Jack Ninteman & Aly Khater
ECEN524

Project 1

All code utilized in this report can be found on our Google Colab notebook found at: https://colab.research.google.com/drive/1V45pcAzDPCUzWAEcOyTRCFJlyecO_ZU2?usp=sharing
Files found in the code can be found in Google Drive: https://drive.google.com/drive/folders/1fYME4rdouP7vQ4zf9uv3N0l7sQIHh_VO?usp=sharing
We would like to acknowledge Gemini and ChatGPT for helping out with parts of the code.

A. Plot Data

1) **The first step is to understand the data that are given. Prepare plots and include them in your report and provide a description of what the plots show.**

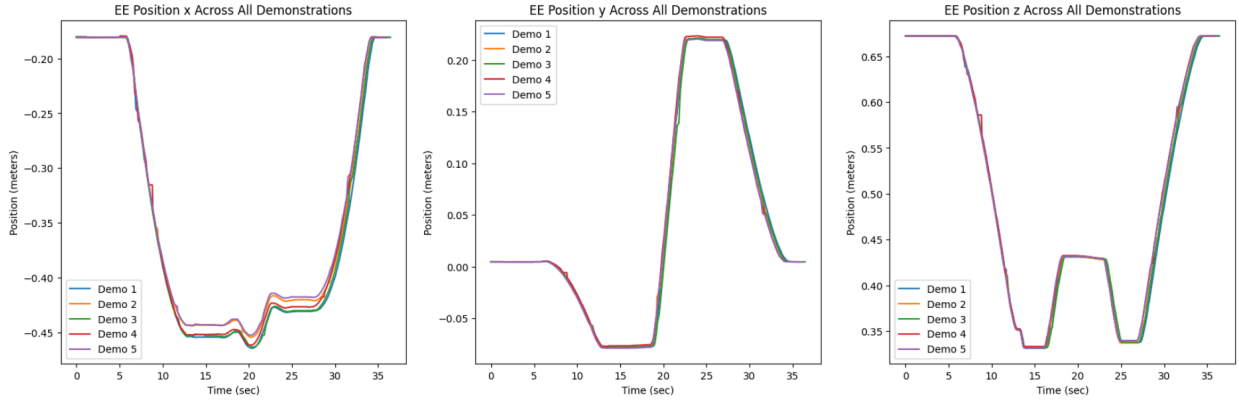
We are given files for 5 different demonstrations of the Panda robotic manipulator that contain data of its trajectory performing a pick and place task. The pick and place task consists of the following steps:

1) Starting at the robot's initial position
2) Moving the arm towards the object of interest
3) Placing the arm's end effector around the object of interest to prepare for grasping
4) Actuation of the end effector to grasp the object
5) Moving the arm to a new position
6) Actuation of the end effector (Deactuation) to let go of the object
7) Moving the arm to its final position

The data provided from the files are split into their relevant aspects: X-Position, Y-Position, Z-Position, X-Orientation, Y-Orientation, Z-Orientation, W-Orientation, Left Finger-Position, and Right Finger-Position. In Figure 1, we showcase the trajectory of the arm across all demonstrations.

Figure 1: Trajectories of End-Effector Position, Orientation, and Finger Positions Across Demonstrations

The plots shown are preprocessed from each demonstration by having the time normalized across each and having the peaks and valleys of each fall around the same area.



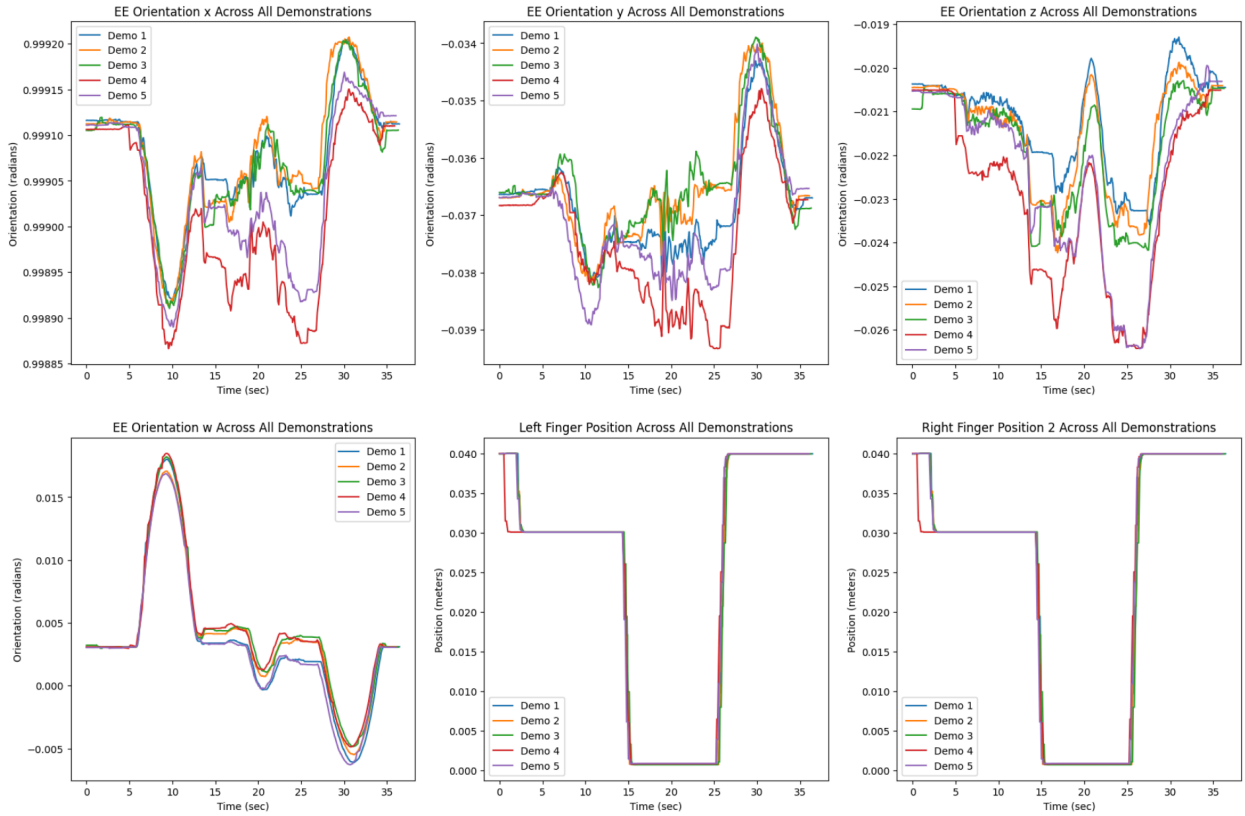


Figure 1: Plots of all demonstrations for all significant data fields

2) **Identify points of interest in your plots (e.g. gripper actuation and deactuation)**

As said in Problem 1 of Part A, the plots are normalized across time, making it much simpler to visually inspect the graphs to determine the states and actions the arm and its end-effector are currently in. The left and right fingers of the arm act simultaneously, so for the purpose of identifying points of interest, we will use the left finger demonstration as shown in Figure 2.

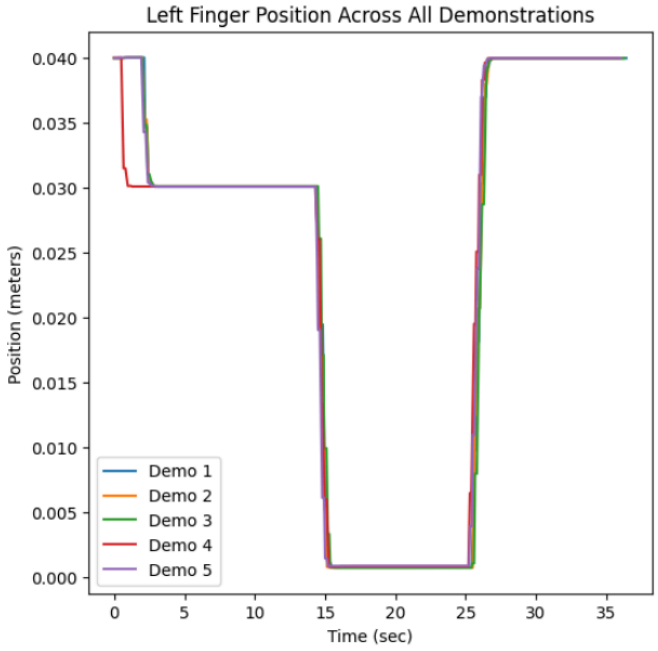


Figure 2: Trajectory of the left finger of all 5 demonstrations.

Around 14 seconds, the gripper starts to close, demonstrating that the gripper must be at the object of interest. As the graphs of Figure 1 follow the same Time axis, we can follow the movement of the arm as it moves towards the object. Around 25 seconds, the gripper starts to open, demonstrating that the gripper is at the point of placing the object. We can relate this to the plot of the Z position of the end effector as shown in Figure 3.

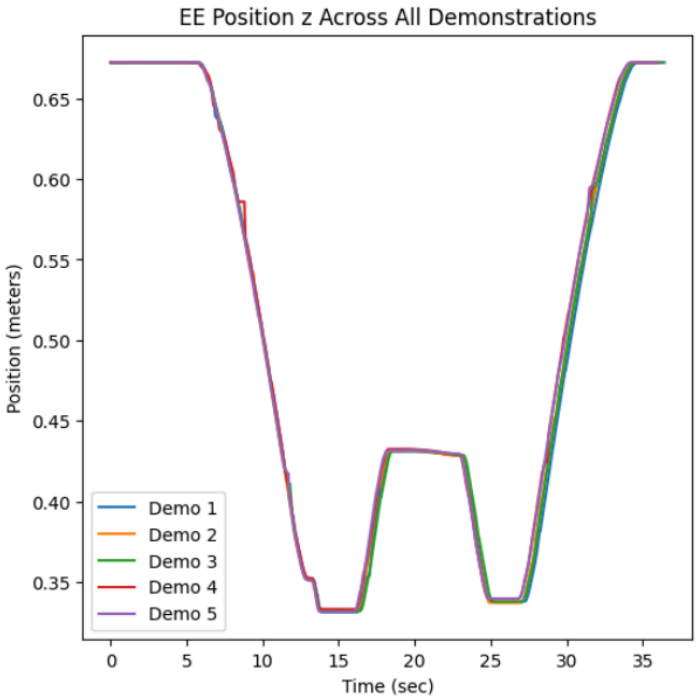


Figure 3: Trajectory of the Z-position of all 5 demonstrations.

From around 6 to 14 seconds, the arm is moving downwards towards the object to prepare for grasping. At 16 seconds, the robot arm is moving upwards after grasping the object. At 24 seconds, the robot arm is moving downwards to place the object. Finally at 27 seconds, the arm is moving upwards away from the object.

