MIE1075 – AI IN ROBOTICS

Group 10 – AI Shopping Assistant Robot Project Report

Zicheng Wang #1005730130
Xingjian Li #1011563096
Hankun Wang #1011796344
Sourabh Prasad #1009727217

DECEMBER 25, 2024

UNIVERSITY OF TORONTO
27 King's College Cir, Toronto, ON M5S 1A1

Project code available on GitHub - https://github.com/LesterLi33/MIE_1075

Table of Contents and Figures

ABSTRACT

The advancement of Artificial Intelligence (AI) has significantly contributed to state-of-the-art research and provided solutions to simplify various aspects of daily life. This study proposes the design and implementation of a shopping assistance robot that leverages multiple AI methodologies to elevate the customer experience and reduce operational costs. The robot is designed to interpret customer queries, provide appropriate responses, guide customers to the requested items, and physically retrieve goods upon request, offering a seamless and interactive shopping experience. To achieve these functionalities, the primary task was decomposed into several subtasks – 1) Human-Robot Interaction (HRI), 2) Navigation, 3) Image Processing and 4) Grasping.

1. INTRODUCTION

Recent advancements in artificial intelligence (AI) have introduced state-of-the-art (SOTA) models that achieve remarkable accuracy across various tasks. Vision Transformers (ViTs) have emerged as a groundbreaking innovation in image processing, excelling in both predictive accuracy and spatial efficiency [1]. Similarly, diffusion models have transformed generative AI by enabling the synthesis of high-quality data, significantly advancing capabilities in generative modelling.

In object detection, the YOLO (You Only Look Once) framework continues to deliver exceptional performance, with ongoing research enhancing its efficiency and accuracy [2]. Meanwhile, large language models (LLMs) such as ChatGPT represent a paradigm shift in natural language processing, showcasing the immense potential of fine-tuned pre-trained model [3]. Additionally, the fine-tuning of models like BERT (Bidirectional Encoder Representations from Transformers) has streamlined the development of domain-specific solutions by enabling the use of tailored training datasets, fostering specialized applications across various fields. These advancements highlight the versatility and transformative potential of AI across diverse disciplines.

Reinforcement Learning (RL) has emerged as a pivotal approach in robotics, enabling agents to learn complex behaviors through interaction with their environment [4]. RL frameworks allow robots to autonomously develop optimal policies for decision-making and control by maximizing cumulative rewards [4]. Advanced techniques, like Deep Reinforcement Learning (DRL), combine RL with deep neural networks, enabling robots to handle high-dimensional state and action spaces. DRL algorithms such as Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) have shown great promise in robotics tasks, including grasping, manipulation and navigation [2, 3, 4].

In this work, we present the development of an AI-powered robot designed to assist customers during their shopping experience. The primary motivation behind such a robot is the higher operational costs associated with physical retail stores compared to e-commerce platforms. Over the past few years, the popularity of e-commerce platforms has surged, particularly during the COVID-19 pandemic, with global sales reaching an estimated $4.28 trillion [5]. In contrast, physical retail stores face higher operational expenses, including rent, utilities, and in-store staffing costs [5]. A shopping assistant robot addresses these challenges by reducing the reliance on in-store staff for day-to-day operations, lowering operational costs. By integrating advanced AI techniques, the robot demonstrates robust performance and the ability to interact seamlessly with its environment, making it a practical and innovative solution for modern retail challenges.

Our robot is designed to perform two primary tasks: item retrieval and customer guidance. Item retrieval involves fetching an item requested by the customer. This process includes:

- Processing auditory instructions provided by the customer to identify the requested item.
- Navigating to the item's location while avoiding collisions with the environment and dynamic obstacles.
- Detecting the item among other items and determining its characteristics, such as size, position, and orientation.
- Grasping the item securely to ensure it is neither dropped nor damaged.
- Returning to the customer and handing over the retrieved item.

Customer guidance follows a similar sequence of steps; however, instead of fetching the item, the robot escorts the customer to the item's location. The following section provides a detailed explanation of the processes involved in the robot's development, beginning with its physical design in Section 2. Subsequently, the focus will shift to Human-Robot Interaction (HRI), highlighting the use of Large Language Models (LLMs) for interpreting and responding to customer queries Section 3. Next, image processing techniques, including facial recognition and image segmentation, will be discussed in Section 4 and 5 respectively. Section 6 will then delve into navigation algorithms, detailing the methodologies for efficient and safe movement within the environment. Section 7 details an overview of the algorithms implemented for robust grasping.

2. PHYSICAL DESIGN

To fulfil the functional requirements of the robot, several critical physical specifications must be achieved:

2.1 Mobility

The robot must be capable of navigating through the environment, which includes manoeuvring around static obstacles such as shelves and displays and avoiding dynamic obstacles like customers and trolleys. This requires a robust locomotion system, such as wheels or tracks, capable of smooth operation on flat surfaces commonly found in retail environments. The mobility system must also support precise movements to align with target objects accurately.

2.2 Reach

The robot must have sufficient vertical reach to access items stored at various heights on shelves. This may involve an extendable or articulated arm capable of positioning the end-effector at the desired height with high precision. The reach should account for the maximum height of shelves typically found in supermarkets or convenience stores while maintaining stability during operation.

2.3 Grasping:

A reliable grasping mechanism is essential for the robot to pick up securely and transport items of varying shapes, sizes, and textures. The gripper or end-effector must be designed to handle delicate objects without causing damage and be strong enough to lift heavier items. This mechanism should also adapt to different item geometries, such as cylindrical bottles, rectangular boxes, or irregularly shaped packages, ensuring versatility and functionality in grasping tasks. Various commercially available robot platforms were reviewed, and considering the functional requirements, three options were shortlisted: Apollo by Apptronik, Everyday Robot by Google, and Reflex Robot by Reflex Robotics.

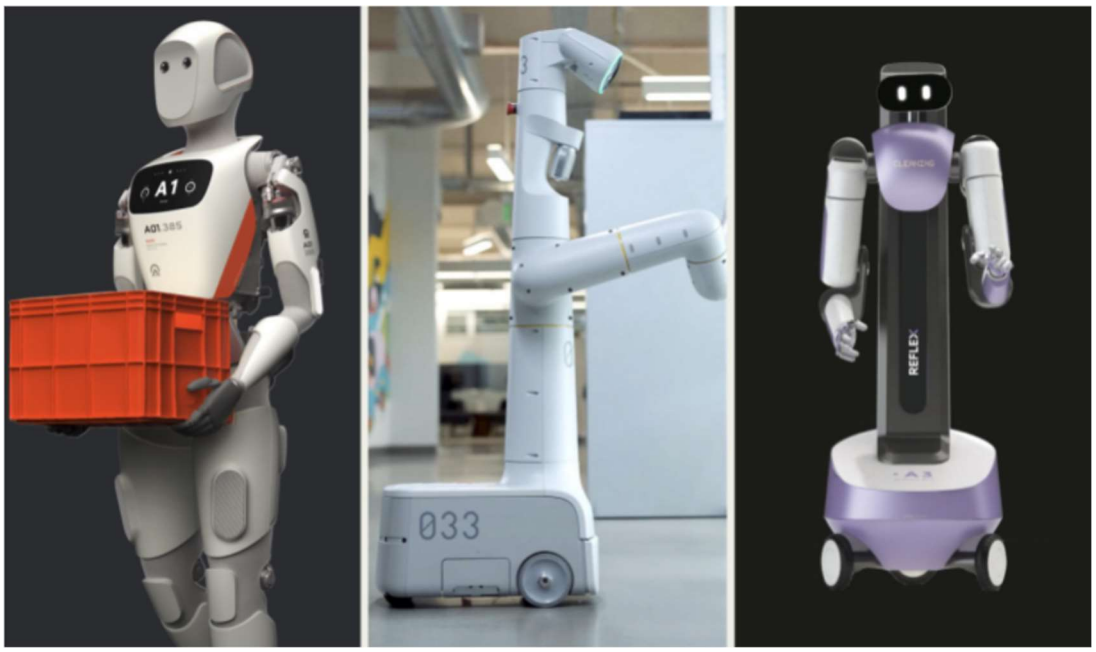


Figure 1 Options for robotic platforms. From left to right, Apollo [6], Everyday robot [7] and Reflex robot [8]

Apollo is a humanoid robot, and its design makes it potentially more approachable and comfortable for humans to interact with. However, its bipedal design results in clunky and slow movement, which may not be ideal for the fast-paced environment of retail settings. Everyday robot is smaller and more compact, making it suitable for navigating tight store spaces. However, it has limited vertical reach and grasping ability due to being equipped with only a single manipulator arm. These limitations may hinder its effectiveness in handling diverse tasks, especially those requiring interaction with shelves at varying heights. The Reflex robot offers excellent vertical reach and is equipped with dual manipulator arms, significantly enhancing its grasping and handling capabilities. However, its larger size than the other options may make it unsuitable for smaller shops with limited space.

3. HUMAN-ROBOT INTERACTION

The speech instruction from the user must be converted into a form that can be used by the large language model further down the pipeline. We utilized the AssemblyAI API to achieve voice-to-text conversion, leveraging its high accuracy and user-friendly design.

3.1 Voice Processing

AssemblyAI was chosen for its robust performance in transcription tasks and its ease of integration into diverse applications. The primary objective of this implementation is to enable seamless and accurate transcription of customer voice inputs. Additionally, this process has provided valuable insights into integrating the AssemblyAI API into our project workflow, ensuring efficient handling and processing of auditory data for downstream tasks.

The dataset used for testing was recorded by our team members and included a variety of speech scenarios to rigorously evaluate the robot's voice-to-text functionality. In addition to regular speech, noisy factors such as heavy accents, background music, and environmental noise were deliberately introduced to test the robustness of the system under challenging conditions. To address the needs of diverse target groups, we also incorporated Chinese speech data into the testing set. This approach ensures that the robot is capable of handling complex communication scenarios and effectively understanding customer needs, regardless of linguistic or environmental challenges.

