Github Link
https://github.com/AlaxNeon/ParkingSlotDetector_2

"""

Parking Space Detection System using YOLO Object Detection

ATTRIBUTION & ACKNOWLEDGMENTS:

- YOLO (You Only Look Once) algorithm: Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016)

- YOLOv5 implementation: Ultralytics (https://github.com/ultralytics/yolov5)

- Pre-trained COCO weights: Microsoft COCO Dataset (https://cocodataset.org)

- OpenCV library: https://opencv.org

- This implementation combines YOLO object detection with custom parking space management

ACADEMIC INTEGRITY NOTE:

This code uses publicly available pre-trained models and properly attributes all sources.

The parking space detection logic and ROI management is original implementation.

"""

```
import cv2
import numpy as np
import pandas as pd
from datetime import datetime
import json
import os
import torch
import csv


class YOLOParkingDetector:
    def __init__(self):
        self.parking_spaces = []
        self.model = None
        self.device = 'cpu'
        self.prev_status = {}  # smoothing memory
        self.status_consistency = {}  # track consistency over frames
```

```
    self.current_frame = None
    self.frame_buffer = []  # for temporal smoothing

def download_yolo_model(self):
    """Load YOLO model with enhanced error handling"""
    print("Setting up YOLO model...")
    try:
        from ultralytics import YOLO
        print("Using Ultralytics YOLOv8s")
        self.model = YOLO("yolov8s.pt")
        # Warm up the model
        dummy_frame = np.zeros((640, 640, 3), dtype=np.uint8)
        _ = self.model(dummy_frame, verbose=False)
        print("✓Model loaded and warmed up successfully")
        return True
    except Exception as e:
        print(f"Error loading YOLO: {e}")
        print("Run: pip install ultralytics torch torchvision")
        return False

def detect_vehicles_enhanced(self, frame, confidence_threshold=0.25):
    """Enhanced vehicle detection with multiple preprocessing techniques"""
    if self.model is None:
        return []

    vehicles = []

    # Method 1: Standard detection
    results = self.model(frame, imgsz=1280, verbose=False, conf=confidence_threshold)
    vehicles.extend(self._extract_vehicles_from_results(results, "standard"))

    # Method 2: Enhanced contrast detection (for poorly lit vehicles)
```

```
        enhanced_frame = self._enhance_contrast(frame)
        results_enhanced = self.model(enhanced_frame, imgsz=1280, verbose=False,
conf=confidence_threshold-0.05)
        vehicles.extend(self._extract_vehicles_from_results(results_enhanced, "enhanced"))

        # Method 3: Histogram equalized detection (for shadow areas)
        eq_frame = self._histogram_equalize(frame)
        results_eq = self.model(eq_frame, imgsz=1280, verbose=False, conf=confidence_threshold-0.05)
        vehicles.extend(self._extract_vehicles_from_results(results_eq, "histogram_eq"))

        # Remove duplicates using NMS
        vehicles = self._remove_duplicate_detections(vehicles)

        return vehicles

    def _extract_vehicles_from_results(self, results, method_name):
        """Extract vehicle information from YOLO results"""
        vehicles = []
        for result in results:
            if result.boxes is not None:
                for box in result.boxes:
                    conf = float(box.conf[0])
                    cid = int(box.cls[0])
                    # Vehicle class IDs in COCO: car=2, motorcycle=3, bus=5, truck=7
                    if cid in [2, 3, 5, 7]:
                        x1, y1, x2, y2 = box.xyxy[0].tolist()
                        vehicles.append({
                            "bbox": [int(x1), int(y1), int(x2), int(y2)],
                            "confidence": conf,
                            "class": result.names[cid],
                            "method": method_name
                        })
        return vehicles
```

```
def _enhance_contrast(self, frame):
    """Enhance frame contrast using CLAHE"""
    lab = cv2.cvtColor(frame, cv2.COLOR_BGR2LAB)
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
    lab[:, :, 0] = clahe.apply(lab[:, :, 0])
    return cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)

def _histogram_equalize(self, frame):
    """Apply histogram equalization"""
    yuv = cv2.cvtColor(frame, cv2.COLOR_BGR2YUV)
    yuv[:, :, 0] = cv2.equalizeHist(yuv[:, :, 0])
    return cv2.cvtColor(yuv, cv2.COLOR_YUV2BGR)

def _remove_duplicate_detections(self, vehicles, iou_threshold=0.5):
