Unsupervised Real-Time Visual Tracking Via Support Vector Classification

Giacomo Baggio [*], *Gianluca Giorgini* [†] *and Marco Michielan* [‡] *DEI, University of Padua*
[*]giacomo.baggio@studenti.unipd.it, [†]gianluca.giorgini@studenti.unipd.it, [‡]marco.michielan.1@studenti.unipd.it

Visual tracking has become one of the most promising and extensively studied topic in Computer Vision. Due to real-time constraints, environment conditions, cluttered background and other practical limitations, tracking an object is usually a challenging and nontrivial task to achieve. Following a tracking-by-detection approach (see *e.g.* Avidan's seminal work in [1]), in our work we present visual tracking as a problem of binary classification between foreground (*i.e.* the target object) and background. In this framework we propose an unsupervised tracking algorithm which exploits Support Vector Classification (here considered as a penalization problem), Kalman filtering, to ensure reliability and robustness, and an adaptive threshold, to improve computational performance. The good compromise between adaptivity, robustness and real-time processing of our solution makes it a simple and attractive alternative to the strategies already listed in literature. The comparison with the CamShift algorithm [10], in particular, has shown interesting results and seems to confirm its validity.

Index Terms – Support Vector Machines, Detection-based tracking, Kalman filter, Newton-Raphson minimization.

1. Introduction

The proliferation of affordable high-end computers, the availability of high definition video cameras, and the increasing need for automated video analysis, in the field of video surveillance in particular, have generated a great deal of interest in visual tracking algorithms in the last decades. In this introductory section we briefly review the most striking applications of visual tracking, we try to outline the evolution and recent advances of tracking algorithms and, finally, we describe the proposed algorithm, defining where our approach can be located in the general framework of object tracking.

1.1. Object tracking: statement of the problem and relevant applications

In its simplest form, tracking can be defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene. There are a lot of relevant applications of object tracking. For example it is used in these fields:

- motion-based recognition, that is, human identification based on the movement of an object in the space [12, 20];
- automated surveillance, that is, monitoring a scene to detect suspicious activities or unlikely events [24];
- video indexing, that is, the retrieval of videos in multimedia databases [35];
- human-computer interaction, that is, gesture recognition, eye gaze tracking for data input to computers, etc. [13];
- traffic monitoring, that is, real-time gathering of traffic statistics to direct traffic flow [27];
- vehicle navigation, that is, video-based path planning and obstacle avoidance capabilities [25].

Tracking objects can be complex due to:

- loss of information caused by projection of the 3D world on a 2D image,
- noise in images,
- complex object motion,
- nonrigid or articulated nature of objects,
- partial and full object occlusions,
- complex object shapes,
- scene illumination changes, and
- real-time processing requirements.

One can simplify tracking by imposing constraints on the motion and/or appearance of objects. For example, almost all tracking algorithms assume that the object motion is smooth with no abrupt changes. One can further constrain the object motion to be of constant velocity or constant acceleration based on *a priori* information. Prior knowledge about the number and the size of objects, or the object appearance and shape, can also be used to simplify the problem. Numerous approaches for object tracking have been proposed. These primarily differ from each other based on the way they approach the following questions: Which object representation is suitable for tracking? Which image features should be used? How should the motion, appearance, and shape of the object be modeled? The answers to these questions depend on the context/environment in which the tracking is performed and the end use for which the tracking information is being sought. A large number of tracking methods have been proposed which attempt to answer these questions for a variety of scenarios.

1.2. State-of-the-art of tracking algorithms

Visual tracking is a research topic in constant evolution and trying to delineate a detailed map of visual tracking state-of-the-art and/or a *trait d'union* between different strategies goes beyond the aim of this work (for an exhaustive tractation we refer the reader to [41], [40], [23]). Thus in the following we dramatically simplify the complex dynamic of this state-of-the art by assuming that tracking strategies can be divided into four main categories. These are:

- **Point tracking or prediction-based tracking**: objects detected in consecutive frames are represented by points, and the association of the points is based on the previous object state which can include object position and motion. In this category it can be distinguished:
  - Deterministic methods;
  - Statistical or Probabilistic methods;
- **Kernel tracking or 2-D image region tracking**: kernel refers to the object shape and appearance. For example, the kernel can be a rectangular template or an elliptical shape with an associated color histogram. Objects are tracked by computing the motion of the kernel in consecutive frames;
- **Silhouette tracking or moving blob tracking**: tracking is performed by estimating the object region in each frame. Silhouette tracking methods use the information encoded inside the object region. This information can be in the form of appearance density and shape models which are usually in the form of edge maps. Given the object models, silhouettes are tracked by either shape matching or contour evolution;
- **Online learning tracking**: this approach is based on tracking algorithms that are able to learn continuously, updating and improving incrementally their representations of foreground and background. Generally speaking, these algorithms can be divided into two subcategories:
  - Generative methods;
  - Discriminative methods also termed as *tracking-by-detection* or *object detection based tracking*.

A caveat might be necessary: the aforementioned division is not meant to be tight, in fact some of the tracking algorithms could be placed in more than one group. A graphical representation of this classification is depicted in Fig.1. In the next subsections we review, more in detail, some tracking strategies for each of the categories devised above.

1.2.1. Point tracking

Visual tracking can be formulated as the correspondence of detected objects represented by points across frames. Point correspondence is a complicated problem, specially in the presence of occlusions, misdetections, entries, and exits of objects.

In this category the deterministic methods use qualitative motion heuristics (see [39]) to constrain the correspondence problem. More specifically, these methods define a cost of associating each object in frame $k-1$ to a single object in frame k using a set of motion constraints (usually a combination of constraints related to proximity, maximum velocity, small velocity change, common motion, rigidity, proximal uniformity). Minimization of the correspondence cost is formulated as a combinatorial optimization problem.

On the other hand, statistical or probabilistic methods explicitly take the object measurement and their uncertainties into account to establish point correspondence. This statistical correspondence methods use the state-space approach to model the object properties such as position, velocity, and acceleration. In this case, if we suppose that information representing the object (*e.g.* position and/or velocity) is defined by a sequence of states $\{\mathbf{x}_k; k = 1, 2, \dots\}$ and the sequence $\{\mathbf{y}_k; k = 1, 2, \dots\}$ represents the measurement process, the evolution of state over time is governed by the state-space model,

$$\mathbf{x}_{k+1} = f_{k+1}(\mathbf{x}_k) + \mathbf{w}_k \tag{1}$$

$$\mathbf{y}_k = h_k(\mathbf{x}_k, \mathbf{v}_k) \tag{2}$$

where $\{\mathbf{w}_k; k = 1, 2, \dots\}$ and $\{\mathbf{v}_k; k = 1, 2, \dots\}$ are independent white noises. For the single object case, if f_{k+1} and h_k are linear functions and the initial state $\mathbf{x}_1$ and noises have a Gaussian distribution, then the optimal state estimate is given by the Kalman filter. If the assumption of Gaussianity is not verified and/or the functions f_{k+1} and h_k are not linear, state estimation can be performed using a non linear filtering tecnique, *e.g.* the Extended Kalman Filter or the Unscented Kalman Filter (which typically work well if the posterior p.d.f. $p(\mathbf{x}_k|\mathbf{y}^k)$ is still unimodal) or particle filters (for non-unimodal posterior p.d.f.). Particle filtering, which is based on Monte Carlo integration methods, recently became very popular in Computer Vision and it is worth to briefly illustrate how it works (for the details refer to [22]). In particle filtering, the conditional state density $p(\mathbf{x}_k|\mathbf{y}^k)$ at time k is represented by a set of samples (particles) $\{s_k^{(n)}; n = 1, \dots, N\}$ with

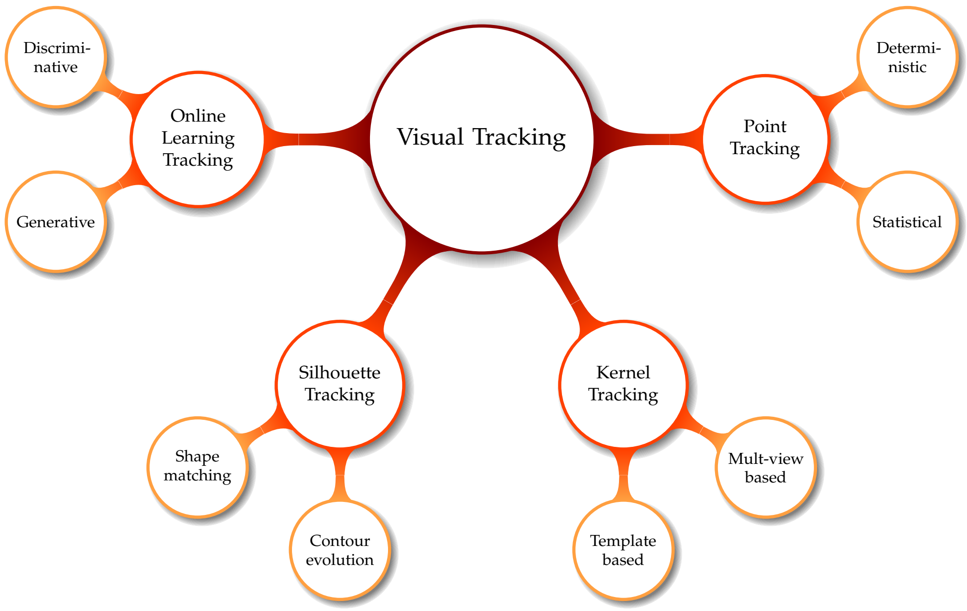


Figure 1: *Taxonomy of tracking methods.*

weights $\pi_k^{(n)}$ (sampling probability). The weights define the importance of a sample, that is, its observation frequency. To decrease computational complexity, for each tuple $(s^{(n)}, \pi^{(n)})$, a cumulative weight $c^{(n)}$ is also stored, where $c^{(N)} = 1$. The new samples at time k are drawn from the set $\mathbf{S}_{k-1} = \{(s_{k-1}^{(n)}, \pi_{k-1}^{(n)}, c_{k-1}^{(n)}); k = 1, \dots, N\}$ at the previous time $k-1$ step based on different sampling schemes (the most common is *importance sampling*). Using the new samples $\mathbf{S}_k$, one can estimate the new object position by $\varepsilon_k = \sum_{n=1}^{N} \pi^{(n)} g(s_k^{(n)}, \mathbf{n})$, where $\mathbf{n}$ is a zero mean Gaussian error and $g(\cdot)$ is a non-negative function used in the sampling procedure. In addition to keep track of the best particles, an additional resampling technique is usually employed to eliminate samples with very low weights (the most quoted ones are *multinomial, residual and sistematic resampling*).

Generally, prediction-based trackers provides robust performance; particle filter tracking, in particular, is used in several applications, see for instance [43], [7]. However there are two main drawbacks in using this approach:

1. point trackers are suitable for tracking very small objects which can be represented by a single point. Multiple points are needed to track larger objects. In the context of tracking objects using multiple points, automatic clustering of points that belong to the same object is an important problem because there is not a discriminative model of the object class;
2. these trackers (particle filters especially) can become impractical and computationally expensive (in particle filtering this can be due to the size of the state vector and the large number of particles).

1.2.2. Kernel tracking

Kernel tracking is typically performed by computing the motion of the object, which is represented by a primitive object region, from one frame to the next. These trackers can be divided into two subcategories based on the appearance representation used, namely, templates and density-based appearance models, and multiview appearance models.

In the template-based subcategory a computationally efficient and very popular approach is mean-shift tracking [15]. The mean-shift tracker maximizes the appearance similarity iteratively by comparing the histograms of the object, Q, and the window around the hypothesized object location, P. Histogram similarity is defined in terms of the Bhattacharyya coefficient, $\sum_{u=1}^{b} P(u)Q(u)$, where b is the number of bins in histograms. At each iteration, the mean-shift vector is computed such that the histogram similarity is increased and this process is repeated until convergence is achieved. The CamShift algorithm [10] is basically a variant of the mean-shift. CamShift applies mean-shift to find the best-matching region for a target, then updates the size of the object according to the zero-th moment, *i.e.* the sum of probability contributions of the current window. Another approach to track a region defined by a primitive shape is to compute its translation by use of an optical flow method.[1] The well-known KLT (Kanade-Lucas-Tomasi) feature-tracker [34], based on the Lucas and Kanade pioneer work [28], iteratively computes the translation $(\mathrm{d}u, \mathrm{d}v)$ of a region centered on an interest point:

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_y I_x & \sum I_y^2 \end{bmatrix} \begin{bmatrix} \mathrm{d}u \\ \mathrm{d}v \end{bmatrix} = \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}, \qquad (3)$$

where $I(x(t), y(t), t)$ denotes image intensity at time t, and $I_\theta := \frac{\partial I}{\partial \theta}$, $\theta = x, y, t$. If the sum of square difference

[1]Optical flow of the image intensity at time t, $I(x(t), y(t), t)$, is a dense field of displacement vectors which defines the translation of each pixel in a region. It is computed using the brightness constraint, which assumes brightness constancy of corresponding pixels in consecutive frames, *i.e.* $I(x, y, t) - I(x + \mathrm{d}x, y + \mathrm{d}y, t + \mathrm{d}t) = 0$.

between the current patch and the projected patch is substantial, the features will be eliminated, because the differences indicate that the selected object is not the same object from the previous frame.

Using template-based strategies, the objects may appear different from different views, and if the object view changes dramatically during tracking, the appearance model may no longer be valid, and the object track might be lost. To overcome this problem, multi-view appareance tracking methods can be used, in which different views of the object can be learned offline and used for tracking. An example of this approach is given by the eigentracking algorithm proposed in [6].

The greatest advantage of kernel tracking relies in his simplicity and real-time applicability. On the other hand, the main problem in these tracking methods is that many of these algorithms (mean-shift and CamShift in particular) fail in the case of occlusions and quick appearance changes, when the color distribution of the background is too similar to that of the target object or when the object moves outside of the kernel search area.

