# Mixed Reality based assembly instruction and continuous tracking of the object using external camera

Subin Raj[1], Bikram Karmakar[2], Gyanig Kumar[1], Harshitha[1], Abhishek Mukhopadhyay[1], Yash Sahoo[1], Amrit Chatterjee[2], Pradipta Biswas[1]

[1]Indian Institute of Science, Bangalore, India

[2]Collins Aerospace Systems

**Executive Summary:** Assembling a complex component often require expert assistance. This report is proposing an Augmented Reality (AR) and Artificial Intelligence (AI) assistance for helping assembly process of complex machine component related to advanced manufacturing. In particular, the project compared:

- Hardware Configuration
- Software Architecture
- Machine Vision Algorithms to combine AR and AI based systems for assembly process.

The report has investigated and compared different hardware configurations and software architecture and compare them with respect to **latency**, **accuracy**, and **human factors**.

# Table of Contents

# 1. Introduction

The recent development in rendering techniques, artificial intelligence, and networking has paved a new path for the utilization of XR technologies, which encompass Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR). Currently, XR technologies are demonstrating promising results in various industrial areas, especially in assembly and maintenance tasks. In these contexts, instructions are provided to guide users through tasks. Firstly, it is essential to understand how and what instructions should be given to the user. Instructions can be delivered through various means, such as visual, audio, and speech. They should be made visible to the user when needed. When providing instructions about an object, markers or object detection models can be employed. However, attaching markers in real-time assembly processes is impractical due to varying component sizes and frequent changes. Moreover, the time-consuming nature of attaching markers on each object in the production line is a limitation. To address this issue, using object detection techniques is a more viable option. Implementing object detection models involves capturing images and labeling, which can consume considerable time and manpower. Recent advancements in synthetic data generation and automatic labeling have mitigated these challenges. Nevertheless, training models solely with synthetic data may not perform well in the real world due to differences between synthetic and real-world data. Generating synthetic data that accurately reflects real-world attributes is difficult, as models trained on synthetic data may not learn these attributes. To incorporate real-world attributes, a more effective approach is to train the model using a combination of real and synthetic data. This approach can enhance the model's ability to learn real-world attributes while reducing the required manpower and time.

When employing XR technologies to instruct workers, it is crucial that the instructions are accurately rendered and easily understood. Using a marker to display instructions ensures continuous trackability of objects, facilitated by the marker's continuous tracking capabilities. Consequently, the position of instructions can be easily adjusted based on the marker's position. Facilitating these capabilities is crucial for the successful implementation of object detection in rendering instructions. Presently, HMD cameras capture real-world images and transmit them to the object detection model, with spatial perception being employed to render instructions. However, using an HMD camera presents challenges such as capturing noiseless images due to its placement on the user's head, resulting in focus difficulties and increased shaking. This, in turn, leads to noisy images and reduced effectiveness in object detection. Spatial perception contributes to the computational load on the HMD, causing a slowdown in the system. To mitigate these challenges, the use of an external camera for image capture is proposed. This approach aims to decrease the likelihood of capturing noisy images. Additionally, machine learning models, including linear regression, polynomial regression, and support vector machines, can be utilized to render holograms on real objects. In this context, we introduce a method that involves continuously tracking real objects on an assembly table using an external camera and mapping instructions onto the real objects through regression methods. The main contribution of this work is:

- Continuously adjusting the holograms according to position of the objects in the real world using external camera.
- Analysing the effectiveness of using synthetic image in object detection model.
- Analysing the effectiveness of giving instructions through different modalities.

# 2. Preliminary Study

Initially, we compared different hardware platforms and deep learning models with respect to accuracy and latency. The best combination was selected for subsequent development and user study.

## 2.1. GPU Comparison

In this study, we introduced an external camera for capturing images and sending them to the object detection model to consistently map holograms onto real objects. The speed of the continuous mapping of holograms depends primarily on the type of processor and GPU. In this work, we analysed of three distinct hardware components: the GeForce RTX 3090, GeForce RTX 3060, and Jetson Nano. The objective behind selecting these three hardware options was to assess their performance in the context of deep learning applications. The GeForce RTX 3090 is a top-tier graphics card primarily engineered for gaming and personal use. In contrast, the GeForce RTX 3060 is positioned as a mid-range graphics card, tailored for game development purposes. Jetson nano is a small, powerful computer that allows us to run multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing.

## GeForce RTX 3090

The GeForce RTX 3090 is a high-end graphics card produced by NVIDIA, renowned for its powerful performance and cutting-edge features. Released as part of the RTX 30 series, it boasts a significant leap in graphics capabilities. The RTX 3090 is built on the Ampere architecture, featuring 10,496 CUDA cores, a base clock of 1.4 GHz, and an impressive 24 GB of GDDR6X video memory with a 384-bit memory interface. This massive VRAM capacity makes it ideal for demanding tasks such as 3D rendering, video editing, and gaming at ultra-high resolutions. The card also supports real-time ray tracing and AI-enhanced graphics through NVIDIA's advanced technologies, delivering an unparalleled gaming and creative experience for enthusiasts and professionals alike. The RTX 3090 stands as a flagship model, catering to users with demanding computational needs and a desire for top-tier graphical performance.

## GeForce RTX 3060

The GeForce RTX 3060, developed by NVIDIA, is a graphics card designed for gaming and creative workloads. Launched as part of the RTX 30 series, it features the Ampere architecture, offering significant advancements in performance and ray tracing capabilities. The RTX 3060 is equipped with 12 gigabytes of GDDR6 video memory, providing ample resources for handling demanding graphics tasks. With a base clock speed of around 1320 MHz and a boost clock that can reach higher frequencies, the RTX 3060 delivers smooth and immersive gaming experiences. Ray tracing, a key feature of the RTX series, is enhanced on the RTX 3060, allowing for realistic lighting, reflections, and shadows in supported games. This mid-range graphics card strikes a balance between performance and affordability, making it a popular choice for gamers and content creators seeking a robust solution for their computing needs.

## Jetson Nano

The Jetson Nano is a compact and powerful single-board computer designed for edge AI applications. Manufactured by NVIDIA, this device boasts a quad-core ARM Cortex-A57 CPU coupled with a 128-core Maxwell GPU, providing a robust platform for running AI workloads efficiently. With 4 GB of LPDDR4 RAM and a microSD card slot for storage, the Jetson Nano supports various AI frameworks, including TensorFlow and PyTorch. Connectivity features include Gigabit Ethernet, four USB 3.0 ports, and HDMI output for display. Its small form factor and low power consumption make it an ideal choice for projects requiring AI processing at the edge, such as robotics, image recognition, and autonomous systems. The Jetson Nano empowers developers and enthusiasts to implement sophisticated AI applications in a cost-effective and energy-efficient manner.

## Comparison

The GeForce RTX 3090, GeForce RTX 3060, and Jetson Nano represent a spectrum of NVIDIA's GPU offerings catering to different segments of the market. The GeForce RTX 3090 is a high-end graphics card designed for gaming and professional applications, featuring an impressive 24GB of GDDR6X VRAM, 10,496 CUDA cores, and robust ray tracing capabilities. In contrast, the GeForce RTX 3060 is a more mainstream option with 12GB of GDDR6 VRAM and 3,584 CUDA cores, delivering excellent performance for mid-range gaming. On the other hand, the Jetson Nano is geared towards edge computing and AI development, equipped with a Quad-core ARM Cortex-A57 CPU, 128-core Maxwell GPU, and 4GB of LPDDR4 RAM. While the RTX series focuses on gaming prowess, the Jetson Nano caters to developers and enthusiasts keen on exploring AI and robotics applications at a more accessible level. Each of these GPUs serves distinct purposes, reflecting NVIDIA's diverse product lineup. GeForce RTX 3090 and 3060 has better latency because of high end GPU and graphics card. It can manage all deep learning models which we used in this project. We did not experience any crashing while we train the model with this. In contrast to that, jetson nano has poor latency and unable to handle the modules, and frequent crashing. we compared three different hardware setups, outlining their respective advantages and disadvantages in the Table 1.

Table 1: Hardware Specification and detection quality

| | Training System NVIDIA GeForce RTX 3090 | Inference System NVIDIA GeForce RTX 3060 | Jetson Nano |
|---|---|---|---|
| **System Spec** | *GPU*: 24 GB<br>*CPU*: Intel i9 11th gen<br>*RAM*: 16 GB | *GPU*: 6 GB<br>*CPU*: Intel i7 11th gen<br>*RAM*: 16 GB | *GPU*: 128 CUDA cores<br>*CPU*: Quad-core ARM Cortex-A57 MPCore<br>*RAM*: 4 GB |
| **Quality of Detection** | Best Latency<br>CPU can handle all modules.<br>No crashing | Better Latency<br>CPU can handle all modules.<br>No crashing | Poor Latency<br>CPU unable to handling modules.<br>Crashing |

## 2.2. Deep learning Model Comparison

Deep learning models are a subset of machine learning algorithms inspired by the structure and function of the human brain. They consist of artificial neural networks with multiple layers (deep neural networks), allowing them to automatically learn and represent complex patterns from data. These models excel at tasks such as image and speech recognition, natural language processing, and other complex pattern recognition problems. Deep learning models leverage backpropagation, optimization algorithms, and activation functions to iteratively adjust parameters and improve their ability to make accurate predictions or classifications. One of the applications of deep learning labels is object detection. Object detection algorithms can be classified into three types: traditional object detection algorithms, two-stage methods, and one-stage methods. Traditional object detection algorithms operate based on knowledge and statistics, relying on manual design and having limitations. These algorithms work on cascades, support vector machines, Histograms of Oriented Gradient, deformable part models, etc. The two-stage method, also known as the deep learning classification algorithm (e.g., RCNN, Fast RCNN, Faster RCNN, SPP-net), uses a neural network to extract information from the detected image. The one-stage method is referred to as a deep learning regression algorithm, such as YOLO, SSD, etc. Unlike the two-stage method, it directly provides the position and category of the detected object in the output layer, enhancing its speed.

The vast amount of existing work compared the different deep learning models. Recently, wang et al (2023), please refer to citation 2, compared different one stage and two stage models on MS COCO dataset. The result is shown in figure 1. From the study, they found that one stage method gave better precision at low inference time compared to two stage methods.

