Tracking With OpenCV

Introduction

OpenCV is a powerful library for computer vision and image processing. It is an open-source library and has a large community of developers. It supports a wide range of programming languages such as C++, Python, Java, etc. OpenCV is used for a variety of applications such as facial recognition, object detection, motion tracking, etc. In this tutorial, we will learn how to track objects using OpenCV.

Object Tracking

For this workshop we will be focusing on object tracking using OpenCV. Object tracking is the process of locating a moving object over time. It has a variety of applications such as surveillance, human-computer interaction, etc. There are different algorithms for object tracking such as MeanShift, CamShift, etc. In this workshop, we will using opencv's built-in object tracking algorithms.

Steps

Prerequisites

1. Python
2. OpenCV
3. Matplotlib

Step 1: Install the required libraries

```
pip install opencv-contrib-python==4.5.5.62 matplotlib
```

Step 2: Import the required libraries

```
import cv2
import matplotlib.pyplot as plt
# Well also be using the built-in time module
import time
```

Step 3: Read the video file

Let's start by reading the video file and displaying it in a preview window. We will be using the `cv2.VideoCapture` class to read the video file.

```
# Read in the video
video = cv2.VideoCapture("race_car.mp4")
ok, frame = video.read()

if not video.isOpened():
    print("Error opening video file")
    exit()
else:
    width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Make the window a quarter of the original size
    cv2.namedWindow("Object Tracker", cv2.WINDOW_NORMAL)
    cv2.resizeWindow("Object Tracker", width // 2, height // 2)

while True:
    ok, frame = video.read()

    if not ok:
        break

    cv2.imshow('Object Tracker', frame)

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

    # Slow down the video so we can see the frames
    time.sleep(0.05)

# Release the video capture object and close the window
video.release()
cv2.destroyAllWindows()
```

Step 4: Drawing in frame

Let's create a few helper functions that will let us draw rectangles and text in the frame.

We'll use matplotlib to display the frames and the tracking results.

```
def drawRectangle(frame, bbox):
    p1 = (int(bbox[0]), int(bbox[1]))
    p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
    cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)


def displayRectangle(frame, bbox):
    plt.figure(figsize=(20, 10))
    frameCopy = frame.copy()
    drawRectangle(frameCopy, bbox)
    frameCopy = cv2.cvtColor(frameCopy, cv2.COLOR_RGB2BGR)
    plt.imshow(frameCopy)
    plt.axis("off")


def drawText(frame, txt, location, color=(50, 170, 50)):
    cv2.putText(frame, txt, location, cv2.FONT_HERSHEY_SIMPLEX, 1, color, 3)



## Video display code...
```

Step 5: Object Tracking

Now that we have the video file and the helper functions in place, let's move on to the object tracking part. OpenCV provides us with a variety of object tracking algorithms, we'll write our code in such a way so we can swap them out easily.

Each of the tracking algorithms has its own strengths and weaknesses. Some are faster, some are more accurate, some are better at tracking objects that change in size, etc.

Here's some properties of a few of them:

- MEDIANFLOW
    - Good for predictable slow motion
- GOTURN
    - Most accurate (deep learning-based)
- MOSSE
    - Fastest

```
# Set up tracker
tracker_types = [
    "BOOSTING",
    "MIL",
    "KCF",
    "CSRT",
    "TLD",
    "MEDIANFLOW",
    "GOTURN",
    "MOSSE",
]

# Change the index to change the tracker type
tracker_type = tracker_types[6]

# Initialize tracker
if tracker_type == "BOOSTING":
    tracker = cv2.legacy.TrackerBoosting.create()
elif tracker_type == "MIL":
    tracker = cv2.legacy.TrackerMIL.create()
elif tracker_type == "KCF":
    tracker = cv2.TrackerKCF.create()
elif tracker_type == "CSRT":
    tracker = cv2.legacy.TrackerCSRT.create()
elif tracker_type == "TLD":
    tracker = cv2.legacy.TrackerTLD.create()
elif tracker_type == "MEDIANFLOW":
    tracker = cv2.legacy.TrackerMedianFlow.create()
elif tracker_type == "GOTURN":
    tracker = cv2.TrackerGOTURN.create()
else:
    tracker = cv2.legacy.TrackerMOSSE.create()


# Video Display code...
```

Step 6: Tracking the object

Now that we have the video file, the helper functions, and the tracker set up, let's put it all together and track the object.

