POLITECNICO DI TORINO

Master Course
in ICT for Smart Societies

Master Degree Thesis

Multiple Object Tracking and trajectory prediction for safety enhancement of autonomous driving

Supervisor:
Prof. *Enrico Magli*
Assistant Supervisors:
Eng. *Daniele Brevi*
Eng. *Edoardo Bonetto*

Candidate:
Francesca Pacella

Academic Year 2018-2019

Abstract

In the autonomous vehicles scenario, the V2I communication between vehicles and the road infrastructure is becoming a useful way to increase safety conditions of vulnerable road users. By equipping the infrastructure with a camera, it is possible to handle dangerous situations such as occlusions in a cross road. This can be done by running a Multiple Object Tracking and trajectory prediction algorithm for all road users and inform incoming intelligent vehicles about their position. In this thesis, has been developed a multi-thread framework able to manage the Multiple Object Tracking problem. The framework aims to define the best object tracking algorithm provided by *OpenCV C++* library and to solve the Multiple Object Tracking (MOT) and trajectory prediction problem in order to assist intelligent vehicles in their maneuvers.
The provided solution is an Online and Partially Detection-Based Multiple Object Tracking algorithm. It runs an online Single Object Tracking algorithm for each single target in the scene. Moreover, it receives information about an accurate detection which runs at lower speed and matches each tracker target with its corresponding detection by solving an assignment problem. This allows to reduce tracking drift, improve the online algorithm update and to manage incoming and outgoing objects. The matching algorithm works based on values of extracted features both on the detection side and on the tracking one. These features are geometrical parameters, color distribution and image key points. Thus, the framework acts as a bridge between the detection stage and the tracking one and as a supervisor for each single tracker.
For the *OpenCV* tracking algorithms evaluation, a set of KPIs has been defined. Through this evaluation stage, the best OpenCV tracking algorithm has been identified for the specific use case scenario. The framework il able to process video at high frame rate, since it uses easy and lightweight algorithm for the tracking task and it is not dependent on the detection stage. Moreover, thanks to the match stage with accurate detection, the algorithm does not suffer of tracker drift problems as well as it can keep updated the total number of target to track.
For what concern the trajectory prediction, the *linear Kalman filter* has been identified as a good solution since it meets both low complexity and good performances.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The rapidly development of digital technologies over the past decades and the gradually integration of them in the transport system scenario, has pushed vehicles toward the Intelligent Transport System (ITS) world. Indeed, ITS focuses on providing digital technologies and intelligent devices both on vehicles and on the infrastructure in order to let vehicles be safer, cleaner, and more intelligent. As direct consequence, also communication between them has been developed as well as communication between vehicles and infrastructure. In this way, vehicles had transformed from autonomous systems into Cooperative systems, commonly referred to as *cooperative* ITS (C-ITS) or *car-2-X communication* [1]. According with [1] "The development of C-ITS is primarily driven by applications for active road safety and traffic efficiency, which help drivers to be aware of other vehicles, disseminate warnings about road hazards, and provide real-time information about traffic conditions for speed management and navigation." Thus, when C-ITS is applied to the autonomous vehicles scenario, each single actor can use messages and information exchanged in order to monitor their environment and safely act their maneuvers.
The level of automation of a vehicles have been defined over a scale of six levels by the Society of Automotive Engineers. These levels range from 0 (fully manual) up to 5 (fully autonomous): from level 0 up to level 2 the human monitors the driving environment, while from level 3 on is the automated system that keeps control of it. In details:

- **level 0** NO AUTOMATION: The human perform all driving tasks (steering, acceleration, braking, etc.)
- **level 1** DRIVER ASSISTANCE: The vehicles features a single automated system (e.g. speed control monitoring through cruise control)
- **level 2** PARTIAL AUTOMATION: Advanced driver-assistance systems (ADAS). The vehicle can perform steering and acceleration. The human still monitors all tasks and can take control at any time
- **level 3** CONDITIONAL AUTOMATION: Environmental detection capabilities. The vehicles can perform most driving tasks, but human override is still required

- **level 4** HIGH AUTOMATION: The vehicle performs all driving tasks under specific circumstances. Geofencing is required and human override is still an option.
- **level 5** FULL AUTOMATION: The vehicle performs all driving tasks under all conditions. Zero human attention or interaction is required.

Indeed, in this scenario, a detailed and reliable description of the surrounding environment is a fundamental element to allow vehicles safety operations. These information can be collected through three main different approaches:

- Vehicle-to-vehicle (V2V) communication
- Vehicle-to-infrastructure (V2I) communication
- on board sensors

When a vehicle is equipped with sensors like cameras or radars, it is able to sense its surrounding as it can see through these devices. Since sensors have a limited range and are not always able to effectively manage hidden objects and surprising moves from other cars, V2V and V2I technologies are necessary to let vehicles safety perform their maneuvers. V2V communication enables cars to communicate and exchange data through each other in real time, thereby greatly increasing the distance of a vehicles view. However, while V2V requires that other vehicles on the road are also connected, V2I needs this only for the infrastructure and the car itself. In this scenario, the infrastructure would assist vehicles in case of occlusion, not connected object (like vulnerable users) with the aim of enhance the road safety. V2I allow also to move all computations out of the vehicles and notify them with already processed information related to their surrounding.

The objective of this work is to provide an efficient Multiple Object Tracking and trajectory prediction algorithm based on data collected by a camera placed on a cross road area and on a lane merging point. The algorithm has to provide reliable and real time information about road users, their position and their possible future position.

In the following Sections, 1.1 gives an overview about the context where the system will work, Section 1.2 makes a more detailed explanation of the problem to solve. Finally, Section 1.3 reports what has been implemented to solve the Multiple Object Tracking and trajectory prediction problem.

1.1 Context and Scenario

This thesis is part of the European project ICT4CART and has been developed in collaboration with *Links Foundation*. The main goal of ICT4CART is to design, implement and test in real-life conditions a versatile ICT infrastructure that will enable the transition towards higher levels of automation (up to L4).

An urban intersection in city of Verona (Figure 1.1) is monitored by the ICT4CART infrastructure that is in charge of detect road users, predict their dynamics and broadcast an environmental model of the road users present in the intersection, i.e., indicate the

presence and position of road users that are in the intersection and where they are going. The considered road users are not limited to vehicles, but they include also Vulnerable

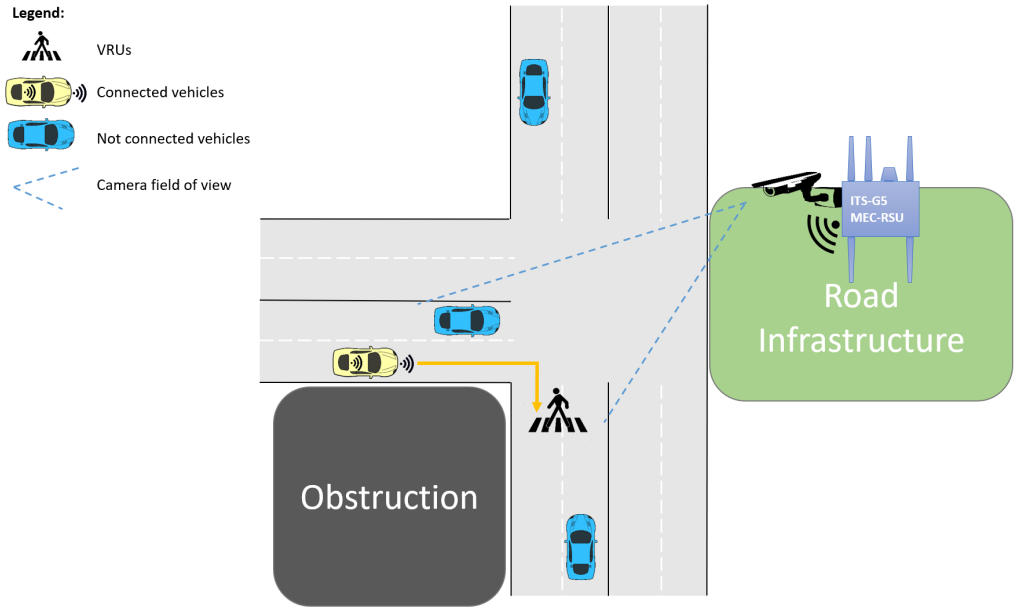


Figure 1.1: Urban intersection use case scenario

Road Users (VRUs) such as pedestrians and bicyclists. The connected and automated vehicle can refine its driving action based on the information received by the ICT4CART infrastructure since it can be aware of road users even if they are in blind spots and the VRUs can be alerted about possible dangers.
Indeed, the ICT4CART infrastructure monitors the intersection exploiting both information from sensors (e.g., magnetic coils, visual cameras) and from CAMs sent by the connected road users. Thus, the infrastructure defines an environmental model of the intersection specifying the position of all the identified road users as well as their dynamics.

Environment Perception model (EPM) is a technology used by intelligent vehicles (a.k.a. autonomous vehicles) to perceive and understand their surroundings in real time. the EPM aims to provide vehicles all needed information about fixed and dynamic obstacles, lane and road detection, traffic sign recognition, vehicle tracking and behavior analysis, and scene understanding. Detailed EPM are required by vehicles to safely perform their movements and avoid that high automation level vehicles step down to lower automation level and handover the partial/full control to the driver.
Each vehicles builds its model starting from data collected by on board LiDAR, cameras, Internal Navigation System and GPS. Vehicles can also share data and information so that each vehicle can enlarge its knowledge about its surrounding. Moreover, vehicles can receive signals from RSUs with environment information.

In ICT4CART project, an environment perception model is build on the roadside infrastructure. This model aims to detect, localize and track moving object on a local reference system.

Another use case of this project is the dynamic adaptation of vehicle automation level on Trento highway (Italy). This scenario focuses on the exit ramp and on the approach of the vehicle to the toll station (Figure 1.2).

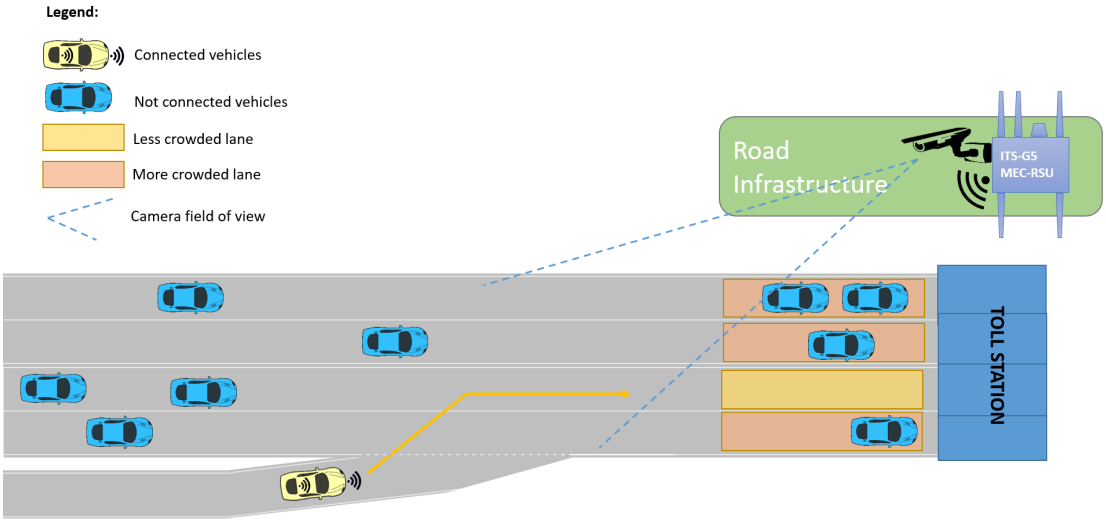


Figure 1.2: Lane merging use case scenario

The ICT4CART infrastructure is in charge of detecting all the involved vehicles and predicting their dynamics while they are leaving the motorway. The ICT4CART infrastructure has to broadcast an environmental model considering the local map of the exit ramp and of the toll station, and the presence of involved vehicles, available toll lanes, and possibly the AD reserved one. Thanks to this information the connected and automated vehicle can refine its driving action, adapting its velocity on the exit ramp, selecting the less congested toll station lane, and possibly down scaling the automation level (or even disengage automation). Another possible application could be the guidance of an autonomous vehicle toward a particular toll lane reserved to self-driving vehicles. Other applications can address gas stations and the approach of vehicles to the refuelling lane. The ICT4CART infrastructure can advise vehicles of possible dangers like trucks in manoeuver, wrong-way driving or broken down vehicles.

1.2 The Multiple Object Tracking (MOT) problem

The task of MOT is largely partitioned to locating multiple objects, maintaining their identities, and yielding their individual trajectories given an input video [4]. Compared with the Single Object Tracking (SOT) problem, MOT has additional issues that make the algorithm development more complex. Indeed, SOT algorithms manly focus on designing appearance models, and/or motion models to deal with challenging factors such as object rotation, illumination changes and scale variation. In addition to this, MOT require to determine the number of object to track that usually change over time and to maintain, for each single object, its identity for all the subsequent frames. Moreover, MOT has to deal with frequent intra-object occlusions, initialization and termination of tracks and similar appearance.

By looking at the MOT algorithms state-of-the-art [4], it is possible to divide these algorithm into 4 main categories, based on how they are initialized and on how the algorithms work for updating their results.
Based on the initialization step, most MOT job can be grouped into two sets: Detection-Based Tracking (DBT) and Detection-Free Tracking (DFT).
Detection-Based Tracking algorithms initialize objects based on data coming from a previous stage of detection. Indeed, Given a sequence of frame, an object detection stage is applied for each single frame to obtain a set of objects. These objects have to be linked together in order to obtain trajectories, one for each subject. This method comes with two main problems: it strongly depends on the detection algorithm used and its performances, and also require that this detection stage is performed at each frame and this can have a negative effect on the final performances due to the detection algorithm latency.
Detection-Free Tracking algorithm instead require a manual initialization of the objects to track on the first frame. Then the algorithm will provide itself the position of the selected algorithm in the subsequent frames. This method free up the algorithm from the detection stage but, on the other hand, it does not have the possibility to understand whether a new object comes in the scene since the number of tracked object is fixed at the first frame.
By looking at the processing mode, MOT can be also categorized into online tracking and offline tracking.
Online tracking methods only rely on the past information available up to the current frame. This means that based on where the objects were in the past instants, the algorithm predict their future location. So based on the up-to-date observations, trajectories are produced on the fly. An online algorithm is also able to learn frame by frame the object appearance and improve its prediction capabilities at each step based on the new object position.
On the other hand, **Offline tracking** require in advance observations from all frames. These batch of data are jointly analyzed in order to estimate a final output. As direct consequence of this method, a real-time MOT cannot be obtained by applying an offline algorithm.

So based on their learning schemes [5] these methods can be defined also offline-learning and online-learning methods. In offline-learning, learning is performed before the actual tracking takes place. In this case, algorithms use supervision from ground truth trajectories to learn a similarity function between detection and tracker for data association. As a result, offline-learning cannot resolve ambiguities, especially when it needs to reassign missed or occluded objects when they appear again. In contrast, online-learning conducts learning during tracking. A common strategy is to construct positive and negative training examples according to the tracking results, and then to train a similarity function for data association. Online-learning is able to utilize features based on the status and the history of the target. However,there are no ground truth annotations available for supervision. So the method is likely to learn from incorrect training examples if there are errors in the tracking results, and these errors can be accumulated and result in tracking drift.

For a better overview of the four categories, in Table 1.1 are reported advantages and drawbacks of each method.

	DBT	**DFT**	**Online**	**Offline**
Advantages	Handle varying number of objects	Not dependent on object detection algorithm	Suitable for real-time applications	Can obtain global optimal solution
Drawbacks	Performance depends on object detection algorithm	Manual initialization	Problems when few past observation available. Tracking drift	No real-time

Table 1.1: Advantages and Drawbacks of different MOT solutions

1.3 Thesis objective and structure

By looking at Table 1.1 and considering the use case scenario described in section 1.1, has been developed an online partially Detection-Based Multiple Object Tracking. The online decision is trivial since the project requires a real time MOT algorithm. Moreover, being the use case scenario a crosswalk in a road intersection, it is necessary a fast and reliable algorithm. This cannot be achieved if the algorithm employs detection results at each frame. An accurate detection algorithm requires deep and sophisticate neural networks which involves latency issues. On the other hand, it is necessary to monitor whether new objects come in the scene, and these can be spotted only through detection. For this reason, has been used an algorithm able to follow its objects on its own. It is initialized through detection results and, based on when detection data are ready to be used, it matches its currently tracked object with the ones provided by the detection. In this way, the algorithm runs separately from the detection one and, at the same time, can keep track of each single object.

Moreover, to better manage each object, the MOT algorithm has been obtained by merging results of multiple Single Object Tracking algorithms. The developed algorithm is a *multi-thread C++* framework where each object is represented by a thread and each thread runs the tracking algorithm for that specific object. For the tracking stage, the object tracking algorithms provided by *OpenCV* library has been used. The main thread has the following jobs:

- verify and supervise each thread work
- notify each thread whether there are new data from the detection stage
- plan threads schedules

Next to the *OpenCV* algorithm, has been developed also a trajectory prediction algorithm. It can be used for both get the tracking algorithm more reliable and to perform the prediction path.
For a more accurate evaluation of which *OpenCV* algorithm to use, a set of KPI has been defined and each algorithm has been tested.

The thesis is organized as follow: Section 2 gives an overview of all tracking algorithm available from the state of the art and a more accurate description for the *OpenCV* tracking algorithms, Section 3 explains how the frameworks works while in Section 4 is described how the object tracking works based on detection results. Section 5 shows how the trajectory prediction has been obtained. Finally, in Section 6 and Section 7 are reported obtained results and KPI evaluation for best tracking algorithm definition, conclusions and future improvements.

Chapter 2

Object tracking

Object tracking is the process of identifying a region of interest in a sequences of frames. Generally, the object tracking process is composed of four modules [2]:

- **target initialization**: is the process of defining object position or region of interest with a bounding box as possible representation. Usually, an object bounding box is provided in the first frame of a sequence and the tracker has to estimate the target position in the consecutive frames.
- **Appearance modeling**: is the process of identifying visual object features for a better representation of the region of interest.
- **Motion estimation**: is the process of estimating the target location in subsequent frames. Its returns a set of possible positions.
- **Target positioning**: is the process of applying posterior prediction in order to identify the most likely position for the target in the new frame.

As reported in Figure 2.1, tracking algorithms can be manly divided into two categories [2]: Correlation Filter Tracker (CTF) and Non-correlation Filter Tracker (NCFT).

2.1 Correlation Filter Tracker

A CF-based tracking algorithm scheme is represented in Figure 2.2 [3]:
Trackers based on correlation filter track an object by correlating a filter over a search window; the position corresponding to the maximum value of the in the correlation output indicates the new position of the target. By starting from an initial representation of the object of interest, the filter is trained starting from variation of this first Region Of Interest (ROI). From this point on, tracking and filter training work together since new estimated position is used to update the filter. This means that at each frame after initialization, an image patch at previous estimated position is cropped as current input. Subsequently, visual features can be extracted for better describe the input, and a cosine window is usually applied for smoothing the discontinuities at window boundaries.

