Controlling an AlphaBot Robot with a Wii Balance Board

Bruno Rosendo
Faculty of Engineering of the
University of Porto, Portugal
up201906334@fe.up.pt

João Mesquita
Faculty of Engineering of the
University of Porto, Portugal
up201906682@fe.up.pt

Rui Alves
Faculty of Engineering of the
University of Porto, Portugal
up201905853@fe.up.pt

ABSTRACT

This paper presents the development of a real-time kernel for embedded systems, with a focus on optimizing task scheduling. Two versions of the kernel were developed using Python and C languages, respectively. The system employed a Raspberry Pi, a Wii Balance Board, and an AlphaBot2-Pi robot. Challenges were encountered due to the limitations of the Raspberry Pi, which heavily relied on Python as its primary language, making low-level languages impractical for development. All tasks within the system were implemented using Python, while the C version of the kernel acted as a bridge, calling and managing these Python tasks. The C version provided superior precision and consistent interrupts compared to the Python version. The project resulted in a functional system that met the specified requirements, demonstrating the feasibility of using a real-time kernel to optimize task scheduling in an embedded system.

KEYWORDS

embedded, systems, real, time, AlphaBot, raspberry, Wii, balance, remote, control, camera, sensors, python, timer, interrupts, bluetooth

1 INTRODUCTION

The present work showcases the development of an embedded real-time system with the goal of controlling an AlphaBot2-Pi [9] using a Wii Balance Board [10] connected to a Raspberry Pi [6] via Bluetooth. The AlphaBot, a mobile robot platform, offers various applications in robotics and automation. The integration of the Wii Balance Board, originally designed for gaming, allows for intuitive control based on weight distribution and balance metrics.

The paper starts by **specifying the problem**, where the goal of developing a real-time kernel [8] to control the AlphaBot is further explained, followed by a detailed explanation of the **system architecture**, including hardware and software. Afterward, the necessary **tasks** and respective **performance** is analyzed in detail with conclusions on the system's schedulability and the appropriate periods. Lastly, the paper shows the conclusions taken and a video displaying the final version of the system.

2 SPECIFICATION OF THE PROBLEM

This project aims to enhance the team's understanding and expertise in embedded and real-time systems. Consequently, a key area of focus lies in creating a **real-time kernel** that can handle various tasks with adjustable periods and delays. The ability to select between preemptive and non-preemptive capabilities in the kernel is desired, depending on the system's specific requirements.

Another primary objective of this project is to construct and program a **robot car** (AlphaBot) that can be remotely controlled using **Bluetooth**, specifically through a **Wii Balance Board**. In addition, the robot incorporates a front camera that provides a real-time stream of its point of view (POV). To ensure safe navigation, the robot is equipped with a pair of infrared sensors that help prevent collisions with objects. This feature greatly assists the operator in controlling the robot while using the board.

The integration of the Wii board allows users to control the robot car intuitively by leveraging their movement and balance. By translating weight shifts and tilting from the board into corresponding actions, such as acceleration and steering, the control system offers a unique and immersive user experience.

The robot car should be powered by batteries, offering unrestricted driving range and eliminating any cable-related steering limitations. This design ensures flexible movement and allows the car to operate within the Bluetooth connection range.

3 SYSTEM ARCHITECTURE

3.1 Hardware

The system's hardware architecture comprises several components that collectively enable the remote control of the robot car using the Wii Balance Board. These components include the AlphaBot2-Pi, a Raspberry Pi, the Wii board, a front-facing camera module, a small buzzer, and infrared sensors.

The core of the hardware architecture is the **AlphaBot2-Pi**, which serves as the mobile robot platform. It consists of a chassis equipped with **two motor wheels**, and connectors for various additional modules. The AlphaBot provides the physical mobility and functionality required for the project.

The **Raspberry Pi**, a powerful microcontroller board, acts as the central processing unit and control hub of the system. It is responsible for running the real-time **kernel**, receiving input from the board, processing the sensor data, and generating control signals to drive the AlphaBot's motors. The Raspberry Pi's **GPIO** (General Purpose Input/Output) pins are utilized to interface with the AlphaBot, enabling motor control and sensor integration.

The Wii Balance Board serves as the primary input device for controlling the robot car. It communicates wirelessly with the Raspberry Pi via Bluetooth, providing **weight distribution** and **balance metrics** to determine the desired movements and acceleration of the robot.

