Imperial College London

MEng 3rd Year Consultancy Project

Imperial College London

Department of Electrical and Electronic Engineering

IBM A.I K-9+

Authors:
Louise Davis
Gian-Luca Fenocchi
Pablo Romo Gonzalez
Hamed Mohammed
Conan Quinlivan

Supervisor:
Dr. Ayush Bhandari

July 7, 2023

Abstract

This documentation report presents the development of K9, an innovative quadruped companion pet designed to introduce AI technology to a broad audience, with a focus on the elderly. K9 aims to enhance the lives of users by providing interactive engagement, encouraging positive behavior, promoting physical exercise, and offering entertainment and information. Through a comprehensive development process encompassing research, conceptualization, design, prototyping, and software implementation, K9 has emerged as a sophisticated companion pet with features such as voice command response, music playback, daily news updates, and support for healthy exercise habits.

This project was conducted in collaboration with IBM, leveraging their expertise in AI technologies and industry insights. The partnership with IBM has greatly contributed to the project's success, providing valuable resources, guidance, and opportunities to engage with IBM's research and development teams. The incorporation of AI technology in K9 addresses the social and cognitive needs of the elderly, offering them a familiar and accessible format to improve their overall well-being.

This documentation report provides an overview of the development process and current capabilities of K9 while highlighting future possibilities for further enhancements. The project showcases the potential of AI-integrated companion technology and its positive impact on the elderly population.

Acknowledgements

We would like to express our sincere gratitude to our project supervisors, Dr. Ayush Bhandari and Jon McNamara, for their invaluable guidance, support, and expertise throughout the development of the K9 project. Their continuous encouragement, insightful feedback, and unwavering commitment to our success have been instrumental in shaping the project's direction and outcomes. Their vast knowledge and experience in the field have been an invaluable resource, and we are truly grateful for their mentorship.

Contents

Chapter 1

Introduction

1.1 Initial Specification

Figure 1.1: K9

The aim of this project is to create an articulated companion pet for the elderly. This pet will also:

- Respond to a limited number of voice commands
- Encourage healthy exercise such as walking
- Allow the pet to find podcasts the user might like and play them
- Allow the pet to read the daily news and encourage positive behaviors
- Encourage a set of positive behaviors (Gratitude meditation, exercise, reading/listening to educational material, etc.)
- Play music on request

Our design for K9 was inspired by a combination of existing designs and models that we researched and studied. We drew inspiration from various sources and incorporated design elements and ideas that we found to be innovative and effective. Some of the design inspirations include:

https://www.instructables.com/DogBot-V2-Make-Your-Own-Quadruped-Robot-From-Scrat/

https://grabcad.com/library/quadruped-robot-w-code-1/

1.2 Materials

Electronics

- Raspberry Pi 4 (4GB)
- MG996R Servo Motor (x10)
- SG90 Servo Motor (x2)
- 16-Channel PWM Controller
- Stereo Speaker Set
- Adafruit Speaker Bonnet
- 3A Buck Converter
- 2A Buck Converter
- 18650 Li-ion Battery (x4)
- Battery Management System
- 1.3" OLED Display Screen
- Full HD Webcam
- Microphone
- XBox Game Controller
- TTL USB Serial converter cable (debugging)
- 9V Power Delivery USB-C Charging Port

Other Hardware & Supplies

- Screws and nuts
- PLA Filament

Tools

- 3D Printer
- Laser-cutter
- Screwdriver set
- Soldering iron

Python APIs

- IBM Watson Assistant
- IIEleven Labs TTS
- Spotify API
- NewsData API
- OpenWeatherMap API

- Spoonacular Recipe API
- ThingSpeak (smart home automation) API

1.3 Project Management and Meetings

Our team employed an iterative development approach with continuous improvement throughout the management of the K9 project. During initial development, the team prioritised delivering a functional robot. Subsequently we aimed to iteratively refine individual components of the robot. We designed a marketing leaflet and technical poster using the same method, creating an early first draft and constantly working on it. This strategy was particularly well-suited to our objectives, allowing for a systematic progression through different stages. This method also provided us with a clear and flexible working structure with objectives and milestones to work towards.

While the iterative model served as a roadmap, it was the core values of our team - flexibility and constant communication - that truly drove our project's success. Creating an environment that welcomed ideas and feedback from both team members and departmental peers at every stage of the project was key to our creative and iterative development process. As this ensured user inputs were recognised and taken into account, and there was always a constant source of inspiration.

We utilised *WhatsApp* for everyday communication and quick updates, ensuring all members were always aligned on the project's status. Face-to-face meetings were held regularly on campus, providing opportunities for brainstorming and decision-making discussions.

In addition to our internal meetings, we had regular interactions with our two supervisors. These sessions were conducted both in-person and through Microsoft Teams, depending on the need and convenience. Our supervisors' guidance and insights were instrumental in tackling challenges and refining our design approaches.

In retrospect, a continuous iterative approach with a flexible and communicative environment proved successful. It allowed us to efficiently manage tasks, meet deadlines, and maintain a high level of motivation, creativity and engagement within the team. Through this experience, we've gained valuable insights into project management, teamwork, and effective communication.

1.4 Design Thinking

Our journey in design thinking was influenced significantly by the Enterprise Design Thinking principles we learned through the IBM EDT course recommended by John McNamara. The principles from this program framed our understanding of the problem and solution development, specifically in the following ways:

Empathise

We began by seeking to understand our users deeply - the elderly. The Enterprise Design Thinking principles guided us to research the social and cognitive challenges which the elderly face through online resources. This empathic approach gave us profound insights into their needs, motivations, and the challenges they face in interacting with technology.

Define

We synthesised our observations into a meaningful and actionable problem statement, underlining the social and cognitive needs of the elderly. IBM's methodology helped us define these user needs and insights clearly, creating a strong foundation for our subsequent ideation process.

Ideate

Next, we entered the ideation stage where we brainstormed innovative solutions that K9 could offer. The principles we learned from the Enterprise Design Thinking badge program taught us how to facilitate effective brainstorming sessions, encouraging divergent thinking and nurturing a multitude of ideas before narrowing down our choices.

Prototype

Due to the scope of project K9, no prototypes were built since K9 itself could be considered a prototype for a larger project. Alternatively, we planned to have developed a functioning robot which meets the initial project specifications as early on as possible. As we progressed, any individual components of the system could be adjusted accordingly without the need to restart a whole new prototype. Our design process involved separating the Watson Assistant chatbot and movement code until the later stages of development. This decision proved effective as it enabled independent testing of the two layers with ease.

Testing

Once a functioning robot was built, we began testing K9 in our department labs. We regularly showcased K9's operations to our departmental peers, actively seeking their feedback, which significantly influenced our design process. We also held a product demonstration on the 29th of June during Imperial College's Open day, highlighting the project's features and capabilities and providing a feedback form to gather input and address questions. This iterative approach allowed us to continuously enhance K9 by incorporating new features and refining user interaction.

In essence, the Enterprise Design Thinking principles have been crucial to K9's development, enabling us to ensure that K9 was not only technologically proficient but also driven by a deep understanding of users' needs and empathetic considerations.

1.5 Main Specification

The K9 project is not only a home assistant for the elderly but also a versatile robotic companion with a wide range of possible applications. We feel that the project specification below illustrates the K9 project more accurately.

1.5.1 Project Overview

The K9 project aims to develop an innovative quadruped companion pet which uses AI for collaborative human-robot interactions. Primarily K9 serves as an interactive and engaging companion, designed to enhance the lives of users by promoting positive behavior, physical exercise, and mental stimulation. K9 caters to a broad audience, including the elderly, providing them with a familiar and accessible format to improve their overall well-being.

Secondarily, K9 also has potential for the following uses: K9 can serve as an educational toy and tool. Users can program and expand on the K9 project in an enjoyable way, possibly designing and implementing additional functionalities and customisations. K9's integration of artificial intelligence makes it also an ideal platform for exploring human-robot interaction. Researchers can leverage K9's capabilities to study emotional companion features, natural language processing, and the social dynamics between humans and robots.

1.5.2 Project Scope:

Requirements

-K9 should provide emotional support and companionship to enhance the well-being of the elderly.

-Utilise AI to interpret and respond accurately to voice commands.

-Integrate APIs for music streaming, news updates, and other interactive functionalities.

-Incorporate Multiple Ways to Sense and Interact with the Environment. E.g. Speakers, Microphone, Motion.

-Continuous Improvement- conduct group meetings to discuss feedback and identify areas to refine.

Constraints

The following limitations may impact the development of the project and its design:

-Time and budget: The project timeline and budget is not large enough to realistically develop a full product from scratch, and thus its assumed to be more for proof of concept purposes.

-Physical Constraints: K9's size and physical features may impose limitations on it being able to navigate all environments. It is primarily intended to navigate living spaces.

-Technical Constraints: K9's AI capabilities, including emotional recognition and voice command response, are also influenced by the constraints of the project.

Chapter 2

Project History

2.1 Design History

Initially, we explored a CAD design for the robot based on the instructions provided in the Instructables tutorial titled "DogBot V2 - Make Your Own Quadruped Robot From Scratch:"

https://www.instructables.com/DogBot-V2-Make-Your-Own-Quadruped-Robot-From-Scratch/

However, upon further evaluation, we determined that this CAD design posed challenges in terms of the printing process and overall bill of materials. As a result, we opted to search for an alternative design that better suited our needs. We discovered a suitable CAD model on GrabCAD:

https://grabcad.com/library/quadruped-robot-w-code-1/

This became the reference for our final product. We made several modifications to this design to accommodate the integration of all the necessary components, such as speakers. Additionally, we incorporated a camera module mounted on a custom designed head, which provides added mobility and functionality. These modifications enabled us to create a well-adapted physical design that meets our project requirements and enhances the overall capabilities of K9.

2.2 Build History

Initially, the physical assembly process faced challenges as not all the components were readily available. However, once we obtained all the necessary parts and components, the process of assembling them together became much smoother and more efficient. One notable complication arose when we needed to incorporate an additional DC buck converter to establish a connection between the battery and the Raspberry Pi. Due to space constraints inside the robot's body, we had to address this issue by carefully stripping and reconnecting wires to create enough space for accommodating all the components. On the software side, development began at the inception of the project and progressed iteratively throughout the entire build. We built upon each section, incorporating new features and functionalities to enhance the overall capabilities of K9.

2.3 Meeting History

We conducted both online and in-person meetings with Jon McNamara and Dr. Ayush Bhandari to ensure effective collaboration and guidance throughout the project. Within our team, recognizing the project's combined hardware and software nature, we maintained regular communication and discussion on a daily basis. We leveraged tools like *WhatsApp* to stay connected and foster continuous collaboration among team members.

Meetings with Jon McNamara:

Meeting - 11:30 on Tuesday 9th May 2023

Meeting - 11:30 on Friday 19th May 2023

Meeting - 11:30 on Friday 26th May 2023

Group Trip to IBM London - 12:00 - 17:00 on Friday 2nd June 2023

2.4 Gantt Chart

To effectively manage the project timeline and tasks, we utilised a Gantt chart as a visual planning tool. The Gantt chart provided a clear overview of the project's schedule, milestones, and dependencies. It helped us allocate resources, track progress, and ensure timely completion of project deliverables. By organizing the tasks into distinct phases and assigning responsibilities, we were able to effectively coordinate the efforts of the team members. The Gantt chart served as a reference point throughout the project, allowing us to monitor progress, identify potential bottlenecks, and make adjustments to the schedule when necessary. Its visual representation provided a valuable framework for project management and played a crucial role in ensuring the successful execution of the project.

