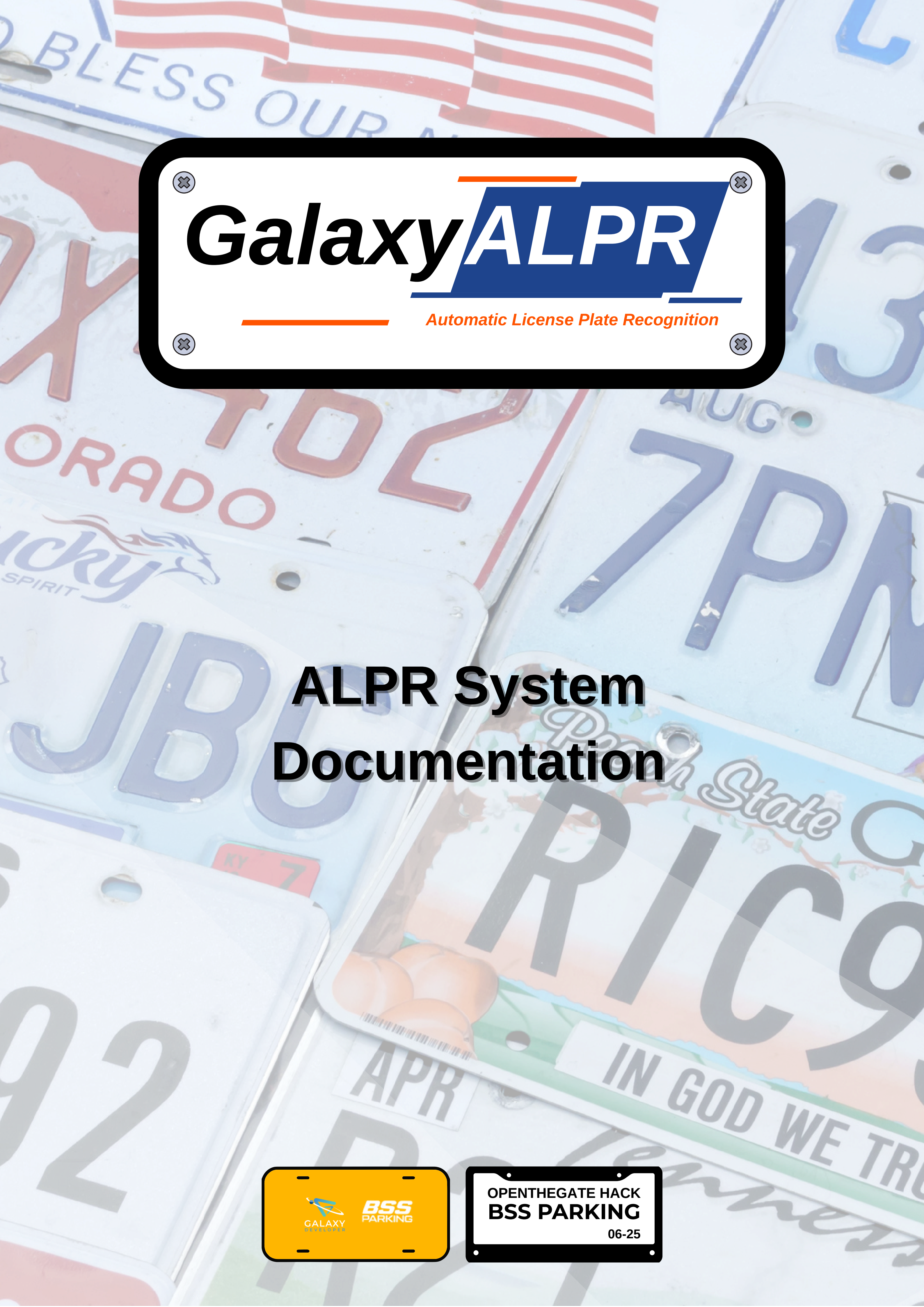

GalaxyALPR

Automatic License Plate Recognition

ALPR System Documentation

GALAXY DEVELOPER

BSS PARKING

OPENTHEGATE HACK
BSS PARKING
06-25

Galaxy ALPR Core Documentation

📋 Table of Contents

🚀 Quick Start

Galaxy ALPR Core is a comprehensive AI-powered Automatic License Plate Recognition system that combines advanced computer vision models with generative AI for accurate vehicle and license plate detection and recognition.