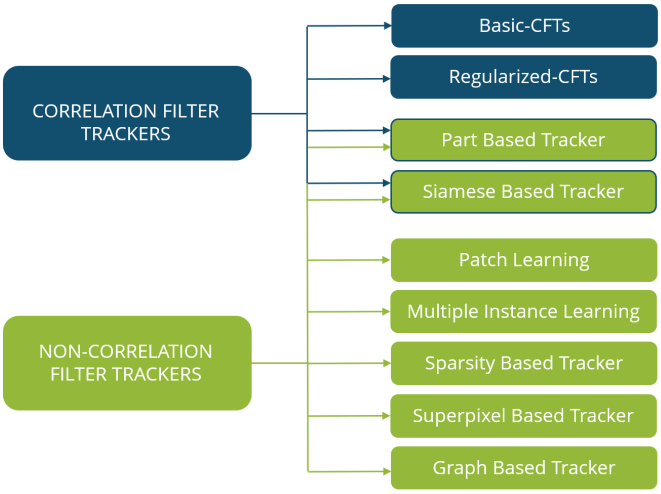


Figure 2.1: Object tracking classification [2]

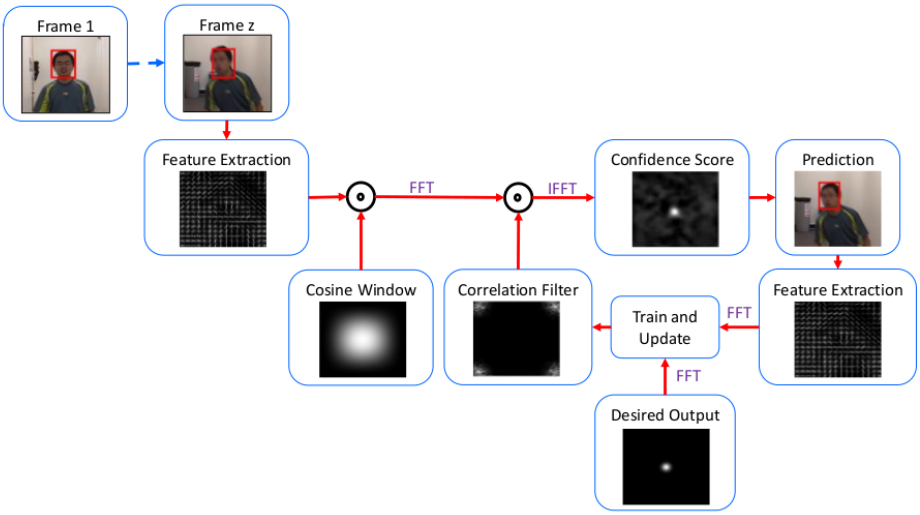


Figure 2.2: Correlation Filter Tracker structure [2]

To create a fast tracker, correlation is computed in the Fourier domain. Indeed, the Convolution Theorem states that correlation becomes an element wise multiplication in the Fourier domain. After the correlation, a spatial confidence map is obtained by Inverse FFT (IFFT), whose peak can be predicted as the new position of target. Lastly, appearance at the newly estimated position is extracted for training and updating the correlation filter with a desired output.

As shown in Figure 2.1, can be divided in *Basic-CFTs* and *Regularized CFTs* and the main difference is on the patch and the filter size. Indeed, Basic-CFTs requires that

these two size are the same and, as consequence, the filter may learn the background form irregularly-shaped objects. The filter update while the object is moving and the tracking algorithm is following it. being the batch and the filter of the same size, if the moving object change its shape and dimension, the filter won't keep it in considerations and will learn its background as part of the object. To solve this problem, Regularized CFTs have been introduced. These filters keep in consideration the background and try to suppress it by assigning weights to the filter coefficient s based on where they are located. As drawback, since background recognition require additional computations usually through deep neural networks, this method pays its higher performances with a bigger latency.
Part Based Correlation Filter Trackers learns the target appearance in parts. In this way, the algorithm better handle target variation due to intra-class variability, clutter, occlusion and deformation.
Lastly, *Siamese Based Correlation Filter Trackers* are that algorithms that join two consequent inputs and produce a single output with the objective of determining whether there are similar objects in the images. In this way the algorithm learns similarity and based on these features it build the filter.

As can be seen by looking at Figure 2.1, these last two methods (*Part Based Tracker* and *Siamese Tracker*), are used also to classify Non-correlation filter trackers since they represent a technique and not a category.

2.1.1 Non-Correlation Filter Tracker

As the name suggest, all those tracking algorithms which do not use correlation filters to predict the new object position are defined as Non-Correlation Filter. In this category, most used solutions are based on neural networks (deep neural network, convolution neural network), particle filters, Linear Discriminant Analysis, Reinforcement Learning, spanning tree, clustering and other machine learning techniques.

Parch Learning Based Trackers exploit both target and background patches. This is manly done through neural networks previously trained on positive and negative examples with the aim of exploiting inner geometry and local structure of the target to distinguish it from the background
Multiple Instance Learning Based Trackers training examples are presented in bags instead of individual and the bags, not the instances, are labeled. One of the most common approach is to take one positive example at the new estimated position and negative ones around it. In this way, if predicted position is not precise, the model get worse over time and cause tracker drift. By considering positive example along with negative ones the model does not downgrade its performances. For this, a bag is labeled positive if it contain at least one positive example, otherwise it is labeled negative.
In *Sparsity based Tracker* the objective is to discover an optimal representation of the target which is sufficiently sparse and minimizes the reconstruction error. A dictionary and a set of coefficients is learned such that, given a set of images, they can be sparsely represented with a small error. These technique is used to discriminate a target from the background patches by sparsely encoding target and background coefficients.

In *Superpixel Based Tracker* pixels information are clustered in order to give perceptual information about rigid structure of pixel grid. Since pixels represent the color intensities of the objects in images, superpixels represent the group of pixels having identical pixel values and same color intensity. Also this technique is used to discriminate target from the background. Indeed, by looking at *"cyan"* superpixel over the frames, it is possible to spot out the sky from a scene.
Finally, according to [2], "*Graph Based Trackers* use superpixels and node to represent the object appearance, while edges represent the inner geometric structure. Another strategy being used in graph-based trackers is to construct graphs among the parts of objects in different frames." This technique is widely used in many applications such as object, human activity recognition and face recognition.

According to Section 1.3, an online single object tracking algorithm has to be implemented. For this reason, the C++ *OpenCV* library has been used.

2.2 OpenCV algorithms

OpenCV (Open Source Computer Vision Library) [22] is an open source computer vision and machine learning C++ software library. It gathers more than 2500 optimized computer vision and machine learning algorithms which can be used detection and recognition tasks, human action classification, object tracking, 3D point cloud production etc.

OpenCV 3.2 provides these six online object tracking algorithms:

1. Boosting 2.2.1
2. Multiple Instance Learning (*MIL*) 2.2.2
3. Kernelized Correlation Filter (*KCF*) 2.2.3
4. Tracking, Learning and Detection (*TLD*) 2.2.4
5. Medianflow 2.2.5
6. Discriminative Correlation Filter with Channel and Spatial Reliability (CSRT) 2.2.6

   In next Sections, more details about each single algorithm are provided.

2.2.1 BOOSTING

This tracker [23] is based on an online version of *AdaBoost*, short for *Adaptive Boosting*, which is a machine learning meta-algorithm. Thus *Boosting* is not a tracking algorithm, but a method of converting a set of weak learners into strong learners. Indeed, the output of an *AdaBoost* classifier is exactly the weighted combination of N weak classifiers. Each weak classifier is *Haar-like feature* based.

Classification is a supervised machine learning instance that, given a set of possible classes and a new object, allow to identify which class the object belong to. The classification task is performed based on a training data set since supervised learning means

that a set of correctly identified observations is available. While training the classifier, significant feature are extracted; in this case, *Haar-like feature* are used. These features are extracted by applying a kernel window of fixed size across the whole image. By starting form top left image corner, the convolution between image pixels and the filter is computed. Example of *Haar features* are reported in Figure 2.3. Indeed, each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle.

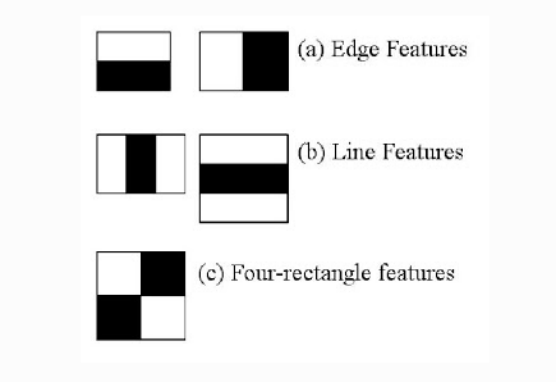


Figure 2.3: Haar feature examples [24]

Thus, the *BOOSTING tracking algorithm* is a combination of online classifiers trained at run time with positive and negative examples of the object. When initialized, the classifier takes as positive example the initial bounding box supplied by an object detection algorithm (if available), or by the user. This first image patch is taken as positive example, while many other image rectangles around the bounding box are treated as background. Then, at the consequent frame, the classifier is run over each pixel around the previous true positive patch position and the its scores are recorded. The new location of the object is the one where the score is maximum. This new position is used to update the classifier since a new positive example is given.

2.2.2 MIL

By looking at the algorithm implementation [10], it turns out that this tracker is an *Online-Multiple Instance Learning Boosting* algorithm. The whole system is made of three components: image representation, appearance model and motion model.
Image representation is handled, as in *BOOSTING tracking algorithm*, by means of Harr-like features (refer to Section 2.2.1 for more details).
The appearance model is a discriminative classifier which evaluate the probability of the object of interest presence in an image patch give the representation of that patch in the feature space. In other world, at each new frame, the classifier compute $p(y = 1|x)$ (where x is an image patch) for a set of image patches that are within a search radious s around the object position in the previous frame. Then, for all image patches in the searching area, the updated target position is defined as the one that, for all patches,

gives the maximum of $p(y|x)$. In other world, being l_t^* the object location at time t, $l(x)$ the image patch location and X^s the search area, the location update is reported in (2.1)

$$l_t^* = l(\underset{s \in X^s}{\operatorname{argmax}}\, p(y|x)) \tag{2.1}$$

where $X^s = \{x|s > ||l(x) - l_{t-1}^*||\}$.
Once the tracker location is updated, the appearance model is update as well. Since *MIL* tracker works with bags of feature (as explained in Section 2.1.1), appearance model update step is performed by extrapolating image patches from a true positive area and a true negative area. Indeed, a set of patches $X^r = \{x|r > ||l(x) - l_t^*||\}$, with $r < s$, are labeled as positive bag while a random subset of patches extracted from the annular region $X^{r,\beta} = \{x|\beta < ||l(x) - l_t^*|| < r\}$, are used to populate the negative bags.

Finally, the discriminative classifier that estimate $p(y|x)$ has to be trained. Also in this case, a Boosting algorithm has been used in order to maximize the log likelihood of labeled bags. As explained in Section 2.2.1, several weak classifiers are combined in order to achieve this goal. In the this case, the k^{th} weak classifier h_k is defined as:

$$h_k(x) = \log \left[\frac{p_t(y = 1|f_k(x))}{p_t(y = 0|f_k(x))} \right] \tag{2.2}$$

where f_k represent the Haar-like frature extracted.

2.2.3 KCF

A reported in Section 2.1, the key point of this algorithm is the link between Furier analysis and the discriminative classifier train step. Indeed, as explained in [13], when training classifiers it is required to deal between latency and accuracy, since a bigger dataset for the online training lead to higher performances while pushing the algorithm far away from the real-time processing skill. In this algorithm, the theory of Circulant matrix is used to add more and more samples by providing the possibility of extremely fast learning and detection with the Fast Fourier Transform.
When performing the update step, each algorithm uses a particular strategy of sparse sampling around the target location. In this case, it has shown that the process of *dense sampling* of an image includes *circular structure*. This assumption comes from the classification step, since it is known that Kernel Trick can improve performances by allowing classification on a rich high-dimensional feature space. With this technique, the inputs are mapped on a feature space by means of a function defined by a kernel k. Then, according with [13], an efficient learning is obtained when the kernel matrix becomes circulant. So given a one-dimensional images x expressed as a nx1 vector with a single feature (ie., the pixel value), samples obtained through dense sampling are defined as:

$$x_i = P^i x, \qquad \forall i = 0, ..., n - 1 \tag{2.3}$$

where P is the permutation matrix that cyclically shifts vectors by one element. In this way, the samples are all possible translated version of x. Then, based on what said

in Section 2.1, the classifier output is computed at each subwindows, in a sliding-window manner.

Thus, this algorithm implement an efficient Kernel Regularized Least Squares solution that uses only Fast Fourier Transform (FFT) and element-wise operations. Indeed, operation done on circular matrix, like multiplication and inversion, can be done element-wise on the original vectors, if they are transformed to the Fourier domain. Finally, according with [13] "For nn images, the proposed algorithm has a complexity of only $\mathcal{O}(n^2 \log n)$, while a naive KRLS implementation would take $\mathcal{O}(n^4)$ operations. This is done without reducing the number of samples, which would sacrifice performance."

2.2.4 TLD

According with the algorithm paper implementation [14], "The tracking framework (TLD) explicitly decomposes the long-term tracking task into tracking, learningand 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 detectors errors and updates it to avoid these errors in the future." Thus, TLD is a framework designed for long-term tracking of an unknown object in a video stream. The components of the framework are reported in Figure 2.2.4.

Tracker estimates the objects motion between consecutive frames under the assump-

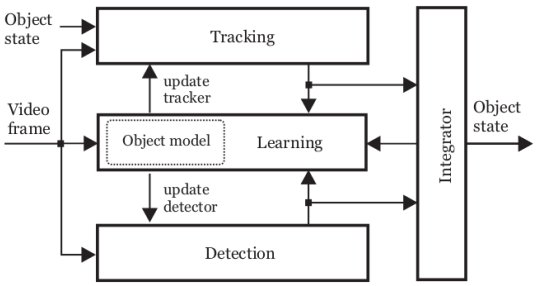


Figure 2.4: Block diagram of the TLD framework [14]

tion that the frame-to-frame motion is limited and the object is visible. It is based on Median-Flow tracker (explained in Section 2.2.5) extended with failure detection.

Detector treats every frame as independent and performs full scanning of the image; for each patch decides about presence or absence of the object.

Learning observes performance of both, tracker and detector, estimates detectors errors and generates training examples to avoid these errors in the future. It assumes that both the tracker and detector can fail. Again based on [14], "The learning paradigm is called P-N learning. The detector responses can be identified by two types of experts. P-expert identifies only false negatives, N-expert identifies only false positives. [...] The crucial element of P-N learning is the estimation of the classifier errors. The key idea is to separate the estimation of false positives from the estimation of false negatives. P-expert analyzes examples classified as negative, estimates false negatives and adds

them to training set with positive label. N-expert analyzes examples classified as positive, estimates false positives and adds them with negative label to the training set. The P-expert increases the classifiers generality. The N-expert increases the classifiers discriminability."
Thus the learning phase is a nearest neighbor classifier trained on the relative similarities between the *Object model* and the considered image patch. Indeed, *Object model* is a data structure that represents the object; it is a collection of positive and negative patches. Given an arbitrary image patch, a set of similarity measure were defined to indicate how much the patch resembles the appearances in the model.

2.2.5 MEDIANFLOW

The innovation introduced by this tracker is the method used to detect failure. Based on the algorithm implementation ([15]), this algorithm implement the Lucas-Kanade tracker with additional constraint for the prediction phase. Indeed, The quality of the point predictions is estimated and for each point it is assigned an error. Then, 50% of the worst predictions are filtered out.
The proposed method for defining the quality of tracked points is based on the forward-backward (FB) consistency assumption which assert that correct tracking should be time-flow independent. FB assumption can be summarized in the following steps:

- **Step 1**: The tracker produces a trajectory by tracking the point forward in time.
- **Step 2**: the point location in the last frame initializes a validation trajectory through backward tracking from the last frame to the first one.
- **Step 3**: the two trajectories are compared and if they differ significantly, the forward trajectory is considered as incorrect.

Finally, according with [15]: "Estimation of the bounding box displacement from the remaining points is performed using median over each spatial dimension. Scale change is computed as follows: for each pair of points, a ratio between current point distance and previous point distance is computed; bounding box scale change is defined as the median over these ratios." For the sake of clarity, a block diagram of the described algorithm is reported in Figure 2.5.

2.2.6 CSRT

This algorithm had introduced the Channel and Spatial Reliability (CSR) concepts to discriminative correlation filters (DCF) tracking. DCF, as already explained in Section 2.1, learns a filter with a predefined response on the training image and it uses circular correlation which allows to implement learning efficiently trough FFT transformation. Based on what reported in [16], Channel and Spatial Reliability can overcome the two main problem of DCF: the limitation on the bounding box shape (must be rectangular and both filter and the patch size to be equal) and the problem of using unrealistic target patch for the train step due to the circular correlation. Indeed, the spatial reliability map

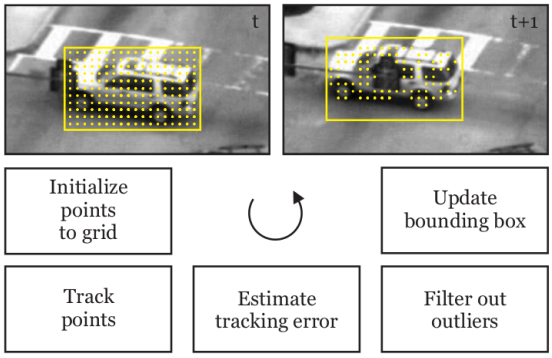


Figure 2.5: Median Flow tracking algorithm [15]

adapts the filter support to the part of the object suitable for tracking. This technique can improving the search range and performance for irregularly shaped objects Thus, by looking at Section 2.1, this algorithm falls in the group of *Regularized CFTs*.

Chapter 3

MOT Framework

The MOT algorithm is a framework which aims to track each single object in the scene as it was the only one. Indeed, the framework is a *multi-thread* system that runs a single thread for each object in the scene. Each one of these threads has to:

- run the tracking algorithm for that specific object
- run the feature extraction and matching algorithm for its target w.r.t. all detection objects

Obviously, all these threads are scheduled and controlled by a main thread. It performs all scheduling tasks, overviews all thread results and puts them together such that each thread will keep working on its original target.
Moreover, since a wide range of single object tracking algorithm are open source and available, the framework should be able to provide a structure that allow to test all of them. Indeed, since has been chosen to use online tracking algorithm (as said in Section 1.3), the framework structure is such that it allow to run any kind of online tracking algorithm. Indeed, being these algorithms structure into the two main steps of *initialization* and *update*, also each single object thread has this form.
Summarizing, the framework has been developed to achieve two main goal:

- evaluate available open-source single object tracking algorithms
- solve the multiple object tracking problem

As described in Section 1.3, the MOT framework relies on detection results for object identification, but at the same time it tracks each single object based on the single object tracking algorithm. For this reason, the framework has to deal with both a detection stage and a tracking stage which work separately on two different hardware. The framework receives the detection results, initializes new trackers, runs already started trackers, stops to track object going out of the scene and matches the tracked objects with their corresponding bounding box provided by the detection stage in order to align and update tracker algorithms data.

In the following Section 3.1 is provided a detailed description of the framework while Section 3.2 describe how the open-source tracking algorithms have been integrated and supervised.

