UNIVERSITÀ DEGLI STUDI DI TRENTO

Department of Information Engineering
and Computer Science

Master's Degree in

Computer Science and Technologies

FINAL DISSERTATION

Integration of multiple deep learning algorithms for real-time tracking of a person in complex scenarios

Professor

Luigi Palopoli

Co-Supervisor

Stefano Divan

Co-Supervisor

Fabiano Zenatti

Student

Stefano Leonardi

Academic year 2019/2020

Ringraziamenti (Acknowledgements)

Mentre scrivo queste parole giungendo infine alla laurea, mi rendo conto di quanto sia stato lungo questo percorso accademico iniziato cinque anni fa. Da quando ho cominciato innumerevoli sono le persone che m'hanno supportato. C'è chi ha fatto parte solo di una parentesi della mia vita e chi fin dagli albori non è mai mancato. Ora che ho raggiunto questo grande traguardo vorrei ringraziarvi tutti.
Un grazie ai compagni di studi con cui ho condiviso le gioie e le fatiche nell'affrontare scogli sempre strani e complessi. Ringrazio quegli amici che, conosciuti in Finlandia, forse non leggeranno mai queste parole, thank you. Grazie agli amici più cari con cui condivido le più stravaganti passioni che sempre e comunque riescono a farmi sorridere e star bene. Altrettanto ringrazio i parenti a cui molto sono legato.

Per finire vorrei ringraziare chi ha direttamente contribuito a realizzare questo lavoro di tesi. Elena per la paziente e minuziosa revisione linguistica dei contenuti. Il professor Luigi Palopoli per la sua preziosa guida. Stefano e Fabiano per la semplicità con cui in ogni momento hanno saputo consigliarmi su come affrontare questo complesso progetto complicato dal periodo che noi tutti stiamo vivendo.

Contents

List of Acronyms

- BFM (Brute Force Matcher)
- BRIEF (Binary Robust Independent Elementary Features)
- CNN (Convolutional Neural Network)
- CPU (Central Processing Unit)
- CSRT (Channel and Spatial Reliability Tracker)
- DCF-CSR (Discriminative Correlation Filter with Channel and Spatial Reliability)
- DNN (Deep Neural Network)
- FAST (Features from Accelerated Segment Test)
- FCN (Fully Convolutional Network)
- FFT (Fast Fourier Transformations)
- FLANN (Fast Library for Approximate Nearest Neighbors)
- FoV (Field of View)
- FPS (Frames Per Second)
- GoogLeNet (Google LeNetwork)
- GOTURN (Generic Object Tracking Using Regression Networks)
- GPU (Graphics Processing Unit)
- IoU (Intersection Over Union)
- KCF (Kernelized Correlation Filters)
- KNN (K-Nearest Neighbors)
- LIDAR (Laser Imaging Detection And Ranging)
- mAP (mean Average Precision)
- MIL (Multiple Instance Learning)
- MM (Motion Model)
- MOSSE (Minimum Output Sum of Squared Error)
- NMS (Non-Maximum Suppression)

- NN (Neural Network)
- ORB (Oriented FAST and Rotated BRIEF)
- PAF (Part Affinity Fields)
- PRID (Person Re-IDentification)
- R-CNN (Region-based Convolutional Neural Networks)
- ResNet (Residual Network)
- RGB (Red Green Blue)
- RPN (Region Proposal Network)
- SIFT (Scale-Invariant Feature Transform)
- SSD (Single Shot multibox Detector)
- SSP-ReID (Saliency-Semantic Parsing Re-IDentification)
- STAF (Spatio-Temporal Affinity Fields)
- SURF (Speeded-Up Robust Features)
- SVM (Support Vector Machine)
- TLD (Tracking-Learning-Detection)
- YOLO (You Only Look Once)

List of Figures

Summary

This final dissertation is dedicated to the master thesis research project[1] of **Stefano Leonardi** at the **University of Trento** for the **Dolomiti Robotics group**[2]. The goal of this project is to implement a software able to drive a robot into a real environment.

The scenarios that we are dealing with is a situation in which both robots and humans work. This interaction is extremely complex because the two parts need to cooperate efficiently avoiding accidents. The robot interacts with the environment with two sensors. First, the LIDAR instrument that will guarantee the driving safeness. Second, an RGB camera is used for this project to implement a new functionality: *the follow procedure*. While a human is walking the robot should in real-time recognise him as the main subject, the leader, which it must follow across time through the real environment. The procedure should manage conditions in which the leader is completely hidden or it is out of the field of view for a long period of time.

The software that will be presented has a modular structure composed of blocks. This architecture is used in order to be adaptable to future replacements of the technologies used. The general tracking problem is solved with three modules:

- **Object detection** is used to periodically localize bounding boxes of people inside a frame. We have used the two actual states of the art algorithms: SSD[3] and YOLOv3[4].
- **People recognition** is used to understand if the people found correspond to the leader or not. We have ideated a method based on the KNN algorithm that uses images converted into points with the DNN image classifiers: ResNet50[5] and GoogLeNet[6].
- **Object tracking** is periodically started on the leader location and it will track it for few frames avoiding the problems generated due to long-term sequences. The methods found to be the best are: KCF[7], CSRT[8] and MOSSE[9]. Eventually, in future improvements GOTURN[10].

We have tested the implemented solution with an ad hoc dataset created by manually moving the robot around to recorded videos.
The experiments show that the software is able to be executed in real-time on low powerful processing units. In fact, robots cannot carry top quality hardware due to the high cost of the GPUs and also because these components have a too high power consumption. The known weakness of the algorithm relies on the reliability of the people recognition module. In fact false-negative predictions[1] will affect the KNN potentialities and cause bigger and bigger problems as long as the algorithm is executed. Despite this, the procedure is able to manage minute-long frame sequences.

[1] A false-negative prediction occurs when the leader is classified as a random person.

1 Introduction

This chapter offers an overview of the project on which the thesis is based. The goal is to explain in detail how the practical problem has been approached in order to analyse the physical constraints, ideate a software method able to solve them, and how these ideas were then implemented into a working algorithm.

1.1 Physical context

The physical component in this project is a robot. Its definition can vary a lot basing on the context in which it is used. For this project, a robot can be described as a vehicle able to move in the space. A **LIDAR (Laser Imaging Detection And Ranging)** sensor is mounted. Which allows to drive in the space avoiding physical obstacles during the movement. In addition, it is installed a computational device connected to a webcam that can record streams of images representing the space in front of the robot itself. The robots that we are working with are shown in Figure 1.1.
The video camera becomes the eye of the robot itself, and the captured video stream is used as the input of the algorithm working on the computational device. This computer can be composed of a **CPU (Central Processing Unit)** or more likely it is built with a **GPU (Graphics Processing Unit)** that can speed up parallelized computation, applied on the **DNNs (Deep Neural Networks)** used as the core of the algorithm. The software does not assume one component over the other, the only variation is in the performances: a GPU computation speed can be much higher than a CPU.
Instead, the output of the algorithm is a position composed of X and Y pixel coordinates calculated on a single frame captured from the webcam. This location can be then elaborated and, with the use of LIDAR sensor, the robot can estimate which is the 3D position of the element tracked from the software.
Finally, it moves to reach that position, in order to follow the tracked subject not only into the virtual space but also into the real environment.

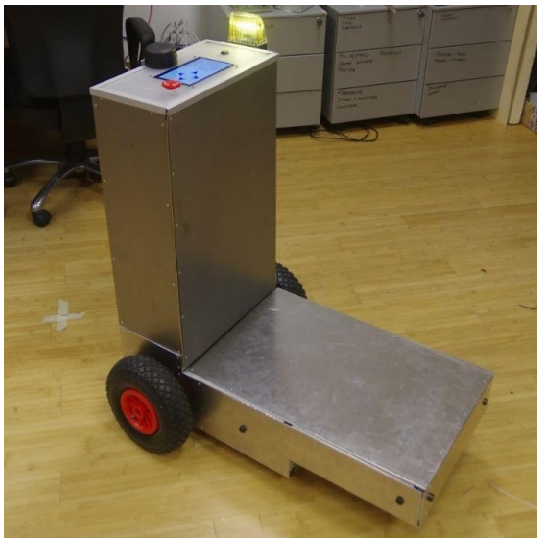
1.2 The problem

The thesis project[1] is based on an internship with Dolomiti Robotics[2], a company working on self-driving robots.
These vehicles are designed to work in an industrial environment. This scenario is populated not only by robots but also people, making the driving task even more complex to achieve.

1.2.1 Robot only environment

A completely automated environment, where humans cannot access looks to be a similar context. Instead, it is completely different because each vehicle has is its own logic that can be designed to fit the requirements of all the other robots working in that area.
The typical solutions to drive a vehicle in this scenario are two:

- Based on a centralized decision unit that moves all the robots simultaneously around.



(a) The first prototype designed to test applications that will be used in an industrial environment.

(b) A small model designed to build a fleet of robots that interact with each other while moving.

Figure 1.1. These are the two model of robots on top of which we have designed this thesis project. The two models share the same physical characteristics.

This unit is responsible for avoiding collisions by knowing the exact position of each single moving robot.

- Based on fixed rules of movement that each robot has to follow. The rules do not allow collision and the automated vehicles respect them.

Both these methods work because an automated vehicle uses a deterministic decision process and does not take arbitrary choices.

1.2.2 Environment shared between robots and humans

Instead, in a shared environment, there are a lot of elements that are not controlled by a deterministic rule. The changes in the scenario are random and prediction cannot be done. There are both fixed object that may have changed position due to external interaction, and also human that walk around with no defined rules.
In this scenario, it is fundamental to choose an input method that can measure the area around, in order to create an autonomous moving vehicle. Therefore LIDAR has been chosen. LIDAR is a technology that measures distances around the robot in a horizontal plane. The effect is that the robot knows in each direction which is the distance from the surrounding objects. This key idea has been used from Dolomiti Robotics, to design a software able to drive robots around avoiding collision with fixed obstacles or people walking.
While a robot moves around, it can construct a map of the fixed objects in the environment, measured with LIDAR. Instead, the moving objects, such as other robots or people, that are recognised as not fixed elements are not stored in the map as obstacles. This reconstruction allows the vehicle to move autonomously from one position to another knowing exactly which path to follow to reach the destination.

1.2.3 Purpose of the internship

The shared environment does not offer any real human-robot exchange. The two parts only share the same spaces. The purpose of this thesis project is to create a physical interaction between the two.
The goal is to create a new functionality *"that allows a robot to follow a person into the real environment"*. How it works:

- Track/follow is the interaction of a robot and a single person (called from now on **leader**).
- The leader starts the *"follow"* functionality standing in front of the webcam of the robot.
- The robot has few seconds to recognise the person inside the camera **FOV (Field Of View)** as the leader.
- Then the leader can freely move around in the space.
- In the meanwhile the algorithm is processing the webcam stream of images recognising the position of the leader and start tracking it in the virtual space, while following it in the real one.
- The tracking continues for a long period, up to several minutes until this functionality is stopped.

1.2.4 Technical problems

This "follow" functionality may be easily solved under certain conditions. However, solving the general scenario, it is a much harder task.
Below is listed a small collection of the principal problems that make this functionality an extremely general one, therefore hard to solve.

- The tracking should be done in real-time. It is impossible to follow a person if the processing speed is too slow. A high **FPS (Frames Per Second)** rate should be respected.
- The robot needs to physically follow the person meaning that the webcam cannot be fixed. By consequence, also the background is not fixed and the entire captured image, subject included, might be blurred.
- The person can move freely around walking fast, slow, or staying.
- The leader is a random person, it is not known while the algorithm was designed (no parameters can be fixed in advance).
- While the leader is walking around there might be also other people that interfere with the algorithm.
- The leader can be hidden from the webcam due to moving or static elements placed between the leader and the webcam itself.
- The leader can exit the field of view of the webcam disappearing until the robot rotates to watch it back again.
- The tracking should be performed for a long period.

1.3 The solution

The problem is complex due to its generality and the necessity to cover a lot of complementary conditions. For this reason more than one solution exists. In this thesis is presented a solution based on the combination of three methods. Each one is designed to solve sub-problem compared to this one, and none of them alone can overcome the challenge of the general task.

1.3.1 Existing technologies

The three technologies are:

- **Object Detection (or Localization):** given an image the object detection task aim at processing the image and recognise which objects exist there. The detection not only need to produce a list of all the classes[1] of objects visible in the image, but also recognise in which section of the frame every single element is.
  The output of detection is a list of: class to which the element belongs, the probability associated and the **bounding box** defined as the smallest rectangle that contains the entire element.

- **Object Tracking:** in this case the input is not a single image but a video stream and an initial section[2] of it. The goal is to remember this portion of the image and recognise it in all the frames after the first one. It is important to note that the tracking procedure it is not designed to follow a person, a car or other it is designed to follow a rectangle of coloured pixels, no matter what these pixels represent.

- **Object Recognition:** this is a comparison between several pictures. These often represent a bounding box of the object that needs to be recognised. The procedure has a database of images each one with a specific class, and the input value is another picture, called **query**, that does not exist in the database but it represents a subject known. The goal is to extract from the database all the images that have a subject that looks similar to the one represented in the query.
  This application is mainly used to recognise humans, often in the video surveillance context. The database is composed of the bounding box of all the people seen, i.e. in a supermarket over the last week, and when a thief is captured and it is used as a query. So, the system should return all the images containing the thief itself.

1.3.2 Limitation of known technologies

The challenges presented previously in Section 1.3.1, can solve a small part of the general problem but each one has a technical problem Section 1.2.4 that cannot be solved:

- Object detection is a computationally expensive task, on a powerful GPU can run in real-time but that is not the case of the robots we are working with.
  In addition, the detection works frame by frame and each one is independent of the previous one. So, if a person is recognised in a frame, and in the next one, there is more than a single person the algorithm does not know the relation between all of them. Meaning that a person cannot be tracked from one frame to the next one.

[1] There are a set of types to which each element can be associated i.e. person, dog, car, bicycle, bottle and so on.

[2] A portion of the image: a rectangle.

- Object tracking, according to the name, seems the task that better match the requirement of the general problem.
  Despite that, the tracking does not consider that the tracked subject, the leader, cannot be hidden from the webcam. The leader should always be visible into the recorded video, and that is not the case. In addition, the leader can also exit the field of view of the robot while walking around.
  Lastly, all the trackers are designed to follow the subject for small periods[3], after a while the tracked rectangle of coloured pixel changes and the precision of the output is no more not guaranteed. This phenomenon is known as the **drift effect**, after a while the drift is so wide that the tracked cannot be trusted anymore.

- Object recognition due to its requirement was not designed at all to run in real-time. In fact, it is enough to run this procedure only when a query occurs, and that does not happen more than one every second.
  Except that, there is a more intrinsic problem with the recognition to approach the general problem. The procedure requires a query that can be the subject at the actual frame, but then it should work on a dataset composed of old frames and these are useless to solve the actual frame.
  In addition, this algorithm cannot be independent, because the input values are bounding boxes of people, but these regions can be computed only with an object detection algorithm. Hence this approach cannot solve the problem independently.

This explanation shows that none of the existing proposed technologies can solve the general problem in all its parts.

1.3.3 Combine known methods to solve the general task

To solve the problem and manage all the requirements it is necessary to create a combination of known methods.
An example of integration of methods to solve a complex task was done by Jiang et al, in their paper[11] that presents a fusion of **YOLO9000**[12] (the second version of **YOLO**[13]) used as object detector and **SURF**[14] used as short-term object tracker.
The paper illustrate an innovative approach based on two thresholds that are used to understand when the drift of the tracker is too large and it is necessary to reinitialize it. So YOLO is executed to find the tracked subject back again and after the initialization the loop can start again.

The method presented in that paper is an integration of two class of methods. Instead in this thesis is presented an integration of three. The third method is necessary because an additional technical problem exists (Section 1.2.4). Jiang et al. work on sport video clips where athletes are always followed by the camera and never disappear out of the field of view. In addition, occlusion can exist but are very short and the tracker is often able to overcome them.
Instead, in our scenario we need to manage the disappearance of the leader behind a corner for a relatively long time. So, the object recognition method was introduced to solve this condition.
These are the main steps of the entire algorithm:

1. The detection module is executed and the bounding box of the leader is found. By assumptions, in this phase, if more than one person is simultaneously found the

[3]Each tracker works on a video of few seconds.

leader is chosen according to the area of the bounding boxes. The biggest area represents the most important person hence the leader.
This step is executed several time to train the people recognition module.

2. A new detection is executed and D people are found.
3. Ad hoc optimizations and the people recognition module is used to choose if the leader is contained in the list of people found:
   - If **yes**: the procedure starts from point 4 (tracking)
   - If **not**: the procedure loops again from point 2 (detection again)
4. The tracker module is initialized with the bounding box found.
5. The tracker runs for the next F frames. When stopped loop back to point 2.

This flow shows how detection, tracking and recognition are combined together to build a complete algorithm, that can run in real-time due to the alternation of slow and fast methods and to manage all the problematic scenarios.
The details will follow.

