Loyola Institute of Technology and Science

DEPARTMENT OF

COMPUTER SCIENCE ENGINEERING

Completed the Project named as

Capstone Project-Autonomous Drone Vision System

Gesture Recognition System

Smart Parking Detection System

Submitted by

Jeya Rizafa J S-961222104025

Loyola
Institute of Technology and Science

LOYOLA NAGAR, THOVALAI,
KANYAKUMARI DISTRICT - 629302.

EMPOWERMENT through KNOWLEDGE
LOYOLA INSTITUTIONS
ARISE AND SHINE

RECORD NOTE BOOK

NM1093- Open CV

Register No:

DEPARTMENT OF COMPUTER SCIENCE
& ENGINEERING

Register No:

Certified that, this is the bonafide record of work done by Mr/Ms……………………………………………………………….. of VI Semester Computer Science and Engineering of this college, in the ***Open CV (NM1093)*** *during 20 -20 in partial fulfillment of the requirements of the B.E. Degree course of the Anna University, Chennai.*

Staff-in-charge **Head of the Department.**

This record is submitted for the University Practical Examination held on…………………………..

Internal Examiner **External Examiner**

Abstract

In today's era of automation and artificial intelligence, computer vision plays a vital role in enabling machines to "see" and interpret their surroundings. This capstone project showcases the practical implementation of computer vision using OpenCV, a widely-used open-source library. The project includes three key systems: an Autonomous Drone Vision System, a Gesture Recognition System, and a Smart Parking Detection System. These applications demonstrate the versatility of OpenCV in addressing real-time challenges across domains such as robotics, human-computer interaction.

The Autonomous Drone Vision System replicates the visual processing of drones, enabling them to detect and track objects for autonomous navigation. The Gesture Recognition System allows users to control applications with hand gestures, creating a touch-free interface that is especially relevant in modern user environments. The Smart Parking Detection System automates parking slot monitoring by analyzing camera footage.

Through the integration of live video processing, object detection, and motion analysis, this project exemplifies the power of OpenCV in building intelligent systems. Each module is developed in Python, with real-time outputs and user-friendly interfaces, proving that complex vision-based solutions can be built efficiently using accessible tools.

Introduction

The ability of machines to interpret visual information just as humans do is one of the most transformative advancements in modern computing. Computer vision, a subfield of artificial intelligence, enables machines to extract, analyze, and understand visual data from the world. Whether it's facial recognition, medical imaging, or autonomous vehicles growing.

One of the most powerful tools for building computer vision applications is OpenCV (Open Source Computer Vision Library). Developed by Intel, OpenCV is an open-source library that provides a comprehensive suite of tools for image and video processing, object detection, pattern recognition, and machine learning. Its real-time capabilities and compatibility with Python have made it a top choice for developers, students, and researchers alike.

This capstone project is an exploration of OpenCV's capabilities through the development,

1. **Autonomous Drone Vision System** – A simulation of visual tracking and object-following behavior in drones, using real-time video input and motion analysis.
2. **Gesture Recognition System** – A user-friendly interface that recognizes and interprets hand gestures for controlling applications without physical touch.
3. **Smart Parking Detection System** – A monitoring system that identifies empty and occupied parking spaces from a video feed using image processing techniques.

Each of these systems reflects a different application of OpenCV: Object Detection in drones, Contour and Gesture Analysis in hand movement, and Region of Interest (ROI) Monitoring in parking detection. These projects aim to not only demonstrate OpenCV's technical strengths but also its practical usability in solving real-world problems. As we move toward a future increasingly reliant on automation and intelligent systems, mastering tools like OpenCV is essential for creating impactful, responsive, and scalable solutions.

Open CV

Aim:

The aim of this project is to design and implement three computer vision-based systems using OpenCV: an Autonomous Drone Vision System, a Gesture Recognition System, and a Smart Parking Detection System. The Autonomous Drone Vision System is developed to simulate intelligent drone functionalities such as object detection, tracking, and navigation using real-time video input. The Gesture Recognition System is intended to detect and interpret hand gestures for touchless application control. The Smart Parking Detection System is designed to identify vacant and occupied parking slots from video feeds to support real-time parking management.

