In Proceedings of Eurographics, Granada, Spain, 2003, 313-322

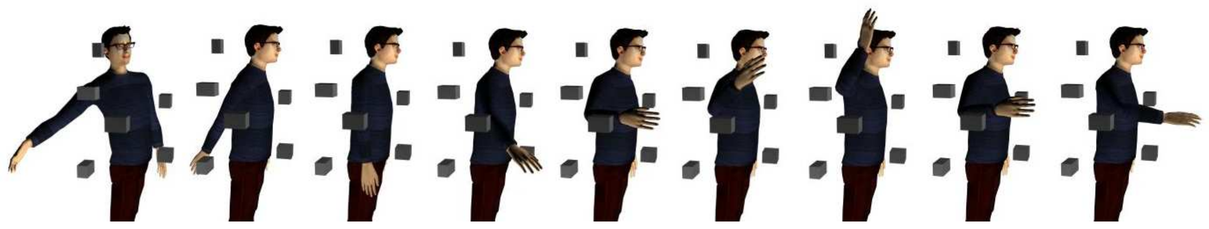
EUROGRAPHICS 2003 / P. Brunet and D. Fellner (Guest Editors)

Volume 22 (2003), Number 3

Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping

Marcelo Kallmann,[1†] Amaury Aubel,[2†] Tolga Abaci[3] and Daniel Thalmann[3]

[1] Robotics Research Lab, University of Southern California, Los Angeles, United States, kallmann@usc.edu
[2] DreamWorks Animation, Glendale, United States, aaubel@anim.dreamworks.com
[3] Virtual Reality Lab, Swiss Federal Institute of Technology, Lausanne, Switzerland, {tolga.abaci|daniel.thalmann}@epfl.ch



Abstract

We present new techniques that use motion planning algorithms based on probabilistic roadmaps to control 22 degrees of freedom (DOFs) of human-like characters in interactive applications. Our main purpose is the automatic synthesis of collision-free reaching motions for both arms, with automatic column control and leg flexion. Generated motions are collision-free, in equilibrium, and respect articulation range limits. In order to deal with the high (22) dimension of our configuration space, we bias the random distribution of configurations to favor postures most useful for reaching and grasping. In addition, extensions are presented in order to interactively generate object manipulation sequences: a probabilistic inverse kinematics solver for proposing goal postures matching pre-designed grasps; dynamic update of roadmaps when obstacles change position; online planning of object location transfer; and an automatic stepping control to enlarge the character's reachable space. This is, to our knowledge, the first time probabilistic planning techniques are used to automatically generate collision-free reaching motions involving the entire body of a human-like character at interactive frame rates.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Recent research in the character animation domain has mainly concentrated on the generation of realistic movements using motion capture data. Probably on account of its difficult nature, the problem of automatically synthesizing collision-free motions for object manipulation has received little attention from the Computer Graphics community. Most of the techniques developed[1] have not sufficiently explored this domain.

The automatic generation of collision-free grasping sequences has several direct applications in virtual reality, games, and computer animation. And yet, producing collision-free grasping motions currently involves lots of tedious manual work from designers.

Motion planning originated in Robotics, with an emphasis on the synthesis of collision-free motions for any sort of robotic structure[2]. Some works have applied mo-

† Work done while at EPFL - Virtual Reality Lab

tion planning to animate human-like characters manipulating objects[3,4]. However, the nature of the articulated structures being controlled is usually not taken into consideration. Most often, only one arm is used for reaching while the rest of the body remains static.

We present in this paper a collection of new techniques based on probabilistic motion planning for controlling human-like articulated characters. Our goal is to synthesize valid, collision-free grasping motions while taking into account several human-related issues, such as: control of the entire body (including leg flexion, spine and clavicle-shoulder complex), joint coupling (e.g. spine), articulation limits, and comfort criteria.

We mainly concentrate on the reaching phase problem, i.e., how to compute a valid collision-free motion between two postures. Our method operates on 22 degrees of freedom (DOFs) of an abstract control layer, mapped to the actual DOFs of the character (over 70).

In order to deal with the reduced yet high dimensional configuration space defined by the abstract control layer, we make use of a pre-computed Probabilistic Reaching Roadmap encoding comfort criteria. This roadmap is constructed with a carefully tailored sampling routine that efficiently explores the free regions in the configuration space and favors postures most useful for grasping.

In addition, we present several extensions applied to the problem of object manipulation: a probabilistic inverse kinematics method used to automatically propose goal postures for designed grasps, a technique to dynamically update the roadmap when obstacles change position, a method for planning transfer motions when objects are attached to the character's hands, and an automatic stepping control mechanism to enlarge the character's reachable space.

We have fully implemented the methods proposed herein as integrated interactive tools for the production of object manipulation sequences. After a preprocessing of a few minutes (to compute the required roadmaps and manipulation postures), manipulation motions are quickly synthesized. Several animation sequences of a character reaching for and manipulating objects are presented to demonstrate the effectiveness of our method.

2. Related Work

It is common sense that the use of motion captured data is the best approach to achieve realistic human-like motions for characters. Several advances have been proposed on this subject[5,6,7,8,9]. However, for motions such as object manipulation, the main concern is on the precise control and correctness of motions, and thus captured data are hard to reuse.

The key problem for object manipulation is to solve the reaching phase for a given target 6-DOF hand location. The most popular approach is to somehow solve the underlying inverse kinematics (IK) problem[9,10,11,12]. However, three main difficulties appear when devising IK algorithms. First, as the problem is under determined, additional criteria are needed in the system formulation in order to select valid and natural postures among all possible ones. Second, IK algorithms alone do not ensure that generated postures are free of collisions. Last, IK algorithms are more suitable for synthesizing postures than animations.

