REAL-TIME PHYSICS-BASED MOTION CONTROL WITH AN EFFICIENT INVERSE DYNAMICS METHOD

VERİMLİ BİR TERS DİNAMİK YÖNTEMİ İLE GERÇEK ZAMANLI FİZİKSEL HAREKET KONTROLÜ

ERSAN KAVAFOĞLU

ASSIST. PROF. DR. SERDAR ARITAN

Supervisor

Submitted to Institute of Informatics of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Master of Science
in Computer Graphics

2018

ABSTRACT

REAL-TIME PHYSICS-BASED MOTION CONTROL WITH AN EFFICIENT INVERSE DYNAMICS METHOD

Ersan Kavafoğlu

Master of Science, Department of Computer Graphics

Supervisor: Asst. Prof. Dr. Serdar Arıtan

January 2018, 74 pages

Realistic character animations can be obtained using motion capture techniques. However, these captured motions can be recorded only for predetermined scenarios. A virtual character can have infinite variety of physical interactions with the virtual world, and it is not possible to anticipate all these interactions and record appropriate motions. But in real life when people interact with the surrounding environment, their motions are generated as a result of physics laws. In physics based animation studies, it is aimed that the reaction movements of the characters in the virtual environment occur in a natural way according to the laws of physics.

In order to generate movements naturally, a simplified physical model needs to be controlled only by applying joint torques. These joint torques are calculated by benefiting from studies in fields such as robotics, biomechanics and physics. In order to be used in real-time applications, these calculations should be performed with very low processing costs, even for multi-body systems with very high degrees of freedom.

In the literature, studies on physics-based animation can be grouped under two main titles: local controllers and equations of motion based controllers. The most commonly used local controller is the Proportional Derivative controller because of its simplicity of integration and problem modeling. One of the main drawbacks of local controllers is the need to tune gain parameters for each movement and character manually. Moreover they are not quite stable at the speeds required for real-time applications.

In equations of motion based methods, modeling the problem and integration are more complex. However, these methods generate better results which are more stable than the results of local methods. Inverse dynamics constitutes the core component of equations of motion based methods. In physics based animation, Newton-Euler and Euler-Lagrange methods are used for inverse dynamics calculations. While Newton-Euler method is used to calculate the torques iteratively, Euler-Lagrange method is often used to obtain the analytical equations of motion needed for optimization problems.

In this thesis, we obtained generalized equations of motion for multi-body systems in 3D space, whose orientations are represented by quaternions and consisting of rotational joints with 3 degrees of freedom, by using Kane's method. During this study, we have observed that for complex multibody systems, it is not feasible to calculate the inverse dynamics solution analytically neither by hand nor by using symbolic programming from these equations of motions. In order to be usable in real-time applications, we derived a recursive inverse dynamics algorithm from these equations. This algorithm is equivalent to recursive Newton-Euler algorithms, which are often used in physics based animation applications. Unlike other studies, since we obtain this algorithm from an analytical equation, we have introduced an integrated approach that can be used for both motion planning and motion generation as well as for inverse dynamics. We tested the results of our method in different scenarios and observed that for all scenarios our method produces stable results even at large timesteps. We also compared our method with some of the widely used methods in the literature.

Keywords: motion control, physics-based animation, data-driven animation, equations of motion, inverse dynamics.

ÖZET

VERİMLİ BİR TERS DİNAMİK YÖNTEMİ İLE GERÇEK ZAMANLI FİZİKSEL HAREKET KONTROLÜ

Ersan Kavafoğlu

Yüksek Lisans, Bilgisayar Grafiği Bölümü

Tez Danışmanı: Yrd. Doç. Dr. Serdar Arıtan

Ocak 2018, 74 sayfa

Günümüzde hareket yakalama teknikleri kullanılarak oldukça gerçekçi karakter animasyonları elde edilebilmektedir. Fakat yakalanan bu hareketler yalnızca önceden belirlenmiş senaryolar için kaydedilebilmektedir. Sanal bir karakter, içinde bulunduğu sanal dünya ile sonsuz çeşitlilikte fiziksel etkileşime girebilir ve tüm bu etkileşimlerin önceden tahmin edilip, uygun hareketlerin kaydedilmesi mümkün değildir. Gerçek hayatta ise insanlar içinde bulundukları ortamla etkileşime girdiklerinde, hareketleri fizik kanunlarının bir sonucu olarak oluşmaktadır. Fizik tabanlı animasyon çalışmalarında, sanal ortamlardaki karakterlerin tepki hareketlerinin de tıpkı gerçek hayattaki gibi fizik kanunlarına göre doğal bir biçimde oluşması hedeflenmektedir.

Animasyon uygulamalarında doğal hareketlerin elde edilebilmesi için, basitleştirilmiş bir fiziksel modelin, yalnızca eklem torkları uygulanarak hareket ettirilmesi gerekmektedir. Bu eklem torklarının hesaplanabilmesi için robotik, biyomekanik ve fizik gibi alanlarda yapılan çalışmalardan faydalanılmaktadır. Gerçek zamanlı uygulamalarda kullanılabilmesi için bu hesaplamaların çok yüksek serbestlik derecesine sahip çoklu vücut sistemleri için bile çok düşük işlem maliyeti ile gerçekleştirilebilmesi gerekmektedir.

Literatürde, fizik tabanlı animasyon alanında yapılan çalışmalardaki kontrolcüler, lokal ve hareket denklemi tabanlı olmak üzere iki ana başlık altında toplanabilir. Lokal kontrolcülerden en yaygın kullanılanı, entegrasyon ve problemi modelleme basitliği sebebiyle tercih edilen Oransal Türev (Proportional Derivative) kontolcüdür. Lokal kontrolcüler genellikle

her harekete ve karaktere göre el ile ayarlanması gereken parametreler içerirler ve gerçek zamanlı uygulamalarda kullanılabilecek hızlarda tutarlı değildirler.

Hareket denklemi tabanlı yöntemlerde ise problemi modellemek ve entegrasyon çok daha karmaşıktır. Ancak hareket denklemi tabanlı yöntemler lokal yöntemlerden çok daha kaliteli ve tutarlı sonuçlar verir. Hareket tabanlı yöntemlerin temel bileşeni ters dinamik yöntemleridir. Fizik tabanlı animasyon alanında bugüne kadar ters dinamik için Newton-Euler ve Euler-Lagrange yöntemleri kullanılmıştır. Newton-Euler yöntemi daha çok iteratif olarak torkların hesaplanması için kullanılırken, Euler-Lagrange yöntemi ise genellikle optimizasyon problemlerinde ihtiyaç duyulan analitik hareket denklemlerinin elde edilmesi için kullanılmıştır.

Biz bu tez çalışmasında Kane yöntemini kullanarak 3 boyutlu uzayda, oryantasyonu quaternionlar ile belirtilen, 3 serbestlik derecesine sahip dönel eklemlerden oluşan çoklu vücut sistemleri için genelleşmiş hareket denklemleri elde ettik. Bu hareket denklemlerini kullanarak, karmaşık çoklu vücut sistemleri için ters dinamik çözümünün, elle veya sembolik programlama yardımıyla analitik olarak hesaplanabilmesinin uygulanabilir olmadığını gözlemledik. Gerçek zamanlı uygulamalarda kullanılabilir olması için bu denklemlerden yola çıkarak, fiziksel animasyon uygulamalarında sıkça kullanılan özyineli Newton-Euler algoritmasına eşdeğer özyineli bir ters dinamik algoritmasına ulaştık. Diğer çalışmalardan farklı olarak, bu algortimaya analitik denklemlerden ulaştığımız için, hem hareket planlama ve hareket oluşturma hem de ters dinamik için kullanılabilecek bütünleşik bir yaklaşım ortaya koymuş olduk. Yöntemimizn sonuçlarını değişik senaryolarda test ettik ve tüm senaryolar için yöntemimizin yüksek zaman adımlarında bile tutarlı sonuçlar ürettiğini gözlemledik. Aynı zamanda yöntemimizi literatürde sıkça kullanılan diğer yöntemlerle de karşılaştırdık.

Anahtar kelimeler: hareket kontrolü, fizik tabanlı animasyon, veri güdümlü animasyon, hareket denklemleri, ters dinamikler.

ACKNOWLEDGEMENT

First, I would like to express my sincere gratitude to my supervisor Assist. Prof. Dr. Serdar Arıtan for providing valuable guidance throughout my thesis research. He made me gain a better perspective of research and science.

I would also like to thank Prof. Dr. Haşmet Gürçay, Prof. Dr. Fatih Yaşar, Assist. Prof. Dr. Mehmet Serdar Güzel and Assist. Prof. Dr. Arif Mithat Amca for their valuable feedbacks on my research.

I'm grateful to my dear parents Sevil Kavafoğlu and Musa Kavafoğlu, for their continuous support during this intensive period, as they've always been throughout my life.

The last thanks are to the most precious, to my beloved wife Zümra, who made me feel her love and support in every single moment.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Motivation and Scope of the Work

Motion of articulated models, like humanoid characters, are widely used in computer animation applications. Nearly all necessary motions can be generated by using motion capture technology. Although these motions seem realistic enough, their use is limited to only some specific scenarios. However, especially real time interactive applications, like computer games, include lots of unplanned interactions that cannot be constrained to a specific scenario. For example, in a computer game, when a random ball hits a character in an unexpected way, the response animation of the character depends on lots of parameters: the weight of the ball, its movement direction, where it hit the character, the character's pose and momentum at the impact moment and etc. Moreover, the properties of the virtual environment, like the slipperiness, smoothness or motion of the ground, also effect the response of the character. Likewise, endless amount of interactions may occur, which are unplanned and specific to each application. It's impossible to pre-generate kinematic motion data for all of these different conditions. Instead of this, researchers aim to automatically achieve realistic interactions by constructing the virtual environment and the character in accordance with the physics laws as in real life. All of the techniques developed with this aim constitute the physics based animation research area.

In commercial applications, forward dynamics simulation and collision handling of rigid bodies connected with joint constraints are successfully applied, by using forward physics simulation libraries, also known as physics engines. However, as in robotics, physics-based animation of human motion is a multibody system motion control problem, which is based on calculating harmonious joint torques that generate the desired full body motion. Joint torque calculation is a highly complicated problem due to the complex structure of the character model and its high degrees of freedom. In addition to that, since the target applications need to work in real-time, this complicated process should be time-efficient. In real-time applications, at least sixty frames per second should be displayed in order to obtain a smooth view, which is equivalent to nearly 16 milliseconds per frame. But the process need to be handled at each frame is not limited to the calculation of joint torques, at each frame process power is consumed for lots of other components of the system like rendering, artificial intelligence calculations, sound, input system, logic etc. and also the inner process of the physics

engines like geometric collision detection and performing dynamic calculations. Moreover, the applications usually include more than one character, which should be animated simultaneously. Considering the process cost of all these components, joint torque calculations should be completed within a time nearly less than 1 or 2 milliseconds.

A joint torque calculation method should be able to produce stable results with large time-steps in order to obey the time constraint mentioned above. The process cost of a method increases as it's processed at small time-steps. This is why a method does not have the chance of being used in real-time commercial applications, when it works with small time-steps, approximately less than 0.01. On the other hand, running the method with large time-steps reduces the stability of the results due to the incorrect floating point calculations and integrator errors of the physics engines. Since the integrators of the physics engines perform not continuous but discrete calculations, as the time-steps are increased the errors originating from the physics engines increase [2]. Moreover, when the errors made in joint torque calculations are added, the system cannot produce stable results. Therefore, calculating accurate joint torques as much as possible is quite important for achieving stable results in large time-steps. In other words, torque calculation method of a controller is one of the most decisive factors for being able to use it in real-time applications.

Joint space controller methods, mostly Proportional Derivative Controller(PD Controller), are frequently used in physics based animation research. They are easy to implement and model but on the other hand they have disadvantages in terms of stability and locality. In order to obtain stable results with joint-space controllers, simulation time step needs to be decreased, which makes the controller less efficient. Moreover, because of their local nature, these methods don't take into account the overall system, the effect of body-parts on each other or the external effects like gravity. Usually additional methods are employed in joint-space controller based systems for compensating their local nature. Figure 1.1 shows the system diagram of a typical joint-space controller based method. As seen in the figure, the system is formed of two main components, one for generating the kinematic target and the other one for physics based simulation. Kinematic target generation component includes a reference motion, a higher level controller and an inverse kinematics solver, which together generate the desired joint orientations and velocities θ_d and $\dot{\theta}_d$. The simulation part is a closed loop system, which includes a local feedback controller, usually PD controller, that calculates joint torques by using the desired joint state and current joint state. It also includes

an additional torque generation component that usually includes some techniques for compensating the locality disadvantages of local controller. The total torque calculated by these two components and the external torques and forces are fed to the physics simulator, which outputs the current state of the joints after applying these forces and torques. With such kind of a system structure, good results can be obtained at smaller simulation time steps. But local controller based systems cannot produce stable and accurate results at large simulation time steps unlike methods based on equations of motion.

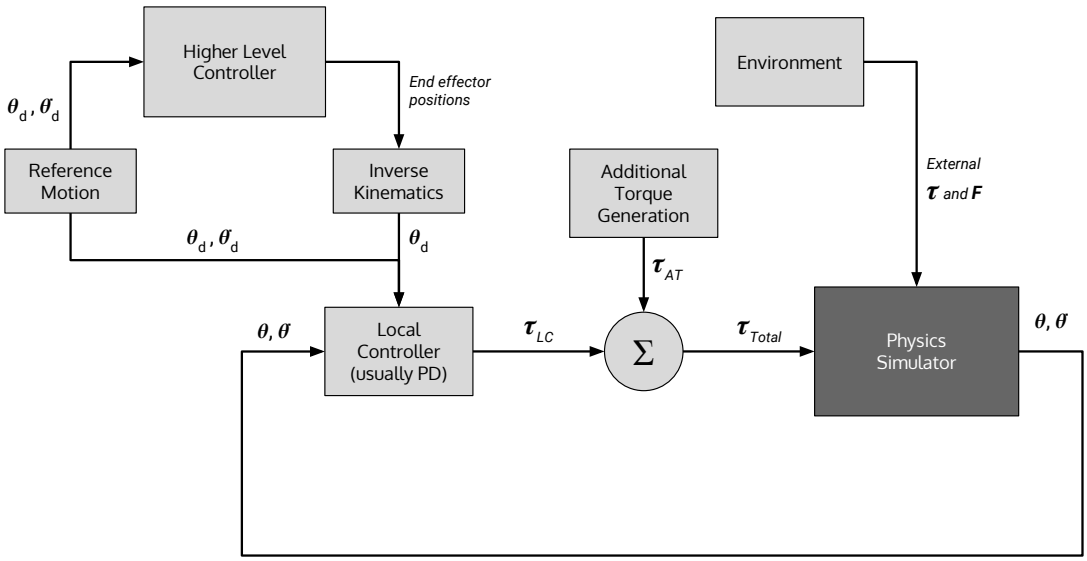


Figure 1.1: An overview of typical joint-space controller based system

Another fundamental approach for joint torque calculation is to build a recursive algorithm based on the Newton-Euler equations of motion. The implementation of these algorithms is not as simple as the local controllers, but as opposed to local controllers they offer a joint torque calculation technique which considers the whole multi-body system. Hence, these methods calculate accurate joint torques with no need of additional compensating components. The main ingredient of these methods is to calculate the appropriate joint torques by using the local acceleration information of the joints and taking into account the whole system structure and the external effects on the body parts. These techniques have the ability of creating stable results with large simulation time-steps.

In this thesis study, we aimed to achieve a simpler and generic analytic inverse dynamics solution by using Kane's method [3], which can be employed in both control and planning of motions in physics based character animation. Although, we can solve analytic equations

for systems with small degrees of freedom, as the system's degrees of freedom increase we observed that it's not applicable to solve the equations neither manually nor by using symbolic calculation libraries due to the long and complex operations. Therefore, after a step, we formed the solution as a recursive algorithm. This recursive algorithm we obtained happened to be identical to the current recursive Newton-Euler algorithms. However, as opposed to other approaches in the literature, we obtained the algorithm from an analytical equation. This equation is also feasible for planning and generation of motions, which cannot be handled by other Newton-Euler based techniques. In addition to these, since our method calculates highly accurate joint torques, its results are stable at large simulation time-steps, which makes it quite suitable for using in real-time applications.

We mainly focus on multibody systems composed of rotational joints with 3 degrees of freedom and rigid bodies whose configurations are represented by quaternions. To achieve generalized equations of motions for this kind of systems, we first make appropriate motion variable choices under the guidance of [4] to obtain the resulting equations in a simple form. By using the patterns that occur for our multibody system and the algebraic properties of vectors, we simplify the equations as much as possible and by using these equations we form a recursive algorithm. We use this algorithm for our inverse dynamics calculations. Our inverse dynamics algorithm takes desired joint accelerations and predictable external forces and torques as input. With the word predictable, we denote the forces and torques that are known in advance to effect on the system, like gravity, ground reaction forces or the possible impulses of predicted collisions. We use a critically damped PD controller for calculating the input desired accelerations. Figure 1.2 shows the general overview of our proposed system. In this study we only use reference motions for generating desired joint orientations and velocities, but other high level controllers can also be integrated into our system easily as in other typical systems (see Figure 1.1). We evaluated the results of our controller both in 2D and 3D simulation scenarios. We compared our results with PD Controller, which is one of the most widely used controllers in the literature. We also made comparisons with different variants of PD controller, like stable PD controller, PD controller with feedforward term and PD controller with torque bubble-upping and gravity compensation. We performed all of these comparisons under various configurations and scenarios.

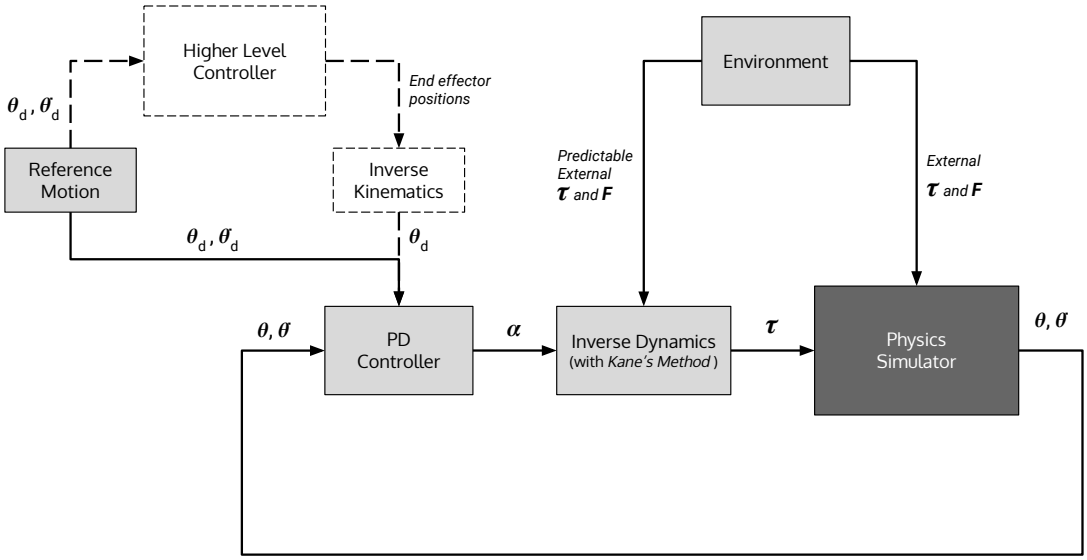


Figure 1.2: Overview of our system

1.2 Thesis Outline

The thesis consists of three more sections which are briefly explained below.

Section 2 includes the general explanation of the fundamental techniques and literature overview that form a background for the thesis content. In this section, first, the need for physics based motion control in animation applications and the fundamentals of physics based animation techniques are described. After that, joint-space control methods are explained in detail, focusing on the widely used Proportional Derivative Control method. The technical details of Proportional Derivative Control method, its advantages and disadvantages and the literature review about it are presented in this section. In addition to these, different techniques that modify and advance Proportional Derivative Control method are also explained in this section. This section also covers the main techniques for obtaining equations of motion which are used to build inverse dynamics algorithms. The covered techniques include Newton-Euler method, Euler-Lagrange method and Kane's method.

In Section 3.1 to 3.6 we explained the rules,concepts and steps in detail that are necessary to obtain the equations of motion by using Kane's Method. In Section 3.7 we obtained equations of motion for an arbitrary multibody system consisting of rotational joints with 3 degrees of freedom by using Kane's Method and solved inverse dynamics of the system by using these equations. In Section 3.8 we built generalized equations by benefiting from the

patterns that we observed in Section 3.7 and simplified the equations further to enable real-time usage. We built a recursive algorithm identical to simplest forms of recursive Newton-Euler algorithm by using the simplified equations. In Section 3.9 we gave details about our implementation and compared our proposed system with various types of PD controller in different scenarios.

In the last section, the conclusion of the thesis is presented which discusses the limitations of the proposed techniques together with the future research ideas that can arise from the thesis.

2 BACKGROUND

This chapter summarizes the general concepts and the technical background for the thesis. The first part gives a brief overview of physics based motion control. The next part introduces the state of the art control methods used in computer graphics. And the last part explains the methods used to calculate inverse dynamics for a multibody system.

2.1 Physics Based Motion Control

Computer graphics industry and researchers always pursue realism to improve believability and the feel of immersion into the virtual world. The two main aspects which determine the realism perception in applications are graphics and animations. Although the graphics techniques advanced a lot through the years to reach the desired realism, animation techniques improved a little for the last 20 years compared to them. Is it because the animations are perfect enough? The answer is no in most of the cases.

Generally speaking, the animations look great until there occurs a simple interaction with the surrounding virtual environment. Currently, for any expected interaction, recording the necessary animation with motion capture technique is the mainly used solution in industry. However, since the virtual environments are dynamic, most of the interactions happen in an unexpected way and lack of realistic responses to them prevent the user's feel of immersion. Recording many possible interaction scenarios and creating a huge database using those clips need a great amount of human power and in return this enormous effort makes the results just a bit better.

For a real human, the responsive motions occur naturally as a result of physics laws. The main idea behind the physics based animation techniques is to let a real-time physics engine to generate the natural response motions using the same physics laws as in real world. Although the responsive motions would occur naturally for a single rigid object by using physics engine, controlling a multibody system of rigid objects connected to each other with joints is not an easy task.

Real-time physics engines are frameworks that simulate motions of objects using Newton's law of motions. In physics engines, the objects are defined as rigid bodies and constrained to each other by using joints. A rigid body represents a non-deformable rigid object in a world

created by a physics engine. At each instant, each rigid body has a state consisting of its position, orientation, velocity and angular velocity. The physics engine simulates the motion of rigid bodies simply by taking their current state as input and calculating their next state after a small timestep Δt also by taking the possible collisions into account.