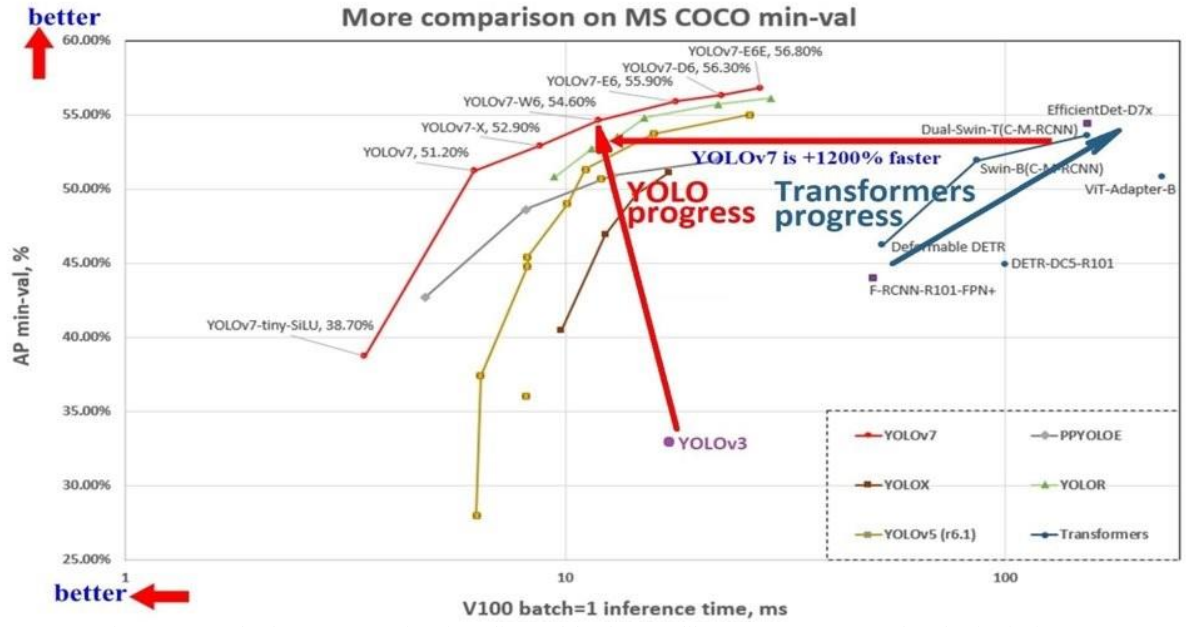



Figure 1: AP and Inference Comparison in earlier models of YOLO like v5, X, v7, DETR , referred to in citation 22

The figure 2 explains the effectiveness different version of the Yolo model with the different model size. It shows Yolov8 works better compared to other versions of the Yolo model with less model parameters. From the existing work, we found that one stage method worked better compared to two stage methods. In addition to that, we found that Yolov8 worked better compared to other Yolo versions. However, we planned to validate above results on custom dataset is explained in following section.

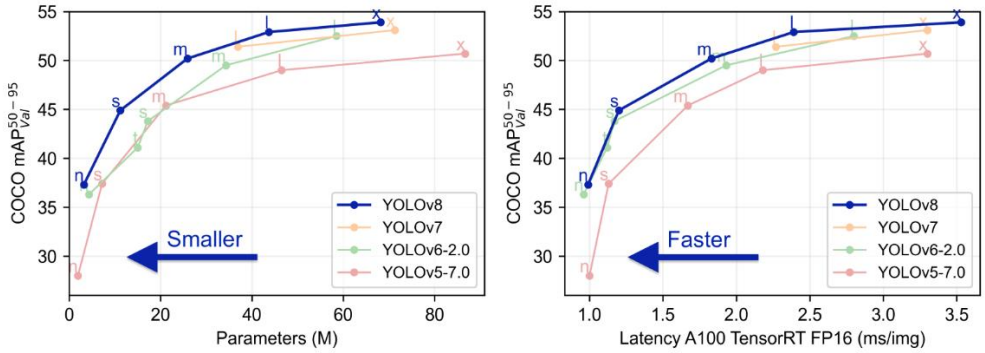



Figure 2: Comparison of Model Size, FPS between v5, v7, v8

## 3. Synthetic Data Generation

So far, four variants of YOLO and DeTR have been compared in detecting components in the assembling process. However, the performance of any object detection model relies heavily on training, which is a data-centric and resource-intensive process. Creating datasets comes with high costs, time constraints, and sometimes infeasibility due to security, privacy, and local regulations. A recent survey highlighted that 96 percent of enterprises encounter challenges in training and labeling data. Please refer to citation 35. Synthetic data has shown promise previously, please refer to citation 36 & 37, significantly reducing the time needed to generate labeled data for new product variants refer to citation 35. Using synthetic data allows rapid iteration without the time-consuming data acquisition process. Despite its advantages, challenges such as realism disparities hinder complete reliance on synthetic data. Tremblay, please refer to citation 36 & 37, aimed to bridge this gap between real and synthetic data using Domain Randomization. This study specifically focuses on comparing various Image-to-Image Translation GAN models in generating realistic data.

**Generative Adversarial Network**: The generative models have gained attention in the field of unsupervised learning via generative adversarial network due to its outstanding data generation capability. The idea of adversarial networks was inspired by the blog of Olli Niemitalo in 2010, please refer to citation 4. GAN model can be summarized as training of two networks, generator, and discriminator. Discriminator is a binary classifier that learns to classify real and generated data. In contrast, generators confuse discriminators by generating realistic data from noise vectors. GAN have introduced many applications such as hand-written font generation referred in citation 2-3, image manipulation applications referred in citation 7-9, sketch synthesis referred in citation 10-12, image-to-image translation referred in citation 13-16, face frontal view generation referred in citation 17-19 and so on. Based on working principle, GAN has different variants.

- Conditional GAN referred in citation 20: It introduces conditioning variables to control the generated output, enabling targeted synthesis based on specific constraints or labels. They extend GANs by incorporating additional information during the generation process, facilitating tasks like image-to-image translation and controlled content generation.
- Deep Convolutional GAN referred in citation 21: DCGAN uses convolutional and convolutional-transpose layers in the generator and discriminator, respectively. Generators use CNN architecture, and the other network works as de-CNN called discriminator.
- Laplacian GAN referred in citation 22: The Laplacian GAN is a type of generative adversarial network that uses a Laplacian pyramid as the generator architecture. This allows the model to generate high-resolution images with detailed and realistic features. The Laplacian GAN has been shown to be effective in generating realistic images of faces, objects, and scenes. It has also been used for image-to-image translation tasks, such as converting a photo of a face into a painting.
- Energy-Based GAN referred in citation 23: Energy-based GANs (EBGANs) are a type of GAN architecture that uses an energy function to evaluate the quality of generated samples. Unlike traditional GANs, which use a discriminator network to distinguish between real and fake samples,

EBGANs use an energy function to assign a score to each sample, based on how well it matches the desired distribution. This allows for more flexible and efficient training, as the energy function can be optimized using a variety of techniques, such as maximum likelihood estimation or Markov chain Monte Carlo. Additionally, EBGANs can be more stable and easier to interpret than traditional GANs, as the energy function provides a continuous measure of sample quality, rather than a binary classification.

- Wasserstein GAN referred in citation 24: WGANs are a type of GAN architecture that uses the earth mover's distance to measure the similarity between the generated and real data distributions. Unlike traditional GANs, which use a discriminator network to distinguish between real and fake samples, WGANs use the earth mover's distance to compute a loss function that measures the distance between the generated and real data distributions. This allows for more stable and efficient training, as the loss function is less sensitive to the choice of hyperparameters and can be optimized using a variety of techniques, such as gradient descent. Additionally, WGANs can be more interpretable than traditional GANs, as the loss function provides a continuous measure of how well the generated samples match the desired distribution.

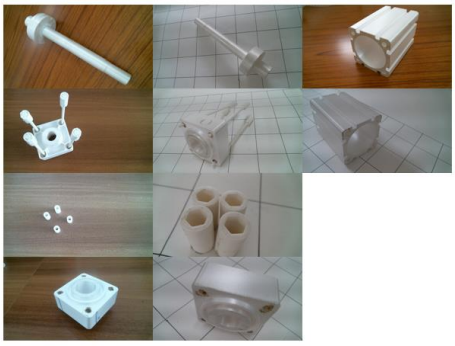
**Image-to-Image Translation:** The purpose of the image-to-image (I2I) translation methods is to renovate the visual illustration of an image with the new visual representation. The GAN based Pixel-to-Pixel (pix2pix) referred in citation 25 translation method adopts a supervised learning technique for I2I translation. The generator of pix2pix translates the source image into the target image based on the condition applied, and the discriminator confirms whether the used condition is meet-up by considering the pixel-wise loss. Similarly, Perceptual Adversarial Networks (PAN) referred to in citation 13 introduced another method for I2I where the perceptual loss is minimized instead of minimizing the pixel loss. Cycle-GAN referred to in citation 14, and Disco-GAN referred to in citation 26 introduced I2I methods for unpaired training data. They have two different GAN coupled together; one translates style from one domain to another, and another does the inverse. Similar to previous unpaired image translation methods, Wasserstein Distance-Dual GAN (WDGAN) referred to in citation 27 method used Wasserstein distance instead of a sigmoid cross entropy generative adversarial loss in addition to cyclic reconstruction loss. Star-GAN referred to in citation 28 is another unsupervised I2I method that uses a single model for multi-domains I2I translation. Star-GAN has promising results on expression synthesis and facial attribute transfer tasks. Unsupervised I2I Translation (UNIT) referred to in citation 29 combines GAN referred to in citation 30 for the generation of the corresponding image in two domains through shared-latent space constraints and VAE referred to in citation 31 for an edited image with the input image in the relevant fields. The limit of the UNIT is that they fail to generate different results. To deal with this curb, Multimodal UNIT (MUNIT) referred to in citation 32 generates different results from a given basis domain image. The MUNIT first decomposed the latent space of image into style and content codes and then recombined content code in the target style space with a random style code. Diverse I2I Translation via Disentangled Representations Image-to-Image (DRIT) referred to in citation 33 model is a multi-model generation framework that can generate different results with un-paired training data. DRIT purpose-specific disentangled representation, which enables the efficient generation of multi-modal outputs. Likewise, DRIT++ referred to in citation 34 expands the DRIT by 1) integrating sample diversity mode-seeking maximum likelihood, and 2) generalizing the two-domain structure to tackle multi-domain I2I translation issues.

