Real-time Physics Based Character Control carrying Load

Bharat Vyas, B.Tech.

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Augmented & Virtual Reality)

Supervisor: Carol O'Sullivan

August 2020

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Bharat Vyas

31st August, 2020

Permission to Lend and/or Copy

Acknowledgments

I would like to thank my thesis supervisor, Carol O'Sullivan, for her invaluable feedback and encouragement from the beginning to the end of this project.
In addition, I would like to thank my family and friends for their continuous support during my studies.

Bharat Vyas

University of Dublin, Trinity College
August 2020

Real-time Physics Based Character Control carrying Load

Bharat Vyas, Master of Science in Computer Science
University of Dublin, Trinity College, 2020

Supervisor: Carol O'Sullivan

The goal of physics-based computer animation is to create virtual characters that automatically synthesise realistic human motion. Simulation of a dynamic articulated body is an extremely difficult task, especially when considering lifelike motor skills to control gravity and contact forces. In this project, we demonstrate a pose-based control in which a physical articulated model follows the key frames of animation sequence while maintaining physical interactivity with its virtual environment. This avoids the need to create motor skills of character for basic tasks like balancing or walking. Also, the displacement of center of mass in human body is replicated in virtual character while carrying a load. Along with this, various other techniques like a root balance method and optimization approach have been combined together to make character behave more naturally. The motion generated from this implementation naturally adapt according to user inputs, adjust the posture and joint configuration on the basis of its environment conditions.

Contents

List of Figures

Chapter 1

Introduction

Physically-based animation intends to simulate a natural behaviour of virtual objects in an environment where these objects interact with each other and with external forces such as gravity, pressure, etc.
Recently, it has caught the attention of many computer game programmers and researchers. In computer games, physics-based animation has been used in the past for creating outstandingly realistic deaths with the use of ragdoll, or even simulations of cloth and hair. The motion of articulated structures is usually key-framed to obtain the animation. This technique of key-framing is expensive in terms of modelling, as in the past, the artist would have to design every single frame. With physics-based animation, this work can be done by applying the physical laws of the real world. Moreover, unfeasible but realistic looking motions can be computed from feasible input motion data. Other applications can be in the field of robotics, real-time simulations of medical surgeries, virtual reality, etc.

1.1 Motivation

Realistic human motion or, in fact, any realistic simulation of 3D character which involves some kind of interaction with external forces, has been a subject of interest recently. Its applications open up to a wide range of possibilities such as computer game productions, real-time simulations for scientific research, real-time control of a character, virtual reality. In recent years a significant body of work has been developed for dealing with the physics-based simulation of rigid and soft bodies. This physics based simulation is often touted as most suitable method to automatically create realistic motion. Yet this simulation technique is underutilized in real-time virtual environment. Most of the game engines are integrated with physics engine to simulate real world physics with some approximation. As mentioned before, key-framing is comparatively an inefficient method to create animation as it requires designing of almost every single frame. There are various human tasks which can be animated using physiology which can give more satisfactory results, for example lifting a load. Both upper and lower body posture changes when a character carry any extra load. This change in posture also depend upon the weight of load carried. Animating such tasks can be very repetitive and time consuming. Also, the data for such tasks will be very large as there will be different motion trajectories of points and their corresponding geometries for slight change in load.

1.2 Objectives

The main objective of this project is to create a virtual character that can physically interact to its environment and carry a load defined by user. The character must be able to behave more naturally and adapt according to any change in virtual environment. As the character will be carrying load so it should naturally adapt its posture and joint configuration according to the change in mass of load. Unity game engine is used for all the rigid body simulation and it allows to see results in real-time. As the character is developed in real-time environment, it is important for character to be interactive. A framework is used such that a user can control the walking and some basic tasks of character, while also changing few of the parameters of load and its environment.

1.3 Summary

This dissertation follows the similar pattern as the work flow used to develop the implementation. It is split up into different chapters with each chapter dealing with either a large area of the project or the results and conclusions of the outcome. Chapters 2 detail the introduction along with background research that was done both prior and during the implementation of the simulation. Chapter 3 consist of some relevant theory and concepts which are used in implementation. Chapter 4 deals with the setup of virtual character used in the implementation. This section also includes the process for creating an active ragdoll. Chapter 5 focuses on the implementation of some techniques and methods which makes the character behave physically real. Chapter 6 discusses the results and any observations made during the course of this project. It also highlights some of the limitation of work done in this project. Chapter 7 is the final chapter that concludes this work with details of contribution and any possible future work as an extension of this project.

Chapter 2

Related Work

In this section, several approaches that belong to different aspects of physics based animation are discussed. The following approaches are most popular solutions and relevant to the objectives of this project. It also includes an extensive description of the examined papers which resulted in techniques used for implementation.

2.1 Physics-based Character Animation

Despite the impressive history of progress in creating and animation virtual humans, physics-based virtual characters with realistic motion have not been developed at same pace. In [1], the process of creating virtual humans in computer graphics was discussed decades ago. It is clear that simulation and animation of humanoid characters is a challenging task in many ways. Ever since then, many approaches have been developed and used for plausible physical motion. The simulation techniques are used to synthesise physical motion for various tasks such as walking, running, jumping etc [2]. The idea is to compute some motion trajectories for such locomotion tasks and use simplified physics constraints. A similar approach of simplified constraints is used in [3] for balancing and cyclic walking. However, such constraints are limited to specific tasks and usually not applicable to new tasks or require excessive modification according to new task specification.

The equations of motion are used to formulate the balancing constraints for full body motion computation. In such a case, it is important to maintain the equilibrium

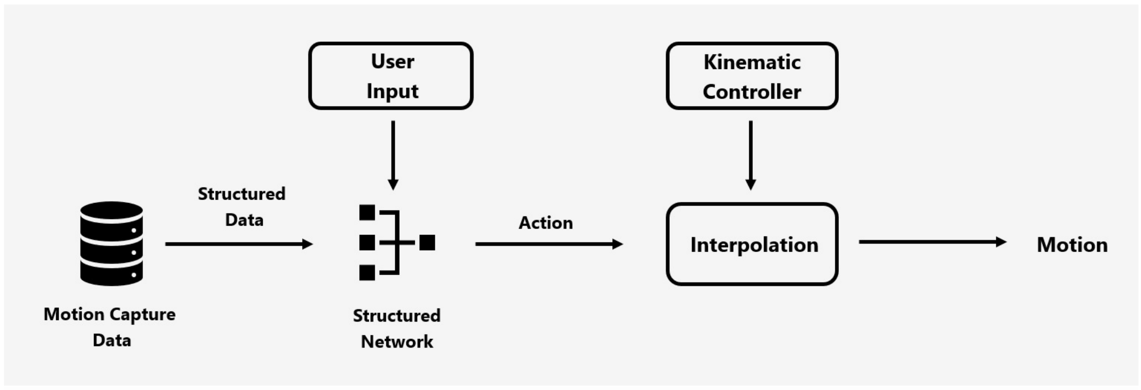

Figure 2.1: Data-driven Animation

among internal and external forces exerted on the character. These balancing constraints are handled either in motion optimization equations or by considering different contact forces and sampling them on large motion data. The optimization approach to simulate various tasks is discussed in [4] by satisfying different balancing constraints. It is an active area of research in other fields like robotics, bio-mechanics and neuroscience to simulate human motion and behaviour in virtual environment. In many ways, the proper definition of plausible physics-based characters have applications in various subjects. There are many different ways to achieve physics-based character animation as discussed in [5], but the two main style of it are: Data-driven and Dynamics-driven. These are discussed below in detail.

2.1.1 Data-driven Animation

Data-driven animation approach rely heavily on collection of motion captured data. Fig. 2.1 shows a basic outline of Data-driven animation technique. The motion capture or key frame data is first structured and then sent to a network. This network takes input from user and implement any approach to synthesise the data. This further go for interpolation which interpolates between last and current frames. An optional kinematic controller can be used to control character based on kinematics principle. Then, a final motion is obtained and applied for animating the virtual character. This simulation method is very intuitive, easy to use and give very good results in terms of realism. One major drawback of such technique is that it is very costly in terms of space and time. Also in order to capture large motion data, a large number of test has to be performed to avoid any extra costs. For this reason, such animation technique can be useful when it is combined with dynamics, which will decrease the costs and save from repetitive work of key-frame animation.

There are various approaches developed to use large collection of these mo-cap dataset and process them further into animation pipeline. As an example, [6] use some particle filters constrained by human physiology and identify different part of the human body. This data then act as reference for animation sequence, and character simulates the exact same pose and action by tracking the trajectories of motion data. In a similar manner, [7] combines the tracking with a number of controllers that enables the character to interact with its environment and react to different unpredictable situations. This helps character to maintain balance and react to changes in more natural manner.

Another approach that falls into this category is to use the motion captured data and generate totally new animation sequence, also known as motion blending. [8] describe such a framework in which the constraint limit is detected automatically. The different constraints computed for the character are determined after analysis of the input motion data. A small set of linear and angular momentum constraints is applied on character model to achieve much realistic motion. The use of input motion data here helps to avoid the complex computation of joint forces and torques.

There have been various attempts to use motion data and solve the problem of requirement of more data for similar actions. As an example, [9] divide the data into different transitions between some predefined states. For each state, some set of pre-conditions are defined which enables to actually find the state of model. Then, the next state is predicted from possible available states on the character model automatically based on a Support Vector Machine (SVM) learning theory. The demonstrated controller works in conjunction with the joints and links of virtual character and allows the easy integration of dynamics into a framework. This is a great way to allow users combine the kinematic animation with the physics-based animation.

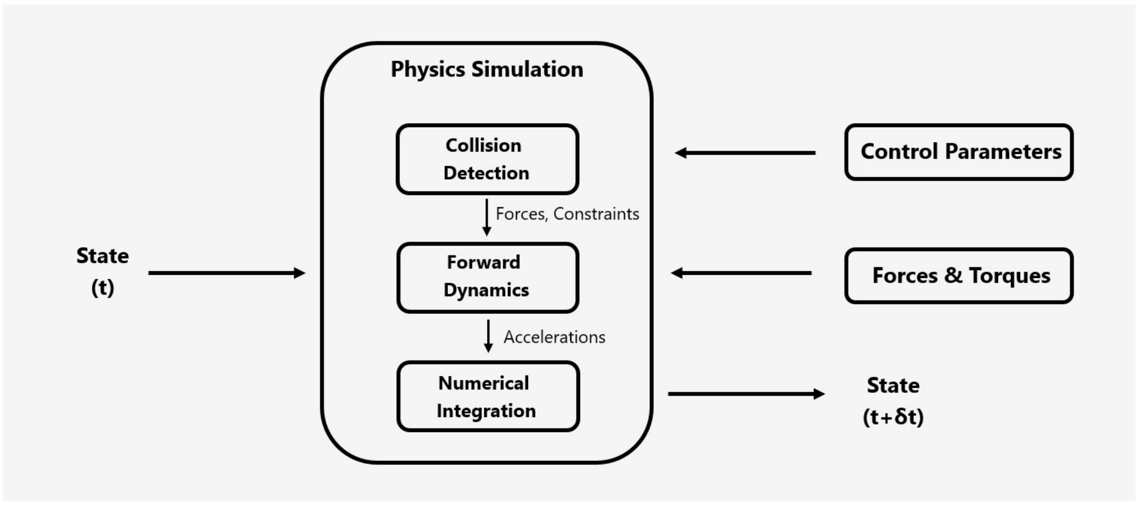

Figure 2.2: Dynamics-driven Animation

2.1.2 Dynamics-driven Animation

The dynamics-driven animation technique only considers the joint actuation value, forces and toques to move an object from one position to another. Fig. 2.2 show a basic outline of dynamics driven animation approach. The motion at a particular state go through a physics simulation cycle which considers input of some control parameters, forces and torques. The physics simulation then forward the data to next state and this next state become the current state and act as input.

