Adaptive Cane for Visually Impaired People

Simon Dourte
Edwyn Eben
Nathan Lambrechts
Louca Mathieu
Florian Stormacq
simon.dourte@student.unamur.be
edwyn.eben@student.unamur.be
nathan.lambrechts@student.unamur.be
louca.mathieu@student.unamur.be
florian.stormacq@student.unamur.be
University of Namur, Namur Digital Institute (NaDI) and Research Center on Information Systems Engineering (PReCISE)
Namur, Belgium

Figure 1: Overview of the Adaptive Cane System with Intel RealSense camera, microphone, Raspberry Pi processing unit, and haptic feedback motors

AIM 2025, Namur, Belgium

ACM ISBN 000-0-0000-0000-0/00/00
https://doi.org/00.0000/0000000.0000000

Abstract

People with visual impairments represent a significant proportion of the global population. It is estimated that at least 2.2 billion people have some form of vision impairment. It is therefore essential to develop solutions that meet their needs and improve their independence. With this in mind, this project presents an adaptive connected cane for blind or visually impaired people designed to improve their perception of their surroundings and increase their safety in the face of potential obstacles and dangers. The cane is based on two main sensors: an Intel RealSense D435 depth camera and a USB microphone. The data from these sensors is processed in real time by a Raspberry Pi 3. The microphone assesses the noise level of the environment, while the camera detects the position and distance of obstacles and the recommended area to avoid them. This information is used to determine one of the device's three alert modes. User feedback is provided by three vibrating motors, each associated with a direction. The intensity and duration of the vibrations indicate the proximity of the danger and guide the user as they move. This prototype is therefore an assistance solution aimed at improving the mobility and safety of blind or visually impaired people in their daily lives.

ACM Reference Format:
Simon Dourte, Edwyn Eben, Nathan Lambrechts, Louca Mathieu, and Florian Stormacq. 2025. Adaptive Cane for Visually Impaired People. In *Proceedings of the UNamur Symposium on Advanced Interaction Methods (AIM 2025 Proceedings), December 25, 2025, Namur, Belgium.* ACM, New York, NY, USA, 5 pages. https://doi.org/00.0000/0000000.0000000

1 Introduction

The goal of the project is to develop an adaptive smart cane designed to improve spatial awareness and safety for visually impaired people. The system relies on two main sensors: a depth camera and a microphone to detect obstacles or danger in real-time. The data is collected by a Raspberry Pi which analyzes visual and audio information from the environment.

The system focuses entirely on haptic feedback. A set of 3 vibration motors is placed on the cane to provide tactile information to the user. Each motor has a direction which means one direction is missing. Indeed, we have left out the back direction because the camera will be settled as glasses so they cannot see behind the user. The vibration intensity and duration reflect the proximity of the danger, allowing the user to interpret the environment with the haptic output.

This cane aims to support blind people during urban navigation or any activity that has potential danger to the user.

2 Related work

[TODO: Related Work section to be completed - literature review needed]

Note: A preliminary reference exists about haptic feedback for navigation. This needs to be expanded significantly.

3 Methodology

The project is structured into several key components to ensure optimal functionality, performance, and maintainability.

3.1 Project Architecture

3.1.1 Sensors. **[TODO: Audio and Video Sensor sections to be completed by team members]**

3.1.2 Processing Unit. The processing unit used is a Raspberry Pi 3, which handles data acquisition, processing, and communication with the Arduino board. The Raspberry Pi runs multiple threads to manage audio and video data streams, synchronization, and command generation for the output layer.

The producer-consumer architecture is implemented with separate queues for the audio, video, synchronization buffer, and Arduino commands. Each queue has a maximum size to prevent memory overflow: the microphone and video queues can hold up to 10 items each, while the Arduino command queue is limited to 5 items. The queues use a LIFO (Last-In-First-Out) strategy to ensure real-time performance by prioritizing the most recent data. To avoid blocking the processing and communication threads, when a queue reaches capacity, the three oldest items are automatically dropped, and the dropped count is tracked for monitoring purposes.

This architecture allows for efficient and parallel processing of multimodal data, ensuring real-time feedback to the user. The system monitors queue statistics including total items processed, current queue sizes, and processing frequencies, which are logged every 10 seconds to track system performance.

