Master Thesis Game and Media Technology

Live Control of a Physically Plausible Virtual Biped, Using the Kinect

Author:
Olaf Zwennes

Supervisors:
dr. Thomas Geijtenbeek
dr. ir. Frank van der Stappen

Thesis number:
ICA-3858936

Department of Information and Computing Sciences
Faculty of Science, Utrecht University

February 20, 2014

Abstract

With the availability of affordable motion capture solutions, like the Microsoft Kinect, live data–driven control of a virtual character has become more relevant. In order to provide both intuitive control of a virtual biped, as well as grounding that character in the virtual environment, we propose a system for real–time control of a physically plausible virtual biped using the Microsoft Kinect sensor. In order to deal with the varying skeletal dimensions present in Kinect data, we propose an inverse kinematics based real–time data processing method, which avoids positional errors of the limbs in the virtual character, while maintaining the pose present in the Kinect data. In order to allow for arbitrary in–place motions to be performed by the physically plausible character without losing balance, we combine proportional derivative controllers with a simple balance strategy that controls the center of mass of the character by rotating the ankle joints. Experimental results show that with a limited range of balanced in–place motions, our system is able to reproduce these motions with the physics simulated character in close to real–time. Our system achieves this using solely physically plausible internal torques and using the Kinect data stream as its only motion data input.

Keywords: real–time human character animation, Microsoft Kinect, motion capture, data–driven control, physically based animation

Contents

1 Introduction

With affordable consumer hardware capable of live markerless motion capture, like the Microsoft Kinect or the Asus Xtion, a lot of research has gone into finding novel and valuable uses for the motion data generated. Examples are the use of the Kinect for physical rehabilitation [CCH11] and gesture recognition [LKI12]. One intuitive use can be found in controlling the movement of a virtual character, by tracking the movement of the person in front of the sensor, from now on referred to as the *user*, to a virtual biped. However, allowing a user to perform any motion, means the virtual character will have to be able to react naturally in a virtual environment, regardless of its motion. Even in the simplest virtual environment, like an infinite flat floor, a random motion can look disconnected from this environment. For instance, when a characters feet are not firmly on the floor during the motion. Physics–based animation allows a virtual character to track a motion whilst staying grounded within the environment, by interacting with it using plausible physical collisions and forces.

There are several ways for a physically simulated character to use forces and torques to track a reference motion [GP12]. There are two types of forces: external and internal forces. External forces are a result of an interaction with the virtual environment, or are artificially applied to the physics character to control it. In the latter case the force is not physically correct, as there is no counteracting force acting on the virtual environment. Internal forces are defined as the forces resulting from the movement or limits of the joints of the physics character. An example of this is the use of torques applied to body parts to simulate muscle activation, resulting in the bending of a joint.

Many approaches use physically inaccurate external forces to track the reference motions or to assist in positioning and balancing the character within the virtual world [LZ11] [SH12]. Restricting the method to only use internal torques applied to the joints, gives a physically plausible simulation. Additionally, the virtual character has an unprecedented freedom in showing physically grounded motions. For instance, it could lose its balance as a result of some motion or external perturbation and fall over. Depending on the application of this kind of approach, the fact that the reference motion is only loosely tracked, can either be a benefit or an issue.

An application has to handle the unpredictable behavior. If it does, benefits can be found in this dynamic and real–time control of a virtual character. An example of this is a video game where the player has to fight other characters, pushing them over without falling over himself. This way, the application uses the physically grounded motion control as a game mechanic. Another example is simulating weightlessness, where a virtual character experiences zero gravity, and the user controls that character in its interactions with the environment. In this scenario use of external forces to control the character would invalidate the simulation. Therefore, restricting the character control to internal torques, is a requirement for an application like this.

From these applications, we define the following goal for our system: *to perform intuitive real–time in–place control of a physically plausible virtual biped using consumer grade video–based motion capture hardware.*

We elaborate this goal further by noting the aim is to 'intuitively control' a virtual character, which means the aim is not to animate the virtual character realistically. Therefore, faithful recreation of the 'performance' of the user, as is the aim with motion capture animation, is neither required nor targeted by our system. Instead, merely having the movements of the virtual biped logically follow the movements of the user is sufficient, as that is an intuitive means to control a virtual character.

Additionally, the term 'physically plausible biped' is used to describe a physically simulated virtual character that moves solely using internally consistent torques applied to the joints of the character, analogous to muscles contracting resulting in movement. Specifically, this means that

external forces on the character can only come from interactions with the environment, and cannot be applied to the character artificially. These artificial forces control a physics character like a puppet and are commonly used in order to simplify control and balance [GP12]. However, we emphasize physically correct character control over simplified control, and therefore disallow these artificial external forces.

Note also that our goal for the system does not specify the shape of the virtual character, other than that it is a biped. Our system allows for the morphology of the character to change, though not necessarily in real–time. Our system allows for control of the virtual character during in–place motions. This both simplifies the problem by not taking into account locomotion, as well as allowing the system to work within the limited capture range of a popular consumer grade motion capture system, the Microsoft Kinect [Mic13]).

We also note that some of the above subgoals, like intuitive control and variable morphology, require a highly adaptive system. This leads us to introduce another requirement for the system, which is that it cannot rely on existing motion data. An example of using existing data is to combine real–time motion data with a matched motion from a pre–processed motion database. This is not uncommon for these types of applications [LZ11] [SH12]. This requirement has an added benefit, because it means a motion database does not have to be constructed and the only motion data used is a single data source, either real–time or pre–recorded. This leads to the last subgoal of our system, which is that the movement control can be performed in real–time. This means all subsystems, from the data processing to the physics simulation, should handle at least 30 frames of motion data per second, which is the rate at which the Microsoft Kinect generates motion data [Mic13].

The previously mentioned markerless motion capture techniques rely on analysis of a depth image to determine spatial information about the body parts of the person in front of the sensor. The single perspective of the sensor means that body parts can be occluded or out of frame, and the image analysis results are less accurate than with a motion capture technique using markers. These drawbacks can result in missing and imperfect pose data [OKO*12]. Fortunately, since the emphasis of the method is on the control of a virtual character, the target motion is not required to be visually plausible. In cases where the sensor is not capable of returning accurate pose information, control by the user is limited. Because the sensor data is the only data source used, this limit is inherent to the chosen approach.

There are also systematic issues with markerless motion capture data that require solutions for our system to be useful. The first issue is that the Kinect sensor returns the absolute positions of a set of joints. The depth image analysis that returns this data does not enforce a fixed distance between adjacent joints in the skeletal structure. So even when the pose data is not missing, the measurements of the skeleton can differ between subsequent frames [OKO*12]. Figure 1 shows length differences between the left and right legs and arms during a leaning pose, when viewing the absolute joint positions on the right. Note the gap between the right hip joint and right upper leg, compared to the left leg. The Kinect SDK [Mic13] includes functionality which uses the relative positions of joints to calculate the orientations of the joints. The left character shows the effect of the length differences when applying the data as joint orientations to a skeleton with fixed length. We observed the feet are on the ground in the original data on the right, but the left character has one foot in the ground and the other above it. In order for the target pose of the virtual character to have both the visual signature of the input data, as well as predefined skeletal measurements, an inverse kinematics (IK) approach can be used that combines tracking of a few essential end–effector positions with tracking of the joint orientations representing the pose.

Kinect data can be used on a virtual biped with a similar skeletal structure, but not an identical morphology, as that information is not provided by the sensor. We observed these inaccuracies inherent to the Kinect data, and saw that controlling a physics simulated character using this data resulted in balance issues during idle or in–place motions. So at least for the first application described above, a balance strategy is needed in order for the virtual character to remain balanced

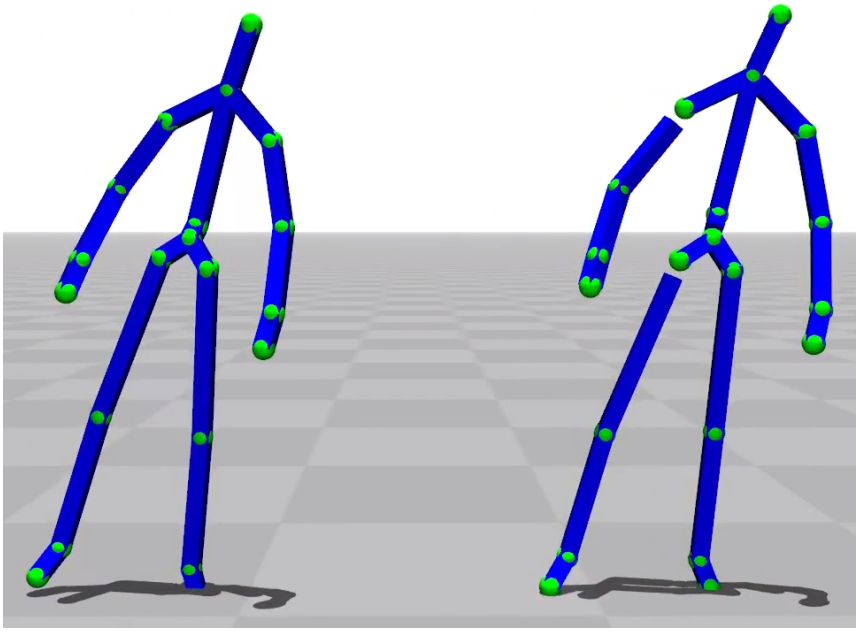


Figure 1: Right: spheres representing the joint position data returned by a Microsoft Kinect Sensor, with fixed size boxes representing the bones connecting the joints. Left: the same pose expressed in joint orientations, with the distance between joints determined by the same fixed size skeleton.

while performing a motion. The center of mass (COM) of the virtual character is entirely dependent on its morphology, therefore the balance control needs to correct the target pose of the character to both be faithful to the input data and to ensure the COM is positioned above the feet for increased balance.

These issues and solutions provide the main structure of our approach, which centers on a damped least squares (DLS) IK implementation to solve the issues of the Kinect motion data. It smoothly tracks five joint positions using a manually tuned damping factor, and projects all joint orientations in the input data onto the null–space of the Jacobian matrix used in the IK method. The latter allows for the pose to be tracked without interfering with the end–effector tracking. The resulting kinematic target pose is tracked by the physics character using PD–controllers, except for the ankle joints. The orientation of those ankle joints is continuously adjusted by a simple balance strategy, ensuring the COM of the character is positioned above the feet. We will show that this approach is suitable for control of the physics character during a range of arbitrary in–place motions, limited only by the inability of the balance strategy to keep the character upright during some less stable movements.

Thesis outline In the next section (2), related work is discussed, focusing on on–line control of a physics simulated character. Section 3 contains a schematic overview of our method, followed by a detailed motivation and explanation of the method. In section 4 the experimental setup, experiments and results are described and elaborated. The last section (5) is a discussion, analyzing the experimental results and suggesting future work that could complement our method.

2 Related Work

Our work builds on the rich field of physics based character animation. Specifically, we build on work in the field of on–line control of a physics simulated biped. Recently, the availability of consumer grade motion capture solutions like the Microsoft Kinect sensor makes on–line control of a virtual character more accessible. Applying these new motion capture systems to a physics simulated character could allow for a user to perform novel on–line interactions with a virtual environment. This potential has resulted in various promising works, that focus specifically on the problems associated with that setup.

2.1 Physics Based Character Animation

Physics based animation is a wide field that encompasses all situations where a physics simulation is used to calculate the movement of an object or substance. Various types of physics based animation have seen wide adoption in virtual simulations, like cloth or fluid simulations. On the other hand, character animation using a physics simulation has mainly been limited to rag–doll physics [GP12], and interactive animation of a physics based character is still an ongoing problem.

The field of physics based character animation can be roughly subdivided into solutions that focus on locomotion and solutions that focus on in–place or general purpose motion. Physics based locomotion requires specific motion controllers to (re)create a locomotion cycle, as well as to stay balanced during locomotion. An influential example of such a locomotion control strategy is the *SIMBICON* framework, proposed by Yin, Loken and van de Panne [YLvdP07]. Their framework uses flexible controllers that can be tuned in real–time using a few parameters or can be informed by motion capture data. Like the work by Yin et al., the physics based locomotion controller of Lee, Kim and Lee [LKL10] make a distinction between the stance hip and the swing hip in order to balance the character during the locomotion cycle. The stance hip is used to control the posture of the character, while the swing hip is used to control the foot placement during locomotion.

In–place motions cannot benefit from controlling foot placement for balance, but are generally easier to balance than locomotion. Solutions that focus on an in–place motion controller generally focus on a wide variety of in–place motions. Zordan and Hodgins [ZH02], for instance, combine trajectory tracking on motion capture data with a general purpose balance controller that controls the COM of the character. With their approach, a physics based character is able to perform a variety of motions, like boxing, fencing and dancing.

2.2 On–line Character Control

Certain applications require the physics based character to be controlled in real–time, for instance in a video game. Although on–line control of a physics based character does not share the same goals as the animation of a physics–based character, it does share many of the same challenges. In both cases, the physics based character should perform a visually feasible motion while staying balanced.

On–line control of a virtual character has the additional challenge of not being able to optimize the motion controllers to a specific motion sequence. The motion and balance controllers have to be able to handle unpredictable changing conditions and adapt to the input provided by the user in real–time. It is possible to divide this field into the solutions that provide on–line control through symbolic input data, and the solutions that use literal motion input for on–line control. An example of the former is the work of Coros, Beaudoin and van de Panne [CBvdP10], that propose a control system for locomotion of a physics character, that allows for users to control a wide variety

of high–level parameters, like direction, speed of movement and even specific characteristics of the gait.

Using literal motion data, captured in real–time, to control a physics based character should be the most intuitive control mechanism, because the virtual character simply mimics the motions of the user. With the arrival of motion capture sensors that are affordable to the general public, like the Microsoft Kinect, the interest in using motion capture data to control a virtual character in real–time have increased, as is evidenced by works released since 2010 [LZ11] [SH12].

Apart from being intuitive, the motion capture data is also guaranteed to be physically plausible, because it is performed by the user [GP12]. The challenges when using motion capture data arise from the fact that there are always some differences between the captured motion and the simulated motion. Differences in skeletal dimensions and simplified representations of real–world physical properties in the physics simulations cause issues when using motion capture data to control a physics character [GP12]. Data processing and balance controllers have to be used in order to overcome these issues.

2.3 Motion Data Processing

A popular data processing method to deal with the issues stated above, is to use a database of pre–recorded motion clips to increase the quality and consistency of live motion capture data [IWZL09] [LZ11] [SH12]. One approach to improving the data quality is to replace missing joint data in live Kinect motion data with a matched pre–recorded motion, by Shum and Ho [SH12]. Alternatively, Ishigaki, White, Zordan and Liu [IWZL09] embed context–sensitive pre–recorded motions inside an annotated environment, which are then combined with live motion capture to allow for environment–specific motions to be performed live.

Another possible difference between live motion capture data and the virtual character performing the motion, is the difference in skeletal structure or proportions. This skeletal configuration can even change between frames of motion data, as is the case with Kinect captured data [OKO*12]. This problem of retargeting the live motion data is resolved in different ways. Shum and Ho [SH12] solve this using a posture solver, which defines positional constraints of joints by taking the raw joint positions of a Kinect captured pose and tracking them using external forces applied to the joints of the dynamic character. Joint orientations are tracked similarly, by applying internal joint torques. Liu and Zordan [LZ11] propose a similar system of balancing forces and torques on individual body parts to position and pose a dynamic character based on Kinect data.

Other retargeting methods rely on inverse kinematics (IK) to calculate a pose for the skeleton so a set of end–effector points on the skeleton reach a particular target position, or minimize the distance to that target if it is not reachable. Choi and Ko [CK99] propose a method for on–line motion retargeting that uses IK. Their method uses damped least squares IK in order to retarget a prerecorded motion of a biped to a character with different proportions. A few crucial points on the body, like hands and feet, are tracked as end–effectors by the IK approach. Using the redundant degrees of freedom (DOFs) of the character, the original pose of the recorded motion is matched by the new character.

Our work builds on the work by Choi and Ko [CK99], by adapting it to the issues specific to Kinect generated motion data. Because of the fact that Kinect motion data does not enforce the proportions of the character, this retargeting method can be used to create a stable kinematic target pose for a fixed skeleton.

2.4 Motion Control

In order for a physics simulated character to perform a target motion, it has to use forces and/or torques to dynamically track a kinematic target motion. There are various techniques for calculating and applying these forces and torques to the dynamic character, that can be categorized as either physically plausible or not physically plausible. The latter introduces artificial forces into the simulation in order to control the character, similar to a puppet master controlling a puppet. The former approach limits itself to using internally consistent forces and torques, analogous to movement due to muscle activation.

