Automatic Observation and Synthesis of Human Motion

Thesis by

Luis Goncalves

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2000

(Submitted March 8, 2000)

To my Mom, and my Dad,
and my Brother and Sisters;
the ones that are always with me,
even though they are far away.

Acknowledgements

First and foremost, I would like to thank my advisor, Pietro Perona, for his constant support, insight, and optimism. I feel very fortunate to have worked in his lab, and have learned a lot from him. His extremely creative, analytical, and insightful approach to research has been a tremendous inspiration to me. Pietro also has the amazing ability to always look on the bright side of things, and thus inspire people and bring out the best in them. At times when my research seemed to be stuck in a black hole, he was there to pull me out. When Enrico and I first started thinking about modeling human motion to improve tracking robustness, we were on what one might call a wild goose chase. Pietro's support was none-the-less unfaltering. He let us wander, and when we decided to study the (somewhat unrelated to computer vision) problem of human motion synthesis, he embraced the idea with enthusiasm. I cannot imagine a better advisor. Thank you, Pietro!

Second, I would like to thank my great friend and partner in crime, Enrico Di Bernardo. The work described here is just as much his as it is my own. Without him, it would have taken four times as long and been one fourth as fun (our empirically confirmed square laws relating team size to performance and enjoyment). In fact, I think that without him the motion synthesis work would never have come to fruition. At a critical point in the research we decided that to progress further we would have to embark on the ominous task of building our own 3-D motion capture system. Together we took the bull by the horns and built a very well designed system (appendix B) in 4 short (and very intense) summer months in 1998. Without him, I would have never even contemplated such a task. Enrico and I work extremely well together. It has been a great, rewarding experience to do so, and I look forward to continued collaboration with him in the future.

Thank you also to my thesis (and candidacy) committee: Richard Andersen, Al Barr, Joel Burdick and Shuki Bruck. Their support and advice over the years is much appreciated. Al Barr in particular has been a great inspiration and friend, with his Socratic method of teaching, and willingness to discuss ideas.

I have had the good fortune during my years at Caltech to encounter a wonderful set of people who have become great friends. Life would not be the same without them and all the 'Missions' we shared. Thank you, Commander, AD, AC, ACW, CA, MA, A, DM, GE, BA, B, AA, YK, M!

Finally, I thank my family for their constant love, support, and encouragement.

Abstract

Over the past few decades Computer Vision and Computer Graphics have experienced a rapid evolution, thanks in part to the continual improvement in computer hardware, which enables the investigation of increasingly complex problems.

In Computer Graphics this evolution is visible on a nearly day-by-day basis. For instance, computer-generated special effects in feature films have evolved to such a level of sophistication that it is often impossible to distinguish what is real from what is not. However, one challenging problem that still stands, considered by many experts in the field to be a Holy-Grail of Computer Graphics, is the automatic synthesis of life-like human character animation. Although rendering and modeling techniques have reached a stage where a computer generated image of a person is nearly indistinguishable from the real thing, as soon as that model begins to move the illusion is broken. The problem is difficult because no-one yet knows how to model human motion in all it's intricacy and subtlety, and also because humans are so well tuned to perceive these subtleties that they can only be fooled if the modeling is done with complete perfection.

In this thesis, we explore a novel method of automatic synthesis of human motion that brings us one step closer to the ultimate goal. The method is based on decomposing human motion into elemental, nameable actions such as walking, running, and throwing, and using observations of people performing these actions to create mathematical models of the actions. Various samples of an action are acquired, and each sample is labeled according to state (initial body configuration), goal (desired outcome of the motion, such as direction of a throw or placement of a foot for a step), and mood & style parameters. Then established and novel techniques of machine learning are applied to derive a function that can synthesis a motion given some desired parameters. We explore the use of polynomial interpolants, radial basis function networks (RBFs), feed-forward neural networks (FFNNs) with sigmoidal activation functions, as well as a new method with local linear models. We find that a linear model more often that not works quite well, whereas higher order polynomial interpolants, RBFs and FFNNs are unable to extrapolate robustly when the motion parameters lie outside of the convex hull of the parameters of the available sample motions. The method with local linear models successfully improves the fidelity of the synthetic motions compared to the linear model, and also provides robust extrapolation. We also investigate the use of a recursive, probabilitic model where motions are specified by defining the initial and final body poses of the motion, and synthesis is done by computing the most likely motion to satisfy the boundary constraints. Although the results with this method are not yet completely satisfactory, it holds promise, and under certain types of conditions can re-synthesize the sample motions more accurately than any of the other methods.

With the additional development of methods to smoothly concatenate actions together and to interactively map synthesized motions to a 3-D polygonal character model, a real-time interactive demo was created that successfully demonstrates the level of realism and interactivity achievable by our method of human motion synthesis.

Our interest in the problem of realistic human motion synthesis arose from an initial study of the (in some sense) inverse problem in Computer Vision of the automatic observation (rather than synthesis) of human motion. Although progress in Computer Vision has not yet reached a level enabling it's widespread use in daily life, this state will most likely be achieved within the next decade. One large class of problems for which this is the case is the endowment of computers with visual perceptual skills similar to those of humans. Among the vast set of visual tasks imagineable, the automatic detection, recognition, and estimation of humans and human motion is a particularly interesting set of problems since there are many possible applications of such a technology in modern life, ranging from security and monitoring systems, to systems for biometric analysis, to novel human-machine interfaces.

In this thesis we describe a method of robustly estimating the motion of a human body from a monocular view. The method is based on the use of a 3-D model of the body, and comparing the actual image to an expected image based on the 3-D odel to update the estimate of the body pose at each time step. The method was implemented in real-time as a human-machine interface. This system demonstrated that the method can be used to robustly track a human arm with a hand-tip positioning resolution of 2cm under close viewing conditions (where perspective projection causes significant changes in the appearance of the arm in the camera view).

Contents

List of Figures

List of Tables

Part I

Observation

Chapter 1 Automatic observation of human motion

1.1 The applications of automatic observation of human motion

There are many applications and environments where it would be beneficial to be able to observe the motion of humans automatically. The three broad categories of applications are those of monitoring, bio-metrology, and interfaces.

Monitoring and Security

Often it is necessary to monitor the activity of people in order to assure their safety and to preserve the environment itself. Camera-based security systems are commonly installed in parking lots, museums, and other high-risk environments. Unfortunately, these systems currently require a human operator to be present in order to perform the tedious task of monitoring the human activity. If methods for automatic observation (and recognition) of human motion could be developed, this tedious job could be eliminated, or at least made more efficient (less human observers required per unit area) and reliable (no important events missed due to inattention).

Beyond replacing humans in current monitoring systems, there are many new applications that could be made practical. For instance, traffic at busy city intersections could be monitored to optimize traffic lights, making pedestrians lives easier. Active safety systems on board vehicles could be used to monitor the behavior of surrounding pedestrians. In stores, besides preventing shoplifting, automatic observation could be used to collect statistics on customer behavior, seeing where they spend most of their time in the store, and which displays attract their attention the most.

Bio-metrology

In medical settings, and in situations involving top-performance athletes, it is often necessary to measure body motion accurately in order to assess a physical disability, or to optimize athletic performance. Systems capable of making such measurements are already in existence today; however, they are cumbersome to use. They require the placement of special markers and/or measuring devices on the body, which can even restrain the motion being measured. The techniques described in this thesis can be used to build systems capable of accurately measuring the motion of unencumbered bodies, making the practice more widely applicable.

In the entertainment industry, similar systems (called 'motion capture systems') are also used. Typically, an actor wears a special suit (tight, body-fitting Lycra, the 'rubber suit') fitted with markers, and performs various needed motions. The motions are registered ('captured'), and later used with sophisticated 3-D computer animation software to create virtual humans, robots, or monsters that move in a very realistic manner. In this scenario, the actor's task could be made much easier if he didn't have to wear a special suit with markers. Furthermore, if a marker-less motion capture system could be developed, it would give directors more creative freedom - it would be possible to capture the motion of anyone that has ever been filmed before, such as Marilyn Monroe or Charlie Chaplin.

Also for scientific study of human and animal behaviour, such as the interaction of a mother and child, or the social interactions of a group of primates, it would be highly desirable to have a non-invasive method of collecting data over prolonged periods of time.

Human-machine interfaces

The third broad area of application for automatic observation of human motion is that of human-machine interfaces. There are many situations where a keyboard and mouse are not satisfactory input devices. For example, in immersive virtual environments, where the user needs to interact with the virtual world, gesturing and other body motions are a natural way to interact. Once again, it would be ideal if the user did not have to wear any special equipment. Some 3-D design software could also benefit from simpler, more adapted interfaces. For example, many companies now analyze the ergonomics of products on the computer, rather than building a mock-up of their design (ex. plane cockpits). This means that a virtual human has to be inserted in the design, to test for instrument visibility, pilot comfort, that all switches and controls are easily reached, and that all parts are easily accessed and removed by maintenance crews. The keyboard and mouse are poorly adapted for manipulating a virtual human, but if the user's head and torso rotations, as well as arm movements, can be observed by the computer, a much more efficient and natural interface arises. As a last example, we can also point out that young children may not be able to use a mouse and keyboard effectively, but they certainly know how to point at things and clap their hands.

1.2 Focus of this thesis

Although the concepts and techniques discussed in this thesis can be applied to any of the application areas mentioned above, we will focus on the area of human-machine interfaces. This focus allows us to actually implement a mock application, with which we can test the limitations of the approach, and gain an understanding of what assumptions are necessary and what simplifications can be made while retaining a highly functional interface.

Any system for automatic human motion observation has to accomplish two tasks: initialization, and tracking. When the machine is first turned on, or when a human first enters the field of view, the computer must detect whether or not a human is present, and if one is present, it must obtain an initial estimate of the pose of the body. The body pose is defined in terms of a set of parameters which define the global position and orientation of the body with respect to a world coordinate frame, as well as the internal configuration of the body (how much each body segment/limb is bent). Once a body is detected and an initial pose estimate made, the computer can then proceed to track incrementally the pose changes in time. The two problems are quite distinct, since the assumptions under which a body pose is being estimated are quite different in the two cases. A method for initialization will most likely not be an efficient way to track, since it does not make use of the fact that the previous frame's pose estimate is a good approximation of the current pose.

In this thesis, we concentrate on solving the tracking problem*. The initialization problem is circumvented by having the user start from a predetermined body pose. Ultimately

*And so we will refer to our system as a 'human motion tracking system', or 'tracking system', or 'tracker', for short.

the initialization problem must also be dealt with properly; this is the subject of another thesis in our laboratory.

1.3 System design considerations

In designing a human motion tracking system, there are several choices and assumptions that have to be made. In this section we present the various design aspects to consider, the possible solution choices, and the choice made for our system. Many of the possible choices arise from a direct analysis of the problem, others from studying the approaches of other researchers. At the end of this section a comparison of the approaches of various research groups is presented.

The topics that are important in developing a tracking system are:

- General system characteristics: a clear definition of the final output and form of use of the system, from which specific criteria for evaluating performance can be defined.
- Choice of representation: how will the observation process and the observed body be modeled, with what level of detail and simplifying assumptions.
- Measurements to make: what information should be extracted from the images, and how.
- Detection/Initialization algorithm: how is the initial body pose detected and estimated.
- Tracking algorithm: what is the general scheme used for tracking.
- Numerical techniques for parameter estimation: what are the numerical techniques that are used to estimate body pose given the image measurements obtained.

1.3.1 General system characteristics

It is important to have, from the onset, a clear definition of the purpose and function of the tracking system. This sets some clear design principles and goals, and defines criteria by which the system can be evaluated. For example, the specific information to be extracted from the image stream, and the accuracy of the estimation should be defined. In our case, we wanted to be able to estimate the 3-D pose of a human arm with enough accuracy to allow the user to manipulate virtual objects (with approximately 1 cm resolution).

One obvious and fundamental characteristic should be that the system performs robustly. That is, the system should be able to cope with a normal range of speeds and motions, and under normal (not very simplified or controlled) imaging conditions, without losing track of the person. This is in fact very difficult to achieve, a sort of asymptotic goal. It is surprising, though, how often it happens that research papers do not explicitly state nor demonstrate the robustness of a proposed system.

One reason why it is often hard to determine the robustness of a system is due to the fact that the system is not implemented in real-time. With an off-line implementation, it becomes quite difficult and cumbersome to test the behavior of the system thoroughly. A real human-machine interface, of course, is by definition a real-time process. In order to gain a deeper understanding of how well our system works, it was decided that our system should be implemented in real-time.

Finally, care should be taken that the system is not designed around any critical, undue assumptions without which the method cannot work. In designing a complex system, it is usually necessary to simplify the problem in order to make it tractable. Eventually, these simplifying assumptions should be removed and the more complicated scenario dealt with. It would be unwise, say, to detect the initial presence of a person by assuming a particular skin color (unless it is known that the algorithm can be generalized to other skin colors and function equally well).

1.3.2 Choice of representation (model)

Many choices have to be made as to how to model the observation process and the body being observed.

Perspective vs. orthogonal projection

A choice as to whether to interpret the image as an orthogonal projection or as a perspective projection needs to be made. Real cameras generate images with perspective projection. However, in some situations it may be possible to simplify the projection model. For example, in a monitoring application where a large area is being viewed and the camera is mounted high above the ground, a scaled orthographic projection can be used. Wherever the human may be located in the field of view, the relative variation of depth (distance to camera) of different body parts is small. Thus the depth of all body parts can be assumed to be a global value Z, and the appearance of the body in the camera image is accurately modeled as a scaled orthographic projection.

Scaled orthographic projection has the advantage that it is a linear projection model, resulting in simpler equations to work with. This is the main reason why it is often chosen over (the non-linear) perspective projection as a camera projection model whenever possible. In our case, though, we need to use perspective projection because the user is close to the camera and the relative change in depth of the hand, say, during different poses can be very large.

2D vs. 3D model

For some applications, such as monitoring applications, it may suffice to use only a 2D model of the human body, so that only the position of the body on the 2D screen is known. For other applications, such as our human-machine interface where the user is expected to be able to manipulate virtual objects in 3D, a 3D model of the human body needs to be used.

Number of DOF to estimate

A choice needs to be made as to how complicated a model to use to represent the body and its motion. The simplest model is to just represent the body by its 2-D screen position (or 3-D position in space, if working in 3-D). Depending on the application, this could in fact be all the information necessary (ex. a smart door or elevator, or shopping cart).

A more sophisticated model may incorporate parameterized models of particular motions. For example, the motion of a walking human can be parameterized quite accurately by a single parameter. This parameter denotes the phase of the walk within the standard human walk cycle. With this type of model, given the assumption that the human is waking, the motion can be accurately estimated.

The most difficult case is the one where the motion of the observed human is apriori unrestricted. In this case, a sufficient number of degrees of freedom need to be modeled in order to be able to represent any possible action.

Estimate positions/angles and/or velocities

Depending on the measurements extracted from the image stream, and on the estimation techniques used, it may be possible and/or necessary to estimate both positions of different body parts as well as their velocities. In our system, both the joint angles of the shoulder and elbow, as well as the joint velocities, are estimated. This is done because knowing position and velocity aids in predicting where the arm will move to next, making the tracking process more robust.

Simplifying assumptions

Despite the fact that the ideal system should be completely automatic, and able to function properly under wildly varying circumstances, strategic assumptions are often made to make the problem tractable when developing a system. These assumptions should be removed at a later stage of development. Possible assumptions are:

- special clothing on the human
- known starting pose and location of the body
- known geometry of the body
- static background

1.3.3 Measurements to make

Given a stream of images, there are several different types of information that can be extracted and used to estimate the model's DOFs.

- stereoscopic information from two or more cameras.
- foreground/background segmentation based on color, image differencing, or texture detection. This can define the outline of the body.
- edge detection: The outline of the body can be detected.
- point feature detection: Certain easily-identifiable points on the body can be individually detected and tracked throughout the image sequence.

- optical flow: The apparent 2-D velocity on the screen resulting from the 3-D motion of the body can be estimated for the entire image.

1.3.4 Detection/Initialization algorithm

Since the tracking algorithm assumes that an accurate estimate of the body pose is available in the previous frame, such an accurate estimate must somehow be obtained at the start of the tracking session. In our system the problem is solved by having the user start in a pre-determined position. The tracking system converges to the exact pose given that the arm is nearby the pre-determined pose in configuration space.

1.3.5 Tracking algorithm

The algorithms used to estimate the model parameters from frame to frame can be categorized into three broad categories:

- 2-tier approach

  The simplest approach is to break down the tracking algorithm into two separate steps:

  - 1: extract features from the image
  - 2: estimate parameters from features

  This method has the disadvantage that it is an open-loop process, relying on the correctness of the extracted features. The method can fail if the wrong feature is detected and used in the estimation step.

  The method also has the disadvantage that the optimal values for the individual features may result in a sub-optimal estimation of the model parameters. For example, suppose that we are looking for squares in the image, and we use edges as our elementary features. Once the positions of four edges have been detected, they can be joined together to estimate the position and size of the square. However, due to possible shadows and low contrast between the square and the background, the estimated positions of the edges done individually will most likely be different than the estimated positions that could be derived if knowledge that the four edges are part of a square were used during edge detection. In the latter case, a more accurate estimate of the square position will result.

Another disadvantage of this method is the high computational cost associated with searching for features throughout the entire image, as well as the potentially large combinatorial search to select the correct set (and order) of features.

- Loose feedback approach

  This approach improves upon the 2-tier approach by using the currently estimated model parameters to predict where the image features should be in the next frame:

  - 1: extract features from image
  - 2: estimate parameters from features
  - 3: use the estimated parameters to predict where the features should be in the next time step.

  Thus a 'loose' feedback between feature extraction and parameter estimation is created. This should help avoid the problem of extracting the wrong feature in the subsequent frames (assuming correct initial conditions). This approach also has the added benefit that it reduces the computational cost of feature extraction, since processing of the entire image is avoided.

  However, this technique still suffers from the problem of creating sub-optimal parameter estimations.

- Direct matching (implicit features) approach

  The final possible tracking technique is the one that is shown through experimentation to be the most robust and accurate of the three, and is the one we use in our system. In this approach, image properties are encoded directly as a function of the model parameters. For a given set of values of the model parameters, an 'expected' image can be generated. The parameter values are optimized such that the expected image matches the actual image as closely as possible. One way to think of this technique is that the features are detected implicitly.

  In the example of tracking a square in an image stream, the model parameters would be the position of the center of the square, the orientation of the square, and the length of a side. Based on the values of the parameters, an 'expected image' with the expected location of the sides of the square can be generated. The optimal parameter estimates are then found by finding the parameter values that cause the expected sides to lie on image regions with the highest image intensity gradient. Implicitly, four edges have been detected, incorporating the knowledge that the edges form a square.

1.3.6 Numerical techniques for parameter estimation

Irrespective of which tracking algorithm is used, there are also several choices for how to use the measurements to estimate the model parameters (or do the direct image comparison) at each time step:

- Local search for minima (for discrete parameter space)

  If the parameter space is discrete, the best estimate of the model parameters can be found by a local search that maximizes the fit to the extracted features (or direct image matching). The parameter space may be discretized for the sake of computational efficiency; if the parameter space is highly non-linear, it may be desirable to discretize and search rather than compute a complicated gradient function and attempt a continuous minimization. However, there is an inherent trade-off between the resolution of discretization and the computational cost of optimization. For some applications, the required accuracy may be such that discretization is not viable.

- Gradient descent methods (continuous parameter space)

  If the parameter space is continuous, gradient descent methods such as those of conjugate-gradients or Levenberg-Marquardt can be used. Starting from an initial guess of the parameters (based on the previous time step), the parameters are iteratively modified to maximize the match with the extracted features (or direct image match). The usage of such iterative techniques involves the definition of criteria to decide when to stop the optimization. These are typically based on the desired level of accuracy and a maximum amount of time/iterations permitted per time step.

- Optimal filtering

An alternative way to estimate continuous parameters is to make use of a Kalman Filter [1]. This approach inherently takes into account the time dimension of the tracking problem, by inclusion of a dynamical model of the evolution of the model parameters. Also, because of its recursive formulation, multiple iterations at each time-step are not necessary, making it suitable for real-time implementations. For linear problems with a Gaussian model for the noise in the measurements and the uncertainty in the dynamical modeling of the system, a Kalman filter provides the optimal estimate of the system state. We use a Kalman filter in our approach, even though our system is highly non-linear (joint angles define the pose of the body, and perspective projection defines the apparent shape in the image). An Extended Kalman filter (where the system is linearized around the current operating point at each time-step) is used instead. Also, since our system does not extract explicit measurements (as required by the typical Kalman filter method), but rather direct image matching is done instead, another modification of the Kalman filter is required. Implicit measurement equations are used, in what is known as an implicit Kalman filter [1]. (See appendix A for a detailed description of a Kalman filter and its variants).

- Condensation filtering

  Recently a new type of estimation technique called the 'condensation filter' has been developed [21]. It is similar to the Kalman filter, in the sense that a model of the dynamical evolution of the system being tracked can be incorporated into the estimation process. However, unlike the Kalman filter, a non-parametric representation of the probability of the system state is used. In this way, multi-modal distributions can be represented.

  At each time step, the state probability distribution of the system is evolved according to the stochastic system dynamics equation. Then, many sample states are drawn from the probability distribution. The amount of evidence existing in the image for each sample state is measured. Finally, given the a priori probability of each sample state and the amount of evidence in the image, the overall probability distribution of the system state is re-estimated. At any instance in time, the best guess of the state of the system is given as the state with maximal probability (density).

The claimed advantage of this estimation technique is that it has increased robustness to noise and scene clutter, as compared to the Kalman filter. The reason for this is that whereas the Kalman filter can only keep track of a single mode (of a Gaussian distribution) to represent the system state, here any (potentially multi-modal) distribution can be represented. This means that several likely states are explored at each time step. Thus, while the Kalman filter may get fooled by an extraneous feature (a shadow, say), and then lose track of the object, the condensation filter would track both simultaneously and will be able to recover from the error once the image changes.

This technique is usually used with the implicit feature algorithm, where the boundary of the object to be tracked is encoded by the (2D) model state parameters. The measurements (image evidence) accumulated for each sample state is the sum over the expected contour of how well the expected contour lies on a large image gradient (edge).

Although we have only experimented briefly with condensation filters, it seems that one of the major difficulties in implementing them is the number of sample states that need to be generated at each time-step. In effect, the state probability density function (pdf) is represented by the distribution of the samples in state space. Thus, the number of samples required to obtain an equivalent accuracy in the pdf depends exponentially on the number of degrees of freedom of the system (or at least on the dimension of the 'likely' sub-manifold within the state-space). A 1 dof system needs approximately 50 samples to properly estimate the pdf, so that our 4 dof arm model would need $50^4 = 6250000$, a number too large for practical use. Even if one argues that the arm really only has 3 effective degrees of freedom (because you reach positions in 3-D, but for each reach position there is a standard arm pose), we would still need on the order of $50^3 = 125000$ samples.

1.4 Comparison of existing methods

Based on the ultimate goal of the system and different choices for the design considerations described above, various groups have developed quite different human motion tracking systems. Here we present brief descriptions of the work of the pioneers in this field. Our own system stands out as being the only one able to handle scenes where perspective projection needs to be taken into consideration (the human is close to the camera), and able to do so utilizing only a monocular view.

Rohr [32]

In 1993, Rohr [32] developed a system to track a person walking. To simplify the task, the person was assumed to be walking parallel to the image plane, and to be of a known height. With these assumptions, it is possible to parameterize the walking motion with just one parameter; the instantaneous phase of the person in the canonical walk cycle. This canonical walk cycle was previously known from biomedical studies, which showed that the coordinated movement of the torso and limbs is quite consistent amongst individuals. Rohr modeled the body as a set of links of rectangular body segments. By measuring angle and distance errors between the rectangular body section to the nearest image edges, a Kalman Filter could be used to update the estimate of the walking phase. The main disadvantage of this system is that it is constrained to tracking a specific motion (walking), and only in 2D (fronto-parallel walking).

Gavrila & Davis [12]

In 1996, Gavrila & Davis [12] presented a system to track the unconstrained movement of the human body which is represented by a 3D model with 22 degrees of freedom. To estimate the 3D pose, multiple cameras (3 or more) were used. To facilitate measurement extraction, the subjects wore special suits where each limb was of a different color. The pose space was parameterized discretely, and pose estimation was done by performing a hierarchical parameter search to minimize error in matching the edges of the model to those in the images. The main disadvantages of this system are that the subject needs to wear a special suit, and that the discretized pose space limits the resolution of the tracking.

Darrell et al. [37]

In 1997, Darrell et al. [37] presented a system that could track the 2D or 3D position of the human body. Flesh color detectors were used to detect the position of the hands, head, and feet (with 1 camera, 2D information is obtained, with multiple cameras, 3D positions are obtained via triangulation). The pose of the body itself is approximated, by imposing kinematic constraints and using inverse kinematics. The main disadvantage of this system is that only limited information of the body pose can be obtained; by only observing the head, feet, and hands, only a guess of the body pose can be derived.

Wren and Pentland [38]

In 1998, Wren and Pentland [38] improved upon the system of Darrell et al. to estimate the pose of a 30 dof torso with 15 dof (kinematic) constraints. 'Blob trackers' were used that not only could detect the position of the head and hands, but also estimate its orientation (using the statistics of the ellipsoidal blobs). The position and orientation of each blob was tracked with a (separate) Kalman Filter, and the estimated poses were fed to a solver that would determine the torso's 30 dof given the kinematic constraints. The main disadvantage of this approach is similar to that of Darrell et al.; even with the extra orientation information of the head and hands, it is still only possible to guess the body pose and not infer it exactly.

Bregler [8]

In 1998, Bregler developed a method of tracking the entire body (with DOFs at major joints) where no special clothing is necessary. Scaled orthographic projection is assumed (the person far away from the camera), and the body is modeled with ellipsoidal rigid links. Tracking is achieved by implicitly measuring the optical flow between consecutive images, and expressing that flow in terms of the joint velocities. To do so requires associating image regions to the appropriate body segement, and this is done concurrently via an EM-based algorithm which segments the image based on how well each link's optical flow corresponds to the image data. The method can work with either a single camera or multiple views, multiple views generating more robust tracking since it reduces the number of body part occlusions. Due to the fact that optical flow is measured implicitly, and it is directly related to the joint velocities, the method can work even though a direct estimate of the image's optical flow is usually difficult and ill-conditioned. The main limitation of this technique is that since tracking is achieved by integrating the estimated joint velocities through time, errors in joint angle can accumulate and eventually the tracking is lost.

Goncalves et al. [16], [3]

In 1995 we developed a system capable of tracking the motion of a human arm with 4 dof using a monocular view. Perspective projection was used to model the appearance of the arm in the image plane since the user was assumed to be close to the camera. A Kalman Filter was used to directly estimate the arm's dofs by implicitly estimating the edges of the arm's silhouette. The silhouette was obtained from the image by performing a foreground/background segmentation based on pixel color statistics. The main disadvantage of the system is that a fairly accurate model of the arm shape is required.

Current trends

With the advent of more powerful computers, human tracking systems of increasing complexity are being developed and implemented in real-time, with increased robustness and reliability. One such example is the Who-When-Where-What (W^4) system of Davis et al. [17]. This system, through a series of robust heuristics, can segment people from the background in a monochromatic image sequence, and track them even in the presence of self and inter-person occlusions. Rather than estimating a large number of degrees of freedom to define the 3D pose of the body, a simple 2D connected blobs model is used to represent the major body parts. The premise of the system is that this representation is sufficient to extract higher-level information on what actions the people being tracked are performing.

Another interesting trend is to avoid the use of exact 3D models of the body, and instead to use probabilistic inference and machine learning techniques to detect a person [34] and estimate their 3D pose [25] from a single view.

Chapter 2 Monocular tracking with implicit measurements

In this chapter, we describe the details of our body tracking system. We begin with a review of the goals and design principles of the system, and then present a detailed description of the assumptions, models, and estimation methods involved.

2.1 Goals and design principles

As previously mentioned in section 1.2, our system was designed in the context of a human-machine interface; a new way of manipulating 3-D data. Examples of applications where this may be useful are 3-D CAD programs, human factor analysis (ergonomic design and testing of products), and navigation/interaction in 3-D virtual worlds.

Given that our system is to be used as a human-machine interface for 3-D interaction, we therefore need to be able to estimate at least 3 degrees of freedom in the human body. We certainly want our method to be extendible to estimate many more degrees of freedom (for, say, detecting the pose of the entire body when the user is immersed in a 'Star-Trek holodeck'-type environment), but at least three dofs must be estimated.

We chose to use only one camera in our system, rather than obtaining stereoscopic views, in order to explore the limits of what information can be extracted from a monocular view. It is certainly challenging to try to estimate the 3-D pose of an object given only one viewpoint, but is it impossible? It turns out that if you have a good 3-D model of the object being observed, it is possible to recover the pose from a single view [27]. Thus one consequence of our choice of using only one camera is that we will need to have a 3-D geometrical model of the human body observed.

As discussed in section 1.3.1 on general system characteristics, an important goal for any system is robustness of operation. Making our system as robust as possible influenced many design choices.

First, a direct image matching approach for tracking was developed, since this formalism is clearly more robust than the 2-tier or loose-feedback approaches. Second, a Kalman filtering scheme was created to perform the parameter estimation since it can include a dynamical model of the system being observed, thus potentially increasing the robustness of the system (depending on whether or not a good predictive dynamical model of human motion can be found). Finally in order to gain good insight into how well the system functioned, to test the system robustness, and to be able to experiment and tune it easily, it was decided that our system should be implemented in real-time.

The decision for real-time implementation also influenced other design decisions. Namely, the algorithms used must be as computationally efficient as possible in order to achieve a high real-time frame rate, so that it would most likely not be viable to search over the entire image to detect low-level features, nor would it be viable to use an estimation method which required many iterations for convergence. Fortunately, the use of a Kalman filter and the direct image matching technique both satisfy these conditions. The Kalman filter does not use multiple iterations to generate a pose estimate, and by performing direct image matching, the number of low-level image computations necessary is minimized (as will be explained in detail later).

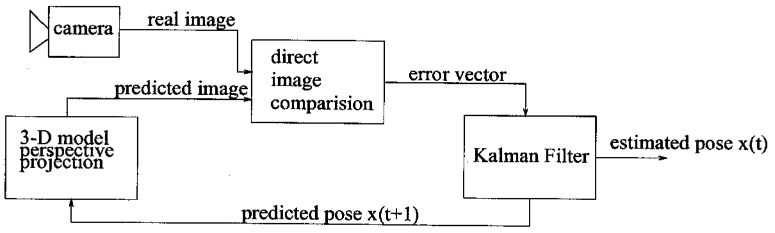


Figure 2.1: The system block diagram for the tracking system.

Figure 2.1 shows a schematic of the overall tracking system. The next sections will describe each component in detail.

2.2 Technique

2.2.1 Assumptions and simplifications

Given the goals of robust, real-time estimation of body pose from a monocular view, some initial simplifying assumptions were made in order to test the general approach on a tractable problem.

Body (Arm) model

A first simplification is to not try to estimate the pose of the entire body right from the start, but to first test the methods by trying to estimate the pose of an arm. After a successful technique is developed, more of the body can be modeled and tracked.

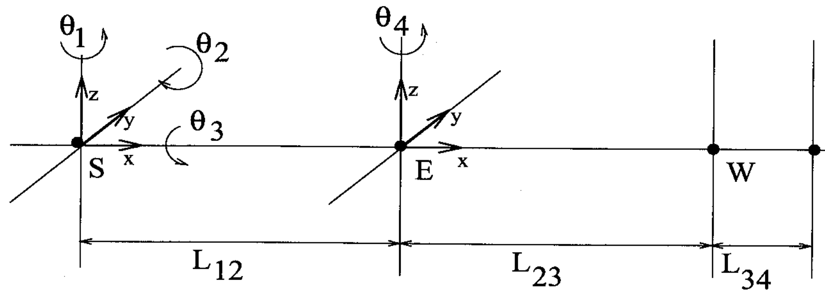


Figure 2.2: The kinematic arm model: There are 3 degrees of freedom at the shoulder, one at the elbow, and the wrist is assumed to be stationary.

The arm model itself will also be quite simplified. Figure 2.2 shows the kinematic arm model. The shoulder has 3 rotational degrees of freedom which pivot around a single point. The elbow has one degree of freedom, and for simplicity, the wrist is modeled with zero degrees of freedom; the hand is assumed to be held open and parallel to the forearm.

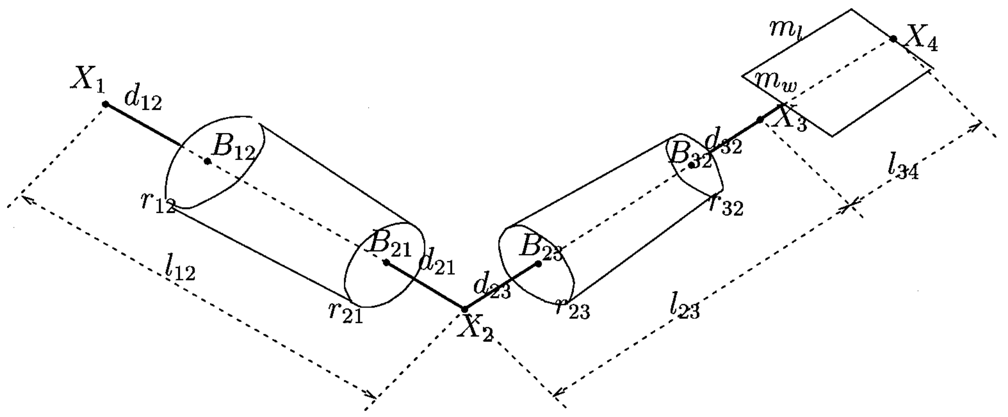


Figure 2.3: The geometric arm model: The upper and lower arm are modeled as truncated cones, with a distance of 5cm between the top and bottom of the cones and the joint centers. In all there are seven parameters to fit the arm of a person.

Figure 2.3 shows the associated 3-D geometrical model of the arm. The upper and lower arm are modeled as truncated cones. The top and bottom of each cone is assumed to be some distance ($d'_{ij}s$ in the figure) away from the respective joint centers, since near the joints a rigid conical shape is not very accurate. To define the arm model of a person, 7 parameters are needed: the lengths of the upper arm, lower arm, and hand (L_{12}, L_{23}, and L_{34} in the figure), and the upper and lower radii of the truncated cones ($r_{12}, r_{21}, r_{23}, r_{32}$). In the future, some automatic way of estimating the arm shape could be developed, but currently these values must be estimated by hand with an accuracy of approximately 1cm.

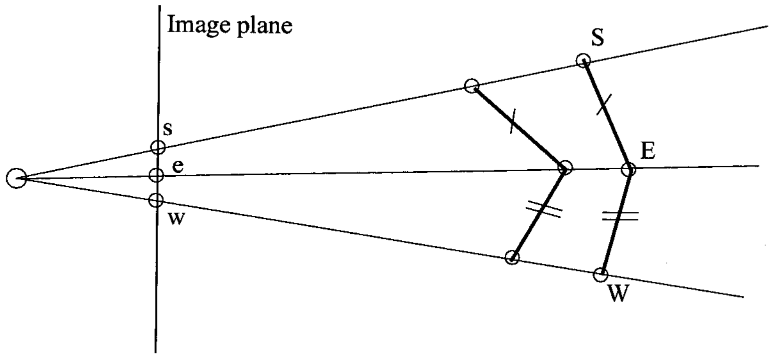


Figure 2.4: If only the joint projections on the image plane were observed, for known upper and lower arm lengths there is a continuum of poses at different depths that produce the same image projections.

Since we are only tracking the movement of an arm, and doing so from a monocular view, it is necessary to define the depth (distance from camera) of the shoulder. As figure 2.4 shows, supposing that we know the projections of the shoulder, elbow, and wrist in the image plane, there is a range of shoulder depths over which the arm can be configured to fit the shoulder, elbow and wrist projections. In reality we do not observe joint positions on the image plane, but rather the entire arm contour, so that changes in apparent size due to perspective projection could be used to determine the depth. However, this would most likely be a very sensitive measurement, resulting in uncertainty in depth on the order of tens of centimeters. To further simplify the tracking problem, we assume that we not only know the depth of the shoulder, but the actual 3-D position in space. This means that the user will have to sit in a particular position so that the shoulder is at the correct 3-D location. Once more of the body is tracked, this restriction can be eliminated.

Camera model

In order to estimate the pose of the body it is necessary to relate the pose parameters (joint angles for the 4 degrees of freedom) to the observed image of the arm. As discussed previously, since in our initial application the user will be sitting right in front of the computer (and the camera placed nearby the computer in order to retain good 3-D resolution), the depth of the hand can change significantly throughout time, so that a perspective projection model must be used.

The camera's intrinsic (focal length, image center, radial distortion factor) and extrinsic parameters (position and orientation of a global reference frame with respect to the camera) also need to be known in order to compute the arm's 3-D to 2-D projection. The appendix section B.2.1 describes the method of estimating the intrinsic and extrinsic camera parameters.

Scene and tracker assumptions

We make a few assumptions with regards to the scene being imaged too. First, that there is one person seated in the proper position for their arm to be tracked. Second, that nothing in the background near the person (in the image) moves. Third, the user's skin and clothing should not be the same color as the nearby background. These last two assumptions are necessary for the foreground-background segmentation (to be described below) to work properly. Finally we assume that the user's clothing is not too loose, so that our arm model is a close representation of the actual appearance of the arm.

Finally, we also make simplifications as to the functioning of the tracker proper. Namely, we circumvent the problem of automatic tracker initialization by having a default start position, and providing the user with a keyboard control button which signals to the tracker that the arm is in the proper position and to start tracking.

2.2.2 3-D arm model pose and 2-D appearance computation

In this section we describe how to compute the 3-D pose and the 2-D projection of the arm model onto the image plane. These computations will be done at every time step to extract measurements from the real image.

The computation is done in four steps:

1. compute the 3-D arm pose (3-D position of the shoulder, elbow, wrist, and hand-tip) in the shoulder reference frame

2. transform arm pose to camera reference frame

3. compute the approximate arm contour (the projection of the 3-D geometrical arm model) in homogeneous coordinates

4. Convert the arm contour projection from homogeneous to screen coordinates

Step 1: Compute 3-D arm pose in shoulder reference frame

Since we assume that the wrist is held rigid and that the shoulder is at a known position in 3-D, the pose of the arm is described by 4 rotational degrees of freedom, represented by the joint angles $\theta_1, \ldots, \theta_4$. The first three are degrees of freedom of the shoulder, and the fourth is of the elbow, as per figure 2.2.

We use the composition of exponential maps formalism [28] to compute the 3-D pose of the arm as a function of the joint angles.

Working in the reference frame of the shoulder, the location of a point P on the forearm (and hand) after rotations through angles $\theta_1, \ldots, \theta_4$ is given by

$$P(\theta_1, ..., \theta_4) = g_1(\theta_1) g_2(\theta_2) g_3(\theta_3) g_4(\theta 4) P_0 \tag{2.1}$$

where P_0 is the initial position of the point (expressed in homogeneous coordinates) when all the joint angles are zero, and

$$g_1(\theta_1) = e^{\hat{\xi}_1\theta_1} = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$g_2(\theta_2) = e^{\hat{\xi}_2\theta_2} = \begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_2 & 0 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$g_3(\theta_3) = e^{\hat{\xi}_3\theta_3} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_3 & -\sin\theta_3 & 0 \\ 0 & \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

$$g_4(\theta_4) = e^{\hat{\xi}_4\theta_4} = \begin{bmatrix} \cos\theta_4 & 0 & \sin\theta_4 & -L_1\cos\theta_4 + L_1 \\ \sin\theta_4 & \cos\theta_4 & 0 & -L_1\sin\theta_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$\tag{2.6}$$

where $\xi_1, \ldots, \xi_4$ are the twists associated with each degree of freedom and are defined as

$$\hat{\xi}_1 = \begin{bmatrix} \hat{e}_z & 0_3 \\ 0_3' & 0 \end{bmatrix}, \hat{\xi}_2 = \begin{bmatrix} \hat{e}_y & 0_3 \\ 0_3' & 0 \end{bmatrix}, \hat{\xi}_3 = \begin{bmatrix} \hat{e}_x & 0_3 \\ 0_3' & 0 \end{bmatrix}, \hat{\xi}_4 = \begin{bmatrix} \hat{e}_x & \begin{bmatrix} 0 \\ -L_1 \\ 0 \end{bmatrix} \\ 0_3' & 0 \end{bmatrix} \tag{2.7}$$

where e_x, e_y, e_z are the unit vectors of the shoulder coordinate system.

