Sheffield Hallam University | College of Business, Technology and Engineering

MSc Dissertation Report

"Engineering Generative Models to Synthesize Human Hand Grasping Poses Based on Object Affordances for Robots"

A dissertation submitted in partial fulfilment of the requirements of Sheffield Hallam University for the degree of Master of Science in Artificial Intelligence

Student Name	Siavash Mortaz Hejri
Student ID	32056757
Supervisor	Dr. Alejandro Jimenez Rodriguez, Dr. Hamed Pourfannan
Date of Submission	12th September

ABSTRACT

This dissertation explores the engineering of generative models for synthesizing human hand poses for grasping action using object visual information alone, without relying on manual object affordance labeling of the object. The proposed model generates grasping poses for humanoid robotic systems without requiring explicit programming or manual intervention. The core of this research lies in the use of Conditional Variational Autoencoders (CVAE), which predict and synthesize hand poses upon perception of object affordances. These models are trained on the HO-3D_v3 dataset, which contains diverse 3D hand-object interactions. This paper presents the performance of CVAE in enhancing adaptability and efficiency of robot grasping of everyday objects which has the potential to facilitate the ability of humanoid robots for autonomous affordance-based object manipulation in real-world tasks.

The key contributions of this work are to provide a robust CVAE architecture that integrates object information to generate realistic hand poses and demonstrate its generalization across various objects and scenarios. In this study, several versions of the CVAE architectures with different complexity levels were iteratively evaluated to obtain the optimal performance toward reducing prediction error and improving accuracy in synthetic grasp poses. The final model, CVAE_02_3, further enhances this by removing the encoder's condition layer, which allows for a latent space that generalizes better and provides for more accurate, contextually appropriate hand poses.

This work represents a step into the next phase of robotic affordance-based object grasping and manipulation, where generative models open up a new avenue toward synthesizing human-like hand activities that not only help bridge the gap from static, predefined robotic behaviors to dynamic, flexible interactions but also point out some opportunities of long-term improvement for CVAEs in enriching the capability of robotic systems to be autonomous, responsive, and effective in real-world environments. The source code of this project (Appendix D) is available at the link: https://github.com/Siavash-Mortaz/Grasp_Pose_Generation.

ACKNOWLEDGEMENTS

I would like to express my gratitude to the tutors at Sheffield Hallam University, especially my supervisors, whose support and encouragement were crucial to the completion of this research project. Their guidance made this work possible. I am also deeply thankful to my friends and family, whose ongoing support and assistance, including help with proofreading, played a vital role in the success of this project.

Contents

1.0 INTRODUCTION

How can the human hand pose for grasping action be synthesized based on object affordances using generative models?

1.1 Project Rationale

Nowadays, robots have become an inseparable part of our daily lives. They have the potential to significantly contribute to and perform daily tasks, such as cleaning, cooking, and providing companionship. For example, robotic vacuum cleaners like the Roomba (Roomba Robot Vacuums, 2024) can efficiently clean floors without human intervention. Additionally, robotic chefs can prepare meals based on specific dietary preferences and allergies (BARAKAZI, 2022). Recently, their tasks have become more complicated and demanding. To navigate complex and real-world environments—such as busy city streets or disaster zones—and carry out meaningful tasks like autonomous driving or search and rescue operations, they need more than just advanced mechanical abilities; they also require sophisticated decision-making and problem-solving capabilities to interact with the environment and objects adaptively and efficiently. This requires robots not only to recognize objects but also to understand how they can be approached, grasped, and manipulated based on their functionality—an ability known as object affordance (Kim et al., 2014).

Most robots are traditionally designed and programmed based on pre-defined methods such as global path planning, local motion planning, and layered navigation stack (Cèsar-Tondreau et al., 2021). While global path planning depends on pre-existing maps to compute an optimal route, local motion planning makes real-time adjustments of the movements to avoid obstacles. The layered navigation stack combines both approaches. All these traditional methods may not be flexible enough to handle unforeseen situations or real-time interaction with objects. Moreover, in these situations updating their software is a high-cost and time-consuming job. Due to these limitations, researchers are moving toward new and advanced techniques that allow the robots to generate and execute the actions autonomously, one of the most complicated types of which is the fine hand movements required for object manipulation. Therefore, this dissertation explores whether grasping actions can be reconstructed based on object affordances. This serves as a practical definition of what an affordance is. Prediction and simulation of human-like

grasping from object affordances is another key step in developing more autonomous and adaptive robotic systems capable of interacting with their environment like humans.

1.2 Project Scope

This dissertation focuses on leveraging generative AI models to synthesize human hand poses based on the concept of object affordances. There are many Generative AI models (like Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs)) that can learn from vast datasets of human-object interactions that are collected through real-world environments from sensors, cameras, and other devices. Through this training, they can create a realistic simulation of their future hand poses based on object affordances. Simulation allows robots to engage in mental imagery by enabling them to visualize and predict actions in a virtual environment (Lin et al., 2023). This process refines grasping strategies for more accurate and adaptive behavior in real-world applications just like cognitive processes work in humans to develop intelligent responses to dynamic environments and complex tasks. The research is focused on generating grasps from object features, rather than conducting robot experiments. Using simulated grasping poses, the system predicts the best hand pose configurations to interact with different objects due to their affordances without the explicit programming of every scenario. This will give insight into how grasping poses can be derived from object characteristics, which is the central aim of this work.

Such an ability also allows the robots to be more flexible and autonomous in their interaction and finally synthesize human-like hand action in response to different objects and their functionality. It will be achieved by embedding the generative models described in robots that are in a better position to understand how to manipulate objects and adjust their strategies in real-time, making efficient and contextually appropriate decisions. The current study represents a relevant milestone concerning the development of intelligent robotic systems able to handle objects by hand-grasping poses in complex and dynamic real-world scenarios.

1.3 Project Aims

This work aims to reconstruct human hand poses considering object properties and leverage generative models for the purpose. By emphasizing how object features can guide the model through appropriate grasping poses, this work develops the generative model's ability to predict and generate the grasping pose to enhance object affordances in human-robot interaction. In the short term, the aim is to develop the current capabilities of generative models, specifically Variational Autoencoders (VAEs), so that they will be

able to reconstruct such poses as well as possible. The long-term view is to contribute to the development of more adaptive and autonomous robotic systems, able to use such insights for their interactions with the environment and objects in a more human-like and flexible manner, which enhances the utility and effectiveness of robots in various real-world applications.

1.4 Project Objectives

I. To identify and evaluate existent generative models for synthesizing human hand poses from the affordances of objects for robotic systems.

II. Develop and train a generative model, particularly VAEs, for the prediction and generation of human hand poses based on different objects to be used for robotic object manipulation tasks.

III. To explore the effectiveness of different VAEs' architectural choices in the quality of the hand pose reconstruction as well as possible.

IV. Evaluate the efficiency of the trained model with improvements in human-like hand poses in various dynamic environments.

1.5 Project Benefits

A Variational Auto Encoder VAE model is trained with data on human hand poses and object affordances, designed to synthesize and predict appropriate hand poses for robotic systems. This tool can be utilized as a standalone resource by researchers and roboticists for integration into their systems, to allow their systems to perform human-like object grasping and manipulation in different environments.

1.6 Achieving the Research Objective

To achieve the objectives of this research, the current dissertation is approached through a systematic framework that begins with a broad exploration of generative models for synthesizing human hand poses, which is then refined to focus on their specific application in robotic systems. Initially, the research considered the general concept of engineering generative models for robots to simulate future actions through mental imagery. It is narrowed to concentrate specifically on the synthesis of human hand poses using generative AI models, ensuring a more focused and practical scope.

This paper identifies and critically reviews relevant generative models appropriate for the task, such as Variational Autoencoders, which can be used for predicting and generating human-like hand poses from object affordances. Experiments are conducted to test the

effectiveness of the models and the accomplishment of the research objective. The initial proposal can be found in the research project plan (Appendix A).

2.0 LITERATURE REVIEW

Utilizing generative models to synthesize human hand poses based on object affordances is an emerging area that includes cognitive science, robotics, and artificial intelligence. This thematic literature review considered key findings and theoretical perspectives from relevant studies. This review seeks to explain the significance, challenges, and future directions to provide a comprehensive understanding of the field.

2.1 Object Affordances

2.1.1 Definition and Importance

Object affordances refer to a property of the object that relates to potential actions that it can afford or offer to an individual (Hou et al., 2021). For instance, a door handle is for grasping and thus for pulling, and therefore this turns out to be its use without any other indication. This concept is crucial in design, particularly in designing user interfaces, where the user can be directed through visual marks on how to respond or interact with various components. These are foundational ideas in psychology, robotics, and human-computer interaction since they allow humans to interact with their environment and describe how machines could do the same. Recent studies (Ventura, Sara and Tessari, Alessia, 2021) (Borghi, Anna M, 2021) have further elaborated this traditional understanding of affordances concerning their dynamic nature in human-robot interactions and virtual environments, particularly underlining the modulations induced by contextual and social factors and their implications for embodiment in virtual environments. The development of this concept underlines the persistent role affordances play in the design of systems that are intuitive to human abilities and expectations.

2.1.2 Application in Robotics

Affordance-based models help robots understand and predict the use of objects in their environment, therefore enabling more natural and effective interaction.

Renaudo et al. (2022) highlighted that the affordance-based models could offer benefits in robotics by improving object recognition by paying attention to the functionalities of objects, besides bringing simplicity in decision-making. Affordances- a psychological concept, is applied in robotics; the way robots perceive the environment and interact with it. Primary contributions are automatic affordance-based object generation, autonomous learning through sensorimotor contingencies, and online affordance detection using deep

learning. This article pinpoints an increasingly active role that affordances are beginning to play in the design of more autonomous and cognitively endowed robots.

Imre et al. (2019) presented a computational framework in which robots can exhibit altruistic behavior by predicting human intentions from affordance-based actions. With the model, robots will thus be aware of whether or not humans need help and interfere accordingly, similar to human behavior. While this research looked into the development of robots behaving like useful assistants, challenges also remain regarding the gentleness of action to be performed by robots, understood by humans more easily.

Furthermore, improving robots' ability to understand human speech involves teaching them to recognize objects affordance (like a cup can be used for drinking) and then using that understanding to carry out tasks based on what the human wants. Mi et al. (2019) proposed a framework that combines deep learning techniques with the semantics of spoken language to help the robot understand what and where to pick up an object given the context. It enables them to understand verbal commands and identify objects with respect to their affordances, like drinking from a cup or writing using a pen.

Nguyen et al. (2022) formulated a model that enables the robot to follow natural language directives using the functional affordance of objects rather than naming or specific features. The robots can perform tasks such as "Give me something to cut" since the robot identifies objects that can fulfill that action if the robot is not familiar with them. Using the relationship between verbs and object function, the method increases the adaptability and intuitiveness of robots in human-robot interactions.

Beßler et al. (2020) presented a new model that helps robots understand object affordances or the action possibilities afforded by objects to perform tasks flexibly. This model lets a robot decide which objects can be used for a given task and what actions can be executed on a given set of objects. It assumes the presence of a simulation-based mechanism that verifies these affordances and confirms that the actions can be carried out in realistic environments.

However, in some domains, humanoid robots are to perform complex manipulation tasks under challenging conditions, like nuclear decommissioning. Pohl et al. (2020) described a semiautonomous system that combines human expertise and robotic control, whereby the remote operator selects grasp affordances in a Virtual Reality (VR) environment and lets the robotic system execute them. The concept has been tested on the ARMAR-6 robot to enable advanced object manipulation in novel and cluttered settings where full

autonomy would struggle. Indeed, the results show that with this sort of hybrid approach, operational efficiency is enhanced and grasp failure is reduced. It is suitable for complex tasks, such as handling radioactive material in nuclear power plants.

Moreover, Seepanomwan (2019) designed a computational model for teaching humanoid robots to grasp an object that is far from them using tools. Experimentation and reinforcement learning let the robots learn fundamental motor skills and simulate concepts in their mind. The robots do imaginative reasoning to evaluate which ones will be more effective for certain actions, hence saving time and energy.

2.2 Generative Models: Envisioning the Future

2.2.1 The Rise of Generative Models

This work will utilize generative models to reconstruct human hand poses based on object affordances. These days, Generative models, such as Variational Autoencoders (VAEs), Conditional Variational Autoencoders (CVAEs), and Generative Adversarial Networks (GANs) have been experiencing a revolution within artificial intelligence, particularly due to their ability to generate coherent, novel data based on learned patterns. Also, they can learn from unlabeled datasets to create new data examples (Lamb, 2021).

Generative adversarial networks (GANs) are good at making things like pictures look real. They are used for many different tasks like generating human-like actions. Degardin et al. (2022) introducesd Kinetic-GAN, a new framework combining Generative Adversarial Networks (GANs) and Graph Convolutional Networks (GCNs) for synthesizing human actions. Kinetic-GAN generates realistic body movements for up to 120 actions. It leverages spatiotemporal graph convolutions to maintain skeletal structure and uses latent space disentanglement and stochastic variations to enhance action diversity and quality.

Variational Auto-Encoders (VAEs) have been very successful in generating images, predicting mutations, and designing proteins. Tahir et al. (2021) proposed a method for 3D object reconstruction from a single 2D image using deep learning techniques: autoencoder (AE) and Variational Auto-encoder (VAE). One thing emphasized in the approach is the development of a 3D variant model that should be smoother and more accurate. The encoder trains a 2D image compressed to its latent representation, and a decoder is used to reconstruct a 3D model.

On the other hand, VAEs conditional variant (CVAEs), have been successfully applied to human motion reconstruction. Cai et al. (2021) presented a unified model of 3D human

motion synthesis with CVAE given tasks like motion prediction, completion, interpolation, and spatial-temporal recovery. It takes any input as a masked motion series and estimates the missing regions to synthesize it realistically. It includes an AAM that controls motion styles according to action labels and a cross-attention mechanism to enhance realism and consistency.

However, working with these models remains challenging because the design relies on game theory, a mathematical theory, that is different from most other programs' structures (Goodfellow et al., 2020).

By considering these models, this dissertation focuses on improving the models and generalization of hand pose generation, allowing the generative model to predict appropriate grasping configurations based on the object.

2.2.2 Applications in Robotics

3D simulated environments are places for robots to learn skills before applying them in a real-world environment. It can be practical with generative models of 3D content. Chaudhuri et al. (2020) explored generative models of 3D shapes and environments and examined various methodologies such as probabilistic models, deep generative models, and neural network models for structured data. Also, probabilistic learning models are applied to another study to cope with variations in object size, position, and orientation (Tanwani, 2018). The paper developed the model to adapt it in real-time scenarios and learn new skills online.

With the aid of deep generative models (DGMs), Bütepage et al. (2020) use a new probabilistic latent variable model that predicts movements in a hidden space. Their experiments show that successful human-robot interaction (HRI) depends on a model that can consider human motion and task dynamics. Consequently, it can create synchronized behavior between humans and robots. In addition, Krupnik et al. (2023) work on modern deep generative models accelerated by GPUs to enhance robot ability to moderate divers' situations.

Despite all this research, there are various challenges in robotics around understanding spatial environments. To address this, DGSM, a new deep generative spatial model for robotics is introduced by (Pronobis & Rao, 2017). The article claims that DGSMs could learn a deep probabilistic representation of robotic environments from different views including low-level features, geometry, and semantics.

For mimicking the actions of humans, researchers are inspired by the structure of human motor control to design a hierarchical model for robots which could enable them to autonomously complete their duties (Yuan et al., 2023). Due to extensive simulations, robots successfully completed complex tasks and had their performance toward obstacles or damages.

2.3 Human Hand Action Synthesis

Synthesizing human hand actions involves generating plausible hand movements that can interpret and interact with objects as done by humans. One could synthesize realistic hand actions from generative models trained from large datasets of human-object interactions.

The article, (Cordella et al., 2014) aims at the reproduction of human grasping motions to enable robotic hands to grasp objects stably in a human-like manner. The paper reports on the analysis of human hand motion by acquiring data using an optoelectronic system from six subjects performing a transverse volar grasp. These quantitative indicators have been extracted for joint positions, finger behavior, etc., and used to optimize grasping postures in robots.

Authors Zhou et al. (2020) propose a novel approach for real-time estimation of hand pose and shape using only a single RGB camera. Their system can predict 3D positions and rotations of hand joints at the extraordinary frame rate of 100 fps through the combination of image data with either 2D or 3D annotations and motion capture data without images. It employs two neural networks, DetNet for joint detection and IKNet for inverse kinematics.

Garcia-Hernando et al. (2018) introduce a new dataset, which is one of the hand manipulations in egocentric videos. This dataset contains over 100,000 frames of hands contacting several objects and is 3D annotated on the hand pose. The paper shows that the performance in action recognition dramatically improves if the 3D hand poses are used in cases where hands are partially obscured by objects.

Although the study provides the framework for generating realistic actions, this challenge is supposed to annotate the 3D poses of hands and objects. Hampali et al. (2020) present a 3D annotation approach for hand and object poses in images. Its key feature is to focus on challenging interactions, including mutual occlusions of hands and objects. Sequences captured with one or several RGB-D cameras and applying joint optimization across frames to get very accurate 3D pose estimates. This approach has led to the creation of the HO-3D dataset, which is used for this dissertation.

2.4 Integration of Affordances and Generative Models

To show how people use their hands to interact with objects in various ways, Jian et al. (2023) introduce AffordPose, a dataset of human hands manipulating objects, putting particular emphasis on the affordances of the object. Instead of general tasks of grasping, focusing on the specific functional roles an object can play allows for better prediction and interaction generation in computer models. They report that different affordances give rise to specific hand configurations and can enhance the models on affordance understanding or interaction generation.

Manual labeling of important objects is highly laborious and does not scale. Bertasius et al. (2017) present the Visual-Spatial Network (VSN), a self-supervised network to detect important objects in first-person view videos. It consists of a segmentation agent proposing important objects and a recognition agent, which refines these proposals based on both appearance features and object locations. Therefore, VSN is far more efficient and can scale better by learning from unlabeled data than supervised methods using labeled data.

Further, Do et al. (2018) defined a new deep learning model for real-time object recognition, AffordanceNet, with which computers and robots could understand the possible use of an object. AffordanceNet detects objects and estimates the affordance labels for every pixel in RGB images, using advanced techniques like deconvolutional layers, resizing methods, and a multi-task loss function. Results showed that AffordanceNet promises very great potential for real-time robotic object grasping and manipulation tasks.

2.5. Literature Review Summary

Previous studies on robotic systems have extensively focused on the relation of an object to its inherent affordances. However, most of these studies have used the objects themselves as basic entities in their analysis and often neglected to consider the dynamic relation between object features and hand poses. Indeed, no reported work identified predicts the poses induced by the object's features which, in turn, reconstruct the affordance. Here, affordance is reconceptualized in this context as not just a property of the object but a relational model between the object and the hand, which is encoded within the predictive system. This represents a significant development in the integration of generative models with this nuanced understanding of object affordance and synthetic hand pose. This approach enhances the capability of the robot to handle complex tasks

that require a great deal of dexterity and flexibility in their operations, much like human capabilities, which adds to their effectiveness in a wide variety of applications.

Theme		Author	Key Points	Challenges	Future Directions
Object Affordances	Definition and Importance	Hou et al., 2021	Definition and significance of affordances in cognitive science, design, and robotics.	Understanding object affordances in complex environments.	Refining models for human-like robotic object interaction.
		Ventura, Sara and Tessari, Alessia, 2021	Dynamic nature of affordances in human-robot interaction and virtual environments.	Modulations due to social and contextual factors.	Improving virtual and robotic affordance modeling for varied contexts.
	Application in Robotics	Renaudo et al., 2022	Affordance-based models enable robots to predict object use and enhance object recognition.	Real-time affordance prediction and autonomous learning difficulties.	Enhancing real-time decision-making and affordance mapping.
		Mi et al., 2019	Affordance models help robots interpret human speech and commands more intuitively.	Adapting to novel commands or objects not previously encountered.	Further refining verbal command understanding in robots.
		Pohl et al., 2020	Semi-autonomous models combining human control and robotic grasp affordances.	Complex tasks in unpredictable environments require human-robot collaboration.	Improving human-robot collaboration for high-stakes tasks.
Generative Models	The Rise of Generative Models	Degardin et al., 2022	Kinetic-GAN framework synthesizes realistic human actions with GAN and GCN integration.	Action diversity and long-term synthesis challenges.	Enhancing action diversity and quality for long-term action generation.
		Tahir et al., 2021	3D object reconstruction using Variational Autoencoders for smoother, accurate models.	Training deep models for high-resolution 3D reconstructions.	Developing more efficient 3D object generation systems.
		Cai et al., 2021	Human motion reconstruction via CVAEs for realistic human motion prediction.	Improving synthesis realism and consistency	Incorporating advanced motion control and

				across varied motion tasks.	prediction models.
		Goodfellow et al., 2020	Challenges in generative models due to complex game theory-based designs.	Designing stable and effective generative models for new applications.	Addressing design and training challenges with GAN-based models.
	Applications in Robotics	Chaudhuri et al. (2020); Bütepage et al. (2020); Pronobis & Rao (2017)	Generative models can be used to simulate realistic robotic actions and learn from 3D environments, while challenges remain in real-time application and generalization.	Adapting generative models to dynamic, real-time scenarios with variable object features.	Enhancing the adaptability of generative models across diverse real-world environments and their integration within robotic setups.
Human Hand Action Synthesis		Cordella et al., 2014	Analyzes human hand motion to optimize robotic grasps using a Nelder-Mead algorithm for configuration.	The robotic hand's mechanical constraints limited optimal grasping, especially thumb opposition.	Improve thumb design and generalize grasp for diverse, natural handling.
		Zhou et al., 2020	A real-time hand motion capture system uses RGB images and two neural networks, achieving 100fps.	Monocular RGB hand tracking faces depth ambiguity, occlusions, fast movements, and limited data.	Using dense 3D scans for texture capture and tracking two interacting hands from one RGB image.
		Garcia-Hernando et al. (2018); Hampali et al. (2020)	Human hand action synthesis mainly deals with generating realistic hand motions, normally requiring accurate 3D pose data and solving occlusions in hand-object interactions.	Occlusion handling and ensuring proper 3D hand pose annotation in cluttered scenes.	3D hand pose estimation is further improved to consider situations with occlusions and dynamic conditions.

Integration of Affordances and Generative Models	Jian et al. (2023); Bertasius et al. (2017); Do et al. (2018)	Combining affordances with generative models, enhancing the robot's understanding and execution of tasks, particularly object recognition and manipulation based on functional affordances.	Scaling of these models to real-time applications and manipulation of objects precisely.	Advancing of a combination of generative models with a theory of affordances to create better autonomous interaction by robots.

Table 1: Literature review of key values, challenges, and future directions

3.0 METHODOLOGY

This section describes the research methodology, including the guiding philosophy, tools, techniques, and processes for developing generative models of human hand-grasping poses based on object affordances. It covers data analysis, model training frameworks, and evaluation techniques, providing a foundation for understanding the experimental design and model refinement.

3.1 Research Philosophy, Approach, and Methodology

This study adopts a mixed philosophical approach, combining pragmatism and positivism. Pragmatism focuses on the development of practical models and positivism assumes affordances are real and can be objectively modeled. Additionally, the core hypothesis is that hand poses can be reconstructed from object affordances based on prior works that used generative models like VAEs. For model development and refinement, the study follows abductive reasoning which means that the solutions could be exploratory through iterative testing and data-driven adaptation. Experimentation and simulation of model performance evaluate and refine the algorithms for further model improvement. This is an interdisciplinary approach, which lays down a good basis for assessing how well models predict and generate appropriate hand poses for various objects and making significant strides in the field of human-like robotic manipulation. Figure 1 shows the diagram based on the methodologies of the dissertation.

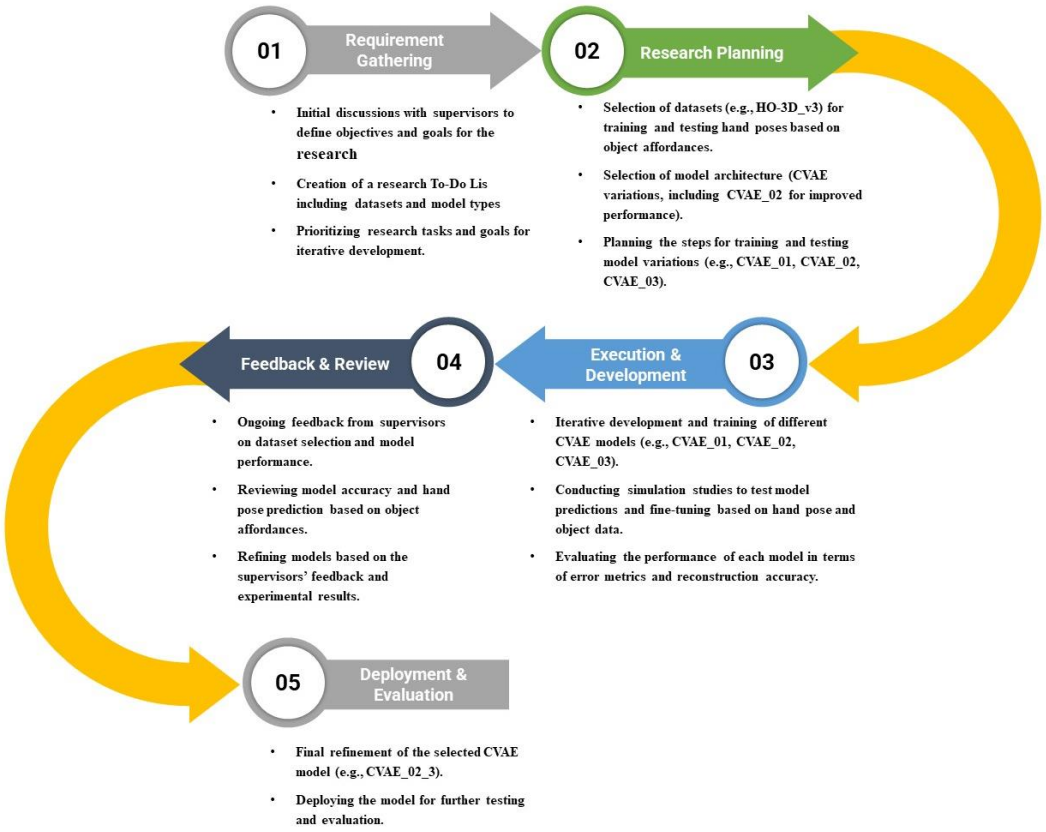


Figure 1: This diagram illustrates a five-phase process for developing and evaluating Conditional Variational Autoencoder (CVAE) models for hand pose prediction based on object affordances.

3.2 Tools and Techniques

Generative models of robotic hand-posed synthesis based on object affordances are developed and evaluated using several tools, techniques, and resources. This involves generative models, software frameworks, data preprocessing libraries, development environments, computing hardware, and visualization tools.

I. **Generative Models:**

- **Variational Autoencoders (VAEs):** VAEs are a better solution for feature representation, especially when capturing the underlying structure typically demonstrated by human hand poses.

II. **Software Frameworks:**

- **PyTorch** (Pytorch, 2024)**, TensorFlow** (TensorFlow, 2024)**, Keras** (Keras, 2024)**:** Each of these has built-in functions for the creation of neural networks and training of neural networks, which accelerates the whole development process. PyTorch is noted for flexibility and ease in debugging; TensorFlow is noted for scalability and production deployment, while Keras is noted for simplicity and user-friendliness.

III. **Data Preprocessing Libraries:**

- **NumPy** (NumPy, 2024)**, Pickle** (Pickle, 2024)**, Scikit-learn** (scikit-learn, 2024)**:** These are the libraries that are utilized to generate and handle data in Python: formatting and cleaning, saving, and preparing before input into the generative models.

IV. **Development Environment:**

- **Visual Studio Code** (Visual Studio Code, 2024)**, PyCharm** (PyCharm, 2024)**, and Jupyter Notebooks** (Jupyter, 2024)**:** All these development environments are extremely supportive of coding, integrating numerous frameworks and libraries.

V. **Computing Hardware:** A personal laptop has been used for training, calculating errors, and evaluation of all the models. The hardware and software configuration of the laptop is mentioned in Table 2.

Brand And Model	Lenovo Legion Y540
CPU	Intel(R) Core(TM) i7-9750H (2.60GHz 2.59 GHz)
GPU_0	Intel(R) UHD Graphic 630
GPU_1	NVIDIA GeForce GTX 1650 (4.0 GB)
RAM	24 GB
OS	Windows 11 Version 22H2 (OS Build 22621.3880)
PyCharm	2023.1.5 (Community Edition)
Python	3.10

Table 2: Configuration of laptop and details of software which is used for training and evaluation in this dissertation.

VI. **Visualization Tools:**

- **Matplotlib** (matplotlib, 2024)**:** Matplotlib is highly customizable for producing a wide range of plots and graphs; it is particularly suitable for data analysis and model performance assessment.

3.3 Loss Function Optimization

For measuring the divergence between two probability distributions, Kullback-Leibler (KL) Divergence is used in the models' architecture.

$$D_{KL}(\mathcal{Q}||\mathcal{P}) = \sum_{i} \mathcal{Q}(i) \log \frac{\mathcal{Q}(i)}{\mathcal{P}(i)}$$

$\mathcal{Q}$ and $\mathcal{P}$ are defined as, respectively, $\mathcal{Q}(i)$ is the probability of event i given distribution $\mathcal{Q}(i)$, and $\mathcal{P}(i)$ is the probability with respect to the distribution $\mathcal{P}$ of that very same event.

In the dissertation, KL Divergence is applied in terms of Conditional Variational Autoencoder models. The encoder network of the CVAE outputs a pair of vectors representing the mean μ and the logarithm of the variance $log(\sigma^2)$ of a Gaussian distribution. These represent the approximate posterior distribution $q(z|x)$, where z is the latent variable and x is the input data.

The KL Divergence is used to measure the degree by which this approximate posterior distribution $q(z|x)$ differs from the prior distribution $p(z)$, which is usually set to a standard normal distribution $N(0, I)$.

The KL Divergence for a single latent variable z which follows Gaussian Distribution is computed as:

$$D_{KL}\big(q(z|x) \parallel p(z)\big) = \frac{1}{2} \sum_{i=1}^{d} (\sigma_i^2 + \mu_i^2 - 1 - log(\sigma_i^2))$$

where:

- d is the dimensionality of the latent space.
- μ_i and σ_i^2 are the mean and variance for the i-th dimension of the latent variable.