3.1 Framework structure

The framework is a *multi-thread C++* program since it has to be integrated on an embedded system for future developments. In this system, all thread are perfectly synchronized by means of a set of *mutexes* and *condition variables* that allow asynchronous communication between them.
To better understand how it works, Figure 3.1 gives an overview on what the main thread does. In the following each block will be analyzed and its functionalities will be

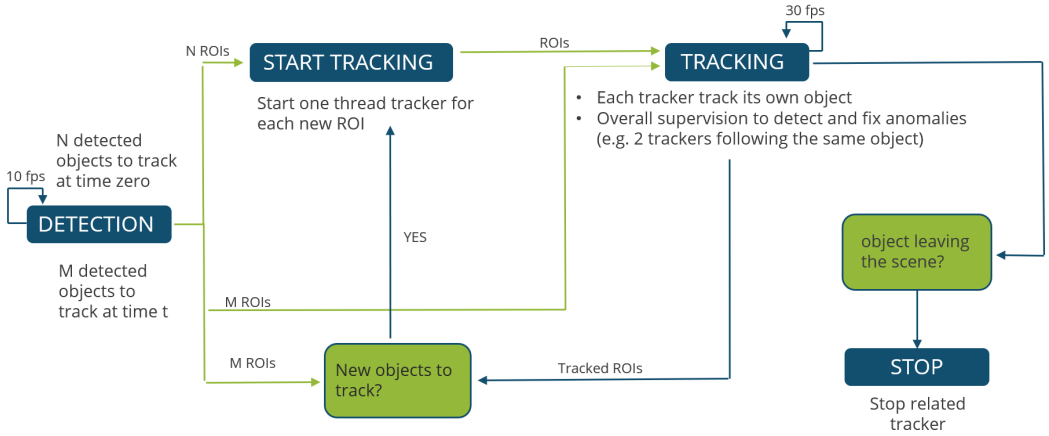


Figure 3.1: Framework operation

explained. **DETECTION** As already said in Section 1.1, being the provided detection stage not able to produce real time data, the framework receives its results and integrate them with the ones of each tracker thread. In this use case, detection data were sent to the main thread through a *JSON* file containing information about each object detected in the scene. It provides, for each target:

- ROI coordinates
- type of object
- confidence level

The ROI coordinates are composed of 4 values: x and y coordinate of the ROI's upper left corner, width and height. The confidence level is a measure on how confident the detection algorithm was while classifying that particular object. Moreover, it is worth noting that as detection algorithm has been used *YOLOv3*. No changes neither network

re-train have been applied; coco dataset weights have been used.
At the very first frame (time zero), the framework receive these data and it use them to initialize all required trackers.

START TRACKING There can be N objects to track at time zero, and these ROIs are used to start N tracker threads, each one implementing the chosen tracking algorithm on the given ROI. Each thread is identified through an ID which is also the object ID.

TRACKING Each tracker tracks its own target while the main thread performs an overall supervision and tasks scheduling for each thread. This means that, every time that a new frame is available or that there are new data from the detection part, the main thread notifies each single tracker. On the other hand, every time that a thread has finished his tasks, it notifies the main thread and waits for its notification before going on. Moreover, each single thread keeps in memory a queue of last n previous positions. These data will be used in the path prediction step (Section 5).

New objects to track? Detection algorithm is assumed to run around 10 FPS. Indeed, every $100ms$, the main thread receives a *JSON* as the one described above. Indeed, at time $t + \Delta_t$, the main thread will see M ROIs present in the scene. Firstly, it has to notify each thread and provide them a data structure containing all useful information. Then, each thread will use these data to perform the matching phase. During this process, each thread measure which detection ROI is more similar to the one of its tracked object. More detail about this phase will be provided later on. After this, once all similarities have been registered by each thread, the main thread used these information to ultimate the matching phase, notify each thread if and which detection ROI has been spotted as the most similar one for that specific thread, and understand whether there are new objects in the scene that require to start a new thread. Section 4 gives a more detailed explanation on how the threads cooperate in order to perform the matching phase. This phase is extremely important since ROI matches are used to re-initialize tracking algorithms and also to understand if new objects have come in the scene.

Object leaving the scene? Thread tracking objects who are leaving the scene have to be stopped in order to avoid misleading situation in the following frames. To handle this task, the easiest solution have been developed. Indeed, if the tracker ROI is placed close to the frame edges and the queue points collected are more than 9, in means that the object is moving from the center of the frame toward the edges and so it is going to exit the scene. This solution is not the more robust one. Indeed, if an object start moving in the same direction of the frame edge, the main thread will always see it as a outgoing object and thus will shout down it thread. When new detection data will be available, that object will be still in the scene, and so the main will start a new thread for it. This loop will keep on until the object is moving in that direction. This problem can be avoided by waiting until no detection is associated to a specific object before turn off its thread and introduce information about its trajectory prediction in order to understand if it is going out of the scene.

STOP Lastly, once the main thread has chosen which objects cannot be tracked

any more, it has to stop their relative thread. This phase require to notify only the interested thread such that it will exit from it tracker loop and conclude all its tasks. This is necessary since the main thread has to join it before it can stop it. Moreover, since the thread and the object ID must be the same, the main thread has to keep in memory that this specific thread has no more data to wait for. It is not deleted from the object positions since it will create disorder in the overall thread organization. Also this approach can be improved in order to do not use such amount of memory. Indeed, this technique require to restart the whole system if the total number of tracked object rise above the hardware memory limit.

In the following, more detail about the tracking block since it represents each single thread tracking code.

3.2 Tracking Thread

As reported in Section 1.3, the idea behind this framework is to let simple and fast object tracking algorithms work together in order to solve a MOT problem. Since the MOT algorithm has to be online, also the SOT algorithm will be of this type. Indeed, each thread has been structured as explained in Figure 3.2 and the used tracking algorithms are the ones provided by the *OpenCV* library describied in Section 2.2.

INIT Once the main thread find out a new ROI, it starts a thread. Each thread

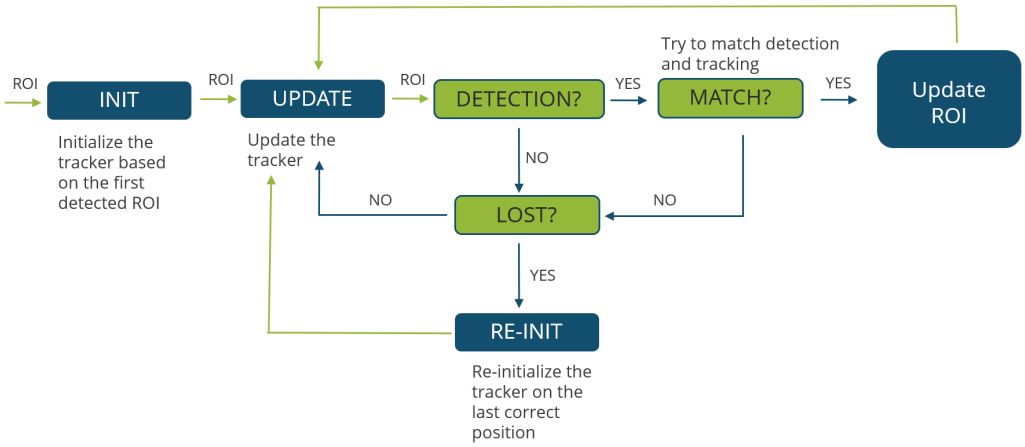


Figure 3.2: Framework operation

first initializes the chosen tracking algorithm based on the first detected ROI. This step depends on which *OpenCV* tracking algorithm has been chosen. Indeed, as explained in Section 2.2, each single algorithm extract different features and use different classifier or machine learning algorithms to learn how its object is done.

UPDATE Once the initialization is done, it starts waiting for a new frame; it waits for a new frame every time it comes back at the update step. When a new frame is

available, it starts updating the tracking parameters and predicts the new target position. Also this step depends on which algorithm has been chosen at initialization time. For example, if the KCF algorithm is selected, at the initialization step the algorithm will evaluate HOG features, and for the update part it will compute a correlation filter map response to define the most probable position in the new frame.

DETECION? After each update, the thread checks whether available information from the detection has been notified. If no data are available, the algorithm goes on. Both new frame and detection data check are done through *shared mutexes* and *condition variables any*. Indeed, *shared mutex* allow multiple threads to access in readable mode the same resource. In this way, threads understand if new detection data have been uploaded by the main thread. In a similar way, *condition variable any* allow the main thread to weak up multiple thread at the same time. This is done to notify them that na new frame is available.

MATCH? If there are information from the main thread about the detection stage, each thread starts a matching phase where it tries to find out which detection corresponds to its tracked target. This stage is more complex and is handle both by each tracker thread and the main thread. Section 4 provides a more detailed explanation of this part.

Update ROI Based on the matching phase output, the thread will update the ROI position with the one provided by the detection. Moreover, since ROI provided by this matching phase are for sure more accurate then the ones of the tracking algorithm, also tracking re-initialization is performed. This is done to reduce the tracker drift which is a consequence of consecutive wrong estimation and parameters update. If no possible detection has been found, the tracker keeps going on its way. The matching part is crucial since a wrong match will exchange object between two or more threads causing confusion on past state collected data.

LOST? Based on which SOT algorithm is running, it may provide information about its confidence on the new predicted position. If the confidence level is below a certain threshold, the algorithm says that it has lost. If that occur, the thread looks for the last correct position available. Indeed, since there are no other possible safe position for the tracker, it step back of one frame assuming that the object has not move to far. Most of the time this technique is useful to recover an higher confidence. Moreover, it is worth noticing that, if the object in not under occlusion, it would be aligned with its detection ROI in, at most, 9 frames.

RE-INIT Based on ROI provided by the lost stage or the matching one, the thread runs the tracking algorithm re-initialization based on the last known correct position.

Chapter 4

Tracking by detection

As said in Section 3.2, the algorithm has to match trackers ROIs and detection ROIs every time that these last one are available. This problem goes under the name of *Image Recognition* or *Image Matching*.
Image matching [6] is a sub domain of computer vision, which focuses on finding a similarity or multiple similarities between a set of images and eventually matching them i.e. considering them the same. This particular task of matching similar images has been accomplished through 2 main steps: **Features Extraction** and **Image Matching algorithm**.
In this scenario, most of the times the matching task is easier than in state-of-the-art use case application. Indeed, give a target, the algorithm does not have to find out whether and where a similar object exist. assuming no latency in the transmission of data between the detection and the tracker framework, the problem to solve is to find out whether there are two identical image patches. The only differences between the two targets to match are their ROI coordinates: the detection ROI is, with more probability, correctly placed around the object while the tracker one, due to drift or occlusion, may be not perfectly including the tracked subject. On the other hand, since it is a MOT problem, the algorithm has to deal with missing detection because of the (intra) object occlusions or detection failures. This means that, in some cases, a tracker that is following an object does not have a corresponding detection ROI only in a few frames. Moreover, in crowd scenarios, both in the detection and in the tracker output more than one ROI can overlap. These problems have been solved by means of an ad hoc feature extraction and matching algorithm, as suggested in [25].

Section 4.1, after a brief overview about state-of-the-art image recognition algorithms, explains how the feature extraction and the matching phase work in the developed algorithm. Extracted features and their evaluation are reported, respectively, in Section 4.2 and 4.4. Section 4.3 explain how the matching algorithm works and, lastly, algorithm results and evaluation are reported in Section 4.5.

4.1 Image recognition

According to [17], appearance based re-id can be considered as a general *image retrieval* problem, where the goal is to find the images from a database that are more similar to the query. In this case, the database is the set of ROIs produced by the detection algorithm and the query is the target wrapped in the tracker ROI.

In order to achieve this goal, the algorithm first extrapolate significant information about each tracker ROI and its neighborhood, and then solve the matching problem by treating it as an assignment problem starting from a cost matrix which costs are chosen based on the extracted features value.
Because of the multi-threading framework structure, it is worth noting that each single thread will extract features related to its own object. It will evaluate the relationship between its object features and those related to detection surrounding targets. Finally, each thread will write down its corresponding costs in the cost matrix. Once all threads have done their matches evaluation, the main thread solves the assignment problem and communicates to each thread its match result.

4.2 Feature extraction

By assuming that the tracker drift never pushes the tracker's ROI far away from the real object position, each tracker should find its corresponding detection ROI in a circular area with center on ROI's center coordinates and ray equal to two times the ROI width (Figure 4.1). Moreover, still because of this assumption, geometrical differences between the ROIs can be used to evaluate the goodness of a possible match. Indeed, three feature have been defined useful for matching algorithm: **Geometrical parameters**, **Colors** and **Corners**.

4.2.1 Geometrical parameters

The geometrical parameters include three measurements between the tracker ROI and the detection ones that fall in the searching area:

- the Euclidean distance;
- the overlap area;
- the ROIs proportion difference;

For each tracker, the Euclidean distance has been calculated between its ROI and all detections ROIs corners by choosing the smallest one, as reported in (4.1), where D represent all available detection ROIs data set.

$$min = \sqrt{(x_{detection_i} - (x_{tracker}))^2 + (y_{detection_i} - (y_{tracker}))^2} \qquad \forall i \in D \tag{4.1}$$

The overlap area is a measure of how much the tracker ROI and the detection ones are occupying the same portion of frame (4.1).

$$overlap = \frac{ROI_{detection_i} \cap ROI_{tracker}}{ROI_{tracker}} \qquad \forall i \in D \tag{4.2}$$

The ROIs proportion difference has been evaluated as reported in (4.3) and it is a measure of how much the two ROIs have similar proportion in their dimensions.

$$R_i = |height_{traker} - height_{detection_i}| + |width_{tracker} - width_{detection_i}| \qquad \forall i \in D \tag{4.3}$$

Figure 4.1 shows the tracker searching area (yellow circle). all that ROIs that fall in this area are then considered for the feature extraction and matching algorithm step. Figure 4.2 explain how the geometrical parameters are evaluated. In each picture, green rectangle represent the tracker ROI while the bue ones are the detection outputs. As can be seen, due to the subject motion and to the tracker drift, the tracker ROI is smaller than the corresponding detection one and it is not perfectly placed on its target (ID 1).

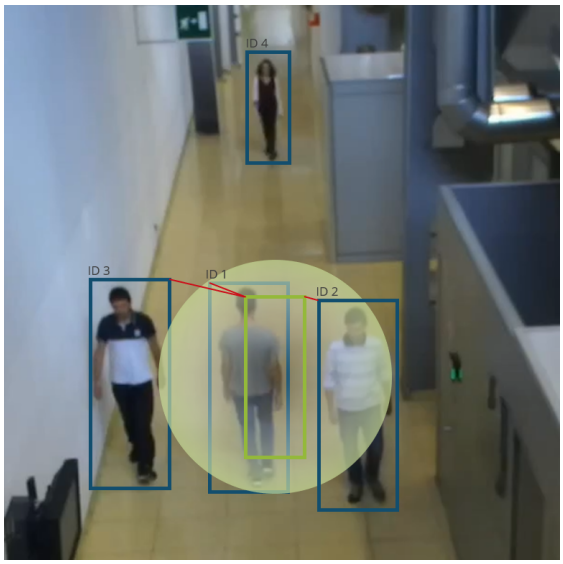
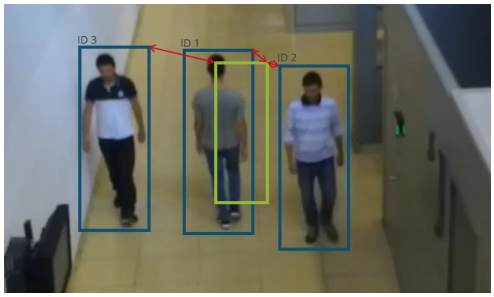
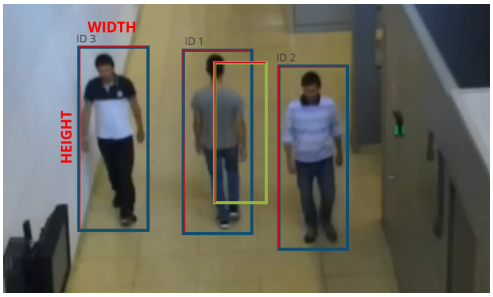


Figure 4.1: Tracker searching area

The first extracted is the distance between the ROIs (red lines in Figure 4.2a). As can be seen, the closest one is ID 2. If only distance were used for the matching algorithm, this situation would lead to a mistake. Then, the relationship between the ROI dimensions is evaluated. If the results is close to zero, it means that the two ROIs have similar proportion between the height and the width. In the case of Figure 4.2b, there is no advantages in using this measure since all detection ROIs have the same dimensions. Finally, the overlap is considered (Figure 4.2c). With this measure is



(a) Distance measure



(b) Diagonal measure

(c) Overlap measure

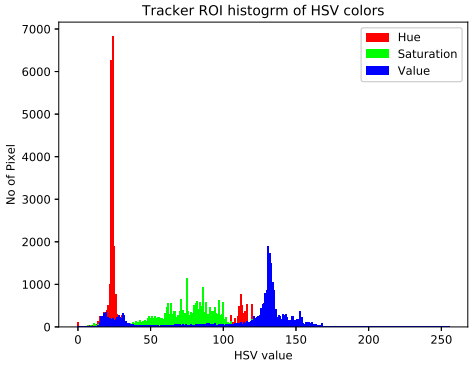
Figure 4.2: Geometrical parameter extraction

evaluated the amount of area that is common to both detection and tracker ROI (red area). In this case, the overlap is the discriminant feature. Indeed, only the ID 1 detection has common area with the tracker ROI.
Even if the overlap result is pointing toward the correct match, there is still the distance which is minimum with the wrong detection. Further features have been added to handle these kinds of situations.

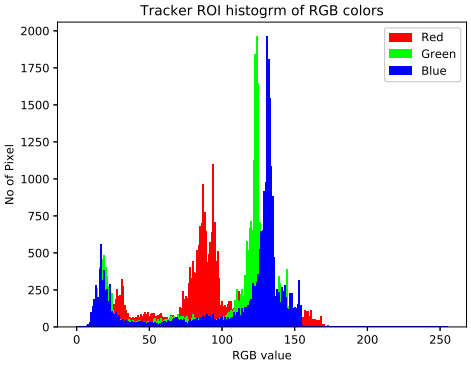
4.2.2 Color Histograms

Thus, a color-based feature has been extracted. Indeed, when pedestrians start walking next to each other, one of the most common feature to look at is the predominant color worn by each actor in the scene.
The idea is to extrapolate color information by each single ROI through histograms of color and then, by measuring the distance between the tracker ROI histogram and the detection ones, find out the best match. By calculating the color histograms, it is evaluated how many pixels have a value (e.g. a color) that fall in a certain range. In this way, a representation of the distribution of the colors in the image is obtained. There are 2 main color space often used in the computer vision word: **RGB** and **HSV**. Figure 4.3 represents RGB and HSV histograms of Figure 4.5b.

By looking at the state-of-the-art about color histogram and color histogram distance, the color space and the distance configuration has been chosen. According with [7], an initial configuration to solve the problem of re-identification is to use the HSV



(a) HSV histogram of Figure 4.5b



(b) RGB histogram of Figure 4.5b

Figure 4.3: Kalman Filter error evaluation over different tracker trajectories

color space along with one of these three possible distances: the Chi Square distance, the Bahattacharyya distance and the Intersection distance.
HSV (Hue, Saturation, Value) is an alternative representation of the RGB color model that is closer to the way humans perceive colors. In this model, as reported in (Figure 4.4), colors of each hue are arranged in a radial slice, around a central axis of neutral color which ranges from black to white. This model represents how paints of different color mix together, and this gives a more realistic representation of the real world color distribution.

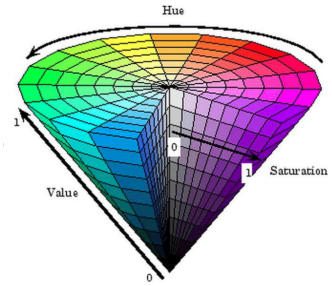
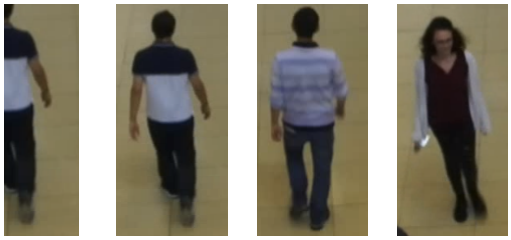


Figure 4.4: HSV color space

Finally, one of the three possible distances has been chosen by looking at their performances in our case scenario. *OpenCV* available color histogram distances are: *Correlation*, *Chi-Square*, *Intersection* and *Bhattacharyya distance*.