Computing collision-free reaching motions is in fact a motion planning problem[2]. Among the several existing methods, those based on probabilistic roadmaps[13,14,15,16] are particularly amenable to high-dimensional configuration spaces. Roadmaps can typically be computed in a preprocessing step and re-used for fast on-line querying.

Different strategies have been proposed to construct roadmaps. Visibility-based Roadmaps[16] use a visibility criterion to generate roadmaps with a small number of nodes. Rapidly-Exploring Random Trees (RRTs)[14,17] generate a tree that efficiently explores the configuration space. Because of this important property, we make use of RRTs as the growing strategy to construct our roadmap.

Other kinds of motion planners have been applied to the animation of human-like characters[3,4,18]. However, these works are limited to the control of only one arm at a time. In another direction, a posture interpolation automaton[19] was proposed, however with collision avoidance treated as a post process based on a force-field approach, which is highly sensitive to local minima.

In order to compute complete grasping and object manipulation sequences, several extensions are required. We follow the idea of predefining grasps (hand locations and shapes) for each object to be manipulated[20,21,22], and developed a probabilistic IK algorithm to automatically propose goal postures matching the predefined grasps. The IK algorithm is inspired by some approaches based on Genetic Algorithms[23,24].

Other works have also addressed the dynamic update of roadmaps[25], allowing to cope with object displacement in the workspace. We present here a similar technique except that we maintain a roadmap that is always a single connected component.

3. Method Overview and Paper Organization

A character posture is defined using a 22-DOF abstract control layer divided as follows: 9 DOFs to control each arm, 3 DOFs to control spine and torso movements, and 1 DOF to control leg flexion. Hereafter, when we speak of configurations, postures and DOFs, we are referring to configurations, postures and DOFs of the abstract control layer.

Let C be the 22-dimensional configuration space of our control layer. Let C_{free} denote the open subset of valid configurations in C. A configuration is said to be valid if the corresponding posture: is collision free, respects articulation range limits, and is balanced.

A sampling routine is responsible for generating random valid configurations in C_{free}. As the dimension of C is high, several heuristics are implemented in order to favor the generation of postures most useful for grasping. The configuration sampling routine and the abstract control layer specification are presented in Section 4.

The sampling routine is used to construct our roadmap. We first run the standard RRT algorithm[14, 17] to build a roadmap from the initial rest posture, and then apply a process which adds extra valid edges (or links) to existing nodes. Valid extra edges are added only if they represent a shorter path in the roadmap, i.e. if they represent a shortcut. The final step is to perform a proper weighting according to comfort criteria. As a result we obtain a Probabilistic Reaching Roadmap, hereafter simply referred to as roadmap. The roadmap construction process is detailed in Section 5.

Let R be a roadmap which was computed during an off line phase. Let q_c be the current character configuration and let q_g be a given valid goal configuration. A path P in C_{free} joining q_c and q_g is determined by finding the shortest path in R joining the nearest nodes of q_c and q_g in R. Path P is said to be valid if all configurations interpolated along P are valid, and in this case a final smoothing process is applied in order to obtain the final reaching motion. This entire process is described in Section 6.

As configurations have 22 DOFs, we allow designers to specify only target 6 DOFs hand locations to be reached, and a probabilistic IK algorithm automatically proposes valid goal configurations. This probabilistic IK algorithm and several other extensions useful for creating animations involving grasping and displacement of objects are presented in Section 7.

Section 8 presents and discusses obtained results, and finally Section 9 presents conclusions and future work.

4. Configuration Sampling

Configuration Definition. A complete configuration of our abstract control layer is defined by a set of 22 DOFs. Each arm is defined by 9 DOFs, five of which are devoted to the shoulder complex, the four remaining ones being equally distributed on the elbow and wrist. The character's spine, which comprises the lumbar and thoracic vertebrae, is completely determined by three DOFs, each of which controlling a unique rotational direction. Finally, 1 translational DOF controls the flexion of the legs.

We represent the arm's kinematic chain by four rotational joints: clavicle, shoulder, elbow and wrist. Except for the elbow, which is parameterized by two Euler angles (flexion and twist), we use the natural swing-and-twist decomposition defined by Grassia[26]:

$$R = R^{twist} R^{swing}, \text{ where}$$
$$R^{twist} = R_z(\theta), \text{ and } R^{swing} = \begin{bmatrix} S_x & S_y & 0 \end{bmatrix}$$

The swing motion is performed by a rotation parameterized by the above axis-angle. Note that the rotation axis for the swing always lies in the x-y plane perpendicular to the skeleton segment. The axial rotation (or twist) that follows occurs around the (arbitrarily chosen) z-axis of the local frame. In practice, the axial rotation is not used for clavicle and wrist joints and we simply set $\theta = 0$.

The one singularity of the parameterization is reached when the swing vector has norm π. Consequently, the singularity is easily avoided for the motion range of human joints by choosing an appropriate zero posture (i.e., when $S_x = S_y = 0$). For the shoulder joint for instance, we choose a reference posture in which the arm is outstretched.

We place limits on each arm joint individually. Elbow flexion, elbow twisting and shoulder twisting are limited by confining the corresponding angles to a given range. The direction of the upper arm is restricted to the interior of a spherical polygon[27]. The same kind of directional limit is applied to the clavicle and wrist joints.

