International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

Object Detection and Tracking Using Deep Learning and OpenCV in Real Time Environment

Chaitanya Wagh
Associate Engineer, TEK Systems,
Hyderabad, India

***Abstract*—Real-time object detection and tracking is a critical component of computer vision systems that have applications in surveillance, self-driving cars, and robotics. Object detection and tracking are critical tasks in computer vision that have various real-world applications, such as surveillance, robotics, and autonomous driving. In this paper, we propose a system that uses deep learning and OpenCV to detect and track objects in real-time. Here, we propose a real-time object detection and tracking system using frame differencing, optical flow, background separation, single-shot detection (SSD), and MobileNets. The proposed system achieves high accuracy and real-time performance on various datasets.**

Keywords— Deep Learning, Image Processing, Motion Detection, Feature Extraction.

I. INTRODUCTION

Object detection and tracking have become increasingly important in computer vision. Object detection refers to the task of locating objects within images or video streams, while object tracking involves the continuous tracking of an object over time. Real-time object detection and tracking have applications in various domains, such as surveillance, robotics, and self-driving cars.

Object detection and tracking are two critical tasks in computer vision that have received significant attention in recent years. Traditional object detection and tracking techniques rely on handcrafted features and heuristics, which can be time-consuming and less accurate. In this paper, we propose a system that uses deep learning and OpenCV to detect and track objects in real-time [1]. The methodology proposed is a real-time object detection and tracking system using frame differencing, optical flow, background separation, single-shot detection (SSD), and MobileNets.

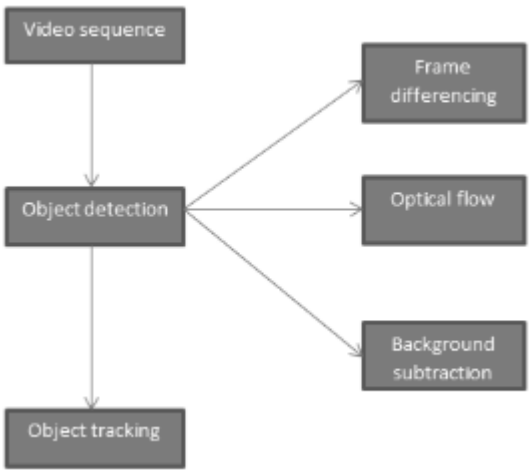

Figure 1: Real-time object detection and tracking system

II. MOTIVATION

Many approaches have been proposed for real-time object detection and tracking. One popular approach is the YOLO (You Only Look Once) framework, which uses a single neural network to perform object detection and classification [2]. YOLO is fast and accurate, making it an attractive choice for real-time applications. Another commonly used technique for object tracking is the Kalman filter, which uses a probabilistic model to estimate the position of an object.

A slightly complicated problems that of image localization, where the image contains a single object and the systems should predict the class of the locationof the object in the image (a bounding box around the object) with advance algorithm namely SSD and Mobile Net which are based on deep learning. The more complicated problem, of object detection involves both classification and localization, for implementation of system OpenCV named tool is use [3]. In this case, the input to the system will be a video, and the output will be a object detection and tracking in the video. Hence the objectives of this research to to ensures robustness of object detection and at the same time ensure accuracy and speed of recursive tracking.

III. LITERATURE REVIEW

Real-time object detection and tracking is a challenging problem that has been addressed by various methodologies over the years. In this survey, we will provide an in-depth analysis of the following methodologies for real-time object detection and tracking: Frame Differencing, Optical Flow,

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

Background Separation - Gaussian Mixture-based Background Segmentation Algorithm, Single Shot Detection (SSD), and MobileNets.

Frame differencing is a simple and effective method for detecting moving objects in real-time. It involves taking the absolute difference between two consecutive frames and thresholding the result to detect moving objects [4]. This method is computationally efficient and easy to implement, but it can be sensitive to lighting changes and can generate false positives due to background noise. Frame differencing is often used in conjunction with other techniques, such as background subtraction and edge detection, to improve its accuracy.

