Task-Parameterized Transformer for Learning Gripper Trajectory from Demonstrations

by

Yinghan Chen

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2024

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis by articles contains one article for which I am the main author and another for which I am the co-author.

Article details: Task-Parameterized Transformer for Modelling Manipulation Trajectories. ICRA 2024. Yinghan Chen, Xueyang Yao, Bryan Tripp. The article was submitted to ICRA 2024 IEEE International Conference on Robotics and Automation.

Personal contributions: I am entirely responsible for the modifications made to the transformer model required for trajectory prediction, the training process, the implementation of the trajectory rotation data augmentation technique, and the collection and analysis of the results.

Article details: Improved Generalization of Probabilistic Movement Primitives for Manipulation Trajectories. Xueyang Yao, Yinghan Chen, Bryan Tripp

Personal contributions: I aided in the setup, collection, processing, labelling of the demonstration data set, and the application of the Dynamic Time Warping(DTW) algorithm for trajectory realignment.

The capture of task demonstrations, gripper control, and code for camera recording is done in collaboration with Xueyang Yao. Camera calibration, gripper tracking system setup, trajectory and image frame synchronization, and object position estimation are done entirely by Xueyang Yao.

Abstract

The goal of learning from demonstration or imitation learning is to teach the model to generalize across unseen tasks based on available demonstrations. This ability can be important for the stable performance of a robot in a chaotic environment such as a kitchen when compared to a more structured setting such as a factory assembly line. By leaving the task learning up to the algorithm, human teleoperators can dictate the action of robots without any programming knowledge and improve overall productivity in various settings. Due to the difficulty of manually collecting gripper trajectories in large qualities, successful application of learning from demonstrations would have to be able to learn from a sparse number of examples while still providing a high degree of predicted trajectory accuracy.

Inspired by the development of transformer models for large language model tasks such as sentence translation and text generation, I seek to modify the model for trajectory prediction. While there have been previous works that managed to train end-to-end models capable of taking images and contexts and then generating control output, those works rely on a massive quantity of demonstrations and detailed annotations. To facilitate the training process for a sparse number of demonstrations, we created a training pipeline that includes a DeeplabCut model for object position prediction, followed by the Task-Parameterized Transformer model for learning the demonstrated trajectories, and supplemented with data augmentations that allow the model to overcome the constraint of limited dataset. The resulting model is capable of outputting the predicted end effector gripper trajectory and pose at each time step with better accuracy than previous works in trajectory prediction.

Acknowledgements

I would like to thank all the people who made this thesis possible. I would like to thank my supervisor Bryan Tripp who offered valuable knowledge and insights toward the development of this research project. I also appreciate the help of my Ph.D. labmate Xueyang Yao for his hard work in helping to collect the demonstrations together and being a wonderful labmate to work with.

I would also like to thank my fellow lab-mates for making my stay at the lab enjoyable with board games and companionship.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the advancement of automation in recent years, robots have developed the potential to be used in a wide range of applications from everyday cleaning tasks to industrial-scale assembly lines. Nonetheless, these machines require a specific set of step-by-step instructions that they must follow to accomplish a task all the while accounting for any physical constraints and different variations of the same task. Even if an experienced programmer can write a custom script for a specific task when given enough time, this is often not feasible when the workers or the users lack the necessary programming knowledge nor is it straightforward to write for more complex interactions. To enable the average layman to be capable of teaching the robot without expert coding knowledge, there has to be an intuitive way to instruct the robot on the steps of a task. In addition, the robot must be capable of learning flexibly and adapting to previously unseen situations rather than memorizing the exact motions. The collection of randomly placed trash and the hammering of nails at different locations are some simple examples where generalization would benefit. The learning can be done through natural demonstrations by directly operating the robot and then training a generalized model of the motions based on those demonstrations [6][13][20][24].

With the introduction of machine learning models, robots can be trained to a high degree of accuracy while allowing the average worker without any programming knowledge to provide them with natural demonstrations and context[16]. These robots are then able to replicate these actions, freeing up human labour in various settings and improving overall productivity. While some of the current work in machine learning has produced promising models that perform well across many different tasks, these models often rely on a massive collection of simulated or real-world training sets in the hundreds of thousands of demonstrations due to their end-to-end nature, converting from multiple modalities such

as images and languages directly to control output[16]. Particularly for real-world data sets, their collections can be very complicated and labour-intensive depending on the tasks involved and can be tedious to collect in large numbers. Our work hopes to provide a lightweight alternative capable of learning multiple tasks similar to the larger models while requiring significantly fewer task demonstrations.

Chapter 2 will describe the previous works for learning trajectories based on task demonstrations, some of which inspired the modifications made to our own model. Chapter 3 will then list and explain our data collection setup, the tools used in the demonstrations, the data preprocessing methods before the model prediction, and finally our modification to the original transformer model to allow it to predict trajectories. Chapter 4 then performs an analysis and comparison of the different models' performance.

1.1 Motivation

I hypothesize that previous transformer models' large data set requirement resulted from the need to perform image pose detection or interpret context from languages rather than strictly trajectory prediction. Hence our goal here is to develop a novel alternative pipeline and trajectory prediction model that depends on very few demonstrations and can learn multiple tasks simultaneously. Furthermore by reducing the size of the model in terms of the parameters, the model can be more easily deployed on older and cheaper hardware. Doing so will allow a wider range of users without specialized, expensive hardware or expert programming knowledge to set up customized models for their automated tasks and improve their productivity.

1.2 Contribution

A summary of the contributions that this thesis made to the field of deep learning and robotics is as follows:

- Modified the transformer model originally used for language translation tasks to be capable of predicting gripper positional trajectories and orientations given a scene's object configuration. This is done through providing information such as task and object identifiers in addition to their poses from the environment as input to the model, allowing the model to adapt to the various arrangements.

- Introduced a novel data augmentation for demonstration trajectory inspired by image augmentation for Convolutional Neural Networks (CNN). This allows for improved generalization across previously unseen object configurations and improves the overall model accuracy.
- The collection and marking of the pick-and-place, water pouring, and hockey shooting demonstrations used for the training and testing of the TP-Transformer model.

Finally, in the conclusion I provided further instructions on how to potentially implement other common features such as via-points or force-based feedback control in real-time on top of the TP-Transformer model, but these were not added due to the time constraint.

Chapter 2

Background

2.1 Movement Primitives

In motor control, movement primitives or motor primitives are related to the muscle activity involved in motion and make up the structure of complex movements such as grasping and pouring. By breaking down the demonstrated motion into atomic primitives and understanding them, these elements can be recomposed into new higher-level motor skills [11]. Furthermore, the primitives can be defined at different levels that consider the kinematic elements, purely motion-based, or kinetic elements, involving force and muscle control [11]. Additional capability of the movement primitives includes the modulation of motion speed, following via-points, periodic motions, and extrapolating new configurations based on previous demonstrations. Below we provide a background of the popular movement primitive models that inspired our current work.

2.1.1 Dynamic Movement Primitive

Inspired by biological systems and their complex motor controls, dynamic movement primitive (DMP) is a non-linear dynamical system and can be used to generate complex motor commands for robotic systems [24]. In addition to the guaranteed convergence to the target, the algorithm can adapt to changes in the environment in real-time [24]. As there are many variations of the DMP algorithm such as Orientation and Periodic DMP, the one described next will be the base discrete DMP model encoding discrete point-to point motions [24]. The components of the classic discrete DMP include a second-order dynamical

system where the first term of the equation is an attractor term pushing the current trajectory position y towards the goal g. α_z and β_z are the learning parameters controlling the attractor's behaviour. The z term is the first-order derivative of y times τ or the current velocity. The second term has x as the phase variable for the stage of the trajectory and f as a function of the linear weighted basis function ϕ that allows for a complex trajectory as can be seen in Equation 2.2. Alternatively, the basis function can be modified to be sinusoidal to allow for rhythmic motion such as swiping. The number of weights N allows the resolution of the trajectory to be fine-tuned according to task complexity. Finally, the τ term in all the equations modulates the time or the speed at which the phases change and the corresponding speed of the trajectory motion [24].

To learn the weight values, the example demonstration's trajectory and its corresponding derivatives with respect to the time are needed. Then Equation 2.1 is rearranged to approximate the weights in $f(x)$. The weight learning process can be done through Locally Weighted Regression in which the $f(x)$ is broken down into a basis matrix and weight vector component, where the weight values are updated iteratively [24].

$$\begin{aligned} \tau\dot{z} &= \alpha_z(\beta_z(g-y)-z)+f(x) \\ \tau\dot{y} &= z \\ \tau\dot{x} &= \alpha_x x \end{aligned} \tag{2.1}$$

$$f(x) = \frac{\sum_{i=1}^{N} w_i\phi(x)}{\sum_{i=1}^{N}\phi(x)} \tag{2.2}$$

2.1.2 Task-Parameterized Gaussian Mixture Model

To flexibly adapt to different object positions, the task-parameterized Gaussian mixture model (TP-GMM) [6] models the local trajectory relative to the objects within the scene. The task parameter in this work refers to the configuration of the objects or the state of the environment. In this case for the jth out of of all P reference frames at time t, $\boldsymbol{b}_{t,j}$ represents the center of the object and $\boldsymbol{A}_{t,j}$ is its corresponding transformation matrix. Beginning with the global trajectory data represented by $\boldsymbol{\xi}_t$, it is converted into the corresponding relative trajectory $\boldsymbol{X}_t^{(j)}$ in the j-th local reference frame using the transformation Equation 2.3. A k-Component Gaussian mixture model of the local reference trajectory $\boldsymbol{X}_t^{(j)}$ for the j-th reference frame can be learned by fitting the mixing coefficient π_i, the mean $\boldsymbol{\mu}_i^{(j)}$,

and the covariance matrix $\mathbf{\Sigma}_i^{(j)}$ for all components $i = 1, 2, ..., k$ through the expectation-maximization algorithm [6].

The resulting normal distribution from each reference frame containing the variability and correlation from multiple demonstrations, $\hat{\mathbf{\Sigma}}_{t,i}^{(j)}$, at the estimated trajectory point $\hat{\boldsymbol{\xi}}_t^{(j)}$ can then be combined as a product of Gaussian as shown in the Equation 2.4 [6]. The certainty of the trajectory from a particular local reference frame is reflected by the variability of that distribution's covariance matrix. The final probabilistic encoding of the trajectory at time t for component i, $N(\hat{\boldsymbol{\xi}}_{t,i}, \hat{\mathbf{\Sigma}}_{t,i})$, from multiple coordinate systems is most influenced by the local frame with the highest trajectory certainty. As a result, the work's use of multiple local reference frames can adapt to unseen configurations of the task parameters by aggregating multiple predictions and guaranteeing that trajectories are accurately following segments that have highest level of certainty. For example, demonstrations for tightening nut and bolt would consistently see the gripper approaching and staying near the local reference of the nut for the final segment of the motion trajectory.

With algorithmic runtime independent of the number of demonstrations, Gaussian mixture regression of the trajectory can then be conditioned on the continuous timestamp t which gives the distribution shown in 2.5. This is done through modelling the joint distribution of input dimension (i.e. time or the state of environment for $\boldsymbol{\xi}_t^I$) and the output dimensions (i.e. position, velocity, joint angles, etc. for $\boldsymbol{\xi}_t^O$) with K-component GMM [6].

$$\boldsymbol{X}_t^{(j)} = \boldsymbol{A}_{t,j}^{-1}(\boldsymbol{\xi}_t - \boldsymbol{b}_{t,j}) \tag{2.3}$$

$$\begin{aligned}
N(\hat{\boldsymbol{\xi}}_{t,i}, \hat{\mathbf{\Sigma}}_{t,i}) &= \prod_{j=1}^{P} N(\hat{\boldsymbol{\mu}}_{t,i}^{(j)}, \hat{\mathbf{\Sigma}}_{t,i}^{(j)}) \\
\hat{\mathbf{\Sigma}}_{t,i} &= \Big(\sum_{j=1}^{P} \hat{\mathbf{\Sigma}}_{t,i}^{(j)-1}\Big)^{-1} \\
\hat{\boldsymbol{\xi}}_{t,i} &= \hat{\mathbf{\Sigma}}_{t,i} \sum_{j=1}^{P} \hat{\mathbf{\Sigma}}_{t,i}^{(j)-1} \hat{\boldsymbol{\mu}}_{t,i}^{(j)} \\
\hat{\mathbf{\Sigma}}_{t,i}^{(j)} &= \boldsymbol{A}_{t,j} \mathbf{\Sigma}_i^{(j)} \boldsymbol{A}_{t,j}^{\intercal} \\
\hat{\boldsymbol{\mu}}_{t,i}^{(j)} &= \boldsymbol{A}_{t,j} \boldsymbol{\mu}_i^{(j)} + \boldsymbol{b}_{t,j}
\end{aligned} \tag{2.4}$$

$$
\begin{aligned}
P_r(\boldsymbol{\xi}_t^O|\boldsymbol{\xi}_t^I) &\sim \sum_{i=1}^{K} h_i(\boldsymbol{\xi}_t^I) N(\hat{\boldsymbol{\mu}}_i^O(\boldsymbol{\xi}_t^I), \hat{\boldsymbol{\Sigma}}_i^O) \\
\hat{\boldsymbol{\mu}}_i^O(\boldsymbol{\xi}_t^I) &= \boldsymbol{\mu}_i^O + \hat{\boldsymbol{\Sigma}}_i^{OI} \hat{\boldsymbol{\Sigma}}_i^{I\,-1} (\boldsymbol{\xi}_t^I - \boldsymbol{\mu}_i^I) \\
\hat{\boldsymbol{\Sigma}}_i^O &= \hat{\boldsymbol{\Sigma}}_i^O - \boldsymbol{\Sigma}_i^{OI} \boldsymbol{\Sigma}_i^{I\,-1} \boldsymbol{\Sigma}_i^{IO} \\
h_i(\boldsymbol{\xi}_t^I) &= \frac{\pi_i N(\boldsymbol{\mu}_i^I, \boldsymbol{\Sigma}_i^I)}{\sum_{j=1}^{K} \pi_j N(\boldsymbol{\mu}_j^I, \boldsymbol{\Sigma}_j^I)}
\end{aligned}
\tag{2.5}
$$

2.1.3 Probabilistic Movement Primitives

Similar to the TP-GMM model, the Probabilistic Movement Primitives (ProMP) model also aims to provide a probabilistic way of learning from demonstrations and sampling new trajectories [20]. The trajectory model represents the demonstrations as a multivariate normal distribution from which compact representations of the learned trajectory or weights $\boldsymbol{w}$ can be sampled from. The distribution of the weight $\boldsymbol{w}$ is modelled by the Gaussian distribution $p(\boldsymbol{w}|\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ consist of the learned parameters mean vector, $\boldsymbol{\mu}_w$, and the covariance matrix, $\boldsymbol{\Sigma}_w$ [20]. The weight is then multiplied by a matrix of time-dependent or phase-based Gaussian basis functions $\boldsymbol{\Phi}_t$ as seen in 2.6, where $\boldsymbol{\phi}_i(t)$ is the k-dimension time-dependent basis function for each of the D output dimensions, to generate the mean of the trajectories and its corresponding time-varying covariance matrix [20]. The modelling of the covariance allows the model to better learn trajectories for demonstrations with a high level of variation. The sampled trajectory is represented by $\boldsymbol{\tau} = \{\boldsymbol{y_t}\}_{t=0,\ldots,T}$ conditioned on the weight for a total of T time steps and its probability function is shown in Equation 2.7. Finally, the conditional probability for $\boldsymbol{\tau}$ given the learned task parameters or the likelihood of the trajectory is shown in Equation 2.8.

In addition to modelling the position and velocity of the gripper, the model is also capable of modelling the periodic/rhythmic motions by including separate Von-Mises basis functions, modulating via-points, adjusting velocity, and blending movement primitives [20]. The coupling between the different degrees of the joints can also be accounted for by adding or removing the non-diagonal basis functions in the $\boldsymbol{\Phi}$ matrix [20]. this can be done at the cost of requiring significantly more demonstrations for learning the parameters.

$$\boldsymbol{\Phi}_t = \begin{bmatrix} \boldsymbol{\phi}_1(t) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \boldsymbol{\phi}_D(t) \end{bmatrix} \tag{2.6}$$

$$p(\boldsymbol{\tau}|\boldsymbol{w}) = \prod_{t=0}^{T} N(\boldsymbol{y_t}|\boldsymbol{\Phi}_t^\intercal \boldsymbol{w}, \boldsymbol{\Sigma}_y) \qquad \boldsymbol{y_t} = \boldsymbol{\Phi}_t^\intercal \boldsymbol{w} + \epsilon_y \tag{2.7}$$

$$p(\boldsymbol{\tau}|\boldsymbol{\theta}) = \int p(\boldsymbol{\tau}|\boldsymbol{w}) p(\boldsymbol{w}|\boldsymbol{\theta}) d\boldsymbol{w} \tag{2.8}$$

2.1.4 Kernel Movement Primitives

The Kernel Movement Primitives (KMP) method attempts to combine the work of ProMP and TP-GMM by replacing the predefined basis functions with kernel functions [13]. The local-KMP version of the algorithm is capable of learning trajectories in the local reference frames in the manner of TP-GMM through the use of the coordinate transformation represented by $\boldsymbol{A}^{(p)}$ and $\boldsymbol{b}^{(p)}$ for reference frame p. These modifications allow the model to leverage the extrapolation capability while providing an improvement over the ProMP algorithm.

