University of Southern Denmark
Mærsk McKinney Møller Institute

2. Semester - Master in Advanced Robot Technology

Blend of several skills

Kode: RD4-PRO **Group nr. 4** **Date: 26-05-2023**

Supervisor: Yitaek Kim - yik@mmmi.sdu.dk

Emil Siggi Asgeirsson
Emasg18

Ibrahim Haj Yousef
Ibyou18

Kasper Gjødesen Mikkelsen
Kamik18

Abstract

This project aims to blend several robot skills. In the project, two tasks each with two skills are implemented with suitable Learning from Demonstration (LfD) methods. The methods for the LfD are Dynamic Movement Primitives (DMP) and Gaussian Mixture Regression (GMR).

The blend of the skills showcased improvement in smoothness by reducing the acceleration, and overall optimization in the movement of the robot. Furthermore, the quadratic programming (QP) and the variance obtained from the GMR proved to be beneficial for the blending, by reducing the distance between skills. This resulted in more smooth movements during the blending of skills in the tight space positions, such as between insert and extract.

The use of DMP demonstrated the blend's ability to adapt if a skill position has changed. The blend adapted smoothly and proved to operate generically to the changes. Furthermore, mixing DMP and GMR works seamlessly using the implemented blend.

Keywords: Simulation, admittance control, Trajectory Control, Blend of Several Robot skills.

Project description

Optimization to Blend Several Robot Skills

Learning from Demonstration (e.g. Dynamic Movement Primitives (DMP)) has been getting attention in the robotics fields to reproduce human skills and transfer them into a robot. However, it is not easy to improve or generalize the learned behaviors unless we demonstrate human skulls again. Therefore, this challenge can be resolved by reusing and blending previous data to create new behaviors.

The project will cover the following topics:

- Choose and implement suitable LfD methods to a task (possibly two).
- Implementation optimization methods to blend the skills (e.g. Quadratic programming)
- Evaluating the learned paths from each other.

Contents

1 Introduction

Learning from Demonstration (LfD) has gained significant attention in robotics, for reproducing human skills on robots instead of the old way of manually programming the robot. Generalizing learned behaviors has been a challenging task, as it requires repeating the demonstration process and blending multiple skills.

LfD is a technique that allows robots to reproduce human skills by observing a human demonstration.

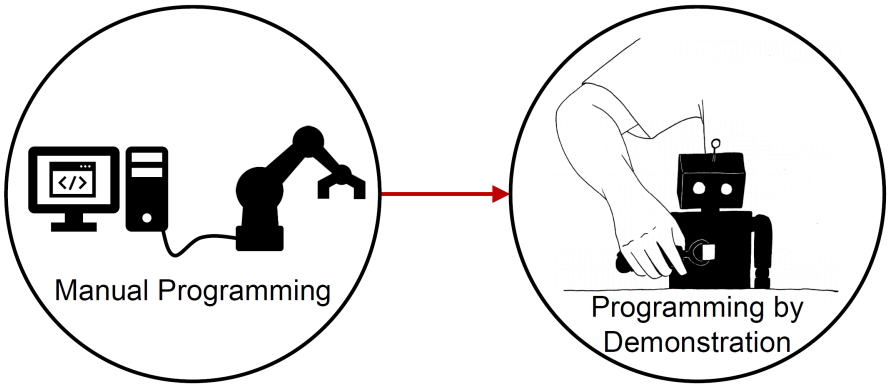


Figure 1: *Image from "Learning from Demonstration By Iñigo Iturrate".*

One of the problems to be assessed regarding the LfD is to ensure that the robot is compliant and capable of recording accurate demonstrations.

There are two popular LfD methods for modeling and reproducing demonstrated movement, which are Dynamic Movement Primitives (DMP) and Gaussian Mixture Regression (GMR).

Blending multiple skills is a technique that allows robots to combine multiple skills to achieve a desired goal. This technique is useful for robots to perform complex tasks that require multiple skills. However, blending multiple skills is no easy task, and is highly dependent on the demonstrated task in order to achieve a satisfying result. The project proposes a method for blending multiple skills using parabolic blend and Quadratic Programming (QP) based on the Dynamic Movement Primitives (DMP) and Gaussian Mixture Regression (GMR). The proposed method is evaluated on a simulated and physical robot that performs a pick-and-place task, and an insertion and extraction task.

2 Problem statement

The task is to blend two different LfD methods to achieve a smooth trajectory. The objective is to create a seamless path by combining abilities from various LfD techniques. These abilities should be mergeable in a manner that enables the robot to execute them consecutively.

The success criteria are to have a robot that can blend two different LfD methods and use the skills in a sequence as well as having a smooth trajectory and being able to generalize the skills. Finally, mix the generalized skills independent of the order or presence of the skills.

To achieve generalized skills from the LfD and be able to adapt to changes in a dynamic environment, an optimization of the path using an LfD approach such as DMP is used. In a static environment where multiple demonstrations are recorded, a GMR is a strong LfD approach to handle noise and optimize a path.

With these two methods, the goal is to be able to select which one is optimal for a specific skill and be able to interchange between the two methods for all skills. Finally, the project aims to blend the skills in order to remove point-to-point / discrete-like movement and achieve smooth trajectories between skills.

3 Applied theory and implementation

This section will describe the theory behind the implementation of the admittance controller and the skills used in this project. The admittance controller is used to control the robot in the cartesian space, by following a human-taught trajectory, creating a skill. The skills are complex movements that the robot must perform, to achieve a task.

However, to create the skills the LfD methods DMP and GMR are used. And the final part of the implementation is the blending of the skills, to create a smooth trajectory. This part blends the skills together, to create a sequence of skills, that the robot can perform, which are smooth and continuous.

3.1 Admittance Control

In this project, a UR5e robot is provided to achieve the desired objective of the project. The robot has a freedrive option that employs an impedance controller. However, operating the robot can be difficult due to its stiffness, particularly in situations where high precision is required. Unfortunately, Universal Robots does not provide any access to the impedance controller which means that there is no way to adjust its parameters in order to reduce its stiffness. Hence, it is essential to have admittance control, a technique that enables the movement of a cooperative robot to be controlled by applying an external force.

Figure 2 shows an overview of how the admittance controller works.

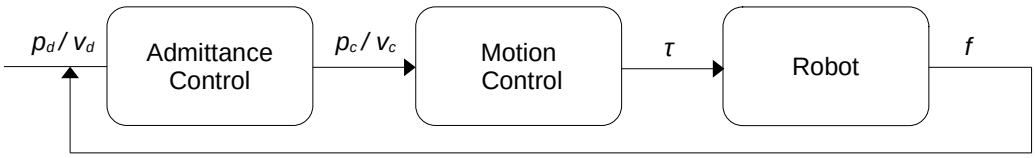


Figure 2: *Admittance control.*

On top of motion control, admittance control is applied. It requires a reference position and input from a force sensor from the robot. Hence, it should be emphasized that motion control is not being implemented in this project as the UR5e already provided it.

However, in the admittance control, the robot works as a spring-mass-damper system. Where the mass represents the robot's inertia, the spring represents the robot's stiffness, and the damper represents the robot's damping effect.

$$M_p \Delta \ddot{p}_{cd} + D_p \Delta \dot{p}_{cd} + K_p \Delta p_{cd} = f \tag{1}$$

Equation 1 represents the transnational part of the admittance controller. Where:

- M_p is a 3×3 gain matrix, representing the inertia of the robot.
- D_p is a 3×3 gain matrix, representing the damping effect of the robot.
- K_p is a 3×3 gain matrix, representing the robot's stiffness.
- Δp_{cd} is the difference in position between the compliant frame and reference frame.
- $\Delta \dot{p}_{cd}$ is the difference in velocity between the compliant frame and reference frame.
- $\Delta \ddot{p}_{cd}$ is the difference in acceleration between the compliant frame and reference frame.
- $f \in \mathbb{R}^3$ is the force given in the tool frame.

A decent D_p and K_p gain matrix are selected to achieve desired performance. $K_p = eye(10)$, $D_p = eye(25)$. With K_p and D_p, the gain matrix M_p can be computed by the following formula.

$$M_p = \frac{K_p}{\omega} \tag{2}$$

where $\omega = \frac{1}{t_r}$ [Hz], ω is the natural frequency and t_r is the raising time [s], which is selected to be 1.0 [s].

The orientational part also employs the admittance control which operates based on the same principle as the spring-mass-damper system. The orientational component of the admittance controller is demonstrated in equation 3.

$$M_o \Delta \dot{\omega}^d_{cd} + D_p \Delta \omega^d_{cd} + K'_o \epsilon^d_{cd} = \mu^d \tag{3}$$

Where:

- M_o is a 3×3 gain matrix, representing the inertia of the robot. Equation 2 is used to calculate the M_o where t_r is selected to be 0.9 [s].
- D_o is a 3×3 gain matrix, representing the damping effect of the robot. D_o is selected to be $eye(0.5)$.
- K'_o is a 3×3 gain matrix, representing the rotational stiffness of the robot. The stiffness matrix is given as follows $K'_o = 2E^T((\eta_{cd}, \epsilon^d_{cd}))K_o$, where K_o is the stiffness in Euler angle representation and $E(\eta, \epsilon) = \eta I - S(\epsilon)$. Where K_o is selected to be $eye(0.1)$.
- $\Delta\epsilon^d_{cd}$ is the imaginary part of the quaternion between the compliant frame and reference frame. It is obtained as follows $\epsilon^d_{cd} = \eta_d\epsilon_c - \epsilon_c\eta_d - S(\epsilon_c)\epsilon_d$
- $\Delta\dot{p}_{cd}$ is the difference in angular velocity between the compliant frame and reference frame.
- $\Delta\ddot{p}_{cd}$ is the difference in angular acceleration between the compliant frame and reference frame.
- $\mu^d \in \mathbb{R}^3$ is the torque to the end effector given in the desired frame.

The above equation 3 uses quaternion as a representation of the rotation due to the ability to prevent singularities. To acquire the quaternion, angular velocity can be integrated, as demonstrated in the equation. 4.

$$q(t + dt) = exp\left(\frac{dt}{2}\omega^d_{cd}(t + dt)\right) \cdot q(t) \tag{4}$$

Where

$$exp(r) = (\eta,\ \epsilon) = \left(cos||r||,\ \frac{r}{||r||}sin||r||\right)$$

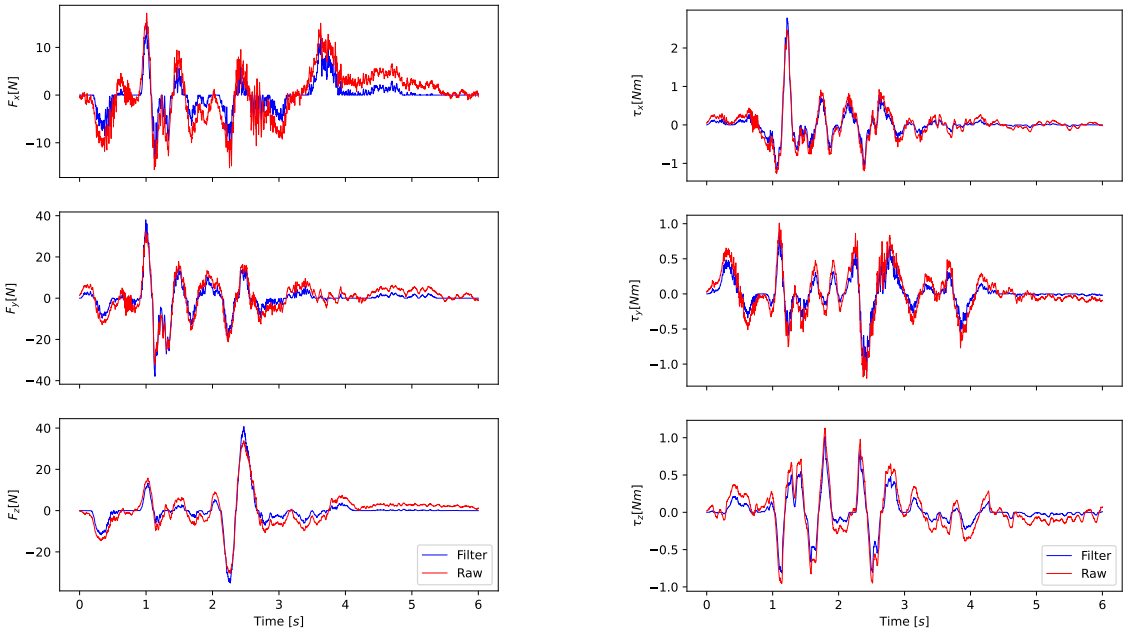
$exp(r)$ is the exponential map of the quaternion. The admittance control for the translation and rotation work separately and they are used to record a demonstration of the desired trajectory. The two video recordings, [7] and [8], demonstrate a comparison between the

performance of the admittance control and impedance control (freedrive). It is obvious that the admittance control is easier to operate than the impedance control as it is less stiff and more precise.