Installation

1. **Clone the repository**

   ```
   git clone <repository-url>
   cd galaxy_alpr_core
   ```

2. **Install dependencies**

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

3. **Set up environment variables** Create a `.env` file in the project root:

   ```
   GEMINI_API_KEY=your_gemini_api_key_here
   ```

4. **Download AI models** Place the following models in the `models/` directory:

- `model_vehicle_detector_yolo11n_v2.pt`
- `model_plate_detector_yolo11n_v3.pt`

Basic Usage

```
from galaxy_alpr_core.main import run_alpr_image

# Process a single image
```

```python
result = run_alpr_image("path/to/your/image.jpg")
print(result)
```

🏗️ System Architecture

Core Components

```
┌───────────────────────────────────────────────────────────────┐
│                       Galaxy ALPR Core                        │
├───────────────────────────────────────────────────────────────┤
│                         Entry Point                           │
│  ┌─────────────┐  ┌──────────────┐  ┌─────────────────────┐   │
│  │   Main.py   │──│ GalaxyALPR   │──│   ALPRAnalyzer      │   │
│  │ Entry Point │  │ Orchestrator │  │Vehicle-Plate Pairing│   │
│  └─────────────┘  └──────────────┘  └─────────────────────┘   │
├───────────────────────────────────────────────────────────────┤
│                     AI Detection Layer                        │
│  ┌─────────────┐       ┌──────────────┐       ┌─────────────┐ │
│  │   Vehicle   │   →   │    Plate     │   →   │    OCR      │ │
│  │  Detector   │       │   Detector   │       │ Recognition │ │
│  │ (YOLOv11n)  │       │ (YOLOv11n)   │       │(Gemini 2.0) │ │
│  └─────────────┘       └──────────────┘       └─────────────┘ │
├───────────────────────────────────────────────────────────────┤
│                      Processing layer                         │
│  ┌─────────────┐  ┌──────────────┐  ┌─────────────────────┐   │
│  │   Vehicle   │  │    Plate     │  │       Config        │   │
│  │Output Format│  │Output Format │  │       Manager       │   │
│  └─────────────┘  └──────────────┘  └─────────────────────┘   │
└───────────────────────────────────────────────────────────────┘
```

AI Models Used

1. Vehicle Detection Model

- **Model**: YOLOv11n.pt (v2)
- **Purpose**: Detects cars and motorcycles
- **Training Data**: 10,000 images
  - Training set: 7,000 images
  - Validation set: 2,000 images
  - Test set: 1,000 images
- **Classes**: Car, Motorcycle

2. Plate Detection Model

- **Model**: YOLOv11n.pt (v3)
- **Purpose**: Detects license plates
- **Training Data**: 10,000 images
  - Training set: 7,000 images
  - Validation set: 2,000 images
  - Test set: 1,000 images
- **Classes**: Plate

3. Plate OCR Model

- **Model**: Gemini 2.0 Flash-Lite
- **Purpose**: OCR for plate text extraction and attribute recognition
- **Provider**: Google AI Studio
- **Description**: Smallest and most cost-effective model, built for at-scale usage
- **Token Usage per Request**: ~1,834 tokens (1,715 prompt + 119 response)

Gemini 2.0 Flash-Lite Pricing & Limits:

Tier	Input Price	Output Price
Free Tier	Free of charge	Free of charge

Rate Limits (Free Tier):

- **Requests per minute (RPM)**: 30
- **Tokens per minute (TPM)**: 1,000,000
- **Requests per day (RPD)**: 1,500

Note: Rate limits are not guaranteed and actual capacity may vary

🔄 AI Workflow Pipeline

System Workflow Overview

```
Input Image → Vehicle Detection → Plate Detection → Plate Recognition → Pair Analysis
     ↓              ↓                  ↓                 ↓                    ↓
 [Original]    [YOLO 11n v2]      [YOLO 11n v3]    [Gemini 2.0 Flash]  [Smart Pairing]
                                                                              ↓
                                                                    ┌─ vehicle type
                                                                    ├─ plate number
                                                                    ├─ plate date
                                                                    ├─ plate type
                                                                    ├─ plate text color
                                                                    ├─ plate bg color
                                                                    └─ plate region
```