There are various instances in which an input motion is combined with a dynamic response so that the character can interact with its environment based on physics laws. [10] uses a similar approach where given an input motion, it simulates a plausible physical response when any unpredictable environment variable influences the character. It is called "Quasi-physical control" as it only comes into action when needed, otherwise the character is controlled by motion data and other balance controller.

[11] defines the character motion as constraint optimization problem. The constraints used are called "Spacetime Constraints" and are required together with Newton's laws to create character animation. The spacetime method discussed is capable of producing basic animation attributes such as squash-and-stretch, anticipation etc.

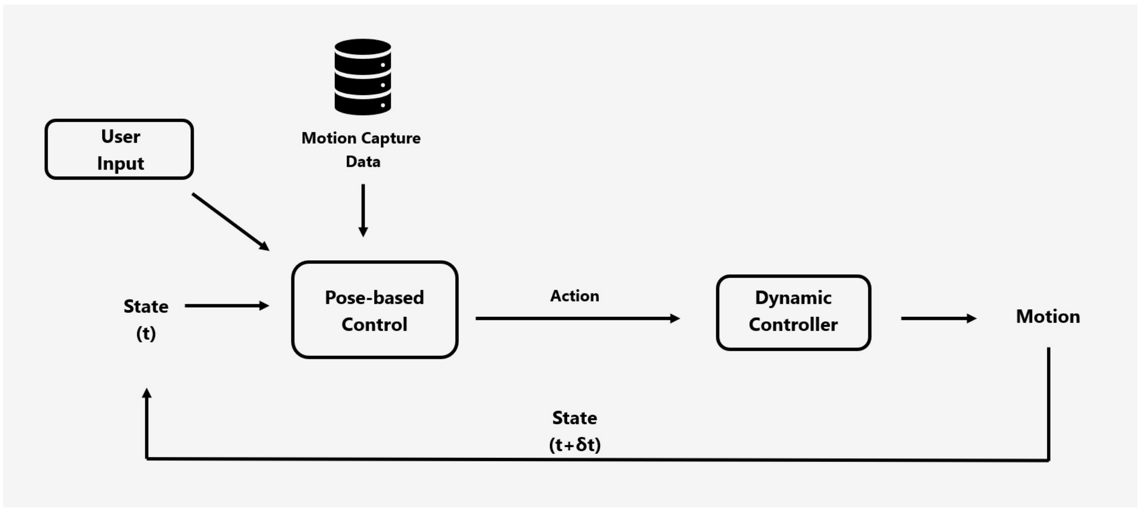

Figure 2.3: Pose-based Animation

without defining any extra constraints. By applying only a few kinematic constraints, it developed realistic and complex motion which are physically valid. Equations for kinetic energy, muscle joints and pose equality constraints make the spacetime method a breakthrough in dynamics animation approach.

Dynamics-driven approach is very well researched in bio-mechanical simulation. In [12], a biomechanics based human model is constructed and simulated by applying dynamic equations of motion. The model is then synthesised to obtain realistic swimming animation. A specific learning approach is used to synthesis muscles of desired length and then a first order damping approach is applied for muscle control. A well defined locomotion controller is designed, which after learning a few parameters automatically generates muscle contraction variables that further enable human model to perform swimming motion. This approach can be used to obtain various different motions by changing the locomotion controller design.

2.2 Pose-based Animation

The pose-based animation approach originates from the theory of inverse dynamics. It attempts to control the character skeleton by defining kinematic poses or say targets for all the body joints, and then use the feedback loop of control system to calculate the joint torques. Most of the work discussed in this subsection uses a similar mechanism as shown in fig. 2.3. The current state take motion capture data and user input in controller, where motion data provide a reference position for each articulated joint. The joint dynamic values are calculated in this pose-based controller. Then next, it goes into a dynamic controller which controls all the physics simulation of system. And finally the next pose is generated which then again go through same process.

This approach can be seen as a hybrid strategy to take the features of both data-driven and dynamics-driven approach. In [13], a method is introduced which allow the characters to physically interact with environment and make them react to any unpredictable changes. Even after the impact, this method determines the next pose with the help of contact forces and a search routine. This system produces realistic body motion for martial-arts under varying external conditions.

[14] introduces a technique which generates new motion from the existing original motion on the basis of user input. At first, the model is set to its original motion and its parameters are taken into consideration. Then the method generate a different pose by deforming the model. This model deformation is done with joint rotation, inverse kinematics and position of center of mass.

A simulation platform is developed by [15] named "DANCE" for physically based character controllers. The main feature of this system is that it provided a platform for flexible use of controllers in articulated human models. A user can develop a dynamic controller using this platform, and combine it with the parameters from kinematic controller. It also allow user to introduce spring forces in joints using drag and spring actuators. This framework is then improved in [16], where it is presented as a toolkit for animators. It is modified to support both visual and scripting tools with much reduced dimensional physics. The interactive control of characters can be done through key-framed based control or combining dynamics with the other control.

An interactive framework is introduced by [17] which allows editing of momentum and external forces in an animation sequence. The approach takes the input motion, analyze it and differentiate it for the either momentum or force editing. The editing is done with user defined constraints and then trajectory optimization is done. Finally, the synthesised motion is generated based on desired pose, new center of mass trajectory and new momentum profile.

A unique and interesting technique is introduced by [18] that allows a virtual humanoid character to set and maintain its pose in dynamic virtual world. The idea is to

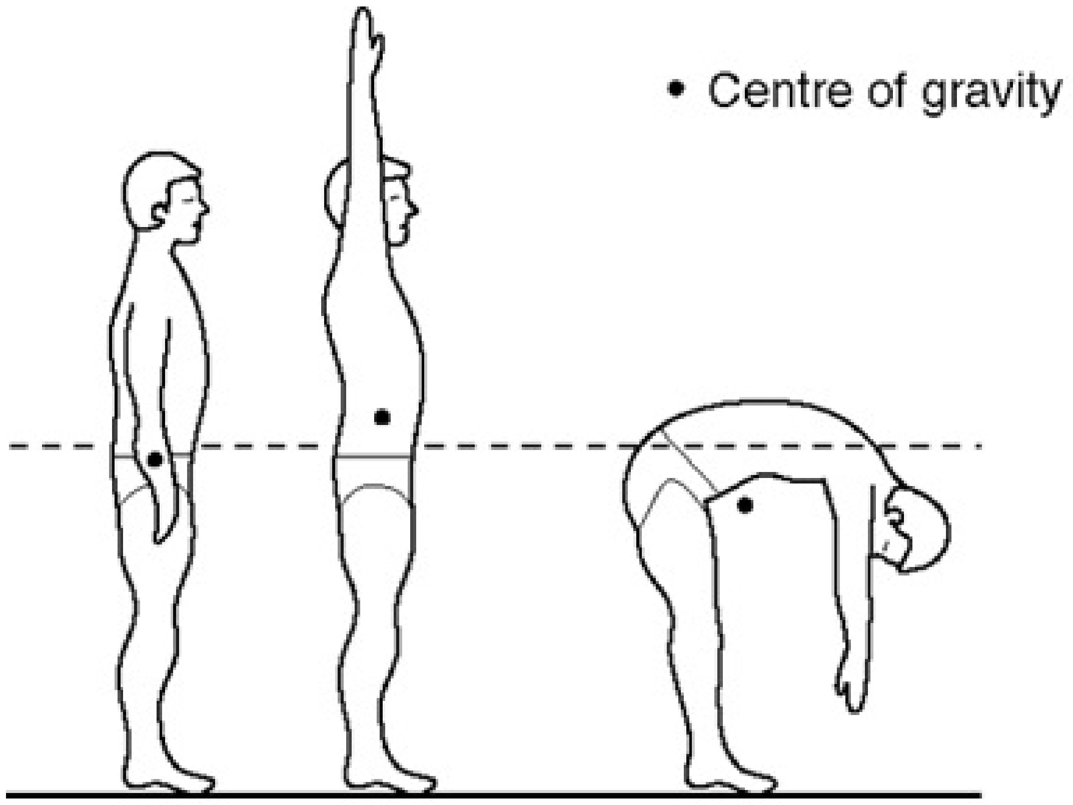

Figure 2.4: Center of Gravity of Human Body

maintain the pose of character by applying a weak root spring force to it. This weak spring allow the character to maintain its pose under weak external forces. But as soon as the external forces are greater than the threshold values, the weak root spring breaks. This approach generate plausible balance in the character body by computing world-space torques. This increased stability allow the dynamic character to react more realistically in virtual environment.

2.3 Center of Gravity in Human Body

Human body is indeed a complex system and animating it perfectly is the most challenging task in computer animation. Creating a physics based character means replicating a human body in virtual world. There are various parameters that can be considered in the task of carrying a weight. One of them is center of gravity, which is equivalent to center of mass under constant gravitational force. From here on, the center of gravity is referred as center of mass to avoid any confusion in discussion. Various practical tests were conducted by [19] on human body to compute the precise position of center of mass. It is discussed in detail how small changes in limb position can alter the position of COM. The experiments were performed on human body and results were very similar to position as shown in fig. 2.4. In normal standing position with both hands sideways, the center of mass is in the middle line of body around pelvis. This point shifts upward if both arms are lifted up and the point shift downward if the body is bend down as shown. The experiment results also showed that the center of mass can lie outside the body depending on the distribution of mass in a particularly specific body posture.

The center of gravity point in living human body is also determined by [19] when body is loaded with weight. The results showed that position of extra load on body affect the new center of mass position. Also, the position of center of mass play an important role in maintaining stability of whole body which is discussed in [20]. As shown in fig. 2.5, when a load is carried the new center of mass shift in the direction of extra load. If the position of this load is moved further away, the center of mass also shift away from body. But the contact support of human body lie in between our feet. When the line of gravity does not match with base of support, the body can get unstable. This is also clear from the experiment results where the upper body of officer curved in the middle, which adjust the center of mass position to coincide with the central line of support.