    """Remove duplicate detections using Non-Maximum Suppression"""
    if len(vehicles) <= 1:
        return vehicles

    boxes = np.array([v["bbox"] for v in vehicles], dtype=np.float32)
    scores = np.array([v["confidence"] for v in vehicles], dtype=np.float32)

    # Apply NMS
    indices = cv2.dnn.NMSBoxes(boxes.tolist(), scores.tolist(), 0.1, iou_threshold)

    if len(indices) > 0:
        indices = indices.flatten()
        return [vehicles[i] for i in indices]
    return vehicles

def select_parking_spaces(self, image_path):
    """Interactive parking space selection with enhanced UI"""
```

```
print("=== PARKING SPACE SELECTION MODE ===")
print("Instructions:")
print("• Click and drag to select each parking space")
print("• Press 'n' to confirm current selection")
print("• Press 'r' to reset current selection")
print("• Press 'd' to delete last space")
print("• Press 'q' to finish selection")

image = cv2.imread(image_path)
if image is None:
    print(f"Error: Could not load image {image_path}")
    return False

clone = image.copy()
drawing = False
start_point = None
current_rect = None

def mouse_callback(event, x, y, flags, param):
    nonlocal drawing, start_point, current_rect, image

    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        start_point = (x, y)

    elif event == cv2.EVENT_MOUSEMOVE and drawing:
        image = clone.copy()
        self._draw_existing_spaces(image)
        current_rect = (start_point[0], start_point[1], x - start_point[0], y - start_point[1])
        cv2.rectangle(image, start_point, (x, y), (255, 0, 0), 2)
        cv2.putText(image, f'Space {len(self.parking_spaces)+1} (selecting...)',
                    (start_point[0], start_point[1]-5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 1)
```

```
        elif event == cv2.EVENT_LBUTTONUP and drawing:
            drawing = False

    cv2.namedWindow('Select Parking Spaces', cv2.WINDOW_NORMAL)
    cv2.resizeWindow('Select Parking Spaces', 1200, 800)
    cv2.setMouseCallback('Select Parking Spaces', mouse_callback)

    while True:
        cv2.imshow('Select Parking Spaces', image)
        key = cv2.waitKey(1) & 0xFF

        if key == ord('n'):  # Confirm space
            if current_rect and abs(current_rect[2]) > 30 and abs(current_rect[3]) > 30:
                space = {
                    'id': len(self.parking_spaces) + 1,
                    'x': min(current_rect[0], current_rect[0] + current_rect[2]),
                    'y': min(current_rect[1], current_rect[1] + current_rect[3]),
                    'w': abs(current_rect[2]),
                    'h': abs(current_rect[3])
                }
                self.parking_spaces.append(space)
                print(f"✓Space {len(self.parking_spaces)} added: {space['w']}x{space['h']} at
({space['x']},{space['y']})")
                current_rect = None
                image = clone.copy()
                self._draw_existing_spaces(image)
            else:
                print("⚠ Please select a larger area (minimum 30x30 pixels)")

        elif key == ord('r'):  # Reset current selection
            image = clone.copy()
            self._draw_existing_spaces(image)
```

```
            current_rect = None
            print("Current selection reset")

        elif key == ord('d'):  # Delete last space
            if self.parking_spaces:
                deleted = self.parking_spaces.pop()
                print(f"✗ Deleted space {deleted['id']}")
                image = clone.copy()
                self._draw_existing_spaces(image)

        elif key == ord('q'):  # Quit
            break

    cv2.destroyAllWindows()

    if self.parking_spaces:
        with open('parking_spaces_config.json', 'w') as f:
            json.dump(self.parking_spaces, f, indent=2)
        print(f"\n✓ Saved {len(self.parking_spaces)} parking spaces to config file")
        return True
    else:
        print("No parking spaces selected!")
        return False

def _draw_existing_spaces(self, image):
    """Draw existing parking spaces on image"""
    for i, space in enumerate(self.parking_spaces):
        cv2.rectangle(image, (space['x'], space['y']),
                      (space['x'] + space['w'], space['y'] + space['h']),
                      (0, 255, 0), 2)
        cv2.putText(image, f'Space {i+1}',
                    (space['x'], space['y']-5),
```

```
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)

def load_parking_config(self, config_file="parking_spaces_config.json"):