Thus the expected positions of the shoulder, elbow, wrist, and hand-tip in the shoulder reference frame are

$$\begin{aligned} S_s &= 0_3 \ (origin) && (2.8) \\ E_s &= g_3(\theta_3)g_2(\theta_2)g_1(\theta_1)E_0 && (2.9) \\ W_s &= g_4(\theta_4)g_3(\theta_3)g_2(\theta_2)g_1(\theta_1)W_0 && (2.10) \\ H_s &= g_4(\theta_4)g_3(\theta_3)g_2(\theta_2)g_1(\theta_1)H_0 && (2.11) \\ & && (2.12) \end{aligned}$$

where E_0, W_0, H_0 are the coordinates of the elbow, wrist, and hand-tip in the initial arm configuration (all joint angles zero).

Step 2: Change from shoulder coordinates to camera coordinates

From the camera calibration procedure, the translation from the shoulder reference to the ambient (calibration pattern) reference, T_{as} is known, as well as the rigid transformation from ambient to camera coordinates, (R_{ca}, T_{ca}), so for each joint we can compute $X_c = R_{ca}(X_s + T_{as}) + T_{ca}$.

Step 3: Compute approximate arm contour

Given the 3-D coordinates of the joints in the camera reference frame, we can now compute the image appearance of the arm model. In principle, one could derive a closed form solution for the occluding contour of a cone as observed from a specified viewpoint. However, the closed form expression for the case of perspective projection is quite difficult to compute. Another alternative is to finely sample the 3-D shape and 'render' it; compute the corresponding 2-D image points, and then detect the border of the rendered region. This is a rather inefficient method. To achieve a balance between speed and accuracy of solution, we approximate the calculation of the 2-D boundary of the arm model in the following way:

1. For each arm part, compute the perspective projection of the top and bottom points of the centerline of the truncated cones (points T and B project to t and b in figure 2.5).

2. Compute the scaled radii r of the top and bottom of the cones by dividing the 3-D radius by the depth of the respective points T or B. The top and bottom points of

the contour are drawn on a line perpendicular to the projection of $T - B$ on the image plane (see figure 2.5).

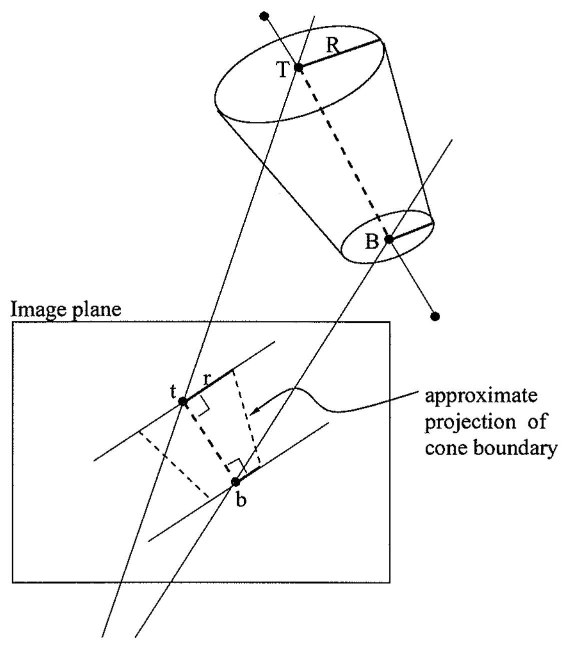


Figure 2.5: An approximated projection of the truncated cones to find the cone boundary. The top and bottom points of the cone axis are projected, then the scaled radii are drawn perpendicular to the projection of the cone axis.

Figure 2.6 shows the approximation incurred by simply scaling the diameter of the cone to compute the projection of the border. In this top view, one can see that if the cone is close to the image plane, using the perpendicular diameter will slightly underestimate the width of the projection. However, this error is noticeable only if the arms's distance from the camera is comparable to its width. This situation does not occur in practice; the arm is always much further away from the camera.

Step 4: Convert projection to screen coordinates

The above perspective projection was computed in homogeneous coordinates. The camera intrinsic parameters (focal length f, optical center c, and radial distortion factor k) are used to convert the homogeneous coordinates $(u, v, 1)$ to screen coordinates (x, y) according to:

$$\begin{pmatrix} x \\ y \end{pmatrix} = f \begin{pmatrix} u \\ v \end{pmatrix} (1 + k(u^2 + v^2)) \tag{2.13}$$

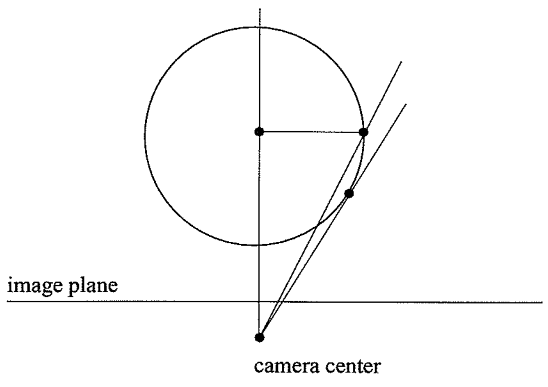


Figure 2.6: The error incurred by the projection approximation. When the cone is very close to the image plane, its projected radius is underestimated.

In principle, due to the radial distortion, straight lines will become curved; however, once again we approximate the result by computing the image coordinates of the two endpoints of each side of a cone's boundary, and assuming the boundary appears as a straight line in the image. This approximation is reasonable, since the radial distortion is not too strong (distorting by a few pixels near the corners of the image, but less than 1 pixel distortion elsewhere).

Besides the two side boundaries of the two truncated cones for the upper and lower arm, a boundary for the hand tip is also computed as being perpendicular to the axis of the forearm. The resulting arm projection is shown in figure 2.7.

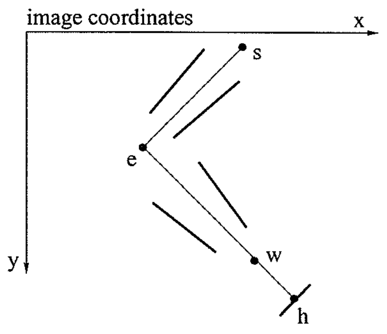


Figure 2.7: The set of solid black lines is the resulting predicted arm boundary in image coordinates.

2.2.3 Measurements

Given the projected contour of the arm onto the image, measurements can be made to compare the expected contour to the actual observed image. As mentioned previously in the section on the goals and design principles of our system, we do not try to extract explicit features from the image (such as an explicit detection of where the arm boundary is in the real image). Rather, we use the more robust and computationally efficient method of implicit feature detection (implicit boundary detection in this case) via direct image comparison.

The measurements are obtained in a two step process. First, a foreground/background segmentation is computed based on the pixel color statistics of the image. This operation generates a new image where the person (foreground) is white (value 1) and the rest of the scene (background) is black (value 0). The second step uses the predicted appearance of the arm contour to choose locations on the segmented image at which to extract image intensity values. The two steps are now described in detail:

Foreground/Background segmentation

Given the assumption that the colors of the person's skin and clothing are different from the colors in the rest of the scene, it is possible to segment the person from the rest of the scene as follows:

- On a sequence of images of the background only (without the person in the scene), collect pixel-wise statistics on the mean and variance of the YCrCb color values of each pixel in the image. YCrCb is one of many encodings of the 3-dimensional color space where Y represents the intensity (brightness) and Cr and Cb encode the color information. This representation is advantageous for our use because it represents color and intensity independently; if two tones are the same hue but with differing brightness (as can happen when part of a person or object is in shadow), the Cr and Cb values will be the same, but the Y values will differ.

- For each image that needs to be segmented, use the acquired statistics to decide whether each individual pixel belongs to the background or not. If a pixel's Cr or Cb value is more than 3 standard deviations away from the background mean, the pixel is deemed to be foreground, and is assigned a value of 1. The rule for discrimination with Y values is slightly different. The foreground object may cast shadows on the background, thus changing the color appearance of those pixels significantly. However, we do not want to classify the shadow as part of the foreground. To a first approximation, shadowing does not affect the color of a region (Cr and Cb stay unchanged), but does reduce the intensity (Y) value. Thus to avoid incorrectly labeling shadowed areas as foreground, the test on Y is stricter: a pixel is labeled as foreground if the Y value is greater than the mean value by more than 3 standard deviations.

This scheme works quite well in practice. One improvement that could be made is to make the background statistics adaptive; by continually updating the statistics, slowly changing lighting conditions (due to the movement of the sun, or clouds) could be compensated for.

Extracting measurements from the segmented image

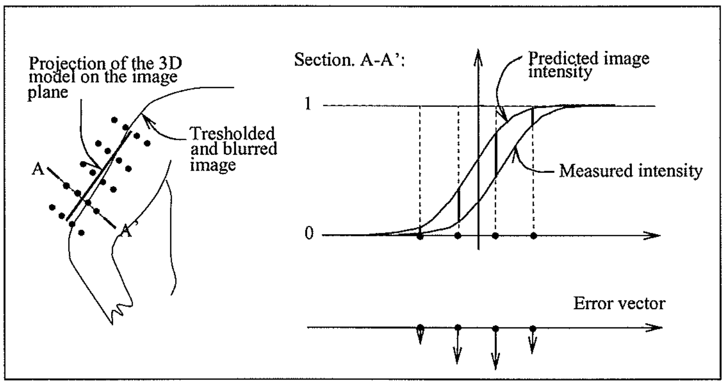

Projection of the 3D model on the image plane
Tresholded and blurred image
A
A'
Section. A-A':
Predicted image intensity
1
0
Measured intensity
Error vector

Figure 2.8: The resulting predicted arm boundary in image coordinates.

Given the projected contour of the arm onto the image, and the foreground/background segmented image, we can now compare the expected (segmented) image to the actual one. This is done by comparing the intensity of the segmented image with those of the expected image at various locations around the expected arm contour, as shown in figure 2.8. The segmented image is blurred with a Gaussian kernel of predetermined size, and so the intensity profile of a cross-section of the image perpendicular to the expected contour (line A-A' in fig. 2.8) is a smoothly increasing curve; 0 in the background away from the contour, reaching 0.5 right at the contour, and 1 inside the arm. Now if the actual arm contour does not lie exactly where predicted, the cross-section profile will be shifted left. Sampling the intensity of the segmented image along various perpendicular cross-sections to the expected contour (figure 2.9), we can build up a long vector of differences between the expected and actual (segmented) images.

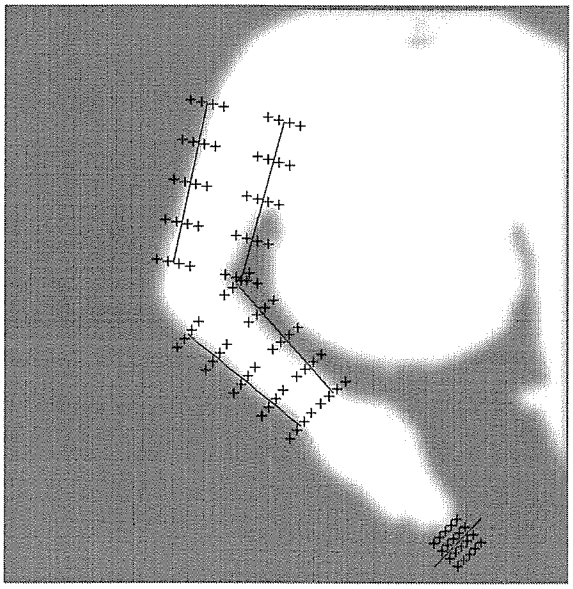


Figure 2.9: Example of sampling image intensities perpendicular to the predicted arm contour.

This error vector, $e(\Theta, I)$, is a function of the observed image, I, and the joint coordinates $\Theta = (\theta_1, \ldots, \theta_4)$, and can be computed with sub-pixel accuracy by doing a bi-linear interpolation on the four adjacent pixels to the sub-pixel location to be sampled. In our system we make measurements at 4 locations perpendicular to the boundary (spaced 5 pixels apart), and do this sampling at 5 locations along the boundary. Since there are 5 arm boundaries (including the hand tip), we make a total of 100 image comparison measurements. The vector $e(\Theta, I)$ is the feedback error vector used in the estimation loop, described next.

2.2.4 Estimation technique

Given the vector of implicit measurements (direct image comparisons), we use a Kalman filter to update the estimate of the arm pose. The state of the system is the four arm joint angles and their joint velocities. The state evolution equations assumes a constant joint velocity, with additive Gaussian noise $w \sim \mathcal{N}(0, Q)$, where the noise covariance Q is tuned through experimentation and depends on the image sampling frequency and the speed of movement:

$$\begin{aligned} \theta_i(t+1) &= \theta_i(t) + \dot{\theta}_i(t) & (2.14) \\ \dot{\theta}_i(t+1) &= \dot{\theta}_i(t) + w(t). & (2.15) \end{aligned}$$

Since the system measurements $e(\Theta, I)$ cannot be written as a function of the system state alone but rather depend implicitly on the state and image (and furthermore, for the true correct values of the state $e(\Theta, I)$ represents an implicit constraint on the system, since $e(\Theta, I) = 0$ in that case), we use an Implicit Kalman Filter, as described in appendix A.

Chapter 3 Arm tracking experiments

3.1 Off-line experimental results (Matlab experiment)

Our system was first developed and tested using Matlab, a high-level and interactive mathematical programming language. To test how well the system functioned, an experimental arm motion image sequence was recorded.

3.1.1 Experimental setup

The subject sat at a table, with the camera facing him, 1.2m away. A rectangular path was imprinted faintly on the tabletop. The subject is intended to trace out this path with his finger-tips, and the imprinted path serves as a ground-truth comparison for what the reconstructed arm trajectory should be.

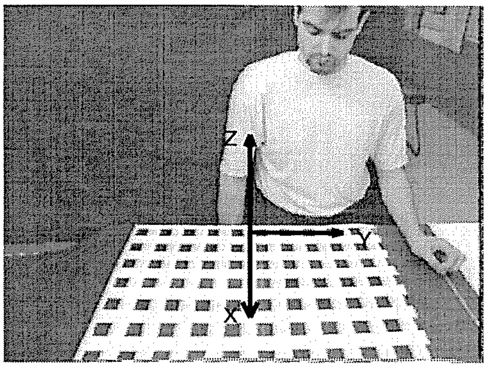


Figure 3.1: The calibration setup. A checkerboard pattern is used to determine the camera parameters and the transformation between the camera reference frame and the ambient reference frame.

The position of the camera with respect to the table was determined via a calibration procedure similar to that described in B.2.1. Figure 3.1 shows the calibration setup. Additionally, the position of the subject's right shoulder with respect to the tabletop reference frame is also measured.

To facilitate image processing, the subject was wearing a light colored T-shirt, and the tabletop and surroundings were dark colored.

The motion of the subject tracing out the path was recorded on video, the subject moving his hand at an approximately constant speed of 3 cm/s. Since the goal of the experiment was to verify the accuracy of reconstruction of 3-D arm pose, the subject tried to trace out the path as accurately as possible. The subject also took care to ensure that his shoulder did not move from the calibrated position, and that he did not rotate his wrist with respect to the forearm.

3.1.2 Results

Ground-truth motion on desktop

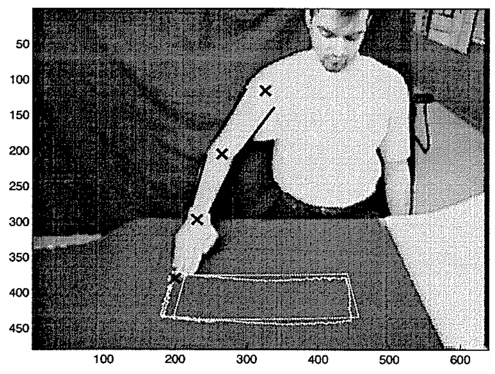


Figure 3.2: The tracked trajectory: The initial arm position is shown with the initial estimate. (X) mark estimated shoulder, elbow, wrist and fingertip positions, solid lines outline estimated arm. Ground truth trajectory as well as estimated trajectory are shown. Shift of estimated trajectory is due to the fingertip not lying on the forearm axis.

Figure 3.2 shows (from the point of view of the camera) the ground-truth trajectory and the resulting estimated hand-tip trajectory. One can see that there is a slight shift to the right of the estimated trajectory. This is due mainly to the fact that the arm model does not estimate degrees of freedom at the wrist, and the subject did not have his hand in exact alignment with his forearm. Thus the wrist (and hand-tip) are shifted to a location slightly to the left of where the model thinks the hand-tip should be based on the pose of the forearm arm. If the hand were tilted even more to the left, the tracker would not find any evidence at all for where it expected the hand to be (recall, the hand is detected by matching the expected fingertip positions to the actual positions). In this case, the tracker would eventually converge on a compromise solution where neither the arm nor the hand were perfectly fit.

Occlusions do not bias solution

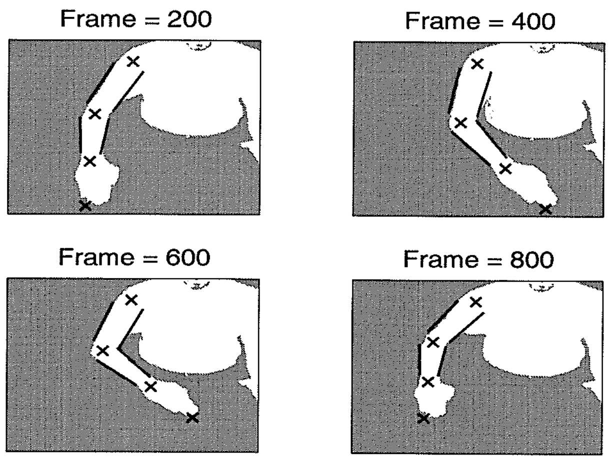


Figure 3.3: The tracked arm: The estimated arm position projected onto the thresholded image for 4 frames of the sequence.

Figure 3.3 shows the image plane projection of the estimated arm pose in four frames of the sequence. The model fits the silhouette of the arm quite well, mainly because the parameters defining the geometric shape of the model were measured on that particular subject. In these images we can see again that the subject had his hand tilted slightly to the left with respect to where the model estimated the hand-tip center to be.

An important thing to notice is that the system estimates the arm pose quite well despite missing information on the inner side of the upper arm (due to loose clothing, and indistinguishability from the torso). This success is due to two factors:

First, in the computation of the pose estimate by the Kalman filter, a multiplicative term corresponding to the Jacobian of the image measurement (error between expected and actual image) with respect to the arm pose parameters (joint angles) is present. At an arm boundary occlusion, such as at the inner upper arm in these images, the measurement error is large, but does not change with infinitesimal changes of arm pose (in a sense the measurement is saturated) so the corresponding Jacobian entry will be zero. Thus, measurements at an occlusion do not affect the pose estimation.

The second reason for robustness to occlusions is the choice of tracking scheme itself. Since the pose estimation is done via direct image matching with implicit measurements, the image is matched to the model by varying the pose parameters directly. If we had separated the tracking task into two separate subtasks of feature detection and pose estimation given the features, the system would not have worked as well. In that case, both the inner side of the upper arm, and the hand-tip location, would be difficult to detect and most likely detected erroneously, resulting in a degraded pose estimate. In contrast, the direct image matching method can be thought to 'holistically' and implicitly find the best edges consistent with the arm model, and the fact that the overall model fits well counteracts the fact that the model may not be fitting perfectly at some boundaries.

In summary, the naive method is prone to errors due to occlusions and incomplete information, whereas the direct image matching technique is quite robust with respect to these problems.

Resulting accuracy

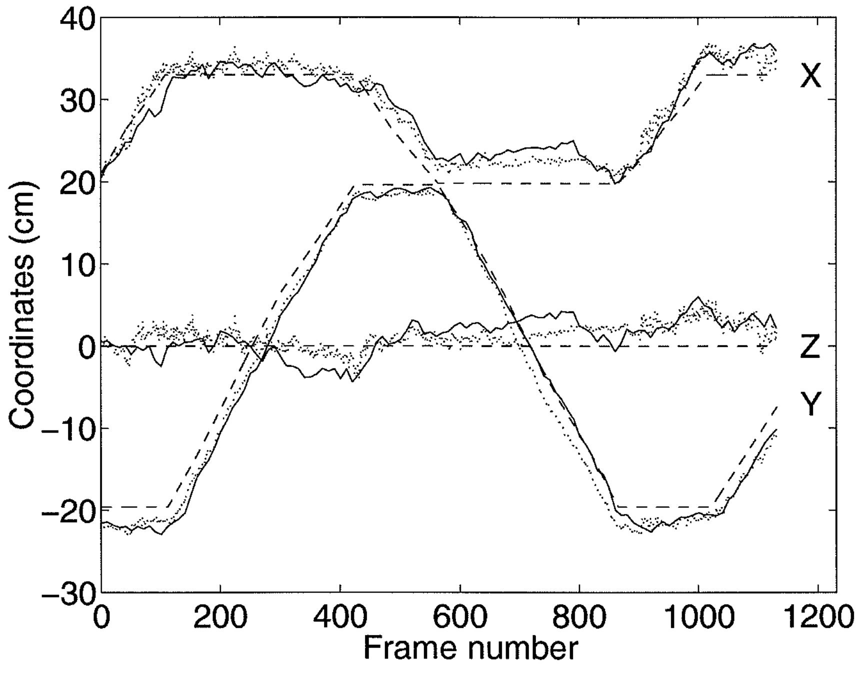


Figure 3.4: Position versus time: The tracking error when every frame was tracked (dotted) and when every 10th frame was tracked (solid) as compared to ground truth (dashed). The errors of the two tracks are comparable and under 5 cm in all coordinates.

Figure 3.4 shows the estimated 3-D hand-tip coordinates (in the tabletop reference frame) throughout the frame sequence. Besides the ground-truth trajectory (dashed), two traces are shown. One (solid) is the result of tracking using only every 10th frame of the sequence (tracking at 3 Hz), the other (dotted) is the result when all frames are used (tracking at

30 Hz). The overall offset of the two traces from the ground-truth is similar (due to the inaccurate hand-tip pose). In both cases, the error with respect to the ground-truth can be as high as 5 cm. The fact that the 3Hz tracking worked almost as well as the 30Hz tracking is due mainly to the speed with which the subject traced the trajectory. Since the movement was quite slow, at a virtually constant speed of 3cm/s, the dynamical model of the Kalman filter was able to make accurate predictions of the arm motion even at a 3Hz frame rate. Had the subject moved his arm quicker, with more sudden accelerations and decelerations, the tracking at 3Hz would not have been as successful.

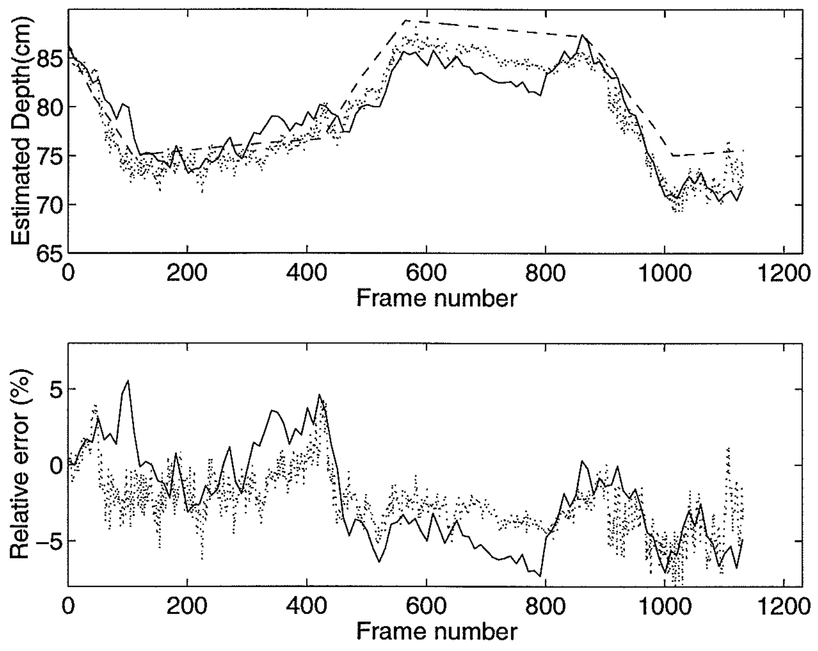


Figure 3.5: Depth error versus time: Top plot shows tracking error in the direction of the camera optical axis. Dotted - every frame tracked, Solid - every 10th frame tracked, Dashed - ground truth. Maximum error is less than 5 cm. Bottom: relative depth error is less than 7.5%.

Figure 3.5 shows the estimated depth of the hand-tip (distance along the line-of-sight to camera) throughout the sequence. The line-of-sight coordinate is the position of the hand-tip in the direction perpendicular to the image plane, and therefore the most difficult coordinate to estimate from a monocular view. Comparing the estimated depth to that of the ground truth gives us a good sense of how accurately our monocular tracking system can estimate 3-D.

3.2 Real-time implementation

Given the success of the off-line experiment, we then embarked on building a real-time implementation of the system. The reasons for doing so were three-fold:

1. Real-time experimentation to study robustness. With a real-time, interactive system, it is much easier to understand when and how the system fails, as well as how to adjust the tracking parameters for best performance. Off-line experimentation requires recording the motion on video, recording to a laser video disk, then processing each frame by digitizing from video disk under computer control. This process would take several hours to set up, and then a few hours (to process the sequence) to see the effect of a change to the algorithm.

2. Study performance and ease of use as a 3-D input device. Since the eventual purpose of the system is to act as a real-time sensing device, it makes sense to test it as such. Real-time experimentation alone can give a good sense for how easy to use and how well the system works as a 3-D input device, say.

3. A test-bed system with which to test new algorithms.

3.2.1 System

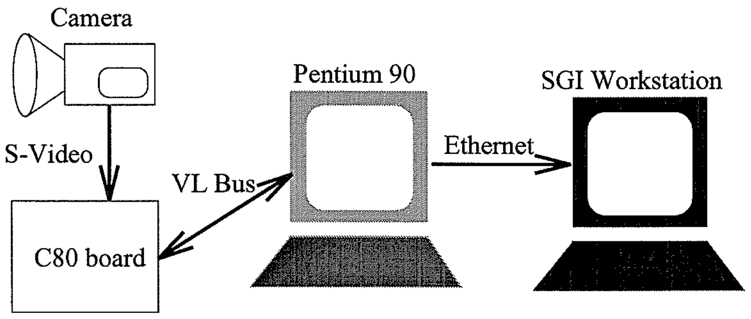


Figure 3.6: The hardware of the real-time system. The DSP board performs background subtraction and implicit measurements, the host PC runs the tracker proper, and the application computer runs a 3-D test-bed application using the camera-based tracker as input device.

At the time of the design of the system, the state-of-the-art PC was a Pentium 133Mhz. This did not provide enough computational to do all the computations in real-time (30Hz), so an additional DSP board was used. Figure 3.2.1 shows the setup:

- The DSP board computes the background subtracted image, as well as the results of direct image comparisons at locations specified by the host PC. The DSP itself is a 50Mhz Texas Instrument C-80 based board. The C-80 processor has 5 CPUs, 4 fixed-point, and one floating-point. If programmed properly, the combined power is approximately 2 billion operations per second. (Let us mention in passing that programming the C-80 was not a trivial matter.)

- The host PC runs the main tracking loop, including the Implicit Extended Kalman Filter. It sends measurement requests to the DSP, and sends the resulting arm pose estimates through the ethernet to the application test-bed PC.

- The application test-bed computer runs a simple 3-D application, where the user has to pick up a virtual cube and move it to a new location. Figure 3.7 shows a typical display. The cube to be picked up is red, and the desired destination is in blue. The virtual arm of the user is also displayed, and the display uses perspective projection, so that a cube being manipulated becomes smaller and larger as it is moved further or closer to the camera. To aid the user's manipulation in 3-D, X, Y, and Z coordinate scales are also shown, with the current and desired coordinates marked on the scales.

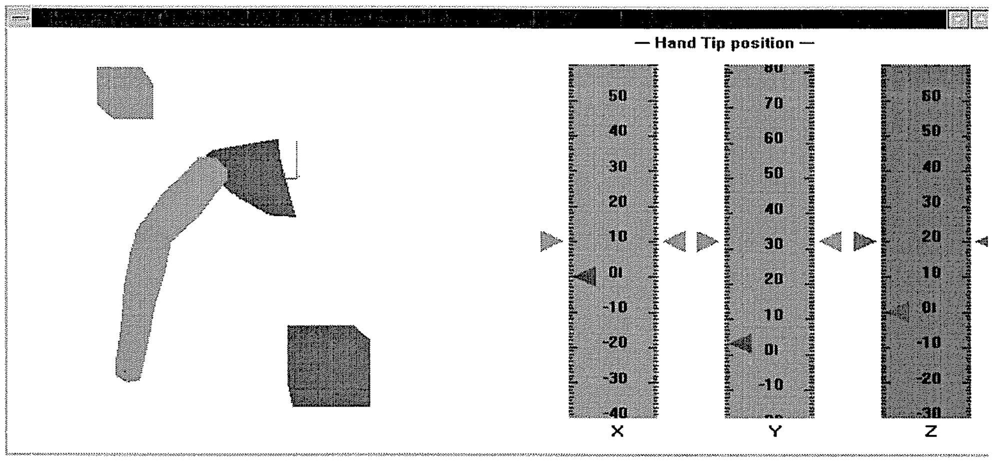


Figure 3.7: An experimental task: The user has to grab hold of a virtual box and relocate it to a target position. The display shows a perspective view of the arm, box, and target location. To further aid the user, target and initial box locations are indicated on the XYZ scales as well.

3.2.2 Performance

With the real-time system in place and functioning, we tested it with respect to: ease of use; accuracy and resolution; and robustness to occlusions, fast movement, and loose-clothing.

Ease of use

Experiments with several (6 or 7) users revealed that with 5 to 10 minutes of practice, a new user could successfully and reliably manipulate the virtual cubes. The most difficult part of the task was for the user to sit such that his shoulder was in the right position. The system, recall, expects the shoulder of the user to be in a specific 3-D position, and so a new user has to figure out where to sit so that the shoulder is in the right place. Another difficulty in using the system was for the users to keep their hand extended along the axis of the forearm. It is very easy for people to forget and start rotating their wrist.

The most noticeable effect of using the system, however, was that after 5 minutes of cube manipulation the subject's arm became very tired. This is due to the fact that during cube manipulation, the subject's arm is moving unsupported through space, and this tires certain muscles of the upper arm (policemen guiding traffic at intersections suffer from the same problem). Thus the ergonomic design of the interface is not optimal. In a real application, this issue needs to be dealt with carefully, since it greatly impacts the usability of the system.

Accuracy and resolution

From the results of the off-line experiment, we know that the accuracy with which the position of the hand-tip is estimated in 3-D is approximately 5 cm. This is the absolute positioning accuracy of the system. Of more relevance (for a human-machine interface) is the repeatability and resolution of the system. In a human-machine interface, the human can compensate for inaccuracies and distortions in the mapping from real 3-D to virtual 3-D (this is not as true for augmented reality scenarios), as long as the output is repeatable (moving to the same point in real 3-D should give you the same point in virtual 3-D), and has a high enough positioning resolution. Figure 3.8 shows the results of an experiment to measure the repeatability of the system. The user moved his hand repeatedly between two points on a string hung obliquely in front of him. At both points, the standard deviation of the X,Y, and Z coordinates is approximately 1 cm.

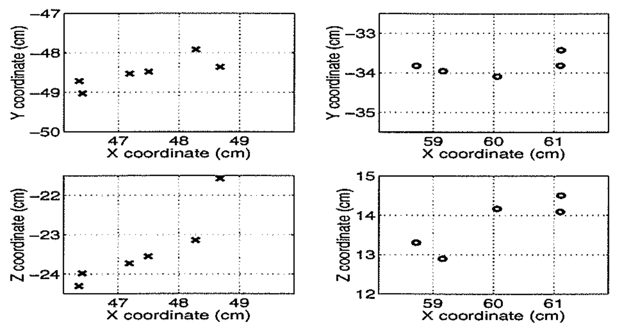


Figure 3.8: Repeatability of hand-tip position estimation: Scatter plots of estimated hand-tip position when the user repeatedly moves to the same locations in the workspace. Left: a location on the tabletop, Right: a location along a tightly strung string between the tabletop and the ceiling. The coordinates are relative to the shoulder position; X is forwards (towards camera), Z is upwards, and Y to the left. The camera was 130 cm from the shoulder, looking downwards with an inclination of 30 degrees to the horizon.

To estimate the resolution of the system, we used the virtual cube manipulation game, and varied the accuracy with which cubes had to be picked up and placed. We found that with a placement accuracy of 2 cm (the cube would be picked up or placed if the arm was within 2 cm of the exact location), users were able to reliably control the cubes. With 1 cm placement accuracy, the task became quite difficult. Thus the effective resolution of the system is approximately 2 cm.

Robustness to occlusions

The system is quite robust to occlusions due to the use of implicit measurements and the efficient Kalman filtering tracking scheme. Even crossing one arm over the other does not disrupt the system.

Robustness to fast movement

The Kalman Filter used in the system includes estimates of the arm velocity, and 'expects' changes in velocity up to a certain magnitude (modeled as the Gaussian uncertainty in the update of the velocity estimates). Thus the system can track the arm even when it is moving and changing direction quite quickly. For example, the arm can be tracked when waving left-and-right over a distance of 30 cm at a frequency of approximately 1.5Hz. (This translates to a maximum velocity of $1.2m/s$ and a maximum acceleration of $7m/s^2$.)

Robustness to loose clothing

Loose clothing can make the system less robust and reduce the accuracy of the pose estimates. Loose clothing deforms the appearance of the arm, and thus makes it more difficult for it to fit to the model. The most sensitive area with respect to this is the area of the fore-arm near the wrist, since the change in appearance due to loose clothing is most noticeable there. A loose shirt with sleeve rolled half-way up the fore arm is quite bearable.

3.2.3 Failure modes

The typical mode of failure for the tracking is due to two effects: 1) the subject does not sit with his shoulder in the exact correct spot, so that the arm model does not fit the arm perfectly for all possible orientations of the arm. 2) Due to changes in illumination or background scenery, the background subtraction processing is not completely clean, so that some regions of the background are misclassified as foreground.

With these two conditions present (and the condition worsened by a bad fit of the model to the arm due to, say, loose clothing), it can happen that for some poses of the arm the match is so bad that it is lost completely.

Chapter 4 Conclusions and future work on human observation

In this part of the thesis we have presented a method for tracking the motion of a human body in 3-D using an articulated model of the body. The method was tested by implementing a real-time arm tracker. This system is able to track the movement of the arm robustly, with an accuracy of 2 cm (measured on the finger-tip position). Moreover, it can do so using a monocular viewpoint, and when there is great variation in the appearance of the arm throughout time (due to perspective effects of the person being close to the camera).

The success of the method is due mainly to the use of implicit features to extract information from the sequence of images. Rather than detecting low-level features and making hard decisions on scene content at that stage, the image is compared to an 'expected image generated based on the current estimate of the body pose. The differences between the actual and expected images at relevant image locations is used to update the pose estimate.

It is interesting to speculate on the use of top-down model-fitting strategies for perception by humans. Some optical illusions seem to be based on this principle, such as the apparent violation of size constancy of a person given a deceptive monocular view of the person moving about in a non-rectangular room. Below are two anecdotal personal experiences where top-down models seem to be in place and are temporarily fooled.

Glasses in Santa Barbara

On a weekend trip to Santa Barbara with some friends, I decided to remove the lenses from my semi-rimless glasses to see how long it would take for them to notice the missing lenses. To my amazement, they never did notice. Their internal models of my face (and glasses) was so strong, that they filled in the missing information, believing that my usual glasses were there. I did, of course, notice strange looks from people walking by on the street, and from the waiters with whom I had to interact. The illusion was finally broken when, by coincidence, we met another common friend from Caltech. Since this person was not used to seeing me everyday such as my other friends, her model of me was not as strong and reinforced. Also, unlike the total strangers that I encountered that day, she was not afraid to point out the seemingly absurd observation. She noticed there was something wrong with my face right away. My other friends were quite surprised that they hadn't noticed it themselves.

Man with hat walking away/towards me

Another mismatch of models to data occurred to me one day as I was walking on campus. Straight ahead of me, perhaps 50 meters away, a man was walking directly away from me along the same straight line path (or so it seemed to me). I found his motion rather unnatural, but yet could not figure out exactly what was wrong. This eerie perception lasted for about 10 seconds, by which time the man was close enough that I was able to extract more input to fit to my model. I had assumed that the man was walking away from me because I could not see his face. When he moved closer, I realized that his hat kept his face in complete shadow, so that at such a distance it was difficult to distinguish whether I was looking at the front or the back of his head. As soon as I saw his face, I could fit the model to the data much better, and realized that the man was actually walking towards me, not away from me.

4.1 Future directions of research

Among the (now) many human body tracking systems being developed around the world, ours stands out as being the only one able to estimate 3-D at close range (where perspective projection must be taken into account) using only a single camera. There are many ways in which the system can be expanded and improved. The path we chose to explore, improving the dynamical model used by the Kalman Filter, quickly led to a completely new research topic, motion synthesis, discussed in the second half of this thesis. Here, we list some of the other potential topics for work in body tracking.

Model more of the body

Since tracking the arm worked so well, it is reasonable to attempt to model and track more of the body. Tracking the torso should improve the robustness of the arm tracker itself, because it removes the restriction that the shoulder has to be in a pre-calibrated position. Tracking the torso should work quite well, because a good model can be defined: the sides of the body and the top of the shoulders 'bound' the torso. Ultimately, the entire body could be modeled and tracked. It seems reasonable to expect that this should work since the degrees of freedom of the legs are the same as those of the arm. Also, if the system incorporates knowledge of human locomotion, the further constraint that (stationary) feet are in contact with the floor can greatly aid the estimation process.

Automatic body model generation

Another assumption of the current system that can be improved is to no longer assume that the 3-D model of the body is known apriori, but rather to estimate it for each user. This estimation could be done as a precursor to tracking, our it could be done adaptively while the tracker is in use. In this latter case, the system would start off with an initial guess of the body model (given the users height), and slowly adapt the model parameters given the tracking observations.

Multi-resolution measurements

One potential way to improve the robustness of the system could be to perform implicit measurements at multiple scales. Currently, the real image is compared to the expected image only within 10 pixels of the predicted arm boundaries. If the arm moves too fast from one frame to the next, it could be that it will lie beyond the regions where the measurements are being made (this is especially true of the forearm and hand). However, if the image comparisons were done at a larger scale (say 30 pixels), some useful measurements would still be made. In fact, perhaps a clever solution would be to make the implicit measurements adaptive, based on the estimated velocities (and 2-D projection of the velocities) of the various arm boundaries. This would cause the measurements of the forearm to be done on a larger scale than those of the upper arm, which seems reasonable and useful.

Velocity measurements

Although our system estimates joint velocities, it does not use velocity information in the images directly. However, implicit measurements of the optical flow could also be used to estimate joint velocities. To implement this successfully, the key point is to NOT attempt to estimate the optical flow directly, but to incorporate it into the arm pose estimation process in the form of an implicit measurement, such as h(I(t), I(t+1), X, dX/dt) = 0. Here, I(t) and I(t+1) are the images at time t and t+1, and X and dX/dt are the 3-D positions and velocity estimates for the degrees of freedom of the arm/body. As is well known, to compute optical flow directly is a very difficult and ill-conditioned process because only incomplete information can be obtained locally at most locations in the image (the aperture problem; motion parallel to a boundary is invisible). However, given a parameterized model of the flow, it is very easy to predict what the flow should be. In our case, given the current estimate of the pose and velocity of the arm, one can predict how the image of the arm at time t should transform into the image at time t+1. Usually, the use of such a constraint requires several iterations for the convergence of the motion parameters. With the use of a Kalman Filter, however, it is hopefully the case that just one update per image is sufficient.