1. Autonomous Drone Vision System

Description:

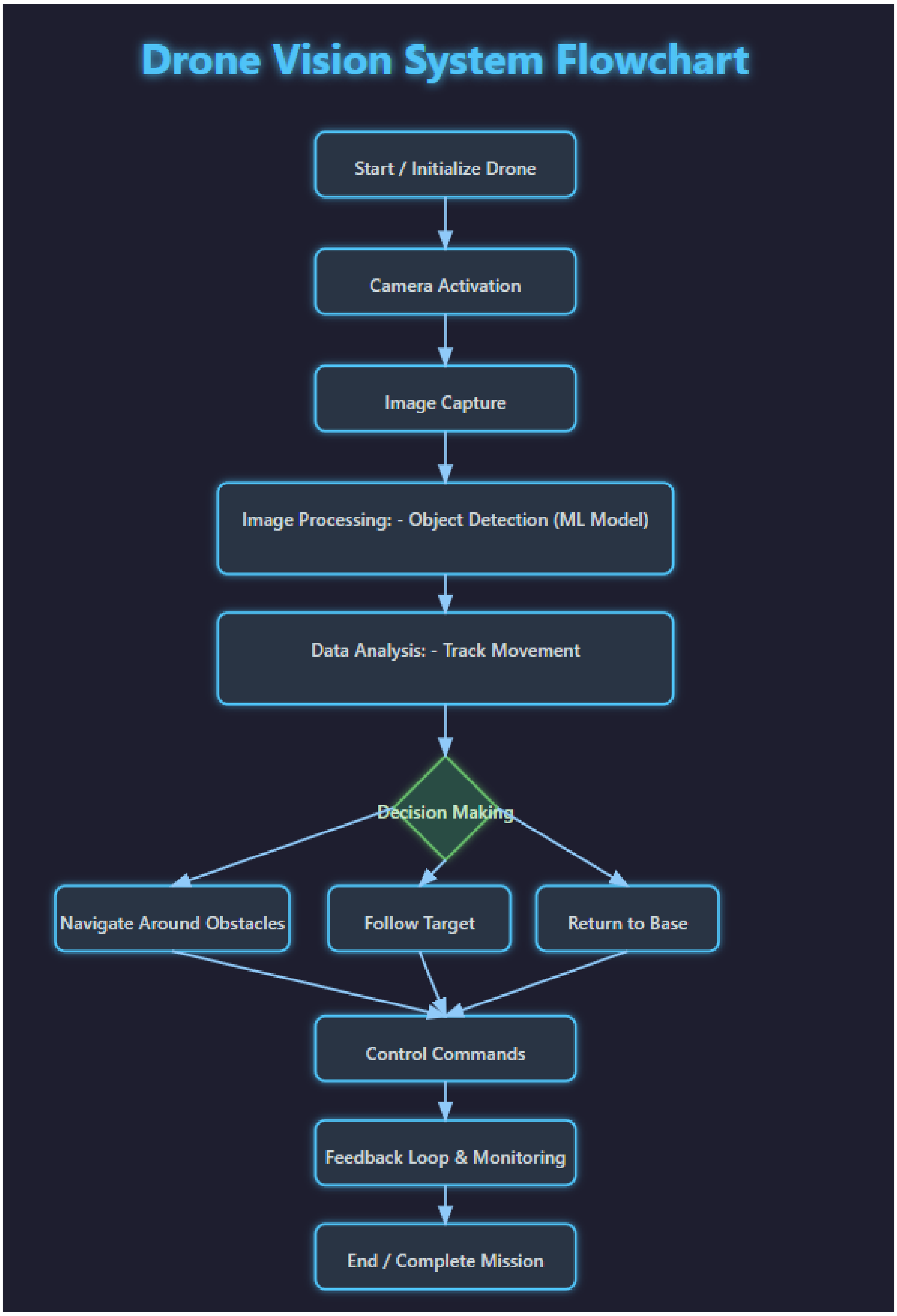
The Autonomous Drone Vision System is a computer vision application that enables drones to perform intelligent tasks like object detection, tracking, and navigation. It mimics the human visual system using a camera feed and processes it in real-time to make decisions. This system is useful in surveillance, agriculture, and delivery services where autonomous operation is required.

Design Strategy:

- Simulate drone vision using a webcam or video input.
- Convert frames to grayscale or HSV for better detection accuracy.
- Use Haar cascade classifiers or color-based filtering to detect specific objects (e.g., humans or signs).
- Track the object using bounding boxes or contours.
- Analyze object position and simulate drone actions (e.g., turning, following).

- Optional: Integrate edge detection or obstacle avoidance logic.