Visual Workflow Process

Galaxy ALPR Workflow

The system processes images through 5 main stages: Vehicle Detection using YOLO, Plate Detection using YOLO, OCR Recognition using Gemini AI, intelligent Vehicle-Plate pairing analysis, and structured output generation with comprehensive metadata.

Detailed Step-by-Step Process

Step 1: Vehicle Detection

- **Component**: `VehicleDetector`
- **AI Model**: YOLO 11n v2
- **Function**: Detects and crops vehicles (cars, motorcycles)
- **Input**: Original image
- **Output**: Vehicle bounding boxes, confidence scores, cropped vehicle images

```
# Configuration
vehicle_confidence_threshold: 0.25
vehicle_padding: 25 pixels
supported_classes: ["car", "motorcycle"]
```

Step 2: Vehicle Output Formatting

- **Component**: `VehicleOutputFormatter`
- **Function**: Structures vehicle detection results
- **Processing**: Assigns indices, formats paths, organizes metadata

Step 3: Plate Detection

- **Component**: `PlateDetector`

- **AI Model**: YOLO 11n v2
- **Function**: Detects license plates in vehicle images
- **Input**: Cropped vehicle images from Step 1
- **Output**: Plate bounding boxes, confidence scores, cropped plate images

```
# Configuration
plate_confidence_threshold: 0.25
plate_padding: 25 pixels
supported_classes: ["plate"]
```

Step 4: Plate Recognition (OCR)

- **Component**: `PlateRecognizer` + `PlateOCRGemini`
- **AI Model**: Google Gemini 2.0 Flash Lite
- **Function**: Extracts text and attributes from license plates
- **Advanced Features**:
  - Indonesian plate format recognition
  - Region code mapping (120+ regions)
  - Plate type classification (private, public, government, etc.)
  - Visual attribute detection (colors, icons, blue strips)

```
# OCR Output Structure
{
    "plate_number": "B 1234 CD",
    "plate_date": "08/29",
    "plate_text_color": "black",
    "plate_background_color": "white",
    "plate_icon": "government_seal",
    "plate_icon_color": "red",
    "plate_blue_strip": "no",
    "plate_type": "private",
    "confidence_score": 0.95,
    "plate_region_code": "B",
    "plate_region_name": "Jakarta/Metro Jaya"
}
```

Step 5: Results Combination

- **Function**: Merges detection and recognition data
- **Processing**: Combines plate detection metadata with OCR results

Step 6: Plate Output Formatting

- **Component**: `PlateOutputFormatter`
- **Function**: Structures final plate information with recognition data

Step 7: Vehicle-Plate Analysis & Pairing

- **Component**: `ALPRAnalyzer`
- **Function**: Intelligent pairing of vehicles with their license plates
- **Algorithms**:
  - **Containment Analysis**: Checks if plates are within vehicle bounding boxes
  - **Overlap Analysis**: Uses IoU (Intersection over Union) for partial overlaps
  - **Confidence Scoring**: Combines detection confidence with geometric relationship

```
# Pairing Methods
1. "containment" - Plate fully within vehicle box (preferred)
2. "overlap" - Plate partially overlaps with vehicle box
3. None - Unmatched vehicles or plates

# Detection Types
- "vehicle_with_plate" - Successfully paired
- "vehicle_without_plate" - Vehicle detected, no matching plate
- "plate_without_vehicle" - Plate detected, no matching vehicle
```