This is a part of the loss function of CVAEs. The total loss functions that CVAEs strive to minimize is the sum of the reconstruction loss and the KL Divergence:

$$Total\ Loss = Reconstruction\ Loss + \beta \times D_{KL}(q(z|x) \parallel p(z))$$

where:

- **Reconstruction Loss:** Measures the reconstruction capability of the decoder regarding the input data x and the latent representation z.

The Mean Squared Error (MSE) is very sensitive to large errors because of the squaring of error terms; hence, it is preferred in tasks such as hand pose estimation, where large errors lead to unrealistic outcomes. Its differentiability makes it convenient for training

models, especially with gradient descent optimization. However, the Mean Absolute Error (MAE) is robust to outliers and correspondingly provides a more balanced evaluation of errors. It is generally more realistic regarding the average size of the error magnitude. The MAE is further interpretable since it gives an average of errors in the same units as the observations themselves.

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

where:

- n is the number of data points.
- y_i is the actual value.
- $\hat{y}_i$ is the predicted value

$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$

where:

- n is the number of data points.
- y_i is the actual value.
- $\hat{y}_i$ is the predicted value.

Using the combination of these methods ensures that models, such as those predicting hand poses for robotics, are both accurate and robust, providing reliable and practical results.

3.4 Data Selection and Analysis

The dataset is taken from Hampali et al. (2020). HO-3D_v3 is specifically designed for 3D pose annotation for interacting hand-object. The dataset contains 103,462 RGB images along with their 3D hand–object poses and corresponding depth maps. The dataset contains 10 human subjects, 3 females, and 7 men; further, it contains 10 objects selected from the YCB dataset. MANO model is selected for the estimation of hand pose.

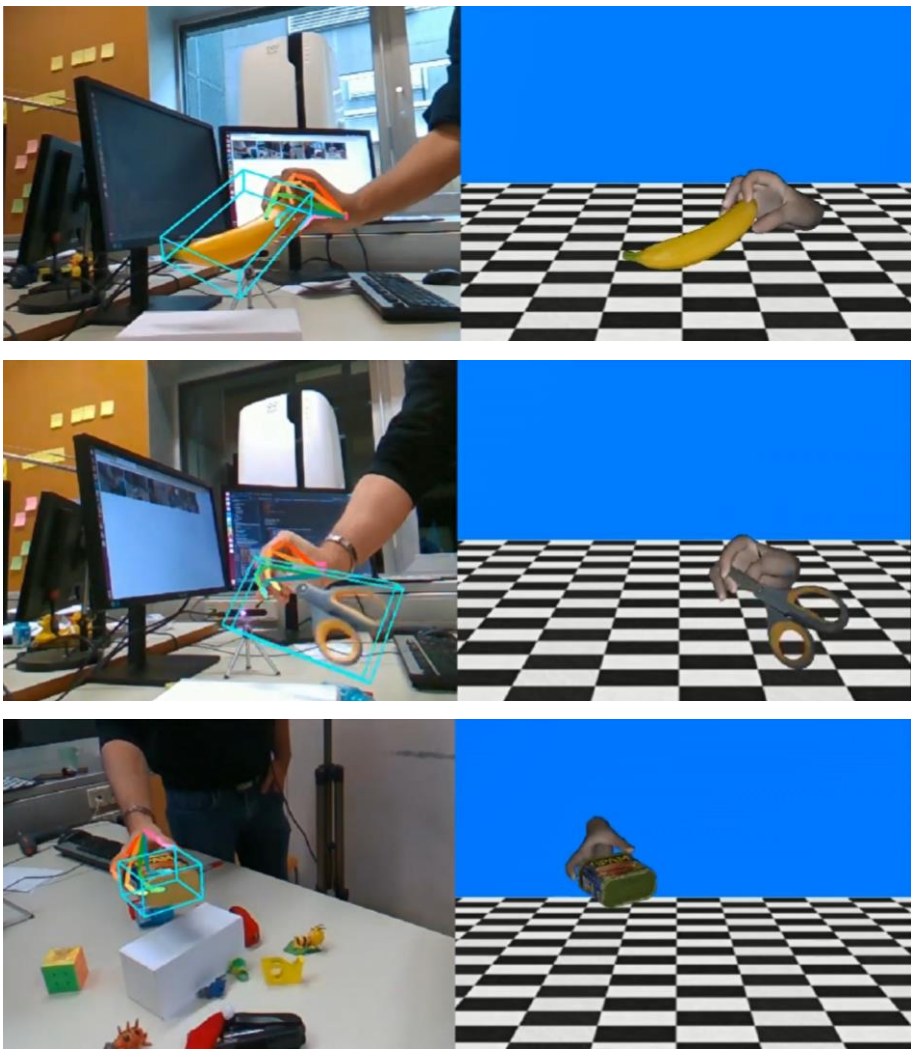


Figure 2: Samples of hand pose annotated based on objects from HO-3D_v3, captured from the video that is shared on (Hand-Object 3D Pose Annotation, 2020)

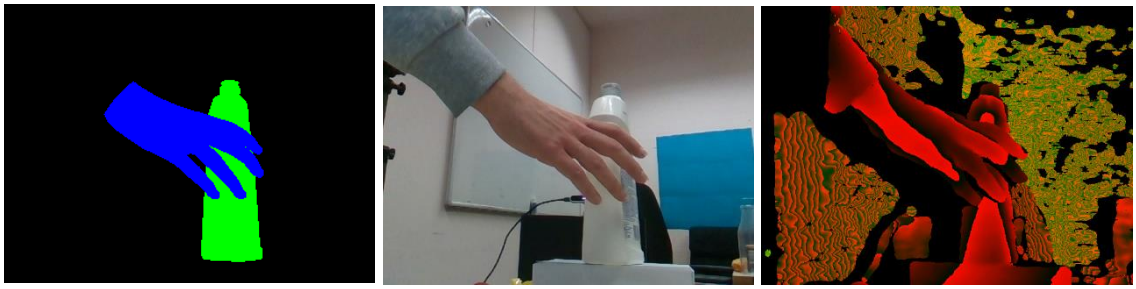


Figure 3: Samples of hand-object relation from the HO-3D_v3 are downloaded from the (Hand-Object 3D Pose Annotation, 2020) *website. The images are the first frame of the hand pose related to object '*02_bleach_cleanser*' based on the YCB dataset. These are from the 'seg', 'rgb', and 'depth' folders respectively, which are in the 'ABF10' subfolder of the 'train' folder in the HO-3D_v3 dataset. The rest YCB folder names in respect of HO-3D_v3 dataset is in Table 4.*

The YCB Dataset (Calli et al., 2015) is a benchmark collection of objects for robotic manipulation, varying in shapes, sizes, and textures. Its detailed 3D and high-resolution scans underline a robust framework for comparing new research techniques in robotic manipulation with the existing test performances on uniform objects and protocols.

The MANO (Model-based Anthropomorphic Hand Optimization) (MANO, 2020) hand model is a 3-D model that has been used in computer vision and graphics to better represent the human hand realistically and animate it. It has great value for researchers and developers in industries like virtual reality, gaming, and animation that demand natural hand motion (Romero et al., 2017).

3.4.1 Dataset Ethics

It is permitted to be freely used by all researchers and the researchers from educational institutions and/or public-private research labs in their non-commercial research. Redistribution, modification, or commercial use of the dataset is not allowed unless explicitly permitted by the authors.

3.4.2 Detailed Explanation of Annotations in HO-3D_v3 Dataset

Annotations contain a rich source of information that can be used for the material concerning 3D pose estimation and analysis. For this study, hand pose attributes such as 'handPose', 'handTrans', and 'handJoints3D' and object-related information, including 'objTrans', 'objRot', 'objCorners3D', and 'objName' are selected as most important data fields to training generative models.

The details of what each key in the annotation dictionary contains are explained in Table 3 below.

No.	Key name	Format	Description
1	***objTrans**	[x, y, z]	A 3x1 vector represents the translation of the object in 3D space and is the object relative to the camera.
2	***objRot**	[rx, ry, rz]	A 3x1 vector representing the rotation of the object using the axis-angle representation, the most common method of describing rotations in 3D.
3	***handPose**	[r1x, r1y, r1z, ..., r16x, r16y, r16z]	A 48x1 vector representing the 3D rotation of 16 hand joints, including the root joint, in axis-angle representation. It is following the MANO model.
4	***handTrans**	[x, y, z]	A 3x1 vector representing the translation of the hand in 3D space and with respect to the camera.
5	**handBeta**	[b1, b2, ..., b10]	A 10x1 vector representing the MANO hand shape parameters. It can be utilized to render dissimilar shapes of the hand using the MANO model.
6	***handJoints3D**	[[j1x, j1y, j1z], ..., [j21x, j21y, j21z]]	A 21x3 matrix representing the 3D locations of 21 hand joints, is crucial for the estimation and analysis of poses.

7	***objCorners3D**	[[c1x, c1y, c1z], ..., [c8x, c8y, c8z]]	An 8x3 matrix representing the 3D coordinates of the object's bounding box corners after applying transformations (final pose)
8	**objCorners3DRest**	[[c1x, c1y, c1z], ..., [c8x, c8y, c8z]]	An 8x3 matrix representing the 3D coordinates of the object's bounding box corners before applying any transformations
9	***objName**	String	The name of the object that is being interacted with within the scene as specified in the YCB dataset
10	**objLabel**	String	The label of the object as given in the YCB dataset which is useful in classification and investigation.
11	**camMat**	3x3 matrix	Intrinsic camera parameters which are necessary to project 3D points onto 2D image coordinates.
12	**handVertContact**	[c1, c2, ..., c778] (boolean values)	A 778D boolean vector indicates if each MANO vertex is contacted within 4 mm of an object.
13	**handVertDist**	[d1, d2, ..., d778]	A 778D float vector representing the distances of MANO vertices to the object surface
14	**handVertIntersec**	[i1, i2, ..., i778] (boolean values)	A 778D boolean vector indicating whether each MANO vertex is inside the object surface.
15	**handVertObjSurfProj**	[[p1x, p1y, p1z], ..., [p778x, p778y, p778z]]	A 778x3 matrix representing the projection of MANO vertices on the object surface.

Table 3: Annotations details in HO-3D_v3 Dataset. Selected data filed is marked by an asterisk (*) beside the key names

YCB Dataset ('model' folder) - object name	**HO-3D_v3 ('train' folder)**		**YCB Dataset ('model' folder) - object name**	**HO-3D_v3 ('train' folder)**
02_bleach_cleanser	ABF10		003_cracker_box	MC1
	ABF11			MC2
	ABF12			MC4
	ABF13			MC5
	ABF14			MC6
	SB10		035_power_drill	MDF10
	SB12			MDF11
	SB14			MDF12

011_banana	BB10			MDF13
	BB11			MDF14
	BB12			ND2
	BB13		004_suggar_box	ShSu10
	BB14			ShSu11
	SiBF10			ShSu12
	SiBF11			ShSu13
	SiBF12			ShSu14
	SiBF13			SiS1
	SiBF14			SS1
010_potted_meat_can	GPMF10			SS2
	GPMF11			SS3
	GPMF12		006_mustard_bottle	SM2
	GPMF13			SM3
	GPMF14			SM4
037_scissors	GSF10			SM5
	GSF11		025_mug	SMu1
	GSF12			SMu40
	GSF13			SMu2
	GSF14			

Table 4: Object folder names and its corresponding folder name in the 'train' folder of HO-3D_v3 dataset.

Some of the images in the train and evaluation folders are not annotated. Such images all have their respective fields marked as None. Only the images listed within the 'train.txt' and 'evaluation.txt' files are correctly annotated and may thus be used for training and evaluation exercises.

3.4.4 Data Analysis Techniques

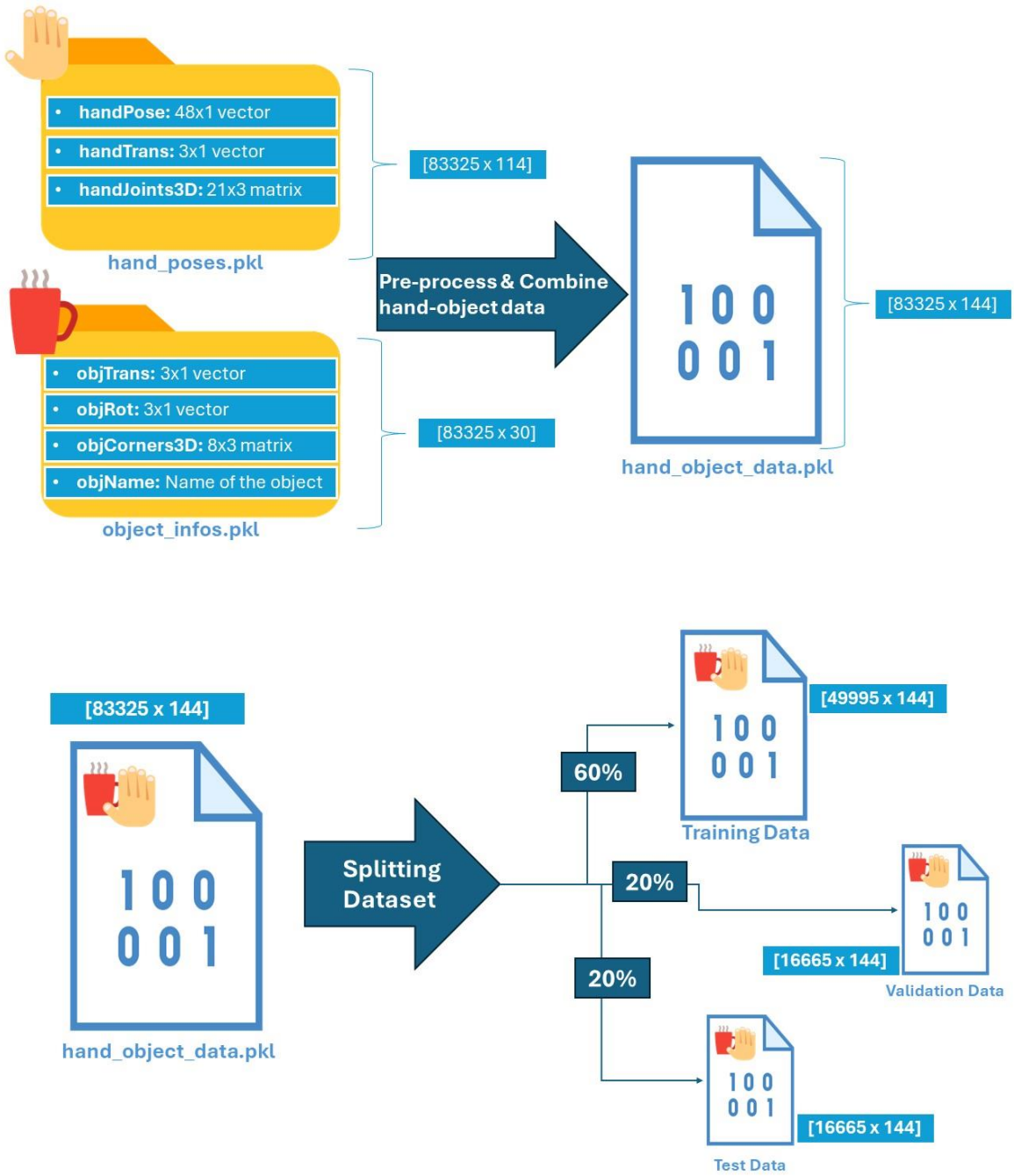


Figure 4: Summary schematic of data preprocessing and preparation (data extraction, normalization, and splitting) for training models. The file "hand_object_data" contains the training, testing, and validation data regarding hand and object features alongside their respective identifiers.

Data has been pre-processed in a few steps, such as data feature extraction, normalization, and data splitting before finally being fed into the CVAE models (Figure 4, 5). These are done as follows:

I. **Feature Extraction:** The first step involves the extraction of relevant features from the dataset. Based on the 'meta' file for every frame of the video, hand pose attributes such as 'handPose', 'handTrans', and 'handJoints3D' and object-related

information, including ‘objTrans’, ‘objRot’, ‘objCorners3D’, and ‘objName’ are selected. This data is saved into separate pickle files: "hand_poses.pkl" and "object_infos.pkl". The meta file is in the 'train' folder, part of which contains up to 9 objects (Table 4) including grasping hand pose information for each.

II. **Normalization:** After that, the preprocessing for uniform data normalization is done using class instances of StandardScaler coming from module sklearn.preprocessing. The output of the normalized data is saved into the "scalers.pkl" file. (Table 6) shows key names and descriptions of the file.

III. **Data Splitting:** The ‘train_test_split‘ function from the StandardScaler library is applied in split processing. The data is divided into train-validation-test sets which means that models can be trained using one partition of the data, another partition is used to validate, and then an independent subset tests how well the model generalizes the information (Figure 4). The output is saved in "hand_object_data" file.

All Frames		**83325**	
Hand Information	handPose	48	48+3+63=114
	handTrans	3	
	handJoints3D	21x3=63	
Object Information	objectTrans	3	3+3+24=30
	objectRot	3	
	objectCorners3D	8x3=24	
Hand Object Data		**114+30=144**	

Table 5: Following feature extraction from all frames in the meta files within the 'train' folder, 83,325 dimensions are identified, representing 83,325 frames of hand-object data. The hand pose data has dimensions of 83,325x114, while the object information is 83,325x30. After concatenating these, the final result is a dataset of 83,325x144 dimensions.

Using current data analysis techniques, the generative models are trained with the very best quality and seamlessly organized data, therefore raising its potential to give more realistic hand poses based on the object affordances. A flow diagram in (Figures 4, and 5) shows how data selection and preprocessing have been done. In addition, Table 5 provides details on the dimensionality of the extracted data prepared for training models.

Key Name	**Description**
scaler_hand_joints	Reducing scaling variability minimizes its impact on hand joint position estimates, ensuring unbiased machine learning model training.
scaler_obj_corners	Normalizing object corner data prevents scale differences across dimensions from negatively impacting the model's learning process.

scaler_hand_pose	Scaler normalizes hand pose data to zero mean and unit standard deviation. It can improve training models' performances.
scaler_hand_trans	This scaler normalizes hand translation data to ensure all features are equally significant, preventing domination by wider-ranged features.
scaler_obj_trans	The scaler normalizes object translation data, ensuring consistent input scales, which improves the learning process for object positioning.
scaler_obj_rot	This scaler normalizes object rotation data, ensuring consistent values and preventing issues during model learning from rotations.

Table 6: Details of scalers.pkl file and their explanation.

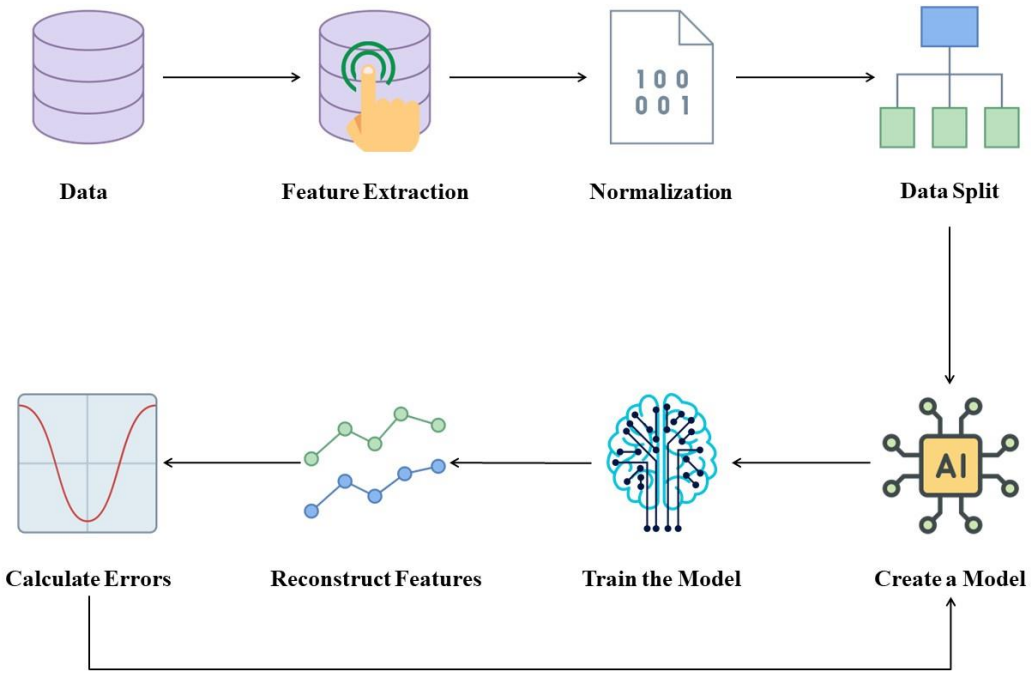


Figure 5: This flowchart depicts the end-to-end process of machine learning model development, focusing on feature extraction and model training. This process is iterative, enabling continuous refinement of the model.

3.4.5 Visualization of the Dataset

Figure 6 shows that information regarding random objects from the first frame of the dataset HO-3D_v3 is applied to 3D points information of objects in the YCB dataset. This is done to further understand how the combination of the YCB dataset and object rotation and translation works. Similarly, the information of the 3D hand joints poses with its rotation and translation for different objects and the first frame of them is plotted (Figure 7).

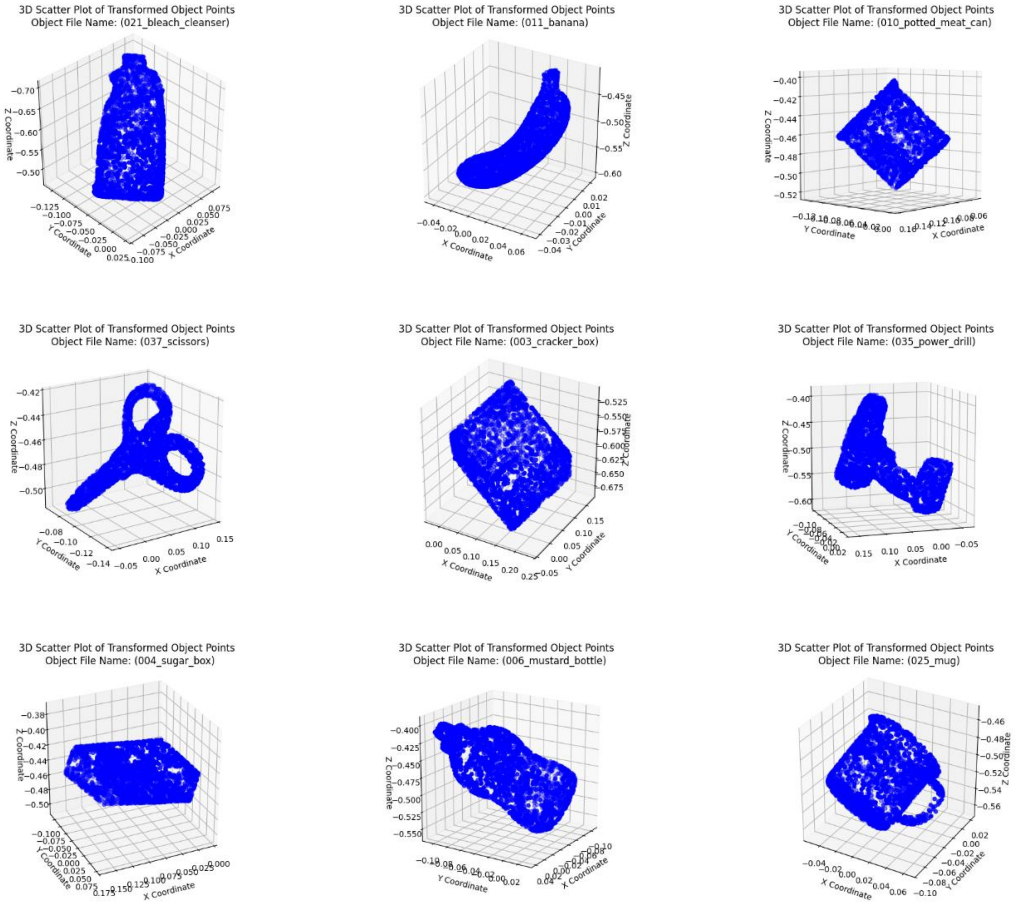
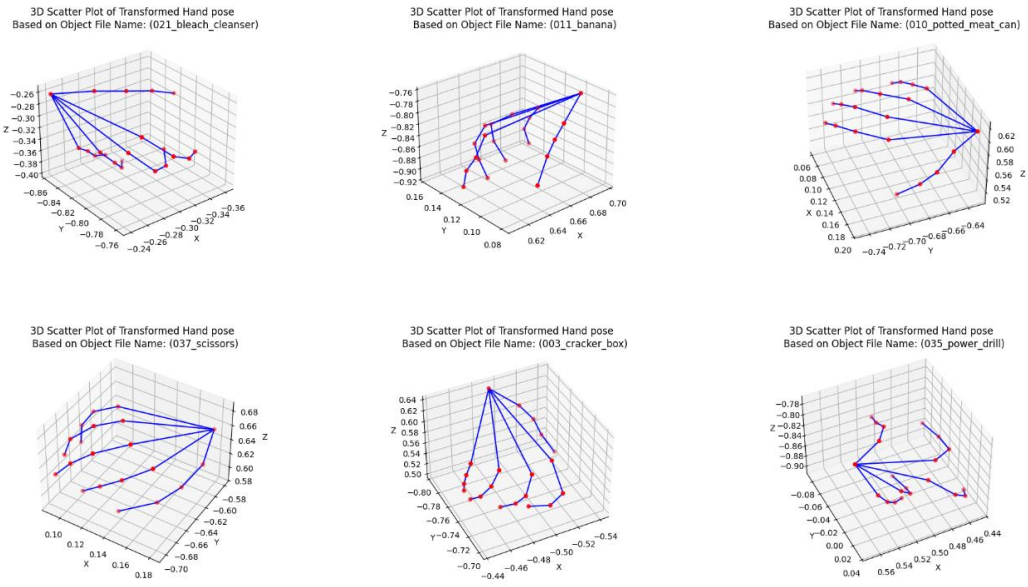


Figure 6: 3D objects (including: 'bleach cleanser', 'banana', 'potted meat can', 'scissors', 'cracker box', 'power drill', 'sugar box', 'mustard bottle' and 'mug') in YCB dataset are used in HO-3D_v3 dataset. All these objects are plotted from the first frames of each video in HO-3d_v3 Dataset



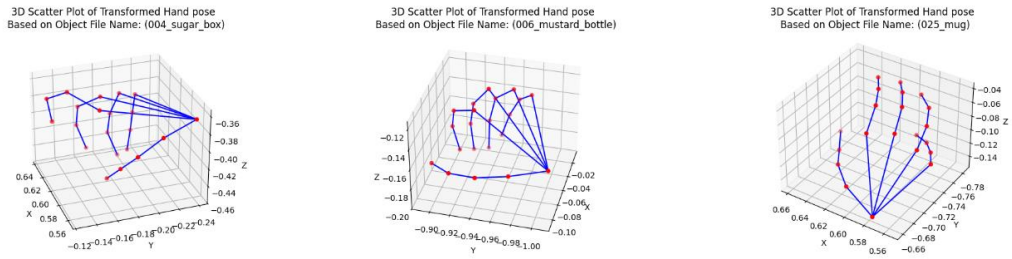


Figure 7: 3D annotated hand joints in the HO-3D_v3 dataset based on the 9 objects in the YCB dataset. These hand poses correspond to each object plotted in Figure 6 with the same frames in the HO-3D_v3 dataset

3.5 Implementation and Architecture

3.5.1 Implementation

The implementation phase takes this research from the initial data extraction, preprocessing, and preparation through finding a sufficient model and designing and running the model architecture to training and final evaluation to achieve human-like hand pose performance in producing grasping in response to object affordances.

For this aim, the dissertation initiates the development of the basic version of the CVAE model by acquiring minimal layers in both sections of the architecture: encoder and decoder. The development of this model is incremental with the addition of more layers as an effort to seek better performance.

3.5.2 Model Architecture

Initially, a three-member set of different Conditional Variational Autoencoder models, CVAE_01, CVAE_02, and CVAE_03, have been built. After training these three models, the study considers the CVAE model as the candidate model for improvement. Therefore, two new variations, named CVAE_02_1 and CVAE_02_2, have been created by adding more layers to CVAE_02. In the following, the conditional input is removed from the encoder in CVAE_02_3 to simplify the model to maintain its performance and only the decoder uses the conditional input.

The different designs of the CVAE models that have been used throughout this study are described here in further detail:

3.5.2.1 CVAE_01

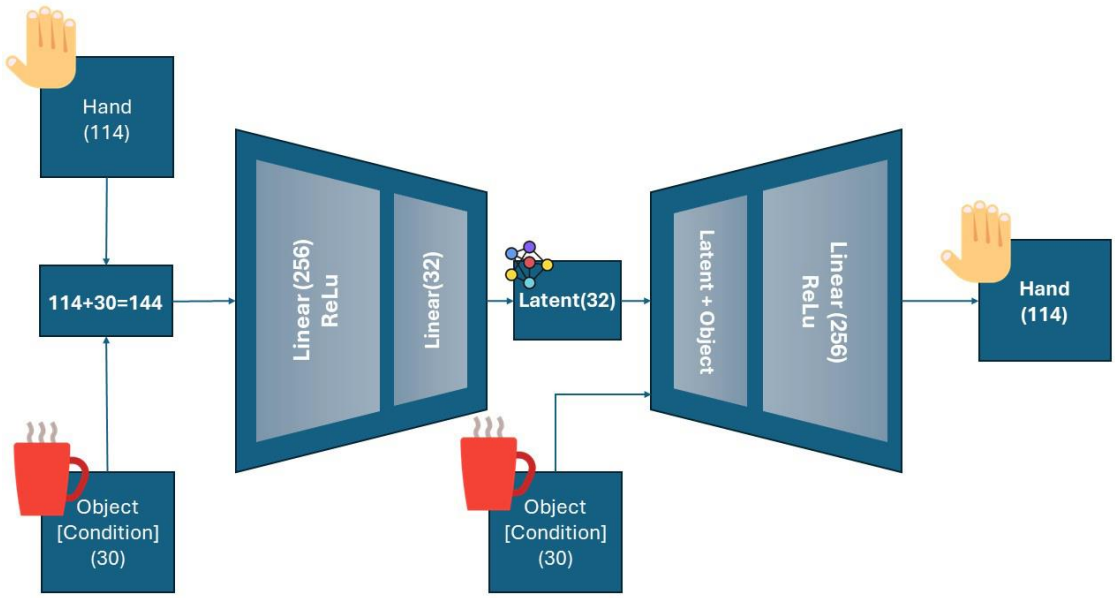


Figure 8: The architecture of Conditional Variational Autoencoder Model CVAE_01, in total the model has 6 layers (4 Linear layers and 2 ReLU activation functions).

