POLITECNICO DI TORINO

MASTER DEGREE THESIS

Department of Control and Computer Engineering

Mechatronic Engineering

*a.y. 2023/2024*

# Predictive algorithm and guidance for human motion in robotics teleoperation

*Supervisors:*

Alessandro RIZZO
(*Politecnico di Torino*)

Domenico PRATTICHIZZO
(*Università di Siena*)

Enrico TURCO
(*Istituto Italiano di Tecnologia*)

Valerio BO
(*Istituto Italiano di Tecnologia*)

*Candidate:*

Francesco STOLCIS

March 2024

POLITECNICO DI TORINO

# *Abstract*

Department of Control and Computer Engineering

Mechatronic Engineering

**Predictive algorithm and guidance for human motion in robotics teleoperation**

by Francesco STOLCIS

This thesis proposes a novel approach to Shared Control Architecture in the context of the teleoperation of a robotic arm, with the primary objective being the development of a robust system capable of recognizing the operator's intention and smoothly guiding him toward predicted objects. To achieve this goal, the framework employs an Intent Recognition Model comprising of multiple Long Short-Term Memory (LSTM) classification models, able to recognize the goal object chosen by the operator.
Furthermore, haptic guidance is provided to the operator through the use of Artificial Potential Fields, manoeuvring the haptic controller in the direction of the predicted object.
By combining the predictive power of LSTM with the reactive nature of potential fields, the system achieves a seamless fusion of human expertise and autonomous control, ensuring both efficiency and safety in teleoperation tasks.
The proposed approach is implemented and validated through comprehensive simulations and real-world experiments using a teleoperated robotic arm platform and a UR5 Universal Robot. Performance evaluations demonstrate the effectiveness and reliability of the shared control system, showcasing precise human intent recognition, improved overall execution time, scalability of the model and adaptability to new scenarios. Additionally, the system is customizable within limitations, making it tailor-made for any complex scenario.

# Contents

**Abstract** **i**

**1 Introduction** **1**
1.1 Thesis Objective . . . . . . . . . . 1
1.2 Thesis structure . . . . . . . . . . 2

**2 State of the Art** **3**
2.1 Teleoperation . . . . . . . . . . 3
2.1.1 Control Architectures . . . . . . . . . . 4
2.1.2 Shared Control . . . . . . . . . . 5
2.2 Intention Recognition . . . . . . . . . . 6
2.2.1 Methodologies . . . . . . . . . . 7
2.3 Haptic Guidance . . . . . . . . . . 9
2.3.1 Methodologies . . . . . . . . . . 10

**3 Materials** **12**
3.1 Hardware . . . . . . . . . . 12
3.1.1 Universal Robot UR5 . . . . . . . . . . 12
3.1.2 Omega 7 haptic device . . . . . . . . . . 13
3.1.3 qb SoftHand . . . . . . . . . . 14
3.1.4 Kinect v2 . . . . . . . . . . 15
3.2 Software . . . . . . . . . . 16
3.2.1 ROS . . . . . . . . . . 16
3.2.2 Coppeliasim . . . . . . . . . . 16
3.2.3 Programming Language . . . . . . . . . . 17
3.2.4 Libraries . . . . . . . . . . 17
3.3 Teleoperation System . . . . . . . . . . 19

**4 Methods** **20**
4.1 Scenario . . . . . . . . . . 20
4.2 Human Intention Recognition . . . . . . . . . . 24
4.2.1 Recurrent Neural Networks . . . . . . . . . . 24
Vanishing Gradient Problem . . . . . . . . . . 27
4.2.2 Long Short-Term Memory (LSTM) Networks . . . . . . . . . . 28
4.2.3 LSTM Intent Recognition Architecture . . . . . . . . . . 31
4.2.4 Training the model . . . . . . . . . . 34
4.3 Guidance System for Teleoperation . . . . . . . . . . 35
4.3.1 Artificial Potential Fields . . . . . . . . . . 35
4.3.2 Haptic Guidance . . . . . . . . . . 36
4.4 Teleoperation System with Shared Control . . . . . . . . . . 37

**5 Implementation and Results** **39**

5.1 Virtual Experimental Setup . . . . . 39
5.1.1 Evaluation Prediction without guidance . . . . . 40
5.1.2 Evaluation Prediction with guidance . . . . . 41
5.1.3 Skilled operator with guidance . . . . . 43
Dynamic Scenario . . . . . 43
5.2 Real Experimental Setup . . . . . 45

**6 Discussion and Future Works** **49**

**A Support Code** **50**
A.1 LSTM model . . . . . 50

**Bibliography** **51**

**Acknowledgements** **54**

# List of Figures

2.1 telerobotic system . . . 4
2.2 Types of Shared Control [19] . . . 6
2.3 Representation of a Hidden Markov Model (HMM) . . . 7

3.1 Universal Robot Ur5 . . . 12
3.2 Control Unit Ur5 . . . 13
3.3 Teach Pendant Ur5 . . . 13
3.4 Omega.7 Haptic Device . . . 14
3.5 qb SoftHand Research . . . 15
3.6 Kinect 2 for windows . . . 15
3.7 Teleoperation System . . . 19

4.1 Virtual Scenario. The object labeling, from left to right, is ′2′,′0′,′3′,′1′ . 20
4.2 End effector position with respect to the objects . . . 21
4.3 Hand reference system . . . 22
4.4 Direction of end effector with respect to each object (in degrees) . . . 23
4.5 Distance of end effector with respect to each object . . . 23
4.6 Recursive Neural Network (RNN) . . . 24
4.7 Recursive Neural Network single layer . . . 25
4.8 The sigmoid function and its derivative . . . 28
4.9 LSTM Cell Structure . . . 29
4.10 Complete Intent Recognition Model . . . 31
4.11 Single Intention Recognition Model . . . 32
4.12 Effect of Dropout layer. [4] . . . 33
4.13 Online Architecture with Shared control implementation . . . 38

5.1 Virtual Trials Scenario . . . 39
5.2 Virtual Trials Scenario . . . 42
5.3 Predictions and Object movement along $xy$ plane . . . 44
5.4 Real Experimental Setup.Ur5 robotic arm (1), Kinect Two RGB-D camera (2) and the Objects are indicated with IDs . . . 45
5.5 Visual real camera and bounding boxes . . . 46
5.6 Point Cloud and bounding boxes (represented in the World reference frame) . . . 46
5.7 Predictions . . . 47
5.8 End effector and Object, $x$ axes (world frame) . . . 47
5.9 End effector and Object, $y$ axes (world frame) . . . 47
5.10 End effector and Object, $z$ axes (world frame) . . . 48

# List of Tables

3.1 Omega.7 Characteristics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

5.2 Prediction with no guidance (Training Scenario) . . . . . . . . . . . . . . 40
5.4 Prediction with no guidance (Random Scenario) . . . . . . . . . . . . . . 40
5.5 True Positive Rate of each model . . . . . . . . . . . . . . . . . . . . . . . . 41
5.6 True Positive Rate for 40 timestamp Model . . . . . . . . . . . . . . . . . 41
5.7 Average Percentage of Each Row . . . . . . . . . . . . . . . . . . . . . . . 43
5.8 Properties of tested objects. . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

# List of Abbreviations

| | |
|---|---|
| **OCU** | **O**perator **C**ontrol **U**nit |
| **SAC** | **S**emi-**A**utonomous **C**ontrol |
| **SGSC** | **S**tate-**G**uidance **S**hared **C**ontrol |
| **SFSC** | **S**tate-**F**usion **S**hared **C**ontrol |
| **HMM** | **H**idden **M**arkov **M**odel |
| **IRL** | **I**nverse **R**einforcement **L**earning |
| **POMDP** | **P**artially **O**bservable **M**arkov **D**ecision **P**rocess |
| **RNN** | **R**ecurrent **N**eural **N**etwork |
| **LSTM** | **L**ong **S**hort **T**erm **M**emory |
| **APF** | **A**rtificial **P**otential **F**ields |
| **DWA** | **D**ynamic **W**indow **A**pproach |
| **RRT** | **R**apidly-exploring **R**andom **T**rees |
| **PPA** | **P**ure **P**ursuit **A**lgorithm |
| **VFH** | **V**ector **F**ield **H**istogram |
| **BDWA** | **B**iomedical **D**ynamic **W**indow **A**pproach |
| **CPU** | **C**entral **P**rocessing **U**nit |
| **ROS** | **R**obot **O**perating **S**ystem |
| **KDL** | **K**inematics and **D**ynamics **L**ibrary |
| **BPTT** | **B**ack**P**propagation **T**hrough **T**ime |
| **SGD** | **S**tochastic **G**radient **D**escend |
| **IRM** | **I**ntention **R**ecognition **M**odel |
| **YCB** | **Y**ale-**C**MU-**B**erkeley |

# Chapter 1

# Introduction

## 1.1 Thesis Objective

In recent years, the concept of "shared control" has emerged as a fundamental paradigm across various domains, ranging from computer science to engineering, robotics to psychology. This shift reflects a fundamental transition in how humans and machines collaborate to accomplish tasks, marked by an increasing recognition of the synergy between human expertise and autonomous capabilities.
Shared control involves a collaborative model wherein both human operators and automated systems share responsibilities, decision-making, and control authority. This collaborative approach has been particularly transformative in the development of autonomous systems and robotics, where human-robot interaction demands seamless integration of human intent with machine precision.

Within the context of robotics, shared control architecture has found significant application in the teleoperation of robotic arms.
Robotic arms are indispensable tools in various industries, including manufacturing, healthcare, and space exploration, where they perform tasks ranging from assembly and manipulation to surgical procedures and maintenance.
However, operating robotic arms remotely presents unique challenges, including complex manoeuvrability and excessive workload on the operator.

Various approaches have been explored to address these challenges and facilitate effective, shared control in the teleoperation of robotic arms.
In traditional shared control methods, robotic arms are guided through a combination of error feedback regulation and collision-free trajectory planning. Techniques such as adjusting control signals based on error feedback ensure precise motion, while path planning algorithms compute optimal routes to reach targets while avoiding obstacles, while cost functions further optimize decision-making processes, balancing factors like task completion time and safety to enhance overall system performance in teleoperation scenarios.
Nevertheless, traditional shared control methods have limitations in adaptability and scalability. With the advent of deep machine learning, new horizons have opened for developing lightweight, versatile, and robust systems. Deep learning models offer the potential to substantially improve shared control architectures by providing adaptability to diverse contexts, reducing manual tuning, and enhancing real-time decision-making capabilities.

This thesis aims to implement a novel approach to Shared Control architecture, utilizing deep learning techniques. Primarily, it seeks to develop a more generalized and fully scalable Intention Recognition model capable of precisely recognizing the

operator's intent in a broad variety of scenarios. Additionally, the guidance system is engineered to facilitate expedited goal attainment for the human operator while concurrently mitigating a portion of their cognitive workload.
By incorporating deep learning into shared control architectures, this thesis seeks to overcome the limitations of traditional methods and advance the field of teleoperation in robotics.

## 1.2 Thesis structure

The thesis is structured into five chapters:

**Chapter 2** *State-of-art*:
This chapter extensively explores teleoperation systems, beginning with a comprehensive overview of the field's foundational concepts. It delves into the operational framework of teleoperation systems, elucidating key components. Furthermore, it discusses various control architectures prevalent in teleoperation, ranging from direct control to shared control paradigms. Additionally, the chapter explores intention recognition methodologies within the context of shared control systems. It concludes with a detailed investigation of haptic guidance methodologies, highlighting techniques and showcasing their roles in enhancing human-robot interactions in teleoperation environments.

**Chapter 3** *Materials*:
This chapter provides an in-depth exploration of materials essential for teleoperation systems, commencing with a detailed examination of foundational hardware components. It describes the operational architecture of teleoperation setups, delineating critical elements such as robotic arms, haptic devices, and sensing technologies. Furthermore, it delves into the software infrastructure underpinning teleoperation. Additionally, the chapter explores the programming languages and libraries employed in teleoperation system development.

**Chapter 4** - *Methods*:
This chapter delves into the core algorithms and techniques of the methodology. It outlines the operational environment for the shared control architecture and elucidates the characteristics of the input data. Furthermore, it presents both the theoretical and practical aspects of the chosen methodology for intent recognition. Similarly, the guidance system is delineated in detail. Concluding with a schematic detailing the integration of the shared control architecture within the teleoperation system.

**Chapter 5** - *Implementation and Results*:
This chapter presents the experimental setup from both virtual and real-world scenarios. Overall, it provides a comprehensive exploration of the research's implementation and experimentation phases.

**Chapter 6** - *Discussions and Future Works*:
The chapter explores the achieved goals of the architecture and discusses potential new refinements and research possibilities.

# Chapter 2

# State of the Art

This thesis project focuses on developing a shared control framework applied to a teleoperation system involving a robotic arm. To this aim, this chapter provides an overview of the state of the art of the approaches and ideas employed in the following chapters. Three main areas are covered: teleoperation systems, intention recognition, and haptic guidance strategies.

## 2.1 Teleoperation

Teleoperation, as described in [27], is a field that combines robotics, control systems, and telecommunications, enabling a human operator to remotely control a machine or robot from a distance, irrespective of the distance being significantly minimal or extensively significant. It involves the transmission of control signals from the operator to the robot and the feedback of sensory information from the robot back to the operator. This bidirectional flow of information allows the operator to execute tasks and make decisions based on real-time data, even when physically separated from the machine.

The overall operational framework of teleoperation systems is represented by several key components, as shown in Fig. 2.1:

- **The operator control unit (OCU):** it serves as the interface through which the human operator interacts with the teleoperation system, typically featuring control devices like joysticks, gloves, or even more sophisticated haptic devices that provide tactile feedback, simulating the sensation of touch. This feedback is crucial, as it enhances the operator's ability to perform complex manipulations remotely by offering a semblance of physical presence at the teleoperator's location.
- **The teleoperator:** equipped with various sensors, actuators, and sometimes autonomous control capabilities, it executes the tasks directed by the operator. These tasks can range from simple pick-and-place operations to intricate surgical procedures, depending on the system's design and intended application. The sophistication of the teleoperator is a critical factor in the system's overall performance, particularly its ability to execute precise movements and provide feedback that accurately reflects the remote environment.
- **The communication link:** it bridges the OCU and the teleoperator, and it can be wired or wireless, depending on the application's requirements and constraints. This link transmits control signals from the operator to the teleoperator and sends sensory feedback (visual, haptic, etc.) back to the operator. The quality of

this link, in terms of latency, bandwidth, and reliability, significantly impacts the system's effectiveness and the operator's ability to control the teleoperator accurately and responsively.

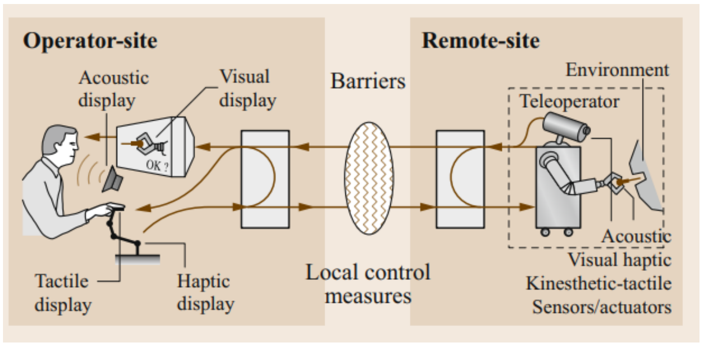


FIGURE 2.1: Overview of a telerobotic system [27]

### 2.1.1 Control Architectures

In teleoperation systems, a wide spectrum of control levels exists, spanning from direct controls to more intricate ones that incorporate algorithms designed to assist the human operator. The most notable are:

- **Direct Control:** it is the simplest form of teleoperation, where every action taken by the human operator is directly translated into movements or actions of the teleoperator. This one-to-one correspondence mean that the operator has immediate and complete control over the teleoperator's actions.
  Direct control systems are highly dependent on the operator's skill and attention, as there is minimal to no automation involved.
  This architecture is most effective in environments where precise, real-time control is necessary, and the task complexity is manageable by the human operator without additional automated support.

- **Supervisory Control:** it takes a step further in integrating automation into teleoperation. In this architecture, the human operator takes on a supervisory role, monitoring the operation and intervening only when necessary. The teleoperator, equipped with a higher degree of autonomy, can perform tasks independently based on pre-programmed instructions or algorithms.
  The operator's primary responsibilities include setting goals, specifying constraints, monitoring progress, and intervening in case of unexpected events or system failures. Supervisory control is well-suited for operations in highly dangerous or inaccessible environments, or for tasks that require extended periods, where continuous direct human control is impractical.

- **Shared Control:** it represents a more advanced teleoperation approach, where control is divided between the human operator and autonomous system functions [15].
  In this architecture, the operator and the teleoperation system share the task of controlling the teleoperator. The distribution of control can be dynamic, with the level of automation adjusting based on the task complexity, operator workload, or other contextual factors.
  Shared control systems are designed to combine human cognitive abilities with the precision and reliability of automated systems, optimizing task performance

and potentially reducing operator fatigue. This approach is particularly beneficial in complex or unpredictable environments, where human judgment is essential, but so is the efficiency and consistency of automated processes.