Please find in Appendix D, the gantt chart.

Chapter 3

Hardware

The hardware components of K9 play a crucial role in bringing its functionality to life. Careful selection and integration of various hardware elements have been employed to ensure optimal performance and a seamless user experience. From the mechanical structure to electronic components, each aspect has been meticulously designed and implemented to create a reliable and interactive companion pet. In this section, we will discuss the key hardware features of K9, highlighting their significance in enabling its capabilities and enhancing the overall user interaction.

3.1 Electronic Components

The electronic components of K9 play a crucial role in its functionality and capabilities. They form the backbone of its operation, enabling communication, control, and sensory input. From the powerful Raspberry Pi 4 as the central computing unit to the precise MG96R servos for motion control, each component has been carefully chosen and integrated to bring K9 to life.

- The **Raspberry Pi 4** serves as the central processing unit for K9, providing computational power and serving as the brain of the companion pet. With its robust processing capabilities and GPIO pins, the Raspberry Pi 4 enables seamless integration and control of various components.

- The **MG996R servos** play a crucial role in K9's leg movements. With a total of ten servos, two for each back leg and three for each front leg (hip rotation), they enable precise and coordinated leg motions. These servos are medium-high torque and provide the necessary power to support K9's full weight.

- In order to control the servos, a **16-channel PWM servo controller** is utilised. This controller communicates with the Raspberry Pi through the I2C interface, allowing for smooth and synchronised servo control. The I2C communication protocol enables efficient and reliable servo operation, ensuring precise and coordinated movements.

- **Voltage regulators** are incorporated to provide stable and regulated power supply to the various components of K9. These regulators ensure that each component receives the appropriate voltage level, preventing voltage fluctuations and optimising performance.

- Additionally, K9 features two smaller **SG90 servos** for pan-tilt functionality in the head. These servos enable controlled movement of the head, allowing K9 to track objects, scan its surroundings, and engage with users more effectively.

The integration of these electrical components forms the foundation of K9's physical movements, ensuring precise control and synchronised actions. The combination of the Raspberry Pi, PWM servo controller, voltage regulators, and various servos enables the implementation of sophisticated movement and interactive capabilities in K9, making it a dynamic and engaging companion pet.

3.2 Sensory Input

The incorporation of a microphone and webcam in the head of K9 brings a significant level of interactivity and perception to the companion pet. These carefully chosen sensors play a crucial role in enhancing K9's functionality and responsiveness. A **Full HD USB webcam** was dismantled and integrated into the head of K9.

The **microphone** enables K9 to receive audio input and process voice commands from users. By integrating voice recognition technology, K9 can understand and respond to spoken instructions, creating a natural and intuitive interaction between the user and the pet. This feature allows users to engage in conversations, give commands, and receive audio feedback from K9, fostering a sense of companionship.

The webcam, on the other hand, provides visual input, allowing K9 to "see" its surroundings and respond accordingly. With the ability to capture images and video, K9 can perceive objects, detect movements, and even recognise faces. This visual perception enhances the pet's interactive capabilities, enabling it to react to visual cues and track objects or individuals.

The design choice of mounting the microphone and webcam on the head of K9 is intentional. Placing these sensors in a central position ensures a wide field of view and optimal audio reception.

3.3 Display and Feedback

In addition to the microphone and webcam, K9 is equipped with a **1.3-inch OLED I2C display** and **speakers**, further enriching the interactive experience with visual feedback and audio output.

The OLED screen in K9 serves a dual purpose, providing both a visually engaging mouth animation when K9 speaks and essential information regarding the Raspberry Pi's IP address (for development). The speaking animation adds a charming and lifelike element to K9's interactions, allowing the companion pet to visually express its communication.

Alongside the visual display, the integrated speakers enable K9 to deliver audio output, including spoken responses, sound effects, and music playback. The speakers provide crisp and clear audio quality, ensuring that the pet's vocalisations and audio feedback are audible and engaging. By leveraging sound, K9 can communicate effectively, express emotions, play music, and even provide auditory cues or notifications.

3.4 Power System

The power system of K9 consists of a USB-C input for charging, capable of delivering quick charge with 9V input and a current of 3A. To ensure efficient power distribution, a custom battery was developed.

The power requirements were carefully calculated, considering the Raspberry Pi's maximum current draw of 2A, the servos' maximum requirement of 5A, and an additional 1A for peripherals. Based on these calculations, a configuration of **4 Lithium-ion 18650** cells was chosen, connected in 2 series and 2 parallel, resulting in a total voltage of 7.4V. These cells, rated for 20A continuous discharge, provide the necessary power for the system.

To ensure secure and reliable connections between the lithium-ion battery cells in K9, a spot welder was used. Spot welding is a commonly used technique for joining battery cells, providing strong and low-resistance connections. This method ensures the cells are securely interconnected, minimising the risk of loose connections and potential hazards. By carefully controlling the welding parameters, such as welding time and current, precise and consistent welds were achieved, enhancing the overall safety and performance of the battery pack.

Given that lithium-ion batteries can pose a fire hazard if mishandled or improperly charged, a thorough risk assessment was previously conducted to mitigate any potential risks. The assessment involved evaluating factors such as proper insulation, protection against short circuits, and appropriate charging protocols.

To facilitate safe charging, a balanced battery management system was incorporated to ensure equal and secure charging of each cell. A **voltage regulator** was utilised to step down the charger voltage to the appropriate 8.4V charge voltage. Additionally, a diode was employed to prevent reverse current flow from the battery through the voltage regulator, as the input and output terminals are shared. **XT30 battery connectors** were used to allow convenient removal of the battery.

Moving on to powering the components, it was initially attempted to use a single voltage regulator for both the servos and the Raspberry Pi. However, under full load conditions, current spikes caused the Raspberry Pi to experience shutdowns and reboots. To address this issue, two separate voltage regulators were implemented. A dedicated 6V 3A regulator was employed to power the servos, while a separate 5V 2A regulator was dedicated to supplying power to the Raspberry Pi. This configuration ensured stable power delivery to both components, eliminating the previous instability observed during high servo load situations.

To conclude, by employing the spot welding technique for cell connections and conducting a comprehensive risk assessment, the battery system in K9 was designed and implemented with safety as a top priority, whilst meeting power requirements.

3.5 CAD Design

The CAD (Computer-Aided Design) phase played a crucial role in the development of K9, enabling the creation of precise and detailed digital models of the robot's mechanical structure. This section highlights the research-driven approach, the utilisation of 3D printing technology, and the use of *SOLIDWORKS* software in the design process.

Before diving into the CAD design, we conducted thorough research on current solutions and examined numerous models available in the market. This comprehensive exploration provided valuable insights into different design approaches, functionalities and user feedback. By analysing and understanding these existing models, the team identified key design ideas that aligned with the specific requirements of K9.

Based on the research findings, we embarked on a creative process that involved combining and integrating design elements from multiple models. This approach allowed for the development of a unique and customised design that addressed the specific needs and objectives of K9. By leveraging the best features from different models, we were able to create a design that offered a balance between innovation, functionality, and user experience.

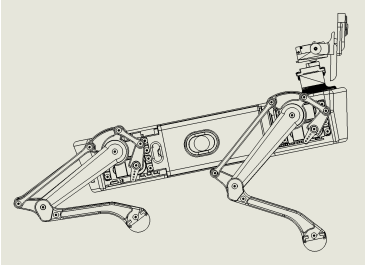


Figure 3.1: Side Profile

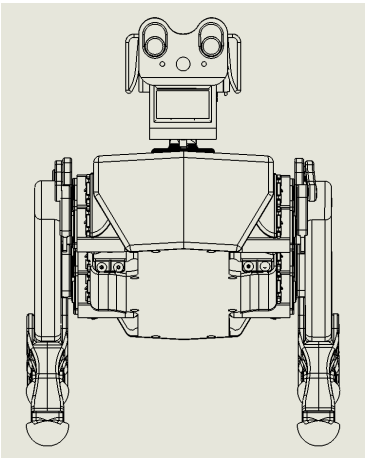


Figure 3.2: Front Profile

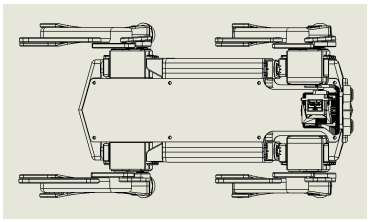


Figure 3.3: Top Profile

The implementation of the CAD design relied heavily on 3D printing technology. With the help of a 3D printer, the design team materialised the virtual CAD models into physical prototypes. This process enabled us to evaluate the design's fit, form, and functionality in a tangible manner. It also facilitated the iterative refinement of the design, allowing for adjustments and improvements

to be made based on practical considerations and user feedback. The utilisation of 3D printing technology offered significant advantages in terms of speed, flexibility, and cost-effectiveness. It allowed for rapid prototyping, enabling the design team to quickly iterate and test different design iterations.

3.6 Construction Material

During the construction of K9, careful consideration was given to selecting appropriate materials for different components based on specific requirements. The choice of materials played a vital role in ensuring the durability, weight, cost-effectiveness, and overall aesthetics of the robot.

For the top and bottom sections of the body, black acrylic sheets with a thickness of 3mm were chosen. Black acrylic not only offered the desired strength for structural integrity but also added a sleek and sophisticated appearance to K9's design. The use of black acrylic complemented the overall aesthetic theme and enhanced the visual appeal of the robot.

In contrast, the majority of other components were 3D printed using PLA (Polylactic Acid) plastic. PLA is a popular thermoplastic material known for its ease of use in 3D printing, affordability, and biodegradability. PLA provided sufficient strength and rigidity for various structural parts of K9, such as the legs, brackets, head and side body. The versatility of PLA allowed for precise customisation and rapid prototyping, facilitating iterative design improvements throughout the development process.

In the construction of K9, a consistent approach was taken for securing components, employing machine head M3 screws throughout the assembly process. The use of M3 screws offered a reliable and versatile fastening solution, ensuring sturdy connections between different parts of the robot. To enhance the integrity of the fastened joints, *Loctite threadlocker* was applied to prevent loosening due to vibrations or external forces. The modular design of K9 allows for convenient access to individual components, enabling quick and straightforward repairs or replacements when necessary.

Chapter 4

Software

The software section of our project is built upon Python, a versatile and powerful programming language. Python serves as the backbone of K9's software architecture, allowing us to seamlessly integrate various modules and APIs to enhance its functionality. With Python's extensive libraries and frameworks, we were able to implement features such as music streaming, voice recognition, and information retrieval effortlessly. The flexibility and ease of use provided by Python have been instrumental in enabling K9 to deliver a seamless and interactive user experience.

4.1 User Interface

This section provides an overview of the methods used for both speech recognition and text-to-speech capabilities in the development of K9.

4.1.1 Speech Recognition

We implemented a speech recognition system using the `SpeechRecognition` library in Python. This allows K9 to listen to user input and convert it into text for further processing. We used the Microphone class provided by `SpeechRecognition` to access the audio input device. By selecting the appropriate audio input device and using the `recognize_google` function, we were able to capture the user's speech and convert it into text format. This text could then be used for various purposes such as responding to user commands or performing specific actions based on the recognised speech.

