POLITECNICO DI TORINO

Master's Degree in Biomedical Engineering

Master's Degree Thesis

Development of a controller for robotic manipulation through learning from demonstration

Supervisors

Prof. ALESSANDRO RIZZO

Prof. DOMENICO PRATTICHIZZO

Ing. ENRICO TURCO

Candidate

GABRIELE GIANNINO

JULY 2024

Abstract

This thesis proposes a Learning from Demonstration (LfD) approach designed to generalize and extract relevant features of desired motion trajectories for robotic manipulation tasks, with the specific objective of learning a sliding and picking task exploiting environmental constraints and force sensor data. Learning from Demonstration is a powerful approach in robotics, since robots can acquire new skills by observing, modeling and imitating human demonstrations of a task. This method leverages human expertise to teach robots complex movements, reducing the need for explicit programming. Research in the field of Lfd is facilitating, even for non-expert users, to teach new tasks to robots with few demonstrations, enabling robotics and skills learning to be used in a variety of fields of applications and dynamic environments.

The developed method is based on probabilistic modeling, specifically a mixture of Hidden Markov Model(HMM) and Gaussian Mixture Model(GMM). Pose data and force data are processed,cleaned and aligned in time through Dinamic Time Warping. A crucial role in the method is represented by forces and torques data sensed by the force sensor during the contact with the environment, through which the movement is divided in three primitives. After this division, a continuous HMM model is trained to be able to switch between primitives and enabling a high level control of the trajectory. At a lower level a GMM for each primitive is trained to model the pose of the end effector and predicting it, giving time as input, with Gaussian Mixture Regression.Both HMM and GMM models are tested on training and test set in order to fine tune parameters, such as the number of Gaussian distributions.

The proposed approach, combining these two models, is implemented and validated through simulations and real-world experiments, showing good performances in predicting the primitives and generalizing the trajectory. The main advantage of this algorithm is its ability to generalize from few demonstrations, resulting in high-quality motion reproduction. The probabilistic approach enables modeling of complex trajectories with a limited number of demonstrations, leveraging force sensor data and environmental constraints to enhance robustness.

Table of Contents

List of Figures

List of Tables

Acronyms

Lfd

Learning from demonstrations

AI

artificial intelligence

GMM

Gaussian Mixture Model

GMR

Gaussian Mixture Regression

HMM

Hidden Markov Model

EC

Environmental constraints

IIT

Istituto Italiano di tecnologia

IRL

Inverse Reinforcement Learning

DMP

Dinamic Movement Primitives

CNN

Convolutional Neural Networks

DTW

Dinamic Time Warping

LSTM

Long Short Term Memory

SVM

Support Vector Machines

PDF

Probability Density Function

BIC

Baesian information criterion

MSE

Mean Squared Error

RMSE

Root Mean Squared Error

SSG

Soft Scoop Gripper

Chapter 1

Introduction

1.1 Thesis context and motivation

Learning from demonstration (LfD) has emerged as a pivotal paradigm in the field of robotics, enabling robots to acquire complex skills through the imitation of an expert's demonstrations. Unlike traditional robot programming methods, which need detailed coding of every action and sequence, LfD allows for the implicit learning of tasks from demonstrated behaviors. This makes it particularly advantageous in scenarios where ideal behavior cannot be easily scripted or defined as an optimization problem but can be shown through demonstrations. The ability of LfD to generalize from a few examples and adapt to new and unstructured environments underscores its potential to revolutionize robotic programming and automation.

The adoption of LfD is especially promising in domains such as manufacturing, healthcare, and domestic robotics, where the need for adaptive and intelligent robotic systems is paramount. By reducing the requirement for extensive programming expertise, LfD facilitates the involvement of non-expert robot programming, thereby democratizing the use of robots across various fields. This paradigm shift not only enhances the versatility and applicability of robotic systems but also accelerates the deployment of robots in real-world applications.

1.2 Thesis objective

The primary objective of this thesis is to develop a controller for robotic manipulation tasks using learning from demonstrations. Specifically, the goal is to reproduce a task of picking objects from a stack using environmental constraints (EC) and force sensor's data. The research is based on the Franka Emika Panda robot, equipped with the Soft Scoop gripper as the end effector, developed at Italian

Institute of Technology (IIT) and a force sensor.

To achieve this, 15 demonstrations were conducted with different objects, where the robot was manually guided to perform the task. Data on the robot's pose and force during each demonstration were recorded and subsequently preprocessed using MATLAB. The core of the thesis revolves around the development and implementation of an algorithm based on Hidden Markov Models (HMM) and Gaussian Mixture Models (GMM). This algorithm is designed to segment the demonstrated task into three movement primitives, with transitions determined by the HMM based on force sensor data. At a lower level, the GMM/GMR (Gaussian Mixture Regression) models each primitive's trajectory from time to end-effector pose.

1.3 Thesis structure

- **Chapter 2: State of the Art** This chapter provides an extensive review of the existing literature on learning from demonstration, covering recent advancements, various methodologies, and their applications in robotic manipulation. It includes an updated taxonomy and characterization of existing methods, emphasizing their strengths and limitations.

- **Chapter 3: Materials** This chapter details the experimental setup, including the hardware and software used. It describes Franka Emika Panda robot, the Soft Scoop gripper, and the force sensor, as well as the data acquisition systems and preprocessing tools employed in the study, such as Matlab,Python, Coppelia Sim and related libraries.

- **Chapter 4: Methods** This chapter outlines the process of data collection and preprocessing, focuses on the design and implementation of the HM-M/GMM algorithm, and explains how the algorithm segments the task into movement primitives, switches between them and models the trajectories for each primitive.

- **Chapter 5: Results** This chapter presents the results of the developed controller. It includes the description of the simulations set-up and an analysis of the robot's performance, using plots of calculated metrics and obtained trajectories. This is done on the two models alone and integrated, in order to highlight the differences.

- **Chapter 6: Discussions,applications and future works** This chapter discusses potential improvements and extensions to the current work. It suggests future research directions, including the integration of additional sensory inputs, improvements in the algorithmic framework, and applications to more complex tasks and environments in different fields.It discusses the application of this approach to various fields with a deeper view into the biomedical field.

- **Chapter 7: Conclusions** This chapter summarizes the key findings of the thesis, highlighting the contributions to the field of learning from demonstration and robotic manipulation. It reflects on the challenges encountered and the solutions proposed, emphasizing the broader implications of this work for the development of adaptive and intelligent robotic systems.

Chapter 2

State of the art

Learning from Demonstration (LfD) is a paradigm in robotics where robots acquire new skills by observing human demonstrations rather than being explicitly programmed. This approach leverages human intuition and expertise to create adaptable and efficient robotic systems . LfD is significant in robotics as it simplifies the programming process, making it accessible to non-experts and enabling robots to learn complex tasks quickly. This approach is particularly beneficial in dynamic environments where traditional programming is impractical . This chapter aims to provide a comprehensive overview of LfD, detailing various methods for acquiring demonstrations, classifying different learning methods, and exploring advanced techniques. Additionally, it will highlight the integration of environmental constraints and force data into LfD.

2.1 Learning from demonstrations

LfD involves teaching robots by example, allowing them to replicate demonstrated behaviors. Key principles include generalization from limited demonstrations, handling variability in human demonstrations, and ensuring robust performance in diverse scenarios . Traditional robot programming involves manual coding of every action, which is time-consuming and inflexible. In contrast, LfD allows robots to learn from human examples, significantly reducing development time and increasing adaptability . LfD has broad applications, including industrial automation, healthcare, and service robotics. It enables rapid deployment of robotic systems in tasks such as assembly, surgery, and household chores, enhancing productivity and safety. For all of these reasons Lfd has gone through an increase in research in last years, as this chart shows[1].(see Fig 2.1)

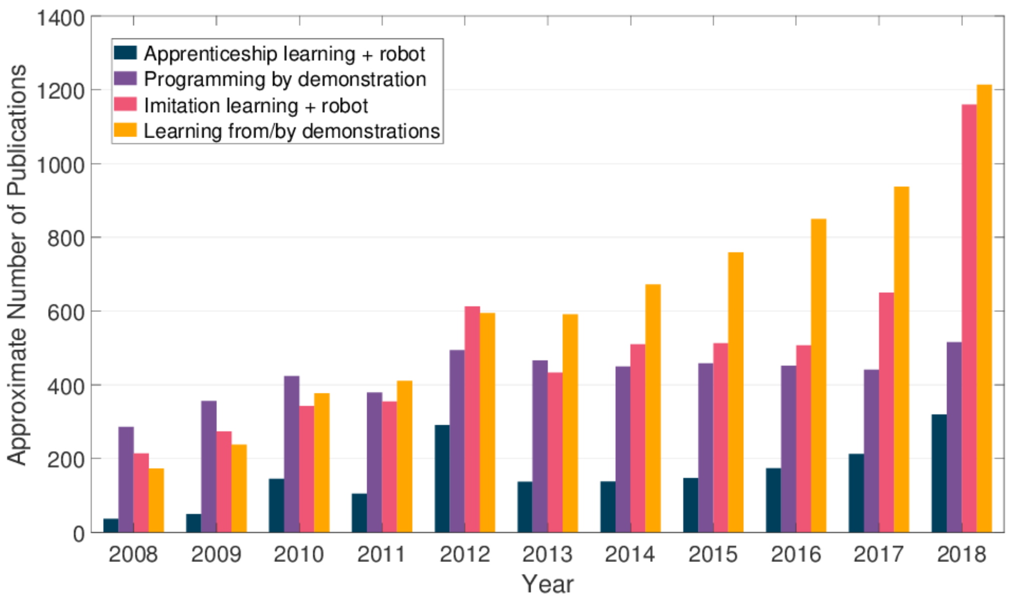

Figure 2.1: Evolution of the research in Lfd [1]

2.1.1 Methods for Acquiring Demonstrations

As described in Fig. 2.2, from Figure 3 in [1], the methods used to acquire demonstrations are listed below:

- Kinesthetic Teaching: it involves physically guiding the robot through the desired motions. This method provides precise control over the demonstration but can be physically demanding for the demonstrator .
- Teleoperation: the human controls the robot remotely, providing demonstrations through a user interface. This method allows for demonstrations in hazardous environments but may suffer from latency and precision issues .
- Passive Observation: it involves the robot watching the human perform a task without interaction. This method is less intrusive but may require sophisticated perception systems to accurately interpret the demonstrations .
- Active and Interactive Demonstrations: it involves the robot engaging with the human during the demonstration, asking questions, or requesting repetitions to clarify ambiguities. This interactive approach can enhance the quality of learning but requires advanced communication capabilities .

In our case, due to the level of difficulty of the task in exam we decided to use kinesthetic teaching, guiding our robot through 15 demonstration with different objects.

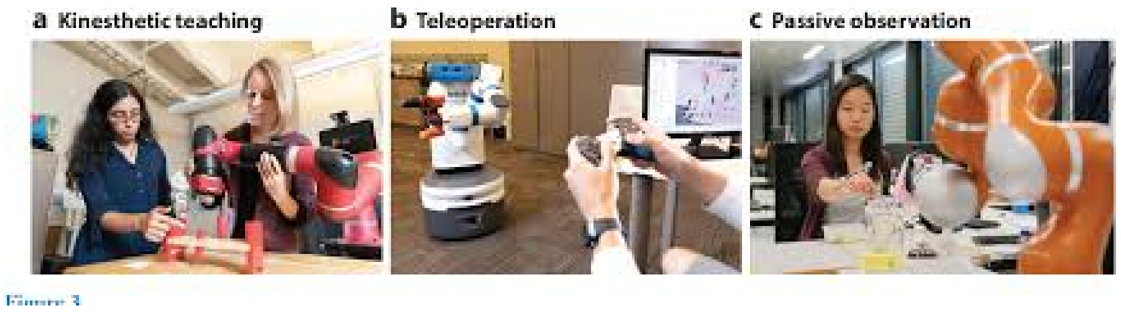

Figure 2.2: Demonstration's acquiring methods [1]

2.1.2 Classification of Learning Methods in Lfd

Policy Learning

Policy learning focuses on mapping states to actions, allowing the robot to determine the appropriate action in any given state.

- Time-based Policies Time-based policies rely on a fixed sequence of actions, often used in tasks with strict temporal requirements .
- State-based Policies State-based policies adapt actions based on the current state, providing flexibility and robustness in dynamic environments .
- Raw Observation-based Policies These policies use raw sensory data to make decisions, often employing deep learning techniques to process complex inputs .

Cost or Reward Function Learning

This approach involves learning a cost or reward function that the robot uses to evaluate the desirability of different actions.

- Trajectory Optimization Trajectory optimization seeks to find the most efficient path to achieve a task, balancing speed, safety, and energy consumption .
- Inverse Reinforcement Learning (IRL) IRL infers the underlying reward structure from observed behavior, allowing robots to understand the demonstrator's goals and preferences .

Plan Learning

Plan learning involves learning high-level plans or strategies that consist of sequences or hierarchies of actions.

- Primitive Sequences This method focuses on learning sequences of basic actions or primitives to complete tasks .
- Primitive Hierarchies Primitive hierarchies organize these actions into structured layers, enabling more complex behavior synthesis .
- Multilevel Learning Outcomes Multilevel learning integrates both high-level plans and low-level actions, providing a comprehensive approach to task execution.

This classification is taken from Table 2 in the review [1].

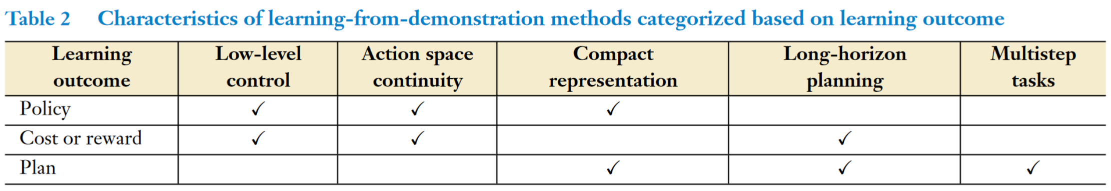
Table 2 Characteristics of learning-from-demonstration methods categorized based on learning outcome

Learning outcome	Low-level control	Action space continuity	Compact representation	Long-horizon planning	Multistep tasks
Policy	✓	✓	✓		
Cost or reward	✓	✓		✓	
Plan			✓	✓	✓

Figure 2.3: Classification of learning methods [1]

Following this classification each specific method or algorithm can be considered to be part of one of the said groups or more than one. In the next paragraph the most used methods will be briefly discussed, knowing that sometimes in the literature can be find mixtures of these methods or combinations of them.

2.2 Detailed Techniques in LfD

2.2.1 Probabilistic Approaches

-
- Higher-level Probabilistic Models Higher-level probabilistic models capture abstract representations of tasks and behaviors, facilitating planning and decision-making under uncertainty .
- Lower-level Probabilistic Models Lower-level probabilistic models focus on the fine-grained aspects of task execution, such as motion control and sensory processing, providing robustness to variations in the environment and execution.

Both GMM and HMM can be considered part of probabilistic models, HMM at a higher level, GMM at a lower level. Following the classification highlighted in [2] HMM can be classified as symbolic learning, while GMM as encoding skills at trajectory level. Both can be considered as statistical learning.

Gaussian Mixture Models (GMM)

GMMs are probabilistic models that represent the distribution of data points in the demonstration space. They are useful for capturing the variability in demonstrations and are widely used in gesture recognition and trajectory modeling. They are used as clustering methods when a more deterministic method such as k-menans would be inaccurate in the modeling of a distributed dataset(see Fig.2.4) . Gaussian mixture regression allows to obtain multivariate values given one of the variables, so it can be used to predict or generalize a trajectory knowing time, for example. Pros and cons of this method are here highlighted:

- Pros:
    1. Ability to model complex, multivariate data distributions
    2. Flexibility in representing different types of demonstrations
- Cons:
    1. Computational complexity can be high
    2. May require a large amount of data to accurately model distributions.

Hidden Markov Models (HMM)

HMMs are statistical models that represent sequences of observations with hidden states (see Fig 2.5). They are particularly effective for tasks that involve temporal dependencies, such as speech recognition and sequential decision-making .

- Pros:
    1. Good at handling time series data and temporal patterns
    2. Robust to variations in demonstration speed and execution
- Cons:
    1. Requires careful tuning of model parameters
    2. Can be computationally intensive for long sequences

Dynamic Movement Primitives (DMP)

DMPs provide a framework for representing and generating complex movements through a combination of learned attractor dynamics and external inputs. They are widely used for motor skill learning and adaptation in robotics .

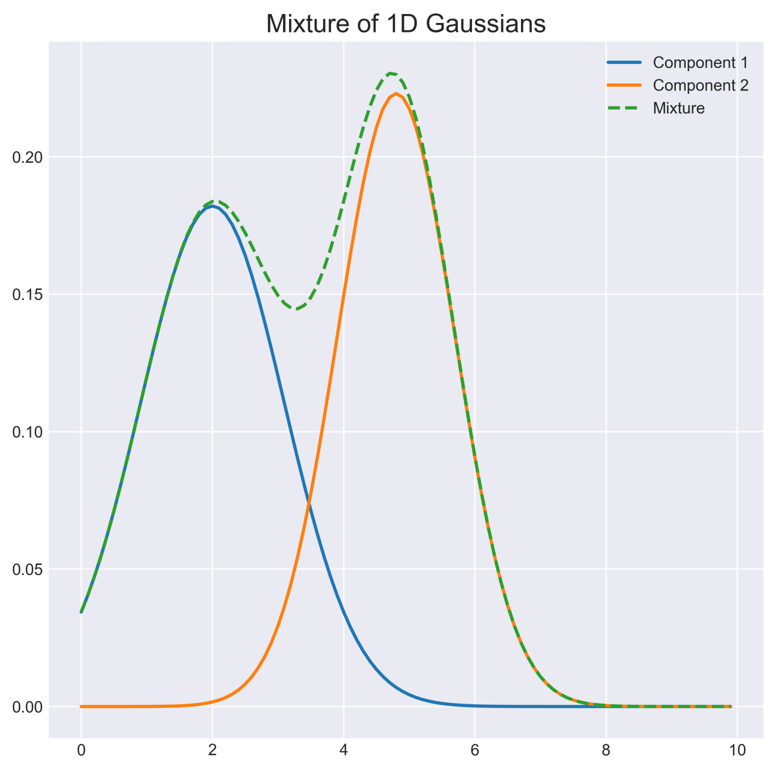

Figure 2.4: GMM

- Pros:
  1. Flexibility in encoding various types of movements
  2. Ease of generalization and adaptation to new situations
- Cons:
  1. Limited expressiveness for highly complex tasks
  2. Requires careful design of attractor dynamics

2.2.2 Deep Learning Techniques

Deep learning techniques leverage neural networks to model complex relationships in data. In LfD, they are used for tasks such as visual recognition, policy learning,

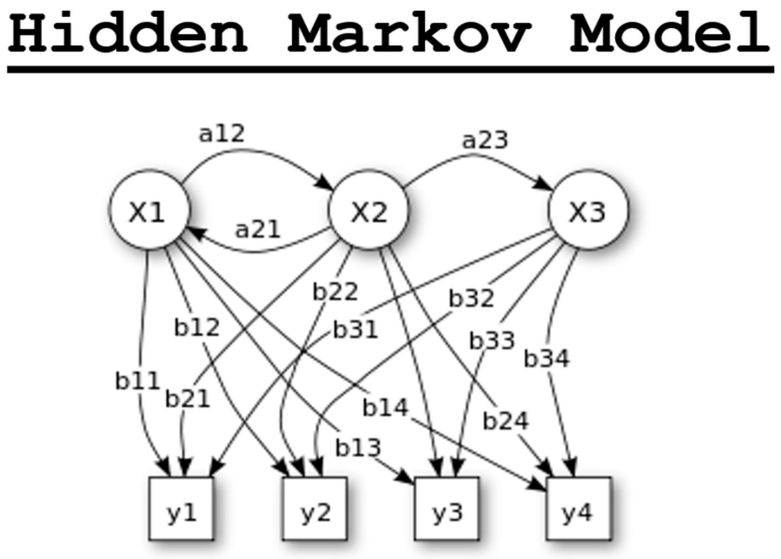

Figure 2.5: HMM

and end-to-end skill acquisition . Deep learning techniques, such as Convolutional Neural Networks(CNN), probably represent the future in everything where vision is included, in Lfd without vision is still not the gold standard.

- Pros:
    1. High capability for learning from raw sensory data
    2. Potential in discovering intricate patterns and representations
- Cons:
    1. Requires large amounts of data and computational resources
    2. Can be difficult to interpret and debug learned models
    3. It is widely used in computer vision

2.3 Environmental Constraints and Force Data

2.3.1 Role of Environmental Constraints in LfD

Environmental constraints are critical in ensuring that learned behaviors are feasible and safe in real-world scenarios. They guide the robot to operate within acceptable limits and avoid hazards. Moreover can act as a guidance in specific tasks where they are used to model the trajectory. In literature the implementation of EC is often based on force sensor's data.

2.3.2 Incorporating Force Data into Learning

Force data provides essential information about the interaction between the robot and its environment. Incorporating force data into LfD enables the robot to perform tasks that require delicate manipulation and adaptation to varying resistance. It can be crucial in fine tuning the control in particular tasks

Case Studies and Applications

Case studies demonstrate the practical applications of integrating environmental constraints and force data in LfD. Examples include robotic assembly, surgical assistance, and household chores, where precise and adaptive behaviors are crucial.By focusing on the constraints imposed by the environment, robots can adapt learned tasks to new situations with similar constraints, as shown in[3]. Force data further aids in refining the robot's actions, ensuring robust execution in contact-rich tasks.(see Fig 2.6)

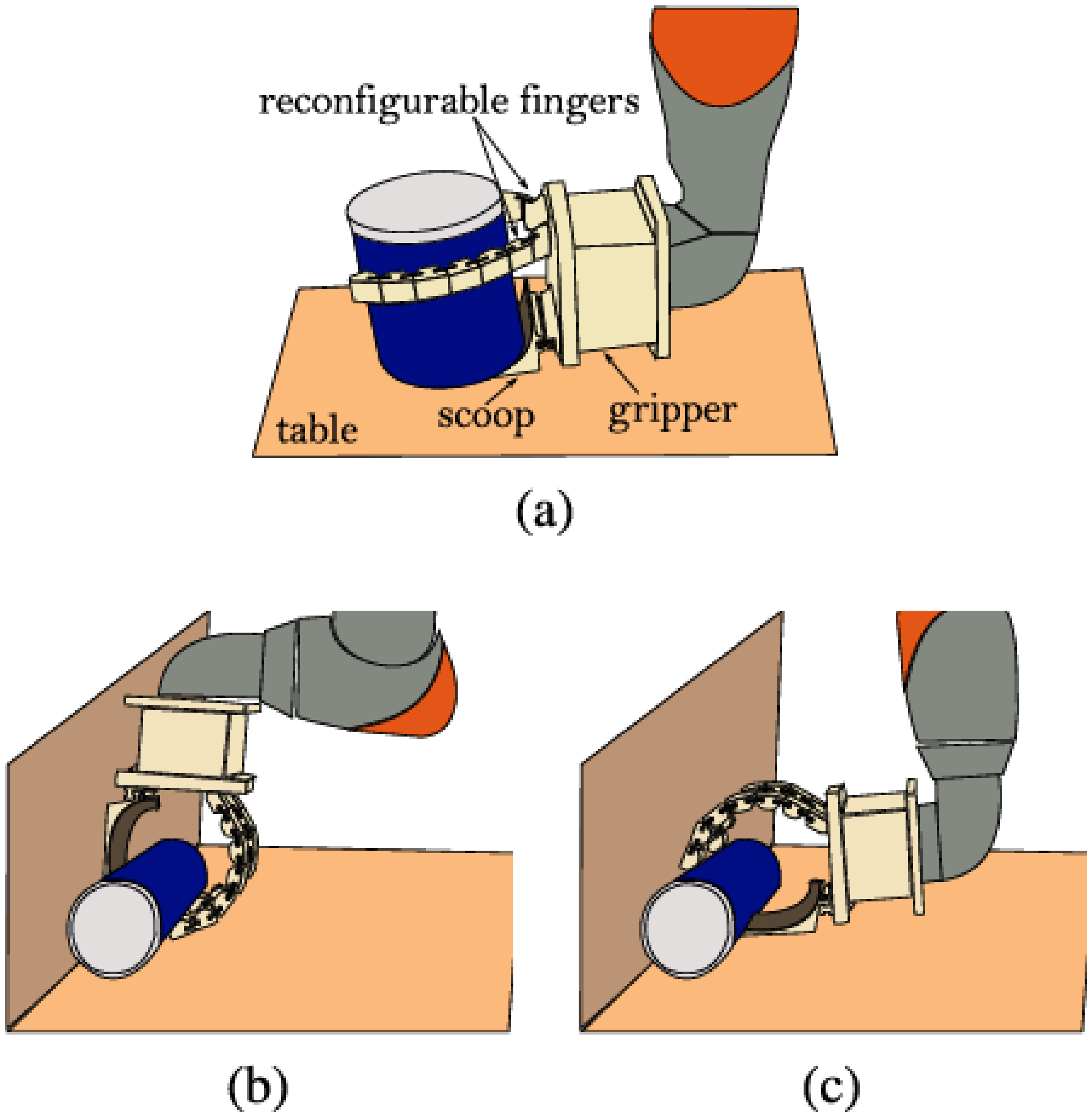

Figure 2.6: Environmental constraints with Soft Scoop Gripper [4]

Chapter 3

Materials

This chapter provides a comprehensive overview of the hardware and software components used in this research. The Franka Emika Panda robot, equipped with the Soft Scoop Gripper and Gamma F/T sensor, formed the core of the robotic setup.The experimental set-up was made of a stack of boxes and objects on top. The software stack, primarily based on ROS, C++, MATLAB, Python and CoppeliaSim facilitated data acquisition, processing, and model implementation, enabling the successful execution and learning of complex robotic manipulation tasks

3.1 Hardware Components

3.1.1 Franka Emika Panda Robot

The robotic arm used in this research is the Franka Emika Panda. This advanced robotic arm features 7 degrees of freedom, allowing for precise and flexible movement. It has a payload capacity of up to 3 kg and is equipped with integrated torque sensors in all joints, which enhance its sensitivity and safety in human-robot interactions. The Panda robot is known for its high precision making it ideal for complex manipulation tasks.