1.4 Prior technical knowledge

In this section are presented some technologies that should be known in order to well understand the algorithms presented in the next chapters.

1.4.1 NN (Neural Network)

A NN[4] is a black box containing several layers (Figure 1.2a). Each layer takes as input the output values of the layer before. The input of the NN goes into the first layer and the output comes from the last one. Each layer is composed by a certain amount of neurons (Figure 1.2b). Each of these neurons computes a weighted summation of all the output values of the neurons in the previous layer. For each couple of neurons in consecutive layers exists a weight that multiplies the output of the first one to generate the input of the next one. The final output of a neuron is then computed by summing up a bias value and by applying an activation function such as ReLU (Rectified Linear Unit), SoftMax or Identity.

Gradient descend

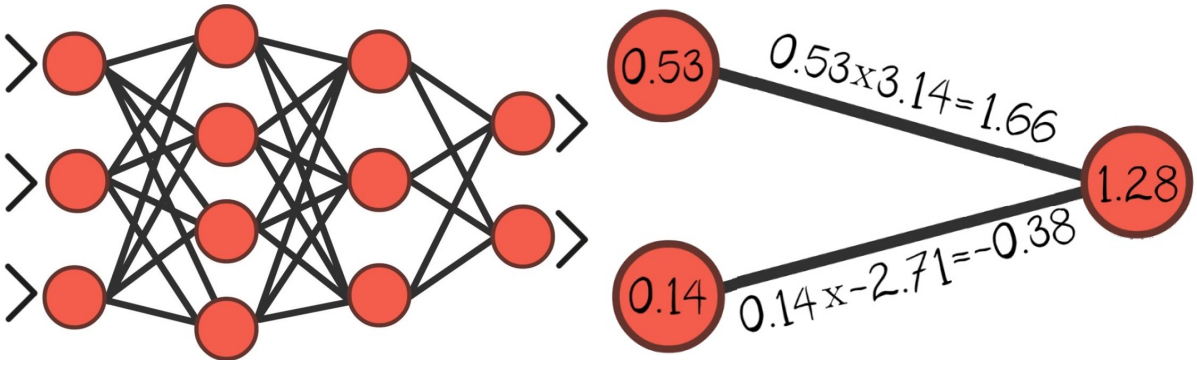
The final prediction of the NN corresponds to the label of the neuron, in the last layer, with the highest value. The overall output of the last layer is evaluated with a **cost function**. The higher the cost the worst were the prediction. A perfect prediction will have cost equal to zero. The general cost of a model is the average cost computed on the entire input training set.
The training of the NN is based on the minimization of the cost function. Since that the cost function is based on thousands of dimensions that represent the weights over the entire NN, the minimization cannot be analytically solved. The solution is the **gradient descend** technique. Basically, a set of input values, and all the associated outputs,

[4]A visual and very well done explanation on how a NN works can be found online. The 3Blue1Brown youtube channel has published videos about the mechanics of gradient descent and about the learning phase based on back-propagation.

produce a specific average cost. This generated value is a point of the cost function, and it has an associated gradient. The measure of the gradient tells us how to modify the weights in order to produce a lower cost on the same input training set. Hence how to improve the NN performances by descending the cost function along the steepest path to reach a local minimum[5].
Unfortunately, the generation of the gradient with analytics techniques is unfeasible. Therefore we need the **back-propagation** technique to measure it.

Back-propagation

Essentially the gradient represents the direction of the steepest path to the local minimum of the function. Another interpretation is that the magnitude of gradient represent for each dimension how the associated weight is useful for the final output prediction. For each input of the NN the neurons in the last layer should be tuned to produce a cost equal to zero. The variation for each neuron, compared to the correct value, depends on all the parameters of its summation. Practically, the elements involved are the neuron's output of the second-last layer and the associated weights. These parameters are linked one after the other, up to the left-most layer (the first one).
Therefore the last layer parameters can be computed to generate the best possible score. This variation will then influence the previous layer and its weights, then the one before and so on and so forth up to the first layer. This is the key idea of the back-propagation. Once plenty of input has been passed through the NN and the tuning have been performed a lot of times, the weights are finally calibrated. The result, defined as the combination of the NN architecture and its weights is the called **model**. In this thesis work, we have never built our model but we have always used pre-trained ones.



(a) The general structure of a NN. The input, taken from the left, is passed through the four layers that generate the NN output on the right side. The four layers are respectively composed of 3, 4, 3 and 2 neurons.

(b) The functioning scheme of a neuron. The outputs of the left-neurons are multiplied with ad hoc weights and then summed up. In addition, the right-neuron will add a bias value and apply an activation function to produce its output.

Figure 1.2. Two schemes that represent the key mechanics of a NN.

[5]We are "only" interest into local minimum since to find the global minimum it is necessary to explore the entire cost function and this requires a huge amount of time.

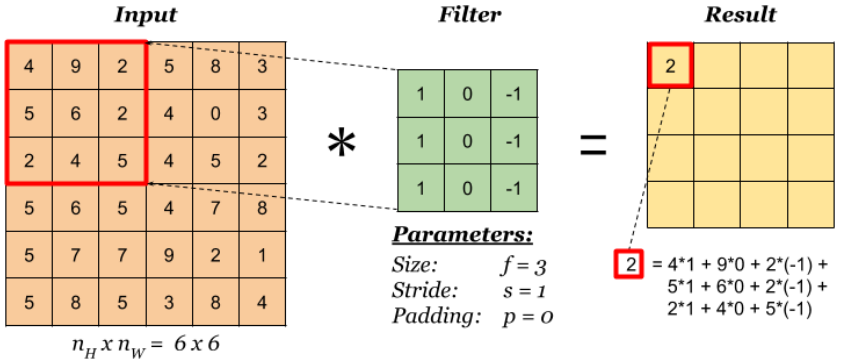
1.4.2 CNN (Convolutional Neural Network)

A CNN[6], also called **ConvNet**, is a standard NN extended to process images.
The novelty of this NN is characterized by a new type of layers: the **convolutional layers**. These layers are the main components of a CNN and are stacked one after the other. Often the end of the NN is composed of few traditional layers that compress the huge output of the previous layers, to generate the overall output of the CNN. A convolutional layer is a block created to discover patterns inside an image. These patterns are initially simple like stripes and corners (a visual example is shown in Figure 1.3a). Then, by combining a lot of convolutional layers the learned patterns are always more complicated. Whether the CNN is trained to recognise people these advanced schemes may represent entire objects such as hands, arms, legs, and so on and so forth.

Convolution filter

A convolution layer is built on top of the mathematical operation of convolution.
This operation is based on a **filter** that is a **small squared matrix** of real numbers ($\in [0, 1]$). This matrix will slide over all the image pixel by pixel, this shifting is called **convolving**. For each shift, the filter produces an output that is the dot-product of the actual portion of the image times the filter. In Figure 1.3b is shown a numeric example of how the convolution filter works.

(a) Example of simple patterns generated by a CNN.



(b) A numeric example of a convolution operation. The top-left, 3x3, section of the image is multiplied with a dot-product by the filter. The result is placed in the output matrix in the top-left corner.

Figure 1.3. The overall mechanics of a CNN.

1.5 Structure of the thesis

The next chapters are organized as follows. This section concludes the introduction (Chapter 1).
Then follow three chapters one for each main method: object detection in Chapter 2, object tracking in Chapter 3 and object recognition in Chapter 4.
An overview of the entire algorithm and how it works together follow in Chapter 5.
In the end, the conclusions are presented in Chapter 6.

[6] A visual explanation of the CNN, created by the DeepLizard youtube channel, is available here.

2 Object Detection

This chapter explains into details what is the object detection task and the methods that can solve it efficiently. Moreover an overview of other methods is given among whose there not efficient ones and others that may seem to solve the task but do not.

2.1 Task definition

Object detection, also known as **object localization**, is an evolution of the **image classification** (Figure 2.2). In classification, an algorithm should produce a list of all the classes of objects inside the image. Instead, the detection not only calculates which object class exists but also how many occurrences are present for each class. Then, the complex part, and the most interesting one for this thesis application, is localizing where those elements are placed inside the image. The position is not considered as a point but as a bounding box defined as the smallest rectangle that contains the entire element. An example of object detection is shown in Figure 2.1.

Figure 2.1. Object detection applied on a sample image[1].

2.1.1 Similar tasks

Object detection can be additionally improved to extract even more information from an image.

[1]Several images and implementing ideas of this thesis come from the online blogs pyImageSearch[15] and LearnOpenCV[16].

The main evolutions, shown in Figure 2.2 are:

- **Semantic segmentation:** takes all the bounding boxes produced by an object detector, and for each one, it calculates the pixels that belong to the object itself and the ones that do not. By doing this each class has its own colour associated. As a result, the algorithm knows for each pixel if it belongs to one label associated with the image (semantic division) or to the background (yellow in the image).
- **Instance segmentation:** is similar to semantic segmentation, but in this case, each instance of the object is considered as a new element. In fact, the three cubes in the figure have different colours associated to them.
  This task is solved by the **Mask-R-CNN algorithm** (Section 2.3.1).
- **(Human) pose estimation:** is the most complex task among the five. Mainly applied to people, this challenge consists in the estimation of the 3D position of the body. The idea is to build up a skeleton of the person in the image and understand how its body limbs are positioned. This functionality is important to understand what a person is doing in the image.
  This task is solved by the **OpenPose estimation algorithm** (Section 2.3.2).

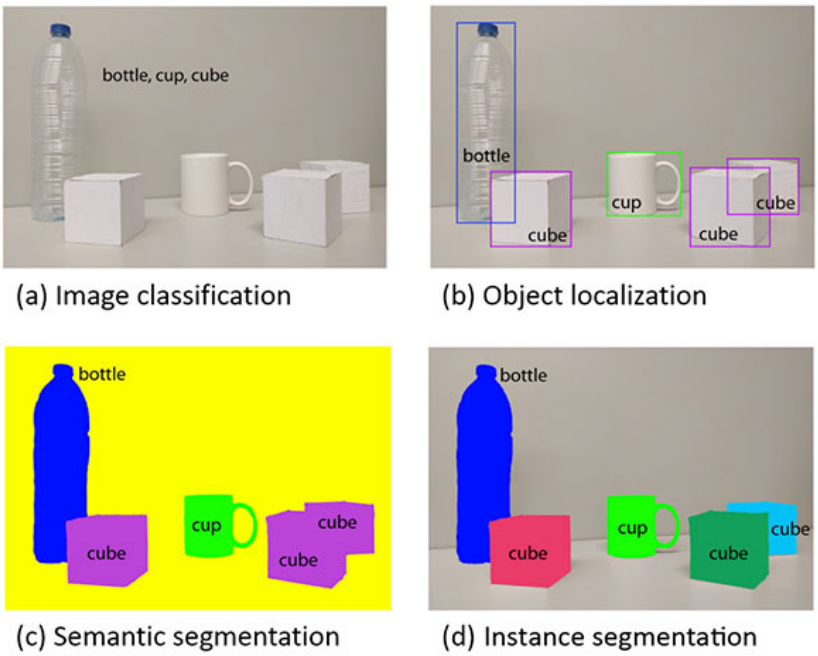


Figure 2.2. Similar problems respect to object localization/detection.

2.2 State of the art algorithms

Object detection has a lot of applications both in real-time, such as in this thesis, but also in safety-critical scenarios like cars with autonomous driving. This division brings out two different metrics: precision and speed. The ideal detector is both fast and precise, however this algorithm does not exist yet. The methods can be divided into two categories.
The solutions mainly focus on speed: YOLO (Section 2.2.1) and SSD (Section 2.2.2).
Instead, the one mainly focused on precision is R-CNN (Section 2.2.3).

2.2.1 YOLO (You Only Look Once)

YOLO[13] was initially designed in 2016. At that time it was the first object detector approach to use a single **CNN (Convolutional Neural Network)**. Redmon et al. goals were to create an extremely fast detector. An overview of the overall procedure is shown in Figure 2.3, and the architecture appears in Figure 2.6.
The image shows a two step procedure, but these steps are solved in parallel. This is the core idea of the paper. A single CNN can be highly optimized.
The YOLO procedure works as follows[2]:

- Preprocess: the image is resized to fit the standard input dimension of the CNN.
- Left-image: the picture it is divided into a grid of CxC cells.
- Top-image: each cell suggests some bounding boxes centred on it, that can match elements in the background. To each box is associated a value describing the probability that it contains one of the elements of the image.
  At most one detection per box can be selected as correct. This relies on the assumption that two correct bounding boxes cannot share the centre. This is both an efficient idea but also a big limitation. Too small elements, close to each other, cannot be both detected.
- Bottom-image: each cell has an associated probability regarding a label that represents the class that can be found in that cell if an element exists in it.
  I.e. the cyan cells means: "if there is something here, it will belong to class 'DOG' ".
- Right-image: the two partial elaborations are merged. The most likely bounding boxes are chosen and classes are associated to them according to the value for each cell in the probability map.

That was the first YOLO version, in this thesis is used the third[4]. Mainly, the changes were about recognition of a wider set of classes and small implementing details to improve the overall precision of the algorithm.

The output of the CNN is generated extremely fast and it is accurate however has a big problem. Often if two classes have similar probabilities or the shape of the element is not perfect YOLO might propose more than one bounding box for each element. That is the case of Figure 2.4c where the truck is classified both as "truck" and as "car". The same happens to the person that has been seen twice.
To solve this, it is necessary to apply a new technique: Non-Maximum Suppression.

NMS (Non-Maximum Suppression)

This technique[17] is a post-processing that works on the bounding boxes suggested, as output, from YOLO or other detectors. NMS does not consider the source image. The goal of this procedure is to refine the bounding boxes proposed and to choose which subset of them is better to fit the final image prediction. Two examples of applications are shown in Figure 2.4.
The main flow of the algorithm is as follows:

[2]The original presentation of YOLO by Redmon at CVPR 2016 conference can be found here.

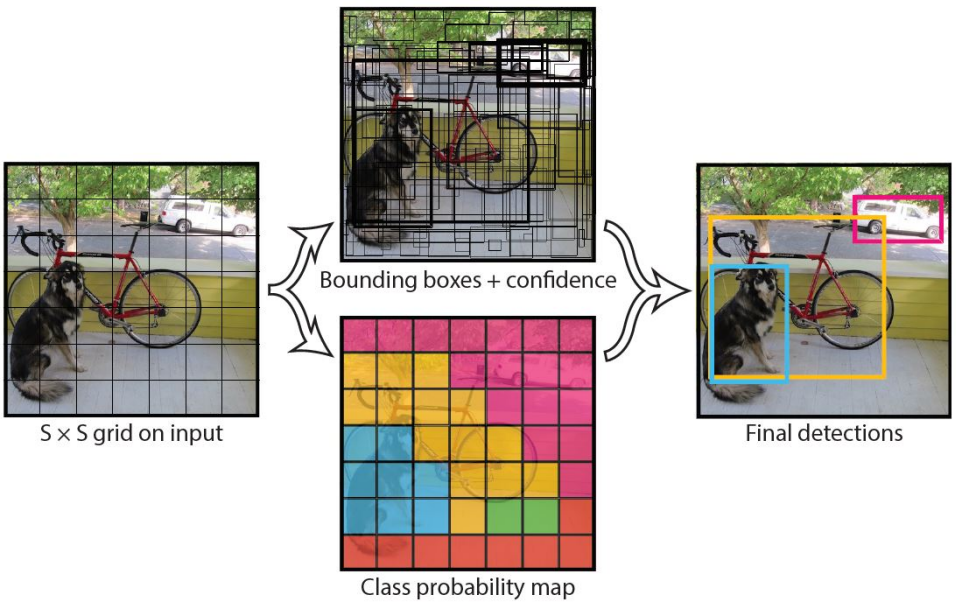


Figure 2.3. The YOLO image elaboration based on bounding box proposal and class probability map.

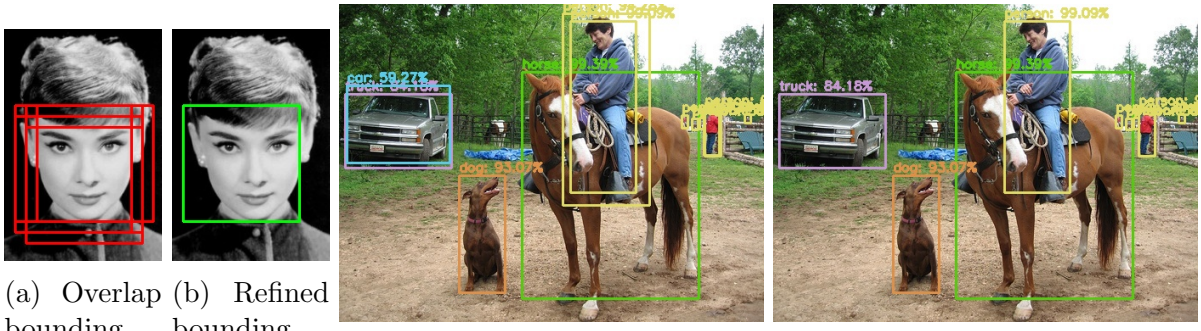
- The input is a list of all the boxes generated for a single image. Associated to each one there is its probability.
- The boxes are sorted in decreasing order according to the probability associated.
- Then, each box is accepted or rejected according to the **IoU (Intersection Over Union)**. That is the percentage of overlapping area with an already accepted box.
  - If the IoU is above a certain threshold, meaning that the two boxes overlap too much, the one with the lowest probability is discarded.
  - If that is not the case, the box is accepted as a new prediction.