1.2.3. Silhouette tracking

Silhouette-based methods provide an accurate shape description for objects that cannot be well described by simple geometric shapes. The goal of a silhouette-based object tracker is to find the object region in each frame by means of an object model generated using the previous frames. This model can be in the form of a color histogram, object edges or the object contour. Silhouette trackers can be divided into two subcategories, namely, shape matching and contour tracking.

Shape matching approaches search for the object silhouette in the current frame and its underlying philosophy is similiar to that of kernel template-based trackers. In this case, silhouette detection is usually carried out by background subtraction. In [21] a edge-based representation is used as shape matching criterion and the Hausdorff distance is adopted to construct a correlation surface from which the minimum is selected as the new object position.

Contour tracking approaches, on the other hand, evolve an initial contour to its new position in the current frame by either using the state space models or direct minimization of some energy functional. In the paper [5] is proposed an algorithm that computes the motion vectors in the edge of the silhouette iteratively for each contour position using level set representation.

The most important advantage of silhouette tracking is the ability to track objects of various shapes, which can evolve in time. However the complexity of these algorithms are usually high.

1.2.4. Online learning tracking

Recently, online learning tracking methods (and the so-called tracking-by-detection approach, in particular) have gathered momentum in the visual tracking scenario since Avidan's paper on Support Vector Tracking [1]. The peculiarity of this approach relies in its adaptivity, that is, online learning algorithms can handle, in principle, both intrinsic (*i.e.* pose variation, and/or shape deformation) and extrinsic (*i.e.* changes resulting from different illumination, camera motion, camera viewpoint, and occlusion) appearance variations of the tracked object. Indeed, these adaptive methods are able to incrementally update their information on foreground and background representations. These methods can be split into two subcategories, called, generative methods and discriminative methods.

Generative methods have been exploited to handle the variability of a target. These methods learn a model to represent the appearance of an object. This model is then udpated online. Tracking is then expressed as finding the most similar object appearance to the model. It is worth noting that traditional generative tracking methods are trained based on object appearance without considering background information. This approach shares many similarities with the multi-view based kernel tracking approach discussed in §1.2.2.

Discriminative tracking methods instead aim to find a decision boundary that can best separate the object from the background. In these methods a classifier is trained and updated online to distinguish the object from the background. This method is also termed as tracking-by-detection, in which a target object identified by the user in the first frame is described by a set of features. A separate set of features describes the background, and a binary classifier separates target from background in successive frames. To handle appearance changes, the classifier is updated incrementally over time. In other words, the philosophy behind tracking-by-detection is very simple and can be formulated as: optimally discriminate the target object (foreground) from the background at each video frame using ad-hoc binary classification methods, *e.g.* Support Vector Classification (this intuition is graphically represented in Fig.2). Two examples of this approach can be found in [2] and [14]. In the first paper, Avidan proposes to use an ensemble of online learned weak classifiers to label a pixel as belonging to either the object or the background. To accommodate object appearance changes, at every frame, new weak classifiers replace part of old ones that do not perform well or have existed longer than a fixed number of frames. In the second paper, Collins and Liu described a method to adaptively select color features that best discriminate the object from the current background.

Despite the good premises, these trackers present three possible disadvantages:

1. the detection performance is usually a trade-off between the detection rate and the false alarm rate. The missed detections and false alarms provide misleading information to the tracking algorithms;

2. trackers suffer from the drifting problem, *i.e.* amplifying small errors and adaptating to other objects.

3. typically the online training samples are collected in supervised manner; this may not fit well with the real-time constraints.

In Fig.3 it is illustrated a indicative scheme of tracking algorithms performance focused on three fundamental

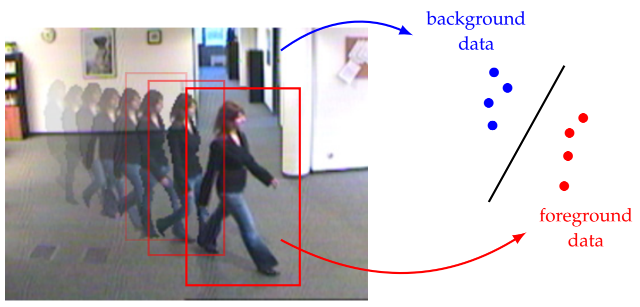


Figure 2: *Tracking-by-detection methods or discriminative methods deal with the issue of optimally distinguish at each video frame between the foreground* (i.e. *the target) and the background exploiting classifiers.*

aspects: robustness, adaptivity and real-time processing. Our solution, partially inspired by tracking-by-detection strategy, aims to solve the "tertium non datur" paradigm in tracking algorithms, providing a balance between the above listed characteristics. In the next section a preliminary description of this strategy will be reported.

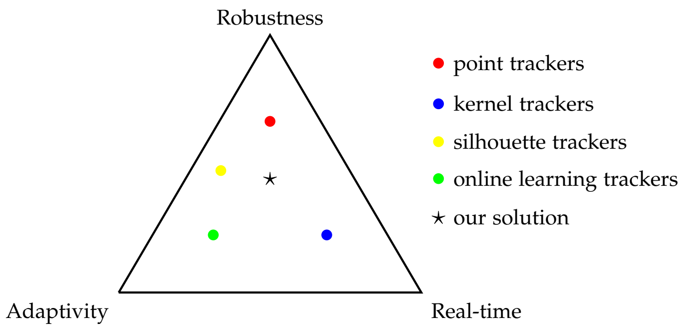


Figure 3: *A graphical representation of general characteristic of tracking algorithms compared to our solution. Our strategy aims to be a good compromise between adaptivity, robustness and real-time requirements.*

1.3. Our contribution

Following previous work in [18] and [3], in our paper we propose and refine an unsupervised tracking method which combines two theoretical tools: linear Support Vector Machines (briefly SVM) theory and Kalman filter. Coupling binary class SVMs with Kalman filter gives the advantage of taking into account the dynamics of the SVM separating hyperplane, consequently increasing tracking algorithm reliability and robustness. Broadly speaking the core of the algorithm can be described as follows (*cf*. Fig.4): the first step consists in the acquisition of a video frame (N pixels of resolution), then, after SVM foreground/background classification, a loss function $\ell_i(\cdot)$ is assigned to each pixel of the frame which represents the error committed in the classification, hence using a minimization algorithm the total loss function $\mathfrak{L}(\cdot) = \sum_{i=1}^{N} \ell_i(\cdot)$ is minimized and a new optimal separating hyperplane is computed; in the final step the Kalman filter gives an estimation of the separating hyperplane whereby next frame pixels will be classified.

Our contribution, presented in this paper, is twofold:

- ✦ we improved the overall computational efficiency of the algorithm:
  - ◇ using OpenCV libraries [9] and C++ programming language;
  - ◇ adopting a dynamic bounding box and an adaptive threshold to discard uninformative data in the classification.
- ✦ we increased the accuracy in detection and the adaptivity of the solution choosing HSV instead of RGB values as tracking features.

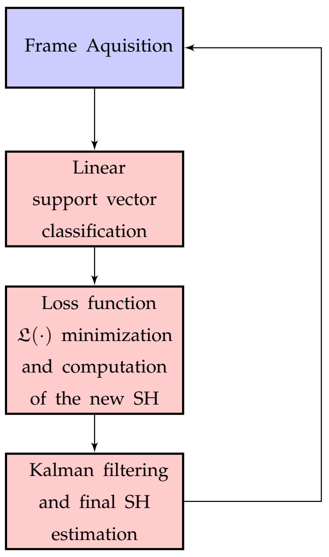

Frame Aquisition
Linear support vector classification
Loss function $\mathfrak{L}(\cdot)$ minimization and computation of the new SH
Kalman filtering and final SH estimation

Figure 4: *Schematic representation of the proposed tracking strategy.*

The rest of this paper is organised as follows: in section 2 we review the mathematical tools we used in our algorithm, section 3 presents a description of the solution we implemented, section 4 shows video simulations and a comparison with the CamShift algorithm, finally, in section 5 we point out the pros and cons of our approach.

A word on notations. In the following we let bold fonts indicate vectors and random variables, depending on context, normal fonts indicate scalars.

2. Theoretical background

In this section we provide a description of the mathematical tools used in our algorithm, namely, SVM theory, Newton-Raphson minimization and Kalman filter in information form.

2.1. Support Vector Machine Theory: a brief review

The support vector machine framework is currently one of the most popular strategy for supervised machine learning and classification. According to the theory of SVMs [37, 38], while traditional techniques for pattern recognition are based on the minimization of the *empirical risk* that is, on the attempt to optimize the performance on the training set, SVMs minimize the *structural risk*, that is, the probability of misclassifying yet-to-be-seen patterns for a fixed but unknown probability distribution of the data. What makes SVMs attractive is:

- the ability to condense the information contained in the training set (stored in few Support Vectors);
- the use of families of decision surfaces of relatively low VC-dimension.[2]

In the following, we restrict the tractation focusing on the linear SVM case which is the version used in our algorithm. The interested reader can find a good introduction of SVMs theory in [11], while more advanced references are [17, 30].

2.1.1. Linear SVM: the separable case

Consider the problem of binary classication also termed dichotomization. We define the training set as

$$\mathfrak{T} := \{(\mathbf{x}_i, y_i) \ : \ \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{-1, +1\}, i = 1, \dots, N\}, \tag{4}$$

where $\mathbf{x}_i$ are the training data and y_i their corresponding labels. First of all we give the definition of linearly separable training set.

Definition 1. *A training set $\mathfrak{T}$ as defined in (4) is called* linearly separable *if*

$$\exists \boldsymbol{\beta} \in \mathbb{R}^n, \beta_0 \in \mathbb{R} \ : \ y_i(\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0) \geq 1, \forall i = 1, \dots, N. \tag{5}$$

Otherwise $\mathfrak{T}$ is said to be non-linearly separable.

Relation (5) implicitly defines a Separating Hyperplane (SH) of equation:

$$f_{\mathrm{SH}}(\mathbf{x}) := \mathbf{x}^\top \boldsymbol{\beta} + \beta_0 = 0 \tag{6}$$

In the case of linearly separable data, Linear Support Vector Classification – among all the SHs that minimize the training error (*i.e.*, empirical risk) – finds the one with the largest margin.

[2]VC (Vapnik-Chervonenkis) dimension is a measure of the capacity of a statistical classification algorithm, defined as the cardinality of the largest set of points that the algorithm can *shatter*.

Consider now the points for which the equalities in (5) holds.[3] For $y_i > 0$ these points lie on hyperplane $\mathcal{H}_+ : \mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0 = 1$ with normal $\boldsymbol{\beta}$ and perpendicular distance from the origin $|1 - \beta_0| / \|\boldsymbol{\beta}\|$. Similarly for $y_i < 0$, these points lie on the hyperplane $\mathcal{H}_- : \mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0 = -1$, with normal again $\boldsymbol{\beta}$ and perpendicular distance from the origin $|-1 - \beta_0| / \|\boldsymbol{\beta}\|$. Hence the minimum Euclidean distance of positive and negative samples from SH is, respectively, $d_\pm := 1/\|\boldsymbol{\beta}\|$ and the margin is simply $2/\|\boldsymbol{\beta}\|$. Note that $\mathcal{H}_+ // \mathcal{H}_-$ and that no training points fall between the two hyperplanes. Thus we can find the margin maximizer SH by minimizing $\|\boldsymbol{\beta}\|^2$, subject to constraints (5). A mathematical model for the problem is hence the following:

$$\begin{aligned} &\min_{\boldsymbol{\beta}, \beta_0} \frac{1}{2} \|\boldsymbol{\beta}\|^2, \\ &\text{s.t.} \quad y_i(\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0) \geq 1, \qquad \forall i = 1, \dots, N, \\ &\qquad (\boldsymbol{\beta}, \beta_0) \in \mathbb{R}^n \times \mathbb{R} \end{aligned} \tag{7}$$

Consider the Lagrangian relaxation of problem (7)

$$\min_{\boldsymbol{\beta}, \beta_0} \mathscr{L}(\boldsymbol{\beta}, \beta_0; \boldsymbol{\lambda}), \tag{8}$$

where,

$$\mathscr{L}(\boldsymbol{\beta}, \beta_0; \boldsymbol{\lambda}) := \frac{1}{2} \|\boldsymbol{\beta}\|^2 + \sum_{i=1}^N \lambda_i (1 - y_i(\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0)) \tag{9}$$

is the Lagrangian and $\boldsymbol{\lambda} := (\lambda_1, \dots, \lambda_N) \in \mathbb{R}_+^N$ is the vector of Lagrangian multipliers. From convex programming theory [8, Chap.5], we know that the Lagrangian has a saddle point in $(\boldsymbol{\beta}^*, \beta_0^*; \boldsymbol{\lambda}^*)$ if and only if $(\boldsymbol{\beta}^*, \beta_0^*)$ is a solution of problem (7). Computing the gradient of the Lagrangian with respect to $\boldsymbol{\beta}$ and β_0 and setting it to zero, gives:

$$\frac{\partial \mathscr{L}(\boldsymbol{\beta}, \beta_0; \boldsymbol{\lambda})}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \sum_{i=1}^N \lambda_i y_i \mathbf{x}_i = \mathbf{0}, \tag{10}$$

$$\frac{\partial \mathscr{L}(\boldsymbol{\beta}, \beta_0; \boldsymbol{\lambda})}{\partial \beta_0} = \sum_{i=1}^N \lambda_i y_i = 0, \tag{11}$$