Specifications:

- **Degrees of Freedom:** The Panda robot has 7 degrees of freedom, allowing for complex and flexible movements.
- **Payload Capacity:** The robot can handle payloads of up to 3 kg, making it suitable for a variety of tasks.
- **Precision:** It offers a repeatability of ±0.1±0.1 mm, ensuring high accuracy in manipulation tasks.

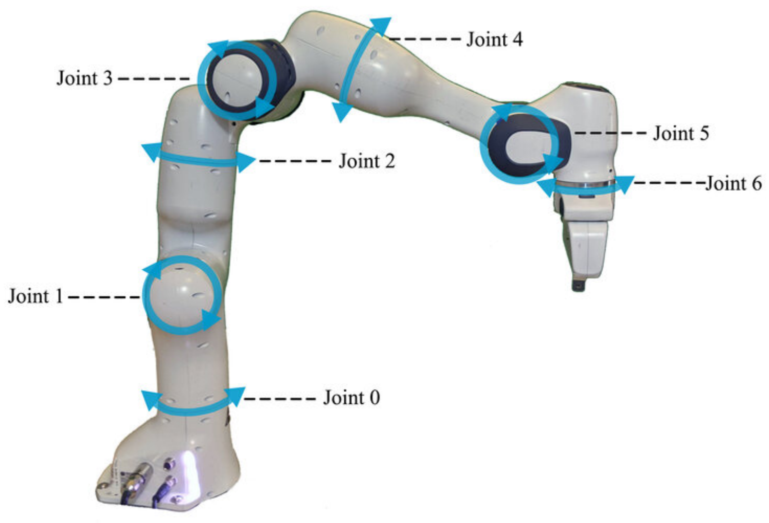

Figure 3.1: Franka Emika Panda 7 degrees of freedom from [5]

- **Joint Range:** The range of each joint varies, but collectively they provide a wide operational workspace.
- **Joint Velocity:** Maximum joint velocities range from 2.175 rad/s to 2.610 rad/s, depending on the joint.
- **Joint Torque:** Maximum joint torques range from 87 Nm to 176 Nm, depending on the joint.
- **Weight:** The robot arm itself weighs approximately 18 kg, allowing for easy installation and relocation.
- **Interface:** The robot supports Ethernet, USB, and other communication interfaces for integration and control.

Control Mechanisms: The control system of the Franka Emika Panda robot is sophisticated and designed to ensure precise, smooth, and safe operation.

1. Control Unit: The control unit houses the central processing unit (CPU) and other electronics that manage the robot's operations. Key functionalities include:

- **CPU:** Acts as the brain of the robot, executing control algorithms and processing sensor data. It performs calculations, manages tasks, and ensures real-time responsiveness.

- **Communication Interfaces:** Includes Ethernet and USB ports for connectivity with external devices and networks.

- **Safety Features:** Integrated safety protocols ensure safe operation around humans and in various environments.

2. Torque Sensors: The Panda robot is equipped with torque sensors in all joints, providing real-time feedback on the forces and torques experienced during operations. This feedback is crucial for:

- **Collision Detection:** Ensures the robot can detect and respond to unexpected obstacles, enhancing safety.

- **Force Control:** Allows the robot to perform delicate tasks that require precise force application.

- **Adaptive Control:** Enables the robot to adapt its movements based on the forces it encounters, improving interaction with objects and environments.

3. Control Software: The control software for the Franka Emika Panda robot is built on the Robot Operating System (ROS), providing a flexible and powerful platform for developing and executing control algorithms.

- **Franka Control Interface (FCI):** The FCI is a real-time interface that allows users to control the robot at a high level of precision. It provides access to the robot's state, including joint positions, velocities, torques, and external forces.

- **Motion Generation:** The software includes built-in motion generation capabilities, allowing for smooth and precise trajectory execution.

- **Programming:** Users can program the robot using various programming languages supported by ROS, including C++ and Python.

- **Integration:** The control software can be integrated with external systems and sensors, enabling complex applications and research experiments.

Role in Experiments: The Franka Emika Panda robot was utilized to perform demonstrations and collect data for the learning from demonstration (LfD) approach. It was responsible for executing the sliding and picking tasks, interacting with various objects, and providing the necessary force and position data through its sensors.

3.1.2 Soft Scoop Gripper

Description: As outlined in [4], the Soft Scoop Gripper (SSG) is an innovative, non-anthropomorphic, underactuated robotic gripper. It features two soft-rigid fingers and a flat surface (referred to as the "scoop") connected to the hand palm via a flexible hinge. The fingers are driven by a single motor through a tendon-driven differential system, allowing them to flex simultaneously. The scoop can be actuated separately by another motor, enabling it to close towards the fingers. This design, which incorporates the scoop as an integral constraint, is optimized for leveraging environmental constraints and ensuring secure grasping of objects placed on it. From a control perspective, the SSG allows for simple commands to open and close the fingers[6]. The gripper is made from flexible materials that allow it to conform to the shape of the objects it handles. It has been designed at University of Siena and developed in collaboration with Istituto Italiano di Tecnologia in Genova.

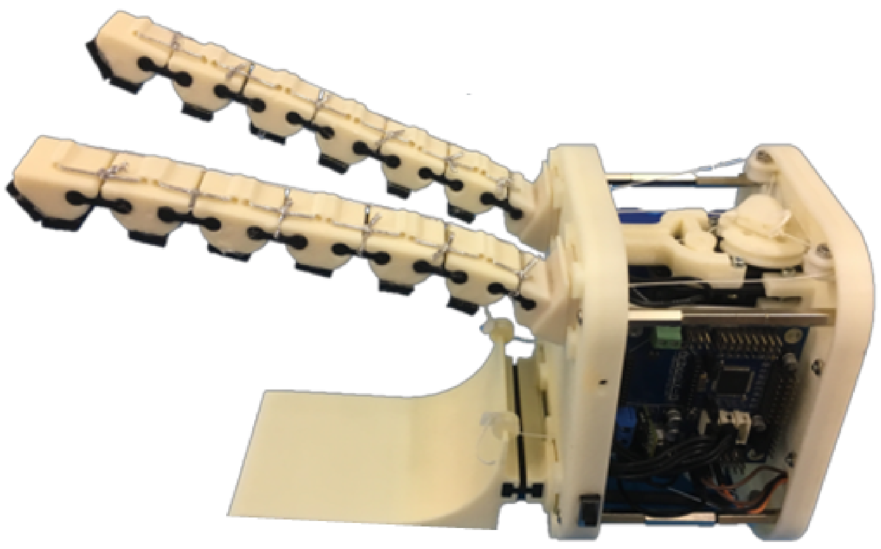

Figure 3.2: Soft Scoop Gripper from [6]

Integration: The Soft Scoop Gripper was integrated with the Franka Emika Panda robot to enhance its manipulation capabilities. This integration involved attaching the gripper to the robot's end effector and calibrating the system to ensure precise control over the gripping actions.

3.1.3 Gamma F/T Sensor from Schunk

Description: The Gamma Force/Torque (F/T) sensor from Schunk is a high-precision sensor that measures the forces and torques exerted during manipulation

SCHUNK
FT
Force/torque sensor
Rigid 6-axis force/torque sensor for precision measuring in all six degrees of freedom
Variants
Do you have a question?

Figure 3.3: Gamma Force/Torque (F/T) sensor from [7]

tasks. The technical specifications of the sensor are given below in 3.1

Table 3.1: Gamma F/T Sensor from Schunk Specifications [7]

Specification	**Value**
Measurement Ranges	Fx, Fy: ±200 N Fz: ±250 N Tx, Ty: ±10 Nm Tz: ±12 Nm
Resolution	Fx, Fy: 0.01 N Fz: 0.02 N Tx, Ty: 0.001 Nm Tz: 0.001 Nm
Overload Capacity	Fx, Fy: 300% Fz: 300% Tx, Ty: 300% Tz: 300%
Nonlinearity	0.2% of full scale
Hysteresis	0.2% of full scale
Crosstalk	<2%
Sampling Rate	Up to 7000 Hz
Operating Temperature Range	-10°C to 45°C
Communication Interface	Ethernet, EtherCAT
Weight	370 g

Data Acquisition: Sensor data was collected during the demonstrations, capturing the forces and torques experienced by the gripper. This data was essential for training the probabilistic models and understanding the interaction between the gripper and the objects.

3.1.4 Experimental Setup

Environment: The experimental setup consisted of the Franka Emika Panda robot mounted on a wooden table. At a distance of approximately half a meter, a stack of objects was placed, composed of three boxes made of plastic or paper. These boxes acted as environmental constraints. On top of the last box, five different objects were placed: a book, a clamp, a calculator, a plate, and a wooden block. These objects were used for performing three demonstrations each, involving sliding and picking actions. Here below an image of the objects used in demonstrations and experiments is given. see Fig. 3.4 Objects specifications are summarized in table 3.2

Table 3.2: Object Dimensions and Weight 3.4

Object	**Dimensions (cm)**	**Weight (g)**
1. Book	19 x 12 x 3	395
2. Notebook	14 x 9 x 2	140
3. Clamp	12 x 9 x 2	52
4. Plate	18 (diameter)	46
5. Wooden Block	14 x 5 x 6	225

Calibration: The calibration process for the robot and sensors involved ensuring accurate data collection. This included transforming the end effector to the scoop configuration, rebalancing the robot to handle kinesthetic teaching, and setting up ROS nodes for data acquisition. In order to handle kinesthetic teaching mode, the robot has to compensate the gravity to mantain the balance during the movement.

3.2 Software Components

3.2.1 Robot Operating System (ROS)

Description: ROS, an open-source meta-operating system designed specifically for robots, offers a comprehensive suite of services akin to those found in conventional operating systems. These services encompass essential functionalities such as hardware abstraction, precise low-level device control, integration of commonly-used features, efficient inter-process communication through message-passing mechanisms, and streamlined package management capabilities.

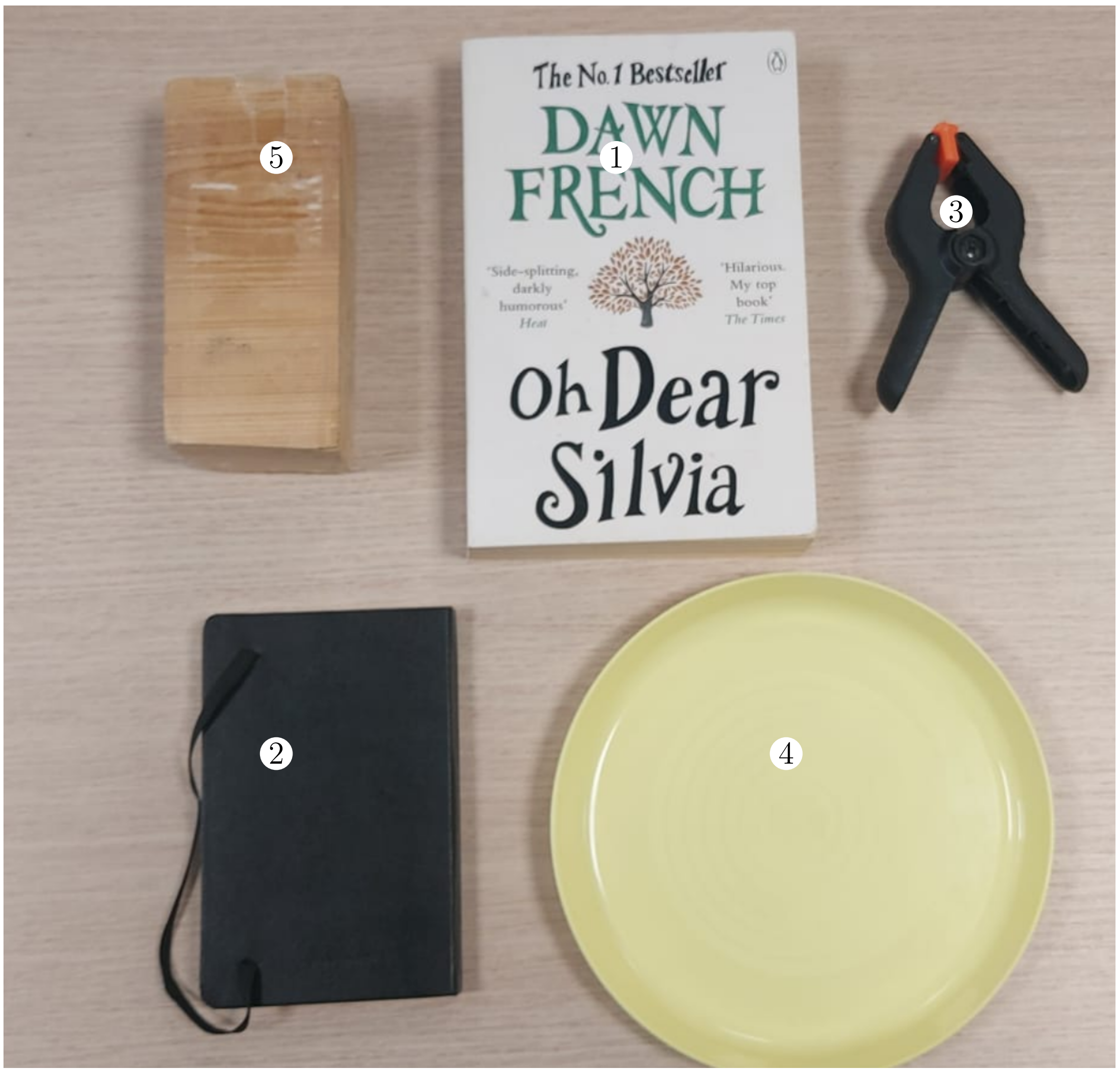

Figure 3.4: Objects used in the experiment.

Additionally, ROS provides a rich ecosystem of tools and libraries tailored for tasks such as code acquisition, building, writing, and execution across multiple computing platforms. In terms of communication, ROS supports multiple styles including synchronous Remote Procedure Call (RPC)-style communication via services, asynchronous data streaming over topics, and centralized data storage facilitated by a Parameter Server.[8] Although not inherently designed as a real-time framework, ROS presents the flexibility to integrate with real-time code, ensuring synchronization with time-sensitive operations.

The architectural foundation of ROS relies on a decentralized Peer-to-Peer (P2P) communication model, fostering seamless interaction between distinct nodes. In the context of ROS, nodes represent individual software entities tasked with specific functionalities, capable of execution on single or multiple computing platforms interconnected within a network. This distributed architecture inherently promotes modularity, scalability, and fault tolerance, critical for the development of complex robotic systems.

Implementation: ROS was used to control the Franka Emika Panda robot and manage data flow during the experiments. It facilitated the communication between the robot, sensors, and control algorithms, ensuring smooth operation and data acquisition.

3.2.2 MATLAB

MATLAB played a crucial role in data processing and analysis. Its built-in functions and toolboxes were used extensively for filtering, visualizing data, and implementing probabilistic models like HMM and GMM.

Key Functions and Toolboxes:

- **Signal Processing Toolbox:** Used for filtering and preprocessing force data.
- **Statistics and Machine Learning Toolbox:** Utilized for implementing GMM and HMM.
- **Custom Scripts:** Developed for data normalization, transformation, and trajectory analysis.

Some functions related to HMM and GMM were modified from Calinon's code [9]

3.2.3 Python

Python was initially used for implementing some of the probabilistic models and data manipulation due to its extensive libraries and ease of use. However, MATLAB was preferred for its comprehensive toolboxes and ease of integration with the rest of the experimental setup.

Key Libraries:

- **NumPy:** For numerical computations and array manipulations.
- **Pandas**: For data manipulation and analysis.
- **SciPy:** For advanced mathematical functions and optimization.
- **scikit-learn:** Used for preliminary implementations of GMM and HMM.
- **Matplotlib**: For creating visualizations and plotting data.

3.2.4 CoppeliaSim

CoppeliaSim, formerly known as V-REP, is a versatile robot simulation software used for developing, testing, and validating the robot's control algorithms in a simulated environment. Its powerful simulation capabilities allowed for precise modeling of the physical interactions and validation of the control strategies before deploying them on the actual hardware.

Features:

- **Customizable Simulation Environment:** Allows for precise modeling of physical properties such as friction, mass, and inertia.
- **Scripting Functionality:** Embedded Lua scripts for controlling simulations, implementing control algorithms, and simulating sensor feedback.
- **Integration with ROS:** Enables seamless control of simulated robots within CoppeliaSim through ROS.

The software was used in the experimental part, building a reproduction of the real world environment and deploying simulations of the obtained model to different objects to evaluate performances before applying it in real world.(See Fig. 4.1)

3.2.5 Data Processing and Storage

Data acquired during the experiments were stored in ROS bag files, which are a convenient format for recording and replaying ROS message data. The data from these bag files were then converted to a format suitable for MATLAB and Python analysis.

Key Processes:

- **Data Recording:** ROS bag files were used to record demonstration data, including robot joint states, force/torque sensor readings, and gripper states.
- **Data Conversion:** Scripts were developed to convert ROS bag files into MATLAB and Python-friendly formats for further analysis.

3.2.6 Provided MATLAB Code

For various preprocessing steps, data normalization, and probabilistic modeling, MATLAB code from Calinon and custom implementations were utilized. This included functions for dynamic time warping (DTW), GMM, HMM, and Gaussian Mixture Regression (GMR). The main code for training the two models is given in A.2 and A.1.

Chapter 4

Methods

This chapter describes the complete process implemented to develop a robotic manipulation controller with learning from demonstrations and exploiting environmental constraints. It details the demonstrations acquisition, the preprocessing of pose and force data, the training of the HMM/GMM model and the testing on acquired data.

4.1 Setup and Demonstrations

4.1.1 Environmental Setup

The experimental setup for the Learning from Demonstration (LfD) approach was designed to facilitate the demonstration and data acquisition process. The setup consisted of the Franka Emika Panda robotic arm, which was securely attached to a sturdy wooden table to ensure stability during the demonstrations. The robot was positioned at a fixed location on the table, with its base at a distance of approximately half a meter from the stack of objects used for the demonstrations.

The stack of objects served as environmental constraints, providing a structured and consistent context for the robot's task execution. This stack was composed of three boxes made of either plastic or paper, each of varying dimensions. The boxes were carefully selected and arranged to create a stable and repeatable environment for the demonstrations. The height of the stack was approximately 40 cm from the surface of the table, ensuring that the objects placed on top were within the robot's reach.

On top of the stacked boxes, five different objects were positioned sequentially for the demonstrations. These objects were chosen to represent a variety of shapes and sizes, adding complexity and diversity to the task. The objects included a book, a clamp, a calculator, a plate, and a wooden block. Each object was placed on the stack one at a time, and three demonstrations were performed for each object, resulting in a total of 15 demonstrations.

The image below provides a visual representation of the demonstration setup, reproduced in CoppeliaSim software, highlighting the arrangement of the Franka Emika Panda robot, the stack of boxes, and the objects used for the demonstrations. (see Fig 4.1) By maintaining a consistent and controlled environment, this setup

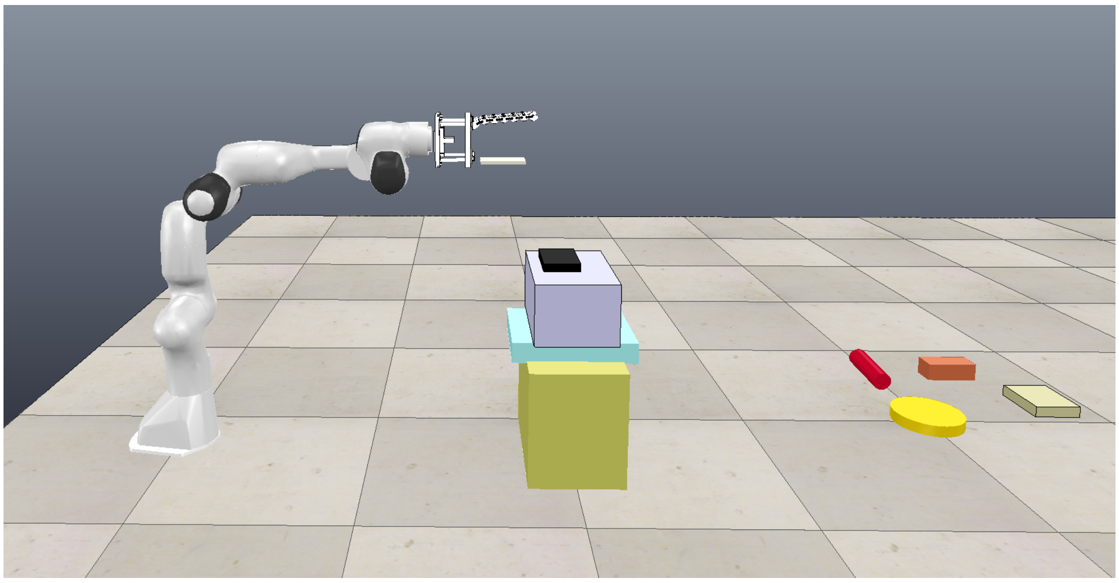

Figure 4.1: Environmental set up in Coppelia Sim

allowed for precise data acquisition, crucial for the subsequent processing and modeling steps in the LfD approach.

4.1.2 Robotic System Configuration

As shown in Fig. 4.1 the Franka Emika Panda robotic arm is used to perform demonstrations with the Gamma F/T Schunk sensor attached and the Soft Scoop Gripper as end effector. After setting the hardware up, the software configuration is changed using the integrated Franka API's to handle the different balance and the new end effector position. The ROS nodes are created to control the robot, acquire data from it and from the force sensor. Finally the robotic arm is put in Kinesthetic teaching mode to perform demonstrations with the robot only supporting its own weight throughout the trajectory imposed by the teacher. As can be seen in Fig.4.2 the robot is set up with the force sensor and the new end effector, but the reference frames are not aligned, so there is the need to apply some transformations as described in subsection 4.2.2 .

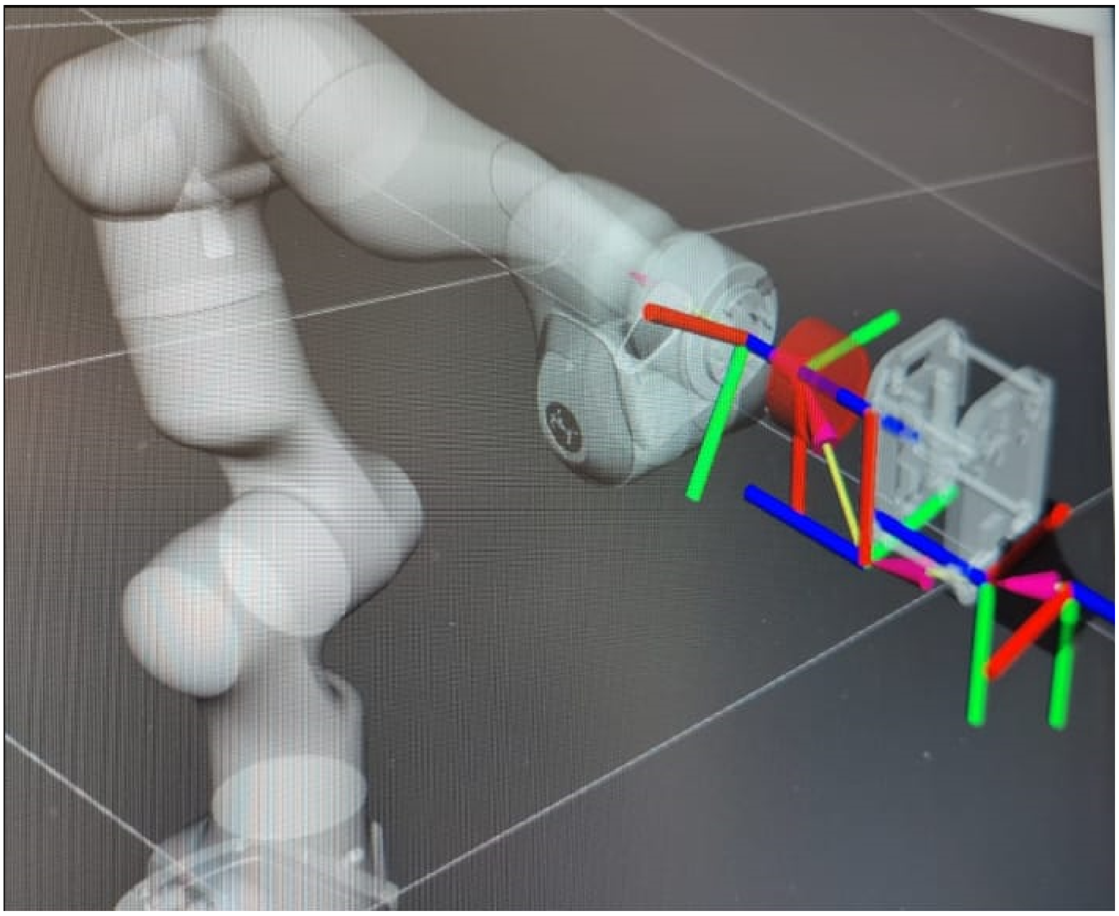

Figure 4.2: Robot set up and reference frames

4.1.3 Demonstration Procedure

The demonstrations were performed using a method known as kinesthetic teaching. This technique involves a human operator physically guiding the robot's arm to execute the desired task, allowing for precise recording of the motion trajectories and force data. The Franka Emika Panda robotic arm, equipped with a force sensor and a Soft Scoop Gripper, was utilized to perform these demonstrations. (see Fig 4.3) During each demonstration, the following sequence of motions was performed:

1. **Approach**: The human operator moved the robot's arm towards the stack of objects. The arm was positioned so that the Soft Scoop Gripper was aligned with the topmost object on the stack.

