University of Liege
Faculty of Applied Science
Department of Computer Science

Real-time Simultaneous Modelling and Tracking of Articulated Objects

Arnaud Declercq

June 2012

PhD thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy (Computer Engineering).

Abstract

In terms of capability, there is still a huge gap between the human visual system and existing computer vision algorithms. To achieve results of sufficient quality, these algorithms are generally extremely specialised in the task they have been designed for. All the knowledge available during their implementation is used to bias the output result and/or facilitate the initialisation of the system. This leads to increased robustness but a lower reusability of the code. In most cases, it also majorly limits the freedom of the user by constraining him to a limited set of possible interactions.

In this thesis, we propose to go in the opposite direction by developing a general framework capable of both tracking and learning objects as complex as articulated objects. The robustness will be achieved by using one task to assist the other. The method should be completely unsupervised with no prior knowledge about the appearance or shape of the objects encountered (although, we decided to focus on rigid and articulated objects). With this framework, we hope to provide directions for a more difficult and distant goal: that of completely eliminating the time consuming prior design of object models in computer vision applications. This long term target will allow the reduction of the time and cost of implementing computer vision applications. It will also provide a larger freedom in the range of objects that can be used by the program.

Our research focuses on three main aspects of this framework. The first one is to create an object description effective on a wide variety of complex objects and able to assist the object tracking while being learnt. The second is to provide both tracking and learning methods that can be executed simultaneously in real-time. This is particularly challenging for tracking when a large number of features are involved. Finally, our most challenging task and the core of this thesis, is to design robust tracking and learning solutions able to assist each other without creating counter-productive bias when one of them fails.

Acknowledgments

I would like to thank all the people who made this work possible.

My supervisor, Justus H. Piater, for his precious guidance through my research and his availability despite the distance.

Professor Jacques Verly and Jonathan Hall for their thorough proofreading.

My wife for her support and love that kept me motivated until the end.

My parents and friends for all those marvelous moments that helped me to relax and for the numerous discussions that inspired my work.

Finally, I would like to thank the Belgian National Fund for Research in Industry and Agriculture (FRIA) for believing in me and funding my research.

Contents

Chapter 1

Introduction

In this chapter, we present the objectives and motivations of this thesis. We also present the surprisingly small amount of work that has already been done on simultaneous learning and tracking (existing research on articulated models, tracking, and learning will be presented respectively in Chapters 2, 3, and 4). We then define the challenges addressed here and how we approach them. Finally, we give a brief overview of the structure of this thesis.

1.1 Objectives

The goal of this thesis is to develop a framework for tracking articulated objects while modelling them in real time. The method should be completely unsupervised with no prior knowledge of the appearance or shape of the objects (although we decided to focus on rigid and articulated objects).

More specifically, we seek to learn object models from real-time videos provided by a webcam and directly use these models to assist the tracking of the corresponding objects. This objective raises a series of questions such as:

- *Can a model of an object (although incomplete) assist its own tracking while being learnt?*
- *Is it possible to simultaneously learn and track objects in real-time?*
- *How to define a model flexible enough to (a) adapt to a wide variety of complex objects and (b) evolve during its learning?*

By answering these questions, we hope to provide directions for a more difficult and distant goal: to completely eliminate the time consuming design of object models from the development of computer vision applications. This long term target will allow the reduction of the time and cost of vision applications implementation. It will also provide more freedom in the range of objects that can be used to interact with the program.

1.2 Motivation

The human visual system is probably the most amazing sense we have. It allows us to discover, recognise, and interact with our surroundings in very challenging situations. When an unknown object appears, we are usually able to follow it and mentally separate it from its surrounding without any apparent effort. Even in a situation where well known objects suddenly appear in unexpected configurations (like an arm bent in the wrong direction or fixed element of the decor suddenly moving), our brain quickly adapts. As a result, following these particular objects quickly becomes as natural as following any other well known object.

If we compare state of the art computer vision research with the human visual capability, the difference in performance is outstanding. Despite nearly half a century of research and some significant results in specific applications, computers still struggle to perform tasks that are easily done by humans. Consequently, any real-time computer vision application is usually limited to a very specific task and uses as much prior knowledge as possible to reduce the computational load. This strongly limits the re-usability of existing programs and often means that each new software must develop its own specific models from scratch. Another weakness of this approach is the complete lack of flexibility of the applications produced. While any unexpected event or object configuration usually causes tracking to fail dramatically, unknown objects are simply ignored entirely.

In the gaming industry, where more elaborate human-computer interaction is always a selling point, a new wave of prototypes offering more freedom to the user has appeared in the last few years. Sony, for example, released a program for the PlayStation 3 called *EyePet* at the end of 2009. This program allows the users to interact with a virtual pet using drawings or common objects. Similarly, Microsoft has recently launched the *Kinect* for the *XBox360* to provide a controller-free gaming experience (games controlled with the body) and even enable the user to include personal objects (at least their appearance) into the game. On the open-source side, CamSpace have proposed, since 2008, a free application that uses everyday products to control a series of mini games. Even if these three applications actually provide a very limited freedom on the objects that can be used, they clearly express a desire to go further in terms of freedom for the user.

Another domain where a limited database of objects is a real handicap is robotics. An ability to adapt and learn from its environment would make the robot a useful tool in a variety of important applications, ranging from planetary exploration to elderly care. For these applications, it is obviously impossible to provide detailed prior models of all the objects that might be encountered. Consequently, an autonomous robot has to continuously acquire perceptual information about the world to successfully execute manipulation tasks in unstructured environments.

With this in mind, an ideal solution would be to include in every vision program an on-line learning package able to constantly update the database

of known objects. This would simplify the program design by removing any flexibility constraint(s) and low level tasks. Obviously this is an out-of-reach goal for a single thesis but this work seeks to provide a few steps in this direction.

1.3 Related Work

Surprisingly very little research has been done on simultaneous learning and tracking of object models. On-line tracking is often addressed by using a prior model of the object. Unsupervised learning of complex object models is computationally demanding and thus processed off-line. The main problem with these off-line learnt models is that they usually do not encompass all the possible variations of the object appearance. This causes the tracking to fail in uncontrolled contexts.

Attempts to improve the classic Lucas-Kanade tracker [56] with on-line updates of the model were recently made by Matthews et al. [59]. They proposed an algorithm to update the appearance of a template defined at the first frame while avoiding the drift problem of the naive update algorithm (template updated every image). Their solution is quite simple: the template is updated with the last observation if the tracking was considered successful. If not, the old template is kept. A more sophisticated solution was proposed by Lim et al. [54] with an appearance-based tracker that incrementally learns a low dimensional eigenbasis representation of the object appearance. With time, evidence of the variability of the object is acquired and added to the model. Even if it requires an initial bounding box, their method has the advantage to provide an object model that becomes more robust over time. However, their solution is limited to mostly rigid objects. It would indeed require a large number of clusters to represent all the possible appearance variations of an articulated object. Downson and Bowden [29] presented an N-tier hierarchical model that represents the object with a hierarchical set of visual features and their relative spatial positions. On the one hand, the model requires a manual initialisation of the features and offers no possibility to represent complex spatial relations. On the other hand, the combination of a graphical model and local visual features proved to be a good solution for modelling and tracking objects.

Local visual features are useful to extract objects from a video without a manual initialisation. Rigid feature graphs have already been used to simultaneous learn and track small objects. Leordeanu et al. [52] proposed to detect rigid objects in a video by grouping features with coherent spatial relations. Tang et al. [79] used a dynamic feature graph learnt on-the-fly to improve the tracking. They focus on dynamically adding and deleting features from the graph and on tracking using a relaxation labelling for graph matching. Yin et al. [91] implemented a persistent tracker that builds a set of view-dependent appearance models adaptively and automatically while tracking an object under different viewing angles. The main drawback of these three methods is their limitation to rigid objects or objects small enough for their features displacements to appear coherent.

Another domain where tracking and modelling are simultaneously applied to rigid elements is the *Simultaneous Localisation and Mapping* (SLAM) problem [31, 32]. The idea of SLAM is to place a robot at an unknown location in an unknown environment. The robot is then asked to incrementally build a consistent map of this environment while simultaneously determining its location within this map. Both the trajectory of this robot and the location of all extracted visual features are estimated on-line without the need for any prior knowledge. It is interesting to notice that solutions to this problem have existed since 1995, which is far earlier than any of the work discussed above. The problem is in fact much simpler since all the observed features are always correlated (the entire environment is considered as fixed).

Unsupervised learning of an articulated model from a video sequence usually relies on existing feature trajectories that are processed off-line [89, 69]. Ramanan et al. [66] proposed an off-line unsupervised method that simultaneously discovers, tracks, and learns articulated models of animals from videos. Unfortunately, their method is slow and requires the whole video to be treated as a block, making it impossible to adapt to an on-line process. Krahnstoever et al. [51] presented an automatic on-line acquisition and initialisation of articulated models. Their method extracts independently moving surfaces and tracks them using Expectation-Maximisation [27] during a short initialisation period. An articulated model is extracted from these motions and then used for tracking in subsequent images. This requires that all the parts of the model be visible as well as in motion in the initialisation period selected for segmentation. Finally, Droin et al. [30] developed a method to incrementally segment the rigid parts of an object on-line. Their solution keeps a set of possible models learnt from the previous frames and uses them for tracking. Although interesting, their technique suffers from two main drawbacks: it requires the extraction of a foreground area and neither learns or uses any spatial relations during the tracking.

1.4 Challenges and Contributions

As we have seen in the previous section, real-time simultaneous modelling and tracking of articulated objects is still an open problem. To address it, we decided to focus on the following challenges in this thesis:

General Model. Since we have no prior knowledge of the object appearance or structure, the model must be able to efficiently represent a wide range of complex objects. While the graphical model has become a classical solution to object structure modelling, the choice of possible visual features to represent the object appearance is vast and provides no ideal solution. For example, corner point features have been a popular choice but they are not useful for uniform and curved objects (consider for example a hand or a plate). Conversely, colour blobs or edges can be extracted from every object but colour blobs are not robust to illumination changes and edges are subject to clutter problems. Surfaces covering each rigid part of the

object are usually tracked more easily (either as image templates or as histograms) but they require a manual initialisation. We will explore in detail the choice of possible local features and propose a two-level graphical model that uses local features at one level to assist the extraction of rigid blocks at the other.

Real-time Tracking. Tracking huge graphical models connecting hundreds of visual features in real-time is probably one of the biggest challenges in this work. Each local feature is too small to provide a robust tracking on long sequences. Information must therefore be propagated through the graph so that each feature can benefit from its surrounding to improve its tracking result. The common solutions to this problem are *Belief Propagation* (BP) and its variations but none of them are fast enough to deal with big feature graphs in real-time. Consequently, we will propose a new information propagation scheme called *Affine Warp Propagation* that is much faster than BP but still provides a robust tracking solution.

Uncertain Learning. Given that there is no learning phase prior to tracking, very little information is available when the model is used for tracking in the first few frames (particularly if the object is not rigid). This means that during a more or less long period of time, the model used by the tracker will be incomplete or inaccurate. Used without precaution, this kind of model will generally cause the tracking to fail. We will present an *uncertain* model [25, 26] of relation between features that explicitly accounts for its uncertainty. Thus, a relation will contribute stability to tracking without exerting overly strong bias that would hamper tracking. Two types of *uncertain* models will be used: one for the rigid relations between features or between features and rigid blocks, and the other for articulated relations between rigid blocks.

Real-time Learning. Spatial relations learning and rigid parts discovery will also have to work in real-time if they need to be processed simultaneously to tracking. In order to respect this constraint, we will present learning and segmentation algorithms with a computational time negligible compared that for tracking, leaving as much time as possible to the tracking. Note that by real-time, we mean at least ten frames per second on a modern personal computer. As a reference for this thesis, we used a laptop with an Intel Core 2 T7200 processor clocked at 2GHz (from 2006).

Stability Issues. Tracking and learning depend on each other. It is clear that a mistake made by one of them will have an influence on the results of the other. When tracking can provide an accurate estimation of its output reliability, instability issues can usually be avoided. Unfortunately this is generally not the case in challenging situations. We will highlight these scenarios and present solutions to prevent errors from either tracking or learning to lead to a complete failure of the system.

1.5 Outline

The thesis is organised as follows: Chapter 2 starts with a definition of the graphical model used in this thesis. For the nodes of the graph, different possible visual features are presented and compared. For the edges of the graph, we provide a definition of the rigid and articulated spatial relations that will be used in the model (their learning being kept for later).

Chapter 3 focuses on tracking. It first presents a literature review of the different *Belief Propagation* solutions and gives a detailed insight into their respective solution for reducing the computational time. Our alternative solution called *Affine Warp Propagation* is presented and its results compared to those of the other methods. Experiments in this chapter focus on real-time tracking and are made with the assumption that the model is already learnt.

Chapter 4 presents rigid and articulated relation learning as well as the dynamic creation and removal of nodes. This chapter also presents the dynamic process used to detect and split non-rigid blocks. The experiments are conducted assuming that the tracking provides a good estimate of its reliability.

Finally, Chapter 5 deals with the stability issues due to erroneous feedback from either tracking or learning and proposes solutions using the available information. The experiments of this chapter focus on the robustness of the system and the communication between its different parts.

Chapter 2

Object Model Definition

In any computer vision application, the first question we have to ask ourselves is "How to describe the relevant content of an image or a video?". Obviously, the answer to that question is strongly correlated to the task at hand. If the goal is to detect and/or track a specific object (or body), a carefully designed model of that object will probably provide the most robust solution. If one is only concerned by the positions or movements of the objects, it is probably better to use foreground/background segmentation or motion segmentation. If the goal is to discover objects with no prior knowledge about them, extracting visual features and analysing their correlation is one approach to consider. In any situation, this decision will strongly influence the strategy to adopt in the rest of the application and must therefore be consider with great care.

In order to approach all the possible solutions with a clear idea of the specifics of our particular task, let us first review its requirements:

- **Large Range of Objects and Appearances**. Since our goal is to discover and track all the objects in the scene, it is extremely important for our model to describe all of them properly. For example, if local visual features are used (such as corner points or salient points), it is mandatory that they cover all the parts of the objects. We should also keep in mind that we do not impose any restriction on the type of images produced in the video. Those can, for example, be infra-red images or depth images instead of the classic colour images.

- **Model for Rigid and Articulated Objects**. Although we seek to provide a model as general as possible, we decided to focus on rigid and articulated objects. Our model might be able to represent flexible objects too, but it will be with less fidelity and robustness than a model especially designed for that type of object. The reason for our choice is twofold: first the existence of rigid parts will certainly simplify the creation of a global model and, second, we consider the rigid and articulated objects as the most relevant ones in human-computer interaction and robotic.

We therefore want to make sure that their description is as effective as possible.

- **Unsupervised Learning**. We cannot assume that there will be someone able or willing to highlight the objects of interest. It is therefore important that no initialisation phase like bounding boxes drawing or object positioning be required. For the same reason, we assume that no manual corrections to the model will be provided.
- **Robust Tracking**. Being able to learn a model of an object would be useless if this model cannot be used to track the object robustly.
- **Real-time Context**. Working in a real-time context, it is mandatory that the model require as little computational time as possible. This requirement is probably the most difficult to achieve but also the most important. Any method that is computationally expensive should therefore be discarded, even if it provides better results.

Each of these requirements has already been addressed individually in the literature [30, 39, 66, 78, 7][30, 39, 66, 78, 7]. Our main task will thus be to address them all together. Two requirements in particular tend to lead toward opposite solutions: *unsupervised learning* and *robust tracking*. *Unsupervised learning*, when applied to articulated objects, is usually fulfilled by using small elements. This guarantees that each of them corresponds to a rigid part of the object. *Robust tracking*, usually requires using as much information as possible (such as largest possible surfaces) to reduce the risk of clutter.

Keeping these requirements in mind, we will first present in Section 2.1 an overview of the existing solutions used to represent objects. We will then see in Section 2.2 how some of them can be combined to provide a model that fulfils the above requirements.

2.1 Background

There is a large variety of existing models from simple templates to fully articulated models. It is difficult to give an exhaustive list of all the possible models but the most common model categories can be listed as follows.

Primitive Geometric Shape The simplest way to represent an object is probably to define a bounding box (or ellipse, or any basic shape) around it and to build the model using the pixels inside this bounding box. If the object of interest is completely rigid, the model can be reduced to the image patch inside the bounding box (the template). This approach is extremely fast and is therefore often used to describe local visual features [4, 72]. In order to be more robust to illumination variations, the template can also be based on edges instead of colour [35, 76]. If the object is not rigid, the pixel grid is usually replaced by a description of its pixel distribution such as a colour or gradient histogram of the pixels inside the bounding box [17, 21].

(a) Original. (b) Primitive Shape. (c) Active Shape. (d) Constellation.

(e) Original. (f) Active Contour. (g) Skeleton. (h) Part-Based Articulated.

Figure 2.1.1: Examples of possible models

While this model requires very little computational time, it also offers a very limited description of the object since it is incapable of describing the structure of any non-rigid object. This being said, it can be a good solution to represent the rigid parts of an articulated object.

Active Contour The active contour model [10, 49] (also often called *snake*) is defined as a contour that can be deformed to lock onto features of interest within in an image (usually lines, edges, or object boundaries). In practice, the active contour is represented by an ordered collection of points in the image plane. These points iteratively approach the boundary of an object through the solution of an energy minimisation problem. The energy to minimise for each point is a weighted combination of two energy terms. The first term, called the external energy, is defined by the distance between the contour points and the features of interest. The second term, called the internal energy, is intended to control the shape on the deformable contour and the distance between its points. Generally, the energy is minimal if the contour is short and smooth. This means that the minimal shape is a circle if the contour is closed and a short line if not (a single point if the internal energy is minimal for a null distance between points).

This model is perfect to track unknown non-rigid objects but it does not provide a sufficient description. It could be used as a basis to assist the construction of a more complex model but its low robustness to clutters and its high tendency to get stuck in local minima makes it an even less appealing candidate.

Point Distribution Model The point distribution model is a model capable of representing a non-rigid object using the mean geometry of its shape and a statistical description of its variations around that mean shapes. Two common models belonging to this category are the Active Shape Model (ASM) [19] and the Active Appearance Model (AAM) [18].

An ASM can be seen as an extension of the active contour model where the single predefined shape is replaced by a statistical shape model learnt from a set of exemplar shapes (training set). This means that the active shape can only deform in directions that fit the training set. To define the statistical shape model, each exemplar shape is first represented with a $2n$ elements vector made from the aligned positions of n labelled points along the shape. Principal Component Analysis (PCA) is then applied to this set of vectors to pick out the main modes of variation of the training set. These modes correspond to the eigenvectors of the covariance matrix across all training shapes. The resulting statistical shape model is then defined by $x = \hat{x} + Pb$ where $\hat{x}$ corresponds to the mean shape, P is a $2nt$ matrix whose columns are the t eigenvectors with the highest eigenvalue of the training set, and b is a t element vector of weights used as shape parameters. By varying the shape parameters b within limits learnt from the training set, the active shape model can adapt to the object in the current image.

An AAM is similar to an Active Shape Model but is using an image template of the inside of the object instead of contours. Both of these methods are more robust to variation in the shape of the object than the active contour but require a costly training phase in order to generate the statistical model. The most successful results using ASM and AAM have been obtained for the human face where the variations in the shape are localised and simple.

Skeleton The skeleton is the dual of the contours. If the silhouette of an object can be extracted, then the skeleton is simply defined as its medial axis [3, 81]. Skeleton representation can be used to model both articulated and rigid objects and seems therefore to be a strong candidate for our application. The main problem with the skeleton is its strong dependency on the silhouette: a small change in the latter can cause large changes in the skeleton.

In a scenario like ours where there is absolutely no guarantee that a robust silhouette can be extracted, it is therefore very risky to use a model as unstable as the skeleton.

Constellation Model The constellation model represents an object by a set of N features or parts under mutual geometric constraints. It can be seen as a graph where the nodes correspond to the appearance of the features/parts and the edges to the spatial constraints between them. This approach presents two main advantages: the model explicitly accounts for shape variations and for the randomness in the presence/absence of features due to occlusion and detection errors. This model has been very popular for object category recognition and unsupervised learning [33, 73, 82]. The inter-class shape variations are indeed

well represented by the geometric constraints and the use of features provides a good basis for unsupervised learning. Those two advantages also make it a popular solution for tracking objects in case of frequent occlusions or even of discovering object in videos [29, 52, 74, 79].

The popularity of this model for both tracking and unsupervised learning makes it a strong candidate for our application. This being said, there are a few drawbacks that we have to keep in mind. First, the spatial constraints between features are only useful locally. For a large graph, where the shortest path between two nodes can become quite long, the uncertainty (flexibility) on the spatial constraints will indeed accumulate along the path causing its usefulness to quickly fade. This results in a larger flexibility in the model than needed and, therefore, to a lower robustness to clutter. Second, the computational time needed to propagate the information between the features makes it difficult to track constellations of hundreds of features in real-time.

Part-Based Articulated Graphical Model This model corresponds to a set of rigid body parts spatially constrained by the position of their common articulation/joint. This type of model is the more popular when it comes to tracking or detecting known articulated objects [67, 78, 87]. Thanks to its minimal number of parts, the computational cost is as low as possible. In comparison to the constellation model, the more global oriented description also tends to provide a more robust tracking result. The main drawback of this model is that it requires a correct segmentation of the object into rigid parts. While features are easy to extract, rigid body parts are usually manually defined or obtained through a long offline learning process using entire video sequences or image sets [66, 69, 89].

The articulated graphical model is ideal in term of tracking but is extremely difficult to learn in a real time on-line process. This means that, even if we want to consider it as a candidate for the output of our method, another model will be needed to assist its learning phase.

From the list of model descriptions above, we can see that there is not a single model that fulfils all our requirements. The closest to succeed is the feature constellation model due to its popularity for both tracking and unsupervised learning. In term of pros and cons, its dual is the part-based articulated graphical model that provides a more global approach and a smaller computational cost but is less adapted to unsupervised learning. Combining these two models into a single one therefore seems to be a good approach, especially since they both use a graph as their basic structure.

The two following sections focus on the definition of the graphical model and on an overview of the possible visual features. From there, we will describe in Section 2.2 how we create our own model by combining constellation and articulated models.

2.1.1 Graphical Model Definition

Graphical models provide a general and powerful method for encoding the statistical structure of a set of random variables. Their general purpose makes them convenient to represent a variety of problems in computer vision. The objective of this section is to give a short introduction to basic concepts from graph theory. More complete discussions about graphical models can be found in several books [9, 48, 53]. Graphical models can be defined as follow:

Definition 1. A *graphical model* is a probabilistic model for which a graph G denotes the conditional independence structure between random variables [86]. This graph $G = (V; E)$ is composed of a finite set of nodes V and a set of edges E. Each node $v_i \in V$ is associated to a random variable $\mathbf{x}_i$, and each edge $e_{ij} \in E$ between nodes v_i and v_j expresses a relation between the corresponding variables.

We can distinguish between two different types of graphs: directed and undirected graphs. Undirected graphical models, also called Markov Random Fields (MRFs) or Markov networks, have a simple definition of independence: two sets of nodes A and B are conditionally independent given a third set C if all paths between the nodes in A and B are separated by a node in C. By contrast, directed graphical models, also called Bayesian Networks (BNs), have a more complicated notion of independence which takes into account the directionality of the arcs. Bayesian Networks are useful for expressing causal relationships between random variables, whereas undirected graphs are better suited to expressing soft constraints between random variables [9]. Undirected graphical models are more popular with the vision community and are indeed what we will use here.

Definition 2. A *Markov Random Field* is an undirected graphical model G in which any two subsets of variables are conditionally independent given a separating subset C if every path from a node in A to a node in B passes through C.

It means that a probability distribution $p(\mathbf{x}_i)$ of a random variable $\mathbf{x}_i$ of the graph depends only on the knowledge of the outcome of its neighbours:

$$p(\mathbf{x}_i \mid X_{\setminus i}) = p(\mathbf{x}_i \mid \{\mathbf{x}_j \in \mathcal{N}_i\}), \tag{2.1.1}$$

where $X_{\setminus i}$ represents all the random variables in the graph except $\mathbf{x}_i$ and $\mathcal{N}_i$ is the set of neighbours of node i. Thanks to the Hammersley Clifford Theorem [15], the joint probability distribution $p(X)$ of a set of random variables X corresponding to a graph G can be computed as a Gibbs distribution.

Definition 3. A *clique* in an undirected graph G is a subset of the vertex set $C \in V$, such that for every two vertices in C, there exists an edge connecting the two [85]. This is equivalent to saying that the sub-graph induced by C is complete.

Definition 4. A *maximal clique* is a clique that cannot be extended to include additional nodes without losing the property of being fully connected [48].

Definition 5. A probability distribution $P(X)$ on an undirected graphical model G is called a *Gibbs distribution* if it can be factorised into positive functions defined on cliques that cover all the nodes and edges of G. That is,

$$p(X) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(X_C)\,, \tag{2.1.2}$$

where $\mathcal{C}$ is a set of all maximal cliques in G and the potential function $\psi_C(X_C)$ is a function of the possible realisations of the clique X_C [48]. Z is the normalisation constant obtained by summing the numerator over all assignments of values to the random variables set X :

$$Z = \sum_{x} \prod_{C \in \mathcal{C}} \psi_C(X_C)\,. \tag{2.1.3}$$

By considering only potential functions that satisfy $\psi_C(X_C) \geqslant 0$ we ensure that $p(X) \geqslant 0$.

Since a potential function defined over a large set of random variables may become computationally demanding in Markov Random Fields, it is convenient to consider a subclass of Markov random fields, called Pairwise Markov Random Fields (PMRFs) which is defined as follows:

Definition 6. A *Pairwise Markov Random Field* is an undirected graphical model G in which all the cliques are of size two.

It means that the set of cliques in the graph will always correspond to its set E of edges. This results in a simpler probability distribution

$$p(X) = \frac{1}{Z} \prod_{\{i,j\} \in E} \psi_{i,j}(\mathbf{x}_i, \mathbf{x}_j) \prod_{i \in V} \psi_i(\mathbf{x}_i)\,, \tag{2.1.4}$$

where $\psi_{i,j}(\mathbf{x}_i, \mathbf{x}_j)$ is the pairwise potential function between random variables $\mathbf{x}_i$ and $\mathbf{x}_j$ and $\psi_i(\mathbf{x}_i)$ provides a local prior for $\mathbf{x}_i$.

In many computer vision applications, this local prior is computed using some quantity $\mathbf{y}_i$ observed in the image. These values are expressed as a set of observable random variables Y, where each $\mathbf{y}_i$ is defined as the image likelihood of its associated hidden random variable $\mathbf{x}_i$. When observations are available, PMRFs can be used to express the structure of the posterior distribution $p(X \mid Y)$. This is done by introducing local observations $\mathbf{y}_i$ in the probability model through local potential functions $\psi(x_i, y_i)$ that link a hidden random variable $\mathbf{x}_i \in X$ to a corresponding observed random variable $\mathbf{y}_i \in Y$. The posterior distribution $p(X \mid Y)$ then factorises as [90]

$$p(X \mid Y) = \frac{1}{Z} \prod_{\{i,j\} \in E} \psi_{i,j}(\mathbf{x}_i, \mathbf{x}_j) \prod_{i \in V} \psi_i(\mathbf{x}_i, \mathbf{y}_i) \,. \tag{2.1.5}$$

In the case of tracking, $\psi_i(x_i)$ can also account for temporal information and then be decomposed into

$$\psi_{i,t}(\mathbf{x}_{i,t}) = \psi_{i,t}(\mathbf{x}_{i,t}, \mathbf{y}_{i,t}) \cdot \psi_{i,t}(\mathbf{x}_{i,t}, \mathbf{x}_{i,t-1}) \,. \tag{2.1.6}$$

Notice the index t indicating that values are given for time t. The posterior distribution $p(X_t \mid Y_t, X_{t-1})$ accounting for temporal information now factorises as

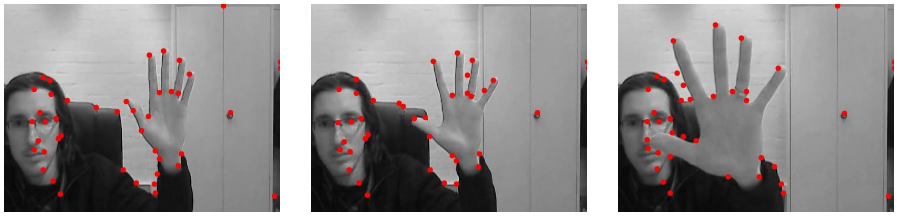
$$p(X_t \mid Y_t, X_{t-1}) = \frac{1}{Z} \prod_{\{i,j\} \in E} \psi_{i,j,t}(\mathbf{x}_{i,t}, \mathbf{x}_{j,t}) \prod_{i \in V} \psi_{i,t}(\mathbf{x}_{i,t}, \mathbf{y}_{i,t}) \cdot \psi_{i,t}(\mathbf{x}_{i,t}, \mathbf{x}_{i,t-1}) \,. \tag{2.1.7}$$

Note that, by contrast with Bayesian Networks, the choice of potential functions is not restricted to those with a specific probabilistic interpretation like marginal or conditional distributions. One consequence is that the product computed in the equation above will generally not be correctly normalised. Therefore, while the joint distribution was automatically normalised for directed graphs, undirected graphs require the explicit normalisation factor Z. This is a major limitation of undirected graphs: in the case of a graph with N discrete nodes each having M states, the evaluation of Z involves summing over NM states and is then exponential in the size of the model. However, for evaluation of local conditional distributions, the partition function is not needed because a conditional probability is the ratio of two marginals, and the partition function cancels between numerator and denominator when evaluating this ratio [48]. Similarly, for evaluating local marginal probabilities, we can work with the unnormalised joint distribution and then normalise the marginals explicitly at the end.

In the case of visual feature graphs, the hidden variables set X corresponds to the position of the features while the observable variables set Y corresponds to their image likelihood. The choice of visual features then has an impact on two levels. First, it determines where the graph nodes will be created (for example, along the edges or only on corner points). Second, it defines how to estimate the likelihood of the nodes (for example, local gradient or colour distribution). In the next section, we will introduce different possible visual features that can be considered for our model.

2.1.2 Visual Features

Visual feature is a very generic term that may correspond to any distinctive visual aspect observed in an image. It can be a colour or a texture but also a shape or a corner point. In our situation, we seek to discover and describe rigid parts of objects by grouping visual features. Since our system does not offer



(a) (b) (c)

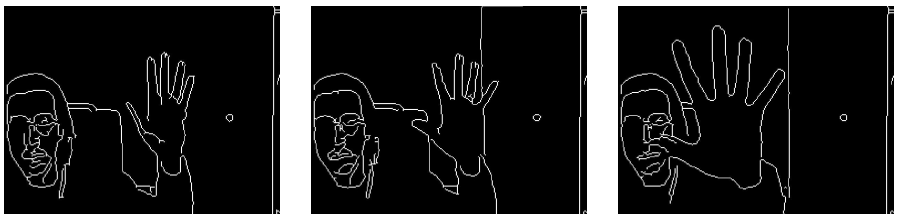
Figure 2.1.2: Example of detection of interest points using the Harris detector [40]. As we can see, textureless objects like the hand are not well covered by features points. Another problem we can observe is the creation of features on intersections between background and foreground lines (see for example the thumb and pinky on Figure 2.1.2b).

a method to fragment visual features, it is important that each of them is not used to describe more than one rigid area. Our choice is then limited to local visual features i.e. features covering a very small area of the image.

A very large number of feature detectors have been developed through the years. At an overview level, they can be divided into the following groups (with some overlap):

Corners / Interest Points The term interest points refers to a point in an image which has a well-defined position and can be robustly detected. Similarly, a corner point corresponds to the intersection of two edges and can therefore be distinguished from its direct neighbourhood. The term *interest point* is actually more general than the term *corner point* and can correspond to other structures like blobs. Despite this, *corner*s and *interest points* are sometimes used interchangeably in the literature; which can be a little confusing. So, to make things clear, we focus here on pointwise visual features. Wider "interest zones" like blobs will be discussed later.

Interest points are probably the most widely used type of feature in computer vision and many different methods have been proposed to extract them [6, 40, 55, 60, 72]. The success of interest points is due to a number of qualities such as a well-defined position, a usually rich local information content, stability under affine transformations, ... One thing that is against them though is their inability to spread evenly on objects with no or limited texture. For example, objects like the hand in Figure 2.1.2 do not contain many robust corners making these objects difficult to model with a set of interest points. Moreover, features are also created on intersections between background and foreground lines. With the number of valid foreground points quite low, this means there is a high risk to get a significant percentage of outliers in the initialisation of the model; increasing the risk for the tracking to fail (at least during the beginning of the model learning phase).



(a) (b) (c)

Figure 2.1.3: Example of edge detection using the Canny edge detector with the same images as for the Harris detector in Figure 2.1.2. While the result is not perfect, it provides a more intuitive coverage of the hand than the corners/interest points. While the hand would have indeed been difficult to identify using the corners only, it is easily recognised using the edges.

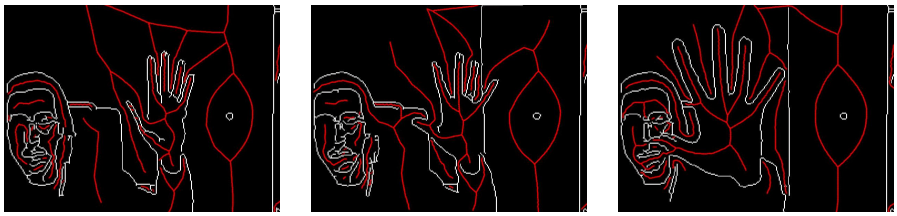
Edges Edges are locations where there is a boundary between two image regions. In practise, edges are usually defined as sets of points in the image with a strong gradient magnitude in one direction. These points can then be chained together to form a more complete description of an edge. Depending on the length of these chains and their shape, the edges are more or less local and more or less distinctive from their surrounding (the longer the chain, the higher the risk of covering more than a single rigid part, and the more complex the shape, the lower the risk to confuse it with its surrounding).

The main drawback of edges is that, locally, they are only defined in one dimension: perpendicular to the gradient. Tracking them therefore generally requires more than the local knowledge provided by a single edge. Edges are also very sensible to clutter since they don't have a descriptive power as good as the interest points. However, edges provide a far better coverage of textureless objects than interest points (see the examples in Figure 2.1.3). They also are largely invariant to lighting conditions and variations in object colour and texture, making them a better candidate to model object categories. On top of that, they can be matched accurately along the object boundary, while image patches and local descriptor vectors usually used with interest points tend to be more difficult to match when the background is changing.

Blobs/Regions of Interest In computer vision, the term *region of interest* usually refers to a region in the image that is either brighter or darker than its surrounding (in the case of colour images, this definition must be true for at least one channel). These regions can be detected using, for example, convolutions with kernels such as a Laplacian of Gaussian or a difference of Gaussians. This approach is for example used by the SIFT detector [55] which then tends to overlap with the techniques used for the interest points detectors. Another popular detector is the *Maximally Stable Extremal Region* (MSER) detector originally defined in [58]. The idea of this method is to find regions stable

(a) (b) (c)

Figure 2.1.4: Examples of region detection using the MSER detector with the same images than for the Harris detector in Figure 2.1.2. While the result is far from ideal in this case, there is one thing important to notice: the features can cover large areas that are not guaranteed to be rigid. For example some features are covering entire fingers while those should be divided in order model the flexibility of the fingers.



(a) (b) (c)

Figure 2.1.5: Examples of region detection using the pruned Voronoi diagrams of the edges obtained in Figure 2.1.3. As we can see, small changes on the contours can cause large changes on the resulting skeleton.

over a wide range of thresholding of a grey-scale image. Examples of region detection using MSER are shown in Figure 2.1.4. As we can see, the main problem with this type of detector is that it does not insure that the features surfaces are limited to rigid parts. Since this property conflicts with one of our main requirement, the region detector does not seem appropriate for our case.

Skeleton and Ridges Skeleton points or medial axis points provide an intuitive, compact representation of a shape, making them appealing for many applications in computer vision. Numerous algorithms have been developed for skeleton extraction using for example distance transform, topological thinning, Voronoi diagrams or gradient vector flow [14, 41, 61, 80]. Apart from their high sensitivity to noise in the object's boundary, the skeleton points cannot be computed directly from the raw image and require a contour or a silhouette in order to be extracted. It can therefore be used in combination with contour points [1] but is difficult to use as a feature on its own. Moreover, skeleton extraction

requires a high computational cost in order to deal with all the edges detected inside the object (and therefore not relevant for the skeleton extraction).

Ridges are defined as the local maxima of the image intensity. They are useful to detect the medial axis of elongated objects or parts. These features are not frequent in computer vision since they are highly scale sensitive and are only effective if the objects are significantly lighter (or darker in the inverted image) than they surrounding. Their main application field is in the medical sector to detect blood vessels for example.

From the list above, it appears that none of the features is exactly ideal. The most adapted for our situation are the edges because they provide a better coverage of the objects and a very good robustness to illumination and colour changes. Psycho-physical studies [8, 23] have also shown that we can recognise objects using fragments of the outline contours alone. This fact makes them a better candidate than interest points if we expect to learn comprehensive models. They have also been used successfully for tracking [43], object categories recognition [73] and motion detection [44, 70].

In the following sections, we will give a bit more details on the methods used to extract, detect, and track edge-based models.

2.1.2.1 The Canny Edge Detector

The Canny edge detector is widely considered to be the standard edge detection algorithm. It was first presented by John Canny in 1986 [13] and still outperforms many of the newer algorithms that have been developed. The edge detector can be decomposed into 4 steps as follows:

1. Since the detector uses a filter based on the first derivative of a Gaussian, it is susceptible to noise. The first step is therefore to smooth the original image with a two dimensional Gaussian before trying to locate and detect any edges. This way, the detector is more robust to single noisy pixels.

2. Calculate the gradients I_x and I_y of the image in the x and y directions using for example the Sobel operator. From them, the absolute gradient magnitude (edge strength) and direction can be determined:

$$G = \sqrt{I_x^2 + I_y^2} \tag{2.1.8}$$

$$\theta = \arctan\left(\frac{I_y}{I_x}\right) \tag{2.1.9}$$

3. Since edges occur at points where the gradient is at a maximum, all points not at a maximum should be suppressed. So each pixel is compared to its direct neighbours in the positive and negative directions along the gradient and kept as an edge only if the gradient magnitude is maximum. This non-maxima suppression step makes all edges one pixel thick.

4. The final step in Canny's edge detection algorithm is the extraction the edges using two thresholds T_{high} and T_{low}, with $T_{\text{high}} > T_{low}$. The image is scanned from left to right, and top to bottom. The first pixel in the non-maxima suppressed magnitude image with a magnitude above T_{high}, is declared as an edge. Then, all its neighbours are recursively traces using the directional information derived earlier. Those with a magnitude above the threshold T_{low} are marked as edges as well. This solution allows faint sections of edges to be traced as long as other sections are strong enough to be selected using threshold T_{high}. If needed, very short chain of edges can be discarded in a final step.

Once the edge detector has been applied to the image, the extraction of the edgels (edge elements) is fairly straightforward since it simply consists of sampling the contours obtained in a chain of equidistant points.

2.1.2.2 Edge-based Model Detection and Tracking

Template detection and tracking is always based on some sort of similarity measure between the template and parts of the image at particular locations. Consider that the model is composed of a set of n points $\mathbf{x}_i = [x_i, y_i]^{\mathrm{T}}$ and their associated direction vectors $\mathbf{d}_i = [t_i, u_i]^{\mathrm{T}}$with $i = 1, \ldots, n$. Consider also that this model is aligned with the image I using an affine transformation $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ and a translation $\mathbf{t}$ resulting on the projected positions $\mathbf{x}'_i = A\mathbf{x}_i + \mathbf{t}$ and directions $\mathbf{d}'_i = A\mathbf{d}_i$. One way to compute the similarity measure between the projected model and the image at that position is to sum the normalised dot product of the direction vectors of the transformed model and the image gradient directions I_{dir} over all points of the model [76], i.e.

$$s = \frac{1}{n} \sum_{i=1}^{n} \frac{|\langle \mathbf{d}'_i, I_{\text{dir}}(\mathbf{x}'_i) \rangle|}{\|\mathbf{d}'_i\| \cdot \|I_{\text{dir}}(\mathbf{x}'_i)\|} \,. \tag{2.1.10}$$

This measure is robust against occlusion and clutter because, if a feature is missing either in the model or in the image, noise will lead to random direction vectors, which, on average, will contribute nothing to the sum. Thanks to the normalisation of the direction vectors this measure is also invariant to arbitrary illumination changes.

Another way to estimate the similarity between the template and the parts of the image is obtained by using the *oriented chamfer distance function* proposed by Shotton et al. [73]. This function augments the classic chamfer distance function [5] with a continuous and explicit cost for orientation mismatch (making it more robust against occlusion and clutter):

$$d_\lambda(\mathbf{x}'_i) = (1 - \lambda) \cdot d_{\text{cham},\tau}(\mathbf{x}'_i) + \lambda \cdot d_{\text{orient}}(\mathbf{x}'_i, \phi'_i) \,, \tag{2.1.11}$$

where ϕ'_i and $\mathbf{x}'_i$ represent respectively the orientation (i.e. angle) and coordinates of the projected edgel i in the image coordinates. The parameter λ defines

the trade-off between the classic chamfer distance $d_{\text{cham},\tau}$ and the orientation distance d_{orient}.

The chamfer distance $d_{\text{cham},\tau}(\mathbf{x})$ can be efficiently computed via the *distance transform* (DT) which gives the distance to the closest contour points for every pixel in an image. The chamfer distance of an edgel i is then obtained using a simple look-up operation,

$$d_{\text{cham},\tau}(\mathbf{x}'_i) = \frac{1}{\tau} \min(\text{DT}(\mathbf{x}'_i), \tau) \,, \tag{2.1.12}$$

where the distance transform is truncated to a maximum value τ. Similarly, the orientation distance d_{orient} uses the *argument distance transform* (ADT) which gives the locations of the closest contour for every pixel in an image. The orientation distance is then obtained with

$$d_{\text{orient}}(\mathbf{x}'_i\phi'_i) = \frac{2}{\pi} \left|\phi'_i - I_\phi(\text{ADT}(\mathbf{x}'_i))\right| \,, \tag{2.1.13}$$

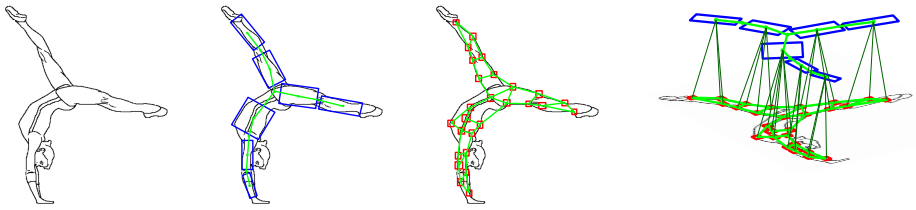
where $\phi(\mathbf{x})$ gives the orientation of edgel at $\mathbf{x}$ modulo π, $I_\phi(\text{ADT}(\mathbf{x}))$ gives the image orientation at $\mathbf{x}$ and $|\phi_1 - \phi_2|$ gives the smallest circular absolute difference between ϕ_1 and ϕ_2. Orientations are taken modulo π because, for edgels on the outline of an object, the sign of the gradient is not reliable as it depends on the intensity of the background. Normalisation by $\frac{\pi}{2}$ ensures that $d_{\text{orient}}(\mathbf{x}) \in [0, 1]$. Since the exact Euclidean DT and the ADT can be computed simultaneously in linear time and only once per image (regardless of the number of edgels), the calculation of $d_\lambda(\mathbf{x})$ for every edgel is a very fast operation.

This second similarity function is more desirable than the one defined in Equation 2.1.10 because it gives a smoother similarity function. Indeed, with the similarity function from Equation 2.1.10, the edgels must be projected exactly on the contours in order to get a positive result while with Equation 2.1.11, there is a tolerance τ on the projection result. This solution therefore requires testing fewer projections than for the first one. Moreover, the ADT already provides the position of the closest contour making the number of iterations needed for tracking far more lower (ideally one, but the position of the closest contour point does not always match the exact position of the edgel as required by the alignment of the template).

2.2 Our Model

2.2.1 A Two-Level Graphical Model

When an object is tracked using a graphical model representation, two choices are usually encountered in the literature: the nodes of the graph either correspond to visual features covering the object [79, 91] or to bounding boxes around the rigid parts [78, 87]. The first case is often related to an object discovery context while the second case usually occurs when an object model is defined by hand during the implementation of the program or during an initialisation phase. Obviously bounding boxes are a more desirable solution not only because



(a) Original Frame. (b) Rigid Blocks Graph (c) Visual Feature Graph (d) Two-Level Graph

Figure 2.2.1: Three possible representations of an articulated object using a graphical model. On the left, the original frame. In the centre, the two graphical models usually used. On the right, our two-level model where they are combined in a two-level hierarchy.

they cover the object with a minimal set of elements (implying faster tracking) but also because they provide the largest possible surface for each rigid part and therefore a more robust tracking. Local visual features however are small and therefore quite difficult to track on long sequences. However, their small size guarantees that they always cover a rigid region of the object making them suitable as an unsupervised representation.

Since our goal in this thesis is to provide both robust tracking and unsupervised modelling, we propose to use a graphical model based on two levels as shown in Figure 2.2.1: one composed of visual features and the other of potential rigid blocks. The rigid blocks are estimated by using the tracking of local features in the previous frames and by grouping them into coherent groups. Since no significant tracking result is available in the first frames, we use a single block to initialise the graphical model. This block will be divided into smaller parts once enough evidence about independent motions is accumulated. With time, these blocks will become more and more reliable and therefore more useful for tracking. By using this two-level model, we always have access simultaneously to a local and a global representation of objects; making this model robust and flexible at the same time.

More formally, on the graph G presented in Section 2.1.1, the hidden variable set X_t is now divided into the set of variables $F_t = [\mathbf{f}_{1,t}, \ldots, \mathbf{f}_{N,t}]$ corresponding to the visual features and the set of variables $B_t = [\mathbf{b}_{1,t}, \ldots, \mathbf{b}_{M,t}]$ corresponding to the rigid blocks resulting in $X_t = \{F_t, B_t\}$. The index t is used to associate the values of the hidden variables to a given time t. For both blocks and features, the hidden variables correspond to their affine positions in the current image. To every hidden variable, there correspond a node and an observable variable resulting in the set of nodes $V_t = \{V_t^f, V_t^b\}$ and the set of observable variables $Y_t = \{Y_t^f, Y_t^b\}$. Finally, the set E_t of edges is divided into three sets, i.e. $E_t = \{E_t^f, E_t^b, E_t^c\}$, where E_t^f expresses spatial relations between two visual features, E_t^b expresses relations between two blocks and E_t^c expresses relations

between one block and one visual feature. As for X_t and Y_t, V_t and E_t are indexed with t because the number of nodes and their connections will evolve with the object modelling process, and thus with time.

2.2.2 The Nodes

As discussed earlier, the nodes on the feature level are associated with edgels. The observable variable $\mathbf{y}^f_{i,t}$ of a node i corresponds to the *oriented chamfer distance function* proposed by Shotton et al. [73] and presented in Section 2.1.2.2. The hidden variable $\mathbf{f}_{i,t}$ of a node i corresponds to the affine parameters needed to align this edgels and its surrounding with the image at time t. Edgels suffer from the aperture problem and are, at best, only defined by their position and orientation. This means that, on their own, they are not able to provide a reliable value for the 6 affine parameters of their hidden variable. In order to align themselves properly in a new image, the feature nodes will then need to use the information provided by the neighbouring edgels. We will see in the next chapter how feature nodes can benefit from the largest possible rigid neighbourhood while still being directly connected only to a very limited set of close neighbours.

Similarly to the feature nodes, the hidden variable $\mathbf{b}_{i,t}$ of a block node i corresponds to the affine parameters needed to align it with the image at time t. The appearance of a rigid block is based on edges as well. This means that the appearance of the block is actually defined by its spatial relations with the feature nodes. A block can therefore be seen as being a star constellation of edgels. This is quite convenient since the constellation model demonstrates some tolerance on non-rigidity of its sub-parts. The block nodes could also be associated with a second global appearance based on colours or depth. This could be used for example to filter the image edges used for its alignment. We choose, however, to use only edges in order to be as independent as possible of the type of image used. Hopefully, with this approach, the model will be able to provide a representation of an object class that can be used with different video formats without any modification.

2.2.3 The Spatial Relations Between Nodes

The spatial relations used in our model can be divided into two groups: rigid (i.e. fixed) and articulated relations. We do not model flexible relations (i.e. neither rigid nor articulated). Relations between blocks are obviously not rigid otherwise the two blocks would count as one. Similarly, relations between a block and a feature are rigid by definition. Relations between features are rigid as well, which means that relations only exist between features associated with the same rigid block.