To further enhance the speech recognition capabilities of K9, we integrated a wake word detection system using *Picovoice* technology. Picovoice provides an efficient and accurate wake word detection engine that allows K9 to activate and start listening for user commands only when "Hey K9" is detected. At its core, Picovoice is built upon deep learning models to process audio input and recognise wake words quickly and efficiently. The integration of the Picovoice Wakeword Engine significantly improves the user experience by providing a more natural and intuitive way of initiating voice commands, making K9 even more responsive and user-friendly.

4.1.2 Text-to-Speech

In order to provide K9 with the ability to speak, we implemented a text-to-speech (TTS) functionality. The TTS module allows K9 to convert text-based input into natural-sounding speech. Our implementation relied on the industry-leading *ElevenLabs API*, which offers a wide range of high-quality voices for generating customised natural speech.

The `speak` function serves as the core component for text-to-speech conversion. When called, it takes a text input as a parameter and proceeds to generate the corresponding audio file. The text is passed to the ElevenLabs API, which utilises deep learning models to synthesise the speech. The generated audio is then saved as an MP3 file and played using the `play_sound` function.

By incorporating text-to-speech capabilities, K9 gains the ability to provide audible responses, engage in natural conversations, and effectively communicate with users. This feature greatly enhances the interactivity and user-friendliness of K9.

4.2 IBM Watson Assistant

The following section provides an overview of the powerful voice command and assistant capabilities integrated into our K9 system. By leveraging the IBM Watson Assistant service, all voice commands from users are seamlessly passed to the assistant, which utilizes its advanced natural language processing capabilities to accurately detect the intent behind each request.

Once the intent is determined, the Watson assistant then triggers the corresponding function through a well-defined intent mapping. This enables our assistant to perform various tasks based on user requests, such as retrieving weather information, getting the current day, playing music, fetching news articles, suggesting activities, providing jokes and several more.

On the IBM Cloud, an instance of a Watson Assistant has been set up. The IBM Cloud Watson Assistant platform lets you define multiple actions, each corresponding to a user intent, by providing example sentences from the user that correspond to that intent. The more examples provided the better, as the Watson Assistant performs Natural Language Processing techniques to train a Machine Learning model. This machine learning model is capable of mapping phrases different to the examples provided to the correct intent if the intent of the phrase is the same, and can also map phrases with typos to the right intent.

The Watson Assistant cloud platform provides a draft environment to test the chat-bot, as well as an API key, a draft and live environment URL, and an assistant URL to use as credentials. These credentials are then used in our code to perform API calls to our Watson Service and make use of the assistant's intent detection capabilities to perform the correct subsequent function.

In Appendix A, you may find the complete set of functions that can be called by the Watson Assistant. They are the top-level functions that call other auxiliary functions that are required to perform user requests.

4.2.1 WatsonAssistant Class

The **WatsonAssistant** Class is a key component of the system and serves as the central hub for controlling the K9 assistant functionality. It is the interface between the service that interprets voice commands, and robot's functionalities to handle such commands. This class encapsulates all of the functionality required to communicate with the Watson Assistant API and execute actions based on user intents.

The class constructor takes several parameters, including the API key, assistant ID, service URL, and a file path for intents. These parameters are used to authenticate and establish a connection with the IBM Watson Assistant service. Upon initialization, the **WatsonAssistant** class sets up the IBM Watson Assistant SDK by creating an instance of **AssistantV2** and configuring it with the provided API key and service URL. It also creates a session with the assistant using the provided assistant ID, obtaining a session ID for subsequent interactions.

The **WatsonAssistant** class defines a mapping between user intents and corresponding functions. This intent mapping allows the system to determine the user's intention based on the response from the Watson Assistant service and execute the appropriate function accordingly.

watson_chat()

The `watson_chat` method is responsible for processing user speech input and initiating a conversation with the Watson Assistant. It sends the user input to the Watson Assistant API, retrieves the response, and extracts the action intent from the assistant's JSON reply. The method also considers the detected intents and triggers the corresponding function based on the intent mapping. If no intents are detected or if the assistant's reply is unavailable, a fallback action is taken.

read_intents_from_csv()

The class includes a helper method, `read_intents_from_csv`, which reads intents ID's and their corresponding text from a CSV file and returns them as a dictionary. The intent text is then utilised the call the appropriate function corresponding to the detected intent.

4.3 Emotional Companion

The `'Emotional Companion'` feature is a cornerstone of the K9 project. It encompasses functionalities such as suggesting activities to the user (`'get_activity'`), offering mental support when the user exhibits signs of distress (`'mental_support'`), and reciprocating affection when the user conveys feelings of love (`'love_you'`).

get_activity()

The `'get_activity'` function is invoked when the user asks for an activity or something positive to engage in. This function randomly selects an activity from categories including exercise, gratitude, learning, and reading, stored in relevant CSV files under `'assets/behaviours'`. It retrieves the current weather conditions for a preset location, then scans through the CSV file to find suitable activities for the current forecast and temperature. The selected activity is voiced to the user, and the AI K9 offers the possibility of suggesting another activity. The user has a maximum of three attempts to accept an additional recommendation.

mental_support()

The `'mental_support'` function is triggered when the user communicates feelings of sadness or distress to the AI K9. In response, the function selects and voices a supportive message from the `'Mental_Support.txt'` file, found in `'assets/behaviours'`. If the user's prompt contains specific keywords or phrases related to depression or suicide, the function provides the details of the Samaritans helpline.

love_you()

The `'love_you'` function is activated when the user expresses affection towards the AI K9. The function selects a random response that reciprocates the expression of love from the `'Love_You.txt'` file in `'assets/behaviours'`, and communicates this to the user.

The `'Emotional Companion'` feature significantly contributes to the AI K9's capability to emotionally interact with the user. By providing empathetic responses and promoting positive behaviours such as exercise, reading, meditation, and expressing gratitude, it offers a meaningful and fulfilling user experience.

4.4 Movement

This section provides an in-depth overview of the code responsible for controlling K9's movements. The ability to precisely control the legs of a quadruped robot is crucial for achieving stable and coordinated movement in various environments. Our approach focuses on utilising inverse kinematics calculations and Bézier curves to generate smooth trajectories, ensuring accurate leg positioning and efficient gait patterns.

To control the 10 *MGR95 servos* using I2C communication, we utilise the `Adafruit ServoKit library`. The ServoKit library provides an interface for controlling multiple servo motors simultaneously through a single I2C bus connection.

4.4.1 Quadruped Class

The `Quadruped` class serves as the central component for controlling the K9. It encapsulates the functionality and properties required to manage the robot's movements and interactions. The class utilises servo motors, inverse kinematics algorithms, and other control mechanisms to achieve precise leg movements and coordinated locomotion.

The class constructor, `__init__(self)`, initialises the object and sets up the necessary components. It instantiates the `ServoKit` and defines the lengths of the upper and lower legs. Additionally, it configures the pulse width range for each servo motor.

The class provides various methods for controlling the quadruped's movements. These include functions such as: `set_angle(self, motor_id, degrees)`, `calibrate(self)`, `inverse_positioning(shoulder, elbow, x, y, z=0)` and `leg_position(leg_id, x, y, z=0)`.

The 'Quadruped' class provides a robust foundation for controlling the robotic quadruped's movements, enabling precise and coordinated locomotion in various scenarios.

4.4.2 Core Functions

`set_angle(motor_id, degrees)` This function sets the angle of a specified servo motor. It takes the motor ID and the desired angle in degrees as input.

`inverse_positioning(shoulder, elbow, x, y, z=0)` The `inverse_positioning()` function performs inverse kinematics calculations for a leg, given the shoulder and elbow servo motors, and the desired x, y, and z coordinates. It calculates the angles required to position the leg accurately and sets the corresponding servo angles. The function also handles adjustments based on leg side and the availability of a hip servo motor.

`leg_position(leg_id, x, y, z=0)` This function moves a specific leg to a desired position using inverse kinematics. It takes the leg ID ('FL', 'FR', 'BL', or 'BR') and the x, y, and z coordinates as input.

`calibrate(self)` The `calibrate()` function sets K9 into its default resting position by adjusting the angles of the shoulder, elbow, and hip servo motors. This allows the leg assembly to be mounted at the correct positions during construction.

4.4.3 Game Controller Input

In addition to its autonomous capabilities, K9 features a game controller integration that allows for manual control of the robot at any time. By connecting a compatible game controller to the system, users have the ability to take direct control of K9's movements and actions. This feature offers a more interactive and immersive experience, allowing users to navigate K9 through their environment, perform specific tasks, or engage in playful interactions. The game controller provides intuitive and responsive control inputs, empowering users to guide K9 with precision and ease. Whether it's exploring new spaces, performing tricks, or simply enjoying the experience of controlling a robot companion, the game controller integration adds a dynamic element to K9's functionality, giving users the flexibility to switch between autonomous and manual control modes as desired.

4.4.4 Inverse Kinematics

Inverse kinematics is a computational technique used to determine the joint angles required to position an end-effector (in this case, a leg) at a desired coordinate in space. In the context of the robotic quadruped, inverse kinematics is employed to calculate the servo angles necessary to achieve a specific leg position.

The inverse kinematics process begins by determining the orientation of the leg in 3D space. Given the desired coordinates, the algorithm calculates the angles required to position the leg accurately. This involves solving a set of trigonometric equations based on the geometry of the leg structure.

To achieve this, the algorithm leverages the lengths of the upper and lower leg segments, as well as the specified coordinates. It computes the necessary joint angles by applying trigonometric formulas such as the Law of Cosines and the Law of Sines.

Inverse Kinematics formula used for calculating the servo angles:

$$y' = -\sqrt{(z+L)^2 + y^2}$$

$$\theta_z = \text{atan2}(z+L, |y|) - \text{atan2}(L, |y'|)$$

$$c_2 = \frac{x^2 + y'^2 - a_1^2 - a_2^2}{2 \cdot a_1 \cdot a_2}$$

$$\theta_2 = \text{atan2}(s_2, c_2)$$

$$c_2 = \cos(\theta_2)$$

$$s_2 = \sin(\theta_2)$$

$$c_1 = \frac{x \cdot (a_1 + (a_2 \cdot c_2)) + y' \cdot (a_2 \cdot s_2)}{x^2 + y'^2}$$

$$s_1 = \frac{y' \cdot (a_1 + (a_2 \cdot c_2)) - x \cdot (a_2 \cdot s_2)}{x^2 + y'^2}$$

$$\theta_1 = \text{atan2}(s_1, c_1)$$

$\mathbf{y'}$: Height offset of the leg from the body.
$\mathbf{z}$: Vertical position of the leg.
$\mathbf{L}$: Constant representing the leg length offset.
$\mathbf{y}$: Horizontal position of the leg.
$\boldsymbol{\theta_z}$: Angle between the leg and horizontal plane.
$\mathbf{c2}$: Cosine of θ_2.
$\mathbf{x}$: X-coordinate of the leg.
$\mathbf{a1}$: Length of the upper leg segment.
$\mathbf{a2}$: Length of the lower leg segment.
$\mathbf{s2}$: Sine of θ_2.
$\boldsymbol{\theta_2}$: Angle between upper and lower leg segments.
$\mathbf{c1}$: Cosine of θ_1.
$\mathbf{s1}$: Sine of θ_1.
$\boldsymbol{\theta_1}$: Angle of the shoulder joint.

