Design and Implementation of an Operating Systems Layer for a Low-Power Camera System with Multiple Modes

Kaustubh Sadekar
Portland State University

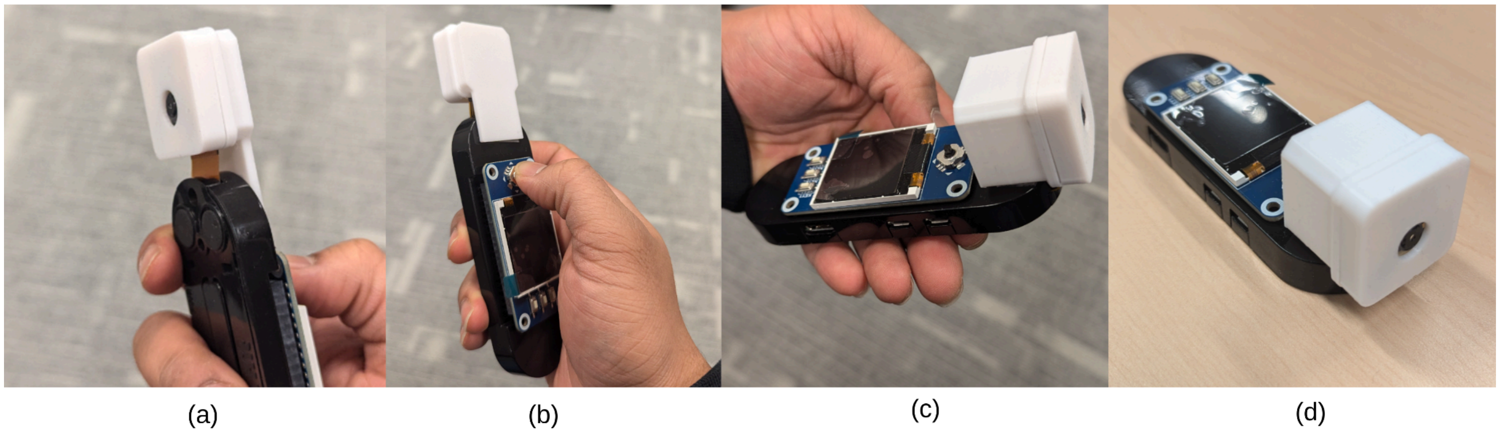

Figure 1. Hardware assembly of prototype 1. This hardware setup is used for initial debugging, setup, and configuration and does not include the battery. The next iteration will incorporate the battery UPS hat and a 3D-printed case.

Abstract

This project aims to develop an operating systems (OS) layer for a Raspberry Pi-based camera system with power-efficient features and multiple functional modes. The OS layer will manage multitasking, resource allocation, power optimization, and hardware abstraction for seamless operation. The system will support three modes—Image Capture, Video Capture, and Timelapse Capture—along with a Default Mode acting as a low-power viewfinder. The focus will be on task scheduling, interrupt handling, power-aware resource management, and real-time performance. A custom software image signal processor (software-ISP) will also be implemented using the *libcamera* library which will allow the dynamic selection of different ISP stages based on battery power status.

Hardware Description

1. Raspberry Pi Zero WH
    - Low-power single-board computer with GPIO support.
2. Raspberry Pi Camera (Version 2)
    - Capable of capturing high-resolution images and video.
3. Waveshare camera Module (M Type)
    - Compatible with Raspberry Pi Fisheye Lens Wider Field of View (200° angle of view)
4. 1.3-inch LCD Display HAT (Waveshare)
    - Used for live frame display and user interaction.
5. UPS Module for Raspberry Pi
    - Provides uninterrupted power supply and monitors battery levels.
6. Push Buttons and Joystick
    - For user input, mode selection, and system wakeup.