Improved dynamical model

Another way to improve the performance of the system could be to improve the dynamical model used by the Kalman Filter. Currently, the model is simply a random walk in joint velocity, with the noise model tuned for best performance. This type of model contains no knowledge specific to arm motion. In fact, a typical motion generated by this model does not resemble typical arm movement at all. In contrast, there are some models of arm motion in the biomechanics (and robotics) literature that mimic particular types of movements quite well. For instance, it is well known that the velocity profile of the hand-tip during ballistic arm movements is bell shaped. With this model, given a desired speed and destination, a convincing arm motion can be computed. Unfortunately, this model (and others) which does so well at synthesizing a motion given the desired characteristics, is difficult to convert into a predictive model. For example, it only describes certain types of movements, and in a tracking situation, one would have to first detect when such a movement was occurring to use the model.

Nevertheless, due to the hope that a better adapted dynamical model might improve the performance of the system dramatically, we embarked on the development of general models of human motion that could be used in a tracking framework. We soon realized that these models could serve another purpose: the automatic synthesis of realistic human motion, and this is the subject of the second half of this thesis.

Part II

Synthesis

Chapter 5 Introduction to motion synthesis

5.1 Goals

The automatic synthesis of realistic human character motion is a difficult and challenging problem. In recent years, it is increasingly common to see virtual, human-like characters on the big screen in movie theaters, and on television. The characters are so life-like and convincing that one may be led into believing that the problem of motion synthesis has been satisfactorily solved. This is far from the truth. The virtual characters seen in the mass media are the result of thousands of hours of work by talented animators. One minute's worth of animation can take weeks to produce. If it is desired to have a virtual character whose motion is synthesized automatically in real-time (such as for a character on a web-page, or in a computer game), current techniques produce animation with much lower levels of realism and life-likeness.

In this thesis we attempt to move one step closer to the ultimate goal of realistic and automatic motion synthesis. This ultimate technique can be envisioned to have the following properties:

- controllability at various levels of detail, from manipulation of individual body parts to a very high level, script-like set of commands to act out.
- control over a variety of moods and styles
- embedded in a physically-simulated universe where the character interacts through physical laws with its environment
- has autonomous behaviors and reflexes
- runs in real-time (synthesis can be done on-line at interactive rates)
- the resulting motion is life-like, indistinguishable from that of a real person.

To that end, we have extended our initial work [15] and have developed a system where

- The behavior of the character is defined by a sequence of actions.

- Actions are defined with specific parameters (where to look, step, throw).
- Actions are modifiable with moods and styles.
- Motion synthesis is efficient and can be executed in real-time
- The motion approaches being life-like, because mathematical models of human motion are created through a data-driven process of observing the motion of real people.

5.2 Previous work

The automatic generation of realistic human motion has been a topic of research within the computer graphics and robotics communities for many years. Here we briefly review the existing techniques:

Dynamic Simulation

One method of attempting to synthesize realistic human motion is to use a 3-D model of a body with rigid segments with associated masses and moments of inertia, and to simulate the dynamics of this body via the application of torques to the various joints. Hodgins et al. [19] use this approach to animate human athletics. By creating hand-crafted controllers which generate foot placements, hip rotation, and phasing of arm and leg swing, they can successfully simulate a running human (as well as spring board vaulting, and diving). Their method is successful* because athletic motions are dynamically constrained: there are not many ways to move to achieve a particular dive maneuver, say. However, for ordinary non-athletic motions such as walking around or picking up a light object, the motion can be performed in many ways (each person has a particular, identifiable walking style, and a good actor can imitate various styles). In this case, it is the particular synchrony the brain's muscle control which defines the motion, and the dynamics of motion is virtually irrelevant. Thus for everyday actions dynamic simulation methods will fail to generate realistic motion unless the brain's motor control signals can be properly modeled and imitated.

*Their method is partially successful; since the controllers are all hand-crafted, the resulting motions seem somewhat awkward and robotic

Space-time Constraints

Another dynamics-based technique is that of 'Space-time constraints' developed by Witkin et al. [36]. The idea is to specify initial and final (and intermediary) constraints as to where the character is and how he is to move (for example initial and final positions and velocities and a distance to jump). Then a constrained optimization is solved for the joint torques that will produce the desired motion. The method was successfully applied to an anthropomorphized lamp with three movable links to generate convincing stomping, jumping, and smooth landing on a ski jump. However, it seems unlikely that the same method could be applied to models with many more degrees of freedom, such as a human body. In that case it seems difficult to come up with sufficient constraints to make the motion solution well-defined and realistic. This is more so the case with non-athletic motions, as argued in the previous subsection.

Inverse Kinematics

Another approach to motion synthesis is to use the robotics-based techniques of inverse kinematics [28]. When applied to a robot, these techniques allow the computation of the robot's configuration (set of joint angles) that will place the end-effector at a certain position and orientation in space (subject possibly to additional constraints imposed by the environment or other considerations). When combined with energy or torque minimizing principles, these methods can be used to produce efficient and well-behaved robot motions. Badler et al. [2] have used such solutions in the development of their Virtual Jack project. Virtual Jack has a complex virtual skeleton with over 100 joints, and was designed to simulate the posing (and to some extent the motion) of the human body in order to provide accurate ergonometric information for ergonimc studies. However, Jack is known to have a stiff back, stand in awkward poses, and move in a robotic fashion. The reason for this is that these robotic approaches have no notion of the naturalness of a pose, or of the intricate, characteristic phases between the motions of different body parts.

Manipulation of recorded motions

Many methods of generating realistic motion by manipulating motion capture data (3-D recordings of people's movements) have and continue to be developed. Bruderlin [9] uses the technique of Motion Signal Processing, whereby multi-resolution filtering of the motion signals (changing the gain in different frequency bands) can produce different styles of motion. For examples, increasing the high frequency components produces a more nervous-looking motion.

Gleicher [14] introduced the method of motion editing with space-time constraints. Here, a motion is adapted to satisfy some new constraints, such as jumping higher, or further. The new motion is solved for by minimizing the required joint displacements over the entire motion, and this way attempting to preserve the characteristics of the original motion. It can be thought of as a generalization of the inverse-kinematics techniques, where instead of computing the pose to satisfy the constraint only during a single frame, the modification of pose is done through time. However, since there is no notion of how realistic (or human-like) a modification is, the method can be used only to generate small changes - otherwise, the laws of physics appear to be defied.

Cohen et al. [33] describe a method very similar to ours. They represent motions in terms of 'verbs' and 'adverbs', similar to our 'actions' and 'moods and styles parameters'. However, our system also includes the concepts of 'state parameters' and 'goal parameters', enabling a more detailed control over the motion. Another difference is that they encode motion using B-spline coefficients defined at certain key-frames to represent relative joint angles along the character skeleton, whereas we encode motion with the 3D Euclidean coordinates of markers placed on the body and use a real-time method to map the motion to an articulated skeleton. Because errors in relative joint angle along the kinematic chain accumulate, their synthesis can result in awkward-looking motions.

Procedural Animation

Finally, there is the technique of procedural animation developed by Perlin [29]. In this technique, life-like motion is generated by layering several simple hard-coded motions defined for different body parts. The different motions are activated according to a pre-defined probability distribution function, and their amplitude and duration are also stochastically generated. For example, modeling the rise and fall of the chest with breathing, the blinking of the eyes, occasional swings and turns of the head, as well as occasional changes in standing posture, can create convincing animation. However, the amount of control of the character (and interaction with the environment) is limited; the method does not lend itself easily to controlling the walk of a character, where the entire body is involved in a complicated fashion.

Chapter 6 Action-based synthesis

Our method of generating realistic human motion is based on the idea that rather than deriving mathematical principles which define control laws (a robotics approach), models of realistic motion can be learned by observation of people performing various actions, and then deriving data-driven models of human motion from these observations.

6.1 Technique overview

We need a generative model; from a high-level description of the desired motion, we want to create a motion that is as realistic as possible. The human body is capable of an infinite variety of motions (especially since any particular type of motion/action can be executed with an infinite variety of moods and styles). Although one could envision a monolithic, universal model capable of generating all types of human motions, we simplify the problem by attempting to learn compartmentalized models for specific actions. In this thesis we describe techniques that are general enough to be applicable to any particular action chosen for modeling.

With this approach, the problem of realistic human motion synthesis becomes tractable, but we do run the risk of limiting the set of motions that are learnable to those motions that can be categorized as a nameable action. In speech recognition and synthesis, there is a small, finite set of elemental sounds, phonemes, from which all words can be generated. To our knowledge, there is no equivalent set of elemental movements, supposedly to be called 'movemes', from which all movements can be generated. Thus, we choose as 'elemental actions' such nameable actions as walking, running, looking, and throwing. Each of these actions can be parameterized by such things as speed of movement, the direction of the step or throw, as well as the mood and style with which the motion is performed. Thus, a synthetic character can be animated by specifying the actions (with corresponding parameters) to be executed through time.

There is no guarantee that this system will model all movements done by real people; some movements, such as hand gestures, may be difficult to name and to parameterize, and if so do not easily fit within the framework. For those actions that do fit the framework, the method consists of the following main steps:

- Acquisition of data with a motion capture system
- Segmentation of individual sample motions from the data
- Labeling of each sample with an appropriate vector that describes the sample action
- Generating a mathematical model which computes the resulting motion given the motion label
- Utilizing the mathematical models of several actions to synthesize new realistic motion sequences.

In the following sections of this chapter, the techniques developed for each of the main steps (except for the generation of mathematical models) are discussed in detail. The description of the mathematical models used for motion synthesis is dealt with in the next chapter, where the use of established techniques such as polynomial interpolators, and neural networks, as well as two novel techniques (local linear models and a recursive probabilistic formulation) are discussed.

6.2 Building datasets

6.2.1 How to acquire data

The first step in the modeling process is to acquire human motion data. Since the methods to be developed will be data driven, we need to be able to acquire large amounts of human motion data. There are commercially available systems (with mechanical, magnetic, or optical sensors) to acquire human motion data (so called 'motion capture', or 'mocap' systems), but they are very expensive, costing from \$20,000 to \$250,000. There are also motion capture services, where one can spend the day, pay \$3,000, and acquire 30 to 60 minutes of data. However, we would undoubtedly need to use such a service several times, so that it would also be too costly. We thus decided to build our own motion capture system for a total hardware cost of approximately \$3,000 (PCs donated by Intel). Appendix B describes the function and design of the mocap system in detail. Here we briefly outline the main features of the system.

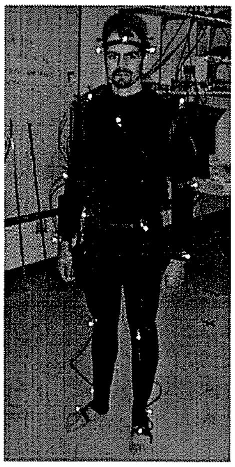


Figure 6.1: Mocap suit: In order to capture the movement of the human body in 3-D,18 light-bulb markers are placed on the major joints of the body.

Our motion capture system works by placing 18 battery-powered light-bulbs on the major joints of the body, as in figure 6.1. The actor moves around in semi-darkness, and the motion of the light-bulbs is observed by 4 cameras. Because the relative positions of the cameras are known (through a calibration procedure), the 3-D position of each light-bulb can be triangulated from the multiple viewpoints. Within the capture volume of 3m by 5m (by 2m vertically), 3-D positions are reconstructed with a standard deviation of error of 1.5mm (i.e., the projective rays from each viewpoint come within 1.5mm of the computed 3-D position). Data is captured at 60 frames per second, and a typical capture sequence last 2 minutes.

6.2.2 How to represent motion

The output of a motion capture session is a large matrix defining the X, Y, Z coordinates of the 18 markers (light-bulbs) on the body throughout time. When the data is rendered on a computer screen as a moving dot display, it generates a very compelling perception of human motion. This remarkable ability of the human visual system to infer bio-motion from a moving dot display was first observed by Johansson [22]. In subsequent experiments he found that bio-motion can be distinguished from non-bio-motion (motions that have no interpretation as the motion of a human, such as if the dots were displaced randomly before the onset of the motion) in only a few hundreds of milliseconds of presentation, and that with stimuli presentations lasting a few seconds, many details of the person can be inferred, such as the gender and mood. Thus the motion of the dots themselves are a good medium to evaluate the perceptual quality of the models learned.

An alternative representation is to map the motion capture data to a skeleton; compute all the joint rotations of the skeleton so that it follows the motion of the markers. In this form, the motion can then be visualized by rendering a 3-D character. We chose not to represent motion this way because of the following three reasons:

- conversion from mocap to joint rotations is cumbersome and prone to inaccuracy if the skeleton setup is not correct
- since joint rotations are relative, errors in the models would propagate throughout the limbs resulting in unnatural motions. This would be particularly evident at the feet, which would be seen to be sliding on the floor.
- joint coordinates are a nonlinear space (rotation angles) whereas marker space is Euclidean, so that it will be easier to build accurate models in Euclidean space.

6.2.3 Design of motion capture sessions

When capturing sample motions, several considerations must be kept in mind. Since we want to build a mathematical model that can later be used to generate new, synthetic motions, we will need some sort of control parameters to be able to specify the exact motion desired. For instance, in a reaching motion, we will at least need to specify the location that needs to be reached. Thus, in order to learn the motion model properly, it is necessary to provide sample motions over the entire range of control (goal) parameters.

Another important aspect of data acquisition is that the sample motions must all be done in a consistent fashion. For instance, the same style or mood must be maintained throughout the samples of a particular action. If this is not done, there will be some variability in the samples which is not modeled properly since it is not parameterized, and the motion learning procedure will not function properly. As a specific example, suppose we are capturing examples of a looking action, where the actor starts off facing forwards, and then looks in a particular direction. The actor should strive to perform the looking motions with a consistent speed, because, say, very fast motion affects the dynamics of the entire body. Alternatively, one could choose to include speed as a motion parameter, and then one would need to ensure that enough samples at each speed are collected.

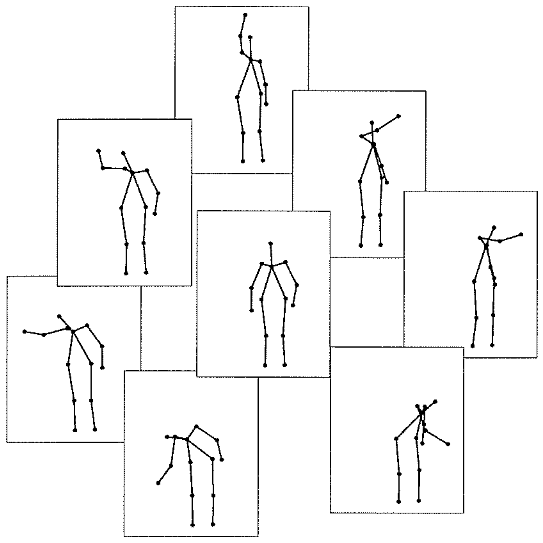


Figure 6.2: Some of the reaching poses. Starting from the rest position (center), the subject reached with his right hand to various locations around him.

6.2.4 Pre-processing of data sets

Once motion capture sessions have been executed, one final pre-processing step must be done; individual sample motions must be segmented out from the bulk data. This is typically done in a semi-automatic fashion; some method of detecting the onset and end of a motion sample is developed, and the segmentation process is applied to the dataset, subject to manual verification and adaptation. For example, in a looking motion, one can detect the start of the sample by detecting the onset of a non-zero velocity of the head marker (assuming the actor starts each sample by standing still).

6.3 Data sets

Below is a description of all the datasets acquired to test and develop the motion synthesis techniques. They are referred to in subsequent sections in order to illustrate and analyze the techniques.

2D reach

Figure 6.2 shows some snapshots of a reaching motion. This was one of the first motions acquired, before the 3-D motion capture system was available. Thus the data is from one camera only, and is in 2-D. In this motion, the actor stood facing the camera, with arms by

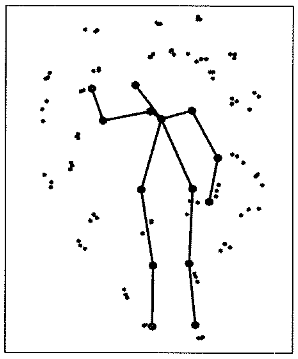


Figure 6.3: The 91 reach locations.

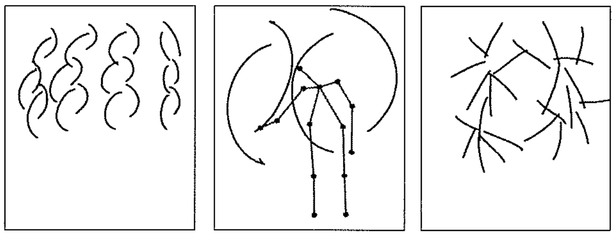


Figure 6.4: Some of the sample drawing strokes. Both straight and curved strokes at various scales were drawn, for a total of 378 samples.

his sides, and then reached to a random location around him, as if picking an apple from a tree (or from the ground). In order to make the dataset consistent, the actor always stood in the same initial position at the onset of each sample, and returned to that pose at the end of each sample. 91 samples of reach motions were collected, the end locations shown in figure 6.3. The actor reached around in all directions, near the limit of his reach range, and half way. The duration of each reach motion varied from 90 to 117 frames (almost 2 seconds), and they were all uniformly resampled to be a length of 117 frames long.

2D draw

Figure 6.4 shows some examples of 2-D drawing data. To perform this action, the actor stood and faced the camera. Starting at a random location, the actor performed simple strokes, consisting of either straight lines or curved arcs (but never curves with inflection of curvature). In all, 378 samples of drawing motions were collected, with the strokes varying in position in space, size, and amount of curvature.

3D walk

The first datasets captured with the 3-D mocap system were of walking. The subject walked back and forth along straight and curved paths, starting and stopping. With a consistent style of walking, the aim was to capture samples of walking with as much variability as possible in terms of length of step, and curvature of path. Due to a limitation of the mocap system (restricted space and too few cameras), the actor had to always face the same general direction (+/- 30 degrees) and was not free to walk around naturally, which would have been ideal. Instead, short sequences of walking along a straight or curvy path for 5 steps, followed by walking back to the starting position, were repeated over and over. After pre-processing, 124 samples of stepping with the left foot, and 119 samples with the right foot were obtained.

3D happy walk

In this dataset, a few examples of walking in a different style - 'happy' were acquired. The aim was to use this dataset to learn an incremental model of a new style of walking based on the original walking model (as will be discussed in section 6.6 on learning moods and styles). Thus, only 16 samples of happy walking (16 steps with left and right foot each) were acquired.

3D look

In the look dataset, starting from the same initial position, looking straight ahead, the actor turned his head, neck and torso (entire body, in fact) to look in various directions; looking straight up, down to the floor, to the left, to the right. The visual hemi-field was approximately sampled every 20 degrees (vertically and horizontally) to generate the samples. In total, 34 samples were acquired.

3D step over

In this dataset, many examples of stepping over an object were recorded. The variability of the samples included the size of the object (roughly three heights, 15cm, 30cm, and 50 cm were used). Because an actual physical object would interfere with the motion capture process, an imaginary object of variable height and length was used. Also, the angle of attack, and curvature of the walk path during the stepping over was varied. In all, 29 steps with the left foot and 34 with the right foot were captured.

6.4 Desired properties of motion models

Given a dataset of sample motions, various methods of machine learning can be developed and applied to create motion models. Before describing such methods, it is worth discussing the desirable properties that one requires of them.

Accuracy of representation

Clearly one very important property of a learning method is the accuracy with which it can synthesize motions. There are various aspects to consider when assessing the accuracy of a model.

First, one has to realize that there could be a significant difference between numerical accuracy and perceptual accuracy. For example, if a resynthesized motion has some phase delays with respect to an original sample, a numerical, frame-by-frame sum-of-squares error could be large, whereas perceptually the two motions may be indistinguishable. Conversely, the numerical errors may be seemingly small, and yet produce perceptually very noticeable errors, such as foot sliding. Thus, evaluation of the accuracy of a model should include both types of error measurements. In fact, given the nature and goal of the application, the perceptual error is more important (although the numerical error is much easier to compute, since evaluation of perceptual error is a subjective task involving tests with several subjects).

Second, for data driven models, one has to distinguish between errors from samples in the training set, and errors from samples in a test set. It is important to have separate test data that is not used to learn the model, because that way one can assess the generalization ability of the model. Separate test data will allow the distinction between the case where the training error is small due to overfitting of the model (say, because there are too many parameters in the model), and a genuinely good model.

Closely related to the issue of generalization is the issue of interpolation versus extrapolation. Depending on the values of the parameters specified for a newly synthesized motion, the motion model may be performing an interpolation (the input parameters lie within the convex hull of the inputs of all the samples used to learn the function), or an extrapolation.

The behaviour of the model in the two regions of operation can be quite different (as will be demonstrated in forthcoming sections). In principle, the samples for learning should cover the entire range of input parameters, so that one would never have to extrapolate, but in practice one may end up trying to extrapolate for several reasons. First, one may simply want to try to reach a bit further, or walk faster, or with a larger stride. Second, for actions with high dimensional input parameters, it may not be clear where all the boundaries of interpolation are.

Number of samples required

For practical reasons, all other things being equal, a method that requires few samples is to be preferred over one which requires many more samples. In the limit, one would like to be able to learn a new motion from a single sample. Certainly some methods can be thought of that require so many samples that they are completely impractical. One way to compare different methods with respect to the amount of data required is to measure how many samples are necessary in order to achieve a certain level of accuracy.

Size of model, efficiency of synthesis

Two other important factors to consider when developing motion models is the size of the model and the amount of computation required to synthesize a motion. In our case, since we are interested in developing methods for real-time interactive animation, it is important that motion be synthesized efficiently.

Level of control and flexibility

Finally, a choice has to be made as to the level of control and flexibility that the motion model provides. Here, we chose to control the character at a high level; move-by-move as opposed to frame-by-frame. By specifying the action with some goal parameters, along with some desired mood or style, the character can be made to follow some sort of 'script' - but it will not be possible to control the character down to the level of individual frames, as in the classical animation key-framing technique.

6.5 Labeling each motion

The first step in creating a model of an action is to associate with each sample motion a set of control input parameters, called a 'label', so that the model can represent the relationship between the label and the resulting motion. Later, a new motion can be synthesized by applying the appropriate label to the model.

6.5.1 Different types of parameters

A label consists of three different types of parameters: state parameters, goal parameters, and mood and style parameters.

State parameters

State parameters provide information about the current status of the character; information which is necessary to compute subsequent motion. The state can include information about the current velocity, position, and orientation of the body, for instance. It is also necessary to decide upon a reference frame. For the walking action, it is convenient to distinguish between steps with the left foot, and steps with the right foot, and to make the stationary foot the reference frame (align the x axis to be along the foot, y pointing leftwards, and z upwards). Continuing with this example, to define a (left-footed) step, it is necessary to know (at least) where the left foot was at the beginning of the step (w.r.t. the right foot origin), and where it will end up. The starting foot position becomes the state parameter.

Goal parameters

Goal parameters determine the specific desired outcome of the action. In the left step example, the goal parameter is the desired position of the foot at the end of the step. Often, there could be alternative definitions of the state and goal parameters. Instead of defining foot positions, one could parameterize a step by specifying the initial body centroid position, velocity, and the desired final position and velocity. This particular alternative will be discussed in detail in section 6.5.2 where the description of the labels for all the datesets is provided. In general, the choice of parameterization depends on the actual control parameters desired (whether one wants to define foot locations or step direction and velocity), but is subject to the constraint that it must provide a sufficient representation of the state.

Style and mood parameters

Style and mood parameters are used to specify a particular mood, such as happy, sad, or tired, or a particular style, such as walking with a limp, or like John Wayne, or Marilyn Monroe. Immediately the question arises as to how to parameterize these inherently non-numeric qualities. The proposed solution is to learn styles and moods as secondary models; given a 'nominal' walk motion, one modifies it to become 'happy' by adding some fraction of the residual happy motion (this process to be elaborated in section 6.6 on learning moods and styles). Thus each mood or style has an associated parameter which can be varied between 0 (mood or style not expressed) to 1 (mood or style fully expressed).

6.5.2 Examples of labelings

2D reach

In the reach dataset, since the actor always started from the same initial condition, no state parameters are needed. The goal parameters are the 2-D screen coordinates of the position reached. Figure 6.3 (in section 6.3) shows the locations of the 91 sample reaches. No examples of reaching with different moods or styles were acquired.

2D draw

The goal of the 2D drawing dataset is to draw a simple stroke. Thus the goal parameters need to describe the path of the drawing hand. To represent the path in a compact form, it is represented by the coefficients of a cubic interpolant:

$$x(t) = \sum_i a_i t^i \tag{6.1}$$

$$y(t) = \sum_i b_i t^i \tag{6.2}$$

where time has been rescaled to be $t = 0$ at the beginning of motion and $t = 1$ at the end. Finding the coefficients to solve the above equations is a minimization problem, and does not guarantee a perfect fit. It is important, at least to represent the starting and ending position accurately (i.e., equations 6.1 must be hard constraints for $t = 0$ and $t = 1$, and soft constraints for $0 < t_i < \ldots < t_n < 1$). Let

$$P(t) = [\ 1 \quad t \quad t^2 \quad t^3\ ] \tag{6.3}$$

and let C^* be a solution of

$$\begin{bmatrix} P(0) \\ P(1) \end{bmatrix} C^* = \begin{bmatrix} x(0) \\ x(1) \end{bmatrix} \tag{6.4}$$

Also let C^{n_1}, C^{n_2} be a basis to the null space of $[P(0)P(1)]^T C^* = 0$. Then a solution satisfying the hard constraints and minimizing the error in the soft constraints can be found by solving for w_1 and w_2 which solve the least squares problem:

$$X - PC^* = [PC^{n_1}PC^{n_2}] \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \tag{6.5}$$

where $X = [x(t_1); \ldots ; x(t_n)]$ and $P = [P(t_1); \ldots ; P(t_n)]$

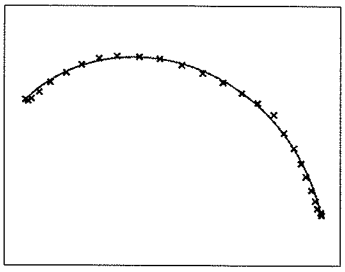


Figure 6.5: A sampled stroke, and the spline interpolation defined by the 8-dimensional stroke label.

Figure 6.5.2 shows a sample fit of a stroke (it is difficult to visualize the resulting coefficients themselves since they lie in an 8-dimensional space).

3D walk

The 3D walking dataset is more complicated than the 2D reach or draw datasets, in the sense that it requires a parameterization of the state of the character at the beginning of the step. Also, it is not as obvious how to parameterize a step.

One possible parameterization is to label each step with an initial position (and/or orientation) and velocity (this will be the state parameters), and define the goal in terms of the desired displacement and final velocity. Since we are in fact not interested in the 'external pose' (the orientation and position of the body in space: a step is still a step, if it's done here or on the moon), it is convenient to normalize all the steps to a standard reference frame. We pick the reference frame centered on the centroid of the two hip markers, with the x-axis pointing towards the front of the character. (During synthesis, the appropriate rotation and translation will have to be applied to the character to position it in space). In this standard position, the initial state can be defined by the position of the feet on the floor, and the initial velocity of the hip centroid. The goal is defined by the displacement of the hip centroid from start to end of the step, as well as the final hip centroid velocity.

Unfortunately, whether a certain parameterization works well or not can only be determined through experimentation. After some experimentation with learned models, it was determined that this parameterization of a step did not function well. The reason for this was that the parameterization did not provide explicit control over the final position of the feet. After a few iterations of generating steps to walk along a straight path, the character would converge to a fixed point solution of the foot placement, and invariably ended up waddling like a duck, no matter what displacement goal was used.

To avoid this unwanted fixed point dynamics of the foot steps, it is necessary to control the placement of the feet explicitly. To do so, the reference frame of each step is taken as the (initial) position of the stationary (support) foot, with the x-axis along the foot direction. The state of the character is defined by the initial position of the moving foot, and the goal is defined by the final position. Although this representation encodes the movement at a lower level of representation (before only the desired direction of motion had to be supplied, whereas now the actual foot placement must be given), it can be used to produce synthetic steps that work well. Figure 6.6 shows the labels for all the examples of left and right footed steps.

3D happy walk

In this dataset, the labeling of the samples is done exactly as in the 3D walking case. The level of happiness is assumed to be constant (the actor needs to perform the motions consistently). During synthesis (after the appropriate models have been learned), the happy

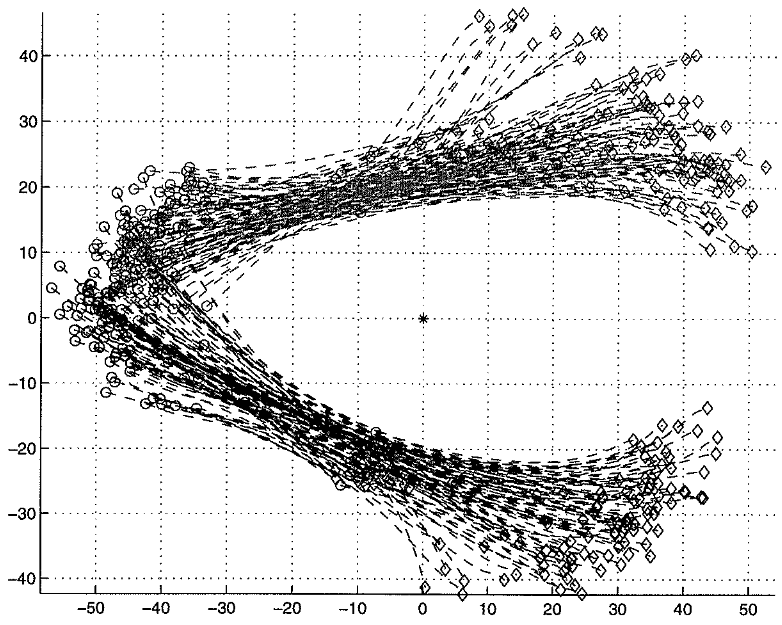


Figure 6.6: Plot of foot start and stop positions and trajectories for walking data. The reference frame is that of the pivot foot, with ankle marker on origin (marked by *) and foot marker aligned along x-axis. Circles denote the start position of the stepping foot, and diamonds denote the end position. Trajectories on top are those with the left foot stepping (right foot pivot). Units in cm.

mood can be combined with nominal walking. Figure 6.7 shows the labels for the dataset.

3D step over

For the 'stepping over' action, not only does the initial and final position of the stepping foot have to be specified, but also the height of the object (actually the height to raise the foot to) is part of the goal parameters. This value was extracted from the examples as the maximum height that the moving foot achieved. See figure 6.8 for the labels of all the step-over examples. Note that for stepping over an object, there are four possible motions involved; the lead step with the left or the right foot, and the follow-through step with the other foot. The dynamics of the movement of the lead foot is quite different from that of the follow-through foot, and so those motions should be learned separately. Due to a limitation in the mo-cap system (the light-bulb suit limited the range of motion of the knees), it was not possible to capture examples of the follow-through motion.

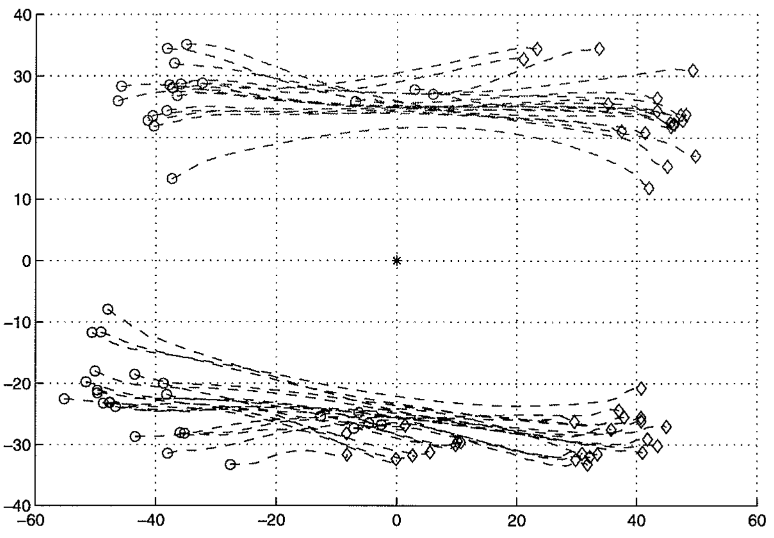


Figure 6.7: Plot of foot start and stop positions (in cm) and trajectories for walking data acquired with a new mood: happy walking.

3D look

In the 3D look dataset, the actor always started from the same initial position, thus it is not necessary to encode any state parameters. The goal parameter is the gaze direction (azimuth and elevation), computed as the azimuth and elevation of the vector from the centroid of the two ear markers to the forehead marker. Figure 6.9 shows the labels for the samples.

6.6 Learning moods and styles

Before describing the methods for learning actions per se, we describe the general principle behind learning a mood or style.

Given enough examples, one can always learn a new style or mood of, say, walking, by learning an entirely new and separate action. However, this can be rather inefficient and impractical, since a large number of samples would need to be provided for every new mood or style. Instead, it is possible to learn a new mood or style as a modification of the original action. In this fashion, a new mood or style can be learned with significantly fewer examples.

Suppose $F : \mathbf{L} \mapsto \mathbf{M}$ is a function which maps a given label $\mathbf{l} \in \mathbf{L}$ to a motion $\mathbf{m} \in \mathbf{M}$.

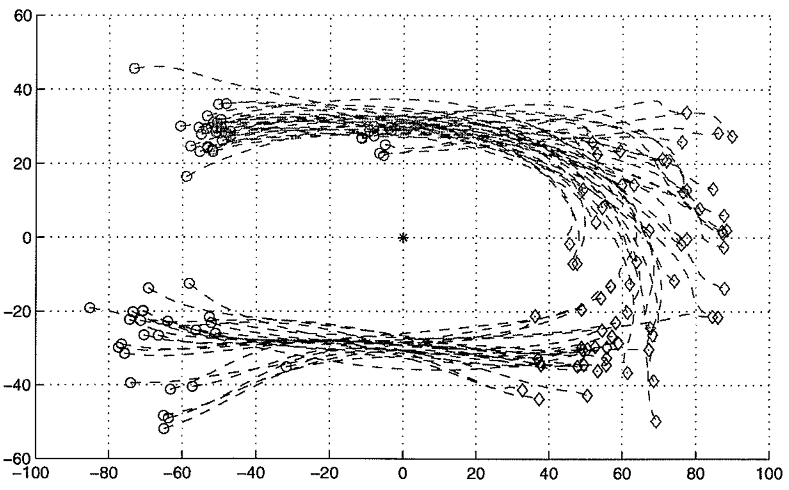


Figure 6.8: Plot of foot start and stop positions (in cm) and trajectories for the stepping-over action. Note that these steps are much longer than during normal or happy walking.

Suppose also that samples $\{\mathbf{l_i}, \mathbf{m_i}\}$ of motions performed with a new style are avaliable. Then we can calculate the residual motions due to the new style as $\{\mathbf{r_i} \equiv \mathbf{m_i} - F(\mathbf{l_i})\}$.

Using the residual motion and the sample labels, $\{\mathbf{l_i}, \mathbf{r_i}\}$, a residual function $R : \mathbf{L} \mapsto \mathbf{M}$ can be learned. Then, to synthesize a new motion with the new mood or style, the original function F and the residual function R can be combined. In fact, a modulating parameter, α, can be used to control how much of the new mood to use:

$$\mathbf{m_{newmood}} = F(\mathbf{l}) + \alpha R(\mathbf{l}) \tag{6.6}$$

This is similar to the recent paper by Blanz and Vetter for face morphing [5] where linear combinations of sample faces are combined to produce new faces, except here entire motions are morphed.

Experimentally, it was verified that this technique does in fact work; with fewer examples, it is possible to learn a new mood. Intuitively, one can argue that a mood or style typically can be viewed as a small perturbation of the overall action. While the nominal function needs to be learned with many examples to ensure that a proper motion is generated throughout the label's domain, the new mood's residual is a small signal that is not as complicated (does not change as much as a function of the label). Thus, fewer examples are needed to encode it. Furthermore, perceptually, given that the overall motion is correct,

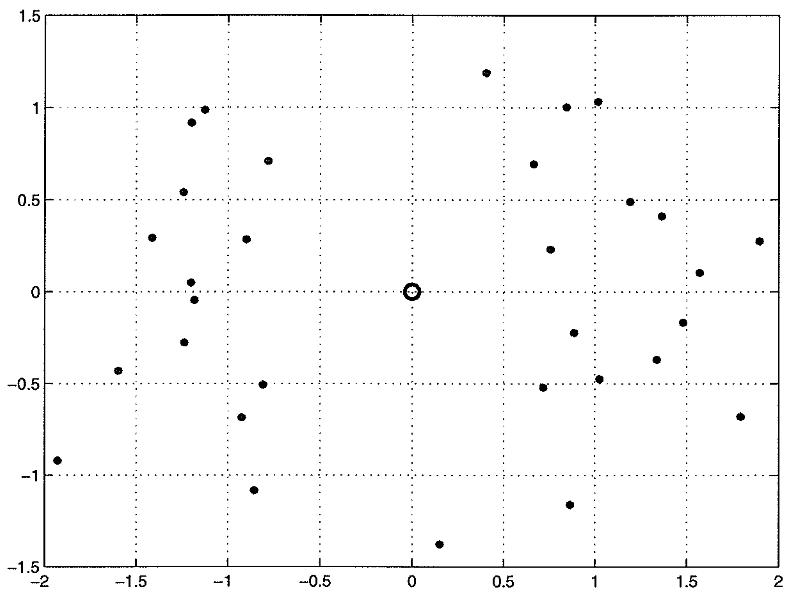


Figure 6.9: The labels for the 34 acquired samples of Look action. X-axis is azimuth of head, Y-axis is elevation (angles in radians), where origin (open circle) is when subject is looking straight ahead.

'errors' in mood or style are difficult to perceive.

In the next section the methods for synthesizing new motions given a learnt motion model are described. The methods for learning the motion models per se are deferred to the next chapter.

6.7 Synthesizing new motions

Once models of several actions are learned, they can be used to animate a character interactively and in real-time. The character will execute a series of moves under the control of the user. To do so successfully, special consideration has to be given to the transition from one move to the next. Since each movement is computed separately, it is necessary to ensure that when movements are strung together, the resulting sequence retains the same level of realism as that of the individual movements. Thus, it is important to ensure that the final pose of one movement matches the initial pose of the next movement. This match is achieved in two ways:

- for actions that are intended to be concatenated, special care in the design of the labels and the training set of the actions guarantees that it is possible to control the body pose at the start and end of the actions so that the body poses match.

- Slight discrepancies that may arise between contiguous final and initial poses due to the inexact nature of the action models are blended out to ensure pose continuity through time.

6.7.1 Concatenation of actions

State transition diagram guarantees proper transition from one motion to next.

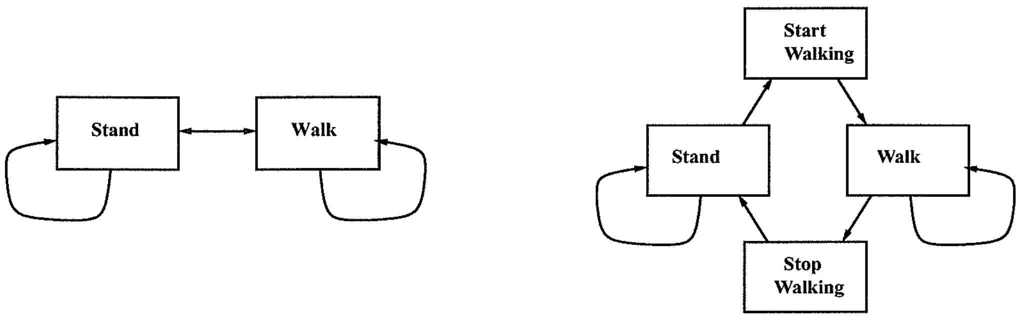


Figure 6.10: (a): a simple state transition diagram with two states. Transition from one to the other will be abrupt. (b): an improved state diagram, where extra transitional states ensure continuous motion during transitions.