Rigid and articulated relations are not known a priori and it is clear that they cannot be learnt from a single frame (unless it is perfectly rigid, which can only be confirmed after an observation over a long time). When connections are created between two nodes, it is then impossible to know whether these

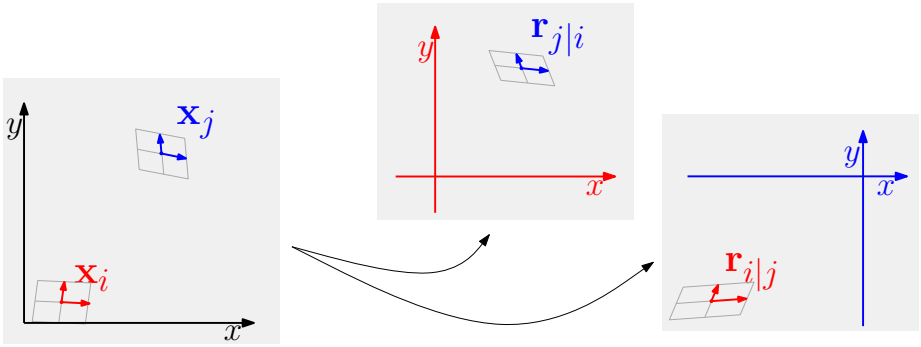


Figure 2.2.2: Relative affine positions $\mathbf{r}_{i|j}$and $\mathbf{r}_{j|i}$ of two nodes computed from their affine positions $\mathbf{x}_i$ and $\mathbf{x}_j$ observed in the image (i.e. affine alignment with the image).

nodes will respect a rigid/articulated relative configuration. A possible solution is therefore to use a representation able to model any kind of distribution, like a non-parametric distribution estimation, and discard the connections once enough evidence is accumulated that it is neither rigid nor articulated (such as between completely uncorrelated elements). Unfortunately, non-parametric representations have a computational cost too high for a real-time process. In comparison, parametric models are very fast to compute but will create a strong bias in the tracking results if the observed spatial relation does not correspond to the model type. To manage such situations, we will define in Chapter 4 an *uncertain model* of relation between two nodes that explicitly accounts for its uncertainty. To do so, the relative spatial configuration between two nodes is represented as a mixture of a parametric distribution (reliable relation) and a uniform distribution (ignorance). The relative weight of the uniform distribution is inversely proportional to the confidence that the model is able to represent the relation. Thus, a model will contribute stability to tracking without exerting overly strong bias that would hamper tracking. In the following two subsections, we introduce the rigid and articulated parametric uncertain models. Their learning is detailed later, in Chapter 4.

2.2.3.1 Model of a Rigid Relation

Since each hidden variable is associated with a feature/block position in the affine space (corresponding to 6 parameters), the potential function modelling their relative positions should be expressed in a 12 dimensional space with a potential proportional to the likelihood to observe these two respective positions simultaneously. However, we choose to not represent the relationships through pairwise potentials $\psi_{i,j}(\mathbf{x}_i, \mathbf{x}_j)$ explicitly but rather to use conditional distributions $\psi_{i,j}(\mathbf{x}_i \mid \mathbf{x}_j)$ and $\psi_{i,j}(\mathbf{x}_j \mid \mathbf{x}_i)$ (notice that $\mathbf{x}$ is used to represent the hidden variable of either a feature or a block). Indeed the inference process used in the graph for tracking (see Chapter 3 for details) only requires conditional distri-

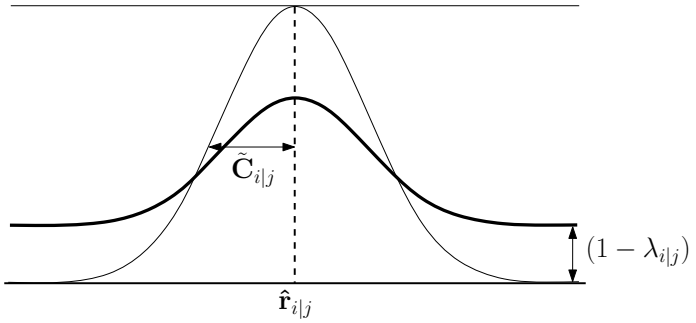


Figure 2.2.3: 1-dimensional example of $\psi_{i,j}(\mathbf{r}_{i|j})$ represented with a rigid uncertain model mixing a Gaussian with an uniform potential function. $\lambda_{i|j}$ is the weight given to the Gaussian distribution, $\hat{\mathbf{r}}_{i|j}$ the mean of the distribution, and $\tilde{\mathbf{C}}_{i|j}$ its covariance matrix.

butions. Combined with the fact these relations are invariant to translation, this allows us to represent them using two models of relative positions $\mathbf{r}_{i|j}$ and $\mathbf{r}_{j|i}$. These models represent the expected affine position of one node expressed in the affine coordinate system of another node as shown in Figure 2.2.2. It reduces the modelling of a spatial relation to two slices in the potential function space of 6 dimensions each and also makes the potential function invariant to the position of the image origin.

Each relative position is expressed using an uncertain rigid model mixing a Gaussian with a uniform potential function. The potential function for the relative position of a node i seen by a node j is then given by

$$\psi_{i,j}(\mathbf{r}_{i|j}) = \lambda_{i|j} e^{-\frac{1}{2}(\mathbf{r}_{i|j}-\hat{\mathbf{r}}_{i|j})\tilde{\mathbf{C}}_{i|j}^{-1}(\mathbf{r}_{i|j}-\hat{\mathbf{r}}_{i|j})} + (1-\lambda_{i|j}), \tag{2.2.1}$$

where $\mathbf{r}_{i|j}$ is computed by expressing $\mathbf{x}_i$ in the space defined by $\mathbf{x}_j$, $\lambda_{i|j}$ is the probability that the spatial relation can be modelled by a Gaussian and $\hat{\mathbf{r}}_{i|j}$ is the mean of the relative configurations already observed. A 1-dimensional example of $\psi_{i,j}(\mathbf{r}_{i|j})$ is given in Figure 2.2.3. The covariance matrix $\tilde{\mathbf{C}}_{i|j}$ represents the variations observed in the relation but also accounts for the uncertainty related to the use an incomplete data set. More details about it are given in Chapter 4.

2.2.3.2 Model of an Articulated Relation

The principle of this uncertain model is the same as for the rigid one: it combines an informative parametric distribution with a non-informative uniform distribution. The major difference between the two models is that an articulated model defines his potential function based on the ability of the two nodes to predict their common joint (i.e. articulation) at the same position. This

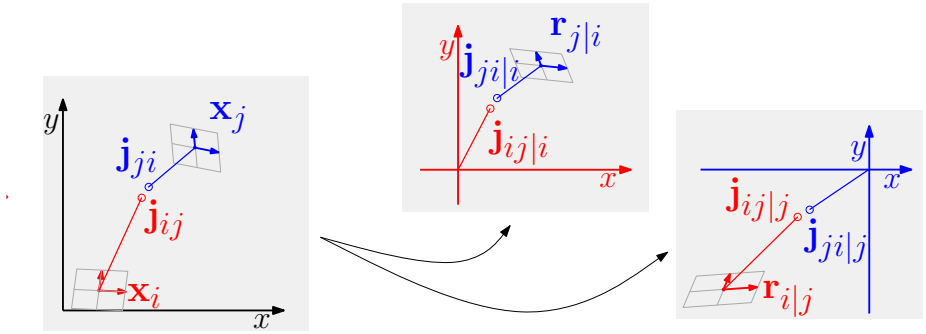


Figure 2.2.4: Expected joint positions projected into the image space ($\mathbf{j}_{ij}$ and $\mathbf{j}_{ji}$) and into each node space.

means that the model first requires to learn the relative 2D positions $\mathbf{j}_{ij|i}$ and $\mathbf{j}_{ji|j}$ of their shared joint (see Figure 2.2.4). The potential function for the relative position of a node i seen by a node j is then computed using the square distance $\mathbf{d}^2_{i|j} = \left\|\mathbf{j}_{ij|j} - \mathbf{j}_{ji|j}\right\|^2$ to get

$$\psi_{i,j}(\mathbf{d}^2_{i|j}) = \lambda_{i|j} e^{-\mathbf{d}^2_{i|j}/\tilde{\sigma}^2_{i|j}} + (1 - \lambda_{i|j}), \tag{2.2.2}$$

where $\lambda_{i|j}$ is the probability that the projection $\mathbf{j}_{ij|j}$ is normally distributed around $\mathbf{j}_{ji|j}$, and $\tilde{\sigma}^2_{i|j}$ represents the variance of $\mathbf{d}^2_{i|j}$ augmented to account for the uncertainty.

On the one hand, more complex spatial relations could be represented using a mixture of Gaussians or a parametric curve in the 6-dimensional space but the computational cost would not be acceptable for a real-time application. On the other hand, the joint position can be learnt more efficiently: only two 2D joint positions to determine and a single dimension model to estimate the variance and model uncertainty. More details about the learning of an uncertain articulated model are given in Chapter 4.

2.3 Experiment

Outside a context of detection or tracking, analysing the reliability of a model is not really relevant. This being said, it is still interesting to test the initialisation of this particular model in order to get an intuition of its repeatability and potential problems. In Figures 2.3.1, 2.3.2, 2.3.3, and 2.3.4, the feature level of the model has been initialised on a series of test images (the block level being a single node at first). All the results are obtained using the same method and parameters:

1. Normalise the image and apply the Canny edge detector using a Gaussian filter with a standard deviation of 2 pixels and the two thresholds $T_{\text{high}} = 0.15$ and $T_{low} = 0.05$.

2. Remove the small edges and extract edgels from the remaining edges using a minimum distance of 5 pixels between the edgels. Merge close junctions (points at the end of edges) if needed.

3. Create connections between the edgels by following the contours. Also create connections between junctions and nearby edgels with a maximum distance of 7.5 pixels.

From these results a few observations can be made:

- The model itself provides a representation easily identified by human eyes. This is particularly interesting if some supervision is needed for labelling learnt models.

- As we can see, the models of the (straight) fingers (Figure 2.3.1), arms (Figure 2.3.2), hands (Figures 2.3.1 and 2.3.3), and bodies (Figure 2.3.4) are fairly similar for each instance of the corresponding object, even when comparing results from grey images and depth maps (obtained from a Kinect).

- Textured surfaces create a lot of edgels (see for example the hand in Figure 2.3.1). This would be the same for interest points but this is more of a concern for edgels since their discriminative power is much lower. Not only will it consumes a lot of the available computational time to deal with these extra edgels before their removal, but it might be detrimental to tracking during the early learning phase when the block level is not very useful yet. Different techniques such as motion or depth segmentation can be used to focus the model on specific edgels. Depth maps are particularly useful since they do not contain texture and provide a simple solution to focus on foreground objects. With the launch of cheap depth camera like the Kinect (which was used to get the depth images in Figures 2.3.1, 2.3.3 and 2.3.4), this solution seems the most interesting even for a home computer.

- When a part of an object is similar to its background (e.g. Figures 2.3.2j and 2.3.3e) or too close to a similar part (e.g. 2.3.3e), edgels are simply not detected. This is actually the same result as for an occlusion. This is a better alternative than with regions of interest or skeleton where we would simply have features covering the combined elements. But it also means that we cannot rely on feature detection from a single frame to build the model. While frequent detection on the unmodeled areas (at least to detect new objects) seems an obvious thing to do, it is extremely important that feature detection should be as fast as possible to still allow for simultaneous learning and tracking in real time. In that sense, the low computational cost of edgels extraction is another advantage in their favour.

- While the model is designed to be robust to affine transformations, it seems clear that our context (2D images, 2D features,...) is more favourable to a

2D representation of the objects. This means that, if objects do not have to stay completely in a single 2D plane for their tracking (once learnt) it is preferable for the model to be initialised and learnt when the object is roughly contained on a plane of the camera. In this regard, the use a a depth map would allow this condition to be verified easily. This may seems like a big drawback but is actually very common when learning models based on 2D data, especially if the object is not rigid. The best example is probably the work of Ramanan et al. [67], where convenient poses are detected in order to learn a proper appearance of the articulated object. This being said, the model is developed in order to work with any type of feature. Simply replacing edgels by 3D features should therefore allow our model to represent objects moving out of the 2D plane.

2.4 Discussion

This chapter focused on the definition of an object model suitable for our case. We saw that two models in particular offered a partial but appealing solution. One was the feature graph model ideal for unsupervised learning and a decent solution for tracking. The other was the part-based articulated model difficult to learn on its own but offering a simpler model and allowing a more robust tracking. Since both of these models were based on a graphical representation, we combined them into a two-level graph model so that each level can compensate the weaknesses of the other.

In terms of the appearance of the model, we decided to use edgels since they offer a low computational cost, a better coverage of the object, and a more comprehensive representation. They are also largely invariant to lighting conditions and changes in object colour, object position, and image type (such as colour, grey, or depth).

Despite these qualities, we also observed that individual edgels are poor detectors on their own. It is only when they are used in large groups that they become more robust. We will see in the next chapters if this weakness makes the edgels inappropriate for simultaneous learning and tracking or if it can be overcome.

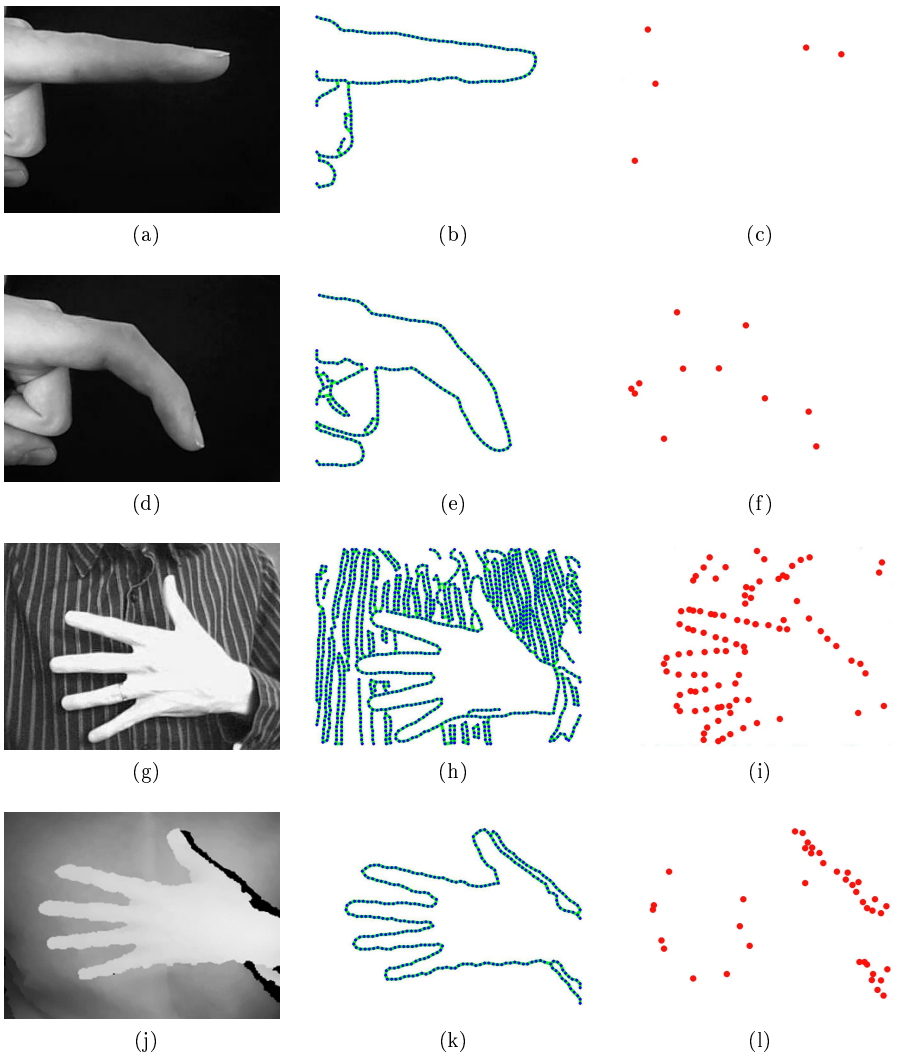


Figure 2.3.1: Examples of initial graphs (second column). As we can see, it provides a representation easily identified by the human eyes and with fairly similar detected features even from grey to depth (last row) image. In comparison, corner points from the Harris detector (third column) usually provide a model far more difficult to recognise and with a lot of points generated from intersection with the background or dependant on the current configuration of the object. Given the strong response of edgels to texture, the depth map seems to be an ideal choice to keep the model as simple (i.e. fast) as possible.

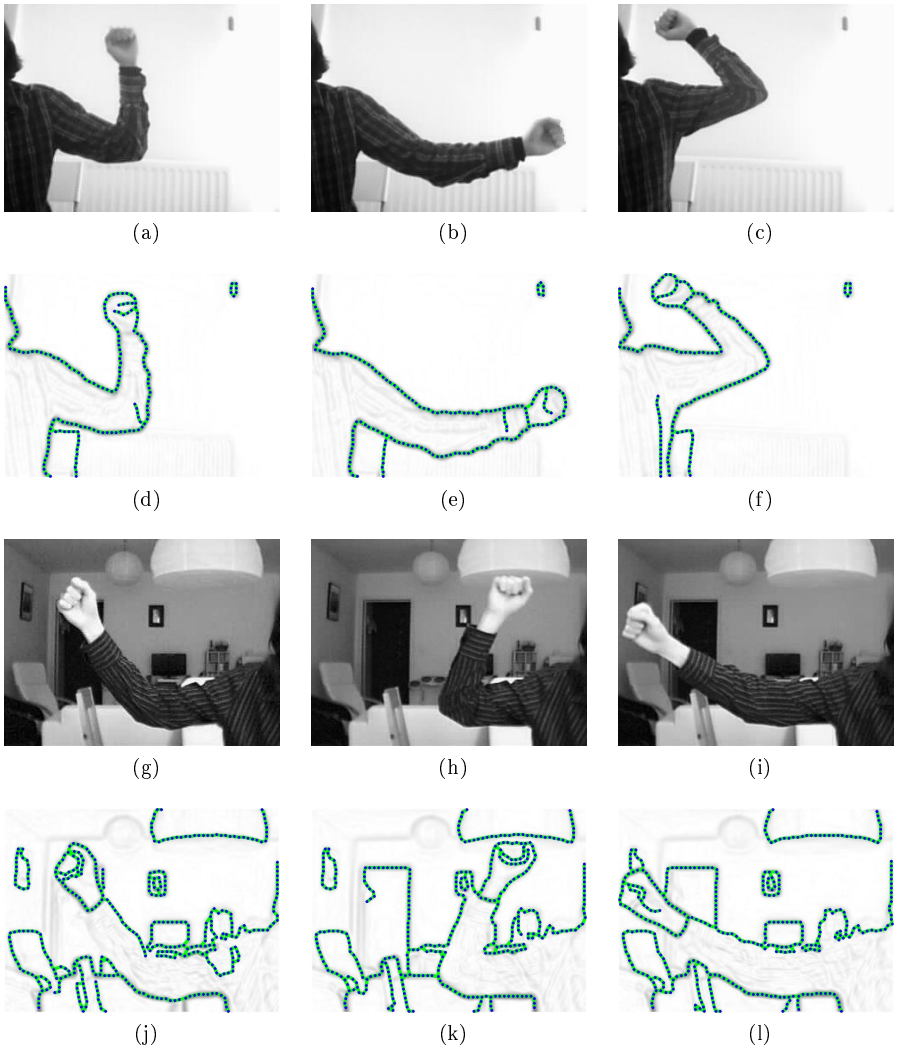


Figure 2.3.2: Examples of graph initialisation for an arm. As we can see, the model is fairly consistent across the images. It is worth noticing though that a background of similar colour acts as an occlusion on the contours (e.g. Figure 2.3.2j). Since this is a problem common to all the features, this will be a weakness in the initialisation of our solution regardless of the features used. Using frequent feature detection to alleviate this problem means that it should be as fast as possible. In that sense, the low computational cost of edgels extraction is an other advantage in their favour.

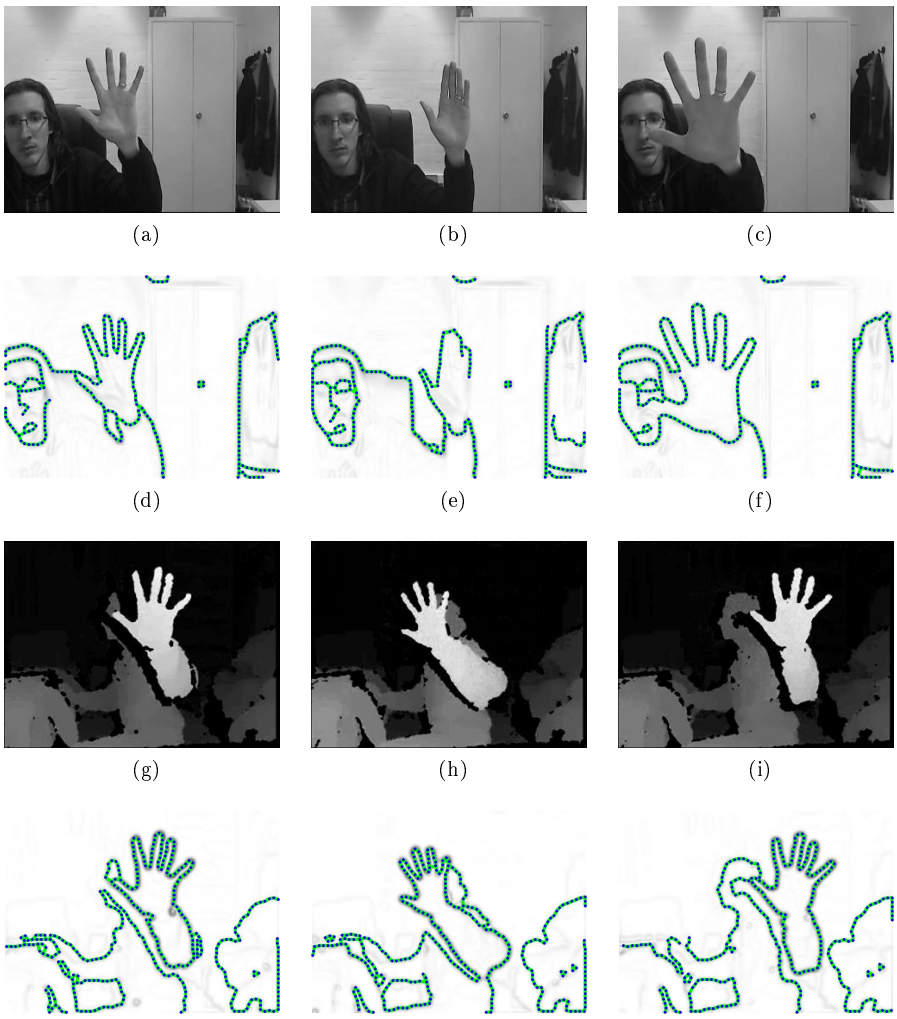


Figure 2.3.3: Examples of graph initialisation for a hand. Observations are similar those of the case of the arm (2.3.2) with the additional issue of edgels missing on fingers when the hand is closed.

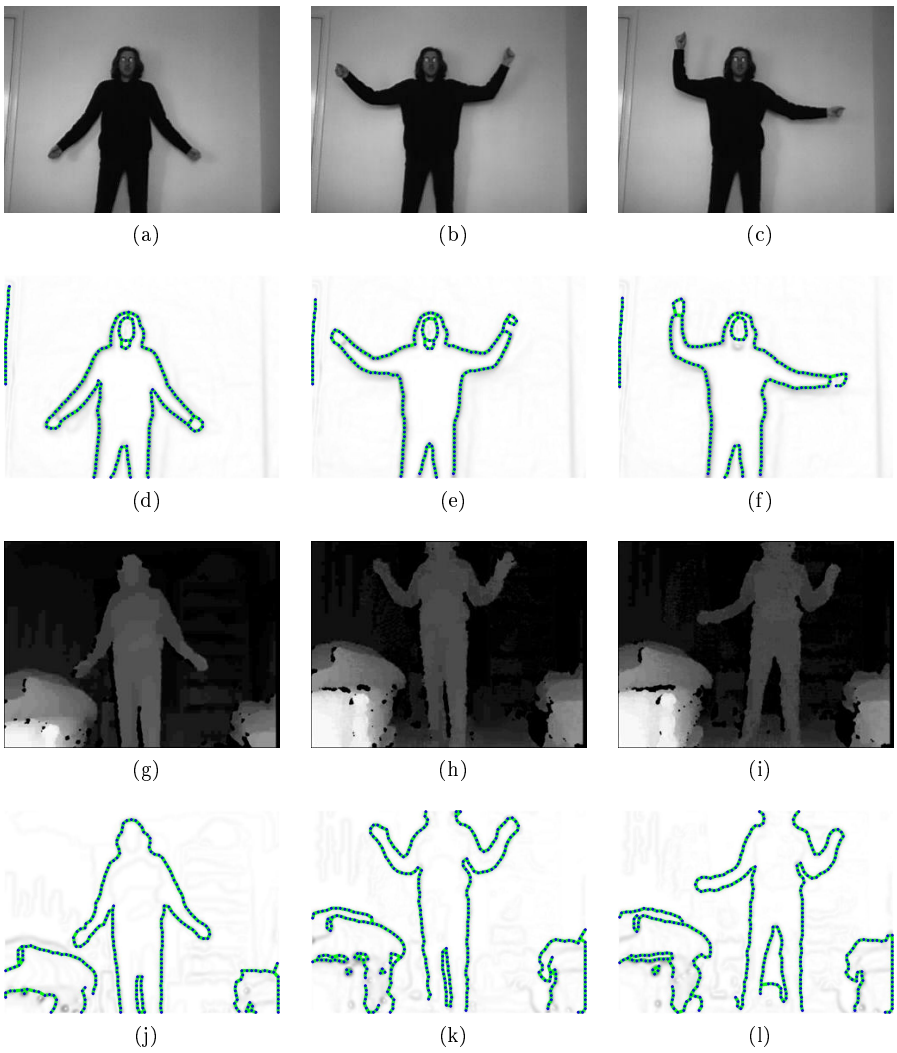


Figure 2.3.4: Examples of graph initialisation for a human body. Observations are similar to those of the case of the arm with the additional issue of edgels missing on the inside of the legs when these are too close to each other.

Chapter 3

Model Tracking

This chapter addresses the problem of tracking an articulated object using its graphical model, independently from the learning process. The tracking occurs in the two levels of the graph: the block level and the feature level. When tracking the blocks, we consider here that the model has already been learnt properly and is thus completely reliable. However, when tracking the features, we use the model as it is defined at time $t = 0$ to verify its ability to provide reliable information for the learning process. While the main goal is to achieve real-time tracking for both levels, each level also has its own requirements: robustness for the block level and enough flexibility for the feature level to follow non-rigid parts until they are segmented.

We will first present in Section 3.1 the most popular methods for tracking either rigid objects or articulated objects. In Section 3.2, we will show how those methods should be modified to fulfil all our requirements (c.f. Chapter 1). We will also address the problem of tracking hundreds of features in real-time by proposing a new tracking method for the feature level that combines some of the advantages of the methods presented in Section 3.1. Finally, in Section 3.3, we demonstrate the benefits of the developed methods as compared to the existing ones.

3.1 Background

When it comes to a single rigid element, we can divide tracking methods into two main groups: the *Single Hypothesis Tracking* (SHT) methods and the *Multiple Hypothesis Tracking* (MHT) methods. The first group of methods tracks an object through the minimisation of an error function (or similarly through the maximisation of a likelihood function) computed locally (i.e. around the last tracked position) while the second group of methods evaluates the likelihood directly in a set of possible positions. SHT methods like Lucas-Kanade [56, 4] or Mean-Shift [17] have been very popular in real-time scenarios due to their low computation time. However, their tracking is limited to a single local maximum

solution per image making them less robust to clutter. At the expense of a higher computational time, MHT methods like Sequential Monte Carlo methods [47] allow to compute the posterior probability distributions of the tracked objects and therefore cover their different possible positions.

For articulated objects, tracking is generally achieved using Belief Propagation (BP) [77, 90]. This allows the known spatial relations between rigid parts to be used to improve the tracking. The idea of this method is to share the probability distribution of each rigid part with its neighbours so that each rigid element can benefit not only from its own information but also from that of its surrounding body parts to compute its posterior probability distribution. Since BP is based on probability distributions, it obviously relies on MHT methods to compute the tracking likelihood of each object part. Very few SHT methods have been developed for articulated objects. While the problem is well addressed by *Active Appearance Models* [18] and *Active Shape Models* [19] for non-rigid "compact" objects (i.e. objects that demonstrate only small local variations of their body parts, like a human face), the explosion of possible poses becomes a major drawback for an articulated object like the human body.

We will see in Section 3.2 that, for the block level, tracking using traditional BP (with a few alterations) provides good results, but, for the feature level, BP requires a computational cost that makes tracking impossible to apply in real-time. We will show that SHT methods can be extended to articulated objects using a propagation scheme similar to BP. We therefore first present some SHT and MHT methods for rigid objects respectively in Section 3.1.1 and Section 3.1.2. We then present several BP methods as MHT solutions for articulated objects in Section 3.1.3 and finally introduce what has been proposed in the literature in term of SHT methods used to track articulated objects in Section 3.1.4.

3.1.1 Single Hypothesis Tracking (SHT) Methods for Rigid Objects

3.1.1.1 Basic Concepts

What we call *Single Hypothesis Tracking* (SHT) methods are methods that provide a single tracking position for each frame. Although this result can be searched in the entire image, the search is generally limited to a local area around the position obtained in the previous frame. Even more, the image information is only extracted inside the bounding box that defines the object area. Each pixel inside this bounding box is asked to suggest a motion given its own image likelihood. The position of the object is then updated in order to match as much as possible the motion required by the pixels. Those two steps are then iterated until the result converges. Since the SHT methods only need to analyse a single local area of the image, they are able to provide a low computational cost solution for real-time tracking.

Notice that what we refer to as a rigid object here might actually not be completely rigid as long as the object remains in the bounding box used to

track it. The notion of rigidity therefore refers more to the fact that the object can be described consistently by some visual values defined inside the bounding box (i.e. image template, histogram,...) through a video sequence. Indeed, a method like Mean-Shift combined with a histogram descriptor is fully capable to track a human body as long as the body remains sufficiently "compact" to be bounded by the bounding box (i.e. the bounding box can be defined so that the body is filling most of the box area while remaining inside its limits) .

3.1.1.2 Mean-Shift

The idea behind Mean-Shift tracking is to find a position $\hat{\mathbf{x}}_t$ that maximises the sum of the likelihood of the pixels $\mathbf{x}_i$ around this position given the image I_t at time t,

$$\hat{\mathbf{x}}_t = \arg\max_{\mathbf{x}_t} f(\mathbf{x}_t, I_t) \tag{3.1.1}$$

$$\text{with} \quad f(\mathbf{x}_t, I_t) = \sum_i k(\left\| \frac{\mathbf{x}_i - \mathbf{x}_t}{h} \right\|^2) \cdot L(\mathbf{x}_i, I_t)\,, \tag{3.1.2}$$

where $L(\mathbf{x}_i, I_t)$ is the likelihood that the pixel $\mathbf{x}_i$ in image I_t belongs to the object, and $k(\cdot)$ is the kernel used to define the local area to consider around a position $\mathbf{x}_t$. The most popular choices for the mean-shift kernel are the Epanechnikov kernel [16] and the Gaussian kernel.

The maximas of $f(\mathbf{x}_t, I_t)$ are located among the zeros of the gradient $\nabla_{\mathbf{x}_t} f = 0$ which leads to

$$\sum_i (\mathbf{x}_i - \hat{\mathbf{x}}_t) \cdot g(\left\| \frac{\mathbf{x}_i - \hat{\mathbf{x}}_t}{h} \right\|^2) \cdot L(\mathbf{x}_i, I_t) = 0\,, \tag{3.1.3}$$

where $g(\mathbf{x}) = -k'(\mathbf{x})$ is minus the derivative of the kernel $k(\mathbf{x})$, assuming that it exists for all $\mathbf{x} \in [0, \infty)$, except for a finite set of points. Note that the derivative of the Epanechnikov profile is the uniform profile [16], while the derivative of the Gaussian kernel remains Gaussian. Solving 3.1.3 for $\hat{\mathbf{x}}_t$ one finds

$$\hat{\mathbf{x}}_t = \frac{\sum_i \mathbf{x}_i \cdot g(\left\| \frac{\mathbf{x}_i - \hat{\mathbf{x}}_t^0}{h} \right\|^2) \cdot L(\mathbf{x}_i, I_t)}{g(\left\| \frac{\mathbf{x}_i - \hat{\mathbf{x}}_t^0}{h} \right\|^2) \cdot L(\mathbf{x}_i, I_t)} \tag{3.1.4}$$

i.e. a simple weighted average of the $\mathbf{x}_i$'s, where $\hat{\mathbf{x}}_t^0$ is the position obtained in the previous iteration. The Mean-Shift procedure consists in iterating Equation 3.1.4 until the solution converges to a local maximum. With a kernel defined on a finite support, the position update can be computed using only local image information, leading to a very fast procedure.

The most popular contribution to Mean-Shift tracking has been made by Comaniciu et al. [17], who use a feature histogram based target representation to compute the image likelihood of the pixels, resulting in a very efficient and

robust tracking method for non-rigid objects with partial occlusions, significant clutter and variations of object scale. Since histograms are not used as visual features in this thesis, we refer the interested reader to [17] for further details.

3.1.1.3 Lucas-Kanade

The goal of the Lucas-Kanade method [56, 4] is to align a template image $T(\mathbf{x})$ to an input image $I_t(\mathbf{x})$ at time t, where $\mathbf{x} = [x, y]^T$ is a vector containing the pixel coordinates. In order to align$T(\mathbf{x})$ and $I_t(\mathbf{x})$, consider an affine warp

$$W(\mathbf{x}_i; \mathbf{p}) = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \tag{3.1.5}$$

where $\mathbf{p} = [p_1, p_2, p_3, p_4, p_5, p_6]$ is the vector of parameters. Alignment between template and image is given by the warp that minimises the sum of squared residuals between the template $T(\mathbf{x})$ and the image $I_t(\mathbf{x})$ warped back onto the coordinate frame of the template:

$$\mathbf{R} = \sum_i w_i \|I(W(\mathbf{x}_i; \mathbf{p})) - T(\mathbf{x}_i)\|^2, \tag{3.1.6}$$

with w_i representing the weight given to the template point $\mathbf{x}_i$. To optimise the expression in Equation 3.1.6, the Lucas-Kanade algorithm assumes that a current estimate of $\mathbf{p}$ is known, and then it iteratively solves for increments $\triangle\mathbf{p}$. The expression

$$\mathbf{R} = \sum_i w_i \|I(W(W(\mathbf{x}_i; \mathbf{p}); \Delta\mathbf{p})) - T(\mathbf{x}_i)\|^2 \tag{3.1.7}$$

is minimised with respect to $\triangle\mathbf{p}$ in each iteration, and then one updates the estimate of the warp with

$$W(\mathbf{x}; \mathbf{p}_{\text{new}}) = W(W(\mathbf{x}; \mathbf{p}_{\text{old}}); \Delta\mathbf{p}). \tag{3.1.8}$$

In the case of affine warp, the parameters update would correspond to

$$\mathbf{p} \leftarrow \begin{pmatrix} p_1 + \Delta p_1 + p_1 \cdot \Delta p_1 + p_2 \cdot \Delta p_3 \\ p_2 + \Delta p_2 + p_1 \cdot \Delta p_2 + p_2 \cdot \Delta p_4 \\ p_3 + \Delta p_3 + p_3 \cdot \Delta p_1 + p_4 \cdot \Delta p_3 \\ p_4 + \Delta p_4 + p_3 \cdot \Delta p_2 + p_4 \cdot \Delta p_4 \\ p_5 + \Delta p_5 + p_5 \cdot \Delta p_1 + p_6 \cdot \Delta p_3 \\ p_6 + \Delta p_6 + p_5 \cdot \Delta p_2 + p_6 \cdot \Delta p_4 \end{pmatrix}. \tag{3.1.9}$$

In order to solve the warp update, the expression in Equation 3.1.7 is linearised by performing a first-order Taylor expansion on $W(W(\mathbf{x}; \mathbf{p}); \Delta\mathbf{p})$ to give:

$$\mathbf{R} = \sum_i w_i \left\| I(W(W(\mathbf{x}_i; \mathbf{p}); \mathbf{0})) + \nabla I \frac{\partial W}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}_i) \right\|^2 \tag{3.1.10}$$

where $\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$ corresponds to the image gradient evaluated at $W(\mathbf{x}_i; \mathbf{p})$ in the coordinate frame of I. Since $W(\mathbf{x}; \mathbf{0})$ is the identity warp, $W(W(\mathbf{x}; \mathbf{p}); \mathbf{0})$ then simplifies to $W(\mathbf{x}; \mathbf{p})$. Following the notational convention that partial derivatives with respect to a column vector are laid out as a row vector [4] and with $W(\mathbf{x}; \mathbf{p}) = [W_x, W_y]^T$, the Jacobian $\frac{\partial W}{\partial \mathbf{p}}$ of the warp of Equation 3.1.17 is given by:

$$\frac{\partial W}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_6} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_6} \end{bmatrix} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}. \tag{3.1.11}$$

Notice that $\frac{\partial W}{\partial \mathbf{p}}$ is computed for $W(W(\mathbf{x}; \mathbf{p}); \mathbf{0})$, which means that the values used in Equation 3.1.11 are the current coordinates of the points in the image.

The solution to the minimisation of Equation 3.1.10 is obtained by setting to zero its partial derivatives with respect to $\Delta\mathbf{p}$ (i.e. $\frac{\partial \mathbf{R}}{\partial \Delta \mathbf{p}} = 0$). We then get

$$\Delta\mathbf{p} = \mathbf{H}^{-1} \sum_i w_i \left[\nabla I \frac{\partial W}{\partial \mathbf{p}}\right]^T \left[T(\mathbf{x}_i) - I(W(\mathbf{x}_i; \mathbf{p}))\right], \tag{3.1.12}$$

where $\mathbf{H}$ is the $n \times n$ Hessian matrix (for affine warp, $n = 6$),

$$\mathbf{H} = \sum_i w_i \left[\nabla I \frac{\partial W}{\partial \mathbf{p}}\right]^T \left[\nabla I \frac{\partial W}{\partial \mathbf{p}}\right]. \tag{3.1.13}$$

The warp update then consists of iteratively applying Equations 3.1.12 and 3.1.8 until the estimates of the parameter vector $\mathbf{p}$ converge.

As we can see from Equation 3.1.12, the solution to the parameter update is found by following the image gradients in order to find a closer match for each pixel of the template. In this idea, $\nabla I \frac{\partial W}{\partial \mathbf{p}}$ is often called the *steepest descent image* and $\sum_i w_i \left[\nabla I \frac{\partial W}{\partial \mathbf{p}}\right]^T \left[T(\mathbf{x}_i) - I(W(\mathbf{x}_i; \mathbf{p}))\right]$ the *steepest parameter update.* Similarly to Mean-Shift, we can see that the Lucas-Kanade method uses only local information to descend along the gradient and to reach a local minimum.

3.1.1.4 Point Matching Based Alignment

Consider, now, a model made of N points and their positions $X = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ in the model reference frame and assume that these points are able to get estimates of their target positions $T = (\mathbf{t}_1, \ldots, \mathbf{t}_N)$ in a given image. These could be estimates based on the image gradients as it was done with Lucas-Kanade, or based on the closest image contour in the case of edgel points. Alignment of the model with those target positions is given by the warp that minimises the sum of squared residuals,

$$\mathbf{R} = \sum_i w_i \|W(\mathbf{x}_i; \mathbf{p}) - \mathbf{t}_i\|^2, \tag{3.1.14}$$

with w_i representing the weight of point i. Assuming that a current estimate of $\mathbf{p}$ is known, Equation 3.1.14 can be iteratively solved for increments $\triangle\mathbf{p}$ of the warp parameters by minimising $\mathbf{R}$ for the incremental warp $W(\mathbf{x}, \Delta\mathbf{p})$:

$$\mathbf{R} = \sum_i w_i \|W(W(\mathbf{x}_i; \mathbf{p}); \Delta\mathbf{p}) - \mathbf{t}_i\|^2 . \tag{3.1.15}$$

Then $\mathbf{p}$ is updated such that

$$W(\mathbf{x}; \mathbf{p}_{\text{new}}) = W(W(\mathbf{x}; \mathbf{p}_{\text{old}}); \Delta\mathbf{p}) . \tag{3.1.16}$$

In order to solve the warp update, the expression in Equation 3.1.15 is linearised in a similar way to that used in the previous section by performing a first-order Taylor expansion on $W(W(\mathbf{x}; \mathbf{p}); \Delta\mathbf{p})$ to give:

$$\mathbf{R} = \sum_i w_i \left\| W(W(\mathbf{x}_i; \mathbf{p}); \mathbf{0}) + \frac{\partial W}{\partial \mathbf{p}} \Delta\mathbf{p}_i - \mathbf{t}_i \right\|^2 . \tag{3.1.17}$$

Again, since $W(\mathbf{x}; \mathbf{0})$ is the identity warp, $W(W(\mathbf{x}; \mathbf{p}); \mathbf{0})$ then simplifies to $W(\mathbf{x}; \mathbf{p})$ and $\frac{\partial W}{\partial \mathbf{p}}$ is computed for $W(W(\mathbf{x}; \mathbf{p}); \mathbf{0})$ (i.e. with the current coordinates of the points in the image).

The solution to the minimisation of Equation 3.1.17 is obtained by setting to zero its partial derivatives with respect to $\Delta\mathbf{p}$:

$$\Delta\mathbf{p} = \mathbf{H}^{-1} \sum_i w_i \left[\frac{\partial W}{\partial \mathbf{p}} \right]^T D_i , \tag{3.1.18}$$

where $D_i = [\mathbf{t}_i - W(\mathbf{x_i}; \mathbf{p})]^T$ is the displacement required of point i, and $\mathbf{H}$ is the $d \times d$ Hessian matrix (with $d = 6$ for affine parameters),

$$\mathbf{H} = \sum_i w_i \left[\frac{\partial W}{\partial \mathbf{p}} \right]^T \left[\frac{\partial W}{\partial \mathbf{p}} \right] . \tag{3.1.19}$$

The warp update then consists of iteratively applying Equations 3.1.18 and 3.1.16 until the estimates of $\mathbf{p}$ converge. In the case of affine transformation, one iteration is sufficient since the system is linear in the parameters. This would not be the case with a parameter such as orientation for example.

This method is mainly a generalisation of Lucas-Kanade but is no longer a gradient descent method since the problem of finding directions for each points of the model is to be solved beforehand. Nonetheless, in situations like edgel tracking, this solution is more convenient since it is easier to find a close contour target than to use gradients to estimate the edgel displacement. Indeed, we saw in Section 2.1.2.2 that the *argument distance transform* (ADT) allows the closest contour to any location to be found in a very effective way.

3.1.2 Multiple Hypothesis Tracking (MHT) Methods for Rigid Objects

3.1.2.1 Basic Concept

In clutter, there are typically several competing observations. The local maximum solution might therefore not always correspond to the correct position (i.e. global maximum) of the object to track. This kind of situation encourages *Multiple Hypothesis Tracking* methods like Sequential Monte Carlo methods [47, 28] (also called particle filters). Instead of simply estimating the object most likely position, particle filters estimate the complete probability distribution $p(\mathbf{x}_t \mid \mathbf{y}_{0:t})$ of the object position, where $\mathbf{x}_t$ is the object position at time t and $\mathbf{y}_{0:t} = [\mathbf{y}_0, \ldots, \mathbf{y}_t]$ are the observations from time 0 to time t. In this way, in cluttered scenes, the multiple possible positions of an object are always accounted for, as the modes of $p(\mathbf{x}_t \mid \mathbf{y}_{0:t})$.

Assuming that the state sequence $[\mathbf{x}_0, \mathbf{x}_1, \ldots]$ is a unobserved Markov process with an initial distribution $p(\mathbf{x}_0)$ and a transition distribution $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ and that the observations $\mathbf{y}_t$ are conditionally independent given $\mathbf{x}_t$, we can decompose $p(\mathbf{x}_t \mid \mathbf{y}_{0:t})$ into

$$p(\mathbf{x}_t \mid \mathbf{y}_{0:t}) = k_t p(\mathbf{y}_t \mid \mathbf{x}_t) p(\mathbf{x}_t \mid \mathbf{y}_{0:t-1}) \tag{3.1.20}$$

$$= k_t p(\mathbf{y}_t \mid \mathbf{x}_t) \int_{\mathbf{x}_{t-1}} p(\mathbf{x}_t \mid \mathbf{x}_{t-1}) p(\mathbf{x}_{t-1} \mid \mathbf{y}_{0:t-1}) d\mathbf{x}_{t-1}\,, \tag{3.1.21}$$

where k_t is a normalisation constant that does not depend on $\mathbf{x}_t$. The effective prior $p(\mathbf{x}_t \mid \mathbf{y}_{0:t-1})$ is actually the posterior $p(\mathbf{x}_{t-1} \mid \mathbf{y}_{0:t-1})$ from the previous time-step, onto which is superimposed the prediction $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ of the dynamical model. Multiplication by the observation density $p(\mathbf{y}_t \mid \mathbf{x}_t)$ then applies the reactive effect expected from observations. Because the observation density is non-Gaussian, the evolving state density $p(\mathbf{x}_t \mid \mathbf{y}_{0:t})$ is also generally non-Gaussian.

3.1.2.2 Tracking with Particle Filter

In cases where $p(\mathbf{y}_t \mid \mathbf{x}_t)$ is sufficiently complex that $p(\mathbf{x}_t \mid \mathbf{y}_{0:t})$ cannot be evaluated simply in closed form, iterative sampling techniques can be used to generate a random variate $\mathbf{x}_t$ from a distribution $\widetilde{p}(\mathbf{x}_t)$ that approximates the posterior $p(\mathbf{x}_t \mid \mathbf{y}_{0:t})$. First, a sample-set $\left\{\mathbf{s}_t^1, \ldots, \mathbf{s}_t^N\right\}$ is generated from the prior density $p(\mathbf{x}_t \mid \mathbf{y}_{0:t-1})$, and then an index $n \in [1, \ldots, N]$ is chosen with probability π_t^n,where

$$\pi_t^n = \frac{p(\mathbf{y}_t \mid \mathbf{s}_t^n)}{\sum_{i=1}^N p(\mathbf{y}_t \mid \mathbf{s}_t^i)}\,. \tag{3.1.22}$$

The samples chosen in this fashion have a distribution that approximates the posterior $p(\mathbf{x}_t \mid \mathbf{y}_{0:t})$ increasingly accurately as N increases. When tracking,

Algorithmus 3.1 *Sampling Importance Resampling* (SIR) algorithm for particle tracking

1. **Importance Sampling**
   From the old sample-set $\left\{\mathbf{s}_{t-1}^n, \pi_{t-1}^n\right\}_{n=1}^N$ at time $t-1$, construct a new sample-set $\{\mathbf{s}_t^n, \pi_t^n\}_{n=1}^N$ for time t. The $i^{\text{th}}$of N new samples is construct as follow:

   (a) **Select** a sample index j with a probability π_{t-1}^j.

   (b) **Predict** by sampling from $p(\mathbf{x}_t \mid \mathbf{x}_{t-1} = \mathbf{s}_{t-1}^j)$ to define $\mathbf{s}_t^i$.

   (c) **Measure** and weight the new position in terms of the observations $\mathbf{y}_t$ to get
   $$\pi_t^i = p(\mathbf{y}_t \mid \mathbf{s}_t^i)\,. \tag{3.1.23}$$

   Once the N samples have been constructed, normalise the weights so that $\sum_n \pi_t^n = 1$.

2. **Resampling**

   (a) Compute an estimate of the effective number of particles as
   $$\hat{N}_{\text{eff}} = \frac{1}{\sum_n \left(\pi_t^n\right)^2}\,. \tag{3.1.24}$$

   (b) If $\hat{N}_{\text{eff}} < N_{\text{thres}}$, then

   - Draw a set of N particles from the current particle set with probabilities proportional to their weights π_t^n and replace the current particle set with this new set.
   - For $i = 1, \ldots, N$, set $\pi_t^i = 1/N$.

samples at time t can be obtained from samples at time $t-1$ using the dynamical model $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$. All in all, the particle filter tracking can be summarised by Algorithm 3.1, where the resampling step is used to avoid the problem of degeneracy of the algorithm, that is, avoiding the situation where all but one of the importance weights are close to zero.

With sufficient samples, particle filters are able to approach the Bayesian optimal estimate of $p(\mathbf{x}_t \mid \mathbf{y}_{0:t})$ and therefore account for multiple tracking hypotheses. Conversely, each sample requires an evaluation of its image likelihood π_t^n. One of the difficulties with the particle filters is then to find the number of particles that provides a good trade-off between computational cost and robustness.