2. **Positioning**: The operator carefully positioned the gripper under the object

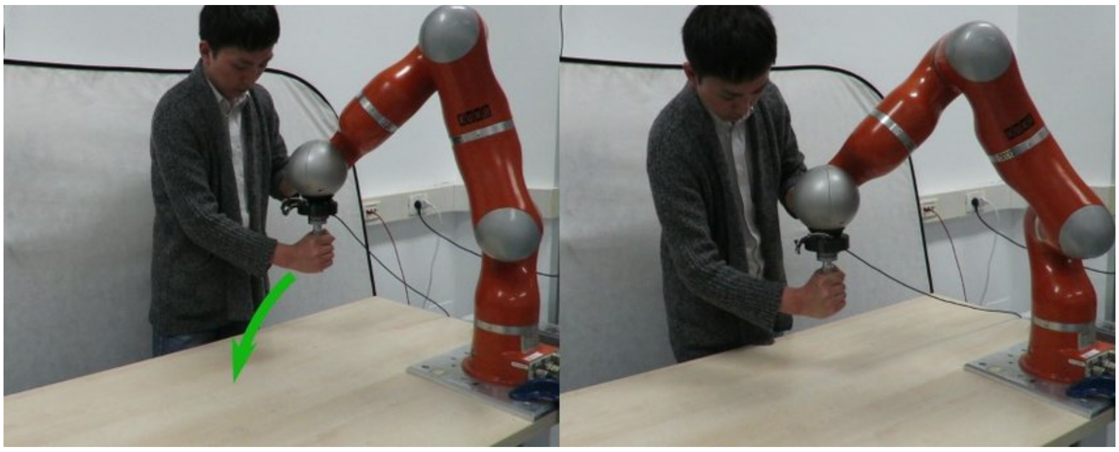

Figure 4.3: Example of Kinesthetic Teaching from [10]

intended for picking. This involved moving the arm downwards to slide the scoop underneath the object, ensuring that the scoop was correctly oriented to support the object from below.

3. **Sliding Motion**: The operator guided the robot to execute a sliding motion. This involved moving the scoop horizontally while maintaining contact with the bottom of the object. The purpose of this motion was to exploit the environmental constraints provided by the stack and facilitate the picking action.

4. **Gripping**: Once the object was securely supported by the scoop, the operator closed the fingers of the Soft Scoop Gripper. This action secured the object, making it ready for lifting.

5. **Lifting**: The operator then moved the robot's arm upwards, lifting the object from the stack. This final motion completed the demonstration, and the robot held the object above the stack

6. **Stopping** the Demonstration: After lifting the object, the demonstration was stopped. The data collection system recorded the entire sequence of motions, capturing the necessary information for each phase of the task.

A total of 15 demonstrations were performed, with three demonstrations conducted for each of the five objects (a book, a clamp, a calculator, a plate, and a wooden block). By utilizing kinesthetic teaching, the demonstrations captured the intricate details of the task execution, providing a rich dataset for the Learning from Demonstration (LfD) approach.

4.2 Data Collection and Preprocessing

During these demonstrations, data is collected at a different frequency for end effector pose and forces from the sensor. The collected data was stored in bag files, a format commonly used in robotic systems for recording time-stamped data streams. These files were subsequently processed to make the data accessible in MATLAB for further analysis and modeling.

4.2.1 Data Acquisition

The data collected can be divided in:

- Pose Data: The position and orientation of the end effector (the Soft Scoop Gripper) were recorded throughout the demonstration. This included both Cartesian coordinates and quaternions, later converted to Euler angles for better interpretability.
- Force Data: Forces and torques experienced by the gripper were recorded using the attached force sensor. This data was crucial for understanding the interaction between the gripper and the objects, especially during the sliding and lifting motions.
- Joints data: the joint angles during the trajectory is collected, but not used in the following process.

As said before, pose data and force sensor data are sampled at difference frequencies, in order to put them on a common frequency, a synchronization function is applied before saving files in Matlab. After applying the synchronization we obtain a file .BAG for each of the three types of data, uniting positions and quaternions, representing orientation, in a single file. In this files data are structured the following way:

- **Pose** is composed of 8 values, the first being the time instant, the following are positions in x,y,z and the quaternion.
- **Force** is composed of 7 values, the first being the time instant, and the following are forces along x,y,z and torques around x,y,z.

4.2.2 Data Transformation

The data collected is referenced to different reference frames. The pose is referenced to the world reference frame, but it is the one of the old end effector of the robot. The forces and torques are referenced to the sensor's reference frame. (see Fig. 4.2) In order to put all the data to the same reference frame, the world in this case, we need to apply some transformation to our data. The decision is to bring the poses and the forces to the scoop of the gripper, in the world reference frame, in this first processing part. The transformation applied to the poses is shown here using Rotation matrices, the same thing could be done using quaternions for the rotation part avoiding transformation from quaternions to rotation matrix and vice versa. In this case the transformation matrix, composed by 4*4 elements, is made by the rotation 3*3 matrix from the end effector to the scoop, the translation along x,y,z of the world and the last row is [0,0,0,1].

$$\mathbf{Transformation} = \begin{bmatrix} Rep_{11} & Rep_{12} & Rep_{13} & xt \\ Rep_{21} & Rep_{22} & Rep_{23} & yt \\ Rep_{31} & Rep_{32} & Rep_{33} & zt \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

The transformation matrix is applied to the pose of the end effector matrix, structured in the same way. The product is show below, the result is the transformed matrix pose in the scoop gripper, then the rotation matrix is brought back to quaternions and stored in each row composing the 7 values of the pose

$$\mathbf{P'} = P * Transformation \tag{4.2}$$

In the given equation '*' represents matrix product. The forces and torques data were referenced to the sensor's reference frame. They are transformed to the scoop gripper to simulate the feeling of the forces and torques on the real end effector.

$$\mathbf{RotationEP} = \begin{bmatrix} Rep_{11} & Rep_{12} & Rep_{13} \\ Rep_{21} & Rep_{22} & Rep_{23} \\ Rep_{31} & Rep_{32} & Rep_{33} \end{bmatrix} \tag{4.3}$$

In this case we only need the rotation matrix coming from quaternion and applying it to forces and torques in the following way:

$$\mathbf{f'} = RotationEP * f \tag{4.4}$$

$$\mathbf{t'} = RotationEP * t \tag{4.5}$$

4.2.3 Data Cleaning and Filtering

After acquiring the data from the 15 demonstrations and transforming them, the first step was to load the data in a structure of 15 cells and visualize it to understand its behavior. The dataset consisted of position and force data recorded during each demonstration. Below, I describe the detailed process of data cleaning and filtering.

- Initial Data Visualization: after loading the data, the positions and forces were plotted to inspect their behavior. This initial visualization revealed significant noise in the force data, as shown in Fig. 4.4, which necessitated further processing to ensure the quality of the dataset.

- Noise Reduction with Low Pass Filter: given the noisy nature of the force data, a low pass filter was applied to smooth the signals. The low pass filter was chosen to retain the essential features of the force data while eliminating high-frequency noise. The filtering process was performed using the following parameters: cutoff frequency of 50 Hz, determined based on the frequency content of the force data, filter type Butterworth of order 4 was used for its smooth frequency response.

  The effect of the low pass filter is illustrated in Fig.4.4, where both the original and filtered force data are shown.

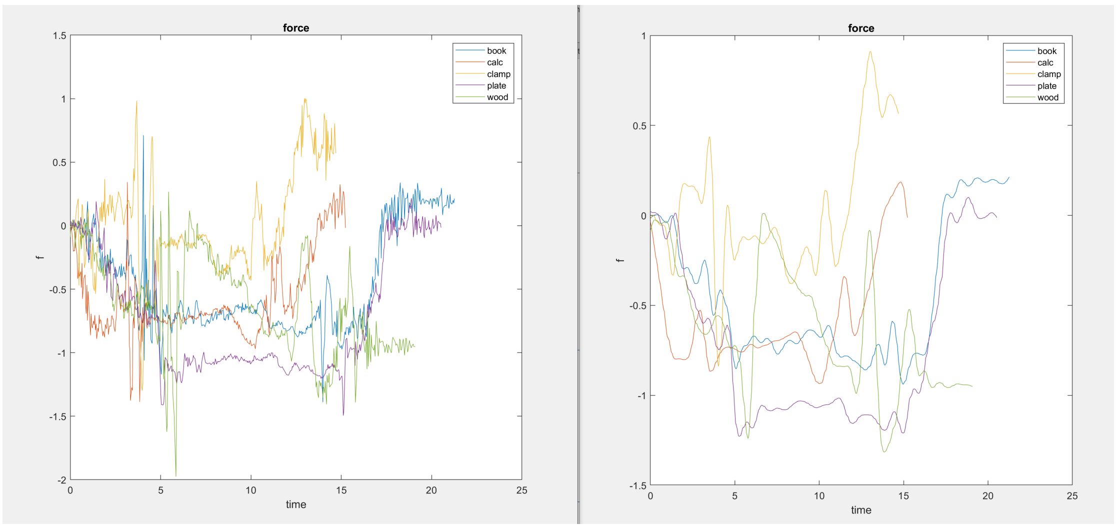

Figure 4.4: Comparison of force data before and after applying the low pass filter. Force along x

- Outlier Detection and Handling:

  to ensure the dataset's integrity, outlier detection was performed using box plots with five-number statistics (minimum, first quartile, median, third quartile, maximum), plotting them for each feature and obtaining a composite score for each demonstration. This analysis helped identify anomalies in the data, such as unusually high or low values that could skew the results. Box plots were used to visualize the distribution of the force and position data, highlighting any potential outliers. (see Fig.4.5 During this analysis, one

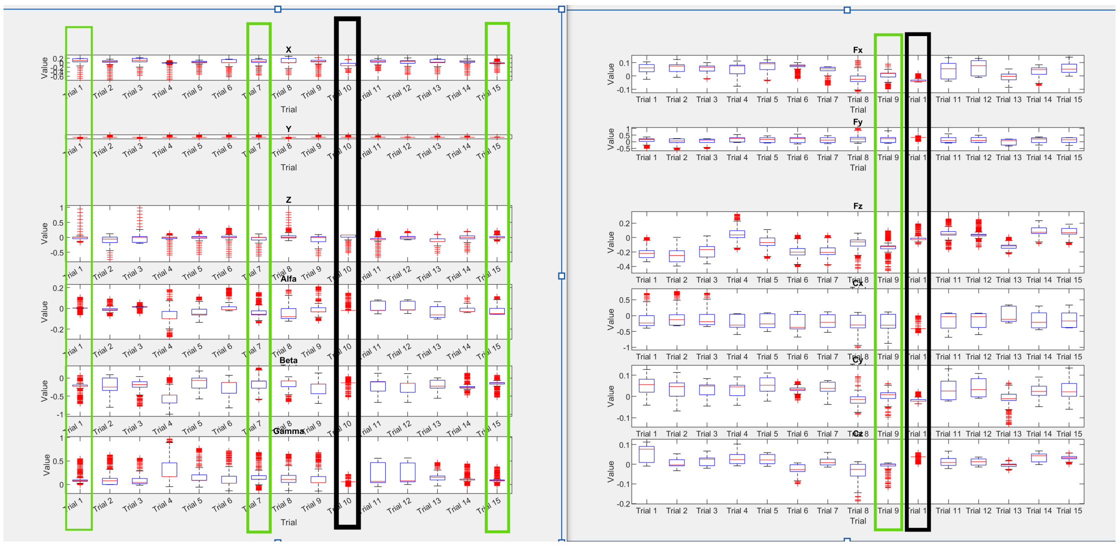

Figure 4.5: Box plots for each feature

demonstration (the 10th demonstration) was found to be excessively noisy, maybe some mistake was done during the teaching phase, and was removed from the dataset. Additionally, three other demonstrations were identified as outliers, analysing box plots for each feature and a global composite score for each demonstration as shown in Fig.4.6. Evaluating box plots for each feature

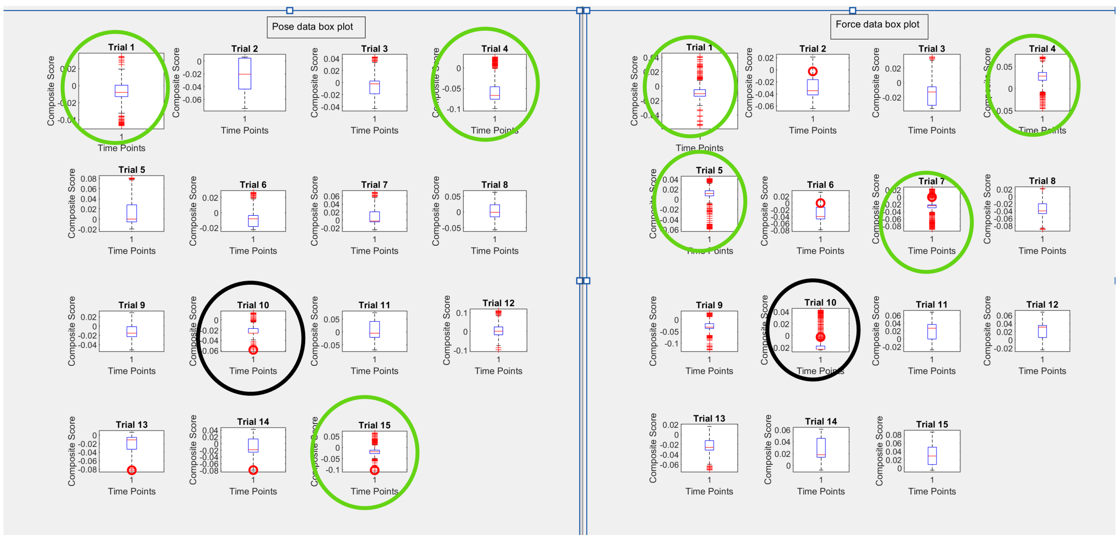

Figure 4.6: Box plots for each demonstration

and the composite for each demonstration the decision was to consider outliers the following demonstrations: 1, the first on the Book object, 4, the first on the Calculator object, 7, the first on the Clamp object and 15 the last on the Wooden Block object. This was predictable, since the first demonstration performed on each object was the one where the teacher was inexperienced. To balance the dataset, two of these outliers were included in the training set, while the third was placed in the test set.

- Data Splitting: Training and Test Sets:

As anticipated before, the cleaned dataset is split into a training set and a test set. The training set comprised 11 demonstrations, while the test set contained the remaining 3 demonstrations. This division ensured that the model could be trained effectively and then tested on a separate subset of the data to evaluate its performance and adjust model's parameters. The choice of where to put outliers was randomic, but proportionate to the size of the two sets, which was decided to be around 70/30 between training set and test set The result was the following:

- Training Set composed by 11 demonstrations (including 3 outliers, demonstration 1,7 and 15)
- Test Set composed by 3 demonstrations (including 1 outlier, demonstration 4)

This approach of splitting the data allowed for robust training and testing of the model, ensuring that it could generalize well to new, unseen data.

In summary, the data cleaning and filtering process involved loading and visualizing the data, applying a low pass filter to reduce noise, detecting and handling outliers, and splitting the dataset into training and test sets. These steps were crucial in preparing a high-quality dataset for the subsequent modeling and analysis.

4.2.4 Data Alignment and Normalization

- **Quaternion to Euler Angle Conversion**: In robotic applications, quaternions are often used to represent orientations due to their robustness against gimbal lock and instability. However, for interpretability and visualization, converting quaternions to Euler angles is advantageous. Euler angles provide a more intuitive understanding of rotational movements around the axes. Quaternions consist of four components: *qw, qx, qy, and qz* representing the imaginary part and the three real components of the axes around which the rotation happens. The conversion from quaternions to Euler angles (*roll, pitch, and yaw*) involves several mathematical operations. In order to perform the transformation in matlab the built in function *quat2eul* and its inverse *eul2quat* are used. Euler angles can be represented both as radians and degrees without significant changes in interpretation. In general Euler angles can suffer from instability in training and in particular movements, but are widely used, as is shown in [9].

- **Relativization Respect to Contact Time for poses**: To ensure that training is done relatively to the contact time, in order to improve generalization of the model, demonstrations are relativized based on the first peak in the torque around x (Tx).Tx is the most representative component of the contact moment during the trajectory. The process involves identifying Contact time through the matlab function *findpeaks*, saving the indexes and then subtracting the relative value to each demonstration for positions, while applying the inverse quaternion for orientation.(see Fig.4.7) This way the results is that we have a new relative reference frame corresponding to the one of first contact with the stack.

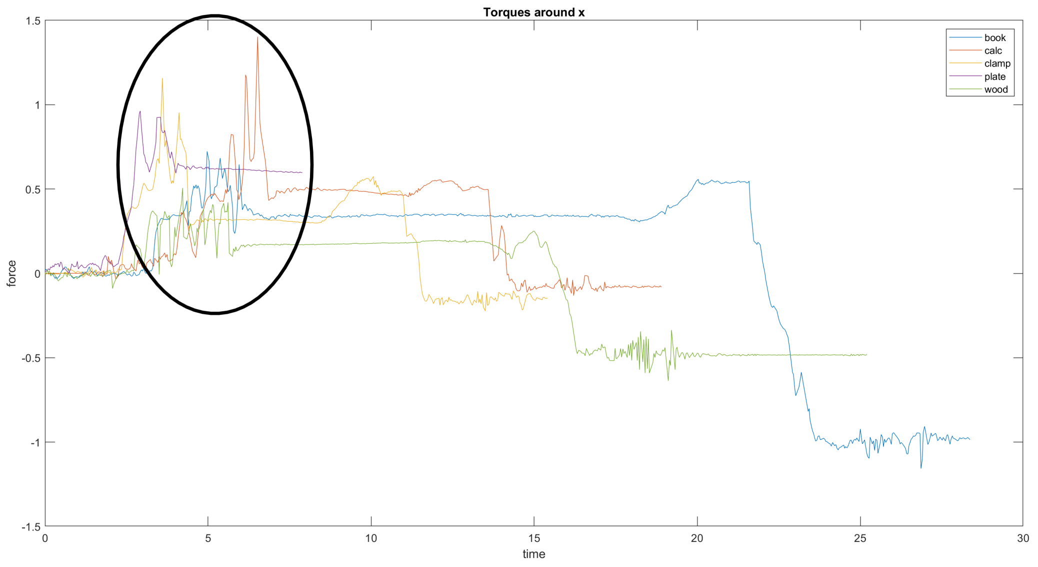

Figure 4.7: First peak Tx

- **Normalization**:

  Normalization is a crucial preprocessing step to ensure that different features contribute equally to the learning process. The normalization technique choice could be different depending on what the main requirements are. In this case the priorities are to maintain signs of poses and forces, to keep the range between demonstrations, to have a better generalization differentiating objects and to retain the different variances in poses and forces features. To satisfy these requirements, the decision was the following:

- **Force sensor data**: a Max-Abs scaling technique was used,to scale values in [-1,1] range, considering all the components of forces and torques and all the demonstrations data.

$$\mathbf{F}_{\text{scaled}} = \frac{\mathbf{F}}{\max(|\mathbf{F}|)} \tag{4.6}$$

- **Pose data**: a Max-abs scaling in [-1,1] range was used, separately for positions and euler angles in order to maintain reasonable values and considering all components and demonstrations.

$$\mathbf{X}_{\text{scaled}} = \frac{\mathbf{X}}{\max(|\mathbf{X}|)} \tag{4.7}$$

- **Dynamic Time Warping (DTW)**: The Dynamic Time Warping (DTW) algorithm is highly regarded for its efficiency as a time-series similarity measure. It mitigates the impacts of temporal shifting and distortion by enabling "elastic" alignment of time series, as described in [11]. It is used as a metric to understand the distortion between signals, but can also be applied with some modifications in order to align time series signals. In robotics field, especially in Lfd, it is used to realign similar signals of trajectories and create a common time vector. This is what is used in [12], before training the GMM/HMM model and in [13] to align demonstrations. In the studied case DTW is used to align demonstrations, with different lengths and time vectors, and create a common time vector based on a reference trajectory. The goal is to obtain demonstrations that have the main events around the same time, and obtaining the same length for all trials. A similar approach has been used in [12] where the reference trajectory is chosen based on the highest log-likelihood: the log likelihood $\mathcal{L}$ for a set of observations $X = \{x_1, x_2, \ldots, x_T\}$ given a model with parameters θ is calculated as:

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \log P(x_t|\theta)$$

  where $P(x_t|\theta)$ is the probability of observing x_t given the model parameters θ.

In our case the reference trajectory was chosen based on the experience of the teacher viewing the plotted data understanding the most representative demonstration, which was acknowledged to be the 8 of the training set. Its length is of 440 elements and its time vector is taken as the common one. An example of DTW algorithm is presented below:

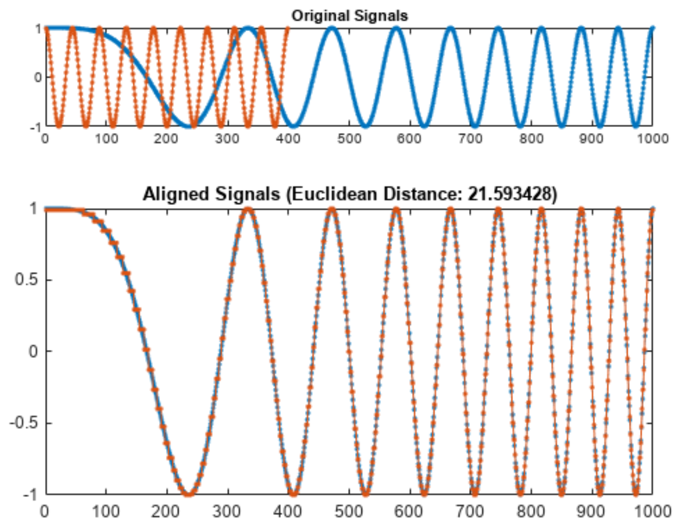

Figure 4.8: Example of DTW alignment between two time series. The optimal path minimizes the total alignment cost.

1. **Initialization:** Given two time series $Q = \{q_1, q_2, \ldots, q_n\}$ and $C = \{c_1, c_2, \ldots, c_m\}$, create an $n \times m$ cost matrix D where $D(i, j)$ represents the distance between q_i and c_j.

2. **Cost Calculation:** Compute the cumulative cost $D(i, j)$ using the following recursive formula:

$$D(i, j) = d(q_i, c_j) + \min\{D(i-1, j), D(i, j-1), D(i-1, j-1)\}$$

   where $d(q_i, c_j)$ is the distance between the points q_i and c_j.

3. **Boundary Conditions:** Initialize the first row and first column of the cost matrix:

$$D(1, 1) = d(q_1, c_1)$$

$$D(i, 1) = d(q_i, c_1) + D(i-1, 1) \quad \text{for } i = 2, 3, \ldots, n$$

$$D(1, j) = d(q_1, c_j) + D(1, j-1) \quad \text{for } j = 2, 3, \ldots, m$$

4. **Optimal Path:** The optimal warping path is found by tracing back from $D(n, m)$ to $D(1, 1)$, following the minimum cost at each step:

$$\text{path} = \{(n, m)\} \cup \{(i, j) \mid (i, j) = \arg\min\{D(i-1, j), D(i, j-1), D(i-1, j-1)\}\}$$

5. **Output:** The total cost of the optimal alignment is given by $D(n, m)$. The aligned sequences can be derived by following the optimal path.

In Fig.4.8 an example of the results of DTW alignment on two time series is presented.