3.1.1 Filter

The admittance control is operating with data from a six-degree-of-freedom (DoF) force torque sensor. This sensor provides the forces and torques from the tool in the base frame. However, this sensor has a lot of noise in all six DoF. In the translational, it receives error forces up ±2 [N]. This causes the admittance control to behave differently than anticipated, and if the robot is left in an unsupported position after a demonstration, it will begin to wander.

Because of this, a filter is applied to the force toque sensor before being used in the admittance control, and the improvement is visualized in figure 3.



(a) *Translational component.* ***(b)*** *Orientation component.*

Figure 3: *Visualization of the force torque sensor with and without a filter.*

The filter is simple and uses the $f(x) = log(x)$, where values less than 1 are set to 0. The axis is scaled appropriately to reduce forces greater than the threshold of $x > 1$ after scaling and removing forces $x \leq 1$. The is then scaled back to normal, and these changes

can be observed in figure 3. Using this filter makes controlling the robot smoother and easier, and the robot TCP will stay in the unsupported position where it is left.

3.2 Selecting skills

Two objects are used to create the demonstrations, see the green object in figure 4. These objects are mounted differently on the table, and a cylinder is created to fit in the holes of the objects, the red cylinder in figure 4.

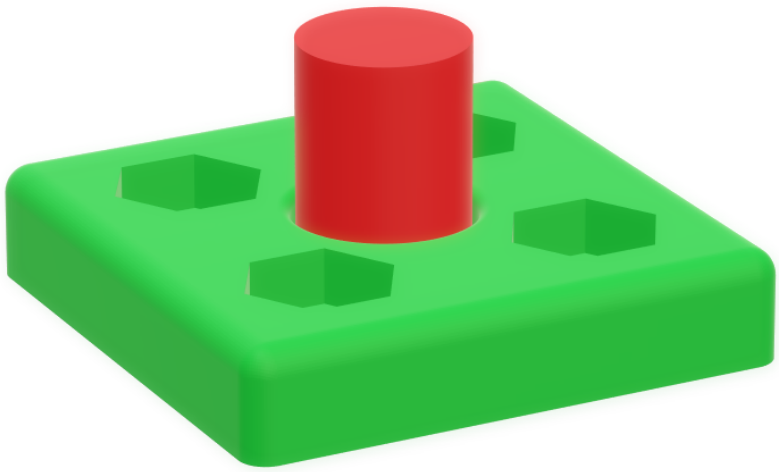


Figure 4: *The object, green, and the cylinder to be inserted and extracted, red.*

Object A is placed on the side of the table, where the robot places a cylinder in the hole of the object and extracts it again. This creates two skills, where the first skill is insertion and the second is extraction. During these skills, the gripper will stay closed holding the cylinder. These skills are chosen to evaluate how well the DMP and GMR algorithms could handle translation and orientation, which are necessary to perform these skills. This has some challenging aspects because the robot does not provide quaternions, but instead an axis-angle. These skills would therefore exploit the issues when an angle change sign.

Object B is placed on the top of the table, and does not require any orientational movement. At this object, 2 skills are created, in the form of a pick-and-place skill. The pick is relatively easy since it will grab around the object and move along the z-axis till the

cylinder is free from the object. Furthermore, the gripper closes around the object once it reaches the position, and movement in the gripper is possible. A pick can therefore accept some movement. However, a pick offset from the center of the cylinder will cause an issue for the other skills when inserting the cylinder into the objects. The place skill requires higher precision to get the cylinder into the object before opening the gripper. These skills at object B differed from the skills at object A by the orientational part, and how precise the skills have to be.

All skills, as seen in the videos [9], [10], [11], and [12], are demonstrated 25 times due to the GMR, and the demonstrations are done slowly to ensure a smooth and controlled movement. Around the holes of the objects, the movement where slower due to the constraints, and the demonstrations have to be very precise at these locations. The four distinct skills are recorded using the admittance control discussed in section 3.1.

3.3 LfD methods

Dynamic Movement Primitives (DMP) is the algorithm used for the dynamical system-based, and Gaussian Mixture Regression (GMR) is the probabilistic model. These are widely utilized techniques in robotics and motion planning, where LfDs is desired. These LfDs often learn complex tasks by observing human demonstrations. These demonstrations can be processed and enable robots to acquire new skills and perform complex tasks with precision and adaptability.

3.3.1 Dynamic Movement Primitives

The DMP [14] is developed to capture the fundamental dynamics of a motion, including the movement path of a robot's arm and the necessary forces to manipulate it. The DMP represents the movement as a second-order system, specifically a spring-mass-damper system and some force term. Equation 5 refers to this system and describes the dynamics of the DMP.

$$\begin{aligned} \tau \dot{z} &= \alpha_y(\beta_y(g - y) - z) + f(x) \\ \tau \dot{y} &= z \end{aligned} \tag{5}$$

Where τ is a time constant that controls the duration of the trajectory, which makes it possible to speed or slow the trajectory down. The DMP is built up around a second-order system represented by a spring-mass-damper system as follows.

$$\dot{z} = \alpha_y(\beta_y(g - y) - z)$$
$$\dot{y} = z$$

The values of α_y and β_y are both positive constants that are carefully selected to ensure that the system is critically-damped. A common guideline for choosing these parameters is to set β_y equal to one-fourth of α_y.

The force term in equation 5, works to approximate any given trajectory(set of force changes over time that shapes the trajectory). The force term is represented as a weighted mixture of Gaussian basis functions

$$f(x) = \frac{\sum_N^{i=1} w_i \phi_i(x)}{\sum_N^{i=1} \phi_i(x)} x(g - y_0) \tag{6}$$

Each $\phi_i(x)$ is a Gaussian function and N is the number of weighted functions. These basis functions are combined in a weighted sum to produce a smooth and continuous trajectory that can be used to guide the movement of a robotic arm or other types of system. Generating a trajectory with a higher number of basis functions N results in a more intricate trajectory, while a lower number of basis functions produces a smoother trajectory.

Looking at equation 6 the force term depends on x rather than time. x is called the phase of the system and it is defined as first-order exponential decay as seen in equation 7:

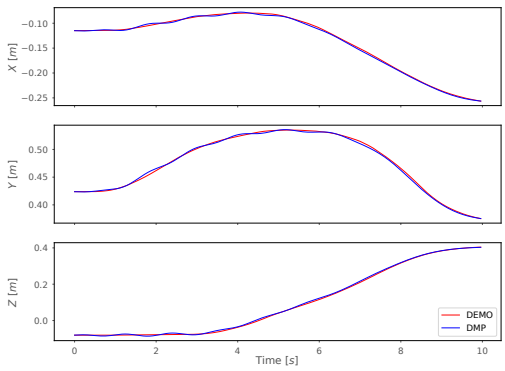
$$\tau \dot{x} = -\alpha x \tag{7}$$

The phase term in the DMP ensures that the movement converges to a position close to zero at the end of the motion. This causes the force term to become zero at the end of the

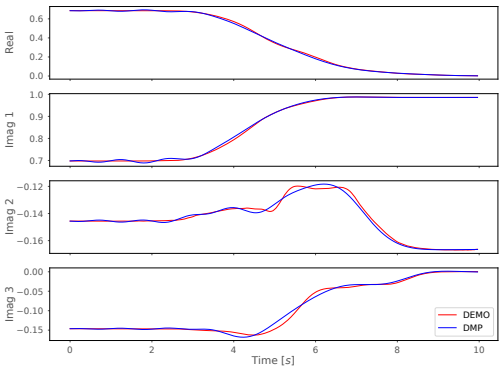
motion, leaving the system with a spring-mass-damper system that ensures convergence to a goal position regardless of external factors. Therefore, the DMP is capable of reaching a goal position at the end of a motion, and one of its key properties is that it is time-independent, making the system autonomous. The DMP is also robust to disturbance, making it an effective technique for responsive movements in dynamic environments. Finally, the DMP requires only a single demonstration to learn the desired trajectory. The DMP can not be better than the demonstration, but it can be as good as the demonstration.

However, implementing the DMP is possible for both joint space and cartesian space. There are some parameters that need to be tuned like α_y, β_y, α, τ, and number of basis function N. In this project, a smooth trajectory is deemed necessary rather than a faster or slower one. Thus, the value of τ is equivalent to the time of the demo training trajectory. To ensure that the spring-mass-damper system is critically damped, α_y is set to 48.0, and β_y is set to 12.0. The value of α in equation 7 is chosen to be $-\ln 0.01$, resulting in a phase system coverage of 99% when $t = \tau$. To achieve a smoother trajectory, the number of Gaussian basis functions N is selected to be 10.

However, the training and generated trajectory using the DMP in Cartesian space is depicted in figure 5.



(a) Position of DMP for generating extract skills.



(b) Orientation of DMP for generating extract skills.

Figure 5: *DMP versus DEMO for skill Down A.*

The aim is to achieve a smoother trajectory obtained as seen in the two figures 5a and 5b. However, if the value of N exceeds 10, the trajectory obtained is identical to the demonstration, while a value lower than 10 leads to oscillation and an imprecise path. Thus, for this particular application, N equals 10 is the most suitable option.

3.3.2 Gaussian Mixture Regression

The GMM is a probabilistic model that represents the probability distribution of a data set. This probability distribution is weighted and put into clusters of Gaussian mixture components representing the data. This can be seen in figure 6, where eight clusters are created from the demonstration. These clusters can then be used for GMR to create a path as seen in the figure, with the uncertainty.

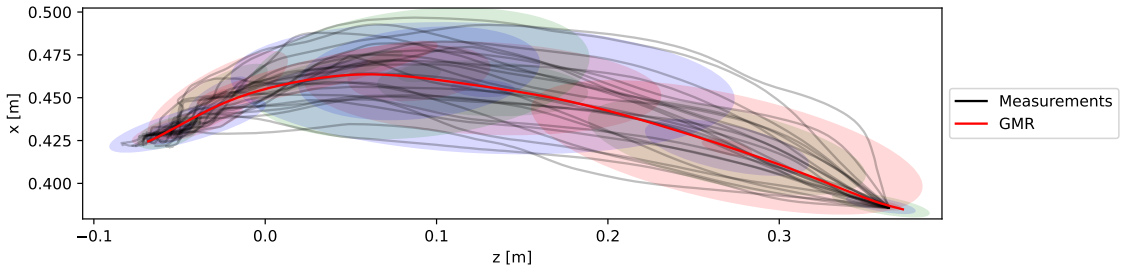


Figure 6: *GMM representation of the data set for inserting an object into object A.*

The GMM is defined by data points denoted as ξ_j, which represents a single observation in the data set[14][p.47-49]. Where the conditional probability of a data point ξ_j given a component k, is denoted as $p(\xi_j|k)$.

