Raspberry Pi Roomba

Joel Yuhas
Personal Hobby Project

Abstract
This project involved repurposing an old Roomba into a custom robotic platform by integrating hardware modifications, electrical components, and software development. Key tasks included removing unnecessary components from the Roomba, integrating a Raspberry Pi with supporting electronics—such as an H-bridge and voltage dividers—into the chassis, and connecting the Roomba's wheel motors and bumper sensors to the new system. A Bluetooth-enabled Xbox controller was used for control, while custom code, developed using object-oriented design principles, handled inputs and controlled the Roomba's functions. Additionally, a speaker was installed to play sounds in response to bumper activations. This project highlights the ability to design and implement a multi-disciplinary solution, transforming an obsolete device into a functional robot. The resulting platform demonstrates technical problem-solving, software engineering, and hands-on hardware integration skills, making it an ideal showcase for professional applications.

1 Introduction

The increasing accessibility of hardware components and development platforms, such as Raspberry Pi, has opened new opportunities for hobbyists and engineers to repurpose outdated devices into functional and innovative systems. This project demonstrates such an effort by transforming an old Roomba chassis into a custom robotic platform. The project was conceived as a personal initiative to explore robotics, software design, and hands-on hardware integration while utilizing object-oriented programming principles.

The primary objective of this project was to breathe new life into an obsolete device by integrating modern electronics and designing custom software to expand its functionality. By combining hardware modification, electrical engineering, and programming, the resulting robot could be controlled wirelessly using a Bluetooth-enabled Xbox controller and respond dynamically to sensor inputs, such as activating sounds upon detecting bumper contact.

This report presents a high-level overview of the project, detailing the objectives, methodology, results, and lessons learned. By focusing on both technical and creative aspects, this project highlights the interdisciplinary skills required to successfully execute complex engineering solutions. The experience and outcomes illustrate not only technical proficiency but also problem-solving and innovation, making this work a valuable addition to a professional portfolio.

2 Objectives

The objectives for this project are divided into "soft" and "hard" objectives. Soft objectives are higher-level, less discrete goals that serve as general guidelines for the project, while hard objectives are specific, measurable requirements.

2.1 Soft Objectives

1. Repurpose an old, broken Roomba as a platform to experiment with integrating new hardware and developing software on a custom setup.

2. Use a Raspberry Pi with Linux to gain a better understanding of implementing solutions on a Unix platform.
3. Leverage this project as a learning opportunity in both electrical and computer engineering, as well as in integrating software and hardware solutions.

2.2 Hard Objectives

1. Ensure the custom hardware can control the robot's movements.
2. Enable the robot's movements to be controlled by the user via an Xbox controller.
3. Implement functional bumper sensors whose signals can be read by the custom hardware.
4. Integrate a working speaker into the robot.
5. Play a random sound on the speaker when the bumper is activated.
6. Ensure the robot operates on its own internal power source.

3 Equipment

- Used Roomba 610 model (broken motherboard)
- Raspberry Pi (3B model)
- LM2596 DC to DC Voltage Regulator
- L298N Motor Driver
- Min battery powered Speaker
- Replacement Roomba Battery

4 Methodology

The design approach for this project can be condensed into two sections, the Hardware design and the Software Design.

4.1 Hardware Design

For the hardware design of the Roomba project, the initial step involved disassembling the Roomba, thoroughly cleaning its components, and reassembling it after removing the broken motherboard, dust bin motor, and primary brush rollers. This created ample space for mounting the Raspberry Pi and other devices inside the chassis. The old battery was replaced with a new one, and the Raspberry Pi, motor driver, and voltage regulators were installed in the underside carriage where the main roller brushes had been.

To power the Raspberry Pi and sensors, a voltage regulator stepped down the battery's 14.8V to 5V. Additionally, the bumper's IR sensors were connected to a breadboard placed in the cavity where the motherboard had once been. To ensure proper sensor operation, pull-down resistors were used to interface with the IR sensors. The motor driver was connected to the battery's 14.8v supply and the output terminals were connected to the Roomba wheel motors. Modifications to the to the wheel motor ports were required to allow for clean connections to the positive and negative terminals.

Finally, the breadboard for the bumper sensors, along with the motor driver and command signals, were all wired to the Pi's GPIO pins. A small, battery-powered speaker was mounted in the Roomba's dust bin cavity and connected to the Raspberry Pi via an aux cable to enable audio output.

Figure 1 shows a high level diagram that demonstrates how the hardware components interface together.

