Help note for assignment 2

Steve Lin, Nov 2013

This help note describes the mathematical definitions and representations of the ASF/AMC format in the viewer distributed in the computer animation class. Implementation issues for motion concatenation, such as motion transformation, motion matching, and motion blending are also briefly discussed. Some inverse kinematics algorithms are also discussed.

I. Notations

Notations

V_i	vector represented in coordinate frame i (the *i*th bone)
$\hat{V}_i$	unit vector
${}^i_j R$	rotation matrix that transforms a vector V_i from coordinate frame *i* to coordinate frame *j*, i.e., $V_j = {}^i_j R V_i$
${}^i_0 R$	rotation matrix that transforms a local vector to a global vector
${}_i T$	global position of the *i*th bone (at the proximal end)
${}_0 T[n]$	root position at the *n*th frame in a motion
${}_0 R_{amc}[n]$	root orientation at the *n*th frame in a motion
R_{ij}	matrix component at the *i*th row and *j*th column

II. Skeleton definition

The skeleton model used to describe an articulated figure is defined based on the Acclaim file format with some modifications. We use a tree structure to represent the hierarchy of the bones in the skeleton. Each node in the tree represents a bone in the skeleton. Table 1 lists each field's definition in the bone structure.

Field name	Description	Notation
Bone *sibling Bone *child Bone *parent	Pointers to the sibling (branch bone), child, and parent of the current bone.	
int idx	Array index in the bone array; the original "id" in ASF file is ignored. If a unique id for each bone is needed, an additional boneID field should be added.	
vector dir	Unit vector describes the direction from local origin to the origin of the child bone. Note: stored in local coordinate system of the bone	$\hat{V}_i$
float length	Bone length	l
vector axis	Correspond to the axis field in ASF file, which specifies the orientation of local coordinate frame at the neutral pose. Note: represented as Euler angle with respect to the glocal coordinate.	
Double rot_parent_current [4][4]	Rotation matrix from the local coordinate of this bone to the local coordinate system of its parent. This matrix is computed from the axis information above, and stored in its transposed version to match OpenGL's convention.	${}_i R_{asf}$
vector dr	Delta rotation angles (joint angles) for this bone at a particular time frame (as read from AMC file) in local coordinate system.	${}_i R_{amc}$
float aspx, aspy	Aspect ratio of bone shape	
Int dof	Number of degrees of freedom of the bone	
int dofx, dofy, dofz	Degree of freedom mask in x, y, z axis. dofx=1 if this bone has a degree of freedom in x-axis, otherwise dofx=0.	

Table 1 Definition of bone structure

All information in the bone structure is represented in local coordinate. The global orientation of the local coordinate frame (${}_0^i R$) and its global location (${}_i T$) of the *i*th bone are computed as follows

$$ {}^{i+1}_{i}R = {}_{i}R_{asf} \cdot {}_{i}R_{amc} \tag{1.a} $$

$$ {}^{i}_{0}R = {}^{1}_{0}R\,{}^{2}_{1}R \cdots {}^{i}_{i-1}R \tag{1.b} $$

$$ V_i = \hat{V}_i \cdot l_i \tag{2.a} $$

$$ {}_{i}T = {}^{i-1}_{0}R V_{i-1} + {}_{i-1}T \tag{2.b} $$

where ${}_{i}R_{asf}$ and ${}_{i}R_{amc}$ are obtained from the Acclaim file. We'll show you how to compute these two matrices from the Acclaim file in the following paragraphs.

Acclaim file reader

The Acclaim file[1] includes two parts: ASF file and AMC file. ASF file describes the skeletal information of the figure and the configuration of the figure's neutral pose. AMC file describes the joint angle changes in a motion. All information in ASF file is specified with respect to the global coordinate, while all information in AMC file is specified with respect to the local coordinate.

There're three sections in the ASF file: root section, bone section, and hierarchy section. These sections begins with ":root", ":bonedata", and ":hierarchy" lines in the file, respectively. Moeditor currently ignores the information in root section and assumes the data order of the root bone is TX TY TZ RX RY RZ in AMC file. The hierarchy section indicates how the bones are connected. All information needed for loading a skeleton is mainly specified in the bone section, which contains the data of bone name, bone id, bone length, degree of freedom, joint limits, global orientation and direction vector to outboard (children) bone at the neutral pose. Because the orientation and direction vector is specified with respect to the global coordinate, we have to convert them to the local coordinate.

A bone's global orientation is specified in the "axis" field. The first three components are the Euler angle and the fourth component is its order. If the Euler angle is in XYZ order, the axis field gives us a rotation matrix

$$ {}^{i}_{0}R = {}^{i}_{0}R_z \cdot {}^{i}_{0}R_y \cdot {}^{i}_{0}R_x \tag{3} $$

Given the orientations of bone *i* and bone *i+1*, we can compute the rotation matrix that transform vector from coordinate fram i+1 to i

$$ {}^{i+1}_{i}R = {}^{0}_{i}R \cdot {}^{i+1}_{0}R \tag{4} $$

[1] We'll just explain the fields we need for loading the ASF skeleton here. For the details, please see the Acclaim file format description in http://www.cs.cmu.edu/~kiranb/animation/StartupCodeDescription.htm.