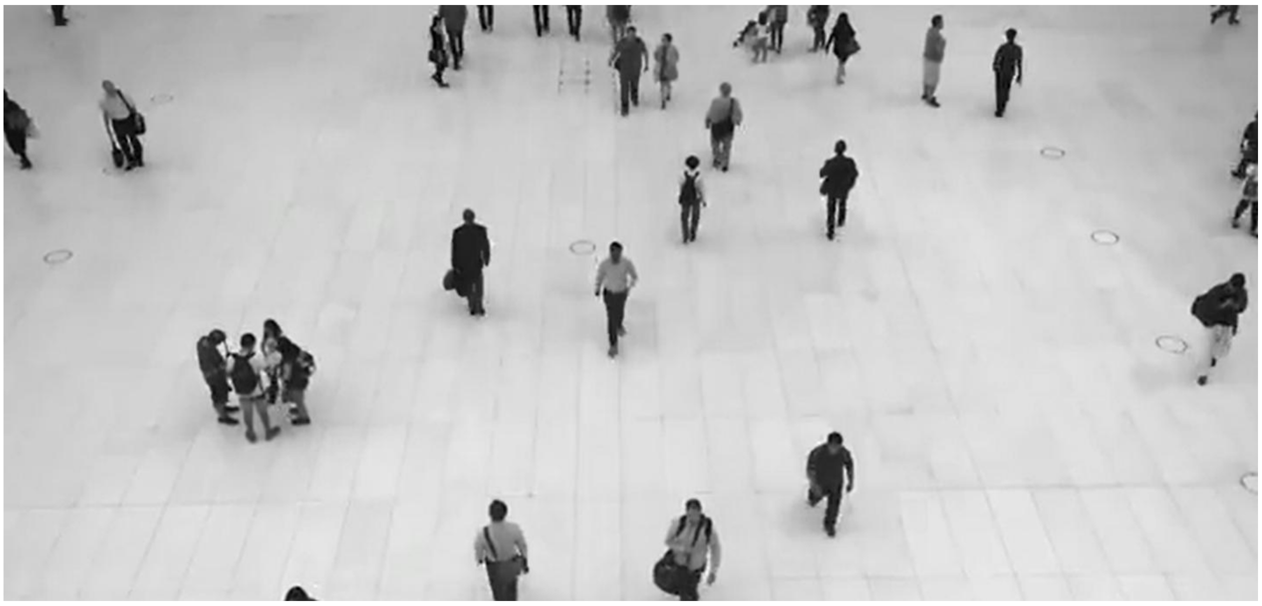
Block diagram:


Drone Vision System Flowchart
Start / Initialize Drone
Camera Activation
Image Capture
Image Processing: - Object Detection (ML Model)
Data Analysis: - Track Movement
Decision Making
Navigate Around Obstacles
Follow Target
Return to Base
Control Commands
Feedback Loop & Monitoring
End / Complete Mission

