AMERICAN UNIVERSITY OF BEIRUT

3D AUTOCOMPLETE: ENHANCING UAV TELEOPERATION WITH AI IN THE LOOP

by
BATOOL ALI IBRAHIM

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Electrical and Computer Engineering
of Maroun Semaan Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
November 2023

AMERICAN UNIVERSITY OF BEIRUT

3D AUTOCOMPLETE: ENHANCING UAV TELEOPERATION WITH AI IN THE LOOP

by

BATOOL ALI IBRAHIM

Approved by:

Dr. Imad H. Elhajj, Professor Electrical and Computer Engineering	Advisor [illegible]
Dr. Daniel Asmar, Professor Mechanical Engineering	Co-Advisor [illegible]
Dr. Naseem Daher, Associate Professor Electrical and Computer Engineering	Member of Committee [illegible]
Dr. Dany Abou Jaoude, Assistant Professor Mechanical Engineering	Member of Committee Jaoude

Date of thesis defense: November 14, 2023

AMERICAN UNIVERSITY OF BEIRUT

THESIS RELEASE FORM

Student Name:	Ibrahim	Batool	Ali
	Last	First	Middle

I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of my thesis; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes

✓ **As of the date of submission of my thesis**

___ **After 1 year from the date of submission of my thesis .**

___ **After 2 years from the date of submission of my thesis .**

___ **After 3 years from the date of submission of my thesis .**

Signature	Date
BATOOL	January 8, 2024

Acknowledgements

First of all, my deepest gratitude and honor goes to the almighty God who gifted me the health, strength, and patience that have enabled me to follow my academic pursuits and complete this thesis successfully.

I would like to express my appreciation and thanks to my advisors Prof. Imad Elhajj and Prof. Daniel Asmar for their guidance, support, and effort throughout my work on this thesis. It was my pleasure to have the opportunity to work with such supportive, inspiring, motivating, and tolerant professors.

I express my sincere thanks to the members of my dissertation committee, Prof. Naseem Daher and Prof. Dany Abou Jaoude for their great attention, discussion, ideas, and feedback that have been invaluable.

Finally, I express my heartfelt gratitude and love to my family for their silent prayers and continuous support throughout the process of researching and writing this thesis. This accomplishment would not have been possible without their love.

Abstract of the Thesis of

Batool Ali Ibrahim for Master of Engineering
Major: Electrical and Computer Engineering

Title: 3D Autocomplete: Enhancing UAV Teleoperation with AI in the Loop

Manually teleoperating a flying robot can be a demanding task, especially for users with limited levels of experience. This is primarily due to the nonlinear properties of such robots in addition to the difficulty of controlling various degrees of freedom at the same time. To help mitigate such limitations, this thesis proposes a framework named '3D Autocomplete' that aids users in teleoperation. It uses artificial intelligence to predict in real-time the operator's intended motion, and mixed reality to convey the predicted motion to the user. Previous Autocomplete systems focused on different 2D motions in the same plane (line, arc, sine). However, since many drone tasks take place in a three-dimensional environment, 3D Autocomplete primarily assists users in navigating challenging 3D motions around 3D geometric primitives (cylinder, cone, and box). During teleoperation, the framework uses a real-time change point detection algorithm called 'just-in-time' to monitor the user's input, and deep learning to early predict the motion type as one of predefined 3D motions. Then, the predicted motion is augmented into the first person view in real-time using a virtual reality headset. Finally, if the users accept the proposed trajectory, 3D Autocomplete completes their desired motion autonomously. We validate the proposed mixed reality teleoperation approach by conducting different experiments on a simulated quadrotor. The results illustrate 3D Autocomplete advantages over traditional teleoperation methods through both subjective and objective evaluations conducted via human subject experiments. The system achieves its primary goal of reducing the users workload, and improves task completion time and covered distance by at least 30% compared to traditional teleoperation. Moreover, it enhanced the system performance and trajectory smoothness by approximately 50%.

Table of Contents

ACKNOWLEDGEMENTS **1**

ABSTRACT **2**

ABBREVIATIONS **7**

1 Introduction **8**

2 Literature Review **11**
- 2.1 Assisted Teleoperation 11
- 2.2 Deep Learning in Teleoperation 13
- 2.3 Mixed Reality in HRI 14

3 Proposed Approach **16**
- 3.1 Overview of System Components 16
  - 3.1.1 Change Points Detection: Just-in-time Algorithm 17
  - 3.1.2 Motion Classifier: Deep Learning Model 19
    - 3.1.2.1 Data 19
    - 3.1.2.2 Model 22
  - 3.1.3 Variable Input Size 24
  - 3.1.4 Mixed Reality User Interface 26
  - 3.1.5 Motion Synthesis 27
    - 3.1.5.1 Motion Around a Cylinder: Cylindrical Helix 27
    - 3.1.5.2 Motion Around a Cone: Conical Helix 28
    - 3.1.5.3 Motion Around a Box: Rectangular Helix 29
  - 3.1.6 Autonomous Navigation 29

4 Results and Experiments **30**
- 4.1 Preparing the Data 30
- 4.2 Deep Learning Model 30
  - 4.2.1 Training 30
  - 4.2.2 Offline Evaluation 31
  - 4.2.3 Real Time Simulations 32
- 4.3 Implementation Details 34
  - 4.3.1 AR.Drone 2.0 34
  - 4.3.2 Mixed Reality 34

4.3.3 VR-headset 35
4.4 Experiments 35
4.5 Results 36
4.5.1 Subjective Evaluation 36
4.5.2 Objective Evaluation 39

5 Discussion **42**
5.1 Findings and Achievements 42
5.2 Limitations and Possible Enhancements 43

6 Conclusion **45**

Bibliography **47**

Illustrations

1.1 3D Autocomplete 9

3.1 3D Autocomplete architecture 16
3.2 Spotting important changes in the user 3D motion 17
3.3 Geometrical interpretation of dot product 18
3.4 Gazebo virtual environment 20
3.5 Simulated AR.Drone 2.0. quadrotor 21
3.6 Some of the considered shapes while collecting the dataset 21
3.7 Deep learning model architecture 22
3.8 Downsampling Algorithm concept 25
3.9 Proposed user interface for 3D Autocomplete 26
3.10 Motion synthesis for 3D motions 28

4.1 Just-in-time algorithm 31
4.2 Learning curves 31
4.3 Confusion matrix 32
4.4 Gazebo Testing Environment 33
4.5 Components of the proposed MR user interface 34
4.6 Oculus Quest 2 35
4.7 Testing environment 36
4.8 NASA-TLX Survey Scores 38
4.9 3D Trajectories 41

Tables

3.1 Overview of Neural Network Components 23

4.1 Real time simulation results of the DL Model. 32
4.2 Quantitative results of the human-subject tests 39
4.3 Prediction and Motion synthesis processing time 40
4.4 P-values of Quantitative Results 40

Abbreviations

UAV	Unmanned Aerial Vehicle
DoF	Degrees of Freedom
MR	Mixed Reality
VR	Virtual Reality
UI	User Interface
AI	Artificial Intelligence
CDP	Change Point Detection
HRI	Human Robot Interaction
FOV	Field Of View
DL	Deep Learning
ROS	Robot Operating System
CNN	Convolutional Neural Network
GRU	Gated Recurrent Unit
PCA	Principle Component Analysis

Chapter 1

Introduction

Human-Robot Interaction (HRI) is an evolving field in robotics [1]. It arises from the need to enhance the communication between humans and robots, primarily in application areas that require human-robot collaboration [2]. A significant aspect of HRI is teleoperation, which has been widely utilized to carry out tasks in hazardous and inaccessible environments through remotely controlled robots. However, until now, robot teleoperation from a remote location is still a challenging task, especially when controlling flying robots, known as Unmanned Aerial Vehicles (UAVs) [3].

Despite the fact that UAVs are already used in various applications such as load transportation [4], farming [5], and surveillance [6]; their teleoperation remains a demanding task that requires a certain level of experience. This is due to several factors: first, the under-actuated characteristics of these robots require mapping a large number of actuator degrees of freedom on the robot's side into a smaller set of control inputs on the user's side [7]. This mapping is often challenging and demands extensive training for users to understand the interactions between the control inputs and the robot's behavior. Second, the UAV control inputs are not always intuitive. In other words, the operators' movements using devices like joysticks or haptic controllers may not directly correspond to the robot's actual motion. Third, the limited perceptual capability and sensory feedback at the operator's end limit the user's awareness of the surroundings [8]. This directly impacts the system performance by compromising the operator's ability to make informed decisions and respond to dynamic environmental changes. All these contributing factors significantly impact robot teleoperation and may lead to frequent crashes and potential damage to the robot or the surrounding environment. Accordingly, for the purpose of mitigating these limitations and assisting the users while remote controlling UAVs, Autocomplete was proposed [9].

Autocomplete aids UAV operators by predicting their desired motions and completing them in an autonomous manner. In more detail, while a person is teleoperating a UAV, Autocomplete uses the input joystick commands (input velocity vectors) to predict the user's intended-motion type with the help of Artificial Intelligence (AI) which classifies the motion as one of several defined motion primitives. Then, the user is shown the predicted motion through a User Interface (UI) and has the option to accept, reject, or ignore. When the predicted motion is accepted, the

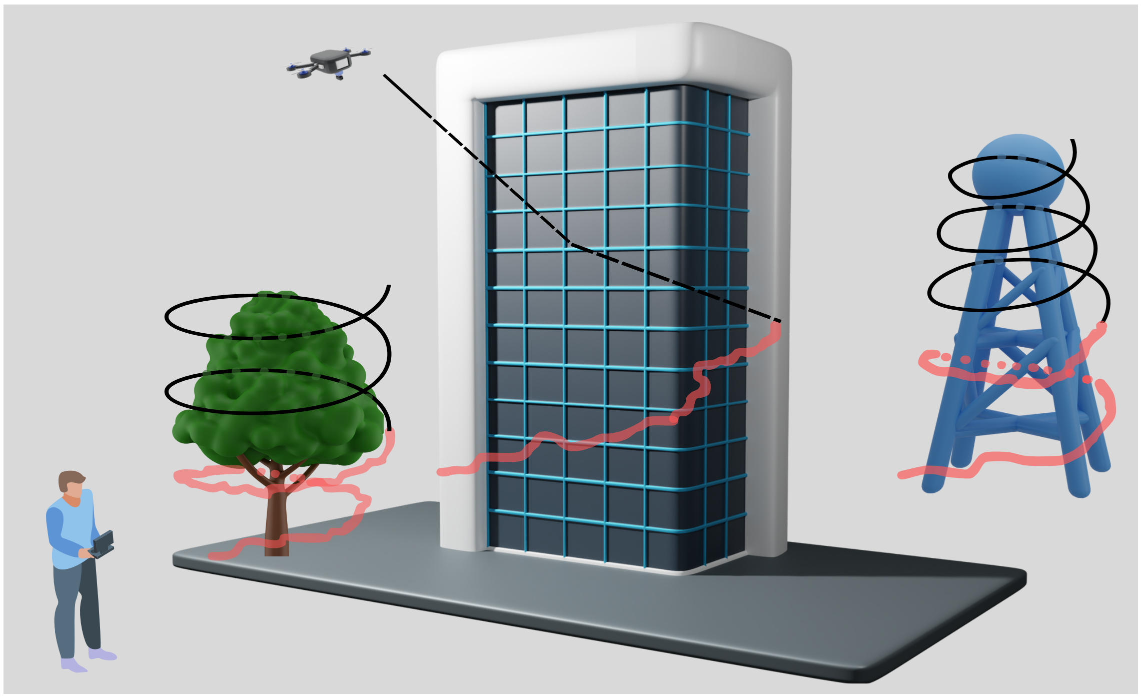


Figure 1.1: 3D Autocomplete: the user's manual motions are represented in red, whereas the 3D Autocomplete autonomous motions are represented in black. The motion around the tree is considered as a motion around a cylinder, around the building as a box, and around the antenna as a cone.

system synthesizes and completes the user desired motion autonomously. However, if the users reject or ignore the predicted motion, they retain the control of the UAV using the joystick controller.

In previous works, Autocomplete was implemented using different approaches. Initially [9], a support vector machine was used to classify the user's partial motions, and a UI was created to augment the predicted motions in the UAV's camera video stream. The conducted experiments in a Gazebo simulated environment on ROS showed that trajectories conducted using Autocomplete are closer to the optimal trajectories than using conventional teleoperation. The framework was upgraded in [10], where a trained deep neural network was used, and mixed reality was introduced to the system to provide the estimated trajectories to the users in an intuitive manner. The deep learning model was improved using online learning [11] and partial-feedback [12], where the user's acceptance or rejection of the motion were used to improve the model. In all of these works, the predicted and synthesized motion primitives were focused on 2D motions, including lines, arcs, and sinusoidal motions. However, teleoperating UAVs in a real-world requires a framework that allows the UAV to perform more complex motions and navigate different 3D trajectories, like flying the UAV around a tree or a building. Such tasks are more difficult for the user and the risk of accidents increase. Accordingly, the need for autocomplete autonomy is more impactful to enhance the safety of UAV operations and reduce the work load on human operators.

In this regard, to make teleoperation of UAVs in complex scenarios more accessible and practical, this thesis presents a 3D Autocomplete framework that handles more complex motions. Mainly, 3D motions around 3D geometric primitives commonly encountered in any 3D environment: cylinder, cone, and box. Handling such primitives enables the system to negotiate 3D objects with similar geometrical properties (see Fig. 1.1), like buildings (box) or towers (cylinder); in addition to more complex shapes that are a combination of those primitives. For example, a motion around a tree can be considered as a motion around a cylinder followed by a motion around a cone.

The proposed 3D Autocomplete framework relies on trained deep neural networks to classify the motion type based on the input velocity vectors. This requires collecting a diverse dataset of 3D motions for model training. Moreover, since motions can have different scales depending on the geometric primitive in which the UAV is flying around (large building versus small tree), my framework uses a just-in-time early motion prediction algorithm that monitors the UAV's motion and allows the system to determine when to predict the motion type in real-time, ensuring scale-agnostic operation. Finally, the predicted motions are communicated to the user through a mixed reality UI and a VR headset for a more efficient teleoperation experience.