So in correspondence of an optimal solution $(\boldsymbol{\beta}^*, \beta_0)$ it must exist an optimal vector $\boldsymbol{\lambda}^*$ such that,

$$\boldsymbol{\beta}^* = \sum_{i=1}^N \lambda_i^* y_i \mathbf{x}_i, \tag{12}$$

$$\sum_{i=1}^N \lambda_i^* y_i = 0. \tag{13}$$

Using these conditions, the Lagrangian (9) could be rewritten as a function of the only λ_i's variables:

$$\begin{aligned} \mathscr{L}(\boldsymbol{\beta}, \beta_0; \boldsymbol{\lambda}) &= \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \\ &= \sum_{i=1}^N \lambda_i - \frac{1}{2} \boldsymbol{\lambda}^\top \Theta \boldsymbol{\lambda} := \ell(\boldsymbol{\lambda}), \end{aligned} \tag{14}$$

[3]By assumption, such points always exist up to a normalization factor for $\boldsymbol{\beta}$ and β_0.

where $\Theta \in \mathbb{R}^{N\times N}$, $[\Theta]_{ij} = y_i \mathbf{x}_i^\top \mathbf{x}_j y_j$. Therefore the problem of finding the optimal value $\boldsymbol{\lambda}^*$ can be traced back to that of solving the dual version of Lagrangian problem in (8)

$$\max_{\boldsymbol{\lambda}\geq 0} \inf_{\boldsymbol{\beta},\beta_0} \mathscr{L}(\boldsymbol{\beta},\beta_0;\boldsymbol{\lambda}) = \max_{\boldsymbol{\lambda}\geq 0} \ell(\boldsymbol{\lambda}), \qquad (15)$$
$$\text{s.t.} \quad \sum_{i=1}^{N} \lambda_i y_i = 0.$$

In addition to (12) and (13), the Karush-Kuhn-Tucker (KKT) conditions (see [29, Thm 12.1]) give the additional constraints

$$\lambda_i^*(1 - y_i(\mathbf{x}_i^\top \boldsymbol{\beta}^* + \beta_0^*)) = 0, \quad i = 1,\dots,N \qquad (16)$$
$$\lambda_i^* \geq 0, \quad i = 1,\dots,N \qquad (17)$$
$$y_i(\mathbf{x}_i^\top \boldsymbol{\beta}^* + \beta_0^*) \geq 1, \quad i = 1,\dots,N. \qquad (18)$$

For condition (16) in particular it must be: $y_i(\mathbf{x}_i^\top \boldsymbol{\beta}^* + \beta_0^*) = 1$ for points with $\lambda_i^* > 0$ (Support Vectors) and $\lambda_i^* = 0$ for points satisfying $y_i(\mathbf{x}_i^\top \boldsymbol{\beta}^* + \beta_0^*) > 1$. Consequently the only points that give a non-zero contribution to the calculation of $\boldsymbol{\beta}^*$ are the Support Vectors (SVs), *i.e.* those points nearest to the Optimal Separating Hyperplane (OSH). This is also shown in Fig.5. Once computed the optimal vector of multipliers $\boldsymbol{\lambda}^*$ the OSH is defined as

$$\boldsymbol{\beta}^* = \sum_{i=1}^{N} \lambda_i^* y_i \mathbf{x}_i, \qquad (19)$$

$$\beta_0^* = -\frac{1}{2}\left(\min_{i:\lambda_i^*>0}(\mathbf{x}_i^\top \boldsymbol{\beta}^* - y_i) + \max_{i:\lambda_i^*>0}(\mathbf{x}_i^\top \boldsymbol{\beta}^* - y_i)\right), \qquad (20)$$

and the classificator for a new datum $\hat{\mathbf{x}}$ is

$$f_{\text{OSH}}(\hat{\mathbf{x}}) := \text{sgn}(\boldsymbol{\beta}^* \hat{\mathbf{x}} + \beta_0^*). \qquad (21)$$

2.1.2. Linear SVM: the non separable case

If the training set $\mathfrak{T}$ is non-linearly separable, instead of using a hard-margin linear classifier we can use a soft-margin linear classifier. Therefore, to handle non-separable datasets, we relax the constraints by making the inequalities easier to satisfy. This is done with slack variables $\xi_i \geq 0$ one for each constraint:

$$y_i(\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0) \geq 1 - \xi_i, \quad i = 1,\dots,N. \qquad (22)$$

In this way the previous model (7) can be generalized as follows:

$$\min_{\boldsymbol{\beta},\beta_0} \left\{ \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C\sum_{i=1}^{N}\xi_i \right\}, \qquad (23)$$
$$\text{s.t.} \quad y_i(\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0) \geq 1 - \xi_i, \quad \forall i = 1,\dots,N$$
$$\xi_i \geq 0, \quad \forall i = 1,\dots,N$$
$$(\boldsymbol{\beta},\beta_0) \in \mathbb{R}^n \times \mathbb{R}$$

where C is a parameter to be carefully chosen by the user, a larger C corresponding to assigning a higher penalty to errors. We can similarly look at the dual problem of (23) by introducing Lagrange multipliers. We arrive at

$$\max_{\boldsymbol{\lambda}} \left\{ \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \right\}, \qquad (24)$$
$$\text{s.t.} \quad \sum_{i=1}^{N}\lambda_i y_i = 0,$$
$$0 \leq \lambda_i \leq C, \quad i = 1,\dots,N.$$

As before, when $\lambda_i = 0$ the i-th point is not a Support Vector and can be ignored. When $0 < \lambda_i < C$, it can be shown using KKT complementarity conditions that $\xi_i = 0$, *i.e.*, the i-th point is on the margin. When $\lambda_i = C$, the i-th point is inside the margin if $\xi \leq 1$, or on the wrong side of the decision boundary if $\xi > 1$. An example is illustrated in Fig.6. The discriminant for a new datum $\hat{\mathbf{x}}$ is again of the form in (21) and the offset β_0^* can be computed on Support Vectors with $0 < \lambda_i < C$.

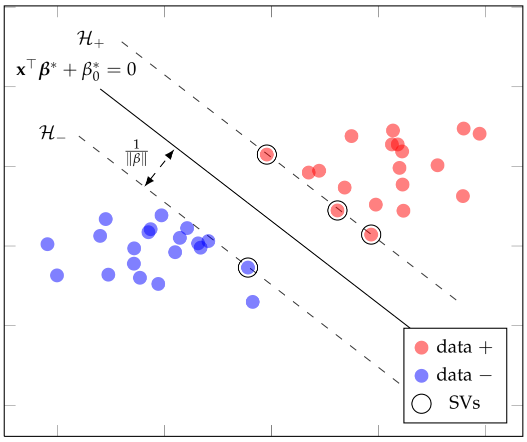


Figure 5: *OSH and SVs in the linearly separable case.*

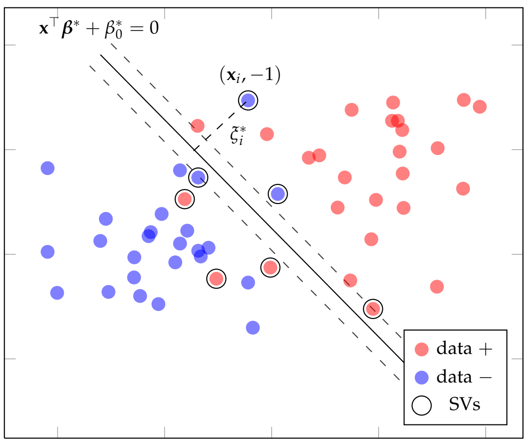


Figure 6: *OSH and SVs in the non-linearly separable case.*

2.1.3. Non linear SVM and the kernel trick

In practice, linear classifiers generally do not provide an acceptable classification in presence of a non-linearly separable dataset. To improve the goodness of pattern classification, different non-linear criteria have to be adopted, even if this could deteriorate the computational performance of the classification strategy. An efficient method to perform non-linear classification using SVMs consists in exploiting *kernel functions*. The idea behind the use of kernel functions can be explained by an example. Consider the following training dataset $\mathfrak{T}_{1D} = \{(x_i, y_i),\ i = 1, \dots, 3\} = \{(-1, 1), (0, -1), (1, 1)\}$, where $x \in \mathbb{R}$. This is not a linearly separable dataset. However, if we map x to a three dimensional vector (Fig. 7)

$$\phi(x) = [1, \sqrt{2}x, x^2]^\top \tag{25}$$

the dataset becomes linearly separable in the three dimensional space (equivalently, we have a non-linear decision boundary in the original space). The map does not actually increase the intrinsic dimensionality of x: $\phi(x)$ lies on a one dimensional manifold in $\mathbb{R}^3$. Nonetheless, this suggests a general way to handle linearly non-separable data: map $\mathbf{x}$ to some higher dimensional space (the *feature-space*) by the function $\phi(\mathbf{x})$. However, if $\phi(\cdot)$ is very high dimensional, representing it and computing the inner product becomes an issue. Kernel theory [32] provides an efficient solution to this problem. Note the dual problem (24) and its solution involves inner product of feature vectors $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ only. Thus it might be possible to use a feature representation $\phi(\mathbf{x})$ without explicitly representing it, as long as we can compute the inner product using an adequate kernel function. For instance, the inner product of (25) can be computed as

$$K(x_i, x_j) = \phi(x_i)^\top \phi(x_j) = (1 + x_i x_j)^2. \tag{26}$$

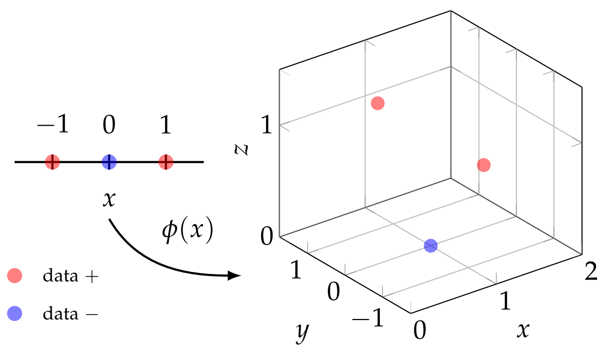


Figure 7: *Toy example of feature-space method: the training data, initially non-linearly separable, become linearly separable in a higher dimensional space.*

Not all functions K are suitable kernel functions. The following theorem, due to Mercer, gives a sufficient and necessary condition to estabilish whether a certain function can be a kernel function.

Theorem 1 (Mercer - 1909). *Let $\mathfrak{X}$ a compact subspace of $\mathbb{R}^n$, a symmetric function $K(\mathbf{x}, \mathbf{y}) : \mathfrak{X} \times \mathfrak{X} \to \mathbb{R}$ can be expressed as an inner product $K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$ for some function $\phi(\cdot)$ if and only if $K(\mathbf{x}, \mathbf{y})$ is positive semidefinite, i.e.*

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y})\, \mathrm{d}\mathbf{x}\, \mathrm{d}\mathbf{y} \geq 0, \qquad \forall g \in L_2(\mathfrak{X}), \tag{27}$$

or equivalenty, if $\mathfrak{X}$ is a finite input space, the Gram matrix $\mathbf{K}$ is positive semidefinite, i.e. for any finite sequence of input vectors $(\mathbf{x}_1, \dots, \mathbf{x}_n)$

$$\mathbf{K} = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots \\ K(\mathbf{x}_2, \mathbf{x}_1) & \ddots & \\ \vdots & & \end{bmatrix} \succeq 0. \tag{28}$$

Proof. Can be found in [16, Chap. III §5]. □

Commonly used kernels include:

- ✦ Polynomial kernels: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^\top \mathbf{x}_j)^p$;
- ✦ Radial Basis Function (RBF) or Gaussian kernels: $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^\gamma}{2\sigma^2}\right)$;
- ✦ Sigmoid kernels: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(\gamma\ \mathbf{x}_i^\top \mathbf{x}_j + \delta\right)^p$.

2.1.4. SVM as a penalization problem

There is a perfectly equivalent formulation of the SVM problem previously reviewed. SVM classification can be considered as a unconstrained minimization problem in the form “loss+penalty”.

Theorem 2. *The SVM problem can be equivalenty replaced with the problem of minimizing the quantity w.r.t. $\boldsymbol{\beta}, \beta_0$:*

$$\mathfrak{L} := \sum_{i=1}^{N} \ell(y_i, f(\mathbf{x}_i)) + \lambda \|\boldsymbol{\beta}\|^2, \tag{29}$$

here the loss function $\mathfrak{L}(\cdot)$ is the Hinge Loss function, defined as

$$\ell(y_i, f(\mathbf{x}_i)) := [1 - y_i f(\mathbf{x}_i)]_+, \tag{30}$$

where operation $[\cdot]_+$ sets all negative values equal to zero. The loss function weighs the error committed in the classification of the point $\mathbf{x}_i$ and the second term in the sum (29) is a regularization term (quadratic penalty) which takes the value $\lambda := \frac{1}{2C}$.