The major additions over the previous works include [13]:

- Learning a probabilistic representation based on multiple demonstrations.
- Adoptions and superposition of multiple trajectories.
- Generalizing to extrapolated trajectories.
- Better performance when given high dimensional inputs by replacing basis functions with kernel functions.

The algorithm minimizes the KL-Divergence objective function between the trajectory distribution given the weight distribution from ProMP, $P_p(\boldsymbol{\xi}|\boldsymbol{s}_n)$, and the trajectory distribution based on the Gaussian Mixture Regression, $P_r(\boldsymbol{\xi}|\boldsymbol{s}_n)$, conditioned on the $\boldsymbol{s}_n$ input, typically time. This is shown in Equation 2.9 where n is the order in the trajectory, $\boldsymbol{\xi}$ is the

output trajectory, and $\boldsymbol{\Phi}_n = \boldsymbol{\Phi}(s_n)$ is the same basis function matrix as described in the Probabilistic Movement Primitives subsection. The conditional distribution equations are from the ProMP and TP-GMM algorithm with time-driven input as shown in Equation 2.10.

$$D_{KL}(P_p(\boldsymbol{\xi}|\boldsymbol{s}_n)||P_r(\boldsymbol{\xi}|\boldsymbol{s}_n)) = \int P_p(\boldsymbol{\xi}|\boldsymbol{s}_n) \cdot log(\frac{P_p(\boldsymbol{\xi}|\boldsymbol{s}_n)}{P_r(\boldsymbol{\xi}|\boldsymbol{s}_n)})\delta\boldsymbol{\xi} \tag{2.9}$$

$$\begin{aligned} P_r(\boldsymbol{\xi}|\boldsymbol{s}_n) &= N(\boldsymbol{\xi}|\hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n) \\ P_p(\boldsymbol{\xi}|\boldsymbol{s}_n) &= N(\boldsymbol{\xi}|\boldsymbol{\Phi}_n^\intercal \boldsymbol{\mu}_w, \boldsymbol{\Phi}_n^\intercal \boldsymbol{\Sigma}_w \boldsymbol{\Phi}_n) \end{aligned} \tag{2.10}$$

Then by using the property of two Gaussian distributions under KL-divergence, the resulting objective function can be broken down into mean and covariance minimization problems which optimizes the mean $\boldsymbol{\mu}_w$ and $\boldsymbol{\Sigma}_w$ of the parametric trajectory distribution [13]. The scalar $\lambda > 0$ is introduced to regulate the penalty term $||\boldsymbol{\mu}_w||^2$ in the mean minimization problem 2.10. In short, the optimal parameters can be derived through Equation 2.11. The two separate equations can then have their original basis functions replaced with kernel functions by defining the kernel as shown in Equation 2.12. Lastly to generate the trajectory given the input s^*, the expected output, $\mathbb{E}(\boldsymbol{\xi}(s^*))$, is calculated from $\boldsymbol{\Phi}(s^*)^\intercal \boldsymbol{\mu}_w^*$ and its corresponding covariance, $\mathbb{D}(\boldsymbol{\xi}(s^*))$, is calculated from $\boldsymbol{\Phi}(s^*)^\intercal \boldsymbol{\Sigma}_w^* \boldsymbol{\Phi}(s^*)$ [13]. The final equations of the expected mean and covariance after replacing the basis functions with the kernel functions are shown in Equation 2.14.

$$\begin{aligned} \boldsymbol{\mu}_w^* &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi} + \lambda \boldsymbol{\Sigma})^{-1} \boldsymbol{\mu} \\ \boldsymbol{\Sigma}_w^* &= N(\boldsymbol{\Phi} \boldsymbol{\Sigma}^{-1} \boldsymbol{\Phi}^{*\intercal} + \lambda \boldsymbol{I})^{-1} \end{aligned} \tag{2.11}$$

$$\begin{aligned} \boldsymbol{k}(s_i, s_j) &= \boldsymbol{\phi}(s_i)^\intercal \boldsymbol{\phi}(s_j) \\ \boldsymbol{K} &= \begin{bmatrix} \boldsymbol{k}(s_1, s_1) & \dots & \boldsymbol{k}(s_1, s_N) \\ \vdots & \ddots & \vdots \\ \boldsymbol{k}(s_N, s_1) & \dots & \boldsymbol{k}(s_N, s_N) \end{bmatrix} \end{aligned} \tag{2.12}$$

$$\begin{aligned} \boldsymbol{\phi} &= [\boldsymbol{\phi}(s_1), ...\boldsymbol{\phi}(s_N)] \\ \boldsymbol{\Sigma} &= blockdiag(\hat{\boldsymbol{\Sigma}}_1, ...\hat{\boldsymbol{\Sigma}}_N) \\ \boldsymbol{\mu} &= [\hat{\boldsymbol{\mu}}^\intercal, ...\hat{\boldsymbol{\mu}}^\intercal]^\intercal \\ \boldsymbol{k}^* &= [\boldsymbol{k}(s^*, s_1), ...\boldsymbol{k}(s^*, s_N)] \end{aligned} \tag{2.13}$$

$$\begin{aligned}\mathbb{E}(\boldsymbol{\xi}(s^*)) &= \boldsymbol{k}^*(\boldsymbol{K}+\lambda\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu} \\ \mathbb{D}(\boldsymbol{\xi}(s^*)) &= \frac{\boldsymbol{N}}{\lambda}(\boldsymbol{k}(s^*,s^*) - \boldsymbol{k}^*(\boldsymbol{K}+\lambda\boldsymbol{\Sigma})^{-1}\boldsymbol{k}^{*\intercal}\end{aligned} \tag{2.14}$$

2.1.5 Task-Parameterized Probabilistic Movement Primitive

The task-parameterized probabilistic movement primitive (TP-ProMP) model makes use of the same local reference frames from TP-GMM and expands upon the ProMP model [33]. In comparison to KMP, TP-ProMP uses basis function rather than kernel functions and Maximum A-Posteriori to estimate the parameters rather than KL-Divergence. One of the problems that TP-ProMP addresses is mode collapse where multiple variations of accomplishing the same task with identical task parameters lead to the model generating poor trajectories. For example, if there is the option of pouring from the left or right side of a cup, the model trained on such demonstrations would generate the mean of the demonstrated trajectory even if it is not a valid option. For training and testing, the TP-ProMP uses a set of common household tasks and actions such as pick-and-placing teabag, pouring water, shooting puck into a net, and sweeping trash [33].

Now we will summarize the structure of the TP-ProMP model in this section. As with TP-GMM, The starting pose and position of the objects are represented as $\boldsymbol{A}_j$ and $\boldsymbol{b}_j$ respectively and are also used for converting between the object reference frame and the global reference frame. Based on the concatenated trajectories, shown in Equation 2.15, from each local reference frame $\boldsymbol{\tau}' \in \mathbb{R}^{DR\times T}$ where the D is the number of dimensions being modelled, R local reference frames, and T time steps, a ProMP model is fitted [33]. Given $M = DR$, the ProMP learns a weight distribution for $\boldsymbol{w}$ with the weight parameters $\boldsymbol{\mu}_w$ for the mean weight, $\boldsymbol{\Sigma}_w$ for the weight covariance, and $\boldsymbol{\Sigma}_\epsilon$ for the trajectory covariance which can then be used to replicate $\boldsymbol{\tau}'$ in combination with the M-block basis function matrix $\boldsymbol{\Phi_t}$ from Equation 2.7. This gives the trajectory probability conditioned on learned weight parameters as can be seen in Equation 2.16 where the trajectory mean $\boldsymbol{\mu}_{\chi_t}$ and covariance $\boldsymbol{\Sigma}_{\chi_t}$ at time t can be split by local reference frame. Combining the Gaussian product of the local reference trajectory estimates as in TP-GMM yields the optimal trajectory points and corresponding variance.

To deal with mode collapse, TP-ProMP must also be able to differentiate the variations in weight values $\boldsymbol{w}$ for their corresponding trajectory. This is done by breaking the model training down into two steps. The first step involves clustering the N demonstrations into C groups with a Gaussian Mixture and learning the distribution parameters through

expectation-maximization [33]. The second step takes the clustered demonstrations and separately trains an individual model for each variation using the previously learned cluster parameters, i.e. $\boldsymbol{\mu}_w$, $\boldsymbol{\Sigma}_w$, and $\boldsymbol{\Sigma}_\epsilon$, as the initial parameters to aid in further training [33]. These individual models can then be separately sampled according to the mode of the action.

The work also introduces the paired-object reference frame in which local reference frames are centred at the source object but the local x-axis is pointed toward a target object [33]. This is useful for situations where directed motion is required for a task, for example in a puck shooting task where the puck must be launched along the local x-axis relative to the net. These modifications allow the TP-ProMP model to tackle a variety of tasks at the cost of additional computation time.

$$\begin{aligned} \boldsymbol{\tau}' &= \{\boldsymbol{X}_t\}_{t=1}^{T} \\ \boldsymbol{X}_t &= (x_{1,t}^{(1)}, ..., x_{D,t}^{(1)}, ..., x_{1,t}^{(R)}, ..., x_{D,t}^{(R)}) \end{aligned} \tag{2.15}$$

$$\begin{aligned} p(\boldsymbol{X}_t|\boldsymbol{w}) &= \int N(\boldsymbol{X}_t|\boldsymbol{\Phi}_t^\intercal \boldsymbol{w}, \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}) N(\boldsymbol{w}|\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w) d\boldsymbol{w} \\ &= N(\boldsymbol{X}_t|\boldsymbol{\mu}_{\chi_t}, \boldsymbol{\Sigma}_{\chi_t}) \\ \boldsymbol{\mu}_{\chi_t} &= \boldsymbol{\Phi}_t^\intercal \boldsymbol{\mu}_w \\ \boldsymbol{\Sigma}_{\chi_t} &= \boldsymbol{\Phi}_t^\intercal \boldsymbol{\Sigma}_w \boldsymbol{\Phi}_t + \boldsymbol{\Sigma}_{\boldsymbol{\epsilon}} \end{aligned} \tag{2.16}$$

$$\begin{aligned} \boldsymbol{\mu}_{\chi_t} &= [\boldsymbol{\mu}_{\chi t}^{(1)}, ..., \boldsymbol{\mu}_{\chi t}^{(R)}] \\ \boldsymbol{\Sigma}_{\chi_t} &= \begin{bmatrix} \boldsymbol{\Sigma}_{\chi t}^{(1)} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \boldsymbol{\Sigma}_{\chi t}^{(R)} \end{bmatrix} \end{aligned} \tag{2.17}$$

2.2 Deep Learning

Deep learning has shown broad applicability in fields ranging from image recognition to answering complex inquiries. With modifications to model structures and data pre-processing, these algorithms have demonstrated the incredible ability to successfully perform various

narrow tasks much more proficiently than their human counterparts such as in the case of playing Go, an ancient board game[25], and video games. Yet despite their super-human ability, their internal working remains mostly a mystery and are prone to adversarial attacks [32]. This part will explore the use of deep learning algorithms in robotic-related tasks and describe their advantages and shortcomings.

The most basic blocks of deep learning models generally consist of neural network layers of learned parameters. The basic fully connected (FC) layer or multi-layer perceptron (MLP) can be represented as a matrix of weights where each of the output neurons is connected to each of the input neurons by a row of learnable weights. To mimic the working of the biological neural process, non-linear activation functions are added between the layers to determine whether parts of the input are important enough to be "fired" or passed down to the next layer, leading to the learning of complex relationships between various features. Different models such as the Convolutional Neural Network (CNN) [15] commonly applied to image tasks mainly differ in the type of connections between the different neurons of each layer. In general, the deeper the layers of the model the better it is at learning and processing a higher level of abstraction.

The actual learning of parameters in deep learning is achieved through the application of the chain rule and the back-propagation of errors [22] and can be done in multiple ways. Supervised learning maps out the relationship between the input and the output values and learns using the error between the ground truth and the model prediction. This method requires the target value of the data set to be labelled or collected beforehand and is useful for predicting categories or values that may otherwise be difficult to determine. A method commonly used in robotics, reinforcement learning relies on a feedback reward system and environmental observations to find the optimal action required to maximize the expected reward [27]. The reward can be whether the agent accomplishes a task while following the physical constraints. While this may make reinforcement learning appealing for robotic tasks, it is often not straightforward to define the reward function for complex tasks and requires extensive testing and training in the real world.

2.2.1 Transformers

Originally applied to learning language tasks such as translations and sentence completion, the transformer model has recently seen its application to general robotic control tasks [29]. The transformer model is an improvement over previous state-of-the-art Recurrent Neural Network (RNN) [23] and long short-term memory (LSTM) neural networks [12]. These are sequence-to-sequence models capable of processing a sequence of inputs and output

another sequence, for example, converting sentences from English to French. This is done through an encoder and a decoder component with the original sentences passed through the encoder and the targeted translation generated by the decoder. In the transformer model's case, this is done through the use of the attention mechanism [29]. Individual layers of the model's encoder and decoder contain the attention mechanism which processes sequences of embedded vectors each representing a word. This mechanism allows for each word in the sequence to attend to all other words in different ways depending on the type of relationship such as whether an adjective describes a particular noun or the object a pronoun refers to. The resulting output therefore encodes higher level relationships between the elements and is useful for understanding the overall context.

How the attention mechanism learns the relationship between the word tokens is by first linearly mapping them into their corresponding key $\boldsymbol{K}$, query $\boldsymbol{Q}$, and value vectors $\boldsymbol{V}$. The level of attention between a word token and all other words in the sequence is mapped out by the dot-product of that word's query and the keys of all the words in the sequence. A softmax function is then applied to the scaled dot-product to generate a set of corresponding weights for each value element in the sequence as shown in Equation 2.18 where d_k is the dimension of the key embedding [29]. The weighted sum of the value tokens is then combined to generate the final output, $\boldsymbol{x}$, for the query token, which is then passed to a feed-forward layer, $FFN(\boldsymbol{x})$, as shown in Equation 2.19 where $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ are the matrix weight and the vector bias [29]. The outputs, each corresponding to a query token, are then combined to form the output sequence which is then passed to the next attention block.

$$Attention(Q, K, V) = softmax(\frac{QK^\intercal}{\sqrt{d_k}})V \tag{2.18}$$

$$FFN(\boldsymbol{x}) = max(0, \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2 \tag{2.19}$$

The decoder takes the contextual information and generates the next translated token based on the translation so far. This is done with the cross-attention mechanism which uses the same attention structure but combines the key and value output sequences from the encoder with the query sequence from the decoder. A mask can be added to the decoder to hide the tokens that follow the current token to make the model auto-regressive.

Compared to RNN which must generate hidden states for each word before the next word can be processed, the transformer can process all the words in parallel which results in

faster training speed and the ability to work with long input sequences [29]. As a result in recent times, transformer models have replaced RNN and LSTM as the state-of-art model for various tasks involving sequence prediction. Below we will introduce some promising transformer models used in robotics-related tasks and address their shortcomings when it comes to their practicality for learning from demonstrations.

2.2.2 Gato

The main goal of the Gato model is to train a single generalist agent, through offline supervised learning or reinforcement learning, capable of learning a vast array of different tasks proficiently and leverage that knowledge to learn new tasks with a minimal number of demonstrations [21]. The Gato model structure consists of a decoder that auto-regressively predicts the distribution of the next discrete token, either as texts or agent actions, given the inputs so far. The actions can then be fed to the agent to generate the next set of observations from the environment and repeat the process.

The tasks introduced in the paper include dialogues, captioning images, playing Atari games, navigation, and, more relevant to our work, stacking objects with robotic arms. For the model to be able to accept different modalities of data including images, discrete button presses, continuous inputs, text, and joint torque, the raw input must be converted into machine-interpretable vectors. This is done through the tokenization of the various input modalities and ordered into a sequence of embedded tokens acceptable as inputs to the decoder [21].

To test for skill mastery and generalization, the Gato model is applied to a simulated RGB Stacking robotics task and the same real-world stacking task using a Sawyer robot arm [21]. The setup uses 128x128 camera images and the robot's joint and poses information to stack red blocks on top of blue ones while ignoring the green blocks [21]. While Gato is capable of performing competitively on the stacking benchmark, the requirement of up to hundreds of thousands of simulated and real training data sets is not feasible for more practical applications where human-collected demonstrations are the only data sets available and are sparse in quantity.