The many joints in the spine are controlled by a reduced set of three DOFs, which determine respectively the total spine flexion (bending forward and backward), roll (bending sideways) and twist. While roll and twist are distributed uniformly over the spine joints, we apply the flexion mainly on the lumbar vertebrae. This distribution ensures that when the character bends forward, its back remains straight and not unnaturally hunched. This approach gives good results (see Figure 2) and is simpler than other spine coupling strategies[28].

Leg flexion is determined through the use of an IK solver. We first constrain the position and orientation of the feet to remain fixed with respect to the ground. Then we analytically compute the required rotations at the hips, knees and ankles to lower the waist according to the value of the DOF, which represents the vertical translation of the pelvis. Note that the same DOF could be used to make the character stand on tiptoes so as to reach higher.

Sampling. The sampling routine is responsible for generating random valid configurations. The random generation takes place in the 22-DOFs parameter space. For the most part, parameters are randomly generated directly within the allowed range (i.e., articulation limits). For DOFs that control a swing movement, however, directly generating parameters that respect directional limits is difficult for lack of an analytic formulation of the spherical polygon. In such cases, we simply keep iterating until the directional limits are respected (shoulder), or project the current direction onto the borders of the spherical polygon (wrist). Before accepting the random posture, we finally check if it is balanced and free of collisions.

The balance test performs a projection of the character's center of mass onto the floor, and check if it lies inside the support polygon defined as the convex hull of the feet base.

Rigid objects representing body parts are attached to the skeleton and used for collision checking. We first deactivate collision checking for all pairs of body parts that intersect in the rest posture (assumed to be valid collisions). Deactivated pairs are mainly adjacent body parts in the skeleton. After this initialization process, a collision is said to take place if any activated pair of body parts intersects, or if any body part collides with the environment. Note that rigid body parts are used for collision checking but that a regular skinning technique is used for displaying a realistic character with deformable skin. We employ the V-Collide library[29] for collision checking.

As the dimension of our configuration space is high, we bias the random distribution of configurations in order to favor postures most useful for reaching and grasping. More specifically, we distinguish two posture types:

- Regular postures have little spine motion, little leg flexion, no clavicle motion and random arm poses.
- Distant-reaching postures have large spine motion and/or large leg flexion, little elbow flexion, shoulder-clavicle coupling, arm-legs coupling, and arm-torso coupling.

In our current implementation, regular and distant-reaching postures are generated with a respective likelihood of 60% and 40%. Also, 66% of distant-reaching postures use the right hand as if the character were right-handed (see Figure 1).

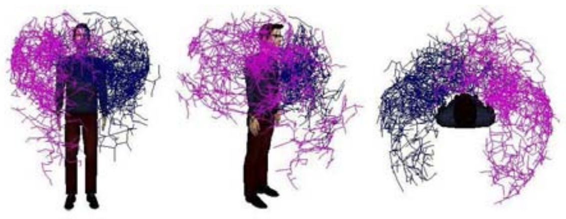


Figure 1: *Example roadmap (with little knee flexion). Each configuration in the roadmap is graphically represented with two graph nodes, which are the positions of the right (purple color) and left (blue color) wrist. Asymmetry results from the preference given to the right arm.*

Regular postures are useful when spine motion is not needed to reach a location with the hand. Note that, however, we always generate a small amount of spine motion to avoid robotic-like motions due to a completely static vertebral column.

Distant-reaching postures are suitable for remote objects that cannot be reached with arm motion only. Example postures can be seen in Figure 2. An important concept for generating distant-reaching postures is the use of couplings. These serve to favor configurations that expand the reachable space of the character. Leg flexion for instance, is authorized if one arm points downwards as if to reach for a low object. Similarly, the spine is bent so that the up-most thoracic vertebra travels in roughly the same direction as that of the arm. Finally, the clavicle is moved in such a way that another few centimeters are gained toward the imaginary location the arm points at.

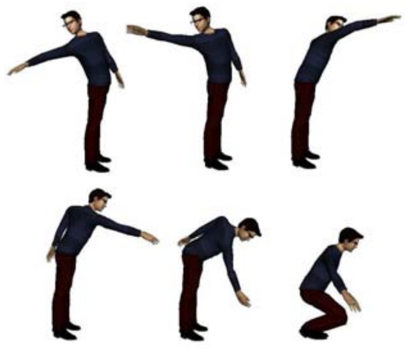


Figure 2: *Examples of distant-reaching postures.*

Instead of computing two separate roadmaps for the right and left arms, we generate a single roadmap encoding motions of both arms. Besides reducing memory consumption and storage costs for large roadmaps, we guarantee that while reaching with one arm, possible motions of the spine will not induce collisions with the other arm. A further benefit is that our roadmap can also handle multi-hand reaching motions.

5. Roadmap Computation

The roadmap construction relies mainly on three functions: the sampling routine, the distance function and the interpolation function. We now describe the latter two.

Distance function. Good results are usually obtained with distance functions based on the sum of the Euclidean distances between corresponding vertices lying in the shape of the articulated structure[2, 7]. We use a similar yet simplified approach based on a selected set of articulations: the bottom-most lumbar vertebra, the top-most thoracic vertebra, the articulations of both arms (shoulder, elbow and wrist), and finally the thumb and pinky base joints to capture arm twisting and wrist rotations. Let q_1 and q_2 be two configurations. Our distance function $dist(q_1, q_2)$ returns the average sum of the Euclidean distances between the corresponding selected articulations at configurations q_1 and q_2.

The primary advantage of our distance function is its speed. Another important property is that it remains independent of both the skin deformation module and the vertices density distribution in the skin mesh.