## 3.1. Procedures

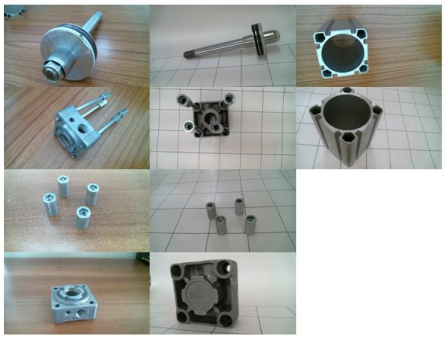
**Dataset Preparation:** The study aims to generate realistic images resembling metal components. An alternative approach involves using VR models or creating 3D printed components, yet these methods fail to capture the precise texture of metal parts. Therefore, all images were categorized into two classes: Class A, comprising 3D printed and synthetic images, and Class B, comprising images of actual metal components across five different types. Images were gathered for both white and dark backgrounds across synthetic and 3D printed categories, ensuring a balanced dataset, as summarized in Table 2. The dataset was designed to be background-agnostic, incorporating images from both light and dark backgrounds. Figure 3 displays the representative summary of both classes. Meanwhile, Figure 4 illustrates the histogram of image intensities for the training and test splits of the ILD dataset. It's evident from Figure 4 that the mean intensity of the dataset is distributed within the range of 60 to 250 for both the training and test splits. However, images with intensities lower than 60 (indicative of low-light conditions) and greater than 130 (indicative of high illuminative conditions) are present in both the train and test splits.

Table 2: Summary of the metal, 3D printed, and synthetic images in the dataset

| Class | # Images with White Background | # Images with Dark Background | # Components with White Background | | | | | # Components with Dark Background | | | | | Toal Images |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | |  |  |  |  |  |  |  |  |  |  | |
| **Train** | | | | | | | | | | | | | |
| **Metal** | 638 | 628 | 126 | 124 | 116 | 144 | 128 | 120 | 130 | 134 | 123 | 121 | **1266** |
| **3d-Printed** | 738 | 674 | 164 | 151 | 130 | 143 | 150 | 134 | 148 | 114 | 138 | 140 | **1417** |
| **Synthetic** | 600 | 600 | 120 | 120 | 120 | 120 | 120 | 120 | 120 | 120 | 120 | 120 | **1200** |
| **Test** | | | | | | | | | | | | | |
| **Metal** | 150 | 166 | 28 | 26 | 35 | 33 | 28 | 33 | 38 | 26 | 32 | 37 | **316** |
| **3d-Printed** | 168 | 182 | 43 | 32 | 26 | 35 | 32 | 39 | 37 | 31 | 38 | 37 | **350** |



(a) Class A



(b) Class B

Figure 3. Representative images of different components with light and dark background. (a) Class A contains synthetic, and 3D printed components, (b) Class B contains images of metal components.

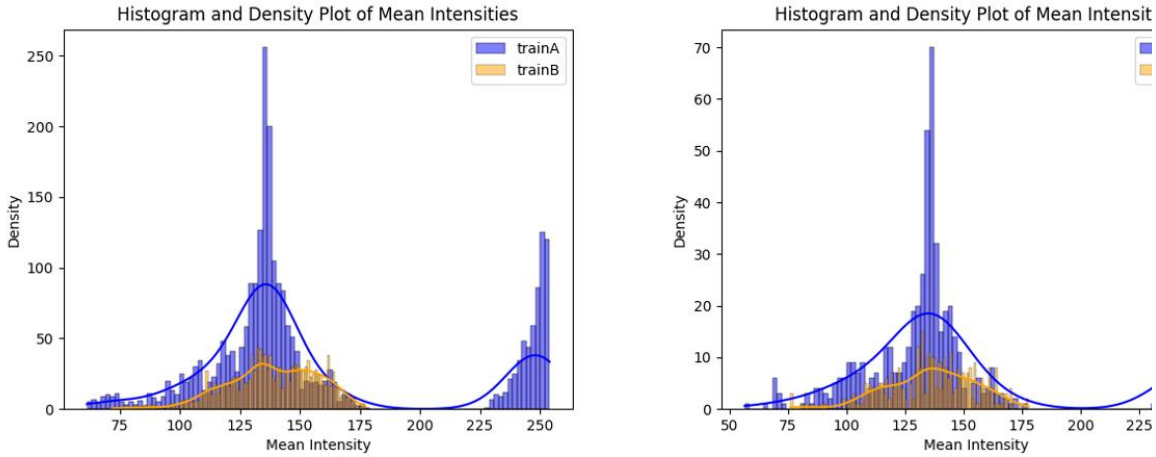



Figure 4. Mean Intensity histogram plot for both training and test dataset.

## 3.2. Description of the Models

The three selected models operate on unpaired datasets, enabling image translation across domains without necessitating paired data. In this context, regardless of the specific models, the generator produces images akin to metal types using synthetic and 3D-printed images (Class A), while the discriminator distinguishes between generated and real images (Class B). Details about each model are outlined in the subsequent paragraphs.

CycleGAN is a type of Generative Adversarial Network (GAN) that is designed to perform cycle-consistent image-to-image translation. Here are some important features of CycleGAN:

- Cycle-consistency: It is trained to translate an image from one domain to another domain and then back to the original domain, while preserving the original image's content and structure. This cycle-consistency is achieved by using a pair of generators and discriminators, one for each domain, and a cycle-consistency loss function that encourages the generated images to be like the original image.
- Domain adaptation: It can be used for domain adaptation, where the goal is to translate images from one domain to another domain that has different characteristics, such as different lighting, pose, or style.
- Image-to-image translation: CycleGAN can be used for image-to-image translation tasks, such as translating a photo of a cat to a painting or converting a black-and-white photo to color.
- Preservation of image content: It is trained to preserve the content of the original image, such as the object's shape, color, and texture, while applying the desired transformation.

DRIT++ is a state-of-the-art image-to-image translation model that builds upon the original DRIT (Deep Image-to-Image Translation) model. Here are some important features of DRIT++:

- Improved architecture: DRIT++ uses a modified version of the original DRIT architecture, with several improvements that enhance its performance. These improvements include using a multi-scale feature fusion module, a spatial attention module, and a new loss function that encourages the model to produce more realistic and diverse outputs.
- Multi-scale feature fusion: DRIT++ uses a multi-scale feature fusion module that allows it to capture features at different scales, which helps to preserve the details of the input image. This module combines low-level, middle-level, and high-level features to form a more comprehensive feature representation, leading to better performance.
- Spatial attention: DRIT++ uses a spatial attention module that helps the model focus on the most relevant regions of the input image when generating the output. This attention mechanism allows the model to produce more accurate and realistic translations, especially when the input and output images have different sizes or aspect ratios.
- Improved loss function: DRIT++ uses a new loss function that combines the traditional adversarial loss with a diversity loss and a reconstruction loss. The diversity loss encourages the model to produce diverse outputs, while the reconstruction loss encourages the model to preserve the content of the input image.

The UNIT GAN model is a type of Generative Adversarial Network (GAN) that has several main features that distinguish it from other GAN models. Here are some of the main features of UNIT GAN:

- Unified Training: UNIT GAN combines the training of the generator and discriminator into a single, unified process. This contrasts with traditional GANs, where the generator and discriminator are typically trained separately.
- Non-Adversarial Loss: UNIT GAN uses a non-adversarial loss function, which means that the generator is not trained to fool the discriminator. Instead, the generator is trained to produce high-quality images that are similar to the real data.
- Noise Injection: UNIT GAN injects noise into the generator network to create diversity in the generated images. This noise is sampled from a learned distribution and helps to prevent the generator from producing overly repetitive or modeled outputs.
- Self-Attention: UNIT GAN uses self-attention mechanisms in the generator and discriminator networks. This allows the model to focus on specific regions of the image and generate more detailed and realistic features.
- Progressive Residual Learning: UNIT GAN uses a progressive residual learning strategy, which means that the generator learns to produce images in a progressive, step-by-step manner. This allows the model to generate high-quality images with fewer training examples.

## 3.3. Evaluation Metrics

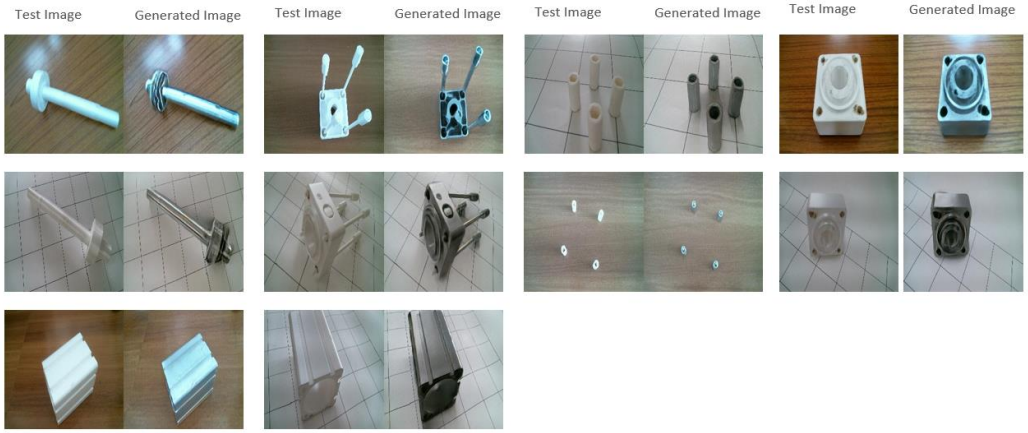
To compare the performance of these three models, we used two evaluation metrics, Inception score, and Fréchet-Inception distance. Table 3 summarizes these two metrics.