B. Trajectory Alignment

1) Apply DTW by selecting the two demonstrated trajectories that are the more similar. Provide the DTW library you implemented, screenshots of your results, and an explanation of which demonstrated trajectories were selected and why.

We used the dtw-python package to implement the DTW as found here: https://dynamictimewarping.github.io/python/. As an example, we utilized the first and second demonstration trajectories to get a grasp of what the graphs mean as shown in Figure 4.

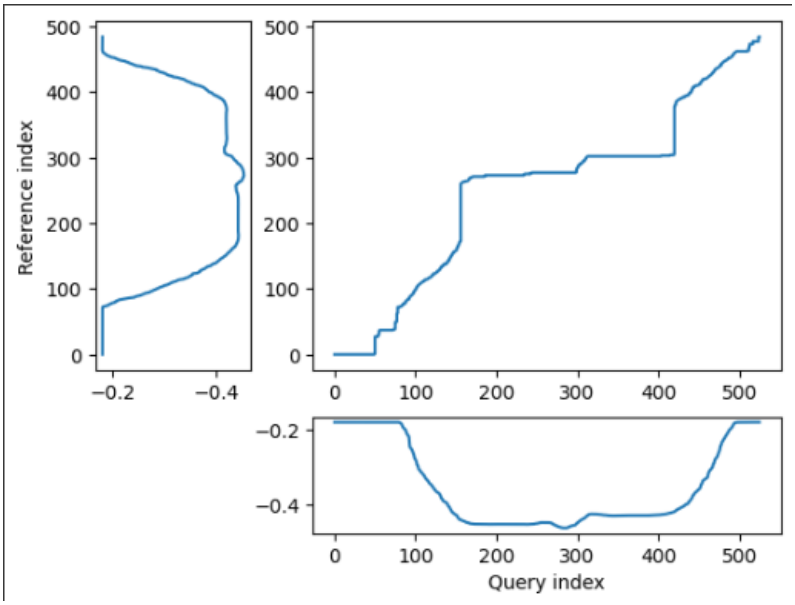


Figure 4: Plot of reference trajectory (demonstration 2), query trajectory (demonstration 1), and the reference and query plotted against each other.

As you can see, the reference and query index graphs follow a similar pattern, and when plotted against each other, combine into a graph that follows a diagonal path. As seen in Figure 5, we can see the alignment of the two time series.

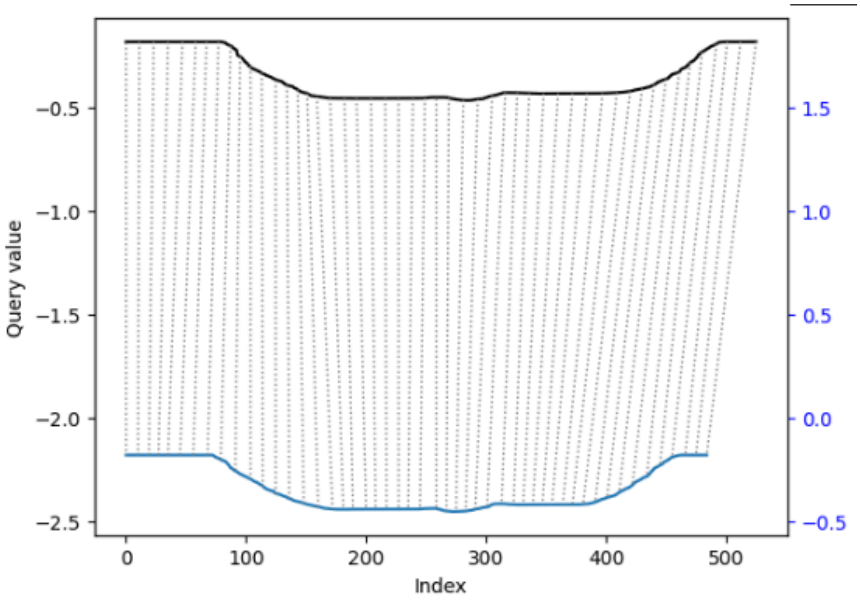


Figure 5: Alignment of Demonstration 1 and 2. Query on the left and Reference on the right.

We plotted every trajectory against each other to see which two trajectories plot the smoothest line, allowing us to choose which demonstrated trajectories are the most similar. All the plots can be found in our Python Notebook and in Appendix A. We ultimately decided on settling with Demonstration 1 and Demonstration 3, as the plot visually shows the smoothest line between the trajectories as shown in Figure 6.

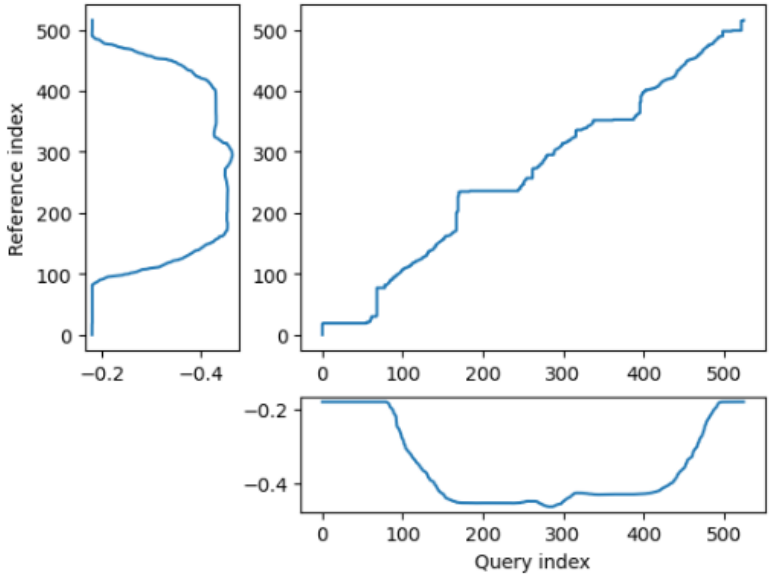


Figure 6: Plot of reference trajectory (demonstration 3), query trajectory (demonstration 1), and the reference and query plotted against each other.

All paired trajectory plots can be found in Appendix A.

We also utilized the DTW distance to help solidify our choice of demonstrations in the form of a distance matrix that showcased the cost between each demonstration's trajectory as shown in Figure 7.

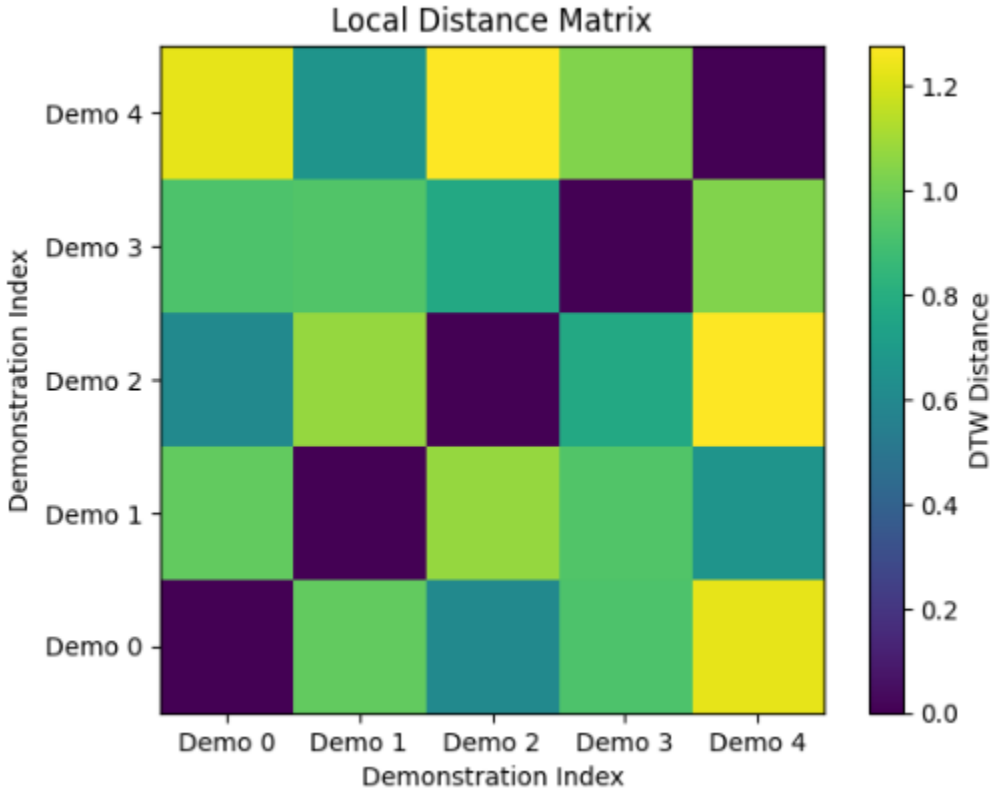


Figure 7: Distance Matrix visualizing the cost of each paired trajectory. Zero-indexed.

Since we are choosing two different trajectories, demonstrations paired with themselves can be safely ignored. The color gradient on the right represents the cost according to the DTW-distance of each paired trajectory. This solidified the choice of using demonstration 1 and 3.

2) Identify a method for aligning multiple demonstrated trajectories or develop your own method for aligning demonstrated trajectories. Explain the method you used/developed in a mathematical way and provide screenshots of the code and the results.

For this section of the problem, we needed to align all five demonstration's trajectories to one singular reference trajectory. In order to perform GMM/GMR successfully, we need all demonstrations to contain the same number of samples and have optimal time alignment. Since we already performed pre-processing on the data to normalize the time and remove time offsets, all that needs to be done is to regulate the number of samples in each demonstration. Our approach for this issue was to iteratively perform DTW on demonstration pairs, starting with the optimal pair, and recalculating the next optimal pair each time. Once this was done, we performed down-sampling to make sure that all the aligned demonstrations contained the same number of samples. Figure 8 displays how the number of samples changed after each iteration.

```
Performing DTW on demos 3 and 1:
Number of samples prior to performing DTW and down sampling:
Demo 3: 517
Demo 1: 526
Number of samples after performing DTW and down sampling:
Demo 3: 846
Demo 1: 846

Performing DTW on demos 5 and 3:
Number of samples prior to performing DTW and down sampling:
Demo 5: 501
Demo 3: 846
Number of samples after performing DTW down sampling:
Demo 5: 846
Demo 3: 846

Performing DTW on demos 1 and 2:
Number of samples prior to performing DTW and down sampling:
Demo 1: 846
Demo 2: 485
Number of samples after performing DTW down sampling:
Demo 1: 846
Demo 2: 846

Performing DTW on demos 4 and 5:
Number of samples prior to performing DTW and down sampling:
Demo 4: 445
Demo 5: 846
Number of samples after performing DTW down sampling:
Demo 4: 846
Demo 5: 846

Final sample numbers of aligned data:
Demo 0: 846
Demo 1: 846
Demo 2: 846
Demo 3: 846
Demo 4: 846
```

Figure 8: Verbose output of iterative DTW process

As you can see, this process was able to align all the data and we now have 5 demonstrations with equal numbers of samples, meaning that we can use all 5 to train GMM/GMR. However, it is crucial that our iterative DTW process did not augment the shape of the demonstrations too much, so we also need to plot the aligned demonstrations to confirm that we have maintained the integrity of our dataset. Figure 9 shows plots of all the aligned data.