These distances have been evaluated and compared for images of Figure 4.5. Figure 4.5a represent the frame cut by the tracker ROI and Figure 4.5b is its corresponding detection image. Other pictures are detection targets that were in the scene. Distances results are reported in Table 4.1.

AS can be seen, all distances give better value when evaluated between histogram of



(a) (b) (c) (d)

Figure 4.5: Example case of matching images

Figures		Correlation ↑		Chi-Square↓		Intersection↑		Bhattacharyya↓	
		RGB	HSV	RGB	HSV	RGB	HSV	RGB	HSV
4.5a	4.5a	1	1	0	0	41.47	27.39	0	0
4.5a	4.5b	0.86	0.90	15.08	8.01	22.41	15.19	0.37	0.27
4.5a	4.5c	0.46	0.55	298.97	2000.95	19.84	17.52	0.69	0.62
4.5a	4.5d	0.26	0.52	893.16	1526.13	19.57	17.80	0.67	0.62

Table 4.1: RGB and HSV Histogram distances

HSV colors. Since the final goal is to use these value to assign scores at each match,has been chosen the method that gives an higher range of values between the bad match and the good match. Indeed, the Chi Square distance has been used.

Additionally, [7] also suggest a number of bin in the range [16-32] and a stripes image division. Thus, the 2D histograms have been evaluated by means of the *OpenCV CalcHist()* function with the following parameters configuration:

- **Channels = [0, 1]** for processing both Hue and Saturation
- **Bins = [50, 60]** 50 for H plane and 60 for S plane.
- **range = [[0,180],[0,256]]** since Hue value lies between 0 and 180 and Saturation lies between 0 and 256.

For what concern the image division in strips, because of the tracker ROI drift, it is more convenient to consider the image as a whole.

Moreover, in case of intra-object occlusion, may be that two different tracker ROIs are containing the same target. In this case, there is no information in the histogram comparison since the other object is not visible. To handle these misleading scenarios, the object histogram has been evaluated only if there is no occlusion between trackers ROIs. Indeed, when the tracker overlap with all surrounding trackers is lower than 20% of its area, the object HSV histogram is evaluated and stored. When detection data are available, last saved color histogram is used.

4.2.3 Feature Detection

Also in this case, the *OpenCV* library has been used. Indeed, it offers a wide range of useful function that can extract representative image point (e.g. corners). The used function is *GoodFeatureToTrack()* being it is also used in the Lucas Kanade's Optical flow algorithm implementation. This function is a corner detector that finds N strongest corners in the image by using the Shi-Tomasi method [21].
The corner detector algorithm runs on the tracker ROI area and on detection ROIs in the searching area. Then, based on the top left corner coordinate of both tracker and detection ROIs, the two images are aligned in order to have the same reference point for pixel coordinate. In this way, if there are extracted points that overlap, this means that they are the same in both images and that the two ROI areas delimit the same image (Figure 4.6). Also in this case, to avoid misleading situations, corners are evaluated only if the tracker ROIs are not overlapping, as explained in Section 4.2.2.

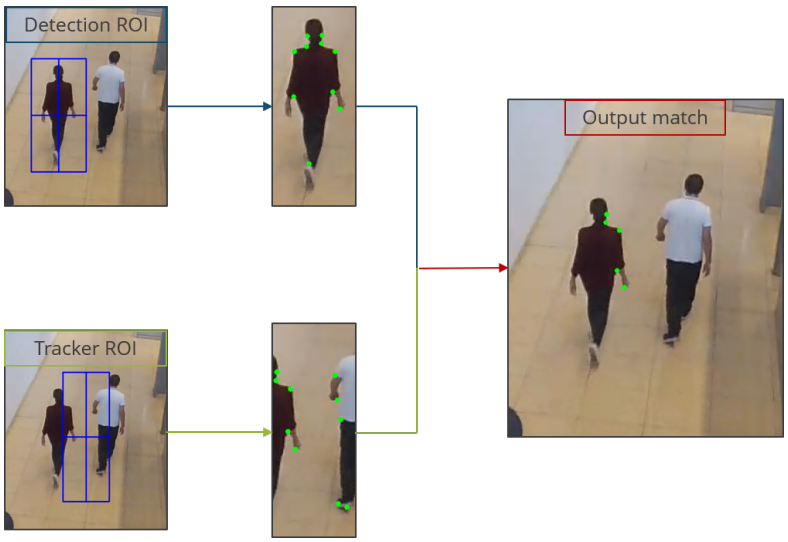


Figure 4.6: Corner extraction and matching

4.3 Matching Algorithm

Once the feature have been extracted, based on their value w.r.t. empirically defined thresholds, a TxD cost matrix has been built where T is the number of active trackers and D are the detection outputs. The cost matrix is a representation of the match goodness. Indeed, if tracker t_j finds in its searching area three detection (d_k, d_p and d_i), on the j-th row of the matrix will be a number greater than zero in correspondence of the k-th, p-th and i-th columns. The costs written in the matrix will depend on the goodness of the extracted feature (explained in Section 4.4).

Once the matrix has been written by all the threads, the main thread solve the assignment problem for the cost maximization as reported in Section 4.4.1.

4.4 Feature evaluation

By extracting these features from a ground truth video, it has been evaluated how the values variate and which of these can be considered a good match indicator. The ground truth has been obtained by running the framework on a video with only one target and with detection data available for each frame. In this way, by extracting features from each tracker and detection ROI, it can be seen how these values can change. Then, by looking at the most frequent feature values, a set of threshold have been defined in order to discriminate between good matches and bad ones, since an higher cost is assigned to better matches.
In the case of study, the procedure has been evaluated offline. A more robust and accurate solution would be a continuously updating function that adjust the parameters trend at each correct match. In this way, when environmental changes occur (e.g. illumination variation, different target to track etc.), threshold values can be updated.

Figure 4.7 show histograms of features values extracted form the ground truth evaluated while using the *KCF* algorithm and detection at each frame. Figure 4.8 report the same feature values collected when matching between detection and tracking were performed every 10 frames. As can be seen, in Figure 4.8 values and ranges are changed since the difference between the tracker ROI and the detection one are more evident than before. Also this problem could be solved by means of an online parameter tune.
Based on Figure 4.7, Figure 4.8 and framework performances, a trial and error approach has been used to define features threshold and their corresponding cost. So, cost matrix values have been defined as reported in Table 4.2. These values give sufficiently good overall performances with all tracking algorithms used and allow to make good comparison between them.

To better understand Table 4.2, it is worth notice that the arrows indicate whether bigger or smaller value refer to better matches and i is the iteration at which that value has been found. Indeed, since values are sorted before cost assignment, the second smallest distance (found at iteration $i = 1$) will be for sure bigger than the previous one. Indeed, if two measures follow in the fist range (the one that assign 10 points to the cost matrix), there would be no discrimination between the two. Thus, by penalizing the value found at the second step, its added cost will be 9.
for the sake of clarity, features values and respectively added costs by tracker ID 1 of (Figure 4.9) are reported in Table 4.3 and matrix 4.4 is the cost matrix related to Table 4.3.

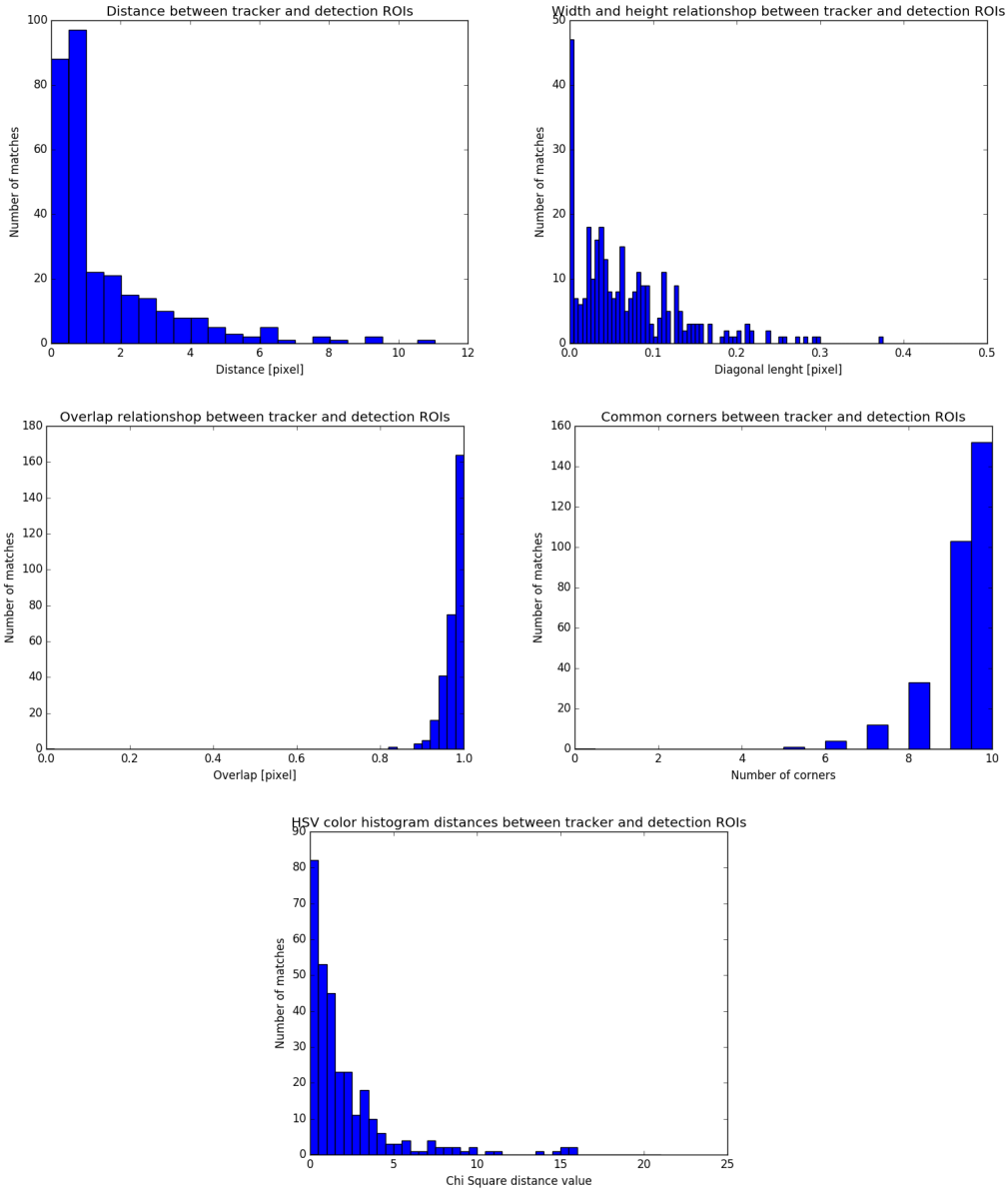


Figure 4.7: Comparison between different features values

4.4.1 Hungarian solution for assignment problem

After features evaluation, each thread will save its corresponding row in the final cost matrix. Matrix 4.4 is the cost matrix written by each thread of Figure 4.9, where, as before, blue rectangle represent detection ROIs and the green ones are the four active threads. In the cost matrix, the row index corresponds to the tracker IDs while the column index are the detection ones.

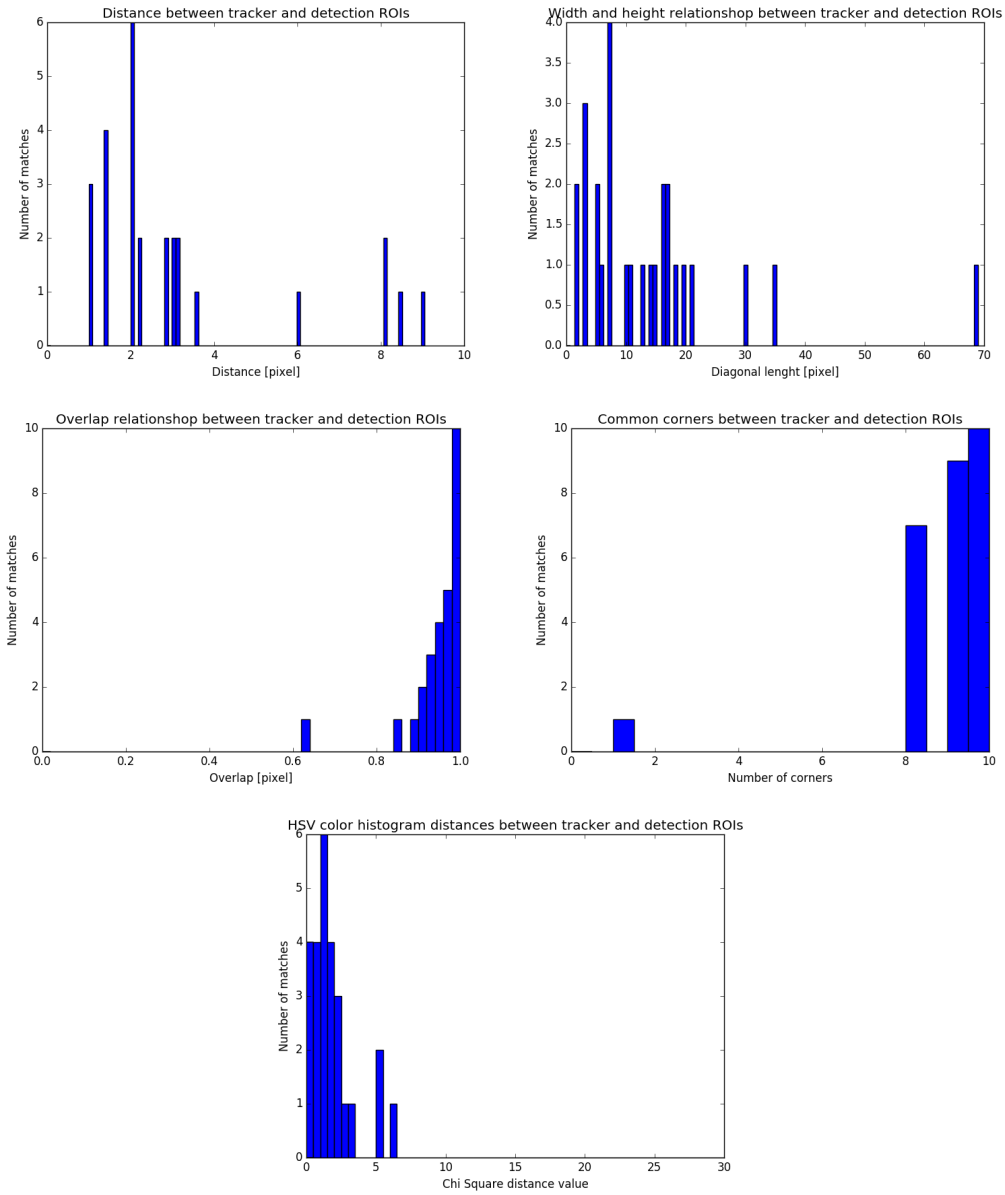


Figure 4.8: Comparison between different features values

$$M = \begin{bmatrix} 40 & 0 & 7 & 9 \\ 15 & 15 & 37 & 8 \\ 11 & 42 & 22 & 8 \\ 5 & 2 & 7 & 39 \end{bmatrix} \tag{4.4}$$

As can be seen, bigger value of each row are placed in the column that corresponds

Feature	Threshold	Cost
Distance $[dist]\downarrow$	$dist \leq \frac{ROI_{width}}{2}$	$10 - i$
	$\frac{ROI_{width}}{2} < dist \leq ROI_{width}$	$5 - i$
Measure proportion $[proportion]\downarrow$	$\Delta \leq 1$	$10 - i$
	$1 < \Delta \leq 2$	$5 - i$
Overlap $[overlap]\uparrow$	$overlap \geq 0.9$	$10 - i$
	$0.7 \leq overlap < 0.9$	$5 - i$
Matching corners $[corners]\uparrow$	$9 \leq corners \leq 10$	$10 - i$
	$8 \leq corners < 9$	$5 - i$
Histograms distance $[hist]\downarrow$	$hist \leq 30$	$10 - i$
	$30 < hist \leq 200$	$5 - i$
	$hist \geq 200$	$-(10 - i)$

Table 4.2: Features extracted thresholds and assigned costs

Feature	*Detection ID 0*	*Detection ID 1*	*Detection ID 2*	*Detection ID 3*
dist value	50.24	26.83	5	36.13
dist score	$5 - 3$	$10 - 1$	10	$5 - 2$
diag value	2	14	15	36
diag score	10	$5 - 1$	$10 - 3$	0
overlap value	0	0	0.91	0.15
overlap score	0	0	10	0
corners value	0	0	10	2
corners score	0	0	10	0
hist value	129.92	153.53	53.91	32.33
hist score	$5 - 2$	$5 - 3$	$5 - 1$	5

Table 4.3: Features value related to Figure 4.9

to the correct detection match. The assignment problem aims to assign at each row only one column without repetition in order to maximize the total cost. In this way, only one detection can be assigned to a given tracker and only one tracker can be assigned to a

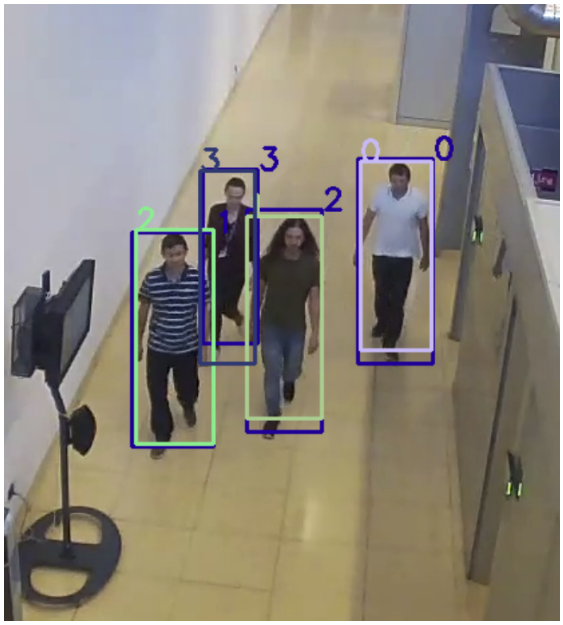


Figure 4.9: Tracker and Detection match

certain detection.
This is a *Generalized Assignment Problem* referable to the well known *Knapsack problem*. In this case, given t trackers and d detection, is defined $c_{i,j}$ as the cost of assigning tracker i to detection j. The objective is maximize M defined as in (4.5)

$$M = \sum_{i=0}^{t} \sum_{j=0}^{d} c_{i,j} x_{i,j} \tag{4.5}$$

subject to constraints of (4.6)

$$\begin{aligned} \sum_{j=0}^{d} x_{i,j} = 1 \qquad & \forall i \in T = \{1, ..., t\} \\ \sum_{i=0}^{t} x_{i,j} = 1 \qquad & \forall j \in D = \{1, ..., d\} \\ x_{i,j} = \begin{cases} 1 & if\ t_i\ assigned\ to\ d_j \\ 0 & otherwise \end{cases} & \end{aligned} \tag{4.6}$$

Moreover, before running the optimization solution, the main thread checks it there are two thread that are choosing as only possible match the same detection ROI. When this situation occurs, it is better to do not involve the corresponding trackers and detection in the optimization step. Thus, the main will remove their corresponding rows and columns from the cost matrix. Of course, the corresponding tracking thread will

not receive a matching ROI to align with.
The optimized solution is found out by applying the Hungarian method, a combinatorial optimization algorithm that solves the assignment problem in polynomial time and finds an optimal assignment for a given cost matrix.
Matrix 4.7 is the matching algorithm output related to 4.4.