I. **Encoder Network Architecture:**

The encoder network takes the concatenated input (hand information as input data + object information as condition data) and passes it through a few fully connected layers. The output is a vector produced by the encoder, separated into two parts: the mean, and the log-variance, both corresponding to the latent Gaussian distribution.

II. **Decoder Network Architecture:**

It would take in the sampled latent vector, z, and conditioning data as inputs, combine it into one component, and pass the output through several fully connected layers for the reconstruction of the original input data.

3.5.2.2 CVAE_02

CVAE_02 provides higher representational capacity for complicated relationships, not only because of its depth but also because of larger layer sizes. This may result in improved performance for more challenging tasks, though at a higher computational requirement and with a risk of overfitting unless regularized properly.

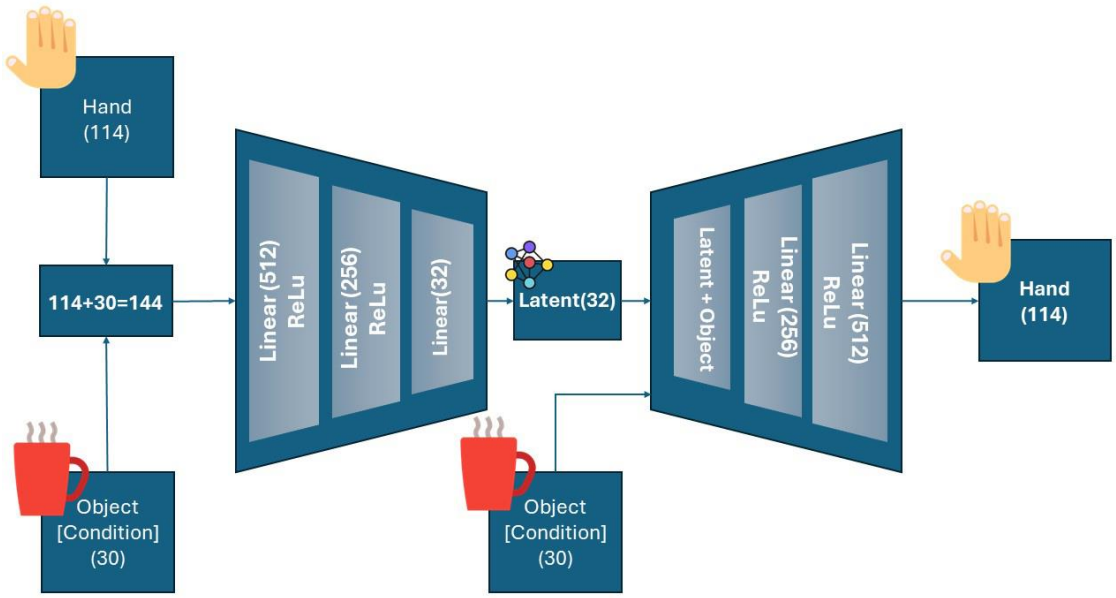


Figure 9: The architecture of Conditional Variational Autoencoder Model CVAE_02, the model has a total of 10 layers.

III. Encoder Network Architecture:

The encoder is more complex with three layers.

i. **First Layer:** like the previous model, the input and conditioning are concatenated, but then, they are sent to a linear layer of 512 units and put through a ReLU activation.

ii. **Second Layer:** A 256-unit linear layer, activated by ReLU activation, between the input and output layers.

iii. **Output Layer:** Thus, it outputs a mean and log-variance of size latent dimension*2 like CVAE_01.

IV. Decoder Network Architecture:

Similarly, the decoder contains three layers to increase complexity.

i. **First Layer:** After the concatenation of latent space (z) with the condition, it is fed through a 256-unit linear layer with ReLU activation.

ii. **Second Layer:** This is then followed by another dense linearity layer with 512 units before the final output layer with ReLU activation.

iii. **Output Layer:** An additional linear layer with input dimension units are used for the final output.

3.5.2.3 CVAE_03

The CVAE_03 model introduces the use of dropout and batch normalization to make the model more resistant to overfitting specifically when the dataset might be noisy or when generalization is a concern.

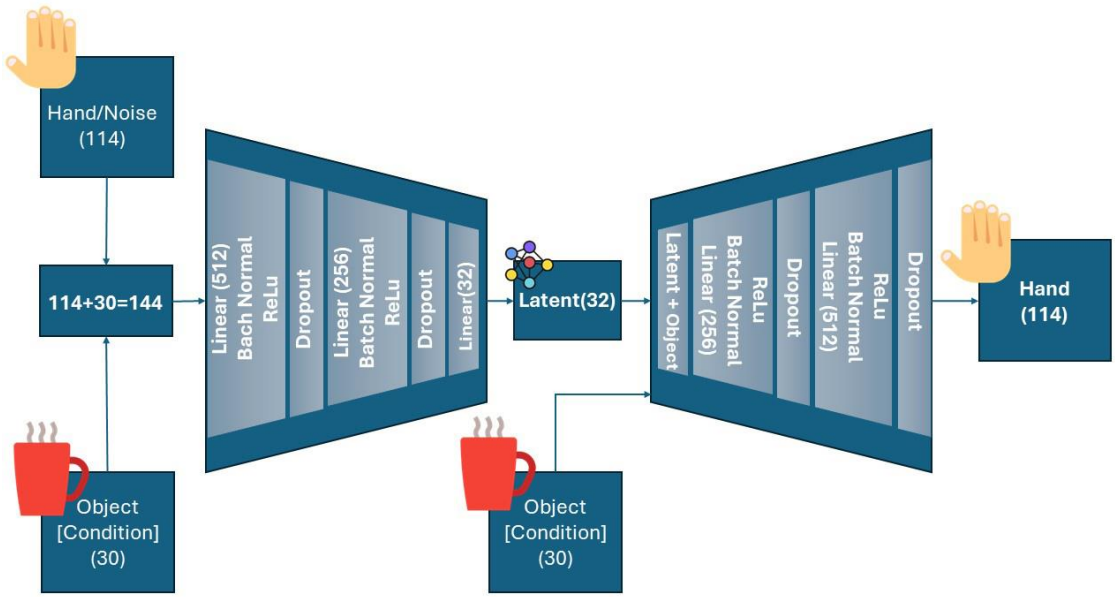


Figure 10: The architecture of Conditional Variational Autoencoder Model CVAE_03, the model has 18 layers in total (9 in the encoder and 9 in the decoder).

I. Batch Normalization (BatchNorm1d):

Includes batch normalization after each linear layer in both the encoder and decoder module. Batch normalization gets the input normalization in every layer close to a mean of 0 with a standard deviation of 1. It makes the training faster and more stable without being troubled by internal covariate shift reduction.

II. Dropout:

This architecture introduces dropout with a probability of 0.5 after every ReLU activation in both the encoder and decoder. It provides a regularization technique that randomly sets a fraction of input units to zero during training to avoid the overdependency of the model on any one neuron, hence reducing overfitting.

3.5.2.4 CVAE_02_1

CVAE_02_1 is a more complex and deeper model with 4 layers in both the encoder and decoder consequently, it has more representational power at the cost of increased computational requirements.

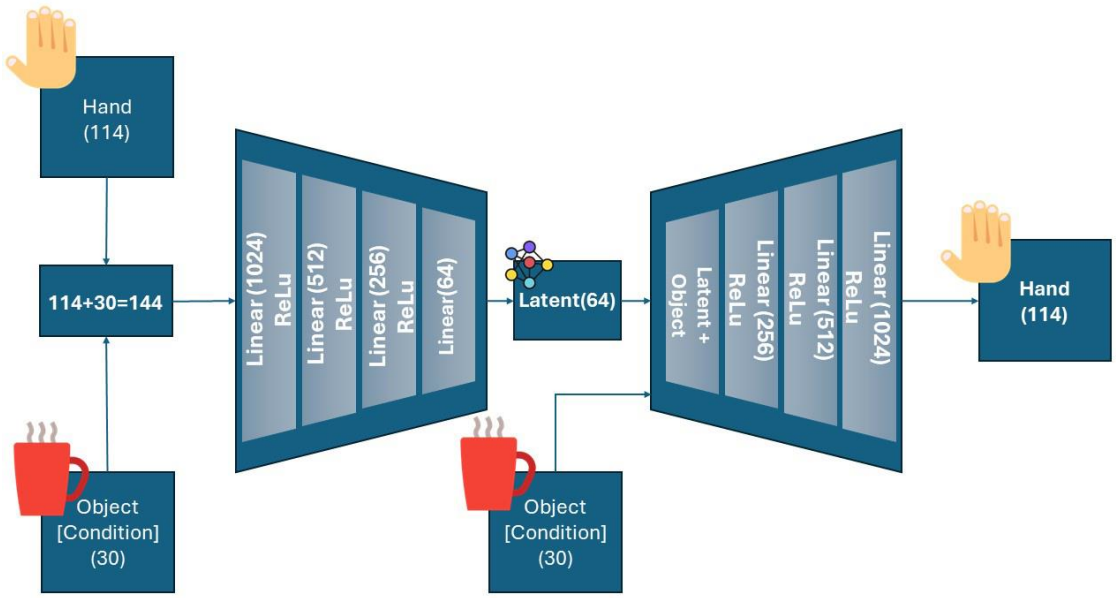


Figure 11: The architecture of Conditional Variational Autoencoder Model CVAE_02_1, the model has a total of 14 layers.

I. Encoder Network Architecture:

The encoder consists of 4 fully connected layers:

i. **Layer 1**: input dimension + condition dimension → 1024 units

ii. **Layer 2:** 1024 units → 512 units

iii. **Layer 3:** 512 units → 256 units

iv. **Layer 4:** 256 units → latent dimension * 2 units (for mean and log-variance)

II. Decoder Network Architecture:

The decoder has 4 fully connected layers:

i. **Layer 1:** latent dimension + condition dimension → 256 units

ii. **Layer 2:** 256 units → 512 units

iii. **Layer 3:** 512 units → 1024 units

iv. **Layer 4**: 1024 units → input dimension units

3.5.2.5 CVAE_02_2

CVAE_02_2 is deeper than CVAE_02_1 directly correlated with additional layers that introduce more parametric information. These extra layers may increase the model's

capacity to learn complex patterns in the data, but they also raise the probability of overfitting while increasing the computational resources.

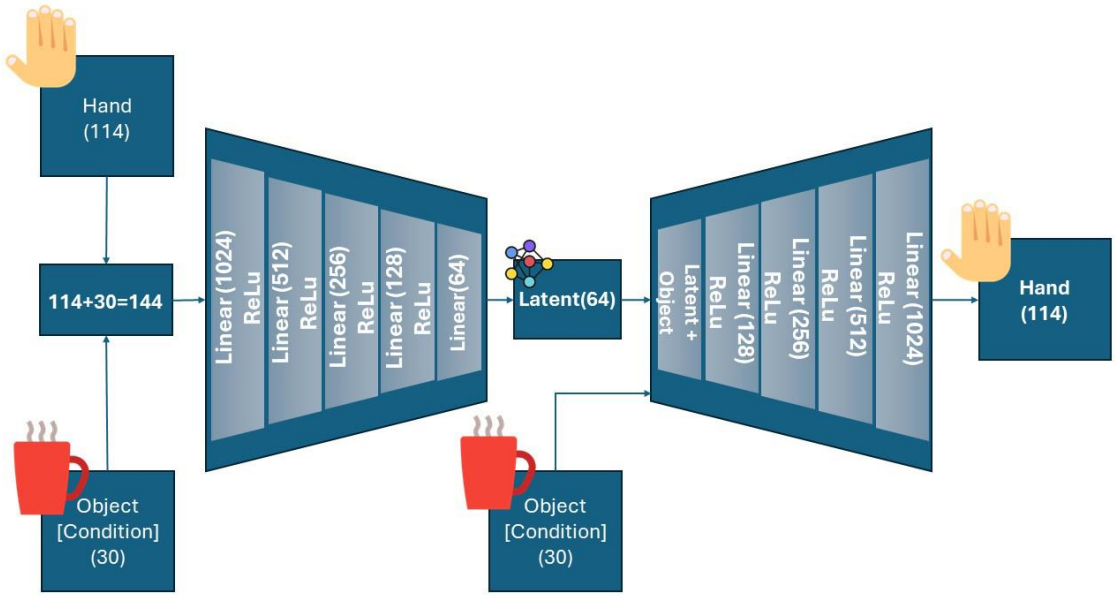


Figure 12: The architecture of Conditional Variational Autoencoder Model CVAE_02_2, the entire model has a total of 18 layers.

I. Encoder Network Architecture:

The encoder is basically the same as in CVAE_02_1 but with an additional layer, so now it has five linear layers of sizes 1024, 512, 256, and 128, which output to latent dimension * 2, and the sequence goes as follows:

Hand pose information + Object information (condition)→ 1024 → 512 → 256 → 128 → latent dimension * 2

II. Decoder Network Architecture:

Like the encoder, the decoder comprises five linear layers, with sizes 128, 256, 512, and 1024, and finally, it generates an output of input dimension. The sequence of layers is:

Latent dimension + Object information (condition) → 128 → 256 → 512 → 1024 → input dimension (Hand pose information)

3.5.2.6 CVAE_02_3

This structure keeps the condition out of the encoder, making the latent space more general. Instead, it relies on the decoder to apply the condition. By removing the condition from the encoder, the latent space is more general. It may make the model more flexible

and better able to handle a variety of conditions. In such a case, the decoder must compensate by adopting the general latent space to the specific condition, which might be more challenging and could result in less specific reconstructions.

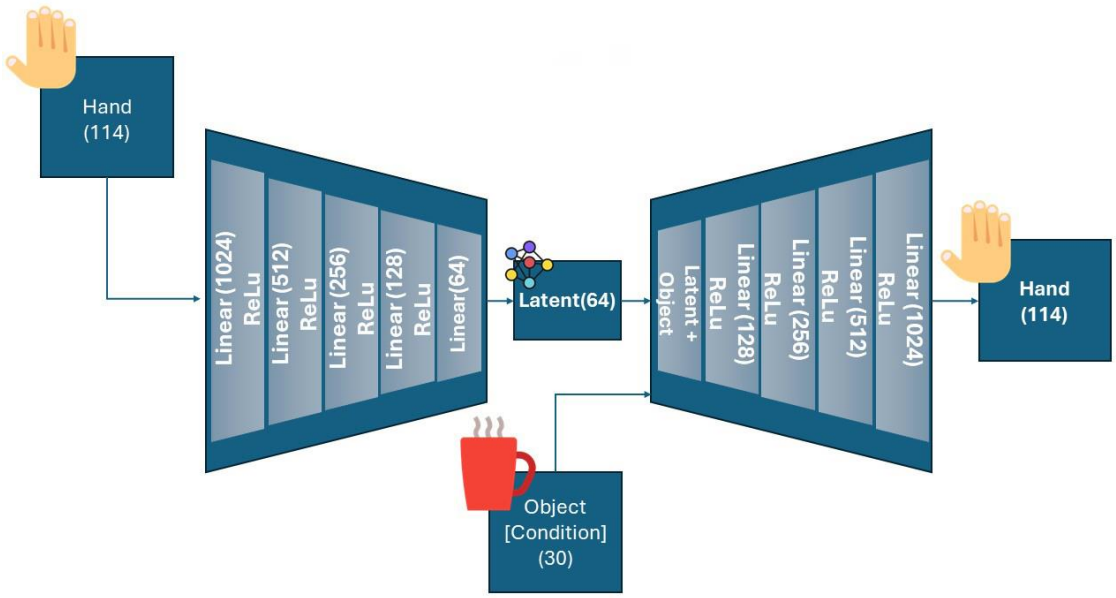


Figure 13: The architecture of Conditional Variational Autoencoder Model CVAE_02_3, the entire model has a total of 18 layers.

I. Encoder Network Architecture:

i. **No Conditioning Information in Encoder:** It only receives the input (Hand pose information) and generates the latent space without any direct influence of the condition.

The idea behind this architecture is that the latent space of this model is condition-agnostic. The conditioning information passes only through the decoding phase, so, the decoder is responsible for tailoring the latent representation to the specific condition.

II. Latent Space Representation:

The latent space is more general since the condition does not affect the encoding.

III. Decoder Network Architecture:

Like the previous model, the decoder takes both the latent variable and the object information (condition) as inputs. It shows that the reconstruction is always conditioned on the provided condition, regardless of how the latent space is generated.

4.0 RESULTS AND EVALUATION

4.1 Training and Evaluation

Training the CVAE model is pretty regular, following a variational autoencoder approach with conditional inputs. In each epoch, it goes through batches of data concerning hand poses and objects. Moving through the forward pass, it encodes the input data to a latent space and reparametrizes the latent representation. Then, it decodes the latent variables together with the conditional data to reconstruct the original input. It consists of two terms: the reconstruction loss-mean squared error between original and reconstructed data, and a regularization term, the KL divergence that keeps the latent distribution close to the normal distribution. The optimizer does the backpropagation to update the model weights with the goal of minimizing the overall combined loss. In addition to the training loss, it evaluates reconstruction loss and specific pose, joints, and translation errors on the validation set to monitor the model's performance over epochs.

The KL Divergence term in the loss function plays a critical role in balancing the balance between the accuracy of data reconstruction and the regularization of a smooth latent space. The minimization of the divergence term prevents the overfitting of the training data and also includes the proper behavior of the latent variables z, which improves generalization and makes the interpolations of the latent space more interpretable.

The following are training and evaluation processes which are done in four phases in this study. Table 8 shows the summary of the phases with details of the models' layers and their training.

Model Name	Epoch	Latent Space	Learning Rate	Encoder Layers	Decoder Layers	Description
First Phase						
CVAE_01	50	32	0.001	3	3	144→256→32→256→114
CVAE_02*	50	32	0.001	5	5	144→512→256→32→114
CVAE_03	50	32	0.001	9	9	Includes batch normalization and Droupout
CVAE_03	50	32	0.001	9	9	Adds Noise in Data
Second Phase						
CVAE_02	50	32	0.001	5	5	144→256→32→256→114
CVAE_02	100	64	0.0005	5	5	144→256→64→256→114
CVAE_02*	200	64	0.0005	5	5	Increasing epochs
Third Phase						
CVAE_02	200	64	0.0005	5	5	144→256→64→256→114

CVAE_02_1	200	64	0.0005	7	7	144→1024→512→256→64→256→512→1024→114
CVAE_02_2*	200	64	0.0005	9	9	144→1024→512→256→128→64→128→256→512→1024→114
Fourth Phase						
CVAE_02_1	200	64	0.0005	7	7	144→1024→512→256→64→256→512→1024→114
CVAE_02_2	200	64	0.0005	9	9	144→1024→512→256→128→64→128→256→512→1024→114
CVAE_02_3*	200	64	0.0005	9	9	Remove the object information (Condition) From the Input to the Encoder

Table 8: This provides an overview of the training and evaluation phases, including detailed information about the models' layers and their training hyperparameters. The model identified for potential improvement for each phase is marked with an asterisk (*)

4.1.1 First Phase: CVAE_01, CVAE_02, and CVAE_03

The first three models, CVAE_01, CVAE_02, and CVAE_03, have been trained by using 32 as a latent space dimension, with a learning rate of 0.001, and for 50 epochs. In the case of CVAE_03, its training starts with information regarding only the hand and the object as condition, and then random noise has been added to the hand information. Finally, these models' weights have been saved in four pickle files such as, `cvae_01_weights.pkl`, `cvae_02_weights.pkl`, `cvae_03_weights.pkl`, and `cvae_03Noise_weights.pth`.

For the assessment of performance, several loss functions and error metrics are calculated for the models. These included the Train Loss, Validation Loss, Hand Pose Information Error, Hand Joints Error, Hand Translation Error, and Reconstructed Hand Error. All calculated metrics are stored in corresponding pickle files for each model, ensuring proper tracking and future analysis of the results.

The analysis revealed that the CVAE_02 model has demonstrated superior performance relative to the others, as evidenced by its lowest loss and error across the evaluated metrics. Conversely, the performance of CVAE_03, regardless of whether it is trained using standard hand information or incorporated additional noise, exhibited only slight fluctuations in loss and error measurements. This lack of variability implies that the introduction of noise does not considerably influence the model's effectiveness, thereby affirming CVAE_02 as the most advantageous model among the three assessed (Figure 14).

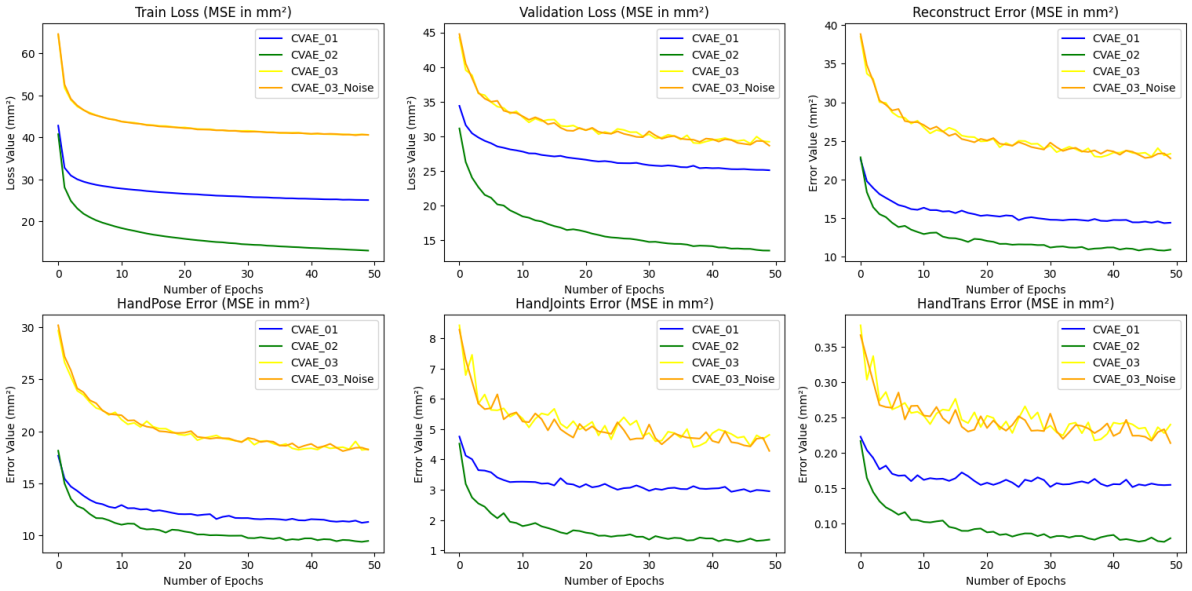


Figure 14: Train and Validation Loss values, Reconstruct, Hand Pose, Hand Joints, and Hand Translation Error information for Comparison between CVAE_01, C VAE_02, and CAVE_03 models. All three models have been trained with a hyperparameter of 50 epochs with a latent space dimension of 32 and learning rate of 0.001. The CAVE_03 is trained twice, first, it is trained with normal hand-object data, then it is trained with normal object information and adding noise values into hand information.

Mean square errors and mean absolute errors between the reconstructed hand pose and ground truth are calculated for every test sample to estimate the quality of the reconstructed hand pose on test data. After that, mean values are calculated and visualized in a chart. From the chart in (Figure 15), it is obvious that the average MSE and MAE for a test dataset considerably decreased while using the CVAE_02 model. That means that CVAE_02 is better at generating more realistic and precise hand poses compared to the rest, hence, being more effective in minimizing errors compared to the others.

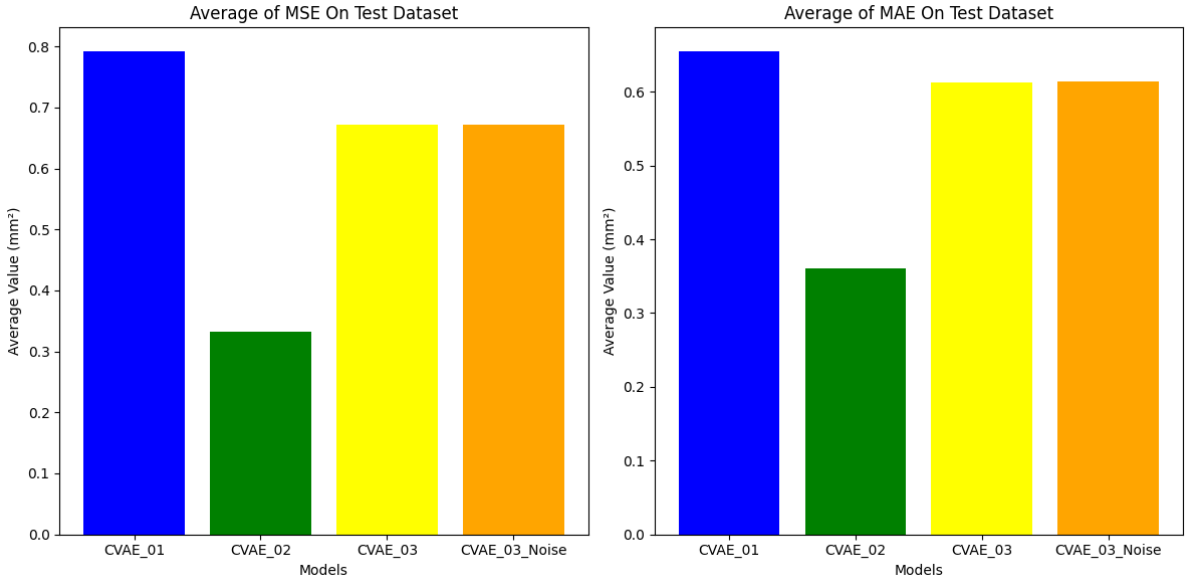


Figure 15: Calculating the average of mean square and absolute error for each trained model on test data. The second model (CAVE_02) trained with normal hand-object had the lowest errors than others.

CVAE_02 has been taken for further iterations and refinements due to such change of several sets of hyperparameters for better effectiveness and, hence, model comparison.

4.1.2 Second Phase: CVAE_02 With Different Hyperparameters

In the next, there are some changes in a hyperparameter and the model CVAE_02 is retrained with these new settings. As a result, increasing the latent space to 64 is the most optimal, while more epochs of training further minimize errors and losses (Figure 16). Since the specific setting of three of the hyperparameters stabilized model CVAE_02, it goes further to fine-tune based on this setting. The average errors confirm the model's superior performance, validating the decision to focus on this configuration for continued optimization.

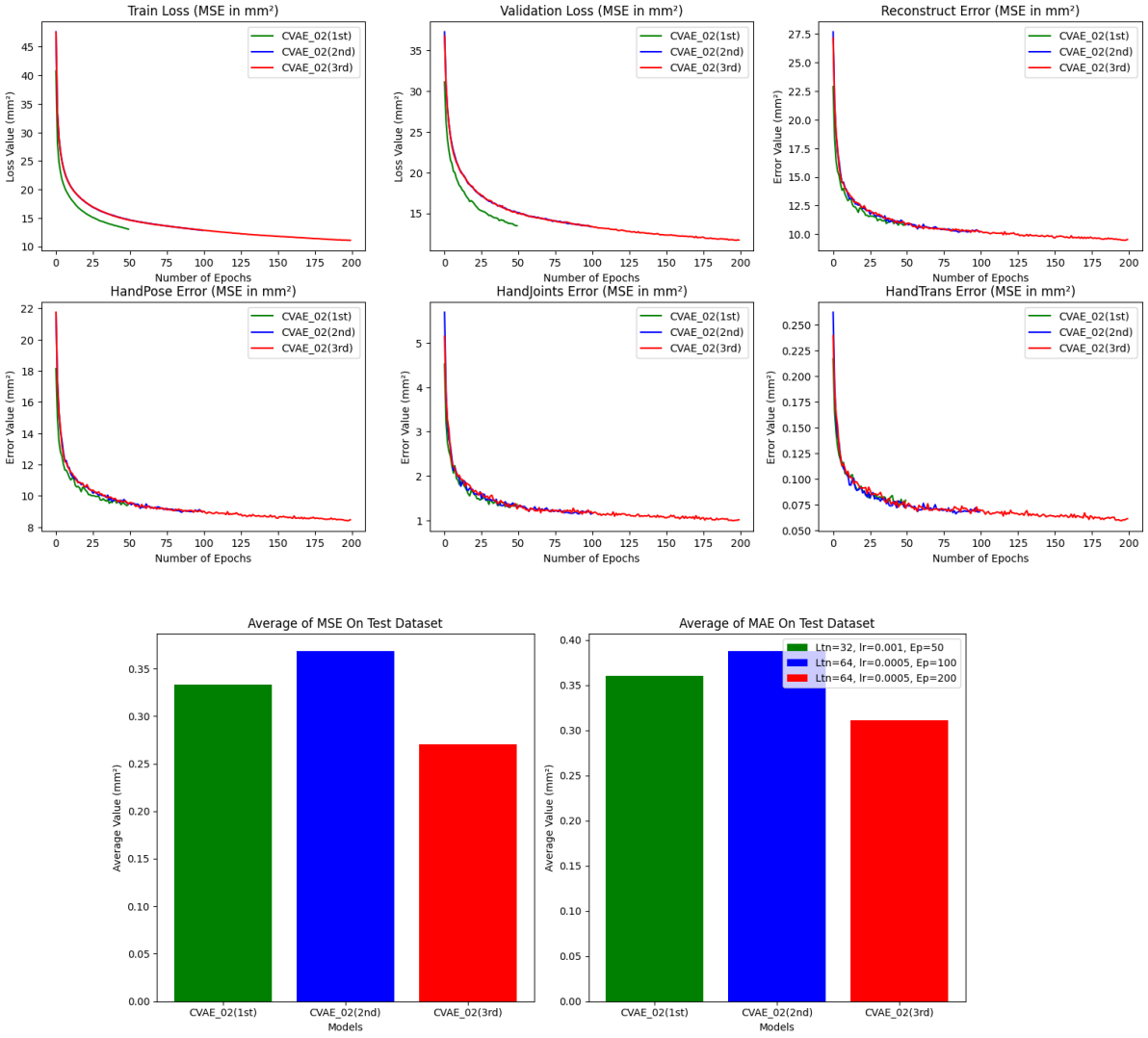


Figure 16: Trying to retrain the second model (CVAE_02) with different hyperparameters and calculating Loss values and errors for each to find out which hyperparameter works better. The hyperparameter that is used for this stage is:
CVAE_02(1st) = { Latent Space=32, Learning Rate=0.001, Number of Epochs=50 }
CVAE_02(2nd) = {Latent Space=64, Learning Rate=0.0005, Number of Epochs=50 }
CVAE_02(3rd) = { Latent Space=64, Learning Rate=0.0005, Number of Epochs=200 }

4.1.3 Third Phase: CVAE_02, CVAE_02_1 and CVAE_02_2

In this third step, two versions of CVAE_02, CVAE_02_1, and CVAE_02_2, are trained under the same hyperparameters, which are stable from the previous phase, in order to compare performances with the third iteration of the CVAE_02 model. Although adding more layers led to significant changes in losses and errors, the outcomes of CVAE_02_1 and CVAE_02_2 are very close to each other. This consistency suggests that the additional layers did not substantially impact the overall performance, indicating that both versions are equally effective under the tested conditions, with no significant advantage over one another (Figure 17).

