Automated Parking Space Detection System

Complete Technical Documentation

Table of Contents

---

1. Project Overview

1.1 Introduction

The Automated Parking Space Detection System is a computer vision-based solution designed to identify and classify parking spaces as either **occupied** or **empty** in real-time. The system leverages machine learning techniques, specifically Support Vector Machines (SVM), combined with advanced image processing methods to achieve accurate parking space detection.

1.2 Key Features

- **Real-time Detection**: Live camera feed processing with immediate results
- **Interactive ROI Selection**: Mouse-based parking space region definition
- **High Accuracy Classification**: SVM-based machine learning model
- **Comprehensive Evaluation**: Detailed performance metrics and visualizations
- **Robust Image Processing**: Advanced preprocessing pipeline
- **Model Persistence**: Save and load trained models
- **Multi-mode Operation**: Training, testing, and real-time modes
- **Visual Feedback**: Color-coded bounding boxes for easy interpretation

1.3 Applications

- **Smart Parking Management**: Optimize parking lot utilization
- **Traffic Flow Improvement**: Reduce time spent searching for parking
- **Revenue Optimization**: Dynamic pricing based on availability

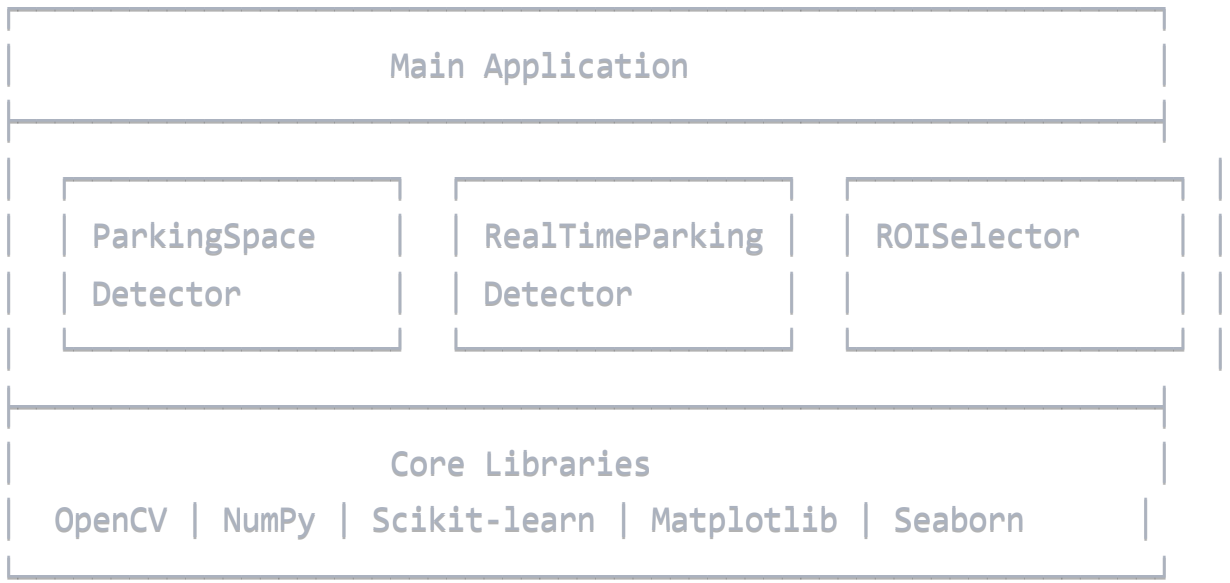
- **Urban Planning**: Data collection for parking infrastructure planning
- **Accessibility Enhancement**: Guide drivers to available spaces

---

2. System Architecture

2.1 Overall Architecture

The system follows a modular architecture with three main components:



2.2 Component Breakdown

2.2.1 ParkingSpaceDetector Class

- **Purpose**: Core detection and classification logic

- **Responsibilities**:
  - Dataset loading and parsing
  - Image preprocessing
  - Feature extraction
  - Model training and prediction
  - Performance evaluation