Game developers have made use of state transition diagrams for many years in order to design the fixed animation cycles used in games [18]. Figure 6.10(a) shows a simple state diagram where the character can only be in one of two states. The 'walk' motion cycle is a looping animation in which the character takes two steps (so that the first frame matches with the last). The 'stand' motion cycle is also a looping animation where the character (for example) stands there, tapping his foot, waiting. With this simple diagram, each individual motion is seamless; the character can repeatedly walk (or stand) and the motion is seamless. However, transitioning between standing and walking is abrupt, the character suddenly 'pops' from one activity to the next. This problem is fixed by introducing two new transition states, as shown in Figure 6.10(b) designed specifically to smoothly animate the character from the last frame of standing to the first frame of walking (and vice versa). Now, all motion discontinuities (pops) have been eliminated; each arrow in the diagram (including self-referential connections) connects animations that concatenate smoothly.

Figure 6.11 shows a similar state transition diagram for our system, where the character

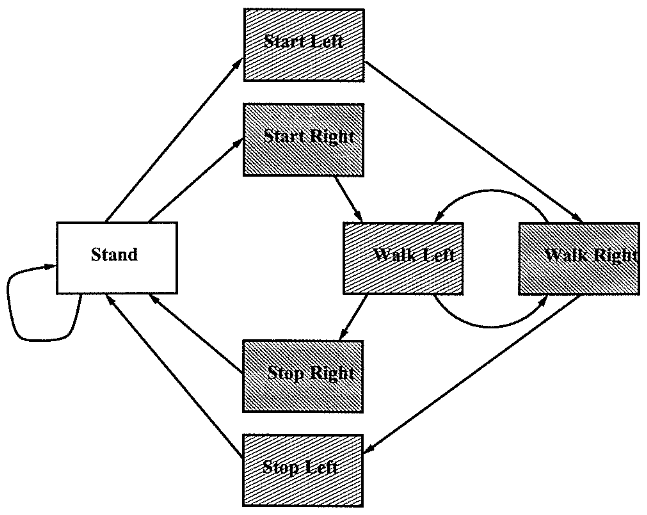


Figure 6.11: A state transition diagram allowing transitions from walking to standing (looking around). Since walking is generated step-by-step, left and right steps are separate states. Starting, stopping, and normal steps with the same leg are part of the same model, indicated by the shading of the rectangles.

can either be walking, or standing (stand and look around). Since walking is computed one step at a time, it is represented as two separate actions, taking a step with the left foot, and taking a step with the right foot. Transition states between standing and walking are also necessary, and the grouping of the states denoted by the shading in the figure denotes that 'start L', 'stop L', and 'walk L' are all the same action (taking a step with the left foot). In designing and learning the individual actions, care must be taken to ensure that the actions and their label parameters are flexible enough to be able to generate proper transitional motions.

For this example, the Walk (L and R) action label parameters include the initial and final feet positions, and that is enough to be able generate continuous walking motion along any desired path. The Look action is controlled solely by the direction of gaze, and the feet remain in a fixed position, so to transition from standing to walking (or vice-versa), one uses the standing feet position as the initial (final) feet position labels for the walking action. However, for a realistic first (final) walking step to be computed, it is essential that the set of training examples for the Walk action include examples of starting and stopping to walk.

Blending of initial and final poses smoothes out slight discrepancies.

Even with the proper design of label parameters and action training samples, body poses of two actions may not align exactly. For example, in the transition from walking to standing, although at the end of the Walk action the character will be standing, there is no way to guarantee that he will be standing in exactly the same pose (aside from the directly controlled feet position) as in the first frame of the Look action. The position of the hands, elbows, and other joints may differ by a few centimeters. This discontinuity is readily perceived in the animation.

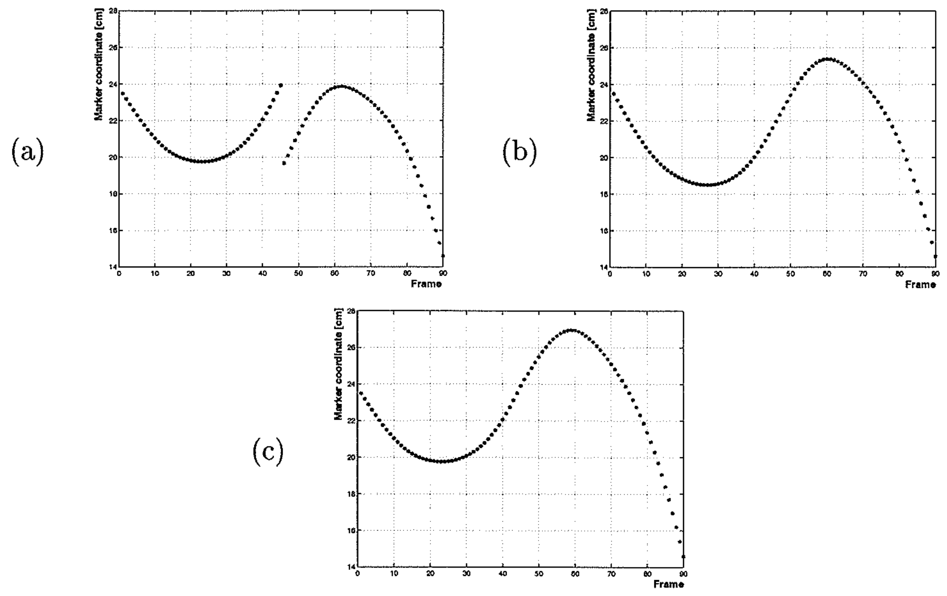


Figure 6.12: Blending to eliminate discontinuities. (a): the X coordinate of a head marker for two consecutive steps. Since each movement is computed separately, there is a slight discontinuity between the two. (b): the result of spreading out the discontinuity amongst the two movements. (c): the result of spreading out the discontinuity only in the second movement.

The discontinuity can be removed by blending it out over all the frames of the two motions. Figure 6.12 shows an example. In part (a), the coordinates of the elbow computed for two motions is shown. Part (b) shows the result of blending the discontinuity throughout all the frames of the two motions. Part (c) shows the result when blending takes place only in the second motion. While blending should be done over as large a number of frames as possible (to minimize the magnitude of the incremental modification of each frame), in a real-time application it may not be possible to blend over the first frame because it has already been animated, and so only the newly computed motion can be blended (referred to as causal blending).

6.7.2 Timing and generation of motions

Below, we outline the scheme for real-time motion generation. The basic principle is that animation to be rendered is held in an animation buffer, which is continually updated with animation for new actions in real-time.

Accept character input control from the user.

Input is accepted until the very last moment, i.e., until the animation buffer is nearly empty and a new motion must be computed and made available for rendering. User input includes the selection of the next action, and specifying values for some of the label parameters. The interface between the user and the character may ease the task by inferring label parameter values from higher level user control. For instance, in one implementation of the walking action, the user controls the walking direction with the mouse, and the interface computes the actual required foot positions required to walk in the direction specified by the user.

Compute the action with moods and styles.

Once input from the user has been acquired, the next motion is computed according to the following steps:

- transform user input and current pose to local ref frame of the action,
- generate state, goal and style parameters from current pose of character and user input,
- synthesize motion using action and style models,
- transform resulting motion to global ref frame,
- blend causally with previous motion and put in animation buffer.

6.7.3 Mapping motion to 3-D model

Given the motion in the animation buffer, the 'mapping problem' consists of mapping the marker motion onto a 3-D articulated character to be rendered on a computer display (in real-time). We solve this problem by creating an appropriately proportioned articulated polygonal character model, defining the 3-D position of the body markers with respect to the polygonal model, and using a Kalman filter to make the polygonal model follow the motion of the marker animation data in real-time.

Creating a skeleton model

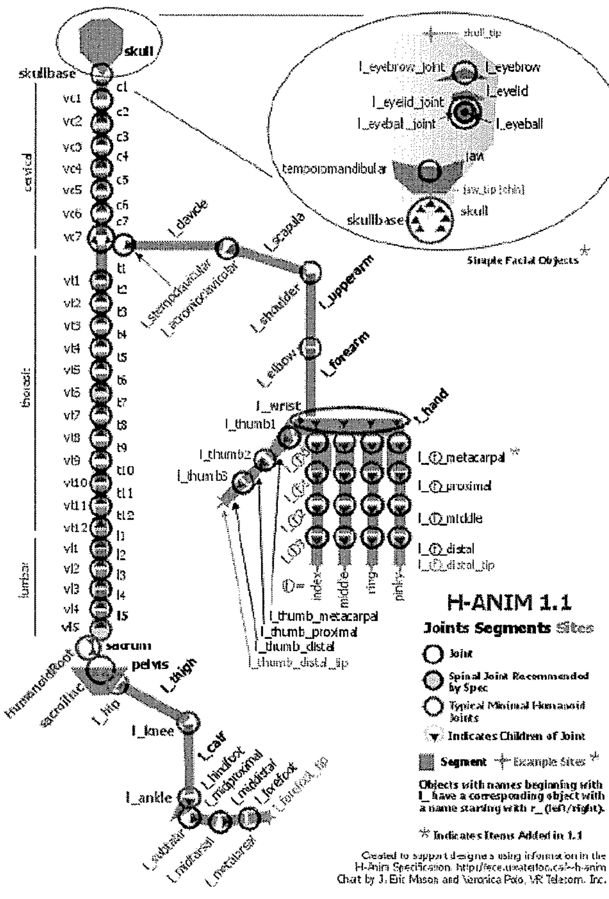

skull
skullbase
H-ANIM 1.1
Joints Segments Sites
Joint
Spinal Joint Recommended by Spec
Typical Minimal Humanoid Joints
Indicates Children of Joint
Segment
Example Sites
Objects with names beginning with l_ have a corresponding object with a name starting with r_ (left/right).
Indicates Items Added in 1.1
Simple Facial Objects

Figure 6.13: The H-Anim standard human skeleton model.

To create a 3-D virtual character, we used Hiro, an H-Anim compliant hierarchical polygonal human model. H-Anim is a standard being developed for representing virtual humans (for VRML and virtual worlds, and for MPEG compression standards). The standard specifies how to define a hierarchical representation of the human body. The body is represented as a series of nodes corresponding to joints in the body connected by rigid segments (figure 6.13). Several levels of detail of representation are possible with the standard. For instance, the most detailed model includes 25 joints along the spine, one for each vertebra, while the simplest model represents the degrees of freedom of the spine with just two nodes. For our application, since data from only 18 markers on the major joints of the body is available, the simplest model is most adequate (otherwise we would have too many extraneous degrees of freedom on the model). Note, however, that using such a model introduces some inaccuracies in the resulting motions, since not only the spine is simplified, but the shoulder complex is as well.

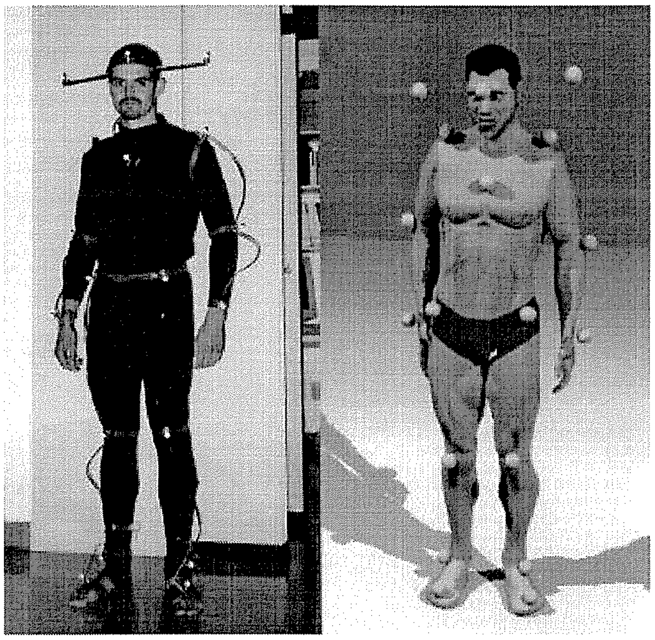


Figure 6.14: In order to map synthetic motions to a 3-D polygonal model, the model's body size has to be adjusted to that of the actor, and the corresponding placement of virtual markers has to be determined.

In order to map the data to the character as accurately as possible, it is necessary to scale the character model to the size of the actor (by adjusting the scale of all the 'bones' connecting the model's joints) and to estimate the corresponding position of the body markers with respect to the model (figure 6.14). During the motion capture sessions, each marker was placed on some part of the actor's body, (ideally) in a fixed position and orientation to that body part. For example, the elbow marker is placed on the forearm, on the outer side of the arm, approximately 4 cm away from the outer 'corner' of the elbow. Modeling the elbow joint as a planar joint with 1 degree of freedom, and the forearm and upper arm as rigid segments attached to the planar joint, we assume that the marker is attached rigidly to the forearm segment, so the motion of the marker is in one-to-one correspondence with the motion of the forearm. With this simplification of the human body, there is a corresponding position of the virtual marker with respect to the forearm segment of the character model. Thus, adaptation of the character model to the actor can be expressed mathematically as the estimation of scale parameters which scale the segments of the model, as well as the estimation of the (relative) 3-D position of virtual markers with respect to the rigid body segment to which they are attached.

In principle, the estimation of the scale and position parameters could be done automatically, given enough motion capture data. For example, the motion of the shoulder, elbow, and wrist markers could allow the estimation of the elbow joint center as a function of time, and likewise the motion of the upper arm with respect to the torso (as observed through the motion of shoulder and elbow markers with respect to torso markers) allows the estimation of the shoulder joint position as a function of time. Given the positions of the two joints, the size of the upper arm can be estimated (one of the scale parameters). If all segment sizes and joint centers are estimated this way, then the relative (fixed) position of the markers with respect to the corresponding segments can also be estimated from the motion data. In all, this model estimation problem is an interesting and difficult nonlinear problem, to which no solution has yet been found.

In lieu of the optimal, automatic model estimation, we proceed with a semi-manual procedure. Direct measurements on the actor are used to estimate body segment lengths, and the 3-D model is scaled correspondingly. Then, an initial estimate of the marker positions with respect to the 3-D model is made. Given the estimated skeleton scale and marker position guess, an iterative least squares procedure is run to optimize the marker placement. The iterations consist of two steps:

- Given the current estimate of marker placement, a motion capture sequence where all joints are moved significantly is mapped to the 3-D model (using the procedure outlined in the next section).

- Given the motion of the model, the positioning of the virtual markers on the 3-D model is optimized so as to reduce the RMS error between the virtual markers and those in the motion capture data.*

*This optimization is linear in the parameters being optimized, the positions of the markers with respect

Empirically, this iterative method is found to converge after approximately 10 iterations, and results in the motion capture data being mapped to the 3-D model with an accuracy of approximately 1cm RMS error per marker per frame between the true data marker positions and those connected to 3-D model.

Note that, in principle, for each motion capture session the parameter estimation technique needs to be redone; although the actor's skeleton size has not changed, the markers have been removed and re-applied between sessions, but not necessarily in the same exact positions. In practice, this is found not to be necessary; once the model parameters have been estimated, they provide satisfactory motion mapping with subsequent motion capture data as well. This is due mainly to the expert placement of markers on the body, using bone landmarks to position the markers with an accuracy of 1 to 2 cm. Also, positional variance of that order of magnitude does not affect the perceived quality of the motion mapping procedure.

Motion mapping with a real-time Kalman filter

To map motion capture data to the character model involves estimating at each instant in time the global rotation and translation of the root segment of the character, as well as all the joint rotations. These motion parameters are estimated in such a way as to minimize the 3-D error between the marker positions in the motion capture data and the corresponding virtual markers connected to the character model.

A convenient and efficient framework with which to perform this motion estimation is that of (Extended) Kalman Filtering (A). In this formalism, the state of the system is the set of motion parameters (global rotation and translation, and relative joint angles) along with the respective linear and angular velocities. The state evolution equation is the constant velocity predictor, with a Gaussian modeling error (with an experimentally derived magnitude, based on the expected range of acceleration in human motion) for each degree of freedom. The measurements are the observed 3-D marker positions of the motion capture data, modeled with an independent Gaussian measurement error of standard deviation 0.5 cm for each marker (magnitude of measurement error obtained from the characteristics of the motion capture system). The system output is the set of virtual marker positions

to the 3-D model. If we had also attempted to optimize over the 3D-model scale parameters, there would be non-linear coupling amongst the scale parameters and between the scale and marker position parameters - a much more difficult optimization problem to solve.

computed via the system measurement equation, which applies the translation and rotations of the system state to the character polygonal model in default position in order to compute the resulting virtual marker positions.

By computing an analytical expression for the Jacobian of virtual marker positions with respect to the system state, and by optimizing the C code implementation to use Intel MMX instructions, it is possible to implement the Kalman filter running at a rate of 4ms per iteration (time step). The resulting real-time, interactive demonstration described in the next section runs at 20 frames per second, most of the CPU time being spent on graphical rendering.

6.8 Real-time interactive demo

In order to test and demonstrate the motion synthesis method a real-time, interactive program was created where, under user control, a virtual character can:

- stand waiting, looking around randomly.
- transition naturally from standing to walking and vice versa.
- walk around, with variable step size, turning radius, and level of happiness.
- step over an obstacle, automatically adjusting the step to avoid hitting it.

The looking and walking motions were learned using the Global Linear Model (described in the next chapter), which proved to provide a satisfactory level of realism and computational efficiency during synthesis. The control of the looking action is done automatically; when the character is in the 'standing' state, the computer generates random elevation and azimuth values to control the character's gaze direction. The control of the walking action is more complicated. The user uses sliders to define the desired step size and level of happiness, and the mouse to define the turning radius of the walking step (as one uses a steering wheel, clicking on the center of the screen causes the character to walk straight ahead; the more on to the left of the screen one clicks, the tigher the leftward turn of the character). The user also has buttons to tell the character to stand or to walk.

A simple character state machine is used to convert the user's input into label parameters suitable to generate motions. The state transition diagram of figure 6.11 is used to transition from walking to standing and vice versa. Since foot positions are necessary to generate walking actions, the character state machine keeps track of the character's current position, heading, and turning radius, and computes the next foot placement in the following way:

- When the character is walking, the heading direction is updated with the current turning rate, and the new body position is computed by moving the desired step length in that direction. Then the new foot location is determined as being 5cm away from the new body center position (taking into account the orientation of the body).
- If the new foot position causes the foot to collide with an obstacle, the step is either shortened or lengthened. If it is lengthened, then it is lengthened enough to guarantee that the obstacle is cleared, and the stepping-over action is used instead of the normal walking action.
- When the character is stopping, the new foot position is placed 15cm away from the new body center position in order to open the stance to a natural standing position.

Happiness is computed as a residual (global linear) map of the walking action according to the procedure described in section 6.6 on learning moods and styles. Stepping-over is also modeled as a mood/style of walking, with the height of the step being the controlling parameter (variable between 15cm to 50cm).

Chapter 7 Motion models

In this chapter, we discuss the various methods that have been developed to learn actions given a set of labeled examples. After the description of all the methods, the various datasets will be used to illustrate and compare them.

7.1 Global polynomial interpolation

7.1.1 Global linear map

The simplest model one can use is a global linear model. If $\{x^s\}$ is the set of all label vectors (written as column vectors) for a particular dataset with N_s samples, we can form an augmented input matrix, X: $X = [1 \dots 1; x^1 \dots x^{N_S}]$. X is the matrix of sample inputs, augmented by a row of 1's. This row is used to learn the constant bias term in the model.

Likewise, if $\{y^s\}$ represents all the sample motions, where for sample s, the column vector y^s consists of the stacked up coordinates of all the markers throughout all the frames of the motion, then we can form the sample output matrix Y: $Y = [y^1 \dots y^{N_S}]$.

Now the best linear model can be found by solving for L in the least squares problem $Y = LX$, and this is easily done as $L = YX^t(XX^t)^{-1}$.

Surprisingly, this simple method often yields quite good results, as will be shown in the chapter with experimental results.

7.1.2 Higher order maps

Given that one has learned a global linear map, one may not be satisfied with the results because: a) the root mean square error of the model is too large, or b) the synthesized motions are not perceptually satisfactory. The question then arises if other models can do better. The answer to that question depends on the source of the errors in the linear model.

The error in the output generated by the model could be due to noise, in the sense that the actor who performed the sample motions is not a perfect machine, and there will be some uncorrelated variability in the sample motions. For instance, even though the actor is supposed to start each sample reaching motion with his arms by his side, there will be some randomness in the initial arm position which is uncorrelated to the goal position of the reaching action. This kind of error cannot be eliminated by any sort of model.

On the other hand, the errors could be due to the fact that there are non-linearities in the motion which cannot be captured by the linear model. In this case, there is hope that a more complicated model can reduce the error.

Unfortunately, for functions with such high-dimensional outputs (and multi-dimensional inputs) such as ours, it is difficult to visualize the goodness of fit of the linear model to determine whether there are non-linearities present or whether the error is due only to noise. However, one can still resort to using more complicated models and checking if the error (and perceived motion quality) improves. One must do so always keeping in mind that if the model is overly sophisticated, it may over-fit the data and then perform worse with new inputs (will not generalize properly).

The next obvious choice for a model, then, is to include higher order terms in the multi-dimensional polynomial interpolant. One can learn a global quadratic model by adding additional rows to the input matrix X corresponding to the pairwise products of individual label parameters. For a 2-dimensional label, three such products can be formed; with a 4-dimensional label there are 10.

The process can be formalized by defining N_b polynomial basis functions, with n^{th} function $\Phi_n(x)$ defined as

$$\Phi_n(x) = \prod_{i=1}^{N_D} x_i^{c_{i,n}} \tag{7.1}$$

where x is a sample motion label, x_i is the i^{th} component of the label vector, and $c_{i,n}$ is the power to which the i^{th} label component is raised in the n^{th} basis function. For the constant basis function, all c's would be zero; for linear basis functions, only one c is 1 and all others zero; and for quadratic basis functions the sum of the exponents has to equal 2 (either one is 2 and the others zero, or two of them are 1).

If we denote $\Phi(X)$ the matrix generated by applying all the basis functions to all the sample motion labels

$$\Phi(X) = \begin{bmatrix} \Phi_1(x^1) & \cdots & \Phi_1(x^{N_S}) \\ \cdots & \cdots & \cdots \\ \Phi_{N_b}(x^1) & \cdots & \Phi_{N_b}(x^{N_S}) \end{bmatrix} \tag{7.2}$$

,

then the best global polynomial model can be found by solving for the coefficient matrix W in $Y = W\Phi(X)$, which is also solved by the least squares pseudo-inverse method.

In principle, with basis functions of higher and higher polynomial degree the function approximation can be more and more accurate (if the basis includes all possible polynomial terms up to a certain degree, the model is a multi-dimensional Taylor expansion around the zero vector). However, the model can quickly become unwieldy, as the number of basis functions (and size of the coefficient matrix W) grows exponentially with the degree of the approximation.

Although it will be shown in the experimental section that global quadratic models improve the fidelity of the re-synthesis of the sample motions, it will also be shown that the synthesized outputs do not degrade gracefully when the input label goes outside the range of the training examples. This is the same behaviour as in the simple scalar input and output case, where it is known that the higher the polynomial order, the worse the extrapolation error.

7.2 Local radial basis functions

A further generalization of the method of learning weights for a set of basis functions is to use other sets of functions other than polynomial functions for the set of basis. One set of functions that has been widely studied is that of radial basis functions [4]. The basic idea behind radial basis functions is that instead of using global polynomials of high degree to learn a highly non-linear function, one can use many basis functions with local support, and each one encodes the local behaviour of the system. The n^{th} radial basis function is defined as:

$$\Phi_n(x) = exp((x - \mu_n)^t \Sigma_n^{-1} (x - \mu_n)) \tag{7.3}$$

Thus the n^{th} radial basis function is centered at μ_n in the input space, has value 1 when $x = \mu_n$, and the value decays to 0 as x moves away from μ_n at a rate controlled by the covariance 'spread' matrix Σ_n. As explained in [4], if we associate a radial basis function with each sample input such that for Φ_n we let $\mu_n = x^n$ (where x^n is the n^{th} sample input), then a matrix of weights can be learnt that will interpolate through the samples exactly. Just as in the global polynomial models, we solve for W in $Y = W\Phi(X)$, where now the matrix $\Phi(X)$ consists of the values of all the radial basis functions for all the sample inputs. Furthermore, for an appropriate value of the spread parameters Σ_n, the model can be made to interpolate smoothly between samples.

However, one may not want an exact interpolation through all the samples because a) one knows that there is noise in the samples, and/or b) it would result in too large a model (if there were many samples). In this case, one uses many less radial basis functions, and the basis centers, μ_n, also have to be learnt, along with the coefficient matrix W and the spread matrices Σ_n.

There are many different learning algorithms for optimizing the model parameters. Because of the highly non-linear nature of the model, it is very computation intensive to optimize over all the parameters. One simplification is to replace the spread matrices with a constant, an a priori specified value that is the same for all basis functions. This is the form of the model that we experimented with.

Although radial basis function models can provide good results in some instances, it suffers from three drawbacks that make it mostly unsuitable for learning models of human motion. First, the number of basis functions required to 'fill the input space' grows exponentially with the number of dimensions of the motion labels. Second, the basis functions are placed where the data lies in the input space, but if the input space is not sampled uniformly, there may be gaps, and then the model is not guaranteed to interpolate properly within large gaps. Finally, because of the local extent of each basis function, the model cannot extrapolate very well.

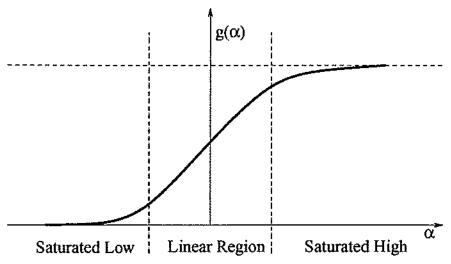
7.3 Feed-forward neural networks

The widely known method of feed-forward neural networks trained with the back-propagation scheme [24] can also be used to learn motion models. A network with two layers could be used , where the hidden units compute nonlinear functions $\phi_i(x)$ of the input label x, and the output units compute linear transformations of those hidden variables, $y_j = \sum_i W_{ji}\phi_i(x)$. The structure of the network is identical to that of the global polynomial interpolators, or the radial basis function networks; the only thing that changes is the functional form of the nonlinearities $\phi_i(x)$. In this type of network, the nonlinearities are sigmoidal activation functions:

$$\phi_i(x) = g(w_i^T x + b_i) \tag{7.4}$$

where

$$g(\alpha) = \frac{1}{1 + e^(-\alpha)} \tag{7.5}$$

(a) 

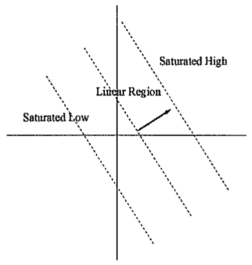
(b) 

Figure 7.1: (a): The nonlinear sigmoidal activation function that generates hidden layer outputs $\phi_i(x)$ (b): The partitioning of a 2-D input space into a linear region and two saturation regions. The width of the linear region is determined by the magnitude of the weight vector w.

Figure 7.1(a) shows the activation function of $g(\alpha)$. Near zero, g is linear, until it saturates at a value of 1 above, and 0 below. In equation 7.4, w_i and b_i define a separating hyper-plane in the input space (depicted for a 2-dimensional input space in the figure 7.1(b)). The width of the linear region depends on the magnitude of w_i; the smaller the magnitude, the larger the region.

When a FFNN is used for pattern classification, any particular output of the network is 'high' (value 1, say) when the input vector belongs to that class. For this type of application, the network produces useful computations because of the saturating property of the hidden units. Each hidden unit determines whether an input vector is on one side or the other of a hyper-plane boundary in the input space. The combined outputs of several hidden units can be used to define intricate boundaries between different regions of the input space, each region representing a particular class of inputs.

When a FFNN is used to learn a continuous function, the behaviour is very different. In this application, it is the linear region of each hidden unit that is important. Over the linear region, a hidden unit provides a linear approximation of the function being learned. In the saturation regions, it provides only a constant bias. A function is properly approximated only if the linear regions of each hidden unit cover the entire range of the input space. Because of the fact that each hidden unit saturates above and below, FFNNs inherently have difficulty extrapolating; beyond the input range specified by the training examples, it is likely that all hidden units become saturated.

There are ways to overcome this difficulty. For example, the input range over which extrapolation is desired can be specified a priori, and during the network training procedure the hidden unit weights w_i can be constrained to be small enough to guarantee that the linear region of each hidden unit is at least as wide as the desired extrapolation range. Another solution might be to use a new type of nonlinearity, which saturates only on one side. With such a nonlinearity, each hidden unit becomes a linear interpolant which 'turns on' on just one side of a hyper-plane that divides the input space in two halves.

Besides the extrapolation deficiency, FFNNs have other characteristics that make them less desirable for motion learning. They are very slow to train, require the use of validation data to determine when to stop training the network to prevent over-fitting, and the trained network needs to have its weights regularized to guarantee that marker outputs are continuous through time. For these reasons, the use of FFNNs was not further explored. Nevertheless, the analysis of FFNNs was fruitful, in the sense that it provides a clearer idea of the functional properties a good motion model should have. Namely, the input space should be separated into regions, with each region activating a local (linear) approximator. Some of the regions must be unbounded, so that the system extrapolates properly. To do so, a new network topology is required, as will be shown in the next sections.

7.4 Mixture models as local linear interpolants

We will arrive at a new model for motion learning by considering the learning problem from a different perspective: consider the set of input/output samples as the output of a stochastic system, and try to estimate a probabilistic model of the output of that system. This statistical approach to machine learning has gained considerable attention (and advocates) in recent years [23], mainly because of the rediscovery of a powerful class of algorithms (EM)[11] that enable the efficient estimation of complex probabilistic models. These new methods compete with and surpass the 'traditional' neural network approach to machine learning, not only because of the sophistication of the models that are possible to create, but also because those models are interpretable and analyzable, as opposed to the 'black box' nature of neural networks.

7.4.1 Connection between Gaussian model and linear interpolation

We start off by demonstrating the (simple, yet fundamental 'miracle' of linear algebra and statistics) relationship between Gaussian probability models and least squares linear interpolation. Then, by using a 'Mixture of Gaussians' probability model, we can derive the related local linear interpolant. Suppose we have N samples of input/output pairs, $\{(x_n, y_n), n = 0 \dots N\}$ (where in our case x is the label of a motion, and y is the resulting motion vector). Assuming we have enough samples, we can estimate a joint Gaussian distribution to model input/output pair system:

$$p(x,y) = \mathcal{N}\left(\begin{bmatrix}\mu_x \\ \mu_y\end{bmatrix}, \begin{bmatrix}\Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy}\end{bmatrix}\right) \tag{7.6}$$

where

$$\begin{bmatrix}\mu_x \\ \mu_y\end{bmatrix} = \frac{1}{N}\sum_n \begin{bmatrix}x_n \\ y_n\end{bmatrix} \tag{7.7}$$

and

$$\Sigma = \frac{1}{N}\sum_n \left(\begin{bmatrix}x_n \\ y_n\end{bmatrix} - \begin{bmatrix}\mu_x \\ \mu_y\end{bmatrix}\right)\left(\begin{bmatrix}x_n \\ y_n\end{bmatrix} - \begin{bmatrix}\mu_x \\ \mu_y\end{bmatrix}\right)^T \tag{7.8}$$

Given this joint Gaussian distribution, we can compute the conditional probability of the output y for a given input x:

$$p(y|x) = \mathcal{N}\left(\mu_y + \Sigma_{yx}\Sigma_{xx}^{-1}(x - \mu_x), \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy}\right) \tag{7.9}$$

From the conditional probability distribution we can compute the expected value of the output y given the value of the input x:

$$E(y|x) = \mu_y + \Sigma_{yx}\Sigma_{xx}^{-1}(x - \mu_x) = (\mu_y - \Sigma_{yx}\Sigma_{xx}^{-1}\mu_x) + \Sigma_{yx}\Sigma_{xx}^{-1}x \tag{7.10}$$

Note that $E(y|x)$ is a linear function of x. If we compare it to the least squares linear interpolant, we find that it is identical, as quickly shown below:

We want to find b and A such that $Ax + b$ minimizes the sum of square errors over all the samples.

$$E = \sum_n (y_n - (Ax + b))^T(y_n - (Ax_n + b)) \tag{7.11}$$

We find the optimal A and b by solving the system of equations

$$\begin{cases} \frac{\partial E}{\partial A} = 0 \\ \frac{\partial E}{\partial b} = 0 \end{cases} \tag{7.12}$$

Evaluating the derivatives we have

$$\begin{cases} \sum_n (y_n - (Ax_n + b))x_n^T = 0 \\ \sum_n (y_n - (Ax_n + b)) = 0 \end{cases} \tag{7.13}$$

and can be rewritten as

$$\begin{cases} A\sum_n x_n x_n^T + b\sum_n x_n^T = \sum ny_n x_n^T \\ A\sum_n x_n + bN = \sum_n y_n \end{cases} \tag{7.14}$$

solving for b in the second equation we have

$$b = \frac{1}{N}\left(\sum_n y_n - A\sum_n x_n\right) = \mu_y - A\mu_x \tag{7.15}$$

substituting this expression for b into the first equation we find

$$A = \cdots = \Sigma_{yx}\Sigma_{xx}^{-1} \tag{7.16}$$

Thus, from the parameters of the joint Gaussian model $p(x, y)$ we can derive the first, simplest motion model that we proposed, the global linear model (the least squares linear interpolant), as $E(y|x)$.

7.4.2 Only statistics of x needs to be accurate

Before extending our estimator to a more complex probabilistic model, we point out an important property of the method. We assumed in the derivation of $E(y|x)$ that we had sufficient input/output pair samples to build up an accurate joint Gaussian model. However, for our application this is impossible in practice, since the output vector y consists of all the marker coordinates in all the frames, and typically has over 2000 dimensions, whereas we can typically only acquire 100 or so samples. Yet, the GLM, now known to be intimately connected to the joint Gaussian distribution, does a fine job as a motion model. This apparent paradox is resolved by noting that, while with too few samples the covariance matrix Σ of the joint Gaussian model is severely rank deficient, the deficiency occurs in the Σ_{yy} sub-block, and this sub-block is not needed in computing $E(y|x)$. In fact, only the Σ_{xx} sub-block needs to be properly estimated and full rank (which is feasible, since x typically has only 3 or 4 dimensions), and the covariance of each dimension of y with x (corresponding to a row of Σ_{yx}) can be considered separately.

7.4.3 Mixture of Gaussians model

Suppose we wanted to create a more detailed/accurate model of the joint distribution of input/output pairs than a simple Gaussian. One choice is to use what is known as a Mixture of Gaussians [4]:

$$p(x, y) = \sum_i p(x, y|c_i)p(c_i) \tag{7.17}$$

According to this model, the overall distribution is the sum of the distributions of N_c individual Gaussian clusters $p(x, y|c_i)$, each cluster having an a priori probability of occurrence (called a mixture coefficient) $p(c_i)$ (and $\sum_i p(c_i) = 1$). Figure 7.2(a) shows a simple example where three Gaussian clusters are used to model the distribution of some observed data. In this two-dimensional example, the two variables measured could be the weight and the height of a person, and the sample set could be all the Olympic Marathon medalists. Then, given the measurements of a new person, one can perform a pattern recognition task - determine whether or not they are (likely to be) a marathon medalist. Figure 7.2(b) shows how a MOG model can be used for regression. If the data consists of dependent (y) and independent (x) variables, and the data is known to be representable by an injective function $y = f(x)$ (i.e.; for any given x there is a unique y), then a MOG model can be used to estimate y for a given value of x as $E(y|x)$.

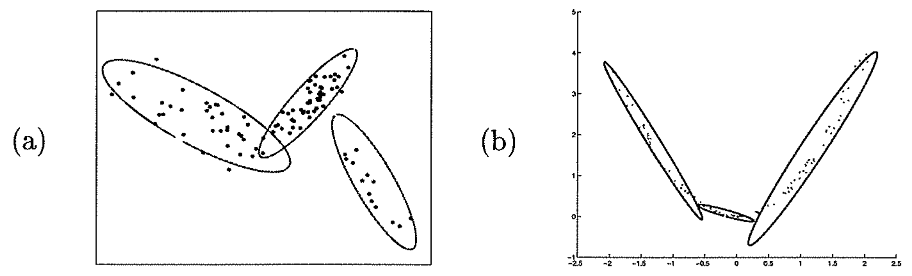


Figure 7.2: (a): A Mixture of Gaussians model can be used to model the distribution of data. Given a new sample, one can estimate the likelihood that it belongs to the distribution (a pattern recognition application). Ellipsi are 3-sigma bounds of the individual Gaussian clusters (b): given (possibly noisy) samples of a function $y = f(x)$, a MOG model can model the joint probability $p(x, y)$, which can then be used for regression to estimate y given x as $E(y|x)$.

Supposing that the parameters of the model (the mean and covariance of each cluster, as well as the cluster mixture coefficients) have been estimated in some way from the sample data, we can compute the conditional probability of the output given the input as:

$$\begin{aligned} p(y|x) &= \frac{p(x,y)}{p(x)} && (7.18) \\ &= \frac{\sum_i p(x,y|c_i)p(c_i)}{\sum_j p(x|c_j)p(c_j)} && (7.19) \\ &= \frac{\sum_i p(y|x,c_i)p(x|c_i)p(c_i)}{\sum_j p(x|c_j)p(c_j)} && (7.20) \\ &= \sum_i p(y|x,c_i)p(c_i|x) && (7.21) \end{aligned}$$

where

$$p(c_i|x) = \frac{p(x|c_i)p(c_i)}{\sum_j p(x|c_j)p(c_j)} \tag{7.22}$$

We can then compute the expected output given the input as:

$$E(y|x) = \sum_i E(y|x, c_i)p(c_i|x) \tag{7.23}$$

Thus, $E(y|x)$ is a weighted sum of Gaussian conditional expectation sub-models, where $p(c_i|x)$ defines the region of the input space each sub-model is responsible for. Figure 7.3 shows the corresponding network topology, which is different from those of all previous models. This new network has 'sub-model responsibility' nodes ($p(c_i|x)$), which combine multiplicatively with the output of the respective sub-models to generate the final output.

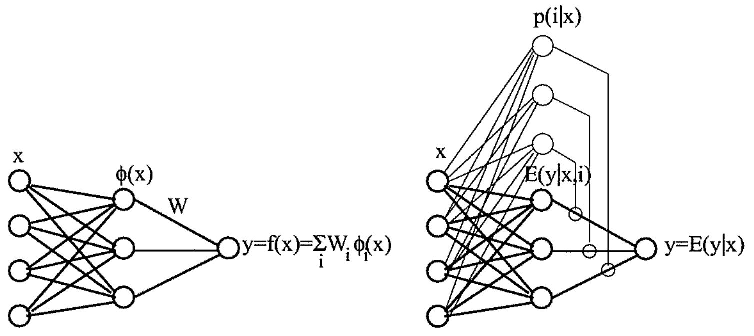


Figure 7.3: Comparison of network topologies. On the left is the network corresponding to a classical feed-forward network. Depending on the activation functions of the hidden layer, $\phi_i(x)$, the network can act as a global polynomial interpolator, as a RBF network, or the usual sigmoidal back-prop network. On the right is the network corresponding to a MOG as LLM (Mixture of Gaussians as a Local Linear Model). Each hidden unit (thick circles) is a (local) linear model corresponding to a single cluster, $E(y|x, i)$. The cluster weight neurons (thin circles, $p(i|x)$) modulate each cluster's output multiplicatively to form the network output.