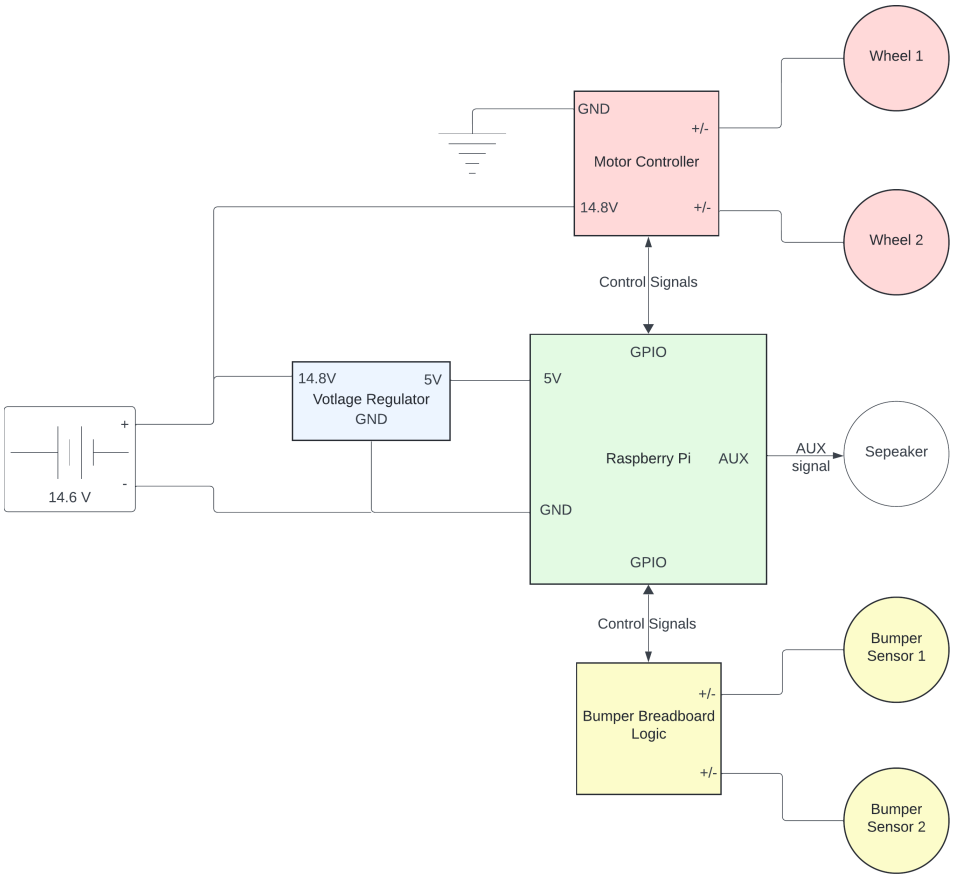

Figure 1: High Level Hardware Diagram.

4.2 Software Design

The software running on the Raspberry Pi for this project was designed using a class-based, object-oriented approach to enhance readability, maintainability, and scalability. This design ensures that the code is well-organized and can be easily extended, providing a solid foundation for future project expansions. The code is structured into several wrapper and handler classes, each dedicated to specific functionalities within the system.

The Libraries Directory houses the primary classes. The three "Manager" classes are pivotal for managing distinct aspects of the Roomba's operation. The AudioManager class organizes audio files, sets up the audio drivers, and controls audio playback. The ControllerManager class is responsible for instantiating the Xbox controller and processing input from its buttons and joysticks. The GPIOManager class initializes the Raspberry Pi's GPIO pins, facilitating communication and signal processing between the Pi and the Roomba's hardware. In addition to these, there are two "Roomba" classes that operate in their own threads to handle specific Roomba functionalities. The RoombaMotion class uses input from the Xbox controller's left analog stick to control the wheel motors, guiding the Roomba in a given direction. The RoombaBumper class monitors the bumper sensor input, and upon detecting a collision, triggers the playback of a random sound.

The Tests Directory contains tests that were created to validate the software and hardware components. Some tests instantiate the classes for functionality verification, while others run the modules directly as sanity checks to ensure all systems are working properly.

In the Audio Directory, audio files in .wav format are stored (with .mp3 backups available).

The AudioManager class automatically scans this directory at startup and compiles a dynamic list of valid audio files for use.

Finally, run_roomba.py is the central file that initiates all the necessary objects and is configured to run automatically upon the Pi's startup, ensuring that the Roomba operates as intended immediately after booting.

Figure 2 shows a high level UML diagram that shows the relations of these software components.