### 2.1.2 Shared Control

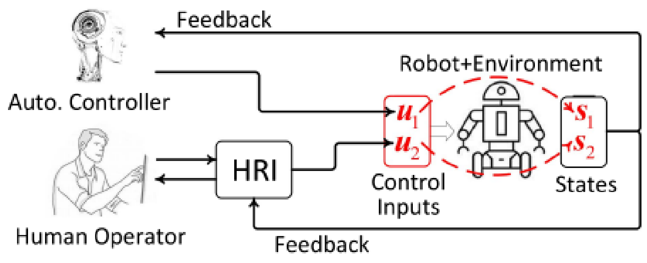
As mentioned before, a shared control teleoperation architecture can be ideal to combine human cognitive capabilities with robotic efficiency to reduce the operator's workload and enhance task performance.
More specifically, there are three main ways to implement the shared control architecture (see Fig. 2.2): Semi-Autonomous Control (SAC), State-Guidance Shared Control (SGSC), and State-Fusion Shared Control (SFSC).

- **Semi-Autonomous Control (SAC):** In SAC the division of control tasks between the human operator and the autonomous system is distinctly demarcated.
  In this approach, the autonomous system and the human operator control separate variables, allowing each to leverage their strengths for improved task execution. This delineation enables the human operator to focus on high-level decision-making and strategic task elements while the autonomous system manages the execution of routine or precise actions.
  The SAC model is particularly beneficial, as shown in [19], in complex and dynamic environments where human intuition and strategic oversight are crucial for success but are complemented by the precision and reliability of autonomous robotic actions. It facilitates a collaborative interaction where the operator's cognitive load is reduced, and the efficiency and safety of operations are enhanced by offloading specific control tasks to the robot.

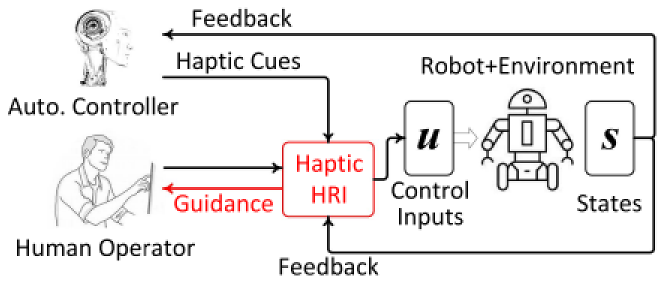
- **State-Guidance Shared Control (SGSC):** In SGSC the autonomous system provides direct guidance to the human operator, which can take various forms such as visual, auditory, or haptic feedback. This guidance is designed to assist the operator in making more informed decisions or in executing tasks with greater precision.
  The key principle behind SGSC is to enhance the operator's situational awareness and to facilitate the decision-making process without overtaking the human's control. This is achieved by allowing the autonomous system to suggest or warn the operator based on the system's perception and analysis of the environment or the task at hand. SGSC strategies are particularly useful in scenarios where human intuition and judgment are critical, but where the complexity or danger of the task necessitates additional inputs that can augment human capabilities. By providing guidance, SGSC aims to reduce the cognitive load on the operator, improve task performance, and enhance safety without diminishing the human's role in the control loop.

- **State-Fusion Shared Control (SFSC):** In SFSC, the strategy goes beyond merely dividing tasks or providing guidance; it integrates the intentions and control signals of the human operator and the robot through a fusion mechanism. The fusion process often involves sophisticated algorithms that can weigh the contributions of the human and the robot based on factors like task complexity, operator expertise, and system capabilities.
  The goal of SFSC is to harness the strengths of both human and machine: the adaptability, judgment, and intuition of the human, alongside the precision, reliability, and speed of the machine. This integrated approach allows for a more

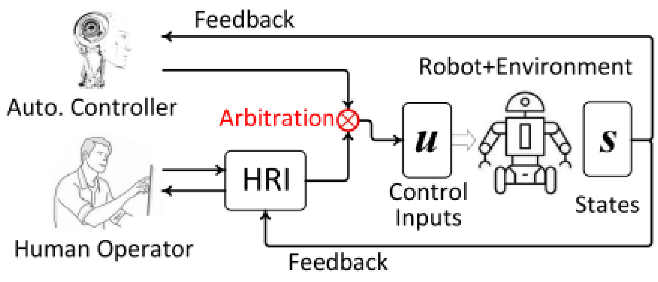
seamless and efficient execution of tasks, particularly in environments that are too complex for fully autonomous robotics or require a level of decision-making beyond current autonomous capabilities. By dynamically adjusting the level of influence between human and machine based on real-time feedback and task requirements, SFSC strategies ensure that the control system remains adaptable and responsive to the nuances of each specific operation. This not only improves the performance and safety of telerobotic systems but also enhances the user experience by creating a more intuitive interaction between the human operators and the robotic system.



(A) Semi-Autonomous Control



(B) State-Guidance Shared Control



(C) State-Fusion Shared Control

FIGURE 2.2: Types of Shared Control [19]

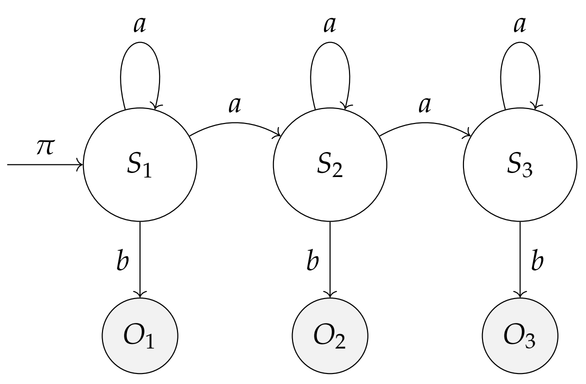
## 2.2 Intention Recognition

Intent Recognition, within the framework of a shared control system, analyzes the operator's control inputs, delivered through controlling devices, and identifies their underlying intentions. The core challenge of intention recognition lies in accurately inferring the user's goals from a set of possible actions, often in real-time and under uncertainty.

### 2.2.1 Methodologies

Intention recognition methodologies span a range of approaches, reflecting the diversity of applications and the complexity of interpreting human intent.
At this day, there isn't a single, universally accepted method for intention recognition that applies across all scenarios in the teleoperation of robotic arms. Instead, the field comprises a variety of techniques, each with its strengths and limitations, tailored to specific contexts, tasks, and environments

- **Probabilistic Models:** Hidden Markov Models (HMMs), as extensively explained in [5] and [1], is a statistical model that represents systems with probabilistic properties. It is characterized by its ability to model systems where the states are hidden and not directly observable, but the outcomes or observations associated with these states are observable.
  As shown in Fig. 2.3 an HMM is defined by three primary elements: the state transition probability matrix **A**, which specifies the likelihood of transitioning from one state to another; the observation probability matrix **B**, detailing the probability of observing a certain symbol when in a specific state; and the initial state probability vector $\boldsymbol{\pi}$, indicating the likelihood of the system starting in each state.



**Legend:**
S: State
O: Observation
$a$: Transition Probability
$b$: Observation Probability
$\pi$: Initial State Probability

FIGURE 2.3: Representation of a Hidden Markov Model (HMM)

As outlined in [18], while the application of HMMs in intention recognition for teleoperated robotic arm manipulation offers notable advantages in handling temporal dynamics and model flexibility, it concurrently presents substantial challenges, particularly in the necessity for precise and well-documented a priori knowledge. The complex nature of human decision-making processes makes difficult the task of accurately predicting operator behavior at each timestep of the teleoperation task; this is evident in the methodology, where the transition matrix **A** is constructed based on empirical observations, needing a large batch of different teleoperators do be determined. This approach results in a relatively simplistic policy for state transitions, potentially oversimplifying the nuanced and variable nature of human intent during teleoperation tasks.

- **Inverse Reinforcement Learning (IRL):** Inverse Reinforcement Learning (IRL), as detailed in [11], is a sophisticated method for deciphering and forecasting the intentions behind human operators' actions in teleoperation, where understanding complex decision-making is key to enhancing human-robot interactions.
  Ziebart et al. [34] extend IRL's capabilities through a probabilistic framework grounded in the maximum entropy principle, adept at navigating the uncertainties in teleoperation decision-making. This approach, by favoring the least biased action distribution based on observed behaviors, provides a refined representation of operators' reward functions, thus accurately modeling their intent.
  The incorporation of maximum entropy IRL into intent recognition significantly improves how robotic systems can preemptively adjust to and fulfill human operator goals, shifting from merely reactive to proactive adaptations. These advantages are, however, constrained by the computational burden encountered during the training phase, which is characterized by slow convergence. Due to their complexity, real-time applications are generally not well-suited.

- **Partially Observable Markov Decision Processes (POMDPs):** While HMMs are used for modeling and inferring hidden state sequences based on observations, without a focus on decision-making; POMDPs, [31], integrate the aspect of decision-making under uncertainty. Partially Observable Markov Decision Process are a class of decision-making models that consider situations where the state of the system is partially observable or uncertain to the decision-maker. In a POMDP framework, the decision-maker must rely on observations that may provide indirect or incomplete information about the system's state to make decisions. This model extends the Markov Decision Process (MDP) framework by incorporating a layer of uncertainty regarding the system's actual state, making it particularly suited for complex environments where full information is not available.
  In the context of shared control systems, POMDPs play a crucial role in modeling the interaction between a human user and an autonomous system, especially when the system's understanding of the user's intent is uncertain. The authors in [16] delve into this by formalizing shared autonomy as a POMDP, which assists in minimizing the expected cost-to-go with an unknown goal. This approach acknowledges that while the autonomous system may not confidently predict the user's specific goal until it is nearly achieved, it can still offer valuable assistance by optimizing actions that are generally helpful across multiple potential goals. When the system has high confidence in a single user goal, the framework focuses assistance more narrowly to support that specific objective. This balance allows for more effective and adaptive assistance, even in the face of uncertainty about the user's exact intentions.
  Despite its great ability to handle uncertainty, give focused help based on goals, and use hindsight optimization to make real-time processing better, this method still encounters problems with how complex its computations are, requiring substantial computational resources to function effectively. Additionally, the performance of this approach is deeply interconnected with the precision of its observational models for accurately perceiving and interpreting actions. Inaccurate models significantly impair the system's capability to discern the

user's intentions, leading to challenges in providing timely and relevant assistance. Therefore, it is crucial to enhance the accuracy of these models to ensure they accurately reflect real-world scenarios, thereby optimizing the method's efficiency

- **Recurrent Neural Networks:** Recurrent Neural Networks (RNNs) are a specialized type of artificial neural networks crafted to recognize and interpret patterns in sequential data. What sets RNNs apart from traditional feedforward neural networks is their unique architecture featuring directed cycles in their connections.This design allows them to retain information over time, enabling the network to maintain a form of 'memory'.
  Such a capability is invaluable for tasks where understanding the context or the sequence in which data points appear is crucial.
  By iterating through elements in a sequence and maintaining a state that accumulates information seen thus far, RNNs effectively process sequences by leveraging their internal state (memory) to manage a range of inputs sequentially. This characteristic renders them incredibly useful for a variety of applications, including language modeling, speech recognition, and forecasting in time series data. Despite their advantages, RNNs are not without their challenges. Notably, they can be difficult to train effectively due to issues like vanishing and exploding gradients, which hinder their ability to learn long-range dependencies within the data.
  As elaborated in [33] to overcome these obstacles, Long Short-Term Memory (LSTM) networks emerge as a powerful solution within the domain of shared control of robotic arms, especially for tasks like intention prediction. The LSTM, with its unique ability to learn and remember over long periods, has significantly improved the predictability and accuracy of user intention in shared control systems. This advanced form of RNN addresses the core limitations of its predecessors by efficiently handling long-term dependencies. The robustness of LSTM networks against time gaps in data input, coupled with their capacity to adapt to individual user preferences, significantly boosts the efficiency and personalization of shared control systems. However, the sophisticated nature of LSTMs brings about challenges, including increased computational complexity and a greater need for extensive training data. Despite these challenges, the integration of LSTMs into RNN technology marks a significant leap forward. It expands the potential for more intuitive and adaptable human-robot collaboration within shared control environments, navigating through the practical difficulties associated with their implementation.

## 2.3 Haptic Guidance

In the domain of shared control systems, guidance embodies a sophisticated integration of feedback and control strategies, facilitating seamless and intuitive interactions between humans and robots. At the heart of guidance lies the objective to harmonize the decision-making capabilities and adaptability of human operators with the accuracy, reliability, and operational efficiency of robotic systems. This harmony is achieved through a continuous exchange of information, where the system furnishes the human operator with immediate, pertinent data, feedback, and actionable advice. This enables the operator to make well-informed decisions that guide the robot's actions. Such a reciprocal flow of information significantly enhances the effectiveness of task performance, elevates safety standards, and guarantees an advanced level of control and adaptability. This acts as a critical link between human operators and automated systems, effectively reducing the workload.

### 2.3.1 Methodologies

Guidance methodologies in shared control systems showcase a broad spectrum of approaches, highlighting the complexity of human operators and robotic mechanisms, especially in teleoperation scenarios;among these, Artificial Potential Fields (APF), the Dynamic Window Approach (DWA), Rapidly-exploring Random Trees (RRT) and pure pursuit algorithm (PPA) assisted by vector field histogram (VFH), have been proven to be particularly effective in guiding robotic systems in shared control environments.

- **Artificial Potential Fields (APF):** The implementation of Artificial Potential Fields (APF) [30] is integral to enhancing real-time shared control in teleoperation frameworks, focusing on obstacle avoidance and efficient path planning. APF employs virtual forces that guide the robotic system by creating a potential field where obstacles generate repulsive forces, and the goal exerts an attractive force. This dynamic allows the robot to navigate by following the gradient of the potential function, akin to traversing a landscape of peaks and valleys designed to steer clear of obstacles while being drawn toward the target. Upon setting a goal, APF aligns the robot's movement with the operator's intent by modulating these virtual forces, thereby facilitating an intuitive interaction between the human operator and the robotic system.
  The primary challenges associated with APF include the robot's susceptibility to getting trapped in local minima—regions where the potential gradient is zero, preventing further movement towards the goal—and the method's purely reactive nature to collision avoidance. In standard APF applications, the robot alters its course only when in close proximity to obstacles, which may not be efficient for advanced navigation and obstacle avoidance in cluttered environments.

  However, as presented in [13], the problem of obstacle avoidance in cluttered environments can be efficiently overcome with the dynamic generation of escape points around obstacles. These escape points are designed to help the robotic system bypass obstacles smoothly while avoiding the pitfalls of local minima.
  This approach not only addresses the issue of the robot getting stuck but also allows for more proactive obstacle avoidance by modifying the robot's trajectory in advance, rather than merely reacting when close to obstacles.

- **Dynamic Window Approach (DWA):** This approach is a fundamental technique in mobile robotics, utilized for motion planning and obstacle avoidance in dynamic environments. It operates by constraining the robot's velocity to a "dynamic window," considering its current state and capabilities. Within this window, potential velocities are evaluated using a cost function that incorporates factors such as obstacle distance, goal direction, and robot velocity. By selecting the velocity that maximizes this function while ensuring collision-free navigation, the DWA facilitates rapid decision-making in real-time motion control.
  To fit this method to a shared control contex, in [2] a novelty method has been developed. Building upon DWA principles, the Biomedical Dynamic Window

Approach (BDWA) integrates human navigation behaviors into robotic motion planning. This extension aims to generate robot movement resembling human path preferences, enhancing user interaction with assistive devices.
BDWA achieves this by selecting velocities aligning with human-like paths, guided by a reward function assessing trajectory resemblance. Through iterative refinement, BDWA balances safe navigation with human-like movement, optimizing the shared control experience in assistive technologies.

- **Rapidly-exploring Random Trees (RRT):** stand as fundamental algorithms for guiding autonomous systems , functioning by iteratively constructing a tree-like structure from an initial state towards a goal. By randomly sampling points within the configuration space and extending the tree towards these points, RRTs efficiently explore high-dimensional spaces, dynamically adapting to obstacles and terrain. As shown in [22], through this process RRTs provide real-time decision-making capabilities, balancing between thorough exploration and efficient path generation to navigate safely and effectively in various applications, including autonomous vehicles and robotic systems
- **Pure Pursuit Algorithm (PPA) and vector field histogram (VFH)):** Integrating them, as detailed in [3], offers a low computational demanding guidance, both scalable and energy-efficient. The PPA, responsible for path tracking through predefined waypoints, ensures that the robot adheres to its intended trajectory with a degree of precision dictated by the dynamically adjusted look-ahead distance. Meanwhile, the VFH algorithm, utilizing LiDAR sensor data, enhances obstacle avoidance capabilities by generating a polar histogram that represents obstacle density around the vehicle, guiding it away from potential collisions.

# Chapter 3

# Materials

## 3.1 Hardware

### 3.1.1 Universal Robot UR5

The robotic arm used for this research was the Universal Robots UR5 (see Fig.3.1). Originating from Universal Robots the UR5 robot is part of the UR series [29], which also includes the smaller UR3 and the larger UR10 models. The series is distinguished by its payload capacity, denoted by the numerical value in each model's name, indicating 3 kg, 5 kg, and 10 kg for the UR3, UR5, and UR10, respectively. The robotic arm of the UR5 collaborative robot is designed with a modular architecture, offering six degrees of freedom to ensure versatile and precise maneuvering. Its mechanical structure comprises six revolute joints, each powered by a servo motor.

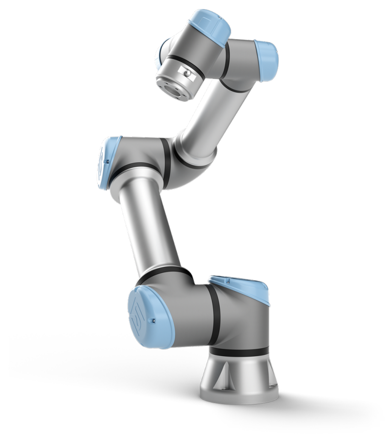


FIGURE 3.1: Universal Robot Ur5

The robot's control unit, houses the CPU (Fig. 3.2) and Teach Pendant (Fig. 3.3). It orchestrates the robot's movements, ensuring smooth and safe operations. Key functionalities include:

- **CPU:** Serves as the primary processing unit for the robot, executing instructions from the operating system and application software. It performs calculations, processes data, and controls the logic operations essential for robot functionality. The CPU enables the robot to interpret and execute commands, manage its sensors and actuators, and maintain connectivity with external devices or networks for control and data exchange.

