Python script for object detection

import cv2

import numpy as np

from picamera import PiCamera

from picamera.array import PiRGBArray

from imutils.video import FPS

# Load YOLOv8 model

net = cv2.dnn.readNet('yolov8.weights', 'yolov8.cfg')

classes = []

with open('coco.names', 'r') as f:

classes = f.read().strip().split('\n')

layer\_names = net.getUnconnectedOutLayersNames()

# Initialize camera

camera = PiCamera()

camera.resolution = (640, 480)

camera.framerate = 32

rawCapture = PiRGBArray(camera, size=(640, 480))

# Allow the camera to warmup

time.sleep(0.1)

fps = FPS().start()

for frame in camera.capture\_continuous(rawCapture, format="bgr", use\_video\_port=True):

image = frame.array

# YOLOv8 object detection

height, width, \_ = image.shape

blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(layer\_names)

# Process detections

for out in outs:

for detection in out:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

center\_x = int(detection[0] \* width)

center\_y = int(detection[1] \* height)

w = int(detection[2] \* width)

h = int(detection[3] \* height)

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

# Draw bounding box and label

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.putText(image, classes[class\_id], (x, y - 5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

# Display the resulting frame

cv2.imshow('Object Detection', image)

key = cv2.waitKey(1) & 0xFF

# Break the loop if 'q' key is pressed

if key == ord('q'):

break

rawCapture.truncate(0)

fps.update()

fps.stop()

print(f"FPS: {fps.fps()}")

# Release the camera and close all windows

cv2.destroyAllWindows()

**GUI**

import cv2

import numpy as np

import tkinter as tk

from tkinter import ttk

from PIL import Image, ImageTk

from picamera import PiCamera

from picamera.array import PiRGBArray

from imutils.video import FPS

import RPi.GPIO as GPIO

import time

# Global variables for servo control

pan\_pin = 18 # GPIO pin for pan servo

tilt\_pin = 23 # GPIO pin for tilt servo

# Set GPIO mode and setup

GPIO.setmode(GPIO.BCM)

GPIO.setup(pan\_pin, GPIO.OUT)

GPIO.setup(tilt\_pin, GPIO.OUT)

# Function to set servo angle

def set\_angle(pin, angle):

duty = (angle + 90) / 18 + 2

GPIO.output(pin, True)

pwm = GPIO.PWM(pin, 50)

pwm.start(0)

pwm.ChangeDutyCycle(duty)

time.sleep(1)

pwm.stop()

GPIO.output(pin, False)

# Function to perform auto-scan

def auto\_scan():

# Perform auto-scan from left to right

for angle in range(-90, 91, 10):

set\_angle(pan\_pin, angle)

time.sleep(0.5)

# Perform auto-scan from right to left

for angle in range(90, -91, -10):

set\_angle(pan\_pin, angle)

time.sleep(0.5)

# Perform auto-scan from down to up

for angle in range(-90, 91, 10):

set\_angle(tilt\_pin, angle)

time.sleep(0.5)

# Perform auto-scan from up to down

for angle in range(90, -91, -10):

set\_angle(tilt\_pin, angle)

time.sleep(0.5)

class CameraApp:

def \_\_init\_\_(self, window, window\_title):

self.window = window

self.window.title(window\_title)

# Variables

self.camera = PiCamera()

self.camera.resolution = (640, 480)

self.raw\_capture = PiRGBArray(self.camera, size=(640, 480))

self.current\_frame = None

self.is\_camera\_seen = True

# Create left frame for camera feed

self.left\_frame = tk.Frame(self.window)

self.left\_frame.grid(row=0, column=0, padx=10, pady=10)

# Create right frame for control buttons

self.right\_frame = tk.Frame(self.window)

self.right\_frame.grid(row=0, column=1, padx=10, pady=10)

# Label for camera status

self.camera\_status\_label = tk.Label(self.left\_frame, text="Camera not Seen", font=("Helvetica", 16))

self.camera\_status\_label.pack(pady=10)