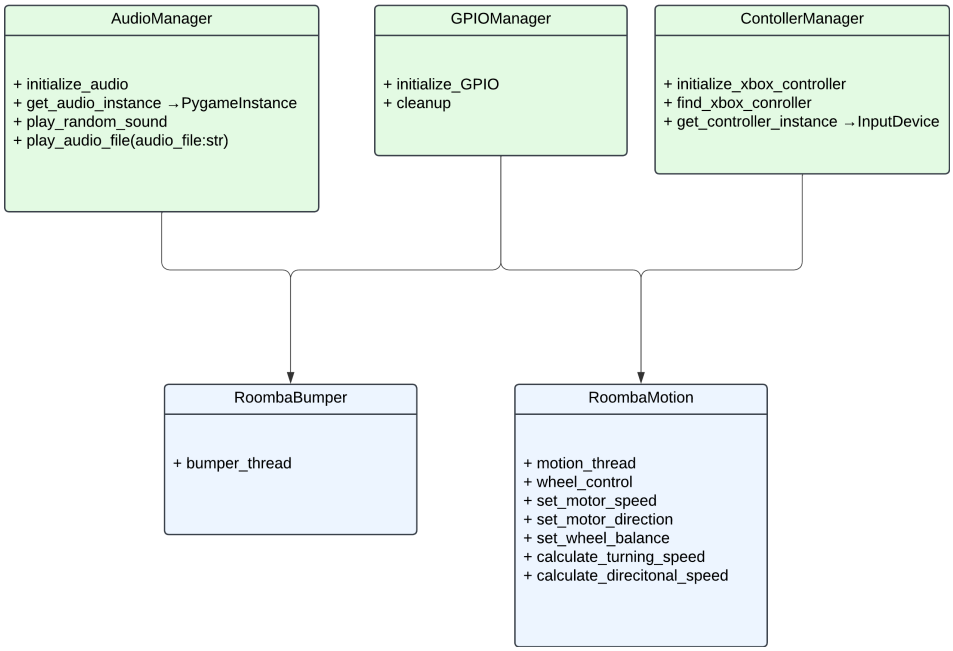

Figure 2: High Level UML Diagram of the software relation.

5 Results and Discussion

5.1 Results

The project successfully modified the Roomba hardware by attaching the Raspberry Pi and integrating the necessary supporting connections. The software logic for controlling the Roomba was also implemented and performed as expected. All major objectives were accomplished, and in some cases, the results exceeded expectations. For example, the integration with the Xbox controller for manual control of the Roomba was highly responsive, and the battery lasted several hours under heavy use before requiring a recharge.

Additionally, the Xbox controller was successfully connected via Bluetooth, eliminating the need for wired connections. This enhancement allowed the robot to operate fully wirelessly, improving its mobility and ease of use without relying on external power sources.

Figure 6 shows a photo of the undercarriage of the Roomba, highlighting key components such as the Raspberry Pi, voltage regulator, motor controller, and speaker. The speaker is positioned inside the transparent dust bin, which is visible in the image. Additionally, the custom battery connectors are shown, used to extract power from the Roomba's battery to power the components. The wheel motors and other core hardware are also visible.

Figure 4, shows the top portion of the Roomba

with the cover removed, revealing the cavity where the motherboard is typically located. This space houses the breadboard logic and supporting connections for the bumper control. This design allowed for easy adjustments and troubleshooting, which was particularly useful during the testing phase.

Finally, Figure 5 provides a close-up view of the control logic that connects the breadboard to the Roomba's wheel motors and bumper sensor. The red circle on the right highlights the connection points for the wheel motors, while the circle on the bottom shows how the wires interface with the Roomba's existing control pin plug to enable the operation of the left bumper sensor's IR controls.