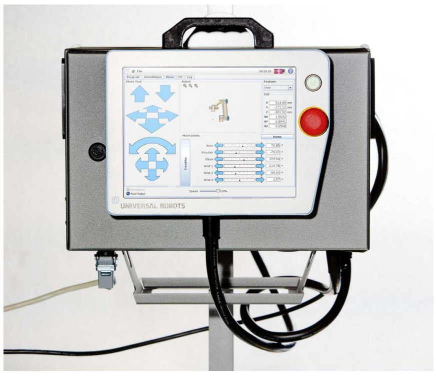


FIGURE 3.2: Control Unit Ur5

- **Teach Pendant:** The UR5 robot's teach pendant is engineered with a 12" full-color touchscreen, serving as the gateway to PolyScope—Universal Robots' proprietary graphical user interface. This advanced interface is designed for the seamless programming, management, and real-time monitoring of the UR5 robotic arm's operations. It supports direct code input and provides a comprehensive visual representation of program execution, including detailed error diagnostics with categorization for efficient troubleshooting.Equipped with three physical buttons, the teach pendant ensures operational control and safety are always within the operator's reach. The power button manages the system's states—powering on, shutting down, and rebooting the UR5 robot system.

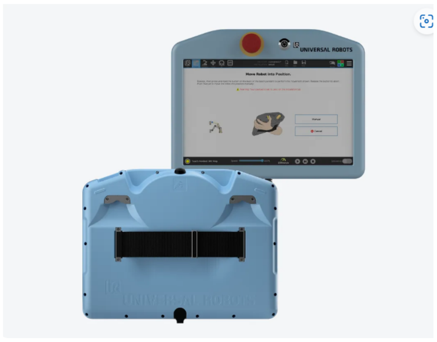


FIGURE 3.3: Teach Pendant Ur5

### 3.1.2 Omega 7 haptic device

The teleoperation controller used was the omega.7 haptic [12]. The device features an active grasping extension, enhancing its versatility.this is possible due to the design of its end-effector, which mimics the natural range of motion of the human hand, ensuring compatibility with bi-manual teleoperation consoles. This device is distinguished by its full gravity compensation and driftless calibration, aimed at improving user comfort and precision. The active gripper is capable of generating a grasping force up to 8 Newtons in both directions. Available in configurations for either the left or right hand, the omega.7 utilizes the traslational force feedback capabilities inherent to the omega.3 model.

| Category | Specification |
|---|---|
| **Workspace** | Translation: Ø 160 x 110 mm<br>Rotation: 240 x 140 x 180 deg<br>Grasping: 25 mm |
| **Forces** | Translation: 12.0 N<br>Grasping: ± 8.0 N |
| **Resolution** | Translation: < 0.01 mm<br>Rotation: 0.09 deg<br>Grasping: 0.006 mm |
| **Stiffness** | Closed-loop: 14.5 N/mm |
| **Electronics** | Interface: Standard USB 2.0<br>Refresh rate: Up to 4 KHz<br>Power: Universal 110V - 240V |

TABLE 3.1: Omega.7 Characteristics

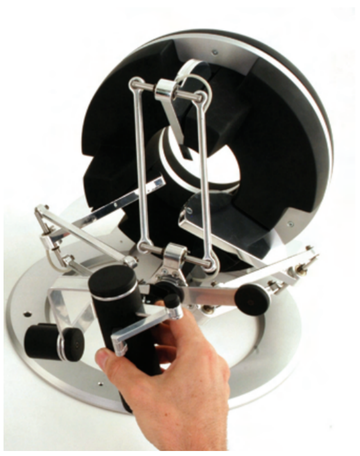


FIGURE 3.4: Omega.7 Haptic Device

### 3.1.3 qb SoftHand

The end-effector used was the qb SoftHand [24, 7], an advanced end-effector, characterized by an underactuated architecture, this robotic hand is adept at replicating up to 75% of the grasping functions of a human hand, thanks to its built-in mechanical intelligence which facilitates seamless adaptation to a variety of objects without the reliance on complex sensor arrays or intricate programming. The device is structured with five anthropomorphic fingers and nineteen phalanges, driven by a singular motor that operates through a differential tendon system to mimic human joint articulations accurately. This desensitized hand dynamically adjusts its grip to objects of varying shapes and sizes through its mechanical configuration, negating the necessity for sensor-based feedback. The incorporation of soft robotics technology ensures that the qb SoftHand Research is not only flexible and adaptive but also capable of safe interactions within its operational environment, minimizing the risk of damage to both objects and human

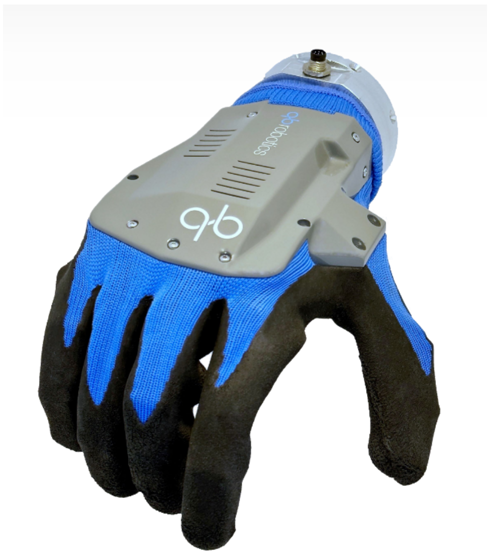


FIGURE 3.5: qb SoftHand Research

### 3.1.4 Kinect v2

The Kinect for Windows version 2 (Kinect v2) RGB-D camera is a sophisticated sensor that combines high-definition color imaging with depth sensing capabilities, providing detailed environmental perception for a wide range of applications. Its key features and specifications are:

- **RGB Camera**: Captures high-resolution color images at 1920x1080 resolution, enabling the recognition of objects, people, and environmental details.
- **Depth Sensor**: Utilizes time-of-flight technology to measure the distance between the sensor and objects in the environment with high precision.
- **Infrared Projector and Sensor**: Emitting and capturing infrared light, respectively, to calculate depth information by measuring the time it takes for light to travel to and from objects.
- **Depth Sensing**: Provides depth information at a resolution of 512x424 pixels, allowing for accurate distance measurements and 3D scene reconstruction.
- **Color Imaging**: Produces full-color RGB images with vibrant detail, facilitating object recognition and scene analysis.
- **Simultaneous Multi-Modal Capture**: Captures depth and color data simultaneously, enabling synchronized analysis of spatial and visual information.
- **Low-Light Performance**: Performs well in various lighting conditions, including low-light environments, thanks to its IR-based depth sensing technology.

FIGURE 3.6: Kinect 2 for windows

## 3.2 Software

### 3.2.1 ROS

ROS, is an open-source meta-operating system designed specifically for robots, offering 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.
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. 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. The Teleoperation System was fully deployed within the ROS framework, utilizing its robust set of tools and libraries tailored for robotic development. ROS serves as an ideal platform for the implementation of Teleoperation Systems due to its comprehensive suite of functionalities and its modular and extensible architecture.

### 3.2.2 Coppeliasim

To get data and create the intention recognition model and subsequently test the guidance, virtual models of the UR5 robotic arm and the qb Softhand were deployed. The virtual environment where it was done was Coppeliasim, a robot simulation platform designed to facilitate the development, testing, and deployment of robotic applications [9].
It was chosen for its highly customizable simulation environment, allowing for the precise modeling of physical properties such as friction, mass, and inertia, as well as the simulation of various sensors and actuators.
CoppeliaSim's powerful scripting functionality, based on embedded Lua scripts, enables users to control simulations, implement control algorithms, and simulate sensor feedback with high fidelity. Additionally, it offers an intuitive graphical user interface that simplifies the process of creating, configuring, and managing simulation scenes.

One of CoppeliaSim's distinctive advantages is its support for a wide range of programming languages and interfaces, including remote API clients in Python, C++, Java, and Matlab. This flexibility allows for seamless integration with external software tools and libraries, facilitating interdisciplinary research and the development of complex robotic systems.

Most importantly allows integration with Robot Operating System (ROS), allowing to control the robot from the ROS environment.
The initial step in the integration process involves establishing a communication link between CoppeliaSim and ROS. This is typically achieved through the use of a dedicated ROS package, such as `coppeliasim_ros_interface`, which facilitates direct interaction between CoppeliaSim's simulation environment and ROS nodes. The package utilizes CoppeliaSim's remote API capabilities to create a bridge that allows for the bidirectional exchange of information and commands.
Once the connection is established, ROS can control simulated robots within CoppeliaSim by publishing messages to specific topics that correspond to the robots' actuators and sensors.
This setup enables the simulation of sensor data acquisition, actuator control, and even complex robotic behaviors such as navigation, manipulation, and interaction with dynamic environments.
ROS nodes can subscribe to CoppeliaSim's simulated sensors' data, such as camera feeds, laser scans, or joint states, allowing for real-time monitoring and decision-making.
Similarly, nodes can publish commands to control the simulated robot's movements, manipulator positions, or any other actuator-driven actions, mimicking the control flow of a real robot.

### 3.2.3 Programming Language

The primary programming language for the model implementation of this research was Python, chosen for its versatility, rich ecosystem, and widespread use in machine learning. Python's readability and extensive libraries facilitated efficient coding and experimentation. In contrast, for the connection between the robot and the teleoperating device, C++ was chosen as the programming language. C++ was selected due to its efficiency, low-level control over hardware resources, and suitability for real-time applications

### 3.2.4 Libraries

**Numpy**
NumPy is an indispensable library in the Python ecosystem for scientific computing. It excels in providing support for large, multi-dimensional arrays and matrices, along with a comprehensive collection of mathematical functions to perform operations on these data structures. The foundation of NumPy lies in its powerful ndarray object, which enables high-performance operations on large datasets with ease, especially in the domain of machine learning and artificial intelligence.

The integration of NumPy with Keras TensorFlow utlises its arrays as the basic data structure for its operations, from defining models to feeding data into the network. This integration simplifies the process of data manipulation and transformation before it is passed into neural network models for training or inference.

**Pandas**
Pandas is a powerful data manipulation and analysis library for Python, known for its robust data structures such as DataFrames, which are instrumental in handling

structured data efficiently.
It stands out in tasks related to data cleaning, manipulation, and exploration, providing a wide array of functions for indexing, merging, grouping, and aggregating data. The seamless integration of Pandas with other pivotal libraries, including NumPy, underscores its versatility in managing labeled data and time series effectively.
Pandas serves as the foundation for data preprocessing, enabling the transformation of raw data into a structured form that is compatible with the requirements of Keras models. By facilitating tasks such as data normalization, handling missing values, and encoding categorical variables, Pandas ensures that the data fed into Keras models is of high quality and structured appropriately for deep learning tasks. The integration of Pandas with Keras simplifies the process of converting DataFrames into formats directly usable by Keras, such as NumPy arrays or TensorFlow tensors

**Kinematics and Dynamics Library**
The Kinematics and Dynamics Library (KDL) [**orocos_kdl**]is part of the broader Orocos (Open Robot Control Software) project, aimed at providing a comprehensive suite of libraries for the modeling and control of robotic systems. KDL specifically focuses on the kinematic and dynamic modeling of robots, offering tools to describe robot geometry and motion, solve kinematic chains, and compute forces and torques. KDL is implemented in C++, making it efficient and suitable for high-performance applications. Its modular design allows for the integration with other parts of the Orocos project, like the Real-Time Toolkit (RTT) for real-time execution of control algorithms.

**Keras**
Keras [8] is a Python-based, open-source library designed to facilitate the development and training of deep learning models efficiently. Acting as an interface for TensorFlow, it simplifies complex processes with predefined layers and algorithms for rapid experimentation. Emphasizing user-friendliness and modularity, Keras supports various back-end engines, operates on both CPUs and GPUs, and enables quick prototyping with minimal code. This makes it accessible to users at all levels in the fields of machine learning and artificial intelligence.

**Concurrent.futures**
The 'concurrent.futures' module in Python seamlessly integrates with predictions from Keras models, facilitating efficient parallel and asynchronous execution of prediction tasks. By leveraging the Executor interface, users can submit prediction tasks to a pool of threads or processes using the *submit()* method, which returns Future objects representing the asynchronous execution of the predictions.
This enables concurrent processing of multiple prediction requests, especially beneficial when dealing with large datasets or computationally intensive models. With proper exception handling and result retrieval mechanisms provided by Future objects, users can efficiently manage and process prediction results as they become available. Additionally, the*map()* function allows for parallelized prediction tasks across multiple inputs, further enhancing throughput and performance. When combined with Keras models, the 'concurrent.futures' module offers a streamlined solution for scaling prediction tasks, maximizing resource utilization, and accelerating model inference in Python applications

**Threading**
The Threading library in Python offers a straightforward approach to concurrency by enabling multiple threads to execute concurrently within a single process. When integrated with predictions from Keras models, the Threading library facilitates

parallel execution of prediction tasks by creating and managing lightweight threads. By utilizing the Thread class or higher-level constructs like ThreadPoolExecutor, users can spawn multiple threads to handle prediction requests concurrently. This concurrency model is particularly useful for I/O-bound tasks, such as loading data or preprocessing inputs, allowing the main thread to continue executing while waiting for prediction results. This functionality enables the parallel computation of the IRM and the Haptic Guidance.

## 3.3 Teleoperation System

In the process of controlling a robotic arm through haptic devices, an essential step involves establishing a mapping between the motion of the haptic handle and the corresponding actions of the robot's end-effector. Activating the clutch on haptic devices establishes a linkage between the movements of the interface's handle and the robotic arm's end-effector, recording their initial positions as $\boldsymbol{h}(t_0)$ for the interface and $\boldsymbol{s}(t_0)$ for the end-effector. Upon repositioning the handle to $\boldsymbol{h}(t_1)$, the system computes the displacement $\Delta\boldsymbol{h}$, represented by the position and orientation changes from $\boldsymbol{h}(t_0)$ to $\boldsymbol{h}(t_1)$, as demonstrated below:

$$\begin{aligned}\Delta\boldsymbol{h}^p &= \boldsymbol{h}^p(t_1) - \boldsymbol{h}^p(t_0)\\ \Delta\boldsymbol{h}^o &= \boldsymbol{h}^o(t_1)\boldsymbol{h}^o(t_0)^{-1},\end{aligned}$$

where $\boldsymbol{h}^p \in \mathbb{R}^{3\times1}$ denotes the position vector and $\boldsymbol{h}^o \in \mathbb{R}^{3\times3}$ indicates the orientation matrix of the interface's handle. Subsequently, the calculated $\Delta\boldsymbol{h}$ is adapted to the reference system of the robotic arm and integrated with $\boldsymbol{s}(t_0)$, facilitating the determination of the targeted new pose.

The 6-DoF manipulator was controlled in velocity and the Levenberg-Marquardt inverse kinematics solver available within the Kinematics and Dynamics Library (KDL) [28] was used to let the robot reach the goal pose.

Fig. 3.7 shows the control architecture.

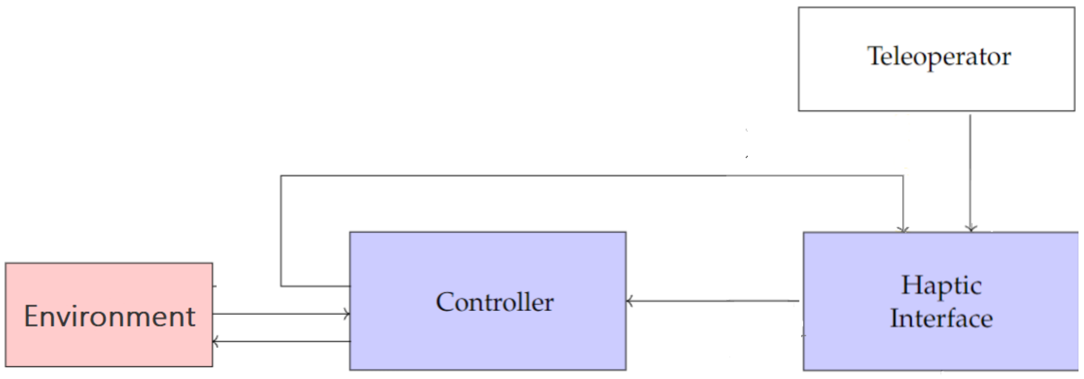


FIGURE 3.7: Teleoperation System

# Chapter 4

# Methods

This chapter presents a full description of the methodologies used to achieve a complete Shared Control architecture and its location in a robotic arm control system. To optimize the balance between guiding the teleoperator's actions and preserving its autonomy, a Shared Guided and Supervised Control (SGSC) architecture was selected.The architecture was implemented with two main components: Intention Recognition, utilizing deep learning recurrent neural networks (RNNs), and Haptic Guidance, utilizing artificial potential fields (APF).

