CU Anveshan (BotBrain Campus Navigator): System Documentation

Author: Abhilash N S Reddy
Date: September 24, 2025

Abstract

This document provides comprehensive technical documentation for the BotBrain AI Campus Navigator, a desktop application designed to provide intelligent pathfinding and location-based information for a university campus. The system integrates a graphical, interactive map with a suite of classic pathfinding algorithms and a novel natural language query interface powered by a local Large Language Model (LLM). The architecture combines a robust PyQt5-based Graphical User Interface (GUI), an AI module for natural language understanding (NLU), and a simple Retrieval-Augmented Generation (RAG) system for providing location-specific details. This document details the system architecture, core components, algorithms, data structures, and operational procedures.

I. Introduction

Navigating a large university campus can be a challenge for new students, visitors, and even existing members of the community. Traditional static maps lack interactivity and the ability to provide optimal, real-time routing. The BotBrain AI Campus Navigator addresses this challenge by offering a dynamic and intelligent solution.

The primary goal of this project is to create an intuitive tool that allows users to find the most efficient path between two points on campus. The system achieves this through two main modes of interaction: a manual, point-and-click interface for selecting start and end points on a visual map, and an innovative AI-driven interface that allows users to make requests in natural language (e.g., "Show me the way from the library to the sports complex").

Key features include:

- **Interactive Visual Map:** A graphical representation of the campus where users can select locations and see the calculated path highlighted.
- **Multiple Pathfinding Algorithms:** Implements Breadth-First Search (BFS), Uniform Cost Search (UCS), Dijkstra's, and A* search, allowing users to select an algorithm based on their needs.
- **AI-Powered Querying:** Utilizes a local LLM (phi3:mini) to parse user queries, providing a hands-free, intuitive way to define routes.
- **Location Information System:** A simple RAG implementation provides users with relevant details about their destination, retrieved from a local knowledge base.

II. System Architecture

The application is built on a modular architecture, separating the user interface, AI processing, and data. This design enhances maintainability and scalability. The system consists of three primary components, as illustrated in the architecture diagram.

A. Core Components

1. **Main Application (main_app.py):** This is the central component of the system, built using the PyQt5 framework. It is responsible for:
    - Rendering the main application window and all UI elements (buttons, dropdowns, labels).
    - Managing user interactions.
    - Instantiating and controlling the GraphicsGraphWidget for map display.
    - Executing the selected pathfinding algorithms.
    - Coordinating with the AI module for processing natural language queries.
2. **AI Brain (botbrain_ai.py):** This module encapsulates the artificial intelligence capabilities of the application. Its primary functions are:
    - **Natural Language Understanding (NLU):** It uses the ollama library to interface with a locally-run LLM (phi3:mini) to parse user text and extract source and destination entities.
    - **Information Retrieval:** It implements a function to retrieve descriptive text about campus locations from the campus_info.json file.
3. **Data Files:**
    - **Knowledge Base (campus_info.json):** A JSON file that acts as a simple database, storing descriptions and operational details for each named location on campus. This serves as the retrieval source for the RAG system.
    - **Image Assets (/images directory):** Contains .png files corresponding to campus locations, used for the image preview feature in the GUI.

B. Asynchronous Processing

To ensure the GUI remains responsive while the LLM processes a query (which can take several seconds), the application employs a multi-threaded approach. An AIWorker object is moved to a separate QThread. When a user submits a query, a signal is emitted to the worker thread, which performs the LLM call. Upon completion, the worker emits a finished signal containing the result, which is then processed by the main UI thread. This prevents the application from freezing.

III. Core Functionality and Implementation

A. Graphical User Interface (main_app.py)

The GUI is designed to be user-friendly and informative, providing multiple ways to interact with the pathfinding system.

- **Main Window (CampusNavigatorApp):** This class initializes the entire UI, including layouts for the AI query bar, mode selection, control panels, and the graph widget. It also connects all signal-and-slot mechanisms for event handling.
- **Interactive Map (GraphicsGraphWidget):** This custom widget, inheriting from QGraphicsView, is responsible for rendering the campus map.
    - **Graph Rendering:** It draws nodes (NodeItem) and edges (QGraphicsLineItem) onto a QGraphicsScene based on predefined coordinates. Edges are drawn first to

appear underneath the nodes.
    - **Node Representation (NodeItem):** Each location is represented by a NodeItem object, which handles mouse hover events (for visual feedback and image previews) and click events (for selecting start/goal locations).
    - **Path Highlighting:** When a path is found, a series of thick, brightly colored QGraphicsLineItem objects are drawn on top of the graph to clearly visualize the route.