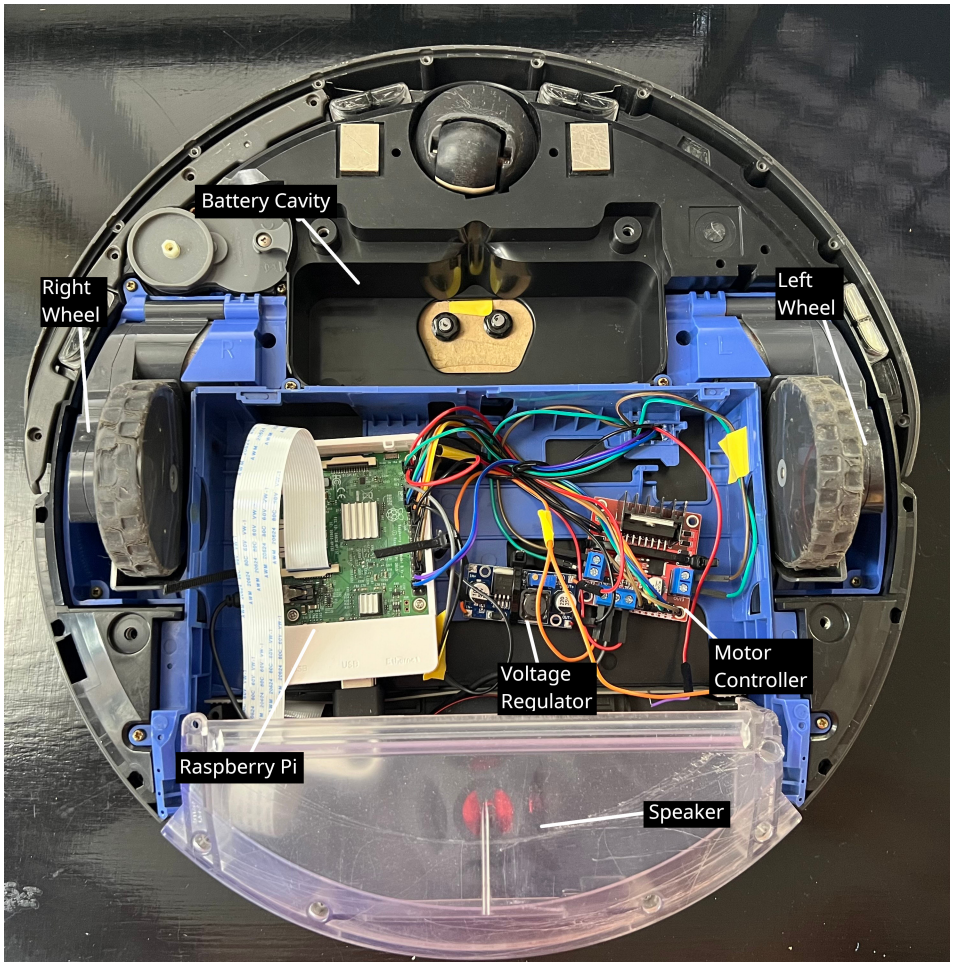

Figure 3: Bottom view of the Roomba with installed components.

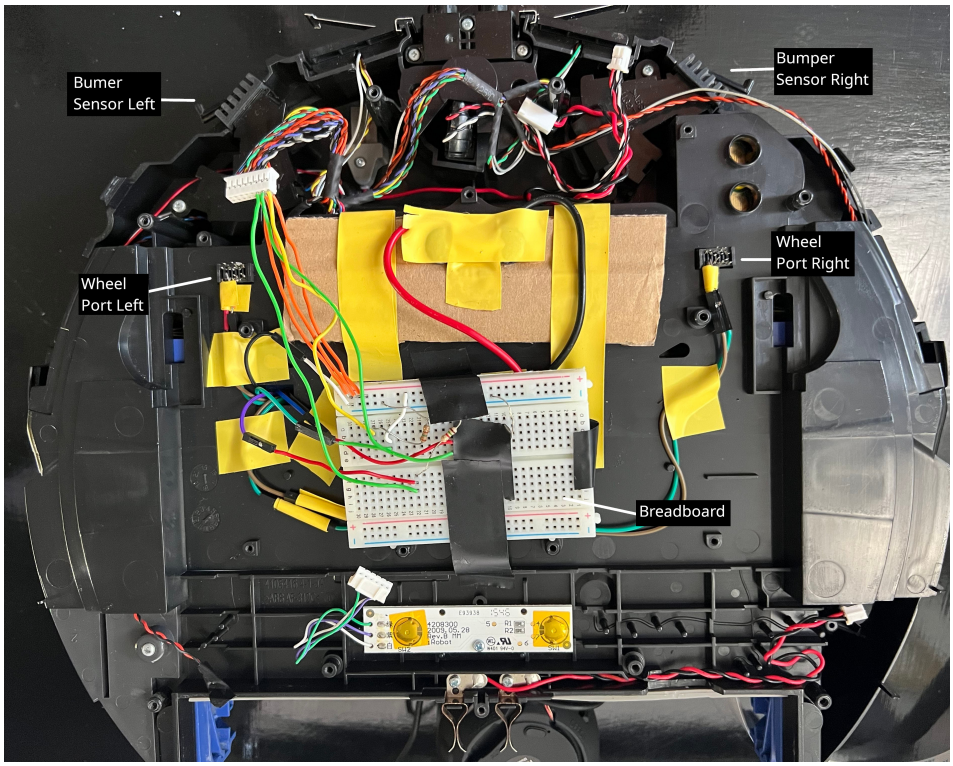

Figure 4: Top view of the Roomba with the breadboard and connections.

