Real Time Vehicle Counting using YOLO-Tiny

Real-time vehicle detection, tracking and counting from surveillance cameras is a main part for many applications in smart cities. Usually, this task encounters some problems in practice, like the lack of real-time processing of the videos or the errors in detection and/or tracking.

Tiny YOLO for detection and
Fast motion estimation for tracking.

The application is running in Ubuntu with GPU processing.

Experimental results show that the approach achieves high accuracy at real time speed (33.5 FPS) on real traffic videos (real-time operation).

The next step: running on a low-budget devices, as Jetson Nano.
Experimental results: ~ 9 FPS using an IP camera.

Considering the pre-trained models provided by the YOLO web site, due to the real time demands of our application we chose to use YOLOv3-tiny version.

YOLOv3-tiny assure the proper balance between accuracy and speed. YOLOv3 is more accurate but too slow for real time processing, even on GPU.

We solve the accuracy problem of YOLOv3-tiny by the fact that every vehicle is for sure detected in at least one of the frames (passing through the active ROI).

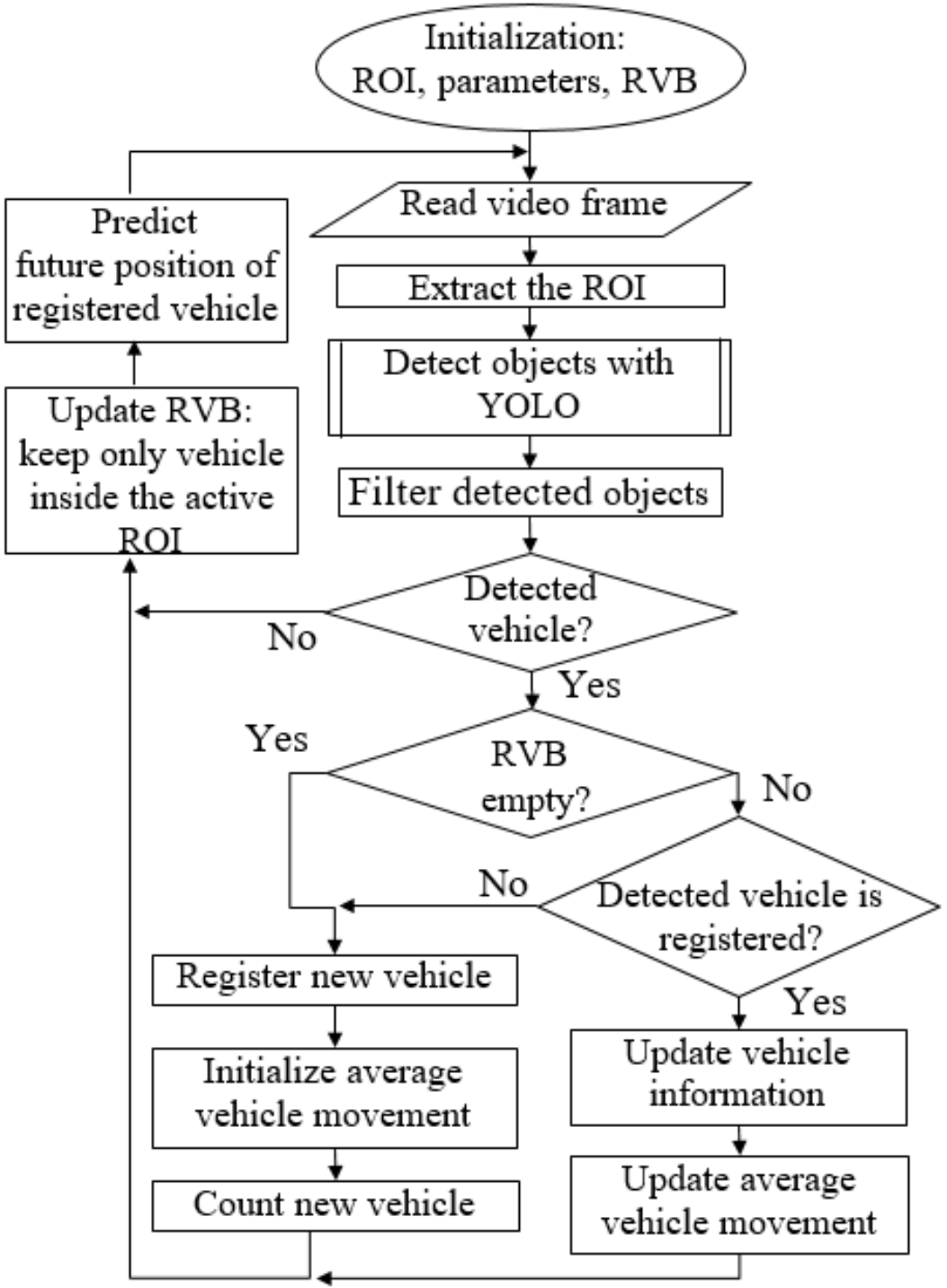
Once we detect a vehicle in the current frame, we track it by checking if it appears detected in the next frames, or, if not detected, we estimate its position in the frame taking in account its last positions and the velocity and direction of movement.