To enable using physics based animation techniques, we have to control rigid bodies. The only way to control rigid bodies correctly is to send forces and torques to the physics engine which have to be applied on each rigid body at each timestep. But calculating the correct forces and torques that will drive the rigid bodies from their current state to a desired state is challenging since we have to take gravity, joint reaction forces, external forces and inertia into account. Also to be usable in realtime applications, all of those calculations have to be made at least more than 30 times a second.

2.2 Joint-Space Control Methods

One of the most preferred control methods in physics based character animation is joint-space control, which aims to handle the control problem in local space of each joint of the articulated character. Generally, these methods take target state of each joint and some controller parameters as input and the output of these controllers are joint torques, which dissipate the difference between current states and target states when applied to the joints(see Figure 2.1). Here the term *joint state* stands for the joint orientation and angular velocity.

The main advantage of joint-space motion control techniques is their ability to describe the control problem from a clear point of view, by just defining some kinematic targets for the joints. This makes the design and implementation of these controllers very simple. However, in exchange with this simplicity, the necessity of coordinated movement of the joints is ignored, which retains these controllers from being sufficient for achieving high-level tasks like keeping balance. Moreover, this kind of methods don't take into account the external forces and torques acting on the body-parts. Considering the fact that, at least there is gravity acting on each body-part, we can conclude that these controllers alone are not sufficient even in simplest scenarios. Therefore, these methods should definitely be supported by some other techniques, which would differ specific to the control tasks.

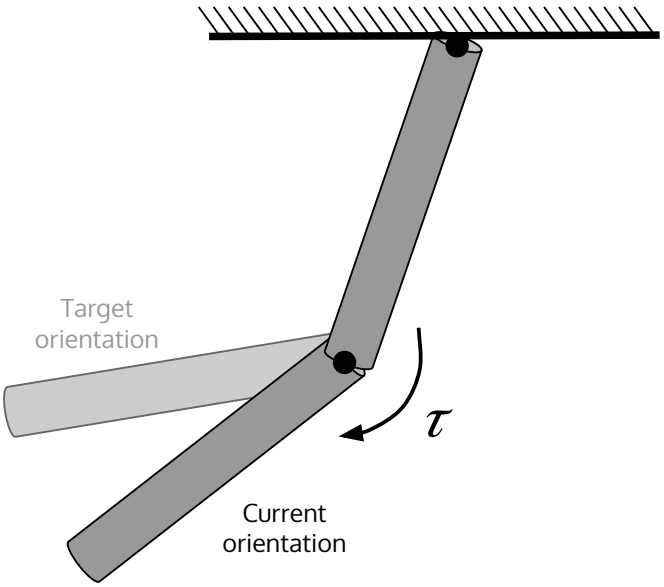


Figure 2.1: Target and current joint orientations and the output joint torque τ

2.2.1 Proportional Derivative Control

Proportional Derivative(PD) Control is by far the most commonly used joint-space motion control method in physics-based character animation. It originates from the Proportional Integral Derivative(PID) controllers of the control theory field, which is generally formulated as below.

$$u(t) = k_p e(t) + k_d \frac{de}{dt} + k_i \int_0^t e(x)dx \tag{2.1}$$

Here $u(t)$ is the control signal and $e(t)$ is the error between the reference and the current values at time t. k_p, k_d and k_i are the parameters of the controller, which are called the proportional gain, derivative gain and integral gain, respectively. Here, the output of the controller is the sum of three terms, which can be interpreted as follows: the proportional term $k_p e(t)$ is a multiple of the current error, the derivative term $k_d \frac{de}{dt}$ is a multiple of the predicted future errors and the integral term $k_i \int_0^t e(x)dx$ is a multiple of the sum of the past errors.

In physics based character animation, the integral term is excluded, since the reference value is more likely to change at every time-step, which makes the feedback from the sum of the past errors useless. In its most general form, the control signal $u(t)$ is regarded as the joint

torque τ, the error term is taken as the difference between target and current joint orientations and the derivative of the error term is taken as the difference between target and current angular velocities of the joint. Therefore, the formulation of PD controller for physics-based character animation is as follows

$$\tau = k_p(q_r - q_c) + k_d(\dot{q_r} - \dot{q_c}) \tag{2.2}$$

Here q_r and q_c are target and current joint orientations and $\dot{q_r}$ and $\dot{q_c}$ are target and current joint velocities, respectively. This formulation makes the calculation of joint torques completely from the local error terms. Therefore, in this form we can call the PD controller a local feedback controller.

Because of its simplicity, PD controller is widely used in computer animation and robotics fields [5, 6, 7, 1]. The very early employment of PD controller in motion control of visual characters is by Hodgins et al.[5]. In their work, joint torques are calculated by PD controller for the animation of running, bicycling and vaulting.

Addition of feedforward control In tracking controller problems, reference motion can be benefited not only to measure the error term but also to give a clue about the desired acceleration. Several works in the literature, use this clue from the reference motion, by including a feedforward component to their tracking controllers.

In several works[8, 9, 10], desired acceleration for each body-part is calculated with a modified version of the PD-Controller formula, which includes a feed-forward acceleration term. (see Equation 2.3)

$$\ddot{q_d} = k_p(q_r - q_c) + k_d(\dot{q_r} - \dot{q_c}) + \ddot{q_r} \tag{2.3}$$

Here $\ddot{q_r}$ is an acceleration feedforward term which can be calculated from the reference motion by finite differences or inverse dynamics. In these works, tracking control is handled as a multi-objective optimization problem. This optimization problem includes separate objectives for tracking and balance control. Tracking objective aims to minimize the difference between desired and current joint accelerations, where desired accelerations include a feedforward term as mentioned above.

There are also other approaches for combining feedforward and feedback controllers for tracking purposes. One of the very early examples of them is by Yin et al.[11]. They first estimate feedforward torques from reference motion capture data with an inverse dynamics preprocessing. And then they calculate feedback torques for compensating small drifts and disturbances with a formula similar to Equation 2.2. Yin et al.[12] also calculate joint torques with the sum of PD Controller and feedforward torques, but this time they make use of feedback error learning [13, 14] for calculating feedforward torques. da Silva et al. [15] propose a combined controller which also calculates the control signal with the sum of two controller components. As in [11], one of these components is a PD controller that provides a feedback control for responding to external disturbances. The other controller component is the predictive control component, which aims to reproduce the joint accelerations of the reference motion in a short period of time. This predictive component defines a quadratic optimization problem that solves for the joint torques and contact forces, which minimize the difference between reference and current accelerations. Kwon and Hodgins [16] also compute feedforward torques with a hybrid dynamics solver in addition to PD Controller for tracking joint configurations of a reference human running motion. Lee et al.[17] also compute desired joint accelerations with the feedforward term added PD controller formula(2.3) and calculate the joint torques from these accelerations with inverse dynamics.

Gain Tuning The main aim of these approaches, which include a feedforward term in their torque or desired acceleration formulation is to achieve a lower-gain tracking. High-gain tracking, which means assigning high values to PD Controller gains k_p and k_d is undesirable because it leads to stiff motions which cannot respond to external perturbations in a realistic way. On the other side, low-gain tracking enables more realistic interactions with the environment, but it falls short to track a reference motion with exact timing. To obtain a compromise between realistic responses and accurate tracking, Zordan et al. [7] lower the gains of their PD controller for a short period of time after encountering an external disturbance. In a similar way, Yin et al. [12] offer the option of increasing the gains some time after responding to impacts for obtaining a natural recovery behavior. Although these approaches seem to offer a straightforward solution of choosing gains compatible to unpredicted changes, they raise lots of questions like what is the best period of time to have the most natural responses, how much to increase or decrease parameters etc.

As apparent from its formulation(2.2), the values of the proportional gain k_p and derivative gain k_d have the role of completely changing the amount of joint torques to be applied. These gains, the so-called controller parameters, specify the stiffness, oscillation and timing of the resulting motion. As described in the previous paragraph, the stiffness and timing of the resulting motion change as the gains are increased or decreased simultaneously. On the other hand, the oscillation of the resulting motion depends on the ratio of the gains. To be more specific, if k_d/k_p ratio is too low then the resulting motion will be an oscillatory one with frequent overshootings. In this case, the controller is called under-damped. On the contrary, if k_d/k_p ratio is too high, then the controller is called over damped, cause the output motion cannot reach the specified target. The aim of gain tuning is to find the critically-damped controller providing an output motion that reaches the target motion without oscillating around it. In industry and robotics, the critically damped gain ratio is found to be $k_d = 2\sqrt{k_p}$.

One of the main disadvantages of PD Controller is the need for gain tuning. Although most of the approaches in the literature prefer to manually tune the PD Controller gains, there are several works which address the inconvenience of manual gain tuning and propose different approaches to sort it out. Wang et al. [18] optimize the PD gains and target DOF angles of [12] with Covariance Matrix Adaptation[19] optimization in order to obtain a physics based motion with the user-defined style. The main disadvantage of this work is the need of an expensive optimization process for each task. Allen et al. [20] propose an analytical solution for calculating critically-damped PD Controller gains. Zordan and Hodgins [7] scale the gains of the controller proportional to the moment of inertia of the body chain affected by the joint. A similar approach by Coros et al. [1] scales the gains for each bodypart according to their mass but they still need to tune the gains of one reference body part.

2.2.2 Stable Proportional Derivative Control

As we mentioned in "Gain Tuning" subsection, high-gain tracking is undesirable because it causes stiff and unnatural behaviors. However, employing high gains is needed for tracking, if the reference motion includes quick deviations. In the case of large simulation timesteps, PD Controller formula with high gains can result in enormously large joint torques, which destroy the stability of the control. Therefore, in order to achieve a stable high-gain control, simulation timestep should be reduced. But reducing simulation time-step means loosing

simulation efficiency. Tan et al. [21] define this situation as *the undesirable coupling of tracking accuracy and simulation efficiency* and they propose a new controller called Stable Proportional Derivative Controller to avoid this coupling.

To explain SPD controller, we will re-write the PD Controller equation(eq 2.2) by adding time-step information:

$$\tau^n = k_p({q_r}^n - q^n) + k_d({\dot{q_r}}^n - \dot{q}^n) \tag{2.4}$$

Here superscript n denotes n^{th} time step and accordingly joint orientation, joint velocity, reference joint orientation and reference joint velocity at time-step n are denoted by q^n, $\dot{q}^n$, ${q_r}^n$ and ${\dot{q_r}}^n$ respectively. Therefore, in PD Controller equation, joint torque at a specific time step is calculated by using information from the same time-step. Unlike PD Controller, SPD Controller calculates joint torques at a specific timestep by using information from the next time. SPD Controller equation is as follows:

$$\tau^n = k_p({q_r}^{n+1} - q^{n+1}) + k_d({\dot{q_r}}^{n+1} - \dot{q}^{n+1}) \tag{2.5}$$

In this equation $n + 1$ denotes the next time-step after n. The reference orientation and reference velocity at the next-time step can be calculated from the reference motion. But the joint orientation and velocity at the next time step cannot be known before the simulation reaches this time-step. Therefore, they approximate the orientation and velocity values with the equations below:

$$q^{n+1} = q^n + \Delta t \dot{q}^n \tag{2.6}$$

$$\dot{q}^{n+1} = \dot{q}^n + \Delta t \ddot{q}^n \tag{2.7}$$

Here Δt denotes the fixed simulation time-step.

By substituting equations 2.6 and 2.7 in equation 2.5, we obtain the equation below:

$$\tau^n = k_p({q_r}^{n+1} - q^n - \Delta t \dot{q}^n) + k_d({\dot{q_r}}^{n+1} - \dot{q}^n - \Delta t \ddot{q}^n) \tag{2.8}$$

Tan et al. [21] shows that with this equation they achieve a stable high-gain tracking at quite low simulation time-steps. Moreover they claim that their proposed method eliminates

the problem of gain tuning by enabling tuning the same gain for all joints without stability problems.

2.2.3 Proportional Derivative Control with torque bubble-upping and gravity compensation

As we mentioned in the previous sections, joint-space motion controller methods provide local control, in other words, they calculate a joint torque for each bodypart of the system without taking into account their mutual effect on each other or the effect of gravity on them. However, because of their simplicity and efficiency, joint space control methods, especially proportional derivative control, are widely used for motion simulation, together with some additional techniques for compensating the disadvantages due to their local nature.

In this section we'll mention two techniques used in [1]: torque bubble-upping and gravity compensation.

Torque bubble-upping As a natural result of joint articulation, calculated joint torques effect both of the bodyparts connected by that joint. Although, in physics simulations there isn't any hierarchical relation between rigid bodies, we always represent our articulated system as a hierarchical structure in order to solve problems in an organized way. For example, let τ_i be the calculated joint torque for joint J_i, which connects bodyparts B_{i-1} and B_i, where B_{i-1} is parent of B_i. Then τ_i is applied on B_i and $-\tau_i$ is applied on B_{i-1}. When we consider the joint torque τ_{i-1} calculated for joint J_{i-1}, the total torque on bodypart B_{i-1} becomes $\tau_{i-1} - \tau_i$. However, the resulting torque on body part B_{i-1} is expected to be only τ_{i-1}. In order to tackle this problem, Coros et al. apply a technique called torque bubble-upping. Simply adding τ_i to the joint torque of J_i easily compensates the undesired effect generated by $-\tau_i$. When the same logic is applied through the bodypart chains in the system, those compensating torques bubble up till the root of the chains (see Figure 2.2).

Gravity Compensation In a physics-based simulation scene, each physically defined object is affected by gravity. Joint-space controller techniques make all their calculations according to the desired and current state of the joint without taking into account the effect of gravity. Therefore, body-parts cannot reach the desired orientation, unless the controller

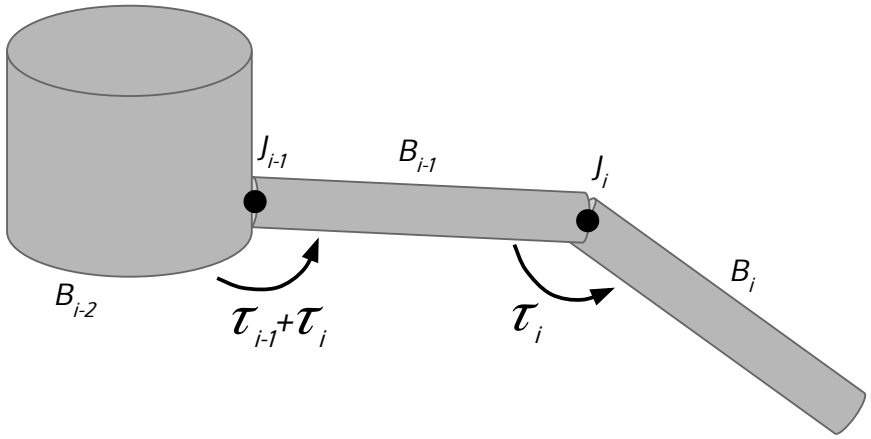


Figure 2.2: Calculating joint torques for compensating gravity effect on leaf bodypart

gains are too high. Gravity compensation is based on the simple strategy of applying additional joint torques against the effect of gravity on the body part. In order to calculate these additional joint torques, Jacobian transpose method is used. Jacobian transpose method calculates joint torques that mimic the effect of applying a linear force on a body part.

Let's examine gravity compensation with Jacobian transpose method on a three link system as shown in Figure 2.3. To compensate the effect of gravity on the leaf body-part B_1 we can imagine a virtual force G_1 which is the opposite of the gravity force $m_1 g$ acting on B_1, where m_1 is the mass of B_1 and g is the gravitational constant. This force is called a virtual force, because it's not applied on the body-part, but instead joint torques that mimic the effect of applying it on the bodypart are calculated along the chain between the bodypart and the root. Therefore, for our example $G_1 = -m_1 g$ and we will calculate joint torques τ_1 and τ_2 with the equations below:

$$\tau_1 = (P_1 - P) \times G_1 \tag{2.9}$$

$$\tau_2 = (P_2 - P) \times G_1 \tag{2.10}$$

Here P is the point where the gravity is applied, which is the center of mass of B_1 and P_1 and P_2 are the corresponding joint positions. As understood from the equations, each joint torque is calculated as the cross product of the virtual force vector and the difference vector between joint position and force application position.

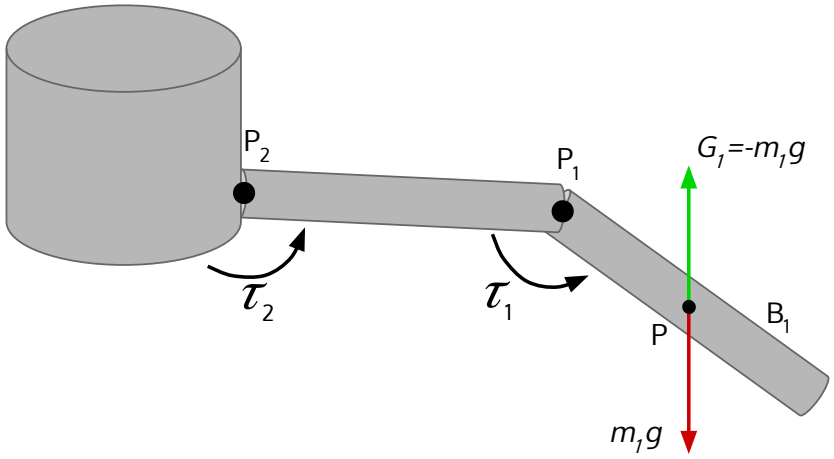


Figure 2.3: Calculating joint torques for compensating gravity effect on leaf bodypart

2.3 Multibody Dynamic Modeling

Calculating the correct amount of forces and torques that generate the desired accelerations for each rigid body in a multibody system is an inverse dynamics problem. Physics based animation isn't the first research field that needs to solve inverse dynamics. The same problem has been worked by researchers on fields like robotics and biomechanics for decades and hence there are plenty of methods to solve inverse dynamics originating from these fields. But, in order to be usable in physics based animation applications, an inverse dynamics method has to be able to solve for more than one multibody system, each of which is formed of more than 15 rigid bodies and 40 degrees of freedom. Also to be usable in real-time applications, the method has to be capable of performing these calculations more than 60 times in a second. Considering these conditions, we aim to build a highly efficient inverse dynamics algorithm suitable for real-time motion tracking in physics based animation applications.

Motion generation and motion planning are also very important components of physics based animation applications. Generating a motion that satisfies complexly dependent objectives and constraints is also equally important as tracking that motion. Generally, those objectives and constraints are combined into an optimization problem along with the equations of motion in analytic form. Moreover, motion planning is also critical to generate realistic and natural motions in a virtual environment. To plan motions in realtime, analytic equations of motion are again useful since they allow querying future configurations of a multibody system under known or estimated external forces.

To build an inverse dynamics algorithm, first we have to obtain the equations of motion for any given multibody system. In this sub section we'll briefly cover the main methods in physics to build the equations of motion for multibody systems [22].

2.3.1 Newton-Euler Method

Newton-Euler method is the mainly used method for obtaining numerical solutions for multibody systems. The core of the method completely relies on $F = ma$ and $\tau = I\alpha$ equations, which point where the name of the method comes from. In this method, Newton-Euler equations are written for each body in the kinematic chain of a multibody system and applied iteratively. The forces and torques acting on each body have to be taken into account including constraint forces. Although it's easy to model simple multibody systems with Newton-Euler method, for a human model consisting of nearly 15 rigid bodies and 41 degrees of freedom, calculating those forces and torques iteratively at each timestep is complex and computationally inefficient.

Many researchers worked on building an efficient recursive algorithm to tackle this issue using Newton-Euler formulation [23]. Luh, Walker and Paul developed an efficient recursive Newton-Euler algorithm with $O(n)$ complexity [24]. Using this algorithm, inverse dynamics for a multibody system can be calculated in real-time. Although recursive Newton-Euler algorithm works fairly good for controlling multibody systems, the solution is numeric and lacks the access to analytic equations of motion. That makes recursive Newton-Euler algorithm not suitable for motion generation and planning applications.

2.3.2 Euler-Lagrange Method

Euler-Lagrange equations depend on finding the total energy change of the rigid bodies in a multibody system using the methodologies of Lagrange, D'Alembert and Euler [25]. The main difference between Newton-Euler and Euler-Lagrange equations is in dealing with the constraints. In Newton-Euler equations, the forces required to enforce joint constraints have to be calculated for each joint since each rigid body in the multibody system is handled separately. As opposed to that, in Euler-Lagrange equations, generalized coordinates are used to parametrize the system therefore the constraint forces are eliminated from the dynamic equations and the resulting system of equations become simpler. Moreover, Newton-Euler method is an iterative approach chasing a numerical result while Euler-Lagrange method is

a technique to obtain the equations of motion in analytic form. Since the solution is analytic, the calculations can be made at any desired time with no extra performance cost. That also makes the Euler-Lagrange method perfectly suitable for any motion planning problem.

Although it seems like Euler-Lagrange method satisfies everything we need for both motion tracking and motion generation, there are some serious drawbacks that prevent researchers from using that method. To derive equations of motions with Euler-Lagrange method, *Lagrangian* of the multibody system has to be symbolically differentiated [26]. *Lagrangian* is expressed in terms of generalized coordinates and their derivatives. But as in our problem for controlling a multibody system composed of joints with 3 rotational degrees of freedom, orientations of the bodies have to be represented by quaternions. Therefore generalized coordinates should be taken in terms of quaternions. Building and symbolically differentiating the *Lagrangian* equations become more complex due to the quaternion terms. Also for each different multibody system, those symbolic calculations have to be done manually or using symbolic software libraries which don't guarantee a solution and very slow for large systems of equations.

2.3.3 Kane's Method

Thomas Kane proposed a new method to obtain motion equations of multibody systems in analytic form [27]. Later on, Kane and Wang published a complementing paper that explained the use of motion variables (later called *generalized speeds*) [3]. These two papers formed the core of a new technique for obtaining equations of motion called *Kane's Method*, also known as *Lagrange's form of D'Alembert's principle*. As opposed to Euler-Lagrange method, there's no need to differentiate kinetic and potential energy functions or calculate Lagrange multipliers [25]. Moreover, since motion variables can be selected as any combination of the time derivatives of generalized coordinates as explained in [3], the complexity of the resulting equations can be reduced dramatically by selecting proper motion variables. Mitiguy and Kane published a paper that completely focuses on the influence of the choice of motion variables over the complexity of the resulting equations [4].