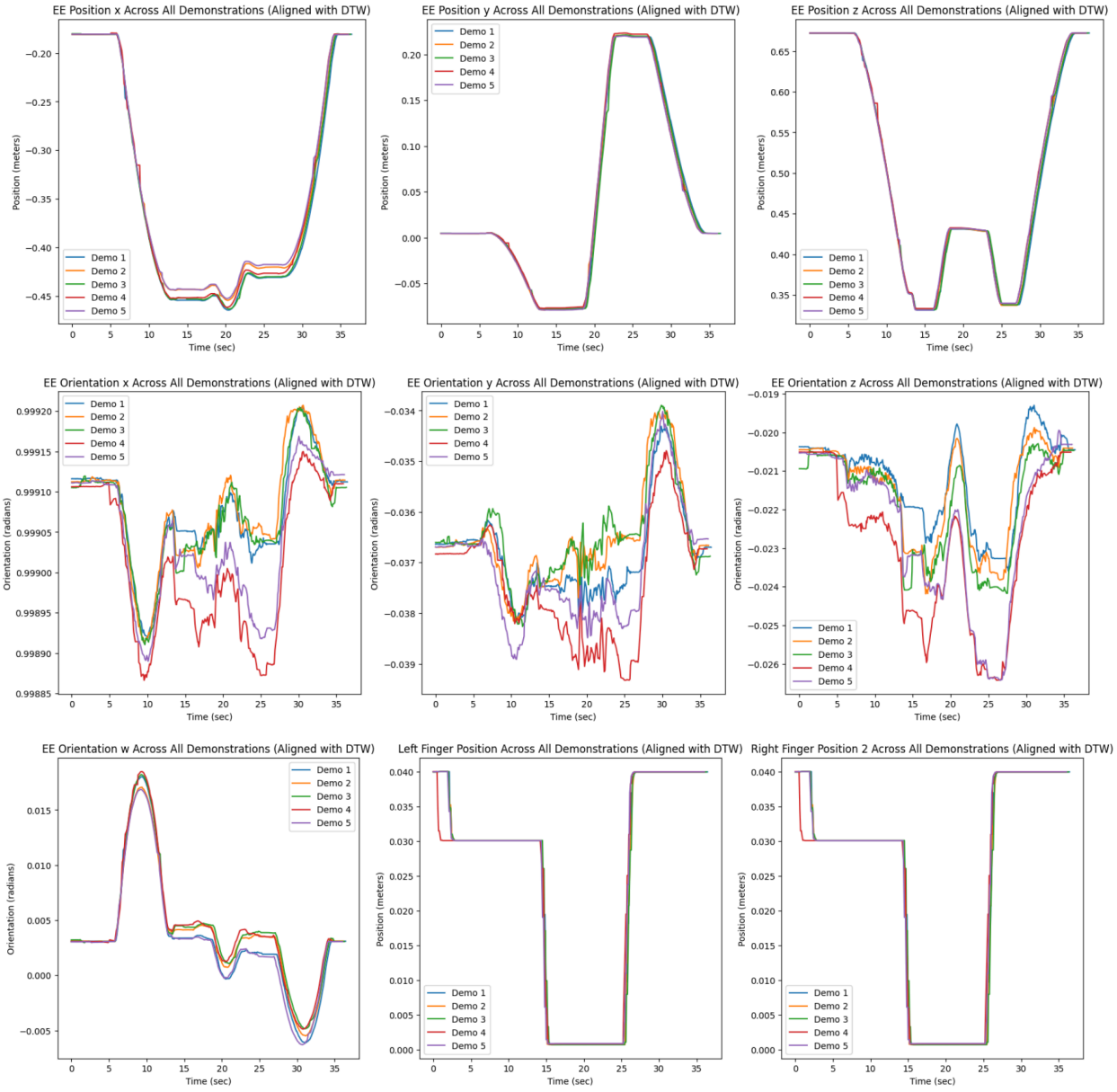


Figure 9: Plots of all aligned demonstrations for all significant data fields

As you can see, the shape of the demonstrations has been maintained, so our dataset is ready to be used to train GMM/GMR.

C. Gaussian Mixture Model/Gaussian Mixture Regression

1) Apply GMM/GMR in your aligned data. Provide details of the package you used.
We utilized a Python implementation of GMM/GMR called found here: https://github.com/BatyaGG/Gaussian-Mixture-Models. We utilized the GMM_GMR class provided by this package to create our GMM/GMR models. The math behind GMM can be found here: https://github.com/Ransaka/GMM-from-scratch. The package utilizes our aligned data as an input for our GMR function. We initialize the GMM_GMR class with one argument for the number of clusters we would like to use, which determines how many Guassian distributions the class will use. In the case of the X-position, we utilized 16 clusters which output four plots as shown in Figure 10.

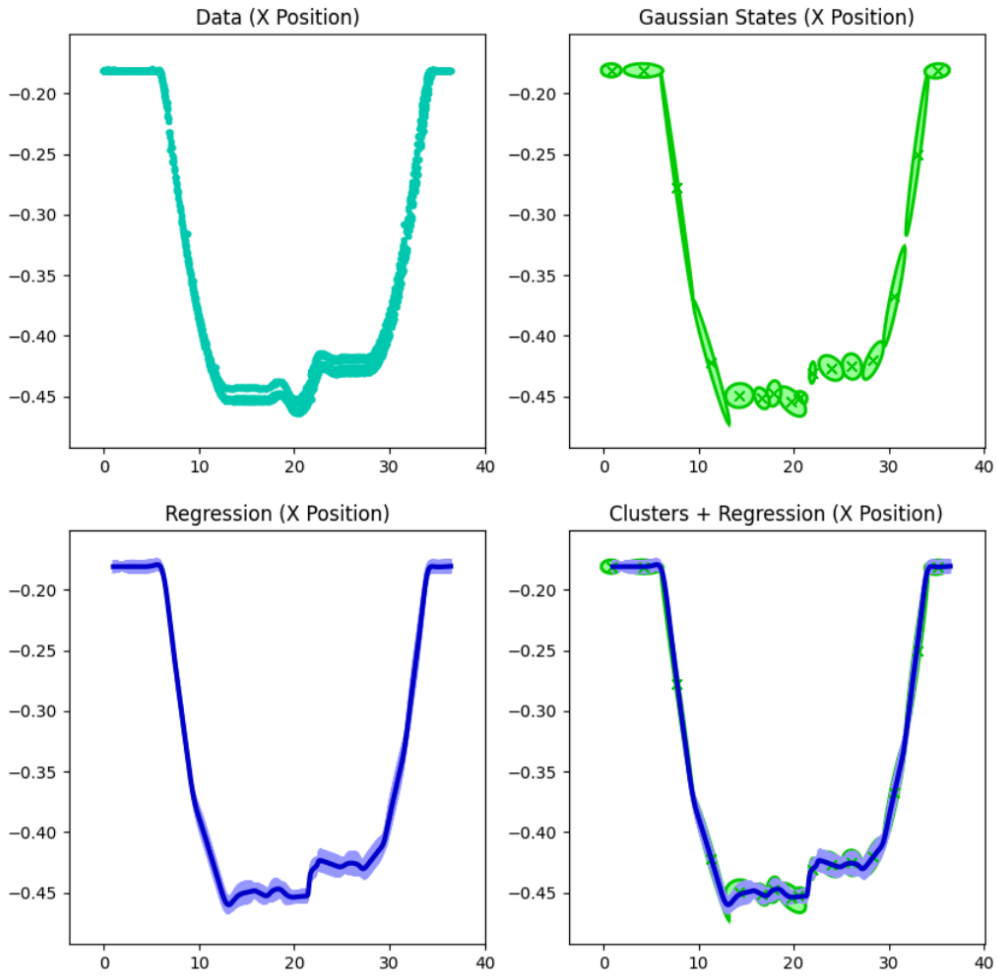


Figure 10: GMM/GMR of X-Position aligned data

Our inputted data is shown on the top left of the figure. The estimated Guassian states (GMMs) are on the top right. The regression (GMR) is on the bottom left. The combination of the GMM and GMR are shown on the bottom right, showcasing the alignment of the data and our estimations. The rest of our figures for GMM_GMR can be found in the Python notebook and in Appendix B.

2) **Identify a method that will find the optimal number of Gaussians. You can use an already available method or develop your own. Explain the method you used/developed in a mathematical way and provide screenshots of the code and the results.**

There are two methods we identified to find the optimal number of Guassians:

- The Bayesian Information Criterion (BIC)
- The Akaike Information Criterion (AIC)

Both methods are showcased in the plot in Figure 11, but we utilized only BIC in determining the optimal amount of Gaussians.

