# Import required modules

**import** cv2

**import** numpy as np

**import** os

**import** glob

# Define the dimensions of checkerboard

CHECKERBOARD **=** (6, 9)

# stop the iteration when specified

# accuracy, epsilon, is reached or

# specified number of iterations are completed.

criteria **=** (cv2.TERM\_CRITERIA\_EPS **+**

            cv2.TERM\_CRITERIA\_MAX\_ITER, 30, 0.001)

# Vector for 3D points

threedpoints **=** []

# Vector for 2D points

twodpoints **=** []

#  3D points real world coordinates

objectp3d **=** np.zeros((1, CHECKERBOARD[0]

**\*** CHECKERBOARD[1],

                      3), np.float32)

objectp3d[0, :, :2] **=** np.mgrid[0:CHECKERBOARD[0],

                               0:CHECKERBOARD[1]].T.reshape(**-**1, 2)

prev\_img\_shape **=** None

# Extracting path of individual image stored

# in a given directory. Since no path is

# specified, it will take current directory

# jpg files alone

images **=** glob.glob('\*.jpg')

**for** filename **in** images:

    image **=** cv2.imread(filename)

    grayColor **=** cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    # Find the chess board corners

    # If desired number of corners are

    # found in the image then ret = true

    ret, corners **=** cv2.findChessboardCorners(

                    grayColor, CHECKERBOARD,

                    cv2.CALIB\_CB\_ADAPTIVE\_THRESH

**+** cv2.CALIB\_CB\_FAST\_CHECK **+**

                    cv2.CALIB\_CB\_NORMALIZE\_IMAGE)

    # If desired number of corners can be detected then,

    # refine the pixel coordinates and display

    # them on the images of checker board

**if** ret **==** True:

        threedpoints.append(objectp3d)

        # Refining pixel coordinates

        # for given 2d points.

        corners2 **=** cv2.cornerSubPix(

            grayColor, corners, (11, 11), (**-**1, **-**1), criteria)

        twodpoints.append(corners2)

        # Draw and display the corners

        image **=** cv2.drawChessboardCorners(image,

                                          CHECKERBOARD,

                                          corners2, ret)

    cv2.imshow('img', image)

    cv2.waitKey(0)

cv2.destroyAllWindows()

h, w **=** image.shape[:2]

# Perform camera calibration by

# passing the value of above found out 3D points (threedpoints)

# and its corresponding pixel coordinates of the

# detected corners (twodpoints)

ret, matrix, distortion, r\_vecs, t\_vecs **=** cv2.calibrateCamera(

    threedpoints, twodpoints, grayColor.shape[::**-**1], None, None)

# Displaying required output

print(" Camera matrix:")

**print**(matrix)

print("\n Distortion coefficient:")

print(distortion)

**print**("\n Rotation Vectors:")

print(r\_vecs)

**print**("\n Translation Vectors:")

print(t\_vecs)

**QUESTION 1:**

# Import required modules

import cv2

import numpy as np

import os

import glob

CHECKERBOARD = (6, 8)

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 30, 0.001)

threedpoints = []

twodpoints = []

objectp3d = np.zeros((1, CHECKERBOARD[0] \* CHECKERBOARD[1], 3), np.float32)

objectp3d[0, :, :2] = np.mgrid[0:CHECKERBOARD[0], 0:CHECKERBOARD[1]].T.reshape(-1, 2)

prev\_img\_shape = None

images = glob.glob('\*.jpg')

for filename in images:

image = cv2.imread(filename)

grayColor = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

ret, corners = cv2.findChessboardCorners(

grayColor, CHECKERBOARD,

cv2.CALIB\_CB\_ADAPTIVE\_THRESH

+ cv2.CALIB\_CB\_FAST\_CHECK +

cv2.CALIB\_CB\_NORMALIZE\_IMAGE)

if ret == True:

threedpoints.append(objectp3d)

corners2 = cv2.cornerSubPix(

grayColor, corners, (11, 11), (-1, -1), criteria)

twodpoints.append(corners2)

image = cv2.drawChessboardCorners(image,

CHECKERBOARD,

corners2, ret)

cv2.imshow('img', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

h, w = image.shape[:2]

ret, matrix, distortion, r\_vecs, t\_vecs = cv2.calibrateCamera( threedpoints, twodpoints, grayColor.shape[::-1], None, None) print(" Camera matrix:")

print(matrix)

print("\n Distortion coefficient:")

print(distortion)

print("\n Rotation Vectors:")

print(r\_vecs)

print("\n Translation Vectors:")

print(t\_vecs)

**INPUT:**







**OUTPUT:**

Camera matrix:

[[ 36.26378216 0. 125.68539168]

[ 0. 36.76607372 142.49821147]

[ 0. 0. 1. ]]

Distortion coefficient:

[[-1.25491812e-03 9.89269357e-05 -2.89077718e-03 4.52760939e-04

-3.29964245e-06]]

Rotation Vectors:

[array([[-0.05767492],

[ 0.03549497],

[ 1.50906953]]), array([[-0.09301982],

[-0.01034321],

[ 3.07733805]]), array([[-0.02175332],

[ 0.05611105],

[-0.07308161]])]

Translation Vectors:

[array([[ 4.63047351],

[-3.74281386],

[ 1.64238108]]), array([[2.31648737],

[3.98801521],

[1.64584622]]), array([[-3.17548808],

[-3.46022466],

[ 1.68200157]])]

**QUESTION 2:**

TASK 1:

RESULTSANDDISCUSSION

Input:Trainingdataset

Output:Tocreatedatamodel.

