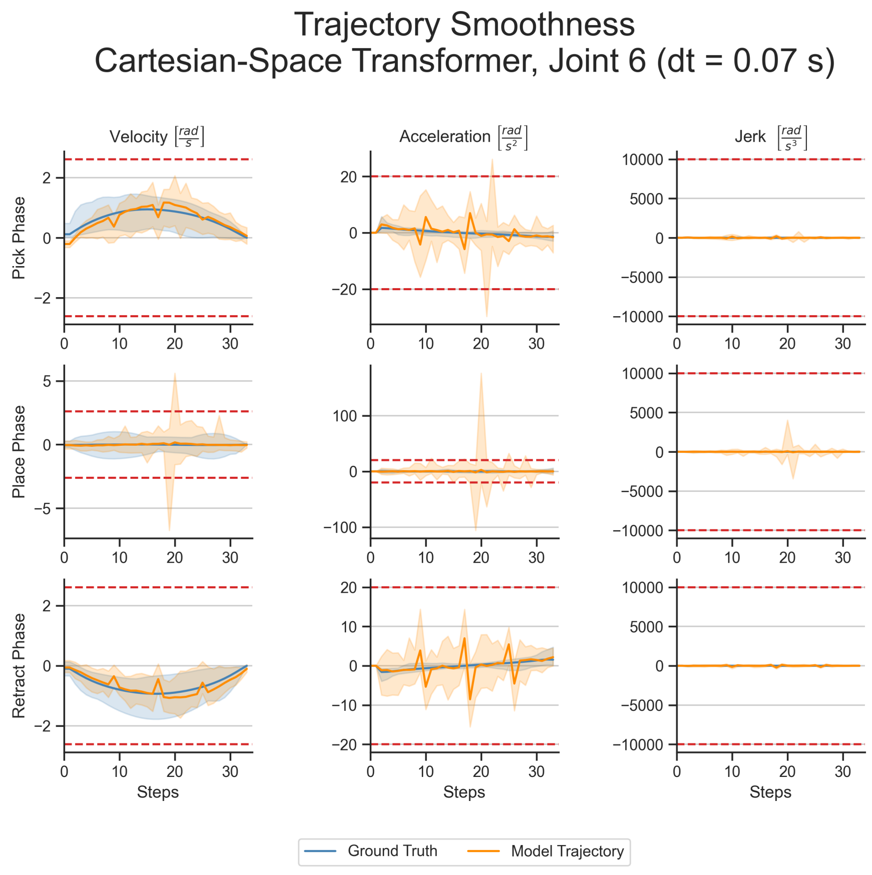

Trajectory Smoothness
Cartesian-Space Transformer, Joint 6 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.34