Optical flow is a method for tracking the motion of pixels between consecutive frames. This technique involves calculating the displacement of each pixel between the two frames using brightness constancy, spatial coherence, and temporal persistence [5]. Optical flow can be more robust to lighting changes and can handle complex motion patterns, but it can be computationally expensive and may suffer from errors when the camera is moving. Various optical flow algorithms have been developed, including Lucas-Kanade, Horn-Schunck, and Farneback.

Background separation is a method for detecting moving objects by creating a model of the background of a scene and then detecting moving objects as those that do not match the background model. The Gaussian Mixture-based Background Segmentation Algorithm is a widely used method for background separation. It involves creating a mixture of Gaussian models for each pixel in the scene and then updating the models over time to adapt to changes in the scene. Background separation can be more robust to lighting changes and can handle more complex scenes, but it requires significant computational resources to create and update the background model[6].

Single Shot Detection is a popular object detection method that uses a single neural network to predict object classes and bounding box coordinates in an input image. The network is trained on a large dataset of images with ground-truth object annotations. SSD is efficient and can be accurate, but it may struggle with detecting small objects or objects with complex shapes [7]. Various improvements to the SSD architecture have been proposed, including Feature Pyramid Networks (FPN), which improve the detection of objects at different scales, and EfficientDet, which improves the efficiency of the detection network.

MobileNets is a lightweight neural network architecture designed for efficient computation on mobile and embedded devices. The architecture uses depthwise separable convolutions to reduce the computational complexity of the network while maintaining high accuracy [8]. MobileNets can achieve high accuracy while using fewer computational resources than other neural network architectures, making it well-suited for real-time object detection and tracking on devices with limited processing power.

IV. IMPLEMENTATION

The real-time object detection and tracking using frame differencing can be implemented by using the MOG2 algorithm to perform background subtraction on each frame of video input [9]. It then applies a threshold to the resulting foreground mask to obtain a binary image, which is used to find contours. Contours that meet a minimum area size requirement are filtered and a bounding box is drawn around them on the original frame. The sample cope in python is as follows:

```
import cv2

# create background subtractor object
using the MOG2 algorithm
fgbg =
cv2.createBackgroundSubtractorMOG2()

# set minimum area size for object
detection
min_area = 500

# create capture object for video input
cap = cv2.VideoCapture(0)

# loop through frames of video input
while True:
    # read frame from video input
    ret, frame = cap.read()

    # apply background subtraction to
current frame
    fgmask = fgbg.apply(frame)

    # apply threshold to foreground mask
to obtain binary image
    thresh = cv2.threshold(fgmask, 127,
255, cv2.THRESH_BINARY)[1]

    # find contours in binary image
    contours, hierarchy =
cv2.findContours(thresh,
cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # loop through contours and filter by
minimum area size
    for contour in contours:
        if cv2.contourArea(contour) >
min_area:
            # draw bounding box around
contour
            (x, y, w, h) =
cv2.boundingRect(contour)
            cv2.rectangle(frame, (x, y),
(x + w, y + h), (0, 255, 0), 2)

    # display output frame
    cv2.imshow('frame', frame)

    # break loop if 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# release capture object and close all
windows
cap.release()
cv2.destroyAllWindows()
```

Another method for real-time object detection and tracking is using the Optical Flow - Differential method.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