Kane's Method has clear advantages over the existing methods especially for three dimensional multibody systems since it eliminates the need for differentiating energy functions. For proper choice of motion variables, only the differentiation of vectors are needed which

can be simplified to simple vector products. That makes Kane's method a vector based approach and suitable for software applications. We decided to work with Kane's Method since it fits well for both motion controlling and motion generation purposes. Our motion variable choice and the details of the method can be found in Section 3.

3 REAL-TIME PHYSICS BASED PRECISE MOTION CONTROL

This chapter presents the steps to obtain an analytic solution for the underlying dynamics of an arbitrary multibody system with rotational joints and to build an efficient generalized inverse dynamics algorithm by using this analytic solution. The process is mainly based on Kane's method for Equations of Motion.

3.1 Kinematics of Multibody Systems

First step of solving the dynamics of a multibody system analytically is to determine the kinematics of the system. Kinematics is basically the geometric definition of a system which defines the position, velocity and acceleration of the points or bodies of the system together with their angular counterparts. The equations that describe the mathematical relations between these kinematic properties are called the kinematic equations. Since Kane's Method is a vector based approach, we'll write the kinematics equations of the system in vectorized and dyadic equation forms. Before building the kinematic equations of a multibody system, we have to determine the reference frames and critical points of the system by taking into account the particles, bodies and constraints involved in the system.

The critical points of a multibody system are basically the points where forces may be applied. For example the center of mass points of the bodies of the system are critical points, since the gravity force mg is applied from the center of mass of each body. It's clear that there may be lots of other points on a multibody system where forces can be applied. But we should narrow down our choices to the most essential points for the sake of simplicity. These critical points would be different according to the problem and multibody system in hand.

Every rigid body in a multibody system has a reference frame fixed on it which is called the *local reference frame*. So the reference frames that we're interested in are the local reference frames of each rigid body and the world reference frame.

Below we describe some basic equations which are widely used for calculating kinematic equations of the system. In these equations B denotes a body which moves relative to a reference frame R.

Derivative of a vector on a body :

Let c be a non-zero vector fixed on body B. The time derivative of the vector c in reference frame R is defined as below where ${}^{R}\omega^{B}$ denotes the angular velocity of body B relative to reference frame R.

$$\frac{{}^{R}dc}{dt} = {}^{R}\omega^{B} \times c \tag{3.1}$$

Linear velocity of a point on a body :

Let P and Q be fixed points on the body B and r be a vector such that $r = P - Q$. If the linear velocity of point Q is known, then the linear velocity of point P can be obtained as follows

$$P = Q + r \tag{3.2}$$

by taking the derivatives of both sides in reference frame R we obtain

$${}^{R}v^{P} = {}^{R}v^{Q} + \frac{{}^{R}dr}{dt} \tag{3.3}$$

Here ${}^{R}v^{P}$ and ${}^{R}v^{Q}$ denote the linear velocities of point P and point Q respectively, relative to reference frame R. Since r is fixed on body B, by equation 3.1 we obtain the following.

$${}^{R}v^{P} = {}^{R}v^{Q} + {}^{R}\omega^{B} \times r \tag{3.4}$$

Linear acceleration of a point on a body:

By taking the derivative of the linear velocity equation 3.4 we obtain

$${}^{R}a^{P} = {}^{R}a^{Q} + \frac{{}^{R}d{}^{R}\omega^{B}}{dt} \times r + {}^{R}\omega^{B} \times ({}^{R}\omega^{B} \times r) \tag{3.5}$$

Here ${}^{R}a^{P}$ and ${}^{R}a^{Q}$ denote the linear accelerations of the points P and Q respectively, relative to the reference frame R. The angular acceleration(${}^{R}\alpha^{B}$) of body B relative to R is equal to the derivative of its angular velocity relative to R, as shown in the following

$$ {}^{R}\alpha^{B} = \frac{{}^{R}d^{R}\omega^{B}}{dt} \tag{3.6} $$

Therefore by substituting 3.6 in equation 3.5 we gather the resulting linear acceleration equation

$$ {}^{R}a^{P} = {}^{R}a^{Q} + {}^{R}\alpha^{B} \times r + {}^{R}\omega^{B} \times ({}^{R}\omega^{B} \times r) \tag{3.7} $$

3.2 Generalized Speed Concept

The parameters that uniquely describe the instantaneous configuration of a multibody system relative to some reference configuration are called *generalized coordinates* and usually denoted by q_i. These parameters are also known as *configuration variables* and they constitute the general way of representing the configurations of the multibody systems in analytical solution methods. These parameters can be selected as desired, provided that they describe every possible configuration of the multibody system in a unique way. Moreover, the choice of the parameters has a dramatic effect on the complexity of the resulting equations of motions. Therefore, the parameters which provide easier solutions should be preferred.

In the simplest terms, *generalized speeds* are the linear combinations of the time derivatives of *generalized coordinates*. *Generalized speeds* are also called *motion variables* since they describe the motion of the system. They are usually denoted by u_i. For most of the simple systems and scenarios, choosing *generalized speeds* as the time derivatives of the generalized coordinates ($u_i = \dot{q}_i$) is sufficient. However, for more complex systems the selection of the generalized speed parameters may not be that straightforward. This follows from the fact that the selection of the generalized speeds also has a profound effect on the complexity of the further calculations, just like in the case of *generalized coordinates*. To emphasize the importance of *generalized speed* parameters' selection, Mitiguy and Kane published an article which solely focuses on this topic [4]. The article also contains a section entitled

Guidelines for Choosing Generalized Speeds, which covers most of the common cases in robotics.

The general form of generalized speed parameters $u_r, r = 1, ..., n$ of a system with n generalized coordinates $q_i, i = 1...n$ is as follows

$$u_r \triangleq \sum_{s=1}^{n} Y_{rs}\dot{q}_s + Z_r \qquad r = (1, ..., n) \tag{3.8}$$

Here Y_{rs} and Z_r are functions of the generalized coordinates $q_1, ..., q_n$ and time. We are free to select *generalized speed* parameters in this form as long as the equations 3.8 can be solved uniquely for each $\dot{q_1}, ..., \dot{q_n}$.

Let S be a multibody system with *generalized speeds* $u_1, ..., u_n$, which are not independent of each other. This means that there is at least one soft constraint in the system which involves velocity terms in its equation and S is said to be subject to *motion constraints*. Such systems, which are subject to *motion constraints* are called *nonholonomic systems*. If all of the *generalized speeds* $u_1, ..., u_n$ are independent of each other, the system is called *holonomic*. Each equation that relates *generalized speeds* to each other is called a *nonholonomic constraint equation*.

3.3 Partial Velocities

Partial velocities are vector quantities which hold valuable information about the underlying kinematics of a multibody system. These quantities are closely related to *generalized speeds*. Geometrically each *partial velocity* term represents how much the velocity of a point gets effected from a change in each generalized speed term. *Partial velocities* have to be calculated for center of mass points of each rigid body and possible force application points on a system. *Partial velocities* are used in further stages of the method to calculate *generalized active forces* and *generalized inertial forces*.

From the equation 3.8, the velocity of a point P can be uniquely expressed as

$$v^P = \sum_{r=1}^{n} v_r^P u_r + v_t \tag{3.9}$$

where v_r^P $(r = 1, ..., n)$ and v_t are functions of $q_1, ..., q_n$ and time t. Here the vector term v_r^P is called the *r'th partial velocity of point P*.

Similarly each *partial angular velocity* term represents how much the angular velocity of a reference frame gets effected from a change in each generalized speed term. *Partial angular velocities* have to be calculated for each rigid body. The angular velocity of a reference frame B can be expressed uniquely as the angular analogous of equation 3.9 where ω_r^B $(r = 1, ..., n)$ and ω_t are again functions of $q_1, ..., q_n$ and time t.

$$\omega^B = \sum_{r=1}^{n} \omega_r^B u_r + \omega_t \tag{3.10}$$

Here the vector term ω_r^B is called the *r'th partial angular velocity of reference frame B*.

From equations 3.9, 3.10 the *partial velocity* and *partial angular velocity* terms can be simplified as follows

$$v_r^P = \frac{\partial^R v^P}{\partial u_r} \tag{3.11}$$

$$\omega_r^B = \frac{\partial^R \omega^B}{\partial u_r} \tag{3.12}$$

3.4 Kane's Equation

To get a better understanding of *Kane's equations* for multibody systems, it would be better to start with a single particle as explained in [25]. Let P be a particle in 3 dimensional reference frame R, F be the sum of all forces acting on P and F^* be the inertia force for P. Since P is a particle, the inertia force F^* is equal to $-ma$ where m is the mass and a is the acceleration of particle P. According to ***D'Alembert's principle***

$$F + F^* = 0 \tag{3.13}$$

Here F and F^* are 3 dimensional vector quantities and the equation 3.13 contains sufficient information about the motion of particle P. Let v be the velocity of P. When we dot product both sides of equation 3.13 with v, we get

$$v\cdot F + v\cdot F^* = 0 \tag{3.14}$$

Since the result of a dot product operation is scalar, the equation 3.14 doesn't contain the sufficient information about the motion of particle P. But the equation 3.14 has a major advantage over equation 3.13 if the motion of P is constrained. In equation 3.14 the constraint forces are eliminated automatically. The main idea behind *Kane's method* is to construct equations like 3.14 which contain both the sufficient information about the motion of particle P and also get constraint forces eliminated automatically. To achieve that, the equation 3.14 can be replaced with

$$v_r\cdot F + v_r\cdot F^* = 0 \qquad r = (1, ..., n) \tag{3.15}$$

where n is the number of degrees of freedom and v_r is the ***r'th partial velocity*** of P.
Now we can introduce new variables as

$$\mathcal{F}_r = v_r\cdot F \qquad , \qquad \mathcal{F}_r^* = v_r\cdot F^* \tag{3.16}$$

Then the equation 3.15 becomes

$$\mathcal{F}_r + \mathcal{F}_r^* = 0 \qquad r = (1, ..., n) \tag{3.17}$$

Here $\mathcal{F}_r$ and $\mathcal{F}_r^*$ are the *r'th generalized active force* and *r'th generalized inertia force* of P respectively (see section 3.5). The equation 3.17 is known as *Lagrange's form of D'Alembert's principle* or *Kane's equations*.

3.5 Generalized Forces

Generalized forces are the scalar quantities which describe the effects of forces and torques on the linear and angular velocities of the system. Since Kane's Method is based on D'Alembert's principle, we'll take into account two types of forces acting on a multibody system. These are active and inertial forces.

3.5.1 Generalized Active Forces

The term *active force* stands for external forces and torques acting on a multibody system. For example the gravity force or an actuation torque acting on a rigid body are *active forces*. The equation for *i'th generalized active force* can be written as

$$\mathcal{F}_i = \sum_j F_j \cdot v_i^{P_j} + \sum_k \tau_k \cdot \omega_i^{B_k} \tag{3.18}$$

where F_j is the j'th active force applied at point P_j, $v_i^{P_j}$ is the i'th partial velocity at P_j, τ_k is the k'th active torque applied on body B_k and $\omega_i^{B_k}$ is the i'th partial angular velocity at B_k.

3.5.2 Generalized Inertial Forces

Inertial forces are the fictitious forces which act against the active forces to keep the multibody system in a *dynamic equilibrium* at each instant according to D'Alembert's principle. Those forces are supposed to be generated by the mass and moment of inertia of rigid bodies

and proportional to their acceleration. The equation for *i'th generalized inertial force* can be written as

$$\mathcal{F}_i^* = \sum_j (-m_j a_j) \cdot v_i^{C_j} + \sum_k (-I_{B_k}\ {}^N\alpha^{B_k}) \cdot \omega_i^{B_k} \tag{3.19}$$

where m_j , a_j and C_j are the mass, acceleration and center of mass of j'th rigid body respectively, $v_i^{C_j}$ is the i'th partial velocity at C_j, $-I_{B_k}$ and ${}^N\alpha^{B_k}$ are the moment of inertia and angular acceleration of k'th rigid body B_k respectively, $\omega_i^{B_k}$ is the i'th partial angular velocity of B_k.

3.6 Solving Kane's Equation For Inverse Dynamics

Once the generalized active forces and generalized inertial forces are obtained, we can solve the series of equations 3.17. Since each equation of 3.17 is scalar, we can solve this system of equations in matrix form.
To achieve that, first the equations 3.17 are rearranged as

$$\begin{aligned} \mathcal{F}_1^* &= -\mathcal{F}_1 \\ \vdots &= \vdots \\ \mathcal{F}_n^* &= -\mathcal{F}_n \end{aligned} \tag{3.20}$$

From equations 3.18 and 3.19 we can write the equation 3.20 in matrix form as follows

$$M\ddot{Q} = T + E \tag{3.21}$$

The terms of this equation are matrices which are explained in detail below [28].

M : Mass Matrix

The *mass matrix* is a square matrix with dimension $n \times n$ which contains information about the instantaneous mass distribution of the multibody system. M^{-1} always exists since M is symmetric and positive semi-definite. Since M is the instantaneous mass distribution matrix, terms of M are functions of position and don't depend on velocity or acceleration.

$\ddot{Q}$: Acceleration Vector

Let q_i denote the i'th generalized coordinate of multibody system S. Then the *acceleration vector* $\ddot{Q}$ is defined as

$$\ddot{Q} = \begin{bmatrix} \ddot{q_1} \\ \ddot{q_2} \\ \vdots \\ \ddot{q_n} \end{bmatrix} \tag{3.22}$$

T : Torque Vector

For a multibody system composed of n rigid bodies connected with n actuated joints, let τ_i be the actuation torque for i'th joint. Then the torque vector will be

$$T = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_n \end{bmatrix} \tag{3.23}$$

E : Vector of Moments From Other Forces and Torques

E is a vector with dimension $n \times 1$ composed of terms which contain moments of gravitational, centrifugal and external forces and torques.

If we aim to obtain forward dynamics equation then we would need to solve the equation 3.21 for $\ddot{Q}$. By simply multiplying both sides of the equation 3.21 with M^{-1} from left we obtain the following

$$M^{-1}M\ddot{Q} = M^{-1}(T + E) \tag{3.24}$$

and the resulting equation will be the matrix form of forward dynamics equation

$$\ddot{Q} = M^{-1}(T + E) \tag{3.25}$$

In order to use this equation in a real-time simulation, we have to either solve the inverse of M in a symbolic way as an offline process or calculate it numerically at runtime.

In our study, our goal is to obtain the inverse dynamics equation of the system. Therefore we just need to solve the equation 3.21 for T as below

$$T = M\ddot{Q} - E \tag{3.26}$$

Since we don't need to calculate any matrix inverse with this equation, it's easier to build an efficient algorithm to calculate the proper actuation torques for any given vector of angular accelerations of a multibody system with rotational joints.

3.7 Deriving Equations of Motion for a Multibody System With Mobile Base

We can build generalized equations for kinematic properties as a starting point to obtain a generalized inverse dynamics algorithm. Before writing a generalized algorithm, it would be better to start with a simple concrete example.

Let S be a multibody system in fixed world reference frame N containing 7 rigid bodies connected to each other via rotational joints with no loop. Although there is no hierarchy between rigid bodies connected with joints, for deriving the equations, we'll regard the chain of the bodies as if they're in a hierarchical structure with the body with name R as root (see Figure 3.1). Let B be an arbitrary body of S, $B^\bullet$ be the position of the joint connecting body B to it's parent and B^* be the center of mass position of rigid body B. Let r_B be the vector $B^* - B^\bullet$, p_B be the vector starting from the center of mass point of parent body of B to $B^\bullet$.

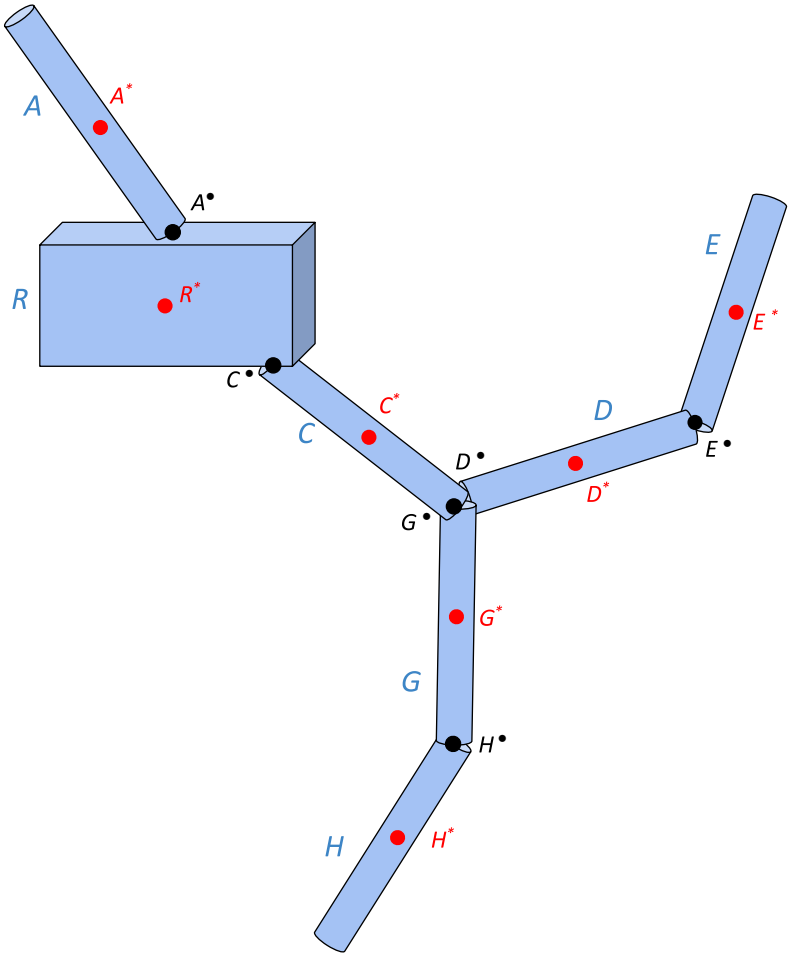


Figure 3.1: Multibody system S

By using equation 3.4 the velocity equations for each rigid body and joint of system S can be written as follows

$$
\begin{aligned}
{}^{O}v^{A^\bullet} &= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_A) \\
{}^{O}v^{A^*} &= {}^{O}v^{A^\bullet} + ({}^{N}\omega^{A} \times r_A) \\
&= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_A) + ({}^{N}\omega^{A} \times r_A) \\
{}^{O}v^{C^\bullet} &= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) \\
{}^{O}v^{C^*} &= {}^{O}v^{C^\bullet} + ({}^{N}\omega^{C} \times r_C) \\
&= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) \\
{}^{O}v^{D^\bullet} &= {}^{O}v^{C^*} + ({}^{N}\omega^{C} \times p_D) \\
&= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) + ({}^{N}\omega^{C} \times p_D) \\
{}^{O}v^{D^*} &= {}^{O}v^{D^\bullet} + ({}^{N}\omega^{D} \times r_D) \\
&= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) + ({}^{N}\omega^{C} \times p_D) + ({}^{N}\omega^{D} \times r_D) \\
{}^{O}v^{E^\bullet} &= {}^{O}v^{D^*} + ({}^{N}\omega^{D} \times p_E) \\
&= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) + ({}^{N}\omega^{C} \times p_D) + ({}^{N}\omega^{D} \times r_D) \\
&\quad + ({}^{N}\omega^{D} \times p_E) \\
{}^{O}v^{E^*} &= {}^{O}v^{E^\bullet} + ({}^{N}\omega^{E} \times r_E) \\
&= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) + ({}^{N}\omega^{C} \times p_D) + ({}^{N}\omega^{D} \times r_D) \\
&\quad + ({}^{N}\omega^{D} \times p_E) + ({}^{N}\omega^{E} \times r_E) \\
{}^{O}v^{G^\bullet} &= {}^{O}v^{C^*} + ({}^{N}\omega^{C} \times p_G) \\
&= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) + ({}^{N}\omega^{C} \times p_G) \\
{}^{O}v^{G^*} &= {}^{O}v^{G^\bullet} + ({}^{N}\omega^{G} \times r_G) \\
&= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) + ({}^{N}\omega^{C} \times p_G) + ({}^{N}\omega^{G} \times r_G) \\
{}^{O}v^{H^\bullet} &= {}^{O}v^{G^*} + ({}^{N}\omega^{G} \times p_H) \\
&= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) + ({}^{N}\omega^{C} \times p_G) + ({}^{N}\omega^{G} \times r_G) \\
&\quad + ({}^{N}\omega^{G} \times p_H) \\
{}^{O}v^{H^*} &= {}^{O}v^{H^\bullet} + ({}^{N}\omega^{H} \times r_H) \\
&= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) + ({}^{N}\omega^{C} \times p_G) + ({}^{N}\omega^{G} \times r_G) \\
&\quad + ({}^{N}\omega^{G} \times p_H) + ({}^{N}\omega^{H} \times r_H)
\end{aligned} \tag{3.27}
$$

The acceleration equations for multibody system S can be written using equation 3.7

$$
\begin{aligned}
{}^{O}a^{A^\bullet} &= {}^{O}a^{R^*} + {}^{N}\alpha^{R} \times p_A + {}^{N}\omega^{R} \times ({}^{N}\omega^{R} \times p_A) \\
{}^{O}a^{A^*} &= {}^{O}a^{A^\bullet} + {}^{N}\alpha^{A} \times r_A + {}^{N}\omega^{A} \times ({}^{N}\omega^{A} \times r_A) \\
&= {}^{O}a^{R^*} + {}^{N}\alpha^{R} \times p_A + {}^{N}\omega^{R} \times ({}^{N}\omega^{R} \times p_A) + {}^{N}\alpha^{A} \times r_A + {}^{N}\omega^{A} \times ({}^{N}\omega^{A} \times r_A) \\
{}^{O}a^{C^\bullet} &= {}^{O}a^{R^*} + {}^{N}\alpha^{R} \times p_C + {}^{N}\omega^{R} \times ({}^{N}\omega^{R} \times p_C) \\
{}^{O}a^{C^*} &= {}^{O}a^{C^\bullet} + {}^{N}\alpha^{C} \times r_C + {}^{N}\omega^{C} \times ({}^{N}\omega^{C} \times r_C) \\
&= {}^{O}a^{R^*} + {}^{N}\alpha^{R} \times p_C + {}^{N}\omega^{R} \times ({}^{N}\omega^{R} \times p_C) + {}^{N}\alpha^{C} \times r_C + {}^{N}\omega^{C} \times ({}^{N}\omega^{C} \times r_C) \\
{}^{O}a^{D^\bullet} &= {}^{O}a^{C^*} + {}^{N}\alpha^{C} \times p_D + {}^{N}\omega^{C} \times ({}^{N}\omega^{C} \times p_D) \\
{}^{O}a^{D^*} &= {}^{O}a^{D^\bullet} + {}^{N}\alpha^{D} \times r_D + {}^{N}\omega^{D} \times ({}^{N}\omega^{D} \times r_D) \\
&= {}^{O}a^{C^*} + {}^{N}\alpha^{C} \times p_D + {}^{N}\omega^{C} \times ({}^{N}\omega^{C} \times p_D) + {}^{N}\alpha^{D} \times r_D + {}^{N}\omega^{D} \times ({}^{N}\omega^{D} \times r_D) \\
{}^{O}a^{E^\bullet} &= {}^{O}a^{D^*} + {}^{N}\alpha^{D} \times p_E + {}^{N}\omega^{D} \times ({}^{N}\omega^{D} \times p_E) \\
{}^{O}a^{E^*} &= {}^{O}a^{E^\bullet} + {}^{N}\alpha^{E} \times r_E + {}^{N}\omega^{E} \times ({}^{N}\omega^{E} \times r_E) \\
&= {}^{O}a^{D^*} + {}^{N}\alpha^{D} \times p_E + {}^{N}\omega^{D} \times ({}^{N}\omega^{D} \times p_E) + {}^{N}\alpha^{E} \times r_E + {}^{N}\omega^{E} \times ({}^{N}\omega^{E} \times r_E)
\end{aligned}
\tag{3.28}
$$

${}^{O}a^{G^\bullet}$, ${}^{O}a^{G^*}$, ${}^{O}a^{H^\bullet}$ and ${}^{O}a^{H^*}$ can be calculated in a similar way with calculations of ${}^{O}a^{D^\bullet}$, ${}^{O}a^{D^*}$, ${}^{O}a^{E^\bullet}$ and ${}^{O}a^{E^*}$ respectively.