2.4 Lifting a Box

The physical simulation of lifting task is widely researched in robotics. It is often required to simulate the optimization constraints before applying them on a physical robot. Such an optimization-based method is introduced by [21] for predicting the dynamics in lifting. The motion is generated by minimizing an objective function which already has few kinematic and dynamic constraints. As many as four objective functions are formulated which resulted in different lifting strategies.

A high-level task control is formulated in [22] to control complex characters in real-

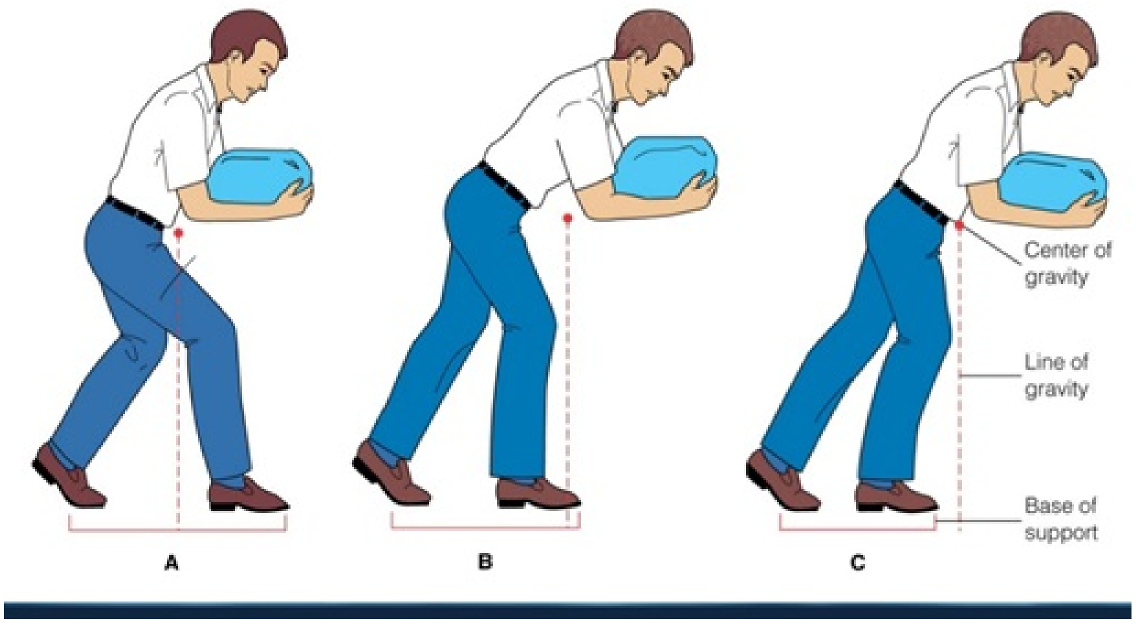

Figure 2.5: Lifting a box

istic and interactive environment. In the control algorithm, joint torques are computed so that the character can achieve desired manipulation. An interesting approach of open and closed loop is implemented which depends upon the configuration of upper body limbs. The inactivate joints are moved according to the manipulations of actuated joints which is computed through multi-task control. The task-space forces are computed by the force limits which are placed in manipulation for tracking and stabilizing.

Chapter 3

Relevant Concepts

3.1 Physics Simulation

The continuous development in simulation physics allowed much precision in simulated environments. The physics engine is a software which introduces the real world physics laws into a simulated environment. Typically, simulation physics is a close approximation to actual physics, and computation is performed using discrete values. A physics engine is an integral part of game development as it produces results in real-time. In most game engines, the aim to have "perpetually correct" approximation rather than a real simulation. There are various methods integrated to make the virtual environment behave as real world such as collision detection, rigid body dynamics, soft body dynamics, particle motion etc. Some of the relevant concepts are discussed below [23]:

3.1.1 Collision Detection

Collision detection is a computational problem that gives solution by detecting the collision between two or more 3D objects. In simulation, this algorithm is called for each time step. Collision response is computed by the information provided, which further computes the motion of the objects involved in collision. The information used for the algorithm include:

- Time of Impact:

This is actually the exact time step in which the detection has already happened. As collision detection is run for each time step and the moment has already passed so it is not provided normally. An approximation of time of impact via linear interpolation or any other method is taken for simulation. The time of impact may not be accurate in this case, as time steps are a division of the timeline that the algorithm requires but there is usually nothing in real-time in simulation applications.

- Velocity:
  The velocity of each body involved in the collision is returned at the moment of impact. The current position and previous position is taken into account and velocity is calculated as follows:

  $$v_k = \frac{q_k + q_{k-1}}{\Delta t}$$

  where q_k is the vector of angle of the joints of object at the time step k. This equation is further translated for real-time simulation as follows:

  $$v_{k+1} = v_k + \Delta t a_k$$

- Acceleration:
  The acceleration of the joints of each object depends upon the current, previous and next pose as follows:

  $$a_k = \frac{q_{k+1} - 2q_k + q_{k-1}}{\Delta t^2}$$

  This is further translated in simulation as:

  $$q_{k+1} = q_k + \Delta t v_{k+1}$$

For the purpose of efficiency, a simplified mesh is used which is usually invisible. This simplified mesh encapsulate the actual complex object. In this way, only the boundaries of simplified mesh are considered during the collision. The Unity game engine used in this project has *colliders* in basic shapes for collision detection.

3.1.2 Rigid Body Dynamics

Dynamics is a branch of physics which deals with the effect of forces on motion of a body. The physics simulation mainly deals with two types namely rigid body and soft body simulation. As in this project, only rigid body simulation is used so soft body is not discussed further. A rigid body is a solid body which has zero or negligible deformation. In other words, the distance between any two given points on a body remains constant irrespective of any external force applied to it. A rigid body is similar to a particle system which has mass and can be defined by its position but a rigid body also has orientation. Imagining a rigid body made up of large no. of particles, the position of the ith particle in world space at time t is:

$$r_i(t) = R(t)r_{0i} + x(t) \tag{3.1}$$

where R(t) is the orientation matrix, r_{0i} is constant location of ith particle in local space and x(t) is the position of rigid body in world space.

The velocity $\dot{r}_i$(t) of the ith particle is obtained by differentiating equation 3.1

$$\dot{r}_i(t) = \omega(t) * (r_i(t) - x(t)) + v(t) \tag{3.2}$$

where ω is the angular velocity of centre of mass of body which is discussed later. Also, here velocity has two components: a linear component and an angular component.

3.1.3 Numerical Integration

The initial value problem along with an initial condition specifies the value for an unknown function and its derivatives. In other words, it is to solve the differential equations. Such systems of Ordinary Differential Equation (ODE) are described as:

$$\dot{x} = f(x, t)$$

where x is the state of the system and $\dot{x}$ is its time derivative. The problem is described by giving the ODE and an initial value of the unknown function at a given time t_0. The solution of this problem specifies the evolution of the system with time. The choice of integration technique and size of time step δt are crucial for the numeric stability and performance of the simulation. Typically, a larger time step will decrease simulation stability (caused by accumulation of integration errors), while a smaller time step will decrease simulation performance. Several methods have been developed to allow greater robustness at larger time steps, such as the Runge-Kutta methods.

3.2 Degree of Freedom (DOF)

The degree of freedom is a parameter which defines the state of any system in a physical manner. The combination of different states of such physical system is called its phase space, and the dimensions of this phase space is its DOF. [24].
As the location of any point in 3D space is described by three different position coordinates. In the same way, velocity components of a body can be defined in 3D space to describe its direction and speed of movement. So, there are three components for position and other three for orientation which defines the degree of freedom of any body. In general, every body or particle has six degrees of freedom which can be altered depending upon the constraints. This allows to restrict the motion of a body in a particular dimension - for instance, if a body is restricted to rotate along an axis then it lowers the degree of freedom of this body. Also, when two or more bodies are joined together then overall degree of freedom increases.
The degrees of freedom in robotics defines the amount of movable joints that can be used to achieve a particular orientation or position. Greater the DOF of any articulated structure, greater is its ability to interact with its environment. The concept of DOF is inspired by human body and is often replicated in robots. A similar approach in DOF can be used for rigid body characters in animation. The total DOF of any character would be the total amount of all the DOFs of every moving joint of the articulated body.

3.3 Joints

In physics simulation, joint is a component which connects one rigid body to another rigid body or any fixed point in space. Joints apply forces that move rigid bodies along the direction of force applied, and joint limits restrict that movement which can be specified for different conditions. In a rigid body, existing 6 degrees of freedom is limited depending on the type of joints.

- Fixed Joint: It is a joint with no DOFs. Once a fixed joint is created, it restricts any motion in all dimensions of connected body in reference to each other.

- Prismatic Joint: It is a joint with one DOF that allow a linear movement between connected bodies. specified axis.

- Revolute Joint: It is also a joint one DOF which act aas a hinge joint. It allows the rotation of one body w.r.t another in one specified axis only.

- Spherical Joint: It is a joint with three DOFs which allows the freedom of orientation in any axis.

- Planar Joint: It is also a joint with three DOFs which allows the translation motion in any of three axis.

In this project, different joints are used for the human skeleton system. The unity game engine defines its joints differently. The configurable joints emulate like any skeletal joint very similar to its character joints. This joint can be configured to force and restrict rigid body movement in any degree of freedom.

3.4 PD Control

PID control is by far the most common way of using feedback in natural and man-made systems. PID controllers are commonly used in industry and a large factory may have thousands of them, in instruments and laboratory equipment. In engineering applications the controllers appear in many different forms: as a stand alone controller, as part of hierarchical, distributed control systems, or built into embedded components.

The P, I and D terms can be used independently or in combination with one another as suited for work [25]. A Proportional-Derivative (PD) control is a control loop mechanism having feedback loop which is widely used in applications requiring continuous control. A PD controller continuously calculates an error value e(t) as the difference between a desired goal and a measured variable and applies a correction based on proportional and derivative terms.
The mathematical form of overall control function:

$$u(t) = \kappa_p e(t) + \kappa_d \frac{de(t)}{dt}$$

where κ_p, κ_d, non-negative, denote the coefficients for proportional and derivative terms respectively.

The proportional tuning involves correcting a target proportional to the difference. Thus, the target value is never achieved because as the difference approaches zero, so too does the applied correction. Derivative tuning attempts to minimize this overshoot by slowing the correction factor applied as the target is approached.

3.5 Centre of Mass

A center of mass is a point defined for any object or a system of objects. It is the position at which the average position of all parts of the system are positioned, affected by their masses. A rigid body can be considered to be a collection of particles and the weighted average position of combination of particles is COM of this rigid body. In general, a rigid body is considered to be positioned at COM and it rotates around COM point.
The center of mass is the mid-point of the mass distribution of a body. If a rigid body with mass M is made up of N tiny particles, each with mass m_i and location r_i inside the body, the center of mass can be computed as:

$$\frac{\sum m_i r_i}{M}$$

This formula shows that the center of mass is the average of the particle positions weighted by their mass. If the density of the body is uniform throughout, the center of mass is the same as the geometric center of the body shape, also known as the centroid. Game physics engines usually only support uniform density, so the geometric center can be used as the center of mass.