2.2.2 RealTimeParkingDetector Class

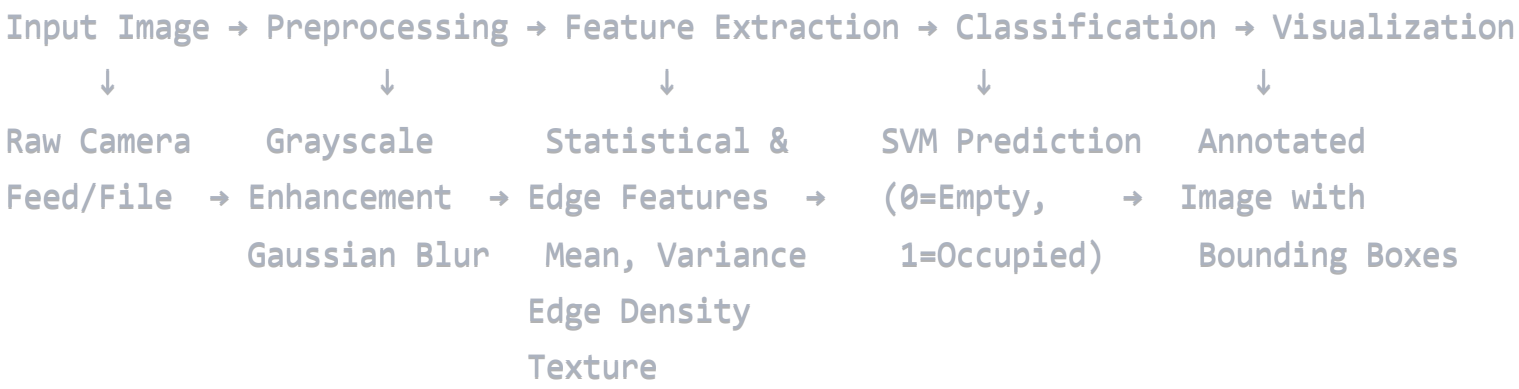
- **Purpose**: Real-time detection from camera feeds
- **Responsibilities**:
  - Camera interface management
  - Live frame processing
  - ROI management
  - Visual output generation

2.2.3 ROISelector Class

- **Purpose**: Interactive region of interest selection
- **Responsibilities**:
  - Mouse event handling
  - ROI drawing and management

- User interaction processing

2.3 Data Flow



---

3. Dataset Information

3.1 PKLot Dataset Overview

The PKLot dataset is a comprehensive collection of parking lot images specifically designed for parking space detection research.

3.1.1 Dataset Structure

```
PKLot/
├── PUCPR/
│   ├── Cloudy/
│   ├── Rainy/
│   └── Sunny/
├── UFPR04/
│   ├── Cloudy/
│   ├── Rainy/
│   └── Sunny/
└── UFPR05/
    ├── Cloudy/
    ├── Rainy/
    └── Sunny/
```

3.1.2 Data Characteristics

- **Total Images**: ~12,000 images
- **Parking Lots**: 3 different locations
- **Weather Conditions**: Cloudy, Rainy, Sunny
- **Image Resolution**: 1280×720 pixels
- **File Format**: JPEG images with XML annotations
- **Annotation Format**: Rotated rectangles with occupancy status

3.1.3 XML Annotation Structure

xml

```xml
<parking>
    <space id="1" occupied="0">
        <rotatedRect>
            <center x="123.5" y="456.7"/>
            <size w="50.0" h="25.0"/>
            <angle d="15.2"/>
        </rotatedRect>
    </space>
    <!-- More spaces... -->
</parking>
```

Annotation Fields:

- `id`: Unique identifier for each parking space
- `occupied`: 0 for empty, 1 for occupied
- `center`: Center coordinates (x, y)
- `size`: Width and height of the parking space
- `angle`: Rotation angle in degrees

3.2 Data Preprocessing

3.2.1 Coordinate Conversion

The system converts rotated rectangles to axis-aligned bounding boxes using:

```python
def convert_rotated_rect_to_bbox(cx, cy, w, h, angle):
    # Convert angle to radians
    angle_rad = np.deg2rad(angle)

    # Calculate corner points
    corners = compute_rotated_corners(cx, cy, w, h, angle_rad)

    # Get bounding box
    x_min, y_min = np.min(corners, axis=0)
    x_max, y_max = np.max(corners, axis=0)

    return [x_min, y_min, x_max, y_max]
```

