

REAL-TIME MULTI-VIEW HUMAN
BODY TRACKING USING 3D VOXEL
RECONSTRUCTION AND
HIGH-LEVEL MOVEMENT
PREDICTION.

A TRANSFER REPORT SUBMITTED TO THE UNIVERSITY OF MANCHESTER
IN THE FACULTY OF SCIENCE AND ENGINEERING

August 2002

By
Fabrice CAILLETTE
Department of Computer Science

Contents

Abstract	5
Declaration	6
Copyright	7
1 Introduction	8
2 Background	11
2.1 Image based tracking	11
2.2 Visual Hull	12
2.2.1 Capture	12
2.2.2 Silhouette extraction	13
2.2.3 3D projection	14
2.3 Model	15
2.4 Inverse Kinematics	17
2.4.1 Problem formulation	17
2.4.2 First method: Jacobian Inverse	18
2.4.3 Second Method: CCD	19
2.5 Kalman filter	19
2.6 Recognition	23
3 Problem formulation and approach overview	27
3.1 Problem description	27
3.2 Proposed approach	29
4 Data processing and tracking	31
4.1 Capture and pre-processing	31
4.2 Visual Hull	32
4.2.1 Design choices	33
4.2.2 Implementation	34
4.3 Model	35
4.3.1 The kinematic model	35
4.3.2 Inverse kinematics	37

4.3.3	Matching	38
4.4	Conclusion	41
5	Motion prediction	42
5.1	The Kalman Filter	42
5.2	Image-based features tracking	44
5.3	Movement recognition and prediction	44
5.4	conclusion	45
6	Conclusion and future work	46
6.1	Summary of work currently done	46
6.2	Future works	46
6.3	Work schedule	47
	Bibliography	48

List of Figures

2.1	The visual hull is defined by the intersection of the generalized cones extruded from the silhouettes (Image reproduced from [MBR ⁺ 00])	13
2.2	Model-based tracking (adapted from O'Rourke and Badler [OB80])	15
2.3	(left) Standard human joints and bones hierarchy as defined by the MPEG-4 standard. In practice, this representation is too complex for real-time tracking, so a simplified model (right) is preferred	16
2.4	Local optimization in the CCD method.	19
2.5	A typical Finite State Machine as used in [HTH00].	24
2.6	(a) In a classical HMM, the probability of being in a given state depends only on the previous state. A state represents all the processes. (b) In a linked HMM, the processes are separated and the output is a joint probability of the state of the other processes. (c) The CHMMs states depend on the previous states of all the processes.	25
3.1	Functional overview of the whole tracking system.	29
4.1	Silhouette extraction: The upper row shows the images of the avatar as seen by the 3 cameras (resolution of 128x128), and the bottom row contains the corresponding silhouettes obtained by thresholding.	33
4.2	The same visual-hull construction from two different viewpoints. Voxels are represented by points. A voxel-space of size 128x128x128 is used here, with 3 cameras. The inside voxels have also been removed.	36
4.3	Each voxel exerts an attraction force to the nearest bone, making the model converge toward the data	39
4.4	A given voxel exerts on each bone an attraction force which depends on the distance between this voxel and the bone.	40
4.5	Functional description of the tracking process, without motion prediction.	41
5.1	Functional diagram of the motion prediction process.	45
6.1	Work schedule for the rest of the PhD	47

Abstract

Human body tracking has been an intensive subject of research for the last 15 years. Many exciting applications are at stake in the domain of computer vision and human-computer interfaces. Recently, some basic attempts to track people from a multi camera-view reconstruction have been carried out. The aim of this thesis is to conceive and implement a framework extending and trying to improve this type of 3D-based tracking with state of the art techniques. The system presented here can be decomposed into two parts. The first one is the proper tracking: A kinematic model is matched to the 3D voxels obtained by a shape-from-silhouette reconstruction. This process is self-sufficient for tracking a simple range of movements, but the need for more robustness leads to the addition of a motion prediction module, which constitutes the second part of the system. An extended Kalman filter is the central point of the motion-prediction process. Its measurement-input is the positions of the hands and head obtained by 2D feature tracking. The Kalman filter also incorporates the physics-based prediction of a dynamic model and the high-level indications of a movement recognition module. This last module of behaviours recognition and prediction is probably the most challenging and even if it is not yet fully designed, it will be based on hidden Markov models, or one of its variants.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Copyright

Copyright in text of this thesis rests with the Author. Copies (by any process) either in full, or of extracts, may be made **only** in accordance with instructions given by the Author and lodged in the John Rylands University Library of Manchester. Details may be obtained from the Librarian. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the Author.

The ownership of any intellectual property rights which may be described in this thesis is vested in the University of Manchester, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.

Further information on the conditions under which disclosures and exploitation may take place is available from the head of Department of Computer Science.

Chapter 1

Introduction

Tracking objects and especially human beings is still a very open and challenging task in computer vision and virtual reality. Some techniques are already operational, but most of them rely on hardware markers or other intrusive devices to identify the position of the various parts of the body. Since these markers represent a constraint, a very popular goal in machine vision is to track human beings exclusively from the images captured by one or more standard cameras. Of course, using only visual information without reliable markers is a very difficult problem and that is why no perfect solution has yet been found despite the many attempts that have been carried out for the last 15 years.

Tracking human motion is an exciting field of research, but also a practical issue because of all the possible induced applications. A reliable and accurate tracking would be for example one of the basis of human-computer interaction via movement recognition. A new type of interface for virtual-reality applications could indeed be based on the body pose and gesture recognition, enabling users to manipulate virtual objects with bare hands, or to control precisely and intuitively a virtual avatar. Body and face tracking is consequently the first step toward tele-presence via the use of photo-realistic avatars. It is then perfectly possible to imagine a meeting held in a virtual environment, each of the participant remaining in his office, but being immersed with 3D glasses in this virtual environment. The other side applications of tele-presence include virtual classrooms, virtual products presentation and sale, virtual guided historic and cultural visits, etc. . . Many other annex applications are imaginable such as security monitoring, very-low-bandwidth video compression or smart rooms.

Movement recognition is usually independent from the tracking in the sense that it is used as a post-process to interpret the gestures. Just as much as human tracking, recognition is a very active field of research leading to a number of practical applications. Some attempts concerning the interpretation of human face emotions, or the automatic reading of the American Sign Language have been quite successful. Human-computer interfaces are the first to benefit from a proper gesture recognition, but almost all the applications cited in the context of body tracking could make a good use of it. Recognition is generally done by comparing the current pose to a learned database which is usually a statistic data structure. An extensive use of AI techniques and structures is also required in this domain, but it is also more and more the case for tracking. A review of current recognition methods can be found in the background section 2.6. One idea that will be developed in the rest of this report could be to

use movement recognition as a feedback for the tracking process improving its robustness. Indeed, understanding the current movement could help to predict accurately what should be the next one, giving an additional information to the tracker and helping the detection and recovery of errors.

Concerning the tracking process, the main difficulties arise from the ambiguities and the missing information in the views. Ambiguities are omnipresent in human-body tracking, mainly because of the complexity of the human body with regard to the poor information available from the camera views. Recognizing the head from a hand, or choosing one among several plausible poses are examples of problems that occur frequently. Missing information is mainly due to occlusions: Since the tracking is only done using camera views, some parts of the body can be hidden at some time from all of the views. The only possibility is then to guess the position of the missing parts of the body, using for example a dynamic model.

In practice, using only one camera view proved too unreliable mainly because of the occlusion problem. No depth information could also be extracted from one only view, which leads to too many plausible pose hypothesis for the same view. The current trend is then to use more than one camera (usually three or more) in a configuration such as most of the body can be seen at all time. From this frameworks, two main approaches have been explored. The first one is to firstly perform the tracking on each view individually, resulting in a set of hypothesis associated with each camera, and then confront the hypothesis from all the cameras to find the likeliest solution. The other alternative is to fuse the data from all the views into a structure containing all the relevant information, and then to perform the tracking on this representation which is hopefully as unambiguous as possible.

This last approach (prior fusion of the data) has been explored only very recently [CKBH00, LSL02], but not yet extensively. The common idea is to use a volumetric reconstruction called the *visual hull* as reference data for tracking. The visual hull is constructed by projection and intersection of the silhouettes at the different views as detailed in the section 2.2. The approach of Cheung and Kanade [CKBH00] is to fit ellipsoids with gaussian distribution on the volumetric data using an EM scheme. No model is used and the output is barely exploitable because it is highly inaccurate. The recent article from Luck, Small and Little [LSL02] describes a system using a kinematic model to fit the volumetric data, and claims encouraging results with exploitable outputs. Nonetheless, their system is very simple, and could probably gain in performance by using state of the art techniques like motion prediction or incorporation of a feature-based tracker.

The general idea that will be developed through this report is to use a volumetric reconstruction as basis for tracking, and to develop techniques to improve the reliability and efficiency of the tracking as well as the recognition of movements. Using a model is now unavoidable for reliability and further interpretation of the data. There seem to be a general consensus on the use of models: Almost no recent paper tries to track human body without the help of at least a kinematic model. The section 2.3 is a brief overview of the different types of models. For example, the use of a dynamic model with motion prediction would improve the reliability of the tracking, especially when the model fitting technique is based on an attraction-to-data scheme. The more common method to perform motion prediction while being as insensitive to the noise as possible is to use a Kalman filter: A general description and a mathematic formulation of the Kalman filter can be found in the section 2.5.

Feature-based tracking methods are not to neglect. There has been some spectacular improvements

in this domain over the past few years. The general principle is to track in 2D the position of a specific feature over a video sequence. For example, tracking the position of the hands on the different camera views can prove very useful as a starting point for the tracking of the rest of the arm. Some colour-based, or visual flow methods have been used, but the most successful one is CONDENSATION [IB98]. This algorithm was introduced in 1998 by Isard and Blake, and proved very reliable when tracking objects, even in difficult circumstances. The section 2.1 is a short review of some 2D feature-tracking methods, including the CONDENSATION algorithm.

When the position of some parts of the body is known (using feature-based tracking), it can be convenient to compute a first approximation of the global pose of the model. This approximation can be combined with the position predicted from the dynamic model to obtain an accurate prior estimation of the state of the model. It is important that this prior estimation is as close as possible to the real state because it reduces the risks of misclassifications and improves the performance of the proper tracking task. An estimation of the state of the model knowing only the position of some joints can be done using a well known animation technique called Inverse Kinematic (IK). Even if IK is widely used, the algorithms are not trivial especially when constraints have to be modeled. An overview of IK methods is presented in the section 2.4.

After the background chapter, a general overview of the system is presented (page 27), starting from a description of the problems and the corresponding constraints, and then attempting to address those. During this overview, the fact that tracking and motion prediction are two complementary but separate processes is highlighted, leading to the individual description of each. The chapter 4 is then consecrated to the only tracking procedure, which covers the acquisition of data, the silhouettes extraction and visual-hull construction, the kinematic model and inverse kinematics, and finally the model-matching process. Even if this part of the system is self contained and could give acceptable results, the benefits of a proper motion-prediction scheme are pointed out. The chapter 5 then focuses on motion-prediction. The central piece is a Kalman filter whose parameters are described concretely at the beginning of the chapter. The modules feeding information to the Kalman filter are then successively described.

The conclusion of this report provides a quick summary of the work currently done, and more extensively a scope of the work to be done. The axis of future development are outlined, and a work schedule gives an estimation of the scale of what is still to implement.

Chapter 2

Background

This chapter is mainly a literature review of the algorithms and techniques that will be used in the implementation of the tracking system. Most of these techniques are very general, but the description is always made from the point of view of tracking.

2.1 Image based tracking

This section describes briefly some of the most popular 2D feature-tracking techniques. Their common denominator is that they all work on a single image sequence (only one view at a time) and consequently cannot use any volumetric information like the one presented in the section 2.2. These techniques can nonetheless prove useful in our scheme a volumetric data tracking: Tracking a given feature in each of the camera views allow to find its 3D coordinates by projection (the cameras are supposed to be calibrated). This position can then be used to help the rest of the tracking process.

After proper segmentation of the foreground (see section 2.2.2), the Pfinder algorithm [WADP97] grow blobs in a spatio-colour space. This means that the position of each pixel (x, y) is associated with its colour to aggregate them by likelihood into gaussian clusters. In fact, to make the algorithm more robust, some blobs are also generated from the contour of the object, and these two classes of blobs are then blended together. This quite simple feature-tracking method proved surprisingly robust and efficient when used to drive a kinematic model.

Active shape models were developed in 1995 by Cootes *et al.* [CTCG95]. An active shape model is a set of splines (or snakes) which are constrained in their movements by a learned model, and which can converge toward some image features. Typically, the snakes will tend to align with the edges of the images. There are too many parameters to describe the movements of the whole shape, so a Principal Component Analysis is performed on the parameter space (using a prior training set), and only a few principle axis are kept. The movement of the whole shape can then be described by a restricted number of parameters, making a further recognition process much easier. An extension of the Active shape models was introduced in 1998 as active appearance models [CET98]: The shape then includes

some information about the texture patches in addition to the shape itself. Both these models have been primarily used for tracking and recognition of faces. They can handle small changes of parameters, but an utilization in human-body tracking would require a much bigger range of motion, which may prove difficult to handle.