Interpolation function. The interpolation function $interp(q_1,q_2,t)$ returns, for each $t \in [0,1]$ a configuration varying from q_1 $(t=0)$ to q_2 $(t=1)$. The interpolation function applies spherical linear interpolation between corresponding joints, except for the translational joint controlling leg flexion and the elbow, which is parameterized with Euler angles. For these joints, linear interpolation is applied.

The interpolation is said to be valid if, for all values of $t \in [0,1]$, the $interp$ function returns a valid configuration. The implementation of the interpolation validity test is approximate: An interpolation is considered valid if n equally spaced interpolated configurations between $t=0$ and $t=1$ are valid. The number n trades computation precision for speed, and its value is also adjusted according to the distance between the two configurations.

Roadmap growing process. The roadmap construction starts with the tree growing process of the RRT algorithm[14, 17]. The initial posture is the character's rest posture, i.e. standing straight with arms lying by the side. This rest posture is well suited for generating a tree with nearly uniform branch depth.

In the RRT algorithm a random configuration q_{rand} is used as a growing direction. The nearest configuration q_{near} in the current tree is determined and a new configuration q_{new} is computed as:

$$q_{new} = interp(q_{near}, q_{rand}, t), \text{ where}$$
$$t = \varepsilon/d,\ d = dist(q_{near}, q_{rand})$$

If q_{new} is valid and the interpolated path to q_{near} is also valid, q_{new} is linked to q_{near}, making the tree grow by one node and one edge. The new edge is assigned the cost ε. The factor ε represents the roadmap edge length, i.e. the incremental step by which the tree is grown. Large steps make the roadmap grow quickly but with more difficulty to capture the free configuration space around obstacles. Inversely, too small values generate roadmaps with too many nodes, thus slowing algorithms down. Good values for ε mainly depend on the complexity of the environment.

The tree generation process runs until a specified number of nodes is reached. In our experiments, we worked with graphs made up of around 1500 nodes.

Shortcuts. Once the tree is constructed we transform it into a graph with the process of shortcuts creation: for each pair of leaf nodes (l_1,l_2) in the tree, an edge linking l_1 to l_2 is added to the roadmap if:

- the interpolation between l_1 and l_2 is valid,
- $dist(l_1,l_2)$ < shortest path in the tree joining l_1 and l_2,
- $dist(l_1,l_2) < r$.

Shortest paths are easily determined by running an A^* algorithm[2] over the roadmap, taking into account the costs associated to the roadmap edges. The radius r is a parameter that specifies a limit on the length of shortcuts created and serves also to control the total number of shortcuts added to the roadmap.

Figure 3 illustrates the effect of adding shortcuts in the simpler 2-dimensional configuration space problem. In this example, the square is the sole obstacle and the root of the tree is on the left side of the square.

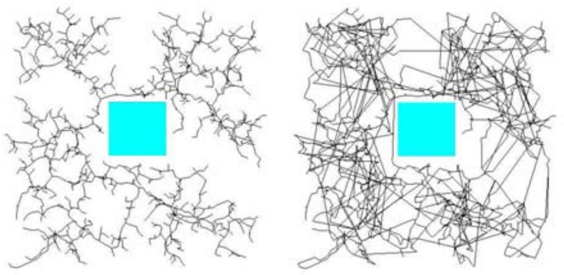


Figure 3: *The roadmap before (left image) and after (right image) the insertion of shortcuts.*

Single arm costs. The roadmap constructed so far encodes in each edge a cost defined as the distance between the two configurations linked by the edge. These configurations contain random positions for both arms of the character. As we use the same roadmap to determine single-arm motions as well, we also store in each edge of the roadmap two additional costs. The right arm motion cost is calculated with a distance function that simply does not take into consideration the joints in the left arm. Conversely, the left arm motion cost is determined by ignoring joints in the right arm.

Weights. In the final stage, we assign to each roadmap edge a proper weight to favor the determination of paths with better comfort characteristics. Different heuristics can be used to determine such weights. We use the central idea of favoring motions passing near the rest posture. We first compute the distance between the rest posture and the midpoint of each roadmap edge. Each edge is assigned this distance scaled to the interval $[k,1], k \in [0,1)$. Parameter k gives the amount of influence of the weighting and is controlled through the user interface. The weighting scheme helps, among others, to generate motions with the arm closer to the body.

6. Roadmap Querying

Let R be a roadmap computed during an off line phase, as described in the previous section. Let q_c be the current (valid) character configuration and let q_g be a given valid goal configuration to reach. Note that q_c and q_g are not necessarily contained in R (and in fact, normally they are not). The desired reaching motion is obtained by determining a valid path in C_{free} having q_c and q_g as endpoints. The path is determined in two phases: path finding and path smoothing.

Path finding. We first find $N(q_c) \in R$ and $N(q_g) \in R$, which are respectively the nearest nodes to q_c and q_g in R. It is required that the interpolation between $N(q_c)$ and q_c and the interpolation between $N(q_g)$ and q_g are valid. If not, it means that the graph has not grown sufficiently, or that the goal configuration is not reachable.

Then, the correct cost in the roadmap is activated according to the desired grasping type (right, left or both hands), and the shortest path in R joining $N(q_c)$ and $N(q_g)$ is found. Note that R ensures that the shortest path is valid. Valid paths are represented as a sequence of configurations, where the interpolation of each pair of consecutive configurations is valid. The final reaching path P is obtained by adding to the shortest path configurations q_c and q_g as end nodes.

Path Smoothing. Because of the random nature of the nodes in R, P normally does not represent a useful motion and a smoothing process is required. We basically smooth P by incremental linearization.