Proof. See Appendix A. □

Notice that Hinge Loss function (30) has a non-differentiable point at 1. Many numerical minimization algorithms requires that minimizing function is a $\mathcal{C}^2$ class function. Therefore it is convenient to replace the Hinge Loss with an approximating function, *e.g.* the Binomial Deviance,

$$\ell_{BD}(y_i, f(\mathbf{x}_i)) = \log(1 + e^{-y_i f(\mathbf{x}_i)}), \tag{31}$$

Binomial Deviance and other approximating function are plotted in Fig.8, a detailed analysis of properties of these functions can be found in [19]. Among these the (negative) log-likelihood or Binomial Deviance has the relevant properties of being a smooth function and a "margin maximizing loss function". Finally, the minimization problem (29) using Binomial Deviance as loss function can be rewritten as

$$\min_{\beta,\beta_0} \mathfrak{L}_{BD}, \tag{32}$$

where,

$$\mathfrak{L}_{BD} := \sum_{i=1}^{N} \log\left(1 + e^{-y_i(\mathbf{x}_i^\top \beta + \beta_0)}\right) + \lambda \|\beta\|^2. \tag{33}$$

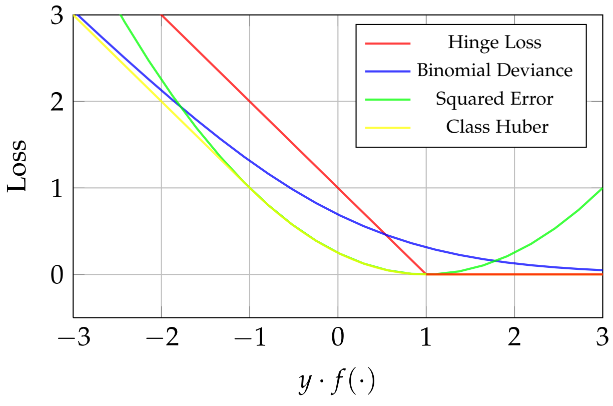


Figure 8: *Comparison of approximating loss functions.*

2.2. Minimization algorithms: Newton's method

The formulation we adopted inherently reveals a problem of unconstrained convex optimization, such as $\min f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$, where $f : \mathbb{R}^n \to \mathbb{R}$ is a continuous convex function. It is usually hard to solve such non-linear optimization problems using only the first and second order conditions, *i.e.*

$$\nabla f(\mathbf{x}^*) = \mathbf{0}, \tag{34}$$

$$\mathbf{d}^\top \nabla^2 f(\mathbf{x}^*) \mathbf{d} \geq 0, \quad \forall \mathbf{d} \in \mathbb{R}^n. \tag{35}$$

To accomplish this task there are some iterative algorithms known as *line search algorithms*, the most noteworthy among those ones are: the gradient method, the Newton-Raphson method and the Quasi-Newton methods. This kind of algorithms need to be initialized, choosing a starting point reasonably close to the objective functions minimum. Then they generate a sequence of points that asymptotically approach to the optimal solution. Given the current point, the following one is determined by choosing an appropriate descent direction of the objective function and the length of the step along this one. The main differences among the mentioned three methods regard the way the direction and the step length are chosen. This is a crucial decision because it influences the algorithms performances, such as convergence to a stationary point and convergence speed. The gradient method is very intuitive, it is based on the idea that the minimum of a function must be a stationary point, as confirmed by the first order condition (34). Hence, the algorithm checks at every iteration if the current solution sets the gradient to zero and terminates if such condition is satisfied within a tolerance range, otherwise it takes as descent direction $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$ which is that one that guarantees the highest descent rate. Along the found direction it always exists a value $\alpha_k \in \mathbb{R}_+$ for which $f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) < f(\mathbf{x}_k)$ is satisfied. The determination of an appropriate value for α_k leads to an optimization problem in a single variable which, with exception for particular cases, for example if $f(\cdot)$ is a quadratic function, is computationally expensive to solve with an exact algorithm. It is for this reason that it is usually accepted an approximated solution for α_k. The most efficient method consists of increasing iteratively the value of α_k, initially set to a small value, since it satisfies the Wolfe conditions, reported below:

$$f(\mathbf{x}_k + \alpha_k \mathbf{x}_k) \leq f(\mathbf{x}_k) + c_1 \alpha_k \nabla f(\mathbf{x}_k)^\top \mathbf{d}_k, \tag{36}$$

$$\nabla f(\mathbf{x}_k + \alpha_k \mathbf{d}_k)^\top \mathbf{d}_k \geq c_2 \nabla f(\mathbf{x}_k)^\top \mathbf{d}_k, \tag{37}$$

where $0 < c_1 < c_2 < 1$ are fixed coefficients. The convergence of the gradient method is guaranteed and rigorously demonstrated and the convergence speed is linear and therefore quite slow. Unlike the gradient method, the Newton-Raphson method computes the descent direction and the step length with a single calculation, using a local approximation of the function $f(\cdot)$ provided by the Taylor series in the current point, up to second order terms

$$\tilde{f}(\mathbf{x}_k + \Delta\mathbf{x}_k) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \Delta\mathbf{x}_k + \frac{1}{2} \Delta\mathbf{x}_k^\top \nabla^2 f(\mathbf{x}_k) \Delta\mathbf{x}_k \tag{38}$$

If the Hessian of $f(\cdot)$ is definite and non-singular in the current point the new solution at $k+1$ iteration, can be determined as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k) \tag{39}$$

It is noteworthy that the function $f(\cdot)$ we are treating for our purposes is convex, this guarantees the positive-definiteness of the hessian and the validity of the above equation (39). In a possible interpretation of this algorithm the descent direction can be identified by

$$\mathbf{d}_k = -(\nabla^2 f(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k) \tag{40}$$

and the step length is always unitary. However for the real application we implemented an adaptive step length in order to avoid eventual undesidered behaviours.

The Quasi-Newton method is the most common alternative to the Newton-Raphson method. It identifies a descent direction and a non unitary step length, determined with a line search algorithm. It usually guarantees better performances in terms of convergence speed. The descent direction is identified using an approximation of the hessian matrix in the current point. This approximation is improved every iteration in order to maintain certain properties (such as symmetry and definite-positiveness). The Newton-Raphson method could result more computationally expensive due to the inversion of the hessian matrix, needed every iteration, however this is not guaranteed because the Quasi-Newton method needs to solve every iteration an optimization problem to find the step length.

2.3. Kalman Filter in information form

In this section we briefly describe the Kalman filter in information form. It is a set of equations that are only a transformation of the usual Kalman filter equations but they can be considered computationally better than standard equations because it is required the inversion of a $n \times n$ matrix instead of a $m \times m$ matrix (usually m, the number of measures, is very larger than n, the dimension of state space). Relately to the model

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + \mathbf{w}_t \tag{41}$$
$$\mathbf{y}_t = C\mathbf{x}_t + \mathbf{v}_t \tag{42}$$

with $\mathbf{v}_t \sim \mathcal{N}(0,R)$, $\mathbb{E}[\mathbf{v}_t\mathbf{v}_s] = R\delta(t-s)$, $\mathbf{w}_t \sim \mathcal{N}(0,Q)$, $\mathbb{E}[\mathbf{w}_t\mathbf{w}_s] = Q\delta(t-s), \mathbf{x}_0 \sim \mathcal{N}(\bar{\mathbf{x}}_0, P_0)$ and $\mathbf{v}_t, \mathbf{w}_t, \mathbf{x}_0$ are uncorrelated each other, the Kalman equations in information form are:

$$\hat{\mathbf{x}}_{t+1|t+1} = P_{t+1|t+1}(P^{-1}_{t+1|t}\hat{\mathbf{x}}_{t+1|t} + C^\top R^{-1}\mathbf{y}_{t+1}) \tag{43}$$
$$P_{t+1|t+1} = (P^{-1}_{t+1|t} + C^\top R^{-1} C)^{-1} \tag{44}$$

used updating and

$$\hat{\mathbf{x}}_{t+1|t} = A\hat{\mathbf{x}}_{t|t} \tag{45}$$
$$P_{t+1|t} = AP_{t|t}A^\top + Q \tag{46}$$

used for prediction.

We can notice that while in standard Kalman equations we have to invert the matrix

$$(CP_{t+1|t}C^\top + R) \in \mathbb{R}^{m\times m} \tag{47}$$

that is a $m \times m$ matrix, in equations number (44) we have to invert a $n \times n$ matrix. Since that to invert a square matrix the complexity is $\mathcal{O}(n^3)$ where n is its dimension, the computational complexity using Kalman equations in information form is clearly inferior than the case of standard Kalman equations.

3. SVC tracking algorithm

For the implementation of our algorithm we used C++ programming language with OpenCV libraries, frequently used in Computer Vision applications. Moreover we chose the values of HSV scale as features for tracking instead of RGB values because HSV scale is more suitable for image processing and analysis (our case) than RGB values (more used in computer graphics).

3.1. Description of the algorithm

The first part of the algorithm is the initialization step, the only supervised part of the whole algorithm. When the program grabs the first video frame, user selects the area where the object lies dragging the mouse and then he sets the thresholds of HSV values. The pixels of this frame are classified with label $+1$ if they belong to object and -1 if they belong to the background. This operation is realized by a built-in function of OpenCV libraries. After the inizialization, the algorithm works unsupervised until the end of the video. When t-th frame is grabbed, its pixels are classified using the hyperplane estimated at the previous frame. Then uninformative data are discarded using an adaptive threshold since they do not contribute to determination of the new SH parameters. Then pixels within the threshold are used by Newton-Raphson algorithm to find the SH parameters which minimize the Binomial Deviance. At the end Kalman filter in information form combines these SH parameters and those ones of the previous frame to compute the new optimal hyperplane. A schematic representation of the whole algorithm is depicted in Fig.9.

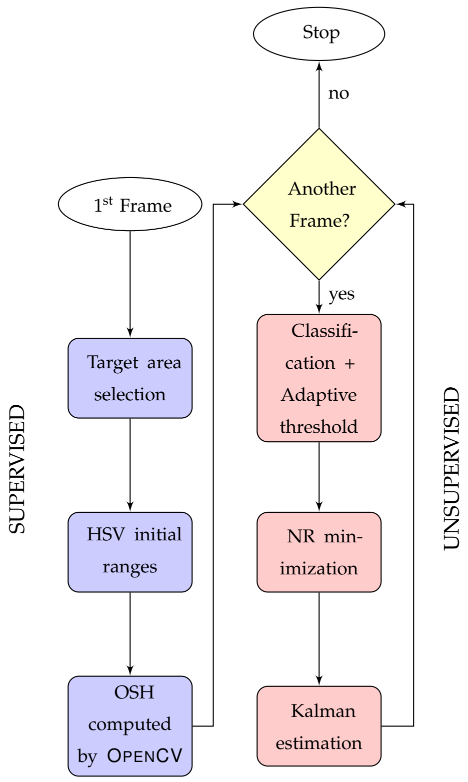


Figure 9: *Scheme of the algorithm.*

3.2. Implementative aspects

3.2.1. Feature selection

For the sake of simplicity, in our algorithm we chose three features based on colors as tracking features. Instead of using the standard RGB (Red-Green-Blue) scale, we opted for the HSB/HSV (Hue-Saturation-Brightness/Value) color scale, often used in Computer Vision applications. The reason of this choice is that HSV (as well as other colorspaces, *e.g.* YCbCr, Lab, etc.) separates *luma*, or the image intensity, from *chroma* or the color information.

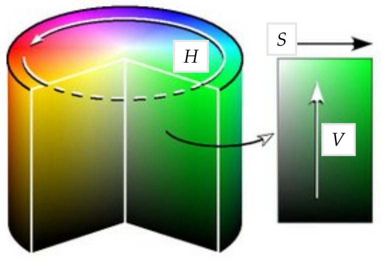
In general the color of a picture can be represented by a model, that is a tool which can specify, create and visualize the color itself without ambiguity. Color models can be classified in:

- Hardware dependent, that is, used to reproduce colors in videos, computers and printers (RGB, CMY, YUV, YIQ); for the user it is difficult dealing with these models because they do not directly refer to the notions of tint, saturation and brightness;

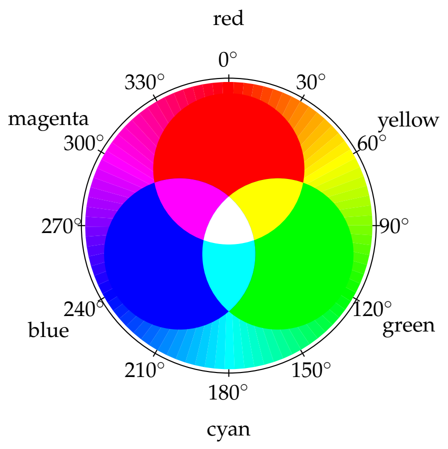
- User oriented, that is, based on the human perception of colors (HLS, HSV) and they are more intuitive and uniform

RGB model is based on 3 colors: R = red (wavelength = 700 nm), G = green (wavelength = 546.1 nm) and B = blu (wavelength = 435.8 nm). If these colors are summed, they produce white. RGB model is inefficient for the characteristics of human eye because human eye is not sensible in the same way to red, green and blue and it is more sensible to luminosity differences than color differences. The most significant defect of RGB scale is that there is a *high correlation* among the three channels (for example changing intensity of color, changes all the three components). On the other hand, HSV model is closer to the way we "see" colors and it is usually represented by a cylinder or a cone (see Fig.10). It is based on 3 parameters: H = hue (angle between 0° and 360°), S = saturation (value between 0 and 1) and V = value (value between 0 and 1). Hue represents the color itself, saturation is the distance of the color from the nearest grey and value represents how much white there is in the color. HSV space is very used in image processing and presents some benefits: in fact, for example, hue is very robust when illumination changes and it is invariant to shadows and highlights. This space also presents some disadvantages: for example when V has a value near to 0, H and S are not defined and when S is near to 0, H is not defined. In this problematic cases we can even observe some discontinuities in the representations of colors. For further information on this topic we refer to [26].

A comparison of RGB classification vs. HSV classification carried out using our algorithm is provided in Fig.11. Although in both cases the background and foreground data are not tightly clustered (we deal with non-separable data sets), in the HSV case we can see that brightness variations (V axis) not lead to wrong classification due to the particular shape of the separating hyperplane.