The input in Figure 2.4a, is processed and only one box is accepted (Figure 2.4b) because the IoU is very high. Instead, in Figure 2.4c, two boxes are removed respectively from two other separated boxes (Figure 2.4d) because two different subjects are involved.

2.2.2 SSD (Single Shot multibox Detector)

The principal competitor of YOLO is SSD[3]. Both are based on the same principle: the use of a single Convolutional Neural Network to propose bounding boxes and associate them to classes. The CNN is optimized as much as possible to improve the speed performance and eventually even the accuracy.
The difference relies on how the two algorithms deal with the bounding boxes suggestion. YOLO for each cell of the grid chooses a couple of options and at most one can be chosen. On the other hand, SSD works as follows (Figure 2.5):

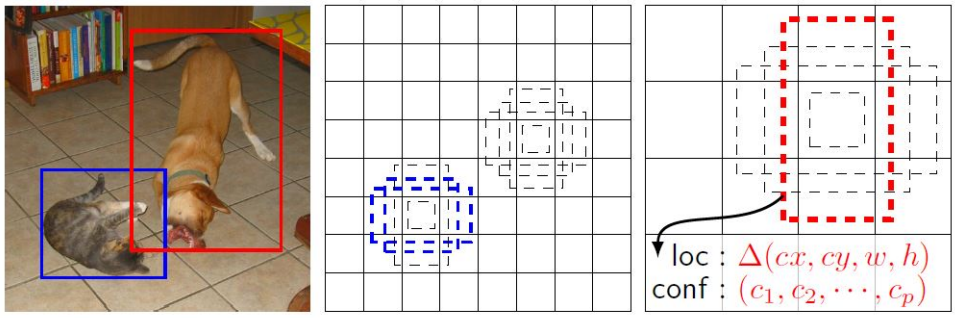
- The image is divided into a grid of CxC cells, called **feature map**.
- Each cell can propose a set of default boxes that has a size measured in cells (i.e. 3 cells high and 2 wide).



(a) Overlap bounding boxes. (b) Refined bounding box. (c) YOLO generated bounding boxes. (d) Apply NMS to refine the YOLO's output.

Figure 2.4. Two scenarios of application of Non-Maximum Suppression algorithm. First: choose which of the 6 manual generated bounding boxes, on Audrey Hepburn's face, should be considered the correct one. Second: refinement of the YOLO prediction output, by removing the "car" and "person" prediction.

- The process is repeated many times varying the value of C: the **granularity of the grid**. This guarantees that the algorithm is scale-independent matching both, big and small, subjects.
  In Figure 2.6 is shown how the convolution layers blocks are matched together only at the end.
- All the suggestions are merged together to produce the final proposals.
- SSD internally performs NMS to remove unnecessary detections.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 2.5. The SSD image processing and how the bounding box proposal is elaborated.

MobileNet

The implementation of the project does not use a traditional version of SSD, but a lighter one. This model is a combination of SSD and mobileNet[18].
MobileNet is a methodology that approaches Convolutional Neural Networks to transform the architecture structure to build a much lighter version of the model. The concept was first ideated to allow low power devices, such as smartphones, to run computational expensive algorithms based on CNN.

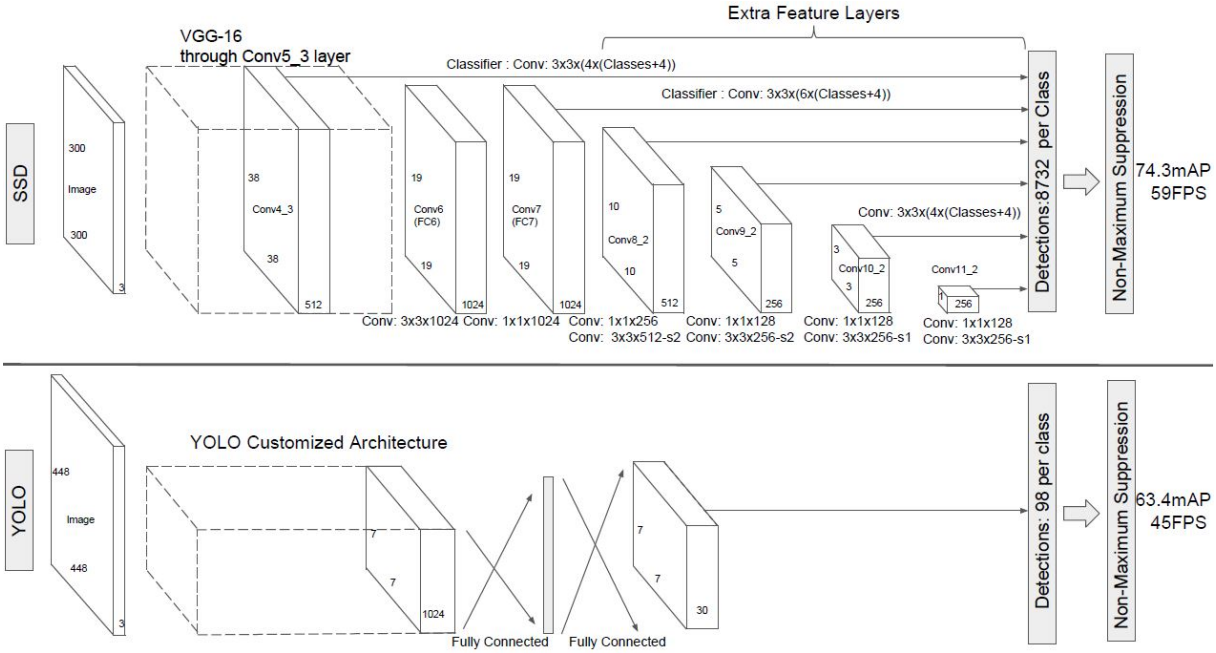


Figure 2.6. A comparison of architectures between SSD and YOLO which is designed as a compact block. Instead, SSD is modular, it is divided into convolution layers of different scales, combined at the end, to make the algorithm scale-independent.

The principle is to replace each standard convolution (Figure 2.7a) with a **Depthwise separable filter**. A standard convolution works on a grid of DxD pixels and for each one produces output features of depth M. This operation can be repeated N times for each source feature.
The operation is broken into two other simpler convolutions:

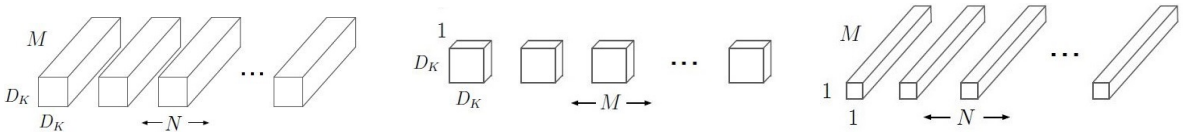
- **Depthwise convolutional filters** (Figure 2.7b): produces only one feature output at a time, repeated M times for each DxD grid.
- **Pointwise convolution filters** (Figure 2.7c): extends the output feature of the depthwise filter to N output features.

The original paper demonstrates how these two operations stacked in a row, can produce results close to the correct ones.
The computational cost determined by the number of parameters, used by depthwise and pointwise filters, can be further reduced by randomly removing a percentage of these parameters. According to the portion of parameters removed (25%, 50%, 75%), the algorithm precision is affected.

2.2.3 R-CNN (Region-based Convolutional Neural Networks)

R-CNN[19] was the first invented method of the three in this section. Differently from YOLO and SSD, it is mainly focused on performing detections with high precision, despite the processing time.
This algorithm is a two step object detector. The workflow, shown in Figure 2.8, works as follows:

1. A region proposal algorithm is executed on the input image and it produces 2000 bounding boxes.



(a) Standard convolution filters

(b) Depthwise convolutional filters

(c) 1x1 convolutional filters called Pointwise convolution filters

Figure 2.7. The novelty of mobileNet is that it converts a traditional convolution (A), into a combination of two lighter convolutions (B-C), that produce almost the same output.

2. Each of these proposals are elaborated independently.
   For each box, an image classifier, based on CNN, produces features from the image and then predicts which classes they might contain.
   Any kind of image classifier can be used for this task resulting in an algorithm that can be easily adapted with new networks.

The main problem is that overlapping proposals are elaborated independently. The result is that feature extraction is performed on the same area of the image multiple times. These have been solved with a second version of the algorithm, called **Fast-R-CNN**[20]. The feature extraction for the full image is performed before the image classification that now works on an already generated image of features.
An incremental improvement, comes from the third version: **Faster-R-CNN**[21]. It performs the feature extraction as the first step and based on it the bounding boxes are proposed with **RPN (Region Proposal Network)**. The modified structure allows ad hoc optimizations to improve the low processing time.

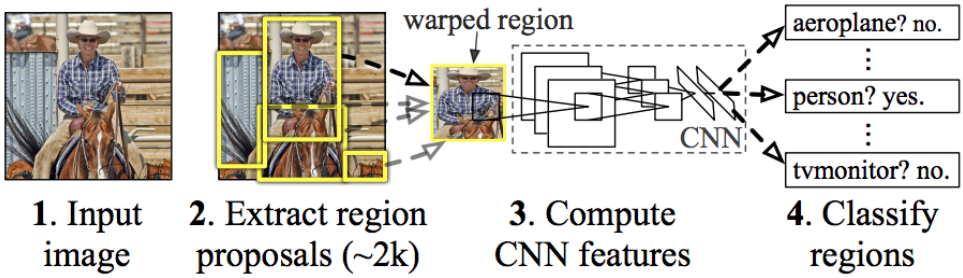


Figure 2.8. The 2-step elaboration of R-CNN model on a sample image.

2.3 Other famous algorithms

2.3.1 Mask R-CNN

A variation of R-CNN that aims to solve the instance segmentation task (Figure 2.2) is Mask R-CNN[22].
Mask R-CNN is built on top of two technologies:

- Faster R-CNN used as an object detector.
- **FCN (Fully Convolutional Network)**[23] that performs semantic segmentation.

For each bounding box, it is known the class, then FCN computes the segmentation of that class. All the shapes of elements in the image are then merged together to build the instance segmentation result. An application of this algorithm is shown in Figure 2.9.

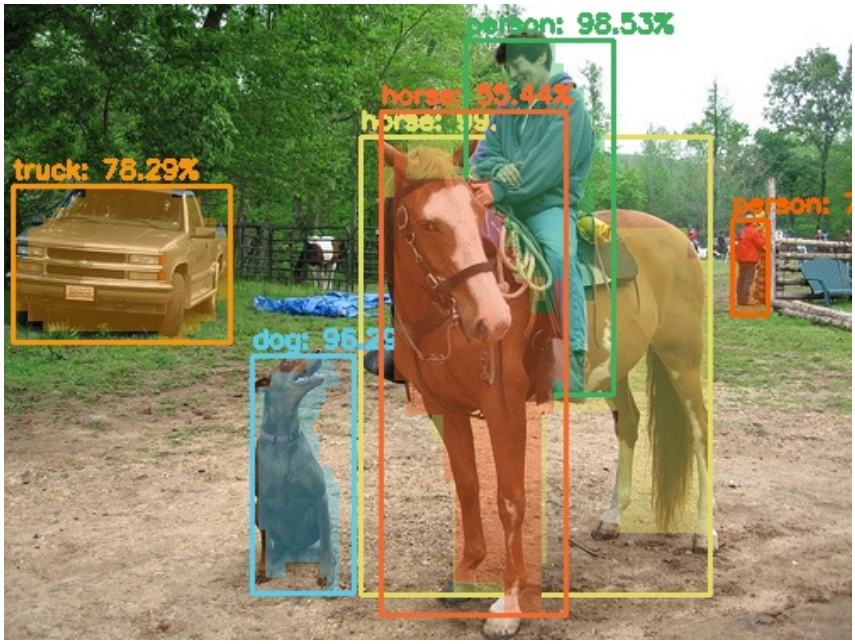


Figure 2.9. Instance segmentation of an image using mask R-CNN.

2.3.2 Open Pose

Firstly designed in 2017 OpenPose[24] aims at processing an image and recognising the position of the people in it. The position is the skeleton of a person, it is the interconnection of limbs that link 15 points on the human body.
This algorithm opens al lot of possibilities because until then estimating the body position was achieved with 3D cameras, extremely expensive hardware that now can be easily substituted. Recently, OpenPose has been improved in terms of speed, to process frames in a video with **STAF (Spatio-Temporal Affinity Field)**[25], and it is also been extended to understand the 3D human position.
An overview of the overall procedure of the algorithm is shown in Figure 2.10, instead some output examples are shown in Figure 2.11.
The OpenPose procedure works as follows[3]:

a) Preprocessing: the input image is reshaped to match the requirements of the two-branch CNN.

b) Part Confidence Maps: the first branch of the CNN process the image to extract the location of the 15 body parts.
Each body part (i.e. left shoulder, left knee, right wrist...) is detected by an ad hoc filter. These filters do not process one person at a time but the entire image simultaneously. The result is that a filter recognises all the visible right elbows in the image. This is extremely important because, by doing this, the algorithm process speed is independent respect to the number of people in the image.

[3]The original demo of OpenPose by Zhe Cao at CVPR 2017 conference can be found here.

c) Part affinity fields: the second branch of the CNN process the image to recognise the limbs that can connect all the body points found so far.

d) Bipartite matching: has the goal to match all the elements found to reconstruct the skeleton of the entire person.
This reconstruction is a greedy approach mainly based on geometry that wants to minimize the distance of two parts that must be connected.

e) Parsing results: the output image is assembled with the full-body poses for all people in the image.

It is important to note that OpenPose is a bottom-up approach. The algorithm has no knowledge about the positioning of people in the image, it only tries to reconstruct the skeleton from small sections of the body.

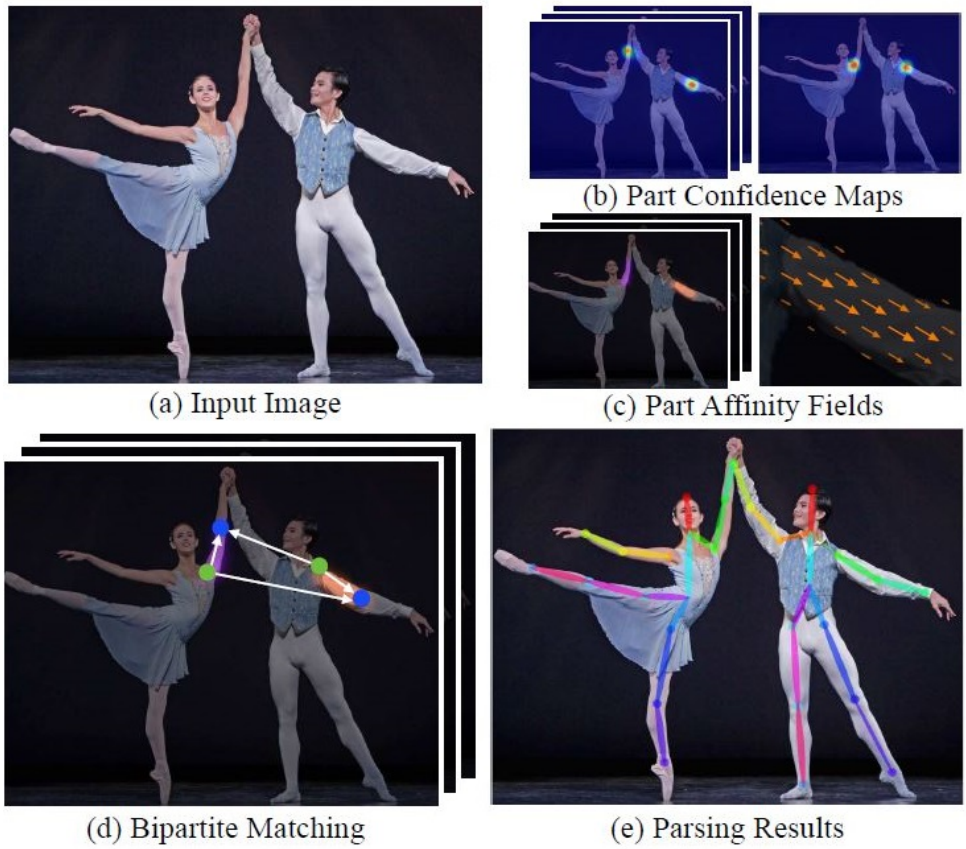


Figure 2.10. The elaboration of the OpenPose algorithm to recognise the skeleton of the two dancers in the image.

2.4 Overview of the algorithms

The algorithms presented in this chapter represent the state of the art methods for their field of application.
All those methods can be used to achieve the long-term tracking that is the goal of this thesis, but the different information generated should be used to solve the problem with different approaches. We have chosen to use the methods that perform only the detection task. This results on one hand into general information as output, and on the other hand into a very high processing speed that is fundamental for real-time application.

