University of California, Merced

Controlling The Physics of Humanoids

A thesis submitted in partial satisfaction of the requirements
for the degree Master of Science

in

Electrical Engineering & Computer Science

by

Robert Backman

Committee in charge:

Marcelo Kallmannn, Committee Chair
Stefano Carpin
David Noelle

2014

The thesis of Robert Backman is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

David Noelle

Stefano Carpin

Marcelo Kallmannn, Committee Chair

University of California, Merced

2014

To my parents . . .

Table of Contents

List of Figures

List of Tables

Vita

2010-2013	Teaching Assistant UC Merced
2010-2013	Graduate Student Researcher UC Merced, Computer Graphics
2003-2010	Freelance 3D artist.
2006-2008	Content contributor PSD Magazine.
2005-2008	Graphic Designer IMD Graphics Napa.

Publications

Computer Animation and Virtual Worlds 2013 (CAVW). Designing Controllers for Physics Based Characters with Motion Networks. R. Backman, M. Kallmann

Motion in Games 2012 (MIG). Modeling Controllers for Physically Simulated Characters with Motion Networks.R. Backman, M. Kallmann

UCM SoE Technical Report 2011. Walking Gait Generation for HOAP3 Humanoid Robot. Y. Huang, R. Backman M. Kallmann

ICRA 2010. A Skill-Based Motion Planning Framework for Humanoids. M. Kallmann, Y. Huang, R. Backman

CHAPTER 1

Introduction

There is a huge drive in the computer animation world to understand the physics of human motion in order to synthesize realistic motions. The industry standard approach is to use either motion capture devices (Figure 1.1) or key framed animation (Figure 1.2) to generate kinematic(position/angle based) motion data as input to a virtual character.

This approach is sufficient for many tasks and results in very nice looking motion, however it has several limitations. The first problem is that it takes a large amount of human effort and sophisticated equipment to capture the motions or a large amount of time for animators to hand make the motions. Kinematic motions are also difficult to transfer between characters of different morphologies; an active field of research trying to solve this is called kinematic retargeting. Additionally, since kinematic animation is a purely data driven approach, the resulting animations are limited to the motions that have been captured and typically are unrealistic when there are disturbances or changes in the environment. Since the range of human motions is so vast, possibly infinite, a purely data driven kinematic approach can not store all the possible human motions. To solve the inherent issues with kinematic animation techniques researchers are looking at kinetic or dynamic(energy/force based) motion synthesis techniques. A dream of many researchers in animation is to understand the underlying principals of human motion to allow a character to generate new motions and quickly react to a dynamic environment.

Using physics to generate motions is more difficult since the designer no longer has direct control over the characters configuration but instead has control over the forces that are being applied. Controlling a character at the force/torque level, as opposed to the position/velocity level has the advantage that it is easier to react to disturbances in the environment, since these can be incorporated into the characters internal forces and feedback mechanisms. Additionally controlling the forces of the characters allows for a much larger range of output motions from an equivalent amount of kinematic data. Finally, discovering the low level principals of control for a physics based character has the potential to solve issues with retargeting since a physics based controller structure is independent of a character morphology.

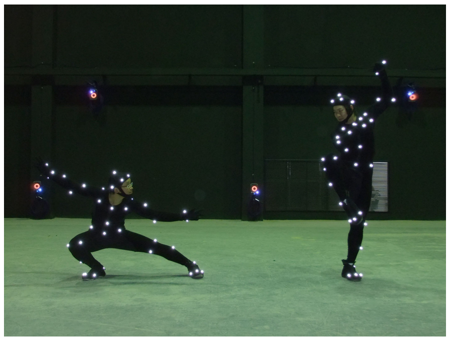


Figure 1.1: Vicon motion tracking system.

1.1 Problem Statement

Developing physics based character controllers is an active field of research, however, little work has gone towards transferring these technologies to people with no programming experience. Additionally, there is no standardization across controllers since the framework is often tightly coupled with the related skill, so it is usually difficult to extend them to new skills.

