Visual tracking: comparing opencv trackers

1 Introduction

The objective of this session is to practice with Visual Tracking. You can get more information about lectures and associated practical sessions on my website: `http://chateaut.fr`. Essential needed knowledges includes a beginning level of computer vision, computer science (Linux, Python programming with `Opencv`, `Numpy`, `Matplotlib`, and `Pytorch` (for sessions on Deep Learning)) and applied mathematics.

The listing 1 shows a python script that can be used to compare several `opencv` trackers.

2 Opencv and Visual Tracking

The `opencv` library includes several trackers. figure 1 shows the inheritance diagram for the class `cv::Tracker` (`https://docs.opencv.org/3.4/d0/d0a/classcv_1_1Tracker.html`) The aim of this practical is to analyse and compare several of these trackers:

- **TrackerBoosting:** [3] the principle of this method is to learn a discriminative model between the object and the background. Online training using an AdaBoost algorithm allows adaptation of the classifier while tracking the object. The resulting model selects the most discriminating features for tracking resulting in stable tracking results. By using fast computable features (e.g. Haar-like wavelets, orientation histograms, local binary patterns) the algorithm runs in real-time.
- **TrackerKCF:** [4] this tracker is based on a discriminative model using correlation filters. Computations are made in the Fourier transform domain and produces a high speed method.
- **TrackerCSRT:** [7] this method is an extension of the previous one and introduces channel and spatial reliability concepts to discriminative correlation filters (DCF).
- **TrackerMedianFlow:** [5] the tracking is performed forward and backward in time and the discrepancies between these two trajectories are measured. The resulting error enables reliable detection of tracking failures and selection of reliable trajectories in video sequences. Basic tracking is achieved by a commonly used normalized cross-correlation (NCC).
- **TrackerMIL:** [1] this tracking-by-detection algorithm learns an discriminative adaptive appearance model online to separate the object from the background. This classifier bootstraps itself by using the current tracker state to extract positive and negative examples from the current frame. Slight drifts can be reduced with a Multiple Instance Learning (MIL) instead of traditional supervised learning.
- **TrackerMOSSE:** [2] like KCF, this tracker uses correlation filters that can track complex objects through rotations, occlusions and other distractions at over 20 times

the rate of current state-ofthe-art techniques. This paper presents a type of correlation filter that uses a Minimum Output Sum of Squared Error (MOSSE) filter. Occlusion is detected based upon the peak-tosidelobe ratio, which enables the tracker to pause and resume where it left off when the object reappears.

- **TrackerTLD**: [6] this method decomposes the long-term tracking task into tracking, learning and detection. The tracker follows the object from frame to frame. The detector localizes all appearances that have been observed so far and corrects the tracker if necessary. The learning estimates detectorâĂŹs errors and updates it to avoid these errors in the future. The authors propose a novel learning method (P-N learning) which estimates the errors by a pair of âĂIJexpertsâĂİ: (i) P-expert estimates missed detections, and (ii) N-expert estimates false alarms. The learning process is modeled as a discrete dynamical system and the conditions under which the learning guarantees improvement are found.

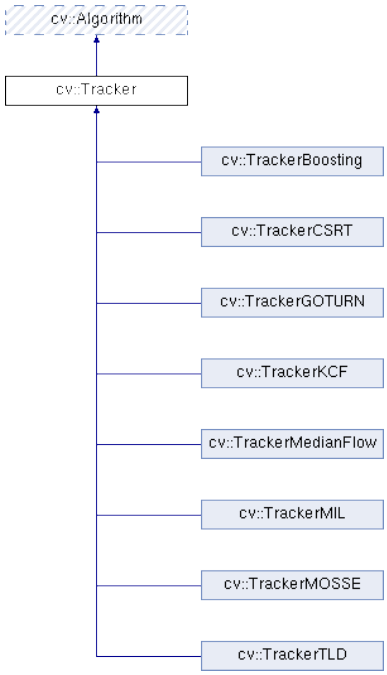

Figure 1: Inheritance diagram for the class `cv::Tracker`

3 Questions

`opencv 3.4` includes a `MultiTracker` class that you should use to compare several trackers `https://docs.opencv.org/3.4/d8/d77/classcv_1_1MultiTracker.html`. Figure 2 illustrates one example of multi tracking with the `track.py` python script that uses this class.

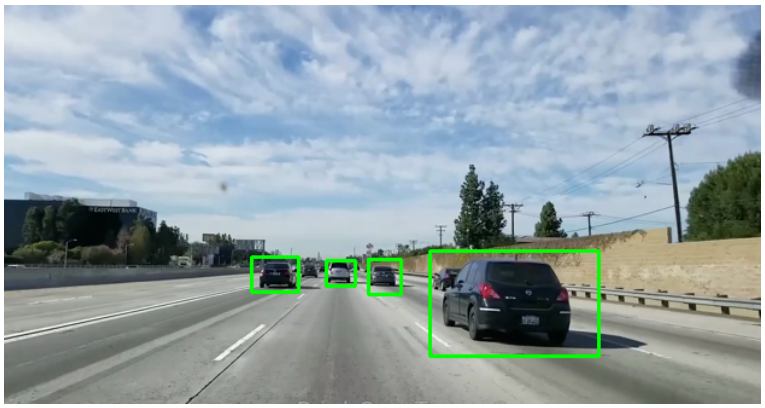

Figure 2: Illustration of script `track.py`

You should start your work using the python script `track.py`. It loads a video file (stored as sequences of image files) into the directory `images`. You have to manually select ROIs of objects to be tracked using the `s` key to start with a new tracker, the mouse to select the ROI and `space` or `return` key to select this tracker. Several trackers may be selected. Once all trackers have been selected, press `q` to start the tracking. Three video sequences are provided: `los_angeles`, `nascar` and `soccer_01`