Let q_1, q_2 and q_3 be a corner of P, i.e three consecutive configurations in P, and let $q = interp(q_1, q_3, t)$ where t is set according to the relative distances of the three configurations. If the interpolation between q_1 and q, and between q and q_3 are both valid, a local smooth can be applied and q_2 is replaced with q.

Our smoothing algorithm always selects the corner of P that deviates most from a “straight line”. The distance between q_2 and q gives us a measure of the deviation. This process quickly results in a smooth path. After a while, it also tends to bring the path closer to obstacles.

We propose additional operations, which are applied during the basic smoothing process:

- Whenever two consecutive configurations in P get too close, they get merged into a single one; conversely, if they are too distant, a new configuration is inserted in-between by interpolation with $t = 0.5$.
- At every k steps during the iterative local smoothing process, we try to apply a group smoothing: two configurations in P are randomly selected and, if their interpolation is valid, all nodes between them are removed from P (note that a re-sampling may occur due to the operation described in the previous item). This procedure greatly accelerates the process in many cases, and even permits to escape from local minima. In our experiments we have used k as the number of nodes in P. Finally, we also obtained significant speed-ups by applying group smoothing hierarchically before entering the iterative loop: from both endpoints in P, we perform a recursive binary partition until pairs are smoothed or until consecutive pairs are reached.
- Last but not least, when the application of one of the smoothing procedures fails due to non-valid interpolation between configurations, we test again the same interpolation on different combinations of groups of DOFs (and not on all DOFs at the same time). Groups of DOFs are defined as: left arm, right arm, spine and legs. This process keeps smoothing for instance the motion of one arm when spine motion cannot be smoothed anymore because of obstacles.

7. Extensions for Object Manipulation and Grasping

Probabilistic IK. It is not an easy task for the artist to specify a realistic 22-DOFs goal configuration q_g (used as input for roadmap querying). To overcome this difficulty we allow designers to simply place a three-dimensional model of a hand anywhere in the workspace, and run a probabilistic IK algorithm to automatically propose goal configurations matching the specified 6-DOFs hand location. A probabilistic approach is effective because we already have nodes in the roadmap with the hand close to the required posture. In addition, our framework enables us to easily generate collision-free postures. In contrast, Jacobian-based methods exhibit better convergence but cannot guarantee collision-free postures.

Our method is inspired by some Genetic Algorithms implementations[23, 24]. Let H be a target hand location. We first select the k closest configurations q_i, $i \in 1,\ldots,k$ in the roadmap, according to a distance function that only considers the distance between H and H_i, where H_i is the same object H, but placed at the hand location specified by configuration q_i. The distance function takes the average sum of the Euclidean distances between each corresponding pair of vertices in H_i and H.

Configurations q_i, $i \in 1,\ldots,k$ constitute the initial population that converges towards H by minimizing the distance function. Usually the configurations in the initial population are already very close to H, and thus, instead of developing all usual operators of a Genetic Algorithm approach, we obtain satisfactory results with simple and faster strategies (see Figure 4).

We use a variation of the roadmap growing procedure. Random configurations q_{rand} are generated, and if the incremental interpolation from q_i towards q_{rand} gives a valid and closer configuration to H, q_i is replaced by the incremented version. However, if H is located close to the limits of the reachable workspace, the convergence of this method may become problematic. In such cases, we adopt a different strategy and apply perturbations on random DOFs of every q_i. A perturbed configuration is kept only if it is valid and closer to H. In both methods, backtracking is applied when a local minimum is reached.

Grasping. We follow the usual approach of having predesigned hand shapes for every object to be grasped[20, 21, 22]. A complete grasping sequence results from the concatenation of any number of reaching motions and a final grasping.

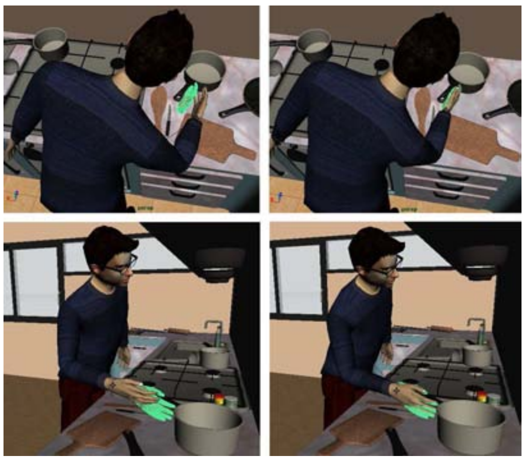


Figure 4: *The closest configuration in the roadmap in relation to the target hand (left column), and after the probabilistic IK process (right column).*

The final grasping is generated like a reaching motion except that it moreover includes the interpolation of finger joint angles towards target angles defined in the grasping hand shape. An additional finger-object collision test can be used to ensure perfect grasping as well as to diminish the required design precision of hand shapes.

Dynamic roadmap update. Each time an object in the workspace moves, the roadmap needs to be updated accordingly. The approach is to detect and remove the nodes and edges that become invalid after a change in the workspace. As a result, disconnected roadmap components may appear. Instead of managing disconnect parts[25], we follow the philosophy of keeping a single connected roadmap that represents the reachable configuration space C_{free} at all times. Whenever an obstacle in the workspace is inserted, removed, or displaced, a global roadmap validation routine is performed in three steps:

- All invalid edges and nodes are removed. Disconnected components may appear.
- We try to connect each pair of disconnected components by adding valid links joining the k closest pairs of nodes in each component (in our experiments, $k = 0.2n$ where n is the number of nodes in the smaller component of a pair). If disconnected components still linger, we simply keep the largest component.
- Due to the operations described above, the roadmap may no longer cover C_{free} very well. "Holes" in the coverage of C_{free} are likely to appear because some regions become free due to an obstacle displacement, and because of removed components in the roadmap. Hence, the roadmap is grown again (see Figure 5) as described in Section 5. Note that the sampling routine can easily be biased to generate postures only in the parts of C_{free} that are insufficiently covered.