    """Load parking configuration with validation"""
    try:
        with open(config_file, 'r') as f:
            self.parking_spaces = json.load(f)
        print(f"✓ Loaded {len(self.parking_spaces)} parking spaces from config")
        return True
    except FileNotFoundError:
        print(f"⚠ Config file {config_file} not found")
        return False
    except json.JSONDecodeError:
        print(f"⚠ Invalid JSON in {config_file}")
        return False

def is_parking_space_occupied_enhanced(self, vehicles, space, overlap_threshold=0.2):
    """Enhanced occupancy detection with temporal smoothing"""
    space_id = space['id']
    x1, y1 = space['x'], space['y']
    x2, y2 = x1 + space['w'], y1 + space['h']
    area = space['w'] * space['h']

    # Method 1: Vehicle overlap detection
    vehicle_detected = False
    best_vehicle = None
    best_overlap = 0

    for v in vehicles:
        vx1, vy1, vx2, vy2 = v['bbox']
        ix1, iy1 = max(x1, vx1), max(y1, vy1)
        ix2, iy2 = min(x2, vx2), min(y2, vy2)
```

```
    if ix1 < ix2 and iy1 < iy2:
        inter = (ix2 - ix1) * (iy2 - iy1)
        overlap_ratio = inter / area
        if overlap_ratio > overlap_threshold and overlap_ratio > best_overlap:
            vehicle_detected = True
            best_vehicle = v
            best_overlap = overlap_ratio

# Method 2: Visual analysis fallback
visual_occupied = False
if self.current_frame is not None:
    roi = self.current_frame[y1:y2, x1:x2]
    if roi.size > 0:
        visual_occupied = self._analyze_roi_occupancy(roi, area)

# Method 3: Temporal consistency check
current_decision = vehicle_detected or visual_occupied

# Initialize consistency tracker
if space_id not in self.status_consistency:
    self.status_consistency[space_id] = {'occupied_count': 0, 'free_count': 0, 'history': []}

# Update consistency tracker
consistency = self.status_consistency[space_id]
consistency['history'].append(current_decision)

# Keep only last 5 frames for smoothing
if len(consistency['history']) > 5:
    consistency['history'].pop(0)

# Calculate consistency score
occupied_frames = sum(consistency['history'])
```

```
    total_frames = len(consistency['history'])

    # Decision logic: require majority vote for status change
    if total_frames >= 3:
        final_decision = occupied_frames >= (total_frames // 2 + 1)
    else:
        final_decision = current_decision

    return final_decision, best_vehicle, vehicle_detected, visual_occupied

def _analyze_roi_occupancy(self, roi, area):
    """Analyze ROI for visual occupancy indicators"""
    if roi.size == 0:
        return False

    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

    # Edge detection for car outlines
    edges = cv2.Canny(gray, 50, 150)
    edge_density = np.sum(edges > 0) / area

    # Intensity analysis (darker = more likely occupied)
    mean_intensity = np.mean(gray)
    intensity_factor = max(0, (120 - mean_intensity) / 120)

    # Color variance (cars have more varied colors)
    color_variance = np.var(roi)
    variance_factor = min(color_variance / 5000, 1.0)

    # Texture analysis using standard deviation
    texture_score = np.std(gray) / 255.0
```

```
        # Combined occupancy score
        occupancy_score = (
            min(edge_density * 15, 0.35) +    # Edge contribution
            intensity_factor * 0.25 +         # Shadow contribution
            variance_factor * 0.25 +          # Color variance contribution
            texture_score * 0.15              # Texture contribution
        )

        return occupancy_score > 0.42

    def process_video(self, video_path, output_video="enhanced_yolo_parking.mp4",
csv_path="parking_analysis.csv"):
        """Enhanced video processing with improved detection"""
        if not self.parking_spaces:
            print("❌ No parking spaces defined! Run space selection first.")
            return

        if self.model is None:
            if not self.download_yolo_model():
                return

        cap = cv2.VideoCapture(video_path)
        if not cap.isOpened():
            print(f"❌ Could not open video: {video_path}")
            return

        fps = int(cap.get(cv2.CAP_PROP_FPS))
        w, h = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

        print(f"🎥 Processing video: {w}x{h} @ {fps}fps, {total_frames} frames")
```

```
# Try multiple codec options for best compatibility
codec_options = [
    ('X264', cv2.VideoWriter_fourcc(*'X264')),  # H.264 alternative
