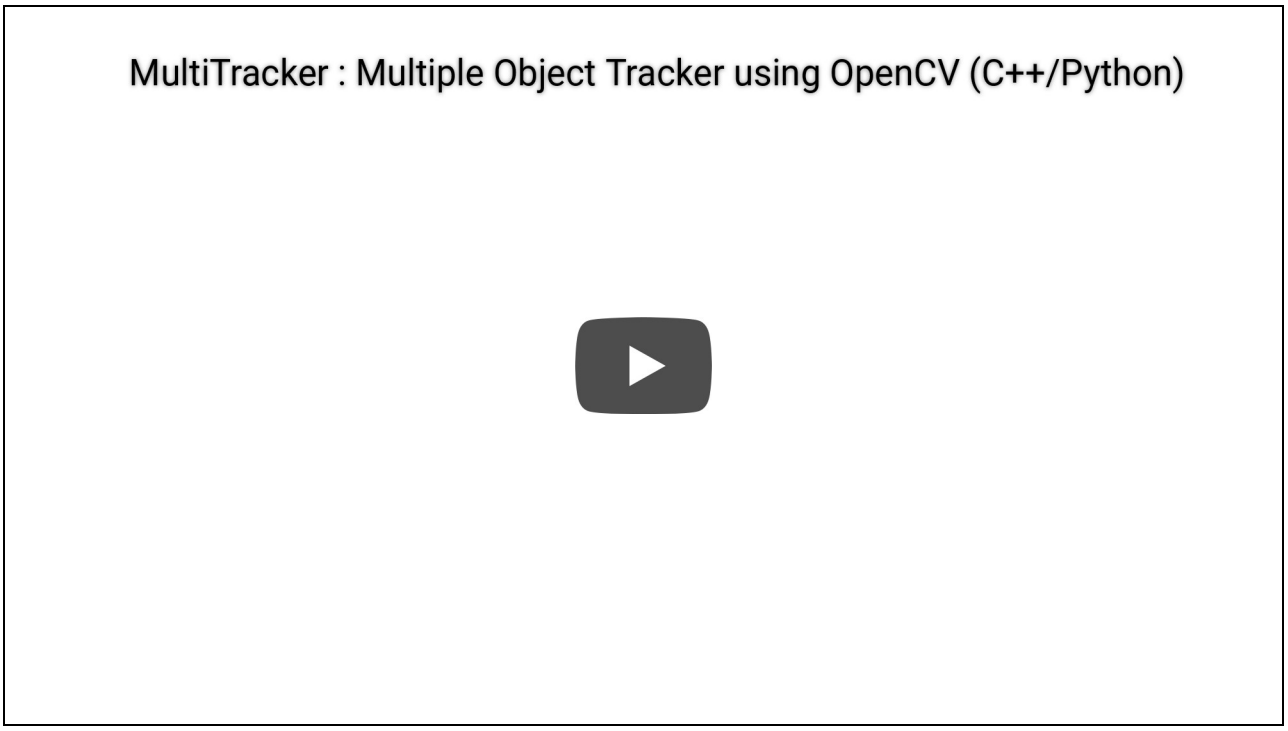
MultiTracker : Multiple Object Tracking using OpenCV (C++/Python)

Satya Mallick (https://www.learnopencv.com/author/spmallick/)

AUGUST 5, 2018



In this post, we will cover how to use OpenCV's multi-object tracking API implemented using the **MultiTracker** class. We will share code in both C++ and Python.

Before we dive into the details, please check previous posts listed below on Object Tracking to understand the basics of single object trackers implemented in OpenCV.

1. Object Tracking using OpenCV (/object-tracking-using-opencv-cpp-python)

Why do we need Multi Object Tracking

Most beginners in Computer Vision and Machine Learning learn about object detection. If you are a beginner, you may be tempted to think why do we need object tracking at all. Can't we just detect objects in every frame?

Let's explore a few reasons why tracking is useful.

First, when there are multiple objects (say people) detected in a video frame, tracking helps establish the identity of the objects across frames.

Second, in some cases, object detection may fail but it may still be possible to track the object because tracking takes into account the location and appearance of the object in the previous frame.

Third, some tracking algorithms are very fast because they do a local search instead of a global search. So we can obtain a very high frame rate for our system by performing object detection every n-th frame and tracking the object in intermediate frames.

So, why not track the object indefinitely after the first detection? A tracking algorithm may sometimes lose track of the object it is tracking. For example, when the motion of the object is too large, a tracking algorithm may not be able to keep up. So many real-world applications use detection and tracking together.

In this tutorial, we will focus on just the tracking part. The objects we want to track will

The **MultiTracker** class in OpenCV provides an implementation of multi-object tracking. It is a naive implementation because it processes the tracked objects independently without any optimization across the tracked objects.

Let's go over the code step by step to find out how can we use OpenCV's multi-object tracking API.