## 4.1 Scenario

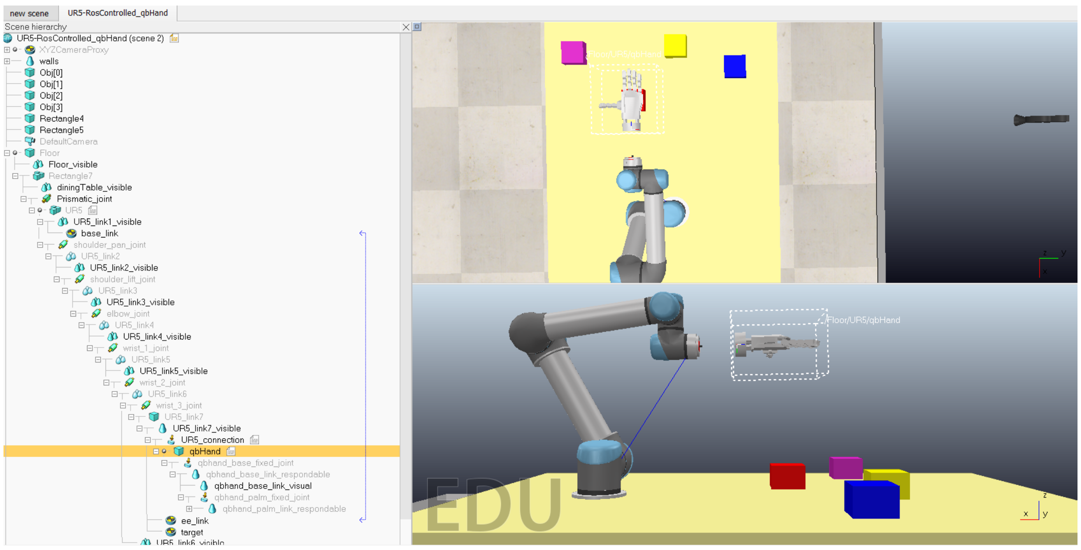
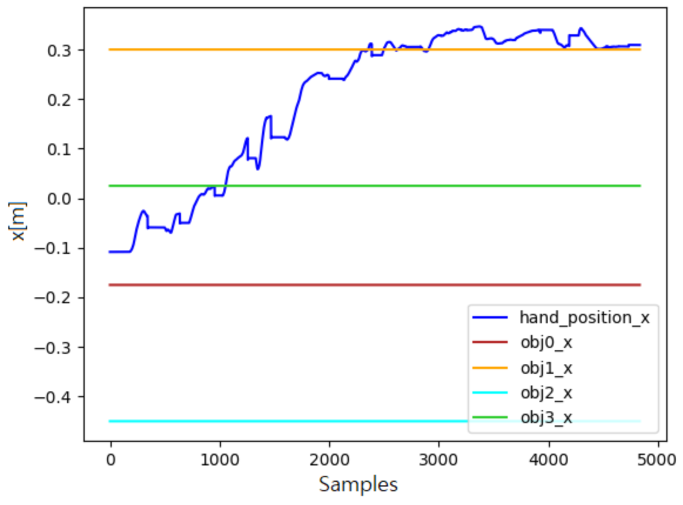
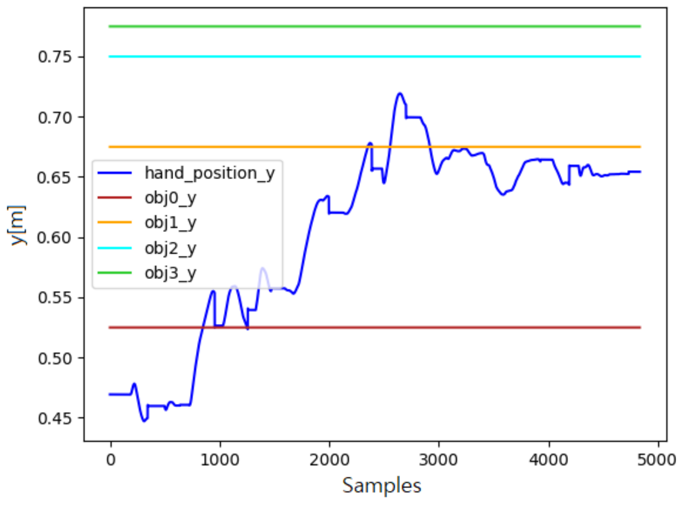


FIGURE 4.1: Virtual Scenario. The object labeling, from left to right, is '2','0','3','1'

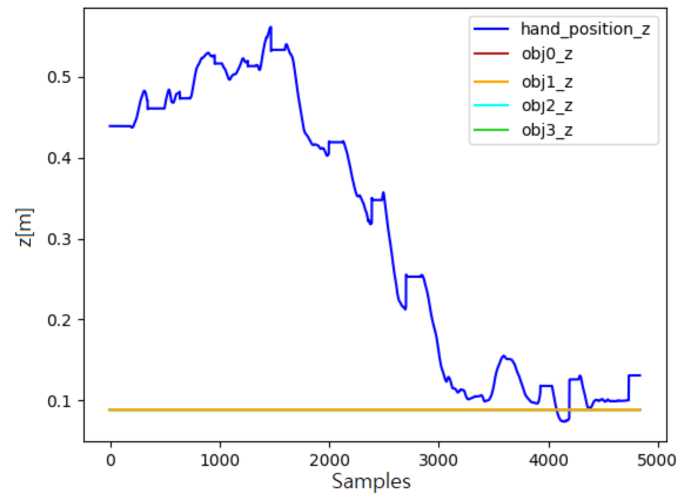
Fig. 4.1 illustrates the virtual scenario utilized for data analysis, training, and testing of the shared control architecture, featuring a robotic arm equipped with an end effector, a table, and various objects designated for pick-up by the teleoperator. On the left the hierarchy of child script, which are attached to the simulation objects and models, enabling them to execute Lua programming commands that interact directly with the CoppeliaSim API. Four objects were selected for the simulation environment to balance between fully utilizing the robotic arm's workspace and avoiding an overly cluttered environment. The experimental teleoperations were conducted by a skilled operator, who, chosen a goal object, moved towards the desired goal until touching it.



(A) End effector position on $x$-axes



(B) End effector position on $y$-axis



(C) End effector position on $z$-axis

FIGURE 4.2: End effector position with respect to the objects

As shown in Fig. 4.2, the end effector position with respect to the four objects, is not enough to fully represent the human intention. For example, when attempting to reach the goal object, the operator could pass near another object, resulting in a wrong evaluation of the goal object. In the context of human intention recognition, studies as [18], manifest the benefits of incorporating the direction of the end effector, creating a more complex but complete environment on which creating the mode (see Fig. 4.4). As shown in Fig.4.3, the reference frame most suited to evaluate the direction of the hand is the $z$ axis.

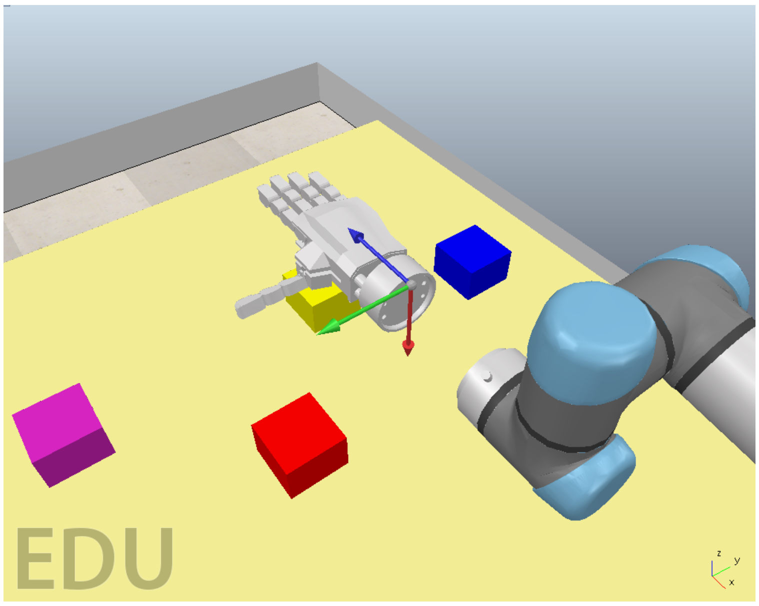


FIGURE 4.3: Hand reference system

To determine the relative direction of the end effector to an object, we first need to find the end effector's direction, involving the calculation of the end effector's absolute direction using its quaternion representation. We can achieve this by computing the rotation matrix from the quaternion and extracting its last column, which represents the $z$ axis orientation:

$$\text{Rotation Matrix} = \begin{bmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y-q_zq_w) & 2(q_xq_z+q_yq_w) \\ 2(q_xq_y+q_zq_w) & 1-2(q_x^2+q_z^2) & 2(q_yq_z-q_xq_w) \\ 2(q_xq_z-q_yq_w) & 2(q_yq_z+q_xq_w) & 1-2(q_x^2+q_y^2) \end{bmatrix}$$

$$\text{End Effector's Direction} = \begin{bmatrix} r_{31} \\ r_{32} \\ r_{33} \end{bmatrix}$$

Subsequently, the distance vector was calculated by subtracting the end effector's position from the object's position:

$$\text{Distance Vector} = \begin{bmatrix} x_{obj}-x_{ee} \\ y_{obj}-y_{ee} \\ z_{obj}-z_{ee} \end{bmatrix}$$

Finally, the angle between the two vectors was calculated using the dot products:

$$\text{Angle between two vectors} = \arccos\left(\frac{\text{End Effector's Direction} \cdot \text{Distance Vector}}{\|\text{End Effector's Direction}\| \cdot \|\text{Distance Vector}\|}\right)$$

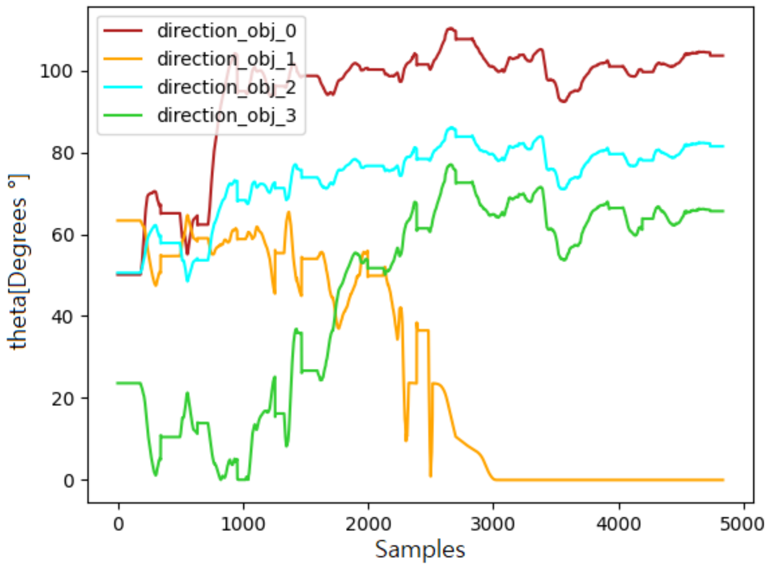


FIGURE 4.4: Direction of end effector with respect to each object (in degrees)

To further generalise the scenario and the input features of the model, the position has been linked to each object in relative manner, by, as shown in Fig. 4.5, calculating the distance with respect to each object.

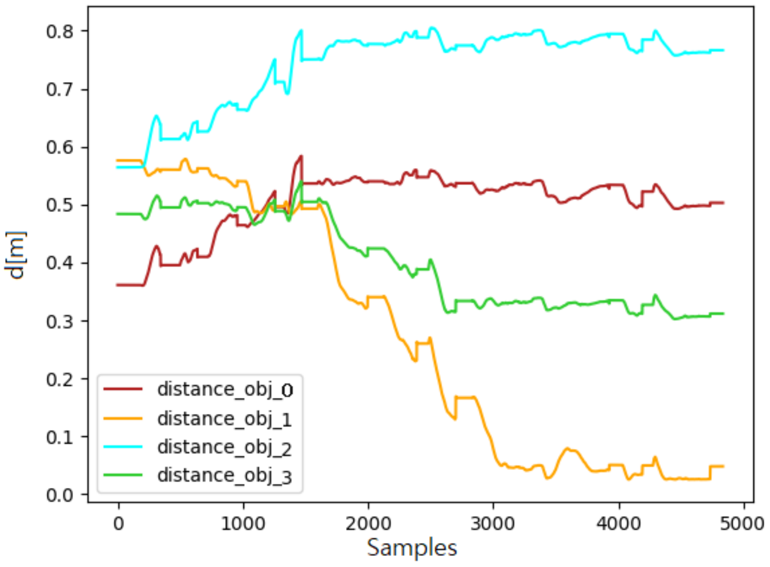


FIGURE 4.5: Distance of end effector with respect to each object

## 4.2 Human Intention Recognition

Distance and direction were finally chosen as input for the Intent Recognition model. Since the decision process of an operator to fully mimic the human intention decision process, it was also taken into account the temporal correlation between these two features. To tackle relations between two time series data inputs, two methodologies have been proven to be the most effective: a Hidden Markov Model approach and a Recurrent Neural Network approach. As shown in [18], although being HMM a valid methodology with tangible results, has as a trainable parameter only the rate parameter $\lambda$, representing the decision to stick at a chosen goal object mapped as an exponential distribution. This model will, inevitably, result biased and will simplify the complex nuances of human intention. Since the need to comprehend the intrinsic connection between distance and direction and how this relation varies in time, a more complex model was needed.
A recent and novel approach in intent recognition is to use Recurrent Neural Networks, more specifically Long Short Term Memory Networks, a type of RNN that is capable of learning long-term dependencies of multiple inputs.

### 4.2.1 Recurrent Neural Networks

A recurrent neural network (RNN) [21] is a specialized form of artificial neural network designed to handle time series data or sequential data. Unlike standard feedforward neural networks, which assume that data points are independent of each other, RNNs are tailored for scenarios where the relationship between sequential data points is crucial.
In such cases, where one data point is dependent on the previous ones, the neural network architecture must be adapted to capture these dependencies. RNNs achieve this through the concept of "memory," enabling them to retain information from previous inputs to inform the generation of subsequent outputs in the sequence.

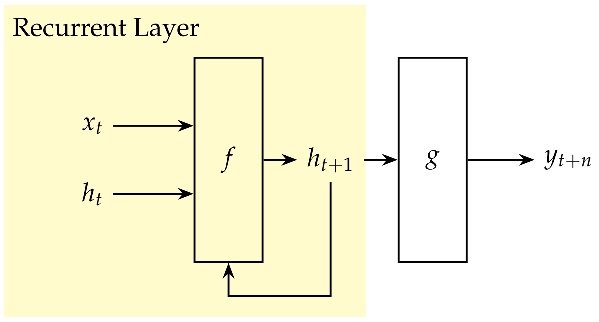


FIGURE 4.6: Recursive Neural Network (RNN)

As shown in Fig. 4.6, the RNN is characterized by:

- $x_t \in \mathbb{R}$ denotes the input at time step $t$. For simplicity, $x_t$ is assumed to be a scalar with a single feature. This concept can be generalized to a $d$-dimensional feature vector.
- $y_{t+1} \in \mathbb{R}$ represents the output of the network at time step $t+1$.
- $h_t \in \mathbb{R}^m$ stores the values of the hidden units or states at time $t$, also known as the current context.
- $h_{t+1} \in \mathbb{R}^m$ denotes the hidden state at time $t+1$

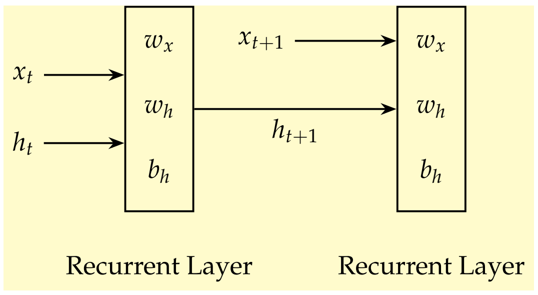


FIGURE 4.7: Recursive Neural Network single layer

At every time step, the network can be unfolded for $k$ time steps to obtain the output at time step $k + 1$. This unfolded network is akin to a feedforward neural network. The rectangle in the unfolded network indicates an operational sequence. An example of single layer RNN is shown in Fig. 4.7, where:

- $w_x \in \mathbb{R}^m$ are weights associated with inputs in the recurrent layer
- $w_h \in \mathbb{R}^{m \times m}$ are weights associated with hidden units in the recurrent layer
- $w_y \in \mathbb{R}^m$ are weights associated with hidden units to output units
- $b_h \in \mathbb{R}^m$ is the bias associated with the recurrent layer
- $b_y \in \mathbb{R}$ is the bias associated with the feedforward layer

For instance, with an activation function $f$, the next hidden state is computed as:

$$h_{t+1} = f(x_t, h_t, w_x, w_h, b_h) = f(w_x x_t + w_h h_t + b_h)$$

The output $y_t$ at time $t$ is determined by:

$$y_t = f(h_t, w_y) = f(w_y \cdot h_t + b_y)$$

Here, $\cdot$ signifies the dot product.
Consequently, during the feedforward pass of an RNN, the network computes the values of the hidden units and the output after $k$ time steps. The network's weights are temporally shared. Each recurrent layer possesses two sets of weights: one for the input and another for the hidden units. The last feedforward layer, which calculates the final output for the $k$th time step, resembles a conventional layer in a traditional feedforward network.

**The Activation Function** Any activation function may be employed within the recurrent neural network. Commonly used functions include:

- Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Tanh function: $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$
- ReLU function: $\mathrm{ReLU}(x) = \max(0, x)$

**Training and Vanishing Gradient** Training neural networks involves adjusting their weights to minimize the difference between the predicted output and the actual target values. For networks that process sequential data, like time series or natural language, this training must account for the temporal dependencies within the data. Recurrent Neural Networks (RNNs) are specifically designed for this purpose, capable of maintaining state across sequential inputs through their looping architecture. However, training RNNs presents unique challenges due to their recurrent nature, necessitating specialized techniques to effectively learn from sequences.