Figure 2.11. Some examples, taken from the original paper, on how OpenPose works.

Performances comparison

A comparison of the speed, measured as **FPS (Frames Per Second)**, and of the precision, measured as **mAP (mean Average Precision)**, is shown in Table 2.1. The data are based on the Pascal VOC 2007 dataset[26] and come from multiple papers[13][12].
The measures do not show YOLOv3 because compared to the other methods it is more recent and a fair comparison does not exist. Instead, for Mask R-CNN and OpenPose, only the frame rate is shown because the mAP can be computed. However it is completely irrelevant respect to the other presented in the table. These two algorithms perform different tasks hence the precision of the result cannot be compared.
By looking at the data, it clearly appears that the single-stage algorithms (SSD and YOLO) are much faster respect to the two-stage methods (R-CNN), in fact, they run around 5-10 times faster than R-CNN. Instead, the precision of the three detectors is almost the same. For these reasons, we have chosen to use in this project the last version of YOLO and a lighter version of SSD: mobileNet-SSD. This idea pays a few percentage points in term of mAP but implements the CNN with fewer parameters, results in a low power consumption method. This aspect is important because the robots do not mount top quality hardware, therefore the light version of SSD can be executed more easily.

Output visualization

To visually show the potentialities of these algorithms we have applied all of them on the same picture. This elaboration is presented in Figure 2.12. Independently from the task that they solve, it appears evident that all of them recognise the 5 people in the foreground, but there are some differences:

- SSD has troubles with the player on the left. In fact, the percentage associated with him is only a 32%.
- YOLO is able to detect even a sixth person in the background that none of the others have seen.
- YOLO is trained to recognise a wide variety of objects respect to the other detectors, in fact, it is able to recognise even the "sports ball".

Even SSD is adaptable and can be trained to recognise the ball. However the big advantage of the second version of YOLO (also called **YOLO9000**), is that it was integrated with the **Wordnet graph**[27] to be scalable in terms of the number of classes recognised. The result is a detector that is able to recognise up to 9000 different classes.

- Mask R-CNN is, in this example, the most accurate algorithm. It recognises four people with a precision of 99% and the last one with 92%.
- OpenPose, by estimating the body parts, can even understand the orientation of the people in the soccer field. This big advantage can be used to understand where these people will move in the frame after this one.

Considering these reasons, it is evident that discarding OpenPose and Mask R-CNN, in favour of SSD and YOLO, is only a choice for this project. All these algorithms have potentiality that can be used to create a good object tracker.

Pascal VOC 2007	YOLO			R-CNN				
Algorithm	v1	v2	SSD	R-CNN	Fast	Faster	Mask	OpenPose
FPS	45	**67**	46	0.05	0.5	7	7	10
mAP	66	**76**	74	53	70	73	X	X

Table 2.1. Overview of the performances of the algorithms presented. The evaluation is based on the mAP and the processing speed. In bold the best scores and underlined the mAP of the three states of the art detectors.

(a) YOLOv2: detection

(b) SSD: detection

(c) Mask R-CNN: instance segmentation

(d) OpenPose: human pose estimation

Figure 2.12. An example of application of the four main algorithms, based on the same sample image.

3 Object Tracking

This chapter is focused on a few methods that solve the traditional task of object tracking. For each algorithm the potentiality is shown and the advantages, compared to other methods, will be discussed. In addition, an overview of similar problems is given in Section 3.1.3.

3.1 Task definition

Object tracking is a challenge that, differently from the detection, does not work with images but deal with videos. A video is technically defined as a sequence of images, called **frames**, combined together at a certain frame rate. Typically this rate is 15, 30 or 60 FPS (Frames Per Second).
The interconnection of the frames of videos is used from the object tracker algorithms to create knowledge, based on the previously analysed images, by storing information. An object detector elaborates a picture at a time, each one independently from the other. There is no connection among images of the same database. Instead, a tracker process images one at a time but it uses the knowledge, of the previously analysed frames, to understand the current incoming image.
The task baseline is defined as:

> ***The traditional tracking challenge*** *consists of tracking a single well-defined subject over a short video clip, no matter if it is a person, an animal or an inanimate object. The target of the tracking clearly fully appears into all the frames of the sequence.*

3.1.1 Interface

Structure of the tracking algorithms interface:

- One input value is composed of an image, typically the first frame of a video. In our case of real-time elaboration the image corresponds to the actual view of a webcam connected to the device.

- The other input corresponds to the bounding box coordinates of the subject that needs to be tracked. The bounding box can be manually generated or can be automatically identified with an object detector.

- All future steps get as input a new frame. The task is to understand, in this new image, where is located the subject defined by the initial bounding box.
  Then, if all these elaborated images and the generated bounding boxes, are combined together, the result is a video. The initial subject of the video while moving is always centred into a rectangle that follows it across the entire **FoV (Field of View)** of the camera.

3.1.2 Scope of the tracking

It is important to note that the task is called "*Object tracking*" and not "*Person tracking*" this is because there are no limitations to the subject that needs to be tracked, it does not necessarily need to be a human. A better name for the challenge could be "*Area tracking*" because the algorithms should follow the rectangle of coloured pixels that was located in the initial bounding box. No assumption can be done of which type of subject exists inside the bounding box.
Generally, these methods work if the initial position is centred around an object that moves consistently in the space and does not change aspect. Some examples that could break these methods are:

- The initial bounding box contains two or more objects that move independently from each other. The algorithm will recognise one of the two as the main subject and will track it over the video and lose the other one.
  This is a problem if accidentally the bounding box contains something that is "more important" than the "*real subject*". It can also be an advantage that allows to easily discard the background that is an "*object*" itself.
  Since two independent objects move differently in the space compared to a single unique object, tracking them can easily create unexpected errors due to inconsistent movements.

- The subject changes aspect too rapidly due to different luminosity in the environment, or elaborated video sequences that are not "*natural*". This last possibility often breaks methods designed for real situations, so it is not a problem only in this case. Instead, the change of luminosity often occurs but a solution can be achieved with frame pre-processing, such as the standardization of the luminosity of the image to always process a figure with the same luminosity.

3.1.3 Deal with special conditions

The tracking task can be seen as a set of problems. This is because there are a lot of conditions that can modify the scenario where the algorithms should work. By modifying the type of difficulties in the videos some methods may fail while others may not.
The goal of this thesis is to build a "person tracker", that should work under certain conditions. The problem is that the combination of a lot of requirements makes the problem harder.
Despite the time spent for a programmer to ideate, implement and test a solution there is a trade-off to consider. Solving a hard task requires a more computational complex solution compared to solving a similar easier task. This complexity influences the performances of the proposed algorithm. To conclude, it is important to understand which are the requirements of the problem that we are dealing with, in order to choose the algorithm that it is better to use, to solve the task and to perform it fast.

Below are listed a set of requirements that make the baseline harder (definition is in Section 3.1):

- **Changes of 2D shape**: the target due to movements might change its ratio, aspect, and shape. We are interested in what the camera sees of the subject (2D space) and not the effective actual condition of the subject (3D space).

- Partial occlusion (Figure 3.1f): it may happen that during the video, part of the subject is occluded, by an object. If this happens the algorithms must be designed to be robust and to localize the target even with a small section of it. The bounding box generated may contain only the visible part of the subject but it may also happen that an estimation of the entire subject is done and the bounding box contains both the visible and the estimated missing area of the target.
- Rotations and deformations (Figure 3.1b): the object tracked while moving can rotate and show to the camera a different side, or if it is deformable, change the shape (i.e. a person walking change the shape continuously). This might influence even colours.

- **Total occlusion** (Figure 3.1i): the subject completely disappears behind an object or out of the camera field of view.

  - Short-time occlusion: the subject is hidden for a very small number of frames (i.e. 5 or less). This often happens when the subject or an obstacle is moving and the three elements, subject, object and camera, are aligned. These very short occlusions may be solved using a little memory that stores the subject information for the last few frames (i.e. 10).
  - Long-time occlusion: the occlusion lasts for a bigger number of frames, even seconds. Often it occurs when the subject exits the field of view of the camera and does not enter it again for a while. It can also happen if the subject is behind a big obstacle such as a vehicle or a wall.
    It is harder to solve compared to the previous scenario, because the solution requires a long term memory associated with a recognition procedure to understand when the subject is visible back again.

- **Fast-moving object**: the subject shifts for a big portion of the picture from the old position to the new one in a single frame. Meaning that there was a big 2D movement. This could happen because the subject is effectively moving fast, or because it is close to the camera and even a small movement looks big.

  - Blurred subject (Figure 3.1d): due to its movement the subject is blurred and this heavily changes its aspect. The modified elements are the shape and the colours that are somehow faded. In addition, especially for humans, moving fast can make parts of the body such as arms or legs disappear.
  - Proximity assumption (Figure 3.1g): a lot of trackers are based on the assumption that the subject moves around only a little bit. If it moves fast this principle is broken.
    After knowing the exact location of the subject on the previous frame starts the estimation on the following one.
    At this point there are two things to focus on:

    ➤ If the subject does not move is probable that the prediction will place the target in the exact same position as before. Instead, if it moves the probability that the tracked object will be located close to the previously known position is higher than to be located far away. Specifically, the likelihood can be seen as a **Multivariate Normal Distribution** centred

at the previous position and stretched towards the direction where the subject is moving (Figure 3.1h).

- ➤ If multiple similar objects exist in the frame and the tracking is following one of them, it is probable to lose it. This happen with a fast movement that generates an unpredictable big shift, and the target moves far away from the previous position while the tracker recognise its subject into a similar object. From now on, the tracking is broken because it follows the wrong physical element.
- ➤ A big shift can also be wrongly interpreted as a total occlusion.
- ➤ On the other hand, reducing the tolerance to big movement allows the algorithms to only search locally around the last known position resulting in a big improvement of the overall performance.

- **Low-resolution images** (Figure 3.1a): as with many other computer vision challenges the resolution of the input image is fundamental. A low number of pixels per frame results into a bounding box (often a small portion of the entire image) of very low resolution. Because of this, recognising the key elements that identify the subject is hard, but also faster and computationally cheaper.

- **Moving background/camera**: working on a fixed camera allows the use of a set of techniques based on background subtraction. The key idea is to get prior knowledge of the background and understand which objects are there, by removing the known part pixel by pixel. A big change of luminosity can make this harder, but still possible.
  A camera that rotates always on the same section can be managed as a fixed camera. This can be done by combining all the images along with the rotation as a unified one, creating a panorama image.
  Instead, a moving camera implying a moving background makes everything harder:

  - No background subtraction can be done.
  - The subject may change aspect even if it does not move.
  - The occlusion, both partial and total, can occur more easily.
  - A zoom or rotation of the camera causes a similar effect such as the "fast-moving object".
  - To estimate the movement speed and direction of the subject it is necessary to know the movement of the camera because one is relative to the other. However, almost always the displacement of the camera in the space is unknown.

- **Real-time**: designing a method to run in real-time requires to focus on the computational capability and respectively to the processing speed. Real-time might vary from 1 to 60 FPS according to the application. To achieve this speed it is often necessary to choose a faster method instead of an accurate one. This reduces the overall precision of the entire designed algorithm.

- **Long-term video sequences**: the input frame sequence is longer than a few seconds, up to minutes. Firstly, a long video might easily contain some of the problems listed above. In addition, trying to locate the same subject over and over

again without a reinitialization can fail due to the drift problem.
The **drift problem** consists of an accumulation of small errors along the tracking period. A tracking algorithm refers to the bounding box generated for the previous frame. The subject that should be located again in the new frame is extracted from that bounding box. If the tracking lasts for a small number of frames, the subject may look similar to the original one. Instead, during a long processing the bounding box starts to derive from the original subject. Due to partial occlusion, the box might be reduced. Due to a strong change of luminosity, the bounding box might be misaligned. Due to a fast movement, the box can be linked to the background. By continuing summing up all these little problems the main subject will be recognised as a superfluous element. Therefore the tracker stops to consider it useful and the track fails.
To solve this problem the solution is to re-initialize the tracker often before reaching the limit of the drift. This can be done with different methods such as a **Kalman filter** or by recognising the main subject with a detection, that is the method presented in this thesis.

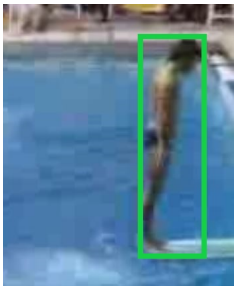
- **Multiple subjects** (Figure 3.1e): if there is more than a subject to follow the naive solution is to apply an object tracker to each of them. Each tracker works independently from the others therefore given X subjects the performances f measured in FPS are reduced to f/X.
  An efficient solution consists of considering all the tracked subject at the same time with a single algorithm instance.

- **Type of the subject** (Figure 3.1c): the baseline problem does not assume any kind of prior knowledge about which type of subject should be tracked. However sometimes ad hoc solutions are required, which simplify the problem. Frequent choices are people (such in this thesis), animals, vehicles or inanimate objects. For example, if vehicles are chosen the change of shape is not a problem because a car always looks the same.

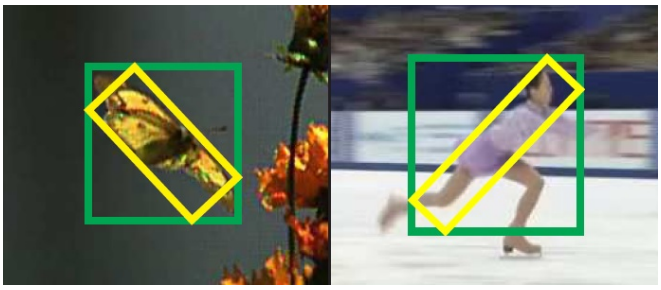
3.1.4 The traditional tracking problem compared to the thesis project

Some of the aspects explained in the previous section are extremely frequent in a lot of videos and databases, hence they are considered as a normal scenario. Others instead are often not even considered in the samples and the majority of the algorithms do not officially solve them.

- The conditions that are often respected in object tracking are:
  - The low-resolution images to allow the elaboration even without powerful computation devices, such as a smartphone.
  - A moving camera because a great majority of videos are in movement and shaking, except for the video surveillance field where the video camera is placed.
  - The changes of 2D shape and the fast-moving objects should be respected, the robustness on these two points make the algorithms more or less reliable.
  - A short-time total occlusion might sometimes be managed, but it is rarely guaranteed.



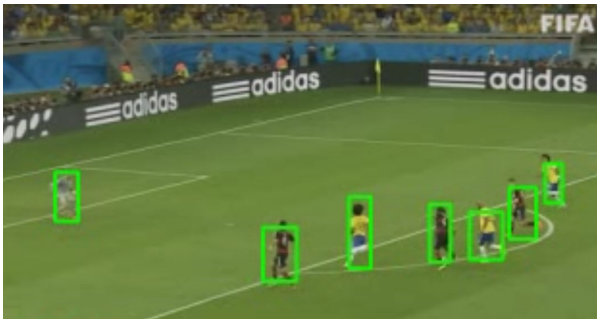
(a) Images with low resolution.



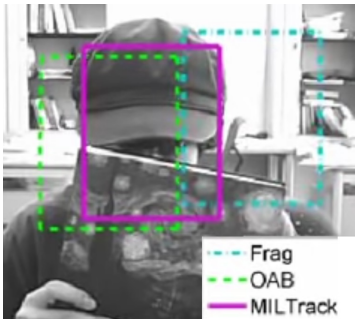
(b) Two subject that can vary their appearance very quickly.

(c) Tracking deals with objects, humans, and even animals.

(d) The limbs disappears.

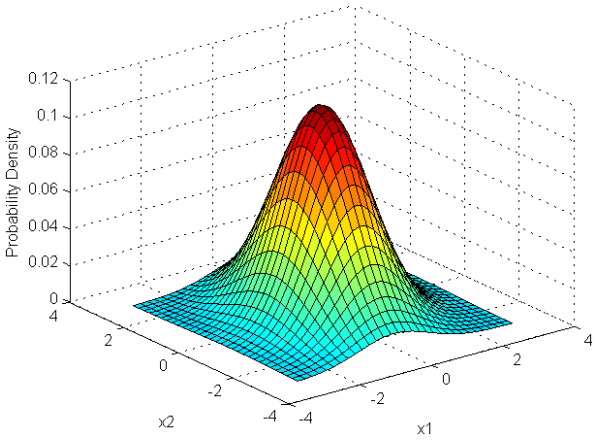


(e) A soccer match action where all the players are tracked.



(f) A face partially hidden by a hat and a book.

(g) The new location is close to the older one.



(h) The probability of where the new location is respect a multivariate normal distribution.

(i) A motorcycle running on a street, is totally hidden by a tree for multiple frames.

Figure 3.1. Some visual examples of the tracking challenge conditions.

- The requirements for this thesis, except the traditional ones, are:
  - Long-time occlusion. In our scenario where the robot physically follows a

person, the leader can disappear behind a corner and will be hidden as long as the robot reaches it.