The mixture model gives the overall distribution of ξ_j, and the equation is seen in equation 8. The equation calculates the sum of probabilities of ξ_j belonging to the component k, weighted by the probability of component $p(k)$. This gives $p(\xi_j)$ which is the probability of observing ξ_j in the data set.

$$p(\xi_j) = \sum_{k=1}^{K} p(k)\ p(\xi_j|k) \tag{8}$$

In the Gaussian Model, the $p(\xi_j|k)$ is calculated using the Gaussian probability density function (PDF).

In equation 9, the component k's relative contribution to the overall mixture is calculated. The probability of component $p(k)$ in the mixture model is denoted as π_k, while μ_k and Σ_k denote the mean vector and covariance matrix for the component k.

$$\begin{aligned} p(\xi_j|k) &= \mathcal{N}(\xi_j; \mu_k, \Sigma_k) \\ &= \frac{1}{\sqrt{(2\pi)^D |\Sigma_k|}} e^{-\frac{1}{2}\left((\xi_j-\mu_k)^T \Sigma_k^{-1} (\xi_j-\mu_k)\right)} \end{aligned} \tag{9}$$

The Gaussian model assumes the data within each component to be normally distributed. Furthermore, it uses the Mahalanobis distance to measure the dissimilarity between the data point and the mean of the component in the feature space. This allows the GMM to effectively model the distribution of the entire data set, and capture complex data structures, and identify underlying patterns in the data. GMM is suitable for complex tasks as demonstrated in this project, where a skill is learned by demonstrations. The GMM estimates a path from the demonstrations, which can simplify the task of reducing the noise or the requirement for one perfect demonstration. This is because the path can be smoothened out to be within the variances which are visualized in figure 6. This highlights one of the benefits of using a probabilistic model for LfD.

The estimation of GMM is done through the following 3 steps:

1. Decide on the number of mixture components
2. K-means clustering
3. Expectation Maximization (EM) Algorithm
    - Expectation (E) Step
    - Maximization (M) Step

The GMM output is used for the GMR, which is a regression technique that adds regression analysis to the GMM. It estimates the output variables with covariance. GMR combines the flexibility of GMM in capturing complex data distributions, with the regression analysis making GMR a flexible and robust approach to regression problems. Regardless if the data is heterogeneous or multimodality.

To find the expected spatial value and covariance, GMM joint probability distribution is calculated by equation 10.

$$\mathcal{P}(\xi_t, \xi_s) : \begin{bmatrix} \xi_t \\ \xi_s \end{bmatrix} \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k) \tag{10}$$

Where μ_k and Σ_k are defined by separate temporal t, and spatial s components:

$$\mu_k = \{\mu_{t,k}, \mu_{s,k}\} \tag{11}$$

$$\Sigma_k = \begin{pmatrix} \Sigma_{t,k} & \Sigma_{ts,k} \\ \Sigma_{st,k} & \Sigma_{s,k} \end{pmatrix} \tag{12}$$

This gives the output for the GMR[14][p.52], which is the expected spatial value and covariance, as seen in equation 13 and 14.

$$\hat{\xi}_s = \sum_{k=1}^{K} \beta_k \hat{\xi}_s, k \tag{13}$$

$$\hat{\Sigma}_s = \sum_{k=1}^{K} \beta_k^2 \hat{\Sigma}_s, k \tag{14}$$

Where β_k is the probability component k that is responsible for the time, ξ_t. The conditional spatial expectation for a given time based on the mean and covariance is denoted by $\hat{\xi}_{s,k}$. It uses the deviation of ξ_t from its mean $\mu_{t,k}$ and utilizes the covariance relationship between ξ for s, k and t, k. $\hat{\Sigma}_{s,k}$ is the adjusted covariance for a given time. It uses the original covariance matrix $\hat{\Sigma}_{s,k}$.

$$\beta_k = \frac{\mathcal{P}(\xi_t|k)}{\sum_{i=1}^{K} \mathcal{P}(\xi_t|i)} \tag{15}$$

$$\hat{\xi}_{s,k} = \mu_{s,k} + \Sigma_{st,k}(\Sigma_{t,k})^{-1}(\xi_t - \mu_{t,k}) \tag{16}$$

$$\hat{\Sigma}_{s,k} = \Sigma_{s,k} - \Sigma_{st,k}(\Sigma_{t,k})^{-1}\Sigma_{ts,k} \tag{17}$$

In figure 7, the measurements for the orientation and translation in cartesian space, as well as the path found using this method. This figure shows the uncertainty for each DoF

during the LfD. It can be noticed that the tolerances must be higher for some DoF. Looking at the translational z-axis, the robot starts in a high position and ends low next to the table, with little variation on both ends. Whereas the y orientation starts with a specific orientation, but has a high variance during the insertion section at the end. These graphs show which axis must be precise, and which can accept some variation. This information is later used for blending where it reduces the error to be blended, and subsequently provides a smother blend.

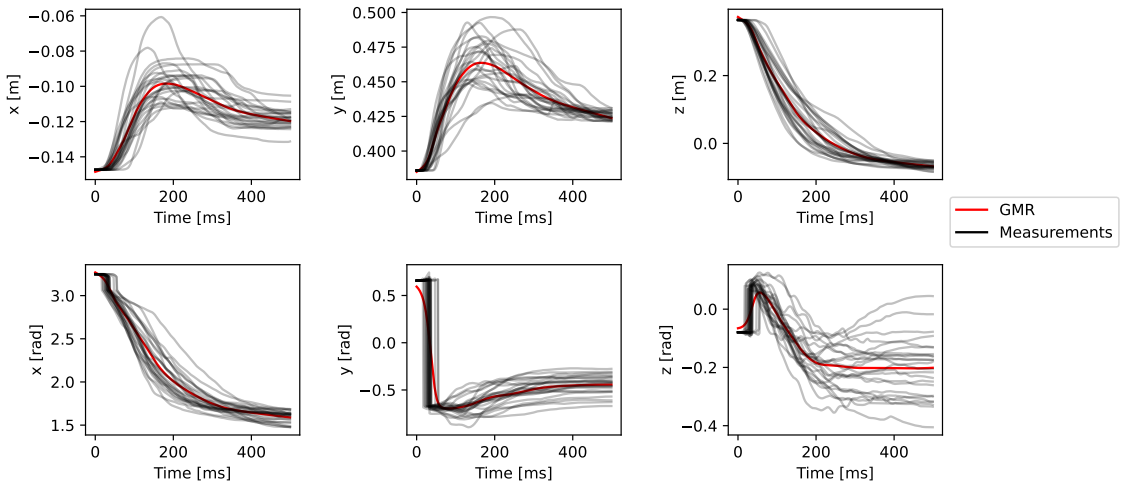


Figure 7: *Visualization for orientation and translation*

A 3D visualization of the translational path in figure 7 is seen in figure 8. This shows an ellipsis for each point on the GMR path. These ellipses show the tolerances at those points. These cone-shaped tolerances around the GMR path, show the path which must be followed. Visualizing some of the benefits of using a probabilistic model to model the demonstrations.

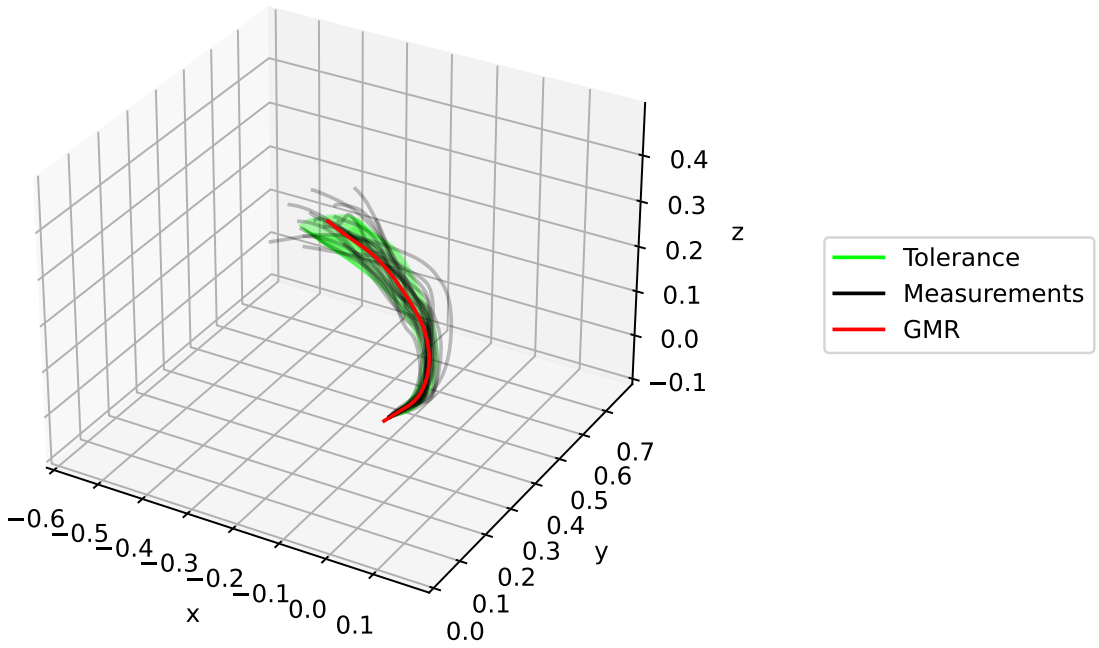


Figure 8: *3D visualization with uncertainty*

3.4 Blend of several skills

The blend operation serves the purpose of smoothing out the movement trajectory, eliminating abrupt changes, and improving the overall path optimization. This smoothness not only enhances the top speed and acceleration but also allows for longer paths.

3.5 Trajectory blending with Parabolic blend

The method used for blending the skills is a parabolic blend. The main method used is from Github [3] with changes to utilize it for this project. This method works for one-dimensional trajectories and smoothens the trajectory between two or more points, making the trajectory continuous.

To implement the method, the approach has been to import two trajectories from which three or more points are extracted for the blend. If two trajectories meet at the same configuration, three points are used but if two trajectories do not meet at the same configuration, an extra point is added to create the trajectory between the last configuration in the first trajectory and the first configuration in the second trajectory.

Between all points, a linear trajectory is created, and for each point, a parabolic blend is created between the points. With this method, achieving blends without two trajectories meeting in the same configuration is possible and reduces the number of trajectory creations required.

For the linear segments, the speed is constant and calculated as

$$v_i = \frac{q_{i+1} - q_i}{\Delta T_i} \tag{18}$$

where q_{i+1} and q_i are waypoints, and ΔT_i is the time between the points.

In the blend phases at the waypoints, the linear segments are replaced with a parabola that follows a constant acceleration.

$$a_i = \frac{v_i - v_{i-1}}{t_i^b} \tag{19}$$

where v_i and v_{i-1} are velocities at each waypoint and t_i^b is the duration of the blend phase at waypoint i.

The parabolic blend is given by

$$b(t) = b_0 + \dot{b}_0 t + \frac{1}{2} a t^2 \tag{20}$$

where b_0 is the position for the start of the blend, $\dot{b}_0$ is the velocity at the start of the blend and a is the constant acceleration during the blend. For a blend around a waypoint i, these are given as

$$b_0 = q_i - v_{i-1} \frac{t_i^b}{2} \tag{21}$$

$$\dot{b}_0 = v_{i-1} \tag{22}$$

$$a = a_i \tag{23}$$

The one-dimensional trajectory in figure 9 displays both the original path with waypoints using linear segments between each point and the blended path showing where the blending starts and ends for each point using a parabolic blend [4].

