DOI: 10.1111/cgf.13514
ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2018
T. Beeler and N. Thuerey
(Guest Editors)

Volume 37 (2018), Number 8

Robust Physics-based Motion Retargeting with Realistic Body Shapes

Mazen Al Borno[1,2,3], Ludovic Righetti[3], Michael J. Black[3], Scott L. Delp[1], Eugene Fiume[2,4] and Javier Romero[3]

[1] Stanford University
[2] University of Toronto
[3] Max Planck Institute for Intelligent Systems
[4] Simon Fraser University

Abstract

Motion capture is often retargeted to new, and sometimes drastically different, characters. When the characters take on realistic human shapes, however, we become more sensitive to the motion looking right. This means adapting it to be consistent with the physical constraints imposed by different body shapes. We show how to take realistic 3D human shapes, approximate them using a simplified representation, and animate them so that they move realistically using physically-based retargeting. We develop a novel spacetime optimization approach that learns and robustly adapts physical controllers to new bodies and constraints. The approach automatically adapts the motion of the mocap subject to the body shape of a target subject. This motion respects the physical properties of the new body and every body shape results in a different and appropriate movement. This makes it easy to create a varied set of motions from a single mocap sequence by simply varying the characters. In an interactive environment, successful retargeting requires adapting the motion to unexpected external forces. We achieve robustness to such forces using a novel LQR-tree formulation. We show that the simulated motions look appropriate to each character's anatomy and their actions are robust to perturbations.

CCS Concepts

•***Computing methodologies*** → ***Animation; Physical simulation;***

1. Introduction

We all move differently. How we perform an action depends on our body shape. The motions of a tall heavy man and a small slim woman balancing on one foot, hopping and bending are distinct. Much of this is due to the specific distribution of mass over the body. Physical differences result in movement differences. Our goal is to model this efficiently and, given motion capture data from one subject, realistically animate humans with different body shapes doing the same movement. The result will be motions that differ in ways that are tailored to the body of the individual preforming the action.

Motion capture data is one of the main sources of animated behavior in graphics, due to its ground-truth realism. While high-end commercial animation can afford to record reference motions with actors that closely resemble the physique of their animated counterparts, this is typically not available in crowd simulations, and in many games having, for example, interchangeable characters. In these scenarios, it is not uncommon to apply mocap data to characters whose bodies do not match the physical characteristics of the original actors. This mismatch results in well documented artifacts like footskating [KSG02], shape interpenetration, or simply a perceived lack of realism. We would like to adapt a given motion capture sequence to any possible human shape in a fully automated, realistic manner.

To a large extent, the lack of realism in traditional mocap results from physical implausibility of the motion when applied to different target bodies. A natural solution to that problem is to enforce physical constraints on the retargeted mocap data. The classic spacetime optimization framework [WK88] has been used for that purpose [LHP05], by imposing physical constraints on optimization problems that minimize the deviation with respect to reference motions. However, such systems have been traditionally constrained to restrictive, expert-designed constraints like foot placements [LHP05], or are dedicated to very specific cyclic motions like walking, running or cycling [HP97].

Moreover, most of the previous work considers unrealistic, mannequin-like body proportions (with the notable exception of [HP97]), or retargets mocap data to non-human embodiments, like animals, imaginary creatures, robots or characters with human kinematic structure but implausible body proportions [LYG15].

Our approach is different: Rather than retargeting motion *despite* large shape differences, we would like to exploit common, natu-

ral shape differences among people to generate natural variations of human motion given a single source mocap sequence. Due to differences in body shape (limb lengths, proportions, fat distribution, etc.) each target subject will require different foot placements and will execute different motions while attempting to replicate the source motion. Some of these subjects will also struggle to imitate the action exactly, having to adapt it to their own bodies. In this paper, we explore how natural motion, including natural tracking "failures", can emerge from a simple optimization objective. By using physical models derived from real 3D scans, we show how our optimized physical motions have certain characteristics present in real sequences, like heavier subjects raising their arms to improve balance or tripping when the motion is too difficult to follow. This is different from previous work, which showcases motion mapped to imaginary creatures for which we have less intuition about how a natural motion should look.

Given a mocap sequence, we estimate how the sequence would be performed by actors of different body shapes. To that end, we use a realistic, parametric, 3D model of the human body [LMR*15]. Given any body shape, we approximate it with a set of "capsules" that capture the coarse shape of the body, yet are simple enough to enable practical physics simulation. We then generate plausible motions for these bodies that try to replicate the mocap but take into account that body shape influences what we can do and how we do it.

Another factor that makes motion unique for a particular shape, and varied in different situations, is their way of reacting to unexpected events. We would like to generate motions that are robust to external perturbations. To achieve this, we combine recent work on LQR-trees [Ted09] and domain of attraction expansion [ABvdPF17] to adapt existing controllers to new observed states in an efficient manner. This approach provides significant robustness to external perturbations that might be expected in an interactive game environment.

Summarizing, our overall goal is to make mocap more useful by adapting it to new characters and external forces in a physically plausible way. That is, we wish to make mocap data generalize more naturally and more robustly to new shapes and new situations. Our approach is based on physical simulation of the motion and has several components. First, we generate more varied and realistic motions than previous methods by removing hard constraints on the retargeting objective and using real but diverse target body shapes. Second, we make the physical simulation of different body shapes tractable by approximating their shape with "capsules". Third, we develop a new efficient algorithm for creating a tree of robust LQR controllers that can sustain disturbances. This represents another step toward realistic animated characters that behave realistically in a wide range of scenarios.

2. Related work

There is significant prior work on modeling human body shape accurately and on retargeting mocap data to new characters, but there is little work combining these threads. We argue that as human body models become more pervasive, it will become more important for realistic avatars to move realistically.

Motion capture is often used to directly animate a character given a mapping from the mocap skeletal motion to the character rig. Although the problem has received significant attention from the graphics community, today there is an enormous amount of manual labor in the process of mapping and refining these motions. A typical solution for this involves solving the pose for the target character through inverse kinematics [Gle98]. Since this problem is under-constrained, some approaches have limited the frequency of the motion and minimized the amount of induced pose change [Gle98]. Similar constraints can be applied to simplified skeletons [MBBT00] in order to deal with both geometric and topological differences between skeletons. Molla et al. [MDB17] introduce a kinematic approach to transfer the semantics of postures with self-body contacts to characters with different body shapes. Even contacts between multiple retargeted skeletons can be maintained through pure kinematic retargeting [HKT10]. Some kinematic methods attempt to encode some physical plausibility, such as the balance optimization in Lyard et al. [LMT08]. In contrast, our physics-based approach does not rely on heuritiscs and is not limited to specific movements. Note that physics-based and kinematic approaches can be combined, e.g., a physics-based motion retargeting can be performed after an initial kinematic motion retargeting phase that could incorporate animator needs, such as desirable self-contact constraints. Our work can incorporate kinematic constraints as soft constraints in the trajectory optimization.

An alternative approach is to learn a statistical motion model from several example mocap sequencess [LBJK09, FLFM15]. This allows for the animation of characters with natural variability. These models typically do not take into account the dynamics of the motion, so directly applying this procedural motion to new characters can produce unrealistic results. Moreover, the creation of some of these models requires motion recordings that are varied and aligned in time (e.g. cyclic motion). This limits their applicability.

Exploiting physics for processing and editing motion capture data has a long history in graphics. Zordan et al. [ZVDH03] proposed a physics prior to estimate joint trajectories from 3D marker position data. The approach does not reconstruct the control torques to replicate the mocap motion, which is a challenging problem. Some researchers have imposed strict physical plausibility on either motion captured data [LHP05] or human-edited animations [PW99, SHP04]. Spacetime optimization methods [MTP12, WK88] have been particularly popular for motion tracking [SP05, PW99]. These methods require manually pre-specifying the contacts to allow for gradient-based optimization methods, and they have yet to be demonstrated on a wide variety of movements.

In recent work, Han et al. [HENN16] show how to enable mocap data tracking in near real-time by using smoothed contact dynamics. The main limitations are on the physical accuracy of the contact dynamics, which causes visible ground penetrations and on the realism of the control torques used (the torque limits are not mentioned, but similar methods [TET12] require unrealistically large torques to avoid local minima). Other researchers have investigated stochastic global optimization and derivative-free methods. For instance, Liu et al. [LYvdP*10] show how a sampling-based approach can be used for the reconstruction of contact-rich movements. The method is improved in [LYG15] to synthesize dy-

namic movements and to reduce motion wobbliness. This is accomplished by computing multiple solutions and averaging them. Our work uses a sampling-based optimization method first introduced in [ABDLH13] to synthesize movements from high-level objectives, without prior data. Here, we show that the method can be used for motion tracking. It produces good quality motions without requiring an optimization for multiple solutions as in [LYG15]. We also show how motion retargeting can be performed with realistic body shapes and produce plausible motion variations, which was not demonstrated in previous work.