(a)



(b)

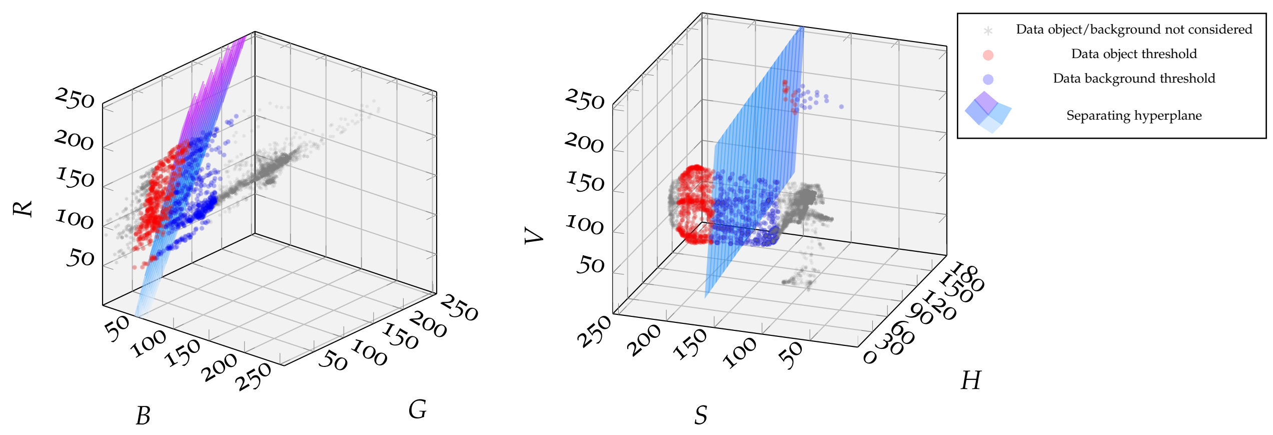
Figure 10: (a) *HSV scale,* (b) *this illustration shows samples of the HSV color space and how they relate to the RGB color model (the saturation and value parameters are set to $S = V = 1$).*

3.2.2. Newton-Raphson implementation

The Binomial Deviance function is a $\mathcal{C}^2$ class approximation of the Hinge Loss function and therefore guarantees the existence of the first and second order derivatives. It is possible thus to apply the Newton-Raphson iterative scheme which adapted to our case results:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \epsilon[\nabla^2\mathfrak{L}(\mathbf{x}_k)]^{-1}\nabla\mathfrak{L}(\mathbf{x}_k), \quad k = 1,\dots,k_{\max} \tag{48}$$

where $\mathbf{x}_k := [\boldsymbol{\beta}_k \ \beta_{0,k}]^\top$ and ϵ is an adaptive coefficient which varies the step length in case of pathological behaviors such as cycles due to the function shape where the optimization get stuck to a value that does not improve every iteration. To unlock this unfortunate situation it is sufficient to vary the step length to a smaller value. The algorithm ends when norm of the gradient $\|\nabla\mathfrak{L}(\mathbf{x}_k)\|$ lies within a fixed range $[0, \tau]$. It was convenient to impose a limit number of iterations $k_{\max}$ in case the selected tolerance is not reached within an adequate time. The best results have been obtained imposing a limit of thirty iterations. This guarantees the real time functioning and a sufficient precision for the approximate solution.



(a) RGB colorspace

(b) HSV colorspace

Figure 11: *Comparison of HSV and RGB color features:* Frame #4 *of* `Terzo.avi`. *In* OpenCV *the HSV ranges are:* $H \in [0, 180]$, $S \in [0, 256]$, $V \in [0, 256]$.

3.2.3. Implementation of Kalman filter

In our case the Kalman equations are

$$\hat{\mathbf{x}}_{t+1|t+1} = P_{t+1|t+1}(P^{-1}_{t+1|t}\hat{\mathbf{x}}_{t+1|t} + H_{k^*}\mathbf{y}_{t+1}) \quad (49)$$

$$P_{t+1|t+1} = (P^{-1}_{t+1|t} + H_{p^*})^{-1} \quad (50)$$

used for updating and

$$\hat{\mathbf{x}}_{t+1|t} = \hat{\mathbf{x}}_{t|t} \quad (51)$$

$$P_{t+1|t} = P_{t|t} + Q \quad (52)$$

used for prediction.

Here we use the following hypothesis: first of all we refer to a state space model (41)-(42) where matrices A and C are the identity matrices and the state is formed by the coefficients of the hyperplane which divides the object points from the background ones. Then we model the noises $\mathbf{w}$ and $\mathbf{v}$ as white Gaussian noises with zero mean and with Q and R as variance matrices. We choose Q as a diagonal matrix where the first three elements take “small” values (our conjecture refers to a slow dynamics for the first three coefficients of the hyperplane) while the last one is “big” (fast dynamics).

The matrix R is replaced by the inverse of the Hessian matrix H_{k^*} of the function Binomial Deviance computed at final iteration k^* of Newton-Raphson algorithm.

This choice hinges on the fact (explained in [33,36]) that Binomial Deviance $\mathfrak{L}_{BD}$ (33) (as any other loss function) can be regarded as defining a (negative) log-posterior probability for the state $\mathbf{x} = [\boldsymbol{\beta}\ \beta_0]^\top$ of the SVM given a training set $\mathfrak{D}$ of N points. If we make the further assumption that this posterior density $p(\mathbf{x} \mid \mathfrak{D})$ can be approximated by a Gaussian density (Laplace approximation), *i.e.* $p(\mathbf{x} \mid \mathfrak{D}) \sim \mathcal{N}(\mathbf{x}_{k^*}, R)$, the covariance matrix is derived from the Hessian matrix of the loss function. In fact, under Gaussianity assumption, we have

$$\mathfrak{L}_{BD}(\mathbf{x}) = -\log p(\mathbf{x} \mid \mathfrak{D}) = \\ = \frac{N}{2}\log 2\pi + \frac{1}{2}\log\det(R) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_{k^*})^\top R^{-1}(\mathbf{x} - \mathbf{x}_{k^*}) \quad (53)$$

which is a quadratic function of the components in $\mathbf{x}$. By taking partial differentiations with respect to $(\mathbf{x}_i, \mathbf{x}_j)$, the (i, j) component of the Hessian matrix can be obtained as

$$H_{(i,j)}(\mathbf{x}_{k^*}) = \left.\frac{\partial^2 \mathfrak{L}_{BD}(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_j}\right|_{\mathbf{x}=\mathbf{x}_{k^*}} = (R^{-1})_{(i,j)} \quad (54)$$

so the covariance matrix is equal to the inverse of the Hessian matrix computed at point $\mathbf{x}_{k^*}$:

$$R = H(\mathbf{x}_{k^*})^{-1} =: H^{-1}_{k^*}. \quad (55)$$

As measure $\mathbf{y}_t$ we use the estimate calculated in NR algorithm. The matrix variance P of the state at the first step is zero because we make the assumption that the coefficients are precise at the beginning and there is no uncertainty. The Kalman filter is a very important mathematical tool used for a correct working of the whole algorithm and for its robustness because to calculate the new coefficients of hyperplane for the new classification of the points of the successive frame, it considers not only the coefficients estimated by NR algorithm but also considers the dynamics of the past history of the coefficients of the previous frames. With the updating equation relative to the state we can notice that the new estimate is a sort of combination of the contribute due to NR algorithm and to the state of the previous frame weighed by the variance matrices.

3.2.4. Adaptive threshold

To further improve algorithm computational speed we consider in the minimization step and in the Kalman equations only a subset of object and background data. We tackled the problem of choosing the optimal subset of data at each step using the solution described below. Other adaptive threshold solutions have also been implemented and compared, the results are analyzed in Appendix C.

The “nearest-hyperplane” data threshold

An intuitive method of setting the threshold is that of considering only those points nearest the hyperplane. This intuition is confirmed by the following proposition and by its subsequent corollary.

Proposition 1. *The best single point approximation of the minimizing argument in (32) can be found choosing that point which satisfy*

$$\arg\min_{i=1,\dots,N} y_i \mathbf{x}_i^\top \boldsymbol{\beta}. \tag{56}$$

Proof. See Appendix A. □

Corollary 1. *The best approximating quantity of the argument in (29) under k points cardinality constraint can be found selecting those k points nearest (with respect to the Euclidean metric) to the separating hyperplane (SH) described by equation* $\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0 = 0$.

Proof. See Appendix A. □

Since $|f(\mathbf{x})| \propto \operatorname{dist}(\mathbf{x}_i, \mathrm{SH})$ the problem of selecting those k_+ object points and k_- background points nearer to SH can be formally stated as a Integer Linear Programming (ILP) problem

$$\begin{aligned} \min \quad & \sum_{i=1}^{N} |f(\mathbf{x}_i)|\chi_i \\ \text{s.t.} \quad & \sum_{i=1}^{N} \chi_i = k, \\ & \sum_{i:\mathbf{x}_i \in \mathfrak{D}_+} \chi_i \geq \kappa_+, \quad \sum_{i:\mathbf{x}_i \in \mathfrak{D}_-} \chi_i \geq \kappa_-, \\ & \chi_i \in \{0,1\}, \qquad \forall i = 1, \dots, N. \end{aligned} \tag{57}$$

The first constraint in (57) expresses the cardinality constraint, the second states that for each data set it must be taken at least $\kappa_\pm \in \mathbb{N}, \kappa_\pm \geq 1$ data.

In order to adaptively update the threshold based on video conditions at each frame we implement the Algorithm 1.

The algorithm can be split in four blocks:

1. Separate data according to label values ($\mathcal{O}(N)$);
2. Sort the data in decreasing order for $\mathfrak{D}_+$ and in increasing order for $\mathfrak{D}_-$ ($\mathcal{O}(\max\{|\mathfrak{D}_+|, |\mathfrak{D}_-|\} \log\{|\mathfrak{D}_+|, |\mathfrak{D}_-|\})$ using quick-sort algorithm);
3. save the first $k_\pm$ data per set in $S_\pm$ ($\mathcal{O}(\max\{k_+, k_-\})$);
4. update the value of $k_\pm$: if the average distance from SH of points in $\mathfrak{D}_\pm$ is greater than the average distance computed in the previous cycle then decrease $k_\pm$ of $\delta \in \mathbb{N}$ else increase of the same quantity ($\max\{|\mathfrak{D}_+|, |\mathfrak{D}_-|\}$).

Algorithm 1 Adaptive Threshold

```
    // Divide data according to label values
1:  $\mathfrak{D}_+ = \emptyset, \mathfrak{D}_- = \emptyset$;
2:  for $i = 1$ to $N$ do
3:      if $y_i = +1$ then
4:          $\mathfrak{D}_+ = \mathfrak{D}_+ \cup \{\mathbf{x}_i\}$;
5:      else
6:          $\mathfrak{D}_- = \mathfrak{D}_- \cup \{\mathbf{x}_i\}$;
7:      end if
8:  end for
    // Sort by distance data saved in $\mathfrak{D}_+$ and $\mathfrak{D}_-$
9:  $\mathfrak{D}_+ \to \mathfrak{D}_{+,\text{sorted}\downarrow}, \mathfrak{D}_- \to \mathfrak{D}_{-,\text{sorted}\uparrow}$
    // Save the first $k_+$-th data in $S_+$
10: $S_+ = \emptyset$;
11: for $i = 1$ to $k_+$ do
12:     $S_+ = S_+ \cup \{\mathbf{x}_i \in \mathfrak{D}_{+,\text{sorted}\downarrow}\}$;
13: end for
    // Save the first $k_-$-th data in $S_-$
14: $S_- = \emptyset$;
15: for $i = 1$ to $k_-$ do
16:     $S_- = S_- \cup \{\mathbf{x}_i \in \mathfrak{D}_{-,\text{sorted}\uparrow}\}$;
17: end for
    // Compute average dist. of points in $\mathfrak{D}_+$ from SH
18: $\text{dist}_+ = \overline{\text{dist}}(\mathbf{x}_i, \text{SH}), \mathbf{x}_i \in \mathfrak{D}_+$;
19: if $\text{dist}_+ < \text{dist}_{+,\text{prev}}$ then
20:     $k_+ = k_+ + \delta$;
21: else
22:     $k_+ = k_+ - \delta$;
23: end if
    // Compute average dist. of points in $\mathfrak{D}_-$ from SH
24: $\text{dist}_- = \overline{\text{dist}}(\mathbf{x}_i, \text{SH}), \mathbf{x}_i \in \mathfrak{D}_-$;
25: if $\text{dist}_- < \text{dist}_{-,\text{prev}}$ then
26:     $k_- = k_- + \delta$;
27: else
28:     $k_- = k_- - \delta$;
29: end if
```

The utility of updating the number of data (and therefore the threshold) described in the 4-th block, can be explained by a simple and straightforward argument: if the points, on average, move away from hyperplane then there are more points likely considered well-classified and we can take into account a fewer number of points in the minimization, vice versa otherwise.

It is worth noting that since we have split the data in two subset $S_\pm$ each with cardinality $k_\pm$ problem formulation (57) still holds defining $k = k_+ + k_-$, imposing $\kappa_\pm = k_\pm$ and replacing inequalities of the second constraint with equalities.

This algorithm can be seen as a good compromise between overall computational complexity, number of points considered at each step and accuracy in the classification. A possible disadvantage can be found in how to choose empirically the δ parameter and how to initialize $k_\pm$ values.

A comparison between algorithm processing time using both adaptive threshold and a fixed threshold is depicted in Fig.12. The use of the adaptive threshold makes it possible to greatly reduce the processing time of initial frames.

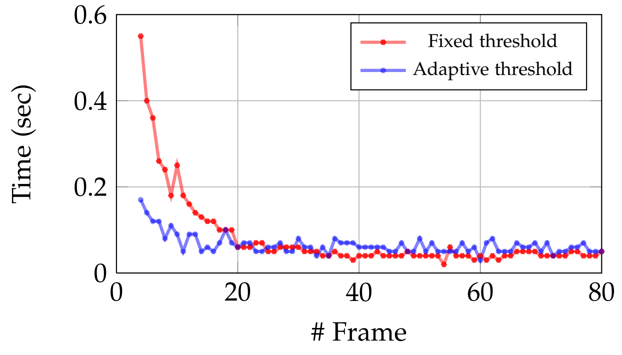


Figure 12: *Fixed threshold* ($|f(\mathbf{x})| \leq 20$) *vs. adaptive threshold, test performed on video* `Terzo.avi`.