In our case the DTW is applied using both poses and forces data, considering components along y,z for forces, torques around x and the whole pose.The DTW produces warping paths that align key events, but doesn't necessarily fit all trajectories on a common length. For this reason, along with the alignment of the demonstrations based on the reference, a cubic interpolation is implemented to get all trials on the same time vector of 440 elements. A comparison between signals before DTW and interpolation and after is shown below. (see Fig.4.9)

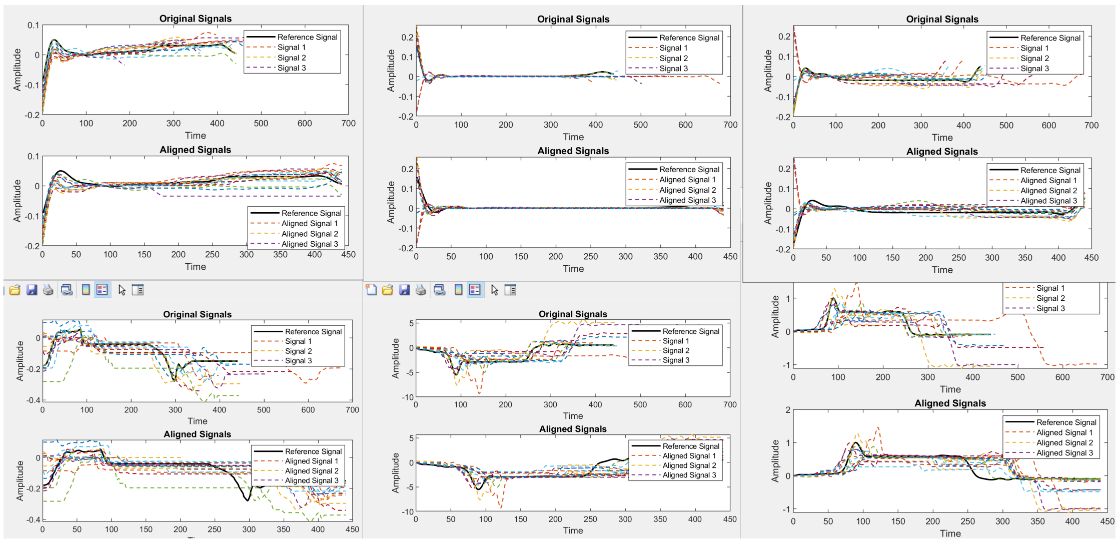

Figure 4.9: Signals before and after DTW and interpolation

As we can see, all the demonstrations are now equally long and aligned to the reference one, represented by the black signal.(see Fig.4.9)

4.2.5 Data Structure

After all the preprocessing phase the data is structured in a convenient way to be used in the following training. Since forces data and poses are used separately, the two datasets are split. Moreover the dataset is divided in Training Set and Test Set following the previously described outlier management, obtaining this way four matlab (.mat) files containing 3D matrices: Training Set Poses, Training Set Forces, Test Set Poses, Test Set Forces and the common time vector. The dimensions are presented below in Table 4.1.

	Samples	**Features**	**Demonstrations**
Training Set Poses	440	6	11
Training Set Forces	440	6	11
Test Set Poses	440	6	3
Test Set Forces	440	6	3

Table 4.1: Dataset Information for Training and Test Sets

4.3 Probabilistic Modeling

Probabilistic modeling describes a group of algorithms and techniques based on probability and statistics, used in a lot of different fields. As explained in [14]probabilistic robotics is a new and growing area in robotics, concerned with perception and control in the face of uncertainty. Reality is not deterministic in the majority of cases, this means that if we want to model real world problems it is a good idea to treat them in a stochastic way, using probability. In [15] **Thrun** says:"*The ultimate goal of robotics is to build robots that do the right thing. I conjecture that a robot that takes its own uncertainty into account when selecting actions will be superior to one that does not*". So this is why the choice in this study is to apply probabilistic modeling to our learning from demonstration approach. Analysing literature a lot of research has been done in the field of Lfd, trying new approaches like Deep Learning techniques, but still in order to guarantee high robustness to environmental changes, adaptation to real world and a good interpretability, models like **GMM/GMR**, **DMP** (Dinamic Movement Primitives) and **HMM** are still the gold standard as shown in [16] which is pretty recent study. Moreover Deep Learning techniques such as Convolutional Neural Networks (CNN) and **Long Short Term Memory (LSTM)** are starting to be used a lot, but in situations where robotics vision is implemented, which is not our case.(See[17] and [18]) In order to implement probabilistic modeling and exploit environmental constraint, the decision was to use two different models, one for an high level control and one for a low level control. A similar approach has been used in [19],in [12] and in [20].(see Fig.4.10) Specifically the idea is to implement an HMM to divide trajectories in 3 primitives (approach,sliding,lifting up) based on the force sensor data, exploiting environmental constraints. After implementing the HMM, three different GMMs are trained, one for each primitive, modeling poses based on time.The three GMMs models are concatenated to smooth transitions between them following the approach used by Calinon in [21]. In the end the trajectory is predicted through GMR giving time as input, switching primitives based on the forces felt by the scoop and controlled by HMM. A detailed description of the process and models is given below.

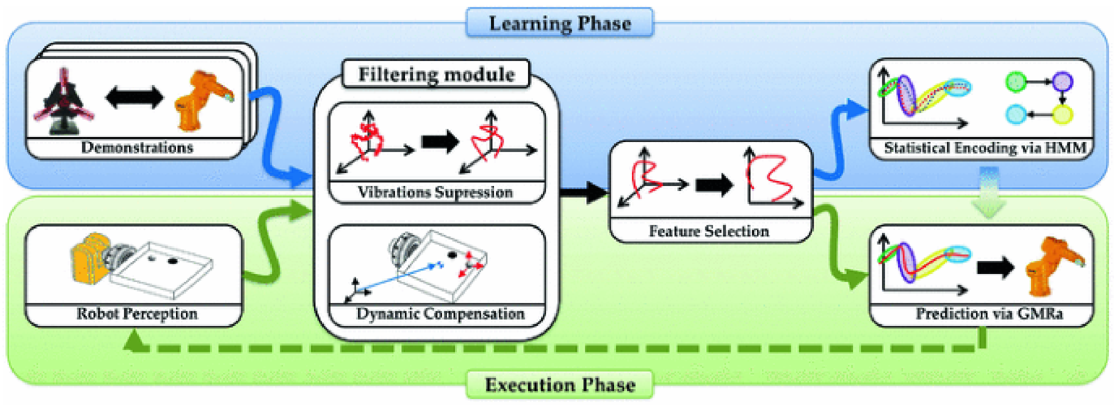

Figure 4.10: Example of Probabilistic modeling in Lfd from [19]

4.3.1 High-Level Control with HMM

Primitive Division

Before training models, trajectories are divided in three main primitives based on forces and torques data. Since the movement can be symbolically modeled as a sequence of primitive movements, the used approach is similar to the one implemented in [22] and [23]. In our case based on experience and plots of data, the primitives chosen are three:

1. **approach** It represents the first part of the trajectory where the robots approaches the stack of objects and forces are near zero in this phase
2. **contact and sliding** this is the crucial phase when the scoop gets in contact with the stack and starts feeling forces and torques from the environment. The sliding phase under the object happens after contact and should be repeated if the scoop doesn't fit the gap between the objects which is a pretty challenging task to perform.
3. **lifting up** This is the final part where the robot picks up the object, closes the grip and lifts up the object

Symbolically this sequence of primitives can be described as a graph, and the transitions can be modeled with an HMM. In [23] this transitions were modeled with a GMM or Support Vector Machines(SVM). Our approach is using an HMM as a classifier to predict primitives based on forces exploiting environmental constraints. This means that from a symbolic modeling we have to translate the transitions into a technical and measurable form. The reason to use forces to classify the primitive is that our sensor is really sensitive, robust and interpretable in showing transitions between states. As can be understood by the Fig. 4.11,representing force along y and torque around x, the end of the first primitive can be approximated as the first high peak in force or negative peak in torque and it is reasonable because when the scoop touches the stack there is a sudden impulse of vertical force and torque around x.(see Fig.4.11)

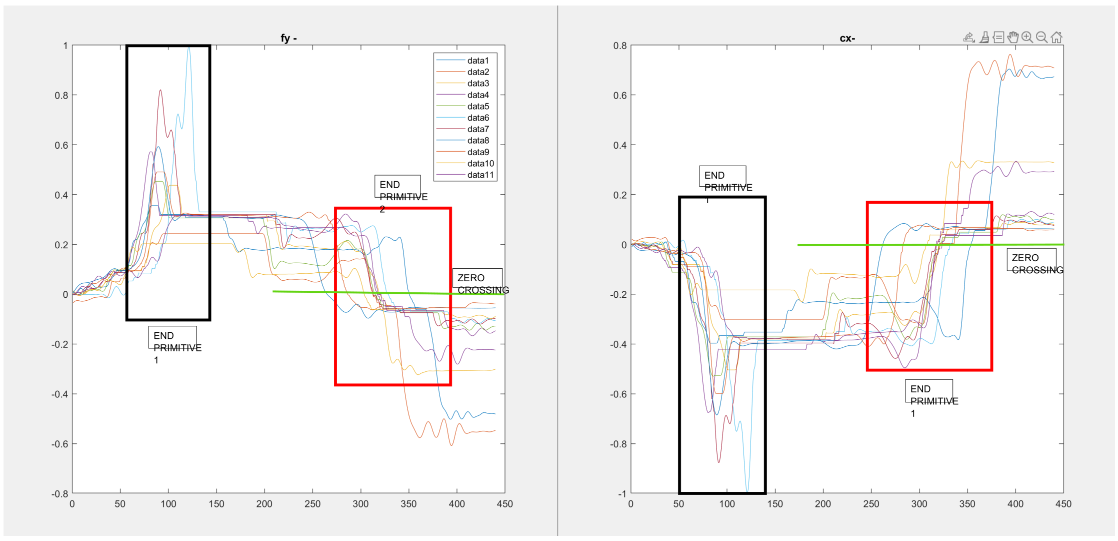

Figure 4.11: Force y and Torque x

For this reason the approach is to find the indices of that peak and label the elements before that as **primitive 1**. Since poses have been already relativized to the contact moment we can also take the element with pose's components equal to zero.

In order to find the moment of detachment from the stack and identify the end of primitive 2 a zero crossing technique is used, following the approach presented in [24]. The component used to find the zero crossing can be either force along y or torque around x, which represent almost the same events. In this case the zero crossing from negative to positive values of torque around x is taken as reference of the end of primitive 2. From the next element to the end of each demonstration is considered primitive 3. (see Fig.4.11)

At the end of this process we have a matrix of integer 1,2 or 3 for each demonstration representing the three different primitives, which then will become the HMM states. The length of each primitive is different for each demonstration so this has to be addressed in the following steps.

HMM Training

A Hidden Markov Model (HMM) is a statistical framework used to describe a system that progresses through a series of hidden states over time. It is commonly applied in sequence analysis, where the system is presumed to follow a Markov process with states that are not directly observable. As explained in [25], considering a sequence of state variables *q1, q2, ..., qi*, a Markov model embodies the Markov assumption on the probabilities of this sequence: that when predicting the future, the past doesn't matter, only the present. Mathematically speaking, the Markov Assumption can be written this way:

$$P(q_i = a \mid q_1 \dots q_{i-1}) = P(q_i = a \mid q_{i-1})$$

So each following state is only based on the present state and not on the history of the whole process, which can be applied to model different fields.

An HMM is characterized by the following components:

- **States** (S): The set of hidden states the model can be in. These states are not directly observable.
- **Observations** (O): The set of observable symbols that can be emitted from the states.
- **Transition Probabilities** (T): The probabilities of transitioning from one state to another.
- **Emission Probabilities** (E): The probabilities of observing a particular symbol from a state.
- **Initial State Probabilities** (pi): The probabilities of starting in each state.

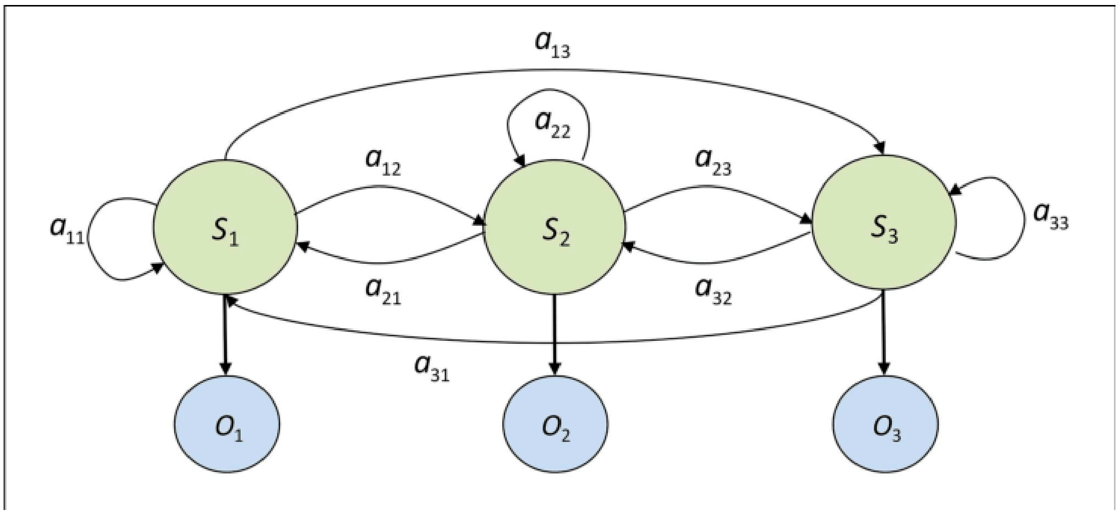

Figure 4.12: 3 states HMM example

In Fig. 4.12 an example of a 3 states HMM graph is shown. It is really similar to the model used in this study, but the observation in our case are not discrete, they are continuous, so they have to be modeled with a Probability Density Function(PDF), such as Gaussian distributions. A continuous Hidden Markov Model is an HMM where there is no discrete observation emitted, but continuous values. This kind of model is used in [26] and in [27] where if the observations are modeled as Gaussian distributions, given that data fit this distribution, is called Gaussian HMM. An example of the model is given below in Fig. 4.13. In our case the states are the

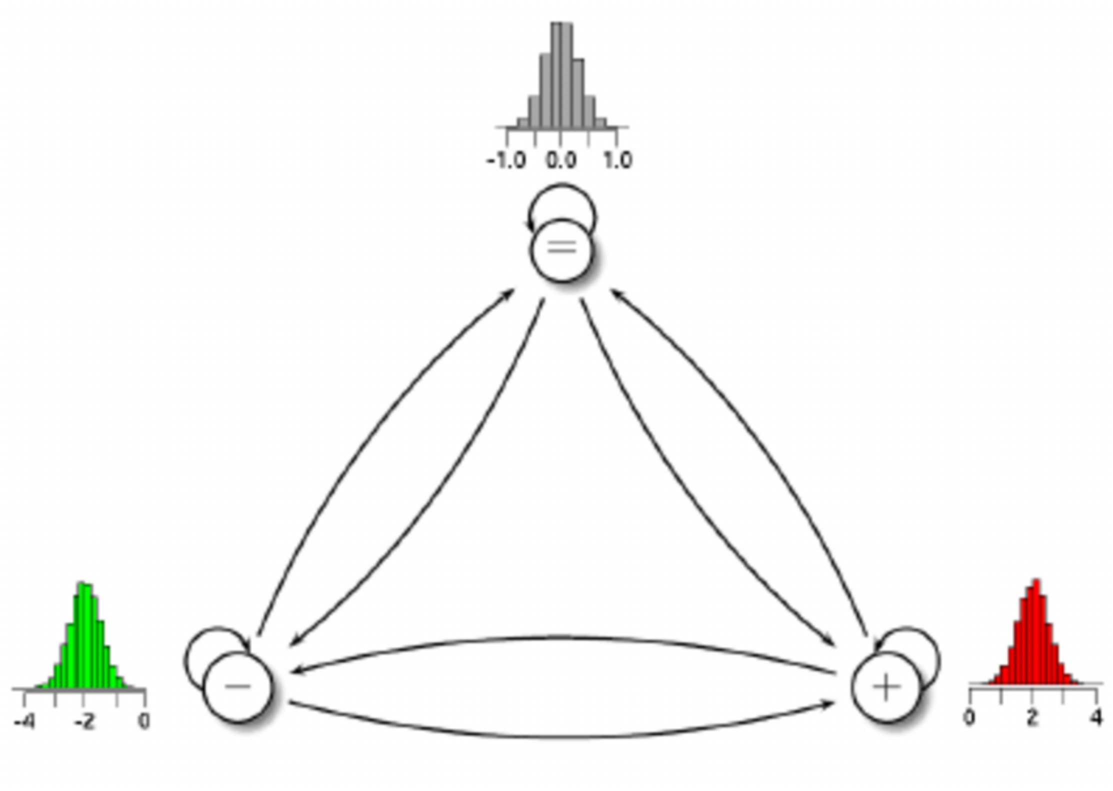

Figure 4.13: 3 states HMM continuous Gaussian example

primitives, while observations are forces, the emission probabilities represent the probability of that primitive to produce that value of force. When a Gaussian HMM has to be modeled the parameters to be set are the number of Gaussians used to code data,priors probabilities,transition probability matrix, emission probability matrix, means and covariances of gaussians. There are different way to set these parameters as can be seen in [28]:

- **random**
- **uniform**

- **left-right**

The initialization of matrices affects relatively the model since with Expectation Maximization algorithm it is possible that convergence happens to the same values. On the other hand the number of Gaussians and the stopping criteria can change the results in a significant way. This is why the model was trained with different combinations of this parameters and tested on both training set and test set to evaluate performance and fine tune them. After trying different values and testing them the decision was to use the model with 3 Gaussians per state. The table with the performances on the different parameters will be shown in the next chapter. In order to get the HMM model the approach is to train an HMM model with the Baum-Welch Algorithm, which is an iterative Expectation Maximization algorithm explained below,for each demonstration and finally choosing the one with the highest log-likelihood. (See equation 4.8)

Algorithm 1 Baum-Welch Algorithm

```
1: Initialize the HMM parameters: transition probabilities $A$, emission probabilities $B$, and initial state distribution $\pi$.
2: repeat
3:     E-step:
4:     Compute the forward probabilities:
           $\alpha_t(i) = P(O_{1:t}, q_t = S_i|\lambda)$
5:     Compute the backward probabilities:
           $\beta_t(i) = P(O_{t+1:T}|q_t = S_i, \lambda)$
6:     M-step:
7:     Re-estimate the transition probabilities:
           $\hat{A}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$
8:     Re-estimate the emission probabilities:
           $\hat{B}_j(k) = \frac{\sum_{t=1}^{T} \gamma_t(j) \cdot I(O_t = V_k)}{\sum_{t=1}^{T} \gamma_t(j)}$
9:     Re-estimate the initial state distribution:
           $\hat{\pi}_i = \gamma_1(i)$
10: until Convergence
```

Here's the formula for the log-likelihood used in training the HMM:

$$\log P(O|\lambda) = \sum_{t=1}^{T} \log\left(\sum_{i=1}^{N} \alpha_t(i)\right) \tag{4.8}$$

In order:

1. **Loading Data**: The data, including force and pose data, is loaded.
2. **Standardization**: Force data is standardized to have zero mean and unit variance to fit Gaussian distribution.
3. **Dimension Reduction**: The number of force components used can be adjusted (1, 2, or 3). Data is reduced accordingly.

4. **HMM Initialization**: Initial parameters for the HMM are set, including transition probabilities, prior probabilities, means, covariances, and mixture matrix.

5. **HMM Training**: The continuous HMM is trained using the Baum-Welch algorithm with regularization.

6. **Model Selection**: The best HMM model is selected based on the highest log-likelihood.

7. **Evaluation**: The model is evaluated on both training and test sets. Accuracy is calculated based on the predicted states compared to actual states.

8. **Prediction**: The Viterbi algorithm is used to predict the sequence of states for given observations.

Force and torque data are crucial for identifying the transitions between primitives. These physical interactions with the environment provided key indicators of state changes, essential for accurate segmentation and modeling of the movement. More specifically two components of forces and torques are used: force along y and torque around x.

In our case the choice was between training the model in a supervised way using the labels of primitives we calculated before, improving accuracy but losing in generalization and robustness to perturbations or training with the unsupervised Baum Welch algorithm losing some accuracy but gaining robustness. In our case the choice is to train with the Baum Welch algorithm applying boundary and initial conditions based on the known primitives. In order to classify the primitive the Viterbi Algorithm is used as shown in [28]. The Viterbi Algorithm is used to estimate best state sequence from observations. It is widely used in various fields such as speech recognition, bioinformatics, and natural language processing. As said in [29]:"The algorithm is based on the calculation of a distance measure between the received signal at time ti and all paths in the trellis diagram that arrive in every possible state at that time ti".

The algorithm is given below.

Viterbi Algorithm

Initialization:

$$\delta_1(i) = \pi_i b_i(O_1) \quad \forall\, 1 \leq i \leq N \tag{4.9}$$

Recursion:

$$\delta_t(j) = \max_i \left[\delta_{t-1}(i) a_{ij}\right] b_j(O_t) \quad \forall\, 2 \leq t \leq T,\ 1 \leq j \leq N \tag{4.10}$$

$$\psi_t(j) = \arg\max_i \left[\delta_{t-1}(i) a_{ij}\right] \quad \forall\, 2 \leq t \leq T,\ 1 \leq j \leq N \tag{4.11}$$

Termination:

$$P^* = \max_i \delta_T(i) \tag{4.12}$$

$$q_T^* = \arg\max_i \delta_T(i) \tag{4.13}$$

Path Backtracking:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \ldots, 1 \tag{4.14}$$

This first part of probabilistic modeling was evaluated on training and test set data before training the GMM.

4.3.2 Low-Level Control with GMM/GMR

The low-level control of the robotic system is implemented using Gaussian Mixture Models (GMM) and Gaussian Mixture Regression (GMR). This approach allows for the generalization of the end-effector's pose based on the temporal progression of the task, as described in different articles such as [30],[9],[12] and a very recent study [31]. Below is a detailed explanation of the methodology.

Gaussian Mixture Models

Gaussian Mixture Models (GMM) are a powerful probabilistic tool used in various fields such as statistics, machine learning, and robotics. A **GMM** is a parametric probability density function represented as a weighted sum of Gaussian component densities, as explained in [32]. This model is particularly useful for representing data that is assumed to be generated from a mixture of several Gaussian distributions with unknown parameters.A **GMM** is defined as follows:

$$p(\mathbf{x}|\lambda) = \sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}|\mu_i, \Sigma_i) \tag{4.15}$$

where:

- x is a D-dimensional continuous-valued data vector (i.e., measurement or features).
- K is the number of Gaussian components.
- Pi is the mixing coefficient for the i-th Gaussian component, with

$$\sum_{i=1}^{K} \pi_i = 1 \quad \text{and} \quad 0 \leq \pi_i \leq 1 \tag{4.16}$$

.

- $$N(x \mid \mu_i, \Sigma_i) \tag{4.17}$$
  is the Gaussian density function with mean vectorμ_i and covariance matrix Σ_i.

Expectation-Maximization (EM) Algorithm

The parameters of a **GMM**, $\lambda = \{\pi_i, \mu_i, \Sigma_i\}$, are typically estimated using the Expectation-Maximization (EM) algorithm. The EM algorithm is an iterative method to find maximum likelihood estimates of parameters in probabilistic models, where the model depends on unobserved latent variables.[33]

1. **Initialization**: Initialize the parameters $\{\pi_i, \mu_i, \Sigma_i\}$.
2. **Expectation Step (E-step)**: Calculate the responsibility *yik* which is the probability that the *k-th* component generated *xi*.
   $$\gamma_{ik} = \frac{\pi_k N(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_i \mid \mu_j, \Sigma_j)} \tag{4.18}$$
3. **Maximization Step (M-step):** Update the parameters $\{\pi_i, \mu_i, \Sigma_i\}$.
   $$\pi_k = \frac{1}{N}\sum_{i=1}^{N} \gamma_{ik} \tag{4.19}$$
   $$\mu_k = \frac{\sum_{i=1}^{N} \gamma_{ik} x_i}{\sum_{i=1}^{N} \gamma_{ik}} \tag{4.20}$$
   $$\Sigma_k = \frac{\sum_{i=1}^{N} \gamma_{ik}(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{N} \gamma_{ik}} \tag{4.21}$$

These steps are repeated until the parameters converge.

Applications of GMM

GMMs are widely used in:

- **Clustering**: Identifying subgroups within a dataset.
- **Density Estimation**: Estimating the probability density function of the dataset.
- **Pattern Recognition**: Identifying patterns and classifying data points.

GMMs have proven effective in applications such as speech recognition [34], image processing [35], and biometrics [36].

Gaussian Mixture Regression (GMR)

Gaussian Mixture Regression (GMR) is an extension of GMM used for regression tasks. GMR leverages the probabilistic framework of GMM to perform regression, enabling it to model complex, nonlinear relationships between variables.

Mathematical Formulation

Given a GMM trained on the joint distribution of input and output variables, GMR estimates the conditional distribution of the output variables given the input variables.

The conditional mean and covariance are given by:

$$\mu_{y|x} = \sum_{k=1}^{K} h_k(x) \left(\mu_{k,y} + \Sigma_{k,yx}(\Sigma_{k,xx})^{-1}(x - \mu_{k,x})\right) \tag{4.22}$$

$$\Sigma_{y|x} = \sum_{k=1}^{K} h_k(x) \left(\Sigma_{k,yy} - \Sigma_{k,yx}(\Sigma_{k,xx})^{-1}\Sigma_{k,xy}\right) \tag{4.23}$$

$$h_k(x) = \frac{\pi_k N(x|\mu_{k,x}, \Sigma_{k,xx})}{\sum_{j=1}^{K} \pi_j N(x|\mu_{j,x}, \Sigma_{j,xx})} \tag{4.24}$$

where:

- $\mu_{k,x}$, $\mu_{k,y}$ are the means of the k-th Gaussian component for the x and y variables respectively.
- $\Sigma_{k,xx}$ is the covariance matrix of the x variables for the k-th Gaussian component.
- $\Sigma_{k,yy}$ is the covariance matrix of the y variables for the k-th Gaussian component.
- $\Sigma_{k,xy}$ is the cross-covariance matrix between the x and y variables for the k-th Gaussian component.
- $\Sigma_{k,yx}$ is the transpose of $\Sigma_{k,xy}$.