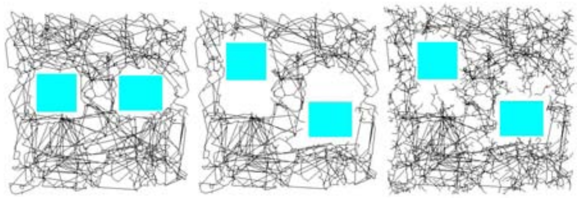


Figure 5: *Left: original roadmap. Middle: roadmap exhibiting "holes" due to obstacles displacements. Right: roadmap is grown again.*

Transfer paths. We call a transfer path a motion enabling the character to move an object from one place to another.

The main difficulty of computing transfer paths is that roadmap nodes can no longer be guaranteed to be valid because of objects attached to the character's hand(s). One first option is to pre-compute specific roadmaps for each object that needs to be carried by the character. This solution could be used, for instance, to plan motions for a character with a sword in its hand.

We developed an alternative method that returns transfer paths on the fly. We first compute a reaching path P between the first and the last posture of the desired transfer path, without considering the object being transferred. Then the object is attached to the character's hand and P is checked for validity, this time taking the attached object into account. If the object is small, P may still be valid and directly useful as a transfer path. If not, the invalid nodes and edges of P are removed, and disconnected nodes in P are reconnected at run time using a standard single query RRT[17].

The performance of this method greatly depends on the complexity of the transfer path to be computed, ranging from extremely fast in simple cases to extremely slow in complex transfer cases.

After a transfer path has been computed, the displaced object needs to be removed from the roadmap and inserted again at its new position. This must be done in both methods, i.e. when using additional pre-computed roadmaps or when planning transfer motions on the fly.

Stepping control. Although we control the entire body of the character when generating motions, we are still limited to grasping sequences with the feet fixed on the floor. In some cases, much more realistic results are achieved if the character takes some steps. Typically, steps are used for obtaining better balance and for reaching distant objects.

We developed a multi state approach to let the character step. As a pre-process, we create k short stepping animations with different lengths and directions. Each of these animations transfers the character's rest posture p_r into the final posture p_i of the stepping animation i, $i \in 1, \ldots, k$. Each final posture is considered to be a valid state if the animation linking p_r and p_i can be played without collisions. Furthermore, adjacent states are also linked with pre-defined stepping animations if these do not incur collisions.

For each valid p_i, a roadmap R_i is computed considering p_i as the start node of the roadmap generation. In the end, several roadmaps co-exist, the original reaching roadmap R being centered at p_r while adjacent roadmaps R_i are centered at various positions p_i around R (see Figure 6).

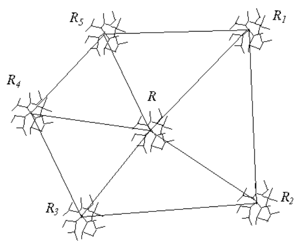


Figure 6: *Distinct roadmaps are generated and connected with predefined stepping animations.*

Let q_i and q_g be the initial and goal configurations of a reaching animation to be determined with stepping control. We first detect the closest configurations to q_i and q_g in any of the roadmaps, determining the initial and goal roadmaps to be used (R_i and R_g respectively). Then we determine path P_1 joining q_i and p_i in R_i, and path P_2 joining p_g and q_g in R_g. The final path P is obtained by concatenating P_1, the shortest sequence of animations joining p_i and p_g, and finally P_2.

A final smoothing process is applied over P, taking into account only the motions of the upper body limbs. Further explanations and examples are omitted for lack of space.

8. Analysis and Results

Results. Figure 7 presents animation stills of a virtual character reaching for objects in a fridge, and as well an example of object relocation. In these examples and others shown in the videos accompanying this article, we have grown roadmaps using an incremental distance ε of 4 cm until reaching 1500 nodes. In each example, the design work was limited to the definition of a few goal postures. Then, the motions were automatically generated by the planner without any user intervention. The only exception is the head orientation, which was specified by hand in some sequences.

All motions were produced with constant velocity along planned paths. The timing could easily be improved by the designer or automatically adjusted e.g. according to Fitts' law[30]. Note also that the left arm is not animated while the right hand is reaching, which creates a somewhat stiff look in some postures. Additional controllers need to be integrated in order to correct this, for instance simulating dynamics over that arm.

The method works extremely well providing that enough free space exists between obstacles. The roadmap encounters some difficulty to explore portions of the workspace containing many obstacles, especially when these are situated on the borders of the character's reachable space. This is exemplified by the refrigerator sequence. Even for small values of ε, postures where the hand reaches inside the refrigerator are not present in the first computed roadmap. Our probabilistic IK routine elegantly solves this problem. The designer simply specifies 6-DOF hand postures within the refrigerator and thus forces the roadmap to grow inside the refrigerator.

The encoded comfort criteria help to favor natural looking movements. However, the simple criterion currently used cannot capture all the subtleties of human movement. More complicated comfort criteria e.g. from biomechanics should be introduced. It is also important to note that the efficacy of the weighting scheme is restricted to cases where several paths exist.