AssemblyAI offers several key advantages that make it an ideal choice for our shopping assistant robot. Its high accuracy, powered by advanced machine learning models, ensures precise transcription even in noisy environments, which is essential for our robot operating in the noisy setting of a supermarket [9]. Additionally, AssemblyAI supports transcription in multiple languages, catering to a global audience [10].

This feature allows the robot to serve diverse customer bases, including international supermarkets, ensuring that speech in various languages, such as Chinese, is accurately converted to text. When combined with Large Language Models (LLMs), this capability significantly enhances the robot's ability to understand and respond to customer needs. Furthermore, AssemblyAI offers customization options, such as punctuation, formatting, and speaker identification, enabling us to tailor the transcription process to better suit the specific needs of our robot and its users.

[10]. Lastly, the API's scalability allows it to efficiently handle bulk requests, making it suitable for both

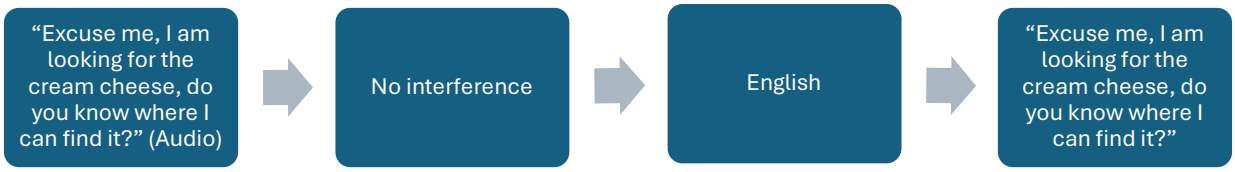
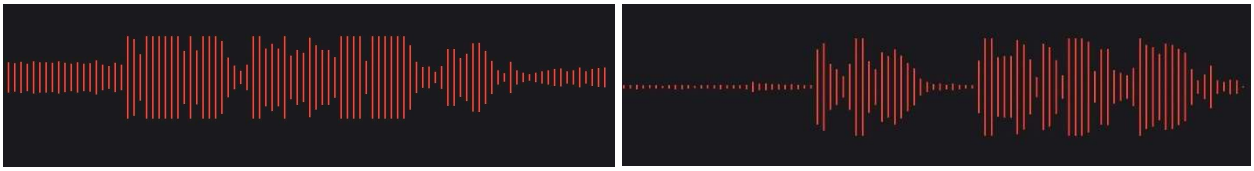


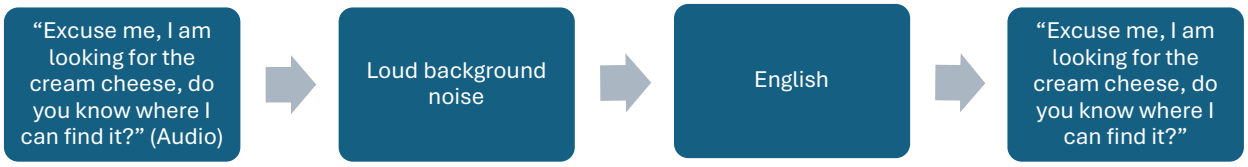
Figure 2: Sample voice conversion. The audio phrase is input to AssemblyAI which converts it to text

small-scale projects and large enterprise-level applications, ensuring reliable performance as the robot's usage grows [9].

To simulate a supermarket conversation environment, we recorded the same content but with loud background music, representing a type of noise the robot may encounter in real-world scenarios. The following spectrograms illustrate the differences between the audio files with and without background noise, providing a visual comparison of how the presence of noise affects the clarity of the speech signal. These spectrograms highlight the challenge of distinguishing speech from noise, which is crucial for evaluating the effectiveness of the robot's voice-to-text system in real-world environments.



(a) *(b)*



(c)

Figure 3: (a) Spectrogram with noise; (b) Spectrogram without noise; (c) Voice conversion with background noise

To ensure that our voice-to-text function works effectively with multiple languages, we collected Chinese voice data as part of our testing process. After converting the voice data into text using AssemblyAI, we employed the Google Translator from the "deep translator" library to translate the text into a consistent language, such as English. This step is essential for enabling convenient processing and analysis by the Large Language Model (LLM) that will be used later in the system.

Figure 4: Voice-to-text conversion with translation

Based on all the test results so far, the voice-to-text function provided by AssemblyAI meets our preset requirements.

3.2 Large Language Models (LLM)

A customer-assistance robot that can help shoppers locate items is supposed to have the ability to understand customers' words. As the voice-to-text model was already introduced before, the next step of the workflow is leveraging a large language model for understanding and analyzing the input text. This LLM should enable the robot to classify the query accurately, paving the way for effective task execution and customer guidance.

Our objective in this phase is to fine-tune a pre-trained BERT model and make it classify input sentences accurately, optimizing its performance for our use case. We used the "BertForSequenceClassification" model from the Hugging Face library as our initialized pre-trained model . To fine-tune this model, we need a labeled dataset relevant to our shopping store scenario. Since no such dataset exists, we generated 20,678 labeled sentences for training and validation and 100 sentences for testing using ChatGPT.

The input to our model is the text generated by the voice-to-text model from the previous step, while the output is a probability distribution representing the likelihood of the input text being classified under each label. For training simplicity, we only set 10 labels. Details can be found in our GitHub repository.

We conducted comparative experiments by adjusting learning rate. Taking into consideration that the scale of our training set is small, three models with different learning rates were trained on the same dataset, and

their performance was evaluated based on validation and test accuracy. The model that achieves the highest validation and test accuracy will be selected as the final choice for integration into the robot system. Since our dataset is generated by AI, we will also conduct overfit analysis based on the results.

3.2.1 Related Works

Text classification has become a cornerstone of natural language processing (NLP). In recent years, significant progress has been made in developing high-accuracy models for this task. Traditional approaches, such as the C-LSTM model—which combines convolutional neural networks (CNNs) with long short-term memory networks (LSTMs)—have demonstrated impressive performance [11]. However, the publication of the paper "Attention Is All You Need" introduced the Transformer architecture, revolutionizing NLP and becoming the foundation for state-of-the-art solutions, including text classification [12].

The rise of Transformer-based models has also fueled the development of large language models, with BERT (Bidirectional Encoder Representations from Transformers) standing out as a key player in text classification [13]. Training a BERT model for specific downstream tasks typically involves two stages: pre-training and fine-tuning.

- Pre-training: This stage involves training BERT on large volumes of unlabeled text data to help it learn general language representations.
- Fine-tuning: In this stage, the pre-trained BERT model is further trained on labelled datasets specific to the target task, allowing it to adapt to downstream applications like text classification.

Among these stages, pre-training is the most resource-intensive and costly. Fortunately, the Hugging Face community provides a wide range of pre-trained BERT models designed for various high-level tasks. Researchers can leverage these models by focusing solely on fine-tuning for their specific applications. This approach significantly reduces the computational cost and time required to build effective models, making BERT-based solutions more accessible and efficient for text classification and other NLP tasks.

3.2.2.1 Model

As previously mentioned, we selected the BERT for Sequence Classification model as our base. In this project, our task is to fine-tune the hyperparameters and further train the model to achieve higher accuracy in text classification. According to the paper BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding , the BERT model is pre-trained on a large amount of unlabeled data. We only

need to fine-tune it with an additional output layer to achieve exceptional performance on our downstream task.

3.2.2.2 Tokenizer

BERT's tokenizer uses a tokenization method called WordPiece [14]. This approach allows the model to efficiently process unknown or rare words while maintaining efficient encoding of common words. The basic idea of the WordPiece word segmentation method is to break down words into smaller meaningful units (called tokens), which can be complete words, parts of words, or even individual characters.

We choose "bert-uncased" to be the type of our tokenizer. It is particularly suitable for case-insensitive tasks, where all English letters are converted to lowercase when preprocessing text data, ignoring the case distinction between letters. Research shows that Bert-uncase tokenizer performs better than Bert-cased when doing text classification [15]. In our dataset, preserving case information may not result in additional performance gains, but rather interfere with the efficiency of training.

3.2.2.3 Configurations

We chose AdamW as its optimization algorithm [16]. It provides an effective way for AdamW to implement adaptive learning rate adjustment and a more reasonable weight decay mechanism at the same time. AdamW is a variant of the Adam optimization algorithm, which modifies the weight decay strategy in Adam to solve some problems of the original Adam algorithm when applying weight decay.

Cross-Entropy Loss is chosen as the loss function for our model. Our task is to classify the input sentence and generate a probability distribution across 10 labels, making this a multi-label classification task. Therefore, PyTorch's Cross-Entropy Loss function, combined with softmax as the output activation, is the most suitable choice for our model.

3.2.3.1 Experiments

Our scenario is that customers are asking for items occurring in the store. We searched online for existing tasks or datasets, but it turns out to be nothing. Therefore, without any other choices, we generate 20,678 sentences with ChatGPT [17], and 100 sentences for a simple test. The dataset contains input texts and their related labels. The categories and their encodings are displayed in Figure 5, where each text has only one category. For example, if a customer asks for an apple, the output probability distribution for the sentence will be t(x) = (1,0,0,0,0,0,0,0,0,0).

```
apple -> Encoded: 0
calculator -> Encoded: 1
candle -> Encoded: 2
coffee capsules -> Encoded: 3
cream cheese -> Encoded: 4
light bulb -> Encoded: 5
milk -> Encoded: 6
pumpkin -> Encoded: 7
slippers -> Encoded: 8
toothpaste -> Encoded: 9
```

Figure 5: Categories and their encodings

Since our dataset is generated by AI, there is no noise or outliers in our dataset. Therefore, the dataset can be directly fed to the model. We split the training dataset into training and validation sets with the ratio of 8:2. The number of instances for our dataset is listed into Table 1.