3.1.3 Multiple Hypothesis Tracking (MHT) Methods for Articulated Objects

When tracking articulated object, the methods described above can only be used to track each rigid body part individually. A popular way to use known spatial constraints between the body parts in order to improve their individual tracking is to use Belief Propagation. The idea is to use a graphical representation of the articulated object as presented in Section 2.1.1 to include in the model spatial constrains between the rigid parts. Local evidence gathered by each individual rigid part can then be shared (propagated) among neighbours in order to improve their tracking.

Although it is clear that other MHT methods exist for articulated objects, BP is definitely the most successful [7, 11, 45, 77]. In this section, we will then focus on some of the most popular adaptations of BP.

3.1.3.1 Basic Concept

To simplify matters, let us first consider the graphical model as it was presented in Equation 2.1.5, i.e. without the time aspect:

$$p(X \mid Y) = \frac{1}{Z} \prod_{\{i,j\} \in E} \psi_{i,j}(\mathbf{x}_i, \mathbf{x}_j) \prod_{i \in V} \psi_i(\mathbf{x}_i, \mathbf{y}_i) \,. \tag{3.1.25}$$

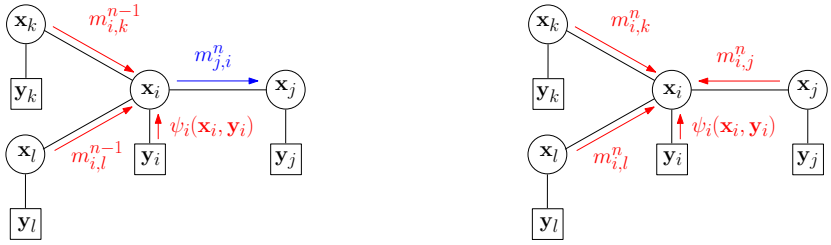
The marginal posterior distribution $p(\mathbf{x}_i|Y)$ of the feature i at time t can, then, be obtained with

$$p(\mathbf{x}_i|Y) = \int ... \int p(X \mid Y) \mathrm{d}x_1 \mathrm{d}x_{i-1} \mathrm{d}x_{i+1} \mathrm{d}x_N \,. \tag{3.1.26}$$

This kind of brute computation of marginal probabilities becomes quickly intractable because of the computation of multiple integrals. It is possible to reorder this integral to exploit the structural properties of the graphical model using the elimination algorithm to reduce the computational effort. However, determining such an ordering in the most general case whilst trying to minimise the computations is an NP-hard problem [77]. The (Loopy) Belief Propagation algorithm [64] is a more efficient algorithm that computes $p(\mathbf{x}_i|Y)$ for $i = 1, ..., N$ through an iterative local message passing process where each node i sends to its neighbour j a message that contains its current belief on the state of node j. At any time during the execution of the algorithm, the current marginal distribution estimated for a node i is the normalised product of its local evidence $p(\mathbf{y}_i|\mathbf{x}_i)$ and of all incoming messages from the neighbouring nodes,

$$p^l(\mathbf{x}_i|Y) \propto \psi_i(\mathbf{x}_i, \mathbf{y}_i) \prod_{j \in N(i)} m^l_{i,j}(\mathbf{x}_i) \,, \tag{3.1.27}$$

where $p^l(\mathbf{x}_i|Y)$ represents the marginal distribution estimated at the l^{th} iteration and $N(i)$ is the set of nodes adjacent to i. $m^l_{i,j}(\mathbf{x}_i)$ is the message sent from node j to node i at the l^{th} iteration.



(a) Computation of $m_{j,i}^n(\mathbf{x}_j)$ (b) Computation of $p^n(\mathbf{x}_i|Y)$

Figure 3.1.1: The message passing process in Belief Propagation.

To prepare a message for node j, the node i first compute an estimation of its own state based on its local evidence and the messages sent by all its neighbour but j. This product is then multiplied with the potential function related to i and j to produce the message from node i to node j ,

$$m_{j,i}^{l}(\mathbf{x}_j) \leftarrow \int_{x_i} \left[\psi_i(\mathbf{x}_i, \mathbf{y}_i) \cdot \psi_{i,j}(\mathbf{x}_i, \mathbf{x}_j) \cdot \prod_{k \in N(x_i) \setminus j} m_{i,k}^{l-1}(\mathbf{x}_i) \right] \mathrm{d}\mathbf{x}_i \,. \tag{3.1.28}$$

Figure 3.1.1 illustrates the method. In the case of tree-structured graphical models, the algorithm is guaranteed to converge towards $p^l(\mathbf{x}_i|Y)$. In graphical models with loops, convergence is not guaranteed but good empirical performance has been shown in the literature [84].

3.1.3.2 Nonparametric Belief Propagation (NBP)

In graphical models with continuous, non-Gaussian variables, there is usually no tractable analytic form for the messages underlying BP. Nonparametric Belief Propagation (NBP) [77, 78] addresses this problem by representing each message nonparametrically as a Gaussian mixture. Given that $G(\mathbf{x}; \mu, \Lambda)$ denotes a normalised Gaussian density with mean μ and covariance Λ, an M components mixture approximation of $m_{j,i}(\mathbf{x}_j)$ takes the form

$$m_{j,i}(\mathbf{x}_j) = \sum_{k=1}^{M} w_j^k G(\mathbf{x}_j; \mu_j^k, \Lambda_j) \,, \tag{3.1.31}$$

where w_j^k is the weight associated with the k^{th} kernel mean μ_j^k and Λ_j is a bandwidth or smoothing parameter. The weights are normalised so that $\sum_{k=1}^{M} w_j^k = 1$.

If performed exactly, each message update would then produce a new product mixture with an exponentially large number of components. NBP approximates this product mixture using an efficient Gibbs sampling procedure [36, 46],

Algorithm 3.2 Gibbs sampling for the product of several mixtures

Given d mixtures of M Gaussians, where $\left\{w_j^k, \mu_j^k, \Lambda_j\right\}_{k=1}^M$ denotes the parameters of the j^{th} mixture, a sample is obtained as follows:

1. For each $j \in [1:d]$, choose a starting label $l_j \in [1:M]$ by sampling $p(l_j = k) \propto w_j^k$.
2. For each $j \in [1:d]$,
   (a) Calculate the mean μ^* and variance $\Lambda^*$of the product $G(\mathbf{x}; \overline{\mu}, \overline{\Lambda}) \propto \prod_{s \neq j} G(\mathbf{x}; \mu_s^{l_s}, \Lambda_s)$ using
   $$\overline{\Lambda}^{-1} = \sum_{s \neq j} \Lambda_s^{-1} \qquad \overline{\Lambda}^{-1}\overline{\mu} = \sum_{s \neq j} \Lambda_s^{-1} \mu_s^{l_s} . \tag{3.1.29}$$
   (b) For each $k \in [1:M]$, calculate the mean $\overline{\mu}^k$ and variance $\overline{\mathbf{\Lambda}}^k$ of $G(\mathbf{x}; \mu^*, \Lambda^*) \cdot G(\mathbf{x}; \mu_j^k, \Lambda_j)$. Using any convenient $\mathbf{x}$, compute the weight
   $$\overline{w}^k = w_j^k \frac{G(\mathbf{x}; \mu^*, \Lambda^*) \cdot G(\mathbf{x}; \mu_j^k, \Lambda_j)}{G(\mathbf{x}; \overline{\mu}^k, \overline{\Lambda}^k)} \tag{3.1.30}$$
   (c) Sample a new label l_j according to $p(l_j = k) \propto \overline{w}^k$.
3. Repeat step 2 κ times.
4. Compute the mean $\overline{\mu}$ and variance $\overline{\Lambda}$ of the product $\prod_{j=1}^d G(\mathbf{x}; \mu_j^{l_j}, \Lambda_j)$. Draw a sample $\hat{x} \sim G(\mathbf{x}; \overline{\mu}, \overline{\Lambda})$.

allowing a natural trade-off between statistical accuracy and computational efficiency. Details about the sampling from the product of several Gaussian mixtures are provided in Algorithm 3.2. To complete the stochastic integration, each sample $\mathbf{x}_j^k$ from the message product is propagated to the neighbouring node i by sampling $\mathbf{x}_i^k$ from $\psi_{i,j}(\mathbf{x}_i, \mathbf{x}_j^k)$. Finally, having produced a set of independent samples from the output message $m_{i,j}(\mathbf{x}_i)$, NBP selects a kernel bandwidth to complete the nonparametric density estimate.

3.1.3.3 Sequential Belief Propagation (SBP)

The Markov network presented in Section 3.1.3.1 is a generative model at one time instant. To track it, Equation 3.1.25 is extended to account for correlation between positions in successive frames. Under the conventional Markov assumption for the time dimension, i.e.

$$p\left(X_t \mid X_{t-1}\right) = \prod_{i \in \mathcal{V}} p\left(\mathbf{x}_{i,t} \mid \mathbf{x}_{i,t-1}\right) , \tag{3.1.36}$$

the posterior probability of the joint state X_t given image measurements $Y_{0:t}$ can be expressed as

$$p(X_t \mid Y_{0:t}) = \frac{1}{Z} \prod_{(i,j) \in \mathcal{E}} \psi_{i,j,t}(\mathbf{x}_{i,t}, \mathbf{x}_{j,t}) \prod_{i \in \mathcal{V}} \psi_{i,t}(\mathbf{x}_{i,t}, \mathbf{y}_{i,t}) \cdot p(\mathbf{x}_{i,t} \mid Y_{i,0:t-1}) , \tag{3.1.37}$$

where

$$p(\mathbf{x}_{i,t} \mid Y_{i,0:t-1}) = \int_{\mathbf{x}_{i,t-1}} p(\mathbf{x}_{i,t} \mid \mathbf{x}_{i,t-1}) \cdot p(\mathbf{x}_{i,t-1} \mid Y_{i,0:t-1}) \cdot \mathrm{d}\mathbf{x}_{i,t-1} . \tag{3.1.38}$$

As shown by Hua and Wu [45] and later by Briers et al. [11], inference can be performed by Sequential Belief Propagation (SBP) through an iterative, local message passing process. The local message passed from node i to node j at time t and iteration l is given by

$$\begin{aligned} m_{j,i,t}^{l}(\mathbf{x}_{j,t}) \quad \leftarrow \quad & \int_{\mathbf{x}_{i,t}} \left[\psi_{i,t}(\mathbf{x}_{i,t}, \mathbf{y}_{i,t}) \cdot \psi_{i,j,t}(\mathbf{x}_{i,t}, \mathbf{x}_{j,t}) \prod_{k \in \mathcal{N}(i,t) \setminus j} m_{i,k,t}^{l-1}(\mathbf{x}_{i,t}) \right. \\ & \left. \times \int_{\mathbf{x}_{i,t-1}} p(\mathbf{x}_{i,t} \mid \mathbf{x}_{i,t-1}) \cdot p(\mathbf{x}_{i,t-1} \mid Y_{0:t-1}) \cdot \mathrm{d}\mathbf{x}_{i,t-1} \right] d\mathbf{x}_{i,t}, \end{aligned} \tag{3.1.39}$$

with $\mathcal{N}(i,t) \setminus j$, the set of neighbours of i at time t except j, and the marginal posterior probability at time t is given by

Algorithm 3.3 Sequential Belief Propagation

1. **Sequential Monte Carlo**

   (a) For each node, resample the particles $\left\{\mathbf{s}_{i,t-1}^{n}\right\}_{n=1}^{N}$ from time $t-1$ according to the weights $\left\{w_{i,t-1}^{n}\right\}_{n=1}^{N}$ to get $\left\{\mathbf{s}_{i,t-1}^{n}, \frac{1}{N}\right\}_{n=1}^{N}$.

   (b) For each node, propagate the particles by sampling from $\left\{p(\mathbf{x}_t \mid \mathbf{x}_{t-1} = \mathbf{s}_{i,t-1}^{n})\right\}_{n=1}^{N}$ to define $\left\{\mathbf{s}_{i,t}^{1,n}\right\}_{n=1}^{N}$.

2. **Initialisation of Belief Propagation**

   (a) For each relation between two nodes i and j, send a first message $m_{i,j,t}^{1}(\mathbf{x}_{j,t}) = \left\{w_{j,i,t}^{1,n}\right\}_{n=1}^{N}$ from j to i computed with

   $$w_{i,j,t}^{1,n} = \sum_{m=1}^{N} \psi_{i,t}(\mathbf{s}_{i,t}^{1,n}, \mathbf{y}_{i,t}) \psi_{i,j,t}(\mathbf{s}_{i,t}^{1,n}, \mathbf{s}_{j,t}^{1,m}) \,. \tag{3.1.32}$$

   (b) For each node i, compute $p^{1}(\mathbf{x}_{i,t} \mid Y_{0:t}) \sim \left\{\mathbf{s}_{i,t}^{1,n}, w_{i,t}^{1,n}\right\}_{n=1}^{N}$ with

   $$w_{i,t}^{1,n} = \psi_{i,t}(\mathbf{s}_{i,t}^{1,n}, \mathbf{y}_{i,t}) \prod_{j \in \mathcal{N}(i,t)} w_{i,j,t}^{1,n} \,. \tag{3.1.33}$$

3. **Iteration l of Belief Propagation**

   (a) For each node i, sample new particles from an importance function $q_{i,t}^{l}(\mathbf{x}_{i,t})$ based on the posterior probability $p^{l-1}(\mathbf{x}_{i,t} \mid Y_{0:t})$.

   (b) For each sample $\mathbf{s}_{i,t}^{l,n}$ and each $j \in \mathcal{N}(i,t)$, set the weights

   $$\begin{aligned} w_{i,j,t}^{1,n} \quad = \quad & \frac{1}{q_{i,t}^{l}(\mathbf{s}_{i,t}^{l,n})} \sum_{m=1}^{N} \Big[\psi_{i,j,t}(\mathbf{s}_{i,t}^{l,n}, \mathbf{s}_{j,t}^{l-1,m}) \psi_{j,t}(\mathbf{s}_{j,t}^{l-1,m}, \mathbf{y}_{j,t}) \\ & \times \left(\frac{1}{N} \sum_{r=1}^{N} p(\mathbf{s}_{j,t}^{l-1,m} \mid \mathbf{s}_{j,t-1}^{r}) \right) \\ & \times \prod_{k \in \mathcal{N}(j,t) \setminus i} w_{j,k,t}^{l-1,m} \Big] \,. \end{aligned} \tag{3.1.34}$$

   (c) For each node i, compute $p^{l}(\mathbf{x}_{i,t} \mid Y_{0:t}) \sim \left\{\mathbf{s}_{i,t}^{l,n}, w_{i,t}^{l,n}\right\}_{n=1}^{N}$ with

   $$w_{i,t}^{l,n} = \frac{\psi_{i,t}(\mathbf{s}_{i,t}^{l,n}, \mathbf{y}_{i,t})}{q_{i,t}^{l}(\mathbf{s}_{i,t}^{l,n})} \left(\frac{1}{N} \sum_{r=1}^{N} p(\mathbf{s}_{i,t}^{l,n} \mid \mathbf{s}_{i,t-1}^{r}) \right) \prod_{j \in \mathcal{N}(i,t)} w_{i,j,t}^{l,n} \tag{3.1.35}$$

   (d) $l \leftarrow l + 1$ for a given number of iterations or until convergence.

$$p(\mathbf{x}_{i,t} \mid Y_{0:t}) \propto \psi_{i,t}(\mathbf{x}_{i,t}, \mathbf{y}_{i,t}) \prod_{j \in \mathcal{N}(i,t)} m^l_{i,j,t}(\mathbf{x}_{i,t}) \times \int_{\mathbf{x}_{i,t-1}} p(\mathbf{x}_{i,t} \mid \mathbf{x}_{i,t-1}) \cdot p(\mathbf{x}_{i,t-1} \mid Y_{0:t-1}) \cdot \mathrm{d}\mathbf{x}_{i,t-1}. \tag{3.1.40}$$

Another contribution of Hua and Wu is the new way to compute the product of messages through particle filters. While NBP [77] already used particles to represent the nodes distribution, the messages in BP are represented by Gaussian mixtures and therefore require complex MCMC (Markov chain Monte Carlo) samplers to sample the new Gaussian mixture kernels of the updated messages. This process is far too slow to be applied in real-time. Hua and Wu proposed to consider the messages as a set of weights for the particles of the node that receives the message. In this way, the product of messages only consists of a product of scalars, which requires much less computational time than for a product of Gaussian mixtures.

The particles of each node are propagated from time $t-1$ to time t and used as a support on which the messages are propagated and the nodes distribution are evaluated. Each message from i to j at iteration l corresponds then to a set of weights $w^{l,n}_{j,i,t}$ for the N particles associated to j, i.e., a message is represented as

$$m^l_{j,i,t}(\mathbf{x}_{j,t}) \sim \left\{ w^{l,n}_{j,i,t} \right\}_{n=1}^{N}. \tag{3.1.41}$$

The marginal posterior probability resulting from propagation is defined as

$$p(\mathbf{x}_{i,t} \mid Y_{0:t}) \sim \left\{ \mathbf{s}^n_{i,t}, w^n_{i,t} \right\}_{n=1}^{N}, \tag{3.1.42}$$

where $s^n_{i,t}$ are the N samples associated with i, and $w^n_{i,t}$ are their posterior weights. The whole process can be repeated as a succession of particle sampling and Belief Propagation until convergence. For each iteration of SBP, particles are resampled in order to provide the best possible support for the messages and the posterior distribution. Since the distribution of the particles represents a probability distribution, the effect of the importance function $q^l_{i,t}(\mathbf{x}_{i,t})$ used for the resampling is cancelled by dividing each weight $w^{1,n}_{i,j,t}$ by $q^l_{i,t}(\mathbf{s}^{l,n}_{i,t})$.

The different steps of the methods are shown in Algorithm 3.3 and illustrated in Figure 3.1.2. Notice that the temporal information is not represented like the other sources of information during the initialisation of SBP. Indeed, while all of the other sources are computed through a product of weights for the particles, the temporal information is represented through the distribution of the particles.

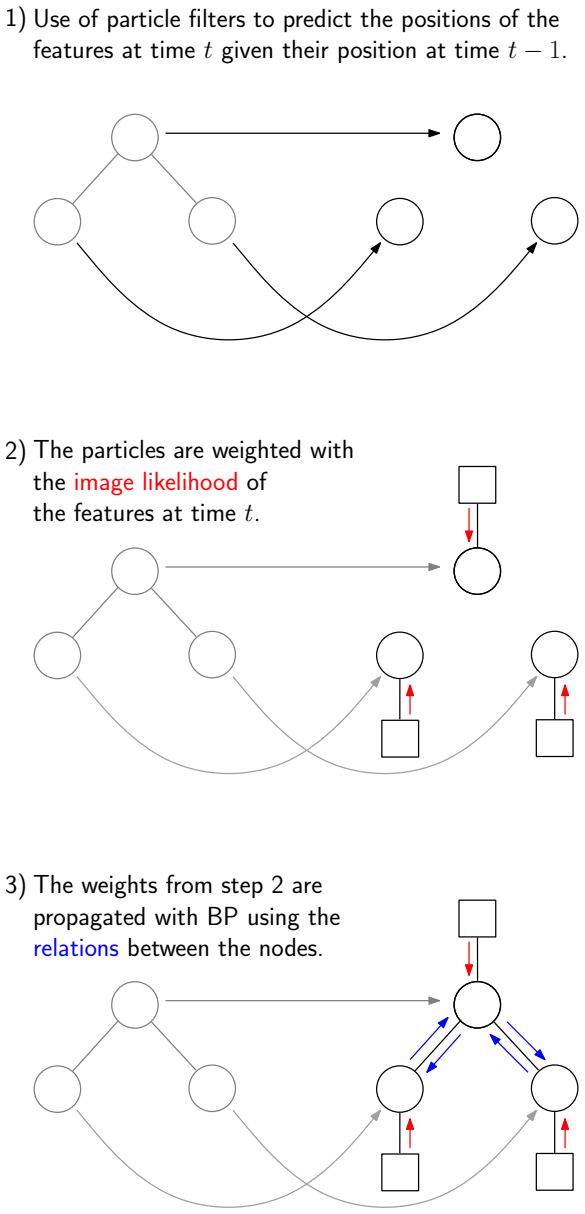


Figure 3.1.2: The 3 steps and 3 sources of information of SBP.

3.1.4 Single Hypothesis Tracking (SHT) Methods for Articulated Objects

The main drawback of the MHT methods discussed in the previous section is their computational cost. On top of the particle sampling process, SBP requires N^2 evaluations of $\psi_{i,j,t}(\mathbf{x}_{i,t}, \mathbf{x}_{j,t})$ for each message exchanged between two nodes and a computation of $\psi_{i,t}(\mathbf{x}_{i,t}, \mathbf{y}_{i,t})$ for every new sample created. Given that each node usually requires at least hundreds of particles to be tracked properly, it is clear that these solutions are limited to small graphs when real-time tracking is required. When computational time is very limited or when the graph contains a large number of nodes, it seems reasonable to sacrifice the robustness of multi-modal solutions for the speed gained by limiting oneself to a single mode. To the best of our knowledge, very few solutions have been developed in this direction. We present here two methods that have been proposed recently (2008), one combining Mean Shift with BP and the other solving a system of articulated planes for the entire articulated object at once.

3.1.4.1 Mean Shift Belief Propagation (MSBP)

Instead of using samples to evaluate the probability density of the complete hidden variable space, Mean Shift Belief Propagation (MSBP) [62] works within a local regular grid of samples centred at the predicted state. This grid of samples becomes a hidden variable space within which BP message passing is performed to compute a weight for each sample. Once the weights are computed, Mean Shift is applied to the samples at each node to reach a new predicted state for each node. A new discrete grid of samples is then generated centred on this predicted state, and the process repeats. Because the Mean-Shift algorithm only needs to examine the values of the belief surface within its local kernel window, it needs a significantly smaller number of samples than particle filtering and therefore a lower computation time compared to NBP or SBP. If Mean Shift were done on each node using only its local image likelihood, then it might have settled on a local mode of no interest, caused by clutter or noise. However, the application of BP before each Mean Shift highly reduces the risk of false local modes in the posterior distribution.

The MSBP procedure is summarised in Algorithm 3.4. One thing that is interesting to notice here is that the particles are not resampled between two iterations of BP. This means that this approach completely removes the costly sampling process from BP but also that the evaluations of $\psi_{i,t}(\mathbf{x}_{i,t}, \mathbf{y}_{i,t})$ and $\psi_{i,j,t}(\mathbf{x}_{i,t}, \mathbf{x}_{j,t})$ are only done once per Mean Shift iteration (instead of per BP iteration as before). This point is really interesting in large graphs where the number of BP iterations is significantly larger than the number of Mean Shift iterations. Moreover, it makes more sense since the BP iterations are used to reach an increasingly larger neighbourhood while the Mean Shift iterations (like the particle resampling) focus on the convergence toward the proper value of of $\mathbf{x}_{i,t}$.

This being said, MSBP offers a reduced computational time only when the

dimensionality of $\mathbf{x}_{i,t}$ is small. Indeed, for an affine space, even with 10 steps per dimension, this method would requires around 10^6 samples. In this situation, it seems very likely that a good tracking result can be obtained with SBP and a far smaller set of particles (say $O(10^3)$), actually making MSBP a slower solution.

3.1.4.2 Linear Motion Estimation for Systems of Articulated Planes

In this method, Datta et al. [22] propose to solve the articulated object tracking as an alignment of the rigid body parts with the constraints that the joint shared by two blocks must be predicted in the same position by the two blocks. The image alignment of a single block i is done using the same equation as for Lucas-Kanade (see Section 3.1.1.3):

$$\Delta\mathbf{p}_i = \mathbf{H}_i^{-1}\mathbf{S}_i\,, \tag{3.1.46}$$

with

$$\mathbf{S}_i = \sum_n w_n \left[\nabla I \frac{\partial W}{\partial \mathbf{p}_i}\right]^T [T(\mathbf{x}_n) - I(W(\mathbf{x}_n;\mathbf{p}_i))] \tag{3.1.47}$$

$$\mathbf{H}_i = \sum_n w_n \left[\nabla I \frac{\partial W}{\partial \mathbf{p}_i}\right]^T \left[\nabla I \frac{\partial W}{\partial \mathbf{p}_i}\right]\,. \tag{3.1.48}$$

For M blocks, their independent linear systems may be combined into a larger system,

$$\begin{bmatrix} \mathbf{H}_1^{-1} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{H}_M^{-1} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{p}_1 \\ \vdots \\ \Delta\mathbf{p}_M \end{bmatrix} = \begin{bmatrix} \mathbf{S}_1 \\ \vdots \\ \mathbf{S}_M \end{bmatrix}\,, \tag{3.1.49}$$

or, in matrix form, $\mathbf{\Gamma P} = \mathbf{\Lambda}$. Each articulation (i.e. joint) $\mathbf{a}_{i,j} = [x_{i,j}, y_{i,j}]$ between two blocks i and j must be aligned in the same position in the new image and therefore provide a constraints that can be expressed as

$$W(\mathbf{a}_{i,j};\mathbf{\Delta p}_i) = \mathbf{a}'_{i,j} = W(\mathbf{a}_{i,j};\Delta\mathbf{p}_j)\,. \tag{3.1.50}$$

For the affine case, this equation can rewritten as

$$\frac{\partial W(\mathbf{a}_{i,j})}{\partial \mathbf{p}}\mathbf{\Delta p}_i - \frac{\partial W(\mathbf{a}_{i,j})}{\partial \mathbf{p}}\mathbf{\Delta p}_j = 0\,, \tag{3.1.51}$$

where

$$\frac{\partial W(\mathbf{a}_{i,j})}{\partial \mathbf{p}} = \begin{bmatrix} x_{i,j} & 0 & y_{i,j} & 0 & 1 & 0 \\ 0 & x_{i,j} & 0 & y_{i,j} & 0 & 1 \end{bmatrix}\,, \tag{3.1.52}$$

Algorithm 3.4 Mean Shift Belief Propagation

1. **Initialisation of Belief Propagation**

   (a) For each node i, define the particle set $\left\{\mathbf{s}_{i,t-1}^{n}\right\}_{n=1}^{N}$ as a grid around the initial mean position $\hat{\mathbf{x}}_{i,t}^{0}$.

   (b) For each node i, compute the weights $\left\{\pi_{i,t}^{k,n}\right\}_{n=1}^{N}$ using the image likelihood $\psi_{i,t}(\mathbf{x}_{i,t}, \mathbf{y}_{i,t})$ and, if it is defined, the prior probability $p(\mathbf{x}_{i,t} \mid Y_{i,0:t-1})$.

   (c) For each relation between two nodes, evaluate the N^2 potential functions $\pi_{i,j,t}^{k,n,m} = \psi_{i,j,t}(\mathbf{s}_{i,t}^{k,n}, \mathbf{s}_{j,t}^{k,m})$ between the particles of the two nodes.

2. **Iteration l of Belief Propagation**

   (a) For each sample $\mathbf{s}_{i,t}^{k,n}$ and each $j \in \mathcal{N}(i,t)$, set the weight

   $$w_{i,j,t}^{k,l,n} = \sum_{m=1}^{N} \left[\pi_{i,j,t}^{k,n,m} \pi_{i,t}^{k,n} \prod_{r \in \mathcal{N}(j,t) \setminus i} w_{j,r,t}^{k,l-1,m} \right] . \quad (3.1.43)$$

   (b) $l \leftarrow l + 1$ for a given number of iteration or until convergence.

3. **Inference result**

   (a) For each node i, compute the marginal distribution $p^{k,l}(\mathbf{x}_{i,t} \mid Y_{0:t}) \sim \left\{\mathbf{s}_{i,t}^{k,n}, w_{i,t}^{k,n}\right\}_{n=1}^{N}$ with

   $$w_{i,t}^{k,n} = \pi_{i,t}^{k,n} \prod_{j \in \mathcal{N}(i,t)} w_{i,j,t}^{k,l,n} . \quad (3.1.44)$$

4. **Mean Shift**

   (a) For each node i, compute the mean shift update as

   $$\hat{\mathbf{x}}_{i,t}^{k} = \frac{\sum_n g(\left\| \frac{\mathbf{s}_{i,t}^{k,n} - \hat{\mathbf{x}}_{i,t}^{k-1}}{h} \right\|^2) \cdot w_{i,t}^{k,n} \cdot \mathbf{s}_{i,t}^{k,n}}{\sum_n g(\left\| \frac{\mathbf{s}_{i,t}^{k,n} - \hat{\mathbf{x}}_{i,t}^{k-1}}{h} \right\|^2) \cdot w_{i,t}^{k,n}} . \quad (3.1.45)$$

5. **Iteration**: $k \leftarrow k + 1$, iterate steps $1 \rightarrow 5$ until convergence.

as in Equation 3.1.11. If we express the set of constraints between the nodes of the graph in the matrix form $\mathbf{\Theta P} = \mathbf{0}$, the constraint between node i and j will appear as

$$\begin{bmatrix} & \vdots & & \vdots & \\ \cdots & \frac{\partial W(\mathbf{a}_{i,j})}{\partial \mathbf{p}} & \cdots & -\frac{\partial W(\mathbf{a}_{i,j})}{\partial \mathbf{p}} & \cdots \\ & \vdots & & \vdots & \end{bmatrix} \begin{bmatrix} \vdots \\ \mathbf{\Delta p}_i \\ \vdots \\ \mathbf{\Delta p}_j \\ \vdots \end{bmatrix} = \mathbf{0}\,. \tag{3.1.53}$$

Using Equations 3.1.49 and 3.1.53, the displacements $P = [\mathbf{\Delta p}_1, \ldots, \mathbf{\Delta p}_i]^{\mathrm{T}}$ can be obtained by solving $\mathbf{\Gamma P} = \mathbf{\Lambda}$ subject to $\mathbf{\Theta P} = \mathbf{0}$.

For tracking in an affine space, $\mathbf{\Gamma}$and $\mathbf{\Theta}$ are respectively a $6M \times 6M$ matrix and a $2K \times 6M$ matrix, where M is the number of nodes and K is the number of constraints. While this method is clearly faster than BP for small graphs (since it does not require particles), it quickly becomes intractable when M increases.

3.2 Developed Solutions

The two-level graphical model we presented in Section 2.2.1 is likely to have hundreds of nodes into the feature level. Tracking this level in real-time is therefore impossible with any of the methods presented in Section 3.1. The message-passing process presented for BP is an excellent way to reduce the computational time compared to solving the full system as in Section 3.1.4.2. Unfortunately, the use of particles in BP makes it quickly intractable for graphs with more than a few nodes. Solving a small system as in Section 3.1.1.4 locally for every node using its neighbourhood would definitely provide a lower computational time but we cannot afford to learn the spatial relations between every possible pair of nodes (at least if we want a real-time solution). A message-passing process would allow us to share information between all the nodes of the graph using only a small set of relations between the closest nodes. Based on those observations, we present in Section 3.2.1 a solution that combines a non-probabilistic message-passing process and the image alignment method from Section 3.1.1.4 and allows tracking of large graphs in real-time. The content of Section 3.2.1 has been presented in our article *Affine Warp Propagation for Fast Simultaneous Modelling and Tracking of Articulated Objects [24].*

We then address the tracking of the block level. Given its smaller number of nodes, tracking all the blocks can be achieved in real-time even with MHT methods. Using a combination of particle filters and SHT methods, we show in Section 3.2.3 how we can benefit from both the robustness of MHT and low computational cost of SHT.

Finally, in Section 3.2.4, we start the discussion on combining the tracking results of the two levels, and how the results from one can benefit the results of the other.

3.2.1 Feature Level Tracking - Affine Warp Propagation

The point matching based alignment method presented in Section 3.1.1.4 allows a rigid element to be aligned using the displacement proposed by its feature points. The computational cost of this method makes it a good candidate for real-time tracking. Indeed, the cost of Equation 3.1.18 is $O(d^2N + d^3)$ where $d = 6$ for affine parameters and N is the number of points used for the alignment (for more details on the total cost of this method, see for example [4]).

In the case of the feature graph presented in Section 2.2.1, each one has only access to a set of learnt relations with its direct neighbours. This means that a message-passing process must be used in order for each node to benefit from the largest possible neighbourhood. Clearly, we saw that BP, as it is, would not provide a real-time solution and we therefore propose a non-probabilistic message-passing process combined with the point matching based alignment method presented in Section 3.1.1.4.

After a short definition of our non-probabilistic graphical model in Section 3.2.1.1, we discuss the content of a message needed to solve the alignment of each feature node in Section 3.2.1.2. We then discuss in Section 3.2.1.3 the message-passing process itself and how spatial relations should be used in this process to guarantee a robust tracking. Finally, we show in Section 3.2.1.4 that other types of information like the image likelihood can be propagated through the message for a small additional cost.

3.2.1.1 Non-Probabilistic Graphical Model Definition

Going back to Section 3.1.1.4, we can see that a model consists in a series of N points, their weights $\varpi = (w_1, \ldots, w_N)$, and their positions $X = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ in the model reference frame. These points are aligned with their target positions $T = (\mathbf{t}_1, \ldots, \mathbf{t}_N)$ in a given image using a set of affine parameters $\mathbf{p} = [p_1, p_2, p_3, p_4, p_5, p_6]$.

No reliable global model being provided (at least in the beginning of the learning phase), each feature must play the role of its own local model. The local model of a feature node i consists in a set of the N other nodes, their weights $\varpi_i = (w_{1|i}, \ldots, w_{N|i})$,and their relative positions $R_i = (\hat{\mathbf{r}}_{1|i}, \ldots, \hat{\mathbf{r}}_{N|i})$ in the reference frame of node i (where the notation $\hat{\mathbf{r}}_{1|i,t}$ is used in order to be coherent with Section 2.2.3.1). Using Equation 3.1.18 and the nodes target positions $T_t = (\mathbf{t}_{1,t}, \ldots, \mathbf{t}_{N,t})$, each node would then be able to update its affine parameters $\mathbf{f}_{i,t} = [f_{1,i,t}, f_{2,i,t}, f_{3,i,t}, f_{4,i,t}, f_{5,i,t}, f_{6,i,t}]$ using its rigid neighbourhood (the level of rigidity being accounted by the weights ϖ_i).

Clearly, learning the N^2 weights and relative positions between every pair of nodes in the graph would be a waste of computational time. The size of the neighbourhood could be tuned to provide a trade-off between computational cost and robustness but it would need to be adjusted for each situation. By using a message-passing process as in BP, it is possible to limit the number of relations to a minimum and still allow each node to benefit from the rest of the graph. The type of message needed to achieve this goal will be discussed in the

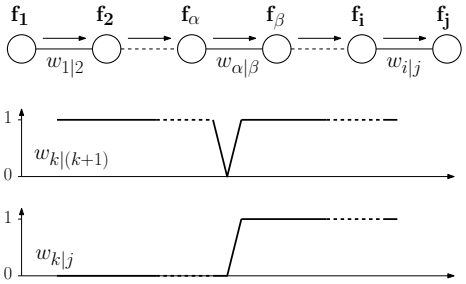


Figure 3.2.1: Simple line graph situation where all spatial relations between connected points are rigid except for the relation between α and β. This means that this graph represents two uncorrelated rigid sets of nodes: one on the left and one on the right of the relation between α and β. The learnt weights between a node k and the nodes to its right are given by $w_{k|(k+1)}$ (centre panel). In the case where messages are propagated from left to right, the bottom panel illustrates the resulting influence $w_{k|j}$ on the rightmost point j of each upstream point k. Here, the null weight $w_{\alpha|\beta}$ eliminates the influence on node j of the nodes that are not in the same rigid set.

next section.

Given what we discussed here, we could already deduce that an edge going from node i to node j will represent two types of information. First, it contains the learnt affine parameters $\hat{\mathbf{r}}_{j|i}$ of node j as they were previously observed in the affine space of node i (i.e. the parameters that align j and its surroundings from the origin to their observed positions in the space of i). Second, it contains the weight $w_{j|i}$ node i should give to the information coming from node j. This weight is directly related to the rigidity of the relation: the lower the correlation between two features, the lower the weight and then the smaller the influence of messages passing through this connection. This means that information coming from points behind a non-rigid connection will have no influence on image alignment (see Figure 3.2.1 for an example). We then define the weight between two unconnected nodes k and j as the product of the weights encountered going from k to j.

3.2.1.2 Message Definition

In this section, we use, for the sake of explanation, the simple line graph shown in Figure 3.2.1. If node j had access to the displacement $D_{k,t}$ of all the points on the graph, its warp update using Equation 3.1.12 would be

$$\Delta\mathbf{f}_{j,t} = \mathbf{H}_{j,t}^{-1} \sum_{k \leq j} w_{k|j} \left[\frac{\partial W_{k|j}}{\partial \mathbf{f}_{j,t}} \right]^{\mathrm{T}} D_{k,t}\,, \tag{3.2.1}$$

with

$$\mathbf{H}_{j,t} = \sum_k w_{k|j} \left[\frac{\partial W_{k|j}}{\partial \mathbf{f}_{j,t}}\right]^{\mathrm{T}} \left[\frac{\partial W_{k|j}}{\partial \mathbf{f}_{j,t}}\right] , \tag{3.2.2}$$

and where we use the notation $\frac{\partial W_{k|j}}{\partial \mathbf{f}_{j,t}}$ to indicate that the Jacobian is computed for $W(W(\hat{\mathbf{r}}_{k|j}; \mathbf{f}_{j,t}); \mathbf{0})$, i.e. for the projection of $\hat{\mathbf{r}}_{k|j}$ in the image coordinates. The displacement $D_{k,t}$ is obtained using the target position $\mathbf{t}_{k,t}$ with $D_{k,t} = \left[\mathbf{t}_{x,k,t} - W_x(\hat{r}_{k|j}; \mathbf{f}_{j,t}) \ , \ \mathbf{t}_{y,k,t} - W_y(\hat{r}_{k|j}; \mathbf{f}_{j,t})\right]^{\mathrm{T}}$, where x and y are used to indicate that the values are taken respectively for the x and y coordinates.

Now, if we define $w_{k|j} = w_{k|i} w_{i|j}$ for the graph of Figure 3.2.1, the sum can be decomposed the following way:

$$\begin{aligned}
\mathbf{S}_{j,t} &= \sum_{k \leq j} w_{k|j} \left[\frac{\partial W_{k|j}}{\partial \mathbf{f}_{j,t}}\right]^{T} D_{k,t} && (3.2.3)\\
&= w_j \left[\frac{\partial W_{j|j}}{\partial \mathbf{f}_{j,t}}\right]^{T} D_{j,t} + w_{i|j} \sum_{k \leq i} w_{k|i} \left[\frac{\partial W_{k|i}}{\partial \mathbf{f}_{i,t}}\right]^{T} D_{k,t} && (3.2.4)\\
&= S_{j,t} + w_{i|j} \mathbf{S}_{i,t} \, , && (3.2.5)
\end{aligned}$$

where $w_j = w_{j|j}$ is the weight node j gives to its own feature point and, more importantly, where we made the assumption that

$$\frac{\partial W_{k|j}}{\partial \mathbf{f}_{j,t}} = \frac{\partial W_{k|i}}{\partial \mathbf{f}_{i,t}} \, . \tag{3.2.6}$$

This assumption means that we consider that point k is projected in the same image coordinates by nodes i and j, i.e. $W(\hat{r}_{k|j}; \mathbf{f}_{j,t}) = W(\hat{r}_{k|i}; \mathbf{f}_{i,t})$. This is actually correct if nodes i, j, and k are linked by perfectly rigid spatial relations. Since the weights are different from zero only for rigid relations, this assumption seems valid. Unfortunately, we will see in Section 3.2.1.3 that, even if the weights are all correct (which is not guaranteed during the learning phase), small numerical errors can trigger a drift from the correct tracking solution. Nevertheless, let us continue with this assumption in order to understand what type of information a message should contain.

Decomposing $\mathbf{H}_j$ in a similar way to Equation 3.2.5, we obtain

$$\begin{aligned}
\mathbf{H}_{j,t} &= \sum_{k \leq j} w_{k|j} \left[\frac{\partial W_{k|j}}{\partial \mathbf{f}_{j,t}}\right]^{\mathrm{T}} \left[\frac{\partial W_{k|j}}{\partial \mathbf{f}_{j,t}}\right] && (3.2.7)\\
&= w_j \left[\frac{\partial W_{j|j}}{\partial \mathbf{f}_{j,t}}\right]^{T} \left[\frac{\partial W_{j|j}}{\partial \mathbf{f}_{j,t}}\right] + w_{i|j} \sum_{k \leq i} w_{k|i} \left[\frac{\partial W_{k|i}}{\partial \mathbf{f}_{i,t}}\right]^{T} \left[\frac{\partial W_{k|i}}{\partial \mathbf{f}_{i,t}}\right] && (3.2.8)\\
&= H_{j,t} + w_{i|j} \mathbf{H}_{i,t} \, . && (3.2.9)
\end{aligned}$$

Equations 3.2.5 and 3.2.9 mean that the warp update for a node j based on all the points of the graph in Figure 3.2.1 can be computed using only the

information from itself and that accumulated by node i. In the case of affine warps, S_j and H_j are given by

$$S_j = w_j \quad [\ x_j D_{x,j} \quad x_j D_{y,j} \quad y_j D_{x,j} \quad y_j D_{y,j} \quad D_{x,j} \quad D_{y,j}\ ]^T \tag{3.2.10}$$

$$H_j = w_j \begin{bmatrix} x_j^2 & 0 & x_j y_j & 0 & x_j & 0 \\ 0 & x_j^2 & 0 & x_j y_j & 0 & x_j \\ x_j y_j & 0 & y_j^2 & 0 & y_j & 0 \\ 0 & x_j y_j & 0 & y_j^2 & 0 & y_j \\ x_j & 0 & y_j & 0 & 1 & 0 \\ 0 & x_j & 0 & y_j & 0 & 1 \end{bmatrix}, \tag{3.2.11}$$

where we used $D_j = [D_{x,j}, D_{y,j}]$, $x_j = \mathbf{f}_{x,j,t}$, $y_j = \mathbf{f}_{y,j,t}$, and where we dropped the time index t for brevity (as we will do for the rest of Section 3.2.1 since the propagation is for one frame anyway). Note that H_j has a lot of zero or identical elements. Without any loss of information, we can then reduce it to the vector

$$\overline{H}_j = w_j \left[\ 1 \quad x_j \quad y_j \quad x_j^2 \quad x_j y_j \quad y_j^2\ \right]^T . \tag{3.2.12}$$

A message containing the two vectors $\mathbf{S}$ and $\overline{\mathbf{H}}$ is then enough to convey all the information needed to align the nodes in the new image. Each time the message comes through a node, it can easily add the information S_i and $\overline{H}_i$ from this node using Equations 3.2.5 and 3.2.9. Notice that the messages are not expressed in the same space as the feature points or the nodes. In this way, displacements can be accumulated through small 12-elements messages in order to compute the affine alignment of the nodes without any loss of information. Even more, an affine warp can be computed with those messages while each node only needs to provide a displacement (given that enough nodes have been visited). With a propagation scheme inspired from BP [90, 77], the computation of the warp update for each node i can be summarised in 3 steps :

1. Initialise the information for each node i of the graph using Equations 3.2.10 and 3.2.12, and send a first message to each neighbouring node $k \in \mathcal{N}(i)$,

$$\mathbf{S}^0_{k,i} = S_i \tag{3.2.13}$$

$$\overline{\mathbf{H}}^0_{k,i} = \overline{H}_i\,. \tag{3.2.14}$$

2. Propagate the information between the nodes for l iterations (for a message sent from node i to node j):

$$\mathbf{S}^l_{j,i} = S_i + \sum_{k \in \mathcal{N}(i) \setminus j} w_{k|i} \mathbf{S}^{l-1}_{i,k} \tag{3.2.15}$$

$$\overline{\mathbf{H}}^l_{j,i} = \overline{H}_i + \sum_{k \in \mathcal{N}(i) \setminus j} w_{k|i} \overline{\mathbf{H}}^{l-1}_{i,k}\,. \tag{3.2.16}$$

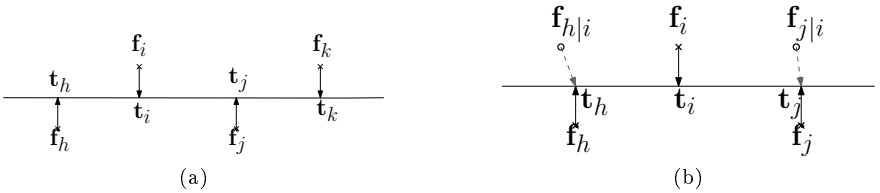


Figure 3.2.2: Consider the nodes h, i, j, and k and their respective targets $\mathbf{t}_h$, $\mathbf{t}_i$, $\mathbf{t}_j$, and $\mathbf{t}_k$ in Figure 3.2.2a. These nodes correspond to edgels that should be in straight line but are not due to some tracking inaccuracy. If nodes i and j are aligned in a new image using the displacement of their direct neighbours, they will drift even further from each other while they should align along the contour. Conversely, if nodes i and j know where their neighbours should be and compute their displacement from there, they will be able to correct the current drift properly (see Figure 3.2.2b, where $\mathbf{f}_{h|i} = W(\hat{r}_{h|i}; \mathbf{f}_i)$ and $\mathbf{f}_{j|i} = W(\hat{r}_{j|i}; \mathbf{f}_i)$ represent the position of nodes h and j expected by node i).

3. Compute the update of the warp parameters for each node j,

$$\mathbf{S}_j = S_j + \sum_{k \in \mathcal{N}(j)} w_{k|j} \mathbf{S}^l_{j,k} \tag{3.2.17}$$

$$\overline{\mathbf{H}}_j = \overline{H}_j + \sum_{k \in \mathcal{N}(j)} w_{k|j} \overline{\mathbf{H}}^l_{j,k} \tag{3.2.18}$$

$$\mathbf{H}_j \leftarrow \overline{\mathbf{H}}_j \tag{3.2.19}$$

$$\Delta \mathbf{f}_j = \mathbf{H}_j^{-1} \mathbf{S}_j \,, \tag{3.2.20}$$

with $\mathcal{N}(j)$ representing the set of nodes that are neighbours of node j. This solution has a message-passing computation cost of $O(12)$ and thus provides a very fast propagation method, which allows each node to align itself in a new image using as much information provided by other feature points as possible.

3.2.1.3 Message Correction

In the previous section, we made the assumption that $W(\hat{r}_{k|j}; \mathbf{f}_{j,t}) = W(\hat{r}_{k|i}; \mathbf{f}_{i,t})$ in order to obtain Equations 3.2.5 and 3.2.9. This assumption means that a node k should be expected in the same position by all the nodes belonging to the same rigid block. Even if all the nodes are indeed rigid with each other, this assumption might not be correct simply because of some numerical inaccuracy in the tracking results. Consider, for example, the case of tracking a set of nodes as shown in Figure 3.2.2. Edgels have been extracted along a line segment and tracked for a few frames. Due to some inaccuracy in the tracking, the edgels are not in a straight line any more. If node i aligns itself using the displacement information of its two direct neighbours, it will be pushed up

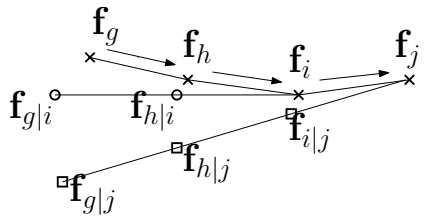


Figure 3.2.3: Position correction. $\mathbf{f}_k$ represents the current configuration of node k in the graph (where k can be any node), $\mathbf{f}_{k|i}$ its position as expected by i, and $\mathbf{f}_{k|j}$ its position as expected by j. In this example, nodes should be aligned along a straight line whereas they are more in a curve configuration. If node i has already corrected the position of the nodes on its left into a straight line, all that is left to do for node j is to apply a single warp to all the $\mathbf{f}_{k|i}$'s to align them with the $\mathbf{f}_{k|j}$'s. The warp needed to do this is the one that aligns $\mathbf{f}_i$ with $\mathbf{f}_{i|j}$.

while it should actually go down (it will also be dramatically scaled down on the vertical direction in case of affine nodes). Same for node j that will be forced to go down making it drift even further from the correct solution. The reason for this problem is quite simple: each node acts as the model described in Section 3.1.1.4 but does not actually evaluate the displacement of its points itself. This latter is indeed provided by each individual node without any consideration of whether it belongs to the model of another node or not. When the assumption $W(\hat{r}_{k|j}; \mathbf{f}_{j,t}) = W(\hat{r}_{k|i}; \mathbf{f}_{i,t})$ is verified, the points used by a model are exactly where they are supposed to be and the displacement information is therefore correct. When the assumption is not verified (e.g. for numerical reason), the node will simply try to match this new configuration of edgels to the current image instead of trying to go back to its initial configuration. For the example of Figure 3.2.2a, this means matching the v-shape formed by h, i, and j to a line segment.