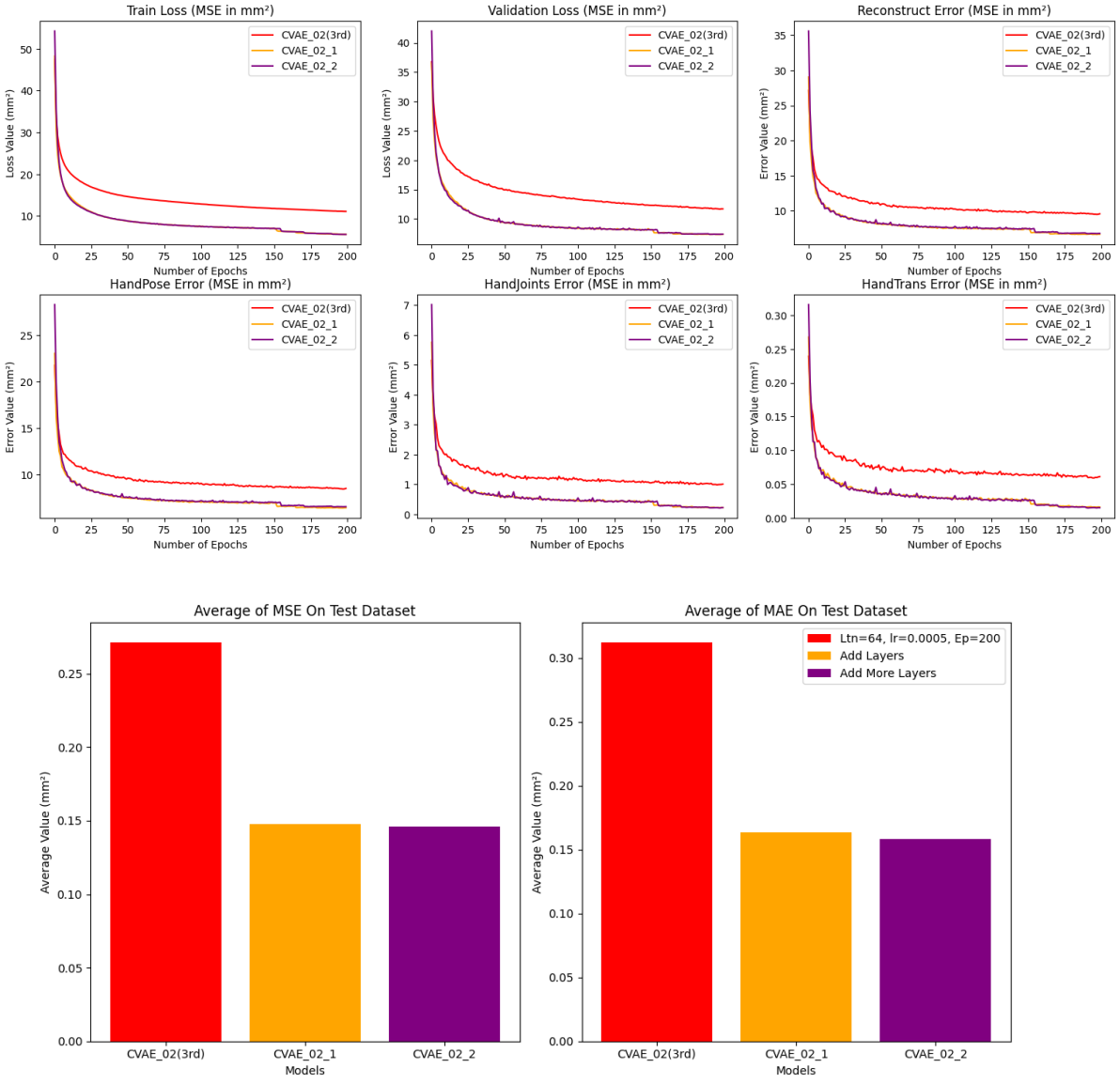


Figure 17: CVAE_02_1: Adding one more layer to the second model (CVAE_02), CVAE_02_2: Adding one more layer to the model CVAE_02_1, Training with the two new architectures of the second model with latest hyperparameters and Losses and Errors are calculated for comparison of the performance between these three models.

The bar chart shows the average of the absolute mean errors obtained for the second instance of the model, CVAE_02_2, which is slightly smaller compared to the base version, indicating a minor performance improvement.

In fact, the second model version has significantly outperformed all the previously compared models, thus justifying the second model version to be more efficient.

4.1.4 Fourth Phase: CVAE_02_3 (Remove Conditional part)

A final experiment has again been trained using this CVAE_02_2 model, but without the conditional part of the decoder, with exactly the same settings for the hyperparameters. The testing results give a more considerable insight into how the condition segment of the decoder is influencing general model response and helps to arrive at the best setup for getting a good hand pose grasping based on object affordances.

As Figure 19 illustrates, all three models (CVAE_02_1, CVAE_02_2, CVAE_02_3) are highly stable in their behavior during the epochs. On average, "CVAE_02_3" seems to be the best model, because it presents lower errors in several key metrics. Oppositely, the "CVAE_02_2" model had a higher error, especially with regard to Reconstruction and HandTrans Error. No model shows high variation in both training and validation loss, which indicates good stability in their training. However, "CVAE_02_2" probably requires more tuning since it has higher error rates, whereas "CVAE_02_3" is the most balanced regarding performance.

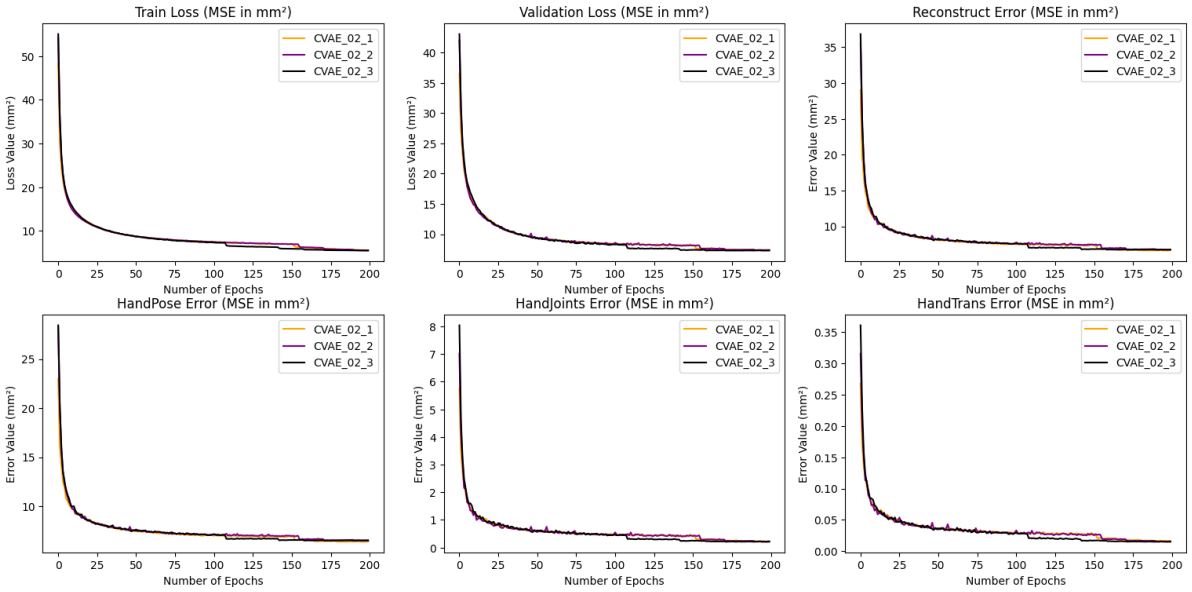


Figure 18: CVAE_02_3: Remove the condition layer from the encoder part of the model CVAE_02_2 and train it with the same hyperparameters. For comparison the performance the Losses and Errors are calculated and showed to see the best performance of the last three models.

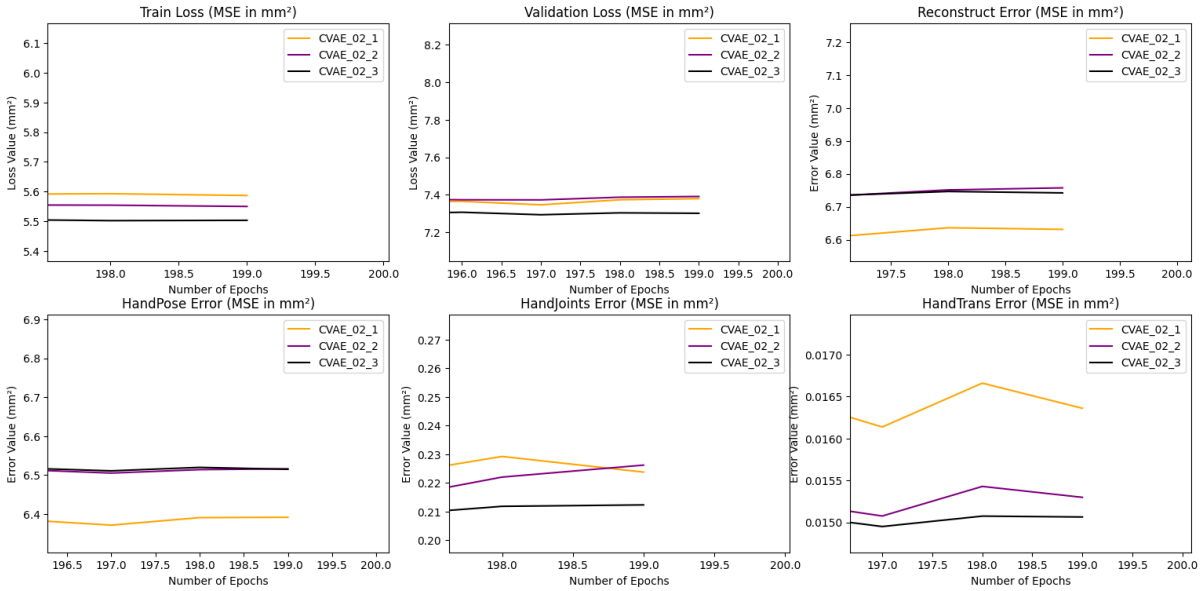


Figure 19: To have a better view of the difference in performances between the three last models, the graph in Figure 15 is zoomed.

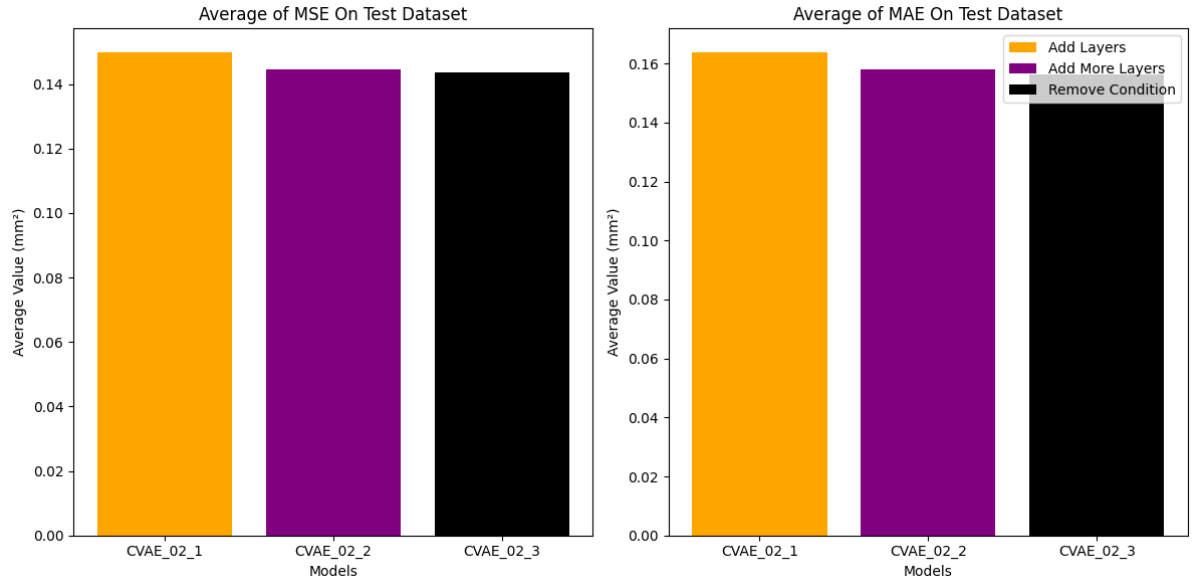


Figure 20: Average of mean squared and absolute errors for models CVAE_02_1, CVAE_02_2 and CVAE_02_3

The following graphs (Figure 20) show that removing the condition part barely decreases model errors and therefore CVAE_02_3 is the final model to be used in this dissertation for regenerating hand postures based on object information. Later, this model is used to infer conditioned on object information parameters (Object name= "cracker box") and is used to generate reconstructions of hand poses. The foregoing comparison shows that the model can simulate very human-like hand movements, given the object, which in turn further confirms the effectiveness for the purpose of this study.

4.2 Final Result Visualization (Reconstructed hands)

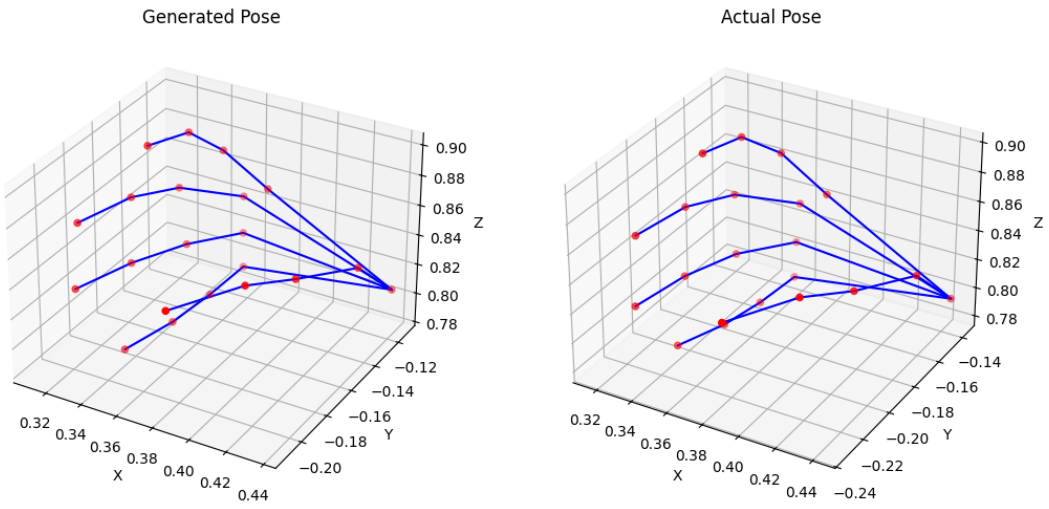


Figure 21: Hand pose generated based on a random object in test data by CVAE_02_3 model versus actual hand pose.

To plot the objects, the YCB dataset is used in combination with the HO3D dataset. Each object in the dataset contains a file called 'points.xyz' that contains 3D coordinates x, y, and z. With the aid of using the object's name, translation, and rotation for each frame in combination with the generated hand poses from the final model, plotting both the object and the corresponding reconstructed hand pose for every frame is feasible. This approach allows for accurate visualization of the interactions between the hand and object across the dataset (Figure 22).

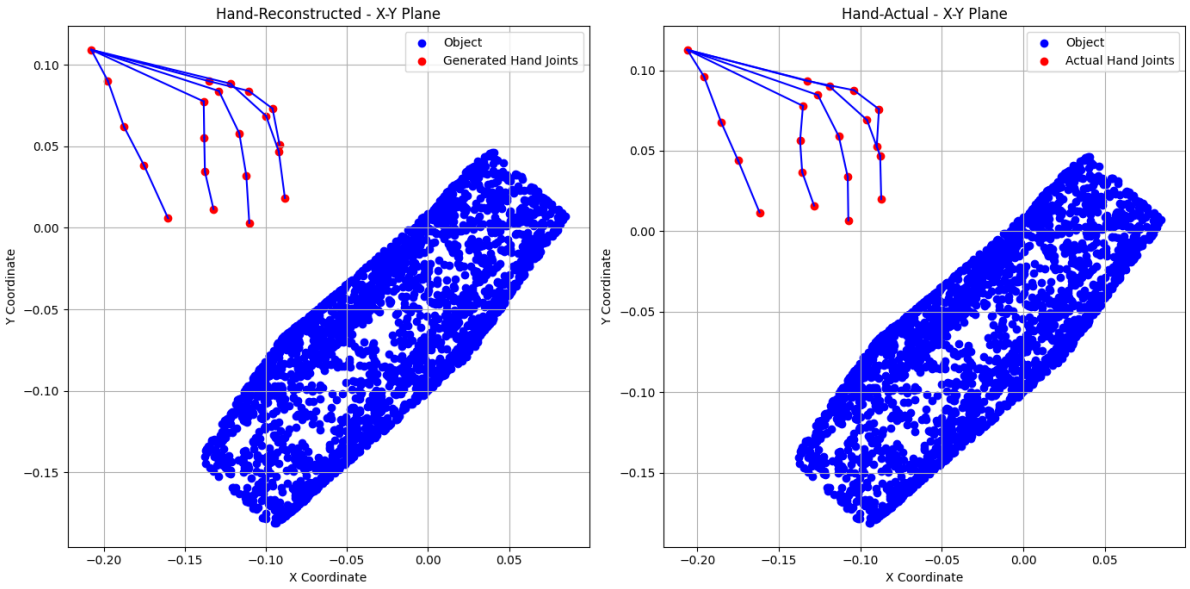
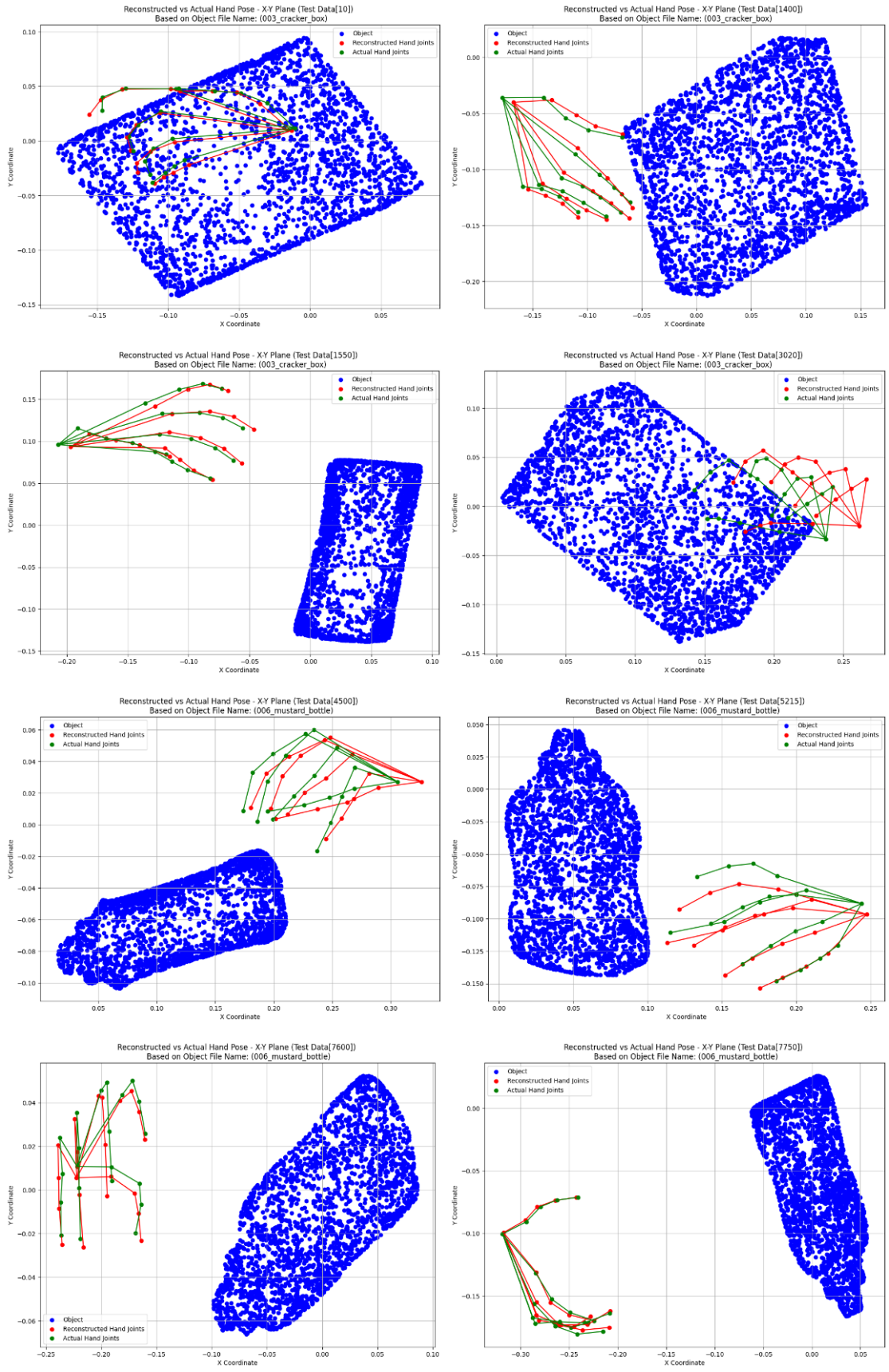
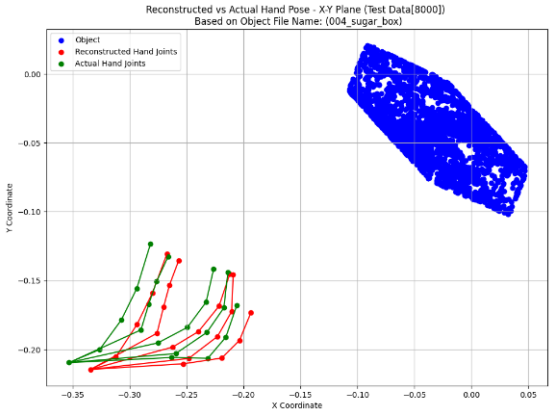
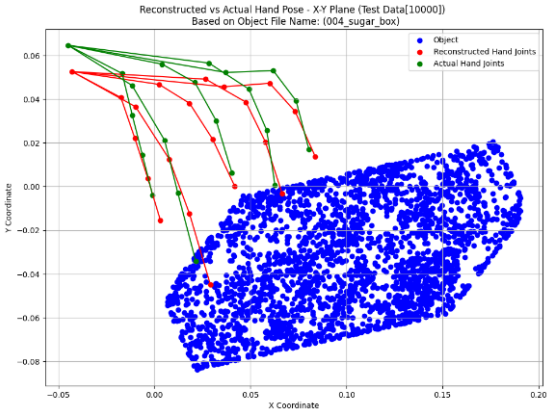


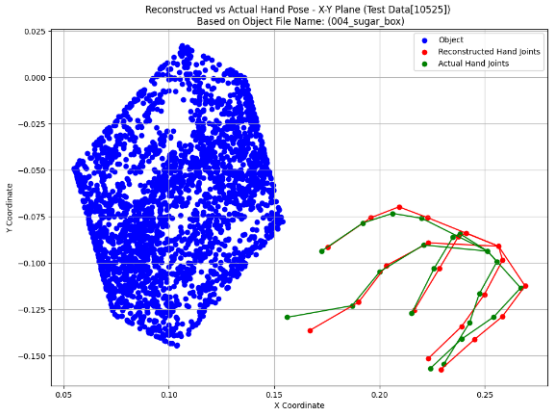
Figure 22: Reconstructed Hand Pose based on an Object from test data the object name based on YCB Dataset is '003_cracker_box'

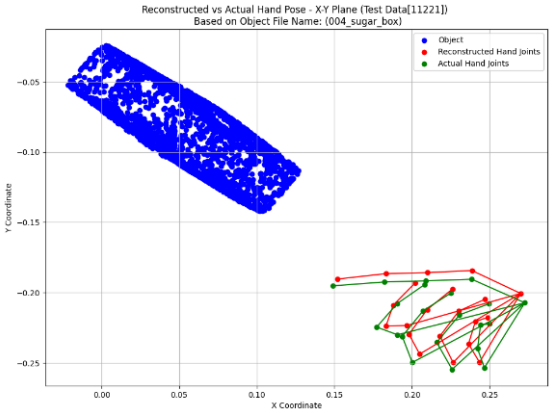
Next, the model is tested by generating hand poses based on randomly annotated object affordances from the test dataset.

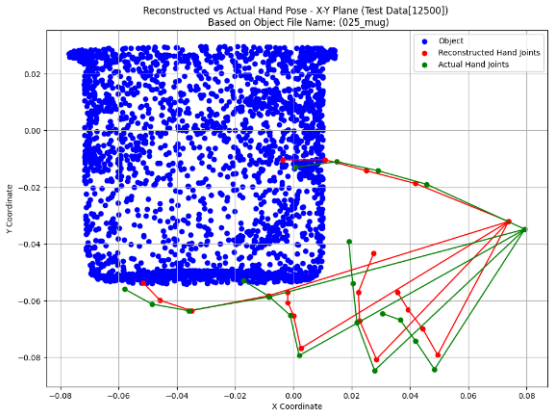


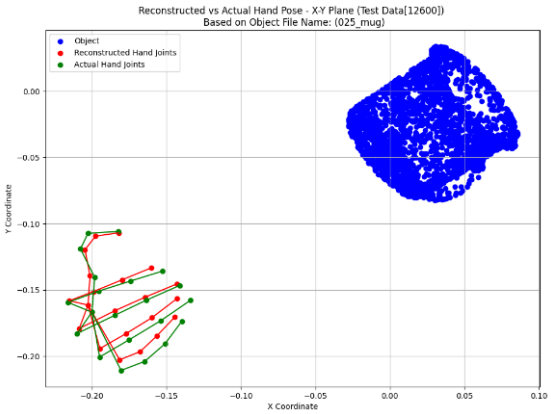

Reconstructed vs Actual Hand Pose - X-Y Plane (Test Data[8000])
Based on Object File Name: (004_sugar_box)
Object
Reconstructed Hand Joints
Actual Hand Joints
X Coordinate
Y Coordinate

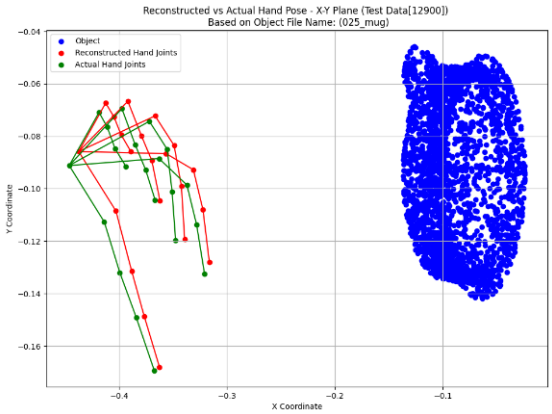

Reconstructed vs Actual Hand Pose - X-Y Plane (Test Data[10000])
Based on Object File Name: (004_sugar_box)
Object
Reconstructed Hand Joints
Actual Hand Joints
X Coordinate
Y Coordinate

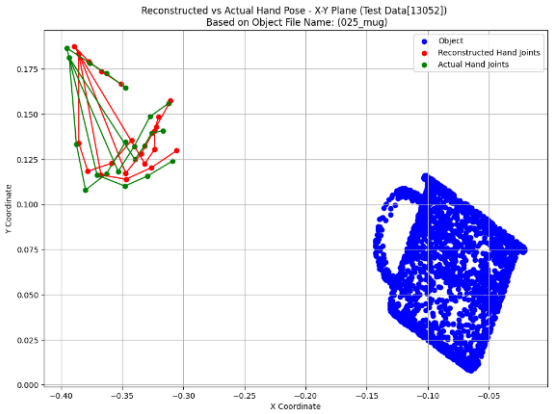

Reconstructed vs Actual Hand Pose - X-Y Plane (Test Data[10525])
Based on Object File Name: (004_sugar_box)
Object
Reconstructed Hand Joints
Actual Hand Joints
X Coordinate
Y Coordinate


Reconstructed vs Actual Hand Pose - X-Y Plane (Test Data[11221])
Based on Object File Name: (004_sugar_box)
Object
Reconstructed Hand Joints
Actual Hand Joints
X Coordinate
Y Coordinate


Reconstructed vs Actual Hand Pose - X-Y Plane (Test Data[12500])
Based on Object File Name: (025_mug)
Object
Reconstructed Hand Joints
Actual Hand Joints
X Coordinate
Y Coordinate


Reconstructed vs Actual Hand Pose - X-Y Plane (Test Data[12600])
Based on Object File Name: (025_mug)
Object
Reconstructed Hand Joints
Actual Hand Joints
X Coordinate
Y Coordinate


Reconstructed vs Actual Hand Pose - X-Y Plane (Test Data[12900])
Based on Object File Name: (025_mug)
Object
Reconstructed Hand Joints
Actual Hand Joints
X Coordinate
Y Coordinate


Reconstructed vs Actual Hand Pose - X-Y Plane (Test Data[13052])
Based on Object File Name: (025_mug)
Object
Reconstructed Hand Joints
Actual Hand Joints
X Coordinate
Y Coordinate

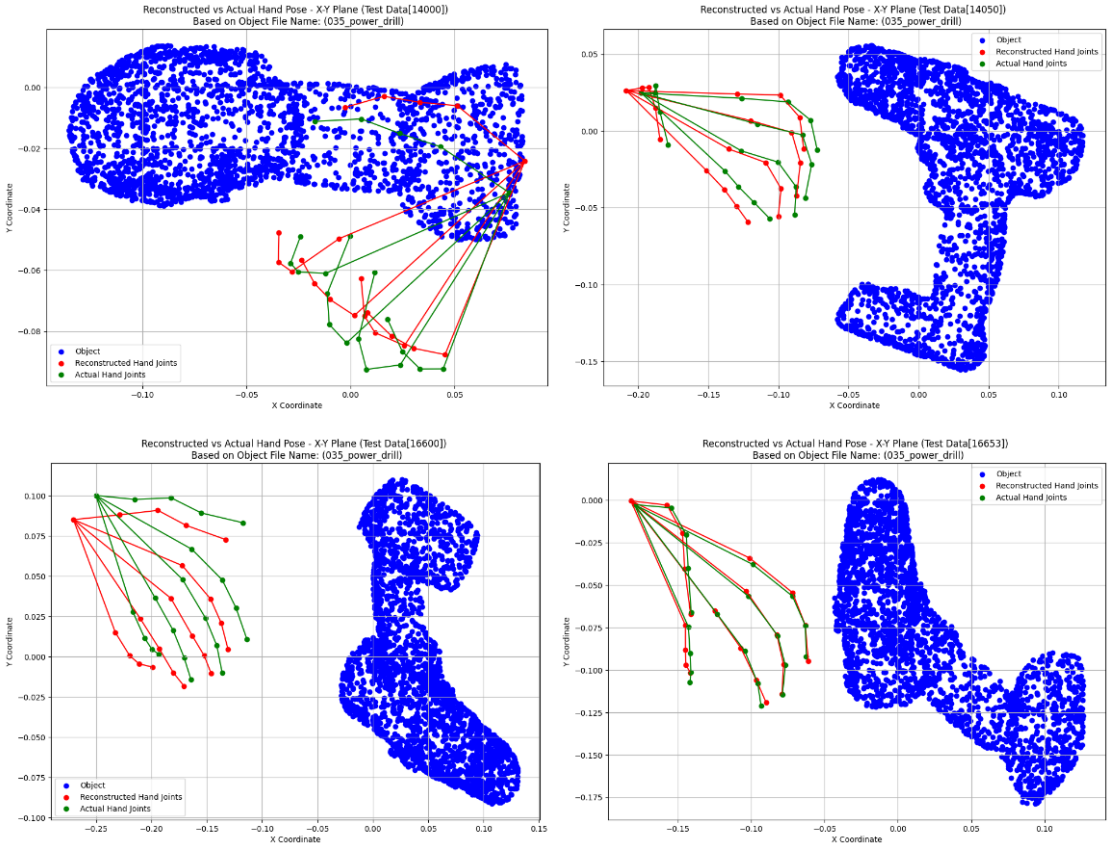


Figure 23: Testing the model CVAE_02_3 by generating hand poses based on randomly 3D rotation of five annotated object affordances from the test data (cracker box, mustard bottle, sugar box, mug, and power drill) - The blue represents the object, the red and green dots represent the actual and reconstructed hand joint positions, respectively. The closeness of the reconstructed joints to the actual indicates a high correlation between the two sets of data, with only slight deviations obvious between them, hence accurate reconstruction of hand

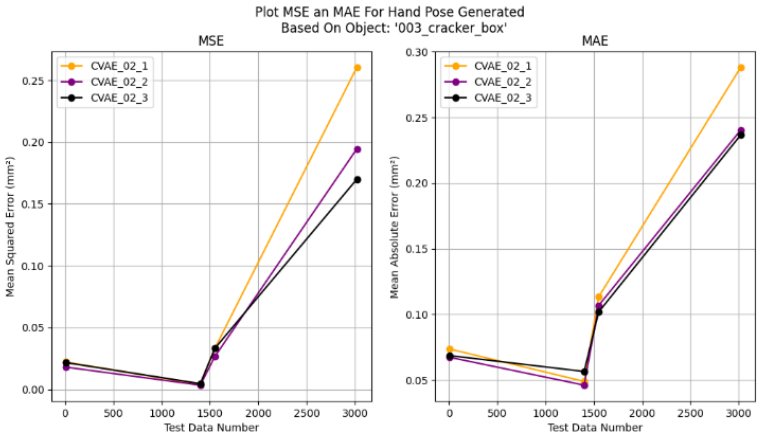
Figure 23 shows some samples of these results, while the corresponding errors and their details are presented in Table 9. Due to the strong correlation between the two sets, as indicated by the close alignment of the reconstructed joints with the real data, the deviations are very minimal. This demonstrates that the differences are very slight. Therefore, the reconstruction of the hand pose is highly accurate and reliable, effectively capturing the nuances of the original joint positions with precision. This provides a comprehensive evaluation of how well the model could predict hand poses over a range of object affordances and clear insight into its performance.