Both Liu and Zordan [LZ11] and Shum and Ho [SH12] perform dynamic live tracking by applying forces to body parts to position them absolutely, using linear tracking controllers. The pose of the character is tracked by using controllers to calculate the torque required for each joint to reach a certain orientation. While using torques applied to the joints represents movement in a physically plausible way, applying external forces to the body parts to position a character within the world frame is not physically plausible.

When using the restriction that the dynamic character can only move by using physically plausible internal joint torques, the common method for calculating these torques is to use PD–controllers [ZH02] [LKL10] [GPvdS12] [GP12]. PD–controllers use tuned gain parameters to calculate a torque proportional to the difference between the current and target angle and angular velocity, for each controllable DOF of the character. The gain parameters require manual or automated tuning, as they depend heavily on the exact skeletal configuration and proportions of the character, as well as environmental effects and the desired responsiveness of the dynamic character to the kinematic target motion.

Additionally, the PD gains of a DOF depend on the body parts affected by a rotation of this DOF, as moving a larger mass requires a larger torque. Zordan and Hodgins [ZH02] remove this dependency, by dynamically tuning the gain parameters of each DOF by a value proportional to the moment of inertia of the body parts affected by that DOF. This allows for less manual PD gain tweaking, without losing the characteristics of good PD gains: responsive dynamic tracking, while the motion remains stable and smooth.

Further automation of the PD gain tuning is only possible with pre–recorded motions that can be fed into a learning algorithm to find appropriate gain parameters for that particular motion. Geijtenbeek, Pronost and van der Stappen [GPvdS12] use Covariance Matrix Adaptation for off–line optimization of the PD gains and balance control parameters. This allows for automated tuning of the system to a particular motion or set of motions, including the effects of the physics simulation. However, these motions have to be pre–recorded to be tuned off–line, making this approach not viable for an on–line application.

2.5 Balance Control

A motion tracking method that relies solely on internal joint torques, in order to provide physically plausible dynamic animation, requires a balance strategy in order to ensure that the simulated character stays upright during the motion. Various balance strategies have been suggested, but most have in common that the strategy aims to control the position of the center of mass (COM) of the character [ZH02] [LKL10] [GPvdS12].

A simple way to position the COM of the character is to control the orientation of some lower body joints, so the position of the COM relative to the ground plane can be controlled. This is exactly the approach of Zordan and Hodgins [ZH02], who compute an angular offset for the hip and ankle joints in order to reduce the COM position error, when projected on the ground plane.

The second approach of Zordan and Hodgins [ZH02] shares aspects with the approach of Geijtenbeek et al. [GPvdS12], namely the use of virtual actuation. Virtual actuation uses internal joint torques on a chain of joints to achieve a 'virtual force' on the root of the skeleton. This approach has parallels to inverse kinematics, as it aims to position an endpoint of a chain of joints, by finding the configuration of the joints that achieves this. In virtual actuation however, the target is to simulate a force on the skeleton root through the joints in the legs and the interaction of the character with the ground plane. Both Zordan and Hodgins and Geijtenbeek et al. use virtual forces to position the COM above the center of support of the character. Geijtenbeek et al. expand this approach by also using virtual torques to control the global orientation of the character.

A balance strategy based on virtual actuation is more complicated than an approach based on orienting hip or ankle joints in the kinematic target pose, as the latter does not add additional torques to the dynamic tracking. In our work, we use a simplified balance strategy inspired by Zordan and Hodgins [ZH02] that orients only the ankle joints in order to control the COM position and velocity of the character.

3 Method

Our system aims to allow a user to intuitively control a virtual character with almost complete freedom of movement, by having the character mimic the movements of the user. In–place movements of a person are captured using consumer level motion capture hardware, specifically the Microsoft Kinect. The virtual character remains 'grounded' within its virtual environment during any movement the user can perform, by applying a physics simulation and moving the character using internally consistent torques applied to its joints.

In order for this real–time system to perform these tasks, some underlying systems have to be in place. Some mapping has to be created from the input motion capture data to the virtual character. Then, the target motion has to be tracked in real–time by the physics simulated biped, using appropriate forces. All the while, the virtual character has to remain balanced and posed appropriately within the simple virtual environment . The overview that follows describes the difficulties that we encountered with these subsystems, and how those informed the method as a whole.

3.1 Overview

The solutions presented in this method are derived from the following challenges with the hypothesis: issues with Kinect data, requiring physically plausible tracking and the balance issues associated with that.

Kinect We have observed that the Kinect data can be temporally unstable, because the previous pose data does not seem to be considered when calculating the current pose of the user. This means that a joint can move unfeasibly between two frames of pose data. Related to this, is the fact that the Kinect does not enforce a fixed length of bones in its detected skeletal structure, and joints connected to the same body part can move closer or further away relative to each other. Because constantly changing morphology are not realistic and cannot be handled reliably by a physics engine, the Kinect data needs to be projected onto a fixed skeletal structure.

These issues with the Kinect data are resolved by processing the kinematic target, before it is used as a tracking target by the simulated character. Inverse kinematics (IK) is a collection of techniques for calculating the configuration of a multi–joint system in order for a point, or

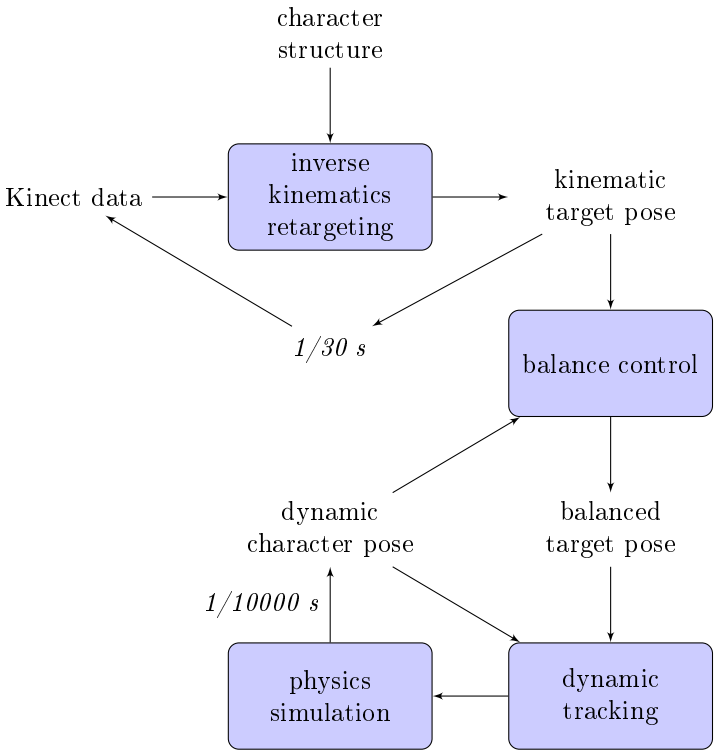


Figure 2: Data processing flow

end–effector, in that system to reach a desired location. We use IK to track several end–effectors, essential to describing and grounding the motion. This approach aims to circumvent the differences in skeletal proportions, both between frames of Kinect data, and the Kinect skeleton and physics character. The temporal instabilities in Kinect data are addressed by damping the convergence of the IK solution. This requires manual tweaking of a damping constant, but aims to smooth out the instabilities. In order to ensure the kinematic target pose is visually similar to the pose recognized by the Kinect, joint degrees of freedom (DOFs) not used to position the end–effectors, are used to match this pose. However, these redundant DOFs are limited, which means the Kinect pose cannot be perfectly matched in the kinematic target pose.

Physically plausible tracking Another challenge of our goal is that only physically plausible torques on joints are allowed. This allows a free and convincing interaction of the simulated character with the virtual environment. However, this also restricts the possible control methods of the virtual character. For instance, the global position and orientation of the character can never be directly controlled and there is no guarantee that it stays upright when performing a motion. Allowing the global position and orientation of the character to differ from the data captured by the Kinect sensor still supports various valuable applications, as described previously. The balance issues that we observed even when the character performs simple in–place motions, do however limit potential applications and intuitive control. If simple motions cannot be performed without the simulated character falling over, it limits the control a user experiences when using this method.

In order to generate physically plausible joint torques, the established approach of using PD–controllers is used. PD–controllers require a gain parameter tuned to a specific physics body to generate torques that reliably drive the physics simulated character towards a target pose. This target pose is the frequently updated kinematic target pose. An idle standing motion can be tracked by these controllers, but this tracking is not guaranteed to be balanced. In order to address the balance issues, we propose a simple balance strategy. We observe that the orientation of the feet in the Kinect data is more unstable than the rest of the body, which means we choose to ignore

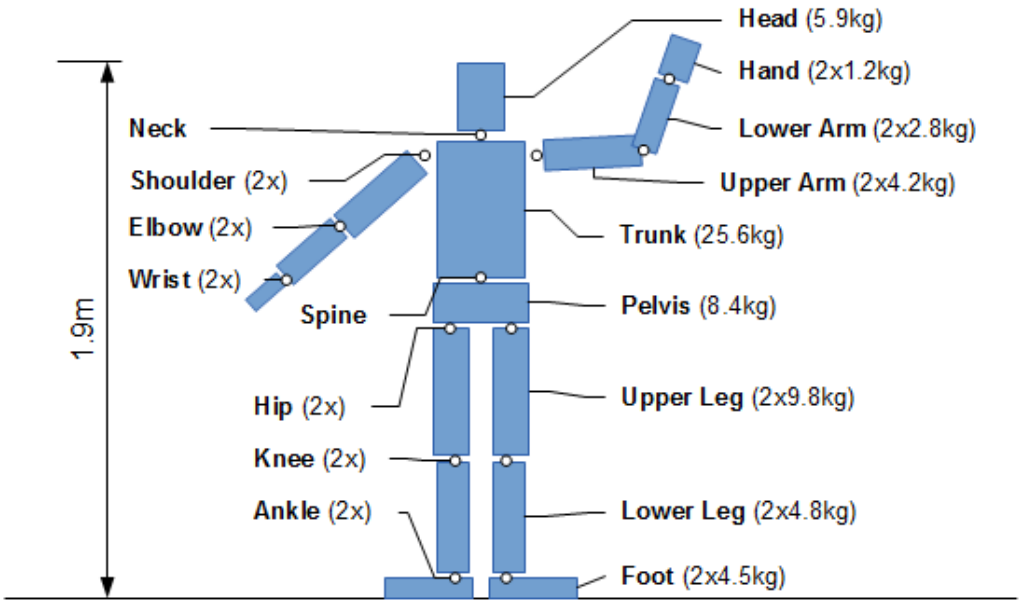


Figure 3: The structure of the physics character. Joint names and counts are shown on the left. Body part names and weights are shown on the right.

that data and instead use the ankle orientations to balance the character. This does result in poses that rely on a specific ankle orientation, like leaning, to potentially lose their visual signature. The ankles are rotated in order to control the position of the center of mass (COM), to ensure it stays above the centroid of the surface of the character contacting the floor plane. We define this position indicator of the ground contact of the character as the center of support (COS).

3.2 Physics Character

The Kinect sensor detects the absolute positions of a total of 20 'joints', which are implicitly connected by bones if they are adjacent in the skeletal structure the Kinect uses. The Kinect detects a skeletal structure, but does not provide information about the volume of body parts of the user. The physics character on the other hand, requires reasonable collision volumes in order to have plausible physical interactions with the virtual environment. Our simple collision volumes occasionally encompass several of the bones in the Kinect skeletal structure. For example, we combine the spine and the bones connecting the neck joint with the shoulder joints (see Figure 1) into a single torso body part for the physics character.

Skeleton A character with fewer joints results in a smaller computational cost for the IK implementation and the physics simulation. Fewer body parts also reduces the complexity of the physics simulation of the character, which means the simulation is likely to be more stable. We therefore define a skeleton that has the fewest body parts (and joints) while still being able to visually express whatever motion the user performs. Additionally, a similar structure can be found in related work, like Geijtenbeek et al. [GPvdS12] and Lee, Kim and Lee [LKL10]. Figure 3 shows the skeletal structure of the physics character. Each adjacent body part is connected by a joint with 3 rotational DOFs, because the Kinect SDK does not differentiate between different joints and assigns 3 rotational DOFs to all [Mic13].

The proportions of the character are not fixed, and our system can handle arbitrary changes to the morphology of the virtual biped. However, the absolute positions of end–effectors like ankles

and wrists are tracked by the IK solution, regardless of the proportions of the simulated character. This means that if the proportions of the virtual character are significantly different from the person being tracked, their pose might be distorted. This distortion occurs when the end–effectors are tracked to positions that are either unreachable or where the position relative to the body is significantly different between the two skeletal configurations. This means that this method is more appropriate for controlling a virtual biped that has proportions closely resembling that of the person controlling the virtual character.

3.3 Kinematic target processing

Motion data generated by a Kinect sensor has some systematic issues, and on–line processing is required to get a target motion that is stable and compatible with the physics simulated character. Firstly, the Kinect sensor is not necessarily positioned parallel to the ground, causing the data captured by the sensor to be tilted. This is addressed by applying a sensor tilt correction.

As previously mentioned, the skeletal structure of the physics character is different from the skeletal data captured by the Kinect. This is not an issue when only end–effectors are tracked using IK, as long as there is an equivalent end–effector in both structures. However, when using redundant DOFs of the physics character to match the pose of the input data, a mapping is required that describes the orientation of each joint of the physics character in terms of orientation(s) of input joints.

In order to ensure the absolute positioning of essential end–effectors, like feet, we propose a damped IK solution. The IK solution ensures these end–effectors are positioned correctly even when the skeletal structure of the input data differs from the virtual character, as well as smoothing out instabilities inherent in the Kinect data.

3.3.1 Sensor Tilt Correction

If the Kinect sensor is not positioned in such a way that its line of sight is parallel to the floor plane, the absolute positions of joints captured by the sensor are tilted in the virtual environment. When some of these joint positions are then used as end–effector targets, the kinematic target pose will also be tilted, resulting in a systematic error and balance issues.

In order to resolve this issue, we use information about the orientation of the sensor relative to the floor plane. Fortunately, the Microsoft Kinect SDK bundled with the sensor uses the depth image the Kinect captures to detect the floor plane if it is visible [Mic13]. This is required for the image processing to separate the person from the environment in the depth image, in order to accurately capture a motion. Both the orientation of the floor plane within the reference frame of the sensor, as well as the height of the sensor from this floor plane are returned by the SDK.

Using this information, we calculate the rotation required to match the virtual floor plane with the view of the floor from the point of view (POV) of the sensor. Then we apply that rotation to all input joint positions around the origin, which coincides with the position of the sensor. The data is also translated so the floor plane detected in the input data overlaps the virtual ground plane. After these corrections the joints are correctly positioned relative to the ground plane in the virtual environment.

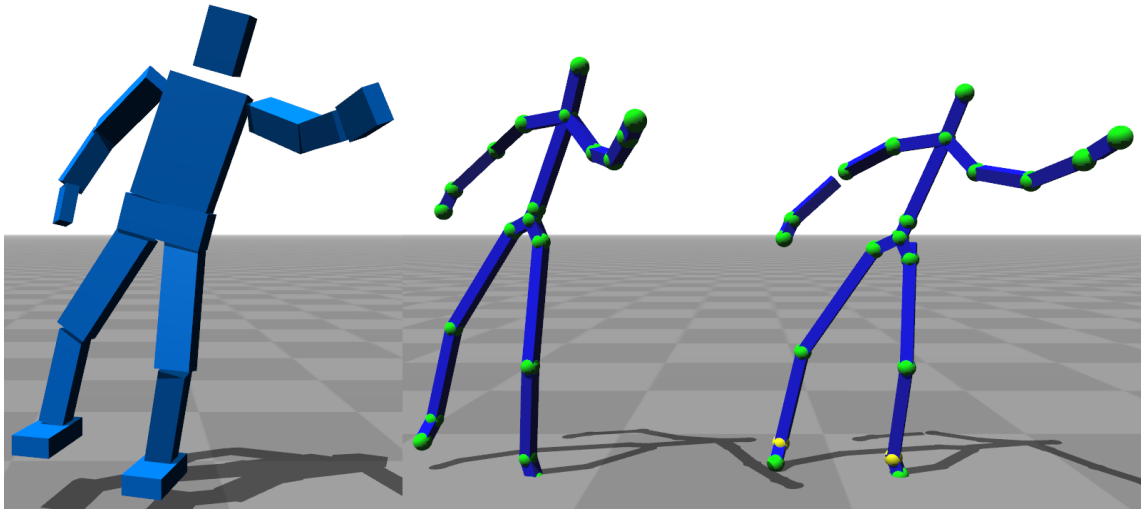


Figure 4: A side-by-side view of the same pose as expressed by the processed kinematic target (left) and the two representations of the Kinect input data: joint orientation (middle) and joint position (right) data.