Download Code

To easily follow along this tutorial, please download code by clicking on the button below. It's FREE!

DOWNLOAD CODE (HTTPS://BIGVISIONLLC.LEADPAGES.NET/LEADBOX/143948B73F72A2%3A173C9390C346DC/5649050225344512/)

Step 1: Create a Single Object Tracker

A multi-object tracker is simply a collection of single object trackers. We start by defining a function that takes a tracker type as input and creates a tracker object. OpenCV has 8 different tracker types : **BOOSTING**, **MIL**, **KCF**,**TLD**, **MEDIANFLOW**, **GOTURN**, **MOSSE**, **CSRT**.

If you want to use the GOTURN (/goturn-deep-learning-based-object-tracking/) tracker, please make sure to read this post (/goturn-deep-learning-based-object-tracking/) and

This will be later used to populate the multi-tracker.

Python

```
trackerTypes = ['BOOSTING', 'MIL', 'KCF','TLD', 'MEDIANFLOW', 'G

def createTrackerByName(trackerType):
  # Create a tracker based on tracker name
  if trackerType == trackerTypes[0]:
    tracker = cv2.TrackerBoosting_create()
  elif trackerType == trackerTypes[1]:
    tracker = cv2.TrackerMIL_create()
  elif trackerType == trackerTypes[2]:
    tracker = cv2.TrackerKCF_create()
  elif trackerType == trackerTypes[3]:
    tracker = cv2.TrackerTLD_create()
  elif trackerType == trackerTypes[4]:
    tracker = cv2.TrackerMedianFlow_create()
  elif trackerType == trackerTypes[5]:
    tracker = cv2.TrackerGOTURN_create()
  elif trackerType == trackerTypes[6]:
    tracker = cv2.TrackerMOSSE_create()
  elif trackerType == trackerTypes[7]:
    tracker = cv2.TrackerCSRT_create()
  else:
    tracker = None
    print('Incorrect tracker name')
    print('Available trackers are:')
    for t in trackerTypes:
      print(t)

  return tracker
```

C++

Note: In addition to including opencv2/opencv.hpp, you also need to include opencv2/tracking.hpp.

```
using namespace std;

vector<string> trackerTypes = {"BOOSTING", "MIL", "KCF", "TLD",

// create tracker by name
Ptr<Tracker> createTrackerByName(string trackerType)
{
  Ptr<Tracker> tracker;
  if (trackerType ==  trackerTypes[0])
    tracker = TrackerBoosting::create();
  else if (trackerType == trackerTypes[1])
    tracker = TrackerMIL::create();
  else if (trackerType == trackerTypes[2])
    tracker = TrackerKCF::create();
  else if (trackerType == trackerTypes[3])
    tracker = TrackerTLD::create();
  else if (trackerType == trackerTypes[4])
    tracker = TrackerMedianFlow::create();
  else if (trackerType == trackerTypes[5])
    tracker = TrackerGOTURN::create();
  else if (trackerType == trackerTypes[6])
    tracker = TrackerMOSSE::create();
  else if (trackerType == trackerTypes[7])
    tracker = TrackerCSRT::create();
  else {
    cout << "Incorrect tracker name" << endl;
    cout << "Available trackers are: " << endl;
    for (vector<string>::iterator it = trackerTypes.begin() ; it
      std::cout << " " << *it << endl;
  }
  return tracker;
}
```

Step 2: Read First Frame of a Video

2. Location (bounding boxes) of all objects we want to track.

Given this information, the tracker tracks the location of these specified objects in all subsequent frames.

In the code below, we first load the video using the **VideoCapture** class and read the first frame. This will be used later to initialize the MultiTracker.

Python

```
# Set video to load
videoPath = "videos/run.mp4"

# Create a video capture object to read videos
cap = cv2.VideoCapture(videoPath)

# Read first frame
success, frame = cap.read()
# quit if unable to read the video file
if not success:
  print('Failed to read video')
  sys.exit(1)
```

C++

```
5     vector<Rect> bboxes;
6
7     // create a video capture object to read videos
8     cv::VideoCapture cap(videoPath);
9     Mat frame;
10
11    // quit if unabke to read video file
12    if(!cap.isOpened())
13    {
14      cout << "Error opening video file " << videoPath << endl;
15      return -1;
16    }
17
18    // read first frame
19    cap >> frame;
20
```

Step 3: Locate Objects in the First Frame

Next, we need to locate objects we want to track in the first frame. The location is simply a bounding box.

OpenCV provides a function called **selectROI** that pops up a GUI to select bounding boxes (also called a Region of Interest (ROI)).

In the C++ version, selectROI allows you to obtain multiple bounding boxes, but in the Python version, it returns just one bounding box. So, in the Python version, we need a loop to obtain multiple bounding boxes.