1. Analyse the script `track.py` to extract the following parts:
   - the command that gets a mouse based selection of a ROI
   - the command that initializes a `MultiTracker` object,
   - the command that updates a `MultiTracker` object,
   - the command that displays the resulting ROI on the current image
2. Test several trackers on the three sequences. What are the characteristics of each sequence in terms of scale variation, motion variation, static or dynamic camera? You may have a look to scientific papers describing the trackers before testing them.
3. Several trackers handle scale variation: which ones?
4. Compute the execution time of each tracker and give a ranking according to their temporal performance. You may use `tic` and `toc` function of library `time`.
5. Modify the script `track.py` in order to initialise several types of trackers from one selection. The aim is to compare trackers.
6. Using only qualitative observation, which is the best tracker for object tracking with scale variation?
7. Using only qualitative observation, which is the best tracker for object tracking with high speed variation?
8. **Additionnal question** : try to build a **MetaTracker**: ie fusion of at least three trackers

Listing 1: `track.py` Python script: test several opencv trackers

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Sep  9 08:36:33 2018
Visual Tracking with particle filter
@author: thierrychateau
"""
import cv2
import os
import numpy as np
import argparse
def main():
    # construct the argument parser and parse the arguments
    ap = argparse.ArgumentParser()
    ap.add_argument("-v", "--video", type=str,
        help="path to input video file")
    ap.add_argument("-t", "--tracker", type=str, default="kcf",
        help="OpenCV object tracker type")
    args = vars(ap.parse_args())
    Basename = 'images/nascar/nascar'   #basename of image files
    ext = 'png' # extension used
    FirstImage = 1 # first image of the sequence
    LastImage = 150 # last  image of the sequence
    step = 1 #step between 2 images
    first = 1 # Flag for first frame
    # initialize a dictionary that maps strings to their corresponding
    # OpenCV object tracker implementations
    OPENCV_OBJECT_TRACKERS = {
        "csrt": cv2.TrackerCSRT_create,
        "kcf": cv2.TrackerKCF_create,
        "boosting": cv2.TrackerBoosting_create,
        "mil": cv2.TrackerMIL_create,
        "tld": cv2.TrackerTLD_create,
        "medianflow": cv2.TrackerMedianFlow_create,
        "mosse": cv2.TrackerMOSSE_create }
    # initialize OpenCV's special multi-object tracker
    trackers = cv2.MultiTracker_create()
    for i in range(FirstImage, LastImage,step):
        filename = "%s%0.5d.%s" % (Basename,i,ext)
        print(filename)
        frame = cv2.imread(filename)
        if first :
            while True :
                cv2.imshow('frame',frame)
                key = cv2.waitKey(1) & 0xFF
                # if the 's' key is selected, we are going to "select" a bounding
```

```
                # box to track
                if key == ord("s"):
                    # select the bounding box of the object we want to track (make
                    # sure you press ENTER or SPACE after selecting the ROI)
                    box = cv2.selectROI("Frame", frame, fromCenter=False, showCross
                    # create a new object tracker for the bounding box and add it
                    # to our multi-object tracker
                    tracker = OPENCV_OBJECT_TRACKERS[args["tracker"]]()
                    trackers.add(tracker, frame, box)
                # if the `q` key was pressed, break from the loop
                elif key == ord("q"):
                    cv2.destroyAllWindows()
                    break
            first = 0
        else:
            # grab the updated bounding box coordinates (if any) for each
            # object that is being tracked
            (success, boxes) = trackers.update(frame)
            # loop over the bounding boxes and draw then on the frame
            for box in boxes:
                (x, y, w, h) = [int(v) for v in box]
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.imshow('frame',frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
if __name__ == "__main__":
    # execute only if run as a script
    main()
```

References

[1] B. Babenko, M. Yang, and S. Belongie. Visual tracking with online multiple instance learning. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 983–990, June 2009.

[2] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2544–2550, June 2010.

[3] Helmut Grabner, Michael Grabner, and Horst Bischof. Real-time tracking via on-line boosting. In *BMVC*, 2006.

[4] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37:583–596, 2015.

[5] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-backward error: Automatic detection of tracking failures. In *2010 20th International Conference on Pattern Recognition*, pages 2756–2759, Aug 2010.

[6] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422, July 2012.

[7] Alan LukeźIăź, Tomáš Vojíř, Luka Čehovin Zajc, Jiří Matas, and Matej Kristan. Discriminative correlation filter tracker with channel and spatial reliability. *Int. J. Comput. Vision*, 126(7):671–688, July 2018.