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

The Hungarian algorithm has been used since it allow to solve the assignment problem in polynomial time. This algorithm uses a recursive matrix manipulation in order to obtain the optimal solution for the minimization cost assignment problem. Since here is required a maximization cost solution, at the beginning the maximum of the matrix is subtracted to each matrix element. In this way the corresponding matrix for the minimization cost is obtained and the algorithm goes head with the minimization cost solution. This is the resulting balanced matrix used to solve the corresponding minimization problem.

$$M = \begin{bmatrix} 2 & 42 & 35 & 33 \\ 27 & 27 & 5 & 34 \\ 31 & 0 & 20 & 34 \\ 37 & 40 & 35 & 3 \end{bmatrix} \tag{4.8}$$

The three algorithm's steps have been applied to matrix of Figure 4.5 in order to make a more comprehensive explanation of this algorithm.

1. **Step 1** For each row of the matrix, the smallest element is subtracted from every row element

$$M = \begin{bmatrix} 0 & 40 & 33 & 31 \\ 22 & 22 & 0 & 29 \\ 31 & 0 & 20 & 34 \\ 34 & 37 & 32 & 0 \end{bmatrix} \tag{4.9}$$

2. **Step 2** For each column of the matrix, the smallest element is subtracted from every column element

$$M = \begin{bmatrix} 0 & 40 & 33 & 31 \\ 22 & 22 & 0 & 29 \\ 31 & 0 & 20 & 34 \\ 34 & 37 & 32 & 0 \end{bmatrix} \tag{4.10}$$

3. **Step 3** Cover all zeros in the matrix using minimum number of horizontal and vertical lines. In this case, each column contains a zero, so this step is not required.

4. **Step 4** Test for *Optimality*: if the minimum number of covering lines is equal to the matrix dimension, an optimal assignment is possible. If not, go to **Step 5**

$$M = \begin{bmatrix} \cancel{0} & \cancel{40} & \cancel{33} & \cancel{31} \\ \cancel{22} & \cancel{22} & \cancel{0} & \cancel{29} \\ \cancel{31} & \cancel{0} & \cancel{20} & \cancel{34} \\ \cancel{34} & \cancel{37} & \cancel{32} & 0 \end{bmatrix} \tag{4.11}$$

5. **Step 5** Determine the smallest entry not covered by any line. Subtract this entry from each uncovered row, and then add it to each covered column. Return to Step 3.

From **Step 4**, since 4 line were required to cover all zeros, the optimal solution was found out. Zero positions are the ones corresponding to the initial problem solution.

Finally, it is important to underline that, whatever a new object comes in the scene, the matching algorithm is able to understand this and so it will start a new tracker thread for that new object. Indeed, when the matching algorithm is done, each thread will receive information about which ROI it should be matched with. Last word is given to the single thread, which can decide to re-initialize its tracker on the matched detection ROI or not. It looks back at its matches and if the matched ROI is within its range of choices. If not, the tracker will not use that ROI and will notify the main thread about its decision. At this point, the main thread realize that there is a ROI detection that cannot be associated to any tracker, and so it decide that it will be a new object to track and will start its own thread. In the same way, if from the matching phase the main thread ends up with more detection than the total number of active threads, it will use the remaining ROIs to start new trackers.

Last observation about the matching phase is related to intra-object occlusion problem. Indeed, as happen in frame 50 of Figure 4.12, tracker ID 3 has not been associated to any detection. In fact, it bounding box is not present in the picture. This situation are handled by manipulating the cost matrix before the optimization step. Indeed, if more then one row have as only choice the same detection, the corresponding tracker rows in the matrix and the detection column are eliminated. In this way, these object are not considered while solving the assignment problem. Instead, they are matched in dedicated assignment problem. Thus, the main thread looks at each tracker detection match score and, based on their values, it match one tracker to the only available detection while the other one is let go head based only on the tracking algorithm output.

4.5 Algorithm evaluation

By considering an highly accurate detection algorithm, a latency of 10 frame between each detection output has been adopted. In these 10 frames of only tracking, each thread follows its object only tanks to the used tracking algorithm. Once the detection arrive, the matching algorithm tries to associate each tracker with a detection. In Figure 4.10 is

reported a sequence of matches in which the algorithm is properly working. Indeed, ROIs of the same color represent tracker and detection which have been associate. As can be seen, tracker *ID 2*, after 10 frames of occlusion, is associated at the correct detection.

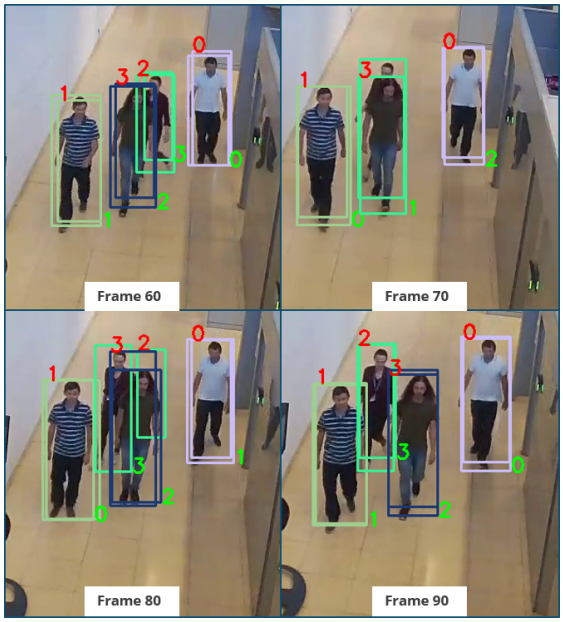


Figure 4.10: Matching algorithm successful matches

To better highlight the contribution of each extracted feature on the overall tracking performances, the framework has been run by using each feature separately and some combination. Figure 4.11 reports a sequence of frame where only geometrical feature were used, while in Figure 4.12 all features were used to write down the cost matrix. Moreover, the algorithm has been tested also on a 4K traffic camera video [27]. Results are reported in Figure 4.13.

As can be seen, at frame 60 of Figure 4.11 tracker ID 1 has exchanged its identity with tracker ID 3. At frame 50, detection of tracker ID 3 is not available because of the intra-object occlusion and tracker algorithm of ID 3 has lost its target. Instead, when all feature have been extracted and used in the matching phase (Figure 4.12), the mistake does not occur.

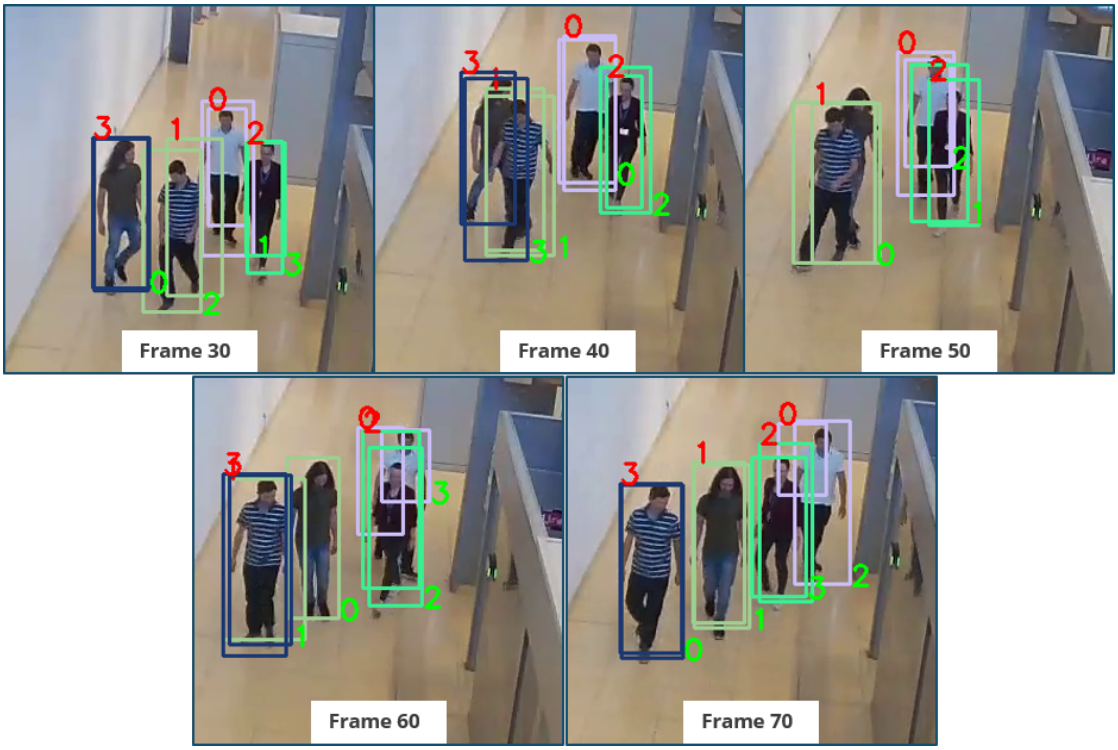


Figure 4.11: Sequence of frame where only geometrical features were used for the cost matrix evaluation

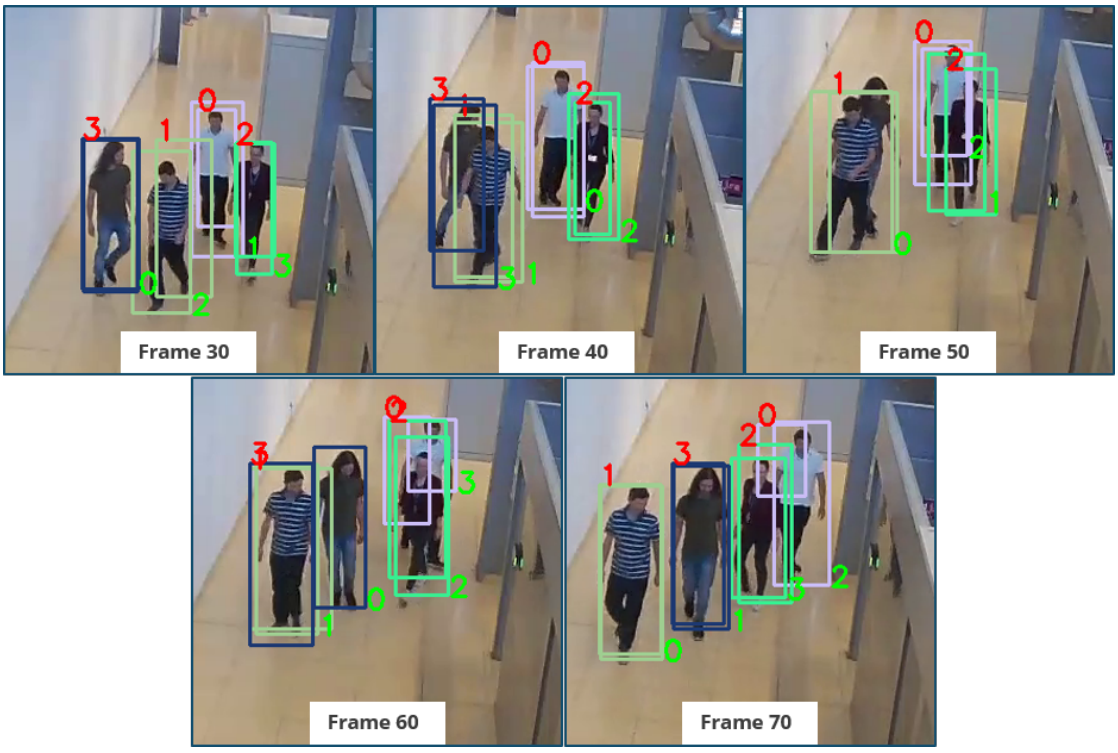


Figure 4.12: Sequence of frame where all features were used for the cost matrix evaluation

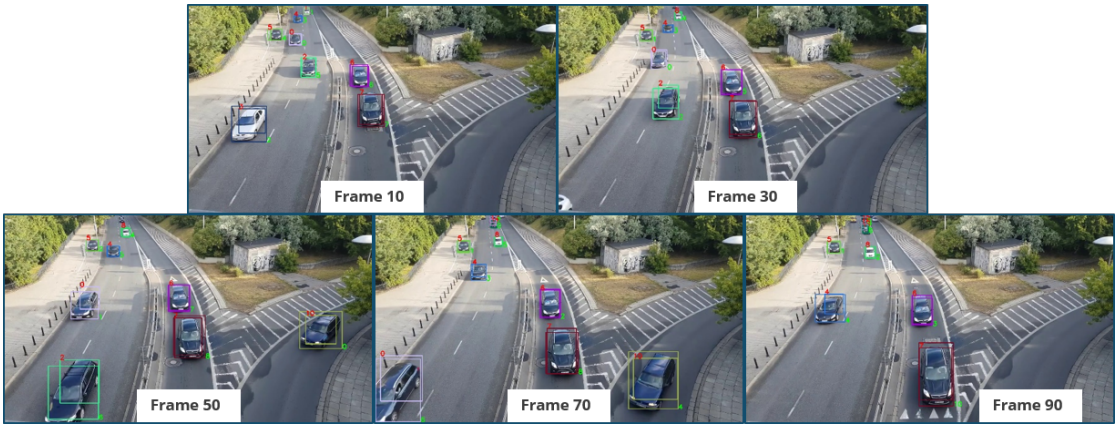


Figure 4.13: Sequence of frame where all features were used for the cost matrix evaluation

Chapter 5

Trajectory Prediction

Path prediction is a fundamental task to predict how pedestrians or vehicles are going to move in the environment by estimating the trajectory along which a target (e.g. pedestrian or vehicle) will move. Most common approach for solving the trajectory prediction problem extract various information form the video in order to obtain an environment estimation. In this case of study, since the algorithm speed is one of the most important requirements, the easiest and fastest algorithm has been implemented, with the aim of improving its performances with additional part that can be developed separately.

In the following section 5.1 is reported a brief overview on Vision-based Path Prediction state-of-the-art while the following section describes the used methods (*Polynomial Interpolation* in Section 5.1.2 and *Kalman filter* in Section 5.1.1). Lastly, Section 5.2 reports algorithms performances when tested on different kind of trajectories.

5.1 State of the art

By looking at the literature ([11]), it follows that knowledge about the surrounding environment, target moving direction or object pose are often required for path prediction solution. For this reason, most of the time path prediction method are developed on top of other computer vision tasks, such as object detection and identification, attribute recognition and semantic segmentation. Moreover, path prediction mostly rely on previous state knowledge. This means that object tracking is also required as background task. In other words, the prediction task localize and predicts the target position in future frame of a video, by means of past observations made until the present time, knowledge on the target motion and prior information on the surrounding environment.

Environment and target information are extracted in most of the proposed method, followed by path prediction task. Environment features can be achieved through semantic segmentation, or probabilistic approaches that implicitly represent probabilities of path as cost functions. For what concern target features, the orientation of the object the most common extracted information. Through these information, it is possible to reduce the prediction error.

Path prediction method can be divided in four main categories:

- Deep learning
- Inverse reinforcement learning
- Energy minimization
- Bayesian models

Deep learning methods solve the path prediction problem through deep learning frameworks such as CNN or LSTM. These methods take as input a series of coordinates which represent past state locations, and produce a series of target coordinates in the successive frames. In this case, feature extraction and path prediction stage are not explicitly separated but the Neural Network manage both at the same time. (These method strongly depend on how the network is trained and also require lots of computation efforts that can cause latency problem.)
Inverse reinforcement learning is a RL that learns a policy to decide actions to be taken by an agent based the current state of the environment. ...
Energy minimization is an offline approach that estimate the entire sequence of coordinate at the same time. This approach use a 2D grid and assign cost to movements from edges in the path, and then finds the combination of edges that gives the minimum cost energy. This method is also known as shortest path problem which can be solved by employing the Dijkstra method.
Bayesian models is an online solution that use Bayesian filters, such as Kalman Filters and particle filters, and infers the model to predict the path.

Being the Bayesian approach an online solution that does not require lots of computation (like CNN and LSTM), this solution will be considered for the use case scenario. Moreover, since the trajectory of an object can be seen on a 2D axes plane as a function, also polynomial interpolation has been analyzed as a possible solution. As reported in Section 3, since the tracking algorithm allow to collect for each object of interest its position in n previous frames, these data have be used to estimate the target path for m future time instant.
To better analyze results, an offline trajectory prediction has been developed, with the aim of integrate it in the framework for each single object. For this reason, has been implemented online methods and the old state data have been treated as if they where available only once at a time.
In the following Section 5.1.2 and Section 5.1.1 are explained how, respectively, the Linear Interpolation and the Kalman Filter have been implemented. Results and best performances are reported in Section 5.2

5.1.1 Kalman Filter

Kalman filter is an algorithm that works with dynamic systems i.e. systems that can be in different states and their behavior depends o which state they are. These states are

unknown, and this algorithm is used to estimate them and make predictions on what could be the next state. The Kalman filter works over a state-space model and on the observation of the real system.
The state-space model is a mathematical model build to approximate the real physic phenomenon and it can be more or less accurate depending on how has been built. The observation is a measure of what can be seen by externally looking at the system.
In this scenario, as linear dynamic system has been assumed the one which describes an object moving on a 2D plane at approximately constant velocity. This mathematical model is described into this system of equations of (5.1).

$$\begin{cases} z_t = A_t z_{t-1} + B_t u_t + \epsilon_t \\ y_t = C_t z_t + D_t u_t + \delta_t \\ \epsilon_t \quad N(0, Q_t) \\ \delta_t \quad N(0, R_t) \end{cases} \qquad A = \begin{bmatrix} 1 & 0 & \Delta & 0 \\ 0 & 1 & 0 & \Delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{5.1}$$

In system (5.1), matrix A contains the coefficients for the state evolution where Δ is the object speed. z is a 4 columns state vector with the x and y components of both the position and velocity. Matrix C contains the coefficients used to obtain the observed models position values. As can be seen, the object position is directly observed, while there is no observed value for the velocity. Moreover, since the given model has no input variables, $B = D = 0$. Finally, ϵ_t and δ_t are noise contributions that tell to the Kalaman filter how much it can trust, respectively, the mathematical model and the observation. Both are assumed to be affected by Gaussian distributed error.
Kalman filter algorithm aims to estimate z_t based on all past observed value $p(z_t|y_{1:t})$ in a recursive way and it is made of two main steps: **Prediction step** and **Update step**. In the **Prediction step** the system compute the state prediction $p(z_t|y_{1:t-1})$ which follow, based on initial system assumption, a normal distribution $N(z_t|\mu_{t|t-1}, \Sigma_{t|t-1})$ where $\mu_{t|t-1}$ and $\Sigma_{t|t-1}$ indicates, respectively, mean and variance distribution values at time t given all past states. State estimation is used to calculated the posterior predictive distribution $p(y_t|y_{1:t-1})$ and the new observed value prediction $\hat{y}_t = \mathbb{E}[y_t|y_{1:t-1}]$. Then, in the **Update step**, the algorithm updates the sate with the new observation at time t: $p(z_t|y_{1:t})$ that will define the μ_t and Σ_t. The algorithm does everything recursively,where new mean is the old one plus a correction factor called Kalman gain (K). In the following, steps are explained in detail.