The main steps in the training process can be summarized as follows:

- **Forward Pass:**

  During the forward pass, we sequentially compute the activations and outputs for each time step, from $t = 1$ to $T$, where $T$ is the sequence length. This process mirrors the execution of a feedforward neural network but incorporates temporal dependencies through the hidden states.

- **Computing the Cost:**

  Following the forward pass, the network's performance is evaluated using a cost function $C$, which measures the discrepancy between the predicted output sequence $\{o_1, o_2, ..., o_T\}$ and the target sequence. The choice of $C$ is dependent on the task, with Mean Squared Error (MSE) and Cross-Entropy being common choices for regression and classification tasks, respectively.

- **Backpropagation Through Time (BPTT):**

  BPTT involves the backward propagation of the cost function's gradients through the unrolled network, extending traditional back-propagation to account for the network's temporal structure. The gradients of $C$ with respect to the weights, such as $\frac{\partial C}{\partial W_{hh}}$, $\frac{\partial C}{\partial W_{xh}}$, and $\frac{\partial C}{\partial W_{ho}}$, are computed using the chain rule and aggregated over all time steps, facilitating the update of weights based on their influence on future outputs.

- **Unrolling the RNN Through Time:**

  To apply Backpropagation through time (BPTT), we first unroll the RNN across time, transforming it into an equivalent deep feedforward network. This unrolled structure maps each time step to a distinct layer in the network, facilitating the application of backpropagation as in traditional neural networks. The unrolled RNN can be mathematically represented as:

$$
\begin{aligned}
&x_t : \text{Input at time step } t, \\
&h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h) : \text{Hidden state at time step } t, \\
&o_t = g(W_{ho}h_t + b_o) : \text{Output at time step } t,
\end{aligned}
$$

  where $f$ and $g$ denote non-linear activation functions, $W_{hh}$, $W_{xh}$, and $W_{ho}$ are weight matrices, and $b_h$, $b_o$ are bias vectors.

- **Weights Update:**

  The final step in BPTT is updating the network's weights using the computed gradients. This is typically done using optimization algorithms like Stochastic Gradient Descent (SGD) or Adam, with update equations of the form:

$$W = W - \eta \frac{\partial C}{\partial W},$$

where $W$ represents the weight matrices ($W_{hh}$, $W_{xh}$, $W_{ho}$) and $\eta$ is the learning rate.

A significant challenge in the BPTT process is the phenomenon referred to as the *vanishing gradient problem*. This issue predominantly arises in scenarios involving the processing of long time series data. As the gradient of the cost function with respect to the network's weights is propagated backward through time, its magnitude diminishes exponentially. This attenuation is a consequence of the repeated multiplication of gradients at each time step, which are often less than one in magnitude. Consequently, this results in exceedingly small gradient values as one moves further back in time. The direct implication of this phenomenon is the negligible adjustment of weights corresponding to the earlier portions of the input sequence, thereby leading to slow convergence rates and a pronounced difficulty for the network in learning dependencies across extended sequences.

### Vanishing Gradient Problem

The core of the issue comes from the characteristics of certain activation functions, notably the sigmoid function and the hyperbolic tangent function (tanh). The sigmoid function, represented as $\sigma(x)$, and the hyperbolic tangent function, denoted as $\tanh(x)$, are mathematically articulated as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}}, \tag{4.1}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{4.2}$$

effectively squeezing an extensive range of input values into a narrow band between 0 and 1, for the sigmoid, and -1 to 1, for the tanh .

This compression of input values means that large variations in input can result in disproportionately small changes in output. Consequently, the derivatives of these functions, crucial for the backpropagation algorithm, are significantly small in magnitude. For both functions, this effect is more pronounced for inputs of large magnitude, where the slope of the curve flattens out, leading to derivatives that approach zero.

The sigmoid function is characterized by its distinctive S-shaped curve, serving as a pivotal nonlinear activation function in neural networks.
One of its notable features is the relationship between the input magnitude and the rate of change in the output. As the absolute value of the input escalates, the incremental change in the output significantly diminishes, leading to a derivative that approaches zero.

Mathematically, the derivative of the sigmoid function, denoted as $\sigma'(x)$, can be expressed as:

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x)), \tag{4.3}$$

where $\sigma(x)$ is the sigmoid function itself.

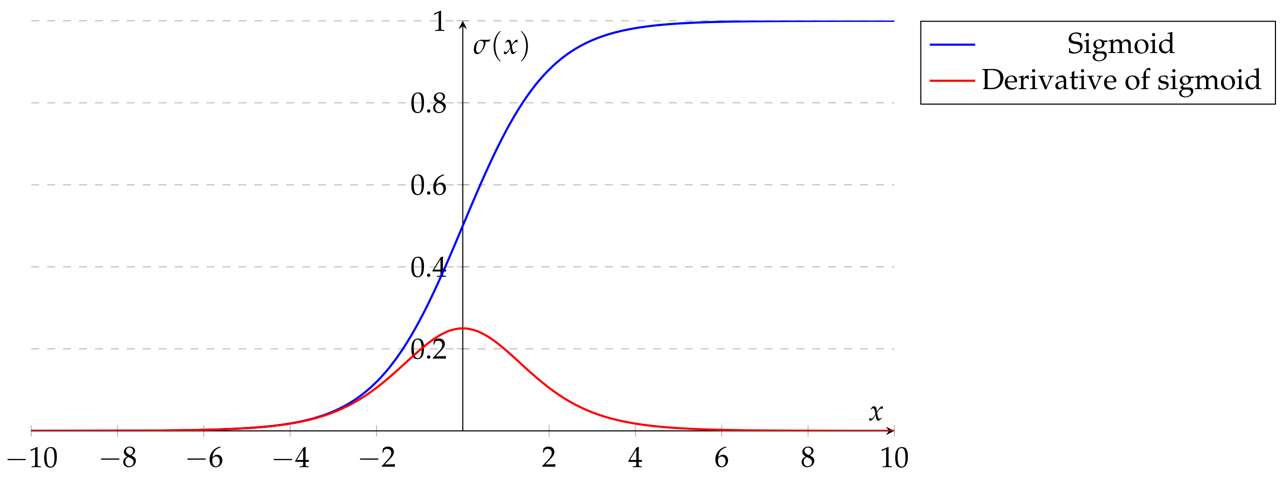


FIGURE 4.8: The sigmoid function and its derivative

In networks with a shallow architecture and fewer layers employing these activation functions, the vanishing gradient is less of an issue. However, the problem escalates with increased network depth, making the gradient too small for effective training.

### 4.2.2 Long Short-Term Memory (LSTM) Networks

LSTMs, as discussed by Graves et al. [14], were designed to specifically address the challenges encountered with traditional RNNs, notably the vanishing gradient problem. Unlike standard RNNs, LSTMs incorporate a sophisticated gating mechanism to control the flow of information and gradients within the network.
This design significantly mitigates the vanishing gradient issue, enabling the network to learn from and remember information over extended sequences more effectively. The figure below illustrates the internal structure of an LSTM cell, showcasing its unique components:

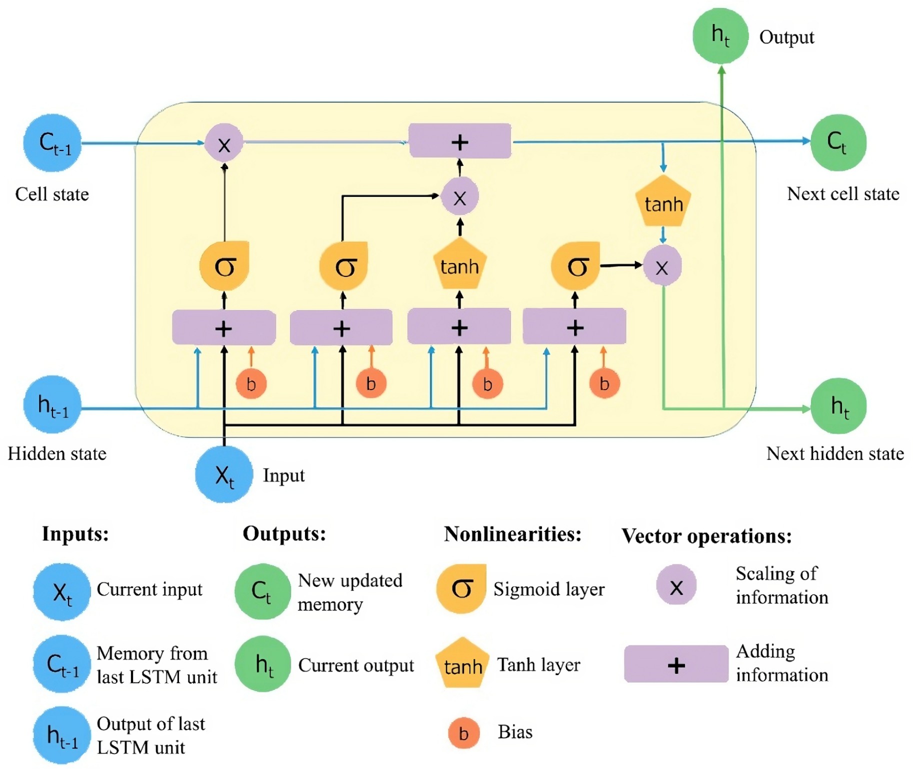


FIGURE 4.9: LSTM Cell Structure

LSTM cells include three critical gates: the input gate, the forget gate, and the output gate. These gates collectively manage the cell's memory, determining what information to retain, discard, and output.
This gating mechanism is pivotal for the LSTM's ability to modulate the gradient flow during backpropagation, directly addressing the vanishing gradient problem by maintaining gradient magnitude across long sequences. The LSTM cell's core components are as follows:

- **Forget Gate :**The forget gate plays a vital role in the LSTM architecture by deciding which information from the cell state should be kept or forgotten. It employs a sigmoid function to generate values between 0 and 1, with these values indicating the extent to which information is preserved or discarded. This selective memory process enables LSTMs to focus on relevant information while minimizing unnecessary data retention, aiding in the preservation of gradient significance.

- **Input Gate and Output Gate :**The input gate is responsible for updating the cell state with new information. It operates in two steps: a sigmoid layer determines which values will be updated, and a tanh layer generates new candidate values. Conversely, the output gate decides what the next hidden state should be, which includes the filtered cell state information to be used in output.
  These gates ensure the network's ability to capture essential details for the task at hand, enhancing its long-term dependency modeling capabilities.

  The operation of both the input and output gates ensures that if the outcome of these gates is close to 1, gradients can pass through without significant attenuation.Conversely, outcomes near 0 block the gradient flow.

This dynamic allows LSTMs to mitigate the vanishing gradient problem effectively.

- **Memory Cell State :**The memory cell state is the core component that allows LSTMs to retain information across time steps. It integrates inputs from the current time step and previous cell state, facilitating the network's ability to maintain a continuous flow of relevant information through the sequence. The additive nature of the cell state updates helps to sustain gradient flow across long sequences, ensuring that LSTMs can capture and leverage long-range dependencies in the data.

- **Forget Gate:** The forget gate plays a vital role in the LSTM architecture by deciding which information from the cell state should be kept or forgotten. It employs a sigmoid function to generate values between 0 and 1, with these values indicating the extent to which information is preserved or discarded.

  $$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

  This selective memory process enables LSTMs to focus on relevant information while minimizing unnecessary data retention, aiding in the preservation of gradient significance.

- **Input Gate:** The input gate is responsible for updating the cell state with new information. It operates in two steps: a sigmoid layer determines which values will be updated, and a tanh layer generates new candidate values.

  $$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

  $$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

  Conversely, the output gate decides what the next hidden state should be, which includes the filtered cell state information to be used in output.

- **Cell State Update:** The memory cell state is the core component that allows LSTMs to retain information across time steps. It integrates inputs from the current time step and previous cell state, facilitating the network's ability to maintain a continuous flow of relevant information through the sequence.

  $$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

  The additive nature of the cell state updates helps to sustain gradient flow across long sequences, ensuring that LSTMs can capture and leverage long-range dependencies in the data.

- **Output Gate:** Determines the next hidden state, which includes the filtered cell state information to be used in outputs. The operation of the output gate ensures that gradients can pass through effectively, making it a key component in mitigating the vanishing gradient problem.

  $$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

  $$h_t = o_t * \tanh(C_t)$$

### 4.2.3 LSTM Intent Recognition Architecture

This thesis explores the LSTM networks for recognizing human intent, mapping the intricate relationship between input features without imposing significant biases. As highlighted by Wu et al. [32], LSTMs are particularly adapt at handling and comprehending the complex temporal patterns present in time series data. The flexibility of the neural network architecture gives a broad variety of possible methods of utilization, from supervised and unsupervised, regression and classification. In this thesis a supervised approach is used, training the model with experimental trials from an expert teleoperator. The approach adopted in this thesis is a binary classification approach, i.e., the model predicts the correct label of a given input data. Thus, it differs from the one used in [32], since the presented approach was based on a regression. The model trained in this work has to receive as input two sequences of predetermined length and it acts as a binary classification, outputting the probability of the operator picking the object. By doing so there is not one single model representing the overall intention of the operator, but a model for each object in the scene, producing N distinct probabilities which will then be used to determine the object to be picked.
The use of relative input features, which are distance and direction, and the use of N identical models, as presented in Figure 4.10, ensures modularity and scalability of the model. Thus making the whole intention recognition process adaptable to scenarios with multiple objects, and the possibility to add and move more objects to the scene without the need to retrain the model.

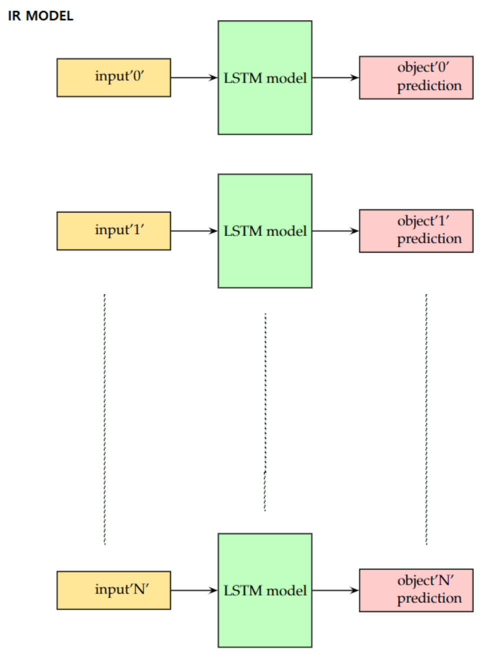


FIGURE 4.10: Complete Intent Recognition Model

Fig. 4.11 illustrates the overall architecture of the Intent Recognition Model (IRM), delineating all the input sequences, the various layers, and the output probability.

The initial step in developing the model involves setting the length of the input sequences, which must be appropriate for representing the complexity of the data

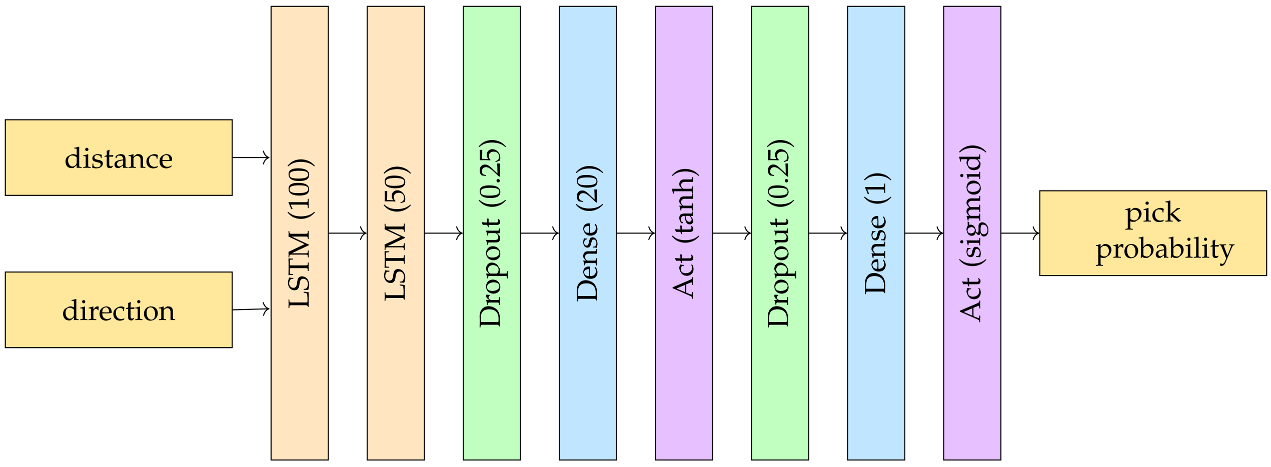


FIGURE 4.11: Single Intention Recognition Model

without overloading the model with useless information.
Empirically it was found that 40 time steps was not only, sufficient to capture the relevant features of the input data while also being computationally feasible, but was able to outperform longer sequences, more will be said in the training paragraph.

Once determined the input length, the number of neurons for the first LSTM layer, which is the size of the hidden state vector produced by the layer at each time step, had to be fixed.
The hidden state vector is the output of the LSTM layer at the i-th time step, encoding information learned from the input sequence up to that point;in this scenario the number of neurons in the first LSTM layer was set to 100, enabling the model to have a large enough capacity to capture complex patterns in the input data.