Result with simple function $y = x^2$

Figure 7.4 shows the results of applying this method to a simple one-dimensional function $y = x^2$. In part(a), the MOG fit to 100 points sampled from the function over the range -2 to 2 (where Gaussian noise is added to the output) is shown. Fig. 7.4(b) shows the responsibility of each local model ($p(c_i|x)$) over the input range and beyond. Fig. 7.4(c)

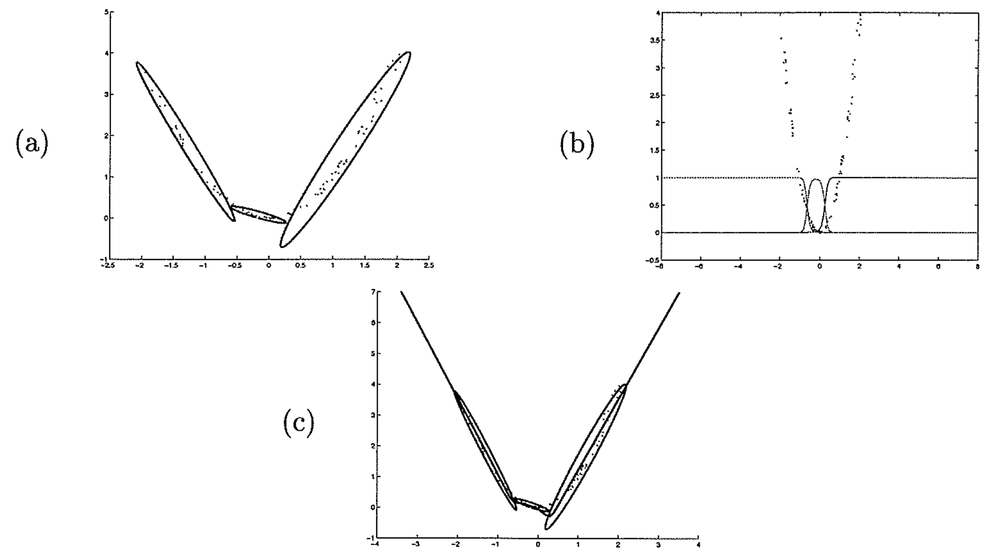


Figure 7.4: Application of MOG as Local Linear Model. (a): the fit of a 3 cluster MOG model to 100 noisy samples of the function $y = x^2$. (b): the responsibility of each local model as a function of x. (c): the resulting interpolation, and extrapolation with the closest local linear model.

shows the resulting interpolation of the function. Note that unlike other previous models (except for the GLM), this model provides a robust extrapolation beyond the training data input range using the closest linear sub-model.

Is the method optimal?

While we cannot provide a proof that this method finds the optimal local linear model (the algorithm does not minimize the interpolation error explicitly), we can argue that it is optimal for the degenerate case where there is only one cluster. Furthermore, empirically, it will be shown that the method reduces the interpolation error on training and testing data as compared to the GLM.

7.4.4 Learning the parameters of a MOG

EM algorithm

In the above sections we explained how to use a learned MOG model to create a LLM. In this section we explain how the MOG itself is derived. The parameters of the model are determined by maximizing the log likelihood of the observed data:

$$\begin{aligned} L(X,Y;\Theta) &= \log \prod_n p(x_n y_n;\Theta) && (7.24) \\ &= \log \prod_n \sum_i p(x_n y_n | c_i;\Theta) p(c_i;\Theta) && (7.25) \\ \Theta^* &= \arg\max_\Theta L(X,Y;\Theta) && (7.26) \end{aligned}$$

where Θ represents the set of model parameters. Due to the summation inside the logarithm, this maximization is a difficult, non-linear problem. However, the recent resurgence of the EM algorithm has brought to light a powerful, robust, and efficient iterative solution to the problem. The basic idea of the algorithm is to replace the log likelihood function with an expectation of the log likelihood, which is much easier to maximize. Appendix C outlines the details of the EM algorithm.

Cannot avoid estimation of Σ_{yy}

Although the EM algorithm is powerful in that it allows the estimation of complex models, it has one caveat with respect to our application. Unlike the simple Gaussian case, where it was shown that it is not necessary to estimate the output covariance Σ_{yy} in order to derive the linear interpolant $E(y|x)$, this can no longer be avoided, since computing $p(xy|c_i)$ is an integral part of the EM algorithm. Thus, for a typical action, we do not have enough sample motions to learn a model on the complete joint input/output space. For example, for the walking action, each sample motion is 45 frames long, and there are 18 body markers with 3-D coordinates, for a total of 45*18*3 = 2430 output dimensions. Since we only have available 120 or so sample motions, it is impossible to build the full statistics.

The solution of the problem is to break down the output into smaller groups, and learn separate models for each output subgroup. This grouping is subject to a perceptually based constraint: the 45 outputs representing an X, Y, or Z coordinate value of a single marker throughout time must appear smooth and continuous. If these outputs are separated into different subgroups, slight discontinuities may arise at the transition from one group to another, since each subgroup model is learned separately and it is difficult to enforce output continuity between models. Since even slight discontinuities are readily perceived and detract from the realism of the motion, all the time samples of a single marker and coordinate must be grouped together. For the walking action, this means each output subgroup must have at least 45 dimensions. This may still be too large compared to the number of samples available (especially since during EM training with N_c clusters, each cluster represents a subset of the samples, and if that subset has less than 45 samples, the cluster's statistics will be rank deficient). To further reduce the dimensionality of a subgroup, a PCA decomposition can be applied to the outputs. With the walking action, 10 principal components provided an encoding with less than 0.01cm error in any dimension. Thus, the MOG as LLM model of the walking action consisted of 18*3 = 54 MOG as LLM submodels with each submodel generating 10 PCA outputs.

How to chose model order

The EM algorithm determines the best parameters of a MOG model given the model order (number of clusters), but cannot determine the model order itself. However, choosing the proper order model is important. As the order model increases, the fit of the model to the data will also increase, but this fit is deceptive, because models of increasing order eventually begin to over-fit the data. As with other parametric models, the proper size of the model to use depends on both the inherent complexity of the data as well as the amount of data available for model estimation.

Some extensions of EM are being developed that enable the estimation of model order. The basic idea of the techniques is to include in the log-likelihood to be maximized a term which penalizes increased order complexity [6], [35]. The classical alternative, which we use here, is to use test data to cross-validate the model; if the interpolation error on the test data is much higher than that on the training data, we can conclude that over-fitting is taking place [4].

7.4.5 Similarity to Gershenfeld

The technique of using a MOG as a LLM outlined here is similar to the work of Gershenfeld et al. [13].

In their application, the digital Stradivarius project, the goal is to model the acoustic output of a violin given some control inputs, namely, the bow position, velocity, and pressure, and the position of the finger along the violin neck.

Their application differs from ours in that they only model a one-dimensional output (amplitude of vibration of a single violin string) whereas we model systems with a large number of outputs. Another major difference is that their inputs and output are continuous signals sampled at a high rate, making it possible to acquire large amounts of training data. Also, because of the continuity and high sample rate, they can make use of the powerful embedded output theorem [10], which can informally be stated as saying that for a continuous nonlinear system with d degrees of freedom there is a one-to-one mapping between the current state of the system and the past $2d + 1$ observed outputs. In our case, since each 'sample' is an entire motion, we can only collect orders of magnitude less training data, and since the samples are not collected as a continuum, we cannot make use of the embedded output theorem.

Finally, their technique is more general than ours, in the sense that they do not restrict the local models for the output given the input to be linear, but allow higher order polynomial interpolants as well. However, in practice, they use linear models because of the combinatorial explosion in the number of polynomial coefficients for higher order models, and the fact that linear interpolants are more robust (i.e.; there is less danger of computing a wild extrapolated value).

7.5 Recursive formulation: Stochastic Space-Time Constraints

The motion models presented so far are all 'one-shot' estimators, in the sense that the entire motion vector (body position throughout time) is used as the output to be learned and computed directly. However, it is also possible to create recursive, or frame-by-frame models, where an entire motion sample is computed with the repeated application of a model which represents human motion one frame at a time. This section describes such a model, called the Stochastic Space-Time Constraints (SSTC) model, in analogy to Witkin et al.'s Space-Time Constraints method of generating motion [36]. The technique is also inspired by Brand's technique of voice puppetry [7], where a similar system of constraints is used to ensure a continuous output signal from a hidden markov model.

The basic idea is as follows: similar to how Witkin uses the laws of physics to determine the motion of an object satisfying certain constraints (such as starting position and speed, and final position), we use a probabilistic model of the kinematics of human motion to compute the most probable (ie, most human-like) motion that satisfies given constraints. We first present the technique using a Gaussian model of the body kinematics, and then extend it to the use of more complex probability models.

Denote by $x_{t,n}$ the pose of the body (i.e., a vector consisting of the stacked 3-D coordinates of the 18 body markers) at time t of motion sample n. Then we can compute the joint probability distribution of consecutive frames, $p(x_t, x_{t+1})$, with mean and covariance given as

$$\begin{pmatrix} \mu_t \\ \mu_{t+1} \end{pmatrix} = \frac{1}{N_s * (T-1)} \sum_{n=1}^{N_s} \sum_{t=1}^{T-1} \begin{pmatrix} x_{t,n} \\ x_{t+1,n} \end{pmatrix} \tag{7.27}$$

$$\begin{bmatrix} \Sigma_{t,t} & \Sigma_{t,t+1} \\ \Sigma_{t+1,t} & \Sigma_{t+1,t+1} \end{bmatrix} = \frac{1}{N_s * (T-1)} \sum_{n=1}^{N_s} \sum_{t=1}^{T-1} \begin{pmatrix} x_{t,n} - \mu_t \\ x_{t+1,n} - \mu_{t+1} \end{pmatrix} \begin{pmatrix} x_{t,n}^T - \mu_t^T & x_{t+1,n}^T - \mu_{t+1}^T \end{pmatrix} \tag{7.28}$$

where T is the number of frames per sample motion, and N_s is the number of sample motions available for training. From the joint distribution we can derive the conditional distribution of x_{t+1} given x_t:

$$\begin{aligned} p(x_{t+1}|x_t) &= \mathcal{N}(\mu_{t+1} + \Sigma_{t+1,t}\Sigma_{t,t}^{-1}(x_t - \mu_t), \Sigma_{t+1,t+1} - \Sigma_{t+1,t}\Sigma_{t,t}\Sigma_{t,t+1}) && (7.29) \\ &\equiv \mathcal{N}(Ax_t + b, \Sigma) && (7.30) \end{aligned}$$

Note that the conditional distribution can be interpreted as a stochastic state evolution equation for the body pose:

$$x_{t+1} = Ax_t + b + \mathcal{N}(0, \Sigma) \tag{7.31}$$

Now suppose that we are given an initial body pose, x_0 as well as some (equality) constraints on the body pose in the future. For example, for the walking action, the constraint can be the position on the floor (w.r.t. the pivot foot) of the stepping foot at the end of the step. We denote this constraint $x_T^k = \mathit{constants}$, where the super-index k denotes the 'known' sub-vector of x_T (and x_T^u will denote the unknown sub-vector). We can use the conditional probability distribution $p(x_{t+1}|x_t)$ to find the most probable body motion that satisfies the initial condition and constraints. Denoting the entire trajectory by $X \equiv (x_0, x_1, \ldots, x_T)$, we find the optimal trajectory, X^*, by solving

$$\begin{aligned} X^* &= \arg\max_X P(X) && (7.32)\\ &\equiv \arg\max_X P(x_0, x_1, \ldots, x_T) && (7.33)\\ &= \arg\max_X p(x_T|x_{T-1}) \cdots p(x_2|x_1)p(x_1|x_0)p(x_0) && (7.34) \end{aligned}$$

subject to the known values of x_0 and x_T^k. Instead of working with $P(X)$ defined above, it is much more convenient to maximize the logarithm of $P(X)$ since it transforms the problem into one of linear algebra. Let L(X) be

$$\begin{aligned} L(X) &\equiv \log P(X) && (7.35)\\ &= (log \prod_{t=0}^{T-1} p(x_{t+1}|x_t))p(x_0) && (7.36)\\ &= \sum_{t=0}^{T-1} log(p(x_{t+1}|x_t)) + log(p(x_0)) && (7.37)\\ &= \sum_{t=0}^{T-1} \left(-\frac{1}{2}(x_{t+1} - (Ax_t + b))^T \Sigma^{-1}(x_{t+1} - (Ax_t + b)) + log(c) \right) + log(p(x_0)) \end{aligned}$$

where c is the normalizing constant of $p(x_{t+1}|x_t)$. Note that we have not calculated the parameters of $p(x_0)$, but they are irrelevant since x_0 is given and therefore constant. We can thus solve for the most likely trajectory X^* with the following steps:

- Generate a set of linear equations by computing the partial derivative of $L(X)$ with respect to the unknown (unconstrained) parts of $x_1, x_2, \ldots, x_T$ and equating them to zero. This creates a block tri-diagonal system of linear equations.
- Solve the system of equations with the efficient LU forward-backward substitution method [30].

The partial derivative of $L(X)$ with respect to x_t consists of two terms, one from

$p(x_t|x_{t-1})$ and the other from $p(x_{t+1}|x_t)$:

$$\frac{\partial L(X)}{\partial x_t} = -\frac{1}{2}(\Sigma^{-1} + \Sigma^{-T})(x_t - (Ax_{t-1} + b)) + \frac{1}{2}A^T(\Sigma^{-1} + \Sigma^{-T})(x_{t+1} - (Ax_t + b)) \tag{7.38}$$

To simplify notation, let $\Lambda \equiv \frac{1}{2}(\Sigma^{-1} + \Sigma^{-T})$. Then the above equation becomes

$$\frac{\partial L(X)}{\partial x_t} = -\Lambda(x_t - (Ax_{t-1} + b)) + A^T\Lambda(x_{t+1} - (Ax_t + b)) \tag{7.39}$$

Now if x_t consists of a known and an unknown part, x_t^k and x_t^u, we want to solve for X* by equating the derivative of $L(X)$ with respect to x_t^u to zero. But the derivative with respect to x_t^u is just the subset of rows of $\frac{\partial L(X)}{\partial x_t}$ corresponding to x_t^u. If we denote by S^{u_t} the matrix which picks out the rows corresponding to x_t^u (i.e., the identity matrix with the rows corresponding to x_t^k deleted; a conjugate definition can also be made for S^{k_T}), then the equation to solve is:

$$\begin{aligned}
& \frac{\partial L(X)}{\partial x_t^u} = 0 \\
\Leftrightarrow \quad & S^{u_t}\frac{\partial L(X)}{\partial x_t} = 0 \\
\Leftrightarrow \quad & -S^{u_t}\Lambda(x_t - (Ax_{t-1} + b)) + S^{u_t}A^T\Lambda(x_{t+1} - (Ax_t + b)) = 0 \\
\Leftrightarrow \quad & S^{u_t}\Lambda Ax_{t-1} - S^{u_t}(A^T\Lambda A + \Lambda)x_t + S^{u_t}A^T\Lambda x_{t+1} = -S^{u_t}(\Lambda + A^T\Lambda)b
\end{aligned} \tag{7.40}$$

Considering the general case where x_{t+1} and x_{t-1} may also have known and unknown parts, we can manipulate equation(7.40) to keep unknown sub-vectors on the left-hand side, and the known sub-vectors on the right. To do this, we use $S_{u_t} \equiv (S^{u_t})^T$ and $S_{k_t} \equiv (S^{k_t})^T$ to extract the corresponding columns of a matrix M multiplying the unknown and known sub-vectors of x_t, viz., $Mx_t = MS_{u_t}x_t^u + MS_{k_t}x_t^k$. The equation thus becomes:

$$\begin{aligned}
& \frac{\partial L(X)}{\partial x_t^u} = 0 \\
\Leftrightarrow \quad & S^{u_t}\Lambda AS_{u_{t-1}}x_{t-1}^u - S^{u_t}(A^T\Lambda A + \Lambda)S_{u_t}x_t^u + S^{u_t}A^T\Lambda S_{u_{t+1}}x_{t+1}^u = \\
& S^{u_t}\Lambda AS_{k_{t-1}}x_{t-1}^k - S^{u_t}(A^T\Lambda A + \Lambda)S_{k_t}x_t^k + S^{u_t}A^T\Lambda S_{k_{t+1}}x_{t+1}^k + -S^{u_t}(\Lambda + A^T\Lambda)b \\
\Leftrightarrow \quad & (\Lambda A)_{u_{t-1}}^{u_t}x_{t-1}^u - (A^T\Lambda A)_{u_t}^{u_t}x_t^u + (A^T\Lambda)_{u_{t+1}}^{u_t}x_{t+1}^u = \tilde{b}_t
\end{aligned} \tag{7.41}$$

where in the last line the notation of selecting rows with S^{u_t} and columns with $S_{u_{t-1}}$

(for example) has been simplified to $(\cdots)^{u_t}_{u_{t-1}}$, and $\tilde{b}_t$ denotes all the constant and known terms of the RHS.

Thus the entire set of equations obtained by equating the partial derivative of $L(X)$ with respect to each of $x_1^u, \ldots, x_T^u$ can be written as a block tri-diagonal system:

$$\begin{bmatrix} \Gamma_1 & -(A^T\Lambda)^{u_1}_{u_2} & & & & \\ & \ddots & & & & \\ & -(\Lambda A)^{u_t}_{u_{t-1}} & \Gamma_t & -(A^T\Lambda)^{u_t}_{u_{t+1}} & & \\ & & & \ddots & & \\ & & & & -(\Lambda A)^{u_T}_{u_{T-1}} & (\Lambda)^{u_T}_{u_T} \end{bmatrix} \begin{bmatrix} x_1^u \\ x_2^u \\ \vdots \\ x_{t-1}^u \\ x_t^u \\ x_{t+1}^u \\ \vdots \\ x_{T-1}^u \\ x_T^u \end{bmatrix} = \begin{bmatrix} \tilde{b}_1 \\ \vdots \\ \tilde{b}_t \\ \vdots \\ \tilde{b}_T \end{bmatrix}$$

where

$$\Gamma_\tau = (A^T\Lambda A + \Lambda)^{u_\tau}_{u_\tau} \qquad \text{for } \tau = 1 \ldots (T-1) \tag{7.42}$$

This system can be efficiently solved by LU back-substitution.

The need to replace Σ^{-1} with the Identity matrix

Experimentation with the method as described above reveals that the computed motions do not have a realistic appearance: the body shrinks and stretches, and distorts. To understand this behaviour, we take a closer look at the function that is maximized to generate the motion, equation (7.35). It can be interpreted as the sum over all time instances of the (negative) Mahalanobis distance between the pose at time $t+1$, x_{t+1}, and the pose predicted for time $t+1$ based on the pose at time t, Ax_t+b. Because of the structure of the covariance matrix Σ of the conditional probability distribution $p(x_{t+1}|x_t)$ differences in some marker coordinates contribute more to the Mahalanobis distance than others. For the walking action, the difference in contributions is very large: an error of 1cm in the X coordinate of the left wrist is 715 times less important than an error of 1cm in the Z coordinate of the right hip. By postulating that differences between actual and predicted poses in the optimizing function 7.35 should be weighted equally in all dimensions, we can replace the Mahalanobis weight Σ^{-1} with the Identity matrix. With this modification, the method produces well-behaved motions.

7.6 SSTC with MOG conditional probability distributions

In this section we extend the method described above to the case where $p(x_{t+1}|x_t)$ is modeled with a mixture of Gaussians instead of a single Gaussian. Just as in section 7.4.4, the EM algorithm can be used to estimate a mixture of Gaussians model for the joint p.d.f. $p(x_t, x_{t+1})$ (and in this case enough data is available to properly estimate the model parameters; the joint poses have 104 dimensions, and we typically have available 5000 sample pose pairs). The conditional p.d.f. is then calculated as

$$p(x_{t+1}|x_t) = \sum_{i=1}^{N_c} p(x_{t+1}|x_t, c_i)p(c_i|x_t) \tag{7.43}$$

where

$$p(c_i|x_t) = \frac{p(x_t|i)p(i)}{\sum_j p(x_t|j)p(j)} \tag{7.44}$$

and for simplicity of notation, the i^{th} (j^{th}) cluster, c_i (c_j) will be represented merely by i (j), and summations will be assumed to be over all clusters, as in the right-hand side of the last equation above. As in the previous section, the most human-like motion is computed by solving the maximization problem

$$\begin{aligned}
X^* &= \arg_X \max L(X) && (7.45)\\
&= \arg_X \max \log P(X) && (7.46)\\
&= \arg_X \max \log \left(\prod_{t=1}^{T-1} p(x_{t+1}|x_t) \right) p(x_0) && (7.47)\\
&= \arg_X \max \log \left(\prod_t \sum_{i_t} p(x_{t+1}|x_t, i_t)p(i_t|x_t) \right) p(x_0) && (7.48)
\end{aligned}$$

given the initial state x_0 and final pose constraints. Unfortunately, due to the summations over all clusters (and the terms $p(i_t|x_t)$) at each time instance, the expression to maximize does not reduce to a sum of quadratic forms as in the previous section. Solving for the optimal X^* becomes a difficult nonlinear optimization problem. Fortunately, an EM-based algorithm (derived in section C.2 of Appendix C) can be used to do the maximization. The basic idea is the following: instead of trying to maximize the log probability of the observed variables X directly, one can iteratively maximize a function $Q(X)$, the expected log of the joint probability of X and the hidden variables $I = (i_1, \dots, i_t, \dots, i_{T-1})$, taking the expectation with respect to I, given some current estimate of the distribution of the hidden variables. Here we derive the method of maximizing $Q(X)$ that is to be done at each EM iteration. The resulting solution is a block tri-diagonal system of linear equations which is the generalization of the system of equations of section 7.5.

Let $Q(X)$ be the expected log probability:

$$\begin{aligned} Q(X) &= E_I(\log P(X, I)) && (7.49) \\ &= \sum_{i_1} \cdots \sum_{i_t} \cdots \sum_{i_{T-1}} r(I) \log P(X, I) && (7.50) \\ &= \sum_I r(I) \log P(X, I) && (7.51) \end{aligned}$$

where $I = (i_1, \dots, i_t, \dots, i_{T-1})$ is summed over all possible clusters at each time instant, and $r(I)$ is computed based on the current estimate $\hat{X} = (\hat{x_1}, \cdots, \hat{x_T})$ of the maximizing trajectory (as per the EM algorithm derived in Appendix C. Namely,

$$r(I) = r(i_1) r(i_2) \dots r(i_t) \dots r(i_{T-1}) \qquad (7.52)$$

where each $r(i_t)$ is of form $r(i_t) = p(i_t|\hat{x}_t)$. $Q(X)$ then becomes

$$\begin{aligned}
Q(X) &= \sum_{i_1} \cdots \sum_{i_{T-1}} r(i_1) \cdots r(i_{T-1}) \log P(X, I) \\
&= \sum_{i_1} r(i_1) \cdots \sum_{i_{T-1}} r(i_{T-1}) \log P(X, I) \\
&= \sum i_1 r(i_1) \cdots \sum_{i_{T-1}} r(i_{T-1}) \log \prod_{t=1}^{T-1} p(x_{t+1}|x_t, i_t) p(i_t) * p(x_0) \\
&= \sum i_1 r(i_1) \cdots \sum i_{T-1} r(i_{T-1}) \sum_{t=1}^{T-1} \log p(x_{t+1}|x_t, i_t) p(i_t) + \log p(x_0) \\
&= \sum i_1 r(i_1) \cdots \sum i_{T-1} r(i_{T-1}) \sum_{t=1}^{T-1} \Big(-\frac{1}{2}(x_{t+1} - (A_{i_t} x_t + b_{i_t})^T \Sigma_{i_t}^{-1} (x_{t+1} - (A_{i_t} x_t + b_{i_t})) \\
&\quad + \log p(i_t) + \log(const)\Big) + \log(p(x_0) \qquad (7.53)
\end{aligned}$$

Now to find the trajectory that maximizes $Q(X)$ we set the partial derivatives of $Q(X)$ with respect to each x_t to zero. Each partial derivative (except for the one with respect to x_T, where one term is missing) has form:

$$\begin{aligned}
\frac{\partial Q(X)}{\partial x_t} &= \sum_{i_1} r(i_1) \cdots \sum_{i_{T-1}} r(i_{T-1}) \Big(-\frac{1}{2}(\Sigma_{i_{t-1}}^{-1} + \Sigma_{i_{t-1}}^{-T})(x_t - (A_{i_{t-1}} x_{t-1} + b_{i_{t-1}})) \\
&\quad + \frac{1}{2} A_{i_t}^T (\Sigma_{i_t}^{-1} + \Sigma_{i_t}^{-T})(x_{t+1} - (A_{i_t} x_t + b_{i_t}))\Big) \\
&= \sum_{i_t} r(i_t) \sum_{i_{t-1}} r(i_{t-1}) \Big(-\frac{1}{2}(\Sigma_{i_{t-1}}^{-1} + \Sigma_{i_{t-1}}^{-T})(x_t - (A_{i_{t-1}} x_{t-1} + b_{i_{t-1}})) \qquad (7.54) \\
&\quad + \frac{1}{2} A_{i_t}^T (\Sigma_{i_t}^{-1} + \Sigma_{i_t}^{-T})(x_{t+1} - (A_{i_t} x_t + b_{i_t}))\Big)
\end{aligned}$$

(the last equality holds since $\sum_{i_\tau} r(i_\tau) = 1$ for all i_τ). Defining $\Lambda_{i_t} = \Sigma_{i_t}^{-1} + \Sigma_{i_t}^{-T}$ we can further simply the partial derivative:

$$\frac{\partial Q(X)}{\partial x_t} = \sum_{i_{t-1}} r(i_{t-1}) \Lambda_{i_t} (x_t - (A_{i_{t-1}} x_{t-1} + b_{i_{t-1}})) - \sum_{i_t} r(i_t) A_{i_t}^T \Lambda_{i_t} (x_{t+1} - (A_{i_t} x_t + b_{i_t}) \qquad (7.55)$$

Now if we define new variables representing various averages over clusters, namely:

$$\overline{\Lambda}_{t-1} = \sum_{i_{t-1}} r(i_{t-1})\Lambda_{i_{t-1}} \tag{7.56}$$

$$\overline{\Lambda A}_{t-1} = \sum_{i_{t-1}} r(i_{t-1})\Lambda_{i_{t-1}} A_{i_{t-1}} \tag{7.57}$$

$$\overline{\Lambda b}_{t-1} = \sum_{i_{t-1}} r(i_{t-1})\Lambda_{i_{t-1}} b_{i_{t-1}} \tag{7.58}$$

$$\overline{A^T\Lambda}_t = \sum_{i_t} r(i_t)A^T_{i_t}\Lambda_{i_t} \tag{7.59}$$

$$\overline{A^T\Lambda A}_t = \sum_{i_t} r(i_t)A^T_{i_t}\Lambda_{i_t}A_{i_T} \tag{7.60}$$

$$\overline{A^T\Lambda b}_t = \sum_{i_t} r(i_t)A^T_{i_t}\Lambda_{i_t}b_{i_t} \tag{7.61}$$

we can equate the partial derivative to zero and arrive at an expression similar to that of the single Gaussian case (equation 7.39):

$$\frac{\partial Q(X)}{\partial x_t} = \overline{\Lambda}_{t-1}x_t - (\overline{\Lambda A}_{t-1}x_{t-1} + \overline{\Lambda b}_{t-1}) - \overline{A^T\Lambda}_t x_{t+1} + (\overline{A^T\Lambda A}_t x_t + \overline{A^T\Lambda b}_t) \tag{7.62}$$

Just as in the single Gaussian case, the partial derivative is equated to zero, and the known parts of each x^t can be separated from the unknown parts to create a similar block tri-diagonal system of linear equations.

In summary the system of equations to solve at each iteration of the EM algorithm is the same as in the single Gaussian model case, with all the entries replaced by averages over the clusters. Just as in the single Gaussian case, all the Σ_{i_t}'s are replaced with the identity matrix in order for the method to work properly.

Chapter 8 Motion synthesis experiments

8.1 Perceptual evaluation of realism

The perceptual quality of the motion synthesis method was evaluated with formal and informal tests with various subjects.

In the formal tests, subjects were presented a two alternative forced choice paradigm where they were asked to distinguish between original and re-synthesized motions. Two actions were tested, 2D reaching, and 2D drawing. The motions were presented as moving light displays [22], and each subject viewed 30 stimuli pairs. For each stimuli pair, an original motion was randomly chosen from samples in the motion capture dataset, and the corresponding motion label was used to create a synthetic motion. For the reaching action, a 3rd order polynomial model was used, while for the drawing action a linear model was used. After presenting the stimuli pair (with random order of appearance between the original and synthesized motion), the subject chose which appeared more realistic. In case the subject was unsure, he was instructed to guess. If the true and re-synthesized motions were completely indistinguishable, subjects should perform the task at chance level (50% correct responses). The results in tables 8.1 and 8.2 show that indeed it was difficult for subjects to distinguish between real and synthetic motions.

Various informal tests were also conducted. During the development of the different

Subject	**% Correct**
B.P.	36.7
L.G.	60.0
Y.S.	46.7
Y.K.	63.3
P.M.	60.0
P.P.	46.6
J.B.	33.3
Mean	49.51
St. dev.	11.93

Table 8.1: Discriminability between original and reconstructed reaching motions.

Table 8.2: Discriminability between original and reconstructed drawing motions.

Subject	**% Correct**
B.P.	50.0
L.G.	50.0
M.M.	46.7
A.B.	43.3
E.D.	46.6
J.B.	53.3
Mean	48.28
St. dev.	3.59

motion model techniques, perceptual tests were always used to assess model quality. Perhaps the most significant perceptual tests were those where the interactive demo described in section 6.8 (and its precursor, a Matlab demo where the user defines a path along which the computer synthesizes a character walking) was shown to professional game developers at a Game Developer's Conference (September 98) and in private meetings (August 99). They were unanimously impressed by the quality of the animation, many claiming that they had never seen such high quality real-time animation.

8.2 Numerical Accuracy of GLM, GQM, MOG as LLM models

In this section we analyze the numerical accuracy of the motion models.

Figure 8.1 shows the RMS error (in cm) of each marker coordinate when the GLM (global linear model), GQM (global quadratic model), and MOG as LLM models are used to learn the Left Step action. All 124 samples were used to train, and the RMS error of the i^{th} output component y^i is computed as

$$E^i_{RMS} = \left(\frac{1}{N_{samples}} \sum_{n=1}^{N_{samples}} (y^i_{true} - y^i_{synth})^2 \right)^{\frac{1}{2}} \quad (8.1)$$

The figure shows that higher order models have less error, with the GQM of comparable accuracy as a MOG as LLM with 4 clusters. However, the improved accuracy may be due to over-fitting.

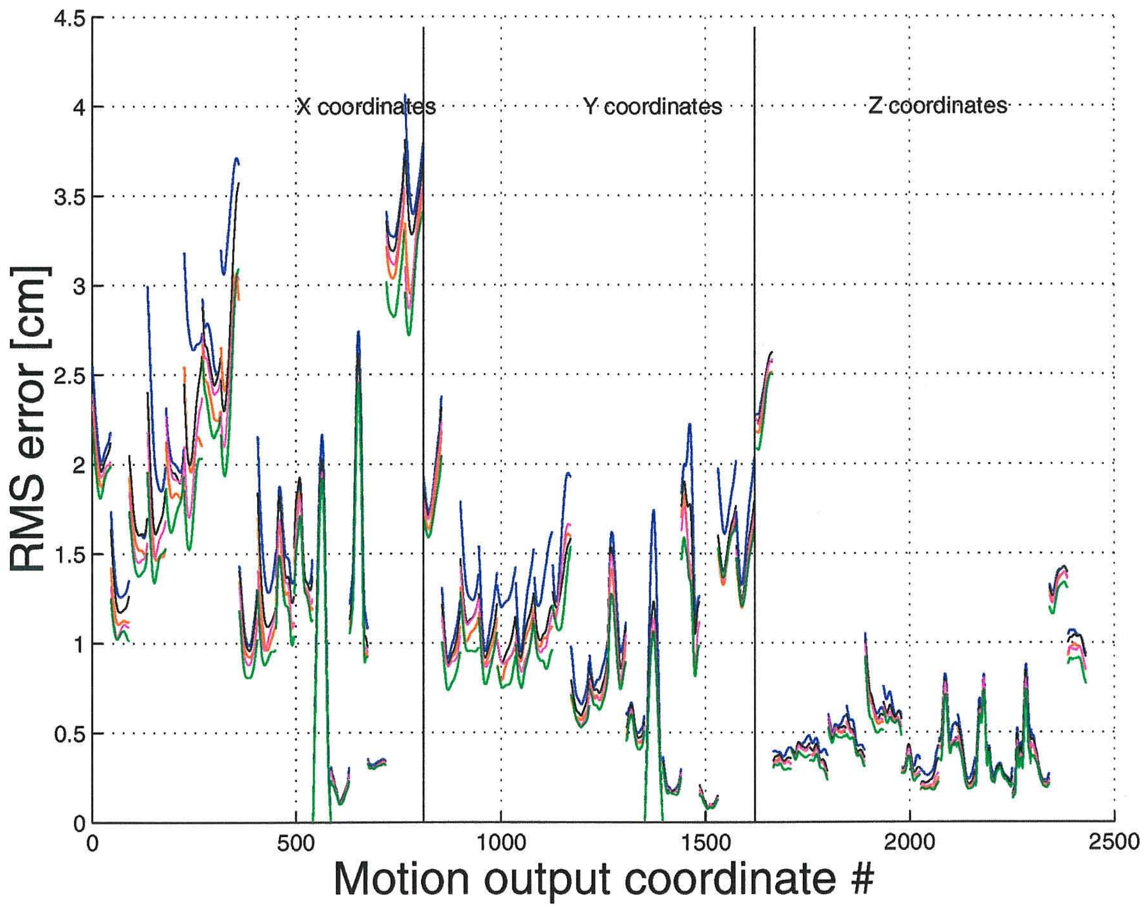


Figure 8.1: RMS error of reconstruction for each motion output coordinate. blue: GLM, red: GQM, black: MOG as LLM with 2 clusters, magenta: MOG as LLM with 4 clusters, green: MOG as LLM with 8 clusters.

To investigate this possibility, a cross-validation test was done. The samples were separated into groups of 4 samples. At each iteration, one of the groups was set aside as testing data, and the model trained with the remaining groups. The RMS error was then computed for the training and testing data. These errors were averaged over all the iterations to create table 8.5. Since the MOG as LLM models require an initial cluster initialization, approximately 1000 iterations were executed (32 separate MOG as LLM initializations for each test group). In this way any possible spurious results due to misfortunate initializations will be averaged out. Also, since separate MOG as LLM models need to be learned for each marker coordinate, for sake of time efficiency only one marker coordinate was analyzed. The marker coordinate analyzed is the X coordinate of the right wrist. This marker in particular was chosen because from fig. 8.1 (8th set of plots counting from left) it seems that using higher

Model	Train rms error (std)	Test rms error (std)
GLM	22.84 (0.30)	22.82 (7.71)
GQM	18.40 (0.20)	20.87 (6.06)
MOG2	19.84 (0.56)	20.88 (6.02)
MOG3	18.72 (0.49)	20.63 (5.86)
MOG4	18.14 (0.48)	20.63 (5.65)
MOG8	16.74 (0.63)	22.16 (4.49)

Table 8.3: Cross-validation results of the various motion models.

order models produced a significant improvement in accuracy.

Table 8.2, just as figure 8.1, shows that for the training set, increased order model always decreases the RMS error. However, this is not always true for the RMS error on the test set. The GQM does reduce the test RMS error compared to that of the GLM, indicating that over-fitting has not occurred. Likewise, the MOG as LLM with 2 and 3 clusters seem to be legitimately improving the modeling accuracy. With 4 clusters, the testing error does not decrease further (whereas the training error does), and with 8 clusters, it increases. Thus for MOG as LLM models with 4 or more clusters the reduction in training RMS error is in fact due to over-fitting of the data.

8.3 Extrapolation comparison of GLM, GQM, MOG as LLM for walking

Besides cross-validation tests of the accuracy of the models, the models must also be tested in the way they will be used to synthesize new motions. This is particularly important for the walking action, because the labels used as input to define the walking step may cause the model to extrapolate. To test how well each model can extrapolate, the interactive tool shown in figure 8.2 was used. Using the mouse, a path is outlined, and the computer then places footsteps along the path (using an algorithm similar to that of the real-time interactive demo) so that the character begins with feet side-by-side (standing), walks along the path, and comes to a stop (feet side-by-side again). Then a model is chosen, and the resulting motion is synthesized and viewed. Figure 8.3 shows the foot trajectories of a path, superimposed on the trajectories of all the training samples (and plotted in the local reference frame of the pivot foot of each step). It is clear from the figure that the automatic footstep placement does not place the footsteps exactly over the region covered by the training samples; the models will be extrapolating. When the animations of the different models are compared, it is found that:

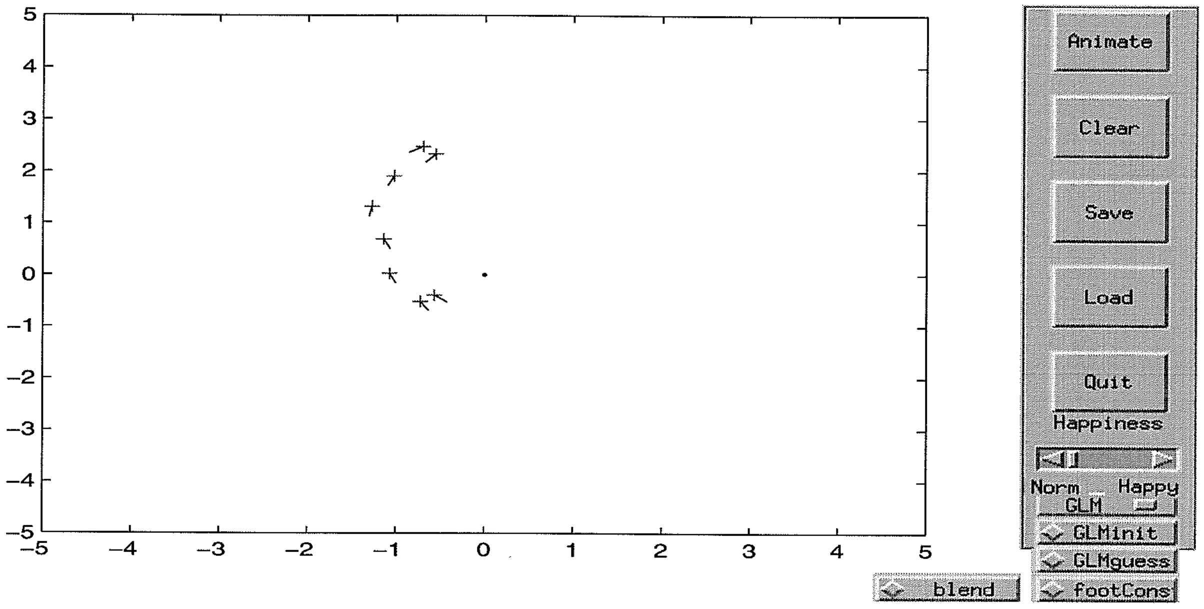


Figure 8.2: Tool for synthesizing new motions. The user can choose which motion model to use, and can input any path for the character to follow.

- The GLM extrapolates quite well. Motion is fluid, natural.
- The GQM does not extrapolate as well. The character oscillates back and forth with each step in an unnatural way. This behaviour is not unexpected, since high-order polynomial models tend to extrapolate poorly.
- The MOG as LLM with 2 and 3 clusters also extrapolate well. The motion is very similar to that of the GLM, but with subtle differences. The motion now seems more variable, with slight differences in the timing of the motion of the different body parts for starting and stopping, and turning steps.
- For the MOG as LLM with 4 or 8 clusters the variations in motion for different types of steps is accentuated, but unnatural. This is clearly a result of the fact that the models have over-fit the data and do not generalize well.