By learning the correct relative positions and using them as the origin of the displacements (as shown in Figure 3.2.2b), the proper relative positions of the nodes can be maintained, and the drift problem eliminated. In term of message to a node i, this means that every occurrence of a position $\mathbf{f}_k$ must be shifted to the expected position $\mathbf{f}_{k|i}$. Following the same idea, every displacement $D_k = \mathbf{t}_k - \mathbf{f}_k$ must be replaced with the expected displacement $D_{k|i} = \mathbf{t}_k - \mathbf{f}_{k|i}$. Notice that the target $\mathbf{t}_k$ is not modified and is thus only an approximation of the true target $\mathbf{f}_{k|i}$ should have provide. Indeed, the correct target $\mathbf{t}_{k|i}$ cannot be computed since the information about $\mathbf{f}_{k|i}$ is merged into $\mathbf{S}_i$ and $\overline{\mathbf{H}}_i$. Since the drift is corrected at each frame, it is kept very small and $\mathbf{t}_k$ is therefore a good estimation of $\mathbf{t}_{k|i}$. By applying this modification, we can see in Figure 3.2.2b that node i will receive consistent information moving it closer to the line instead of pushing it away.

The correction of the positions and displacements of all the points used

in the alignment of a node j is a little tricky. Indeed, the only information available to node j are the expected positions $\mathbf{f}_{k|i}$ of its direct neighbours and the messages $m^l_{j,i} = \{\mathbf{S}^l_{j,i}, \overline{\mathbf{H}}^l_{j,i}\}$ they send. This means that a message coming from a neighbour i must already be corrected for i (since no spatial relation has been learnt with further points) and then adapted for j. Figure 3.2.3 shows an example of this situation where, again, we consider the case where all the points should be in a straight line while they are obviously not. So, assume that all the positions $\mathbf{f}_k$ and displacements D_k of the points k included in the message through $\mathbf{S}^l_{j,i}$ and $\overline{\mathbf{H}}^l_{j,i}$ have already been corrected into $\mathbf{f}_{k|i}$ and $D_{k|i}$ respectively. In order to obtain the positions $\mathbf{f}_{k|j}$ expected by j, the only thing left to do is to adapt the positions $\{\mathbf{f}_{1|i}, \ldots, \mathbf{f}_{i|i}\}$ proposed by node i to the positions $\{\mathbf{f}_{1|j}, \ldots, \mathbf{f}_{i|j}\}$ expected by node j. The correction is the same for all the positions included in the message. Given that $\mathbf{f}_{i|i}$ and $\mathbf{f}_{i|j}$ are known by node j they can be used to compute the transformation needed to go from $\mathbf{f}_{k|i}$ to $\mathbf{f}_{k|j}$. This transformation is simply given by

$$\mathbf{f}_{k|j} = \Delta W_{i,j}(\mathbf{f}_{k|i}) = W(W(\mathbf{f}_{k|i}; \mathbf{f}_i^{-1}); \mathbf{f}_{i|j}) \tag{3.2.21}$$

for any $\mathbf{f}_{k|i}$. Equation 3.2.21 means that a position $\mathbf{f}_{k|i}$ is warped back into the reference frame of node i and then warped into the image using the warp parameters $\mathbf{f}_{i|j}$.

Since we do not have a direct access to positions $\mathbf{f}_{k|i}$, we will have to apply this transformation directly to the message, i.e. to $\mathbf{S}^l_{j,i}$ and $\overline{\mathbf{H}}^l_{j,i}$. Using the notation

$$\begin{bmatrix} x_{k|j} \\ y_{k|j} \\ 1 \end{bmatrix} = W(W(\mathbf{f}_{k|i}; \mathbf{f}_i^{-1}); \mathbf{f}_{i|j}) = \begin{bmatrix} a & c & v \\ b & d & w \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k|i} \\ y_{k|i} \\ 1 \end{bmatrix} \tag{3.2.22}$$

for the correction of the points in the message, we will now apply this correction directly to the Hessian part of the message corrected by node i and sent to j, i.e.

$$\overline{\mathbf{H}}^l_{j,i|i} = \sum_k w_{k|i} \begin{bmatrix} 1 & x_{k|i} & y_{k|i} & x^2_{k|i} & x_{k|i}y_{k|i} & y^2_{k|i} \end{bmatrix}^T . \tag{3.2.23}$$

The corrected Hessian part $\overline{\mathbf{H}}^l_{j,i|j}$ is obtained using Equation 3.2.22 on each term of $\overline{\mathbf{H}}^l_{j,i|i}$, which gives

$$\overline{\mathbf{H}}^l_{j,i|j} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ v & a & c & 0 & 0 & 0 \\ w & b & d & 0 & 0 & 0 \\ v^2 & 2av & 2cv & a^2 & 2ac & c^2 \\ vw & aw+bv & cw+dv & ab & ad+bc & cd \\ w^2 & 2bw & 2dw & b^2 & 2bd & d^2 \end{bmatrix} \overline{\mathbf{H}}^l_{j,i|i} . \tag{3.2.24}$$

To give an example, let us compute the second row ($\overline{\mathbf{H}}^l_{j,i|j}(2)$) of the matrix above. The value of $\overline{\mathbf{H}}^l_{j,i|j}(2)$ using Equations 3.2.22 and 3.2.23 is given by

$$\begin{aligned}
\overline{\mathbf{H}}^l_{j,i|j}(2) &= \sum_k w_{k|i} \cdot x_{k|j} && (3.2.25)\\
&\quad \sum_k w_{k|i} \left[a \cdot x_{k|i} + c \cdot y_{k|i} + v\right] && (3.2.26)\\
&\quad a \cdot \overline{\mathbf{H}}^l_{j,i|i}(2) + c \cdot \overline{\mathbf{H}}^l_{j,i|i}(3) + v \cdot \overline{\mathbf{H}}^l_{j,i|i}(1)\,. && (3.2.27)
\end{aligned}$$

The correction of $\mathbf{S}_{ij}$ is somewhat more difficult because it also depends on $D_{k|i} = \mathbf{t}_k - \mathbf{f}_{k|i}$. The target position $\mathbf{t}_k$ is not modified by the correction, so we replace $D_{k|i} = [D_{x,k|i}, D_{y,k|i}]$ by $[t_{x,k} - x_{k|i}, t_{y,k} - y_{k|i}]$ in

$$\begin{aligned}
\mathbf{S}^l_{i,j|i} = \sum_k w_{k|i} \quad [\ & x_{k|i}D_{x,k|i} \quad x_{k|i}D_{y,k|i} \quad y_{k|i}D_{x,k|i} \quad \cdots \\
& y_{k|i}D_{y,k|i} \quad D_{x,k|i} \quad D_{y,k|i}\ ]^T && (3.2.28)
\end{aligned}$$

and apply the same correction as for $\overline{\mathbf{H}}^l_{j,i|i}$, yielding

$$\begin{aligned}
\mathbf{S}^l_{i,j|j} = &\begin{bmatrix} a & 0 & c & 0 & v & 0 \\ 0 & a & 0 & c & 0 & v \\ b & 0 & d & 0 & w & 0 \\ 0 & b & 0 & d & 0 & w \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{S}^l_{i,j|i} + \begin{bmatrix} 0 & v & 0 & a & c & 0 \\ 0 & 0 & v & 0 & a & c \\ 0 & w & 0 & b & d & 0 \\ 0 & 0 & w & 0 & b & d \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \overline{\mathbf{H}}^l_{j,i|i} \\
&- \begin{bmatrix} \overline{\mathbf{H}}^l_{j,i|j}(4) \\ \overline{\mathbf{H}}^l_{j,i|j}(5) \\ \overline{\mathbf{H}}^l_{j,i|j}(5) \\ \overline{\mathbf{H}}^l_{j,i|j}(6) \\ \overline{\mathbf{H}}^l_{j,i|j}(2) \\ \overline{\mathbf{H}}^l_{j,i|j}(3) \end{bmatrix}\,. && (3.2.29)
\end{aligned}$$

Using Equations 3.2.24 and 3.2.29 to correct the messages allows us to solve the drift problem by maintaining the nodes at their learnt relative positions, making the tracking more robust to occlusions and clutter. The complete algorithm for the propagation with correction is given in Algorithm 3.5. The correction matrix $C_{j|i} = \begin{bmatrix} a & c & v \\ b & d & w \\ 0 & 0 & 1 \end{bmatrix}$ from Equation 3.2.22 is obtained using the expected position $\mathbf{f}_{i|j} = W(\hat{r}_{i|j}; \mathbf{f}_j)$, which depends on the learnt relative position $\hat{r}_{i|j}$ of node i expressed in the reference frame of node j. The learning of these relative positions is explained in Chapter 4. For the moment we will assume that they have been learnt properly already.

Algorithm 3.5 Affine Warp Propagation

1. **Initialisation of Affine Warp Propagation**

   (a) For each node i, compute its local information with

$$\begin{aligned} S_j &= w_j[\ x_j D_{x,j} \quad x_j D_{y,j} \quad y_j D_{x,j} \quad \cdots \\ &\qquad y_j D_{y,j} \quad D_{x,j} \quad D_{y,j}\ ]^T \qquad (3.2.30) \\ \overline{H}_j &= w_j \left[\ 1 \quad x_j \quad y_j \quad x_j^2 \quad x_j y_j \quad y_j^2\ \right]^T . \qquad (3.2.31) \end{aligned}$$

   (b) For each relation between two nodes i and j, evaluate the correction matrix $C_{j|i} = \begin{bmatrix} a & c & v \\ b & d & w \\ 0 & 0 & 1 \end{bmatrix}$ of Equation 3.2.22.

   (c) For each node i, send a first message to each node $k \in \mathcal{N}(i)$:

$$\begin{aligned} \mathbf{S}^0_{k,i|i} &= S_i\,; \qquad (3.2.32) \\ \overline{\mathbf{H}}^0_{k,i|i} &= \overline{H}_i\,. \qquad (3.2.33) \end{aligned}$$

2. **Iteration l of Affine Warp Propagation**

   (a) For each node i, correct the message $m^{l-1}_{i,k|k} = \{\mathbf{S}^{l-1}_{i,k|k}, \overline{\mathbf{H}}^{l-1}_{i,k|k}\}$ coming from each node $k \in \mathcal{N}(i)$ using the precomputed correction matrix $C_{j|i}$ and Equations 3.2.24 and 3.2.29 to get $m^{l-1}_{i,k|i} = \{\mathbf{S}^{l-1}_{i,k|i}, \overline{\mathbf{H}}^{l-1}_{i,k|i}\}$.

   (b) For each relation between two nodes i and j, compute the message $m^l_{j,i|i} = \{\mathbf{S}^l_{j,i|i}, \overline{\mathbf{H}}^l_{j,i|i}\}$ from node i to node j with

$$\begin{aligned} \mathbf{S}^l_{j,i|i} &= S_i + \sum_{k \in \mathcal{N}(i)\setminus j} w_{k|i} \mathbf{S}^{l-1}_{i,k|i} \qquad (3.2.34) \\ \overline{\mathbf{H}}^l_{j,i|i} &= \overline{H}_i + \sum_{k \in \mathcal{N}(i)\setminus j} w_{k|i} \overline{\mathbf{H}}^{l-1}_{i,k|i}\,. \qquad (3.2.35) \end{aligned}$$

3. **Warp Update**

   (a) For each node i, correct the message $m^l_{i,k|k}$ coming from each neighbouring node $k \in \mathcal{N}(i)$ using the precomputed correction matrix $C_{j|i}$ and Equations 3.2.24 and 3.2.29 to get $m^l_{i,k|i}$.

   (b) Compute the update of the warp parameters for each node i,

$$\begin{aligned} \mathbf{S}_i &= S_i + \sum_{k \in \mathcal{N}(i)} w_{k|i} \mathbf{S}^l_{i,k|i} \qquad (3.2.36) \\ \overline{\mathbf{H}}_i &= \overline{H}_i + \sum_{k \in \mathcal{N}(i)} w_{k|i} \overline{\mathbf{H}}^l_{i,k|i} \qquad (3.2.37) \\ \mathbf{H}_i &\leftarrow \overline{\mathbf{H}}_i \qquad (3.2.38) \\ \Delta \mathbf{f}_i &= \mathbf{H}_i^{-1} \mathbf{S}_i\,. \qquad (3.2.39) \end{aligned}$$

3.2.1.4 Likelihood Propagation

Affine Warp Propagation as presented in Algorithm 3.5 is complete and does not need any further improvement in order to work properly. Nevertheless, it is interesting to notice that its message structure actually allows it to convey other type of information than the one needed for the warp update. Indeed, as we saw in Equation 3.2.20, the update of the node parameters $\Delta\mathbf{p}_j$ is calculated using the propagated information contained in $\mathbf{S}_j$ and $\overline{\mathbf{H}}_j$. This means that this information is also available to compute anything else that might be useful for tracking or learning. One such useful item is the image likelihood of the feature points based not only on themselves but also on their surroundings, which is far more robust. Consider the following likelihood L_j of a node j based on the sum of squared distances from Equation 3.1.14,

$$L_j = \mathrm{e}^{-\frac{d_j}{2\sigma_L^2}} \tag{3.2.40}$$

$$d_j = \frac{\sum_k w_k \|D_k\|^2}{\sum_k w_k} = \frac{\sum_k w_k (D_{x,k}^2 + D_{y,k}^2)}{\sum_k w_k} \,. \tag{3.2.41}$$

The only information needed to compute this likelihood is $\sum_k w_k D_{x,k}^2$, $\sum_k w_k D_{y,k}^2$, and $\sum_k w_k$. The latter term is already propagated within $\overline{\mathbf{H}}$. The propagation of the other two can be done in a fashion similar to $\mathbf{S}$ and $\overline{\mathbf{H}}$:

1. Initialise the information for each node i using

$$E_i = w_i \begin{bmatrix} D_{x,i}^2 & D_{y,i}^2 \end{bmatrix}^T \tag{3.2.42}$$

and send a first message to each neighbouring node $k \in \mathcal{N}(i)$:

$$\mathbf{E}_{k,i}^0 = E_i \,. \tag{3.2.43}$$

2. Propagate the information between the nodes for l iterations (for a message sent from node i to node j):

$$\mathbf{E}_{j,i|i}^l = E_i + \sum_{k \in \mathcal{N}(i) \setminus j} w_{k|i} \mathbf{E}_{k,i|i}^{l-1} \,. \tag{3.2.44}$$

3. Apply the message correction to each message using

$$\begin{aligned} \mathbf{E}_{j,i|j}^l = \mathbf{E}_{j,i|i}^l &- \begin{bmatrix} 2(a-1) & 0 & 2c & 0 & 2v & 0 \\ 0 & 2b & 0 & 2(d-1) & 0 & 2w \end{bmatrix} \mathbf{S}_{j,i|i}^l \\ &- \begin{bmatrix} 0 & 2v & 0 & 2a-1 & 2c & 0 \\ 0 & 0 & 2w & 0 & 2b & 2d-1 \end{bmatrix} \overline{\mathbf{H}}_{j,i|i}^l \\ &+ \begin{bmatrix} \overline{\mathbf{H}}_{j,i|i}^l(4) \\ \overline{\mathbf{H}}_{j,i|i}^l(6) \end{bmatrix} \,. \end{aligned} \tag{3.2.45}$$

4. Compute the likelihood for each node j using Equations 3.2.40 and 3.2.41.

This likelihood is useful to detect occlusion or loss of tracking and also plays a role in the learning of the relations between features. Indeed, when features are lost or unobserved, nothing can (nor should) be learnt from the (meaningless) relations observed. So when a new observation is produced, the update of the spatial relation model between two points i and j is weighted by the likelihood product of these points, i.e. L_i and L_j as given in Equation 3.2.40.

3.2.2 Single Block Tracking - Aligned Particle Filter

Given that there are far fewer blocks than features, more expensive methods such as MHT methods become affordable in real-time. This being said, the number of hypotheses should still be kept as low as possible in order to guarantee that BP (which is $O(N^2)$ in the number of hypothesis) does not become too expensive to be applied in combination with the other tracking and learning methods in real-time. As we have seen in Section 3.1, the two extreme tracking solutions are the *Point Matching Based Alignment* (Section 3.1.1.4) and the *Particle Filter* (Section 3.1.2.2). The first one is very fast but is to be discarded due to its tendency to be stuck at local maxima. Unfortunately, the second one is also to be discarded because it usually requires at least hundreds of particles to give a descent tracking in affine space. BP then becomes impossible to apply in real-time even for small graphs.

As an alternative, we combine the two methods in order to integrate their respective advantages. This idea is not completely new and has already been applied with particle filters and Mean-Shift [57, 71]. The principle is quite straightforward and can be applied by iterating the three following steps:

1. Resample the particles based on their weights and propagate them.
2. Improve the position of the particles using *Point Matching Based Alignment*.
3. Compute the likelihood of these results to get the new weights.

Steps one and three correspond to a classic particle filter. Step two guarantees that the particles are on a local maximum. This allows us to use a far smaller number of particles than for the classic method. We will see in Section 3.3 that 10 particles are usually enough to track the block properly.

3.2.3 Block Level Tracking - Belief Propagation

If we only need to consider the best particle for each block, *Affine Warp Propagation* could be used for this level as well. The only thing that would need to be changed from the solution presented in Section 3.2.1 is the propagation of a message between two blocks. In order to benefit from the articulation, the message is used to compute the motion of the joint. This motion is then used to create a new message which will be corrected the same way than in Section 3.2.1.3.

While this approach is extremely fast, it seriously limits the benefit of the particles. It indeed requires that the best particle be chosen before the propagation. In comparison, BP simply modifies the weight of the particles, therefore leaving all the information they provide available for the next frame. Since the block level is the real goal of the learning process and will therefore constitute the final representation of an object, its tracking must benefit from the most robust method available in real-time. This then makes Belief Propagation a better candidate than *Affine Warp Propagation* as long as it can be applied in real-time.

Different research studies [7, 39, 62] have already demonstrate near real-time results for the tracking of a human body with their adapted version of BP. Thanks to the *Aligned Particle Filter* presented in Section 3.2.2, the small number of particles required now allows BP to be used in real-time.

Another element that highly influences the computational cost of BP methods is the number of times the particles are resampled. In this sense, the MSBP presented in Section 3.1.4.1 offers the best results by keeping resampling out of the propagation process. This idea can easily be applied to our version of BP to give the algorithm described in Algorithm 3.6.

3.2.4 Combination of the Two Levels

In order to find a proper way of combining the tracking results from the two levels, let us first make a few observations:

- Once the model has been learnt properly, it is clear that tracking can be done using only the block level.
- The sole purpose of the feature level is to provide a tracking independent of the learnt model and therefore a possibility to detect flaws in the current model.
- *Single Hypothesis Tracking* methods tend to be stuck in local maxima and then need to be initialised not too far from the correct solution.

From those observations, it seems clear that a top-down tracking approach offers a good way to combine the tracking of the two levels. In this way, the block level tracking produces a global solution that is likely to provide a good initial position for the features and therefore increases their chances to converge toward their proper local solution. The distance between global and local solution is what will be used to detect potential inconsistencies in the current model. Anticipating on the next chapters, the bottom-up approach will then be used for updating the model if needed. These conclusions are probably as far as we can go in this chapter. We have indeed limited this chapter to the tracking of a model that is already correct, making the interaction between feature and block levels irrelevant. The combination of the two levels will then be discussed in detail in Chapter 5 where tracking and learning are combined.

Algorithm 3.6 Belief Propagation for Block Level Tracking

1. **Initialisation of Belief Propagation**

   (a) For each node i, apply the *Aligned Particle Filter* to get the new particles $\left\{\mathbf{s}_{i,t}^{1,n}\right\}_{n=1}^{N}$ and their weights $\left\{\pi_{i,t}^{k,n}\right\}_{n=1}^{N}$.

   (b) For each relation between two nodes, evaluate the N^2 potential functions $\pi_{i,j,t}^{k,n,m} = \psi_{i,j,t}(\mathbf{s}_{i,t}^{k,n}, \mathbf{s}_{j,t}^{k,m})$ between the particles of the two nodes.

2. **Iteration l of Belief Propagation**

   (a) For each sample $\mathbf{s}_{i,t}^{k,n}$ and each $j \in \mathcal{N}(i,t)$, compute the weight

   $$w_{i,j,t}^{k,l,n} = \sum_{m=1}^{N} \left[\pi_{i,j,t}^{k,n,m} \pi_{i,t}^{k,n} \prod_{r \in \mathcal{N}(j,t) \setminus i} w_{j,r,t}^{k,l-1,m} \right] . \quad (3.2.46)$$

   (b) $l \leftarrow l + 1$ for a given number of iteration or until convergence.

3. **Inference result**

   (a) For each node i, compute the marginal distribution $p^{k,l}(\mathbf{b}_{i,t} \mid Y_{0:t}) \sim \left\{\mathbf{s}_{i,t}^{k,n}, w_{i,t}^{k,n}\right\}_{n=1}^{N}$ with

   $$w_{i,t}^{k,n} = \pi_{i,t}^{k,n} \prod_{j \in \mathcal{N}(i,t)} w_{i,j,t}^{k,l,n} . \quad (3.2.47)$$

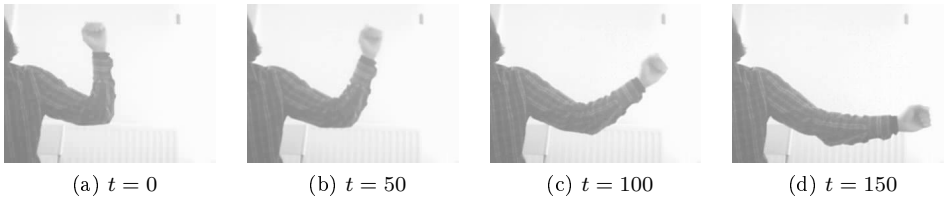
3.3 Experiment

This section is divided into 3 parts: edgel level tracking (*Affine Warp Propagation*), single block tracking, and finally block level tracking (affine warp and BP). Except for the tests on clutter and occlusion, the video in Figure 3.3.1 will always be used to allow a better comparison between the different results. When produced, this video was designed to provide a situation favourable for on-line learning. One thing this video cannot avoid though is the fact that edgels are not very discriminative. A given contour will act as a clutter for any edgel with a similar orientation. Edgels can therefore easily slide along a contour or even jump to a nearby parallel contour.

3.3.1 Edgel Level Tracking

While the goal of the block level is to provide a robust tracking once the model has been learnt, the edgel level is the one that feeds the learning process. Any clear deviation from rigidity that is not yet accounted for in the block level will trigger modifications to it. This means that the edgel level must provide a better ability to adapt to non-rigidity than the blocks. The tracking of the features will dramatically improve with the learning of their spatial relations (between them but also with the rigid blocks). Consequently, we cannot yet provide a definitive evaluation of the quality of their tracking.

What we can already do though is to analyse the impact that the learnt relations between the features have on their tracking. Figures 3.3.2 and 3.3.3 shows the tracking results of the edgel level using only warp propagation and relations with common fixed weights. In the first column of these figures, we track both the feature level with a weight $w = 1$ and the equivalent rigid block used as a reference of the tracking quality. All the experiments were done using a maximum of 150 iterations for warp propagation (in practice, less, since the tracking converge before that). The first thing we can notice is that the tracking results are very similar to the tracking of an equivalent rigid block (shown in red; the edgels, in blue, are hidden behind the block leaving only the green relations visible) if the relation weights are set to 1. This should not be a surprise since *Affine Warp Propagation* does not use any approximation compared to *Point Matching Based Alignment* from Section 3.1.1.4. For the same initial position, the only source of difference is then the numerical errors that may accumulate in the message passing. In the case of the tracking of non-rigid objects, this error can even accumulate from frame to frame since there might be more than one local maximum. This being said, the two results are never far from each other, as we can see in Figure 3.3.6a. The second thing to notice is the increase in flexibility that comes with the reduction of the relation weights. We can therefore see that, by tuning a common weight factor, we can adapt the impact of the current model on the tracking results. To provide the flexibility required by the learning, we can, for example, start with the current learnt weights (this idea will be tested in a later chapter) and then slowly reduce them by a common factor to let the edgels drift from the current model and adapt to a



(a) $t = 0$ (b) $t = 50$ (c) $t = 100$ (d) $t = 150$

(e) $t = 200$ (f) $t = 250$ (g) $t = 300$ (h) $t = 350$

(i) $t = 400$ (j) $t = 450$ (k) $t = 500$ (l) $t = 550$

(m) $t = 600$ (n) $t = 650$ (o) $t = 700$ (p) $t = 750$

(q) $t = 800$ (r) $t = 850$ (s) $t = 900$ (t) $t = 950$

(u) $t = 1000$ (v) $t = 1050$ (w) $t = 1100$ (x) $t = 1150$

Figure 3.3.1: Main test video.

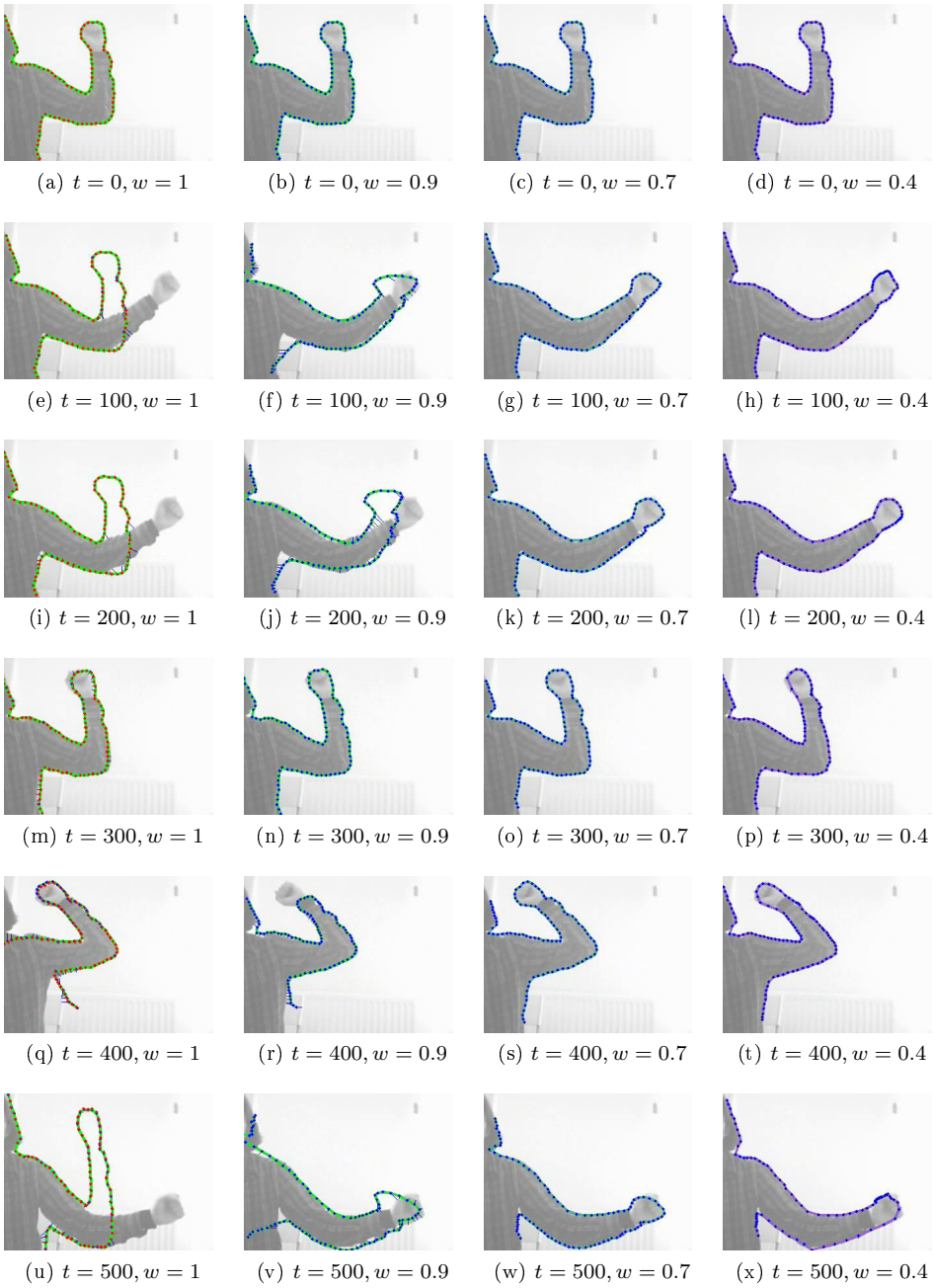
new (unknown) configuration. The last thing to notice is that, obviously, the tracking becomes less and less robust on the long term once we reduce the weight factor. This should not be too much of a concern since the edgel will always initialise their position on each frame by combining their own information with the one provided by the blocks, which are supposed to provide a more robust tracking.

The other parameter that could influence the tracking results (and the computational time) is the number of iterations in the *Affine Warp Propagation*. Its effect is very similar to the weight since it basically controls the size of the neighbourhood used to compute the warp of each edgel. If we consider the equivalent kernel used to filter the information coming from the neighbours, we will simply have a different shape by controlling the weight factor or the number of iterations. It is therefore not surprising that the results obtained in Figures 3.3.4 and 3.3.5 give the same kind of results than for the tuning of the weight factor. More importantly, Figure 3.3.6b shows us the computational time (in milliseconds) with respect to the number of propagation iterations for the sequences in Figures 3.3.4 and 3.3.5 (in red). This time can be decomposed in two terms $T_1(N_e)$ and $T_2(N_r, N_{it})$. $T_1(N_e)$ is proportional to the number N_e of edgels and corresponds to the time needed to extract their own motion from the image and to update them after the propagation. $T_2(N_r, N_{it})$ is proportional to the number N_r of relations and the number N_{it} of propagation iterations, and corresponds to the exchange of messages during *Affine Warp Propagation*. As we can see, even with 50 iterations (which already allows a very large neighbourhood to be covered), the computational time for 94 edgels and 186 relations is only around 4ms on an Intel Core 2 T7200 processor clocked at 2GHz (from 2006).

3.3.2 Single Block Tracking

As discussed in Section 3.2.2, our *Aligned Particle Filter* method is a trade-off between *Point Matching Based Alignment* and *Particle Filters*. The former being a SHT method, it is expected to fail in the presence of clutter. Even when the image seems "clean", edgels can act as clutter for other edgels with similar direction, causing the tracking to fail as well (see for example Figure 3.3.7). Moreover, the tracking of a rigid part of the edgel level using *Affine Warp Propagation* provides the same tracking results as when *Point Matching Based Alignment* is applied to the corresponding block. In order to be of any use, the tracking of the blocks should therefore be more robust than the tracking of the feature level.

Compared to a classical particle filter, the *Aligned Particle Filter* (APF) allows us to use far fewer particles. As we can see in Figures 3.3.8 and 3.3.9 the APF already gives better tracking results with 5 particles than the classical particle filter with 200 particles (which is a minimum for the tracking not to fail). In term of computational time, the tracking for this sequence takes around 0.09ms per particle for the APF against 0.02ms for the classical particle filter. This difference is already largely compensated by the difference in the number of



(a) $t = 0, w = 1$ (b) $t = 0, w = 0.9$ (c) $t = 0, w = 0.7$ (d) $t = 0, w = 0.4$

(e) $t = 100, w = 1$ (f) $t = 100, w = 0.9$ (g) $t = 100, w = 0.7$ (h) $t = 100, w = 0.4$

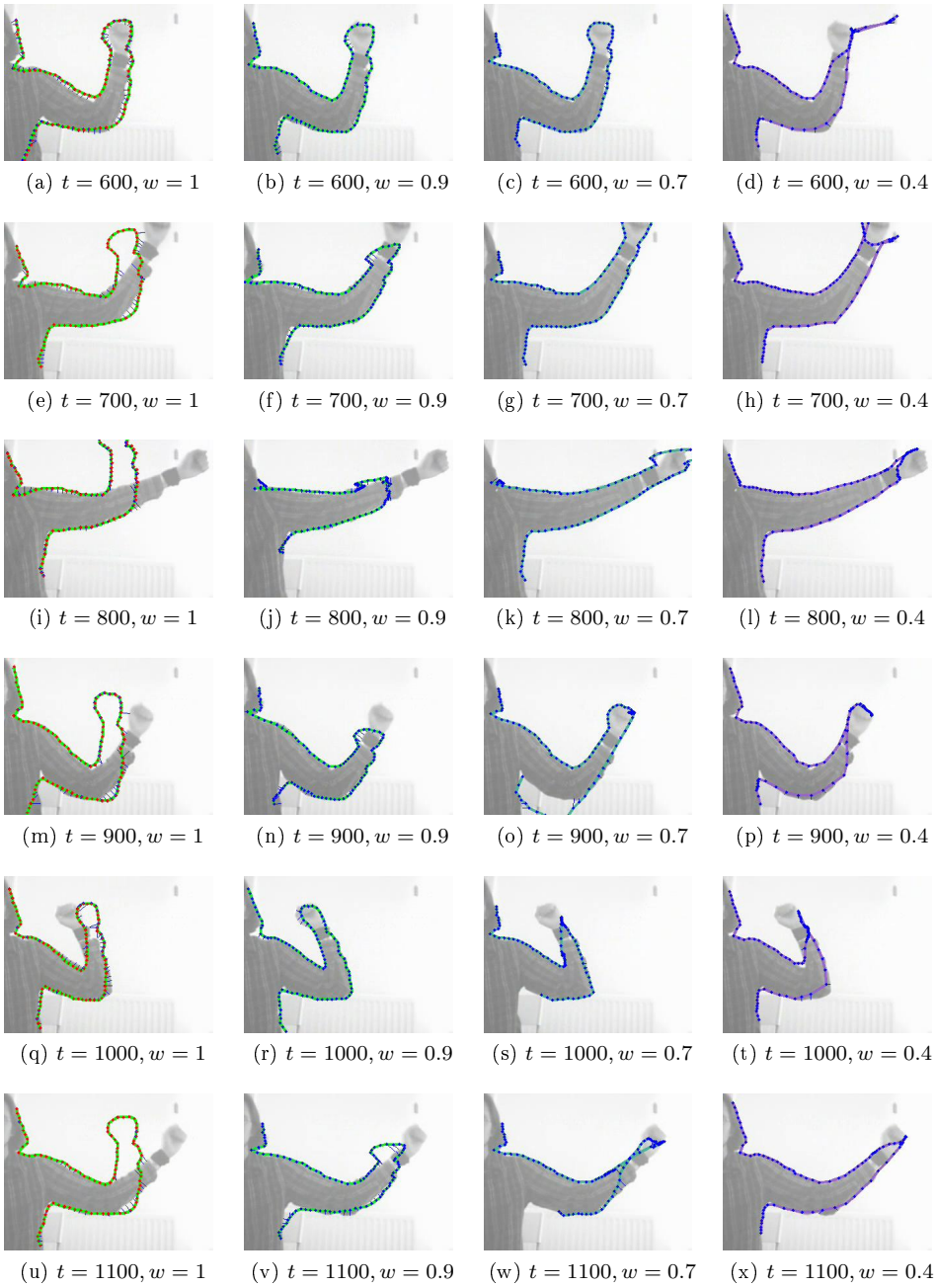
(i) $t = 200, w = 1$ (j) $t = 200, w = 0.9$ (k) $t = 200, w = 0.7$ (l) $t = 200, w = 0.4$

(m) $t = 300, w = 1$ (n) $t = 300, w = 0.9$ (o) $t = 300, w = 0.7$ (p) $t = 300, w = 0.4$

(q) $t = 400, w = 1$ (r) $t = 400, w = 0.9$ (s) $t = 400, w = 0.7$ (t) $t = 400, w = 0.4$

(u) $t = 500, w = 1$ (v) $t = 500, w = 0.9$ (w) $t = 500, w = 0.7$ (x) $t = 500, w = 0.4$

Figure 3.3.2: Edgel level tracking with *Affine Warp Propagation* for different relation weights - frames 0 to 500. The relations weights for each column are, from left to right: 1, 0.9, 0.7, and 0.4. The edgels are represented as blue dots and relations as lines with a colour related to the relation weight (from green for 1 to red for 0). In the first column, we also show, using red dots, the tracking results of a block tracked using *Point Matching Based Alignment* from Section 3.1.1.4. .



(a) $t = 600, w = 1$ (b) $t = 600, w = 0.9$ (c) $t = 600, w = 0.7$ (d) $t = 600, w = 0.4$

(e) $t = 700, w = 1$ (f) $t = 700, w = 0.9$ (g) $t = 700, w = 0.7$ (h) $t = 700, w = 0.4$

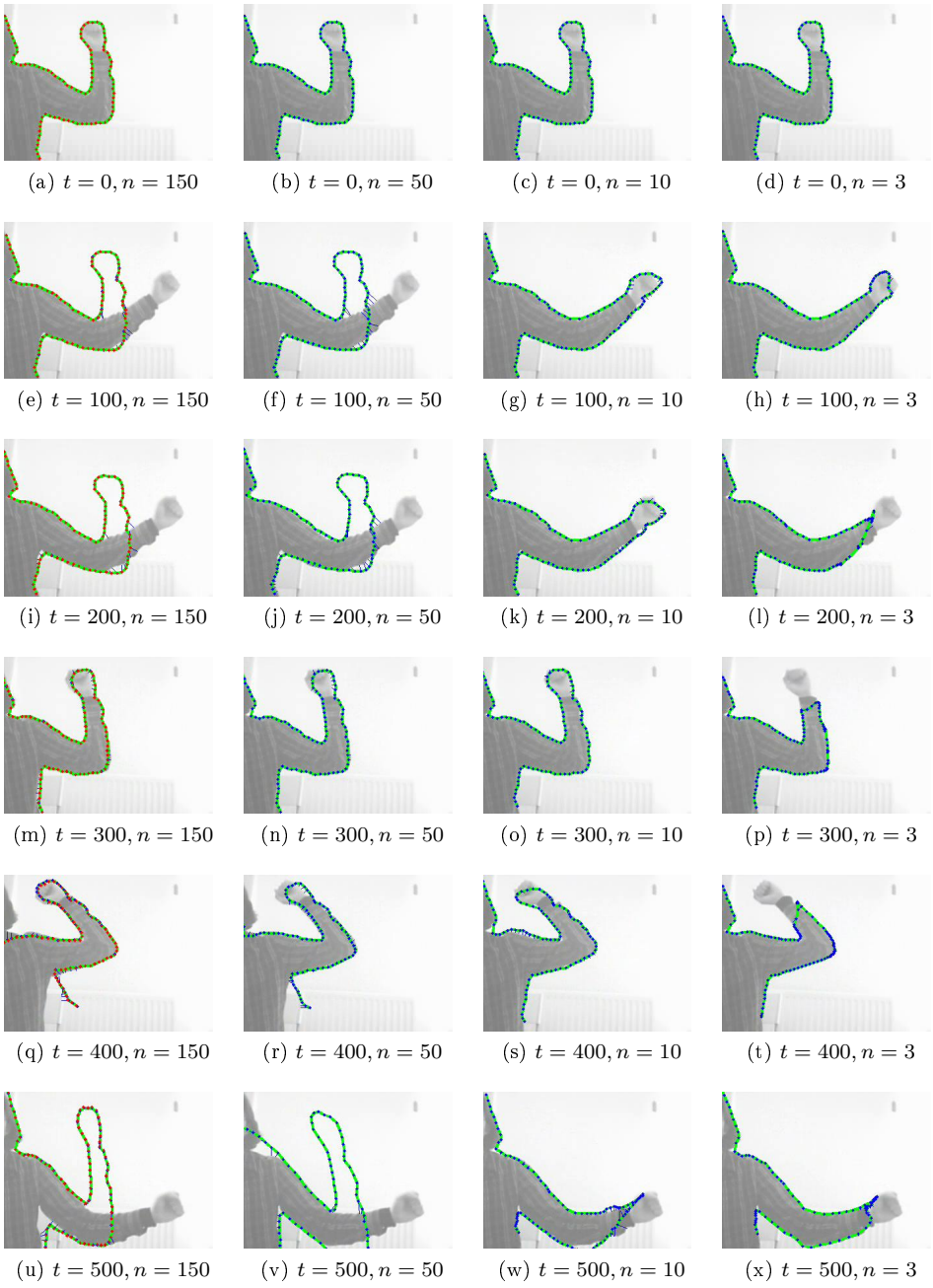
(i) $t = 800, w = 1$ (j) $t = 800, w = 0.9$ (k) $t = 800, w = 0.7$ (l) $t = 800, w = 0.4$

(m) $t = 900, w = 1$ (n) $t = 900, w = 0.9$ (o) $t = 900, w = 0.7$ (p) $t = 900, w = 0.4$

(q) $t = 1000, w = 1$ (r) $t = 1000, w = 0.9$ (s) $t = 1000, w = 0.7$ (t) $t = 1000, w = 0.4$

(u) $t = 1100, w = 1$ (v) $t = 1100, w = 0.9$ (w) $t = 1100, w = 0.7$ (x) $t = 1100, w = 0.4$

Figure 3.3.3: Edgel level tracking with *Affine Warp Propagation* for different relation weights - frames 600 to 1100. The relations weights for each column are, from left to right: 1, 0.9, 0.7, and 0.4. The edgels are represented as blue dots and relations as lines with a colour related to the relation weight (from green for 1 to red for 0). In the first column, we also show, using red dots, the tracking results of a block tracked using *Point Matching Based Alignment* from Section 3.1.1.4. .



(a) $t = 0, n = 150$ (b) $t = 0, n = 50$ (c) $t = 0, n = 10$ (d) $t = 0, n = 3$

(e) $t = 100, n = 150$ (f) $t = 100, n = 50$ (g) $t = 100, n = 10$ (h) $t = 100, n = 3$

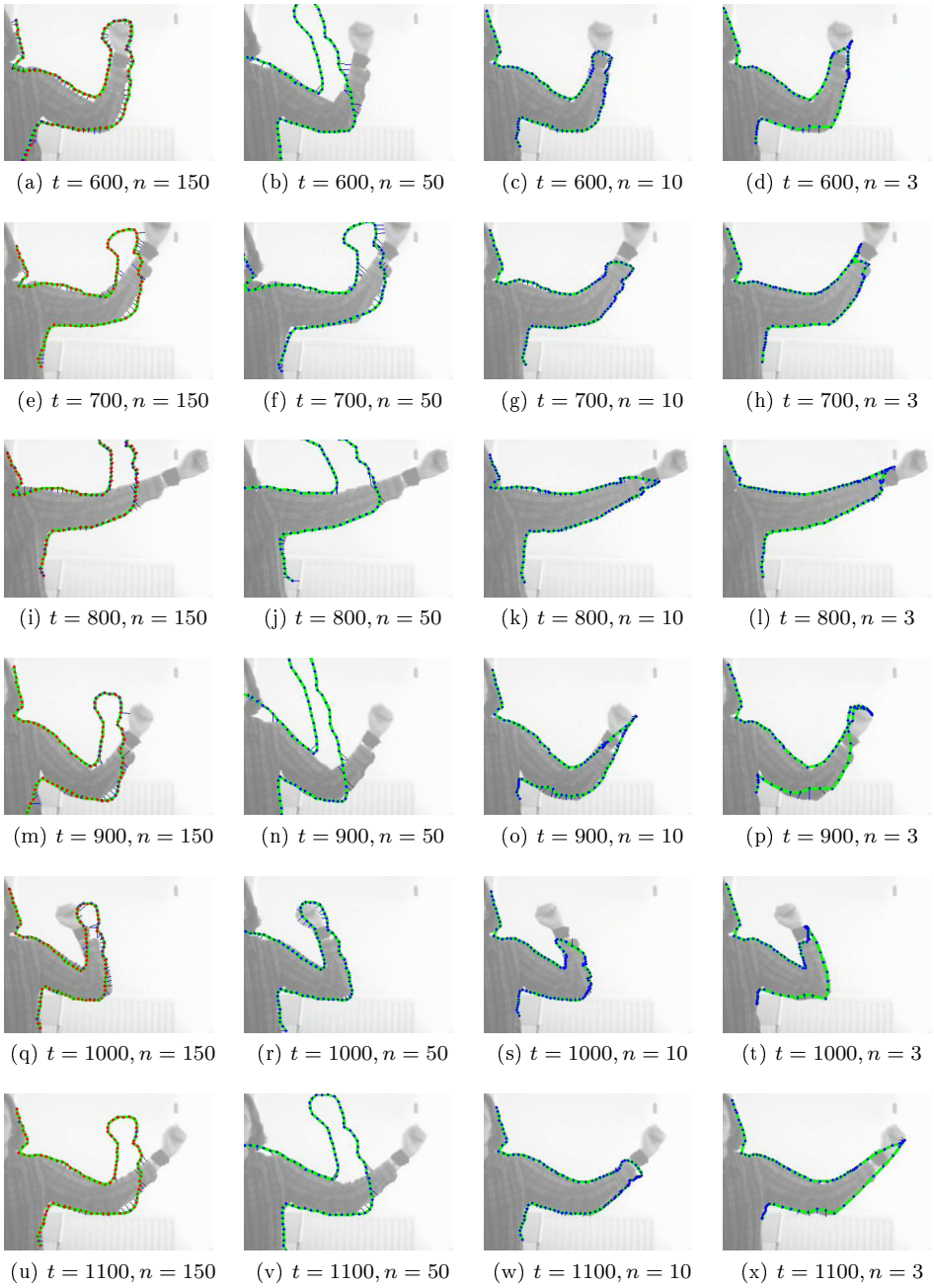
(i) $t = 200, n = 150$ (j) $t = 200, n = 50$ (k) $t = 200, n = 10$ (l) $t = 200, n = 3$

(m) $t = 300, n = 150$ (n) $t = 300, n = 50$ (o) $t = 300, n = 10$ (p) $t = 300, n = 3$

(q) $t = 400, n = 150$ (r) $t = 400, n = 50$ (s) $t = 400, n = 10$ (t) $t = 400, n = 3$

(u) $t = 500, n = 150$ (v) $t = 500, n = 50$ (w) $t = 500, n = 10$ (x) $t = 500, n = 3$

Figure 3.3.4: Edgel level tracking with *Affine Warp Propagation* for different number of iterations - frames 0 to 500. The number of propagation iterations for each column are, from left to right: 150, 50, 10, and 3. The relation weights are fixed to 1. The edgels are represented as blue dots and relations as lines with a colour related to the relation weight (from green for 1 to red for 0). In the first column, we also show, using red dots, the tracking results of a block tracked using *Point Matching Based Alignment* from Section 3.1.1.4. .



(a) $t = 600, n = 150$ (b) $t = 600, n = 50$ (c) $t = 600, n = 10$ (d) $t = 600, n = 3$

(e) $t = 700, n = 150$ (f) $t = 700, n = 50$ (g) $t = 700, n = 10$ (h) $t = 700, n = 3$

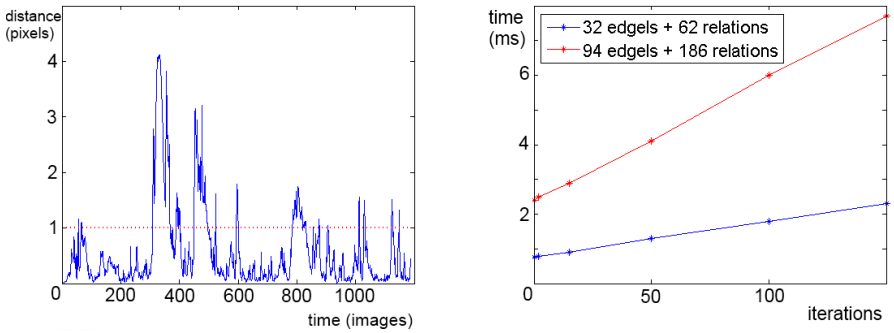
(i) $t = 800, n = 150$ (j) $t = 800, n = 50$ (k) $t = 800, n = 10$ (l) $t = 800, n = 3$

(m) $t = 900, n = 150$ (n) $t = 900, n = 50$ (o) $t = 900, n = 10$ (p) $t = 900, n = 3$

(q) $t = 1000, n = 150$ (r) $t = 1000, n = 50$ (s) $t = 1000, n = 10$ (t) $t = 1000, n = 3$

(u) $t = 1100, n = 150$ (v) $t = 1100, n = 50$ (w) $t = 1100, n = 10$ (x) $t = 1100, n = 3$

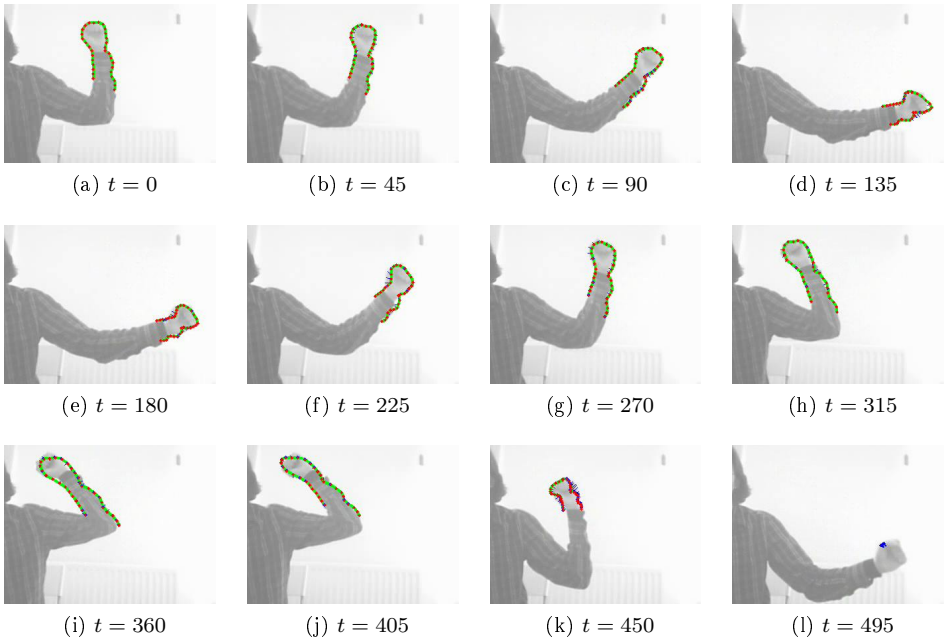
Figure 3.3.5: Edgel level tracking with *Affine Warp Propagation* for different number of iterations - frames 600 to 1100. The number of propagation iterations for each column are, from left to right: 150, 50, 10, and 3. The relation weights are fixed to 1. The edgels are represented as blue dots and relations as lines with a colour related to the relation weight (from green for 1 to red for 0). In the first column, we also show, using red dots, the tracking results of a block tracked using *Point Matching Based Alignment* from Section 3.1.1.4. .