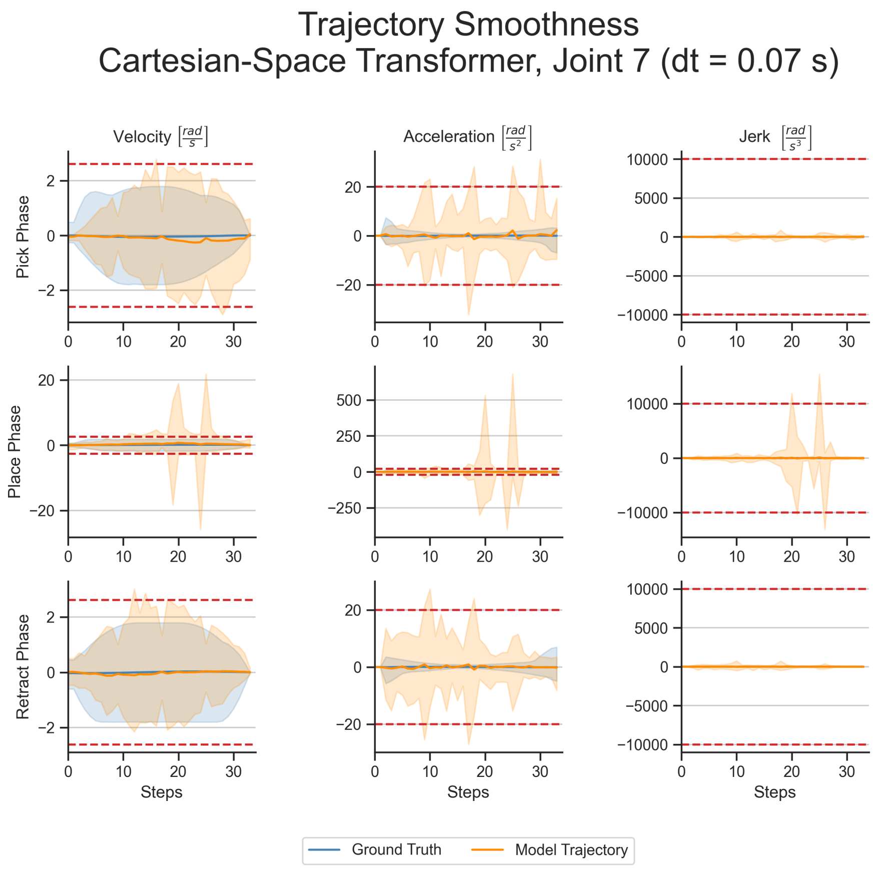

Trajectory Smoothness
Cartesian-Space Transformer, Joint 7 