The contributions of this thesis are summarized as follows:

- I propose a system that detects the user intended 3D motion from the joystick control commands based on trained deep neural networks.
- I propose a just-in-time algorithm to perform an early prediction of the user's motion and to make the framework scale agnostic.
- I deal with the prediction of variable motion sizes instead of considering a constant time interval and a fixed number of velocity vector samples.
- I propose a motion synthesis algorithm that generates the trajectory needed to complete the user's desired motion autonomously after predicting its type.
- I introduce a mixed reality UI that enables the operator to observe the 3D predicted motions augmented directly to the surrounding environment scene using a MR headset.

This thesis report is structured as follows: Chapter 2 covers the related work in Autocomplete, deep learning, and mixed reality in teleoperation. Chapter 3 presents the proposed approach for 3D Autocomplete, including detailed descriptions of the methodology, algorithms, and models employed. Chapter 4 presents and discusses the obtained results considering both quantitative and qualitative assessments. Chapter 5 provides a concise overview of the framework's achievements, limitations, and opportunities for future enhancements. Finally, chapter 6 includes the conclusion of this report and suggests potential paths for future research.

Chapter 2

Literature Review

This thesis introduces 3D Autocomplete, an assisted teleoperation framework designed to support users during teleoperation. It combines deep learning to predict the user's motion type and mixed reality for user communication. Accordingly, to substantiate my contributions, this chapter conducts a review of the state-of-the-art and recent developments in assisted teleoperation, deep learning, and mixed reality within the context of teleoperation.

2.1 Assisted Teleoperation

Given the widespread usage of teleoperated robots, assistive teleoperation has been addressed using different approaches. Zhang et al. [13] use Implicit Neural Field (INF), which relies on neural networks to assist humans in teleoperating surgical robots. The system takes as an input the ′leading hand′ commands conducted by the operator, and corrects them with the help of INF to prevent any collisions with the human tissues. In this approach, the users have to teleoperate the robot manually throughout the entire task, while the system is optimizing their path. In contrast, my proposed 3D Autocomplete requires only a partial user motion as a manual input, and then can predict the desired motion type and complete it autonomously.

Maeda in [14] proposes a policy blending with primitives strategy that combines the control policies of the robot and the human using dynamical movement primitives. These primitives allow the robot to execute complex movements by breaking them down into sub-movements while blending the real-time user inputs. A similar approach is introduced in [15], where Gottardi et al. present a system that generates a sequence of safe points to reach a specific goal while considering the user's intentions. The proposed systems are focused on applications that involve targeting goals or dynamic obstacle avoidance. However, it is not suitable for other application types such as surveillance or robot racing.

Havoutis et al. [16] present a framework that aids underwater vehicle teleoperation. They introduce online Bayesian non-parametric learning algorithms to build models of manipulation based on user demonstrations. Then, model predictive control is used to execute the motions autonomously. Ewerton et al. [17] address cases when changes occur in the surrounding environment after demonstrations were made. They present a reinforcement learning algorithm, pearson-correlation-based relevance weighted policy optimization, to learn and optimize the robot trajectory distributions and assist its teleoperation in static and dynamic environments. The entire framework is used to aid in the teleoperation of a 7-DoF robot arm in a dynamic environment while executing a specific task. In the same context, Dass et al. [18] develop a policy-assisted teleoperation system to collect large datasets for training Machine Learning (ML) robotic models. The primary goal of the system is to reduce the mental load on users by automating repetitive tasks while collecting robot demonstrations. To do so, they employ a hierarchical framework with two levels: a high-level policy for subtask selection, and a low-level policy for generating robot control commands; both policies are based on deep neural networks. The human-subject experiments showed that the proposed framework improved the data collection efficiency while reducing the operators' mental load. The limitation of these approaches is that they are task-centered, which requires unique data for each task. In contrast, my proposed Autocomplete system is user-centered; it identifies the user's intended motion and completes it autonomously regardless of the task.

Aligning with the focus of my study, Wang et al. [19] address assistive teleoperation in flying robots. They propose a system that utilizes the human gaze and the remote controller commands to generate a path that satisfies the operator's intentions. The gaze captures the targeted position, whereas the remote controller input is used to identify the intended speed. Real world experiments have verified the reliability and robustness of the proposed framework. However, such system relies on the accuracy of the operator's gaze data which may be affected by several factors such as the lighting conditions or eye movements. Chen et al. [20] overcome such limitations by proposing a system that adapts to these changes using reinforcement learning. The proposed system is used to control a wheelchair-mounted Jaco arm that performs simple tasks such as controlling light switches, unlocking doors, and manipulating valves by mapping the user's gaze features to joint torques. Although human-subject tests yielded promising results, the system relies on a pre-determined set of tasks for learning, which may not always be available in real-world scenarios.

Another work is presented in [21], where Yang et al. use a selector function to choose a motion that reflects the user's intentions from a subset of motion primitives. This subset is sampled from a dense motion primitives library using importance sampling. Then, the selector function chooses the motion that has the most similar parameterization to the operator joystick inputs. The proposed framework is applied on gait systems and air vehicle (2D motions: line and arc). Although such a system produced encouraging results, it is completely user-dependent where a motion primitives library is generated for each user. My proposed Autocomplete framework is collaborative and reduces the operator workload. Moreover, it considers 3D motions instead of only 2D motions.

Several works have viewed assistive teleoperation as a control problem to avoid collisions. Perez-Grau et al. [22] introduce autonomous obstacle avoidance to the teleoperation of small UAVs to aid inexperienced rescue team members to control the UAV in inspection tasks. The semi-autonomous teleoperation system performs obstacle avoidance while planning the robot trajectory using a local planner, which continuously checks for trajectory re-planning. In case an obstacle is detected by the on-board visual sensors, it takes evasive action to keep the UAV inside the pre-set safety distances. Xu et al. [23] use a control Lyapunov function for assisting human operators to avoid potential obstacles. A collision avoidance rectifier is used to execute a constrained quadratic optimization and modify the robot parameters in order to satisfy the safety conditions. The limitation with such works is that they do not take the user's intentions into consideration, they only focus on bypassing encountered obstacles. In contrast, 3D Autocomplete is designed with the user's intention as its primary focus.

2.2 Deep Learning in Teleoperation

A considerable number of works in the robotics literature rely on deep learning for different tasks such as path and trajectory planning [24], motion control [25], surveillance [26], etc. Moreover, the literature widely introduces prediction and recognition systems for the user intentions in various robotics applications, including teleoperation of mobile robots [27], driving wheelchairs [28], teleoperation of robotic hand [29], etc. However, only few works consider deep learning for assisted teleoperation.

Based on deep learning in teleoperation, Laskey et al. [30] compares the Robot-Centric (RC) sampling to the Human-Centric sampling (HC). A grid world environment and a physical robot object singulation task are used for the comparison. The simulation results showed that policies learned with RC performed better than those learned with HC for linear support vector machines; however, this advantage disappears when highly expressive learning models such as deep models are used. Zhou et al. [31] propose an intelligent interface to aid the teleoperation of an industrial construction robot. The proposed interface reconstructs in Virtual Reality (VR) the surrounding workspace scene model. Accordingly, the human operator could manipulate the remote robotic arms with handheld controllers based on the reconstructed scene. Deep learning is introduced for object detection in the captured scene. The results of testing the proposed framework on Baxter manipulators showed that it is more efficient compared to traditional robotic teleoperation. Li et al. [32] introduce an end-to-end teacher-student deep Convolutional Neural Network (CNN). It is used for dexterous robotic hands teleoperation based on a single-depth image. The proposed network learns mappings between the joint angles of the robot hand and the depth images of a human hand to produce similar poses. Based on the teleoperation experiments done by novice users, TeachNet has shown its superiority compared to other state of art vision-based teleoperation techniques. A similar study was introduced by Zhang et al. [33], who construct a system that uses consumer-grade VR in order to teleoperate a PR2 robot even in the case of complex tasks. They concatenate RGB and depth images to feed a single deep CNN augmented with auxiliary prediction connections. Accordingly, deep visuomotor policies are trained to map pixels directly into actions using behavioral cloning. Using the same model architecture and hyper-parameters across all tasks, the results showed that less than 30 minutes of demonstration data are sufficient for each task to learn a successful policy.

The previously mentioned works capture through CNN the spatial features of the considered data. In this thesis, deep learning is introduced in 3D Autocomplete to predict the user intended motions from the joystick commands. To make this attainable, a dataset of different 3D motions is collected. This allows to build a deep learning model that is able to capture the long-term dependencies alongside the short-term dependencies of the data. Moreover, I propose an algorithm capable of indicating when a prediction should be attempted. This was found to be a critical need when machine learning predictive models are to be used on real-time time-series data. Moreover, I rely on velocity vector commands instead of vision-based data.

2.3 Mixed Reality in HRI

The recent technologies in MR have allowed humans to blend virtual objects with their real surroundings. This has unveiled a new level of interaction and experience in HRI fields. MR is used in manufacturing [34], medical surgeries [35], education [36], etc. However, only a few works have considered MR in the context of assisted teleoperation. Szczurek et al. [37] introduce a framework that assists operators in teleoperating mobile robots in dangerous environments. They present a mixed reality UI that enables both individual and multiple users to interact with 3D holographic representations of the controlled robot and its surrounding environment. Experiments proved the effectiveness of the proposed system in enabling stable remote teleoperation. In similar work, Sun el al. [38] used an MR interface with an interaction proxy for teleoperating industrial robots. The proposed system utilizes a Head-Mounted Display (HMD) interface with tracking technology for both head and hand gestures in the virtual environment. While these works use MR to enable remote teleoperation within a constructed scene, my proposed system directly augments the user's desired trajectory into the real-world scene of the robot's camera.

Hedayati et al. [39] introduce an augmented reality UI to improve the teleoperation of UAVs. In their framework, they integrate information from the robot's visual stream into the user's perspective of the UAV and its surroundings. Additionally, the system provides real-time visual feedback on the UAV camera functionalities. This allows for more effective robot control and task completion. Both quantitative and qualitative results have validated the effectiveness of their system in enhancing the teleoperation of UAVs, similar work is presented in [40]. Moreover, Lee et al. [41] introduced an innovative approach to robot teleoperation by incorporating virtual fixtures into the environment scene. They propose that integrating virtual fixtures is aimed at improving user understanding. The framework is suggested to be used in risky challenges such as cleaning up nuclear waste.

The weakness of the works mentioned earlier lies in their exclusive emphasis on enhancing the user's perspective, and presuming constant visibility of the robot within the user's field of view (FOV). However, this approach may fail when the robot is situated far away from the control station. In contrast, my proposed UI adopts a first-person view approach. In other words, operators observe a live video feed from the robot's built-in camera using a mixed reality headset. This setup allows the user to maintain a view of the robot's environment even when the robot is not within the FOV.

Chapter 3

Proposed Approach

The objective of this thesis is to develop 3D Autocomplete for aiding UAV teleoperation in 3D motion scenarios. Accordingly, this chapter introduces the proposed architecture, detailing each component within the system and exploring the theoretical principles behind the proposed algorithms.

3.1 Overview of System Components

The system structure of 3D Autocomplete is illustrated in Fig. 3.1.

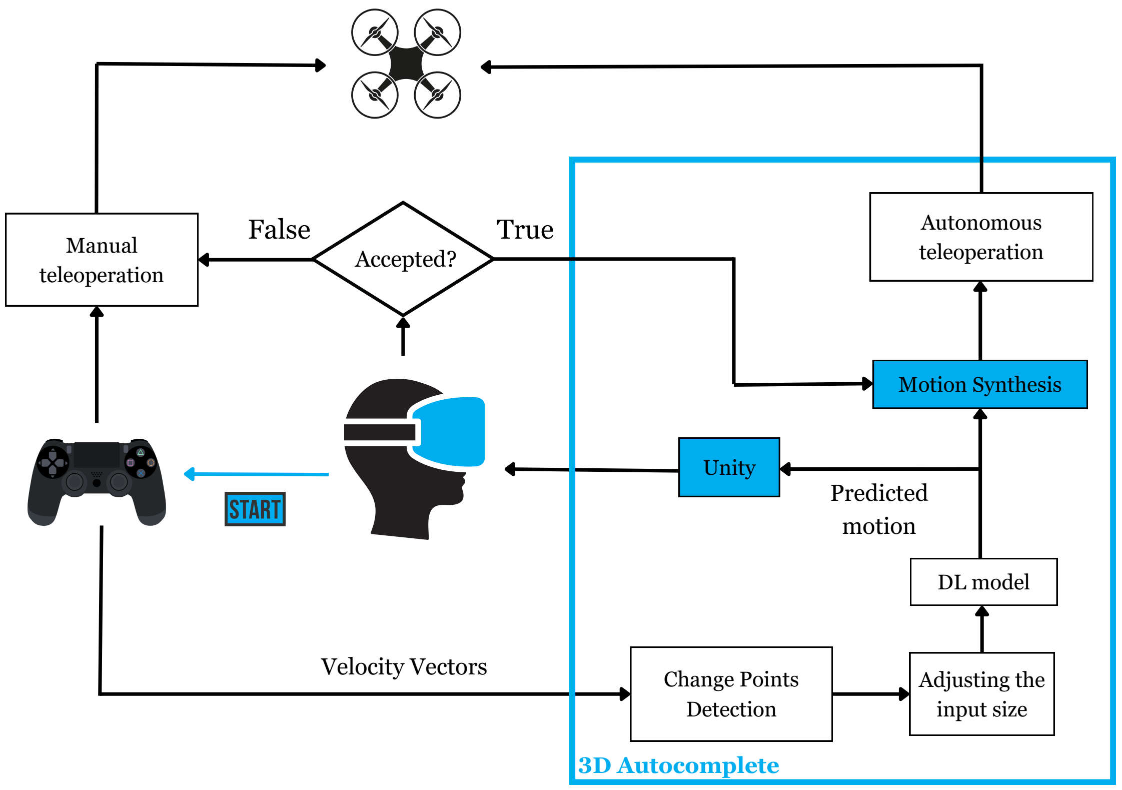


Figure 3.1: 3D Autocomplete architecture

3D Autocomplete takes as input the joystick commands conducted by the user and monitors it using a change point detection algorithm (just-in-time). When significant changes are detected, the input motion is forwarded to a deep learning model to predict its type after resizing it to match the model's input dimensions. Then, 3D Autocomplete proposes the predicted motion through a UI, giving the user the choice to either accept, decline, or ignore. If the user accepts, the system synthesizes the suggested motion and directs it to the autonomous pilot to execute. The components of 3D Autocomplete are addressed in the following subsections.