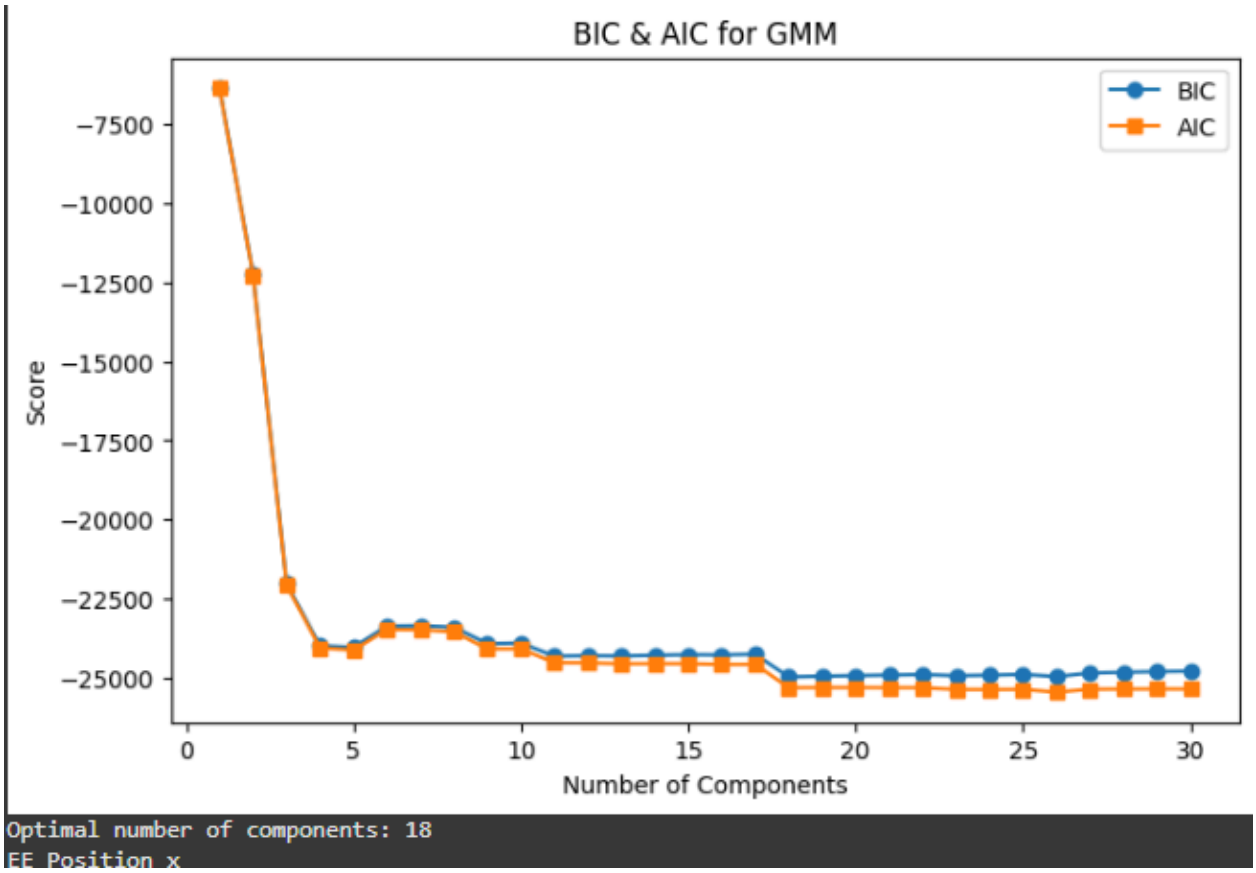


Figure 11: X-position plot of BIC and AIC score against *n* number of components. Optimal amount of components shown below the plot.

We fit the GMM to a number of components *n* (in this case, we used 20 components), utilizing the Expectation-Maximization algorithm to estimate the parameters of the mean and variance, and maximizing the likelihood. After fitting the model, we have the following equations for AIC and BIC as shown in Figure 12.

$$\mathrm{AIC} = 2k - 2\ln(\hat{L})$$

$$\mathrm{BIC} = k\ln(n) - 2\ln(\hat{L}).$$

Figure 12: AIC and BIC equations, where *k* is the number of free parameters and *L* is the likelihood

In the plots, we utilized both to see discrepancies between the models. In the case of the X-Position in Figure 11, the discrepancies between the two were minor. We chose BIC as it chooses a simpler model and we have access to all the data from the provided demonstrations. The code to generate the plot of Figure 11 is shown below.

```
#Credit to ChatGPT for helping out with the code
from sklearn.mixture import GaussianMixture
import numpy as np

def find_optimal_n_components(X, max_components=30):
    """Finds the optimal number of Gaussian components using BIC & AIC."""
    bic_scores = []
    aic_scores = []
    n_components_range = range(1, max_components + 1)

    for n in n_components_range:
        gmm = GaussianMixture(n_components=n, random_state=42)
        gmm.fit(X.reshape(-1, 1))  # Ensure X is 2D
        bic_scores.append(gmm.bic(X.reshape(-1, 1)))
        aic_scores.append(gmm.aic(X.reshape(-1, 1)))

    # Plot BIC & AIC
    import matplotlib.pyplot as plt
    plt.figure(figsize=(8,5))
    plt.plot(n_components_range, bic_scores, marker='o', label='BIC')
    plt.plot(n_components_range, aic_scores, marker='s', label='AIC')
    plt.xlabel('Number of Components')
    plt.ylabel('Score')
    plt.legend()
    plt.title("BIC & AIC for GMM")
    plt.show()
```

```
    # Return the best n_components (based on lowest BIC)
    best_n = n_components_range[np.argmin(bic_scores)]
    print(f"Optimal number of components: {best_n}")
    return best_n

# Go through each trajectory to determine optimal amount of components.
for i in range(len(newGmmStack)):
  optimal_n = find_optimal_n_components(newGmmStack[i])
  print(indexes[i])
```

3) Investigate if GMM/GMR is better to use piecewise trajectories or complete trajectories. Explain which approach is better and why.

To determine our piecewise trajectories, we split each trajectory into three distinct trajectories:

1. Moving the arm to the object
2. Grasping the object and moving towards a new location
3. Placing the object and moving to its final position

In essence, the first part consists of pre-actuation of the gripper, the second part consists of the gripper manipulating the object, and the third part consists of post-deactuation of the gripper. We show the plots of the gripper position in Figure 13.

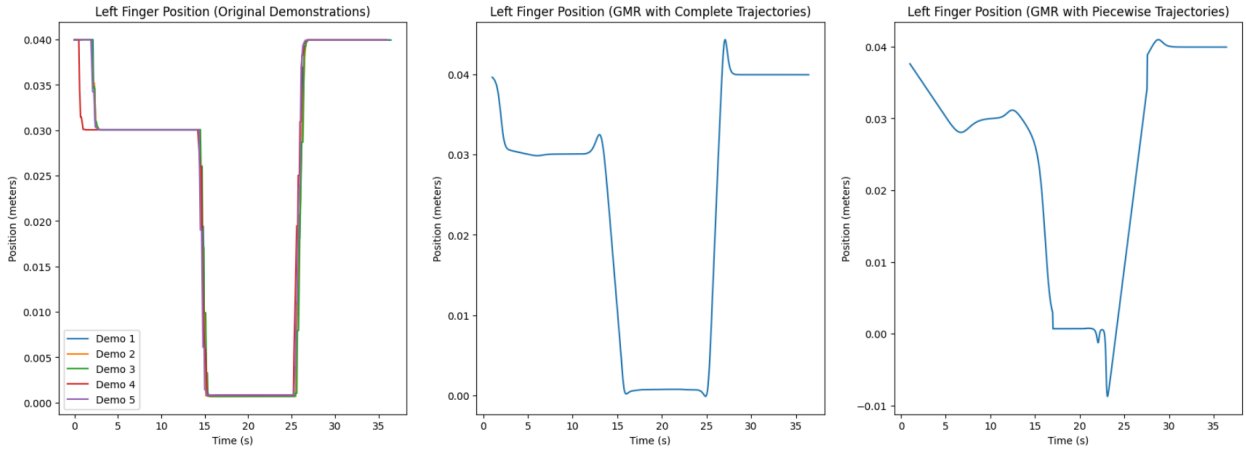


Figure 13: Left finger Position. (Left) Original demonstrations trajectory, (middle) GMM/GMR on complete trajectory, (right) GMM/GMR on piecewise trajectory

In the case of the gripper, the movement is smooth and each demonstration follows a similar trajectory, so the phases are not too different from each. Visually, the complete trajectory matches the original demonstrations much better than the piecewise trajectories. This can also be due to how we split the trajectories. Now let's look at the end-effector's Y-Orientation in Figure 14.

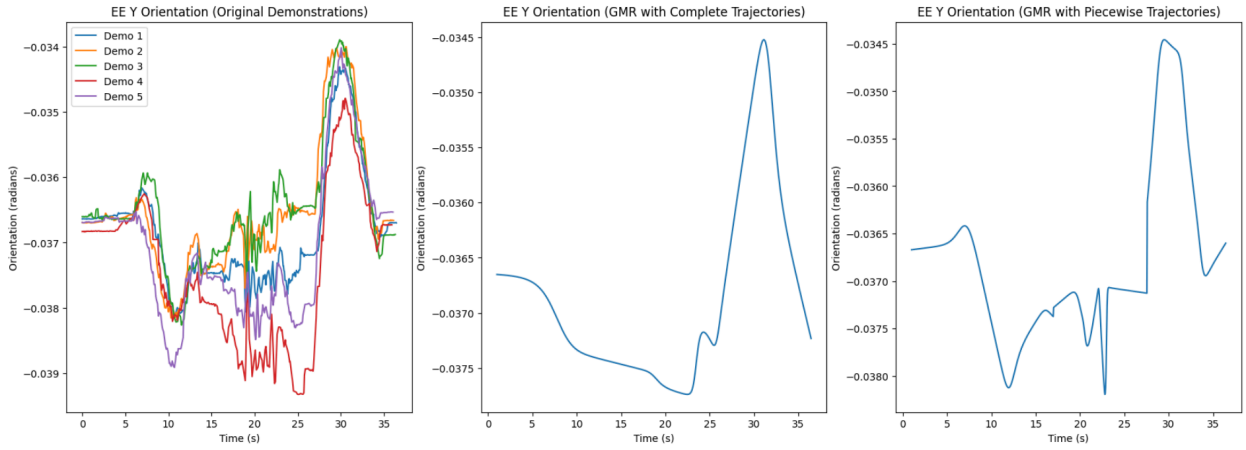


Figure 14: End Effector Y-Orientation. (Left) Original demonstrations trajectory, (middle) GMM/GMR on complete trajectory, (right) GMM/GMR on piecewise trajectory

The trajectory for the Y-Orientation is much more sporadic and varied compared to the Left Finger position. As the trajectories are not smooth across all demonstrations, the GMM/GMR with the complete trajectories does not capture the trajectories as effectively as a smooth trajectory. The piecewise trajectory captures the varied and sporadic nature of the original demonstrations, showcasing that the piecewise approach is better in this case.

In general, it appears that using complete demonstrations to train the GMM/GMR yields better results. The major reason for this in our particular case is the distribution of samples in our demonstrations. In order to perform GMM with piecewise data, we need to split the demonstrations based on index to guarantee that each piece has a set number of samples; however, the distribution of samples across time varies greatly between demonstrations. This means that if we take the first 300 samples from each demonstration, this might mean taking the first 15 seconds of data from demonstration 1, but the first 20 seconds of data from demonstration 2. This issue leads to complications when training GMM/GMR.

Comparison plots for all trajectories can be found in Appendix C.

D. Adaptation to new environments

1) **Identify a method or develop your own method that can generate a trajectory that has different starting and goal positions and avoid obstacles. Explain the method you used/developed in a mathematical way and provide screenshots of the code and the results with different cases. Discuss the limitations of your approach.**

We utilized Dynamic Movement Primitives (DMPs) to generate a trajectory with different starting and end goal positions. We used the package found here: https://github.com/dfki-ric/movement_primitives/tree/main. Initially, we utilized the spatial scaling python file provided in the package to understand how the trajectory is formed. We initialize an array new_start and new_goal, which contains the new starting and goal positions of our trajectories. We utilized our 3 piecewise trajectories to modify our first trajectory with our new_start and our final trajectory with our new_goal. The equation for adapting our shifted trajectory start and goal is shown below:

```
Y_start_shifted = Y - start[np.newaxis] + new_start[np.newaxis]
Y_goal_shifted = Y - goal[np.newaxis] + new_goal[np.newaxis]
```

where Y is our stacked matrix containing the trajectories of x,y,z position. The function np.newaxis increases the dimension of the arrays, allowing the shapes to match to perform arithmetic operations.

