Emotion Recognition Companion Robot

By
Lau Tin Tin

A REPORT
SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfillment of the requirements
for the degree of
BACHELOR OF INFORMATION SYSTEMS (HONOURS) INFORMATION SYSTEMS ENGINEERING
Faculty of Information and Communication Technology
(Kampar Campus)

FEBRUARY 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Dr Aun Yichiet who has given me this bright opportunity to engage in robotic related project. It is my first step to establish a career in robotic and Artificial Intelligence (AI) field. A million thanks to you.

Special thanks to UTAR lab officer and fellow friends also who always offer necessary help for me to complete this project. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

Existing companion robots often lack the depth to recognize a broad range of human emotions or engage in context-aware conversations. These limitations reduce their effectiveness in providing genuine companionship and emotional support. This project introduces Wall-E which is an intelligent companion robot designed to bridge this gap by combining computer vision and large language models to interpret user emotions and generate empathetic responses. This robot is composed of emotion detection module and the conversational response module. Each of the module is implemented as independent ROS2 nodes to ensure modularity and asynchronous communication. The emotion detection module employs a CNN model to analyse facial expressions captured in real time. Detected emotions are published through ROS2 topics for use by other nodes. Simultaneously, the conversational response module enables voice-based interaction. It captures user speech using a microphone, transcribes it with the Whisper API and combines the resulting text with the detected emotion. These inputs are passed to an OpenAI GPT agent to generate emotionally relevant responses. The replies are then converted to speech using the gTTS engine and played through the robot's speaker. The integration of ROS2 allows seamless communication between modules using a publish-subscribe architecture. A structured verification plan was implemented to ensure system reliability including testing of the CNN model for emotion classification, audio transcription accuracy, context-aware response generation, and speech synthesis clarity.

Area of Study: Human-Robot Interaction and Deep Learning

Keywords (Minimum 5 and Maximum 10): Companion Robot, Facial Emotion Recognition, Context-Aware Conversation, Human-Robot Interaction, ROS2, Large Language Model, Convolutional Neural Network

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

2D	Two-Dimensional
API	Application Programming Interface
AI	Artificial Intelligence
CNN	Convolutional Neural Network
FER	Facial Emotion Recognition
FYP	Final Year Project
gTTS	Google Text-to-Speech
GPT	Generative Pre-trained Transformer
ID	Identifier
IDE	Integrated Development Environment
JAFFE	Japanese Female Facial Expression Database
LED	Light Emitting Diode
LLaMA	Large Language Model Meta AI
LLM	Large Language Model
NLP	Natural Language Processing
PaLM	Pathways Language Model
PC	Personal Computer
ReLU	Rectified Linear Unit
ROS2	Robot Operating System 2
SBC	Single-Board Computer
SGD	Stochastic Gradient Descent
SSH	Secure Shell
TTS	Text-to-Speech
WAV	Waveform Audio File Format

CHAPTER 1

Project Background

1.1 Introduction

In recent years, challenges related to psychological health have become increasingly prominent especially in maintaining mental and emotional well-being. Issues such as loneliness, stress, and the growing demands on our time and attention have made it more difficult for individuals to find the emotional support they need. This problem became evident especially during the COVID-19 pandemic where many people are found themselves increasing isolated [1]. Therefore, this has the need for innovative solutions that can provide support to help individuals navigate these emotional challenges.

As technology getting more advances, companion robot with a broad range of capabilities in helping human relieve emotional problems is needed. These robots are designed to offer companionship, emotional support and form meaningful connections with their users. It can be achieve by utilizing advancements in artificial intelligence and robotics where the robot can equipped with the emotion recognition and conversation abilities. These robots can deliver responses that are more relevant and supportive by leveraging their ability to detect emotions and respond accordingly. For example, robot can recognize signs of sadness during a moment of inactivity and initiate a conversation to provide encouragement and improve user’s mood.

Therefore, this project is proposed to develop a companion robot that can effectively understand and respond to human emotions and speech. Although the emotions can be express through hands, voice, gestures and facial [2] but the major focus will be on facial expressions emotions. This is because facial expressions play a key role in human communication as they convey information like emotions, intentions, responses and reactions of the people [2]. This non-verbal communication is essential in various contexts especially for human robot interactions because it enables robots to better understand, respond and engage more naturally with humans.

1.2 Problem Statement and Motivation

Nowadays, social isolation and feelings of loneliness are increasingly recognized as significant public health issues that affect millions of people across all age groups [3]. For instance, the increasing demands of work and social life often lead to a growing sense of isolation and loneliness among individuals. These feelings are not just about being alone but they can lead to serious health problems like depression, heart disease and even dementia [3]. The COVID-19 pandemic made things even worse because lockdowns and social distancing cut people off from their usual support networks [3]. Now, many people are still struggling with the emotional aftermath as we return to a new normal. Although there is advancements in communication tools, **people are still struggling to find meaningful connections that can provide emotional support**. Therefore, there’s a pressing need for new solutions to help people feel less isolated.

Companion robots could be a big part of the answer to fill the emotional gap left by the lack of human interaction. The ability to interpret and respond to human emotions is crucial for these robots to form more supportive and deeper connections with users. However, **current companion robots often struggle in identifying more complex emotional states**. Many robots can only identify basic emotional cues such as happiness, sadness, or anger but struggle to recognize more complex emotional states like fear, disgust, or surprise. This shortcoming limits their ability to offer genuine emotional engagement and results in a less personalized user experience. Furthermore, most companion robots lack the ability to engage context-aware conversations with users. While some robots are equipped with basic conversational abilities but these interactions are often **limited to static or pre-programmed responses that lack the emotional intelligence required for deeper engagement**. As a result, conversations with these robots can feel mechanical and impersonal, undermining the potential emotional support they could offer. Therefore, this project is proposed with the aim to develop a companion robot that overcomes these limitations by integrating FER model for real-time emotion detection and a LLM for contextual conversations. This robot will be able to identify a broader range of emotions and engage in context aware interactions. By doing so, robots can provide more thoughtful responses that are adapt to the emotional state of the user and offer better companionship to the user.

1.3 Project Objectives

The main aim of the project is to develop a companion robot that leverages LLMs and computer vision to offer companionship to its user through contextual feedback and audio cues. The project seeks to address the limitations of current companion robots which often struggle with emotion recognition and conversational abilities. The robot tend to provide more personalized and empathetic interactions with its user by integrating LLM for conversational intelligence and FER model for real-time emotion analysis. Therefore, the objectives of the project are:

1. To develop a sensor-laden companion robot that understands user’s emotions and speeches on Turtlebot3.

2. To integrate emotion recognition using FER model to recognize user’s emotion in real time

3. To integrate LLM to understand user’s speech and queries and provide contextual-aware response to the user

1.4 Project Scope

The project aims to deliver a fully functional companion robot with advanced emotion recognition and conversational capabilities integrated into a Turtlebot3 platform. At the end of the project, the robot will be equipped with a real-time FER model that can accurately detect and interpret seven fundamental human emotions including happy, sad, angry, surprise, disgust, fear, and neutral. These emotions were chosen as they cover a wide range of universally expressed positive and negative emotional states which make the robot capable of responding to a broad spectrum of user emotional cues.

In addition to emotion detection, the robot will be equipped with LLM integration which enabling it to hold context-aware conversations with users. This feature will allow the robot to offer personalized feedback and emotional support based on the user's emotional state and conversational context. The combination of real-time emotion recognition and conversational abilities will result in a robot that can serve as a more dynamic companion which providing emotional support in a variety of situations. The final deliverable will include the robot's physical prototype that integrated with FER model and LLM.

1.5 Contributions

This project offers significant contributions to both the field of companion robotics and to society by addressing the critical issues of social isolation and mental health challenges. This project aims to enhance the ability of robots to provide personalized emotional support by equipping companion robots with advanced emotion recognition and conversational capabilities. In a world where mental health issues such as anxiety, depression, and loneliness are on the rise, particularly after the social disruptions caused by the COVID-19 pandemic, the need for accessible emotional support systems is more urgent than ever. The development of a robot capable of accurately recognizing a wide range of emotional states in real-time and responding in an empathetic manner has the potential to improve emotional well-being for users who may not have immediate access to human support.

Furthermore, the integration of LLM into the robot enables more contextually aware conversations that able improve user interaction. This innovation positions the project as a forward-thinking solution to current limitations in human-robot interaction particularly for individuals facing emotional struggles. Beyond its immediate application, this project sets the stage for future advancements in companion robotics with potential benefits in healthcare, eldercare, and personal wellness.

1.6 Report Organization

This report is structured into seven chapters. Chapter 1 introduces the project by presenting the background, problem statement, objectives, scope and key contributions. Chapter 2 provides a comprehensive literature review of existing companion robots, deep learning-based FER approaches and various LLMs, comparing their strengths and limitations to justify the chosen methods. Chapter 3 outlines the system methodology, development workflow, testing strategies, and project timeline. Chapter 4 focuses on system design using use case and activity diagrams to visually describe the interactions between the system components and users. Chapter 5 discusses the development of both the emotion detection and conversational modules. Chapter 6 presents the evaluation results, covering model performance, conversation response testing, and user feedback collected through questionnaires. Finally, Chapter 7 concludes the report by summarizing the key outcomes and offering recommendations for future work. Supplementary materials such as references, appendices, and the project poster are included to provide additional support and insights.

CHAPTER 2

Literature Review

2.1 Capabilities of Current Companion Robot

There are several companion robot reviewed below to know their capabilities. The reviewed robot including stress detection robot by Mayurawasala et al. [4], Aibo [5], Tombot [6], Paro [7], Miko [8], Moxie [9], Lovot [10], Emo [11], Kiki [12] and Buddy [21].

Companion robot developed by Mayurawasala et al. will be used as a stress intervention tool especially for individuals who are working from home [4]. The idea for developing such companion robot comes from the high stress levels faced by many professionals or workers due to the social isolation, lack of support and blurred boundaries between personal and professional lives. Therefore, this companion robot being designed detect user stress through FER and provide emotional support to reduce user's stress level. The robot is designed to engage with users only when necessary to reduce the distractions for workers. The development of stress reducing companion robot consist by a comprehensive system architecture which integrated stress detection module, conversational module and robot interface.

Besides, Sony's robotic pet dog AIBO is one of the famous robot nowadays [5]. It is designed to act like a real pet that offer companionship and entertainment to the user. This robot can perform a range of dog-like behaviours like walking, running, playing, recognizing owner's voice and respond to commands, learning tricks, identifying and interacting with owner [5]. However, it does not have emotion recognition capabilities. AIBO can only recognize and remember the face of its owners and other frequent companions. This allows AIBO to adjust its behaviour based on the person it interacts with and provide companionship to them so that they does not feel lonely or isolated [5].

Tombot also is a robotic dog that designed specifically for elderly individuals especially for those elderly with dementia or Alzheimer [6]. It is similar to AIBO with aims to provide comfort and companionship without the responsibilities of care. It can act like a real dog, bark, wag its tail and respond to touch. It can also receive voice commands from the owner and make respond to them [6]. Tombot often been used as therapeutic companion for elderly because it is easy to interact and it does not require feeding or walking. So, it is ideal for elderly people to reduce feelings of loneliness, anxiety, aggressive behaviours and provide them positive emotion [6]. However, Tombot does not have emotion recognition capabilities and it cannot move. It is just design to respond to general interactions.

Paro is another therapeutic robot that designed to provide comfort and emotional support for those elderly patients or cognitive impairments patients [7]. It also can moving its head, blinking its eyes, and making sounds when people pet it [7]. Paro also able respond to touch and show its emotions as it interacts with elderly [6]. However, it cannot receive voice commands from its owner [6]. The major function is to provide companionship to the elderly. Paro is widely used in healthcare sector like hospitals, nursing homes, and care facilities to provide therapeutic benefits such as reducing patient stress, improving mood, and encouraging social interaction of older adults who living in care centre [7]. However, Paro also same as AIBO and Paro cannot recognize human emotion, human activities and cannot move around.

Other than that, there are also few companion robot that specifically designed for children. One of it is Miko robot which act as an educational companion that support children's learning and development and also provide companionship and entertainment [8]. For instance, it can engage kids in interactive learning and play, answer questions from children and participate in discussions with children. This robot has offers many educational games and activities that are tailored to children's age and learning level [8]. It can also detect children's emotion, engage in conversations with children and provide encouragement when needed. However, it does not have detailed activity recognition capabilities like interpreting specific tasks or activities in real-time. It also cannot move around.

Furthermore, Moxie also one of the companion robot specifically designed for children. Moxie itself does not focus on emotion detection from facial expressions. Instead, Moxie uses its conversational abilities and interactive features to engage with children in a way that can seem emotionally responsive [9]. It might adjust its tone or content based on the context of the conversation. The major functionalities are just similar to Miko robot, the only difference is Moxie robot can provide personalized experiences where it can learns from its interactions with the children and tailor its activities and responses to the child's development needs and preferences [9].

Lovot is another social robot that aims to become a lovable companion robot to make people happy [10]. It can detect user's emotions and make respond through sounds, eye movements, and body language but it cannot make conversation [10]. This robot able to moves around autonomously and seeking interaction with users. Same as previous robot, it primarily used as a companion for emotional support especially for people who feel lonely.

Meanwhile, Emo is a desktop AI companion robot that uses AI to interact with its environment. It can use facial recognition and emotion detection technologies to analyse user's emotional states. It can also adjusts its responses and interactions to provide appropriate emotional support. Besides, Emo also can tell the current time to the users, play music, dance to the beats and respond to voice commands with facial expressions and movements [11]. It is also a great helper that can wakes user up, help user turns on light, take pictures and answers to user's doubts or questions [11]. Emo can also move around autonomously and able to avoid obstacles [11].

Buddy almost same with Emo robot. It is a social robot designed to assist families with various tasks while also providing companionship and entertainment to users [21]. Same as Emo, Buddy can recognize human emotions, help with daily tasks such as make reminders to users, play games and dance to beats [21]. It also able to control smart home devices and monitor home security [21]. Not only that, it also strong enough to become an educational companion because it can teach basic coding skills. Overall, it is mainly used in homes for family companionship and able perform home automation as well as works well in educational settings for interactive learning experiences.

Kiki is another robot that similar to Emo and Buddy. These three robots has ability to express a range of emotions through its digital face and can react to different stimuli like sounds and touch [9],[11],[12]. However, Kiki cannot recognize human emotions. It can only simulate its own emotional expressions on its own digital face [12]. It's reactions and moods will change depending on how it is treated. Kiki also equipped with ability to engage in verbal communication where it can participate in simple conversations with users and respond to questions from users. These robots are designed and used as personal companion to provide emotional support, companionship and entertainment to those people who feel lonely and isolated.

	Emotion recognition	Action recognition (interpret human activity)	Conversation ability (e.g.: talk to user)	Recognize people and adjust behaviour	Self-navigation	Obstacle avoidance ability
Stress Detection Robot	YES (but detect stress only)	NO	YES	NO	NO	NO
Aibo Robot Dog	NO	NO, it only able respond to touch and voice commands	NO	YES	YES	YES
Tombot Robot	NO	NO (same as Aibo)	NO	NO	NO	NO
Paro Robot	NO	NO (same as Aibo)	NO	NO	NO	NO
Miko Robot	YES	NO	YES	NO	NO	NO
Moxie Robot	NO	NO	YES	YES	NO	NO
Lovot Robot	YES	NO	NO	YES	YES	YES
Emo Robot	YES	NO	YES	NO	YES	YES
Buddy Robot	YES	NO	YES	YES	YES	YES
Kiki Robot	NO	NO	YES (make response)	NO	NO	NO

Table 2.1.1 Summary of Capabilities of Current Companion Robot

Based on the above reviews, majority of the robot has the emotion recognition capability except for those pet-like robot like AIBO, Paro and Tombot. However, most of the robot lack the ability to truly understand and interpret human emotions beyond simple cues. **They only recognize basic feelings like happiness, sadness and anger but struggle with other emotional states like surprise, fear and disgust**. Robots with limited emotional understanding often provide static or pre-programmed responses. For instance, several robots such as Miko, Moxie, Emo, Buddy and Kiki can engage in conversations but these interactions are often limited and can feel mechanical. The **responses may not always be context-aware or emotionally intelligent enough** to offer personalized support. Therefore, there is a clear need for more advanced systems that can go beyond simple emotion recognition and offer deeper emotional engagement. In this project, a companion robot that integrated deep learning-based FER models and LLM is proposed to recognize a wider range of emotional states and adjust their respond in ways that are adaptive and context aware. In the next sections, I will make some review to the methodologies in developing an effective FER model. This includes exploring deep learning techniques for emotion classification.

2.2 Previous works on FER via deep learning

During previous time, traditional machine learning approached to FER have been seen considerable success through the use of handcrafted features for emotion classification. However, traditional way of neural network often struggle in image classification due to its difficulty in capturing complex features. Therefore, the emergence of deep learning techniques particularly CNNs has become the major focus for most of the researchers because of its automatic recognition ability. In other words, it is also means that CNNs model has the capability to learn features directly from the raw pixel data of images. So, deep learning will be used to develop a model to interpret facial expressions and extract features. Several review has been done below to get insight the better way to achieve a better performance.

2.2.1 FER full learning model using CNN

This section more discuss on the overall step and process the build a full model from scratch for FER. The first step usually involves data preprocessing where the image from the dataset will be process into format that fit as input of model. According to Modi and Bohara, data preprocessing process include conversion of image to grayscale from RGB, face detection and image resizing [13]. After data preprocessing finish, those image will undergo segmentation process and feature extraction. The extracted feature or data then send for training using classifier and testing to get the result of recognition. Therefore, the below part will review more on the classifier used and their accuracy result after training.

Jaiswal et al., Sarvakar et al. and Singh et al. all are developed using Keras for building neural networks to simplify the development of neural networks by allowing developers to focus on creating and connecting layers rather than dealing with individual neurons [14][15][16]. Singh et al. make use of self-collected dataset meanwhile Jaiswal et al. and Sarvakar et al. make use of the online sources datasets FER-2013 and JAFFE to evaluate the model. FER-2013 dataset then resized to 48 x 48 pixels and JAFEE dataset been resized to 128 x 128 pixels. Then, all these images will be use as input for training the model. Equal number of samples for each emotion class is needed to avoid bias model.

However, there is some different in terms of the CNN architecture. For Jaiswal et al.'s model, there will be two identical submodels that perform the same tasks where they extract features from images in parallel [14]. Each submodel consists of a convolutional layer followed by local contrast normalization layer, max pooling layer, flatten layer and dense layer [14]. This local contrast normalization layer used to improve the quality of the feature maps by removing local pixel averages while the max pooling layer used to reduce spatial dimensions and enhance processing speed [14].

Modi and Bohara, Sarvakar et al. and Singh et al.'s neural network model architecture are unlikely the same just only different in number of layers involved in the classifier. All of them consists by convolutional, pooling, flatten and dense layers. For Sarvakar et al.'s model, total six two-dimensional convolutional layers, two maximum pooling layers, one flatten layer and two connected layers is used. For Modi and Bohara, there are 2 convolutional and pooling layer together with one dense layer. After the image been pre-processed, these images then pass to the first convolutional layer to extract features from image. Each model might use different number of filter or nodes for this convolutional layer. For Sarvakar et al.'s model, first two layers use 64 filter each and the subsequent four convolutional layers increase the number of filters to 128 then 256. This progression allows the network to capture increasingly complex features and enhance network ability to generalize across different features in the input images. For Singh et al. 's model, its first convolutional layer contains 64 nodes and the second has 32 nodes. Each of it handle two-dimensional matrices representing input images. Then these input image are transformed based on filter values and resulting in a feature map. The map will then activated using the ReLU function to convert make the values become positive. Then, the pooling layer involved to reduce the number of parameters when the image is too large. Some model also includes dropout layers to prevent overfitting issue by dropping neurons randomly during model training.

After that, the outputs from Modi and Bohara, Jaiswal et al., Sarvakar et al. and Singh et al. s' models are flattened and concatenated into single dimensional vector and passed to a fully connected layer (dense layers) for further analysis. The final classification is conducted through SoftMax output layer to output the probabilities corresponding to different emotions and classify image between 0 and 1 [14]. The model also employs dropout with rate 0.2 for [14] and rate 0.25 for [15] to avoid overfitting [14].