- Real-time processing. We have decided that to understand the movements in the real environment is sufficient a processing speed of 5 FPS.
- Long-term video. The algorithm is designed to last for a very long period, no explicit bounds exist since the drift problem has been solved.
- Subject limited to people. We are not interested in following animals or vehicles, even if extending the algorithm to them only require to change the object detectors setup and the internal database of images to train the recognition procedure.

3.2 Principal known algorithms

In this section a set of algorithms that perform well to solve the tracking task is presented. Differently, from object detection, the number of existing methods for tracking is much wider. This happens due to the high variability of the problem. The methods shown below represent a trade-off in terms of speed and reliability.

3.2.1 MIL (Multiple Instance Learning) tracker

The MIL tracker[28][29] is an extension of the older **BOOSTING Tracker**[30], both methods are based on an **online classifier**. "*Online*" means that the classifier is trained "*on the fly*" during the execution of the algorithms and not in advance. This type of training does not allow to use thousands of images but very few. An application of this method is presented in Figure 3.2.
The idea of the online classifier is to trust the initialization of the tracker and to use the initial bounding box as the first training sample. The negative samples are then generated taking rectangles that do not overlap the positive example. The classifier learns during the execution to recognise the tracked subject as positive and the rest as negative.
The frames after the first one are elaborated similarly. The positive subject is searched around the last known position and the classifier assigns a probability to each proposal. The box with the highest score is chosen as positive and it is used to continue the training of the classifier.
The novelty of MIL compared to BOOSTING is shown in Figure 3.3).
Instead of using only the positive sample to fit the online classifier, MIL creates a bunch of bounding boxes proposals around the positive sample, called **bags**. All these boxes should contain the subject and one could even be perfectly centred on it. The training is done with the "Multiple Instance Learning" that takes the bag of positive proposals and selects the best one (the more centred one) to improve the classifier. In the end, the instance is trained with only one box that was chosen starting from a set of good alternatives and not with all of them. The negative samples are then generated as for the first frame.

3.2.2 KCF (Kernelized Correlation Filters) tracker

The KCF tracker[7][33] is an additional extension of MIL tracker.
The key idea is that the sampled images are similar due to the subject that is repeated or thanks to the background that does not change extremely fast compared to the FPS of the video. This repetition of similar patterns can be used to optimize the operations and speed up the computation. The technical improvement comes from the application

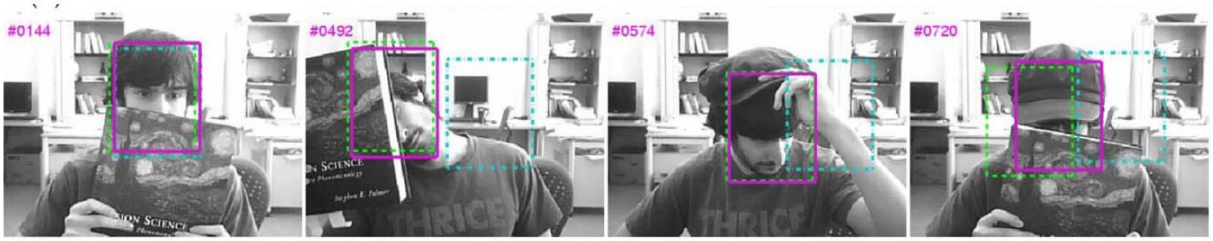


Figure 3.2. A sample of the MIL classifier tracking the face of the author while is partially occluded by a book. MIL tracker is in purple, while **cyan** and **green** are respectively FragTrack[31] and Online Ada-Boost[32].

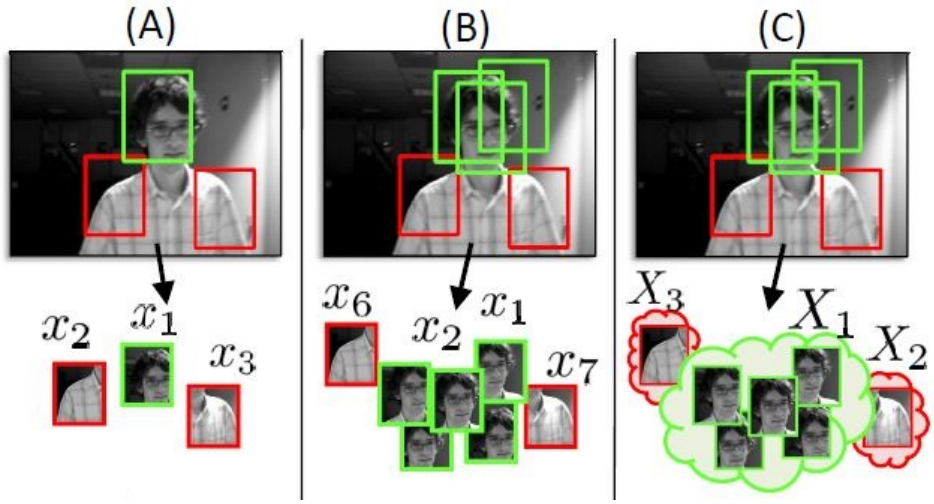


Figure 3.3. A comparison of how positive and negative samples are selected, to train the online classifier. In A the selection of BOOSTING tracker, the positive sample is used. In B a set of proposals generated around the positive are all used to train. In C the selection of MIL tracker, bags of samples are used and only one proposal is extracted and accepted from each one.

of the **FFT (Fast Fourier Transformations)**, which allows to apply the elaboration of images in an efficient manner.
The novelty introduced with KCF allows this algorithm to outperform both BOOSTING and MIL, in terms of accuracy and speed. The weakness of this chain of three methods is the full occlusion. None of them is able to deal with total occlusion that always causes the tracking failure.

3.2.3 Median Flow tracker

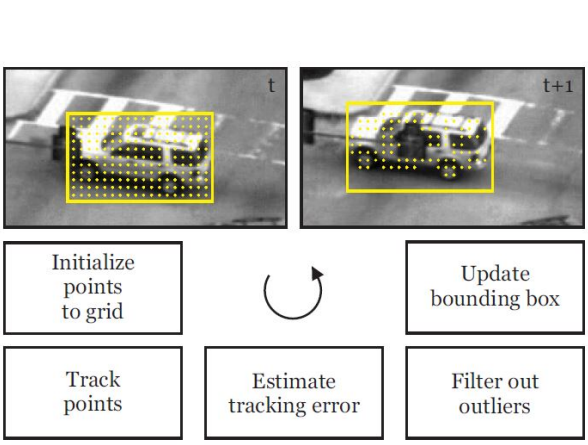
The Median Flow tracker[34] is a reliable method that locates the subject according to its trajectory. The key idea is that the algorithm recognises points in two subsequent frames. These points should be the same physical element in the real space.
The overall procedure is shown in Figure 3.4. The first step (Figure 3.4a) consists of the creation of a grid of points on the initial bounding box, and then the localization of these points in all the future frames. This connection through the frames helps to know the exact motion model of the tracked algorithm. The **Motion Model** (MM) is the combination of actual position (x, y) and velocity. It is defined with the angle or direction of motion (θ) and the module of the speed (s). Essentially knowing how the subject is

moving helps to predict where it will be in the near future.

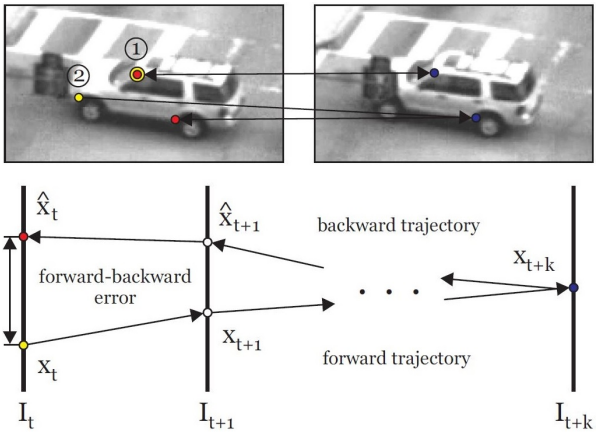
$$MM = ((x, y), (\theta, s))$$

The interaction of the frames works as follows. Every time that a new frame is added, the knowledge of the motion model suggests where the subject can be located. Then, the key points are searched in this new image. Once they are found it is fundamental to check the consistency of the trajectories (Figure 3.4b). Each new point is associated with the most similar already known physical point and vice versa. If this double matching is correct and the two points refer to the same physical element the trajectory is confirmed (point 1 in the figure), otherwise there is a misalignment in the trajectories (point 2 in the figure). In Figure 3.4b point 2 is firstly (forward pass) associated with the front wheel, in the frame after is linked to the back wheel since the other one is hidden from the street signal. The backward pass links the back wheel again on itself. This misalignment of the connection is recognised as an error and so the point cannot be trusted. If no point can be trusted the tracking fails.

Thanks to this double-checking procedure, the Median Flow tracker is a method that is able to well recognise when the tracking has failed to follow the subject. Unfortunately, the requirement to match key points over and over in the frames reduces the capability of the algorithm to manage scenarios where the subject appearance change too much. For this reason, this algorithm is not good for tracking high deformable subjects such as animals and humans.



(a) The first bounding box is divided into a grid of points, and a portion of them are recognised in the next frame. Below, the main steps of the algorithm.



(b) The points found in two consecutive steps are compared to check if the forward and backward trajectories are the same and then it can be trusted.
Point 1 is accepted while point 2 is rejected.

Figure 3.4. The overall procedure of the Median Flow tracker computing forward and backward trajectories to precisely locate the target.

3.2.4 CSRT (Channel and Spatial Reliability Tracker)

The full name of the method is Discriminative Correlation Filter with Channel and Spatial Reliability (DCF-CSR)[8]. Such as KCF (Section 3.2.2) and others, even this method

is based on correlation filters.

A **cross correlation filter** is a technique that aims at localizing into an image the exact position of another one. A representation of the cross-correlation usage done by CSRT is shown in Figure 3.5. In details, the portion of the last frame, delimited by the last known bounding box, is elaborated with multiples correlation filters, each one producing a different output. These elaborations simulate possible changes in the appearance of the subject. Designing good filters is fundamental to well match the variability of the tracked subject and the generated features. The filter outputs are modified images of the last bounding box cropped area.
The output of each filter (an image) is moved along the full picture pixel by pixel (learning stage: Figure 3.5 left) to check which portion of the entire camera view is more probable the subject that we are looking for. The result of this scan is a confidence map that should present a peak in correspondence of the new position of the tracked subject.
All the filters can then be summed up together (localization stage: Figure 3.5 right) to highlight the proposal of each one and comes out with the final response. This response shows exactly where the subject is placed in the new frame.

The characterization of CSRT is focused on the type of correlation filters used, with the idea of using a lot of them and combine the results at the end to produce a more reliable localization. The very high accuracy that this algorithm offers is compensated by the low FPS rate that it achieves.

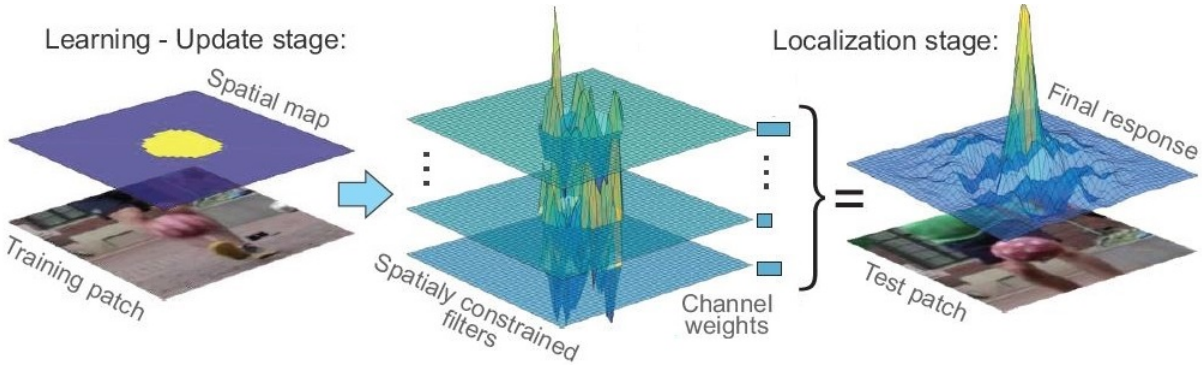


Figure 3.5. The overall procedure of the CSRT algorithm. The learning stage is composed of multiple correlation filters that are applied to the input frame. Each filter produces a confidence map that highlights where the subject should be. The localization stage combines all these confidence maps to produce the final response that precisely locates the subject.

3.2.5 MOSSE (Minimum Output Sum of Squared Error) tracker

The MOSSE algotihm[9] such as KCF (Section 3.2.2) is a method whose strength lies into mathematical smart choices instead of a complex high-level logic like MedianFlow (Section 3.2.3).
The novelty is introduced with a new correlation filter, called MOSSE. It can be applied to the input frames and precisely locate the variations fundamental to understand the movement of the subject.
Despite the original article states robustness against variations in lighting, scale and non-rigid deformations, this algorithm is not so reliable as it appears. On the other hand, the

strength of this method is the extremely fast computations (FPS rate) that outperform all the other trackers presented in this section.

3.2.6 GOTURN (Generic Object Tracking Using Regression Networks)

GOTURN[10] is a tracker based on neural networks. Differently from MIL and KCF (Section 3.2.2), this method is not based on an online NN, such as the online classifier, but it is based on an offline CNN.
An **offline NN** is a traditional NN that is trained on thousands of data: couples of images, in this case, producing the trained model. The process is done in advance, before the effective use, and not "on the fly" while running the tracker. The generated model is used at run time to know how to respond to an input value. The big advantage of offline methods is that the training procedure is the slowest section, for this reason an online classifier could not perform at very high FPS rates.

The general workflow of the tracker is shown in Figure 3.6. The tracker takes as input a frame at a time and always compares it to the previous frame. This choice simplifies and standardizes the input to empower the potentiality of the CNN. However but makes the algorithm to have no chance against total occlusions even for one single frame. The CNN takes as input two squared images cropped from the frames. The crop on the previous frame is a bounding box centred around the last known position with some margins that will contain even the location in the frame afterwards. The current frame is cropped based on the same bounding box, but in this case, the subject is not centred in the square.
Then, both squared images are processed with two independent stacks of convolutional layers, followed by three fully-connected layers. The final output is composed of four values representing the top-left and bottom-right corners of the bounding box. It is centred on the subject in the squared image of the current frame. Some examples of the application of the CNN are shown in Figure 3.7.

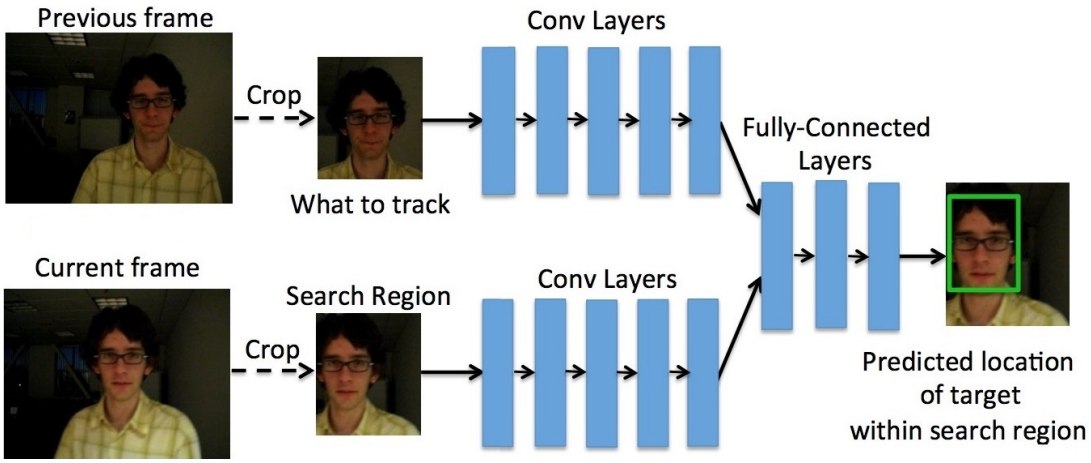


Figure 3.6. The overall procedure of the GOTURN algorithm. On the left, the frames coming from the camera are cropped with the same square, that is centred on the subject in the previous frame (above). The images are elaborated with CNN to produce the output bounding box, that highlights where the subject is in the current frame.

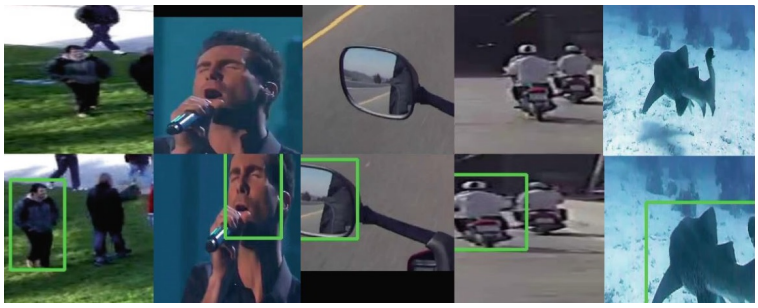


Figure 3.7. Samples of application of the CNN of GOTURN. The two squared images coming from previous (above) and current (below) frame are fed into the network. The answer is the **green** bounding box that locates the tracked subject on the new frame.