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.35

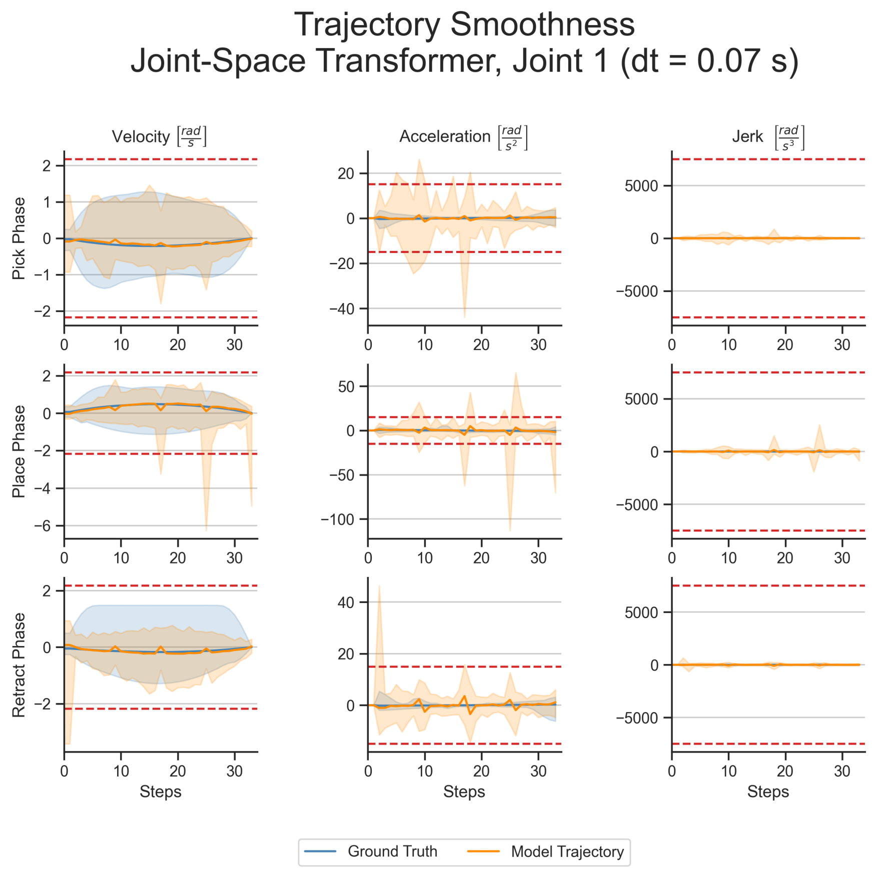
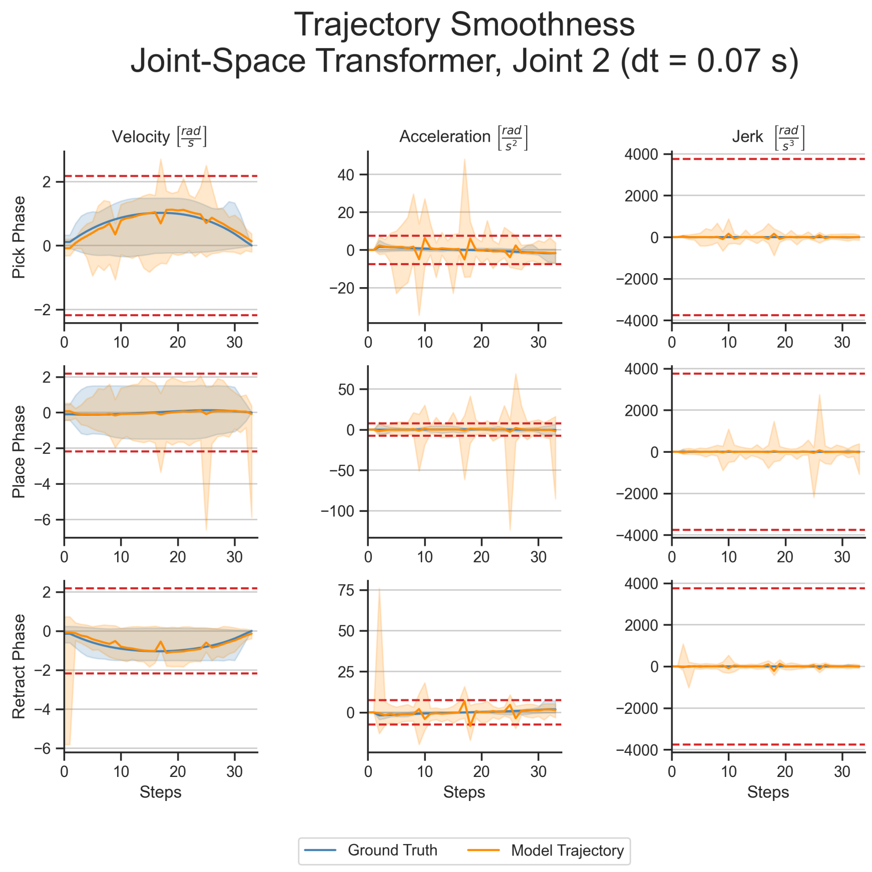