3.1.1 *Change Points Detection: Just-in-time Algorithm*

3D Autocomplete aims to predict the user's intended motion from its partial input. In other words, it tries to provide an early prediction of the user intended motion. To perform such a task, the system should decide in real-time when to predict the motion type. In previous works, Zein et al. [10] [9] considered a constant time interval and a constant number of velocity vector samples to predict the user's motion. However, this might lead to the prediction being too early, where the model might not have sufficient information to distinguish motions, or it might be too late for the benefits of 3D Autocomplete to apply. Moreover, to fly a UAV around a 3D shape, each geometric primitive size would require a different time interval to be covered, and thus each motion is presented by a different number of samples. This also leads to the model being scale-sensitive.

To address these issues, I develop a just-in-time algorithm based on spotting important 'change points' in the operator's 3D motion while flying the UAV. This allows the system to predict the intended motion type after identifying these important changes in the motion. One of the important changes in the considered 3D motions is the change of the motion direction, velocity vector direction, while the UAV is moving around the shape, as shown in Fig. 3.2. These change points indicate that the user has already covered a sufficient part of the 3D geometrical shape by flying around it despite the shape size or type, and the needed time (number of samples).

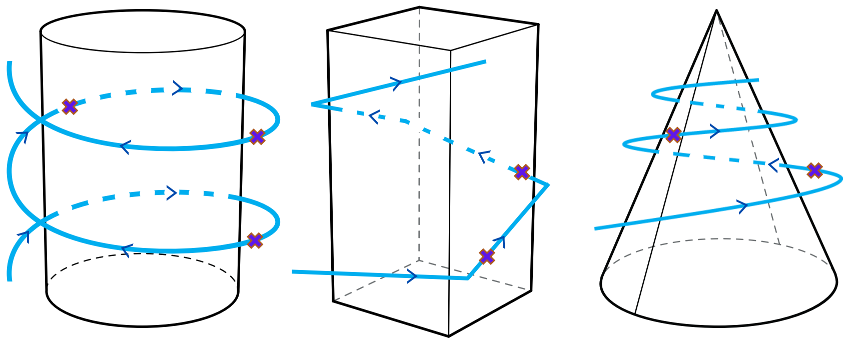


Figure 3.2: Spotting important changes in the user 3D motion: a change point, presented by a cross, is detected when a change in the drone velocity vectors direction take place.

Our approach to detect these changes is summarized in Algorithm 1. While collecting the velocity vectors that correspond to the UAV commands, the dot product between the currently collected velocity vector (V_i) and a reference vector (V_R) is calculated. The reference vector presents the direction of the previously collected vectors. Accordingly, based on the cosine of the angle between them, the dot product interprets the direction of the new vector relative to the direction of the previous ones as shown below:

$$dot(V_i, V_R) = \|V_i\| \cdot \|V_R\| \cdot \cos(V_i, V_R) \tag{3.1}$$

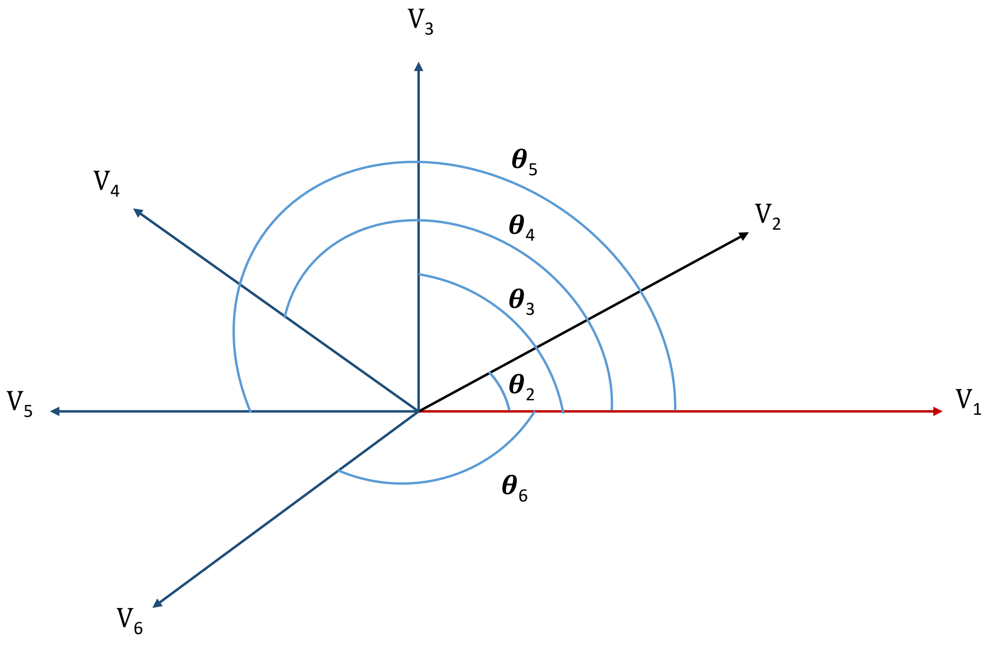


Figure 3.3: Geometrical interpretation of dot product [42]: V_1 and V_2 are in the same direction, where θ_2 is less than 90 degrees (same half-plane). V_1 and V_4 are in opposite direction, where θ_4 is greater than 90 degrees.

Based on the sign of the obtained product, the algorithm can deduce the alignment of the new vector with respect to the previous ones. For example, if the new vector and the previous vectors are in the same direction, the dot product will be positive; if they are in opposite directions, it will be negative. This is because, geometrically, the dot product is related to the angle between two vectors. If the angle is acute (less than 90 degrees), the dot product is positive and the two vectors are in the same half-space. If the angle is obtuse (greater than 90 degrees), the dot product is negative and the two vectors are in opposite half-space (see fig 3.3). This concept applies to 3D space, as the definition of a vector is independent of any specific position. In other words, a vector can have its tail located at any position. Consequently, even though two vectors in 3D space can be oriented in any direction, they can be represented together on a 2D plane [42]. The reason 90 degrees is selected as the change threshold is to allow the system to identify when the drone was commanded ”around a corner”. Each subsequent detection represents turning a corner and this way we are able to identify when the drone has covered a sufficient path to allow for the algorithm to detect the shape.

After detecting a change point, the reference vector is updated to represent the new direction of the motion. Subsequently, each new change point is detected with respect to the updated reference vector as illustrated in Algorithm 1. This allows 3D Autocomplete to identify the changes in the UAV motion direction upon each occurrence. Note that the commands generated by the user can be noisy, especially for an inexperienced user. Therefore, to mitigate the effect of noise, a change point is only detected when the change of the motion direction takes place for multiple consecutive vectors.

Algorithm 1 Change Point Detection in Autocomplete

```
 1: Initialization of Autocomplete components
 2: $k \leftarrow 0$
 3: Collect a velocity vector $V_R$
 4: while Repeat do
 5:     Collect a velocity vector $V_i$
 6:     Find the dot product between $V_R$ and $V_i$ : $d_i \leftarrow V_R.V_i$
 7:     if d$_i < 0$ then
 8:         $k \leftarrow k + 1$
 9:         if $k > 5$ then
10:             Change point is detected at $V_i$
11:             $V_R \leftarrow V_i$
12:             $k \leftarrow 0$
```

3.1.2 *Motion Classifier: Deep Learning Model*

After detecting important changes in the user’s motion, DL is employed to predict the user’s desired motion type. To accomplish this, I collected a dataset of 3D motion primitives and constructed a suitable model that matches the data type.

3.1.2.1 Data

Three 3D motion primitives are considered in my work: motion around a cylinder, a cone, or a box. The Gazebo virtual environment [43] is used to collect the data (see Fig. 3.4), and the “Tum simulator” package is used to simulate an AR.Drone 2.0. quadrotor (see Fig. 3.5). Additionally, both “Ardrone Joystick” and “Joy Node” are used to control the quadrotor using a PS4 joystick connected to the computer.

The first step in the data acquisition process is creating the 3D shape that the motion primitive will cover. For each 3D motion primitive, a corresponding 3D geometric primitive is created in the Gazebo virtual environment. Different shape sizes (heights, radii, widths, etc.) are considered, in addition to different orientations. Examples of the created shapes are shown in Fig. 3.6.

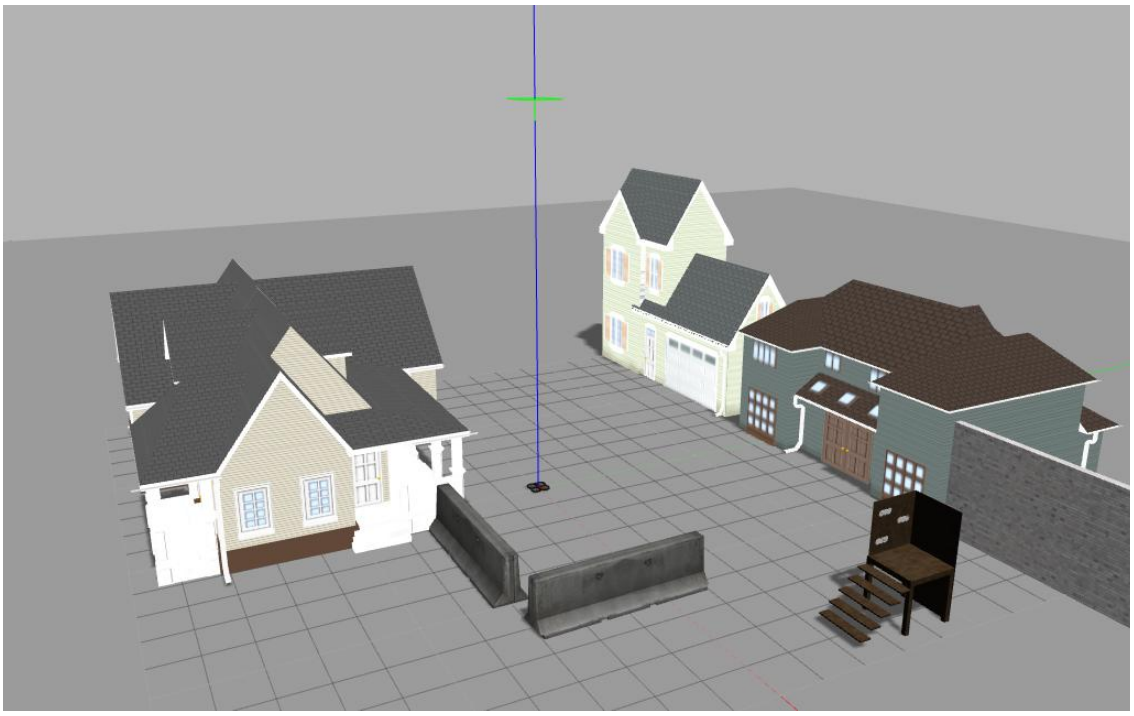


Figure 3.4: Gazebo virtual environment: this environment is used to collect the 3D motions dataset.

After creating the geometrical shape, the quadrotor is moved around it using manual joystick control, thereby forming the 3D motion to be collected. Moreover, in order to ensure the generalizability and diversity of the dataset, several motions with varied directions are conducted for the same shape, i.e., upwards, downwards, forwards, backward, clockwise, anticlockwise, etc. To begin collecting the data, a start/stop button is used to indicate the beginning/end of the 3D motion. The collected data includes the joystick's velocity commands determined by its analog sticks movements (input from the operator), in addition to their corresponding timestamps. This data is gathered by observing the echoed messages being published on the"Joy" topic. Such information presents the linear velocities in the x, y, and z directions. Each motion primitive is represented as a three-channel sequence of velocity commands in the form of $[[v_{x1}, v_{y1}, v_{z1}], [v_{x2}, v_{y2}, v_{z2}], ..., [v_{xN}, v_{yN}, v_{zN}]]$, where N represents the sequence size. I collected 295 motions around cylinders, 279 around boxes, and 217 around cones. Therefore, the available data set consists of 791 motions. The final data was split into 80% for training and 20% for testing and validation.

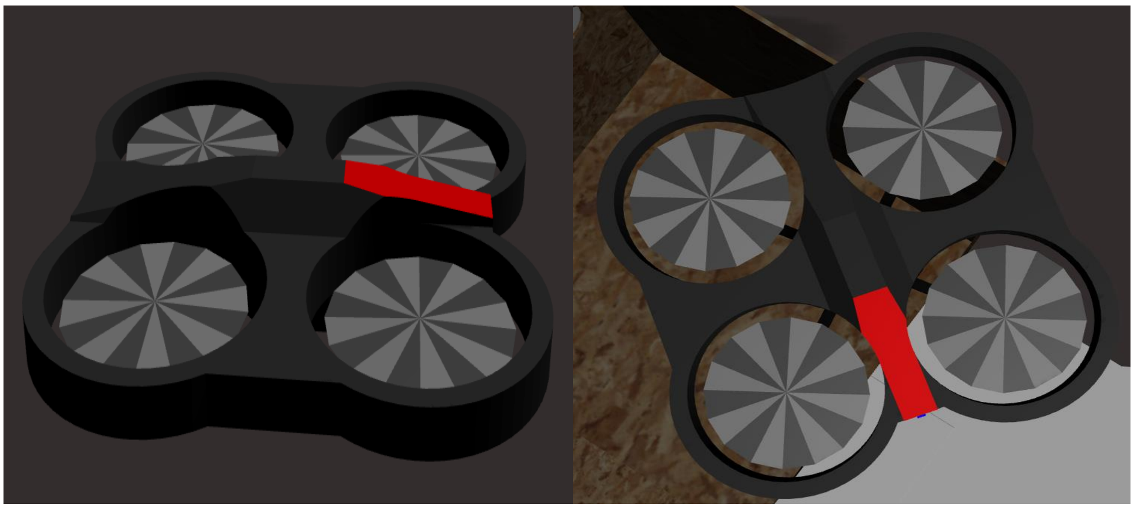


Figure 3.5: Simulated AR.Drone 2.0. quadrotor: the “Tum simulator” simulated drone used to collect the 3D motion dataset.