Recent studies on the application of LSTM networks to time series data, highlighted by [10] and [20], have revealed the efficacy of employing stacked LSTM layers in enhancing model performance, by facilitating hidden layers in capturing progressively higher-level representations of the sequence data.
Deep LSTM models, characterized by multiple LSTM hidden layers, create a structured flow where the output of one layer directly feeds into the subsequent layer, thereby significantly boosting the neural network's capability to learn.
The optimal number of layers for, given this type of data, was determined to be two, with the second layer containing 50 neurons. The choice of a decrease of neurons was made to retrieve only the most valuable information from the previous layer.
This choice was guided by the recognition that, although additional layers may enhance the model's ability to identify complex relationships among feature variables, a point is reached where further layers contribute to diminishing returns and increased risk of overfitting.
Considering the dataset characterized by a relatively small number of input timesteps, incorporating a third layer would likely lead to overfitting, undermining the model's generability and efficiency.

To further mitigate the risk of overfitting in models with multiple layers, such as stacked LSTM networks, dropout layers have been introduced.
This method involves randomly removing a specified proportion of neurons, as determined by the dropout ratio, from the network during each training iteration, effectively preventing their contributions to downstream neuron activation and halting weight updates during backpropagation for that iteration.

For this experiment, the dropout ratio was set to 0.25, meaning 25% of neurons in a layer are ignored at each training step, reducing the model's dependence on specific neurons and encouraging the learning of more robust features through forced feature redundancy.
This approach not only helps in spreading out weights to diminish the likelihood of overfitting but also ensures the model does not rely too heavily on a small set of neurons, thereby improving its ability to generalize to unseen data.
It is crucial to note that dropout is applied exclusively during training, with all neurons active during testing or evaluation phases, although their outputs are adjusted by the dropout rate to maintain consistency with the training environment

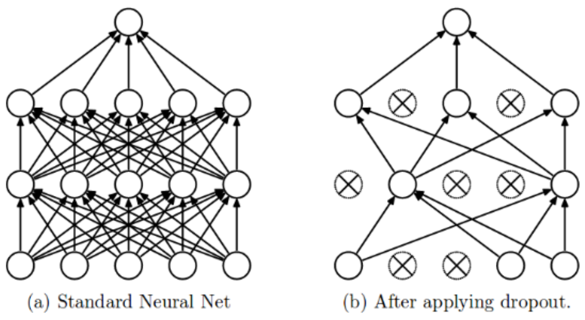


FIGURE 4.12: Effect of Dropout layer. [4]

Another crucial part of the model, is the Fully-Connected layer, often referred to as a Dense layer in neural networks, which ensures a complete connection between the neurons of consecutive layers.
In such a layer, each neuron from the previous layer is linked to every neuron in the next, facilitating the layer's ability to derive a comprehensive linear combination of the input features. This configuration is instrumental in the network's capability to discern complex patterns within the depth of neural network models.
As seen in the stacked LSTM, which saw a reduction of cell from 100 to 50, the first Dense layer was configured to contain 20 neurons.
This configuration aims to refine the feature space, enabling the network to concentrate on the most significant patterns identified by the LSTM layers, culminating in a final output layer comprising a single neuron. Both the LSTM and the first Dense layers employ the *tanh* activation function.
The *tanh* function is frequently chosen for LSTM layers due to its effectiveness in capturing complex patterns by introducing non-linearity into the network.
Non-linearity is essential, as a purely linear model would be incapable of recognizing the intricate patterns present within the data.

Since the model is defined as a binary classificator, the architecture necessitates the inclusion of a Dense layer as the final component, distinguished by a single neuron utilizing a 'sigmoid' activation function.
The 'sigmoid' function, characterized by its output range extending from 0 to 1, permits this concluding layer to map the features refined by antecedent layers into a predictive probability.
This calculated probability serves to articulate the model's confidence in categorizing the input under one of two potential outcomes: the action to pick or the decision against picking the object.

### 4.2.4 Training the model

The IM model defined in the previous section is the result of a fine-tuning process, which involved the training of multiple models with different configurations and hyperparameters.
The training process was conducted, with a fixed number of objects in the scene, to ease the supervisory labeling and to not unnecessarily unbalance the positive and negative classes. Since the training data derived from the recording of 26 different tests, each returning the whole timesteps of the scene; the data had to be preproccesed to be fed into the model.
The entire twentysix tests, each returning four different distance and direction sequences, were subdivided in batches of 40 timestamps each of them labeled with the appropriate value: one for the picked object and zero for the others, returning a 25 to 75 ratio of positive to negative classes. This may seem unbalanced but it was found to be the best to avoid false positives, which could lead to shifting prediction between two objects, returning a jittering prediction.
The dataset was subsequently divided into training and validation sets, with the training set containing 80% of the data and the validation set containing the remaining 20%. Training the LSTM model for binary classification focuses on minimizing the loss function, specifically *'binary_cross-entropy'*, which measures the discrepancy between the predicted probabilities and the actual binary labels, the labels being 1 for pick and 0 for not pick. The binary cross-entropy loss is given by:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

where:

- $N$ is the number of samples in the dataset,
- $y_i$ is the actual label of the $i$-th sample,
- $\hat{y}_i$ is the predicted probability of the $i$-th sample being in class 1.

This loss function aims to reduce the difference between the model's predictions and the actual labels, guiding the LSTM towards accurate predictions.

The optimization is performed using the Adam optimizer, a sophisticated variant of stochastic gradient descent that adaptively adjusts learning rates for each parameter. The Adam optimizer's update rule is:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

where:

- $\theta$ denotes the model parameters,
- $\eta$ is the learning rate,
- $\hat{m}_t$ and $\hat{v}_t$ are the estimates of the first and second moments of the gradients,
- $\epsilon$ is a small scalar added to improve numerical stability.

To solve overfitting and ensure the model's generability, a *ModelCheckpoint* callback is utilized. This callback monitors the validation loss, saving the model configuration with the lowest validation loss, thereby facilitating optimal model evaluation and deployment:

$$\text{ModelCheckpoint: Save model at epoch } n \text{ if } L_{\text{val},n} < L_{\text{val},n-1}$$

where $L_{\text{val},n}$ represents the validation loss at the $n$-th epoch.

## 4.3 Guidance System for Teleoperation

The guidance system is designed to equip the teleoperator with essential navigational cues for accurately reaching target objects in real-time. To this end, the Artificial Potential Field (APF) methodology has been selected for its efficacy, adaptability in dynamic environments and light computational cost.

### 4.3.1 Artificial Potential Fields

APFs [17] serve as a fundamental navigational framework in autonomous robotics, facilitating smooth navigation through complex environments. The robot's movement is influenced by the gradient of the generated potential field, which is the spatial derivative of the potential functions. The potential field itself is a scalar field that assigns a value (the potential) to every point in the robot's environment, constructed from attractive and repulsive components. To tailor the approach to the specific needs of teleoperation, various potential functions have been considered, each offering unique advantages:

- **Paraboloidal Potential Function:** Generates an attractive force that intensifies linearly with distance, defined as:

  $$U_{\text{att}}(d) = \frac{1}{2}k_{\text{att}}d^2$$

  Here, $U_{\text{att}}$ is the attractive potential, $d$ the distance to the goal, and $k_{\text{att}}$ a constant determining the force's strength.

- **Conical Potential Function:** Provides a consistent attractive force towards the goal:

  $$U_{\text{att}}(d) = k_{\text{att}}d$$

  This model is favored for its uniform guidance across different distances.

- **Exponential Decay Function:** Offers a decaying attractive force as the robot nears the goal:

  $$U_{\text{att}}(d) = A \exp\left(-\frac{d}{\sigma}\right)$$

  with $A$ as the amplitude and $\sigma$ the decay rate, dictating the field's curvature.

- **Harmonic Potential Function:** Achieved by solving the Laplace equation, it ensures a gradient free from local minima, barring the goal:

  $$\nabla^2 U_{\text{att}} = 0$$

  The form of $U_{\text{att}}$ is influenced by specific boundary conditions and the environment's layout.

- **Gaussian Potential Function:** Employs a Gaussian profile for a smooth attraction towards the goal:

  $$U_{\text{att}}(d) = A \exp\left(-\frac{d^2}{2\sigma^2}\right)$$

  where parameters $A$ and $\sigma$ regulate the amplitude and dispersion of the Gaussian force, respectively.

### 4.3.2 Haptic Guidance

Extensive empirical analysis has determined the exponential decay function to be the optimal environmental model. This function is characterized by its capacity for a controlled reduction in force magnitude as the robot progresses toward its objective [25]. The gradient of the exponential decay function:

$$\nabla U_{\text{att}}(d) = -\frac{A}{\sigma} \exp\left(-\frac{d}{\sigma}\right).$$

where: - $A$ is a constant defining the magnitude of the attraction force, - $\sigma$ is a scaling parameter affecting the decay of the exponential function, - $d$ is the distance variable.

This gradient modulates the force ensuring a consistent approach to the target, facilitating accurate and smooth engagement with the goal area, granting the operator precise control by nearing zero as it approaches the object, and facilitating precise adjustments to the grasping position.

Although beneficial for moving toward the goal, the exponential nature of this gradient is detrimental at the starting point, returning strong, undesired guidance when the system and, possibly, the user do not have a clear understanding of the goal object. To mitigate this effect, the guidance strategy operates under a dual-phase approach, segmented by a threshold parameter.
Initially, to facilitate operator familiarization and selection freedom, the exerted force follows the inverse of the gradient of a Paraboloidal Potential Function :

$$\frac{1}{\nabla U_{\text{att}}(d)} = \frac{1}{k_{\text{att}} d}$$

where:
-$k_{\text{att}}$ is a constant
-$d$ represents the distance end effector - goal

The $k_{\text{att}}$ constant is chosen to match the gradient of the exponential decay function at the threshold. This approach supports user adaptability during the early stages of interaction. The threshold was empirically set at 0.5 meters, offering the flexibility for operators to adjust it as needed through tuning mechanisms.

Lastly, since the distance, in this regard, between the end effector and the goal object is calculated in the world frame, which coincides with the base link of the robot, the resulting gradient is a vector **G** with three components ($x$, $y$, $z$) in the world reference frame.
As mentioned in Chapter 3, the haptic controller is mapped in the end effector's frame, thus we need to transform the vector guidance into its frame. To transform

the vector **G** from the world reference frame to the end effector's reference frame, the inverse of the rotation matrix **R**, denoted as $\mathbf{R}^{-1}$, was used, representing the orientation of the end effector with respect to the world frame.
The transformation is achieved by multiplying $\mathbf{R}^{-1}$ by **G**, as shown below:

$$\mathbf{G}' = \mathbf{R}^{-1}\mathbf{G}$$

where: - $\mathbf{G} = [G_x, G_y, G_z]^T$ is the original vector guidance in the world reference frame. - $\mathbf{R}^{-1}$ is the inverse of the rotation matrix **R**, which is based on the orientation of the end effector. - $\mathbf{G}' = [G'_x, G'_y, G'_z]^T$ is the vector transformed into the end effector's reference frame.

The inverse rotation matrix $\mathbf{R}^{-1}$ effectively reorients the vector **G** from the world frame into alignment with the end effector's frame. It is important to note that the inverse of the rotation matrix **R** is equal to its transpose when **R** is orthogonal.

## 4.4 Teleoperation System with Shared Control

Eventually, we integrated the shared control model into the teleoperation system. As depicted in Fig. 4.13, the system processes data on object positions directly from the kinect camera or the simulated enviroment, along with the position and orientation of the end effector within the robot frame.

Subsequently, this processed data is inputted into the IRM. Direction and distance vectors, representing the end effector's relation to each object, are then supplied to each LSTM model, whose output is the probability of picking or not picking each object.
The resulting probabilities are then compared, with the highest probability indicating the preferred object. The output of the IRM comprises the position of the goal's objects, which is directly fed into the guidance system. Additionally, the guidance system incorporates the position of the end effector and, in conjunction with the goal object position, calculates the feedback forces to be outputted.
The IRM and Guidance system operate concurrently, thereby significantly enhancing the real-time performance of the entire shared control architecture. At intervals of 0.06 seconds, the Guidance system receives a new prediction from the IRM and executes corresponding actions. During the interval between predictions, the Guidance system continues calculatating the new value of the gradient every time sample (0.01 seconds), consequently determining the feedback force.

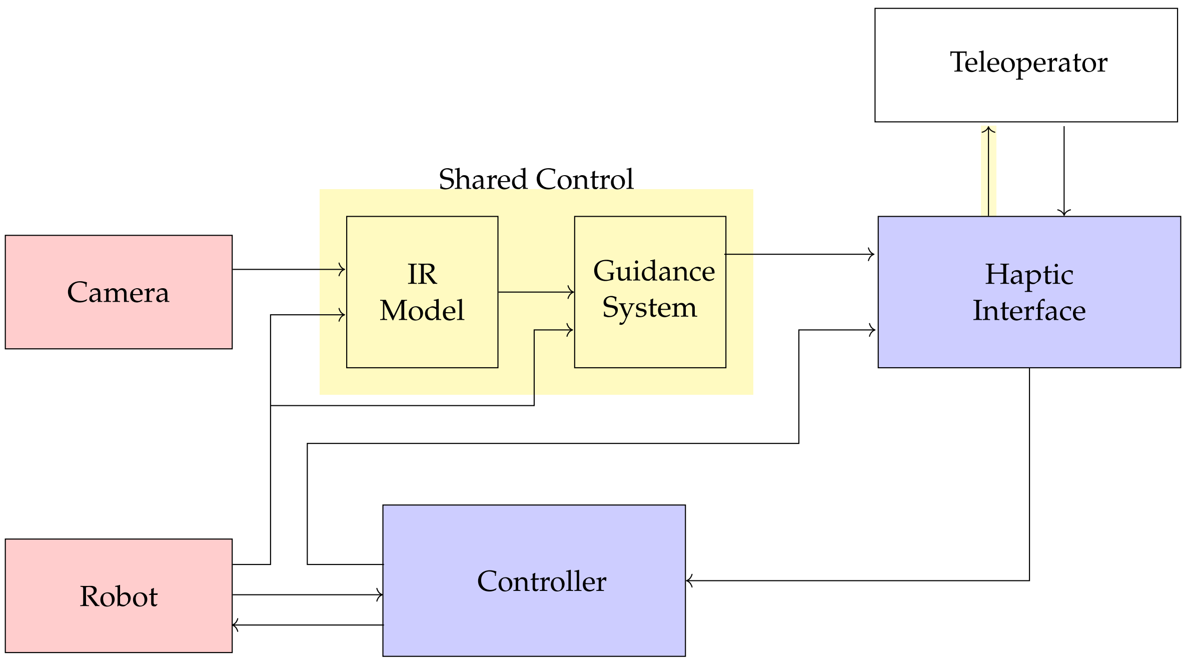


FIGURE 4.13: Online Architecture with Shared control implementation

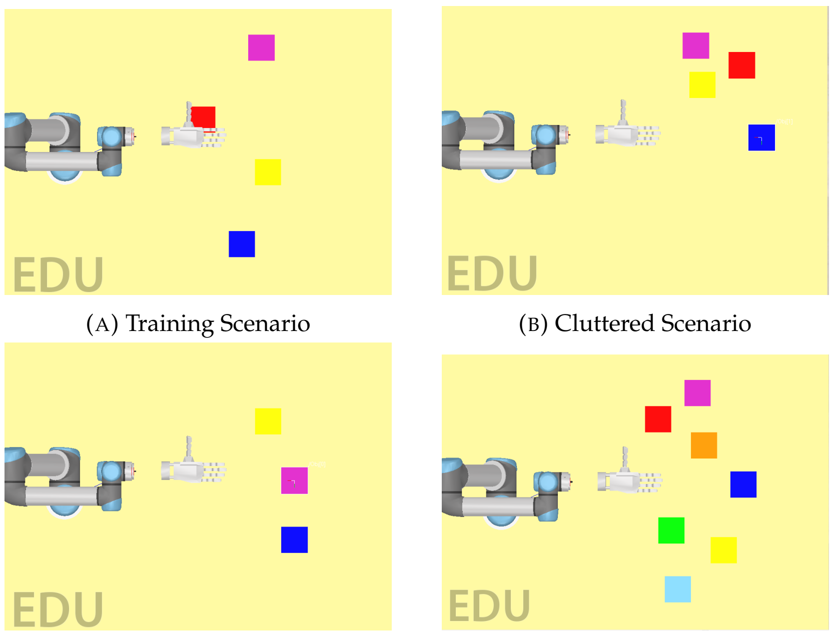
# Chapter 5

# Implementation and Results

This chapter delves into the practical execution of the Shared Guided and Supervised Control System in both virtual and real-world environments, aiming to validate its efficacy and robustness.

## 5.1 Virtual Experimental Setup

Three distinct experiments were conducted in the simulation environment to comprehensively evaluate the performance of the implemented method.To prevent any bias, all of the experiments, apart from the second one, were carried out by a novel operator.



(A) Training Scenario (B) Cluttered Scenario

(C) Three objects scenario (D) Seven objects scenario

FIGURE 5.1: Virtual Trials Scenario

Evaluating the prediction correctness was the initial step for validating and testing the Shared Control System. The metric used to better indicate its effectiveness was the True positive rate (TPR).
TPR is a performance metric used to evaluate the effectiveness of binary classification models in machine learning.
In this case, it represents the proportion of positive instances that were correctly

predicted as positive by the model.
The evaluation of prediction trials was divided into two segments: initially, the system underwent rigorous testing utilizing seven distinct variations of the LSTM model, each characterized by a different input timeseries length, with the sampling frequency of 100 HZ.
Since the goal of the first part was to evaluate the accuracy of the prediction, no guidance was applied.