Chapter 4

Character Setup

Characters in a simulated physics-based environment have attributes which are not required and incorporated in kinematics-based characters. These characters have mass properties and joint constraints which are affected by dynamic simulation. Also, the forces and torques act as actuator to control the motion of rigid bodies and their connected bodies. This section discuss in detail the system and procedure for setup of humanoid character used in this project.

4.1 Model Definition

The model definition consist of designing the skeletal structure and shape of humanoid body using Unity. The mesh for humanoid character is taken from *Mixamo* which has an online animation store featuring 3D models and animation sequences. This mesh comes in a predefined rigged format which is used in this project.

The 3D model used here has only 34 DOFs and 14 joints, whereas the real human body has 244 degrees of freedom and around 230 joints. The model consists of six physical branches and one virtual branch. The physical branches include the right leg, the left leg, the spine, the right arm, the left arm and the head. Each DOF (the z's) represents relative rotation/translation of two body segments connected by a revolute/prismatic joint. Since, unity engine has different definition of its joints, configurable joints are used to connect two body segments.

In most of the physically based characters, a set of rigid bodies are joined together

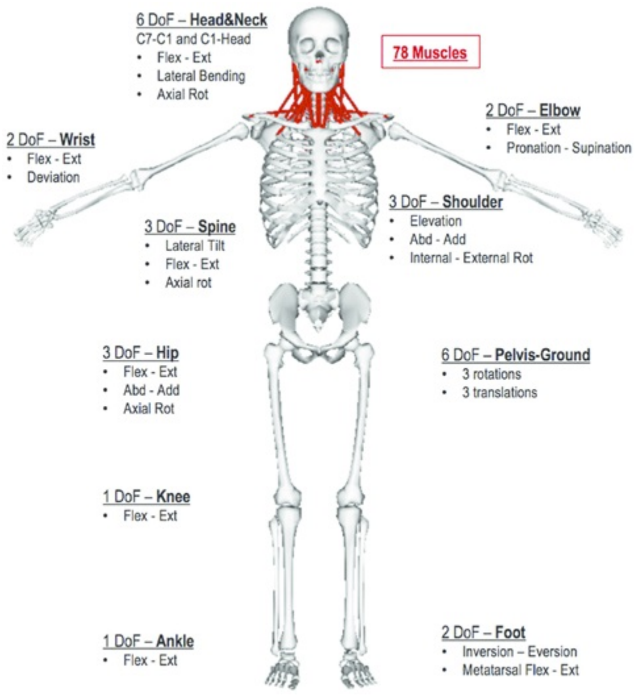

Figure 4.1: Humanoid model definition

and then this system is constrained by various methods to replicate the human motion. As in [11] the simple walking of a hierarchical rigid body system needs so much computation and results obtained are not natural looking. There are various instances of controlling the rigid bodies and making them follow some animation key frames as in [16]. To replicate similar system in Unity, two same 3D models are used in a scene. One model is the basic rigged mesh with an animator where the animation data can be used. The Unity Mecanim animation system is used to manage different animation sequences in the animator. Animation sequences like walking, getting up and idle are used in the animation system. The other model is created using Unity ragdoll wizard, which creates rigid bodies in body segments with a character joint to connect two segments. Also, this automatically set the joints limit in character joints.

4.2 Unity Mecanim

Unity engine has a very sophisticated animation system called Mecanim, which allows setup of animation sequences on humanoid character very easily. This animation sys-

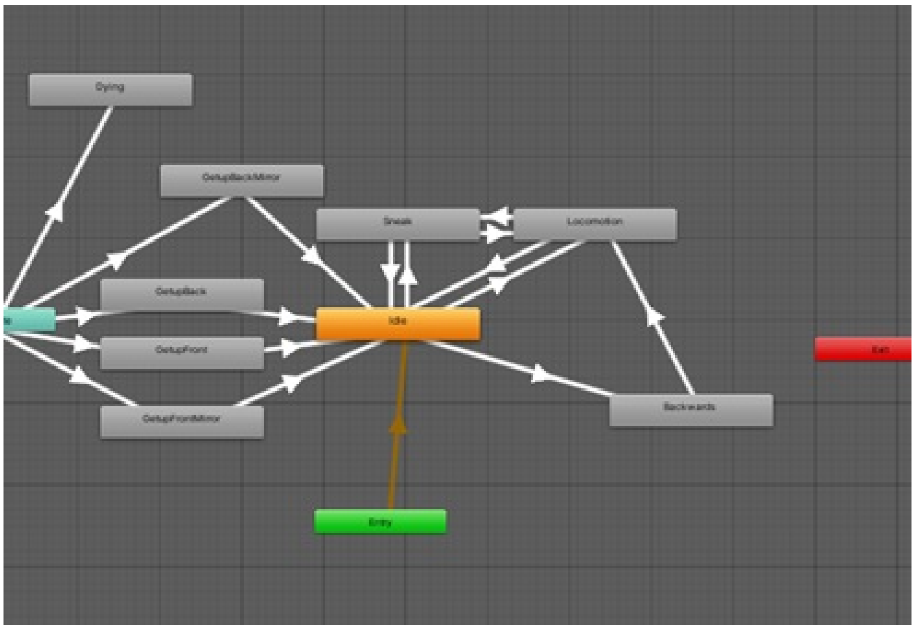

Figure 4.2: Unity Mecanim

tem uses a state machine system to transition between different animation sequences. The figure 4.2 shows the mecanim system used in the unity scene. Each animation is a state in the animation system and they can be linked to one or more other sequences in order to control the flow of states. The default state is set to *Idle* and then linked to other different states. The default state transit to another state by ignoring the current state and this can be controlled by various parameters. These parameters act as conditional checker and include *bool*, *trigger*, *string* and *int* as input.

4.3 Ragdoll

Unity has a built-in system to simulate ragdoll to make a humanoid avatar interact with physics engine. Since, it is primarily composed of rigid bodies so a skeleton is made with constraints on joints. Each limb is assigned with a rigid body, with appropriate weights and connected to one another with different types of joints. The ragdoll wizard of Unity take input of the major limbs from rigged 3D model, and assign the properties for ragdoll. By default, a character joint is created between limbs and limits for the angles of different axes is selected depending upon the limb. To get more control over

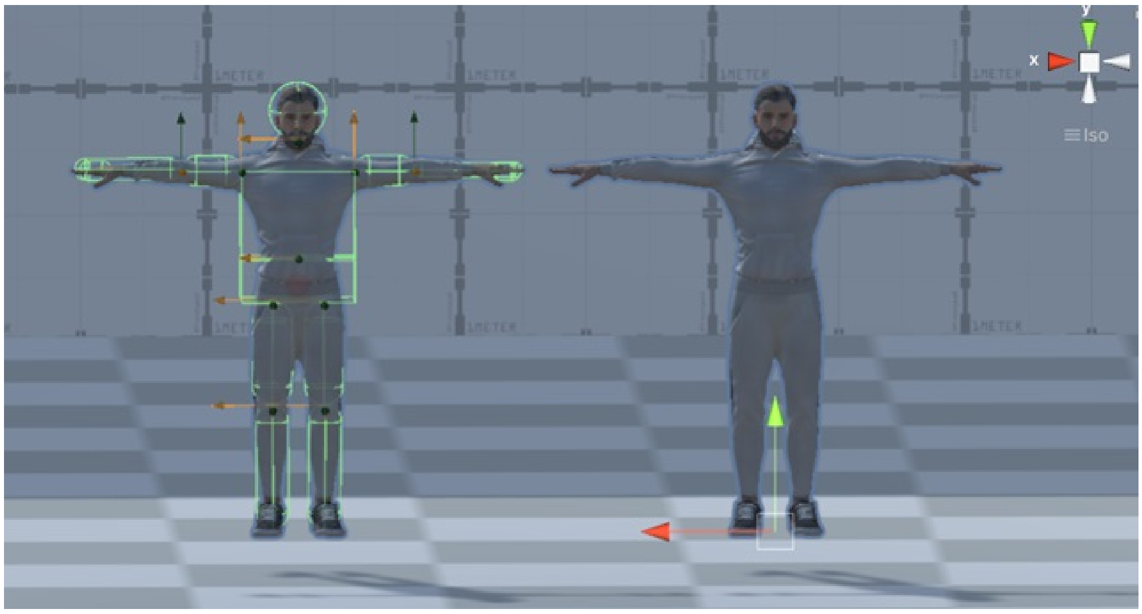

Figure 4.3: Ragdoll

the joints, these character joints are replaced by configurable joints by keeping the same limits for angles of axes. It is interesting to note how these joints can be configured to get different types of physical joints. They have 3 axes of rotation available, and also their high or low values can be set to limit the range of the motion. These configurable joints also allow to limit the translation or angular motion in any of the 3 axes. Apart for limiting and constraining motion, *Drive* values for any of axes can set the forces to rotate the joint in that axis. This is very crucial to create active ragdoll.

4.4 Active Ragdoll

The physics engine comes into action when any rigid bodies interact with each other. Now, in order to make character interact with its environment, the ragdoll must be able to interact with other rigid bodies. As already discussed before, animating ragdoll just from the driving forces create unnatural motion and not a feasible method for complex motions. Therefore, ragdoll must be able to follow the predefined motion sequence to get a more natural look. This will transit the ragdoll in an active state.

4.4.1 Keframes from animator

In this process, two rigged models of character are used, a normal animated character and a ragdoll structure. This idea of dual rig is inspired by a unity asset named *PuppetMaster*, which to some knowledge use similar approach to create active ragdoll. Here, the main purpose is to make ragdoll physically follow the motion and animation of the animated target character.
Since, all the limbs are not attached with rigid body in ragdoll so there are few body parts like hands, fingers which are just mesh without rigid bodies. In rest of this section, rig with animator will be referred as "character" and rig with rigid bodies as "ragdoll". In unity, "transform" refers to the position and orientation of an object. First of all, the transforms in ragdoll without rigid bodies and corresponding transforms in character are filtered out. These ragdoll transforms are updated according to their corresponding transforms in character as:

Algorithm 1 Update ragdoll transform from character animator

```
1: function UpdateTransform()      ▷ Function to update ragdoll transform from character animator
2:
3:     $localTransform1 \leftarrow characterTransform$
4:     $localTransform2 \leftarrow Lerp(localTransform1, localTransform2)$
5:     $ragdollTransform \leftarrow localTransform2$
```

4.4.2 Inverse Dynamics

Inverse Dynamics computes the forces and torques of an articulated body to move from one point in space to another. The transforms with no rigid body are already calculated. Now, the ragdoll transforms with rigid body are computed physically. In other words, the force and torque values are computed to achieve the target orientation. The target orientation is the transform of the character rig. So, the ultimate output here are forces and torques of the rigid body required to make a movement with the input being orientation at each joint. The torque causes a twisting effect so the rotation values from corresponding target are considered for computation. For force, the position of center of mass of character limb is taken as input for computation.