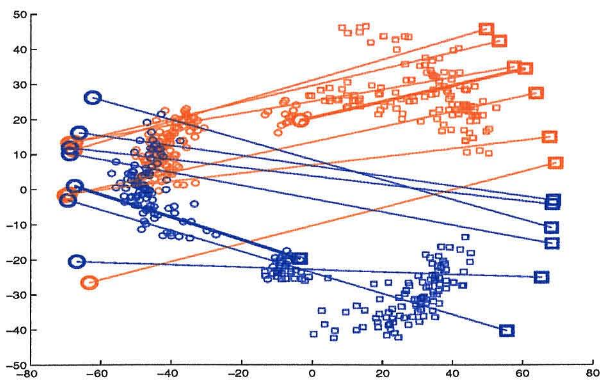


Figure 8.3: Automatic foot placement based on path input by user. Smaller circles/squares are start/stop foot positions in the data used to learn the motion model. Larger circles/squares are the foot positions generated automatically based on the path input by the user. Pivot foot is at origin of plot, steps with the left/right foot are in red/blue.

8.4 Analysis of SSTC and SSTC with MOG

8.4.1 Definition of synthesis conditions

In this section, the behaviour of the SSTC and SSTC with MOG models are analyzed with the left-step walking data set, and by synthesizing new walking animations. To do so, we need to keep in mind that for these models, labels per se are not used to define the motion, but rather initial and final pose conditions are used. There are several choices for how to define the boundary conditions, which affects the resulting motion:

When synthesizing an animation sequence, it is desirable to use the previous motion's (previous step's) final pose as the initial condition for the current step. Alternatively (and necessary at least for the first step of the animation), the initial pose predicted for the current step by the GLM (given the footsteps that define the desired animation sequence) could be used as the initial condition. These choices will be referred to as the prevInit and GLMInit cases. Note that with the prevInit constraint the SSTC method solves for 45 poses (the entire new step), whereas if the GLM initial pose is used, then the method solves for only 44 poses since the very first one is defined. The choices for the final pose constraint could be: 1) the desired feet position at the end of the step (footEnd constraint), or 2) the entire final body pose computed by the GLM (or any other model, for that matter), called the GLMend constraint. Finally, for models with more than one cluster (SSTC with MOG), there is also the freedom to define the initial guess of the cluster's weights. Either an equi-weight initial guess (equiGuess), or the cluster weights that correspond to the GLM motion solution (GLMguess) could be used .

When comparing the accuracy of synthesis with respect to the original samples, the previous final pose is not available*, but we can use the sample's true initial pose as initial condition (trueInit). Again as an alternative, the initial pose predicted by the GLM (based on the label associated with the sample) can be used as initial condition (GLMinit). For the final pose constraint, besides the footEnd and GLMend constraints, there is a third choice: the true final pose from the data sample (trueEnd). Likewise, a third option is available as initial guess for the cluster weights, those derived from the true motion itself (trueGuess).

8.4.2 Results with re-synthesis of acquired data

Figure 8.4 shows the RMS error for all markers throughout time between the SSTC synthesized motion and the sample motion. Each colored line represents different combinations of initial and final pose conditions. The RMS error of the GLM is plotted in black for comparison, and it appears to be a lower bound for the model, regardless of the choice for initial and final pose conditions. Figure 8.5 is a closeup of the plot, showing the RMS output for 11 marker coordinates. The errors corresponding to using the feet final constraint grow monotonically from the first frame to the last, and can end up two to three times as large as the GLM error on the last frame. This can be explained in the following way: the conditional probability model of the pose at time t given the pose at time $t-1$ can be viewed as defining the state evolution of an autonomous (autoregressive) system. Given an initial state, the system will evolve freely - but not to the state that we want, which is why a final pose constraint must be given. The SSTC method finds a trajectory through pose space which satisfies the final constraints while deviating minimally from the natural flow of the system. Now if only a partial final condition is imposed (instead of constraining the entire body pose with the true pose or the GLM-predicted pose), the trajectory is more free to follow the autonomous flow. That the final pose error can be much larger than that of the GLM, and that it increases monotonically even given the proper initial condition, are indications that a single cluster conditional probability model does not model the dynamics

*Technically, the previous step's final pose is available in the original data sequences for most steps - but not all due to occasional missing data in the mocap process. Nevertheless, using the first pose of the actual step is for all effects equivalent since the two poses are contiguous in the data stream, separated by 1/60th of a second.

accurately. Also in the plot it is seen that when the entire final pose is constrained (either with the trueEnd or GLMend constraints), the time evolution of the error is oscillatory, a further indication of not being a very accurate model.

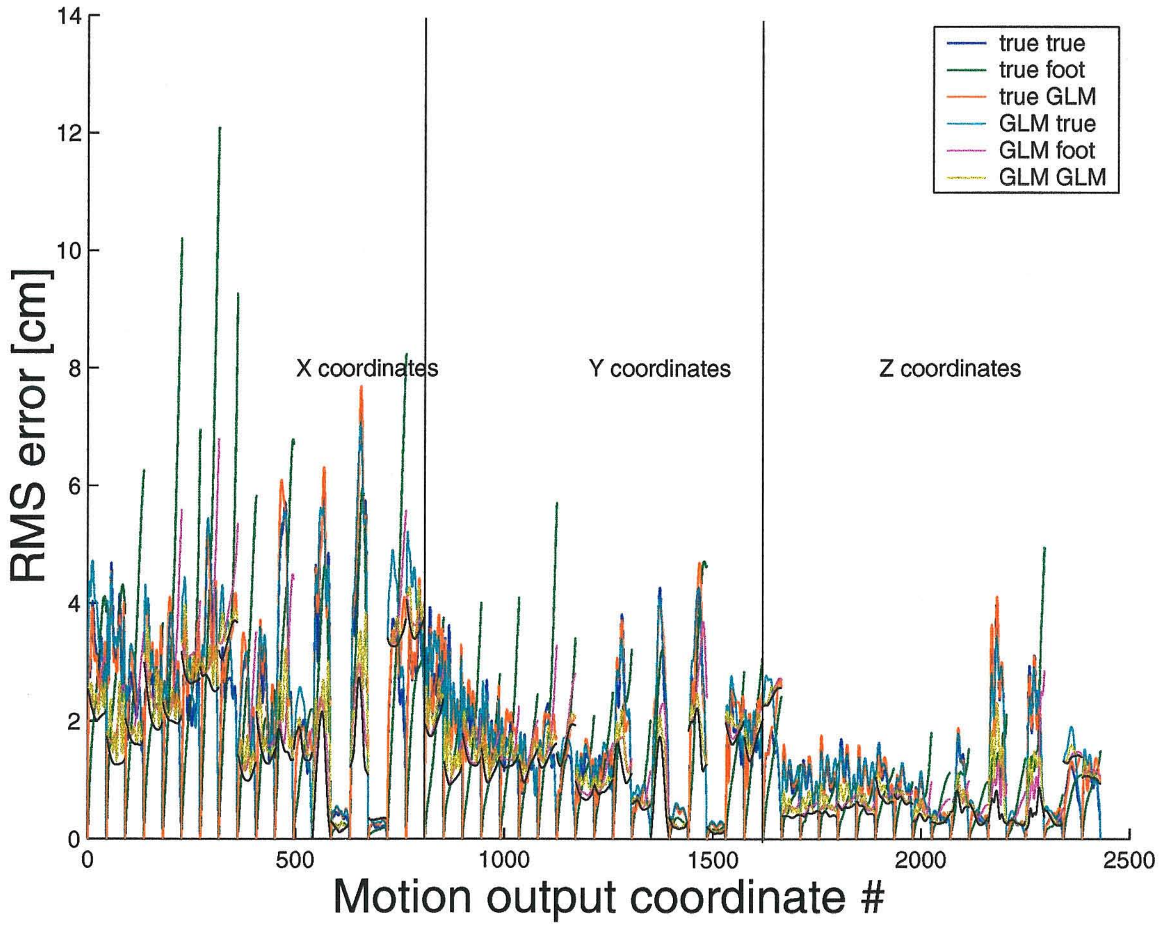


Figure 8.4: RMS error of the SSTC model for all the motion output coordinates. GLM error in black for reference.

Figures 8.6 and 8.7 show RMS error plots for the SSTC with MOG model with 2 clusters. Solid lines represent results with equiGuess cluster weights, dotted lines with GLMguess, and dash-dot lines those of true-motion initial weights. Using two clusters to model the conditional probability function greatly improves the synthesis accuracy. Now, for the cases with the partial footEnd constraints, the error on the last frame is close to that of the GLM (albeit still monotonically increasing). With the (trueInit, trueEnd, trueGuess) case, the resulting motion is approximately twice as accurate as the GLM-computed motion. Furthermore, with strong, full body pose final constraints (either trueEnd or GLMend), the

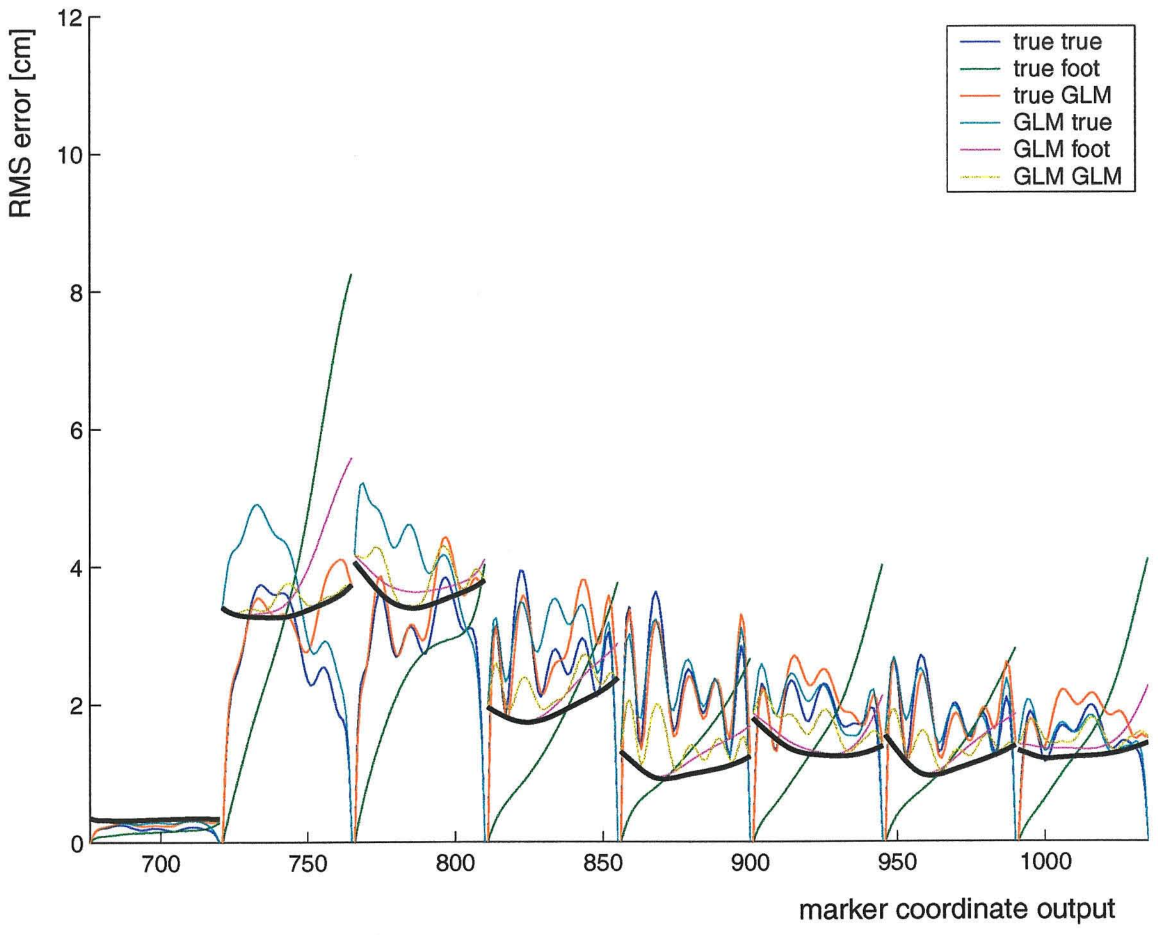


Figure 8.5: Closeup of RMS error of the SSTC model for some motion output coordinates. Shown are the errors for the x-coordinate of the right foot, and two side head markers, followed by the y-coordinate errors of head, neck, left and right shoulder, and left elbow.

errors vary smoothly through time, and are not oscillatory - another indicator of a good model.

Figures 8.8 and 8.9 show the RMS errors for the SSTC with MOG model with 8 clusters. Under almost all settings, the errors are greater than for the model with only 2 clusters. However, for the (trueInit, trueEnd, trueGuess) case the error is even smaller than the 2 cluster case. This behaviour indicates that synthesis with 8 clusters gets stuck in local minima, since a very accurate estimate for the initial cluster weights is needed to produce a good synthesis. Unfortunately, since when synthesizing new motions only the equiGuess or GLMguess can be used for weight initialization, it is likely that the 8 cluster model will not synthesize well.

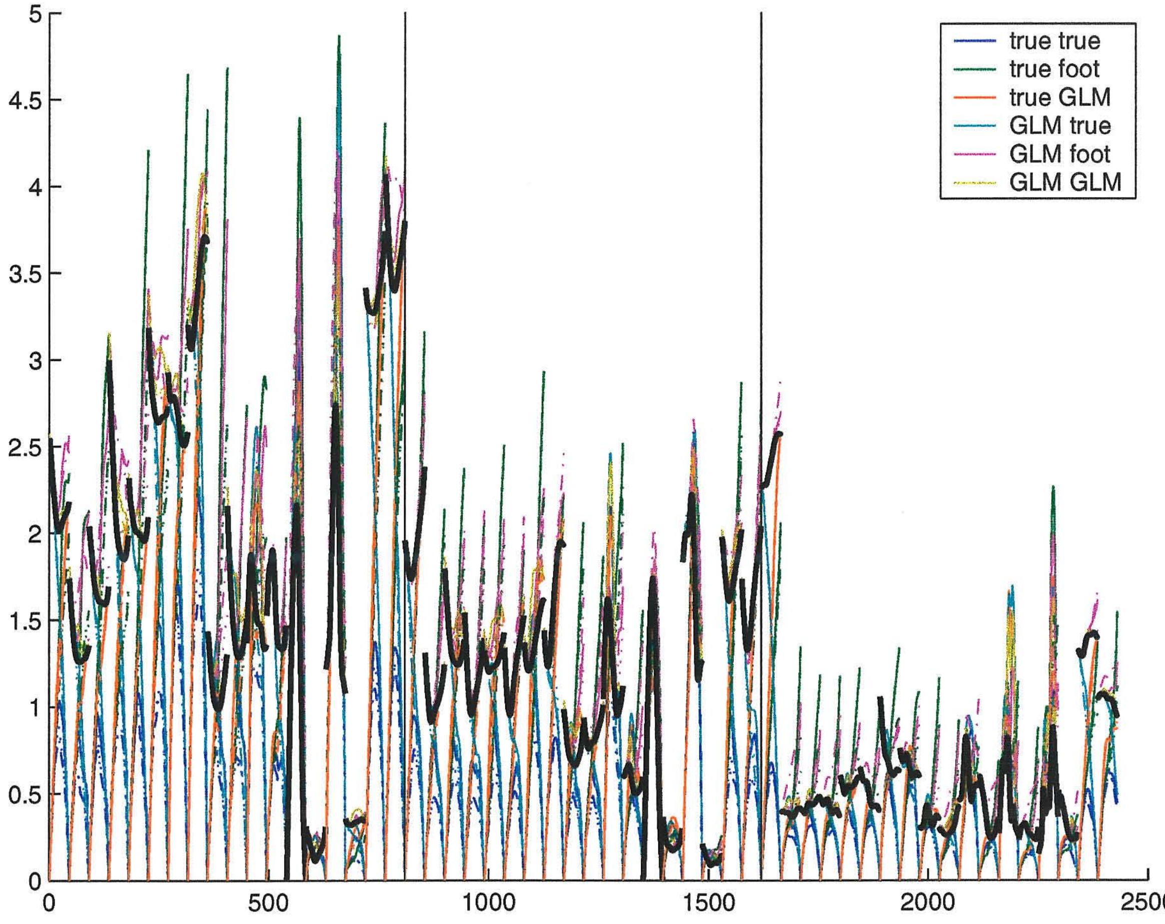


Figure 8.6: RMS error of the SSTC with 2 Cluster MOG probability model for all the motion output coordinates. GLM error in black for reference. Solid lines: equi-weight initial cluster weights, Dotted: GLM initial weight guess, Dashed: True motion weight guess.

8.4.3 Results of synthesizing new motions

Table 8.4.3 summarizes the results of experiments where new motions were synthesized with the SSTC models. The same tool as in section 8.2 was used to generate paths and footsteps.

In principle, the recursive formalism allows for greater adaptability than label-based methods such as the GLM; given the current pose, the most likely motion satisfying future constraints can be generated. In practice, however, this seems difficult to achieve. As the table indicates, when the previous pose is used as initial condition (prevInit) and only the final feet position is used as final constraint (footEnd constraint), the method is in fact unstable. This happens due to three main reasons. First, the conditional probability model $p(x_{t+1}|x_t)$ is scale invariant, in the sense that multiplying both x_t and x_{t+1} by a constant

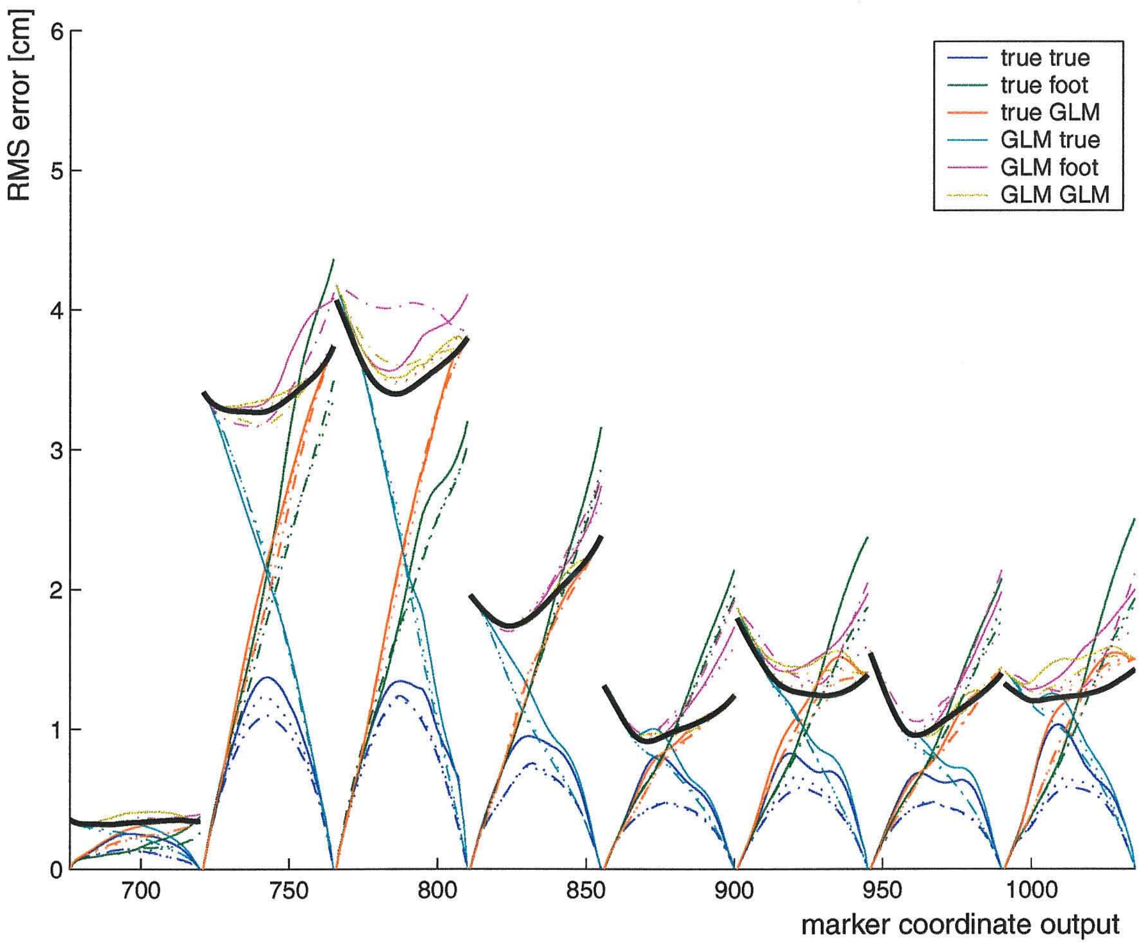


Figure 8.7: Closeup of RMS error of the SSTC with 2 Cluster MOG probability model for some motion output coordinates. Shown are the errors for the x-coordinate of the right foot, and two side head markers, followed by the y-coordinate errors of head, neck, left and right shoulder, and left elbow. Solid lines: equi-weight initial cluster weights, Dotted: GLM initial weight guess, Dashed: True motion weight guess.

does not affect the probability. Second, the resulting state evolution equation derived from $p(x_{t+1}|x_t)$ has some eigenvalues outside of the unit circle in the z-plane. This means that the autonomous state flow is divergent. Finally, using only the footEnd constraint does not sufficiently constrain the flow. Thus, at the end of the step the resulting pose is 'inflated'. Since this pose is then used as initial condition for the next step, the inflation continues, resulting in an unstable synthesis.

One way to eliminate the instability is to impose a stronger end constraint. If the end pose predicted by the GLM is used as end constraint, then the pose is guaranteed to have the proper scale, and the instability is avoided. However, the resulting motion is still not

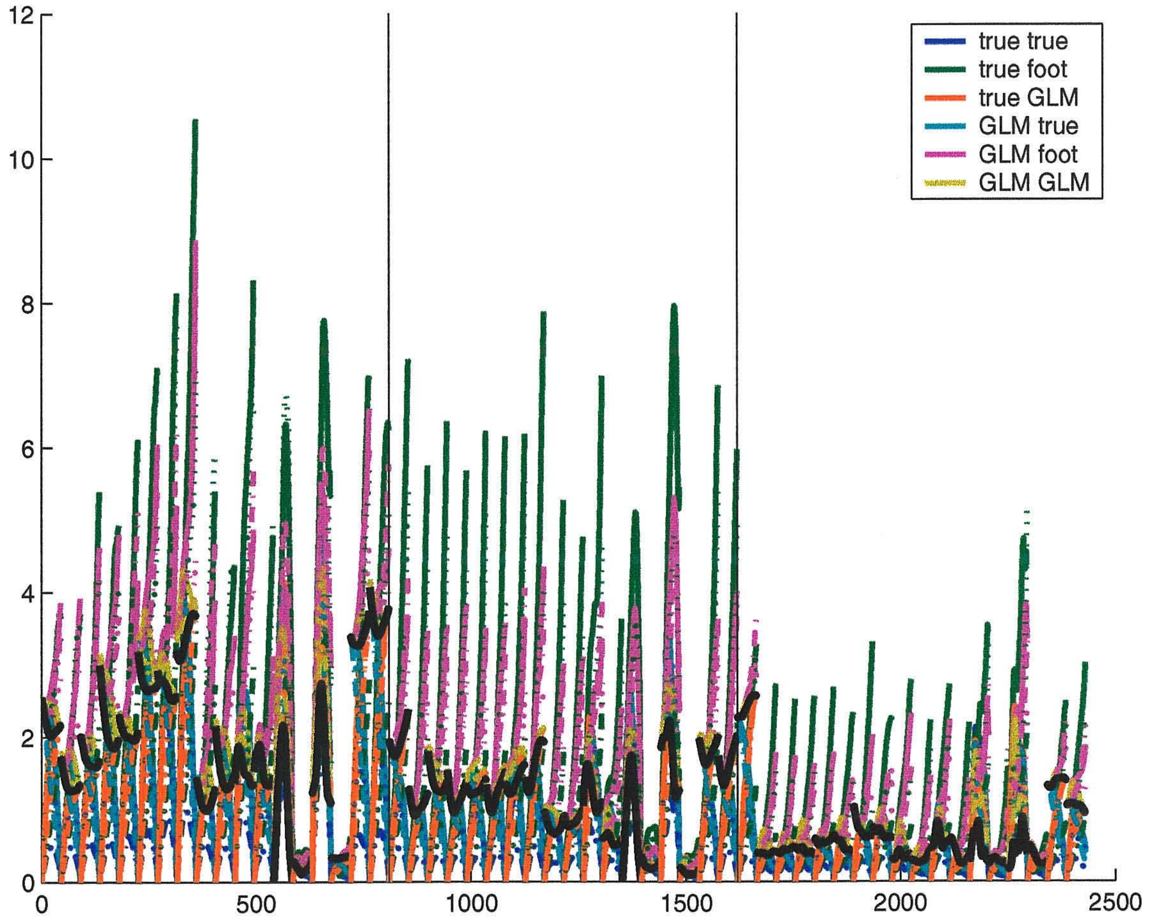


Figure 8.8: RMS error of the SSTC with 8 Cluster MOG probability model for all the motion output coordinates. GLM error in black for reference. Solid lines: equi-weight initial cluster weights, Dotted: GLM initial weight guess, Dashed: True motion weight guess.

perfect. For the model with 1 cluster, the body oscillates wildly. This is due to the fact that the single cluster model is a poor approximation of the dynamics. With the 2 cluster model, the motion is much improved, but the right foot steps awkwardly, and the character appears to be limping. With the 8 cluster model, the stepping of the right foot becomes even more awkward, irrespective of whether the equiGuess or GLMguess is used for initial cluster weights.

Another way to avoid the instability and to improve the motion synthesis is to not use the previous step's final pose as initial condition, but use the pose predicted by the GLM (GLMinit). This guarantees that the system is initialized in a pose where the model is valid, but has the drawback that blending of steps is necessary.

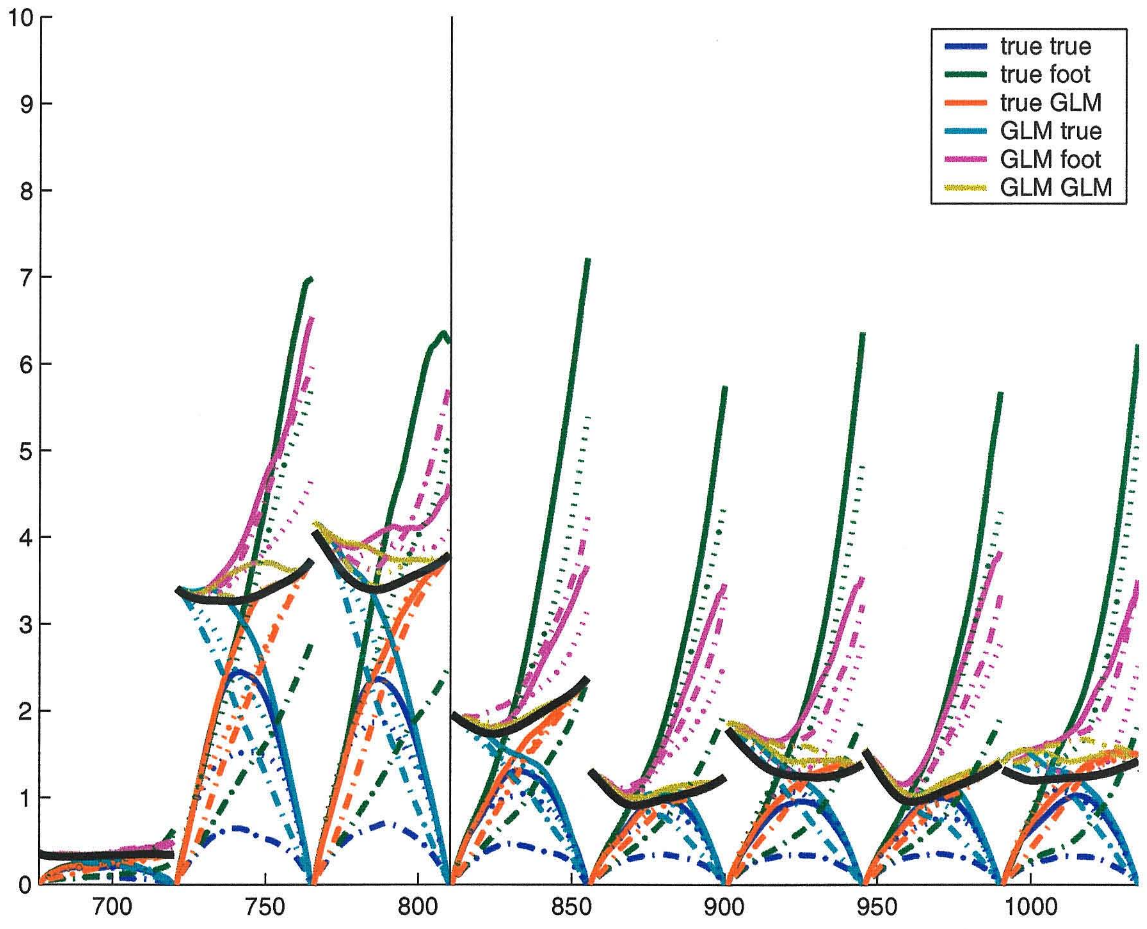


Figure 8.9: Closeup of RMS error of the SSTC with 8 Cluster MOG probability model for some motion output coordinates. Shown are the errors for the x-coordinate of the right foot, and two side head markers, followed by the y-coordinate errors of head, neck, left and right shoulder, and left elbow. Solid lines: equi-weight initial cluster weights, Dotted: GLM initial weight guess, Dashed: True motion weight guess.

With the GLM-based initial pose (GLMinit) and the footEnd constraint, the model with 2 clusters generates motion were the right foot raises to high when the equi-weighted initial cluster weight guess is used, and produces good motion (comparable to that generated by the GLM model) when the GLM-based initial cluster weights are used. The single Gaussian model generates motions where the feet slide at the end of the step. The model with 8 clusters generates motions where the lower leg motion is distorted; the knees seem to buckle and stick out at the end of the step. This happens irrespective of which initial cluster weighting is used, an indication that the model gets stuck in a local minima.

With the GLM-based initial pose and the GLM-based full body final pose constraint

	prevInit	GLMinit
footEnd	always unstable	1C: right foot slides at end of step last left footstep slides inward
		2C equi: right knee, foot raise too much
		2C GLM: good
		8C equi,GLM: distorted leg motion
GLMend	1C: wild oscillations	1C: mild oscillations
	2C equi: right step limping, awkward	2C equi: right foot raises a bit
	2 GLM: similar to above	2 GLM: good
	8 equi: abrupt motion, right step awkward	8 equi: good, seems less fluid
	8 GLM: similar to above	8 GLM: good

Table 8.4: Summary of results of synthesis of new motions with the SSTC models.

(GLMend), the behaviours of the models improve compared to when only the footEnd constraint is used. The foot sliding of the 1 cluster model disappears, although the body oscillates back and forth mildly. With the 2 cluster model and the equi-weight guess, the right foot still raises too high, although not as much as with the footEnd constraint. The 8 cluster model improves remarkedly with the full final pose constraint. With GLMguess weight initialization, the motion is natural, and even with the equiGuess weight initialization there is no distorted leg motion.

In conclusion, the SSTC method needs a MOG conditional probability distribution in order to produce natural motions, but can do so only with certain choices of initial and final pose constraints. The method in its current form falls short of synthesizing realistic motion under the least stringent (but most desirable) conditions, namely, with an arbitrary initial pose constraint from the previous motion and only a few body parts constrained in the final pose.

8.5 Complexity of models

In this section we compare the complexity of the different motion models, with respect to the size of the model (number of parameters), the amount of computation required to build the model, and the amount of computation required to synthesize a new motion. To do so, we define some constants:

- N_F : number of frames in the motion

Model	# of parameters	# of equivalent motion samples
GLM	$3N_M N_F N_L$	5
GQM	$3N_M N_F \frac{N_L^2}{2}$	15
MOG as LLM	$3N_M(N_{PCA}N_F + N_C\frac{(N_{PCA})^2}{2})$	25.4 (2 clusters), 39.9 (4), 68.8 (8)
SSTC (1 cluster)	$(3N_M)^2$	1.2
SSTC with MOG	$N_C\frac{(2(3N_M))^2}{2}$	4.9 (2 clusters), 9.9 (4), 19.6 (8)

Table 8.5: Comparison of model sizes. To compute the number of equivalent motion samples, the walking action was used, with $N_F = 45$, $N_M = 18$, $N_{PCA} = 10$, and $N_L = 4$.

- N_M : number of markers used to represent the body motion
- N_L : dimension of the motion label
- N_C : number of clusters
- N_{PCA} : number of PCA components
- N_{its} : number of iterations
- N_{Ex} : number of examples

Size of model

Table 8.5 shows the order of magnitude of the number of parameters required to represent each of the different types of models. To facilitate comparison, the amount of parameters of each model is compared to the number of coefficients required to represent an entire motion sample ($3N_M N_F$ numbers). The MOG as LLM models are very big because each marker coordinate has to be modeled separately, and for each of those sub-models a covariance matrix of size $N_{PCA}+N_L$ has to be stored. In contrast, the SSTC models are quite compact, since they only represent the behaviour of the system between adjacent time instances.

Computation to train

The amount of computation required to train the different motion models varies by orders of magnitude.

The major cost of training the global polynomial models is the cost of computing a pseudoinverse of dimensions $N_{Ex} \times (1 + N_L)^p$, where p is the polynomial order. Since the pseudo-inverse is a computation of order $O(N^3)$, global polynomial models take on order $O((N_{Ex}(1+N_L)^p)^3)$ computations to train.

For the MOG as LLM models, the training computation involves N_{its} iterations of the EM algorithm with N_{Ex} examples, N_C clusters, and $(N_L + N_{PCA})$ dimensions. Each EM iteration is roughly of order $O(N_{Ex}N_C(N_L + N_{PCA})^2)$.

The SSTC with MOG training also involves EM iterations, but now the problem size is much larger, since there are $N_{Ex} * N_F$ individual examples of pose pairs, and the dimension of the problem is $2 * (3 * N_M)$ (the number of dimensions to define two 3-D poses).

With the amount of data that we usually work with (approximately 6000 frames of data), the global polynomial models take on the order of 1 second to compute, the MOG as LLM take on the order of 1 minute, and the SSTC with MOG takes on the order of 30 minutes.

Computation to use

A similar variance in computational cost during synthesis also occurs. The global polynomial models require a mere matrix-vector multiplication of size $3N_MN_F \times (1 + N_L)^p$. The MOG as LLM require roughly N_C times as many operations, still reasonably efficient. The SSTC method, on the other hand, requires much more computation, since a block tri-diagonal system must be solved with N_F diagonal blocks where each block is of dimension $3N_M \times 3N_M$. This requires on the order of $O(N_F(3N_M)^3)$ operations.

8.6 Conclusions

In this chapter we investigated the perceptual and numerical performance of several models for synthesizing human motion. It was found that a simple Global Linear Model can be used to generate convincing animation. This model is efficient in storage, and in computational cost of use, and provides robust extrapolation when the desired motion label is not within the convex hull of the learning examples. Higher order global polynomial models may increase the quality of motion generation within the convex hull of the examples' label space, but at the risk of producing wildly deviant motions if it is used to extrapolate. The use of Mixture of Gaussians as Local Linear models (MOG as LLM) provides a method of attaining higher fidelity motion than the GLM without compromising the ability to robustly extrapolate, but at the expense of increased number of model parameters. The Stochastic

Space-Time Constraints (SSTC and SSTC with MOG) methods can model a motion with a number of parameters comparable to that of the GLM, but are extremely computationally expensive during motion generation. Furthermore, the quality of the motion generated is comparable to that of the GLM only under restricted constraint conditions, negating any potential advantage in synthesis versatility that they could in principle have (namely, the ability to generate motions continuously, using the end of one motion as the start of the next, without having to compute motions separately and blending them together).

Chapter 9 Conclusions and future work on human motion synthesis

The automatic synthesis of realistic human character animation is a difficult problem, and remains one of the Holy-Grails of computer graphics. In this thesis an attempt was made to move one step closer to that ultimate goal.

The method developed is a data-driven approach, where motion samples from real people are used to derive mathematical models of human motion. The method involves decomposing human motion into elementary actions, such as walking, reaching, looking and throwing, and learning parameterized models of each action. A new motion is synthesized by specifying the desired action, along with parameters which identify the initial state (initial placement of feet or hands, for example), the desired goal of the action (direction of reach, for example), and even some mood or style parameters.

Several types of motion models were developed and experimented with. It was found that a simple linear model mapping the desired motion parameters to a large vector encoding the entire motion can produce good results. This type of model has the properties of being efficient storage and computation-wise, and provides robust synthesis for cases where the desired parameters cause the model to extrapolate beyond the data samples used to derive the model. A new type of model based on weighted local linear models was also developed and shown to improve the fidelity of synthesis, at the cost of increased model complexity and increased computation during synthesis.

Methods of mapping synthesized motions onto a 3-D polygonal character model in real-time were also developed. With these methods an interactive, real-time demo was created that convincingly depicts a character that can walk around with a variable degree of happiness, stop and look around, and step over an obstacle.

Below we list some of the possible topics for future research:

Combining actions

One important topic not dealt with in this thesis is that of combining actions, such as walking and picking something up from the floor at the same time. There are two naive ways of doing so. One option is to learn the combined action as a new motion. While simple, this is impractical since each imaginable combination of actions would have to be learned separately. Alternatively, if the two (or more) actions involve different body parts, the respective body part motions can be blended together. The techniques of Perlin [29] are an example of this approach. However, this approach can quickly breakdown, as in the example of walking while picking something up from the floor, due to the substantial interaction between the two tasks. How can this interaction be taken into account?

Adding autonomous variability

A very powerful way to increase the life-likeness of an animation is to include natural variation in the motion. For example, when people walk, there are subtle variations in the gait, foot placement, and gaze direction. Through the use of goal and mood and style parameters, the techniques developed here can enable such variability. However, another level of learning is required for such variability to be generated automatically: learn the statistics and time evolution of the parameter variability.

Improvement of the SSTC method

Further research should continue on improving the stochastic space-time constraints method, since it is an interesting paradigm to work with. One way to increase the stability of the method may be to alter the probability being maximized to include explicit pose terms of the form $p(x_t)$. Another important aspect is to develop methods of avoiding local maxima, in order to be able to use probability models with a large number of clusters.

One potential future use of the SSTC method is what can be called the 'Reality Filter': If a complex probability model based on extensive human motion observation is built, it can be used to aid traditional key-frame animation techniques. The animator can define sparse body poses throughout time (perhaps with additional, high-level information defining, actions, or moods and styles), and then the Reality Filter can compute the most realistic motion compatible with the specified key-frames.

Another interesting application could be the use of the SSTC method to generate realistic motion for actual robots. Suppose an anthropomorphic robot with a significant number of degrees of freedom is built. The SSTC method can be used to provide constraints on the degrees of freedom and create realistic motion. A preliminary analysis indicates that the SSTC constraints can be incorporated into the Finite Horizon Control formalism, especially as described in [31], where both stability and performance measures can be incorporated.

Modeling in torque space

The motion models developed in this thesis are purely phenomenological. The data happens to be interpretable as human motion, but in principle it could have been anything. Can data-driven methods be used with physical models that take into account the kinematics and dynamics of the human body? One possibility is to not learn the models in 'marker space' (the 3-D coordinates of motion-capture markers on the human body), but rather in 'torque space' (the forces applied to the joints of the body).

Learning from one or two examples

Humans are able to imitate (and recognize) particular motions (and especially particular individual's moods and styles) after viewing only a few examples. How do humans represent/generate motion to make that possible?

Appendix A Implicit Extended Kalman Filter

Appendix 1: Implicit Extended Kalman Filter

A Kalman Filter is what is known in the controls community as an observer of a dynamical system. An important problem in control systems is to estimate the state of a dynamical system, in order to compute an appropriate control feedback signal (the state-space feedback control paradigm). For linear systems with additive Gaussian noise, the Kalman filter is the optimal observer (in the sense that it provides the estimates of the state with the least variance from the true state).

Here we derive the equations of the Kalman filter for the case of interest to us; the case of a discrete time autonomous system. We start with the derivation of the Kalman filter for a linear system, then we derive the equations of the extended Kalman filter which can deal with nonlinear systems. Finally, we derive the filter for nonlinear systems with implicit measurement equations.