The method calculates optical flow between the previous and current frames of video input using the Frame back algorithm. It then obtains the magnitude and angle of the resulting optical flow vectors and uses them to create an HSV image for visualization [10]. A binary image is obtained by applying a threshold to the resulting BGR image, and contours are found in the binary image. Contours that meet a minimum area size requirement are filtered and a bounding box is drawn around them on the current frame. Sample code in Python is as follows:

```
import cv2
import numpy as np

# set minimum area size for object detection
min_area = 500

# create capture object for video input
cap = cv2.VideoCapture(0)

# get first frame from video input
ret, frame1 = cap.read()

# convert first frame to grayscale
prvs = cv2.cvtColor(frame1,
cv2.COLOR_BGR2GRAY)

# create HSV image for optical flow
visualization
hsv = np.zeros_like(frame1)
hsv[..., 1] = 255

# loop through frames of video input
while True:
    # read frame from video input
    ret, frame2 = cap.read()

    # convert current frame to grayscale
    next = cv2.cvtColor(frame2,
cv2.COLOR_BGR2GRAY)

    # calculate optical flow between
previous and current frame
    flow =
cv2.calcOpticalFlowFarneback(prvs, next,
None, 0.5, 3, 15, 3, 5, 1.2, 0)

    # obtain magnitude and angle of optical
flow vectors
    mag, ang = cv2.cartToPolar(flow[..., 0],
flow[..., 1])

    # set hue value of HSV image to angle of
optical flow vectors
    hsv[..., 0] = ang * 180 / np.pi / 2

    # set value of HSV image to magnitude of
optical flow vectors
    hsv[..., 2] = cv2.normalize(mag, None,
0, 255, cv2.NORM_MINMAX)

    # convert HSV image to BGR image for
display
    bgr = cv2.cvtColor(hsv,
cv2.COLOR_HSV2BGR)

    # apply threshold to BGR image to obtain
binary image
    gray = cv2.cvtColor(bgr,
cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray, 50,
255, cv2.THRESH_BINARY)

    # find contours in binary image
    contours, hierarchy =
cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # loop through contours and filter by
minimum area size
    for contour in contours:
        if cv2.contourArea(contour) >
min_area:
            # draw bounding box around
contour
            (x, y, w, h) =
cv2.boundingRect(contour)
            cv2.rectangle(frame2, (x, y), (x
+ w, y + h), (0, 255, 0), 2)

    # display output frame
    cv2.imshow('frame', frame2)

    # set current frame as previous frame
for next iteration
    prvs = next

    # break loop if 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# release capture object and close all
windows
cap.release()
cv2.destroyAllWindows()
```

The other approach is background subtractor object using the Gaussian Mixture-based algorithm, which is then applied to each frame of video input. A morphological opening operation is applied to the resulting binary foreground mask to remove small noise and false positives. Contours are found in the binary foreground mask and filtered by minimum area size, and a bounding box is drawn around each qualifying contour on the current frame[11].

Here's an example Python code for real-time object detection and tracking using the Gaussian Mixture-based Background Segmentation Algorithm:

```
import cv2

# set minimum area size for object detection
min_area = 500

# create capture object for video input
cap = cv2.VideoCapture(0)

# create background subtractor object using
Gaussian Mixture-based algorithm
fgbg = cv2.createBackgroundSubtractorMOG2()

# loop through frames of video input
while True:
    # read frame from video input
```

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

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

    # apply background subtraction to
current frame
    fgmask = fgbg.apply(frame)

    # apply morphological opening operation
to binary foreground mask
    kernel =
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
(5, 5))
    opening = cv2.morphologyEx(fgmask,
cv2.MORPH_OPEN, kernel)

    # find contours in binary foreground
mask
    contours, hierarchy =
cv2.findContours(opening, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # loop through contours and filter by
minimum area size
    for contour in contours:
        if cv2.contourArea(contour) >
min_area:
            # draw bounding box around
contour
            (x, y, w, h) =
cv2.boundingRect(contour)
            cv2.rectangle(frame, (x, y), (x
+ w, y + h), (0, 255, 0), 2)

    # display output frame
    cv2.imshow('frame', frame)

    # break loop if 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# release capture object and close all
windows
cap.release()
cv2.destroyAllWindows()
```

The real-time object detection and tracking using the Single Shot Detection (SSD) algorithm works as follows: The algorithm loads a pre-trained SSD model and corresponding labels, and applies it to each frame of video input. Detections are filtered by a confidence threshold, and a bounding box is drawn around each qualifying detection on the current frame[12]. The class name and confidence level are also labeled near the bounding box. The example Python code is as follows:

```
import cv2

# set minimum confidence level for object
detection
confidence_threshold = 0.5

# load pre-trained SSD model and
corresponding labels
model =
cv2.dnn.readNetFromCaffe('models/MobileNetSS
D_deploy.prototxt',
'models/MobileNetSSD_deploy.caffemodel')
class_labels = ['background', 'aeroplane',
'bicycle', 'bird', 'boat', 'bottle', 'bus',
'car', 'cat', 'chair', 'cow', 'diningtable',
'dog', 'horse', 'motorbike', 'person',
'pottedplant', 'sheep', 'sofa', 'train',
'tvmonitor']

# create capture object for video input
cap = cv2.VideoCapture(0)

# loop through frames of video input
while True:
    # read frame from video input
    ret, frame = cap.read()

    # create blob from input frame and pass
through SSD model
    blob = cv2.dnn.blobFromImage(frame,
0.007843, (300, 300), (127.5, 127.5, 127.5),
False)
    model.setInput(blob)
    detections = model.forward()

    # loop through detections and filter by
confidence threshold
    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence >
confidence_threshold:
            # get class label and bounding
box coordinates
            class_id = int(detections[0, 0,
i, 1])
            x_left_bottom =
int(detections[0, 0, i, 3] * frame.shape[1])
            y_left_bottom =
int(detections[0, 0, i, 4] * frame.shape[0])
            x_right_top = int(detections[0,
0, i, 5] * frame.shape[1])
            y_right_top = int(detections[0,
0, i, 6] * frame.shape[0])

            # draw bounding box around
detection and label with class name and
confidence
            cv2.rectangle(frame,
(x_left_bottom, y_left_bottom),
(x_right_top, y_right_top), (0, 255, 0), 2)
            cv2.putText(frame, '{}:
{:.2f}%'.format(class_labels[class_id],
confidence * 100), (x_left_bottom,
y_left_bottom - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0),
2)

    # display output frame
    cv2.imshow('frame', frame)

    # break loop if 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# release capture object and close all
windows
cap.release()
cv2.destroyAllWindows()
```

Here's an example Python code for real-time object tracking using the MobileNets algorithm:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

```
import cv2

# set minimum confidence level for object
detection
confidence_threshold = 0.5

# load pre-trained MobileNets model and
corresponding labels
model =
cv2.dnn.readNetFromTensorflow('models/mobile
net_v2.pb')
class_labels = ['background', 'aeroplane',
'bicycle', 'bird', 'boat', 'bottle', 'bus',
'car', 'cat', 'chair', 'cow', 'diningtable',
'dog', 'horse', 'motorbike', 'person',
'pottedplant', 'sheep', 'sofa', 'train',
'tvmonitor']

# initialize tracker
tracker = cv2.TrackerMOSSE_create()

# create capture object for video input
cap = cv2.VideoCapture(0)

# read first frame from video input
ret, frame = cap.read()

# select region of interest for tracking
roi = cv2.selectROI(frame, False)

# initialize tracker with selected region of
interest
tracker.init(frame, roi)

# loop through frames of video input
while True:
    # read frame from video input
    ret, frame = cap.read()

    # update tracker with current frame
    success, bbox = tracker.update(frame)

    # if tracker successfully updated, draw
bounding box around tracked object
    if success:
        (x, y, w, h) = [int(i) for i in
bbox]
        cv2.rectangle(frame, (x, y), (x + w,
y + h), (0, 255, 0), 2)
        cv2.putText(frame, 'Tracking', (x, y
- 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
255, 0), 2)

    # create blob from input frame and pass
through MobileNets model
    blob = cv2.dnn.blobFromImage(frame,
1.0/127.5, (300, 300), (127.5, 127.5,
127.5), swapRB=True, crop=False)
    model.setInput(blob)
    detections = model.forward()