# Canvas for displaying camera feed

self.canvas = tk.Canvas(self.left\_frame, width=640, height=480)

self.canvas.pack()

# Control buttons

self.up\_button = ttk.Button(self.right\_frame, text="Up", command=lambda: self.move\_servo("up"))

self.up\_button.grid(row=0, column=0, pady=10)

self.left\_button = ttk.Button(self.right\_frame, text="Left", command=lambda: self.move\_servo("left"))

self.left\_button.grid(row=1, column=0, padx=10)

self.right\_button = ttk.Button(self.right\_frame, text="Right", command=lambda: self.move\_servo("right"))

self.right\_button.grid(row=1, column=2, padx=10)

self.down\_button = ttk.Button(self.right\_frame, text="Down", command=lambda: self.move\_servo("down"))

self.down\_button.grid(row=2, column=0, pady=10)

# Start camera feed and update GUI

self.update()

def move\_servo(self, direction):

if direction == "up":

set\_angle(tilt\_pin, -90)

elif direction == "left":

set\_angle(pan\_pin, -90)

elif direction == "right":

set\_angle(pan\_pin, 90)

elif direction == "down":

set\_angle(tilt\_pin, 90)

def update(self):

try:

for frame in self.camera.capture\_continuous(self.raw\_capture, format="bgr", use\_video\_port=True):

self.current\_frame = frame.array

# Display camera feed on Tkinter window

self.photo = ImageTk.PhotoImage(image=Image.fromarray(self.current\_frame))

self.canvas.create\_image(0, 0, image=self.photo, anchor=tk.NW)

# Perform object detection

# (Add your object detection code here)

# Clear the stream for the next frame

self.raw\_capture.truncate(0)

# Update camera status label

if not self.is\_camera\_seen:

self.camera\_status\_label.config(text="Camera not Seen", fg="red")

self.is\_camera\_seen = True

self.window.update()

except Exception as e:

# Handle camera not found or other exceptions

self.camera\_status\_label.config(text="Camera not Seen", fg="red")

self.is\_camera\_seen = False

print(f"Error: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = CameraApp(root, "Camera App with Object Detection")

root.mainloop()

# Cleanup GPIO

GPIO.cleanup()

Python Script for Motor Control

import RPi.GPIO as GPIO

import time

# GPIO pins for servo motors

pan\_pin = 18 # GPIO pin for pan servo

tilt\_pin = 23 # GPIO pin for tilt servo

# Set GPIO mode and setup

GPIO.setmode(GPIO.BCM)

GPIO.setup(pan\_pin, GPIO.OUT)

GPIO.setup(tilt\_pin, GPIO.OUT)

# Function to set servo angle

def set\_angle(pin, angle):

duty = (angle + 90) / 18 + 2

GPIO.output(pin, True)

pwm = GPIO.PWM(pin, 50)

pwm.start(0)

pwm.ChangeDutyCycle(duty)

time.sleep(1)

pwm.stop()

GPIO.output(pin, False)

# Function to perform auto-scan

def auto\_scan():

# Perform auto-scan from left to right

for angle in range(-90, 91, 10):

set\_angle(pan\_pin, angle)

time.sleep(0.5)

# Perform auto-scan from right to left

for angle in range(90, -91, -10):

set\_angle(pan\_pin, angle)

time.sleep(0.5)

# Perform auto-scan from down to up

for angle in range(-90, 91, 10):

set\_angle(tilt\_pin, angle)

time.sleep(0.5)

# Perform auto-scan from up to down

for angle in range(90, -91, -10):

set\_angle(tilt\_pin, angle)

time.sleep(0.5)

# Main loop

try:

while True:

# Perform auto-scan every 5 minutes

auto\_scan()

time.sleep(5 \* 60) # 5 minutes

except KeyboardInterrupt:

pass

finally:

# Cleanup

GPIO.cleanup()