Application flowchart

RVB - Registered Vehicle Buffer

ROI - Region Of Interest

Vehicles: car, truck, bus



For each ROI the object detection method, based on YOLO, is applied.

YOLO-Tiny predicts bounding boxes using anchor boxes and multi-label classification, therefore, for the same object we can have different bounding boxes detected, or we can have the same object classified in two different classes (with different scores). If in the current frame a vehicle is detected twice, we filter the resulted bounding boxes to obtain unique identified vehicle

All the vehicles from the ROI of the current frame are stored in the buffer RVB

At each frame we check if a detected vehicle is already registered in RVB. If not registered, it is a new vehicle, so we register it in RVB and count it.

If the vehicle is already registered, we update the information about vehicle position and movement.

If a registered vehicle is not detected in the current frame, we keep the vehicle in RVB, and predict its future position at the most likely position, considering just the previous positions and its dynamic average movement

The motion estimation is done after each ROI processing. We consider all the present vehicle and estimate the most likely position in the next frame, considering just the previous positions and its dynamic average movement.

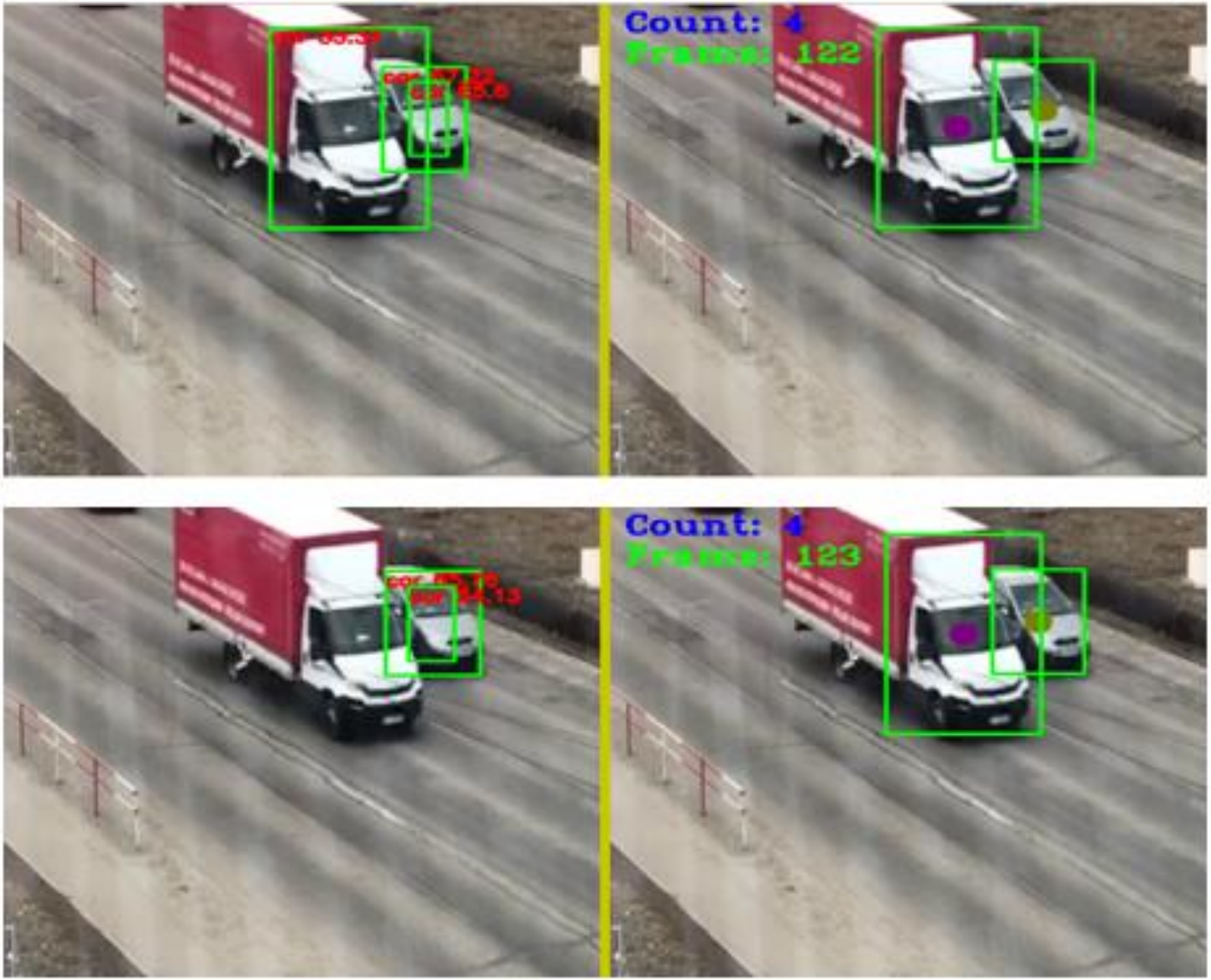
When we are checking if the detected vehicle is registered, we consider the minimal distance of this vehicle to all vehicles in RVB.