By using a scalable DMP algorithm, we were able to take our trained GMR output and form a new regression with entirely new start, pick, place, and end positions. In order to specify all 4 new positions, we had to use the output from our piecewise GMR. Since the piecewise GMR is split into three segments, we were able to adapt to all four new positions at the same time. This is a limitation of this method, because without using the piecewise GMR, we would not be able to adapt to the new pick and place positions, meaning the object would always need to be in the same position.

Overall, the results we gathered from our method were encouraging. We applied this method to EE X Position, EE Y Position, and EE Z Position. The results can be seen below in Figure 15. As you can see, in all three plots, we are able to reach all of the position setpoints. For EE X Position and EE Y Position, the general shape of the motion is also maintained very well, with only small variations. However, in the case of EE Z Position, there is a significant difference in the shape of the motion, despite the goal positions being reached. We attempted to resolve this issue by tuning the parameters of the DMP algorithm, but were unable to produce better results. Because of this, another major limitation of this approach is the ability to maintain the shape of the original curve. If this is an important design factor, this method may not be the safest option.

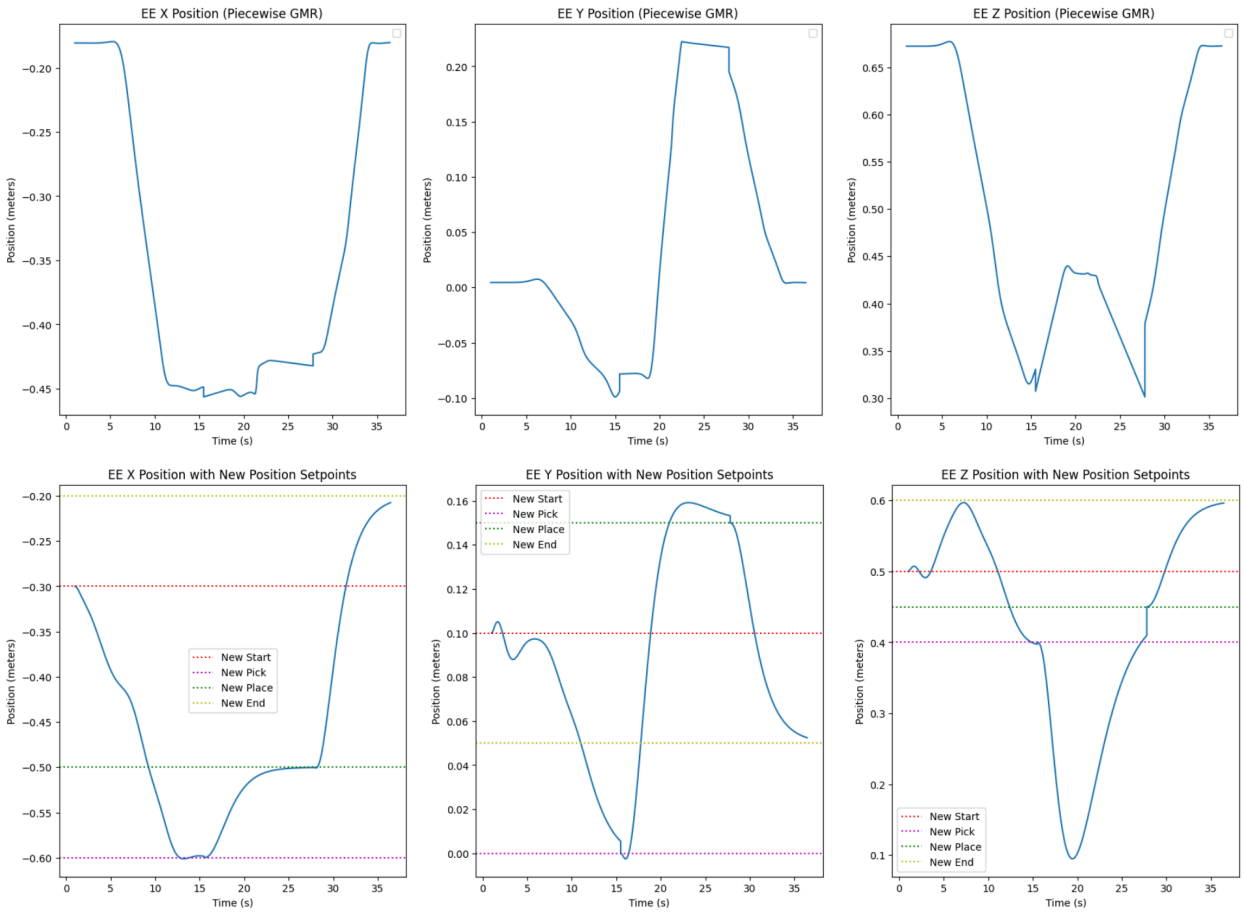


Figure 15: Comparison between original GMR trained output and DMP trained output with new position setpoints

For obstacle avoidance, we utilized the file found in the same package github.com/dfki-ric/movement_primitives/blob/main/examples/plot_dmp_potential_field.py. We initialized our start and goal position from the new trajectory found above in the X-Y positions, then defined an obstacle in a 2d plane for obstacle avoidance as shown in Figure 16. The obstacle was placed in between the start and pick locations, so these plots show only the first segment of the motion, not the entire movement.

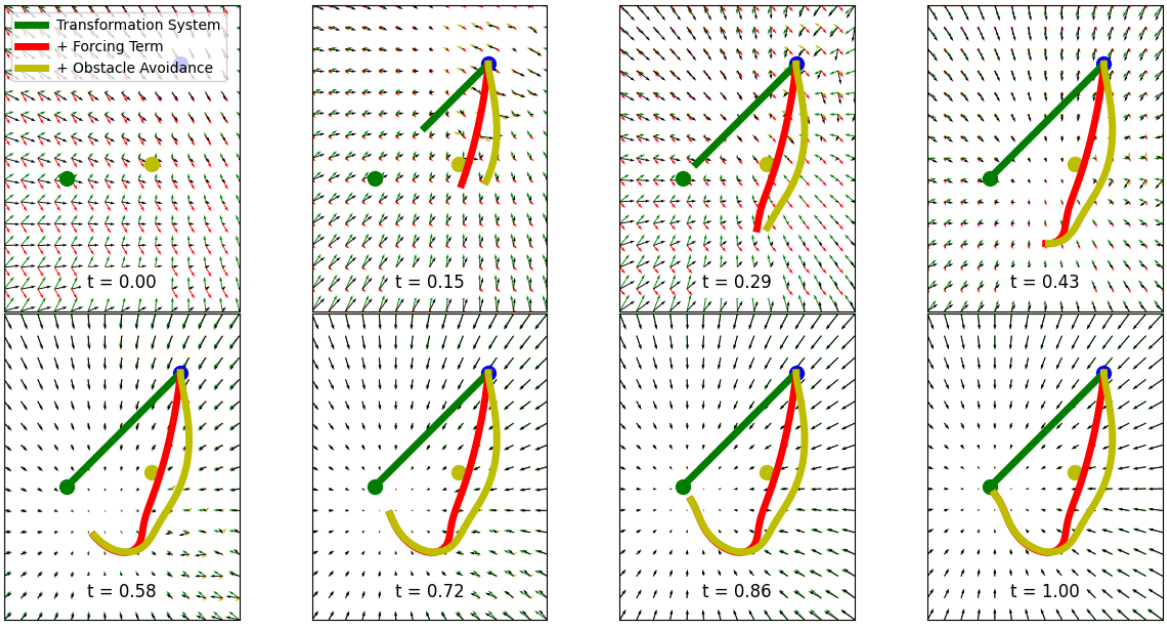


Figure 16: Demonstration of the potential field for obstacle avoidance.

The plot showcases the potential field, or which direction the transformation system can go, with the addition of the forcing term, which is the learned shape. The obstacle avoidance plot are the coupling terms.