For Torque:

Algorithm 2 Compute torque value for rigid body joints

```
function ComputeTorque()        ▷ Function to compute torque value for joints

    $inverted \leftarrow Quaternion.Inverse(ragdollRotation)$
    $targetRotation \leftarrow characterRotation * inverted$
    $targetRotation.ToAngleAxis(out\ torqueAngle, out\ torqueAxis)$
    $torqueError \leftarrow FixEuler(torqueAngle) * torqueAxis$
    $PDControl(out\ torqueSignal)$
    $torqueSignal \leftarrow ClampMagnitude(torqueSignal, maxTorque)$
    $ragdollRigibody.AddTorque(torqueSignal)$
```

For Force:

Algorithm 3 Compute force value for rigid body joints

```
function ComputeForce()        ▷ Function to compute force value for joints

    $ragdollCOM \leftarrow (ragdoll.worldCOM - ragdoll.position)$
    $targetCOMPosition \leftarrow characterPosition + (ragdollRotation * ragdollCOM)$
    $forceError \leftarrow targetCOMPosition - ragdoll.worldCOM$
    $PDControl(out\ forceSignal)$
    $forceSignal \leftarrow ClampMagnitude(forceSignal, maxForce)$
    $ragdollRigibody.AddForce(forceSignal)$
```

4.4.3 PD control

A PD controller is used to combine all the input information and give a final output that is required to advance a current pose of ragdoll towards the key pose. The torque and force values are evaluated on every frame for every joint. Evidently the controller has two terms. The first one accounts for the difference in the target value and calculated value i.e, Error value, whereas the second relates to the difference in the current error value and last error value i.e, derivative. The P and D in the equation are proportional and derivative components which play very distinct and important role. The proportional component drives the value in either way positive or negative according to the error value. The derivative component control the level of change due to the proportional component and causes a damping effect. In general, the values of P and D supervise the responsiveness of the limb behaviour.

Algorithm 4 Definition function for PD control system

```
1: function PDControl($P, D, out\ signal, error, lastError$)    ▷ Function to compute controlled signal value
2:
3:     $signal \leftarrow P * (error + D * (error - lastError))$
4:     $lastError \leftarrow error$
```

The values of P and D can be appropriately achieved through a manual trial-and-error method. If the values are set too low then the joint behaviour appears slow and unresponsive. On the other hand, if the values are set too high then the motion is much faster and joints look stiff and robotic. The values are set as public variables so they can be easily manipulated in real-time and appropriate values can be selected.

Chapter 5

Implementation

This chapter deals with the implementation of different methods mentioned in previously related work section. It is intended to be an in-depth discussion of the choices made for the project, and the technical considerations made for specific cases. For the implementation, a suitable 3D environment is created with planes, stairs and terrain system in Unity.

5.1 Pose-based Control

Physically simulated character needs to be operated by abiding to the laws pf physics. As discussed in the previous section, the method of dual rig works perfectly to make the ragdoll follow animation sequences from animator. Also, various parameters are set in animator to make the character interactive. In this way, character can be controlled by a user in real-time. The joint dynamics, position and orientation are all set in real-time for a much robust natural looking system.

In figure 5.1, red colour mesh is the character with animator and ragdoll is the normal coloured mesh. The figure 5.2 shows that ragdoll follow the character animation very smoothly in idle as well as walking animation. When the ragdoll is hit with a ball, it moves away from the bounding key frames but return to its position quickly as seen in figure 5.3.

It is common practice to apply a PD controller to compute joint torques and forces about each degree of freedom based on the desired and actual angles.

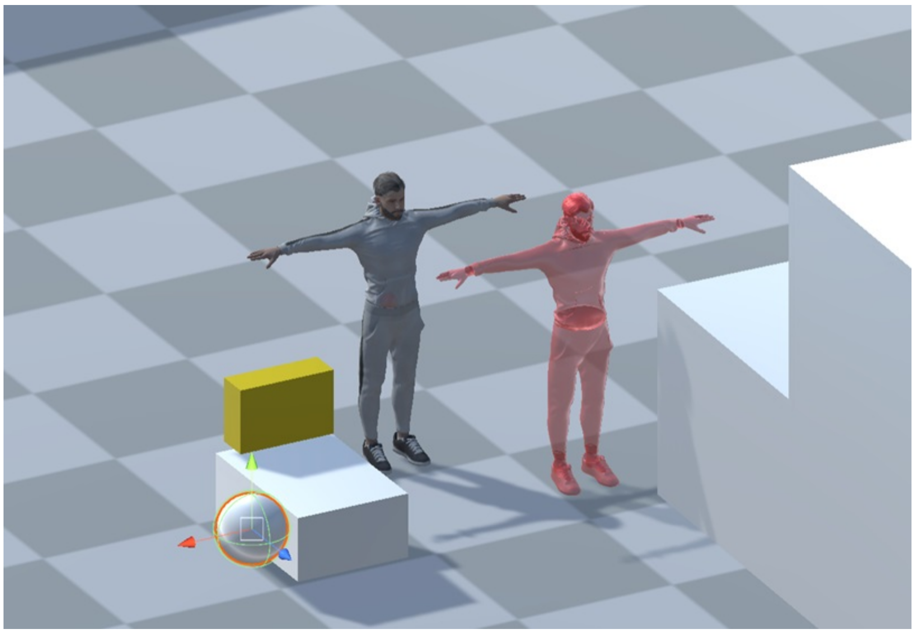

Figure 5.1: Ragdoll on left and character model with animator on right(red rig)

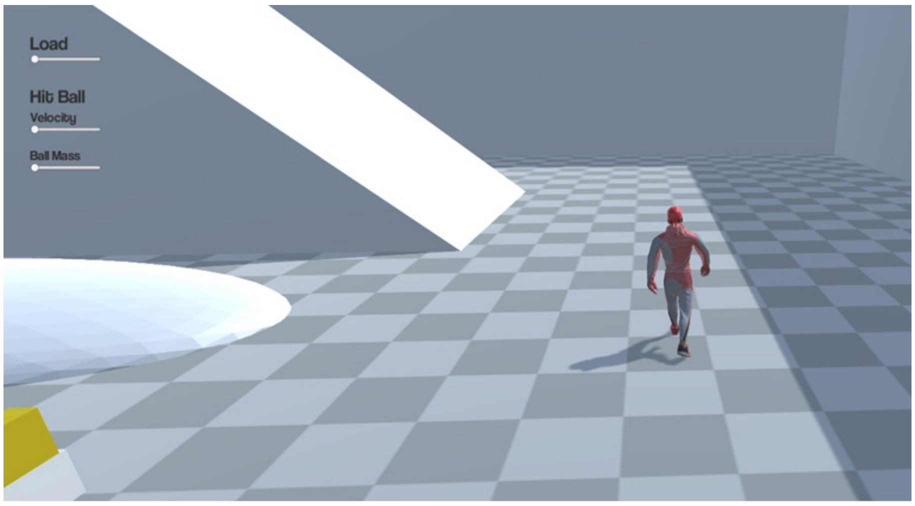

Figure 5.2: Ragdoll following the animation sequence as character rig

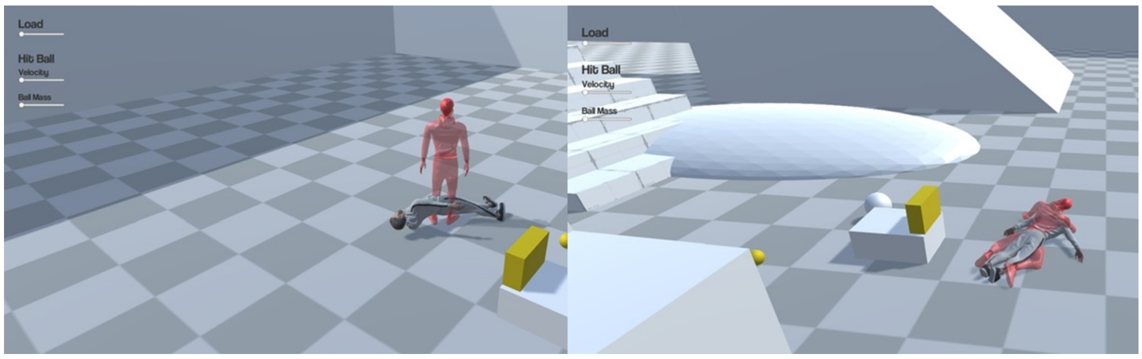

Figure 5.3: Ragdoll falls when hit with a ball(on left); Ragdoll quickly follow the character rig(on right)

5.2 Leg Adjustment

Unity engine has a basic built-in IK system which can be used for the humanoid rigs. Here, this IK system is used for proper foot placement of the character rig which will eventually be followed by ragdoll. The idea is to place each foot at the correct height from ground by applying two bone IK to hip-knee-ankle joint chain of the character. In Unity, *Raycast* physics is used from ankle position to the ground in downward direction to compute a point and its normal where each foot should be placed. While computing the ankle position, foot height and slope angle are also considered which also helps to get the proper orientation of ankle. Figure 5.4 shows an example where one leg of character is on stair and other on ground. The ragdoll is responding to IK system whereas character rig is not. Also, the character height is also adjusted so that lower foot touch the ground. The concept is implemented in the following steps:

1. Initially, the hip, knee and ankle joints of each leg are located.
2. For each leg, a downward raycast vector from ankle position is shot to find intersection point with floor.
3. The ankle target position is computed so that foot is in contact with floor.
4. In case of slope, the pelvis position is offset so that ankle which is lowest from its position reach its target position.