From Eq 1.a, we know that the matrix ${}^{i+1}_{i}R$ in Eq. 4 equals to ${}_{i}R_{asf}$ (stored in the *i*th bone), because ${}_{i}R_{amc}$ is an identity matrix for a neutral pose. The bone direction vector is converted by

$$\hat{V}_i = {}^{0}_{i}R\hat{V}_0 \tag{5}$$

where $\hat{V}_0$ is the unit vector specified in the "direction" field in ASF file.

Loading the AMC file is straightforward once the ASF skeleton is acquired. We just read in the Euler angles of every joint in each frame and store them in the dr field of the Bone structure. The displayer or other motion class functions can be applied to these data.

Currently, there are still some limitations about the Acclaim file reader:

1. Information ignored: root section, and the scale, joint limit, Euler angle sequence, bone ID fields in bone section.

2. The Euler angle order should be in XYZ sequence.

BVH to Acclaim converter

The conversion is done by the following steps,

1. Convert BVH to Acclaim file format using the program, bvh2asfamc.exe, developed by Biomechanics, Inc.

2. Convert the Euler angle order in AMC file from YXZ to XYZ and change dof field in ASF file of each bone from YXZ to XYZ.

To convert the Euler angle from YXZ order to XYZ order, we first form the rotation matrix by post multiplying R_z by R_x and R_y, then extract the angles from the matrix by comparing it with a symbolic rotation matrix formed by $R_z R_y R_x$. A more detailed description can be seen in YXZEulerAngle2EulerAngle() in mathclass library.

III. Motion connecting

The purpose of the motion connection is to connect two motion clips seamlessly. For this goal, we have to generate the transition motion between them. We can generate the transition motion in two ways. The first one is generating from scratch, i.e., utilizing the physics property of the motion to generate the transition. The other way is to search in the motions to be connected, and find the best entry frame to connect them. Although the first approach preserves the original motion, it is not easy to accomplish when two ending postures differ a lot. We adopt the second approach in the implementation.

After finding the best matched frames in the motions, we can connect them either by inserting interpolated motion between ending postures or by creating an overlapped window centered at the best matched frame and blending those frames inside the window. We'll describe the motion transformation, motion matching, motion interpolation and motion blending modules first, and then show how to use these modules to connect two motions in the following paragraphs.

Motion transformation

This module transforms the whole motion clip into a new starting position and facing direction. It applies a 1D rotation and 2D translation on the character's root bone. This help users to connect two mocap files because the character's end position and direction in one mocap file is usually not the same as those in another. Figure 1 shows the procedures in the motion transformation.

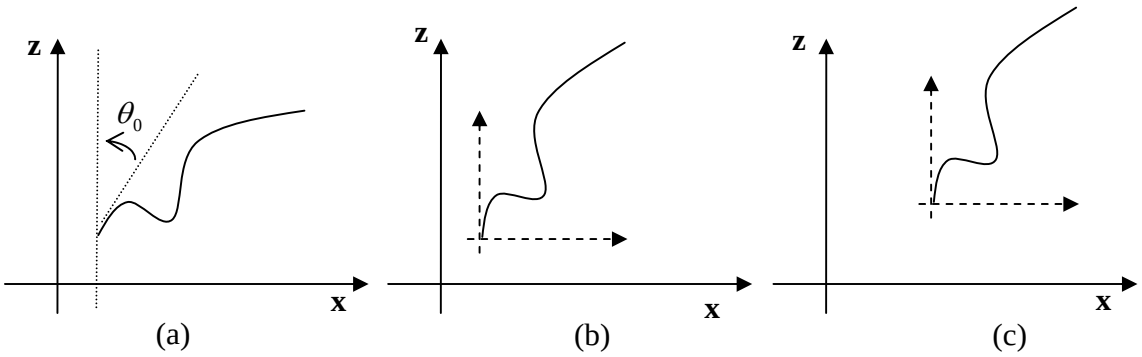

Figure 1 Motion transformation procedures. The curve on the x-z plane represents the projection of the root trajectory. (a) Facing angle computation. (b) Rotate the motion about the root position at the first frame. (c) Translate to the new position.

1. Compute the character's facing direction at the beginning frame

The global coordinate frame and the local coordinate frame at the root bone are aligned at the neutral pose (x—left-handed, y—up, z—forward.) When the character moves, the orientation of the root's local frame would change as well. We can project the local frame's z-axis on the global x-z plane, and measure its angle with respect to the global z-

axis. Thus we can obtain a global facing direction of the character. The mathematical operation is done as follows. First we compute the global coordinate of the local coordinate axis by[2]

$$R = {}_0R_{asf} \cdot {}_0R_{amc} = {}_0R_{amc} \tag{6}$$

Extract the third column from R and project it onto the x-z plane. The facing angle is obtained by

$$\theta_0 = \tan^{-1}(\frac{R_{02}}{R_{22}}) \tag{7}$$

2. Rotate character's facing direction in all frames about the root position of the first frame.

This includes adjusting the root's orientation and position. Let θ_0 and $T_0[0]$ be the character's facing direction and position at the first frame, and we want to transform the character's facing direction to θ_d. The root bone's orientation $R_d[n]$ at the *n*th frame can be computed as follows

$$R_d[n] = R_y(\theta_d - \theta_0) \cdot {}_0R_{amc}[n] \tag{8}$$

The new AMC angle is extracted from $R_d[n]$. To adjust the root position, we need to get its displacement vector with respect to the first frame, and rotate this vector as well.

$$T_d[n] = R_y(\theta_d - \theta_0)({}_0T[n] - {}_0T[0]) \tag{9}$$

3. Translate all frames to new starting position T_{d0} by

$$T_d[n] = T_d[n] + T_{d0} \tag{10}$$

Motion matching

Given a short motion clip, motion matching module finds a best matched portion in another motion. This suggests the user a good entry point to connect two motions. The window distance is defined as follows

$$dist = \sum W_{frame}[f] \sum W_{joint}[j] \cdot joint\_dist(q_j[f] - q_j'[f]) \tag{11}$$

where joint_dist() measures the difference at joint j. This metric may be defined on Euler angle space, quaternion space or normalized joint position space. W_{frame} and W_{joint} are

[2] ${}_0R_{asf}$ is an identity matrix at the root bone.

the weights of frame window and joint window. The window distance we use now is Euclidean distance computed on Euler angles at all joints except the root joint. Figure 2 shows this matching process.

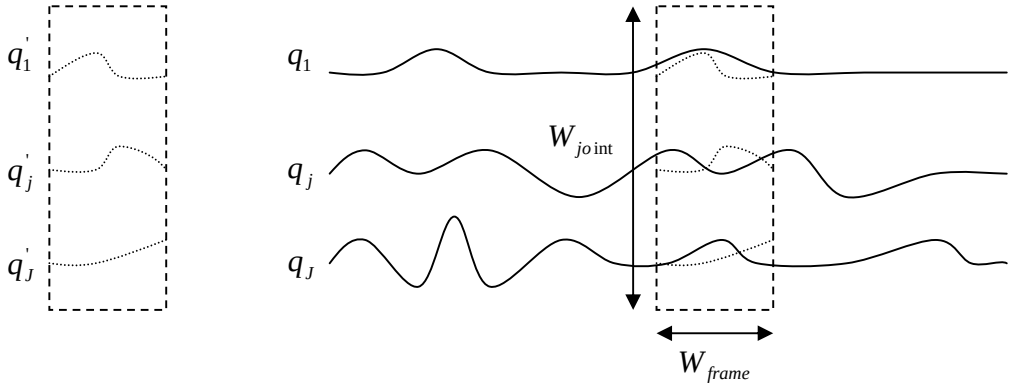

Figure 2 Window distance matching

For the purpose of motion connecting, we support two types of matching—the forward matching and backward matching. In forward matching, we pick a portion at the end of the first motion, and find the best matched portion in the second motion. For the backward matching, we pick a portion in the beginning frames of the second motion, and find the best matched portion in the first one.

Motion interpolation

This module generates interpolated motion between the end posture of one motions and the beginning posture of another. For root position data, we can just apply linear interpolation to them because they are in 3D Euclidean space. As for the joint angle data, we do the interpolation in quaternion space because it's a better representation for 3D rotation. Since the joint angles in AMC files are represented in Euler angle, we need to convert them into quaternion. This conversion is done in two steps: convert Euler angle into a rotation matrix and then convert the matrix to quaternion. Once we convert all joint angles into quaternion, we can apply the spherical linear interpolation (SLERP) to generate in-between joint angles—the interpolation result has to be converted to Euler angle again, of course. The equation for SLERP is listed as follows

$$Slerp(q_1, q_2; u) = \frac{\sin(1-u)\theta}{\sin\theta} q_1 + \frac{\sin(u\theta)}{\sin\theta} q_2 \qquad (12)$$

A big difference for interpolation in the quaternion space from that in 3D Euclidean space is that we do line interpolation along an arc on a unit-sphere in 4D space instead of a straight line on 3D space. Another difference is that $\pm q$ represents the same rotation. We have to make sure q_1 and q_2 form a shorter path on the arc when doing interpolation. For further reference about quaternion, please see [1].

Motion blending

This module blends two motions by summing two postures through all frames. It allows users to determine the weighting for each motion. The underlying technique for blending is similar to motion interpolation. Both of them generate interpolated postures between two key postures. In motion interpolation, there is only one pair of key postures, the end and starting postures of two motions; in motion blending, however, there are multiple pairs of key postures. Each pair corresponds to a frame in the motion sequence, i.e., if we want to blend two motions of 50 frames, we have 50 pairs of key postures.

For ease in/ease out blending, the blending weights are set as follows

$$W_b[f] = \frac{1}{2}\sin(\frac{f}{N_b - 1}\pi - \frac{\pi}{2}) + \frac{1}{2}, \qquad f = 0, \cdots, N_b - 1 \tag{13}$$

where N_b is the length of the blending window.

Connecting two motions by interpolation

Figure 3 illustrates the procedures to connect two motions by interpolation—in forward matching case. We first find the best entry frame for connection in the second motion and then remove the frames inside and before the matching window. The reason to remove these frames is because that portion is "pseudo-overlapped"[3] with the frames inside the reference window of the first motion. Because we want to keep motion 1 intact in the forward matching case, the "pseudo-overlapped" portion is ignored. After trimming the second motion, we need to know the character's facing angle and position at the beginning frame of the trimmed motion. We use linear extrapolation to estimate the start facing angle and position based on the last two frames of the first motion and the number of the frames to be inserted. Since we assume the dynamics of the motion would not change a lot around the connection, the linear extrapolation should work; however, if this not the case, we might need higher order extrapolation—using more frames to estimate the root position and facing angle. The equation for linear extrapolation is listed as follows

$$\theta_{2d} = \theta_1[N_1 - 1] + (N_i + 1) \cdot (\theta_1[N_1 - 1] - \theta_1[N_1 - 2]) \tag{14.a}$$

$$T_{2d} = T_1[N_1 - 1] + (N_i + 1) \cdot (T_1[N_1 - 1] - T_1[N_1 - 2]) \tag{14.b}$$

where $\theta_1[n]$ and $T_1[n]$ are the facing angle and root position at the *n*th frame of motion 1 respectively; N_1 is the number of frames of the first motion; N_i is the number of frames to be inserted.

[3] We call it "pseudo-overlapped" because it's not exactly the same as that in the first motion, but just a best matched portion in the second motion.

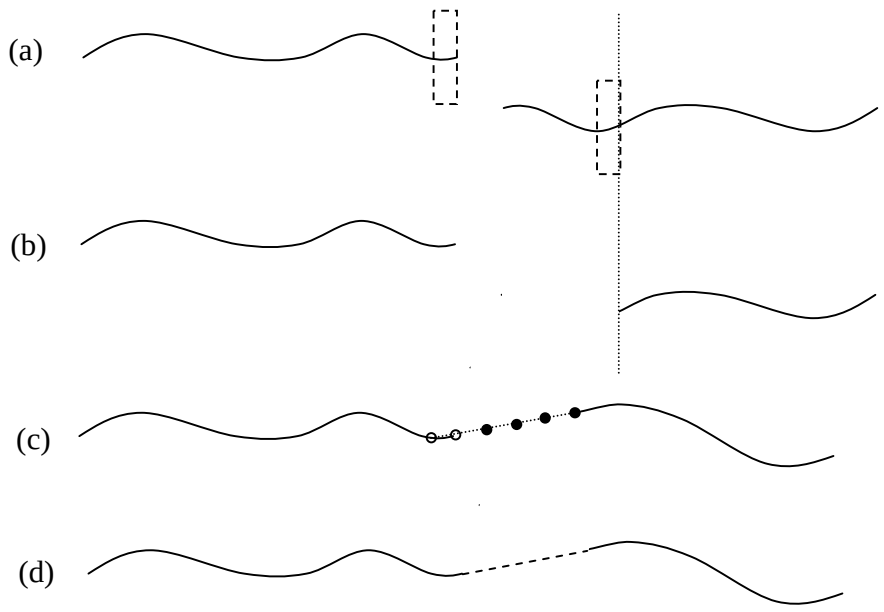

Figure 3 Motion connecting by interpolation—forward matching case. (a) Use forward matching to find the best matched frame in the second motion. (b) Trim the second motion. (c) Estimate the start facing angle and position of the trimmed motion using linear extrapolation based on the last two frames of the first motion. (d) Use interpolation module to generate the interpolated motion, and concatenate the interpolated and the trimmed second motion to the first motion.

Connecting two motions by blending

The procedures to connect two motions by blending are similar to that by motion interpolation. After finding the best matched frames for connection, we remove those frames before the matched window in the second motion because we want to blend these overlapped frames inside the matched window with those in the first motion. The facing angle and root position of the trimmed motion is set to be the same as that of the first frame of the reference window in the first motion. That is

$$\theta_{2d} = \theta_1[N_1 - N_f] \tag{15.a}$$

$$T_{2d} = T_1[N_1 - N_f] \tag{15.b}$$

where N_f is the length of the matching window—number of the frames inside the window. After transforming the facing angle and root position of the second motion, we apply the ease in/ ease out blending to the overlapped frames. The connection procedures are shown in Figure 4.

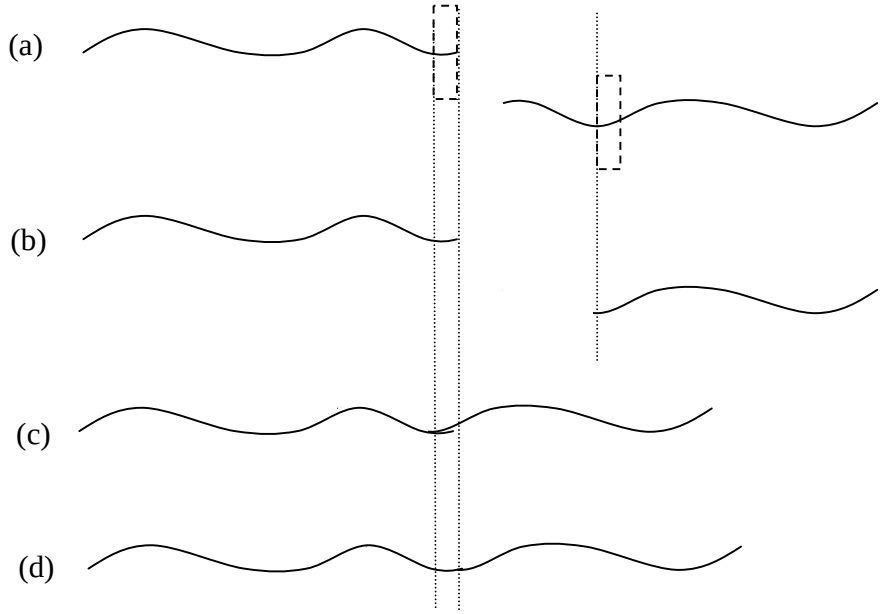

Figure 4 Motion connecting by blending—forward matching case. (a) Use forward matching to find the best matched frame in the second motion (b) Trim the second motion. (c) Set the facing angle and position of the trimmed motion to be that of the first frame in the reference window in the first motion. (d) Perform ease in/ease out blending on the overlapped frames, and concatenate the blended and trimmed second motion to the first motion.

IV. Inverse kinematics

Approaches to solve the inverse kinematics problem can be roughly divided into two categories: numerical approach and analytical approach. The numerical approach is more stable and can avoid singularity problem, but the computation cost is more expensive and, due to the formulation of the problem, the optimal solution is not guaranteed. As for the analytical approach, it computes fast, and the solution is unique once it exists; however, it's less stable and could suffer from the singularity problem. Because of these pros and cons, many researchers combined these two approaches to solve the IK problem in human figure. Instead of tackling the problem in a whole, they separate the problem into several sub-problems, and use different approaches for each sub-problem based on its joint-linkage structure. We'll adopt the same concept and divide the human IK problem into four limb IK problems and a body IK problem. The limb IK problem will be solved using the analytic approach and the body IK will be handled by numerical approach.

Limb IK

A human arm can be modeled as a 7-DOF joint chain—3 at the shoulder, 1 at the elbow, and 3 at the wrist joint[4]. Because there's only 7-DOF in the joint chain, we can uniquely solve it if we know the wrist position (3 DOF) and orientation (3 DOF) and one additional constraint. The method used in the Moeditor is similar to [2]. Their idea can be summarized as follows:

1. Adjust the elbow angle such that the distance from shoulder to the wrist equals that from shoulder to the goal position.

2. Adjust the shoulder angle such that the wrist can reach the goal position.

3. Adjust swivel angle or elbow position to satisfy joint limit or get more natural posture.

4. Adjust wrist angle to satisfy the orientation constraint of the hand.

We take the same procedures but use different approach in step 3. They represent the joint limit in terms of swivel angle, and find a solution that is farthest to the joint limit if multiple solutions exist or a solution nearest to the joint limit if no solution exists. Their analytic solving process is somewhat tedious because of the operation on the Euler angle space. If the joint limit is represented in quaternion space, many redundant problems can be avoided. Instead of solving this analytically, we represent the joint limit in quaternion and use a 1-D minimization algorithm to find a solution. We'll describe the approach in the following paragraphs. Our limb IK solver includes the following steps:

[4]Note that the human leg can be modeled as a 7-DOF joint chain as well. Although the approach described here is based on the human arm structure, the same limb IK solver can be used for the legs.

1. Determine the 1-D rotation axis of the elbow-joint.

2. Form the local coordinate frame at the elbow joint at the reference pose.

3. Calculate the elbow angle according to the distance from the shoulder to the desired wrist position.

4. Use swivel angle to calculate the elbow position.

5. Form the local coordinate frame at the elbow joint at the desired pose.

6. Calculate the desired AMC angle at the shoulder by comparing the local coordinate frames in step 2 and 5.

7. Check if the calculated AMC angle at the shoulder violates the joint limit; if auto-determination of the swivel angle is selected, apply 1-D search algorithm to find an optimal solution.

8. Calculate the AMC angle at the wrist according to the hand orientation constraint.

Elbow rotation axis determination

Because the elbow joint is restricted to be a 1-DOF joint in our model, we have to take into account those ASF skeleton files whose elbow joint is not modeled as 1-DOF joint. Assume this 1-D rotation axis is perpendicular to the upper arm and forearm. If the upper arm and fore arm are not aligned at the neutral pose, we can get its rotation axis by take the cross-product of upper arm direction and forearm arm direction vectors. In this case, we also need to calculate the offset elbow angle at the neutral pose and correct the elbow angle obtained in step 3. If the upper arm and fore arm are aligned, we can get an estimated axis from the mocap file—choose a frame in a motion and compute the cross-product on the upper arm and fore arm direction vector.

One advantage of estimate the rotation axis from the mocap file is that we can get the major rotation axis at the elbow even if it is modeled as a 3-DOF joint. This could prevent the problem in key frame interpolation. Because the additional DOF in the elbow joint, we could use any one to achieve our goal theoretically; however, if the generated IK solution is not applied to the major axis, the abrupt change in joint angle may result undesired interpolation behaviors.

This elbow axis determination process is actually similar to the problem of skeleton mapping—from position markers to get the joint angle on different skeletons. In the future, we may consider to map all different skeletons into a standard one in the file IO stage and don’t need to worry about these skeleton difference in the data processing stage.

Elbow angle calculation

The calculation first applies the cosine rule to get the angle between the upper arm and the fore arm.

$$\theta = \cos^{-1}\left(\frac{{l_1}^2 + {l_2}^2 - {l_3}^2}{2 l_1 l_2}\right) \tag{16}$$

where l_1, l_2, and l_3 are the bone length of the upper arm, fore arm, and the distance from the shoulder to the goal position, respectively.

Because we already got the rotation axis at the elbow joint, we can easily convert this angle into the AMC angle. Given the rotation axis $\hat{V}$ and the rotation angle θ, the rotation can be represented as the quaternion[5]

$$q = \left(\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\hat{V}\right) \tag{17}$$

The AMC angle can then be obtained by converting q to Euler angles. Note that the rotation axis should be represented in the local coordinate because the AMC angle is defined with respect to the local coordinate axis.

Elbow position calculation

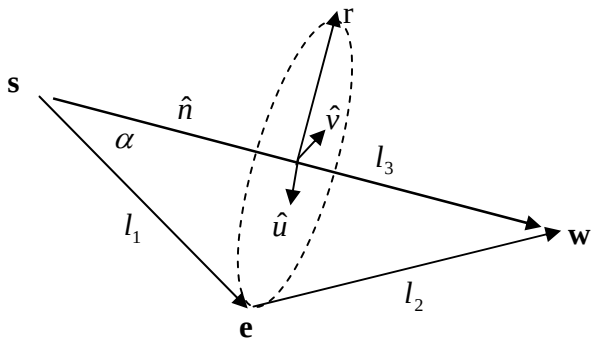

Figure 5. Calculating elbow position according to the swivel angle.

Figure 5 shows the configuration of the variables. s and e are the position vector of the shoulder and elbow. w is the goal position of the wrist. Given the goal position of the wrist w, and the shoulder position s, the elbow position is restricted on a circle. This circle is on the plane whose surface normal $\hat{n}$ is parallel to the direction from shoulder to wrist.

[5] In the program, we actually use the exponential map to get the quaternion for simplicity, but it gives the same result as this equation. For reference on the exponential map, please see [3].

$$\hat{n} = \frac{w - s}{\|w - s\|} \tag{18}$$

The elbow position can be parameterized by a swivel angle ϕ and a local coordinate $(\hat{u}, \hat{v})$ defined on the circle. $\hat{u}$ is the axis where the swivel angle is zero. It is defined by projecting a user-specified unit vector $\hat{a}$ on $\hat{n}$. To save the user's burden, $\hat{a}$ is preset as the negative global y-axis (0, -1, 0). If $\hat{a}$ is parallel to $\hat{n}$, $\hat{a}$ is set as the negative z-axis.

$$\hat{u} = \frac{\hat{a} - (\hat{a} \cdot \hat{n})\hat{n}}{\|\hat{a} - (\hat{a} \cdot \hat{n})\hat{n}\|} \tag{19}$$

$$\hat{v} = \hat{n} \times \hat{u} \tag{20}$$

The elbow position can be computed as follows

$$\alpha = \cos^{-1}\left(\frac{{l_1}^2 + {l_3}^2 - {l_2}^2}{2 l_1 l_3}\right) \tag{20}$$

$$r = l_1 \sin\alpha \tag{21}$$

$$e(\phi) = s + l_1 \cos\alpha \cdot \hat{n} + r(\cos\phi \cdot \hat{u} + \sin\phi \cdot \hat{v}) \tag{22}$$

Shoulder angle calculation

The computation of the shoulder angle is based on the comparison of two elbow coordinate frames defined at the reference pose and the desired pose. We can define a coordinate frame at the elbow based on the upper arm direction, fore arm direction and the elbow rotation axis as long as the arm is not outstretched. Because it is very common that the arm is outstretched in neutral pose, we would use a reference pose in which the arm is bending to avoid this problem. The math equations used to compute the shoulder joint angles are described as follows:

Suppose F_0 is a coordinate frame defined at the elbow joint at the reference pose, then

$${}^{s}_{0}R\,{}^{e}_{s}R_{asf}\,{}^{e}_{s}R_{amc}\,{}^{w}_{e}R_{asf}\,{}^{w}_{e}R_{amc}U = F_0 \tag{23}$$

We can obtain its representation in the local coordinate U by

$$U = ({}^{s}_{0}R\,{}^{e}_{s}R_{asf}\,{}^{e}_{s}R_{amc}\,{}^{w}_{e}R_{asf}\,{}^{w}_{e}R_{amc})^{T} F_0 \tag{24}$$

Given the desired elbow and wrist position, we can form another coordinate frame F_d at the desired pose.

$${}^{s}_{0}\overline{R}\,{}^{e}_{s}R_{asf}\,{}^{e}_{s}\overline{R}_{amc}\,{}^{w}_{e}R_{asf}\,{}^{w}_{e}\overline{R}_{amc}U = F_d \tag{25}$$

Because all matrices in Eq. 25 are known except the shoulder rotation matrix ${}^{e}_{s}\overline{R}_{amc}$, ${}^{e}_{s}\overline{R}_{amc}$ is obtained by

$$ {}^{e}_{s}\overline{R}_{amc} = {}^{e}_{s}R_{asf}{}^{T}\ {}^{s}_{0}\overline{R}^{T} F_d U^{T}\ {}^{w}_{e}\overline{R}_{amc}{}^{T}\ {}^{w}_{e}R_{asf}{}^{T} \tag{26} $$

The Euler angle of the shoulder joint is then computed from the rotation matrix.

Swivel angle searching based on joint limit constraint

To resolve the additional DOF in swivel angle, we can apply a 1-D minimization algorithm to find an optimal solution. The objective function can be defined as a measurement for the margin to the joint limits or the naturalness of the posture. Currently, we only check if the shoulder joint angles satisfy the joint limit and the margin to the joint limit in the Moeditor. For a 3-DOF joint limit constraint, there are 8 extreme joint angle sets. If we convert 8 Euler angles into quaternion, they form a closed region on the unit sphere surface, which bounds the legal range of the joint rotation. Figure 6Figure 6 shows the 3-DOF joint rotation range in the quaternion space. Because there is only 1-DOF remaining for the swivel angle, the minimization problem is like finding a best solution on the arc which is the trajectory of the joint rotation formed by varying ϕ. This swivel arc may intersect the legal region or not. If it intersects, we can choose a solution that is farthest to the joint limit. If not, we'll choose one that is nearest to the joint limit boundary.