> *Were going to be hard coding the bounding box for demonstration purposes, but in a real-world scenario, you would want to have a way to select the bounding box automatically or though a user interface.*

```
# Define a bounding box
bbox = (1300, 405, 160, 120)

displayRectangle(frame, bbox)



# Initialize tracker with first frame and bounding box
ok = tracker.init(frame, bbox)



# Update video display code to include tracking
while True:
    ok, frame = video.read()

    if not ok:
        break

    # Update tracker
    ok, bbox = tracker.update(frame)

    # Draw bounding box
    if ok:
        drawRectangle(frame, bbox)
    else:
        drawText(frame, "Tracking failure detected", (80, 140), (0, 0, 255))

    # Display Info
    drawText(frame, tracker_type + " Tracker", (80, 60))

    cv2.imshow('Frame', frame)

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

video.release()
cv2.destroyAllWindows()
```

Step 7: Conclusion

That's it! We have successfully tracked an object using OpenCV. We learned how to read a video file, display the frames, and track an object using OpenCV's built-in object tracking algorithms. We also learned how to draw rectangles and text in the frame using OpenCV and Matplotlib.

> *Note: The tracking algorithms are not perfect and may fail in some cases. It is important to choose the right tracking algorithm based on the application.*

Final Code

```
import cv2
import matplotlib.pyplot as plt
import time

# Helper functions to draw rectangle, display rectangle, and draw text


def drawRectangle(frame, bbox):
    p1 = (int(bbox[0]), int(bbox[1]))
    p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
    cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)


def displayRectangle(frame, bbox):
    plt.figure(figsize=(20, 10))
    frameCopy = frame.copy()
    drawRectangle(frameCopy, bbox)
    frameCopy = cv2.cvtColor(frameCopy, cv2.COLOR_RGB2BGR)
    plt.imshow(frameCopy)
    plt.axis("off")


def drawText(frame, txt, location, color=(50, 170, 50)):
    cv2.putText(frame, txt, location, cv2.FONT_HERSHEY_SIMPLEX, 1, color, 3)


# Set up tracker
tracker_types = [
    "BOOSTING",
    "MIL",
    "KCF",
    "CSRT",
    "TLD",
    "MEDIANFLOW",
    "GOTURN",
    "MOSSE",
]

# Change the index to change the tracker type
tracker_type = tracker_types[6]

# Initialize tracker
if tracker_type == "BOOSTING":
```

```
    tracker = cv2.legacy.TrackerBoosting.create()
elif tracker_type == "MIL":
    tracker = cv2.legacy.TrackerMIL.create()
elif tracker_type == "KCF":
    tracker = cv2.TrackerKCF.create()
elif tracker_type == "CSRT":
    tracker = cv2.legacy.TrackerCSRT.create()
elif tracker_type == "TLD":
    tracker = cv2.legacy.TrackerTLD.create()
elif tracker_type == "MEDIANFLOW":
    tracker = cv2.legacy.TrackerMedianFlow.create()
elif tracker_type == "GOTURN":
    tracker = cv2.TrackerGOTURN.create()
else:
    tracker = cv2.legacy.TrackerMOSSE.create()

# Read in the video
video = cv2.VideoCapture("race_car.mp4")
ok, frame = video.read()

if not video.isOpened():
    print("Error opening video file")
    exit()
else:
    width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # Make the window a quarter of the original size
    cv2.namedWindow("Object Tracker", cv2.WINDOW_NORMAL)
    cv2.resizeWindow("Object Tracker", width // 2, height // 2)



# Define a bounding box
bbox = (1300, 405, 160, 120)
displayRectangle(frame, bbox)

# Initialize tracker with first frame and bounding box
ok = tracker.init(frame, bbox)

while True:
    ok, frame = video.read()

    if not ok:
        break
```

```
    # Update tracker
    ok, bbox = tracker.update(frame)

    # Draw bounding box
    if ok:
        drawRectangle(frame, bbox)
    else:
        drawText(frame, "Tracking failure detected", (80, 140), (0, 0, 255))

    # Display Info
    drawText(frame, tracker_type + " Tracker", (80, 60))

    cv2.imshow('Frame', frame)

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

    time.sleep(0.05)

video.release()
cv2.destroyAllWindows()

# Release the video capture object and close the window
video.release()
cv2.destroyAllWindows()
```