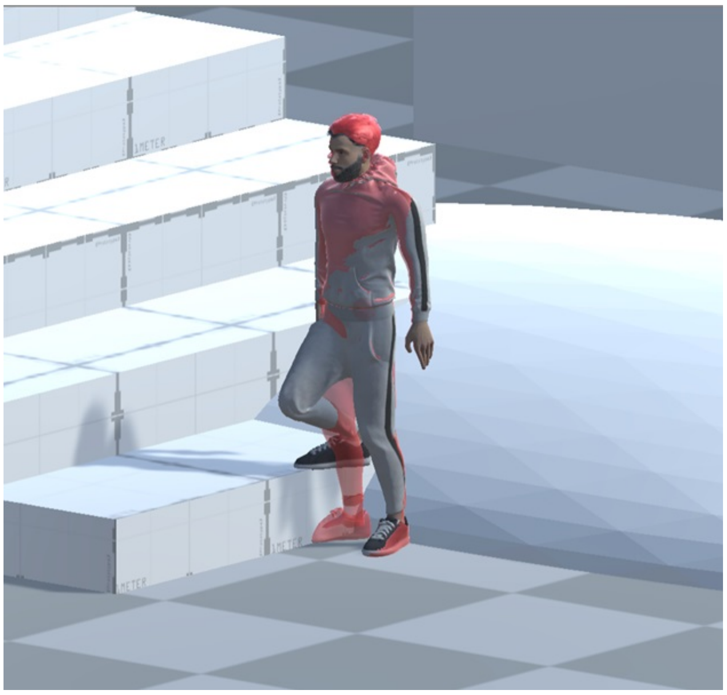

Figure 5.4: Leg adjustment using built-in IK system in Unity

5.3 Constraint Optimization for Motion

The constraint optimization is usually defined with an objective function with some constraint and target position to generate motion. In order to construct more natural motion, it is important to limit the motion boundary with biomechanical constraints. As the ragdoll is already defined so most of the joint constraints are satisfied by it. The optimization problem for lifting task is discussed in [12]. Few of the constraints defined for collision avoidance with box are:

$$X_{boundary} - X_{wrist_d} < 0 \tag{5.1}$$

$$X_{boundary} = X_{wrist_{pr}} + d_x \tag{5.2}$$

$$d_x = MAX(d) \tag{5.3}$$

where $X_{boundary}$ is the horizontal boundary position, X_{wrist_d} is desired horizontal position of wrist, $X_{wrist_{pr}}$ is horizontal wrist position from optimization algorithm, d_x is the horizontal distance to avoid collision and d is penetration limit for box.

- Objective function

In [12], lifting task is considered as a simple inverted pendulum motion. For the stability of body, ankle joint torque is considered for objective function. The function is proposed as integral of ankle torque squares in each time frame.

$$F = \int_{t=0}^{T} \tau_{ankle}^{2} dt \tag{5.4}$$

The approximate integral is calculated from the ankle torque which is obtained by inverse dynamics of ragdoll.

Algorithm 5 Objective Function for stability while lifting

```
1: $ankleTorque \leftarrow Mathf.Pow(ankleTorque, 2f)$
2: $FirstOrderFunction() \leftarrow delegatefloat$
3:
4: function INTEGRATE($first, second, steps, func$)    ▷ Function to compute approximate integral of a function
5:     $dt \leftarrow (second - first)/(steps)$
6:     $value \leftarrow Zero$
7:     for $first < second; first+ = dt$ do
8:         $value+ = func(first) * dt$
9:     $value+ = func(second) * (second - (first - dt))$
10:    $returnvalue$
11:
12: $F \leftarrow Intergrate(0, ankleTorque, T, func)$
```

5.4 Lifting a Box

Here, the task involves lifting a box from its initial position and hold it with both hands. After the unsatisfactory results from optimizations equations, motion capture data of bending and getting up is used. The animation sequence is included in *animator* of the character rig. Now, the ragdoll can be controlled to bend and get up with a key press but it still need to hold box in hands. For this purpose, *fixed joint* of Unity system is used. This joint is created on a condition that the *collider* of hands come in contact with the *collider* of box. Figure5.5 shows a fixed joint being created between hands

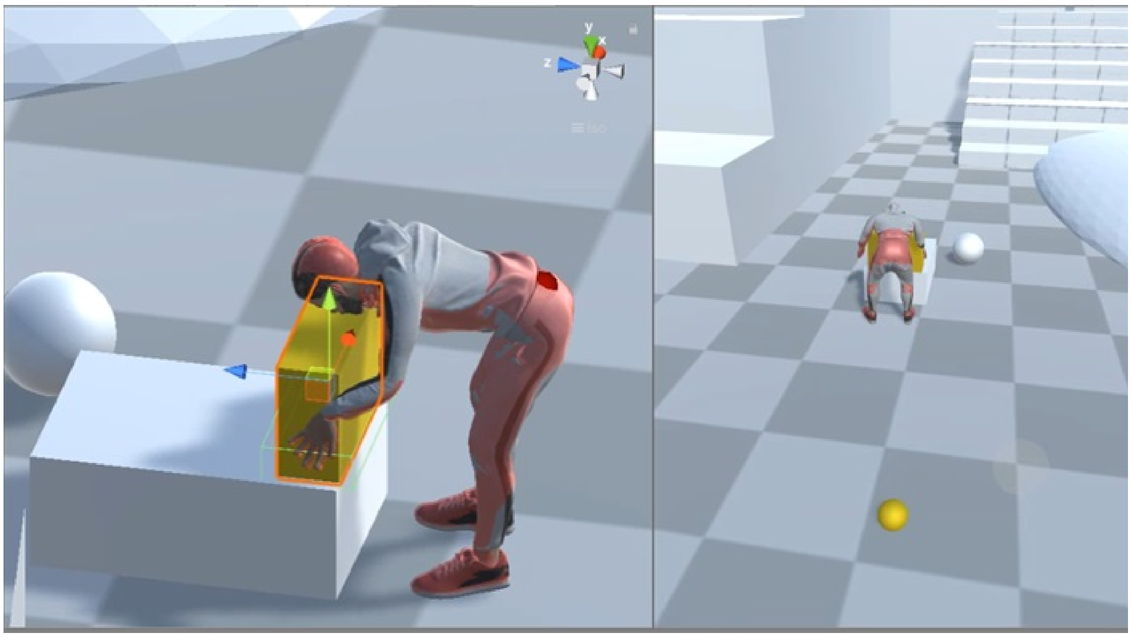

Figure 5.5: A fixed created between hands and box to hold and carry it

and box.

A maximum threshold value is set for this joint. If the box is too heavy, it will exert pressure on this joint and when the force is more than the threshold value joint will break. This threshold value also consider any forces from other external sources.

5.5 Root Balance Spring

In [18], the main reason to apply weak root balance spring was to avoid character fall over with an unpowered root. Here, the ragdoll is following the key frame position and orientation from character animator. In this way, ragdoll will always follow the animation sequence and will never deviate from the animator position. In order to give character more human motion, the weak root balance spring method is applied. Instead of applying some loose spring in the hip joint of character, the force and torque values computed by inverse dynamics are manipulated conditionally to give a similar effect.

Algorithm 6 Root Balance Spring

```
1: function BALANCESPRING($noOfCollision, collisionSpeed$)   ▷ Function that create root balance effect
2:
3:     if $noOfCollision > 0$ AND $collisionSpeed > thsesholdSpeed$ then
4:         $maxTorque \leftarrow Zero$
5:         $maxForce \leftarrow Zero$
6:         $maxJointTorque \leftarrow Zero$
7:         $SetJointTorque() \leftarrow maxJointTorque$
8:     else
9:         $maxTorque \leftarrow residualTorque$
10:        $maxForce \leftarrow residualForce$
11:        $maxJointTorque \leftarrow residualJointTorque$
12:        $SetJointTorque() \leftarrow maxJointTorque$
```

Although physically incorrect, this method produces plausible balance system which gives a sense of realism to the character. It also avoids case of super balance through the conditional values which can drop to zero when seriously imbalanced, producing realistic falls. Any collisions with other rigid bodies are considered as part of condition satisfaction. Also, threshold value is set for the maximum velocity with which collision with rigid bodies can take place. If the collision occur with enough force beyond the threshold limit then the force and torque values becomes zero, and ragdoll falls on the ground. In contrast, if there is no collision then the forces and torques are computed for each frame with maximum values be used to clamp the values, and finally get the output values.

5.6 COM Position

Every rigid body connected in the ragdoll has its own center of mass. Now, the hierarchy of different body segments in ragdoll is such that hip is on the top followed by spine, legs (both left and right). There is shoulders, arms and head under the spine segment. As hip and spine are quite high in structural hierarchy, their change in properties affect the corresponding child components.
As discussed in chapter 2, the center of mass of human body is around the position of pelvis which changes according to the body posture. While lifting any load, center of

mass of the combined system (human body + load) is outside the body in the forward direction of load position. Now, in order to maintain stability in body this point should be as close as possible to pelvis. Humans solve this problem very intuitively and bend the upper body little backward which shift the combined center of mass of system backward around the pelvis position, and thus maintaining a stable posture. This behaviour in human body come through our instincts for natural physics in environment. And if this is replicated in a virtual character behaviour then a more natural looking body movement can be achieved.
After lifting the box, the center of mass of ragdoll hip does not change as it remain in its local position of (0, 0, 0). So, this COM position can be shifted backward as like the human would do after lifting the load. Now, the displacement of this COM can be computed by the equation mentioned in chapter 3. This value of displacement depends upon the distance between hip and load position. Also, it depends upon the mass of box carried. For heavy box the displacement would be large whereas for light box it would be small. The position of new center of mass of hip is computed as:

Algorithm 7 Position COM of hip and spine

```
1:  function UpdateCOM($hip, spine, load$)    ▷ Function to update COM position
2:
3:      if $holding = True$ then
4:          $loadPos \leftarrow Distance(hip.position, load.position)$
5:          $hipPos \leftarrow hipJoint.COM$
6:          $hipPos.z \leftarrow (load.mass * loadPos)/(hip.mass + load.mass)$
7:          $hipJoint.COM \leftarrow hipPos$
8:      else
9:          $hipPos \leftarrow hipJoint.COM$
10:         $hipPos.z \leftarrow zero$
11:         $hipJoint.COM \leftarrow hipPos$
```

Here, the initial local position of hip COM is (0, 0, 0). Also, this equation is applied for spine displacement with a factor of 0.5 in the distance. This is because only mass of rigid body attached to hip is considered in above algorithm and not the mass of full character body. In figure5.6, the red point is the position of COM. When the load is not heavy, COM is around pelvis and inside the body. Whereas, when load is heavy COM is shifted backwards changing the upper body posture.

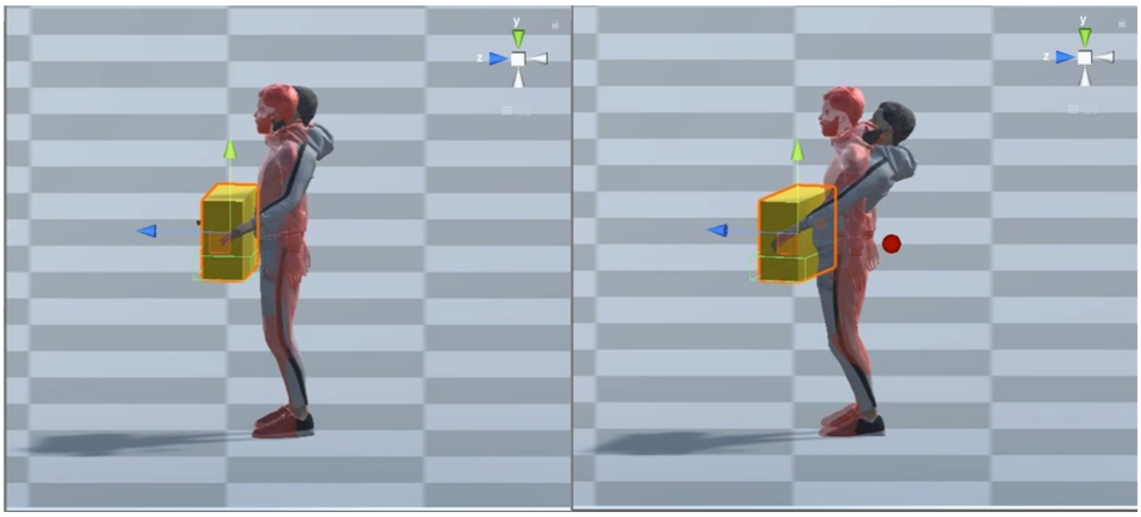

Figure 5.6: COM position: When load is not heavy(on left), When load is heavy(on right)

Chapter 6

Results & Discussion

This chapter discuss the results that we are able to achieve using the pose-control method and other techniques mentioned in previous chapter. The method of using dual rig significantly eliminate the need of reproducing basic body movement and make room for using complex movement with physical simulation. The accuracy of this method is compared during different body movements and reasoned. There are many interesting observations noticed during the implementation which are discussed in this section, along with some limitations.