To proceed and calculate the partial velocity terms we must first write the generalized speed parameters. To end up with a general and simple solution, we choose the generalized speed parameters for our implementation under the guidance of [4]. Let N_i, N_j, N_k be unit orthogonal basis vectors of fixed world reference frame N and B_i, B_j, B_k be the rotational degree of freedoms along those basis vectors respectively for any given rigid body $B \in S$. We define the generalized speeds of multibody system S as follows

$$
\begin{aligned}
u_{R_i^*} &= {}^{O}v^{R^*} \cdot N_i & u_{D_i} &= {}^{N}\omega^{D} \cdot N_i \\
u_{R_j^*} &= {}^{O}v^{R^*} \cdot N_j & u_{D_j} &= {}^{N}\omega^{D} \cdot N_j \\
u_{R_k^*} &= {}^{O}v^{R^*} \cdot N_k & u_{D_k} &= {}^{N}\omega^{D} \cdot N_k \\
u_{R_i} &= {}^{N}\omega^{R} \cdot N_i & u_{E_j} &= {}^{N}\omega^{E} \cdot N_j \\
u_{R_j} &= {}^{N}\omega^{R} \cdot N_j & u_{E_k} &= {}^{N}\omega^{E} \cdot N_k \\
u_{R_k} &= {}^{N}\omega^{R} \cdot N_k & u_{E_i} &= {}^{N}\omega^{E} \cdot N_i \\
u_{A_i} &= {}^{N}\omega^{A} \cdot N_i & u_{H_j} &= {}^{N}\omega^{H} \cdot N_j \\
u_{A_j} &= {}^{N}\omega^{A} \cdot N_j & u_{H_k} &= {}^{N}\omega^{H} \cdot N_k \\
u_{A_k} &= {}^{N}\omega^{A} \cdot N_k & u_{H_i} &= {}^{N}\omega^{H} \cdot N_i \\
u_{C_i} &= {}^{N}\omega^{C} \cdot N_i & u_{G_j} &= {}^{N}\omega^{G} \cdot N_j \\
u_{C_j} &= {}^{N}\omega^{C} \cdot N_j & u_{G_k} &= {}^{N}\omega^{G} \cdot N_k \\
u_{C_k} &= {}^{N}\omega^{C} \cdot N_k & u_{G_i} &= {}^{N}\omega^{G} \cdot N_i
\end{aligned}
\tag{3.29}
$$

Now we can derive the partial velocity equations using the generalized speeds 3.29 and velocity equation 3.27 as explained in section 3.3. To make the rest of the calculations cleaner, the equations 3.27 can be rearranged in dyadic form

$$
\begin{aligned}
{}^{O}v^{A^*} &= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_A) + ({}^{N}\omega^{A} \times r_A) \\
&= {}^{O}v^{R^*} + \Big(({}^{N}\omega^{R}\cdot N_i)N_i + ({}^{N}\omega^{R}\cdot N_j)N_j + ({}^{N}\omega^{R}\cdot N_k)N_k\Big) \times p_A \\
&\quad + \Big(({}^{N}\omega^{A}\cdot N_i)N_i + ({}^{N}\omega^{A}\cdot N_j)N_j + ({}^{N}\omega^{A}\cdot N_k)N_k\Big) \times r_A \\
{}^{O}v^{C^*} &= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) \\
&= {}^{O}v^{R^*} + \Big(({}^{N}\omega^{R}\cdot N_i)N_i + ({}^{N}\omega^{R}\cdot N_j)N_j + ({}^{N}\omega^{R}\cdot N_k)N_k\Big) \times p_C \\
&\quad + \Big(({}^{N}\omega^{C}\cdot N_i)N_i + ({}^{N}\omega^{C}\cdot N_j)N_j + ({}^{N}\omega^{C}\cdot N_k)N_k\Big) \times r_C \\
{}^{O}v^{D^*} &= {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) + ({}^{N}\omega^{C} \times p_D) + ({}^{N}\omega^{D} \times r_D) \\
&= {}^{O}v^{R^*} + \Big(({}^{N}\omega^{R}\cdot N_i)N_i + ({}^{N}\omega^{R}\cdot N_j)N_j + ({}^{N}\omega^{R}\cdot N_k)N_k\Big) \times p_C \\
&\quad + \Big(({}^{N}\omega^{C}\cdot N_i)N_i + ({}^{N}\omega^{C}\cdot N_j)N_j + ({}^{N}\omega^{C}\cdot N_k)N_k\Big) \times r_C \\
&\quad + \Big(({}^{N}\omega^{C}\cdot N_i)N_i + ({}^{N}\omega^{C}\cdot N_j)N_j + ({}^{N}\omega^{C}\cdot N_k)N_k\Big) \times p_D \\
&\quad + \Big(({}^{N}\omega^{D}\cdot N_i)N_i + ({}^{N}\omega^{D}\cdot N_j)N_j + ({}^{N}\omega^{D}\cdot N_k)N_k\Big) \times r_D
\end{aligned}
\tag{3.30}
$$

The equations for the rest of the bodies can be rearranged in dyadic form in a similar way. Now the equations 3.30 are in dyadic form and the partial derivatives with respect to generalized speeds can be calculated easily. Let $v_{C_i}^{D^*}$ denote the partial velocity of D^* with respect to the generalized speed u_{C_i}. Here only the calculation for an arbitrary point D^* and an arbitrary generalized speed u_{C_i} is done as an example.

$$
\begin{aligned}
v_{C_i}^{D^*} &= \frac{\partial\, {}^{O}v^{D^*}}{\partial\, u_{C_i}} \\
&= \frac{\partial\ {}^{O}v^{R^*} + ({}^{N}\omega^{R} \times p_C) + ({}^{N}\omega^{C} \times r_C) + ({}^{N}\omega^{C} \times p_D) + ({}^{N}\omega^{D} \times r_D)}{\partial\, {}^{N}\omega^{C}\cdot N_i} \\
&= \frac{\partial\ \cancel{{}^{O}v^{R^*}} + \cancel{({}^{N}\omega^{R} \times p_C)} + ({}^{N}\omega^{C} \times r_C) + ({}^{N}\omega^{C} \times p_D) + \cancel{({}^{N}\omega^{D} \times r_D)}}{\partial\, {}^{N}\omega^{C}\cdot N_i} \\
&\quad \partial\Big(({}^{N}\omega^{C}\cdot N_i)N_i + ({}^{N}\omega^{C}\cdot N_j)N_j + ({}^{N}\omega^{C}\cdot N_k)N_k\Big) \times r_C \\
&= \frac{+\Big(({}^{N}\omega^{C}\cdot N_i)N_i + ({}^{N}\omega^{C}\cdot N_j)N_j + ({}^{N}\omega^{C}\cdot N_k)N_k\Big) \times p_D}{\partial\, {}^{N}\omega^{C}\cdot N_i} \\
&= N_i \times r_C + N_i \times p_D
\end{aligned}
\tag{3.31}
$$

Calculations for the rest of the point and generalized speed pairs of system S can be done similarly. The resulting partial velocities are given in table 3.1 .

	$\mathbf{R}^*$	$\mathbf{A}^*$	$\mathbf{C}^*$	$\mathbf{D}^*$	$\mathbf{E}^*$	$\mathbf{G}^*$	$\mathbf{H}^*$
$\mathbf{v}_{\mathbf{R}_\mathbf{i}^*}$	N_i	N_i	N_i	N_i	N_i	N_i	N_i
$\mathbf{v}_{\mathbf{R}_\mathbf{j}^*}$	N_j	N_j	N_j	N_j	N_j	N_j	N_j
$\mathbf{v}_{\mathbf{R}_\mathbf{k}^*}$	N_k	N_k	N_k	N_k	N_k	N_k	N_k
$\mathbf{v}_{\mathbf{R}_\mathbf{i}}$	0	$N_i \times p_A$	$N_i \times p_C$	$N_i \times p_C$	$N_i \times p_C$	$N_i \times p_C$	$N_i \times p_C$
$\mathbf{v}_{\mathbf{R}_\mathbf{j}}$	0	$N_j \times p_A$	$N_j \times p_C$	$N_j \times p_C$	$N_j \times p_C$	$N_j \times p_C$	$N_j \times p_C$
$\mathbf{v}_{\mathbf{R}_\mathbf{k}}$	0	$N_k \times p_A$	$N_k \times p_C$	$N_k \times p_C$	$N_k \times p_C$	$N_k \times p_C$	$N_k \times p_C$
$\mathbf{v}_{\mathbf{A}_\mathbf{i}}$	0	$N_i \times r_A$	0	0	0	0	0
$\mathbf{v}_{\mathbf{A}_\mathbf{j}}$	0	$N_j \times r_A$	0	0	0	0	0
$\mathbf{v}_{\mathbf{A}_\mathbf{k}}$	0	$N_k \times r_A$	0	0	0	0	0
$\mathbf{v}_{\mathbf{C}_\mathbf{i}}$	0	0	$N_i \times r_C$	$N_i \times r_C + N_i \times p_D$	$N_i \times r_C + N_i \times p_D$	$N_i \times r_C + N_i \times p_G$	$N_i \times r_C + N_i \times p_G$
$\mathbf{v}_{\mathbf{C}_\mathbf{j}}$	0	0	$N_j \times r_C$	$N_j \times r_C + N_j \times p_D$	$N_j \times r_C + N_j \times p_D$	$N_j \times r_C + N_j \times p_G$	$N_j \times r_C + N_j \times p_G$
$\mathbf{v}_{\mathbf{C}_\mathbf{k}}$	0	0	$N_k \times r_C$	$N_k \times r_C + N_k \times p_D$	$N_k \times r_C + N_k \times p_D$	$N_k \times r_C + N_k \times p_G$	$N_k \times r_C + N_k \times p_G$
$\mathbf{v}_{\mathbf{D}_\mathbf{i}}$	0	0	0	$N_i \times r_D$	$N_i \times r_D + N_i \times p_E$	0	0
$\mathbf{v}_{\mathbf{D}_\mathbf{j}}$	0	0	0	$N_j \times r_D$	$N_j \times r_D + N_j \times p_E$	0	0
$\mathbf{v}_{\mathbf{D}_\mathbf{k}}$	0	0	0	$N_k \times r_D$	$N_k \times r_D + N_k \times p_E$	0	0
$\mathbf{v}_{\mathbf{E}_\mathbf{i}}$	0	0	0	0	$N_i \times r_E$	0	0
$\mathbf{v}_{\mathbf{E}_\mathbf{j}}$	0	0	0	0	$N_j \times r_E$	0	0
$\mathbf{v}_{\mathbf{E}_\mathbf{k}}$	0	0	0	0	$N_k \times r_E$	0	0
$\mathbf{v}_{\mathbf{G}_\mathbf{i}}$	0	0	0	0	0	$N_i \times r_G$	$N_i \times r_G + N_i \times p_H$
$\mathbf{v}_{\mathbf{G}_\mathbf{j}}$	0	0	0	0	0	$N_j \times r_G$	$N_j \times r_G + N_j \times p_H$
$\mathbf{v}_{\mathbf{G}_\mathbf{k}}$	0	0	0	0	0	$N_k \times r_G$	$N_k \times r_G + N_k \times p_H$
$\mathbf{v}_{\mathbf{H}_\mathbf{i}}$	0	0	0	0	0	0	$N_i \times r_H$
$\mathbf{v}_{\mathbf{H}_\mathbf{j}}$	0	0	0	0	0	0	$N_j \times r_H$
$\mathbf{v}_{\mathbf{H}_\mathbf{k}}$	0	0	0	0	0	0	$N_k \times r_H$

Table 3.1: Partial velocities for multibody system S

Similarly, calculations for the rest of the point-generalized speed pairs of system S can be done similarly. The resulting partial angular velocities are given in table 3.2.

	R	**A**	**C**	**D**	**E**	**G**	**H**
$\omega_{\mathbf{R}_\mathbf{i}^*}$	0	0	0	0	0	0	0
$\omega_{\mathbf{R}_\mathbf{j}^*}$	0	0	0	0	0	0	0
$\omega_{\mathbf{R}_\mathbf{k}^*}$	0	0	0	0	0	0	0
$\omega_{\mathbf{R}_\mathbf{i}}$	N_i	0	0	0	0	0	0
$\omega_{\mathbf{R}_\mathbf{j}}$	N_j	0	0	0	0	0	0
$\omega_{\mathbf{R}_\mathbf{k}}$	N_k	0	0	0	0	0	0
$\omega_{\mathbf{A}_\mathbf{i}}$	0	N_i	0	0	0	0	0
$\omega_{\mathbf{A}_\mathbf{j}}$	0	N_j	0	0	0	0	0
$\omega_{\mathbf{A}_\mathbf{k}}$	0	N_k	0	0	0	0	0
$\omega_{\mathbf{C}_\mathbf{i}}$	0	0	N_i	0	0	0	0
$\omega_{\mathbf{C}_\mathbf{j}}$	0	0	N_j	0	0	0	0
$\omega_{\mathbf{C}_\mathbf{k}}$	0	0	N_k	0	0	0	0
$\omega_{\mathbf{D}_\mathbf{i}}$	0	0	0	N_i	0	0	0
$\omega_{\mathbf{D}_\mathbf{j}}$	0	0	0	N_j	0	0	0
$\omega_{\mathbf{D}_\mathbf{k}}$	0	0	0	N_k	0	0	0
$\omega_{\mathbf{E}_\mathbf{i}}$	0	0	0	0	N_i	0	0
$\omega_{\mathbf{E}_\mathbf{j}}$	0	0	0	0	N_j	0	0
$\omega_{\mathbf{E}_\mathbf{k}}$	0	0	0	0	N_k	0	0
$\omega_{\mathbf{G}_\mathbf{i}}$	0	0	0	0	0	N_i	0
$\omega_{\mathbf{G}_\mathbf{j}}$	0	0	0	0	0	N_j	0
$\omega_{\mathbf{G}_\mathbf{k}}$	0	0	0	0	0	N_k	0
$\omega_{\mathbf{H}_\mathbf{i}}$	0	0	0	0	0	0	N_i
$\omega_{\mathbf{H}_\mathbf{j}}$	0	0	0	0	0	0	N_j
$\omega_{\mathbf{H}_\mathbf{k}}$	0	0	0	0	0	0	N_k

Table 3.2: Partial angular velocities for multibody system S

Now using the tables 3.1, 3.2 and equations 3.18, 3.19 generalized active and inertial force equations for multibody system S can be obtained.

$$\begin{aligned}
\mathcal{F}_{R_i} &= (F_{ext_A} + m_A g)\cdot (N_i \times p_A) \\
&\quad + (F_{ext_C} + m_C g + F_{ext_D} + m_D g + F_{ext_E} + m_E g + F_{ext_G} + m_G g + F_{ext_H} + m_H g)\cdot (N_i \times p_C) \\
&\quad + (\tau_R - \tau_A - \tau_C)\cdot N_i
\end{aligned}$$

$$\begin{aligned}
\mathcal{F}_{C_i} &= (F_{ext_C} + m_C g)\cdot (N_i \times r_C) \\
&\quad + (F_{ext_D} + m_D g + F_{ext_E} + m_E g)\cdot (N_i \times r_C + N_i \times p_D) \\
&\quad + (F_{ext_G} + m_G g + F_{ext_H} + m_H g)\cdot (N_i \times r_C + N_i \times p_G) \\
&\quad + (\tau_C - \tau_D - \tau_G)\cdot N_i
\end{aligned}$$

$$\mathcal{F}_{D_i} = (F_{ext_D} + m_D g)\cdot (N_i \times r_D) + (F_{ext_E} + m_E g)\cdot (N_i \times r_D + N_i \times p_E) + (\tau_D - \tau_E)\cdot N_i$$

$$\mathcal{F}_{E_i} = (F_{ext_E} + m_E g)\cdot (N_i \times r_E) + \tau_E \cdot N_i \tag{3.32}$$

$$\begin{aligned}\mathcal{F}^*_{R_i} &= (-m_A a^{A^*})\cdot(N_i \times p_A) \\ &+ (-m_C a^{C^*} - m_D a^{D^*} - m_E a^{E^*} - m_G a^{G^*} - m_H a^{H^*})\cdot(N_i \times p_C) \\ &+ (-I_R \alpha^R)\cdot N_i\end{aligned}$$

$$\begin{aligned}\mathcal{F}^*_{C_i} &= (-m_C a^{C^*})\cdot(N_i \times r_C) \\ &+ (-m_D a^{D^*} - m_E a^{E^*})\cdot(N_i \times r_C + N_i \times p_D) \\ &+ (-m_G a^{G^*} - m_H a^{H^*})\cdot(N_i \times r_C + N_i \times p_G) \\ &+ (-I_C \alpha^C)\cdot N_i\end{aligned}$$

$$\begin{aligned}\mathcal{F}^*_{D_i} &= (-m_D a^{D^*})\cdot(N_i \times r_D) + (-m_E a^{E^*})\cdot(N_i \times r_D + N_i \times p_E) \\ &+ (-I_D \alpha^D)\cdot N_i\end{aligned}$$

$$\mathcal{F}^*_{E_i} = (-m_E a^{E^*})\cdot(N_i \times r_E) + (-I_E \alpha^E)\cdot N_i \tag{3.33}$$

Here we calculated $\mathcal{F}$ and $\mathcal{F}^*$ only for the degree of freedoms R_i, C_i, D_i and E_i. The calculations for the rest of the degree of freedoms can be done similarly. The equation of motion for each degree of freedom can be obtained using 3.17. Here we show the equation of motion just for E_i.

$$\mathcal{F}_{E_i} + \mathcal{F}^*_{E_i} = 0$$

$$(F_{ext_E} + m_E g)\cdot(N_i \times r_E) + \tau_E \cdot N_i + (-m_E a^{E^*})\cdot(N_i \times r_E) + (-I_E \alpha^E)\cdot N_i = 0 \tag{3.34}$$

Since motion controlling is an inverse dynamics problem, we want to calculate τ_E for any given α^E. As a result of our generalized speed choices, the torque term is appeared as a separate sum in the equation 3.34. So the equation can be easily solved for the torque term by lefting $\tau_E \cdot N_i$ alone.

$$\tau_E \cdot N_i = -(F_{ext_E} + m_E g)\cdot(N_i \times r_E) + (m_E a^{E^*})\cdot(N_i \times r_E) + (I_E \alpha^E)\cdot N_i \tag{3.35}$$

Here, we obtained the equation for the torque that needs to be applied to bodypart E along axis N_i to generate an angular acceleration of α^E. The calculations for the rest of the degree of freedoms can be done similarly and the all inverse dynamics equations can be obtained.