Step1:Installingandimportdependencies

Step2:Todeterminethe2Dcropofaninputimage,thefirststageusedthe

popular TensorFlowobjectidentificationmodel.

Step3:Thesecondsteprequiredusingthesecroppedphotostodeterminetheir

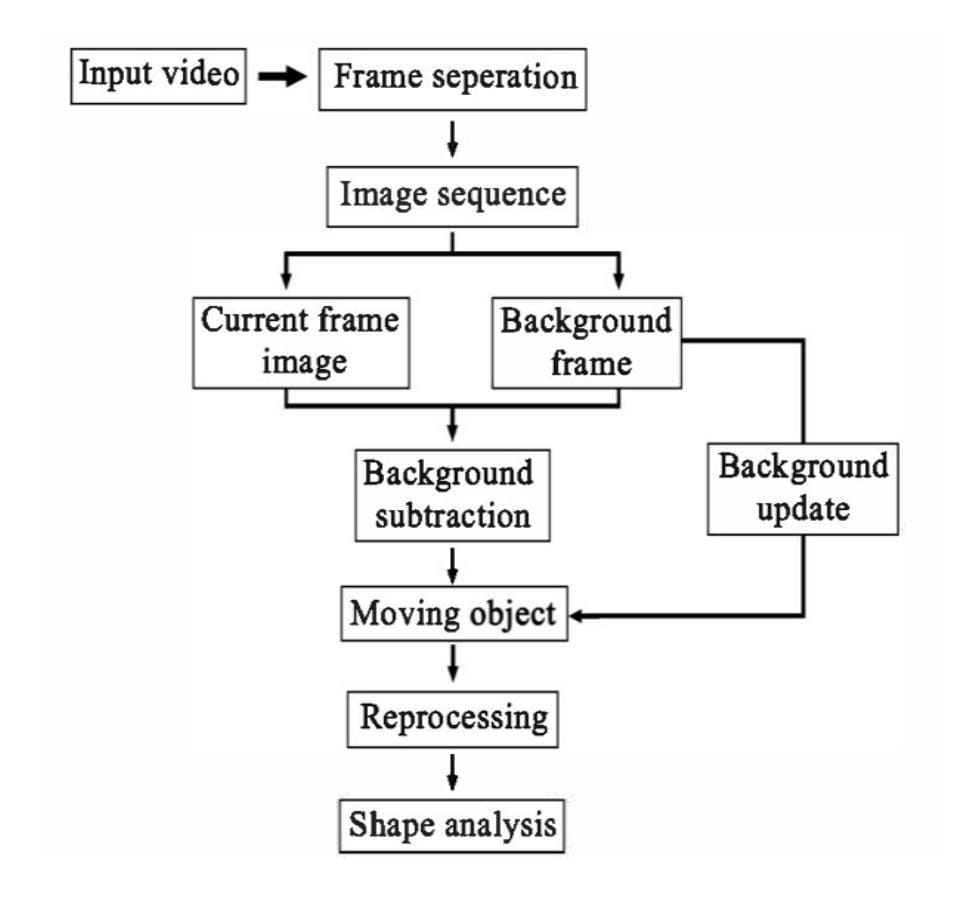
3Dboundingboxesafterthiscroppinghadbeencompleted.

Step4:Uploadingimagesorrealtimesvideos

Step5:PerformingDetectionandidentification

Step6:Drawingtheboxlandmarksandidentifyingimagenameandaccuracy

**SYSTEM ARCHITECTURE**

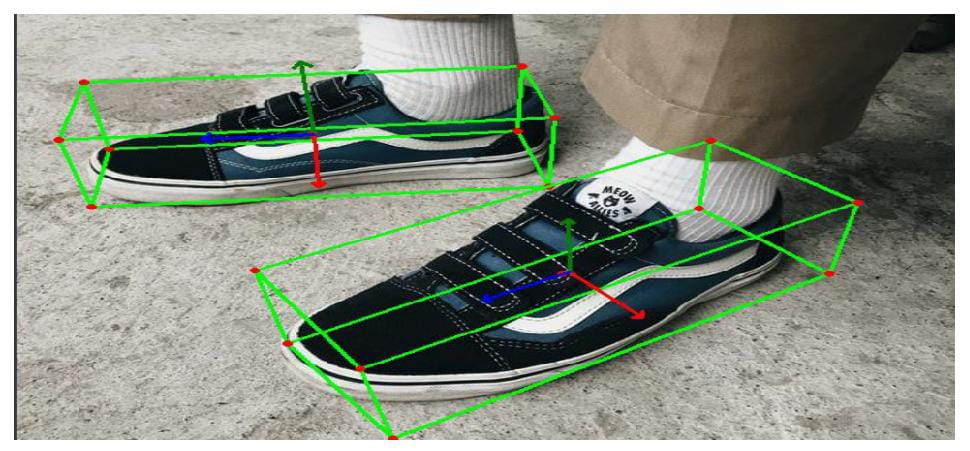


OUTPUT:

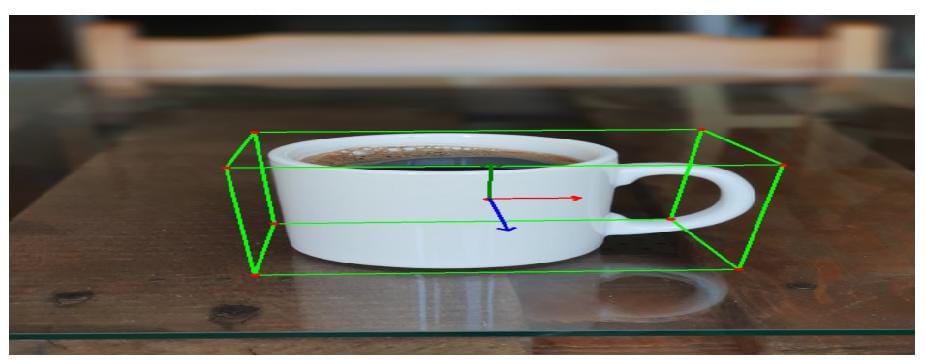
1. 3D Bounding object detection chairs



1. 3D Bounding object detection shoes



1. 3D Bounding object detection CUP



ACCURACY= 89%

**TASK 2:**

**Small object detection**: Small object detection is a computer vision problem where you aim to accurately identify objects that are small in a video feed or image. The object itself does not necessarily need to be small. For instance, small object detection is crucial in aerial computer vision, where you need to be able to accurately identify objects even though each individual object will be small relative to the photo size.

**Source code:**

[net]

batch=64

subdivisions=36

width={YOUR RESOLUTION WIDTH HERE}

height={YOUR RESOLUTION HEIGHT HERE}

channels=3

momentum=0.949

decay=0.0005

angle=0

saturation = 1.5

exposure = 1.5

hue = .1

learning\_rate=0.001

burn\_in=1000

max\_batches=6000

policy=steps

steps=4800.0,5400.0

scales=.1,.1

**Tile images at inference:**

The model was trained with tiling to better recognize cars given their small size and the large size of the source image, but without tiling at inference, it ends up detecting buildings and other large shapes instead of cars because they are closer to the size of the object it was attempting to detect during training.



Below we’ve applied tiling to an image before running inference. This allows us to zoom in to sections of the image and make our cars bigger and easier to detect for the model.



**Bonus Question:**

C code for a singly linked list that stores data of type int.

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

void linkedListTraversal(struct Node \*ptr)

{

while (ptr != NULL)

{

printf("Element: %d\n", ptr->data);

ptr = ptr->next;

}

}

// Case 1: Deleting the first element from the linked list

struct Node \* deleteFirst(struct Node \* head){

struct Node \* ptr = head;

head = head->next;

free(ptr);

return head;

}

// Case 2: Deleting the element at a given index from the linked list

struct Node \* deleteAtIndex(struct Node \* head, int index){

struct Node \*p = head;

struct Node \*q = head->next;

for (int i = 0; i < index-1; i++)

{

p = p->next;

q = q->next;

}

p->next = q->next;

free(q);

return head;

}

// Case 3: Deleting the last element

struct Node \* deleteAtLast(struct Node \* head){

struct Node \*p = head;

struct Node \*q = head->next;

while(q->next !=NULL)

{

p = p->next;

q = q->next;

}

p->next = NULL;

free(q);

return head;

}

// Case 4: Deleting the element with a given value from the linked list

struct Node \* deleteAtIndex(struct Node \* head, int value){

struct Node \*p = head;

struct Node \*q = head->next;

while(q->data!=value && q->next!= NULL)

{

p = p->next;

q = q->next;

}

if(q->data == value){

p->next = q->next;

free(q);

}

return head;

}

int main()

{

struct Node \*head;

struct Node \*second;

struct Node \*third;

struct Node \*fourth;

// Allocate memory for nodes in the linked list in Heap

head = (struct Node \*)malloc(sizeof(struct Node));

second = (struct Node \*)malloc(sizeof(struct Node));

third = (struct Node \*)malloc(sizeof(struct Node));

fourth = (struct Node \*)malloc(sizeof(struct Node));

// Link first and second nodes

head->data = 4;

head->next = second;

// Link second and third nodes

second->data = 3;

second->next = third;

// Link third and fourth nodes

third->data = 8;

third->next = fourth;

// Terminate the list at the third node

fourth->data = 1;

fourth->next = NULL;

printf("Linked list before deletion\n");

linkedListTraversal(head);

// head = deleteFirst(head); // For deleting first element of the linked list

// head = deleteAtIndex(head, 2);

head = deleteAtLast(head);

printf("Linked list after deletion\n");

linkedListTraversal(head);

return 0;

}