To enhance the operator's control experience, a front-facing camera module is incorporated into the system. The camera includes a **servo motor** [11] to rotate it and captures real-time video footage of the robot's POV, providing visual feedback to the operator. This video stream assists in navigation, enabling the operator to maneuver the robot car more effectively and avoid potential obstacles.

For **collision avoidance**, the system utilizes a pair of **infrared sensors** mounted on the AlphaBot. These sensors detect the presence of objects in the vicinity and send the information to the Raspberry Pi which is used to trigger appropriate actions, such as stopping.

3.2 Software

The software architecture of the system encompasses various components that work together to enable the control of the robot car. These components include the real-time kernel, Bluetooth communication, sensor data processing, and motor control functionalities.

3.2.1 ***Real-time Kernel.*** At the core of the software architecture is the real-time kernel, which provides the foundation for managing the system's tasks and ensuring timely execution with fixed priority. The real-time kernel is responsible for scheduling and prioritizing the different software components, allowing for precise control of the robot's movements.

Developing the kernel posed a significant challenge in this project, primarily due to the fact that the Raspberry Pi runs on a full-fledged Operating System (OS). This OS significantly reduced our control over the system, and the majority of utilities and documentation available were specifically created using Python. As a result, low-level languages such as C or C++ were rendered impracticable choices for task development. Thus, we were left with two options: developing the kernel in Python or implementing it in C while utilizing CPython's API to invoke Python tasks [2]. Each option had its own set of advantages and disadvantages. To gain valuable insights, we decided to implement the kernel in both languages and draw our own conclusions based on the results.

Both of the implementations were based on the preemptive multitasking kernel present in SETR curricular unit classes [8]. Additional implementation details can be found there.

Python Kernel: The team encountered difficulties in developing the Python kernel for the project. One significant challenge arose from the inability to directly access the Raspberry Pi's hardware timer, unlike platforms such as Arduino or Raspberry Pi Pico. To simulate timer interrupts, the team attempted to use periodic alarms, leveraging features like *threading.Timer* [3].

However, Python's inherent slowness due to its interpreter and garbage collector posed obstacles. It became exceedingly difficult to achieve consistent alarms with the precision of regular timer interrupts. In the team's tests, only alarms with a period of 100 milliseconds could be consistently obtained. Unfortunately, this value was too high and fell short of meeting the project's task requirements, as elaborated in the subsequent section.

C Kernel: Running on a faster language, the C kernel enabled precise alarm handling through the utilization of *SIGALRM* signals and the *getitimer* API [5]. By employing these mechanisms, the team achieved consistent signal interrupts every 1 millisecond, significantly surpassing the system's required precision.

However, integrating the C kernel with Python tasks proved to be more challenging. To incorporate Python functionality, the team utilized the CPython API accessible via *Python.h*. This involved starting an interpreter, importing necessary modules, and executing tasks accordingly. One notable drawback of this approach was the inability to preempt a task with another if they shared at least one object (e.g., variable, function, class). This limitation stemmed from the abrupt interruption of the interpreter while executing a function, leading to an unrecoverable segmentation fault when initiating conflicting tasks.

After careful evaluation, considering the necessity for precise interrupts, the team concluded that the Python kernel fell short of meeting the project's requirements. Consequently, the decision was made to proceed with the C kernel. Additionally, the team chose to disable preemption in this kernel due to the tasks' dependence on a shared Bluetooth socket and their relatively low average and worst execution times, as detailed in the subsequent section.

3.2.2 ***AlphaBot Control.*** To enhance code organization and modularity, the software responsible for controlling the AlphaBot was divided into two Python classes: *Alphabot* and *Camera*. The *Alphabot* class provides methods for driving the robot in a specified direction or based on input from the balance board, stopping the robot, activating the buzzer, and checking for collisions using the infrared sensors.

On the other hand, the *Camera* class specifically handles control of the camera module, allowing rotation in various directions (pan and tilt), as well as starting and stopping video streaming. The video streaming functionality uses the camera to capture real-time video footage of the robot car's POV and transmits it to the operator's display device. The camera is initialized at the start of the program, so no task is assigned to it.

These classes are utilized by the tasks passed to the kernel, which will be further elaborated in the subsequent section.

The project uses a few libraries that offer interfaces to control hardware: Adafruit [1] for controlling servo motors, RPi.GPIO [7] to control Raspberry Pi's pins, and picamera [4] to program the camera module.