Dataset	Instances
Train	16542
Validation	4136
Test	100

Table 1: BERT Dataset details

3.2.3.2 Model Comparison

Our experiment includes fine-tuning the Bert model by adjusting the learning rate. We trained three models for comparison (as shown in Table 2).

Model one: This model was trained with a learning rate of 1e-5, the highest among our experiments. We intended to use a relatively large learning rate to identify the epoch at which training begins to converge.

Model two: This model was trained with a learning rate of 1e-7. After experimenting with the first model, we designed a training process with a lower learning rate to better understand how the learning rate affects test accuracy.

Model three: Based on observations from the previous two models, we set the learning rate for the third model to 2e-7 to explore how an intermediate learning rate impacts the model's performance. The following subsection provides detailed results and analysis.

Model #	Learning rate
1	1e-5
2	1e-7
3	2e-7

Table 2: Fine-tuned BERT models learning rates

3.2.4 Results and Analysis

We trained our model with 10 epochs. Our results are from three aspects: number of epochs training converge, validation and training loss, accuracy over training, validation and test dataset.

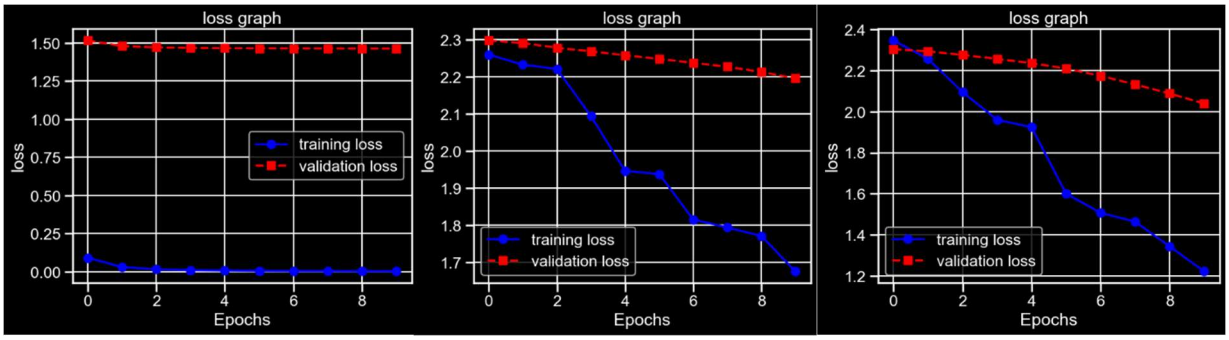
3.2.4.1 Convergence Epoch

Model #	Epoch #
1	1
2	Not converge after 10 epochs
3	8

Table 3: Number of training epochs to converge

Table 3 illustrates epochs for each model when training accuracy starts to converge.

Model One has the largest learning rate, resulting in the optimizer taking the largest steps at each epoch. The results show a remarkably fast convergence, with the training accuracy and validation accuracy



(a) *(b)* *(c)*

Figure 6: Loss Curves (a) model one; (b) model 2; (c) model 3

stabilizing after just one epoch. Model Two has the lowest learning rate, which prevented the model from converging within 10 epochs. Based on the experiments with Model Two, we set an intermediate learning rate for Model Three, which converged at Epoch 8.

3.2.4.2 Validation and Training Loss

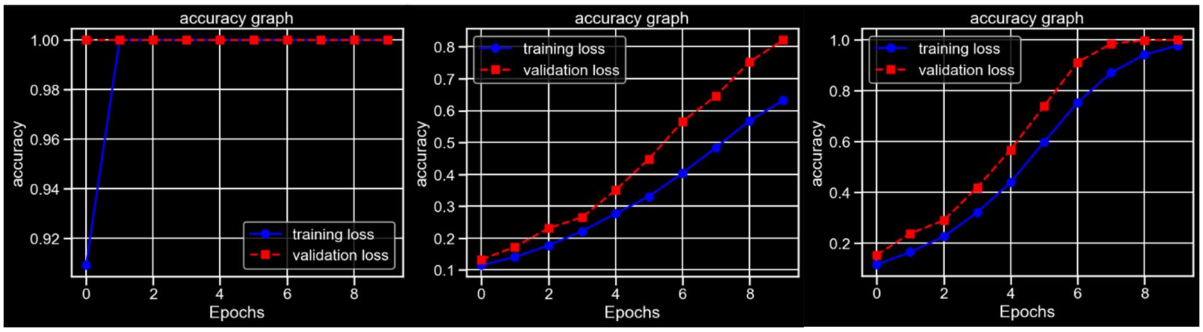
The loss curves for each model across epochs are shown below.

- Figure 6(a) shows the loss vs. epochs plot with a learning rate of 1e-5. Both training and validation loss converged within the first epoch and remained flat for the subsequent epochs.
- Figure 6(b) illustrates the loss vs. epochs plot with a learning rate of 1e-7. The training loss decreased during the first and second epochs and then dropped more rapidly after the third epoch. The validation loss decreased at a consistent rate.
- Figure 6(c) presents the loss vs. epochs plot with a learning rate of 2e-7. Despite a slight fluctuation in the rate of decrease at Epoch 4, the training loss decreased at a relatively steady pace. The validation loss, similar to Figure 5(b), decreased at a constant rate.

3.2.4.3 Training, Validation and Test Accuracy

The accuracy curves for each model across epochs are displayed below. Figure Four represents Model One, Figure Five represents Model Two, and Figure Six represents Model Three. Meanwhile, test accuracy for three models is shown in Table 4.

- Figure 7(a) shows that the training accuracy rapidly increased to 100% after Epoch 1, while the validation accuracy remained at 1.0 throughout all epochs.
- In Figure 7(b), neither the training accuracy nor the validation accuracy converged after 10 epochs of training.
- Figure 7(c) depicts the training accuracy curve converging at the end of the training process, whereas the validation accuracy converged at Epoch 8.



(a) *(b)* *(c)*

Figure 7: Accuracy Graph (a) model 1; (b) model two; (c) model 3

Model #	Test accuracy
1	1.0
2	0.99
3	0.86

Table 4: Test Accuracy after BERT fine-tuning

3.2.4.4 Analysis

Model One trained rapidly, achieving an accuracy of 1.0 after just one epoch. It outperformed the other models across all evaluation metrics. However, considering that our dataset was generated by AI, with similar sentence patterns, Model One might be overfitted to our specific dataset.

Model Two was underfitted even after 10 epochs of training, indicating that a learning rate of 1e-7 is not ideal. To address this, we trained the third model with an intermediate learning rate of 2e-7. The accuracy of Model Three converged at Epoch 8, reaching a value of 0.99. Its performance on the test dataset was also satisfactory. Compared to Model One, Model Three is less likely to be overfitted, making it our final choice.

In conclusion, the final choice of the fine-tuned model is model three, where the learning rate at the fine-tuning phase is 2e-7. The model trained with this learning rate performs well on the validation and test dataset. Meanwhile, the slower convergence speed helps it prevent overfitting. Although the performance of models trained with learning rate 1e-5 stays excellent, it still has a higher risk of overfitting, considering the lack of training data variety.

4 FACE RECOGNITION

The face recognition system is designed to identify a customer's face to ensure that services are provided to the correct customer. The system[1] requires one reference photo of the customer as a benchmark. The benchmark is used for comparison with the input camera frame. The reference photo can either be saved in the database or taken at the store using the robot camera. The system works by detecting all the faces in the camera frame, extracting their features, and comparing these features to the benchmark photo to determine a match. If a detected face is found to be sufficiently similar to the benchmark, the system identifies the detected face as the same person in the reference photo.

The face recognition system is implemented using the "face_recognition" library [18]. The library simplifies complex face recognition tasks by providing a Python interface for detecting, encoding, and comparing faces. This library relies on "Dlib" [19], a high-performance C++ library with powerful machine-learning capabilities. The pre-trained model in the face recognition library operates by extracting face embeddings from an image and comparing these embeddings with embeddings of the benchmark to identify the correct person. Face embeddings are numerical representations of facial features.

4.2 Model Architecture

The face recognition system is divided into two subsystems: Face Detection and Face Comparison shown in Figure 8.

The Face Detection Subsystem acts as the initial stage in the pipeline. It identifies all faces present in the input images and processes the input image to ensure they are consistent for the Face Comparison sub-

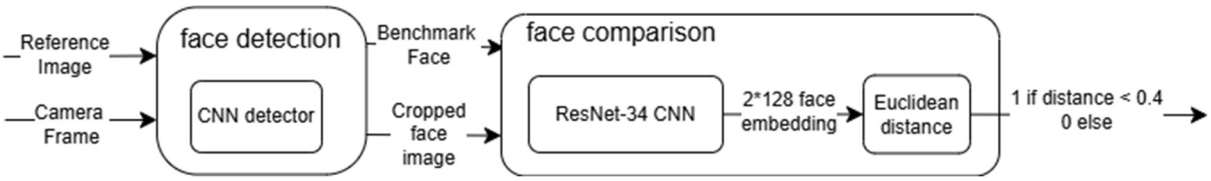


Figure 8: Architecture of the face_recognition model

system. Both the live camera frame and the reference image are fed into this subsystem because it is necessary to remove the background from the reference image and ensure the benchmark face image also fits the Face Comparison sub-system. The system uses a CNN-based face detector provided by Dlib to locate bounding boxes around each detected face. Once a face is detected, the system identifies 68 facial landmarks, such as the eyes, nose, and mouth, to perform alignment. This step ensures that the detected faces are properly oriented, centred, and resized to a standardized dimension. The above process is also applied to the reference image to produce a single cropped and aligned face that serves as the benchmark for the later comparison. The output of the Face Detection Subsystem includes multiple cropped and aligned facial images from the camera frame, as well as one cropped reference face from the reference image.