3.3.2 Mapping

The physics character does not have the same skeletal structure as the input data returned by the sensor. Figure 4 shows this difference. Specifically notice how the input data has hip and shoulder 'bones', which are replaced by a single pelvis and trunk body part. In order to match the visual signature of the motion of the user, the input joint orientation data has to be mapped to the skeleton of the simulated character. In order to achieve this, we propose a simple mapping.

Each pair of parent–child joints of the physics character is mapped to an equivalent pair of joints in the input data. The parent in the parent–child pair is the joint closer to the root of the skeletal structure. The pair of joints in the input data do not have to be directly adjacent in the skeleton, which allows for mapping to a simpler skeletal structure, like our physics character. Additionally, a fixed angular offset can be defined for each mapping, which makes it possible to match the idle pose between the two skeletons.

This mapping is used when describing the target orientation of each joint in the virtual character based on the motion data, by returning the orientation of each joint relative to the parent joint in the coordinate space of that parent joint. This simple mapping allows for the input data to be mapped to a variety of possible biped skeletons of the simulated character, while preserving the relative orientations of body parts.

3.3.3 Inverse Kinematics

We propose IK–based processing of the kinematic data, because of the fact that Kinect captured motion data does not enforce fixed skeleton dimensions [OKO*12]. This means that when the joint orientation data is mapped to a virtual character with fixed length body parts, the pose can be distorted between frames of motion. These distortions can for instance cause a foot to intersect the ground plane or float above it, as in Figure 1. We hypothesize that as a result of the kinematic target motion not having stable foot positioning, that a physics simulated character tracking that target can experience balance issues. Additionally, these distortions mean that the kinematic target motion no longer reflects the motion of the user, as detected by the Kinect sensor.

There are various established IK techniques that can be subdivided into analytical and numerical methods [TGB00]. Analytical methods are characterized as complete, because they calculate the entire solution space for a particular IK problem. However, these methods cannot be generalized, as the analytical solution is derived for a specific joint structure. Additionally, if there is no exact solution then an analytical method returns an empty solution space. This occurs when an end–effector target position is unreachable, for instance.

Numerical or optimization based methods on the other hand use iterative optimization to approximate a single solution to an IK problem. While these iterative approaches are computationally more costly than an analytical method, they can still operate in real–time [Bus04]. Numerical methods can also approximate a solution, even when an analytical method returns an empty solution space. When an end–effector target is unreachable, for instance, an exact solution cannot be determined. However, numerical methods iteratively move the end–effector towards that target, approaching it as close as possible. Additionally, numerical methods do not require a specific solution for a specific joint structure. Instead, an arbitrary structure of joints, end–effectors and targets is described and optimized as a whole.

Because of this general applicability and robustness, we chose to use a numerical method for our IK–based kinematic data processing. Specifically, we consider a selection of popular methods that revolve around calculating a pseudoinverse of the Jacobian matrix of the IK system [Bus04]. Besides being commonly used, these methods allow for a 'secondary objective' to be tracked, by projecting the targets of this secondary objective onto the null–space of the Jacobian matrix [CK99]. We will use this secondary objective to track the pose of the kinematic target motion.

End-effector configuration Our IK solution aims to ensure that a few essential body parts are positioned absolutely, in order to create a stable kinematic target for the simulated character to track. These five end–effectors are the ankle and wrist joints, as well as the neck joint. Tracking these end–effectors anchors the positions of the limbs within the world, which is not possible with pose tracking alone.

To ensure the positioning of the feet is stable and consistent with the user in front of the sensor, we track the position of the ankle joints. Ankle joint positions are used instead of the position of the feet, because we observed the ankle joint positions to be more stable than the feet in Kinect data.

Wrist joints are tracked to allow for more reliable interaction with the virtual environment, by positioning the virtual hands in the same absolute position in the virtual environment as the hands of the user in the real world.

We observed that when the wrist target positions are low enough that they can not be reached by lowering the arms alone, the entire upper body turns so those targets can be reached. This negatively affects the resulting pose of the kinematic target. To resolve this, the position of the neck is also tracked, resulting in more stable and accurate upper body tracking.

Each end–effector has to be connected to a base node within the skeleton, by a chain of joints that are rotated to position that end–effector. This base node is supposed to be a fixed point in space, and should not be affected by the joints in the chain rotating. This is achieved by fixing the position of the root of the kinematic target skeleton and having it act as the base node for all joint chains. The root of the skeletal structure is the body part not connected to a parent body part through a joint. In our structure, this body part is the pelvis. Both the position and orientation of this base node are taken directly from the input data, and applied to the root of the kinematic target.

Using a single base node for the IK solution and giving each limb a single end–effector has the benefit of using most joints only for positioning a single end–effector. This avoids conflicting

joint rotations that have to be balanced. The exception is the joint between the torso and the pelvis body parts, which has competing rotations applied to it to track all three upper body end–effectors. Having a simple joint chain configuration like this also means less joint DOFs are required for end–effector positioning, and can therefore be used to track the pose of the input data.

Jacobian matrix The inverse kinematics (IK) methods that are being considered are: Pseudoinverse IK, Damped Least Squares (DLS) IK and Selectively Damped Least Squares (SDLS) IK. These methods are related in the fact that they all calculate an approximate (least squares) inverse of a Jacobian matrix, to find the rotation required of each joint DOF to move an end–effector to its target position. A Jacobian matrix contains partial derivatives, describing the translation of an end–effector in each axis as a result of a rotation of a joint around each DOF.

These partial derivatives can be calculated if the pose of the character is known. For the Jacobian calculation we use the previous kinematic target pose as the current pose of the character. The elements of the Jacobian matrix are easily calculated using this equation:

$$\frac{\partial s_i}{\partial \theta_j} = v_j \times (s_i - p_j) \tag{3.1}$$

With end–effector i and joint DOF j, where s_i is the position of i, θ_j is the angle of j, v_j is a unit vector along the axis of rotation of j and p_j is the position of j [Bus04].

If an inverse of this Jacobian matrix could be found, then multiplying it with a vector describing the translation required for each end–effector to reach its goal position would return a rotation for each DOF of the character, resulting in each end–effector reaching its target. However, the Jacobian matrix size is 15×45 (five 3D end–effectors, 15 3DOF joints) and not invertible. Additionally, not all end–effector target positions are necessarily reachable. So the IK methods discussed below all find a least squares approximation of this Jacobian inverse matrix, to find the best possible pose that minimizes the difference between the end–effectors and their target position.

Pseudoinverse IK The pseudoinverse IK method is a simple and established approximation based solution. It calculates the required DOF rotation angles $\Delta\vec{\theta}$ using the Moore–Penrose inverse ($J^\dagger$) of the Jacobian matrix:

$$\Delta\vec{\theta} = J^\dagger \vec{e} \tag{3.2}$$

Where $\vec{e}$ is the vector containing vectors from current end–effector positions to their target positions. $\Delta\vec{\theta}$ is the vector with the smallest magnitude that gives the optimal least squares solution to the equation $J\Delta\vec{\theta} = \vec{e}$ [Bus04].

This means that in a single iteration of this IK method, each end–effector should be as close to its target position as it can be. However, when the configuration is near singularity, this method is unstable, causing very large joint rotations even when the end–effector is very close to its target. A near–singular configuration is one where a rotation axis of a joint aligns with a rotation axis of other joints in the chain, for instance when an arm is (almost) stretched. Figure 5 shows this effect. The outline shows that a small clock–wise rotation of the joint (white dot) reduces the distance of the end–effector (black dot) to its target (red dot) by half. This is the value stored in the Jacobian matrix. Rotating twice as far should then reduce the distance to the target to 0, which is the rotation returned by performing the pseudoinverse IK method. The transparent body part shows the actual result of rotating twice as far, where the end–effector is as far away from its target as it was before rotating. This procedure repeats resulting in an oscillating joint, causing this IK method to become unstable under these circumstances.

Damped Least Squares IK DLS IK introduces a damping constant to the Jacobian inverse approximation, which dampens the rotation of joints, resulting in a convergence rate that is more

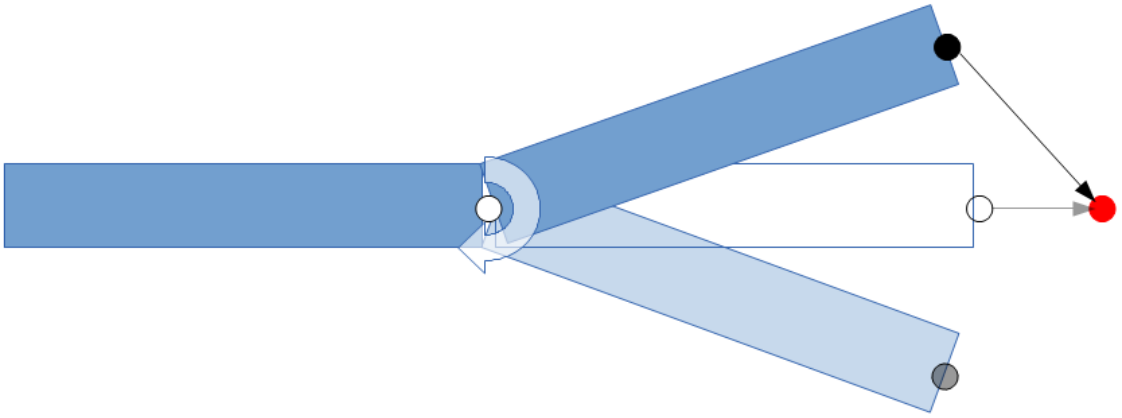


Figure 5: Schematic illustrating the instability caused by end–effector tracking with a near–singular limb and an out of reach end–effector target position.

than a single iteration. We manually tune the damping constant to ensure a stable transition between subsequent poses and to eliminate oscillating motions with unreachable end–effector targets and near–singular configurations. DLS IK calculates the approximate Jacobian inverse as follows:

$$J^{\dagger} = J^{T}(JJ^{T} + \lambda^{2}I)^{-1} \tag{3.3}$$

Where λ is the damping constant. When $\lambda = 0$, DLS IK is the same as pseudoinverse IK. High λ values on the other hand, reduce the responsiveness of the solution severely. In that case, it takes several iterations for the solution to converge and for the end–effector to reach its target. This does result in a smooth motion of the kinematic target.

We find the value of λ by generating kinematic targets for several motions and tuning the value to ensure stable motion with near–singular limbs and smooth transitions between poses even when the Kinect data jumps. Those goals have to be balanced against the goal of remaining responsive enough to keep the visual signature of the motions and not smoothing away important characteristics. Additionally, a high damping constant also introduces a delay to the tracking, because the goal is not reached in a single iteration. This can be detrimental to the real–time user experience, so low damping that still resolves the above issues is preferred over higher damping.

When we compare the pseudoinverse IK method with DLS IK with a sufficiently large damping constant, the DLS IK method completely eliminates the instabilities present with the pseudoinverse method. Additionally, using DLS IK results in a smoothed kinematic target motion during unstable Kinect data. This improvement does come with a noticeably reduced responsiveness of the kinematic target to the motion of the user. Despite reduced responsiveness, DLS IK showed sufficient improvements to the stability of the kinematic target motion, and as such we conclude that DLS IK is more appropriate than pseudoinverse IK for our purposes.

Selectively Damped Least Squares IK SDLS IK is a relatively recent extension of DLS IK, aiming to remove the requirement of manual tuning of the damping constant by automatically determining this value for each vector in the singular value decomposition of the Jacobian matrix [BK05]. SDLS IK increases damping if the rotation calculated for a DOF using the pseudoinverse method moves the end–effector much more than the distance between the end–effector and its target.

During comparison of the SDLS method with DLS IK and pseudoinverse IK, we observed that it indeed succeeded in eliminating the instabilities that the pseudoinverse method suffers from, without requiring any manual tuning of a damping constant. However, it also revealed that the SDLS method converged very slowly towards the end–effector target positions in many situations.

This caused a slow responsiveness of the simulated character to live input data, similar to a very high λ value in the DLS method.

For a real–time system, the reduced responsiveness that SDLS IK introduces is detrimental to the user experience. Therefore, we conclude that DLS IK is a more appropriate kinematic data processing method for our purposes, compared with SDLS IK.

Limiting end–effector distance to target Besides the SDLS IK method, Buss and Kim [BK05] also propose clamping the distance of the end–effector to its target each iteration. Limiting the distance between the current end–effector position and the target position, by clamping it to a manually tuned value, should reduce jitter of limbs by limiting the maximum distance an end–effector has to move each iteration. This clamping is achieved by moving the target position of an end–effector closer to the current position, until the euclidean distance between those two positions is less than or equal to the clamping parameter.

Limiting the target distance of each end–effector should have several benefits. Firstly, it largely eliminates the problem of erroneous joint position data returned by the Kinect, as a single frame with an incorrect end–effector target no longer results in a large rotation of body parts to reach that target. Instead, the end–effector moves in the direction of the incorrect target, but only as far as the clamping parameter is set, which is a less extreme reaction that can be more easily recovered from by tracking subsequent correct data.

Buss and Kim [BK05] also observe that this approach allows for the damping constant in the DLS IK method to be set to a lower value. Because large jumps in end–effector movement no longer have to be smoothed out by the damping component of DLS, but instead are clamped. This allows motion by the kinematic target that is more responsive to the (live) input data, although it does smooth out large movements. Very strong motions, like fast kicks or punches, will be reduced to a maximum velocity depending on the clamping parameter.

Pose Tracking If only the end–effector positions are tracked for the kinematic target, then that target can still take a variety of often unrealistic poses. In order to ensure the pose of the kinematic target is both feasible and resembles the pose of the person in front of the Kinect sensor, we have to track another target. The pose of a character can be expressed by the orientation of each joint, when the skeletal structure of the source and target body resemble each other. The simulated character does not have the same skeletal structure as the Kinect data, so the joint orientations have to be mapped to the simulated character. This mapping has been described in a previous section.

In order to achieve this second target of matching the pose of the kinematic target to the input data, the redundant DOFs of the character can be rotated to match the input pose. In the context of IK, a redundant DOF is a DOF that is not required for positioning an end–effector. This means that a redundant DOF can take any orientation without affecting the end–effector tracking by the IK method.

Using redundant DOFs for pose tracking can be achieved by calculating a matrix that projects a vector onto the null space of the Jacobian matrix [CK99]. This null space describes the redundant DOFs, so when a vector ($\vec{\varphi}$) representing the DOF rotations required to match the input pose is projected onto it, the pose is as closely matched as possible without affecting end–effector tracking.

$$\Delta\vec{\theta^{\dagger}} = (I - J^{\dagger}J)\vec{\varphi} \tag{3.4}$$

Equation 3.4 shows the projection of $\vec{\varphi}$ onto the Jacobian null space, where $\Delta\vec{\theta^{\dagger}}$ are the rotations of each DOF required to move the current kinematic target to the new target pose. Adding $\Delta\vec{\theta^{\dagger}}$ to the result of the DLS IK method, gives the rotations required to move the current kinematic

target to the new kinematic target, both tracking end–effector positions and matching the input pose.

We observed during initial testing that the pose tracking could have a large effect on the kinematic target motion, especially when the Kinect returned unstable motion data. In order to allow for manual control over the effect of pose tracking on the kinematic target, we propose a weighting factor, or pose weight (w), with value between 0 and 1. A weight lower than 1 dampens the pose tracking, allowing for a manually tuned balance between end-effector and pose tracking. A value of 0 means pose tracking does not contribute at all to the kinematic target motion.

$$\Delta\vec{\theta} = J^{\dagger}\vec{e} + w(I - J^{\dagger}J)\vec{\varphi} \tag{3.5}$$

Equation 3.5 shows the final kinematic data processing calculation. It combines the end–effector tracking of Equations 3.2 and 3.3 with the pose tracking described in Equation 3.4, and introduces the pose weight (w) to allow for the two components to be manually balanced.

3.4 Dynamic Tracking

In order to move the physics simulated character to match the kinematic target motion, a dynamic tracking method has to be introduced. A requirement of our system is that it can only control the dynamic character using physically plausible joint torques, analogous to muscle activation in humans. A simple and common method for calculating the torques required to move a dynamic character to a specific target pose is to use PD–controllers [GP12].

Next we describe some specifics of the physics simulation that are relevant to the dynamic tracking problem. Then the specifics of how we use PD–controllers to control the physics character are described. Lastly, a simple balance strategy is proposed to resolve balance issues we observed during initial testing.

3.4.1 Physics simulation

Collision volumes The simulated character has collision volumes that correspond with box–shaped body parts, similar to the characters used by Lee, Kim and Lee [LKL10] or Geijtenbeek et al. [GPvdS12]. Besides the common use of box–shaped body parts, there is another benefit specifically related to the physics simulation. Having a flat surface representing the bottom of the feet results in a large surface for ground contact. This makes for a more stable physical interaction with the ground plane and makes the character easier to balance.

Self–collision For the sizes of the box–shaped body parts, we are visually motivated and set those to loosely resemble a person with average proportions. More importantly, no body parts collide with the rest of the body during an idle pose. We define the idle pose as having the feet next to each other, with parallel legs, and the arms hanging down parallel to the torso. Ensuring that the body is proportioned to avoid self–collision is especially relevant as we chose not to enable self–collision in the physical simulation. Not simulating self-collision means body parts can penetrate other body parts without colliding. While this is not physically realistic, it makes analyzing the results of the dynamic tracking easier. If the physics simulation were to resolve self–collisions within the character, the forces this generates could visibly affect the tracking, making it more difficult to judge the tracking performance.

Friction The physics simulation solution we use for our virtual environment, *Open Dynamics Engine*, approximates friction using the Coulomb friction model [Smi06].

$$F_R = \mu F_N \tag{3.6}$$

Equation 3.6 describes Coulomb friction [Pop10], where F_R is the force of friction as a result of friction coefficient μ and normal force F_N. The value for μ has to be manually defined, and determines the potential strength of the friction force. Different materials interacting exhibit specific friction forces, which can be expressed in a physics simulation by a specific μ value.

A common problem with physics simulations of complex rigid body systems, like our dynamic character, is foot sliding during motion tracking. This issue has previously been observed by Geijtenbeek et. al. [GPvdS12], and appears to occur when the internal torques used for controlling the character cause large and oscillating ground reaction forces. Ground reaction forces are the forces resulting from collision of the feet body parts with the ground plane.

These forces can result in a loss of friction of the feet, which in turn results in the body sliding on the ground plane. As the ground contact is the main, and frequently only interaction of the physics character with the environment, a stable contact makes balance easier and grounds the motion within the virtual environment. In order to combat this issue, we chose a relatively high friction coefficient value of $\mu = 1.0$. This value is roughly equivalent to the static friction between concrete and rubber surfaces [Ser13]. A sufficiently high value like that should ensure a stable contact between the character and the ground.

3.4.2 PD control

Proportional–Derivative (PD) controllers are a simple and established method for calculating joint torques required for tracking a target motion [GP12]. This feedback–based motion control technique is used to generate physically plausible joint torques. It does require manual tuning of the gains, in order to balance responsive and smooth tracking. Some information about the body and pose affected by a particular joint rotating, can be used to simplify this tuning.

Initial results of the dynamic tracking method using PD control, show us that even simple in–place motions cause the simulated character to lose balance. In order to resolve this issue, we propose a simple balance strategy that overwrites the orientation of the ankle joints in the kinematic target with an orientation derived from the center of mass (COM) position relative to the center of support (COS). The COS is defined as the centroid of the surfaces of the character in contact with the ground plane. During a balanced motion, these surfaces are the bottom of the feet colliding with the ground plane and so we approximate the COS by averaging the centers of the feet, if they are in contact with the ground.

$$\tau = k_p(\theta_t - \theta_c) + k_d(\dot{\theta}_t - \dot{\theta}_c) \tag{3.7}$$

PD equation Equation 3.7 shows the calculation of a DOF torque scalar τ, from the current angle θ_c and angular velocity $\dot{\theta}_c$ and the target angle θ_t and angular velocity $\dot{\theta}_t$. DLS IK calculates the difference in angle for each DOF from the current target pose to the next. We use this difference as the target angular velocity for PD control, while the new target pose offers the target angle for each DOF. k_p and k_d are the PD gains that scale the differences in angle and velocity to get an appropriate torque value to move the joints without causing instabilities. If dynamic state of the system is fixed, then the value for k_d can be derived from k_p using the following equation: $k_d = 2\sqrt{k_p}$. While the dynamic state of our system is not fixed, we use this relation to approximate the k_d value, because we found that this approximation does not seem to adversely

affect the performance of the dynamic tracking. Additionally, this means the value for k_d does not have to be manually tuned.

As all joints in the simulated character have three DOFs, the torque scalars calculated by the three PD–controllers are multiplied by their axes of rotation to get a single torque vector (in world coordinates). Those torques are then applied to the child body part and in opposite direction to the parent body part, to rotate the joint towards its target orientation using physically plausible torques.

Moment of inertia compensation In order to reduce the amount of tuning required for the PD gains, we employ a method proposed by Zordan and Hodgins [ZH02]. By scaling manually tuned PD gains by a value dependent on the pose and configuration of the part of the body affected by a particular joint, PD gains tuned individually for each joint DOF can be replaced by a single PD gain multiplier. This multiplier is still manually tuned to our specific character. We then multiply that single value with ratios derived from a heavily simplified approximation of the moment of inertia of the body parts affected by each DOF.

$$t_i = r_c \times m_c \tag{3.8}$$

Equation 3.8 describes the calculation of these ratios between DOFs, with t_i being the simplified moment of inertia approximation for DOF i. r_c is the radius of the COM c of the body parts affected by DOF i with respect to the axis of rotation of i. In order words, r_c is the length of the vector perpendicular to the axis of rotation of i to that COM. m_c is the total mass of the body parts affected by DOF i.

This equation describes all body parts affected by i as a point mass at the COM of those body parts. A physically correct moment of inertia calculation of a point mass would take the square of the radius r_c, but we found through initial testing that this resulted in ratio differences between the lower and upper body that were too large. Because the lower body joints affect the entire upper body when standing upright, the ratios calculated for the DOFs in the lower body with a squared r_c were significantly higher than other DOFs. By assuming a point mass for the calculation, we already heavily simplified the problem at the cost of accuracy. We therefore chose to further simplify the ratio calculation by removing the squared radius term, which during initial testing resulted in promising ratios between DOFs.

Because t_i is a very rough approximation, its value has no intuitive meaning within the dynamics of the system. In order to ensure the value of the PD gain multiplier, which has to be manually tuned, has an intuitive value that can be related to uses of PD–controllers in other work, we divide the t_i for each DOF by t_0. This is the t value for the first DOF in the spine joint, during an idle pose. This means that the value determined for the PD gain multiplier is the k_p value for the first DOF of the character, and the k_p value for any other DOF i is based on the ratio between t_0 and t_i.

3.4.3 Balance control

Early tests showed us that even simple in–place motions caused the character to lose balance, when using the method described above. Without balance control, there is no feedback of the current state of the dynamic character when applying joint torques. So when the character starts losing balance, there is no system in place to avoid this. A balance strategy is therefore necessary to allow for a reasonable array of different in–place motions to be properly performed by the dynamic character. The same tests demonstrated that the Kinect–generated position data of the feet was occasionally unstable. This is likely caused by the difficulty of detecting the feet against the floor plane in the depth image that the sensor captures. This means that the orientation of the ankle joints is frequently incorrect, which is a problem with a character that relies on stable contact with

a ground plane in order to stay balanced. Our initial attempt to resolve these instabilities by fixing the ankles to a specific orientation did not remove the balance issues.

Method These observations leads us to propose a simple balance strategy, where the orientation of the ankle joints is overwritten and continuously adjusted to ensure the simulated character remains balanced during in–place motions. This is an approach similar to that of Zordan and Hodgins [ZH02], who balance the position of the COM of the physics character by controlling the ankle and hip joints. This strategy assumes that the feet of the character are flat on the ground, and rotates the ankles to position the COM above the center of ground support (COS) of the character, under that assumption.

Center of support The COS describes the centroid of the surface area of the character in contact with the ground plane. So when the character has both feet on the ground, this centroid lies somewhere between the positions of the feet. For the centroid of the contact area of each foot, the COM of the foot body part is used. Although this is an approximation, it does not appear to result in adverse performance and the COM of the foot is information provided by the physics simulation, while the centroid of the contact area of a foot with a ground plane would have to be calculated. Since this balance strategy runs each iteration of the physics simulation, the computations should be kept to a minimum so as not to adversely affect the performance of the entire system. The COS shifts more towards one foot as the COM of the kinematic target shifts more above one of the feet. For instance, when the kinematic target shows a character leaning to the right, the COS is closer to the right foot. Because the right foot supports a larger weight in that scenario, the COS shifts to reflect that. Additionally, this ensures that if the user shifts his weight onto one foot, for instance before taking a step, that the balance strategy responds by shifting the average ground contact point onto that foot.

COM target If we assume the bottom of the feet of the dynamic character are flat on the ground, we can adjust the orientation of the ankle joints to position the COM of that character relative to the COS. There are two target positions for that COM that have their own advantages and disadvantages. Firstly, the COM can be positioned above the COS, which offers the maximum balance potential, but does affect certain poses. For example, if the user leans to one side, this leaning pose is characterized by a COM that is offset relative to the COS. If the COM is positioned exactly above the COS, this leaning pose is not performed by the dynamic character.

The second approach is to use the position of the COM relative to the COS in the kinematic target as the COM target for the balance strategy. This approach is similar to rotating the ankle joints so the feet body parts are parallel to the ground plane in the kinematic target pose. The difference with that simple approach is that the COM of the dynamic character is not necessarily in the same position as the COM of the kinematic target, because the pose of the dynamic character is rarely identical to the kinematic target. This second approach better respects motions like leaning, as it attempts to position the COM of the physics character with the same offset as the kinematic target. However, we hypothesize that this approach offers worse balancing performance, because it allows for the COM to be offset from the COS. When the COM is horizontally offset far enough from the COS, the character can easily topple over in that direction.

Because both COM targets have their own benefits and drawbacks, we propose a manually tunable parameter that allows for these two targets to be balanced. Because both of these targets are represented by a position in virtual space, an intuitive parameter could be a weighting factor that allows for linear interpolation between these two positions. We therefore propose a COM target weight parameter w_{com}:

$$\vec{p_{com}} = w_{com} \times \vec{p_k} + (1 - w_{com}) \times \vec{p_c} \tag{3.9}$$

where $\vec{p_k}$ is the position of the COM relative to the COS in the kinematic target and $\vec{p_c}$ is the target of the COM where it is positioned directly above the COS. The $\vec{p_{com}}$ vector then gives the COM target. Our balance strategy attempts to position the COM of the dynamic character on that target by rotating the ankle joints under the assumption that the feet are positioned flat on the ground plane.

COM velocity compensation Rotating the ankles to track a COM target position does not take into account the motion of the dynamic character. If the COM of the character is moving quickly, it can cause the character to lose balance in the future, even if the COM position is balanced in the current iteration of the physics simulation. In order to anticipate potential balance issues because of quick movements by the character, we propose adjusting the position of the COM target $\vec{p_{com}}$ by taking into account the velocity of the COM.

In order to take the COM velocity into account, we add to $\vec{p_{com}}$ a vector derived from the COM velocity vector ($\vec{v_{com}}$), which is expressed in meters per second. Our approach decomposes the COM velocity vector in a vector perpendicular to $\vec{p_{com}}$ ($\vec{v_p}$) and a vector parallel to $\vec{p_{com}}$, which is ignored.

$$\vec{v_p} = \vec{v_{com}} - (\vec{v_{com}} \cdot \frac{\vec{p_{com}}}{\|\vec{p_{com}}\|}) \times \frac{\vec{p_{com}}}{\|\vec{p_{com}}\|} \tag{3.10}$$

We do this in order to only compensate for the COM velocity when it offsets the COM horizontally, as a vertical velocity of the COM does not challenge the balance of the dynamic character. With a vertical COM velocity, the COM stays above the COS and thus the character is likely to stay balanced. This situation occurs, for instance, when the character crouches.

The length of $\vec{v_p}$ is then clamped, so it does not exceed the length of $\vec{p_{com}}$. This ensures the COM velocity correction does not overshadow the original COM target position. During a sudden movement the COM velocity vector could, for instance, have a length of 2 (meters per second). Without clamping that length, the velocity corrected COM target would be 2 meters offset from the original COM target. That is such a large offset, that it can actually cause balance issues, instead of resolving those.

Finally, we introduce a manually tuned parameter (w_{vel}) to further control the COM velocity compensation. This parameter scales $\vec{v_p}$ in order to provide control of the COM target and balance between the position target and velocity compensation.

$$\vec{p} = \vec{p_{com}} + w_{vel} \times \vec{v_p} \tag{3.11}$$

$\vec{p}$ is the final COM target position of our balance strategy. This vector expresses the COM target relative to the position of the COS.

Ankle orientation With the COM target vector $\vec{p}$ and the current COM position relative to the COS, the orientation of the ankle joints can be adjusted so the new COM position vector matches the target vector. However, we observe during initial testing that simply setting the new target orientation of the ankle joints to the orientation that the balance strategy calculates results in large rotations of the ankles between iterations of the physics simulations. These large changes in the target orientation of these joints results in the associated PD–controllers returning very high torques in order to track those targets. Subsequently, the physics simulated character becomes unstable due to the high torques applied to its ankle joints.

In order to resolve these instabilities, we propose a parameter that clamps the angular velocity of the ankle joint target motion. This clamping parameter ensures that the rotation required to move from the current orientation of the ankle joints to the target orientation does not exceed the angular velocity maximum that the parameter dictates. If that maximum is exceeded, then a new

target orientation is calculated for the ankle joint that results in a movement towards the target calculated by the balance strategy, but without exceeding the maximum angular velocity.

Using the ankle joints to maintain balance means the feet are not necessary oriented in a visually plausible manner. In order too maintain a visually convincing orientation of the feet, and because only two DOFs of an ankle joint are required to track the COM target, the third redundant DOF is set so each foot of the dynamic character point forwards, relative to the lower leg. Although this results in visually plausible orientations for the feet, the orientation of the feet as the Kinect sensor detects it is no longer reproduced by the dynamic character. So when the user solely moves its feet, it is not recognized and tracked by our system.

4 Experiments and Results

Various experiments were performed to demonstrate the capabilities of our solution, as well as to show different behavior of the system based on different parameter values and the limits of the system. We performed experiments to demonstrate the following effects:

- The effect of different configurations of kinematic target processing on the kinematic target pose.
- The effect of using different PD gains based on an estimate of the moment of inertia, versus dynamically calculating the moment of inertia.
- The effect of using balance control and IK–calculated target poses on balance.
- The effect of varying the morphology of the dynamic character.

4.1 Experimental Setup

Our method is tested within a home environment, as the Microsoft Kinect sensor is consumer hardware, and a professional motion recording environment is not necessarily available to users of the sensor. This means the sensor is placed within a representative living room scene (Figure 6), with furniture visible to the sensor, providing various levels of depth for the Kinect to capture. Within this scene, the body of the user is still always completely visible, roughly in the center of the FOV of the sensor. This should ensure the results of the experiments are representative of a consumer application of the method, without introducing additional risk of poor motion data quality.

Sensor positioning The Kinect sensor is positioned level, at a height of 90 cm above the ground, pointed at a user standing 2.5 m away from the sensor. This is roughly halfway between the minimum and maximum range of the depth sensor (between 1.2 m and 3.5 m [Mic13]), allowing for the largest freedom of movement while staying within the range of the sensor. As advised by the Kinect guide [Mic13], interference from infrared light sources other than the Kinect is minimized in the test setup. Specifically, sunshine is blocked in the test environment.

Software dependencies The Kinect motion data is retrieved at its maximum rate of 30 times a second, using the Kinect SDK version 1.6.0 [Mic13]. The Kinect SDK is responsible for retrieving the absolute positions of the 20 tracked joints from the sensor output. The SDK also generates the orientation information for the joints, from the position data. This data is smoothed by the

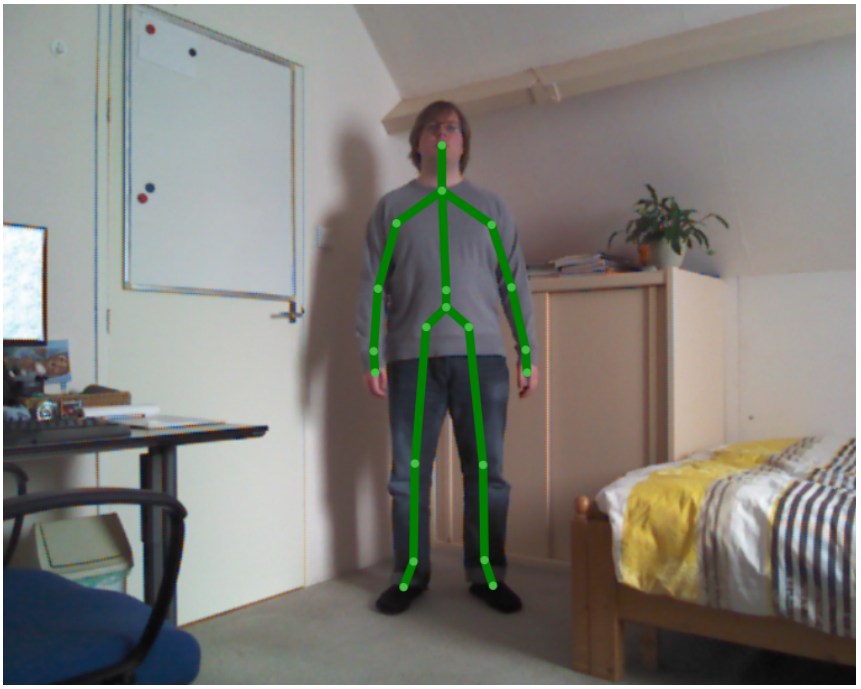


Figure 6: The environment used for recording motion for the experiments, as seen from the perspective of the Kinect.

Kinect SDK using its default smoothing algorithm and parameters [Mic13]. Additionally, the SDK uses the depth image to generate an equation that describes the floor plane, if the floor is visible by the sensor.

Pre–recorded motions Some experiments use a set of increasingly challenging pre–recorded motion clips. Although our method is built to be used in live applications, the use of pre–recorded motions allows for reproducible and comparable experimental results. In on–line applications, there is no guarantee of the quality of the motion data. In the pre–recorded motions, this random quality factor is eliminated, in order to separate the performance of our system from the performance of the motion capture system. To accomplish this, multiple recordings were made of the same motion, and the clip used in the experiments was the one with the least erroneous or missing motion data, after visual analysis.

Clip	Length (s)	Description	Challenge
waving	12.1	An energetic waving motion	Upper body motion
leaning	32.0	Leaning side–to–side	Slow COM shift
bowing	9.2	Bowing down	Sudden COM shift
reaching	8.6	Reaching down, bending knees	Full body controlled motion
boxing	15.6	In–place boxing moves	Energetic full body motion
stepping	10.7	In–place step	Controlled ground support change
balancing	28.3	Balancing on one leg	Limited ground support
jump	21.2	A small jump	Losing ground support

Table 1: A description of the motion clips used in the experiments, including the challenges they pose for the system.

Table 1 shows the set of pre–recorded motion clips used in some experiments below. They are designed to increase in balancing and tracking difficulty, as well as to provide representative motions of a large range of possible in–place motions. The range of motion is limited by the range

within which the Kinect sensor provides motion data of a reasonable quality.

Initialization period In order to ensure that the physics simulated character is stable when starting to perform the live or pre-recorded motions, a short (<1.5s) initialization period is enforced. In this period the physics character is suspended a short distance (<8cm) above the ground plane and its root rotation is controlled to stay upright, while tracking the kinematic target motion. After the initialization time is over, the character is released and experiences gravity and ground reaction forces and is plausibly simulated from then on. All experimental results ignore the initialization period and only start capturing results after that period is over.

Dynamic character Figure 3 shows the schematic of the dynamic character used for the experiments. For the density of the simulated character, an average density of the human body ($1000kg/m^3$) is chosen [DW74]. With the collision volumes described previously having this uniform density, the mass of the character is $94.5kg$. See Figure 3 for the mass of individual body parts. The use of visually motivated, simple collision shapes and a uniform density results in a weight distribution of the character that significantly differs from a realistic human weight distribution [CMY69], as seen in Table 2.

	head	trunk	upper arm	lower arm	hand	upper leg	lower leg	foot
sim. char.	6.24	36.00	4.44	2.96	1.27	10.37	5.08	4.76
human	7.3	50.7	2.6	1.6	0.7	10.3	4.3	1.5

Table 2: A comparison of the weight distribution (in percentage of total weight) of the simulated character and a real human [CMY69].

4.2 Kinematic Target Processing

One of the main contributions of our method is the use of damped least squares inverse kinematics to process the live Kinect motion data for improved stability. Because of the fact that Kinect pose data does not enforce the proportions of its bones, using unprocessed joint orientation data derived from that could result in an inaccurate or unstable kinematic target motion. This is however dependent on the type of motion performed, as some motions cause a larger difference in skeletal dimensions between frames of Kinect data. Using DLS IK aims to resolve this issue, as well as smoothing out temporal instabilities in motion data of poor quality.

Our method for generating the kinematic target motion exposes two main parameters that require manual tuning for best results and that have a significant effect on the target motion. The first is the damping constant introduced by using DLS IK. A low damping constant can cause unstable motion in extended limbs. A high damping constant reduces the responsiveness of the end–effector tracking, making the kinematic target motion lag behind the on–line motion performance, as well as smoothing out fast and small movements.

The second parameter of the Kinematic Processing method is the pose tracking weight. A low pose tracking weight results in unrealistic poses of the generated kinematic target, that no longer visually resemble the motion performance captured by the Kinect. A pose tracking weight that is too high starts affecting the end–effector tracking of the IK solution, as well as causing instabilities by over–correcting to match the target pose, similar to a low damping constant. A balanced configuration of these two parameters should minimize these issues and provide the best performance of the system.

Setup The aim is to demonstrate the effects of varying these parameters, and find a balanced configuration. A single pre–recorded motion clip containing a range of motions (see Figure 7), is either left unprocessed or is processed using various IK parameter configurations, and the quality of the resulting kinematic target motion is recorded. The quality of the kinematic target is expressed by two values: an average end–effector position error and an average joint orientation error.

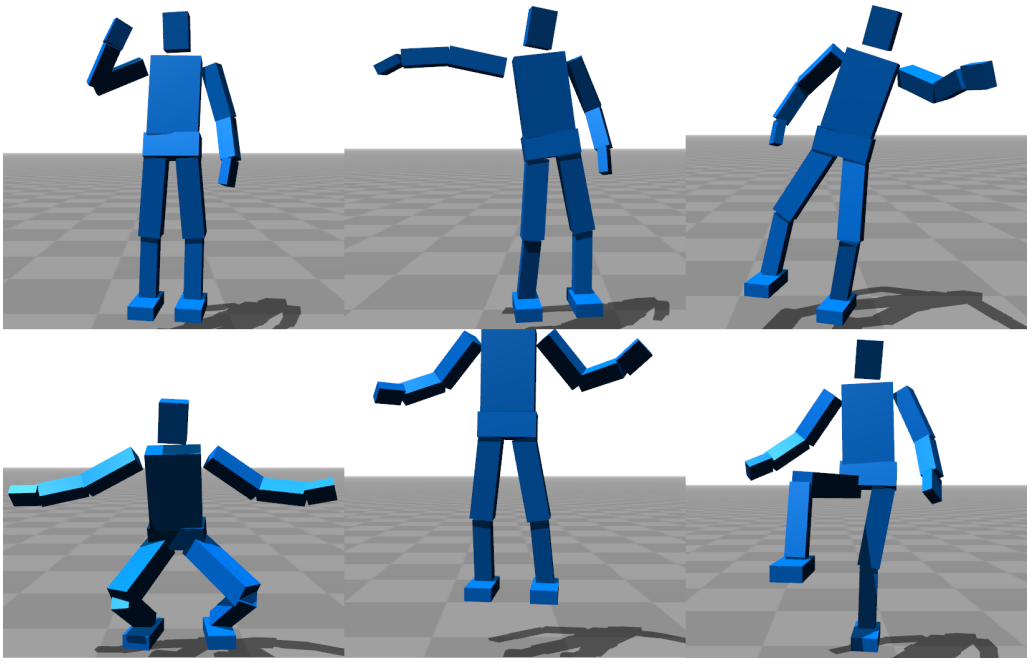


Figure 7: Representative poses of the range of motions in the pre-recorded motion clip used in the kinematic target processing experiment.

The position error is the Euclidean distance between the position of the five end–effectors, relative to the root of the character, in the kinematic target pose and the Kinect sensor data. These end–effectors are the wrist, ankle and neck joints. The position error is averaged over the entire motion clip and all end–effectors.

The orientation error is the difference in angle of each DOF of the eight unique joints, in the kinematic target pose and the Kinect sensor data. These eight unique joints are the ankle, knee, hip, spine, neck, shoulder, elbow and wrist joints. The orientation error is averaged over the entire motion clip and all DOFs.

These errors are first recorded when the motion clip is left unprocessed, which means the joint orientations as returned by the Kinect SDK, are applied directly to the kinematic target. Then all combinations of the two IK parameters within a range, are tested. The damping constant range is between 0 and 1.2 and the range of the pose weight is between 0 and 1.0. Both ranges are traversed with a step size of 0.1.

Results

Figures 8 and 9 show the two errors measured as a result of varying the two main parameters of the DLS IK kinematic target processing technique. This can be compared to the errors when no kinematic processing is applied. Unprocessed, the pose error is 0 as expected, as the joint orientations are left unchanged from the input data. The end–effector error with unprocessed data is 0.118, which is significantly higher than the processed target motion (with most parameter configuations). This error is introduced by the dimension differences of the dynamic character and

pose weight \ damping constant	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2
0	1.813	0.643	0.465	0.467	0.516	0.528	0.515	0.502	0.487	0.470	0.454	0.442	0.431
0.1	1.580	0.214	0.203	0.197	0.194	0.191	0.189	0.186	0.184	0.181	0.179	0.176	0.174
0.2	1.420	0.190	0.175	0.166	0.160	0.154	0.149	0.143	0.138	0.134	0.129	0.125	0.122
0.3	1.386	0.179	0.160	0.149	0.141	0.133	0.125	0.118	0.112	0.106	0.101	0.097	0.093
0.4	1.377	0.171	0.151	0.138	0.128	0.118	0.109	0.101	0.094	0.088	0.083	0.078	0.075
0.5	1.346	0.166	0.144	0.130	0.118	0.107	0.097	0.089	0.081	0.075	0.069	0.065	0.061
0.6	1.442	0.161	0.139	0.124	0.111	0.099	0.088	0.079	0.071	0.065	0.059	0.055	0.051
0.7	1.427	0.158	0.135	0.119	0.104	0.092	0.081	0.071	0.063	0.057	0.051	0.047	0.043
0.8	1.369	0.156	0.131	0.114	0.099	0.086	0.074	0.065	0.057	0.050	0.045	0.040	0.036
0.9	1.389	0.157	0.129	0.111	0.095	0.081	0.070	0.060	0.052	0.045	0.040	0.035	0.031
1	1.327	0.159	0.126	0.108	0.091	0.077	0.066	0.056	0.048	0.042	0.036	0.032	0.028

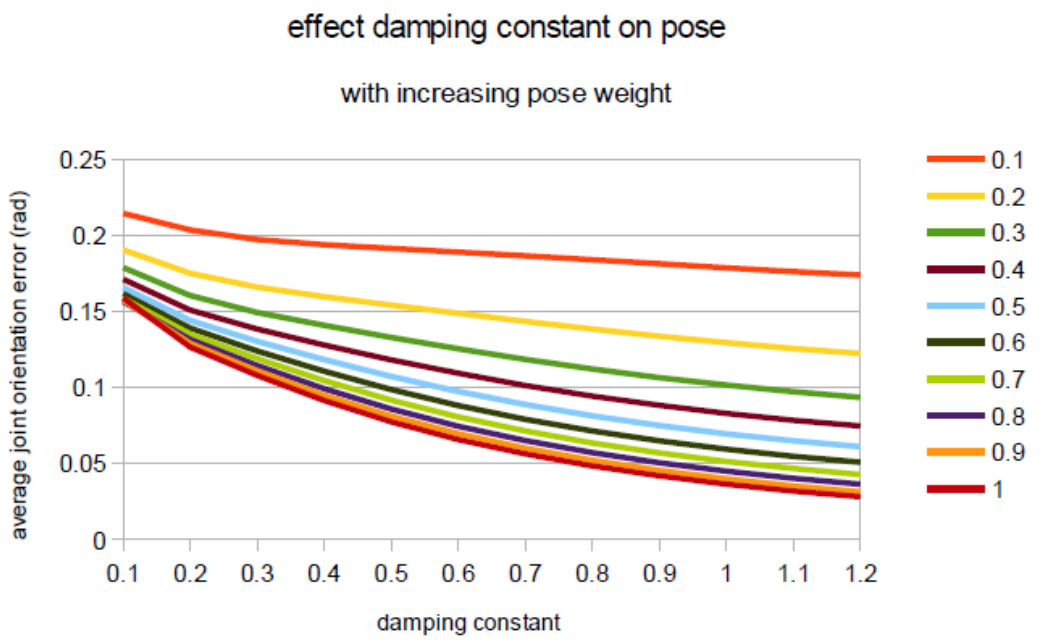


Figure 8: *Pose error*: the average error in radians of eight unique joints between the Kinect source data and the kinematic target data.

the input skeleton, as well as the fact that the dimensions of the input skeleton change during the motion [OKO*12].

Figure 8 shows that a higher pose weight decreases the orientation error of the joints, as the pose tracking is weighted heavier. A pose weight of > 1 was not tested because pose weight $= 1$ means the pose tracking reaches the desired pose in a single IK step. A higher value results in the target pose being overstepped, causing oscillations. A higher damping constant also decreases the pose error, as the end–effector tracking is damped and has a smaller effect on the kinematic target. More interesting are the end–effector position error results in Figure 9. Firstly, it shows that end–effector tracking is unaffected by pose tracking, and pose tracking even increases the end–effector tracking performance. This means projecting the pose target onto the Jacobian null–space has the desired effect of allowing pose tracking without interfering with end–effector tracking.

Secondly, the effect of the damping constant on the end–effector tracking can be clearly observed. A damping constant that is too low (< 0.3) results in unstable tracking and a higher end–effector error. A damping constant that is too high (> 0.6) results in sluggish tracking that is detrimental

pose weight \ damping constant	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2
0	0.579	0.101	0.102	0.104	0.107	0.110	0.113	0.116	0.118	0.121	0.123	0.124	0.126
0.1	0.566	0.101	0.100	0.101	0.102	0.102	0.102	0.103	0.103	0.103	0.104	0.104	0.105
0.2	0.489	0.101	0.100	0.099	0.098	0.098	0.097	0.097	0.097	0.098	0.099	0.100	0.101
0.3	0.466	0.101	0.099	0.098	0.096	0.095	0.095	0.095	0.095	0.096	0.097	0.098	0.100
0.4	0.449	0.100	0.099	0.097	0.095	0.094	0.094	0.094	0.095	0.096	0.097	0.099	0.100
0.5	0.432	0.100	0.098	0.096	0.094	0.093	0.093	0.094	0.095	0.096	0.098	0.099	0.101
0.6	0.490	0.100	0.098	0.095	0.093	0.093	0.093	0.094	0.096	0.097	0.099	0.100	0.101
0.7	0.497	0.100	0.097	0.095	0.093	0.093	0.093	0.095	0.096	0.098	0.100	0.101	0.102
0.8	0.466	0.100	0.097	0.094	0.093	0.093	0.094	0.095	0.097	0.099	0.100	0.102	0.103
0.9	0.490	0.100	0.097	0.094	0.093	0.093	0.094	0.096	0.098	0.100	0.101	0.103	0.104
1	0.440	0.100	0.096	0.093	0.092	0.093	0.095	0.097	0.099	0.100	0.102	0.103	0.105

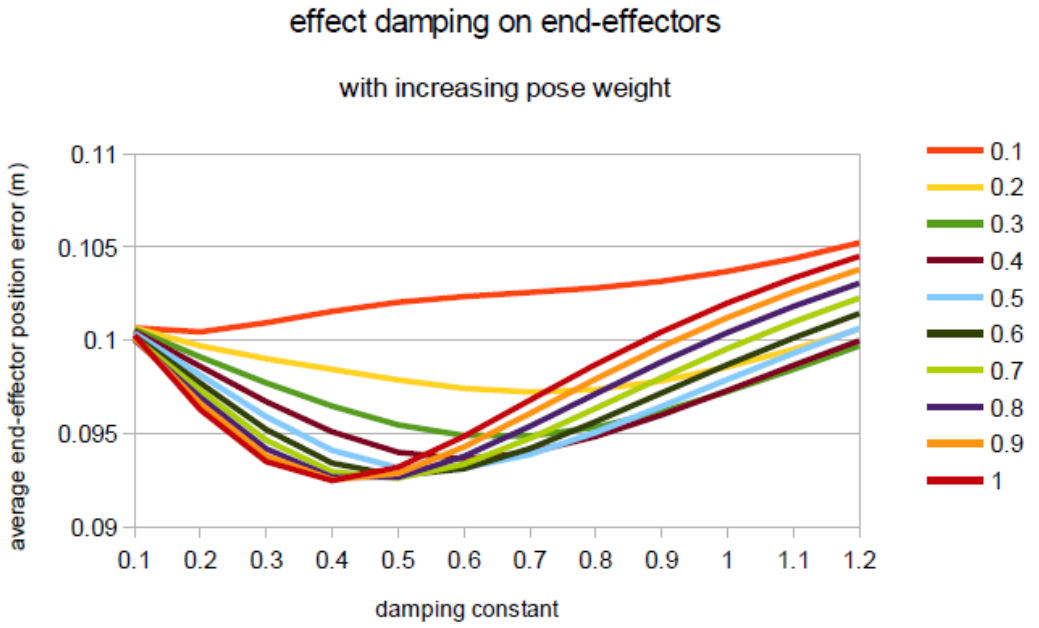


Figure 9: *End–effector error*: the average error in meters of five end–effectors between the Kinect source data and the kinematic target data.

to the performance. It should be noted that contrary to the joint orientation error, the average end–effector position has a best–case error that is larger than 0, as the dimensions of the dynamic character do not exactly match that of the skeletal data returned by the Kinect.

Additionally, the root rotation of the character, shown as the rotation of the pelvis body part in Figure 7, can also make end–effector targets unreachable. See, for instance, the top–middle pose in Figure 7, where both feet have the same end–effector target height. Due to the rotation of the pelvis, the left foot cannot reach the same height as the right foot, resulting in an end–effector position error. Choosing another root rotation than the current one, which uses the sensor input data, is not an option as that means the kinematic target no longer represents the input pose, which relies on the global orientation of the character. These issues combine to make certain end–effector target positions unreachable during certain poses by the dynamic character, resulting in a systematic positional error.

For subsequent experiments, a pose weight of 1.0 is used, as it provides the most faithful pose tracking, without negatively affecting end–effector tracking. As accurate end–effector tracking is required for precise real–time control of hands and feet by the user of the system, a low end–effector

error is targeted. Therefore, a damping constant of 0.4 is chosen for subsequent experiments, as Figures 9 and 8 show that it results in the lowest end–effector error, while introducing a manageable pose error. While a higher damping constant increases pose accuracy without hurting end–effector tracking significantly, it does introduce a more noticeable damping to the tracking response. This is detrimental to on–line control of the dynamic character, as it reduces the reactivity of the character to the input motion.

4.3 PD controller parameters

The PD controller gains (k_p and k_d, see Equation 3.7) determine the size of the torque value applied to a joint of the dynamic character, to reach a certain target orientation of that joint. The amount of torque applied to the physics body, has a significant effect on the simulation and the resulting motion of the character. This effect is also dependent on the skeletal configuration and proportions of the physics character, which necessitates experiments to determine balanced values for the PD–gains. The value for k_d can be derived from the value of k_p, by the following relation: $k_d = 2\sqrt{k_p}$ [GP12]. This means only the value for k_p needs to be found experimentally.

Each joint DOF affects a different part of the body when moving. This means each DOF has to exert a different amount of torque to move those body parts, depending on the mass and the pose of the body. This suggests that a single k_p for all joint DOFs does not result in the best performance. However, our solution scales the k_p value by the moment of inertia of the affected body parts, which means a single parameter ($\hat{k}_p$ or Kp in the figures) can be tuned for all PD controllers, without negatively affecting performance [ZH02]. The $\hat{k}_p$ value multiplies the k_p ratios, whereby the x–axis of the spine joint is roughly given a ratio of 1, and all other DOFs have a k_p ratio relative to that.

The ratio of k_p values between different DOFs can also be found by manual tuning of values, until a visually convincing simulation is shown. Or the k_p ratios can be calculated dynamically each time step, as described above, based on the current pose. The latter approach is more accurate, as it adapts the PD gains to the current pose. The first approach acts more predictably, as the gains are tuned and known beforehand.

Setup This experiment should demonstrate the effect of high and low gains on the dynamic tracking, as well as compare manually tuned k_p ratios with dynamic moment of inertia calculations. High gains are expected to cause oscillating motion, as well as unbalancing the character due to sudden movements. Low gains are expected to make the character less responsive or even collapse, as the effect of gravity cannot be countered by the low torques.

To demonstrate these effects, a range of $\hat{k}_p$ values is used to perform a pre–recorded motion (25s). The pre–recorded motion clip shows a range of motions, similar to Figure 7, except without moving both feet from or on the ground, to maintain a stable stance. $\hat{k}_p$ iterates over a range of 300–1700 with steps of 50. The performance is measured in two ways, comparing the kinematic target and the dynamic character. The average joint orientation error is measured in world coordinate frame and parent joint coordinate frame. Both of these error measures provide different and relevant data. For instance, when the dynamic character falls over whilst performing a balancing motion, the world coordinate error measure shows this falling as a large error in the orientation of all joints, relative to the world. The parent coordinate error measure on the other hand, ignores the fall, as the orientation of each body part relative to its parent body part does not change, even though the global orientation of the character changes as he falls. The parent coordinate error measure instead shows the accuracy of the pose the dynamic character takes and provides that data without being obscured by global orientation errors.

These errors are shown both averaged over all joints and with the worst performing joint (the

worst–case error). The wrist joints are not included in these errors, as they are unstable in the pre–recorded motion and would have a disproportionately large effect on the average and worst–case errors. The data is recorded each time the kinematic target pose changes (30 times a second). The range of $\hat{k}_p$ values is performed twice, once with the manually tuned k_p ratios, and once with the dynamic moment of inertia calculation.

In order for this experiment to return valuable data, the dynamic character should stay balanced and upright. Initial experiments revealed that this is unlikely without the balance control enabled. Therefore, balance control is used during these performances. Additionally, the stability of the physics simulation could not be ensured with a simulation step size large enough to allow for real–time playback of the pre–recorded motions. Therefore, a significantly smaller simulation step size of $0.0001s$ was used for these experiments, resulting in a stable simulation at roughly half the real–time speed: $60ms$ per frame, compared to $33ms$ for real–time playback.

Results

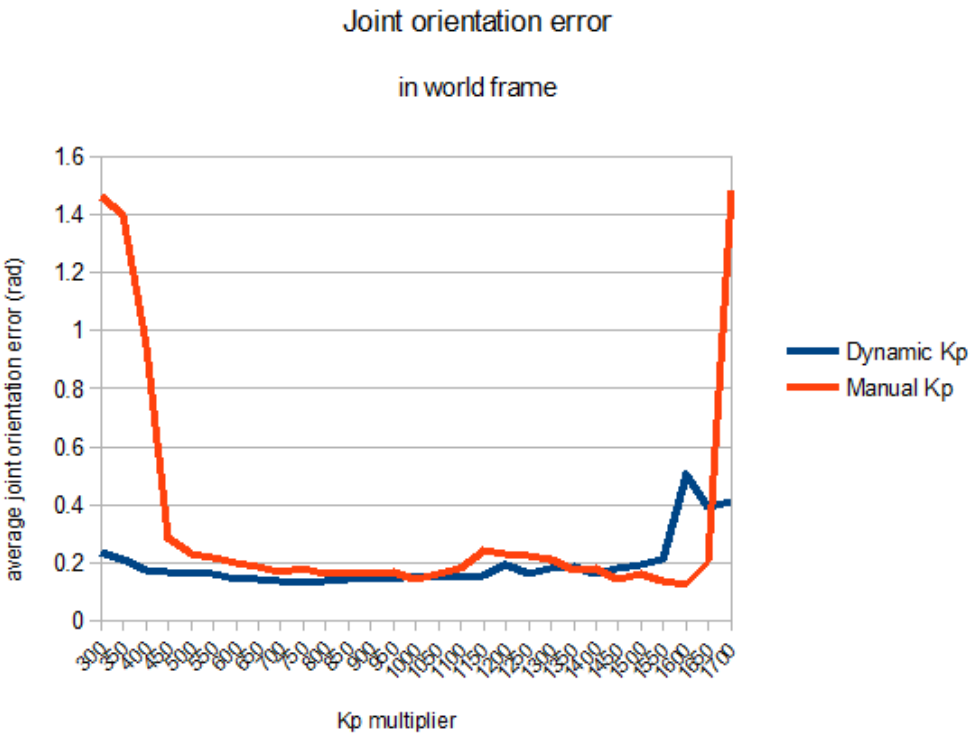


Figure 10: Joint orientation error of dynamic character as a result of varying $\hat{k}_p$. In world coordinate frame.

Figures 10, 12, 11 and 13 show the results of varying the $\hat{k}_p$ value on the performance of the physics simulated character and shows a comparison between using manually tuned k_p ratios and dynamically calculated ratios. The first thing to note about these graphs is that the manually tuned k_p ratios perform significantly worse at low and high $\hat{k}_p$ values, while the dynamically calculated ratios degrade more gracefully, maintaining a respectable performance with very low and very high torques.

Around the optimal $\hat{k}_p$ values, where the errors are the lowest, the two methods compare more closely. These observations combined suggest that while it is possible to manually tweak the k_p ratios for a better performance with a specific amount of torques, the dynamic ratio calculation

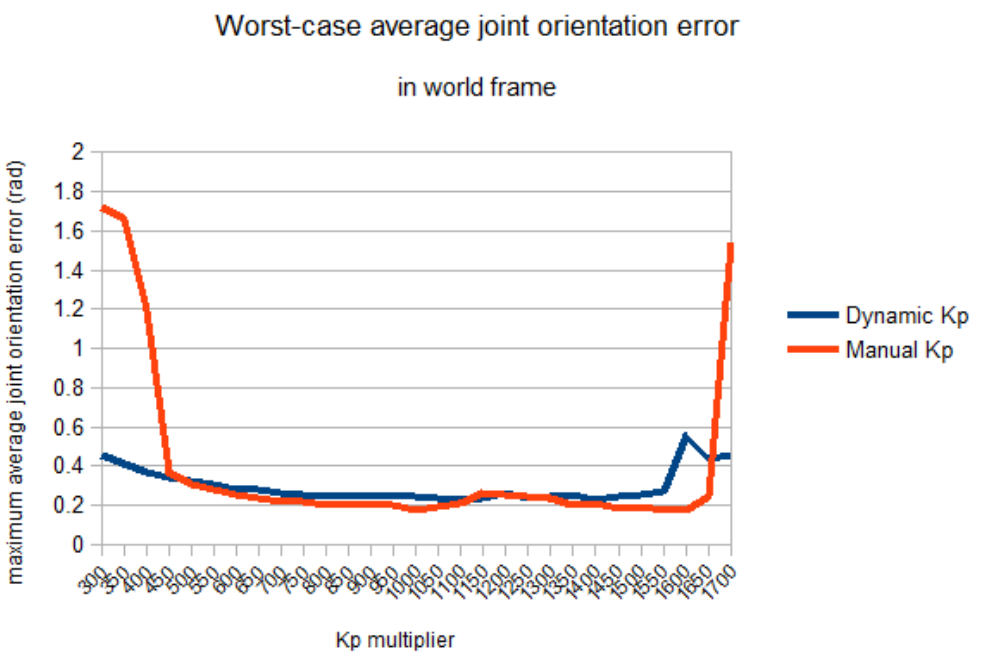


Figure 11: Average joint orientation error of the worst performing joint as a result of varying $\hat{k}_p$. In world coordinate frame.

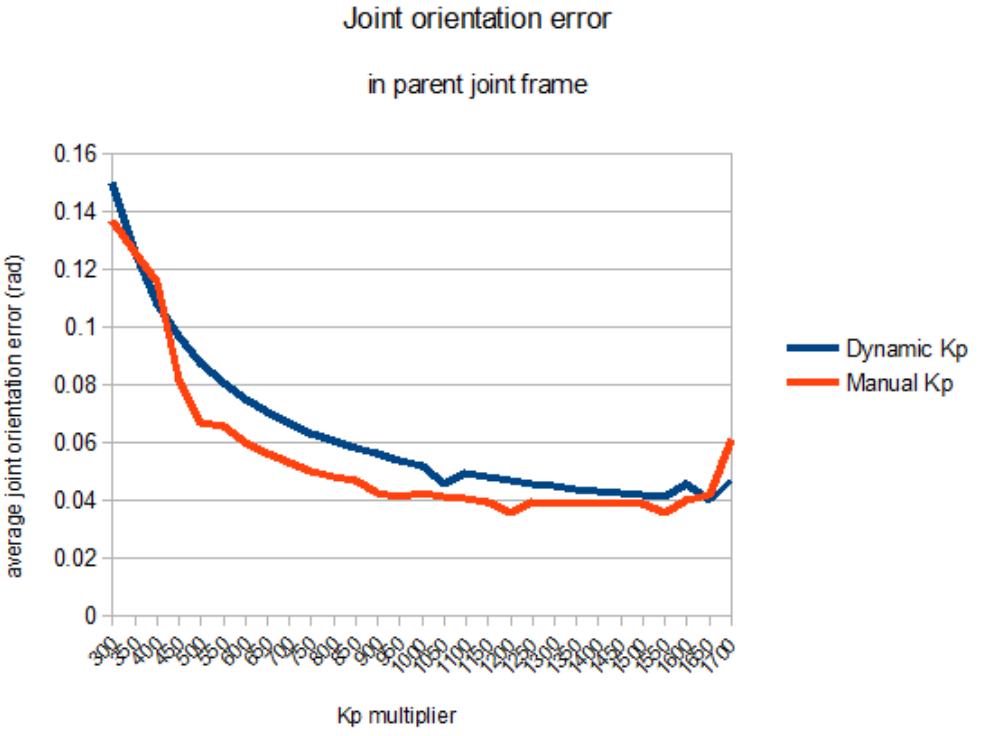


Figure 12: Joint orientation error of dynamic character as a result of varying $\hat{k}_p$. In parent joint coordinate frame.

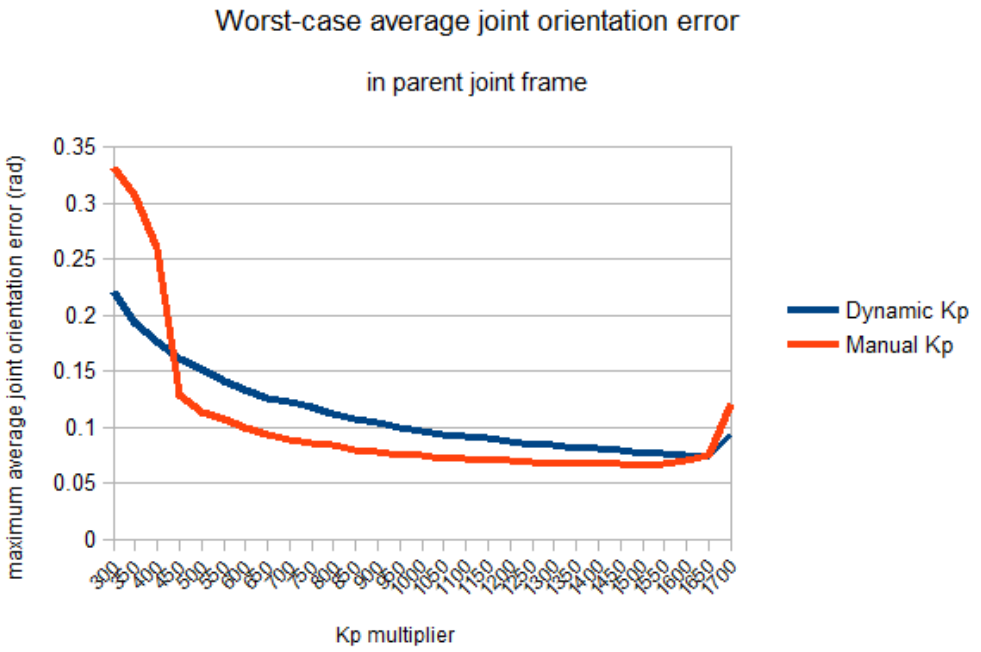


Figure 13: Average joint orientation error of the worst performing joint as a result of varying $\hat{k}_p$. In parent joint coordinate frame.

based on moment of inertia is more robust under varying torque scales. Figure 10 shows that dynamically calculated ratios perform slightly better with a lowest error around $\hat{k}_p = 800$. Figure 11 shows that the worst performing joint with each value of $\hat{k}_p$ performs slightly better with manually tuned ratios.