3.2.2 Data Validation

- **Coordinate Bounds**: Ensure all coordinates are within image boundaries
- **Minimum Size**: Filter out parking spaces smaller than 10×10 pixels
- **XML Structure**: Validate proper XML format and required fields
- **Image Integrity**: Verify image files can be loaded properly

---

4. Technical Implementation

4.1 Image Preprocessing Pipeline

4.1.1 Preprocessing Steps

1. **Color Space Conversion**

   python

   ```
   gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
   ```

2. **Histogram Equalization**

   python

   ```
   equalized = cv2.equalizeHist(gray)
   ```

3. **Noise Reduction**

   python

   ```
   blurred = cv2.GaussianBlur(equalized, (5, 5), 0)
   ```

4.1.2 Benefits of Each Step

- **Grayscale Conversion**: Reduces computational complexity while preserving essential features
- **Histogram Equalization**: Enhances contrast and normalizes lighting conditions
- **Gaussian Blur**: Reduces noise while preserving important edge information

4.2 Feature Extraction

4.2.1 Statistical Features

1. **Mean Intensity**
   - Captures overall brightness of the parking space
   - Helps distinguish between dark (empty) and bright (car) regions
2. **Variance**
   - Measures intensity variation within the region
   - Higher variance often indicates presence of objects
3. **Texture (Standard Deviation)**
   - Quantifies texture complexity
   - Cars typically have more texture than empty asphalt

4.2.2 Edge-Based Features

1. **Sobel Edge Detection**

```python
sobelx = cv2.Sobel(region, cv2.CV_64F, 1, 0, ksize=3)
sobely = cv2.Sobel(region, cv2.CV_64F, 0, 1, ksize=3)
edge_magnitude = np.sqrt(sobelx**2 + sobely**2)
```

2. **Edge Density**

- Average edge magnitude within the region
- Cars typically have more edges than empty spaces

4.2.3 Feature Vector

Each parking space is represented by a 4-dimensional feature vector:

```
Feature Vector = [mean_intensity, variance, edge_density, texture]
```

4.3 Classification Algorithm

4.3.1 Support Vector Machine (SVM)

Configuration:

```python
SVC(
    kernel='rbf',           # Radial Basis Function kernel
    C=1.0,                  # Regularization parameter
    class_weight='balanced', # Handle class imbalance
    probability=True,       # Enable probability estimates
    random_state=42         # Reproducible results
)
```

Hyperparameters:

- **Kernel**: RBF kernel for non-linear classification
- **C Parameter**: Controls trade-off between margin maximization and classification error
- **Class Weight**: Automatically adjusts for imbalanced datasets
- **Probability**: Enables confidence scores for predictions

4.3.2 Feature Scaling

```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Feature scaling ensures all features contribute equally to the distance calculations in the SVM.

---

5. Models and Algorithms

5.1 Machine Learning Model

5.1.1 Support Vector Machine Details

Mathematical Foundation: The SVM finds the optimal hyperplane that separates occupied and empty parking spaces:

$$f(x) = \text{sign}(\Sigma\ \alpha_i y_i K(x_i, x) + b)$$

Where:

- α_i: Support vector coefficients
- y_i: Class labels (-1 or +1)
- $K(x_i, x)$: RBF kernel function
- b: Bias term

RBF Kernel Function:

$$K(x_i, x) = \exp(-\gamma||x_i - x||^2)$$

5.1.2 Model Training Process

1. **Data Collection**: Extract features from all training images
2. **Feature Scaling**: Normalize features using StandardScaler
3. **Model Fitting**: Train SVM on scaled features
4. **Cross-Validation**: Internal validation for hyperparameter tuning
5. **Model Persistence**: Save trained model and scaler

5.1.3 Prediction Process

1. **Feature Extraction**: Extract features from new image regions
2. **Feature Scaling**: Apply same scaling transformation

3. **Classification**: Use trained SVM to predict occupancy
4. **Probability Estimation**: Generate confidence scores

5.2 Performance Metrics

5.2.1 Classification Metrics

1. **Precision**: `TP / (TP + FP)`
   - Proportion of correctly predicted occupied spaces
2. **Recall**: `TP / (TP + FN)`
   - Proportion of actual occupied spaces correctly identified
3. **F1-Score**: `2 × (Precision × Recall) / (Precision + Recall)`
   - Harmonic mean of precision and recall
4. **Accuracy**: `(TP + TN) / (TP + TN + FP + FN)`
   - Overall classification accuracy

5.2.2 Confusion Matrix