Table 3: Summary of the evaluation metrics

| Metric Name | Description | Interpretation |
|---|---|---|
| Inception Score | Evaluates the diversity and quality of generated images. Considering no. of different and meaningful predicted class labels. | Higher is better. |
| Fréchet - Inception distance | Measures the similarity between the distribution of real and generated images in a feature space derived from an Inception network. | Lower is better. |

## 3.4. Experimental Results

The proposed models are implemented using PyTorch, maintaining the original hyperparameters. Evaluation involves training the Inceptionv3 model on the component dataset and subsequently assessing it following standardized procedures. Figure 5 shows the qualitative images as output of all three models. Table 4 compares the performance based on FID and inception score.




(a)

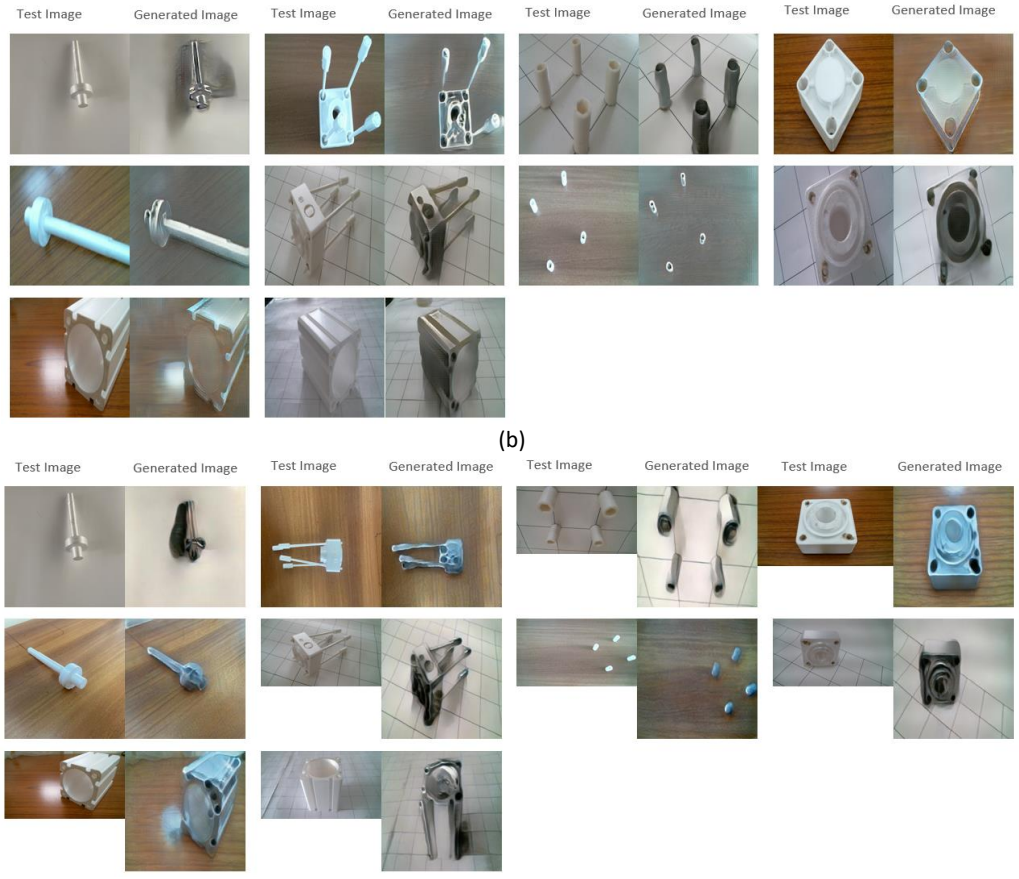



Figure 5. Example results on test images. (a) CycleGAN, (b) DRIT++, (c) UNIT.

Table 4: Comparison between different models based on FID and Inception score

| Model's Name | FID Score | Inception Score |
|---|---|---|
| CycleGAN | 0.0017 | 4.7917 |
| DRIT++ | 0.0533 | 4.6270 |
| UNIT | 0.0079 | 4.6799 |

# 4. Evaluating Model with Custom Dataset

We discovered that Yolov8 outperforms other object detection models in the MS COCO dataset. Nevertheless, we analysed various object detection models using a custom dataset. One of the primary challenges with supervised learning models is data collection and labelling. Evaluating rendering techniques enables the generation of synthetic data and automatic labelling, reducing the time and manpower required for dataset creation. In this study, three datasets were created to assess the effectiveness of the object detection model on a custom dataset.

1. Real Image
2. Synthetic Image
3. Combination of real and synthetic image

## 4.1. Synthetic Data Generation

We created synthetic data using UNITY, which is the game engine. We created the virtual environment of factory and assembly components setup in unity. The assembly components, designed in SolidWorks, were

saved in Unity acceptance format (fbx, obj). Subsequently, we imported these assembly components into Unity. To facilitate labeling, we employed the "Perception package," which can be freely downloaded from the Unity Asset Store. This package automates the labeling process for Unity components. To enhance data generation, we simulated the environment and manipulated the Unity camera to capture images from various angles. During simulation, the Perception package automatically labels components and saves the results. The generated synthetic data is shown in fig3 below.

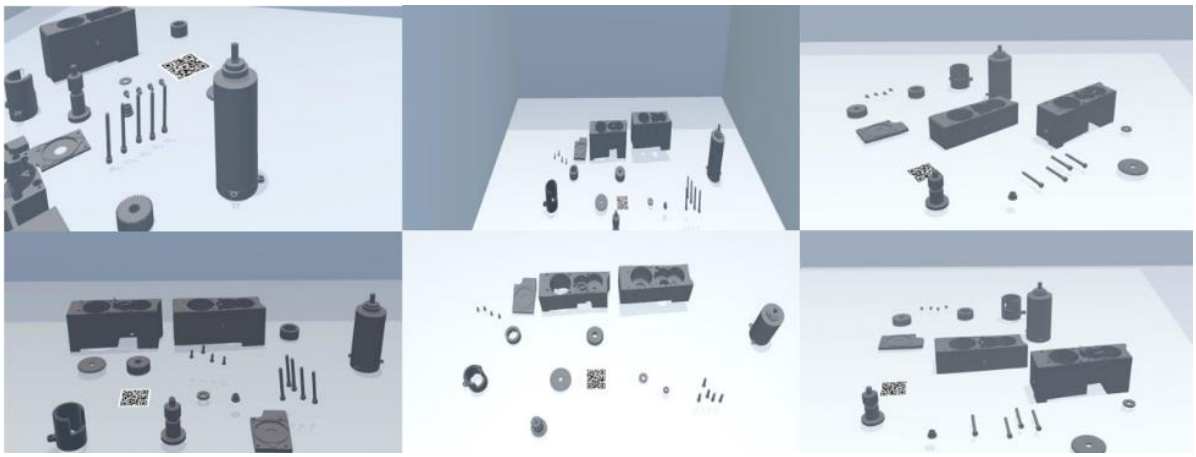


Figure 6: Synthetic Data Generated in Unity

## 4.2. Real Data Generation

We captured images of the assembly components in the real world using both the HoloLens camera and an external camera. We took pictures with different backgrounds, angles, and heights. Afterward, we applied augmentation techniques to create a variety of data. The collected real data is shown in figure 7.

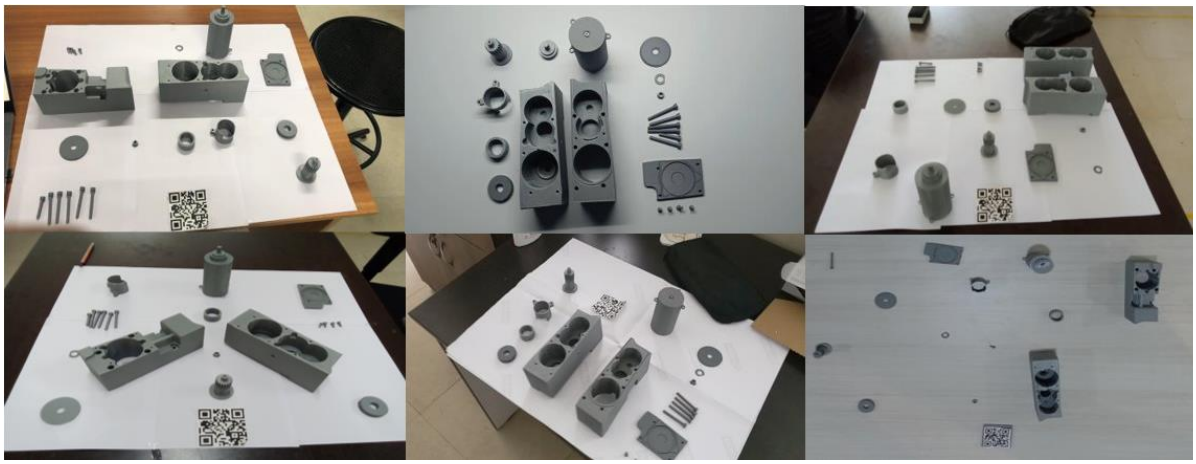


Figure 7: Real Dataset

## 4.3. Mixed Dataset

Training the model with both real and synthetic data has distinct advantages and disadvantages. The advantage of using real data lies in improving the model's object detection accuracy. However, the drawback includes time-consuming processes, the need for manpower, and the possibility of errors in labelling. On the other hand, the advantage of utilizing synthetic data is that it can be generated easily, requiring less time and manpower. Nonetheless, the challenge lies in incorporating real-world attributes into synthetic data. Therefore, we created a

mixed dataset, combining both real and synthetic data, which includes real-world attributes and reduces time and manpower requirements. The mixed dataset comprises 50% synthetic data and 50% real data.

## 4.4. Evaluation

From the existing study, Yolo worked better compared to two stage model. So that, we chose Yolo to object detection model. We compared three different version of Yolo models with custom dataset. We chose Yolo's v5, v7, and v8 for comparative study. We trained three models with three datasets: synthetic data, real data, and a combination of both real and synthetic data. We trained all three models with 600 images of real data, 600 images of synthetic data, and 600 images of mixed data. Then we evaluate all the model with 60 images of the real data. The mixed dataset comprises 50% synthetic data and 50% real data. We trained the model with 30 epochs. The result is shown in table 5 below. From the study, we found that Yolov8 giving better Mean Average Precision (mAP) value in all datasets. The mAP value of the Yolo model with mixed data set gave better result compared to other datasets.