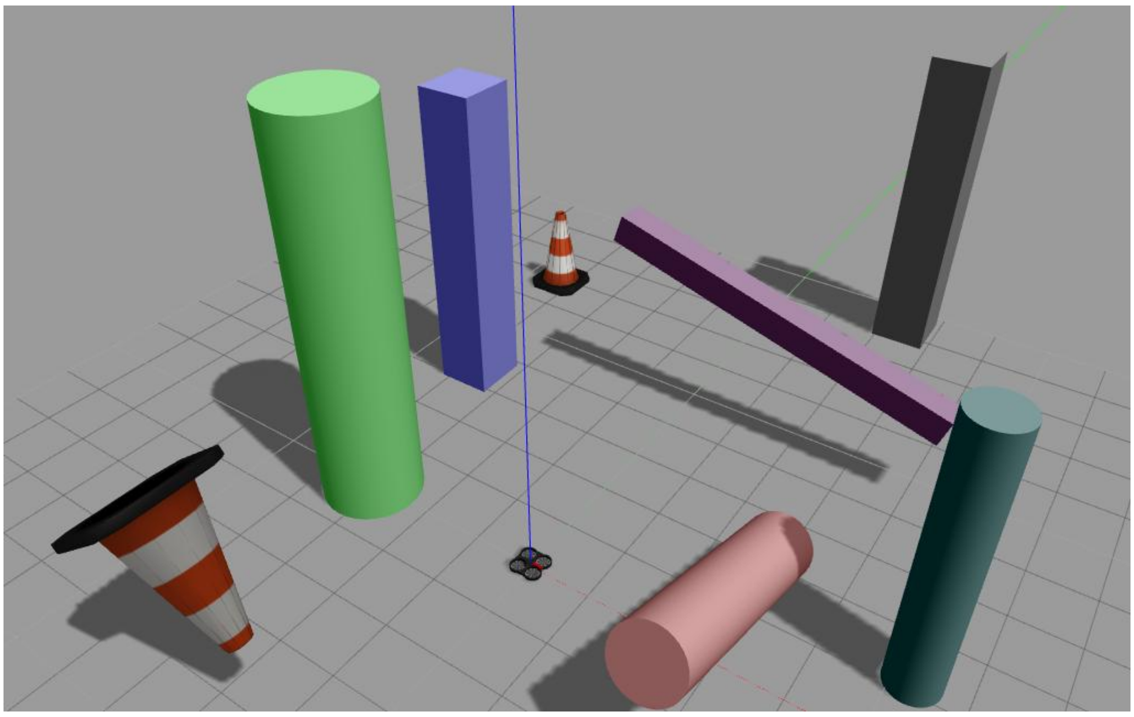


Figure 3.6: Some of the considered shapes while collecting the dataset

3.1.2.2 Model

The collected data is made up of time series sequences that represent the joystick velocity commands. To deal with such type of data, two very efficient state of the art techniques are presented: 1-Dimensional Convolutional Neural Networks (1D-CNNs) and Gated Recurrent Units (GRUs) [44]. 1D-CNN layers are capable of capturing the local/short-term dependencies and positional relationships between adjacent samples in the sequence. However, GRU layers, a type of recurrent neural network (RNN), capture global/long-term dependencies and retain information from past time steps while being computationally and memory efficient. Accordingly, my deep learning model is made up of a combination between these layers in addition to the Adam optimizer, categorical crossentropy loss function, and LeakyReLU activation functions to avoid exploding/vanishing gradients (see table 3.1).

To construct the proposed model, different combinations were tested starting from one convolutional layer combined with one GRU layer. Based on the results, more layers were added. However, the number of GRU layers was limited to a maximum of two layers since two layers have been shown to be sufficient to detect complex features. As a result, the model converged to the deep network shown in Fig. 3.7 consisting of four 1D-CNNs that return low-level feature outputs which are introduced to the two GRU layers that output the high-level features used for classification by Softmax output (SM). Moreover, two Dropout (DO) layers were introduced to the model to minimize overfitting, in addition to Batch Normalization (BN) to provide some regularization (see table 3.1).

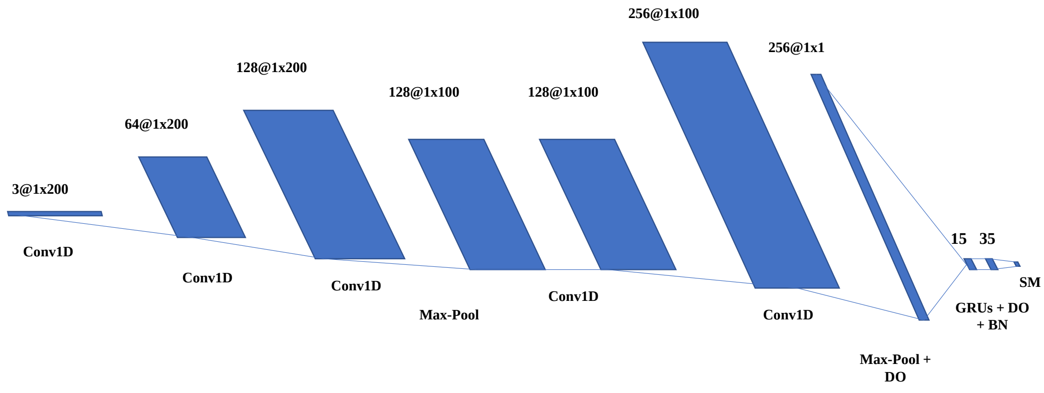


Figure 3.7: Deep learning model architecture

Table 3.1: Overview of Neural Network Components

Component	**Symbol/Function**	**Description**
Adam Optimizer	$\text{Adam}(\eta, \beta_1, \beta_2)$	Adam optimizer is an adaptive optimization algorithm that adjusts the weights and biases of the neural network during training. It adjusts learning rates for each parameter individually based on the historical gradients. η is the learning rate, β_1 and β_2 are exponential decay rates for the moment estimates.
Categorical Crossentropy Loss	$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$	A loss function commonly used for multi-class classification problems. It measures the dissimilarity between the true distribution y and the predicted distribution $\hat{y}$.
LeakyReLU Activation	$\text{LeakyReLU}(x, \alpha)$	An activation function that allows a small negative slope (α) for the negative input region, preventing inactive neurons during training. Here, x represents the input to the activation function, and α is the slope for negative values.
Dropout (DO)	$\text{DO}(x, p)$	A regularization technique where randomly selected neurons are ignored during training with a probability p. It helps prevent over-fitting. Here, x is the input to the dropout layer, and p is the probability of dropping out a neuron.
Batch Normalization	$\text{BatchNorm}(x)$	Normalizes the input of a layer to have zero mean and unit variance across the mini-batch. It helps stabilize and speed up the training of neural networks. Here, x is the input to the batch normalization layer.

3.1.3 *Variable Input Size*

Generally, each motion primitive will have a different sequence length depending on the type of motion, the geometric primitive dimensions, the velocity of the UAV, etc. Moreover, in my proposed framework, the motion prediction takes place after the change point detection, so the number of sample points in each partial motion introduced to the deep learning model varies. However, a deep learning model has a fixed size input layer, so the input motion size could be smaller or larger than the expected input size of the model. To resolve this issue, 3D Autocomplete performs up-sampling or down-sampling allowing the input motion to fit into the input layer of the deep learning model.

For motions with a length size greater than the input size, I propose an approach inspired by piece-wise aggregate approximation [45]. A window size is defined based on the size of the collected samples and the targeted size. Then, depending on how many windows are needed to down sample the motion, windows are distributed among the samples and the velocity vector samples falling in the defined windows are replaced by their average (see fig. 3.8. For motions with a length size smaller than the input size, zero padding is performed.

Algorithm 2 Down and Up sampling in Autocomplete

```
    Input: Input Layer Sequence Length n, Motion of size m $x = [x_1, x_2, ....., x_m]$
    Output: Motion of size n $X = [X_1, X_2, ....., X_n]$
 1: Initialization of Autocomplete components
 2: $n \leftarrow$ Input Layer Sequence Length
 3: Collect a partial motion $x = [x_1, x_2, ....., x_m]$
 4: if $m > n$ then
 5:     $w \leftarrow ceil(m/n)$
 6:     $j \leftarrow 0$
 7:     while $i < m$ do
 8:         if $i\%(m * \frac{w-1}{m-n}) = 0$ then
 9:             $X_j = \frac{1}{w} * \sum_{n=i}^{i+w-1} x_n$
10:             $i \leftarrow i + w$
11:         else
12:             $X_j = x_i$
13:             $i \leftarrow i + 1$
14:         $j \leftarrow j + 1$
15: else
16:     $j \leftarrow m$
17:     while $l < n$ do
18:         $X_j \leftarrow [0, 0, 0]$
19:         $l \leftarrow X$ length
20: return $X$
```

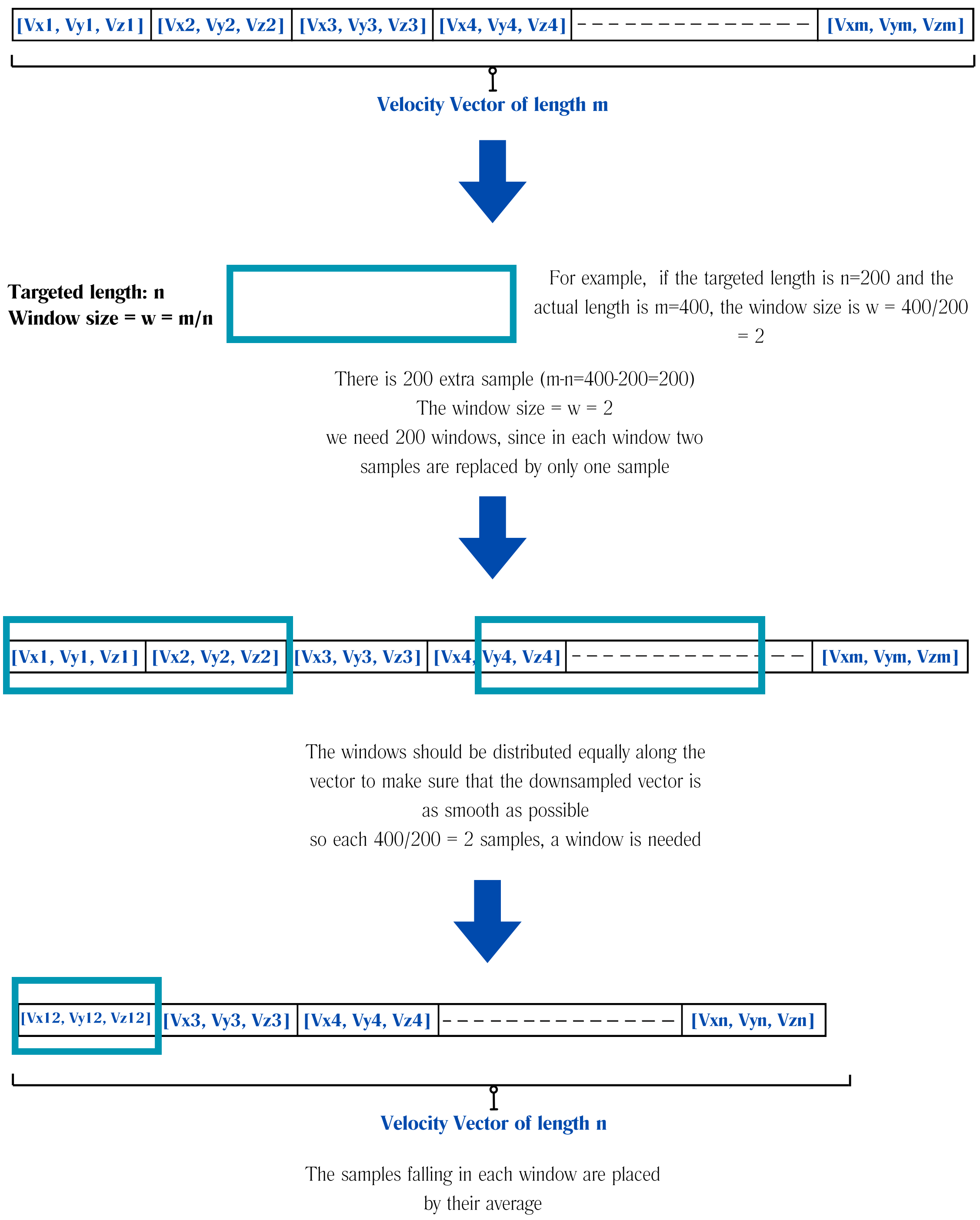

[Vx1, Vy1, Vz1]
[Vx2, Vy2, Vz2]
[Vx3, Vy3, Vz3]
[Vx4, Vy4, Vz4]
[Vxm, Vym, Vzm]
Velocity Vector of length m
Targeted length: n
Window size = w = m/n
For example, if the targeted length is n=200 and the actual length is m=400, the window size is w = 400/200 = 2
There is 200 extra sample (m-n=400-200=200)
The window size = w = 2
we need 200 windows, since in each window two samples are replaced by only one sample
The windows should be distributed equally along the vector to make sure that the downsampled vector is as smooth as possible
so each 400/200 = 2 samples, a window is needed
[Vx12, Vy12, Vz12]
[Vxn, Vyn, Vzn]
Velocity Vector of length n
The samples falling in each window are placed by their average