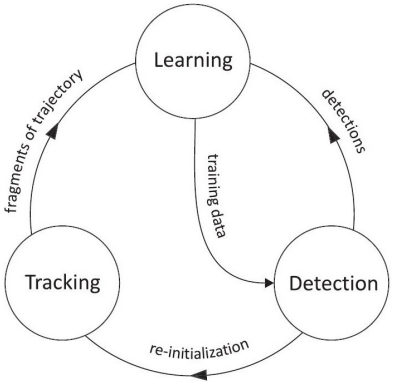


Figure 3.8. The interconnection of the three foundation methods of TLD algorithm.

3.2.7 TLD (Tracking-Learning-Detection)

Differently from all the other methods, TLD[35] aims to be a complete tracker able to deal with extremely complex scenarios. If the previously presented trackers have no chance to manage a long-time total occlusion and a long-term tracking, this method is able to overcome both of the problems. A sample showing potentiality is in the Figure 3.9.
The foundation principle is that TLD is not a single algorithm but it is a combination of three. The interaction of these three parts is shown in Figure 3.8. The key idea to overcome the re-identification problem that occurs after a total occlusion, or the drift problem that happens along a large video, is to often re-initialize the tracker. In fact, the tracking (**T**) of TLD aims at managing short-term video clips. When a small problem occurs the detection (**D**) tries to locate the subject back again. While these two situations take turns, the learning procedure (**L**) extracts the key elements that recognise uniquely the subject and understand how to precisely locate it inside the frame.

The trade-off to use this extremely flexible structure is paid with a not high FPS rate. However, the biggest problem of this method is the huge quantity of false-positive predictions. The learning procedure starts with a few samples, meaning that errors in the beginning phase can occur easily. The failure in the first phase causes wrong learning that will produce more and more errors later on.
Despite the good potentialities, this algorithm is not reliable for the traditional tracking task.

Figure 3.9. A sample that shows the potentialities of the TLD algorithm, dealing with a total occlusion while tracking a motorbike with a dramatic change of size. The **red** dot means that the subject cannot be found.

Algorithm	MIL	KCF	MedianFlow	CSRT	MOSSE	GOTURN	TLD
FPS	9	38	40	15	56	20	10

Table 3.1. An overview of the FPS rate of the trackers described. The performances were measured on an Intel Core i5 CPU and on an Nvidia Jetson TX2 GPU. The speeds measured are almost the same.

3.3 Which tracker could be chosen

In the previous section we have presented seven different tracking algorithms, each one with his own strengths and weaknesses. These methods are the ones that were explored during this thesis work, but many others exist.
The tracking task that we aim at solving is a long real-time sequence with long-lasting total occlusions, as explained in Section 3.1.4. This task is solved with a combination of detection, tracking and recognition, so the internal tracking challenge is much simpler.
The requirement is to solve the baseline tracking task (Section 3.1), and to deal with changes of shape and partial occlusions. Considered the explained methods and known their speed performances (shown in Table 3.1), here it is what we have chosen. Note that the underlined methods are the best trade-off choices.

- **MIL tracker** is not a good choice because it runs at few FPS and a newer version that outperforms both its speed and accuracy exists.
- **KCF tracker** is the new version of MIL. It is one of the fastest methods and it is also reliable. It suffers the rapid change of appearance and, less important for our scenario, it does not manage total occlusions.
- **Median Flow tracker** works well only on not deformable or rigid subjects. Since we are dealing with humans it is a bad choice.
- **CSRT** is the most reliable method, it could even manage short total occlusions. Despite the low FPS rate is a great choice, in fact, our goal is a tracker running at around 5 FPS that is way less than the speed of CSRT.
- **MOSSE** is focused on pure speed. It is not the most reliable method but it can be a good choice to try to reach the highest FPS rate for the general challenge.
- **GOTURN** is a reliable method, running at a good FPS rate. At the moment it is not integrated in this project but it can be a great choice for future improvements.
- **TLD** is a too complex method. It is able to solve long total occlusions but easily fails on simpler scenarios, by proposing a lot of false positives. It is not reliable at all for our purpose.

Note that the principle proposed in this thesis is similar to the one of which TLD is based. The integration of tracking and detection linked together with a third procedure: learning in case of TLD and people recognition for this project. Both algorithms aim at solving the total occlusion problem and the drift problem with a reinitialization of the tracker performed with an object detector. The key difference is the existence in this thesis of the *slow start* phase presented in Section 5.2.

4 People Recognition

The goal of this chapter is to show which types of techniques could be used to understand, given two images of a human, if the represented person is the same or not.

4.1 Problem definition

The **people recognition module**, also known as the **people identifier module**, has the role of connecting, in a consistent way, the two remaining modules: the detector and the tracker. The tracker works for the majority of the time, following the **main subject**, which is called **leader**. When it fails or after a certain amout of time, the detector locates all the people in the incoming frame. Finally, the recognizer has to answer a simple question:

> *Is this person the leader?*
> Given an image cropped on the bounding box of a person and a set of other images with the same characteristics and containing various people labelled with names the recogniser has to chose which is the name of the unknown person.

In our specific scenario, we are interested in understanding whether a bounding box contains a representation of the leader or not. For this reason the label can be considered with only two values: "*positive*" or "*negative*", "*leader*" or "*other*".

4.1.1 Video surveillance application

This challenge is extremely important in the video surveillance field. In fact, one of the most common application is to reconstruct where a person was seen during a certain time slot. The video surveillance application is a little bit different from the one in which we are interested:

> Given a dataset of images of people, and a query containing an unknown image of a person all the images that contain the same person of the query are extracted from the dataset.

From a high-level point of view, the two problems might look different but are essentially the same. The key idea is to look for all the images that might contain the person of the query. Then, according to the task, it is necessary to retrieve this list of people or use them and all the associated information to understand who is the person of the query. Another difference comes from the acquisition of images. In our scenario, the webcam is mounted on the robot that is moving around. In video surveillance the cameras are often static but several of them can simultaneously collect frames. We took inspiration from both single-camera scenarios[36] and multi-camera systems[37].

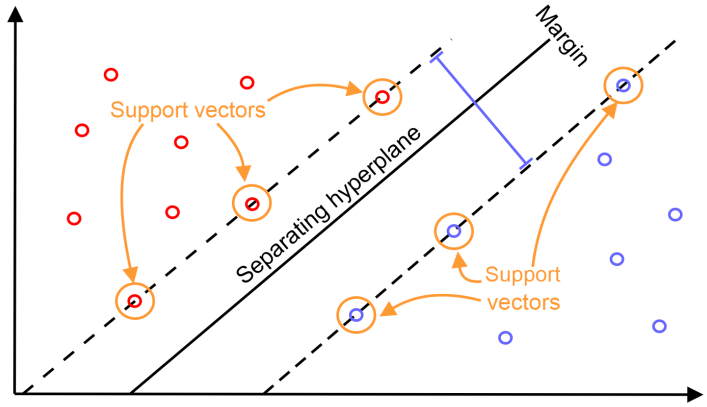


Figure 4.1. The overall mechanics of SVM. The **red** and **blue** classes are divided by the hyperplane that maximizes the margin length, defined as the shortest distance from the closest point on both sides.

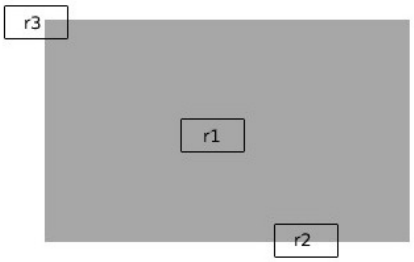


Figure 4.2. Three possible regions that can be used as keypoints. R1 cover a complete flat region. R2 is located on an edge. R3 is placed on a corner, this is the location that can be better recognised.

4.2 Explored methods with bad performances

This section is focused on some unsuccessful technique was explored during the development of the general structure for the project.

4.2.1 Offline algorithms

An offline algorithm (explained in Section 3.2.6) is a great choice for training methods able to judge situations quickly. In this case, we have considered an **SVM (Support Vector Machine) classifier**.
This method is a supervised learning model based on a linear separator. The essential idea is to find the line[1] that divides the dataset into two sections, where respectively the two classes of interest lie (a visual representation is given in Figure 4.1). The method could work if each image of a person is converted into a point, as explained in Section 4.3.1.
However this approach has an essential problem. Offline methods require prior knowledge of the context of application to train ad hoc models. In our case, the element to know is "*Who is the leader?*". Based on this information the dataset can be split into two classes, the images representing the leader and the other ones, then the SVM classifier can be trained. The result is a model that always recognises one person, the same person all the times.
Unluckily, this prior knowledge is not given. The algorithm should recognise all types of people after seeing them for a few seconds. This is the context where an online algorithm can be wisely used.

4.2.2 Key points matching

The key points, or **feature points**, matching is a technique that compares two images and tries to recognise which are the common elements of the two. It is widely used to find small patterns in complex images, understand the variation of orientation, stitch panorama views and judge if the subject of two pictures could be the same.
This technique is based on key points. These elements are positions in the image that can be easily located and identified as unique. In Figure 4.2 are shown three regions, a flat area, an edge and a corner. The most recognizable point is the one placed at the

[1] A line in 2D problems, that is converted into a hyperplane in N-dimensional scenarios.

corner. If an object appears in two images a key points should be recognised in both the appearances.

Technique definition and known algorithms

This technique is divided into two modules:

- **Feature localization**: this module, working on a single image at a time, is internally divided into:
  - **Feature detection**: given the raw image the goal is to identify all the corners that can be easily re-identified in another image.
  - **Feature description**: a small area containing few pixels cannot be easily matched. So, the descriptor needs to take each identified key points and normalize them in order to be independent from the image conditions like illumination and orientation. These key points enriched with additional information are the feature points.

  A visualization of the feature localization is shown in Figure 4.3a.
  To solve this task several methods were used[2]:
  - SIFT (Scale-Invariant Feature Transform)[38] is able to manage both rotations and scale variation of the feature points.
  - SURF (Speeded-Up Robust Features)[14] is the advanced version of SIFT with all its pros and, in addition, has a lower processing time and produce an extremely high number of feature points.
  - ORB (Oriented FAST and Rotated BRIEF)[39] is the only algorithm of the three that is not patented. It is a combination of two sub methods:
    - ➤ FAST (Features from Accelerated Segment Test)[40] is a method completely focused on pure speed.
    - ➤ BRIEF (Binary Robust Independent Elementary Features)[41] is an algorithm extremely sensitive to noise, it uses Gaussian kernels to remove it.
- **Feature matching**: the third module of this technique works on two images after the feature points were computed. The goal is to match the features of the two images together. A match can only be one by one. A feature point should have exactly one corresponding point in the other image and vice versa, to be a correct and reliable connection. If a point is linked to more than one point on the other image, it means that the connection works with multiple inaccurate key points hardly recognisable.
  In Figure 4.3b is presented an example working on two similar images. All the correct matches, drawn as **green** lines, can be used to assign a probability that represents how likely the two images contain the same "object". Moreover the matches can also be used to understand which type of 3D transformation can be applied to an image to convert it into the other one.

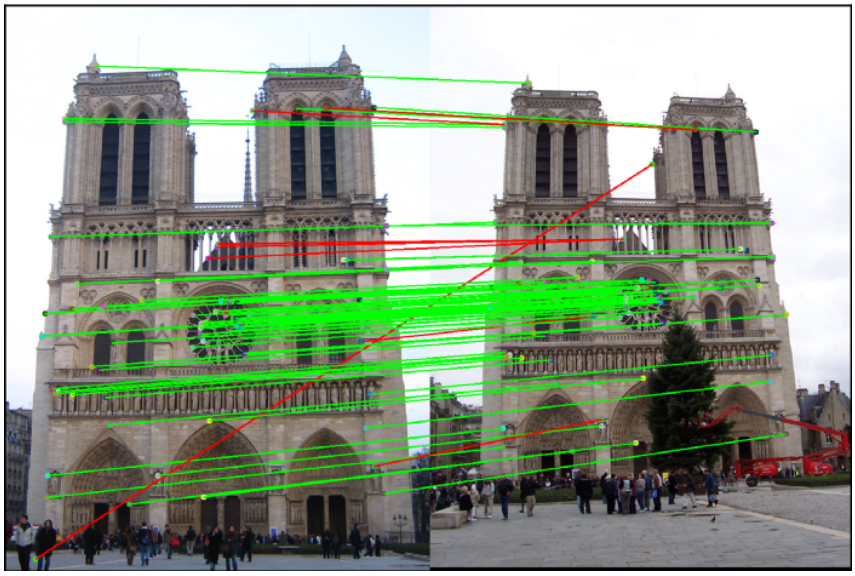
  The existing algorithms for the matching are:

[2]Note that since the mechanics of the methods are similar and these techniques has not actually been implemented in the thesis the algorithms reported here are not explained.

– BFM (Brute Force Matcher). This is the naive solution. All the possible combinations of points of the two different images are being tried. Obviously it is a precise method because it does not work under assumptions but if the number of key points is high this method is infeasible.

– FLANN (Fast Library for Approximate Nearest Neighbors) matcher[42]. Differently from BFM, this method is optimized to work even with a large number of features. The optimization is based on the local search of K-Nearest Neighbors.

(a) The feature localization module applied to a Tesla. The key points are spread along the edges of the car and the background. Instead, the road has no points, because the pattern is often repeated, hence it is not reliable.



(b) The feature matcher is used to align two images of Notre Dame de Paris. Since the images are very clear a lot of points are paired correctly (**green** lines), while a few of them are not (**red** lines).

Figure 4.3. Examples of the two modules of the key points matching: feature localization and matcher.

Key points matching applied to people

The initial idea was to apply the key points matching to people. The tests were done on the **PRID450 (Person Re-IDentification)**[43] dataset that contains thousands of images of cropped people walking outdoors. The dataset is constructed with multiple shots of the same person in different moments and prospectives.
The idea has two big problems:

- The images cropped around the people have a very low resolution. The result is that the details that could distinguish a person from another one cannot be visible, or better cannot be recognised.
- Humans present a high deformable-body, with a surface (clothes) that continuously change aspect. Instead, the key points matching is designed for a pattern that is repeated often and clearly. The consequence of this, is a matching that works as if it was random.

In Figure 4.4 are shown some examples that visually demonstrate the unreliability of this technique applied to humans.

Figure 4.4. Some matching samples show that key points matching easily fails under the conditions previously presented. There are multiple wrong aspects: people who present no key points, objects such as bags that have plenty of features, matches that connect completely different parts of the body like shoulders with legs. Even with the same subject with almost the same position (bottom-left couple of images), the algorithm fails with most of the points. The exception is the bottom-right image that has a perfect matching, but the two pictures are exactly the same one. Therefore it is not a reliable test.

4.3 KNN (K-Nearest Neighbors) with images into N-dimensional space

Since that offline algorithms cannot be used, the official solution is based on online algorithms. The chosen method is KNN (K-Nearest Neighbors) classifier. It is a widely used

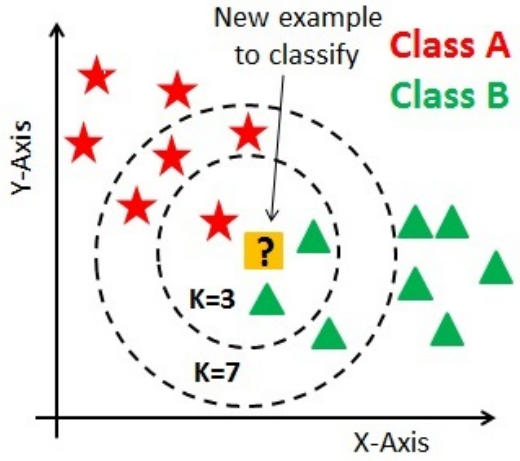


Figure 4.5. Example of the mechanics of KNN. The new point (question mark) will be classified as B if the 3 nearest neighbours are considered. It will be classified as A if K=7 is used.

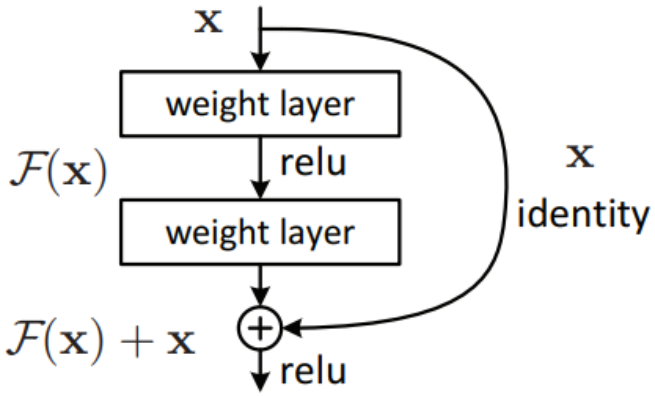


Figure 4.6. The residual block of ResNet. The identity connection allow to the output of the precedent level, to skip two convolution layers. The result $f(x) + x$ is the combination of the convolutions $f(x)$ combined with the identity x.

method that is based on the proximity of points into space. Where each point has a class. When a new point is classified, the K nearest known points are found and the class for the new incoming is chosen according to the majority of classes of the K selected points (an example is shown in Figure 4.5).
About this algorithm, some aspects should be considered:

- KNN is a native online method. The difference between online and offline is about computing the training in advance or not, but KNN has no training phase. The training only requires to store the known points with the associated labels and this is done with almost no cost.