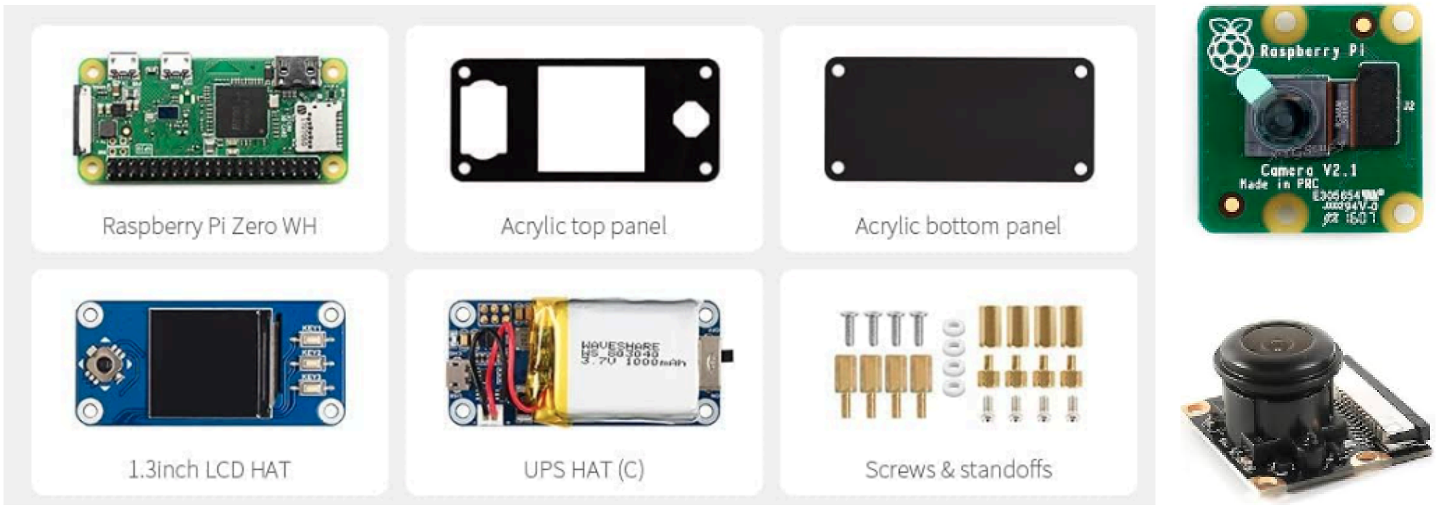

Figure 2. Hardware components for the low-power camera system. The current prototype uses Picam V2 however the final version will use a fisheye camera and the Software-ISP will also contain a fisheye undistortion module.

Project Goals

1. Develop an OS layer to:
    - Manage hardware interactions (camera, LCD, UPS, buttons).
    - Schedule tasks for real-time performance.
    - Optimize power consumption dynamically.
2. Implement three functional modes with an additional Default Mode:
    - **Default Mode:** Low-power live frame display.
    - **Mode 1 (Image Capture):** Capture high-resolution images based on battery level.
    - **Mode 2 (Video Capture):** Record video while balancing real-time display, ISP processing, and storage.
    - **Mode 3 (Timelapse Capture):** Efficient frame capture over extended periods.

OS Layer Design

Key Components of the OS Layer:

1. Task Scheduler:
    - Implements priority-based multitasking.
    - Manages tasks like frame capture, ISP processing, live display, and file saving.
2. Power Manager:
    - Monitors UPS battery levels in real-time.
    - Adjusts system behavior dynamically (e.g., frame rate, resolution, peripheral usage).
3. Interrupt Handler:
    - Handles GPIO-based user inputs for mode selection and wakeup signals.
4. State Manager:
    - Implements a state machine for transitions between Sleep, Default, and functional modes.
5. I/O Manager:
    - Manages efficient and fail-safe file operations for images and videos.
6. Hardware Abstraction Layer (HAL):
    - Provides interfaces for interacting with the camera, LCD, and UPS module.

System Design and Modes

Default Mode:

- Functionality: Low-power live viewfinder with minimal processing.
- Features:
  - Displays live frames at reduced resolution and frame rate.
  - Enters Sleep Mode after inactivity.
- Implementation Details:
  - Suspend non-essential peripherals in Sleep Mode.
  - Wake up via button interrupt.

Mode 1: Image Capture

- Functionality: Capture and process high-quality images.
- Features:
  - Use ISP stack for image processing (e.g., demosaicing, noise reduction).
  - Battery-aware resolution adjustment.
- Implementation Details:
  - Triggered from Default Mode via a button press.
  - Save processed images to disk.

Mode 2: Video Capture

- Functionality: Record video with real-time display and ISP processing.
- Features:
  - Two FPS options (high/low) based on battery level.
  - Compression of video frames using H.264.
- Implementation Details:
  - Multitask live display, ISP processing, and saving frames.
  - Schedule tasks using the OS layer.

Mode 3: Timelapse Capture

- Functionality: Capture frames at intervals for time-lapse video.
- Features:
  - Optimize frame capture and storage for long durations.
  - Dim or turn off the screen during recording.
- Implementation Details:
  - Adjust capture intervals dynamically based on power level.

Software Implementation

Tools and Libraries:

- Languages: Python for rapid prototyping; C/C++ for performance-critical tasks.
- Libraries:
  - `libcamera`: Camera control and ISP processing.
  - `RPi.GPIO` or `pigpio`: GPIO interrupt handling.
  - Framebuffer: LCD updates.

Development Workflow:

1. Implement the Default Mode and Sleep Mode transitions.
2. Develop and test individual functional modes.

3. Integrate the Task Scheduler and Power Manager.
4. Optimize ISP processing for performance and power efficiency.

Expected Outcomes

1. A functional camera system with:
    - Seamless mode transitions.
    - Efficient task scheduling and power management.
    - High-quality ISP processing.
2. A modular OS layer showcasing multitasking, resource management, and real-time performance.
3. Scope of open-sourcing the project in the future with a white paper. It could be an interesting lab toolkit for students to test OS-related concepts for a complete camera system (quick visual feedback, a lot of customization, and easy of use due to the hardware abstraction layer)

Current Progress

- Hardware assembly for Prototype 1 completed (Figure 1)
    - Customized 3D-printed camera covers
- `libcamera` was installed and Picam -v2 interface testing was completed
- LCD screen testing completed

Future Scope

This setup will be open-sourced in the future with additional utilities that will allow researchers and engineers to test custom ISP components or even extend the setup to adapt to other computational imaging experiments for topics like lensless cameras or under-display camera systems. It will also be a useful educational tool for computational imaging and computational photography courses.

References

1. Raspberry Pi Documentation (https://www.raspberrypi.org/documentation/)
2. libcamera Documentation (https://libcamera.org/)
3. GPIO Programming Guides (https://gpiozero.readthedocs.io/)