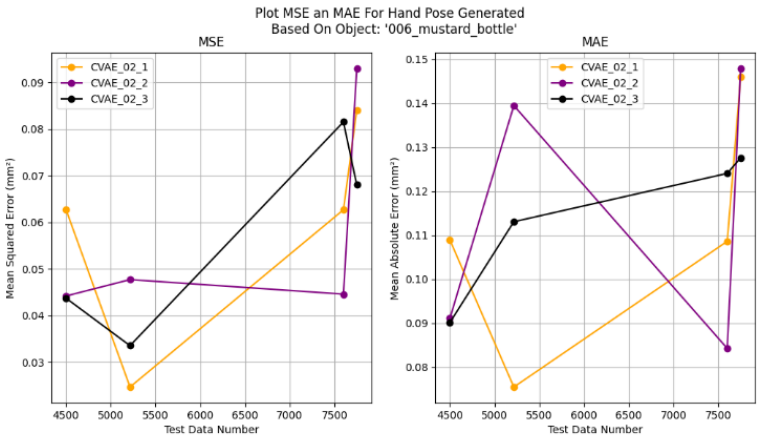
Object File Name	Test Data	CVAE_02_1		CVAE_02_2		CVAE_02_3	
		MSE	MAE	MSE	MAE	MSE	MAE
 003_cracker_box	10	0.02214	0.07350	0.01803	0.06721	0.02168	0.06853
	1400	0.00402	0.04886	0.00346	0.04600	0.00468	0.05642
	1550	0.03320	0.11325	0.02656	0.10665	0.03316	0.10174
	3020	0.26042	0.28807	0.19452	0.24028	0.16994	0.23654
 006_mustard_bottle	4500	0.06275	0.10893	0.04420	0.09111	0.04374	0.09016
	5215	0.02468	0.07551	0.04772	0.13946	0.03355	0.11307
	7600	0.06269	0.10862	0.04460	0.08429	0.08153	0.12405
	7750	0.08407	0.14601	0.09299	0.14788	0.06805	0.12760
 004_sugar_box	8000	0.18972	0.15505	0.06486	0.09280	0.05801	0.08891
	10000	0.02652	0.07554	0.02350	0.06315	0.02405	0.06330
	10525	0.09557	0.11846	0.09420	0.11787	0.10815	0.13472
	11221	0.02668	0.06919	0.03053	0.07746	0.03103	0.07737
 025_mug	12500	0.03833	0.09860	0.04844	0.12349	0.04394	0.10613
	12600	0.00185	0.03277	0.00282	0.04323	0.00162	0.03291
	12900	0.05709	0.07561	0.05632	0.06738	0.05268	0.07012
	13052	0.10840	0.11590	0.09769	0.09024	0.10308	0.09461
 035_power_drill	14000	0.24712	0.30400	0.13530	0.22366	0.05241	0.13036
	14050	0.09182	0.13309	0.09024	0.12743	0.09198	0.12706
	16600	0.03831	0.09022	0.04211	0.08903	0.04188	0.08894
	16653	0.06431	0.08015	0.05800	0.07534	0.06953	0.09124

Table 9: Calculating the mean squared and absolute errors for each generated hand pose in Figure 23.

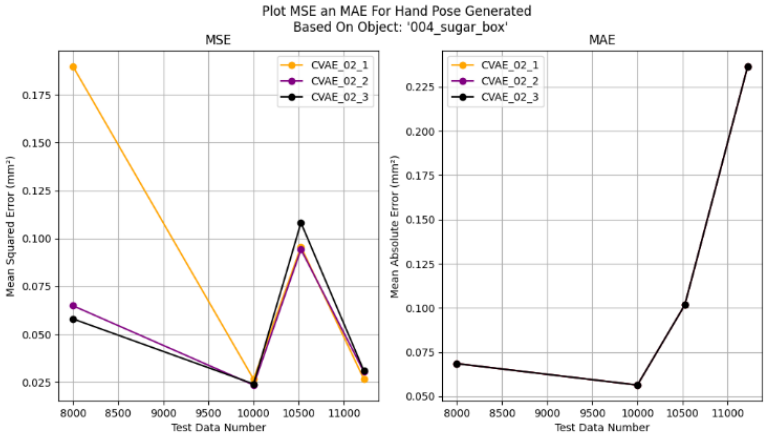
The charts in Figure 24 display the Mean Squared Error (MSE) and Mean Absolute Error (MAE) for three different CVAE models (CVAE_02_1, CVAE_02_2, CVAE_02_3) across various objects: Cracker Box, Mustard Bottle, Sugar Box, Mug, and Power Drill. These errors fluctuate significantly between different test data points of each object, while some models are more stable than others. In general, CVAE_02_3 has relatively lower errors for most objects, especially at the later points of test data, which means that it works better. For the Mustard Bottle and Power Drill cases, all models behaved similarly, though CVAE_02_3 had smaller peaks of MSE and MAE. In the case of a Sugar Box, it is more interesting because of the higher variance, and because CVAE_02_3 showed quite noticeable behavior with a significant rise at the very last test point. Overall, CVAE_02_3 is constantly competitive as far as the balance between MSE and MAE is concerned, therefore indicating better generalization and stability in the estimation of hand poses.


CHEEZ-IT


Plot MSE an MAE For Hand Pose Generated
Based On Object: '003_cracker_box'
MSE
MAE
CVAE_02_1
CVAE_02_2
CVAE_02_3
Mean Squared Error (mm²)
Mean Absolute Error (mm²)
Test Data Number


Plot MSE an MAE For Hand Pose Generated
Based On Object: '006_mustard_bottle'
MSE
MAE
CVAE_02_1
CVAE_02_2
CVAE_02_3
Mean Squared Error (mm²)
Mean Absolute Error (mm²)
Test Data Number


Domino
GRANULATED
SUGAR


Plot MSE an MAE For Hand Pose Generated
Based On Object: '004_sugar_box'
MSE
MAE
CVAE_02_1
CVAE_02_2
CVAE_02_3
Mean Squared Error (mm²)
Mean Absolute Error (mm²)
Test Data Number

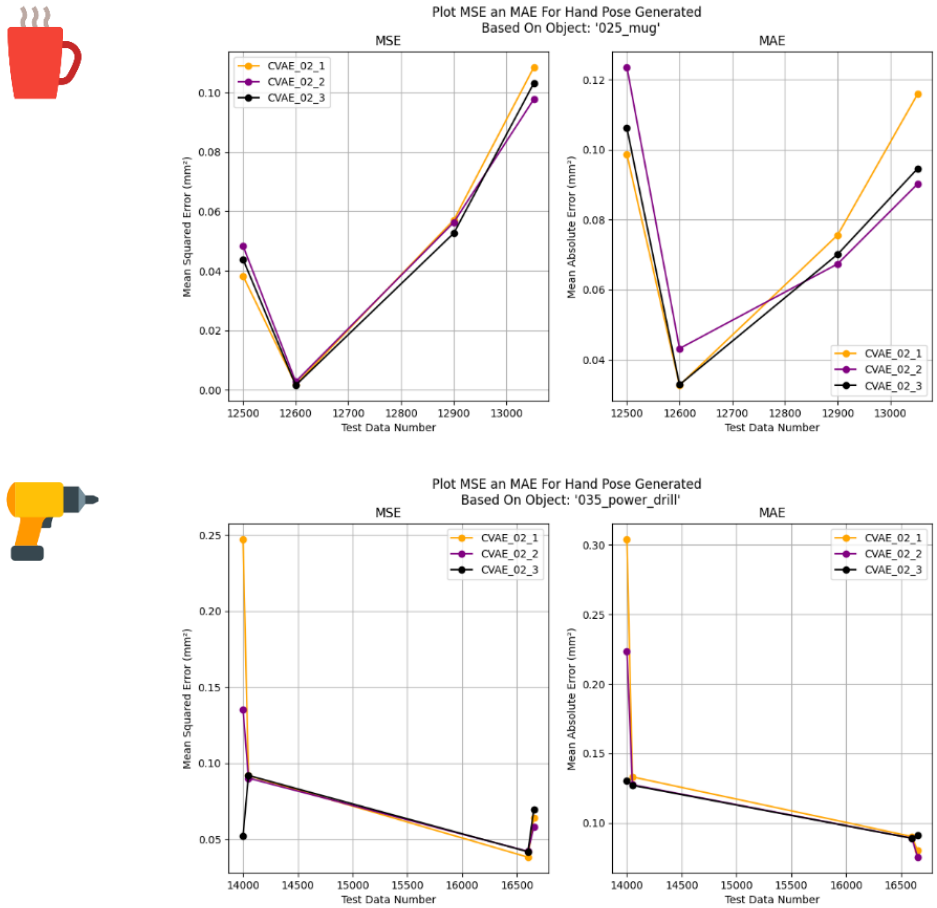


Figure 24: MSE and MAE of reconstructed hand pose with CVAE_02_3 for each object based on Table 9 information.

In the following, the average MSE and MAE of generated hand poses for each object from all frames in the test dataset have been calculated and presented in Figure 25.

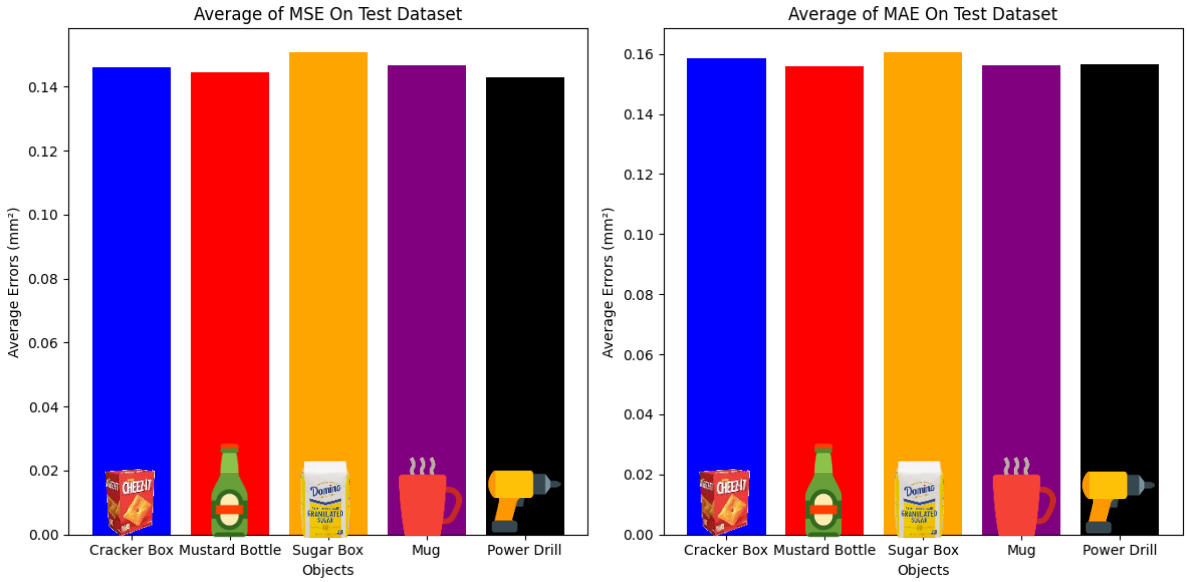


Figure 25: Average MSE and MAE of reconstructed hand pose for each object from all frames in test data.

These bar plots in Figure 25 show the error values can be relatively similar for all objects. For both MSE and MAE, the top error stays consistent as the Sugar Box, which is most problematic for this model. In contrast, Cracker Box, Mustard Bottle, Mug, and Power Drill present rather small and consistent errors, indicative of better hand pose predictions for these objects. Overall, the errors are in between 0.14 to 0.16 mm, indicating that while the model generally performs well, object-specific factors like the shape or affordances of the Sugar Box make it more challenging to predict accurate hand poses.

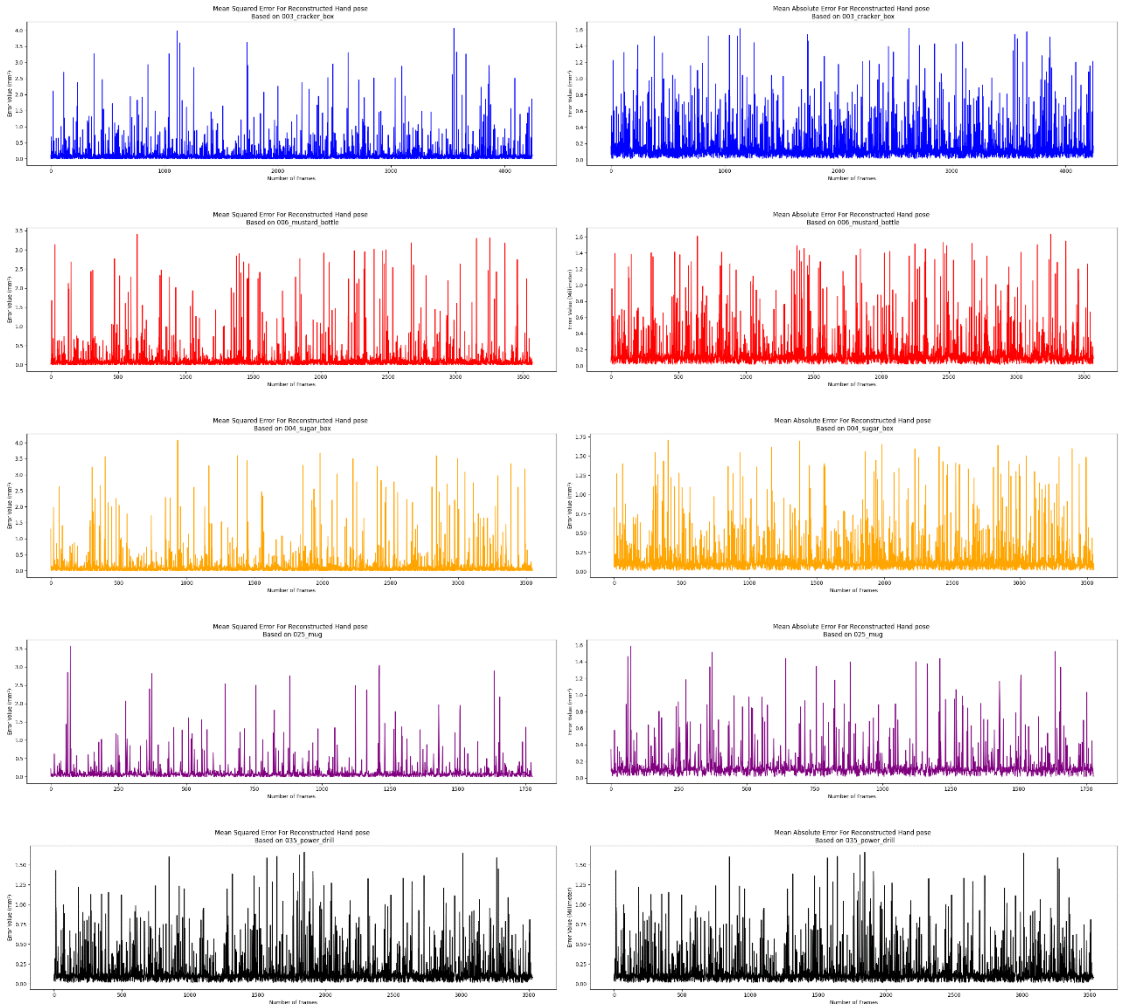


Figure 26: MSE and MAE of reconstructed hand pose for all frames in test data for objects '003_cracker_box', '006_mustard_bottle, '004_sugar_box, '025_mug', '035_power_drill' respectfully.

The graphs in Figure 26 reflect the error trend for many frames on the test dataset, which shows fluctuations of the hand pose reconstruction accuracy for all five objects. The peaks are higher for the MSE charts than for the MAE charts, which is expected since squared errors amplify the effect of larger deviations. Spikes in both graphs hint at periodic higher reconstruction errors in some frames, even though MAE values are generally smaller and more consistent, hence providing a clearer picture of average error magnitudes over time.

5.0 DISCUSSION

This work has introduced the successful usage of the Conditional Variational Autoencoder in synthesizing human-like hand poses from object affordances. In the results, it has been identified that CVAE_02 and its derivatives, particularly CVAE_02_3, attain the highest performance in synthesizing realistic hand poses. Such results point to the potential generative models that may come up in enhancing grasping synthesis for better flexibility and adaptability in different object interactions. These results represent a real contribution to the state-of-the-art in hand pose estimation, with promising implications for future applications in dynamic and complex environments.

5.1 Interpretation of Results

Going through an iterative process of designing and refining the CVAE models (as shown in the results section) points to the importance of model complexity and careful balancing in terms of performance versus computational efficiency. Among the implemented models, CVAE_02 and its variations derivatives (CVAE_02_1, CVAE_02_2, and CVAE_02_3) have the best performance in predicting human-like hand poses from object affordances. It is evident from the fact that MSE and MAE are continuously decreasing for further dimensionalities of latent space and increasing number of epochs that the model learns this complicated relation between hand poses and object characteristics effectively.

The last model, CVAE_02_3, removes the condition part from the encoder and shows a slight improvement in error reduction. Therefore, a generalized latent space could be more helpful when conditioning is applied only to the decoding phase. This would allow more flexibility for the model to generate hand poses, which can adapt to a greater variety of object affordances.

5.2 Comparison with Previous Research

This work is performed by building upon previous studies in the field of generative modeling and robotics, specifically, those works applying VAE and GAN for similar problems. The results also align with the general trend from the broader literature that generative models, given proper training, could significantly enhance the ability of robots to perform such a complicated task within a dynamic environment. This dissertation, however, will add to the literature in the sense that CVAEs for hand action synthesis will

be applied in a very specific way, much more focused on this particular area compared to general-purpose generative models.

5.3 Explanation of Unexpected Results

An unexpected observation occurred when a minor improvement in performance was noted after adding noise to the hand information during the training of CVAE_03. It was initially assumed that adding noise during training would help with generalization and improve the robustness of the model, but there was only a modest change in performance when the model was trained with noise. One explanation for this could be the inherent robust structure of the CVAE architecture, which may have enough design capability to train a robust model without adding noise to the hand representation. Further studies introducing different types of noise at different stages of the training may lend future insights.

5.4 Limitations

Despite the encouraging results described here, several limitations remain. First, models are trained and tested using the dataset HO3D_v3 could not fully express all types of object affordances that might arise in real-world settings. Moreover, the computational resources required to train these models may be highly in demand, especially as the models are complexified further. This may raise practical issues for real-world applications running in real time. Thirdly, this work focused only on the accuracy of hand pose reconstruction alone, while its real effectiveness in a robotic manipulation task needs further research.

5.5 Future Research Directions

The results in Figure 20 show that the actual and reconstructed hand joint paths follow each other quite closely. In other words, the model reconstructed hand poses rather well. However, there are slight discrepancies between the actual and reconstructed points, suggesting that while the reconstruction is generally reliable, some fine-tuning may be needed to improve accuracy, especially in more complex or nuanced movements. This would further enhance the model by reducing biases and making the network focus on the reduction of relative displacements between reconstructed and actual poses by using mean-centered errors (MCE) during training. This will regulate the systematic shifts of the reconstructed poses, can be converged faster, and may also provide a better generalization as it becomes less sensitive to outliers.

Moreover, to convert generated hand points to actual hand pose points, Procrustes Analysis algorithms (Vandeginste et al., 1998) can be used. This is the method of statistical shape analysis, by which two sets of points are aligned. Procrustes analysis does transformations such as translation, scaling, and rotation in order to minimize the difference between the generated and actual points. This is really useful when it comes to aligning two shapes that are similar but may be rotated, scaled, or shifted with respect to one another.

To improve the accuracy of prediction and generation of grasping, the x, y, and z points (three-dimension points) of objects which are allocated in the 'models' folder (the file name is 'points.xyz') which belongs to the YCB dataset, can be considered to concatenate with the object information for training models.

Furthermore, success in the application of CVAEs in this work would therefore imply that such a model can easily be integrated into robotic systems to enhance the manipulation of objects by these robots in a very human-like manner. This could have significant implications for industries involved in the handling of a large variety of objects by robots, such as manufacturing, healthcare, and service industries. This would allow the robot to generate suitable hand actions with respect to object affordances without large-scale retraining or reprogramming. In this manner, more versatile and flexible robotic systems can be achieved.

The dataset could be expanded by a wider variety of objects and their corresponding affordances. It would allow the generative models to be exposed to a diverse set of features and interactions. Therefore, their potential outcomes could enhance the model's ability to generalize to different scenarios. So, when it comes to the simulation of future hand poses action specifically to encounter an unfamiliar object, the models' predictive would work powerfully. Furthermore, with additional affordances—such as different object sizes, shapes, textures, and functional uses— the models can learn more complicated relationships between object and hand pose that ultimately enable robots to perform grasping objects efficiently and be adaptable in a dynamic environment.

However, although autoencoders are a promising start, they may be an imperfect choice to generate this type of precise, structured output like hand pose given object information. Other generative models such as GAN could also be better for exploring multiple objects and affordances which would be valuable in comparison with CVAEs.

6.0 CONCLUSION

This dissertation investigated how Conditional Variational Autoencoders (CVAEs) can be used to imitate human hand poses for robotic systems based on object affordances. Crafting hand poses based on object affordances traditionally requires labor-intensive manual labeling of object parts and hand poses, and optimization-based approaches struggle to account for object functionality. The achievements in the development and improvement of the CVAE models described in this thesis illustrate that generative models can be employed to great effect in the provision of robotic hand poses with high-level abilities to manipulate objects.

The significant contribution of this work is to develop the CVAE architecture, specifically the iterative process of designing, training, and assessing different models of the CVAE architectures, where models like CVAE_02, particularly its variant, CVAE_02_3, have shown promising enhancement in producing hand poses that can be adapted to various object affordances. By using a more generalized latent space, CVAE_02_3 enables adaptive pose generation for a wide variety of objects without extensive retraining. This functionality is quite effective in practical applications where robots need to act upon various objects in dynamic environments with minimal computational cost. These results suggest that the model CVAE_02_3 effectively addressed some of the key challenges of the prior methods with balancing between performance and resource efficiency.

The work further extends the use of CVAEs beyond their usual usage to hand pose synthesis, which has been under-explored compared to other more general applications with generative models. It confirms that, if applied correctly, CVAEs can radically improve the autonomy and adaptability of robotic systems and consequently transform their interaction with the real world. The minimal reconstruction error of hand poses by CVAE_02_3, reflected by the measure of MSE and MAE, points to the efficiency of this model to mimic human-like grasping motions.

However, the investigation also recognized several drawbacks, such as reliance upon a, to some extent, single dataset (HO3D_v3) as well as high model training resources. These factors might reduce the effectiveness of the models for immediate use in real-time applications. Also, although the analysis is carried out to evaluate the reconstruction accuracy of hand poses, less attention is given to how effective those poses would be in real manipulations of robot hands. Future work should explore how poses generated by the model impact functionality in practical manipulation tasks with robotic hands, thus giving a more complete assessment of model performance.

Although the CVAE models are very powerful, they require significant training resources, especially from the viewpoint of computational resources, which may hinder their applications in real-time robotic systems. Further optimization in terms of computational efficiency will help to unlock their applications in systems requiring fast consecutive responses, such as autonomous service robots.

Looking forward, these are several promising avenues for further research. The logical next step would be to integrate CVAE-based models with complete robotic systems and test their performance in real-world tasks. Additional work can also be inspired by even more advanced generative models, such as Generative Adversarial Networks, which might offer new insights and outperform CVAE on some applications. It would also be very interesting to further increase the diversity of the dataset with more complex objects, with variable affordances to push the limits of generalization by the optimized model.

In conclusion, this thesis has provided important steps for generative models in hand pose synthesis and laid a good foundation for further development in the manipulation of robots using object affordances. Results have shown that the CVAE models, especially the best one, CVAE_02_3, are a completely workable pathway toward developing more autonomous and adaptive robotic systems. The contribution of this present work goes to the academic understanding of CVAEs within the context of the generation of hand poses and further proposes practical applications that may shape the future of robotics within complex and dynamic environments. It will be the beginning of further exploration of generative models in robotics, opening ways for robots to interact more intelligently with the world and in a far more effective manner.

7.0 REFERENCES

BARAKAZI, M. (2022). The use of Robotics in the Kitchens of the Future: The example of'Moley Robotics'. *Journal of Tourism & Gastronomy Studies, 10*(2), 895-905.

Bertasius, Gedas; Soo Park, Hyun; Yu, Stella X; Shi, Jianbo. (2017). Unsupervised learning of important objects from first-person videos. *Proceedings of the IEEE International Conference on Computer Vision*, (pp. 1956-1964).

Beßler, Daniel and Porzel, Robert and Pomarlan, Mihai and Beetz, Michael and Malaka, Rainer and Bateman, John. (2020). A formal model of affordances for flexible robotic task execution. In *ECAI 2020* (pp. 2425-2432). IOS Press.

Borghi, Anna M. (2021). Affordances, context and sociality. *Synthese, 199*, 12485--12515.

Bütepage, J., Ghadirzadeh, A., Öztimur Karadağ, Ö., Björkman, M., & Kragic, D. (2020). Imitating by generating: Deep generative models for imitation of interactive tasks. *Frontiers in Robotics and AI, 7*, 47.

Cai, Yujun; Wang, Yiwei; Zhu, Yiheng; Cham, Tat-Jen; Cai, Jianfei; Yuan, Junsong; Liu, Jun; Zheng, Chuanxia; Yan, Sijie; Ding, Henghui. (2021). A unified 3d human motion synthesis model via conditional variational auto-encoder. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, (pp. 11645-11655).

Calli, Berk and Walsman, Aaron and Singh, Arjun and Srinivasa, Siddhartha and Abbeel, Pieter and Dollar, Aaron M. (2015). Benchmarking in manipulation research: Using the Yale-CMU-Berkeley object and model set. *IEEE Robotics \& Automation Magazine, 22*, 36-52.

Cèsar-Tondreau, Brian; Warnell, Garrett; Stump, Ethan; Kochersberger, Kevin; Waytowich, Nicholas R. (2021). Improving autonomous robotic navigation using imitation learning. *Frontiers in Robotics and AI*, 627730.

Chaudhuri, S., Ritchie, D., Wu, J., Xu, K., & Zhang, H. (2020). Learning generative models of 3D structures. In *Computer Graphics Forum* (Vol. 39, pp. 643-666). Wiley Online Library.