2.2.3 RT-1

Similar to that of our works, the goal of the Robotic Transformer (RT-1) is to train a model that can learn from a large, task-agnostic data set such that it exhibits generalization capabilities such as zero-shot learning and adaption to new environments [5]. To facilitate

this process, the RT-1 model must be capable of inputting images and instructions and then outputting the corresponding discrete robot actions and poses. The main model components that permit this capability include a FiLM EfficientNet [28], a TokenLearner, and a transformer. To preserve the efficiency of the model, the images are tokenized through the pre-trained EfficientNet while conditioned on the language instruction embeddings from the FiLM layer which are interweaved between the convolution blocks [5]. TokenLearner then further compresses and sub-samples the output of the EfficientNet into a smaller sequence of tokens. Finally, the decoder-only transformer combines the tokens of all images and produces action tokens which are then discretized.

With a total size of 35M parameters, the model is trained on a broad range of tasks collected using a mobile manipulator arm with 7 degrees of freedom within different kitchen settings [5]. The number of real-world, human-collected demonstrations is around 130k and each demonstration has to be labelled with a description of the task that was performed [5]. The model is capable of generating output at 3Hz as a result of its large size, which is far slower than the Movement Primitive models from previous subsections. Nonetheless, the evaluation shows that the model has a high success rate and generalizes well to new tasks and environmental variations.

2.2.4 PaLM-E

Similar to the Gato model, PaLM-E is a single generalist large language model for embodied language tasks that attempts to plan step-by-step instructions for robotic manipulation tasks based on visual information, the robot's state, and text embedding inputs. The model differs from the original Pathways Language Model(PaLM) which is trained purely for natural language tasks [8]. To be able to process visual data, Vision Transformer (ViT) is added to PaLM-E to convert image data into token embeddings [10]. The output is generated in the same manner as Gato in that PaLM-E is a decoder LLM that accepts inter-weaved prompts from multiple modalities and outputs its corresponding completion auto-regressively. Applications of the model include creating jokes based on given images, processing written math calculations, and comprehending visual-question answering tasks.

The work seeks to combine the model's interpretation of the text with the robot's sensory inputs such as joint configurations and visuals by converting them into embeddings in the same space as the language token [10]. This would allow the model to successfully solve tasks where the visual of the environment and state of the agent is important. The embedded instructions output can then be returned as the answer to the scenario or be used to condition the low-level policy handling the robotic motion control. By training

on multiple tasks similar to Gato, the model has shown the ability to perform one-shot or zero-shot generalization to a previously unseen combination of objects or object types.

To complete the pipeline from the raw inputs to controller output, a low-policy control is required to convert the intermediate step-by-step instruction into the control actions. For example, the Interactive Language model [16] can map text commands and state to robotic action by training a conditional policy. This may involve simultaneously fine-tuning the parameters of both high and low-level policies which is difficult. Furthermore, the model faces the same problem as Gato and requires to be trained on a massive 600,000 demonstration dataset. The collection involves the operator continuously performing various tasks and then manually adding hindsight annotated labels to the relevant segments of the data.

Chapter 3

Methodology

3.1 Introduction

This chapter will go into detail about the components that make up our pipeline and the rationale behind our decisions. The first section will talk about the tasks and the data collection involved. The second section will describe the data preprocessing that allows the model to process the relevant input information and helps to facilitate the model training. Finally, the last section will lay out the modifications made to the original transformer model to generate trajectory predictions.

3.2 Data Collection

3.2.1 Tasks

To thoroughly test the capability of our model, we collected demonstration from a set of tasks with varying levels of trajectory complexity. The same tasks are used for the training of the TP-ProMP model and include pick-and-placing teabags, pouring water from a pitcher, and shooting puck [33]. The data set is collected over the period of roughly two months after the setup of the task environment was completed. Figure 3.1 below shows the setup for each of the tasks.

The pick-and-place task consists of gripping the teabag with the tips of the gripper from the teabag holder and carrying it to the top of the cup before releasing it. This task

mainly tests the model's capability to follow the position of the gripper from the starting position to the end target and to generalize to different object positions. This is followed by the more complex pouring task which requires significant changes in the orientation of the gripper during the demonstration and depends on the direction the pitcher is approaching the target object. The pouring water task involves the gripper grasping the side of the pitcher, moving to the cup, and pouring the liquid content into the cup before placing the pitcher beside the cup. Due to the potential hazard of working with liquid near electronics, we use grains as a rough representation of the water. Finally, the shooting task requires the gripper to launch a puck object, randomly placed in front of the net, directly into the net while holding a mini hockey stick. This task requires the model to learn to shoot the puck in the direction of the net at a sufficient speed and be able to generalize to the various starting positions of the puck. These tasks test the model's ability to learn distinct movements that are likely to be part of any realistic task and allow the model performances on each movement type to be separately measured.

The total number of demonstrations collected for each task is separated into 30 training demonstrations per task and separate sets of validation and test sets consisting of 5-8 demos per task. Here the training set is used for the training of the model, the validation for finding the optimal hyperparameters, and the test set for the final model accuracy and the performance comparison.

3.2.2 Hardware

This subsection will describe the physical setup involved in the collection of the demonstrations, the hardware used, and the procedures for operating the hardware.

Essential to the manipulation of the objects in the scene is the robotic gripper unit previously developed for a naturalistic grasp demonstration system [19]. The unit is directly connected to the computer via a cable to allow the gripper's control to be programmed. The opening and closing motion of the gripper and the corresponding force of the grasp are controlled by a human operating a joystick. The start/end and success/failure of a demonstration are also recorded with the use of separate button inputs on the joystick, with the success button saving the demonstration and the failure button discarding it. To mimic the natural motion of a human grasp, the human operator directly holds and controls the gripper via a handle.

The position and orientation of the gripper are tracked through the NDI Polaris Optical Tracking System with two separate sets of reflective markers mounted on the gripper. Each set of markers is mounted at different positions to ensure one set is always facing the optical

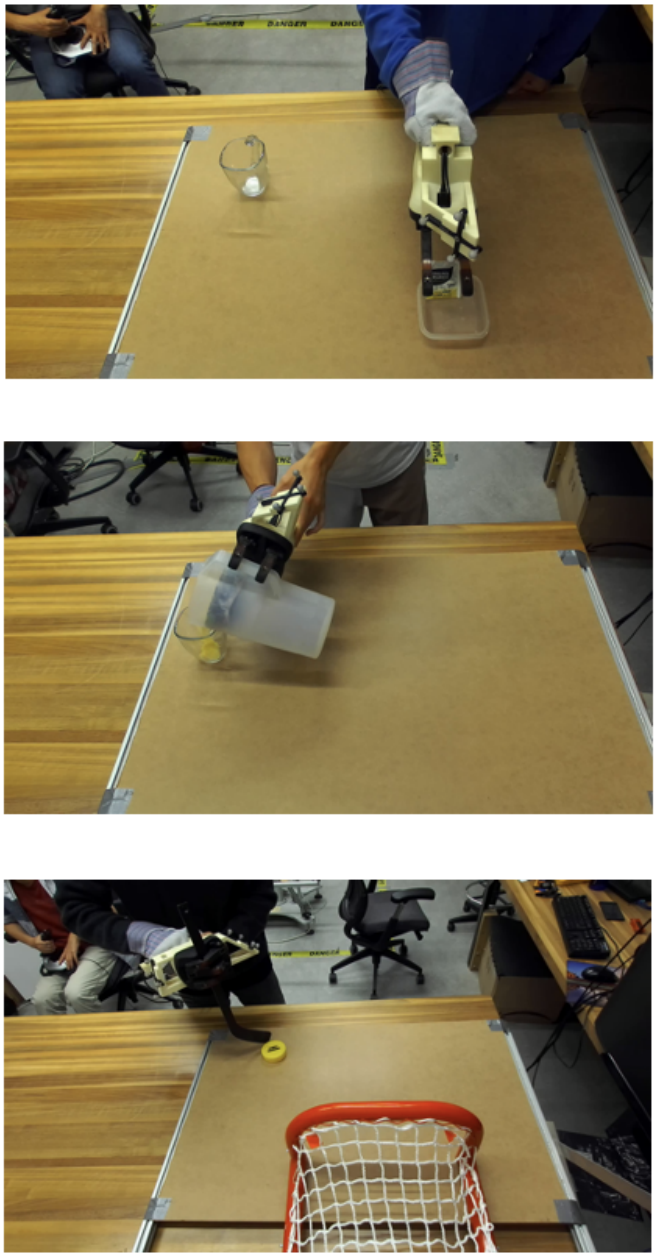

Figure 3.1: Real world task setup for pick-and-place(top), pouring(middle), and shooting puck(bottom).

tracker throughout the duration of the demonstration. Nonetheless, not all orientation of the gripper is possible due to either obstruction by the human hand or poor angles. The NDI tracking system is also connected to a separate computer that record the trajectory and detects when the markers are visible.

In addition to recording the gripper trajectory data, image data is also required to detect the position, orientation, and type of the objects in the scene. This is done through a ZED2 stereo camera connected to the desktop in order to record a continuous stream of video frames. The camera is calibrated with a printed checkerboard to ensure that the marker estimations will be accurate. The left and right eye stereo images for the demonstrations are collected at 1080p resolution which is required for depth perception after further processing. The camera is mounted overhead on a frame and all relevant object manipulations are performed within the view of the camera. Both the NDI optical tracking system and the stereo camera are synchronized and capture a frame every 0.08 seconds or 12.5 frames per second.

The TP-Transformer model was trained on an RTX2080TI graphics processing unit (GPU) with Intel Xeon Bronze 3106 CPU 1.70GHz. In addition, the DeepLabCut image processing and labelling described in the following subsection was also performed on the same hardware [17]. While it is recommended to have similar hardware, lower-grade hardware would likely still work as the TP-Transformer model is fairly lightweight.

Other items used in the collection of the task include cups, a single pitcher, a teabag, a hockey puck and stick, and a net. All of the objects can be in the Figures 3.2 and 3.3.

3.2.3 Object Pose Estimation

DeepLabCut [17] is used for visual labelling and predicting the 3D position of the object markers trained on sample images manually labelled by our group as can be seen in Figures 3.4 and 3.5. This is done by first uploading the videos of the demonstrations and then selecting distinct frames for labelling algorithmically or through visual inspection. The markers are then placed on the distinctive parts of the object that can be consistently pinpointed with a high degree of accuracy and are visible for most of the time, such as the intersections of the handle to the main body of the pitcher, the pitcher's pouring tip, or the sharp corners of any object. Then through the use of DeepLabCut3D, the raw camera images can be combined to triangulate the depth of the marker and predict the 3D positions of each point in the camera coordinate system. Multiple points could be predicted by DeepLabCut that correspond to the same part, and we chose the point with the highest confidence score [17]. An alternative method using a pre-trained LEAstereo

Figure 3.2: A collection of the relevant objects used in all three tasks, which includes teabag, cup, pitcher, hockey stick and puck, and net.

which predicts the disparity map based on the left and right images was also tried [7]. However, due to erratic depth prediction for reflective, metallic objects, the method was dropped from the pipeline and replaced with DeepLabCut3D.

Next, a template of the unique object types needs to be generated to convert between the different coordinate systems. These object templates are generated by placing an object with all markers $\{\boldsymbol{p_i^{templ}}\}_{i=1...N_{templ}}$ visible and the template object center is defined as the mean position $\boldsymbol{b^{templ}}$ of all the markers in the global reference frame or NDI's coordinate system [33]. The $\boldsymbol{A^{templ}}$ matrix is then defined as an identity matrix for the template in the global reference frame and represents the default orientation of the template object. By matching the detected object markers from the actual demonstrations $\{\boldsymbol{p_i^{obj}}\}_{i=1...N_{obj}}$ to the template, the object center and orientation can be estimated even when some points are obscured in the environment. It follows that $N_{obj} \leq N_{templ}$ as a result of the visual obstruction. Given the detected markers, a rigid 3D transformation algorithm can be used to estimate the object position $\boldsymbol{b_{templ}^{obj}}$ and orientation $\boldsymbol{A_{templ}^{obj}}$ relative to the template by minimizing the least-square error as in Equation 3.1 [2][33]. The algorithm solves for the rotation matrix of $\boldsymbol{A_{templ}^{obj}}$ by using singular value decomposition (SVD) [26] on the covariance matrix of the two re-centred point sets, $\{\boldsymbol{p_i^{templ}}\}_{i=1...N_{templ}}$ and

Figure 3.3: Disconnected robotic gripper unit used for grasping object during demonstration.

$\{\boldsymbol{p_i^{obj}}\}_{i=1...N_{obj}}$, and then multiplying the two orthogonal matrices U and V^T to calculate the orthogonal rotation matrix. The resulting object orientation and position in the global reference frame would therefore be calculated as $\boldsymbol{A_{global}^{obj}} = \boldsymbol{A_{templ}^{obj}}\boldsymbol{A^{templ}}$ and $\boldsymbol{b_{global}^{obj}} = \boldsymbol{A^{templ}}\boldsymbol{b_{templ}^{obj}} + \boldsymbol{b^{templ}}$ respectively [33].

$$\boldsymbol{LSE} = \Sigma_{i=1}^{N_{obj}} ||\boldsymbol{p_i^{obj}} - (\boldsymbol{A_{templ}^{obj}}\boldsymbol{p_i^{templ}} - \boldsymbol{b_{templ}^{obj}})||^2 \tag{3.1}$$

Figure 3.4: The cup labelled with markers at the four corners and along the handle in DeepLabCut.

3.2.4 Quaternions

So far the orientation of the objects in the scene is modelled with rotation matrices. To allow the model to predict gripper rotation trajectory, we use an alternative quaternion representation. A quaternion represents a rotation in a 3D coordinate system with four scalar parts, q_w, q_x, q_y and q_z, and the orthogonal basis vectors, $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{k}$, defined in the complex number space as shown in Equation 3.2. These basis vectors can be thought to represent the axis unit vector of a 3D coordinate system and has additional properties described in Appendix A. Addition information about the operations and identities of quaternions can also be found in the appendix section. Our model predicts the four real values as they determine the final rotation. Composition of rotations applied to an object can be easily done with multiplication and following the quaternion identities. Furthermore, the magnitude of the quaternion can be normalized to 1 to transform into a unit quaternion.

To measure the difference between two different rotations, we need a special function that accounts for the unique properties of quaternions. We decided on the Norm of the Difference of Quaternions as a similarity metric for testing the predicted gripper rotation

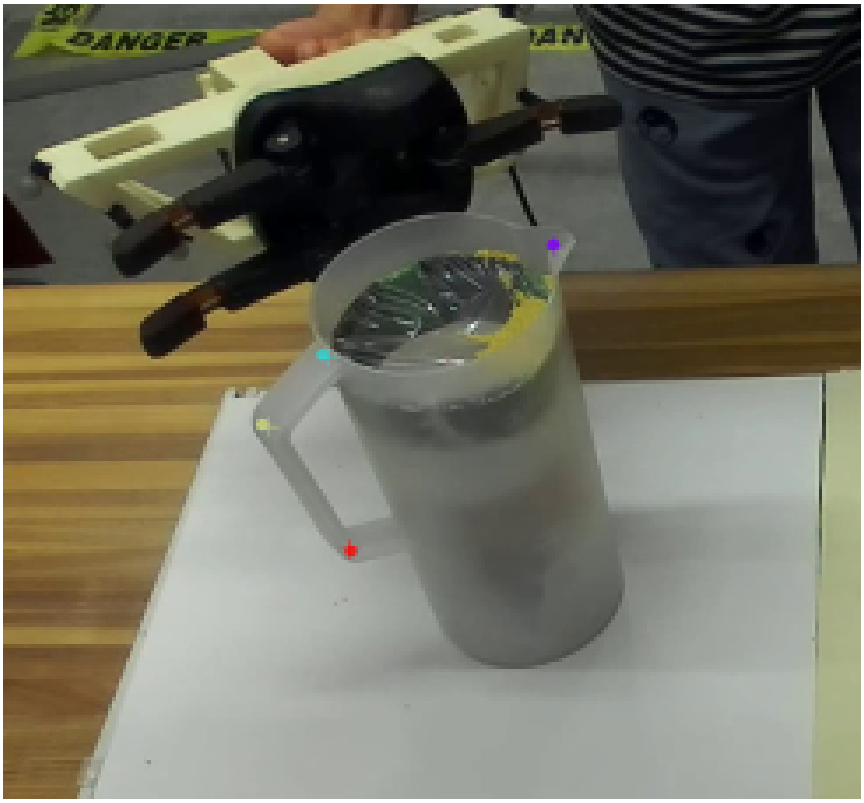

Figure 3.5: The pitcher labelled with markers at the tip of the pitcher and along the distinguishable part of the handle in DeepLabCut.

of the various models against the ground truth rotation [14]. For the metric, the output similarity measure between any two unit quaternions lies between 0 and the square root of 2 where 0 equates to the unit quaternions being the same. In Equation 3.3 below, the selection of the minimum of the norms of the difference and sum of $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$ allows the metric to account for the fact that negative of a quaternion is equal to the original quaternion, or $\boldsymbol{q} = -\boldsymbol{q}$. This means that the Norm of the Difference of Quaternions is also a pseudo-metric for comparing quaternion similarity as the function is mapped two-to-one from two quaternions to a single real number [14]. In the result section, the performances will be compared based on this metric.

$$\boldsymbol{q} = q_w + q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k} \tag{3.2}$$

$$\Phi(\boldsymbol{q_1}, \boldsymbol{q_2}) = min(||\boldsymbol{q_1} - \boldsymbol{q_2}||, ||\boldsymbol{q_1} + \boldsymbol{q_2}||) \tag{3.3}$$

3.3 Data Processing

3.3.1 Outlier Detection

Due to the sensor noises in estimating the object markers or the gripper positions, some demonstrations are invalid and can not be used for the training process. To remove the invalid demonstration, we introduce an algorithm that detects the outlier trajectories among all the demonstrations for a particular task. This is done by comparing the points of the trajectory at the same time step and removing trajectories that significantly deviate from the mean position of the group by three standard deviations. Due to the skewing of the more extreme outliers, the process is repeated until no new outliers are detected after the mean positions and the standard deviations are updated after each pass. The process is applied to both the trajectory start points and end points to detect invalid demonstrations more flexibly.