Table 5: Compare different version of Yolo model with custom dataset

| Training Dataset Type | YOLO version | Dataset Size | Test Data | mAP at IoU:0.5 | Epochs |
|---|---|---|---|---|---|
| | | | | | |
| Real | v5m | 600 | Real | 0.755 | 30 |
| | v7 | 600 | Real | 0.856 | 30 |
| | v8m | 600 | Real | 0.887 | 30 |
| Syn | v5m | 600 | Real | 0.410 | 30 |
| | v7 | 600 | Real | 0.429 | 30 |
| | v8m | 600 | Real | 0.512 | 30 |
| Mixed (50-50) | v5m | 600 | Real | 0.759 | 30 |
| | v7 | 600 | Real | 0.809 | 30 |
| | v8m | 600 | Real | 0.889 | 30 |

The mAP value is less than 0.9 while analyse the model with 600 images. For improving the result, we increased the dataset by augmentation. Image augmentation constitutes a crucial step in the preprocessing pipeline, wherein various transformations are applied to the existing images within the dataset. This procedural augmentation serves the purpose of enhancing the model's capacity for generalization, thereby improving its efficacy when presented with previously unseen images. The various augmentation used in datasets are:

1. Vertical - Horizontal flipping
2. +- 20-degree rotational adjustments
3. Random cropping - 5% of image size
4. Introducing random noise – 2% noise
5. Cutout operations - 3 boxes 5% size of images
6. Mosaic integration

For improving the mAP value, we increased the dataset by using augmentation technique and adding synthetic images. The result is shown in table 6 below.

Table 6: Yolo model comparison

| dataset | YOLO Version | Data set Size | Test data | mAP [IoU:0.5-0.95] | Epochs | inference time |
|---|---|---|---|---|---|---|
| **Original Dataset** | | | | | | |
| | v5m | 516 | Real | 0.734 | 30 | 8.9ms |

| | | | | | | |
|---|---|---|---|---|---|---|
| | v7 | 516 | Real | 0.789 | 30 | 16.8ms |
| | v8m | 516 | Real | 0.884 | 30 | 7.8ms |
| Augmented | | | | | | |
| | v5m | 1500 | Real | 0.756 | 50 | 8.9ms |
| | v7 | 1500 | Real | 0.814 | 50 | 16.8ms |
| | v8m | 1500 | Real | 0.919 | 50 | 7.8ms |
| | | | | | | |
| **Synthetic dataset** | | | Test data | mAP [IoU:0.5-0.95] | | inference time |
| Original Dataset | | | | | | |
| | v5m | 800 | Real | 0.415 | 30 | 8.9ms |
| | v7 | 800 | Real | 0.433 | 30 | 16.8ms |
| | v8m | 800 | Real | 0.529 | 30 | 7.8ms |
| Augmented | | | | | | |
| | v5m | 2400 | Real | 0.424 | 50 | 8.9ms |
| | v7 | 2400 | Real | 0.510 | 50 | 16.8ms |
| | v8m | 2400 | Real | 0.586 | 50 | 7.8ms |
| | | | | | | |
| **Mixed dataset** | | | Test data | mAP [IoU:0.5-0.95] | | inference time |
| Original Dataset | | | | | | |
| | v5m | 1300 | Real | 0.862 | 50 | 8.9ms |
| | v7 | 1300 | Real | 0.867 | 50 | 16.8ms |
| | v8m | 1300 | Real | 0.898 | 50 | 7.8ms |
| Augmented | | | | | | |
| | v5m | 3700 | Real | 0.901 | 80 | 8.9ms |
| | v7 | 3700 | Real | 0.905 | 80 | 16.8ms |
| | v8m | 3700 | Real | 0.948 | 80 | 7.8ms |

The speed of the object detection depends on the hardware which we are using to run the object detection model. We compared the above-mentioned hardware with the Yolov8 model. The result is shown in table 7 below. GeForce RTX 3090 is given better performance compared to RTX 3060 and Jetson Nano.

Table 7: Hardware comparison

| MODELS TRAINED | MODEL PARAMETERS | FPS (CYCLE TIME) |
|---|---|---|
| **YOLOV5M** | 21.2M | RTX 3090 – 13 FPS<br>RTX 3060 – 5 FPS<br>Jetson Nano – 0.5 FPS |
| **YOLOV7** | 36.9M | RTX 3090 - 9FPS<br>RTX 3060 – 4 FPS<br>Jetson Nano – 0.3 FPS |
| **YOLOV8M** | 25.9M | RTX 3090 – 11 FPS<br>RTX 3060 – 4 FPS<br>Jetson Nano – 0.3FPS |

Additionally, we conducted a comparison of the mixed dataset with different object detection models, including DETR and various versions of YOLO is shown in table 8 below. The analysis revealed that Yolov8 works better than two stage method.

Table 8: Comparison of different object detection models

| MODELS | Training Time | GPU Mem Utilized | FLOPS (G) | Inference Time | FPS | Epoch | mAP % [IoU: 0.5] |
|---|---|---|---|---|---|---|---|
| YOLO V5 | 2.93 hrs | 2.9Gb | 48.4 | 89.28ms | 11.2 | 80 | 92.8 % |
| YOLO V7 | 8.8 hrs | 4.2Gb | 105.3 | 24.0ms | 41.6 | 80 | 90.8 % |
| YOLO V7 X | 10.85 hrs | 4.1Gb | 189.2 | 39.0ms | 25.64 | 80 | 91 % |
| YOLO X | 1.75 hrs | 7.8Gb | 73.8 | 90.49ms | 11.1 | 80 | 85.30 % |
| DETR | 11 hrs | 7.42Gb | 85.5 | 2941ms | 0.34 | 150 | 90.0% |
| YOLO v8 | 2.8 hrs | 4.42Gb | 79.1 | 13.0ms | 76.9 | 60 | 95.4 % |

The figure 8 show Yolov8 gives better mAP value for each component compared to Yolov5, Yolov7, and Yolov8. The figure 9 show precision recall curve of Yolov8 model.

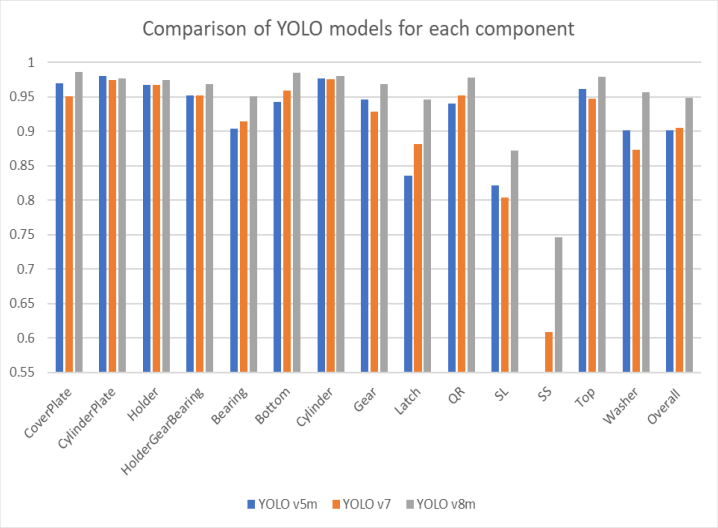



Figure 8: Every single Component training accuracy using different models

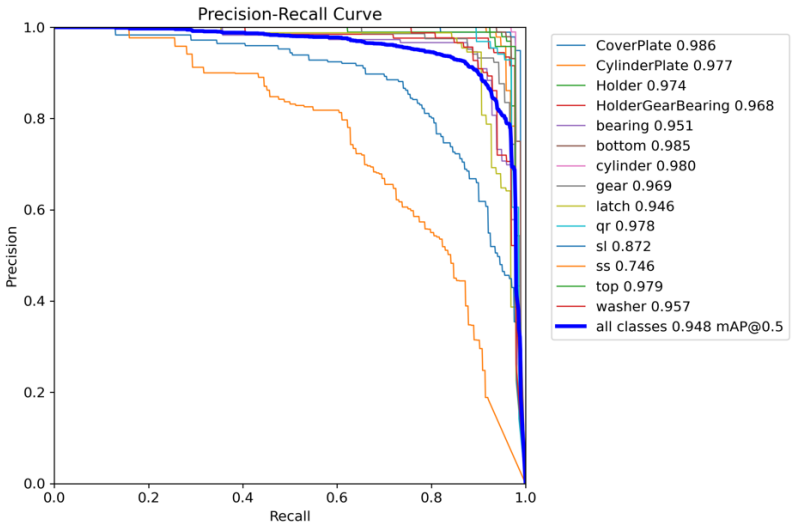



Figure 9: Precision-Recall Curve for YOLOv8 Training on RTX 3090 System

From the table study, we interpreted the mixed dataset gave better result compared to other datasets. From the analysis, we could not find significant differences in mAP value while train the model with real dataset and mixed dataset. However, creating real dataset took more time and manpower for capturing and labelling. In case, synthetic data generation is easy once the environment is created. So that, using mixed dataset reduced the manpower and time. We chose yolov8 and mixed dataset for implementation because it works better compared to another model and dataset.

# 5. Object Detection and Tracking

Object tracking involves a deep learning model that detects each object in its initial state, generates a unique ID for each object, and then tracks each detected object as it continuously moves around the frame. Zhang et al. (2022), please refer to citation 1, compared various object tracking models, as shown in figure 10. They found that Byte Track performs better with high fps compared to other tracking models. The advantage of using object tracking models are,

1. Improved Detection without always depending on perfect detection of Components.
2. Tracking can improve the Overall latency.