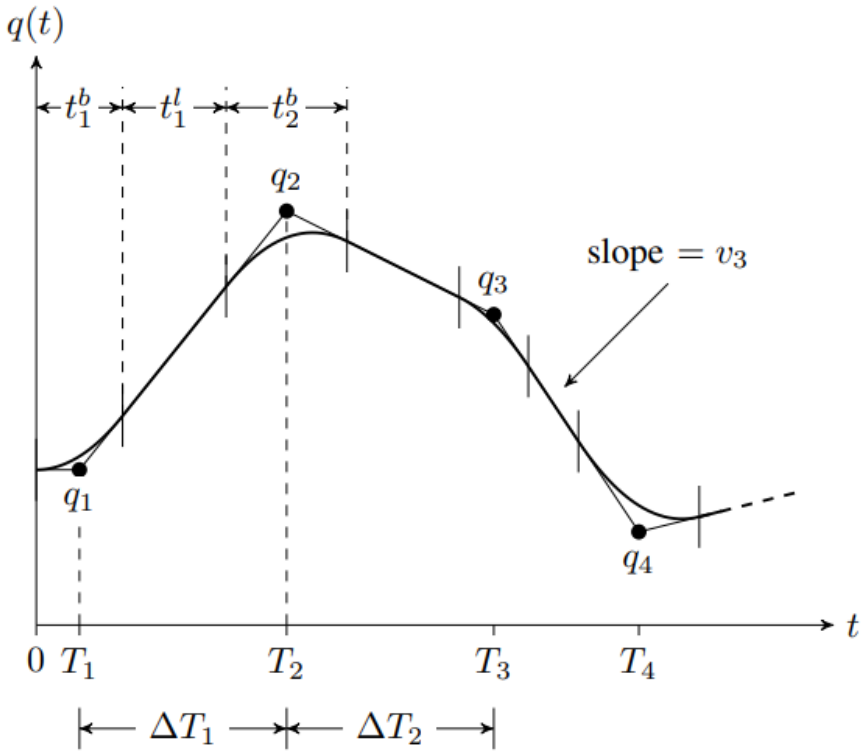
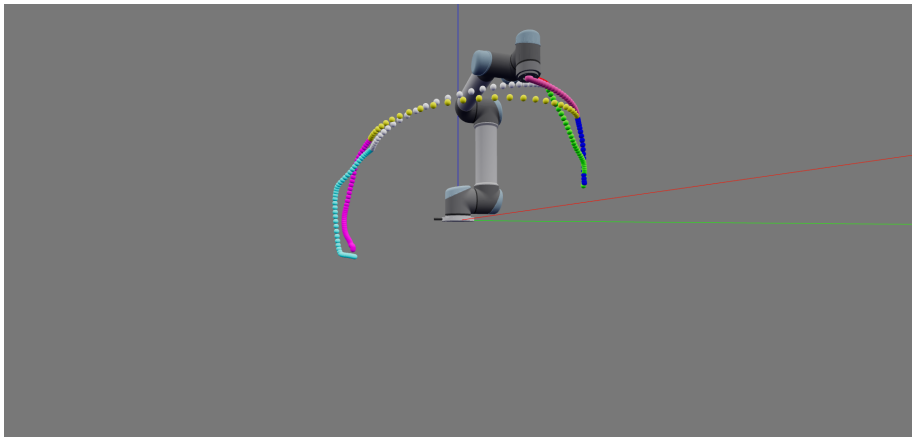


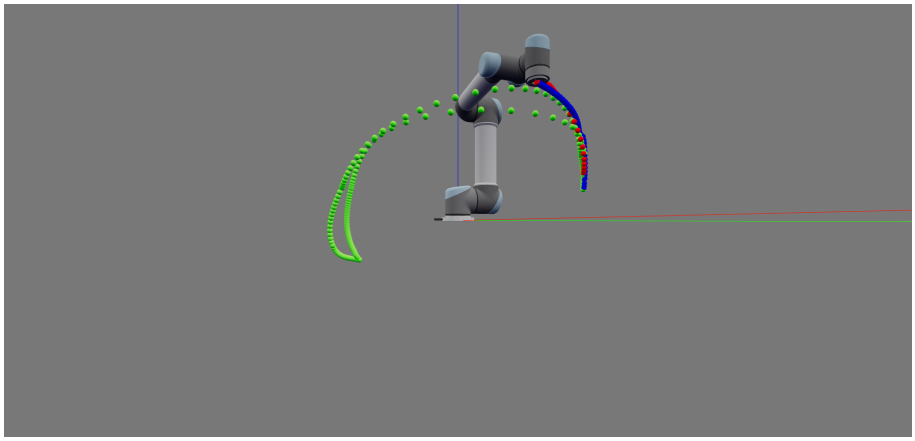
Figure 9: *One-dimensional Linear Segment with Parabolic Blends for 4 of 5 points*

Since the method provides a generic method for one-dimensional path smoothing, the robot poses used are joint angles for each joint.

The parabolic blend method is used in two ways in the project. The paths taken are divided into segments at each step, where it reaches object A or object B. In figure 10 two images are shown. Both display the trajectories as dots of the robot with a different color for each segment used. Note that for figure 10b the number of segments has decreased and the paths are smoothened with blends compared to figure 10a.



(a) *Complete path without optimization*



(b) *Complete path with optimization*

Figure 10: *Differences between the optimized path and the original path*

Figure 11 shows a blend between two paths that a not connected at a single point. For each axis for the translation and the rotation in axis-angles, it shows the position and the waypoints that it has to follow. Due to changes in the length of the trajectories, the standard length in which the method selects the start and end blend position is set to 20 steps.

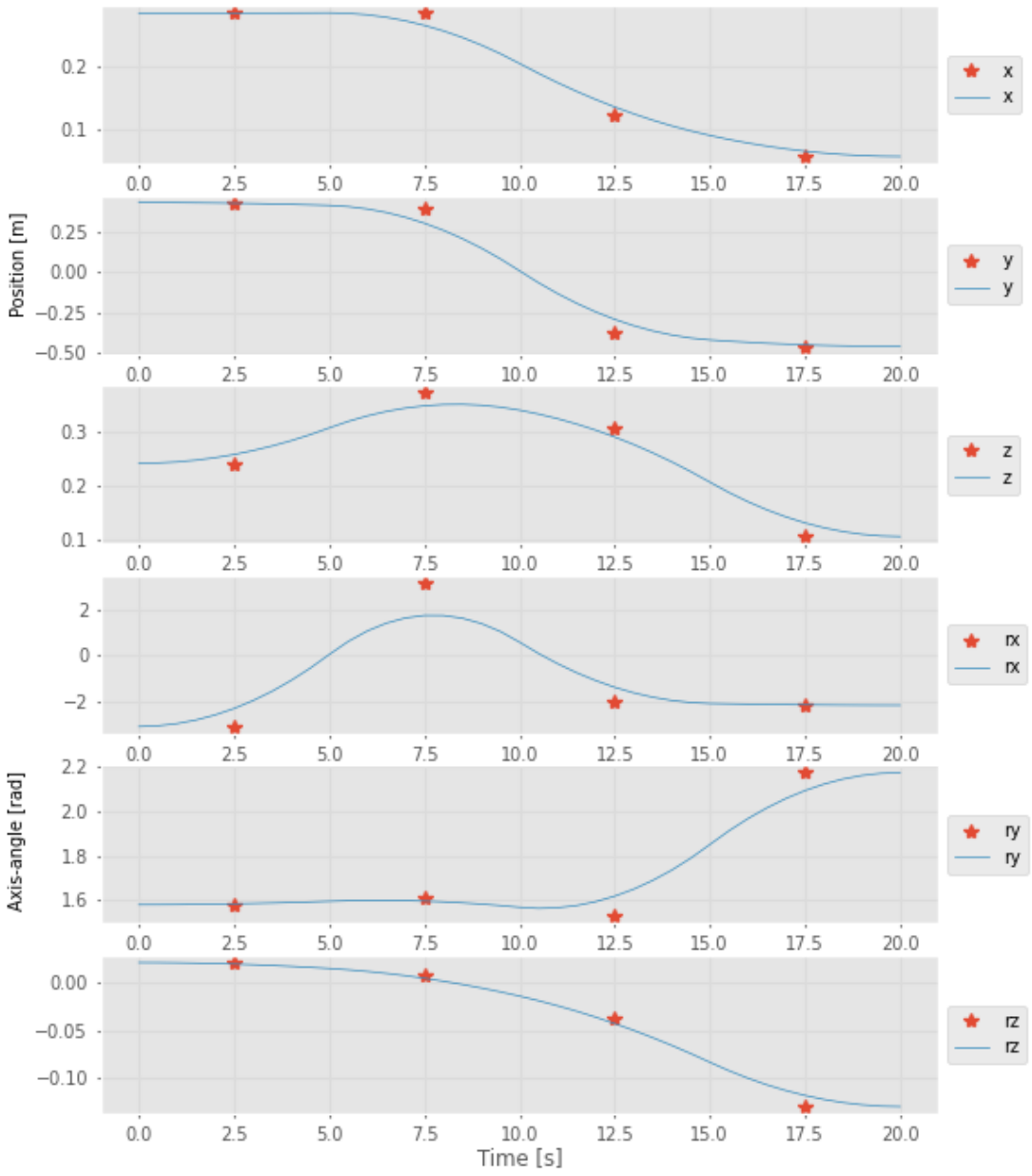


Figure 11: *Blend from point B to Point A - the stars are the waypoints and the lines is the parabolic blend*

3.5.1 Skill blending using GMR tolerances

This section describes the optimization used by GMR to smoothen the transition from one skill to another when they are closely fitted together. From the LfD there are some small offsets in some of the joints that are unwanted, and with the covariances from the GMR, it is possible to reduce this high acceleration movement between skills.

With the GMR, described in section 3.3.2, the covariances are used to reduce the distance between the current position and the next, if the distance between the points is larger than the covariance. If the distance is too large, the current configuration is adjusted by the maximum covariance distance allowed. An example can be seen in figure 13 and figure 12. Both figures display three columns, the left column is the path of the trajectory that is modified. It shows both the trajectory before and after applying the covariance optimization. In the middle column, the original two trajectories are plotted, and the gap between them is where one trajectory ends and the other begins. In the right column, both trajectories are plotted with the optimized points where it is obvious to see that the transition between the skills is blended to reduce the large acceleration.

For the difference in the two trajectories in the left column, the reason is that during the normal iteration of the trajectory, the points all lie within the covariance, and here to algorithm reduces the distance for all points afterward. Therefore they receive a slight displacement. This is an effect of the optimization method that corrects the rest of the trajectory.