Cordella, Francesca; Zollo, Loredana; Salerno, Antonino; Accoto, Dino; Guglielmelli, Eugenio; Siciliano, Bruno. (2014). Human hand motion analysis and synthesis of optimal power grasps for a robotic hand. *International Journal of Advanced Robotic Systems, 11*(3), 37.

Degardin, Bruno; Neves, Joao; Lopes, Vasco; Brito, Joao; Yaghoubi, Ehsan; Proença, Hugo. (2022). Generative adversarial graph convolutional networks for human action synthesis. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, (pp. 1150-1159).

Do, Thanh-Toan; Nguyen, Anh; Reid, Ian. (2018). Affordancenet: An end-to-end deep learning approach for object affordance detection. *2018 IEEE international conference on robotics and automation (ICRA)* (pp. 5882-5889). IEEE.

Garcia-Hernando, Guillermo and Yuan, Shanxin and Baek, Seungryul and Kim, Tae-Kyun. (2018). First-person hand action benchmark with rgb-d videos and 3d hand pose annotations. *Proceedings of the IEEE conference on computer vision and pattern recognition*, (pp. 409-419).

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM,, 63*, 139-144.

Hampali, Shreyas; Rad, Mahdi; Oberweger, Markus; Lepetit, Vincent. (2020). Honnotate: A method for 3d annotation of hand and object poses. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, (pp. 3196-3206).

Hand-Object 3D Pose Annotation. (2020). Retrieved from TU Graz: https://www.tugraz.at/institute/icg/research/team-lepetit/research-projects/hand-object-3d-pose-annotation/

Hou, Zhi; Yu, Baosheng; Qiao, Yu; Peng, Xiaojiang; Tao, Dacheng. (2021). Affordance transfer learning for human-object interaction detection. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, (pp. 495-504).

Imre, Mert and Oztop, Erhan and Nagai, Yukie and Ugur, Emre. (2019). Affordance-based altruistic robotic architecture for human--robot collaboration. *Adaptive Behavior, 27*, 223-241.

Jian, Juntao; Liu, Xiuping; Li, Manyi; Hu, Ruizhen; Liu, Jian. (2023). Affordpose: A large-scale dataset of hand-object interactions with affordance-driven hand pose. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, (pp. 14713-14724).

Jupyter. (2024). Retrieved from Jupyter: https://jupyter.org/

Keras. (2024). Retrieved from Keras: https://keras.io

Kim, David Inkyu; Sukhatme, Gaurav S. (2014). Semantic labeling of 3d point clouds with object affordance for robot manipulation. *2014 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 5578-5584). IEEE.

Krupnik, O., Shafer, E., Jurgenson, T., & Tamar, A. . (2023). Fine-Tuning Generative Models as an Inference Method for Robotic Tasks. . In *Conference on Robot Learning* (pp. 866-886). PMLR.

Lamb, A. (2021). A brief introduction to generative models. . *arXiv preprint arXiv:2103.00265.*

Lin, Yen-Chen; Florence, Pete; Zeng, Andy; Barron, Jonathan T; Du, Yilun; Ma, Wei-Chiu; Simeonov, Anthony; Garcia, Alberto Rodriguez; Isola, Phillip. (2023). Mira: Mental imagery for robotic affordances. *Conference on Robot Learning* (pp. 1916-1927). PMLR.

MANO. (2020). Retrieved from MANO: https://mano.is.tue.mpg.de/

matplotlib. (2024). Retrieved from matplotlib: https://matplotlib.org

Mi, Jinpeng and Tang, Song and Deng, Zhen and Goerner, Michael and Zhang, Jianwei. (2019). Object affordance based multimodal fusion for natural human-robot interaction. *Cognitive Systems Research, 54*, 128-137.

Nguyen, Thao and Gopalan, Nakul and Patel, Roma and Corsaro, Matt and Pavlick, Ellie and Tellex, Stefanie. (2022). Affordance-based robot object retrieval. *Autonomous Robots, 46*, 83-98.

NumPy. (2024). Retrieved from NumPy: https://numpy.org

Pickle. (2024). Retrieved from Python: https://docs.python.org/3/library/pickle.html

Pohl, Christoph and Hitzler, Kevin and Grimm, Raphael and Zea, Antonio and Hanebeck, Uwe D and Asfour, Tamim. (2020). Affordance-based grasping and manipulation in real world applications. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 9569-9576). IEEE.

Pronobis, A., & Rao, R. P. (2017). Learning deep generative spatial models for mobile robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 755-762). IEEE.

PyCharm. (2024). Retrieved from JETBRAINS: https://www.jetbrains.com/pycharm/

Pytorch. (2024). Retrieved from pytorch: https://pytorch.org/

Renaudo, Erwan and Zech, Philipp and Chatila, Raja and Khamassi, Mehdi. (2022). Computational models of affordance for robotics. *Frontiers in Neurorobotics, 16*, 1045355.

Romero, Javier and Tzionas, Dimitrios and Black, Michael J. (2017). *Embodied Hands: Modeling and Capturing Hands and Bodies Together*. Retrieved from MANO: https://mano.is.tue.mpg.de/

Roomba Robot Vacuums. (2024). Retrieved from iRobot: https://www.irobot.com/en_US/us-roomba.html

scikit-learn. (2024). Retrieved from scikit-learn: https://scikit-learn.org/stable

Seepanomwan, K. (2019). How mental imagery helps robot learning. I. In K. Seepanomwan, *n 2019 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engine* (pp. 245-250). IEEE.

Tahir, Rohan; Sargano, Allah Bux; Habib, Zulfiqar. (2021). Voxel-based 3D object reconstruction from single 2D image using variational autoencoders. *Mathematics, 9*(18), 2288.

Tanwani, A. K. (2018). Generative models for learning robot manipulation skills from humans. *EPFL*.

TensorFlow. (2024). Retrieved from TensorFlow: https://www.tensorflow.org/

Vandeginste,B.G.M.; Massart,D.L.; Buydens,L.M.C.; De Jong,S.; Lewi,P.J.; Smeyers-Verbeke,J. (1998). Chapter 35 - Relations between measurement tables. *Data Handling in Science and Technology, 20*, 307-347. doi:10.1016/S0922-3487(98)80045-2

Ventura, Sara and Tessari, Alessia. (2021). Do Object Affordances Modulate the Sense of Embodiment in Virtual Human--Tool Interaction? Reflection for the Interactive Artificial Environment. *PRESENCE: Virtual and Augmented Reality, 30*, 203--212.

Visual Studio Code. (2024). Retrieved from Visual Studio Code: https://code.visualstudio.com/

Yuan, K., Sajid, N., Friston, K., & Li, Z. (2023). Hierarchical generative modelling for autonomous robots. *Nature Machine Intelligence, 5*, 1402-1414.

Zhou, Yuxiao; Habermann, Marc; Xu, Weipeng; Habibie, Ikhsanul; Theobalt, Christian; Xu, Feng. (2020). Monocular real-time hand shape and motion capture using multi-modal data. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, (pp. 5346-5355).

APPENDIX A: RESEARCH PROJECT PLAN

Engineering Generative Models for Robots to Simulate Future Actions Through Mental Imagery

Siavash Mortaz Hejri
MSc Artificial Intelligence
Sheffield Hallam University
c2056757@hallam.shu.ac.uk

1.INTRODUCTION AND JUSTIFICATION

Nowadays, robots have become an inseparable part of our daily life. Recently, their potential abilities to navigate real-world environments and perform meaningful tasks require more than just being a mechanical machine. Robots should be able to adapt their strategies to be useful for future jobs, and all these processes are crucial for them to be safety and frequently interactive with the real-world and approach their goals.

Robots are designed and programmed based on pre-defined methods traditionally which cannot be flexible in some kinds of real-time situations. Moreover, in these situations updating their software with complex scenarios is at a high-cost and time-consuming job for experts. Due to these limitations, researchers have started to explore and use mental imaginary in robots.

This dissertation will illustrate that generative AI models and mental imagery concept can be used as a utility for robots to develop themselves. Generative AI models can be trained to create a realistic simulation of their future actions. Integrating Generative AI in robots can allow them to act more flexibility and when it comes to making decisions, they can do it efficiently.

2. RESEARCH QUESTION, AIMS, OBJECTIVES AND DELIVERABLE

2.1 Research question

How can generative models be engineered to enable robots to simulate future actions through mental imagery?

2.2 Aim

The intention of this research work is to investigate and improve on the generative models specifically, Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) that will aid integration with robots in order to promote generative future action resembling mental imagery in humans.

2.3 Objectives

i. Identifying and evaluating generative models which exist for action simulation in robots.
ii. Create and train a generative model for predicting and visualizing actions in robots.
iii. Make a framework that can be used to integrate the generative model into a control system that helps robots to simulate future actions.

iv. Evaluate the efficiency of the enhanced robots in the real-world environment.

2.4 Deliverable

A specialized developed generative model (VAE or GAN) which is trained with robots' data to predict their future actions. This tool can be standalone for researchers and roboticists to integrate into their systems potentially.

3.Literature review

The literature review is divided into two general themes: Mental imagery and Generative models. Considering key findings and theoretical perspectives from relevant literature, this review seeks to explain the significance and challenges, and future directions in using mental imagery and generative models for enhancing robotic capabilities.

3.1 Mental Imagery: A Cognitive Capability

Understanding Mental Imagery

Mental Imagery is a fundamental cognitive mechanism for human perception and action planning. Although there are still ambiguities about how cognitive processes work, there is much research on investigate the neural and cognitive processes which are involved in mental imagery and their role in cognitive tasks such as problem solving, memory amplification, and motor planning by many psychologists and neuroscientists.

An idea through researchers is that humans imagine pictures in their mind with the exact part of the brain which is invoked when see some pictures. However, (Pylyshyn, 2002) believes that neuroscience findings do not strongly support the idea that mental images are picture-like. The study challenges some assumptions about mental imagery by asking questions like whether mental images are spatially or depictively represented in the brain.

On the other hand, (Kosslyn, 2005) provides some evidence and suggests that mental imagery uses similar brain mechanisms as visual perception, and certain brain areas are involved. It means that humans can imagine objects and everything with their visionary part of the brain.

To discuss how people can combine their ideas and the role of mental imagery in this process, a group of people were asked to think about objects and write down their features. The results showed that when participants used mental pictures their ideas were engaged literally. For instance, they tend to write about features of objects which were visible for them than ones that were hidden or did not see. Consequently, this indicates mental imagery is an integral part of how people think and understand the world around them (Wu, L. L., & Barsalou, L. W., 2009).

Role of Mental Imagery in Robotics

Enhancing robots with the power of imagination they would be able to imagine their next move, anticipating obstacles and simulating future action which leads to interacting more intelligently with their surroundings, humans, and objects.

For this aim, it is required to know how the brain thinks about the body and represents it. This helps both humans and robots do complex tasks. Although many mysteries remain about body representations, including concepts like body schema and body image, (Hoffmann, M., Marques, H., Arieta, A., Sumioka, H., Lungarella, M., & Pfeifer, R.,

2010) explores how human's brain represents its body and examines how robots can be useful for studying biological body representations.

Scientists believe beside visual imagery, there is a link between bodily movements and mental processes. Despite ongoing technologies in robotics, there are still challenges in developing complex behaviors. This motivates further exploration of how mental imagery can improve motor control in robots. (Di Nuovo, A., De La Cruz, V. M., & Marocco, D., 2013) cover different aspects of mental imagery, from dreams to navigation and mental rotation processes.

In the following, another challenge comes up, how robots can use their thoughts to plan actions and manipulate their environments. GNOSYS Cognitive architecture is designed to regulate robots' behavior in terms of smart acts. The two main parts of GNOSYS are controlling what the robot does and helping the robot to think about its future acts (Kasderidis, 2008). By aiding and applying GNOSYS cognitive architecture (Mohan, V., Morasso, P., Metta, G., & Kasderidis, S., 2010) explains more about robots that would be able to generate actions mentally and physically. Using these internal models, robots can simulate movements, executing actions creating mental maps of their environments and interacting effectively.

For boosting the performance of humanoid robot through autonomous learning with using modular artificial neural networks, a model of a neural controller is introduced by (Di Nuovo, A. G., Marocco, D., Di Nuovo, S., & Cangelosi, A, 2013) which can enabling the robot to improve its sensorimotor skills independently. It would be feasible by combining a secondary system that could generate imaginary examples based on the robots' existing skills, practice in its mind by imagining different scenarios and facilitate simulated mental training.

Moreover, (Seepanomwan, 2019) works on a computer model for teaching robots. The main experiment is teaching a humanoid robot how to grab something far away by using a tool. Due to experimentation and reinforcement robots learn not only basic movement but also how they can imagine things in their minds. In addition, robots use the imaginary to figure out whether their action will be efficient or not which can save time and energy potentially.

3.2 Generative Models: Envisioning the Future

The Rise of Generative Models

These days, AI has experienced a revolution via generative models. These models can learn patterns from big data and generate coherent and integrated content. Generative models can learn from its mistake, which allow machines to learn from unlabeled dataset to create new data examples (Lamb, 2021). Generative adversarial networks (GANs) are good at making things like pictures look real. They are used for many different tasks. However, working with these models is still tough because the designing rely on game theory, a mathematical theory, which is different from the most other programs structure (Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., & Bengio, Y, 2020).

Generative Models in Robotics

Scientists survey simulating mental imagery by using generative models in robots. GAN models provide potential solutions for robots whether it is a robot replanning its path in a mazed room or predicting its next movement.

3D simulated environments are places for robots to learn skills before applying them in a real-world environment. It can be practical with generative models of 3D content. (Chaudhuri, S., Ritchie, D., Wu, J., Xu, K., & Zhang, H., 2020) explored generative models of 3D shapes and environments, and examined various methodologies such as probabilistic models, deep generative models, and neural network models for structured data.

With the aid of deep generative models (DGMs), (Bütepage, J., Ghadirzadeh, A., Öztimur Karadağ, Ö., Björkman, M., & Kragic, D., 2020) use a new probabilistic latent variable model that predicts movements in a hidden space. Their experiments show that successful human-robot interaction (HRI) depends on a model which can consider human motion and task dynamics. Consequently, it can create synchronized behavior between human and robot.

Despite all this research, there are various challenges in robotics around understanding spatial environments. To addressing this, DGSM, a new deep generative spatial model for robotics is introduced by (Pronobis, A., & Rao, R. P., 2017). The article claims that DGSMs could learn deep probabilistic representation of robotic environments from different views includes low-level features, geometry, and semantics.

Theme		Author	Key Points	Challenges	Future Directions
Mental Imagery	Understanding Mental Imagery	Pylyshyn (2002)	Mental imagery may not be picture-like but involve general reasoning with visual information	Understanding body representation	–
		Kosslyn (2005)	Mental imagery uses similar brain mechanisms as visual perception	Integrating mental imagery with motor control	Develop robots that can generate actions mentally
		Wu, L. L., & Barsalou, L. W. (2009)	Mental imagery is an integral part of how people think and understand the world	–	Leverage mental imagery for simulated training
	Role of Mental Imagery in Robotics	Hoffmann et al. (2010)	Understanding body representation is crucial for complex tasks	–	–
		Di Nuovo et al. (2013)	Mental imagery has various aspects including navigation and mental rotation	–	–
		Kasderidis (2008)	GNOSYS Cognitive architecture is designed to regulate robots' behavior	–	Develop robots that can think about future acts
		Mohan et al. (2010)	Robots with GNOSYS can simulate movements and create mental maps	–	–
		Di Nuovo et al. (2013)	A neural controller with imaginary examples can improve robot sensorimotor skills	–	Leverage mental imagery for simulated training
		Seepanomwan (2019)	Robots can learn to imagine and use it to figure out efficient actions	–	–
Generative Models	The Rise of Generative Models	Lamb (2021)	Generative models can learn patterns and generate data	Complex design based on game theory	Leverage generative models for 3D environment creation
		Goodfellow et al. (2020)	Generative Adversarial Networks (GANs) are a type of generative model	–	–
	Generative Models in Robotics	Chaudhuri et al. (2020)	Generative models of 3D content can be used for robot skill learning	–	Leverage generative models for 3D environment creation
		Bütepage et al. (2020)	Deep generative models (DGMs) can predict movements for robots	Understanding spatial environments	Develop probabilistic models for human-robot interaction (HRI)
		Pronobis & Rao (2017)	Deep Generative Spatial Models (DGSMs) can improve robot understanding of spatial environments	–	Develop probabilistic models for human-robot interaction (HRI)

Table 1: Literature review of key values, challenges, and future directions

4. RESEARCH DESIGN

In this section, a research structure is described, which includes guiding philosophy, methodologies, and techniques to achieve objectives. It is ensured that the design is consistent with generative AI applications in robotics. The main areas discussed include research philosophy, approach, methodology, tools, techniques, and data collection and analysis methods for practical application.

4.1 Research Philosophy, Approach and Methodology

The research basically figuring out how to use the AI models to make robots perform better, which is all about finding solutions that work. These models help robots adapt and make decisions in real-world situations. This matches well with the practical approach of pragmatism, which emphasizes practical consequences and utility in research.

The study analyses existing research on generative models (VAEs and GANs) and their success in different tasks. Studying how these models have been used to achieve similar goals (for example, predicting future outcomes), can lead to model development. This happens when it draws general conclusions based on specific observations. For this aim an inductive approach can be applied.

Moreover, the study proposes a specific model architecture and training strategy based on the understanding of robot actions, sensor data, and the capabilities of generative models, Then, the model will be tested on real-world robot tasks to see if it effectively predicts future outcomes. For developing and training the generative model abductive approach can be applied due to using existing knowledge to propose explanations for observations and then testing those explanations.

However, deductive approach is not appropriate in this study. Because while existing knowledge is engaged in generative model, the research isn't just about applying general rules to a new situation. Element of innovations and explorations are involved in creating a model specifically for robot action simulation.

Consequently, the dissertation involves mixed approaches. First using induction to learn from existing research and then applying abduction to test and evaluate the generative model for action simulation in robots. This combination can leverage existing knowledge while also introducing a novel application of generative models in robotics.

For the dissertation, some potential research strategies and methodologies could be considered. Each approach offers unique advantages and can contribute differently to achieving the research aims. In the following, some potential choices and their suitability is discussed:

i. Experiment: This is the most favorable option about impact of the model on adaptive and decision-making abilities could be assessed by gauging their performance against similar tasks accomplished by robots lacking the mode function. It could be valuable when it comes to testing generative model's effectiveness in predicting outcomes. Furthermore, experiments can have the highest control over their variables possible and provide stronger empirical evidence for quantitative analyses of results to support the research findings.
ii. Case study: Applying a case study for this research is useful for getting a better understanding of how generative models are integrated into robots. Specific robots can be selected as case studies for investigation and implementation of integrating generative models for predicting future action. With the aid of case study, collection of rich quantitative data through interview, observation or document analysis could be achievable.
iii. Action Research: Designing, executing, and evaluating generative models in robots to aim overcome practical problems and enhance results can be considered with using this method. Also, it can be beneficial if the research would be collaborated by stakeholders in industries like robotics and researchers.

iv. Simulation study: In this case, simulations method can be used to test models in a virtual safe and controlled environment. It could be defined different scenarios and analyzed of how prediction of future actions could be. Simulation study is more flexible in evaluation parameters in different conditions and potentially it is enabled to modification algorithms correctly.

The dissertation is kind of interdisciplinary matters, a combination of strategies is the most appropriate for it. For instance, experimental study could be used in the beginning to evaluate the operation of generative models in controlled adjustments, then study case could come in to consider the implementation in real-world and user experience, or simulation study could be applied for test it in a simulated environment and eventually it could be involved with action research for modification and higher efficiency in generative models. With the aid of this mixed- method the study could cover different sides of the research and get a comprehensive outcome.

4.2 Tools and Techniques

In the following, there are some potential tools, techniques, models, software, architecture, development environments, and equipment for engineering models for robot action simulation:

Generative Models: Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) are the primary choices considering their strengths and weaknesses. For instance, VAEs models are efficient for data representation, while GANs are better in image generation.

Software frameworks: Pytorch, Tensorflow or Keras could be applicable for implementation, training and evaluating generative models. These frameworks have pre-designed properties for creating neural networks and can simplify the whole development process.

Robotics Simulation Software: For creating virtual and simulated environments for testing the models on robots, Gazebo, V-REP, ROS (Robot Operation System) and Webots can be used. Thess applications can design virtual robots, sensors, and environment to follow real-world tasks.

Robot Control Systems: Specific control system or software development kit (SDK) might be required based on the type of robots which are applied to the project.

Data Preprocessing Libraries: For formatting, cleaning, and pre-processing the data before passing through a generative model for training, NumPy, Pandas, or scikit-learn in Python can be used.

Development Environment: Visual Studio Code, PyCharm, or Jupyter Notebooks are options as comfortable environment for coding and working which the frameworks and libraries are supported in them.

Computing Hardware: Depending on the complexity of the models and the amount of data that will be used, a powerful computer enhanced with powerful GPU (Graphic Process Units) might be required. Also cloud computing like Azure might be considered.

Visualization Tools: For visualizing the data generated by models and perceptions of its learning process visualization tools like TensorBoard or Matplotlib can be used.

4.3 Data Collection and Analysis Decisions

Although depending on the specific analyses and methods used for studies, qualitative factors could be applied like interview or observation for understanding human mental imagery, the research is aligned with quantitative approach. Due to focusing on engineering models that is mostly engaged with mathematical for designing models. Moreover, robots which are mentioned illustrate technical and measurable aspects to the research.

4.3.1 Data Gathering Technique

The primary source of data for the research can be collected by robot's sensors like cameras, Light Detection and Ranging (LiDAR), sonars and other ones in the time of experiments or simulations. Depending on robots and sensors there are specific tools for gathering data. Some of them have built-in logging capabilities while another requires specific software to read the data.

4.3.2 Data Analysis Techniques

Collected data by sensor should be cleaned and pre-processed before feeding them to models. This task includes:

i. Remove noises and imbalance data.
ii. Standardize data into a specific format.
iii. Extract specific features from data which help models to learn efficiently. Python libraries like Pandas, Numpy or scikit-learn can be useful for doing that.

5. ETHICS, RISKS, AND ISSUES

5.1 Ethics and Legal Issues

Bias and Fairness: Trained generative models with bias dataset effect on their bias prediction which leads to bias decision making and have negative consequences.

Safety and Control: Robots integrated with simulation actions are needed to be strongly safety to avoid injuring humans.

Transparency and Explainability: This is so important and crucial to knowing about how generative models predict generative data. If the whole process is unclear and unknown (black box) there are concerns about the probability of its unpredictability consequences.

Data Privacy: If the research engaged with personal data which is collected by robot sensors, there are concerns about privacy and consent.

Job displacement: Social and economic affect be considered potentially due to this integration; robots can be more expert in decision-making and doing future tasks.

5.2 Risks

Model Performance: Developed generative models (VAE or GAN) might not get appropriate and efficiency level of accuracy in prediction. To handle this risk pilot testing and iterative development must be applied. It should be started with a simpler model, different architectures and methods and the model must be examined and modified frequently.

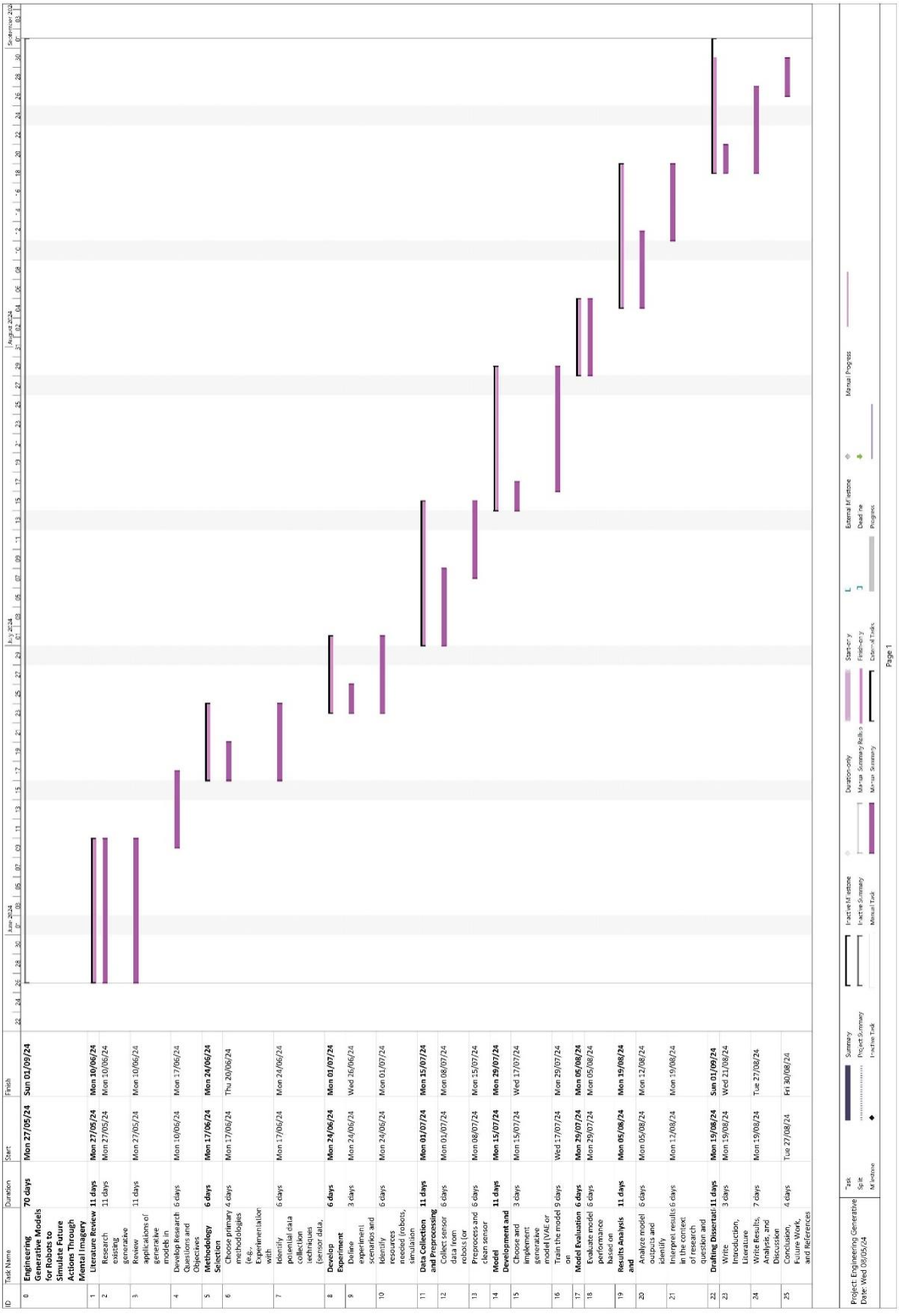
Data Quality and Collection: the quality and quantity of sensors data collection that should be used for training models can considerably affect its performance. For tackle this issue data Augmentation and quality control techniques should be considered.

Integration Challenges: It might be challenging to integrate generative models with robotic control systems due to compatibility or the limitation in robotic platforms. To address this issue standardization and modular design could be applied.

Risk	Potential Impact	Mitigation Strategy
Model Performance	Generative models may not achieve the required level of accuracy, leading to poor predictions.	Start with a simpler model, conduct frequent pilot testing, iteratively develop and improve the model based on results.
Data Quality and Collection	Low-quality or imbalanced data may lead to inaccurate model training and poor performance.	Ensure proper data augmentation, clean and preprocess the data, and apply quality control techniques to improve data quality.
Integration Challenges	Difficulty in integrating generative models with robotic systems due to compatibility or technical constraints.	Use modular design for compatibility and standardization, ensure testing across multiple platforms to confirm integration.

Table 2: Summary of Risks, Potential Impacts, and Mitigation Strategies in the Research Project

6. TIME PLAN


ID
Task Name
Duration
Start
Finish
Engineering Generative Models for Robots to Simulate Future Actions Through Mental Imagery 70 days Mon 27/05/24 Sun 01/09/24
Literature Review 11 days Mon 27/05/24 Mon 10/06/24
Research existing generative models 11 days Mon 27/05/24 Mon 10/06/24
Review applications of predictive models in 11 days Mon 27/05/24 Mon 10/06/24
Develop Research Questions and Objectives 6 days Mon 10/06/24 Mon 17/06/24
Methodology Selection 6 days Mon 17/06/24 Mon 24/06/24
Choose primary methodologies (e.g., Experimentation with 4 days Mon 17/06/24 Thu 20/06/24
Identify potential data collection techniques (sensor data, 6 days Mon 17/06/24 Mon 24/06/24
Develop Experiment 6 days Mon 24/06/24 Mon 01/07/24
Define experiment scenarios and 3 days Mon 24/06/24 Wed 26/06/24
Identify resources needed (robots, simulation 6 days Mon 24/06/24 Mon 01/07/24
Data Collection and Preprocessing 11 days Mon 01/07/24 Mon 15/07/24
Collect sensor data from robots (or 6 days Mon 01/07/24 Mon 08/07/24
Preprocess and clean sensor 6 days Mon 08/07/24 Mon 15/07/24
Model Development and 11 days Mon 15/07/24 Mon 29/07/24
Choose and implement generative model (VAE or 3 days Mon 15/07/24 Wed 17/07/24
Train the model on 9 days Wed 17/07/24 Mon 29/07/24
Model Evaluation 6 days Mon 29/07/24 Mon 05/08/24
Evaluate model performance based on 6 days Mon 29/07/24 Mon 05/08/24
Results Analysis and 11 days Mon 05/08/24 Mon 19/08/24
Analyze model outputs and identify 6 days Mon 05/08/24 Mon 12/08/24
Interpret results in the context of research question and 6 days Mon 12/08/24 Mon 19/08/24
Drafting Dissertati 11 days Mon 19/08/24 Sun 01/09/24
Write Introduction, Literature 3 days Mon 19/08/24 Wed 21/08/24
Write Results, Analysis, and Discussion 7 days Mon 19/08/24 Tue 27/08/24
Conclusion, Future Work, and References 4 days Tue 27/08/24 Fri 30/08/24
May 2024
June 2024
July 2024
August 2024
September 2024
Project: Engineering Generative
Date: Wed 08/05/24
Task
Split
Milestone
Summary
Project Summary
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
Deadline
Progress
External Milestone
Manual Progress
Page 1

7.0 References

Bütepage, J., Ghadirzadeh, A., Öztimur Karadağ, Ö., Björkman, M., & Kragic, D. (2020). Imitating by generating: Deep generative models for imitation of interactive tasks. *Frontiers in Robotics and AI, 7*, 47.

Chaudhuri, S., Ritchie, D., Wu, J., Xu, K., & Zhang, H. (2020). Learning generative models of 3D structures. In *Computer Graphics Forum* (Vol. 39, pp. 643-666). Wiley Online Library.