Figure 3.8: Downsampling Algorithm concept

3.1.4 *Mixed Reality User Interface*

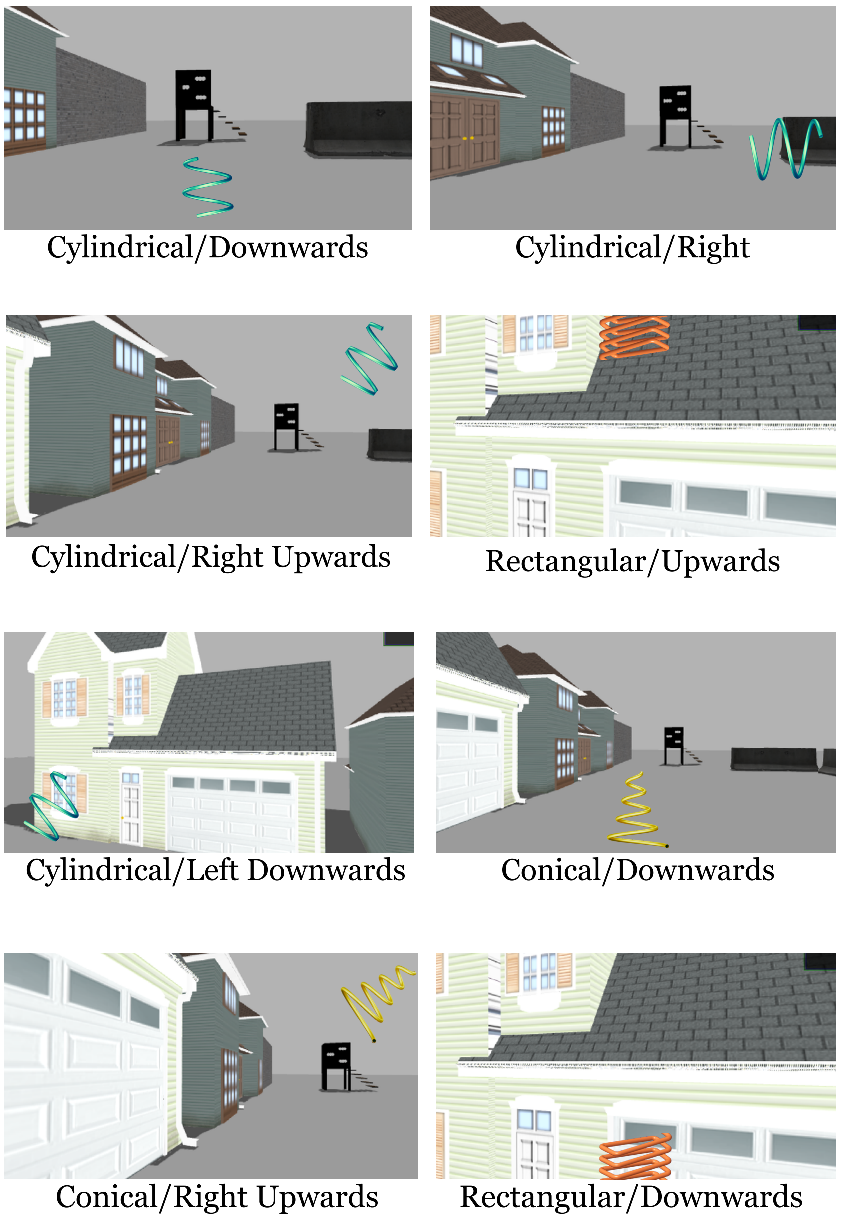


Figure 3.9: Proposed user interface for 3D Autocomplete

After predicting the user intended motion, 3D Autocomplete suggests it through a UI. Initially [9], the idea was to augment predicted trajectories, which were generated after synthesizing the motion, and overlaying them onto the live video feed captured by the drone's onboard camera. However, in 3D Autocomplete, a more advanced approach is presented: mixed reality UI. Integrating mixed reality into teleoperation enhances the user's understanding of the robot's actions and its surrounding environment. This strengthens the interaction between humans and robots and thus improves the performance of the system [37].

First, the real-time video stream from the UAV's onboard camera is presented to the user through a VR headset rather than the traditional screen. This experience eliminates distractions from the physical surroundings and enhances the user's focus during teleoperation. Moreover, it enhances the connection and interaction between the operators and the remote environment.

While 3D Autocomplete is active, the predicted 3D motion is augmented directly into the user's field of view. For example, if the predicted motion is a helix, 3D Autocomplete will display a helical motion within the augmented environment (see Fig. 3.9). Moreover, to take into account the motion direction while displaying the predicted class (a helix to the left, a helix to the right, etc.), I utilize Principle Component Analysis (PCA) to identify the major axis of the motion. Since the primary aim is to convey to the operator the "rough" estimate and no need to show the exact predicted motion, I consider eight orientations for each motion: $0°, 45°, 90°, 135°, 180°, 225°, 270°$, and $315°$. The augmented motion is then selected based on the nearest angle.

3.1.5 *Motion Synthesis*

After predicting the motion type by the deep learning model and accepting it by the user through the MR UI, Autocomplete synthesizes the user-intended motion. Synthesizing the motion generates an estimated 3D trajectory that fits well the user's partial motion and the user's intentions. This allows the UAV to complete the desired motion autonomously by following this generated trajectory.

The 3D motions are synthesized by fitting the partial motion performed by the user into the predicted motion type: motion around a cylinder, motion around a cone, or motion around a box. These motions are presented as a cylindrical helix, a conical helix, and a rectangular helix, respectively. For example, if the predicted motion is a motion around a cylinder, Autocomplete fits the user's partial motion into a cylindrical helix. This will estimate the parameters that define this motion, such as the radius of the helix, the pitch, etc. Accordingly, the rest of the trajectory can be estimated based on these parameters (see Fig. 3.10).

3.1.5.1 Motion Around a Cylinder: Cylindrical Helix

A cylindrical helix has a constant radius and pitch. It is defined by the following parametric equation:

$$\begin{cases} x = r \cdot \cos(t) \\ y = r \cdot \sin(t) \\ z = p \cdot t \end{cases} \tag{3.2}$$

where p is the pitch of the helix, r is the radius, and $t \in [0, 2\pi)$. Fitting the data into a cylindrical helix involves estimating the parameters r and p. Accordingly, Least Square (LS) optimization is used to find their optimal values. LS fits the input

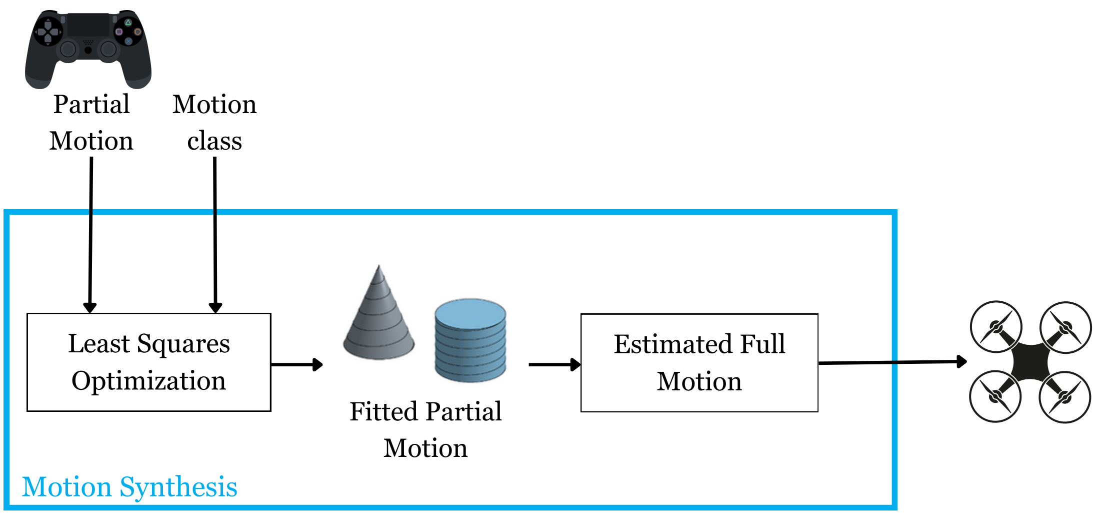


Figure 3.10: Motion synthesis for 3D motions: after predicting the class of the motion, the input partial motion will be fitted into the predicted motion type using Least Squares. Then, the full motion will be synthesized by completing the fitted motion.

data into the predicted motion by minimizing a sum of the squared errors objective function. The objective function is presented below:

$$f(r,p) = \sum_{i=1}^{n} \left[(x_i - x'_i)^2 + (y_i - y'_i)^2 + (z_i - z'_i)^2\right] \tag{3.3}$$

where n is the number of data points. Levenberg-Marquardt (LM) is used to minimize this function. It is one of the most effective algorithms for non-linear LS problems.

3.1.5.2 Motion Around a Cone: Conical Helix

A conical helix has a variable radius that increases/decreases exponentially over time. It is defined by the following parametric equation:

$$\begin{cases} x = r \cdot e^{-k \cdot t} \cdot \cos(t) \\ y = r \cdot e^{-k \cdot t} \cdot \sin(t) \\ z = p \cdot t \end{cases} \tag{3.4}$$

where p is the pitch of the conical helix, r is the radius, k controls the tightness or wideness of the helix, and $t \in [0, 2\pi)$. Fitting the data into a conical helix involves estimating the parameters r, p, and k. To do so, the same approach is used as that of the cylindrical helix.

3.1.5.3 Motion Around a Box: Rectangular Helix

A rectangular helix has a rectangular cross-section. It is defined by the following parametric equation [46]:

$$\begin{cases} x = a \cdot \max\left(-1, \min\left(\frac{4}{\pi}\arcsin\left(\sin\left(\frac{\pi t}{2} + \frac{\pi}{4}\right)\right), 1\right)\right) \\ y = b \cdot \max\left(-1, \min\left(-\frac{4}{\pi}\arcsin\left(\cos\left(\frac{\pi t}{2} + \frac{\pi}{4}\right)\right), 1\right)\right) \\ z = p \cdot t \end{cases} \tag{3.5}$$

where p is the pitch of the rectangular helix, a/b are the dimensions of a $2a{\times}2b$ rectangle that represents the helix cross-section, and $t \in [0, 2\pi)$. LS and LM are also used to fit the user's partial motion into a rectangular helix by estimating the parameters a, b, and p.

Additionally, to account for the orientation and the position of the considered motion, I introduce a rotation matrix and a translation vector into all the parametric equations. This involves incorporating the Euler angles within a rotation matrix and the components of a 3D translation vector into the optimization problem. As a result, the system will have the estimated parameters of the user's motion in addition to its orientation and position. Accordingly, to estimate the future desired trajectory, the system sets the variable 't' to an 'upcoming' value, typically within the range of $[2\pi, 4\pi)$, and calculates the corresponding values for x, y, and z. This process generates target points relative to the UAV's current position, allowing autonomous completion of the user's desired motion.

3.1.6 *Autonomous Navigation*

After accepting the Autocomplete motion that aligns with the user's intentions, the estimated trajectory is fed into the UAV controller for trajectory following. A lot of works in the literature have tackled the drones navigation of 3D helical trajectories [47] [48] [49]. In this thesis, the UAVs used were already controlled using a PID controller. The simulated model of AR.Drone 2.0 includes the needed sensors for implementing a robust autonomous navigation system.

Chapter 4
Results and Experiments

To validate my proposed system, I conducted human-subject experiments that involved multiple stages. First, I trained and tested the proposed deep learning model, and prepared the necessary implementation environment to assess the system. This chapter offers implementation details, discusses the experiments conducted to support the advantages of 3D Autocomplete, and presents the achieved subjective and objective results.

4.1 Preparing the Data

3D Autocomplete predicts the input motion type after detecting important change points in the motion and adjusting its size to fit the deep learning model (up-sampling or down-sampling). Therefore, the collected data used for training and testing the deep learning model should be pre-processed through the just-in-time and the up/down-sampling algorithms to make the model compatible with the overall architecture.

Accordingly, to prepare the data, I detected important change points in the training and testing data. Then, after choosing to predict the motions after the second detected point, Algorithm 2 is introduced for motions with sizes smaller than or greater than the deep learning model input size, which is 200 (the average length of the 3D motions). As a result, I attained a dataset that presents 3D motion primitives after two detected change points and of size 200 for each motion. Finally, this dataset is introduced for the deep learning model. Some samples of change point detection are shown in Fig. 4.1.

4.2 Deep Learning Model

This section details the training and testing phases of the deep learning model.

4.2.1 *Training*

The model presented in Section 3.1.2 is trained over 80% of the collected dataset. The validation data was used in order to tune some parameters presented in the

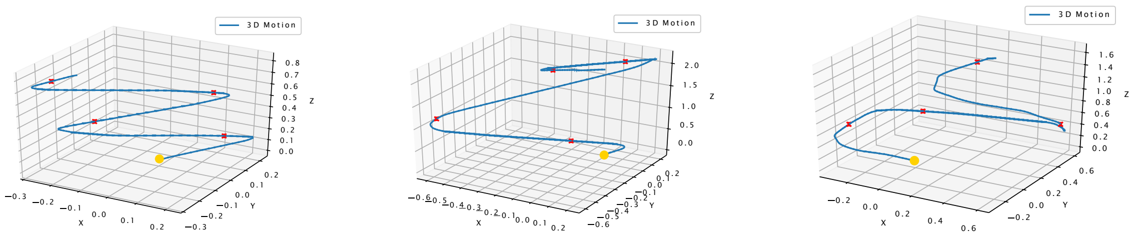


Figure 4.1: Just-in-time algorithm: each plot presents a sample 3D motion primitive, Algorithm 1 is applied on these motions. The yellow dot presents the starting point of the motion and the red cross presents the detection of a change point.

model such as the dropout, the number of filters, and the filter sizes. During training, early stopping was introduced to the process to avoid overfitting by monitoring the validation loss.

The learning curves, learning accuracy and loss, are shown in Fig. 4.2, where the model does not overfit and converges to a training/validation accuracy of 80/70% .

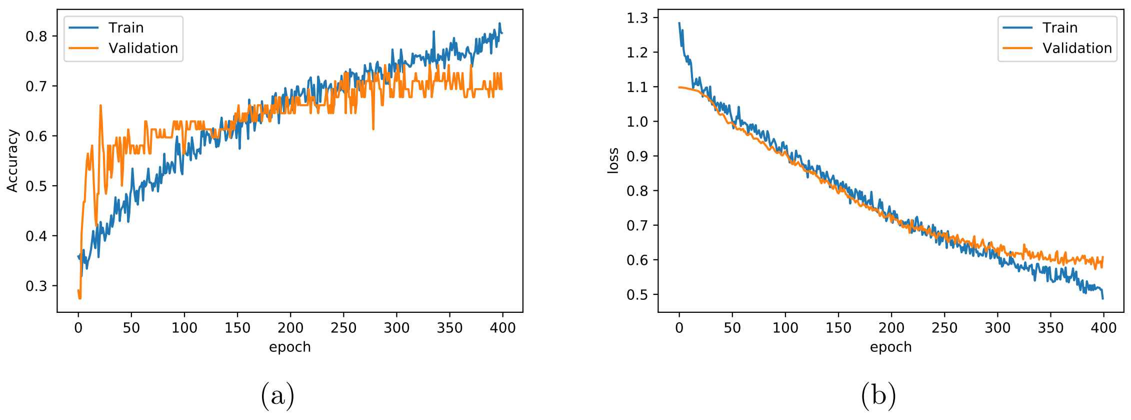


Figure 4.2: Learning curves: training/validation (a) accuracy and (b) loss during training phase.

4.2.2 *Offline Evaluation*

Before testing the trained model in real-time on ROS/Gazebo, I evaluated it using the collected testing data.