Isard and Blake [IB96, IB98] developed an extension of the Kalman filter in order to deal with non linear predictions of movement. The algorithm, called CONDENSATION, uses sampling of distributions, Bayes rule, diffusion of distributions and reactive reinforcement to provide a robust estimate of the next position of an object. Both object properties and motion are used in the model. In order to apply it in practice, distributions have to be estimated. The shape of an object is modeled by splines. The distribution of possible shapes for the object we want to track has been collected from a set of manually annotated images. The motion has been estimated by using a Kalman filter to track the object until this tracking fails. The motion information collected can then be used with the CONDENSATION algorithm to track the object again. Improvements between the two ways of tracking have been observed by Isard and Blake. Eventually, new motion distribution can be derived by the second tracking and used in a third tracking to improve the result.

2.2 Visual Hull

Visual hull generation is also known in the computer vision literature as "shape-from-silhouettes". The basic idea is to take the views of a scene from different angles (usually at least 3 cameras are used), to extract the silhouette of the object of interest from each of these views and finally to construct a 3D representation of this object by intersecting the projection of the silhouettes (see figure 2.1 for a general overview). The main steps of this sequence which will be detailed in the following sections are:

1. the capture of the different views, including camera calibration and synchronization.
2. background removal, and silhouette extraction.
3. genesis of the 3D shape in an adequate data structure.

2.2.1 Capture

The main challenge here is to calibrate the cameras properly, reducing the distortions due to the imperfect optical lenses. A common way to calibrate the cameras is to capture a known pattern (such as a grid) and to compute the camera parameters such as the focal length, the centre pixel location or the coefficients of radial distortion (intrinsic parameters). Placing objects at known positions in the observed space and retrieving their 2D projection on the camera-images allow to compute the world-coordinate position and orientation of each of the cameras (extrinsic parameters).

The Open Source Computer Vision Library ¹ is a good starting point for camera-calibration algorithms. All the useful algorithms are already implemented, and are supposed to work pretty well.

¹<http://www.intel.com/research/mrl/research/opencv/>

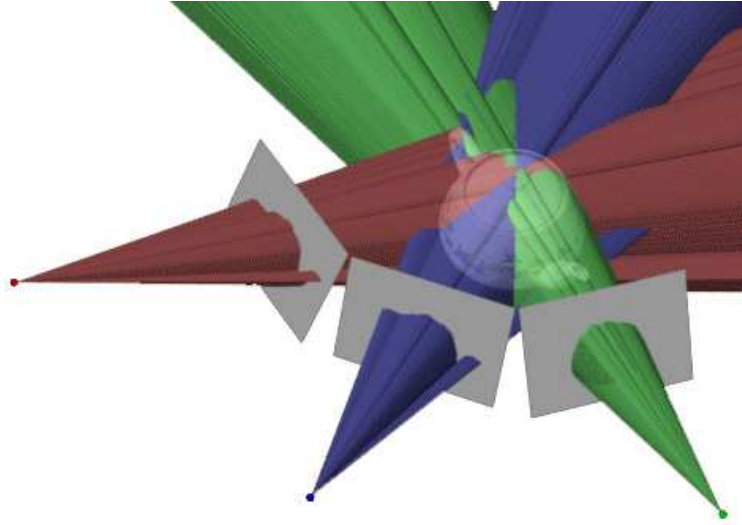


Figure 2.1: The visual hull is defined by the intersection of the generalized cones extruded from the silhouettes (Image reproduced from [MBR⁺00])

2.2.2 Silhouette extraction

The input from the previous section is an image which can be of variable resolution, colour depth and level of noise. The task here is to label each pixel of this image as belonging whether to the background or the object of interest: This task is far from trivial because the colour information of a pixel is often insufficient to label it correctly. The blue-screen technique is probably the simplest: The object or person of interest moves in front of a uniform background (usually blue) and the colour of the background is used to label the pixels. Of course a margin of tolerance has to be used for the noise and measurement errors.

Background subtraction is the other widely used technique which has the advantage of requiring no special equipment. Each camera is supposed to be fixed and so is the background: Taking a reference image of the background (without the object of interest) and subtracting it to the current image then zeroes all the background pixels. The pixels of the object get some values which hopefully are not too close to zero. Thresholding the resulting image gives a first labeling. Of course misclassifications can occur because of the noise, a change of illumination of the background or shadows.

To circumvent misclassifications due to a slow change of illumination of the background, a technique [WADP97, HS99] is to update at a time-step t each background pixel B of the reference background image using the colour of the same pixel in the current image I and a small coefficient α (often chosen arbitrarily) as follows:

$$B_{t+1} = \alpha I_t + (1 - \alpha)B_t \quad (2.1)$$

Of course more complex schemes have been implemented to handle noise and quick variations in the background: Most of them rely on statistical techniques, modeling for example each pixel by a mixture of Gaussians [SG99] or using simultaneously multiple background models [BMG⁺99].

An interesting alternative technique combining the information available from different views has

just been developed [TD02]: The idea is to label as foreground every pixel that has "free space" behind it. That is to say that if anything else can be seen behind the current pixel from any other view, then this pixel has to be part of a foreground object. Of course objects other than the one of interest can be detected by this technique, but there is always a way to get rid of them using masks.

Finally, once a first segmentation has been performed, a post-filter is usually applied to get rid of the obvious misclassifications. Considering that a silhouette is a connected set of points, all the isolated silhouette-pixels can be discarded. A closing kernel is also used to add the missing points inside the silhouette.

2.2.3 3D projection

Having the silhouettes and the calibration information for each camera, each point in a silhouette defines a ray in scene space that intersects the object at some unknown depth along this ray. The union of these visual rays for all the points in the silhouette of an image defines a generalized cone within which the 3D object must lie. Therefore, the intersection of all these generalized 3D cones associated with the cameras defines a volume in which the object is guaranteed to lie. Of course the volume only approximates the true 3D shape depending on the number of views, the position of the viewpoints, and the complexity of the object. In particular, since concave patches are not observable in any silhouette, a silhouette-based reconstruction encloses the true volume. The *visual hull* [Lau94] is defined as the best approximation of the object, obtained by using an infinite number of views: In practice, the approximation resulting from a small number of cameras is assimilated to the visual hull.

Now, an adequate volumetric representation of the scene has to be found. It should be precise enough to allow any further exploitation, and efficient for algorithms well as performances. Just as the silhouettes, the volumetric data-structure is usually binary: A point belongs to the object or not. All the representations also assume that the area of the scene is bounded by a cube. The most common and simple approach is then to use a regular tessellation of cubes called voxels. The basic algorithm is to project each voxel on each view, and to mark them as belonging to the object only if they belong to all the silhouettes. In practice things are not so simple because the projection of a voxel (a cube) onto a view-plane is a polygon which can contain a number of pixels depending on the relative resolution of the voxel space and the cameras: Testing all these pixels, for each camera view, and for each voxel is possible, but too expensive in practice. One easy solution is to project only the centre of each voxel, which is fast but highly inaccurate if the resolution of the voxel-space is low. Some efficient algorithms have also been developed [CKBH00], sampling uniformly the points in the projected polygon and using intensively lookup tables to avoid the cost of projections.

Octrees [CA86] is another spacial data structure that has often been used to generate visual hulls [Sze90]. This structure has the advantage of coding more efficiently the large portions of the scene having the same value. The method is to construct a coarse-to-fine hierarchy: Given an initial cube that encloses the entire scene, the current voxel is projected to each view: If the projections are all totally outside the silhouettes the voxel is labeled as belonging to the background, if the projections are all included in the silhouettes then the voxel is marked as belonging to the object. Otherwise, when the voxel intersects both background and silhouette points in the images, this voxel is subdivided into octants and each sub-voxel is processed recursively.

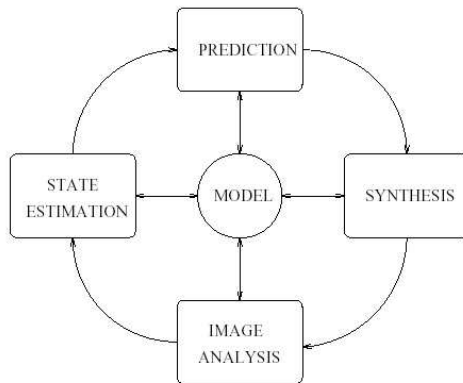


Figure 2.2: Model-based tracking (adapted from O'Rourke and Badler [OB80])

Another interesting representation is to model only the surface of the 3D object by a set of polygons [Chr01]. This approach does not utilize any type of grid, and has the advantage of being particularly adapted to rendering operations. The idea is to segment the contour of the silhouettes, which leads to a polyhedral cone when projecting the silhouette into the scene. Intersecting these cones, gives the proper polyhedral visual hull of the object. A polyhedral representation can also be computed from an octree or a voxel space using the Marching Cubes [LC87] algorithm, in which case an opacity value can be desired instead of the binary information (transparent or opaque).

2.3 Model

A model is not necessarily an explicit or exhaustive description of an object: It can rather be defined as any way susceptible to help the understanding of the static or dynamic properties of a determined object. A model is generally used to interpret brut data, in which case we talk of model-matching (or fitting) on the data. For example, in human-body tracking, a model describes the movements in a functional way (angles between limbs) whereas the brut data is inadequate to interpret the movement.

Of course, tracking can always be performed without any kind of model [IB96, IB98, CKBH00] (see section 2.1 for details), but these methods are generally not very robust toward occlusions and noisy data. Furthermore, an impossible position can be returned due to a misinterpretation of the data. The use of model prevents the generation of absurd positions because the model is maintained in a valid state at all time. The analysis of the data is also simplified by the use of a model using motion prediction: Indeed, knowing the current state of the model as well as its past evolution is usually sufficient to predict (more or less accurately) its future state in the next frame. This predicted position is used as a starting point for the tracking, making it much more efficient. Figure 2.2 provides a schematic summary of the steps commonly used in model-based tracking.

In the context of human body tracking, we are mainly interested in the functional models which help understanding the movement rather than the accurate anthropomorphic models destined to rendering photo-realistic avatars. Nevertheless some interesting work has been achieved in this last domain including the semi-automatic generation of models [ABH⁺94] and the standardization [SCPMT00] of

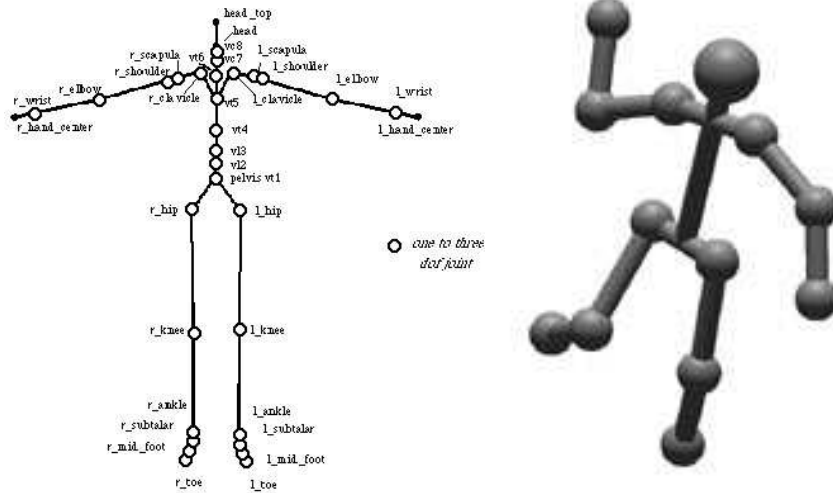


Figure 2.3: (left) Standard human joints and bones hierarchy as defined by the MPEG-4 standard. In practice, this representation is too complex for real-time tracking, so a simplified model (right) is preferred

human avatars via MPEG-4. An interesting domain of research is the link (parameter transformation) between the models used for tracking and the finer models used for rendering.

Almost all the model-based tracking techniques rely on geometric models to describe the links between the parts of the body and the way they should move. Geometric models are all composed of bones and joints. The bones are usually straight, modeled by a cylinder, and are of constant length. The joints are just connections between bones which can have up to 3 rotational degrees of freedom (DOF). All the bones and joints are linked inside a hierarchy (tree) which root is usually the sacrum or the pelvis of the human and the leaves are the extremities (hands, feet and head) as illustrated by figure 2.3.

This simple skeleton model is a first step, but it is usually not sufficient for efficient and robust tracking: Constraints, dynamics and shape estimation have to be implemented on top of it. There are two types of constraints that can be applied to a model. *Hard constraints* must not be violated to guaranty the integrity of the model. The extremum angles at each joint are part of the hard constraints because under no circumstance do we want an impossible pose. Avoiding inter-penetration between body parts (when implemented) is also a hard constraint. *Soft constraints* favour the most likely poses among the all possible ones. For example, minimizing the total energy of the system (position of the limbs as low as possible) is a soft constraint. In most cases, soft constraints can be seen as a time-varying potential field, depending on the current position of the model and sometime on its velocity. A skeleton model including constraints is called a *kinematic* model: It can only describe the static state of a system but not its evolution.

The state vector of a kinematic model consists of the model state, q , and the model parameters p . The model state consists of the joint angles so it is a time-varying function whereas the model parameters contains only constant values like the bone lengths. A system motion is more completely modeled when the *dynamics* of the system are modeled as well. A dynamic model describes the state evolution of the

system over time. In a dynamic model the state vector includes velocity as well as position: q, \dot{q}, p . And state evolves according to Newton's First Law:

$$\ddot{q} = W \cdot Q \tag{2.2}$$

Where Q is the vector of external forces applied to the system and W is the inverse of the system mass matrix which describes the distribution of mass in the system. In fact, some dynamic models are governed by more complex laws, including for example a control parameter that allow to drive the model toward a wanted state. A general formulation of the dynamic of a system can be found in the section 2.5 about the Kalman filter.

