Parking Slot Detection System - Solution Methodology & Technical Approaches

Executive Summary

This document outlines the comprehensive methodology employed to solve the parking slot detection challenge. The solution combines multiple computer vision techniques, deep learning models, and advanced image processing algorithms to achieve robust and accurate parking space occupancy detection in various environmental conditions.

Problem Analysis

Core Challenges Identified

Environmental Variability

- Varying lighting conditions throughout the day
- Shadow interference from buildings and vehicles
- Weather-related visibility issues (rain, fog, bright sunlight)
- Different camera angles and perspectives

Vehicle Detection Complexity

- Partial vehicle occlusion
- Vehicles of different sizes, colors, and types
- Motorcycles and smaller vehicles that are harder to detect
- Vehicles parked imperfectly (crossing lines, diagonal parking)

Real-time Processing Requirements

- Need for consistent frame-rate processing
- Memory and computational efficiency
- Temporal consistency in detection results
- Smooth user experience in dashboard interface

Multi-Layered Solution Architecture

Layer 1: Deep Learning Foundation - YOLO Implementation

Primary Detection Engine: YOLOv8

The foundation of our solution employs the state-of-the-art YOLOv8 (You Only Look Once) object detection model, specifically chosen for its balance of accuracy and speed.

Why YOLOv8 was Selected:

- **Single-pass Detection**: Processes entire image in one forward pass, making it significantly faster than two-stage detectors
- **Real-time Performance**: Capable of processing 30+ FPS on standard hardware
- **Pre-trained COCO Weights**: Leverages Microsoft's COCO dataset with 80+ object classes including vehicles
- **Multi-scale Detection**: Effectively detects objects of various sizes within the same frame
- **Robust Architecture**: Handles partial occlusion and varying lighting conditions better than traditional methods

Model Configuration:

Model: YOLOv8s (Small variant for optimal speed-accuracy trade-off)
Input Resolution: 1280x1280 (higher than default 640 for better small object detection)
Confidence Threshold: 0.25 (balanced to catch more vehicles while minimizing false positives)
Vehicle Classes Detected: Cars (ID: 2), Motorcycles (ID: 3), Buses (ID: 5), Trucks (ID: 7)

Layer 2: Enhanced Multi-Method Detection Strategy

To overcome the limitations of single-method detection, we implemented a sophisticated multi-approach system:

Method 1: Standard YOLO Detection

- Direct application of YOLOv8 on original frames
- Serves as the primary detection baseline
- Optimized for well-lit, clear conditions

Method 2: Contrast-Enhanced Detection (CLAHE)

- **Technique**: Contrast Limited Adaptive Histogram Equalization
- **Purpose**: Improves detection in poorly lit areas and shadow regions
- **Implementation**: Applies CLAHE to L channel in LAB color space
- **Parameters**: Clip limit of 3.0 with 8x8 tile grid for optimal local enhancement

Method 3: Histogram Equalization Detection

- **Technique**: Global histogram equalization on luminance channel
- **Purpose**: Standardizes brightness across the entire frame

- **Application**: Particularly effective for backlit scenarios and extreme lighting conditions
- **Color Space**: YUV color space manipulation for better preservation of color information

Detection Fusion Strategy:

- All three methods run in parallel on each frame
- Results are combined using Non-Maximum Suppression (NMS) with IoU threshold of 0.5
- Duplicate detections are eliminated while preserving the highest confidence detection
- Final result includes the best detection from any method

Layer 3: Advanced Visual Analysis Fallback System

Purpose of Visual Analysis: Even the most sophisticated object detection models can miss vehicles under certain conditions. Our visual analysis system serves as an intelligent fallback mechanism.

Multi-Parameter Analysis Framework:

Edge Detection Component:

- **Algorithm**: Canny Edge Detection with adaptive thresholds (50-150)
- **Rationale**: Vehicles have distinct geometric edges that differ from empty asphalt
- **Metric**: Edge density calculation relative to parking space area
- **Weighting**: 35% contribution to final occupancy score

Shadow and Intensity Analysis:

- **Method**: Mean intensity calculation in grayscale
- **Theory**: Occupied spaces typically have lower average brightness due to vehicle shadows
- **Threshold**: Adaptive based on overall frame brightness
- **Calculation**: Intensity factor = max(0, (120 - mean_intensity) / 120)
- **Weighting**: 25% contribution to final occupancy score

Color Variance Detection:

- **Principle**: Vehicles introduce color diversity compared to uniform asphalt
- **Implementation**: Calculates variance across all color channels
- **Normalization**: Variance factor capped at 1.0 for stability
- **Weighting**: 25% contribution to final occupancy score

Texture Analysis:

- **Method**: Standard deviation of grayscale pixel intensities
- **Logic**: Vehicle surfaces have more textural variation than smooth asphalt

- **Normalization**: Divided by 255 for consistency
- **Weighting**: 15% contribution to final occupancy score

Combined Occupancy Score:

Final Score = (Edge Density × 0.35) + (Intensity Factor × 0.25) + (Variance Factor × 0.25) + (Texture Score × 0.15)
Occupied if Score > 0.42 (empirically determined threshold)

Layer 4: Temporal Smoothing and Consistency Framework

Challenge Addressed: Raw frame-by-frame detection can be noisy, with spaces flickering between occupied and free states due to detection uncertainty.

Temporal Smoothing Implementation:

Frame History Tracking:

- Maintains a rolling window of the last 5 frame decisions for each parking space
- Tracks both vehicle-based and visual-based detection results
- Stores detection confidence and method used

Majority Vote Decision Logic:

```
If total_frames >= 3:
    final_decision = occupied_frames >= (total_frames // 2 + 1)
Else:
final_decision = current_frame_decision
```

Benefits:

- Eliminates single-frame detection errors
- Provides stable transitions between states
- Maintains responsiveness while reducing noise
- Preserves rapid state changes when legitimate

Layer 5: Interactive Region of Interest (ROI) Definition

User-Centric Approach: Rather than attempting automatic parking space detection (which is unreliable across different parking lots), we implemented an intuitive manual selection system.

Interactive Selection Features:

Mouse-Based Drawing Interface:

- Click and drag rectangle selection
- Real-time visual feedback during selection
- Minimum size validation (30x30 pixels) to ensure meaningful regions

Quality Assurance Tools:

- 'n' key: Confirm current selection
- 'r' key: Reset current selection for re-drawing
- 'd' key: Delete last added space for corrections
- 'q' key: Finalize all selections

Persistent Configuration:

- Saves selections to JSON format for reuse
- Validates configuration file integrity
- Allows modification without full re-selection

Visual Feedback System:

- Green rectangles for confirmed spaces
- Blue rectangle for current selection
- Numbered labels for space identification
- Status text showing selection progress

Layer 6: Data Processing and Analysis Pipeline

Real-time CSV Generation: Our solution generates comprehensive frame-by-frame analysis data for post-processing and analytics.

Data Points Captured:

Frame-Level Metrics:

- Frame number and timestamp correlation
- Total vehicles detected across all methods
- Method-specific detection counts (Standard, Enhanced, Histogram Equalized)
- Overall parking lot occupancy statistics

Space-Level Detailed Analysis:

- Individual space occupancy status (occupied/free)
- Detection method used for each decision (vehicle_only, visual_only, vehicle+visual, empty)
- Confidence levels and detection source
- Temporal consistency indicators

Statistical Aggregation:

- Total spaces vs occupied spaces ratio

- Real-time occupancy rate calculation
- Method effectiveness tracking
- Detection confidence distributions

Layer 7: Frontend Visualization and User Interface

React-Based Dashboard Implementation:

Real-time Synchronization:

- Video playback synchronized with CSV data using frame rate calculations
- Dynamic statistics updates based on current video timestamp
- Smooth transitions between data points using interpolation

Component Architecture:

Video Display Component:

- HTML5 video element with custom controls
- Event listeners for timeupdate synchronization
- Support for multiple video formats and codecs

Statistics Visualization:

- Live updating cards for key metrics (Total, Occupied, Available)
- Animated progress bar for occupancy rate
- Color-coded indicators (Green: Available, Red: Occupied)
- Icon integration for intuitive understanding

Interactive Controls:

- Start/Stop video functionality
- Synchronized playback control
- Responsive design for various screen sizes

Advanced Technical Implementation Details

Non-Maximum Suppression (NMS) Algorithm

Purpose: Eliminate duplicate detections from multiple detection methods while preserving the best detection.