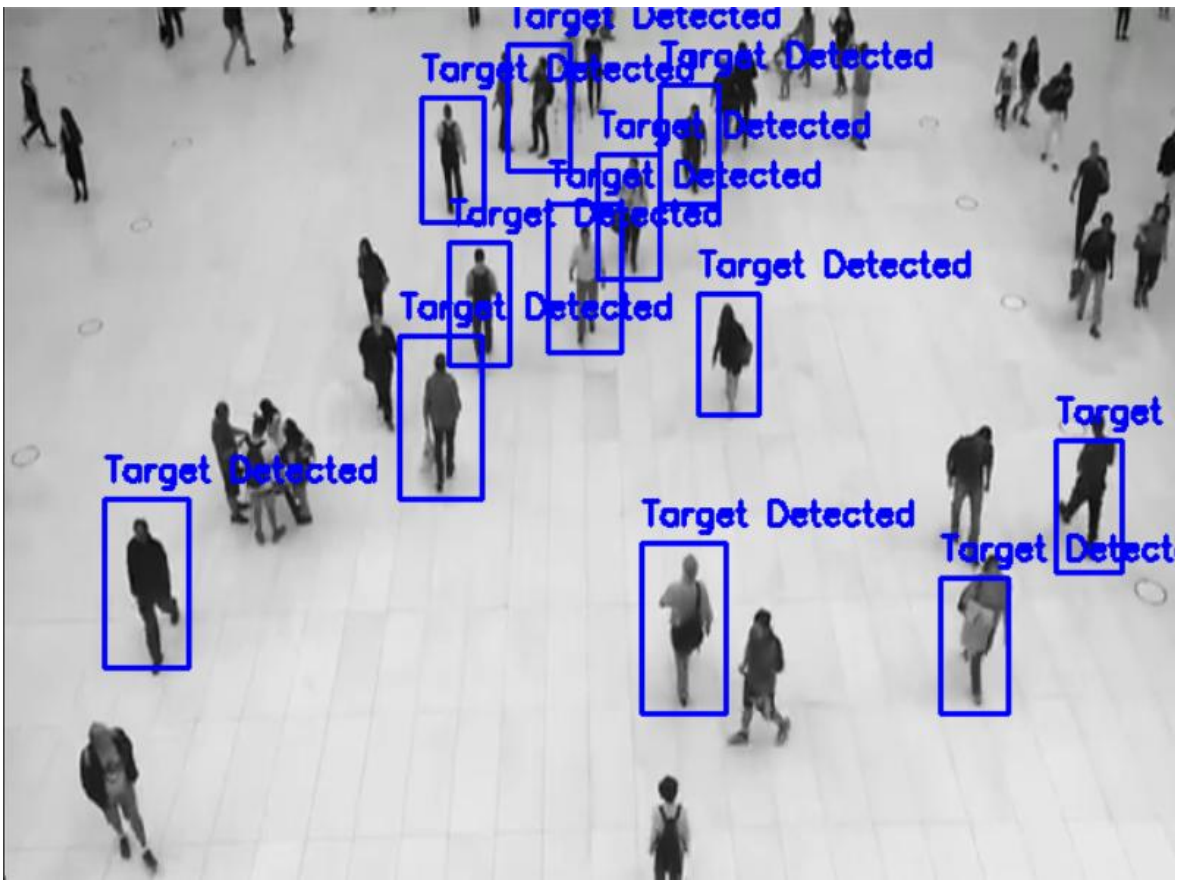
Program:

```
import cv2
cap = cv2.VideoCapture('Drone.mp4')

if not cap.isOpened():
print("Error: Cannot open video source.")
exit()

object_detector = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_fullbody.xml')

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

frame = cv2.resize(frame, (640, 480))
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
objects = object_detector.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5)
for (x, y, w, h) in objects:
cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
cv2.putText(frame, "Target Detected", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)
cv2.imshow('Autonomous Drone Vision System', frame)
if cv2.waitKey(30) & 0xFF == ord('q'):
break
cap.release()
cv2.destroyAllWindows()
```

Input:



Output:



Applications:

Surveillance & Security: Real-time monitoring of restricted or dangerous areas without human presence.

Agriculture: Monitoring crop health, detecting pests or weeds, and analyzing soil conditions.

Disaster Management: Locating victims in hazardous environments or during natural calamities.

Package Delivery: Navigation for autonomous parcel delivery in urban and rural areas.

Traffic Monitoring: Aerial traffic analysis and congestion detection.

2. Gesture Recognition System

Description:

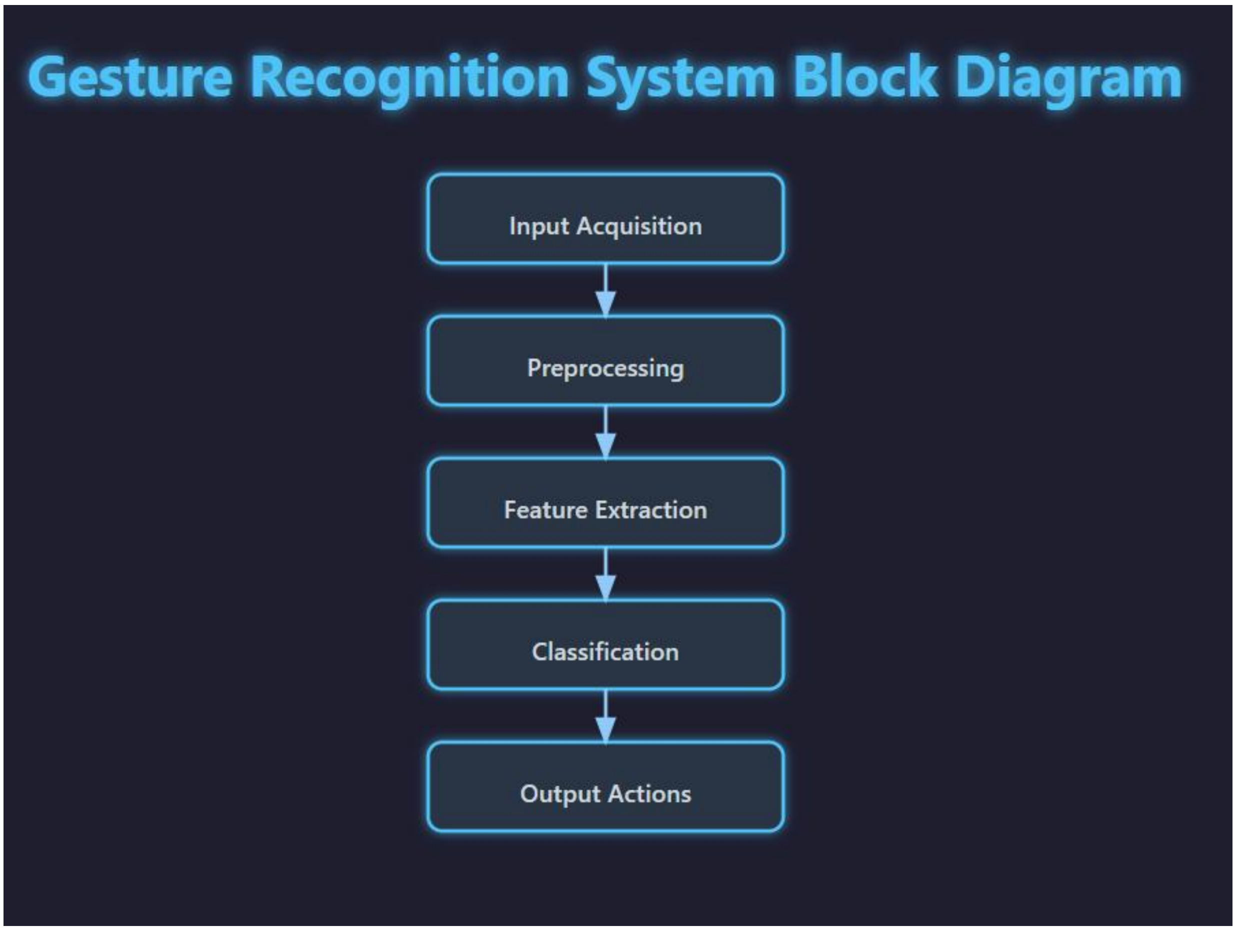
The Gesture Recognition System is designed to identify and interpret hand gestures captured through a webcam. It helps create touchless interfaces for controlling applications, especially useful in gaming, sign language interpretation, and smart devices. The system focuses on detecting the hand region, analyzing its shape, and recognizing specific gestures such as open palm, fist, or finger count.

Design Strategy:

- Capture real-time video from the webcam.
- Apply background subtraction or color filtering to isolate the hand region.
- Find contours and draw a convex hull around the hand.
- Use convexity defects to count fingers and classify gestures.
- Assign each gesture to a command or action (e.g., "5 fingers = Play").

- Improve stability with noise reduction and frame smoothing techniques.

Block diagram:

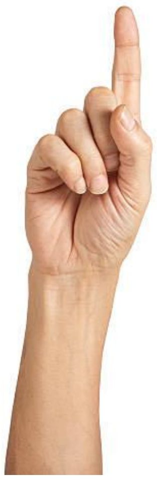


Program:

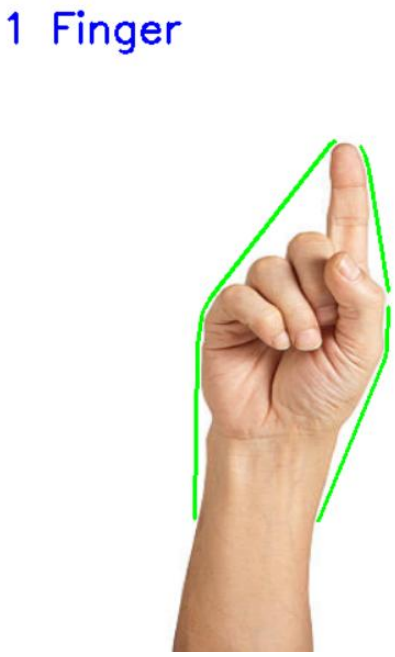
```
import cv2
import numpy as np
import math
image = cv2.imread('hand.jpeg')
if image is None:
   print("Image not found")
   exit()
image = cv2.resize(image, (500, 500))
roi = image[100:400, 100:400]
```

```
hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

lower_skin = np.array([0, 20, 70], dtype=np.uint8)
upper_skin = np.array([20, 255, 255], dtype=np.uint8)

mask = cv2.inRange(hsv, lower_skin, upper_skin)
mask = cv2.GaussianBlur(mask, (5, 5), 0)

contours=cv2.findContours(mask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIM
PLE)

if contours:
    cnt = max(contours, key=lambda x: cv2.contourArea(x))
    epsilon = 0.0005 * cv2.arcLength(cnt, True)
    approx = cv2.approxPolyDP(cnt, epsilon, True)
    hull = cv2.convexHull(approx, returnPoints=False)

    if hull is not None and len(hull) > 3:
        defects = cv2.convexityDefects(approx, hull)
        count_defects = 0

        if defects is not None:
            for i in range(defects.shape[0]):
                s, e, f, d = defects[i, 0]
                start = tuple(approx[s][0])



            end = tuple(approx[e][0])
                far = tuple(approx[f][0])

                a = math.dist(end, start)
                b = math.dist(far, start)
                c = math.dist(end, far)
                angle = math.acos((b**2 + c**2 - a**2) / (2*b*c)) * 57

                if angle <= 90 and d > 10000:
                    count_defects += 1
                    cv2.circle(roi, far, 5, (0, 0, 255), -1)
            cv2.line(roi, start, end, (0, 255, 0), 2)

        if count_defects == 0:
            text = "1 Finger"
        elif count_defects == 1:
            text = "2 Fingers"
        elif count_defects == 2:
            text = "3 Fingers"
        elif count_defects == 3:
            text = "4 Fingers"
        elif count_defects == 4:
            text = "5 Fingers"
        else:
            text = "Unknown"