Applications of GMR

GMR is particularly useful in robotics for tasks such as:

- **Trajectory Learning and Reproduction**: Learning and reproducing trajectories from demonstration [37] [38] [31]
- **Adaptive Control**: Adapting robot control strategies based on sensory feedback [39]

Training of GMM/GMR model

In this section, a Gaussian Mixture Model (GMM) is trained for each primitive of the task to model the pose of the end effector at a low level including time vector. This means that our approach can be classified as **Time Dependent**. The GMMs were trained using the Expectation-Maximization (EM) algorithm(Baum-Welch), and their performance was evaluated using the Bayesian Information Criterion (BIC). Based on the BIC, the optimal number of Gaussian components for each GMM was selected. After training the GMMs, we used Gaussian Mixture Regression (GMR) to generalize the trajectories through Viterbi algorithm, providing time as input to predict the end-effector pose. The GMM training process is based on Calinon's code with some modifications. [40]

1. **Initialization and Training**:
   - *Initialization*:The GMM parameters were initialized using K-means clustering as in [40]. For each primitive, we initialized the GMM with a different number of Gaussian components ranging from 3 to 15, to better evaluate the distributions of poses data
   - *Expectation-Maximization (EM) Algorithm*: The EM algorithm iteratively refines the GMM parameters by alternating between the Expectation (E) step and the Maximization (M) step until convergence.
2. **Evaluation and Selection**:
   - *Bayesian Information Criterion (BIC)*:BIC was used to evaluate the models and select the optimal number of Gaussian components. BIC balances model fit and complexity, penalizing the number of parameters to avoid overfitting.The BIC for a model is defined as:

     $$\text{BIC} = -2\log(\mathcal{L}) + p\log(N) \tag{4.25}$$

     where L is the likelihood of the model, p is the number of parameters in the model, N is the number of data points. The likelihood LL of the GMM is given by:

     $$\mathcal{L} = \prod_{i=1}^{N}\sum_{k=1}^{K}\pi_k\mathcal{N}(x_i \mid \mu_k, \Sigma_k). \tag{4.26}$$

     Where N is the number of data points.
     K is the number of Gaussian components.
     π_k is the weight of the k-th component.
     $N(x_i \mid \mu_k, \Sigma_k)$ is the Gaussian distribution with mean μ_k and covariance matrix Σ_k. During training a plot of the evolution of BIC is produced to

understand which number of Gaussian gives the lowest value, so it is the best in modeling data without overfitting. Here we give an example of the obtained plot in Fig. 4.14:

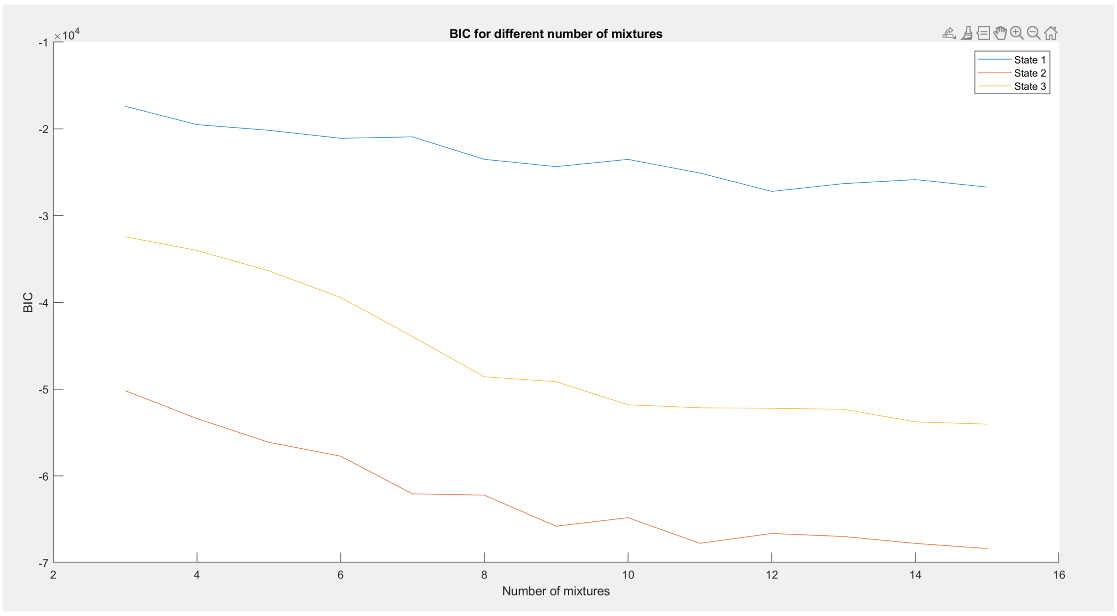

Figure 4.14: Example of BIC evolution

- *Expectation-Maximization (EM) Algorithm*:The EM algorithm iteratively updates the parameters of the GMM until convergence. The Baum-Welch algorithm is used with a convergence tolerance of 10^{-12} and different maximum iterations values. The 3 different GMMs are plotted on top of the position values to view them following Calinon approach [9]. The result of the models are shown here for the 10 number of Gaussian's case.(see Fig.4.15)

Gaussian Mixture Regression (GMR)

GMR uses the trained GMM to predict the end-effector pose given a time input. The algorithm used as in [40] is Viterbi algorithm explained before. The input we give to GMR is the time vector for each state, making all start from zero. Since each primitive has different lengths and the division is different for each demonstration, resampling to the length of the reference demonstration is applied. This means that each state is transformed to make all demonstrations long as the 8 demonstration: 89 elements for the first state, 172 for the second state, 179 for the third state. This is needed to train the models without errors in matlab functions. After regression the three predicted trajectory are analysed comparing them to the

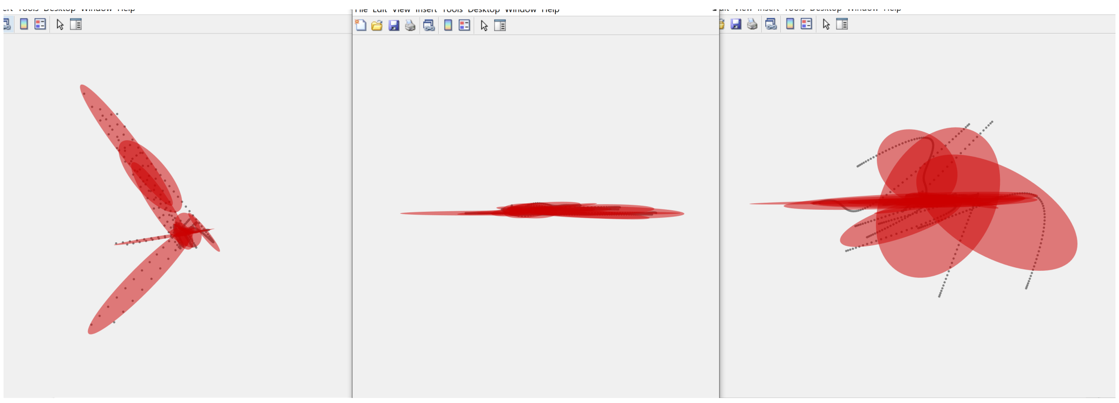

Figure 4.15: GMMs plot

actual trajectories for each state and the Mean Squared Error(MSE) is calculated to understand the performances. An example of positions obtained for each state is given below in Fig. 4.16. As can be seen the positions are predicted well giving an average behaviour compared to the actual trajectories for each state. The problem comes when we concatenate the states to obtain the whole trajectory since the concatenation creates discontinuities both in positions and Euler angles. Here the concatenated result for Euler angles is shown, since it is the orientation the one with higher discontinuity which would have caused difficult movements of the robot. In order to solve this issue the concatenation between GMMs is implemented following the Calinon's approach in [26] and [21]. This way a unique model with smooth transitions is generated managing to obtain smooth trajectories.

Combination of the GMMs models

This approach ensures smooth transitions between the primitives and provides a continuous representation of the task. The combination process is composed by the following passages:

1. **Initialization**:
   - Load the pre-trained GMMs for each primitive.
   - Concatenate the GMM parameters (priors, means, and covariances) to form a single model.
2. **Concatenation**:
   - The combined model's priors are computed by concatenating the priors of each individual model and normalizing them.

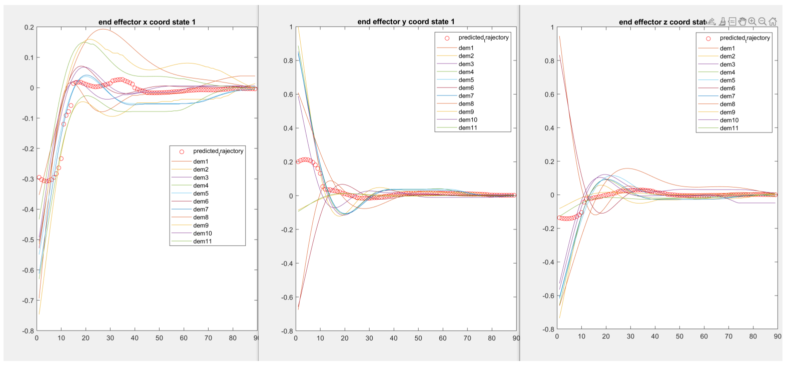

Figure 4.16: state 1 positions example

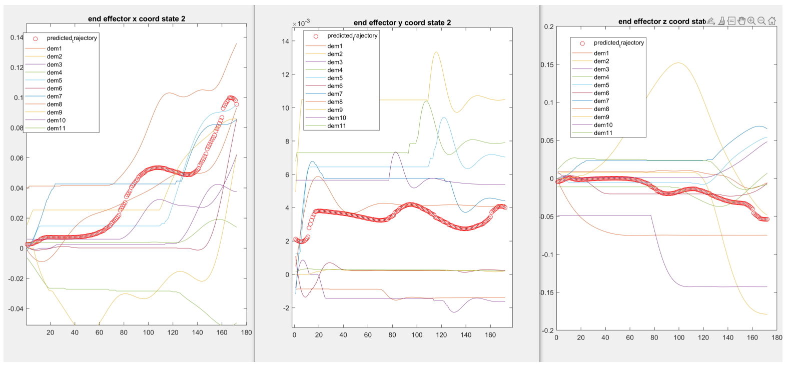

Figure 4.17: state 2 positions example

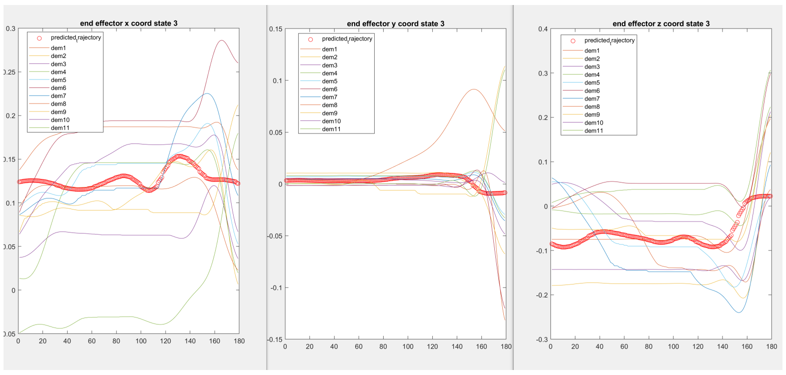

Figure 4.18: state 3 trajectories example

- The means and covariances are concatenated directly since they represent the mixture components.

3. **Generating the Time Vector**:A unified time vector is generated to represent the entire trajectory, from the start of the approach to the end of the picking.

4. **Gaussian Mixture Regression (GMR)**:The combined GMM model is

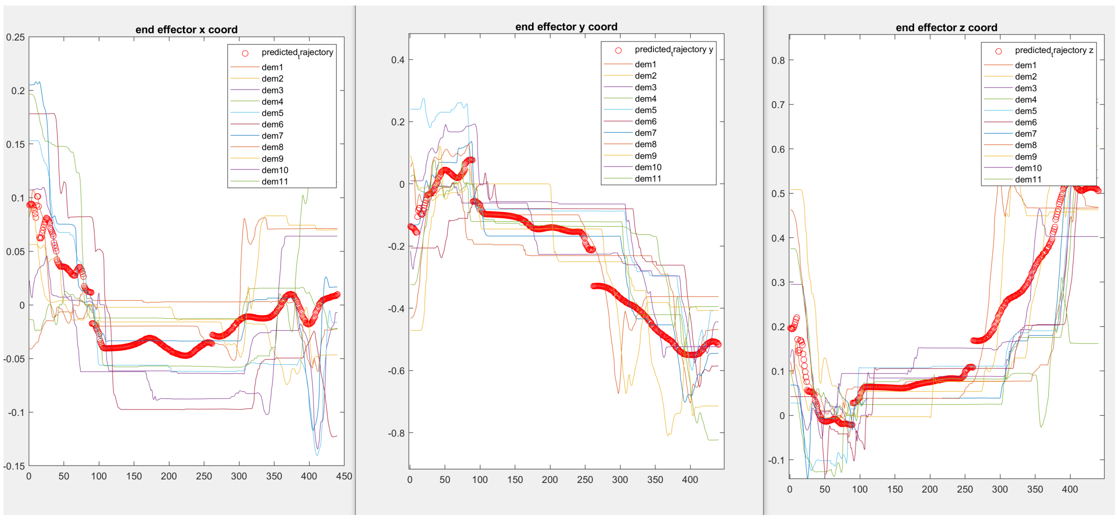

Figure 4.19: Euler angles plots

used in GMR to predict the end-effector pose given the unified time vector. The combined GMM model's priors, means, and covariances are defined as follows:

1. **Combined Priors:**

$$\pi = \frac{1}{3}(\pi_1, \pi_2, \pi_3) \tag{4.27}$$

2. **Combined Means:**

$$\mu = (\mu_1, \mu_2, \mu_3) \tag{4.28}$$

3. **Combined Covariances:**

$$\Sigma = \text{concat}(\Sigma_1, \Sigma_2, \Sigma_3) \tag{4.29}$$

4. **Gaussian Mixture Regression (GMR):**

$$\mu_{y|x} = \sum_{k=1}^{K} h_k(x)(\mu_{ky} + \Sigma_{kyx}\Sigma_{kxx}^{-1}(x - \mu_{kx})) \tag{4.30}$$

$$\Sigma_{y|x} = \sum_{k=1}^{K} h_k(x)(\Sigma_{kyy} - \Sigma_{kyx}\Sigma_{kxx}^{-1}\Sigma_{kxy}) \tag{4.31}$$

$$h_k(x) = \frac{\pi_k \mathcal{N}(x \mid \mu_{kx}, \Sigma_{kxx})}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x \mid \mu_{jx}, \Sigma_{jxx})} \tag{4.32}$$

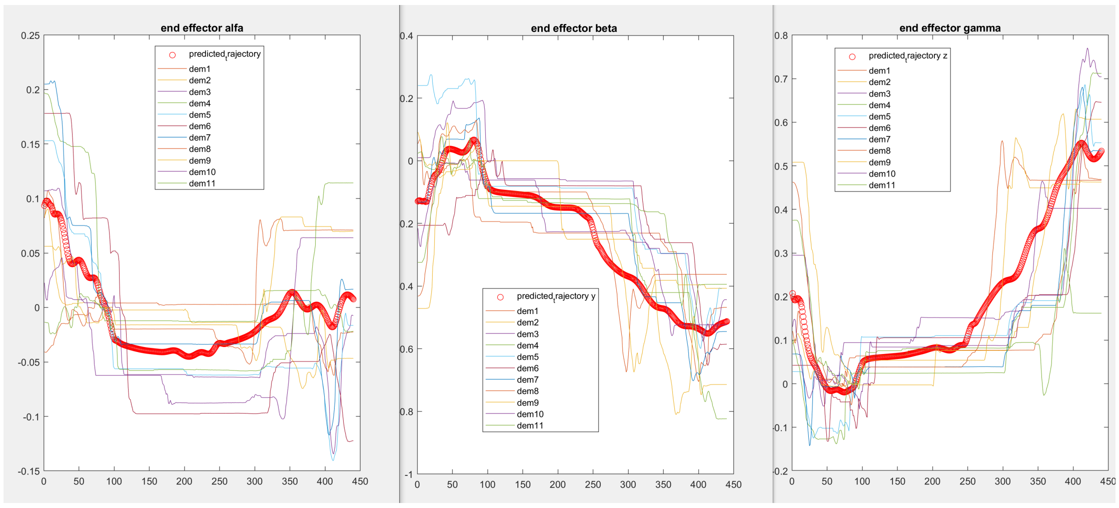

Figure 4.20: Predicted Euler angles combined

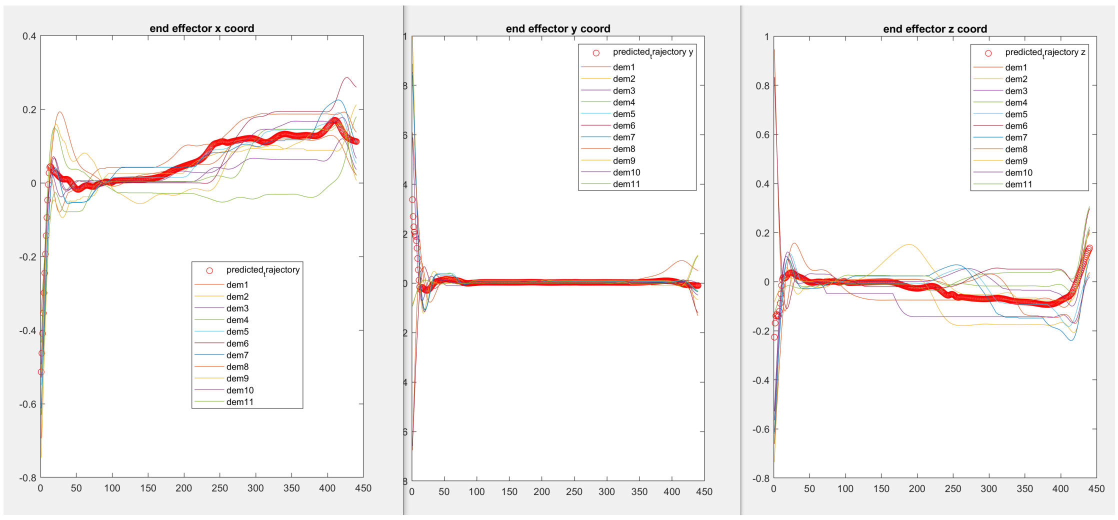

Figure 4.21: Predicted positions combined

The improvements obtained with concatenation of GMMs models are shown in plots of the predicted positions and Euler angles.(see Fig.4.20 and Fig. 4.21)

By combining the models, we can generate smooth and continuous trajectories that generalize well from the demonstrations, ensuring seamless transitions between different primitives. This approach leverages the strengths of probabilistic modeling to enhance the robustness and adaptability of robotic systems in dynamic environments.

4.3.3 Models Integration

After training HMM and GMM, there is the need to integrate them to get the final trajectory, exploiting force data and environmental constraints at a high level and predicting pose at a low level. The integration process is given below. The HMM and GMM models are linked to provide a comprehensive control system. The HMM is used to switch between primitives based on force and torque data, while the GMM models the detailed trajectories within each primitive. The combined system is tested using both training and test datasets. Testing is done by giving forces and torques data we collected, as input to HMM and predicting the pose with GMR, then taking the new force value and repeat the loop.

Control Loop Implementation for real time manipulation

The control loop takes time as input to obtain the pose and uses force data to control the model switching. The process involves the following steps:

1. Predict the current primitive using the HMM based on force and torque data.
2. Use the part of the combined GMM corresponding to the predicted primitive to generate the end effector pose.
3. Update the time index and repeat the process for the next time step.

Predicting Poses and Evaluating Performances

The predictions for both training and test sets are compared to the actual trajectories. The predicted and actual 3D positions and Euler angles are plotted to visualize the performance of the combined model. The combined HMM and GMM approach demonstrates the ability to generalize from few demonstrations, resulting in good motion reproduction. This integration leverages the strengths of both models to handle complex, multi-stage tasks effectively. For sure the sliding phase is challenging for the robot to reproduce, since it has to retry if it doesn't get the exact spot under the object. The algorithm used to integrate the models is explained below.

Following this algorithm the effect is that, if the robot end effector doesn't manage to slide under the object, it goes back and retries that primitive until it works. This way it is possible to control the loop and obtain the desired complex task.

Algorithm 2 Predict Primitives and Poses

```
Require: Best HMM model, GMM Priors, Mu, Sigma, force data, torque data,
    time vector, normalized time segments, lengths reference
Ensure: Predicted positions and Euler angles
 1: Initialize current_primitive to 1 and time_index to 1
 2: for each time step $i$ do
 3:     Predict the primitive using HMM based on force and torque
 4:     if primitive changes then
 5:         Update current_primitive and reset time_index
 6:     end if
 7:     Get the normalized time for the current_primitive
 8:     Predict pose using GMR
 9:     Store the predicted pose
10:     Update time_index
11:     if time_index exceeds range then
12:         if primitive remains the same then
13:             Revert time_index slightly
14:         else
15:             Reset time_index
16:         end if
17:     end if
18: end for
```

Chapter 5

Results

5.1 Introduction

This chapter presents the results of the testing and experiments conducted to evaluate the performance of the Hidden Markov Model (HMM), Gaussian Mixture Model (GMM), and their integrated system. The focus is on the accuracy of state prediction and trajectory generation using these models. The metrics used for evaluation include Root Mean Squared Error (RMSE), Mean Squared Error (MSE), explained variance for GMR and accuracy for HMM. Additionally, we present plots of the trajectories obtained from the training, test datasets and simulations.

5.2 Hidden Markov Model (HMM) Results

5.2.1 Training Set Results

The HMM was trained on the force data collected during the simulated demonstrations. The performance was evaluated based on accuracy,precision, recall and F1 score of state predictions and plotting the predicted states on top of the actual states as suggested in [41]. In the table below 5.1the values of metrics for different numbers of Gaussian mixtures and for different numbers of force components selected are shown.

Formulas of the used metrics are listed below:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{5.1}$$

Accuracy measures the overall correctness of the model by dividing the number of correct predictions by the total number of predictions. High accuracy indicates

Table 5.1: Metrics for Different HMM Configurations in training set

Force Components	**Gaussians**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
2	1	0.943	0.95	0.92	0.94
	2	0.95	0.96	0.937	0.943
	3	0.95	0.96	0.912	0.93
3	1	0.939	0.959	0.919	0.9331
	2	0.945	0.96	0.925	0.937
	3	0.94	0.945	0.946	0.942
6	1	0.854	0.863	0.84	0.851
	2	0.88	0.89	0.885	0.87
	3	0.876	0.865	0.832	0.85

that the model is generally making correct predictions.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \tag{5.2}$$

Precision measures the accuracy of the positive predictions. It is the ratio of true positive predictions to the total positive predictions made by the model. High precision indicates that the model is correctly identifying positive instances with few false positives. [42]

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \tag{5.3}$$

Recall measures the model's ability to find all the relevant cases within a dataset. It is the ratio of true positive predictions to the total actual positives. High recall indicates that the model is successfully capturing most of the positive instances, with few false negatives.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

F1 Score is the harmonic mean of precision and recall, providing a single metric that balances the trade-off between precision and recall. A high F1 score indicates that the model has both high precision and high recall.[43]

Trajectory Plots: The following figures illustrate the predicted and actual state sequences for training demonstrations.(see Fig. 5.1)

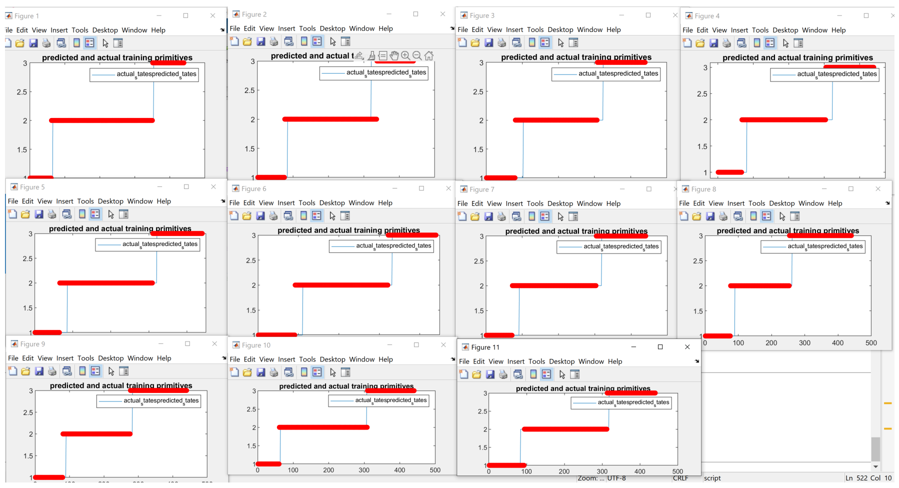
predicted and actual training primitives
actual_statespredicted_states
Figure 1
File Edit View Insert Tools Desktop Window Help

Figure 5.1: HMM - Training Set State Predictions

5.2.2 Test Set Results

The HMM was tested on a separate set of force data composed by 3 demonstrations. Results are shown in the table 5.2.