    # loop through detections and filter by
confidence threshold
    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence >
confidence_threshold:
            # get class label and bounding
box coordinates
            class_id = int(detections[0, 0,
i, 1])
            x_left_bottom =
int(detections[0, 0, i, 3] * frame.shape[1])
            y_left_bottom =
int(detections[0, 0, i, 4] * frame.shape[0])
            x_right_top = int(detections[0,
0, i, 5] * frame.shape[1])
            y_right_top = int(detections[0,
0, i, 6] * frame.shape[0])
# if detection overlaps with tracked object,
update tracker with new region of interest
            if x_left_bottom < x+w and
x_right_top > x and y_left_bottom < y+h and
y_right_top > y:tracker.init(frame,
(x_left_bottom, y_left_bottom, x_right_top -
x_left_bottom, y_right_top - y_left_bottom))

# draw bounding box around detection and
label with class name and confidence
            cv2.rectangle(frame,
(x_left_bottom, y_left_bottom),
(x_right_top, y_right_top), (0, 255, 0), 2)
```

V. IMPLEMENTATIONS ANALYSIS

Implementing the above methodologies for real-time object detection and tracking using a standard dataset can provide valuable insights into their effectiveness and efficiency. In this section, we will discuss the implementation and result analysis of these methodologies using the COCO (Common Objects in Context) dataset [13]. To implement frame differencing for object detection and tracking, we used OpenCV's absdiff() function to calculate the absolute difference between two consecutive frames. We then thresholded the result to obtain a binary image indicating the locations of moving objects. We used morphological operations such as erosion and dilation to remove noise and fill gaps in the binary image. Finally, we used contour detection to identify the boundaries of objects in the image.

The results of our implementation of frame differencing on the COCO dataset showed that the method was effective in detecting moving objects in real-time. However, the method was sensitive to lighting changes and generated false positives due to background noise. The accuracy of the method could be improved by combining it with other techniques, such as background subtraction and edge detection. To implement optical flow for object tracking, we used OpenCV's calcOpticalFlowPyrLK() function to track the motion of pixels between two consecutive frames. We initialized the tracking points using a keypoint detection algorithm such as FAST or Harris, and then tracked the points using Lucas-Kanade optical flow. We used a threshold to discard points with low confidence and applied a Kalman filter to smooth the trajectories.

The results of our implementation of optical flow on the COCO dataset showed that the method was effective in tracking the motion of objects in real-time. The method was more robust to lighting changes and could handle complex motion patterns, but it was computationally expensive and could suffer from errors when the camera was moving.To

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 04, April-2023

implement background separation for object detection and tracking, we used OpenCV's BackgroundSubtractorMOG2() function to create a mixture of Gaussian models for each pixel in the scene. We updated the models over time to adapt to changes in the scene and used the foreground mask to detect moving objects. We used morphological operations to remove noise and fill gaps in the binary image.

The results of our implementation of background separation on the COCO dataset showed that the method was effective in detecting moving objects in real-time. The method was more robust to lighting changes and could handle more complex scenes, but it required significant computational resources to create and update the background model. To implement SSD for object detection, we used a pre-trained SSD model from the TensorFlow Object Detection API. We input the image into the network and obtained the predicted object classes and bounding box coordinates. We used non-maximum suppression to remove redundant detections and obtained the final set of object detections.