Performance. With roadmaps consisting of around 1500 nodes, shortest paths are instantaneously determined (a few milliseconds) with an A^* algorithm. Construction times are listed in Table 1. In order to accelerate the computation of the distance function, joint positions relative to configurations in the roadmap are cached.

The smoothing phase gives good results in less than a second. Actually, in the presented results, we stopped the smoothing process when the time limit of 1 second was reached. It is important to mention that the group smoothing strategy tremendously accelerates the process.

The convergence of the probabilistic IK procedure greatly depends on the distances between the nodes in the initial population and the target hand posture. In our examples, we had cases ranging from a few seconds to a few minutes. Transfer paths calculated in relatively clear spaces, such as the example in Figure 7(b), could be computed in approximately 1 second.

The results and times reported in this paper were obtained with tools developed in a Maya plug-in, running on a Pentium PC at 1.7 GHz.

9. Conclusions

Contribution. This work makes a number of contributions that allow to plan grasping motions for a 22-DOF humanlike character in interactive applications. More specifically, our main contributions are:

- A 22 DOFs abstract control layer that poses the entire body of the character: arms, shoulders, torso, spine and leg flexion.

Scene	Number Triangles	Roadmap Computation	Shortcuts Computation	Shortcuts Created
No Obstacles	0	70	6	565
Only Body Parts	10153	78	7	559
Cubes	10249	78	11	508
Kitchen	26088	145	18	486

Table 1: *Performance measurements. The second column gives the number of triangles considered for collision detection. The third and fourth columns give computation times in seconds. The last column lists the number of shortcuts created using the maximum shortcut length of 16 cm.*

- A biased sampling method that efficiently covers most-used parts of the free configuration space with random human-like grasping postures.
- A new roadmap structure: the probabilistic reaching roadmap, which is dense in connections between nodes and encodes comfort criteria.
- Several extensions for generating complete object manipulation sequences.

Future Work. We believe that the techniques presented herein open several new research directions, and show that motion planning can greatly benefit computer animation.

The notion of balance can be extended to take into account supports when hands or other body parts get in contact with objects. Motion constraints can be added to allow the displacement of objects with fixed orientation, e.g. like a glass of water. Additional DOFs can be included to control movements that are coordinated with objects, e.g. to control chair translation (when sitting), or to plan motions like opening a drawer or pressing a button.

Finally, motion capture data could improve the personality of movements and could be included at two levels: in the posture sampling routine during the generation of the roadmap[7], and during the smoothing process by adding motion texturing[8].

Acknowledgements

We express our thanks to Alias/Wavefront for the granted Maya SDK educational licence. This research has been partially funded by the Federal Office for Education and Science in the framework of the European project STAR (Service and Training through Augmented Reality), IST-2000-28764.

References

1. N. I. Badler, C. B. Phillips, and B. L. Webber. Simulating Humans: Computer Graphics, Animation and Control. ISBN 0-19-507359-2, 1993.

2. J.-C. Latombe. Robot Motion Planning. ISBN 0-7923-9206-X, Kluwer Academic Publishers, 1991.

3. Y. Koga, K. Kondo, J. Kuffner, and J. Latombe. Planning Motions with Intentions. *Proc. of SIGGRAPH'94*, 395-408, 1994.

4. J.J. Kuffner and J.C. Latombe. Interactive manipulation planning for animated characters. *Proc. of Pacific Graphics'00*, poster paper, Hong Kong, October 2000.

5. A. Witkin and Z. Popovic. Motion Warping. *Proc. of SIGGRAPH'95*, 1995.

6. M. Gleicher. Retargeting Motion to New Characters. *Proc. of SIGGRAPH'98*, 1998.

7. L. Kovar, M. Gleicher, and F. Pighin. Motion Graphs. *Proc. of SIGGRAPH'02*, 2002.

8. Y. Li, T. Wang, and H.-Y. Shum. Motion Texture: A Two-Level Statistical Model for Character Motion Synthesis. *Proc. of SIGGRAPH'02*, 2002.

9. C. F. Rose, P.-P. J. Sloan, and M. F. Cohen. Artist-Directed Inverse-Kinematics Using Radial Basis Function Interpolation. *Proc. of Eurographics*, **20**(3), 2001.

10. D. Tolani, and N. Badler. Real-Time Inverse Kinematics of the Human Arm. *Presence*, **5**(4):393-401, 1996.

11. P. Baerlocher, and R. Boulic. Task-priority Formulations for the Kinematic Control of Highly Redundant Articulated Structures. *Proc. of IROS'98*, Victoria, Canada, Oct. 1998.

12. X. Wang, and J.-P. Verriest. A Geometric Algorithm to Predict the Arm Reach Posture for Computer-aided Ergonomic Evaluation. *Journal of Vis. and Comp. Animation*, **9**(1):33-47, 1998.

13. L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic Roadmaps for Fast Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, **12**:566-580, 1996.