Note: This device is not the initial Raspberry Pi 1 that was furnished at the beginning of the project, as it was not powerful enough to handle the processing requirements.

3.1.3 Arduino Board. In this project, Arduino is used as an interface between the central processing unit (Raspberry Pi) and the physical actuators (vibration motors). It receives already-processed instructions that specify how the motors should behave.

First, the Raspberry Pi sends data divided into three components: where the danger comes from, the intensity, and the duration of the vibration. The Arduino receives the information, interprets it, and converts it into appropriate signals for the vibration motors.

In the overall architecture, the Arduino is responsible for executing the physical output (vibration motors) in real-time. It does not process data; it applies the received instructions for the vibration motors.

3.2 Project Implementation

First, to ensure an easy setup of the project, UV was used to manage the Python dependencies. All required libraries are listed in the `pyproject.toml` file, allowing for a straightforward installation process.

3.2.1 Code Structure. The codebase is organized into several modules, each responsible for a specific functionality of the system. The entry point of the project is the `main.py` script, which orchestrates the different threads. This script initializes and starts all the necessary threads for data acquisition, processing, communication, and monitoring of the sent data. The main script supports several command-line arguments: `-no-audio` to disable audio capture, `-no-video` to disable video capture, `-debug` to enable verbose logging, and `-simulate` to simulate inputs for testing purposes. Each producer (audio and video) runs as a daemon thread to ensure proper shutdown when the main process terminates.

In addition to this main script, a secondary Python script (`monitor_serial.py`) is provided to monitor the serial communication between the Raspberry Pi and the Arduino board. This lightweight monitoring script connects to the Arduino via `/dev/ttyACM0` at 115200 baud and displays all incoming messages in real-time. To facilitate the startup of the entire system along with the monitoring script, a shell script (`start.sh`) is provided that launches both the main system and the monitor simultaneously. The script captures the process ID of the main system and ensures proper cleanup when the monitor is stopped with Ctrl+C.

3.2.2 Queue Manager. To simplify the management of the queues used for inter-thread communication, a dedicated module was implemented: `queue_manager.py`. This module creates a singleton class, `QueueManager`, which encapsulates the functionality of all queues used in the project. The `QueueManager` class initializes five separate queues: `micro_queue` and `video_queue` (maxsize=10) for raw sensor data, `audio_processed_queue` and `video_processed_queue` (maxsize=5) for processed features, and `arduino_queue` (maxsize=5) for commands to be sent to the Arduino.

Each queue is configured with a maximum size to prevent memory overflow. The module provides dedicated methods for adding and retrieving data from each queue (`put_micro_data ()`, `get_micro_data ()`, `put_video_data ()`, etc.). When adding data, the current timestamp is automatically attached as a tuple `(data, timestamp)` to facilitate temporal synchronization. The queue manager implements comprehensive monitoring, tracking both the total number of items processed and the number of dropped items for each queue. When a queue becomes full, the system drops the three oldest items before adding the new data, preventing blocking while maintaining real-time responsiveness. The `get_queue_stats ()` method provides detailed statistics about queue usage, enabling performance monitoring and system diagnostics. Encapsulating the queue management logic within a dedicated module simplifies the code, making it easier to understand and maintain.

3.2.3 Sensors Implementation. **[TODO: Audio and Video Sensor implementation sections to be completed by team members]**

3.2.4 Raspberry Pi Module. As the Raspberry Pi is the core processing unit of the system, it hosts several modules that handle different aspects of data processing and communication.

The `raspberry.py` module is the main module that integrates all functionalities, managing data flow between the audio and video modules, synchronization buffer, and Arduino communication.

As discussed earlier, the project architecture is based on a producer-consumer model, with multiple threads handling different data streams. Each sensor modality has its own dedicated thread for data acquisition and pre-processing, which then feeds the shared queues.

The `raspberry.py` script is responsible for all the heavy processing tasks. For audio data, the `heavy_audio_processing()` function computes the Root Mean Square (RMS) of the signal, converts it to dB level using the formula $20 \times \log_{10}(\text{rms}/\text{reference})$, and classifies the sound into four categories: "Chillax" (< -45 dB), "Some noise" (-45 to -30 dB), "Be Careful" (-30 to -15 dB), or "Danger" (> -15 dB). Additionally, it performs Fast Fourier Transform (FFT) analysis to detect the dominant frequency in the audio signal. For video data, the `heavy_video_processing()` function analyzes obstacle positions and distances, computing a danger level (0-3) based on the number and position of detected obstacles, with special priority given to center obstacles. The function also assigns risk classifications ("safe", "medium", "high", or "critical") corresponding to each danger level.