The model reached 89% f1-score. The confusion matrix is shown in Fig. 4.3 where the misclassifications between cylindrical and conical motions are the main source of error. This is because the two motion primitives are geometrically similar in form.

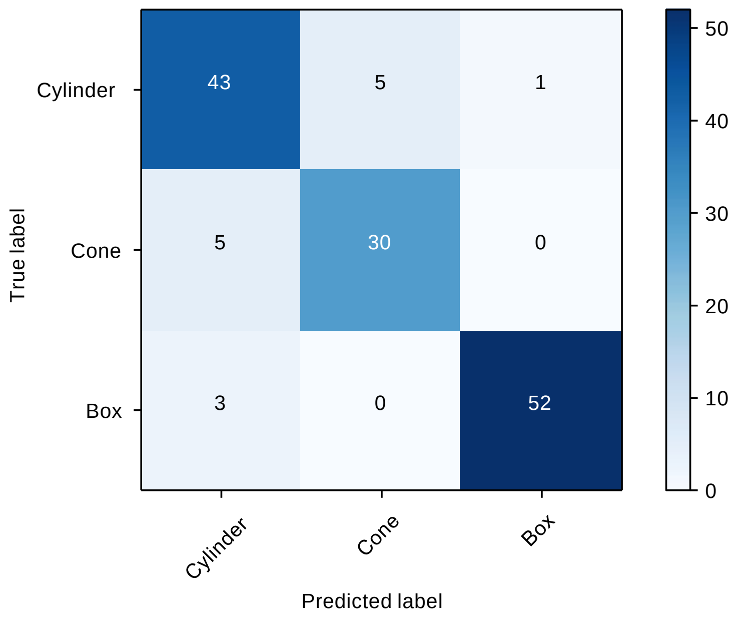


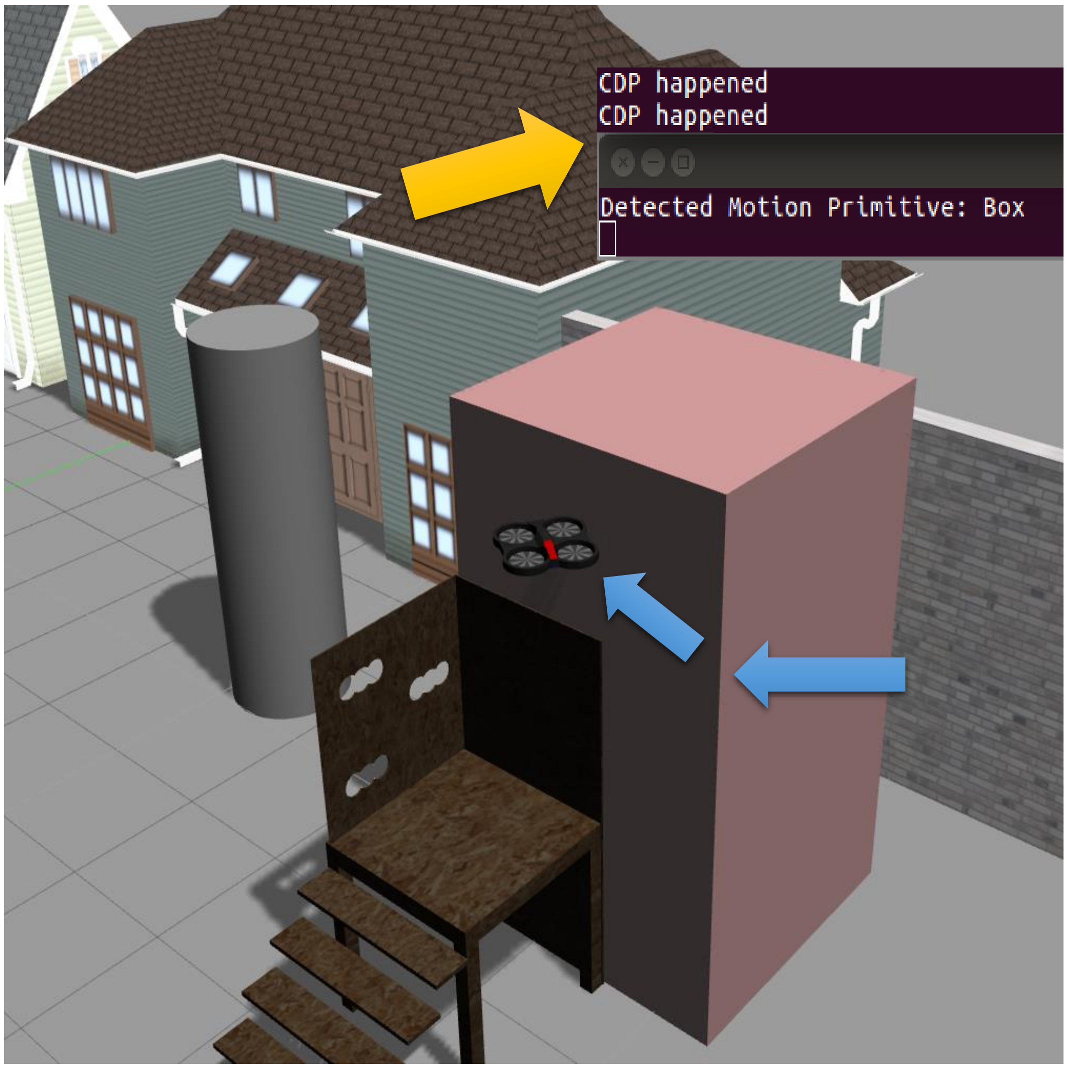
Figure 4.3: Confusion matrix

Table 4.1: Real time simulation results of the DL Model.

Motion Primitive	Accuracy	f1-score	Total number
Cylinder	0.9	0.8	25
Cone	0.8	0.83	23
Box	0.79	0.84	21

4.2.3 ***Real Time Simulations***

To test my system in real time, a Gazebo virtual environment full of different geometrical shapes is prepared as shown in Fig. 4.4. I used an AR.Drone 2.0 quadrotor from "Tum simulator" package and controlled it using a PS4 joystick connected to the computer. We started by moving the UAV around the targeted shape and accordingly we were notified on the terminal window when a change point detection took place (CDP). After two CDPs, a prediction of the intended motion is observed as shown in Fig. 4.4 (upper right corner). The system was tested by different users and the results of the real time simulations are shown in table 4.1. Similar to the offline tests, the real-time experiments show that the proposed 3D Autocomplete is capable of detecting early, with high accuracy, the user's intentions based on the joystick commands only.



```
CDP happened
CDP happened
```

```
Detected Motion Primitive: Box
```

Figure 4.4: Gazebo Testing Environment for the real time simulations: the environment is full of different geometrical shapes to fly the UAV around while testing.

4.3 Implementation Details

4.3.1 *AR.Drone 2.0*

As mentioned in section 3.1.2, an AR.Drone 2.0. quadrotor is used in a Gazebo virtual environment in my simulation study (see Fig. 3.5). The UAV measures 53cm x 52cm and weighs 420g; it is equipped with an HD camera (720p 30fps) for video recording with a field of view of 73.5°x 58.5°, and a vertical QVGA camera (60 fps) to estimate the ground speed. Moreover, it is enhanced with a 3-axis gyroscope, magnetometer, accelerometer, and an ultrasound altimeter which functions at 25 Hz.

4.3.2 *Mixed Reality*

Unity with Vuforia Engine SDK was used to augment the 3D motions into the user's scene. To do so, I emulate the existence of a physical camera using ZeroMQ (ZMQ) network protocol and Magic Camera software. This is because augmenting an object within the physical world through Unity, necessitates the presence of an AR Camera Object, which is typically done when a physical camera is linked to the host PC. Accordingly, the ZMQ publisher on the ROS PC, where the UAV is controlled, transmits the frames captured by the simulated camera on Gazebo to the ZMQ subscriber on the Unity host PC. This will open a real-time video stream window utilizing openCV and captured by a Magic Camera screen recording tool. Then, the recorded video feed is transformed into a Virtual Webcam identified and utilized by Unity. Finally, to transmit the predicted motion class from ROS to Unity, it is transmitted from the DL node to Unity via ROS bridge. This architecture is illustrated in Fig. 4.5.

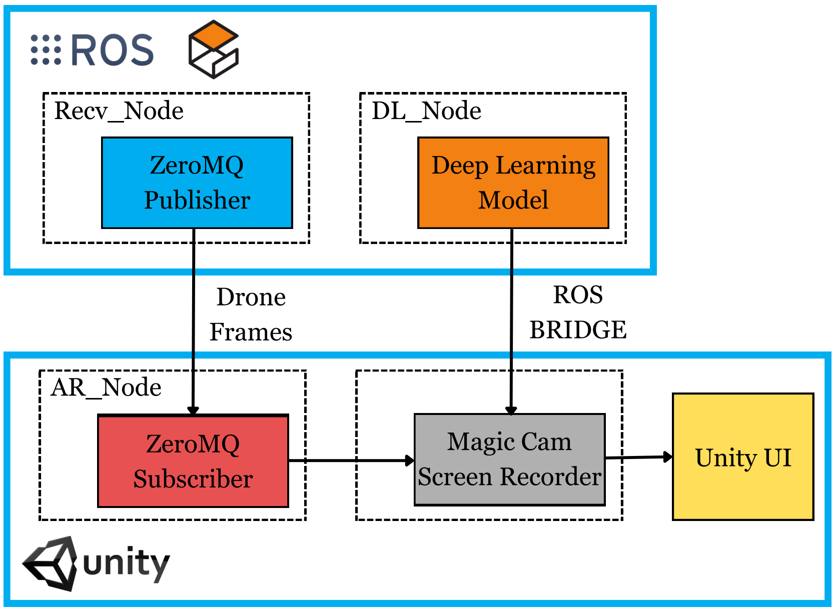


Figure 4.5: Components of the proposed MR user interface

4.3.3 ***VR-headset***

As mentioned before, the user is presented with a real-time video stream from the UAV's onboard camera through a VR headset. In my tests, I used an Oculus Quest 2 from Meta (see Fig. 4.6) which is enhanced with its own operating system. The Oculus device can be connected through USB or Bluetooth.

Figure 4.6: Oculus Quest 2

4.4 Experiments

To test the 3D Autocomplete framework in real-time, I conducted human-subject testing within Gazebo virtual environment integrated with ROS. The primary objective of these experiments was to assess the effectiveness and efficiency of 3D Autocomplete compared to the traditional teleoperation without autocomplete. All human-subject testing was approved by the university's Institutional Review Board (IRB); approval ID SBS-2023-0163. All invited participants were above 18 years old and familiar with driving flying robots, but with limited levels of experience.

Each experiment (per each operator) is divided into five trails or iterations. In each trail the operator is asked to fly a simulated AR.Drone 2.0. quadrotor around a 3D shape (a box, a cone, or a cylinder) twice (once through manual teleoperation and once using 3D Autocomplete). The shapes and their dimensions were chosen randomly, but in each new trail, a new shape with new dimensions is considered. The experimental procedures are as follows:

- The participants are introduced to the experimental tasks and are given some time before the experiment to adapt to the system and its components (joystick, VR headset).
- A 3D shape is chosen randomly, in addition to randomly deciding which case to start with (3D Autocomplete or traditional teleoperation).

- After each trial, the operator completes a survey to assess their experience subjectively. Note that in the both cases the motion is considered complete when it finishes covering the entire shape.

The same procedure is applied in all the experiments. The number of participants was 14, reaching a total of 140 trials divided equally between Autocomplete and manual teleoperation. The testing environment is shown in Fig. 4.7.

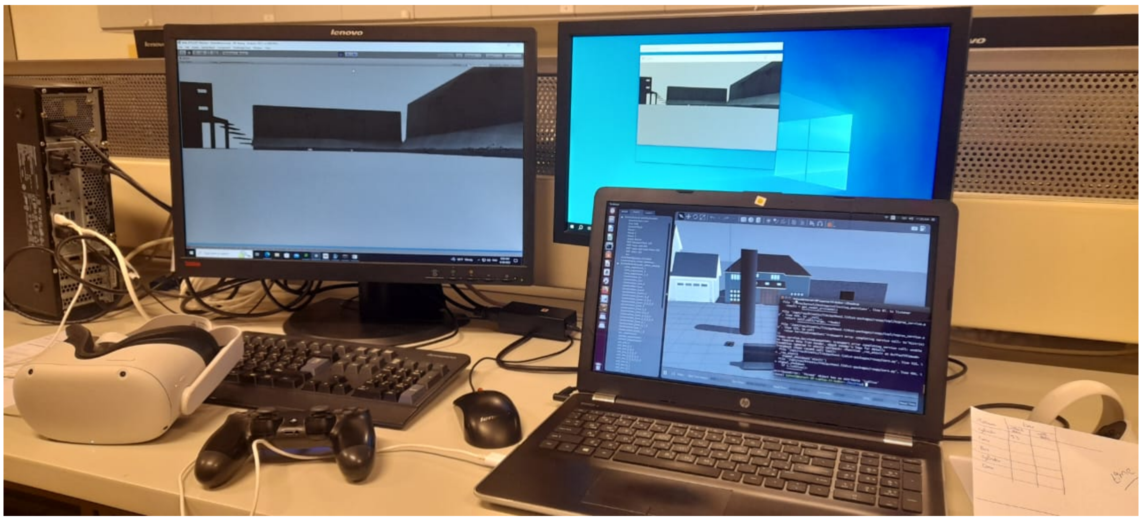


Figure 4.7: Testing environment

4.5 Results

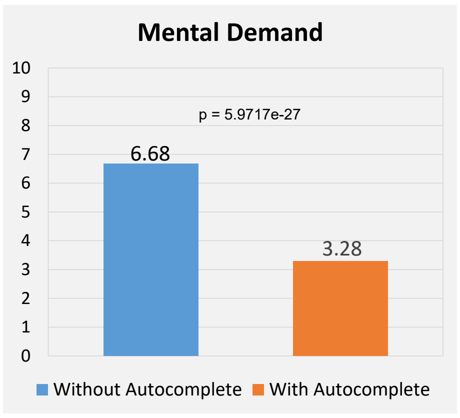
This section presents and analyzes the experimental results, encompassing both subjective and objective assessments.