        cv2.putText(image, text, (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,
0, 0), 2)

cv2.imshow("Result", image)
cv2.imshow("Mask", mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input:



Output:



1 Finger

Applications:

Touchless Interfaces: Control smart TVs, media players, or computers using hand gestures.

Gaming: Gesture-based input for immersive gaming experiences.

Sign Language Translation: Assists in real-time interpretation of sign language for the hearing impaired.

Robotics: Human-robot interaction using intuitive hand gestures.

Healthcare: Hands-free operation of medical systems in sterile environments.

3. Smart Parking Detection System

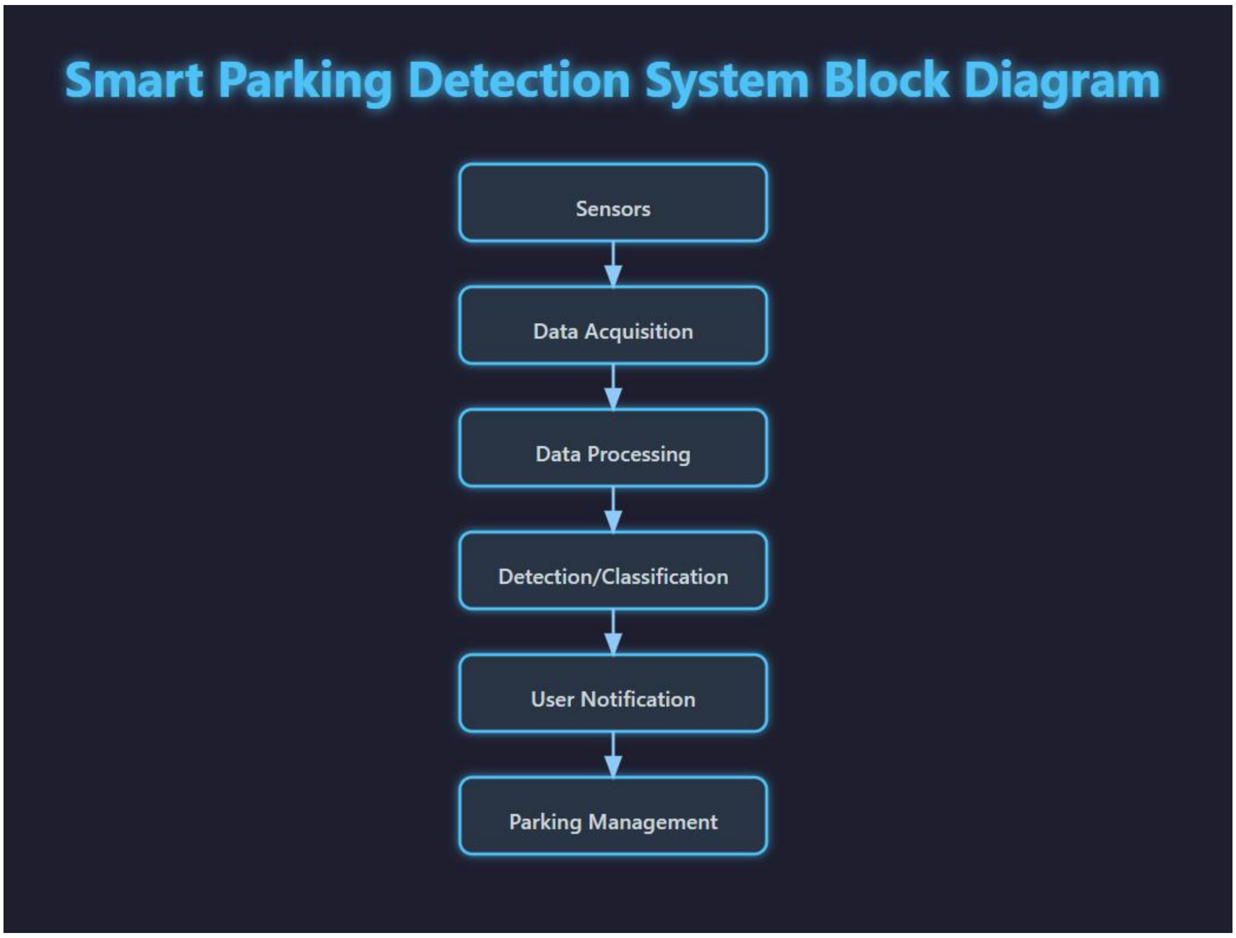
Description:

The Smart Parking Detection System uses computer vision to monitor parking lots and determine which spaces are vacant or occupied. By analyzing video feeds from CCTV or drone cameras, it can automatically identify the status of each slot. This helps drivers find available spaces and supports smart city automation efforts.

Design Strategy:

- Capture a top-down video of a parking area (static or moving).
- Mark Regions of Interest (ROIs) manually for each parking slot.
- Convert frames to grayscale and apply thresholding.
- Use contour detection or pixel count comparison to determine if a car is present.
- Display visual indicators (e.g., green for empty, red for occupied).
- Continuously update the status for real-time monitoring.

Block diagram:

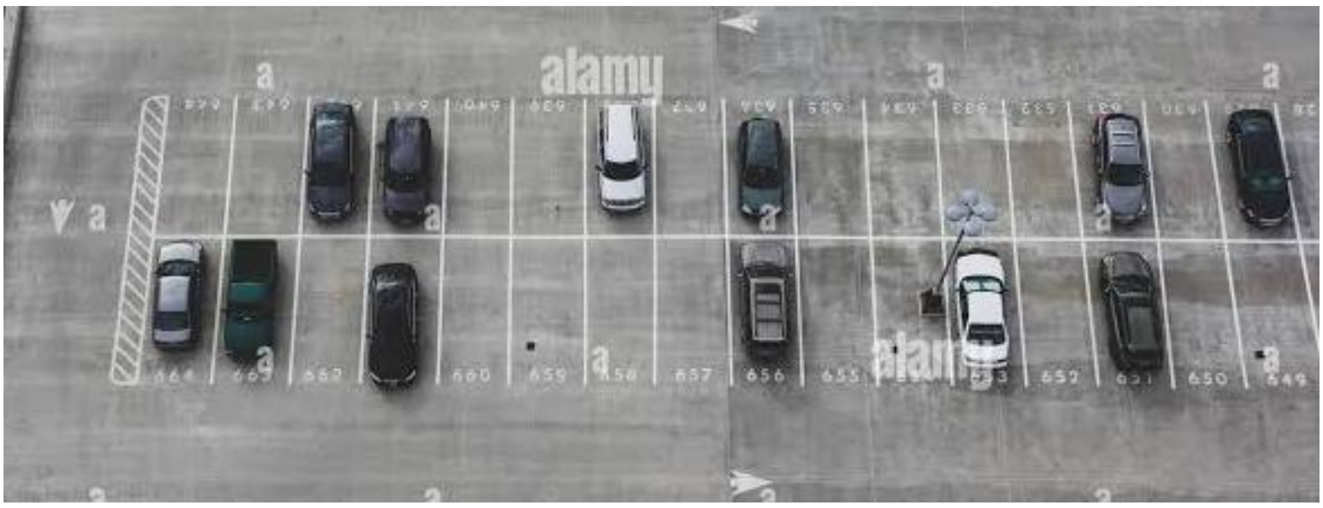


Program:

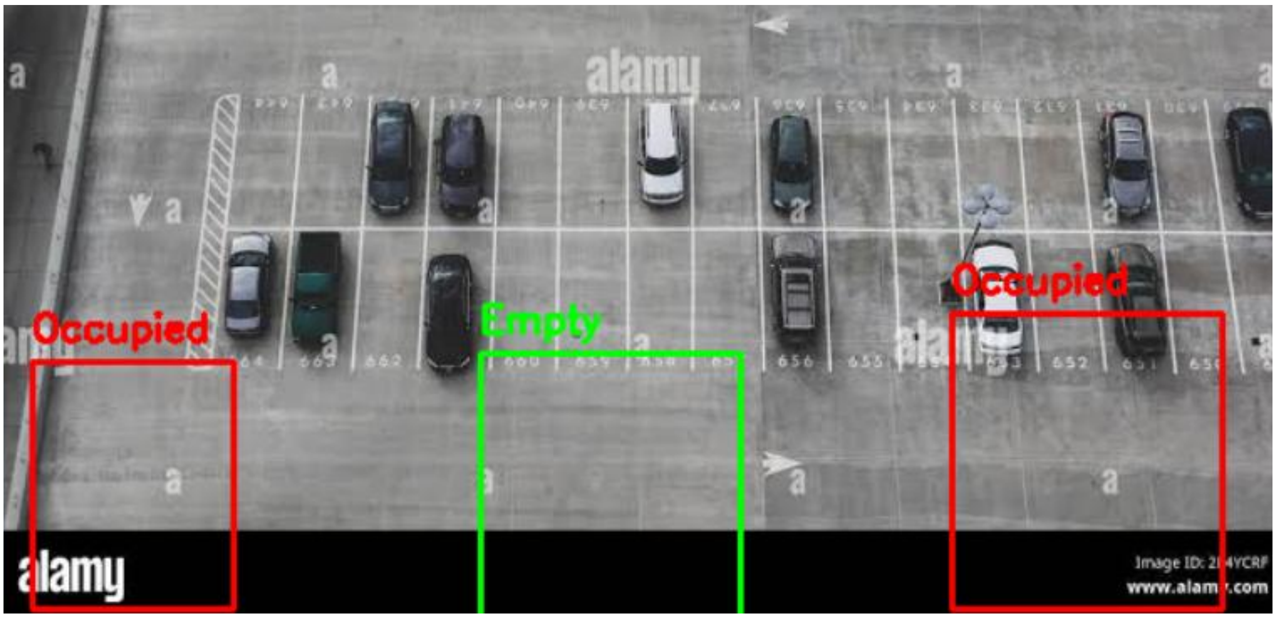
```
import cv2
import numpy as np
img = cv2.imread('parking.jpeg')
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_blur = cv2.GaussianBlur(img_gray, (5,5), 1)
edges = cv2.Canny(img_blur, 50, 150)
parking_spots = [
   (31,350,127,468),
   (245,346,369,474),
   (470,327,599,468),
   (703,323,816,455),
   (920,317,1014,463),