📚 API Reference

Core Classes

GalaxyALPR

Main orchestrator class for the complete ALPR pipeline.

```python
class GalaxyALPR:
    @staticmethod
    def run_alpr_image(input_image_path: str, timestamp: str = None) -> Dict:
        """
        Run complete ALPR pipeline on a single image.

        Args:
            input_image_path: Path to input image
            timestamp: Optional timestamp (YYYY-MM-DD_HH-MM-SS format)

        Returns:
            Complete ALPR results with vehicle-plate pairings
        """

    @staticmethod
    def detect_vehicle_from_image(input_image_path: str, timestamp: str = None) -> Dict:
        """Vehicle detection only"""

    @staticmethod
    def detect_plate_from_image(input_image_path: str, timestamp: str = None) -> Dict:
        """Plate detection only"""
    @staticmethod
    def ocr_plate_from_image(input_image_path: str) -> Dict:
        """OCR processing only"""

    @staticmethod
    def recognize_plate_from_image(input_image_path: str) -> Dict:
        """Complete plate recognition"""
```

VehicleDetector

YOLO-based vehicle detection component.

```
class VehicleDetector:
    def __init__(self, model_name: str = None):
        """Initialize with YOLO model"""

    def detect_vehicle_from_single_image(
        self,
        image: Union[str, np.ndarray],
        conf_threshold: float = None,
        padding: int = None,
        timestamp: str = ""
    ) -> Dict:
        """
        Detect vehicles in image.

        Returns:
            {
                "uploaded_image_path": str,
                "detected_vehicle_image_path": str,
                "list_cropped_vehicle_image_paths": List[str],
                "list_class_vehicle": List[str],
                "list_bounding_box_vehicle": List[List[int]],
                "list_confidence_score_vehicle": List[float],
                "vehicle_detection_processing_time": float
            }
        """
```

PlateDetector

YOLO-based license plate detection component.

```
class PlateDetector:
    def detect_plate_from_single_image(
        self,
        image: Union[str, np.ndarray],
        conf_threshold: float = None,
        padding: int = None,
        timestamp: str = ""
    ) -> Dict:
        """
        Detect license plates in image.

        Returns similar structure to VehicleDetector but for plates.
        """
```

PlateOCRGemini

Gemini AI-powered OCR for license plates.

```
class PlateOCRGemini:
    def __init__(self, model_name: str = None):
        """Initialize Gemini model"""

    def ocr_plate_from_single_image(
        self,
        image: Union[str, np.ndarray, Image.Image]
    ) -> List[dict]:
        """
        Extract text and attributes from single plate image.

        Returns:
            List of OCR results with comprehensive plate information
        """

    def ocr_plate_from_list_images(
        self,
        images: List[Union[str, np.ndarray, Image.Image]]
    ) -> List[dict]:
        """Batch processing of multiple plate images"""
```

ALPRAnalyzer

Intelligent vehicle-plate pairing system.

```
class ALPRAnalyzer:
    @staticmethod
    def analyze_vehicle_and_plate_pairing(
        vehicle_data: Dict,
        plate_data: Dict,
        timestamp: str
    ) -> Dict:
        """
        Analyze and pair vehicles with corresponding license plates.

        Uses containment analysis and overlap scoring for accurate pairing.

        Returns:
            {
                "timestamp": str,
                "uploaded_image_path": str,
                "detected_vehicle_image_path": str,
                "detected_plate_image_path": str,
                "processing_time": {
                    "vehicle_detection_ms": int,
                    "plate_detection_ms": int,
                    "plate_recognition_ms": int,
                    "total_ms": int
                },
                "summary": {
                    "total_detections": int,
                    "vehicles_with_plates": int,
                    "vehicles_without_plates": int,
                    "plates_without_vehicles": int
                },
                "detections": List[Detection]
            }
        """
```

Configuration Options

The system uses YAML configuration (`config/config.yaml`):

```
models:
  vehicle_detector: models/model_vehicle_detector_yolo11n_v2.pt
  plate_detector: models/model_plate_detector_yolo11n_v3.pt

ocr:
  provider: gemini
  model_name: gemini-2.0-flash-lite-001

detection:
  vehicle_confidence_threshold: 0.25
  plate_confidence_threshold: 0.25
  vehicle_padding: 25
  plate_padding: 25

output:
  save_detected_images: true
  uploaded_vehicle_image_dir: images_processed/uploaded_vehicle_images
  uploaded_plate_image_dir: images_processed/uploaded_plate_images
  detected_vehicle_image_dir: images_processed/detected_vehicle_images
  cropped_vehicle_image_dir: images_processed/cropped_vehicle_images
  detected_plate_image_dir: images_processed/detected_plate_images
  cropped_plate_image_dir: images_processed/cropped_plate_images
  results_dir: images_processed/results

image_formats:
  - .jpg
  - .jpeg
  - .png

timezone: Asia/Makassar
```