3.2.3 ***Bluetooth and Motors.*** The Bluetooth communication module in the *WiiBoard* class plays a crucial role in establishing a wireless connection between the Raspberry Pi and the Wii Balance Board. It facilitates seamless and reliable communication by handling Bluetooth pairing and command processing. This enables the Raspberry Pi to receive weight distribution data from the Wii Balance Board.

Upon obtaining the weight distribution data, it undergoes parsing and evaluation using predefined weight thresholds. These thresholds are appropriately adjusted to detect the user's desired actions, such as moving faster, moving slower, or remaining stationary.

Based on the parsed data, the power levels of the AlphaBot2-Pi's motors are defined in four categories: NONE (0% power), LOW (10% power), MEDIUM (20% power), HIGH (30% power), and VERY_HIGH (40% power). This categorization allows for finer control over the speed and enables adjustments as per the user's preferences.

To set the direction of the AlphaBot, the power of individual wheels can be selectively adjusted based on the weight distribution input.

Additionally, the Wii Balance Board features an integrated button that allows the detection of user actions, such as pressing and releasing. This button can activate or deactivate a buzzer on the

AlphaBot, providing audio feedback to enhance the control experience.

4 TASKS AND PERFORMANCE

In order to achieve effective control and coordination of the system, a set of tasks has been implemented within the real-time kernel. Each task is assigned a priority, which determines its execution order within the real-time kernel. This section provides an overview of the implemented tasks, priorities, and respective functionalities. Additionally, the scheduling of these tasks is examined, considering factors such as delays and periods to ensure optimal performance and responsiveness of the system [1].

Table 1: Tasks Properties (sort by highest priority).

Task	Delay	Period	Avg. Time	Max. Time
Connect To Board	0 ms	5000 ms	-	4000 ms
Check Collision	0 ms	50 ms	0.006 ms	0.443 ms
Drive Alphabot	10 ms	50 ms	0.012 ms	0.320 ms
Read Wii Data	0 ms	50 ms	10.083 ms	40.084 ms
Buzz	50 ms	100 ms	0.006 ms	0.468 ms

4.1 Worst-Case Response Times (RWC)

Referring to the information provided in Table 1 [Table 1], and assuming a non-preemptive kernel, we can determine the worst-case response times for each task. When considering the formulas, the maximum execution time is denoted as C. and the period as T It is important to note that tasks 2 to 5 can only be executed once the board is connected. Consequently, t1 and t2-t5 are mutually exclusive, meaning they cannot run concurrently.

- **t1 - Connect to Board:** Spends up to 4 seconds trying to connect to the Wii Balance Board, aborting if unsuccessful. If successful, opens a Bluetooth socket for later use and calibrates the board.

$$Rwc1(0) = C1 = 4000ms$$

- **t2 - Check Collision:** Reads the infrared sensors to check for possible collisions and saves the information.

$$Rwc2(0) = C2 + C4 = 0.443 + 40.084 = 40.527ms$$

- **t3 - Drive Alphabot:** Configures Alphabot's motors based on the saved data from the board and sensors.

$$Rwc3(0) = C3 + C2 + C4 = 0.320 + 0.443 + 40.084 = 40.847ms$$

- **t4 - Read Wii Data:** Uses the Bluetooth socket to read new data from the Wii Balance Board.

$$Rwc4(0) = C4 + C2 + C3 + C5 =$$
$$= 40.084 + 0.443 + 0.320 + 0.468 = 41.315ms$$

- **t5 - Buzz:** Checks if a button press was detected and activates the Alphabot buzzer if so.

$$Rwc5(0) = C5 + C2 + C3 + C4 =$$
$$= 0.468 + 0.443 + 0.320 + 40.084 = 41.315ms$$

[1]Connection's average time depends on the board's operator. The worst case is when the connection fails.

Given that all response times in the system are lower than their respective task's period, we can conclude that the system is **schedulable**. It should be noted that if the kernel was preemptable, the worst-case response times of high-priority tasks would be significantly lower, allowing for smaller periods if lower input response times were desired.

Additionally, there is a firm constraint on the system. Based on conducted tests, it has been determined that the Alphabot has a maximum speed of 60 cm/s, and the infrared sensors can detect obstacles within a range of 18 cm. Therefore, it is crucial for the tasks responsible for detecting and halting the Alphabot to have a maximum reaction time of 0.3 seconds or less. This is accomplished by our system, as seen in the following formula:

$$T2 + Rwc2 + C3 = 50 + 40.527 + 0.320 = 90.847ms$$

This is possible considering the maximum time for t2 to be ready (T2), its Rwc and the maximum execution time for t3 (C3) since it'll certainly be the ready task with the highest priority.