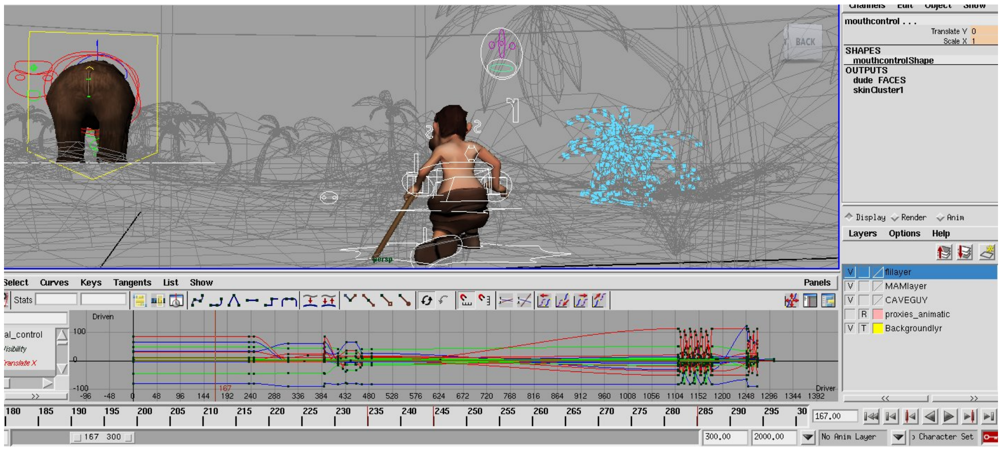


Figure 1.2: Character rig and keyframe animation curves.

1.2 Thesis Overview

Chapter 2 will explore the relevant related work to physics based character animation, starting with kinematic motion transformation, then the development of simple physics based controllers and finally to multi-skill controllers. Chapter 3 will first look at the function of physics simulation then will go to a system overview of our system. Chapter 4 describes the primary contribution of my thesis; which is a user interface that allows non-programmers to develop physics based controllers. Chapter 5 describes my work developing a control system for a real humanoid robot. Chapter 6 describes a side project using the XBox Kinect to recognize hand configurations with the hopes of using it as an interface to control a physics based character.

CHAPTER 2

Related Work and Motivation

This chapter starts by exploring different work with kinematic motion data in section 2.1.It then looks at early control strategies for physics based characters in section 2.2. Then, in section 2.3, it explores some attempts at getting a physics based character to follow motion capture data. Section 2.4 will look at a common strategy for balance control by stepping. In section 2.5 we describe some more comprehensive multi-skill controllers that have been developed. Finally in section 2.6 we summarize the progress of controllers and explain the direction that this research is hoping to open for the field.

2.1 Kinematic motions

Physics based animation has its roots in Kinematic animation. Researchers have been interested in ways to modify a kinematic motion to achieve a new objective, that way a single kinematic motion could be used to produce a whole set of new motions. We will first look at various techniques of kinematic animation to understand the foundation of physics based animation.The state of the art approach for real-time motion generation remains based on kinematic animations, typically using a finite set of motion capture or key-framed clips augmented with some blending and warping. For example, Levine et al. [1] show how a small number of motions can control a kinematic character in real time. The method learns a reduced dimensionally motion model that allows them to synthesize new motions within the range of the training motions. But as with any kinematic approach, the motions generated are limited to the examples given. Physics can also be used to transform kinematic animations. For instance, de Lasa et al. [2] show a feature-based method for creating controllers for dynamic characters of varying morphologies. Objective terms are developed within a prioritization algorithm to allow the character to preserve features such as center of mass, angular momentum and end effector positions.

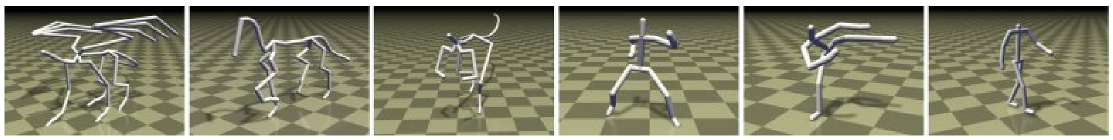


Figure 2.1: Image from Continuous Character Control with Low-Dimensional Embeddings by Levine et al.