🛠️ Advanced Usage

Custom Model Integration

```python
# Use custom YOLO models
detector = VehicleDetector("path/to/custom/model.pt")

# Adjust detection parameters
result = detector.detect_vehicle_from_single_image(
    image="input.jpg",
    conf_threshold=0.5,  # Higher confidence
    padding=50           # More padding around detections
)
```

Batch Processing

```
import os
from galaxy_alpr_core.main import run_alpr_image

def process_directory(input_dir: str, output_dir: str):
    """Process all images in a directory"""
    for filename in os.listdir(input_dir):
        if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
            image_path = os.path.join(input_dir, filename)
            result = run_alpr_image(image_path)

            # Save results
            output_file = os.path.join(output_dir, f"{filename}_result.json")
            with open(output_file, 'w') as f:
                json.dump(result, f, indent=2)
```

Performance Tuning

```
# Optimize for speed vs accuracy
config_fast = {
    'detection': {
        'vehicle_confidence_threshold': 0.5,  # Higher threshold = faster
        'plate_confidence_threshold': 0.5,
        'vehicle_padding': 10,                # Less padding = faster
        'plate_padding': 10
    }
}

# Optimize for accuracy
config_accurate = {
    'detection': {
        'vehicle_confidence_threshold': 0.1,  # Lower threshold = more detections
        'plate_confidence_threshold': 0.1,
        'vehicle_padding': 50,                # More padding = better context
        'plate_padding': 50
    }
}
```

🧪 Examples & Use Cases

Example 1: Basic ALPR Processing

```python
from galaxy_alpr_core.main import run_alpr_image
import json

# Process image
result = run_alpr_image("sample_image.jpg")

# Print summary
print(f"Total detections: {result['summary']['total_detections']}")
print(f"Vehicles with plates: {result['summary']['vehicles_with_plates']}")
print(f"Processing time: {result['processing_time']['total_ms']}ms")

# Access individual detections
for detection in result['detections']:
    if detection['detection_type'] == 'vehicle_with_plate':
        vehicle = detection['vehicle']
        plate = detection['plate']
        print(f"Vehicle {vehicle['vehicle_class']}: {plate['plate_number']}")
        print(f"Confidence: {detection['pairing_confidence']}")
```

Example 2: Component-Level Usage

```python
from galaxy_alpr_core.VehicleDetector import VehicleDetector
from galaxy_alpr_core.PlateDetector import PlateDetector
from galaxy_alpr_core.PlateOCRGemini import PlateOCRGemini
```

```python
# Step-by-step processing
vehicle_detector = VehicleDetector()
plate_detector = PlateDetector()
ocr = PlateOCRGemini()

# 1. Detect vehicles
vehicles = vehicle_detector.detect_vehicle_from_single_image("input.jpg")

# 2. Detect plates in vehicle images
plates = plate_detector.detect_plate_from_single_image(
    vehicles['detected_vehicle_image_path']
)

# 3. OCR on plate images
for plate_path in plates['list_cropped_plate_image_paths']:
    ocr_result = ocr.ocr_plate_from_single_image(plate_path)
    print(f"Plate text: {ocr_result[0]['plate_number']}")
```

Example 3: Indonesian Plate Types

The system recognizes various Indonesian license plate types:

```python
# Private vehicle plates
"B 1234 CD"     # Jakarta private vehicle (white background, black text)

# Public transport
"B 7890 UP"     # Jakarta public transport (yellow background, black text)

# Government vehicles
"RI 1"          # Government vehicle (red background, white text)

# Police vehicles
"POLRI 1234"    # Police vehicle (black background, white text, police badge)
```

```
# Military vehicles
"TNI AU 1234"   # Air Force vehicle (black background, white text, military star)

# Diplomatic vehicles
"CD 1234 A"     # Diplomatic corps (white background, blue text, diplomatic emblem)
```