(a) Distance (in pixels) between the tracking results of the block and the edgel level shown in the first column of Figure 3.3.2, 3.3.3 3.3.4, and 3.3.5.

(b) Computational time (in milliseconds) as a function of the number of propagation iterations for the sequences in Figures 3.3.4 and 3.3.5 (in red) and the sequence in Figure 3.3.7 (in blue).

Figure 3.3.6



(a) $t = 0$ (b) $t = 45$ (c) $t = 90$ (d) $t = 135$

(e) $t = 180$ (f) $t = 225$ (g) $t = 270$ (h) $t = 315$

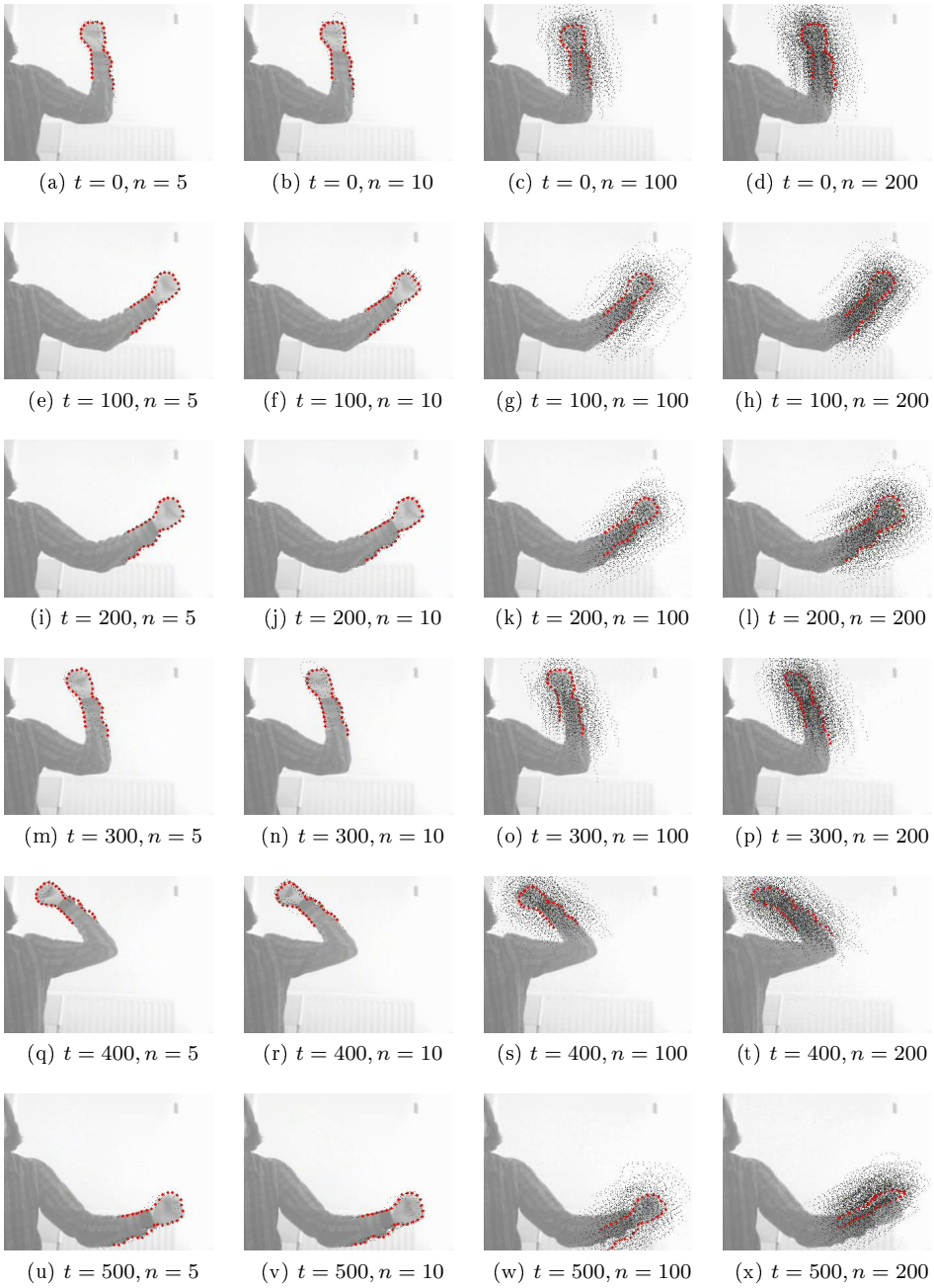
(i) $t = 360$ (j) $t = 405$ (k) $t = 450$ (l) $t = 495$

Figure 3.3.7: Edgel level tracking with *Affine Warp Propagation*. The relation weights are fixed to 1. The edgels are represented as blue dots and relations as green lines. We also show, using red dots, the tracking results of a block tracked using *Point Matching Based Alignment* from Section 3.1.1.4. .

particles required (5 particles and 10 particles with APF takes respectively 0.4 and, 0.9ms while 100 and 200 particles with PF takes 2 and 4ms). Once Belief Propagation is used, the advantage of the APF becomes even more pronounced since computational time of BP is quadratic with the number of particles.

One can argue that the APF will suffer more from clutters than the classic method due to its limited number of particles. Figure 3.3.10 shows the tracking results of the two methods for a more difficult sequence. In this experiment, we created a model using the Canny edge detector with high thresholds so that the model is defined with few edges and is therefore not very discriminative. We then track the object with much smaller thresholds so that many more similar edges are created. As we can see in the third image of the last row, the stripes on the shirt can easily be confused with the contours used in the model. This clutter quickly causes the classical particle filter with 200 particles to fail. In order to get a tracking that does not fail until the end of the sequence, the particle filter requires at least 500 particles. In comparison, the aligned particle filter still provides good tracking results even for 5 particles (notice that 5 particles have failed once during our tests, making 10 particles a safer value).

Another area where the tracking needs to be robust is in the case of occlusions. Figure 3.3.11 shows a first example where around two thirds of a hand are occluded. The method is able to deal with the heavy occlusion until the model is allowed to "rotate" around the thumb axis ($t = 100$) without constrains from any part of the model. The model is able to recover once the index reappears ($t = 200$). Any clutter combined with an occlusion (such as the page border aligned with the finger in $t = 250$ and $t = 500$) is the most delicate situation and is when the particles are the most important. In situations with fewer occlusions, as in Figure 3.3.12, we can see that the tracking results stay quite accurate during the whole sequence. Notice that the fingers are moving slightly in this sequence, forcing the model to adapt its scale. This is something that can potentially create some problems during the learning phase since the features will rely on the information coming from the block level to initialise their tracking.



(a) $t = 0, n = 5$ (b) $t = 0, n = 10$ (c) $t = 0, n = 100$ (d) $t = 0, n = 200$

(e) $t = 100, n = 5$ (f) $t = 100, n = 10$ (g) $t = 100, n = 100$ (h) $t = 100, n = 200$

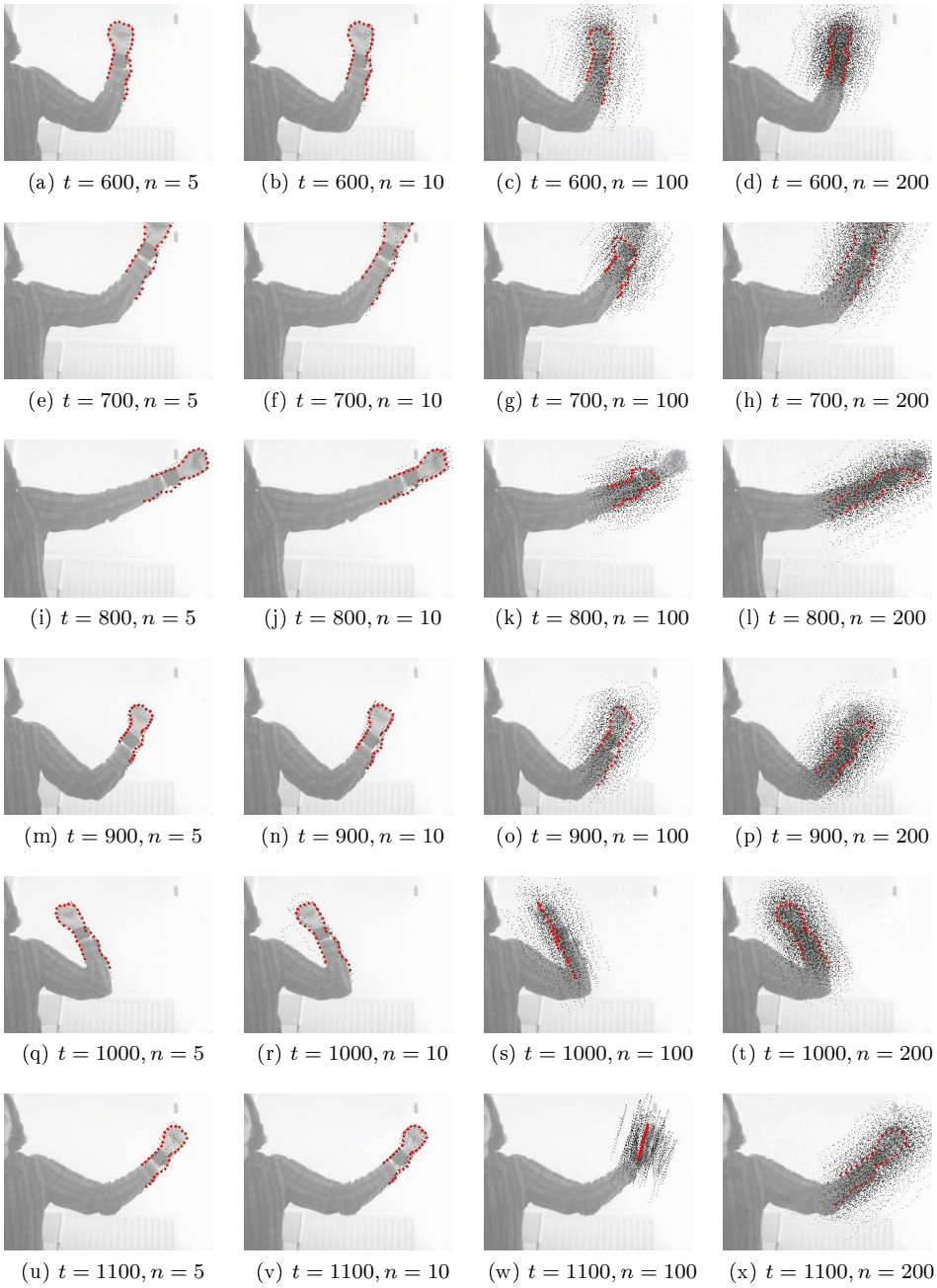
(i) $t = 200, n = 5$ (j) $t = 200, n = 10$ (k) $t = 200, n = 100$ (l) $t = 200, n = 200$

(m) $t = 300, n = 5$ (n) $t = 300, n = 10$ (o) $t = 300, n = 100$ (p) $t = 300, n = 200$

(q) $t = 400, n = 5$ (r) $t = 400, n = 10$ (s) $t = 400, n = 100$ (t) $t = 400, n = 200$

(u) $t = 500, n = 5$ (v) $t = 500, n = 10$ (w) $t = 500, n = 100$ (x) $t = 500, n = 200$

Figure 3.3.8: Block level tracking - frames 0 to 500. The two first columns correspond to the tracking results using the *Aligned Particle Filter* method presented in Section 3.2.2 (with 5 and then 10 particles). The last two columns correspond to the tracking results using the classical particle filter (with 100 and then 200 particles). The particles are shown in black and the most likely one in red.



(a) $t = 600, n = 5$ (b) $t = 600, n = 10$ (c) $t = 600, n = 100$ (d) $t = 600, n = 200$

(e) $t = 700, n = 5$ (f) $t = 700, n = 10$ (g) $t = 700, n = 100$ (h) $t = 700, n = 200$

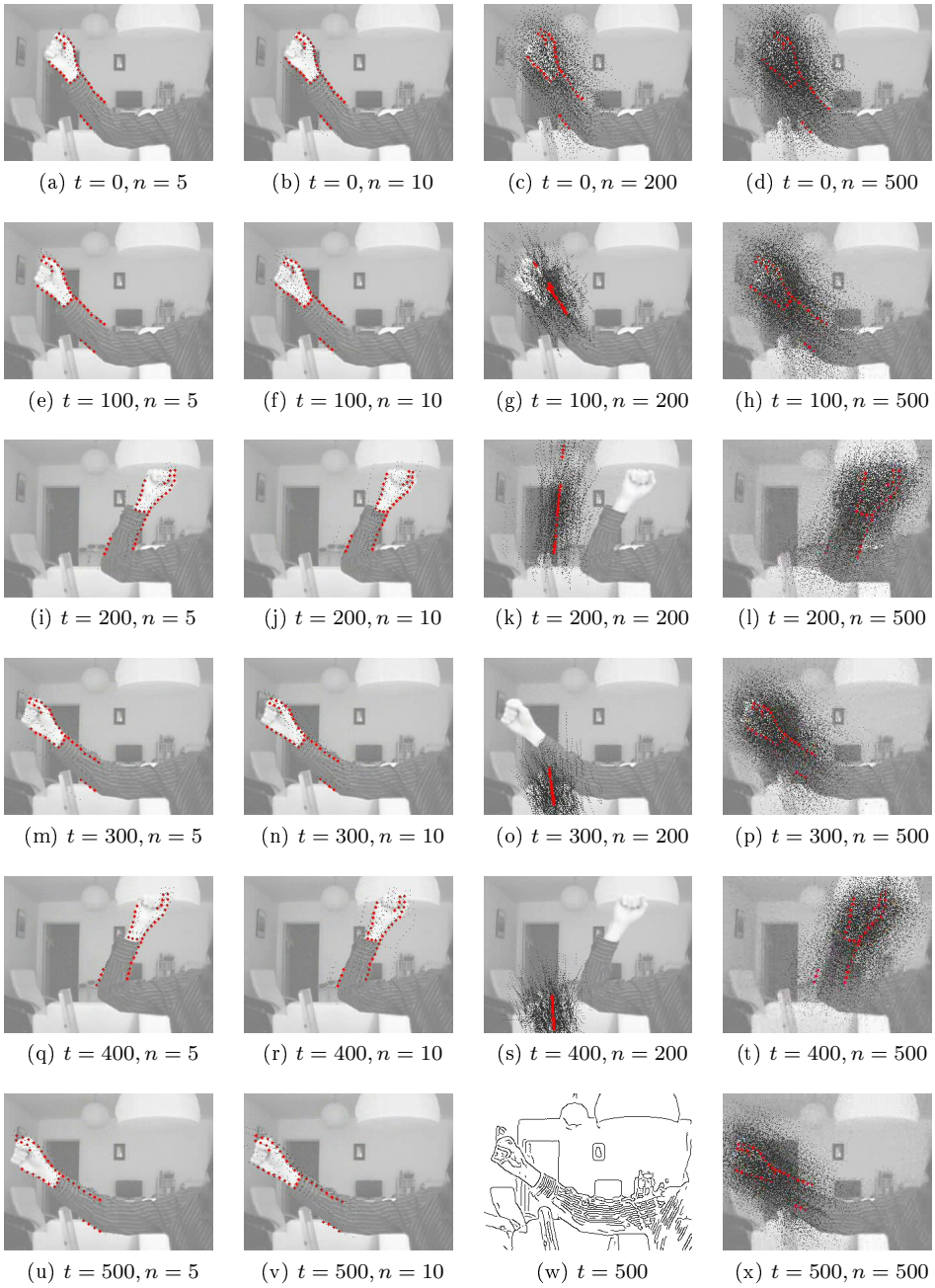
(i) $t = 800, n = 5$ (j) $t = 800, n = 10$ (k) $t = 800, n = 100$ (l) $t = 800, n = 200$

(m) $t = 900, n = 5$ (n) $t = 900, n = 10$ (o) $t = 900, n = 100$ (p) $t = 900, n = 200$

(q) $t = 1000, n = 5$ (r) $t = 1000, n = 10$ (s) $t = 1000, n = 100$ (t) $t = 1000, n = 200$

(u) $t = 1100, n = 5$ (v) $t = 1100, n = 10$ (w) $t = 1100, n = 100$ (x) $t = 1100, n = 200$

Figure 3.3.9: Block level tracking - frames 600 to 1100. The two first columns correspond to the tracking results using the *Aligned Particle Filter* method presented in Section 3.2.2 (with 5 and then 10 particles). The last two columns correspond to the tracking results using the classical particle filter (with 100 and then 200 particles). The particles are shown in black and the most likely one in red.



(a) $t = 0, n = 5$ (b) $t = 0, n = 10$ (c) $t = 0, n = 200$ (d) $t = 0, n = 500$

(e) $t = 100, n = 5$ (f) $t = 100, n = 10$ (g) $t = 100, n = 200$ (h) $t = 100, n = 500$

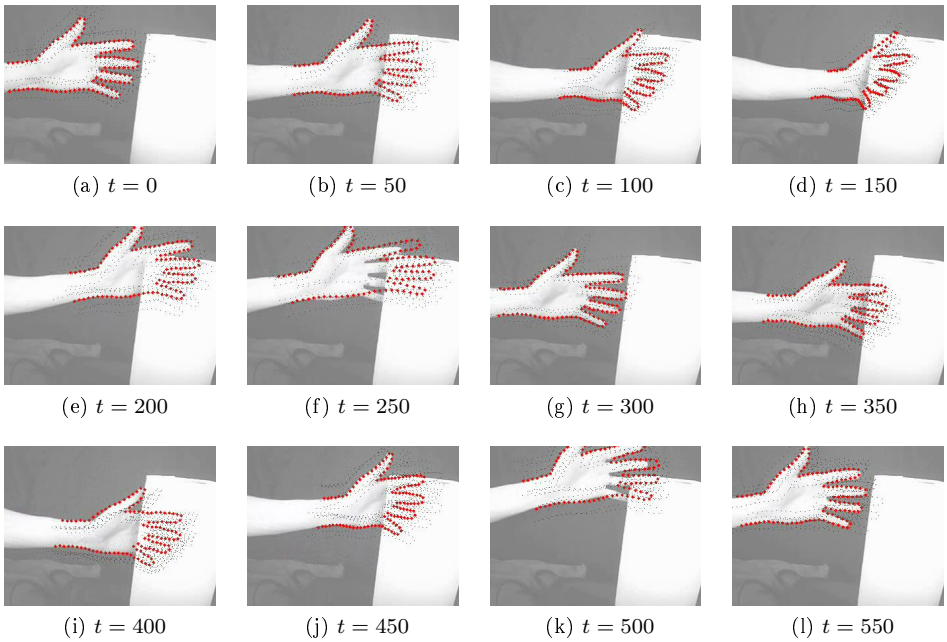
(i) $t = 200, n = 5$ (j) $t = 200, n = 10$ (k) $t = 200, n = 200$ (l) $t = 200, n = 500$

(m) $t = 300, n = 5$ (n) $t = 300, n = 10$ (o) $t = 300, n = 200$ (p) $t = 300, n = 500$

(q) $t = 400, n = 5$ (r) $t = 400, n = 10$ (s) $t = 400, n = 200$ (t) $t = 400, n = 500$

(u) $t = 500, n = 5$ (v) $t = 500, n = 10$ (w) $t = 500$ (x) $t = 500, n = 500$

Figure 3.3.10: Block level tracking in the presence of clutter. The first two columns correspond to the tracking results using the *Aligned Particle Filter* method presented in Section 3.2.2 (with 5 and then 10 particles). The last two columns correspond to the tracking result using the classical particle filter (with 200 and then 500 particles). The particles are shown in black and the most likely one in red. Notice that the most important source of clutter here is not the background but the stripes on the shirt that can easily be confused with the contours used in the model (see last row, third column, where we replaced the failed tracking by the contour image).

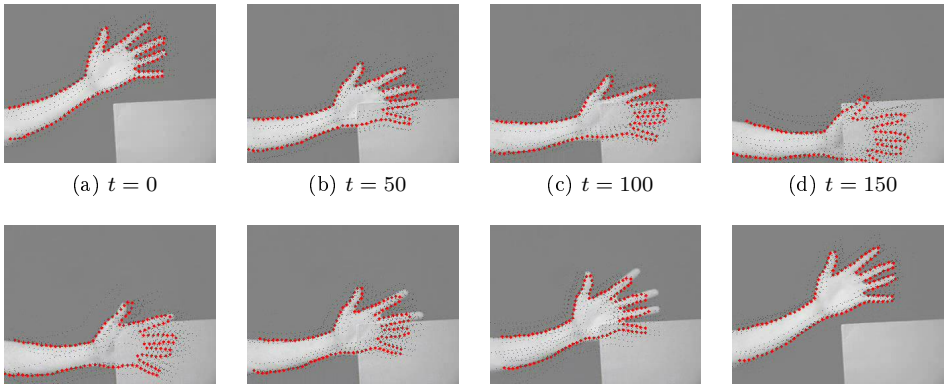


(a) $t = 0$ (b) $t = 50$ (c) $t = 100$ (d) $t = 150$

(e) $t = 200$ (f) $t = 250$ (g) $t = 300$ (h) $t = 350$

(i) $t = 400$ (j) $t = 450$ (k) $t = 500$ (l) $t = 550$

Figure 3.3.11: Block level tracking in the presence of occlusion. The particles ($n = 10$) are shown in black and the most likely one in red. The method is able to deal with heavy occlusion until the model is allowed to "rotate" around the thumb axis ($t = 100$) without constraints from any part of the model. The model is able to recover once the index reappears ($t = 200$). Any clutter combined with an occlusion (such as the page border aligned with the finger in $t = 250$ and $t = 500$) is the most delicate situation and is when the particles are the most important.



(a) $t = 0$ (b) $t = 50$ (c) $t = 100$ (d) $t = 150$

(e) $t = 200$ (f) $t = 250$ (g) $t = 300$ (h) $t = 350$

Figure 3.3.12: Other example of block level tracking in presence of occlusion. The particles ($n = 10$) are shown in black and the most likely one in red. In this case, a longer part of the arm is used in the model giving a much better result. Notice that the results at time $t = 250$ and $t = 300$ are due to the fact that the fingers are further apart than in the original model and not to the occlusion (frame 200 and 350 are indeed fine). .

3.4 Discussion

In this chapter, we focused on tracking the two levels of the graph. Since the feature level is composed of many nodes, we developed a solution that allows the information to be propagated between the nodes using a very small computational time. This approach is not as robust as more expensive MHT methods but is sufficient in our case since the feature level will be able to assist its own tracking using the information coming from the block level.

For the block level, we decided to use the more conventional Nonparametric Belief Propagation but made some modifications to the particle filter in order to obtain a method that can be applied in real-time without compromising its robustness.

From the experiments, we saw that the flexibility of the feature level can easily be tuned using a weight factor on the relations between features. This will allow us to use the feature level to track any non-rigid part of the object until enough evidence is gathered to segment it from the rest of the object.

To summarise, this chapter presented methods able to track both levels in real-time in a way corresponding to the requirement of each level: flexibility for the feature level and robustness for the block level.

Chapter 4

Model Learning

In this chapter, we consider that some tracking results are available and focus on learning a two-level articulated model as defined in Chapter 2. The learning of this type of model can be decomposed into the following processes:

- **Feature Level**
  - **Extraction of the visual features belonging to the object of interest**. The extraction of the visual features itself have been discussed in Chapter 2 (where we choose to use edgels), the only thing left is then to determine which features belong to the object of interest. In our case, this corresponds to defining which features are correlated with one of the rigid blocks of the graph (i.e. those with a rigid relation with one of the blocks).
  - **Detection and representation of rigid spatial relations existing between these features.** Based on the tracking method proposed in Section 3.2.1, the model of a rigid spatial relation must be able to provide a mean relative position between two connected features and a weight that expresses the confidence in the rigidity of this relation.
- **Block Level**
  - **Discovery of rigid blocks based on the motion of the visual features**. Given that all the features are first grouped into a single graph, this step corresponds more to a segmentation of the features into rigid parts. The extracted blocks being modelled as a group of correlated edgels, they are mainly defined by the spatial relative position of their associated features.
  - **Detection and representation of articulated spatial relations existing between these blocks**. The simplest way to represent an articulated spatial relation between two blocks is through the position

of the articulation (the joint) with respect to the two rigid blocks. Given that we chose in Section 3.2.3 to use Belief Propagation for the tracking of the block level, the relations must be represented using a probabilistic distribution. For the relative position of an articulation, a Gaussian model is sufficient but we will also explore other alternatives for the representation of more complex relations.

From the list above, we can see that modelling an articulated object mainly corresponds to two types of processes: the modelling of the spatial relations existing between its different parts (features and blocks) and the segmentation of the features into rigid blocks. Since, in our case, learning is simultaneous to tracking, modelling and segmentation must respect a few constraints:

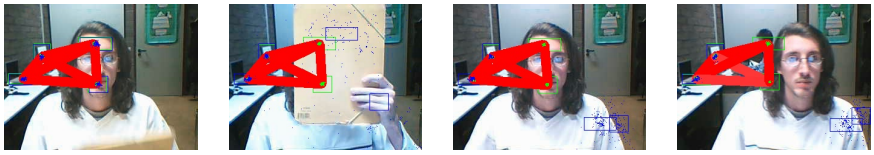
- As for any **on-line learning**, the model is only providing one observation at a time with no possibility to store previous observations or to access to future ones. The model has, therefore, to summarise properly previous observation while maintaining a sufficient informative level to adapt to future data.
- With no information about future observations, there is **no guarantee correlation between past and future observations**. We can, for example, observe an object that only starts moving at a time t. In that case, all the static observations obtained until then will be of no use (to the contrary) to assist its tracking once it starts to move.
- Given that the learnt models are used to assist tracking, we face the necessity to provide a **real-time solution** for tracking and modelling combined. With tracking already being a challenge for real-time on its own, it is mandatory that the learning process require as few computational time as possible.

With those difficulties in mind, we will first present in Section 4.1 existing solutions for spatial relation modelling and for feature segmentation that can be considered in our case. We then explain in Section 4.2 how we adapt those methods or develop new ones in order to address all the difficulties presented above. Finally, we analyse our methods in Section 4.3 for the situation where proper (while noisy) tracking results are provided.

4.1 Background

4.1.1 Incremental Model Learning

Learning a model incrementally brings two main problems. The first one is in the choice of the model level of complexity. If this model is to be used for real-time applications, it should maintain a complexity that is as low as possible. For example, if we are only interested in modelling rigid relations, using a simple parametric model like a Gaussian distribution would be ideal. Unfortunately,



(a) Frame 21. (b) Frame 51. (c) Frame 101. (d) Frame 121.

Figure 4.1.1: Small example of tracking results using a Gaussian mixture model to represent the spatial constraints (thick red lines) between the four nodes of the graph. The green boxes represent areas tracked using the graph and the blue boxes represent areas tracked individually. Since the head is not moving at the beginning of the video, all the spatial relations are learnt as rigid (a single mixture in the Gaussian). These models are useful to assist tracking in case of occlusion (see in frame 51) but also tends to exert overly strong bias that could hamper tracking (as in frame 121).

it is impossible to know at the time of creation of the model if the spatial distribution will indeed be a rigid one. If it is not, the Gaussian model will be unable to properly represent the spatial relation and will cause undesired rigid constraints between the corresponding features that could cause the feature tracking to fail. In the literature, the commonly admitted solution is to use a Gaussian mixture with a variable number of components. In this way, we benefit both from the flexibility of the non-parametric model (by allowing any number of Gaussians in the mixture) and from the efficiency and low memory requirement of the parametric model.

The second problem associated with an incremental modelling arises when the model is directly used for tracking during its learning phase. In this case, we have a closed-loop between learning and tracking where each process influences the result of the other. This means that, even if the Gaussian mixture is able to increase its number of components, the biased tracking results might simply never diverge enough from the model to initiate the creation of a new Gaussian in the mixture. Consider the example of Figure 4.1.1, where the head is first static for a while and then starts to move. With the head moving, we expect new observations to invalidate the rigid model and therefore to cause the mixture models to create new Gaussians. Unfortunately, the bias created by the relations makes the new observations to appear still rigid, preventing the detection of any changes in the rigidity of the model.

While the problem of incremental learning of Gaussian mixture models (GMM) is well addressed in the literature, the case where they are simultaneously used for tracking (and therefore have an influence on future observations) is not as common. In this section, we will present existing methods used to solve the on-line learning problem of increasingly complex models: a single Gaussian, a Gaussian mixture model (GMM) with a fixed number of components, and finally, a GMM with a variable number of components. To the best of our knowledge, no solution has been proposed to address the problem of account-

ing for the unreliability of a model if it is immediately used during its learning phase.

4.1.1.1 Gaussian Model

This is the simplest possible incremental learning scenario: a single Gaussian model. If the dataset $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$ and their respective weights $[w_1, \ldots, w_N]$ were available all at once, the weight π, mean $\hat{\mathbf{x}}$, and covariance matrix $\mathbf{C}$ of the Gaussian model would be given by

$$\pi = \sum_{i=1}^{N} w_i \tag{4.1.1}$$

$$\hat{\mathbf{x}} = \frac{1}{\pi} \sum_{i=1}^{N} w_i \mathbf{x}_i \tag{4.1.2}$$

$$\mathbf{C} = \frac{1}{\pi} \sum_{i=1}^{N} w_i \left(\mathbf{x}_i - \hat{\mathbf{x}}\right) \left(\mathbf{x}_i - \hat{\mathbf{x}}\right)^{\mathrm{T}} . \tag{4.1.3}$$

Now, consider that the dataset $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$ is segmented into a group of subsets $[\mathbf{S}_1, \ldots, \mathbf{S}_M]$ where $\mathbf{S}_i \cap \mathbf{S}_j = \emptyset \forall \{i, j\}$ and $\mathbf{X} = \cup_{k=1}^{M} \mathbf{S}_k$. We then define a set of Gaussian models $[\theta_1, \ldots, \theta_M]$ so that each θ_i is calculated based on the subset $\mathbf{S}_i$. If the set of Gaussian models is the only information available, the Gaussian model of the complete dataset $\mathbf{X}$ can still be obtained using

$$\pi = \sum_{i=1}^{M} \pi_i \tag{4.1.4}$$

$$\hat{\mathbf{x}} = \frac{1}{\pi} \sum_{i=1}^{M} \pi_i \hat{\mathbf{x}}_i \tag{4.1.5}$$

$$\mathbf{C} = \frac{1}{\pi} \sum_{i=1}^{M} \pi_i \left[\mathbf{C}_i + \left(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}\right) \left(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}\right)^{\mathrm{T}}\right] . \tag{4.1.6}$$

These equations allow incremental learning of a Gaussian model when data arrives by blocks. They are a generalisation of the case where the model is updated with a single observation $\{\mathbf{x}_t, w_t\}$ at a time. Indeed, this latter case corresponds to merging two Gaussians where one of them has the parameters $\{\pi = w_t; \hat{\mathbf{x}} = \mathbf{x}_t; \mathbf{C} = 0\}$ and the other has been learnt from the entire remaining dataset $[\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}]$. In this case, the incremental learning of a Gaussian model at time t is given by

$$\pi_t = \pi_{t-1} + w_t \tag{4.1.7}$$

$$\hat{\mathbf{x}}_t = \frac{1}{\pi_t}\left[\pi_{t-1}\hat{\mathbf{x}}_{t-1} + w_t\mathbf{x}_t\right] \tag{4.1.8}$$

$$\mathbf{C}_t = \frac{1}{\pi_t}\Big[\pi_{t-1}\mathbf{C}_{t-1} + \pi_{t-1}\left(\hat{\mathbf{x}}_{t-1} - \hat{\mathbf{x}}_t\right)\left(\hat{\mathbf{x}}_{t-1} - \hat{\mathbf{x}}_t\right)^{\mathrm{T}} + w_t\left(\mathbf{x}_t - \hat{\mathbf{x}}_t\right)\left(\mathbf{x}_t - \hat{\mathbf{x}}_t\right)^{\mathrm{T}}\Big] . \tag{4.1.9}$$

4.1.1.2 GMM with a fixed number of components

This time, the model is defined by a mixture of M Gaussians $\mathbf{\Theta} = [\theta_1, \ldots, \theta_M]$ with weights $[\pi_1, \ldots, \pi_M]$. In this case, the main problem is to know how to divide the dataset $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$ between the M Gaussians. The Expectation Maximisation (EM) algorithm [27] is commonly used to search for the solution. The EM algorithm is an iterative procedure that searches for a local maximum of the log-likelihood of the dataset given the GMM, i.e.

$$\hat{\mathbf{\Theta}} = \arg\max_{\mathbf{\Theta}} \left(\log p\left(\mathbf{X}; \mathbf{\Theta}\right)\right) , \tag{4.1.10}$$

$$\text{with} \qquad p\left(\mathbf{X}; \mathbf{\Theta}\right) = \sum_{i=1}^{N} w_i p_i\left(\mathbf{x}_i; \mathbf{\Theta}\right) / \sum_{i=1}^{N} w_i \tag{4.1.11}$$

$$\text{and} \qquad p_i\left(\mathbf{x}_i; \mathbf{\Theta}\right) = \sum_{j=1}^{M} \alpha_{i,j} p_{i,j}\left(\mathbf{x}_i; \theta_j\right) / \sum_{k=1}^{M} \alpha_{i,j} . \tag{4.1.12}$$

The values $\alpha_{i,j}$ are used to represent the likelihood that sample i was generated by the component distribution j. These values are not known but can be estimated if the parameters $\hat{\mathbf{\Theta}}$ are known. Given that neither the $\alpha_{i,j}$ or $\hat{\mathbf{\Theta}}$ are known, the idea of the EM algorithm is to estimate both simultaneously by iterating between their calculation (using the current value of the other as parameter). For the case of a Gaussian mixture model, the two steps of the EM algorithm are given by

1. **Expectation**

   (a) For each pair $\{i, j\} \in \{[1, \ldots, N], [1, \ldots, M]\}$,

   $$\alpha^*_{i,j} = \frac{1}{(2\pi)^{d/2}\left|\mathbf{C}_j\right|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \hat{\mathbf{x}}_j)\mathbf{C}_j^{-1}(\mathbf{x}_i - \hat{\mathbf{x}}_j)^{\mathrm{T}}\right) , \tag{4.1.13}$$

   where d the dimension of the space and $|\mathbf{C}_j|$ is the determinant of $\mathbf{C}_j$.

(b) For each pair $\{i,j\} \in \{[1,\ldots,N],[1,\ldots,M]\}$,

$$\alpha_{i,j} = \pi_j \alpha^*_{i,j} / \sum_{k=1}^{M} \pi_k \alpha^*_{i,k} \,. \tag{4.1.14}$$

2. **Maximisation**

(a) For each $j \in [1,\ldots,M]$,

$$\pi_j = \sum_{i=1}^{N} \alpha_{i,j} \tag{4.1.15}$$

$$\hat{\mathbf{x}}_j = \frac{1}{\pi} \sum_{i=1}^{N} \alpha_{i,j} \mathbf{x}_i \tag{4.1.16}$$

$$\mathbf{C}_j = \frac{1}{\pi} \sum_{i=1}^{N} \alpha_{i,j} \left(\mathbf{x}_i - \hat{\mathbf{x}}_j\right) \left(\mathbf{x}_i - \hat{\mathbf{x}}_j\right)^{\mathrm{T}} . \tag{4.1.17}$$

In the case of on-line learning, the dataset $\mathbf{X}$ is not available. While we have seen that the Gaussian parameters can be easily updated to include new data, it is impossible to adapt the weights $\alpha_{i,j}$ without the data. The general approach [2, 38, 75, 93] to this problem is to assume that the weights $\alpha^*_{i,j}$ will not vary too much and therefore to approximate them with their value at the moment the sample is observed. If the number of component in the GMM is fixed, it is clear that this assumption is erroneous (since the variations in the parameters of the Gaussians can be important in order to cover all the new data). Conversely, if the number of components can be modified, the process of splitting and merging the components can be designed in order to keep this assumption as true as possible.

4.1.1.3 GMM with a variable number of components

A common approach to on-line learning of GMM with a variable number of components is to consider the fusion of two GMMs of sizes M and N into a GMM of size $K \leq M+N$. To do so, Hall and Hicks [38] propose to first concatenate the two GMMs and then determine the optimal model size by considering models of all lower complexities and choosing the one that gives the smallest penalised log-likelihood. Another solution, proposed by Song and Wang [75], was to compare the Gaussians from the two GMMs and merge the pairs that demonstrate equivalent means and covariances. While the first solution necessitates a high computational cost dues to the test of different possible values for K, the second one has a tendency to produce more clusters than needed. Indeed, the second solution only merges Gaussians if they are similar and not because they can form a proper Gaussian distribution once merged. On top of that, both methods require the new data to come in blocks that can be represented by a GMM.

Zivkovic and van der Heijden [93] proposed a solution where each new sample is added to one of the components of the current GMM. In order to select the number of components, their on-line algorithm starts with a large number of randomly initialised components and uses a forgetting term on the Gaussian's weights to eliminate the components with a negative weight. The main problem with this approach is that it violates the assumption that the weights $\alpha^*_{i,j}$ will not vary too much over time if the samples are temporally coherent.

Arandjelovic and Cipolla [2] proposed to use this temporal coherence assumption in order to be able to add samples one-by-one into the GMM. Their central idea is to use a *Historical Gaussian Mixture Model* (HGMM) that corresponds to the GMM just after the last change in the number of components was done. They show that this HGMM provides enough information to decide when and how to split and merge the components of the current GMM without using the past data samples. Using the one-to-one correspondence between the Gaussians of the two mixtures, they are able to produce, at each time t and for each current Gaussian, a split version of the current Gaussian. This split version is a two-Gaussian mixture model composed of the historical Gaussian and the difference between the current and historical Gaussians. By generating samples from this 2-GMM, and using the Minimal Description Length (MDL) criterion [68], they are able to decide which of the current or split solution provides the best model. The use of this *Historical Gaussian Mixture Model* is an elegant solution because it does not require any previous data sample (low memory cost) and only requires a single splitting test of the Gaussians in the mixture (low computational cost). Unfortunately, this approach relies too much on the temporal consistency of the data and causes unsatisfactory fitting results if this assumption is not verified. For example, it fails if a part of the new data is well explained by the historical model. Indeed, if a Gaussian is split, this new data will still be contained in the component resulting from the difference between its current and historical Gaussians while it should be in the historical model. Another situation that is problematic is when the data correspond to fast moving objects (or when a few frames are missing). In this case, we might end up with "wide" Gaussians that will under-fit the data without any chance of recovery.

4.1.2 Rigid Block Discovery

Rigid block discovery through motion segmentation has been a very popular domain and many solutions have been proposed using different approaches like Multi-body Factorisation [20, 83, 88], Expectation Maximisation [37, 51], Spectral Clustering [89, 92], Affinity Propagation [34, 69], GraphCut [12, 50, 63], ... Giving a full literature review on this topic is outside the scope of this thesis but we can safely say that most of the solutions proposed are based on the three following steps:

1. Compute an affinity matrix/graph representing the likelihood of each pair of elements to belong to the same rigid block. This affinity can be based

on the trajectories, optical flow, distances between points,...

2. Initialise a set of rigid blocks/layers by clustering the points using the affinity matrix/graph. This can be done one block at a time, with an increasing number of blocks, or all at once.

3. Improve the result by iteratively updating the parameters of the blocks and their relations with the points.

In this section, we will limit the literature review to two articles that each address a specific problem we have to face in this thesis. The first one, by Yan and Pollefeys[88], focuses on segmenting the rigid parts of an articulated object. The second one, by Shrinivas and Stanley [65], is one of the rare articles that address the problem of motion segmentation in the context of real-time on-line applications.

4.1.2.1 Articulated Motion Segmentation

Yan and Pollefeys[88] state that segmentation algorithms assuming independent motions cannot generally be applied to articulated motions. To remedy this problem, they complement the popular multi-body factorisation method [20, 83] with a RANSAC approach in order to get a robust extraction of the rigid parts of the articulated motion. Their method is based on the trajectories of a set of features (stored into the trajectory matrix W) and can be summarised as follows:

1. *The Prior Matrix*

   (a) Build an affinity matrix $M = W^T W$ from the trajectory matrix W and normalise it into $N = D^{-1/2} M D^{-1/2}$ where $D_{ii} = \sum_j M_{ij}$.

   (b) Form a matrix $X_{p \times k}$ whose columns are the k dominant eigenvectors of N and then normalise it to get $Y_{p \times k}$. Each row y_i of Y is the normalised spectral representation of the trajectory of feature i in R^k.

   (c) Build the prior matrix P with

   $$P_{ij} = \frac{2}{\sqrt{\pi}} \int_0^{y_i y_j^T} e^{-t^2} \mathrm{d}t \,, \tag{4.1.18}$$

   where P_{ij} represents the probability of trajectory i belonging to the same motion as trajectory j.

2. *RANSAC with Priors*

   (a) Form a sample set of k trajectories and instantiate a motion model from them. The samples are chosen using the prior matrix P by

      i. randomly choosing the first trajectory s_1 based on the probability distribution formed by the sums of each row of P.

ii. Randomly choosing the 2nd to the kth trajectories $s_1, \ldots, s_k$ based on a probability distribution formed by the priors related to s_1.

(b) Determine the set of trajectories S_i that are within a threshold t of the motion model.

(c) Repeat (a) and (b) N times and select the largest consensus set to extract a motion model.

(d) Remove this set from the original data and repeat (c) until either the data is exhausted or no more models with a size bigger than some threshold T can be found.

The *prior matrix* step above is common for the multi-body factorisation methods but is usually directly used to cluster the features into rigid groups. By using this matrix only as a prior to RANSAC, Yan and Pollefeys have shown that it is possible to robustly extract the rigid parts of an articulated object despite their correlation. The main drawback of this method though is its necessity to have an access to the full trajectories of the features to segment.

4.1.2.2 Real-time On-line Motion Segmentation

Most of the methods are based on the entire video sequence (or at least, the full tracking results of the points) and/or are too computationally expensive for real-time applications. One solution that distinguishes itself is the one proposed by Shrinivas and Stanley [65] because it specifically addresses the problem of motion segmentation in the context of real-time on-line applications. Their algorithm is based on the comparison of the movement of a set of feature points between a reference frame and the current one. It can be summarised as follows:

- *Grouping Procedure*

  1. Create a new group starting from a random seed point (feature point) and iteratively
     (a) add neighbouring points that have a movement similar to the group. If no more point is added then stop.
     (b) adjust the movement description of the group.
  2. Repeat (a) until all the points are in a group.
  3. Repeat (b) N times and create groups using the points that were always grouped together in (b). The other points are marked as ungrouped.

- *Maintaining Feature Groups Over Time*

  1. *Grouping.* Run the *Grouping Procedure* for the set of ungrouped points from the previous frame based on their new position in the current frame.

2. *Splitting.* For each group obtained in the previous frame that has now lost at least $\lambda\%$ of their points,
   (a) update the reference frame used to calculate the point movements.
   (b) run the *grouping procedure* on the points of the group.
3. *Assimilation.* For each ungrouped point, include it in a neighbouring group if this group can explain the point movement with an error lower than a given threshold.

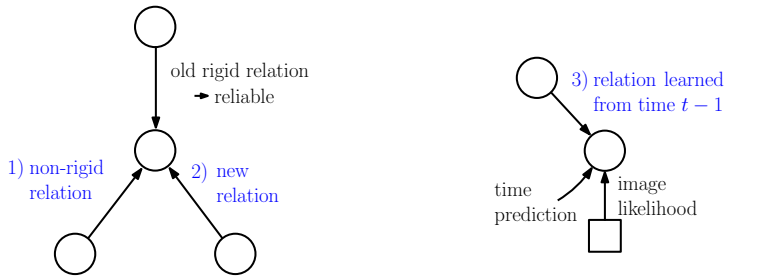
While the method is not specifically designed for articulated objects, it seems, from its comparison with the approach of Yan and Pollefeys, that there is no reason it should not work as well. This approach therefore seems to match all the criteria for our case, i.e. the real-time incremental segmentation of articulated objects. This being said, one drawback of the method is that its segmentation only relies on two positions for each feature without any other information from past observations. It means that the result will be the same for an object that is back to its original position as for an object that has never moved.

4.2 Developed Solutions

In this section, we first present the solutions we developed to learn spatial relations models and then finish with the automatic segmentation of the features into rigid blocks. The learning process of the relation models will be divided into three parts, each for a different type of relations: rigid, flexible, and articulated. Rigid relations will be described using a Gaussian model, flexible relations using a GMM and articulated ones using a parametric model of the position of the joint. The common point between the three models is the fact that they account for the uncertainty in their ability to properly assist the tracking without exerting an overly strong, counterproductive bias.

Section 4.2.1 will discuss the case of rigid relations that we previously published in *On-line simultaneous learning and tracking of visual feature graphs* [25]. Section 4.2.2 will then present the case of flexible relations as it was published in our article *On-line Learning of Gaussian Mixture Models - a Two-Level Approach* [26]. Section 4.2.3 will follow with the learning of articulated relations. Finally, Section 4.2.4 will discuss the automatic segmentation of features into rigid blocks.

In order to be consistent with the other chapter of this thesis, we will keep the notations introduced in Section 2.2.3 of Chapter 2; i.e. a relation between two nodes i and j (feature or block) is represented by the conditional distributions $\psi_{i,j}(\mathbf{r}_{i|j})$ and $\psi_{j,i}(\mathbf{r}_{j|i})$ where $\psi_{i,j}(\mathbf{r}_{i|j})$ represents the distribution of the relative position $\mathbf{r}_{i|j}$ of node i expressed in the affine coordinate system of node j.



(a) Uncertainty 1) in the model and 2) in its parameters.

(b) Uncertainty about the future.

Figure 4.2.1: Sources of uncertainty.

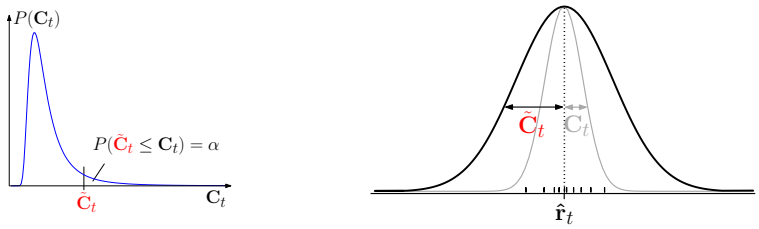
4.2.1 Uncertain Rigid Relation Learning

If we assume that the spatial relations are rigid, we can decide to represent them with a Gaussian model. Given that this model is learnt incrementally, we can consider three sources of uncertainty in its capability to predict the relation in the next frame:

1. **Uncertainty in the choice of model.** When connections between nodes (features or rigid blocks) of the graph are created, there is no way to know which type of spatial relation these nodes will demonstrate. If we decide to represent the relations with a Gaussian model, it will oversimplify the description of all the non-rigid relations and therefore create a bias in the tracking of the associated nodes. Even if we propose to quickly remove all the non-rigid connections from the graph, this lack of rigidity may not appear in the tracking results due to the bias created by the Gaussian model. If we still wish to use the Gaussian model (for example because it has a very low computational cost), it is mandatory to combine it with a representation of the level of confidence in the rigidity of the relation.