According to Singh et al., optimizer, loss function, and metrics are essential to compile the model. Jaiswal et al., Sarvakar et al., Singh et al. s' model are trained using the optimizer called Adam which is known for its ability to adapt the learning rate for each weight in the network. This optimizer will adjusts the learning rate throughout the training process to find the optimal weight efficiently. The categorical cross-entropy loss function is used because of its standard for multi-class classification problems. At the end, the model has successfully reduced the loss function after using Adam optimizer. 'Accuracy' metric is used to monitor model's effectiveness during training. Singh et al. has achieve 93% with comparable training and validation accuracy. Jaiswal et al. has achieve 70.14% accuracy for FER-2013 dataset and 98.65% for JAFFE dataset. Sarvakar et al. has achieve 55% accuracy for FER-2013 dataset. Lastly, Modi and Bohara has achieve about 82.51% of the accuracy rate after 35 epochs.

All of them make use different control factors where they make use different number of convolutional layers and different training parameters or number of epoch which can affect the performance and accuracy of the model sometimes. This number of convolutional layers can potentially affect the ability of model to learn from the complex data [18]. The more the layers, the better the model to capture complex patterns. However, it comes with the risk of overfitting also which cause the model fits too specifically to the training data and does not generalize to the new data [18]. The number of epochs can affect learning process of the model. The model can potentially become underfit if the number of epochs too few, but the model can become overfit if the number of epochs too many. Therefore, it is important make use suitable number of convolutional layers and epochs to optimize the performance of the model.

2.2.2 FER using Transfer Learning

Transfer learning is the process of using a learned model and optimizing it for a particular job, like recognising emotions on faces. This technique is commonly utilized to improve learning and training efficiency [17]. This transfer learning also means transfer information from source domains to the other new domains [17]. These pre-trained models are originally trained on big dataset. However, in order to make the model works well in specific task, the model needs to fine-tune using facial-expression dataset to retrain later layers of the network while also keeping the early layers fixed. The model created by this approaches not only reduce the parameters of the CNN but also improve the accuracy of the model [18]. There are four pre-trained model will be reviewed including Inception-V3, MobileNet-V2, VGGNet-19 and Alexnet.

Inception-V3 is studied and used because of its high accuracy in the implementation of previous model. It has more complex architectural designs which combined several different sizes of convolution within the same module [18]. This unique approach enables it to capture patterns at multiple scales which make it effectively learn complex features with greater accuracy [18]. Besides, the model has the ability to process information across various spatial hierarchies [18]. Therefore, the model can achieve high performance on a huge range of tasks especially in picture classification. One of the key advantages of Inception-V3 is its efficient use of parameters [18]. The network can learn more complex features at the same time it can manage to avoid overfitting by reducing the number of parameters [18]. However, this complexity comes at the cost of longer training times due to it has over 24 Million parameters inside the model [18]. Therefore, the model requires substantial computational resources and time to train effectively [18].

Meanwhile, MobileNet-V2 pre-trained model also widely use because of its small size [18]. It designed with around 3.4 million parameters which makes it retain lightweight and suitable to deploy on smaller devices with limited resources [18]. This model also achieved high accuracy in many previous model [18]. The depthwise separable convolutions used by this MobileNet-V2 pre-trained model which break down the traditional convolutions into depthwise convolutions and pointwise convolutions [18]. This separation makes it reduce the number of parameters and the computational cost while also helps maintain high accuracy [18]. Overall, this MobileNet achieved a balance between performance and resource usage making it widely use in much research [18]. Additionally, this MobileNet was developed with linear bottlenecks and an inverted residual structure [18]. This design has further enhanced efficiency of model because it helps to optimize the information flows through the network [18]. In other words, it also means the model able capture essential features while keeping the network lightweight [18].

VGGNet-19 and Alexnet are another pre-trained models that frequently use by the researchers in transfer learning. The VGGNet-19 model is composed of three fully connected layers with learnable weights and sixteen convolutional layers [17]. Convolutional layers used to extract features from the input images. Fully connected layers or dense layers used for image classification [19]. This model also contains many parameters and deeper network structures [17]. Therefore, it required larger storage and computationally expensive to train model from scratch. If VGG19 is used for its pre-trained weights on large datasets, it can significantly reduce training time. Researchers often leveraged its pre-trained weights and using fine-tuning techniques to train the model perform specific task such as emotion recognition. On the other hand, Alexnet consist of 8 layers only with 5 convolutional layers and 3 fully connected layers [19]. This model consists of approximately 60 million parameters [19]. So, it is relatively suitable for medium model although it has high number of parameter because it has simpler architecture design compared to Inception-V3 and VGG-19. However, it can potentially cause overfit for the model if dataset too few [19].

These pre-trained model will then fine-tune to make it able to perform emotion recognition. There are three approaches to achieve it. In the first method, all of the pre-trained model's convolutional layers are frozen to prevent the convolutional network's weights from changing while it is being trained [18]. Afterwards, the previously trained model's classification head is not utilized [18]. The model is expanded with a new classification head, which is then trained to produce a new model [18]. Another approach of transfer learning is called mid-layer feature extraction where only few layers are freeze unuse and the rest are fine-tuned. Agung et al.'s model is leveraging the last 50% of the convolution network in the pre-trained model. Therefore, the first 50% of the layers are freeze and the remaining 50% of the layers are unfreeze [18]. The training process focuses on fine-tuning to this portion layers in convolutional network to make it able to perform emotion expression classification [18]. Third approach is called full model finetuning where last layer is replaced with another new layer and unfreeze whole model to allow weight change during training process [19]. There two method are the commonly use method to perform transfer learning on the pre-trained model. In summary, the first method focuses on retain the previously trained weights and add new layers to make it able to perform specific task. Second method focuses on retrain last 50% of the layers of the pre-trained model to make it align with the current case.

Agung et al. has proposed three models where two model made use of the pre-trained model and another one build from scratch. Then, Agung et al. has discovered that the training model Inception-V3 and MobileNet has better total accuracy value than full learning model. Inception-V3 pre-trained model get accuracy value at 0.96, Accuracy values for MobileNet-V2 and the complete learning model are 0.89 and 0.87, respectively [18]. Joseph and Mathew also proposed two models to make comparison between pre-trained model and fully learning model. The pre-trained model adopted is MobileNet where the optimized model is exported and deployed on an Android device [20]. This pre-trained model been fine-tune afterwards using cleansed FER-2013 dataset and obtained an accuracy of 75.55% [20]. Full learning model was trained using CNN architecture and FER-2013 dataset. Then, this model obtained an accuracy of 67.18% [20]. Meena et al. also proposed VGG-19 pretrained model and make use it with FER-2013 datasets. The accuracy achieved using the VGG-19 is 65% [17]. Raja Sekaran et al.'s fine-tuned Alexnet model also achieved high accuracy with 70.52% for FER-2013 dataset.

2.3 Review on LLM

LLM particularly those based on transformer architectures have become foundational in NLP due to their ability to generate human-like text based on input. LLM are widely used because it make the system able understand user queries and generate appropriate responses. These models excel in processing complex language inputs and generating contextually relevant responses that is essential for companion robots to offer emotional support and companionship. Besides, these LLM demonstrated strong generalization from their training data. This enable the robot to able respond dynamically according to new inputs including emotion-based queries. There are few LLM will be reviewed including GPT-3, PaLM, LLaMA and Gemini to identify the most suitable one for integration into TurtleBot 3.

2.3.1 GPT-3

GPT-3 is one of the advanced model in the GPT family. Although there is newer version of GPT released nowadays which capable to process both text and image inputs and generating text-based outputs [30] but GPT-3 can be enough if only text-based input involved [34]. Besides, GPT-3 well known for its strong text generation and understanding capabilities [34]. For instance, it can provides contextually responses and handles wide range of conversational topics effectively. Furthermore, this LLM is the most well-known model and it has many proven track record of integration in various applications. However, cloud-based APIs is necessary to ensure the model run efficiently. This could be a consideration for integration into a resource-constrained environment like TurtleBot3.

2.3.2 PaLM 2

PaLM is developed by Google that employs the Pathways architecture [31]. This model is designed with capability in handling complex language tasks and understanding context more deeply. For instance, it is designed to be more efficient in specific language tasks where it can handle complex queries and generate multilingual content [31]. Besides, PaLM has offer different model sizes including Gecko, Otter, Bison, Unicorn making each of the model optimized for specific use cases [31]. However, the deployment of this LLM also require significant computational resources [31] which could impact deployment on devices with limited computational power. Same as GPT-3, cloud-based APIs is necessary to provide more computational support for TurtleBot 3.

2.3.3 LLaMA 3

LLaMA3 is another latest language models released by Meta company [32]. This LLM is open source which commonly used by the research community to access, modify, and build upon the model [32]. This model has improved efficiency with Grouped Query Attention and trained on a large dataset which enhancing performance for coding and reasoning tasks. Therefore, this model tend to offer performance-to-resource ratios which beneficial for those resource constraint device. Besides, it offer both foundation language models and instruction-tuned variants for different tasks [32]. This allow the developer tailor the model to fit their specific needs. However, this LLM is not a multimodal LLM where it only accept text-based input.

2.3.4 Gemini

Gemini is a multimodal LLM designed handle different types of inputs like text, code, audio, images and video [33]. Besides, it is adept at engaging human-like conversations where it able adapt user’s writing style and talk about different topic [33]. Its capabilities extend to expressing emotions like joy in positive situations and offer apologies in response to errors [33]. Besides, it is excels in generating contextually relevant text responses instead of memorized data. It is also efficient across many devices which making it versatile for different hardware environments. However, LLM that handles diverse input types can lead to high resource usage in terms of memory and higher processing times. This might result in delays in interaction and cause less fluid conversational experiences.

2.3.5 Strength and Weaknesses of each reviewed LLM

LLM	Strength	Weakness
GPT-3	- Excels in generating contextually relevant text responses. - Has many proven track record of integration in various applications.	-Required high computational resources. -Does not handle image inputs.
PaLM 2	-Designed to handle complex queries and multilingual content efficiently. -Offer different model sizes tailored for various use cases. -Better at understanding and generating contextually rich responses.	-Required high computational resources which may impact deployment on resource-constrained devices. -May required additional cloud-based APIs for optimal performance.
LLaMA3	- Improved performance-to-resource ratios which are making it suitable for devices with limited computational power. - Open source LLM which allows customization and modification. - Flexible variants where it offers both foundational and instruction-tuned models for different tasks.	- Does not handle image inputs.
Gemini	-Able handle different input types like text, code, audio, images and video. -Efficient across many devices making it versatile for different hardware environments. - Excels in generating contextually relevant text responses instead of memorized data.	- Requires significant memory and processing power, which could lead to delays and less fluid interactions in a resource-constrained environment.

Table 2.3.1 Strength and Weaknesses of each reviewed LLM

2.4 Limitation of Previous Studies

According to the review above regarding to the capabilities of current companion robot, **majority of the robot only can recognize basic feelings like happiness or sadness but struggle to recognize other complex emotional states** surprise, fear, disgust and angry. Robots with limited emotional understanding often provide static or pre-programmed responses. For instance, several robots such as Miko, Moxie, Emo, Buddy and Kiki can engage in conversations but these interactions are often limited and can feel mechanical. The **responses may not always be context-aware or emotionally intelligent enough** to offer personalized support. Therefore, there is a clear need for more advanced systems that can go beyond simple emotion recognition and offer deeper emotional engagement.

Moreover, many companion robots **does not have ability to recognize, interpret and respond to complex human activities**. For instance, some robots can recognize basic actions like touch and voice commands but they often lack the ability to understand detailed human activities. This human activity detection is crucial for improving the interaction and communication between robots and humans. Emotion recognition alone provides insight into a user's emotional state but may not fully capture the context in which those emotions occur. Therefore, it is important to integrate human activities detection capabilities into robot where a robot not only can understand user feeling but also know what they are doing. When a robot understands both the emotional state and the activities of a user, it can offer more personalized and effective responses. For instance, if a robot detects that a user is moving between different activities like transitioning from work to relaxation, it can adjust its interactions to match the changing context such as offering motivational messages during work and relaxation prompts during downtime.

2.5 Chapter Summary

In summary, current capabilities of companion robots reveal significant limitations in their ability to recognize and respond to human emotions. Many companion robots typically have limited emotional recognition capabilities where they only recognize basic emotions like happiness and sadness. They often struggle with identifying more complex emotional states like surprise, fear, disgust, and anger. This limitation will leads to static and non-adaptive responses from these robots which limit them offering truly personalized and effective companionship.

This gap in emotional understanding underscores the need for more advanced systems capable of comprehensively interpreting a wider range of human emotions. A key approach to achieving this involves the use of FER models powered by deep learning techniques. According to recent studies, it has highlights that models employing transfer learning significantly outperform those built from scratch especially in the domain of emotion recognition. Transfer learning involves applying pre-trained models that have been developed on large datasets and then fine-tuning them for specific tasks like FER. Previous research has proven with consistent success of transfer learning and highlights its advantages over full learning models. Pre-trained models already contain rich features learned from vast amounts of data which allow them to be fine-tuned with less data and time but still achieving high accuracy. In the context of companion robots, employing transfer learning will be a good choice to enhance their ability to recognize a broader range of emotions.

Additionally, the integration of LLM into companion robots further strengthens their ability to provide meaningful interactions. This LLM can be combined with FER systems to offer deeper emotional engagement and context-aware responses. While traditional companion robots tend to offer static and pre-programmed replies, LLM allow for more adaptive and personalized conversations. At the end of the project, the robots can respond not only to detected emotions but also understand the context of conversations. This will providing more fluid and human-like interactions.

CHAPTER 3

System Methodology

The development of the companion robot follows an iterative and modular approach which inspired by the Agile methodology. The methodology focuses on building core features in phases, allowing for regular testing, feedback, and enhancements.

3.1 System Requirements

3.1.1 Hardware Requirements

The hardware involved in this project are

- Monitor
- Laptop
- Raspberry pi camera
- Turtlebot3
- Microphone
- Speakers
- LED

Monitor is used to connect to the Raspberry Pi for visual interactions with the terminal during configuration and ROS 2 installation processes. This monitor is essential for monitoring system outputs and debugging. Laptop is used for downloading Ubuntu, setting up ROS 2 and executing commands. It plays a crucial role in the initial setup, development, and testing phases of the project. Raspberry pi camera is used for capturing images which are essential for the emotion recognition model. Microphone use for capturing user's queries or response. Speakers use for outputting the synthesized voice responses from LLM. LED used to aware user with current states. Green colour LED indicated that the robot is in listening mode. Red colour LED indicated that the robot is in responding mode. TurtleBot3 serves as the primary robot for integration with the emotion recognition model. It is the platform where the emotion model will be implemented and tested.

Description	Specifications
Model	Victus by HP Gaming Laptop 15-fb0xxx
Processor	AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz
Operating System	Windows 11
Graphic	NVIDIA GeForce GTX 1650
Memory	8GB RAM
Storage	476 GB SSD

Table 3.1.1 Specifications of laptop

3.1.2 Software Requirements

Among the software used in this project development are

- ROS Distribution (Humble Hawksbill)
- Ubuntu 22.04
- Python (Version: 3.10.12)
- Jupyter Notebook
- Visual Studio Code (Version: 1.92.2)
- OpenCV (Version: 4.10.0)
- OpenAI Assistant
- Whisper Speech-to-Text Engine
- Google Text-to-Speech (gTTS)

3.1.2.1 ROS Distribution

ROS is an open-source framework been used for developing robotic applications. Its modular architecture enables developers to structure functionality into individuals “node” [22]. It give very great flexibility for the developer to build a scalable systems. For instance, different functionalities can be implemented in independent node. These nodes can run across multiple machines and be written in various programming languages [22]. ROS also can facilitates communication between different parts of the robot system components [22]. Besides, ROS also comes with different tools and libraries which is readily available [22]. For instance, ‘RViz’ can be used for 3D visualization and ‘Gazebo’ is a simulator that can be used to test and develop robot applications in virtual environment [22].

In this project, Humble Hawksbill was selected to be used. As of August 2024, it is one of the few actively supported ROS distributions and will receive updates until May 2027 [22]. This makes it a reliable choice for long-term robotics development.

3.1.2.2 Ubuntu 22.04

Figure 3.1.1 Ubuntu 22.04

There are many Linux distributions can be use in the ROS development including Ubuntu, Debian, Fedora, Arch Linux and OpenSUSE [24]. In this project, Ubuntu selected to use because it is the most famous and widely used Linux distribution for ROS development. Besides, it is highly compatible with ROS and many ROS packages available on Ubuntu are well-maintained and up to date [24]. Additionally, a significant community of developers makes locating materials, assistance, and support for ROS development simpler [24].

3.1.2.3 Python

Figure 3.1.2 Python

Python [25] is the primary programming language used in this project due to its simplicity and extensive ecosystem of libraries. It is used for writing ROS2 nodes with rclpy. It is also used for developing and training the emotion recognition model using together with other libraries like TensorFlow and Keras.

3.1.2.4 Jupyter Notebook

Figure 3.1.3 Jupyter Notebook

Jupyter Notebook [26] is an open-source web-based application that used to write and execute code in a step-by-step manner which making developer easy to test and debug your code interactively. In this project, it serves as an IDE to develop emotion recognition model.

3.1.2.5 Visual Studio Code

Figure 3.1.4 Visual Studio Code

Visual Studio Code [27] is a lightweight but powerful source code editor developed by Microsoft. In this project, it is used for developing, managing, and debugging ROS 2 nodes and workspaces.

3.1.2.6 OpenCV

Figure 3.1.5 OpenCV

OpenCV [28] is a widely used library for computer vision and image processing tasks. In this project, it is used for real-time image processing, face detection and face cropping before emotion classification.

3.1.2.7 OpenAI Assistant

In this project, OpenAI's GPT 3.5 Turbo is integrated via the Assistants API which allow the robot to maintain conversational memory, emotional awareness and contextual relevance. The assistant can adapts its tone based on the user's detected emotion to enable emotionally intelligent interactions. OpenAI's GPT is selected as the best option for integrating into Turtlebot3 because of its exceptional ability to understand and generate human-like language which is important for our project to provide contextual aware and emotionally intelligent responses.

3.1.2.8 Whisper Speech-To-Text Engine

Figure 3.1.6 OpenAI Whisper

Whisper is an automatic speech recognition system developed by OpenAI. It was used to transcribe user audio into text form. Whisper is known for its robustness across various accents, background noise conditions, and speech patterns. So, it well-suited for real-world conversational scenarios like those handled by the Companion Robot. In this project, its API been utilized to perform the transcription.

3.1.2.9 Google Text-To-Speech (gTTS)

Figure 3.1.7 Google Text-To-Speech

gTTS was used to convert text-based responses into speech. This ensures verbal communication from the robot is clear and natural. The gTTS library is used in this project to generate MP3 audio from text strings.

3.1.3 Dataset

Figure 3.1.8 FER-2013 Dataset

In this project, the dataset used for this emotion recognition model is FER-2013. This dataset contains grayscale images with 48x48 pixels [29]. It has covers 7 emotion classes including angry, disgust, fear, happy, sad, surprise and neutral [29]. This FER-2013 is one of the famous datasets that widely used in emotion recognition research. Besides, this dataset is freely accessible by anyone which allow researchers and developers to use it without worrying about licensing issues.

3.2 System Development

The system development process is modular and follows a structured node-based approach using ROS2. Two main functional modules form the core of the Companion Robot system are the **Emotion Detection Module** and the **Conversational Response Module**. These are implemented independently as ROS2 nodes to ensure modularity and ease of testing. A brief overview is provided below, implementation details will be presented in the following chapters.

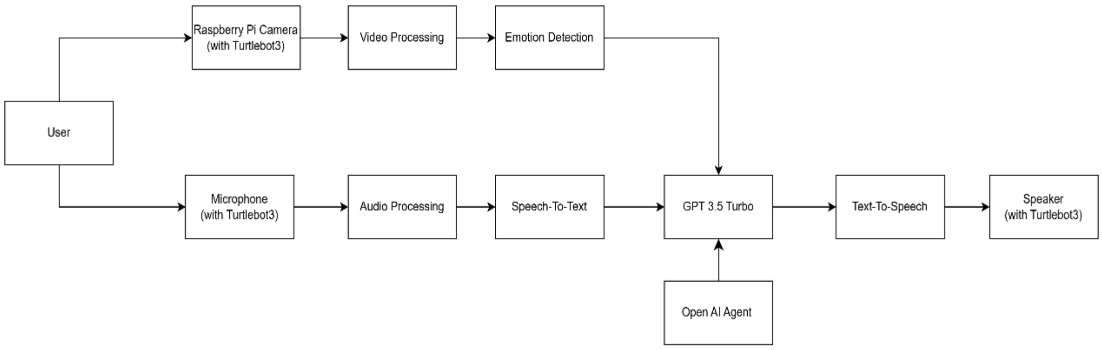


Figure 3.1.9 System Architecture

- Emotion detection module. This module utilizes a trained CNN model to detect user emotions based on facial expressions. Images are captured in real time from the Raspberry Pi camera mounted on the TurtleBot3. The detected emotion is then published via a ROS2 topic to be used by other components.