Physical controllers inferred from mocap can be used to generate new motions with a wide range of variability in response to external disturbances. We can categorize methods that increase controller robustness depending on whether their optimization is online or offline. Recent online optimization methods such as [HET*14, HRL15, HENN16] have proved to be quite robust by planning over a time horizon in the future. However, due to their online nature, they can easily fall into local optima. Recent offline methods [DLvdPY15, LvdPY16] optimize linear feedback policies for motion tracking under disturbances. To move beyond linear policies, Tedrake [Ted09] introduces a method to develop a nonlinear feedback policy that covers a domain of interest in state space by combining local linear quadratic regulators (LQR) controllers [dSAP08, MRS14] in a random tree. The method is only demonstrated on a low-dimensional problem (in their case, one degree-of-freedom). As follow-up work, [ABvdPF17] has shown how to scale the approach to a 25 degrees-of-freedom character. [MLPP09] introduces a nonlinear quadratic regulator for motion tracking that outperforms LQR but that is limited to simulators that rely on linear complementarity problems. Nevertheless, it should be possible to combine such a controller with [Ted09] to efficiently synthesize a more robust controller. Other approaches have studied how to build robust controllers by continuously modulating and synchronizing reference trajectories [LKL10], by tracking multiple reference trajectories in parallel [MPP11] or by combining existing controllers for new tasks [DSDP09]. An interesting alternative explored by [YL10] consists in learning from data how real subjects react to external forces. However, it is challenging to record reactions from a sufficiently varied set of people, motions and disturbances. Our work is closely related to [Ted09] and [ABvdPF17]. In place of the PD controllers used in [ABvdPF17], we use LQR controllers, which explicitly take into account the coupling among all the degrees of freedom, therefore enlarging the region of stability of the controllers. We also show how our method can be used to track a motion sequence by systematically building robust nonlinear feedback policies, whereas previous methods are linear or do not show how to expand their region of stability.

There has been significant work on learning realistic human body shapes from 3D scans; we focus on these rather than hand-constructed models. Most such human shape models can be posed using a skeleton [LMR*15] or per-part rotations [ASK*05], allowing animation from mocap data. The retargeting of mocap to a new body shape in [LMB14] is purely kinematic and ignores the body shape in the process.

There is previous work on modeling how realistic body shape influences the dynamics of soft tissue [PMRMB15, LMR*15]. These methods predict soft-tissue deformations as a function of the motion of the body over time. They do not, however, do the opposite; that is, the body shape never influences the articulated motion.

In addition to shape affecting how a body moves, it also influences the space through which it can move. Specifically in retargeting mocap from one shape to another, one must take care to avoid interpenetrations. To that end we need an efficient method to detect and avoid self penetration. We enable this by using an approximation to the shape that enables efficient intersection testing in the physics engine. Mesh approximation from different types of geometric primitives is a well studied subject [LWH*07, ZYH*15]. These methods, however, do not focus on re-posable simplifications. Re-posable sphere-based hand models are automatically constructed in [SMOT15] and [TPT16], but are more appropriate for computer vision than simulation because they do not define spheres with hard collisions. The sphere-mesh representations from [TGB13] achieves good and re-posable approximations of the shape, at the cost of using a large number of interpolated spheres. Our simple, re-posable representation, which uses a single capsule per body part, models efficiently the variation of human shape and can be easily optimized to fit a large number of subjects.

Hodgins and Pollard [HP97] takes a physical controller and adapts the masses and dimensions of the body to apply the controller to a new body shape. They describe this for a limited number of body shapes and focus only on existing controllers for cyclic motions. Here, we combine the above lines of research and explore how body shape influences motion as captured in a mocap setup.

3. Optimization for Motion Re-Targeting

The motion retargeting problem involves transforming a motion performed by one character to a similar motion performed by a character with a different shape. Typically this motion cannot be performed exactly because of the different physical characteristics.

Let $\bar{q}_{j,t}$ be the observed angular value of joint j at time t, $\bar{\mathbf{q}}_t = [\bar{q}_{1,t}, \ldots, \bar{q}_{N,t}]^T$ be a vector of all joints at time t, and $\bar{\mathbf{q}}_{1:T} = \{\bar{\mathbf{q}}_1, \ldots, \bar{\mathbf{q}}_T\}$ be the sequence of all joint angles over time.

Our goal is to estimate the torques necessary to reproduce the observed motion subject to physical constraints. This is a hard optimization problem that cannot be solved directly. Our approach uses a trajectory optimization (or spacetime optimization) method for physically-based retargeting. In particular, we build on the method of Al Borno et al. [ABDLH13], and optimize a physical model to approximate an observed trajectory of kinematics.

As in [ABDLH13], to simplify optimization, we introduce a fictitious "control" trajectory $\hat{\mathbf{q}}_{1:T-1}$ represented by a cubic B-spline. Optimizing the kinematics is achieved by indirect optimization of this control trajectory. The optimization below is over the spline knots, which correspond to the angles of the actuated joints. For clarity, however, we omit the knots below and simply write the trajectories.

Given a control trajectory, $\hat{\mathbf{q}}_{1:T-1}$, the torques, u_t, at each timestep t are determined by proportional-derivative (PD) control:

$$u_t = k_p(\hat{q}_t - q_t) - k_d\dot{q}_t, \quad (1)$$

where q_t and $\hat{q}_t$ are the current and control values of a joint angle and $\dot{q}_t$ is the angular joint velocity at time t. This is done independently for each joint so we have dropped the subscript j. Running this simulation using a particular control trajectory, $\hat{\mathbf{q}}_{1:T-1}$, gives both the torques and the output motion we desire $\mathbf{x}_{1:T} = [\mathbf{q}_{1:T}, \dot{\mathbf{q}}_{1:T}]$. Here, all joints have PD gain values of $k_p = 700$ Nm/rad and $k_d = 1$ Nms/rad. The joint torques are limited to 100 Nm for all joints.

A good $\hat{\mathbf{q}}_{1:T-1}$ is one that produces motions $\mathbf{x}_{1:T}$ by forward simulation that match the observed mocap states $\bar{\mathbf{x}}_{1:T} = [\bar{\mathbf{q}}_{1:T}, \dot{\bar{\mathbf{q}}}_{1:T}]$, where $\dot{\bar{\mathbf{q}}}_{1:T}$ is evaluated with finite differences. We search for these using a sampling-based approach (described below). Given a sample $\hat{\mathbf{q}}_{1:T-1}$, we run the simulation to get $\mathbf{x}_{1:T}$ and evaluate it given a cost function.

3.1. Cost Function

The cost function $E(\mathbf{x}_{1:T}; \bar{\mathbf{x}}_{1:T})$ encourages the physics simulation motion to be as close as possible to the mocap motion. We use a weighted sum of objectives E_i, with weights w_i:

$$E(\mathbf{x}_{1:T}; \bar{\mathbf{x}}_{1:T}) = \sum_i w_i E_i(\mathbf{x}_{1:T}; \bar{\mathbf{x}}_{1:T}). \quad (2)$$

We now define and discuss the objectives.

The E_{COM} objective encourages agreement between the character's simulated center-of-mass and the corresponding character's center-of-mass in the mocap data:

$$E_{COM} = \sum_t \|\mathbf{c}_t - \bar{\mathbf{c}}_t\|^2. \quad (3)$$

Likewise, we use the E_{COMv} objective for the velocity of the center-of-mass:

$$E_{COMv} = \sum_t \|\dot{\mathbf{c}}_t - \dot{\bar{\mathbf{c}}}_t\|^2. \quad (4)$$

We always use the 2-norm in this paper. Note that the center of mass can be computed from the pose of the body and its shape.

The $E_{trackingPoses}$ and $E_{trackingVel}$ encourages the simulation to have similar poses and velocities as the mocap:

$$E_{trackingPoses} = \sum_{j,t} \beta_j (q_{j,t} - \bar{q}_{j,t})^2, \quad (5)$$

$$E_{trackingVel} = \sum_{j,t} \beta_j (\dot{q}_{j,t} - \dot{\bar{q}}_{j,t})^2, \quad (6)$$

where j indexes the character's joints, excluding the root, and β_j is a weight. We use $\beta = 1$ for all joints, except for the hips and the shoulders, where we use $\beta = 10$ and $\beta = 3$ because of their importance on end-effector position. We do not have an objective to minimize the differences in end-effector positions directly, but it can be included if it is required for some application.

The E_{ROOT} term penalizes the difference between the unit quaternions parametrizing the root orientations:

$$E_{ROOT} = \sum_t \arccos(|\langle q_{root}, \bar{q}_{root} \rangle|), \quad (7)$$

where $\langle \cdot \rangle$ denotes the inner product and $|\cdot|$ the absolute value.

Lastly, the E_{jerk} term is used to reduce motion jitteriness:

$$E_{jerk} = \sum_t \|\ddot{q}_t\|^2. \quad (8)$$

We found the jerk term to produce better looking motions than the commonly used torque term, which is not used in our objective:

$$E_{torque} = \sum_{j,t} \tau_{j,t}^2. \quad (9)$$

The weights used are $w_{COM} = 15$, $w_{COMv} = 0.5$, $w_{trackingPoses} = 0.5$, $w_{trackingVel} = 0.0005$, $w_{ROOT} = 4$, and $w_{jerk} = 1e^{-10}$.

3.2. Optimization

Optimization involves sampling control trajectories (i.e., knots), $\hat{\mathbf{q}}_{1:T-1}$, running the forward simulation to get a motion, $\mathbf{x}_{1:T}$, and then evaluating the cost using $E(\mathbf{x}_{1:T}; \bar{\mathbf{x}}_{1:T})$. The optimization is performed with Covariance Matrix Adaption (CMA) [Han06], using 200 samples and 2000 iterations; that is, 400,000 simulations. The method is initialized with the **0** vector, which corresponds to a standing pose.

We use the MuJoCo [MuJ16] physics simulator with a 0.01 s timestep and a coefficient of friction of 1. The simulator computes the dynamic consequences of torques applied to our body representation based on geometric primitives (see Sec. 3.3), and their interaction with floor geometry, self-intersection and external disturbances. As a result, torques applied to different body shapes result in different, physically plausible poses.

To make the optimization tractable, we perform it within temporal windows. We divide the optimization for the entire motion in sequentially overlapping pairs of windows. As in [ABDLH13], we optimize using two adjacent $0.4s$ windows with 6 knots each, spaced at equal time intervals. Using two windows provides temporal continuity during optimization and we discard the results of the second window. The optimization time is about 25 minutes per pair of windows on a 4Gz Intel Core i7 machine with 4 cores. Our motion capture clips are recorded at 60 fps and have a typical length around 5 seconds, meaning that there are 12 window pairs per clip on average.

Note that varying the window length varies the results. A longer window lets the optimization look further in time, helping it find good solutions. In the results section we show the effect of this.

When retargeting the input motion, the vertical position of the desired position of the center-of-mass in (3) is adjusted based on the difference between the target and source bodies center-of-mass given a standing balance pose. Furthermore, the input motion is translated horizontally at the start of each optimization at timestep i based on the state of the simulated character (i.e., $\bar{\mathbf{c}}_i^{xz} - \bar{\mathbf{c}}_i^{xz} + \mathbf{c}_i^{xz}$), where xz denotes the horizontal components. This helps the simulated character not fall behind the input motion.

3.3. Mesh To Geometric Primitives

Given a realistic human shape, our goal is to retarget motion to this shape realistically. Differences across human bodies are more

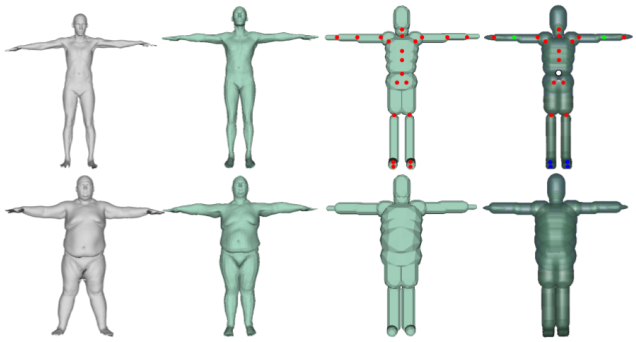

Figure 1: From scans to capsules. *Each row depicts two different Dyna subjects, the first one is* 1.69 *meters tall and* 62.2 *kg, while the second one is* 1.80 *meters tall and* 155.4 *kg. From left to right, we show* **a)** *scans of the subjects,* **b)** *the unposed template as a triangulated mesh,* **c)** *the optimized capsule representation, and* **d)** *the adapted geometric primitives representation. In the first row, two last columns we represent joints color-coded according to their dimensionality: red, green and blue circles represent* 3, 2 *and* 1 *degrees of freedom respectively. The root joint is represented in white since it is represented with a* 4*D quaternion.*

subtle than the ones in classic retargeting experiments, like asymmetric limb proportions [LYG15, Gle98]. To capture those realistic shapes we use data from the Dyna dataset [PMRMB15], which registers real body scans (Fig. 1(a)) and normalizes their pose (Fig. 1(b), data provided by the authors). Those pose-normalized meshes are pre-segmented into parts as defined by the SMPL body model [LMR*15].

To that end, given a mesh representing a human body in a canonical T-pose, we approximate the body shape in terms of simple geometric primitives. Here we use a capsule (i.e. extruded sphere) representation as illustrated in Fig. 1 (c), which approximates the volume of the original mesh. Capsules are one of the most common representations in game engines due to their efficiency for collision detection. Our representation consists of one capsule per body part in the SMPL template, excluding the finger segments. The position of each capsule in the zero pose is determined by the joint location of the corresponding part in the Skinned Multi-Person Linear Model (SMPL); see Eq. (10) in [LMR*15]. By having a one-to-one relationship between capsules and body parts, the kinematic structure in the SMPL body model is directly applicable to our capsules. The orientation of the capsules is set according to the principal direction of variation in the template body part. Furthermore, we optimize the radius and length of each capsule so that it adapts to the individual physique of each body shape.

Given a triangulated mesh template defined in terms of vertices $\mathbf{T}$, which generate a continuous surface mesh $\mathcal{T}$, our task is to compute the radius $\mathbf{r}$ and length $\mathbf{l}$ of all capsules to minimize an energy

$$E(\mathbf{r},\mathbf{l}) = \sum_{\mathbf{v_t}\in\mathbf{T}} \min_{\mathbf{p_c}\in\mathcal{C}(\mathbf{r},\mathbf{l})} \|\mathbf{v_t}-\mathbf{p_c}\| + \lambda \sum_{\mathbf{v_c}\in\mathbf{C}(\mathbf{r},\mathbf{l})} \min_{\mathbf{p_t}\in\mathcal{T}} \|\mathbf{v_c}-\mathbf{p_t}\|. \quad (10)$$

The vectors of radii and lengths, $\mathbf{r}$ and $\mathbf{l}$, generate a set of 3D vertices, $\mathbf{C}$, that form a union of capsules, corresponding to a continuous surface $\mathcal{C}$. This objective penalizes the distance from every template vertex $\mathbf{v_t}$ to the capsule's surface $\mathcal{C}$, and the distance from every capsule vertex $\mathbf{v_c}$ to the template surface $\mathcal{T}$. The contribution of each distance is weighted by a parameter λ that is empirically set to 0.5. The optimal capsule radii and lengths are obtained through dogleg gradient-based optimization. Hands and feet are poorly approximated by capsules. Therefore, we model feet with parallelepipeds and hands with ellipsoids.

After this optimization, our capsule model resembles the original body template (see Fig. 1(c)) and can be kinematically animated using the same pose representation as in SMPL, since they share joint locations and kinematic structure. However, the parameterization of an SMPL pose (a total of 66 degrees of freedom from 21 three-dimensional joints plus translation), is inefficient for physical simulation. We reduce the articulation of the knees and toes to 1D joints, and the elbows to 2D joints as it is common in other systems [LYvdP*10, ABDLH13], resulting in a total of 57 parameters (root rotation is represented with a quaternion). In order to use poses with the SMPL parameterization as observed motion $\bar{\mathbf{q}}$, the poses are converted into Euler angles and adapted to this reduced parameterization. Since both representations can be easily transformed into per-part global transformation matrices (see Eq. 4 in [LMR*15]) we compute the (reduced) Euler angle representation whose global transformations are closest (in terms of Frobenius norm) to the global per-part transformations from the original axis-angle representation.

One of the benefits of using a body representation based on geometric primitives is the efficiency of collision computation. Our capsule shape optimization does not prevent collisions, which is beneficial in order to obtain a better approximation of the shape (see for example capsules in the torso or thighs). Therefore, collision detection is disabled in the body parts that are colliding in the rest pose.

Assuming that the human body is composed by a single material of homogeneous density, we assign a weight to each capsule proportional to its volume minus the volumetric intersection with all its parents (so that volume is not double counted).

4. Robust Motion Tracking

We have seen how to optimize a controller that tracks input mocap data. The controller is not robust to disturbances such as external forces, which is necessary for interactive settings. Intuitively, we would like to learn ways of steering disturbed motions back to the stable control model.

More specifically, we build a random tree of LQR controllers across our control solution. The method is inspired by [Ted09] and [ABvdPF17]. The Lyapunov functions approach in [Ted09] currently does not scale to high-dimensional problems. Instead of the PD controllers used in [ABvdPF17], we use LQR controllers, which

provide locally optimal feedback gains that take into account the coupling between all the degrees of freedom, therefore enlarging the region of stability of the controllers. We also show how trees of LQR controllers can be used to robustly track a motion sequence, which is a problem that was not previously tackled.

4.1. LQR Feedback

Let $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]$ and let $\mathbf{u}$ denote the vector of joint torques; given $\mathbf{x}$ we want to solve for the $\mathbf{u}$ that produce the motion. Using the discrete time-system dynamics form, we have

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t). \tag{11}$$

Given an initial trajectory $\mathbf{x}^0_{1:T}$ and $\mathbf{u}^0_{1:T-1}$, let $\tilde{\mathbf{x}}_t = \mathbf{x}_t - \mathbf{x}^0_t$ and $\tilde{\mathbf{u}}_t = \mathbf{u}_t - \mathbf{u}^0_t$. We linearize Eq. (11) around $\tilde{\mathbf{x}}_{1:T}$ and $\tilde{\mathbf{u}}_{1:T-1}$ to obtain a time-varying linear system

$$\tilde{\mathbf{x}}_{t+1} = \mathbf{A}_t \tilde{\mathbf{x}}_t + \mathbf{B}_t \tilde{\mathbf{u}}_t, \tag{12}$$

where $\mathbf{A}_t = \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}^0_t, \mathbf{u}^0_t)$ and $\mathbf{B}_t = \frac{\partial f}{\partial \mathbf{u}}(\mathbf{x}^0_t, \mathbf{u}^0_t)$. These terms are evaluated with numerical differentiation; see Mason et al. [MRS14] for a discussion on differentiating the orientation of the root, which is an element of SO(3). Let J_t refer to the cost-to-go function, which accumulates current and future quadratic costs in our state and control variables

$$J_T(\tilde{\mathbf{x}}_T) = \tilde{\mathbf{x}}_T^T \mathbf{Q}_T \tilde{\mathbf{x}}_T, \tag{13}$$

$$J_t(\tilde{\mathbf{x}}_t, \tilde{\mathbf{u}}_t) = \tilde{\mathbf{x}}_t^T \mathbf{Q}_t \tilde{\mathbf{x}}_t + \tilde{\mathbf{u}}_t^T \mathbf{R}_t \tilde{\mathbf{u}}_t + J_{t+1}(\mathbf{A}_t \tilde{\mathbf{x}}_t + \mathbf{B}_t \tilde{\mathbf{u}}_t), \tag{14}$$

where $\mathbf{Q}_T$, $\mathbf{Q}_t$ and $\mathbf{R}_t$ are given. The $\mathbf{Q}_T$ matrix determines the cost for being at the terminal state $\tilde{\mathbf{x}}_T$, and the $\mathbf{Q}_t$ and $\mathbf{R}_t$ matrices determine the costs for being at state $\tilde{\mathbf{x}}_t$ and using control $\tilde{\mathbf{u}}_t$ at timestep t. It can be shown, e.g., [Ber95, Chapter 4], that the optimal feedback law is given by

$$\tilde{\mathbf{u}}^*_t = \mathbf{L}_t \tilde{\mathbf{x}}_t, \tag{15}$$

or in terms of Eq. (11),

$$\mathbf{u}^*_t = \mathbf{u}^0_t + \mathbf{L}_t(\mathbf{x}_t - \mathbf{x}^0_t). \tag{16}$$

The feedback term $\mathbf{L}_t$ is given by

$$\mathbf{L}_t = -(\mathbf{B}_t^T \mathbf{K}_{t+1} \mathbf{B}_t + \mathbf{R}_t)^{-1} \mathbf{B}_t^T \mathbf{K}_{t+1} \mathbf{A}_t, \tag{17}$$

where $\mathbf{K}_t$ is the solution of the discrete time algebraic Riccati equation:

$$\mathbf{K}_T = \mathbf{Q}_T \tag{18}$$

and

$$\mathbf{K}_t = \mathbf{A}_t^T (\mathbf{K}_{t+1} - \mathbf{K}_{t+1} \mathbf{B}_t^T (\mathbf{B}_t^T \mathbf{K}_{t+1} \mathbf{B}_t + \mathbf{R}_t)^{-1} \tag{19}$$

$$\mathbf{B}_t^T \mathbf{K}_{t+1}) \mathbf{A}_t + \mathbf{Q}_t. \tag{20}$$

We linearize the dynamics at each timestep since computing the feedback terms takes less than a minute for all our motions. If computation or memory requirements become too cumbersome, it is possible to linearize less often, e.g., at contact changes. We use the same cost-to-go function for all our motions. For simplicity, we choose (for all timesteps) $\mathbf{Q}$ to be the identity matrix, and $\mathbf{R}$ to be a diagonal matrix with entries 1, except for the torso joints that have entries 15. This means that our state cost penalizes errors across joints equally. The same comment applies for our control cost, except that we penalize errors in the torso more because unrealistically large torques were being used there. It would be valuable to investigate better cost functions, so that state errors in the more important joints for the task incur a larger cost. For example, in a walking motion, errors in the hips are more critical than in the wrists.

4.2. Trajectory-Tracking-LQR-Trees

In Sec. 3, we showed how to synthesize a trajectory $\mathbf{x}^0_{1:T}$ and $\mathbf{u}^0_{1:T-1}$ that tracks mocap data. Our goal is to stay close to this trajectory when unexpected disturbances occur. Offline, we provide examples of disturbances. Each time a perturbed state fails to stay close to $\mathbf{x}^0_{1:T}$, we attempt to steer it using trajectory optimization towards a region where it will succeed (a region covered by an LQR controller). If the connection is successful, LQR control is used on the newly optimized trajectory, hence making it more likely that we will be prepared for other disturbances. The LQR-Tree is a random tree of LQR controllers that attempts to cover as much as possible of a domain of interest in state space. The tree is constructed offline, but it can then be used for control in near real-time. Let Γ be a tree of tuples $(\mathbf{x}, \mathbf{u}, \mathbf{L}, p)$, where $\mathbf{x}$ is a state, $\mathbf{u}$ is a feedforward controller, $\mathbf{L}$ is a feedback controller, and p is a pointer to the parent node. We refer to the node with state $\mathbf{x}$ as node $\mathbf{x}$ for brevity. We start by stabilizing the trajectory $\mathbf{x}^0_{1:T}$ and $\mathbf{u}^0_{1:T-1}$ with LQR, and then add the corresponding nodes to Γ, where the parent of node $\mathbf{x}^0_k$ is $\mathbf{x}^0_{k+1}$. We construct random subtrees at regular intervals h across the sequence $\mathbf{x}^0_{1:T}$. A smaller interval makes a more robust controller, but is more computationally expensive. It is possible to perform a first pass with interval h, and then a second pass with $h/2$, and so on.

Let χ_k denote the set of states from which we can track $\mathbf{x}^0_{k:T}$ accurately enough, i.e., according to a user-chosen threshold ζ and distance metric dist (e.g., based on the terms inside the summation in Eq. (3), (4), (5) and (7)). Let $\mathcal{L}$ measure the difference between two kinematic trajectories $\mathbf{x}^1_{1:d}$ and $\mathbf{x}^2_{1:d}$:

$$\mathcal{L}(\mathbf{x}^1_{1:d}, \mathbf{x}^2_{1:d}) = \sum_{i=1}^{d-1} \text{dist}(\mathbf{x}^1_i, \mathbf{x}^2_i)/(d-1) + w_l \text{dist}(\mathbf{x}^1_d, \mathbf{x}^2_d), \tag{21}$$

where w_l is used to emphasize errors in the last state (e.g., $w_l = 10/(d-1)$). We consider that state $\mathbf{x}_1$ is inside χ_k if we have a control solution with simulated trajectory $\mathbf{x}_{1:M}$ such that $\mathcal{L}(\mathbf{x}_{M+k-T:M}, \mathbf{x}^0_{k:T}) < \zeta$. M denotes the number of timesteps required to reach the root $\mathbf{x}^0_T$ when passing through node $\mathbf{x}^0_k$, therefore $T - k + 1 \leq M$.

The Trajectory-Tracking-LQR-Tree method systematically steers states originally outside χ_k inside it. The method samples random states as follows. For a state $\mathbf{x}^0_k$ in the sequence, we apply an external force in a random direction, location and magnitude on the character for a given duration. Let F refer to a list that specifies the maximum magnitude of the external force, e.g., $F = [100\,\text{N}, 200\,\text{N}, 300\,\text{N}, \ldots]$. We make the controller progressively more robust by first sampling states with $F(1)$, then $F(2)$, etc. We currently manually choose the set of forces, where the difference between successive forces is usually 50 or 100 N. It would

be valuable to adapt the set of forces based on the motion being performed. For example, we could evaluate the ratio of states that can track the sequence after applying a perturbation. If the ratio is too high, then the forces applied are too low, and vice-versa.

Let $\mathbf{x}_k^p$ denote a perturbed state outside the sets $\chi_{k-\alpha},\dots,\chi_k,\dots,\chi_{k+\alpha}$, which we denote by $\chi_{k-\alpha:k+\alpha}$. Note that $\mathbf{x}_k^p$ is the state obtained after applying a disturbance for a given duration. We solve a trajectory optimization problem to attempt to connect $\mathbf{x}_k^p$ back to $\chi_{k-\alpha:k+\alpha}$, where $\alpha \in \mathbb{N}$ modulates how important it is to start tracking near timestep k (the start of the disturbance). This is done by choosing a nearby node $\mathbf{x}^n$ in the tree Γ that is connected to node $\mathbf{x}_j^0$, where $j \in [k-\alpha, k+\alpha]$. The trajectory optimization searches for a new trajectory $\mathbf{x}_{1:N}^c, \mathbf{u}_{1:N-1}^c$ with CMA such that $\mathcal{L}(\mathbf{x}_{M+j-T:M}, \mathbf{x}_{j:T}^0)$ is minimized after using the LQR controllers starting from timestep N in the rollouts (i.e., finite-duration simulations) of the branch from node $\mathbf{x}^n$ to the root $\mathbf{x}_T^0$. In this work, we set N to 20 and the number of spline knots to 3. We stop the optimization when the cost function has a value less than ζ. For long mocap sequences, we have found that we could optimize the less expensive objective $\mathcal{L}(\mathbf{x}_{M+j-T:M-\lambda}, \mathbf{x}_{j:T-\lambda}^0)$, where $\lambda \in \mathbb{N}$. This means that our control solutions are optimized to track a portion of the mocap sequence only. This does not cause difficulty if there are subsequent subtrees in the remainder of the sequence, as will be discussed in the next paragraph. If $\mathbf{x}_N^c \in \chi_{k-\alpha:k+\alpha}$, we now have $\mathbf{x}_k^p \in \chi_{k-\alpha:k+\alpha}$ by performing LQR stabilization on $\mathbf{x}_{1:N}^c, \mathbf{u}_{1:N}^c$ and adding the new nodes to Γ (see Fig. 2). We then repeat the process by sampling a new state. In this work, we construct a random tree, not a rapidly exploring random tree as in [Ted09] nor a dense random tree as in [ABvdPF17]. However, the algorithm is compatible with these approaches. The method is described in Algorithm 1.

We now discuss how we determine if a state $\mathbf{x}_k^r$ is inside $\chi_{k-\alpha:k+\alpha}$ (lines 15 and 18). We start by finding the κ nearest nodes to $\mathbf{x}_k^r$ in Γ that are connected to a node in $\mathbf{x}_{k-\alpha:k+\alpha}^0$. We perform rollouts (moving in the root direction) using the LQR controllers associated with each of these nodes. For near real-time control, the rollouts should have short durations (e.g., 3 s), so we do not necessarily reach the root. We say that a method is near real-time when the computation time for a motion is less than the execution time. We evaluate $\mathcal{L}$ for each of the rollouts and choose the control solution with the smallest value, i.e., the solution that tracks a portion of the mocap sequence better. We execute this solution and move forward in the sequence, and then repeat this process for new state (at a later timestep), until the root is reached. In this work, we chose $\kappa = 20$ and the rollouts can be performed in parallel. The computation time for a 3 s mocap sequence is about 0.15 s on our computer with four cores. Note that by this process, a state $\mathbf{x}_k^r$ that is outside the domain of attraction of the nearest subtree can still be brought to track the mocap sequence at a later timestep (see Fig. 3). This is the reason we start by constructing the subtrees at the end of the sequence and move sequentially to the start, i.e., doing so increases the likelihood of a successful trajectory optimization.

Algorithm 1 Tracking-LQR-Tree($\mathbf{x}_{1:T}^0$, $\mathbf{u}_{1:T-1}^0$, $\mathbf{Q}$, $\mathbf{R}$, F, α)

1: $[\mathbf{A}_{1:T-1}, \mathbf{B}_{1:T-1}] \leftarrow$ Linearize $f(\mathbf{x},\mathbf{u})$ around $(\mathbf{x}_{1:T}^0, \mathbf{u}_{1:T-1}^0)$
2: $\mathbf{L}_{1:T-1} \leftarrow$ LQR($\mathbf{A}_{1:T-1}$, $\mathbf{B}_{1:T-1}$, $\mathbf{Q}$, $\mathbf{R}$)
3: Γ.add_node($\mathbf{x}_T^0$, 0, 0, NULL)
4: ...
5: $p \leftarrow$ pointer to the node with $\mathbf{x}_{i+1}^0$
6: Γ.add_node($\mathbf{x}_i^0$, $\mathbf{u}_i^0$, $\mathbf{L}_i$, p)
7: ...
8: $p \leftarrow$ pointer to the node with $\mathbf{x}_2^0$,
9: Γ.add_node($\mathbf{x}_1^0$, $\mathbf{u}_1^0$, $\mathbf{L}_1$, p)
10: **for** $k = T-1 : h : 1$ **do**
11: **for** f_{max} in F **do**
12: **for** $\lambda = 1 : \Lambda$ **do**
13: Apply random external force of max magnitude f_{max} on state $\mathbf{x}_k^0$
14: $\mathbf{x}^p \leftarrow$ the perturbed state
15: **if** $\mathbf{x}^p \notin \chi_{k-\alpha:k+\alpha}$ **then**
16: $\mathbf{x}^n \leftarrow$ nearby node connected to a node in $\mathbf{x}_{k-\alpha:k+\alpha}^0$
17: Solve a trajectory optimization to connect $\mathbf{x}^p$ to $\chi_{k-\alpha:k+\alpha}$
18: **if** $\mathbf{x}_N^c \in \chi_{k-\alpha:k+\alpha}$ **then**
19: $(\mathbf{x}_{1:N}^c, \mathbf{u}_{1:N-1}^c) \leftarrow$ the synthesized trajectory
20: $[\mathbf{A}_{1:N-1}, \mathbf{B}_{1:N-1}] \leftarrow$ Linearize $f(\mathbf{x},\mathbf{u})$
21: $\mathbf{L}_{1:N-1} \leftarrow$ LQR($\mathbf{A}_{1:N-1}$, $\mathbf{B}_{1:N-1}$, $\mathbf{Q}$, $\mathbf{R}$)
22: $p \leftarrow$ pointer to the node that causes $\mathbf{x}_N^c \in \chi_{k-\alpha:k+\alpha}$
23: Γ.add_node($\mathbf{x}_{N-1}^c$, $\mathbf{u}_{N-1}^c$, $\mathbf{L}_{N-1}$, p)
24: ...
25: $p \leftarrow$ pointer to the node with $\mathbf{x}_2^c$
26: Γ.add_node($\mathbf{x}_1^c$, $\mathbf{u}_1^c$, $\mathbf{L}_1$, p)

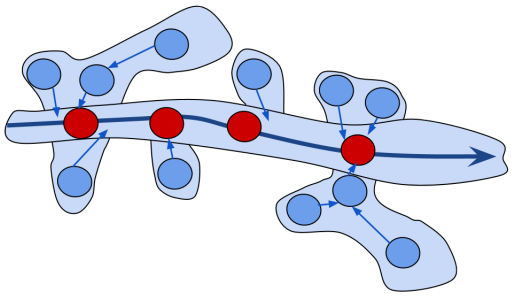

Figure 2: ***Trajectory-Tracking-LQR-Tree****. The blue region illustrates χ. The region is filled with LQR feedback. The long thick arrow is $\mathbf{x}_{1:T}^0$ and the red circles are the root nodes of the subtrees across the sequence. The blue circles are sampled states that are successfully added to the tree. As can be seen in the figure, the nodes do not always connect back to the red circles due to the α term.*

5. Results

To evaluate our method we need movement data from people of different sizes and shapes. To that end we use two sources of data. The first comes from the public Dyna dataset [Dyn15], which contains body shapes and motions of ten subjects with very different body shapes. While most available mocap data in the world is captured from relatively fit people, Dyna also contains motion of heavy peo-

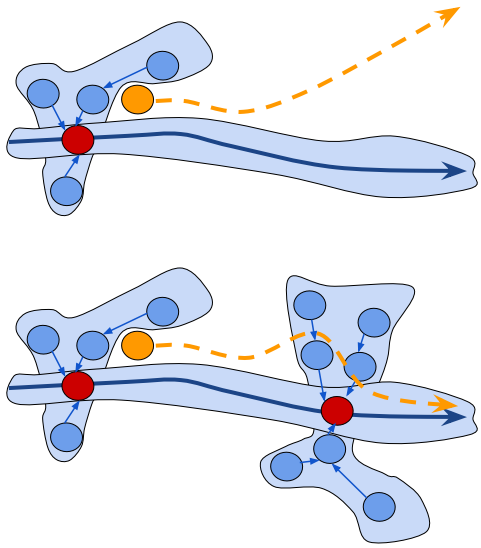

Figure 3: ***Feedback across a sequence****. The orange circle is a perturbed state outside the domain of attraction of the nearest subtree. The state diverges away from the mocap if only the controls of the nearest subtree are used (top figure). The state can be steered back to track the mocap if it enters the domain of attraction of another subtree later in the sequence (bottom figure).*

ple. We also use traditional motion capture data [cmu00] for which we have no ground truth 3D shape information. For this data we use MoSh to extract the 3D body shapes and SMPL-compatible poses [LMB14].

We estimate the controller for the ground truth 3D shape for each sequence. In the **Supplemental Video**, this *source* body is always shown in red. When the target character has the same (estimated) shape as the source character, it does not always perfectly track the motion. This is due to differences in the joint and torque limits, errors in the mocap data itself, local minima in the optimization, and differences between our articulated rigid body and the human body. Nevertheless, the resulting motions approximate the mocap, look believable, and are driven by a physical controller that has many advantages over simple retargeting as we show below.

We vary the body shapes among 10 different shapes from the Dyna dataset and optimize their motion controller. These *target* bodies are shown in green. Given this data we mix and match source and target motions, for example taking the motion of a light person and animating a heavy person and vice versa. Our evaluation is visual and, for this purpose, we provide many examples in the **Supplemental Video**.

The supplemental video shows different characters performing a turning-twist motion. Note that the heavier character stretches its arms more and, interestingly, places it legs in a different configurations than the other characters; to turn-twist, it pivots around one foot instead of moving both feet. We also note that the lighter character has trouble keeping its arms at the same angle as the mocap character and its more bouncy during the motion, while the character with the original shape performs better overall.

When the characters have similar shape, the difference in their motion is subtle (e.g., a slightly lighter character being more bouncy or a slightly heavier character bending more to perform a movement). When the characters have vastly different shapes, the difference can be drastic, even leading to tracking failures. When tracking fails, it does so in a realistic way, similar to how one would expect a person fail to perform the motion. We, however, note that the heavier characters tend to have excessive motion in the lumbar spine when performing recovery movements to stay in balance.

Consider, for example, the dancing motion in Fig. 4. Here the source character is substantially slimmer than the target. It is therefore able to crouch lower and move quicker than the heavy character. To avoid losing balance, the heavy character takes a few steps forward to stabilize itself, and then takes a few steps backwards to get back near the mocap motion. What is interesting is that the steps taken have a natural look and they emerge entirely from the optimization. Imposing foot contact constraints, as is custom in classic spacetime constrained optimization [Gle98], would have discouraged this natural behavior.

We also present the results for a challenging air-kick motion by a slim character in the supplemental video. We show that with high torque limits (± 300 Nm), the heavy character performs the motion reasonably well. The result seems unnatural since we would not expect a heavy character to jump this high. With more severe torque limits, ± 100 Nm and ± 50 Nm respectively, we get a more satisfying result: the character jumps with a lower height than in the former, and just slides on the floor in the latter.

An example showing the importance of the action timing is shown in the supplemental video, where a heavy character tracks two consecutive kicking motions (Fig. 5). The usual time horizon for our optimization is 0.8 s (that is two time windows of 0.4 s each). We see the character losing balance on a kick with the usual time horizon. When the time horizon is progressively increased, the characters improves its motion, for instance by discovering that it cannot stretch its leg back as much as the mocap character.

We also present results where the destination character is holding an object (a stick and a block), while the mocap character does not. The object's weight has a visible effect on the motion; the heavier the object, the more it affects the movement and makes it hard to perform. However, the object can also be helpful to the character; e.g., the stick is used to maintain balance in difficult postures.

We tested the robustness of our solutions by applying on the character random external disturbances that are up to 500 N for 0.1 s. We show how the LQR-tree succeeds in performing the motion whereas the LQR controller fails. In Fig. 6, we compare the reactions to a disturbance on the foot in the air when the character is kicking. As the character is about to fall in one direction, it rotates its arms in the opposite direction, thereby allowing itself to rotate back up and continue performing the motion. We have chosen $\alpha = 25$ timesteps (see Sec. 4.2). A small value of α makes it more difficult to find a solution, while a large value means that we can skip tracking a portion of the sequence. The ankles and the torso movements are at times too jittery when recovering from a

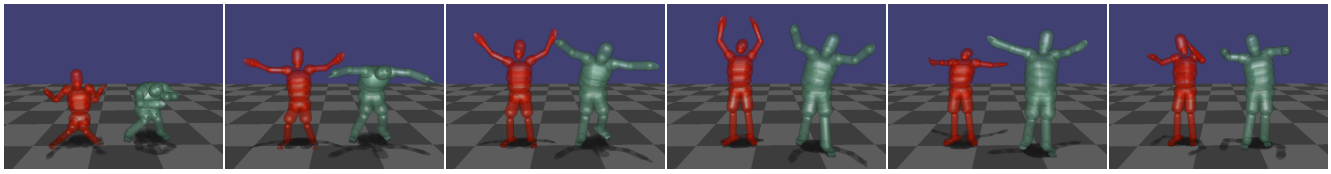

Figure 4: ***Recovery steps.*** *This sequence illustrates how a heavy character can lose balance when getting up too quickly from a crouch position. In order not to lose balance, the character takes a few recovery steps forward. After avoiding the risk of falling, it takes a few steps backwards to return to its desired position, and continue dancing.*

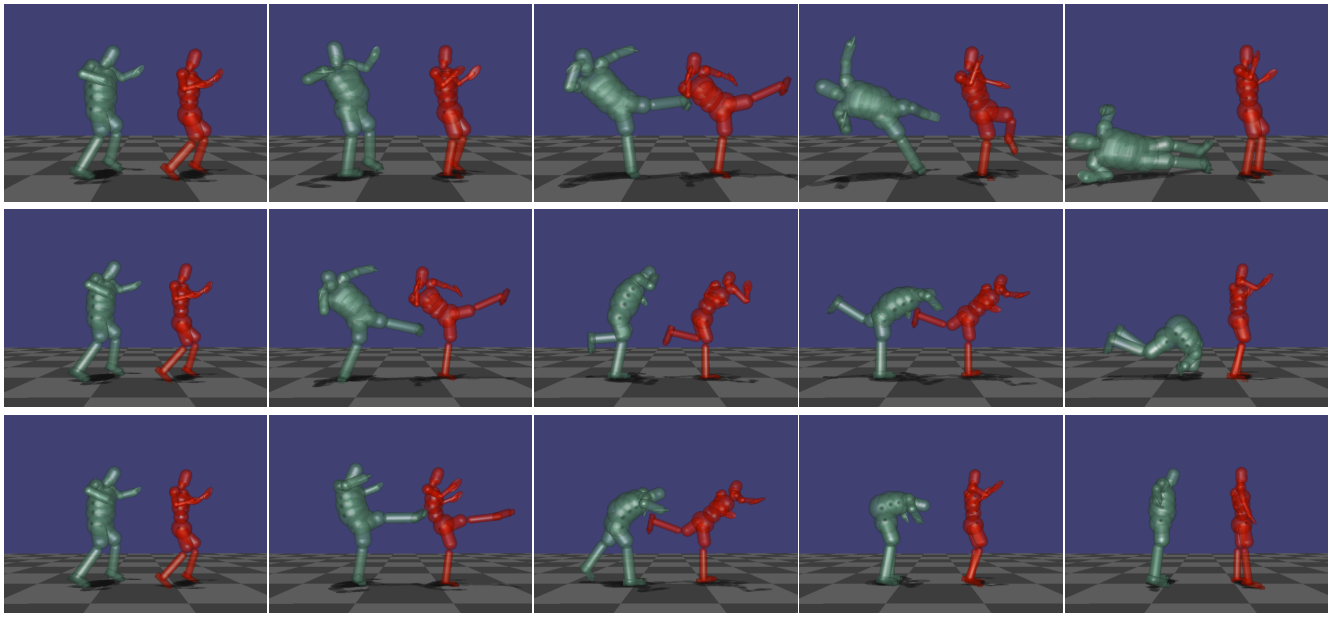

Figure 5: ***Time Horizon.*** *The first sequence uses a 0.8 s window horizon in the optimization, the second 0.9 s, and the third 1.0 s. Note that in the second sequence, the heavy character does not kick as high as in the first sequence to avoid falling, but it still falls when performing the back kick. In the third sequence, the character decides to stretch its leg backwards, but to keep its foot on the floor.*

disturbance. This indicates that tighter torque limits and more severe control costs (see Sec. 4.1), ideally dependent on the subject's shape, are needed on these joints.

In the supplemental video, we also show robustness in a boxing sequence. To obtain a robust controller for this 2.3 s sequence, we build subtrees at 0.1 s intervals with Algorithm 1. We set Λ to 100 and $F = [50, 100, 150]$. Each perturbed state ($\mathbf{x}^p$ in Algorithm 1) that is steered back to the sequence with a trajectory optimization of N timesteps requires $N-1$ nodes to be added to the tree. We provide the numbers of perturbed states and nodes in the subtrees from $k = 230$ to $k = 180$ in Table 1. Some portions of the sequence require far fewer nodes than others. For instance, only 3 perturbed states were added to the subtree at timestep 30, yet the controller is resilient to impulses on the order of 15 Ns. To obtain comparable performance at timestep 90, a subtree with 16 perturbed states is constructed. Constructing the tree for this sequence took two days of computation time. We now discuss the memory requirements. For each node in the tree, we store the vector of feedforward controls of dimension H, the state vector of dimension W and the feedback matrix term L of dimension $H \times W$. A perturbed state added to the tree requires $(N-1) \times (H \times W + H + W)$ entries. In this work, we have $H = 50$, $W = 113$ and $N = 21$. A tree with 500 perturbed states would require 116260 entries, which is less than a megabyte.

Our Trajectory-Tracking-LQR-Trees method shares similarities with the impressive method of [LvdPY16]. Their method divides the sequence in control fragments and linear feedback policies are learned for each fragment. This is akin to our approach of dividing the sequence with subtrees, except that we show how to learn non-linear feedback policies (i.e., we show how linear feedback policies can be connected together to increase robustness). They compute reduced-order linear feedback policies requiring access to a large cluster, while we compute full-order linear feedback policies (i.e., linear quadratic regulators) that are provably optimal and that can be computed on a single computer in a few seconds. Another difference is that their feedback action terms are limited to the waist, the hips and the knees (i.e, perturbations to other body parts must be corrected with motion in these joints), while we allow feedback to occur through all joints. Lastly, they only show robustness to

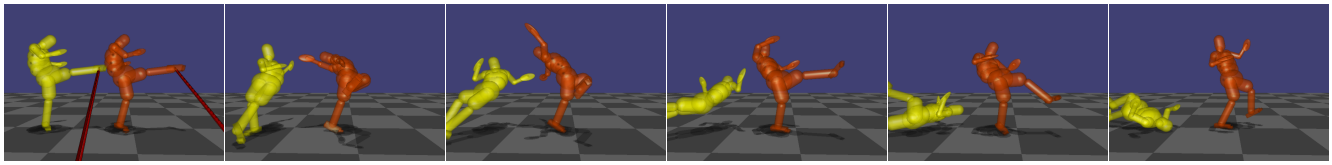

Figure 6: ***Robustness.*** *The red line is an external force applied on the characters. The yellow character has an LQR controller. The orange character has an LQR-tree controller.*

Timestep k	230	220	210	200	190	180
# Perturbed states	7	9	5	17	11	4
# Nodes in subtree	140	180	100	180	220	80

Table 1: *Tree statistics. The table provides the number of perturbed states* $\mathbf{x}^p$ *that are steered back to the sequence and the total number of nodes in the subtrees at various timesteps. This corresponds to the boxing mocap in the supplemental video.*

Random Impulse (Ns)	5	50	75	125	250
# Successful PD (100 trials)	3	4	0	1	0
# Successful LQR (100 trials)	89	28	20	8	5

Table 2: *PD vs LQR. The table compares the number of times a PD controller and a LQR controller can track a standing balance motion when increasingly larger impulses are applied on the character.*

external disturbances applied to the character's trunk along its facing direction, while we show robustness to disturbances in random locations and directions on the character. A promising topic for future work is to merge both methods by building random trees with linear feedback policies obtained with linear regression.

We perform the following experiment to evaluate the difference between PD and LQR control. Given a standing balance motion of 0.8 s, we compare in Table 2 the number of times out of 100 trials that a single PD and LQR controller can withhold and continue successfully tracking the sequence when an impulse is applied on the first timestep. The data shows that the LQR controller has a success rate of 89% compared to 3% for the PD controller given a 5 Ns impulse. The LQR controller is able to withstand an impulse disturbance fifty times larger than the PD controller before degrading to a similar success rate. This explains why [ABvdPF17] reports PD-Trees with hundreds of perturbed states to be robust to impulses of about 10 Ns, while our LQR-Trees are an order of magnitude less in size for the same robustness.

6. Discussion

The experiments demonstrate that our method produces different motions for different body shapes, that these motions are plausible and appropriate for the shape, and that our new LQR-tree formulation is robust to external forces.

One of our experiments also highlights several limitations of our approach. In the supplemental video, we show how a thin character tracks the running-in-place motion of a heavy character, and how a heavy character tracks the running-in-place motion of a thin character. We note that the thin character stretches its arms too widely, trying to imitate the heavy character. But while the heavy character has a good reason to stretch its arms (i.e. it helps it maintain balance), the light character is more stable and does not need to do this. Consequently the motion lacks perceptual believability, but we note that this might also be the intention of a motion capture session.

With multiple examples of the same motion by people of different shapes, one could develop a method to determine which portions of a motion should be tracked closely. We also note that the heavy character has difficulty running at the same pace as the thin character. A method could be developed to adjust the timing of the motion, so that a heavy character could run at a more comfortable pace.

Using LQR controllers instead of the PD controllers in [ABvdPF17] makes the process of finding robust solutions more efficient, as a single LQR controller often replaces dozens of PD controllers. This comes at a very modest computational cost (computing the gains), but at a memory cost (storing the gains), which becomes an issue as the trees grow larger.

One application of our work with objects is in the film industry. For example, when actors simulate a fighting scene with swords, sticks, batons, etc., they are not usually given actual weapons, but replicas that have very little weight. In the final motion, the actor looks like they are holding a massive, heavy object, but the motion may not reflect the weight of the object.

The cases where the physical character departs from the mocap motion (e.g., by losing balance) are very interesting. Our method provides indicators on which portions of a motion will be difficult for people of different shape and strength. When combined with a more detailed musculoskeletal model, it could be useful in healthcare, e.g., to predict how an elderly person's gait would change with a prosthesis. Increasing the time horizon could provide a way to model how people learn motor skills. When attempting to learn a motor skill (e.g. karate, yoga, aerobic exercise, dance, etc.), a novice watches an expert perform the movement, and then attempts to replicate it. Since the novice is not fully aware of his own limitations, he misses here and there, stumbles and falls over. Next time, he has a better idea of what he can and cannot perform. Instead of following the expert exactly, he adapts the experts movements to his own body. In other words, it is as if the novice has a longer horizon. By varying the time horizon, we have a way to model how the motion changes as the performer moves from being a novice to

an expert. It would be very interesting to compare our results with experimental data.

In summary, we have presented a method to take motion capture data from a source person and transfer it realistically to different target bodies in a way that is perceptually realistic and can adapt to external forces. The results suggest that this physics-based approach is viable for generating varied and realistic animations from mocap data.