3.3.2 Dynamic Time Warping

Commonly used for pattern matching, Dynamic Time Warping (DTW) [18], a dynamic programming algorithm, compares the similarity between two different time series to realign and synchronize the corresponding distinctive time periods. Tasks that it has been used on include finding common patterns between stock prices for varying periods, speech recognition, and refining raw data sets for useful features. This section will describe the application of Dynamic Time Warping for realigning multiple trajectory demonstrations for a single common task.

Due to demonstrations persisting for different time lengths as can be seen in Figure 3.8 depicting the speed versus the time step for two random demonstrations of the same task, we wish to apply the DTW algorithm to normalize the duration of all tasks demonstrations [18]. This can be done by first selecting a representative reference as the demonstration with the median number of time steps and then applying the algorithm to the reference along with all other demonstrations from the same task. While the maximum or minimum time steps can also be chosen as the common time step, we picked the median time step reference frame as it is more robust to extreme trajectories that may not be representative of

all the task trajectories. Another reason to use the median time step is that the introduced slope constraints will fail if the reference demonstration's time steps are more than twice as long as that of the target demonstration and vice versa.

For the Dynamic Time Warping algorithm to know what kind of pattern it should be searching for in the time series, it needs to have a cost metric that tests the relevant features. For our purpose, the minimized cost metrics used in the algorithm are defined as the difference in gripper speed between the reference demonstration and that of the target demonstration. As can be seen in Figure 3.8 for pouring task demonstrations, the gripper displays distinct periods of peaks to and away from the cup and a slow down during the pouring action. These distinct features can be used to realign the trajectories to normalize all demonstration duration. The algorithm then functions by taking two trajectories with total time steps N and M and minimizing the cost metric between the matched points.

The minimization is done under the requirement that the index of the start and end of the trajectories are matched and that the realignment path satisfies the monotonicity constraint as can be seen in the equations 3.4 [18]. Here $(i(k), j(k)) = (n, m)$ are a pair of matched indices between two trajectories where k represents a common time step and the two functions, i and j, represent mappings from time step to indices.

While the DTW algorithm can be applied to two time series of any size, this can often result in poor quality matching. To ensure the smooth index matching of the two trajectories and to avoid extreme jumps in trajectory as can be seen in Figure 3.7, additional slope constrain has been imposed that disallow paths with slopes greater than 2 or less than $\frac{1}{2}$ [18]. Hence the time series that are too long and time series that are too short can not be matched without breaking the constrain.

$$\begin{aligned} n &= i(k) \text{ where } k = 1, 2, ..., K \\ m &= j(k) \\ 1 &= i(1) \\ 1 &= j(1) \\ N &= i(K) \\ M &= j(K) \\ i(k+1) &\geq i(k) \\ j(k+1) &\geq j(k) \end{aligned} \tag{3.4}$$

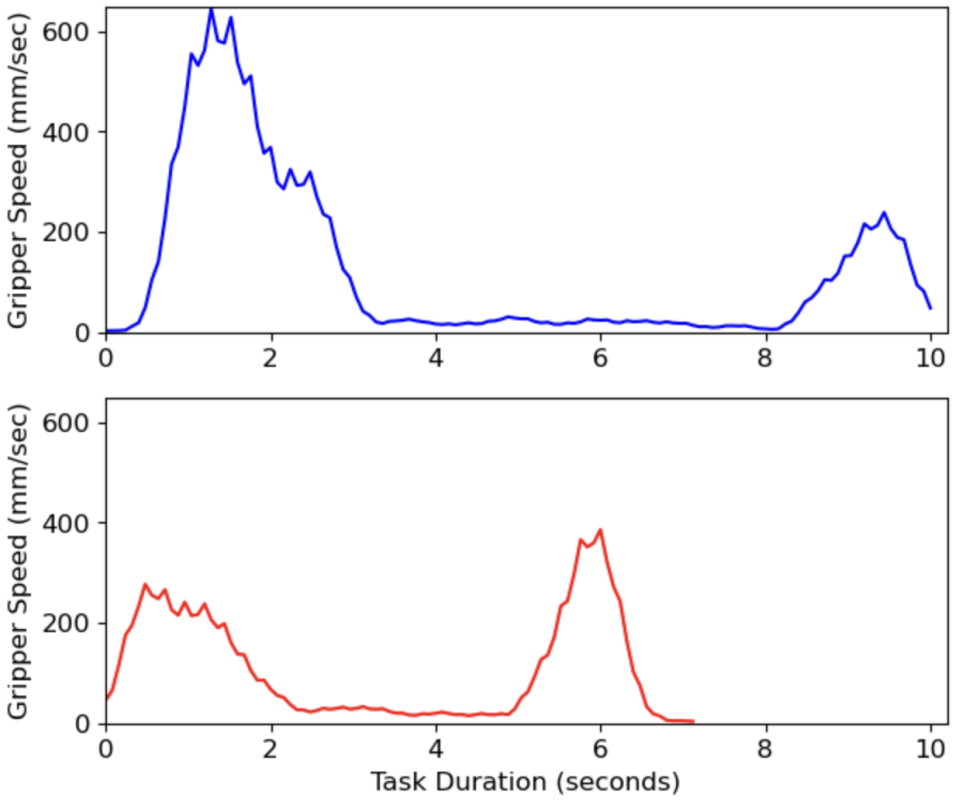

Figure 3.6: The gripper speed versus the time step for two demonstrations of the water pouring task with no realignment.

3.3.3 Data Augmentation

As shown for previous works in language and robotic manipulation tasks, the transformer's capability comes with a heavy demand on the size of the training set or additional fine-tuning on top of a pre-trained model. This is required as the models need to be able to generalize to unseen tasks, particularly for large models with a huge number of learned parameters. To augment the few demonstrations that can be feasibly collected, I draw inspiration from the field of Convolutional Neural Networks and image processing.

In the YOLOv4 object detection paper, the authors introduce a bag of freebies as a way of making the model more robust to detecting objects in various environments and positions [4]. The collection of image transformations includes modification of brightness, saturation, contrast, noises, scaling, flipping and rotation. While some of the transformations can only

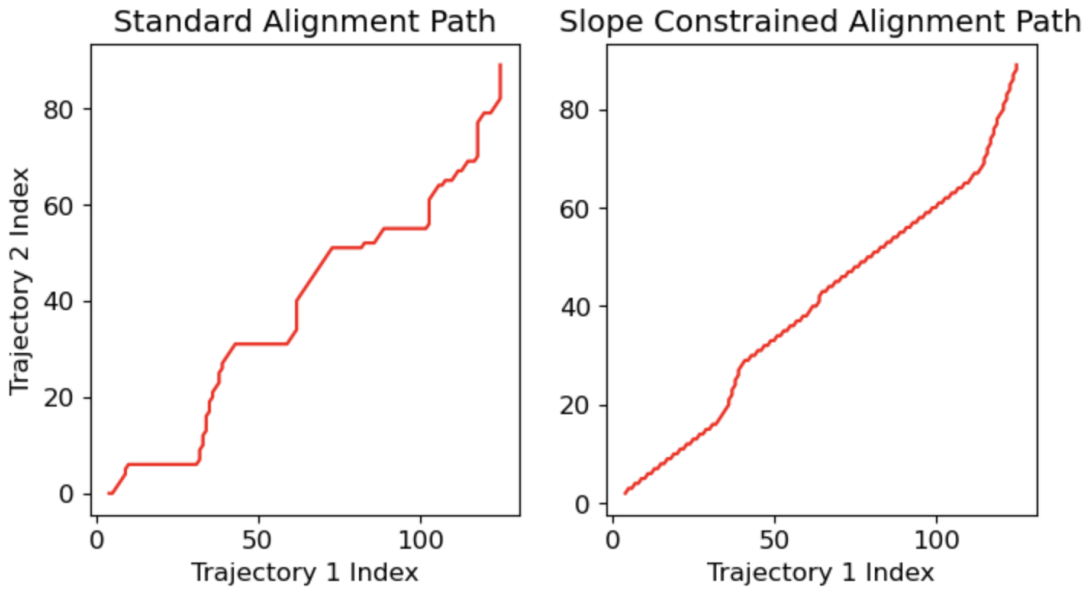

Figure 3.7: The constrained and non-constrained realignment paths for the two water pouring demonstrations.

be applied to images, others such as rotation can be more broadly applied to trajectory transformation.

To help generalize our model to different feasible object configurations, I introduced the rotated trajectory augmentation method. As described in the YOLOv4 paper, the rotation data augmentation is used to randomly rotate the image by a certain degree around the image center. In our case for a single selected trajectory, a point p on the trajectory is first randomly selected. Then a degree of rotation between 0°and 359°is randomly selected and is used to rotate the entire coordinate system around the vertical axis at the point p, generating up to 359 new trajectories per point. The new configuration is still a physically possible configuration after the transformation but becomes distinct from the original trajectory. The initial objects' position and orientation are equally transformed, creating a new set of task parameters. An example is given in Figure 3.9. The model performance improved significantly as a result of introducing the data augmentation technique. For a detailed comparison to models without the augmentation for each task, see Tables 4.1.4, 4.1.4, and 4.1.4 in the Results section.

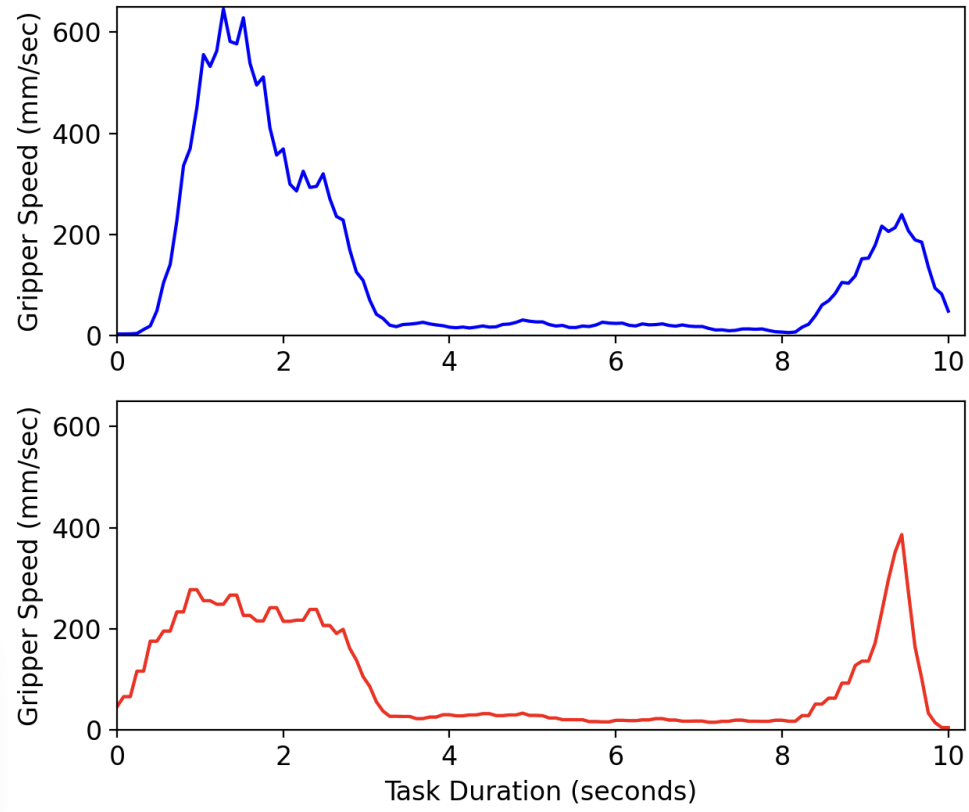

Figure 3.8: The gripper speed versus the time step for two demonstrations of the water pouring task after realignment. The realignment uses the first demonstration as the reference demonstration.

3.3.4 Normalization

As the different task demonstrations can take place in different areas of the task environment and the center of the task platform is off-shifted meters from the center of the global coordinate system, the range of the trajectory points in different axes varies. For example for all three collected task demonstrations, the range of the coordinates on the x-axis is between -1400 and -2000 millimetres while that of the y-axis is between -400 and 0 millimetres. In addition, there exists a difference in the scaling between the distance metric and quaternion metrics, since the unit quaternion's magnitude or L2 Norm of the four coefficients equals 1 where $\sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = 1$ while the unit of the demonstrated trajectories is in millimetres. This will impact the weights of each part when it comes

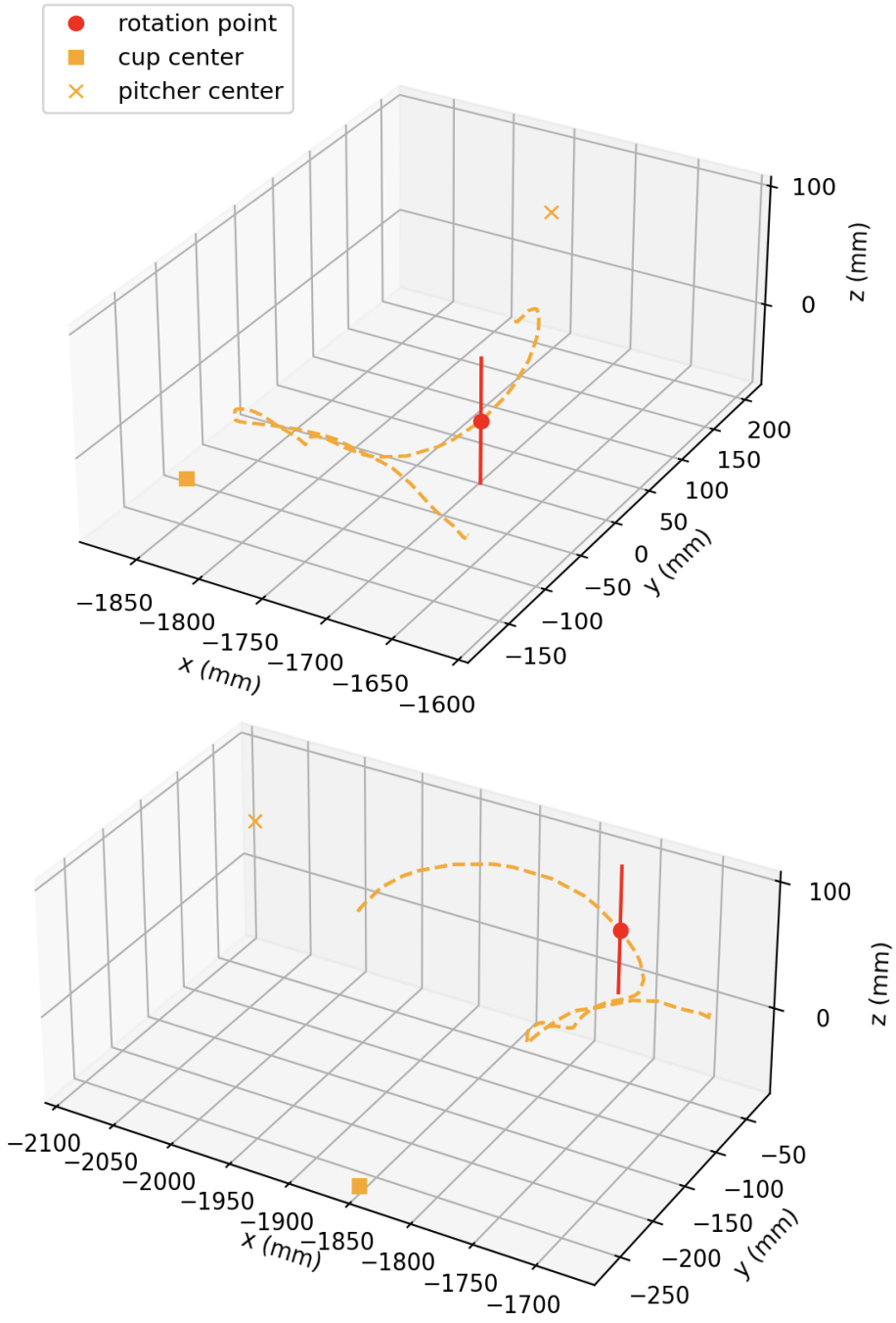

Figure 3.9: An example of a randomly selected point in the trajectory and the vertical axis around which the entire coordinate system will rotate. Original trajectory (top) and rotated trajectory (bottom) are shown.

to calculating the loss function and would have to be manually adjusted to rectify the imbalance.