Table 5.2: Metrics for Different HMM Configurations in Test set

Force Components	Gaussians	Accuracy	Precision	Recall	F1 Score
2	1	0.96	0.967	0.94	0.95
	2	0.965	0.966	0.97	0.961
	3	0.94	0.95	0.96	0.95
3	1	0.952	0.966	0.9345	0.947
	2	0.957	0.975	0.935	0.952
	3	0.95	0.93	0.97	0.95
6	1	0.854	0.862	0.841	0.856
	2	0.915	0.89	0.92	0.89
	3	0.	0.88	0.864	0.873

Trajectory Plots: The following figures illustrate the predicted and actual state sequences for test demonstrations.

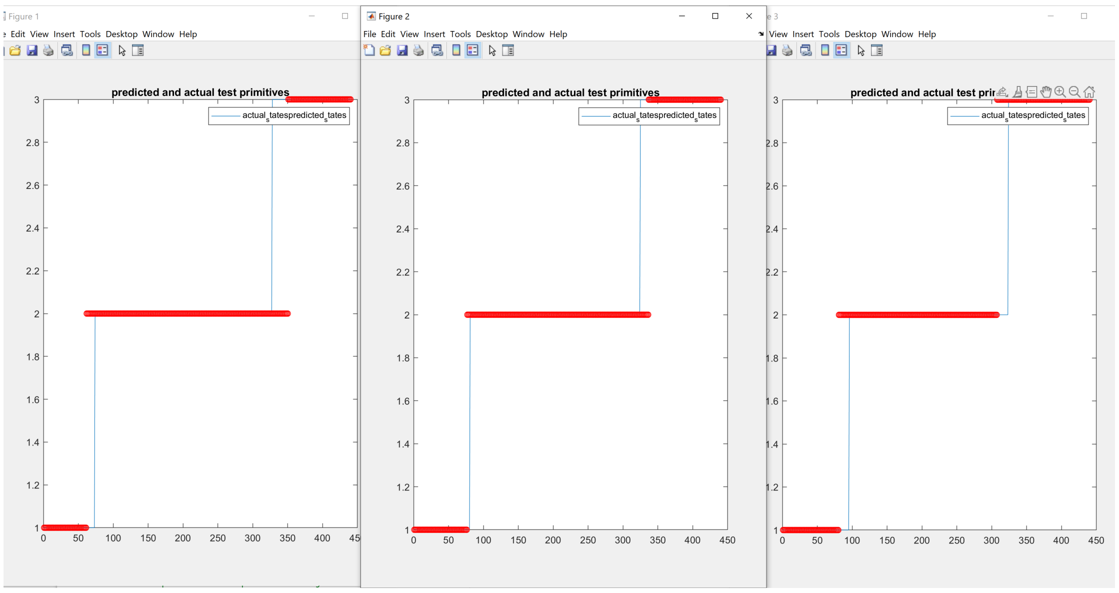

Figure 5.2: HMM - Test Set State Predictions

As can be highlighted from the tables above, evaluating performances on training and test set, the best model is the one with 2 components of forces, specifically force along y and torque around x, and 2 Gaussian mixtures to model emission probabilities.

5.3 GMM/GMR Results

The GMM was trained to model the detailed trajectories within each primitive.After visualizing the results of the trajectories for each state, concatenation was applied in order to create a single model with smooth transitions. The performance of the combined model was evaluated based on the generalization capability of trajectory GMR regressions. Metrics used to evaluated performances are MSE, RMSE and variance. Formulas of these metrics are listed below:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.5}$$

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{5.6}$$

$$\text{Variance} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}})^2 \tag{5.7}$$

5.3.1 Training Set Results

To evaluate performances of the regression on training the following table 5.3 is produced.

Table 5.3: Training Set Metrics for Different Number of Gaussians

Number of Gaussians	**Position MSE**	**Position RMSE**	**Position Variance**	**Euler MSE**	**Euler RMSE**	**Euler Variance**
5	0.0044	0.069	0.47	0.0085	0.095	0.48
10	0.004	0.062	0.53	0.0081	0.089	0.565
15	0.0039	0.059	0.56	0.0072	0.082	0.6
20	0.0042	0.063	0.55	0.0076	0.086	0.58
30	0.0045	0.067	0.52	0.0083	0.091	0.57

Trajectory Plots: The following figures illustrate the predicted and actual trajectories for training demonstrations (see Fig. 5.3).

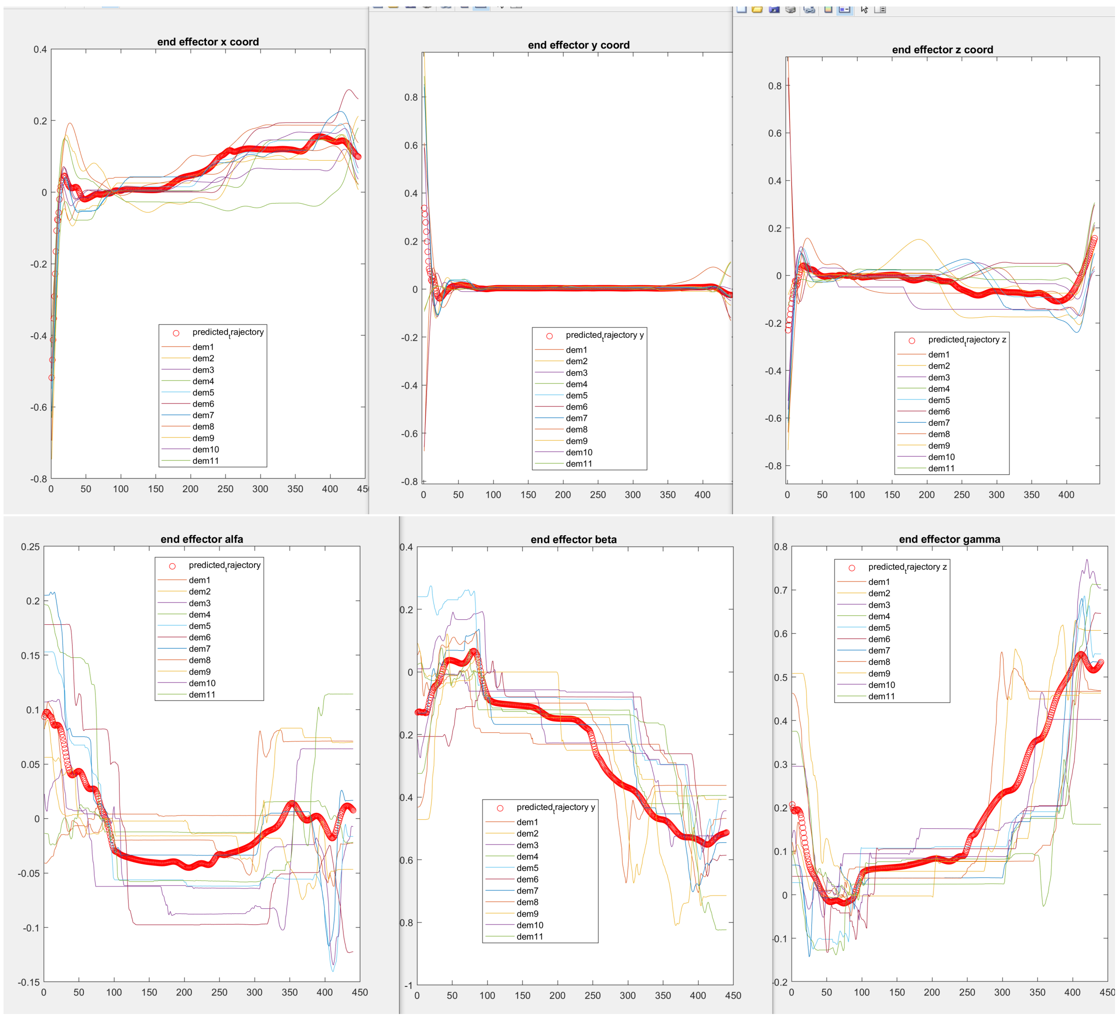

Figure 5.3: GMR - Training Set Trajectory Generalization

5.3.2 Test Set Results

To evaluate performances of the regression on test set the following table 5.4 is produced.

Table 5.4: Test Set Metrics for Different Number of Gaussians

Number of Gaussians	Position MSE	Position RMSE	Position Variance	Euler MSE	Euler RMSE	Euler Variance
5	0.0055	0.0074	0.32	0.027	0.17	0.38
10	0.0046	0.068	0.395	0.024	0.16	0.48
15	0.0042	0.065	0.45	0.02	0.15	0.51
20	0.0048	0.071	0.39	0.023	0.158	0.49
30	0.0053	0.072	0.38	0.025	0.16	0.48

Trajectory Plots: The following figures illustrate the predicted and actual trajectories for test demonstrations (see Fig. 5.4).

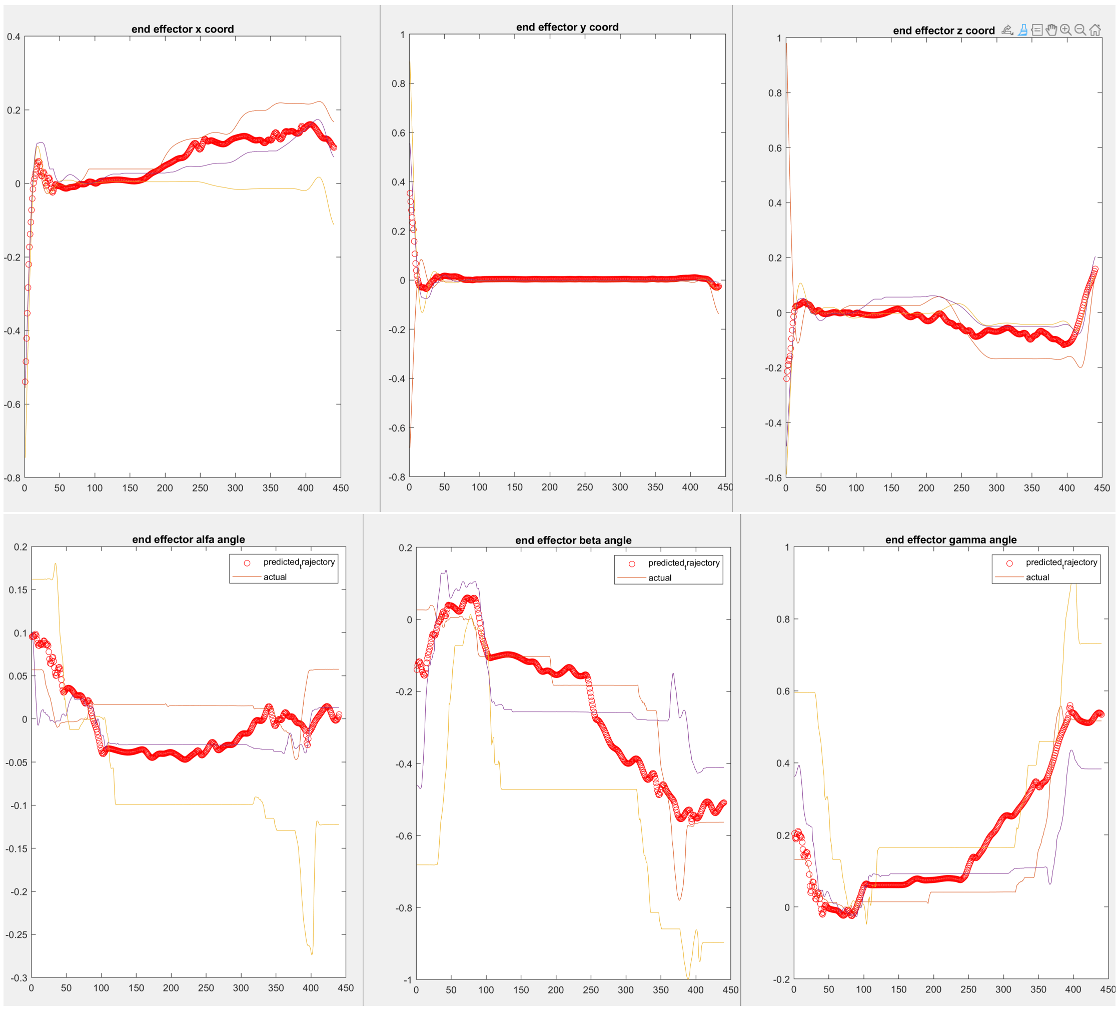

Figure 5.4: GMR - Test Set Trajectory Generalization

From the tables above can be highlighted the fact that the best performances for the GMR are obtained with 15 Gaussians to model data.

5.4 Integrated HMM-GMM System Results

From the previous paragraphs describing parameters tuning, can be understood that the best HMM and GMM/GMR models are respectively, the one with 2 force components and 2 Gaussians as emission distributions, and the one with 15 as number of Gaussian mixtures. The other parameters of the model did not have a significant impact on the performances, probably due to the architecture of the models.

5.4.1 Training Set Results

The integrated HMM-GMM system was evaluated on the training set to assess the combined performance of state prediction and trajectory generation. The evaluation was done giving forces data from each demonstration as input to HMM and obtaining the current primitive, then the pose was predicted from GMR. The analysis has been done comparing each predicted trajectory to the actual demonstration's trajectory. The model's effectiveness was measured using metrics such as MSE, RMSE, and variance for both the position and Euler angles. The following results provide a detailed analysis of the system's performance on the training set.

Training Set Metrics:

- **Position MSE:** 0.00452
- **Position RMSE:** 0.0673
- **Position Variance:** 0.47
- **Euler Angles MSE:** 0.0076
- **Euler Angles RMSE:** 0.0873
- **Euler Angles Variance:** 0.6

These metrics indicate that the integrated system was able to accurately predict the trajectories within the training data, maintaining low error rates and variance.

5.4.2 Test Set Results

The integrated HMM-GMM system was tested on a separate set of data to evaluate its generalization performance. The model's ability to accurately predict trajectories in **unseen data** was assessed using same metrics as those for the training set.

Test Set Metrics:

- **Position MSE:** 0.0049
- **Position RMSE:** 0.07
- **Position Variance:** 0.42
- **Euler Angles MSE:** 0.0248

- **Euler Angles RMSE:** 0.157

- **Euler Angles Variance:** 0.48

These results demonstrate that while the model performs slightly worse on the test set compared to the training set, it still maintains a reasonable level of accuracy and consistency in trajectory prediction.

Trajectory Plots: The figures below illustrate the predicted and actual trajectories for test demonstrations using the integrated HMM-GMM system.

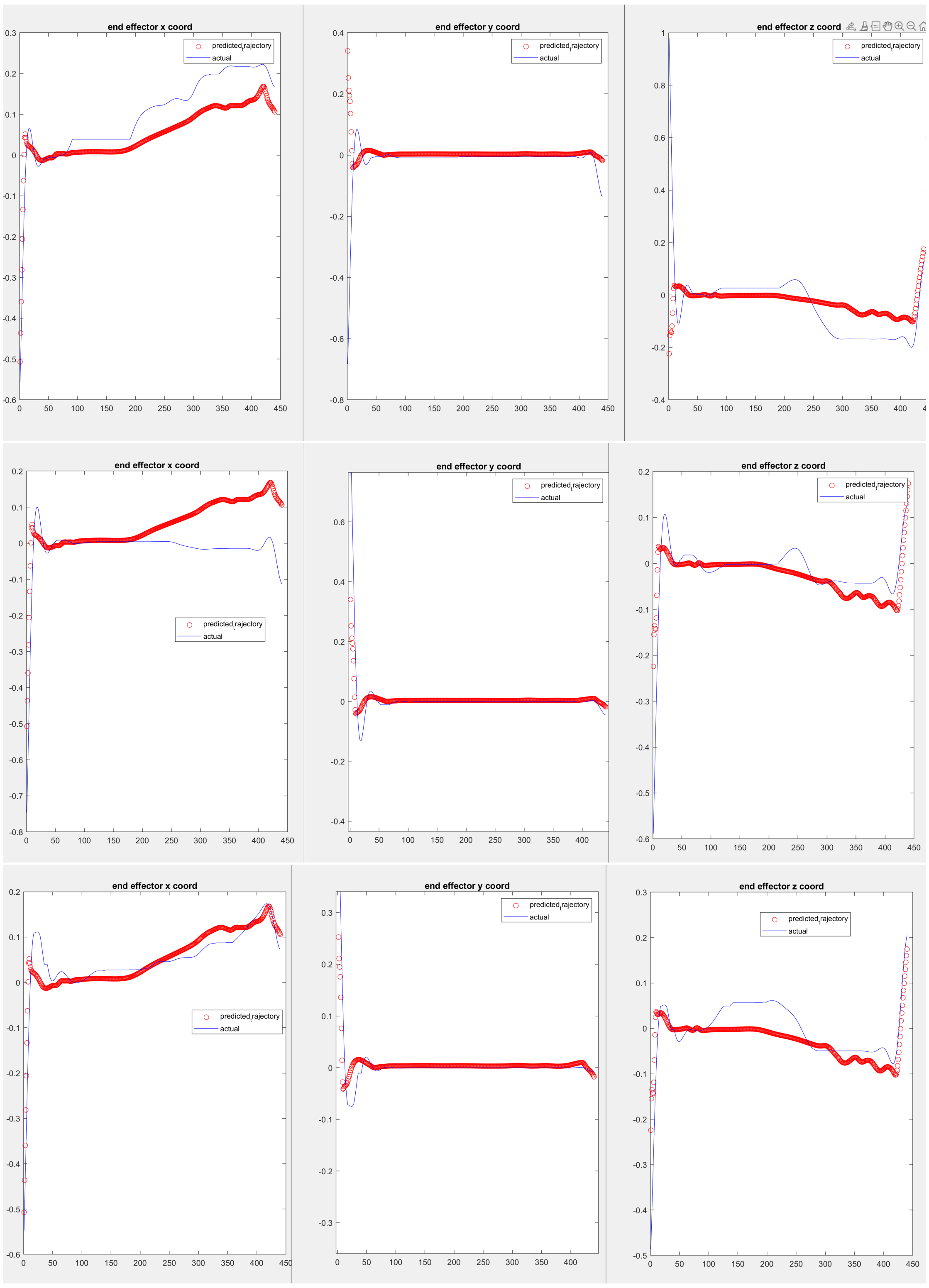

Figure 5.5: Integrated HMM-GMM - Test Set Positions Predictions

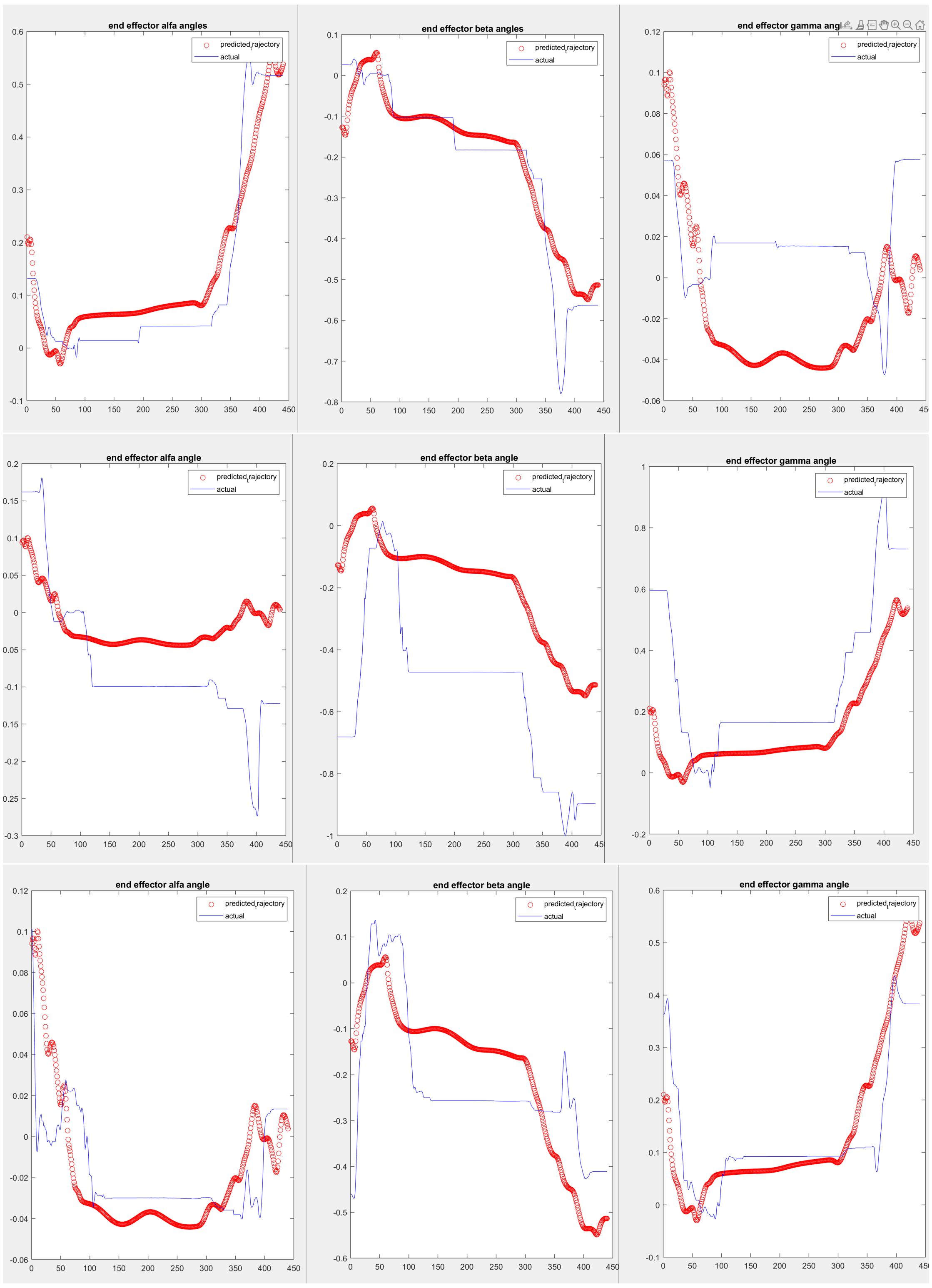

Figure 5.6: Integrated HMM-GMM - Test Set Orientation Predictions

5.4.3 Summary

The integrated HMM-GMM system demonstrated strong performance on both training and test sets, with low RMSE and variance values indicating accurate and consistent trajectory predictions. These results suggest that the combination of HMM for state prediction and GMM for trajectory generation provides a robust method for modeling complex robotic movements.

Chapter 6

Discussions, applications and future works

Lfd is a really promising approach to robot learning which can be applied to a variety of fields. The research has to continue in producing new and better approaches to guarantee robustness, reliability, safety and versatility. In our study this problems have been faced and analysed understanding strength and weaknesses of this approach. More in depth, from results we can see that each model alone performs with a high level of accuracy, robustness and precision. When we integrate two models we lose a little bit of accuracy, still not significant, but we gain a high level control based on Environmental Constraints. This way our robot is able to understand when he has to repeat a primitive, the approach to the object in our case. This improves the robustness of the model and its capabilities of adapting to different scenarios exploiting environmental constraints. Even though the model performs well there are improvements that can be implemented in future research.

6.1 Possible Improvements

While the current methodology using HMMs and GMMs exploiting EC, has shown promising results, there are several avenues for potential improvement and future research. These enhancements can address the limitations identified during the research and expand the applicability and robustness of the model.

6.1.1 Implementing Vision-Based Systems

One significant improvement is the integration of vision-based systems. By incorporating visual feedback, robots can gain a better understanding of their environment, allowing for more precise manipulation and interaction with objects. Vision systems can aid in:

- Object Recognition and Localization: Using cameras and computer vision algorithms, robots can identify and locate objects with higher accuracy.
- Dynamic Adaptation: Vision can enable robots to adapt to changes in the environment in real-time, improving their ability to handle unforeseen situations.
- Enhanced Grasping: Visual feedback can help in adjusting the grasping technique based on the object's shape, size, and orientation.

6.1.2 Extending to Different Real-World Settings

The current experiments are primarily conducted in controlled environments. Extending these methods to more varied and unstructured real-world settings can significantly enhance their robustness and utility. For example changing the positions of the object or their orientation in space in order to make the approach able to generalize the motion.

6.1.3 Developing Time-Independent Models

Current models are time-dependent, which can limit their flexibility and robustness. Future research can focus on developing models that are less dependent on time or even completely time-independent. This can be achieved by using Trajectory Optimization, which consists in employing optimization techniques to find the best path for task execution without being constrained by predefined time sequences.

6.1.4 Reducing Supervision and automate the process

During the preparation of data for HMM training the analysis made by the designer is been crucial to identify the division in primitives. This can be a point of weakness if we want to have an approach completely independent from users. During HMM training, boundary and initial constraints have been applied from knowledge coming from the designer. An improvement would be to automate all this process and transform the approach in a totally unsupervised one.

6.2 Possible Applications

The integration of advanced robotics with probabilistic models like HMMs and GMMs has vast potential across various fields. This section explores some of these applications, focusing on general applications and delving into specific biomedical applications.

6.2.1 General Applications

Industrial Robotics

In industrial settings, robots can be used for tasks such as assembly, welding, painting, and quality inspection. The ability to learn from demonstrations and adapt to changes in the environment makes these models ideal for dynamic manufacturing processes.

Agricultural Robotics

Robots can be employed in agriculture for planting, harvesting, and monitoring crops. Vision-based systems can help in identifying ripe fruits and vegetables, while probabilistic models can optimize the picking process.

