README.md

Real Time Multiple Object Tracking



In order to use this repo *(at least as I'm doing it on Windows 10)* :

1. install Visual Studio Code
2. download and install Python 3.11.6 with **Windows installer(64-bit)**
3. open VS Code **as an Administrator**
4. install VS Code Python extensions (`Ctrl` + `Shift` + `X`) for example Python Extension Pack
5. create a new folder
6. in the command palette (`Crtl` + `Shift` + `P`) :
    i. Python: Create Environment... -> `.Venv`
    ii. Python: Select Interpreter... -> `Python 3.11.6 64-bit`
7. create and execute a new python file

`test.py`

```
print("test")
```

8. in the python terminal :

Import OpenCV cv2

```
> pip3 install opencv-python
> pip3 install opencv-contrib-python
```

In order to use OpenCVTrackers and import pandas

```
> pip3 install pandas
> pip3 install xlsxWriter
> pip3 install openpyxl
```

In order to use YoloV8 and import imutils, ultralytics

```
> pip3 install imutils
> pip3 install ultralytics
```

In order to use DeepSORT and import sklearn, tensorflow

```
> pip3 install scikit-learn
> pip3 install tensorflow
```

9. download and install **Github** : in your VS Code in the pannel **Source Control** (`Ctrl` + `Shift` + `G`) click on `Download Git for Windows` and download the latest **64-bit** version of **Git for Windows**. During the installation, be aware of selecting the `Use Visual Studio Code as Git's default editor` and `Use Windows' default console window` options with the other default options

10. back in VS Code, connect to your account Github

11. lastly, in the python terminal (in your brand new folder with the right environment)

Paste

```
> git clone https://github.com/lauralvd01/RealTimeMOT.git
```

OpenCVTrackers

> OpenCV 4 comes with a tracking API that contains implementations of many single object tracking algorithms. There are 8 different trackers available in OpenCV 4.2 — BOOSTING, MIL, KCF, TLD, MEDIANFLOW, GOTURN, MOSSE, and CSRT.

See this learnopencv article to learn more about the different trackers and their use.

Here we want to test those 1-object trackers on multiple videos and multiple targets so that we can identify the one we will choose to perform real time multiple object tracking under our situations.

To run those tests, place your `inputVideos` folder in your current folder (or somewhere else and then change all `./inputVideos/` by the right path). In the `inputVideos` folder there must be your videos under a `{videos_type}` subfolder. For instance, mines are placed in `./inputVideos/bateau/` or `./inputVideos/velo/`.

For the results folder, create the `./RealTimeMOT/OpenCVTrackers/outputVideos/` folder, with one subfolder per type of video.

1 Tracker 1 Video 1 Target Test

To test one tracker on one target on one video, use `test.py` : select the tracker within the list `tracker_types`, the input folder `"./inputVideos/{videos_type}/"` and the video name and extension.

Run the file, select the target on the first frame and then press Enter, the output video will be in your `outputVideos` folder.

Initialization of the targets

Before testing all the trackers on all your videos, you have to specify the targets. For this use `initialize.py` : select the input folder `"./inputVideos/`, the subfolder corresponding to the type of the videos you want to initialize `"{videos_type}"` and run the file.

You will select, on the first frame of each video, each target you want to track. Select the box of one target, press Enter, then you can select the next target. After you have selected the last target, double press Enter and you will continue with the next video. If you select a target that is too small or outpasses the limits of the frame, the target will not count and you'll pass to the next video.

The results will be stored in the `"./RealTimeMOT/OpenCVTrackers/dataFiles/init/{videos_type}/"` folder.

Test each tracker on each target of each video

After you have initialized all the targets for all videos of each video type, use `testAll.py` : fill `video_types` with the list of your video types, select the input folder `"./inputVideos/`, the folder with the initializations of your targets `"./RealTimeMOT/OpenCVTracvkers/dataFiles/init/"`, the output folder for the video results `"./RealTimeMOT/OpenCVTrackers/outputVideos/"` and the output folder for the data results `"./RealTimeMOT/OpenCVTracvkers/dataFiles/test/"`.

YoloV8

> YOLOv8 is the latest version of the acclaimed real-time object detection and image segmentation model. YOLOv8 is built on cutting-edge advancements in deep learning and computer vision, offering unparalleled performance in terms of speed and accuracy.
> See more

We use YOLOv8 as a multiple object detector, in order to rapidly identify the objects present in each frame. YOLOv8 is able to detect the objects (the objects on wich he was trained for) in a whole image without having to compute several times some parts of the image.

Use yolov8 to detect objects in a video

The file `yolov8_detect.py` allows to use a custom model and apply it to each frame of the video specified. On each frame, the detections computed by the model are filtered in order to keep the ones that have a confidence score higher than the confidence treshold specified. A video with the same properties as the input is written with the bouding boxes of the selected detections drawn on each frame.

To specify the arguments, you need to change the paths in front of `VIDEO_PATH`, `OUTPUT_DIR`, `MODEL` and `CONF_TRESHOLD`. Default `MODEL` and `CONF_TRESHOLD` are respectively `yolov8n.pt` and `0.3`. `yolov8n.pt` is the latest COCO-pretrained model of YOLOv8, trained on the COCO_dataset.