To deal with the scaling problem above, normalization is introduced to our pipeline to better facilitate the stability of the training process. Some of the common normalization methods include scaling the values between 0 and 1 using the minimum and maximum values or applying log-scale to distributions with extremely skewed tail values. These techniques are commonly used in machine learning algorithms before the training process to convert the input values into similar ranges. Here the Z-score normalization is used as there does not appear to be extreme distribution of the trajectory points.

We hope to normalize the demonstrated trajectory path and object placement by subtracting the mean of the overall trajectory path and then dividing it by its standard deviation. This transforms the values to be mostly around 0 with the standard deviation of 1. To keep the weight equal among the x, y, and z axes, the same scaling is applied across all of the axes. The mean position and the standard deviation of the trajectory are then saved and can be used again when converting the predicted trajectories back into the world coordinate system.

3.4 Model Structure

3.4.1 Introduction

This subsection will describe the modifications made to the original language translation model introduced in the paper "Attention is All You Need" [29] adapted for the task of trajectory prediction. A brief description of the model's attention mechanism was given in the background chapter.

3.4.2 Object and Trajectory Input

Before the raw positions and orientation of the object and gripper can be processed by the model for prediction, they have to be modified with the relevant information. In the original transformer language model, the input tokens representing words or other miscellaneous symbols were converted into embedding vector representations before entering the model as inputs. In this case, vectors are created containing the individual information of the objects and gripper in the scene. This includes the object position, orientation, type, and the

current task for the demonstration. Here I let the vector $\boldsymbol{p^{obj}} = [p_x^{obj}, p_y^{obj}, p_z^{obj}]$ represents the object position and $\boldsymbol{q^{obj}} = [q_x^{obj}, q_y^{obj}, q_z^{obj}, q_w^{obj}]$ represents the object orientation.

For the model to know the corresponding object that the pose information is for, it needs to know the object type. For the object type, one hot embedding vector is used to represent each of the possible unique object types $\boldsymbol{o}^n = [o_1, o_2, ..., o_N]$ where N is the total number of unique object types and n is the unique object identification. This also allows for multiple objects of the same type in the scene to be tracked simultaneously. Similarly, the task tag is represented by one hot embedding vector $\boldsymbol{t}^m = [t_1, t_2, ..., t_M]$ of dimension size M for the number of unique tasks to be learned and m is the unique task identification. This allows for a single model to be trained on multiple tasks and perform the corresponding object manipulation for the indicated task. Alternatively, separate embedding for task and object type could potentially be learned and used instead of one-hot vector representations, but as the number of tasks is few, one-hot is sufficient for our purpose.

To combine the various types of task parameter information from the previous sections, the object pose, type, and actions are concatenated into a single vector. This vector is shown in Equation 3.5. Vector for each of the objects can then be aggregated into the object sequence in any arbitrary order. The target sequence follows the same format with the exception that the object tag indicates the vector belongs to a gripper and the pose information is initialized as all zeros when training. These final sequences are then passed through the same shared linear transformation layers before the object sequence is separately passed to the transformer encoder and the target sequence is passed to the decoder. The predicted trajectory sequence contains the pose of the gripper or the end effector across the time steps for the selected task.

$$\begin{aligned} \boldsymbol{e}_{obj,m} &= [\boldsymbol{p}^{obj}, \boldsymbol{q}^{obj}, \boldsymbol{o}^n, \boldsymbol{t}^m] \\ \text{where } \sum_{i=1}^{N} o_i = 1, \sum_{j=1}^{M} t_j &= 1 \\ o_n = 1, t_m &= 1 \\ \text{n is the object type of obj} & \end{aligned} \tag{3.5}$$

3.4.3 Positional Encoding

The positional encoding, in the case of the original translation application, was added to the embedded tokens in the encoder and decoder to indicate the order of the words in the sentences [29]. This is required as the local context in which the words are used determines

the meaning of those words. For example, an adjective that is placed right before a noun most likely describes that noun. But if the adjective is placed further away from the noun in another sentence, then the relationship between the noun and the adjective is likely weak.

For our purposes, we avoided adding positional encoding to the encoder's object sequences as the order of the objects does not provide meaningful information about their relationship. So the desirable property for the encoder object sequence to have is to be object permutation invariant. In contrast, the order of the predicted trajectory corresponding to the time steps is significant as the current point in the trajectory depends on the trajectory followed so far. Hence positional encoding is applied to the incoming decoder target sequence.

While there are many different possible positional encoding, I chose the sinusoidal encoding used in the original paper that allows for the learning of the relative position of the word tokens [29]. The sinusoidal encoding, as shown in Equation 3.6, and the target sequence embedding are combined right before the attention layers as shown in Figure 3.10. *pos* represents the position of the token in the sequence and d_{embed} represents the embedding dimension. The full structure of the modified model with the positional encoding is shown in Figure 3.10.

$$\begin{aligned} pe_{pos,2i} &= sin(\frac{pos}{10000^{2i/d_{embed}}}) \; where \; i = 1, 2, ... \frac{d_{embed}}{2} \\ pe_{pos,2i+1} &= cos(\frac{pos}{10000^{2i/d_{embed}}}) \end{aligned} \tag{3.6}$$

3.4.4 Hyperparameter Selection

Hyperparameters of a model are a set of programmer-defined parameters for training a model. These parameters control the rate at which the weights are updated or how fast the model is trained, the size of the model, and the settings of the model regularization. These are aspects that dramatically impact the final performance of the model and the programmer must take important considerations when choosing their hyperparameter's value. While the programmer could test out their choice of hyperparameters based on their experience, this is often time-consuming and arbitrary. Hence this task is typically automated by testing out the values uniformly or randomly for a fixed number of trials. The problem is exacerbated as the search space increases exponentially along with the number of hyperparameters. The remaining section will explain our hyperparameter tuning procedure and list our selected values.

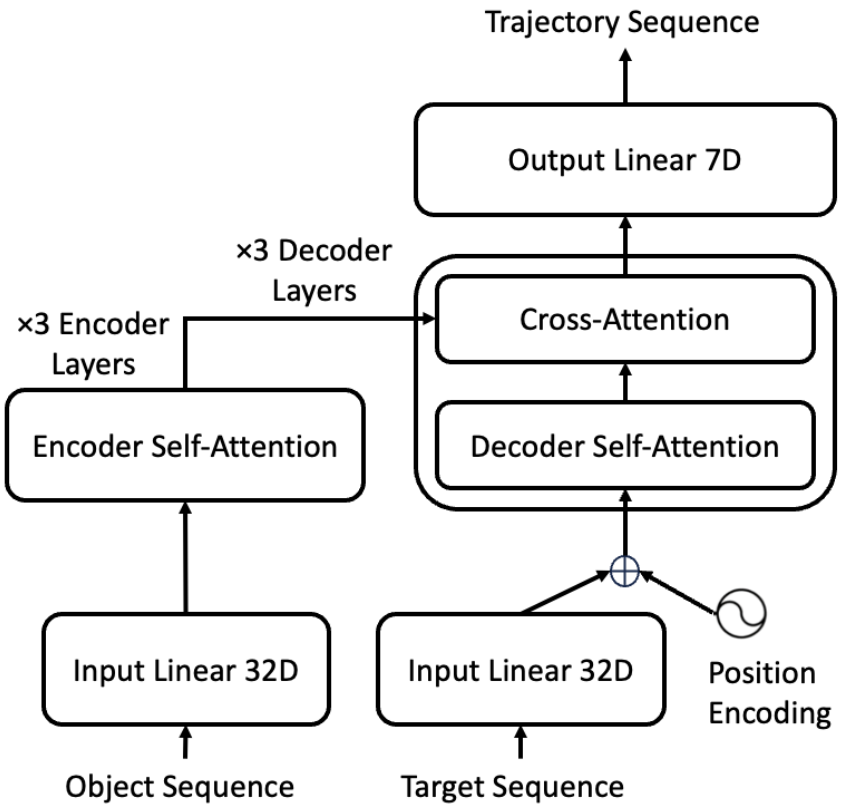

Figure 3.10: A graph of the model structure. Note that the input linear layers share the same weights for both object sequence and target sequence.

Due to the difference in the demand of computation power required for translation and predicting trajectory based on a few demonstrations, our model can be made significantly smaller than those in the original paper or the generalist models such as Gato and PaLM-E [21][8]. While a simple grid search algorithm can do the job of testing all possible combinations and selecting the hyperparameter combination with the optimal score, it is very inefficient due to the number of different hyperparameters. Optuna is an open-source automation tool that allows for the efficient search and the quick pruning of unpromising results through the optimization of a customizable object function [1]. To help minimize the size of the model while retaining its prediction capability, Optuna is used to search for the hyperparameters that impact the model size and training speed which are shown in Table 3.4.4 along with the search ranges. The hyperparameter search algorithm uses Tree Parzen Estimator [31], which builds a probability model 3.7 for the expected score s

conditioned on the hyperparameters, $\boldsymbol{w}_{hparam}$, tested so far through Bayesian optimization [3]. The most promising set of hyperparameters is fetched and used to train the final models. The probability model is then updated with the score and the corresponding hyperparameter information each time the model finishes training and its performance is measured against the objective function. This is far more efficient than grid search as the latter spends additional time searching in the poor ranges of the hyperparameter values.

$$P(s|\boldsymbol{w}_{hparam}) \tag{3.7}$$

hyperparameter	min	max	selected
embedding dim	16	512	32
attention heads	1	32	8
attention layers	1	6	3
learning rate	10e-5	10e-2	10e-3
dropout rate	0	1	0.2

Table 3.1: Hyperparameter values' search space range and their selected values.

A separate validation data set is used as a metric for how well the hyperparameter performed. Around 100 trials are conducted for 1000 epochs and the hyperparameters with minimum validation loss are saved. The final hyperparameters selected for the training of the future models are 8 multi-heads, 3 attention layers for the encoder and decoder, an embedding dimension of 32, a learning rate of 1e-3, and a dropout rate of 0.2. While the selected hyperparameters did not have the highest score during the search, they scored near the performance of the optimal model while being significantly smaller and easier to train. The final parameter size of the model using the chosen hyperparameters is 838,663 compared to the 1.2B required for Gato and 562B for PaLM-E.

3.4.5 Loss Function

To allow the model to learn from the training demonstrations, it has to learn from the error between the ground truth and the predicted trajectory generated by the model. This is done through the back-propagation of the error using the chain rule followed by iterative updates to the parameters of the model. Different prediction tasks require different types of loss functions to be able to measure the error in the model output. Common supervised learning loss functions include Cross Entropy Loss for category classification and Mean

Square Error for regression problems. But for our work, a loss function is needed capable of comparing two trajectories.

For our model, the error is calculated through the Average Displacement Error(ADE) [34] or the mean of the L2 norms between the elements of the predicted sequence of position and orientation, $\boldsymbol{p}_t^{pred}$ and $\boldsymbol{q}_t^{pred}$, and those of their corresponding ground truth, $\boldsymbol{p}_t^{gt}$ and $\boldsymbol{q}_t^{gt}$, at the same time step t. L2 norms measure the Euclidean distance between the vectors or how far the predicted point is from the ground truth. The output trajectory sequence for task m is previously normalized by a fixed number of total time steps T_m using Dynamic Time Warping. Hence the elements with the index $1 : T_m$ are the only valid outputs of the model being compared with the ground truth. Then to minimize the loss function, the model must generate trajectory point that matches those of the ground truth as closely as possible at each time step. The equation for ADE is shown below in Equation 3.8 where, for example, the variable x_t^{gt} represents the gripper's ground truth value in the x-axis at time t and x_t^{pred} represents the gripper's predicted x-axis value at time t.

$$
\begin{aligned}
\boldsymbol{ADE} &= \frac{\sum_{t=0}^{T} \sqrt{||\boldsymbol{p}_t^{gt} - \boldsymbol{p}_t^{pred}||^2 + ||\boldsymbol{q}_t^{gt} - \boldsymbol{q}_t^{pred}||^2}}{T} \\
\boldsymbol{p}_t^{gt} &= \begin{bmatrix} x_t^{gt} \\ y_t^{gt} \\ z_t^{gt} \end{bmatrix} \\
\boldsymbol{q}_t^{gt} &= \begin{bmatrix} q_{xt}^{gt} \\ q_{yt}^{gt} \\ q_{zt}^{gt} \\ q_{wt}^{gt} \end{bmatrix} \\
\boldsymbol{p}_t^{pred} &= \begin{bmatrix} x_t^{pred} \\ y_t^{pred} \\ z_t^{pred} \end{bmatrix} \\
\boldsymbol{q}_t^{pred} &= \begin{bmatrix} q_{xt}^{pred} \\ q_{yt}^{pred} \\ q_{zt}^{pred} \\ q_{wt}^{pred} \end{bmatrix}
\end{aligned}
\tag{3.8}
$$

Chapter 4

Results

4.1 Model Performance

4.1.1 Introduction

The previous chapter described the model structure and the methodology used for training the model. In this chapter, we provide an analysis of the TP-Transformer model when it comes to its performance across all three tasks. Items to be compared include the model's overall ability to make use of the task parameters [6] for generating accurate trajectory, the impact of the demonstration size, an ablation study of the model structure, and the model's performance against those of the previous movement primitive models. Along with each section, we will also describe any modifications made to the model training process to test the different aspect of the model in question.

4.1.2 Model Task Parameterization

For the successful completion of a task, the model must demonstrate the ability to create a valid trajectory based on the task parameters or the object configuration in the scene. This is important as the task parameters provide information about where the gripper should move relative to the objects' starting position at each time step. For example, the newly predicted trajectory based on the previous pouring task demonstrations must learn to start at the position where it is capable of grasping the pitcher and approaching the cup naturally. Similarly, for the pick-and-place task, the gripper must start near the teabag

and end near the cup. As part of the training process, the task parameters can change from demonstration to demonstration. The random placement of the objects' starting location is reflected by the variation in the position values and serves as a way to test the model's ability to adapt to new scenarios.

To visualize the relationship between the gripper and the relevant objects, we compare the position of both items in the horizontal plane. Shown in Figure 4.1 and 4.2 are the correlation graphs corresponding to each of the collected tasks, and the positions in each axis are aligned around 0. The points lying close to the y=x red dashed line show the strong correlation between the object and the gripper position. This can mean that the gripper directly interacted with the object during the demonstration. For example, due to the smaller size of the teabag, the gripper can more easily grasp the teabag closer to the center of the gripper. This results in the tight clustering along the diagonal line for the pick-and-place task.

Amongst the correlation plots that do not show a consistent strong correlation or display an offset, they tend to demonstrate a consistent pattern indicating a task-specific relationship between the gripper and object. It is important to note that due to the way the gripper interacts with the geometric shape and volume of the target object, the correlation can be seen in Figure 4.1 to be consistently offset by a certain amount. This is the case in the water pouring task where the pitcher used in the demonstration is significantly larger than items such as teabags and the gripper could not fully grasp around its diameter. For the shooting puck task, the gripper does not directly grab the puck, hence the correlation is not as strong. Similarly for Figure 4.2, the plots for the pick-and-place and the water-pouring tasks display a weak correlation between the gripper and the target object near the end and the middle of the task demonstration, respectively. This indicates that while the gripper can learn to approach over the cup to place the teabag or pour the water, there are still some natural variations in how the tasks can be completed due to the size of the cup or the different directions of approaching the target.

There exists also unique patterns in tasks where the gripper operates far from the object. For the shooting puck task, the graph shows three distinct clusters in the x-dimension corresponding to the three directions, left, right, and center, that the gripper can approach relative to the net as can be seen in the bottom plot of Figure 4.2. Two of the clusters for left and right are far from the center of the net as the gripper approaches from those directions but never reaches the center. The tight shifted cluster in the y-dimension is also the result of the gripper halting some distance before the net. Hence based on the starting puck position and the relative position of the net, the model learns to start at the puck, then approach and halt the gripper right before the net, and launch the puck towards the net. Based on all of the previous observations, it can be seen that the model can learn

from the task parameters provided and make modifications to its trajectory based on the different configurations.