The distance is computed considering the estimated location of vehicles for the current frame, not the real location where the vehicles were at the previous frame detection. If we detect that the current vehicle is already in the RVB we update the vehicle information, modifying the estimated location with the real one.

We update the average vehicle movement for a more accurate prediction in the next frame.

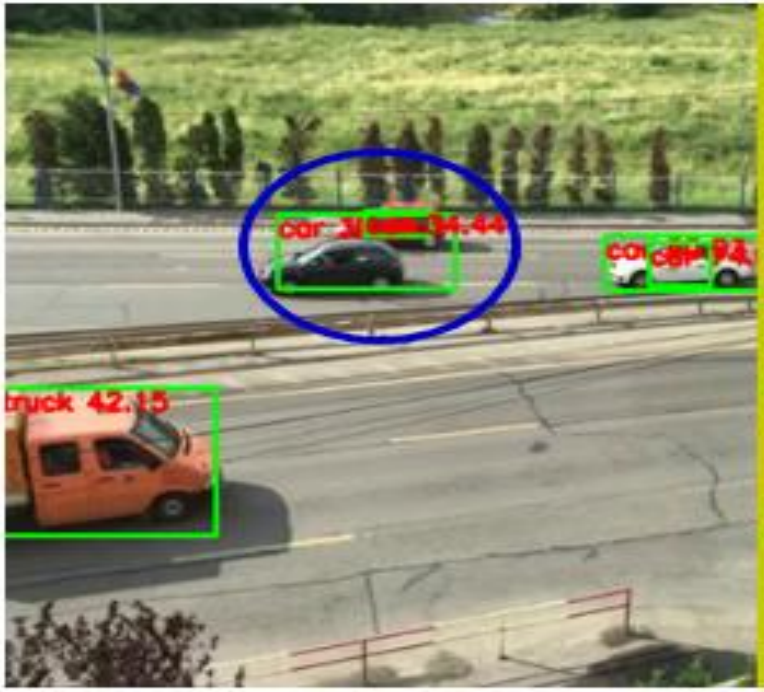
The vehicle detection, correct prediction, tracking and counting is maintained as the vehicles moves on the road