Di Nuovo, A. G., Marocco, D., Di Nuovo, S., & Cangelosi, A. (2013). Autonomous learning in humanoid robotics through mental imagery. *Neural Networks,, 41*, 147-155.

Di Nuovo, A., De La Cruz, V. M., & Marocco, D. (2013). Special issue on artificial mental imagery in cognitive systems and robotics. *Adaptive Behavior, 21*, 217-221.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM,, 63*, 139-144.

Hoffmann, M., Marques, H., Arieta, A., Sumioka, H., Lungarella, M., & Pfeifer, R. (2010). Body schema in robotics: a review. *IEEE Transactions on Autonomous Mental Development,, 2*, 304-324.

Kasderidis, S. (2008). An executive system for cognitive agents. . In *In 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1866-1871). IEEE.

Kosslyn, S. M. (2005). Mental images and the Brain. *Cognitive neuropsychology*, 22(3-4), 333-347.

Lamb, A. (2021). A brief introduction to generative models. . *arXiv preprint arXiv:2103.00265.*

Mohan, V., Morasso, P., Metta, G., & Kasderidis, S. (2010). Actions and imagined actions in cognitive robots. In *Perception-Action Cycle: Models, Architectures, and Hardware* (pp. 539-572). New York, NY: Springer.

Pronobis, A., & Rao, R. P. (2017). Learning deep generative spatial models for mobile robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 755-762). IEEE.

Pylyshyn, Z. W. (2002). Mental imagery: In search of a theory. *Behavioral and brain sciences*, 25(2), 157-182.

Seepanomwan, K. (2019). How mental imagery helps robot learning. I. In K. Seepanomwan, *n 2019 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engine* (pp. 245-250). IEEE.

Wu, L. L., & Barsalou, L. W. (2009). Perceptual simulation in conceptual combination: Evidence from property generation. *Acta psychologica*, 132(2), 173-189.

APPENDIX B: COMPLETED RESEARCH ETHICS CHECKLIST

Sheffield Hallam University

UREC 1 RESEARCH ETHICS REVIEW FOR STUDENT RESEARCH WITH NO HUMAN PARTICIPANTS OR DIRECT COLLECTION OF HUMAN TISSUES, OR BODILY FLUIDS.

All University research is required to undergo ethical scrutiny to comply with UK law. The University Research Ethics Policy (www.shu.ac.uk/research/excellence/ethics-and-integrity/policies) should be consulted before completing this form. The initial questions are there to check that completion of the UREC1 is appropriate for this study. The supervisor will approve the study, but it may also be reviewed by the College Teaching Program Research Ethics Committee (CTPREC) as part of the quality assurance process (additional guidance can be obtained from your College Research Ethics Chair[1]).

The final responsibility for ensuring that ethical research practices are followed rests with the supervisor for student research.

Note that students and staff are responsible for making suitable arrangements to ensure compliance with the General Data Protection Regulations (GDPR), for keeping data secure and if relevant, for keeping the identity of participants anonymous. They are also responsible for following SHU guidelines about data encryption and research data management. Guidance can be found on the SHU Ethics Website www.shu.ac.uk/research/excellence/ethics-and-integrity

Please note that it is mandatory for all students to only store data on their allotted networked F drive space and not on individual hard drives or memory sticks etc.

This form also enables the University and College to keep a record confirming that research conducted has been subjected to ethical scrutiny. Students should retain a copy for inclusion in their research projects, and a copy should be uploaded to the relevant module Blackboard site.

The form must be completed by the student and approved by supervisor and/or module leader (as applicable). In all cases, it should be counter-signed by the supervisor and/or module leader and kept as a record showing that ethical scrutiny has occurred. Students should retain a copy for inclusion in the appendices of their research projects, and a copy should be uploaded to the module Blackboard site for checking.

Please note that it may be necessary to conduct a health and safety risk assessment for the proposed research. Further information can be obtained from the University's Health and Safety Website https://sheffieldhallam.sharepoint.com/sites/3069/SitePages/Risk-Assessment.aspx

[1] College of Social Sciences and Arts – Dr. Antonia Ypsilanti (a.ypsilanti@shu.ac.uk)
College of Business, Technology and Engineering – Dr. Tony Lynn (t.lynn@shu.ac.uk)
College of Health, Wellbeing and Life Sciences – Dr. Nikki Jordan-Mahy (n.jordan-mahy@shu.ac.uk)

ARE YOU COMPLETING THE CORRECT FORM?

Does this study include collecting data or samples from human participants. YES/NO

Is the secondary data used in this study of a sensitive or contentious nature, or does it allow the identification of individuals or organisations (e.g., companies, school, councils, communities). YES/NO

If you have answered ***YES*** to either of these two questions you must complete a UREC2, 3 or 4 as appropriate.

1. General Details

Details	
Name of student	Siavash Mortaz Hejri
SHU email address	c2056757@hallam.shu.ac.uk
Department/College	Computing
Name of supervisor	Hamed Pourfannan, Alejandro Jimenez Rodriguez
Supervisor's email address	h.pourfannan@shu.ac.uk, a.jimenez-
Title of proposed research	Using generative AI to synthesizing human hand actions based on object affordances for robots
Proposed start date	27th May 2024
Proposed end date	1st September 2024
Brief outline of research to include, rationale (reasons) for undertaking the research & aims, and methods (max 500 words).	**Rationale for Undertaking the Research** Nowadays, robots have become an inseparable part of our daily life. Recently, their potential abilities to navigate real-world environments and perform meaningful tasks require more than just being a mechanical machine. Robots are designed and programmed based on pre-defined methods traditionally which often lack the flexibility to handle dynamic in some kinds of real-time situations. Moreover, in these situations updating their software with complex scenario is a high-cost and time-consuming job for expertise. Due to these limitations, researchers have started to explore and use mental imaginary in robots. Generative AI models can be trained to create a realistic simulation of their future actions. Integrating with robots, they can act more flexibility and when it comes to making decisions, they can do it efficiently. **Research Question and Aim**

Details	
	How can human hand actions be synthesized based on object affordances using generative AI? By addressing this question, the research aims to bridge the gap between static, pre-defined hand robotic actions and dynamic, flexible behaviors that adapt to real-time situations, thus enhancing the utility and efficiency of robots in everyday tasks. The intention of this research work is to investigate and improve on the generative models specifically, Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) that will aid integration with robots to promote generative future hand action actions based on object affordances for robots resembling mental imagery in humans. **Methods** The research will involve several key steps: Literature Review: Conduct a comprehensive review of existing work on generative AI models, mental imagery in robots, and object affordances. This will provide a solid theoretical foundation and identify gaps in current research. Model Development: Develop and refine Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) to create realistic simulations of human hand actions. This involves designing the architecture of these models and implementing necessary modifications to enhance their performance. Dataset Collection and Preparation: Finding secondhand datasets that include a variety of human hand actions and corresponding object affordances. This step includes data preprocessing, such as normalizing data, to ensure it is suitable for training the models. Model Training: Train the VAEs and GANs using the prepared datasets. This will involve multiple iterations and adjustments to optimize the models' ability to generate accurate and realistic hand action simulations. Simulation and Testing: Test the trained models in a simulated environment to evaluate their performance. This involves creating virtual scenarios where the models can generate hand

Details	
	actions based on different object affordances and assessing the accuracy and realism of these actions. Evaluation and Analysis: Conduct a thorough evaluation of the models' performance using qualitative and quantitative metrics. This may include measures such as the realism of generated actions, the accuracy of action-object affordance matching, and the models' ability to generalize to new scenarios.

2. Research in External Organisations

Question	Yes/No
1. Will the research involve working with/within an external organisation (e.g., school, business, charity, museum, government department, international agency, etc.)?	NO
2. If you answered YES to question 1, do you have granted access to conduct the research? *If YES, students please show evidence to your supervisor. PI should retain safely.*	–
3. If you answered NO to question 2, is it because: A. you have not yet asked B. you have asked and not yet received an answer C. you have asked and been refused access.	–

3. Research with Products and Artefacts

Question	Yes/No
1. Will the research involve the use of specialist copyrighted documents, films, broadcasts, photographs, artworks, designs, products, programs, databases, networks, processes, existing	Yes

Question	Yes/No
2. If you answered YES to question 1, are the materials you intend to use in the public domain? *Notes: 'In the public domain' does not mean the same thing as 'publicly accessible'.* • *Information which is 'in the public domain' is no longer protected by copyright (i.e., copyright has either expired or been waived) and can be used without permission.* • *Information which is 'publicly accessible' (e.g., TV broadcasts, websites, artworks, newspapers) is available for anyone to consult/view. It is still protected by copyright even if there is no copyright notice. In UK law, copyright protection is automatic and does not require a copyright statement, although it is always good practice to provide one. It is necessary to check the terms and conditions of use to find out exactly how the material may be reused etc.* *If you answered YES to question 1, be aware that you may need to consider other ethics codes. For example, when conducting Internet research, consult the code of the Association of Internet Researchers; for*	No
3. If you answered NO to question 2, do you have explicit permission to use these materials as data? *If YES, please show evidence to your supervisor.*	Yes
4. If you answered NO to question 3, is it because: A. you have not yet asked permission B. you have asked and not yet received and answer C. you have asked and been refused access. *Note: You will only be able to start the research when you have been*	**A/B/C**

4. Does this research project require a health and safety risk assessment for the procedures to be used? (Discuss this with your supervisor)

☐ Yes
☒ No

If **YES** the completed Health and Safety Risk Assessment form should be attached. A standard risk assessment form can be generated through the Awaken system (https://shu.awaken-be.com). Alternatively if you require more specific risk assessment, e.g. a COSHH, attach that instead.

Insurance Check

The University's standard insurance cover will not automatically cover research involving any of the following:

i) Participants under 5 years old
ii) Pregnant women
iii) 5000 or more participants
iv) Research being conducted in an overseas country
v) Research involving aircraft and offshore oil rigs
vi) Nuclear research
vii) Any trials/medical research into Covid 19

If your proposals do involve any of the above, please contact the Insurance Manager directly (fin-insurancequeries-mb@exchange.shu.ac.uk) to discuss this element of your project.

Adherence to SHU Policy and Procedures

<table>
<tr><td colspan="2">Ethics sign-off</td></tr>
<tr><td colspan="2">Personal statement</td></tr>
<tr><td colspan="2">I can confirm that:
I have read the Sheffield Hallam University Research Ethics Policy and Procedures
I agree to abide by its principles.</td></tr>
<tr><td colspan="2">Student</td></tr>
<tr><td>Name: Siavash Mortaz Hejri</td><td>Date: 17/06/2024</td></tr>
<tr><td colspan="2">Signature: Siavash Mortaz Hejri</td></tr>
<tr><td colspan="2">Supervisor ethical sign-off</td></tr>
<tr><td colspan="2">I can confirm that completion of this form has confirmed that this research does not involve human participants. The research will not commence until any approvals required under Sections 2 & 3 have been received and any health and</td></tr>
<tr><td>Name: Alejandro Jimenez Rodriguez</td><td>Date: 19/06/20</td></tr>
<tr><td colspan="2">Signature:
  </td></tr>
</table>

Ethics sign-off	
Independent Reviewer ethical sign off (if required to permit publication of findings with supervisor co-authorship).	
Name:	Date:
Signature:	

Please ensure that you have attached all relevant documents. Your supervisor must approve them before you start data collection:

Relevant Documents	Yes	No	N/A
Research proposal if prepared previously	☒	☐	☐
Any associated materials (e.g., posters, letters, etc.)	☒	☐	☐
Health and Safety Risk Assessment Form	☐	☐	☒

APPENDIX C: PUBLICATION PROCEDURE FORM

Sheffield Hallam University

College of Business, Technology and Engineering

Dissertation for Computing (55-708541).

PUBLICATION PROCEDURE FORM

In this module, while you create your own research question or topic area, your supervisor makes a significant intellectual contribution to this work as the research progresses. Your supervisor will make the decision on whether your work merits publication based on the quality of the work you have produced. Your supervisor will co-author the paper for publication with you and your supervisor will both be listed as authors. You are required to sign the declaration below to confirm that you understand and will follow this procedure.

Declaration:

I, Siavash Mortaz Hejri confirm that I understand will comply with the Publication Procedure outlined in the Module Handbook and the Blackboard Site.		
Student: Siavash Mortaz Hejri	Signature 	Date 28/08/2024
Supervisor: Dr. Alejandro Jimenez Rodriguez, Dr. Hamed Pourfannan	Signature 	Date 05/09/2024

APPENDIX D: Source Code

a_Extract_Data.py

```python
import os
import pickle
import numpy as np
from tqdm import tqdm

#------load_ho3d_best_info FUNCTION-----
"""
It reads sequence information from text files,
extracts relevant annotations from corresponding pickle files,
and then saves the processed data into new pickle files.
"""
"""
The progress of loading the data is displayed using a progress bar,
and finally print the count of loaded entries.
"""
"""
data_dir=Directory where the dataset is stored.
split= Specifies the dataset split to be loaded, default is 'train'
"""
1 usage
def load_ho3d_best_info(data_dir, split='train'):
    hand_poses = []
    object_infos = []

    split_file = os.path.join(data_dir, f'{split}.txt')
    with open(split_file, 'r') as f:
        sequences = f.readlines()

    # Initialize the progress bar
    total_sequences = len(sequences)
    with tqdm(total=total_sequences, desc="Loading data", unit="sequence") as pbar:
        for sequence in sequences:
            seq_name, file_id = sequence.strip().split('/')
            meta_file = os.path.join(
                data_dir+r'\train', seq_name, 'meta', f'{file_id}.pkl')

            # Load annotations
            with open(meta_file, 'rb') as mf:
                annotations = pickle.load(mf)

            if annotations['handPose'] is not None and annotations['objTrans'] is not None:
                hand_poses.append({
                    'handPose': annotations['handPose'],
                    'handTrans': annotations['handTrans'],
                    'handJoints3D': annotations['handJoints3D']
                })

                object_infos.append({
                    'objTrans': annotations['objTrans'],
                    'objRot': annotations['objRot'],
                    'objName': annotations['objName'],
                    'objLabel': annotations['objLabel'],
```

```
                    'objCorners3D': annotations['objCorners3D']
                })

            # Update the progress bar
            pbar.update(1)

    return hand_poses, object_infos

# Load data
data_dir = r'D:\UNI\Sem3\Dissertation\My effort\HOnnotate\ho3d-master\Dataset\HO3D_v3'
hand_poses, object_infos = load_ho3d_best_info(data_dir, split='train')

# Save hand data into hand_poses.pkl
# Save object data into object_infos.pkl
hand_poses_file = 'hand_poses.pkl'
object_infos_file = 'object_infos.pkl'

with open(hand_poses_file, 'wb') as hp_file:
    pickle.dump(hand_poses, hp_file)

with open(object_infos_file, 'wb') as oi_file:
    pickle.dump(object_infos, oi_file)
print("Data saved successfully!")
print("Data loaded successfully!")
print("Number of hand poses loaded:", len(hand_poses))
print("Number of object infos loaded:", len(object_infos))

```

c_Preproccesing.py

```python
import pickle
import os
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from b_Load_ExtractedData import load_saved_data


#============== LOADING DATA ============================
hand_poses_file = r'hand_poses.pkl'
object_infos_file = r'object_infos.pkl'
hand_poses, object_infos = load_saved_data(hand_poses_file, object_infos_file)


#============== Reshape objRot IF NECESSARY ==============
num_objRot_reshaped = 0
for i in range(len(object_infos)):
    if object_infos[i]['objRot'].shape != (3,):
        object_infos[i]['objRot'] = object_infos[i]['objRot'].reshape(3)
        num_objRot_reshaped += 1
print(f'The number of reshaped: {num_objRot_reshaped}')


#============== PREPROCESSING & PREPARING  ============================
"""
It extracts, normalizes, and combines the necessary features.
It splits the data into appropriate sets for training, validation, and testing.
Finally, it saves the processed data and the normalization parameters to disk for later use.
"""
1 usage
def preprocess_data(hand_poses, object_infos):
    # Extract hand pose information
    hand_pose_data = np.array([pose['handPose'] for pose in hand_poses])
    hand_trans_data = np.array([pose['handTrans'] for pose in hand_poses])
    hand_joints_data = np.array([pose['handJoints3D'] for pose in hand_poses])

    # Extract object information
    obj_trans_data = np.array([info['objTrans'] for info in object_infos])
    obj_rot_data = np.array([info['objRot'] for info in object_infos])
    obj_corners_data = np.array([info['objCorners3D'] for info in object_infos])

    # Extract object names (this will not be used for training)
    obj_names = [info['objName'] for info in object_infos]

    # Data is normalized using StandardScaler from sklearn,
    # which standardizes features by removing the mean and scaling to unit variance.
    # Normalize hand joints and object corners
    # Hand joints and object corners (reshaped for 3D data).
    # Hand pose, hand translation, object translation, and object rotation.

    scaler_hand_joints = StandardScaler()
    hand_joints_data = scaler_hand_joints.fit_transform(hand_joints_data.reshape(-1, 63)).reshape(-1, 21, 3)

    scaler_obj_corners = StandardScaler()
    obj_corners_data = scaler_obj_corners.fit_transform(obj_corners_data.reshape(-1, 24)).reshape(-1, 8, 3)

    # Normalize other parameters
    scaler_hand_pose = StandardScaler()
    hand_pose_data = scaler_hand_pose.fit_transform(hand_pose_data)

    scaler_hand_trans = StandardScaler()
    hand_trans_data = scaler_hand_trans.fit_transform(hand_trans_data)

    scaler_obj_trans = StandardScaler()
    obj_trans_data = scaler_obj_trans.fit_transform(obj_trans_data)
```

```
    scaler_obj_rot = StandardScaler()
    obj_rot_data = scaler_obj_rot.fit_transform(obj_rot_data)

    # Combine data
    # Hand data is combined into a single array, consisting of hand pose, translation, and joints data.
    # Object data is similarly combined into a single array, consisting of object translation, rotation, and corner data
    hand_data = np.hstack((hand_pose_data, hand_trans_data, hand_joints_data.reshape(-1, 63)))
    obj_data = np.hstack((obj_trans_data, obj_rot_data, obj_corners_data.reshape(-1, 24)))

    return hand_data, obj_data, obj_names, scaler_hand_joints, scaler_obj_corners, scaler_hand_pose, \
        scaler_hand_trans, scaler_obj_trans, scaler_obj_rot

# Preprocess the data
hand_data, obj_data, obj_names, scaler_hand_joints, scaler_obj_corners, scaler_hand_pose, scaler_hand_trans, \
    scaler_obj_trans, scaler_obj_rot = preprocess_data(hand_poses, object_infos)

#============== SPLITING DATA  ==========================
"""
Split data into training, validation, and test sets
20% of the data is set aside for testing
The remaining 80% is split again, with 25% of it used for validation
(25% of 80% is 0.25 * 0.8 = 0.2, or 20% of the original dataset),
leaving 60% of the original data for training
"""
hand_train, hand_test, obj_train, obj_test = train_test_split(hand_data, obj_data, test_size=0.2, random_state=42)
hand_train, hand_val, obj_train, obj_val = train_test_split(hand_train, obj_train, test_size=0.25, random_state=42)

print("Training data shape:", hand_train.shape, obj_train.shape)
print("Validation data shape:", hand_val.shape, obj_val.shape)
print("Test data shape:", hand_test.shape, obj_test.shape)

#============== SAVE DATA AND SCALERS =============
data_files = {
    'hand_train': hand_train,
    'hand_val': hand_val,
    'hand_test': hand_test,
    'obj_train': obj_train,
    'obj_val': obj_val,
    'obj_test': obj_test,
    'obj_names': obj_names  # Save the object names separately
}

scalers = {
    'scaler_hand_joints': scaler_hand_joints,
    'scaler_obj_corners': scaler_obj_corners,
    'scaler_hand_pose': scaler_hand_pose,
    'scaler_hand_trans': scaler_hand_trans,
    'scaler_obj_trans': scaler_obj_trans,
    'scaler_obj_rot': scaler_obj_rot
}

# Save data into hand_object_data.pkl
with open('hand_object_data.pkl', 'wb') as data_file:
    pickle.dump(data_files, data_file)

# Save scalers into scalers.pkl
with open('scalers.pkl', 'wb') as scalers_file:
    pickle.dump(scalers, scalers_file)

print("Data and scalers saved successfully!")
```

loading_data.py

```python
import torch
import pickle
from torch.utils.data import DataLoader, TensorDataset

#====================== loading_data FUNCTION =================
"""
It loads preprocessed data from the pickle file (hand_object_data.pkl),
converts it into PyTorch tensors,
and prepares it for use in a machine learning model by organizing it into datasets and data loaders,
which are then returned for further use in model training, validation, and testing.
"""

"""
hand_object_data_path = path of the preprocessed data pickle file
batch_size= Specifies how many samples per batch to load.
"""
def loading_data(hand_object_data_path, batch_size):
    # LOAD PREPROCESSED DATA Load
    with open(hand_object_data_path, 'rb') as data_file:
        data_files = pickle.load(data_file)

    hand_train = torch.tensor(data_files['hand_train'], dtype=torch.float32)
    hand_val = torch.tensor(data_files['hand_val'], dtype=torch.float32)
    hand_test = torch.tensor(data_files['hand_test'], dtype=torch.float32)
    obj_train = torch.tensor(data_files['obj_train'], dtype=torch.float32)
    obj_val = torch.tensor(data_files['obj_val'], dtype=torch.float32)
    obj_test = torch.tensor(data_files['obj_test'], dtype=torch.float32)
    obj_names = data_files['obj_names']

    # COMBINATION DATA
    # Combines the hand and object tensors for each dataset
    # (train, validation, test) into a single dataset object
    # Each element in these datasets will consist of a pair (hand_data, obj_data)
    train_dataset = TensorDataset(hand_train, obj_train)
    val_dataset = TensorDataset(hand_val, obj_val)
    test_dataset = TensorDataset(hand_test, obj_test)

    batch_size = batch_size

    #CREATE DATALOADER
    # Creates data loaders for the train, validation, and test datasets
    # shuffle=True =>  Randomly shuffles the data in the training set at every epoch
    # to prevent the model from learning the order of the data
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

    # shuffle=False => For validation and test datasets, data is not shuffled
    # since shuffling is not usually necessary or desirable for these sets.
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    print('Data is loaded successfully')

    return hand_train, hand_val, hand_test, obj_train, obj_val, obj_test, train_dataset, val_dataset, test_dataset, \
        train_loader, val_loader, test_loader, obj_names

```

all_models.py

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

#=================== MODEL 01 - CVAE_01 ==============================
"""
WITH TOTAL 3 LAYERS FOR EACH ENCODER AND DECODER
2 LINEAR LAYERS AND 1 RELU LAYER FOR EACH ENCODER AND DECODER
"""
5 usages
class CVAE_01(nn.Module):
    def __init__(self, input_dim, latent_dim, condition_dim):
        super(CVAE_01, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim + condition_dim, 256),
            nn.ReLU(),
            nn.Linear(256, latent_dim * 2)  # Mean and log-variance
        )
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim + condition_dim, 256),
            nn.ReLU(),
            nn.Linear(256, input_dim)
        )

    1 usage
    def encode(self, x, c):
        combined = torch.cat([x, c], dim=-1)
        h = self.encoder(combined)
        mean, log_var = torch.chunk(h, 2, dim=-1)
        return mean, log_var

    2 usages
    def decode(self, z, c):
        combined = torch.cat([z, c], dim=-1)
        return self.decoder(combined)

    1 usage
    def reparameterize(self, mean, log_var):
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        return mean + eps * std

    def forward(self, x, c):
        mean, log_var = self.encode(x, c)
        z = self.reparameterize(mean, log_var)
        return self.decode(z, c), mean, log_var
#-------------------------------------------------------------------------------


#=================== MODEL 02 - CVAE_02 ==============================
"""
ADD ONE MORE LAYER
WITH TOTAL 5 LAYERS FOR EACH ENCODER AND DECODER
3 LINEAR LAYERS AND 2 RELU LAYERS FOR EACH ENCODER AND DECODER
"""
13 usages
class CVAE_02(nn.Module):
    def __init__(self, input_dim, latent_dim, condition_dim):
        super(CVAE_02, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim + condition_dim, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, latent_dim * 2)  # Mean and log-variance
```

```
        )
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim + condition_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, input_dim)
        )

    1 usage
    def encode(self, x, c):
        combined = torch.cat([x, c], dim=-1)
        h = self.encoder(combined)
        mean, log_var = torch.chunk(h, 2, dim=-1)
        return mean, log_var

    4 usages
    def decode(self, z, c):
        combined = torch.cat([z, c], dim=-1)
        return self.decoder(combined)

    1 usage
    def reparameterize(self, mean, log_var):
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        return mean + eps * std

    def forward(self, x, c):
        mean, log_var = self.encode(x, c)
        z = self.reparameterize(mean, log_var)
        return self.decode(z, c), mean, log_var
#--------------------------------------------------------------------------

#================== MODEL 03 - CVAE_03 ==============================
"""
ADD DROPOUT LAYER (dropout layers can help prevent overfitting)
TOTAL 9 LAYERS FOR EACH ENCODER AND DECODER
3 LINEAR LAYERS, 2 BATCH NORMALIZATION LAYERS, 2 RELU LAYERS, AND 2 DROUPOUT LAYERS FOR EACH ENCODER AND DECODER
"""
7 usages
class CVAE_03(nn.Module):
    def __init__(self, input_dim, latent_dim, condition_dim):
        super(CVAE_03, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim + condition_dim, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, latent_dim * 2)  # Mean and log-variance
        )
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim + condition_dim, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, input_dim)
        )

```

```
    1 usage
    def encode(self, x, c):
        combined = torch.cat([x, c], dim=-1)
        h = self.encoder(combined)
        mean, log_var = torch.chunk(h, 2, dim=-1)
        return mean, log_var

    2 usages
    def decode(self, z, c):
        combined = torch.cat([z, c], dim=-1)
        return self.decoder(combined)

    1 usage
    def reparameterize(self, mean, log_var):
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        return mean + eps * std

    def forward(self, x, c):
        mean, log_var = self.encode(x, c)
        z = self.reparameterize(mean, log_var)
        return self.decode(z, c), mean, log_var
#--------------------------------------------------------------------------

#=================== MODEL 04 - CVAE_02_1 ==============================
"""
ADD MORE LAYERS TO CVAE_02
WITH TOTAL 7 LAYERS FOR EACH ENCODER AND DECODER
4 LINEAR LAYERS AND 3 RELU LAYERS FOR EACH ENCODER AND DECODER
"""
9 usages
class CVAE_02_1(nn.Module):
    def __init__(self, input_dim, latent_dim, condition_dim):
        super(CVAE_02_1, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim + condition_dim, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, latent_dim * 2)  # Mean and log-variance
        )
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim + condition_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, input_dim)
        )

    1 usage
    def encode(self, x, c):
        combined = torch.cat([x, c], dim=-1)
        h = self.encoder(combined)
        mean, log_var = torch.chunk(h, 2, dim=-1)
        return mean, log_var

    4 usages
    def decode(self, z, c):
        combined = torch.cat([z, c], dim=-1)
        return self.decoder(combined)

```

```
    1 usage
    def reparameterize(self, mean, log_var):
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        return mean + eps * std

    def forward(self, x, c):
        mean, log_var = self.encode(x, c)
        z = self.reparameterize(mean, log_var)
        return self.decode(z, c), mean, log_var
#--------------------------------------------------------------------------

#=================== MODEL 05 - CVAE_02_2 ==============================
"""
ADD MORE LAYERS TO CVAE_02_1
WITH TOTAL 9 LAYERS FOR EACH ENCODER AND DECODER
5 LINEAR LAYERS AND 4 RELU LAYERS FOR EACH ENCODER AND DECODER
"""
10 usages
class CVAE_02_2(nn.Module):
    def __init__(self, input_dim, latent_dim, condition_dim):
        super(CVAE_02_2, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim + condition_dim, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, latent_dim * 2)  # Mean and log-variance
        )
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim + condition_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, input_dim)
        )

    1 usage
    def encode(self, x, c):
        combined = torch.cat([x, c], dim=-1)
        h = self.encoder(combined)
        mean, log_var = torch.chunk(h, 2, dim=-1)
        return mean, log_var

    3 usages
    def decode(self, z, c):
        combined = torch.cat([z, c], dim=-1)
        return self.decoder(combined)

    1 usage
    def reparameterize(self, mean, log_var):
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        return mean + eps * std

    def forward(self, x, c):
        mean, log_var = self.encode(x, c)
        z = self.reparameterize(mean, log_var)
```

```python
        return self.decode(z, c), mean, log_var

#--------------------------------------------------------------------------

#=================== MODEL 06 - CVAE_02_3 ============================
"""
REMOVE OBJECT (CONDITION) FROM ENCODER IN CVAE02_2
WITH TOTAL 9 LAYERS FOR EACH ENCODER AND DECODER
5 LINEAR LAYERS AND 4 RELU LAYERS FOR EACH ENCODER AND DECODER
"""
10 usages
class CVAE_02_3(nn.Module):
    def __init__(self, input_dim, latent_dim, condition_dim):
        super(CVAE_02_3, self).__init__()
        # Encoder: Takes only the hand pose as input
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, latent_dim * 2)  # Mean and log-variance
        )
        # Decoder: Takes latent vector and condition as input
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim + condition_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, input_dim)
        )

    3 usages
    def encode(self, x):
        h = self.encoder(x)
        mean, log_var = torch.chunk(h, 2, dim=-1)
        return mean, log_var

    6 usages
    def decode(self, z, c):
        combined = torch.cat([z, c], dim=-1)
        return self.decoder(combined)

    3 usages
    def reparameterize(self, mean, log_var):
        std = torch.exp(0.5 * log_var)
        eps = torch.randn_like(std)
        return mean + eps * std

    def forward(self, x, c):
        mean, log_var = self.encode(x)
        z = self.reparameterize(mean, log_var)
        return self.decode(z, c), mean, log_var


print('All Models are created!')
```

train01_Model01.py

```python
import torch
import torch.nn as nn
import torch.optim as optim
import pickle
import matplotlib.pyplot as plt
"""
This code implements a Conditional Variational Autoencoder (CVAE) that learns to reconstruct hand grasp poses
conditioned on object affordances. It trains the model using a combination of reconstruction loss and KL divergence and
evaluates the model using validation data to track performance across multiple types of errors, such as hand pose,
joint, and translation errors. Finally, it saves the model and plots the losses and errors.
"""
# Load the Model 01 - CVAE_01 From all_models.py
from Models_Trains.all_models import CVAE_01
# Load the loading_data function From loading_data.py to load prepared data
from Models_Trains.loading_data import loading_data

hand_train, hand_val, hand_test, obj_train, obj_val, obj_test, train_dataset, val_dataset, test_dataset, \
    train_loader, val_loader, test_loader, obj_name=loading_data('../PreprocessData/hand_object_data.pkl',64)