```
                Predicted
                Empty  Occupied
Actual Empty    TN     FP
       Occupied FN     TP
```

---

6. Installation and Setup

6.1 System Requirements

6.1.1 Hardware Requirements

- **CPU**: Intel i5 or equivalent (minimum)
- **RAM**: 8GB minimum, 16GB recommended
- **Storage**: 5GB free space for dataset and models
- **Camera**: USB webcam or IP camera (for real-time detection)

6.1.2 Software Requirements

- **Operating System**: Windows 10/11, macOS 10.15+, or Linux Ubuntu 18.04+
- **Python**: Version 3.7 or higher
- **Package Manager**: pip (included with Python)

6.2 Installation Steps

6.2.1 Clone Repository

bash

```
git clone https://github.com/your-repo/parking-detection.git
cd parking-detection
```

6.2.2 Create Virtual Environment

bash

```
# Using venv
python -m venv parking_env
source parking_env/bin/activate  # On Windows: parking_env\Scripts\activate

# Using conda
conda create -n parking_env python=3.8
conda activate parking_env
```

6.2.3 Install Dependencies

bash

```
pip install -r requirements.txt
```

Required Packages:

```
opencv-python>=4.5.0
numpy>=1.19.0
scikit-learn>=1.0.0
matplotlib>=3.3.0
seaborn>=0.11.0
tqdm>=4.62.0
```

6.3 Dataset Setup

6.3.1 Download PKLot Dataset

1. Visit the official PKLot dataset website
2. Download the complete dataset (approximately 2GB)
3. Extract to a local directory

6.3.2 Verify Dataset Structure

```bash
python verify_dataset.py --dataset /path/to/pklot
```

---

7. Usage Guide

7.1 Training Mode

7.1.1 Basic Training

```bash
python parking_detection.py --mode train --dataset /path/to/pklot
```

7.1.2 Advanced Training Options

```bash
python parking_detection.py \
    --mode train \
    --dataset /path/to/pklot \
    --model custom_model.pkl \
    --output training_results/
```

Parameters:

- `--dataset`: Path to PKLot dataset directory
- `--model`: Output model filename (default: parking_model.pkl)
- `--output`: Directory for training results and visualizations

7.1.3 Training Output

The training process generates:

- Trained model file (`parking_model.pkl`)
- Training log with performance metrics
- Feature extraction statistics
- Model validation results

7.2 Testing Mode

7.2.1 Model Evaluation

```bash
python parking_detection.py \
    --mode test \
    --dataset /path/to/test_data \
    --model parking_model.pkl \
    --output test_results/
```

7.2.2 Test Output

- Annotated images with predictions
- Confusion matrix visualization
- Performance metrics report
- Empty spaces comparison chart

7.3 Real-Time Detection Mode

7.3.1 Camera Setup

```bash
python parking_detection.py --mode realtime --camera 0
```

7.3.2 Interactive Controls

Mouse Controls:

- **Click and Drag**: Select parking space regions
- **Multiple Selections**: Define multiple parking spaces

Keyboard Controls:

- **'q'**: Quit application
- **'r'**: Reset all selected regions

7.3.3 Real-Time Workflow

1. **Start Application**: Launch real-time mode
2. **Define ROIs**: Use mouse to select parking spaces
3. **Monitor Results**: View live classification results
4. **Adjust as Needed**: Reset and redefine regions if necessary

7.4 Custom Dataset Usage

7.4.1 Dataset Preparation

For custom datasets, ensure the following structure:

```
custom_dataset/
├── images/
│   ├── image001.jpg
│   ├── image002.jpg
│   └── ...
└── annotations/
    ├── image001.xml
    ├── image002.xml
    └── ...
```

7.4.2 XML Format Requirements

xml

```xml
<parking>
    <space id="1" occupied="0">
        <rotatedRect>
            <center x="x_coordinate" y="y_coordinate"/>
            <size w="width" h="height"/>
            <angle d="rotation_angle"/>
        </rotatedRect>
    </space>
</parking>
```

---

8. API Reference

8.1 ParkingSpaceDetector Class

8.1.1 Constructor

python