Appendix A: Paired Trajectory Reference and Query Plots

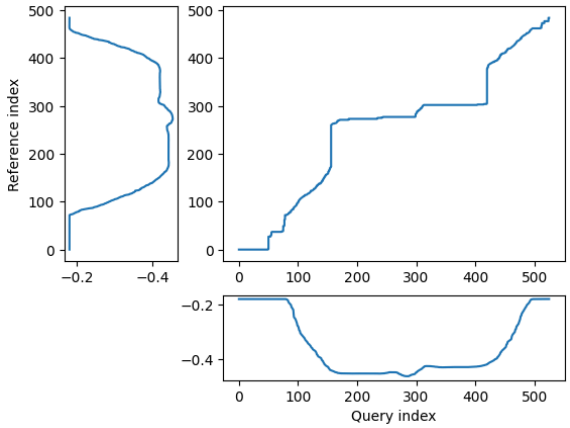


Figure A1: Alignment between Demo 1 and Demo 2

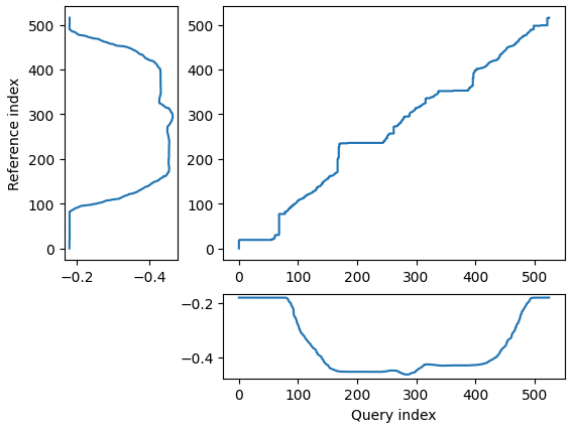


Figure A2: Alignment between Demo 1 and Demo 3

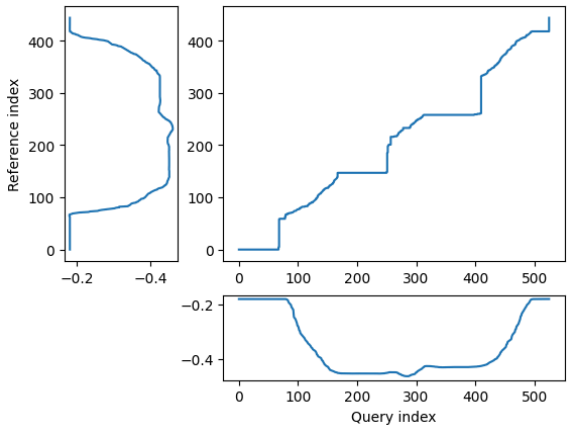


Figure A3: Alignment between Demo 1 and Demo 4

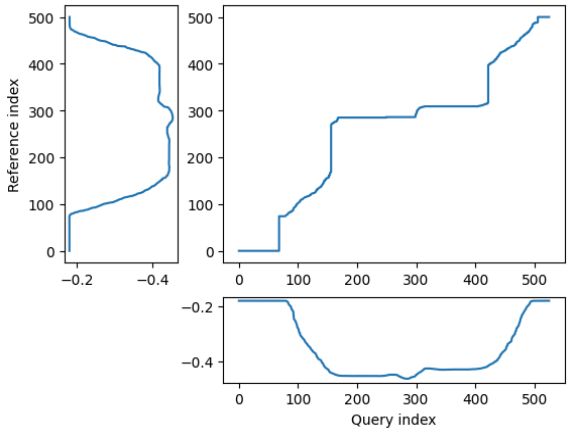


Figure A4: Alignment between Demo 1 and Demo 5

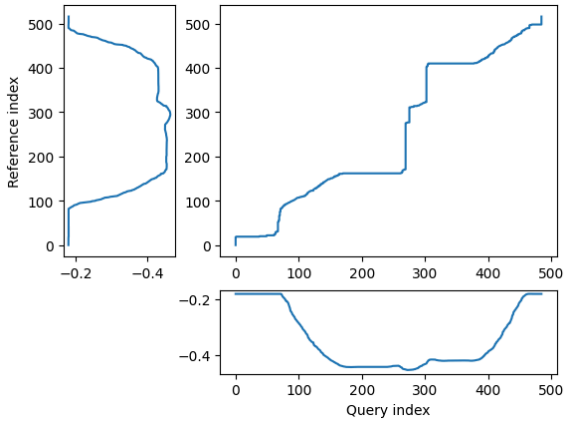


Figure A5: Alignment between Demo 2 and Demo 3

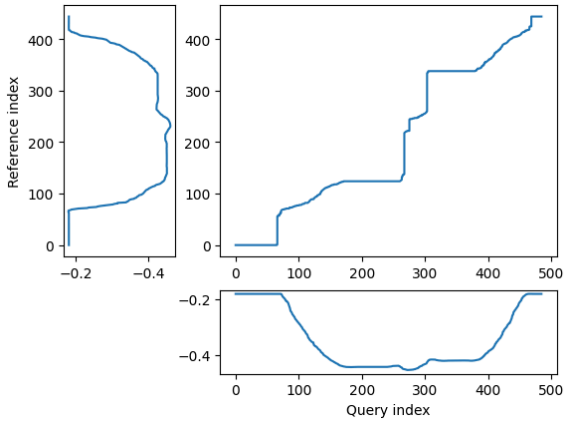


Figure A6: Alignment between Demo 2 and Demo 4

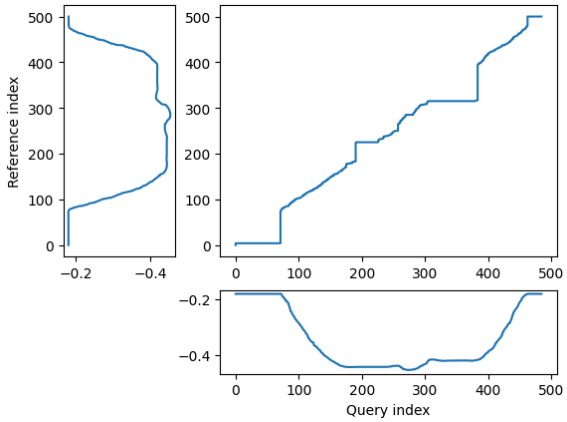


Figure A7: Alignment between Demo 2 and Demo 5

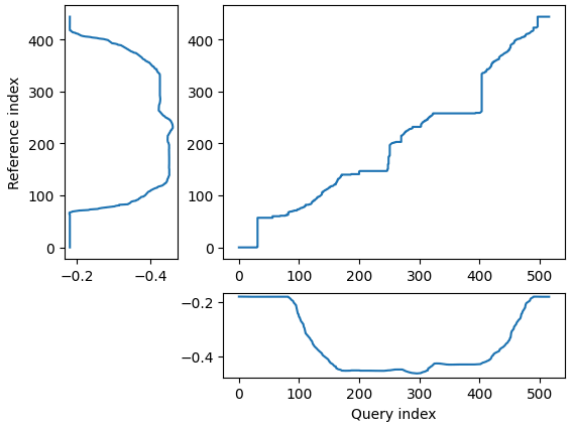


Figure A8: Alignment between Demo 3 and Demo 4

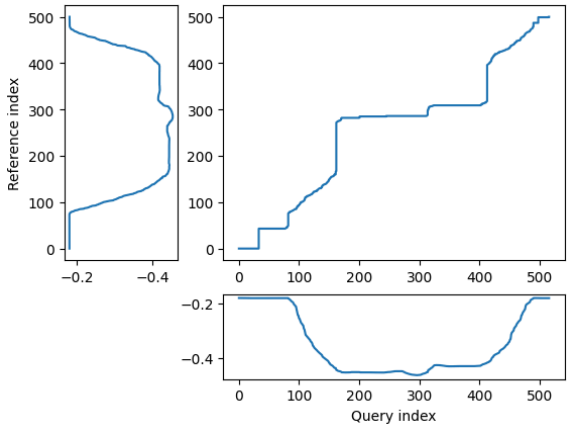


Figure A9: Alignment between Demo 3 and Demo 5