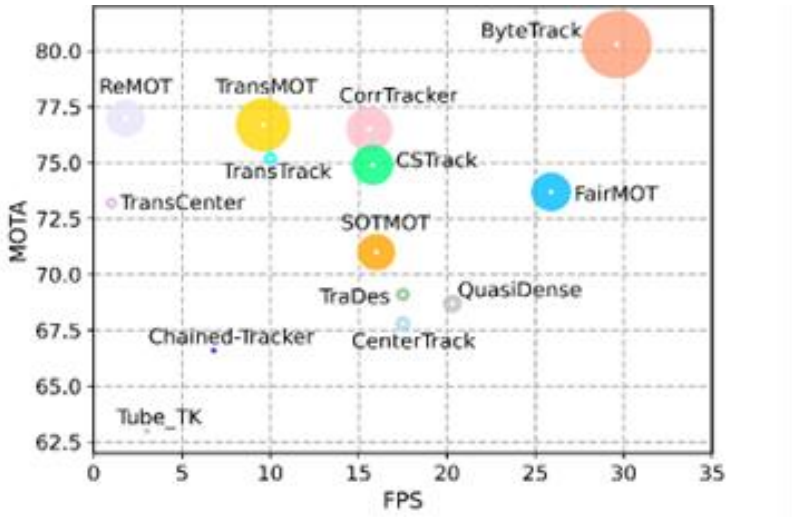



Figure 10: Different State of the Art Tracking Methods with Multi-Object Tracking Accuracy

We analysed the four object tracking methods, as shown in the table 9. The results indicate that Byte Track provides higher accuracy and a greater frames-per-second (fps) compared to other methods. In this study, we selected Byte Track as the object tracking algorithm. The figure 11 shows the byte track detects all the objects.

Table 9: Comparison of different object tracking models

| **Methods Adopted** | **Inference** |
|---|---|
| **April Tags** | Computation Intensive for components>5<br><br>Uses Different April Tags<br><br>Tracking April-Tags for each component is not possible. |
| **TAPNET – Visual Tracking** | Computation Intensive<br><br>3-4 FPS<br><br>Tracking Accuracy is good |
| **Kalman Filters with added Image enhancements** | Added CPU Computation cost.<br><br>1-2 FPS<br><br>Tracking Accuracy is poor |
| **Byte Track** | Uses YOLO Detections as input (simpler implementation)<br><br>4-5 FBS |

| | |
|---|---|
| | Tracking Accuracy is better |

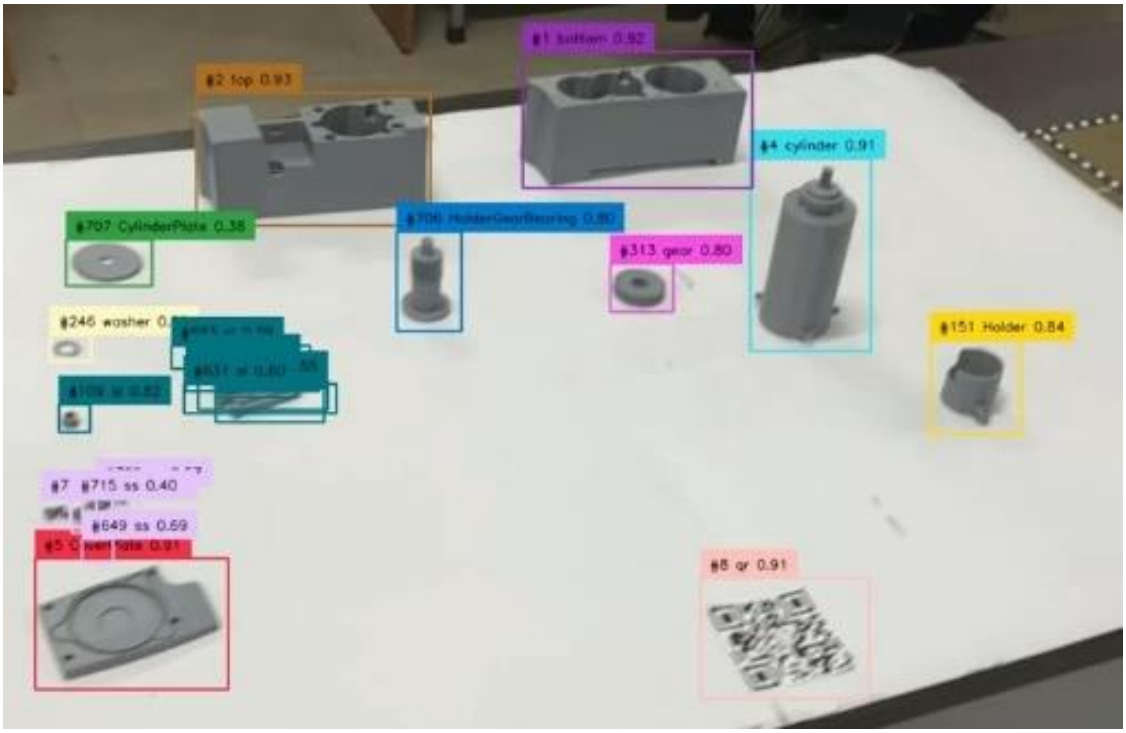



Figure 11: Object Detection

## 6. Integrating Computer Vision with Mixed Reality Environment

The proposed method utilizes HoloLens 2 for experiencing the MR environment. We developed the application in Unity with the assistance of the Mixed Reality Toolkit (MRTK). Subsequently, we deployed it on the MR device. The holograms are created either in SolidWorks or Unity. An external camera, positioned opposite to the user and covering the entire workspace, is employed to capture the assembly workspace. The captured image is fed to the object tracking model to obtain object position values. These values are then transmitted to the regression model to map pixel values to the real world. We utilized a single QR code as a

marker to establish a common reference point for both the external camera and HMD. The HTTP server/client protocol facilitates communication between HoloLens 2 and the server. The object tracking model operates at a speed of 4 to 5 frames per second. Four ArUco markers are employed to obtain a perspective view of the assembly workspace as shown in figure 13.b below, and the resulting image is sent to the object tracking model. The object tracking model determines the positions of objects with markers, and these values are relayed to the regression model to ascertain the objects' positions in the real world. Subsequently, the adjusted holograms are sent to the HoloLens for display. Figure 12 below illustrates the integration of the object tracking model with the external camera.

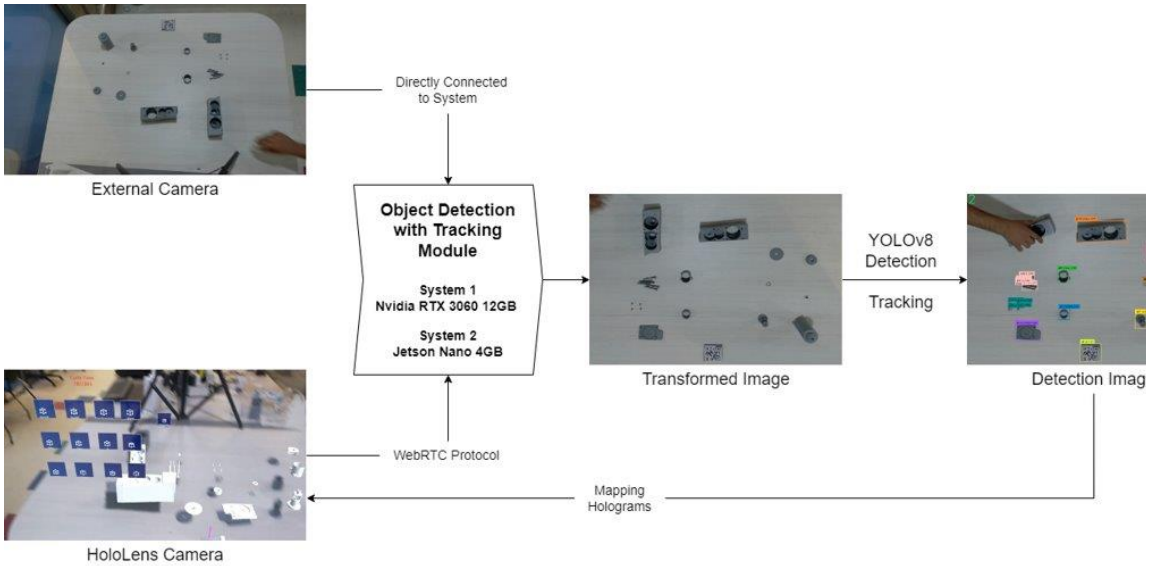
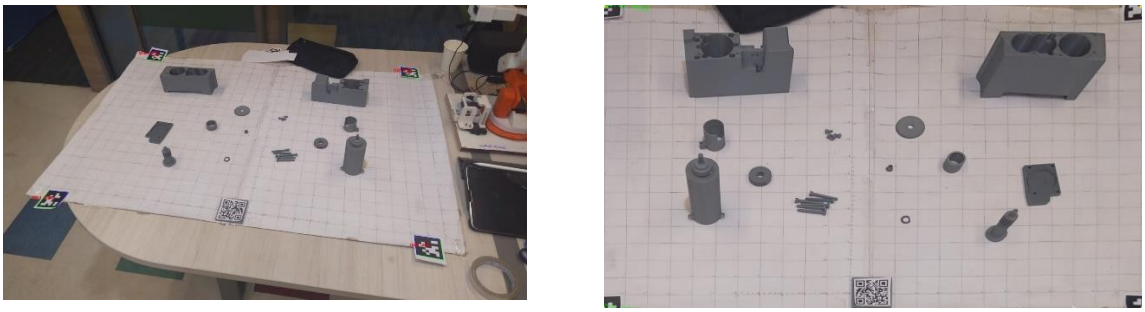



Figure 12. Integration of Computer Vision Module for Mixed Reality Environment

We designed the user interface to initiate assembly instructions, which comprises a list of buttons, input fields, and information panels. In the proposed method, the user receives instructions through changes in hologram colour, a simulation of assembly, and a virtual hand indicating the object to be picked. We implemented a collision detection technique to identify faulty object detection. If the user picks a faulty object at any step, the hologram colour will change to red, indicating that the faulty object has been picked up.



(a) (b)

Figure 13: (a) Detection of April Tags before Transformation, (b) After Transformation

During the assembly process, users can receive both sequential assembly instructions and partial assembly instructions. In the case of sequential assembly instructions, a single instruction is provided to the user when they click the "Next" button. For partial assembly instructions, clicking the "Next" button exposes the user to all possible assembly instructions corresponding to the current state of the assembly process. Once the user completes the task in the current state, they can click the "Next" button to access information about the next state of the assembly process. The table 10 below explains the list of UI buttons and voice commands used to activate specific functions.