Figure 12 shows that the average joint orientation error in parent joint coordinate frame decreases with higher torques. Higher torques causes the dynamic character to more quickly move towards the target pose, reducing the pose error. This decrease is predictable until a value of around $\hat{k}_p = 1500$, where the high torques cause instability in the pose. The worst–case error shown in Figure 13 shows the same results, with manually tuned k_p ratios performing better with most $\hat{k}_p$ values.

Note the small scale of the y–axis of figure 12, which means that any (reasonable) $\hat{k}_p$ value results in a reasonable target pose reproduction by the physics character. Similarly, Figures 10 and 11 show that within a range of multiplier values of $700 < \hat{k}_p < 1200$, the pose error in world frame is equally low. A lower value for $\hat{k}_p$ is preferable, as it causes lower torques to be applied to the character, resulting in a more fluid motion. From this, we can conclude that a value of $\hat{k}_p = 800$ is most preferable, and will thus be used in subsequent experiments.

At a value of $\hat{k}_p = 800$, the graphs show that the manually tuned k_p ratios perform slightly better, especially in the worst–case. However, the dynamically generated ratios perform almost identically, and will automatically deal with differences in character morphology. The results show that manual tuning is not necessary for a good performance, and the dynamic ratios are thus preferable. The dynamically calculated k_p ratios will be used for subsequent experiments.

4.4 Balance

A balance strategy is introduced into our system to ensure that in–place motions that do not offer a significant balancing challenge are performed well. Because the system is restricted to using physically plausible internal joint torques, balance cannot be enforced by introducing external forces. Instead, the kinematic target pose is adjusted on–the–fly by rotating the ankle joints to control the center of mass (COM). By adjusting the kinematic target, the visual signature of the performed motion could change. The kinematic target processing using inverse kinematics (IK) is introduced to the tracking solution, to improve the quality of the kinematic target motion for balanced in–place motion. To demonstrate its performance, this experiment will test both the balance control, as well as the use of IK for the generation of the kinematic target.

Setup The balance strategy has three parameters that need to be varied to find the best configuration. The first parameter is the clamping value used to limit the angular velocity of the ankles as a result of the balance control. The second parameter controls the positioning of the COM by the balance strategy, between positioning it exactly above the center of support (COS) with a value of 0 and positioning it relative to the COS similar to the kinematic target with a value of 1. An intermediate value (i.e. in between 0 and 1) results in the balance strategy positioning the COM of the physics character between these two targets. The latter results in a pose more like the target motion, as the COM offset of the kinematic target is taken into account. The former offers a more balanced pose, as the COM of the dynamic character is positioned exactly above the ground contact of the character, providing the least possibility of the character falling over during an in–place motion.

The third and last parameter controls the effect of the COM velocity on balance control. With a higher value causing the balance strategy to more strongly correct in the opposite direction, to compensate for the global motion of the character. A value of 1 means that the vector moving the COM target in the direction of the COM velocity has a maximum length that is the same as the length of the vector from the COS to the COM target before COM velocity compensation. A parameter value between 0 and 1 causes that COM velocity compensation vector to be multiplied with that value, resulting in a weaker COM velocity compensation of the COM target.

Initial testing with a large variety of configurations of these three parameters (see Figure 14) revealed promising initial values for the parameters. Figure 14 shows five values being used for each of the three parameters. Each graph has results for a particular COM balance target value (see the subtitle above each graph). Each curve in the graphs shows results for a particular value of the COM velocity compensation parameter (see the graph legend). The X–axis of each graph represents the ankle angular velocity clipping parameter.

The figures show some unexpected variation in the results, caused by the physics simulation not being deterministic. However, the rough trends in the graphs are similar, suggesting little interdependence between the parameters. These observations resulted in the decision to use the most promising values of these initial tests while varying each parameter individually, and repeating the experiments a number of times (three), to average out the variable nature of the physics simulation. This greatly reduces the combinatorial complexity of the experiment, and allows for more detailed testing of the individual parameters. Both the COM shift parameter and the COM velocity parameter take values between 0 and 1, and are tested with a step size of 0.1.

The clamping parameter is tested with values between 40π and 180π with steps of 14π, as lower values caused the character to fall over from too little ankle movement, and higher values caused little clamping to occur, resulting in the parameter having little effect. We use the π notation of these values as it denotes the maximum angular velocity (rad/s) of the ankle joints as a result of balance control. The values within this range are all very high angular velocities for a joint. The reason for these high values is that the physics simulation time step is very small ($0.0001s$), and

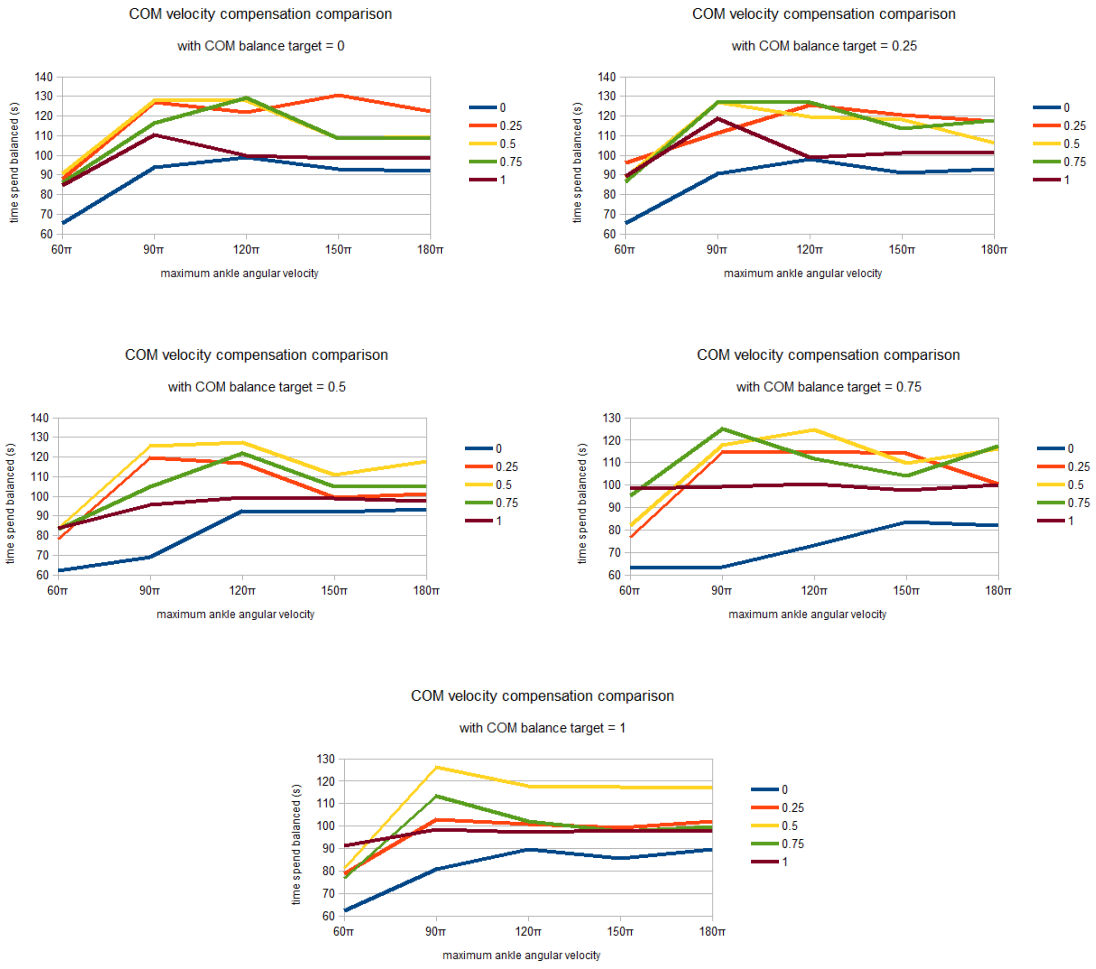


Figure 14: Initial results of all combinations of 5 values of the 3 parameters of the balance strategy

values lower than 40π suffer from the (lack of) floating point precision, resulting in an orientation change in a single time step that is too small for a floating point number to represent. Instead, those lower clamping parameter values were rounded off in a single time step to allow for no rotation, therefore completely negating the balance strategy.

After finding the best configuration of the balance strategy, the eight pre–recorded motions are performed with four configurations of our system:

- *without* IK kinematic processing and *without* balance control.
- *with* IK kinematic processing and *without* balance control.
- *without* IK kinematic processing and *with* balance control.
- *with* IK kinematic processing and *with* balance control.

The duration that the physics character stays balanced during each motion clip is recorded, and added up to a total time spent balanced. The character is considered to have lost its balance if the COM height is less than 80% of the COM height of the kinematic target motion. We chose this value as the COM height of the dynamic character differs more than 10% from the kinematic

target during some motion clips, while still remaining balanced. So to avoid false positives, we chose a safe threshold of 80% of the COM height of the kinematic target as a condition for losing balance.

Results

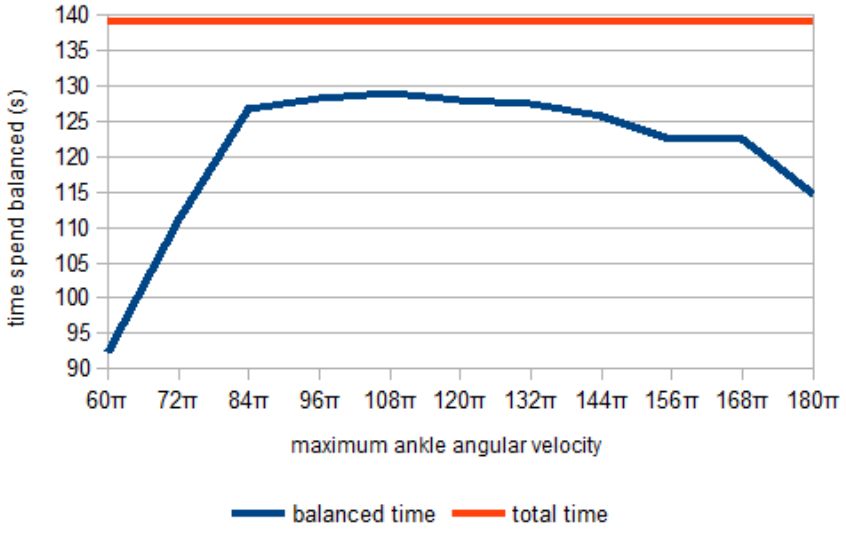


Figure 15: The performance of the balance strategy while clamping the ankle joint angular velocity at different values.

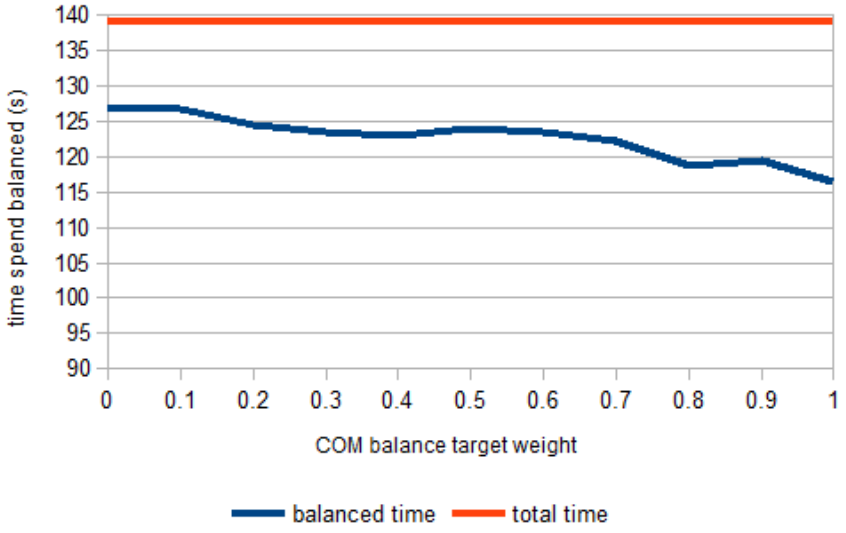


Figure 16: The performance of the balance strategy while varying the position of the COM relative to the COS.

Figures 15, 16 and 17 show the balancing performance of the balance strategy, with different parameter values. The blue curve shows the accumulated time (in seconds) that a character stayed upright while performing a series of pre–recorded motions. The red line is the total time of those

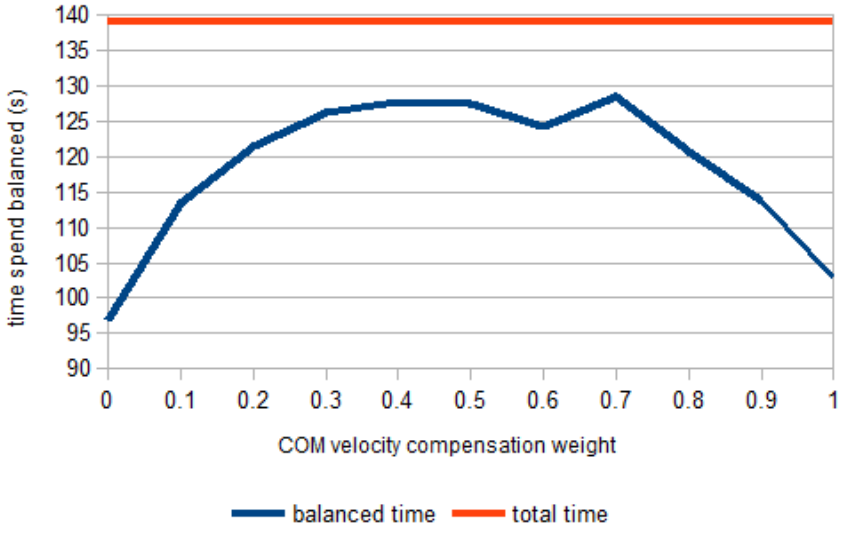


Figure 17: The performance of the balance strategy while varying the COM velocity compensation weight.

total	no IK, no BC	IK, no BC	no IK, BC	IK, BC
139.0	112.2	112.9	128.4	128.1

Table 3: Time (s) spent balanced with different configurations. *BC* = balance control, *IK* = IK–based kinematic processing.

motion clips combined, which is the maximum time the character could stay balanced. Those graphs show the time that the physics character spend balanced averaged over three performances, as the physics simulation introduces some uncertainty into the experiment, thus requiring averaging over multiple runs of the experiments to get a reliable result.

Figure 15 shows that there is a large range of clamping values of the ankle angular velocity, that result in a similar good balance of the physics character. Any value above 84π is high enough to avoid clamping the balance strategy too strictly. Similarly, values below 132π result in sufficient clamping of the ankle orientation to smooth out its movement and avoid loss of balance due to sudden motions of the ankle joints. We found that a value of 108π results in the best performance.

Figure 16 shows that the center of mass (COM) shift parameter does not have a large effect on the performance of the balance strategy. A value of 0 of this parameter means that the COM is positioned directly above the center of support (COS) of the physics character. A value of 1 means that the offset of the COM relative to the COS in the kinematic target motion is calculated, and the balance strategy aims to position the COM of the dynamic character with a similar offset. A value closer to 0 is expected to result in a more balanced performance, which can be seen in Figure 16, but this trend is not pronounced in the graph. Regardless of the minor effect, a value of 0 results in the best performance, and is therefore used for the best balance strategy configuration used for Table 3. A higher value results in a pose more faithful to the target, and might be more preferable, while having only a slightly worse balance performance.

Figure 17 shows that using no COM velocity compensation (a parameter value of 0) and using strong compensation (a value near 1) results in a less balanced character. So COM velocity compensation is necessary for the best balance performance. Overcompensation however is equally harmful, causing the character to lose balance due to large rotations by the ankle joints to com-

pensate for character movement. A value of around 0.4 results in the best performance. a notable anomaly in the graph occurs at a value of 0.7, where the performance is better than the rest of the graph suggests for that value. Even after averaging over three runs of the same experiment, this anomaly was present in the results.

Table 3 shows the performance of the dynamic character when disabling or enabling different parts of our method. These results are again averaged over three runs of the same experiment. When not using balance control, the time spent balanced was considerably shorter compared to dynamic balancing. The effect of the kinematic processing on the balancing performance of the character is negligible. The best performance of the system was achieved without kinematic processing and with balance control. This defies the prediction that the absolute positioning of end–effectors by the kinematic processing method results in a more stable ground contact. The fact that no configuration of the system was able to perform all motions in full, without losing balance, shows that the system is only capable of handling a subset of in–place motions. It fails to perform a motion like a small jump.