Additionally, the inverse kinematics function handles adjustments based on the side of the leg (right or left) and the availability of a hip servo motor. These adjustments ensure that the leg's movement aligns with the quadruped's overall locomotion.

Once the required joint angles are calculated, the function sets the corresponding servo angles using the `set_angle(motor_id, degrees)` method. This completes the inverse kinematics process, allowing the quadruped to position its legs accurately in space.

By incorporating inverse kinematics calculations, the `Quadruped` class enables precise control over leg movements, facilitating coordinated locomotion and enhancing the overall agility and maneuverability of the robotic quadruped.

4.4.5 Bézier Curve Trajectory

The Bézier curve is a mathematical curve that is commonly used in computer graphics and animation to create smooth and natural-looking curves. We used Bézier curves to generate smooth and natural motion trajectories for the quadruped's legs during walking. The control points are defined based on the desired leg motion, such as lifting and extending the leg. By manipulating the control points, you can adjust the shape and trajectory of the leg motion.

This provides a flexible and intuitive way to define and control motion trajectories, making it a valuable tool in animation, robotics, and other areas where smooth and precise motion is desired.

Here is a diagram of the curve used to control the leg movement:

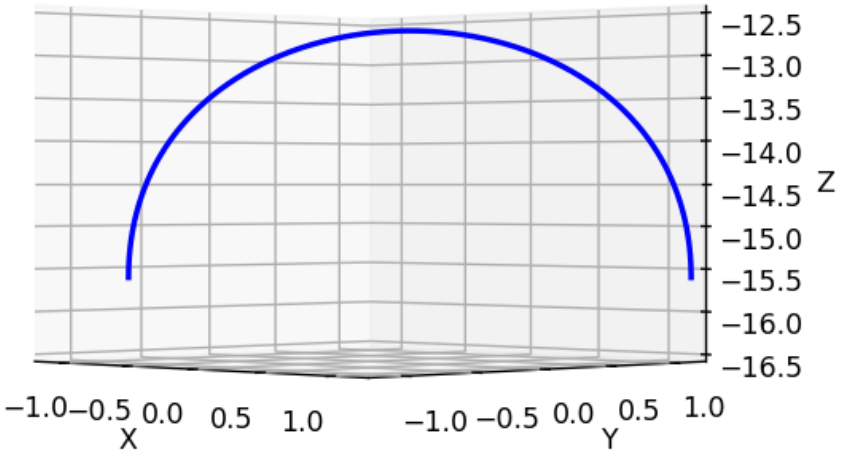


Figure 4.1: Forwards movement trajectory

4.5 Face Recognition

The implementation of facial recognition technology plays a pivotal role in our project, enabling advanced identification and verification of individuals based on their facial features. By leveraging face detection, feature extraction, and face matching algorithms, our system can accurately recognise and differentiate faces, enabling user profiles and a personalised experience.

The main steps are face detection, feature extraction, and face matching achieved through a combination of the *FaceRec* class and three key functions: `'add_face()'`, `'scan_face()'`, and `'greet_me()'`. The `face_recognition` and `cv2` Python libraries are used. The `face_recognition` library uses Histogram of Oriented Gradients (HOG) features combined with a linear classifier, an image pyramid, and sliding window detection scheme to detect faces in an image. This method scans the image at various scales and orientations to identify regions that strongly match facial features.

4.5.1 FaceRec Class

The *FaceRec* class is made up of two main methods:

- `'load_encoding_images()'`: Accepts a directory path of images and an optional save file path. Encodes faces using `face_recognition.face_encodings()` and stores the encodings with their corresponding names. `face_encodings()` uses a pre-trained convolutional neural network to extract a 128-d array of facial features (known as face embeddings) from the detected face. This neural network has been trained on a large dataset of faces and learned to extract these features that capture aspects of the face like the shape of facial components, the texture of the skin, etc. These face encodings are essentially the numerical representation of the detected face, with each number representing a different facial feature. If a save file path is provided and the file exists, it loads the encodings and names directly from this file. If the file does not exist, it creates and stores the encodings in this file.

- `'detect_known_faces()'`: Accepts a video frame as input. With the faces encoded into numerical vectors, these can then be compared for similarity. This is done using the `compare_faces()` function from The `face_recognition` library , which computes the Euclidean distance between the face encoding of the input face and the encodings of known faces. If the distance is below a certain threshold, it considers the faces as a match. The `np.argmin()` function is then used to find the encoding with the shortest distance, indicating the best match among the known faces. It returns the locations of detected faces and their names if recognised.

Core Functions

add_face() Triggered by the "add my face" command. This function interacts with the user to obtain their name through voice, and captures a snapshot of the user's face. This photo is then stored and the facial features encoded for future recognition. Providing a seamless interface for adding faces to the system.

scan_face() Executed on K9 boot up. The function scans for faces until it recognises a face from its stored encodings. If a face is recognised, it returns the name(s). If no face is detected, it returns an error message.

greet_me() Triggered by the "greet me by name" command. This function works similarly to ‘“latex 'scan_face()', but instead of returning the names, it initiates a vocal greeting to the recognised faces.

In conclusion, the AI K9's facial recognition is a combination of advanced machine learning techniques that allow it to accurately detect and recognise individuals from images and video frames.

4.6 Ball Tracking

By tracking a ball, users can engage with K9, leading to activities such as fetch, encouraging an active lifestyle.

Our implementation of ball tracking utilises computer vision techniques to detect and track a green ball in a real-time video stream. By leveraging the capabilities of libraries such as OpenCV, the code processes each frame captured by a webcam, isolates the green color range using a color mask, and applies morphological operations to enhance the ball's visibility. The program detects the largest contour within the color range and calculates a minimum enclosing circle to determine the ball's position and size. Additionally, it stores the ball's tracked points and draws lines connecting consecutive points to visualise the ball's movement over time. The code checks for the position of the ball relative to the frame. Depending on the position of the ball, K9 will adjust its course to keep itself aligned.

This function processes each frame captured by the webcam. It takes a frame as input and performs various operations to track the green ball. The steps involved are:

- Resize the frame to a specified width using the `imutils.resize()` function.
- Apply Gaussian blur to the frame using the `cv2.GaussianBlur()` function with a kernel size of (11, 11).
- Convert the frame from the BGR color space to the HSV color space using `cv2.cvtColor()`.
- Create a color mask by applying a range threshold to the HSV frame using `cv2.inRange()`.
- Perform morphological operations to reduce noise and fill gaps in the mask. This involves erosion using `cv2.erode()` and dilation using `cv2.dilate()`.
- Find contours in the mask using `cv2.findContours()`.
- Determine the largest contour based on its area using `max()` and `cv2.contourArea()`.
- Calculate the minimum enclosing circle for the largest contour using `cv2.minEnclosingCircle()`.
- Compute the centroid of the circle using the moments of the contour.
- Draw the circle and centroid on the frame using `cv2.circle()` and `cv2.putText()`.
- Draw lines connecting the tracked points to visualize the movement using `cv2.line()`.

The position of the ball relative to the camera determines the direction in which K9 changes its movement. If the ball is in the left portion of the frame, K9 adjusts its direction to the left. Similarly, if the ball is in the right portion, K9 turns right. When the ball is in the middle third of the frame, K9 moves forward, and if the ball is not detected, the K9 stops and maintains its current direction.

4.7 Spotify Media Integration

The Spotify integration enhances the system's capabilities by allowing it to interact with the popular music streaming service, Spotify. This section focuses on the key functions and features involved in integrating Spotify into the system, providing an overview of its functionalities and how it enhances the user's music and podcast listening experience.

In combination of the Python module *spotipy*, *Spotify Web API* and *dbcooper's Raspotify service*, K9 can access the user's Premium Spotify account and play from the side mounted speakers.

4.7.1 Raspotify Service

Using *dbcooper's Raspotify service*, the Raspberry Pi could be deployed as a Spotify device, so that it has access to play from the speakers. It used the open source client *librespot* to enable the control of Spotify.

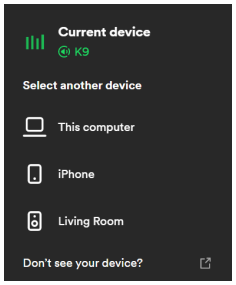


Figure 4.2: Spotify Device: K9

4.7.2 Key Functions

This section explores the functionalities responsible for handling user requests, searching for specific tracks or artists, retrieving track information, and playing the requested media.

Initialising Spotify

Utilising Spotify's *Web API for Developers*, K9 can seamlessly interact with Spotify's streaming services. By creating an app within the Spotify platform, users gain access to a unique client ID, client secret, and redirect URI, which are essential for authorising API requests. These credentials enable K9 to seamlessly integrate with Spotify's extensive music library and deliver a personalised and immersive audio experience.

Within the `initialise_spotify()` function, spotipy's `SpotifyOAuth`, a python wrapper class for the web API, passes these parameters, `client_id`, `client_secret`, `redirect_uri`, `username`, `device_name`,`scope`, creating an instance of `SpotifyOAuth`. This creates a `Spotify` instance for authenicating the details to obtaining access tokens for the interfacing of the web API.

The method `spotify.devices()` retrieves a list of all the devices connected to the Spotify instance that is currently available for playback and is filtered through to find the provided `device_name`. Once matched, the `spotify` instance and device name is returned and then globalised so all the functions within `music.py` file can use it.

Information Extraction

The users' input from the speech-to-text provides the desired commands for specific Spotify actions. While Watson Assistant has already organised these requests, the specific audio track or artist name may still be unknown. To address this, the Python library `re` is used, employing the `search()` function to identify patterns that correspond to each specific action. This allows for precise matching and retrieval of the desired audio content.

```
General Case: re.search(r'{Rules Inserted}', text, re.IGNORECASE)
```

The `text` is the input from the user and `re.IGNORECASE` is a flag used to perform a case-insensitave search.

Retrieving Media URI

Upon successful extraction of the media details, the information is then passed to a specific `Spotipy.Spotify.search()` function.

`General Case: spotify.search(q=media_title, limit={ Number of items returned}, type='{Media Type}')`

Contained within the results of the search is the URI for the playback request.

Specific Media Requests

Specific request functions are dedicated to handling *specific* use cases where the media title and other information are provided in the initial request to K9. This information is then sent to subsequent steps in the process, specifically the **Information Extraction** and **Retrieving URI** functions.

`Specific Case:` *I want to listen to a {Media Type + Details}*

General Media Requests

Furthermore, we have '*general*' request functions, for which the user provides only the type of media without any specific details. These functions use `'recognise_input(local_recogniser)'`, which prompts the user to enter the details of the media.

`General Case:` *I want to listen to a {Media Type}.*

4.7.3 Requesting Songs by Name

This function enables K9 to play a single song on the Spotify device. While providing the song name is crucial for both types of requests, including the artist name is optional and not required for playback.

`Input Request Example 1:` *"Hey K9! Play me Bohemian Rhapsody by Queen."*

`Input Request Example 2:` *"Hey K9! I want to listen to a song."*

extract_song_info()

This function is responsible for extracting song information from user requests. It utilises the search pattern `r'play(?:me)\s(.+?)(?:\sby\s(.+))?$'` to identify the desired song. The pattern first looks for the word `'play'` and optionally `'me'`. It then matches a whitespace character `'\s'` followed by `'(.+?)'`, which captures the group of characters representing the song name. If the given text contains `'by'`, it indicates that an artist has been specified as well.

extract_song_and_artist()

This function is responsible for extracting song and artist information from general requests. It uses the search pattern `r'(.+?) (?:\sby\s(.+))?$'` to identify the desired song and potential artist input. The pattern captures a set of characters using `(.+?)`, representing the song name. It also looks for the presence of `'by'` followed by the artist name.