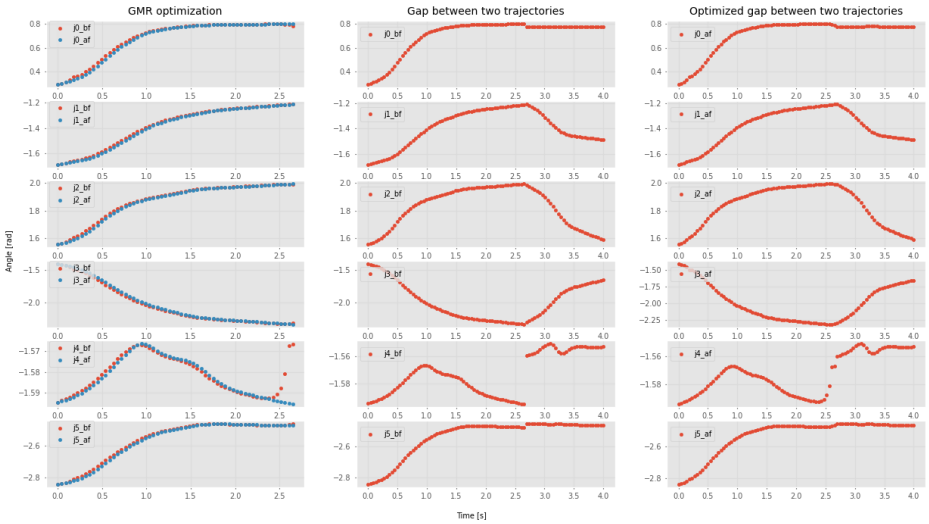


Figure 12: *GMR gap optimization between the insert and retract for picking at object B. j0_bf is the path before optimization and j0_af is the path after optimization.*

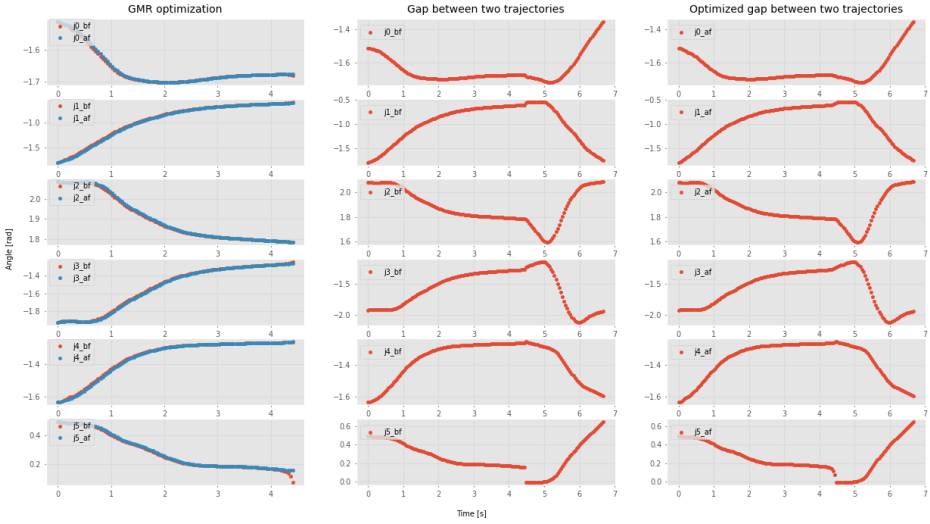


Figure 13: *GMR gap optimization between insert and retract for picking at object A. j0_bf is the path before optimization and j0_af is the path after optimization.*

3.6 Skill blending using quadratic programming optimization

For quadratic programming (QP), the implementation is created using the `Python` library `cvxpy`. The method takes in two configurations of the robot pose, which can be either the joint values or the translation and axis-angles. Both of these are a vector of six values, which then can be put into the `np.linalg.norm()` function. This function computes the norm (magnitude or length) of a vector. By subtracting configuration 1 from configuration 2, the vector representing the displacement between the two points can be obtained. Taking the norm of this displacement vector gives the Euclidean distance between the two points in the same space.

The distance between the points determines the number of waypoints that the QP will generate. The Euclidian distance is a singular value, and a predetermined value of 0.01 is chosen as the division factor to calculate the required number of waypoints. Note that here the waypoints are the positions that the robot reaches at each step in between the two configurations.

The QP is a mathematical optimization problem that aims to minimize a quadratic objective function that is constrained to be affine. The general form is expressed as:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}x^T P x + q^T x + r \\ \text{subject to} \quad & Gx \preceq h \\ & Ax = b \end{aligned} \tag{24}$$

where x is the optimization variable as a vector. The objective function consists of three terms: the quadratic term: $\frac{1}{2}x^T P x$, the linear term $q^T x$ and the constant term r. The matrix $P \in S_+^n$ determines the quadratic relationship between variables, and q and r are vectors that define the linear and constant components, respectively.

The constraints are expressed in the form of linear or affine inequalities $Gx \preceq h$, where $G \in \boldsymbol{R}^{m \times n}$ is a matrix and h is a vector. These inequalities restrict the feasible solutions. Additionally, there can be equality constraints of the form $Ax = b$, where $A \in \boldsymbol{R}^{p \times n}$. m is the number of inequality constraints, p is the number of equality constraints, and n is the number of decision variables, which is the six joint values [5].

In this case, the QP uses upper and lower bounds as the joint limits of the UR5e, with no inequality constraints and equality constraints as the start and end joint configuration.

In figures 14 and 15 the QP combined with the two trajectories are shown. In this method, the original trajectory does not change as seen in the left column of both figures. Here the red dots are barely visible since all the blue dots cover these. The entire path is then unchanged and will be kept as it is created by the trajectory control methods, such as DMP or GMR. The QP steps are added in between the two trajectories, making the entire duration of the movement longer by the number of waypoints added.

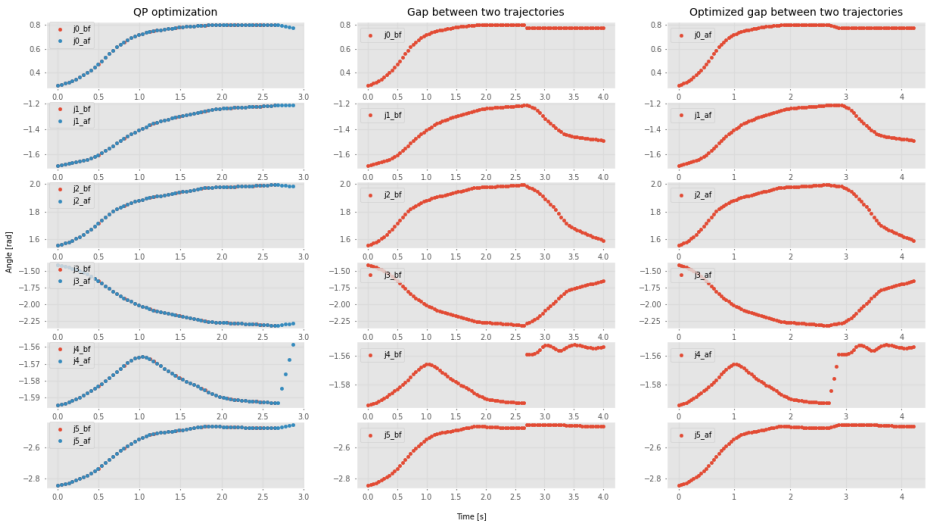


Figure 14: *QP gap optimization between the pick and place at object B*

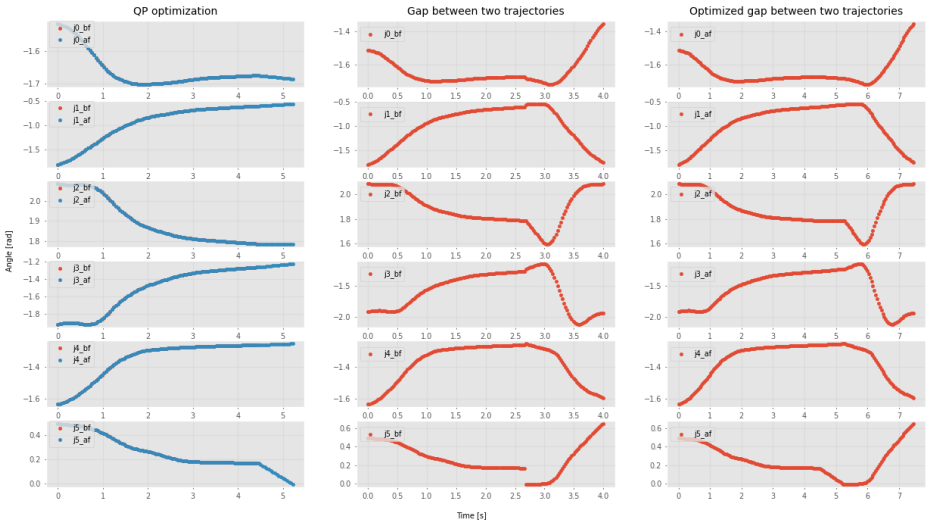
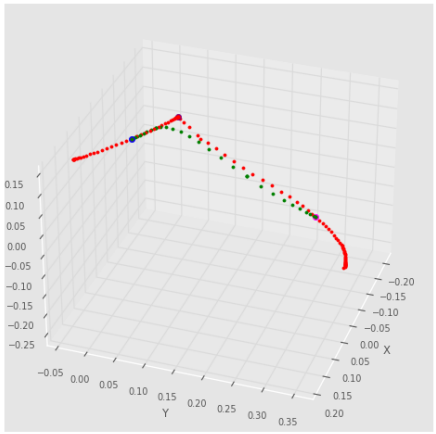


Figure 15: *QP gap optimization between insert and retract at object A*

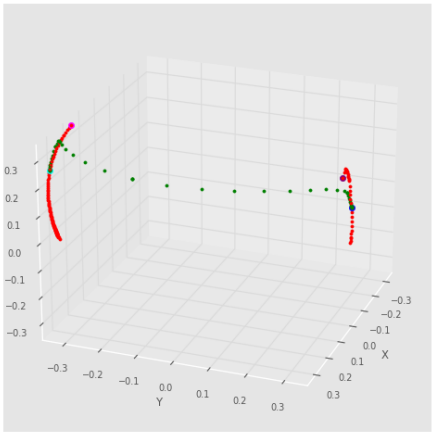
4 Experiments

The trajectory blends versus the original trajectory are set up on top of each other to compare how the trajectory looks, and how the trajectory changes can be seen in figure 16. The red dots illustrate each step in the original trajectory. The green dots show the blended trajectory, combining the trajectories in a smooth way. The other colored dots display the starting point for the blend in trajectory one, the middle point or two middle points when the paths are apart, and the point where the blend ends in the second trajectory, and it then follows this trajectory.

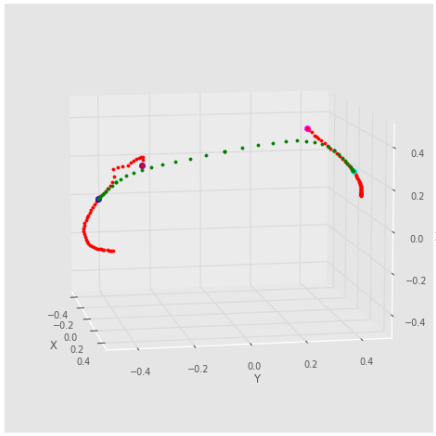
The figure illustrates how the blended trajectory optimizes the path by eliminating abrupt changes and ensuring continuity and a smooth transition between different segments.



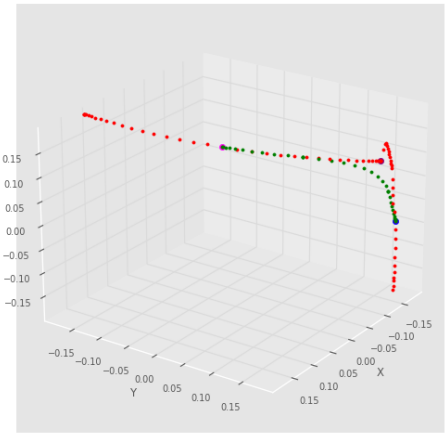
(a) From home to point B



(b) Point B to point A



(c) Point A to point B



(d) From point B to home

Figure 16: *Paths with blends*

To conduct the physical implementation, the experiment must be done using the physical UR5e, `ur_rtde` interface is used for controlling and receiving data from a UR robot using the Real-Time Data Exchange (RTDE) interface of the robot[6].

Recording the data for LfD using the admittance controller requires an external force applied at the end effector, which the `ur_rtde` offer as a function **getActualTCPForce()** and since the admittance control result in position, velocity, and acceleration in cartesian space, **SpeedL** is used to move the robot according to the external force applied at the

end effector expressed in the base frame. **SpeedL** is used to allow for smooth, continuous movement of the robot's end-effector in cartesian space.

However, for the DMP, GMR, and the blend method it has been decided to work in a joint space rather than cartesian space. There are two methods that can be used to communicate with the robot. Figure 17 shows the two different methods that are used in joint space **ServoJ** and **MoveJ**.