Table 10: Interactive commands

| Button Name | Voice Command | Process Initiation |
|---|---|---|
| Start QR | Start QR | Initialize QR code tracking |
| Stop QR | Stop QR | Stop QR code tracking |
| Photo Capture | Photo Capture | Click the picture using HoloLens camera |
| Object Tracking | Object Tracking | Initialize the object tracking algorithm |
| Next | Next | Get to the next assembly step information |
| Previous | Previous | Go back to the previous assembly step information |
| IP | Connection to server | Establish connection between server and HoloLens |

The user engages with the developed MR applications by clicking virtual buttons and using voice commands. The user inputs the server's IP address in the designated field and clicks the "IP" button or uses the corresponding keyword to establish a connection between HoloLens2 and the server. When the user interacts through a speech command, the application repeats and displays the command if understood; otherwise, the user repeats the command, enhancing user awareness while using speech commands. We employed the "Text-to-Speech" class to enable voice output.

## 6.1. Image to MR Mapping

When utilizing an external camera to capture images and input them into the object detection model, it is crucial to map the model's output to the MR environment for successful implementation. The challenges associated with using the HoloLens camera and providing feedback to the object detection models are as follows:

- Production of noisy images due to head movement
- Increased computational load due to spatial perception requirements for mapping.
- Manual input required to enhance mapping accuracy.

To mitigate the issue of noisy images when using an external camera, we opted to fix the camera on the assembly table, covering the entire working space. ArUco markers were employed to capture a perspective image of the assembly table, reducing the likelihood of sending irrelevant objects surrounding the assembly table to the object detection model.

We proposed using linear regression to map the pixel values outputted by the object detection model to the real world. To create the training dataset, we positioned assembly components at different locations on the assembly table and captured images, measuring the components' positions with respect to QR codes. These images were then inputted into the object detection model to obtain pixel values corresponding to marker pixel values, creating a dataset with real-world position values and their corresponding pixel values. For model testing, a new image was captured and sent to the object detection model and mapping algorithm. The average error in X and Y directions are 8.58mm and 7.03mm. The mapping error in both X and Y direction is shown in figure 14.

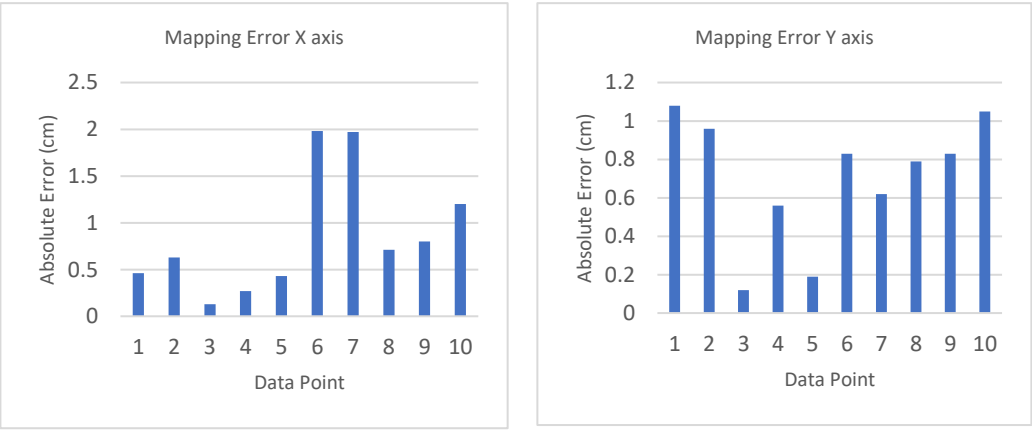



Figure 14. Mapping error on both X and Y axis

### Summary of Results

In this project, we chose mixed dataset to train the object detection model because it gave better result. In addition to that, creating mixed dataset is easier compared to real dataset. It took less time and manpower. Yolov8 selected for object detection model. It detected each component with better accuracy and less latency compared to other models. Byte track is used in this project object tracking. It took Yolov8 detection as input. So that it is easy to implement. It gives better accuracy and run with more FPS compared to other object tracking model. GeForce RTX 3090 is given better performance compared to RTX 3060 and Jetson Nano. But Jetson Nano is portable, so we can use it where speed is not essential. In the generation of synthetic data, the results indicate that CycleGAN outperforms other models in producing realistic images.

## 7. User Study

We evaluated the usability of the proposed method through a user study. We collected qualitative and quantitative data to analyse the effectiveness, cognitive load and usability of the system. The user study aimed to analyse the effectiveness of the MR based system to assist assembly worker. We enabled external camera to capture the assembly workspace. The camera feed gave to computer vision algorithm to adjust the holograms to respective real components. We compared the proposed method with video-based assembly guidance. In video-based assembly guidance, prerecorded video of the assembly sequence played in the screen. The user looked at the video, finished the assembly process (fig 15).

Figure 15. Experiment Set Up

Twelve participants with an age range of 24 to 29 years volunteered for the study. All the participants got acquainted with the MR based and video-based assembly tasks before the actual study. All the components were placed in a random position on the workspace. We presented the two scenarios to the user in a random order to avoid any subjective or order bias. The participants followed the presented steps to finish the assembly task as given instructions. We recorded the time taken to complete the task for two methods. Once the participants finished the task, we asked participants to answer System Usability Scale (SUS) and NASA Task Load Index (TLX) questionnaires to record the subjective preference and perceived cognitive load during the study. Participants took sufficient number of breaks after each scenario and before moving onto the next scenario to avoid any induced fatigue.

We analysed the task completion time between two methods is shown in figure 15 and figure 16 below. The task completion time was 14% faster in mixed reality based system. The minimum task completion time of the proposed and baseline methods are 130 seconds(sec) and maximum task completion time are 247 and 295 sec

(figure 16). We collected System Usability Scale (SUS) and NASA Task Load Index (TLX) questionnaires to understand their subjective preference and perceived cognitive load. From the results, we inferred that developed applications effectively guide the user to complete the assembly task (figures 17 and 18). We observed a mean TLX score of 29.33 ± 19.45 and a mean SUS score of 74.58 ±24.32 across all participants in the proposed method. We observed a mean TLX score of 31.45 ± 17.55 and a mean SUS score of 72.71 ± 17.39 across all participants in the video-based guidance. The TLX was lower and SUS score was higher in mixed reality based system than video based system. We undertook Cronbach's alpha test on the responses of the designed questionnaire to check the consistency of the responses from 12 participants. We obtained the alpha value to be 0.8, which is close to the benchmark value of 0.7 to infer consistency of the responses. We asked set of questionnaires to participants is shown in table 11. From the questionnaires analysis, we interpreted MR based instructions can be understandable and distinguishable. The participants preferred to interact with speech command than virtual buttons. Summarizing from the questionnaire responses, we observed that the participants found the proposed system usable with various components of it well integrated and enabling them to perform complex assembly tasks. The differences in task completion times, TLX and SUS scores are not found to be statistically different in paired t-tests at p=0.05.