4.7.4 Requesting Songs by Artist

This function is designed to play a the most popular songs by a specific artist on the Spotify. Unlike other functions, no song name is required as input for this particular operation.

`Input Request Example 1:` *"Hey K9! Play me songs by Queen."*

`Input Request Example 2:` *"Hey K9! I want to listen to an artist."*

extract_artist_info()

This function is responsible for extracting information about a specific artist. It uses the regular expression `r'by(.+)$'`, which focuses on identifying the keyword `'by'` and capturing any characters that follow, representing the artist name.

Since the general `request_artist` function only requires an artist, there is no need for an `re.search()`. Therefore, the recognised input is passed straight into the `get_artist_uri()`.

4.7.5 Requesting Songs from Albums

The purpose of this function is to enable playback of an album by a specific artist on the Spotify device. When searching, the most popular album is returned and played.

`Input Request Example 1:` *"Hey K9! I want to listen to the album: A Night at the Opera."*

`Input Request Example 2:` *"Hey K9! I want to listen to an album."*

extract_album_info()

This function is designed to extract information for a specific album request. It uses the pattern `r'album(?:\s(.+?))(?:\sby\s(.+))?$'` to search for the presence of the keyword `'album'`. It then captures the album name and, optionally, the artist name from the given text.

For the generalised `'request_album()'` function, the users' input is recognised and doesn't require any feature extraction, since only the album name is inputted.

4.7.6 Requesting Playlists

This function is responsible for playing a playlist on the Spotify device. It handles the request to initiate playback of a specific playlist and provides the necessary functionality to execute this action.

`Input Request Example 1:` *"Hey K9! I want to the playlist: This is Queen."*

`Input Request Example 2:` *"Hey K9! I want to listen to a playlist."*

extract_playlist_info()

This function focuses on extracting information related to a playlist request. It specifically searches for the occurrence of the keyword `'playlist'` in the user input and captures whatever follows it as the playlist name. This extracted playlist name is then used to fetch the specific playlist using the web API. In contrast to the general artist request, the `request_playlist()` function only requires the singular name of the playlist from the user input, eliminating the need for an `re` match search.

4.7.7 Requesting Liked Songs

Users can access their personalised collection of liked songs and enjoy them through K9's playback functionality. This feature provides a seamless way to rediscover and re-listen to favorite tracks, allowing for a more tailored and enjoyable music experience.

`Input Request Example:` *"Hey K9! Play the music I like."*

get_liked_songs

This function utilises the Spotipy.Spotify method `'current_user_saved_tracks()'` iterates fifty at a time, through every song within the liked songs playlist in the library. The results of the Spotify function are then shuffled and sliced to only one hundred.

play_liked_songs()

This is similar to the previously mentioned playback functions, with the exception that multiple URIs are passed as a parameter, rather than a singular URI.

4.7.8 Requesting Podcasts

With the ability to access and play podcasts, users can stay informed, entertained, and engaged with their favorite podcast shows. By simply requesting a specific podcast or genre, K9 can fetch and play the desired podcast episodes, allowing users to enjoy a wide range of audio content on the go.

`Input Request Example 1:` *"Hey K9! I want to listen to a podcast."*

`Input Request Example 2:` *"Hey K9! I want to listen to a true crime podcast."*

request_podcast()

This function prompts the user to enter the podcast name they wish to listen to after they request a podcast.

request_specific_podcast()

This function allows the user to initially specify what type of podcast they want to listen.

extract_podcast_info()

This is similar to the previously mentioned extraction functions. The regex has been changed to extract the genre of podcast from the users' input.

4.7.9 Volume Control

K9's Spotify integration includes voice commands for volume control, offering users convenient and hands-free control over the playback volume. By simply using voice prompts such as "increase volume" or "decrease volume," users can adjust the audio output to their desired level.

`Input Request Example 1:` *"Hey K9! Set the Volume to 70%."*

`Input Request Example 2:` *"Hey K9! Decrease the Volume."*

These functions change the volume in multiple ways, either setting the exact volume level or increasing and decreasing the volume by 10%.

get_current_volume()

Utilises the method `'spotify.current_playback()'` to retrieve Spotify's current volume from within the response, `'current_playback['device']['volume_percent']'`.

increase_vol()

This function uses `'get_current_volume()'` and increases the volume by 10%.

decrease_vol()

This function uses `'get_current_volume()'` and decreases the volume by 10%.

extract_volume_percentage()

Extracts the volume as a numerical value, using the `'re.search()'`, on the users input.

4.8 Podcast Suggestions and Feedback

This feature is designed to provide a personalised user experience. It learns and adapts to the user's evolving tastes and preferences. The K9 uses advanced language processing capabilities to not only recommend podcasts based on the user's known preferences, but also dynamically adjust these recommendations based on ongoing user feedback.

It utilises the user's historical data and listening habits stored in a CSV file. This file records podcast genres along with the user's preference ratings on a scale of 1 to 10. A higher rating implies a stronger preference for the genre. The K9 uses this data to understand the user's podcast preferences and to recommend new listening material that aligns with these preferences.

request_random_podcast()

This function is utilised when a user requests to listen to a random podcast. It fetches the user's favorite podcast genres, selects a random genre, and attempts to fetch a podcast from this genre that hasn't been listened to previously. Once a new podcast is found, it is played, and its details are saved to the user's listening history.

4.8.1 Key Feedback Functions

To process and utilise user feedback, the Podcast Suggestion System uses a series of functions tailored to capture a range of sentiments. These functions provide a robust method to translate natural language feedback into tangible changes in the user's preference ratings. Depending on the user's response, the system adjusts the rating of the corresponding podcast genre, refining the user's preferences for future recommendations.

podcast_feedback_fav()

`Input Request Example:` *"Hey K9! The last podcast was my favourite"*

Activated when the user expresses strong favoritism towards the prior podcast genre. This function adjusts the corresponding genre's rating to the maximum preference score of 9.

podcast_feedback_love()

`Input Example:` *"Hey K9! This podcast is fantastic"*

Activated when the user's feedback implies they loved the podcast. The function fetches the last played podcast's genre and rating and adjusts the corresponding genre's rating by increasing the genre's rating by 2, up to a maximum of 10.

podcast_feedback_like()

`Input Example:` *"Hey K9! I found that podcast entertaining"*

Activated when the user's feedback indicates they liked the podcast. The function fetches the last played podcast's genre and rating and adjusts the corresponding genre's rating by mildly increasing the genre's rating by 1, up to a maximum of 10.

podcast_feedback_dis()

`Input Example:` *"Hey K9! It was a boring podcast"*

Activated when the user expresses that they disliked the podcast. The function fetches the last played podcast's genre and rating and adjusts the corresponding genre's rating by decreasing the genre's rating by 1, but never below 1, ensuring that even less favored genres maintain a chance of recommendation.

podcast_feedback_stdis()

`Input Example:` *"Hey K9! The previous podcast was terrible"*

Activated when the user strongly dislikes the podcast. The function fetches the last played podcast's genre and rating and adjusts the corresponding genre's rating by decreasing the genre's rating by 2, down to a minimum of 1.

podcast_feedback_hate()

`Input Example:` *"Hey K9! I loathe this podcast"*

Activated when the user mentions that they hated or despised the podcast. This function fetches the last played podcast's genre and rating and significantly decreases the genre's rating to 2, ensuring the minimal likelihood of it being chosen again.

4.8.2 Helper Functions

init_podcast_ratings()

This function initially randomizes the user's podcast genre ratings. It reads the genre ratings from a CSV file, iterates over each genre, assigns a random score between 4 and 8 to each genre, and saves the updated ratings back to the file.

fetch_podcast_ratings()

This function fetches the user's podcast genre ratings from the CSV file and returns a dictionary containing each genre and its corresponding rating.

fav_podcast_genres()

This function fetches the top 3 favorite podcast genres of the user, based on the highest ratings. If any other genres have the same rating as the top 3, they are also fetched.

change_podcast_rating()

This function accepts a genre and a rating and verifies if the new score is within the range of 1 to 10. If the score is within range and the genre exists in the file, the rating is updated.

fetch_prev_podcast()

This function retrieves and returns the last listened-to podcast genre and its current rating from the user's listening history in the 'assets/podcast/podcast_history.csv' file.

podcast_history()

This function is called after any podcast has been played for the user and accepts the genre, podcast name, and artist. It logs the details of the listened-to podcast into a CSV file.

podcast_history_check()

This function accepts a podcast and artist's names and checks the user's previously listened podcast history for an exact match. If a match is found the function returns True, otherwise it returns False.

get_podcast_genre_uri()

This function uses a Spotify Search with the arguments set to `limit=50` and `type='show'`. Using the retrieved fifty podcasts, one is chosen at random for the playback.

This system ensures a seamless integration of user preferences and novel recommendations, providing a tailored podcast listening experience.

4.9 News

`Input Request Example 1 :` *"Hey K9! Update me on the latest news."*

`Input Request Example 2:` *"Hey K9! What is the news on technology?"*

The News Request feature allows users to easily access the latest news and event updates, eliminating the need to manually navigate news websites or newspapers.

The core functionality of the News Request feature lies in the implementation of the `get_news()` and `get_specific_news()` functions. These functions make API calls using the **NewsData.io** API to fetch news headlines from a diverse range of reliable sources. By making use of intuitive voice commands, users can ask the assistant for general updates or news on a particular topic or category, and the assistant diligently retrieves the most recent and relevant news articles.

get_news()

This function makes an API call to the **NewsData.io** API to fetch recent article headlines, and consequently reading an article if the user so desires. The API call is made with a URL that includes the API key and filters so that the headlines returned are in English, from the UK and from the BBC domain. The response data is received in JSON format, and includes several articles and their respective titles, contents, sources, subjects, authors, etc. Five article titles are enunciated, and the user is prompted for one of the following responses: repeat, exit or an article number between 1 and 5. The function recognizes the user response, and either exits, repeats the article titles or proceeds the read out the contents of the article requested by the user. The article number is extracted from the user reply using the `extract_article_number()` function, and the contents are then read using the `speak()` function.

get_specific_news()

Similarly to the `get_news()` function, this function uses the **NewsData.io** API to fetch article headlines. Before the call is made, a regex search is made to detect the prepositions "on" or "about" and extract the topic of the news requested. After this, the API call is made, using a URL constructed with the API key, a topic filter specifying the desired subject of the news, and a language (English) and country (UK) filter. The rest of the functionality is as the `get_news()` function. The article headlines are read out, the user reply is recognised and the contents of an article are read if desired by the user.

extract_article_number()

This function extracts the number of an article from a user reply. It performs a regex search to detect the presence and position of the word "number", fetches the following word and converts it to an index which is then used to access the correct entry of the dictionary containing article titles and contents.

4.10 Weather Forecast

`Input Request Example:` *"Hey K9! What's the weather today?"*

By incorporating a weather function, users can easily obtain weather updates without the need to manually check weather websites or apps. This functionality enhances the user experience and provides convenience by delivering real-time weather information directly through the system's voice output.

The weather functionality is implemented in the `get_weather` function. This function retrieves weather information based on the user's input. If the user specifies a location, it fetches the weather data for that location; otherwise, it defaults to London. The function makes an API call to **OpenWeatherMap** using an API key and retrieves the weather data for the specified location.