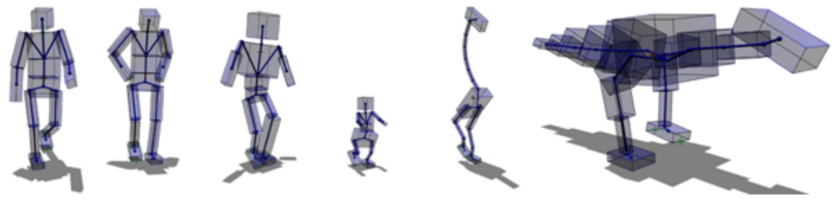


Figure 2.2: Image from Feature-Based Locomotion Controllers by deLasa et al.

Popovic et al. [3] use a space-time optimization approach to transform motions in physically accurate ways. Their method combines the expressive richness of the input animation with the controllability of space-time optimization to allow motions to be transferred to kinematic characters with very different morphologies while preserving the style.This approach is ideal but it must be customized for the given task and the developer is left with the task of determining the objective function to optimize.

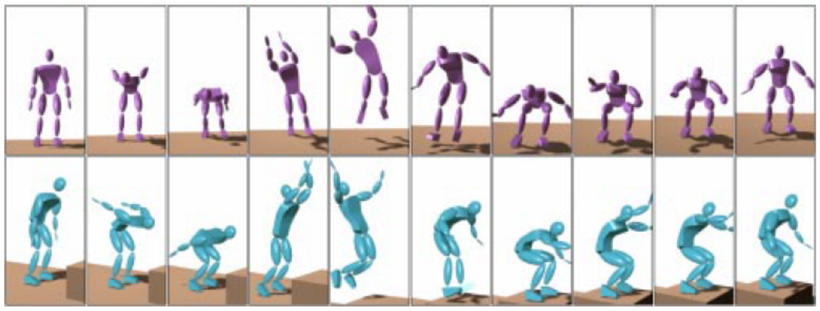


Figure 2.3: Image from Physically Based Motion Transformation by Popovic et al.

2.2 Early control strategies

Since the 90s researchers have been implementing increasingly robust control strategies for achieving a diverse set of skills for physics based characters. Early work by Hodgins et al. [4] showed how controllers could be adapted to new characters. They start with a running controller for a man then show how it can be adapted to a woman, child and a fictional character. Their system is robust enough to adapt online, so a character can change morphologies at runtime and the controller can maintain functionality.This was an interesting first work but their method relies on a functioning controller to be created, which can take a large amount of work and it is not clear that it generalizes beyond locomotion.

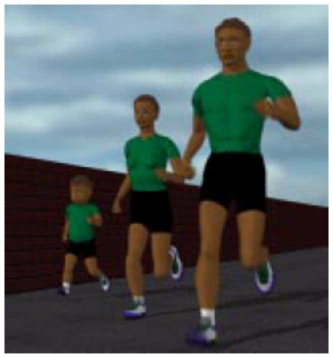


Figure 2.4: Image from Adapting Simulated Behaviors For New Characters by Hodgins et al.

Laszlo et al. [5] explored limit cycle control to combine a closed loop animation cycle with open loop feedback control. Their system is able to maintain balance and follow paths while preserving the style of the input motion cycles. Virtual model control developed by Pratt et al. [6] was a key insight to how to control individual parts of an articulated figure using a Jacobian transpose method initially developed for robotics. This work was further developed by Pratt et al. [7] where they used velocity based control strategies to achieve fast walking. Virtual model control is widely used due to its simplicity for implementation and its stability. Our work uses Virtual Model Control extensively, thus becoming the foundation for our balance and tracking control.

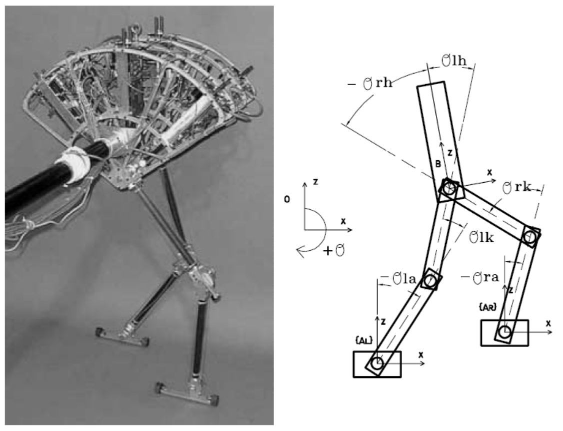


Figure 2.5: Image from Virtual Model Control: by Pratt et al.