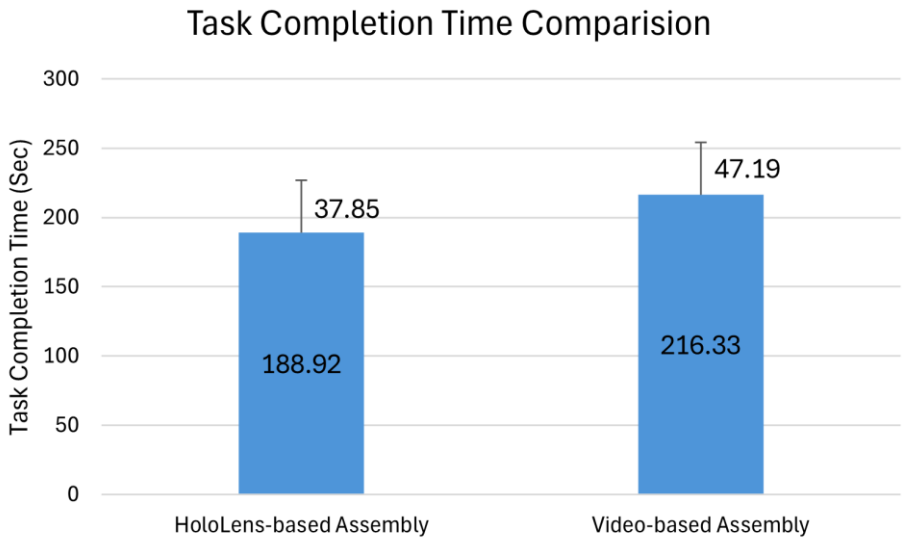



Figure 16. Task Completion Time comparison of two methods

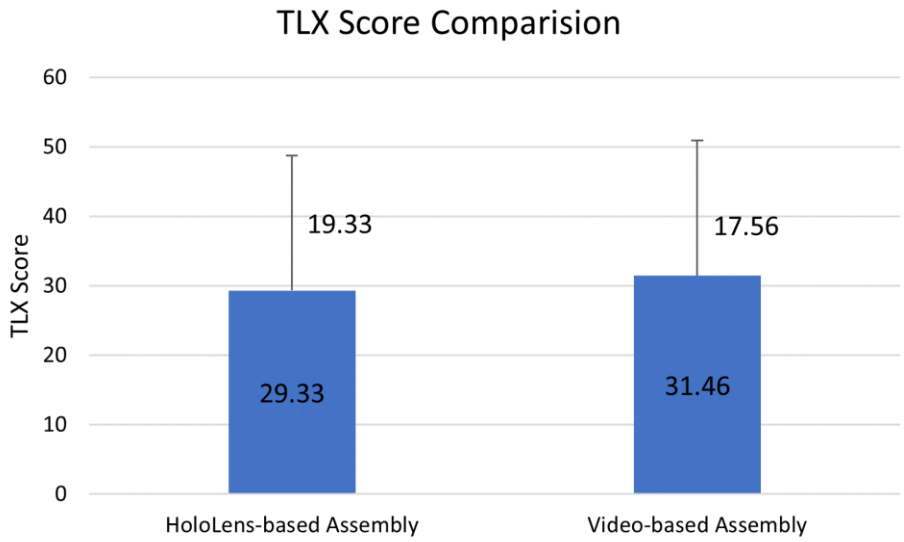



Figure 17. TLX score comparison of two method

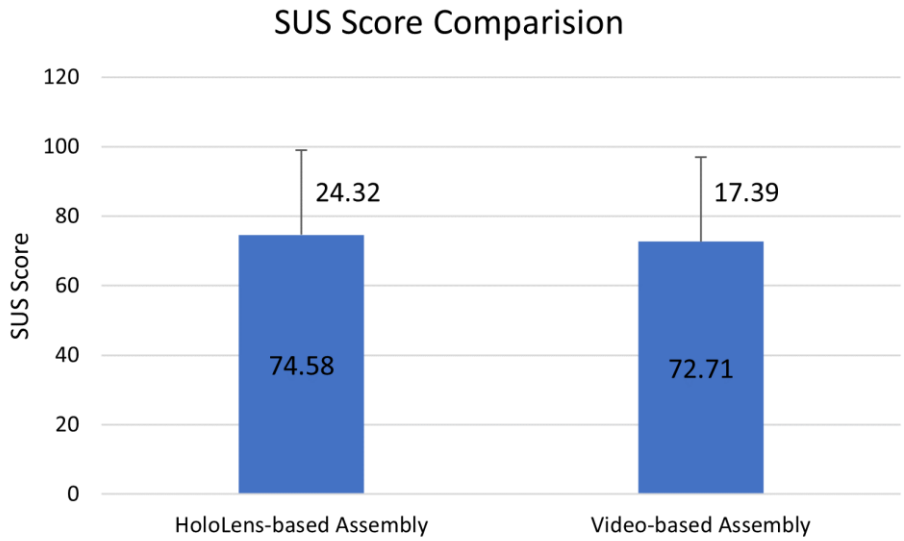



Figure 18. SUS score comparison of two methods

*Table 11: List of Questionnaire*

| ***Questionnaire*** | **Score** |
|---|---|
| *Using Mixed Reality, Instructions can be understood* | 8.67 |
| *Using different colour and animation, instructions can be perceived* | 8.67 |
| *After projecting the virtual object on the real object, the real and virtual object can be distinguished* | 8.5 |
| *Focusing on the task with the help of Head Mounted Display is* | 7.33 |
| *Users can interact with the buttons in User Interface* | 7.33 |
| *Users can interact with speech command* | 8.5 |
| *Projection final stage of each step helps to verify and inspect each step* | 8.67 |
| *While wearing Head Mounted Display, Real world can be visualized* | 8.17 |

# 8. Conclusions

The research significantly contributes to XR technology by introducing innovative methods for rendering instructions and ensuring accurate object detection in assembly tasks. By utilizing a combination of synthetic and real-world data, the study addresses challenges in training object detection models and emphasizes the importance of synthetic data for model accuracy. Specifically, the paper delves into continuously adjusting holograms based on the position of real-world objects using an external camera, examines the effectiveness of synthetic data in object detection, and assesses the impact of providing instructions through various modalities. Furthermore, the study compares YOLOv8 and ByteTrack for object detection and tracking, respectively, highlighting their superior accuracy and efficiency in component detection and tracking compared to other models. Additionally, the research evaluates the performance of different GPUs, emphasizing the balance between performance and portability. The findings demonstrate that CycleGAN excels in generating realistic synthetic images, showing potential for enhancing data generation in the field. Overall, the research amalgamates advancements in XR technologies and computer vision techniques to optimize instructions and object detection, offering valuable insights into improving assembly tasks and energy management processes.

This report contributed towards include continuously adjusting holograms according to real-world object positions using an external camera, analysing the effectiveness of synthetic image usage in object detection, and evaluating instructions' effectiveness across different modalities. However, challenges persist in bridging the realism gap between synthetic and real data, prompting ongoing research in Image-to-Image Translation GAN models to

generate more realistic data. The study also emphasizes the importance of combining real and synthetic datasets for object detection model training, citing improved results from a mixed dataset compared to solely real datasets. Furthermore, the analysis identified the most effective object detection model, YOLOv8, and object tracking model, ByteTrack, for better accuracy and efficiency. Finally, the performance of different hardware options for various requirements was assessed, highlighting their respective advantages and trade-offs.

# 9. References


1. Zhang, Yifu, et al. "Bytetrack: Multi-object tracking by associating every detection box." *European Conference on Computer Vision*. Cham: Springer Nature Switzerland, 2022.
2. Wang, Chien-Yao, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023.
3. YOLOv8 - Ultralytics YOLOv8 Docs
4. Aggarwal, A., Mittal, M. and Battineni, G., 2021. Generative adversarial network: An overview of theory and applications. International Journal of Information Management Data Insights, 1(1), p.100004.
5. B. Chang, Q. Zhang, S. Pan, and L. Meng, “Generating Handwritten Chinese Characters using CycleGAN”, in the IEEE Winter Conference on Applications of Computer Vision (WACV), 2018.
6. X. Lin, J. Li, H. Zeng, and J. Rongrong, “Font generation based on least-squares conditional generative adversarial nets,” in Multimedia Tools and Applications, 2018.
7. J-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, “Generative visual manipulation on the natural image manifold,” in European Conference on Computer Vision, pages 597–613, 2016.
8. S. Nam, Y. Kim, & S. Kim, “Text-adaptive generative adversarial networks: Manipulating images with natural language,” in Proceedings of International Conference on Neural Information Processing, 2018.
9. Z. He, W. Zuo, M. Kan, S. Shan, and X. Chen, “Attgan: Facial attribute editing by only changing what you want,” in IEEE Transactions on Image Processing, 2019.
10. W. Xian, P. Sangkloy, V. Agrawal, A. Raj, J. Lu, C. Fang, F. Yu, and J. Hays, “TextureGAN: controlling deep image synthesis with texture patches,” arXiv preprint arXiv: 1706.02823, 2017.
11. W. Chen, and J. Hays, “SketchyGAN: Towards Diverse and Realistic Sketch to Image Synthesis,” arXiv preprint arXiv: 1801.02753, 2018.
12. J. Yu, X. Xu, F. Gao, S. Shi, M. Wang, D. Tao, and Q. Huang” Towards Realistic Face Photo-Sketch Synthesis via Composition-Aided GAN,” arXiv preprint arXiv: 1712.00899, 2020.
13. C. Wang, C. Xu, C. Wang, and D. Tao, “Perceptual adversarial networks for image-to-image transformation,” in IEEE Transactions on Image Processing, vol. 27, no. 8, pp. 4066–4079, 2018.
14. J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in International Conference on Computer Vision, 2017.
15. H. Lee, H. Tseng, J. Huang, M. Singh, and M. Yang, “Diverse image-to-image translation via disentangled representations,” in European Conference on Computer Vision, pp. 35–51, 2018.
16. X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, “Multimodal unsupervised image-to-image translation,” in European Conference on Computer Vision (ECCV), pp. 172–189, 2018.
17. L. Tran, X. Yin, and X. Liu, “Disentangled representation learning gan for pose-invariant face recognition,” in IEEE Conference on Computer Vision and Pattern Recognition, 2017.
18. R. Huang, S. Zhang, T. Li, and R. He, “Beyond face rotation: Global and local perception gan for photorealistic and identity preserving frontal view synthesis,” in International Conference on Computer Vision, pp. 2439–2448, 2017.
19. X. Yu, K. Sohn, X. Liu, and M. Chandraker “, Towards large-pose face frontalization in the wild,” in Proceedings of the IEEE International Conference on Computer Vision. 2017.
20. M. Mirza, and S. Osindero, “Conditional generative adversarial nets,” arXiv preprint arXiv: 1411.1784, 2014.
21. A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” arXiv preprint arXiv: 1511.06434, 2015.
22. E. Denton, S. Chintala, R. Fergus, et al., “Deep generative image models using a laplacian pyramid of adversarial networks,” in Neural Information Processing Systems, pp. 1486–1494, 2015.
23. J. Zhao, M. Mathieu, and Y. LeCun, “Energy-based generative adversarial network,” arXiv preprint arXiv: 1609.03126, 2016.
24. M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in International Conference on Machine Learning, pp. 214–223, 2017.

25. P. Isola, J-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” arXiv preprint arXiv: 1611.07004, 2016.
26. T. Kim, M. Cha, H. Kim, J. Lee, and J. Kim, “Learning to discover cross-domain relations with generative adversarial nets,” arXiv preprint arXiv: 1703.05192, 2017.
27. Z. Yi, H. Zhang and P. T. Gong, “Dualgan: Unsupervised dual learning for image to-image translation,” arXiv preprint arXiv: 1704.02510, 2017.
28. Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim and J. Choo, “Stargan: Unified generative adversarial networks for multi-domain image-to-image translation,” in IEEE Conference on CVPR, pp. 8789–8797, 2018.
29. M.-Y. Liu, T. Breuel, and J. Kautz. "Unsupervised image-to-image translation networks." Advances in neural information processing systems. 2017.
30. I. Goodfellow, J. Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengi, “Generative adversarial nets,” in Neural Information Processing Systems, pp. 2672– 2680, 2014.
31. D. P. Kingma, and M. Welling, “Max. Auto-Encoding Variational Bayes,” in Internal Conference on Learning Representation, 2013.
32. X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, “Multimodal unsupervised image-to-image translation,” in European Conference on Computer Vision (ECCV), pp. 172–189, 2018.
33. H. Lee, H. Tseng, J. Huang, M. Singh, and M. Yang, “Diverse image-to-image translation via disentangled representations,” in European Conference on Computer Vision, pp. 35–51, 2018.
34. H-Y. Lee, H. Tseng, Q. Mao, J. Huang, Y-D. Lu, M. Singh, M. Yang, “DRIT++: Diverse image-to-image translation via disentangled representations,” arXiv preprint arXiv: 1905.01270, 2019.
35. Srinivas Annambhotla, Cesar Romero and Alex Thaman. 2020. Synthetic data: Simulating myriad possibilities to train robust machine learning models. Retrieved October 7, 2020 from https://blogs.unity3d.com/2020/05/01/synthetic-data-simulating-myriad-possibilities-to-train-robust-machine-learning-models/
36. Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In International Conference on Intelligent Robots and Systems, IROS. 23–30. https://doi.org/10.1109/IROS.2017.8202133
37. Hinterstoisser, Stefan, Olivier Pauly, Hauke Heibel, Marek Martina, and Martin Bokeloh. 2019. "An annotation saved is an annotation earned: Using fully synthetic training for object detection." In Proceedings of the IEEE International Conference on Computer Vision Workshops, pp. 0-0
38. Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. 2018. Training Deep Networks With Synthetic Data: Bridging the Reality Gap by Domain Randomization. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops