Chapter 1

Related Work

Inverse kinematics is a widely studied topic in robotics. Of special interest for this project are ways of improving inverse kinematics for interactive and data driven character animation specifically. One very data heavy approach this project was motivated by is proposed in "Motion Fields for Interactive Character Animation" [6], which uses a motion field over all joints that maps possible joint configurations to a set of motions that can follow from the current skeleton state as well as its desired velocity. Given the right database, this enables the character to react to changes quickly and in a natural looking manner. Inverse kinematics is used to correct for ground adaptation and to prevent foot sliding by attaching a bit to each database entry that encodes whether the foot is currently up or down. Unfortunately this approach requires a lot of motion data, which makes creating motion controllers for non-human characters hard.

A data driven method that focuses more on enhancing conventional Inverse Kinematics is "Example Guided Inverse Kinematics" developed by Seyoon Tak and Hyeong-Seok Ko [7]: A reference pose provides the IK solver with an additional objective that tries to imitate the joint angles. This reference pose can come from a skeleton with different proportions, but does not adapt depending on the actor's current state and, without further motion blending, is not well suited for interactive applications.

Andreas Aristidou and Joan Lasenby [1] developed "Forward And Backward Reaching Inverse Kinematics" (FABRIK), which is used in many popular 3D programs (Blender, Unreal Engine etc.) as it is computationally cheap and usually produces believable looking motion. FABRIK approximates a pose by moving a skeleton subtree back and forth between target and root until their distance to the subtree is small enough. This method requires many iterations that are relatively cheap, it allows adding joint constraints easily and it preserves the initial pose well. It is however a heuristic method that does not optimise an energy function and doesn't allow adding additional objective functions such as a regulariser, inertia objectives or data driven objectives.

Chapter 2

Inverse Kinematics

The IK system is the last segment in the animation pipeline. It is given an input pose and supplies an output pose that is optimised towards given objectives.

2.1 Skeleton initialisation

The skeleton initialisation has to be performed only once per actor. An actor's skeleton is given by a list of n bones $b_0 \dots b_n$ where b_0 is the root bone, and a list of bone indices $p_1 \dots p_n$ where p_i is the parent index of the i-th bone. For the IK system the user defines bone indices $e_0 \dots e_k$ and depths $d_0 \dots d_k$ for k end-effectors. The depth d_i for the i-th end-effector b_{e_i} defines at least how many bones from b_{e_i} towards the tree root b_0 are affected by this end-effector. Multiple of these paths can overlap and result in a forest of $l < k$ trees with roots $r_0 \dots r_l$. During initialisation, a list of children for every bone is created as the tree has to be walked from the roots every iteration.

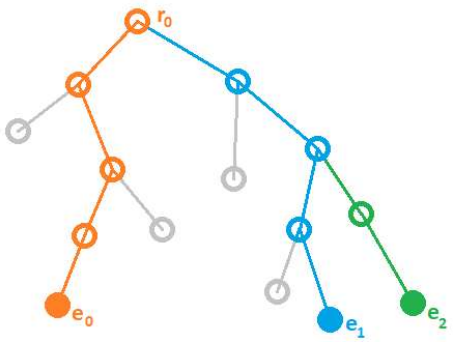

FIGURE 2.1: **Orange** is a first end-effector with depth $\geq$4, **Blue** is a second end-effector with depth $\geq$4 and **Green** is a third end-effector with depth $\geq$2. **Grey** are unused bones that are not affected by the IK system. All of the end-effectors shown create one tree, but multiple are possible.

2.2 Transform space

Transforms are simplified to only rotation and translation as skewing and scaling are unused. Be the set of transforms $\mathbb{T} = \mathbb{R}^4 \times \mathbb{R}^3$ tuples of quaternions and translation vectors. A transform $t = (t_q, t_t) \in \mathbb{T}$ can be rotated by a quaternion $q \in \mathbb{R}^4$ using quaternion multiplication:

$$q \cdot t = (q \cdot t_q, q \cdot t_t)$$

t can also be transformed by another transform $t' = (t'_q, t'_t) \in \mathbb{T}$ by first rotating then translating:

$$t' \cdot t = (t'_q \cdot t_q, t'_t + t'_q \cdot t_t)$$

2.3 IK configuration

For simplicity, all bones are assumed to be hinges where every bone b_i has a rotation axis a_i in bone space. A pose L is given by a list of bone transforms $L_0, \ldots, L_n \in \mathbb{T}$ in bone space (relative to their parent) and a from L precomputed list C of bone transforms $C_0, \ldots, C_n \in \mathbb{T}$ in component space (relative to the actor). The IK configuration $\vec{q} = (q_0, \ldots, q_n) \in \mathbb{R}$ defines all current joint angles in radians. It is additive, meaning its initial value $\vec{q} = \vec{0}$ describes the unchanged input pose. A transform operation $C = F(L, \vec{q})$ is used to update C to the input pose L transformed by the configuration $\vec{q}$: For every bone b_i that is relevant to the IK system and isn't a root, its component transform C_i is computed recursively by first rotating L_i around the bone axis by converting a_i, q_i from Axis-Angle to quaternion $r \in \mathbb{R}^4$ and $L'_i = r \cdot L_i$, then transforming it into component space with the parent bone transform $C_i = C_{p_i} \cdot L'_i$. By computing translation and rotation separately, the rotation axis $a'_0, \ldots, a'_n$ of all bones in component space can be extracted and cached for the IK algorithm.

2.4 Newton's Method

The energy function $E = \sum E_x$ defines a sum of objectives that is minimised iteratively using Newton's method. The IK configuration is changed every iteration by $\Delta q = -H^{-1} g \cdot \theta$ with an adaptive step size $\theta \in [0, 1]$ (bisection starting at $\theta = 1$), gradient $g = \nabla E = \frac{d}{dq} E$ and Hessian $H = \nabla^2 E = \frac{d^2}{dq^2} E$. Both the gradient g and Hessian H can be computed as the sum of all gradients or Hessians from each objective such that $g = \sum g_x$ and $H = \sum H_x$. The gradient and Hessian are computed analytically and validated using finite difference. The Hessian H is assumed to be symmetric positive definite and can be solved using the Cholesky decomposition (fall-back to gradient descent on a saddle point).

2.5 Independent Objectives

Independent objectives are objectives on bones that don't depend on other bones. In the Hessian they add only to the diagonal. The regulariser tries to keep the output pose close to the input pose so the IK configuration's magnitude can be used: With weight w_{Reg} the regulariser defines the energy function $E_{Reg} = w_{Reg} \cdot \frac{1}{2} \cdot ||q||^2$, the gradient $g_{Reg} = w_{Reg} \cdot q$ and the Hessian $H_{Reg} = diag(w_{Reg})$. Reversely, to keep the configuration values between $[-\pi, \pi]$ the limiter objective defines a soft limit: The energy function with weight w_{Lim} is a piecewise third order polynomial function that spikes at $-\pi$ and π where $E_{Lim} = w_{Lim} \cdot \sum_{i=0}^{n} l(q_i)$ and $l : \mathbb{R} \mapsto \mathbb{R}$ where for a stiffness $s \in \mathbb{R}$:

$$l(x) = \begin{cases} ((x + (\pi - \frac{\pi}{s})) \cdot -s)^3 & \text{if } x < -(\pi - \frac{\pi}{s}) \\ ((x - (\pi - \frac{\pi}{s})) \cdot s)^3 & \text{if } x > (\pi - \frac{\pi}{s}) \\ 0 & \text{otherwise} \end{cases}$$

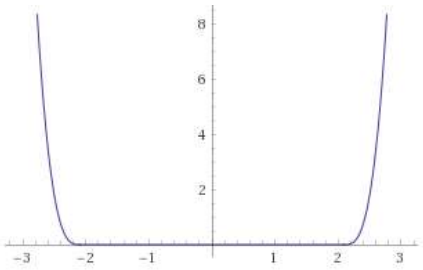

FIGURE 2.2: **Plot** of $l(x)$ for $x \in [-\pi, \pi]$

The gradient g_{Lim} and Hessian H_{Lim} are computed with the first and second derivative of $l(x)$ respectively.

2.6 End-effector objective

Every end-effector b_{e_i} has a user defined target location o_i with weight w_{End_i}. For the objective, the distance between end-effector and target location is minimised with the energy function $E_{End} = w_{End_i} \cdot \frac{1}{2} \cdot ||(c_{e_i t} - o_i)||^2$, its gradient $g_{End} = w_{End_i} \cdot J^T \cdot (c_{e_i t} - o_i)$ with Jacobian $J = \frac{d}{dq}(c_{e_i t} - o_i)$ and its Hessian $H_{End} = w_{End_i} \cdot (T \cdot (c_{e_i t} - o_i) + J^T J)$ with tensor $T = \frac{d^2}{dq^2}(c_{e_i t} - o_i)$. For any bone b_j between the end-effector b_{e_i} and its root, the Jacobian column J_j is computed analytically with the cross product between their distance vector and the rotation axis in component space:

$$J_j = a'_j \times (c_{e_i} - c_j)$$

For any bone $b_{j'}$ between the bone b_j and its root, the Tensor entry $T_{jj'}$ is computed analytically with the cross product between the Jacobian entry and the rotation axis in component space:

$$T_{jj'} = a'_{j'} \times J_j$$

2.7 Normal objective

It is crucial for many applications to also define a desired joint rotation. While a similar behaviour can be achieved adding virtual bones with end-effector objectives to the skeleton it is more convenient and provides better performance to add a normal objective. For this, every end-effector b_{e_i} has a user defined target normal n_i with weight w_{Norm_i}. For the objective, the angle between end-effector and target normal is minimised with the energy function $E_{Norm} = w_{Norm_i} \cdot (1 - a'^T_{e_i} n_i)$, its gradient $g_{Norm} = w_{Norm_i} \cdot -J^T \cdot n_i$ with Jacobian $J = \frac{d}{dq} a'_{e_i}$ and its Hessian $H_{End} = w_{Norm_i} \cdot -T \cdot n_i$ with tensor $T = \frac{d^2}{dq^2} a'_{e_i}$. The Jacobian J_i and Tensor T_i are computed analytically analogous to the end-effector objective.

2.8 Inertia objectives

There are two inertia objectives that are time dependent over a time interval Δt and try to maintain joint acceleration. One of the two objectives is an independent objective that, given a weight w_{Iner} for a bone b_i, maintains angular acceleration with energy function $E_{Iner} = w_{Iner} \cdot \frac{1}{2} \cdot (q_i - (q'_i + v'_i \cdot \Delta t))^2$ where q'_i is the joint angle and

v_i' the angular velocity from last frame. The second objective is an end-effector objective that sets the objective location from b_j to $o_j = p_j' + v_j' \cdot \Delta t$ where p_j' is the joint location and v_j' its velocity from last frame.

2.9 Extensions

Since usually a character has a separate root bone parenting a pelvis bone, the pelvis bone is used for rotation of the whole actor and the root bone's axis is used for its translation instead. Replacing rotation with translation causes hardly any overhead as neither the IK configuration nor computation of the Hessian H_{End} are affected. The system is also expanded to more than one rotation axis by extending q for every degree of freedom and applying all rotations as if they were additional hinges attached to the same bone. Similar techniques can be applied to introduce scaling and skewing.

2.10 Takeaways

- The normal objective is linear, not quadratic like most other objective functions. The error in respect to the finite difference test therefore increases deceptively fast for smaller differences.

- While the inertia objectives can greatly stabilise motion they also highly increase the reaction time to big end-effector target changes.

- Applying different weights to individual bones, specifically on the regulariser, can improve the output pose significantly. E.g. too strong regulariser on root bones with translation axis can push the actor away from the optimal solution.

Chapter 3

Foot Trajectories

The foot trajectory system is the root node in the animation pipeline and generates a walk cycle. It produces both the input pose and the k feet end-effector locations $\vec{p}_0 \dots \vec{p}_k \in \mathbb{R}^3$ used by the IK system. This project contains two segments for the foot trajectory system, step generation $S(I, t)$ and ground adaptation $G(I, W, t)$, given an idle animation $I(t')$ and a walking animation $W(t')$ with timestamps $t, t' \in \mathbb{R}$ in seconds. The walk cycle for each foot consists of 2 timestamps t_i^{up} and t_i^{dw} for the i-th foot stepping up or down.

3.1 Step generation

Step generation $S(I, t)$ produces an output pose without reference walk animation. The first frame of the idle animation $I(0)$ provides foot anchor points $\vec{q}_0 \dots \vec{q}_k \in \mathbb{R}^3$ that with additive vectors $\vec{v}_0(t) \dots \vec{v}_k(t) \in \mathbb{R}^3$ produce the final end effector locations with $\vec{p}_i(t) = \vec{q}_i + \vec{v}_i(t)$ for every i-th foot. $\vec{v}_i(t_i^{dw})$ is Interpolated linearly towards $\vec{v}_i(t_i^{up})$ when the foot is down and using piecewise cubic splines when the foot is up. In most cases, $v_i(t)$ differs for all feet only with time offsets and length/height multipliers. The foot rotation is generated the same way by setting the foot angle to 0 when the foot is down and a piecewise cubic spline when the foot is up. Step generation can plug right into the ground adaptation system instead of the walking animation W(I).

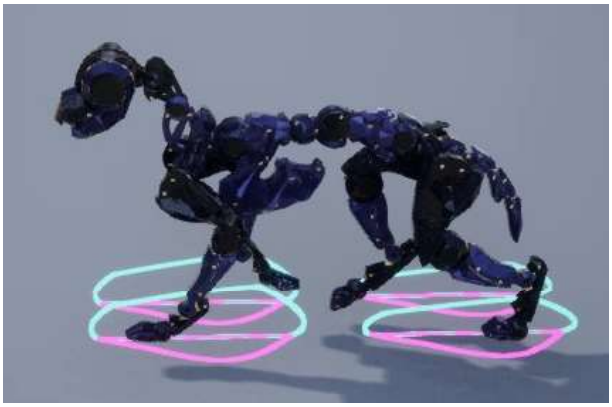

FIGURE 3.1: **Cyan** plots the foot trajectory $\vec{v}_i(t)$ over all t. **Magenta** plots the foot angle over all t.

3.2 Ground adaptation

Ground adaptation $G(I, W, t)$ produces an output pose that moves the foot end effectors from an input animation W(t') to the ground using a line trace. The first frame from the idle animation $I(0)$ provides foot anchor points to compute the desired distance between the foot and the ground. The foot trajectory is desired to be smooth even over non-smooth terrain. To achieve this, the ground is only traced in 2 locations, when the foot leaves the ground and when it steps down. All other foot locations are projected to a plane spanned by these 2 locations. Be $f_i(t) \in \mathbb{R}^3$ the projected i-th foot location in world space from $W(t)$ at time t. While the foot is down it stays at the location $f_i(t_i^{dw})$ to prevent foot sliding, but as $f_i(t_i^{dw}) \neq f_i(t_i^{up})$ there is a residual vector $d = f_i(t_i^{up}) - f_i(t_i^{dw})$ that needs to be compensated and linearly interpolated towards 0 during the step ark: $p_i(t) = f_i(t) + d \cdot (1 - \frac{t - t_i^{up}}{t_i^{dw} - t_i^{up}})$. The foot rotation is added (quaternion multiply) to the ground rotation. The ground rotation is computed using the surface normal interpolated between the 2 trace locations.

FIGURE 3.2: **Foot trajectories** adapt to the terrain.

3.3 Takeaways

- Step generation requires enabling the inertia objectives on the IK system to prevent the actor from swaying. As the motion field system needs the IK system to adapt quickly, step generation can be used together with the motion field system only for skeleton subtrees below the inertia objectives.

- For smoother transition between foot up and down, the residual vector can be reduced using cubic interpolation. The plane spanned by the two trace locations can be made a cubic mesh by adding at least a third trace location. Both made the animation look smoother but less natural.

- Using a spline for the actor translation over the terrain allows for this system to be forwarded and rewound. Depending on whether the spline is closer or further away from the ground the actor's feet stick through or hover over the terrain due to the IK regulariser.

Chapter 4

Motion Fields

The motion field segment reads a user defined state $s \in \mathbb{R}^m$ and outputs a configuration $q = (q_0, \ldots, q_n) \in \mathbb{R}^n$ for $n \in \mathbb{N}$ degrees of freedom, where each q_i defines a rotation around an axis r_i or translation along an axis a_i. This configuration additively transforms the input pose for the IK segment using motion data. The motion field database D maps a set of k user defined states $s'_0, \ldots, s'_k \in \mathbb{R}^m$ to a set of configuration entries $q'_0, \ldots, q'_k \in \mathbb{R}^n$. The input state s interpolates the database entries to an output state q.

4.1 Interpolation

q is computed using weighted arithmetic mean, using the inverse square distance between the input state and the database for weighting. That way, states from the database closest to the input state get the highest weight. For numerical stability the weight is clamped to the cut-off $\sigma \in \mathbb{R}$. To prevent division by zero, states with distance smaller than $\epsilon \in \mathbb{R}$ default to σ.

$$q = \frac{1}{Z} \sum_{i=0}^{k} q'_i \cdot w(s, s'_i) \text{ with } Z = \sum_{i=0}^{k} w(s, s'_i)$$

$$w(s, s') = \begin{cases} \sigma & ||s - s'_i||^2 < \epsilon \\ min(\sigma, \frac{1}{||s - s'_i||^2}) & \text{otherwise} \end{cases}$$

4.2 Data collection

While traversing the world, the actor is always aware of its input states. Those can be terrain attributes like ground slope or actor attributes like current feet location or change in direction. Linear states like the current ground slope are computed by interpolating the projection plane normals from the foot trajectory system. The change in movement direction is computed using finite difference between past and future actor orientation. The current state being always defined, new configuration entries can always be inserted into the motion field database and the actor's pose adapted to fit its current situation better. For this project, the actor can be stopped at any time t and configuration features can be changed during runtime.

4.3 Takeaways

- As the motion field segments are additive, multiple can be used in sequence for input states that are independent. Multiple segments can use the same database for different input states, for example if there are segments for each leg.

- Many segments with low dimensional states tend to produce better looking results. From all the tested states, up to 3 dimensional input states seem to perform best.

- Input states that depend on foot locations always oscillate greatly. For more consistent input states it makes sense to query the idle pose $I(0)$ used by the foot trajectory system where possible e.g. tracing the ground for the surface normal from the idle pose feet locations.

- The motion field system is unsuited to prevent strong disturbances such as the sway the step generation system applies to the actor. The user would need to add more entries to the motion field database than a traditional animation has key-frames.

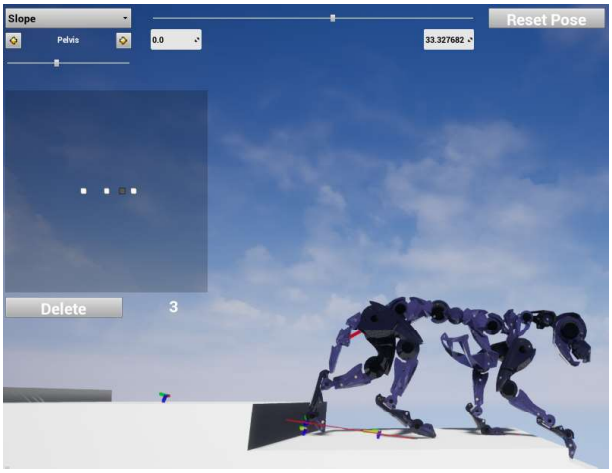

FIGURE 4.1: **Motion Field Interface** to insert data entries into the database. The white dots to the left are a projected 2D representation of the slope state feature. The current state can be edited with a joint selector and slider on the top left. The currently displayed timestamp can be changed with the slider on top.

Chapter 5

Conclusion

The introduced motion field system, when split into small, independent segments, is very close to blend spaces. It does however propose a different work-flow: Instead of animations being prepared for blending in advance, the actor can be sent through an obstacle course and data points inserted where the IK system produces unwanted results. This way, corner cases can be found and dealt with intuitively.

5.1 Drawbacks

The quality of the motion field output greatly depends on the input state features. Covering all desired states completely and smoothly is not always possible. Additionally, the motion field system can only directly impact the regulariser of the IK system. Only the foot trajectory system can adapt the end-effector objectives.

5.2 Outlook

In game design, Inverse Kinematics often work in tandem with blend spaces. While the motion field and IK system proposed in this project are well suited for that, the foot trajectory system is not. Fixed foot up and down timestamps can only be enforced if the blend space itself keeps the walk cycle synchronised. A more general foot trajectory system can be proposed that requires only an input pose but sacrifices the ability to rewind/forward the animation to any given time t.

Bibliography

[1] Andreas Aristidou and Joan Lasenby. “FABRIK: A fast, iterative solver for the Inverse Kinematics problem”. In: 2011.

[2] Epic Games. “Paragon Assets”. In: (2018). https://www.unrealengine.com/en-US/paragon.

[3] Epic Games. “Unreal Engine”. In: (2014). https://www.unrealengine.com.

[4] Lina Halper. “Creating Custom Animation Nodes”. In: (2015). https://www.unrealengine.com/en-US/blog/creating-custom-animation-nodes.

[5] Vel at LaTeXTemplates.com. “Template for a Masters / Doctoral Thesis”. In: 2017.

[6] Yongjoon Lee, Gilbert Bernstein, Jovan Popovic, and Zoran Popovic. “Motion Fields for Interactive Character Animation”. In: https://pdfs.semanticscholar.org/b899/e0ab35da3018e38ca50b74b1cccdd4a6dbec.pdf. 2010.

[7] Seyoon Tak and Hyeong-Seok Ko. “Example Guided Inverse Kinematics”. In: http://graphics.snu.ac.kr/publications/conference_proceedings/egik.pdf. 2000.