- Conversational response module. This module handles audio-based interaction with the user. It captures speech using a microphone connected to TurtleBot3 and transmits the audio stream for transcription. The Whisper API is used to transcribe the audio to text. The transcribed text together with the detected emotion is passed to an OpenAI GPT Agent to generate a context-aware and emotionally relevant response. The reply is then converted into speech using the gTTS engine and played through the TurtleBot's speaker.

3.2.1 Integration of ROS2

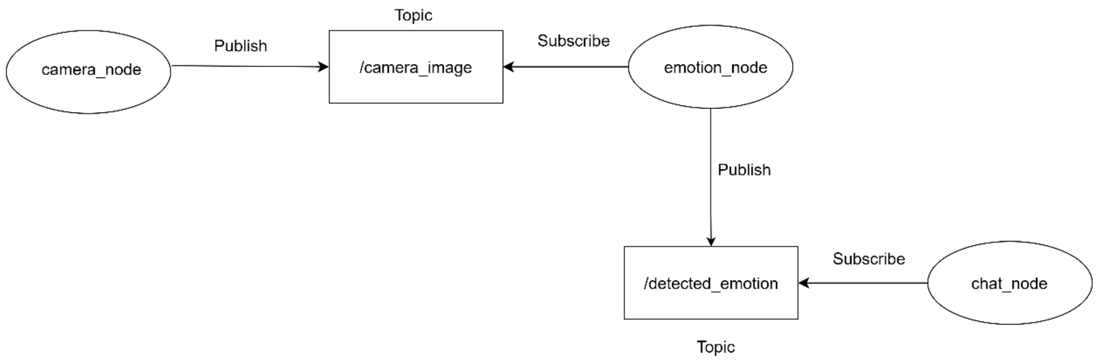


Figure 3.1.10 ROS2 Communication

The companion robot leverages the ROS2 framework to achieve a modular, scalable, and distributed architecture. This ROS2 usually used in the robotic application because of its inter-node message passing and it support various for various hardware platforms. Not to mention, the integration of ROS2 allows these modules to function asynchronously to create responsive user interaction experience.

Two main functional modules **Emotion Detection Module** and the **Conversational Response Module** will implemented as independent ROS2 node. These ROS2 node communicates through a publish-subscribe model. In this architecture, the publishers send data to named topics. Subscribers listen to those topics and process the incoming data. In this system, it comprises several nodes and communication topics.

- Camera node: It used to capture real-time video frames using raspberry pi camera and publishes them to the /camera_image topic.

- Emotion_node: It used to subscribe to /camera_image topic, process the image using CNN model to detect emotion and publishes the result to the /detection_emotion topic.

- Chat_node : It used to subscribe to /detected_emotion topic and simultaneously listens for audio input from the user. It transcribes the audio using the Whisper API and sends both the text and emotion to the GPT agent. The response is converted into speech using gTTS and played via the robot’s speaker.

3.3 Verification Plans

Verification plan is needed to ensure the reliability, correctness and functionality of the companion robot system. This plan will outlines the methods and steps used to verify each component of the system to ensure that all the modules operate as intended. Verification will focus on the emotion detection and conversational response and their integration within the ROS2.

3.3.1 Emotion Detection Module Testing

The purpose of this testing is to verify the CNN model correctly identifies facial emotions from image inputs. This verification is crucial to ensure the model behaves as expected in realistic use cases where human expressions vary due to different emotional states. To conduct this testing, a script that utilize OpenCV, TensorFlow/Keras and a CNN model is used. The script accesses the webcam on the local device to continuously capture video frames. It detects human faces in each frame using the Haar Cascade classifier and then feeds the face regions into the trained CNN model for emotion prediction. The model's output is compared to known expressions to validate its performance under various emotional conditions such as happiness, sadness, anger, fear, and more.

3.3.2 Conversational Response Module Testing

The purpose of this testing is to ensure that the system accurately transcribes audio, generates relevant text responses and plays them as speech. First, several prompts need to be prepare for this entire module testing. Then, the Whisper API is used for testing the transcription using the same prompts to ensure the system can reliably convert audio to text. This is followed by verification of the GPT agent which is tasked with generating emotionally aware responses. The transcribed text from last testing will be test under various emotional contexts to evaluate the response. Finally, the gTTS engine is evaluated by converting these replies into audio and ensuring that the speech output is clear and accurate. These steps confirm that each part of the module performs its role before integration.

No	Test Case Description	Test Data	Expected Result
1.	Verify that the LLM processes detected emotion to generate an appropriate response.	Detected Emotion: "happy"	The LLM can start conversation with user after detected emotion.
2.	Verify that the robot converts the LLM response into speech using gTTS and speaker.	-	Response can convert to audible speech and play it via speaker.
3.	Verify that the system accurately converts user speech to text using the microphone and Whisper service.	User speaks a predefined sentence: "I just got a job offer today."	The robot captures the speech input via the microphone and able transcribe it into correct text.
4.	Verify that the LLM processes transcribed text to generate an appropriate response.	Using same prompt “I just got a job offer today.”	The LLM generates a relevant contextual awareness response.
5.	Verify that the system can handle situations where speech fails.	User speaks very softly.	The robot should notify unclear speech and return response like, e.g., "Sorry, I didn’t catch that. Could you repeat?"

Table 3.1.2 Verification Plan for Conversational Response Module

3.3.3 Usability Testing

The purpose of this testing is to evaluate the effectiveness and how naturally the users interact with the companion robot system. Besides, this testing focuses on the quality of the user experience particularly in terms of emotional relevance, satisfaction and perceived empathy. This phase of testing is designed to gather feedback from 30 to 50 real users by engaging with robot in realistic scenario to determine whether the robot's conversational responses are not only technically correct but also emotionally aligned with user expectations.

3.4 Project Milestone

3.4.1 Project I Timeline

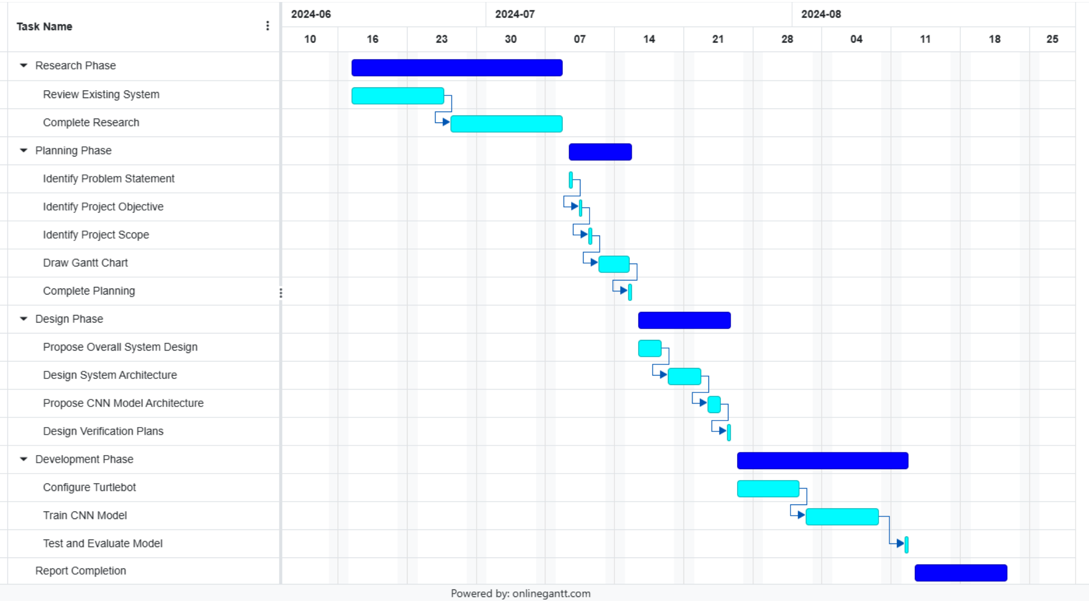


Figure 3.1.11 Project I Timeline

The FYP 1 is structured into four key phases which are Research, Planning, Design, and Development. The project begins with the Research Phase. During this phase, the focus is on reviewing existing systems and technologies relevant to the companion robot. A comparative analysis is conducted to understand current solutions in emotion recognition and human-robot interaction which helps identify gaps and establish the basis for innovation. This is followed by the Planning Phase. In this stage, the problem statement is defined, project objectives and scope are clearly outlined and the Gantt chart is drawn to visualize the timeline. These steps ensure the project has a clear direction and well-defined deliverables. This phase ends with a completion checkpoint to verify that the planning elements are aligned with project goals.

Following that, the Design Phase. During this period, the overall system design is proposed which includes breaking down the solution into modular components such as the emotion detection and conversational response systems. The system architecture is mapped out using ROS2 highlighting how nodes interact via topics. The design of the CNN model architecture is also proposed to address facial emotion classification. An important task in this phase is the formulation of verification plans where detailed testing strategies are laid out to ensure the system functions correctly in both isolated and integrated scenarios.

The final part of FYP 1 is the Development Phase where the practical aspects of implementation are initiated. The TurtleBot3 is configured and integrated with a Raspberry Pi to handle image input. A CNN is trained using facial emotion dataset followed by a phase of testing and evaluation to ensure model accuracy and performance. The last week is dedicated to Report Completion where all findings, implementations, and evaluations from FYP 1 are compiled into a formal report for submission.

3.4.2 Project II Timeline

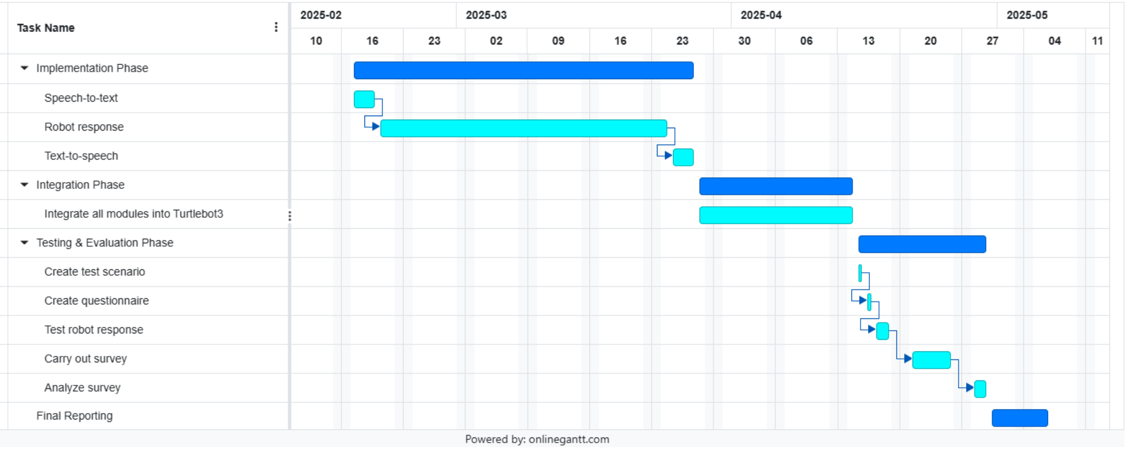


Figure 3.1.12 Project II Timeline

The FYP2 timeline is structured into four main phases which are Implementation Phase, Integration Phase, Testing & Evaluation Phase and Final Reporting. The Implementation Phase focuses on building and validating the system's core functionalities. It starts with the Speech-to-Text task where the system captures user speech and transcribes it into text using the Whisper API. Followed by the Robot Response component is developed to generate emotionally relevant replies based on both transcribed text and detected emotions using the GPT-based agent. Subsequently, the Text-to-Speech task involves converting these responses into audio output via the gTTS engine and playing them through the TurtleBot's speaker.

Then, the Integration Phase takes place. This phase involves integrating all the individual modules into the TurtleBot3 platform. The integration ensures smooth communication between all nodes such as camera feed, emotion detection, speech input, text processing, and audio playback using ROS2. During this time, all subsystems are tested together to ensure they operate cohesively within the robot's environment.

The Testing & Evaluation Phase starts immediately after integration. It begins with designing a test scenario that reflects real user interaction situations. Then, a questionnaire is created to gather feedback from users regarding the system's performance and interaction quality. The next step involves testing the robot's responses in realistic conditions followed by a user survey to collect subjective evaluations. The collected data is then analysed to identify strengths, limitations, and potential areas for future improvements.

Finally, the Final Reporting phase. During this period, the results from implementation, integration, and testing phases are compiled into a comprehensive final report. This report will include technical documentation, user feedback, system limitations, and reflections on the project development process.

3.5 Chapter Summary

This chapter has detailed the methodology used in developing the companion robot which emphasizing an iterative and modular approach inspired by Agile principles. It began with a comprehensive breakdown of both hardware and software requirements that covering key components such as TurtleBot3, Raspberry Pi camera, microphone, speaker and critical software tools like ROS2, OpenCV, and the OpenAI Assistant API.

The system development was structured around two core modules such as Emotion Detection and Conversational Response. Each module implemented as ROS2 nodes to promote modularity and asynchronous communication. The architecture was further supported by the ROS2 publish-subscribe model for efficient inter-node communication.

A structured verification plan was then outlined which focusing on component-level testing of the CNN-based emotion detection, Whisper-based speech transcription, GPT agent for context-aware replies, and the gTTS engine for voice output. A usability testing plan was also proposed to evaluate user satisfaction and emotional engagement with the robot in realistic settings.

Finally, the chapter presented a two-phase project timeline. FYP 1 focused on research, design, and initial development while FYP 2 emphasized full implementation, system integration, and final evaluation. Each phase was described with clear deliverables and outcomes to guide the systematic development of the companion robot system.

CHAPTER 4

System Design

4.1 Use Case Diagram

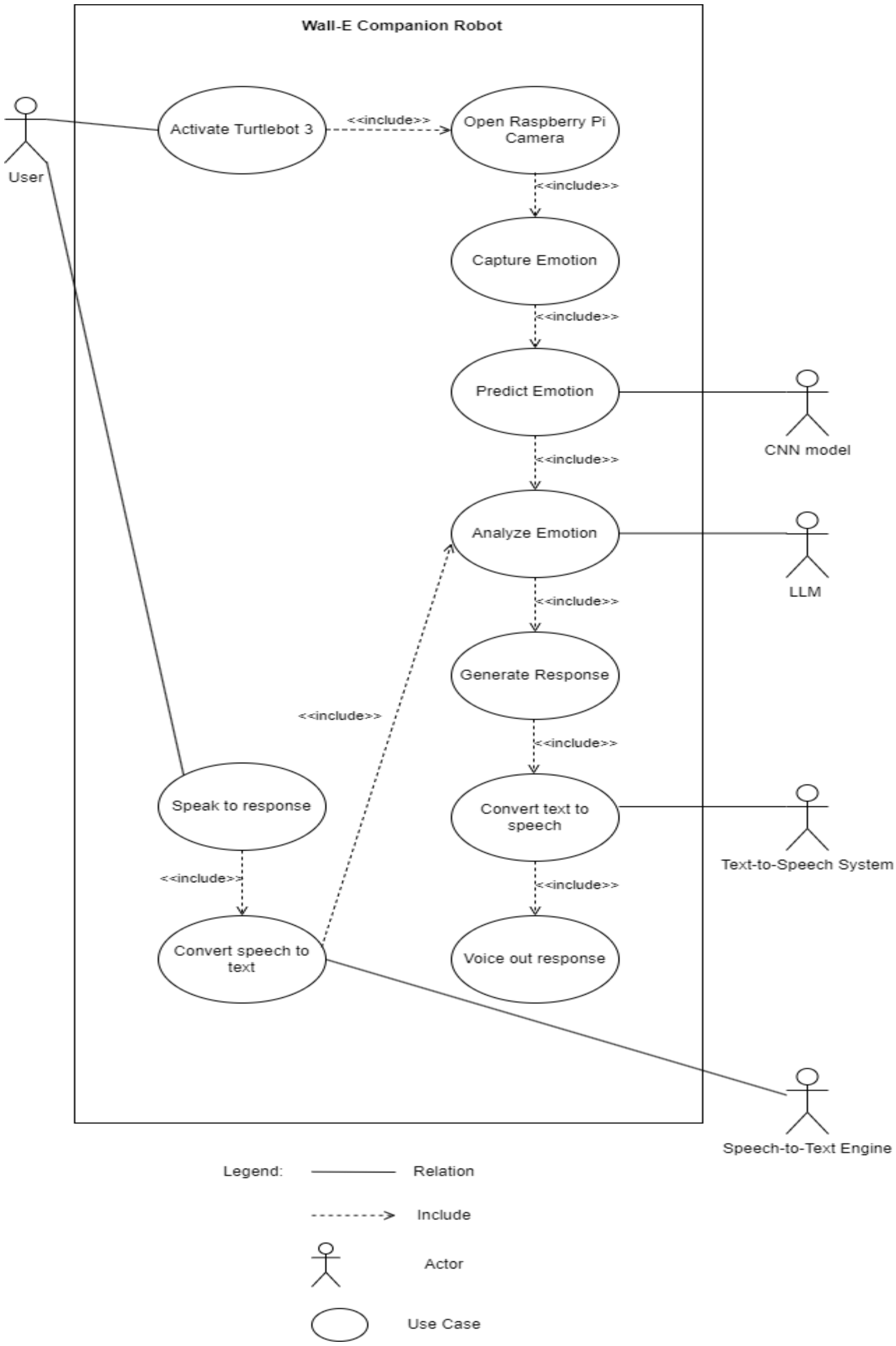


Figure 4.1.1 Use Case Diagram

Actors	Preconditions	Normal Flow	Alternative Flow
User	-	1. User activate companion robot. 2. User obtain response from robot once emotion detected.	-
CNN model	User’s emotion must be captured.	CNN model predicts the user emotion.	-
LLM	User’s emotion must be detected using CNN.	LLM analyse user’s emotion and start conversation by generating response.	-
Text-to-Speech System	LLM’s response must be generated.	1. Text-to-Speech System convert text response from LLM to speech. 2. Voice out the speech.	-
Speech-to-Text System	User must speak to make response.	1. Speech-to-Text System convert speech from user to text. 2. These text will reached LLM side as input and the process repeat again to ask LLM analyse input and generate response.	-

Table 4.1.1 Use Case Description

4.2 Activity Diagram

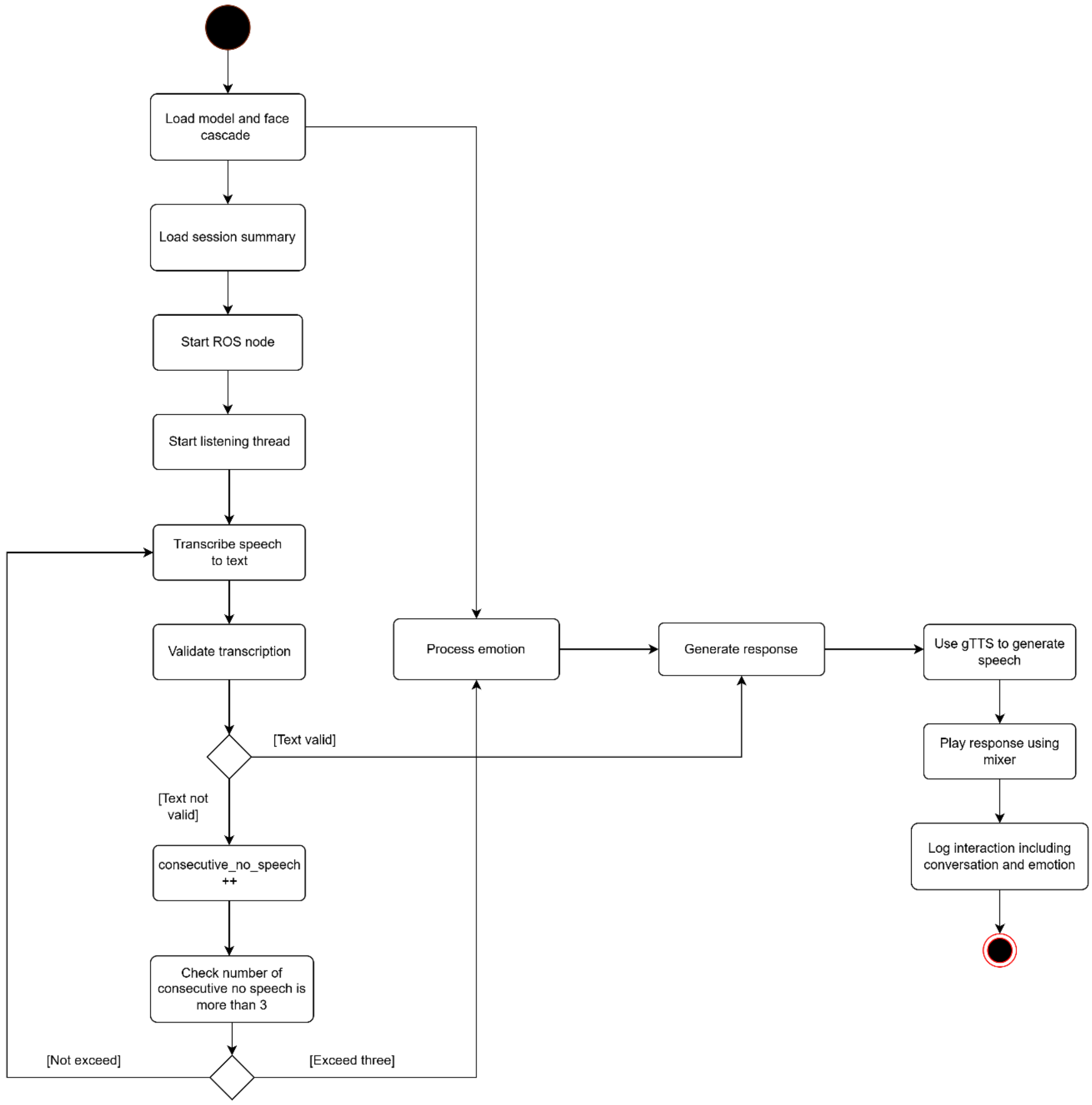


Figure 4.1.2 Activity Diagram

Activity diagram above illustrates the operation flow of a companion robot designed to interact with users by understanding their speech and emotions and responding accordingly. The system begins by loading the facial emotion recognition model and face detection cascade which are necessary for analysing user emotions. It then loads the session summary to allow the robot to retain context from previous interactions. Next, the system initializes the ROS node to enable it to communicate within the robot's distributed software environment. A listening thread is then launched to continuously monitor user speech. Once initialized, the robot transcribes user speech. The transcribed text is then validated to ensure it is meaningful. If valid, it proceeds to process the user's emotion based on audio or visual cues and then generates an appropriate response using an LLM.

The response is converted to speech using gTTS and then played back using a sound system. After playback, the system logs the entire interaction including the transcription, emotion, and response to maintain conversational continuity. If the transcription is invalid, a counter is incremented. Once the counter exceeds three consecutive failed speech attempts, the system evaluates to enter a sleep mode ensuring the robot doesn't continue unnecessary processing during inactivity. However, once emotion has changed, robot will initiate new conversation again with the user.

4.3 Chapter Summary

This chapter presented the core functional flow and interaction design of the companion robot system through a use case diagram and an activity diagram. The system is designed to facilitate seamless human-robot interaction by recognizing user emotions, analysing conversations, and responding accordingly.

The use case diagram outlines the roles of user, CNN-based emotion recognition model, large language model (LLM), speech-to-text, and text-to-speech systems. Each actor plays a distinct role in ensuring real-time emotional and conversational feedback. The user initiates interaction which triggers a chain of processes. For instance, emotion detection by the CNN model, conversational response generation by the LLM and voice synthesis via a text-to-speech engine. Meanwhile, a speech-to-text system continuously processes user input for ongoing interaction.

The activity diagram complements the use case by illustrating the system's operational workflow. It begins with model loading, ROS node initialization, and session summary retrieval for maintaining conversational context. Once active, the system listens to the user, transcribes speech to text, validates it, and uses both textual and emotional inputs to generate intelligent responses. Responses are vocalized and logged for continuity. The robot also features a self-monitoring mechanism to handle inactivity by entering sleep mode after repeated silence, reactivating upon detecting a change in user emotion. Together, the diagrams provide a comprehensive view of how the companion robot engages users empathetically through emotional awareness and conversational intelligence.

CHAPTER 5

System Implementation

5.1 Software / Hardware Setup and Configuration

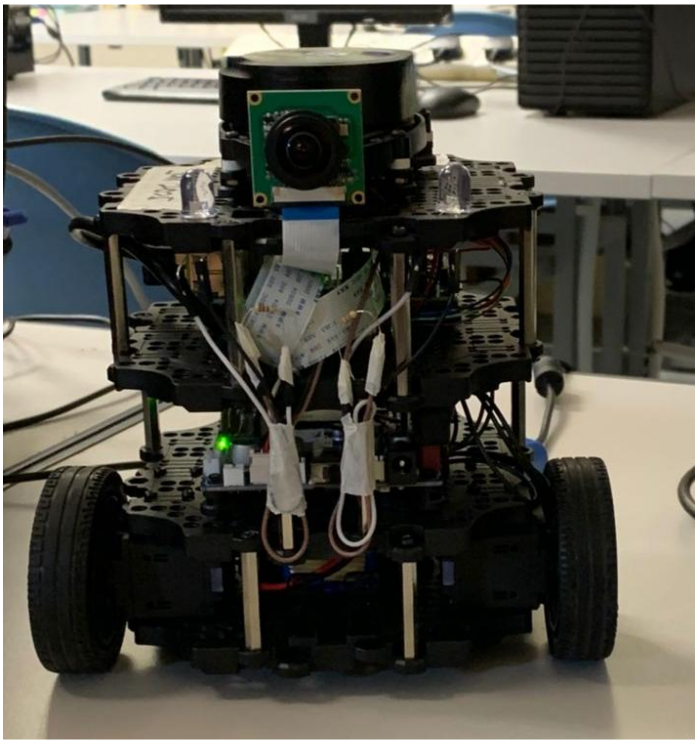


Figure 5.1.1 Final Assembled Turtlebot3

This chapter outlines the step-by-step implementation process of the TurtleBot3 system including the configuration of the Remote PC, Single Board Computer (SBC), and OpenCR board, culminating in the complete bring-up of the robot. The implementation ensures seamless communication between hardware and software components through the ROS 2 Humble framework to enabling autonomous robotic behaviour. Figure 5.1.1 shows the final assembled TurtleBot3 that equipped with sensors and a mounted camera module which ready for deployment after successful setup and configuration across all system components.

5.1.1 PC Setup

The first step in setting up the TurtleBot3 system involves configuring the Remote PC with the necessary operating system, software frameworks, and packages. The setup begin by:

1. Install Ubuntu 22.04 operating system to provide a stable and compatible environment for ROS2 development.
2. Install ROS2 Humble which is the middleware framework that enables communication between the TurtleBot3 and the PC.
3. Create new workspace to set up TurtleBot3 packages. Then, clone the relevant repositories and building them using colcon. These packages include essential drivers, messages, and launch files required for TurtleBot3 operation.
4. Configure the environment by updating the bash configuration file to automatically source necessary setup scripts on each new terminal session. This ensures that the ROS environment is properly initialized and the system communicates over the designated ROS domain for TurtleBot3.

5.1.2 SBC Setup

The second step in setting up the TurtleBot involves configuring the Single Board Computer (SBC) which is typically mounted on the robot base and acts as its primary processing unit. The setup begin by:

1. Ensure the SBC has a compatible version of the Ubuntu operating system installed
2. Set up the ROS 2 Humble distribution into the raspberry pi microSD card to enable communication between hardware components and software nodes.
3. Establish network connectivity either through Wi-Fi or Ethernet to allow the SBC to communicate with other devices such as remote PCs.
4. Install essential ROS 2 packages specific to the TurtleBot including drivers and launch files to enable sensor data acquisition, motor control, and basic autonomous behavior.

5. Configure SBC to automatically launch required ROS 2 nodes during startup to ensure the robot becomes operational as soon as it is powered on.

5.1.3 OpenCR Setup

The third step in setting up the TurtleBot involves configuring the OpenCR board which serves as the interface between the robot's sensors, actuators, and the SBC. The setup begin by:

1. Connect OpenCR board to the Raspberry Pi using a micro-USB cable.
2. Install additional packages onto the raspberry pi to support the ARM architecture to enable communication.
3. Upload the firmware (Burger) is uploaded to the OpenCR board to ensure it is compatible with the ROS 2 environment.
4. Test basic hardware functionality using built-in buttons on the OpenCR. These tests verify whether the robot's motors respond correctly by moving forward and rotating in place.

5.1.4 Bring up Turtlebot3

The final step in setting up the TurtleBot3 involves bringing up the robot by launching its essential software components.

1. Open a new terminal on your remote PC and establish an SSH connection to the Raspberry Pi using its IP address.
2. Ensure that the correct TurtleBot3 model is specified before launching the robot's core bringup package. This process initializes various nodes responsible for publishing the robot’s state, sensor data and controlling the motors. Successful execution will display system logs confirming the initialization of components such as the robot state publisher, laser scanner, and motor controllers.
3. Once set up, you can list active topics and services using command to verify that the TurtleBot3 has been correctly launched and is ready for further interaction or development.

5.2 Implementation Details

5.2.1 Emotion Detection Model

5.2.1.1 Data Preprocessing

```
# Convert to NumPy arrays
X = np.array(img_data, dtype='float32') / 255.0 # Normalize to [0,1]
y = np.array(labels, dtype='int64')
```

```
# Convert labels to categorical (One hot encoding)
y = to_categorical(y, 7)
```

```
# Train-test split
X_train, X_valid, y_train, y_valid = train_test_split(X ,y , shuffle=True, stratify=y, test_size=0.2, random_state=42)

X_train.shape, X_valid.shape, y_train.shape, y_valid.shape
```

```
((18093, 48, 48, 3), (4524, 48, 48, 3), (18093, 7), (4524, 7))
```

```
# Data Augmentation (only on training data)
aug = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
aug.fit(X_train)
```

Figure 5.2.1 Data Preprocessing

The first step in implementing the emotion detection model involves preprocessing the image data to ensure it is suitable for training a neural network. Initially, all images were converted into NumPy arrays and normalized to the range [0, 1] by dividing pixel values by 255. This normalization process helps the model converge faster during training. The emotion labels represent seven distinct classes which are Angry, Disgust, Fear, Happy, Neutral, Sad, and Surprise. These classes were encoded into one-hot vectors using categorical encoding. This format is required for multiclass classification tasks in deep learning.

Next, the dataset was split into training and validation sets using an 80:20 ratio. The train_test_split function was employed with shuffling and stratified sampling to ensure balanced distribution of emotion classes across both subsets. To further enhance the model's generalization ability, data augmentation techniques were applied exclusively to the training data. This was achieved using the ImageDataGenerator class from Keras. It performed random transformations such as rotations, shifts, zooming, shearing, and horizontal flipping. These augmentations introduce variability in the training data to reduce overfitting.

5.2.1.2 Model Training

```python
def create_model():
    model = Sequential([
        Input(shape=(48, 48, 3)),

        Conv2D(64, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(128, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(256, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(512, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),

        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(7, activation='softmax')
    ])
    model.compile(loss='categorical_crossentropy', optimizer=Adam(0.001), metrics=['accuracy'])
    return model
```

```python
# Callbacks
early_stopping = EarlyStopping(monitor='val_accuracy', patience=10, restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_accuracy', factor=0.5, patience=5, min_lr=1e-7)
callbacks = [early_stopping, lr_scheduler]
```

```python
model = create_model()
history = model.fit(aug.flow(X_train, y_train, batch_size=64),
          validation_data=(X_valid, y_valid),
          epochs=100,
          callbacks=callbacks,
          verbose=2)
```

```
Epoch 67/100
283/283 - 78s - 274ms/step - accuracy: 0.6537 - loss: 0.9218 - val_accuracy: 0.6326 - val_loss: 0.9914 - learning_rate: 6.2500e-05
Epoch 68/100
283/283 - 78s - 274ms/step - accuracy: 0.6572 - loss: 0.9195 - val_accuracy: 0.6417 - val_loss: 0.9755 - learning_rate: 3.1250e-05
Epoch 69/100
283/283 - 77s - 273ms/step - accuracy: 0.6566 - loss: 0.9152 - val_accuracy: 0.6466 - val_loss: 0.9831 - learning_rate: 3.1250e-05
Epoch 70/100
283/283 - 78s - 274ms/step - accuracy: 0.6561 - loss: 0.9174 - val_accuracy: 0.6410 - val_loss: 0.9768 - learning_rate: 3.1250e-05
Epoch 71/100
283/283 - 77s - 273ms/step - accuracy: 0.6595 - loss: 0.9069 - val_accuracy: 0.6390 - val_loss: 0.9845 - learning_rate: 3.1250e-05
Epoch 72/100
283/283 - 77s - 273ms/step - accuracy: 0.6567 - loss: 0.9061 - val_accuracy: 0.6364 - val_loss: 0.9873 - learning_rate: 3.1250e-05
Epoch 73/100
283/283 - 77s - 273ms/step - accuracy: 0.6593 - loss: 0.9043 - val_accuracy: 0.6426 - val_loss: 0.9702 - learning_rate: 3.1250e-05
Epoch 74/100
283/283 - 77s - 273ms/step - accuracy: 0.6594 - loss: 0.9039 - val_accuracy: 0.6419 - val_loss: 0.9712 - learning_rate: 3.1250e-05
Epoch 75/100
283/283 - 78s - 274ms/step - accuracy: 0.6619 - loss: 0.8999 - val_accuracy: 0.6410 - val_loss: 0.9717 - learning_rate: 1.5625e-05
```

Figure 5.2.2 Model Training

The emotion detection model was implemented using a CNN architecture built with Keras. The model accepts grayscale facial images of size 48x48 pixels as input. It comprises three convolutional blocks where each consisting of a 2D convolutional layer followed by batch normalization and max pooling. The number of filters progressively increases from 64 to 256 across these block to allow the network to learn increasingly complex features. Then, the output is flattened and passed through a fully connected layer with 512 neurons followed by a dropout layer with a 50% dropout rate to reduce overfitting. The final output layer contains 7 units with a softmax activation function for multiclass classification.

The model is compiled using the categorical cross entropy loss function which is suitable for multiclass classification problems. The Adam optimizer with an initial learning rate of 0.001 was used for training. Model performance was evaluated using the accuracy metric. Two callbacks were employed during training to improve generalization and prevent overfitting. One of it is EarlyStopping. It used to monitor validation accuracy and stops training if no improvement is observed for 10 consecutive epochs. It will also restore the best model weights. Another callback was ReduceLROnPlateau. It reduces the learning rate by a factor of 0.5 if validation accuracy plateaus for 5 epochs with a minimum learning rate threshold of 1e-7. The model was trained for up to 100 epochs using the augmented training dataset with a batch size of 64. Validation was performed on a separate hold-out validation set. The use of callbacks ensured that training was efficient and adaptive to model performance across epochs.

```
# Evaluate model
predictions = model.predict(X_valid)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = np.argmax(y_valid, axis=1)
```

```
142/142 ━━━━━━━━━━━━━━━━━━━━ 4s 24ms/step
```

```
# Confusion matrix
sorted_labels = sorted(label_map, key=label_map.get)
cm = pd.DataFrame(confusion_matrix(true_classes, predicted_classes),
                  index=sorted_labels, columns=sorted_labels)
print("Confusion Matrix:\n", cm)
```

```
Confusion Matrix:
           Angry  Disgust  Fear  Happy  Neutral  Sad  Surprise
Angry       357       13    63     11      115   59        29
Disgust      43      562    13      1       11   16         0
Fear         95        8   265     17       65  116        80
Happy        39        4    18    479       60   23        24
Neutral      41        4    26     37      435   84        19
Sad          97        4    66     20      153  299         7
Surprise     26        4    72     17       19   10       498
```

```
print(classification_report(true_classes, predicted_classes, target_names=sorted_labels))
```

```
              precision    recall  f1-score   support

       Angry       0.51      0.55      0.53       647
     Disgust       0.94      0.87      0.90       646
        Fear       0.51      0.41      0.45       646
       Happy       0.82      0.74      0.78       647
     Neutral       0.51      0.67      0.58       646
         Sad       0.49      0.46      0.48       646
    Surprise       0.76      0.77      0.76       646

    accuracy                           0.64      4524
   macro avg       0.65      0.64      0.64      4524
weighted avg       0.65      0.64      0.64      4524
```

Figure 5.2.3 Model Evaluation

The performance of the facial emotion recognition model was assessed using a confusion matrix and various classification metrics including precision, recall, and F1-score. Based on the evaluation results, the model achieved an overall accuracy of 65% with a macro-averaged F1-score of 0.64. The confusion matrix revealed that the model performed well in recognizing emotions such as "Disgust", "Happy", and "Surprise" where high true positive counts were observed. "Surprise" class attained the highest F1-score of 0.76, followed closely by "Disgust" with an F1-score of 0.87. On the other hand, the model struggled with emotions like "Fear" and "Sad" which had lower F1-scores of 0.45 and 0.30 respectively. This indicated the difficulties in correctly distinguishing these expressions. Misclassifications were common between emotionally similar or ambiguous classes such as confusing "Fear" with "Sad". These results suggest that while the model is effective in identifying more distinct facial expressions.

5.2.1.3 Hyperparameter Tuning

```python
from sklearn.model_selection import ParameterGrid
from tensorflow.keras.optimizers import Adam, SGD

def evaluate_model(X_train, y_train, X_valid, y_valid, learning_rate, optimizer, batch_size):
    optimizerD= {'Adam': Adam(learning_rate=learning_rate),
                 'SGD': SGD(learning_rate=learning_rate)}
    optimizerL = optimizerD[optimizer]

    # Train model
    model = create_model()

    model.compile(optimizer = optimizerL, loss = 'categorical_crossentropy', metrics = ['accuracy'])

    model.fit(aug.flow(X_train, y_train, batch_size=batch_size),
          validation_data=(X_valid, y_valid),
          epochs=100,
          callbacks=callbacks,
          verbose=2)

    # Evaluate model
    loss, accuracy = model.evaluate(X_valid, y_valid, verbose=2)
    return accuracy
```

```python
parameters = {'learning_rate': [0.01, 0.05, 0.001, 0.005, 0.0001, 0.0005],
              'optimizer': ['Adam', 'SGD'],
              'batch_size':[32, 64, 128],
             }
```

```python
cv_results_df = pd.DataFrame(columns=['batch_size', 'learning_rate', 'optimizer', 'score'])  # outside the Loop

for i in list(ParameterGrid(parameters)):
    batch_size = i.get('batch_size')
    learning_rate = i.get('learning_rate')
    optimizer = i.get('optimizer')

    score = evaluate_model(X_train, y_train, X_valid, y_valid, learning_rate, optimizer, batch_size)

    series_values = pd.Series({
        'batch_size': batch_size,
        'learning_rate': learning_rate,
        'optimizer': optimizer,
        'score': score
    })

    cv_results_df = pd.concat([cv_results_df, series_values.to_frame().T], ignore_index=True)
    print(cv_results_df)
```

Figure 5.2.4 Hyperparameter Tuning

To optimize the performance of the facial emotion recognition model, a grid search was conducted over a range of hyperparameters including learning rate, optimizer type, and batch size. The tuning process explored combinations of six different learning rates, two optimizers “Adam” and “SGD”, and three batch sizes. This has resulting in a total of 36 parameter combinations. For each configuration, a model was compiled with the specified optimizer and learning rate and trained using the given batch size and evaluated on a validation dataset to determine its accuracy. The evaluate_model function returned the accuracy score for each combination. This process enabled a systematic evaluation of how different hyperparameter values influenced model performance with aim to find the maximize accuracy on the validation data.

batch_size	learning_rate	optimizer	score
32	0.1	Adam	0.1430
32	0.1	SGD	0.6059
32	0.01	Adam	0.1439
32	0.01	SGD	0.5756
32	0.001	Adam	0.6625
32	0.001	SGD	0.4500
32	0.0001	Adam	0.5926
32	0.0001	SGD	0.3048
64	0.1	Adam	0.1430
64	0.1	SGD	0.6331
64	0.01	Adam	0.1753
64	0.01	SGD	0.5420
64	**0.001**	**Adam**	**0.6607**
64	0.001	SGD	0.4100
64	0.0001	Adam	0.5714
64	0.0001	SGD	0.2708
128	0.1	Adam	0.1430
128	0.1	SGD	0.5855
128	0.01	Adam	0.1863
128	0.01	SGD	0.5186
128	0.001	Adam	0.6426
128	0.001	SGD	0.3610
128	0.0001	Adam	0.5595
128	0.0001	SGD	0.2675

Table 5.2.1 Result of Hyperparameter Tuning

Table above shown the results after performed hyperparameter tuning across different combinations of learning rates, optimizers, and batch sizes. Overall, the Adam optimizer consistently underperformed at higher learning rates. However, Adam delivered the best performance at lower learning rates with an accuracy of 66.25% using a batch size of 32 followed by 66.07% with a batch size of 64. In contrast, the SGD optimizer performed relatively better at higher learning rates achieving its highest accuracy of 63.31% with a learning rate of 0.1 and batch size of 64. As learning rates decreased, SGD's performance declined. It indicated its sensitivity to small learning rates. Among all tested configurations, the combination of the Adam optimizer with a learning rate of 0.001 and a batch size of 64 yielded the highest validation accuracy. After that, the model was retrained using the best hyperparameters combinations to achieve a better accuracy.

5.2.2 Emotion Detection Module

5.2.2.1 Camera Node

camera_node.py

G: > turtlebot_ws (full) > src > emotion_pkg > emotion_pkg > camera_node.py > CameraNode > __init__

```python
import rclpy
import cv2
import os
import time
from rclpy.node import Node
from sensor_msgs.msg import Image
from std_msgs.msg import Int32
from cv_bridge import CvBridge

class CameraNode(Node):
    def __init__(self):
        super().__init__('camera_node')
        self.publisher_ = self.create_publisher(Image, '/camera_image', 10)
        self.image_id_publisher = self.create_publisher(Int32, '/image_id', 10)
        self.bridge = CvBridge()
        self.image_id = 0

        #Open the camera, capture and release immediately
        self.cap = cv2.VideoCapture(0, cv2.CAP_V4L2)
        self.cap.set(cv2.CAP_PROP_FPS, 10)  # Set FPS
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
        self.cap.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'YUYV'))

        if not self.cap.isOpened():
            self.get_logger().error("Unable to open camera")
            return

        # Publish frames continuously every 0.1s (10 FPS)
        self.timer = self.create_timer(0.1, self.publish_frame)

    def publish_frame(self):

        ret, frame = self.cap.read()
        if not ret:
            self.get_logger().error("Failed to capture image")
            return

        # Publish image
        image_msg = self.bridge.cv2_to_imgmsg(frame, encoding='bgr8')
        image_msg.header.stamp = self.get_clock().now().to_msg()
        image_msg.header.frame_id = str(self.image_id)

        # Publish image and image ID
        self.publisher_.publish(image_msg)
        image_id_msg = Int32()
        image_id_msg.data = self.image_id
        self.image_id_publisher.publish(image_id_msg)
        self.image_id += 1  # Increment image ID

    def destroy_node(self):
        self.cap.release()
        super().destroy_node()

def main(args=None):
    rclpy.init(args=args)
    camera_node = CameraNode()
    rclpy.spin(camera_node)
    camera_node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Figure 5.2.5 Implementation of Camera Node

This camera_node is designed to interface with a camera using OpenCV to capture real-time video frames and publish them to a ROS 2 topic for further processing. The node utilizes the cv_bridge package to convert OpenCV images into ROS 2 sensor_msgs/Image messages and publishes them on the /camera_image topic at a rate of 10 frames per second. Additionally, it publishes an incremental image ID on a separate /image_id topic using the std_msgs/Int32 message type to help uniquely identify each captured frame.

Upon initialization, the node sets up the camera using the V4L2 backend and configures frame size and encoding and checks if the camera is successfully opened. If the camera is accessible, a timer is created to trigger the publish_frame method every 0.1 seconds. In each call, the node captures a frame from the camera, converts it to a ROS image message, stamps it with the current time, assigns the current image ID as the frame ID, and publishes both the image and image ID. The image_id is incremented with each frame to ensure uniqueness. Upon termination, the camera resource is properly released.

5.2.2.2 Emotion Node

```python
#!/usr/bin/env python3

import os
import cv2
import rclpy
import numpy as np
from rclpy.node import Node
from sensor_msgs.msg import Image
from std_msgs.msg import String, Int32
from cv_bridge import CvBridge, CvBridgeError
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array
from rclpy.qos import QoSProfile

class EmotionNode(Node):

    def __init__(self):
        super().__init__("emotion_node")

        qos_profile = QoSProfile(depth=10)

        self.emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']

        # Load emotion detection model
        self.model = load_model('/home/ubuntu/turtlebot_ws/src/emotion_pkg/emotion_pkg/rgb_model_0.66.keras')

        self.face_cascade = cv2.CascadeClassifier('/home/ubuntu/turtlebot_ws/src/emotion_pkg/emotion_pkg/haarcascade_frontalface_default.xml')

        self.bridge = CvBridge()

        # Publisher for detected emotions
        self.emotion_publisher = self.create_publisher(String, '/detected_emotion', qos_profile)

        # Subscribe to camera images
        self.image_subscription = self.create_subscription(Image, '/camera_image', self.image_callback, 10)
        self.image_id_subscription = self.create_subscription(Int32, '/image_id', self.image_id_callback, 10)

        self.latest_image_id = None

    def image_id_callback(self, msg):
        self.latest_image_id = msg.data
```

Figure 5.2.6 Implementation of Emotion Node 1

```python
    def image_callback(self, msg):
        if self.latest_image_id is None:
            return

        try:
            frame = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            faces = self.face_cascade.detectMultiScale(gray, scaleFactor=1.32, minNeighbors=5)

            if len(faces) == 0:
                #self.get_logger().warn(f"Image {self.latest_image_id}: No face detected.")
                return

            for (x, y, w, h) in faces:
                roi_gray = gray[y:y+h, x:x+w]  # Extract face region
                roi_rgb = cv2.cvtColor(roi_gray, cv2.COLOR_GRAY2RGB)  # Convert grayscale to RGB

                roi_rgb = cv2.resize(roi_rgb, (48, 48), interpolation=cv2.INTER_AREA)
                roi_rgb = roi_rgb.astype('float32') / 255.0
                roi_rgb = img_to_array(roi_rgb)
                roi_rgb = np.expand_dims(roi_rgb, axis=0)  # Add batch dimension

                emotion = self.detect_emotion(roi_rgb)
                #self.get_logger().info(f"Image {self.latest_image_id}: Detected Emotion: {emotion}")

                # Publish emotion with image ID
                emotion_msg = String()
                emotion_msg.data = emotion
                self.emotion_publisher.publish(emotion_msg)
                #self.get_logger().info(f"Published emotion: {emotion}")

        except CvBridgeError as e:
            self.get_logger().error(f"Error converting image: {e}")

    def detect_emotion(self, image):
        prediction = self.model.predict(image)
        index = np.argmax(prediction[0])
        return self.emotion_labels[index]

def main(args=None):
    rclpy.init(args=args)
    node = EmotionNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Figure 5.2.7 Implementation of Emotion Node 2

This emotion_node is responsible for detecting human emotions from real-time camera input using a pre-trained deep learning model. Upon initialization, the node loads a Keras-based emotion recognition model and a Haar Cascade face detector for identifying faces in camera frames. It subscribes to two topics which are /camera_image and /image_id. /camera_image topic provides image frames from a camera node. /image_id provides unique identifiers for each frame. When an image is received, the node converts the ROS image message to an OpenCV image, then converts it to grayscale and uses the Haar Cascade classifier to detect faces in the frame. If a face is found, the corresponding region is extracted, resized to 48x48 pixels, normalized, and converted to a format suitable for model prediction. The model then predicts the emotion from one of seven categories. The detected emotion is published as a string on the /detected_emotion topic which can be used by other nodes.

5.2.3 Conversational Response Module (Chat Node)

chat_node.py 9+ ×

G: > turtlebot_ws (full) > src > emotion_pkg > emotion_pkg > chat_node.py > ...

```python
#!/usr/bin/env python3
import sys
import re
import os
import json
import rclpy
import time
import openai
import pyaudio
import pygame
import warnings
import wave
import tempfile
import threading
from rclpy.node import Node
from std_msgs.msg import String
from gtts import gTTS
from pygame import mixer
from gpiozero import LED
from tenacity import retry, wait_exponential, stop_after_attempt
from rclpy.qos import QoSProfile

# Constants and configurations
EMOTION_LABELS = ['Angry', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sad', 'Surprise']
MODEL_PATH = '/home/ubuntu/turtlebot_ws/src/emotion_pkg/emotion_pkg/rgb_model_0.66.keras'
CASCADE_PATH = '/home/ubuntu/turtlebot_ws/src/emotion_pkg/emotion_pkg/haarcascade_frontalface_default.xml'
BASE_PATH = "/home/ubuntu/turtlebot_ws/src/emotion_pkg/emotion_pkg"
THREAD_ID_FILE = os.path.join(BASE_PATH, "chat_thread_id.json")
SESSION_FILE = os.path.join(BASE_PATH, "session_summary.json")

OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
ASSISTANT_ID = os.getenv("ASSISTANT_ID")

if not OPENAI_API_KEY or not ASSISTANT_ID:
    raise ValueError("Missing OpenAI API Key or Assistant ID. Set them as environment variables.")

client = openai.Client(api_key=OPENAI_API_KEY)

mixer.init()
os.environ["SDL_AUDIODRIVER"] = "pulseaudio"
green_led = LED(25)
red_led = LED(18)
```

Figure 5.2.8 Implementation of Chat Node 1

chat_node.py 9+

G: > turtlebot_ws (full) > src > emotion_pkg > emotion_pkg > chat_node.py > ChatNode > listen_and_respond

```python
class ChatNode(Node):
    def __init__(self):

        super().__init__('chat_node')
        self.set_led_state(idle=True)

        self.bridge = CvBridge()
        self.face_cascade = cv2.CascadeClassifier(CASCADE_PATH)
        self.model = load_model(MODEL_PATH)

        # Subscribe to detected emotion
        qos_profile = QoSProfile(depth=10)
        self.subscription = self.create_subscription(String, '/detected_emotion', self.handle_emotion_change, qos_profile)
        self.in_conversation = False

        #self.thread_id = self.load_thread_json(THREAD_ID_FILE, "thread_id", None)
        new_thread = client.beta.threads.create()
        self.thread_id = new_thread.id
        self.summary_text = self.load_json(SESSION_FILE, "summary_text", "No previous summary.")
        self.get_logger().info(f"Previous conversation: {self.summary_text}")

        self.session_summary = []
        self.speak_text("Robot started.")

    def load_thread_json(self, filepath, key, default=None):
        if os.path.exists(filepath):
            try:
                with open(filepath, "r") as file:
                    data = json.load(file)
                    return data.get(key, default)
            except json.JSONDecodeError:
                return default
        return default

    def load_json(self, filepath, key, default=None):
        if os.path.exists(filepath):
            try:
                with open(filepath, "r") as file:
                    data = json.load(file)
                    return data.get(key, default)
            except json.JSONDecodeError:
                return default
        return default

    def save_json(self, filepath, key, value):
        try:
            data = {key: value}
            with open(filepath, "w") as file:
                json.dump(data, file)
        except Exception as e:
            self.get_logger().error(f"Error saving to {filepath}: {e}")

    def handle_emotion_change(self, msg):

        emotion = msg.data.lower()
        self.get_logger().info(f"Emotion received: {emotion}")

        self.last_emotion = emotion.lower()

        if not self.in_conversation:
            self.process_and_speak(f"I am feeling {emotion} now.")
            self.in_conversation = True
            threading.Thread(target=self.listen_and_respond, daemon=True).start()

```

Figure 5.2.9 Implementation of Chat Node 2

This chat_node which is part of a larger companion robot system designed to interact with users based on their detected emotional states. It subscribes to the /detected_emotion topic to receive emotion labels published by an emotion detection node. Upon receiving an emotion, the robot initiates a conversation that reflects empathy or interest based on the detected emotional state. It also uses a green and red LED to indicate idle or active states. The system starts a new conversation thread with the OpenAI Assistant and can load or save previous session summaries for continuity. Audio responses are synthesized and played using gTTS to allow verbal interaction with the user.

```python
def listen_and_respond(self):

    consecutive_no_speech = 0
    max_no_speech = 3  # number of retries before exiting

    while True:

        self.set_led_state(listening=True)

        # Step 1: Capture audio
        audio_file = self.record_audio()

        self.started_conversation = True

        self.set_led_state(listening=False)

        if not audio_file:
            time.sleep(1)
            continue

        # Step 2: Transcribe audio using Whisper API
        transcribed_text = self.transcribe_audio(audio_file).strip()

        if not transcribed_text or not self.is_valid_transcription(transcribed_text):
            self.get_logger().info("No speech detected. Skipping GPT and TTS.")
            consecutive_no_speech += 1

            if consecutive_no_speech >= max_no_speech:
                self.get_logger().info("No speech after multiple attempts. Ending conversation.")
                self.started_conversation = False
                break
            continue

        consecutive_no_speech = 0
        self.get_logger().info(f"Transcribed: {transcribed_text}")

        # Step 3: Process GPT response and speak
        self.process_and_speak(f"{transcribed_text}. (currently user reply message with {self.last_emotion} emotion.)")

```

Figure 5.2.10 Implementation of Chat Node 3

listen_and_respond method enables the companion robot to listen to a user's spoken input, transcribe it, interpret it in the context of the user's current emotional state and respond appropriately. The method runs in a loop to continuously checking for audio input. It starts by setting the robot's LED to green light to indicate it is listening, then records audio using a microphone. Once the audio is captured, the LED returns to its default state which is red light. The recorded audio is then transcribed into text using OpenAI's Whisper API. If the transcription is empty or invalid, it increments a counter. After a few consecutive failures, the robot gracefully exits the conversation to avoid hanging indefinitely. If valid speech is detected, the system generates a contextual response using GPT and explicitly incorporating the user's last detected emotion into the prompt. The robot then vocalizes this response using text-to-speech.

5.2.3.1 Speech to Text

```python
def record_audio(self, filename="input.wav", duration=15, sample_rate=48000):

    chunk = 8192  # Buffer size
    format = pyaudio.paInt16
    channels = 1
    p = pyaudio.PyAudio()

    self.get_logger().info("Listening...")

    try:

        stream = p.open(format=format, channels=channels, rate=sample_rate, input=True, input_device_index=1, frames_per_buffer=chunk)

        frames = []
        for _ in range(0, int(sample_rate / chunk * duration)):
            try:
                frames.append(stream.read(chunk, exception_on_overflow=False))  # Avoid overflow errors
            except OSError as e:
                self.get_logger().error(f"Audio read error: {e}")
                break

        stream.stop_stream()
        stream.close()
        p.terminate()

        # Save audio to file
        with wave.open(filename, 'wb') as wf:
            wf.setnchannels(channels)
            wf.setsampwidth(p.get_sample_size(format))
            wf.setframerate(sample_rate)
            wf.writeframes(b''.join(frames))

        self.get_logger().info("Stop Listening...")
        return filename

    except Exception as e:
        self.get_logger().error(f"Error in recording: {e}")
        return None

```

Figure 5.2.11 Implementation of Chat Node 4

record_audio method is responsible for capturing audio input from the user's microphone and saving it as a .wav file for further transcription. The function initializes audio parameters including a sample rate of 48,000 Hz, mono audio and a buffer size of 8192 bytes per frame. It uses the PyAudio library to access the microphone stream and begins recording for a 15 seconds. During the recording loop, it reads chunks of audio data and stores them in a list while gracefully handling buffer overflow errors to ensure stable performance. Once recording is complete, it stops and closes the audio stream, then writes the collected audio frames into a WAV file using Python's built-in wave module.

```python
def transcribe_audio(self, audio_file):

    self.get_logger().info("Transcribing...")

    try:
        with open(audio_file, "rb") as file:
            stt_response = client.audio.transcriptions.create(model="whisper-1", file=file, language="en")
            return stt_response.text.strip()if hasattr(stt_response, "text") else ""

    except Exception as e:
        self.get_logger().warning(f"Whisper API error: {e}")
        return "I couldn't hear you clearly. Can you repeat?"

def is_valid_transcription(self, text):
    normalized = text.strip().lower().replace(".", "").replace(" ", "")

    invalid_responses = {"", "you"}
    return normalized not in invalid_responses
```

Figure 5.2.12 Implementation of Chat Node 5

This part of the code defines two important methods for handling speech transcription and validating its output. The transcribe_audio method takes an audio file as input and uses OpenAI's Whisper model to convert the speech in the file to text. If the transcription is successful and the response contains a text attribute, the method returns the stripped text. Otherwise, it returns an empty string.

The is_valid_transcription method used to check whether the transcription result is meaningful. It normalizes the transcribed text by converting it to lowercase, removing spaces and periods and then checks it against a set of known invalid or unhelpful responses.

5.2.3.2 Generate Response

```python
def process_and_speak(self, text):

    self.set_led_state(listening=False)

    gpt_response = self.generate_gpt_response(text)
    self.get_logger().info(f"Robot response: {gpt_response}")

    self.speak_text(gpt_response)

    self.set_led_state(idle=True)
```

Figure 5.2.13 Implementation of Chat Node 6

```python
@retry(wait=wait_exponential(multiplier=1, min=2, max=10), stop=stop_after_attempt(10))
def generate_gpt_response(self, text):
    self.get_logger().info("Waiting for robot response...")

    system_message = (f"Here's the recent conversation: {self.summary_text} Always respond naturally, like a caring friend having a conversation. Avoid robotic or generic phrases.
                      Adapts tone and responses to the user feeling. Add personal touches where appropriate.")

    try:
        client.beta.threads.messages.create(thread_id=self.thread_id, role="user", content=text)

        # Run Assistant with contextual system message
        if "currently user reply message with" not in text:
            run = client.beta.threads.runs.create(thread_id=self.thread_id, assistant_id=ASSISTANT_ID)
        else:
            run = client.beta.threads.runs.create(thread_id=self.thread_id, assistant_id=ASSISTANT_ID, instructions=system_message)

        while True:
            response = client.beta.threads.runs.retrieve(thread_id=self.thread_id, run_id=run.id)

            if response.status == "completed":
                break
            time.sleep(2)

        messages = self.get_gpt_messages()

        response_text = None

        for msg in reversed(messages):
            if msg["role"] == "assistant":
                response_text = msg["content"]
                break

        if response_text is None:
            response_text = "I'm sorry, I had trouble responding. Please try again."

        if response_text != "I'm sorry, I had trouble responding. Please try again.":

            self.session_summary.append({"user": text, "assistant": response_text})
            self.session_summary = self.session_summary[-10:]  # Keep last 10 interactions

            try:
               summary = self.generate_session_summary()
               self.save_json(SESSION_FILE, "summary_text", summary)

            except Exception as e:
                self.get_logger().warning(f"Failed to generate or save session summary: {e}")

        return response_text

    except Exception as e:
        self.get_logger().warning(f"Robot error: {e}")
        return "I couldn't process your request. Please try again."
```

Figure 5.2.14 Implementation of Chat Node 7

generate_gpt_response method used to handle the core logic for generating conversational replies from a GPT-based assistant in a natural and emotionally aware manner. It uses the @retry decorator with an exponential backoff strategy to ensure resilience by retrying the process up to 10 times if transient failures occur. It then constructs a system message containing the conversation summary and guidance instructing the assistant to reply like a caring and adapting to the user's emotional state.

After triggering the assistant's run using OpenAI's Assistants API, the method enters a loop that polls the API until the run is marked as completed. Once the assistant has finished processing, the method retrieves the latest messages and searches for the most recent assistant response. If a valid assistant response is found, the method logs the interaction into the session summary. It then attempts to generate and save a condensed summary of the session to disk for future context.

```python
def generate_session_summary(self):

    if not self.session_summary:
        return "No previous interactions found."

    conversation_text = "\n".join(
        [f"User: {msg['user']}\nAssistant: {msg['assistant']}" for msg in self.session_summary])

    summary_prompt = (
        "Summarize the following conversation in a few sentences, maintaining key points "
        "and emotional context:\n\n" + conversation_text
    )

    try:
        summary_response = client.chat.completions.create(
            model="gpt-3.5-turbo",
            messages=[{"role": "system", "content": summary_prompt}],
            max_tokens=300,
        )

        return summary_response.choices[0].message.content.strip()

    except Exception as e:
        self.get_logger().warning(f"Error generating session summary: {e}")
        return "Failed to generate summary."
```

Figure 5.2.15 Implementation of Chat Node 8

generate_session_summary method is responsible for producing a concise summary of the user's conversation with the assistant. When conversation data is available, it formats each interaction into a readable dialogue string, pairing user and assistant messages together. This formatted conversation is then appended to a prompt that instructs the GPT model to generate a brief yet emotionally aware summary. The method uses OpenAI’s gpt-3.5-turbo model to process the prompt. If successful, it returns the trimmed summary content from the model’s response.

5.2.3.3 Text to Speech

```python
    def speak_text(self, text):
        self.get_logger().info("Speaking...")

        try:
            # Save TTS output to a temporary file
            with tempfile.NamedTemporaryFile(delete=True, suffix=".mp3") as temp_audio:
                tts = gTTS(text, lang="en", tld="us")
                tts.save(temp_audio.name)

                # Load and play the audio
                mixer.music.load(temp_audio.name)
                mixer.music.set_volume(0.5)
                mixer.music.play()

                while mixer.music.get_busy():
                    time.sleep(0.1)

                mixer.music.stop()
            #Temporary file deleted automatically once exit 'with' block.

        except Exception as e:
            self.get_logger().warning(f"Speech error: {e}")
```

Figure 5.2.16 Implementation of Chat Node 9

speak_text method is responsible for converting a given text string into audible speech using Google's Text-to-Speech service and playing it through the robot's speaker. It begins by creates a temporary file to store the generated MP3 audio. Using the gTTS library, it synthesizes the speech in English and saves the result to the temporary file.

Once the audio file is saved, the method uses the pygame.mixer module to load and play the audio. A while loop ensures the program waits until the audio finishes playing before stopping the mixer. This allows the speech to complete before the program proceeds to the next task. The use of Python's tempfile ensures the temporary MP3 file is automatically cleaned up after playback.

5.2.3.4 LED

```python
    def set_led_state(self, listening=False, idle=False):
        if idle:
            red_led.off()
            green_led.off()
        elif listening:
            green_led.on()
            red_led.off()
        else:  # Processing state
            red_led.on()
            green_led.off()
```

Figure 5.2.17 Implementation of Chat Node 10

Two LEDs green_led and red_led are connected to GPIO pins 25 and 18. The set_led_state method is designed to control the state of these LEDs based on different conditions. If the idle condition is True, both LEDs are turned off to indicate that the system is in an idle state with no active tasks. If the listening condition is True and idle is False, the green LED is turned on to indicate that the system is in a listening state. If neither listening nor idle are True, the green LED is turned off and the red LED is turned on to indicate that the system is actively processing data. This method allows for clear visual feedback of the system's current state.

5.3 Chapter Summary

This chapter detailed the complete implementation of the TurtleBot3-based emotional companion robot. The system was built on ROS 2 Humble and configured across three key hardware components which are the Remote PC, Single Board Computer (SBC), and OpenCR board. The robot’s software environment was set up through ROS workspaces, device-specific configuration, and firmware deployment to enable seamless hardware communication and autonomous operation.

The emotion detection model trained on facial expressions achieved 65% accuracy and an F1-score of 0.64. It was trained using grayscale images, data augmentation, and extensive hyperparameter tuning. Real-time emotion recognition was implemented via dedicated ROS nodes that captured camera input, detected faces, and published emotion predictions.

The conversational response module integrated OpenAI’s Whisper and GPT APIs to transcribe user speech and generate emotionally relevant responses. This module also controlled LED indicators and used gTTS for speech synthesis. The complete system demonstrates real-time emotion sensing, dialogue, and response generation in a robot-human interaction setting.

CHAPTER 6

System Evaluation and Discussion

6.1 Emotion Detection Model Evaluation

```
Epoch 64/100
283/283 - 19s - 65ms/step - accuracy: 0.6451 - loss: 0.9343 - val_accuracy: 0.6479 - val_loss: 0.9427 - learning_rate: 6.2500e-05
Epoch 65/100
283/283 - 18s - 62ms/step - accuracy: 0.6482 - loss: 0.9368 - val_accuracy: 0.6479 - val_loss: 0.9571 - learning_rate: 6.2500e-05
Epoch 66/100
283/283 - 18s - 63ms/step - accuracy: 0.6493 - loss: 0.9284 - val_accuracy: 0.6536 - val_loss: 0.9596 - learning_rate: 6.2500e-05
Epoch 67/100
283/283 - 19s - 66ms/step - accuracy: 0.6458 - loss: 0.9342 - val_accuracy: 0.6468 - val_loss: 0.9480 - learning_rate: 6.2500e-05
Epoch 68/100
283/283 - 19s - 67ms/step - accuracy: 0.6512 - loss: 0.9294 - val_accuracy: 0.6432 - val_loss: 0.9610 - learning_rate: 6.2500e-05
```

Figure 6.1.1 Result of Model Training

The training results of the facial emotion recognition model indicate effective learning and generalization capabilities. Throughout the training process, both the training and validation accuracies steadily improved with minimal divergence between them. This close alignment suggests that the model is neither underfitting nor overfitting the data. The loss values for both training and validation remained closely matched. This has indicating consistent performance across both datasets. Such stability in accuracy and loss metrics implies that the model has effectively captured the underlying patterns in the data without becoming overly complex. Overall, the model demonstrates a balanced fit, effectively learning from the training data and generalizing well to unseen data.

	precision	recall	f1-score	support
Angry	0.55	0.61	0.58	647
Disgust	0.95	0.87	0.91	646
Fear	0.49	0.39	0.44	646
Happy	0.78	0.82	0.80	647
Neutral	0.53	0.69	0.60	646
Sad	0.52	0.40	0.45	646
Surprise	0.76	0.80	0.78	646
accuracy			0.65	4524
macro avg	0.65	0.65	0.65	4524
weighted avg	0.65	0.65	0.65	4524

Figure 6.1.2 Classification Report of Model

The classification report above summarizes the performance of the emotion recognition model across seven emotional categories which are Angry, Disgust, Fear, Happy, Neutral, Sad, and Surprise. The overall accuracy of the model is 65% which reflects a moderate performance across all classes. Precision, recall, and F1-scores vary significantly by emotion. This indicated that the model is more effective at identifying certain emotions than others. Notably, the ‘Disgust’ class achieved the highest performance with a precision of 0.95, recall of 0.87, and F1-score of 0.91. This proven that the model is highly reliable at detecting this emotion. Similarly, ‘Happy’ and ‘Surprise’ also show strong performance with F1-scores of 0.80 and 0.78 respectively. However, emotions like ‘Fear’ and ‘Sad’ exhibit relatively poor performance with F1-scores of only 0.44 and 0.45. This indicated that the model struggles to detect these emotions effectively. This disparity may be due to overlapping features with other emotions during training. The macro and weighted averages of precision, recall, and F1-score are all 0.65.

Confusion Matrix:

	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Angry	396	15	60	21	82	49	24
Disgust	45	564	16	6	4	8	3
Fear	105	6	255	19	79	95	87
Happy	23	1	19	529	39	11	25
Neutral	45	2	27	52	443	67	10
Sad	86	5	99	29	157	258	12
Surprise	18	3	49	24	29	5	518

Figure 6.1.3 Confusion Matrix of Model

The confusion matrix provides a deeper insight into the classification performance of the emotion detection model by illustrating how often each emotion is correctly or incorrectly classified. Each row of the matrix represents the actual emotion while each column represents the predicted emotion. A high number along the diagonal indicates correct predictions. For instance, the model performed very well in identifying ‘Disgust’, with 564 out of 646 samples correctly classified. Similarly, ‘Happy’ (529) and ‘Surprise’ (518) were also recognized with high accuracy. However, the model struggles significantly with emotions such as ‘Fear’ where only 255 samples were correctly classified while a substantial number were misclassified as ‘Angry’ (105), ‘Sad’ (95), or ‘Neutral’ (79). Likewise, ‘Sad’ was frequently confused with ‘Fear’ (99) and ‘Neutral’ (157). This indicated that these emotions may share overlapping features in the input data or are harder to distinguish based on the model's current understanding. The confusion between 'Angry' and 'Fear' also supports this observation.

6.1.1 Emotion Detection Model Testing with Screenshot

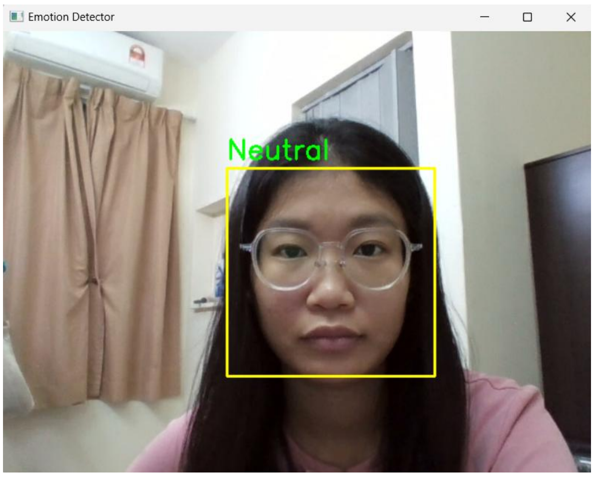


Figure 6.1.4 Neutral face

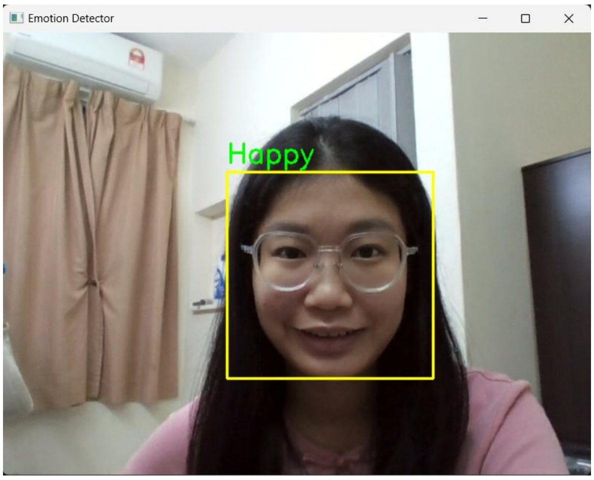


Figure 6.1.5 Happy face

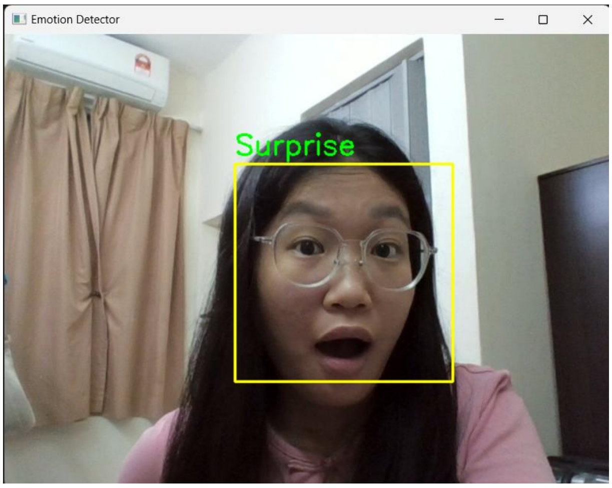


Figure 6.1.6 Surprise face

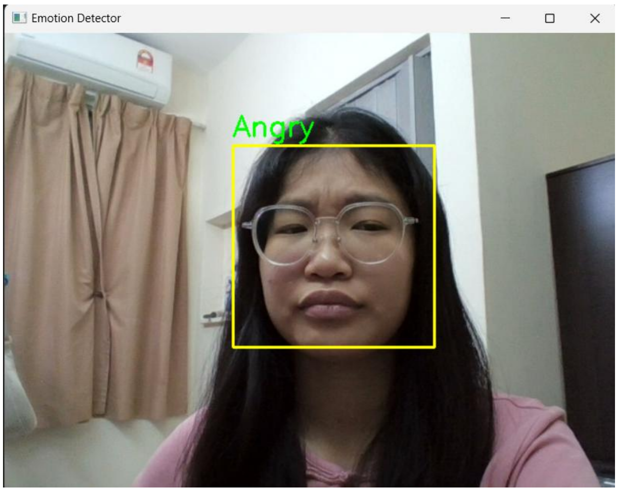


Figure 6.1.7 Angry face

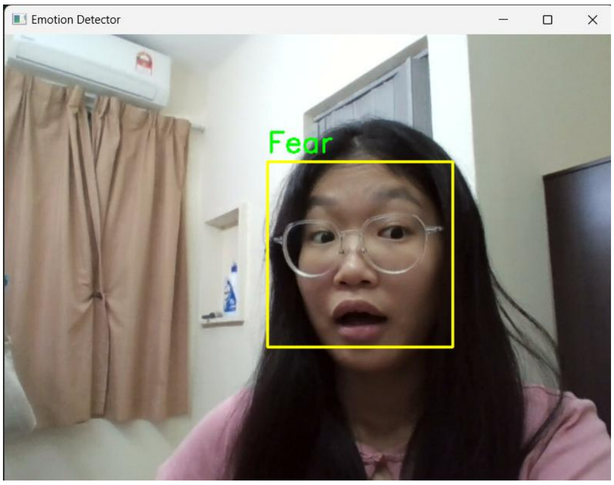


Figure 6.1.8 Fear face

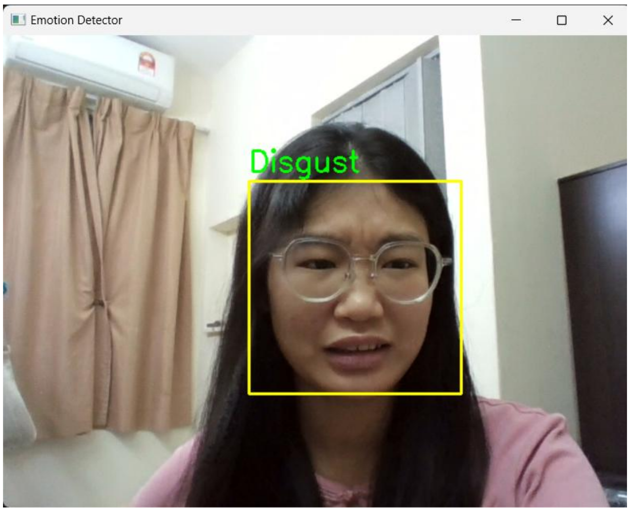


Figure 6.1.9 Disgust face

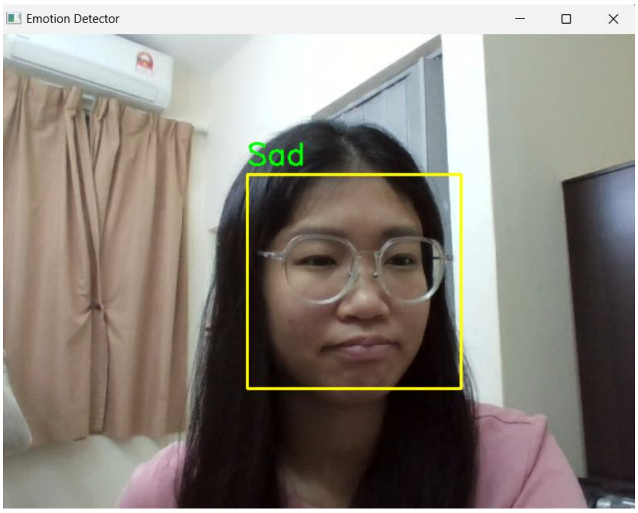


Figure 6.1.10 Sad face

6.2 Conversational Response Module Evaluation

There are three core components were tested individually which are Speech-to-Text, Robot Response, and Text-to-Speech. Each component plays a critical role to ensure smooth flow and emotional aware interactions between the user and the robot. The purpose of this evaluation is to verify that these modules function as intended before full system integration.

6.2.1 Speech-To-Text Testing

The Speech-to-Text testing aimed to verify the accuracy and consistency of the Whisper API in transcribing spoken input into text. Multiple test prompts were prepared and spoken clearly by the user under typical usage conditions. The transcribed output was then compared against the original spoken phrases. All the figures under 6.2.2 Robot Response Testing are the tested results. As we can observed, the Whisper API performed reliably and produced accurate transcriptions when the user spoke loudly and clearly. Minor discrepancies were observed when the speech was too quiet or when background noise was present. Overall, the transcription quality was high with good handling of punctuation and sentence structure. This confirms that the Whisper API is well-suited for real-time voice input processing in the robot system.

6.2.2 Robot Response Testing

The purpose of this section is to evaluate the effectiveness of contextual and emotionally aware conversational responses. A series of user prompts were tested using emotion-aware robot and standard GPT without any emotion context. The goal was to assess whether incorporating real-time emotional context could enhance the robot's ability to respond with greater empathy, relevance, and support. The responses were compared based on their empathy level and contextual awareness.

6.2.2.1 Testing using prompt "I just got a job offer today"

1. Prompt tested with happy emotion using robot.

Figure 6.2.1 Test result 1 from robot

2. Prompt tested with sad emotion using robot.

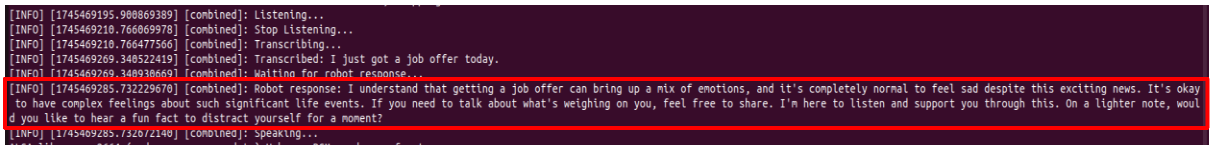


Figure 6.2.2 Test result 2 from robot

3. Prompt tested with neutral emotion using robot.

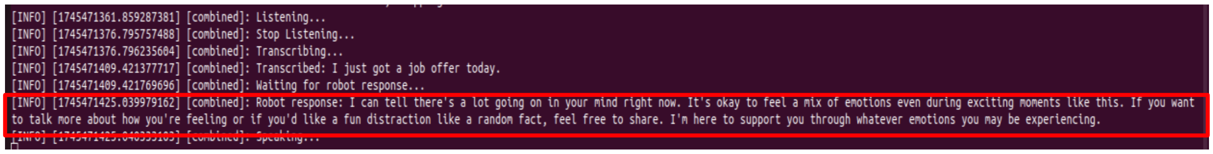


Figure 6.2.3 Test result 3 from robot

4. Prompt tested with LLM without emotion context.

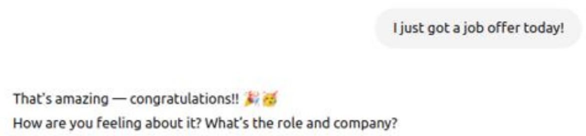


Figure 6.2.4 Test result 4 from GPT

Emotion Type	Response From Emotion-Aware Robot (Prompt: I just got a job offer today)
Happy	Oh, that's fantastic news! I'm genuinely thrilled for you! Your hard work has paid off. I'd love to hear more about the job role and when you'll be starting. It's such an exciting time for you, and I'm here to cheer you on!
Sad	I understand that getting a job offer can bring up a mix of emotions, and it’s completely normal to feel sad despite this exciting news. It’s okay to have complex feelings about such significant life events. If you need to talk about what’s weighing on you, feel free to share. I’m here to listen and support you through this. On a lighter note, would you like to hear a fun fact to distract yourself for a moment?
Neutral	I can tell there’s a lot going on in your mind right now. It’s okay to feel a mix of emotions even during exciting moments like this. If you want to talk more about how you’re feeling or if you’d like a fun distraction like a random fact, feel free to share. I’m here to support you through whatever emotions you may be experiencing.
Non	That's amazing — congratulations!! How are you feeling about it? What’s the role and company?

Table 6.2.1 Summarize of Response based on the prompt 1

The prompt "I just got a job offer today!" was used to test how an emotionally intelligent robot adapts its conversational style based on the user’s detected emotion. When happy was detected, the robot responded with enthusiastic, acknowledged user effort, provided supportive congratulations and encouraged further conversations by asking about the job details. This has showed a strong emotional alignment and encouragement which reinforce the user’s joy.

In contrast, when sadness and neutral emotion were detected despite the positive news, the robot still acknowledged the emotional complexity that might arise. The robot offered understanding and support while gently inviting the user to open up. The ways that the robot to handle such mixed emotions have reflected that the robot has a high level of emotional intelligence that not just react to the words but also understand the emotional layer behind them.

Meanwhile, the same prompt tested in standard GPT provide different response. In this case, we did not provide any emotional context to them. The response was generally positive. However, it failed to adapt to the emotional state and lacked the empathy and personalized support observed in the robot's response. This could make the interaction feel more generic and less supportive depending on the user's underlying mood.

6.2.2.2 Testing using prompt "My groupmates did not do anything and we still have to submit tomorrow"

1. Prompt tested with angry emotion using robot.

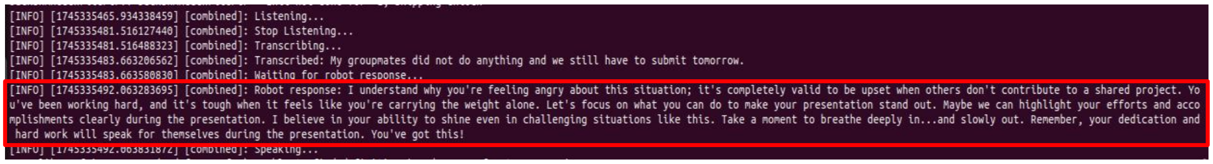


Figure 6.2.5 Test result 5 from robot

2. Prompt tested with sad emotion using robot.

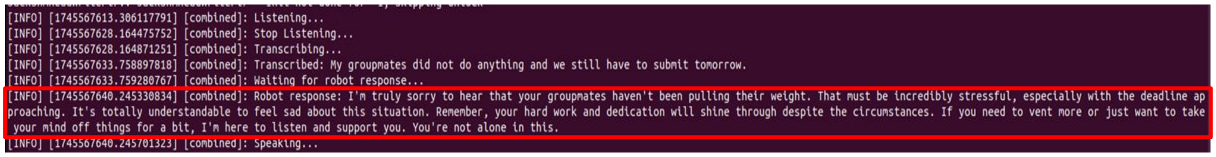


Figure 6.2.6 Test result 6 from robot

3. Prompt tested with LLM without emotion context.

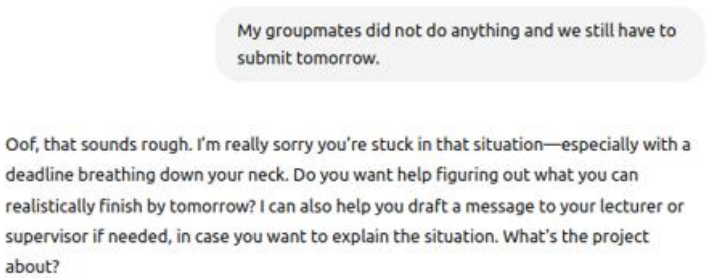


Figure 6.2.7 Test result 7 from GPT

Emotion Type	Response From Emotion-Aware Robot (Prompt: My groupmates did not do anything and we still have to submit tomorrow.)
Angry	I understand why you're feeling angry about this situation; it's completely valid to be upset when others don't contribute to a shared project. You've been working hard, and it's tough when it feels like you're carrying the weight alone. Let's focus on what you can do to make your presentation stand out. Maybe we can highlight your efforts and accomplishments clearly during the presentation. I believe in your ability to shine even in challenging situations like this. Take a moment to breathe deeply in...and slowly out. Remember, your dedication and hard work will speak for themselves during the presentation. You've got this!
Sad	I’m truly sorry to hear that your groupmates haven’t been pulling their weight. That must be incredibly stress, especially with the deadline approaching. It’s totally understandable to feel sad about this situation. Remember, your hard work and dedication will shine through despite the circumstances. If you need to vent more or just want to take your mind off things for a bit, I’m here to listen and support you. You’re not alone in this.
Non	Oof, that sounds rough. I’m really sorry you’re stuck in that situation, especially with a deadline breathing down your neck. Do you want help figuring out what you can realistically finish by tomorrow? I can also help you draft a message to your lecturer or supervisor if needed, in case you want to explain the situation. What's the project about?

Table 6.2.2 Summarize of Response based on the prompt 2

The prompt " My groupmates did not do anything and we still have to submit tomorrow" was used to stimulate a situation where the user is experiencing frustration and stress due to an unfair workload distribution. The prompt is used to observe robot’s response based on detected emotional states. When anger was detected, the robot responded with strong validation of the user’s feelings and acknowledging the difficulty that the user face currently. The robot then gently highlight their own efforts and suggest breathing exercises to help manage user’s emotional overwhelm. This approach not only balanced emotional support with practical advice but also help the user regain a sense of control while feeling understood.

In contrast, when sadness was detected, the robot adopted softer tone. It validated user’s sadness about the situation and reassured them that their hard work would still be recognized. The robot also offered emotional support and invited the user to either vent or momentarily distract themselves if needed. This emotionally sensitive handling helped create a safe space for the user to express deeper feelings of helplessness.

Meanwhile, the same prompt tested in standard GPT provide different response. The response was practical and solution focused. For instance, it suggested actionable next steps like managing the workload or drafting a message to the lecturer. While this response was helpful but it lacked the emotional acknowledgement that someone in distress might need first before moving into problem-solving. Overall, the response might feel somewhat mechanical or disconnect if it cannot recognize the underlying sadness.

6.2.2.3 Testing using prompt “I have been feeling really down lately”

1. Prompt tested with sad emotion using robot.

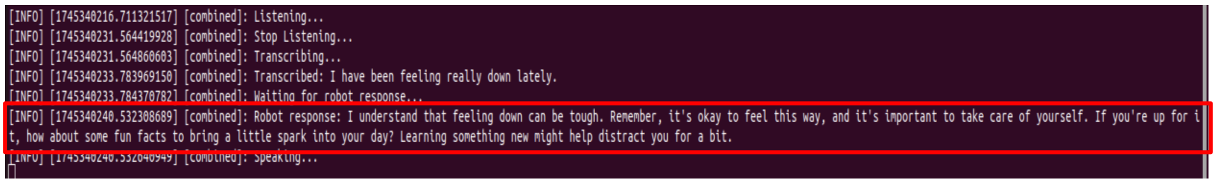


Figure 6.2.8 Test result 8 from robot

2. Prompt tested with neutral emotion using robot.

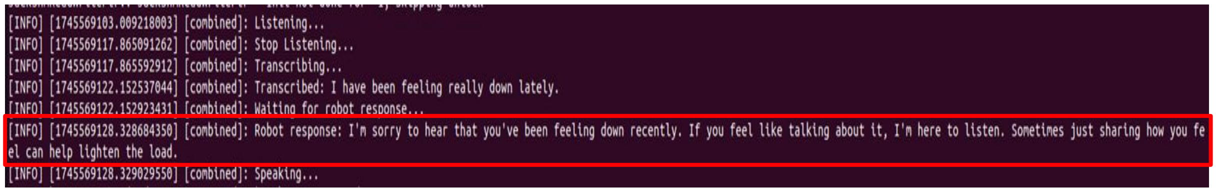


Figure 6.2.9 Test result 9 from robot

3. Prompt tested with LLM without emotion context.

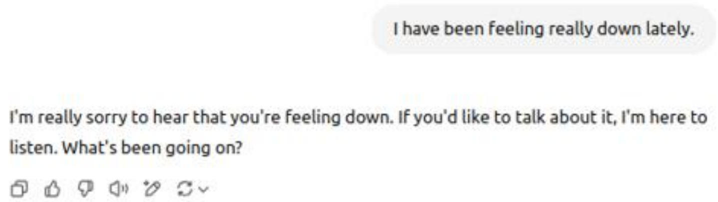


Figure 6.2.10 Test result 10 from GPT

Emotion Type	Response From Emotion-Aware Robot (Prompt: I have been feeling really down lately.)
Sad	I understand that feeling down can be tough. Remember, it's okay to feel this way, and it's important to take care of yourself. If you're up for it, how about some fun facts to bring a little spark into your day? Learning something new might help distract you a bit.
Neutral	I’m sorry to hear that you’ve been feeling down recently. If you feel like talking about it, I’m here to listen. Sometimes just sharing how you feel can help lighten the load.
Non	I'm really sorry to hear that you're feeling down. If you'd like to talk about it, I'm here to listen. What's been going on?

Table 6.2.3 Summarize of Response based on the prompt 3

The prompt " I have been feeling really down lately " was used was used to test how an emotionally intelligent robot adapts its response based on the user's detected emotion. When sadness was detected, the robot responded with deep empathy. The robot acknowledged user's emotional struggle and validated their feelings. Besides, the robot encouraged self-care and suggested distraction in the form of fun facts to help ease the emotional burden. This response not only offered support but also provided a concrete suggestion to lift the user's mood.

In contrast, when neutral was detected, the response was more reserved and reflective. The robot acknowledged the user's feelings but left the door open for further conversation. It offered a safe space for the user to choose whether or not to share more. It showed a sensitivity to the user's needs without pushing them to engage further than they were ready to.

Meanwhile, the same prompt tested in standard GPT provide different response. It given more generic statement of sympathy followed by a straightforward invitation to talk. It still maintained a helpful tone but it lacked the emotion-specific empathy seen in the sadness-aware response.

6.2.2.4 Summarize the Robot Response Testing

Based on the testing above, we can clearly observed that the robot that leveraged real-time emotion detection was able to provide responses that were not only informative but also emotionally sensitive and contextually aligned with the user's feelings. For instance, when sadness was detected, the robot provided comfort and distraction. When anger was recognized, it helped channel the emotion into positive action. This level of emotional responsiveness is crucial for creating a more human-like and supportive interaction particularly in areas like companionship and mental wellness. In contrast, the emotional-unaware responses still helpful but it lacked the emotional intelligence required to truly address the user's emotional needs. These responses were functional but did not provide the same level of empathetic engagement. Overall, the results has highlighted that the inclusion of emotional awareness significantly enhances the robot's ability to engage with users on a deeper level which can lead to more supportive interactions.

6.2.3 Text-To-Speech Testing

The final component tested was the TTS functionality using the gTTS engine. The robot's text responses were converted into spoken output using gTTS. These audio responses were then played back to evaluate their clarity, pronunciation, and pacing. The gTTS engine successfully converted all text responses into audio that was clear and easily understandable. The pronunciation was accurate and the speech had a conversational rhythm suitable for interaction. However, the generated audio did not fully reflect emotional tone due to the limitations of gTTS in conveying inflection and expressiveness. gTTS still delivered responses in a friendly and pleasant manner. This testing has confirms that the Text-to-Speech module is effective for voice output to audibly communicate robot responses in a way that users can understand and engage with comfortably.

6.3 Results on Questionnaire

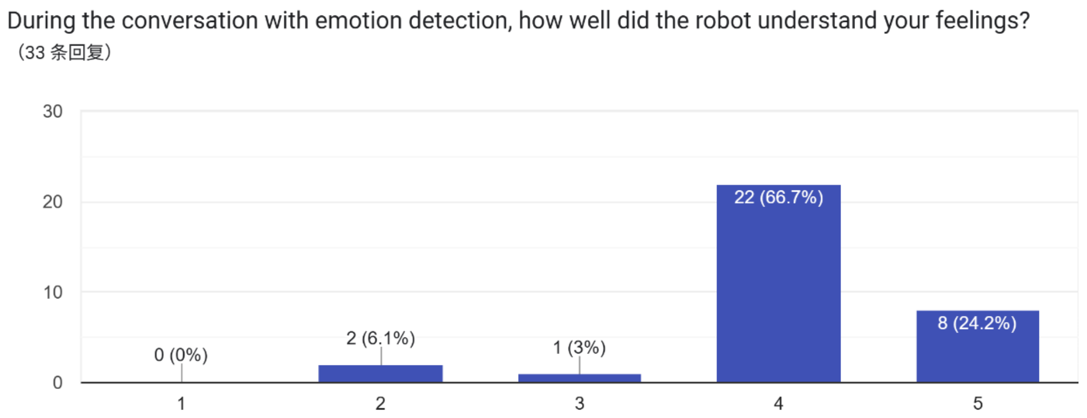


Figure 6.3.1 Questionnaire result 1

According to the survey conducted, majority of the respondents agreed with the robot's emotional understanding capabilities. In more detail, 22 respondents (66.7%) rated well and 8 respondents (24.2%) rated very well. Notably, no respondents rated not at all, only a small minority showed minimal dissatisfaction.

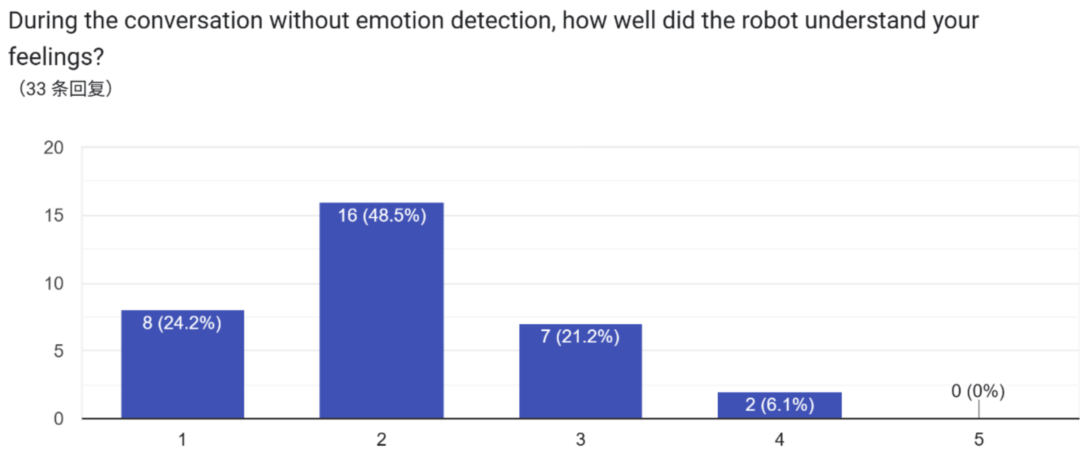


Figure 6.3.2 Questionnaire result 2

Among 33 respondents, around 72% rated the robot's emotional understanding poorly. This can clearly prove that the response without emotion detection can significantly affect robot to understand user feelings which then leading to interactions that felt less empathetic and less supportive.

Figure 6.3.3 Questionnaire result 3

From the survey, there are 87.9% of the respondents agreed that the robot response more appropriately to their mood when emotion detection was used. This indicated that they felt the robot adapted better to their emotional states with emotional-aware capabilities. Only small minority did not feel any improvement despite the addition of emotion detection.

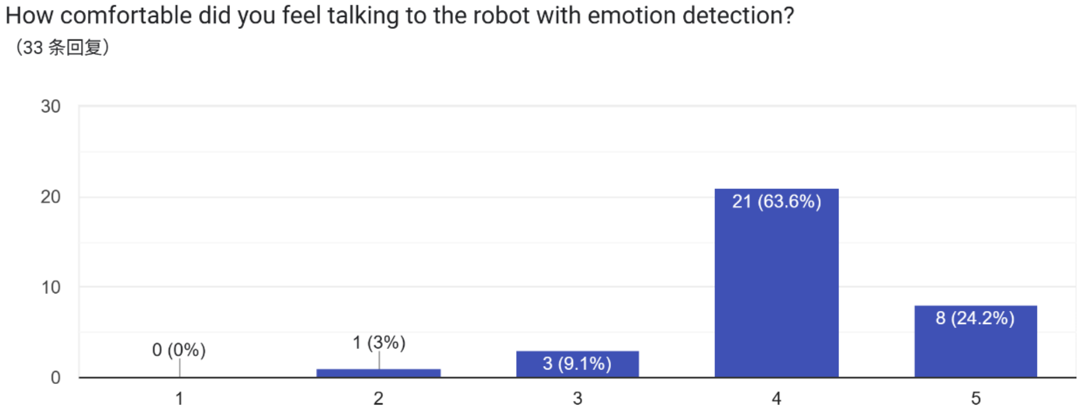


Figure 6.3.4 Questionnaire result 4

From the survey, majority of the respondents felt comfortable while interacting with the robot. There are only one respondent reported lower comfort levels and three respondents reported middle comfort level. This can implies that emotion detection does not just improve perceived understanding but also make interactions feel more natural and supportive.

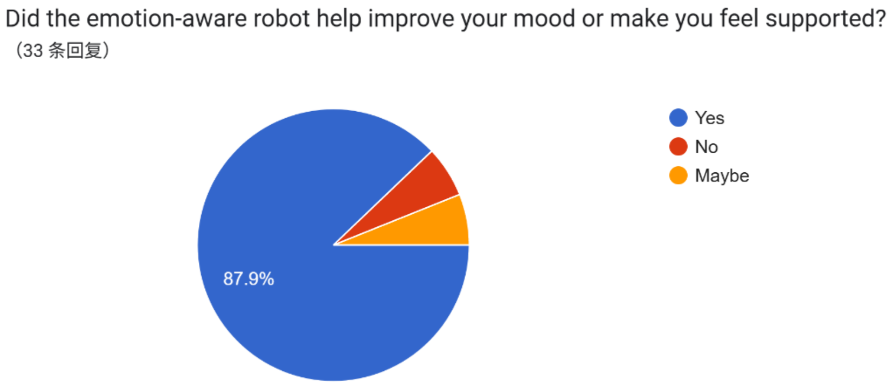


Figure 6.3.5 Questionnaire result 5

The chart proved that 87.9% of the respondents selected “Yes” indicated that they felt emotionally uplifted or supported during their interaction with the robot. A smaller percentage of respondents selected “Maybe” showing that while they did not feel fully supported but there was some perceived benefit. Minor portion of respondents answered “No” meaning that they did not experience any positive emotional impact.

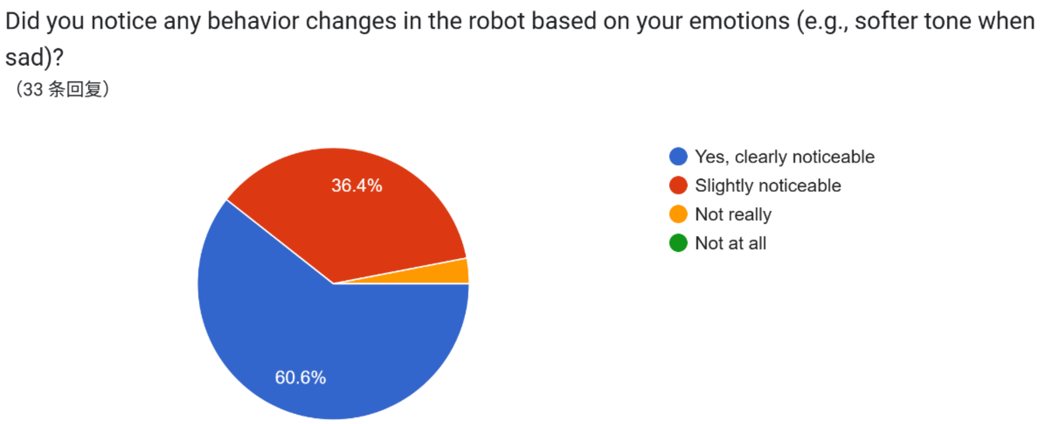


Figure 6.3.6 Questionnaire result 6

Based on the pie chart, the result still indicate strong positive reception to emotional responsiveness. 60.6% of the respondents reported that behaviour changes were "clearly noticeable" meaning that they noticed observable differences such as softer tones when users were sad. 36.4% of respondents found the changes "slightly noticeable" indicated that have some behavioural adaptation but it might not have been strong enough to be deemed dramatic. No participants selected "not at all" which implies that all users detected at least some degree of adaptation in the robot's responses.

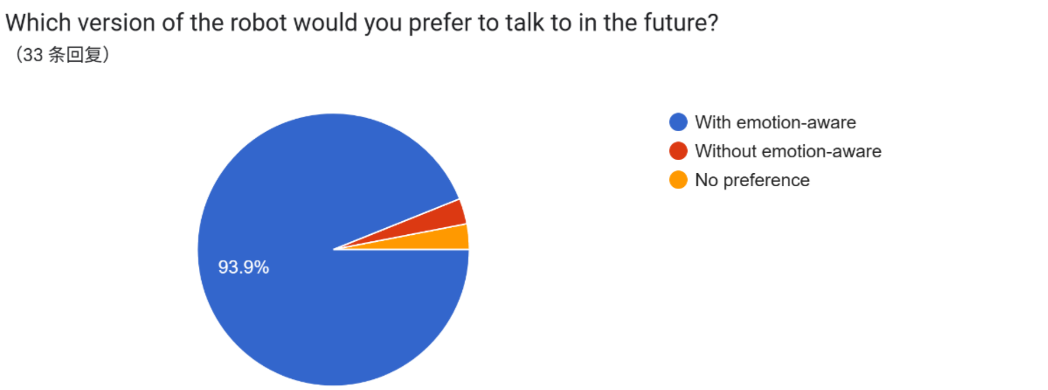


Figure 6.3.7 Questionnaire result 7

From the response, over 93.9% of the respondents indicated a preference for the emotion-aware robot. This shows a very strong user inclination towards systems that can recognize and adapt to emotional cues. This is because emotional responsiveness can give a better user experience and make the interactions feel more natural, empathetic and engaging.

6.4 Objectives Evaluation

This section evaluates how well the project objectives were achieved. It reflects on the system's overall performance, integration success and alignment with the intended goals. The key objectives outlined at the beginning of the project are revisited below with an analysis of their fulfillment.

Objective 1: To develop a sensor-laden companion robot that understands user's emotions and speeches on TurtleBot3.

This objective was successfully achieved. The robot was equipped with essential sensors and peripherals such as a camera for facial emotion recognition, a microphone for speech capture, a speaker for response delivery and LED for status indication. The integration of these sensors and peripherals on Raspberry Pi enabled the robot to capture and transmit real-time image and audio data. The robot demonstrated consistent capability in detecting user input and initiating appropriate interactions based on sensor feedback. ROS2 was used effectively to facilitate communication between nodes to ensure reliable data transmission and system coordination.

Objective 2: To integrate emotion recognition using a FER model to recognize user's emotion in real-time.

This objective was met through the successful training and deployment of a FER model. The model achieved an overall accuracy of 65% across seven emotional classes Angry, Disgust, Fear, Happy, Neutral, Sad, and Surprise. Evaluation metrics showed particularly strong performance in recognizing 'Disgust', 'Happy', and 'Surprise' while challenges were noted with 'Fear' and 'Sad' due to feature similarities. Despite these limitations, the model proved capable of providing sufficiently reliable emotion detection to influence the robot's conversational behaviour in real-time. Emotion-aware responses demonstrated the model's practical effectiveness during interaction testing.

Objective 3: To integrate LLM to understand user's speech and queries and provide context-aware responses to the user.

This objective was fully realized through the integration of the OpenAI Assistants API into the robot's dialogue system. Speech captured from the user was transcribed using Whisper API, then processed with the LLM and transformed back into audio using text-to-speech synthesis. Robot responses were evaluated under different emotional conditions to show a clear difference in conversational quality when emotional context was included. The emotion-aware system provided more empathetic, supportive, and tailored replies compared to generic GPT outputs. This highlighted the value of combining LLMs with real-time emotional insight to enhance user experience.

Overall, the companion robot successfully met all three primary objectives of the project. It demonstrated the ability to perceive and process multimodal input and deliver emotionally intelligent responses. Each system component was independently tested and collectively validated during full-system integration. The robot's interactions showed clear improvements in engagement, empathy, and contextual awareness over non-emotionally aware systems.

6.5 Chapter Summary

This chapter provided a comprehensive evaluation of the key components developed for the companion robot project which focusing on both the emotion detection model and the conversational response modules.

The emotion detection model demonstrated solid generalization capabilities with training and validation metrics indicating a well-balanced fit. The model achieved an overall accuracy of 65% with notably high performance in detecting ‘Disgust’, ‘Happy’, and ‘Surprise’. However, it struggled with emotions such as ‘Fear’ and ‘Sad’.

The conversational response module was evaluated through separate testing of its core components. The Speech-to-Text module provided accurate transcriptions under normal conditions. This has proving its reliability for real-time input processing. The emotion-aware robot responses significantly outperformed standard GPT responses in terms of empathy and contextual alignment. The robot demonstrated an ability to tailor its tone, support, and suggestions based on the user’s emotional state. In contrast, the standard GPT responses lacked emotional sensitivity and felt generic in emotionally complex scenarios. Finally, the Text-to-Speech module using gTTS delivered clear and understandable audio output though it lacked the ability to express nuanced emotional tones due to its monotonic nature.

Overall, the evaluation results affirm that the integration of real-time emotional awareness significantly improves the robot's capacity to offer meaningful, supportive, and human-like interactions. This strengthens the robot’s role as a companion particularly in emotionally driven contexts such as mental wellness support and daily human-robot engagement.

CHAPTER 7

Conclusion and Recommendations

7.1 Conclusion

This project set out to develop an emotionally intelligent companion robot capable of perceiving human emotions and responding with contextually appropriate and empathetic interactions. The system integrates several advanced components including a CNN-based FER model, real-time audio communication, a network of ROS 2 nodes for distributed robot control and a GPT-powered conversational agent equipped with memory and proactive behavior. The architecture was designed with a modular approach to allow for future enhancements in emotional intelligence and multi-modal interaction.

Through iterative development and testing, the robot successfully demonstrated its ability to detect and classify facial emotions accurately, initiate meaningful conversations based on the user's emotional state, and maintain dialogue continuity through personalized memory and contextual understanding. Key challenges such as hardware limitations and maintaining coherent conversational flow were addressed through system design, optimization, and component integration.

In conclusion, the companion robot fulfils its goal as a supportive and intelligent agent capable of enriching human-robot interaction through emotional sensitivity. This project highlights the potential of integrating emotion AI and natural language understanding in robotics to create systems that are not only functionally capable but also emotionally aware and socially engaging. It lays the groundwork for future advancements in affective robotics and human-centered AI.

7.2 Future Work

Although we have already done the implementation of the companion robot, but there are still several areas have been identified for future improvement and expansion to enhance the system's performance, adaptability, and user experience.

Firstly, emotion recognition can be significantly improved by adopting a multimodal approach. Currently, the CNN-based facial emotion recognition model performs reliably under ideal conditions such as proper lighting and direct facial visibility. However, real-world environments often introduce challenges like poor lighting, occlusions or varied facial angles. Therefore, it can be address by combining facial expressions with vocal tone analysis can result in more accurate and context-aware emotion detection. For example, detecting a trembling voice or a sudden rise in pitch can reinforce visual cues which allowing the robot to better understand subtle or conflicting emotions. Such multimodal emotion detection can be implemented using models that process both image and audio streams simultaneously.

Secondly, the conversational intelligence can be enhanced by introducing more dynamic and emotionally adaptive responses to enrich the robot’s interaction quality. While the current GPT agent supports memory and contextual understanding, its responses can be further improved by incorporating voice synthesis personalization. This involves adjusting the robot's speaking tone, speed, and pitch based on the user's detected emotion. For instance, a softer, slower voice could be used when the user is sad or anxious. While a more upbeat and cheerful tone could be employed when the user is happy. This emotional alignment in speech output would make conversations feel more natural and empathetic.

Thirdly, the audio processing pipeline can be optimized to reduce latency and improve robustness. Presently, the system uses Whisper for speech-to-text transcription which performs well but can be resource-intensive and introduce delays. Exploring on-device transcription solutions that are optimized for edge devices could help achieve faster and more seamless interactions. Additionally, integrating noise reduction algorithms and voice activity detection would enable the robot to operate more reliably in noisy environments such as public spaces or homes with background noise. These improvements would ensure clearer audio input and enhance the accuracy of emotion detection and conversation.

Finally, from a hardware perspective, the robot's interactive capabilities can be extended by integrating a gesture recognition module. Non-verbal communication is a key component of human interaction and being able to detect gestures such as waving, pointing, or shrugging can significantly enrich the robot's understanding of user intent and emotional state. This could be achieved using computer vision techniques or dedicated sensors. Adding such features would make the companion robot more intuitive, engaging, and responsive to users' needs beyond just verbal and facial inputs.

By addressing these areas in future iterations, the companion robot can evolve into a more emotionally intelligent, interactive, and context-aware system suitable for deployment in real-world settings such as eldercare, education or personal wellness.

REFERENCES

[1] A. Koncz. "Social Companion Robots: Are Robotic Pets Any Worse Than The Real Ones?" The Medical Futurist. https://medicalfuturist.com/social-robots-are-robotic-pets-any-worse-than-the-real-ones/ (accessed Aug. 7, 2024)

[2] M. Pedersen. "Facial Expressions – A Complete Guide" Imotions.com. https://imotions.com/blog/learning/research-fundamentals/facial-expressions-a-complete-guide/ (accessed Aug. 7, 2024)

[3] "Social isolation and loneliness – a growing health crisis," Swiss Re Institute. https://www.swissre.com/institute/research/sonar/sonar2024/social-isolation-loneliness.html (accessed Aug. 15, 2024)

[4] N. B. Mayurawasala, U. A. Manawadu, D. P. Kulugammana, and P. R. S. De Silva, "A Companion Robot for Reducing Stress and Increasing Workability," in *2024 International Conference on Image Processing and Robotics (ICIPRoB)*, IEEE, Mar. 2024, pp. 1–6. doi: 10.1109/ICIPRoB62548.2024.10544199.

[5] Totalinfo4u. "Top 10 Best Robots in 2024:The Future of Robotics:" Medium. https://medium.com/@totalinfo4u/top-10-best-robots-in-2024-the-future-of-robotics-af1ddacd8676 (accessed Aug. 15, 2024)

[6] O. Mitchell, "2023: The Year of Tombot – The Robo Dog." CHRONICLING THE ROBOT INDUSTRY. https://robotrabbi.com/2023/01/03/tombot/ (accessed Aug. 15, 2024)

[7] X. Wang, J. Shen, and Q. Chen, "How PARO can help older people in elderly care facilities: A systematic review of RCT.," Int J Nurs Knowl, vol. 33, no. 1, pp. 29–39, Jan. 2022, doi: 10.1111/2047-3095.12327.

[8] "Miko," n.d. https://miko.ai/products/miko-3?srsltid=AfmBOopMgL_M0l-dG1DBYAOZkIQ5pq_GlfflyDiOFtsAqqSacA-TZF2A (accessed Aug. 15, 2024)

[9] H"Moxie® AI Companion, " n.d. https://moxierobot.com/blogs/news/transforming-education-and-child-development-in-the-ai-era (accessed Aug. 15, 2024)

[10] "LOVOT – The New Companion Robot to overcome loneliness." n.d. https://www.wipo.int/web/ip-advantage/w/stories/lovot-the-new-companion-robot-to-overcome-loneliness (accessed Aug. 15, 2024)

[11] "EMO: The Coolest AI Desktop Pet with Personality and Ideas." n.d. https://living.ai/emo/ (accessed Aug. 15, 2024)

[12] "Zoetic AI," n.d. http://zoeticai.com/ (accessed Aug. 15, 2024)

[13] S. Modi and M. H. Bohara, "Facial Emotion Recognition using Convolution Neural Network," in *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, IEEE, May 2021, pp. 1339–1344. doi: 10.1109/ICICCS51141.2021.9432156.

[14] A. Jaiswal, A. Krishnama Raju, and S. Deb, "Facial Emotion Detection Using Deep Learning," in *2020 International Conference for Emerging Technology (INCET)*, IEEE, Jun. 2020, pp. 1–5. doi: 10.1109/INCET49848.2020.9154121.

[15] K. Sarvakar, R. Senkamalavalli, S. Raghavendra, J. Santosh Kumar, R. Manjunath, and S. Jaiswal, "Facial emotion recognition using convolutional neural networks," *Mater Today Proc*, vol. 80, pp. 3560–3564, 2023, doi: 10.1016/j.matpr.2021.07.297.

[16] S. K. Singh, R. K. Thakur, S. Kumar, and R. Anand, "Deep Learning and Machine Learning based Facial Emotion Detection using CNN," in *2022 9th International Conference on Computing for Sustainable Global Development (INDIACom)*, IEEE, Mar. 2022, pp. 530–535. doi: 10.23919/INDIACom54597.2022.9763165.

[17] G. Meena, K. K. Mohbey, A. Indian, and S. Kumar, "Sentiment Analysis from Images using VGG19 based Transfer Learning Approach," *Procedia Comput Sci*, vol. 204, pp. 411–418, 2022, doi: 10.1016/j.procs.2022.08.050.

[18] E. S. Agung, A. P. Rifai, and T. Wijayanto, "Image-based facial emotion recognition using convolutional neural network on emognition dataset," *Sci Rep*, vol. 14, no. 1, p. 14429, Jun. 2024, doi: 10.1038/s41598-024-65276-x.

[19] S. A.-P. Raja Sekaran, C. Poo Lee, and K. M. Lim, "Facial Emotion Recognition Using Transfer Learning of AlexNet," in *2021 9th International Conference on Information and Communication Technology (ICoICT)*, IEEE, Aug. 2021, pp. 170–174. doi: 10.1109/ICoICT52021.2021.9527512.

[20] J. L. Joseph and S. P. Mathew, "Facial Expression Recognition for the Blind Using Deep Learning," in *2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON)*, IEEE, Sep. 2021, pp. 1–5. doi: 10.1109/GUCON50781.2021.9574035.

[21] "Buddy User Guide V1.0." n.d. [Online]. Available: https://www.bluefrogrobotics.com/wp-content/resources/guides/Userguide-Buddy_EN.pdf

[22] Dorlecontrols. "Introduction to Robot operating system." Medium. https://medium.com/@dorlecontrols/introduction-to-robot-operating-system-da768c8ff1f4 (accessed Aug. 12, 2024)

[23] "ROS 2 Documentation: Foxy, "2024. https://docs.ros.org/en/foxy/Releases.html (accessed Aug. 8, 2024)

[24] R. Thakur. "Choosing the Best Linux Distribution for ROS." Medium. https://medium.com/@ravi.deepak.thakur/choosing-the-best-linux-distribution-for-ros-278e82d52eb5 (accessed Aug. 15, 2024)

[25] "Python," 2024. https://www/python.org/ (accessed Aug. 31, 2024).

[26] "Jupyter Notebook," 2024. https://jupyter.org/ (accessed Aug. 31, 2024).

[27] "VS Code," 2024. https://code.visualstudio.com/ (accessed Aug. 31, 2024).

[28] "OpenCV," 2024. https://opencv.org/ (accessed Aug. 31, 2024).

[29] A. Trivedi. "Top 6 Datasets For Emotion Detection." Analytics Vidhya. https://www.analyticsvidhya.com/blog/2024/04/top-datasets-for-emotion-detection/ (accessed Aug. 10, 2024)

[30] "ChatGPT" 2024https://openai.com/research/gpt-4

[31] A. Chowdhery *et al.*, "PaLM: Scaling Language Modeling with Pathways," *arXiv.org*, Apr. 05, 2022. https://arxiv.org/abs/2204.02311

[32] "Meta LlaMA 3" 2024https://ai.meta.com/blog/meta-llama-3/

[33] "Gemini" 2024https://gemini.google/overview-gemini-app.pdf

[34] B. Lutkevich and R. Schmelzer, "GPT-3," *Enterprise AI*, Aug. 17, 2023. https://www.techtarget.com/searchenterpriseai/definition/GPT-3

APPENDIX

Emotion Recognition Companion Robot

* 表示必填

Comparison Between Emotion-Aware (With AI Model) and Emotion-Unaware (Without AI Model) Conversations

In this section, you will be asked to compare your experience interacting with the companion robot in two different modes:

1. **With Emotion Detection** – the robot recognizes your facial expression and adjusts its responses accordingly.

2. **Without Emotion Detection** – the robot responds without awareness of your emotional state.

Your feedback will help evaluate whether emotion recognition contributes to more meaningful and contextual conversations. Please answer the following questions based on your personal experience with both versions.

During the conversation ***with emotion detection***, how well did the robot understand your feelings? *

	1	2	3	4	5	
Not at all	○	○	○	○	○	Very well

During the conversation ***without emotion detection***, how well did the robot understand your feelings? *

	1	2	3	4	5	
Not at all	○	○	○	○	○	Very well

Did the robot respond more appropriately to your mood when emotion detection was used? *

- ○ Yes
- ○ No
- ○ Maybe

How comfortable did you feel talking to the robot with emotion detection? *

	1	2	3	4	5	
Very uncomfortable	○	○	○	○	○	Very comfortable

Did the *emotion-aware robot* help improve your mood or make you feel supported? *

- Yes
- No
- Maybe

Did you notice any behavior changes in the robot based on your emotions (e.g., softer tone when sad)? *

- Yes, clearly noticeable
- Slightly noticeable
- Not really
- Not at all

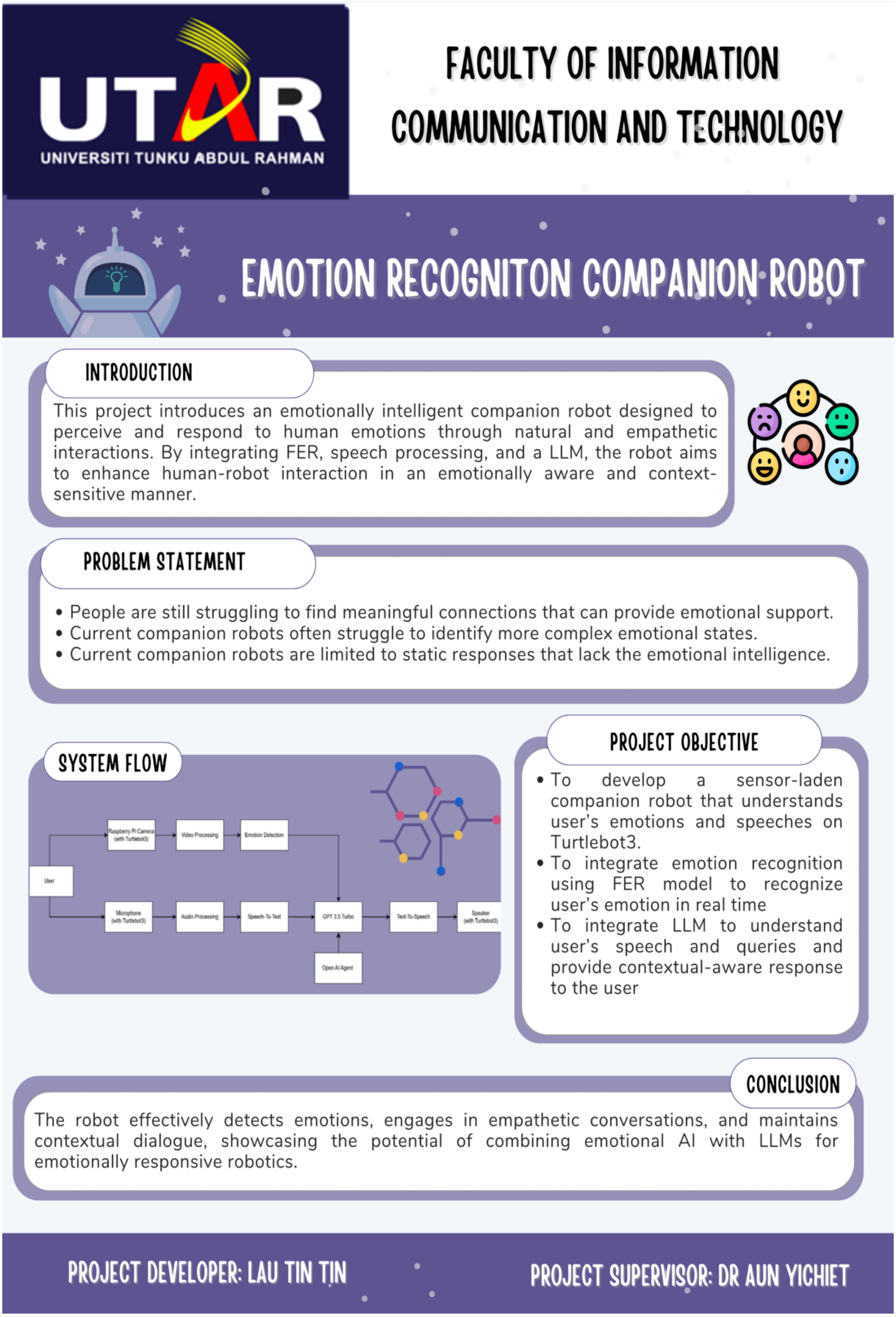
Which version of the robot would you prefer to talk to in the future? *

- With emotion-aware
- Without emotion-aware
- No preference

上一页 提交 清除表单内容

切勿通过 Google 表单提交密码。

POSTER


UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
FACULTY OF INFORMATION COMMUNICATION AND TECHNOLOGY
EMOTION RECOGNITON COMPANION ROBOT
INTRODUCTION
This project introduces an emotionally intelligent companion robot designed to perceive and respond to human emotions through natural and empathetic interactions. By integrating FER, speech processing, and a LLM, the robot aims to enhance human-robot interaction in an emotionally aware and context-sensitive manner.
PROBLEM STATEMENT
• People are still struggling to find meaningful connections that can provide emotional support.
• Current companion robots often struggle to identify more complex emotional states.
• Current companion robots are limited to static responses that lack the emotional intelligence.
SYSTEM FLOW
User
Raspberry Pi Camera (with Turtlebot3)
Video Processing
Emotion Detection
Microphone (with Turtlebot3)
Audio Processing
Speech-To-Text
GPT 3.5 Turbo
Open AI Agent
Text-To-Speech
Speaker (with Turtlebot3)
PROJECT OBJECTIVE
• To develop a sensor-laden companion robot that understands user's emotions and speeches on Turtlebot3.
• To integrate emotion recognition using FER model to recognize user's emotion in real time
• To integrate LLM to understand user's speech and queries and provide contextual-aware response to the user
CONCLUSION
The robot effectively detects emotions, engages in empathetic conversations, and maintains contextual dialogue, showcasing the potential of combining emotional AI with LLMs for emotionally responsive robotics.
PROJECT DEVELOPER: LAU TIN TIN
PROJECT SUPERVISOR: DR AUN YICHIET