Many model-driven tracking methods rely on an "attraction to the data" scheme to lead the model to the correct position. This can be achieved in 2D using for example contours [DF99] or in 3D using voxels from a prior 3D reconstruction [LSL02]. A notion of shape more realistic than the simple bone-sticks is then needed: Some implementation use simple geometric primitives (boxes, cylinders and spheres), others use NURBS or superquadrics to model the external surface of the model. Some additional information like the colour or the texture at some positions can be included in the model. The sum of all these visual data is called *appearance model*.

The conjunction of the kinematic, dynamic and appearance models can give a very robust tracker [NH01], especially when used within a statistical framework (see section 2.5).

2.4 Inverse Kinematics

The usual way to describe a system is to specify its parameters q . The pose of the system is computed from these parameters and the position of each part or feature can then be deduced. This approach is called *forward kinematics* and works very well as long as the parameters of the system are known. If now we want to specify (more or less completely) the final state of the system and compute its internal parameters, the usual scheme does not work anymore. In fact, this last problem called *inverse kinematics* is much more complex than forward kinematics because it is usually not entirely defined and can lead to many solutions.

In the case of human body tracking, inverse kinematics can help the fitting of a model to the data when the position of some features have been detected. If, for example, the position of the head, hands and feet of a person are known, then an inverse kinematics algorithm associated with some constraints can compute the full position of the model, and deduce its parameters. As will be explained later, the inverse kinematics methods all rely on an optimization scheme which make them quite appropriate to the incorporation of constraints. It is then possible to mix inverse kinematics with another model-fitting method in order to obtain better results.

2.4.1 Problem formulation

To explain the principle of inverse kinematics and the resolution methods, let us first consider a simple kinematic chain such as an articulated arm. The position of the en effector (the hand) is noted x and M_i

is matrix which describes the coordinate system of a joint i relatively to its father's (the transformation from one part to the next). The relationship between two coordinate systems i and j in the chain is found by concatenating the transformations at the joints encountered during the traversal from joint i to joint j :

$$M_i^j = M_i M_{i+1} \cdots M_{j-1} M_j \quad (2.3)$$

So the position and orientation of the end effector with respect to the base is found by simply concatenating the transformation at each joint along the chain from the base to the point of interest. If we call f this concatenation of matrices, then the forward kinematic problem is simply

$$x = f(q) \quad (2.4)$$

If we want now to place the end effector in a specific position and find the model parameters (inverse kinematic problem), the problem becomes formulated as follow:

$$q = f^{-1}(x) \quad (2.5)$$

Solving the equation (2.5) is not so simple, mainly because f is not linear and thus not easily invertible. A general analytic solution does not exist; instead the problem must be solved with numerical methods for solving systems of non-linear equations. The most common solution methods are based either on matrix inversion or optimization techniques.

2.4.2 First method: Jacobian Inverse

A first method is to linearize locally the problem about the current position of the model. By deriving the equation (2.3), the velocity of the end effector is

$$\dot{x} = \frac{\partial f}{\partial q}(q) \dot{q} = J(q) \dot{q} \quad (2.6)$$

where J is the matrix of derivatives of f with respect to q , called the Jacobian of f . J is an $m \times n$ matrix, where n is the number of joint variables (the dimension of q), and m is the dimension of x (usually 6 with orientation constraints, 3 without). Inverting the equation (2.6) allow us to find the incremental variation in the parameter space provoked by a move of the effector:

$$\dot{q} = J^{-1}(q) \dot{x} \quad (2.7)$$

The principle of the algorithm is now to iterate the equation 2.7 along the movement of the effector, re-computing at each time step the velocity \dot{x} of the effector and the new Jacobian matrix, and deducing the corresponding update \dot{q} in the state vector q . Of course, this scheme assumes that the Jacobian matrix is invertible i.e. J is both square and non-singular. This assumption is in general not valid, but some methods exist to alleviate this problem and to compute a pseudo-inverse or even to transpose J instead of inverting it (extensive details about these technique can be found in [Wel93]). Considering now a complex kinematic model such as a human body, it is possible to place several end-effectors simultaneously by adding successively their respective action to the state vector q . The incorporation

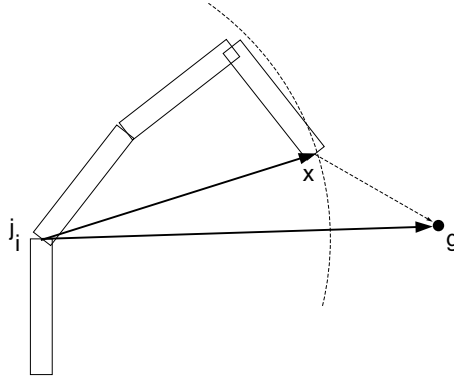


Figure 2.4: Local optimization in the CCD method.

of constraints is not trivial but possible, and even mathematically very formal. The general idea is to compute a constraint Jacobian matrix J_c from which some constraint forces can be deduced using Lagrange multipliers. These forces are then integrated in the state vector to keep the system in a valid state (see [WGW90] for details).

Even if methods based on the Jacobian Matrix are the more formal and the more susceptible to give good results, they are not widely used for fast and interactive computations. This is mainly due to their relative complexity with regard to other iterative techniques like the Coordinate Cyclic descent (CCD).

2.4.3 Second Method: CCD

The idea behind the CCD method is to decompose the global optimization problem into much simpler sub-problems. Instead of trying to find the global change \dot{q} in the state vector, each joint is now optimized independently, and the iteration of these local optimizations results in a global optimization very similar to the one that could have been obtained by inverting the Jacobian matrix.

For a given joint j_i , the idea is to find the angle θ_i which minimizes the distance between the end-effector and its desired position. When only a rotation is possible at the joint, minimizing the distance to the goal is similar to aligning the vector $\vec{j_i x}$ from the joint j_i to the effector x and the vector $\vec{j_i g}$ from j_i to the goal g , as illustrated by the figure 2.4.

The optimization problem at each joint is sufficiently simple to admit an analytical solution (see [Wel93]), and is consequently very fast to solve. A few passes on each joint are generally sufficient to reach a satisfying global solution: The number of passes can be fixed, or dynamically computed by an error measurement. The convergence of the algorithm is ensured by the fact that each local optimization can only close the end effector to the goal.

2.5 Kalman filter

In almost every practical problem, it is impossible to know accurately and certainly the real state, x_k , of a system at the time-step k . There is always a measurement error due to the imperfection of the

captors, as well as the noises. In the case of model-based tracking, the problem is even worse because an imperfect matching of the model can also produce errors in the model pose. In order to have a more robust estimation of the real state of the system (pose of the model), we need to use all the information available: The observed state of the model, z_k , obtained after fitting the model to the data is the first obvious one. The second one is the predicted state of the model, \hat{x}_k^- , that can be deduced from the previous state(s) and the dynamic behaviour of the model (see section 2.3 for details about dynamic models).

The goal is now to combine the two variables z_k and \hat{x}_k^- to obtain the best possible estimation \hat{x}_k of the real state x_k . The Kalman filter [Kal60] is a set of mathematical equations that implement a predictor-corrector type estimator. It is particularly well adapted to solve the fusion problem described earlier, and it is furthermore optimal. Since its introduction forty years ago, the Kalman filter has been used successfully in a quantity of applications, but it only became really interesting in the past few years with advances in digital computing. Indeed, due to its recursive nature, the Kalman filter is very efficient and easy to implement. Concerning human-body tracking, the Kalman filter has been used [KM96, WP98, DF99, WP99, NH01] almost each time a motion prediction was implemented. An introduction to the Kalman filter can be found in [WB01] and a more extensive description has been done in [Gel74].

In fact, the Kalman filter as it was introduced in 1960 is only able to handle linear systems. That is why an Extended Kalman Filter (EKF) has been developed for non-linear systems such as human models. The process $x_k \in \mathbb{R}^n$ to estimate is governed by the non-linear equation

$$x_k = f(x_{k-1}, u_k, w_{k-1}) \quad (2.8)$$

where u_k is a control parameter allowing the change of the standard behaviour of the model. For example, if the model only evolves according to physical dynamic equations, then u_k is always null and the next state of the model is computed only from the previous state. However, if a behavioural understanding is implemented, then the control parameter moves the model toward the wanted position. The parameter w_k is the process noise, and is always present but obviously unmeasurable. It is usually an uncorrelated white noise of known covariance Q_k i.e. $w_k \sim N(0, Q_k)$. Because the noise cannot be predicted, an approximation (*a priori* estimate) of the process state is defined as

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_k, 0) \quad (2.9)$$

where \hat{x}_k is some *a posteriori* estimate of the state (from a previous time step k). The problem with these two equations (2.8 and 2.9) is that we know nothing about the function f : It is for example impossible to invert f easily due to its non-linearities. A linear approximation of x_k will then be defined from (2.8) and (2.9) to allow some further operations. This linear approximation is

$$x_k \approx \hat{x}_k^- + A_k(x_{k-1} - \hat{x}_{k-1}) + W w_{k-1} \quad (2.10)$$

where A is the Jacobian matrix of partial derivatives of f with respect to x

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}(\hat{x}_k, u_k, 0) \quad (2.11)$$

and W is the Jacobian matrix of partial derivatives of f with respect to the process noise w

$$W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}}(\hat{x}_k, u_k, 0) \quad (2.12)$$

In practice, the matrix W is often taken as being equal to the identity because we don't know how to compute the derivative of f with relation to the noise.

Just like for the process state x , the measurement process $z \in \mathfrak{R}^n$ can be modeled by the non-linear equation

$$z_k = h(x_k, v_k) \quad (2.13)$$

where v_k is the measurement noise which is as unmeasurable as the process noise, and which is also an uncorrelated white noise of covariance R_k i.e. $v_k \sim N(0, R_k)$. However, the problem is different for the measurement because the real value of z_k is directly accessible which was not the case for x_k . An estimated measurement (which will be used afterwards) can nevertheless be defined as

$$\hat{z}_k^- = h(\hat{x}_k^-, 0) \quad (2.14)$$

and a linear approximation of the measurement z_k is deduced from these previous equations as

$$z_k \approx \hat{z}_k^- + H_k(x_k - \hat{x}_k^-) + V v_k \quad (2.15)$$

where H is the Jacobian matrix of partial derivatives of h with respect to x

$$H_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}(\hat{x}_k, 0) \quad (2.16)$$

and V is the Jacobian matrix of partial derivative of h with respect to v . Just like previously for the Jacobian matrix W , the matrix V is usually approximated by the identity matrix.

The principle of the Kalman filter is then to find an equation that computes an *a posteriori* state estimate \hat{x}_k as a linear combination of an *a priori* estimate \hat{x}_k^- and a weighted difference between an actual measurement z_k and a measured prediction \hat{z}_k^- . This equation can be written as

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - \hat{z}_k^-) \quad (2.17)$$

or, using (2.14), as

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0)) \quad (2.18)$$

where K_k is an $n \times m$ matrix called the Kalman gain. This matrix is updated at each time step and is responsible for the blending between the estimated state and the actual measurement. We can now define the *a priori* estimate error covariance P_k^- and the *a posteriori* estimate error covariance P_k as

follow:

$$P_k^- = E[(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T] \quad (2.19)$$

$$P_k = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] \quad (2.20)$$

The Kalman gain K_k from equation (2.18) is chosen to be the matrix that minimizes the *a posteriori* error covariance from equation (2.20). After resolution of this optimization problem (the details can be found in [May79]), the optimal value of K_k can be expressed as

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \quad (2.21)$$

In the same way, the error covariance matrices P_k^- and P_k are computed from the equations (2.19) and (2.20) as

$$P_k^- = A_k P_{k-1} A_k^T + Q_{k-1} \quad (2.22)$$

$$P_k = (1 - K_k H_k) P_k^- \quad (2.23)$$

All the equations composing the Kalman filter have now been written. Before summarizing them, we can just remark that they fall in two categories. The equations (2.9 and 2.22) that compute some variable at a time step k from data of a time step $k - 1$ are called time update equations. They can also be thought of as predictor equations and are responsible for projecting forward the current state and error covariance estimates to obtain the *a priori* estimate for the next step. The other equations (2.21, 2.18 and 2.23) refer only to data from the same time step and use the actual measurement to compute the *a posteriori* estimate state of the system. These last equations are called measurement update equations, and can be thought of as correctors. The Kalman filter is then a *predictor-corrector* algorithm with a prediction step summarized in the table 2.1 and a correction step in the table 2.2.

$\hat{x}_k^- = f(\hat{x}_{k-1}, u_k, 0) \quad (2.9)$
$P_k^- = A_k P_{k-1} A_k^T + Q_{k-1} \quad (2.22)$

Table 2.1: EKF time update equations

$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \quad (2.21)$
$\hat{x}_k = \hat{x}_k^- + K_k (z_k - h(\hat{x}_k^-, 0)) \quad (2.18)$
$P_k = (1 - K_k H_k) P_k^- \quad (2.23)$

Table 2.2: EKF measurement update equations

2.6 Recognition

Recognizing a movement is being able to give a sense to it: to interpret it. Movement recognition is typically a classification problem where the current movement has to fall in a category of pre-learned patterns. To understand what is at stake, the spoken language is a good analogy: if the sound waves represent the brut data (before tracking), then a succession of letters (basic sounds) are extracted by the tracking process. These letters correspond typically to the model states. Then the role of the recognition process is to try to make some meaningful words and sentences out of this succession of letters. The applications are twofold: The most obvious one is that further analysis can be performed from this recognition, opening the way to human-computer interactions. The other application is the detection and correction of errors in the tracking: Indeed, just like a meaningless sequence of letters can be detected and hopefully corrected into the most likely real sequence, an improbable sequence of movements can be adjusted. The knowledge of the meaning of the sequence also gives a very high level predictive power. Using again the analogy with the human language, as in a sequence of letters it is possible to compute the probability of the next letter to come given the beginning of the sentence, it should also be possible to predict the next probable position of the model knowing its current sequence of movements.