It extracts the weather condition and temperature from the response JSON. The weather condition represents the main weather category, such as "Clear," "Clouds," or "Rain." The temperature is rounded to the nearest whole number. Finally, the function uses the `speak` function to communicate the weather information to the user, including the location, temperature, and weather condition. All the weather data is fetched by using a simple HTTPS GET REQUEST.

4.11 Calendar

`Input Request Example:` *"Hey K9! Plan an appointment for 10th of April."*

`Input Request Example:` *"Hey K9! Fetch my calendar events.*

The Calendar API functionality is implemented within the '`gCalendar.py`', '`update.py`' and '`functions.py`' modules, employing the Google Calendar API, Python's built-in datetime and calendar modules, and regular expressions to parse and extract key information from user inputs. This handles the creation, retrieval, and speaking out of events from the user's Google Calendar. The system processes user commands, extracts necessary information from the commands, and carries out the requested action based on the command contents. Here are the key functions and helper functions involved in this process:

Key Functions

update_me()

This is the primary entry point for the calendar functionality. It's called when a user says "update me", "update me on today" or anything similar. This function gets weather information, calendar events information for the time frame specified, and the news by calling the relevant functions.

process_text()

This function is called when a user asks for their calendar events like "What do I have to do today?" or "Fetch calendar events for next week" or "Anything in my calendar for next weekend?". The function takes the user input as text and searches for keywords that indicate what time frame to fetch calendar events from. Then fetches events for specific days like today, tomorrow, the day after tomorrow, next week, next weekend, etc. If it doesn't find any matching keywords, it asks the user to repeat the command.

extract_calendar_info()

This function is invoked when the user asks the system to add an event to their calendar. It extracts event details from the user's input, like the event's name, date, and time, and creates a new calendar event using these details, using the functions above.

extract_calendar_event()

Uses regular expression patterns to parse user's input and extract the event details. It is highly flexible, accommodating different sentence structures and keyword arrangements.

extract_calendar_date_and_time()

Uses regular expressions to extract date and time information from user input. It defaults to the current date and time if no specific date or time is provided. If user arranges a meeting for 4 pm, and it is already 6 pm, the current date is the default and is incremented to consider the next available time this event could take place.

create_calendar_event_easy()

This function creates and adds a calendar event to Google Calendar. The function is part of another module and it is used by '`extract_calendar_info()`' function to create an event. The '`GoogleCalendar`' object, instantiated with the provided client secret json file, is used to create the event. If the time is specified as 'all day', the start and end times cover the entire day. If a specific time is provided, the function formats it into a datetime object and sets the event duration to one hour from the specified start time. The event is then added to the calendar, and a confirmation message is printed with the event details.

Helper Functions

fetch_day_events()

This function fetches events for a specific date from the user's calendar.

fetch_week_calendar_events()

This function accepts the start date of the week and fetches and stores calendar events for the entire week in a 2D array called `week_events`.

fetch_weekend_calendar_events()

This function accepts the start date of the weekend and fetches and stores calendar events for the weekend in a 2D array called

increment_date()

This function accepts an integer indicating the day of the week (0-6 for Monday to Sunday). When the input is 0, it calculates and returns the date for the current Monday and the start of the next week from the current date. When the input is 5, it calculates and returns the date for the current and next Saturday from the current date.

speak_events()

This function is used to communicate events from a single day to the user. It iterates over the events and speaks out the title and time of each event using the `speak` function.

speak_events_range()

This function is used to communicate multiple days' events, such as a week or a weekend, to the user. It iterates over the days and events and speaks out the details of each day's events using the `speak` function, similar to the `speak_events` function.

4.12 Recipe Retrieval

`Input Request Example 1:` *"Hey K9! I want to try something new for dinner. Any ideas?*

`Input Request Example 2:` *"Hey K9! How do I make a beef burger?*

The Recipe functionality is implemented within the '`recipes.py`' module, utilising various helper functions and external API calls. It enables users to search for recipes, retrieve instructions, and obtain random meal suggestions. The system makes use of the **Spoonacular API** and incorporates features such as extracting recipes, removing HTML tags, and handling user input for meal types. Here are the key functions and helper functions involved in this process:

Key Functions

search_recipe_by_name()

This function searches for a recipe by the specified dish name. It constructs a URL with the provided dish name and the Spoonacular API key. It calls the `get_data_from_api` function to retrieve data from the API. If the data contains recipe results, it extracts the recipe ID and retrieves the recipe instructions using the `get_recipe_instructions` function. Finally, it returns a list of dictionaries, each containing the recipe title and instructions.

get_recipe_instructions()

This function fetches the recipe instructions for a given recipe ID. It constructs a URL with the provided recipe ID and the Spoonacular API key. It calls the `get_data_from_api` function to retrieve data from the API. If the data contains instructions (as identified by the presence of "steps" in the first element), it extracts the instructions and joins them into a single string using newline characters. If no instructions are available, it returns the string "No instructions available."

get_random_recipes()

This function retrieves random recipes of the specified meal type. It constructs a URL with the provided number of recipes, meal type, and the Spoonacular API key. It calls the `get_data_from_api` function to retrieve data from the API. If the data contains recipes, it extracts the recipe title and cleans the instructions by removing any HTML tags using the `remove_html_tags` function. Finally, it returns a list of dictionaries, each containing the recipe title and cleaned instructions.

Helper Functions

get_data_from_api()

This function fetches data from an API based on the provided URL. It sends an HTTP GET request to the specified URL, including the headers defined in the `HEADERS` dictionary. If the request is successful (status code 200), it returns the response data in JSON format. Otherwise, it prints an error message and returns `None`.

get_random_meal_by_phrase()

This function generates random meal suggestions based on the provided phrase. It determines the meal type based on the phrase and calls the `get_random_recipes` function to retrieve random recipes of the specified meal type. It plays sound files to provide audio feedback to the user and interacts with the speech recognition system to process user input and provide appropriate responses.

extract_meal_name()

This function extracts the name of a meal from the prompt. It searches for a pattern that matches specific phrases related to requesting a recipe, such as "how to make," "recipe for," or "tell me how to make." If a match is found, it returns the remaining text after the matched phrase, which represents the meal name. If no match is found, it returns `None`.

search_meal()

This function searches for a recipe based on the meal name extracted from the prompt. It calls the `search_recipe_by_name` function with the extracted meal name to retrieve matching recipes. If recipes are found, it provides audio feedback with the recipe instructions. If no recipes are found, it plays a sound file to indicate that no recipes were found.

4.13 Other APIs

4.13.1 Jokes

`Input Request Example:` *"Hey K9! Tell me something funny.*

The `get_random_joke` function is designed to fetch a random joke from an API and deliver it to the user. By calling this function, the system can provide users with a lighthearted and entertaining experience. The function sends a request to a joke API and retrieves a joke, ensuring that it meets certain criteria by filtering out jokes that are considered inappropriate or offensive. The retrieved joke can be in either a two-part format (setup and punchline) or a single-part format. Once the joke is obtained, it is spoken out to the user using the speak function. This feature adds an element of humor and amusement to the system, making interactions more enjoyable for users.

4.13.2 Get Date and Time

`Input Request Example:` *"Hey K9! What day is it today?*

`Input Request Example:` *"Hey K9! What's the time?*

The `get_day` function is responsible for retrieving the current day of the week. It utilises the `datetime` module to obtain the current date and then formats it to extract the day of the week using the `%A` directive. This information is then spoken to the user using the `speak` function.

By including this function in the system, users can easily inquire about the current day without needing to manually check calendars or devices. It enhances the user experience by providing quick and accurate information on the current day, allowing users to stay informed and organised.

When fetching the time of day, **Watson Assistant** is used to return the system time. This saves unnecessary local computation.

4.13.3 Smart Home Automation

`Input Request Example:` *"Hey K9! Please turn on/off the lights.*

We have included home automation functionality , specifically for controlling lights. The `update_thingspeak` function is responsible for updating the status of the lights in the home automation system by sending an HTTP POST request to the **ThingSpeak API**.

ThingSpeak is an Internet of Things (IoT) platform that allows users to collect, analyse, and visualise data from various IoT devices. It provides an API and web-based interface for managing and accessing data streams. It uses the provided `ThingSpeak` write API key and the status value (either 1 for on or 0 for off) to create the payload data. The function handles potential errors during the request and provides feedback on the success or failure of the update.

The `light_on` function is triggered when a command to turn on the lights is received. It calls the `update_thingspeak` function with a status value of 1 to indicate that the lights should be turned on. It also plays a sound to provide an audible indication that the lights have been turned on.

Similarly, the `light_off` function is invoked when a command to turn off the lights is received. It calls the `update_thingspeak` function with a status value of 0 to indicate that the lights should be turned off. It plays a sound to indicate that the lights have been turned off.

These functions enable users to control the lights in their smart home system through voice commands, providing convenience and automation to the home environment.

Chapter 5

Ethics & Sustainability

The development of K9 is guided by a strong commitment to ethics and sustainability. We recognise the profound impact that artificial intelligence (AI) systems can have on individuals, communities, and the environment. As such, we have integrated ethical considerations and sustainable practices into every aspect of K9's design and implementation.

5.1 Ethics

Ethical considerations play a crucial role in the design and development of K9. As creators of an AI-powered companion pet, we recognise the importance of addressing ethical implications and ensuring the well-being of both users and society as a whole. Our commitment to ethical practices is reflected in several key areas:

Privacy and Data Protection

We prioritise the privacy and security of user data. K9 adheres to strict data protection measures, ensuring that any personal information collected is handled responsibly and in accordance with applicable privacy laws. All user data, including facial recognition data, is held locally on K9. In order to access the data on K9, users are required to enter a password.

Bias and Fairness

We actively mitigate bias in K9's algorithms and decision-making processes. We implement rigorous testing and validation to identify and rectify biases that may emerge. Our goal is to ensure that K9 treats all users fairly and without discrimination, regardless of factors such as race, gender, or ethnicity.

Transparency and Explainability

We strive to make K9's behavior transparent and understandable to users. This includes providing clear information about the data collected, the algorithms used, and how decisions are made. We believe that users have the right to know and understand how K9 operates, fostering trust and accountability.

5.2 Sustainability

At the heart of our development process for K9 is a strong commitment to sustainability. We recognise the urgent need to address environmental challenges and actively work to minimise the ecological footprint of our AI system.

Energy Usage

We carefully selected energy-efficient hardware components for K9, optimising power consumption without compromising performance. By minimising energy usage, we not only reduce the environmental impact but also enhance the overall efficiency of the system.

Product Life-Cycle

We strive to design K9 with longevity in mind, using durable materials and components that withstand the test of time. Additionally, we focus on repairability, allowing users to easily replace or repair components when needed. By extending the lifespan of the product, we minimise waste and contribute to a circular economy.

Recyclability

Furthermore, we promote responsible disposal practices. We encourage users to recycle electronic components and dispose of batteries in accordance with local regulations. K9 uses rechargeable batteries, so waste is minimised. Additionally, we provide guidance on eco-friendly practices, such as using energy-efficient settings and powering down the system when not in use. Through these efforts, we aim to reduce electronic waste and prevent harmful materials from entering the environment.