A.1 Kalman Filter for discrete time autonomous systems

A discrete time autonomous system can be modeled as

$$x_{k+1} = A_k x_k + G_k w_k \tag{A.1}$$

$$y_k = C_k x_k + v_k \tag{A.2}$$

where $w_k \sim N(0, Q_k)$ is a zero-mean Gaussian random process which takes into account the uncertainty and inaccuracy in modeling the state transition equation of the system, and where $v_k \sim N(0, R_k)$ is also Gaussian zero-mean and is used to model the noise in the measurement process.

Given a sequence of measurements y_k, and an initial (uncertain) state x_0 with covariance $P_{0|0}$, we would like to estimate the state of the system throughout time, x_k. Moreover, we would like to derive a recursive estimator, where x_k is estimated given the estimate at the previous time step.

Let $\hat{x}_{k+1|k}$ denote the estimate of x at time $k+1$ given all the measurements up to and including time k. Let $\hat{x}_{k+1|k+1}$ denote the estimate of x at time $k+1$ when the measurement at time $k+1$ is also used. We have then that

$$\hat{x}_{k+1|k+1} = E(x_{k+1}|z_{k+1}, z_k, ..., z_0) = E(x_{k+1}|Z_{k+1}) \tag{A.3}$$

where Z_{k+1} denotes the sequence of observations $z_{k+1}, z_k, ..., z_0$ likewise

$$\hat{x}_{k+1|k} = E(x_{k+1}|z_k, z_{k-1}, ..., z_0) = E(x_{k+1}|Z_k) \tag{A.4}$$

Let us also define $\tilde{z}_{k+1} = z_{k+1} - E(z_{k+1}|z_k, ..., z_0)$. The sequence $\{\tilde{z}_k\}$ is called an innovation sequence and has the following important properties:

- The sequence is orthogonal, i.e., $E(\tilde{z}_k\tilde{z}_l') = 0$ if $k \neq l$. This is shown by application of the Projection Theorem, which states that for jointly distributed random variables X and Y, the error of $X - E(X|Y)$ is orthogonal to Y, i.e., $E((X - E(X|Y))Y') = 0$.

- Since the sequence $\{\tilde{z}_k\}$ is derived from $\{z_k\}$ by casual linear operations, the estimate of any jointly distributed random variable given $\{\tilde{z}_k\}$ is equal to that obtained from $\{z_k\}$. In our particular case, we have $E(x_{k+1}|Z_{k+1}) = E(x_{k+1}|\tilde{Z}_{k+1})$.

- Finally, due to the orthogonality of the $\{\tilde{z}_k\}$, a conditional expectation given the sequence can be decomposed to a sum of conditional expectations given each individual innovation:

$$E(x_{k+1}|\tilde{z}_{k+1}) = E(x_{k+1}|\tilde{z}_{k+1} + E(x_{k+1}|\tilde{z}_{k+1} + \cdots + E(x_{k+1}|\tilde{z}_0) - (k-1)E(x_{k+1}). \tag{A.5}$$

In our particular case, in order to derive a recursive equation for $\hat{x}_{k+1|k+1}$, it will be convenient to rewrite the conditional expectation as the sum of two terms:

$$E(x_{k+1}|\tilde{z}_{k+1}) = E(x_{k+1}|\tilde{z}_{k+1}) + E(x_{k+1}|\tilde{Z}_k) - E(x_{k+1}). \tag{A.6}$$

Thus we have $\hat{x}_{k+1|k+1} = E(x_{k+1}|\tilde{z}_{k+1}) + E(x_{k+1}|\tilde{Z}_k) - E(x_{k+1})$, and we proceed by deriving the two conditional expectations. To do so, it is also convenient to define $\tilde{x}_{k+1} =$

$x_{k+1} - \hat{x}_{k+1|k}$ so that consequently $\tilde{z}_{k+1} = C_{k+1}\tilde{x}_{k+1} + v_{k+1}$.

The second term, $E(x_{k+1}|\tilde{Z}_k)$, is quickly recognized as $\hat{x}_{k+1|k}$ which we will keep as part of our recursive formula for $\hat{x}_{k+1|k+1}$.

The first term, $E(x_{k+1}|\tilde{z}_{k+1})$, can be calculated as follows:

$$E(x_{k+1}|\tilde{z}_{k+1}) = E(x_{k+1}) + cov(x_{k+1}, \tilde{z}_{k+1})cov(\tilde{z}_{k+1}, \tilde{z}_{k+1})^{-1}\tilde{z}_{k+1}, \tag{A.7}$$

which is simply the equation for the conditional expectation of jointly distributed Gaussian variables.

The first covariance term can be further expanded as:

$$cov(x_{k+1}, \tilde{z}_{k+1}) = cov(\tilde{x}_{k+1} + x^h at_{k+1|k} - E(x_{k+1}, C_{k+1}\tilde{x}_{k+1} + v_{k+1}). \tag{A.8}$$

Since $\hat{x}_{k+1|k}$ is orthogonal to $\tilde{x}_{k+1}$ and v_{k+1} is orthogonal to both $x^h at_{k+1|k}$ and $\tilde{x}_{k+1}$, we have

$$cov(x_{k+1}, \tilde{z}_{k+1}) = E(\tilde{x}_{k+1}(C_{k+1}\tilde{x}_{k+1})') = \Sigma_{k+1|k}C'_{k+1}, \tag{A.9}$$

where $\Sigma_{k+1|k}$ is the covariance of the estimate $\hat{x}_{k+1|k}$ and is equal to $E(\tilde{x}_{k+1}\tilde{x}'_{k+1})$.

The second covariance term is evaluated as:

$$\begin{aligned} cov(\tilde{z}_{k+1}, \tilde{z}_{k+1}) &= cov(C_{k+1}\tilde{x}_{k+1} + v_{k+1}, C_{k+1}\tilde{x}_{k+1} + v_{k+1}) && \text{(A.10)} \\ &= E(C_{k+1}\tilde{x}_{k+1}(C_{k+1}\tilde{x}_{k+1})') + E(v_{k+1}v'_{k+1}) && \text{(A.11)} \\ &= C_{k+1}\Sigma_{k+1|k}C'_{k+1} + R_{k+1}. && \text{(A.12)} \end{aligned}$$

where the second last equality holds since $\tilde{x}_{k+1}$ and v_{k+1} are both zero mean and orthogonal to each other.

Thus our first recursive formula is:

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + \Sigma_{k+1|k}C'_{k+1}(C_{k+1}\Sigma_{k+1|k}C'_{k+1} + R_{k+1})^{-1}(z_{k+1} - C_{k+1}\hat{x}_{k+1|k}) \tag{A.13}$$

This formula is called the 'measurement update', because it updates the best estimate of x_{k+1} given information from time $k(\hat{x}_{k+1|k})$, by using the latest measurement, z_{k+1}.

A recursive formula for $\hat{x}_{k+1|k}$ can also be computed as

$$\begin{aligned}\hat{x}_{k+1|k} &= E(x_{k+1}|\hat{x}_{k|k}) &\quad\text{(A.14)}\\ &= E(F_k\hat{x}_{k|k} + G_k w_k) &\quad\text{(A.15)}\\ &= F_k\hat{x}_{k|k}. &\quad\text{(A.16)}\end{aligned}$$

This is called the 'time update' since the best estimate of x_k at time k is evolved to time $k+1$ via the state evolution equation of the system.

To make the derivation complete, we also need to derive recursive formulas for $\Sigma_{k+1|k+1}$ and $\Sigma_{k+1|k}$.

$$\begin{aligned}\Sigma_{k+1|k} &= cov(x_{k+1} - \hat{x}_{k+1|k}) &\quad\text{(A.17)}\\ &= cov(F_k + G_k w_k - F_k\hat{x}_{k|k}) &\quad\text{(A.18)}\\ &= cov(F_k(x_k - \hat{x}_{k|k} + G_k w_k) &\quad\text{(A.19)}\\ &= F_k E((x_k - \hat{x}_{k|k})(x_k - \hat{x}_{k|k})')F_k' + G_k E(w_k w_k')G_k' &\quad\text{(A.20)}\\ &= F_k \Sigma_{k|k} F_k' + G_k Q_k G_k' &\quad\text{(A.21)}\\ \Sigma_{k+1|k+1} &= cov(x_{k+1} - \hat{x}_{k+1|k+1}) &\quad\text{(A.22)}\end{aligned}$$

Using the recursion relation for $\hat{x}_{k+1|k+1}$ we note that

$$x_{k+1} - \hat{x}_{k+1|k+1} + \Sigma_{k+1|k}C_{k+1}'(C_{k+1}\Sigma_{k+1|k}C_{k+1}' + R_{k+1})^{-1}(z_{k+1} - C_{k+1}\hat{x}_{k+1|k}) = x_{k+1} - \hat{x}_{k+1|k} \quad\text{(A.23)}$$

Since $\tilde{z}_{k+1}$ ($= z_{k+1} - C_{k+1}\hat{x}_{k+1|k}$) is orthogonal to $(x_{k+1} - \hat{x}_{k+1|k+1})$, the two sides of the equation can be computed straightforwardly as:

$$\Sigma_{k+1|k+1} + \Sigma_{k+1|k}C_{k+1}'(C_{k+1}\Sigma_{k+1|k}C_{k+1}' + R_{k+1})^{-1}C_{k+1}\Sigma_{k+1|k} = \Sigma_{k+1|k} \quad\text{(A.24)}$$

so

$$\Sigma_{k+1|k+1} = \Sigma_{k+1|k} - \Sigma_{k+1|k}C'_{k+1}(C_{k+1}\Sigma_{k+1|k}C'_{k+1} + R_{k+1})^{-1}C_{k+1}\Sigma_{k+1|k} \tag{A.25}$$

To summarize, given a discrete autonomous system $x_{k+1} = A_k x_k + G_k w_k$, $z_k = C_k x_k + v_k$ where $w_k \sim N(0, Q_k), v_k \sim N(0, R_k)$, and given an initial estimate of x, $\hat{x}_{0|0}$ with variance $\Sigma_{0|0}$, as new measurements of the system are obtained, $z_1, z_2, ...z_k$, estimates of the system state can be calculated recursively as follows:

- time update:

$$\hat{x}_{k+1|k} = A_k \hat{x}_{k|k} \tag{A.26}$$

$$\Sigma_{k+1|k} = A_k \Sigma_{k|k} A'_k + G_k Q_k G_k \tag{A.27}$$

- measurement update:

$$\begin{aligned}
\hat{x}_{k+1|k+1} &= \hat{x}_{k+1|k} + \Sigma_{k+1|k}C'_{k+1}(C_{k+1}\Sigma_{k+1|k}C'_{k+1} + R_{k+1})^{-1}(z_{k+1} - C_{k+1}\hat{x}_{k+1|k}) \\
\Sigma_{k+1|k+1} &= \Sigma_{k+1|k} - \Sigma_{k+1|k}C'_{k+1}(C_{k+1}\Sigma_{k+1|k}C'_{k+1} + R_{k+1})^{-1}C_{k+1}\Sigma_{k+1|k}
\end{aligned}$$

A.2 Extended Kalman Filter

The system under consideration in the previous section was a linear time varying system (because the matrices A_k, G_k, and C_k could depend on the time instance, k). What if the state evolution were a nonlinear function of the state x_k, or the measurement equation were a nonlinear function of the state, as in the system below?

$$x_{k+1} = f(x_k) + G_k w_k \tag{A.28}$$

$$z_k = g(x_k) + v_k \tag{A.29}$$

The Kalman Filter of the previous section can be modified into an 'Extended Kalman Filter' capable of estimating the state of a nonlinear system. This is done by linearizing the system at the appropriate operating point. For the measurement equation, the approximation is:

$$g(x_k) = g(\hat{x}_{k|k-1}) + C_k(x_k - \hat{x}_{k|k-1}) + \dots \tag{A.30}$$

where $C_k = \frac{\partial g(x_k)}{\partial x}$ evaluated at $x = \hat{x}_{k|k-1}$. With this approximation the residual is approximated as:

$$\tilde{z}_k = z_k - \tilde{z}_{k|k+1} = g(x_k) + v_k - g(\hat{x}_{k|k-1}) = C_k(x_k - \hat{x}_{k|k-1}) + v_k \tag{A.31}$$

Thus, the expression for the residual is exactly the same as for the linear system case. Likewise, $f(x_k)$ is approximated as:

$$f(x_k) = f(\hat{x}_{k|k}) + F_k(x_k - \hat{x}_{k|k}) + \dots \tag{A.32}$$

where $F_k = \frac{\partial f(x_k)}{\partial x}$ evaluated at $x = \hat{x}_{k|k}$. With this definition, $\tilde{x}_{k+1}$ is calculated as:

$$\tilde{x}_{k+1} = x_{k+1} - \hat{x}_{k+1|k} = f(x_k) + G_k w_k - f(\hat{x}_{k|k}) = F_k(x_k - \hat{x}_{k|k}) + G_k w_k \tag{A.33}$$

Again, this is exactly the same expression as in the linear system case, so it is clear that the same expressions for the updates of the state and covariance matrix can be derived. Note, however, that in applying these equations another approximation is made, namely that the probability distribution of x_k and z_k is Gaussian. For the linear case, given that the noise terms w_k and v_k are Gaussian, this is valid, but for a general nonlinear system this is not the case. Thus whereas for a linear system the Kalman Filter was the optimal estimator, it is not true that the Extended Kalman filter is also optimal, but rather it is suboptimal. Typically, however, it performs nearly optimally, as long as the initial estimate of the state is accurate enough (unlike the linear case, the EKF has a bounded region of stability and convergence).

A.3 Implicit Extended Kalman Filter

Finally we consider the case of a nonlinear system where the measurement equation is expressed as an implicit constraint between a measurable quantity y_k and the system state x_k, where it may not be convenient or possible to express y_k as a function of x_k:

$$\begin{aligned} x_{k+1} &= f(x_k) + G_k w_k & \text{(A.34)} \\ h(x_k, y_k) &= 0 & \text{(A.35)} \end{aligned}$$

Due to measurement noise, one cannot measure y_k exactly but as $z_k = y_k + v_k$. We can now express $h(x_k, y_k)$ in terms of the Taylor series expansion of h at a convenient operating point:

$$h(x_k, y_k) = h(\hat{x}_{k|k-1}, z_k) + C_k(x_k - \hat{x}_{k|k-1}) + D_k(y_k - z_k) + \dots \qquad \text{(A.36)}$$

where $C_k = \frac{\partial h(x,y)}{\partial x}$ evaluated at $x = \hat{x}_{k|k-1}, y = z_k$ and $D_k = \frac{\partial h(x,y)}{\partial y}$ evaluated at $x = \hat{x}_{k|k-1}, y = z_k$. Since by definition $h(x_k, y_k) = 0$, we have that

$$h(\hat{x}_{k|k-1}, z_k) = -C_k(x_k - \hat{x}_{k|k-1}) - D_k v_k \qquad \text{(A.37)}$$

This is none other than (the negative of) the innovation vector $\tilde{z}_k$, and so we can once again compute the time and measurement updates for $\hat{x}_k$ and $\hat{P}_k$, just as in the EKF case.

Appendix B Motion Capture System

This appendix describes the optical motion capture system designed for the purpose of acquiring large datasets of 3-D human motion. The basic idea of the technique is to use multiple cameras to observe the motion of an actor who is wearing a special 'lightbulb suit'. Once the motion of 18 lightbulbs on the body have been detected by the four cameras, triangulation of the data from the four viewpoints is done to obtain the 3-D coordinates of the lightbulbs throughout time. The system we designed is able to detect the motion with an accuracy of 1 mm in a 2 m wide by 3 m long floor space, at a rate of 60 frames per second.

Here is a brief summary of the appendix:

Hardware:

- Vision system: a description of the equipment needed to acquire data
- Body suit: design and placement of the lightbulbs on the body
- Calibration tools: the devices needed to determine the camera lens parameters, as well as the position of the cameras with respect to a global reference frame

Software

- Real-time dot detection: the algorithm used to accurately and rapidly detect the positions of the dots in each camera
- Calibration:
  - Intrinsic calibration: the method for finding the camera lens parameters
  - Extrinsic calibration: the method for finding the position and orientation of the cameras in space
- 3-D reconstruction: the algorithm for triangulating 2-D data from 4 cameras to produce 3-D data
  - Tracking of 2-D dot segments in time

  - Identification of potential matching segments
  - Grouping of 2-D segments into 3-D candidates

- Labeling: Assigning each 3-D dot its respective body part

  - Weave matched 2-D segments into 3-D threads
  - Connect threads into labeled chords

B.1 Hardware

The hardware needed to reconstruct the motion of a human in 3-D is: a vision system (computers and cameras), a special body suit with easy-to-identify markers, and calibration tools to determine physical and geometrical camera parameters necessary for obtaining accurate 3-D data.

B.1.1 Vision system

Camera choice

3-D marker positions are obtained by triangulating the positions as observed from different camera viewpoints. Thus, each marker must be seen by at least two cameras. Due to inter-body occlusions, more than two cameras are needed in order to maximize the likelihood that each marker is always seen by at least two cameras. Currently, our system uses four cameras, arranged on a 180 degree arc with respect to the center of the motion capture area. To be able to capture good data, the actor has to move cooperatively; he cannot turn more than 40 degrees beyond a central orientation, or markers will be hidden from too many cameras. In previous test configurations, it was determined that to be able to acquire motion with a full 360 degree freedom of body orientation four cameras are not sufficient, and at least two additional cameras would be required.

One requirement for accurate 3-D reconstruction is that the image acquisition process of the cameras be synchronized (or at least that the time delay between different camera acquisitions be known accurately). We have used the Sony XC-73 CCD camera, which is well suited to our application because it provides the user with a great deal of control over camera function. To optimize the accuracy of 3-D reconstruction, we adjusted the cameras in the following ways:

- One camera was set as 'master', and the other three were set as 'slaves', so that they received the horizontal and vertical sync signals from the master camera. In this way, the cameras were perfectly synchronized, generating video signals synchronized to within a fraction of a microsecond.

- The electronic shutter speed of each camera was set to 1/2000 of a second, to reduce the effect of image distortion caused by motion blur.

- The gain of each camera was $\overset{c}{a}$djusted so that when in use capturing motion, the lightbulb markers appeared as a reasonable brightness against the dark background, not too dark and not too bright. In this way the markers could be detected reliable irrespective of where the actor stood in the workspace.

- The cameras were switched to operate in 'field' mode. In the more common 'frame' mode, the camera would generate 30 frames per second, but each frame is 'interlaced' as an even and odd field. Every 1/60th of a second the camera would alternately transmit either the even lines or the odd lines of the image. In field mode, the even field only is transmitted every 1/60th of a second. By using the field mode, we were able to acquire 60 frames per second without having to worry about correcting for pixel shifts between even and odd fields. More importantly, in field mode the camera uses adjacent even and odd CCD pixels as one equivalent 'even' pixel. This increases camera sensitivity, and greatly reduces sampling alias effects by effectively increasing the pixel fill-factor on the CCD.

The cameras were equipped with 6mm lenses, which provided for a sufficient field of view. One camera, however, used a 4.2mm lens because it needed to be placed nearer to the motion capture workspace due to room geometry constraints.

Computer choice

Although most of the data processing for 3-D reconstruction is done off-line, the initial step of detecting the body markers (lightbulbs) in each camera view in each frame must be done in real-time. If it were not real-time, we would have to store all the images, instead of just 18 screen coordinates per frame, and would quickly run out of disk space. It became apparent that we would need to use one computer per camera, and that the computer should be fast! We found that 233Mhz MMX machines (with the code hand-optimized with MMX instructions) were fast enough to process images of size 320x240 pixels in real-time, at a rate of 60 frames per second.

Frame-grabber choice

Each computer was equipped with an Imagenation PXC-200 frame-grabber in order to digitize the video stream in real-time. The PXC is ideal for our application because it is relatively inexpensive, yet reliable and producing good quality images. Two particularly useful features of the PXC are:

- 1) it enables acquisition to be triggered by an external signal, so that we could use a pushbutton trigger to activate all computers at the same exact time and thus establish initial correspondence between frames from different cameras on different computers.
- 2) it also timestamps frames, so that we are able to detect when a frame has been missed (1/1000 probability of occurrence at 60Hz acquisition of 320x240 images).

B.1.2 Body suit

Design of the body suit that the actor must wear involved determining what type of markers to use to detect the motion of different body parts, and figuring out how to attach those markers to the body reliably.

Choice of marker

After several experiments (with fluorescent paint and UV lighting, 3M reflective paint and flood lights, and lightbulbs) we found that the simplest way to generate images with easily-identifiable body markers was to attach lightbulbs to the body, and perform the desired motions in near darkness.

The lightbulb itself had to satisfy several criteria:

- Low voltage and low current, for both safety reasons and to conserve battery life (so that the brightness of the lightbulbs is constant throughout the motion capture session, which typically last 2 hours).

- Should have a spherical form factor, so that its appearance in the image is the same from any viewpoint. This avoids viewpoint-based biases in the sub-pixel accuracy estimation of the marker screen coordinates.
- Not too big, and not too small, so that it is not cumbersome to use, and yet has a usable appearance in the acquired images.

Such a lightbulb was found, the ML-1483 . At 6V, it consumes 4mA. In use, the 18 markers are connected in parallel to a 6V lantern battery, which results in a steady brightness of an acceptable level for several hours (6 or more).

The bulbs were 1 cm wide, which is an acceptable width for imaging. However, one residual problem was that the bulb filament was too small, causing the image appearance to be just one pixel wide and sensitive to aliasing and viewpoint changes. To avoid these problems, the bulb surface was coated with translucent paper-towel tissue. This transformed the bulbs into translucent spheres. This modification made the image appearance become a nice round spot with a width of approximately 3 pixels, and the dot detection algorithm (described later) is able to detect the screen coordinates with an accuracy of 1/10th of a pixel.

Marker placement on the body

The placement of the body markers on the actor's body is subject to several practical and theoretical constraints. Theoretically, the more markers there are, the more accurately (in the least mean square sense) the body motion can be detected. However, increasing the number of markers observed makes the 3-D triangulation more difficult to execute, because it becomes more difficult to correctly determine the intra-camera correspondence of the markers (i.e., which dot in camera 1 corresponds to a dot in camera 2). Practically, the number of markers that can be placed on the body is also limited by the ease with which markers can be attached to the body, and the finite amount of power available to light the lightbulbs.

As a working compromise, our system uses 18 markers on the body. There is a marker placed near each shoulder, elbow, wrist, as well as near each hip, knee, ankle, and foot tip (14 markers in total). There is also a marker placed at the base of the neck, and markers on the front and sides of the head. In order to maximize the chance of observation of the makers on the side of the head, a metal piece of head-gear was constructed to enable the side markers to protrude 12 cm away from the head.

The markers were attached using Velcro, to spots on the body which are as rigidly connected to the bone structure as possible. Also, markers were positioned forward-facing, so that they are seen by as many cameras as possible.

B.1.3 Calibration tools

Besides the vision system and the body suit, the third component necessary for data acquisition is the calibration tools. These tools were used to estimate intrinsic camera parameters (focal length, optical center, and radial distortion), as well as to estimate extrinsic parameters (the 3-D position and orientation of the cameras in space).

To estimate the intrinsic camera parameters, a large black-and-white checkerboard pattern of 12 by 16 squares (4.25 cm by 4.32 cm) is used. The intrinsic calibration software (described below) uses the precise knowledge of the dimensions of the checkerboard to determine the camera focal length, optical center, and radial distortion (all defined below) given three acquired images of the checkerboard.

To estimate the extrinsic parameters (the 3-D position and orientation of the cameras in space), a 1.5m rod with lightbulbs on the end, as well as a 1.5m by 1m right-angle triangle with lightbulbs on the vertices, are used.

By moving the lightbulb rod around the entire workspace (always keep it vertical so as to make it easy to match the top and bottom lightbulbs in the different camera views), a large set of 2-D observations of 3-D points in space is collected. Using a clever motion estimation algorithm described below, it's possible to use this data to determine the 3-D positions and orientations of all the cameras with respect to, say, the first camera.

The lighted triangle is then placed on the floor, to define a floor reference frame (it is more convenient and natural to work with the 3-D data in a floor reference frame rather than one of the camera reference frames). Reconstructing the 3-D position of the 3 vertices of the triangle defines an X an Y axis, and hence a Z axis for the floor reference frame.

B.2 Software

There are four major software components necessary for acquiring motion capture data. One, an accurate calibration of the vision system must be done. Two, the body markers must be detected in real-time. Three, the 2-D data from the four cameras must be merged to create 3-D data. And four, the 3-D data must be labeled frame-by-frame with the appropriate body part. In this section we describe the algorithms used for each of these components.

B.2.1 Calibration

As mentioned briefly in section B.1.3 describing the calibration hardware, there are two calibrations that need to be done: the intrinsic optical parameter calibration for each camera, and the extrinsic geometrical calibration defining each cameras position and orientation in space.

Intrinsic parameter calibration

The intrinsic parameters of a camera define the relationship between a point in space $P = (X, Y, Z)$ (in the camera reference frame), and its observed image coordinates $p = (x, y)$.

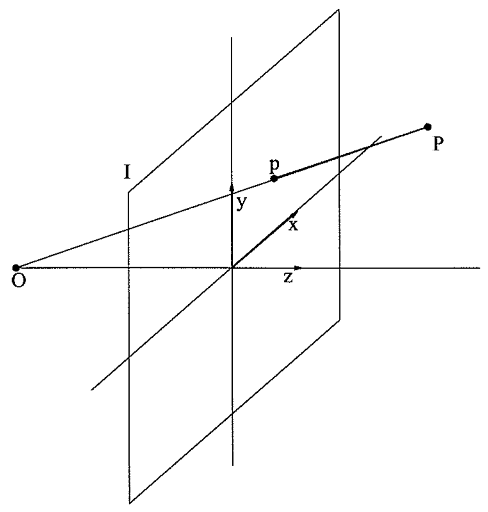


Figure B.1: A simple camera model. Optical axis z is perpendicular to image plane I. Camera focal point is at point C along optical axis, and a point P in 3-D space is projected along the ray from C to point p in image plane.

Figure B.1 shows a simple model of a camera. O is the camera focal point, or origin, and

I is the camera image plane (oriented perpendicular to the camera Z-axis). A 3-D point, P, and its projection on the image, p, lie on the same 3-D ray originating from the camera origin, O, so that if the image plane I is a distance 1 from the camera origin we have:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{B.1}$$

which is the equation for perspective projection, where $[xy1]'$ are the coordinates of the projection on the image plane, and $[XYZ]'$ are the coordinates of the point in 3-D.

$[xy1]'$ are also known as the homogeneous coordinates of the screen projection p, and is a useful representation of 3-D projections that will be used in the 3-D reconstruction algorithm.

Unfortunately, real cameras do not provide homogeneous coordinates directly, but rather provide screen coordinates (u, v), which must be converted to homogeneous coordinates using the intrinsic parameters.

The first thing to note is that in the homogeneous coordinates $[xy1]'$, x and y are unitless (or, if 1 is taken to be 1 meter, x and y are in meters). In a real camera, (u, v) are measured in units of pixels. Thus a scaling factor, f, (which can be thought of as a camera focal length) is used to convert homogeneous coordinates to screen coordinates; $[uv] = f[xy]$. A further complication that occurs in real cameras, however, is that CCD pixels may not be placed on a perfectly square grid, but may have different scales in the u and v directions. Thus the focal length parameter f must be estimated independently for both directions; $[uv]' = [f_u x f_v y]'$.

A second thing to note is that whereas homogeneous coordinates x and y are measured from point c (called the optical center) in the image plane (the intersection of the camera optical axis (z-axis) with the image plane), the origin of the screen coordinates is the top left pixel of the screen. Thus an offset must be added to obtain the screen coordinates of a point. If (c_u, c_v) represents the screen coordinates of the optical center, we have $[uv]' = [f_u x f_v y]' + [c_u c_v]'$.

Finally, one must take into account the non-ideal nature of camera lenses. A typical camera lens introduces several different types of distortions. One very noticeable distortion is called radial distortion, where the screen coordinate of a point is not simply just a scaled version of homogeneous coordinate, but the scale factor depends on how far away the point is from the optical center. Radial distortion can be modeled by estimating a radial distortion coefficient, k, and computing the effective homogeneous coordinates $[\tilde{x}, \tilde{y}]'$ given the homogeneous coordinates of a point as:

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = (1 + k * (x^2 + y^2)) \begin{bmatrix} x \\ y \end{bmatrix} \tag{B.2}$$

Thus to accurately predict the observed screen coordinates (u,v) of a 3-D point (X,Y,Z), the intrinsic parameters f=[fu fv] (focal length), c=[cu cv] (optical center), and k (radial distortion) must be determined. Then the screen coordinates are computed as

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{B.3}$$

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = (1 + k(x^2 + y^2)) \begin{bmatrix} x \\ y \end{bmatrix} \tag{B.4}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = f \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} + c \tag{B.5}$$

Intrinsic parameter calibration algorithm

The intrinsic parameter calibration algorithm used is one perfected by Jean-Yves Bouguet-Cauchy-Gutenberg. Multiple images (typically three) with different views of the calibration checkerboard pattern are captured. The program first detects the corners of the checkerboard squares in all the images. It then runs a non-linear least-squares optimization routine to estimate the intrinsic parameters of the camera as well as the orientation of the checkerboard in space with respect to the camera. The basic principle of the algorithm is that since the dimensions of the calibration pattern are known accurately, if its position in space is known, then given an initial guess of the intrinsic parameters it is possible to predict the screen coordinates of the checkerboard square corners. These predicted screen coordinates are compared to the actual detected coordinates to generate an error vector which is minimized.

Extrinsic parameter calibration

Given the intrinsic parameter calibration for each camera, it is also necessary to accurately estimate the positions and orientations of the cameras in space. This will allow for the accurate triangulation of the 3-D position of a marker if it is detected by more than one camera. This is done in two steps. First, using data collected by moving the light-bulb rod throughout the workspace, the 3-D positions and orientations of all the cameras with respect to the first camera is determined. Second, the light-bulb triangle is used to define a floor reference, which is a much more convenient reference frame in which to view and work with motion capture data. Before describing the algorithms used in the two steps, we define the mathematical representation for camera pose.

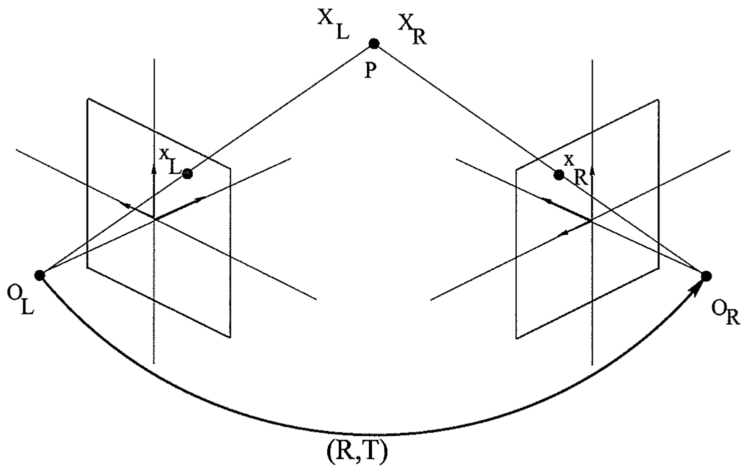


Figure B.2: A point P viewed from two cameras. A rigid transformation (R, T) can be used to transform the coordinates of the point in one view to that of the other.

In order to compute screen coordinates from 3-D coordinates, or vice-versa, it is necessary to be able to represent 3-D coordinates in the reference frame of each camera. In figure B.2, X_L and X_R are the coordinates of a 3-D point expressed in the reference frames of camera L and camera R, respectively. These coordinates are related by the rigid transformation (R, T), where R is a rotation matrix and T is a translation vector, which can be thought of as representing the pose of camera R with respect to camera L. The relationship is:

$$X_R = RX_L + T \tag{B.6}$$

Thus if we have a set 1..N of cameras, the relative pose of the cameras is defined by a set of rigid transformations $(R_i, T_i), i = 2..N$ defining the pose of cameras 2..N with respect to camera 1. By composition of rigid transformations, one can transform the coordinates of a point expressed in camera 3 reference frame into the reference frame of camera 4 by first applying the inverse transformation from camera 3 to camera 1 coordinates, and then the transformation from camera 1 to camera 4 coordinates.

It is worth pointing out that the set-up of figure B.2 has two interpretations. O_L and O_R could be viewed as two cameras of a stereo-pair system, in which case (R,T) describes the relative pose of O_R w.r.t. O_L. O_L and O_R could also be viewed as the same camera (observing a static scene) at two time instances. In this case, (R,T) describes the relative motion of the camera from one time instance to the other. In the sequel, the terms 'pose estimation' and 'motion estimation' will be interchanged indiscriminately, due to this equivalence of interpretations.

Obtaining inter-camera position and orientation

The method for estimating the relative pose between the cameras can be summarized as follows:

- Use the intrinsic calibrations to compute homogeneous coordinates for the observed data of the light-bulb rod. Use only the data where the rod was observed by all four cameras. (Typically there may be a few frames where the dots are not observed in one of the cameras).

- Make initial pose estimates for each camera with respect to camera 1 using only the data from those two cameras. This is done using the well-known motion estimation algorithms of Longuet-Higgins [26] and Horn [20].

- Estimate a consistent set of scales for the poses of the cameras. The initial pose algorithms of the previous steps can estimate pose up to an arbitrary scale factor. Assuming the scale between camera 1 and 2 to be the unit scale, we estimate what the other scales must be so that the reconstructed structure (3-D positions of the light-bulb rod through time) from any two views is of the same size.

- Using the initial pose estimates with consistent scale, refine the poses. This is done with an iterative algorithm developed by Jean-Yves Bouguet-Cauchy-Guttenberg which alternately estimates the optimal 3-D structure using the current estimate of all the poses, and then re-estimates the pose of each camera given the current estimate of the structure. After convergence, the accuracy of the pose estimates are typically improved by a factor of 2 to 5.
- Find the global scale factor using the known length of the light-bulb rod.

This extrinsic calibration algorithm was found to work quite well. Given the detected screen coordinates of a light-bulb from any two camera views, the projective rays intersect in space to within 1.5 mm, i.e., 3-D coordinates can be reconstructed with 1.5mm accuracy. This is the 'local' figure of merit. One can also estimate a 'global' accuracy by calculating the standard deviation of the estimated length of the rod (by finding the 3-D positions of the light-bulbs on either end). The rod length is estimated with a standard deviation of 1.2 cm (1% relative error), ten times as large as the local accuracy. The reason for this is that, due to inaccuracies in the intrinsic parameters and camera lens model (as well as other unknown sources of error, such as the frame digitization), the 3-D reconstructible space is warped; the calculated length of the rod varies as it is moved throughout the workspace.

The next sections describe the individual steps of the extrinsic calibration in detail.

Initial pose estimates

Given the homogeneous coordinates of the observed light-bulbs, we compute an initial estimate of each camera pose with respect to camera 1 in two steps. First the explicit method of Longuett-Higgins computes a coarse estimate, then the estimate is refined with Horn's minimization algorithm.

Longuett-Higgins' Algorithm

Suppose x_L and x_R are the respective projections of the 3-D coordinates X_L and X_R of a point P expressed in two reference frames, L and R. The coordinates are related by $X_R = RX_L + T$, and we wish to determine the appropriate rigid transformation (R, T).

Note that the 3-D vectors O_LP, O_LO_R, and O_RP are coplanar. Longuett-Higgins makes use of this coplanarity by defining a trilinearity constraint in the L reference frame. The trilinearity constraint is a scalar constraint that states that for any three coplanar vectors a, b and c, $a \cdot (b \times c) = 0$ (i.e., any one vector is perpendicular to a vector that is perpendicular to the other two). In the L reference frame, the three vectors can be expressed as follows:

$$O_LP = X_L \tag{B.7}$$

$$O_LO_R = T \tag{B.8}$$

$$O_RP = R'X_R \tag{B.9}$$

And the trilinearity constraint becomes

$$(R'X_R)' \cdot (T \times X_L) = 0 \tag{B.10}$$

However, we do not know the 3-D coordinates X_R and X_L, but only the observed homogeneous (projective) coordinates x_R and x_L. By definition, though, x_R and x_L lie on the same projective ray as X_R and X_L, and so the trilinearity constraint also holds as:

$$(R'x_R)' \cdot (T \times x_L) = 0 \tag{B.11}$$

This constraint can be written in a more convenient form by using the 'wedge' operator, '$\wedge$', to represent the cross product of two vectors, $a \times b$, as the product of a matrix and a vector, $a^\wedge b$. The wedge operator is defined as

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^\wedge = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{B.12}$$

The trilinearity constraint now becomes

$$(x_R)'R(T^\wedge x_L) = 0 \tag{B.13}$$

If we define $Q = RT^\wedge$, then we obtain a constraint which is linear in the unkown coefficients of Q:

$$x'_R Q x_L = 0 \tag{B.14}$$

Now if we divide both sides of the constraint by the Z coordinates of X_R and X_L, we obtain a constraint on the homogeneous coordinates x_R and x_L. To take advantage of the linearity of the constraint with respect to Q, we represent Q as a vector q, and rewrite the constraint as

$$\chi q = 0 \tag{B.15}$$

where

$$\chi = [x_{R_1} x_{L_1}, x_{R_2} x_{L_1}, \ldots 1] \tag{B.16}$$

$$q = [Q_{11} Q_{21} \ldots Q_{33}]' \tag{B.17}$$

. Now if many 3-D points are observed in both cameras, each one can provide a scalar constraint, and we can form a tall matrix χ where each row corresponds to a different point.

We can now begin to solve for (R,T) by noting that $q \equiv Q = RT^{\wedge}$ is in the null-space of χ.

Let q be a vector in the null-space of χ, and let Q be the corresponding matrix.

Let $Q = U\Sigma V'$ be the singular value decomposition of Q. It can be shown through geometrical reasoning of the properties of $T^{\wedge}$ that the singular value decomposition is of form

$$U\Sigma V' = \left(R \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & T \end{bmatrix}' \tag{B.18}$$

where the third column of V is the unit norm translation vector T, and the two singular values λ_1, λ_2 determine the magnitude of the translation. If the observed homogeneous coordinates were noiseless, the two values would be equal, and the third singular value would be exactly zero. In practice, due to measurement noise, the two values are not exactly equal, and the third singular value is very small but not exactly zero. Note, however, that any scaled version of the vector q is also in the null-space of χ, so that the magnitude of the translation vector T is not recoverable. This restriction can also be given a geometrical interpretation: if the translation between the two cameras were made twice as large and the points in 3-D were placed twice as far away, the homogeneous coordinates of the points would be exactly the same, and the same exact trilinearity constraint will hold. (In fact, this argument is general and holds for any motion or pose recovery scheme based on observed projections of points; the global scale factor is unobservable).

Thus the Longuett-Higgins algorithm can be used to recover the relative pose of two cameras, up to a scale factor. The algorithm is not optimal, however, because the solution space of q does not enforce the fact that Q is created from a rigid transformation (R, T). $RT^{\wedge}$ only has 5 degrees of freedom (3 from the rotation matrix, and only 2 from the translation vector, since the magnitude of the translation (which defines the overall scale) is unrecoverable). $q \in nullspace(\chi)$ has 9 degrees of freedom. Projection of the found solution q onto the proper 5-dimensional sub-manifold is done when we ignore the singular values of the decomposition of q.

An optimal and more accurate estimate of the rigid transformation can be obtained by forcing the solution q of the trilinearity constraint to lie directly in the 5-dimensional sub-manifold, and this is what Horn's algorithm does, using the estimate provided by Longuett-Higgins as an initial guess.

Horn's algorithm

Given an initial (good) estimate of the rigid transformation between two cameras, Horn's algorithm refines the estimate by minimizing the trilinearity constraint

$$(x_R)' R T^{\wedge} x_L = 0 \tag{B.19}$$

directly over the 5 observable degrees of freedom. The 5 degrees of freedom are encoded as a 3-dimensional rotation vector ω, and the 2-dimensional spherical coordinates (θ, ϕ) defining the orientation of the unit-norm translation vector.