In the paper "Understanding Purposeful Human Motion" [WP99], Wren and Pentland describe a general framework for using high-level understanding of the motion into the tracking process. The example they use is the tracking of the circular motion of a hand where every physic-based model fails to predict correctly the next position of the hand. However, including a behaviour model coupled to a Kalman filter to control the motion results in an acceptable prediction. Of course a different behaviour model is needed for each type of motion, but the idea is to switch dynamically to the model which matches best the observations. The behaviour models are represented by Hidden Markov Models (HMMs).

Even if it is not directly correlated to movement recognition, the paper "Style machines" [BH00] from Brand and Hertzman can prove very useful to implement behaviour models. The idea developed in this paper is that a movement can be decomposed into its fundamental motor theme (walking, jumping, etc.) and a stylistic variation. An extension of the classic HMMs called Stylistic Hidden Markov Model is introduced, allowing some variations around the standard HMM behaviour via a style variable. Some methods of training by entropy minimization are described, making this representation of behaviour models very practical. The principal advantage of separating the style information from the fundamental movement is that only the last one is necessary for understanding and prediction. Having a way to discard the style variations make the further analysis step much simpler.

Zhao, Wang and Shum describe an approach very closely related to the language recognition problem in their paper "Learning a Highly Structured Motion Model for 3D Human Tracking" [ZWS02]. Their idea is to define an alphabet of the possible body positions: In their implementation, only the tracking of the arms is enabled so they use only 21 clusters (or letters). The conversion of a body position into a letter is just a classification problem that they address using a vector quantization and a fuzzy C-mean algorithm. From the obtained sequence of letters, a dictionary of words is built using the minimum description length (MDL) criterion. These words can be seen as transitions between two states: Identifying the states of the model between the transitions, and merging the similar states leads to the construction of a graph. This graph, built offline during the learning step can then be used to help

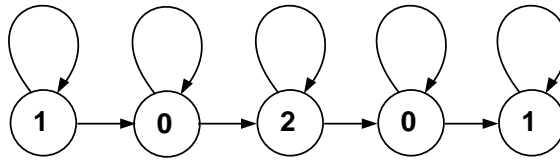


Figure 2.5: A typical Finite State Machine as used in [HTH00].

the tracking, or to generate new movement sequences.

Hong, Turk and Huang [HTH00] have an approach quite similar to the one described above. They use as input the 2D positions of the hands and the head. The vector obtained by concatenation of these positions defines a point in the parameter space. The learning phase consists in acquiring some clouds of these points when performing a particular gesture. No temporal information is used at this point, but it will be incorporated later. A state is modeled as a multidimensional Gaussian, and corresponds to a range of positions. The maximal variance of the states is a value that is determined at the beginning of the algorithm. The states are then extracted from the clouds of points using a dynamic k-means clustering algorithm. Using these states, a Finite State Machine (FSM) is constructed manually during the training sequence. A FSM is a linear automate, as illustrated by the figure 2.5: The FSM presented here can recognize for instance the sequence $\{1, 1, 0, 2, 2, 2, 2, 0, 0, 0, 1\}$. A given FSM then represents a succession of states, that is a movement. For each recognizable movement, a different FSM is trained, and then, the recognition step simply consists in following the succession of states in each FSM: The first to reach a final state has his associated movement recognized. The advantage of using FSMs instead of the more classic HMMs is the simplicity of use, once trained. The inconvenient is that no unsupervised learning method has yet been developed by the authors, even if it is one of their future axis of research.

The approach of Krahnstver, Yeasin and Sharma [KYS01] is more basic. The input is a raw video sequence of a movement, without the use of any kind of model. The part of the video which corresponds to the movement of interest (for example, the region of the legs if we are interested in recognizing the gait) is submitted to a Principal Component Analysis. The images are then projected into the resulting low-dimensional eigenspace. A HMM is trained with the low-dimensional image-sequence. The recognition and the movement prediction are then done using the same treatment for the video sequence in conjunction with the trained HMMs. The idea is interesting, but the lack of model makes it quite unreliable, and anyway unadapted to complex-motions recognition.

Bregler [BM97] uses a hierarchical framework to recognize human dynamics in video sequences of persons running. The framework can be decomposed into four steps: The raw sequence, a model of the movements by mixture of Gaussians, a model of linear dynamics and a model of complex movements. The basic movements are modeled with mixtures of Gaussians using the expectation maximization algorithm on a probabilistic estimation of the motion based on the gradient of the raw sequence. The Gaussians are then grouped into blobs. Each blob is tracked by a Kalman filter. the Kalman filter is chosen to model a movement with constant velocity because the specific motion of each blob is unknown. A cyclic HMM is then used to model the high-level complexity of motions. The HMM is trained with an iterative procedure that requires an initial guess of the model parameters. This guess is obtained by dynamic programming and is fine-tuned by the procedure.

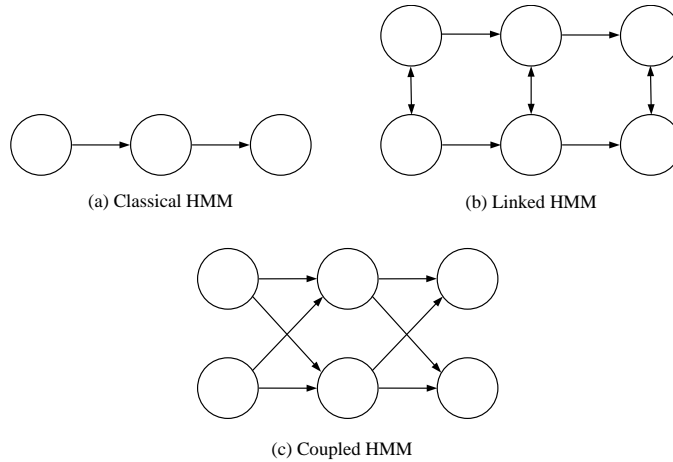


Figure 2.6: (a) In a classical HMM, the probability of being in a given state depends only on the previous state. A state represents all the processes. (b) In a linked HMM, the processes are separated and the output is a joint probability of the state of the other processes. (c) The CHMMs states depend on the previous states of all the processes.

Even if most of the techniques presented above make use of standard HMMs, Brand *et al.* [BOP96] claim that these are too restrictive to learn properly complex processes such as human movements. The main assumption of standard HMMs is that the current state contains all the information needed to infer the next state: They are ill-suited to systems with multiple interacting processes that have structure in both time and space. The example used in the paper is the tracking of the movement of the hands of a subject performing Thai Chi gestures. Each hand is a separate process which clearly depends on the movement of the other hand. The concept of Coupled Hidden Markov Model (CHMM) is then introduced, along with an efficient training algorithm [Bra96]. The figure 2.6 shows a structural comparison between the classical HMMs, the linked HMMs and finally the CHMMs. The test results show that CHMMs outperform standard HMMs and LHMMs for the example of the Thai Chi gestures, but it should suit as well any application including more than one interacting process.

In [GJH99], Galata, Johnson and Hogg propose an interesting trajectory-based approach to movement and behaviour learning. In their paper, the subject is a human facing the camera and the corresponding parameters are some points sampled along the contour of the body. In fact, the parameter space contains not only the position of the points, but also their derivative (the velocity). The dimension of the parameter space is then reduced using a PCA, so that a succession of states corresponds to a trajectory in the reduced parameter space. Just like in [HTH00], some prototypes are extracted from these trajectories using a clustering algorithm. The difference is that a variable length Markov model [RST96] is used instead of a finite state machine to learn the succession of states. The learning phase is totally unsupervised, and the VLMM outperforms standard HMMs for motion recognition and prediction. This first method of clustering is relatively low-level because it does not reflect the behaviours of the subject. Another method is then developed to isolate the different behaviours of the subject. The used assumption is that a movement is characterized by a non-null velocity, so that a transition is a still state. The states with a velocity below a pre-determined threshold are marked as key-prototypes, and the VLMM learns only the succession of these transition states instead of all the states. Recognizing or

predicting intermediate states between the key-prototypes is done with a statistical framework. The test results show that this high-level behaviour learning has a good predictive power and could be used to help the tracking. Instead of points sampled from the contour of the subject, the parameters of a human-body model could be directly used, making this method suitable for model-based motion-prediction and tracking.

Chapter 3

Problem formulation and approach overview

This short chapter is a direct continuation of the introduction in the sense that it will first explore the applications of the system to determine the wanted features and constraints. After this description of the problem, an approach is proposed and described briefly. This overview can be seen as an introduction to the two following chapters which will provide a more precise description of the proposed solution.

3.1 Problem description

As explained briefly in the introduction, the leading idea of this thesis is to use a conjunction of state of the art techniques to perform an accurate and reliable tracking of human-body motions. Even when combining multiple techniques, a general choice of approach has to be made: tracking 2D features is for example quite a different problem than tracking directly in 3D. Some of the decisions are directed by the specifications of a reliable tracking system. Here are some chosen specifications of the system:

- The tracking process should run in real-time to allow application such as human-computer interactions or virtual conferences.
- No hardware device such as captors or markers should be used for tracking: The process should be as non-intrusive as possible. Using structured lights or lasers should also be avoided for the same reason. Cameras are then then only remaining choice.
- No clothing restriction should be imposed to the subject. Of course, tight clothing makes the tracking much easier than large clothes, but the system should be robust enough to track people in a maximum of cases.

- The background should not be imposed to the user. For example, the blue-screen method is too constrainable. Some automatic background segmentation techniques have then to be implemented.
- Occlusions should be dealt with. If part of the body is hidden during a limited amount of time, then the tracking process should be robust enough to guess the position of the missing features and to recover properly when they become visible again.
- The initialization of the system should be as fast and automatic as possible. Many systems rely on imposed positions and movements to initialize the tracking process: This should be limited to the minimum.

These specifications are a first indication of what the system should do, but they are too general to be covered by any implementation. Some assumptions have to be made to limit the generality of the problem, and thus simplify the solution. The assumptions listed below are very common and limit as few as possible the usability of the system:

- The subject remains inside the workspace. This means that a portion of space is defined inside which the subject can be tracked. If the subject goes outside this workspace, then the tracking process stops and starts again only when the subject reappears inside the workspace.
- No camera motions. The cameras should be fixed because adding the problem of camera positioning and calibration is superfluous and can be a source of errors.
- Known camera parameters. Automatic camera calibration is an interesting problem that should be eventually included in any complete tracking system, but it is not the central focus of this thesis.
- Only one person can be tracked at the same time. This assumption is convenient for a first implementation of the system but also discards complex cases when two or more people interact with each other. Nevertheless, an obvious extension could be to track multiple people at the same time, even in physical contact.
- Static background. Foreground segmentation is greatly simplified when the background does not contain any object in motion. This is usually not a big constraint since almost all the backgrounds are static. Nevertheless, if a part of the background happens to change, then the system should be able to take this change in consideration.
- The lighting of the scene should be constant or vary slowly. This assumption is meant to avoid flashing lights which would change totally the representation of the scene from one frame to the next one. A robust background extraction method has then to be used, taking into account the possible changes of luminosity.

From this set of specifications, the next section will be an attempt to provide a solution.

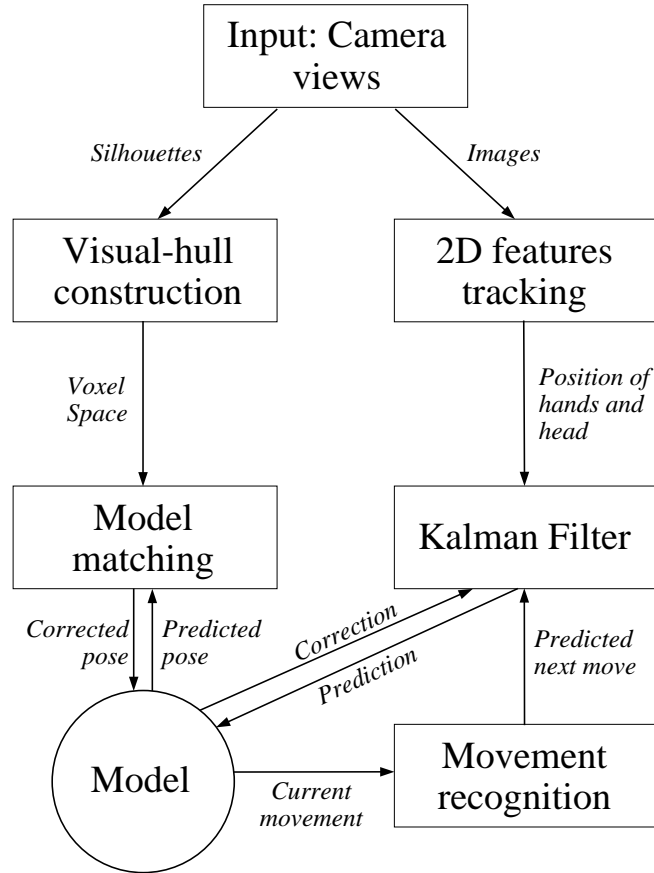


Figure 3.1: Functional overview of the whole tracking system.

3.2 Proposed approach

Figure 3.1 presents an overview of the tracking system that is proposed in this thesis to fulfill the constraints from the previous section. A brief description will follow, but a precise explanation with implementation details can be found in the next two chapters.