3.2.5. Dynamic bounding-box

In order to reduce the computational burden of the algorithm and to partially avoid the drifting problem, we considered in foreground/background classification only a portion of the total number of frame pixels. This is achieved using a "smart" dynamic bounding box. The bounding box behavior is illustrated in Algorithm 2 and its main properties can be summarized as follows:

- the position of the bounding box is updated at each frame in order to follow the target (line 16);
- the dimension of the box can slightly change during time, proportionally – according to coefficients $c_w, c_h \in (0, 1]$ – to the speed variations of the tracked object (line 10);
- if the object disappears from the search region (this can be due to abrupt changes of speed and/or sudden illumination variations), the box increases its area until the object is recovered (line 12).

Algorithm 2 Dynamic Bounding Box

```
    // Initialization of upper left corner box coords
 1: $x_{box} := x_0$; $y_{box} := y_0$;
    // Initialization of box height and width
 2: $w_{box} := w_0$; $h_{box} := h_0$;
    // Initialization of object coords and velocity
 3: $x_{obj,prev} := 0$; $v_{x,obj,prev} := 0$;
 4: $y_{obj,prev} := 0$; $v_{y,obj,prev} := 0$;

 5: while Another_Frame do
        // Compute object centroid coords and velocity
 6:     $\bar{x}_{obj} := \frac{1}{N_{obj}} \sum_i x_{obj,i}$;
 7:     $\bar{y}_{obj} := \frac{1}{N_{obj}} \sum_i y_{obj,i}$;
 8:     $v_{x,obj} := \frac{\bar{x}_{obj} - \bar{x}_{obj,prev}}{t_{proc}}$;
 9:     $v_{y,obj} := \frac{\bar{y}_{obj} - \bar{y}_{obj,prev}}{t_{proc}}$;

        // Update box width and height
10:     $w_{box} = w_{box} + c_w(v_{x,obj} - v_{x,obj,prev})$;
11:     $h_{box} = h_{box} + c_h(v_{y,obj} - v_{y,obj,prev})$;
        // If tracked object is lost then increase box area
12:     if $\bar{x}_{obj} = 0$ and $\bar{y}_{obj} = 0$ then
13:         $w_{box} = c_{\ell,w}$·Frame_width;
14:         $h_{box} = c_{\ell,h}$·Frame_height;
15:     end if
        // Update box coords
16:     $x_{box} = \bar{x}_{obj} - w_{box}/2$;
17:     $y_{box} = \bar{y}_{obj} - h_{box}/2$;

        // Update previous values
18:     $x_{obj,prev} = x_{obj}$; $v_{x,obj,prev} = v_{x,obj}$;
19:     $y_{obj,prev} = y_{obj}$; $v_{y,obj,prev} = v_{y,obj}$;
20: end while
```

3.2.6. Misdetection alarm

Since we use a linear classifier based on color features, if the original data are not linearly separable (background and foreground share some color features) it is probable that the tracking algorithm wrongly classifies the background as target object or vice versa. A possible way to eliminate the drifting problem, is to use a non-linear classifier, as described in §2.1.3. This solution, however, negatively affects the algorithm real time performances, therefore to avoid excessive complexity we use a linear classifier (*i.e.* a hyperplane), as discussed in previous sections. In order to limit the drifting problem, we implemented a sort of *automatic fault-detection strategy* which informs the user when the algorithm might fail in the tracking task. This strategy is actually very simple and it is based on the variations of number of object pixels, Δn_{obj}, during two subsequent video frames: in the case of a significant variation ($\Delta n_{obj} > \Delta N_{max}$ or $\Delta n_{obj} < \Delta N_{min}$) the alarm is setted on. This function is reported in Algorithm 3.

Algorithm 3 Misdetection Alarm

```
    // Initialization of number of object pixels
 1: $n_{obj,prev} := n_0$;

 2: while Another_Frame do
        // Initialization of alarm state
 3:     alarm:=off;
        // Compute number of object pixels and $\Delta n_{obj}$
 4:     $n_{obj} := \sum pixel_{obj}$;
 5:     $\Delta n_{obj} := n_{obj} - n_{obj,prev}$;
        // If $\Delta n_{obj}$ is relevant alarm is setted on
 6:     if $\Delta n_{obj} < \Delta N_{min}$ or $\Delta n_{obj} > \Delta N_{max}$ then
 7:         alarm=on;
 8:     end if

        // Update previous value
 9:     $n_{obj,prev} = n_{obj}$;
10: end while
```

4. Simulations and performance evaluation

In order to prove the validity of our algorithm we tested it on a dozen of videos in which the target is a red ball with black spots. In this paper we present the most significant ones. A relevant case is video `Terzo.avi`. Looking at the three frames depicted in Fig.16, it is worth to point out three facts:

1. we deal with non linearly separable data sets, this is indeed usually the case in real life situations;
2. the separating hyperplane adapts itself according to the variation of the dynamics of the two clusters of points;
3. the number of points selected by the adaptive threshold decreases over time since the average distance from SH of the foreground/background clusters increases over time.

From Fig.13, 14 and 15 representing the dynamics experienced by hyperplane parameters $\boldsymbol{\beta}$ and β_0 in three different videos, one can observe how the values of hyperplane coefficients tend to settle to some optimal "equilibrium" values.

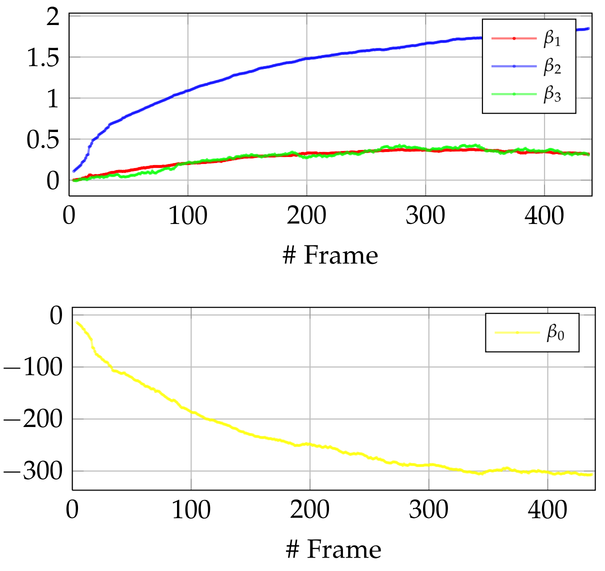


Figure 13: *Dynamics of hyperplane parameters* $\boldsymbol{\beta} = [\beta_1\ \beta_2\ \beta_3]^\top$, β_0, *video* `Primo.avi`.

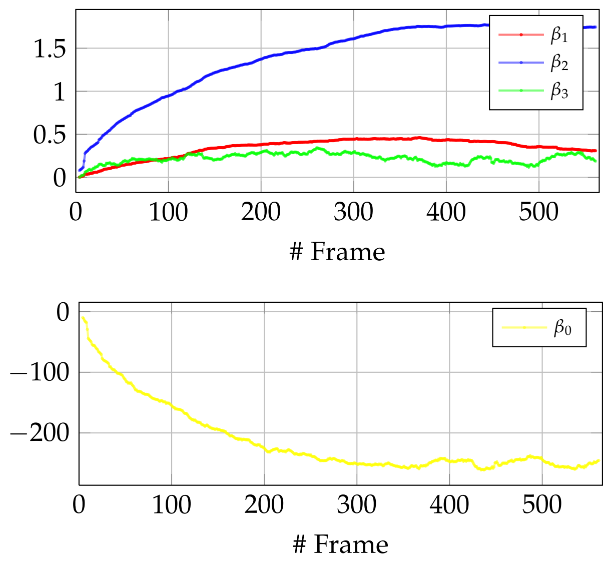


Figure 14: *Dynamics of hyperplane parameters* $\boldsymbol{\beta} = [\beta_1\ \beta_2\ \beta_3]^\top$, β_0, *video* `Secondo.avi`.

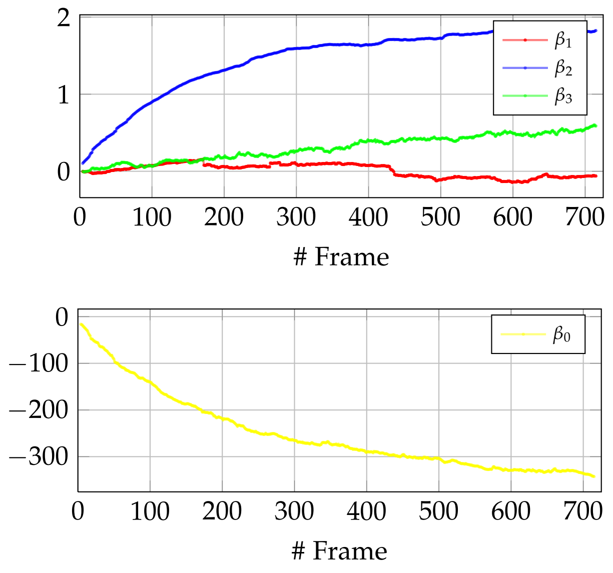
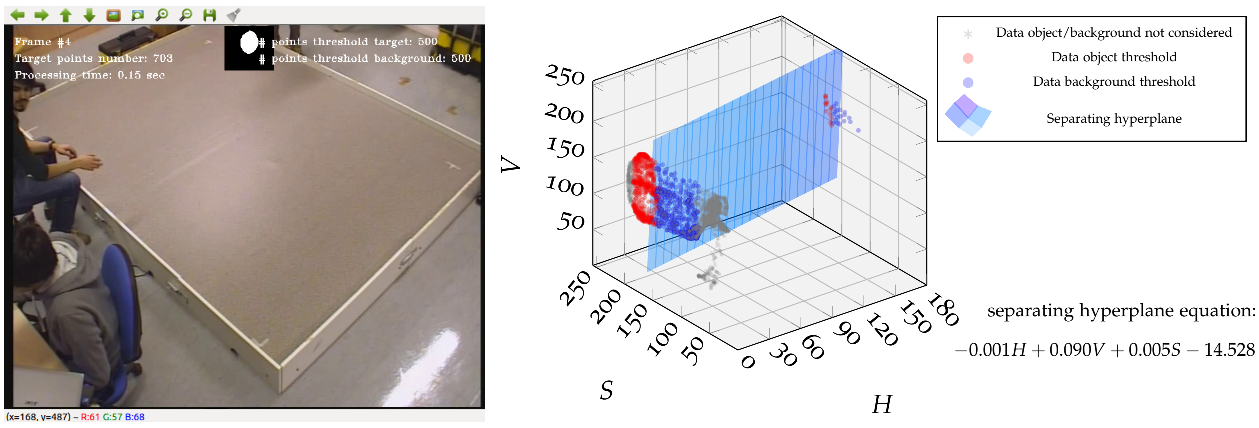


Figure 15: *Dynamics of hyperplane parameters* $\boldsymbol{\beta} = [\beta_1\ \beta_2\ \beta_3]^\top$, β_0, *video* `Terzo.avi`.

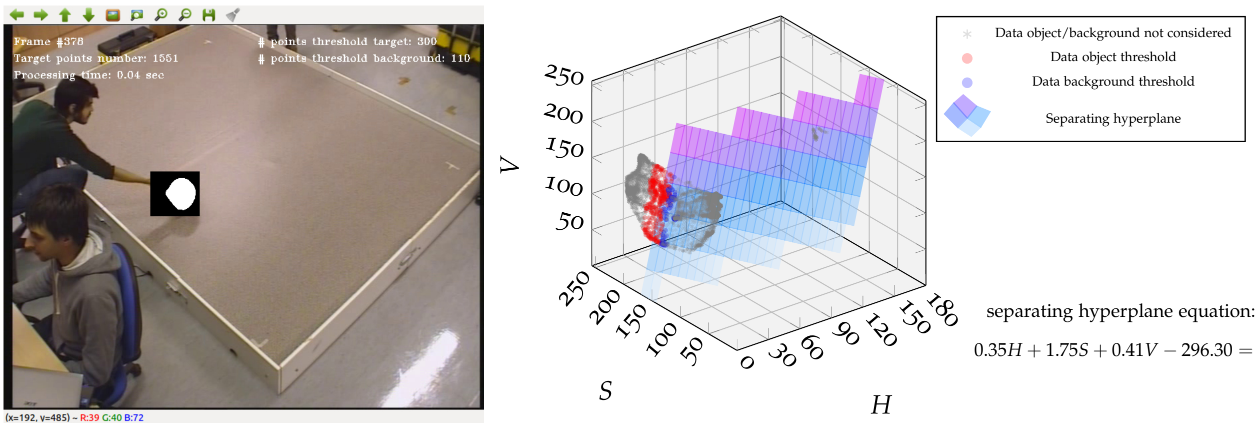
We then compared our strategy to the CamShift algorithm [10], since the C++ program that implements this strategy is directly available in OpenCV libraries. In most cases the two algorithms share similar performances, even if the CamShift algorithm is on average faster than SVC. Nevertheless in two videos in particular the SVC tracking algorithm performs better in terms of adaptivity and robustness. These videos are `Quinto.avi` and `Video_3.avi`. Comparison results on those two videos are illustrated in Fig.17 and Fig.18. The first frames sequence shows that, as a result of a sudden change of brightness, CamShift crashes since it loses the tracked object. SVC tracking algorithm instead is able to adapt to this abrupt brightness variation. It is worth noting that, immediately after the change of brightness, SVC tracking algorithm can not distinguish well the object from the background and the misdetection alarm (represented by the upper right red warnings in central frames) is on. Frames sequence of Fig.18 shows another particular situation: here the red ball undergoes sudden changes in speed and direction. We can see that between 2nd and 3rd frame Camshift algorithm confounds the target with a human hand. This however is not the case of SVC tracking algorithm which is capable to distinguish and track correctly the object.



separating hyperplane equation:

$-0.001H + 0.090V + 0.005S - 14.528 = 0$

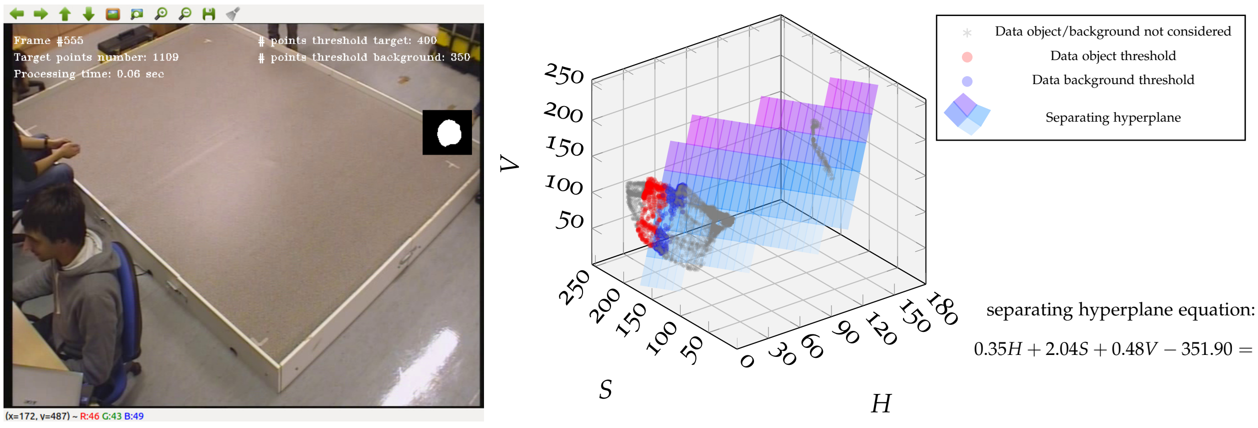
(a) Frame # 4



separating hyperplane equation:

$0.35H + 1.75S + 0.41V - 296.30 = 0$

(b) Frame # 378

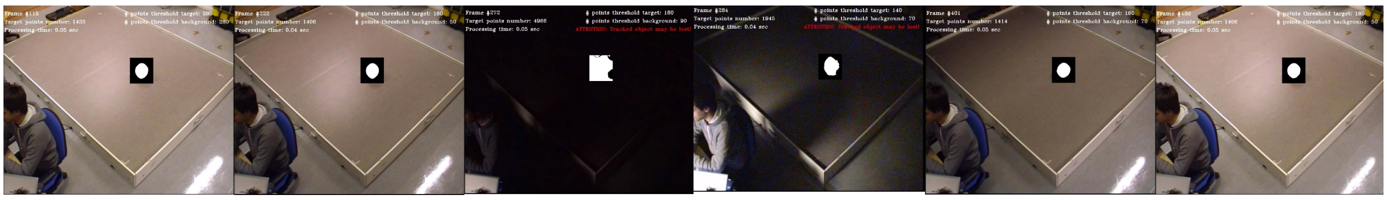


separating hyperplane equation:

$0.35H + 2.04S + 0.48V - 351.90 = 0$

(c) Frame # 555

Figure 16: *Video sequence* `Terzo.avi`


(a) SVC

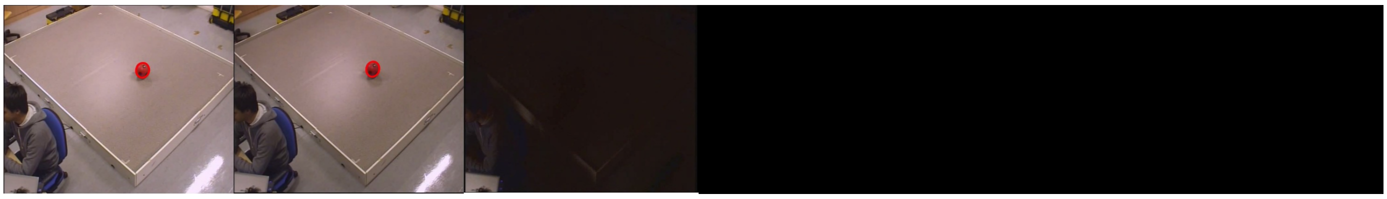
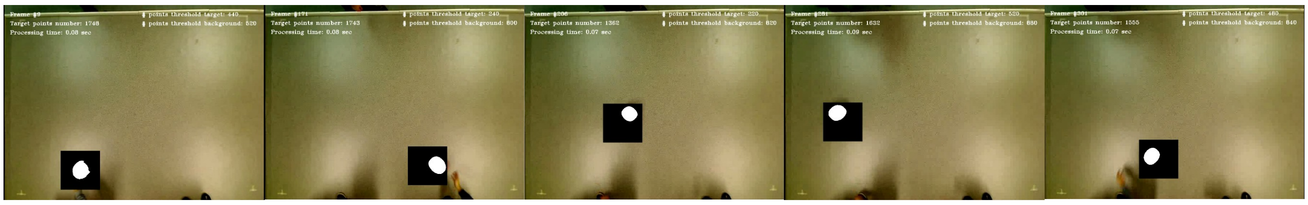

(b) CamShift

Figure 17: *Comparison SVC vs. CamShift* `Quinto.avi`.


(a) SVC

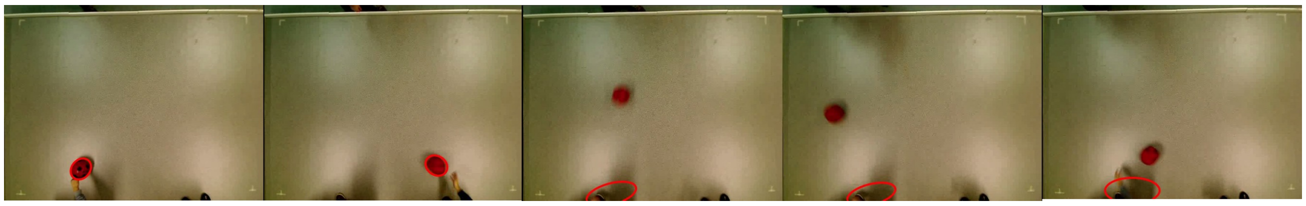

(b) CamShift

Figure 18: *Comparison SVC vs. CamShift* `Video_3.avi`.

5. Conclusions

To sum up our work, we can point out the most two significant results: first, we improved the computational efficiency (we reduced it up to three orders for the first frames of the video respect the previous work [3]) using OpenCV libraries, C++ programming language, the adaptive threshold and the bounding box; second, we increased the accuracy in detection using HSV scale instead of RGB scale.

The good results obtained from the comparison with the CamShift algorithm, a very popular and computationally efficient approach in tracking applications, enforce the idea that our strategy could be a valid alternative to the others proposed in nowadays scientific literature. Nevertheless there are still some weaknesses in our algorithm due specifically to the color based feature targeting. It is inherent for a linear classifier to encounter some difficulties to classify objects with similar features. It is therefore difficult for the problem structure itself to correctly classify an object of a similar color of the background. The most intuitive improvement would be adding some features or finding some relations between the current features to make the algorithm more selective. Nowadays some cameras are capable of depth recognition (for example the Microsoft Kinect®) this would be a great feature to add to our algorithm without deeply modifying the core we built. Another more elegant approach would be to use some suitable kernel functions to generate a non-linear classifier.

Acknowledgements

The authors would like to acknowledge Piero Donaggio for the useful suggestions, Luca Schenato and Gianluigi Pillonetto for fruitful discussions and Nicholas Felicini for the help given in videos realization.

Appendix A Extensive proofs

Proof of Thm 2. Consider the generalized SVM problem in the primal form as represented in (23). The first constraint can be rewritten as

$$\xi_i \geq 1 - y_i f(\mathbf{x}_i). \tag{58}$$

Now we note that in the minimization problem inequality (58) can be written as an equality. Hence from the positivity of the ξ_i's imposed by the second constraint in (23), we have

$$\min_{\beta,\beta_0} C \left[1 - y_i(\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0)\right]_+ + \frac{1}{2}\|\boldsymbol{\beta}\|^2. \tag{59}$$

For the non-linearly separable case the result come straightforwardly dividing by $C > 0$ and by definition of λ. For the linearly separable case $C = \infty$ and prove the correspondence between (29) and (59) becomes tricky. It can be demonstrated that Hinge Loss function (and other loss function among which the Binomial Deviance) are "margin maximizing loss function" *i.e.* the limit of $\boldsymbol{\beta}_\lambda$ in (29) as $\lambda \to 0$ defines the optimal separating hyperplane. For a formal proof of this fact we refer to [31]. $\square$

Proof of Prop 1. The result follows by a direct calculation

$$\begin{aligned}
&\arg\min_{i=1,\dots,N} \Bigg\{ \underbrace{\sum_{i=1}^{N} \log\left(1 + e^{-y_i f(\mathbf{x}_i)}\right) + \lambda\|\boldsymbol{\beta}\|^2}_{=\text{constant}} - \\
&\qquad - \log\left(1 + e^{-y_i f(\mathbf{x}_i)}\right) - \lambda\|\boldsymbol{\beta}\|^2 \Bigg\} = \\
&= \arg\min_{i=1,\dots,N} \left\{ -\log\left(1 + e^{-y_i f(\mathbf{x}_i)}\right) \right\} \\
&= \arg\max_{i=1,\dots,N} \left\{ \log\left(1 + e^{-y_i f(\mathbf{x}_i)}\right) \right\}
\end{aligned}$$

since logarithm is a strictly increasing monotonic function,

$$= \arg\max_{i=1,\dots,N} \left\{ 1 + e^{-y_i f(\mathbf{x}_i)} \right\}$$

since $\exp(\cdot)$ is a strictly increasing monotonic function,

$$\begin{aligned}
&= \arg\max_{i=1,\dots,N} \{-y_i f(\mathbf{x}_i)\} \\
&= \arg\min_{i=1,\dots,N} \{y_i f(\mathbf{x}_i)\} \\
&= \arg\min_{i=1,\dots,N} y_i \mathbf{x}_i^\top \boldsymbol{\beta}.
\end{aligned}$$

$\square$

Proof of Corollary 1. The distance of a point $\mathbf{x}_i$ from the separating hyperplane is

$$\text{dist}(\mathbf{x}_i, \text{SH}) = \frac{y_i(\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0)}{\|\boldsymbol{\beta}\|}. \tag{60}$$

We now note that

$$\begin{aligned}
\arg\min_{i=1,\dots,N} \text{dist}(\mathbf{x}_i, \text{SH}) &= \arg\min_{i=1,\dots,N} \frac{y_i(\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0)}{\|\boldsymbol{\beta}\|} \\
&= \frac{y_i\beta_0}{\|\boldsymbol{\beta}\|} + \arg\min_{i=1,\dots,N} \frac{y_i \mathbf{x}_i^\top \boldsymbol{\beta}}{\|\boldsymbol{\beta}\|}.
\end{aligned} \tag{61}$$

Hence thesis immediately follows from Proposition 1, noting that the contribution of every point to the Binomial Deviance is strictly positive. $\square$

Appendix B Collins-Liu feature selection method

Another approach to select color features $\mathbf{f} := [f_1, f_2, f_3]^\top$ consists of choosing a linear combination of the color features in order to increase the separability of data, *i.e.*,

$$\mathbf{f} = \mathbf{A} \cdot \mathbf{c}, \tag{62}$$

where $\mathbf{A} \in \mathscr{S}^{3\times3} \subseteq \mathbb{R}^{3\times3}$ and $\mathbf{c} \in \mathbb{R}^3$ are the original colorspace coords, *e.g.* in our case $\mathbf{c} := [H, S, V]^\top$. Moreover, for the sake of simplicity, we took $\mathscr{S} := \{-2, -1, 0, 1, 2\}$. A possible method to choose the optimal combination of color features is the one discussed in [14]. This approach can be summarized as follows

1. choose a candidate feature f;
2. compute histogram of that feature's values for pixels on the object and background, respectively, $H_+(i)$ be a histogram of that features values for pixels on the object and $H_-(i)$ be a histogram for pixels from the background sample, where index i ranges from 1 to 2^b, b the number of histogram buckets;
3. form an empirical discrete probability density $p(i)$ for the object, and density $q(i)$ for the background, by normalizing each histogram by the number of elements in it:
$$p(i) = \frac{H_+(i)}{n_+} \tag{63}$$
$$q(i) = \frac{H_-(i)}{n_-} \tag{64}$$
with n_+ and n_- being the number of object and background samples, respectively.
4. compute the log likelihood of a feature value f:
$$L(i) := \log \frac{\max\{p(i), \delta\}}{\max\{q(i), \delta\}}, \tag{65}$$
where δ is a small value that prevents dividing by zero or taking the log of zero;
5. compute the *variance ratio* $\text{VR}(L; p, q)$ of L in order to quantify the separability of object and background classes under feature f,
$$\text{VR}(L; p, q) := \frac{\text{Var}(L; (p+q)/2)}{\text{Var}(L; p) + \text{Var}(L; q)}, \tag{66}$$

which is the total variance of L over both object and background pixels, divided by the sum of the within class variances of L when object and background pixels are treated separately. It is worth pointing out that given a discrete probability density function $a(i)$, it can be used the equality $\mathrm{Var}(\mathbf{x}) := \mathbb{E}\mathbf{x}^2 - (\mathbb{E}\mathbf{x})^2$ to define the variance of $L(i)$ with respect to a as

$$\mathrm{Var}(L; a) = \sum_i a(i)L^2(i) - \left(\sum_i a(i)L(i)\right)^2 . \qquad (67)$$

The intuition behind the variance ratio is that we would like log likelihood values of pixels on the object and background to both be tightly clustered (low within class variance), while the two clusters should ideally be spread apart as much as possible (high total variance). The denominator enforces that the within class variances should be small for both object and background classes, while the numerator rewards cases where values associated with object and background are widely separated.