Given the rotation vector ω, the rotation matrix R is computed as the matrix exponential

$$R = e^{\hat{w}} = I + \hat{w} + \hat{w}^2 + ... \tag{B.20}$$

This matrix exponential can be computed efficiently via the Rodrigues' formula:

$$R = e^{\hat{w}} = I + \frac{\hat{w}}{\|w\|}\sin(\|w\|) + \frac{\hat{w}^2}{\|w\|^2}(1 - \cos(\|w\|)) \tag{B.21}$$

The minimization search for the optimal parameters $[\omega\ \theta\ \phi]$ was done with MATLAB's non-linear least-squares estimation routine 'leastsq'. At each iteration, an error vector is generated by applying the trilinearity constraint (equ.B.19) to the current estimate of the parameters. This error vector is used to determine how to incrementally modify the parameter estimate so as to continually minimize the norm of the error vector.

Although Horn's algorithm suffers from local minima, given a good initial guess it will converge to the right solution. Using Longuett-Higgins as initial guess enables Horn's algorithm to find this global minima.

Estimation of consistent scales

Although by using L-H's and Horn's algorithm we can arrive at an optimal estimate of the relative pose of each camera with respect to camera 1, our job is still not done, by two accounts.

First, the calibration of the entire system as a whole is not optimal, because we have not explicitly used the data from, say, cameras 2 and 4 together. In fact, if we evaluate the trilinearity constraint between cameras 2 and 4 (by composing the two rigid motions from camera 2 to camera 1, and from camera 1 to camera 4), we find that the error vector is considerably larger than the analogous error vector between, say, camera 2 and camera 1. Thus we must refine the pose estimates using all the available information directly.

But before doing that, we must deal with the second aspect of the unfinished business. We have mentioned already that the relative pose between two cameras can only be estimated up to a scale factor. However, if more cameras are used to observe the 3-D structure, the scale factors between those other cameras and camera 1 must be consistent with the scale factor between the first two cameras. We may not know whether cameras 1 and 2 are 1m or 5m apart, but having chosen a default scale (unit norm translation), there is a well defined scale for subsequent camera relative poses, so that the size of the reconstructed 3-D structure is the same irrespective of which pair of cameras are used. To obtain a first estimate for consistent scale factors, a robust estimate of the size of the structure can be computed as the root-mean-square distance of all the structure points from the structure mean for each camera view. Then the respective scale factors of each camera with respect to the first is the inverse of the ratio of their structure sizes.

Iterative pose refinement

Given initial pose estimates, (R_i, T_i), for each camera $i = 2..N$ with respect to camera 1, and given that these poses have scales consistent with each other, we now describe the algorithm used to refine the pose estimates using all the available information directly. This algorithm was suggested and perfected for us by none other than the great mathematico-geometrico-lexical genius Jean-Yves Bouguet-Cauchy-Gutenberg.

The algorithm is very clever, and simple in concept: Given the current pose estimates, an optimal 3-D point structure is determined. Then, given the current optimal structure, a new estimate for the pose of each camera (w.r.t. camera 1) is computed. The iterations are repeated until no significant improvement is seen in the 3-D reconstruction error of the points. In use, typically 10 or so iterations are necessary for convergence, and the 3-D reconstruction error is reduced by a factor of 2 to 5. The following two sections describe the details of the two steps in the iterative procedure.

Estimating structure given pose

Figure B.3 illustrates the geometry of the problem to be solved. Given the homogeneous coordinates p_i of a 3-D point P in $i = 1 \ldots N$ views, and given the relative pose (R_i, T_i) of view $i = 2 \ldots N$ with respect to the first view, we wish to triangulate the 3-D position of P using all of the observed views.

Mathematically, triangulation means finding the point P in space which minimizes the perpendicular distances between itself and all the projective rays, Λ_i, from each view. Suppose the minimum distance point along the projective ray of view i is a distance λ_i from the respective camera center. The coordinates of this point in the reference frame of view

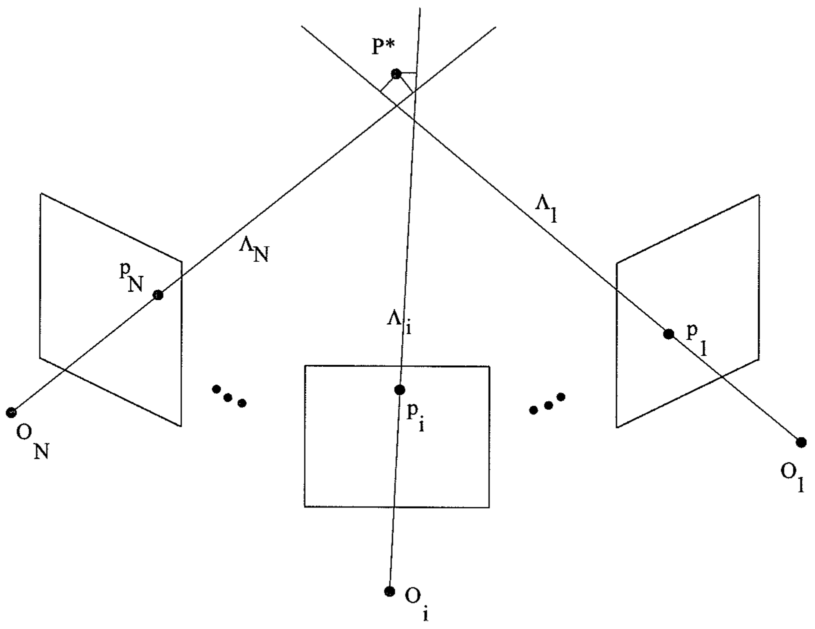


Figure B.3: The 3-D reconstruction problem. Given observations of a point in N cameras, there is a unique point P in 3-D which minimizes the sum of squares of perpendicular distances to all the projective rays.

i are

$$P_{min\ dist,\ view\ i\ coords} = \lambda_i p_i \tag{B.22}$$

And in the reference frame of view 1 the minimum distance point on the projective ray of view i becomes

$$P_{min\ dist,view\ 1\ coords} = \lambda_i R_i p_i + T_i \tag{B.23}$$

Thus we can write a system of over-constrained equations which we can solve for in the least-square sense for P, the coordinates of the 3-D point in the reference frame of view 1, and all the λ_i's:

$$\begin{aligned} P - (\lambda_1 p_1) &= 0 && \text{(B.24)} \\ P - (R_2 \lambda_2 p_2) &= 0 && \text{(B.25)} \\ &\vdots && \text{(B.26)} \\ P - (R_N \lambda_N p_N) &= 0 && \text{(B.27)} \end{aligned}$$

This system of equations can be rewritten as a total least squares problem:

$$\begin{bmatrix} -I & \Lambda_1 & & T_1 \\ \vdots & & \ddots & \vdots \\ -I & & \Lambda_N & T_N \end{bmatrix} \begin{bmatrix} P \\ \lambda_1 \\ \vdots \\ \lambda_N \\ 1 \end{bmatrix} = 0 \tag{B.28}$$

where $\Lambda_i = R_i p_i$. Thus the solution lies in the null-space of the right-hand-side matrix, which can be found through singular value decomposition as the vector corresponding to the smallest singular value, renormalized so that the last component is 1.

Note that if the optimal solution for P and λ's is used in equation B.28, the right-hand side of the equation is the set of minimum distances between P and each projective ray. The root of the mean of the squares of these distances can be used as a measure of the accuracy of the 3-D reconstruction.

Estimating pose given structure

With the newly estimated 3-D structure, $\{P^k\}$, where P^k is the 3-D coordinates of point k in the first camera frame, it is possible to refine the pose estimates of each camera with respect to camera 1. This is done for each camera independently, using a least-squares optimization procedure. It is even done for the first camera, so in actuality the pose estimates are with respect to the 'initial' position of camera 1. After all the pose-structure iterations have been done, the new position of camera 1 is redefined as the reference for all the other cameras.

For view i, compute the expected homogeneous coordinates $\tilde{p}_i^k$ for point k given the 3-D structure and the current pose estimate (R_i, T_i):

$$\tilde{p}_i^k = Proj(R_i P_i^k + T_i) \tag{B.29}$$

where $Proj()$ is the perspective projection operator (computes the homogeneous coordinates by dividing a 3-vector by its Z component).

Next the error e_i^k between the expected homogeneous coordinates and the actual ones observed is computed:

$$e_i^k = \tilde{p}_i^k - p_i^k \tag{B.30}$$

This error is minimized. In order for the minimization to work properly, the rigid transformation (R_i, T_i) is represented with the appropriate degrees of freedom, namely, a rotation vector ω_i to encode the rotation, and the translation vector itself, T_i, to encode translation. Note that here the magnitude of the translation is an observable degree of freedom because we know the actual scale of the 3-D structure $\{P^k\}$.

Estimation of global scale

The final step for extrinsic calibration is to estimate the global scale parameter. Recall that in the previous steps the rigid transformation from camera 1 to camera 2 was arbitrarily chosen to have unit-scale translation (which may have drifted slightly due to the structure-pose iterative refinement). Given the best possible consistent set of camera poses (up to scale), we can calculate the best possible 3-D structure. The structure is nothing more than the position of the two light-bulbs on the rod in a thousand or so frames. We can compute the mean value of the reconstructed rod length, and determine the global scaling factor needed to make this mean value equal to the true measured rod length. Multiplying all the rigid transformation translations by this global scale factor gives us the true physical distances between the cameras, and allows the 3-D reconstruction to have true world lengths.

For sake of convenience, since during reconstruction any particular combination of cameras may be used, the set of all rigid transformations from camera $i = 1 \dots N$ to camera $j = 1 \dots N$ is computed given the transformations from cameras i and j to camera 1 as

$$(R_{ij}, T_{ij}) = \left(R'_{j1} R_{i1}, R'_{j1}(T_{i1} - T_{j1})\right) \tag{B.31}$$

The outcome of this complicated and laborious calibration is the ability to very accurately reconstruct the 3-D position of observed light-bulbs. The typical reconstruction error of quasi-intersecting projective rays from different cameras is 1.5mm (standard deviation) over a workspace volume of 2m by 3m, and for which the cameras are spaced at a distance that varies from 4m to 8m.

B.2.2 Real-time dot detection

As mentioned previously in the hardware description section, the position of the light-bulbs throughout time are detected in real-time in order to eliminate the need to store a large number of images for off-line processing. The process of real-time detection executed on each camera image sequence consists of the following steps:

- The 320x240 pixel even field image is acquired every 1/60th of a second with the camera set to 'field-mode' (adjacent even/odd rows are used as one row and deliver an image at field rate).
- The image is blurred in both directions by the smoothing kernel [1 2 1]. This allows local maxima to be detected reliably.
- Brightness local maxima are detected by finding pixels with intensities that are greater than those of its eight surrounding pixel neighbours. Only maxima greater than some pre-determined threshold are kept. This eliminates false, random detections due to noise in the background. Since the acquisition process is done in near-darkness, each local maxima represents a lightbulb. By picking an appropriate threshold level, and adjusting the camera image gain, it is possible to detect the lightbulbs quite reliably.
- For each detected local maxima, the center of brightness is calculated with sub-pixel accuracy (1/10th of a pixel accuracy). This is done by fitting a paraboloid to the image intensity on the 3x3 pixel block centered on the local maxima.
- The sub-pixel estimates of the dot positions are written to disk, along with the frame number time stamp.

The slowest parts of the algorithm are the blurring and initial detection of the local maxima, since they have to be done over the entire image. To speed up these computations, they were hand-coded using the Intel MMX instruction set. As a result of that, it was possible to detect the dots reliably at 60Hz on 233MHz PII machines, missing on average only 1 out of 1000 frames.

After the entire sequence is acquired, the data files on disk are post-processed. The data is converted to a large MATLAB matrix, where blank rows are inserted wherever missing frames are detected. In this way, correspondence between data frames from different cameras is assured.

B.2.3 3-D reconstruction

Once data has been acquired, and the intrinsic and extrinsic camera calibrations have been done, it is possible to combine the data from all the cameras to reconstruct the 3-D position of the light-bulbs throughout time. The task is not completely straightforward because of the 2-D feature correspondence problem: each camera observes up to 18 markers. In order to compute the 3-D coordinates, the dots detected in one camera have to be associated with the appropriate dots in the other cameras. With 4 cameras and 18 markers, there are $18!^3$ possible combinations, of which only one will produce the correct result. The process used to solve this problem in an efficient manner consists of three steps:

- Tracking of 2-D segments: for each camera, the dots detected in each frame are grouped with corresponding dots in subsequent frames as much as possible (limited by occlusions of dots and ambiguous correspondences).
- Identification of potential matching segments: For all the pairs of 2-D segments from different cameras, the 3-D reconstruction error over the time interval for which the two 2-D segments overlap is computed. If the two segments represent the same marker in different views, the reconstruction error will be quite small, on the order of 3mm. If they represent different markers, the reconstruction error will typically be very large. A matrix of possible segment matches is built consisting of all the pairs of 2-D segments that match with less than 0.7 cm error.
- Group 2-D segments into 3-D candidates: Using the information in the matrix of possible segment matches, for each frame in time, all the segments present in that frame are grouped together so that each group represents the 2-D camera support for a 3-D marker. Once the maximal 2-D support for each marker is determined, the optimal 3-D positions of the markers are computed.

Tracking of 2-D segments

Rather than trying to find the inter-camera correspondences of observed dots on a frame-by-frame basis, it is advantageous to first track the motion of the dots in each camera to form "2-D segments". The advantage is two-fold:

First, the number of times that the 3-D inter-camera correspondence problem has to be solved can be greatly reduced. In the ideal case, where the 18 markers are detected in every frame in every camera, it may be possible to end up with only 18 2-D segments for each camera. That is, for each camera, given the positions of the markers in the first frame, it may be possible to follow the motion of each dot in subsequent images until the end of the data sequence. In that case, the 3-D inter-camera correspondence of the dots can be solved for the entire sequence in one fell swoop.

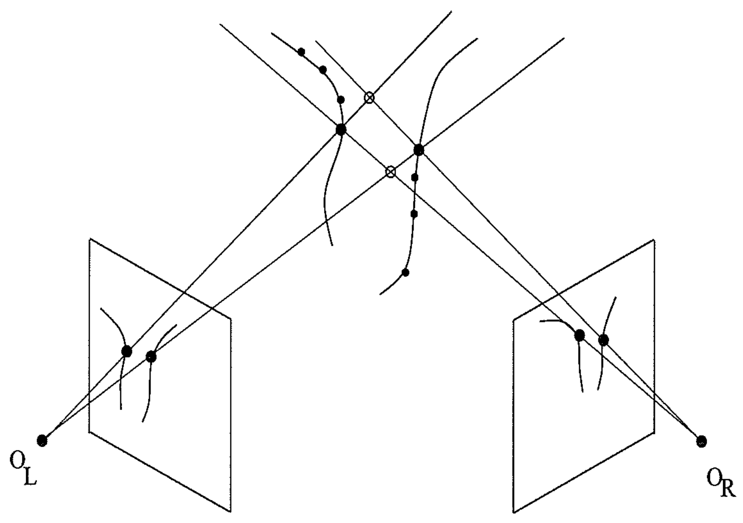


Figure B.4: During a single frame, it could happen that two markers are coplanar with two camera centers. In that case, reconstruction is ambiguous and incorrect correspondences can result in ghost markers being reconstructed (open circles). If markers are tracked in each view, these errors are virtually eliminated, since it is unlikely that markers will remain coplanar in subsequent frames.

Second, by using 2-D segments instead of just 2-D dots in a single frame, the 3-D correspondence problem can be solved much more reliably. In any given frame, it is likely to happen from time to time that two markers are in the same plane as two camera centers (fig. B.4). In that case it is impossible to know which of two choices of correspondences is correct, and if the incorrect choice is made, a 3-D reconstruction that is completely wrong will result. However, in subsequent frames, due to the motion of the body, the markers will no longer be coplanar. Thus, if 2-D segments rather than individual dots in a single frame are matched, these 3-D correspondence ambiguities can be avoided.

For each camera, tracking of the dots to build 2-D segments is done in the following way:

- Keep a list of the segments present in the current frame (in the first frame of the sequence, new segments are initialized, one per dot).

- For each segment in the current frame, make a prediction as to where the dot will be in the next frame. If the dot has been tracked from the previous frame, the 2-D velocity of the dot is used to predict the new position; otherwise, if it's a newly formed segment, the predicted position is the same as the current position.

- A matrix is generated where each row represents the distance between the predicted position of a segment's dot and all the dots that are actually observed in the next frame.

- The matrix is sorted within each row to find the nearest and second-nearest observed dot for every segment.

- Apply thresholds to guarantee that the dot is tracked conservatively, without mistakes. For each segment, if a velocity-based prediction of the new position was made, then the closest dot must be less than 1.5 pixels away. If a position-based prediction was used, then the closest dot must be less than 6 pixels away. To be sure that the choice is unambiguous, the second closest dot must be sufficiently away from the prediction. For velocity-based predictions, it must be more than 2 pixels away, and for position-based predictions it must be more than 6 pixels away. If the first and second closest dots don't satisfy these conditions, the current segment is ended, and new segments will be generated with the unmatched dots in the next frame.

- A final test for consistency for the segments that survived the threshold test is that the closest dot must be unique for each segment. If two segments share the same closest dot, the matching is deemed ambiguous and neither segment is continued into the next frame.

Since the rest of the 3-D reconstruction algorithm relies on the correctness of the 2-D segments, it is crucial to do the 2-D tracking and building of the segments in a reliable and conservative way. If two dots representing different markers become part of the same segment due to incorrect tracking, this may produce errors in later stages of the reconstruction process. With the threshold values mentioned above, the tracking is in fact quite reliable, producing no errors in over 20 sequences of 7000 frames (2 minutes of action) each. In a typical sequence, approximately 1500 segments are generated per camera. Thus the mean length of a tracked segment is 7000*18/1000 = 80 frames. This statistic is a bit misleading, though. The median length of segments is 1 frame; approximately 70% of the segments are 1 frame long. But the remaining 30% of the segments can be quite long, sometimes thousands of frames long, and still contain enough information to be able to generate a perfect 3-D reconstruction.

Identification of potential matching segments

After all the dots in each camera have been tracked and grouped into 2-D segments, a large matrix is generated which stores the accuracy of reconstruction and length of overlap in time for any two 2-D segments from different cameras. This matrix is used to identify which pairs of 2-D segments correspond to the same 3-D marker in space.

Let p_m^t and p_n^t be the homogeneous coordinates of the dots at time t of two segments m and n that come from cameras i and j, respectively, and suppose the two segments overlap in time from $t = t_0$ to $t = t_N$). We test the hypothesis that the two segments correspond to projections of the same 3-D marker by computing the 3-D reconstruction error of the triangulation for each time instance of the segments' overlap.

At each time instance t, we need to find the distances λ_i^t, λ_j^t along each projective rays at which the two rays intersect. In the reference frame of camera j, these two rays are:

$$R_{ij}\lambda_i^t p_i^t + T_{ij}, \quad and \tag{B.32}$$

$$\lambda_j^t p_j^t \tag{B.33}$$

At the optimal choice for λ_i^t and λ_j^t the distance between the two points on the projective rays is minimum, so we can solve for the two parameters by solving in the least squares sense:

$$\begin{bmatrix} R_{ij}p_i^t & -p_j^t & T_{ij} \end{bmatrix} \begin{bmatrix} \lambda_i^t \\ \lambda_j^t \\ 1 \end{bmatrix} = 0 \tag{B.34}$$

The 3-D reconstruction error at time t, e^t can then be computed as 1/2 of the value obtained using the optimal λs in equation B.34.

From the extrinsic calibration procedure we know that the reconstruction error of our system has a standard deviation of 1.5mm. Thus a conservative decision (not wanting to discard any potentially matching segments) as to whether or not the two segments indeed correspond and represent the same 3-D marker can be made as follows: if the maximum norm of the reconstruction error in time is less than 7.0mm, the segments are considered as possibly matching. The mean (over time) reconstruction error and the length of the overlap of the two segments satisfying the test are stored in the matrix.

The matrix is called the matrix of 'possibleSegmentMatches'. Since the criteria for selection was quite conservative (more than 4 standard deviations of the typical reconstruction error), it is likely that many of the entries in the matrix in fact are not corresponding segments. This may be the case for segments which overlap for a very short time, or for which the corresponding markers do not move much over the overlap interval (and thus potentially remain co-planar, causing the 3-D correspondence ambiguity).

In the next step of the reconstruction process, the information in the matrix of possible segment matches is combined to find the optimal set of correspondences amongst all cameras for each 3-D marker.

Group 2-D segments into 3-D candidates

Given the list of 2-D segments in all the cameras, and the matrix of possible segment matches, this information must be processed in such a way as to produce the optimal 3-D estimates for the markers. The processing proceeds frame by frame, and is based on the concept of 'groups of candidates'. A 'candidate' is a set of 2-D segments that have been deemed to be in correspondence and represent a 3-D marker. A 'group' is a set of candidates that have the same camera support. That is, all the candidates in a group have 2-D segments in the same set of cameras.

As an example, at the start of processing the sequence on the very first frame, we will have 4 groups, each one representing candidates with single camera support from camera 1, camera 2, etc. Each group thus lists all the 2-D segments present in a particular camera, as in Table B.2.3. Ideally, after processing of the groups, the groups have been merged into one group with camera support [1 2 3 4]. This can happen if all the markers were observed in all the cameras. Then the 3-D marker position can be computed using the information

Camera Support		
[1]	[2]	[3]
a_1 – –		
b_1 – –		
c_1 – –		
d_1 – –		
	– a_2 –	
	– b_2 –	
	– c_2 –	
		– – a_3
		– – c_3
		– – d_3

Camera Support		
[1 2 3]	[1 2]	[1 3]
a_1 a_2 a_3		
c_1 c_2 c_3		
	b_1 b_2 –	
		d_1 – d_3

Table B.1: Grouping of candidates into groups with maximal camera support. Initially, there is once candidate for each marker observed in each camera. At the end of the grouping process, candidates are grouped into new groups with maximal camera support. Here, markers a and c were seen by all 3 cameras, but markers b and d only by two.

from all the segments. In practice, all markers are not seen by all the cameras, but the point of the method is that after processing the groups, each 3-D marker is represented by a candidate that has maximal camera support.

The algorithm for processing the groups of candidates proceeds as follows:

For each frame

- build initial groups
- find two groups to match
- find matching candidates in the two groups
- put the matching candidates in a new group
- repeat from second step above until all groups have been matched
- compute 3-D positions for all the candidates

Although the algorithm proceeds frame-by-frame throughout the data, all the computations for matching candidates are done with segments and not individual dots (in order to eliminate instantaneous 3-D correspondence ambiguities). It may appear then at first sight that traversing through the data one frame at a time is sub-optimal and that many calculations are repeated. However, this is not the case; when the initial groups in a new frame are generated, information of previously matching segments is used. If exactly the same set of segments exist in all the cameras in the new frame, the initial groups will in fact be the final grouping result of the previous frame.

Instead of proceeding through the data one frame at a time, it would also be possible to just jump to the next frame where a new segment is present or an old one disappears. It is algorithmically equivalent, but the frame-by-frame method is simpler to implement.

Below is a description of each step of the candidate grouping algorithm:

Build initial groups

To build the initial groups of candidates for a new frame, first a list of the (so far) unmatched 2-D segments present in each camera in that frame is made. Then two passes are made to match segments in the list.

In the first pass, for each segment in the current list of unmatched segments, a search is made to find other segments that have already been matched with it in previous frames. This is done with the help of the 'matchedSegments' matrix, which is updated at the end of each frame's grouping process. It is the same size as the 'possibleSegmentMatches' matrix described previously, except that while the 'possibleSegmentMatches' matrix indicated which segments might be in correspondence, the 'matchedSegments' matrix represents the set of segments which are known to match.

For example, suppose a segment in camera 1, say, is found to have already been matched to segments in cameras 3 and 4, say, which are also present in the current frame. Then the three segments will be added as a new candidate to the group with camera support $\{134\}$ (a new group being generated if no such group already exists), and the three segments will be removed from the list of unused segments.

Thus at the end of the first pass, the groups have incorporated the information of all previously determined matches within the set of segments present in the frame. However, the list of unused segments may not be empty, if new segments have appeared in this frame and/or previously matched segments have disappeared.

In the second pass, whatever unmatched segments are left in the list are searched for new matches amongst themselves using the information in the 'possibleSegmentMatches' matrix. Theoretically, this pass is not necessary. The unmatched segments could be left in groups with single camera support, and later stages of the algorithm would match them.

But if the later algorithm were to try to match candidates from two groups with single camera support, it would perform the same computations that were done to generate the 'possibleSegmentMatches' matrix. Thus, for sake of efficiency, the segments are matched pair-wise right here, using the same criteria as the later stage will use. Namely, the segments must overlap long enough in time (20 frames), and the mean 3-D reconstruction error must be small enough (0.3 cm).

Find two groups to match

Given the initial groups of candidates (and then again for subsequent iterations of group merging), two groups of candidates that have not yet been analyzed for matches are found. Each group maintains a list of the groups it has already been matched against. Thus starting with the first group, the other groups are checked to see if they qualify to be matched against. A group will qualify if 1) it hasn't been matched before, and 2) its camera support is 'orthogonal' to the first; since we want to merge the candidates of the first group with that of the second, this only makes sense if the two groups do not have any cameras in common (a 3-D marker can have only one 2-D segment associated in each camera for a particular frame, thus for each camera, either a segment exists in the first group, or in the second - but not both).

If no two groups satisfying the matching criteria are found, then we know that the groups have been maximally merged, and we can proceed to the last step of the algorithm, computing the optimal 3-D marker positions given the maximal camera support groups. If two groups are found, then the candidates must be analyzed to see if any of them represent the same 3-D marker.

Find matching candidates in the two groups

Suppose two groups, g_1 and g_2, have not been matched yet, and g_1 has N_1 candidates, and g_2 has N_2. The search for matching candidates in the two groups is done in two steps. First, an $N_1 \times N_2$ matrix with the 3-D reconstruction errors for each possible combination of candidates is computed. Then the matrix is processed to find good matches.

Computing the match matrix

For each candidate $c_1 = 1..N_1$ in g_1 the 3-D reconstruction error resulting from triangulating with each candidate $c_2 = 1..N_2$ in g_2 must be computed. The result is stored as entry (c_1, c_2) of match matrix M. Rather than computing each triangulation outright, a quick test is first performed to immediately eliminate mismatches and increase the robustness of the algorithm. The test is to check that each combination of segments from the two candidates is a possible segment match (i.e., it is a non-zero entry in the possibleSegmentMatches matrix), and that the overlap in time of the two segments is at least 20 frames (1/3 of a second). If the candidates do not pass this test, a huge distance, 100 cm, is stored as the match error M_{c_1,c_2}. If they do pass the test, then the match error is computed in a fashion similar to that used in the section on estimating structure given the pose (equation B.28).

Selecting good matches

Once the match matrix M is computed, the candidates are matched in a safe fashion, similar to that for tracking 2-D segments. For each row in M (i.e., for each candidate in group 1), the best and second-best matching candidates in group 2 are found. Rows for which the best match is too far away (reconstruction error > 0.3 cm), or for which the second-best match is too close (recon error < 1.0 cm) are thrown away. Of the remaining rows, only those that have a unique matching candidate from group 2 are kept. These rows (and they're respective best-match columns) are the candidates deemed to be matching in the two groups.

Put the matching candidates in a new group

After finding the list of matching candidates in the two groups, several bookkeeping operations are performed. First, entries are added to the matchedSegments matrix (to be used in the group building phase of forthcoming frames). Second, the data structure has to be updated: the matched candidates are removed from their old groups, and appended to a new group that has the combined camera support (this new group may have to be generated if it does not already exist). If the old groups are now empty, they are removed from the list of groups. Each group's list of checked groups also needs to be updated.

Compute 3-D positions for all the candidates

Once groups have been completely matched, so that candidates have been reliably merged into groups of maximal camera support, we can compute the optimal 3-D marker positions for the current frame, solving for P in equation B.28. For each marker computed, the set of 2-D segments used to compute it is also stored (to be used later when labeling the markers).

At this point of the 3-D reconstruction process, for a typical motion capture sequence, we have managed to accurately reconstruct almost 100% of the data. Some marker positions are occasionally missing, due to temporary occlusions or the tracked 2-D segments being of too short duration.

Labeling

In order for the 3-D data to be useful, we need to know which body part is associated with each marker in time. There are several ways to arrive at this information. One simple but laborious way would be to label each marker in each frame as 'Head', 'Neck', etc. One could also conceive of a method of tracking the 3-D marker motion in space, and then just needing to label the very first frame. We have developed a method that uses the information in the matchedSegments matrix to 'weave' the matched 2-D segments into 3-D 'threads'. These threads are then connected together into labeled 'chords' using a statistical method.

Weaving matched 2-D segments into 3-D threads

Suppose we start with a 2-D segment of long duration. Using the matchedSegments matrix, we can find all the 2-D segments in other cameras that this segment has been matched against. We can then repeat the process with each of the found segments in turn, until no new matched segments are found. This set of inter-matched segments is called a 'thread'. Since the segment matching was done robustly and reliably, matched segments exhibit a transitive property, so that we know that the entire set of inter-matched segments, the thread, corresponds to a unique 3-D marker. Also, since when 3-D marker positions were computed the corresponding 2-D segments used were stored, it is possible to associate a 3-D marker for each frame of each segment that the thread exists in.

All the segments in the matchedSegment matrix are threaded in the manner described above. In the ideal case, where all markers are always seen by at least one camera, we would have 18 threads (one per marker) that would span the entire sequence. Then labeling the initial frame would give us the labeling for the entire sequence. In a typical sequence, due to occlusions, 20 to 30 threads are usually found. We can label the first 18 threads that appear in the initial frame, but eventually some of these threads will disappear and new ones will appear. Since there aren't usually that many threads, one could easily label each one manually. But there is also an automatic way of labeling the new threads.

Connecting threads into labeled chords

We define a chord to be the list of threads that correspond to the same, labeled body marker. For the first frame of the sequence, the chords are initialized manually with the appropriate threads. The statistics for the mean and standard deviation of 3-D distances between chords is computed over the frames where threads have already been associated to chords. Then, whenever threads disappear and new ones appear, a fit coefficient is calculated between the new threads and the labels that do not have a thread associated. The new threads can be associated with the proper label by picking the correspondences that have the best fit coefficients.

Appendix C EM and extension for MAP inference

C.1 The EM algorithm for parameter estimation

(This derivation of EM thanks to Dr. Max Welling.)

Let $p_\theta(x, y)$ be a probability distribution function parameterized by parameters θ. Suppose that y are observed variables, and x are hidden (or latent) variables of the model. To make the analogy clear, in the Mixture of Gaussians models described in section 7.4 the hidden (latent) variable is c_i, a discrete quantity identifying one of many Gaussian clusters. The observed variables (here denoted by y) are the motion output vector and the label (confusingly named y and x, but one should think of the compound vector $(x'y')'$ of section 7.4 as being y here). The parameters of the model are the a priori cluster probabilities, $p(c_i)$, and the mean and variance of each individual cluster, μ_i and Σ_i.

Let $\{y\}$ be a set of observations of the observed variables. We want to find the parameters $\hat{\theta}$ that maximize the (log) likelihood of the observed data:

$$\begin{aligned}\hat{\theta} &= \arg\max_\theta L(y|\theta) \\ &= \arg\max_\theta \sum_{\{y\}} \log p_\theta(y) \\ &= \arg\max_\theta \sum_{\{y\}} \log \int p_\theta(x, y) dx\end{aligned}$$

Due to the integral (or summation, if the latent variables are discrete) inside the logarithm, the maximization is difficult to compute. The EM algorithm provides a simple and robust way to perform the maximization by iteratively maximizing the expected log-likelihood of the complete data $(E_x(L(x, y|\theta)))$ given a current guess of the distribution of the latent variables.

Let $r(x)$ be a distribution over the latent variables, so that $\int r(x)dx = 1$. Then

$$\begin{aligned}
L(y|\theta) &= \sum_{\{y\}} \log p_\theta(y) \\
&= \sum_{\{y\}} \int r(x) \log p_\theta(y) dx \\
&= \sum_{\{y\}} \int r(x) \log \frac{p_\theta(x,y)}{p_\theta(x|y)} dx \\
&= \sum_{\{y\}} \int r(x) \log \frac{p_\theta(x,y)}{r(x)} \frac{r(x)}{p_\theta(x|y)} dx \\
&= \sum_{\{y\}} \int r(x) \log p_\theta(x,y) dx - \sum_{\{y\}} \int r(x) \log r(x) + \sum_{\{y\}} \int r(x) \log \frac{r(x)}{p_\theta x|y} dx \\
&\doteq Q(r(x), p_\theta(x,y)) + H(r(x)) + KL(r(x), p_\theta(x|y))
\end{aligned}$$

The third term in the last expression, $KL(r(x), p_\theta(x|y))$, is the Kullback-Leibler distance between the two distributions and has the property that

$$\begin{aligned}
KL(r(x), p_\theta(x|y)) &\geq 0 & \text{(C.1)} \\
&= 0 \; iff \; r(x) = p_\theta(x|y) & \text{(C.2)}
\end{aligned}$$

Now suppose that at the k^{th} iteration of the EM algorithm we have parameter estimate θ_k. Let $r_k(x) = p_{\theta_k}(x|y)$. Then

$$\begin{aligned}
L_k &= Q(r_k, p_{\theta_k}(x,y)) + H(r_k) + KL(r_k, p_{\theta_k}(x|y)) \\
&= Q(r_k, p_{\theta_k}(x,y)) + H(r_k)
\end{aligned}$$

Now let $\theta_{k+1} = \arg\max_\theta Q(r_k, p_\theta(x,y))$. Then we have

$$\begin{aligned}
L_{k+1} &= Q(r_k, \theta_{k+1}) + H(r_k) + KL(r_k, \theta_{k+1}) \\
&\geq Q(r_k, \theta_{k+1}) + H(r_k) \\
&\geq Q(r_k, \theta_k) + H(r_k) = L_k
\end{aligned}$$

Thus the log likelihood of the data increases (or stays the same) with each iteration. The EM algorithm converges to a maxima, but unfortunately not necessarily the global maxima.

C.2 EM for probability maximization

In section 7.6 we needed to find the point in trajectory space that maximized the probability of the trajectory. This maximization is difficult because the trajectory probability is modeled with latent variables. Here we show how the EM algorithm can be used to perform the maximization iteratively.

If $p(y) = \int p_\theta(x,y)dx$ is the latent variable model, we want to find y that maximizes $p(y)$. If we exchange the roles of θ and y in the EM algorithm, it can be used to perform the maximization. We treat θ as the 'data', and y as the 'parameter' to estimate. At each iteration, the current estimate of y, y_k is used to define $r_k(x) = p(x|y_k)$, and $Q(r_k, p(x,y))$ is maximized over y.

Just as with the use of EM for parameter estimation, the use of EM for probability maximization also suffers from getting trapped in local maxima, so that either a good initial estimate is needed, or some extension of EM robust to local maxima needs to be used, such as that of [35].

Bibliography

[1] B. Anderson and J. Moore. *Optimal Filtering.* Prentice-Hall, 1979.

[2] N. Badler, C.B. Phillips, and B.L. Webber. *Simulating Humans.* Oxford University Press, 1993.

[3] E. Di Bernardo, L. Goncalves, and P. Perona. Monocular tracking of the human arm: Real-time implementation and experiments. In *Proc. 13th Int. Conf. Pattern Recognition*, pages 622–626, Wien, August, 1996.

[4] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press - Oxford, 1995.

[5] V. Blanz and T. Vetter. A morphable model for the synthesis of 3-d faces. In *SIGGRAPH 99 Conference Proceedings*, pages 187–1894, August 1999.

[6] M. Brand. Structure learning in conditional probability models via an entropic prior and parameter extinction. *Neural Computation*, (11):1155–1182, 1999.

[7] M. Brand. Voice puppetry. *Siggraph 99 conference proceedings*, pages 21–28, August 1999.

[8] C. Bregler and J. Malik. Tracking people with twists and exponential maps. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, pages 8–15, Santa Barbara, CA, 1998.

[9] Armin Bruderlin and Lance Williams. Motion Signal Processing. In *Int. Conf. on Computer Graphics (SIGGRAPH) '95*, pages 97–104, August 1995.

[10] Martin Casdagli. A dynamical approach to modeling input-output systems. *Nonlinear Modeling and Forecasting*, 1992.

[11] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, B*, 39(1):1–38, 1977.

[12] D.M. Gavrila and L.S. Davis. 3-d model-based tracking of humans in action: a multi-view approach. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, pages 73–80, San Francisco, June, 1996.

[13] N. Gershenfeld, B. Schoner, and E. Metois. Cluster-weighted modeling for time series analysis. *Nature*, 397:329–332, 1999.

[14] M. Gleicher. Motion editing with spacetime constraints. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, 1997.

[15] L. Goncalves, E. Di Bernardo, and P. Perona. Reach out and touch space (motion learning). In *Proceedings of the IEEE Third International Conference on Automatic Face and Gesture Recognition*, pages 234–239, April 1998.

[16] L. Goncalves, E. Di Bernardo, E. Ursella, and P. Perona. Monocular tracking of the human arm in 3d. In *Proc. 5^{th} Int. Conf. Computer Vision*, pages 764–770, Boston, June, 1995.

[17] I. Haritaoglu, D. Harwood, and L. Davis. W4 - real-time detection and tracking of people and thier parts. In *Proceedings of the IEEE Third International Conference on Automatic Face and Gesture Recognition*, pages 273–279, April 1998.

[18] J. Hayes. Designing moves for characters. *Game Developers Conference, San Jose*, October 1998.

[19] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating Human Athletics. In *Computer Graphics Proceedings, Annual Conference Series*, pages 71–78, 1995.

[20] B.K.P. Horn. Relative orientation. *Int. J. of Computer Vision*, 4:59–78, 1990.

[21] M. Isard and A. Blake. The condensation algorithm: conditional density propagation and applications for visual tracking. *Advances in Neural Information Processing Systems*, 9:361–368, 1997.

[22] G. Johansson. Visual perception of biological motion and a model for its analysis. *Perception and Psychophysics*, 12, 1973.

[23] M. Jordan. *Learning in Graphical Models (Adaptive Computation and Machine Learning).* MIT Press, 1999.

[24] A. Krogh, R. Palmer, and J. Hertz. *Introduction to the Theory of Neural Networks.* Persues Press, 1991.

[25] M. E. Leventon and W. T. Freeman. Bayesian estimation of 3-d human motion. Technical report, Mitsubishi Electric Research Labs, 1998.

[26] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[27] D. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5), May 1991.

[28] R. Murray, Z. Li, and S. Sastry. *A Mathematical Introduction to Robotic Manipulation.* CRC Press, Inc., 1994.

[29] Ken Perlin. Real Time Responsive Animation with Personality. *IEEE Transaction on Visualization and Computer Graphics*, 1(1):5–15, March 1995.

[30] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes in C.* Cambridge University Press, 1988.

[31] Jim Primbs. *Nonlinear Optimal Control: A Receding Horizon Approach.* PhD thesis, California Institute of Technology, 1999.

[32] K. Rohr. Incremental recognition of pedestrians from image sequences. In *Computer Vision and Pattern Recognition*, pages 8–13, New York City, June, 1993.

[33] C. Rose, M. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, pages 32–40, September/October 1998.

[34] Y. Song, L. Goncalves, E. Di Bernardo, and P. Perona. Monocular perception of biological motion - detection and labeling. In *IEEE Proceedings of the Seventh International Conference on Computer Vision*, pages 805–812, September 1999.

[35] N. Ueda and R. Nakano. Deterministic annealing EM algorithm. *Neural Networks*, 11:271–282, 1998.

[36] A. Witkin and M. Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, 1988.

[37] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder : real-time tracking of the human body. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, July 1997.

[38] C. Wren and A. Pentland. Dynamic modeling of human motion. *Proceedings of the IEEE Third International Conference on Automatic Face and Gesture Recognition*, pages 22–27, April 1998.