Heron Deterrent Solution Architecture

Design Document (Revised)

---

1. Overview

Herons are a primary predator for ornamental and commercial fish ponds. They typically land near a pond and approach slowly before striking. The goal of this solution is to **detect approaching herons in near real time and deter them automatically** using loud deterrent sounds, while continuously improving detection accuracy through a feedback and retraining loop.

This system is designed as an **edge-based computer vision solution**, running locally near the pond on low-cost hardware (Raspberry Pi or Coral Dev Board). All software components must be **open source and commercially usable**.

Key objectives:

- Detect herons reliably with low latency
- Trigger immediate deterrent actions
- Alert the user when herons or unknown objects are detected
- Capture and store detections for review
- Support manual labeling and future model retraining

---

2. Target Hardware

2.1 Supported Platforms

Phase 1 (Baseline / Development / PoC):

- Coral Dev Board
- Onboard Edge TPU for accelerated inference
- eMMC or USB SSD for storage

Phase 2 (Optimized / Cost-Reduced Production):

- Raspberry Pi 4 (4GB or higher)

- Optional Coral USB Accelerator
- MicroSD or USB SSD for storage

Phase 2 (Optimized / Production):

- Coral Dev Board or Raspberry Pi + Coral USB Accelerator

2.2 Peripherals

Webcam

- USB UVC-compatible webcam
- 720p or 1080p
- Mounted with a fixed field of view covering pond perimeter

External Speaker

- USB or 3.5mm speaker
- Capable of producing sudden, loud deterrent sounds

2.3 Environmental Considerations

- Weatherproof enclosure
- Passive cooling
- Future support for PoE or solar power (out of scope for initial release)

---

3. High-Level Architecture

The system runs primarily as a **single edge application**, composed of logical services that may later be decoupled using MQTT if scaling to multiple devices.

Core logical components:

- Video Capture & Motion Detection
- Object Detection (AI)
- Deterrent & Alerting Engine
- Data Storage
- Web User Interface
- Manual Labeling & Dataset Export

---

4. Video Capture & Motion Detection

4.1 Motion Detection

Motion detection is used to reduce unnecessary inference and storage.

Approach:

- OpenCV background subtraction (MOG2) or adaptive frame differencing
- Configurable sensitivity thresholds
- Masking of water surface areas to reduce false positives from ripples

4.2 Capture Strategy

When motion is detected:

- Capture a **short video clip (3–5 seconds)**
- Extract representative frames for AI inference
- Apply debounce logic to avoid repeated triggers from the same event

Captured media is forwarded to the Object Detection service and Storage service.

---

5. AI-Enabled Object Detection

5.1 Detection Model

The system uses **object detection (not classification)** to ensure bounding box support and future retraining.

Recommended Models:

- YOLOv5 or YOLOv8 (open source)
- TensorFlow Lite or Edge TPU variants for optimized inference

Supported Classes (initial):

- heron
- bird (generic)
- human
- cat
- unknown

5.2 Detection Logic

- Inference is run on captured frames
- Confidence threshold is configurable
- Detection tracking is used to suppress duplicate alerts for the same object

5.3 Deterrent Trigger

If a **heron** is detected:

- Select a deterrent WAV file at random from a configured catalog
- Play audio immediately through the speaker
- Trigger alerts and record the event

---

6. Data Storage & Retention

6.1 Storage Model

Media Storage:

- Local filesystem
- Structured directory layout

```None
data/
 ├── images/
 │    ├── heron/
 │    ├── other/
 │    └── unknown/
 ├── videos/
 ├── labels/
 └── models/
```

Metadata Database:

- SQLite (Phase 1)
- Upgradeable to PostgreSQL if remote/cloud storage is added

6.2 Retention Policy

- Media and metadata retained for **12 months**
- Automated pruning of expired data

---

7. Manual Object Detection & Labeling

7.1 Unclassified Objects

Objects that:

- Fall below confidence thresholds
- Do not match known classes

Are marked as **unclassified** and surfaced in the UI for review.

7.2 Labeling Interface

Users can:

- Draw bounding boxes on images
- Assign labels (e.g., heron / not heron)
- Edit or delete incorrect detections

7.3 Dataset Export

Labeled data is exported in **YOLO format**:

- `images/` – image files
- `labels/` – text files with bounding box metadata

This dataset is used for future offline retraining and fine-tuning of the detection model.

---

8. Messaging (MQTT)

MQTT is used as an **optional internal message bus**, particularly useful when scaling beyond a single device.

8.1 Example Topics

```None
heron/detection
heron/alert
heron/unclassified
heron/system/status
```

8.2 Initial Scope

- Local MQTT broker
- Used for decoupling detection, alerting, and UI components

---

9. Alerting

9.1 Alert Types

Severity	Description
INFO	System events, non-critical detections
WARNING	Unknown objects
CRITICAL	Heron detected

9.2 Alert Channels

- Web UI notifications
- SMS/Text message (CRITICAL alerts only)

9.3 Alert Controls

- Alert throttling (e.g., one SMS per X minutes)
- Quiet hours configuration

---

10. User Interface

The system provides a **local web-based UI** hosted on the edge device.

10.1 UI Features

1. Live webcam stream
2. Alert dashboard
3. Review heron detections
4. Review unknown / other detections
5. Manual labeling and bounding box editor
6. Manual deterrent trigger
7. Video upload for offline testing (MPEG4)
8. System configuration panel

10.2 UI Organization

- Live View
- Alerts
- Review & Label
- Model Management
- Settings

---

11. Configuration & Reliability

11.1 Configuration Options

- Detection sensitivity
- Confidence thresholds
- Sound volume
- Active detection hours
- Alert frequency limits

11.2 Fault Handling

- Camera disconnect detection
- Model load failure fallback
- Disk space monitoring
- Automatic service restart

---

12. Development Roadmap (Summary)

Phase 1 – Proof of Concept (Coral Dev Board)

- Motion detection
- Heron detection using Edge TPU
- Sound deterrent
- Local storage

Phase 2 – Core System (Raspberry Pi)

- Web UI
- Alerts
- Detection history

Phase 3 – ML Feedback Loop

- Manual labeling
- YOLO dataset export
- Model update workflow

Phase 4 – Hardening

- Performance optimization
- MQTT scaling
- Deployment readiness

---

13. Conclusion

This design provides a scalable, edge-first, and continuously improving heron deterrent system. By focusing on object detection, local processing, and user-driven model refinement, the solution balances effectiveness, cost, and long-term adaptability.