We exploit this approach to select the three most discriminative features that maximize the variance ratio. We note that using these features in some cases the discriminability of data slightly increases, sometimes, however, this improvement is not so significant. An example of application of Collins-Liu method is depicted in Fig.19, here the matrix $\mathbf{A}$ is

$$\mathbf{A} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} . \qquad (68)$$

Appendix C Other adaptive threshold solutions

In this appendix we review some of the strategies we implement to realize an adaptive threshold. A comparison of the various solutions is also provided.

A simple heuristic adaptive threshold

Noting that the algorithm slows down during the first frames, a really simple and naïve approach to update the adaptive threshold is to change threshold value according to a suitable monotonic increasing function, *e.g.*

$$h(x_{th}) = A\left(1 - \frac{B}{A}e^{-\frac{x_{th}}{\tau}}\right) . \qquad (69)$$

Using this method in the initial frames the threshold takes small values while as the number of frames increases the transient term disappears and $x_{th} \to A$. There are many drawbacks in using such a strategy, first of all the threshold is not genuinely adaptive since in function $h(x_{th})$ there are parameters to tune on a experimental basis (A, B coefficients and τ time constant in (69)) to optimize the trade-off between computational speed and good classification of object/background points. Maybe the only advantage of this strategy lies on its simplicity and particularly it does not increment the overall complexity of the algorithm.

The "convex-hull" threshold

In this strategy we use two results. The first can be seen as a corollary of Theorem 2.

Corollary 2 (Thm 2)**.** *Since the equivalence of primal SVM problem (23) and penalization formulation of SVM (29) the only terms that influence the solution of the minimization problem in (29) are the Support Vectors.*

Proof. Writing down the terms in SVM penalization problem (29) explicitely

$$\min_{\beta,\beta_0} \left\{ \sum_{i=1}^{N} \left[1 - y_i(\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0)\right]_+ + \lambda\|\boldsymbol{\beta}\|^2 \right\} , \qquad (70)$$

we notice that the minimizing hyperplane parametres $\boldsymbol{\beta}^*$ and β_0^* correspond to quantities $\bar{\boldsymbol{\beta}}$ and $\bar{\beta}_0$ which minimize primal SVM problem (23), since these two problem are equivalent and the solutions are in one-to-one correspondence. Hence proof follows immediately knowing that the only terms which give a non-zero contribution to the determination of $\bar{\boldsymbol{\beta}}$ and $\bar{\beta}_0$ are Support Vectors. □

The other mentioned result is a well-known result in SVM theory.

Proposition 2. *Consider the linearly-separable SVM problem stated in (7). The interior points in the convex hulls (here denoted as $\mathfrak{H}(\cdot)$) of the two data set $\mathfrak{D}_+ = \{\mathbf{x}_i, y_i = +1\}$ and $\mathfrak{D}_- = \{\mathbf{x}_j, y_j = -1\}$ are not Support Vectors.*

Proof. Since Support Vectors in the linearly separable case are characterized as the nearest point to Optimal Separating Hyperplane, these points lie on the margin of the convex hull and satisfy $y_i(\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0) = 1$. Thus they can not be interior points of the convex hull of the two data training sets. □

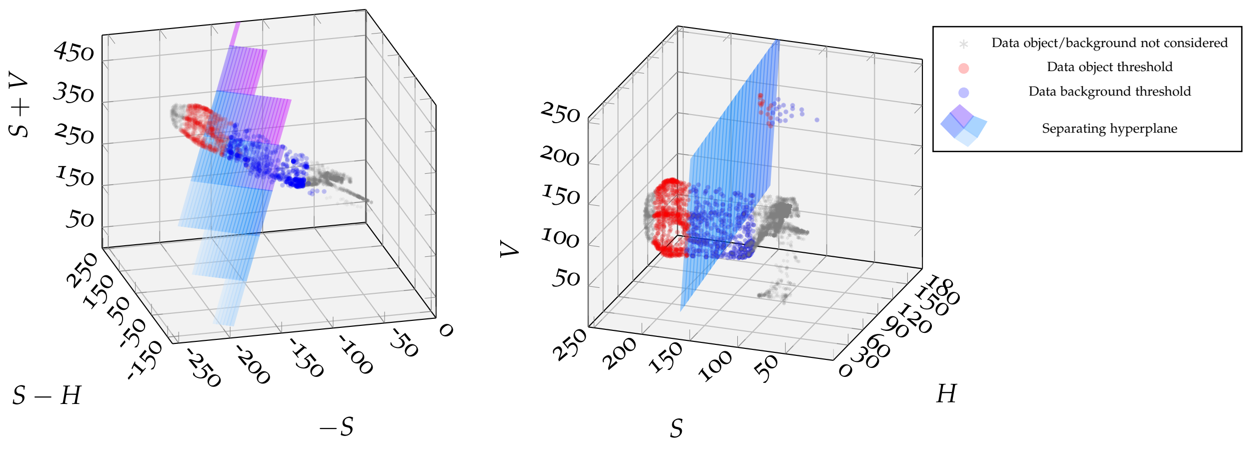
Now we note that in our case,

- ✦ during the unsupervised part of the algorithm, we dealt with a linearly-separable SVM problem, since before minimization the data are classified using the hyperplane computed in the previous step;
- ✦ the result stated in Corollary 2 it is likely to apply in an approximate way if we replace Hinge Loss function with Binomial Deviance in (29).

Under these assumptions, a strategy used in order to make threshold adaptive is described by Algorithm 4.

The algorithm can be summarized as follows:

1. Separate data according to label values ($\mathcal{O}(N)$);
2. Compute the convex hull for each data sets ($\mathcal{O}(\max\{|\mathfrak{D}_+|, |\mathfrak{D}_-|\})$ using *quick-hull algorithm* [4] or worse using other algorithms);
3. Consider in the final reduced data set S only those data whose distance from hyperplane is lower than a fixed maximum value ($\mathcal{O}(|\mathfrak{H}(\mathfrak{D}_+) \cup \mathfrak{H}(\mathfrak{D}_-)|)$).



(a) Collins-Liu features (b) standard HSV features

Figure 19: *Features selected using Collins-Liu method* (a) *vs. standard HSV features* (b).

Algorithm 4 Convex-hull threshold

```
1: $\mathfrak{D}_+ = \emptyset$, $\mathfrak{D}_- = \emptyset$;
2: for $i = 1$ to $N$ do
3:     if $y_i = +1$ then
4:         $\mathfrak{D}_+ = \mathfrak{D}_+ \cup \{\mathbf{x}_i\}$;
5:     else
6:         $\mathfrak{D}_- = \mathfrak{D}_- \cup \{\mathbf{x}_i\}$;
7:     end if
8: end for
---
9: Compute $\mathfrak{H}(\mathfrak{D}_+)$ and $\mathfrak{H}(\mathfrak{D}_-)$;
10: $S = \emptyset$;
11: for all $i \in \mathfrak{H}(\mathfrak{D}_+) \cup \mathfrak{H}(\mathfrak{D}_-)$ do
12:     if $|f(\mathbf{x}_i)| < MAX$ then
13:         $S = S \cup \{\mathbf{x}_i\}$;
14:     end if
15: end for
```

The main adavantage of this algorithm consists in considering only a very small number of data in the minimization (and in particular only those that are more "informationally relevant"). However this method increases the computational complexity of the algorithm (due in particular to the computation of the convex hull) and in case of a high number of data could deteriorate the algorithm computational speed. Moreover since the Binomial Deviance is an approximation of the Hinge Loss function the Corollary 2 approximately holds and the result in the classification of data could not be acceptable.

A general comparison of various threshold solutions based on C++ tests is illustrated in Tab.1. It can be seen that the best solution is the "nearest-hyperplane" data-threshold, which actually is the solution adopted in the SVC tracking algorithm.

	Few # Points	Low complexity	Accuracy
Nearest-hyperplane	tunable	✓	✓✓
Convex-hull	✓✓	✘	✘
Heuristic	✓	✓✓	✘

Table 1: *Comparisons of performance of different adaptive threshold solutions.*

References

[1] S. Avidan. Support vector tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(8):1064 –1072, aug. 2004.

[2] Shai Avidan. Ensemble tracking. In *In CVPR*, pages 494–501, 2005.

[3] F. Baldo and L. Carozza. Real-time visual tracking: applicazioni del filtraggio alla Kalman e ottimizzazione mediante Newton-Raphson, 2011-12. Available online at `http://automatica.dei.unipd.it/tl_files/utenti/lucaschenato/Classes/PSC11_12/Projects/PSC_11-12_Visual%20Tracking_relazione.pdf`.

[4] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, December 1996.

[5] Marcelo Bertalmìo, Guillermo Sapiro, and Gregory Randall. Morphing active contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7):733–737, 2000.

[6] Michael J. Black and Allan D. Jepson. Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. In *International Journal of Computer Vision*, pages 329–342, 1998.

[7] N. Bouaynaya, Wei Qu, and D. Schonfeld. An online motion-based particle filter for head tracking applications. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 2, pages 225 – 228, 18-23, 2005.

[8] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[9] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[10] Gary R. Bradski. Computer vision face tracking for use in a perceptual user interface, 1998.

[11] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, June 1998.

[12] Claudette Cedras and Mubarak Shah. Motion-based recognition: A survey. *Image and Vision Computing*, 13:129–155, 1995.

[13] A. Chella, H. Dindo, and I. Infantino. A system for simultaneous people tracking and posture recognition in the context of human-computer interaction. In *Computer as a Tool, 2005. EUROCON 2005.The International Conference on*, volume 2, pages 991 –994, nov. 2005.

[14] Robert Collins, Yanxi Liu, and Marius Leordeanu. On-line selection of discriminative tracking features. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 27(10):1631 – 1643, October 2005.

[15] Dorin Comaniciu, Peter Meer, and Senior Member. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619, 2002.

[16] Richard Courant and David Hilbert. *Methods of Mathematical Physics*, volume 1. Interscience Publishers, Inc., New York, NY, 1953.

[17] Nello Cristianini and John Shawe-Taylor. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press, New York, NY, USA, 2000.

[18] G. Gottardo, A. Lanzini, and C. Zanin. Applicazioni di tecniche di machine learning per problemi di real-time tracking in reti di videosorveglianza, 2010-11. Available online at `http://automatica.dei.unipd.it/tl_files/utenti/lucaschenato/Classes/PSC10_11/PSC11_Gruppo6_relazione.pdf`.

[19] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, corrected edition, July 2003.

[20] B. Heisele and C. Woehler. Motion-based recognition of pedestrians. In *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, volume 2, pages 1325 –1330 vol.2, aug 1998.

[21] D. P. Huttenlocher, J. J. Noh, and W. J. Rucklidge. Tracking non-rigid objects in complex scenes. In *Proceedings IEEE International Conference on Computer Vision*, pages 93–101, 1993.

[22] Michael Isard and Andrew Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29:5–28, 1998.

[23] Anand Singh Jalal and Vrijendra Singh. The state-of-the-art in visual object tracking. *Informatica*, 36:227–248, 2012.

[24] Omar Javed and Mubarak Shah. Tracking and object classification for automated surveillance. In *Proceedings of the 7th European Conference on Computer Vision-Part IV*, ECCV '02, pages 343–357, London, UK, UK, 2002. Springer-Verlag.

[25] Zhen Jia, A. Balasuriya, and S. Challa. Recent developments in vision based target tracking for autonomous vehicles navigation. In *Intelligent Transportation Systems Conference, 2006. ITSC '06. IEEE*, pages 765 –770, sept. 2006.

[26] H. Levkowitz. *Color Theory and Modeling for Computer Graphics, Visualization, and Multimedia Applications*. International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1997.

[27] Ching-Po Lin, Jen-Chao Tai, and Kai-Tai Song. Traffic monitoring based on real-time image tracking. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 2, pages 2091 – 2096 vol.2, sept. 2003.

[28] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[29] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, August 2000.

[30] Massimiliano Pontil and Alessandro Verri. Properties of support vector machines. *Neural Computation*, 10:955–974, 1998.

[31] Saharon Rosset, Ji Zhu, and Trevor Hastie. Margin maximizing loss functions. In *In Advances in Neural Information Processing Systems (NIPS) 15*, page 16. MIT Press, 2003.

[32] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[33] Anton Schwaighofer and Volker Tresp. The bayesian committee support vector machine. In *Proceedings of the International Conference on Artificial Neural Networks*, ICANN '01, pages 411–420, London, UK, UK, 2001. Springer-Verlag.

[34] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593–600, 1994.

[35] Cees G.M. Snoek and Marcel Worring. Multimodal video indexing: A review of the state-of-the-art. *Multimedia Tools and Applications*, 25:5–35, 2003.

[36] P. Sollich. Probabilistic interpretations and bayesian methods for support vector machines. In *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470)*, volume 1, pages 91 –96 vol.1, 1999.

[37] Vladimir Vapnik. *Estimation of dependencies based on empirical data*. Springer Series in Statistics. Springer-Verlag, New York, 1982.

[38] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[39] C. J. Veenman, M.J.T. Reinders, and E. Backer. Resolving motion correspondence for densely moving points. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:54–72, 2001.

[40] Hanxuan Yang, Ling Shao, Feng Zheng, Liang Wang, and Zhan Song. Recent advances and trends in visual tracking: A review. *Neurocomput.*, 74(18):3823–3831, November 2011.

[41] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4), December 2006.

[42] Huiyu Zhou, Yuan Yuan, Yi Zhang, and Chunmei Shi. Non-rigid object tracking in complex scenes. *Pattern Recogn. Lett.*, 30(2):98–102, January 2009.

[43] Shaohua Zhou, Rama Chellappa, and Baback Moghaddam. Visual tracking and recognition using appearance-adaptive models in particle filters. *IEEE Transactions on Image Processing*, 13:1434–1456, 2004.