The script runs separate processing threads for audio and video data (`micro_processing_thread()` and `video_processing_thread()`), each continuously consuming data from their respective queues. These processed results are then added to intermediate queues before being synchronized using the `sync_buffer.py` module. Finally, commands are generated for the Arduino board based on the synchronized information, and these commands are sent via serial communication at 115200 baud.

Synchronization Buffer: **[TODO: Add detailed explanation of sync_buffer.py module and synchronization algorithm]**

Intensity Calculator: To adjust the output intensities based on the processed sensor data, the `intensity_calculator.py` script is used. This module implements the `IntensityCalculator` class with two static methods for converting sensor data into intensity values ranging from 0 to 100.

[TODO: Add detailed explanation of intensity calculation algorithms and thresholds]

Message Generator: A final relevant module is the `lcr_message_generator.py`, which is responsible for formatting the computed intensity levels into the specific command protocol required by the Arduino board. This module implements the `LCRMessageGenerator` class that maintains state including the last sent message and a message counter for tracking.

The command format used is `LxxxCxxxRxxx`, where `L`, `C`, and `R` represent the left, center, and right output intensities, respectively, each followed by a three-digit zero-padded intensity value (000-100). The module provides two generation methods: `generate_synchronized_message` for cases where both audio and video data are available, and `generate_fallback_m` for single-modality scenarios.

The synchronization method implements a sophisticated weighted averaging scheme to merge audio and video intensities. For the center zone, the formula is: $(4 \times \text{vision\_intensity} + \text{audio\_intensity})/5$, giving 80% weight to visual data and 20% to audio. For lateral zones (left and right), audio influence is further reduced to 70% of its original value: $(4 \times \text{vision\_intensity} + 0.7 \times \text{audio\_intensity})/5$. This weighting strategy prioritizes visual obstacle detection while still incorporating ambient sound information, particularly emphasizing visual data for the critical center zone where the user is heading. The weighted average calculation ensures smooth transitions between different environmental conditions.

These weighting values can be modified in the `lcr_message_generator.py` module to adjust the system's sensitivity according to user preferences, and this opens the possibility for future improvements, such as implementing a user-tunable sensitivity setting.

All these modules work together seamlessly to ensure that the system operates in real-time, providing appropriate feedback to the user based on the multimodal sensory input. The entire processing pipeline—from sensor data acquisition through queue management, processing, synchronization, intensity calculation, and message

generation—is designed to maintain low latency and high responsiveness, crucial for assistive navigation applications.

3.2.5 Arduino Board Implementation. **[TODO: Arduino implementation section to be completed by team members]**

Note: The Arduino module receives pre-processed commands from the Raspberry Pi via serial communication at 115200 baud using the LxxxCxxxRxxx protocol format.

4 Contribution

[TODO: Contribution section to be completed by team members]

Preliminary notes: The project introduces a multimodal approach combining depth and ambient audio information. Directional haptic feedback conveys both presence and position of obstacles.

5 Evaluation

The system was not evaluated in real-world conditions due to time constraints and the unavailability of certain hardware components. However, each prototype module was tested individually to ensure correct functionality. Additionally, the prototype was tested in a controlled environment to validate the overall system integration, as well as the visual feedback provided by the LED strip.

6 Discussion

6.1 Limitations

To begin with, the choice of hardware components imposed certain limitations on the system's performance. To ensure real-time processing, the Raspberry Pi 3 was selected as the processing unit. To replicate this project, it is important to use a Raspberry Pi 3 or a more powerful model, as older versions struggle to handle the computational load of processing all sensor data in real-time.

Another limitation of the prototype is the limited processing rate of the audio module. While the video module can process data at a rate of approximately 15 frames per second, the audio module is limited to processing only 6 items per second. This leads to a situation where audio data becomes the bottleneck for the overall system performance, as the synchronization buffer assembles data based on the produced timestamps. Consequently, the system can only provide updates at a maximum rate of 6 items per second, which may not be sufficient for fast-changing environments.