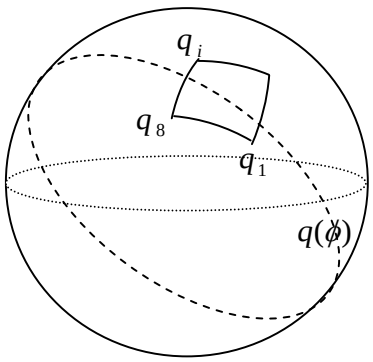

Figure 6 Joint limit region and the arc of swivel angle in quaternion space

The problem now turns to be how to define the distance on the 4-D unit sphere. We use a metric introduced in [3]

$$ dist(q_1, q_2) = \left\| \log(q_1^{-1} q_2) \right\| \tag{27} $$

where

$$\log(q_1^{-1}q_2) = \log(w,v) = \begin{cases} \dfrac{\pi}{2}v, & \text{if } w = 0 \\ \dfrac{v}{\|v\|}\tan^{-1}\dfrac{\|v\|}{w}, & \text{if } 0 < |w| < 1 \\ 0, & \text{if } w = 1 \end{cases} \tag{28}$$

w is the scalar part and v is the vector part of the quaternion. The objective function is defined as the sum of the current joint rotation to each corner of the joint limit region.

$$f(q(\phi)) = \sum_{i=1}^{8} dist(q(\phi) - q_i) \tag{30}$$

where $q(\phi)$ represents the rotation quaternion at the shoulder joint. It can be obtained from Eq. 26 by converting ${}^{e}_{s}\overline{R}_{amc}$ to quaternion and the searching algorithm loops through step 4 to step 7 to find a feasible solution for swivel angle.

Wrist angle calculation

The orientation of the hand can be fully specified by assigning its pointing direction $\hat{V}_{hd}$ and a rolling angle θ rotating about this pointing direction. We'll first adjust the wrist angle to satisfy the constraint of the hand's pointing direction (bone direction). If $\hat{V}_{hd}$ is the desired pointing direction, we'll need to find a rotation matrix ${}^{h}_{w}\overline{R}_{amc}$ satisfying the following equation

$${}^{w}_{0}\overline{R}\,{}^{h}_{w}R_{asf}\,{}^{h}_{w}\overline{R}_{amc}\hat{V}_h = \hat{V}_{hd} \tag{31}$$

Because we can always find a rotation matrix ΔR to rotate a vector $\hat{V}_h$ to $\hat{V}_{hd}$

$$\hat{V}_{hd} = \Delta R\hat{V}_h \tag{32}$$

We can insert Eq. 32 into Eq. 31, and equate the matrix on both sides, which gives us

$${}^{h}_{w}\overline{R}_{amc} = ({}^{w}_{0}\overline{R}\,{}^{h}_{w}R_{asf})^T\,\Delta R \tag{33}$$

The matrix ${}^{h}_{w}\overline{R}_{amc}$ obtained would satisfy the requirement in Eq. 31. Note that ${}^{h}_{w}\overline{R}_{amc}$ is just one of the solutions because the point direction is just a 2-DOF constraint, there will be an infinite number of solutions of ${}^{h}_{w}\overline{R}_{amc}$ for the wrist is modeled as a 3-DOF joint.

To adjust the rolling angle, we post-multiplied ${}^{h}_{w}\overline{R}_{amc}$ by another rotation matrix R_{roll} whose rotation axis is parallel to pointing direction.

$$ {}^{h}_{w}\overline{R}_{amc} = {}^{h}_{w}\overline{R}_{amc} R_{roll}(\theta, \hat{V}_h) \tag{34} $$

Note that the rotation axis of R_{roll} is represented in the local coordinate.

Numerical IK

The inverse kinematics problem can be formulated as a nonlinear optimization problem. If we represent the goal position as a position vector T_g and the orientation as two orthonormal vectors $\hat{x}_g$ and $\hat{y}_g$, the objective function can be defined as follows [5]

$$ P(T_e, \hat{x}_e, \hat{y}_e) = (T_g - T_e)^2 + c_{dx}^2 (\hat{x}_g - \hat{x}_e)^2 + c_{dy}^2 (\hat{y}_g - \hat{y}_e)^2 \tag{35} $$

where T_e, $\hat{x}_e$, and $\hat{y}_e$ are the position vector, and the direction vectors to specify the end effector's orientation. c_{dx} and c_{dy} are constant factors to adjust the unit difference of length and angle. To make one length unit commensurate with d degree in angle, we can define them as follows

$$ c_d = \frac{180}{\pi d} \tag{36} $$

Gradient of the objective function

As the gradient of the objective function is needed in the optimization process, we need to compute it.