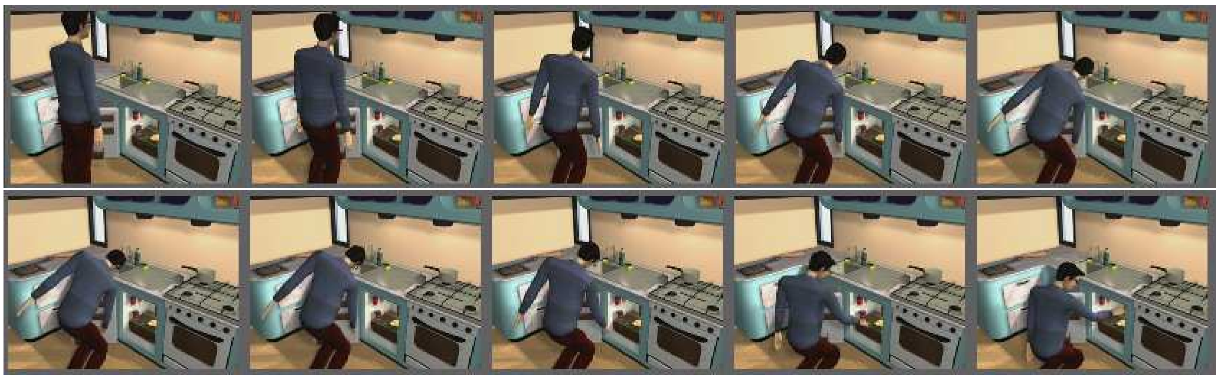
14. S. La Valle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. *Technical Report* 98-11, Computer Science Dept., Iowa State University, Oct. 1998.

15. R. Bohlin and L. Kavraki. Path Planning using Lazy PRM. In Proc. of IEEE Int. *Conference on Robotics and Automation*, ICRA, 2000.

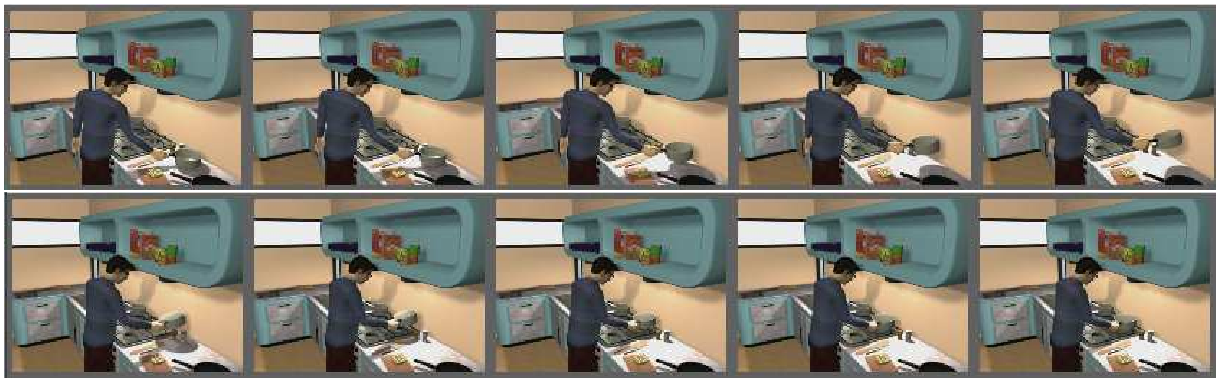
16. T. Simeon, J. P. Laumond, and C. Nissoux. Visibility Based Probabilistic Roadmaps for Motion Planning. *Advanced Robotics Journal*, **14**(2), 2000.

17. J.J. Kuffner and S.M. La Valle. RRT-Connect: An efficient approach to single-query path planning. *In Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'2000)*, San Francisco, CA, April 2000.

18. S. Bandi and D. Thalmann. Path Finding for Human Motion In Virtual Environments. *Computational Geometry* **15**(1-3):103-127, 2000.



(a)



(b)

Figure 7: *Animation sequences within a kitchen scenario. In the first example (a), leg flexion is used for reaching two specified locations in the fridge: at the door, and inside it. The second example (b) shows the object relocation of a saucepan.*

19. R. Bindiganavale, J. Granieri, S. Wei, X. Zhao, and N. Badler. Posture interpolation with collision avoidance. *Computer Animation '94*, Geneva, Switzerland, 1994.

20. Y. Aydin, and M. Nakajima. Database guided computer animation of human grasping using forward and inverse kinematics. *Computer & Graphics*, **23**:145-154, 1999.

21. H. Rijpkema and M. Girard. Computer Animation of Knowledge-Based Human Grasping. *Proc. of SIGGRAPH'91*, 339-348, 1991.

22. M. Kallmann. Object Interaction in Real-Time Virtual Environments. DSc Thesis 2347, EPFL, January 2001.

23. A. A. Khwaja, M. O. Rahman, and M.G. Wagner. Inverse Kinematics of Arbitrary Robotic Manipulators using Genetic Algorithms. J. Lenarcic and M. L. Justy, editors, *Advances in Robot Kinematics: Analysis and Control*, 375-382. Kluwer Academic Publishers, 1998.

24. M.-H. Lavoie, and R. Boudreau. Obstacle Avoidance for Redundant Manipulators Using a Genetic Algorithm. *CCToMM Symp. on Mechan., Machines, and Mechatronics*, Canada, 2001.

25. T.-Y. Li, and Y.-C. Shie. An Incremental Learning Approach to Motion Planning with Roadmap Management. *Proc. of International Conference on Robotics and Automation (ICRA)*, 2002.

26. S. Grassia. Practical Parametrization of Rotations Using the Exponential Map. *Journal of Graphics Tools*, **3**(3):29-48, 1998.

27. J. U. Korein. A Geometric Investigation of Reach. The MIT Press, Cambridge, 1985.

28. G. Monheit and N. Badler. A Kinematic Model of the Human Spine and Torso. *IEEE Computer Graphics and Applications*, **11**(2):29-38, 1991.

29. S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. *Proc. of ACM SIGGRAPH*, 171-180, 1996.

30. P. Fitts. The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology*, **47**:381-391, 1954.