Service Robots

Service robots can assist in various domains such as hospitality, retail, and customer service. They can guide customers, manage inventories, and provide information, improving efficiency and customer experience.

Space Robotics

Robots in space missions perform tasks like satellite servicing, space station maintenance, and planetary exploration. The ability to adapt to unpredictable environments and execute tasks with high precision is crucial in space robotics. These applications require robust and reliable models due to the harsh and variable conditions in space.

6.3 Biomedical Applications

The integration of robotics in biomedical engineering presents unique opportunities and challenges. This subsection highlights the applications of these methods in the biomedical field, emphasizing the stringent requirements for precision, robustness, and ethical considerations.

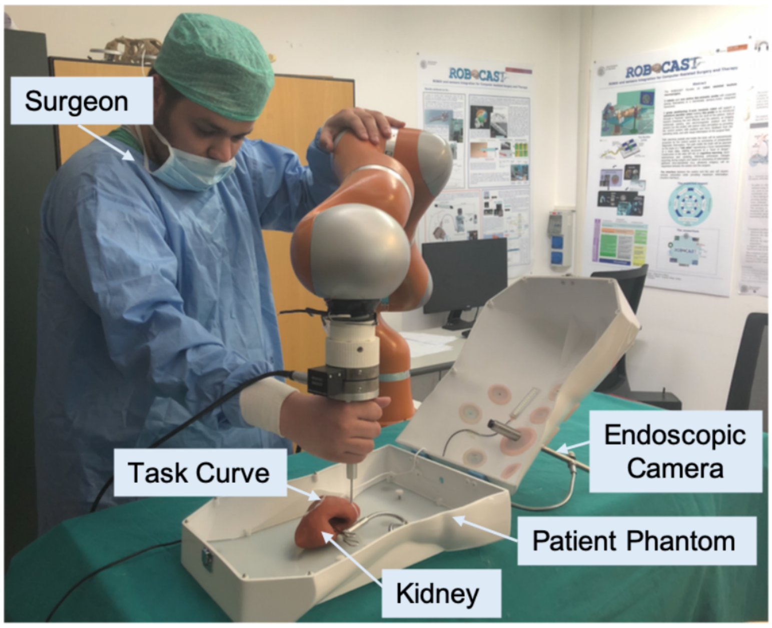

Fig. 5. Demonstration setup scene in open surgery. A kidney tissue model with a size of around $135 \times 45 \times 30mm^3$ is presented in the 3-D patient phantom ($170 \times 210 \times 100mm^3$). The patient phantom is opened, and a metal clip fixes the kidney model in the abdominal cavity. A white task curve is drawn in advance along a blood vessel on the surface of the kidney to serve as the specific tracking task. The robot is activated in hands-on control mode to enable the surgeon to relocate the surgical tip by hand. The "surgeon" is commanded to do multiple demonstrations of tracking the white task curve with the surgical tip.

Figure 6.1: Figure 5 from [44]

6.3.1 Surgical Robotics

Surgical robots, such as the da Vinci Surgical System, enhance the precision and control of surgeons during minimally invasive procedures. Probabilistic models can improve the robots' ability to predict and adapt to the surgeon's movements, ensuring accurate execution of complex surgical tasks. The requirements for surgical robotics include:

- High Precision: Any deviation from the planned trajectory can have significant consequences.
- Robustness: The system must be reliable and capable of handling unforeseen complications.

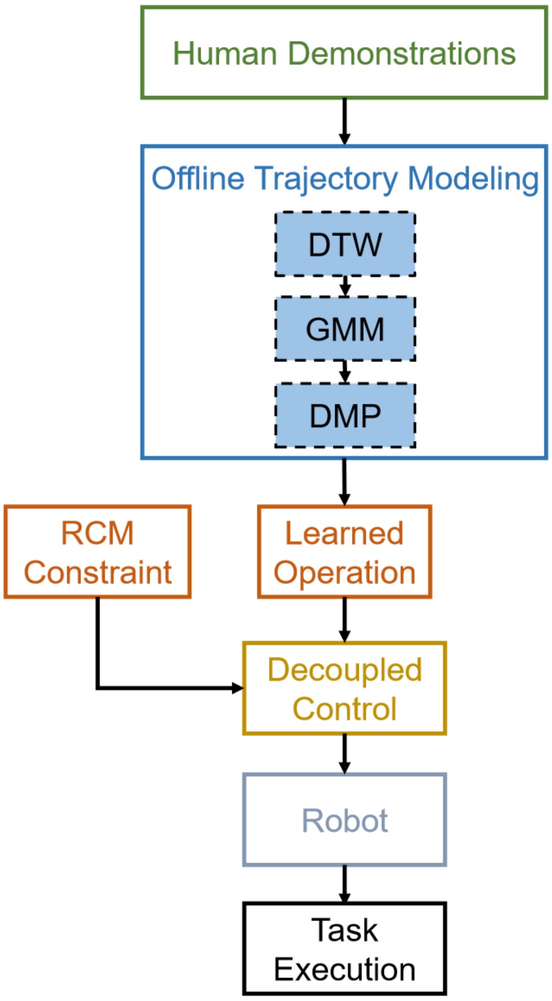

Figure 6.2: Lfd process from [44]

- Specificity: The model must be tailored to specific surgical procedures to ensure optimal performance.

Our approach is similar to the one implemented in recent studies in Surgical Robotics where Lfd is used to teach and learn surgical tasks such as in [44] and [45]. (see Fig. 6.1 and Fig. 6.2 from [44])

6.3.2 Robotic Assistants in Operating Rooms

Robots can assist surgeons by handling instruments, manipulating tissues, and providing real-time feedback. They can help reduce the physical strain on surgeons and improve the overall efficiency of surgical procedures. These systems must meet high standards for:

- Safety: Ensuring the safety of both patients and medical staff is paramount.
- Precision: Accurate manipulation of instruments and tissues is essential.
- Reliability: The system must be dependable and function correctly throughout the procedure.

Our approach implements a movement that exploits environmental constraints to slide under an object and picking it up which makes it pretty suitable to this field, to help nurses in picking and handling surgical instruments in critical moments.

6.3.3 Automating Pharmacies

Automated pharmacy systems can dispense medications, manage inventory, and ensure that patients receive the correct dosage. These systems can significantly reduce human errors and improve efficiency in pharmaceutical operations. Key requirements include:

- Accuracy: Ensuring the correct medication and dosage is dispensed.
- Efficiency: Managing high volumes of prescriptions and inventory.
- Security: Safeguarding against theft and unauthorized access.

In situations in which medications are stacked, our approach would be crucial in picking boxes.

6.3.4 Ethical and Legal Considerations

The application of robotics in the biomedical field raises important ethical and legal issues, such as:

- Safety: since tasks performed in this field can put life at risk the level of safety to guarantee is high.
- Patient Privacy: Ensuring that patient data is protected and used ethically.
- Consent: Obtaining informed consent for the use of robotic systems in medical procedures.

- Accountability: Determining responsibility in case of malfunctions or errors.
- Accessibility: Ensuring that advanced robotic systems are accessible to all patients, regardless of socioeconomic status.

Chapter 7

Conclusions

The research presented in this thesis has demonstrated the feasibility and effectiveness of using **HMMs** and **GMMs** for robotic learning and adaptation exploiting environmental constraints. By combining these probabilistic models, we have developed a robust framework for enabling robots to learn from demonstrations, adapt to dynamic environments, and perform complex tasks. The potential applications, particularly in the biomedical field, highlight the transformative impact these technologies can have in different fields. Research and development in this area will lead to more advanced, efficient, and accessible robotic solutions, ultimately benefiting society as a whole.

Appendix A

Support Code

A.1 GMM/GMR Training and Evaluation Code

Listing A.1: GMM/GMR Training and Evaluation Code

```
% Clear workspace and load data senza PCA
clear all;
clc;
load('data_carteul');
load('primitives_matrix.mat');
load('primitives_matrix_test.mat');
load('training_quat_cart.mat');
load('training_eul_force.mat');
load('training_eul_cart.mat');
load('test_eul_cart.mat');
load('test_eul_force.mat');
load('data_cart_norm2_quat.mat');
load('test_quat_cart.mat');
load('time.mat');
a=[1,2,5,6,7,8,9,11,12,13,15];

% Parameters
min_mixtures = 3; % Minimum number of mixtures to ensure complexity
max_mixtures = 35; % Maximum number of mixtures to test
nbStates = 3; % Number of states in the GMM (number of primitives)
model.nbVar = 1 + 3 + 3; % Number of variables [time, translation (x,
    y, z), rotation (roll, pitch, yaw)]
nbSamples = 11; % Number of training demonstrations
nbTestSamples = 3; % Number of test demonstrations
resample_lengths = [89, 172, 179]; % Lengths for resampling each
    primitive
pose_data = training_eul_cart;
regularization = 1e-8;

```

```
% Training and test sets for pose data
train_pose_data_euler = pose_data;
test_pose_data = test_eul_cart(:, :, [1 2 4]);
test_primitives_matrix = primitives_matrix_test;
train_primitives_matrix = primitives_matrix;

% Divide and normalize time vector between -1 and 1
time_vector1 = time(1:resample_lengths(1));
time_vector2 = time(resample_lengths(1)+1:resample_lengths(1)+resample_lengths(2));
time_vector3 = time(resample_lengths(1)+resample_lengths(2)+1:end);

% Resample data for each state
Data = cell(1, nbStates);
time_vectors = {time_vector1, time_vector2, time_vector3}; % Store time vectors separately
for state = 1:nbStates
    Data{state} = [];
    repeated_time_vector = repmat(time_vectors{state}', 1, nbSamples);
    repeated_time_vector = repeated_time_vector(:)'; % Convert to row vector
    for n = 1:nbSamples
        state_indices = find(train_primitives_matrix(:, n) == state);
        demo_resampled = spline(1:length(state_indices), train_pose_data_euler(state_indices, :, n)', linspace(1, length(state_indices), resample_lengths(state)));
        Data{state} = [Data{state}, demo_resampled];
    end
    Data{state} = [repeated_time_vector; Data{state}];
end

% Train GMM models and plot BIC values
gmmModels_nopca_time7 = cell(nbStates, 1);
BIC_values = zeros(nbStates, max_mixtures - min_mixtures + 1);
figure;
hold on;

for state = 1:nbStates
    [gmmModels_nopca_time7{state}, BIC_values(state, :)] = train_GMM(Data{state}, min_mixtures, max_mixtures, resample_lengths(state), regularization);
    plot(min_mixtures:max_mixtures, BIC_values(state, :), 'DisplayName', ['State ' num2str(state)]);
end

xlabel('Number of mixtures');
ylabel('BIC');
legend;
```

```
title('BIC for different number of mixtures');

% Save GMM models
save('gmmModels_nopca_time7.mat', 'gmmModels_nopca_time7');

% Plot GMM results
for state = 1:nbStates
    figure('position', [10, 10, 700, 500]); hold on; axis off;
    plot(Data{state}(2,:), Data{state}(3,:), '.', 'markersize', 8, 'color', [.5 .5 .5]);
    plotGMM(gmmModels_nopca_time7{state}.Mu, gmmModels_nopca_time7{state}.Sigma, [.8 0 0], .5);
    axis equal; set(gca, 'Xtick', []); set(gca, 'Ytick', []);
end

% Generalize trajectories using GMR and calculate metrics
total_mse_train = 0;
num_train_points = 0;
total_mse_test = 0;
num_test_points = 0;

% Create resampled test data
TestData = cell(1, nbStates);
for state = 1:nbStates
    TestData{state} = [];
    repeated_time_vector_test = repmat(time_vectors{state}', 1, nbTestSamples);
    repeated_time_vector_test = repeated_time_vector_test(:)'; % Convert to row vector
    for n = 1:nbTestSamples
        state_indices = find(test_primitives_matrix(:, n) == state);
        demo_resampled = spline(1:length(state_indices), test_pose_data(state_indices, :, n)', linspace(1, length(state_indices), resample_lengths(state)));
        TestData{state} = [TestData{state}, demo_resampled];
    end
    TestData{state} = [repeated_time_vector_test; TestData{state}];
end

for state = 1:nbStates
    [pos, euler] = generalize_GMR(gmmModels_nopca_time7{state}, time_vectors{state}', model.nbVar-1);

    % Training metrics
    actual_data_train = Data{state}(2:end, :);
    predicted_data_train = repmat([pos; euler], 1, nbSamples);
    predicted_data_train = predicted_data_train(:, 1:size(actual_data_train, 2)); % Adjust size to match

    % Ensure sizes match before calculating MSE
    if size(predicted_data_train, 1) == size(actual_data_train, 1) && size(predicted_data_train, 2) == size(actual_data_train, 2)
        mse_train = immse(predicted_data_train, actual_data_train);
        total_mse_train = total_mse_train + mse_train * numel(actual_data_train);
        num_train_points = num_train_points + numel(actual_data_train);
    else
        disp(['Size mismatch in training data for state ', num2str(state)]);
        disp(['Predicted size: ', num2str(size(predicted_data_train)), ', Actual size: ', num2str(size(actual_data_train))]);
    end

    % Test metrics
    actual_data_test = TestData{state}(2:end, :);
    predicted_data_test = repmat([pos; euler], 1, nbTestSamples);
    predicted_data_test = predicted_data_test(:, 1:size(actual_data_test, 2)); % Adjust size to match

    % Ensure sizes match before calculating MSE
    if size(predicted_data_test, 1) == size(actual_data_test, 1) && size(predicted_data_test, 2) == size(actual_data_test, 2)
        mse_test = immse(predicted_data_test, actual_data_test);
        total_mse_test = total_mse_test + mse_test * numel(actual_data_test);
        num_test_points = num_test_points + numel(actual_data_test);
    else
        disp(['Size mismatch in test data for state ', num2str(state)]);
        disp(['Predicted size: ', num2str(size(predicted_data_test)), ', Actual size: ', num2str(size(actual_data_test))]);
    end

    % Display state-wise metrics
    disp(['State ' num2str(state) ' - MSE (Train): ' num2str(mse_train)]);
    if exist('mse_test', 'var')
        disp(['State ' num2str(state) ' - MSE (Test): ' num2str(mse_test)]);
    end

    plot_trajectory(pos, euler, Data{state}(2:7, :), time_vectors{state}');
end

% Overall metrics
overall_mse_train = total_mse_train / num_train_points;
```

```
overall_mse_test = total_mse_test / num_test_points;

disp(['Overall MSE (Train): ', num2str(overall_mse_train)]);
disp(['Overall MSE (Test): ', num2str(overall_mse_test)]);

function [gmmModel, BIC_values] = train_GMM(data, min_mixtures,
    max_mixtures, resample_lengths, regularization)
     nbStatesOptions = min_mixtures:max_mixtures;
     minBIC = Inf;
     bestModel = [];
     BIC_values = zeros(1, length(nbStatesOptions));

     for idx = 1:length(nbStatesOptions)
          nbStates = nbStatesOptions(idx);
          model.nbStates = nbStates;
          model.nbVar = size(data, 1); % Including time
          model = init_GMM_kmeans(data, model);
          model = EM_GMM(data, model, regularization);

          BIC = calculate_BIC(data, model);
          BIC_values(idx) = BIC;
          if BIC < minBIC
               minBIC = BIC;
               bestModel = model;
          end
     end

     gmmModel = bestModel;
end

function BIC = calculate_BIC(data, model)
     % Calculate the Bayesian Information Criterion for the GMM model
     logL = calculate_log_likelihood(data, model);
     numParams = model.nbStates * (2 * model.nbVar + 1); % Approximate
      number of parameters
     BIC = -2 * logL + numParams * log(size(data, 2));
end

function logL = calculate_log_likelihood(data, model)
     % Calculate the log-likelihood of the GMM model
     logL = 0;
     for t = 1:size(data, 2)
          prob = 0;
          for i = 1:model.nbStates
               prob = prob + model.Priors(i) * gaussPDF(data(:, t),
    model.Mu(:, i), model.Sigma(:, :, i));
          end
          logL = logL + log(prob);
     end
```

```
end

function [Mu, Sigma] = GMR(Priors, Mu, Sigma, x, in)
    % Gaussian Mixture Regression (GMR)
    nbData = size(x, 2);
    nbVar = size(Mu, 1);
    nbStates = size(Sigma, 3);
    MuTmp = zeros(nbVar, nbData);
    SigmaTmp = zeros(nbVar, nbVar, nbData);

    for t = 1:nbData
        for i = 1:nbStates
            H(i) = Priors(i) * gaussPDF(x(:, t), Mu(in, i), Sigma(in,
     in, i));
        end
        H = H ./ sum(H);
        for i = 1:nbStates
            MuTmp(:, t) = MuTmp(:, t) + H(i) * Mu(:, i);
            SigmaTmp(:, :, t) = SigmaTmp(:, :, t) + H(i) * (Sigma(:,
    :, i) + Mu(:, i) * Mu(:, i)') - MuTmp(:, t) * MuTmp(:, t)';
        end
    end
    Mu = MuTmp;
    Sigma = SigmaTmp;
end

function prob = gaussPDF(Data, Mu, Sigma)
    % Gaussian Probability Density Function (PDF)
    [nbVar, nbData] = size(Data);
    Data = Data' - repmat(Mu', nbData, 1);
    prob = sum((Data * inv(Sigma)) .* Data, 2);
    prob = exp(-0.5 * prob) / sqrt((2 * pi)^nbVar * (abs(det(Sigma))
    + realmin));
end

function [pos, euler] = generalize_GMR(model, timeVector, dimensions)
    % Generalize trajectories using GMR
    [Mu, Sigma] = GMR(model.Priors, model.Mu, model.Sigma, timeVector
    , 1);
    pos = Mu(2:4, :); % Assuming positions are in dimensions 2 to 4
    euler = Mu(5:7, :); % Assuming Euler angles are in dimensions 5
    to 7
end

function plot_trajectory(pos, euler, actualData, timeVector)
    figure;
    subplot(2, 1, 1);
    plot3(pos(1, :), pos(2, :), pos(3, :), 'r'); hold on;
    plot3(actualData(1, :), actualData(2, :), actualData(3, :), 'b');
    title('3D Positions');

    subplot(2, 1, 2);
    plot(timeVector, euler, 'r'); hold on;
    plot(repmat(timeVector, 1, size(actualData, 2) / size(timeVector, 2)), actualData(4:6, :), 'b');
    title('Euler Angles');
end

function normalized_time = normalize_time(time_vector)
    % Normalize time vector to range from -1 to 1
    normalized_time = (time_vector - min(time_vector));
end

function model = init_GMM_kmeans(Data, model)
    % K-means initialization for GMM
    [nbVar, nbData] = size(Data);
    [Data_id, Centers] = kmeans(Data', model.nbStates);
    for i = 1:model.nbStates
        idtmp = find(Data_id == i);
        model.Priors(i) = length(idtmp);
        model.Mu(:, i) = mean(Data(:, idtmp), 2);
        model.Sigma(:, :, i) = cov(Data(:, idtmp)') + 1E-8 * diag(ones(nbVar, 1));
    end
    model.Priors = model.Priors ./ sum(model.Priors);
end

function model = EM_GMM(Data, model, regularization)
    % EM algorithm for GMM
    [nbVar, nbData] = size(Data);
    loglik_threshold = 1e-10;
    max_iter = 10000;
    loglik_old = -realmax;

    for iter = 1:max_iter
        % E-step
        for i = 1:model.nbStates
            Pxi(:, i) = model.Priors(i) * gaussPDF(Data, model.Mu(:, i), model.Sigma(:, :, i));
        end
        Px = sum(Pxi, 2);
        Pix = Pxi ./ repmat(Px, 1, model.nbStates);
        E = sum(Pix);

        % M-step
        for i = 1:model.nbStates
            model.Priors(i) = E(i) / nbData;
            model.Mu(:, i) = Data * Pix(:, i) / E(i);
            Data_tmp1 = Data - repmat(model.Mu(:, i), 1, nbData);
            model.Sigma(:, :, i) = (Data_tmp1 * diag(Pix(:, i)) *
    Data_tmp1' + regularization * diag(ones(nbVar, 1))) / E(i);
        end

        % Compute log-likelihood
        loglik = sum(log(Px));
        if abs((loglik / loglik_old) - 1) < loglik_threshold
            break;
        end
        loglik_old = loglik;
    end
end

function plotGMM(Mu, Sigma, color, alpha)
    % Plot the GMM components
    nbStates = size(Mu, 2);
    for i = 1:nbStates
        plot_gaussian_ellipsoid(Mu(2:3, i), Sigma(2:3, 2:3, i), 2,
    color, alpha);
    end
end

function plot_gaussian_ellipsoid(Mu, Sigma, scale, color, alpha)
    [V, D] = eig(Sigma);
    t = linspace(0, 2 * pi, 100);
    xy = [cos(t); sin(t)];
    k = scale * (V * sqrt(D)) * xy;
    fill(Mu(1) + k(1, :), Mu(2) + k(2, :), color, 'FaceAlpha', alpha,
    'EdgeColor', 'none');
end
```

A.2 HMM Training and Evaluation Code

Listing A.2: HMM Training and Evaluation Code

```
load('primitives_matrix.mat');
load('primitives_matrix_test.mat');
load('training_quat_cart.mat');
load('training_eul_force.mat');
load('training_eul_cart.mat');
load('test_eul_cart.mat');
load('test_eul_force.mat');
load('data_cart_norm2_quat.mat');
load('test_quat_cart.mat');
load('time.mat')

```

```
b=[1 2 4];
% Main script for training and evaluating Continuous HMM with
    Gaussian emissions
force_data = training_eul_force(:,2:4,:);
force_test_data = test_eul_force(:,2:4,b);
%force_data= force_data(:,1:2:3,:);

% Main script for training and evaluating Continuous HMM with
    Gaussian emissions

% Standardize the force data
for i = 1:size(force_data, 3)
    force_data(:, :, i) = (force_data(:, :, i) - mean(force_data(:,
    :, i), 1)) ./ std(force_data(:, :, i), 0, 1);
end
% Standardize the force data
for i = 1:size(force_test_data, 3)
    force_test_data(:, :, i) = (force_test_data(:, :, i) - mean(
    force_test_data(:, :, i), 1)) ./ std(force_test_data(:, :, i), 0,
    1);
end

% Select the number of force components to use (1, 2, or 3)
num_force_components = 2; % Change this value to 1, 2, or 3 as needed

% Reduce the dimension of force_data if needed
if num_force_components < 3
    force_data = force_data(:, 1:num_force_components:3, :);
    force_test_data = force_test_data(:, 1:num_force_components:3, :)
    ;
end

% Training and test sets (already provided)
train_indices = 1:11;
test_indices = 12:14;
train_force_data = force_data;
test_force_data = force_test_data;
train_primitives_matrix = primitives_matrix;
test_primitives_matrix = primitives_matrix_test;

% Initialize HMM parameters
n_states = 3; % Number of hidden states
n_mix = 2; % Number of Gaussian mixtures per state
max_iter = 3000000; % Maximum number of iterations for EM algorithm

% Initialize variables to store HMM models and their log-likelihoods
hmm_models = cell(length(train_indices), 1);
log_likelihoods = zeros(length(train_indices), 1);

```

```
% Improved initialization
for demo = 1:length(train_indices)
    % Reshape data for continuous HMM
    reshaped_force_data = reshape(train_force_data(:, :, demo), [], num_force_components); % (440) x num_force_components
    reshaped_labels = train_primitives_matrix(:, demo); % 440 x 1

    % Initialize transition matrix and prior probabilities
    trans_guess = ones(n_states, n_states) / n_states; % Uniform transition probabilities
    trans_guess = trans_guess ./ sum(trans_guess, 2); % Normalize rows to sum to 1
    prior_guess = ones(n_states, 1) / n_states; % Initialize prior probabilities equally

    % Initialize Gaussian mixture components
    mu_guess = cell(n_states, 1);
    sigma_guess = cell(n_states, 1);
    mixmat_guess = ones(n_states, n_mix) / n_mix;

    for i = 1:n_states
        mu_guess{i} = randn(n_mix, num_force_components); % num_force_components is the dimension of the observations
        sigma_guess{i} = repmat(eye(num_force_components), [1, 1, n_mix]); % Identity matrices for covariances
        % Ensure positive definiteness
        for j = 1:n_mix
            sigma_guess{i}(:, :, j) = sigma_guess{i}(:, :, j) + 1e-6 * eye(num_force_components);
        end
    end

    % Train the continuous HMM with regularization
    [prior_est, trans_est, mu_est, sigma_est, mixmat_est, loglik] = train_chmm_em(reshaped_force_data, prior_guess, trans_guess, mu_guess, sigma_guess, mixmat_guess, max_iter);

    % Store the trained HMM model and its log-likelihood
    hmm_models{demo} = struct('prior', prior_est, 'trans', trans_est, 'mu', {mu_est}, 'sigma', {sigma_est}, 'mixmat', mixmat_est);
    log_likelihoods(demo) = loglik;
end

% Select the best HMM model based on the highest log-likelihood
[~, best_model_idx] = max(log_likelihoods);
best_hmm_model = hmm_models{best_model_idx};
all_states_pred=[];
% Evaluate the best HMM on the training set
train_accuracies = zeros(length(train_indices), 1);
```

```matlab
for demo = 1:length(train_indices)
    reshaped_force_data = reshape(train_force_data(:, :, demo), [],
    num_force_components); % (440) x num_force_components
    actual_states = train_primitives_matrix(:, demo); % 440 x 1

    % Predict states using the best HMM model with fixed constraints
    [predicted_states, ~] = predict_hmm_states_with_fixed_constraints
    (reshaped_force_data, best_hmm_model.prior, best_hmm_model.trans,
    best_hmm_model.mu, best_hmm_model.sigma, best_hmm_model.mixmat);
    all_states_pred = [all_states_pred, predicted_states];
    % Calculate accuracy for the current demonstration
    train_accuracies(demo) = sum(predicted_states == actual_states) /
     length(actual_states);
end

% Calculate overall training accuracy
overall_train_accuracy = mean(train_accuracies);
all_states_pred_test=[];
% Evaluate the best HMM on the test set
test_accuracies = zeros(length(test_indices), 1);
for demo_idx = 1:length(test_indices)
    reshaped_force_data = reshape(test_force_data(:, :, demo_idx),
    [], num_force_components); % (440) x num_force_components
    actual_states = test_primitives_matrix(:, demo_idx); % 440 x 1

    % Predict states using the best HMM model with fixed constraints
    [predicted_states, ~] = predict_hmm_states_with_fixed_constraints
    (reshaped_force_data, best_hmm_model.prior, best_hmm_model.trans,
    best_hmm_model.mu, best_hmm_model.sigma, best_hmm_model.mixmat);
    all_states_pred_test = [all_states_pred_test, predicted_states];
    % Calculate accuracy for the current demonstration
    test_accuracies(demo_idx) = sum(predicted_states == actual_states
    ) / length(actual_states);
end

% Calculate overall test accuracy
overall_test_accuracy = mean(test_accuracies);

% Display accuracies
disp('Training accuracies for each demonstration:');
disp(train_accuracies);
disp(['Overall training accuracy: ', num2str(overall_train_accuracy)
    ]);
disp('Test accuracies for each demonstration:');
disp(test_accuracies);
disp(['Overall test accuracy: ', num2str(overall_test_accuracy)]);

% Function definitions