5 CONCLUSIONS AND FUTURE WORK

The paper discusses the development of two versions of a multi-tasking real-time kernel, one implemented in the C language and the other in Python. These kernels were utilized to create a task-based system that enables control of an Alphabot2-Pi robot through communication with a Wii Balance Board. The system encompasses additional functionalities such as a camera streaming to the operator's screen, infrared sensors for collision avoidance, and a buzzer triggered by pressing a button on the board. Detailed explanations of the project's objectives, system architecture, developed tasks, and overall performance are provided in the paper.

The results of the project were favorable, leading to the successful creation of a functional system that meets the specified requirements. However, there are areas that could be further improved in future work. Contrary to the implemented Python kernel, one significant aspect to improve in its C counterpart involves the development of a fully preemptable and multi-stack kernel capable of executing any type of Python task, even when sharing resources. This enhancement would entail substantial transformations to the kernel's functionality, primarily due to its multi-stack nature. In addition to maintaining Bluetooth sockets through the C kernel and refactoring Python tasks, the overall structure and operation of the kernel would need to be reimagined, to permit the use of more than one Python interpreter and the synchronization between them through *inter-process communication* mechanisms.

6 VIDEO AND MEMBERS' CONTRIBUTION

A short video showing the project can be found on Youtube.

The contribution of the members was even, 1/3 for each student.

REFERENCES

[1] CircuitPython. 2023. Adafruit servokit library. Retrieved June 4, 2023 from https://docs.circuitpython.org/projects/servokit/en/latest/index.html.

[2] Python Software Foundation. 2023. The python programming language. Retrieved June 5, 2023 from https://github.com/python/cpython.

[3] Python Software Foundation. 2023. Thread-based parallelism. (June 2023). Retrieved June 5, 2023 from https://docs.python.org/3/library/threading.html#threading.Timer.

[4] Dave Jones. 2016. Picamera. Retrieved June 4, 2023 from https://picamera.readthedocs.io/en/release-1.13/.

[5] Michael Kerrisk. 2022. Getitimer(2) - linux manual page. (Dec. 2022). Retrieved June 5, 2023 from https://man7.org/linux/man-pages/man2/setitimer.2.html.
[6] Raspberry Pi. [n. d.] Raspberry pi 4 model b. Retrieved June 4, 2023 from https://www.raspberrypi.com/products/raspberry-pi-4-model-b.
[7] PyPI. 2022. Rpi.gpio. (Feb. 2022). Retrieved June 4, 2023 from https://pypi.org/project/RPi.GPIO/.
[8] Luis Almeida & Mário Sousa. 2023. Setr curricular unit slides. (Feb. 2023). Retrieved June 4, 2023 from.
[9] Waveshare. [n. d.] Alphabot2-pi - waveshare wiki. Retrieved June 4, 2023 from https://www.waveshare.com/wiki/AlphaBot2-Pi.
[10] WiiBrew. 2022. Wii balance board. (Sept. 2022). Retrieved June 4, 2023 from https://wiibrew.org/wiki/Wii_Balance_Board.
[11] Wikipedia. 2023. Servomotor. (Jan. 2023). Retrieved June 4, 2023 from https://en.wikipedia.org/wiki/Servomotor.

A APPENDIX 1 - PHOTOGRAPH OF SYSTEM

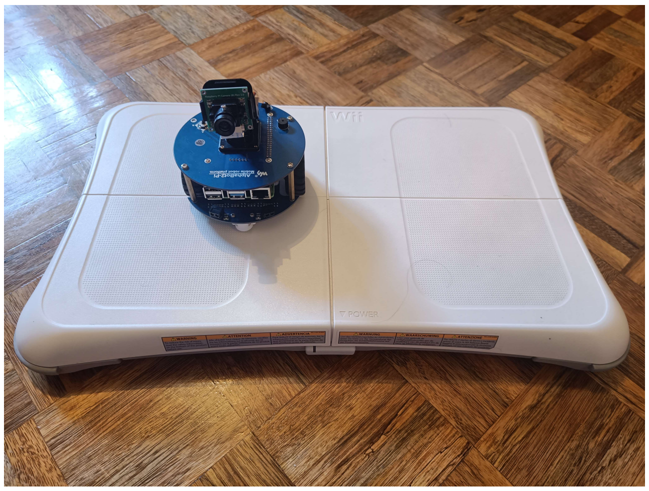

Figure 1: Alphabot and Wii Balance Board.