Prediction:

1. $\mu_{t|t-1} = A_t \mu_{t-1}$
2. $\Sigma_{t|t-1} = A_t \Sigma_{t-1} A_t^\intercal$

Update:

1. $\hat{y}_t = C_t \mu_{t|t-1}$
2. $K_t = \Sigma_{t|t-1} C_t^\intercal (C_t \Sigma_{t|t-1} C_t^\intercal + R_t)^{-1}$
3. $r_t = y_t - \hat{y}_t$
4. $\mu_t = \mu_{t|t-1} + K_t r_t$
5. $\Sigma_t = (I - K_t C_t) \Sigma_{t|t-1}$

At time t, after predicting and updating μ and Σ, the algorithm iterates n times on the *Update* steps, by considering $y_t = \hat{y}_t$. In this way, the algorithm predict future object position by following the mathematical model as if their new predicted states are perfectly aligned with the observed values.
This choice comes from [12], that had shown how a simple Constant Velocity Model can achieve competitive performance in the pedestrian motion prediction task.

Finally, covariance matrices Q and R have been defined. Firstly, has been assumed that noise components are statistically independent. Thus, both matrices are diagonal with all zeros out of it. Moreover, variance of each component has been assumed to be the same. This means that each matrix has the same number on the diagonal, as shown below.

$$Q = \begin{bmatrix} \sigma_q & 0 & 0 & 0 \\ 0 & \sigma_q & 0 & 0 \\ 0 & 0 & \sigma_q & 0 \\ 0 & 0 & 0 & \sigma_q \end{bmatrix} \qquad R = \begin{bmatrix} \sigma_r & 0 \\ 0 & \sigma_r \end{bmatrix}$$

In order to define σ_q and σ_r, a ground truth trajectory has been extracted from a 4K traffic camera video [27]. Then, the Kalman filter algorithm has been run for 400 different value combination of σ_r and σ_q ranging from 0.1 to 2. it is worth noting that the ground truth has been evaluated by means of our framework while using KCF tracking algorithm and detection match every 10 frames. This choice has been done since KCF algorithm is the one with higher performances (as shown in Section 6.1). Moreover, detection match every 10 frame allow to obtain trajectories closer to the worst possible scenario. At each prediction step, the distance between the predicted points and the ground truth has been evaluated. Figure 5.1 shows how mean value and the standard deviation of collected errors change. Red points in Figure 5.1 indicate minimum value of both mean and standard deviation. The best configuration has been defined as the one with $\sigma_q = 0.1$ and $\sigma_r = 0.4$. This solution confirms also what said in [12]. Indeed, $\sigma_q < \sigma_r$ means that the filter can trust more the mathematical model than the observed values.

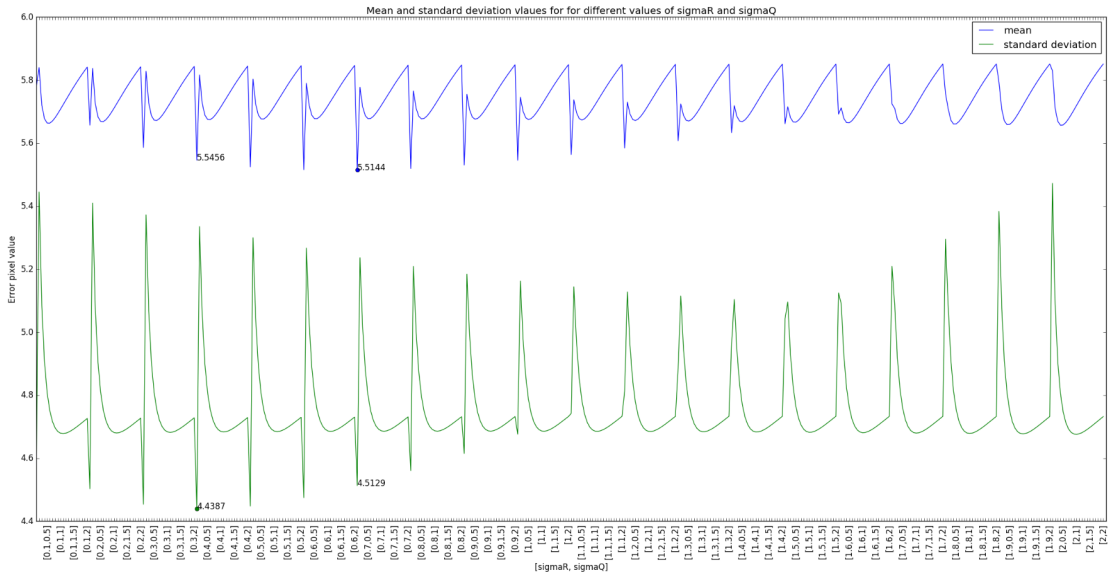


Figure 5.1: σ_r and σ_q evaluation

5.1.2 Polynomial Interpolation

Polynomial interpolation is the a mathematical technique that, given a set of points, allows to find out the the polynomial of lowest possible degree that passes through these points. So by interpolating a set of points, it is found out the set of coefficients that define the function which better approximate the points trend.
For the trajectory prediction problem, polynomial interpolation has been performed separately on x and y position points w.r.t. the time. Then, by looking at the values assumed by the function in the next m time instants, path prediction is performed. Two different method have been implemented: in the first one, the points used for the polynomial interpolation are all those available from the ground truth, while in the second one, only last 10 points were considered. Moreover, the polynomial degree has been changed from 1 to 3. In this case, all available ground truth have been used to test all possible parameters configurations. Figure 5.2 shows obtained values of mean and standard deviation of errors evaluated as in Section 5.1.1.

Also in this case, the configuration that gives the smaller error has been chosen. Indeed, polynomial interpolation of degree 2 while considering only last 10 available states have been implemented and compared to the Kalman filter performances.

5.2 Algorithm evaluation

Kalman filter path prediction with parameters of Section 5.1.1 has been run on 12 vehicles ground truth taken from [27] and on 4 human ground truth trajectories. An example of path predictions are reported in Figure 5.3 and Figure5.4.

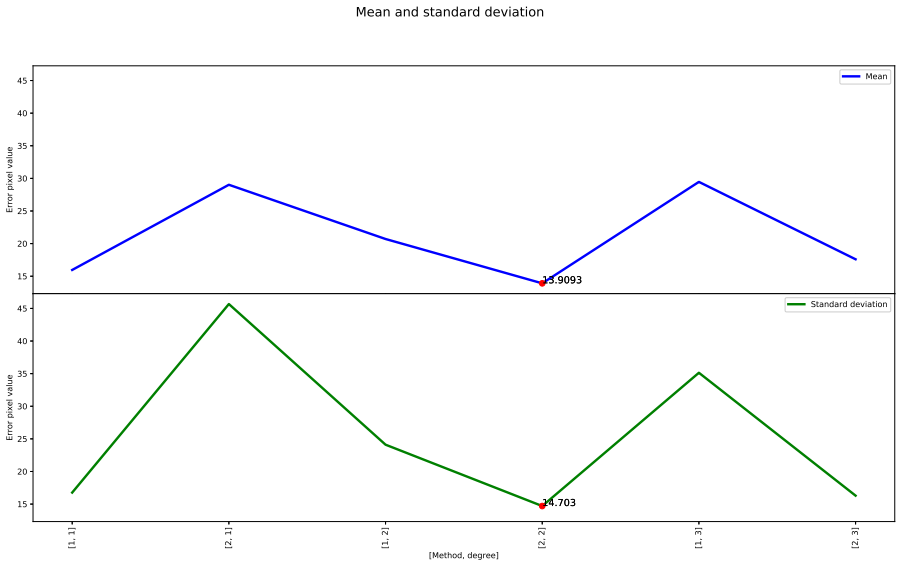


Figure 5.2: Polynomial degree and method evaluation

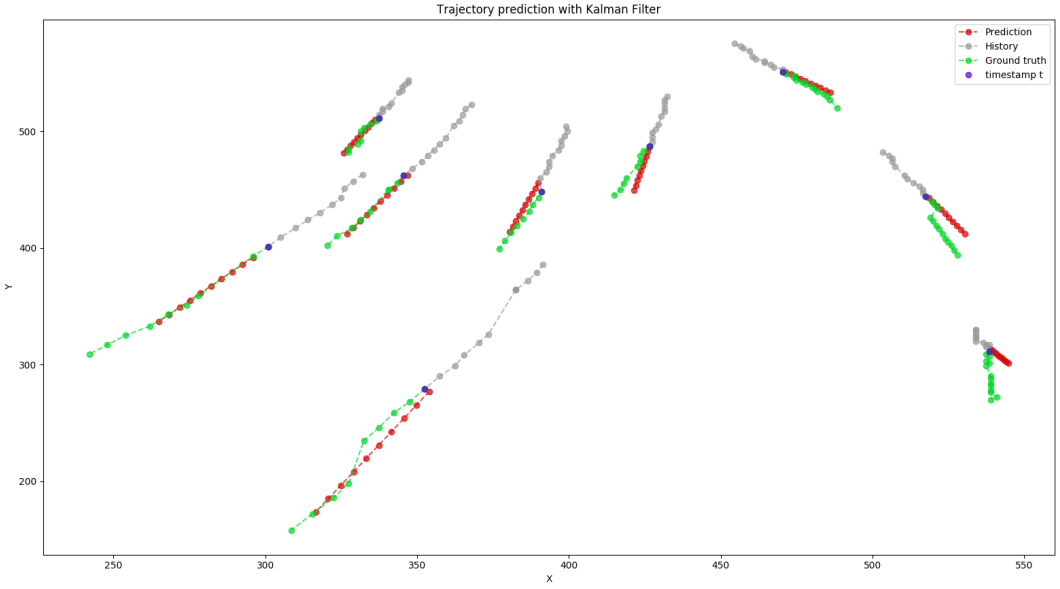


Figure 5.3: Human trajectory prediction with Kalman filter

From Figure 5.5 and Figure 5.6 error trend for each tracked object are reported. Indeed, Figure 5.5a and Figure5.5b show how errors are distributed when applying Kalman filter ($\sigma_q = 0.1$ and $\sigma_r = 0.4$) prediction over trajectories collected with detection match every 10 frames respectively for vehicles and pedestrians. The same has been done in

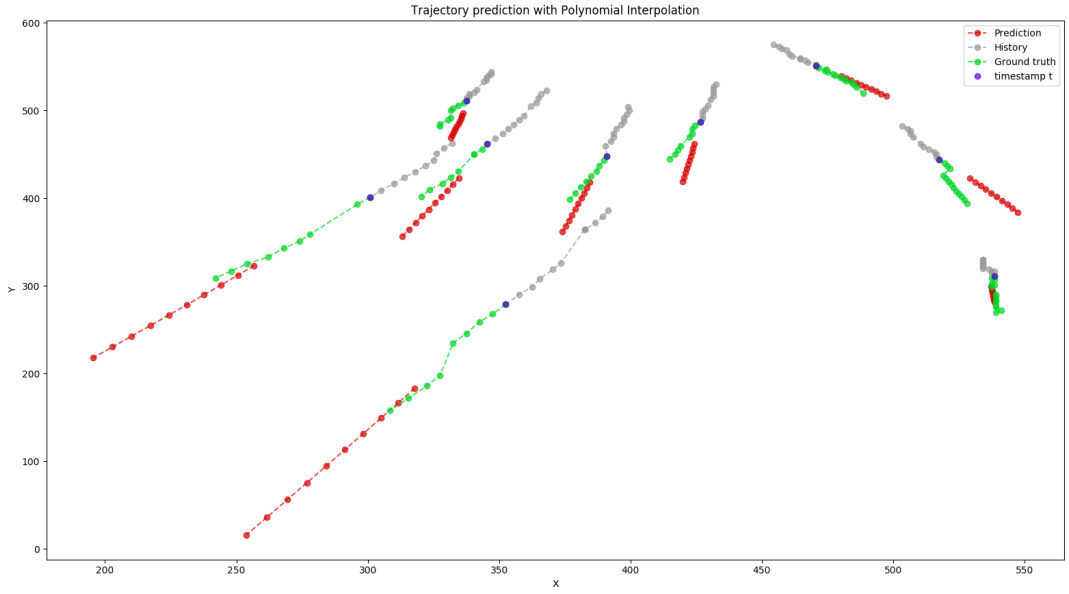
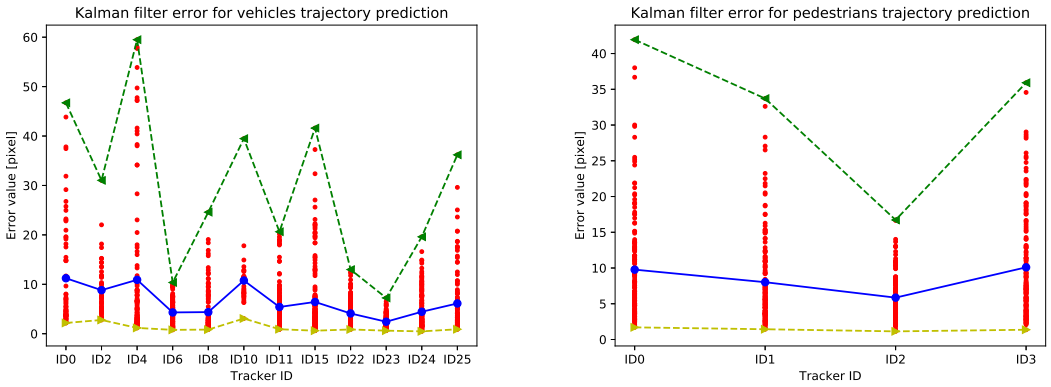


Figure 5.4: Vehicle trajectory prediction with Kalman filter



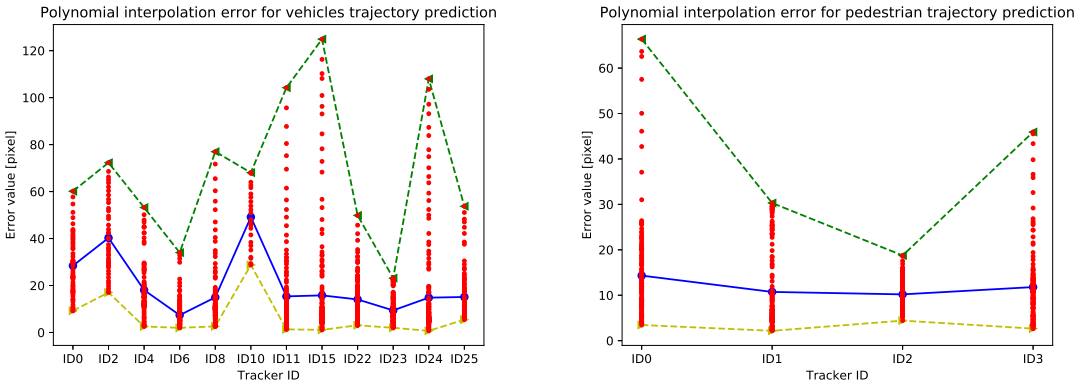
(a) Vehicles error evaluation

(b) Pedestrian error evaluation

Figure 5.5: Kalman Filter error evaluation over different tracker trajectories

Figure 5.6a and Figure 5.6b but when using the polynomial interpolation with parameters as evaluated in Section 5.1.2. As expected, errors for polynomial interpolation are bigger than the ones obtained when applying Kalman filter.

Finally, in order to have a more complete view of how prediction errors depends on the trajectory goodness, Figure 5.7 has been reported. Indeed, it shows how Kalman filter prediction error change for the vehicles reported in Figure 5.8 when using trajectory data evaluated with different frame rate between detection and tracking. Against the common sence, when applying detection at each frame trajectory prediction has not



(a) Vehicles error evaluation

(b) Pedestrian error evaluation

Figure 5.6: Polynomial interpolation error evaluation over different tracker trajectories

the best performances. This is a consequence of continuously oscillation of the target ROI. Indeed, if the object is moving relatively slow, the detection ROI can oscillate a lot around the same point because of its continuously resize and the resulting trajectory may be perturbed. At the same time, if detection match is performed every 10 frame, tracker drift may be consistent and so it can cause strong trajectory drift as well.

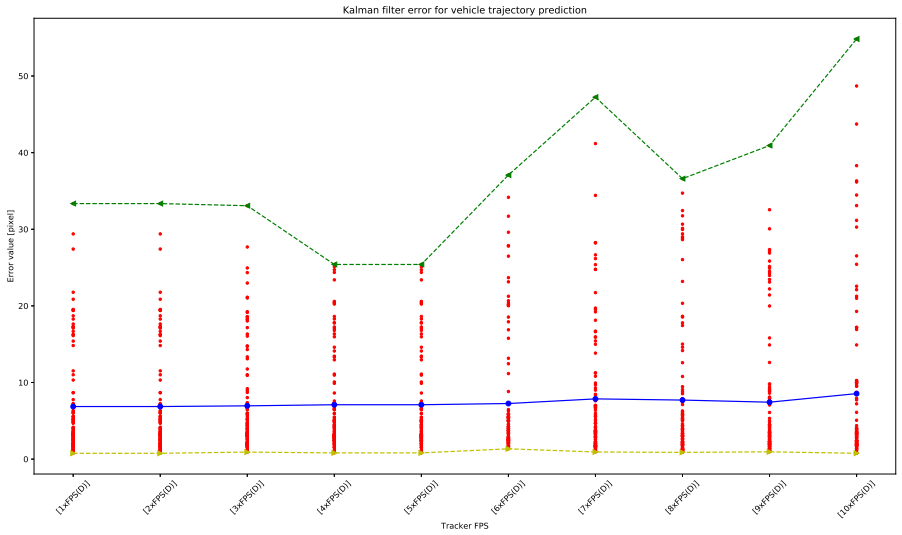


Figure 5.7: Kalman Filter error for different frame rate of detection stage

As can be seen, Kalman filter provide lower value for the mean error and also smaller standard deviation. Indeed, its performances are better that the ones achieved by the polynomial interpolation.

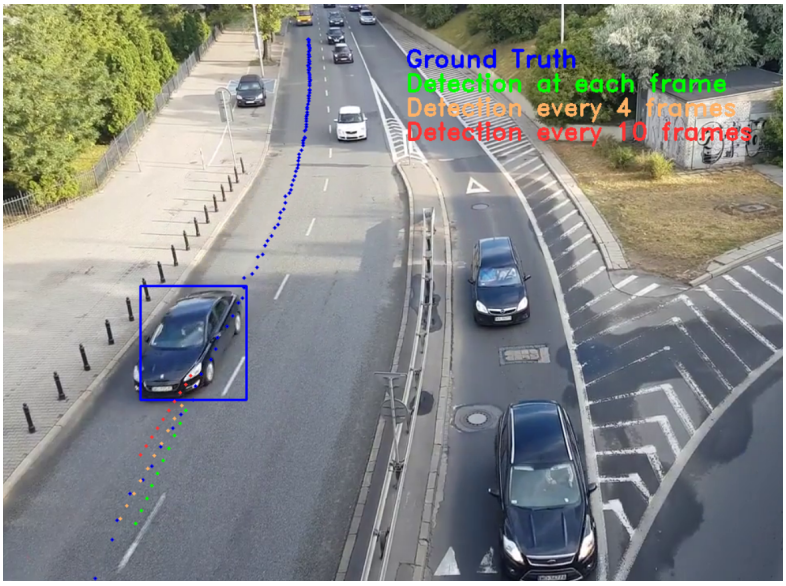


Figure 5.8: Trajectories prediction based on tracking detection FPS configuration

Chapter 6

Tracking algorithms benchmark and framework evaluation

Since the framework has been developed also to benchmark different tracking algorithms, a set of Key Performance Indicators (KPIs) has been defined. As the name suggest, these parameters are used to give an unified unit of measure for each algorithm parameter that is relevant for the use case study. In this way, these different performances can be compared in different scenarios and it is possible to make a comparison among all different tracking algorithms. The used KPI parameters have been decided based on experimental analysis and on results about single-target tracking performance evaluation method clustering performed by [8]. KPIs value are then normalized and integrated in a Weight Sum Model [WSM]. This model is multi-criteria decision making method that allow to classify the different tracking algorithms by means of a weighted sum of each KPI.
Obviously, for the algorithm evaluation, a ground truth is required. By running the framework in an easy scenario (three targets walking next to each other in straight line) with detection matching at each frame, the ground truth value of 300 frames has been extrapolated. Figure 6.1 represents the ground truth trajectories made of consecutive bottom center ROI positions. The same approach has been used to generate vehicles video ground truth.
Moreover, these KPIs have been defined and evaluated for four different environment perturbation (as explained in Section 6.1). Finally, the WSM output for each tracker and comparison between them are reported in Section 6.2.