$$ \nabla_\theta P = -(\frac{\partial T_e}{\partial \theta})^T \cdot 2(T_g - T_e) - (\frac{\partial \hat{x}_e}{\partial \theta})^T \cdot 2c_{dx}^2 (\hat{x}_g - \hat{x}_e) - (\frac{\partial \hat{y}_e}{\partial \theta})^T \cdot 2c_{dy}^2 (\hat{y}_g - \hat{y}_e) \tag{37} $$

where $\theta = [\theta_1 \quad \cdots \quad \theta_i \quad \cdots \quad \theta_N]^T$ are the joint variables, and

$$ \frac{\partial T_e}{\partial \theta} = \left[\frac{\partial T_e}{\partial \theta_1} \quad \cdots \quad \frac{\partial T_e}{\partial \theta_i} \quad \cdots \quad \frac{\partial T_e}{\partial \theta_N} \right] \equiv J_T \tag{38} $$

$$ \frac{\partial \hat{x}_e}{\partial \theta} = \left[\frac{\partial \hat{x}_e}{\partial \theta_1} \quad \cdots \quad \frac{\partial \hat{x}_e}{\partial \theta_i} \quad \cdots \quad \frac{\partial \hat{x}_e}{\partial \theta_N} \right] \equiv J_x \tag{39} $$

$$\frac{\partial \hat{y}_e}{\partial \theta} = \left[\frac{\partial \hat{y}_e}{\partial \theta_1} \quad \cdots \quad \frac{\partial \hat{y}_e}{\partial \theta_i} \quad \cdots \quad \frac{\partial \hat{y}_e}{\partial \theta_N} \right] \equiv J_y \tag{40}$$

are the Jacobian matrices. If we assemble Eq. 38~40 into a big Jacobian matrix

$$J = \begin{bmatrix} J_T \\ J_x \\ J_y \end{bmatrix} \tag{40}$$

$$b = \begin{bmatrix} T_g - T_e \\ \hat{x}_g - \hat{x}_e \\ \hat{y}_g - \hat{y}_e \end{bmatrix} \tag{41}$$

Eq. 37 can be rewritten in matrix form

$$\nabla_\theta P = -2J^T \cdot b \tag{42}$$

Computation of Jacobian

The elements in Eq. 38~40 are computed as follows

$$\frac{\partial T_e}{\partial \theta_i} = \hat{u}_i \times (T_e - T_i) \tag{43}$$

$$\frac{\partial \hat{x}_e}{\partial \theta_i} = \hat{u}_i \times \hat{x}_e \tag{44}$$

$$\frac{\partial \hat{y}_e}{\partial \theta_i} = \hat{u}_i \times \hat{y}_e \tag{45}$$

where $\hat{u}_i$ is the rotation axis of θ_i. It can be extracted from the local coordinate frame j. If the rotation matrix is ${}^j_0R$, and θ_i corresponds to *k*th DOF of the joint j, then u_i is the *k*th column vector in ${}^j_0R$. The meanings of Eqs. 43~45 can be interpreted as how the end effector's position and orientation would change due to the small amount of change in individual DOF of the joint. They also indicate how the linear velocity at the end effector relates to each individual DOF of the joint[6].

[6] Recall that $v = \frac{dx}{dt} = \frac{dx}{d\theta}\frac{d\theta}{dt} = (\hat{u}_i \times r)w_i$

In Eqs. 43~45, it's actually assumed all the joints are revolution joints. If we also want the root position be modified in the IK process, we need to consider the Jacobian calculation for a translational joint[7], which is given as follows

$$\frac{\partial T_e}{\partial \theta_i} = \hat{u}_i \tag{46}$$

$$\frac{\partial \hat{x}_e}{\partial \theta_i} = 0 \tag{47}$$

where $\hat{u}_i$ is the translation axis of θ_i. In our case, because the translation DOF only exists in the first 3 DOFs of the root, $\hat{u}_i$ is x-axis, y-axis, or z-axis depending on the translation direction. The result of Eq. 46 and 47 are not surprising since the contribution of translational joint to the end effector's velocity is equal for all DOFs and it doesn't affect the orientation of the end-effector.

After computing the objective function, and its gradient, we can now apply the optimization algorithm to solve our IK problem. The algorithm we use is the conjugate gradient method[8] from [6].

Implementation issues

Although the objective function in Eq. 35 is fixed, we actually design the program that allows the objective function has flexible number of terms—there might be different goal position and orientation for different joints. We use a constraint class to specify the goal position and orientation in the objective function. Also, to restrict the optimization only performing on specific joints, we use a bone mask to determine if the joint variable at this bone would be altered. All of this makes the computation of Jacobian look complicated but it just wants to fit the different IK settings.

References

[1] K. Shoemake, "Animating Rotation with Quaternion Curves," SIGGRAPH'85.

[2] D. Tolani, A. Goswami, and N. Badler, "Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs," Graphical Models, 62, pp 353-388, 2000.

[3] J. Lee, "A Hierarchical Approach to Motion Analysis and Synthesis for Articulated Figures," Ph.D. Thesis, Korea Advanced Institute of Science and Technology, 2000. (http://www.cs.cmu.edu/~jehee).

[4] F. Grassia, "Practical Parameterization of Rotations Using the Exponential Map," The Journal of Grpahics Tools, vol. 3.3, 1998.

[7] The root position is kept unchanged by default in the program since we don't want the optimizer just move the character to achieve the goal without adjusting the posture at all.

[8] There is a minor difference from the original code though. Please see the mathlibrary section in Appendix.

[5] J. Zhao and N. I. Badler, “Inverse Kinematics Position Using Nonlinear Programming for Highly Articulated Figures,” ACM Transactions on Graphics, vol. 13, no. 4, 1994.
[6] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, “Numerical recipes in C,” Cambridge University Press.