# Hyperparameters and Model Initialization
input_dim = hand_train.shape[1]  # input data dimensions
latent_dim = 32
condition_dim = obj_train.shape[1]  # condition data dimensions
num_epochs = 50

model = CVAE_01(input_dim, latent_dim, condition_dim)
optimizer = optim.Adam(model.parameters(), lr=0.001)

t_loss=[]
v_loss=[]
v_rec_loss=[]
v_pose_error=[]
v_joints_error=[]
v_trans_error=[]


# Training loop
"""
model.train() sets the model to training mode.
For each batch, hand and object data are fed into the model.
The model reconstructs the hand pose and generates mean and variance (from the latent space).
The reconstruction loss is calculated using Mean Squared Error (MSE) between the original and reconstructed hand pose.
KL divergence is calculated to ensure that the latent space follows a normal distribution.
The total loss is the sum of reconstruction loss and KL divergence.
The optimizer backpropagates (loss.backward()) and updates model weights (optimizer.step()).
"""
for epoch in range(num_epochs):
    model.train()
    train_loss = 0
    for batch in train_loader:
        hand_pose_data, object_data = batch
        optimizer.zero_grad()
        reconstructed, mean, log_var = model(hand_pose_data, object_data)
        recon_loss = nn.functional.mse_loss(reconstructed, hand_pose_data, reduction='sum')
        kl_divergence = -0.5 * torch.sum(1 + log_var - mean.pow(2) - log_var.exp())
        loss = recon_loss + kl_divergence
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

    train_loss /= len(train_loader.dataset)
    t_loss.append(train_loss)

```

```
    # Validation loop
    """
    Validation is performed using model.eval() to disable dropout and other training-specific layers.
    No gradients are calculated in the validation phase (torch.no_grad()).
    Similar to the training loop, reconstruction loss and KL divergence are computed, but without backpropagation.
    Hand pose, joints, and translation errors are calculated separately
    """
    model.eval()
    val_loss = 0
    val_recon_loss = 0
    val_pose_error = 0
    val_joints_error = 0
    val_trans_error = 0
    with torch.no_grad():
        for batch in val_loader:
            hand_pose_data, object_data = batch
            reconstructed, mean, log_var = model(hand_pose_data, object_data)
            recon_loss = nn.functional.mse_loss(reconstructed, hand_pose_data, reduction='sum')
            kl_divergence = -0.5 * torch.sum(1 + log_var - mean.pow(2) - log_var.exp())
            loss = recon_loss + kl_divergence
            val_loss += loss.item()
            val_recon_loss += recon_loss.item()

            # Calculate errors for hand pose, joints, and translation
            hand_pose_dim = 48
            joints_dim = 21 * 3
            trans_dim = 3

            pose_error = nn.functional.mse_loss(reconstructed[:, :hand_pose_dim], hand_pose_data[:, :hand_pose_dim],
                                                reduction='sum')
            joints_error = nn.functional.mse_loss(reconstructed[:, hand_pose_dim:hand_pose_dim + joints_dim],
                                                  hand_pose_data[:, hand_pose_dim:hand_pose_dim + joints_dim],
                                                  reduction='sum')
            trans_error = nn.functional.mse_loss(
                reconstructed[:, hand_pose_dim + joints_dim:hand_pose_dim + joints_dim + trans_dim],
                hand_pose_data[:, hand_pose_dim + joints_dim:hand_pose_dim + joints_dim + trans_dim], reduction='sum')

            val_pose_error += pose_error.item()
            val_joints_error += joints_error.item()
            val_trans_error += trans_error.item()

    val_loss /= len(val_loader.dataset)
    val_recon_loss /= len(val_loader.dataset)
    val_pose_error /= len(val_loader.dataset)
    val_joints_error /= len(val_loader.dataset)
    val_trans_error /= len(val_loader.dataset)
    v_loss.append(val_loss)
    v_rec_loss.append(val_recon_loss)
    v_pose_error.append(val_pose_error)
    v_joints_error.append(val_joints_error)
    v_trans_error.append(val_trans_error)

    print(
        f'Epoch {epoch + 1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f},'
        f' Val Recon Loss: {val_recon_loss:.4f}')
    print(
        f'Val Pose Error: {val_pose_error:.4f}, Val Joints Error: {val_joints_error:.4f},'
        f' Val Trans Error: {val_trans_error:.4f}')

# Save the model
torch.save(model.state_dict(), '/cvae_01_weights.pth')
print("Model saved successfully!")

# Save the losses and errors
losses_errors = {
    't_loss': t_loss,
```

```
    't_loss': t_loss,
    'v_loss': v_loss,
    'v_rec_loss': v_rec_loss,
    'v_pose_error': v_pose_error,
    'v_joints_error': v_joints_error,
    'v_trans_error': v_trans_error
}

with open('/cvae_01_losses_errors.pkl', 'wb') as f:
    pickle.dump(losses_errors, f)

print("Losses And Errors saved successfully!")

# Plot Losses and Errors
fig, axes = plt.subplots(2, 3, figsize=(12, 6))
axes = axes.flatten()

axes[0].plot(t_loss)
axes[0].set_title('Train Loss ')

axes[1].plot(v_loss)
axes[1].set_title('Validation Loss')

axes[2].plot(v_rec_loss)
axes[2].set_title('Reconstruct Error')

axes[3].plot(v_pose_error)
axes[3].set_title('HandPose Error')

axes[4].plot(v_joints_error)
axes[4].set_title('HandJoints Error')

axes[5].plot(v_trans_error)
axes[5].set_title('HandTrans Error')

for i in range(6, len(axes)):
    fig.delaxes(axes[i])

plt.tight_layout()
plt.show()

```

evaluation_1.py

```python
import torch
import torch.nn.functional as F
import matplotlib.pyplot as plt

""" ------ Evaluation the First Phase - CVAE_01, CVAE_02 and CVAE_03
Evaluating the performance of three different Conditional Variational Autoencoders (CVAE) on a test dataset.
Comparing the Mean Squared Error (MSE) and Mean Absolute Error (MAE) between the hand poses predicted by each model
and the actual hand poses from the test dataset.
"""

# Load the Models CVAE_01, CVAE_02 and CVAE_03 From all_models.py
from Models_Trains.all_models import CVAE_01
from Models_Trains.all_models import CVAE_02
from Models_Trains.all_models import CVAE_03
# Load the loading_data function From loading_data.py to load prepared data
from Models_Trains.loading_data import loading_data

# The models hyperparameters
input_dim = 114
latent_dim = 32
condition_dim = 30

# LOAD DATA
hand_train, hand_val, hand_test, obj_train, obj_val, obj_test, train_dataset, val_dataset, test_dataset, \
    train_loader, val_loader, test_loader, obj_names = loading_data('../PreprocessData/hand_object_data.pkl', 64)


# Function to compute the Mean Squared Error (MSE) between the predicted and actual hand poses.
1 usage
def mean_squared_error(pred, target):
    return F.mse_loss(pred, target)


# Function to compute the Mean Absolute Error (MAE) between the predicted and actual hand poses.
1 usage
def mean_absolute_error(pred, target):
    return F.l1_loss(pred, target)


weights_path = ['cvae_01_weights.pth', 'cvae_02_weights.pth', 'cvae_03_weights.pth', 'cvae_03Noise_weights.pth']
mse_models = []
mae_models = []
for i in range(len(weights_path)):
    if weights_path[i] == 'cvae_01_weights.pth':
        model = CVAE_01(input_dim, latent_dim, condition_dim)
        model_name = 'Model_01'
    elif weights_path[i] == 'cvae_02_weights.pth':
        model = CVAE_02(input_dim, latent_dim, condition_dim)
        model_name = 'Model_02'
    elif weights_path[i] == 'cvae_03_weights.pth' or weights_path[i] == 'cvae_03Noise_weights.pth':
        model = CVAE_03(input_dim, latent_dim, condition_dim)
        model_name = 'Model_03'

    model.load_state_dict(torch.load(f'../Models_Trains/{weights_path[i]}'))
    model.eval()
    mse_list = []
    mae_list = []

    with torch.no_grad():
        for i in range(len(obj_val)):
            object_data = obj_test[i].unsqueeze(0)  # Example object data
            actual_hand_pose = hand_test[i].unsqueeze(0)  # Actual hand pose data

            z = torch.randn(1, latent_dim)
            generated_hand_pose = model.decode(z, object_data)
```

```
            mse = mean_squared_error(generated_hand_pose, actual_hand_pose)
            mae = mean_absolute_error(generated_hand_pose, actual_hand_pose)

            mse_list.append(mse.item())
            mae_list.append(mae.item())

    # Calculate average metrics
    avg_mse = sum(mse_list) / len(mse_list)
    avg_mae = sum(mae_list) / len(mae_list)

    mse_models.append(avg_mse)
    mae_models.append(avg_mae)

    print(f'Average MSE_{model_name}: {avg_mse:.4f}')
    print(f'Average MAE_{model_name}: {avg_mae:.4f}')

# PLOT ALL 3 Models MSE and MAE
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

axes[0].bar('CVAE_01', mse_models[0], color='blue')
axes[0].bar('CVAE_02', mse_models[1], color='green')
axes[0].bar('CVAE_03', mse_models[2], color='yellow')
axes[0].bar('CVAE_03_Noise', mse_models[3], color='orange')

axes[0].set_xlabel('Models')
axes[0].set_ylabel('Average Value (mm²)')
axes[0].set_title('Average of MSE On Test Dataset')

axes[1].bar('CVAE_01', mae_models[0], color='blue')
axes[1].bar('CVAE_02', mae_models[1], color='green')
axes[1].bar('CVAE_03', mae_models[2], color='yellow')
axes[1].bar('CVAE_03_Noise', mae_models[3], color='orange')

axes[1].set_xlabel('Models')
axes[1].set_ylabel('Average Value (mm²)')
axes[1].set_title('Average of MAE On Test Dataset')

plt.tight_layout()
plt.show()
```

losses_errors_1.py

```python
import pickle
import matplotlib.pyplot as plt
"""
Plot the First Phase - CVAE_01, CVAE_02 and CVAE_03 Losses and Errors together
"""
# Load losses and errors for the Model_01
with open('../Models_Trains/cvae_01_losses_errors.pkl', 'rb') as f:
    losses_errors_1 = pickle.load(f)
t_loss_1 = losses_errors_1['t_loss']
v_loss_1 = losses_errors_1['v_loss']
v_rec_loss_1 = losses_errors_1['v_rec_loss']
v_pose_error_1 = losses_errors_1['v_pose_error']
v_joints_error_1 = losses_errors_1['v_joints_error']
v_trans_error_1 = losses_errors_1['v_trans_error']

# Load losses and errors for the Model_02
with open('../Models_Trains/cvae_02_losses_errors.pkl', 'rb') as f:
    losses_errors_2 = pickle.load(f)
t_loss_2 = losses_errors_2['t_loss']
v_loss_2 = losses_errors_2['v_loss']
v_rec_loss_2 = losses_errors_2['v_rec_loss']
v_pose_error_2 = losses_errors_2['v_pose_error']
v_joints_error_2 = losses_errors_2['v_joints_error']
v_trans_error_2 = losses_errors_2['v_trans_error']

# Load losses and errors for the Model_03
with open('../Models_Trains/cvae_03_losses_errors.pkl', 'rb') as f:
    losses_errors_03 = pickle.load(f)
t_loss_3 = losses_errors_03['t_loss']
v_loss_3 = losses_errors_03['v_loss']
v_rec_loss_3 = losses_errors_03['v_rec_loss']
v_pose_error_3 = losses_errors_03['v_pose_error']
v_joints_error_3 = losses_errors_03['v_joints_error']
v_trans_error_3 = losses_errors_03['v_trans_error']

# Load losses and errors for the Model_03_Noise
with open('../Models_Trains/cvae_03Noise_losses_errors.pkl', 'rb') as f:
    losses_errors_03_2 = pickle.load(f)
t_loss_3N = losses_errors_03_2['t_loss']
v_loss_3N = losses_errors_03_2['v_loss']
v_rec_loss_3N = losses_errors_03_2['v_rec_loss']
v_pose_error_3N = losses_errors_03_2['v_pose_error']
v_joints_error_3N = losses_errors_03_2['v_joints_error']
v_trans_error_3N = losses_errors_03_2['v_trans_error']


fig, axes = plt.subplots(2, 3, figsize=(15, 9))
axes = axes.flatten()

# Plotting training loss
axes[0].plot(t_loss_1, label='CVAE_01', color='blue')
axes[0].plot(t_loss_2, label='CVAE_02', color='green')
axes[0].plot(t_loss_3, label='CVAE_03', color='yellow')
axes[0].plot(t_loss_3N, label='CVAE_03_Noise', color='orange')
axes[0].set_title('Train Loss (MSE in mm²)')
axes[0].set_xlabel('Number of Epochs')
axes[0].set_ylabel('Loss Value (mm²)')
axes[0].legend()

# Plotting validation loss
axes[1].plot(v_loss_1, label='CVAE_01', color='blue')
axes[1].plot(v_loss_2, label='CVAE_02', color='green')
axes[1].plot(v_loss_3, label='CVAE_03', color='yellow')
axes[1].plot(v_loss_3N, label='CVAE_03_Noise', color='orange')
axes[1].set_title('Validation Loss (MSE in mm²)')
```

```
axes[1].set_xlabel('Number of Epochs')
axes[1].set_ylabel('Loss Value (mm²)')
axes[1].legend()

# Plotting reconstruction error
axes[2].plot(v_rec_loss_1, label='CVAE_01', color='blue')
axes[2].plot(v_rec_loss_2, label='CVAE_02', color='green')
axes[2].plot(v_rec_loss_3, label='CVAE_03', color='yellow')
axes[2].plot(v_rec_loss_3N, label='CVAE_03_Noise', color='orange')
axes[2].set_title('Reconstruct Error (MSE in mm²)')
axes[2].set_xlabel('Number of Epochs')
axes[2].set_ylabel('Error Value (mm²)')
axes[2].legend()

# Plotting hand pose error
axes[3].plot(v_pose_error_1, label='CVAE_01', color='blue')
axes[3].plot(v_pose_error_2, label='CVAE_02', color='green')
axes[3].plot(v_pose_error_3, label='CVAE_03', color='yellow')
axes[3].plot(v_pose_error_3N, label='CVAE_03_Noise', color='orange')
axes[3].set_title('HandPose Error (MSE in mm²)')
axes[3].set_xlabel('Number of Epochs')
axes[3].set_ylabel('Error Value (mm²)')
axes[3].legend()

# Plotting hand joints error
axes[4].plot(v_joints_error_1, label='CVAE_01', color='blue')
axes[4].plot(v_joints_error_2, label='CVAE_02', color='green')
axes[4].plot(v_joints_error_3, label='CVAE_03', color='yellow')
axes[4].plot(v_joints_error_3N, label='CVAE_03_Noise', color='orange')
axes[4].set_title('HandJoints Error (MSE in mm²)')
axes[4].set_xlabel('Number of Epochs')
axes[4].set_ylabel('Error Value (mm²)')
axes[4].legend()

# Plotting hand translation error
axes[5].plot(v_trans_error_1, label='CVAE_01', color='blue')
axes[5].plot(v_trans_error_2, label='CVAE_02', color='green')
axes[5].plot(v_trans_error_3, label='CVAE_03', color='yellow')
axes[5].plot(v_trans_error_3N, label='CVAE_03_Noise', color='orange')
axes[5].set_title('HandTrans Error (MSE in mm²)')
axes[5].set_xlabel('Number of Epochs')
axes[5].set_ylabel('Error Value (mm²)')
axes[5].legend()

plt.tight_layout()
plt.show()
```

vis_hand_3D.py

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.transform import Rotation as R
import pickle

# 3D Visualizing
"""
This code loads a hand's 3D data (joints, pose, and translation) from a .pkl file (located in the meta folder
and visualizes it using 3D plotting. The key transformations applied include translation and rotation
(using axis-angle to rotation matrix conversion). The hand's skeleton is drawn based on MANO joint connections,
and the result is displayed in a 3D plot.
"""
1 usage
def plot_hand_3d(handJoints3D, handTrans, handPose,objname):
    # Applying translation
    handJoints3D_transformed = handJoints3D + handTrans

    # Converting axis-angle to rotation matrix and applying rotation
    rotation = R.from_rotvec(handPose[:3])
    handJoints3D_transformed = rotation.apply(handJoints3D_transformed)

    # Plotting
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Extracting coordinates
    xs = handJoints3D_transformed[:, 0]
    ys = handJoints3D_transformed[:, 1]
    zs = handJoints3D_transformed[:, 2]

    # Plotting joints
    ax.scatter(xs, ys, zs, c='r', marker='o')

    # Connecting joints to form the skeleton (MANO joint order)
    connections = [
        (0, 7), (7, 8), (8, 9), (9, 20),  # Little finger
        (0, 10), (10, 11), (11, 12), (12, 19),  # Ring finger
        (0, 4), (4, 5), (5, 6), (6, 18),  # Middle finger
        (0, 1), (1, 2), (2, 3), (3, 17),  # Index finger
        (0, 13), (13, 14), (14, 15), (15, 16)  # Thumb
    ]

    for (i, j) in connections:
        ax.plot([xs[i], xs[j]], [ys[i], ys[j]], [zs[i], zs[j]], 'b')

    # Setting labels
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title(f'3D Scatter Plot of Transformed Hand pose \n Based on Object File Name: ({objname})')

    plt.show()



# Load Data from meta folder in H3D_v3 Dataset
filepath = r"D:\UNI\Sem3\Dissertation\My effort\HOnnotate\ho3d-master\Dataset\HO3D_v3\train\SMu42\meta\1000.pkl"

hand_poses = []
object_translations = []
object_rotations = []
hand_translations = []
object_corners_3d = []
hand_joints_3d = []

```

```python
with open(filepath, 'rb') as f:
    data = pickle.load(f)
    # Extract required fields from the loaded data
    hand_pose = data.get('handPose', np.zeros((48, 1))) #  48x1 vector
    hand_trans = data.get('handTrans', np.zeros(3))  # Optional field, default to zeros if not available
    hand_joints_3d_value = data.get('handJoints3D',np.zeros((21, 3)))  # Optional field, default to zeros if not availab
    obj_name=data['objName']


handJoints3D = hand_joints_3d_value
handTrans = hand_trans
handPose =  hand_pose


# Visualize the Hand
plot_hand_3d(handJoints3D, handTrans, handPose,obj_name)


```

vis_object_3D.py

```python
import numpy as np
import matplotlib.pyplot as plt
import pickle
from scipy.spatial.transform import Rotation as R
"""
This code loads a 3D model of an object from the YCB dataset, applies rotation and translation transformations to
it based on metadata (from a pickle file), and visualizes the transformed object points in a 3D scatter
plot using Matplotlib.
"""
# Function to load the points from an XYZ file in YCB Dataset ('models' folder in H3D_v3 Dataset)
1 usage
def load_xyz(file_path):
    points = []
    with open(file_path, 'r') as file:
        for line in file:
            if line.strip():  # Ensure the line is not empty
                x, y, z = map(float, line.split())
                points.append([x, y, z])
    return np.array(points)

# Function to apply rotation and translation to points
1 usage
def apply_transformations(points, obj_trans, obj_rot_matrix):
    # Apply rotation
    rotated_points = np.dot(points, obj_rot_matrix.T)
    # Apply translation
    translated_points = rotated_points + obj_trans
    return translated_points


# Load the hand-object data from the pickle file
filepath = r"D:\UNI\Sem3\Dissertation\My effort\HOnnotate\ho3d-master\Dataset\HO3D_v3\train\SS1\meta\0055.pkl"

with open(filepath, 'rb') as f:
    data = pickle.load(f)

    obj_trans = data.get('objTrans', np.zeros(3)) # Assuming this is a 3-element vector
    obj_rot = data.get('objRot', np.zeros(3)) # Assuming this is a 3-element vector
    obj_name = data['objName']

# Path to the points.xyz file of object coordination
xyz_file_path = rf"D:\UNI\Sem3\Dissertation\My effort\HOnnotate\ho3d-master\Dataset\models\{obj_name}\points.xyz"

# Load the points
points = load_xyz(xyz_file_path)

# Apply transformations to the points
rotation = R.from_rotvec(obj_rot.flatten()).as_matrix()
transformed_points = apply_transformations(points, obj_trans, rotation)

# Extract x, y, z coordinates for transformed points
x = transformed_points[:, 0]
y = transformed_points[:, 1]
z = transformed_points[:, 2]

# Plot the points in 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Create a 3D scatter plot
ax.scatter(x, y, z, c='b', marker='o')

# Set labels and title
ax.set_xlabel('X Coordinate')
ax.set_ylabel('Y Coordinate')
ax.set_zlabel('Z Coordinate')
ax.set_title(f'3D Scatter Plot of Transformed Object Points \n Object File Name: ({obj_name})')

# Set the aspect ratio to be equal
ax.set_box_aspect([1, 1, 1])  # Aspect ratio is 1:1:1

plt.show()

```

vis_reconstruct_hand_object_2D_1.py

```python
import pickle
import torch
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.transform import Rotation as R

#Load the Models From all_models.py
from Models_Trains.all_models import CVAE_02_1
from Models_Trains.all_models import CVAE_02_2
from Models_Trains.all_models import CVAE_02_3

# Load the loading_data function From loading_data.py to load prepared data
from Models_Trains.loading_data import loading_data

1 usage
def mean_squared_error(pred, target):
    return F.mse_loss(pred, target)

1 usage
def mean_absolute_error(pred, target):
    return F.l1_loss(pred, target)


"""
    Visualize reconstructed and actual hands based on an object in 2D given their respective transformations
    and coordinates.
    Parameters:
    hand1Trans (np.array): A 3x1 vector representing the reconstructed hand's translation.
    hand1Joints3D (np.array): A 21x3 matrix representing the 3D coordinates of the reconstructed hand's joints.
    hand2Trans (np.array): A 3x1 vector representing the actual hand's translation.
    hand2Joints3D (np.array): A 21x3 matrix representing the 3D coordinates of the actual hand's joints.
    objTrans (np.array): A 3x1 vector representing the object's translation.
    objRot (np.array): A 3x1 vector representing the object's rotation in axis-angle format.
    points (np.array): A Nx3 matrix representing the 3D coordinates of the object's points.
    objname: the object name from YCB dataset
    testNumber: Number of frame

"""
1 usage
def visualize_two_hands_and_object_2d(hand1Trans, hand1Joints3D, hand2Trans, hand2Joints3D, objTrans, objRot, points, ob

    # Apply hand translations
    hand1Joints3D_transformed = hand1Joints3D + hand1Trans
    hand2Joints3D_transformed = hand2Joints3D + hand2Trans

    # Apply object rotation and translation
    rotation_matrix = R.from_rotvec(objRot.flatten()).as_matrix()
    rotated_points = np.dot(points, rotation_matrix.T)
    translated_points = rotated_points + objTrans

    # Extract coordinates for transformed points
    obj_x = translated_points[:, 0]
    obj_y = translated_points[:, 1]

    # Extract coordinates for transformed hand joints
    hand1_x = hand1Joints3D_transformed[:, 0]
    hand1_y = hand1Joints3D_transformed[:, 1]

    hand2_x = hand2Joints3D_transformed[:, 0]
    hand2_y = hand2Joints3D_transformed[:, 1]

    fig, ax = plt.subplots(figsize=(10, 8))

```

```
    # Plot Object on X-Y plane
    ax.scatter(obj_x, obj_y, c='b', marker='o', label='Object')

    # Plot Hand 1 (Reconstructed) on X-Y plane
    ax.scatter(hand1_x, hand1_y, c='r', marker='o', label='Reconstructed Hand Joints')
    for (i, j) in [(0, 7), (7, 8), (8, 9), (9, 20), (0, 10), (10, 11), (11, 12), (12, 19),
                   (0, 4), (4, 5), (5, 6), (6, 18), (0, 1), (1, 2), (2, 3), (3, 17),
                   (0, 13), (13, 14), (14, 15), (15, 16)]:
        ax.plot([hand1_x[i], hand1_x[j]], [hand1_y[i], hand1_y[j]], 'r')

    # Plot Hand 2 (Actual) on X-Y plane
    ax.scatter(hand2_x, hand2_y, c='g', marker='o', label='Actual Hand Joints')
    for (i, j) in [(0, 7), (7, 8), (8, 9), (9, 20), (0, 10), (10, 11), (11, 12), (12, 19),
                   (0, 4), (4, 5), (5, 6), (6, 18), (0, 1), (1, 2), (2, 3), (3, 17),
                   (0, 13), (13, 14), (14, 15), (15, 16)]:
        ax.plot([hand2_x[i], hand2_x[j]], [hand2_y[i], hand2_y[j]], 'g')

    ax.set_xlabel('X Coordinate')
    ax.set_ylabel('Y Coordinate')
    ax.set_title(f'Reconstructed vs Actual Hand Pose - X-Y Plane (Test Data[{testNumber}]) \n '
                 f'Based on Object File Name: ({objname})')
    ax.legend()
    ax.grid(True)

    plt.tight_layout()
    plt.show()

"""
This function reads a file containing 3D points (X, Y, Z) from the YCB object model. These points are later
used to plot the object's shape."""
1 usage
def load_xyz(file_path):
    points = []
    with open(file_path, 'r') as file:
        for line in file:
            if line.strip():  # Ensure the line is not empty
                x, y, z = map(float, line.split())
                points.append([x, y, z])
    return np.array(points)



# HYPERPARAMETERS
input_dim = 114
latent_dim=64
condition_dim = 30

# LOAD DATA
hand_train, hand_val, hand_test, obj_train, obj_val, obj_test, train_dataset, val_dataset, test_dataset, \
    train_loader, val_loader, test_loader,obj_names=loading_data('../PreprocessData/hand_object_data.pkl', 64)

# LOAD MODEL AND SCALERS
model = CVAE_02_1(input_dim, latent_dim, condition_dim)
model.load_state_dict(torch.load('../Models_Trains\cvae_02_1_weights.pth'))

with open('../PreprocessData/scalers.pkl', 'rb') as scalers_file:
    scalers= pickle.load(scalers_file)

# Extract the individual scalers
scaler_hand_joints = scalers['scaler_hand_joints']
scaler_obj_corners = scalers['scaler_obj_corners']
scaler_hand_pose = scalers['scaler_hand_pose']
scaler_hand_trans = scalers['scaler_hand_trans']
scaler_obj_trans = scalers['scaler_obj_trans']
scaler_obj_rot = scalers['scaler_obj_rot']
```

```
127
128 model.eval()
129
130 with torch.no_grad():
131     file=16653
132     object_data = obj_test[file].unsqueeze(0)
133     object_name=obj_names[file]
134     hand_data=hand_test[file].unsqueeze(0)
135     z = torch.randn(1, latent_dim)
136     generated_hand_pose = model.decode(z, object_data)
137     mse = mean_squared_error(generated_hand_pose, hand_data)
138     mae = mean_absolute_error(generated_hand_pose, hand_data)
139     print("Generated hand pose:", generated_hand_pose)
140     print(generated_hand_pose.shape[1])
141
142 #------------------- GENERATED HAND ----------
143 #RECONSTRUCTED HAND INFORMATION
144 generated_hand_pose_np = generated_hand_pose.numpy().reshape(-1)
145 hand_pose_generated = generated_hand_pose_np[:48]  # First 48 elements for hand pose
146 hand_trans_generated = generated_hand_pose_np[48:51]  # Next 3 elements for hand translation
147 hand_joints_generated = generated_hand_pose_np[51:]  # Remaining 63 elements for hand joints (21x3)
148 hand_joints_generated = hand_joints_generated.reshape(-1, 3) # Reshape hand joints back to (21, 3)
149 # Inverse transform each component using the corresponding scaler
150 hand_pose_original = scaler_hand_pose.inverse_transform(hand_pose_generated.reshape(1, -1)).reshape(-1)
151 hand_trans_original = scaler_hand_trans.inverse_transform(hand_trans_generated.reshape(1, -1)).reshape(-1)
152 hand_joints_original = scaler_hand_joints.inverse_transform(hand_joints_generated.reshape(1, -1)).reshape(-1, 3)
153 #-----------------------------------------------
154
155 #------------------- ACTUAL HAND ----------
156 hand_data_np = hand_data.numpy().reshape(-1)
157 hand_pose_actual = hand_data_np[:48]  # First 48 elements for hand pose
158 hand_trans_actual = hand_data_np[48:51]  # Next 3 elements for hand translation
159 hand_joints_actual = hand_data_np[51:]  # Remaining 63 elements for hand joints (21x3)
160 hand_joints_actual = hand_joints_actual.reshape(-1, 3)# Reshape hand joints back to (21, 3)
161 # Inverse transform each component using the corresponding scaler
162 hand_pose_act_original = scaler_hand_pose.inverse_transform(hand_pose_actual.reshape(1, -1)).reshape(-1)
163 hand_trans_act_original = scaler_hand_trans.inverse_transform(hand_trans_actual.reshape(1, -1)).reshape(-1)
164 hand_joints_act_original = scaler_hand_joints.inverse_transform(hand_joints_actual.reshape(1, -1)).reshape(-1, 3)
165
166 #------------------- OBJECT  ----------
167 object_data_np = object_data.numpy().reshape(-1)
168 obj_trans=object_data_np[:3] # First 3 elements for object translation
169 obj_rot=object_data_np[3:6] #Next 3 elements for object rotation
170 # Inverse transform each component using the corresponding scaler
171 obj_trans_original = scaler_obj_trans.inverse_transform(obj_trans.reshape(1, -1)).reshape(-1)
172 obj_rot_original = scaler_obj_rot.inverse_transform(obj_rot.reshape(1, -1)).reshape(-1)
173
174 xyz_file_path = fr"D:\UNI\Sem3\Dissertation\My effort\HOnnotate\ho3d-master\Dataset\models\{object_name}\points.xyz"
175 points = load_xyz(xyz_file_path)
176
177 print(f'Object Name: {object_name}')
178 print(f'The test number: {file}')
179 print(f'Mean Square Error: {mse}')
180 print(f'Mean Absolute Error: {mae}')
181
182 # Visualize hand and object in 2D
183 visualize_two_hands_and_object_2d(hand_trans_original, hand_joints_original, hand_trans_act_original,
184                                   hand_joints_act_original, obj_trans_original, obj_rot_original,
185                                   points,object_name,file)
```