2. **Uncertainty in the model parameters.** Even if the relation can be represented by a Gaussian model, we must be aware that the Gaussian parameters might not be exact yet. Indeed, the model is used for tracking from the beginning, where the estimation of the parameters is based on a small set of observations. If the variance have been underestimate, it will again create a bias in the tracking that might result in losing the element of interest (after the first frame, we just have one observation so this is very likely that the variance is indeed underestimate).

3. **Uncertainty in the correlation between previous and future observations.** Assuming that a relation will always be rigid simply because it has demonstrated a Gaussian behaviour for a long period is a mistake. Consider, for example, the case of a person sitting on a chair for a while.



(a) (b)

Figure 4.2.2: One-dimensional example of the variance distribution. We choose a variance $\tilde{\mathbf{C}}_t$ in a way that bounds the risk of underestimating the true variance.

> From observing this situation, we could create a rigid model between the chair and the person and give a complete confidence to its rigidity. Once the person moves away from the chair, the rigid relation will force the two of them to respect the rigid relation, thus creating a bias that will likely cause the tracking to fail.

The uncertainty in the model and its parameters can be estimated using only past observations. This means that, after a while, it will be possible to decide with high confidence if the relation can be modelled with the chosen model or not. By contrast, the uncertainty about the future will remain constant since the relation can be stationary for an undetermined period of time. Let us discuss those three sources of uncertainty and see how we can adapt the classic Gaussian model to account for them.

4.2.1.1 Uncertainty in the Parameters

When we have only a few observations, the Gaussian parameters obtained from the maximum-likelihood estimation presented in Section 4.1.1.1 are uncertain (i.e. they can be incorrectly estimated). By getting more and more samples, the estimations of these parameters will become more and more precise but, in the mean time, we need to make sure that an incorrect estimation of the parameters will not create a negative bias in the tracking. It turns out that the influence of the uncertainty in the Gaussian's mean is insignificant compared to the influence of the uncertainty in its covariance; we therefore neglect the uncertainty related to the mean. This leaves us with the necessity to adapt the value of the covariance. It is clear that the smaller the covariance, the stronger the bias on the tracking. We thus need to augment it as a function of the number of observations. We consequently choose a covariance $\tilde{\mathbf{C}}$ in such a way that it bounds the risk of underestimating its true value, i.e., $P(\tilde{\mathbf{C}} \leq \mathbf{C}) = \alpha$, where conventionally $\alpha = 0.05$ (see Figure 4.2.2). Since empirical estimates of

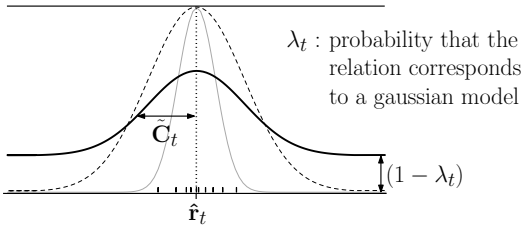


Figure 4.2.3: Example of an *uncertain* relation.

variance follow a chi-square distribution,

$$\tilde{\mathbf{C}}_t = \frac{\pi_t}{\chi^2_{\pi_t-1}(\alpha)}\mathbf{C}_t\,, \tag{4.2.1}$$

where $\chi^2_{\pi_t-1}(\alpha)$ is the chi-square inverse cumulative distribution function. A Gaussian model that takes the uncertainty in the parameters into account is then simply defined by

$$\psi_{i,j,t}(\mathbf{r}_{i|j,t}) = e^{-\frac{1}{2}(\mathbf{r}_{i|j,t}-\hat{\mathbf{r}}_{i|j,t})\tilde{\mathbf{C}}^{-1}_{i|j,t}(\mathbf{r}_{i|j,t}-\hat{\mathbf{r}}_{i|j,t})}\,. \tag{4.2.2}$$

4.2.1.2 Uncertainty in the Parametric Model

Since we use a parametric model, the range of relations that can be represented with this model is limited. It is obvious that not all the observed relations will correspond to the chosen model. As motivated earlier, we are mainly interested in learning these relations that fit the chosen parametric model. The relations that do not correspond are simply discarded. The problem is that we may need a lot of observations to be able to conclude that the model is not appropriate. And during this time, the model is still used to track the features which may create a strong bias in the tracking causing it to fail.

As the parametric model corresponds here to a Gaussian distribution around a rigid position, the questions are "How to estimate the probability that the observed relation is rigid?" and "How to modify the potential function so that it does not create bias in the tracking?"

A Potential Function with Limited Bias

Let us first assume that we are able to estimate the probability that the relation corresponds to a Gaussian model. We call it the *fidelity* λ_t for the probability at a given time t. Consider now the two extreme situations $\lambda_t = 1$ and $\lambda_t = 0$.

If $\lambda_t = 1$, then we can simply use the potential function of Equation 4.2.2 without risk of creating bias.

If $\lambda_t = 0$, then the learnt potential function is completely wrong and must be replaced with something else. But since the observations are not kept in

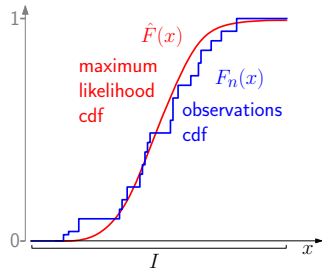


Figure 4.2.4: Method inspired from the Kolmogorov-Smirnov test.

memory, what can be used to compute this "something else"? The answer is obvious: "Nothing!". In this case, all we can do is to say that we know nothing about the distribution of the relation. Then, the only thing we can do is to represent this lack of knowledge with a uninformative uniform potential.

In practise, nothing is black or white and a model may be more or less appropriate for the relation. Therefore, we represent the potential function by a weighted sum of the learnt model and a uniform potential. The probability of observing a relation $\mathbf{r}_{i|j,t}$ between features i and j at time t is then given by

$$\psi^{+}_{i,j,t}(\mathbf{r}_{i|j,t}) = \lambda_t e^{-\frac{1}{2}(\mathbf{r}_t - \hat{\mathbf{r}}_t)\tilde{\mathbf{C}}_t^{-1}(\mathbf{r}_t - \hat{\mathbf{r}}_t)} + (1 - \lambda_t), \tag{4.2.3}$$

where we remove the i and j indices for conciseness. Notice that the value of the uniform distribution is equal to one since we are working here with potential function and not probability distributions. The only value that does not affect a product, which is what we want for an uninformative relation, is then one. The only question left is "How to estimate the probability that the observed relation is rigid?"

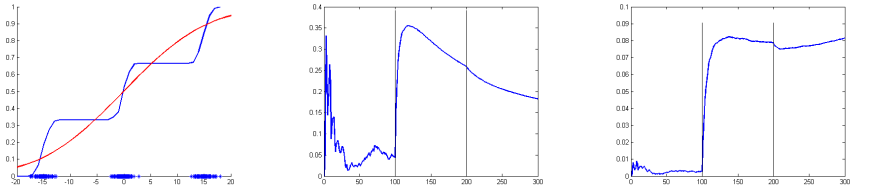
The Probability of the Parametric Model

To estimate the fidelity λ_t, we introduce a method inspired from the Kolmogorov-Smirnov test. Recall that the Kolmogorov-Smirnov distance is given by

$$K_n = \sqrt{n} \max_{-\infty<x<\infty} \left| \hat{F}(x) - F_n(x) \right|, \tag{4.2.4}$$

where n is the number of observations, $F_n(x)$ is the empirical cumulative distribution function of the n observations, and $\hat{F}(x)$ is the cumulative distribution of the maximum-likelihood Gaussian model. This distance is compared to a threshold, say, to classify a dataset as Gaussian or non-Gaussian.

Our context is different: We are not interested in precise Gaussianity (i.e. level of correspondence between the distribution of the data and a Gaussian distribution) but in relations that are about as *predictive* as Gaussians. Therefore, the original Kolmogorov-Smirnov test is unsuitable in that its sensitivity



(a) Cumulative distribution of the samples (in blue) and of the Gaussian model (in red).

(b) $D = \max \left| \hat{F}(x) - F_n(x) \right|$

(c) $D = \frac{1}{|I|} \int_I \left| \hat{F}(x) - F_n(x) \right| \mathrm{d}x$

Figure 4.2.5: Comparison between the use of the max (Figure 4.2.5b) and the integral (Figure 4.2.5c) for the computation of the distance from Gaussianity of the dataset. Samples are generated from a sequence of three Gaussians with 100 samples for each Gaussians.

grows without bounds with n. Instead of Equation 4.2.4, we use an expression independent of the number of observations,

$$D = \frac{1}{|I|} \int_I \left| \hat{F}(x) - F_n(x) \right| \mathrm{d}x, \tag{4.2.5}$$

where I is the interval within which the two functions are compared. Notice that we use an integral instead of the maximum as it renders the measure both more robust and more discriminative in our context. Computing the integral of a cumulative function may seems surprising but can be understood with the example in Figure 4.2.5. In this example, we generate samples from a sequence of three Gaussians with 100 samples for each Gaussian. We can see that, once we start to generate samples from the second Gaussian, the distance from Gaussianity increases in a significant manner for both the max function and the integral (although the increase is not as clear for the max function). With the number of samples from the second distribution increasing, the mean of the Gaussian model starts to shift toward a central position making the cumulative function of the model move "inside" the stairs-shaped cumulative distribution of the samples (see Figure 4.2.5a). While this has nearly no influence on the integral of the absolute difference between the two distributions, it clearly reduces the maximum distance between them. If we add samples from a third Gaussian, the situation becomes even worse because the steps of the stairs-shaped cumulative distribution become even smaller. In practise, this situation happens quite often because we compute the distance D for each dimension separately (for computational cost reasons). Any articulated relation will then appears mainly as a mixture of two Gaussians and one mostly uniform distribution when projected on one of the dimensions.

The model fidelity λ_t will then result in the product of its Gaussianity in each dimension. In order to compute λ_t based on the distance D, we assume that D

has a Gaussian distribution, which leads to the pseudo-probabilistic weighting function

$$\lambda_t = \prod_{k=1}^{n_d} \mathrm{e}^{\frac{-D_k^2}{T_{k,D}^2}}, \tag{4.2.6}$$

where n_d is the number of dimensions of the relation space and T_D is a user-settable parameter that represents the allowed deviation of observed relations from Gaussianity.

Notice that the cumulative density function of the observations $F_n(x)$ seems to be a simpler choice to model the relation than the parametric model. You may, then, wonder why we do not use it since it is able to represent any king of distribution. The problem comes from the fact that $F_n(x)$ is only calculated on a limited interval I (and in practise, even worse, on a limited discrete interval). Since the uncertainty on the model implies that the potential function can be greater than zero in an infinite interval, $F_n(x)$ cannot be used without creating bias in the tracking. On the other side, it is enough to compare $F_n(x)$ and $\hat{F}(x)$ to a limited interval to get a good approximation of the correspondence between them.

4.2.1.3 Uncertainty in the Correlation Between Past and Future

In our scenario, the relations are learnt incrementally, which sometimes leads to situations where an observation at the current time t is in fact not well predicted by the relations learnt up to time $t-1$. This may happen if, for example, the two features connected through a relation were motionless until time $t-1$, and one of them starts to move at time t. In this case, it is clear that the rigid relation learnt from previous frames is no longer appropriate to track these features. The problem with this case is that the rigid relation will constrain the relative positions of the two corresponding nodes and therefore produce tracking results that continues to reinforce the belief of a rigid relation. This type of uncertainty is different from the two others because it cannot be estimated from the previous samples. With no guarantee that the past and future data are correlated, it might seems that the learnt model is actually useless. Fortunately, a given relation is only used for tracking its corresponding nodes in the next frame. Given that the observations in a video are spatially correlated in time, the next observation will not be too far from the current model. If we assume that the (application-dependent and fixed) likelihood of making an observation a distance Δ away from the learnt model follows a Gaussian distribution of variance $\mathbf{C}_\Delta$, a suitably augmented potential $\psi^-_{i,j,t}$ can be obtained from the learnt model $\psi^+_{i,j,t-1}$ defined in Equation 4.2.3,

$$\psi^-_{i,j,t} = \psi^+_{i,j,t-1} \circledast N(0, \mathbf{C}_\Delta)\,. \tag{4.2.7}$$

Putting together Equations 4.2.32 and 4.2.3, we obtain the complete *uncertain* potential function used for the feature tracking,

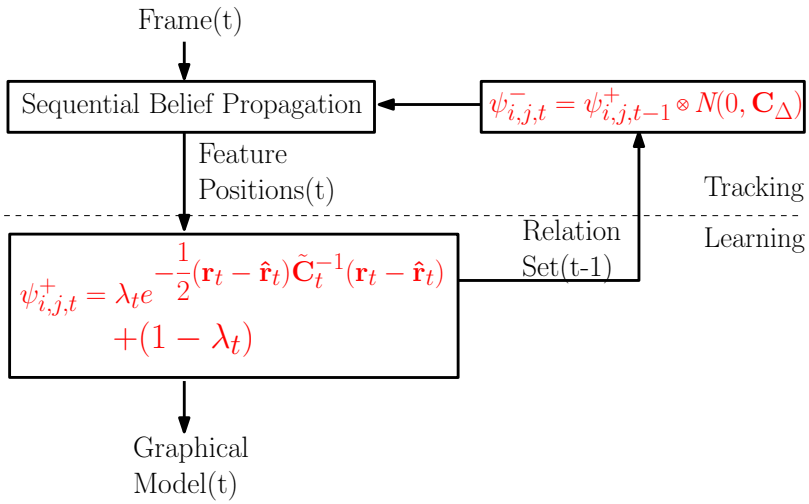


Figure 4.2.6: The whole algorithm loop for tracking using *uncertain* Gaussian model to represent the spatial relations between nodes.

$$\begin{aligned} \psi^{-}_{i,j,t} &= \lambda_{t-1} e^{-\frac{1}{2}(\mathbf{r}_{t-1} - \hat{\mathbf{r}}_{t-1}) \left(\tilde{\mathbf{C}}_{t-1} + \mathbf{C}_{\Delta}\right)^{-1} (\mathbf{r}_{t-1} - \hat{\mathbf{r}}_{t-1})} \\ &\quad +(1 - \lambda_{t-1})\,. \end{aligned} \tag{4.2.8}$$

Figure 4.2.6 shows a diagram of the algorithm loop used for tracking a graph using an *uncertain* Gaussian model to represent the spatial relations between nodes.

4.2.2 Uncertain Flexible Relation Learning

As we discussed in Section 4.1.1.3, learning and refining a mixture model incrementally is not an easy task. How is a given model to be updated when new data points arrive? If the data points underlying the current model have been discarded, then there seems to be no general answer to this question. Conversely, keeping all data around defeats the purpose of learning parametric models incrementally. Thus, a compromise needs to be found. The approach proposed by Arandjelovic and Cipolla [2] seems to provide a good step in that direction but the historical model as it was defined was insufficient to provide a robust general solution. What we need is to keep around enough information to be able to accurately refine a model with the new data points. Conversely, the quantity of this information should grow much more slowly than the number of raw data points. We address this problem by seeking to represent the data points with (1) sufficient fidelity that we can safely discard them, while at the same time (2) committing to no more predictive precision as the original data support.

These two objectives are mutually exclusive, as the former tends to over-fit the data and the latter to under-fit it. We therefore propose a two-level representation. The first level seeks to summarise the data with high precision, allowing us to discard underlying data without significantly impairing our ability to refine the model. We therefore call it the *precise* model. The second level provides a model that represents no more detail than is supported by the underlying data and thereby avoids counterproductive bias in future predictions; we call it the *uncertain* model. Each *uncertain* component is then represented by a precise mixture model that allows it to be split properly when it becomes obvious that it provides an excessive simplification of the underlying data.

4.2.2.1 Level 1: The Precise Mixture Model

Update of the Gaussian Mixture Model

Suppose we have already learnt a precise GMM from the observations up to time t. This GMM is currently defined by

$$\psi^+_{i,j,t} = \frac{\sum_{k=1}^{N} \pi_{k,t} G(\mathbf{r}; \hat{\mathbf{r}}_{k,t}, \tilde{\mathbf{C}}_{k,t})}{\sum_{k=1}^{N} \pi_{k,t}}, \tag{4.2.9}$$

where each Gaussian is represented by its weight $\pi_{k,t}$, its mean $\hat{\mathbf{r}}_{k,t}$, and its covariance $\tilde{\mathbf{C}}_{k,t}$. When we receive a new data point $\mathbf{r}_t$, we represent it by the distribution $G_t(\mathbf{r}; \mathbf{r}_t, \mathbf{C}_t)$ and its weight π_t. $\mathbf{C}_t$ here represents the observation noise. As suggested by Hall and Hicks [38], the new resulting GMM is computed in two steps:

1. **Concatenate** – produce a model with $N + 1$ components by trivially combining the GMM and the new data into a single model.
2. **Simplify** – if possible, merge some of the Gaussians to reduce the complexity of the GMM.

The GMM resulting from the first step is simply

$$\psi^+_{i,j,t} = \frac{\sum_{i=1}^{N} \pi_{k,t-1} G(\mathbf{r}; \hat{\mathbf{r}}_{k,t-1}, \tilde{\mathbf{C}}_{k,t-1}) + \pi_t G_t(\mathbf{r}; \mathbf{r}_t, \mathbf{C}_t)}{\sum_{i=1}^{N} \pi_{k,t-1} + \pi_t} \tag{4.2.10}$$

The goal of the second step is to reduce the complexity of the model while still giving a precise description of the observations. Since the solution proposed by Hall and Hicks [38] is too slow for an real-time process, we use the fidelity estimator λ_t we developed in Section 4.2.1.2 for a single Gaussian model.

Simplification of the Gaussian Mixture Model

To decide whether two Gaussians G_i and G_j can be simplified into one, we merge them together and check if the resulting Gaussian has a fidelity λ close to one, say, exceeding a given threshold $\lambda^+_{\min} = 0.95$. The resulting Gaussian

is computed using the usual equations augmented by the combination of the cumulative density functions:

$$\pi = \pi_i + \pi_j \tag{4.2.11}$$

$$\hat{\mathbf{r}} = \frac{1}{\pi}[\pi_i \hat{\mathbf{r}}_i + \pi_j \hat{\mathbf{r}}_j] \tag{4.2.12}$$

$$\begin{aligned} C = & \frac{\pi_i}{\pi}\left[C_i + (\hat{\mathbf{r}}_i - \hat{\mathbf{r}})^T(\hat{\mathbf{r}}_i - \hat{\mathbf{r}})\right] \\ & + \frac{\pi_j}{\pi}\left[C_j + (\hat{\mathbf{r}}_j - \hat{\mathbf{r}})^T(\hat{\mathbf{r}}_j - \hat{\mathbf{r}})\right] \end{aligned} \tag{4.2.13}$$

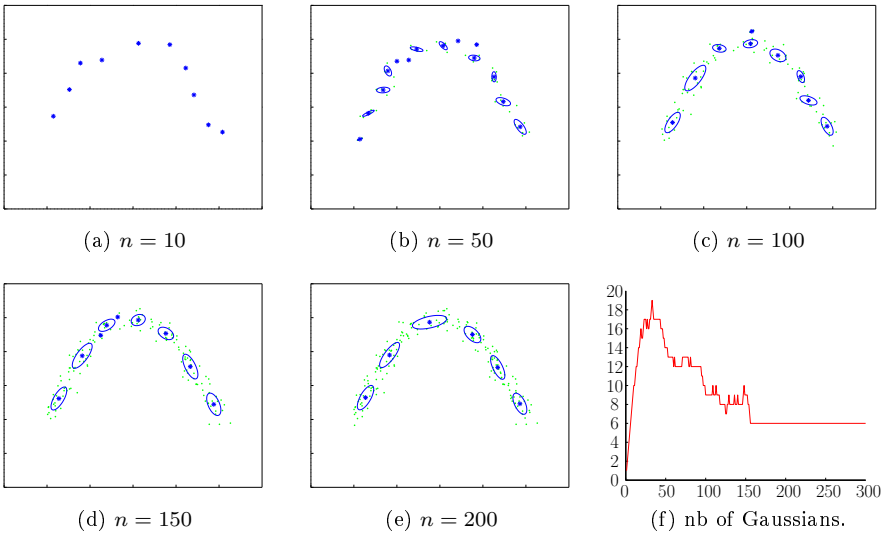
$$F(x) = \frac{1}{\pi}\left[\pi_i F_i(x) + \pi_j F_j(x)\right] . \tag{4.2.14}$$

At each time, if the current GMM before the concatenation is already the simplest possible precise model of the data, the only Gaussian that has a chance to be merged with another is the one representing the new data point. If this Gaussian is successfully merged, the resulting Gaussian is, in turn, the only available candidate for a simplification. The merging then continues iteratively until the best candidate merge drops below λ^+_{min}. This algorithm is very fast since it corresponds, on average, to two nested loops containing only one nearest neighbour search and one merge, respectively. We only try to merge the new Gaussian with its nearest neighbour since this is most likely to provide a precise simplification. While this approach is simplistic, it gives very good results in practise while inducing only a low computational cost.

Illustrations

Figure 4.2.7 shows a first example of the evolution of the GMM for an increasing number of data points generated from an arc-shaped distribution. At the beginning, none of the Gaussians can be merged since there is clearly no Gaussian distribution that can summarise more than one observation without a significant loss of information. The complexity of the mixture thus increases by one with each new data point. As the shape of the distribution appears more clearly, the simplification step takes effect, and the number of Gaussians in the mixture decreases until it converges to a trade-off between complexity of the mixture and its accuracy. This trade-off is controlled with the parameter T_D defined in Equation 4.2.6. The bigger this parameter is, the farther the model is allowed to deviate from the data and the lower the complexity of the model will be. This dependence will be analysed in more detail in Section 4.3.

Let us now consider the evolution of the model for data generated from a Gaussian distribution with a large covariance. Figure 4.2.8 shows an example of the evolution of the GMM for an increasing number of data points generated from a Gaussian distribution. Figure 4.2.8f shows the mean evolution of the number of Gaussians in the mixture for a series of 50 tests. As one would expect, we first observe an explosion of the complexity of the model before it converges to a single Gaussian. This shows that the effort to faithfully represent the observations leads to gross over-fitting of sparse data. Thus, our method is useful to *summarise* past observations but not to *predict* future observations.



(a) $n = 10$ (b) $n = 50$ (c) $n = 100$

(d) $n = 150$ (e) $n = 200$ (f) nb of Gaussians.

Figure 4.2.7: Evolution of the precise mixture model for an increasing number of data points drawn from an arc-shaped distribution.

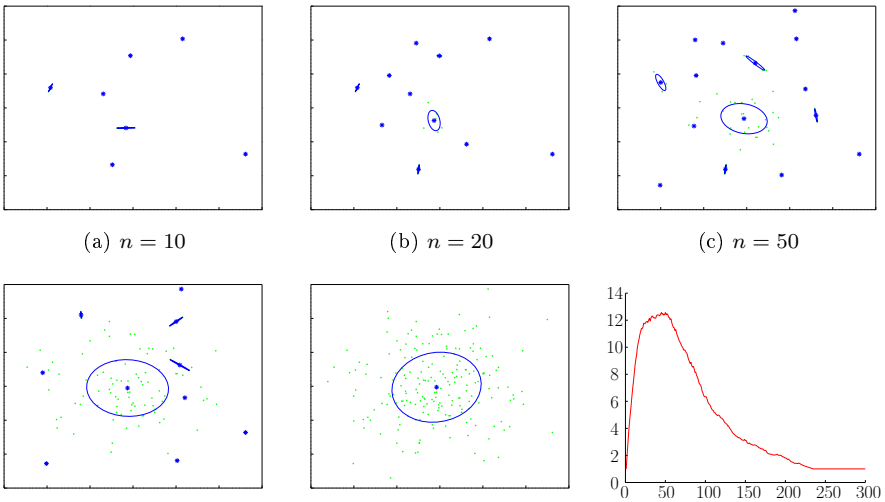
To address prediction, we propose, in the following section, a 2-level mixture model containing one level for a precise summary of the data and one for a non-over-fitted representation of the data.

4.2.2.2 Level 2: The Uncertain Mixture Model

Let us consider again the case of a GMM learnt from Gaussian-distributed data. What should be the value of the parameter T_D to guarantee that the model will always be a one-Gaussian mixture with a fidelity λ exceeding $\lambda^+_{\min}$? To answer this question we have computed the distribution of the distance D (Equation 4.2.5) for a number of observations between 1 and 200 estimated over 1000 tests. Figure 4.2.9 shows the results we obtained. As expected, the variance of the distance D is very large when the number of observations is low. We then have to choose T_D such that the probability of incorrectly splitting the Gaussian is bounded by a constant α. Since T_D is similar to a standard deviation (Equation 4.2.6), and since empirical estimates of variance follow a chi-square distribution, we can limit this probability to, say, $\alpha = 0.005$, by replacing T_D by an adjusted $\tilde{T}_D$ defined as

$$\tilde{T}_D^2 = \frac{N}{\chi^2_{N-1}(\alpha)} T_D^2 \,, \tag{4.2.15}$$

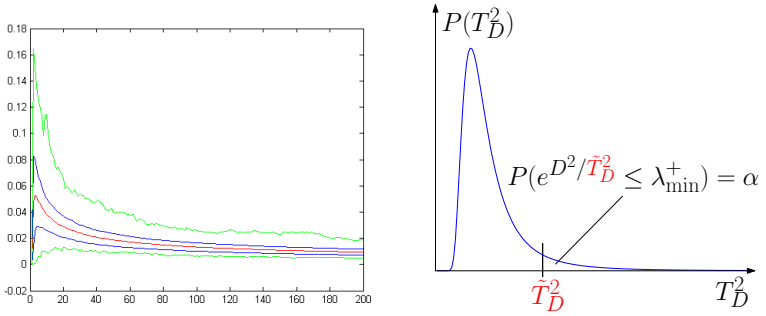
where $\chi^2_{N-1}(\alpha)$ is the chi-square inverse cumulative distribution function. The new fidelity criterion is then defined by



(a) $n = 10$ (b) $n = 20$ (c) $n = 50$

(d) $n = 100$ (e) $n = 250$ (f) nb of Gaussians.

Figure 4.2.8: Evolution of the precise mixture model for an increasing number of data points drawn from a Gaussian distribution.



(a) (b)

Figure 4.2.9: (a) Distribution of D as a function of the number of observations estimated over 1000 trials. The red line represents the mean, the blue the standard deviation, and the green lines the extrema of the samples. (b) We then choose T_D such that the risk of incorrectly splitting the Gaussian is bounded by α.

$$\exp\left(\frac{-D^2}{\tilde{T}_D^2}\right) \geq \lambda_{\min}^{+} . \tag{4.2.16}$$

In order to use the same fidelity λ_t from Equation 4.2.6 in both levels, we need to define a new fidelity threshold $\lambda_{\min}^{-} < \lambda_{\min}^{+}$. This way, $\lambda_{\min}^{+}$ is used in the precise GMM and $\lambda_{\min}^{-}$ is used in the *uncertain* GMM. Substituting Equation 4.2.15 yields

$$\frac{-D^2\chi_{N-1}^2(\alpha)}{NT_D^2} \geq \log\lambda_{\min}^{+} , \tag{4.2.17}$$

$$exp\left(\frac{-D^2}{T_D^2}\right) \geq \exp\left(\frac{N\log\lambda_{\min}^{+}}{\chi_{N-1}^2(\alpha)}\right) . \tag{4.2.18}$$

The complexity of the *uncertain* GMM is then controlled by a lower threshold on the fidelity

$$\lambda_{\min}^{-} = \exp\left(\frac{N\log\lambda_{\min}^{+}}{\chi_{N-1}^2(\alpha)}\right) , \tag{4.2.19}$$

which can be precomputed in a table since it only depends on $\lambda_{\min}^{+}$. Thanks to this new threshold, we are able to avoid the over-fitting due to an explosion of the GMM complexity.

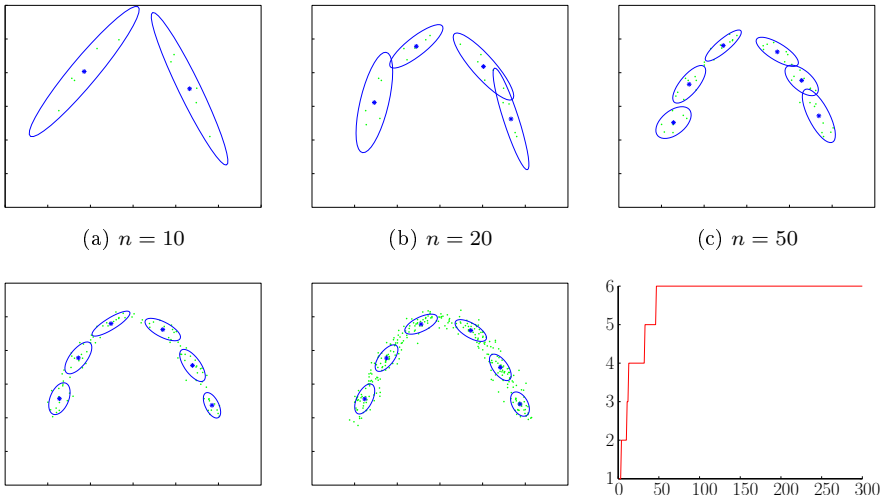
However, even if we have reduced the complexity of the model, we still face the problem of over-fitting through the Gaussian model itself. Indeed, the Gaussian learnt from a dataset corresponds to the maximum likelihood estimate of these data and not of the complete distribution: Consider, for example, the case of a Gaussian learnt from a single observation: it is clear that this Gaussian is not representative of the complete distribution. To solve that problem, we use the *uncertain* Gaussian model developed in Section 4.2.1 for each Gaussian of the *uncertain* mixture model.

Updating a Two-Level Gaussian Mixture Model

The algorithm used to update the GMM proceeds along the following steps:

1. Merge the new data point with the nearest *uncertain* Gaussian,
2. **if** the resulting Gaussian has a value of λ below the corresponding $\lambda_{\min}^{-}$, replace it with two Gaussians learnt from its underlying GMM with EM [27],
3. **else** continue to merge the current *uncertain* Gaussian with its nearest neighbour until the resulting Gaussian has a value of λ lower than the corresponding $\lambda_{\min}^{-}$.

Merging two *uncertain* Gaussians also involves merging their respective underlying mixture models. This can be done by simply summing the components from both mixtures, and using the simplification step only on the precise Gaussian that contains the new observation. Even if other precise Gaussians could



(a) $n = 10$ (b) $n = 20$ (c) $n = 50$

(d) $n = 100$ (e) $n = 300$ (f) nb of Gaussians.

Figure 4.2.10: Evolution of the *uncertain* mixture model with the number of data points drawn from an arc-shaped distribution (compare Figure 4.2.7).

possibly be merged together, we leave this for later when these Gaussians merge with the current observation. This way, we distribute the computational cost through different time instants.

Illustrations

Figure 4.2.10 shows an example of the evolution of the GMM with data points generated from an arc-shaped distribution. This time, the complexity of the GMM only increases when there is enough evidence that the observed distribution is too complex for the current model. If we compare Figure 4.2.10 with Figure 4.2.7, we see that the two-level GMM and the precise mixture model converge to the same distribution. The two-level approach then provides a more stable non-over-fitted model that can still become more accurate thanks to the precise model level.

4.2.3 Uncertain Articulated Relation Learning

All the observed relations are not necessarily informative enough to be worth keeping. Since we decided to limit the relation set to rigid and articulated relations, a model as complex as the Gaussian mixture might be replaced with something simpler (and therefore easier to learn). Thanks to the concept of *uncertain* model, a specific parametric model can be used on all the relations

without creating counterproductive bias on tracking if a relation does not correspond to the chosen model.

4.2.3.1 Maximum Likelihood Joint Position

An articulated relation between two nodes can be described by the position of its joint. This position is obtained by finding a location in the reference frame of one node that is always projected into the same position in the reference frame of the other node. Let us call $\mathbf{b}_{i|j,t}$ the affine position of a node i expressed in the reference frame of a node j. If there is a joint between these two nodes, it can be found by solving [51]

$$\mathbf{j}_{ij|i,t} = \arg\min_{\mathbf{r}_{|i}} \frac{1}{t-t_0} \sum_{k=t_1}^{t} \left(T_{i,j,k}(\mathbf{r}_{|i}, \mathbf{b}_{i|j,k}) - T_{i,j,k}(\mathbf{r}_{|i}, \mathbf{b}_{i|j,t_0}) \right)^2 , \quad (4.2.20)$$

where $T_{i,j,k}(\mathbf{r}_{|i}, \mathbf{b}_{i|j,k}) = A_{i,j,k}(\mathbf{b}_{i|j,k})\mathbf{r}_{|i} + t_{i,j,k}(\mathbf{b}_{i|j,k})$ is the affine transformation that projects a position $\mathbf{r}_{|i}$ from the reference frame of i to a location in the reference frame of j. The sum achieves its minimum at

$$\mathbf{j}_{ij|i,t} = -\left(\sum_{k=1}^{t} (\Delta A_{i,j,k})^T \Delta A_{i,j,k} \right)^{-1} \sum_{k=t_1}^{t} (\Delta A_{i,j,k})^T \Delta t_{i,j,k} , \quad (4.2.21)$$

where $\Delta A_{i,j,k} = A_{i,j,k} - A_{i,j,t_0}$ and $\Delta t_{i,j,k} = t_{i,j,k} - t_{i,j,t_0}$.

In order to learn $\mathbf{j}_{ij|i,t}$ sequentially with each new frame, its computation can be adapted in the following way:

$$\begin{aligned}
H_{i,j,t} &= H_{i,j,t-1} + w_{i,j,t} (\Delta A_{i,j,t})^T \Delta A_{i,j,t} && (4.2.22)\\
S_{i,j,t} &= S_{i,j,t-1} + w_{i,j,t} (\Delta A_{i,j,t})^T \Delta t_{i,j,t} && (4.2.23)\\
V_{i,j,t} &= V_{i,j,t-1} + w_{i,j,t} (\Delta t_{i,j,t})^T \Delta t_{i,j,t} && (4.2.24)\\
W_{i,j,t} &= W_{i,j,t-1} + w_{i,j,t} && (4.2.25)\\
\mathbf{j}_{ij|i,t} &= -H_{i,j,t}^{-1} S_{i,j,t} , && (4.2.26)
\end{aligned}$$

where we have included a weight factor $w_{i,j,t} = w_{i,t}^p w_{j,t}^q$ corresponding to the product of the weight of the two best particles to account for the reliability of the observations.

4.2.3.2 Uncertain Joint Model

With the learnt joint positions $\mathbf{j}_{ij|i,t}$ and $\mathbf{j}_{ji|j,t}$, we are now able to compute the distance $\mathbf{d}_{i|j,t}^2 = \left\| \mathbf{j}_{ij|j,t} - \mathbf{j}_{ji|j,t} \right\|^2$ between the two predictions of the joint position into the reference frame of node j. In order to obtain the complete *uncertain* potential function from Section 2.2.3.2, i.e.

$$\psi^{+}_{i,j,t}(\mathbf{d}^2_{i|j,t}) = \lambda_{i|j,t} e^{-\mathbf{d}^2_{i|j,t}/\tilde{\sigma}^2_{i|j,t}} + (1 - \lambda_{i|j,t}) \,, \tag{4.2.27}$$

we still need to learn $\tilde{\sigma}^2_{i|j,t}$ and $\lambda_{i|j,t}$.

Considering $\mathbf{j}_{ij|j,t}$ is normally distributed around $\mathbf{j}_{ji|j,t}$, the maximum likelihood variance $\sigma^2_{i|j,t}$ is easily obtained by summing the square distance $\mathbf{d}^2_{i|j,t}$ between the projected positions $\mathbf{j}_{ij|j,t}$ and $\mathbf{j}_{ji|j,t}$ obtained at each time t,

$$\sigma^2_{i|j,t} = \sum_{k=t_0}^{t} \mathbf{d}^2_{i|j,k} \,. \tag{4.2.28}$$

$\tilde{\sigma}^2_{i|j,t}$ is then obtained from $\sigma^2_{i|j,t}$ using the exact same equation as in Section 4.2.1.1,

$$\tilde{\sigma}_{i|j,t} = \frac{\pi_t}{\chi^2_{\pi_t - 1}(\alpha)} \sigma_{i|j,t} \,, \tag{4.2.29}$$

where $\chi^2_{\pi_t - 1}(\alpha)$ is the chi-square inverse cumulative distribution function and $\alpha = 0.05$.

The model fidelity λ_t is also computed in a manner very similar to that of Section 4.2.1.2:

$$\lambda_t = \mathrm{e}^{\frac{-D^2}{T_D^2}} \,, \tag{4.2.30}$$

where T_D is a user-settable parameter that represents the allowed deviation from Gaussianity and D is the distance between the observed and expected cumulative distributions,

$$D = \frac{1}{|I|} \int_I \left| \hat{F}(x) - F_n(x) \right| \mathrm{d}x \,. \tag{4.2.31}$$

The only difference is that, this time, x corresponds to $\mathbf{d}^2_{i|j}$. Since distances based on a normally distributed dataset have an exact chi-square distribution the expected cumulative distributions $\hat{F}(x)$ is now the cumulative chi-square distribution of $(\mathbf{d}^2_{i|j}/\sigma^2_{i|j,t})$.

When this relation is used for tracking, the articulated potential model is again treated identically to the rigid model using

$$\psi^{-}_{i,j,t} = \psi^{+}_{i,j,t-1} \circledast N(0, \sigma^2_{\Delta}) \,, \tag{4.2.32}$$

where σ^2_{Δ} is the variance of the expected motion model.

4.2.4 Rigid Block Discovery

The real-time on-line motion segmentation method presented in Section 4.1.2.2 is a good basis for us to develop a solution for a rigid block discovery specific to our graphical model. As we have seen, the method can be divided into two parts: *Grouping procedure* and *Maintenance of Feature Groups Over Time.* We

will therefore organise this section in the same way, starting with the grouping procedure.

4.2.4.1 Grouping Procedure

In order to group features, any algorithm requires a measure of the distance or affinity between these features. In our case, this affinity matrix can be computed using different approaches:

- **Approach 1**: Through its learnt rigid relation with its corresponding block, a feature is able to predict at any time the affine position of its block. Features that belong to a same rigid group are supposed to give similar predictions. Affinity between the features of a given block can then be estimated using the distances between these predictions,

$$A_{i,j,t} = \sum_{k=1}^{t} \exp\left(-\frac{1}{2} D^2_{i,j,k}\right) \tag{4.2.33}$$

$$\text{with} \quad D^2_{i,j,k} = \left(\mathbf{b}_{\mathcal{B}(i)|i,k} - \mathbf{b}_{\mathcal{B}(j)|j,k}\right) \mathbf{C}_{\mathrm{T}}^{-1} \left(\mathbf{b}_{\mathcal{B}(i)|i,k} - \mathbf{b}_{\mathcal{B}(j)|j,k}\right) , \tag{4.2.34}$$

  where $A_{i,j,t}$ is the affinity between features i and j at time t, $D^2_{i,j,k}$ is the normalised squared distance at time k between the predictions of the block position. $\mathbf{C}_{\mathrm{T}}$ is a covariance matrix that defines the tolerance on the dissimilarities of the predictions.

- **Approach 2**: We can also consider that the affinity matrix is not necessary symmetric and that $A_{i,j,t}$ represents the ability of feature i to predict properly the position of feature j. In this case the affinity matrix can be used in a procedure like the one presented in Section 4.1.2.1 to select the features able to predict the higher number of points. By using the affine position of a block as predicted by a selected feature, we can project the positions of all the other features back into the image. Through the connection with the block, a feature is then able to predict the position of any other feature of the block. The affinity between two features i and j can then be evaluated using the normalised distance $D^2_{i,j,k}$ between the actual position of j and its prediction by i ,

$$D^2_{i,j,k} = \left(\mathbf{f}_{i,k} - \mathbf{f}_{i|j,k}\right) \mathbf{C}_{\mathrm{T}}^{-1} \left(\mathbf{f}_{i,k} - \mathbf{f}_{i|j,k}\right) . \tag{4.2.35}$$

- **Approach 3**: Instead of computing the distance between projection and actual position, we can also use the oriented chamfer distance of the projection in the current image. This approach has the advantage to provide a good estimation of the affinity even if the projected feature has not been tracked properly. This is especially important in the case of edgels because, the feature can slide along a contour during its tracking. This can create a distance between the tracked position and the projection even if

Algorithm 4.1 Affinity Propagation

1. **Initialisation of Affinity Propagation**

   (a) Initialise the Affinity matrix with $A^1_{i,j,t} = 0$ for each pair $\{i, j\}$.

   (b) For each node i, initialise its self-affinity with $A^1_{i,i,t} = 1$.

   (c) For each node i, send a first message $m^l_{i|j} = w_{i|j} \cdot A_{i,:,t}$ to each node $k \in \mathcal{N}(i)$ where $A_{i,:,t}$ represent the ith row of matrix A_t.

2. **Iteration l of Affine Warp Propagation**

   (a) For each node i, update every kth element of the ith row of the affinity matrix with

   $$A^l_{i,k,t} = \max\left(A^{l-1}_{i,k,t}, \max_{j \in \mathcal{N}(i)}\left(w_{j|i} \cdot A^{l-1}_{j,k,t}\right)\right). \tag{4.2.37}$$

   (b) Stop if $\max_i\left(w^l_{i,t} - w^{l-1}_{i,t}\right) < \text{thres}$ where $w_{i,t} = \sum_k A_{i,k,t}$ corresponds to the ability of a node to provide a large rigid cluster.

the projection is correctly placed on a contour. The distance $D_{i,j,k}$ is here defined using the oriented chamfer function presented in Section 2.1.2.2,

$$D_{i,j,k} = d_\lambda(\mathbf{f}_{i|j,k}). \tag{4.2.36}$$

- **Approach 4**: Since some relations between features have been learnt into the feature level, the information necessary to group the features is already available. Only a small set of the N^2 relations between the N features has been learnt but the simple propagation scheme presented in Algorithm 4.1 allows the affinity matrix A_t to be recovered. Basically, the affinity $A_{i,j,t}$ between two features i and j corresponds to the product of weights of the *uncertain* relations included in the path between i and j that provided the maximal affinity. So the affinity between two features will be high if they can connect through a path of rigid relations and low if not. The advantage of this approach is that it benefits from the expertise of the learnt *uncertain* relations in order to provide a more robust clustering. Note that "*Affinity Propagation*" is also the name of a clustering algorithm from Frey and Dueck [34]. Our algorithm has for only purpose to recover the full affinity matrix based on the learnt relations of the graph and is therefore not related to their work.

Once the affinity matrix is obtained from one of those approaches, the clusters can be obtained easily using an algorithm similar to these presented in Section 4.1.2:

1. Compute the weights $w_{i,t} = \sum_k A_{i,k,t}$ of each feature i in order to represent its ability to provide a large rigid cluster.

(a) (b) (c)

Figure 4.2.11: Initialisation of the feature tracking using *Affine Warp Propagation*. The relations in red are given a weight of 0.1 and the relations in green are given a weight of 0.99. These relations are used only for generating the feature tracks and not for the Affinity Propagation where relation are learnt directly from the tracking results.

2. Select the feature i that has the largest weight $w_{i,t}$ and create a new group with all the points k that verify $A_{i,k,t} \geq \mathrm{A}_{\mathrm{thres}}$.

3. Remove those inliers and repeat the two steps above until no group can be formed with at least $N_{\mathrm{minInliers}}$ inliers.

4. Refine the clustering by associating each point k with the group with the highest affinity towards k. The affinity vector of a group is defined by the row $A_{i,:,t}$ of the affinity matrix corresponding to the point i used to create it.

The four affinity matrix computation methods combined with the clustering described above are tested in Figures 4.2.12, 4.2.13, 4.2.14 for respectively an arm, a finger, and a hand. In order to provide the proper features tracking results, we manually define the weights of the relations in the graph (see Figure 4.2.11) and track the feature level using *Affine Warp Propagation*. From these tracking results, we then learn *uncertain* relations, compute the four different affinity matrices and cluster the features into rigid groups. With this approach, we base the learning on tracking results unbiased by the learnt graph (this case will be discussed in the next chapter) but with a level of noise closer to reality than artificially generated tracking results. All the experiments have been performed using the same parameters. We choose $\mathrm{A}_{\mathrm{thres}} = 0.6$ ($\simeq exp(-0.5)$) and $N_{\mathrm{minInliers}} = 7$ ($1 + 6$ affine parameters). From the results shown in Figures 4.2.12, 4.2.13, 4.2.14, we can see that the affinity matrix obtained using the learnt *uncertain* relations and *Affinity Propagation* (approach 4) provides a more robust clustering than the three other approaches. This result can be explained by different reasons:

- The block position obtained from the prediction of a feature is very sensitive to the noise in the affine position of this feature. Indeed, a small change in the orientation or scale, for example, in the position of a feature

can have a important impact on the predictions of the block position if the block centre is far from the feature location.

- In many articulated objects (such as an arm or a finger), the parts modelled by rigid blocks may not be entirely rigid. Consider, for example, the flesh on the middle segment of a finger. The flesh is compressed when the finger is bent and stretched when the finger is extended. This situation is particularly important in the case of edgels where we assist to compression or dilatation of the features along the contours around the joints. With edgels, this effect tends to diffuse deeper into the rigid parts than with other features due to their tendency to slide along the contours. This creates noise into the predictions of the block positions.

- The use of the oriented chamfer distance (approach 3) is able to reduce the effects above (specially the second) since it only verifies whether a feature is able to project others on the proper edges. Unfortunately the effect of clutter largely balance that benefit.

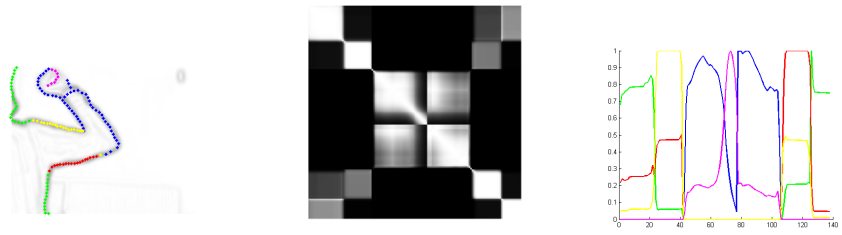
- *Uncertain* relations are learnt only between close features. The effect of the noise on their projection into the reference frame of each other is then far more limited than for the other approaches.

On top of that, *uncertain* relations have been designed to provide a level of information related to the reliability of the learnt model. They are therefore perfect candidates to use for segmentation in a simultaneous learning and tracking approach. This being said, the computational cost of the affinity matrix is $O(KN^2)$, where K is the size of the longest path maximising the affinity inside a rigid group (since the number of affinity propagation iterations is limited by K). A second step is also required to group clusters demonstrating a rigid relation but not connected by a rigid path in the graph. We will see in the next section how these two problems can be solved by maintaining the groups over time.

4.2.4.2 Maintenance of Feature Groups Over Time

Since the segmentation is based on the *uncertain* relations learnt from all the observations from the first frame to the current one, the grouping procedure demonstrates a really good stability from one frame to the next. Maintaining groups over time is then here more a matter of computational time than stability. We can see from the previous section that the only rows of the affinity matrix A_t that really matter are the rows associated with the features used to create the groups. By maintaining a small set of M "leader" features able to represent the different rigid groups, we can then limit the affinity matrix to a matrix of $M \times N$ elements instead of the N^2 elements used in the previous section.

Moreover, once these leaders are known, we can create connections between them into the feature level. These M^2 *uncertain* relations between them are not only creating bridges between unconnected parts of the graph that could be correlated but also represent the affinity between the groups.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Figure 4.2.12: Example of direct segmentation of an arm using the four different affinity matrices presented in Section 4.2.4.1. The four rows of images correspond to the four approaches (in the order of their number). The first column shows the clustering results, the second column represents the affinity matrix, and the third column shows the lines of the affinity matrix corresponding to the features selected as group leaders.