3.8 Building and Simplifying Generalized Equations for Real-time Systems

When we check Tables 3.1 and 3.2, we observe that there is a clear pattern that can be interpreted hierarchically. Therefore we infer general rules for calculating partial velocities by benefiting from the hierarchical relations of the bodyparts which are formulated as below (see Equations 3.36 and 3.37). In these formulas B and T denote arbitrary bodyparts of multibody system S. To keep the equations shorter and cleaner we define the following sets. B_{first} is the set of the first order(direct) children, $B_{children}$ is the set of all children and $B_{all} = \{B\} \cup B_{children}$ for arbitrary bodypart B of multibody system S.
The formulas for only degree of freedoms along N_i are written. Formulas for the rest of the degree of freedoms can be obtained similarly. In the rest of the equations, if B is the root bodypart R then r_B is simply taken as zero.

$$v_{B_i}^{T^*} = \begin{cases} N_i \times r_B & ,if\ T = B \\ 0 & ,if\ T \notin B_{children} \\ N_i \times r_B + N_i \times p_\psi \qquad where\ \psi \in B_{first}\ and\ T \in \psi_{all} & ,if\ T \in B_{children} \end{cases} \tag{3.36}$$

$$\omega_{B_i}^{T} = \begin{cases} N_i & ,if\ T = B \\ 0 & ,if\ T \neq B \end{cases} \tag{3.37}$$

By using these partial velocity equations, generalized active and inertial force equations can also be simplified.

$$\begin{aligned} \mathcal{F}_{B_i} = (N_i \times r_B) \cdot \sum_{\psi \in B_{all}} (F_{ext_\psi} + m_\psi g) + \sum_{\psi \in B_{first}} \Big[(N_i \times p_\psi) \cdot \sum_{\Phi \in \psi_{all}} (F_{ext_\Phi} + m_\Phi g) \Big] \\ + \Big[\tau_B - (\sum_{\psi \in B_{first}} \tau_\psi) \Big] \cdot N_i \end{aligned} \tag{3.38}$$

$$\mathcal{F}^*_{B_i} = (N_i \times r_B)\cdot \sum_{\psi \in B_{all}} (-m_\psi a^{\psi^*}) + \sum_{\psi \in B_{first}} \Big[(N_i \times p_\psi)\cdot \sum_{\Phi \in \psi_{all}} (-m_\Phi a^{\Phi^*})\Big] - I_B \alpha^B \cdot N_i \tag{3.39}$$

By using equation 3.17 which states that the sum of the active and inertial forces are zero, we can write the following equation for $\tau \cdot N_i$.

$$N_i \cdot \tau_B = N_i \cdot \sum_{\psi \in B_{first}} \tau_\psi - (N_i \times r_B) \cdot \sum_{\psi \in B_{all}} \Big[F_{ext_\psi} + m_\psi (g - a^{\psi^*})\Big] - \sum_{\psi \in B_{first}} \Big\{(N_i \times p_\psi) \cdot \sum_{\Phi \in \psi_{all}} \big[F_{ext_\Phi} + m_\Phi (g - a^{\Phi^*})\big]\Big\} + I_B \alpha^B \cdot N_i \tag{3.40}$$

For a bodypart ψ if we denote the total linear force on psi with TLF_ψ then

$$TLF_\psi = F_{ext_\psi} + m_\psi (g - a^{\psi^*}) + \sum_{\Phi \in \psi_{first}} TLF_\Phi \tag{3.41}$$

If we substitute TLF in equation 3.40 the we obtain the recursive simplified formula for $N_i \cdot \tau$.

$$N_i \cdot \tau_B = N_i \cdot \sum_{\psi \in B_{first}} \tau_\psi - (N_i \times r_B) \cdot TLF_B - \sum_{\psi \in B_{first}} \Big[(N_i \times p_\psi) \cdot TLF_\psi\Big] + I_B \alpha^B \cdot N_i \tag{3.42}$$

For any vectors v_1, v_2 and v_3 the equation below holds

$$(v_1 \times v_2) \cdot v_3 = v_1 \cdot (v_2 \times v_3) \tag{3.43}$$

Therefore we can rearrange 3.42 as follows:

$$N_i \cdot \tau_B = N_i \cdot \sum_{\psi \in B_{first}} \tau\psi - N_i \cdot (r_B \times TLF_B) - N_i \cdot \sum_{\psi \in B_{first}} [p_\psi \times TLF_\psi] + N_i \cdot I_B \alpha^B \tag{3.44}$$

And moreover any vector v can be written as $v = (\ N_i \cdot v\ ,\ N_j \cdot v\ ,\ N_k \cdot v\ )$. And therefore τ_B can also be written in terms of N_i, N_j and N_k as follows

$$\tau_B = (\ N_i \cdot \tau_B\ ,\ N_j \cdot \tau_B\ ,\ N_k \cdot \tau_B\ ) \tag{3.45}$$

Then the general recursive simplified formula of τ_B is obtained as follows:

$$\begin{aligned} \tau_B &= \sum_{\psi \in B_{first}} \tau_\psi - (r_B \times TLF_B) - \sum_{\psi \in B_{first}} (p_\psi \times TLF_\psi) + I_B \alpha^B \\ &= \sum_{\psi \in B_{first}} \Big[\tau_\psi - (p_\psi \times TLF_\psi)\Big] - (r_B \times TLF_B) + I_B \alpha^B \end{aligned} \tag{3.46}$$

The TLF term that we used in this resulting torque calculation equation is written as a recursive formula where parent bodies are using the TLF values of their children. This TLF formula also contains linear acceleration terms. So the linear acceleration terms have to be calculated beforehand in order to calculate the joint torques. As seen in equation 3.28 the linear acceleration of each bodypart is calculated by using the linear acceleration of its parent bodypart. Therefore, the overall algorithm of calculating joint torques of a multibody system S consists of two passes. In the first pass, the linear acceleration of each bodypart is calculated from root bodypart to leaf bodyparts. And in the second pass, the TLF and actual joint torque values are calculated by traversing from leaf bodyparts to root bodypart (see Figure 3.2).

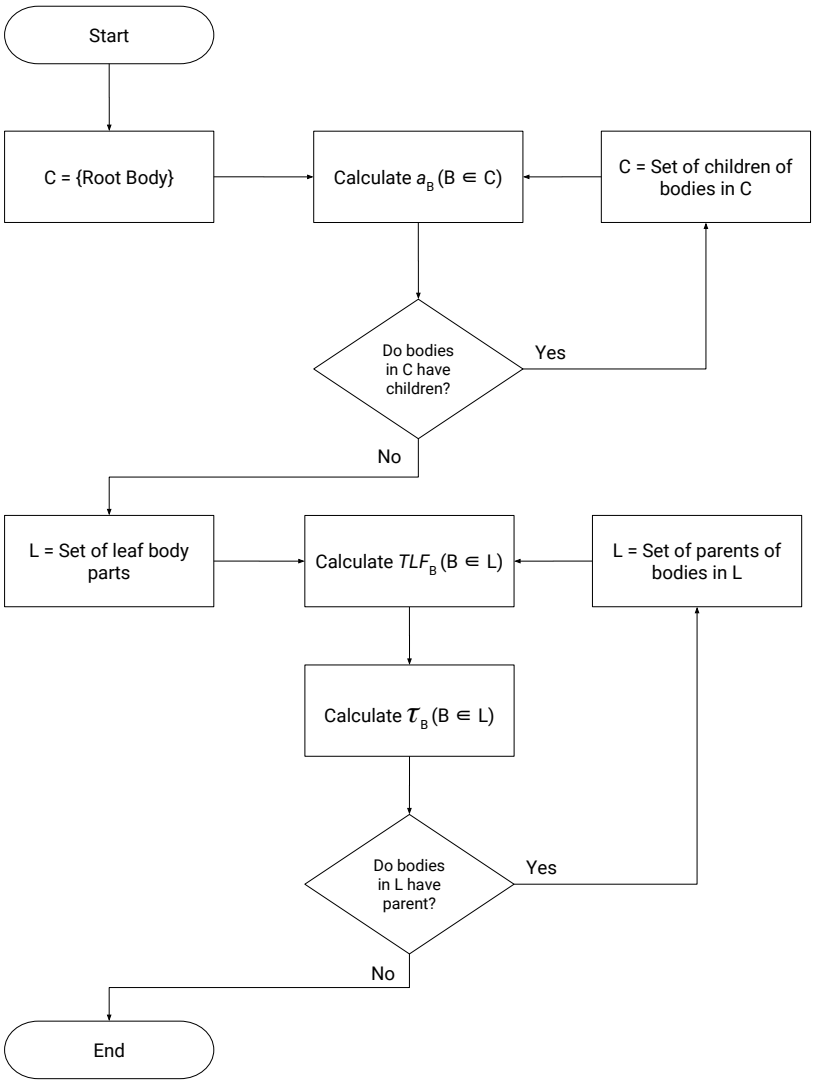


Figure 3.2: Flowchart of our resulting inverse dynamics algorithm

3.9 Implementation and Results

We've implemented the resulting algorithm in $C\#$ and run our tests in *Unity runtime environment*. We used a system with 4 core i7 CPU, 16GB ram and Radeon R9 M370X graphics card as the testbed. Our algorithm had an average of calculation time less than $1ms$ for a character model with 51 degrees of freedom. We run all of the test scenarios using Unity's built in physics engine ($Physx$) with a fixed timestep of 1/100 seconds. Decreasing the timestep increases the simulation stability but the smallest ideal timestep that can also be used in real world applications is 1/100 seconds.

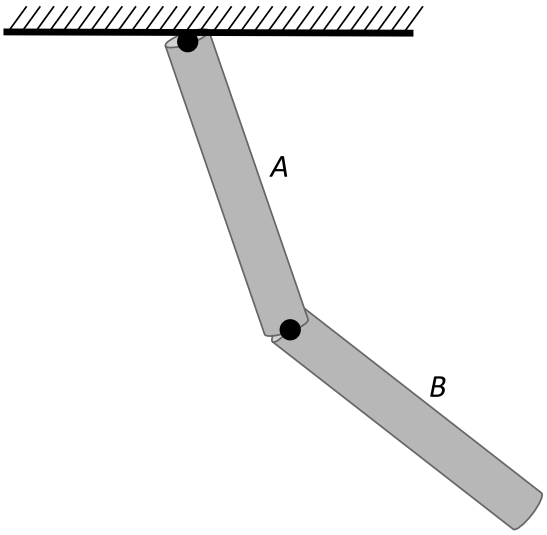


Figure 3.3: Simple multibody system with 2 links in 2D space

We compared our method with some of the most commonly used controllers in physics based animation researches. Although all calculations in our method are made with 3D vectors and quaternions, we first created a simple multibody system with 2 rigid bodies in 2D space just to show the comparisons more clearly (see Figure 3.3). Each rigid body has a cylinder shape of 30cm height and 7cm base diameter with uniformly distributed mass of 1.54 kilograms. To observe the controllers' reactions to external perturbations, we dropped a ball with a mass of 1 kilogram from 1 meter height (see Figure 3.4). Since the requirements and results vary dramatically between steady pose tracking and motion tracking for some cases, we added results for both when needed.

As we also mentioned in section 2.2.1, PD controller is the most widely used method to calculate joint torques in physics based animation research area due to its simplicity and efficiency. So as a first step, we compared our system with a manually tuned, critically damped PD controller. Figure 3.5 shows the comparison of resulting angles both from our method and the PD controller. We lowered our controller's stiffness value to 200 just to make the visual comparison easier. The tracking of body B with PD controller is stable and critically damped except the steady state error which is caused by the gravity. On the other hand, the PD controller working on body A overshoots its target orientation and then settles down to steady state with a huge error.

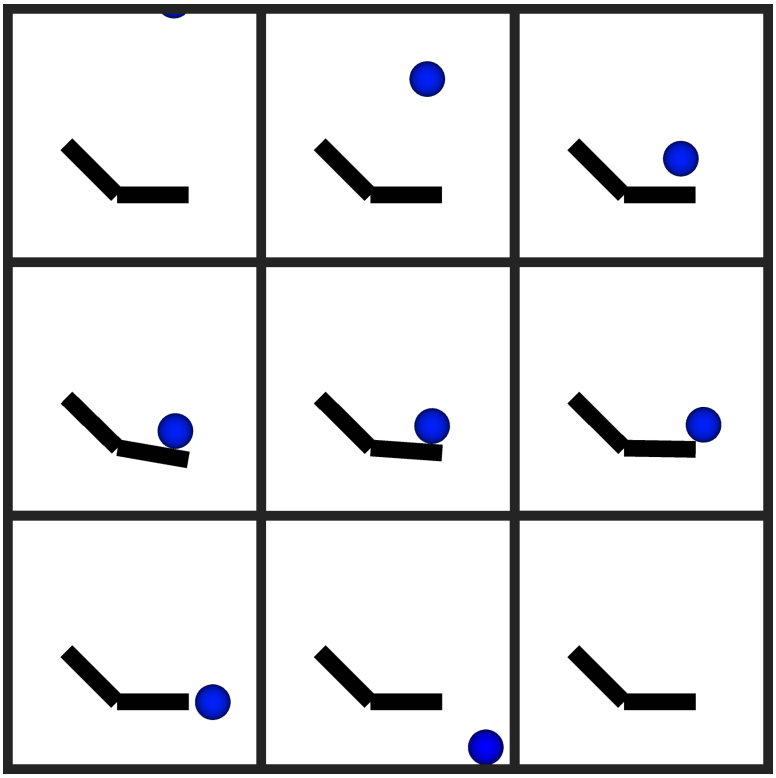
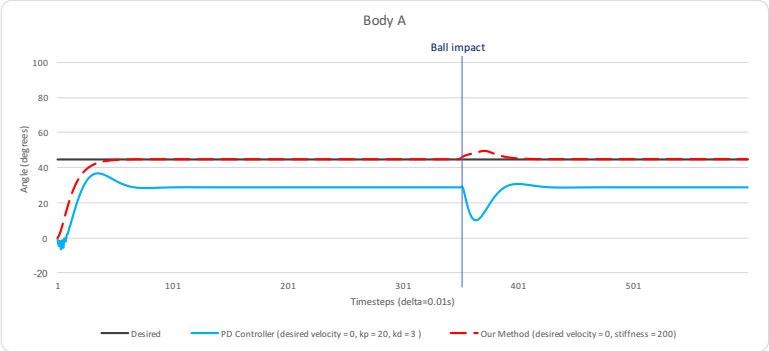


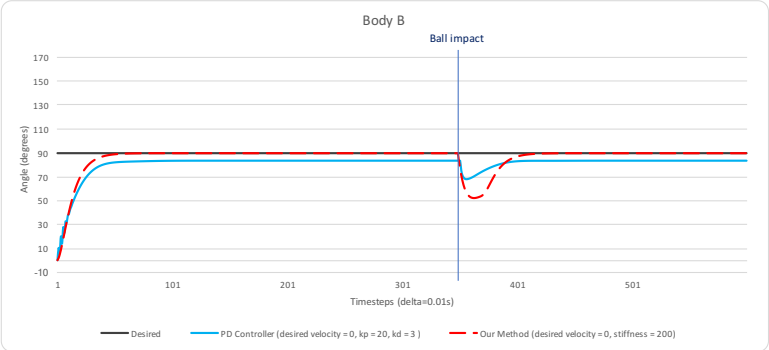
Figure 3.4: Screenshots of 2d scenario

As the next scenario, we tuned the gains of the PD controllers to get more stiff tracking results and to check whether it fixes the steady state errors and parent body's overshoot problems. Increasing the gains of PD controllers is only possible up to $kp = 50$ and $kd = 3$ values for our setup. The simulation becomes instable when greater values are given as gains. We set our system's stiffness value accordingly to make comparison easier. As seen in Figure 3.6 the results for the PD controller are a bit better but still far from perfect. The overshoot problem for body A remains and although the steady state errors for both rigid bodies are smaller compared to Figure 3.5, but still there exists some errors. The steady state error is caused by the constant gravitational force acting on the rigid bodies. To tackle this issue, [1] proposed a method called *gravity compensation* as explained in Section 2.2.3.

When we test the same scenario for motion tracking, the errors and instability for body A increase dramatically as seen in Figure 3.7. The main cause of this instability issue is the fact that the resulting total torque acting on body A isn't equal to the torque calculated by its PD controller. It's clear to see that the negative of the torque generated by the PD controller



(a) Rigid body A



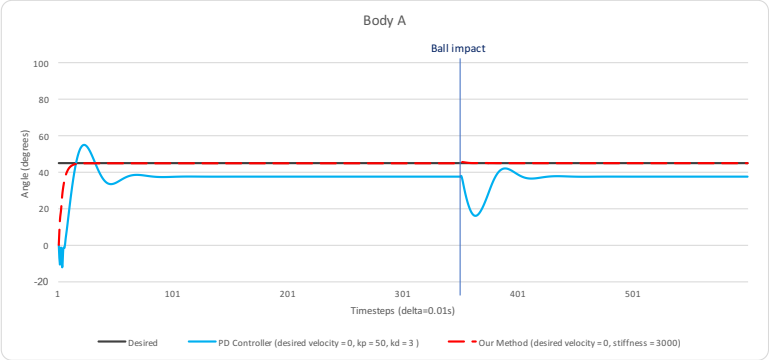
(b) Rigid body B

Figure 3.5: Steady pose tracking comparison with critically damped PD controller

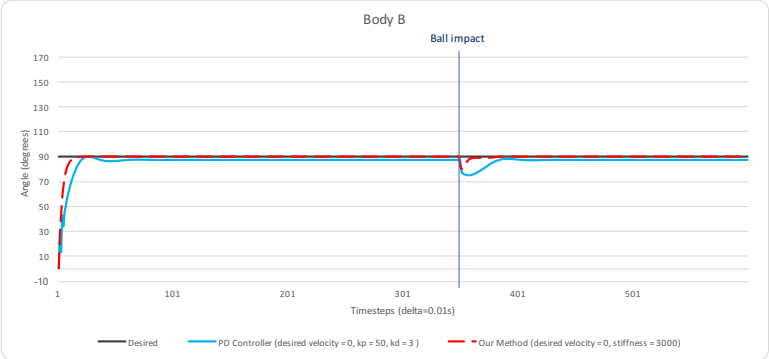
of body B is automatically applied to body A as a result of joint articulation. That issue is also addressed by [1] and fixed using a simple method called *torque bubble up* as mentioned in Section 2.2.3.

As we mentioned above, [1] proposed some fixes for these problems of PD controller. So we compared our method with the controller proposed by [1]. For the steady pose tracking scenario, the method proposed by [1] doesn't produce any steady state error as seen in Figure 3.8. Although the overshooting issue is still noticeable, it's clear to see that the results are better than pure PD controller's results.

Figure 3.9 shows the comparison between [1] and our controller for motion tracking. The results for [1] aren't as good as its steady tracking results. Especially body A wobbles quite significantly. The main reason behind this issue is the fact that the underlying dynamics of the whole multibody system isn't taken into account when calculating joint torques. From Newton's second law, we know that the mass and acceleration of body B generates a force



(a) Rigid body A



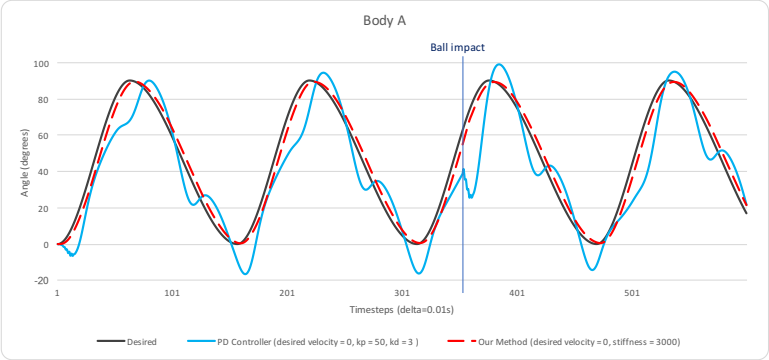
(b) Rigid body B

Figure 3.6: Steady pose tracking comparison with stiff PD controller

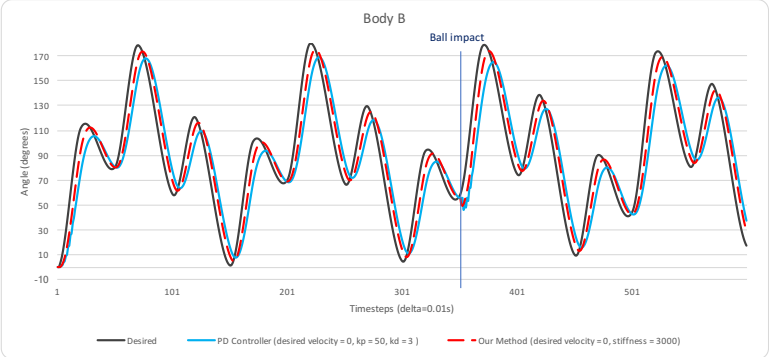
acting on body A. Some portion of this force also translates into torque since the application point of the force isn't on body A's center of mass.

As we mentioned in section 2.1, [29] addressed some issues in the main formulation of PD controller and represented a new formulation called SPD(Stable PD) Controller which promises a stable tracking with better timing. We also implemented their method as explained in the paper but we didn't see any improvement for the tracking results (see Figure 3.10). Furthermore we observed that the simulation stability is worse than PD controller so we have to lower the gains to be able to run the tests. The results for steady pose tracking are even worse than PD controller due to lowered gain values.

We didn't see any improvements in the results of SPD controller over PD controller for motion tracking, too (see Figure 3.11). SPD controller doesn't seem to solve neither steady state errors nor the upper body wobbling issues.



(a) Rigid body A

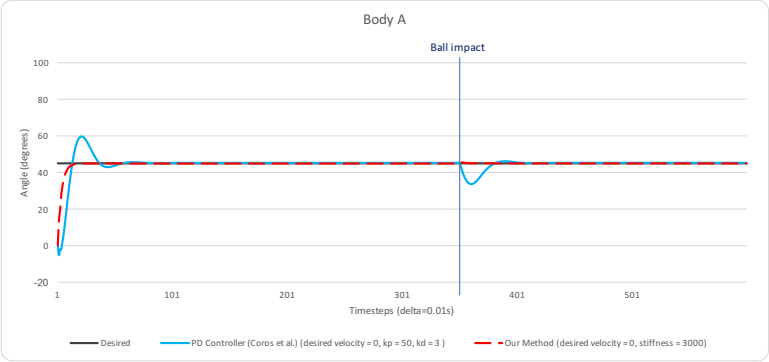


(b) Rigid body B

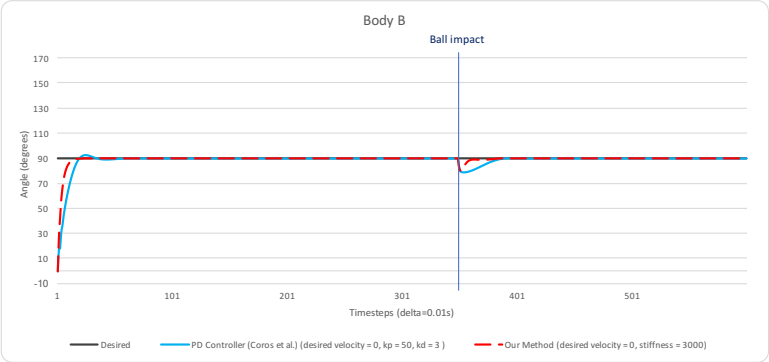
Figure 3.7: Motion tracking comparison with stiff PD controller

As the next scenario we evaluated our method with different stiffness values for tracking a steady pose, as seen in Figure 3.12. The results clearly show that our controller doesn't overshoot or produce any steady state error and remains critically damped for all stiffness values. In our method, changes in stiffness values reflect to the resulting tracking behaviour consistently without need to tune two complexly related parameters as in PD controller.

In another scenario, again we evaluated our method with different stiffness values but this time for tracking a non-steady motion, as seen in Figure 3.13. The results show that for rigid body A stiffness values effect the timing of the tracking behaviour. As the gain increases, tracking gets more accurate. Loosing the time synchronization with low gains also result in loss of accurate tracking of peak points. While this timing inaccuracy doesn't cause big deviations from the reference motion for rigid body A, for rigid body B the results are quite different than the reference. As seen in Figure 3.13b, tracking with stiffness value of 30, results in a motion which cannot even reflect the pattern of the reference motion. This differ-



(a) Rigid body A

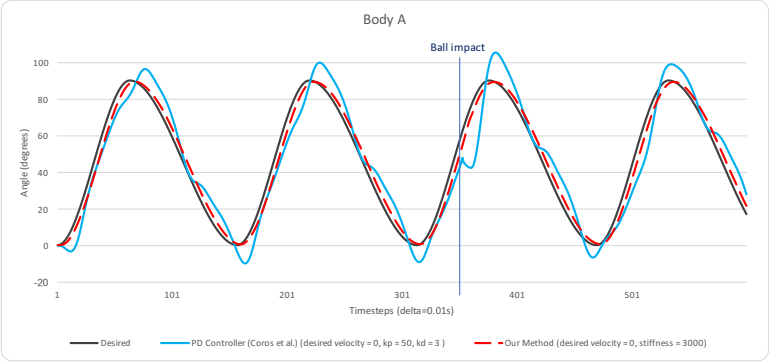


(b) Rigid body B

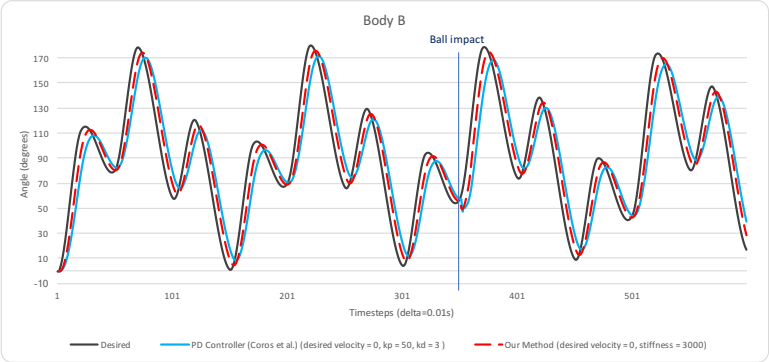
Figure 3.8: Steady pose tracking comparison with the controller proposed by [1]

ence between results of rigid body A and rigid body B is due to the different characteristics of two reference motions. The frequency of the reference motion that rigid body A intends to track is lower than the frequency of the reference motion of rigid body B. In other words the reference motion of rigid body B changes more rapidly than the other. Therefore, falling short to catch accurate timing results in falling short to catch the rapid change in the reference motion.

In all of the former scenarios, desired velocity has been taken as zero, which means that we calculated the desired angular acceleration by taking into account only the orientational deviation from the reference motion. As an another scenario, we also test our method by taking the desired velocity from the reference motion, and compare the results with the results of zero desired velocity case. We extract the desired velocity from reference motion by finite differences method. Finite differences method is a widely used numerical method for



(a) Rigid body A



(b) Rigid body B

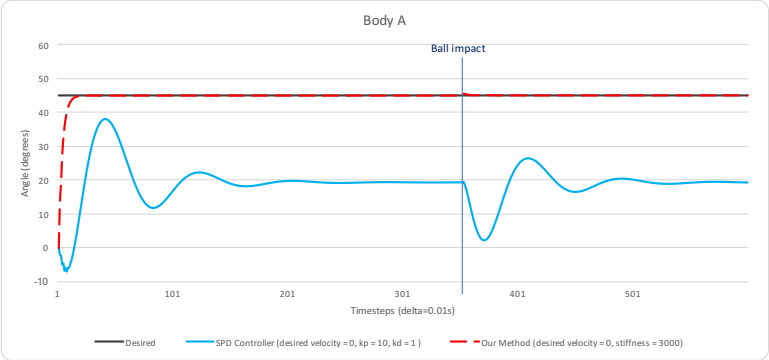
Figure 3.9: Motion tracking comparison with controller proposed by [1]

calculating differential equations. The equation for calculating desired velocity w_d at time t, from reference motion with finite differences is as follows:

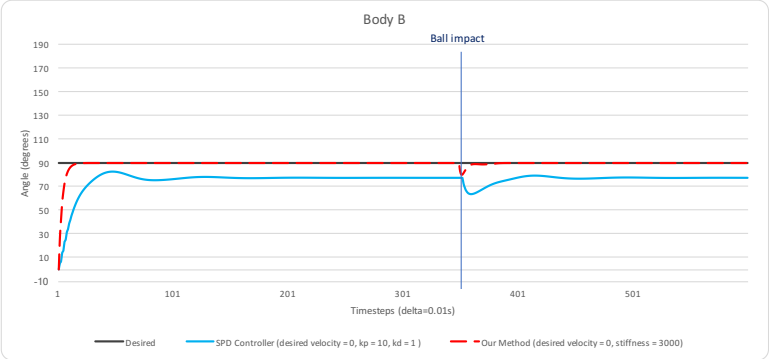
$$w_d(t) = \frac{q_r(t) - q_r(t - \Delta t)}{\Delta t} \tag{3.47}$$

Here, Δt is the fixed timestep of the simulation, t is the current time, $q_r(t)$ and $q_r(t - \Delta t)$ are the reference orientations at time t and $t - \Delta t$.

Figure 3.14a demonstrates the results of two simulations of rigid body A, one with the desired velocity from reference motion and the other one with desired velocity of zero. In both cases, stiffness is set to a constant value of 500. It's seen that, tracking with desired velocity from motion, has nearly the exact timing of the reference motion, while the other one, with zero desired velocity, falls short to catch the timing. The only flaw of the dvfm(desired



(a) Rigid body A



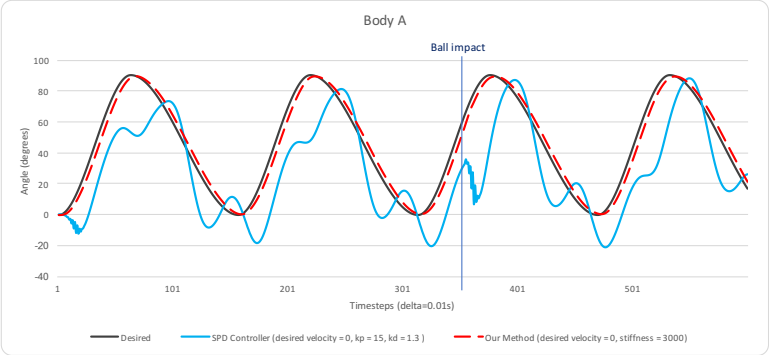
(b) Rigid body B

Figure 3.10: Steady pose tracking comparison with SPD controller

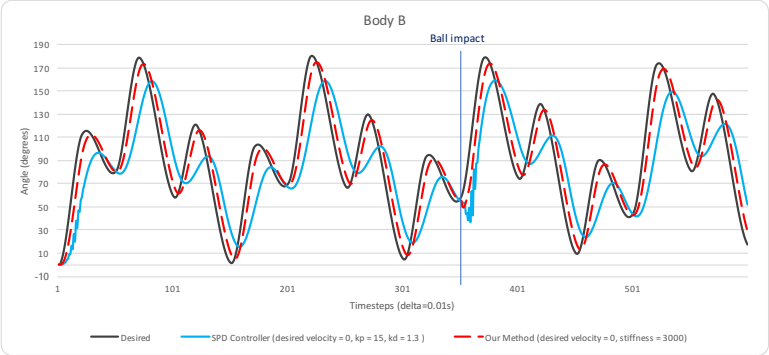
velocity from motion) result seems to be the minor overshoots at the peek points. These deviations at the peek points are seen more clearly in Figure 3.14b. Our interpretation is that our calculation of desired velocities with finite differences method results in these deviations.

As a different scenario, again we take the desired angular velocity from the reference motion, but this time we work with a lower stiffness value. As seen in Figure 3.15a, we achieve a far better tracking with dvfm compared to tracking with zero desired velocity. However, timing is not as perfect as in tracking with higher stifness value(see Figure 3.14a).

In the next three scenarios, we use the desired acceleration calculation formula with feedforward term. In this formula an additional acceleration term is calculated from the reference motion with finite diferences and added to the simple PD Controller formula (see equation 2.3). The equation for calculating the feedforward term from reference motion is as follows:



(a) Rigid body A



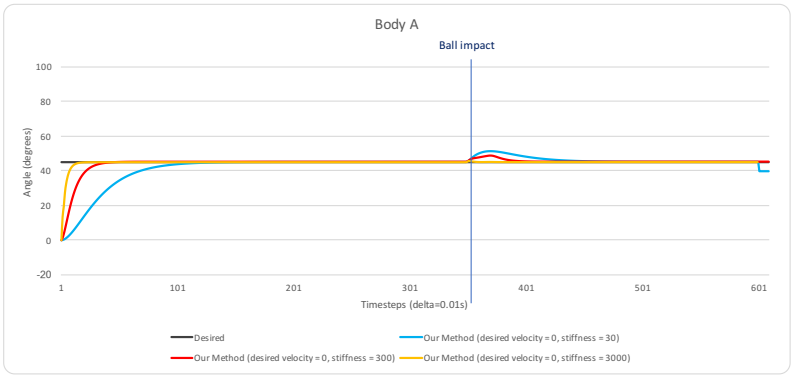
(b) Rigid body B

Figure 3.11: Motion tracking comparison with SPD controller

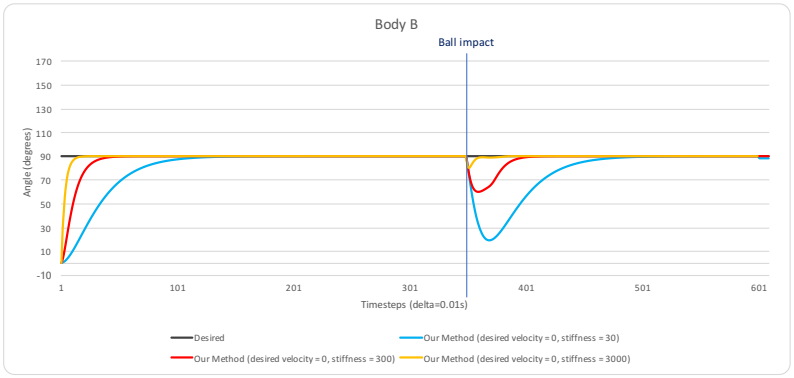
$$\alpha_r(t) = \frac{w_r(t) - w_r(t - \Delta t)}{\Delta t} \tag{3.48}$$

Here, Δt is the fixed timestep of simulation, t is the current time, $w_r(t)$ and $w_r(t - \Delta t)$ are the reference angular velocities at time t and $t - \Delta t$.

In the first scenario with feedforward term, we compare the results of two simulations, one is our method with feedforward term and the other one is our method without feed forward term(see Figure 3.16). In both of the simulations we take the desired velocity as zero and the stiffness value as 30. The results of simulating with rigid body A (see Figure 3.16a) and with rigid body B (see Figure 3.16b) show that adding feedforward term does not increase the tracking quality of the simulation. However, as mentioned in Section 2.2, adding feedforward term is expected to give better results. Our interpretation for the reason of this discrepancy is



(a) Rigid body A

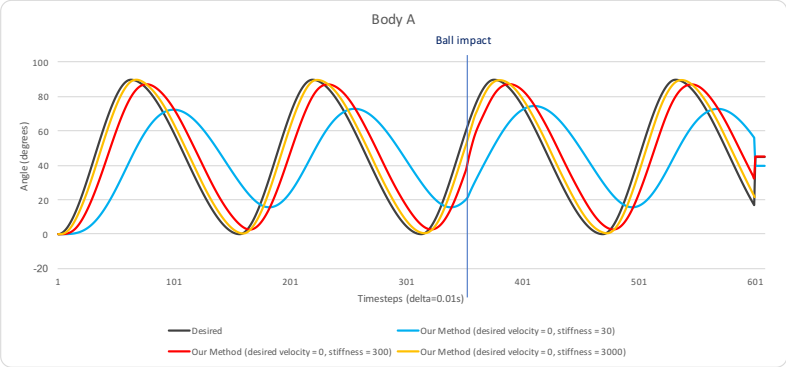


(b) Rigid body B

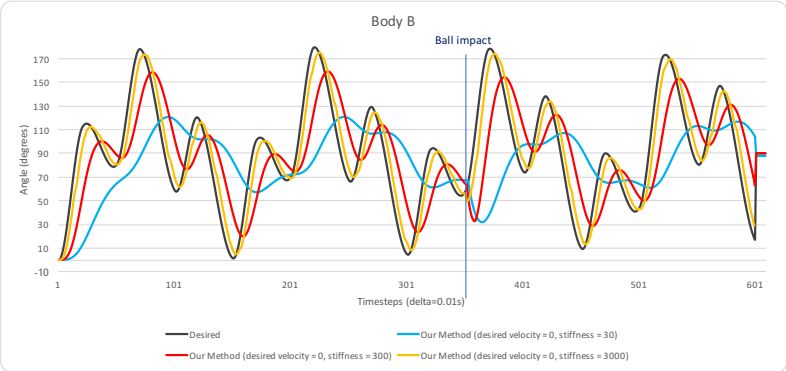
Figure 3.12: Steady pose tracking of our controller with different stiffness values

the choice of desired angular velocity as zero. We think that, when desired angular velocity is set to zero, the velocity term of the PD Controller formula and the feedforward term work against each other. While the feedforward term contributes to the acceleration of the simulated motion, the velocity term pulls it back in order to keep the velocity constantly at zero.

The results of our second scenario seem to support this interpretation. In the second scenario, again we compare the results of two simulations with our method, one with feedforward term and the other one without it. The stiffness value is the same as the previous scenario but this time we take the desired angular velocity values from the reference motion. As seen in Figure 3.17 using feedforward term and taking desired angular velocity from reference motion create a motion which is almost identical to the reference motion, especially when there are no external perturbations. According to our interpretation, this time the velocity



(a) Rigid body A

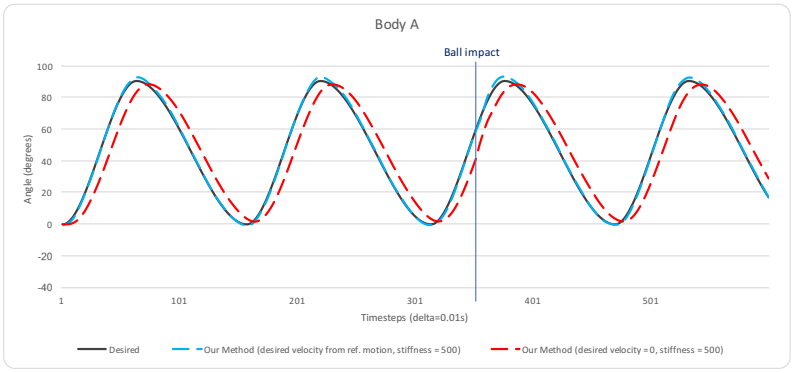


(b) Rigid body B

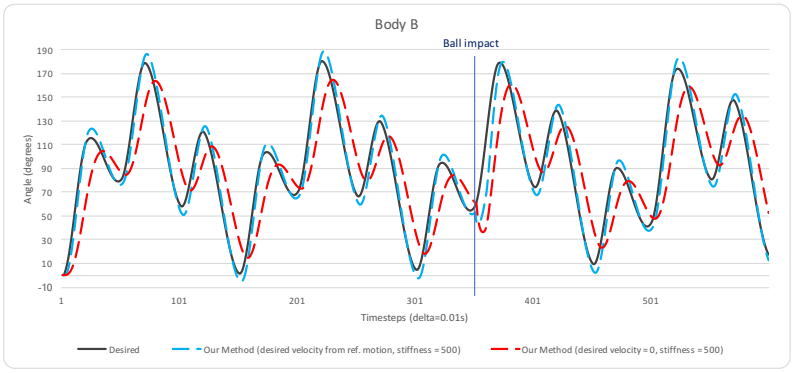
Figure 3.13: Motion tracking of our controller with different stiffness values

term and feedforward term, which are both extracted from the reference motion, have the same goal, therefore they support each other. Only, when there is a ball impact on rigid body B, the simulated motion deviates from the reference motion as a natural effect of the impact. But after some time it continues to track the reference motion, almost perfect, again.

In the third scenario we evaluate our method using feedforward term and desired velocity from reference motion, with two different stiffness values. We especially chose a very high stiffness value(3000) and very low stiffness value(30), to test the stability and timing of the resulting motions at extreme values. As seen in Figure 3.18, the tracking quality doesn't change with changing stiffness values. The only difference can be seen in 3.18b, when there is a ball impact on rigid body B. Here, we see that, the controller with high stiffness value recovers much faster than the other. This is a natural result of stiff control, which enables converging to target faster by producing higher accelerations.



(a) Rigid body A



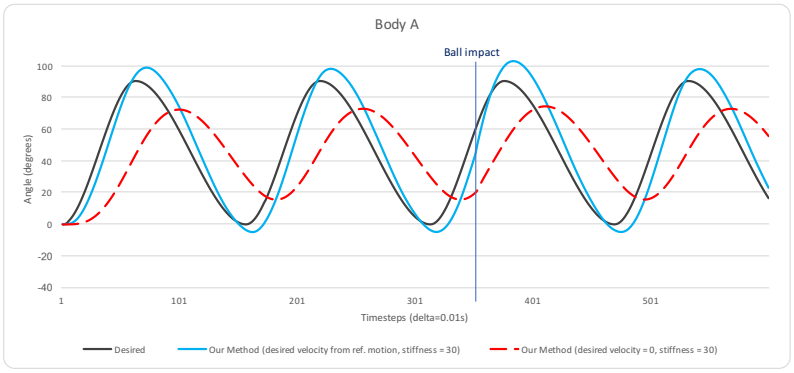
(b) Rigid body B

Figure 3.14: The effect of using desired angular velocity from reference motion for our controller

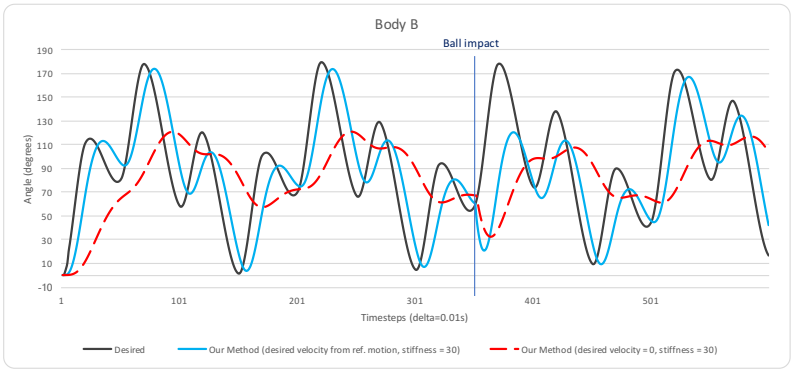
This configuration of our method, with feedforward term and desired velocity taken from the reference motion seems to produce the best tracking results for this simple 2D test scenario.

We also created another scene involving a humanoid multibody system with 51 degrees of freedom in 3D space to validate our method for real application scenarios. The humanoid model consists of 16 rigid bodies, which are connected with ball-socket joints consisting of 51 degrees of freedom. The model weighs 52 kilograms and it's 1.6 meters high. This humanoid model can be seen in Figure 3.19.

We choose a highly dynamic human motion with several sprints, jumps and sudden stops. Although our system worked great with the default timestep $1/100$, we aren't able to run a stable simulation for any variant of the PD controller with the same timestep. To tackle the stability issues we had to lower the timestep dramatically to $1/2000$. We didn't need to lower



(a) Rigid body A

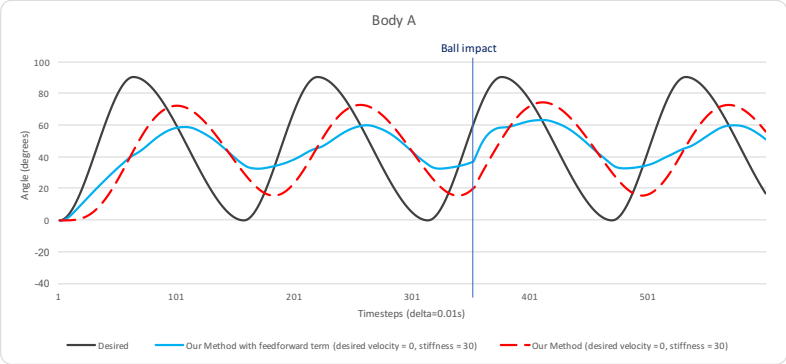


(b) Rigid body B

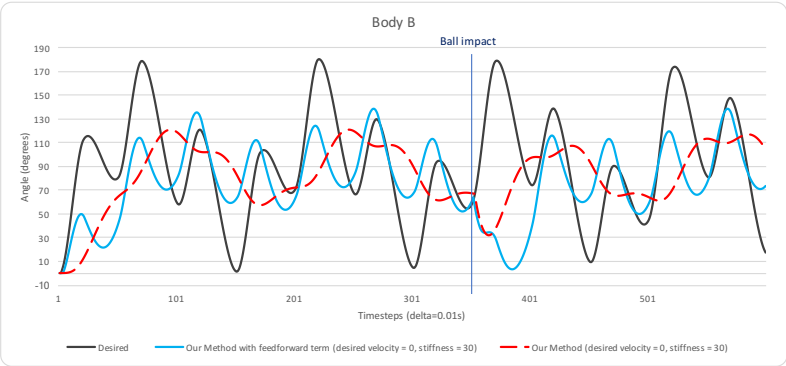
Figure 3.15: The effect of using desired angular velocity from reference motion for our controller with low gains

the timesteps for our controller since we observed that our controller produces great tracking results event at $1/100$. We run tests with all variants of PD controller for our 3D scenario. PD controller with gravity compensation and torque bubble upping produce the best results among other variants. So we compared our system's results only with that variant of PD controller.

Figure 3.20 shows the comparison of average rotational errors in degrees for all bodyparts during the simulation. Although the results of our method are better in this comparison, the difference is smaller than it seems in the simulation videos. According to our interpretation, this mismatch between the graphical result and the visual result in simulation video is due to the fact that the rotational errors of some body parts effect the positional errors more. Therefore, the average rotational errors of the body parts don't reflect the success or failure



(a) Rigid body A



(b) Rigid body B

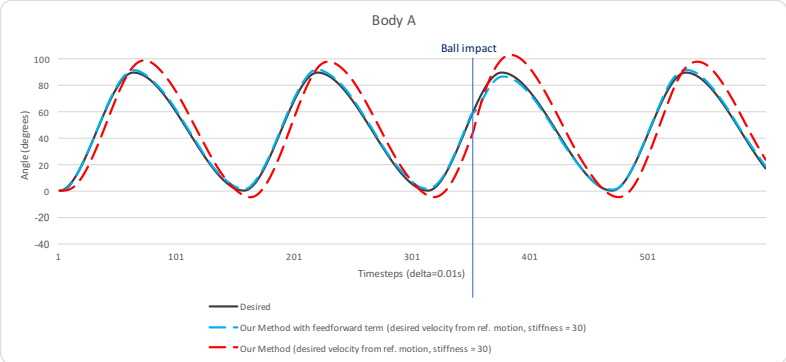
Figure 3.16: The effect of using feedforward term for our controller

of the visual results, while the rotational errors of the bodyparts which have lower degrees in the hierarchy are more compliant with the visual results. The graphs shown below support this interpretation of ours (Figure 3.21, Figure 3.22, Figure 3.23)

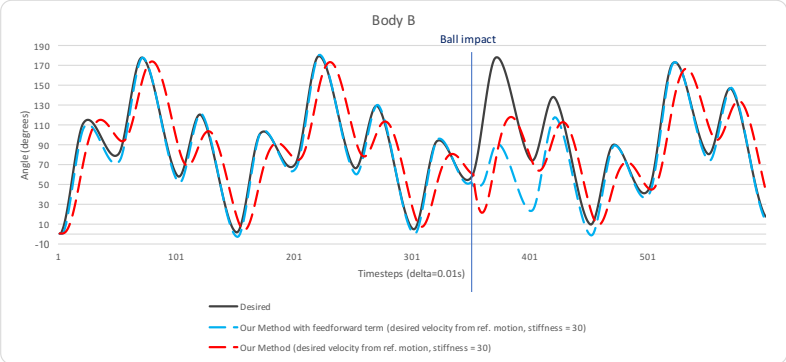
The first graph (Figure 3.21) displays the orientation error comparison of the right foot, which is a leaf body part, in other words one of the bodyparts with the highest degree in the hierarchy. As seen in the figure, there isn't a dramatic difference between the errors produced by our method and the other method.