4.5 Morphology

Our method should handle a variety of different morphologies of the dynamic character. There are however factors which affect the motion performance when the morphology is changed. For instance, because end–effector positions are tracked absolutely, a smaller character will reach towards these positions, distorting the pose. This experiment demonstrates these effects, by showing how our method handles morphologically differing characters without changes to the system or the parameters.

Setup In order to demonstrate the performance of our system, the same balance experiment as described above is performed by the different characters. The results can then be compared to the results of the character morphology used in the previous experiments (Figure 3). Several types of morphology changes are tested:

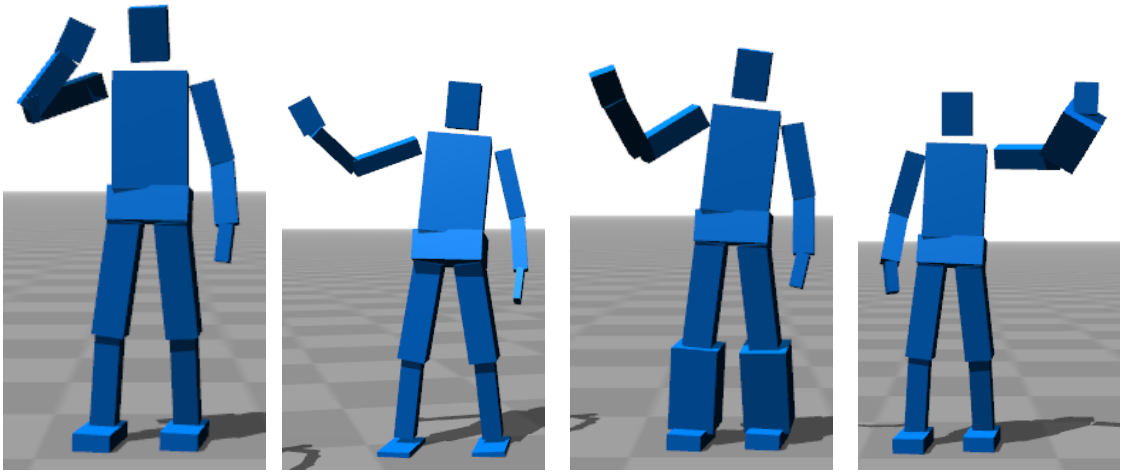
- Correct body weight distribution, as seen in Table 2.
- Significantly thicker lower legs, with each having a weight of 19.2 kg instead of 4.8 kg, to show performance with a lower COM.
- A single significantly thicker lower arm, with a weight of 11.2 kg instead of 2.8 kg, to show performance of an asymmetric character.

Results

original	weight distribution	lower COM	asymmetric
128.1	42.4	128.0	92.7

Table 4: Time (s) that characters with different morphologies stay upright and balanced while performing several different motions.

When observing the various motions being performed by the character with a correct weight distribution (Figure 18b), it appears to suffer from the feet sliding on the floor plane and as a result losing balance. This is reflected in the poor performance of our method with this character (see Table 4). The foot sliding can be explained by the fact that the character with correct weight distribution has feet body parts with a significantly lower weight than the original simulated character. This means that the large torques applied to the ankles to keep the character upright



(a) Original (b) Weight distribution (c) Low COM (d) Asymmetric

Figure 18: The different character morphologies tested (not to scale).

cause oscillations of the feet body parts, causing the character to wiggle its feet and appear to slide on the ground plane as a result. This effect is a result of the way in which our physics engine (ODE) resolves collisions, and the effect has previously been observed by Geijtenbeek et al. [GPvdS12].

The performance of the character with a lower COM (and higher weight) shown in Figure 18c is equal to the original physics character (Figure 18a). While a lower COM can be beneficial to balance, the higher torques required to move the large lower legs introduces a risk of foot sliding or instability. Our method correctly adapts the torques applied to the relevant joints to compensate for the larger weight of those body parts, allowing for most motions to be performed while staying balanced, resulting in a performance similar to the original dynamic character.

The performance of an asymmetric character (Figure 18d) with larger weight in a single arm fall significantly short of the performance of the original physics character. This shows that while the balance strategy balances the COM of the character, which takes into account a asymmetric weight distribution, it is not capable of keeping a character with a significant weight offset balanced. Additionally, the fact that a large weight is connected to the arm, means that movement of that arm has an exaggerated effect on the motion of the entire upper body, causing stability issues.

5 Discussion and Future Work

The goal of our system, as outlined in the introduction, was: *to perform intuitive real–time in–place control of a physically plausible virtual biped using consumer grade video–based motion capture hardware.* The experimental results give an indication of the performance of our system with respect to the various subgoals contained in this goal statement. The following section discusses the performance of the system.

5.1 Discussion

There are several keywords in the above goal statement that indicate specific requirements for the system to fulfill the goal.

Intuitive control Intuitive control of a virtual character is achieved by our system, as the virtual character copies the motion of the user directly. The physics simulated character faithfully reproduces the poses of the user, as is evident from the low joint orientation errors in Figures 10 and 12. However, the fact that the virtual biped is incapable of staying balanced during some in–place motions, as shown in Table 3, means the user has to consider whether the virtual character will be able to stay upright when deciding what movement to perform in front of the motion sensor. This detracts somewhat from the ability of our system to provide intuitive control, as not all motions can be predictably reproduced by the physics character.

Real–time control We were unable to run the experiments in real–time. The physics engine that was used required a very small time step for its simulation to be stable with the large torques applied to the body parts of the virtual biped. Requiring 10000 iterations per simulated second, the physics engine was not able to run the simulation in real–time. Instead the experiments had to be performed with pre–recorded motion clips at roughly half the speed of real–time: 60ms per motion frame, as opposed to 33ms in real–time between motion frames of the Kinect sensor. We think real–time control of a physics–based character with this method is likely to be within reach, because we did not spend significant effort optimizing the implementation.

Physically plausible virtual biped We define a physically plausible virtual character as a physics simulated character that moves solely using internal torques that act on its joints. This is in contrast to the common use of a physics character controlled by external forces acting on the body. Our method does exclusively use internal torques to control the virtual biped, and in doing so allows the user to control a physically plausible character. Restricting the system to only using plausible joint torques does introduce balance issues, that we were only able to partially relieve by using balance control. This is best demonstrated by the results in Table 3, which shows that of 139 seconds of motion, the dynamic character controlled using internal torques had fallen over and was not upright for more than 10 seconds (7.6%). Introducing external forces to keep the physics simulated biped upright would have resolved this issue, but would also likely have resulted in a character that moves less visually realistic and reacts less realistically to perturbation from the virtual environment, due to the use of unrealistic external forces.

Consumer grade video–based motion capture hardware In order to use motion data from a user in real–time (as well as for pre–recording), we used a Microsoft Kinect motion sensor. The use of this piece of consumer grade motion capture hardware introduced some challenges that are not likely to occur with professional motion capture setups. Firstly, the single perspective of the Kinect sensor means that one part of the body can be occluded by another part of the body, from the perspective of the sensor. This, together with the fact that the sensor uses depth image analysis rather than physical markers to determine the pose of the user, results in occasionally unstable motion data. While we allowed this incorrect motion data to be used by the physics character, we did not allow it to negatively impact the balance of that character. The use of a balance strategy ensured the balance of the virtual biped, even with minor instability in the input data, as evidenced by Table 3.

A systematic issue with the Kinect motion data is that the length of body parts is not enforced, which means that during a motion the body proportions can change. In order to address this, we developed a novel use of the damped least squares inverse kinematics (DLS IK) technique. This IK method is used to track the absolute position of some end–effectors, like the ankle and wrist joints, thereby ensuring the stable positioning of those end–effectors in the kinematic target motion. This is shown in Figure 9, which demonstrates that using DLS IK decreases the positional error of those end–effectors significantly. In order to ensure that the pose of the kinematic target matches the input motion data, the joint orientations in the input data are projected onto the null–space of the Jacobian matrix used for DLS IK. This means the degrees of freedom of the character that are not

used to position the end–effectors, are used to match the pose of the user of our system. Figures 8 and 9 show that this use of the DLS IK method with pose tracking as secondary objective is successful, resulting in a smaller error in the positioning of the end–effectors, while the pose data of the input motion is faithfully recreated in the kinematic target motion.

We hypothesized another benefit of the more stable positioning of the ankle end–effectors specifically. By using DLS IK to absolutely position the feet according to the input data, the character would have a more stable footing on the ground plane during challenging motions. We did not, however, observe this effect during our balance experiment, as shown in Table 3. The virtual biped stayed balanced for the same amount of time whether using DLS IK for kinematic processing, or not.

Varying character morphology Our system is capable of handling differently proportioned virtual bipeds, without requiring additional tuning of parameters. This is possible because it is an entirely real–time system, without off–line optimization or the use of pre–processed or pre–existing data. Additionally, by scaling the torques applied to the physics simulated character by an estimate of the moment of inertia, the PD–controllers used to control the biped automatically adapt to differently proportioned or heavier characters. Table 4 shows that our system is capable of handling some character morphologies, like significantly larger lower legs, while performing significantly worse with other morphologies, like a significant asymmetric weight distribution or a human–like weight distribution.

The performance of the biped with a human–like weight distribution reveals an issue with the methods we use to control and balance the physics–based characters. The much smaller weight of the feet of this character, compared to the original visually motivated character, resulted in the character with realistic weight distribution being unable to remain stationary on the ground plane. This is caused by the large torques applied to the ankle joints to keep the character balanced, which cause the lightweight feet to oscillate quickly and make this character slide on the floor plane and lose balance. This demonstrates that our system does not gracefully degrade, when facing different character morphologies or difficult motions. This limits the usability of our system to some extent, as certain character cannot be used and certain motions cause irrecoverable stability issues. The acceptable range of use cases of our system is therefore somewhat limited.

In conclusion, the main contribution of our method is the novel use of a Jacobian–based inverse–kinematics technique for the accurate positioning of some essential end-effectors, while using projection onto the Jacobian matrix null–space to ensure a pose faithful to the input data. This real–time kinematic data processing technique deals well with the varying body proportions and occasionally unstable data of Kinect sensor motion data, resulting in a more stable kinematic target for the physics simulated biped to follow. Additionally, within a limited range of in–place motions that do not significantly challenge the balance of the physics character, our system faithfully reproduces the input motion with the physics character, in close to real–time and without requiring any off–line optimization or existing motion data. Also, the system achieves this by using only physically plausible internal joint torques, which means the virtual character reacts realistically to physical interactions with the virtual environment, while being controlled by the user.

5.2 Future Work

Real–time application A logical direction of future work would be to focus on the real–time application of our method. Besides optimizing the implementation for true real–time use, there is an interesting venue of investigation in the interaction between a user of the system and the physics–based character. Specifically the interaction of the user with the virtual environment and whether or not a user performs corrective movements when the virtual character is at risk of losing its balance, for instance.

Locomotion We limited the use of our system to in–place motion to simplify the balance problem and as a result of the limited range of the Microsoft Kinect sensor. Future work could focus on making this system ready for locomotion, which requires a completely different balance strategy. Additionally, locomotion within a virtual environment that does not reflect the real world environment, like a virtual sloped floor, introduces a whole new set of challenges to the control and balance of the physics character.

Accurate virtual biped Our system uses a significantly simplified representation of a human. This means there are several valuable directions of future work, focusing on a more accurate physical representation of a human for the physics simulation.

For example, the use of self–collision, where a body part of the virtual character can collide with another body part. While this results in a more realistic simulation, self–collision can make motion tracking more difficult due to the constraints it introduces.

Another improvement with similar benefits and difficulties is the use of realistic joint orientation limits. These limits ensure that a virtual joint cannot be in an orientation that is impossible for the same joint in a human to achieve.

Another direction of future work that would benefit the visual and physical realism of the simulation, is to model the virtual character more closely to a human. This is achieved by using more complex collision volumes than boxes for the physics character, in order to model a human more accurately both for the physics simulation, as well as visually.

Physical accuracy could also be improved by modeling the forces and torques applied to the virtual biped more closely to the forces and torques that human joints are subject to during different motions. An example of this would be the use of a human muscle–based model for control of the virtual character, similar to the work Geijtenbeek, van de Panne and van der Stappen [GvdPvdS13] have done on bipedal locomotion. This can have a significant positive effect on the stability of the virtual movement and the physics simulation.

References

[BK05] Buss S. R., Kim J.-S.: Selectively damped least squares for inverse kinematics. *journal of graphics, gpu, and game tools 10*, 3 (2005), 37–49.

[Bus04] Buss S. R.: Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation 17* (2004).

[CBvdP10] Coros S., Beaudoin P., van de Panne M.: Generalized biped walking control. In *ACM SIGGRAPH 2010 papers* (New York, NY, USA, 2010), SIGGRAPH '10, ACM, pp. 130:1–130:9.

[CCH11] Chang Y.-J., Chen S.-F., Huang J.-D.: A kinect–based system for physical rehabilitation: A pilot study for young adults with motor disabilities. *Research in Developmental Disabilities 32*, 6 (2011), 2566–2570.

[CK99] Choi K.-J., Ko H.-S.: On–line motion retargetting. In *Computer Graphics and Applications, 1999. Proceedings. Seventh Pacific Conference on* (1999), IEEE, pp. 32–42.

[CMY69] Clauser C. E., McConville J. T., Young J. W.: *Weight, volume, and center of mass of segments of the human body.* Tech. rep., DTIC Document, 1969.

[DW74] Durnin J., Womersley J.: Body fat assessed from total body density and its estimation from skinfold thickness: measurements on 481 men and women aged from 16 to 72 years. *British Journal of Nutrition 32*, 1 (1974), 77–97.

[GP12] Geijtenbeek T., Pronost N.: Interactive character animation using simulated physics: A state–of–the–art review. *Computer Graphics Forum* (2012), 1–25.

[GPvdS12] Geijtenbeek T., Pronost N., van der Stappen A. F.: Simple data–driven control for simulated bipeds. *Eurographics/ACM SIGGRAPH Symposium on Computer Animation* (2012).

[GvdPvdS13] Geijtenbeek T., van de Panne M., van der Stappen A. F.: Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 206.

[IWZL09] Ishigaki S., White T., Zordan V. B., Liu C. K.: Performance–based control interface for character animation. In *ACM SIGGRAPH 2009 papers* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 61:1–61:8.

[LKI12] Lai K., Konrad J., Ishwar P.: A gesture–driven computer interface using kinect. In *Image Analysis and Interpretation (SSIAI), 2012 IEEE Southwest Symposium on* (2012), pp. 185–188.

[LKL10] Lee Y., Kim S., Lee J.: Data–driven biped control. In *ACM SIGGRAPH 2010 papers* (New York, NY, USA, 2010), SIGGRAPH '10, ACM, pp. 129:1–129:8.

[LZ11] Liu C. K., Zordan V. B.: Natural user interface for physics–based character animation. In *Motion in Games*, Allbeck J., Faloutsos P., (Eds.), vol. 7060 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 1–14.

[Mic13] Microsoft Corporation: Kinect for windows programming guide version 1.6. http://msdn.microsoft.com/en-us/library/hh855348.aspx, 2013. Accessed: 2013–09–01.

[OKO*12] Obdržálek Š., Kurillo G., Ofli F., Bajcsy R., Seto E., Jimison H., Pavel M.: Accuracy and robustness of kinect pose estimation in the context of coaching of elderly population. In *34th International Conference of the IEEE Engineering in Medicine and Biology Society* (2012), EMBC '12, IEEE.

[Pop10] Popov V. L.: Coulomb's law of friction. In *Contact Mechanics and Friction*. Springer Berlin Heidelberg, 2010, pp. 133–154.

[Ser13] Serway R.: *Physics for Scientists and Engineers with Modern Physics, 9th ed.* Cengage Learning, 2013.

[SH12] Shum H. P. H., Ho E. S. L.: Real–time physical modelling of character movements with microsoft kinect. In *VRST '12: Proceedings of the 2012 ACM symposium on Virtual reality software and technology* (New York, NY, USA, 2012), ACM.

[Smi06] Smith R.: Open dynamics engine v0.5 user guide. http://ode.org/ode-latest-userguide.html, 2006. Accessed: 2014–01–28.

[TGB00] Tolani D., Goswami A., Badler N. I.: Real–time inverse kinematics techniques for anthropomorphic limbs. *Graphical models 62*, 5 (2000), 353–388.

[YLvdP07] Yin K., Loken K., van de Panne M.: Simbicon: simple biped locomotion control. In *ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM.

[ZH02] Zordan V. B., Hodgins J. K.: Motion capture–driven simulations that hit and react. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2002), SCA '02, ACM, pp. 89–96.