```

```
function [nearest_time_idx, rel_time] = find_nearest_pose_time(
    demo_pose_data, current_pose)
    % Find the nearest pose in the demo_pose_data to the current_pose
    distances = sqrt(sum((demo_pose_data - current_pose).^2, 2));
    [~, nearest_time_idx] = min(distances);
    rel_time = nearest_time_idx / size(demo_pose_data, 1); %
    Calculate relative time
end

function [alpha, beta, gamma, xi] = forward_backward(obs, prior,
    trans, mu, sigma, mixmat)
    T = size(obs, 1); % Number of observations
    K = length(prior); % Number of states

    % Initialize alpha, beta, gamma, xi
    alpha = zeros(T, K);
    beta = zeros(T, K);
    gamma = zeros(T, K);
    xi = zeros(T, K, K);

    % Forward pass
    for k = 1:K
        alpha(1, k) = prior(k) * gmm_likelihood(obs(1, :), mu{k},
    sigma{k}, mixmat(k, :));
    end
    alpha(1, :) = alpha(1, :) / sum(alpha(1, :)); % Normalize

    for t = 2:T
        for j = 1:K
            sum_alpha = 0;
            for i = 1:K
                sum_alpha = sum_alpha + alpha(t-1, i) * trans(i, j);
            end
            alpha(t, j) = sum_alpha * gmm_likelihood(obs(t, :), mu{j
    }, sigma{j}, mixmat(j, :));
        end
        alpha(t, :) = alpha(t, :) / sum(alpha(t, :)); % Normalize

        % Apply state constraints based on known intervals
        if t <= 60
            alpha(t, 2:3) = 0; % Enforce state 1
        elseif t > 110 && t <= 250
            alpha(t, [1, 3]) = 0; % Enforce state 2
        elseif t > 360
            alpha(t, 1:2) = 0; % Enforce state 3
        end
        alpha(t, :) = alpha(t, :) / sum(alpha(t, :)); % Normalize
    end

    % Backward pass
    beta(T, :) = 1; % Initialize beta at T to 1

    for t = T-1:-1:1
        for i = 1:K
            sum_beta = 0;
            for j = 1:K
                sum_beta = sum_beta + trans(i, j) * gmm_likelihood(
    obs(t+1, :), mu{j}, sigma{j}, mixmat(j, :)) * beta(t+1, j);
            end
            beta(t, i) = sum_beta;
        end
        beta(t, :) = beta(t, :) / sum(beta(t, :)); % Normalize
    end

    % Compute gamma and xi
    for t = 1:T-1
        gamma(t, :) = alpha(t, :) .* beta(t, :);
        gamma(t, :) = gamma(t, :) / sum(gamma(t, :)); % Normalize

        for i = 1:K
            for j = 1:K
                xi(t, i, j) = alpha(t, i) * trans(i, j) *
    gmm_likelihood(obs(t+1, :), mu{j}, sigma{j}, mixmat(j, :)) * beta(
    t+1, j);
            end
        end
        xi(t, :, :) = xi(t, :, :) / sum(xi(t, :, :), 'all'); %
    Normalize
    end

    gamma(T, :) = alpha(T, :) .* beta(T, :);
    gamma(T, :) = gamma(T, :) / sum(gamma(T, :)); % Normalize
end

function likelihood = gmm_likelihood(x, mu, sigma, mixmat)
    % Compute the likelihood of x given the GMM parameters (mu, sigma
    , mixmat)
    n_mix = size(mu, 1);
    likelihood = 0;

    for j = 1:n_mix
        likelihood = likelihood + mixmat(j) * mvnpdf(x, mu(j, :),
    sigma(:, :, j));
    end
end

function [prior, trans, mu, sigma, mixmat, loglik] = train_chmm_em(
    obs, prior, trans, mu, sigma, mixmat, max_iter)
    K = length(prior); % Number of states
    T = size(obs, 1); % Number of observations

    loglik = -inf;
    for iter = 1:max_iter
        % E-step
        [alpha, beta, gamma, xi] = forward_backward(obs, prior, trans
    , mu, sigma, mixmat);

        % M-step
        prior = gamma(1, :)';
        trans = squeeze(sum(xi, 1));
        trans = trans ./ sum(trans, 2);

        for i = 1:K
            gamma_sum = sum(gamma(:, i));
            mu{i} = sum(gamma(:, i) .* obs) / gamma_sum;
            sigma{i} = zeros(size(sigma{i}));
            for t = 1:T
                sigma{i} = sigma{i} + gamma(t, i) * (obs(t, :) - mu{i
    })' * (obs(t, :) - mu{i});
            end
            sigma{i} = sigma{i} / gamma_sum;
            % Ensure positive definiteness
            for j = 1:size(sigma{i}, 3)
                sigma{i}(:, :, j) = sigma{i}(:, :, j) + 1e-6 * eye(
    size(sigma{i}, 1));
            end
        end

        % Compute log likelihood
        new_loglik = sum(log(sum(alpha .* beta, 2)));

        if abs(new_loglik - loglik) < 1e-9
            break;
        end
        loglik = new_loglik;
    end
end

function [predicted_states, log_prob] =
    predict_hmm_states_with_fixed_constraints(obs, prior, trans, mu,
    sigma, mixmat)
    T = size(obs, 1);
    K = length(prior);
    alpha = zeros(T, K);
    predicted_states = zeros(T, 1);

    % Forward pass
    for k = 1:K
        alpha(1, k) = prior(k) * gmm_likelihood(obs(1, :), mu{k}, sigma{k}, mixmat(k, :));
    end
    alpha(1, :) = alpha(1, :) / sum(alpha(1, :));

    for t = 2:T
        for j = 1:K
            sum_alpha = 0;
            for i = 1:K
                sum_alpha = sum_alpha + alpha(t-1, i) * trans(i, j);
            end
            alpha(t, j) = sum_alpha * gmm_likelihood(obs(t, :), mu{j}, sigma{j}, mixmat(j, :));
        end
        alpha(t, :) = alpha(t, :) / sum(alpha(t, :)); % Normalize

        % Apply state constraints based on known intervals
        if t <= 60
            alpha(t, 2:3) = 0; % Enforce state 1
        elseif t > 110 && t <= 210
            alpha(t, [1, 3]) = 0; % Enforce state 2
        elseif t > 350
            alpha(t, 1:2) = 0; % Enforce state 3
        end
        alpha(t, :) = alpha(t, :) / sum(alpha(t, :)); % Normalize
    end

    % Backtrace to find the most probable states
    [~, predicted_states(T)] = max(alpha(T, :));
    for t = T-1:-1:1
        [~, predicted_states(t)] = max(alpha(t, :) .* trans(:, predicted_states(t+1))');
    end

    log_prob = sum(log(sum(alpha, 2)));
end

function current_state = estimate_state_continuous_with_fixed_constraints(force_value, rel_time, prior_est, trans_est, mu_est, sigma_est, mixmat_est, previous_states_probs)
    n_states = length(prior_est);
    likelihoods = zeros(n_states, 1);

    for i = 1:n_states
        likelihoods(i) = gmm_likelihood(force_value, mu_est{i}, sigma_est{i}, mixmat_est(i, :));
    end

    if isempty(previous_states_probs)
        previous_states_probs = prior_est;
    end

    current_states_probs = trans_est' * (previous_states_probs .*
   likelihoods);
    current_states_probs = current_states_probs / sum(
   current_states_probs);

    % Enforce state constraints based on known intervals
    if rel_time <= 0.11
        current_states_probs(2:3) = 0; % Enforce state 1
    elseif rel_time > 0.22 && rel_time <= 0.45
        current_states_probs([1, 3]) = 0; % Enforce state 2
    elseif rel_time > 0.79
        current_states_probs(1:2) = 0; % Enforce state 3
    end
    current_states_probs = current_states_probs / sum(
   current_states_probs); % Normalize

    [~, current_state] = max(current_states_probs);
    previous_states_probs = current_states_probs;
end

function predicted_pose = estimate_pose(current_state, current_time,
   gmms)
    gmm = gmms{current_state};
    mu = gmm.mu;
    sigma = gmm.Sigma;

    mu_t = mu(:, 1);
    sigma_tt = sigma(1, 1, :);

    mu_p = mu(:, 2:end);
    sigma_pp = sigma(2:end, 2:end, :);

    sigma_tp = sigma(1, 2:end, :);

    % Normalize current time to [0, 1]
    norm_time = (current_time - 1) / (440 - 1);

    predicted_pose = zeros(1, size(mu_p, 2));
    for k = 1:size(gmm.ComponentProportion, 2)
        mu_p_k = mu_p(k, :)' + sigma_tp(:, :, k) / sigma_tt(:, :, k)
   * (norm_time - mu_t(k));
        predicted_pose = predicted_pose + gmm.ComponentProportion(k)
   * mu_p_k';
    end
```

```
end
```

Bibliography

[1] Harish Ravichandar, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. «Recent advances in robot learning from demonstration». In: *Annual review of control, robotics, and autonomous systems* 3 (2020), pp. 297–330 (cit. on pp. 4–7).

[2] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. «Survey: Robot programming by demonstration». In: *Springer handbook of robotics* (2008), pp. 1371–1394 (cit. on p. 7).

[3] Xing Li and Oliver Brock. «Learning from demonstration based on environmental constraints». In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 10938–10945 (cit. on p. 11).

[4] Enrico Turco, Valerio Bo, Mehrdad Tavassoli, Maria Pozzi, and Domenico Prattichizzo. «Learning Grasping Strategies for a Soft Non-Anthropomorphic Hand from Human Demonstrations». In: *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE. 2022, pp. 934–941 (cit. on pp. 12, 16).

[5] Alec Reed, Doncey Albin, Anuh Pasricha, Alessandro Roncone, and Christoffer Heckman. *Transformer-based Learning Models of Dynamical Systems for Robotic State Prediction*. Feb. 2024. DOI: `10.21203/rs.3.rs-3919154/v1` (cit. on p. 14).

[6] Gionata Salvietti, Muhammad Zubair Iqbal, Monica Malvezzi, Touraj Eslami, and Domenico Prattichizzo. «Soft Hands with Embodied Constraints: The Soft ScoopGripper». In: Mar. 2019. DOI: `10.1109/ICRA.2019.8793563` (cit. on p. 16).

[7] SCHUNK. *Gamma Force/Torque Sensor*. `https://schunk.com/it/it/tecnologia-di-automazione/sensore-di-forza/coppia/ft/c/PGR_1680`. Accessed: 2024-06-25 (cit. on p. 17).

[8] NVIDIA. *DRIVE OS Linux SDK Development Guide - Bootloader*. `https://docs.nvidia.com/drive/archive/drive_os_5.1.12.0L/nvvib_docs/DRIVE_OS_Linux_SDK_Development_Guide/Bootloader/robot_os.html`. Accessed: 2024-06-25 (cit. on p. 19).

[9] Sylvain Calinon. «A tutorial on task-parameterized movement learning and retrieval». In: *Intelligent service robotics* 9 (2016), pp. 1–29 (cit. on pp. 20, 31, 43, 47).

[10] Yanlong Huang, Leonel Rozo, João Silvério, and Darwin Caldwell. «Kernelized Movement Primitives». In: *The International Journal of Robotics Research* 38 (May 2019), pp. 833–852. DOI: `10.1177/0278364919846363` (cit. on p. 25).

[11] Pavel Senin. «Dynamic Time Warping Algorithm Review». In: (Jan. 2009) (cit. on p. 32).

[12] Najdan Vuković, Marko Mitić, and Zoran Miljković. «Trajectory learning and reproduction for differential drive mobile robots based on GMM/HMM and dynamic time warping using learning from demonstration framework». In: *Engineering Applications of Artificial Intelligence* 45 (2015), pp. 388–404 (cit. on pp. 32, 35, 43).

[13] Affan Pervez, Arslan Ali, Jee-Hwan Ryu, and Dongheui Lee. «Novel learning from demonstration approach for repetitive teleoperation tasks». In: *2017 IEEE World Haptics Conference (WHC)*. IEEE. 2017, pp. 60–65 (cit. on p. 32).

[14] Sebastian Thrun. «Probabilistic robotics». In: *Communications of the ACM* 45.3 (2002), pp. 52–57 (cit. on p. 35).

[15] Sebastian Thrun. «Probabilistic algorithms in robotics». In: *Ai Magazine* 21.4 (2000), pp. 93–93 (cit. on p. 35).

[16] Qiang Cheng, Wei Zhang, Hongshuai Liu, Ying Zhang, and Lina Hao. «Research on the path planning algorithm of a manipulator based on GMM/GMR-MPRM». In: *Applied Sciences* 11.16 (2021), p. 7599 (cit. on p. 35).

[17] Simge Nur Aslan, Recep Özalp, Ayşegül Uçar, and Cüneyt Güzeliş. «New CNN and hybrid CNN-LSTM models for learning object manipulation of humanoid robots from demonstration». In: *Cluster Computing* 25.3 (2022), pp. 1575–1590 (cit. on p. 35).

[18] Rouhollah Rahmatizadeh, Pooya Abolghasemi, and Ladislau Bölöni. «Learning manipulation trajectories using recurrent neural networks». In: *arXiv preprint arXiv:1603.03833* (2016) (cit. on p. 35).

[19] Leonel Rozo, Pablo Jiménez, and Carme Torras. «Robot learning from demonstration of force-based tasks with multiple solution trajectories». In: *2011 15th International Conference on Advanced Robotics (ICAR)*. IEEE. 2011, pp. 124–129 (cit. on pp. 35, 36).

[20] Sylvain Calinon, Florent D'halluin, Eric L Sauser, Darwin G Caldwell, and Aude G Billard. «Learning and reproduction of gestures by imitation». In: *IEEE Robotics & Automation Magazine* 17.2 (2010), pp. 44–54 (cit. on p. 35).

[21] S. Calinon, F. Guenter, and A. Billard. «On Learning, Representing and Generalizing a Task in a Humanoid Robot». In: *IEEE Transactions on Systems, Man and Cybernetics, Part B* 37.2 (2007), pp. 286–298 (cit. on pp. 35, 48).

[22] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. «Robot learning from demonstration by constructing skill trees». In: *The International Journal of Robotics Research* 31.3 (2012), pp. 360–375 (cit. on p. 36).

[23] Simon Manschitz, Jens Kober, Michael Gienger, and Jan Peters. «Learning to sequence movement primitives from demonstrations». In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 4414–4421 (cit. on p. 36).

[24] Peter Kazanzides, Joel F Zuhars, Brent D Mittelstadt, and Russell H Taylor. «Force sensing and control for a surgical robot.» In: *ICRA*. 1992, pp. 612–617 (cit. on p. 37).

[25] Daniel Jurafsky James H. Martin. «9 Hidden Markov Models». In: (2024). URL: `https://web.stanford.edu/~jurafsky/slp3/A.pdf` (cit. on p. 38).

[26] S. Calinon. *Robot Programming by Demonstration: A Probabilistic Approach*. EPFL Press ISBN 978-2-940222-31-5, CRC Press ISBN 978-1-4398-0867-2. EPFL/CRC Press, 2009 (cit. on pp. 39, 48).

[27] Michael Seifert, Ali Mohammad Banaei-Moghaddam, Jens Keilwagen, Michael Mette, Andreas Houben, Francois Roudier, Vincent Colot, Ivo Grosse, and Marc Strickert. «Array-based Genome Comparison of Arabidopsis Ecotypes using Hidden Markov Models.» In: Jan. 2009, pp. 3–11 (cit. on p. 39).

[28] S. Calinon and D. Lee. «Learning Control». In: *Humanoid Robotics: a Reference*. Ed. by P. Vadakkepat and A. Goswami. Springer, 2019, pp. 1–52. DOI: `10.1007/978-94-007-7194-9_68-1` (cit. on pp. 39, 42).

[29] Patrick Verlinde. «Error Detecting and Correcting Codes». In: *Encyclopedia of Information Systems*. Ed. by Hossein Bidgoli. New York: Elsevier, 2003, pp. 203–228. ISBN: 978-0-12-227240-0. DOI: `https://doi.org/10.1016/B0-12-227240-4/00062-9`. URL: `https://www.sciencedirect.com/science/article/pii/B0122272404000629` (cit. on p. 42).

[30] Sonia Chernova and Andrea L. Thomaz. «Learning Low-Level Motion Trajectories». In: *Robot Learning from Human Teachers*. Cham: Springer International Publishing, 2014, pp. 25–35. ISBN: 978-3-031-01570-0. DOI: `10.1007/978-3-031-01570-0_4`. URL: `https://doi.org/10.1007/978-3-031-01570-0_4` (cit. on p. 43).

[31] Meng Xiao, Xuefei Zhang, Tie Zhang, Shouyan Chen, Yanbiao Zou, and Wen Wu. «A study on robot force control based on the GMM/GMR algorithm fusing different compensation strategies». In: *Frontiers in Neurorobotics* 18 (2024), p. 1290853 (cit. on pp. 43, 45).

[32] S Preethi and B Arivu Selvam. «Automatic speech recognition system for real time applications». In: *International Journal of Engineering Innovations and Research* 2.2 (2013), p. 157 (cit. on p. 43).

[33] Praat. *Expectation-Maximization*. `https://www.fon.hum.uva.nl/praat/manual/expectation-maximization.html`. Accessed: 2024-06-23 (cit. on p. 44).

[34] Douglas A Reynolds. «Speaker identification and verification using Gaussian mixture speaker models». In: *Speech communication* 17.1-2 (1995), pp. 91–108 (cit. on p. 44).

[35] Manan Vyas. «A Gaussian mixture model based speech recognition system using Matlab». In: *Signal & Image Processing* 4.4 (2013), p. 109 (cit. on p. 44).

[36] David KY Chiu. «Book review:" Pattern classification", RO Duda, PE Hart and DG Stork». In: *International Journal of Computational Intelligence and Applications* 1.03 (2001), pp. 335–339 (cit. on p. 44).

[37] Sylvain Calinon, Florent Guenter, and Aude Billard. «On learning, representing, and generalizing a task in a humanoid robot». In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37.2 (2007), pp. 286–298 (cit. on p. 45).

[38] Noémie Jaquier, David Ginsbourger, and Sylvain Calinon. «Learning from demonstration with model-based Gaussian process». In: *Conference on Robot Learning*. PMLR. 2020, pp. 247–257 (cit. on p. 45).

[39] Luka Peternel, Nikos Tsagarakis, Darwin Caldwell, and Arash Ajoudani. «Robot adaptation to human physical fatigue in human–robot co-manipulation». In: *Autonomous Robots* 42 (2018), pp. 1011–1021 (cit. on p. 45).

[40] S. Calinon. «A Tutorial on Task-Parameterized Movement Learning and Retrieval». In: *Intelligent Service Robotics* 9.1 (2016), pp. 1–29. DOI: 10.1007/s11370-015-0187-9 (cit. on pp. 46, 47).

[41] Florence Babatunde, Bolanle Ojokoh, and Samuel Oluwadare. «Automatic Table Recognition and Extraction from Heterogeneous Documents». In: *Journal of Computer and Communications* 03 (Jan. 2015), pp. 100–110. DOI: 10.4236/jcc.2015.312009 (cit. on p. 54).

[42] Ovidiu Boitor, Florin Stoica, Romeo Mihăilă, Laura Florentina Stoica, and Laura Stef. «Automated machine learning to develop predictive models of metabolic syndrome in patients with periodontal disease». In: *Diagnostics* 13.24 (2023), p. 3631 (cit. on p. 55).

[43] Joao Batista Rocha Bezerra Junior. «Overcoming Imbalanced Class Distribution and Overfitting in Financial Fraud Detection: An Investigation Using A Modified Form of K-Fold Cross Validation Approach to Reach Representativeness». PhD thesis. 2023 (cit. on p. 55).

[44] Hang Su, Andrea Mariani, Salih Ertug Ovur, Arianna Menciassi, Giancarlo Ferrigno, and Elena De Momi. «Toward teaching by demonstration for robot-assisted minimally invasive surgery». In: *IEEE Transactions on Automation Science and Engineering* 18.2 (2021), pp. 484–494 (cit. on pp. 69, 70).

[45] Amritpal Singh, Wenqi Shi, and May D Wang. «Autonomous Soft Tissue Retraction Using Demonstration-Guided Reinforcement Learning». In: *arXiv preprint arXiv:2309.00837* (2023) (cit. on p. 70).

Acknowledgements

Desidero ringraziare il Professore Alessandro Rizzo e il Professore Domenico Prattichizzo, per avermi dato l'opportunità di misurarmi in un contesto di alto livello. Ringrazio l'Ing. Enrico Turco per essere stato una guida disponibile e amichevole in questo percorso di tesi, insieme a tutto il gruppo di ragazzi fantastici con cui ho avuto modo di confrontarmi. Ringrazio i miei amici da sempre e per sempre:Ciccio,Vicio,Dani,Dami,Giachi,Giando, Edo,che anche se spesso da lontano, hanno contribuito a spingermi verso questo traguardo.

Ringrazio Lorenzo, Ludovica e Lollo, amici incontrati durante questo lungo percorso,che lo hanno reso un po' meno duro di quello che sarebbe stato.

Ringrazio i miei coinquilini di questi anni, diventati una seconda famiglia, Richi,Ale,Marti,Ori.

Un ringraziamento speciale va al mio, ormai fratello acquisito, Nando. Abbiamo condiviso ogni momento di questi anni, la sofferenza e le gioie ci hanno unito in un legame spero indissolubile.

Ringrazio la mia famiglia allargata, Zia Sabina, Zio Mario, Chiara e Robi. Siete sempre stati al mio fianco nei momenti peggiori e in quelli migliori. Rappresentate per me famiglia, casa e ristoro e spero lo sarete sempre.

Ringrazio te Nonna, la persona più genuina che conosca, che mi ha cresciuto e a cui voglio un bene enorme anche se non lo dimostro. Sei l'unico baluardo rimasto a ricordarci le nostre radici.

Ringrazio la persona che ,anche quando tutto avrebbe indicato la via di fuga, è rimasta al mio fianco, insegnandomi ad amarmi e ad amare.Spero che continuerai a essere il mio Faro nel buio e che anche io lo sarò per te, Alessia.

Ringrazio la mia Famiglia, che mi ha sostenuto, rialzato, spinto, ognuno a suo modo. Questo percorso ha messo alla prova il nostro legame, senza mai distruggerlo. Spero di diventare l'uomo che sperate io sia, felice, sereno e capace di ridarvi un po di quello che mi avete donato. Le parole sarebbero troppe da racchiudere in questo piccolo spazio, ma sappiate che vi Amo tutti.

Infine due ringraziamenti speciali... Uno a chi purtroppo non può vedere e gioire con me di questo sofferto traguardo. A voi Nonni, ognuno dei quali mi ha trasmesso un pezzo di sè. In particolare a te Nonno Vittorio, guida, colonna, esempio per tutti noi. Con il tempo mi sono reso conto di quanto di te ci sia in me, delle tue passioni, della tua curiosità, del tuo modo di essere. Mi hai guidato da lontano e mi sono spesso rivolto al tuo ricordo per rialzarmi. Spero di diventare un decimo dell'uomo che eri per tutti noi e spero di averti reso orgoglioso.

A me, per avere lottato contro ogni ostacolo, interiore e esteriore per dimostrarmi di potercela fare. Questo percorso per me rappresenterà un capitolo che mi ha forgiato, lasciandomi al tappeto per tanto tempo, facendomi capire quanta voglia ho di vivere e di dare. Non so dove mi porterà il futuro, ma so che mollare non è un'opzione.

Gabriele