]
```

```
def check_occupancy(spot):
    x1, y1, x2, y2 = spot
    roi = edges[y1:y2, x1:x2]
    non_zero_count = cv2.countNonZero(roi)
    if non_zero_count > 500:
        return True
    return False
for spot in parking_spots:
    occupied = check_occupancy(spot)
    color = (0,0,255) if occupied else (0,255,0)
    cv2.rectangle(img, (spot[0], spot[1]), (spot[2], spot[3]), color, 2)
    label = "Occupied" if occupied else "Empty"
    text_pos = (spot[0], spot[1] - 10 if spot[1] - 10 > 10 else spot[1] + 20)
    cv2.putText(img, label, text_pos, cv2.FONT_HERSHEY_SIMPLEX, 0.6, color,
2)

cv2.imshow('Parking Detection', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input:



Output:



Applications:

Smart Cities: Helps drivers locate available parking, reducing traffic congestion.

Shopping Malls & Airports: Automated parking guidance for high-traffic areas.

Corporate Campuses: Monitors employee and visitor parking availability.

Public Parking Lots: Enables real-time parking analytics and automated payment systems.

Event Management: Dynamic parking space allocation during large-scale public events.

Result:

The Autonomous Drone Vision System successfully simulated drone behavior by detecting and tracking objects from video input. The Gesture Recognition System accurately recognized various hand gestures, such as open palm and finger count, for gesture-based control. The Smart Parking Detection System effectively monitored parking spaces and displayed real-time status using visual indicators for occupied and vacant slots.