### 5.1.1 Evaluation Prediction without guidance

Twenty different tests were executed by a novel operator, each time moving towards a, not predetermined, goal object, the trial lasted until reaching it.
For ten trials the test was kept as in the training scenario, while in the remaining ten trials the objects were randomly moved each time, while keeping always the same number of objects, resulting in potentially cluttered scenarios.
This initial test was crucial for evaluating which of the seven trained models was the most effective in recognizing the operator's intention.

| **Time** | **Trials** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 20 | 68.5% | 72.0% | 86.1% | 87.9% | 86.4% | 94% | 87.3% | 80.7% | 78.0% | 84.3% |
| 40 | 94.5% | 92.2% | 93.1% | 99.0% | 96.0% | 100% | 97.3% | 91.0% | 96.1% | 94.3% |
| 60 | 86.0% | 84.3% | 89.1% | 79.8% | 85.8% | 85.3% | 88.7% | 96.1% | 84.3% | 90.9% |
| 80 | 86.8% | 83.2% | 85.0% | 85.8% | 86.3% | 90.9% | 87.0% | 80.7% | 86.1% | 84.3% |
| 100 | 89.5% | 92.4% | 93.2% | 90.0% | 96.4% | 95.9% | 96.3% | 84.3% | 95.7% | 94.3% |
| 120 | 89.5% | 89.1% | 91.0% | 86.3% | 93.4% | 90.2% | 94.5% | 82.0% | 93.8% | 92.9% |

TABLE 5.2:
Prediction with no guidance (Training Scenario)

| **Time** | **Trials** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 20 | 85.1% | 83.2% | 68.6% | 88.2% | 82.5% | 84.8% | 84.2% | 86.7% | 84.9% | 73.0% |
| 40 | 95.6% | 93.5% | 97.6% | 95.2% | 96.2% | 94.1% | 95.8% | 96.0% | 94.9% | 95.1% |
| 60 | 88.1% | 87.6% | 92.2% | 81.5% | 90.8% | 88.2% | 89.7% | 92.9% | 87.1% | 90.2 |
| 80 | 85.9% | 83.0% | 87.9% | 88.9% | 87.5% | 84.8% | 85.2% | 86.7% | 84.9% | 87.1% |
| 100 | 93.0% | 90.3% | 94.5% | 93.0% | 94.6% | 89.2% | 95.0% | 96.1% | 94.0% | 94.6% |
| 120 | 82.4% | 93.0% | 90.4% | 88.3% | 92.0% | 88.8% | 87.4% | 91.7% | 92.9% | 80.4% |

TABLE 5.4:
Prediction with no guidance (Random Scenario)

| Timestamps | TPR % |
|---|---|
| 20 | 82.82% |
| **40** | **95.19%** |
| 60 | 88.82% |
| 80 | 87.31% |
| 100 | 93.63% |
| 120 | 89.84% |

TABLE 5.5: True Positive Rate of each model

Tab. 5.2 and Tab. 5.4 exhibit analogous outcomes, serving as evidence of the effectiveness of the training procedure, thus confirming the avoidance of both overfitting and underfitting of the model.
The overall True Positive Rate (as referenced in Tab. 5.5) functions as a pivotal metric, elucidating the efficacy of diverse models in accurately capturing the operator's intention. Notably, the 20 timestamp model stands as an exception, exhibiting inadequacy attributed to its temporal sequence being insufficient to comprehensively represent the intended actions.
Conversely, the model utilizing a 40 timestamp input signal showcased superior performance compared to others. It struck an optimal balance between providing a sufficiently long temporal input and maintaining flexibility.

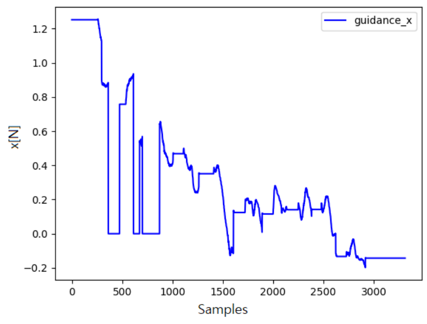
### 5.1.2 Evaluation Prediction with guidance

After selecting the optimal LSTM model, the Shared Control System underwent comprehensive testing, incorporating the integration of Haptic guidance.
To assess the system's robustness and efficacy in varied operational environments, six additional tests were conducted in randomized and potentially cluttered scenarios. These tests aimed to evaluate the effectiveness of the guidance system in enhancing the operator's control and maneuverability, particularly in complex environments.

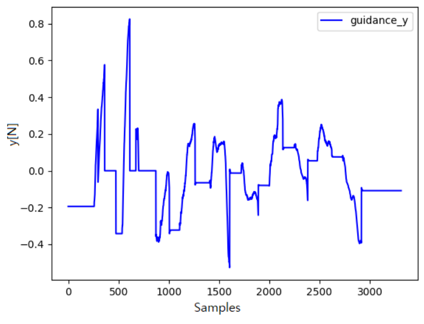
| Trial with 40 timestamp | True Positive Rate (%) |
|---|---|
| 1 | 95.8% |
| 3 | 97.2% |
| 4 | 95.7% |
| 5 | 96.1% |
| 6 | 94.1% |
| 7 | 96.0% |
| **TPR** | **95.82%** |

TABLE 5.6: True Positive Rate for 40 timestamp Model

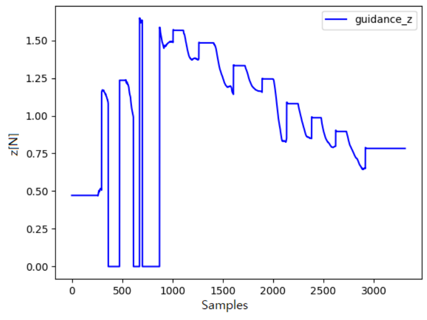
Tab. 5.6 illustrates a slight increase in prediction accuracy, mostly due to an improvement in teleoperation. By subjecting the operator to an attractive force, terratic behaviors during teleoperation are significantly mitigated. This force imposition serves to stabilize the control process, enabling smoother and more predictable teleoperation; leading to an overall improvement in the accuracy of predictions. The graph below depicts the trajectory of the guidance feedback along the three axes over the whole trial. Noticeable is its relation with respect to the prediction, especially in the initial phase. It's important to note that the graph doesn't fully depict the magnitude of the forces, as they are further multiplied by a constant before being applied to the haptic device. This constant is directly tunable by the operator within specified constraints, thereby ensuring a customized user experience.



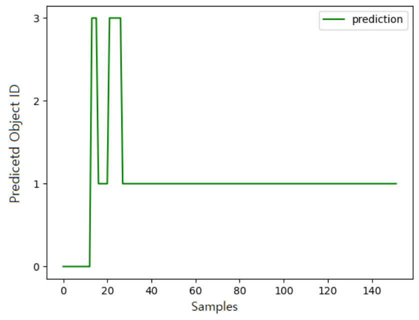
(A) Guidance along $x$ axis



(B) Guidance along $y$ axis



(C) Guidance along $z$ axis



(D) Prediction of trial

FIGURE 5.2: Virtual Trials Scenario

### 5.1.3 Skilled operator with guidance

Six tests, in the same scenario of 4, were done to evaluate the improvement in the overall completion task. The first four towards object '1', the furthermost and most complex to reach, and the remaining two towards object '0', the closest (see Fig.5.2).

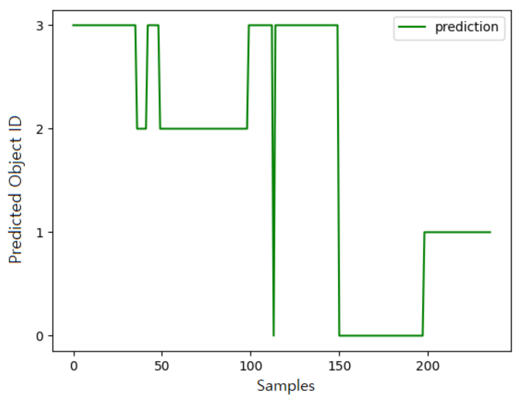
| **Trials** | **Guidance [s]** | **No Guidance [s]** |
|---|---|---|
| 1 | 31.02 | 33.93 |
| 2 | 30.05 | 33.06 |
| 3 | 37.28 | 41.97 |
| 4 | 30.10 | 34.47 |
| 5 | 27.68 | 27.09 |
| 6 | 29.79 | 30.17 |
| **Average Time [s]** | | |
| Tot | 30.99 | 33.44 |

TABLE 5.7: Average Percentage of Each Row

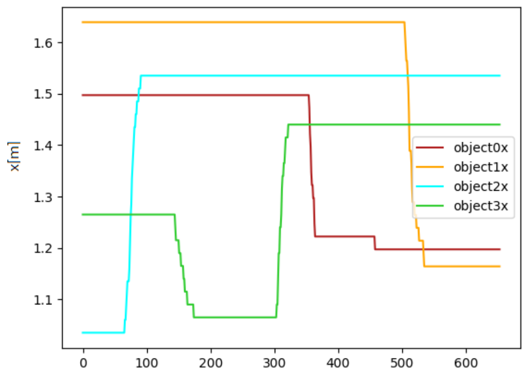
Significantly evident is an average reduction in task completion time by 7.3%

#### Dynamic Scenario

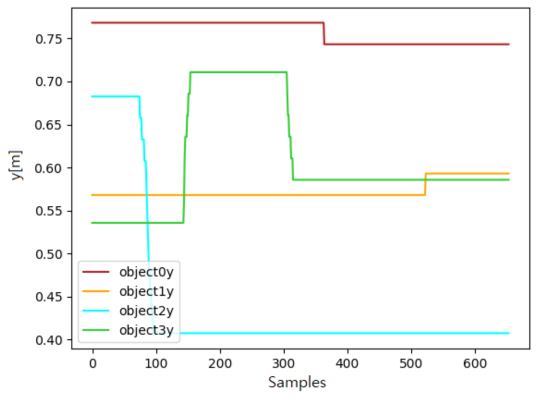
The final experiment aimed to validate the scalability of the Intention Recognition Model (IRM) and its capability to handle moving objects. In the IRM each LSTM model is associated with an object, enabling it to handle scenes with different numbers of objects. To test it, two scenes, with three and seven objects, were prepared (as depicted in Fig. 5.2).
The only notable observation was a slight reduction of prediction accuracy in the scenario with seven objects, achieving an overall accuracy of 94.7%. This decrease in accuracy was predominantly attributed to scene cluttering rather than an inherent limitation in the model's scalability.
The end effector was kept still in the position ($x = 1.35, y = 0.61$) to test the capability of handling moving objects while the objects were moved.
As shown in Fig.5.3, the shift in prediction coincides with the movement of the object.



(A) Predictions



(B) Object movement along $x$ axis



(C) Object movement along $y$ axis

FIGURE 5.3: Predictions and Object movement along $xy$ plane

## 5.2 Real Experimental Setup

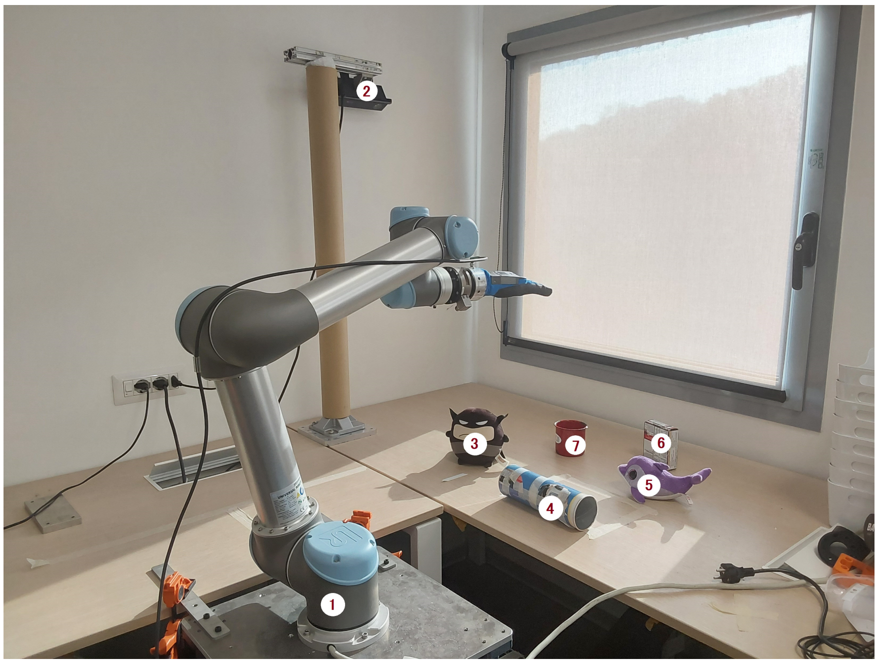


FIGURE 5.4: Real Experimental Setup.Ur5 robotic arm (1), Kinect Two RGB-D camera (2) and the Objects are indicated with IDs

As shown in Fig. 5.4, the experimental setup includes the Universal Robot UR5 robotic arm, a qb SoftHand attached to the end effector. A Kinect Two RGB-D camera was used to detect the objects. The objects chosen for our experimental setup are everyday objects; in particular, some of them (object IDs: 4, 6 and 7) belong to the YCB Benchmark Objects[6]. Table 5.8 reports the object characteristics.

During the experiments, the camera was positioned opposite the robot's base to capture a top view of the workspace.
Initially, the table was detected by identifying the dominant plane in the scene. A table-top pipeline for object detection (ORK) [23] then isolated and extracted clusters of 3D points belonging to each object on the table, as shown in Fig. 5.5.
These clusters were treated as separate objects, and their bounding boxes were computed using the oriented bounding box method provided by the Point Cloud Library (PCL)[26] (see Fig.5.6).

TABLE 5.8: Properties of tested objects.

| ID | Object | Size (mm) | Mass (g) |
|---|---|---|---|
| 3 | plush | $140 \times 70 \times 140$ | 102 |
| 4 | chips can (YCB) | $\varnothing 66 \times 230$ | 165 |
| 5 | toy dolphin | $80 \times 200 \times 90$ | 84 |
| 6 | pudding box (YCB) | $35 \times 110 \times 89$ | 187 |
| 7 | metal mug (YCB) | $\varnothing 90 \times 80$ | 118 |

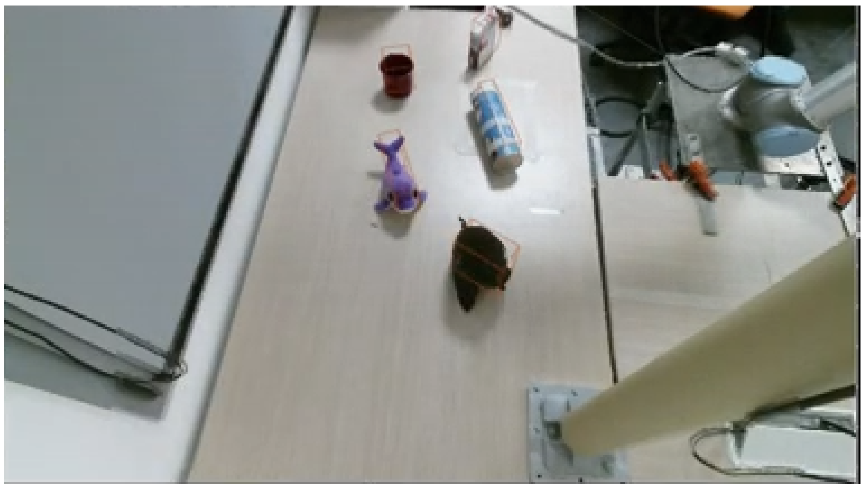


FIGURE 5.5: Visual real camera and bounding boxes

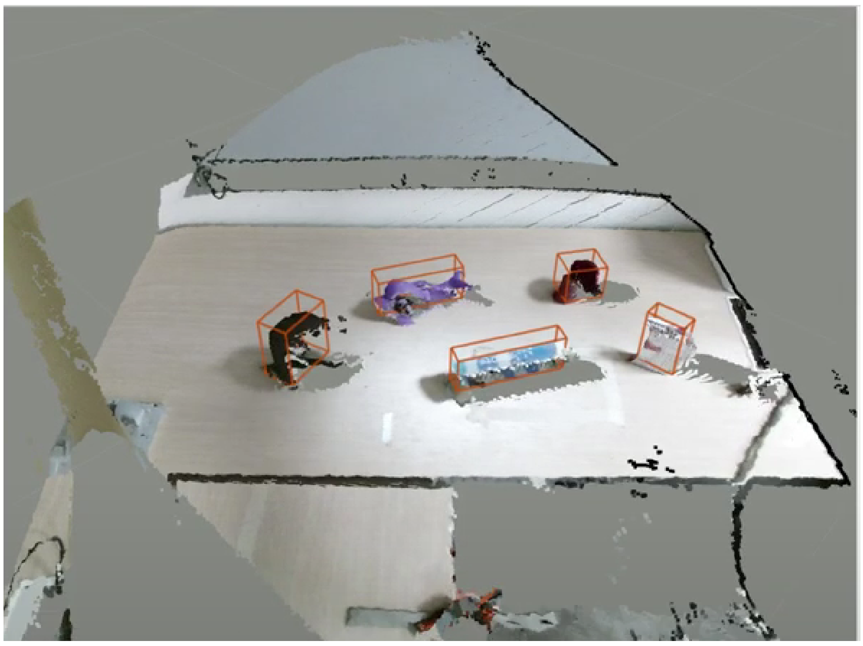


FIGURE 5.6: Point Cloud and bounding boxes (represented in the World reference frame)

At the onset of the trial, the bounding box parameters were recorded. This preemptive measure was necessitated by potential interference between the robotic arm and the camera, resulting in inaccurately defined bounding boxes. As elaborated earlier, a novel operator conducted the experiment with the sole objective of grasping two specific objects.