Figure A.36

A.3.4 Joint-Space Transformer

The plots A.36, A.37, A.38, A.39, A.40, A.41, and A.42 show the per joint values of the velocity, acceleration and jerk for the joint-space transformer model.



Trajectory Smoothness
Joint-Space Transformer, Joint 2 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.37

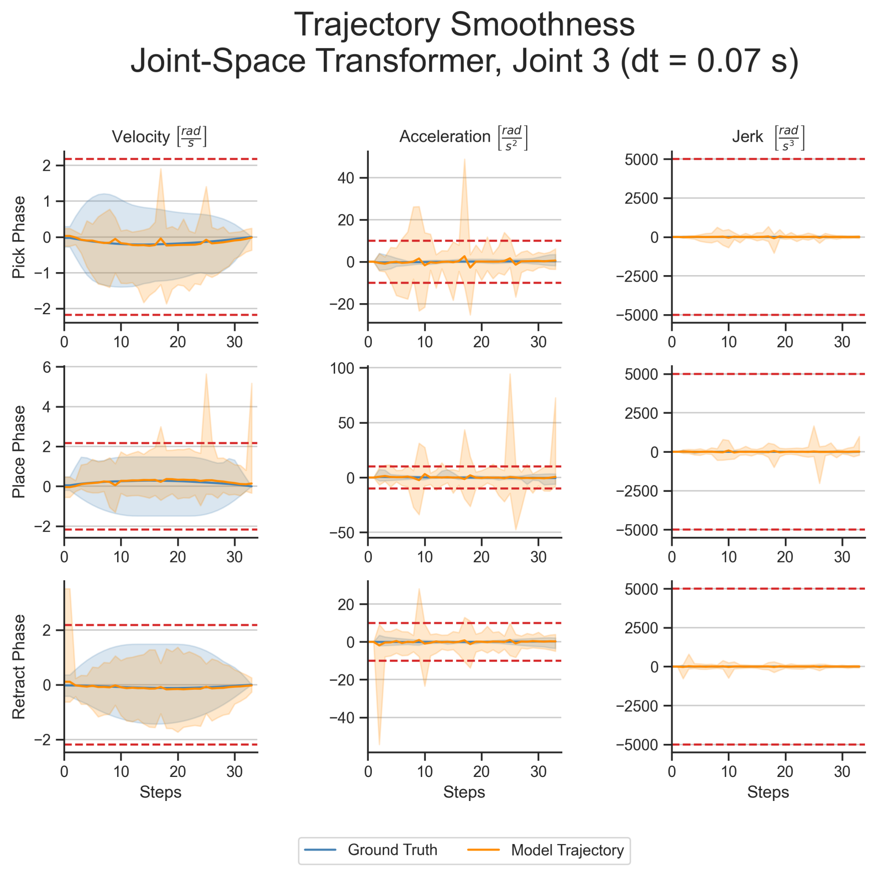
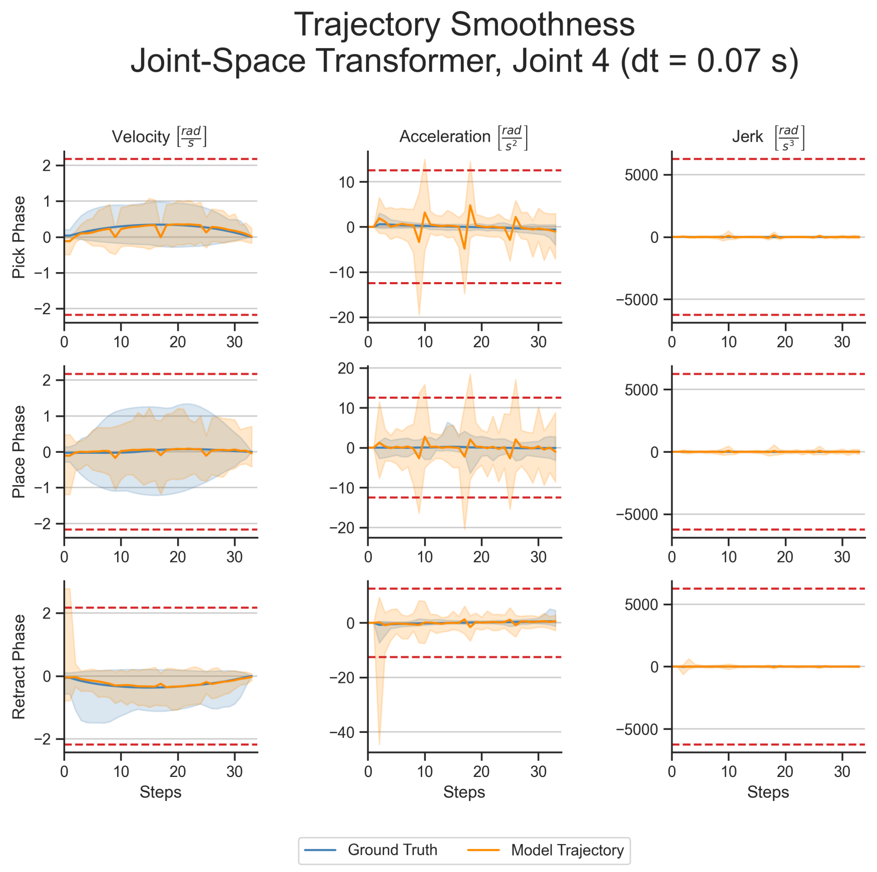

Trajectory Smoothness
Joint-Space Transformer, Joint 3 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.38



Trajectory Smoothness
Joint-Space Transformer, Joint 4 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.39