4.1.3 Performance in Single and Multi-Task Learning

Similar to how Gato can quickly adapt to new tasks by fine-tuning itself to previously unseen data [21], this section examines our TP-Transformer model's ability to perform well on multiple tasks when compared to the same model trained for a single task. By training the model on multiple tasks, it is expected that the movements learned from one task can be used by the model to refine the motions for a separate task. This would impact the model efficiency since the multiple tasks can all be learned on a single model and still perform adequately, resulting in the reduction of the overall parameters.

To determine if the TP-Transformer model benefits from multi-task learning, a series of TP-Transformer models are separately trained on the single-task demonstrations and then compared to the models trained on all three tasks. Both types of models are trained for an equal number of epochs with the single-task model requiring one-third of the time due to the reduction in data set size. The results of the performance are shown in Table 4.1.3 with separate columns for single and multi-tasks. The difference between the single-task model and the multi-task model is that the single-task model slightly outperforms the multi-task model in all tasks except for water pouring when few demonstrations are available.

Alternatively, the size of the model parameters is also doubled in terms of encoder and decoder layers to test if the model is bottle-necked by its learning capacity. However, we have found that a larger model does not imply an increase in performance and efficiency and the training time is significantly longer due to additional parameter updates. Based on our observation of the task size and previous experience, we expect that a larger model would only be beneficial with a wider set of tasks and that our model size can be further reduced on single-task learning.

4.1.4 Impact of the Encoder Layers

In the original introductory usage of a transformer for translation tasks, the encoder is responsible for the processing of the embedding of the original language before passing the output sequence to the decoder [29]. The output generated by the encoder itself alone has versatile applications and can contain very useful information as can be seen by the popularity of the BERT encoder-only models [9]. For example, the embedded output of the BERT model can be further processed by a classifier that determines the category that

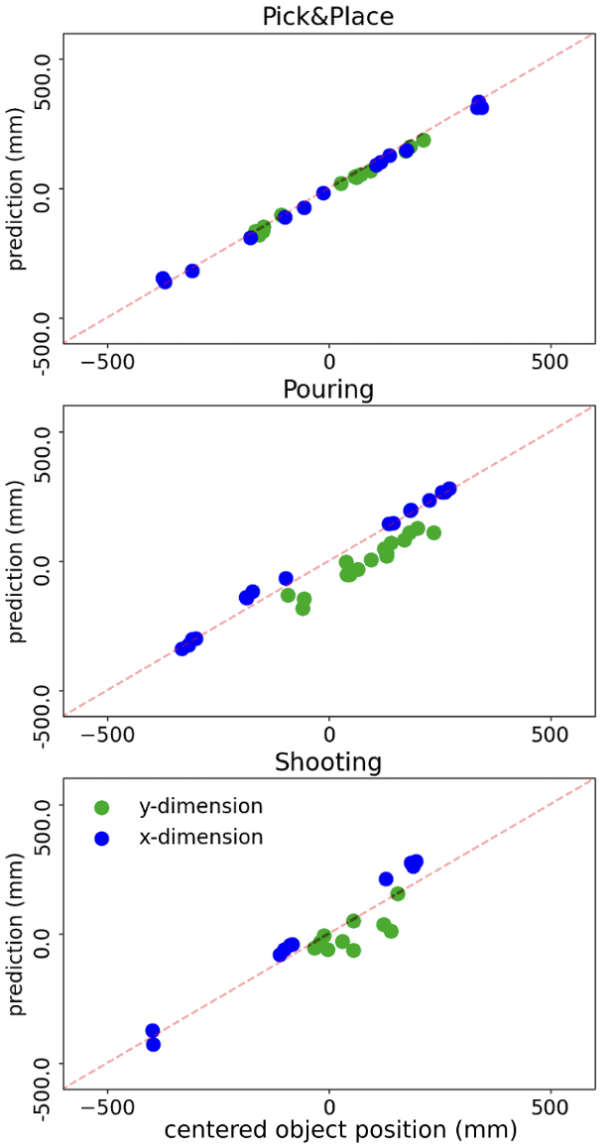

Figure 4.1: The predicted starting positions of the gripper versus the positions of the corresponding teabag(top), pitcher(middle), and puck(bottom) over multiple inference trials from the validation and test data set.

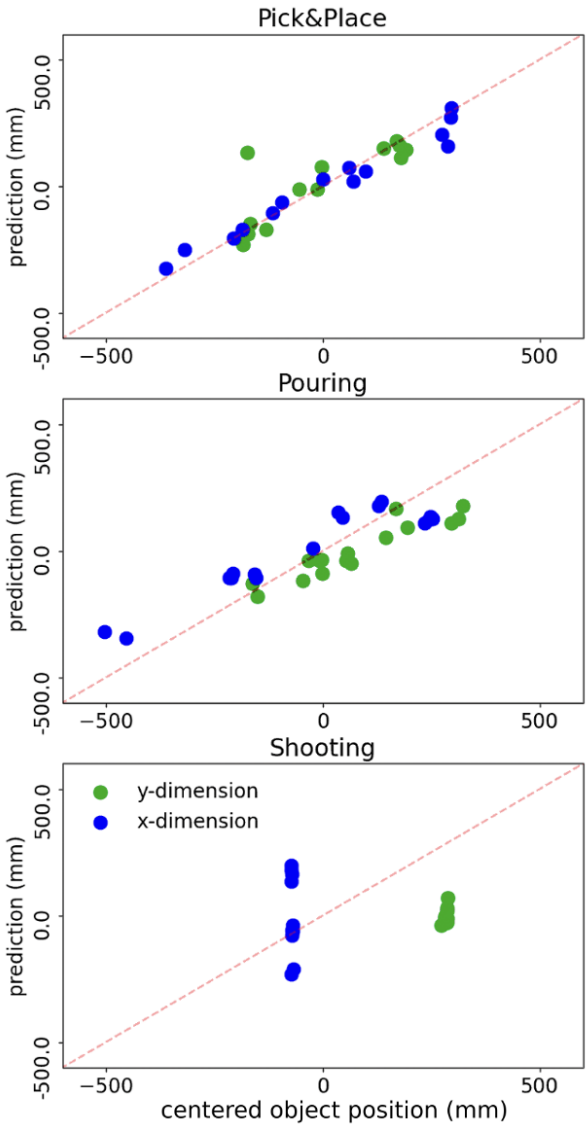

Figure 4.2: The predicted positions of the gripper versus the positions of the corresponding cup(top), cup(middle), and net(bottom) over multiple inference trials. Selected time steps are those when the gripper is closest to the targeted object. That is, the end of the task for pick-and-place and shooting puck and near the middle of the task for pouring water.

the sentence belongs to such as in sentiment analysis or for predicting the order of a set of sentences. These usages usually require that the BERT model be pre-trained due to the size of language models and are done by masking parts of the input text tokens and predicting the missing token through bi-directional self-supervised learning. Any additional parameters and model layers are then fine-tuned on the specific tasks with the learned word embedding.

Given that the encoder processes the task parameters in our model, it is important to examine the effect of the encoder size on the accuracy of the prediction and if useful information about the relationship between the objects is derived from the object sequence. To test that hypothesis, I vary the number of encoder layers from 0 to 3 layers, which would also allow us to reduce the model parameters and the training time. Note that with 0 layers, the embedded object sequence is passed directly to the decoder's cross-attention block. The results are shown in Tables 4.1.4, 4.1.4, and 4.1.4 for all three tasks. Note that the orientation column measures how well the predicted gripper rotation matches the ground truth rotation, while the position column shows how closely the predicted trajectory matches the ground truth which is demonstrated within a roughly one-meter squared task space.

Based on the results, the models follow a general trend of improved accuracy as the number of encoder layers increases across all tasks. While the average of the 3-layer encoder in the water pouring task is not optimal, the difference is minuscule and may be due to the randomness in the model training process. Overall, the result shows that the encoder can generate useful information about the object relationship that aids in the prediction of the gripper trajectory.

The start and end gripper position errors shown in the tables also reflect the certainty of the trajectory at each stage of the task similar to what was demonstrated in the task parameterization result section. Both of the predicted gripper position errors are generally smaller than the average error in most tasks, except for water pouring where the end placement of the pitcher is random.

Also included in the table is the full model without the trajectory rotation augmentation introduced in the Data Augmentation section. The non-augmented model demonstrates a significant loss of performance in all tasks due to the lack of generalization ability. As the trajectory rotation data augmentation takes very little computation power, this technique can be considered a freebie to improve model accuracy.

	Pick-and-Place		Pouring		Shooting	
Demos	Single	Multi	Single	Multi	Single	Multi
5	**51.78**	56.82	88.95	**75.50**	**52.81**	76.56
10	**43.31**	51.79	79.24	**70.53**	**47.50**	55.96
20	**37.31**	45.18	**58.40**	61.99	**42.10**	48.05
30	**35.97**	38.70	**54.06**	54.83	**35.41**	43.73

Table 4.1: Mean gripper position trajectory error between the prediction and ground truth for models learned on single and multiple tasks. The error is measured using Euclidean distance(mm).

	Position (mm)			Orientation
Encoder Layers	start	end	average	average
0	32.79	40.46	49.4	0.064
1	**19.04**	51.44	45.7	0.060
2	21.67	41.98	41.06	0.059
3	19.44	**36.24**	**38.70**	0.060
3 (No Aug)	70.49	97.69	83.02	**0.047**

Table 4.2: Pick-and-place model performance for different number of encoder layers. The error for the gripper trajectory position and orientation are separately measured using the Euclidean distance and the Norm of the Difference for gripper rotation comparison, respectively.

	Position (mm)			Orientation
Encoder Layers	start	end	average	average
0	27.36	81.04	66.35	0.093
1	26.43	**77.99**	**53.75**	**0.080**
2	27.32	81.20	55.77	0.094
3	**24.93**	79.11	54.83	0.083
3 (No Aug)	55.28	115.06	115.12	0.112

Table 4.3: Water Pouring model performance for different number of encoder layers. The error for the gripper trajectory position and orientation are separately measured using the Euclidean distance and the Norm of the Difference for gripper rotation comparison, respectively.

4.2 Model Comparison

4.2.1 Introduction

This section will compare the performance between the TP-Transformer and previous movement primitive models such as KMP, TP-GMM, and TP-ProMP using the test data set. The goal will be to clarify the advantages and disadvantages of using the TP-Transformer model over the other models. With this in mind, the previous models were modified to be trained on the same training data set split and the same validation set was used to find the optimal hyper-parameters of their respective models.

4.2.2 Demonstration Size

Due to the inherent difficulty of demonstration collection in the real-world setting by human operators, the ability of the model to generate accurate trajectories based on a few demonstration examples is a useful property. Here our model is tested by varying the size of the unique demonstrations and measuring the corresponding model's performance in terms of the ADE and the average Norm of the Difference of Quaternions introduced in the metrics section of the previous chapter.

The modification made to the training process involves splitting the total 30 demonstrations per task into sizes of 5, 10, 20, and 30 unique demonstrations per task. Three separate models are then trained independently per demonstration size. For sizes greater than 10, the total 30 demonstrations per task are truncated into the appropriate size with overlaps in demonstrations. For example for a size of 10 demonstration, the first model will learn based on the first 1-10 demonstration indices, the second model based on the 11-20 demonstration indices, and so on. For the size of 20 demonstrations, the first model would take the 1-20 demonstration indices, 11-30 for the second, and 21-30 and 1-10 for the third, resulting in a partial overlap of data. For the size 30 demonstration models, the model is trained on the full data set. Hence when considering the results below, the variance of the prediction error will be dependent on the size of the unique demonstrations and will decrease as the size increases.

Based on the error plots 4.6 and 4.7, all the models generally display increasing accuracy as the size of the demonstration training set increases. This is expected as the model has more demonstrations that it can learn from. The performance jump is particularly visible between the models trained on the size 5 demonstration set and those trained on size 10. This can be seen in Figures 4.8, 4.9, and 4.10 where the trajectories generated by a demo

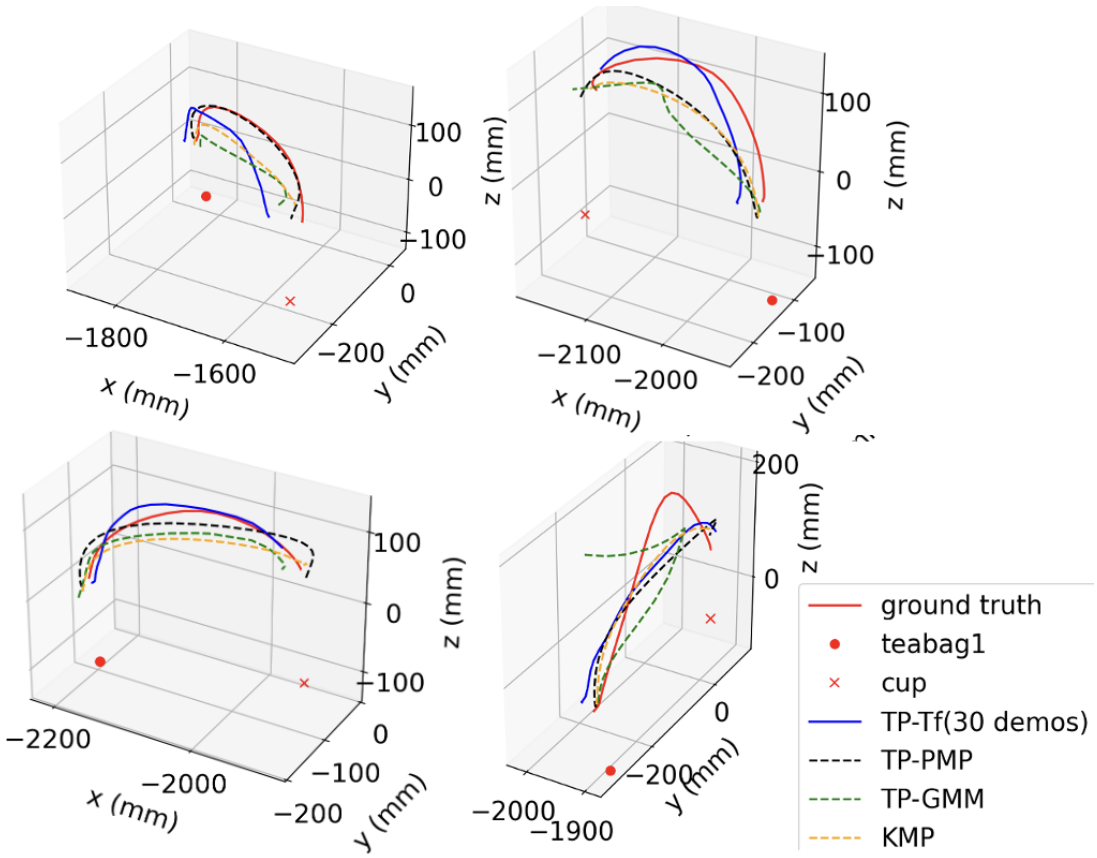

Figure 4.3: Four of the test demonstrations for the pick-and-place task and their corresponding trajectory predictions from the TP-Transformer, TP-GMM, TP-ProMP, and KMP models.

	Position (mm)			Orientation
Encoder Layers	start	end	average	average
0	43.78	33.52	47.21	0.051
1	38.64	37.08	48.95	0.056
2	37.65	32.22	46.29	**0.047**
3	**32.44**	**31.77**	**43.73**	0.050
3 (No Aug)	37.82	32.43	55.14	0.066

Table 4.4: Puck shooting model performance for different number of encoder layers. The error for the gripper trajectory position and orientation are separately measured using the Euclidean distance and the Norm of the Difference for gripper rotation comparison, respectively.