The input at the top of the diagram is formed by a conjunction of the views from the cameras (the methods for extracting the data correctly are described in the section 4.1 of the next chapter). The background is then removed from the images, and the silhouettes of the subject of interest are created. The next step is the construction of the visual-hull from these silhouettes (process described in the section 4.2). The voxel space which is the output of the visual hull algorithm is then used in conjunction with a kinematic model to perform some model-matching (see section 4.3.3) based on a pulling force scheme. This chain of processes (the left part of the diagram) is basically a succession of treatment of the data, with no interpretation or understanding of them. Indeed, apart from the kinematic model, nothing here is specific to human-body tracking, and could be applied to any other purpose.

The right part of the diagram is specific to human body-tracking, and can be seen as an assistance to the tracking process previously described. Indeed, it will be pointed out in the next chapter that the model needs to be positioned near the data for the model-matching to be efficient, and this can only

be achieved by some kind of motion prediction. The central part of this motion-prediction process is a Kalman filter which is the perfect mathematical tool for prediction and fusion of information. The Kalman filter takes its data from as many sources as possible. The model itself is one of them: It is a perfect source of information because its parameters are directly exploitable and these are supposed to be reliable (errors in the correction have been corrected by the model-matching). Input from 2D feature tracking can also prove very useful because it provides an accurate piece of information (typically, the position of recognizable features like the hands and the head). A module of movement recognition will also feed the Kalman filter in precious data: Indeed, movement predictions which are only based on pure physics and dynamic have proved to be unable to handle some kinds of movements (like for example a waving hand). The module of movement recognition will be described in the section 5.3, but it is sufficient for now to see it as a database of movements. The movements of the model are compared to this database, and the current movement is deduced, from which the next probable position of the model can be deduced.

After integrating all the data from these various sources, the Kalman filter computes a predicted position for the model. This prediction is used as a basis for the model-matching process which leads to the final position of the model. The advantage of using a Kalman filter is that this final position is then used to correct the parameters of the Kalman filter, and to improve the next prediction. The sources leading to accurate predictions will be more and more trusted while the others will have their weight diminished. For further information about the Kalman filter and its benefits for tracking, refer to the background section 2.5.

Chapter 4

Data processing and tracking

After the global overview of the previous chapter, this part focuses on the acquisition of the data, the way to process them, and the proper tracking task. Most of the techniques used in this section directly refer to the one described in the background section. The limitations of this single approach will be pointed out in the last section, leading to the next chapter.

4.1 Capture and pre-processing

The capture of the data is the first step of any process: From the specifications of the previous chapter, the only possible sources of data are standard cameras. The characteristics of these devices is crucial for the rest of the tracking process. Their resolution and colour depth are two important parameters: A poor-resolution camera will provide inaccurate data while a too high resolution will generate big data being too expansive to process. In fact, high-resolution cameras are often preferred because it is very easy to adapt their output to the needs of the rest of the system via simple resizing filters. Usually, a medium resolution of 320x200 is sufficient for effective human tracking, but it depends greatly on the size of the workspace and on the final application. Frame-rate is another very important parameter: The general rule is to prefer the highest possible frame-rate. The reason why frame-rate is so important is that most of the tracking algorithms use the data from the previous frames as a starting point for the tracking on the current frame: The higher the frame-rate is, the closer the data will be between two successive frames, and the easier the tracking will be. As stated in the previous assumptions, the cameras are supposed to be fix, so as long as the fundamental matrix is known, the capture is straightforward.

The use of multiple cameras is one of the central points of the proposed system. The justification for using more than one camera is mainly that occlusions are greatly reduced if multiple views are available. A hand hidden behind the back of a subject can be seen only if there is a camera placed in the proper position. Therefore the cameras are usually placed in a way that they cover the maximum of space and different viewpoints. Usually 3 cameras are sufficient to alleviate most of the ambiguities of the mono-view approach, but some systems use much more cameras relying on the redundancy of

information to improve the robustness of their tracking. If all the cameras are properly calibrated, then the fusion of their individual data is easy, otherwise the images have to be transformed to be coherent with each other. In the proposed system, the assumption is that the cameras are properly calibrated at all time which means that spotting a feature in all the views must allow the computation of the 3D position of this feature unambiguously. Synchronization also becomes an important issue when multiple cameras are used simultaneously: A shift of just one frame between video streams would lead to incoherent data and would constitute an inappropriate basis for tracking. In some systems, each camera is attached to a separate computer and the video streams are transmitted to a central server through a network. Other hardware configurations allow the use of multiple cameras on a single machine. There is no decisive advantage of a configuration over the other since using only one computer eliminates the network overhead but limits at the same time the parallelization of the processing.

Depending on the algorithm used, background removal is a process that can be operated on each view independently. It is part of the pre-processing step as it does not fully interact with the tracking process. According to the specifications of the previous chapter, the algorithm has to handle constant backgrounds with slow luminosity changes. The most simple solution described the background section 2.2 is to keep a reference background image in memory, and to proceed by background subtraction. The reference background is updated with a multiplying coefficient at each frame, and some post-filters are used to get rid of the misclassified pixels. The advantage of such a simple background removal scheme is that most of it can be done in hardware using the programmability and the processing capacities of recent video-cards. If a new object suddenly appears in the background, then it will be first classified as belonging to the foreground but as the reference background image is slowly updated with the current frames, this object will eventually be integrated in the background.

Once the background is removed, the silhouette extraction is done by simple thresholding. The resulting binary image is then processed by some filters to remove the isolated pixels and to fill the small gaps inside the silhouette, those having a high probability to be due to noise. The current implementation takes in input the views from an avatar instead of real camera-images. This is mostly due to the fact that the necessary devices are unavailable yet. The principal inconvenient of this method of virtual cameras is that the resulting image is noiseless and suffers from no distortion, which never happens in real life. The screen-shot 4.1 shows the output of the current implementation of the silhouette extraction, which is mainly limited to a thresholding algorithm.

Most of the classic tracking methods use directly the silhouettes obtained by background subtraction and thresholding to perform tracking. Nevertheless, the approach chosen in this thesis is to build an intermediate 3D structure and to perform the tracking on it. The advantage of this is that there is no need to emit hypothesis for each view and then to confront them: All the information is contained in a single data-structure so that its use should be optimal. Of course there is an overhead in adding this extra-step, but with a careful implementation, the whole process should still be able to run in real time.

4.2 Visual Hull

The visual hull algorithm is described in the background section 2.2: It is a volumetric representation of the object of interest obtained by intersection of the projections of its silhouettes.

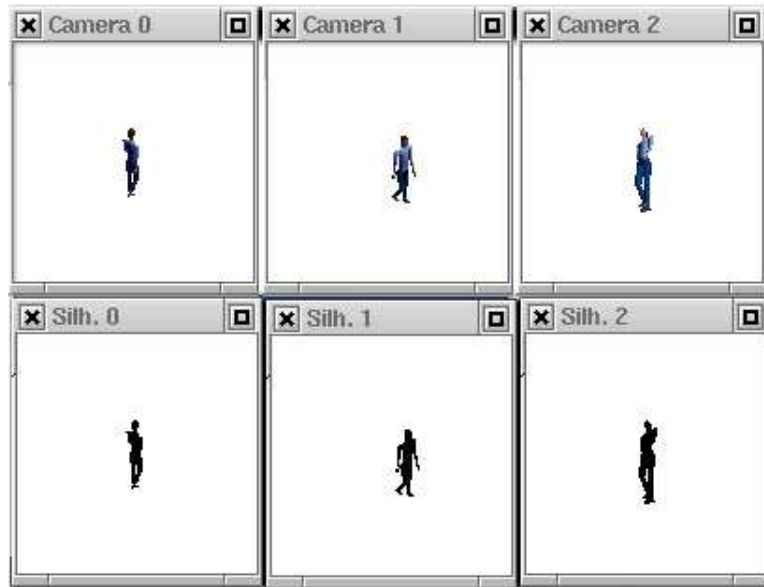


Figure 4.1: Silhouette extraction: The upper row shows the images of the avatar as seen by the 3 cameras (resolution of 128x128), and the bottom row contains the corresponding silhouettes obtained by thresholding.

4.2.1 Design choices

Different algorithms and data-structures were discussed in the background section, but a choice of implementation has to be made at this point. Considering that speed and usability are the two major goals, the voxel data-structure has been chosen. Of course, this solution suffers from a higher memory occupation than octrees or polygonal representations, but it has the advantage of simplicity. It is indeed very easy to access and modify any element of a voxel table.

Mainly for speed purpose, and also because the goal of the constructed visual hull is not to reach photo-realism, a simple projection scheme has been chosen. This means that within all this section, we will consider that a given voxel projects onto a single pixel on each camera view, and reciprocally a given pixel on a camera view projects as a cone in the voxel-space. This approximation allow some speed optimizations such as the implementation of a pixel-voxel lookup table while keeping a sufficient level of accuracy. Indeed, if the size of the voxel-space is comparable to the size of the camera-views, then the projection of a voxel on a camera-view is anyway a surface covering approximatively one pixel.

The above simplification leads to another interesting one: Given that a voxel always projects on a single pixel for each camera-view and that these pixels have binary values (because of the silhouette extraction), then each voxel can only have a binary value too. Some other approaches use intermediate values reflecting the occupancy of a voxel: This is not necessary for tracking purposes because an accuracy of 1 voxel is sufficient. The main benefit of this binary representation is that the memory occupation of the voxel-space can be reduced to the minimum.

4.2.2 Implementation

The input of the algorithm is a set of silhouettes, each of them being a binary image. As stated earlier, the size of the silhouette images should be comparable to the size of the voxel-space, which means that the resolution of the silhouette has to be relatively low to keep the size of the voxel space reasonable. Depending on the power of the machine and the application, a size from 64x64 to 256x256 is perfectly manageable. There are basically two approaches for the implementation of the visual-hull algorithm with a voxel space:

- The first one is to go through each individual voxel, projecting it on each of the silhouettes, and marking it as full if all the projections fall inside the silhouettes, empty otherwise. The projection of a voxel onto the silhouette associated with a camera-view is done using the fundamental matrix of the concerned camera. Nevertheless, even a simple matrix-vector multiplication performed 3 times (for 3 cameras) for each voxel represents a too big amount of computation for real-time. An optimization is to realize that the cameras being fix, there is no need to re-compute the projection of the voxels each time. A lookup table is then built offline, associating each pixels with the pixels it should project onto. In the same way, another optimization is to continue to project a voxel on the camera-views only if it fell inside the silhouettes of the previous views. Indeed, as soon as a voxel projects itself outside one silhouette, it can be marked as empty.
- The other method is to start from the silhouettes, projecting the pixels in the voxel-space. As each silhouette is treated independently from the others, a layered construction of the visual hull is adopted. All the voxels then contain an accumulator which is incremented each time a pixel belonging to a silhouette projects into it. At the end of the process, only the voxels which accumulator contains a value equal to the number of cameras are marked as full. As stated earlier, each pixel projects as a cone of voxels in the voxel-space: Computing this projection each time is too far expensive for real-time, so a lookup-table approach is adopted. Each pixel of the camera-view images is associated with the list of voxels it projects into. The lookup table is pre-computed offline by projecting each voxel into the camera-views, storing the correspondence between this voxel and the pixels concerned by the projection. The obvious optimization at this point is to update only the voxels attached to pixels lying inside a silhouette.

Both methods are almost equivalent regarding the computational complexity and the memory occupation. However, the second algorithm has a slight advantage in the facts that only the voxels that project in at least one silhouette are visited. Considering that the voxel-space is the largest data-structure, and that the silhouettes usually occupy a reduced portion of the camera-views, it is an appreciable improvement.

Both the two previously described algorithms have been implemented as part of this thesis. The chosen programming language in C++, which proved very powerful along with an object-oriented programming approach and which also performs very well at the same time. In fact, C++ will be the programming language utilized everywhere through this thesis. However, a major inconvenient of C++ is that it does not contain in itself some standard libraries for managing the common data-structures such as lists and trees. The Standard Template Library¹ (STL) is aimed at providing to the C++ language a standard and generic way to handle a wide range of data-structures. The STL will then be used

¹<http://www.sgi.com/tech/stl/>

extensively. Another important part of the implementation tools is the graphic library. OpenGL² is the unavoidable cross-platform standard 3D library for all the rendering process. It is used here through a higher level library called MAVERIK [HDG96, HCK⁺01]. MAVERIK is not only an interface for the OpenGL graphic library, but a full-featured virtual-reality micro kernel. Among all its functionalities, one can find the support for a wide range of input and output devices, some navigation schemes, a framework for managing the display and interaction of objects, some algorithms for culling, and many more. In fact, the main benefit of using MAVERIK is that most of the common functionalities are already there, so only the interesting part needs to be implemented.

From the silhouettes extracted in the previous section (128x128 images were used), the visual-hull algorithm produced the screen-shots shown on the figure 4.2. By examining these images, we can see that the constructed visual-hull should be precise enough for the tracking, even if some artifacts are present, mostly due to zones that are visible by none of the cameras. The frame-rate of the algorithm is currently only of a few frames per second on a Pentium 450MHz processor. Future optimizations should allow speed improvements, especially because the bottleneck seems to be the silhouette extraction and the display process.

An additional post-processing step has also been implemented: The removal of the internal voxels. This step will prove useful in the rest of the tracking process, while clarifying the visualization of the visual-hull. Moreover, profiling tests show that even the basic algorithm which consists on removing the voxels having neighbours on every side runs in a very small time compared to the rest of the process.