    ('XVID', cv2.VideoWriter_fourcc(*'XVID')),  # XVID codec
    ('MJPG', cv2.VideoWriter_fourcc(*'MJPG')),  # Motion JPEG
    ('mp4v', cv2.VideoWriter_fourcc(*'mp4v'))   # Original fallback
]

out = None
for codec_name, fourcc in codec_options:
    print(f"Trying codec: {codec_name}")
    out = cv2.VideoWriter(output_video, fourcc, fps, (w, h))
    if out.isOpened():
        print(f"✓ Successfully using {codec_name} codec")
        break
    else:
        out.release()

if out is None or not out.isOpened():
    print("❌ Could not initialize video writer with any codec")
    return

# Setup enhanced CSV output
csv_file = open(csv_path, 'w', newline='')
fieldnames = ['frame', 'timestamp', 'vehicles_detected', 'vehicles_standard', 'vehicles_enhanced',
              'total_spaces', 'occupied', 'free', 'occupancy_rate']
fieldnames += [f"space_{s['id']}" for s in self.parking_spaces]
fieldnames += [f"space_{s['id']}_method" for s in self.parking_spaces]
writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
writer.writeheader()

frame_count = 0
```

```
start_time = datetime.now()

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame_count += 1
    timestamp = frame_count / fps
    self.current_frame = frame.copy()

    # Enhanced vehicle detection
    vehicles = self.detect_vehicles_enhanced(frame)

    # Count vehicles by detection method
    standard_vehicles = len([v for v in vehicles if v.get('method') == 'standard'])
    enhanced_vehicles = len([v for v in vehicles if v.get('method') in ['enhanced', 'histogram_eq']])

    occupied = 0
    free = 0
    row = {
        "frame": frame_count,
        "timestamp": round(timestamp, 2),
        "vehicles_detected": len(vehicles),
        "vehicles_standard": standard_vehicles,
        "vehicles_enhanced": enhanced_vehicles,
        "total_spaces": len(self.parking_spaces)
    }

    # Process each parking space
    for space in self.parking_spaces:
        sid = space['id']
```

```
        is_occ, best_vehicle, vehicle_detected, visual_detected =
self.is_parking_space_occupied_enhanced(vehicles, space)

        # Determine detection method
        if vehicle_detected and visual_detected:
            method = "vehicle+visual"
        elif vehicle_detected:
            method = "vehicle_only"
        elif visual_detected:
            method = "visual_only"
        else:
            method = "empty"

        row[f"space_{sid}_method"] = method

        if is_occ:
            occupied += 1
            row[f"space_{sid}"] = "occupied"
            color = (0, 0, 255)  # Red
            status = "OCCUPIED"

            # Draw vehicle bounding box if detected
            if best_vehicle:
                vx1, vy1, vx2, vy2 = best_vehicle['bbox']
                cv2.rectangle(frame, (vx1, vy1), (vx2, vy2), (255, 0, 255), 2)
                cv2.putText(frame, f"{best_vehicle['class']} {best_vehicle['confidence']:.2f}",
                            (vx1, vy1-5), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (255, 0, 255), 1)
        else:
            free += 1
            row[f"space_{sid}"] = "free"
            color = (0, 255, 0)  # Green
            status = "FREE"
```

```
            # Draw parking space with enhanced visualization
            thickness = 3 if is_occ else 2
            cv2.rectangle(frame, (space['x'], space['y']),
                          (space['x'] + space['w'], space['y'] + space['h']), color, thickness)

            # Enhanced status text
            status_text = f"P{sid}: {status}"
            if method != "empty":
                method_short = {"vehicle+visual": "V+V", "vehicle_only": "VEH", "visual_only": "VIS"}
                status_text += f" ({method_short.get(method, method)})"

            cv2.putText(frame, status_text, (space['x'], space['y'] - 5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 1)

        # Update row data
        row["occupied"] = occupied
        row["free"] = free
        row["occupancy_rate"] = round((occupied / len(self.parking_spaces)) * 100, 1)
        writer.writerow(row)

        # Enhanced UI display
        cv2.putText(frame, f" 🅿️ ENHANCED PARKING DETECTION | Total: {len(self.parking_spaces)} |
Occupied: {occupied} | Free: {free}",
                    (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)

        cv2.putText(frame, f" 🚗 Vehicles: {len(vehicles)} (Std:{standard_vehicles}
Enh:{enhanced_vehicles}) | Frame: {frame_count}/{total_frames}",