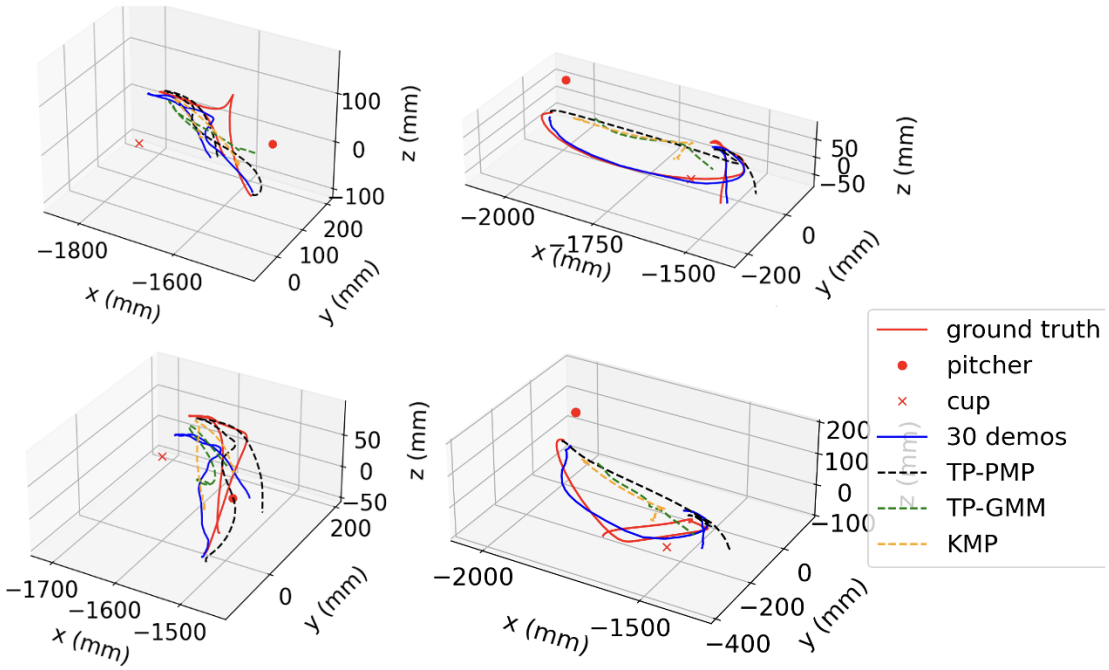

Figure 4.4: Four of the test demonstrations for the water pouring task and their corresponding trajectory predictions from the TP-Transformer, TP-GMM, TP-ProMP, and KMP models.

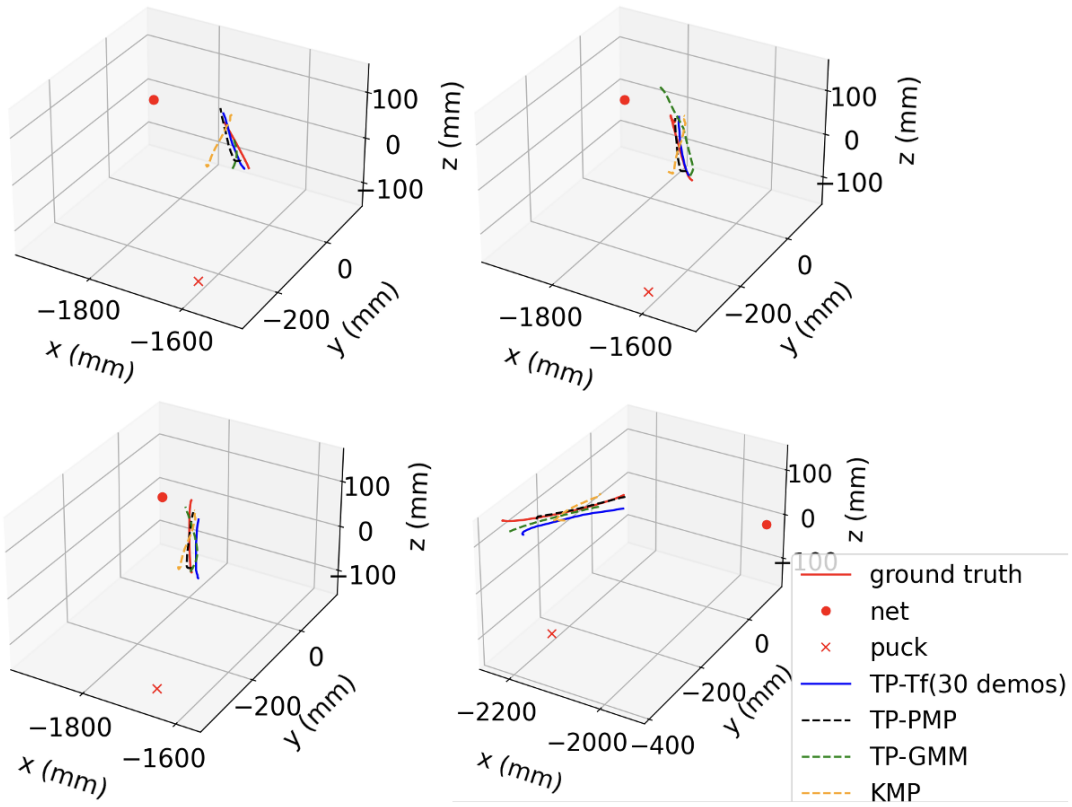

Figure 4.5: Four of the test demonstrations for the shooting puck task and their corresponding trajectory predictions from the TP-Transformer, TP-GMM, TP-ProMP, and KMP models.

size 5 model often deviate significantly from those generated by models with larger demo sizes. Hence it is recommended that a minimum of 10 demonstrations are used to train the TP-Transformer model.

In terms of performance across different models, the TP-Transformer shows a noticeable performance improvement over some of the previous models. The TP-Transformer surpasses the TP-GMM and KMP models by at least 20 mm at larger training sizes and stays close to the performance of the TP-ProMP model in most tasks.

For the quaternion metrics, the demonstration size does not affect the model performance for pick-and-place and shooting puck as those tasks do not involve a high degree of gripper orientation change during their demonstrations. Hence it is only useful to examine the water pouring task for pose prediction accuracy. According to Figure 4.7, the TP-Transformer is consistently better than most previous models across different demonstration sizes except for TP-ProMP, where the performances are also very close.

4.2.3 Runtime Comparison

This section will examine how quickly the model can be trained and used to generate the predicted trajectory. This can be important as bigger models can require a significant amount of time to train and process new inputs. For example, having a fast algorithm could mean that the model can more rapidly adapt to new changes in the environment or be deployed quickly after demonstration collection.

While the training time for all models differs from hardware to hardware, to ensure a fair comparison between the different models we run the models on the same computer except for when the TP-Transformer model leverages the GPU for training and inference. Here it can be observed that the training time for the TP-Transformer model is significantly longer than those of the movement primitive models. Each training epoch took 1.384s for 30 demos per task or 90 demonstrations in total. The models are trained for 20,000 epochs or a total of around 7.7 hours. This is significantly longer than the training time of the established models as can be seen in Table 4.2.3. However, as shown in Figure 4.11 displaying the loss versus the number of epochs for all three tasks, TP-Transformer performance had converged around the 2500 epochs mark or around one hour. Further fine-tuning of the learning rates during the training could potentially improve the training time.

The TP-Transformer model's inference time is longer than that of most of the movement primitive models except for TP-ProMP as shown in Table 4.2.3 on a CPU, but still requires only around 1/100 of a second to generate the entire trajectory. This is promising as it

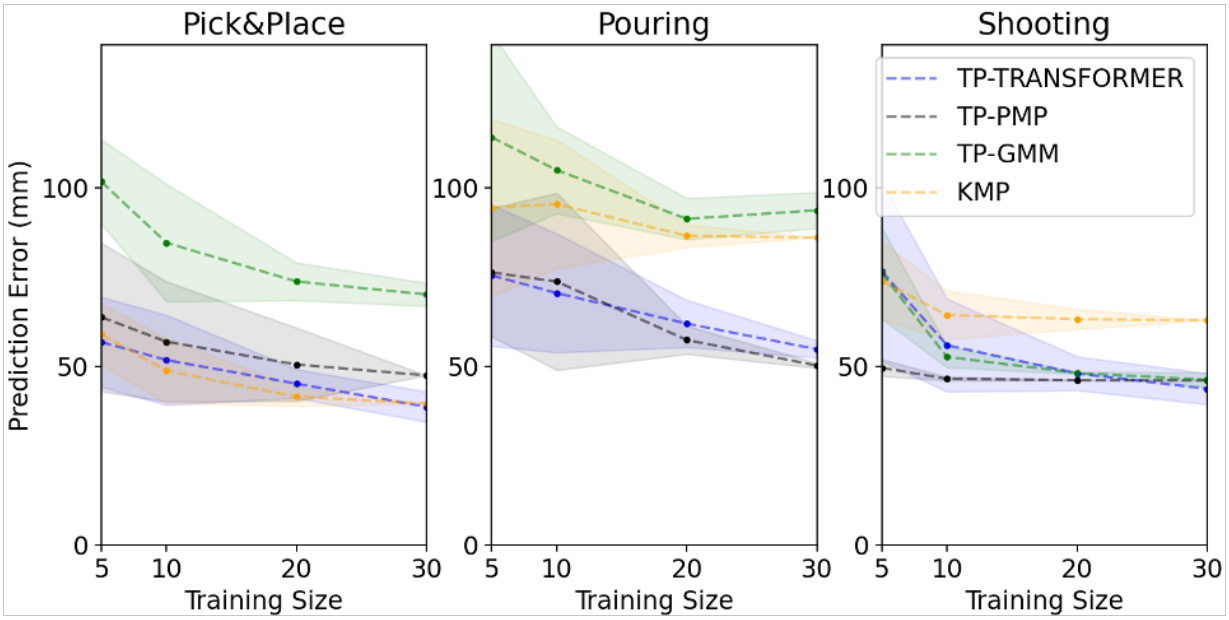

Figure 4.6: A comparison of the model performance in terms of predicted gripper positional error versus the size of the training demo set. The shaded area delimits the first standard deviation based on three separately trained models for each training set size.

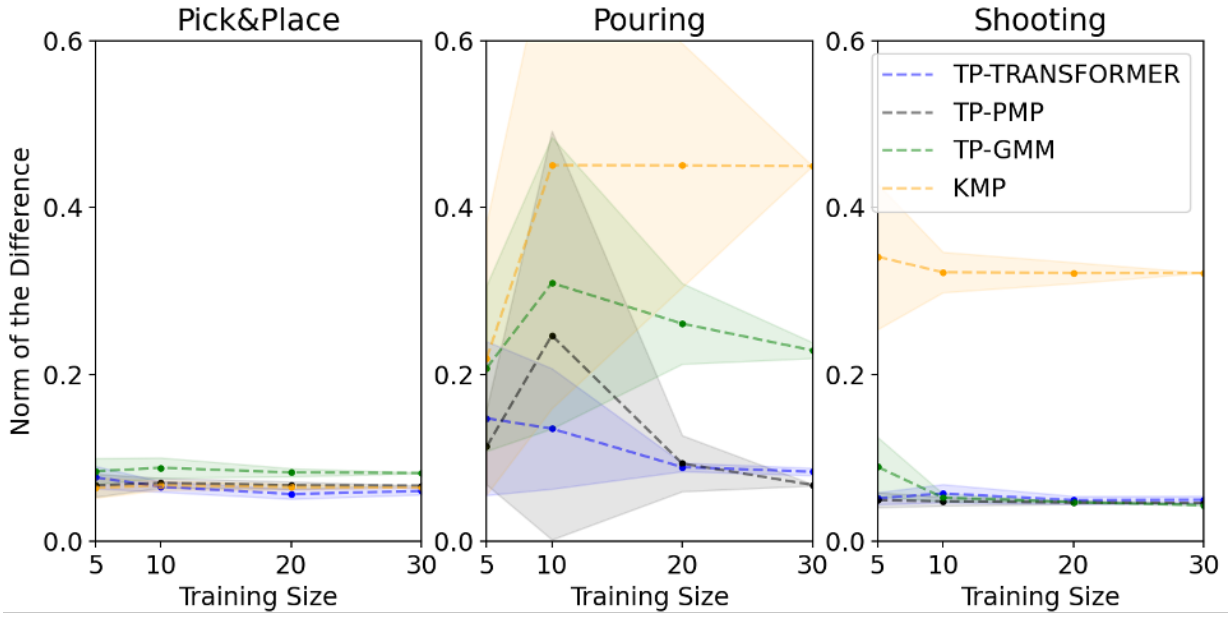

Figure 4.7: A comparison of the model performance in terms of the predicted gripper orientation versus the size of the training demo set. The shaded area delimits the first standard deviation based on three separately trained models for each training set size.

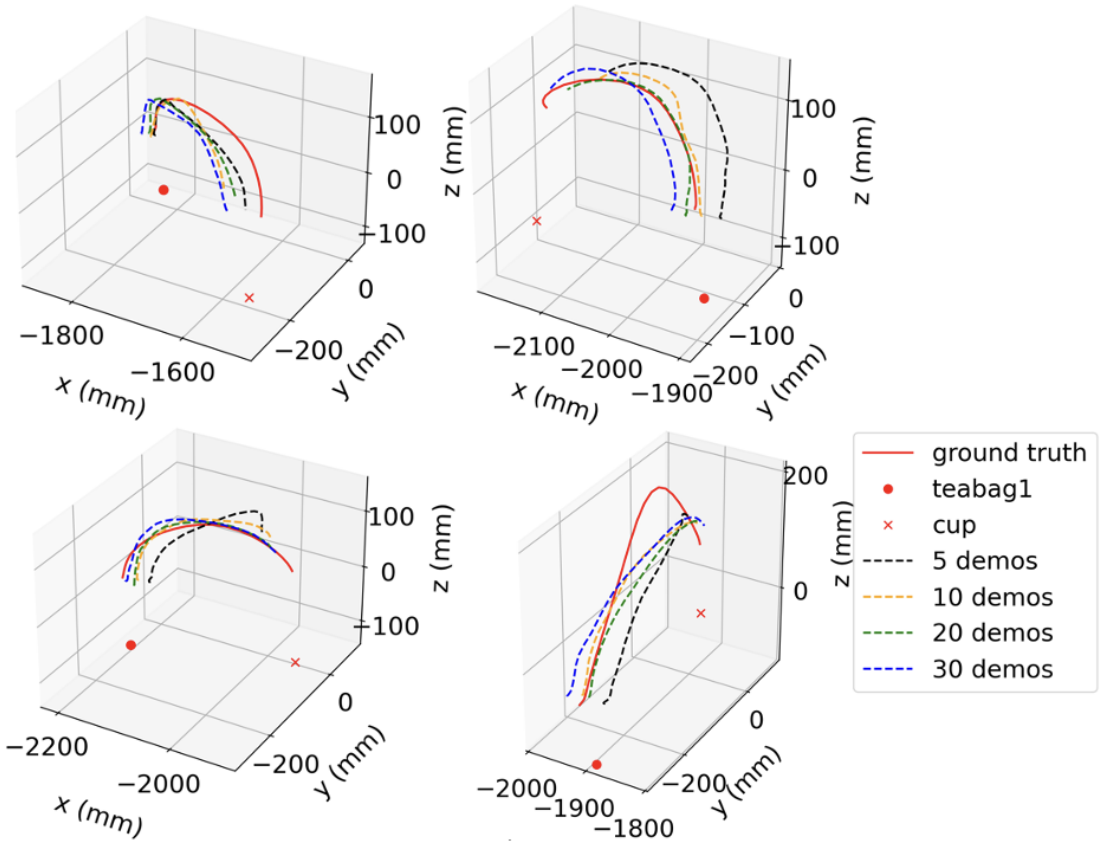

Figure 4.8: Samples of pick-and-place test demonstrations and their corresponding trajectory predictions based on the 5/10/20/30 data split.

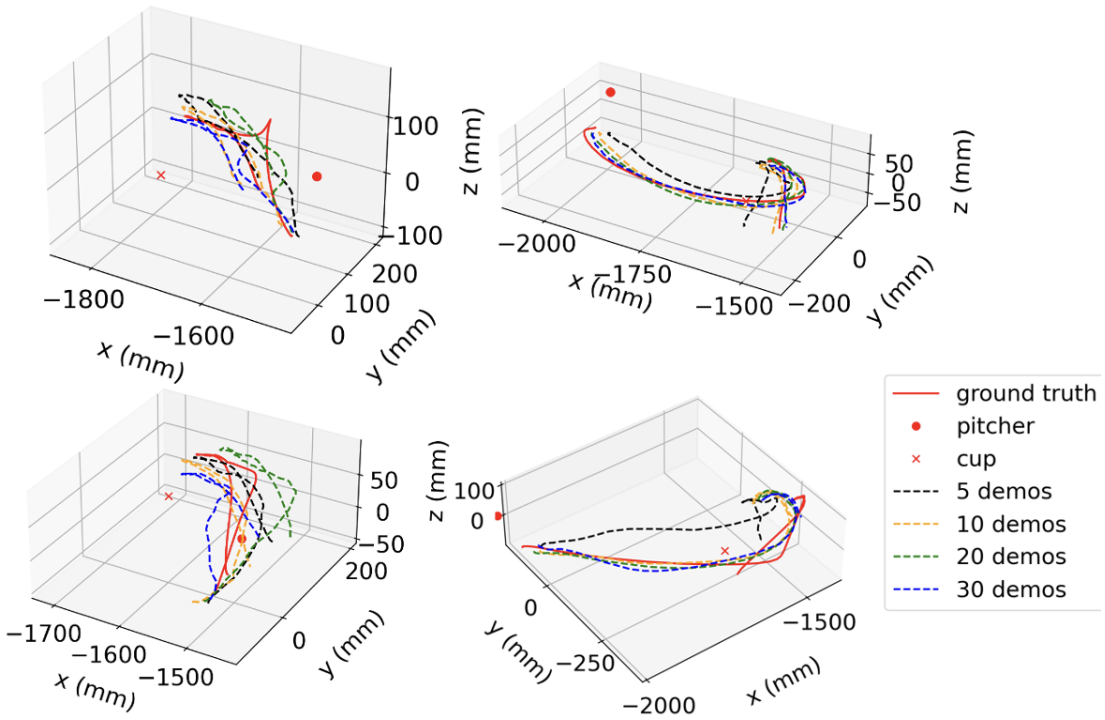

Figure 4.9: Samples of water pouring test demonstrations and their corresponding trajectory predictions based on the 5/10/20/30 data split.