6.1 KPIs definition and evaluation

Based on [8], six KPIs have been defined:

1. **[AFPS]** *Algorithm Frame Per Second: measures how many frames the algorithm process in one second.*
   This parameter is one of the most important ones. Indeed, if the algorithm is no

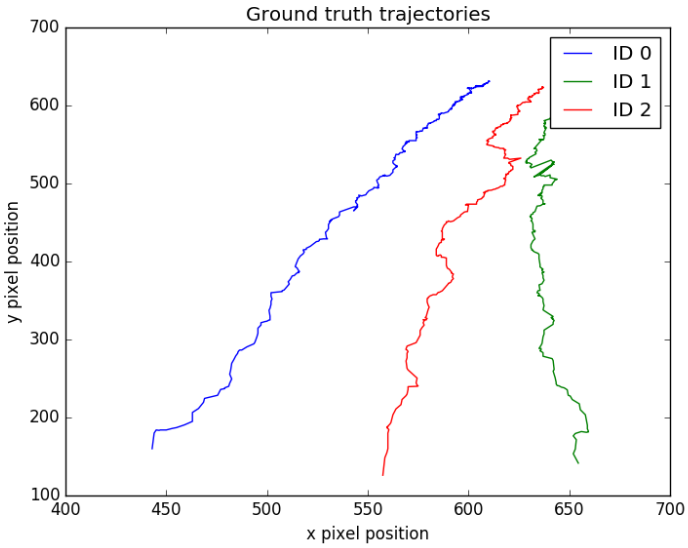


Figure 6.1: Ground truth trajectories

able to keep on with the video frame rate, there could be bottleneck and other latency issues. This kind of issue has to be avoided since the real time requirement is the most important one.

2. [**CE**] *Center Error: measures the difference between tracker ROI center and the ground truth one.*
   This measure should give an idea on how far from the real position the tracker is pushing the target ROI and, make a comparison between trackers drift. These parameter has to be normalized w.r.t. a known and common measure in order to have the same range of value for each single tracker. For this reason, the distance between tracker ROI center and detection one are divided by the 10^{th} part of the frame diagonal.

3. [**RO**] *Region overlap: measures the overlap between the predicted tracker ROI and the ground truth one.*
   The overlap is evaluated as reported in (6.1) and it is based on Figure 6.2, which gives an illustration of the overlap of ground-truth region with the predicted region.

$$\frac{R_G \cap R_T}{R_G \cup R_T} \tag{6.1}$$

This measure has the same meaning of the *Center Error* but it takes into account smaller errors. Indeed, when the tracker drift is strong, the overlap goes to zero.

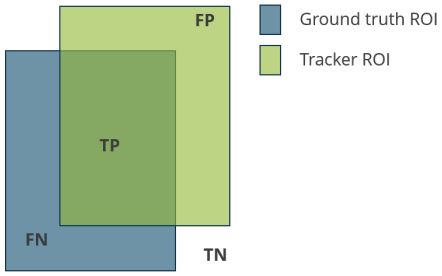


Figure 6.2: Ground truth and tracker overlap

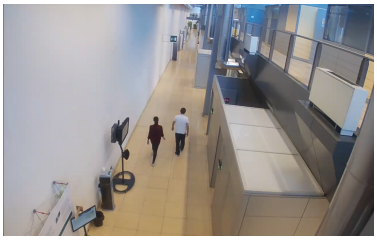
On the other hand, if the distance between tracker and detection is not that big, the *Center Error* would be very small and may be not appreciable for a comparison, while the overlap will be greater than zero.

4. [**PCFT**] *Percentage of Correctly Tracked Frames: measures the overall true-positive prediction.*
   By choosing an overlap threshold $\tau = 0.5$ to identify a correct prediction [9], it is evaluated as the average true positive. This measure let understand how many times the tracker drift is over the threshold that allow to identify that prediction as a good one. Indeed, if that threshold would be used to define when the tracker parameter has to be update, this KPI let understand how many time a certain tracker require this process. Since the tracker re-initialization takes more time than the only update, smaller is this percentage, lower will be the related performance of the tracker.

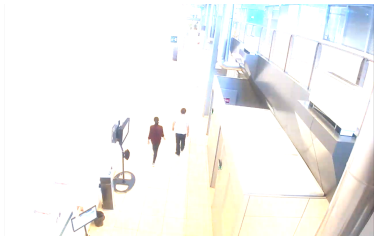
5. [**TL**] *Tracking length: measures the number of successfully tracked frames from trackers initialization to its (first) failure.*
   By means of the previous threshold, first tracker failure is identified. Longer is the algorithm first correctly tracked series of frame, better would be the related performance. Indeed, at the very beginning, there are no past state to relay on if case of occlusion or tracker drift. This means that if a tracker is able to do not lost its target at the beginning, there could be more chances to recover the object in case of occlusion. Of course this measure strongly depends on the used video. In this case, as said before, the easiest use case scenario has been used. Thus, if an algorithm loose its object after a few frame in this conditions, it means that it will not be a good choice for the MOT solution.

6. [**FR**] *Failure rate: measures how many times the tracker fails.*
   Also failure are identified by looking at the overlapped area threshold. As the *Percentage of Correctly Tracked Frames*, also this parameter should tell how many time the tracker has to be re-initialized. Indeed, big failure rate leads to lower performances and so it has to penalize the final algorithm performance score.

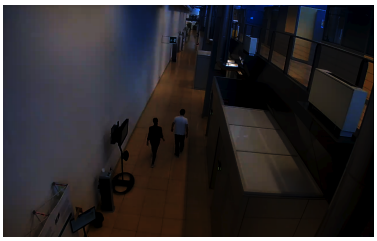
These six KPIs have been evaluated in four different environment conditions:



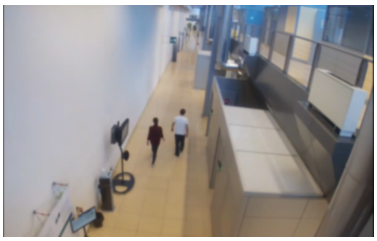
(a) Normal condition



(b) brighter scenario



(c) darker scenario



(d) blurred scenario

Figure 6.3: Example case of matching images

- Normal condition scenario (Figure 6.3a): no perturbations ha ve been applied.

- Brighter scenario (Figure 6.3b)
  For this image perturbation each RGB pixel vale has been increased of constant value. For each value saturation at 255 has been used. Indeed, it there where already light pixels (which values where close to 255), the pixel have been set to the maximum value.

- darker scenario (Figure 6.3c)
  Here gamma correction has been applied with $\gamma = 5$. Gamma correction is a non linear transformation used for bit usage optimization when encoding images. It is based on the fact that human perceive light and color in a non-linear manner.

- blurred scenario (Figure 6.3d)
  This last scenario has been obtained by means of a image-blurring filter that uses a Gaussian function, widely used to reduce image noise and detail. The filter is applied on a sliding windows across the whole image. The kernel size (and the windows as well) has been defined as a square of 23 pixel by side. Moreover, both mean and standard deviation of the Gaussian filter has been defined as $\sigma = 0.3 * ((ksize - 1) * 0.5 - 1) + 0.8$ as suggested by the *OpenCV* guide [26].

Finally, all the KPIs in each different environment variation have been evaluated in two different cases: when the framework matching algorithm was running and when only the *OpenCV* tracking algorithms were working. The framework matches with detection data has been done every 10 frames.

6.2 Weighted Sum Model for tracking algorithm evaluation

In order to have an overview of each tracker performances, a Weighted Sum Model [WSM] for tracking algorithm evaluation has been developed. Indeed, WSM is a multi-criteria decision analysis method that allow to compare different alternatives in terms of a certain number of decision criteria. In this case, the WSM has been applied to all trackers KPI and the final score has been used to understand which one would be the best tracking algorithm for our use case scenario.
Firstly, it is important to notice that this method is applicable only when all the data are expressed in exactly the same unit. Thus, each KPI has been normalized w.r.t. the total number of frames. Only the frame rate has been normalized w.r.t. the video frame rate. Indeed, if the algorithm is able to process frame with a higher throughput, this KPI value would be grater than 1.
Then, based on the importance of the KPI measure, a set of weight has been defined. Indeed,more a KPI is important in the algorithm evaluation, higher would be its weight value.
The normalized values are reported in (6.3). The final score for each single tracker is reported in (6.2), where s indicates the different scenarios and m the total number of KPI values.

$$S = \frac{1}{4}\sum_{s=0}^{3}\sum_{j=0}^{m} w_j KPI_j \tag{6.2}$$

$$\begin{aligned}
FPS &= \frac{AFPS}{Video_{FPS}} \\
AvgCE &= \neg\left(\frac{\sum frameCE}{total\ number\ of\ frames}\right) \quad where \quad frameCE = \frac{CE}{\frac{\text{Frame diagonal length}}{10}} \\
AvgRO &= \frac{\sum RO}{total\ number\ of\ frames} \\
AvgTL &= \frac{\sum TL}{total\ number\ of\ frames} \\
AvgFR &= \neg\left(\frac{\sum FR}{total\ number\ of\ frames}\right)
\end{aligned} \tag{6.3}$$

At each single normalized value has been associated a weight as reported in Table 6.1.

KPI	FPS	AvgCE	AvgRO	PCTF	AvgTL	AvgFR
Weight	3	1	2	1	3	2

Table 6.1: KPI weights

As said before, weights have been assigned based on the KPI value importance for the overall algorithm performances. Indeed, according with [8], *AvgCE* has a small weight since this measure does not take into account the targets size. On the other hand, *AvgTL* has more relevance since it indicates the number of consequently and correctly tracked frames from the tracker begin. This is important since, at the very beginning, there are no information about past states and, if the trajectory prediction was used to guide the tracker, there would be no enough past state information for a reliable prediction. As can be seen from Table 6.1, the algorithm speed is an important parameter too.

Tracker type algorithm	Average score No reinit	Average score 10 frame reinit
BOOSTING	6.33	9.48
MIL	5.43	9.08
KCF	9.78	13.99
TLD	3.88	4.93
MEDIANFLOW	13.31	14.86
CSRT	7.10	10.79

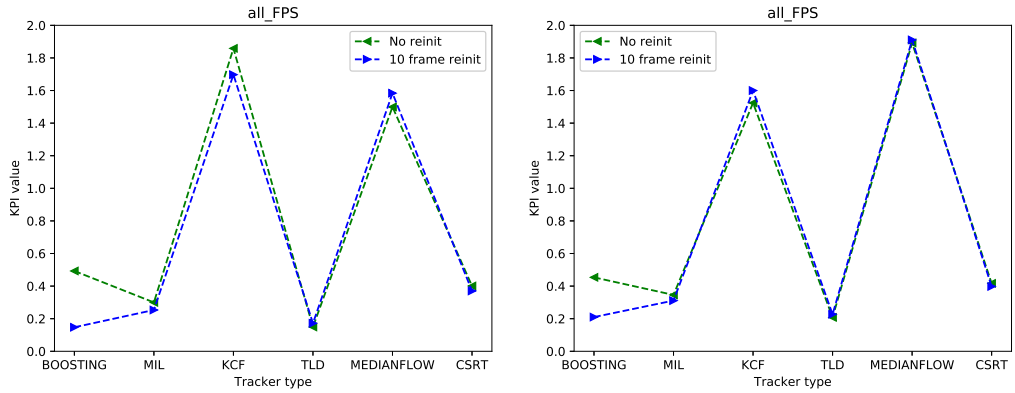
Table 6.2: WSM results for vehicles tracking

Tracker type algorithm	Average score No reinit	Average score 10 frame reinit
BOOSTING	6.01	10.95
MIL	5.81	11.53
KCF	7.92	15.47
TLD	2.53	4.32
MEDIANFLOW	7.46	14.63
CSRT	10.46	10.57

Table 6.3: WSM results for pedestrian tracking

From Table 6.3 and Table 6.2 it is easy to seen how the framework match and ROIs alignment increases the performances of each single tracker. To better understand how each score represent the algorithm performances,it has to be considered that, if the algorithm frame rate and the video one are exactly the same ($FPS = 1$), maximum score would be 12. Indeed, all that trackers that have a higher score have for sure a throughput greater than 25 fps. From both tables, red highlight rows refer to those trackers that have seen the bigger improvement in terms of performances thanks to the framework job. On the other hand, blue rows highlight those algorithms who did not reach good performances both with and without the framework assistance.

Starting from the worst tracker, it is worth noticing that *TLD* algorithm si the one that mostly suffer of ROI drift. Indeed, in the detection stage (as explained in Section

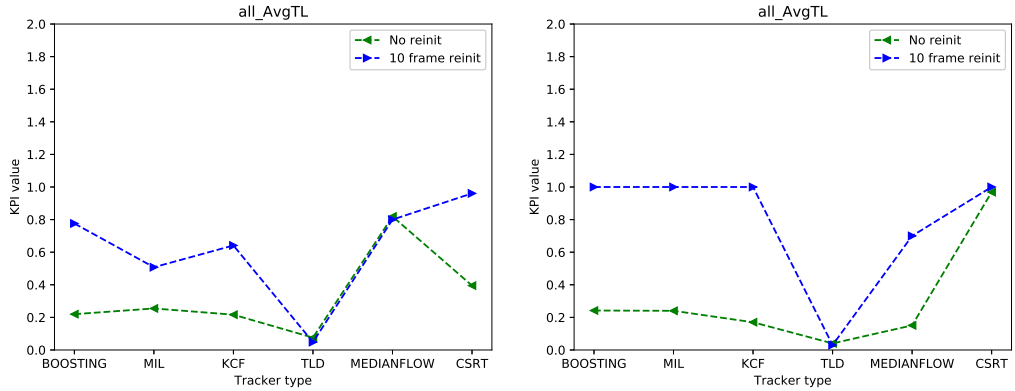


(a) FPS KPI when using vehicles video (b) FPS KPI when using pedestrian video

Figure 6.4: FPS KPI trend for all tracking algorithms

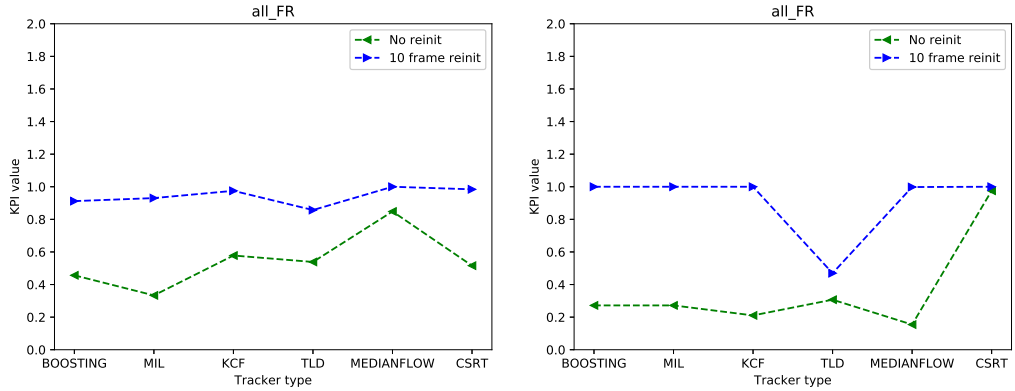
2.2.4), the algorithm scans the whole image in order to find out where its target is. This algorithm has been developed mostly to handle tracking scenario where there is only one object which stands out in the scene. Instead, in our scenario, multiple similar objects are close to each other. Indeed, the algorithm continuously jumps from one pedestrian to another making confusion with objects IDs. The same happen when it is applied to vehicles video scenario. Thus, this algorithm cannot be used.
On the other hand, *KCF* tracker is the one that makes more benefits from the framework support. Indeed, since this algorithm use measure *HOG* features every time it is updated, by placing the ROI in the correct position every 10 frame helps the algorithm in avoiding consistent ROI drift. Moreover, this where already performing good even without the framework supervision. Indeed, by looking at Figure 6.4a and Figure 6.4b, it is possible to notice how the *KCF* FPS KPI stands out even when it was running alone. for the sake of completeness, also other KPI values plots have reported in Appendix A.
Always looking at Figure 6.4a and Figure 6.4b, it is easy to catch the FPS peaks also for *MEDIANFLOW* algorithm both with and without the framework alignment. Indeed, also this tracker has final score higher than the maximum, being its throughput around 30 FPS. On the other hand, when running alone on the pedestrian video it has a very low score if compared with the same score but when evaluated on the vehicles video (orange cells). This is a direct consequence of how this algorithm estimate errors. Indeed, as explained in Section 2.2.5, the algorithm uses *Forward-Backward* key point tracking, and it they are not connected across multiple frames, they are considered as false negative. When tracking cars, the object shape and appearance do no change so much and, so, the key point to track do not break their connection across frames. Indeed, because of the not rigid human shape, this algorithm cannot have good performances when following pedestrian without supervised re-initialization.
To better underline the framework contribution, also the KPI that most benefit of it has been analyzed. Thus, both *AvgTL* and *AvgFR* have been reported, respectively, in

Figure 6.5 and Figure 6.6.



(a) KPI when using vehicles video (b) KPI when using pedestrian video

Figure 6.5: *all_AvgTL* KPI trend for all tracking algorithms



(a) KPI when using vehicles video (b) KPI when using pedestrian video

Figure 6.6: *all_FR* KPI trend for all tracking algorithms

These two KPIs are the ones that had the most improvement but they are also the ones with higher weights. Indeed, as reported in Table 6.1, *AvgTL*'s wight is 3 and the other one is 2. As can be from Figure 6.5, the number of successfully tracked frames from trackers initialization to its (first) failure has drastically increased especially when working with pedestrians. After the framework introduction, half of the algorithm have been able to track all their objects from the beginning until the end of the video without loosing them at all. As consequence, also the failure rate *allFR* of those algorithm it is equal to one. This means that they never had an overlap with the ground truth smaller than the above defined threshold ($\tau = 0.5$). It may be misleading to see a failure rate equal to one as indication of good performances. It must keep in mind that, while

defining normalization criterion, the final KPI of this value has been defined as the one's complement of the real one because of the way how WSM works. Indeed, for these three algorithm, the failure rate is equal to zero. Also in the vehicle scenario the failure rate has improved significantly. This reduction of tracker drift is a direct consequence of accurate algorithm re-initialization tanks to the developed framework.
Finally, also *CSRT* algorithm deserve a few observations. Indeed, this tracker uses a *regularized CFT* (refer to Section 2.2.6 and Section 2.1 for more details). This kind of tracker use a cosine windows to smooth the filter wights from the center toward the ROI edges. Indeed, when applied to objects that do not move too fast, this algorithm can give higher performances since the ROI centers would not be that far from one frame to another. This assumption is confirmed by looking at green cells in Table 6.3 and Table 6.2 that highlight algorithm better performances when used on pedestrian instead of vehicles.
Lastly, to validate the assumption of average the final score over 4 different scenarios (as expressed in (6.2)), KPI values of *KCF* tracker for each single scenario have been reported in Table 6.4. As can be seen, value does not change significantly; thus, the average of different scenarios is a good solution.