B. Pathfinding Algorithms

The application provides a selection of fundamental graph traversal algorithms, each suited for different optimization criteria. The campus layout is represented as a weighted, undirected graph, defined in the get_campus_graph() function.

- **Breadth-First Search (BFS):** An unweighted algorithm that finds the path with the minimum number of edges (or "steps"). It is ideal when the cost of traversing each edge is uniform.
- **Uniform Cost Search (UCS) & Dijkstra's Algorithm:** Both are algorithms that find the path with the minimum total cost (distance). They explore paths in increasing order of cost. Dijkstra's is used as the default for the "Automatic" mode due to its efficiency and guarantee of finding the shortest path.
- **A* Search:** An informed search algorithm that improves upon Dijkstra's by using a heuristic function to guide its search towards the goal. In this implementation, the heuristic is the Euclidean distance between a given node and the target node, which helps prune branches that are moving away from the destination.

C. AI and Natural Language Processing (botbrain_ai.py)

The integration of an LLM allows for a highly intuitive user experience.

- **Prompt Engineering (parse_user_query):** The core of the NLU capability lies in a carefully crafted prompt. The prompt instructs the phi3:mini model to act as a parsing assistant and provides it with a strict list of VALID_LOCATIONS. It is explicitly told to map common names to official names and to respond *only* with a JSON object containing "source" and "destination" keys. This structured prompting, combined with the format='json' parameter in the ollama call, ensures a high degree of reliability in the model's output.
- **Output Validation:** After receiving the JSON response from the LLM, the system performs a final validation step to ensure that the extracted source and destination are present in the VALID_LOCATIONS list. This prevents errors that could arise if the LLM "hallucinates" a non-existent location.
- **Retrieval-Augmented Generation (RAG):** The get_building_info function implements a simple form of RAG. When a path is found, this function is called with the destination's name. It reads the campus_info.json file (the knowledge base) and retrieves the corresponding description, which is then displayed in the UI. This "augments" the pathfinding result with useful, context-specific information.

IV. Setup and Operation

A. Prerequisites

- Python 3.x
- PyQt5: pip install PyQt5
- Ollama: Installation instructions available at https://ollama.com/

B. Configuration

1. **Install and Run Ollama:** After installing Ollama, pull the required model from the command line:
   ollama pull phi3:mini

2. **Project Structure:** Ensure the project directory is structured as follows:
```
/BotBrain_Navigator
    |-- main_app.py
    |-- botbrain_ai.py
    |-- campus_info.json
    |-- /images
        |-- Admin_Block.png
        |-- Library.png
        |-- ... (other location images)
```

C. Execution

To run the application, navigate to the project's root directory in the terminal and execute the main script:
python main_app.py

V. Conclusion and Future Work

The BotBrain AI Campus Navigator successfully demonstrates the powerful combination of traditional pathfinding algorithms with modern AI-driven natural language interfaces. The system provides a functional, interactive, and intelligent tool for campus navigation. The use of a local LLM ensures user privacy and removes reliance on external cloud services.

Potential areas for future development include:

- **Dynamic Data:** Integrating with a proper database to allow for dynamic updates to the campus map and location information without modifying the source code.
- **Real-time Updates:** Adding features for real-time events, such as temporary closures or congestion information.
- **Expanded Knowledge Base:** Enhancing the RAG system with more detailed information, such as office directories, event schedules, and accessibility notes.
- **Voice Commands:** Integrating a speech-to-text engine to allow for fully hands-free operation.

VI. References

[1] Ollama. (2024). *Large language models running locally.* [Online]. Available: https://ollama.com/

[2] Riverbank Computing. (2024). *PyQt5 Reference Guide.* [Online]. Available: https://www.riverbankcomputing.com/static/Docs/PyQt5/