```python
detector = ParkingSpaceDetector(model_path="parking_model.pkl")
```

Parameters:

- `model_path` (str, optional): Path to save/load model file

8.1.2 Key Methods

load_pklot_dataset()

```python
image_paths, rois, labels = detector.load_pklot_dataset(dataset_path)
```

Purpose: Load and parse PKLot dataset **Returns**: Tuple of (image_paths, ROI_lists, label_lists)

train_classifier()

```python
detector.train_classifier(image_paths, rois, labels)
```

Purpose: Train SVM classifier on provided data **Parameters**: Image paths, ROI coordinates, ground truth labels

predict_spaces()

```python
predictions, probabilities = detector.predict_spaces(image, rois)
```

Purpose: Classify parking spaces in an image **Returns**: Predictions (list) and confidence probabilities (array)

save_model() / load_model()

```python
detector.save_model("model.pkl")
detector.load_model("model.pkl")
```

Purpose: Persist and restore trained models

8.2 RealTimeParkingDetector Class

8.2.1 Constructor

```python
rt_detector = RealTimeParkingDetector(detector, parking_rois=None)
```

Parameters:

- `detector`: Trained ParkingSpaceDetector instance
- `parking_rois` (list, optional): Predefined parking space regions

8.2.2 Key Methods

detect_from_camera()

python

```
rt_detector.detect_from_camera(camera_id=0, display=True)
```

Purpose: Start real-time detection from camera feed **Parameters**:

- `camera_id` (int): Camera device ID
- `display` (bool): Show live video feed

8.3 ROISelector Class

8.3.1 Usage

python

```
roi_selector = ROISelector()
cv2.setMouseCallback('window_name', roi_selector.mouse_callback)
```

Purpose: Handle interactive ROI selection through mouse events

---

9. Performance Evaluation

9.1 Evaluation Metrics

9.1.1 Standard Metrics

The system provides comprehensive evaluation through multiple metrics:

1. **Classification Accuracy**: Overall percentage of correct predictions
2. **Precision per Class**: Accuracy for each class (empty/occupied)
3. **Recall per Class**: Coverage for each class
4. **F1-Score**: Balanced measure of precision and recall
5. **Confusion Matrix**: Detailed breakdown of classification results

9.1.2 Visual Evaluations

1. **Annotated Images**: Visual verification of predictions
2. **Confusion Matrix Heatmap**: Easy interpretation of classification results
3. **Bar Charts**: Comparison of actual vs predicted empty spaces

9.2 Typical Performance

9.2.1 Expected Results

Based on PKLot dataset evaluation:

- **Overall Accuracy**: 85-92%
- **Precision (Occupied)**: 80-88%
- **Recall (Occupied)**: 75-85%
- **F1-Score**: 77-86%

9.2.2 Performance Factors

Positive Factors:

- Good lighting conditions
- Clear parking space boundaries
- Consistent camera angle
- Minimal occlusions

Challenging Conditions:

- Poor lighting or shadows
- Wet surfaces (reflections)
- Partial occlusions
- Unusual vehicle types

9.3 Performance Optimization

9.3.1 Feature Engineering

- **Additional Features**: Consider HOG, LBP, or color features
- **Feature Selection**: Remove irrelevant or redundant features
- **Dimensionality Reduction**: PCA for high-dimensional features

9.3.2 Model Improvements

- **Hyperparameter Tuning**: Grid search for optimal SVM parameters
- **Ensemble Methods**: Combine multiple classifiers
- **Deep Learning**: CNN-based approaches for complex scenarios

---

10. Troubleshooting

10.1 Common Issues

10.1.1 Dataset Loading Problems

Issue: "No images found in dataset" **Solution**:

- Verify dataset path is correct
- Ensure .jpg and .xml files are present
- Check file permissions

Issue: "XML parsing error" **Solution**:

- Validate XML file format
- Check for missing required fields
- Ensure coordinate values are numeric

10.1.2 Training Issues

Issue: "All features are zero" **Solution**:

- Verify image preprocessing pipeline
- Check ROI coordinates are within image bounds
- Ensure images are loading correctly

Issue: "Class imbalance warning" **Solution**:

- This is normal and handled by `class_weight='balanced'`
- Consider data augmentation if performance is poor

10.1.3 Real-Time Detection Issues

Issue: "Camera not accessible" **Solution**:

- Check camera is connected and not used by other applications
- Try different camera IDs (0, 1, 2, etc.)
- Verify camera permissions on your system

Issue: "Low detection accuracy" **Solution**:

- Ensure model is properly trained
- Check lighting conditions
- Verify ROI selections are accurate

10.2 Performance Issues

10.2.1 Slow Processing

Symptoms: High CPU usage, low frame rate **Solutions**:

- Reduce image resolution
- Optimize ROI sizes
- Use fewer parking spaces per frame
- Consider hardware acceleration

10.2.2 Memory Issues

Symptoms: Out of memory errors during training **Solutions**:

- Process dataset in batches
- Reduce image resolution
- Use data generators instead of loading all images

10.3 Debug Mode

10.3.1 Enable Debugging