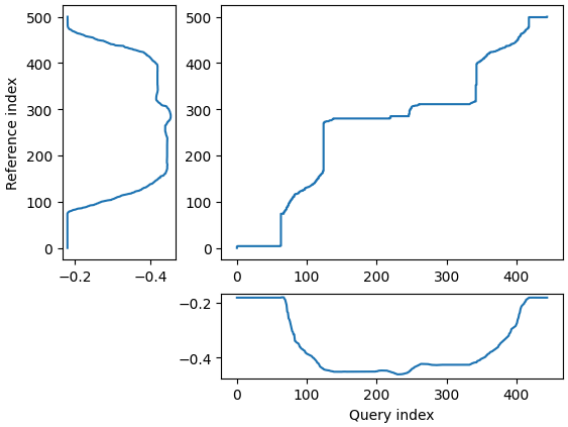


Figure A10: Alignment between Demo 4 and Demo 5

Appendix B: GMM/GMR of Each Trajectory Component

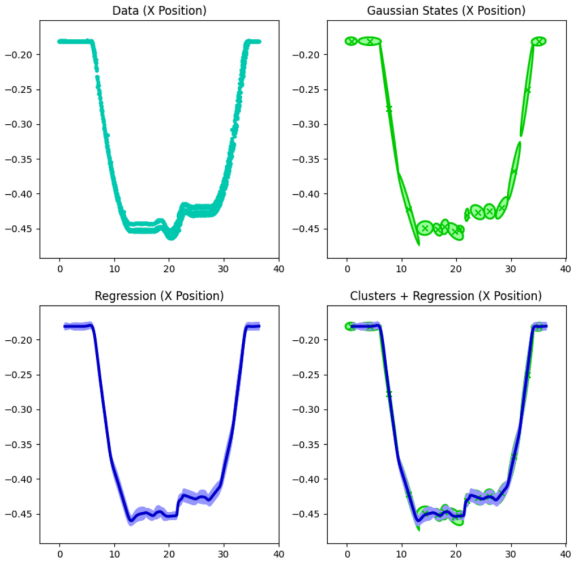


Figure B1: X-Position GMM/GMR

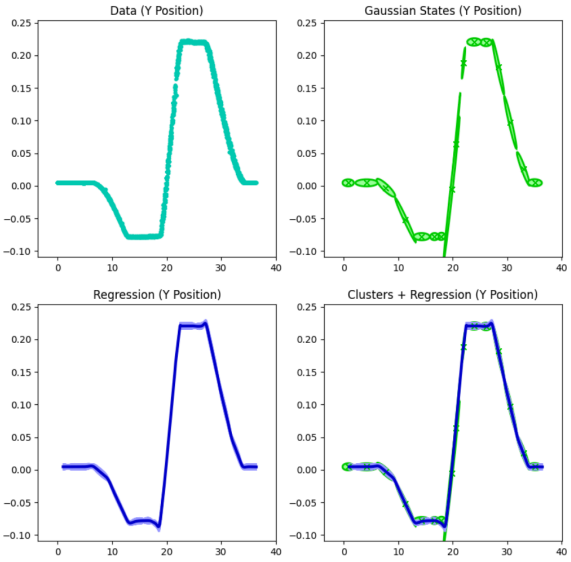


Figure B2: Y- Position GMM/GMR

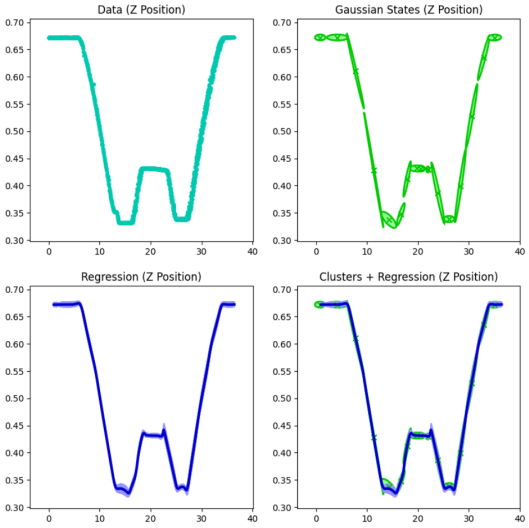


Figure B3: Z-Position GMM/GMR

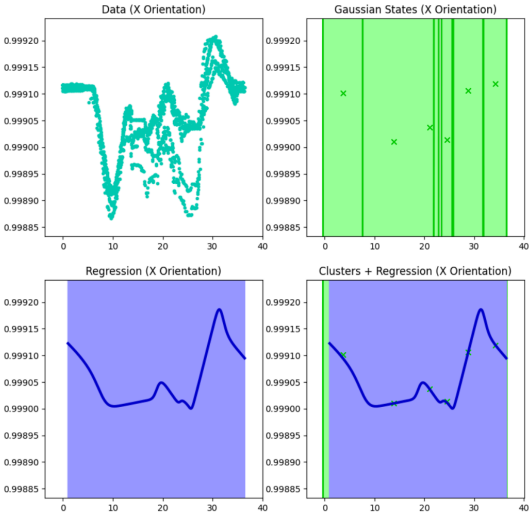


Figure B4: X-Orientation GMM/GMR

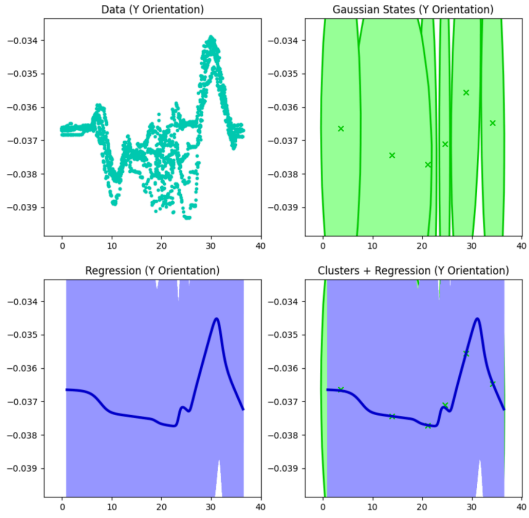


Figure B5: Y-Orientation GMM/GMR

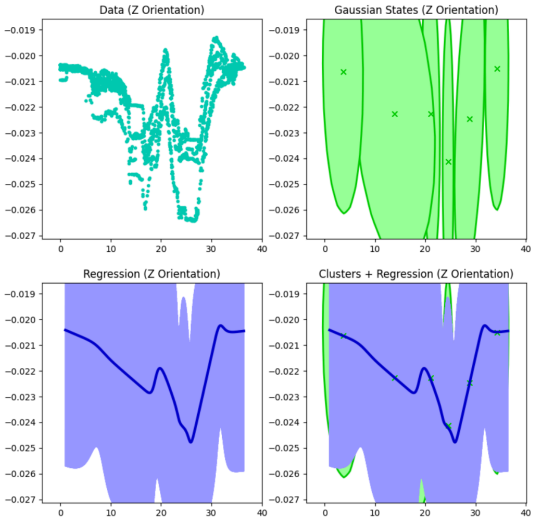


Figure B6: Z-Orientation GMM/GMR

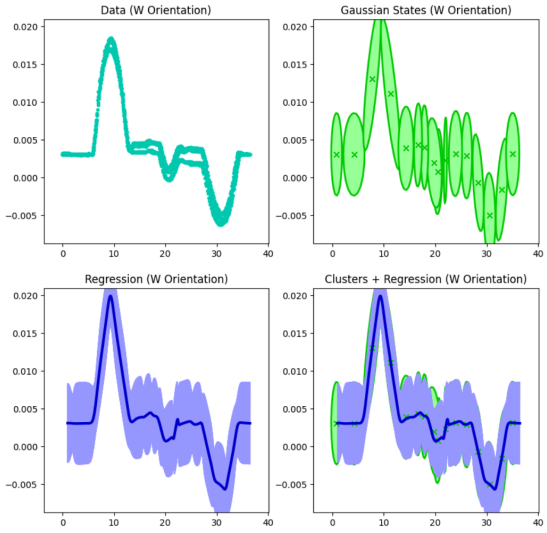


Figure B7: W-Orientation GMM/GMR

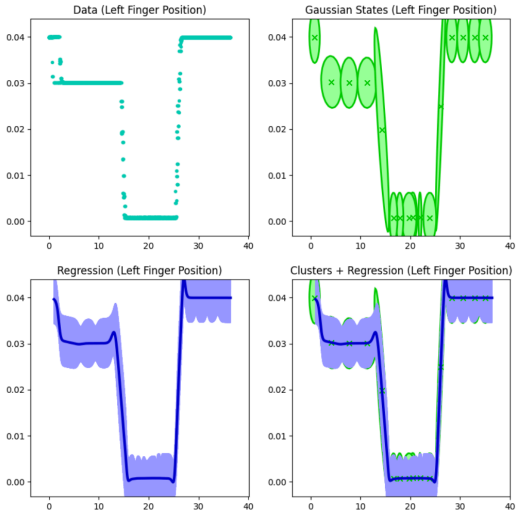


Figure B8: Left Finger Position GMM/GMR

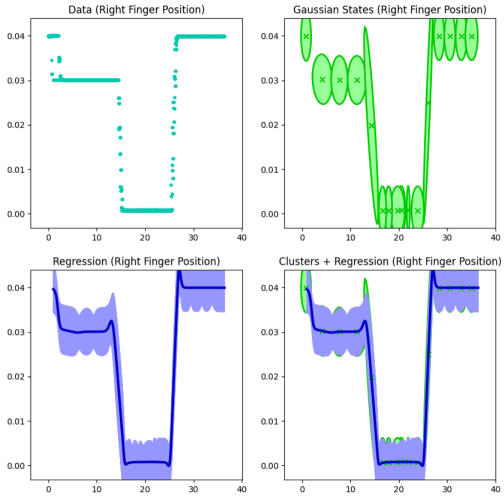


Figure B9: Right Finger Position GMM/GMR

Appendix C: Comparison of Original, Complete, and Piecewise Trajectories.