For every object, we also select a random color to display the bounding box.

```python
## Select boxes
bboxes = []
colors = []

# OpenCV's selectROI function doesn't work for selecting multipl
# So we will call this function in a loop till we are done selec
while True:
  # draw bounding boxes over objects
  # selectROI's default behaviour is to draw box starting from t
  # when fromCenter is set to false, you can draw box starting f
  bbox = cv2.selectROI('MultiTracker', frame)
  bboxes.append(bbox)
  colors.append((randint(0, 255), randint(0, 255), randint(0, 25
  print("Press q to quit selecting boxes and start tracking")
  print("Press any other key to select next object")
  k = cv2.waitKey(0) & 0xFF
  if (k == 113):  # q is pressed
    break

print('Selected bounding boxes {}'.format(bboxes))
```

C++

```
85    bool fromCenter = false;
86    cout << "\n==========================================================
87    cout << "OpenCV says press c to cancel objects selection process
88    cout << "It doesn't work. Press Escape to exit selection process
89    cout << "\n==========================================================
90    cv::selectROIs("MultiTracker", frame, bboxes, showCrosshair, fro
91
92    // quit if there are no objects to track
93    if(bboxes.size() < 1)
94      return 0;
95
96    vector<Scalar> colors;
97    getRandomColors(colors, bboxes.size());
```

The **getRandomColors** function is rather simple

```
41    // Fill the vector with random colors
42    void getRandomColors(vector<Scalar>& colors, int numColors)
43    {
44      RNG rng(0);
45      for(int i=0; i < numColors; i++)
46        colors.push_back(Scalar(rng.uniform(0,255), rng.uniform(0, 2
47    }
```

Step 3: Initialize the MultiTracker

Until now, we have read the first frame and obtained bounding boxes around objects. That is all the information we need to initialize the multi-object tracker.

We first create a **MultiTracker** object and add as many single object trackers to it as we have bounding boxes. In this example, we use the **CSRT** single object tracker, but you try other tracker types by changing the **trackerType** variable below to one of the 8

You can also use different trackers wrapped inside the same MultiTracker, but of course, it makes little sense.

The MultiTracker class is simply a wrapper for these single object trackers. As we know from our previous post, the single object tracker is initialized using the first frame and the bounding box indicating the location of the object we want to the track. The MultiTracker passes this information over to the single object trackers it is wrapping internally.

Python

```
# Specify the tracker type
trackerType = "CSRT"

# Create MultiTracker object
multiTracker = cv2.MultiTracker_create()

# Initialize MultiTracker
for bbox in bboxes:
  multiTracker.add(createTrackerByName(trackerType), frame, bbox
```

C++

```
// Specify the tracker type
string trackerType = "CSRT";
// Create multitracker
Ptr<MultiTracker> multiTracker = cv::MultiTracker::create();

// Initialize multitracker
for(int i=0; i < bboxes.size(); i++)
  multiTracker->add(createTrackerByName(trackerType), frame, Re
```

use the **update** method of the MultiTracker class to locate the objects in a new frame. Each bounding box for each tracked object is drawn using a different color.

Python

```
# Process video and track objects
while cap.isOpened():
  success, frame = cap.read()
  if not success:
    break

  # get updated location of objects in subsequent frames
  success, boxes = multiTracker.update(frame)

  # draw tracked objects
  for i, newbox in enumerate(boxes):
    p1 = (int(newbox[0]), int(newbox[1]))
    p2 = (int(newbox[0] + newbox[2]), int(newbox[1] + newbox[3]
    cv2.rectangle(frame, p1, p2, colors[i], 2, 1)

  # show frame
  cv2.imshow('MultiTracker', frame)


  # quit on ESC button
  if cv2.waitKey(1) & 0xFF == 27:  # Esc pressed
    break
```

C++

```
114     // Stop the program if reached end of video
115     if (frame.empty()) break;
116
117     //Update the tracking result with new frame
118     multiTracker->update(frame);
119
120     // Draw tracked objects
121     for(unsigned i=0; i<multiTracker->getObjects().size(); i++)
122     {
123       rectangle(frame, multiTracker->getObjects()[i], colors[i],
124     }
125
126     // Show frame
127     imshow("MultiTracker", frame);
128
129     // quit on x button
130     if  (waitKey(1) == 27) break;
131
132   }
```

Subscribe & Download Code

If you liked this article and would like to download code (C++ and Python) and example images used in this post, please subscribe (https://bigvisionllc.leadpages.net/leadbox/143948b73f72a2%3A173c9390c346dc/5649050225344512/) to our newsletter. You will also receive a free Computer Vision Resource (https://bigvisionllc.leadpages.net/leadbox/143948b73f72a2%3A173c9390c346dc/5649050225344512/) Guide. In our newsletter, we share OpenCV tutorials and examples written in C++/Python, and Computer Vision and Machine Learning algorithms and news.

/5649050225344512/)