Materials

In our pursuit of sustainability, we made a conscious decision to utilise PLA (Polylactic Acid) plastic for various components of K9. PLA is a biodegradable and compostable thermoplastic polymer derived from renewable resources, such as corn starch or sugarcane. By opting for PLA, we reduce our reliance on traditional petroleum-based plastics and minimise the environmental impact associated with their production and disposal.

In conclusion, ethics and sustainability are integral considerations in the development of K9. By adhering to ethical principles and incorporating sustainable practices, we strive to minimise the environmental impact of the project and ensure responsible use of resources. Our commitment to sustainability extends beyond the materials and technologies we employ; it encompasses ethical considerations such as privacy, safety, and inclusivity. By approaching the design, construction, and operation of K9 with a strong ethical and sustainable foundation, we aim to create a socially responsible and environmentally conscious project that aligns with our values and contributes positively to the world.

Chapter 6

Future Improvements

As we continue to develop and refine K9, we are committed to further enhancing its sustainability and ethical practices. Our goal is to create an AI companion pet that not only provides a delightful user experience but also aligns with principles of environmental consciousness and data privacy.

6.0.1 Renewable Energy Source

One area of future improvement is the integration of renewable energy sources, such as solar power. By harnessing clean and sustainable energy, K9 can reduce its reliance on traditional power sources and operate more efficiently. This not only contributes to reducing carbon emissions but also promotes a greener and more sustainable approach.

6.0.2 Custom Behaviour

In addition, we plan to focus on increasing user interface (UI) customisability. By allowing users to personalise their interactions with K9, we aim to create a more inclusive and tailored experience. This customisation can include options for adjusting voice preferences, choosing different modes of interaction, and adapting to individual user preferences. This flexibility ensures that K9 can adapt and cater to a wide range of user needs and preferences.

6.0.3 Data Privacy

Furthermore, data protection and privacy are of paramount importance to us. As data privacy regulations continue to evolve, we are committed to complying with the latest standards and best practices. We prioritise the collection of only necessary data and implement robust security measures to safeguard user information. This includes utilising encrypted data transmission, such as the HTTPS protocol, to ensure secure communication between K9 and external systems.

6.0.4 Canine Behaviour

To further enhance K9's behavior and create a more authentic dog-like experience, we are exploring various avenues for improvement. One exciting possibility is the addition of a tail that moves and reacts to K9's interactions and surroundings. This tail, equipped with sensors and actuators, can provide expressive movements that mirror a real dog's tail wagging in response to different stimuli. By incorporating this feature, we aim to create a more engaging and lifelike interaction between users and K9. By continuously advancing K9's behavior and incorporating features that closely resemble those of a real dog, we aim to create an AI companion pet that truly captures the essence of canine companionship.

6.0.5 Input Noise Cancellation

The ability to effectively filter out background noise is crucial for enhancing the accuracy and reliability of voice recognition systems. By incorporating advanced noise cancellation algorithms, K9 will be able to better distinguish and capture the user's voice, even in noisy environments.

The integration of noise cancellation technology will significantly improve the speech recognition capabilities of K9, allowing it to accurately interpret user commands and inquiries with greater precision. This enhancement will enable a more seamless and effortless interaction between users and K9, regardless of the surrounding noise levels.

6.0.6 Upgraded Servo Motors & Power System

The integration of more powerful servos will enable K9 to handle tasks that require higher force, such as navigating rough terrain or carrying objects. With increased torque, K9 will be able to perform more complex movements and interact with its environment in a more dynamic and agile manner. Additionally, a larger battery will provide the required energy capacity to support the enhanced servo performance. This will extend K9's operational time, allowing for longer periods of interaction and exploration without the need for frequent recharging.

In conclusion, the development of K9 as an AI companion pet has been driven by our commitment to innovation, sustainability, and ethical practices. Through the integration of advanced technologies, meticulous design considerations, and continuous improvement efforts, we strive to create a robotic companion that not only mimics the behavior and characteristics of a dog but also embodies the principles of sustainability and ethical use of resources.

Chapter 7

Setup Instructions

All the code can be found on our GitHub Repository: https://github.com/SpaceBod/K9AI. In the parent directory, you will the **requirements.txt**. The required packages can be installed by running:

`pip install -r requirements.txt`

(Make sure to install Python 3.11)

Before you run the code: `python chatbot/main.py`, you must first enter all the API keys in `settings.json`.

7.1 Watson Assistant

First log in to Watson Assistant Website: https://www.ibm.com/products/watson-assistant and create a new instance. Once this has been created, go to the tab called **'actions'** in the build section, and this action page should be empty. Click on the settings cog in the top right corner. Scroll along the top tabs until **Upload/Document** and within this you can upload the file **K9-Assistant-action.json** from the K9AI GitHub Repository.

The Watson Assistant Keys can be found from both the Watson Assistant Dashboard and IBM's Cloud dashboard https://cloud.ibm.com/. From the Watson Assistant Dashboard, the `'id'` and `'service_url'` can be found under the **Assistant Settings** tab on the left bar. Navigating to the **Assistant IDs and API details** you'll find `'id'` under **Draft Environment ID**, and `'service_url'` under **Service instance URL**.

Within the IBM's Cloud Dashboard, go to the **Resource list** tab (third down on the left side). Go to **AI / Machine Learning**, which contains the **Watson Assistant-no**. This should show the **API Key** for the `'api_key'` in the file. These should all be added to the `settings.json` file:

```
"watson_assistant": {
    "api_key": "[API_KEY]",
    "id": "[ID]",
    "service_url": "[IBM Watson Assistant Instance URL]",
    "intents_file": "assets/intents.csv"
}
```

7.2 Spotify Credentials

In order for K9 to access Spotify, you are required to register as a developer at:

https://developer.spotify.com/

You need to locate your **Client ID**, **Client Secret** and **Redirect URI** from the Spotify Developer Console. Then fill in the details within `settings.json`.

```
"spotify": {
        "client_id": "[CLIENT_ID]",
        "client_secret": "[CLIENT_SECRET]",
        "redirect_uri": "http://localhost:8888/callback",
        "username": "[SPOTIFY_USERNAME]",
        "device_name": "K9",
        "scope": "user-read-private user-read-playback-state
        user-modify-playback-state user-library-read user-read-currently-playing"
}
```

7.3 Other API Credentials

For the **Porcupine** wake word detection, you must create an account at:

https://console.picovoice.ai/login

Once on the dashboard, you can acccess the API key. This must be added to the `'settings.json'` file under `'api_key'`:

```
"porcupine_wake_word": {
    "api_key": "[API_KEY]",
    "model_path": "./assets/models/PiModel.ppn"
}
```

Similarly, for **ElevenLabs** TTS, you need to create an account to access the API key. The website URL is:

https://beta.elevenlabs.io/

Make changes to the `'settings.json'` file:

```
"elevenLabsAPI": {
    "api_key": "[API_KEY]"
}
```

Within the `speak()` function, you can change voice presets and settings.

Finally, create accounts for:

OpenWeatherMap: https://home.openeathermap.org/users/sign_in

Spoonacular: https://spoonacular.com/food-api/console#Dashboard

NewsData.io: https://newsdata.io/register

ThingSpeak: https://thingspeak.com/login?skipSSOCheck=true

Then fill in the `'settings.json'` file:

```
"weatherAPI": {
    "api_key": "[API_KEY]"
},

"spoonacular": {
```

```
    "api_key": "[API_KEY]"
},

"newsdata": {
    "api_key": "[API_KEY]"
},

"thingSpeakAPI": {
    "api_key": "[READ_KEY]"
}
```

7.4 RaSpotify

To install RaSpotify for the Raspberry Pi, refer to:

https://github.com/dtcooper/raspotify

Then navigate to: `sudo nano /etc/raspotify/conf` and edit the following lines:

`LIBRESPOT_USERNAME=""`, enter your Spotify Username

`LIBRESPOT_PASSWORD=""`, for your Spotify Password.

An example of what the file should resemble can be found in Appendix B. Finally, restart the system with: `sudo systemctl restart raspotify`

7.5 Speakers

To enable the I2C connection between the Raspberry Pi and the Adafruit I2S 3W Stereo Speaker Bonnet, refer to the setup guide provided by Adafruit:

https://learn.adafruit.com/adafruit-speaker-bonnet-for-raspberry-pi

You will be instructed to edit: `sudo nano /etc/asound.conf`

Please see Appendix C for more details.

Appendix A

IBM Watson Assistant Functions

- `get_weather()` Calls upon the weather API to report the current weather.
- `get_news()` Using a News API, this provides the most recent headlines to the user.
- `get_specific_news()` Upon asking for a specific topic, the K9 robot will read out any news related to the topic.
- `add_face()` Adding a new person to the library of faces by name for facial recognition.
- `greet_me()` Using the saved faces, K9 will greet you by given name.
- `get_activity()` If the user requires advice on what to do, K9 can supply multiple different options.
- `get_random_joke()` A joke API request responds with a funny joke to increase the users mood.
- `get_random_meal_by_phrase()` If the user does not know what to make for dinner, K9 can supply multiple different suggestions from an API
- `search_meal()` If the user wants to make a particular dish but doesn't know how, K9 can read out the recipe for that specific meal
- `mental_support()` Whenever the mood is low, talk to K9 and it can give you amazing advice
- `love_you()`
- `extract_calendar_info()` When the user has a new event happening and need to add it to their calendar, K9 can add it given a specific date and time as well as whatever details that go with the event.
- `process_text()` K9 is programmed to find and read out what activities are in your calendar on a given day
- `update_me()` Find out all the relevant information for the day, including Weather, Calendar, News.
- `command_sit()` Voice command to make the dog sit whenever there is no need for movement.
- `command_stand()` To be able to get the dog to stand so it can start to move around on commmand.
- `command_track()` To make K9 follow a tennis ball to move around, which eventually would lead to following the user.
- `command_stop_track()` Once the user no longer wants K9 to follow, this command instructs the robot to stop tracking the ball.
- `light_on()` For automative home assistance, the lights can be commanded on when talking to K9.

- `light_off()` Alternatively, the lights can be switched off with a voice command to K9.
- `pause_music()` Pause the playback on the Spotify device
- `play_music()` Play the playback on the Spotify device
- `play_next()` Skip the playback to the next track on the Spotify device
- `play_previous()` Skip the playback to the previous track on the Spotify device
- `request_specific_song()` Specific request for songs
- `request_song()` General request for songs
- `request_specific_artist()` Specific request for artists
- `request_artist()` General request for artists
- `request_specific_album()` Specific request for albums
- `request_album()` General request for albums
- `request_specific_playlist()` Specific request for playlists
- `request_playlist()` General request for playlists
- `request_specific_podcast()` Specific request for podcasts
- `request_podcast()` General request for podcasts
- `request_genre_podcast()` Specific request for a podcast genre
- `request_random_podcast()` Request for a random podcast
- `podcast_feedback_fav()` Rate the most recent listened to podcast because the user thought it was perfect for them.
- `podcast_feedback_love()` Giving a love rating, means that you thought it was awesome.
- `podcast_feedback_like()` If the user liked the previous podcast, they let K9 know and the ratings for the genre will be updated.
- `podcast_feedback_dis()` Upon not particularly liking a podcast, giving a dislike rating pushes the genre lower on the scale of genre ratings.
- `podcast_feedback_stdis()` If you strongly dislike the podcast played, reducing the likelyhood of playing a similar podcast.
- `podcast_feedback_hate()` Absolutely hating the previous podcast played, knocks the genre down so it is less likely to be played again.
- `increase_vol()` This function sets the volume 10% higher than the previous setting
- `decrease_vol()` Decreasing the volume by 10% from the previous set volume
- `set_vol()` This is the function to set a the Spotify Volume to a given percentage.