The output of the visual-hull construction is a binary voxel-space (an 3-dimension array) containing the volumetric representation of the subject of interest. At this point, no knowledge about the geometry or dynamics of the subject has been used: The silhouettes extraction and visual-hull construction could work on any type of object in exactly the same way. The next stage is to try to give a structure to the data by using a model and trying to match it on the visual-hull. A kinematic model (see section 2.3 for background introduction) of the human body is then used along with an inverse-kinematics scheme (section 2.4).

4.3 Model

4.3.1 The kinematic model

As a reminder, an kinematic model describes the relations between the bones and joints, but not the attributes of the body-parts. Some simple constraints can also be part of the kinematic model, such as the range of rotation for the joints. As explained in the background section, a kinematic model is only composed of joint and bones, arranged in a hierarchy. Given that in a human body (like for most of the skeletons) a bone always follows a joint, only one common structure will be used in the implementation. This structure is a conjunction of a 1 DOF-joint and a bone in the sense that it has a rotation axis, a rotation angle, and a length. This structure also contains a list of sons and a link to its father to include it in the hierarchy of the model.

²<http://www.opengl.org>

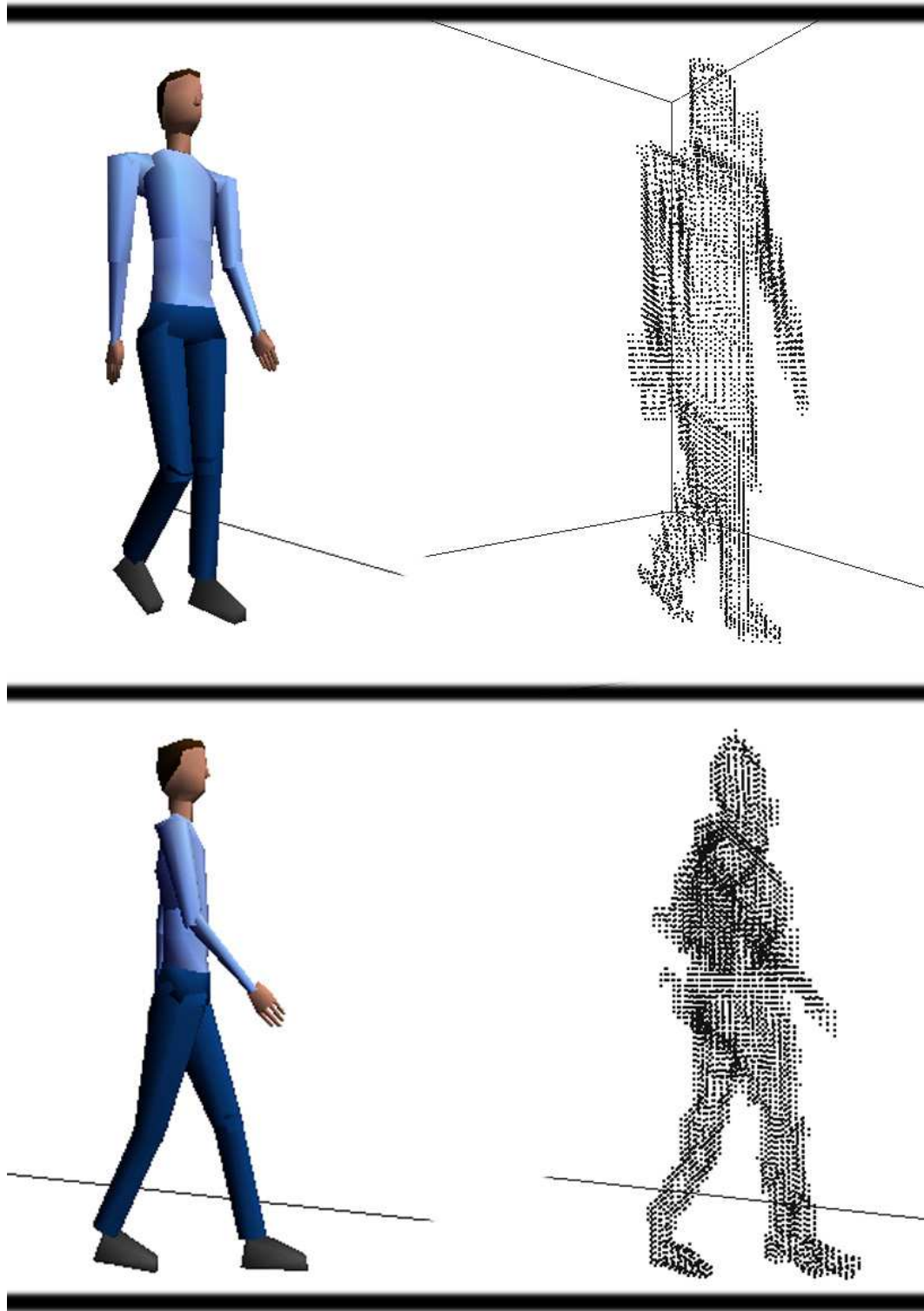


Figure 4.2: The same visual-hull construction from two different viewpoints. Voxels are represented by points. A voxel-space of size $128 \times 128 \times 128$ is used here, with 3 cameras. The inside voxels have also been removed.

Only one degree of freedom is implemented for each joint, mainly for generality and simplicity purpose. In fact, most of the joints in the human body have more than one degree of freedom, but their modelisation is then done by superposition of two or tree 1-DOF joints. For example, a shoulder-joint with 3 DOFs will be modeled by three 1-DOF joints rotating around 3 perpendicular axis. Each joint is then defined by its axis of rotation (which is fixed at the creation of the joint, and does not change afterwards), and its current angle of rotation. The angle of rotation is just a scalar real value. The actual rotation and the translation corresponding to the bone length are stored in a 4x4 homogeneous transformation matrix which is updated using a quaternion.

In fact, each of these joint/bone combination exists in its own local coordinate system. This means that the global position of each joint is relative to the one of its father. For example, rotating a joint in the shoulder of the model will rotate the whole arm. Far from being a problem, this even represents an advantage because most of the updates are done this way. The only negative point is that when the global position of a joint is needed, the whole hierarchy has to be walked through until the root.

Simple hard constraints were also implemented. The rotation angle of each joint is constrained to remain inside a range of values. When the angle gets over one of the allowed extrema, it is clamped to this extremum. Of course, this implementation of the constraints is simplistic and should be improved in the next versions of the program. Dependance between constraints should for example be added, allowing a more precise definition of the zone of allowance for 2-DOF and 3-DOF joints. Other constraints like forbidding inter-penetration of bones would also be a benefit.

4.3.2 Inverse kinematics

Inverse kinematics is used to update in a coherent way the posture of the model when the position of some of the joints are specified. Typically, the position of some extremities like the hands and the head can be deduced from colour-based information, leading to a good starting position for the tracking of the rest of the body-parts. Even during process of matching each joint to the data, no joint can be moved independently from the others because it would break the coherence of the global pose of the model. This is why inverse kinematics is extensively used for positioning individual joints.

Two methods were described in the background section, and even if the one base on the Jacobian Inverse is mathematically more rigourous, the one chosen here is the CCD. Once again, speed and simplicity considerations were at stake in this choice. Furthermore, each joint having only one degree of freedom, the CCD method become very easy to implement and gives good results while allowing the incorporation of constraints.

The principle of the CCD method is first, for each joint J_i , to find if one of its descendants has to be moved. If the joint J_k has to be moved to the goal-position G , then the only way to get J_k closer to G by only affecting the current joint J_i is to minimize the angle between the vectors $\overrightarrow{J_i J_k}$ and $\overrightarrow{J_i G}$. Each joint has only one degree of freedom (a rotation around an axis \vec{R}), so that the new rotation angle is the only parameter to compute at the current joint. The formulae below are used to compute the optimal rotation angle of the Joint J_i (the one that minimizes the angle between $\overrightarrow{J_i J_k}$ and $\overrightarrow{J_i G}$).

$$k_1 = \overrightarrow{J_i G} \cdot \vec{R} \times \overrightarrow{J_i J_k} \cdot \vec{R} \quad (4.1)$$

$$k_2 = \overrightarrow{J_i G} \cdot \overrightarrow{J_i J_k} \quad (4.2)$$

$$k_3 = \overrightarrow{R} \cdot (\overrightarrow{J_i G} \wedge \overrightarrow{J_i J_k}) \quad (4.3)$$

$$OptimalAngle = \arctan\left(\frac{k_3}{k_2 - k_1}\right) \quad (4.4)$$

In fact, this new angle is not directly affected to the joint J_i because we want all the other joints to contribute as well to the movement. The CCD method goes successively through all the joint from the effector to the root of the kinematic tree, and then reiterates this process a number of times until the movement is stabilized. That is why the value of the angle at the current joint is only updated by a fraction of the optimal angle, in a way to limit instabilities while eventually reaching the optimal value after a few iterations.

$$NewAngle = PreviousAngle + \alpha \times (OptimalAngle - PreviousAngle) \quad (4.5)$$

where α is a coefficient depending on the number of iterations of the CCD algorithm. A value around $\alpha = 0.1$ proved to give good results.

Moving a joint and having all the other joints moving accordingly can however pose some problems in the tracking process. If, for example, the position of a shoulder is known and we want to move the corresponding hand toward its position, then we do not want the shoulder to move as it would be the case with a standard inverse kinematics algorithm. Instead, a convenient solution would be to handle at the same time the fact that the hand should move to its goal position, and that the shoulder should stay as much as possible in place. The CCD method should then be implemented in a way that it handles multiple goals at the same time.

The chosen way to implement this is to affect a possible goal to each joint. When the CCD algorithm is applied to a given joint, all the descendants of this joint are explored to find the ones affected to a goal. The vectors pointing to these joints, as well as to their goals are stored in a stack, so that a new rotation angle for the current joint can be computed for each goal. These rotation angles are then averaged and the angle of the current joint is updated toward this average goal. This way, if all the goals can be satisfied, then the algorithm will converge toward a solution, otherwise the algorithm will still converge, but toward a compromise between all the goals.

Simple constraints are also handled by the current implementation of the Inverse Kinematics algorithm. The CCD method visits each joint at a turn, so it is quite easy to maintain constraints for this joint: The only thing to do is to ensure that the angle remains within the acceptable range specified at the creation of the joint. If the new angle computed by the CCD algorithm falls out of this range, then the effective new angle will be clamped to the allowed extremum. The CCD algorithm performs a number of passes on each joint on the kinematic chain, so even if one joint is constrained, the other joints will eventually compensate and the effector will still converge toward the goal.

4.3.3 Matching

Now that a proper model has been defined and implemented, the next step is to match it on the data produced by the visual-hull algorithm. This is probably the most difficult problem until now: The reason

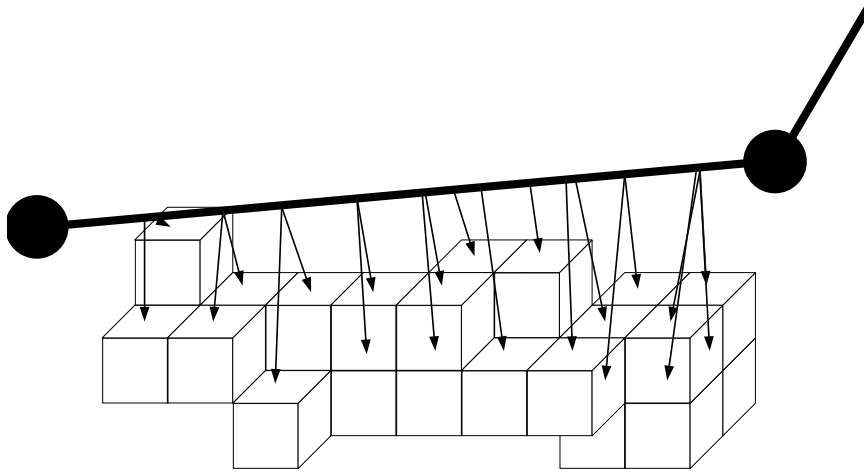


Figure 4.3: Each voxel exerts an attraction force to the nearest bone, making the model converge toward the data

is that there is no general solution to match correctly a 3D model on a set of voxels. Furthermore, the proposed solution has to be very robust to handle all the noise present in the voxels. Some assumptions will also have to be made, because no algorithm has yet been developed to handle model-matching in a general context. The first of these assumptions is that the model should correspond quite precisely to the data to track. This is not an easy task because all the bones can have different lengths from a subject to another. Moreover, the global scale of the model has to correspond to the one of the data. One way to adjust the model parameters to the data is to perform an initialization step where a special pose is taken by the subject (this part has still to be implemented). The other assumption is to suppose that the model is at all time sufficiently close to the data. This means that the initial position of the model has to be chosen carefully because only small adjustments will be handled by the model-matching algorithm.

The chosen algorithm for model-matching is an attraction scheme. The principle is that the voxels are supposed to attract the bones of the model so that eventually, after a few iterations, the model is superposed with the data. One can also visualize this attraction scheme by virtual springs being attached between the voxels and the bones of the model as illustrated by the figure 4.3.

At first glance, this approach seems to be the perfect solution to the model-matching problem. However, a big inconvenient of the algorithm is that there is no way to know which bone should be attracted by a given voxel. This is because the voxel-space as produced by the visual-hull algorithm has no structure: A voxel cannot be said to belong to the forearm or the leg. The only solution to this problem is to consider that the model is already sufficiently close to the data in a way that the nearest bone to each voxel is the one that should be attracted.