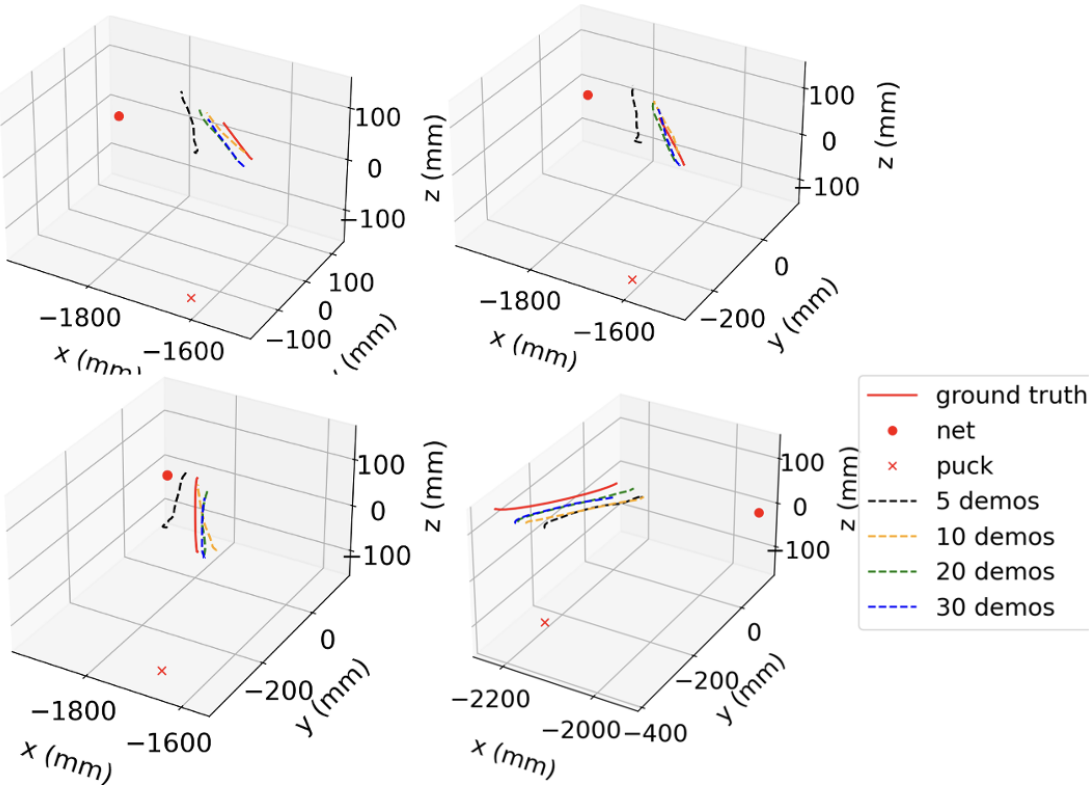

Figure 4.10: Samples of puck shooting test demonstrations and their corresponding trajectory predictions based on the 5/10/20/30 data split.

opens up the model to applications where we would want to update the model with new data about the environment and rapidly react by generating new trajectories on the go. Another discrepancy between the inference time across different tasks is due to the different number of time steps which also correlates with the complexity of the task.

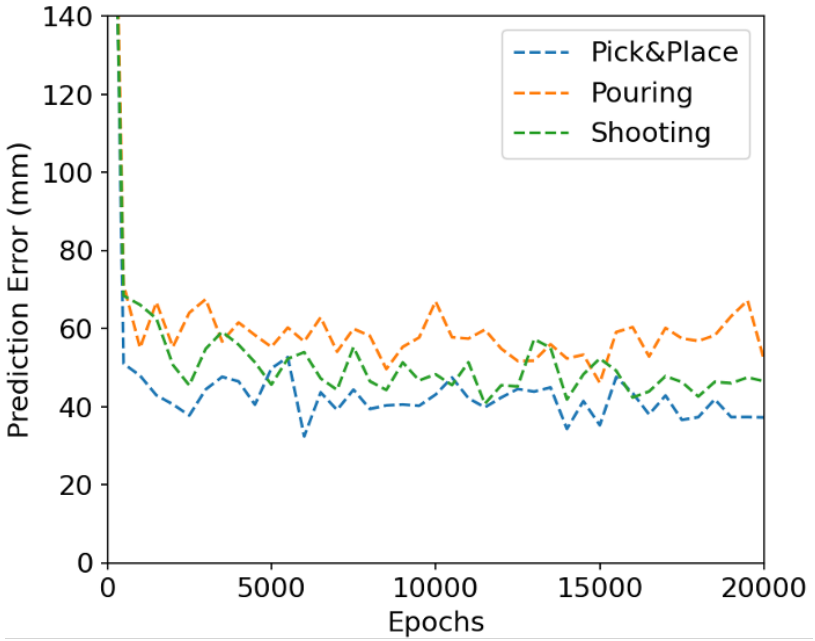

Figure 4.11: TP-transformer ADE of test set demonstrations at each 500 epochs for all three tasks.

Model	Pick-and-Place	Pouring	Shooting	Total
	Training Time(seconds)			
TP-GMM	0.17	0.54	0.07	0.78
KMP	0.22	0.92	0.10	1.24
TP-ProMP	29.70	172.18	108.76	310.64
	Training Time(hours)			
TP-Transformer	-	-	-	7.7

Table 4.5: Model training time comparison between the TP-transformer, TP-GMM, KMP, and TP-ProMP models. The TP-Transformer model is trained on the GPU and the remaining models are trained on the CPU.

	Inference Time(milliseconds)		
Model	Pick-and-Place	Pouring	Shooting
TP-GMM	3.174	6.872	2.464
KMP	0.145	0.421	0.550
TP-ProMP	17.253	96.518	38.365
TP-Transformer(CPU)	12.262	9.722	11.004
TP-Transformer(GPU)	10.727	10.732	10.737

Table 4.6: Model inference time comparison between the TP-transformer, TP-GMM, KMP, and TP-ProMP models.

Chapter 5

Conclusions

5.1 Summary

The previous chapters describe the training process behind the TP-Transformer for accomplishing a variety of everyday tasks. The modification I made to the model is inspired by previous works as described in related literature presented in Chapter 2, and I attempted to incorporate their methodologies to overcome the disadvantages of other models such as requiring large data sets and task generalization. Chapter 3 further explains the processes and the structure of our pipeline. Our usage of DeeplabCut helps us facilitate the capture of object pose information from the environment recorded with a stereo camera. Furthermore, it is shown that by adopting trajectory data augmentation and combining the task parameters into an object sequence for trajectory generation, the model can generalize across scenarios more efficiently and improve prediction accuracy. Finally, I have described and compared the TP-Transformer with previous KMP, TP-ProMP, and TP-GMM models in Chapter 4 and have found that despite requiring longer training time, the TP-Transformer model has better overall performance and the capacity to learn multiple tasks on a single model.

5.2 Future Directions

Due to time constraints, there are some promising properties of the TP-Transformer that I am not able to test out further on more complex and demanding tasks. In this section, I

will offer some potential future directions direction to explore to improve the applicability of the model to a wider range of tasks.

One of the areas that future works on the TP-Transformer model can explore is the use of the model on feedback tasks such as in the case of nut-and-bolt assembly. In this task, a robotic arm would have to learn to pick and place both the nut and bolt parts onto a fixed jig. The jig contains multiple slots that roughly fit both the nut and bolt and holds in place the hexagonal nut when screwing the bolt. The steps required to complete the task involve first correctly picking up and placing the nut in the jig. This is then followed by vertically picking and placing the bolt into the bolt slot next to the nut. The robotic end-effector will grasp the exposed head of the bolt, move on top of the location of the placed nut and perform a screwing motion. Finally, the assembled piece is placed into a box dedicated to the finished product. This is a long-horizon task different from the previous tasks introduced in the thesis since multiple steps are required for successful completion.

Force feedback is an important aspect of the environment observation that must be integrated into the current TP-Transformer model when considering the nut-and-bolt assembly task. This is necessary for error detection and correction if the threads on the bolt get stuck when inserted into the nut component. For example, the model, as soon as it detects the high torque caused by a halted screwing action, must rectify its action in real-time by performing an unscrewing motion until the bolt is freed or the forces exerted on the robot are gone. A potential way that this can be done is by predicting only the next sequence of short-term trajectory based on the true current state of the robotic arm. Due to the small size and faster inference time of the model, this is a feasible option.

To fully leverage the encoder capability, the model can make use of the current state of the environment such as the object placement and pose to predict the next short-term action. This can be done by letting the encoder part infer the next action based on the current configuration rather than explicitly providing the model with a task action to perform as is done currently. This is useful when the task can be broken down into shorter stages that involve manipulating the same set of objects and the model would infer which stages of the task the environment is currently at.

While other models can account for via-points for which the generated trajectory must pass through, the current TP-Transformer lacks a similar feature. This can be remedied by training the model with a new object type for via-point, which serves as additional information for the path planning happening in the decoder. Alternatively, the via-point position can also be placed in the input target sequences at a specific time step of the task. For example, placing a via-point location at the end of the target sequence in the pouring

task will help the model decide where to place the pitcher after pouring is completed. Similarly, placing the via-point at the middle time step of the target sequence can be used to decide the direction in which the gripper should approach the cup to perform the pouring motion. Overall, the integration of via-points would offer a greater degree of flexibility and control over how the task can be accomplished.

Lastly, while the current model can successfully deal with tasks involving different types of object manipulation, the data set diversity remains small. Hence it would be interesting to apply the model to a wider range of tasks. Not only will this test the robustness of the model, but it will also be useful to examine more deeply the model's ability to generalize motions across tasks.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[2] K Somani Arun, Thomas S Huang, and Steven D Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on pattern analysis and machine intelligence*, (5):698–700, 1987.

[3] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[5] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, et al. Rt-1: Robotics transformer for real-world control at scale, 2023.

[6] Sylvain Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent service robotics*, 9:1–29, 2016.

[7] Xuelian Cheng, Yiran Zhong, Mehrtash Harandi, Yuchao Dai, Xiaojun Chang, Tom Drummond, Hongdong Li, and Zongyuan Ge. Hierarchical neural architecture search for deep stereo matching, 2020.

[8] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, et al. Palm: Scaling language modeling with pathways, 2022.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[10] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, et al. Palm-e: An embodied multimodal language model, 2023.

[11] Simon F Giszter. Motor primitives—new data and future questions. *Current Opinion in Neurobiology*, 33:156–165, 2015. Motor circuits and action.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[13] Yanlong Huang, Leonel Rozo, Joao Silvério, and Darwin G Caldwell. Kernelized movement primitives. *The International Journal of Robotics Research*, 38(7):833–852, 2019.

[14] Du Q Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35:155–164, 2009.

[15] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, et al. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

[16] Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. Interactive language: Talking to robots in real time, 2022.

[17] Alexander Mathis, Pranav Mamidanna, Kevin M Cury, Taiga Abe, Venkatesh N Murthy, Mackenzie Weygandt Mathis, et al. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature neuroscience*, 21(9):1281–1289, 2018.

[18] Cory Myers, Lawrence R. Rabiner, and Aaron E. Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING*, 28(2):623–635, 1980.

[19] Victor Reyes Osorio, Rajan Iyengar, Xueyang Yao, Presish Bhattachan, Adrian Ragobar, Nolan Dey, et al. 37,000 human-planned robotic grasps with six degrees of freedom. *IEEE Robotics and Automation Letters*, 5(2):3346–3351, 2020.

[20] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. Probabilistic movement primitives. *Advances in neural information processing systems*, 26, 2013.

[21] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-maron, et al. A generalist agent. *Transactions on Machine Learning Research*, 2022. Featured Certification, Outstanding Certification.

[22] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[23] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning internal representations by error propagation, 1985.

[24] Matteo Saveriano, Fares J Abu-Dakka, Aljaz Kramberger, and Luka Peternel. Dynamic movement primitives in robotics: A tutorial survey. *arXiv preprint arXiv:2102.03861*, 2021.

[25] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[26] Olga Sorkine-Hornung and Michael Rabinovich. Least-squares rigid motion using svd, 2016. Technical note.

[27] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[28] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz others, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[30] Leandra Vicci. Quaternions and rotations in 3-space: The algebra and its geometric interpretation. 06 2001.

[31] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2023.

[32] Han Xu, Yao Ma, Haochen Liu, Debayan Deb, Hui Liu, Jiliang Tang, and Anil K. Jain. Adversarial attacks and defenses in images, graphs and text: A review, 2019.

[33] Bryan Tripp Xueyang Yao, Yinghan Chen. Improved generalization of probabilistic movement primitives for manipulation trajectories. page 8, 2023.

[34] Simone Zamboni, Zekarias Tilahun Kefato, Sarunas Girdzijauskas, Christoffer Norén, and Laura Dal Col. Pedestrian trajectory prediction with convolutional neural networks. *Pattern Recognition*, 121:108252, 2022.

APPENDICES

Appendix A

Mathematics

A.1 Quaternion Identities

This section will describe the quaternion identities relevant to the background calculation of the different object orientations in the paper.

Given the $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{k}$ hypercomplex vectors of the quaternions that makes up the original quaternion equation $\boldsymbol{q} = q_w + q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k}$, they follow the following identities. This can be further divided into two separate components. The real quaternion occurs when the quaternion only contains real number such as in the case $\boldsymbol{q} = q_w$, while the vector quaternion is represented by the case where $\boldsymbol{q} = q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k}$ [30]. An alternative sinusoidal representation for unit quaternions is shown in equation A.2 where θ stands for the degree by which the orientation is rotated along the (x, y, z) vector. The general representation would then be written as $\boldsymbol{q} = c\hat{\boldsymbol{q}}$ where the unit quaternion is multiplied by some scaling value c.

$$\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1 \tag{A.1}$$

$$\hat{\boldsymbol{q}} = \cos(\frac{\theta}{2}) + \sin(\frac{\theta}{2})(x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k}) \tag{A.2}$$

The conjugate of a quaternion, $\bar{\boldsymbol{q}}$, for quaternion $\boldsymbol{q} = q_w + q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k}$ is defined as in equation A.3 and has the distribution property where $\bar{\boldsymbol{p}} + \bar{\boldsymbol{q}} = \overline{\boldsymbol{p} + \boldsymbol{q}}$ and $\bar{\boldsymbol{p}}\bar{\boldsymbol{q}} = \overline{\boldsymbol{pq}}$ for any two quaternions $\boldsymbol{q}$ and $\boldsymbol{p}$. Multiplications in quaternions lack the commutative property

with an exception in the case of real quaternions, hence $\boldsymbol{pq}$ does not equal $\boldsymbol{qp}$ in general and distinct left/right quotations, $\boldsymbol{q}_L^{-1}$ and $\boldsymbol{q}_R^{-1}$, can be defined that satisfies $\boldsymbol{q}\boldsymbol{q}_L^{-1} = \boldsymbol{p}$ and $\boldsymbol{q}_R^{-1}\boldsymbol{q} = \boldsymbol{p}$. Using the conjugate, the definition of the magnitude or norm of the quaternion can then be written as in equation A.4. The unit quaternion has a Pythagorean sum of 1 for its magnitude and multiplying it with another quaternion returns a rotated quaternion vector with the same magnitude [30]. By defining $\boldsymbol{p} = 1$ for unit quaternion $\boldsymbol{q}$, then it follows that the inverse can be calculated as $\boldsymbol{q}^{-1} = \frac{\bar{\boldsymbol{q}}}{||\boldsymbol{q}||^2}$ or $\boldsymbol{q}^{-1} = \bar{\boldsymbol{q}}$ since $||\boldsymbol{q}||^2 = 1$[30].

$$\bar{\boldsymbol{q}} = q_w - q_x\boldsymbol{i} - q_y\boldsymbol{j} - q_z\boldsymbol{k} \tag{A.3}$$

$$||\boldsymbol{q}|| = \bar{\boldsymbol{q}}\boldsymbol{q} = \boldsymbol{q}\bar{\boldsymbol{q}} = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} \tag{A.4}$$

A unit quaternion multiplied with another quaternion returns a rotated quaternion vector with the same magnitude. Using this property, the composition of two or more unit quaternions represents a single successive rotation from the initial orientation or $\boldsymbol{q}_3 = \boldsymbol{q}_1\boldsymbol{q}_2$[30]. This allows for the conversion of object orientation from one coordinate system to another by multiplying the corresponding quaternion.