- The elaboration works with points and not images, this remarkably reduces the computational classification cost.
  On the other hand, an image should be collapsed to a point, meaning that a good representation should be used to not lose important information.

- The classification steps should compare the new cropped frame, transformed into a point, with all the other known points. So, a very high number of stored points will slow down the execution. In our real-time scenario, this will occur only if the tracking will last for an extremely long period, but this will not happen.

To apply KNN, the challenging task of converting images into representative points should be solved.

4.3.1 Image classifiers for representative points from images

We have chosen to use CNN to create the representative points that should describe an entire image. Unluckily, it does not exist an explicit field of study that aims to create this kind of points. The solution relies into the adaptation of a widely explored machine learning challenge called **image classification** (Figure 2.2). The goal is to predict which kind of elements exist inside the picture.
The image classifiers based on CNN generally work as follows:

1. The input picture is resized to standard dimensions. In addition, colours and lights are normalized.

2. The image is elaborated with multiple blocks of convolution layers.

3. All the features extracted with the convolutions are collapsed, with a "*flatten*" operation, into an array of thousands of elements.

4. In the end, this vector is eventually reduced and the final predictions, one for each output class, are generated.

The goal is to generate a point from the input image of CNN. The fourth part collapses all the elaborated information into predictions that vary according to the context of the application. Instead, the third level produces an array: a list of N numbers that can be seen as coordinates of a point into an **N-dimensional space**. KNN works independently from the space dimensions, so, it does not matter how many features are produced by the CNN based algorithms.

The classifiers chosen are the DNNs (Deep Neural Networks) **GoogLeNet** and **ResNet**. A DNN has the capability to produce better results than a NN. On the other hand, the huge number of parameters used should be tuned during the training phase. This calibration of the values is extremely hard on small datasets due to the **vanishing gradient problem**. Therefore, both classifiers introduce a novelty aiming at solving the problem connected to the depth of the network.

ResNet (Residual Network)

ResNet[5] is built on the idea of "skip blocks of layers". The residual block is shown in Figure 4.6.
A "*plain CNN*" has convolutions stacked one after the other, in this case, there is an additional element: the **identity connection**. It means that no filters are applied and the input of this block is shifted two layers down. This connection is used to propagate the information deep into the CNN without modifying them. The advantage is that the input image is preserved through the network and it is not affected by an elaboration that lasts for several layers (more than 100). This novelty is very important for small training sets that are not able to fine-tune all millions of parameters of the DNN.
The authors of the original paper has created multiple models characterized by different depths, for this thesis we have chosen ResNet50[44]. This model produces a representation point in **2048 dimensions**.

GoogLeNet (Google Le-Network)

GoogLeNet[6] is based on a new convolution scheme called **Inception module**. The name and the goal of the module come from the quote "*We need to go deeper*"[3].
The naive version of the inception module parallelizes three different convolution filters (1x1, 3x3, 5x5) and a *max-pooling filter*. This special elaboration reduces the depth of the CNN while preserving its potentiality. On the other hand, the number of parameters is still huge. The official version of the inception module (Figure 4.7) stacks each filter (3x3, 5x5 and max-pooling) with a 1x1 convolution to reduce, by a factor of 10, the overall

[3]Quote of the film Inception (meme).

number of parameters.
We have used the model[45] proposed together with the paper. This GoogLeNet implementation produces a representation point in **1024 dimensions** (half of ResNet50).

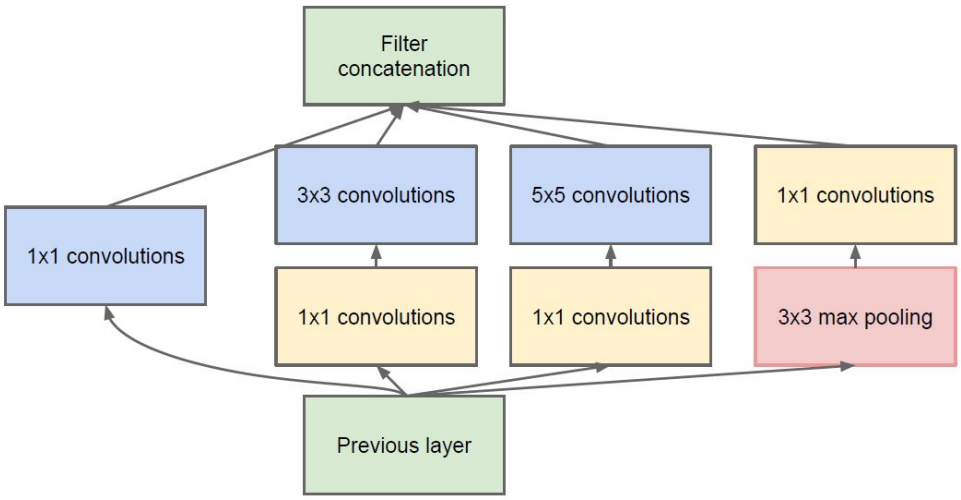


Figure 4.7. The inception module presented with the GoogLeNet model.

SSP-ReID (Saliency-Semantic Parsing Re-IDentification)

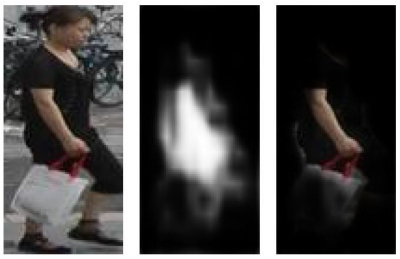
SSP-ReID[46] is a useful technique specifically designed to improve the potentialities of the image classifiers when applied to bounding boxes of people. This method is ideated to create a representative array of features.
The model is based on a "*CNN backbone*" that can be selected from a wide set of pre-existing CNNs such as ResNet or GoogLeNet. The improvement comes from additional pieces of information that are processed together with the backbone.
The extra information generated (shown in Figure 4.8) are:

- **Saliency** (Figure 4.8a): this technology aims at isolating all the pixels that appear at "first glance" to the human eyes. The set of these pixels are the one that may highlight some key aspects of a person such as a bag, the clothes or other.
- **Semantic Parsing** (Figure 4.8b): the body of the person is divided into 5 sections and these are elaborated separately. Each body part can have its own characteristics. The 5 sections are: head, upper body, lower body, shoes and complete body.

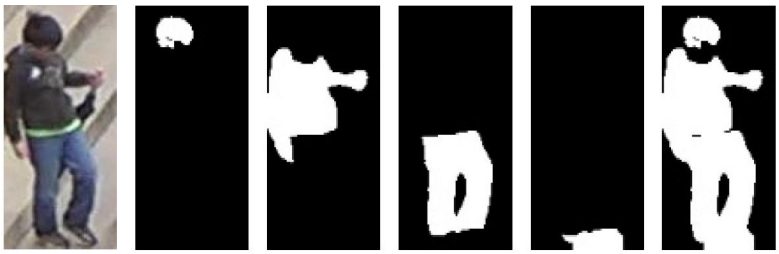
At the moment this method is not integrated into the thesis project but it can be a great choice for future improvements.

4.3.2 Examples of KNN applied to people recognition

The intuition of using KNN has been tested on the **Market1501**[47] dataset. Similarly to the PRID450 dataset, also this one contains sets of images captured from different perspectives and in different moments of hundreds of people. Each real person has it is own ID so different images of the same subject are associated to the same ID.
Some of the results of the experiments are shown in Figure 4.9. This elaboration is done by selecting two small datasets, one made of 18 images of 2 people and the other one made



(a) The saliency elaboration, applied to the woman, highlights the presence of a bag.



(b) The semantic parsing elaboration divides the body of the child in the 5 sections: head, upper body, lower body, shoes and complete body.

Figure 4.8. Examples of saliency and semantic parsing elaborations.

of 99 images of 11 people. For each person were used 9 pictures. KNN was "trained"[4] with the representative points extracted from the images of the datasets. Then, the queries (images of people) were used to retrieve the most similar people. This was done by generating the representative points of the queries and for each one, the K closest points are retrieved together with the associated images.

It is important to focus on the ratio between correct and wrong responses, **green** and **red** respectively. In the test elaborated with ResNet50 (Figure 4.9a) there are almost 50% and 50% of wrong and correct responses. In the test elaborated with GoogLeNet (Figure 4.9b) there is only one false prediction over 14. Despite the different algorithms used the results are independent of them.
In fact, the key difference is that in Figure 4.9a there are 11 classes, so $9 * 1 = 9$ samples of the correct person and $9 * 10 = 90$ samples of the wrong one. Whereas in Figure 4.9b there are only 2 classes so 9 samples against 9. This different ratio between positive and negative training examples affects the result of the predictions.
In this thesis, we are dealing only with 2 classes: the leader and the other people. Hence, we are interested in the scenario with 18 images that works extremely well.

Lastly, for the integration of people recognition, with the tracking and detection modules, we only need to know if the query belongs to a class or to another one. This choice is based on the most likely class on the first K[5] nearest neighbours of the query, so if the majority is **green** or **red**.

[4]The training of KNN consists of storing data and nothing more.
[5]In case of 2 classes, an odd number is chosen to be the value of K.

(a) KNN applied to images elaborated with ResNet50. The training was done with 11 real people and 9 images of each one, for a total of 99 pictures. Here are shown 3 queries with the 8 most similar people.

(b) KNN applied to images elaborated with GoogLeNet. The training was done with 2 real people and 9 images of each one, for a total of 18 pictures. Here are shown 2 queries with the 7 most similar people.

Figure 4.9. In this picture are shown queries computed on the KNN classifier that has pre-processed small datasets of images of people. The query (top-left bounding box with **blue** contour) is used to extract from the database the most similar 7/8 pre-analysed people. The **green** contour means that the extracted person is correct, whereas if it is wrong the **red** is used.

5 Solution

This chapter is focused on the integration of the three modules of this thesis: detection, recognition and tracking. The explanation is based on the flow of the code, with special attention at the choices taken and implementation details. A minimalistic scheme that explains the flow of the code is shown in Figure 5.1. While the source code of the entire project is available on github[1].

5.1 Wrapper function: follow

The code is entirely managed with a single class called **Follower**. This structure only requires to be initialized and then called a "*follow*" method (Algorithm 5.1) every time a new frame needs to be processed.
The class internally loads a new frame *(line 2)* from the webcam or, optionally, from a stored video. Both the sources can be used in "real-time". In fact, if the video is used some frames are internally discarded to simulate the loss of images due to a slow processing rate. Therefore the class *Follower* can be analysed with both real-time tests and also recorded experiments. This feature is useful to replicate scenarios where the code has failed.

The wrapper method *follow* has only one task. It measures the elapsed time from the beginning of the tracking *(line 4)* and, according to this parameter, the *slow start* phase (Section 5.2) is executed *(line 5)* if less than X seconds are gone. Otherwise, the *leader tracking* phase (Section 5.3) is called *(line 7)*.

```
1 follow() -> position:
2     frame = grab_newFrame()
3
4     if elapsedTime() < phase1_length: #phase 1
5     |     position = slowStartPhase(frame)
6     else:                             #phase 2
7     |     position = leaderTrackingPhase(frame)
8
9     return position
```

Algorithm 5.1. It is the pseudocode of the wrapper function that should grab the new incoming frames and redirect them to the first or second phase according to the time elapsed from the tracking begin.

5.2 First phase: slow start

This function is a novelty that we have chosen to introduce to empower the performances of the overall algorithm. The pseudo code of this method is provided at Algorithm 5.2 while an example is shown in Figure 5.2.
This project, differently from the traditional trackers and similarly to TLD (Section 3.2.7), is based on an online learning classifier. Hence this phase is designed to immediately train

KNN a little bit. This first generated knowledge will be then increased in the *leader tracking* phase. KNN can classify the representative points that come from the bounding boxes of people in new frames, only according to other representative points previously archived. The *slow start* phase is used to collect all the bounding boxes used to create the set of positive representative points into the N-dimensional space of KNN.
This elaboration works basing on the assumption that during this first phase "the leader is the most important visible person". The *slow start* computes the detection of the visible people *(line 3)* and if one or more exist *(line 5)*, the leader is chosen as the bounding box, with the biggest area *(line 6)*. The X and Y pixel coordinates are computed to be retrieved *(line 7)* and the leader's box is given to KNN *(line 8)*. Simultaneously, a negative sample is randomly picked up from a database and it is also given to KNN *(line 9)*. This double fed is done to train KNN with a balanced number of positive and negative samples, in order to exploit the potentialities shown in Figure 4.9b. The negative samples come from a custom version of the Market1501 dataset[47], called **NegativePeople**, that we have created ad hoc for this purpose.

The *slow start* phase is executed for a period that can last from 3 up to 20 seconds or more. The value of this hyper-parameter influences the size of samples known by KNN when the official track starts. The longer this phase the better is. The default value is set to 5 seconds. Alternatively, another possibility is that the first phase can be interrupted after X points are given to KNN, but this does not guarantee a precise slot of time so this variant was discarded.

```
1  slowStartPhase(frame) -> position:
2      position = defaultPosition
3      boxes = detector.detectPeople(frame)
4
5      if len(boxes) >= 1:
6      |    box = biggestAreaBB(boxes)
7      |    position = getPosition(box)
8      |    knn.addPositive(box)
9      |    knn.addNegative(pickOneNegative())
10
11     return position
```

Algorithm 5.2. It is the pseudocode of the first phase. The function *slow start* computes only detections in order to train the KNN people classifier.

5.3 Second phase: leader tracking

This function is the core of the entire project. The overall scheme of the pseudocode is shown at Algorithm 5.3. In addition, in Figure 5.4 there is a sequence of frames taken from a video clip where the robot is following the leader while it is shortly occluded twice by another person.
This phase works as follows:

- The detection is performed *(line 7)* only one time every X frames *(line 5)*(details in Section 5.3.1) and if the position of the leader is unknown *(line 6)*.
  The position is unknown immediately after the *slow start* phase *(line 2)*, after the end of the tracking *(line 29)* and after a failed detection *(line 17)*.
  A sample detection over two people is shown in Figure 5.3.

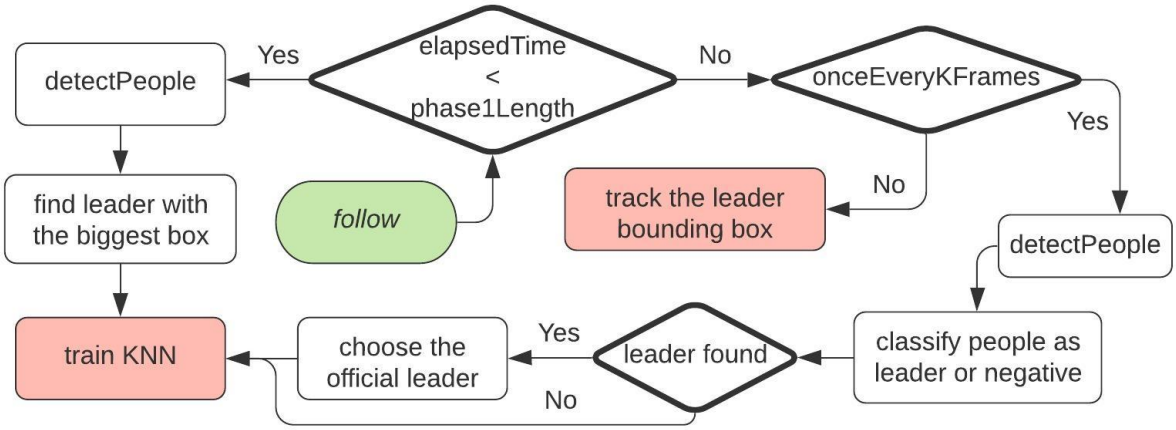


Figure 5.1. The flow of the code presented in the three pseudo-code functions is summarized in this scheme. The **green** region is the starting point that is executed for every frames. The **red** regions represent the ending points of the flow.