4.5.1 *Subjective Evaluation*

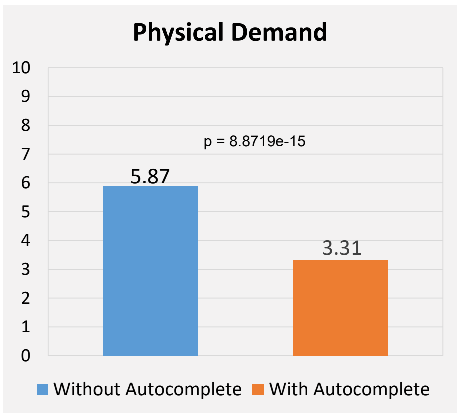
I subjectively validated my system using the NASA Task Load Index (NASA-TLX) which assesses a task by evaluating the perceived workload rates [50]. It considers six sub-scales: the mental demand, physical demand, temporal demand, performance, effort, and frustration level during task execution. This assessment is essential in my study, given that the primary objective of 3D Autocomplete is to reduce user workload during teleoperation. Thus, after each trial, users were asked to complete the NASA-TLX questionnaire (see Fig. 4.8).

The results indicate that 3D Autocomplete outperforms traditional teleoperation by effectively reducing the demands across all the subscales, while sustaining a high level of performance. This is because in 3D Autocomplete the users only have to perform a partial motion, instead of the entire motion, and the autopilot completes the remaining motion, which reduces the workload and effort. Also, the MR UI allows the users to be more focused on the task and isolated from the surrounding noises, which is reflected in the system performance.

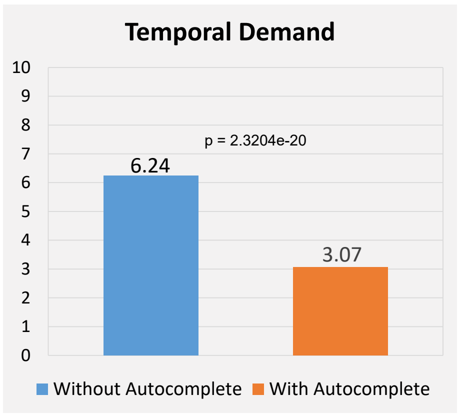
Each user experienced at least one collision during manual control (the trial was restarted in such cases), highlighting the challenges of UAV teleoperation. On the other hand, collisions rarely occurred while using 3D Autocomplete and resulted from the user deviation from the targeted shape in their conducted partial motion. This occurs because the trajectory generated for the autopilot is based on the fitted partial motion. If the partial motion deviates significantly from the intended motion to cover the shape, it can adversely affect the generated trajectory and potentially lead to collisions with the shapes.



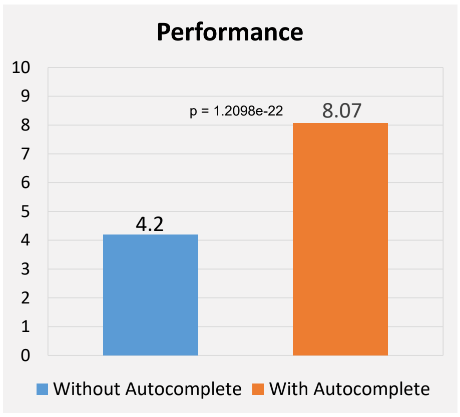
(a)



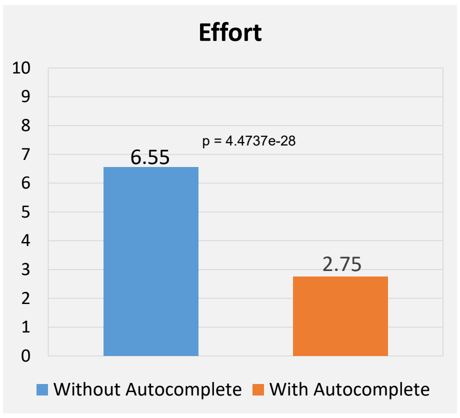
(b)



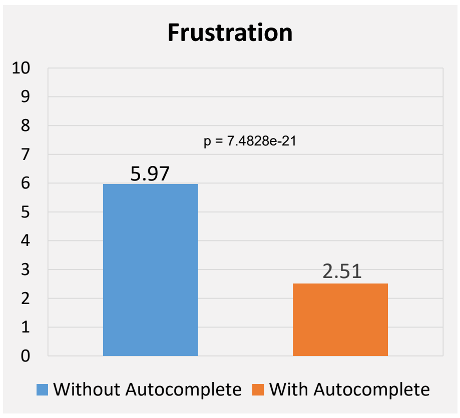
(c)



(d)



(e)



(f)

Figure 4.8: NASA-TLX Survey Scores

4.5.2 *Objective Evaluation*

To evaluate my proposed system in an objective manner, I consider three metrics:

- The duration taken to complete the assigned task: Time is a significant parameter in optimizing the effectiveness of teleoperation missions. The measuring of time begins when the user initiates UAV teleoperation and ends when the user completes the assigned task by fully covering (flying the UAV around) the specified 3D shape.
- The distance traveled by the UAV: This measurement also begins at the start of the teleoperation task and extends to the full coverage of the 3D shape. The distance reflects the user's convergence or divergence from the intended trajectory.
- The smoothness of the conducted path: This parameter is also essential for optimizing the effectiveness of teleoperation missions. It takes into account the curvature or sharp turns in the UAV trajectory.

Consequently, I compare the results of the two teleoperation methods: traditional teleoperation and 3D Autocomplete.

Table 4.2: Quantitative results of the human-subject tests

	Average time [sec]		Average Distance [m]		Average Smoothness [radians]	
	Manual	Auto	Manual	Auto	Manual	Auto
Cylinders	39.12	25.12	29.624	20.73	6450.493	3134.975
Cones	36.6	23.86	30.618	20.603	5010.52	2919.78
Boxes	36.68	22.5	27.45383	16.5568	6137.663	3125.872

The average time, distance, and smoothness of 70 tests per method (total of 140 iterations) are shown in Table 4.2. Notably, 3D Autocomplete surpasses manual teleoperation across all metrics. On average, it demonstrates a significant improvement, with approximately 35% less time required to complete the task, a 30% reduction in the distance traveled, and a remarkable 50% improvement in trajectory smoothness when compared to the manual method. Note that the deep learning model predicted correctly all the users motions.

The reduction in time and distance is primarily attributed to the fact that, in many cases, users tend to deviate from the desired trajectory due to the challenges of teleoperating 3D motions. This deviation not only extends the time required to complete their tasks but also results in increased travel distance. On the other hand, while running 3D Autocomplete during teleoperation, the user may initially deviate from the desired trajectory. However, once they accept the suggested motion by 3D Autocomplete, the autopilot completes their motion by following a smooth helix (conical, cylindrical, or rectangular), effectively reducing both the distance traveled and the time required to complete the task by eliminating any possible user deviations during manual teleoperation. These results highlight the effectiveness and efficiency of the Autocomplete system in enhancing performance in all aspects.

The system required minimal time to predict the motion type upon detecting change points, as well as to synthesize the user's motion after user acceptance. This is crucial to ensure the smoothness of the user's motion and prevent delays between user input and autopilot takeover. Furthermore, it confirms that the conducted experiments were in real-time. The average time taken for 25 iterations is presented in Table 4.3.

Table 4.3: Prediction and Motion synthesis processing time

Average time to predict the motion type [ms]	48.63
Average time to synthesize the motion [sec]	0.55

All obtained p-values for the subjective and objective results are very small (< 0.005, see table 4.4 and fig 4.8) which indicates that the results are statistically significant.

Table 4.4: P-values of Quantitative Results

	Distance	Time	Smoothness
Cylinder	0.000103	9.46×10^{-6}	0.00135
Cone	0.00315	2.07×10^{-5}	0.001519
Box	0.001127	0.000185	0.005045

Finally, Fig. 4.9 presents some of the motions tested using manual teleoperation and 3D Autocomplete. The figure illustrates the challenges faced during user teleoperation, where the user's motions appear irregular and occasionally deviate from the desired trajectory. In contrast, when employing the 3D Autocomplete, the autopilot smoothly completes the user's intended motion after predicting its type from partial inputs. This enhances the objective results in Table 4.2, where the trajectories executed by 3D Autocomplete are smoother than the user manual trajectories by approximately 50% as mentioned before.

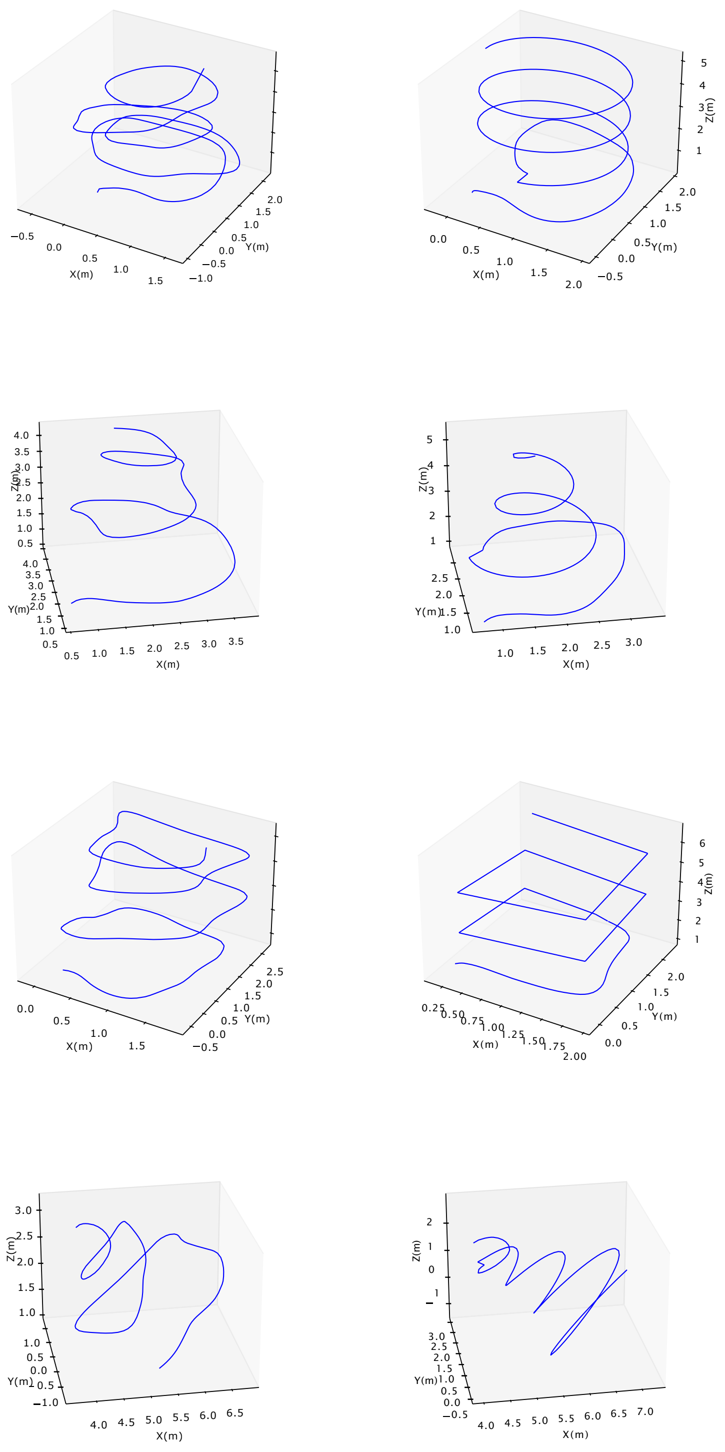


Figure 4.9: 3D Trajectories: using only manual teleoperation (left); and trajectories with 3D Autocomplete (right)

Chapter 5 Discussion

This thesis introduces 3D Autocomplete, a framework designed to assist users in teleoperating UAVs within 3D environments. During teleoperation, 3D Autocomplete uses a change point detection algorithm to identify important changes in the user's motion. When sufficient changes are detected, deep learning is used to classify the user's partial motion as one of the predefined motion primitives (motion around a cylinder, a cone, or a box). Then, mixed reality is used to communicate the proposed trajectories to the user by augmenting the trajectories onto a first-person live video feed from the UAV's camera. Finally, if the user accepts the motion, 3D Autocomplete completes the intended motion autonomously after synthesizing it using LS optimization. To validate 3D Autocomplete, I conducted human-subject tests with 14 participants. These experiments involved teleoperating a simulated UAV on ROS/Gazebo using manual joystick control (traditional teleoperation) and using 3D Autocomplete. In this chapter, I provide a discussion of the findings and achievements resulting from my work. Additionally, I explore the limitations of my system and analyze areas where enhancements can be added for future development.

5.1 Findings and Achievements

Throughout this thesis, I have achieved several significant milestones that contribute to the field of UAV-assisted teleoperation. These accomplishments are detailed below:

- Reducing user's workload during teleoperation: In my subjective evaluation, I assessed the workload using NASA-TLX survey scores. As mentioned in the results, all the demands (effort, temporal demand, mental demand, physical demand) were significantly lower when running 3D Autocomplete during teleoperation. This is because manual teleoperation demands a high level of focus, particularly when controlling a high dimensional system with under-actuated properties. This increases the mental, physical, and temporal loads on the user. However, in the case of 3D Autocomplete, these demands are significantly reduced due to the assistance provided by autonomous driving. Similarly, the effort required from the user while running 3D Autocomplete is distributed between the user and the autopilot, with the autopilot taking over the teleoperation task after a certain period.

- Reducing time required to finish a teleoperation task: As shown in my objective results, the average time needed to finish a certain task using 3D Autocomplete was approximately 35% less than that while using manual driving. This is because 3D Autocomplete takes over the teleoperation task by following a defined (synthesized) trajectory. This approach is more time-efficient compared to an inexperienced user attempting to complete the task due to the challenges of teleoperation, especially in 3D scenarios.

- Reducing covered distance: Based on my objective results, 3D Autocomplete reduced the drone covered distance by 30% compared to manual teleoperation. When the users manually operate the drone, they often perform imprecise helical motion rich in noise and deviations from the intended trajectory. Accordingly, this increases the covered distance during teleoperation. In contrast, 3D Autocomplete synthesizes motion more accurately, enabling the drone to follow a helical path with reduced noise and deviations.

- Increasing the smoothness of the drone trajectory: 3D Autocomplete enhanced trajectory smoothness by 50% when compared to manual teleoperation. This improvement is also attributed to the smooth motions synthesized by 3D Autocomplete during autopilot teleoperation.