References

[ABDLH13] AL BORNO M., DE LASA M., HERTZMANN A.: Trajectory optimization for full-body movements with complex contacts. *IEEE Transactions on Visualization and Computer Graphics 19*, 8 (2013), 1405–1414. 3, 4, 5

[ABvdPF17] AL BORNO M., VAN DE PANNE M., FIUME E.: Domain of attraction expansion for physics-based character control. *ACM Transactions on Graphics 36*, 17 (2017). 2, 3, 5, 7, 10

[ASK*05] ANGUELOV D., SRINIVASAN P., KOLLER D., THRUN S., RODGERS J., DAVIS J.: Scape: shape completion and animation of people. *ACM Transactions on Graphics 24*, 3 (2005), 408–416. 3

[Ber95] BERTSEKAS D.: *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995. 6

[cmu00] CMU graphics lab motion capture database. http://mocap.cs.cmu.edu, 2000. Accessed: 2012-12-11. 8

[DLvdPY15] DING K., LIU L., VAN DE PANNE M., YIN K.: Learning reduced-order feedback policies for motion skills. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2015), ACM, pp. 83–92. 3

[dSAP08] DA SILVA M., ABE Y., POPOVIĆ J.: Interactive simulation of stylized human locomotion. *ACM Transactions on Graphics 27*, 3 (2008), 82. 3

[DSDP09] DA SILVA M., DURAND F., POPOVIĆ J.: Linear bellman combination for control of character animation. *Transactions on Graphics 28*, 3 (2009), 82. 3

[Dyn15] Dyna dataset. http://dyna.is.tue.mpg.de/, 2015. 7

[FLFM15] FRAGKIADAKI K., LEVINE S., FELSEN P., MALIK J.: Recurrent network models for human dynamics. In *ICCV* (2015), IEEE, pp. 4346–4354. 2

[Gle98] GLEICHER M.: Retargetting motion to new characters. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 33–42. 2, 5, 8

[Han06] HANSEN N.: The CMA Evolution Strategy: A Comparing Review. In *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*. 2006, pp. 75–102. 4

[HENN16] HAN D., EOM H., NOH J., NOH J.: Data-guided model predictive control based on smoothed contact dynamics. *Computer Graphics Forum 35*, 2 (2016). 2, 3

[HET*14] HÄMÄLÄINEN P., ERIKSSON S., TANSKANEN E., KYRKI V., LEHTINEN J.: Online motion synthesis using sequential monte carlo. *ACM Transactions on Graphics 33*, 4 (2014), 51. 3

[HKT10] HO E. S., KOMURA T., TAI C.-L.: Spatial relationship preserving character motion adaptation. *ACM Transactions on Graphics 29*, 4 (2010), 33. 2

[HP97] HODGINS J. K., POLLARD N. S.: Adapting simulated behaviors for new characters. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 153–162. 1, 3

[HRL15] HÄMÄLÄINEN P., RAJAMÄKI J., LIU C. K.: Online control of simulated humanoids using particle belief propagation. *ACM Transactions on Graphics 34*, 4 (2015), 81. 3

[KSG02] KOVAR L., SCHREINER J., GLEICHER M.: Footskate cleanup for motion capture editing. In *Proceedings of the 2002 ACM SIGGRAPH Symposium on Computer Animation* (New York, 2002), Spencer S. N., (Ed.), ACM Press, pp. 97–104. 1

[LBJK09] LAU M., BAR-JOSEPH Z., KUFFNER J.: Modeling spatial and temporal variation in motion data. *ACM Transactions on Graphics 28*, 5 (2009), 171. 2

[LHP05] LIU C. K., HERTZMANN A., POPOVIĆ Z.: Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics 24*, 3 (2005), 1071–1081. 1, 2

[LKL10] LEE Y., KIM S., LEE J.: Data-driven biped control. *ACM Transactions on Graphics 29*, 4 (2010), 129. 3

[LMB14] LOPER M. M., MAHMOOD N., BLACK M. J.: MoSh: Motion and shape capture from sparse markers. *ACM Transactions on Graphics 33*, 6 (2014), 220:1–220:13. 3, 8

[LMR*15] LOPER M., MAHMOOD N., ROMERO J., PONS-MOLL G., BLACK M. J.: SMPL: A skinned multi-person linear model. *ACM Transactions on Graphics 34*, 6 (2015), 248:1–248:16. 2, 3, 5

[LMT08] LYARD E., MAGNENAT-THALMANN N.: Motion adaptation based on character shape. *Computer Animation and Virtual Worlds 19*, 3-4 (2008), 189–198. 2

[LvdPY16] LIU L., VAN DE PANNE M., YIN K.: Guided learning of control graphs for physics-based characters. *ACM Transactions on Graphics 35*, 3 (2016). 3, 9

[LWH*07] LIU S., WANG C. C. L., HUI K.-C., JIN X., ZHAO H.: Ellipsoid-tree construction for solid objects. In *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling* (New York, NY, USA, 2007), SPM '07, ACM, pp. 303–308. 3

[LYG15] LIU L., YIN K., GUO B.: Improving sampling-based motion control. *Computer Graphics Forum 34*, 2 (2015), 415–423. 1, 2, 3, 5

[LYvdP*10] LIU L., YIN K., VAN DE PANNE M., SHAO T., XU W.: Sampling-based contact-rich motion control. *ACM Transactions on Graphics 29*, 4 (2010), 128:1–128:10. 2, 5

[MBBT00] MONZANI J.-S., BAERLOCHER P., BOULIC R., THALMANN D.: Using an intermediate skeleton and inverse kinematics for motion retargeting. *Computer Graphics Forum 19*, 3 (2000), 1067–7055. 2

[MDB17] MOLLA E., DEBARBA H. G., BOULIC R.: Egocentric mapping of body surface constraints. *IEEE Transactions on Visualization and Computer Graphics 24*, 7 (2017), 2089–102. 2

[MLPP09] MUICO U., LEE Y., POPOVIĆ J., POPOVIĆ Z.: Contact-aware nonlinear control of dynamic characters. In *ACM SIGGRAPH 2009* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 81:1–81:9. 3

[MPP11] MUICO U., POPOVIĆ J., POPOVIĆ Z.: Composite control of physically simulated characters. *ACM Transactions on Graphics 30*, 3 (2011), 16. 3

[MRS14] MASON S., RIGHETTI L., SCHAAL S.: Full dynamics lqr control of a humanoid robot: An experimental study on balancing and squatting. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on* (2014), IEEE, pp. 374–379. 3, 6

[MTP12] MORDATCH I., TODOROV E., POPOVIĆ Z.: Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics 31*, 4 (2012), 43. 2

[MuJ16] MuJoCo advanced physics simulation. http://www.mujoco.org/, 2016. 4

[PMRMB15] PONS-MOLL G., ROMERO J., MAHMOOD N., BLACK M. J.: Dyna: A model of dynamic human shape in motion. *ACM Transactions on Graphics 34*, 4 (2015), 120:1–120:14. 3, 5

[PW99] POPOVIĆ Z., WITKIN A.: Physically based motion transformation. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 11–20. 2

[SHP04] SAFONOVA A., HODGINS J. K., POLLARD N. S.: Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics 23*, 3 (2004), 514–521. 2

[SMOT15] SRIDHAR S., MUELLER F., OULASVIRTA A., THEOBALT C.: Fast and robust hand tracking using detection-guided optimization. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)* (2015), IEEE, pp. 3213–3221. 3

[SP05] SULEJMANPAŠIĆ A., POPOVIĆ J.: Adaptation of performed ballistic motion. *ACM Transactions on Graphics 24*, 1 (2005), 165–179. 2

[Ted09] TEDRAKE R.: Lqr-trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems (RSS)* (2009). 2, 3, 5, 7

[TET12] TASSA Y., EREZ T., TODOROV E.: Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* (2012), IEEE, pp. 4906–4913. 2

[TGB13] THIERY J.-M., GUY É., BOUBEKEUR T.: Sphere-meshes: shape approximation using spherical quadric error metrics. *ACM Transactions on Graphics 32*, 6 (2013), 178. 3

[TPT16] TKACH A., PAULY M., TAGLIASACCHI A.: Sphere-meshes for real-time hand modeling and tracking. *ACM Transactions on Graphics 35*, 6 (2016), 222. 3

[WK88] WITKIN A. P., KASS M.: Spacetime constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1988), SIGGRAPH '88, ACM, pp. 159–168. 1, 2

[YL10] YE Y., LIU C. K.: Synthesis of Responsive Motion Using a Dynamic Model. *Computer Graphics Forum 29*, 2 (2010), 555–562. 3

[ZVDH03] ZORDAN V. B., VAN DER HORST N. C.: Mapping optical motion capture data to skeletal motion using a physical model. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), SCA '03, Eurographics Association, pp. 245–250. 2

[ZYH*15] ZHOU Y., YIN K., HUANG H., ZHANG H., GONG M., COHEN-OR D.: Generalized cylinder decomposition. *ACM Transactions on Graphics 34*, 6 (2015), 171. 3