However, the second graph (Figure 3.22) displays a different result for the positional errors of the same body part. In this figure, it's seen that our method produces much more accurate tracking results for the right foot position. The third graph(Figure 3.23) explains the reason of this mismatch between these two graphs.

In this third graph, we see the error comparison for the right upper leg, which has a lower



(a) Rigid body A

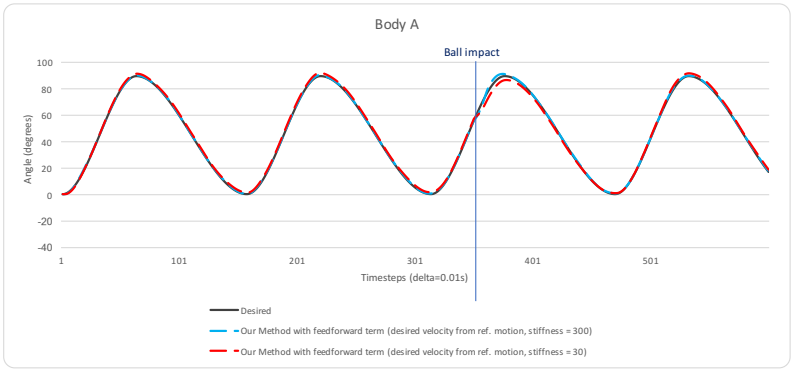


(b) Rigid body B

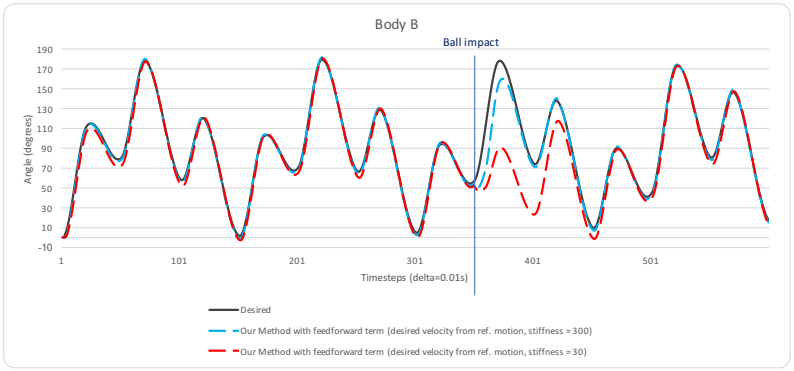
Figure 3.17: The effect of using both feedforward term and desired angular velocity from reference motion for our controller

degree in the hierarchy, as the grand grand parent of the right foot. As seen in the figure, the rotational errors of our method are much less than other method's errors for the right upper leg, which is quite compliant both with the error results for the right foot position and the visual results in the simulation video. Therefore comparing the positional errors of the resulting motions gives more accurate interpretation of the results and differences of the methods. Figure 3.24 displays the average positional errors of the bodyparts at each time step.

We also added Table 3.3 for detailed average orientation and position errors for each body-part. In this table, the orientational errors are given in degrees and the positional errors are given in meters.



(a) Rigid body A



(b) Rigid body B

Figure 3.18: The effect of stiffness term when using both feedforward term and desired angular velocity from reference motion in our controller

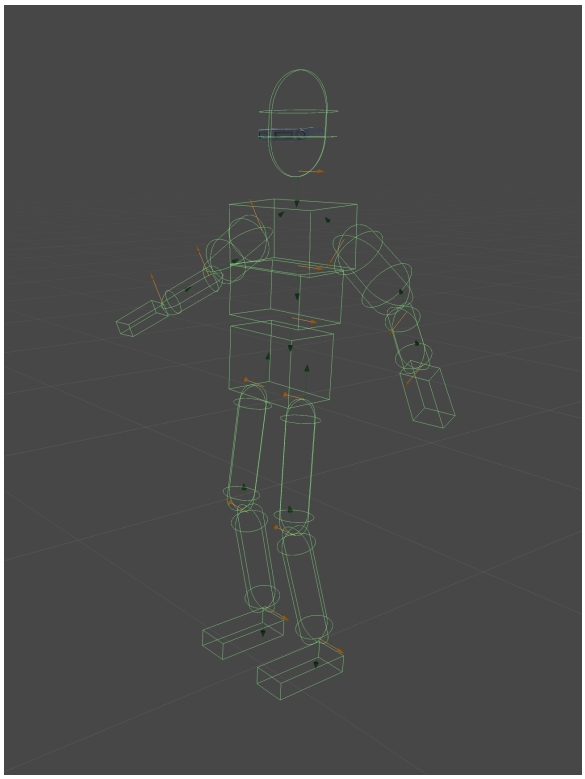


Figure 3.19: The humanoid multibody model

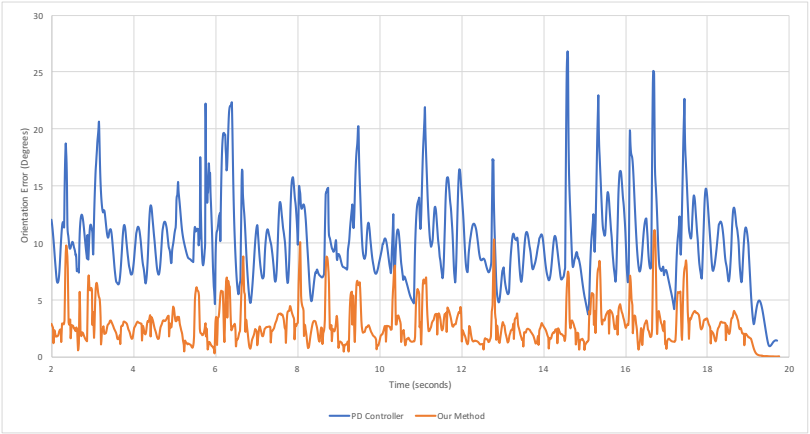


Figure 3.20: Comparison of average orientation error for all bodyparts during a highly dynamic motion

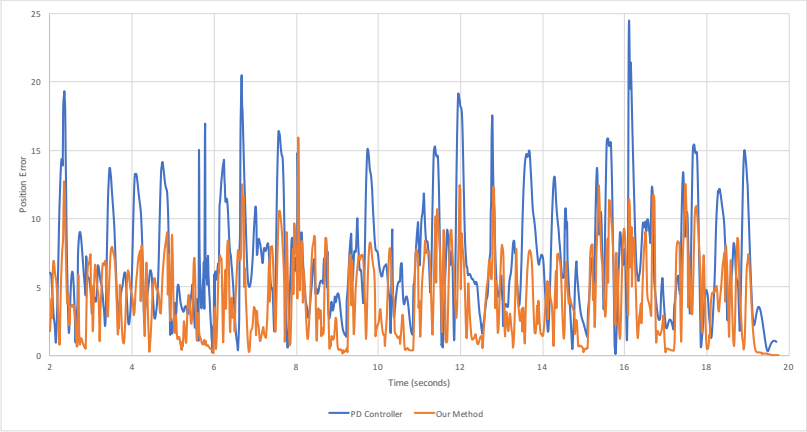


Figure 3.21: Comparison of orientation errors in degrees for right foot during a highly dynamic motion

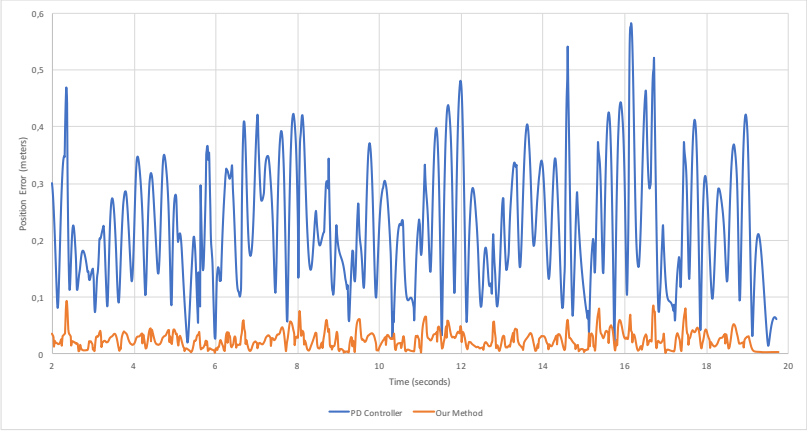


Figure 3.22: Comparison of local position errors for right foot during a highly dynamic motion

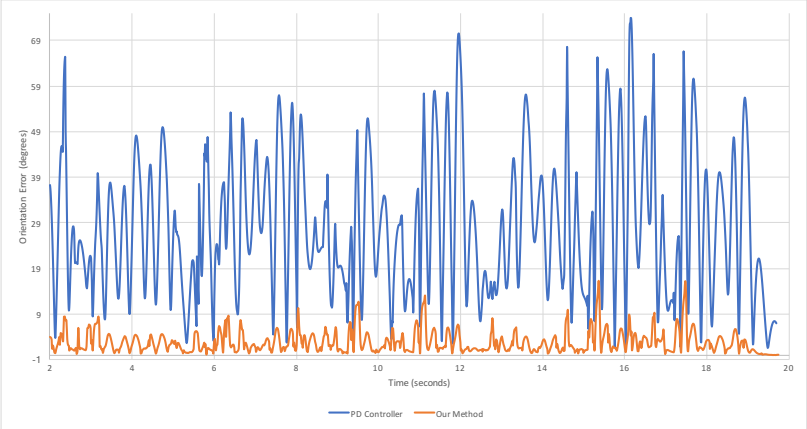


Figure 3.23: Comparison of orientation errors in degrees for right upper leg during a highly dynamic motion

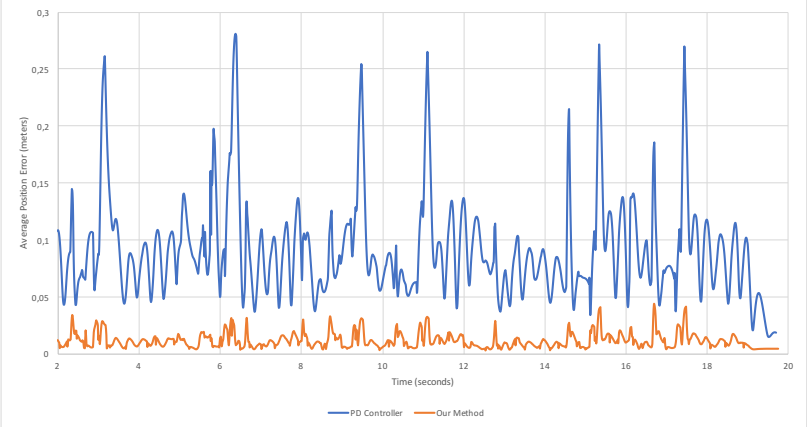
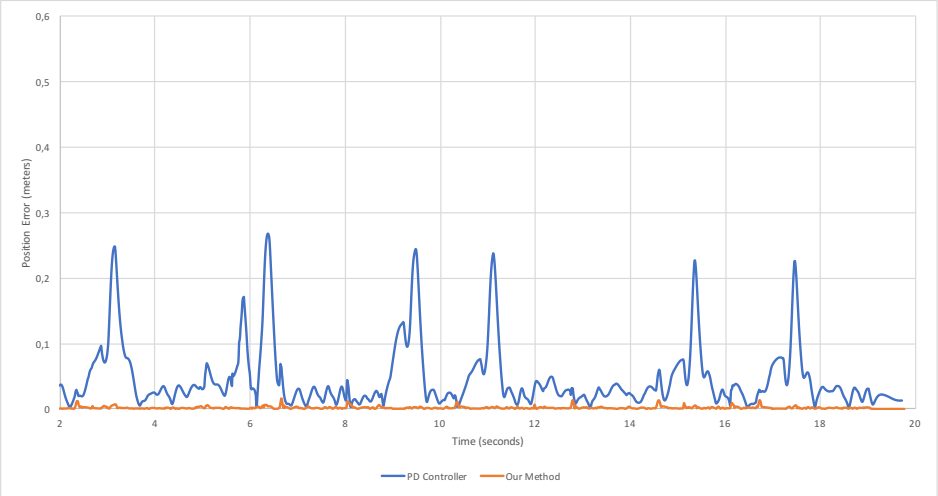


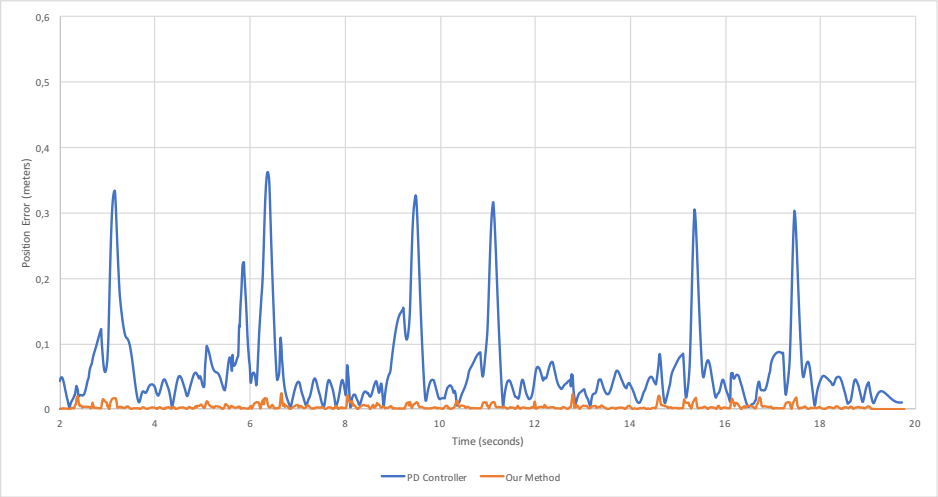
Figure 3.24: Comparison of average local position error for all bodyparts during a highly dynamic motion

	PD				Our			
	Orientation		Position		Orientation		Position	
	Avg. Error	Std. Dev.	Avg Error	Std. Dev	Avg. Error	Std. Dev.	Avg. Error	Std. Dev
Pelvis	14.57	12.58	-	-	0.99	0.95	-	-
Spine	6.91	6.22	0.0413	0.0425	1.23	1.30	0.0018	0.0018
Chest	5.89	4.56	0.0526	0.0542	1.42	1.87	0.0037	0.0035
Head	2.10	1.95	0.0684	0.0656	1.10	1.36	0.0080	0.0096
Right Upper Arm	5.38	4.13	0.0621	0.0596	2.25	2.15	0.0097	0.0064
Right Lower Arm	5.59	4.63	0.0735	0.0634	3.68	3.13	0.0123	0.0098
Right Hand	3.79	3.09	0.0796	0.0672	3.88	2.88	0.0177	0.0129
Left Upper Arm	5.84	5.39	0.0602	0.0606	2.68	2.33	0.0107	0.0078
Left Lower Arm	6.83	5.05	0.0725	0.0759	4.00	3.36	0.0140	0.0131
Left Hand	4.35	3.24	0.0777	0.0754	4.35	3.56	0.0201	0.0175
Right Upper Leg	29.14	16.60	0.0097	0.0052	2.79	2.40	0.0012	0.0012
Right Lower Leg	14.07	8.03	0.1805	0.1010	3.28	2.58	0.0163	0.0141
Right Foot	7.01	4.40	0.2323	0.1170	4.20	2.96	0.0239	0.0150
Left Upper Leg	25.82	16.38	0.0097	0.0052	2.56	2.23	0.0012	0.0012
Left Lower Leg	13.47	7.73	0.1611	0.0992	2.93	2.17	0.0145	0.0119
Left Foot	6.14	4.89	0.2092	0.1113	3.75	2.68	0.0232	0.0143