- Enhancing human-robot communication: 3D Autocomplete uses an MR interface to communicate the predicted motion to the operator. This communication method significantly improved the system performance, double as that of using manual teleoperation (Fig. 4.8). This is because the usage of a VR headset not only enhances user focus but also improves the user's understanding of robot actions.

- Improving teleoperation performance: This is reflected in both subjective and objective evaluation. The overall system performance is improved compared to manual teleoperation, by decreasing the user workload, the time required to finish the task, the covered distance, and the sharpness/noise of the UAV trajectory.

5.2 Limitations and Possible Enhancements

Even though the proposed system achieved promising results, it is essential to consider certain limitations and explore potential avenues for enhancing 3D Autocomplete. These limitations and possible enhancements are listed below:

- Assessment through simulated experiments: The framework was evaluated through simulation experiments on ROS/Gazebo. Nevertheless, it's crucial to acknowledge that simulations may not fully capture the complexities of real-world scenarios. Therefore, it is important to conduct real-world experiments with an actual drone to introduce a more realistic evaluation.

- Consecutive motions: One of the fundamental concepts of 3D Autocomplete is negotiating 3D objects that are combinations of the considered basic geometric primitives (e.g. tree as a cylinder and a cone). Covering such shapes necessitates a direct transition from one motion to another. However, this impacts the data collected from the user joystick by concatenating both the previous and the new motion data. Therefore, given that this data serves as an input to the deep learning model and the motion synthesis algorithm, it influences the motion prediction and the generated trajectory. Accordingly, it is essential to consider such transition for consecutive motions in future development.

- User takeover: In 3D Autocomplete, even when the autopilot is autonomously completing the user's motion, the user is always in the loop and can take over at any time. While this flexibility is advantageous, transitioning from autopilot to manual control is abrupt and potentially frustrating. This is because the drone is already following a predetermined trajectory at a certain speed, in other words, the user needs time to adapt to the UAV's new position/velocity and initiate a new motion.

- Change point detection sensitivity: Change point detection is used in my system to identify when to do the motion prediction. However, since it relies on the direction of the collected velocity vectors, it is sensitive to the user's joystick commands. In some cases, it may detect a change point that is not actually intended by the user, leading to potential mistakes. Therefore, it is important to advance my algorithm to distinguish the intended change points.

- Unreliable partial motions: After the user accepts the predicted motion, 3D Autocomplete uses the partial motion to synthesize and complete it. In some cases, the user's partial motion deviates from the intended trajectory due to the challenges of performing 3D motions. Accordingly, even if the trained deep model correctly predicted the intended motion, fitting such data can lead to the generation of an unreliable trajectory. I encountered such situations during the testing phase, which resulted in collisions with the considered 3D shape.

Chapter 6

Conclusion

In conclusion, this thesis proposed 3D Autocomplete to aid users in teleoperating UAVs while performing 3D motions. My proposed approach has made significant contributions that are highly valuable in the field of robot teleoperation. Results from 140 iterations confirmed the effectiveness of my system, showcasing quantitative and qualitative superiority over traditional manual teleoperation methods. 3D Autocomplete reduced the time needed to cover the 3D shape, the distance traveled, and the trajectory smoothness by at least 30%. Furthermore, it successfully achieved its primary goal by reducing operators' workload and enhancing the system's performance.

Several approaches were introduced before reaching the final proposed framework. Each of these approaches aimed to resolve specific challenges faced by the users or even the system itself during Autocomplete teleoperation. A significant issue addressed was the determination of when to predict the intended motion type. Instead of using a constant time interval, 3D Autocomplete employs a change point detection algorithm to identify substantial changes in the user's partial motion, enabling accurate motion prediction. Real-time simulations demonstrated the algorithm's success in detecting the motion turns around the 3D shapes, leading to approximately 80% F1-score of early motion prediction. Another significant aspect is the proposed deep learning model which is based on recurrent neural networks. It predicts the user's motion from the joystick commands, specifically velocity vectors. This approach can be generalized to accommodate various motion types with similar time series characteristics. Additionally, my framework succeeded in solving the problem of a constant number of samples in the input layer of the deep learning model. In more detail, instead of using only the first 'n' samples of the user's motion, 3D Autocomplete takes the entire sequence and downsamples it smoothly before the prediction. In cases requiring upsampling, zero padding is employed. These have been shown to make my system scale-agnostic while maintaining the accurate predictions of the model. Finally, this thesis enhanced the communication between users and the assisted teleoperation system through the use of an MR interface, allowing users to maintain better focus and consequently enhancing the overall system performance as illustrated in the experimental results.

In future work, I aim to extend my evaluation to real-life test scenarios, building upon the promising results obtained in simulation. This transition from simulation to real-life testing will provide valuable insights into the practical applicability and performance of the system. Moreover, I aspire to evaluate the generalizability of 3D Autocomplete for coordinating a formation of UAVs. Enhancing user teleoperation for such scenarios is crucial, as it addresses the increasing challenges posed by these demanding tasks. Finally, since the communication between the operator and the UAV is an essential component in 3D Autocomplete (through the VR headset or the joystick), it is important to address the possible loss of communication during teleoperation.

Bibliography

[1] T. B. Sheridan, "Human–robot interaction: Status and challenges," 4, vol. 58, SAGE Publications Sage CA: Los Angeles, CA, 2016, pp. 525–532.

[2] K. Hambuchen, J. Marquez, and T. Fong, "A review of nasa human-robot interaction in space," 3, vol. 2, Springer, 2021, pp. 265–272.

[3] H. T. Nguyen, T. V. Quyen, C. V. Nguyen, A. M. Le, H. T. Tran, and M. T. Nguyen, "Control algorithms for uavs: A comprehensive survey," 23, vol. 7, 2020, e5–e5.

[4] D. K. Villa, A. S. Brandao, and M. Sarcinelli-Filho, "A survey on load transportation using multirotor uavs," vol. 98, Springer, 2020, pp. 267–296.

[5] A. D. Boursianis, M. S. Papadopoulou, P. Diamantoulakis, et al., "Internet of things (iot) and agricultural unmanned aerial vehicles (uavs) in smart farming: A comprehensive review," vol. 18, Elsevier, 2022, p. 100 187.

[6] X. Li and A. V. Savkin, "Networked unmanned aerial vehicles for surveillance and monitoring: A survey," 7, vol. 13, MDPI, 2021, p. 174.

[7] S. De and D. Guida, "Control design for an under-actuated uav model," 4, vol. 46, 2018, pp. 443–452.

[8] T. Fong, C. Thorpe, and C. Baur, "Collaborative control: A robot-centric model for vehicle teleoperation," vol. 1, Carnegie Mellon University, The Robotics Institute Pittsburgh, 2001.

[9] M. K. Zein, A. Sidaoui, D. Asmar, and I. H. Elhajj, "Enhanced teleoperation using autocomplete," IEEE, 2020, pp. 9178–9184.

[10] M. K. Zein, M. Al Aawar, D. Asmar, and I. H. Elhajj, "Deep learning and mixed reality to autocomplete teleoperation," IEEE, 2021, pp. 4523–4529.

[11] M. H. Hussein, I. H. Elhajj, and D. Asmar, "Personalized autocomplete teleoperation: Real-time user adaptation using transfer learning with partial feedback," IEEE, 2021.

[12] M. H. Hussein, B. Ibrahim, I. H. Elhajj, and D. Asmar, "Incremental learning for enhanced personalization of autocomplete teleoperation," 2022.

[13] H. Zhang, L. Zhu, J. Shen, and A. Song, "Implicit neural field guidance for teleoperated robot-assisted surgery," IEEE, 2023, pp. 6866–6872.

[14] G. Maeda, "Blending primitive policies in shared control for assisted teleoperation," IEEE, 2022, pp. 9332–9338.

[15] A. Gottardi, S. Tortora, E. Tosello, and E. Menegatti, "Shared control in robot teleoperation with improved potential fields," 3, vol. 52, IEEE, 2022, pp. 410–422.

[16] I. Havoutis and S. Calinon, "Supervisory teleoperation with online learning and optimal control," IEEE, 2017, pp. 1534–1540.

[17] M. Ewerton, O. Arenz, and J. Peters, "Assisted teleoperation in changing environments with a mixture of virtual guides," 18, vol. 34, Taylor & Francis, 2020, pp. 1157–1170.

[18] S. Dass, K. Pertsch, H. Zhang, Y. Lee, J. J. Lim, and S. Nikolaidis, "Pato: Policy assisted teleoperation for scalable robot data collection," 2022.

[19] Q. Wang, B. He, Z. Xun, C. Xu, and F. Gao, "Gpa-teleoperation: Gaze enhanced perception-aware safe assistive aerial teleoperation," 2, vol. 7, IEEE, 2022, pp. 5631–5638.

[20] S. Chen, J. Gao, S. Reddy, G. Berseth, A. D. Dragan, and S. Levine, "Asha: Assistive teleoperation via human-in-the-loop reinforcement learning," IEEE, 2022, pp. 7505–7512.

[21] X. Yang, A. Agrawal, K. Sreenath, and N. Michael, "Online adaptive teleoperation via motion primitives for mobile robots," 6, vol. 43, Springer, 2019, pp. 1357–1373.

[22] F. Perez-Grau, R. Ragel, F. Caballero, A. Viguria, and A. Ollero, "Semi-autonomous teleoperation of uavs in search and rescue scenarios," IEEE, 2017, pp. 1066–1074.

[23] B. Xu and K. Sreenath, "Safe teleoperation of dynamic uavs through control barrier functions," IEEE, 2018, pp. 7848–7855.

[24] L. Zhang, Y. Zhang, and Y. Li, "Path planning for indoor mobile robot based on deep learning," vol. 219, Elsevier, 2020, p. 165 096.

[25] P. Long, W. Liu, and J. Pan, "Deep-learned collision avoidance policy for distributed multi-agent navigation," vol. abs/1609.06838, 2016.

[26] A. Bouguettaya, H. Zarzour, A. M. Taberkit, and A. Kechida, "A review on early wildfire detection from unmanned aerial vehicles using deep learning-based computer vision algorithms," vol. 190, Elsevier, 2022, p. 108 309.

[27] M. Gao, J. Oberländer, T. Schamm, and J. M. Zöllner, "Contextual task-aware shared autonomy for assistive mobile robot teleoperation," IEEE, 2014, pp. 3311–3318.

[28] D. Vanhooydonck, E. Demeester, A. Hüntemann, et al., "Adaptable navigational assistance for intelligent wheelchairs by means of an implicit personalized user model," 8, vol. 58, Elsevier, 2010, pp. 963–977.

[29] K. Khokar, R. Alqasemi, S. Sarkar, K. Reed, and R. Dubey, “A novel telerobotic method for human-in-the-loop assisted grasping based on intention recognition,” IEEE, 2014, pp. 4762–4769.

[30] M. Laskey, C. Chuck, J. Lee, et al., “Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations,” IEEE, 2017, pp. 358–365.

[31] T. Zhou, Q. Zhu, and J. Du, “Intuitive robot teleoperation for civil engineering operations with virtual reality and deep learning scene reconstruction,” vol. 46, Elsevier, 2020, p. 101 170.

[32] S. Li, X. Ma, H. Liang, et al., “Vision-based teleoperation of shadow dexterous hand using end-to-end deep neural network,” IEEE, 2019, pp. 416–422.

[33] T. Zhang, Z. McCarthy, O. Jow, et al., “Deep imitation learning for complex manipulation tasks from virtual reality teleoperation,” IEEE, 2018, pp. 5628–5635.

[34] P. Wang, S. Zhang, M. Billinghurst, et al., “A comprehensive survey of ar/mr-based co-design in manufacturing,” vol. 36, Springer, 2020, pp. 1715–1738.

[35] A. J. Lungu, W. Swinkels, L. Claesen, P. Tu, J. Egger, and X. Chen, “A review on the applications of virtual reality, augmented reality and mixed reality in surgical simulation: An extension to different kinds of surgery,” 1, vol. 18, Taylor & Francis, 2021, pp. 47–62.

[36] Z. Pan, A. D. Cheok, H. Yang, J. Zhu, and J. Shi, “Virtual reality and mixed reality for virtual learning environments,” 1, vol. 30, Elsevier, 2006, pp. 20–28.

[37] K. A. Szczurek, R. M. Prades, E. Matheson, J. Rodriguez-Nogueira, and M. Di Castro, “Multimodal multi-user mixed reality human–robot interface for remote operations in hazardous environments,” vol. 11, IEEE, 2023, pp. 17 305–17 333.

[38] D. Sun, A. Kiselev, Q. Liao, T. Stoyanov, and A. Loutfi, “A new mixed-reality-based teleoperation system for telepresence and maneuverability enhancement,” 1, vol. 50, IEEE, 2020, pp. 55–67.

[39] H. Hedayati, M. Walker, and D. Szafir, “Improving collocated robot teleoperation with augmented reality,” 2018, pp. 78–86.

[40] M. E. Walker, H. Hedayati, and D. Szafir, “Robot teleoperation with augmented reality virtual surrogates,” IEEE, 2019, pp. 202–210.

[41] D. Lee and Y. S. Park, “Implementation of augmented teleoperation system based on robot operating system (ros),” IEEE, 2018, pp. 5497–5502.

[42] K. Sung and S. Gregory, “Basic math for game development with unity 3d,” Springer, 2019.

[43] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” IEEE, vol. 3, 2004, pp. 2149–2154.

[44] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[45] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality reduction for fast similarity search in large time series databases," 3, vol. 3, Springer, 2001, pp. 263–286.

[46] J. M. ain39;t a mathematician (https://math.stackexchange.com/users/498/j-m-aint-a-mathematician), "Equation of a rectangle." eprint: https://math.stackexchange.com/q/69134.

[47] T. Muslimov and R. Munasypov, "Fuzzy model reference adaptive control of consensus-based helical uav formations," IEEE, 2022, pp. 196–201.

[48] S. Kumar and S. R. Kumar, "Barrier lyapunov-based nonlinear trajectory following for unmanned aerial vehicles with constrained motion," IEEE, 2022, pp. 1146–1155.

[49] M. A. H. Abozied and S. Qin, "High performance path following for uav based on advanced vector field guidance law," IEEE, 2016, pp. 555–564.

[50] S. G. Hart, "Nasa-task load index (nasa-tlx); 20 years later," 9, Sage publications Sage CA: Los Angeles, CA, vol. 50, 2006, pp. 904–908.