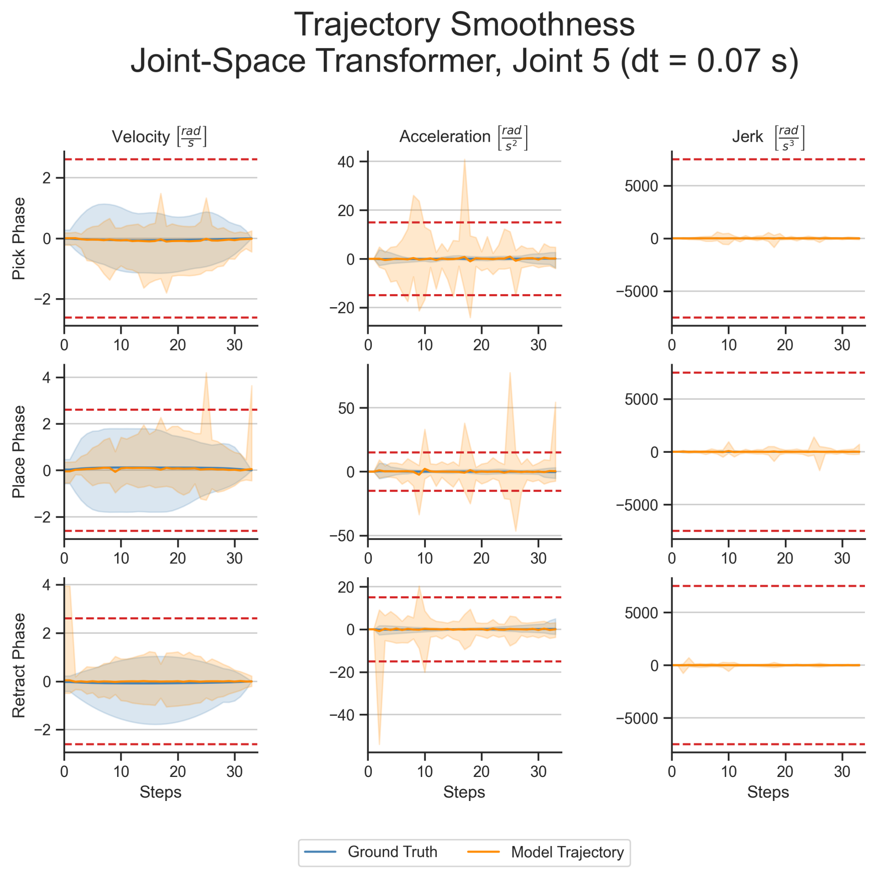
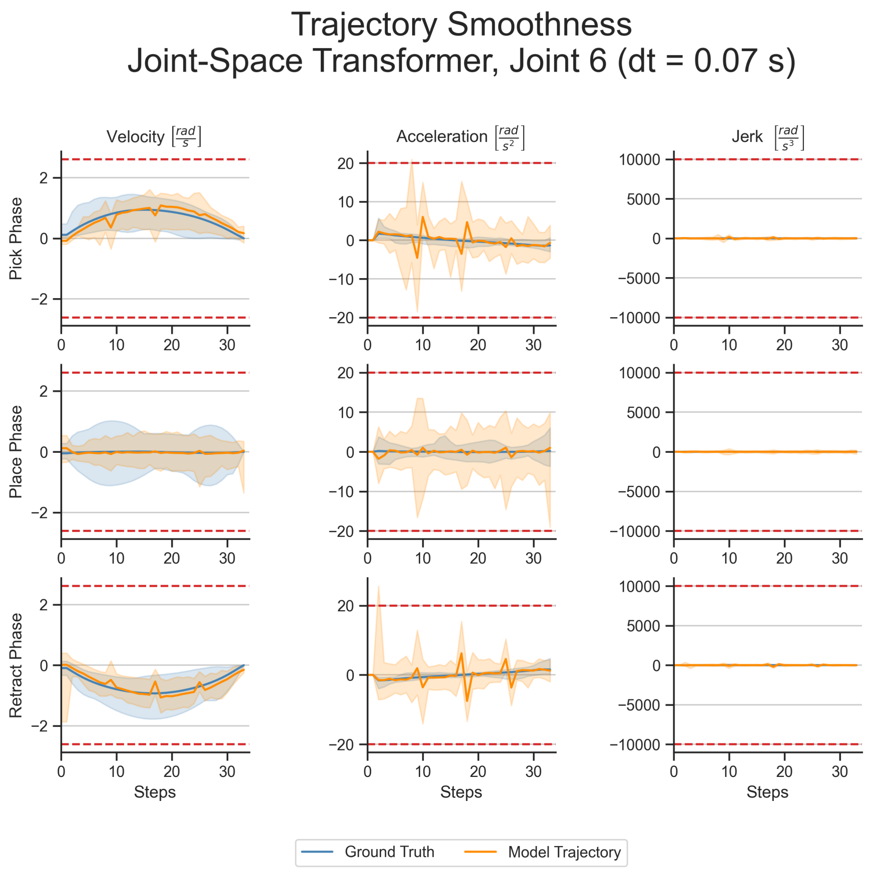