Example 4: Region Code Mapping

```python
# The system automatically maps region codes to province names
region_examples = {
    'B': 'Jakarta/Metro Jaya',
    'D': 'Jawa Barat (Bandung)',
    'L': 'Surabaya',
    'AA': 'Jawa Tengah (Magelang)',
    'DK': 'Bali',
    'PA': 'Papua'
}

# Access region information
for detection in result['detections']:
    if detection['plate']:
        plate = detection['plate']
        print(f"Plate: {plate['plate_number']}")
        print(f"Region: {plate['plate_region_name']}")
```

📊 Output Format

Complete Result Structure

```
{
  "timestamp": "2025-06-04T20:32:03Z",
  "uploaded_image_path": "images_processed/uploaded_vehicle_images/2025-06-04_20-32-03_upload
  "detected_vehicle_image_path": "images_processed/detected_vehicle_images/2025-06-04_20-32-0
  "detected_plate_image_path": "images_processed/detected_plate_images/2025-06-04_20-32-03_de
  "processing_time": {
    "vehicle_detection_ms": 150,
    "plate_detection_ms": 120,
    "plate_recognition_ms": 800,
    "total_ms": 1070
  },
  "summary": {
    "total_detections": 2,
    "vehicles_with_plates": 1,
    "vehicles_without_plates": 0,
    "plates_without_vehicles": 1
  },
  "detections": [
    {
      "detection_id": 1,
      "detection_type": "vehicle_with_plate",
      "pairing_method": "containment",
      "pairing_confidence": 0.92,
      "vehicle": {
        "vehicle_index": 1,
        "vehicle_class": "car",
        "vehicle_confidence_score": 0.95,
        "vehicle_bounding_box": [100, 150, 400, 300],
        "vehicle_image_path": "images_processed/cropped_vehicle_images/2025-06-04_20-32-03_de
      },
      "plate": {
        "plate_index": 1,
        "plate_class": "plate",
        "plate_confidence_score": 0.88,
        "plate_bounding_box": [180, 250, 280, 290],
        "plate_image_path": "images_processed/cropped_plate_images/2025-06-04_20-32-03_detect
        "plate_number": "B 1234 CD",
        "plate_date": "08/29",
        "plate_text_color": "black",
        "plate_background_color": "white",
        "plate_icon": "",
        "plate_icon_color": "",
        "plate_blue_strip": "no",
        "plate_type": "private",
        "confidence_score": 0.95,
        "plate_region_code": "B",
        "plate_region_name": "Jakarta/Metro Jaya"
      }
    }
  ]
}
```

🔧 Installation & Dependencies

Required Packages

```
# Python general packages
python-dotenv
python-multipart
pytz
PyYAML

# Web API
fastapi
uvicorn

# Image processing
opencv-python
numpy
pillow
```

```
# AI / Computer Vision
ultralytics
easyocr

# Generative AI
google.generativeai
```

Environment Setup

1. **Python Requirements**: Python 3.8+
2. **System Requirements**:
    - GPU recommended for faster YOLO inference
    - 8GB+ RAM recommended
    - Stable internet connection for Gemini API
3. **API Keys**:
    - Google Gemini API key required
    - Set in `.env` file as `GEMINI_API_KEY`

Directory Structure

```
galaxy_alpr_core/
├── config/
│   ├── config.yaml          # Main configuration
│   └── config.py            # Configuration loader
├── models/                  # AI model files
│   ├── model_vehicle_detector_yolo11n_v2.pt
│   └── model_plate_detector_yolo11n_v3.pt
├── images_processed/        # Auto-created output directories
```

```
│   ├── uploaded_vehicle_images/
│   ├── detected_vehicle_images/
│   ├── cropped_vehicle_images/
│   ├── uploaded_plate_images/
│   ├── detected_plate_images/
│   ├── cropped_plate_images/
│   └── results/
├── tools/
│   └── Timer.py            # Timestamp utilities
├── ALPRAnalyzer.py         # Vehicle-plate pairing logic
├── GalaxyALPR.py          # Main orchestrator
├── PlateDetector.py       # Plate detection
├── PlateOCRGemini.py      # Gemini OCR integration
├── PlateRecognizer.py     # Plate recognition pipeline
├── PlateOutputFormatter.py # Plate result formatting
├── VehicleDetector.py     # Vehicle detection
├── VehicleOutputFormatter.py # Vehicle result formatting
└── main.py                # Entry point and examples
```