Of course, the above method is not sufficient to implement correctly the attraction scheme. In the real implementation, each voxel can have an effect on all of the bone segments, which overcomes the problem of having to decide which bone segment a voxel belongs to. However, each pull is weighted by the distance, d , to each bone so that a voxel exerts a stronger pull on closer bones than on those further away. For each voxel, let the minimum distance to to any bone be denoted as d_{min} . The weight for each bone is given by $(\frac{d_{min}}{d})^3$. Accordingly as the model is pulled into the correct orientation, the

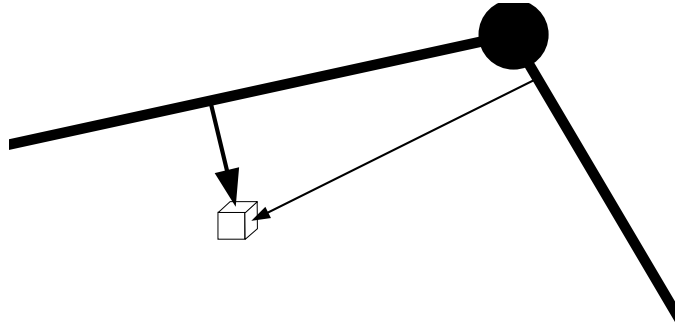


Figure 4.4: A given voxel exerts on each bone an attraction force which depends on the distance between this voxel and the bone.

forces exerted from particular voxel should be almost entirely on the segment closest to the voxel. In addition, a zero weighting is applied to any bone further than a certain distance from the voxel (this maximum distance, d_{max} , is pre-computed as the maximum velocity of a bone divided by the tracking rate). Also, if a voxel is within a very small distance from an bone (less than d_{min}), then it is assumed that it belongs to only that segment, and a zero weight is assigned to all the other segments. These weighting assignments are summarized below:

$$weight = \begin{cases} 1 & \text{if } d \leq d_{min} \\ (\frac{d_{min}}{d})^3 & \text{if } d_{min} < d < d_{max} \\ 0 & \text{if } d \geq d_{max} \end{cases}$$

The figure 4.4 is an illustration of the above table where the pulling force from a voxel is strongest toward the closest bone that toward a more distant one.

The last problem to solve is to transform the forces at the bones into forces at the joints. Indeed, the kinematic model along with the inverse kinematic scheme described earlier only allow to move joints. In fact, it is quite easy to do so by adding for a joint all the forces on the adjacent bones proportionally to the distance these forces apply from the joint. This way, a pulling force applied on a bone near the joint will almost exclusively apply on this joint, and a force applied near the middle of a bone will be equally distributed between the joints.

Even if it is not yet entirely implemented, this model-matching algorithm should work pretty well when the initial position of the model is sufficiently close to the data. To ensure that only small adjustments are to be made by the model-matching algorithm, the frame-rate of the overall tracking process has to be sufficiently high to have only small moves of the subject between two consecutive frames. Even if this is a *sinequanon* condition, it is practically not sufficient because rapid moves of the subject will still result in big changes between two frames. Some methods have then to be utilized to place the model near the data

The rest of this thesis deals with the ways of positioning the model near the data to make the model-matching more efficient. The main way to perform this is to use a scheme for motion prediction. A dynamic model will be used along with filtering techniques such as the Kalman Filter. An important part of movement recognition will also be developed to help the prediction, and also to interpret the movements of the model.

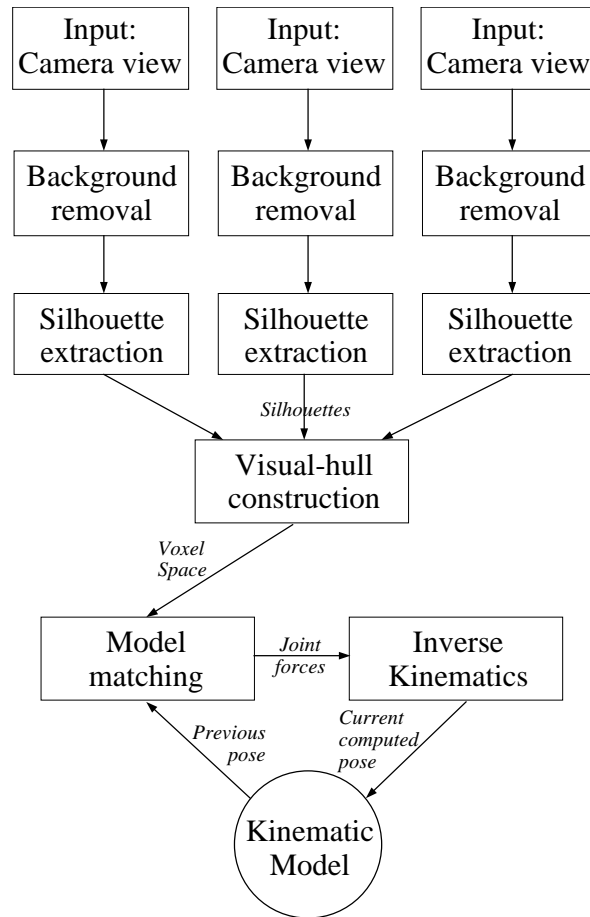


Figure 4.5: Functional description of the tracking process, without motion prediction.

4.4 Conclusion

This chapter presented the adopted approach for tracking human-body from camera views. In the first section, acquisition constraints were described, along with background removal and silhouette extraction techniques. These silhouettes were used to construct the visual-hull. Then, the proper tracking part was the model-matching process using a kinematic model and an inverse kinematic module to move it. A functional representation of this system is shown on the figure 4.5.

Even if its implementation is not yet totally completed, this system should allow a correct tracking of human bodies. However, the system will inarguably benefit from a motion-prediction scheme. This last process will be described in the following chapter.

Chapter 5

Motion prediction

This chapter focuses on the part of the tracking which predicts the position of the model in the next frame. As we will see, there is not a single module achieving this but several all them, all connected to a Kalman filter for the fusion of the data. These modules are reasonably independent from each others, and will be described successively. However, the work on this part has currently not been started, so no implementation is yet available. The description will then be only formal, and not as detailed as it could be.

5.1 The Kalman Filter

The Kalman filter is basically a predictor-corrector algorithm which can fusion the data from various sensors to produce the best estimate of the state of the system (in this precise case, the current pose of the model). A mathematic description of the Kalman filter can be found in the background section 2.5: In this section, we will try to associate the definitions and variables of the abstract description with the practical modules and data that will be used in the human-body tracker.

A quick reminder of the equations of the Kalman filter may be necessary to find the variables and functions which need to be associated with real data. Firstly, the time update equations were:

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_k, 0) \quad (5.1)$$

$$P_k^- = A_k P_{k-1} A_k^T + Q_{k-1} \quad (5.2)$$

and then, the measurement update equations:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \quad (5.3)$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - h(\hat{x}_k^-, 0)) \quad (5.4)$$

$$P_k = (1 - K_k H_k) P_k^- \quad (5.5)$$

Some variables are obvious to associate. For example, \hat{x}_k represents the current state of the model, and \hat{x}_{k-1} the previous state of the model. What is called *state* of the model is in fact the vector of the model's parameters (i.e. the joint angles and the angular velocities). These data are very easy to obtain from the kinematic model described in the previous chapter. The second parameter of the function f in the first equation may be more difficult to understand: The driving force u_k is supposed to be an indication on the next position of the model. Typically, this input can be provided by the movement recognition module. The advantage of using this high-level motion prediction is that it should complements perfectly the other low-level parameter.

The function f is still to define: It is a very important part of the algorithm because this function is responsible for the a-priori estimate of the model's state. It should compute an estimation of the current model state from the last known position of the model \hat{x}_{k-1} and from the driving force u_k . For one part, the function f is based on motion prediction using a dynamic model. This prediction is only based on physics (it follows Newton's laws of movement), and can be seen as the inertia of the system: An arm moving in a direction has strong chances to continue its movement in the same direction in the next time-step. This first part of the function f is the low-level motion-prediction: No knowledge of human behaviour is required. Oppositely, the second part is the one taking into account the driving force u_k , and corresponds to a high-level motion prediction. The exact way to blend these two parts of the function f is still investigated, and will constitute one of the axis of research for the next year. The matrix A is the Jacobian matrix of partial derivatives of f with respect to x , which will make it easy to compute as soon as the function f is properly defined.

The noise covariance matrices Q_k and R_k have not been determined yet, and will probably have to be tuned by experimental testing. In fact, their value will depend greatly on the physical devices used for acquiring the data.

The last thing to determine is the exact nature of the observed process. It cannot be the model because it is only an estimation of the real pose of the subject, and moreover it is already used as a correction parameter for the filter. The voxel-space obtained by the visual-hull algorithm could be a good candidate as the observed data-set, but unfortunately it is far too big to be practically usable. The other problem with the voxel-space is that the mapping between the visual-hull and the model is not obvious even though the function h associating the observation with the model needs to be defined. Another, more simple, observed data must then be used: The constraints are that this observed data-source should be simple enough and not altered with external knowledge like the model.

The 3D position of the hands and the head, computed from an image-based tracking process is a perfect candidate for the observed data-set. Indeed, it generates only a few parameters and is independent from the rest of the tracking process, guaranteeing the "purity" of the data. Of course, a drawback of the simplicity is that this data-set will not describe entirely the state of the model: There is an infinity of possible states of the model for one given position of hands and head. Some tests still have to be done to confirm this, but this lack of completeness should not be a problem because the major part of the parameters of the model can be deduced from the position of a few limbs. The function h defined as $h(\hat{x}_k, 0) = \hat{z}_k$ is then trivial: It is just supposed to compute the position of the hands and head from the model. In the same respect, the matrix H which is the Jacobian matrix of partial derivatives of h will be simple to deduce.

5.2 Image-based features tracking

This module takes the images from the cameras in input, and is supposed to find the positions of some specific features like the hands or the head. Contrarily to the algorithms presented in the previous chapter, this 2D-feature tracking will be a very specialized process, applicable only to human-body tracking. Despite this specialization, the 2D feature-tracking algorithm should be able to handle a wide variety of inputs: Basically, any human being should be treated in the same way.

Because of the too different appearance of the clothes, the only generic and trackable parts of the body are the bare ones, such as the hands and the face. The tracking will then mainly rely on a colour segmentation scheme. Of course, the colour of different skins is not the same, but a prior calibration of the system should allow any skin colour to be properly segmented. Another problem may arise when the subject has more bare body parts (for example when wearing a T-shirt) than the ones destined to be tracked. The idea will be to use some knowledge from the previous frames and possibly from the model to know which are the real features to use.

An overview of the considered method for 2D feature tracking is then the following: Images from the camera-views are the input of the algorithm. In fact, the background will probably be removed from these input images as it would only complicate the later segmentation. The rest of the algorithm is inspired by the Pfinder system [WADP97], of which a description can be found in the background section 2.1: The basic idea is to grow Gaussians blobs by likelihood of colour and proximity. To identify each blob (knowing for example that a particular blob should be the right hand), the previous positions of each feature can be used, as well as some information about the shape of the blob (size, axes, etc...). After identification, the central position of each blob is computed, and finally all the 2D-coordinates of the blobs are projected and intersected in a 3D space to obtain the 3D position of each feature. These 3D positions are the outputs of the algorithm.

5.3 Movement recognition and prediction

The module of movement recognition and high level prediction is probably the most complex among the whole system's. It has to make intensive use of AI techniques such as unsupervised learning or classification. This module is still in elaboration phase, so only the outlines will be described in this section: Most of the work has still to be done.

The input of the recognition module is the only model state. Contrarily to the other modules, the recognition module does not deal with the raw data, which makes it quite independent from the rest of the tracking process. The output of the module is an indication of the probable next move of the model. The exact format of the prediction is still to define, because it will highly depend on the implementation of the function f from the Kalman filter (see section 5.1). It is for example imaginable to output a vector containing the few most probable moves with their associated probabilities.

Human behaviours is a far too complex subject to be dictated by a restricted set of laws, so that the motion recognition and prediction have to be based on a learning process. Supervised learning¹

¹In this case, supervised learning would correspond to a hand-labeling of each state of the model making the classification obvious.

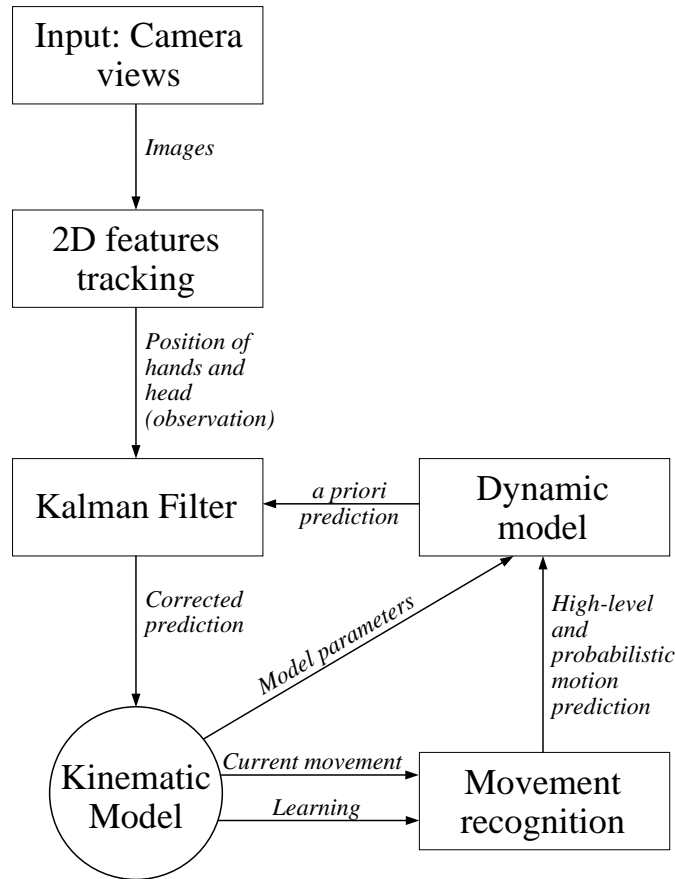


Figure 5.1: Functional diagram of the motion prediction process.

is the easiest way to proceed, but also implies a too important amount of human work. The resulting inconvenient is that no learning can be done from new data without human intervention. Unsupervised learning techniques will then be preferred, even if they are more difficult to implement, and most of the time less reliable. In fact, the main advantage of unsupervised learning is that the learning phase can be continuous and take place at the same time as the tracking.