Once the faces have been detected, aligned, and prepared, they are passed to the Face Comparison Subsystem. The face Comparison Subsystem handles feature extraction and identity verification. In this subsystem, each face image is processed through a deep learning model based on the ResNet-34 architecture. This model generates a 128-dimensional face embedding, which is a compact numerical representation that uniquely captures the distinguishing features of a person's face. The system also generates face embedding for the reference face with the same processing. The system then compares the embeddings of all detected faces with the embedding of the reference face using the Euclidean distance metric one by one. If the distance between the two embeddings is below a predefined threshold which is 0.4, the system determines that the two faces belong to the same person. If no face comparison is below the 0.4 threshold, then none of the people in the camera frame is the right customer. The robot will wait till the right customer appears.

The overall system architecture is designed to take live camera frames and a single reference photo as inputs, ensuring both are preprocessed consistently through the Face Detection Subsystem. The outputs of

the detection subsystem, including the cropped and aligned faces, are sent to the Face Comparison Subsystem, which produces the embeddings and calculates similarity scores. The final output of the system is a binary decision that indicates whether a detected face matches the reference face.

4.3 Sample Results

Figure 9: Face recognition output under normal conditions. Left: reference image; Right: camera frame with red bounding box and label

In sample results, the system uses a red rectangle to show the face detection and shows the name of the face for recognition results. Figure 9 shows the benchmark and the camera frame with associated output.

Figure 10 shows how the system works with certain special conditions including glasses removed and face covered.

4.4 Limitations

The results demonstrate that the system performs well under certain conditions, such as when the face is unobstructed or only minimally covered. However, the system cannot work properly when essential facial features are covered.

Overall, there are some constraints regarding the reference image and face coverings that must be considered for the system to function effectively. For the reference image, it must only contain one face to ensure that the system does not detect any other face and mistakenly treat it as the benchmark. So, if the

(a) *(b)*

(c) *(d)*

Figure 10: (a) Model output without glasses; (b) Output when small part of the face is covered; (c) Failed detection when one eye is covered; (d) The model can detect that the image contains a face but is unable to identify the face accurately.

customer wants to take the photo as a reference image on the first visit, it is recommended to do so in an isolated room to eliminate potential disruptions caused by other people appearing in the reference image.

Another important constraint is that there should be no covering of key facial features, such as the mouth, nose, or eyes. If any of these essential features are covered, the system may fail during either the detection or recognition stages.

Although these limitations make the system imperfect, it is still sufficient to function as a subsystem for the entire robot operating in a supermarket or grocery store.

5 IMAGE SEGMENTATION

Image segmentation is a process used to divide a visual input, such as image, into multiple segments. This technique identifies and groups pixels that share similar characteristics, like color, texture, or intensity, to isolate meaningful parts of the image.

In our vision, the robot should be able to pick up items from shelves and hand them to customers to help people who have difficulty picking up items, such as people with disabilities or children. In addition, if items are placed on higher shelves, the robot must be able to help users pick up the items they need.

To achieve this, the robot must accurately pick up items from the shelves. This requires the ability to distinguish between "shelves" and "goods" and to understand the shape, size, and posture of the items to facilitate precise grasping actions.

The use of image segmentation techniques is important in this process. These techniques enable the robot to differentiate objects from their background, separate overlapping items, allowing for accurate identification and manipulation of the goods.

5.1 U-Net++ (Nested U-Net) Architecture

U-Net is a fully convolutional neural network designed for image semantic segmentation, where each pixel in an image is assigned a label. The architecture consists of two main components: an encoder and a decoder, connected by skip connections. The encoder extracts features at various spatial resolutions, while the decoder reconstructs the segmentation mask by combining the encoder's features with its upsampled features through concatenation (as shown in Figure 11).

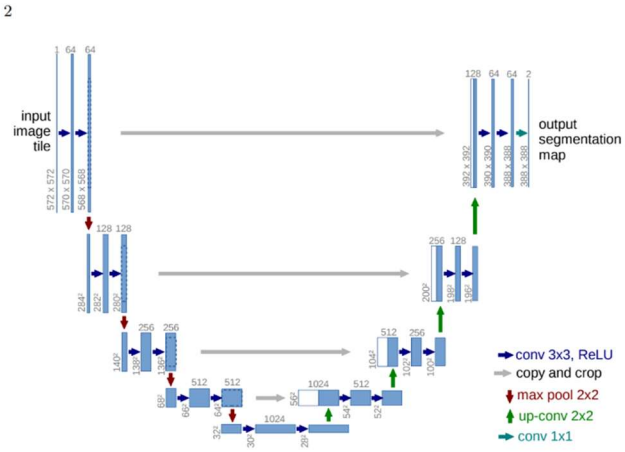


Figure 11: U-Net Architecture

U-Net++ builds upon the original U-Net by introducing nested skip connections. In the traditional U-Net, a single skip connection links an encoder layer to its corresponding decoder layer. U-Net++ incorporates additional intermediate convolutions between encoder and decoder layers at various depths. It creates nested skip paths. These paths progressively refine features before they are passed to the decoder (As shown in Figure 12). Which will enhance the model's capability to handle complex segmentation tasks with precision.

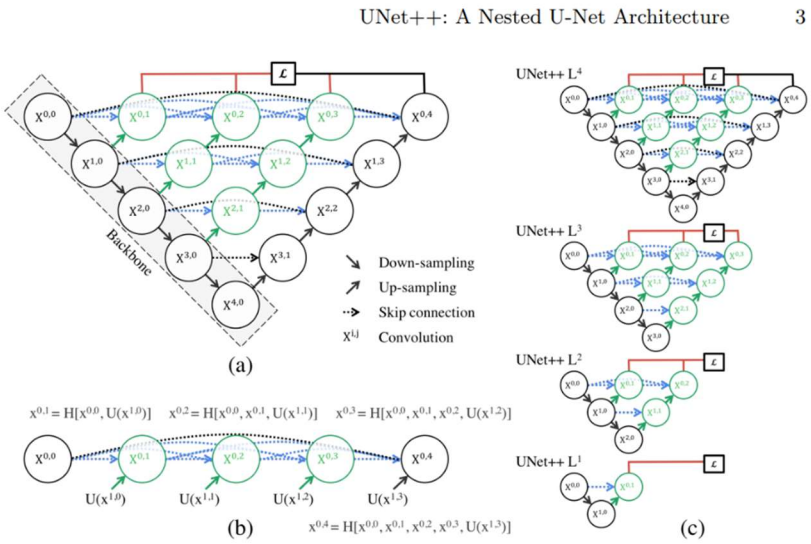


Figure 12: U-Net++ Architecture

5.1.1 Key features of U-Net++ model

5.1.1.1 Encoder

Encoder is one of the key components in U-net++, it is responsible for extracting the features from the image. It could be mainly separated into two parts: convolutional bocks, and max pooling layers. It consists of a series of convolutional blocks, where each block contains convolutional layers, batch normalization and ReLU activation function. These convolutional layers are responsible for learning the feature representations like edges, textures etc. Batch normalization and ReLU activation function are responsible for providing a stable and nonlinearity learning process. After each convolutional block, max pooling layers are applied to progressively downsample the spatial dimensions. This operation allows the encoder to focus on broader spatial relationships and reduce the computational complexity at the same time. The Encoder will generate feature maps for each of those different level spatial resolutions. Then those feature maps will be used in the reconstruction of the decoder part for accurate image segmentation.

5.1.1.2 Decoder

The decoder subnetwork in U-Net++ is responsible for upsampling the feature maps generated by the encoder and reconstructing them to match the original resolution of the input image. This process is done

by a series of upsampling blocks and convolutional layers. The upsampling block in U-net++ uses transposed convolutions to reconstruct feature maps to the original resolution. And it is followed by convolutional layers which can help to refine those feature maps. These blocks integrate the high-level features from the encoder though skip connections with the decoder's upsampled features. The skip connections ensure that the decoder incorporates both low-level and high-level features. It improves the accuracy of the segmentation map. By combining these features, the decoder is able to produce a pixel wise segmentation map of the input image.

The improvement makes U-Net++ particularly effective for tasks requiring fine-grained and accurate segmentation that can be concluded into two key features below.

5.1.1.3 Nested dense skip connections

As the figure shown above, U-net++ has a different skip connection structure compared to the traditional skip connections used in U-net. Instead of directly connecting encoder and decoder features, U-net++ introduces nested dense convolutional blocks in the skip pathways. These convolutional layers contained in the convolutional blocks progressively bridge the semantic gap between encoder and decoder feature maps. As the number of convolutional layers increases, the skip connections grow as well. Adding dense skip connections improves the gradient flow during training, increases feature reuse, and helps to better optimize the network.

5.1.1.4 Deep supervision

Deep supervision is one of the most important features in U-net++. It enables the model to work efficiently in two different modes: accurate mode and fast mode. The accurate mode will take the average of all segmentation branches' predictions. And use it to produce the final segmentation map to achieve the high accuracy. In the fast mode, the final segmentation map is selected from only one of the segmentation branches to speed up the process. Furthermore, this mechanism enables the model to generate auxiliary output at multiple semantic levels (As indicated in Figure 12, {x^0, j, j $\in$ {1, 2, 3, 4}). For each of these layers, applying a combination of binary cross-entropy and dice coefficient as the loss function. Which will help the optimization and improved segmentation accuracy across all branches.

5.1.2 Implementation

The U-net++ model-based image segmentation system is implemented refer to [20]. This model is implemented using the smp. UnetPlusPlus function from the segmentation models PyTorch (smp) library. It uses a pre-trained encoder to extract features and a decoder to create segmentation maps. It processes

RGB images and outputs a single-channel map for binary segmentation. During training, it calculates two losses: Dice Loss for overlap accuracy and Binary Cross-Entropy Loss for pixel-level accuracy. These losses are combined to guide the model's learning.