```python
import logging
logging.basicConfig(level=logging.DEBUG)
```

10.3.2 Debug Information

- Feature extraction statistics
- Model training progress
- Prediction confidence scores
- ROI validation results

---

11. Future Enhancements

11.1 Planned Improvements

11.1.1 Technical Enhancements

1. **Deep Learning Integration**
   - CNN-based feature extraction
   - End-to-end deep learning models
   - Transfer learning from pre-trained models
2. **Advanced Computer Vision**

- Semantic segmentation for precise space boundaries
- Object detection for vehicle identification
- Multi-scale feature extraction

3. **Real-Time Optimization**
   - GPU acceleration support
   - Optimized inference pipeline
   - Parallel processing capabilities

11.1.2 Feature Additions

1. **Multi-Camera Support**
   - Simultaneous processing of multiple camera feeds
   - Camera calibration and synchronization
   - Panoramic parking lot coverage

2. **Advanced Analytics**
   - Parking duration tracking
   - Historical occupancy analysis
   - Predictive availability forecasting

3. **Integration Capabilities**
   - REST API for external applications

- Database integration for data persistence
- Mobile app connectivity

11.2 Research Directions

11.2.1 Algorithm Improvements

1. **Robustness Enhancement**
   - Weather condition adaptation
   - Lighting variation handling
   - Shadow and reflection mitigation
2. **Accuracy Improvements**
   - Ensemble learning methods
   - Active learning for continuous improvement
   - Domain adaptation techniques

11.2.2 Application Extensions

1. **Smart City Integration**
   - Traffic management system integration
   - Dynamic pricing optimization
   - Urban planning data collection

2. **Accessibility Features**
    - Handicapped space detection
    - Size-based parking classification
    - Emergency vehicle space monitoring

11.3 Contributing

11.3.1 Development Guidelines

- Follow PEP 8 coding standards
- Include comprehensive unit tests
- Document all public APIs
- Use type hints for function signatures

11.3.2 Testing Framework

```bash
# Run unit tests
pytest tests/

# Run integration tests
pytest tests/integration/

# Generate coverage report
pytest --cov=parking_detection tests/
```

Conclusion

The Automated Parking Space Detection System provides a robust, scalable solution for parking management applications. With its combination of classical computer vision techniques and machine learning algorithms, the system achieves reliable performance across various conditions while maintaining real-time processing capabilities.

The modular architecture ensures easy maintenance and extension, while the comprehensive evaluation framework enables continuous improvement and optimization. Whether used for research purposes or practical deployment, this system provides a solid foundation for parking space detection applications.

For additional support, feature requests, or contributions, please refer to the project repository and documentation resources.

Appendices

Appendix A: Complete API Documentation

[Detailed API reference with all classes, methods, and parameters]

Appendix B: Dataset Specifications

[Complete PKLot dataset structure and format specifications]

Appendix C: Mathematical Foundations

[Detailed mathematical descriptions of algorithms and methods]

Appendix D: Performance Benchmarks

[Comprehensive performance testing results and comparisons]

Appendix E: Configuration Examples

[Sample configuration files and usage examples]

Document Version: 1.0

Last Updated: June 2025

Authors: AI Development Team