                    (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (200, 200, 200), 1)

        cv2.putText(frame, f" ⏱️ Time: {timestamp:.1f}s | Occupancy: {row['occupancy_rate']}%",
                    (10, h - 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

        # Attribution watermark
```

```
            cv2.putText(frame, "YOLOv8: Ultralytics | Enhanced Detection",
                        (w - 280, h - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.3, (150, 150, 150), 1)

            out.write(frame)

            # Progress update
            if frame_count % 30 == 0 or frame_count == total_frames:
                elapsed = (datetime.now() - start_time).total_seconds()
                fps_actual = frame_count / elapsed if elapsed > 0 else 0
                progress = (frame_count / total_frames) * 100
                print(f"📊 Progress: {frame_count}/{total_frames} ({progress:.1f}%) | FPS: {fps_actual:.1f}")

        cap.release()
        out.release()
        csv_file.close()

        # Final summary
        processing_time = (datetime.now() - start_time).total_seconds()
        print(f"\n✅ Processing complete!")
        print(f"🎥 Output video: {output_video}")
        print(f"📊 CSV results: {csv_path}")
        print(f"⏱️ Processing time: {processing_time:.1f}s")
        print(f"🔄 Average FPS: {frame_count/processing_time:.1f}")

    def extract_reference_frame(video_path, output_path='reference_frame.jpg', frame_number=30):
        """Extract reference frame from video"""
        cap = cv2.VideoCapture(video_path)
        if not cap.isOpened():
            print(f"❌ Could not open video: {video_path}")
            return False

        cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
```

```
    ret, frame = cap.read()

    if ret:
        cv2.imwrite(output_path, frame)
        print(f"✅ Reference frame saved: {output_path}")
        cap.release()
        return True
    else:
        print("❌ Could not extract frame")
        cap.release()
        return False

def main():
    """Enhanced main function with better user experience"""
    print("="*70)
    print("🅿️ ENHANCED YOLO PARKING DETECTION SYSTEM")
    print("="*70)
    print("\n🔧 Features:")
    print("• Multi-method vehicle detection (Standard + Enhanced + Histogram Equalization)")
    print("• Visual analysis fallback for missed vehicles")
    print("• Temporal smoothing for stable results")
    print("• Real-time CSV output with detection methods")
    print("• Interactive parking space selection")
    print("\n📚 Attributions:")
    print("• YOLO Algorithm: Ultralytics YOLOv8")
    print("• COCO Dataset: Microsoft")
    print("• Enhanced detection logic: Original implementation")
    print("="*70)

    detector = YOLOParkingDetector()

    # Get video path
```

```
video_path = "Cam1.mp4"
if not os.path.exists(video_path):
    print(f"❌ Video file not found: {video_path}")
    return

# Load or create parking configuration
if os.path.exists('parking_spaces_config.json'):
    use_existing = input("\n🔧 Found existing parking configuration. Use it? (y/n): ").lower()
    if use_existing == 'y':
        if not detector.load_parking_config():
            print("❌ Failed to load config")
            return
    else:
        print("📸 Extracting reference frame...")
        if extract_reference_frame(video_path):
            if not detector.select_parking_spaces('reference_frame.jpg'):
                return
        else:
            return
else:
    print("📸 Extracting reference frame for parking space selection...")
    if extract_reference_frame(video_path):
        if not detector.select_parking_spaces('reference_frame.jpg'):
            return
    else:
        return

if not detector.parking_spaces:
    print("❌ No parking spaces configured!")
    return

# Process video
```

```
    output_video = input("\n 🎥 Output video name (default: enhanced_yolo_parking.mp4): ").strip()
    if not output_video:
        output_video = "enhanced_yolo_parking.mp4"

    csv_output = input(" 📊 CSV output name (default: parking_analysis.csv): ").strip()
    if not csv_output:
        csv_output = "parking_analysis.csv"

    print(f"\n 🚀 Starting enhanced processing...")
    detector.process_video(video_path, output_video, csv_output)

    print(f"\n 🎉 All done! Check your output files:")
    print(f" 🎥 Video: {output_video}")
    print(f" 📊 Data: {csv_output}")
    print(f" ⚙️ Config: parking_spaces_config.json")

if __name__ == "__main__":
    main()
```