The specifications of computer used for this project are:

Operating System: Windows 10
CPU: Intel® Core$^{\text{TM}}$ i5-8250U (1.6 GHz base frequency)
RAM: 8.0 GB
Graphics Card: AMD Radeon$^{\text{TM}}$ 530 Graphics 2GB

6.1 Results

6.1.1 Error Evaluation

A PD controller continuously calculates an error value as difference between desired value and measured value. This error value determines the amount of correction required in the system to reach its goal. The error values in computation of joint forces of ragdoll is evaluated at different conditions.

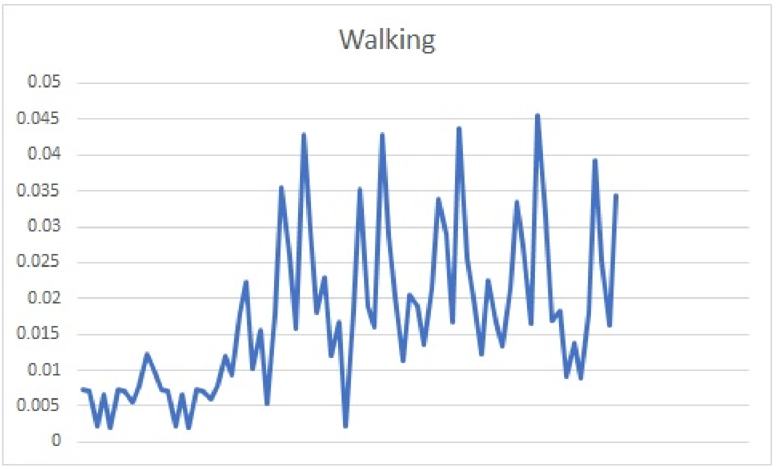

Figure 6.1: Normal walking without any load

- The first condition is when the ragdoll is not carrying any weight and just following the key frames from character animator. Fig. 6.1 shows that error value changes in sinusoidal manner with peaks as the character walk. An important thing to notice here is that maximum value of error reached is 0.04. Also at start, change in error is very low but gradually increases over time. This can be explained as ragdoll follow the key frames with minimum error in idle condition, but as soon as it start walking the error start to increase.

- The next condition is when ragdoll is carrying some light weight and following the key frames from character animator. This is shown in fig. 6.2, where the error value changes abruptly from high to low and then again low to high. Another thing to notice is that the maximum value reached is 0.4 in comparison to 0.04 in case of no extra weight. The sudden change in error is due to continuous walking movement of ragdoll. When ragdoll start walking, error value is large and then it reaches its defined goal so error value decreases. But as soon as goal is reached, because of walking cycle ragdoll state is changed and error value again shoots up high. This cycle continues until the ragdoll stop walking.

- The final condition is when ragdoll is carrying heavy weight and following the key frames from character animator. Fig. 6.3 shows the change in error values

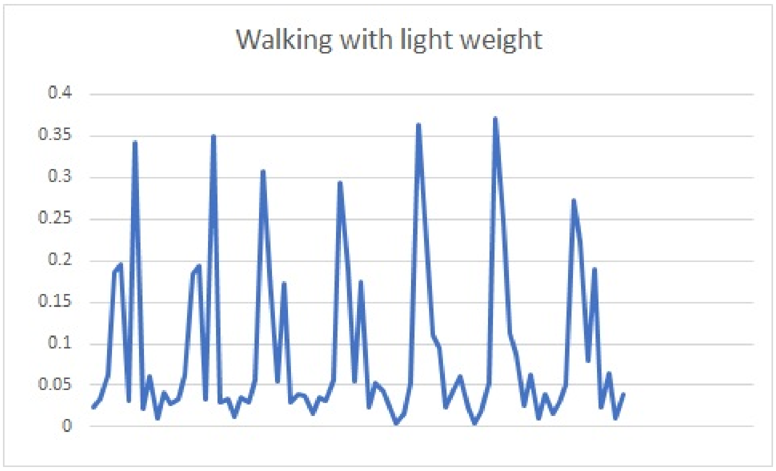

Figure 6.2: Walking with light load

and it is completely different from above two conditions. The error follows a zigzag path suggesting a very unstable value in the walking cycle. The maximum and minimum values are similar to previous condition but the rate of change is extremely high. A possible explanation for such behaviour can be massive change in center of mass position. As discussed in previous chapter, position of center of mass excessively changes the upper body posture. This makes the error value shoot up and as soon as system minimises the error it again increases to maintain changed posture.

6.1.2 Weak Root Balance

The weak root balance is implemented by applying some threshold values for joint parameters. Fig. 6.4 shows the output generated from this method. In part A, the ball is not heavy so the force is not enough to overcome the threshold value. As seen in the figure, ball is hit twice but ragdoll displaces a bit but does not fall. In part B, mass of ball is increased and the force generated is enough to overcome the threshold values. When the heavy ball hit the ragdoll, a significant displacement is observed and eventually falls on ground.

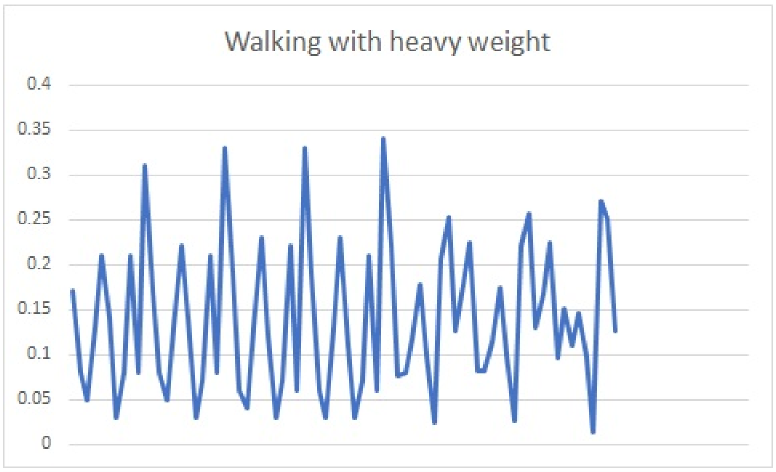

Figure 6.3: Walking with heavy weight

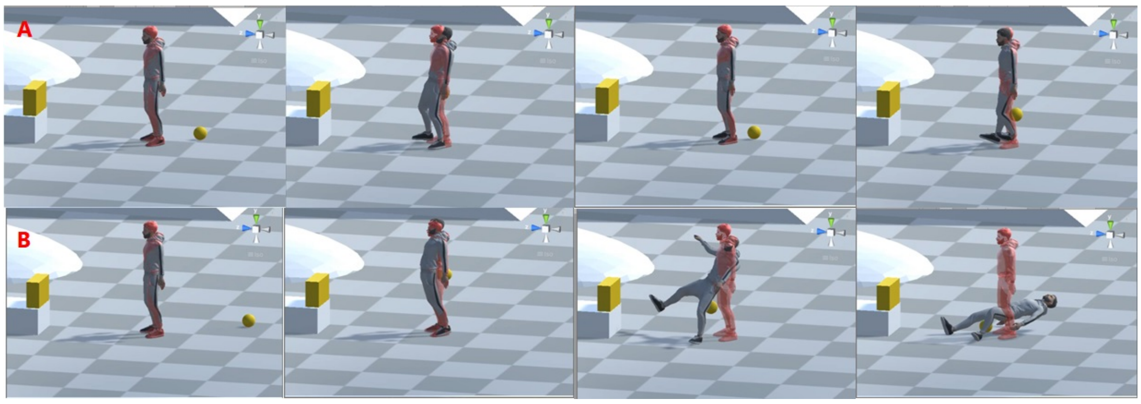

Figure 6.4: Weak root balance output

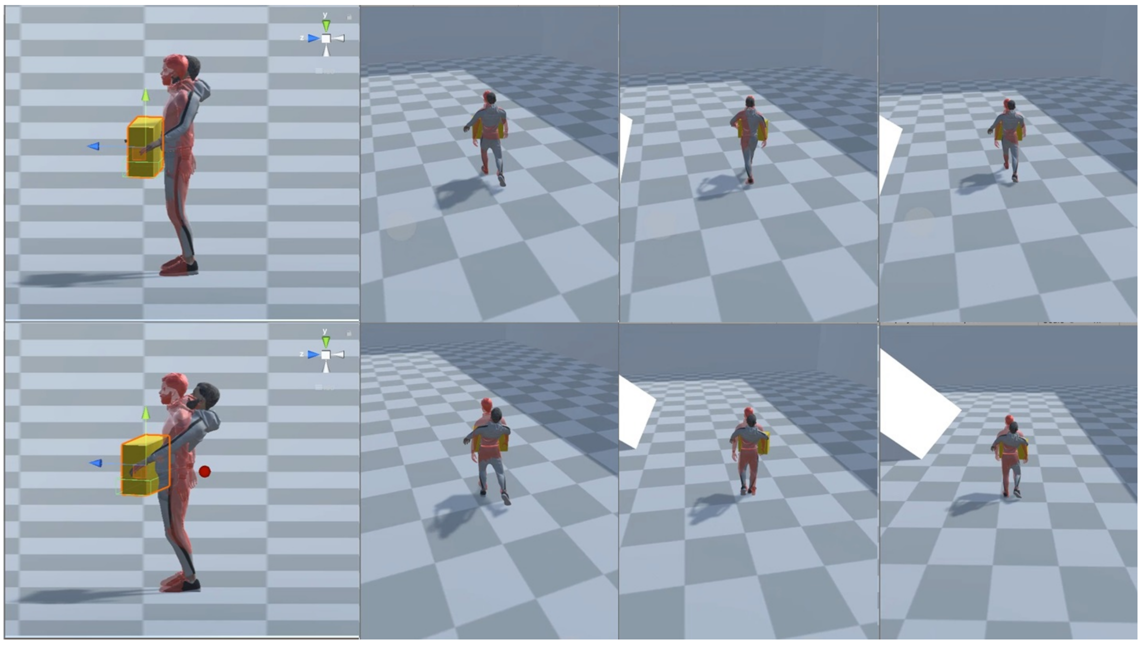

Figure 6.5: Comparison of COM position in light load and heavy load

6.1.3 Center of Mass Position

In the implementation, the COM position of hip and spine are altered according to the load carried by ragdoll. Fig. 6.5 shows the displacement of COM with the change in load. In the first row, ragdoll is carrying a very light load and hence the COM position is inside the body. The walking gait cycle can be observed which is very similar to the normal walk cycle of character. In second row, ragdoll is carrying very heavy load and the COM position is displaced backward. The change in upper body posture is significant when compared to the character rig posture. Also, the walking gait cycle is also changed and legs are either placed far away or very close to each other while walking.