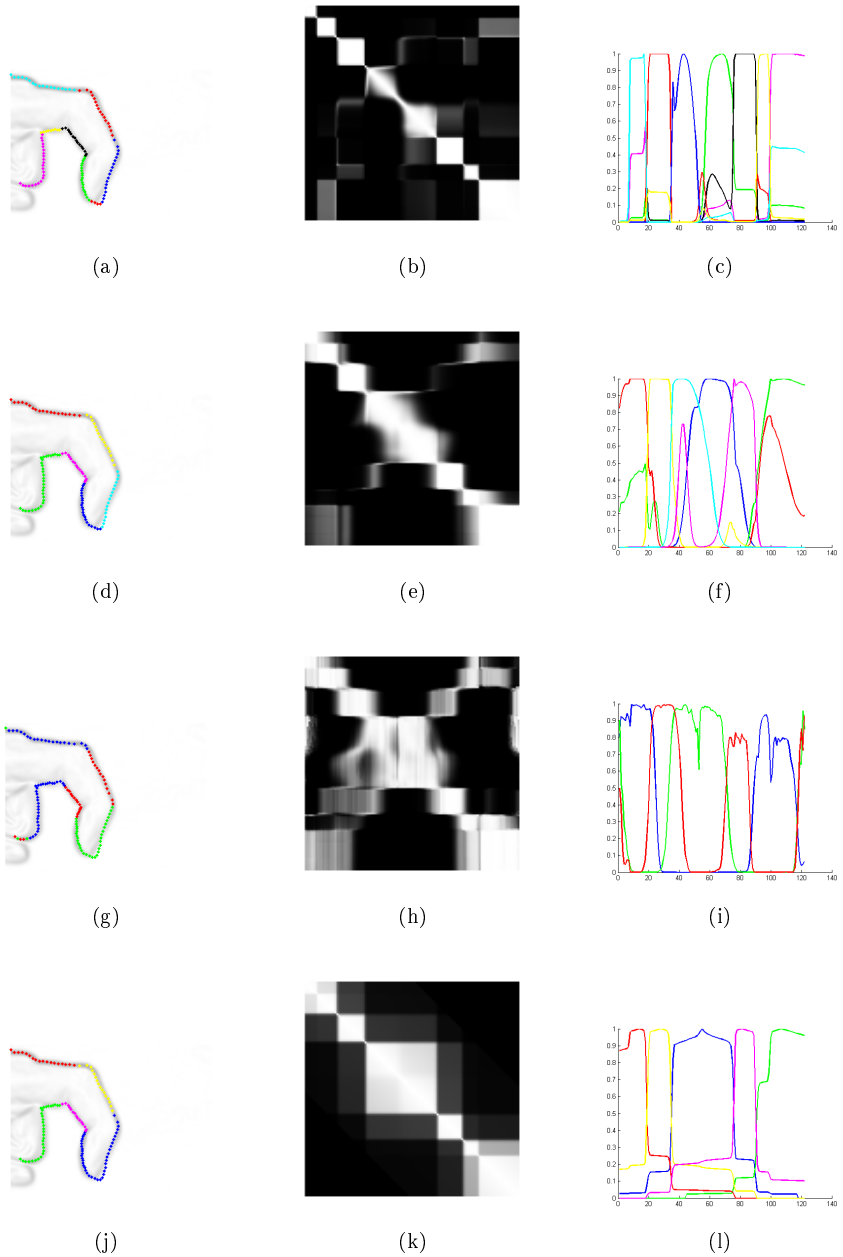
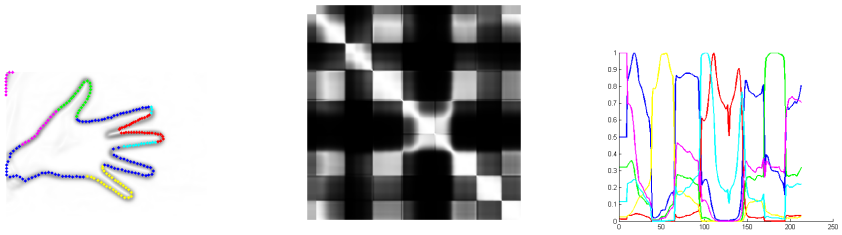
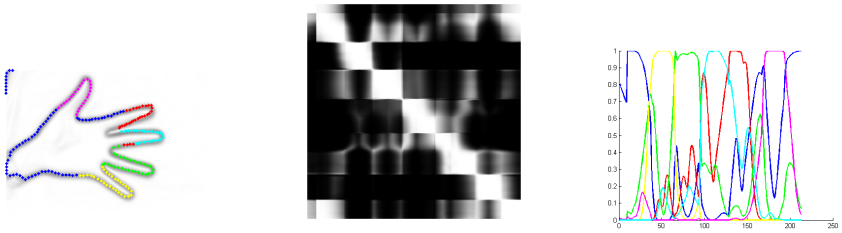


Figure 4.2.13: Example of direct segmentation of a finger using the four different affinity matrices presented in Section 4.2.4.1. The four rows of images correspond to the four approaches (in the order of their number). The first column shows the clustering results, the second column represents the affinity matrix, and the third column shows the lines of the affinity matrix corresponding to the features selected as group leaders.

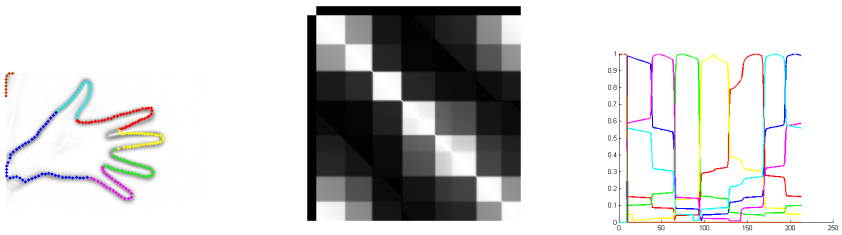


(a) (b) (c)



(d) (e) (f)

(g) (h) (i)



(j) (k) (l)

Figure 4.2.14: Example of direct segmentation of a hand using the four different affinity matrices presented in Section 4.2.4.1. The four rows of images correspond to the four approaches (in the order of their number). The first column shows the clustering results, the second column represents the affinity matrix, and the third column shows the lines of the affinity matrix corresponding to the features selected as group leaders.

Based on these ideas and inspired from the on-line segmentation method presented in Section 4.1.2.2, we can adapt the grouping procedure presented in the previous section into the procedure shown in Algorithm 4.2. This method mainly focuses on (a) maintaining an affinity matrix between leaders and points and (b) creating new leaders once enough correlated outliers have been detected.

Ultimately, the few extra relations created between the leaders will allow the existence of uncorrelated subgraphs to be highlighted, resulting in the creation of new nodes into the block level. The detection of uncorrelated subgraphs can be easily done using a hierarchical clustering [42] of the leaders based on their affinity matrix.

4.3 Experiment

In this section, we will experiment with the different learning processes based on tracking results independent of the learnt model. This allows us to get a better analysis of the learning phase without interferences coming from its combination with the tracking process. When necessary to demonstrate the qualities of the solutions provided, we will use a simplified combination of learning and tracking that focus on the element to test.

The rest of this section is divided into three parts where each corresponds to the learning of a different type of *uncertain* model: rigid, flexible, and articulated. The segmentation of the edgels into rigid blocks has already been experimented with in Section 4.2.4. We therefore postpone a deeper analysis to Chapter 5, where learning will be applied simultaneously to tracking.

4.3.1 Uncertain Rigid Relation Learning

Since *uncertain* rigid models are specifically designed to assist tracking without exerting overly strong counterproductive bias, it does not make much sense to learn them based on some already available tracking results. Learning the maximum likelihood Gaussian model is indeed nothing of a challenge. It is far more interesting to observe how the uncertainty factor allows flexibility and robustness to be balanced in the assisted tracking. In order to do so without interfering too much with the next chapter, we propose to demonstrate the performance of our method on the representative simplified example shown in Figure 4.3.1 and previously published in [25]. In this example, we manually define a very small graph of four nodes in order to limit the number of relations and therefore simplify their analysis. To emphasise the contribution of the relations, we chose to use a very simple feature descriptor. Features are represented by fixed image templates extracted from the first frame. The likelihood of the features is computed using the sum of squared pixel differences. The 2D coordinates of these features are tracked with particle filters and Belief Propagation; no orientation or scale changes are considered. The informative part of the relations is represented by a Gaussian model for each coordinate. We use $\mathbf{C}_\Delta = \begin{bmatrix} \sigma_\Delta & 0 \\ 0 & \sigma_\Delta \end{bmatrix}$

Algorithm 4.2 Feature Groups Maintenance

1. **Affinity Update**

   (a) From the tracking results at time t, update the *uncertain* relations of the graph.

   (b) Compute the affinity matrix A_t between the M leaders and the N points using the *Affinity Propagation* presented in Algorithm 4.1 but for a matrix A_t of size $M \times N$ (i.e. the messages are of size M instead of size N).

2. **Leaders Update**

   (a) For each leader i that does not have the highest affinity with at least $N_{\text{minInliers}}$ points, remove all the connections with the other leaders and delete i from the list of leaders.

   (b) Create the outliers set S_{out} with all the points that do not have an affinity greater or equal to A_{thres} with at least one of the leaders and compute the full K^2 affinity matrix of this subset of K points using *Affinity Propagation*.
   If the point with the maximum weight $w_{i,t} = \sum_k A_{i,k}$ has enough inliers (i.e. $\sum_k (A_{i,k} \geq \text{A}_{\text{thres}}) \geq N_{\text{minInliers}}$), define it as a leader and create *uncertain* relations with all the other leaders.

3. **Groups Update**

   (a) Form a group for each leader using the points for which this leader provide the best affinity.

   (b) From the fully connected graph of leaders, use *uncertain* relations to compute a hierarchical clustering of the groups.
   If a clear lack of affinity appears between the two top clusters of groups, permanently segment them into two rigid blocks and create *uncertain* articulated relations between these two new blocks and their close neighbours.

.

with $\sigma_\Delta = 5$ pixels and $T_D = 0.04$ pixels. The fact that we are using a 2D space instead of the affine space does not have an impact on the computation of the model fidelity λ_t (since it is computed separately for each dimension anyway) and allows us to keep the analyse as simple as possible.

The graph consists of two features from the face and two from the background. The graph is fully connected so that every node is linked by two non-rigid relations and one rigid relation. A perfectly rigid relation is represented by a thick red line. The line becomes thinner if the model variance $\tilde{\mathbf{C}}_t$ increases and whiter if its fidelity λ_t decreases. Green rectangles represent the nodes of the graph and the green dots their associated particles.

In this example, the relations are first modelled with high variance (uncertainty in the parameters) but they quickly become rigid since the scene is initially motionless. As we can see in Figures 4.3.1b and 4.3.2a, the relations related to the mouth are learnt more slowly than the others due to their lower likelihood in the image. Once the head starts to move, the rigid relations connecting the head with the background are rapidly unlearned. The probabilities λ_t of these relations become insignificant, and their variances increase. This clearly separates the graph into two subgraphs, one for the face and another one for the background. Over the following frames, the face is successfully tracked despite the occlusions and its out-of-plane motions. Once the relations between face and background have been detected as non-rigid, they do not influence the tracking any more.

To illustrate the effect of the *uncertain* models on tracking, we track duplicate versions of the features without relations, represented in blue in Figure 4.3.1. As the figure reveals, these features are tracked very poorly and have to be reinitialised many times during the sequence. It is thus clear that it would have been very difficult to learn a relational model from them without exploiting the – albeit uncertain – partially-learnt relations from the start.

The end of the sequence (frames 700–900) is mostly motionless. Figure 4.3.2 shows that the probability of the forehead–mouth relation slowly increases and that all variances decrease. The probabilities of the relations between the facial features and the background do not increase because these relations were clearly non-rigid during the major part of the sequence. It will thus take much more time for their observation distributions to return to a Gaussian shape.

On a Intel Core 2 T7200 at 2GHz, the computational time required to learn this small set of relations is insignificant. By testing our method on a larger graph of 200 relations, we obtain a computational time of 0.3ms for the learning phase of *uncertain* rigid relations. We can therefore safely say that the learning process can be applied simultaneously to the tracking at virtually no extra cost.

While we demonstrate the benefit of using *uncertain* rigid relations over individual feature tracking, we might wonder how an *uncertain* graph compares to a regular one. In Figure 4.3.2, we can indeed notice that the variance and the probability of the model have similar reactions. We might then wonder if the increase in the variance is not enough to avoid the strong counterproductive bias and therefore question the usefulness of the uniform distribution. The example in Figure 4.3.3 is based on the same video and the same initialisation as in

the previous section but the potential functions are learnt without the uniform part. As we can see, even if the variance increases once the head moves, making it possible to track the head successfully at the beginning of its motion, the tracking fails after a few frames. The reason for this result is simple: once the observations are not well represented by the parametric model, any increase in variance will not entirely eliminate the counterproductive bias created by the inappropriate model. Even worse, an inappropriate model will bias future observations that will, in turn, reinforce the belief that the model is correct. The ability of the *uncertain* model to adapt its own influence is then a key element for simultaneous learning and tracking.

4.3.2 Uncertain Flexible Relation Learning

Although the simpler articulated model has been favoured over this flexible model in our specific case, it is still a strong candidate for future improvements where more complex relations might be needed. Moreover, since it relies heavily on the fidelity of the *uncertain* rigid model to detect when to increase its complexity, it is a good way to test the *uncertain* rigid model more extensively.

First, we will use some generated data to analyse the influence of different elements on the final result. We will then demonstrate with a real video sequence that the results obtained with real data behave identically to the empirical results.

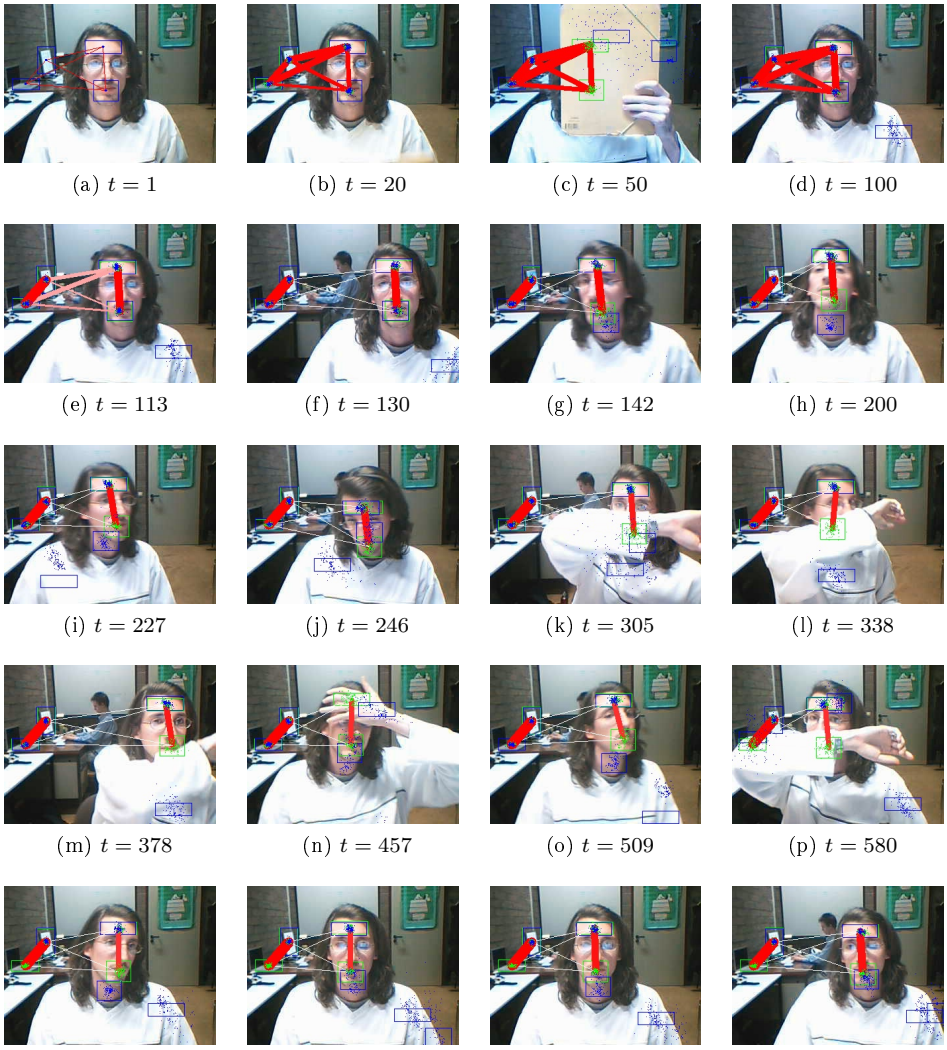
4.3.2.1 Empirical Analysis

To analyse the relation between the complexity of the model and the only parameter T_D, we generated data from a circular distribution (points distributed around a circle) for different values of T_D from 0.01 to 0.25. We ran 30 tests per value of T_D and stopped each test after 500 observations. As we can see in Figure 4.3.4a, T_D provides a simple way to specify the desired trade-off between the model complexity and its accuracy.

Since the learning is incremental, we may wonder whether the model will always converge to qualitatively the same result. We therefore performed the same experiment with $T_D = 0.04$ and with angular velocities between 0.01 and 2 rad/frame for the process that generates the observations. As shown in Figure 4.3.4b, the model complexity is nearly independent of the angular velocity.

4.3.2.2 Learning From Blocks Tracking

Figure 4.3.5 shows an example of the learning of the articulated relation existing between a hand and the image (or any static object). In order to produce results that are easy to visualise, we learn the relation in a 2D space from an independent 2D tracking of the hand with a particle filter. The method is first tested with $T_D = 0.04$ and a learning procedure that uses all frames to update the model (row 1). The same procedure is then tested using only one in ten frames (row 2). As one can see, the resulting model is not influenced by this



(a) $t = 1$ (b) $t = 20$ (c) $t = 50$ (d) $t = 100$

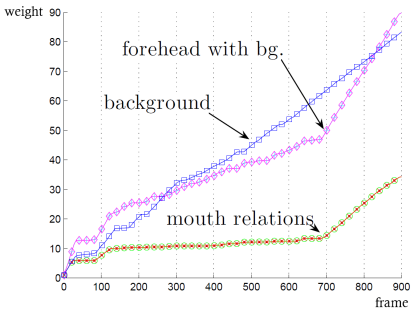
(e) $t = 113$ (f) $t = 130$ (g) $t = 142$ (h) $t = 200$

(i) $t = 227$ (j) $t = 246$ (k) $t = 305$ (l) $t = 338$

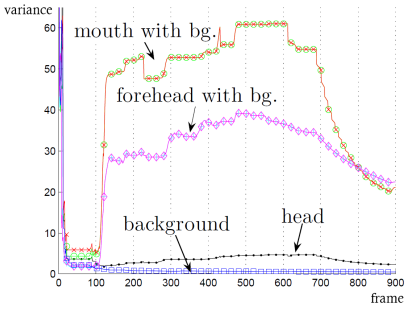
(m) $t = 378$ (n) $t = 457$ (o) $t = 509$ (p) $t = 580$

(q) $t = 683$ (r) $t = 700$ (s) $t = 800$ (t) $t = 900$

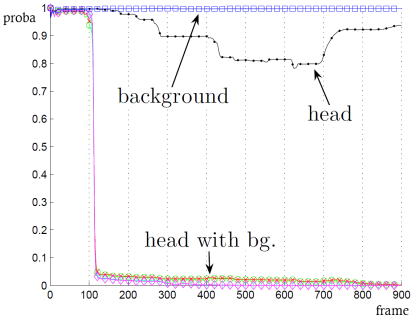
Figure 4.3.1: Simplified example of simultaneous tracking and learning of an *uncertain* rigid graph. The thickness of the lines is inversely proportional to the variance $\tilde{\mathbf{C}}$ of the corresponding relation. Red saturation is proportional to the probability λ_t. Each feature is tracked twice, with relations (green) and without (blue). Frame indices are given below each image.



(a) Weight.



(b) Variance.



(c) Probability.

Figure 4.3.2: Evolution of the weight, variance, and probability of the relations through the video. The two relations between mouth and background are superimposed, as are those between the forehead and the background. In (a), all three mouth relations are superimposed.

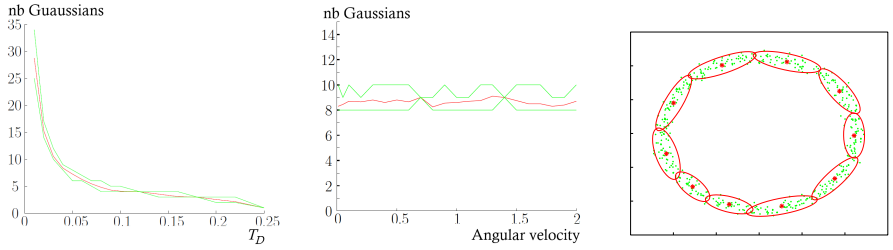
(a) $t = 1$ (b) $t = 51$ (c) $t = 81$ (d) $t = 101$

(e) $t = 105$ (f) $t = 108$ (g) $t = 111$ (h) $t = 113$

(i) $t = 114$ (j) $t = 115$ (k) $t = 116$ (l) $t = 117$

(m) $t = 118$ (n) $t = 119$ (o) $t = 120$ (p) $t = 121$

Figure 4.3.3: Simultaneous learning and tracking without the *uncertain* model. Even if the variance increases once the head starts to move, allowing it to be tracked successfully at the beginning of its motion, the tracking fails after a few frames.



(a) T_D (b) Angular velocity. (c) Example ($T_D = 0.04$).

Figure 4.3.4: Dependency of the number of Gaussians in the mixture model on (a) T_D and (b) the angular velocity. The red line represents the means of the 30 tests for each value, and the green lines represent the extrema.

difference in the dataset (except, of course, for the difference of covariances due to a difference of evidence accumulation). The third row shows the result for a smaller value of T_D. As expected, a smaller value of T_D results in a mixture with more Gaussians.

4.3.3 Uncertain Articulated Relation Learning

4.3.3.1 Empirical Analysis

Here we analyse the robustness of the model to noise and how the model fidelity reflects the correspondence between the observed data and a perfectly articulated model. To do so, we simulate the learning of an *uncertain* articulated model between two nodes i and j by sampling points along an ellipse. To keep things as simple as possible, we will work in a 3D space: the first two dimensions correspond to the position and the third to the orientation. The relative positions of the joints are learnt in the reference frame of each node (both are needed to define the model fidelity) and the fidelity of the model is learnt in the referential of node i. In Figure 4.3.6, everything is represented in the reference frame of node i (so node i will always be at the origin). The positions of node j are sampled along an ellipse with a radius $r = (r_1, r_2)$ and their orientation is defined so that they are pointing toward the centre of the ellipse (here $(2, 0)$). Gaussian noise with a standard deviation of $n * (0.1, 0.1, 0.01)$ is then added to each sample. These samples are shown in green in Figure 4.3.6 while their prediction of the position of the joint is shown in magenta. The red line connects the last sample (the 1000th) to its prediction. Cyan points correspond to the position of the joint as expected by i after upgrading the model with a new sample. The blue line connects the origin and the expected position of the joint after upgrading the model with a new sample. The radius of the blue circle corresponds to the standard deviation of the observed distances between the position of the joint predicted by the two nodes at each time. The graph represents the evolution of the model fidelity as learnt by node i.

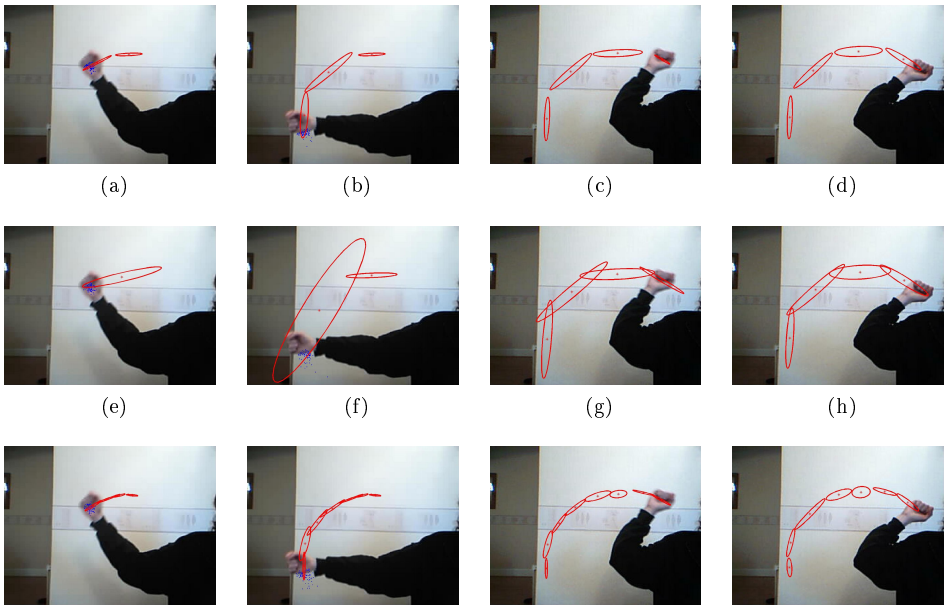


Figure 4.3.5: *Uncertain* flexible model learnt from a webcam at different times of the video. First row: $T_D = 0.04$. Second row: $T_D = 0.04$ but the model is only learnt every ten frames. Third row: $T_D = 0.02$. In each row, the frames no. 70, 110, 220, and 450 are shown. In frame 450, the hand has accomplished a second cycle similar to the one from frame 1 to frame 220.

The first two columns of Figure 4.3.6 correspond to the learning result of a perfectly articulated relation for various values of the noise factor n. While the time needed for the model to stabilise increases with the level of noise, we can see that the result always converges toward a perfectly articulated model.

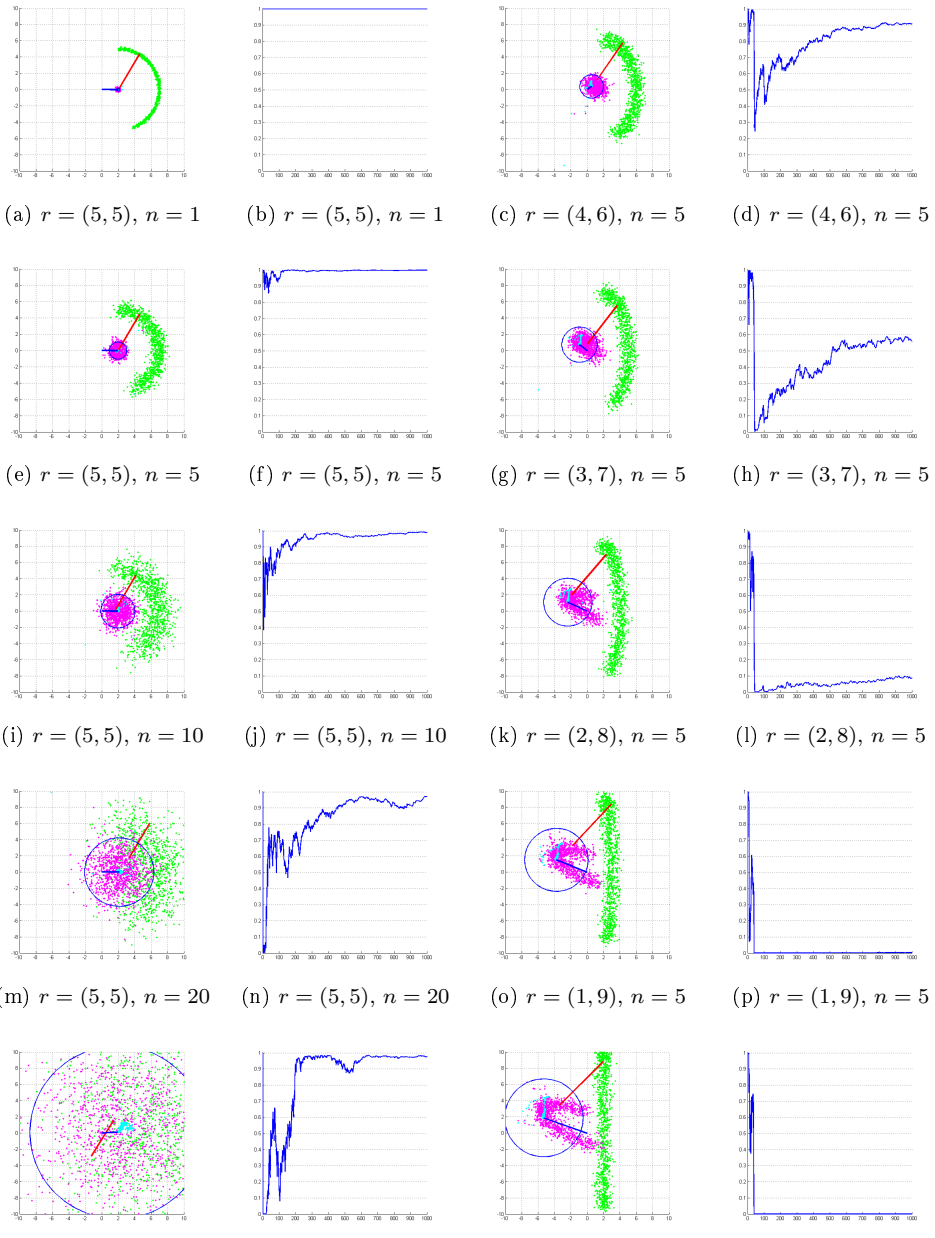
The last two columns correspond to the learning result for different values of the radius r of the ellipse. As we can see, the further the ellipse is from a perfect articulated relation, the lower the fidelity of the learnt model is. Obviously, the tolerance of the model depends on the parameter T_D, as we have demonstrated in Section 4.3.2. Here we chose $T_D = 0.04$ but the model will become more or less tolerant to non-exact relations if we respectively increase or decrease T_D.

4.3.3.2 Learning From Blocks Tracking

In order to demonstrate how the fidelity of the model behaves in real videos, we used the tracking results obtained in Figures 3.3.8 and 3.3.9 from Section 3.3.2 for the tracking of rigid blocks using the *Aligned Particle Filter* method. The learnt *uncertain* articulated model is shown in Figure 4.3.7 at different times of the sequence. The model is represented using lines connecting each node to its prediction of the position of the joint. The colour of these lines corresponds to the fidelity of the learnt articulated model (from green for high fidelity to red for low fidelity). Circles are also used to represent the learnt standard deviation of the observed distances between the two positions of the joint.

As we can see, the relation is learnt as a nearly exact articulated relation until time $t = 300$ (a bit of noise is accounted for as we can see by the size of the circles). Around that time, the upper arm also starts to move. This means that the elbow will not be fixed with respect to the torso area anymore and this relation should not contribute to the tracking any longer. In order to trigger this modification, the current articulated model should first be detected as inappropriate, which is exactly what happens between $t = 340$ and $t = 360$ when the model fidelity decreases from a value close to one to a value close to zero.

On a Intel Core 2 T7200 at 2GHz, the computational time of the learning process is roughly the same as for the rigid relations: 0.3ms for 200 relations. This can be explained by the fact that the reduced dimensionality compensates for the computation of the position of the joint. Given that the number of articulated relations will generally be quite low, we can safely consider that this step is also insignificant in terms of computational cost.



(a) $r = (5, 5)$, $n = 1$ (b) $r = (5, 5)$, $n = 1$ (c) $r = (4, 6)$, $n = 5$ (d) $r = (4, 6)$, $n = 5$

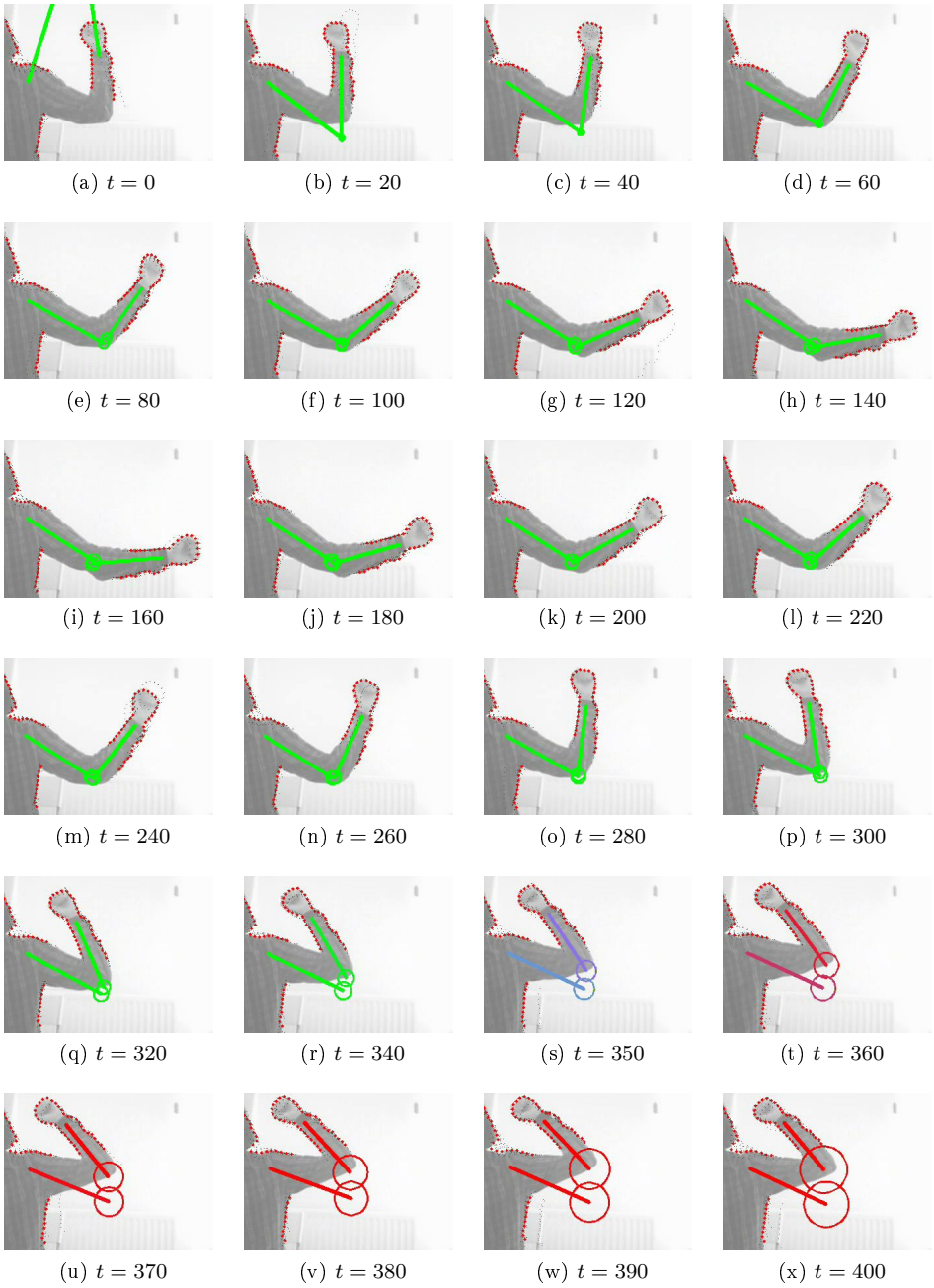
(e) $r = (5, 5)$, $n = 5$ (f) $r = (5, 5)$, $n = 5$ (g) $r = (3, 7)$, $n = 5$ (h) $r = (3, 7)$, $n = 5$

(i) $r = (5, 5)$, $n = 10$ (j) $r = (5, 5)$, $n = 10$ (k) $r = (2, 8)$, $n = 5$ (l) $r = (2, 8)$, $n = 5$

(m) $r = (5, 5)$, $n = 20$ (n) $r = (5, 5)$, $n = 20$ (o) $r = (1, 9)$, $n = 5$ (p) $r = (1, 9)$, $n = 5$

(q) $r = (5, 5)$, $n = 50$ (r) $r = (5, 5)$, $n = 50$ (s) $r = (0, 10)$, $n = 5$ (t) $r = (0, 10)$, $n = 5$

Figure 4.3.6: *Uncertain* articulated relation learning between nodes i and j. In green: positions of j in the reference frame of i. In magenta and cyan: respectively j's and i's predictions of the joint position. In blue: line from origin to learnt joint position, circle with a radius equals to 1.5 the mean distance between the positions of the joint predicted by the two nodes at each time step. The graph corresponding to the evolution of the model fidelity. .



(a) $t = 0$ (b) $t = 20$ (c) $t = 40$ (d) $t = 60$

(e) $t = 80$ (f) $t = 100$ (g) $t = 120$ (h) $t = 140$

(i) $t = 160$ (j) $t = 180$ (k) $t = 200$ (l) $t = 220$

(m) $t = 240$ (n) $t = 260$ (o) $t = 280$ (p) $t = 300$

(q) $t = 320$ (r) $t = 340$ (s) $t = 350$ (t) $t = 360$

(u) $t = 370$ (v) $t = 380$ (w) $t = 390$ (x) $t = 400$

Figure 4.3.7: Leanring of an *uncertain* articulated relation from a real video. The colour of the lines connecting a node to a joint corresponds to the fidelity of the learnt articulated model (from green for high fidelity to red for low fidelity). The circles correspond to the learnt distance between the joint positions as predicted by each node.

4.4 Discussion

In this chapter, we focused on learning the two-level model based on available tracking results. The key element of this chapter is the concept of *uncertain* model. It allows us to use simple and fast parametric models to assist tracking during their learning phase without exerting overly strong, counterproductive bias if these models are inappropriate. The *concept of uncertain* model was applied to a rigid and an articulated model that are used to model the relations respectively in the feature level and the block level. We also created a more versatile *Uncertain Gaussian Mixture Model* that demonstrated the benefit of the concept of uncertainty to automatically adapt the number of Gaussians in the model.

The *uncertain* relations learnt at the feature level also provide an easy and fast starting point to segment the features into rigid blocks. By combining them with the solutions presented in Section 4.2.4, we were able to create a robust and fast solution for on-line discovery of rigid blocks.

Chapter 5

Simultaneous Learning and Tracking

This chapter finally applies simultaneously the different methods developed in the previous chapters. We will address the different problems that occur when the different elements of learning and tracking are combined together.

In Section 5.1, we consider the tracking of an articulated object using a single rigid template. This situation is particularly present for a new object since it is first modelled as a single rigid block. With the need to track properly all the parts of the object in order to discover them, we will propose an adapted version of the particle filter that is able to follow properly all the rigid parts without an appropriate articulated model.

In Section 5.2, we will then discuss the communication between the feature level and the block level and summarise how features and blocks can bring respectively flexibility and robustness to each other. In Section 5.3, we will also briefly discuss how a different choice of features can make the tracking of the feature level more robust (at the cost of a less stable initialisation).

Finally, In Section 5.4, we will improve the robustness of the block tracking in order to cope with the presence of clutter during simultaneous learning and tracking.

5.1 Block Level - Simultaneous Rigid Block Discovery and Tracking

Until now, we have always tracked blocks corresponding to a single rigid part of an object. When a single block covers multiple large rigid parts, the situation becomes more difficult. The first row of Figure 5.1.1 provides an example of this kind of situation where the whole visible part of the body is tracked using a single block and 10 particles. In this example, only the forearm is moving, leaving the rest of the body static. As we can see at time $t = 50$, the particles

are first able to cover both the forearm and the body. At time $t = 60$, the particles are already concentrated on the dominant part of the body. From that moment, any information provided by the block to initialise the tracking of the features situated on the forearm would simply provide a worse result than by leaving the features on their own.

Luckily this situation is easy to detect since the likelihood of the forearm will be quite low. Once the problem is detected, we could simply generate a new set of particles that focuses on that part of the object (in this case, the forearm) using the corresponding features tracked on the feature level. This procedure can be defined with the following steps:

1. The particles are now divided into sets created in previous frames. To each set is associated a list of weights used to describe its affinity with the different features. These weights are used in the gradient descent and image likelihood computation and allow the particle to focus on a specific part of the object.

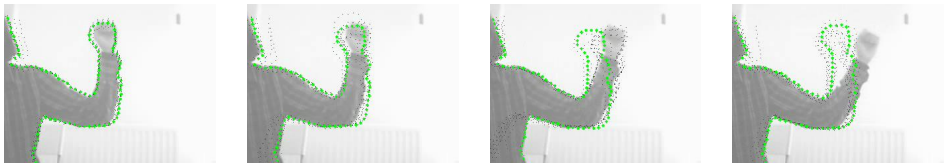
2. When a group of features is not properly described by any of the particle in their set (image likelihood under a given threshold), they each vote for a particle that would position them properly. The particle that receives the larger number of votes is used to generate a new particle set.

3. Each particle is resampled within its own set.

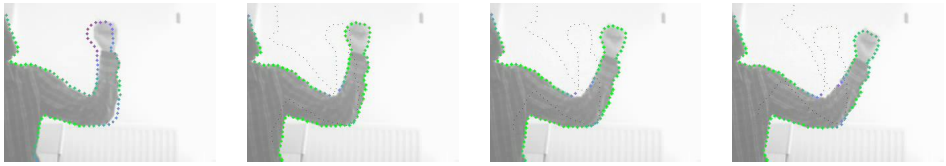
This approach is fairly similar to the Algorithm 4.2 presented in Section 4.2.4.2 for the maintenance of feature groups over time. One could actually wonder why the learning process is not sufficient to solve our problem here. The main reason is linked to the method chosen in Section 4.2.4 to compute the affinity matrix. While this was the most robust solution for rigid block discovery, it was also the slowest. Since, here, our only concerns are tracking and quick detection, it is preferable to use another method. One that is immediately available is the oriented chamfer distance d_λ of the features when projected by the particles since that is already calculated for the particles weight. The affinity matrix between particles and features is therefore available at no extra cost.

With sets of particles that might more frequently over-segment the features and no need to learn anything from them, the priority here should be on providing the simplest sets of particles than can provide a good description of the features (and therefore a good initialisation for their tracking). A solution to this problem has already been discussed in Section 4.2.4.1 and can easily be adapted here to give:

1. Compute the weights $w_{i,t} = \sum_k A_{i,k,t} \cdot d_{\lambda,i,k,t}$ of each particle i in order to represent its ability to cover a large set of features. $d_{\lambda,i,k,t}$ represents the oriented chamfer distance of feature k when projected by particle i in the image from time t. $A_{i,k,t}$ represents the expected affinity between a particle i and a feature k. For simplicity, we used $A_{i,k,t} = d_{\lambda,i,k,t-1}$ but more complex models can be defined.



(a) $t = 30$ (b) $t = 40$ (c) $t = 50$ (d) $t = 60$



(e) $t = 30$ (f) $t = 40$ (g) $t = 50$ (h) $t = 60$

Figure 5.1.1: Examples of block tracking using either the classical particle filter (first row) or the adapted particle filter (second row).

2. Select the particle i with the biggest weight $w_{i,t}$ and generate a set of n particles from it, n being the number of particles by default for the particle filter .

3. Remove those inlier features and repeat the two steps above until no group can be formed with at least $N_{\text{minInliers}}$ inliers.

4. Refine the clustering by associating each point k with the group with the highest affinity towards k. The affinity vector of a group is defined by the row $A_{i,:,t}$ of the affinity matrix corresponding to the point i used to create it.

With this approach, we always use the minimum amount of particles necessary to provide a good coverage of the features. Indeed, there is no need to remove particle sets that do not have enough outliers since they will simply not be selected during step 2. An interesting result of this solution is also that new sets of particles are often created without even having to consult the feature level. Indeed, if we look at the first row of Figure 5.1.1, we can see that, given the chance, a new set of particles could have been created from the particles still clustered around the forearm at time $t = 50$. When this happen, the new set of particles is generated at virtually no extra cost since the particle used to create the new set (and its affinity with the features) was already computed. A case of tracking using this new approach is shown in the second row of Figure 5.1.1.

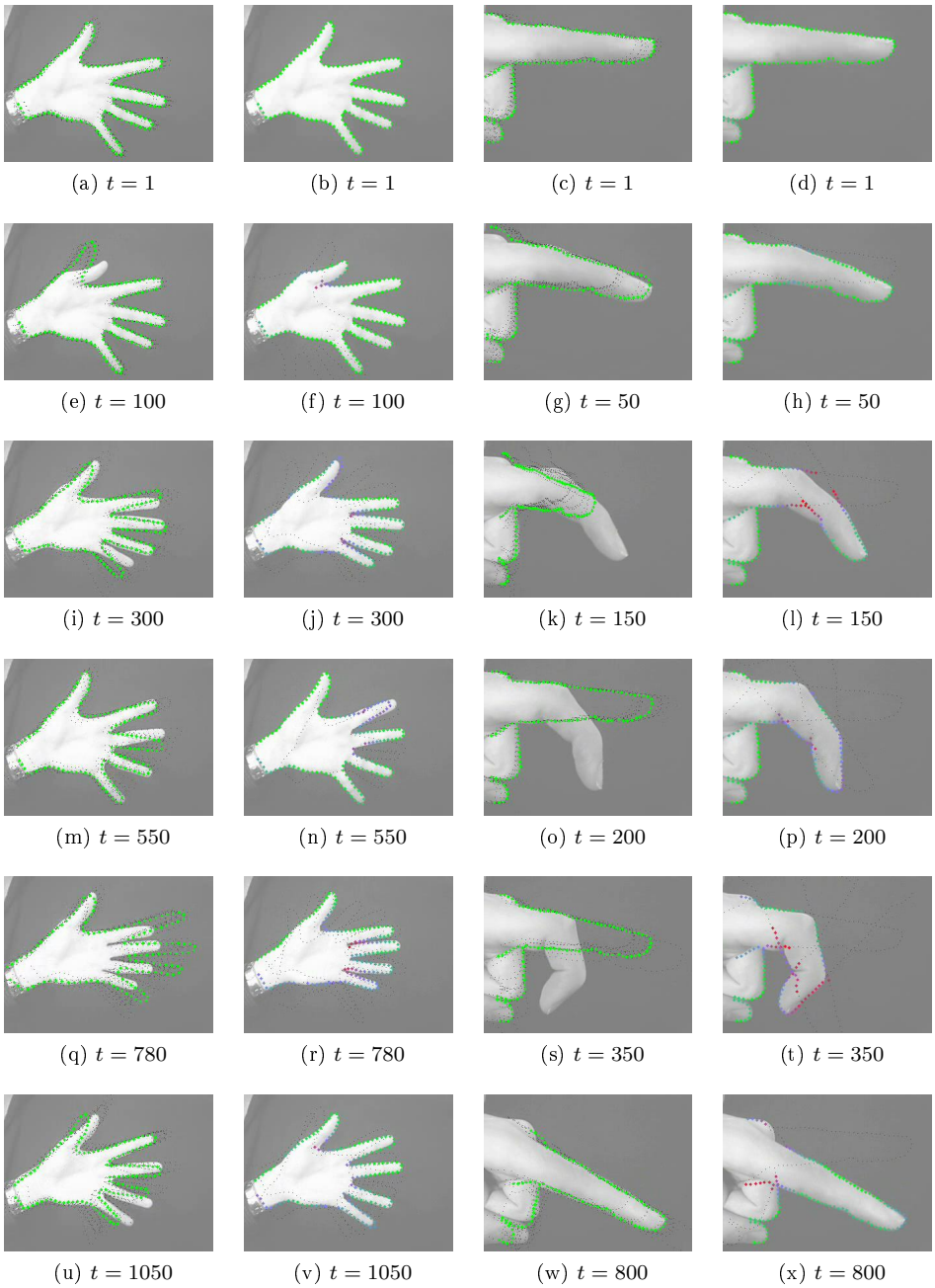
From the result in Figure 5.1.1, we might think that particles could automatically split themselves into the different rigid parts without the help of the feature level. Unfortunately, there is not always a particle covering each part of the object, especially with so few of them (we generally use 10 particles). The most problematic situations are those where a small part is far from the centre

Algorithm 5.1 Particle filter for non-rigid objects. The particles are grouped into sets with more affinity for a specific part of the object. When there are no more particles covering a part, a new set is generated using the feature level.

1. **Particle Tracking**
   (a) Propagate the particles from last frame.
   (b) Compute the weights $w_{i,t} = \sum_k A_{i,k,t} \cdot w_{\lambda,i,k,t}$ of each particle i in order to represent its ability to cover a large set of features. $w_{\lambda,i,k,t}$ represents the image likelihood computed with the oriented chamfer distance $d_{\lambda,i,k,t}$ of feature k when projected by particle i in the image from time t. $A_{i,k,t}$ represents the expected affinity between a particle i and a feature k. For simplicity, we used $A_{i,k,t} = w_{\lambda,i,k,t-1}$ but more complex models can be defined.
   (c) Select the particle i with the biggest weight $w_{i,t}$ and generate a set of n particles from it, n being the number of particles by default for the particle filter.
   (d) Remove those inlier features and repeat the two steps above until no group can be formed with at least $N_{\text{minInliers}}$ inliers.
2. **Particle Set Creation**
   (a) Generate a particle from each outlier feature.
   (b) Compute the weights $w_{i,t} = \sum_k w_{\lambda,i,k,t}$ for each of these particles.
   (c) Select the particle i with the largest weight $w_{i,t}$ (if it cover enough outliers) . Generate a set of n particles from particle i.
   (d) Remove these outliers features and repeat the two steps above until no group can be formed with at least $N_{\text{minInliers}}$ inliers.

of the block. In this case, any rotation from this part will correspond to a big displacement of the centre of the whole block. To keep that part covered, we would need a particle sampled that far from the rest and with the correct orientation. With such a low number of particles, the chances for that to happen are very small. If no proper particle is available, a search could be triggered to find the missing part in the area where it is likely to be but it may be time consuming and there is no guarantee that the outliers can be grouped into a single rigid part. By using the information provided by the feature level we can directly generate particles (one for each outlier) where they are needed and then use the same process as above to select the set(s) that cover(s) the outliers.