A final limitation to consider is the absence of real-world testing with visually impaired users. Due to time constraints and the unavailability of certain hardware components, the system could not be evaluated in practical scenarios. Real-world testing is crucial for assessing the effectiveness and usability of the system, as well as for gathering feedback from actual users to inform future improvements.

6.2 Problems Encountered

During the development of this project, several challenges were encountered that required problem-solving and adaptation. Some of the most notable issues included:

(1) Configuring the Raspberry Pi to work with the RealSense camera, which involved building the PyRealSense2 library from source. This process was time-consuming and error-prone, requiring careful attention to detail and extensive troubleshooting. For more information, please refer to the documentation provided in the GitHub repository.
(2) The initial project aimed to use vibration motors as output devices, one for each direction (left, center, right). These motors would have been placed on a handlebar to provide haptic feedback to the user. However, due to time limitations and difficulties in assembling the desired prototype, the vibration motors were replaced by an LED strip for visual feedback. This change allowed for a simpler and quicker implementation while still demonstrating the core functionality of the system.
(3) Limited documentation for the PyRealSense2 library made it difficult to install and utilize the camera effectively across different operating systems.
(4) Debugging multi-threaded applications proved to be, as expected, a complex task. Ensuring that all threads operated correctly and efficiently required careful design, testing, and the implementation of comprehensive logging and monitoring systems.
(5) Synchronizing audio and video data streams accurately to ensure that the output feedback corresponded correctly to the sensory input, particularly given the different capture rates and processing times of each modality.
(6) Saturating the Arduino communication buffer when sending commands too rapidly, which led to unresponsive output devices. To address this issue, a monitoring script (`monitor_serial.py`) was implemented to visualize the commands being sent to the Arduino in real-time, allowing for better debugging and optimization of the communication process.

6.3 Ideas for Future Work

As this project was developed within a limited timeframe, there are several areas of improvement and expansion that could be explored in future work. Some potential directions for future development include:

(1) Enhancing user interaction by implementing a user interface that allows for real-time adjustment of system parameters, such as the weighted averaging coefficients between audio and video inputs, sensitivity thresholds, and output modes.
(2) Integrating the originally intended vibration motors to provide haptic feedback and evaluating their effectiveness compared to the visual feedback provided by the LED strip. This modification would not require significant changes to the existing codebase, as the communication protocol with the Arduino would remain the same.
(3) Optimizing the audio processing module to increase the data processing rate beyond the current 6 items per second, allowing for more frequent updates and smoother feedback. This could involve implementing more efficient FFT algorithms or utilizing hardware acceleration.
(4) Conducting real-world testing with visually impaired users to evaluate the system's performance in various environments and scenarios, providing valuable insights into its practical usability, effectiveness, and user acceptance. Such

studies would also help identify necessary adjustments to the sensitivity settings and feedback mechanisms.

(5) Implementing adaptive sensitivity that automatically adjusts based on environmental conditions and user preferences learned over time.

7 Conclusion

This project successfully developed a multimodal assistive system that integrates audio and video sensors and provides real-time feedback to the user through a set of output devices. The system architecture, based on a producer-consumer model with multi-threading and sophisticated queue management, effectively handles the complexities of data acquisition, time-sensitive processing, temporal synchronization, and communication. The implementation includes comprehensive modules for audio and video processing, intensity calculation using weighted averaging, and command generation following a well-defined protocol.

As with all prototype projects, there are several areas for improvement that could be explored in future work, including enhancing user interaction through adaptive interfaces, optimizing processing modules for higher throughput, integrating the intended haptic feedback hardware, and conducting real-world evaluations with target users. Overall, this project demonstrates the feasibility and potential of multimodal systems for enhancing mobility and safety for visually impaired individuals through intelligent sensory feedback.

Open Science

The complete code for this project is available on GitHub at the following repository: https://github.com/fstormacq/Master1-IIA-project

Acknowledgments

This work was supported by the University of Namur, the Namur Digital Institute (NaDI), and the Research Center on Information Systems Engineering (PReCISE). The authors would like to thank all individuals who contributed to this project. A particular thanks to the providers of the hardware components that made this research possible.

References