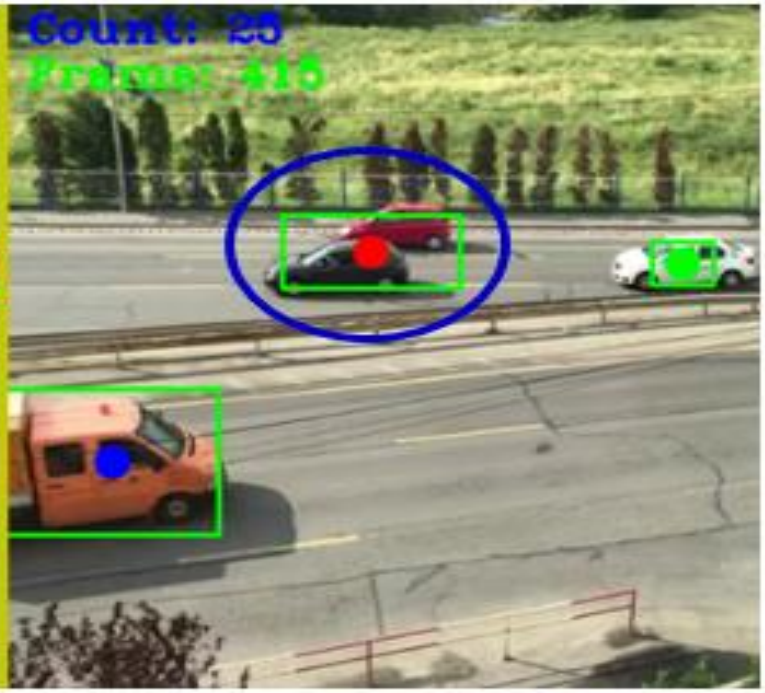
Results

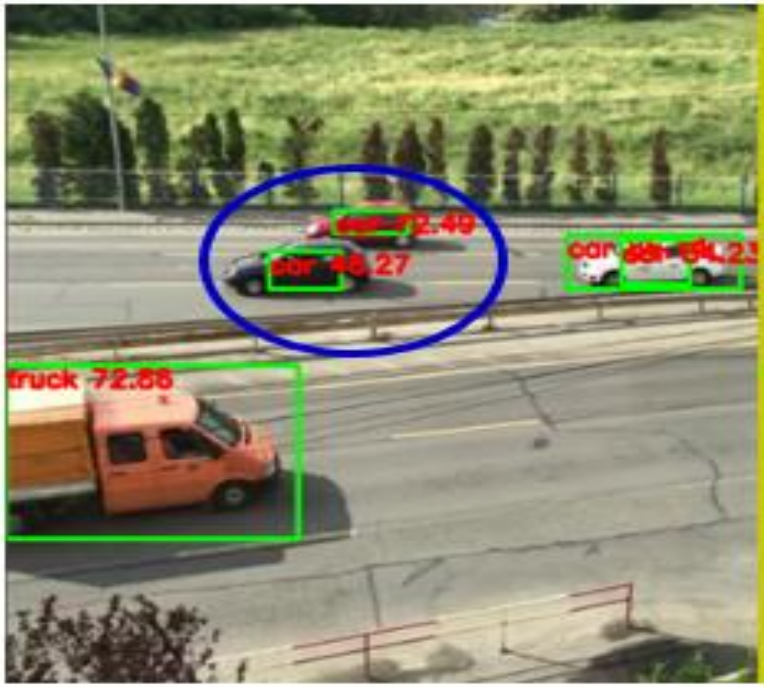


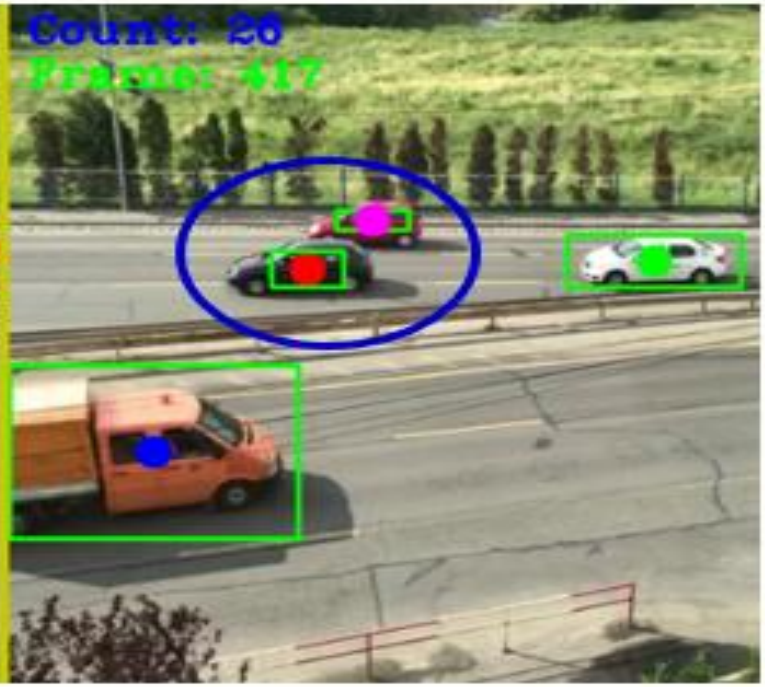

Count: 4
Frame: 128
Count: 4
Frame: 129


Count: 12
Count: 12


truck 42.15


Count: 25
Frame: 416


truck 72.88


Count: 26
Frame: 417