🚨 Troubleshooting

Common Issues

1. **Model Loading Errors**

   ```
   FileNotFoundError: Cannot find model file
   ```

   Solution: Ensure model files are in the `models/` directory with correct names.

2. **Gemini API Errors**

```
EnvironmentError: "GEMINI_API_KEY" is missing
```

Solution: Set up `.env` file with valid Gemini API key.

3. **Low Detection Accuracy Solution**: Adjust confidence thresholds in config or use higher quality input images.

4. **Performance Issues Solution**:

    - Use GPU for YOLO inference
    - Reduce image resolution
    - Increase confidence thresholds
    - Reduce padding values

Performance Benchmarks

Processing Times by Component

Typical processing times on different hardware configurations:

Hardware	Vehicle Detection	Plate Detection	OCR Recognition	Total
CPU Only	800-1500ms	600-1200ms	800-1500ms	2.2-4.2s
GPU (RTX 3060)	50-150ms	40-120ms	800-1500ms	0.9-1.8s
GPU (RTX 4090)	20-80ms	15-60ms	800-1500ms	0.8-1.6s

Note: OCR time is primarily network-dependent due to Gemini API calls

Gemini API Usage Statistics

Typical Token Consumption per Image:

- Prompt tokens: ~1,715
- Response tokens: ~119
- Total tokens: ~1,834

Daily Processing Capacity (Free Tier):

- Maximum requests per day: 1,500 images
- Maximum tokens per day: 1,000,000 tokens
- Estimated processing capacity: ~545 images/day (based on token limits)
- Requests per minute limit: 30 images/minute

Free Tier Benefits:

- Input processing: Free of charge
- Output generation: Free of charge
- **Total cost per image: $0.00 USD**
- Cost per 1,000 images: $0.00 USD

Model Performance Metrics

Vehicle Detection (YOLOv11n v2):

- Training dataset: 10,000 images (7k train, 2k validation, 1k test)
- Classes: Car, Motorcycle
- Default confidence threshold: 0.25

Plate Detection (YOLOv11n v3):

- Training dataset: 10,000 images (7k train, 2k validation, 1k test)
- Classes: Plate

- Default confidence threshold: 0.25

Plate OCR (Gemini 2.0 Flash-Lite):

- Specialized for Indonesian license plates
- Supports 120+ regional codes
- Confidence scoring: 0.0-1.0 scale
- Advanced attribute recognition (colors, icons, plate types)

🤝 Contributing

Development Guidelines

1. **Code Style**: Follow PEP 8 conventions
2. **Documentation**: Add docstrings to all public methods
3. **Testing**: Test with various image types and quality levels
4. **Configuration**: Use config system for all parameters
5. **Error Handling**: Implement robust error handling and logging

Extending the System

To add new features:

1. **New Detection Models**: Extend `VehicleDetector` or `PlateDetector` classes
2. **Additional OCR Providers**: Implement new OCR classes following `PlateOCRGemini` pattern
3. **Custom Analysis**: Extend `ALPRAnalyzer` for specialized pairing logic
4. **Output Formats**: Create new formatter classes for different output requirements

📄 License

This project is developed and maintained by **@GalaxyDeveloper**.

🏷️ Citation

If you use Galaxy ALPR Core in your research or projects, please cite:

```
Galaxy ALPR Core - Advanced AI-Powered License Plate Recognition System
Developed by @GalaxyDeveloper (2025)
Built with YOLOv11n, Google Gemini AI, and intelligent pairing algorithms
```

Galaxy ALPR Core - Advanced AI-Powered License Plate Recognition System
Built with YOLOv11n, Google Gemini AI, and intelligent pairing algorithms

Developed by @GalaxyDeveloper - 2025