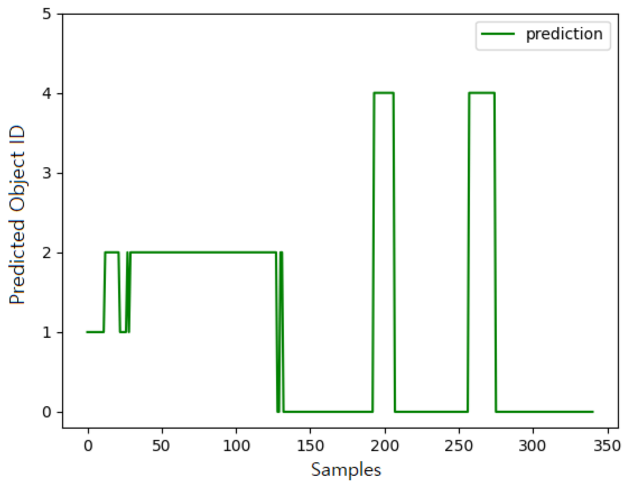
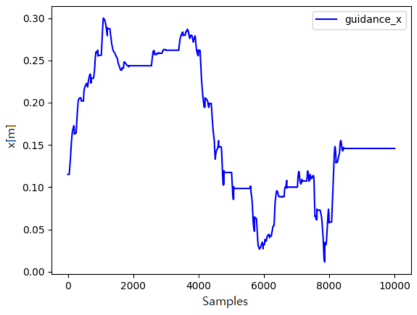
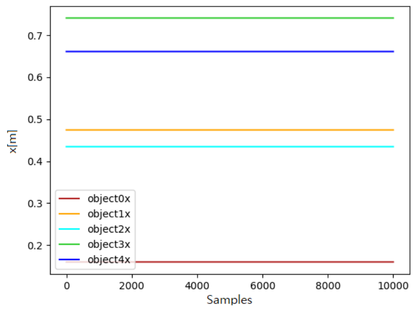


FIGURE 5.7: Predictions

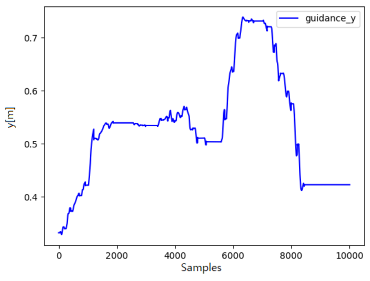


(A) $x$ axes

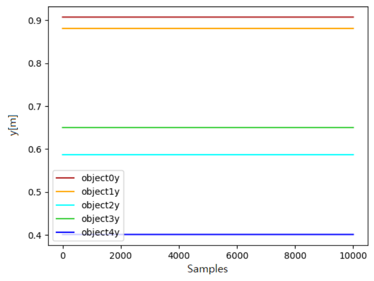


(B) $x$ axes

FIGURE 5.8: End effector and Object, $x$ axes (world frame)

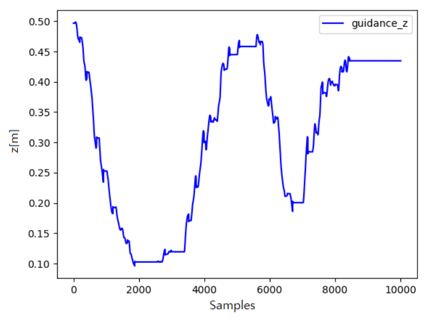


(A) $y$ axes

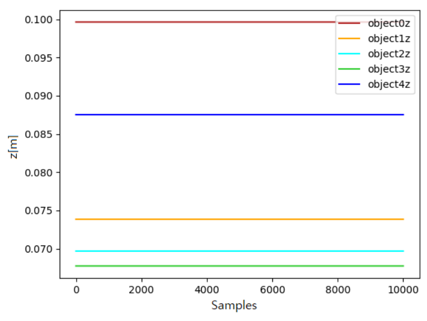


(B) $y$ axes

FIGURE 5.9: End effector and Object, $y$ axes (world frame)



(A) $z$ axes



(B) $z$ axes

FIGURE 5.10: End effector and Object, $z$ axes (world frame)

In Fig. 5.7, the output generated by the Intention Recognition Model showcases a high degree of fidelity to the user's intended actions, with occasional deviations observed in isolated outlier instances.

Further evidence is provided by analyzing the trajectory of the end effector in relation to the probability graph. Initially, the user approaches the first object and then moves towards the second object after grasping and releasing it. The desired object label can be inferred by comparing the end effector's values, particularly along the $x$ and $y$ axes, with the positions of the objects on the table. This inference can be made during two critical points in the teleoperation: upon reaching the first object and at the conclusion of the teleoperation when arriving at the second object. The inferred label corresponds to the one indicated in the prediction graph. This experiment incontrovertibly demonstrates the correctness and generalizability of the shared control system. It has been trained by an expert operator in a virtual simulation and successfully tested by a naïve operator in a real-world environment.

# Chapter 6

# Discussion and Future Works

The thesis introduced a Shared Guided and Supervised Control System devised to aid operators in the teleoperation a robotic arm. We utilized deep neural networks to integrate feature learning and establish intention prediction alongside Artificial Potential Fields for efficient haptic guidance. Notably, the system incorporated an LSTM network renowned for its effectiveness in handling multivariate time series, providing consistent and reliable intention recognition. Using only one LSTM model, the framework showcased scalability and minimal training duration, facilitating straightforward personalized retraining.
Additionally, the model effectively transitioned from being trained with expert operator teleoperations in a virtual scenario to enabling a naïve user to teleoperate in a real-world setting.
Additionally, the system exhibited robustness in handling dynamic scenarios by accurately adapting to the movement of objects and predicting accordingly. Haptic guidance, providing feedback to the user, demonstrated efficacy in reducing the overall time of each trial while serving as necessary assistance for inexperienced operators.
Looking ahead, the research lays the groundwork for future investigations and advancements. One crucial avenue for exploration involves meticulous parameter tuning for specific scenarios, such as highly cluttered scenarios (more than seven objects). This fine-tuning process can increase the system's prediction and performance in diverse settings. Furthermore, a filter for the robotic arm should be integrated to manage dynamic scenarios effectively in the real setup. This filter would enable the camera to accurately assess the bounding boxes of each object at every timestep. A vast case study would be needed to select the most appropriate Artificial Potential Field gradient. To further improve the system, autonomous decisions will be investigated, including introducing an objective function aimed at preserving the manipulability of the robotic arm above a specified threshold. When the manipulability falls below this threshold, the robotic arm will stop receiving input from the operator and reconfigure itself to ensure optimal manipulability, thus avoiding singularities and workspace limitations.

# Appendix A

# Support Code

## A.1 LSTM model

```python
from keras.models import Sequential
from keras.layers import  LSTM, Dense, Dropout
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.models import save_model

def build_model(self,X,Y):


        input_seq = ip.np.array(input_seq)
        label = ip.np.array(Y)


        model = Sequential()
        model.add(LSTM(100, activation='tanh', return_sequences=True,
    input_shape=(self.timestep, 2)))
        model.add(LSTM(50, activation='tanh'))
        model.add(Dropout(0.25))
        model.add(Dense(20, activation='tanh'))
        model.add(Dropout(0.25))
        model.add(Dense(1, activation='sigmoid') )


        cp = ModelCheckpoint('LSTM/', save_best_only=True)
        model.compile(optimizer=Adam(self.learning_rate), loss='
    binary_crossentropy', metrics=['accuracy'])
        model.fit(input_seq, label, epochs=self.epochs ,callbacks=[cp])


        print(model.summary())
        save_model(model, 'model_LSTM.h5')
```

LISTING A.1: LSTM model

# Bibliography

[1] Daniel Aarno and Danica Kragic. “Motion intention recognition in robot assisted applications”. In: *Robotics and Autonomous Systems* 56.8 (2008), pp. 692–705. ISSN: 0921-8890. DOI: `https://doi.org/10.1016/j.robot.2007.11.005`. URL: `https://www.sciencedirect.com/science/article/pii/S0921889007001704`.

[2] Joaquin Ballesteros et al. “A Biomimetical Dynamic Window Approach to Navigation for Collaborative Control”. In: *IEEE Transactions on Human-Machine Systems* 47.6 (2017), pp. 1123–1133. DOI: `10.1109/THMS.2017.2700633`.

[3] Rai Bijay, Matsa Amarendra, and Datta Asim. “Steer guidance of autonomous agricultural robot based on pure pursuit algorithm and LiDAR based vector field histogram”. In: *Journal of Applied Science and Engineering* 26.10 (2023), pp. 1363–1372.

[4] Amar Budhiraja. *Dropout in (Deep) Machine Learning*. Web page. Accessed on: insert$_{a}ccess_{d}ate_{h}ere$. 2016. URL: `insert_url_here`.

[5] Sylvain Calinon et al. “Learning and Reproduction of Gestures by Imitation”. In: *IEEE Robotics and Automation Magazine* 17.2 (2010), pp. 44–54. DOI: `10.1109/MRA.2010.936947`.

[6] Berk Çalli et al. “Benchmarking in Manipulation Research: The YCB Object and Model Set and Benchmarking Protocols”. In: *CoRR* abs/1502.03143 (2015). arXiv: `1502.03143`. URL: `http://arxiv.org/abs/1502.03143`.

[7] Manuel G Catalano et al. “Adaptive synergies for the design and control of the Pisa/IIT SoftHand”. In: *The International Journal of Robotics Research* 33.5 (2014), pp. 768–782.

[8] François Chollet. *Keras: Deep Learning for humans*. `https://github.com/fchollet/keras`. 2015.

[9] Coppelia Robotics. *CoppeliaSim*. `https://www.coppeliarobotics.com/`. [(cit. on p. 31)]. 2023.

[10] Zhiyong Cui et al. *Deep Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction*. 2019. arXiv: `1801.02143 [cs.LG]`.

[11] Peiyuan Fang et al. “A Maximum Entropy Inverse Reinforcement Learning Algorithm for Automatic Parking”. In: *2021 5th CAA International Conference on Vehicular Control and Intelligence (CVCI)*. 2021, pp. 1–6. DOI: `10.1109/CVCI54083.2021.9661263`.

[12] Force Dimension. *Products*. `https://www.forcedimension.com/products`. Accessed: 2024-03-24. 2024.

[13] Alberto Gottardi et al. “Shared Control in Robot Teleoperation With Improved Potential Fields”. In: *IEEE Transactions on Human-Machine Systems* 52.3 (2022), pp. 410–422. DOI: `10.1109/THMS.2022.3155716`.

[14] Alex Graves and Alex Graves. “Long short-term memory”. In: *Supervised sequence labelling with recurrent neural networks* (2012), pp. 37–45.

[15] S. Hayati and S.T. Venkataraman. “Design and implementation of a robot control system with traded and shared control capability”. In: *Proceedings, 1989 International Conference on Robotics and Automation*. 1989, 1310–1315 vol.3. DOI: 10.1109/ROBOT.1989.100161.

[16] Shervin Javdani et al. “Shared autonomy via hindsight optimization for teleoperation and teaming”. In: *The International Journal of Robotics Research* 37.7 (2018), pp. 717–742. DOI: 10.1177/0278364918776060. eprint: https://doi.org/10.1177/0278364918776060. URL: https://doi.org/10.1177/0278364918776060.

[17] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *The international journal of robotics research* 5.1 (1986), pp. 90–98.

[18] Karan Khokar et al. “A novel telerobotic method for human-in-the-loop assisted grasping based on intention recognition”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 4762–4769. DOI: 10.1109/ICRA.2014.6907556.

[19] Gaofeng Li et al. “The Classification and New Trends of Shared Control Strategies in Telerobotic Systems: A Survey”. In: *IEEE Transactions on Haptics* 16.2 (2023), pp. 118–133. DOI: 10.1109/TOH.2023.3253856.

[20] Yangtao Li et al. “The Prediction of Dam Displacement Time Series Using STL, Extra-Trees, and Stacked LSTM Neural Network”. In: *IEEE Access* 8 (2020), pp. 94440–94452. DOI: 10.1109/ACCESS.2020.2995592.

[21] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[22] Iram Noreen, Amna Khan, and Zulfiqar Habib. “Optimal path planning using RRT* based approaches: a survey and future directions”. In: *International Journal of Advanced Computer Science and Applications* 7.11 (2016).

[23] *ORK: Object Recognition Kitchen*.

[24] qbrobotics. *qb SoftHand Research*. https://qbrobotics.com/it/prodotti/qb-softhand-research/. Accessed: 2024-03-24. 2024.

[25] Yadollah Rasekhipour et al. “A potential field-based model predictive path-planning controller for autonomous road vehicles”. In: *IEEE Transactions on Intelligent Transportation Systems* 18.5 (2016), pp. 1255–1267.

[26] Radu Bogdan Rusu and Steve Cousins. “3D is here: Point Cloud Library (PCL)”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011.

[27] Bruno Siciliano and Oussama Khatib, eds. *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer, 2008. ISBN: 978-3-540-23957-4. URL: http://dx.doi.org/10.1007/978-3-540-30301-5.

[28] R. Smits. *KDL: Kinematics and Dynamics Library*. http://www.orocos.org/kdl.

[29] Universal Robots. *UR5 User Manual*. Cited on pages 36, 41, 42.

[30] C. Warren. “Global path planning using artificial potential fields”. In: *1989 IEEE International Conference on Robotics and Automation*. Los Alamitos, CA, USA: IEEE Computer Society, May 1989, pp. 316, 317, 318, 319, 320, 321. DOI: 10.1109/ROBOT.1989.100007. URL: https://doi.ieeecomputersociety.org/10.1109/ROBOT.1989.100007.

[31] Chelsea C. White and William T. Scherer. “Reward revision for partially observed Markov decision processes”. In: *1985 24th IEEE Conference on Decision and Control*. 1985, pp. 1822–1827. DOI: 10.1109/CDC.1985.268877.

[32] Shao-Wen Wu. “Predicting User Intent During Teleoperation Using Neural Networks”. URN: http://nbn-resolving.de/urn:nbn:de:bsz:93-opus-ds-119816. Master’s Thesis. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und

Informationstechnik, 2019, p. 46. DOI: 10.18419/opus-11964. URL: http://elib.uni-stuttgart.de/handle/11682/11981.

[33] Yong Yu et al. “A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures”. In: *Neural Computation* 31.7 (July 2019), pp. 1235–1270. ISSN: 0899-7667. DOI: 10.1162/neco_a_01199. eprint: https://direct.mit.edu/neco/article-pdf/31/7/1235/1053200/neco_a_01199.pdf. URL: https://doi.org/10.1162/neco%5C_a%5C_01199.

[34] Brian D Ziebart et al. “Maximum entropy inverse reinforcement learning.” In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.

# *Acknowledgements*

Desidero ringraziare profondamente tutte le persone che mi son state vicino in quest'avventura.

Innanzitutto, desidero ringraziare il mio relatore, il prof. Alessandro Rizzo, e co-relatore, prof. Domenico Prattichizzo, per avermi dato la possibilità di intraprendere questo progetto.

Ringrazio Enrico per essere stato un ottimo mentore, rendendo quest'esperienza veramente memorabile. Ringrazio anche Valerio e Chiara per l'aiuto fornitomi in questi mesi.

Un ringraziamento speciale va alla mia famiglia, mi avete sempre incoraggiato in ogni mia scelta, rendendo possipile tutto ciò. E' merito vostro se ho potuto affrontare ogni prova e avventura con la spensieratezza w tranquillità necessaria.

Ringrazio anche la mia seconda famiglia, Maldestramente Poeti: Biri,Matte e Tommi.
Gli obbiettivi che raggiungo varrebbero niente se non potessi condividerli con voi.
A questa e a tutte le prossime avventure insieme, sperando di osservare sempre la vita con il nostro solito tocco di romanticismo.

Un ringraziamento al gruppone Kanerva, vivermi Torino e mille feste con voi è stato veramente magnifico.

Un grazie anche a te Bollaz e per tutte le passeggiatine fatte assieme per sanse.

A voi vecchi amici di una vita: Gian, Pier, Ale, Fede ed EVo che siete e sarete sempre una motivazione a spingere per ogni obbiettivo che mi prefisserò.

A voi compagni di università un grandissimo grazie:ad Ali, la mia compagna di lab preferita, Matte, che mi ha spertizzato tutte le volte necessarie, Nicoli e Franco,i miei due patatoni del profondo sud, Luca, sinceramente la persona più veloce che abbia mai visto e Nicco, maestro di vita e di freestyle.

Un ringraizamento a parte va a voi due Vale e Morgy. Vale, passare le giornate con te fra avventure, baretti e film trash è tutto ciò che potevo desiderare, quindi grazie per essermi ed essere sempre al mio fianco.
Morgy, tu ti meriti il tanto agognato capitolo di Lodi. Mai potevo immaginare che quel ragazzo dal capello lungo e dubbia moralità potesse svoltarmi così tanto l'esperienza Universitaria.
Sarei un quarto dell'ingegnere che sono ora senza tutti i folli laboratori fatti assieme, le ore sprecate in cerca di insensate teorie alternative e le produzioni video dall'altissima qualità. Sei un folle genio da cui ho appreso veramente tanto e la tua instancabile passione mi ha veramente contagiato.
Detto ciò, spero di finire un giorno a lavorare per una tua startup.
Grazie ancora Folle Scenziato Pazzo che non sei altro.

Un rigraziamento speciale ai Motociclisti amanti della meccatronica, per i soldi assegnatimi. Sappiate che tiferò sempre per voi!

. Che la pace sia con voi

*Francesco*

Pellicano Kid

*the fog is coming*