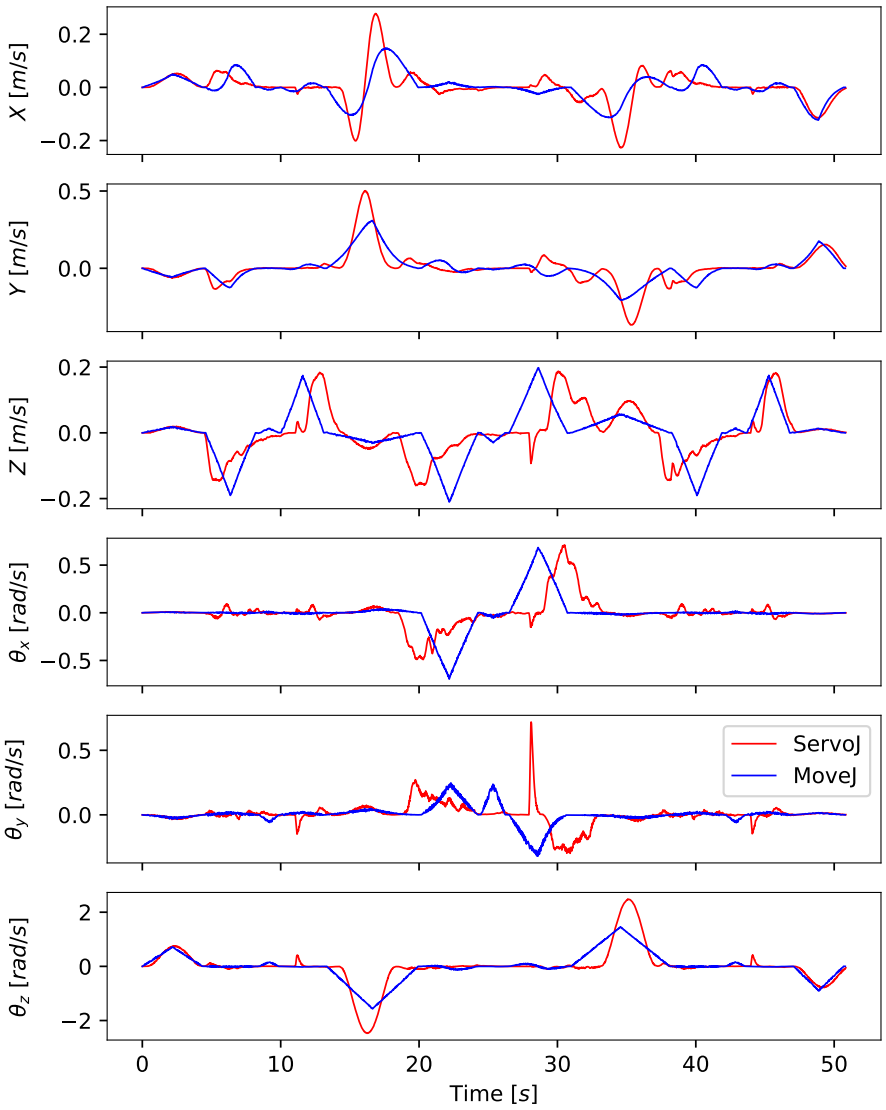


Figure 17: *ServoJ verse MoveJ velocity profile in joint space.*

Looking at the figure, it is obvious that **ServoJ** creates a smoother path, and the linearity in the velocity profile is not present which makes the path smoother, therefore **ServoJ** is going to be used for future experiments.

In order to demonstrate the impact of the trajectory blend, it is necessary to carry out an experiment on a path using both the trajectory blend method and just combing the trajectories without using any optimization method. It is evident from figure 18 that the blend method is quicker.

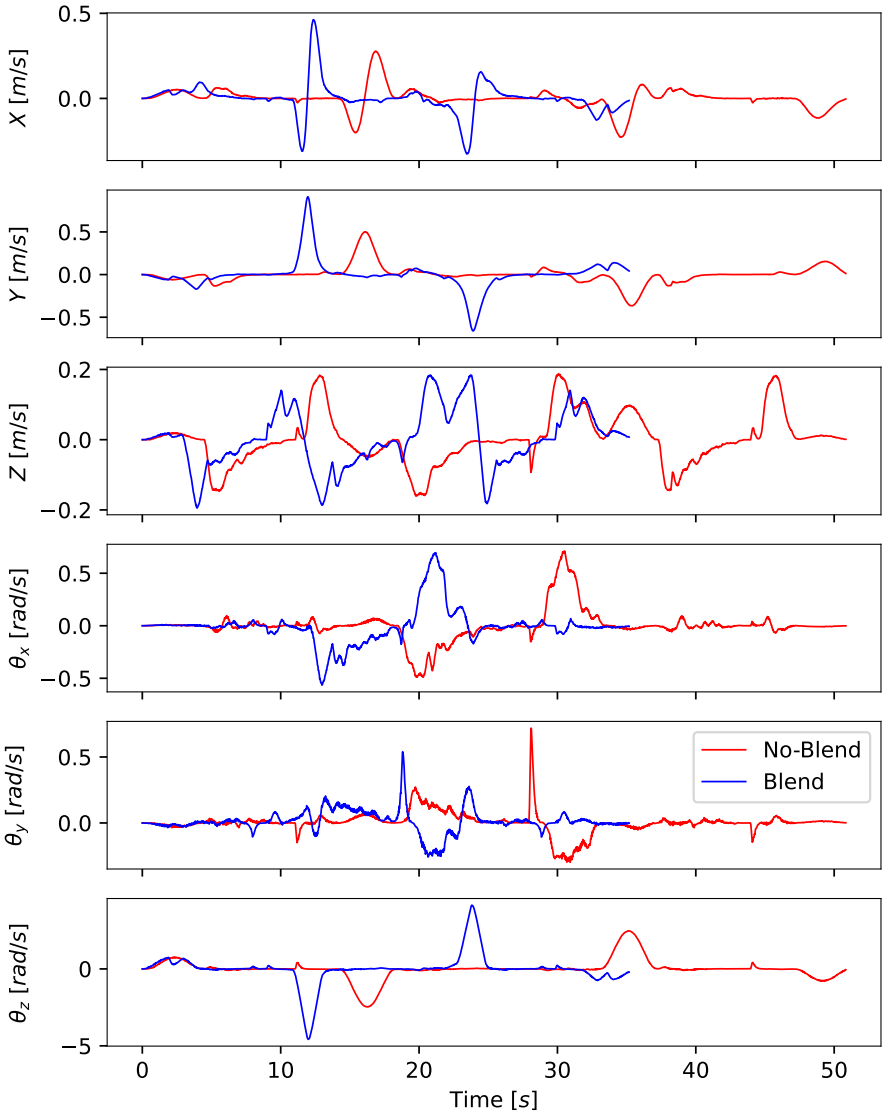


Figure 18: *The velocity profile for a path when employing the blend method versus when not utilizing a blended method.*

The superiority of the trajectory blend method is apparent, hence it will be employed for all remaining experiments.

To evaluate the two LfD methods(GMR and DMP) with a blend between two trajectories, the two LfD methods are blended separately and then merged the two LfD methods together with the different skills. On top of that quadratic programming optimization method is used to blend between the skills.

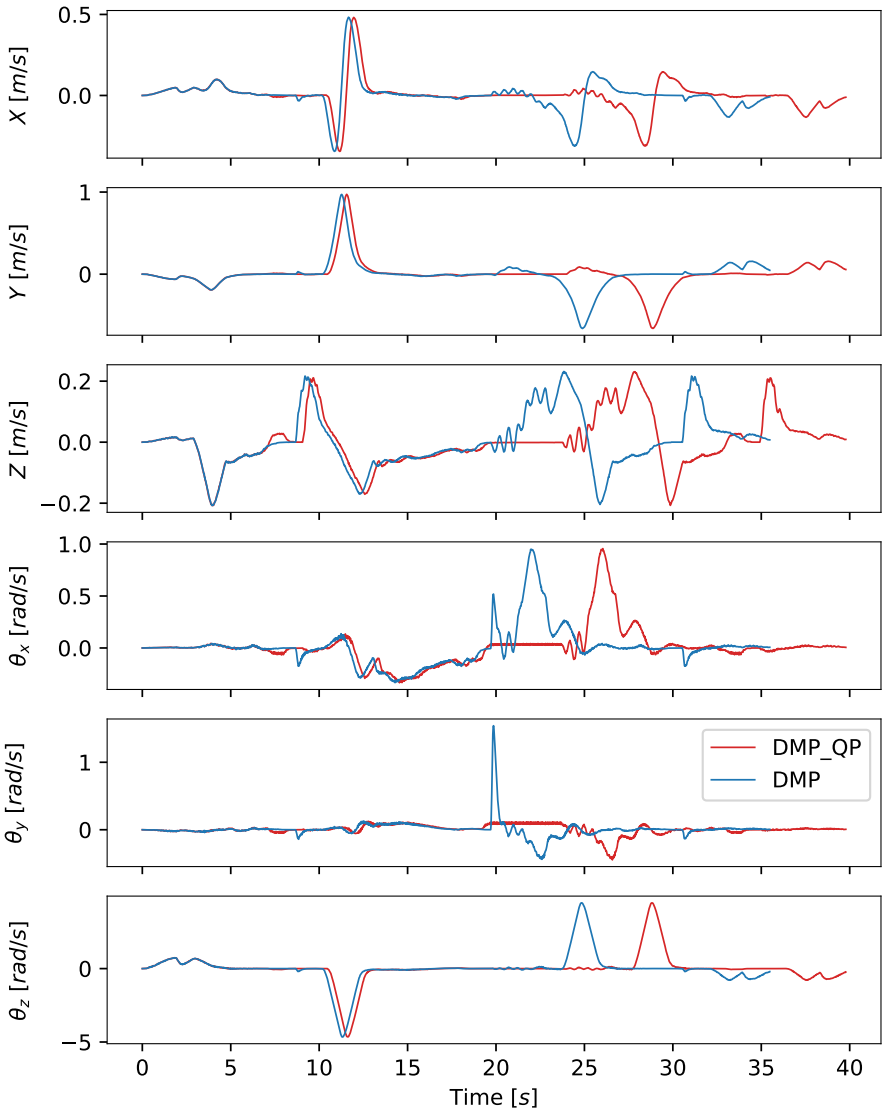


Figure 19: *Velocity profile of blending GMR using QP between the skills and without applying the QP.*

The skill blending method is used to evaluate the two methods in cartesian space as it is more convenient to compare and understand than joint space. The outcome of individually blend methods on the DMP and the GMR is depicted in Figure 19 and Figure 20.