The results of our implementation of SSD on the COCO dataset showed that the method was efficient and accurate in detecting objects in real-time. The method struggled with detecting small objects or objects with complex shapes, but these issues could be addressed by using improved versions of the SSD architecture. To implement MobileNets for object detection, we used a pre-trained MobileNet model from the TensorFlow Object Detection API. We input the image into the network and obtained the predicted object classes and bounding box coordinates. We used non-maximum suppression to remove redundant detections and obtained the final set of object detections. The results of our implementation of MobileNets on the COCO dataset showed that the method was efficient and accurate in detecting objects in real-time. The method used fewer computational resources than other neural network architectures, making it well-suited for real-time object detection and tracking on devices with limited processing power

VI. CONCLUSION:

In conclusion, each of the above methodologies has its own strengths and weaknesses, and the choice of methodology for real-time object detection and tracking depends on the specific use case and hardware constraints. A combination of multiple methods may be used to improve the robustness and accuracy of the system. Ongoing research in this area is focused on improving the accuracy and efficiency of these methodologies, as well as developing new techniques for real-time object detection and tracking.

REFERENCES

[1] R. S. Shankar, L. V. Srinivas, P. Neelima and G. Mahesh, "A Framework to Enhance Object Detection Performance by using YOLO Algorithm," 2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS), Erode, India, 2022, pp. 1591-1600, doi: 10.1109/ICSCDS53736.2022.9760859.

[2] Shailesh Tiwari · Erma Suryani · Andrew Keong Ng · K. K. Mishra · Nitin Singh Editors Proceedings of International Conference on Big Data, Machine Learning and their Applications ICBMA 2019.

[3] Sharma, Manish and Deshmukh, Rutuja, "Intelligent Recommendation System to Evaluate Teaching Faculty Performance Using Adaptive Collaborative Filtering" John Wiley & Sons, Ltd, isbn 9781119842286, Object Detection by Stereo Vision Images,chapter 9,pages159-169.

[4] Chandan G, Ayush Jain, Harsh Jain, "Real Time Object Detection and Tracking Using Deep Learning and OpenCV", presented at International Conference on Inventive Research in Computing Applications (ICCSP), pp. 0676-0680, 2018- 19.

[5] Keerthana T , Kala L , "YOLO based Detection and Classification of Objects in video records", 2018 IEEE International Conference On Recent Trends In Electronics Information Communication Technology,IEEE Sensors Journal, vol. 15, no. 5, pp. 2679-2691.(RTEICT) 2018, India.

[6] Akshay Mangawati, Mohana, Mohammed Leesan, H. V. Ravish Aradhya, "Object Tracking Algorithms for video surveillance applications", International conference on communication and signal processing (ICCSP),Science,pp.7-30, India, 2018.

[7] Mukesh Tiwari, Dr. Rakesh Singhai, "A Review of Detection and Tracking of Object from Image and Video Sequences" ,International Journal of Computational Intelligence Research ISSN 0973-1873 Volume 13, pp. 745-765,2017.

[8] Manjunath Jogin, Mohana, "Feature extraction using Convolution Neural Networks (CNN) and Deep Learning", 2018 IEEE International Conference On Recent Trends In Electronics Information Communication Technology,(RTEICT)pp.593–605,2018.

[9] Wei Liu and Alexander C. Berg, "SSD: Single Shot MultiBox Detector", Google Inc.,(ICCV), Venice, pp. 2980-2988. Dec 2016.

[10] Leonardo Bombonato ; Guillermo Camara-Chavez ; Pedro Silva , "Real-time single-shot brand logo recognition", published in: 2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI) pp. 0676-0680. 2017.

[11] Amanjeet Kaur; Lakhwinder Kaur ,"Concealed weapon detection from images", published in: 2016 Online International Conference on Green Engineering and Technologies (IC-GET) VLSI, pp. 289-295.2016.

[12] Ce Gao ; Liru Guo ; Miao Sun ; Xingfang Yuan ; Tony X. Han ,"Age Group and Gender Estimation in the Wild With Deep RoR Architecture", published in IEEE Access(ICCSP) pp.0570-0575. 2017.

[13] Dinh Viet Sang ; Le Tran Bao Cuong ; Do Phan Thuan," Facial smile detection using convolutional neural networks", published in 2017 9th International Conference on Knowledge and Systems Engineering (KSE)pp. 1606-1611. 2017.