6.2 Observations

While the main objectives of thesis are achieved to a large extent, following are the observations made during the implementation.

- Pose-based control is proved to be an effective method to create physics based character. It avoids the need to make a character stand or walk using standard optimization techniques. This even allows mapping of complex motions in a physically-based character.

- The output generated from the constraint optimization method is very unstable and unsatisfactory. The possible reason being so many approximations used in the implementation of objective function. Also, the values used in the objective function are further approximated values from inverse dynamics. It is clear that this approach need much precise values to get a stable system.

- The weak root spring technique used in the ragdoll joints produces very natural looking motion even in walking animation sequence. It works perfectly in normal conditions but sometimes give very fascinating output. For instance, in case of stairs if ragdoll is moving too fast, then it can fall on stairs.

- The position of center of mass is changed when ragdoll is carrying any load. The idea was to change the posture of articulated body while carrying load. But it is noted that this change in COM position has impact on the walking gait cycle of character. This is due to the fact that COM of hip and spine is changed in implementation, and both of them are quite high in hierarchical chain of articulated structure. Hence, any change in parent body will affect its child bodies as well.

6.3 Limitations

There are few limitations that have been observed in this work.

- As ragdoll is meant to follow animator key frames, it is sometimes noticed that when environment physics is working on it there is still tendency to follow the key frames. This somehow overkill the advantage of physics-based character. But this can be avoided using robust optimization.

- The weak root spring technique generated great output but it can get very unpredictable sometimes.

- The COM of rigid body in hip is considered as COM of whole ragdoll. This made room for some artefacts in simulation, like when ragdoll is carrying load the COM position go way behind sometimes, totally bending the back of character. This can be ignored by proper definition of COM of ragdoll.

Chapter 7

Conclusion

This chapter provides a summary of the contributions made by this project, followed by possibilities for future work that could improve or extend the implementation.

7.1 Contribution

Although the current implementation offers possibility for further improvement, there are few features to consider as main contributions. The project as implemented can prove to be helpful to make physics based characters carry any custom weight which gives virtual character more natural and believable outlook. The control method is based on a similar pose-based control method as discussed in chapter 2, especially the use of inverse dynamics for joint forces & torques. The idea of dual rig come from a unity asset *Puppet Master* which seems to use a similar approach for active ragdoll. The inclusion of PD controller in this control method makes the transition from one body movement to another very smooth. It act as damping factor in the control of motion by not giving any abrupt movements. The combined control method in real-time give a more natural looking outlook.

The procedural leg adjustment on different terrains using built-in IK system give the virtual character a more robust system. A similar approach to the weak root balance spring which is discussed in chapter 2, is applied to the character for replicating a natural balance. So, when character is exposed to some large external force it lose its balance and fall. This external force consists of it own drag forces as well, so when character is moving very fast the drag force value increases which can make it fall, just like humans.

The model is intended to look and behave like human as closely as possible. Another Technique used for this purpose is to position the center of mass of character after lifting the load. When a human lift any load, the center of mass of the combined system shifts. But humans adjust the posture and nullify this shift naturally. This process is replicated by applying the same principle on virtual character. The center of mass of hip and spine are shifted backward when a load is lifted. This makes the character change posture. The change in posture looks very natural and this also affects the walking gait cycle of character. The change in gait cycle depends on the mass of load carried by character, and it proves the capability of physics character. Also, this procedural physics based animation is run in real-time which means it can be implemented with standard CPU performance.

7.2 Future Work

The work presented in this project can be extended in various ways. There are several ways for improvements of existing features. These improvements include specific adjustments to suit different applications and general optimization in the current algorithms and methods.

Physiology of character: In this project, the position of center of mass is manipulated to change the posture of virtual character which in turn changed the gait cycle from its original animation sequence. A similar approach can be applied to obtain different characters with distinct physical characteristics. For example, the physiology of a child is totally different than that of an adult. The rigid bodies defined for body skeleton in ragdoll can be made small to fit the structure and also number of rigid bodies can be reduced. A separate definition of spine might not be required for it, and the position of center of mass can be adjusted to get correct natural posture. Similarly, this method can be applied to characters of different age, sex or any other distinct physical characteristics. The same task will need more time and resources to capture animation sequences using motion capture technology. Also, key frame animation technique can be avoided in every other frame to obtain realistic character motion.

Dynamic Control for different task: Currently, the character is designed to perform just one action of lifting a box and carry it around. And some real world physics principle are applied to replicate the action. In the extension, some other task can be taken into consideration and whole system can be designed according to that task. Here for lifting the box, a fixed joint is created between box and arms. So, for any different task such as grabbing and throwing an object a different approach needs to be considered. Instead of fixed joint, a loose spring joint can be used which will help in creating momentum in object while throwing. Or any other approach can be considered which will give better results. In this way, every task consist of different actions which need similar approach to this project but some different techniques in implementation.

High level optimization equation: In this project. an attempt is made to implement an optimization equation usually applied in robotics simulation of humanoid robots. These optimization equations consist of very complex motion equations following some constraints and satisfying one or more objective functions. A very simplified optimization equation is used in this equation with lots of approximation. The unsatisfactory output from this proves that these equations need a minimum amount of precision in implementation. The approximations used in time intervals and Euler integration made the implementation unstable. As these optimization methods are widely used in robotics they are heavily researched and tested. Also, such methods exist for various different tasks tested for humanoid robots. In the future, high level calculus can be implemented in Unity physics engine using some third party assets or packages.

In-depth character definition The ragdoll defined in the project consist of just one rigid body in the spine position. This can be divided into three spine rigid bodies and each can be controlled separately to give more natural outlook. Also, the center of mass of hip is used to define the COM of whole character body. And the mass considered for computation is of the hip rigid body only. This can be revised by proper hierarchical definition of character.

Bibliography

[1] N. I. Badler, C. B. Phillips, and B. L. Webber, "Simulating humans: computer graphics animation and control," *Oxford University Press*, 1993.

[2] W. L. Wooten and J. K. Hodgins, "Simulating leaping, tumbling, landing and balancing humans," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1, pp. 656–662, IEEE, 2000.

[3] J. E. Seipel and P. J. Holmes, "Three-dimensional running is unstable but easily stabilized," in *Climbing and Walking Robots*, pp. 585–592, Springer, 2005.

[4] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–8, 2012.

[5] T. Geijtenbeek, N. Pronost, A. Egges, and M. H. Overmars, "Interactive character animation using simulated physics.," in *Eurographics (STARs)*, pp. 127–149, 2011.

[6] J. Martınez, J. Nebel, D. Makris, and C. Orrite, "Tracking human body parts using particle filters constrained by human biomechanics," in *Proc. BMVC*, 2008.

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

[8] C. K. Liu and Z. Popović, "Synthesis of complex dynamic character motion from simple animations," *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3, pp. 408–416, 2002.

[9] P. Faloutsos, M. Van de Panne, and D. Terzopoulos, “Composable controllers for physics-based character animation,” in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 251–260, 2001.

[10] S. Levine and J. Popović, “Physically plausible simulation for character animation,” in *Proceedings of the 11th ACM SIGGRAPH/Eurographics conference on Computer Animation*, pp. 221–230, 2012.

[11] A. Witkin and M. Kass, “Spacetime constraints,” *ACM Siggraph Computer Graphics*, vol. 22, no. 4, pp. 159–168, 1988.

[12] W. Si, S.-H. Lee, E. Sifakis, and D. Terzopoulos, “Realistic biomechanical simulation and control of human swimming,” *ACM Transactions on Graphics (TOG)*, vol. 34, no. 1, pp. 1–15, 2014.

[13] V. B. Zordan, A. Majkowska, B. Chiu, and M. Fast, “Dynamic response for motion capture animation,” *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, pp. 697–701, 2005.

[14] I. Ismail, M. S. Sunar, H. W. Qian, and M. A. M. Arsad, “3d character motion deformation technique for motion style alteration,” in *2015 4th International Conference on Interactive Digital Media (ICIDM)*, pp. 1–4, IEEE, 2015.

[15] A. Shapiro, P. Faloutsos, and V. Ng-Thow-Hing, “Dynamic animation and control environment,” in *Proceedings of graphics interface 2005*, pp. 61–70, 2005.

[16] A. Shapiro, D. Chu, B. Allen, and P. Faloutsos, “A dynamic controller toolkit,” in *Proceedings of the 2007 ACM SIGGRAPH symposium on Video games*, pp. 15–20, 2007.

[17] K. W. Sok, K. Yamane, J. Lee, and J. Hodgins, “Editing dynamic human motions via momentum and force,” in *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pp. 11–20, 2010.

[18] P. Wrotek, O. C. Jenkins, and M. McGuire, “Dynamo: dynamic, data-driven character control with adjustable balance,” in *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pp. 61–70, 2006.

[19] W. Braune and O. Fischer, *On the Centre of Gravity of the Human Body: As Related to the Equipment of the German Infantry Soldier.* Springer Science & Business Media, 2012.

[20] W. Erdmann, "Center of mass of the human body helps in analysis of balance and movement," *MOJ App Bio Biomech*, vol. 2, no. 2, pp. 144–148, 2018.

[21] Y. Xiang, J. S. Arora, and K. Abdel-Malek, "3d human lifting motion prediction with different performance measures," *International Journal of Humanoid Robotics*, vol. 9, no. 02, p. 1250012, 2012.

[22] Y. Abe and J. Popović, "Interactive animation of dynamic manipulation," in *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 195–204, 2006.

[23] L. Ye Hu, "Physics-based character animation," B.S. thesis, Universitat Politècnica de Catalunya, 2016.

[24] F. Reif, *Fundamentals of statistical and thermal physics.* Waveland Press, 2009.

[25] D. Honeywell, "Pid control," *Process Automation Handbook*, pp. 155–63, 2000.

Appendix

Abbreviations:

COM: Center of Mass

DOF: Degree of Freedom

IK: Inverse Kinematics

PD Control: Proportional Derivative Control

Third party Assets:

3D Model from Mixamo.com along with few animation clips.

Motion-capture data from CMU Graphics Lab online motion capture database