As outputs you obtain, in the ouput directory you specified, an `img1` folder containing all the original frames of the input video (in `.jpg` format), a `det.txt` file with each detection data, and a `seqinfo.ini` with the input video informations.

The detections data are stored in the MOT format (1 detection per line) :

```
{frame_number},-1,{location.left},{location.top},{width},{height},{score},-1,-1
```

The `seqinfo.ini` file also keeps the MOT format, except for the first line that store the date instead of the sequence number :

```
[Date]
name={video_file_name}
imDir=img1
frameRate={fps}
seqLength={frame_count}
imWidth={frame_width}
imHeight={frame_height}
imExt=.jpg
```

The file `yolov8_app.py` is an old version of the `yolov8_detect.py` file. You shouldn't have to use it, except in case of time mesuring performance, but be careful to cahnge all parameters and arguments. The `yolov8_detect.py` file is truely here to avoid you from thinking of how to change the code for you to use it with your data.

Use yolov8 to train a custom model on a custom dataset

The tutorial is here : Create a dataset with Roboflow and train a custom model on it.

Also, don't hesitate to take a look at the ultralytics documentation.

With these documents, you should be able to create your own custom dataset, export it, and train a model on this dataset (either you start with the latest COCO-pretrained yolov8 model or you continue the training of one of your custom models).

As a reminder, here is the basic command you should execute in your command prompt (after python and ultralytics are installed) to train a model to detection on a certain dataset :

```
yolo task=detect mode=train model=$path_to_the_model_you_want_to_start_with_OR_yo
```

Don't forget there are many options you can use and change to adapt your training process to the situations you to work with : go see the documentation !

DeepSORT

> DeepSORT is an extension of the SORT algorithm of Alex Bewley, a Simple Online and Realtime Tracking algorithm for 2D multiple object tracking in video sequences.
> See more about SORT

> DeepSORT integrates appearance information based on a deep appearance descriptor. The implementation in the `deep_sort-master` folder is the one from Nicolaï Wokje repository, modified in order to be used with YoloV8 detection ouputs.
> See more about DeepSORT

DeepSORT, as it is implemented here, starts with an input video and a detection file, listing all the detections of each frame of the video. Then it goes through the input video to link detections of one frame to the detections of the next frame, in order to assign an identity number to each detection and give the same number to to the detection of the next frame that corrspond to the same object.

So at the beginning, what you have is single images with bounding boxes, unrealated to each other. At the end, for each object detected in one frame, you have its moves through the video, over time : each detection of one frame is related to the bounding boxe that detect the same object in the next frame. Thus you can obtain a video with the bounding boxes drawn with their associated identity number.

To perform this process, you first have to transform the YoloV8 detection ouputs, that are in the MOT format, into a numpy table exported as a npy file. That's what the `tools\generate_detections.py` file does. Then you can run the `deep_sot_app.py` to go through the process. If you want to get the video output, you then have to run the `generate_videos.py` file.

To do so, you either can run those file by a command and list all the arguments, or you can copy the `vscode\launch.json` file to root of your debug folder and change the arguments in this file each time you want to run the DeepSORT algorithm.

The `launch.json` file should go as :

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python: Current File",
            "type": "python",
            "request": "launch",
            "program": "${file}",
            "console": "integratedTerminal",
            "justMyCode": true,
            "args": [
                /*/ Generate_detections
                "--model","./RealTimeMOT/DeepSORT/deep_sort-master/networks/mars-
                "--mot_dir","$path_to_the_yolov8_outputs_folder$",
                "--detection_dir","$path_to_the_yolov8_outputs_folder$",
                "--output_dir","$path_to_your_deepsort_output_folder$"
                /*/
```

```
                /*/ Deep_sort_app
                "--sequence_dir","$path_to_the_yolov8_outputs_folder$",
                "--detection_file","$path_to_the_yolov8_outputs_folder$\detection
                "--output_file","$path_to_your_deepsort_output_folder$\$name_of_y
                "--min_confidence","0.3",
                "--nms_max_overlap","1.0",
                "--min_detection_height", "0",
                "--max_cosine_distance", "0.2",
                "--nn_budget","100",
                "--display","True"
                /*/

                /*/ Generate_videos
                "--mot_dir","$path_to_the_yolov8_outputs_folder$",
                "--result_file","$path_to_your_deepsort_output_folder$\$name_of_y
                "--output_dir","$path_to_your_deepsort_video_output_folder$",
                "--convert_h264","False"
                /*/
            ]
        }
    ]
}
```

Change the arguments as suggested, then transform the `/*/` in `//` just around the arguments of the file you're about to run, then go on this file and run debug.

It should get you, in the ouput directory you provided, the results of your deepsort algorithm as a `txt` file; in the video ouput directory you provided, the video of your results as an `avi` file; and in the input directory, the one where you stored your yolov8 detections results, the same detections results but in a `npy` deepsort-usable format.

Live

The `Live` folder contains some of the last models trained, a beginning of an olgorithm to run both YoloV8 and DeepSORT in a "realtime mode". That means that the algorithm would take a video as an input, and for each frame perform the yolo detection and then the identity deepsort-association before to go with the next frame.