5.4 conclusion

In this chapter, a framework for motion prediction was described. The central piece of the process is an extended Kalman filter, predicting the model pose from an observation and an a priori estimation module. The observation module tracks the position of the hands and the head from 2D images. The a priori estimation of the state of the model is a conjunction of a high level prediction (via motion recognition) and physical simulation (via the dynamic model). The movement recognition module also includes a learning process which makes it more robust as the examples multiply. A graphical overview of the motion-prediction framework is presented in the figure 5.1.

Chapter 6

Conclusion and future work

This last chapter presents a brief summary of the work currently finished, followed by a presentation of the future axis of research and a scope of the things to do. The last section is a schedule for the next 2 years, describing approximatively the time that should be spent on each part of the work.

6.1 Summary of work currently done

The parts considered as finished are the ones with a working implementation. As demonstrated in the earlier chapters, most of the tracking process is implemented, but some algorithms that were simplified will have to be fully implemented in the final version. The acquisition of the data is an example of domain where an alternative method using virtual cameras has been used, but some real video sequences will have to be captured in the final version. The same is true for the silhouette extraction which does not handle completely the noise yet, but will have to be more robust when dealing with real video sequences.

The visual-hull construction is now fully implemented, and even if some speed optimizations can still be done, it works reasonably well. The Kinematic model also works, and the "CCD" Inverse Kinematics algorithms has been implemented and tested on top of it. Some work has been begun on model-matching, but the implementation is still far from complete.

6.2 Future works

As soon as the tracking part will be fully implemented, some work will begin on the motion-prediction process. Even if some ideas have been formulated through this report and a framework has been proposed, a big amount of research has still to be carried out. The parameters of the Kalman filter are already specified for the most part, but the modules involved are still to finalize. For example, the 2D feature tracking module which is supposed to return the position of the hands and the head from images is one of then future directions of research. The dynamic model should be easier to implement because it is mostly a simple physical simulation.

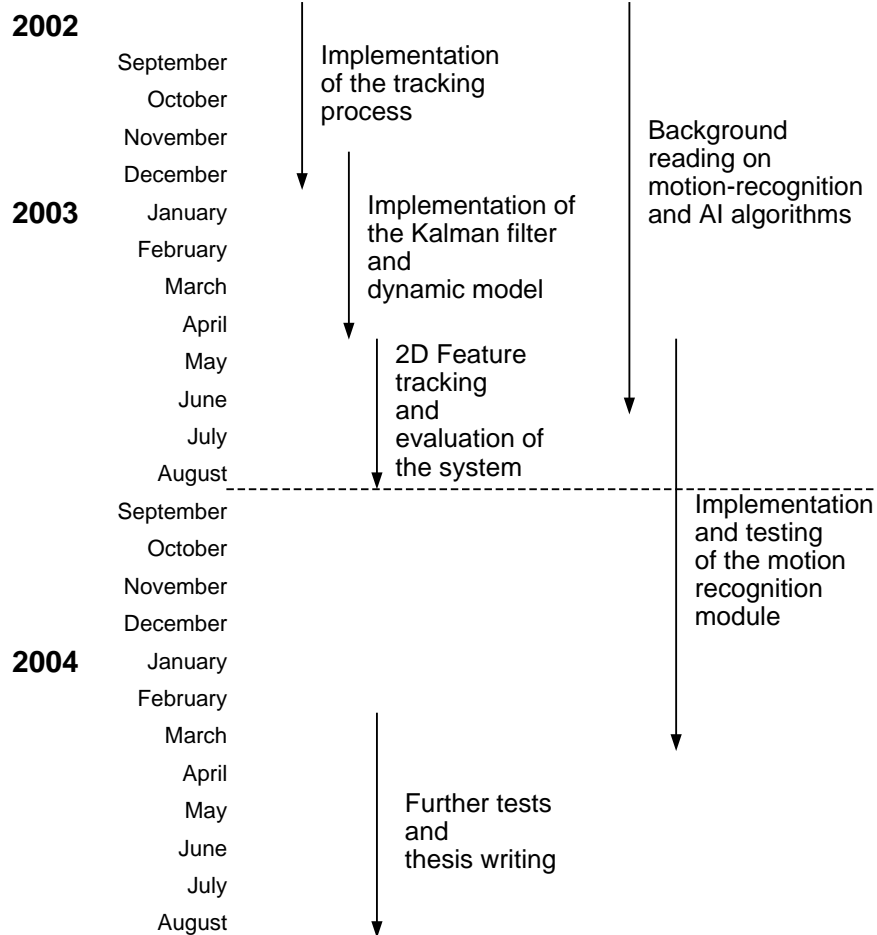


Figure 6.1: Work schedule for the rest of the PhD

The most difficult part of the project, and probably the one that should bring the most interesting benefits is the movement recognition module. The exact implementation of this module is still uncertain, and it will constitute one of my main axis of research for the next years. Hidden Markov Models will probably be used in the conception of this module, mainly for their learning ability and their statistical predictive power. Some interesting variants or extensions of the HMMs also have to be investigated: The Variable Length Markov Models [RST96] can represent an advantageous alternative over standard HMMs because of their efficient storage of information.

6.3 Work schedule

The figure 6.1 is my expected work schedule for the 2 next years of PhD. Of course, the dates are approximative, and subject to modifications since nothing never goes as expected. I will also try to publish at least a paper during the second year of the PhD.

Bibliography

- [ABH⁺94] F. Azuola, N. Badler, P. Ho, I. Kakadiaris, D. Metaxas, and B. Ting. Building anthropometry-based virtual human models. In *In Proceedings of IMAGE VII Conference*, 1994.
- [BH00] Matthew Brand and Aaron Hertzmann. Style machines. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 183–192. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
- [BM97] Christoph Bregler and Jitendra Malik. Learning appearance based models: Mixtures of second moment experts. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 845. The MIT Press, 1997.
- [BMG⁺99] T. E. Boult, R. Micheals, X. Gao, P. Lewis, W. Yin, and A. Erkan. Framerate omnidirectional surveillance and tracking of camouflaged and occluded targets. In *Second IEEE Workshop on Visual Surveillance*, pages 48–55, June 1999.
- [BOP96] Matthew Brand, Nuria Oliver, and Alex Pentland. Coupled hidden markov models for complex action recognition. In *In Proceedings of IEEE CVPR97*, 1996.
- [Bra96] M. Brand. Coupled hidden markov models for modeling interacting processes. Technical Report TR-405, MIT Media lab Perceptual Computing/Learning and Common Sense, November 1996.
- [CA86] C. H. Chien and J. K. Aggarwal. Volume surface octrees for the representation of 3d objects. In *Computer Vision, Graphics and Image Processing*, volume 36, pages 100–113, 1986.
- [CET98] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. In *European Conference on Computer Vision*, volume 2, pages 484–498. H. Burkhardt & B. Neumann, 1998.
- [Chr01] Wojciech Matusik Chris. Polyhedral visual hulls for real-time rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering*, pages 115–125, June 2001.
- [CKBH00] Kong Man Cheung, Takeo Kanade, J.-Y. Bouguet, and M. Holler. A real time system for robust 3d voxel reconstruction of human motions. In *Proceedings of the 2000 IEEE*

-
- Conference on Computer Vision and Pattern Recognition (CVPR '00)*, volume 2, pages 714–720, June 2000.
- [CTCG95] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models - their training and application. *Computer Vision, Graphics and Image Processing*, 61(1):38–59, January 1995.
- [DF99] Quentin Delamarre and Olivier D. Faugeras. 3d articulated models and multi-view tracking with silhouettes. In *ICCV (2)*, pages 716–721, 1999.
- [Gel74] A. Gelb. *Applied Optimal Estimation*. Cambridge, MA: MIT Press, 1974.
- [GJH99] Aphrodite Galata, Neil Johnson, and David Hogg. Learning structured behaviour models using variable length markov models. In *IEEE Workshop on Modelling People, Corfu, Greece*, September 1999.
- [HCK⁺01] Roger Hubbold, Jon Cook, Martin Keates, Simon Gibson, Toby Howard, Alan Murta, Adrian West, and Steve Pettifer. GNU/MAVERIK: A micro-kernel for large-scale virtual environments. In *Presence: Teloperators and Virtual Environments*, pages 10:22–34, February 2001.
- [HDG96] Roger J. Hubbold, Xiao Dongbo, and Simon Gibson. MAVERIK - the manchester virtual environment interface kernel. In M. Göbel, J. David, P. Slavik, and J. J. van Wijk, editors, *Virtual Environments and Scientific Visualization '96*, pages 11–20. Springer-Verlag Wien, 1996.
- [HS99] J. Heikkila and O. Silven. A real-time system for monitoring of cyclists and pedestrians. In *Second IEEE Workshop on Visual Surveillance*, pages 74–81, June 1999.
- [HTH00] Pengyu Hong, Matthew Turk, and Thomas S. Huang. Gesture modeling and recognition using finite state machines. In *Proceedings of the Fourth International Conference on Automatic Face and Gesture Recognition*, pages 410–415, March 2000.
- [IB96] Michael Isard and Andrew Blake. Contour tracking by stochastic propagation of conditional density. In *ECCV (1)*, pages 343–356, 1996.
- [IB98] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. In *International Journal of Computer Vision*, volume 29(1), pages 5–28, 1998.
- [Kal60] R. E. Kalman. A new approach to linear filtering and prediction problems. In *Transaction of the ASME - Journal of Basic Engineering*, volume 82(Series D), pages 35–45, 1960.
- [KM96] Ioannis A. Kakadiaris and Dimitri Metaxas. Model-based estimation of 3d human motion with occlusion based on active multi-viewpoint selection. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, pages 81–87, June 1996.
- [KYS01] N. Krahnstver, M. Yeasin, and R. Sharma. Towards a unified framework for tracking and analysis of human motion. In *Proc. IEEE Workshop on Detection and Recognition of Events in Video, Vancouver, Canada*, July 2001.

- [Lau94] A. Laurentini. The visual hull concept for silhouette-based image understanding. In *IEEE Trans. Pattern Analysis and Machine Intelligence*, volume 16(2), pages 150–162, 1994.
- [LC87] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proc. SIGGRAPH 87*, pages 163–169, 1987.
- [LSL02] Jason P. Luck, Daniel E. Small, and Charles Q. Little. Real-time tracking of articulated human models using a 3d shape-from-silhouette method. In *to appear in CVPRIP'2002, International Conference on Computer Vision and Pattern Recognition*, 2002.
- [May79] Peter S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press, Inc, 1979.
- [MBR⁺00] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 369–374. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
- [NH01] Kevin Nickels and Seth Hutchinson. Model-based tracking of complex articulated objects. In *IEEE Transactions on Robotics and Automation*, volume 17:1, pages 28–36, February 2001.
- [OB80] J. O'Rourke and N.I. Badler. Model-based image analysis of human motion using constraint propagation. In *IEEE Trans. Pattern Analysis and Machine Intelligence*, volume 2(6), pages 522–536, November 1980.
- [RST96] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3):117–149, 1996.
- [SCPMT00] Hyewon Seo, Frederic Cordier, Laurent Philippon, and Nadia Magnenat-Thalmann. Interactive modelling of mpeg-4 deformable human body models. In *Postproceedings Deform 2000*, pages 120–131. Kluwer Academic Publishers, 2000.
- [SG99] C. Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. In *Second IEEE Workshop on Visual Surveillance*, pages 246–252, June 1999.
- [Sze90] R. Szeliski. Real-time octree generation from rotating objects. Technical Report 90/12, Digital Equipment Corporation, Cambridge Research Lab, December 1990.
- [TD02] Leonid Taycher and Trevor Darrell. Range segmentation using visibility constraints. In *To Appear in International Journal of Computer Vision*, 2002.
- [WADP97] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Pentland. Pfunder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.
- [WB01] Greg Welch and Gary Bishop. An introduction to the kalman filter. In *Course 8 of SIGGRAPH 2001*, 2001.

- [Wel93] Chris Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Fraser University, 1993.
- [WGW90] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. In Rich Riesenfeld and Carlo Sequin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24-2, pages 11–21, 1990.
- [WP98] Christopher R. Wren and Alex P. Pentland. Dynamic models of human motion. In *In Proceedings of the Third IEEE International Conference on Automatic Face and Gesture Recognition, Nara, Japan, April 1998*.
- [WP99] Christopher R. Wren and Alex P. Pentland. Understanding purposeful human motion. In *In Proceedings of the IEEE International Workshop on Modelling People (MPEOPLE)*, pages 19–25, 1999.
- [ZWS02] Tao Zhao, Tianshu Wang, and Harry Shum. Learning a highly structured motion model for 3d human tracking. In *ACCV2002: The 5th Asian Conference on Computer Vision*, January 2002.