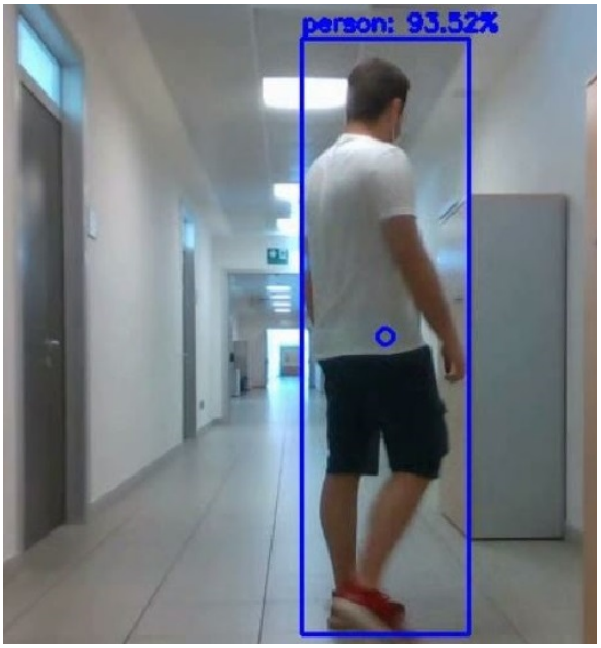


Figure 5.2. Frame of the *slow start* phase where the detection only is working. The leader has a **blue** rectangle, and eventually other people are in **gray**.

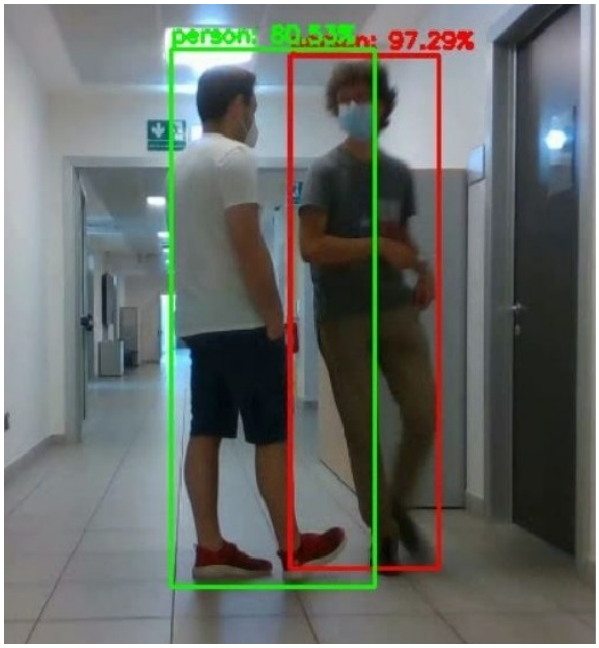


Figure 5.3. A perfect detection where the two people are correctly recognised as leader (positive=**green**) and random person (negative=**red**).

- All the detections are elaborated *(line 10)* to check if they are close to the last known position *(line 11)* and so it can be kept or not (the drift optimization details are in Section 5.3.2). In addition, the KNN classifier is used *(line 12)* to accurately understand which detections contain the leader and which do not (errors tolerance explained in Section 5.3.4).
  Then, the positive predictions are stored for further controls *(line 13)*, while the false ones are added to KNN *(line 15)*.

- All the boxes that seem to contain the leader are further analysed *(line 17-18)*. The official prediction is chosen as the closest feasible box to the last known position

(line 19) (more details in Section 5.3.3). Based on this final choice the tracking is re-initialized *(line 20)*.

- The new position is computed *(line 22)* and the elaborated bounding boxes are stored into KNN according to their content *(line 23-24)*. If only one detection was initially found *(line 25)*, and it was the leader, KNN is fed with a negative sample *(line 26)* coming from the NegativePeople dataset.

- After the initialization, the tracking is updated with new frames, one at a time *(line 30)*. The retrieved bounding box, once converted into a position *(line 31)*, is returned to the *follow* function, both for detection and tracking *(line 33)*.

Special conditions and key aspects to focus on follow in the next sections.

```
1  leaderTrackingPhase(frame) -> position:
2      static stopDetections = False
3      position = defaultPosition
4
5      if onceEveryKTimes(10)
6      |   and not stopDetections: #detection
7      |    boxes = detector.detectPeople(frame)
8      |
9      |    boxesOfleader = []
10     |    foreach box in boxes:
11     |    |    if checkDriftProximity(box) #drift optimization
12     |    |    |   and knn.classify(box)==positive: #recognition
13     |    |    |     boxesOfleader.add(box)
14     |    |    else:
15     |    |    |    knn.addNegative(box)
16     |
17     |    stopDetections = (len(boxesOfleader) > 0)
18     |    if stopDetections:
19     |    |    boxOfleader = pickClosestPosition(boxesOfleader)
20     |    |    tracker.initialize(frame, boxOfleader)
21     |    |
22     |    |    position = getPosition(boxOfleader)
23     |    |    knn.addPositive(boxOfleader)
24     |    |    knn.addNegative(boxesOfleader except boxOfleader)
25     |    |    if len(boxes) == 1:
26     |    |    |    knn.addNegative(pickOneNegative())
27     |
28     else: #tracking
29     |    stopDetections = False
30     |    box = tracker.updateRegion(frame)
31     |    position = getPosition(box)
32
33     return position
```

Algorithm 5.3. It is the pseudocode of the second phase. The function *leader tracking* runs alternatively the detection and tracking modules to constantly know the position of the leader.

Figure 5.4. A sequence of images representing a video clip while the robot is following the leader. The coloured rectangles in the images have different meanings. **Green** is the detection recognised as the leader. **Red** is a detection recognised as a not important person. **Black** is the tracker while following the leader.
In this sample the leader is hidden twice and both times the algorithm is able to detect it back again and continue the tracking.

5.3.1 Detection and tracking ratio

It is fundamental to precisely define the alternation of the detection and the tracking along with the execution of the general method. This calibration is a trade-off among processing speed and localization accuracy. As shown in Table 2.1 and in Table 3.1 the processing speed of the two modules is completely different. The detection is much slower compared to the tracking. Therefore, on one hand the highest FPS rate is reached with a tracker only solution, on the other hand, the highest localization accuracy is achieved

with a detection only technique. Note that with only detection and multiple subjects, the leader is identified thanks to the recognition module.
The disclaimer is to choose the minimum FPS rate required and then hope that the accuracy is enough. In our case, we have fixed 5 FPS as a target. To respect this limit the detection is executed once every 10 frames. It is important to remember that if a single detection fails or the leader is not found, the tracker cannot be started. Consequently, from that moment the frames are processed with detection-only. This recover procedure run at low FPS but the leader is momentarily lost, hence, it is not important.

If the high FPS needs to be reached the solution may consist of changing this mechanic. The detector might be started only after the tracker reports that it has lost the leader. This will reduce the number of detections and improve the overall FPS rate. Unfortunately, a lot of trackers are not precisely able to recognize when the tracked subject has been lost, hence the implementation is not straightforward. At the moment this variation of the code has not been written yet.

5.3.2 Drift tolerance optimization

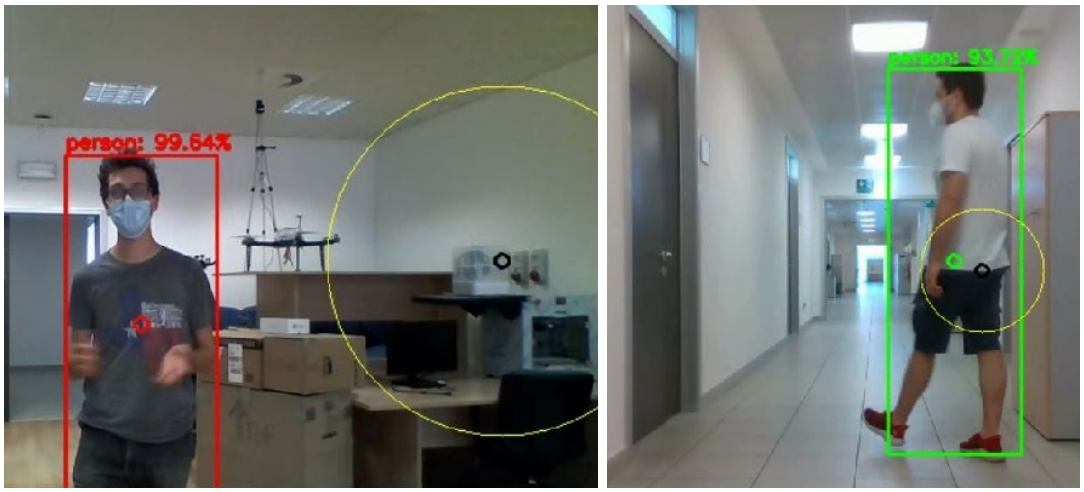
The idea of this optimization comes from two conditions that should be managed. On one hand, the drift problem that is one of the main weaknesses of the trackers, in fact over a long video sequence it can be a huge problem. On the other hand, the proximity assumption (Figure 3.1g) that allows the tracked subject to move only for a limited number of pixels per frame.
These two conditions combined can be used to wisely classify the new detections once the tracker has been stopped. If a new detection is too far away compared to the last known position of the leader, it cannot be the leader itself. Otherwise, the recognition module should be used to normally predict the class of the new bounding box.
We have defined the tolerance of the movement as:

$$d = t \cdot s \cdot \left(\frac{w}{100}\right)^2$$

In the above formula the symbols represent:

- **d** is the distance allowed.
- **t** is the time elapsed from the last correct detection of the leader. It is used to manage the drift problem independently of the ratio between the tracking and the detection.
- **s** is an empirical scale factor, experimentally measured to be around 0.05. It is the hyper-parameter to manage this optimization.
- **w** is the width of the last bounding box during the tracking. It is used to simulate the distance of the leader from the camera. Compared to a far detection, a close leader has a bigger bounding box hence the multiplying factor is greater. This difference is fundamental to manage the fast 2D movement of a close subject.

In Figure 5.5a is shown an example where a person is immediately classified as negative because it is too far away. Instead, in Figure 5.5b the person is inside the tolerance and further analysis with KNN has classified this person as the leader.



(a) The person is classified as Negative because it is outside of the *drift tolerance* circle.

(b) The person is classified as Positive because it is outside of the *drift tolerance* circle.

Figure 5.5. Two samples of the *drift tolerance* optimization represented as a **yellow** circle, centred on the last known position (a small **black** circle).

5.3.3 Multiple leaders corner case

It may happen that there are two people inside the *drift tolerance circle* and both are classified as the leader from KNN. This of course is an impossible scenario in reality (a person cannot be duplicated), hence only one detection can be chosen as the right one. Further analysis can be used but we have chosen a simpler idea: *"The closest subject to the last know position will be the official leader"*.

5.3.4 KNN tolerance against false-negative

The previous section introduces the failure of the KNN classifier that predicts a leader in exceed. This is a false-positive classification. As explained this miss prediction can be managed. Instead, a false-negative classification is a much bigger problem. It happens when the leader is classified as negative.

This wrong prediction is complex because, after the classification, the generated representative point and its new label are fed into KNN. This false-negative represent a point wrongly classified. However, also a false-positive is fed into KNN but if there are multiple leaders further analysis can reduce them. Instead, a leader classified as negative cannot be re-evaluated hence it cannot be converted into a correct prediction, it will be an error forever.

Due to the mechanics of KNN, a small set of wrong classified points can cause a lot of wrong classifications in future analysis. Therefore we absolutely want to avoid false-negative predictions.

6 Conclusions

In this thesis project, a wide variety of algorithms have been explored, implemented and tested. The general flow of the program has been changed often ending up in the version that is presented in the previous pages.

The result is a software application that satisfies all the requirements that were chosen in the beginning. The algorithm is able to be executed in real-time at 5 FPS on not top quality hardware, both on an Intel Core i5 CPU and on an Nvidia Jetson TX2 GPU. The fast execution allows the robot to physically follow the leader in the real environment, while the software is tracking it precisely. Moreover, the tracking can last for several minutes without any problem[1].
The main known weakness is against the false-negative predictions of the people classifier, as widely explained. Hence, according to the *confusion matrix measures*, we can state that our combination of methods, to work precisely, requires a high *recall* value for the people classifier module.

From another point of view, referring to the architecture of the overall project, we have chosen a structure that aims at being modular. During the study of the technologies applied to the project, we have realized that plenty of them can be used in order to solve the tracking challenge proposed in this dissertation. In addition, computer vision technologies are rapidly changing. In fact, the papers with the majority of the methods used in this work, were written in the last few years. Based on the data, these papers were published on average four years ago[2].
This means that in four years this project will probably be obsolete. To avoid this extremely rapid decay, it is fundamental to constantly replace the algorithms that are used as the core of the three main modules: detection, tracking and recognition. A modular structure allows to easily substitute a method without the necessity to change also the others. By doing this, we hope that this thesis project will be useful for more than a few years.

[1] We have tested the algorithm only on the minutes time scale, but there are no substantial reasons that prevent the software to be executed even for hours. It is only a matter of tests that were not done. Hence, we cannot officially state that this algorithm is able to run for hours.

[2] We have measured this value by taking the average of the release dates of the papers of the algorithms that were identified as the best compromise along with all the dissertation. The average year of publication is 2016.

Bibliography

[1] arobottofollowpeople source code. `https://github.com/leopold-lll/aRobotToFollowPeople`.

[2] Dolomiti robotics. `https://dolomitirobotics.it/`.

[3] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[4] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[6] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[7] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *European conference on computer vision*, pages 702–715. Springer, 2012.

[8] Alan Lukezic, Tomas Vojir, Luka ˇCehovin Zajc, Jiri Matas, and Matej Kristan. Discriminative correlation filter with channel and spatial reliability. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6309–6318, 2017.

[9] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 2544–2550. IEEE, 2010.

[10] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *European Conference on Computer Vision*, pages 749–765. Springer, 2016.

[11] Sicong Jiang, Jianing Zhang, Yunzhou Zhang, Feng Qiu, Dongdong Wang, and Xiaobo Liu. Long-term tracking algorithm with the combination of multi-feature fusion and yolo. In *Chinese Conference on Image and Graphics Technologies*, pages 390–402. Springer, 2018.

[12] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[13] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[14] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[15] pyimagesearch. `https://www.pyimagesearch.com/`.

[16] learnopencv. `https://www.learnopencv.com/`.

[17] A. Neubeck and L. Van Gool. Efficient non-maximum suppression. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 3, pages 850–855, 2006.

[18] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[19] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[20] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[21] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[22] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[23] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[24] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1812.08008*, 2018.

[25] Yaadhav Raaj, Haroon Idrees, Gines Hidalgo, and Yaser Sheikh. Efficient online multi-person 2d pose tracking with recurrent spatio-temporal affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4620–4628, 2019.

[26] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[27] Wordnet graph. `https://wordnet.princeton.edu/`.

[28] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In *2009 IEEE Conference on computer vision and Pattern Recognition*, pages 983–990. IEEE, 2009.

[29] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Robust object tracking with online multiple instance learning. *IEEE transactions on pattern analysis and machine intelligence*, 33(8):1619–1632, 2010.

[30] Helmut Grabner, Michael Grabner, and Horst Bischof. Real-time tracking via on-line boosting. In *Bmvc*, volume 1, page 6, 2006.

[31] Amit Adam, Ehud Rivlin, and Ilan Shimshoni. Robust fragments-based tracking using the integral histogram. In *2006 IEEE Computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 1, pages 798–805. IEEE, 2006.

[32] Helmut Grabner, Michael Grabner, and Horst Bischof. Real-time tracking via on-line boosting. In *Bmvc*, volume 1, page 6, 2006.

[33] Martin Danelljan, Fahad Shahbaz Khan, Michael Felsberg, and Joost Van de Weijer. Adaptive color attributes for real-time visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1090–1097, 2014.

[34] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *2010 20th International Conference on Pattern Recognition*, pages 2756–2759. IEEE, 2010.

[35] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409–1422, 2011.

[36] Wiebe Van Ranst, Floris De Smedt, Jonathan Berte, and Toon Goedemé. Fast simultaneous people detection and re-identification in a single shot network. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE, 2018.

[37] Ergys Ristani and Carlo Tomasi. Features for multi-target multi-camera tracking and re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6036–6046, 2018.

[38] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[39] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[40] Miroslav Trajković and Mark Hedley. Fast corner detection. *Image and vision computing*, 16(2):75–87, 1998.

[41] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[42] Marius Muja and David Lowe. Flann-fast library for approximate nearest neighbors user manual. *Computer Science Department, University of British Columbia, Vancouver, BC, Canada*, 2009.

[43] Peter M. Roth, Martin Hirzer, Martin Koestinger, Csaba Beleznai, and Horst Bischof. Mahalanobis distance learning for person re-identification. In Shaogang Gong, Marco Cristani, Shuicheng Yan, and Chen C. Loy, editors, *Person Re-Identification*, Advances in Computer Vision and Pattern Recognition, pages 247–267. Springer, London, United Kingdom, 2014.

[44] Resnet50. `https://github.com/onnx/models/tree/master/vision/classification/resnet`.

[45] Googlenet. `https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet`.

[46] Rodolfo Quispe and Helio Pedrini. Improved person re-identification based on saliency and semantic parsing with deep neural network models. *Image and Vision Computing*, 92:103809, 2019.

[47] Liang Zheng, Liyue Shen, Lu Tian, Shengjin Wang, Jingdong Wang, and Qi Tian. Scalable person re-identification: A benchmark. In *Proceedings of the IEEE international conference on computer vision*, pages 1116–1124, 2015.