5.1.2.1 Dataset

Instead of the original dataset attached by the code, this project used the densely segmented supermarket (D2S) dataset . We selected the D2S dataset because it focuses on 60 categories of groceries and everyday products. The training images feature a single class of objects on a common background, while the validation and test sets are more diverse, with variations in lighting, rotations, and backgrounds. This makes it perfectly suitable for training the image segmentation function of a shopping assistant robot. Additionally, the size of the D2S dataset is sufficiently large, containing 21,000 high resolution images with pixel-wise labels for all object instances, ensuring the model can be trained effectively.

Due to equipment and time constraints, the training process was conducted on a small subset of the dataset, representing only 0.4% of the total data. This was done to simulate the training process efficiently. It is expected that utilizing the entire dataset would lead to improved results.

5.1.2.2 Pre-processing

First, the dataset was divided into two parts: the training set and the validation set. The proportions are 80% and 20% respectively The DS2 dataset contains very high quality data and does not require much data cleaning operation. However, in order to improve the efficiency of model training, the data augmentation operations are applied in the code.

Confusion Matrix	
True Negatives (TN): 172406	**False Positives (FP): 10108**
False Negatives (FN): 10791	**True Positives (TP): 7399**

Table 5: Confusion Matrix

- Training Dataset Augmentation:
    - Resizes all images to a fixed size defined by 224*224 for uniform input to the model.
    - Randomly flips the image horizontally with a 50% probability, simulating different orientations.
    - Randomly flips the image vertically with a 50% probability for further variation.
    - Rotates the image by 90 degrees with a 50% probability, adding rotational invariance.
- Validation Dataset Augmentation:

- Only resizes validation images to 224*224 without applying random transformations to ensure consistency in evaluation.

The accuracy of the model is high (0.90), indicating that it correctly predicts outcomes most of the time. However, precision and recall are relatively low (0.42 and 0.41), suggesting that the model faces challenges in predicting positive classes effectively. To improve the prediction of positive classes, the model would require more extensive training, ideally with a larger portion of the DS2 training set, as opposed to the small subset used in the current setup. Despite these challenges, the current performance demonstrates the model's significant potential for development, as it achieves an accuracy of 0.90 with only 0.4% of the training data.

Accuracy	0.90
Precision	0.42
Recall	0.41

Table 6: Training metrics

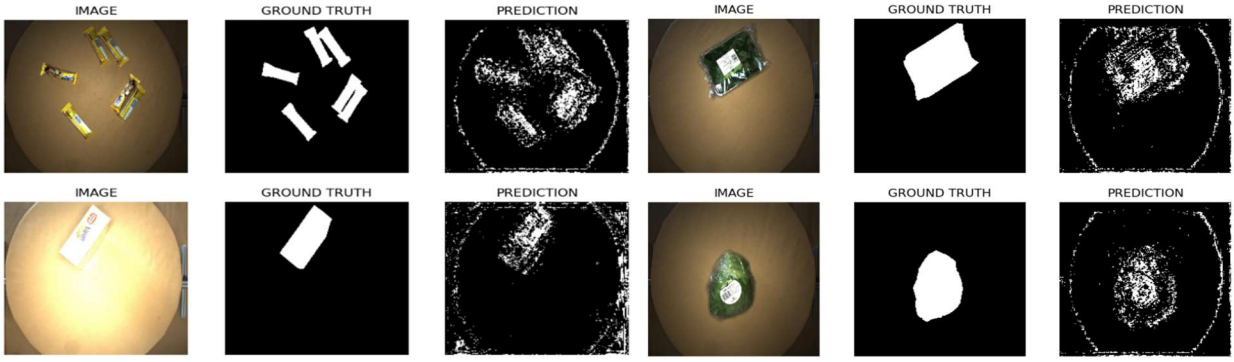


Figure 13: U-Net++ sample result (1)

Figure 13: U-Net++ sample result (2)

5.1.3 Sample Results

In addition to recognizing the shapes and poses of products, U-net++ can also be used to recognize the poses of customers and “hand the product to the customer”. However, this requires the use of a training set of portraits. Due to time constraints, this feature is under development.

As can be seen from the results, the U-net++ is unable to accurately differentiate and segment the items when they are stacked (e.g., the first row of Figure. 13). Besides the reason that the model is not fully trained, this is also related to the characteristics of it. U-net++ is good at semantic segmentation (pixel-wise classification without distinguishing between object instances). While we envision that things in the supermarket should be neatly arranged rather than stacked, there are always accidents. Therefore, we decided to add the Mask-RCNN model to the image segmentation function to improve the Instance segmentation (detection and segmentation of individual objects).

5.2 Mask R-CNN

Mask R-CNN is a framework designed to address the task of instance segmentation which involves identifying, classifying, and precisely segmenting each object in an image. Unlike traditional object detection methods which only provide bounding boxes around objects, Mask R-CNN provides pixel-level segmentation for each detected object instance.

Instance segmentation is critical in applications where accurate object differentiation and precise boundaries are essential. For example, in an image containing multiple objects of the same class, such as several oranges, Mask R-CNN can distinguish and assign a unique segmentation mask to each instance. This can help the robot identify the boundary of items on the shelf. And that will help for the grasping task.

Mask R-CNN is improved from Faster R-CNN. It enhances object detection capabilities by adding a parallel branch to predict pixel-level masks for each object instance. It achieves high accuracy in both detecting objects and producing refined segmentation masks, even in challenging scenarios with occlusions, varying scales, and complex object shapes.

5.2.1 Architecture

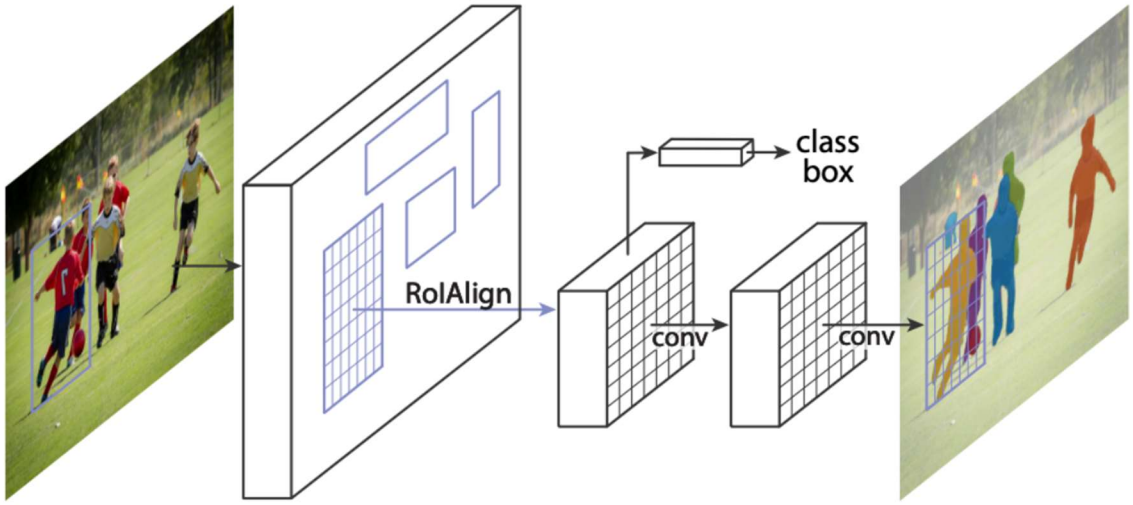


Figure 14: Mask R-CNN Architecture

Figure 15 shows the model architecture of Mask R-CNN. The architecture consists of 3 parts, including the Region Proposal Network (RPN), the first convolution network and the second convolution network

The Region Proposal Network (RPN) is applied to the feature map to generate a set of region proposals, which are areas in the image likely to contain objects. These proposals are represented as rectangular bounding boxes. RoIAlign (Region of Interest Alignment) precisely aligns these proposed regions with the feature map. RoIAlign ensures that spatial information is preserved, which is critical for tasks requiring high localization accuracy, such as segmentation. Overall the RPN part aims to detect any object in the image that needs to be processed later. It eliminates the effects of the background.

The first convolutional network processes each aligned region to generate a class label and refine its bounding box coordinates. This forms the object detection output, identifying the objects and their locations in the image.

The second convolutional network also processes aligned regions from the RPN part, but it aims to generate a Pixel-level mask for that specific object. It outlines the exact shape of the object within its bounding box. Because the bounding box only contains one object, the entire Mask R-CNN system can generate unique masks for individual objects, even if they belong to the same class.

The final output of Mask R-CNN combines both the detection from the first convolutional network and segmentation results from the second convolutional network. Objects in the image are represented with bounding boxes, class labels, and pixel-perfect segmentation masks.

5.2.2 Implementation

The Mask R-CNN model architecture used here is Mask-RCNN-TF2 [21] which is adapted to work with TensorFlow 2. The model is trained with MVTec Densely Segmented Supermarket Dataset [22] which is a comprehensive benchmark designed for instance-aware semantic segmentation within industrial contexts.