5.2 Discussion

From a software perspective, a key challenge was converting the Xbox controller input into the movement output for the Roomba. Since the robot has only two wheels, a "tank" control system was employed, allowing the Roomba to perform 360-degree rotations in place. This control method proved to be intuitive and responsive during testing.

Some issues persisted with the consistency of the bumper logic, specifically concerning the breadboard. After extended periods of inactivity, the bumper sensors sometimes failed to read activations correctly. This is likely due to the breadboard connections being less reliable and inconsistent, particularly when the system was jostled. A more robust solution, such as soldering connections or using a more reliable connection method, may be necessary to address this issue in future iterations.

Charging the battery required removing it from the Roomba and placing it into another Roomba equipped with a charging dock. This process could be streamlined in future versions by integrating a charging circuit directly into the modified Roomba, allowing for easier recharging without removing the battery.

Looking forward, future work could be focused on transitioning the software to a framework similar to the "Robot Operating System" (ROS 2), which is a widely used industry standard in robotics. One key initiative is to modularize the code by creating individual classes for each sensor. This would allow sensors, such as the bumper sensors and wheels, to be used more generically across various systems, making the code more flexible and scalable. This modular approach would also make it easier to expand the system's functionality in future versions.

Additionally, turning the three wrapper classes into Singleton classes could help ensure that no more than one of them are instantiated at a time, which could help prevent issues when declaring the wrapper objects.

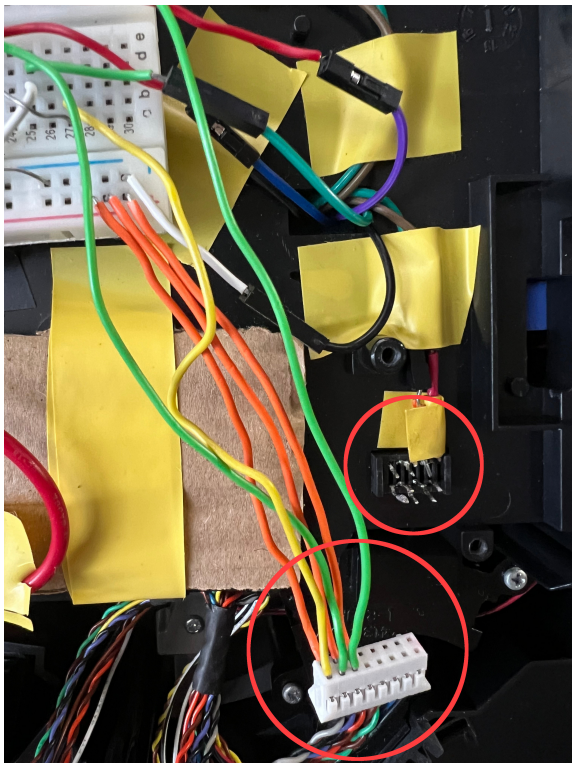

Figure 5: Close up wheel and bumper sensor connections.

Finally, work with the PiCamera is currently under way, and future work that can potentially leverage this camera is being looked into.

6 Conclusions

Figure 6: Photo of the modified Roomba.

This project successfully modified a Roomba by integrating a Raspberry Pi and implementing custom software to control the robot. The project achieved all major objectives, including seamless manual control via an Xbox controller and the creation of a functional, wireless system with an extended battery life. Through these modifications, the Roomba demonstrated

enhanced mobility and ease of use, with the ability to perform 360-degree rotations and respond to environmental interactions through bumper sensors.

While the project was largely successful, several challenges emerged, particularly with the reliability of the bumper sensor logic, which was affected by the use of a breadboard for connections. Future work will focus on addressing these issues by improving the physical connections and exploring alternative solutions for more consistent sensor performance. Additionally, the charging process, while functional, could be streamlined to allow for easier recharging without needing to remove the battery.

The long-term vision for this project includes expanding its capabilities by adopting the Robot Operating System (ROS 2), which would enhance code modularity and facilitate future sensor and control system integrations. This project serves as a solid foundation for further innovation and provides valuable insights into robotic system design, offering opportunities for continued growth and improvement in future iterations.