Appendix B

RaSpotify Configuration File

Location: /etc/raspotify/conf

```
LIBRESPOT_QUIET=

# Automatically play similar songs when your music ends.
LIBRESPOT_AUTOPLAY=

# Disable caching of the audio data.
# Enabling audio data caching can take up a lot of space
# if you don't limit the cache size with LIBRESPOT_CACHE_SIZE_LIMIT.
# It can also wear out your Micro SD card. You have been warned.
LIBRESPOT_DISABLE_AUDIO_CACHE=

# Disable caching of credentials.
# Caching of credentials is not necessary so long as
# LIBRESPOT_DISABLE_DISCOVERY is not set.
LIBRESPOT_DISABLE_CREDENTIAL_CACHE=

# Play all tracks at approximately the same apparent volume.
LIBRESPOT_ENABLE_VOLUME_NORMALISATION=

# Enable verbose log output.
#LIBRESPOT_VERBOSE=

# Disable zeroconf discovery mode.
#LIBRESPOT_DISABLE_DISCOVERY=

# Options will fallback to their defaults if commented out,
# otherwise they must have a valid value.

# Device name.
# Raspotify defaults to "raspotify (*hostname)".
# Librespot defaults to "Librespot".
LIBRESPOT_NAME="K9"

# Bitrate (kbps) {96|160|320}. Defaults to 160.
#LIBRESPOT_BITRATE="160"

# Output format {F64|F32|S32|S24|S24_3|S16}. Defaults to S16.
#LIBRESPOT_FORMAT="S16"

# Displayed device type. Defaults to speaker.
```

```
#LIBRESPOT_DEVICE_TYPE="speaker"

# Limits the size of the cache for audio files.
# It's possible to use suffixes like K, M or G, e.g. 16G for example.
# Highly advised if audio caching isn't disabled. Otherwise the cache
# size is only limited by disk space.
#LIBRESPOT_CACHE_SIZE_LIMIT=""

# Audio backend to use, alsa or pulseaudio. Defaults to alsa.
#LIBRESPOT_BACKEND="pulseaudio"

# Username used to sign in with.
# Credentials are not required if LIBRESPOT_DISABLE_DISCOVERY is not set.
LIBRESPOT_USERNAME="[USERNAME]"

# Password used to sign in with.
LIBRESPOT_PASSWORD="[PASSWORD]"

# Audio device to use, use `librespot --device ?` to list options.
# Defaults to the system's default.
#LIBRESPOT_DEVICE="K9"

# Initial volume in % from 0 - 100.
# Defaults to 50 For the alsa mixer: the current volume.
LIBRESPOT_INITIAL_VOLUME="100"

# Volume control scale type {cubic|fixed|linear|log}.
# Defaults to log.
#LIBRESPOT_VOLUME_CTRL="log"

# Range of the volume control (dB) from 0.0 to 100.0.
# Default for softvol: 60.0.
# For the alsa mixer: what the control supports.
#LIBRESPOT_VOLUME_RANGE="60.0"

# Pregain (dB) applied by volume normalisation from -10.0 to 10.0.
# Defaults to 0.0.
#LIBRESPOT_NORMALISATION_PREGAIN="0.0"

# Threshold (dBFS) at which point the dynamic limiter engages
# to prevent clipping from 0.0 to -10.0.
# Defaults to -2.0.
#LIBRESPOT_NORMALISATION_THRESHOLD="-2.0"

# The port the internal server advertises over zeroconf 1 - 65535.
# Ports <= 1024 may require root privileges.
#LIBRESPOT_ZEROCONF_PORT=""

# HTTP proxy to use when connecting.
#LIBRESPOT_PROXY=""

# ### This is NOT a librespot option or flag. ###
# By default librespot "download buffers" tracks, meaning that it downloads
# the tracks to disk and plays them from the disk and then deletes them when
# the track is over.
TMPDIR=/tmp
```

Appendix C

Alsa Sound Configuration File

Location: /etc/asound.conf

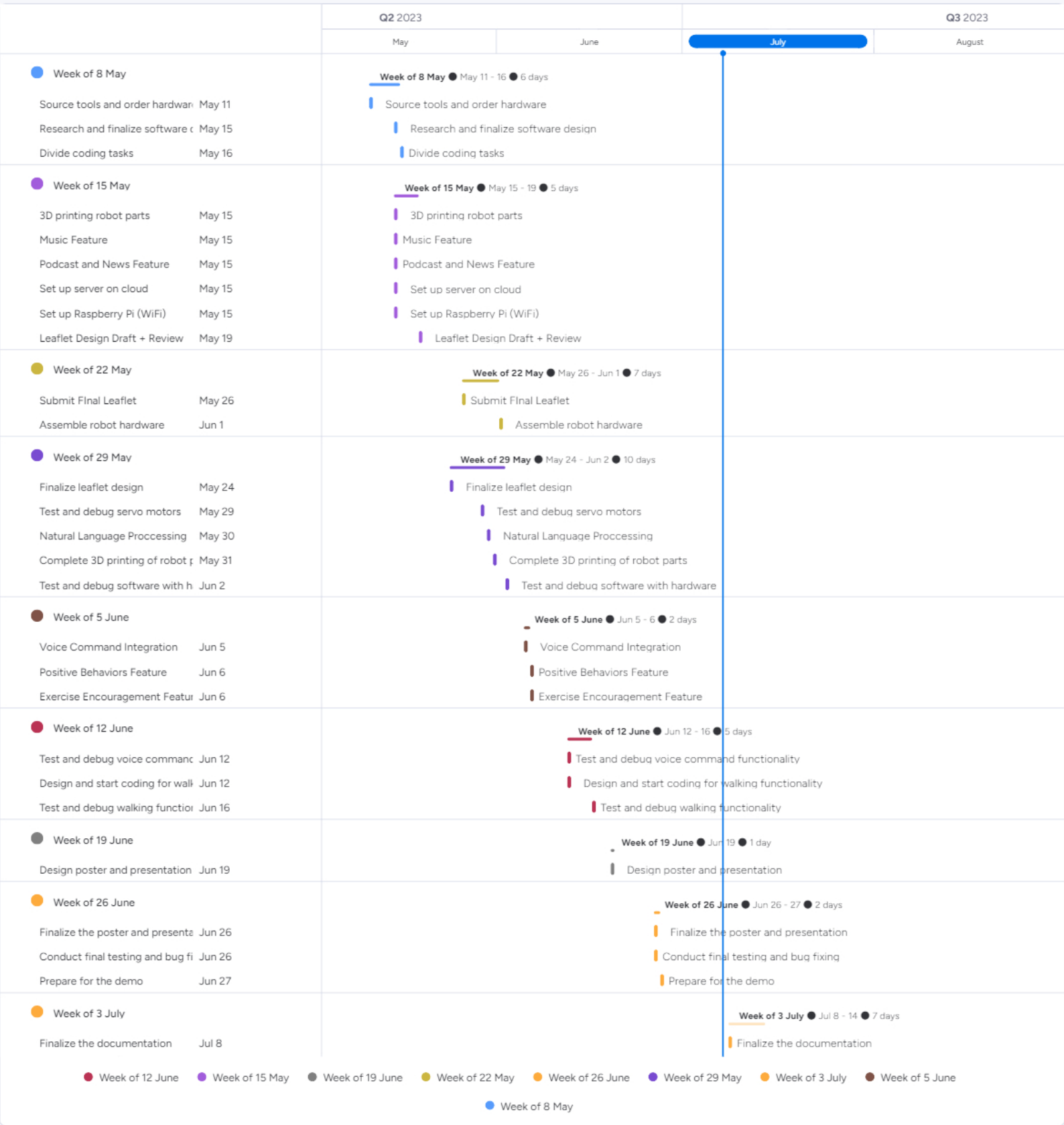
```
pcm.speakerbonnet {
   type hw card 0
}

pcm.dmixer {
   type dmix
   ipc_key 1024
   ipc_perm 0666
   slave {
     pcm "speakerbonnet"
     period_time 0
     period_size 1024
     buffer_size 8192
     rate 44100
     channels 2
   }
}

ctl.dmixer {
    type hw card 0
}

pcm.softvol {
    type softvol
    slave.pcm "dmixer"
    control.name "PCM"
    control.card 0
}

ctl.softvol {
    type hw card 0
}

pcm.!default {
    type             plug
    slave.pcm       "softvol"
}
```

Appendix D

Gantt Chart

Gantt

July 07, 2023 | 16:41:43



Appendix E

References

E.1 Python Packages:

- Numpy : https://numpy.org/doc/
- Bezier : https://pypi.org/project/bezier/
- Random : https://docs.python.org/3/library/random.html
- CSV : https://docs.python.org/3/library/csv.html
- Requests : https://pypi.org/project/requests/
- Re : https://docs.python.org/3/library/re.html
- pyttsx3 : https://pypi.org/project/pyttsx3/
- SpeechRecognition : https://pypi.org/project/SpeechRecognition/
- datetime : https://docs.python.org/3/library/datetime.html
- struct : https://docs.python.org/3/library/struct.html
- os : https://docs.python.org/3/library/os.html
- ElevenLabs : https://github.com/elevenlabs/elevenlabs-python
- Pillow : https://pypi.org/project/Pillow/
- luma.core.interface.serial : https://luma-core.readthedocs.io/en/latest/interface.html
- luma.oled.device : https://luma-oled.readthedocs.io/en/latest/python-usage.html
- Threading : https://docs.python.org/3/library/threading.html
- gcsa : https://pypi.org/project/gcsa/
- Beautiful Date : https://pypi.org/project/beautiful-date/
- json : https://docs.python.org/3/library/json.html
- Calendar : https://docs.python.org/3/library/calendar.html
- ElevenLabLib : https://github.com/lugia19/elevenlabslib
- Pvporcupine : https://pypi.org/project/pvporcupine/
- Pyaudio : https://pypi.org/project/PyAudio/
- Multiprocessing : https://docs.python.org/3/library/multiprocessing.html
- Time : https://docs.python.org/3/library/time.html
- Spotipy : https://spotipy.readthedocs.io/en/2.22.1/
- Pickle : https://docs.python.org/3/library/pickle.html
- Face Recognition : https://pypi.org/project/face-recognition/
- CV2 : https://pypi.org/project/opencv-python/
- IBM Watson : https://pypi.org/project/ibm-watson/
- IBM Cloud SDK Core: https://pypi.org/project/ibm-cloud-sdk-core/

E.2 Websites:

- Github : https://github.com/SpaceBod/K9AI
- Watson Assitant : https://www.ibm.com/products/watson-assistant
- IBM Cloud : https://cloud.ibm.com/
- Picovoice/ Porcupine : https://picovoice.ai/docs/api/porcupine-c/

- Spotify for Developers : https://developer.spotify.com/
- Raspotify : https://github.com/dtcooper/raspotify
- Adafruit Speaker Bonnet Setup : https://learn.adafruit.com/adafruit-speaker-bonnet-for-raspberry-pi
- ElevenLabs : https://beta.elevenlabs.io/

E.3 APIs:

- OpenWeatherMap : https://openweathermap.org/
- Spoonacular : https://spoonacular.com/food-api
- NewData : https://newsdata.io/
- ThinkSpeak : https://thingspeak.com/