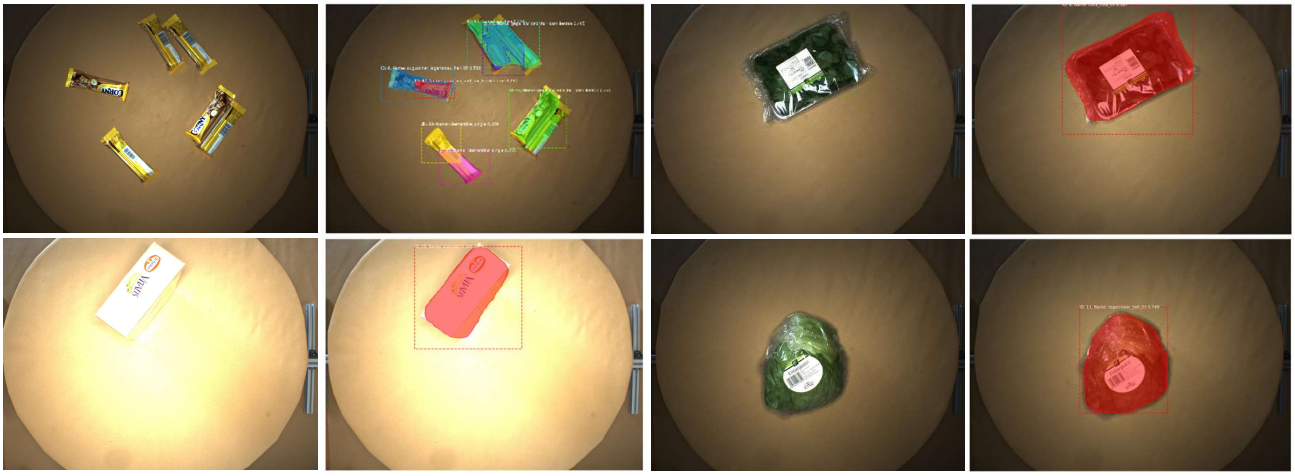


Figure 15: Input and Corresponding output of Mask R-CNN

The dataset comprises 21,000 1920*1440 images, each annotated with pixel-level labels for all object instances, encompassing groceries and everyday products across 60 distinct categories.

5.2.3 Results

Figure 16 shows that the current model trained works well for a single item, it can create a precise pixel-level mask for that single item. However, it messes up for multiple items. There are duplicate masks and bounding boxes that only have half of the object. The masks show that the model can still find the object's location, but it is easy to treat two closed objects as one.

Based on the result, the model needs to be trained more, but the training time is too long due to the large dataset used. There is no time to get the model to train more. Compared with U-Net, which is another choice for image segmentation tasks, Mask R-CNN can create a more uniform mask for the single item. There isn’t any noise inside the mask and the boundary of the mask is clear. Also, the number of parameters used for Mask R-CNN is much larger than the number of parameters used for U-Net. That may be another reason why Mask R-CNN perform better than the U-net. For duplicate objects with the same class, Mask R-CNN can create separate masks for each individual object, although it didn’t show in the result due to lack of training.

6. NAVIGATION

Navigation involves guiding a mobile robot from one location to another, utilising data from various sensors. A robust navigation system is essential to ensure the robot can traverse its environment safely and efficiently. Recent advancements in robot navigation have predominantly emphasised AI-based solutions [23]. However, the selection of an appropriate algorithm is highly influenced by the specific use case, the robot's characteristics, and the properties of the environment.

Our robot is specifically designed to operate in retail environments, such as supermarkets and convenience stores, which typically feature flat terrain and multiple shelves for item display. The layout of these stores is generally static, with alterations occurring only rarely. However, a significant challenge is the presence of numerous dynamic obstacles, particularly humans and human-controlled objects such as trolleys. Ensuring the robot avoids collisions with humans is critical to prevent potential injuries and ensure safety.

Given the environment's nature, a classical, non-AI-based approach to navigation has been adopted. The implemented navigation system comprises four key components: mapping, localization, global planning, and local planning. The following subsections provide a detailed description of each component.

6.1 Mapping

Mapping in robotics involves creating a 2D or 3D representation of the spatial layout of the operating environment. While robots are capable of exploring unknown environments, utilizing a pre-existing map offers several advantages for autonomous navigation. It enables the robot to achieve higher accuracy during localization and facilitates the generation of more efficient global paths.

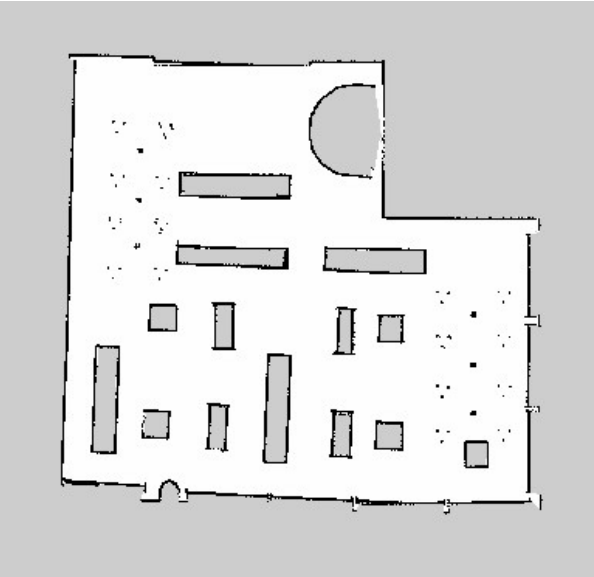


Figure 16: Map of aws-bookstore-world created using gmapping

Mapping is performed using a technique known as Simultaneous Localization and Mapping (SLAM). Various SLAM algorithms exist [24]; our implementation employs a Rao-Blackwellized particle filter-based approach called GMapping [25, 26].

In this method, a finite number of particles are sampled from a proposal distribution, with each particle representing a potential map estimate. Each particle is assigned a weight, computed using the principle of importance sampling. As the robot navigates the environment, the pose of each particle is updated based on its previous pose and odometry data. The particle weights are then refined using observations from the robot's updated pose. Finally, the particles are resampled based on their updated weights. During resampling, particles with low weights (unlikely estimates) are eliminated, while new samples are generated around particles with higher weights (likely estimates) to improve the accuracy of the map estimate.

6.2 Localization

Localization is a critical component of autonomous navigation. It involves estimating a robot's pose—its position and orientation—relative to its environment. The localization problem can be categorized into three main scenarios: position tracking, global localization, and the kidnapped robot problem [27].

In the position tracking scenario, the robot's initial position is known, and the task is to continuously estimate its position as it moves through the environment, accounting for odometry noise. Conversely, global localization deals with situations where the robot's initial state is unknown and must be determined

from scratch. Finally, the kidnapped robot problem arises when a well-localized robot is suddenly displaced to an unknown location, requiring it to re-estimate its current position.

Effective localization should be capable of addressing all three tasks: position tracking, global localization, and the kidnapped robot problem. Our robot utilizes the Adaptive Monte Carlo Localization (AMCL) algorithm, which is also a particle filter-based approach inspired by the Bayes filter.

In Monte Carlo Localization (MCL), a set of particles is sampled from a proposal distribution, with each particle assigned an importance weight. Unlike GMapping, MCL does not associate each particle with a map estimate since the map is provided beforehand. As the robot navigates the environment, particles are updated using the robot's dynamic model, and their weights are adjusted by comparing ray-casted observations from each particle to actual sensor observations.

The "adaptive" aspect of AMCL refers to the enhanced sampling technique employed. While basic MCL relies on Sequential Importance Sampling with Resampling (SISR), AMCL uses the Kullback–Leibler distance to determine the number of particles needed for accurate localization dynamically [28].

6.3 Global Planning

Path planning is the process of determining an optimal and feasible path from an initial pose to a specified final pose while accounting for various constraints and conditions. Global path planning assumes complete knowledge of the environment and is most effective in relatively static settings. Widely used global planning algorithms include A*, Dijkstra's algorithm, and Rapidly-Exploring Random Trees (RRT). However, since global planners typically rely on a pre-existing map of the environment, they do not account for dynamic or unknown obstacles during the planning process.

*6.3.1 A**

A* is a grid-based planning algorithm, meaning that the environment's map is divided into a grid of equally spaced squares, with each square representing a node or state within the environment. The algorithm requires three inputs: the map of the environment, the initial pose, and the final pose.

The algorithm begins by evaluating the grid cell containing the start pose and checks its neighboring grid cells to determine whether they are collision-free. Collision-free cells are added to an open list. This process is repeated for each grid cell in the open list, prioritized by the lowest cost until the goal grid is reached. In A*, each grid cell is associated with three cost values:

- g-cost: The cost of reaching the grid cell from the initial position.
- h-cost: The heuristic cost is an estimate of the cost required to reach the goal pose from a specific grid cell, commonly calculated using methods such as Manhattan or Euclidean distance.
- f-cost: The total cost, calculated as the sum of g-cost and h-cost.

The algorithm uses these cost values to guide the search toward the most efficient path to the goal. Dijkstra's algorithm can be considered a version of A* where the heuristic cost is always zeros.

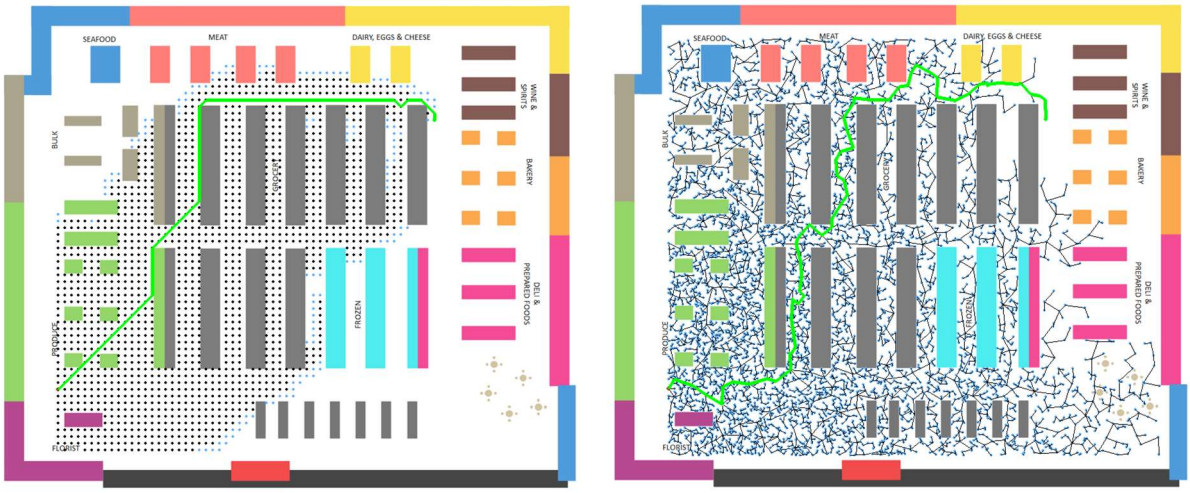


Figure 17: Visualization of path planning algorithms. On the left, the path generated by A. On the right, path generated by RRT*

6.3.2 Rapidly-Exploring Random Tree (RRT)

RRT is a sampling-based planning algorithm that incrementally builds a tree within the free space of the environment. The algorithm samples nodes randomly and connects each new sampled node to the nearest node in the existing tree, provided the connecting path is collision-free. The newly connected node is then added to the tree. Due to its sampling-based nature, the path generated by RRT is typically sub-optimal. Variants such as RRT* address this limitation by incorporating optimization techniques. Unlike RRT, which does not allow for node reconnection, RRT* permits reconnection and includes a cost to each node, making it asymptotically optimal.

6.4 Local Planning

Since the global planner operates on a static map of the environment, a mechanism is required to enable the robot to adapt to unforeseen changes and dynamic obstacles. This necessity serves as the motivation for

local planning. In our implementation, we utilize the Dynamic Window Approach (DWA) provided by ROS.

In the DWA algorithm, the robot's control space is discretely sampled across a range of linear and angular velocities. The sampled space is constrained to include only velocities that the robot can achieve in a single time step, considering its acceleration limits. This constraint reduces the dimensionality of the search space, thereby enhancing computational efficiency.

A forward simulation is then performed for the resultant trajectory of each sampled velocity. Each trajectory is scored based on proximity to obstacles, proximity to the goal, alignment with the global path, and speed. Trajectories that result in collisions are discarded. The trajectory with the highest score is selected for execution, and the process is repeated continuously to ensure dynamic adaptability.

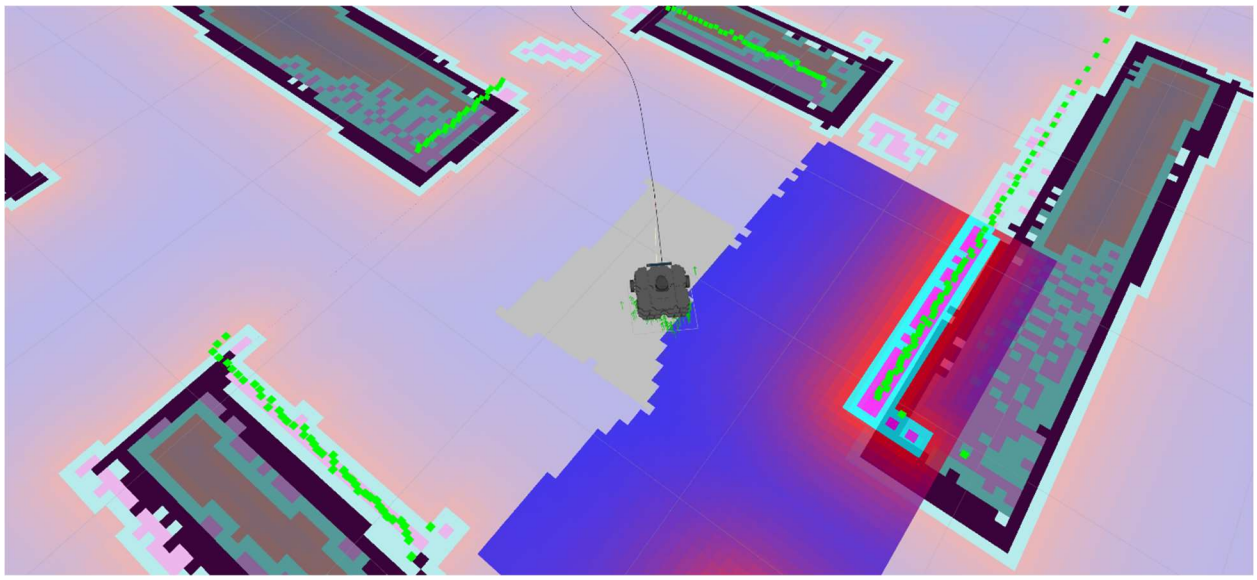


Figure 18: Simulation of autonomous navigation using ROS and Turtlebot. The green arrows represent the each particle estimate for localization. The long red line is the global path plan. The shorted yellow line is the local planner output.

7. GRASPING

One of the primary functions of our robot is to fetch objects for customers. Supermarket items exhibit significant variation in shape, size, and texture, requiring the implementation of a robust grasping algorithm to pick items from shelves effectively. To address this challenge, a learning-based approach, specifically a Deep Reinforcement Learning (DRL) algorithm, has been employed due to its superior performance in generalizing across various scenarios. Various Deep Reinforcement Learning (DRL) algorithms have been

utilized for grasping tasks. This report focuses on two specific algorithms: Double Deep Q-Learning (DDQN) and Proximal Policy Optimization (PPO).

7.1 Double Deep Q-Learning (DDQN)

Real-world tasks such as grasping are too complex to learn all state-action values individually, as done in traditional Q-learning. In Deep Q-Learning (DQN), the value function is parameterized and approximated using a deep neural network. To further stabilize the training process, the algorithm employs experience replay.

The primary motivation for Double Deep Q-Learning (DDQN) is to address the overestimation bias inherent in DQN. DQN often learns unrealistically high Q-values due to the maximization step over overestimated values, which results in suboptimal policies [29]. DDQN mitigates this bias by employing two separate neural networks: an online network and a target network. The online network is used to evaluate the greedy policy, while the target network estimates the Q-value, leading to more accurate value predictions and improved policy quality.

In [30], the authors introduce the Grasp-Q-Network, a model based on the DDQN architecture, and validate its effectiveness through both simulation and real-world experiments using the Baxter robot. The model processes image data from two sources: an overhead RGB-D camera and an RGB camera mounted on the robot's wrist.

The outputs from both cameras are passed through a series of convolutional and batch normalization layers, ending with a max-pooling layer. The max-pooled outputs from the two image streams are concatenated and processed through an additional convolutional layer, followed by a dropout layer and a fully connected layer.

This output is then combined with the output of motorNet, a network consisting of two fully connected layers separated by a dropout layer. The combined data is further processed through two more fully connected layers, again separated by a dropout layer. The final output represents the predicted Q-values for each action in the action space. These Q-values are used to derive an optimal policy for the grasping task.

The authors defined a reward function to guide the learning process of the Grasp-Q-Network. A reward of +10 is assigned for successfully grasping an object, while a reward of +1 is given when the end-effector makes contact with the object during the grasp attempt. A penalty of −1 is applied if the agent takes no

action. Additionally, a small negative reward of -0.025 is provided for each time step, encouraging the agent to complete the grasping task efficiently.

7.2 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a policy gradient algorithm used in deep reinforcement learning and is closely related to the Advantage Actor-Critic (A2C) algorithm. The primary objective of PPO is to take the largest possible improvement step on a policy using the currently available data, without overstepping to the point of causing performance collapse [31].

Similar to DDQN, PPO employs two neural networks: an actor network to approximate the policy function and a critic network to approximate the value function. During the learning process, the current state of the agent is fed into the actor network, which outputs a probability distribution over all possible actions for that state. This action probability distribution is then passed to the critic network, which computes the Advantage function. The advantage function is defined as the difference between the Q-value for a given state-action pair and the value function of that state and is used to update the policy parameters.

To ensure that step sizes during policy updates are neither too small nor excessively large, PPO uses a clipped surrogate objective function. This function clips the ratio of the new to old policy probabilities within a range of $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter. In the original PPO paper [32], the authors also propose an alternative approach using a KL divergence penalty to constrain updates. However, this approach is less commonly used due to its increased implementation and computational complexity.

In [33] the authors employ the PPO algorithm for learning robotic grasping tasks. The policy network is a three-layer Multi-Layer Perceptron (MLP) with two hidden layers, each of size 64, and uses Rectified Linear Unit (ReLU) activation functions. The value network is a two-layer MLP with a hidden layer size of 128, also using ReLU activation.

The reward function for grasping is divided into two phases: reaching and lifting.

- Reaching Phase: The reward is calculated as the weighted sum of the relative position of the gripper site, the velocity vector of the end-effector, and the gripper's open action.

- Lifting Phase: Initiated when the gripper is in close proximity to the object, this phase computes the reward based on similar parameters as the reaching phase (position and velocity), as well as additional factors such as the gripper's close action, contact with the object, and successful task completion.

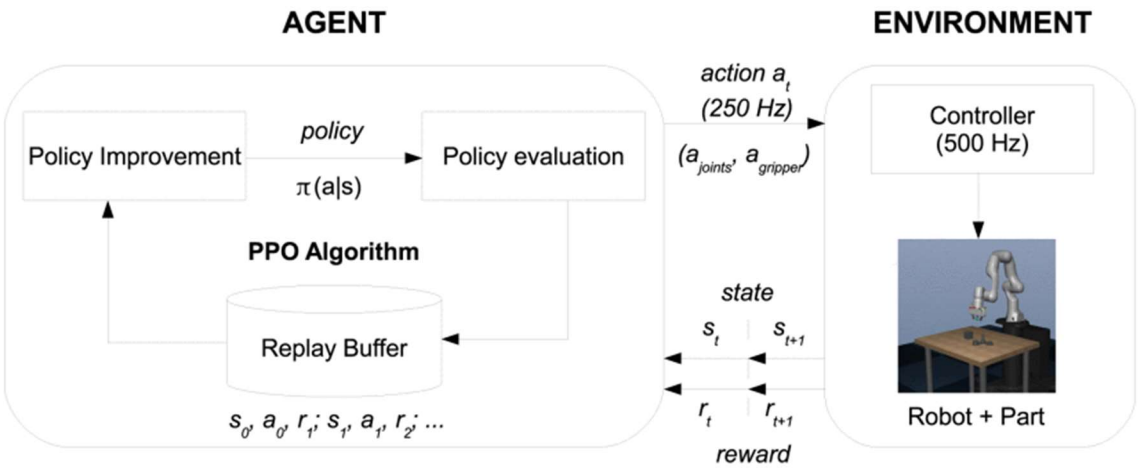


Figure 19: Grasping training pipeline using PPO

We implemented a grasping simulation using a Panda robot in Nvidia's Isaac Sim. The simulation employed a reward function like the one described above. The model was trained over 300 epochs and achieved a maximum reward of 134.79.