Scenario	*Reinit*	FPS	AvgCE	AvgRO	PCTF	AvgTL	AvgFR
Figure 6.3a	No	1.85	0.96	0.26	0.22	0.21	0.57
	10 frames	1.69	0.99	0.78	0.96	0.64	0.97
Figure 6.3b	No	1.87	0.96	0.19	0.16	0.15	0.63
	10 frames	1.60	0.99	0.73	0.91	0.62	0.0.98
Figure 6.3c	No	1.87	0.96	0.26	0.22	0.21	0.58
	10 frames	1.63	0.99	0.71	0.86	0.55	0.92
Figure 6.3d	No	1.96	0.93	0.19	0.17	0.17	0.52
	10 frames	1.72	0.99	0.77	0.95	0.51	0.96

Table 6.4: KPI values for KCF tracking algorithm and video with vehicles.

Chapter 7

Conclusion

Summarizing, with this thesis has been shown how the Multiple Object Tracking problem can be decomposed in multiple Single Object Tracking problems. Moreover, in order to do not have a fully dependency on the detection algorithm used, these data can be used even every 10 frame only to match and refine the tracking of each single object. To achieve this goal, a *C++ multi-threading framework* has been developed.
The developed framework allow to have full control on each single object tracked. Indeed, since object ID and thread ID are kept linked, it is possible to act over each single object separately. Furthermore, the main thread performs supervision and coordination for all trackers. Indeed, it is able to understand new incoming, outgoing and occluded objects and acts in different ways in order to handle each single case.
On the single object tracking side, each thread schedules the tracking algorithm steps and also provide notification about its parameters to the main thread. Moreover, each thread performs all feature extraction and comparison with the detection ROIs in order to let the main thread perform all matching between detection and tracker ROI. In this way, thanks to this alignment step, each thread can provide new "safe" data to the tracker algorithm in order to refine and update their prediction parameters. Figure 7.1 and Figure 7.2 show MOT obtained results while running, respectively, *MEDIANOFLOW* algorithm (vehicles frame sequence) and the *KCF* algorithm (pedestrians frame sequence), both with detection matching every 10 frames.

Finally, an offline trajectory prediction algorithm has shown how an easy and fast path prediction can be integrated with the MOT framework by giving sufficiently good performances. This results have been obtained by applying a linear Kalman filter with constant velocity model run over past state collected by the MOT framework.
In addition to this, a set of Key Performance Indicators have been defined in order to benchmark *C++ OpenCV* library Single Object Tracking algorithms by means of a Weighted Sum Model. From this analysis, has been found out which of these algorithms could better fit our use case scenario. Indeed, *KCF* and *MEDIANFLOW* have been spot as best single object tracking algorithms to solve the Multiple Object Tracking problem when vehicles and pedestrians are the objects of interest.

In conclusion, has been proved that the developed framework is useful for both Single

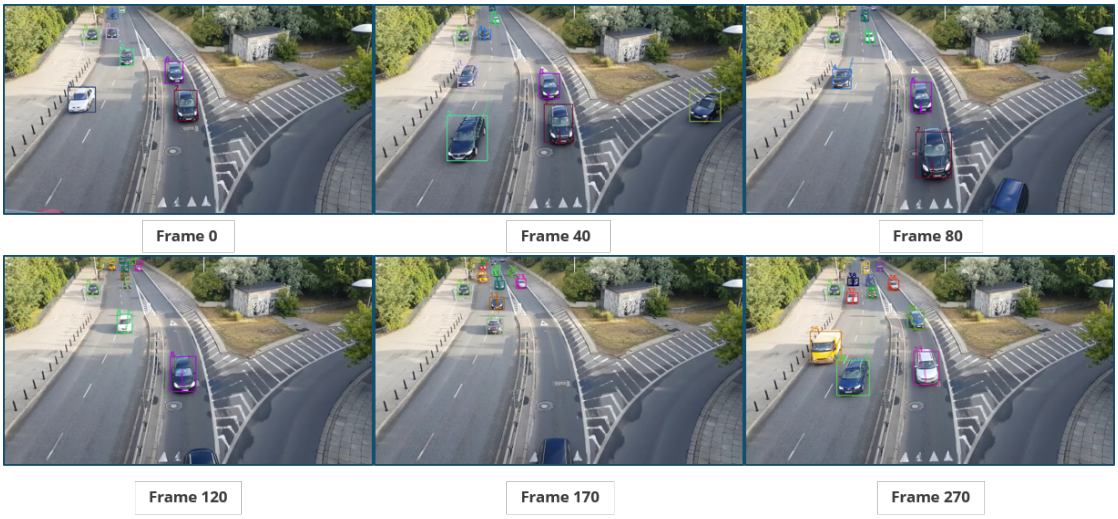


Figure 7.1: MOT framework working on video with vehicles

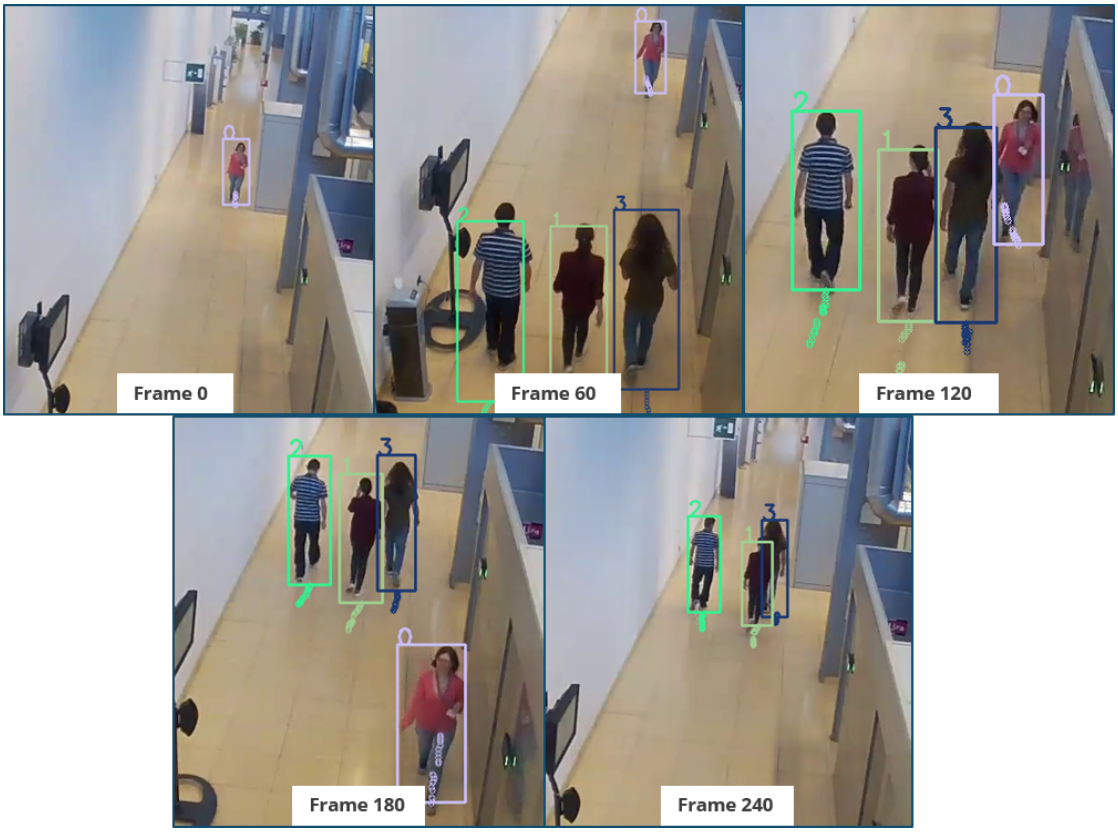


Figure 7.2: MOT framework working on video with pedestrians

Object Tracking algorithm bench-marking and as Multiple Object Tracking solution.
It is scalable since is able to track as many object as present. It keeps updated with the number of present objects and is able to understand whether there are new object to track in the scene. In the same way, it understands also when to stop a tracker thread since the corresponding object has gone. It is easy to improve or change parts of it since each part works independently. Indeed, different feature extraction or ad hoc feature based on which object to track can be added. The algorithm allow also to introduce a completely new SOT algorithm with the only constraint that it has to be structured in the two main steps *Initialization* and *Update*.
From what concern the path prediction, has been demonstrate that good performances can be achieved when using the easiest Kalman filter implementation. This because both humans and vehicles trajectories are almost linear and so the constant speed model fit quite well this scenario. Figure 7.3 shows the offline trajectory prediction applied to a vehicle still running the *KCF* algorithm and detection match every 10 frames. In the frame sequence, the blue path represent the ground truth while the green one is the path prediction at that frame.

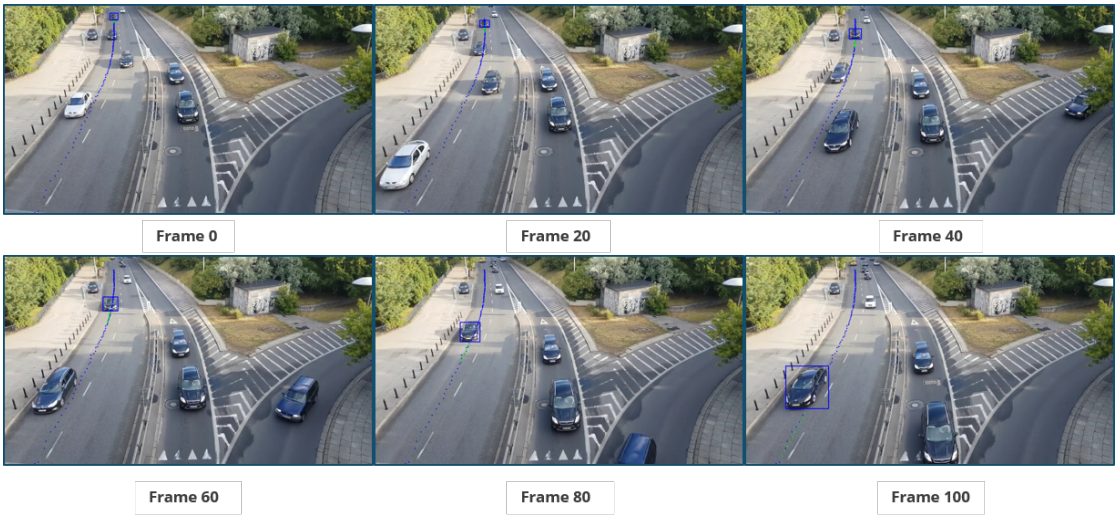


Figure 7.3: Offline trajectory prediction on a vehicle path obtained through *linear Kalman filter*

On the other hand, the framework speed goes down linearly with the number of object to track. Indeed, may be not the best choice as MOT solution when applied to a highly crowded scenario.

7.1 Future improvements

As said before, the framework is able to integrate any kind of improvement: from the tracking algorithm used, to the extracted feature and to the trajectory prediction integration.
For example, as first improvement, the Kalman filter can be integrated into the framework in order to have online and real time path prediction. The path prediction can be used to guide the tracker in case of occlusion and help the tracker in the matching phase when the detection ROI will be again available. Moreover, the patch prediction can be used also to have a further feedback to understand when an object is going out of the scene. Indeed, if the ROI is moving closer to the frame edges and the path prediction point out of it, it may indicate that its correspond tracker can be stopped. Finally, the path prediction step can be reinforced by adding prior information about the environment and the target surrounding objects. For example, information extracted trough periodic segmentation may be useful to give prior information about movements in a certain area.
On the tracking algorithm part, may be useful to use a new SOT algorithm that incorporate features and key points of the best performing trackers. For example, based on what reported in Table 6.2 and Table 6.3, since *KCF* and *MEDIANFLOW* algorithms are the ones that gives higher performances, a new online tracking algorithm that implements both a kernelized correlation filter and a forward-backward error classification could be useful.
Finally, information about object distance may be added in order to improve the system performances. Indeed, by merging cameras data with them coming from a *Lidar*, more information about the object moving direction can be extracted.

Bibliography

[1] FESTAG, Andreas. Cooperative intelligent transport systems standards in Europe. IEEE communications magazine, 2014, 52.12: 166-172.

[2] Fiaz, Mustansar, Arif Mahmood, and Soon Ki Jung. "*Tracking Noisy Targets: A Review of Recent Object Tracking Approaches.*" arXiv preprint arXiv:1802.03098 (2018).

[3] Chen, Zhe, Zhibin Hong, and Dacheng Tao. "An experimental survey on correlation filter-based tracking." arXiv preprint arXiv:1509.05520 (2015).

[4] LUO, Wenhan, et al. Multiple object tracking: A literature review. arXiv preprint arXiv:1409.7618, 2014.

[5] XIANG, Yu; ALAHI, Alexandre; SAVARESE, Silvio. Learning to track: Online multi-object tracking by decision making. In: Proceedings of the IEEE international conference on computer vision. 2015. p. 4705-4713.

[6] JAYANTHI, N.; INDU, S. Comparison of image matching techniques. International Journal of Latest Trends in Engineering and Technology, 2016, 7.3: 396-401.

[7] MARN-REYES, Pedro A.; LORENZO-NAVARRO, Javier; CASTRILLN-SANTANA, Modesto. Comparative study of histogram distance measures for re-identification. arXiv preprint arXiv:1611.08134, 2016.

[8] ehovin, Luka, Ale Leonardis, and Matej Kristan. "Visual object tracking performance measures revisited." IEEE Transactions on Image Processing 25.3 (2016): 1261-1274.

[9] EVERINGHAM, Mark, et al. The pascal visual object classes (voc) challenge. International journal of computer vision, 2010, 88.2: 303-338.

[10] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 983990. IEEE, 2009.

[11] HIRAKAWA, Tsubasa, et al. Survey on vision-based path prediction. In: International Conference on Distributed, Ambient, and Pervasive Interactions. Springer, Cham, 2018. p. 48-64.

[12] SCHLLER, Christoph, et al. The Simpler the Better: Constant Velocity for Pedestrian Motion Prediction. arXiv preprint arXiv:1903.07933, 2019.

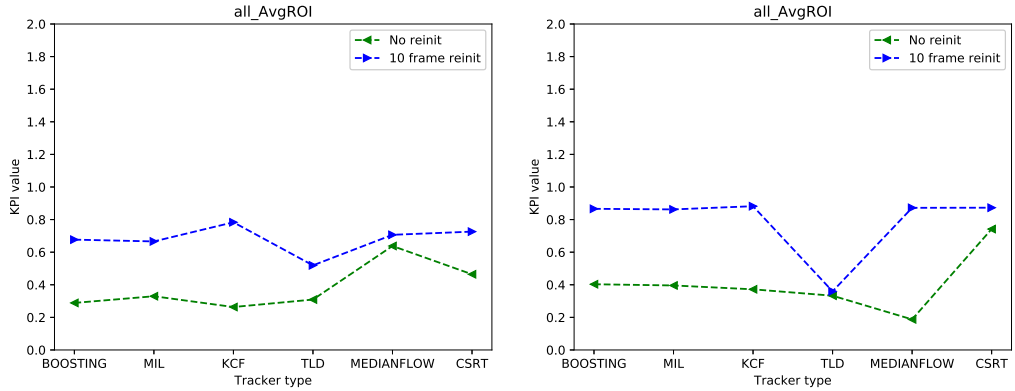
[13] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In proceedings of the European Conference on Computer Vision, 2012.

[14] KALAL, Zdenek; MIKOLAJCZYK, Krystian; MATAS, Jiri. Tracking-learning-detection. IEEE transactions on pattern analysis and machine intelligence, 2011,

34.7: 1409-1422.
[15] KALAL, Zdenek; MIKOLAJCZYK, Krystian; MATAS, Jiri. Forward-backward error: Automatic detection of tracking failures. In: 2010 20th International Conference on Pattern Recognition. IEEE, 2010. p. 2756-2759.
[16] LUKEZIC, Alan, et al. Discriminative correlation filter with channel and spatial reliability. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. p. 6309-6318.
[17] ZHENG, Liang, et al. Person re-identification meets image search. arXiv preprint arXiv:1502.02171, 2015.
[18] https://www.synopsys.com/automotive/autonomous-driving-levels.html
[19] https://en.wikipedia.org/wiki/Polynomial_interpolation
[20] https://docs.opencv.org/master/dd/d0d/tutorial_py_2d_histogram.html
[21] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html
[22] https://opencv.org/about/
[23] https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/
[24] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html
[25] https://towardsdatascience.com/computer-vision-for-tracking-8220759eee85
[26] https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur#Mat%20getGaussianKernel(int%20ksize,%20double%20sigma,%20int%20ktype)
[27] https://www.youtube.com/watch?v=MNn9qKG2UFI

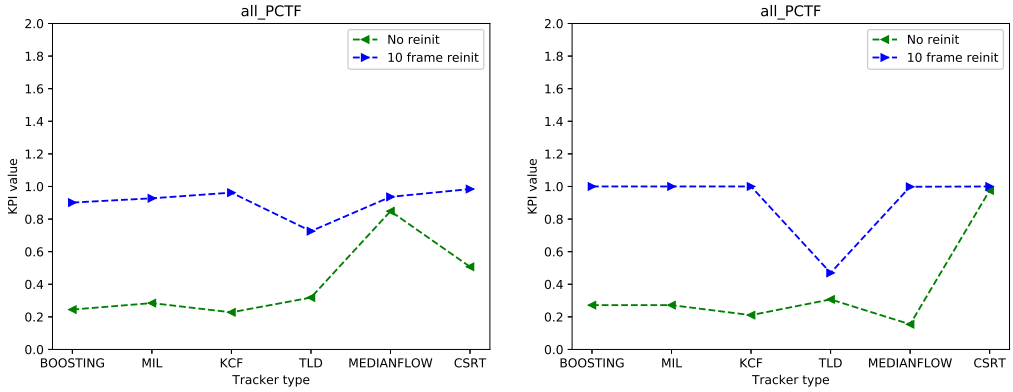
Appendix A

KPI values



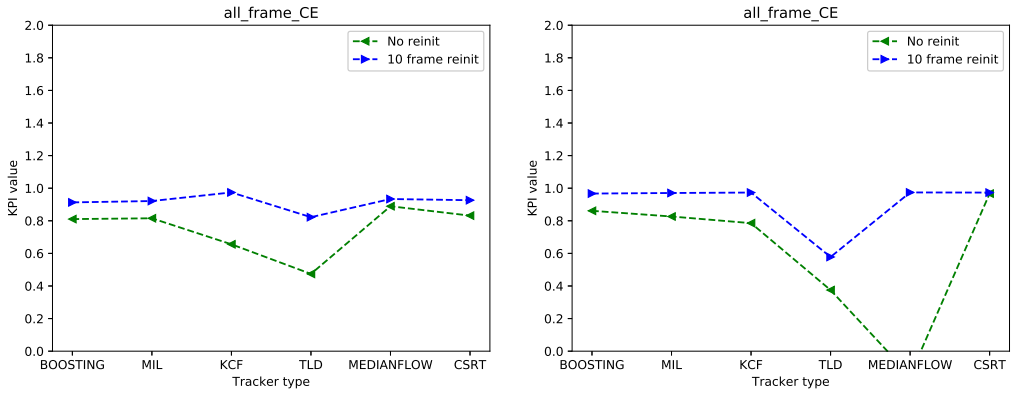
(a) KPI when using vehicles video (b) KPI when using pedestrian video

Figure A.1: *all_AvgROI* KPI trend for all tracking algorithms



(a) KPI when using vehicles video (b) KPI when using pedestrian video

Figure A.2: *all_PCTF* KPI trend for all tracking algorithms



(a) KPI when using vehicles video (b) KPI when using pedestrian video

Figure A.3: *all_frame_CE* KPI trend for all tracking algorithms