Trajectory Smoothness
Joint-Space Transformer, Joint 5 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.40



Trajectory Smoothness
Joint-Space Transformer, Joint 6 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.41

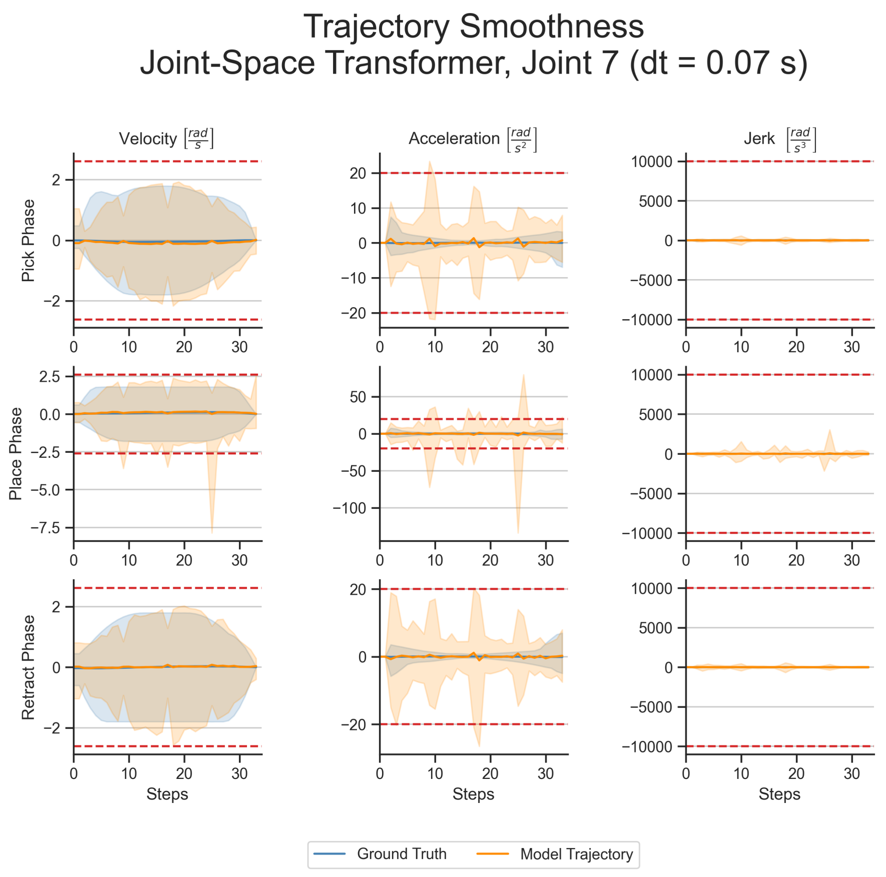

Trajectory Smoothness
Joint-Space Transformer, Joint 7 (dt = 0.07 s)
Velocity $\left[\frac{rad}{s}\right]$
Acceleration $\left[\frac{rad}{s^2}\right]$
Jerk $\left[\frac{rad}{s^3}\right]$
Pick Phase
Place Phase
Retract Phase
Steps
Ground Truth
Model Trajectory

Figure A.42