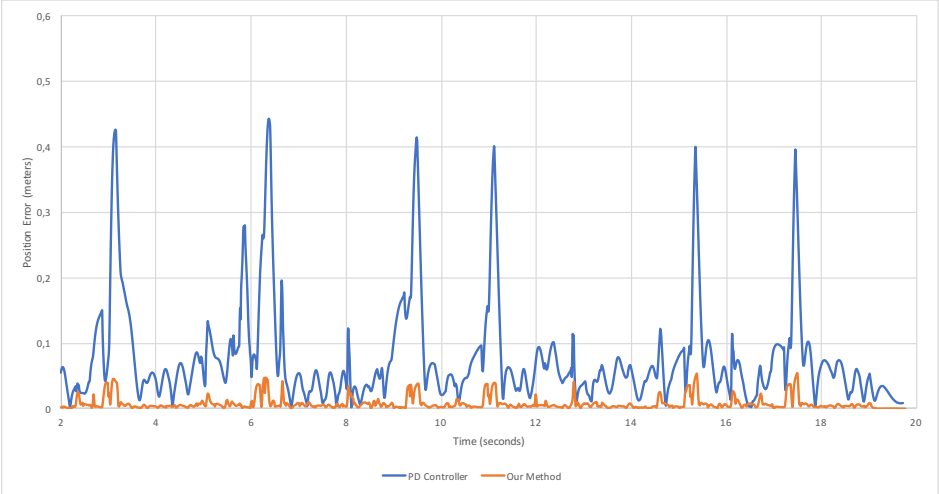
Table 3.3: Error comparisons



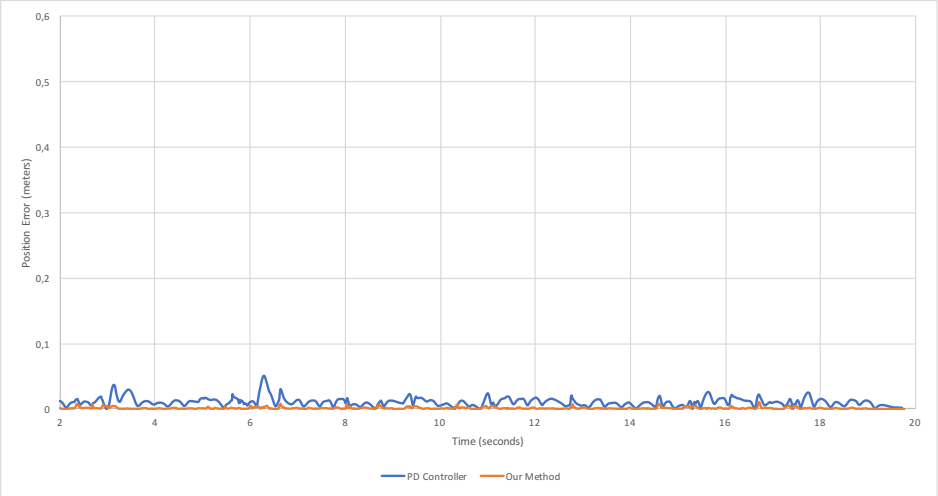
(a) Spine



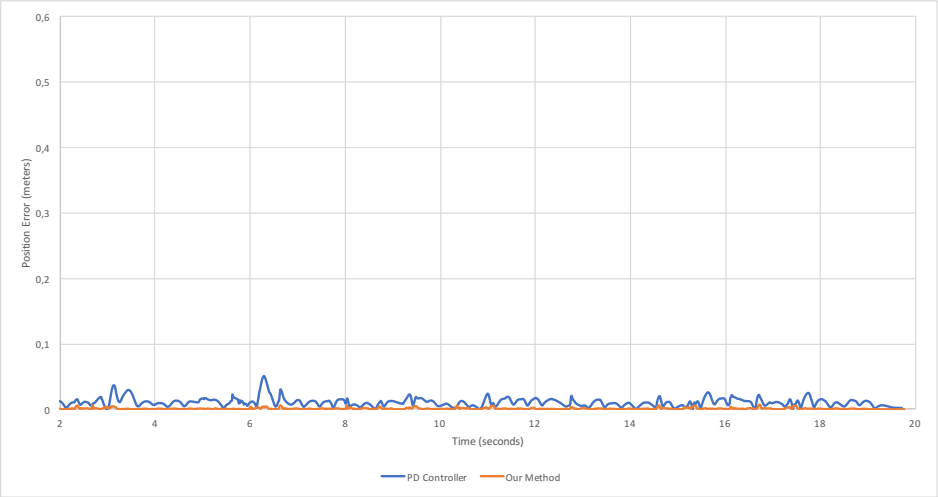
(b) Chest



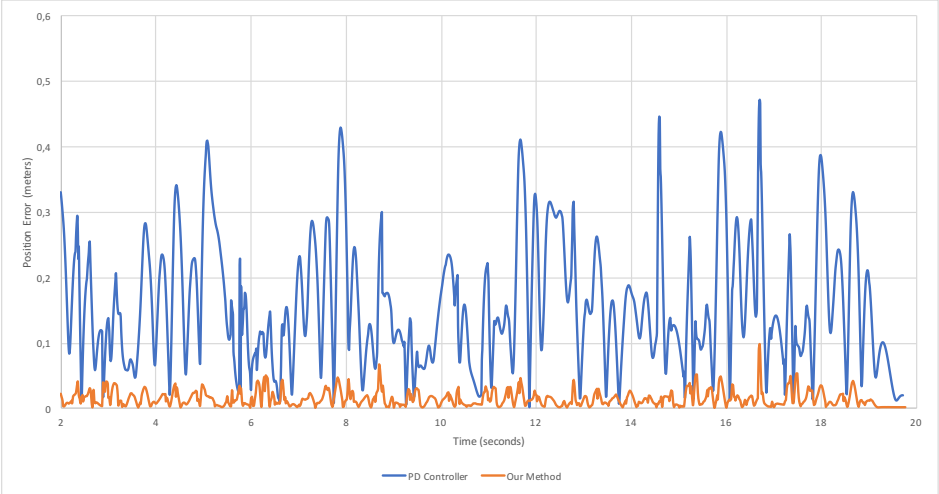
(c) Head



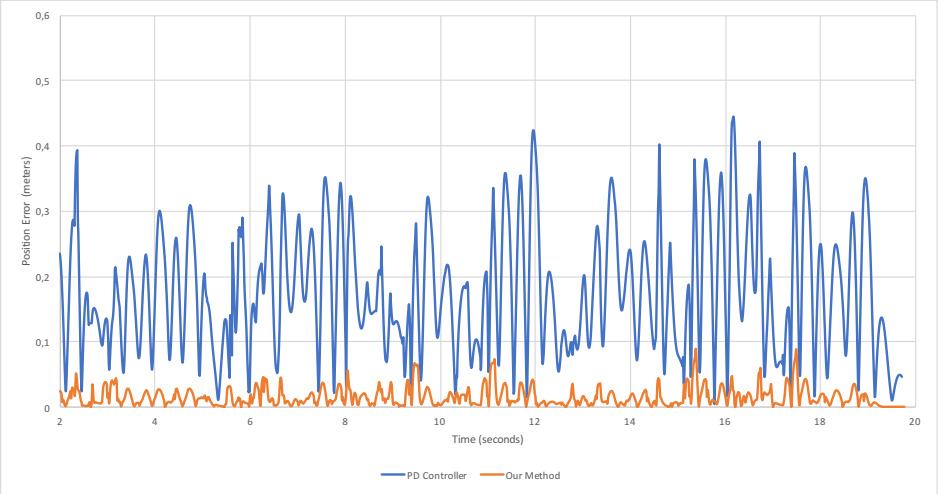
(d) Left Upper Leg



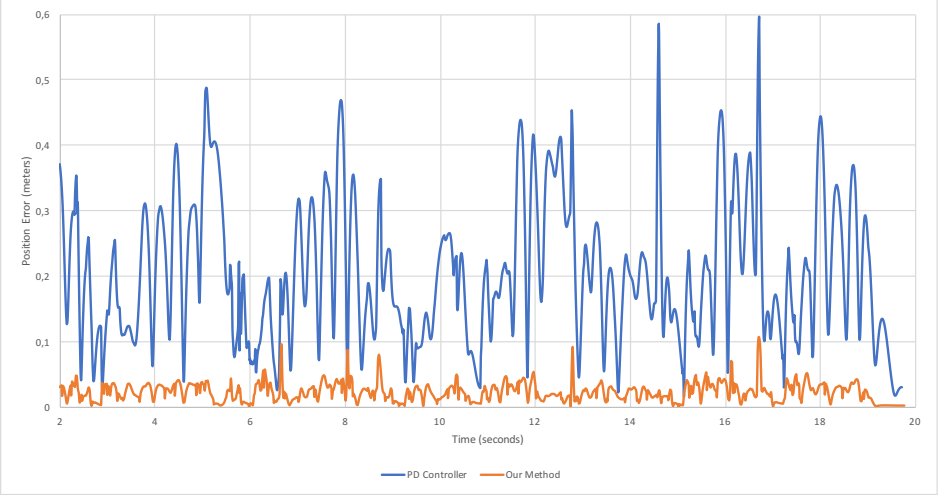
(e) Right Upper Leg



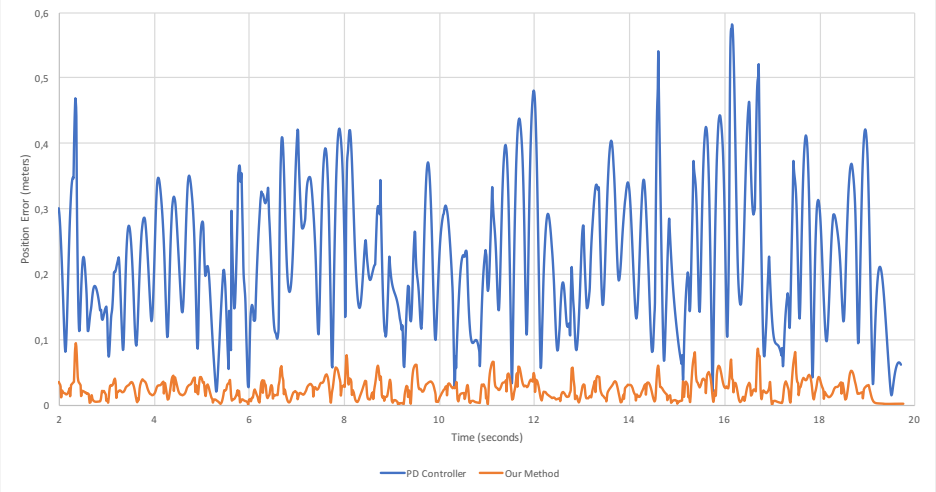
(f) Left Lower Leg



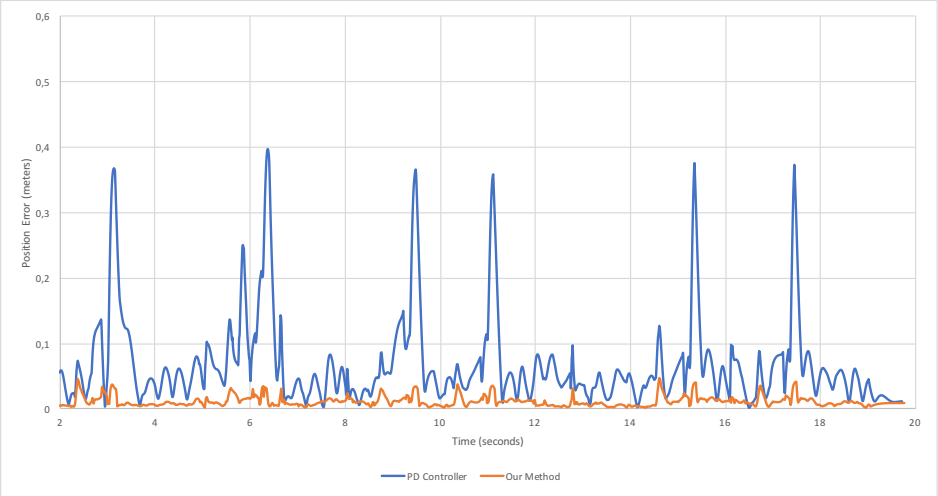
(g) Right Lower Leg



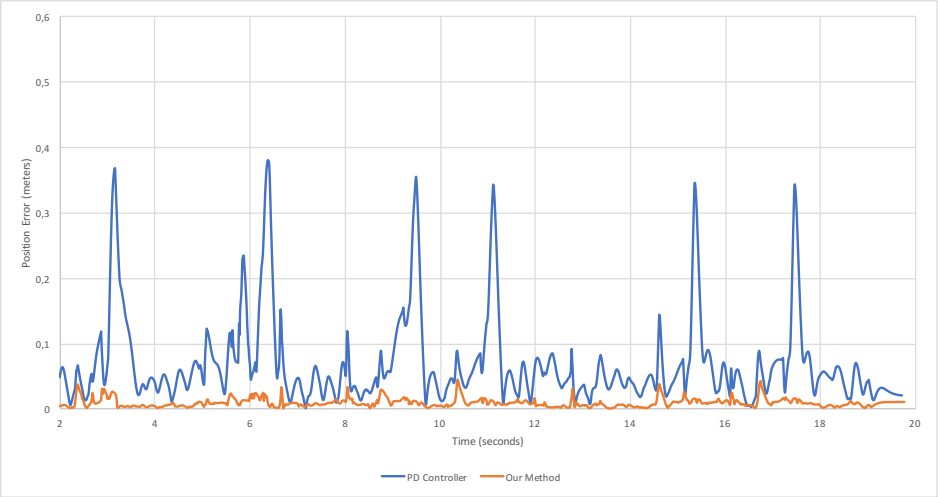
(h) Left Foot



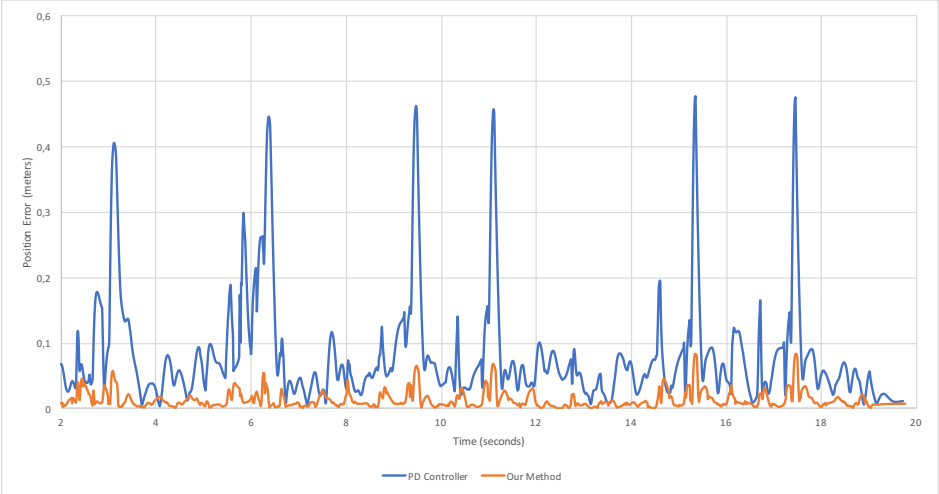
(i) Right Foot



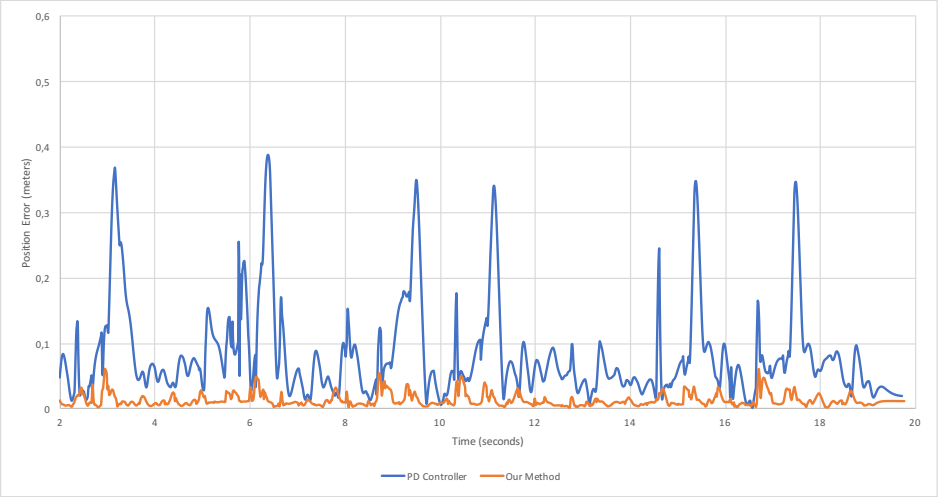
(j) Left Upper Arm



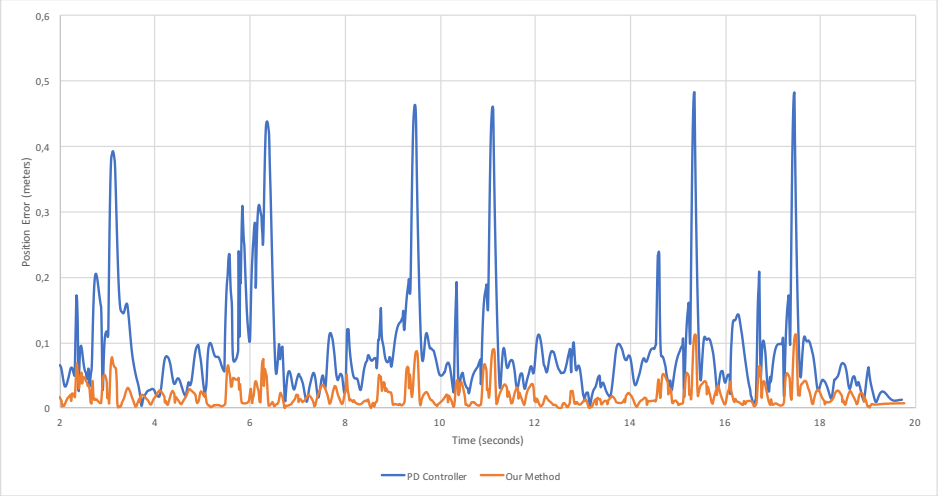
(k) Right Upper Arm



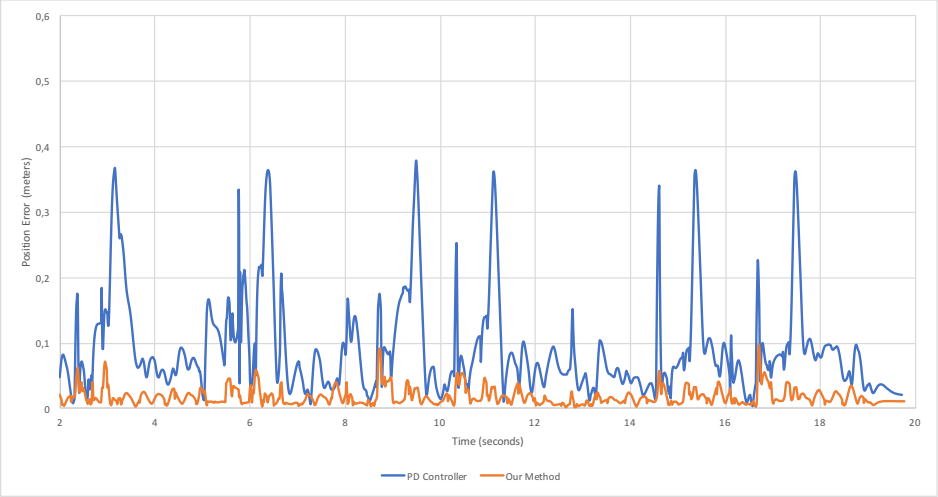
(l) Left Lower Arm



(m) Right Lower Arm



(n) Left Hand



(o) Right Hand

Figure 3.25: Positional errors for each rigid body

4 CONCLUSION

In this thesis study, we obtained analytic equations of motion using Kane's method, which lead to a generalized recursive inverse dynamics algorithm for multibody systems consisting of 3D rotational joints. We built a motion controller that can track highly dynamic motions in a stable way even with large simulation timesteps, which makes it quite suitable for real-time physics based animation applications. We evaluated the results of our system under various scenarios and compared the results with the results of various state of the art controller techniques.

Although, we show that the tracking results for a highly dynamic motion of a complicated humanoid multibody system are quite successful, our demonstrations don't have any strategy for underactuated control problem. It would be a reasonable future direction to include underactuated control to our proposed system, which is necessary for achieving high level goals like locomotion or balance. Ground reaction forces have a critical role for underactuated control. Our analytic equations and inverse dynamics algorithm are suitable for injecting external forces to any bodypart. Calculating and injecting ground reaction forces into our system can be considered as a future work to improve our proposed system. Another solution for underactuated control problem would be the ability of dynamically changing the root body part and rebuilding the hierarchy according to the new root.

Joint torque calculation is one of the crucial components of controllers but motion planning and motion generation are also equally important as well. Motion planning is usually handled as an optimization problem and our analytic equations are quite suitable for using in this kind of problems. We could employ our system in some of the widely worked motion planning problems like aerial or balance control but that is beyond the scope of this thesis. Also the angular momentum control is equally important for motion planning, motion generation and underactuated control.

REFERENCES

[1] Coros, S., Beaudoin, P., van de Panne, M., Generalized biped walking control, *ACM Transactions on Graphics*, 29(4), 1, **2010**.

[2] Boeing, A., Bräunl, T., Evaluation of real-time physics simulation systems, *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia - GRAPHITE '07*, 1(212), 281, **2007**.

[3] Kane, T., Wang, C., On the Derivation of Equations of Motion, *Journal of the Society for Industrial and Applied Mathematics*, 13(2), 487–492, **1965**.

[4] Mitiguy, P.C., Kane, T.R., Motion Variables Leading to Efficient Equations of Motion, *The International Journal of Robotics Research*, 15(5), 522–532, **1996**.

[5] Hodgins, J., Wooten, W., Brogan, D., O'Brien, J., Animating human athletics, *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM, **1995**, 71–78.

[6] Laszlo, J., van de Panne, M., Fiume, E., Interactive control for physically-based animation, *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, ACM Press, New York, New York, USA, **2000**, 201–208.

[7] Zordan, V., Hodgins, J., Motion capture-driven simulations that hit and react, *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, **2002**, 89–96.

[8] Abe, Y., Da Silva, M., Popović, J., Multiobjective control with frictional contacts, *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, **2007**, volume 1, 249–258.

[9] Macchietto, A., Zordan, V., Shelton, C., Momentum control for balance, *ACM Transactions on Graphics (TOG)*, 28(3), 80, **2009**.

[10] Han, D., Noh, J., Jin, X., S. Shin, J., Y. Shin, S., On-line real-time physics-based predictive motion control with balance recovery, *Computer Graphics Forum*, 33(2), 245–254, **2014**.

[11] KangKang Yin, Cline, M., Pai, D., Motion perturbation based on simple neuromotor control models, *11th Pacific Conference onComputer Graphics and Applications, 2003. Proceedings.*, IEEE Comput. Soc, **2003**, 445–449.

[12] Yin, K., Loken, K., van de Panne, M., Simbicon: Simple biped locomotion control, *ACM Transactions on Graphics (TOG)*, 26(3), 105, **2007**.

[13] Kawato, M., Furukawa, K., Suzuki, R., A hierarchical neural-network model for control and learning of voluntary movement, *Biological Cybernetics*, 57(3), 169–185, **1987**.

[14] Nakanishi, J., Schaal, S., Feedback error learning and nonlinear adaptive control, *Neural Networks*, 17(10), 1453–1465, **2004**.

[15] Da Silva, M., Abe, Y., Popović, J., Simulation of human motion data using short-horizon model-predictive control, *Computer Graphics Forum*, 27(2), 371–380, **2008**.

[16] Kwon, T., Hodgins, J., Control systems for human running using an inverted pendulum model and a reference motion capture sequence, *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, **2010**, 129–138.

[17] Lee, Y., Kim, S., Lee, J., Data-driven biped control, *ACM Transactions on Graphics*, 29(4), 1, **2010**.

[18] Wang, J.M., Fleet, D.J., Hertzmann, A., Optimizing walking controllers, *ACM Transactions on Graphics*, 28(5), 1, **2009**.

[19] Hansen, N., The cma evolution strategy: A comparing review, J. Lozano, P. Larrañaga, I. Inza, E. Bengoetxea (eds.), *Towards a New Evolutionary Computation*, Springer Berlin Heidelberg, volume 192 of *Studies in Fuzziness and Soft Computing*, 75–102, **2006**.

[20] Allen, B.F., Neff, M., Faloutsos, P., Analytic proportional-derivative control for precise and compliant motion, *2011 IEEE International Conference on Robotics and Automation*, IEEE, **2011**, 6039–6044.

[21] Jie Tan, Liu, K., Turk, G., Stable Proportional-Derivative Controllers, *IEEE Computer Graphics and Applications*, 31(4), 34–44, **2011**.

[22] Arıtan, S., Biyomekaniğin temel prensipleri.

[23] Featherstone, R., Orin, D., Robot dynamics: Equations and algorithms, *Proceedings-IEEE International Conference on Robotics and Automation*, 1, 826–834, **2000**.

[24] Luh, J.Y.S., Walker, M.W., Paul, R.P.C., On-Line Computational Scheme for Mechanical Manipulators, *Journal of Dynamic Systems, Measurement, and Control*, 102(2), 69, **1980**.

[25] Kurfess, T.R. (ed.), *Robotics and automation handbook*, CRC Press, Boca Raton, **2005**.

[26] Featherstone, R., *Robot dynamics algorithms*, volume 25, **1987**.

[27] Kane, T.R., Dynamics of Nonholonomic Systems, *Journal of Applied Mechanics*, 28(4), 574–578, **1961**.

[28] Yamaguchi, G.T., *Dynamic Modeling of Musculoskeletal Motion: A Vectorized Approach for Biomechanical Analysis in Three Dimensions*, **2005**.

[29] Tan, J., Liu, K., Turk, G., Stable proportional-derivative controllers, *IEEE Computer Graphics and Applications*, 31(4), 34–44, **2011**.

CURRICULUM VITAE

Credentials

Name, Surname : Ersan Kavafoğlu

Place of Birth : Ankara, Turkey

Marital Status : Married

E-mail : ersankavafoglu@gmail.com

Address : Konutkent mah. 2629 cad. Altınyol sit. 6/10,
Çankaya, Ankara, TURKEY

Education

BSc. : 2001-2007 Hacettepe University, Faculty of Science, Department of Mathematics

MSc. : 2012-2018 Hacettepe University, Institute of Informatics,
Computer Graphics

Foreign Languages

English

Work Experience

Software Developer	: 2007-2013	Ministry of Finance
Software Engineer	: 2014-2017	Cinar Engineering Consulting Co.
Co-Founder & Software Engineer	: 2017-	Cinar Software Inc.

Areas of Experience

Computer Animation, Physics Based Character Animation, Game Development, Software Development, Software Architecture, Enterprise Software

Publications

1. Kavafoğlu Z., İlhan H., Kavafoğlu E., Gürçay H., Çapın T. "Simple Vertical Human Climbing Control with End Effector State Machines", Proceedings of EURASIA GRAPHICS 2014, Paper 8, Hacettepe University Press, Ankara, Turkey, Oct 2014.

2. Cimen G., Kavafoğlu Z., Kavafoğlu E., Capin T., Gürçay H., "Skill learning based catching motion control", COMPUTER ANIMATION AND VIRTUAL WORLDS, Volume 26, Issue 3-4, pages 217-225, May-August 2015

3. Kavafoğlu Z., Kavafoğlu E., Egges A., "Robust Balance Shift Control with Posture Optimization", MIG '15 Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games, Pages 183-192, 2015

4. Kavafoğlu Z., Kavafoğlu E., Çimen G., Çapın T., Gürçay H., "Style-based Biped Walking Control", The Visual Computer 2016 (doi:10.1007/s00371-016-1338-5)

5. Kavafoğlu Z., Kavafoğlu E., Egges A., "Robust Standing Control with Posture Optimization", Computer Animation and Virtual Worlds 2017 (doi:10.1002/cav.1746)