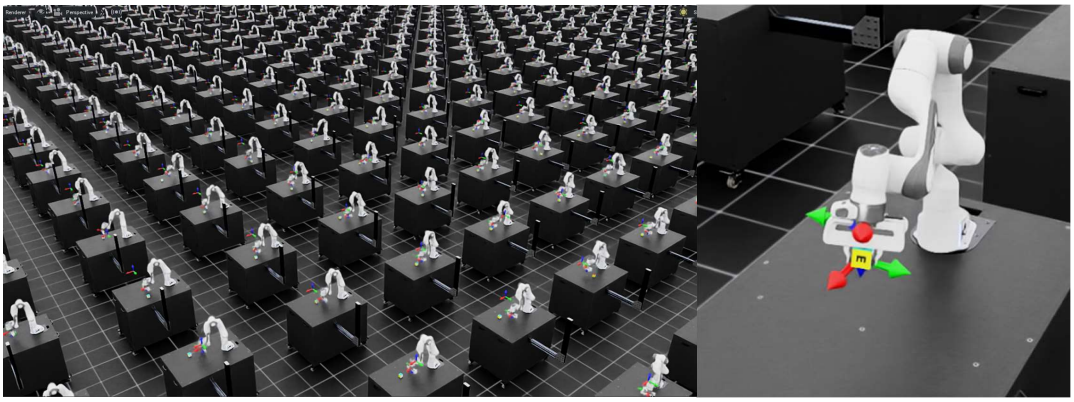


Figure 20: Grasping training simulation using Isaac Sim

8. CONCLUSION & FUTURE WORK

In conclusion, by combining all the subsystems, the robot is able to achieve the two primary tasks: item retrieval and customer guidance. It has:

- Human-robot interaction system to communicate with customers.

- Navigation system to find a path to the correct item's location.
- Image segmentation system and grasping system ensure the robot can grab the correct item.
- Face recognition system ensures the whole service is provided to the right customer.

However, there are still some limitations, future work tends to improve the robot by addressing the limitations.

For the voice detection system, AssemblyAI is used. However, the system can be improved by combining the GPT model with the voice-to-text function to achieve conversation with customers. For the Large Language Model, the most significant challenge is the dataset generated by ChatGPT which lacks variety in the pattern of sentences. More real-world data from customers can be collected when using the robot and that will grant many improvements to our model. The face recognition system can be further improved with advancements in model architecture to better handle covered face features or through training with datasets that include partially covered faces. For instance, replacing the final Euclidean distance metric with a fully connected neural network could enable nonlinear comparisons and make better use of the limited available facial features for identification. Creating a specialized dataset containing a diverse range of images featuring multiple objects helps for image segmentation system. This dataset will be used to further train the image segmentation model, aiming to improve its performance in scenarios involving multiple objects. For navigation learning based techniques could be explored, particularly socially aware navigation models to enable to robot to function around humans seamlessly. Lastly for grasping further inverse reinforcement learning could be applied to generate a denser reward function and further tuning can be performed. Therefore, by improving each subsystem of the robot, we expect the robot to achieve more accurate and reliable results in real-world applications.

9. REFERENCE

[1] W. Peebles and S. Xie, “Scalable Diffusion Models with Transformers,” Proceedings of the IEEE International Conference on Computer Vision, pp. 4172–4182, Dec. 2022, doi: 10.1109/ICCV51070.2023.00387.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2016-December, pp. 779–788, Jun. 2015, doi: 10.1109/CVPR.2016.91.

[3] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” NAACL HLT 2019 - 2019 Conference of the North American

Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference, vol. 1, pp. 4171–4186, Oct. 2018, Accessed: Dec. 23, 2024. [Online]. Available: https://arxiv.org/abs/1810.04805v2

[4] B. Singh, R. Kumar, and V. P. Singh, "Reinforcement learning in robotic applications: a comprehensive survey," Artif Intell Rev, vol. 55, no. 2, pp. 945–990, Feb. 2022, doi: 10.1007/S10462-021-09997-9/FIGURES/13.

[5] K. Sharma, "THE IMPACT OF E-COMMERCE ON OPERATIONAL COST EFFICIENCY IN MODERN BUSINESSES," Sachetas, vol. 3, no. 3, pp. 56–62, Aug. 2024, doi: 10.55955/330006.

[6] "Apollo." Accessed: Dec. 23, 2024. [Online]. Available: https://apptronik.com/apollo

[7] "Matthew Irvine Brown - Everyday Robots." Accessed: Dec. 23, 2024. [Online]. Available: https://www.irvinebrown.com/?p=1748

[8] "(9) Post | LinkedIn." Accessed: Dec. 23, 2024. [Online]. Available: https://www.linkedin.com/posts/reflexrobotics_reflexs-mission-is-to-build-affordable-generalpurpose-activity-7173287145963511809-z69X/

[9] "Home | AssemblyAI Docs." Accessed: Dec. 23, 2024. [Online]. Available: https://www.assemblyai.com/docs

[10] "Speech-to-Text API | AssemblyAI." Accessed: Dec. 23, 2024. [Online]. Available: https://www.assemblyai.com/products/speech-to-text

[11] C. Zhou, C. Sun, Z. Liu, and F. C. M. Lau, "A C-LSTM Neural Network for Text Classification," Nov. 2015, Accessed: Dec. 23, 2024. [Online]. Available: https://arxiv.org/abs/1511.08630v2

[12] A. Vaswani et al., "Attention Is All You Need," Adv Neural Inf Process Syst, vol. 2017-December, pp. 5999–6009, Jun. 2017, Accessed: Dec. 23, 2024. [Online]. Available: https://arxiv.org/abs/1706.03762v7

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional Transformers for language understanding," arXiv.org, https://arxiv.org/abs/1810.04805 (accessed Dec. 25, 2024).

[14] X. Song, A. Salcianu, Y. Song, D. Dopson, and D. Zhou, “Fast WordPiece Tokenization,” EMNLP 2021 - 2021 Conference on Empirical Methods in Natural Language Processing, Proceedings, pp. 2089–2103, Dec. 2020, doi: 10.18653/v1/2021.emnlp-main.160.

[15] M. Jiang and J. Dsouza, “Improving Scholarly Knowledge Representation: Evaluating BERT-based Models for Scientific Relation Classification”.

[16] I. Loshchilov and F. Hutter, “Decoupled Weight Decay Regularization,” 7th International Conference on Learning Representations, ICLR 2019, Nov. 2017, Accessed: Dec. 23, 2024. [Online]. Available: https://arxiv.org/abs/1711.05101v3

[17] Introducing Chatgpt | openai, Accessed Dec. 25, 2024. [Online]. Available: https://openai.com/index/chatgpt

[18] “ageitgey/face_recognition: The world’s simplest facial recognition api for Python and the command line.” Accessed: Dec. 23, 2024. [Online]. Available: https://github.com/ageitgey/face_recognition

[19] “dlib C++ Library.” Accessed: Dec. 23, 2024. [Online]. Available: http://dlib.net/

[20] “Image Segmentation with Pytorch.” Accessed: Dec. 23, 2024. [Online]. Available: https://www.kaggle.com/code/nikhil1e9/image-segmentation-with-pytorch

[21] “ahmedfgad/Mask-RCNN-TF2: Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow 2.0.” Accessed: Dec. 23, 2024. [Online]. Available: https://github.com/ahmedfgad/Mask-RCNN-TF2

[22] “MVTec Densely Segmented Supermarket Dataset (MVTec D2S).” Accessed: Dec. 23, 2024. [Online]. Available: https://www.mvtec.com/company/research/datasets/mvtec-d2s

[23] K. Zhu and T. Zhang, “Deep reinforcement learning based mobile robot navigation: A review,” Tsinghua Sci Technol, vol. 26, no. 5, pp. 674–691, Oct. 2021, doi: 10.26599/TST.2021.9010012.

[24] J. A. Placed et al., “A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers,” 2023.

[25] “gmapping - ROS Wiki.” Accessed: Dec. 23, 2024. [Online]. Available: http://wiki.ros.org/gmapping

[26] G. Grisetti and C. Stachniss, “Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling”.

[27] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, “Robust Monte Carlo localization for mobile robots,” Artif Intell, vol. 128, no. 1–2, pp. 99–141, May 2001, doi: 10.1016/S0004-3702(01)00069-8.

[28] D. Fox, “KLD-Sampling: Adaptive Particle Filters”.

[29] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” arXiv.org, https://arxiv.org/abs/1509.06461 (accessed Dec. 25, 2024).

[30] S. Joshi, S. Kumra, and F. Sahin, “Robotic grasping using deep reinforcement learning,” 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), pp. 1461–1466, Aug. 2020. doi:10.1109/case48305.2020.9216986

[31] “Proximal Policy Optimization — Spinning Up documentation.” Accessed: Dec. 23, 2024. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. K. Openai, “Proximal Policy Optimization Algorithms,” Jul. 2017, Accessed: Dec. 23, 2024. [Online]. Available: https://arxiv.org/abs/1707.06347v2

[33] A. A. Shahid, L. Roveda, D. Piga, and F. Braghin, “Learning Continuous Control Actions for Robotic Grasping with Reinforcement Learning,” Conf Proc IEEE Int Conf Syst Man Cybern, vol. 2020-October, pp. 4066–4072, Oct. 2020, doi: 10.1109/SMC42975.2020.9282951.