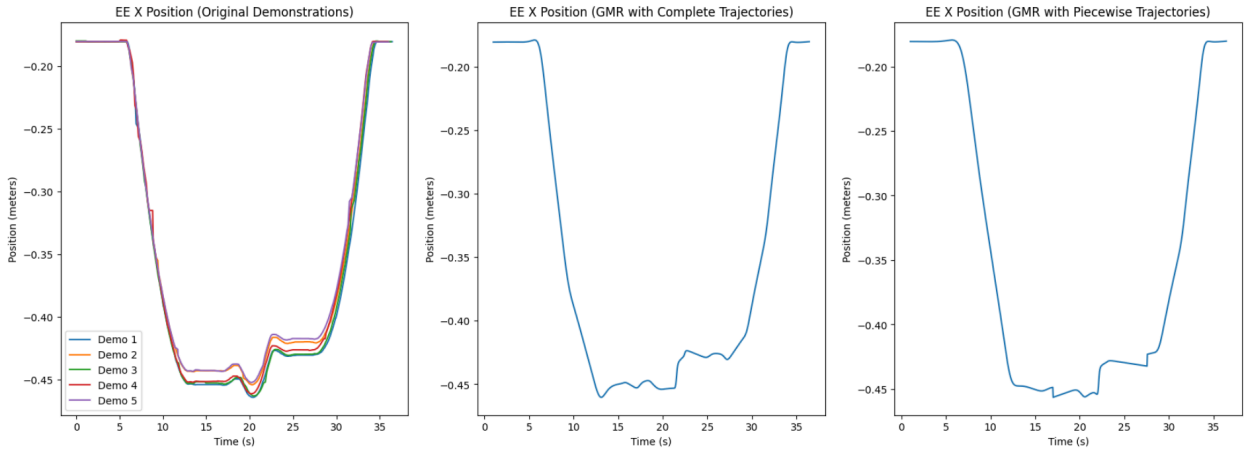


Figure C1: X-Position Comparison

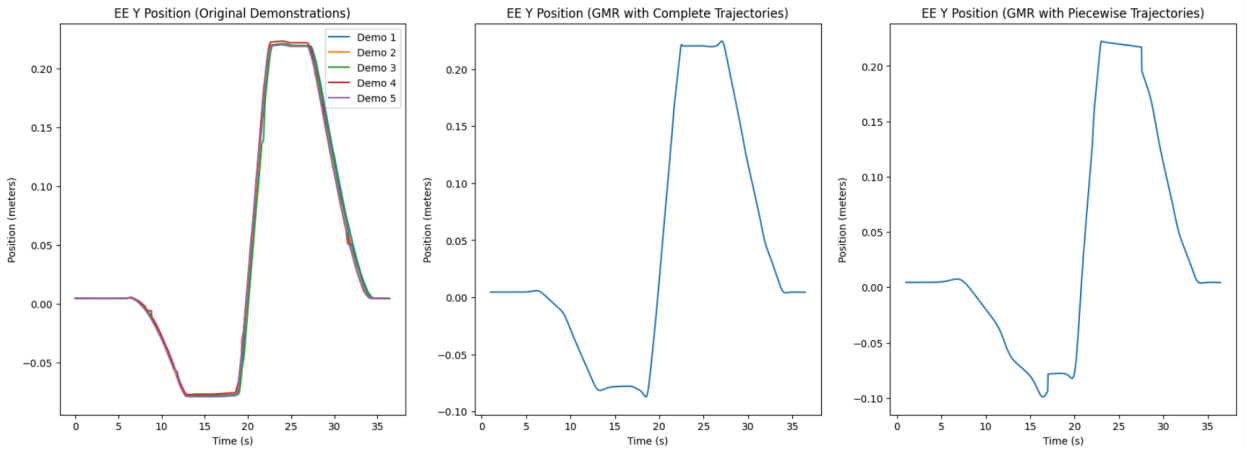


Figure C2: Y-Position Comparison

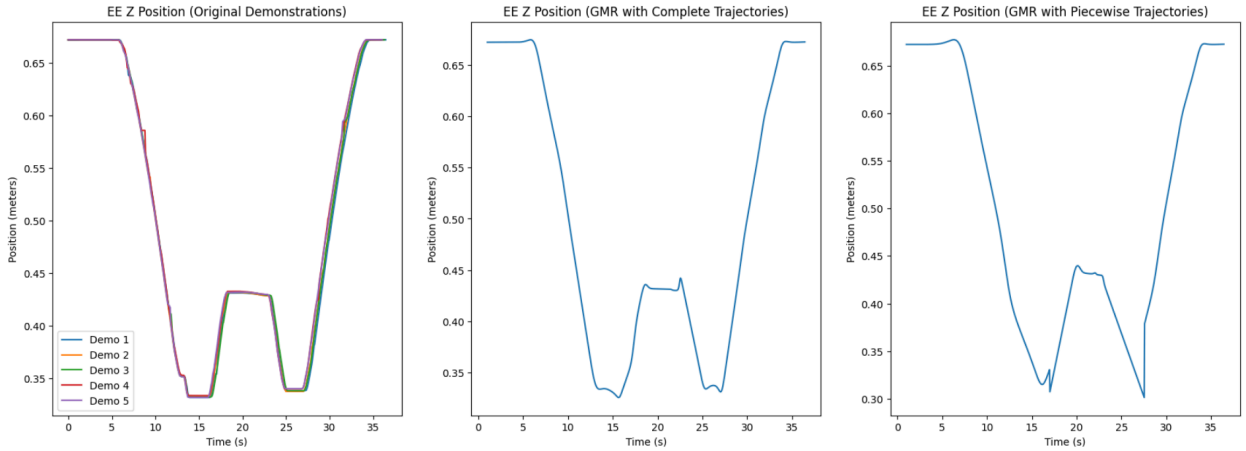


Figure C3: Z-Position Comparison

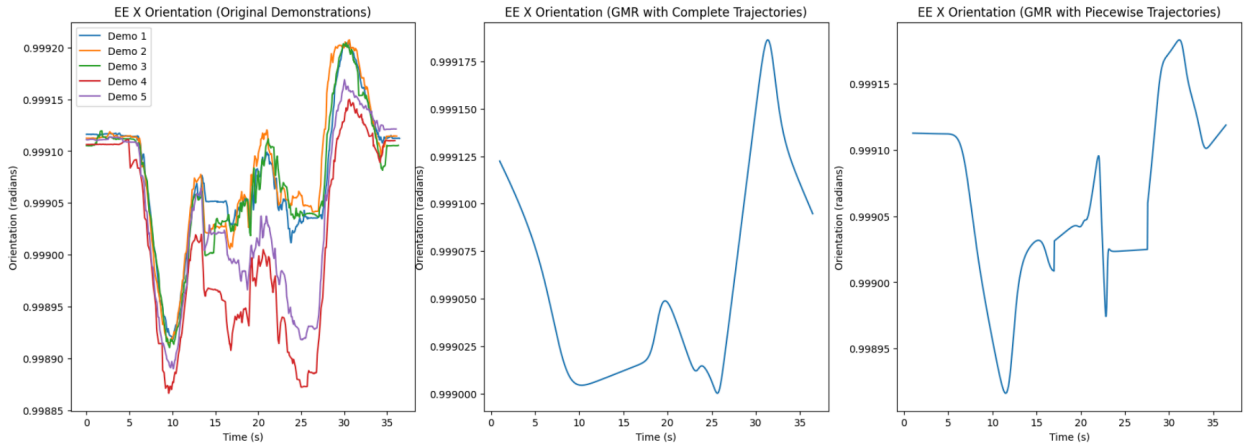


Figure C4: X-Orientation Comparison

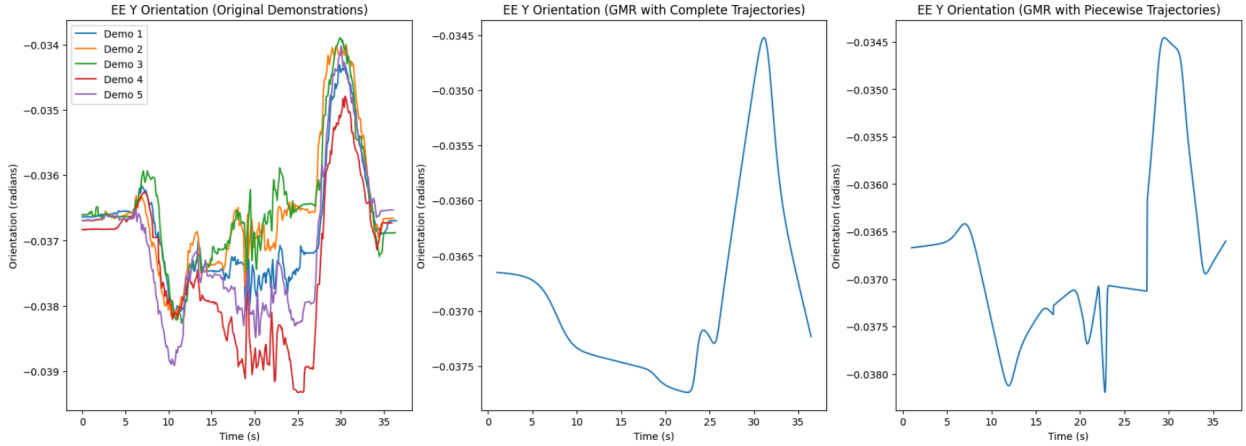


Figure C5: Y-Orientation Comparison

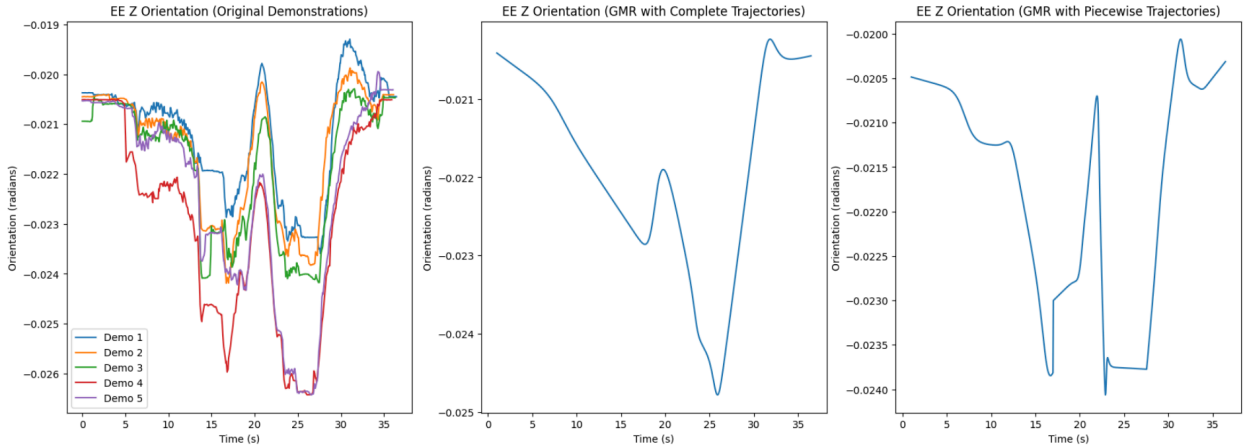


Figure C6: Z-Orientation Comparison

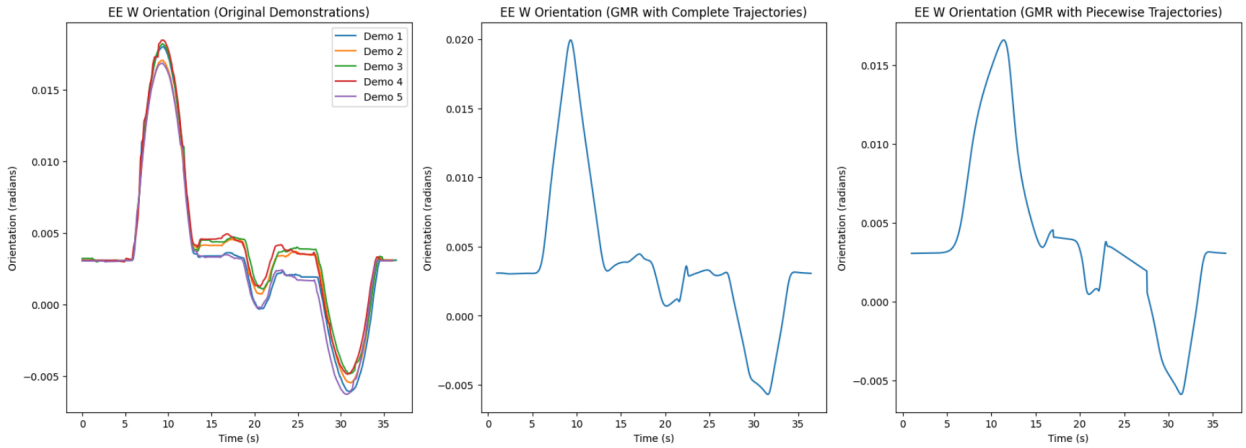


Figure C7: W-Orientation Comparison

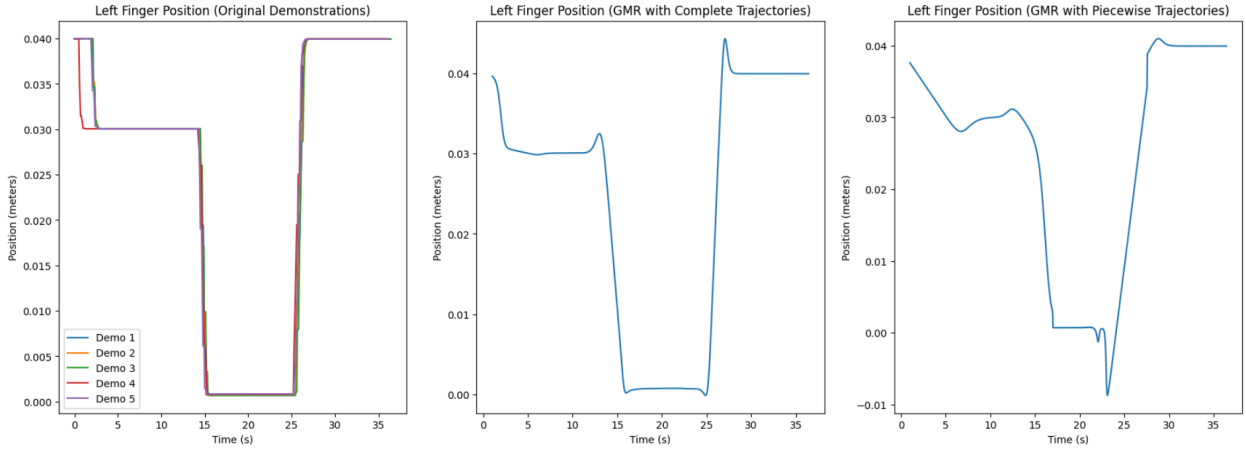


Figure C8: Left Finger Position Comparison

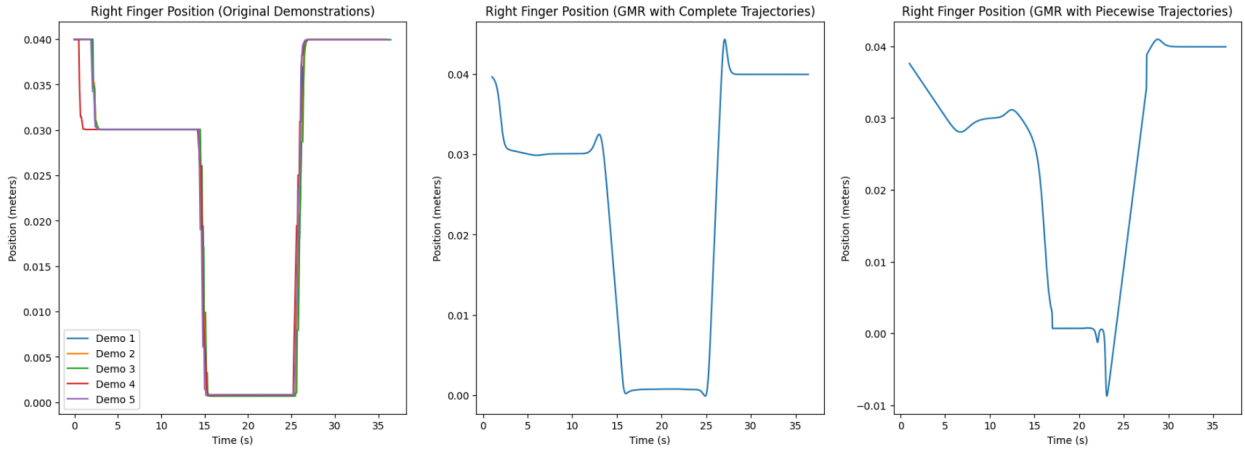


Figure C9: Right Finger Position Comparison