The whole algorithm used to track and maintain the particles is detailed in Algorithm 5.1, and more tracking results are shown in Figure 5.1.2.



(a) $t = 1$ (b) $t = 1$ (c) $t = 1$ (d) $t = 1$

(e) $t = 100$ (f) $t = 100$ (g) $t = 50$ (h) $t = 50$

(i) $t = 300$ (j) $t = 300$ (k) $t = 150$ (l) $t = 150$

(m) $t = 550$ (n) $t = 550$ (o) $t = 200$ (p) $t = 200$

(q) $t = 780$ (r) $t = 780$ (s) $t = 350$ (t) $t = 350$

(u) $t = 1050$ (v) $t = 1050$ (w) $t = 800$ (x) $t = 800$

Figure 5.1.2: Examples of block tracking using either the classical particle filter (columns 1 and 3) or our adapted particle filter (columns 2 and 4).

5.2 Communication Between Levels

In Section 3.2.4, we briefly discussed how each level of the graph could be beneficial for the other. The blocks could assist the tracking of the feature level while the flexibility of the feature level could be used to segment the blocks into rigid parts. While the rigid block discovery has already been discussed in details in Section 4.2.4, nothing has been done yet about using the blocks to improve the tracking of the features.

With the new solution provided in Section 5.1 to track properly a non-rigid block, this block can provide a good initial position to all his inlier features. When a feature is not an inlier to any particle, its initial position will simply be its position from the previous frame. In this way, features are left on their own only if more flexibility is needed (the current particles are too far from the correct position). With this approach, the feature level will therefore benefit both from the more robust tracking of the blocks and from its flexibility to non-rigid transformation.

Overall, the communication between the two levels is fairly simple and can be summarised with

- **Top-down:**
  - The leading particles used to track the blocks are used to initialise the position of their inlier features.
- **Bottom-up:**
  - Features that are outliers to any leading particle are used to generate new particles for the corresponding block.
  - Learnt relations between features are used to detect the lack of affinity between parts of a single block and to define how to segment that block.

Results obtained from simultaneously learning and tracking the feature level using an initialisation from the block level are shown in Figure 5.2.1. In these experiments, we also constrained the movements of the features that were already well described by their block. To do so, the final position of each feature is defined by $\mathbf{x}_i = w_i^b \cdot \mathbf{x}_i^b + (1 - w_i^b) \cdot \mathbf{x}_i^f$ where $\mathbf{x}_i^b$ is the initial position proposed by the block, w_i^b its image likelihood, and $\mathbf{x}_i^f$ its position after the warp propagation. While it makes sense to keep a position that has been validated by more robust tracking results, this approach is also very useful when using edgels for simultaneous learning and tracking. In non-rigid areas, the edgels indeed tend to slide along the contours in order to spread the deformation between as many spatial relations as possible (see Figure 5.2.2, first column second row for an example). Combined with features defined in an affine space, any transformation (rigid or not) of the object can usually be described locally without the need to lower the rigidity of the relations. If they do weaken, it is usually in a wrong position often causing the tracking of the segmented parts to fail

completely. See Figure 5.2.2 for examples of simultaneous learning and tracking using only the feature level. By comparison, we can see that the relations are usually weakened on a single appropriate place when using both levels (see last row of Figure 5.2.1 for the final segmentation).

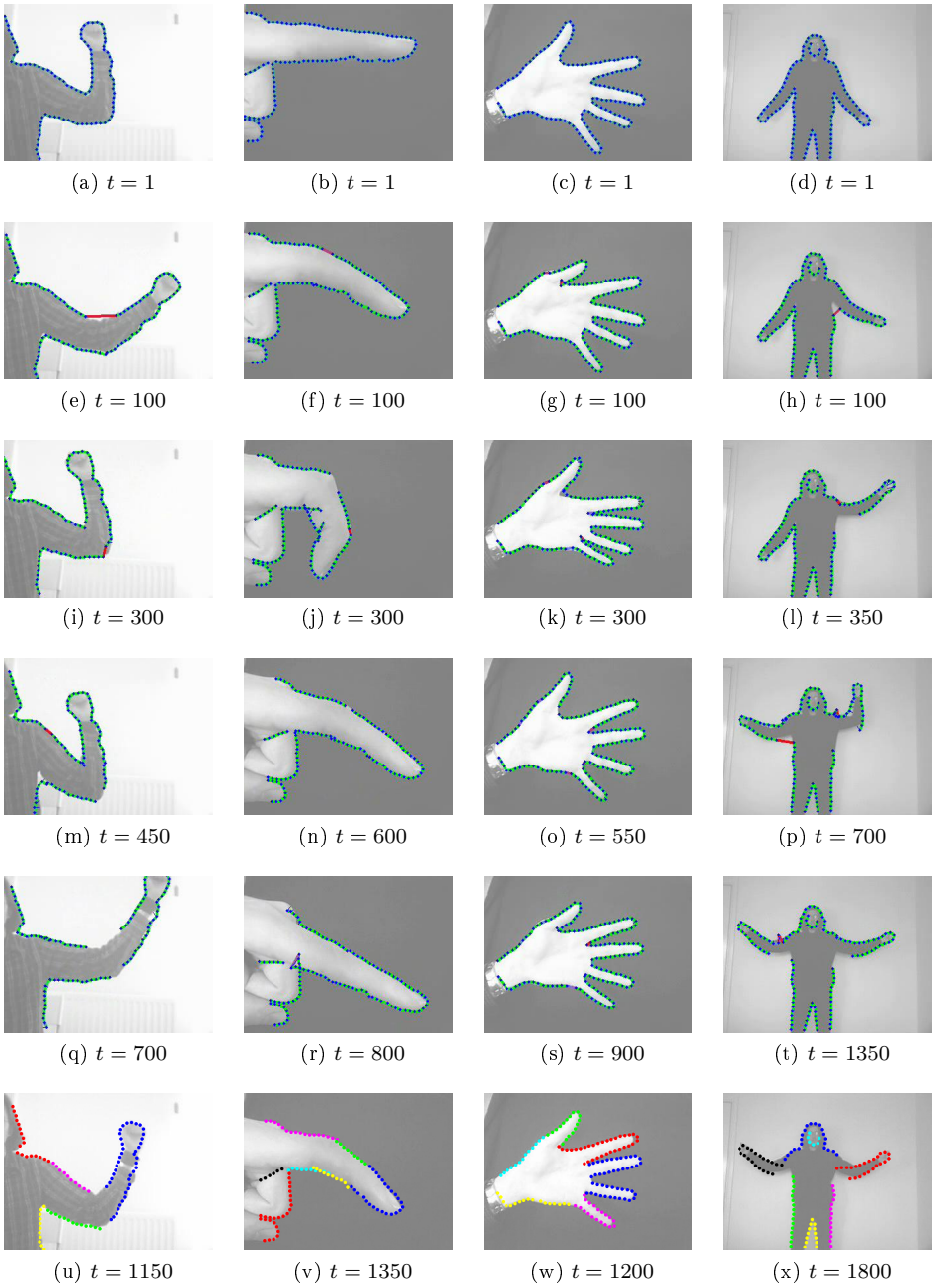
5.3 Skeleton Simultaneous Learning and Tracking

Although the two levels have proved to be mandatory to each other in our particular case, it is important to notice that this is not always true depending on the choice of features. The sole purpose of the block level is indeed to provide a robust tracking that the feature level fails to provide using edgels. For example, in our article *Affine Warp Propagation for Fast Simultaneous Modelling and Tracking of Articulated Objects [24],* we have used skeleton points instead of edgels. Each skeleton point is defined by a set of at least two equidistant edgels used for its tracking. These points are obtained by sampling the Voronoi skeleton of the edgels. In Figure 5.3.1, we show a few results of simultaneous learning and tracking of such a model. As we can see, both tracking and learning provide a satisfactory result without the need for a block level.

As discussed in Chapter 2, the skeleton initialisation is very sensitive to any change in the contour, disqualifying it in our case. It is clear though that, when the initialisation can be done in a more controlled environment, the better tracking robustness of the skeleton might make it a more desirable candidate than the edgels.

5.4 Block Level - Simultaneous Relations Learning and Tracking

With all the problems on the feature level solved, our last task is to make sure that simultaneous learning and tracking work smoothly on the block level as well. With a much more robust tracking, combining learning and tracking should give results similar to those already presented during the experiments on the *uncertain* rigid model (Sections 4.3.1 and 4.3.3). Most of the time, it is the case but the use of contours instead of image templates tends to create more challenging situations. Given that contours are not very discriminative, clutter is indeed more frequent. This makes difficult scenarios such as the one presented in Figure 5.4.1 much more likely. This is again a tracking based on our usual video sequence but where we used the learnt relation to assist the tracking. During the first 300 frames, while the relation is rigid, the result is similar to the one presented in Figure 4.3.7. When the arm starts to rise around frame 350, the forearm moves along its own axis. Moreover, contours exist along the closed fingers (that are now more visible than at the beginning of the sequence) and along the limit between wrist and shirt. This therefore allows the block to



(a) $t = 1$ (b) $t = 1$ (c) $t = 1$ (d) $t = 1$

(e) $t = 100$ (f) $t = 100$ (g) $t = 100$ (h) $t = 100$

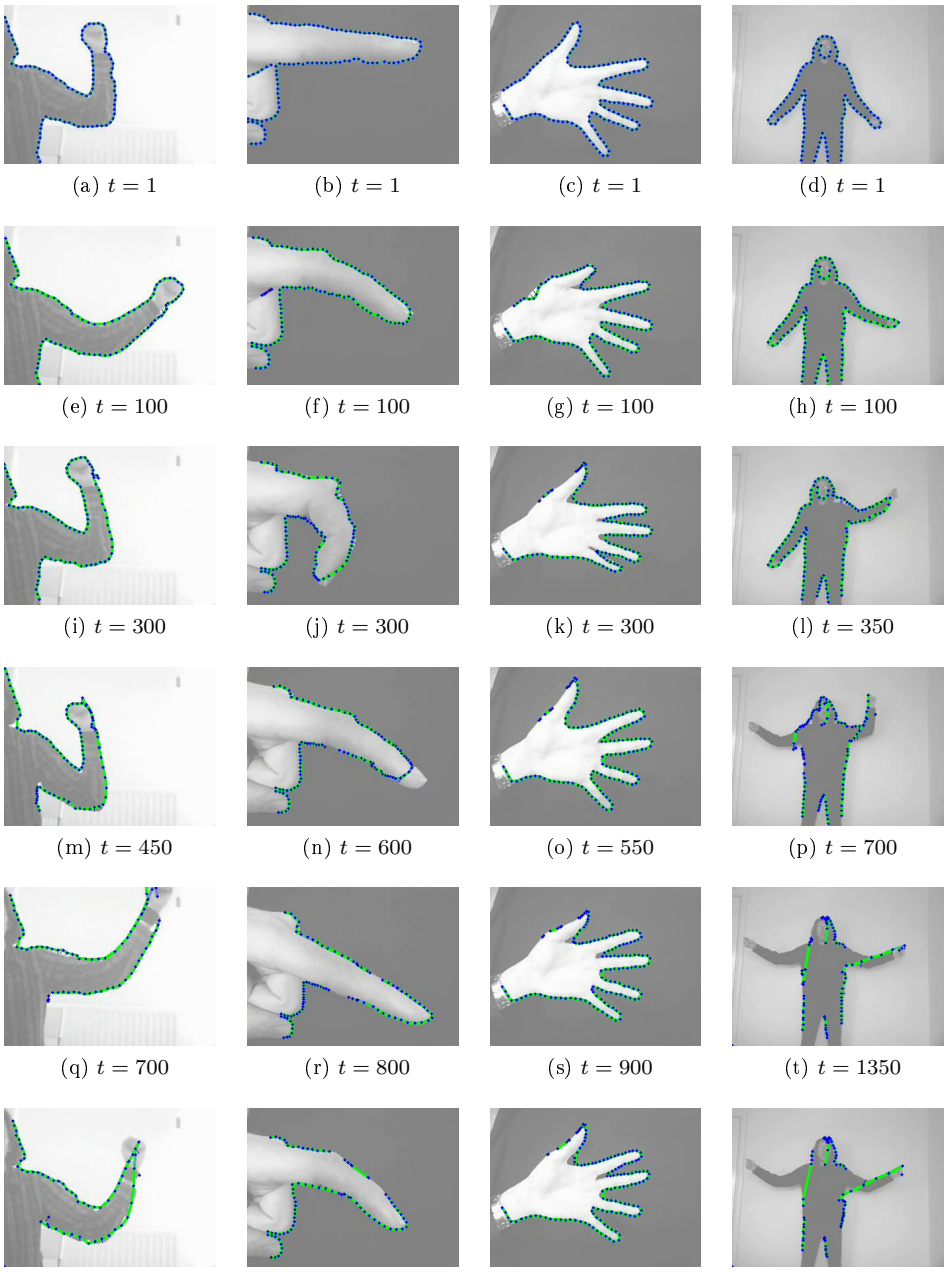
(i) $t = 300$ (j) $t = 300$ (k) $t = 300$ (l) $t = 350$

(m) $t = 450$ (n) $t = 600$ (o) $t = 550$ (p) $t = 700$

(q) $t = 700$ (r) $t = 800$ (s) $t = 900$ (t) $t = 1350$

(u) $t = 1150$ (v) $t = 1350$ (w) $t = 1200$ (x) $t = 1800$

Figure 5.2.1: Example of simultaneous learning and tracking of the feature level using an initialisation from the block level. The last row shows the resulting segmentation using the learnt relations and the segmentation method presented in Section 4.2.4.



(a) $t = 1$ (b) $t = 1$ (c) $t = 1$ (d) $t = 1$

(e) $t = 100$ (f) $t = 100$ (g) $t = 100$ (h) $t = 100$

(i) $t = 300$ (j) $t = 300$ (k) $t = 300$ (l) $t = 350$

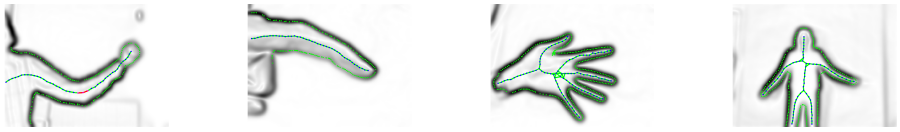
(m) $t = 450$ (n) $t = 600$ (o) $t = 550$ (p) $t = 700$

(q) $t = 700$ (r) $t = 800$ (s) $t = 900$ (t) $t = 1350$

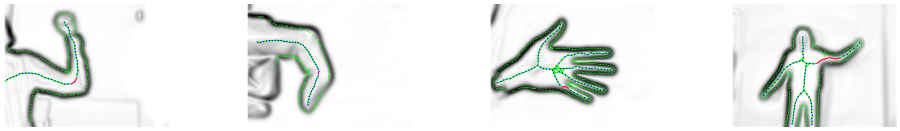
(u) $t = 1150$ (v) $t = 1350$ (w) $t = 1200$ (x) $t = 1800$

Figure 5.2.2: Examples of simultaneous learning and tracking of the feature level using only this level.

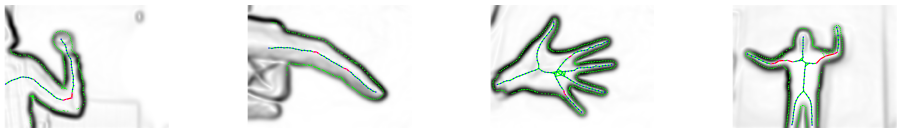
(a) $t = 1$ (b) $t = 1$ (c) $t = 1$ (d) $t = 1$



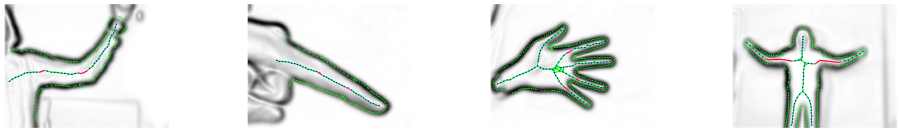
(e) $t = 100$ (f) $t = 100$ (g) $t = 100$ (h) $t = 100$



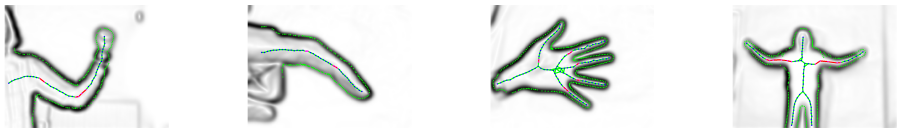
(i) $t = 300$ (j) $t = 300$ (k) $t = 300$ (l) $t = 350$



(m) $t = 450$ (n) $t = 600$ (o) $t = 550$ (p) $t = 700$



(q) $t = 700$ (r) $t = 800$ (s) $t = 900$ (t) $t = 1350$



(u) $t = 1150$ (v) $t = 1350$ (w) $t = 1200$ (x) $t = 1800$

Figure 5.3.1: Examples of simultaneous learning and tracking of the feature level using skeleton points instead of edgels. In this case, we obtain a good result without the help of a block level.

slide along the forearm without significant changes in its likelihood. In these circumstances, there is no pressure on the articulated model to adapt. Once the arm goes down (around frame 450), the block is able to recover. Since there is no clutter above and beside the forearm, the articulated model is finally forced to adapt until it becomes completely uninformative around frame 480.

This example is particularly interesting because it shows that, while the *uncertain* model serves its purpose in normal situation such as at the end of the sequence, it can still be problematic when clutter appears at the exact position where the block is expected. The obvious solution to this problem is to reduce the risk of clutter. A common way to do so is to represent an object using a combination of different descriptors. Given that the edgels act more as shape descriptors, we should combine them with an appearance descriptor. With computational time issues in mind, we decided to describe the appearance of each edgel using the colour value (or depth value depending on the type of image) of a blob on each side of the contour (see Figure 5.4.2). By blurring the image beforehand, this description reduces to the measure of two colour levels per edgel, keeping the additional cost to a minimum.

One of the advantage of the edgels is that they act as a shape descriptor, making them perfect candidates to represent objects that can vary in appearance (such as change of clothes or comparison between a grey image and a depth image). Since this is an attribute that we do not want to lose, we propose to initialise the appearance descriptor at the detection of a block (it is not stored between videos) and to update it at every frame using a simple update with drift correction similar to what Matthews and al. used for image templates tracked with Lukas-Kanade [59]) :

$$\text{if } \|T_{t-1} - I_t(\mathbf{x}_t)\| \leq \Delta_T \text{ then } T_{t-1} = I_t(\mathbf{x}_t) \tag{5.4.1}$$

$$else \qquad T_{t-1} = T_t\,, \tag{5.4.2}$$

where T_t represents the expected appearance and $I_t(\mathbf{x}_t)$ the observed appearance at time t. This way, the appearance model only adapts to smooth variations in luminosity (or depth) but is more robust against occlusion or temporary tracking failure. This approach sufficiently reduces the clutter effect to allow the *uncertain* relation to evolve properly (see Figure 5.4.2). Notice that the appearance descriptor is only used at the block level. The reason for this is that it is only used for weighting the particles and not for template alignment (since a zone without gradient does not provide relevant information for alignment, the appearance descriptor is only useful if the block is already close to its correct position). While more complex descriptors can also be added to further increase the robustness of the model, we decided to stop here to favour low computational time and to focus on a shape-oriented model. On a different (or more specific) context, a different appearance descriptor might provide a better solution.

Since it reduces the clutter, the addition of an appearance model is also beneficial outside the context of *uncertain* relations. We have demonstrated in Chapter 3 that the *Aligned Particle Filter* provides a robust tracking for

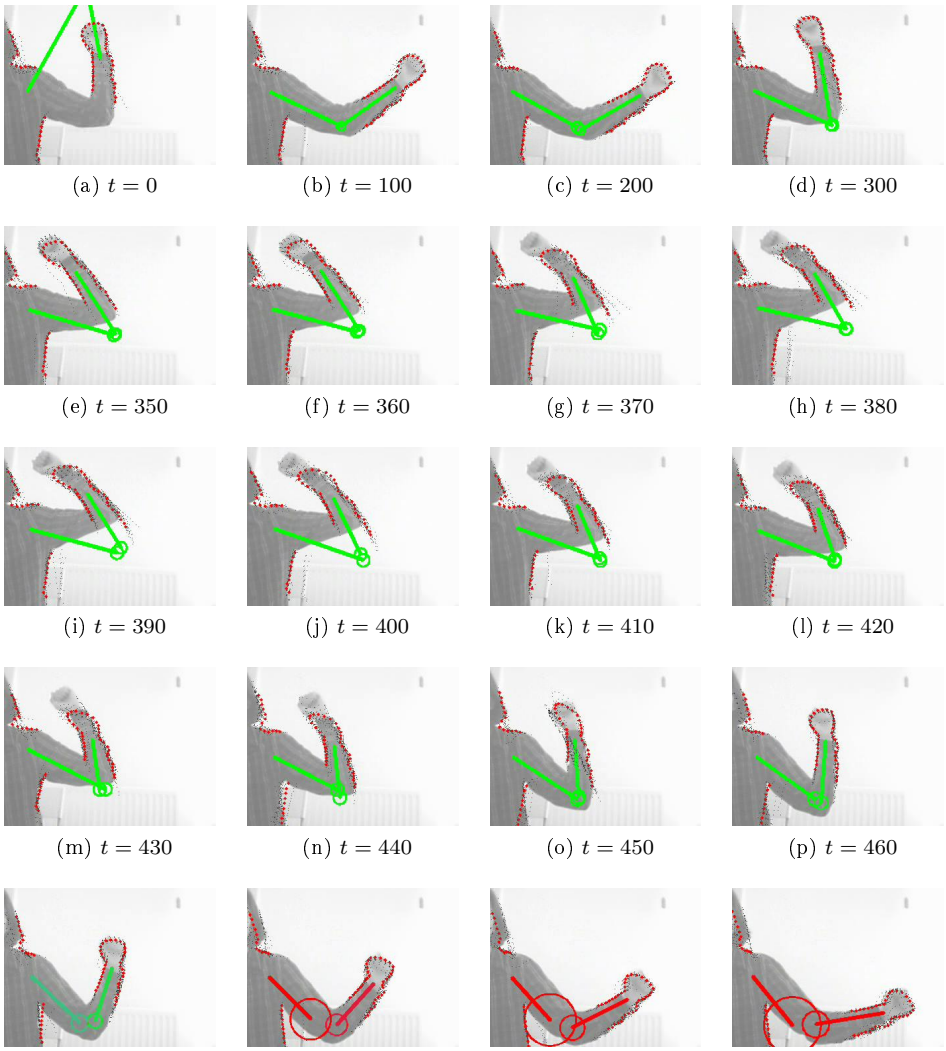
isolated blocks but there is still a specific situation for which they are at risk to fail: when most of the contours of the block are parallel. In this case, the block can collapse on a single line and still get a high image likelihood. This result was already observed in Figures 3.3.9 and 3.3.10 of Chapter 3 for the classical particle filters. It was avoided by the *Aligned Particle Filter* because a slightly better position was always found thanks to the few perpendicular edgels. For more challenging situations, those few edgels might not be enough to prevent the tracking from failing. An appearance model is very useful in this case since the appearance along the line is unlikely to be similar to the expected appearance. In Figure 5.4.3, we illustrate this advantage by tracking a hand with non-rigid movements.

5.5 Discussion

In this chapter, we focused on combining learning and tracking but alos the two levels of the graph. Given that the learning methods were developed with simultaneous learning and tracking in mind, combining the two was pretty straightforward. The only serious problem we had was tracking the blocks under the constraint of incorrect relations using only contours. We showed that the use of an appearance model for each edgel was sufficient to obtain a robust tracking under these circumstances.

We also demonstrated that the use of other features such as skeleton points would provide a more robust result for simultaneous learning and tracking at the cost of a more difficult initialisation process. This result leaves open the discussion about the best choice of features given the context and level of control over the environment.

The main contribution of this chapter is nevertheless the adaptation of the classical particle filter in order to track articulated objects using a rigid template. By solving this problem, we were able to provide a robust initialisation for the tracking of the feature level.



(a) $t = 0$ (b) $t = 100$ (c) $t = 200$ (d) $t = 300$

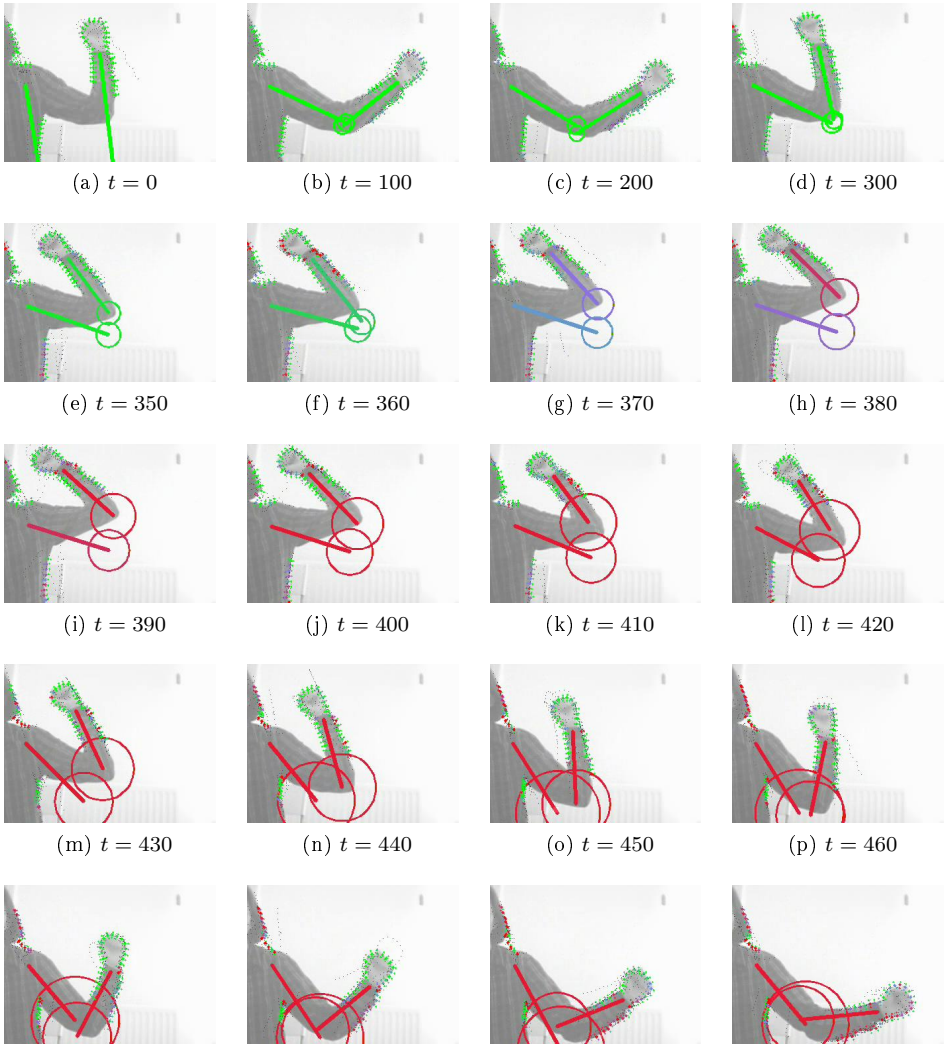
(e) $t = 350$ (f) $t = 360$ (g) $t = 370$ (h) $t = 380$

(i) $t = 390$ (j) $t = 400$ (k) $t = 410$ (l) $t = 420$

(m) $t = 430$ (n) $t = 440$ (o) $t = 450$ (p) $t = 460$

(q) $t = 470$ (r) $t = 480$ (s) $t = 490$ (t) $t = 500$

Figure 5.4.1: Tracking failure due to a combination of articulated constraints and clutter.



(a) $t = 0$ (b) $t = 100$ (c) $t = 200$ (d) $t = 300$

(e) $t = 350$ (f) $t = 360$ (g) $t = 370$ (h) $t = 380$

(i) $t = 390$ (j) $t = 400$ (k) $t = 410$ (l) $t = 420$

(m) $t = 430$ (n) $t = 440$ (o) $t = 450$ (p) $t = 460$

(q) $t = 470$ (r) $t = 480$ (s) $t = 490$ (t) $t = 500$

Figure 5.4.2: Adding an appearance model allows us to reduce the clutter effect observed in Figure 5.4.1. Colours are used to represent the likelihood of the edgels. The positions where the appearance is measured are also represented with the same colour code.

(a) $t = 50$ (b) $t = 100$ (c) $t = 150$ (d) $t = 200$

(e) $t = 250$ (f) $t = 300$ (g) $t = 350$ (h) $t = 400$

(i) $t = 50$ (j) $t = 100$ (k) $t = 150$ (l) $t = 200$

(m) $t = 250$ (n) $t = 300$ (o) $t = 350$ (p) $t = 400$

(q) $t = 50$ (r) $t = 100$ (s) $t = 150$ (t) $t = 200$

(u) $t = 250$ (v) $t = 300$ (w) $t = 350$ (x) $t = 400$

Figure 5.4.3: Non-rigid hand tracking without and with the addition of an appearance model (respectively rows 3 and 4 and rows 5 and 6). Colours are used to represent the likelihood of the edgels. The positions where the appearance is measured are also represented with the same colour code.

Chapter 6

Conclusion

This research has always used, as a long-term goal, the creation of a completely unsupervised system able to build its own database of complex objects in real-time. While there is still a significant amount of work to do, we believe that this thesis has laid a few stepping stones in the correct direction.

6.1 Contributions

In Chapter 2, we introduced a two-level graphical model suitable for both tracking and learning. The feature level can be seen as the local level, using the feature graph model to provide the flexibility required by the unsupervised learning. The block level can be seen as the global level, using the part-based articulated model to achieve a robust tracking. As both of these models are based on a graphical representation, they are naturally combined into a consistent global model where the two levels can assist each other. Thanks to the use of edgels and parametric spatial relations, the model requires minimal computational time while being effective on a wide variety of complex objects.

In Chapter 4, we presented our solutions to track both levels in real-time. The *Affine Warp Propagation* method provides a new framework for efficient propagation of alignment information through a feature-point graph. Instead of propagating potential functions as is usually done, it propagates only the motion information needed to align feature points and their surroundings in the image. We showed that this solution allows us to track articulated objects with a large number of feature points in a few milliseconds per frame (on a laptop with an Intel Core 2 T7200 processor clocked at 2GHz).

For the block level, we decided to use the more conventional Nonparametric Belief Propagation but increased its tracking speed by combining the classical particle filter with a gradient descent approach. With this solution, we were able to obtain robust tracking with as few as five particles while the classical particle filter usually requires hundreds of them. To summarise, this chapter presented methods able to track both levels in real-time in a way that corresponds to the

requirements of each of them: flexibility for the feature level and robustness for the block level.

In Chapter 4, we introduced the concept of *uncertain* model. This type of model allows simple and fast parametric models to assist tracking during their learning phase without exerting overly strong, counterproductive bias if these models are inappropriate. The uncertain model was successfully derived into a rigid and an articulated model used respectively in the feature level and the block level.

We also created a more versatile *Uncertain Gaussian Mixture Model* for incrementally learning a Gaussian mixture model based on a new criterion for splitting and merging mixture components. This criterion depends on a single user-settable parameter that allows easy tuning of the trade-off between the complexity and the accuracy of the mixture model. Our two-level approach provides a solution to the overfitting problem of small data sets without any compromise on the model accuracy. As more data arrives, the mixture complexity can be increased without any propagation of errors due to a previously underfitted model. As we have demonstrated empirically, this method is nearly independent of the order in which the data are observed.

The uncertain relations learnt at the feature level also provide an easy and fast starting point to segment the features into rigid blocks. By combining them with our solutions to segment and maintain group of features over time, we were able to create a robust and fast method for on-line discovery of rigid blocks. Both learning and segmentation algorithms presented in this chapter require a computational time negligible compared to that for tracking, leaving as much time as possible for the latter.

Finally, in Chapter 5, we presented the final modifications required to further improve the communications between the two levels of the graphical model on one hand, and between learning and tracking on the other. We first created an adapted version of the particle filter that uses particle sets to properly follow all the rigid parts of an object without needing an appropriate articulated model. We also defined the information exchanged between the feature level and the block level so that they can bring flexibility and robustness to each other without causing each other to fail.

We ended this chapter by showing the impact of using an appearance model for each edgel or even using a completely different feature such as skeleton points. These results leave open the discussion about the best choice of features given the available computational time, the required robustness, and the level of control over the environment.

6.2 Suggestions for Future Research

All the videos used in this thesis have been generated at the very beginning of the research. At that time, these videos were expected to represent a very mild challenge since they appear extremely simple to the human eyes. Surprisingly, they presented unexpected difficulties for almost every aspect of our project and were therefore used until the end. This demonstrates again the huge gap existing between our visual system and the computer vision algorithms. One of the most important challenges in the future will be to reduce that gap using increasingly difficult videos.

It is expected that the early discovery of a new object will always be a critical phase of tracking. It thus makes sense to require a simpler context for this early stage. Alternatively, it is also expected that tracking will become increasingly robust with the accumulation of knowledge on a particular object. It would be interesting in future research to analyse the robustness of tracking as a function of the knowledge accumulated in the learnt model. Ideally, the results obtained will prove that a model learnt with our method can achieve the same level of robustness as a model designed beforehand.

Since our focus was on the early stage of an object discovery, we also completely neglected the management of models over the long term. In the future, it will be necessary to address the problem of object recognition to deal with regular manifestations of objects. Not only will it be useful for robust tracking of objects from the (nearly) first frame but it will also be mandatory for any task in robotics or human-computer interaction.

While we focused our efforts on 2D models as a proof of concept, 3D models will most certainly be necessary for real-life applications (considering, for example, a robot with a hand). With the appearance of cheap 3D cameras with fast and robust depth map extraction (such as the *Kinect* from Microsoft), research on 3D models should become more accessible in a relatively near future. Although more time would be necessary to validate this idea, it seems that skeleton points would become a more desirable visual feature than in 2D. A 3D model using the duality between skeleton and contours would probably be a good model to explore.

Bibliography

[1] Nagesh Adluru and Longin Jan Latecki. Contour Grouping Based on Contour-Skeleton Duality. *Int. J. Comput. Vision*, 83:12–29, 2009.

[2] O. Arandjelovic and R. Cipolla. Incremental learning of temporally-coherent Gaussian mixture models, 2006.

[3] Xiang Bai, Xinggang Wang, Longin Jan Latecki, Wenyu Liu, and Zhuowen Tu. Active skeleton for non-rigid object detection. In *ICCV*, pages 575–582, 2009.

[4] Simon Baker and Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. *Int. J. Comput. Vision*, 56(3):221–255, 2004.

[5] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and chamfer matching: two new techniques for image matching. In *IJCAI'77*, pages 659–663, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[6] Herbert Bay, Tinne Tuytelaars, Van Gool, and L. SURF: Speeded Up Robust Features. In *9th European Conference on Computer Vision*, 2006.

[7] Olivier Bernier, Pascal Cheung-Mon-Chan, and Arnaud Bouguet. Fast Nonparametric Belief Propagation for real-time stereo articulated body tracking. *Comput. Vis. Image Underst.*, 113(1):29–47, 2009.

[8] I. Biederman. Surface versus edge-based determinants of visual recognition. *Cognitive Psychology*, 20(1):38–64, 1988.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[10] Andrew Blake and M. Isard. *Active Contours: The Application of Techniques from Graphics,Vision,Control Theory and Statistics to Visual Tracking of Shapes in Motion*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1998.

[11] M. Briers, A. Doucet, and S. S. Singh. Sequential Auxiliary Particle Belief Propagation. In *Information Fusion, 2005 8th Int. Conf. on*, volume 1, pages 705–711, 2005.

[12] Aurélie Bugeau and Patrick Pérez. Detection and segmentation of moving objects in complex scenes. *Comput. Vis. Image Underst.*, 113(4):459–476, April 2009.

[13] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[14] Roland T. Chin, Hong-Khoon Wan, D. L. Strover, and R. D. Iverson. A one-pass thinning algorithm and its parallel implementation. *Comput. Vision Graph. Image Process.*, 40:30–40, 1987.

[15] Peter Clifford. Markov Random Fields in Statistics. *Disorder in Physical Systems. A Volume in Honour of John M. Hammersley*, pages 19–32, 1990.

[16] D. Comaniciu and P. Meer. Mean Shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.

[17] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-Based Object Tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(5):564–575, 2003.

[18] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active Appearance Models. *Proceedings of the European Conference on Computer Vision*, 2:484–498, 1998.

[19] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active Shape Models - Their Training and Application. *Computer Vision and Image Understanding*, 61(1):38–59, January 1995.

[20] João P. Costeira and Takeo Kanade. A Multibody Factorization Method for Independently Moving Objects. *International Journal of Computer Vision*, 29(3):159–179, September 1998.

[21] Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection. In *International Conference on Computer Vision & Pattern Recognition*, volume 2, pages 886–893, 2005.

[22] Ankur Datta, Yaser Ajmal Sheikh, and Takeo Kanade. Linear Motion Estimation for Systems of Articulated Planes. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2008.

[23] Joeri De Winter and Johan Wagemans. Contour-based object identification and segmentation: Stimuli, norms and data, and software tools. *Behavior Research Methods*, 36:604–624, 2004.

[24] Arnaud Declercq and Justus Piater. Affine warp propagation for fast simultaneous modelling and tracking of articulated objects. In *Proceedings of the 10th Asian conference on Computer vision - Volume Part III*, ACCV'10, pages 422–435. Springer-Verlag, 2011.

[25] Arnaud Declercq and Justus H. Piater. On-line Simultaneous Learning and Tracking of Visual Feature Graphs. *Online Learning for Classification Workshop, CVPR'07*, 2007.

[26] Arnaud Declercq and Justus H. Piater. Online Learning of Gaussian Mixture Models - a Two-Level Approach. In *3rd International Conference on Computer Vision Theory and Applications (VISAPP)*, pages 605–611, 2008.

[27] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B*, 39(1):1–38, 1977.

[28] Arnaud Doucet and Adam M. Johansen. A tutorial on particle filtering and smoothing: fifteen years later. In *In Handbook of Nonlinear Filtering (eds.* University Press, 2009.

[29] N.D.H. Dowson and R. Bowden. N-tier Simultaneous Modelling and Tracking for Arbitrary Warps. page II:569, 2006.

[30] Stephane Drouin, Patrick Hebert, and Marc Parizeau. Incremental Discovery of Object Parts in Video Sequences. In *Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 1754–1761, Washington, DC, USA, 2005. IEEE Computer Society.

[31] H. Durrant-Whyte and T. Bailey. Simultaneous Localization and Mapping: Part 1. *Robotics and Automation Magazine, IEEE*, 13:99–110, 2006.

[32] H. Durrant-Whyte and T. Bailey. Simultaneous Localization and Mapping (SLAM): Part 2. *Robotics and Automation Magazine, IEEE*, 13:108–117, 2006.

[33] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, 2003.

[34] Brendan J. J. Frey and Delbert Dueck. Clustering by Passing Messages Between Data Points. *Science*, 315(5814):972–976, February 2007.

[35] D M. Gavrila. Multi-Feature Hierarchical Template Matching Using Distance Transforms. In *Proceedings of the 14th International Conference on Pattern Recognition-Volume 1 - Volume 1*, ICPR '98, pages 439–, Washington, DC, USA, 1998. IEEE Computer Society.

[36] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

[37] A. Gruber and Y. Weiss. Multibody factorization with uncertainty and missing data using the EM algorithm. volume 1, pages I–707–I–714 Vol.1, 2004.

[38] P.M. Hall and Y. A. Hicks. A method to add Gaussian mixture models. *Tech. Report, University of Bath*, 2005.

[39] Tony X. Han, Huazhong Ning, and Thomas S. Huang. Efficient Nonparametric Belief Propagation with Application to Articulated Body Tracking. In *CVPR'06*, pages 214–221, 2006.

[40] C. Harris and M. Stephens. A Combined Corner and Edge Detection. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, 1988.

[41] M. Sabry Hassouna and Aly A. Farag. On the Extraction of Curve Skeletons using Gradient Vector Flow. *Computer Vision, IEEE International Conference on*, pages 1–8, 2007.

[42] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Springer, 2 edition, 2009.

[43] Andreas Hofhauser, Carsten Steger, and Nassir Navab. Edge-Based Template Matching and Tracking for Perspectively Distorted Planar Objects. In *Proceedings of the 4th International Symposium on Advances in Visual Computing*, ISVC '08, pages 35–44, Berlin, Heidelberg, 2008. Springer-Verlag.

[44] M. Julius Hossain, M. Ali Akber Dewan, and Oksam Chae. Edge Segment-Based Automatic Video Surveillance. *EURASIP J. Adv. Sig. Proc.*, 2008, 2008.

[45] Gang Hua and Ying Wu. Multi-Scale Visual Tracking by Sequential Belief Propagation. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04)*, pages 826–833, 2004.

[46] Alexander Ihler, Er T. Ihler, Erik Sudderth, William Freeman, and Alan Willsky. Efficient Multiscale Sampling from Products Of Gaussian Mixtures. In *In NIPS 17*, page 2003. MIT Press, 2003.

[47] Michael Isard and Andrew Blake. CONDENSATION – Conditional Density Propagation for Visual Tracking. *International Journal of Computer Vision*, 29(1):5–28, August 1998.

[48] Michael I. Jordan, editor. *Learning in graphical models*. MIT Press, Cambridge, MA, USA, 1999.

[49] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[50] Dov Katz and Oliver Brock. Manipulating Articulated Objects with Interactive Perception. In *Proceedings of the International Conference on Advanced Robotics*, pages 272–277, Pasadena, CA, May 19-23 2008.

[51] N. Krahnstoever, M. Yeasin, and R. Sharma. Automatic acquisition and initialization of articulated models. *Mach. Vision Appl.*, 14(4):218–228, 2003.

[52] Marius Leordeanu and Robert Collins. Unsupervised Learning of Object Features from Video Sequences. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 1142–1149. IEEE Computer Society, 2005.

[53] Stan Z. Li. *Markov Random Field Modeling in Image Analysis*. Springer Publishing Company, Incorporated, 2009.

[54] Jongwoo Lim, David A. Ross, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental Learning for Visual Tracking. In L. Saul, Y. Weiss, and L. Bottou, editors, *NIPS'05*, pages 801–808, 2005.

[55] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[56] 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*, pages 674–679. Morgan Kaufmann Publishers Inc., 1981.

[57] E. Maggio and A. Cavallaro. Hybrid PArticle Filter and Mean Shift tracker with adaptive transition model. In *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Philadelphia, PA, USA, 19–23 March 2005.

[58] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *In British Machine Vision Conference*, volume 1, pages 384–393, 2002.

[59] Iain Matthews, Takahiro Ishikawa, and Simon Baker. The Template Update Problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(6):810–815, 2004.

[60] K. Mikolajczyk and C. Schmid. An Affine Invariant Interest Point Detector. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I*, pages 128–142, London, UK, 2002. Springer-Verlag.

[61] R. L. Ogniewicz and O. Kübler. Hierarchic Voronoi skeletons. *Pattern Recognition*, 28(3):343–359, 1995.

[62] Minwoo Park, Yanxi Liu, and Robert T. Collins. Efficient Man Shift Belief Propagation for vision tracking. *CVPR'08*, 0:1–8, 2008.

[63] M. Pawan Kumar, P. H. Torr, and A. Zisserman. Learning Layered Motion Segmentations of Video. *Int. J. Comput. Vision*, 76:301–319, March 2008.

[64] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, September 1988.

[65] Shrinivas Pundlik and Stanley T. Birchfield. Motion Segmentation At Any Speed. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 427–436, September 2006.

[66] Deva Ramanan, David A. Forsyth, and Kobus Barnard. Building Models of Animals from Video. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(8):1319–1334, 2006.

[67] Deva Ramanan, David A. Forsyth, and Andrew Zisserman. Tracking People by Learning Their Appearance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29:65–81, January 2007.

[68] J. Rissanen. Modeling by shortest data description. *Automatica, vol. 14, pp. 465-471*, 1978.

[69] David A. Ross, Daniel Tarlow, and Richard S. Zemel. Unsupervised Learning of Skeletons from Motion. In *Proceedings of the 10th European Conference on Computer Vision*, pages 560–573, Berlin, Heidelberg, 2008. Springer-Verlag.

[70] Angel D. Sappa and Fadi Dornaika. An edge-based approach to motion detection. In *Proceedings of the 6th international conference on Computational Science - Volume Part I*, ICCS'06, pages 563–570, Berlin, Heidelberg, 2006. Springer-Verlag.

[71] Caifeng Shan, Yucheng Wei, Tieniu Tan, and Frédéric Ojardias. Real time hand tracking by combining particle filtering and Mean Shift. In *Proceedings of the Sixth IEEE International Conference on Automatic Face and Gesture Recognition*, FGR' 04, pages 669–674, Washington, DC, USA, 2004. IEEE Computer Society.

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

[73] Jamie Shotton, Andrew Blake, and Roberto Cipolla. Multiscale Categorical Object Recognition Using Contour Fragments. *PAMI'08*, 30(7):1270–1281, 2008.

[74] Leonid Sigal, Ying Zhu, Dorin Comaniciu, and Michael Black. Tracking Complex Objects Using Graphical Object Models. In *Proc. International Workshop on Complex Motion*, 2005.

[75] M. Song and H. Wang. Highly Efficient Incremental Estimation of Gaussian Mixture Models for Online Data Stream Clustering. *Intelligent Computing: Theory and Application*, 2005.

[76] Carsten Steger. Occlusion, Clutter, and Illumination Invariant Object Recognition. In *Photogrammetric Computer Vision ISPRS Commission III*, 2002.

[77] E. B. Sudderth, A. T. Ihler, W. T. Freeman, and A. S. Willsky. Nonparametric Belief Propagation. In *CVPR'03*, volume 1, pages I–605–I–612 vol.1, 2003.

[78] Erik B. Sudderth, Michael I. Mandel, William T. Freeman, and Alan S. Willsky. Visual Hand Tracking Using Nonparametric Belief Propagation. In *CVPRW'04 - Volume 12*, Washington, DC, USA, 2004. IEEE Computer Society.

[79] Feng Tang and Hai Tao. Object tracking with dynamic feature graph. In *Proceedings of the 14th International Conference on Computer Communications and Networks*, pages 25–32, Washington, DC, USA, 2005. IEEE Computer Society.

[80] J. Toriwaki and S. Yokoi. Distance transformations and skeletons of digitized pictures with applications. *Progress in Pattern Recognition*, pages 187–264, 1981.

[81] Nhon H. Trinh and Benjamin B. Kimia. A Symmetry-Based Generative Model for Shape. *Computer Vision, IEEE International Conference on*, pages 1–8, 2007.

[82] Markus Weber, Max Welling, and Pietro Perona. Unsupervised Learning of Models for Recognition. In *Proceedings of the 6th European Conference on Computer Vision-Part I*, ECCV '00, pages 18–32, London, UK, 2000. Springer-Verlag.

[83] Yair Weiss. Segmentation Using Eigenvectors: A Unifying View. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 975–, Washington, DC, USA, 1999. IEEE Computer Society.

[84] Yair Weiss and William T. Freeman. Correctness of Belief Propagation in Gaussian Graphical Models of Arbitrary Topology. *Neural Comput.*, 13:2173–2200, 2001.

[85] Wikipedia. Clique (graph theory) — wikipedia, the free encyclopedia, 2012. http://en.wikipedia.org/wiki/Clique_(graph_theory).

[86] Wikipedia. Graphical model — wikipedia, the free encyclopedia, 2012. http://en.wikipedia.org/wiki/Graphical_model.

[87] Ying Wu, Gang Hua, and Ting Yu. Tracking Articulated Body by Dynamic Markov Network. In *ICCV'03*, page 1094, Washington, DC, USA, 2003. IEEE Computer Society.

[88] Jingyu Yan and Marc Pollefeys. Articulated Motion Segmentation Using RANSAC with Priors. In *WDV*, pages 75–85, 2006.

[89] Jingyu Yan and Marc Pollefeys. Automatic Kinematic Chain Building from Feature Trajectories of Articulated Objects. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 712–719, Washington, DC, USA, 2006. IEEE Computer Society.

[90] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Understanding Belief Propagation and its generalizations. In *Exploring artificial intelligence in the new millennium*, pages 239–269, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[91] Z.Z. Yin and R. Collins. On-the-fly Object Modeling while Tracking. In *Proceedings of the 2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'07)*, pages 1–8. IEEE Computer Society, 2007.

[92] L. Zelnik-Manor and M. Irani. Degeneracies, dependencies, and their implications in multi-body and multi-sequence factorizations. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–287–93 vol.2, 2003.

[93] Zoran Zivkovic and Ferdinand van der Heijden. Recursive Unsupervised Learning of Finite Mixture Models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26:651–656, May 2004.