Algorithm Implementation:

1. **Bounding Box Collection**: Gather all detections from all methods

2. **Confidence Sorting**: Sort detections by confidence score (descending)
3. **IoU Calculation**: Calculate Intersection over Union for overlapping boxes
4. **Suppression Logic**: Remove detections with IoU > 0.5 with higher confidence detection
5. **Result Compilation**: Return filtered list of unique, high-confidence detections

Benefits:

- Eliminates redundant detections
- Preserves highest quality detection for each vehicle
- Prevents over-counting in occupancy calculations
- Maintains detection diversity across methods

Adaptive Thresholding Strategy

Dynamic Confidence Adjustment:

- Standard method: confidence = 0.25
- Enhanced methods: confidence = 0.20 (slightly lower to catch more marginal cases)
- Visual analysis: threshold = 0.42 (empirically optimized)

Rationale: Lower confidence thresholds for enhanced methods allow capture of vehicles that might be missed by standard detection, while the fusion process eliminates false positives through NMS.

Memory and Performance Optimization

Efficient Processing Techniques:

Frame Buffer Management:

- Limited buffer size for temporal smoothing (5 frames maximum)
- Automatic cleanup of old frame data
- Memory-efficient data structures

Model Optimization:

- Single model loading with reuse across methods
- Warm-up pass to optimize inference speed
- GPU acceleration when available (automatic CUDA detection)

Video Processing Efficiency:

- Streaming processing (frame-by-frame) to handle large videos
- Progress tracking and ETA calculation
- Multiple codec support for output compatibility

Algorithmic Innovation and Original Contributions

Hybrid Detection Fusion

Our solution's primary innovation lies in the intelligent fusion of:

- **Deep Learning Accuracy**: YOLO's sophisticated feature learning
- **Classical Computer Vision Robustness**: Edge detection and texture analysis
- **Temporal Intelligence**: Multi-frame consistency checking
- **User Domain Knowledge**: Manual ROI definition for perfect ground truth

Multi-Scale Visual Analysis

The visual analysis fallback system operates at multiple scales:

- **Pixel Level**: Intensity and color variance
- **Local Feature Level**: Edge density and texture
- **Spatial Level**: Region-based analysis within parking spaces
- **Temporal Level**: Consistency across multiple frames

Adaptive Learning Framework

While not implementing machine learning in the traditional sense, our system "learns" through:

- **Empirical Threshold Optimization**: Thresholds derived from extensive testing
- **Method Weighting**: Different weights for different analysis components
- **Temporal Adaptation**: Adjusting decision confidence based on recent history

Project Limitations and Resource Considerations

Time and Space Constraints

Due to the significant computational requirements and storage limitations, this project demonstration includes processed samples rather than the complete dataset. The full implementation generates substantially larger files that exceed typical sharing platform limitations.

Storage Considerations:

- Full HD video processing generates files of several gigabytes
- Complete CSV datasets can contain hundreds of thousands of rows for longer videos
- Multiple output formats and backup files require substantial disk space

Processing Time Requirements:

- Real-time processing of high-resolution videos requires significant computational resources

- Multi-method detection analysis increases processing time by approximately 3x compared to single-method approaches
- Model loading and warm-up phases add additional initialization time

Access to Complete Implementation

GitHub Repository

For the complete source code, full documentation, and latest updates, please visit our GitHub repository:

Repository Link:

https://github.com/AlaxNeon/ParkingSlotDetector_2

The repository contains:

- Complete source code with detailed comments
- Full implementation of all detection methods
- Additional utility scripts and tools
- Extended documentation and examples
- Issue tracking and community contributions

Google Drive Resources

Due to file size limitations in standard sharing platforms, the complete video datasets and processed outputs are available through Google Drive:

Drive Link:
https://drive.google.com/drive/folders/1IkyCA6gmjZ5qVOWQm5wOeIs_a4ih_Ffd?usp=sharing

Available Resources:

- **Original Video Files**: High-resolution parking lot footage used for testing
- **Processed Output Videos**: Complete enhanced videos with detection overlays
- **Full CSV Datasets**: Comprehensive frame-by-frame analysis data
- **Configuration Files**: Pre-configured parking space definitions for various scenarios
- **Additional Test Cases**: Multiple parking lot scenarios and lighting conditions

Accessing Full Resources

Recommended Workflow:

1. **Start with GitHub**: Clone the repository for the latest code and documentation
2. **Download Sample Data**: Use Google Drive links for full video datasets

3. **Follow Setup Instructions**: Complete installation guide available in repository README
4. **Test with Provided Data**: Use sample videos to verify your installation
5. **Apply to Your Videos**: Adapt the solution to your specific parking lot footage

GitHub Repository

https://github.com/AlaxNeon/ParkingSlotDetector_2

Disclaimer

This implementation is developed for educational and research purposes. All external libraries, models, and frameworks are used according to their respective licenses and terms of service. The parking space detection algorithms and integration methodology represent original work built upon established computer vision techniques and open-source tools.