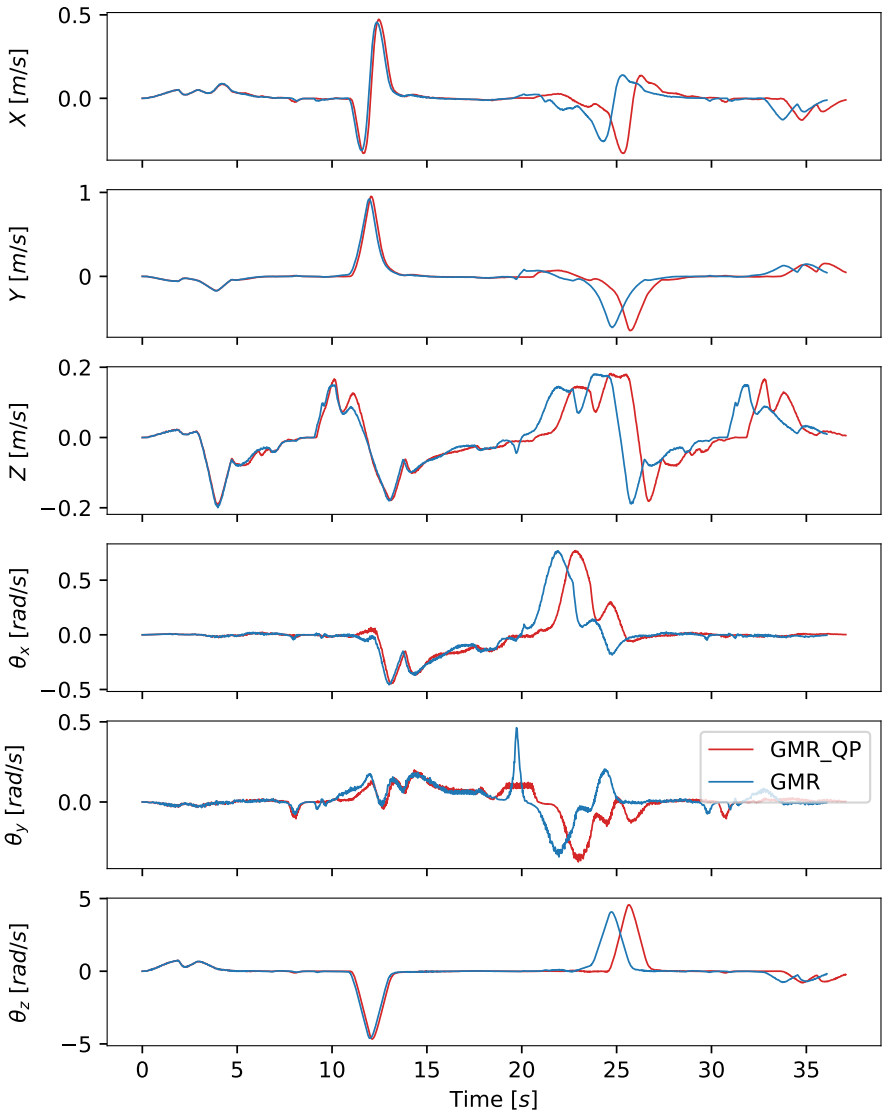


Figure 20: *Velocity profile of blending GMR using QP between the skills and without applying the QP.*

As seen in the above figure 19, the QP optimization method removes the high spike in velocity profile from the change between skills (pick-place, and insert-extract). On the other hand, the QP method expands the trajectory by inserting additional intermediate steps specifically between the skills that are intended to reach the same point which makes

the trajectory longer. The QP adds 10 points between the pick and place skills (object B) and it adds 46 points between the insert and extract skills (object A).

In the case of GMR, as seen in figure 20, the QP inserts 15 and 24 extra steps between the corresponding skills, pick-place and insert-extract respectfully. On the other hand, the QP method seems not to have a big impact on GMR compared with DMP due to the GMR is good to handle deviations between the skills.

GMR appears to be smoother and more robust compared to DMP without the use of QP. This is attributed to the fact that GMR uses multiple demonstrations instead of a single one, resulting in a model that is better to handle noise in the input data. Worth pointing out that both methods are trajectory blended seamlessly.

It has been decided in this project to merge the two skills together, using DMP with skill A and GMR with skill B and vice versa. The following figure 21 shows the velocity profile of merging the skills together.

Clearly, GMR is more adept at handling the complexities of the B skill and is also faster than DMP. On the other hand, it appears that DMP is superior in handling the A skill and has a quicker processing time.

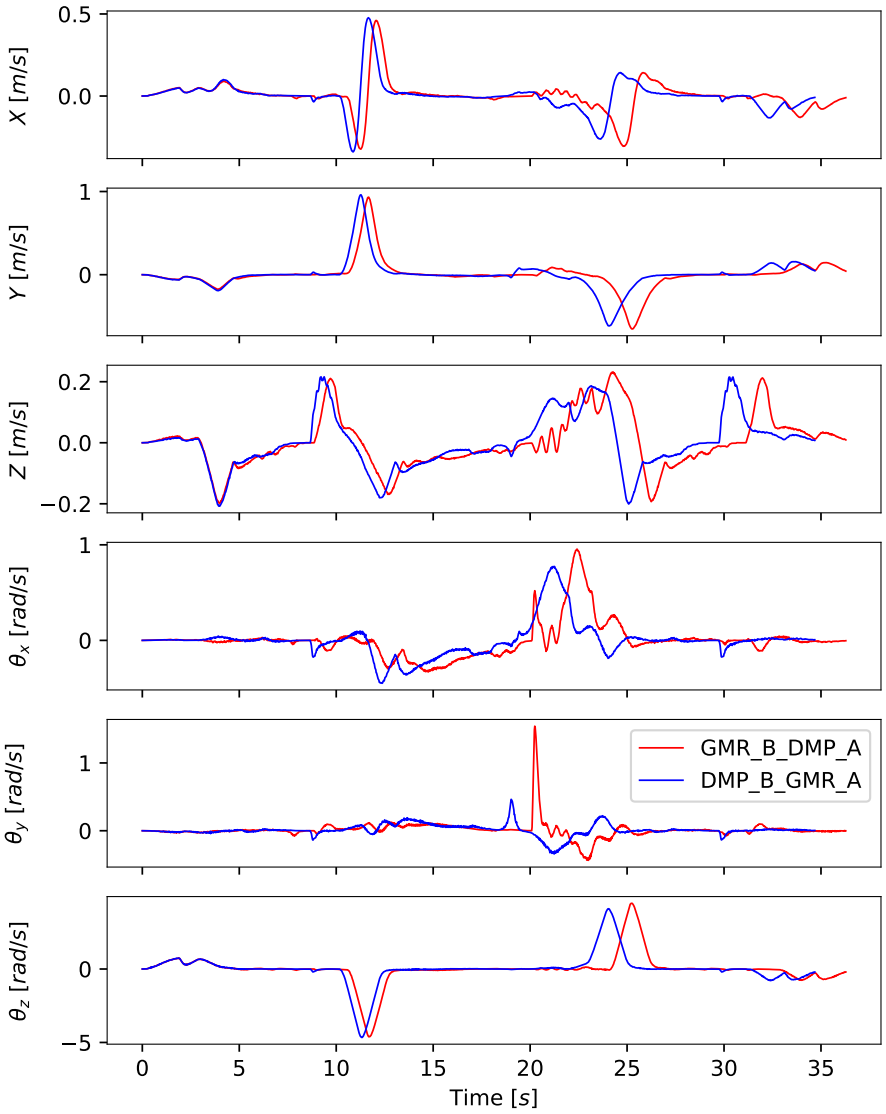


Figure 21: *Velocity profile of blending results from the combination of DMP output with both skills*

.

Quadratic programming (QP) is compared to the GMR covariance distance reduction between the two configurations where it has to meet, which can be seen in figure 22 and figure 23 that are extracted from the individual methods in section 3.5.1 and section 3.6.

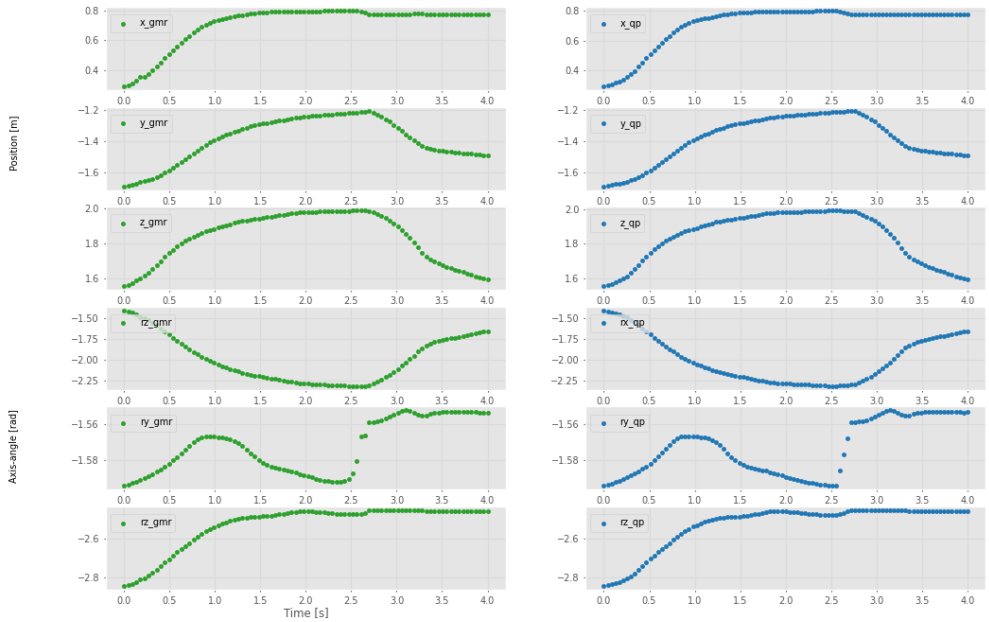


Figure 22: *From home to point B*

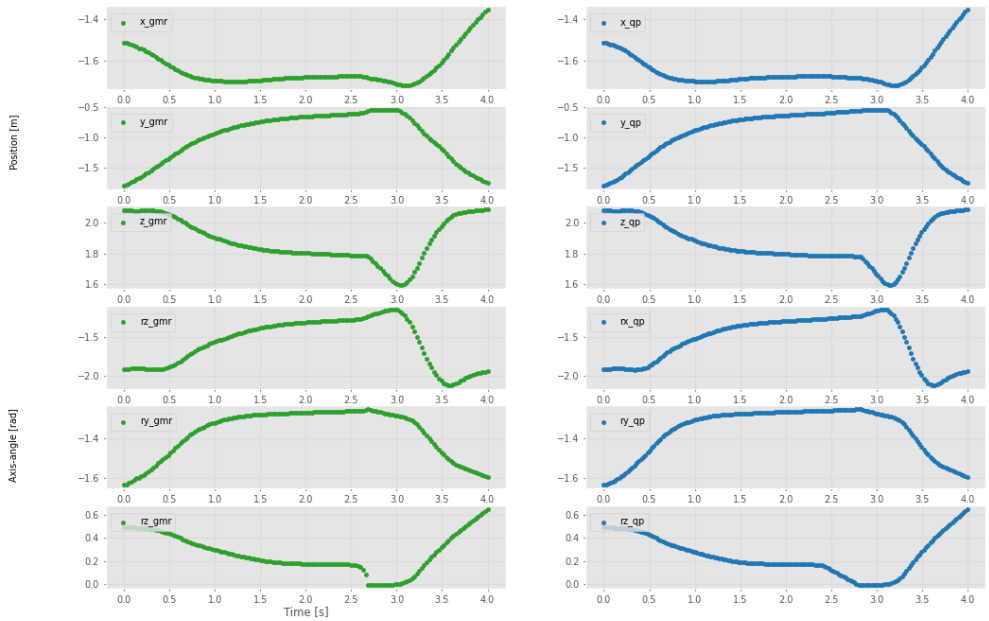


Figure 23: *Point B to point A*

5 Discussion

While testing in simulation, there are issues with inverse kinematics, where the method provides poor robot poses, which increases the trajectory between two points because the difference between joint poses is much larger. Therefore, the robot pose is extracted as joint angles and joint velocities, and the TCP pose is calculated using forward kinematics. As seen in figure 25 in the appendix section it can be seen that in rotation of axis x (rx), there is a discontinuity when the value reaches π and turns into a $-\pi$. This is a limitation caused by the `UR` implementation, which returns the TCP in axis-angle instead of quaternions.

The admittance control uses the 6 DoF force-torque sensor to measure the force applied to the end-effector. This sensor had a lot of noise, which caused unexpected behavior in the admittance control. The applied filter reduces noises when zero force is applied to the end-effector allowing for a smoother tuning of the admittance control parameters. This part is important since the quality of the learned model will be less than or equal to the quality of the recorded data with the data processing algorithms.

The demonstrations are recorded using the admittance control during kinesthetic teaching. This allowed for a more natural movement of the robot, which is important for the learned model to be able to generalize to new situations. However, demonstrating skill with kinesthetic teaching poses the risk of adding noise to the demonstrations, which can cause the learned model to be less accurate. This causes the need for a precise controller, and the implemented impedance controller on the UR5e lacked the ability for precise control with low forces. The impedance control is therefore not suitable for LfD since it is designed stiff to cooperate with anything, rather than being smooth for LfD.

The DMP proves to work well with dynamic environments, as it adapted to the changes in the environment. Furthermore, a single demonstration is enough for the DMP to learn the trajectory, which is a great advantage compared to the GMR, which requires multiple demonstrations to learn the trajectory. DMP, however, depends on having a good demonstration, as it cannot correct the trajectory for errors in the recorded data.

The GMR is capable to generate a trajectory that is able to follow the path, but it is not ideal for a dynamic environment like the DMP. However, the GMR produces a smoother

trajectory compared to the DMP, which is a great advantage when the environment is static. GMR is also benefitting from the probabilistic approach, as it is able to generate a trajectory based on multiple demonstrations. This simplifies the process of generating a trajectory since a demonstration can accept some variance. This benefit is at the same time a disadvantage, as it requires multiple demonstrations to generate a trajectory.

In the project, the method used to follow the trajectory is **ServoJ** which worked well for the different paths and for testing but to make it smooth and keep the same approximate speed for the entire trajectory, it had to be adjusted in the spacing between each point for the sections of the trajectory which had a large number of steps. Another approach is to use the method **SpeedL**. If this is used, the trajectories generated by GMR, DMP, and blend will not be restricted to the spacing between them, as it continuously adjusts the speed based on the configuration read in the trajectory. This will also allow us to define and change the velocity and acceleration profile further compared to using the **ServoJ**, which has the same properties for all movements between configurations.

Additionally, in the project, different lengths of paths are used due to the complexity of the tasks and the recordings from the admittance control. During testing, a dynamic blend size selection is tested but resulted in some blends that started too early. Therefore, blending with a size of 20 for all trajectories is chosen for this project.

The GMR edits the original path where the QP is appended to the end and increases the length. And while the GMR only tries to move the existing points closer to the next point in the second trajectory, the QP creates a number of points in between. The number of points is based on the total distance and every axis needs to have the same number of points in between the two configurations. Therefore some of the axes have unnecessary points where it might not be needed, e.g. y-axis.

On the contrary, in some of the transitions between the two configurations, there is a rapid movement due to the change in position and how the function **ServoJ** moves, which will be reduced with the increased number of transition steps. In testing with both GMR and DMP, the GMR is better at reducing the high velocities experienced between configurations in between the skills. QP is applicable for any type of trajectory control

that doesn't have another type already and is able to reduce speed and acceleration even further than GMR, since it is scalable in the number of intermediate steps.

As the QP is a late implementation in the project, this value for calculating the appropriate number of steps between the configurations has not been verified as the best solution to the length of the QP between configurations.

6 Conclusion

The creation of skills is created with LfD using an admittance control. DMP and GMR are used to process these demonstrations and define skills.

The properties of DMP offer several benefits. Its ability to converge to a goal position makes it suitable for dynamic environment applications, and it is also noteworthy for its time independency. A single demonstration is enough for the DMP but the demonstration must be good to guarantee a smooth path.

GMR is probabilistic and utilizes multiple recordings of demonstrations and provides a path with tolerances. This proved to be smother than DMP and handles the transition from the previous point before the skill and the point after the skill. GMR demonstrated great performance and is effective for human-taught skills, which poses some noise in the demonstrations.

The trajectory blending of the skills reduces the total time for the entire trajectory and provides smooth transitions between trajectories. The results have demonstrated the effectiveness of the parabolic blend technique in achieving smoother trajectories even when the two trajectories are not connected at a single point, which is very useful when using LfD, where chances are the robot is not in the exact same position every time a skill is recorded. The blended trajectories exhibit improves smoothness and reduce acceleration, optimizing the robot's overall movement.

The GMR covariances prove beneficial in reducing the distance between skills in close proximity areas where trajectories can be merged instead of blended. The QP is useful both for the GMR trajectories with and without the GMR covariances, and also for the DMP to create the shortest distance trajectory between skills. Both of these reduce the distance for all joints to their values at the start of the next skill, which also reduces the velocity and acceleration at these points.

References

[1] N. Jaquier, Y. Zhou, J.Starke, et al., *Learning to sequence and blend robot skills via differentiable optimization* in IEEE, 2022

[2] M. Saveriano, F. Franzel, and D. Lee, *Mergning position and orientation motion primitives* in IEEE, 2019

[3] cuezeebee: Trajectory planning. URL: https://github.com/novice1011/trajectory-planning

[4] T Kunz, M. Stilman, *Turning Paths Into Trajectories Using Parabolic Blends* in GT Digital Repository, 2011

[5] Stephen Boyd and Lieven Vandenberghe: Convex Optimization - page 152-. URL: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

[6] interface for controlling and receiving data from an UR robot using the Real-Time Data Exchange (RTDE) interface of the robot URL: https://gitlab.com/sdurobotics/ur_rtde

[7] Recording of the free drive(impedance controller): https://github.com/Kamik18/Project-in-Advanced-Robotics/blob/main/Video/Freedrive.mp4

[8] Recording of the admittance control: https://github.com/Kamik18/Project-in-Advanced-Robotics/blob/main/Video/Admittance.mp4

[9] Recording of skill B Down: https://github.com/Kamik18/Project-in-Advanced-Robotics/blob/main/Video/B_down.mp4

[10] Recording of skill A Down: https://github.com/Kamik18/Project-in-Advanced-Robotics/blob/main/Video/A_down.mp4

[11] Recording of skill A UP: https://github.com/Kamik18/Project-in-Advanced-Robotics/blob/main/Video/A_up.mp4

[12] Recording of skill B UP: https://github.com/Kamik18/Project-in-Advanced-Robotics/blob/main/Video/B_up.mp4

[13] Our GitHub repository:

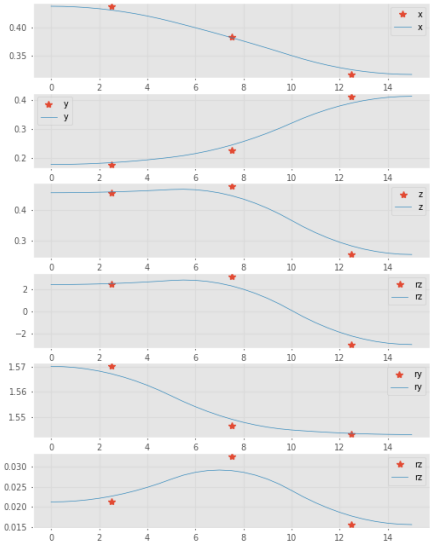
`https://github.com/Kamik18/Project-in-Advanced-Robotics`

[14] Learning from Demonstration:

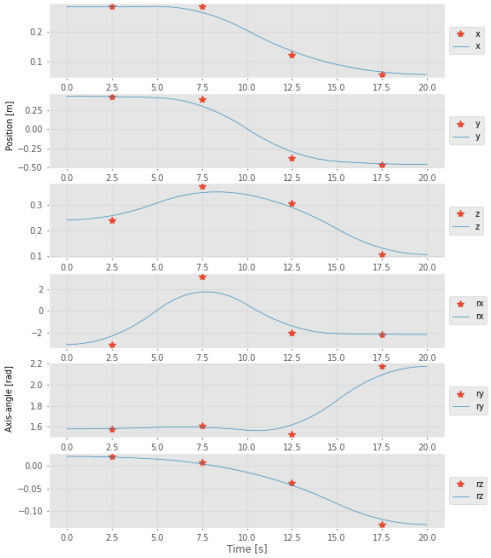
Self-study in Advanced Robot Control by Iñigo Iturrate.

7 Appendix

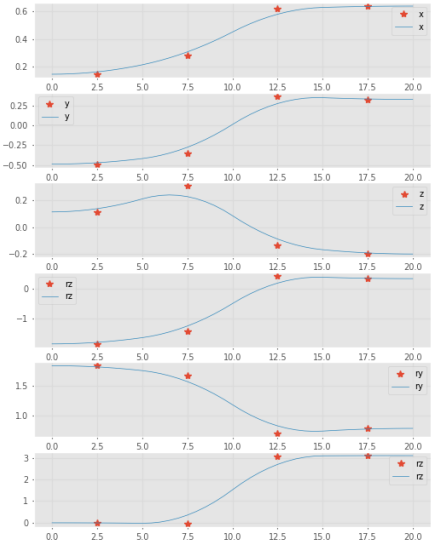
Blend graphs



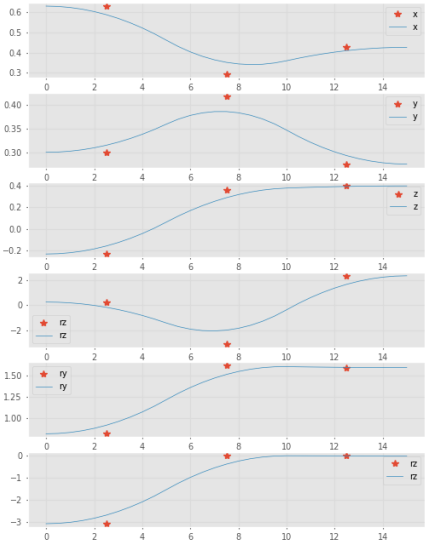
(a) Blend from home to point B



(b) Blend from point B to point A



(c) Blend from point A to point B



(d) Blend for point B to home

Figure 24: *Blends through via points*

TCP graph

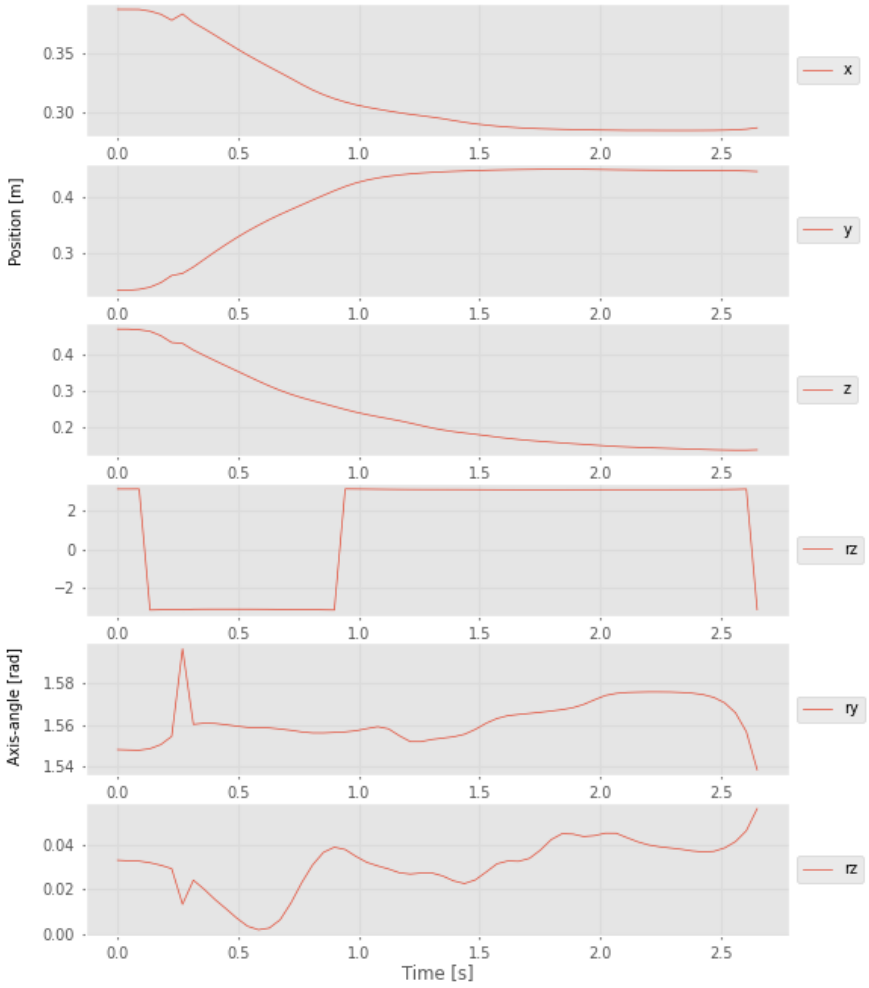


Figure 25: *TCP orientation issue with inverted values*