2.3 Motion capture

The role of physically simulated characters in video games and movies has been anticipated for quite some time but to this date is mostly present as passive rag-doll interactions. Promising research has however been proposed to improve physics based characters while maintaining real-time performance and the realism of motion capture data. Macchietto et al. [8] used momentum control so a character can follow motion capture data while responding to large disturbances. Their system shows impressive results of a character standing on a platform on one foot who is able to maintain balance and produce realistic responses with no input motion, by using a full body optimization to maintain a zero sum angular momentum. If a character is pushed in the head the arms and leg will counter the force by moving against the push. By implementing a control policy to reduce the over all angular momentum of a character during walking a very natural arm swing motion is automatically synthesized. This method shows promising results but it is not clear how this can be generalized beyond standing in place and balancing. Perhaps this can be integrated in a future version of our system.

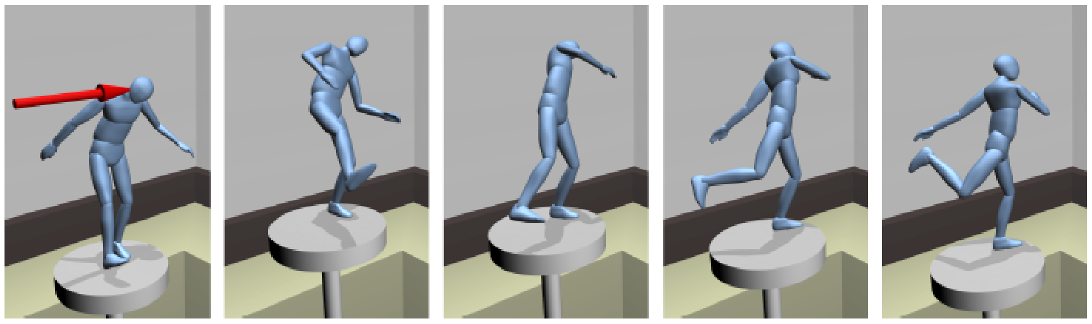


Figure 2.6: Image from Momentum Control for Balance: by Macchietto et al.

Methods for automatically switching from kinematics to physics only when needed have been developed by Zordan et al. [9]. This is an approach that has been adopted by the industry in games such as Grand Theft Auto or Madden football, where the characters follow kinematic motions until a collision with the environment is detected. Then the final state of the kinematic motion is used as the initial state of the physics simulation. Using a hybrid approach with both kinematic and physics based animation seems very promising, and our system is based on a similar hybrid approach. Methods for extracting the style in a motion capture clip have been transferred to physically simulated characters by Lee et al. [10]. They are able to extract the style from a motion capture walk and use it as input to a steerable walking controller for a physics based character. Their controller works by modulating the speed of the input motion and performing a parallel forward dynamics simulation to correct from deviations between the input and output motions. It would be highly desirable to use motion capture sequences as a starting point for a controller, and this paper became an inspiration for our work, since our approach modulates input motions to achieve a stable controller.

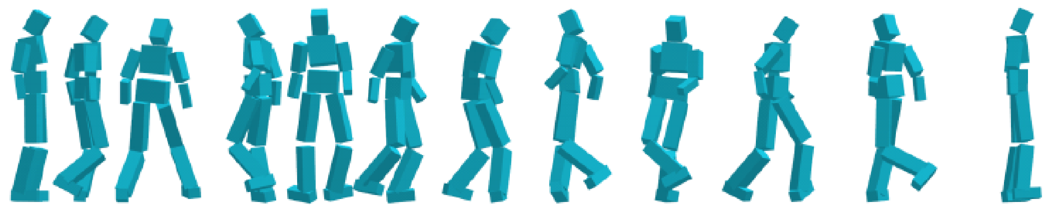


Figure 2.7: Image from Data-Driven Biped Control by Lee et al.

Complex motion capture sequences have also been achieved with physically simulated characters by Lie et al. [11], however this requires computationally heavy sampling strategies and is not robust to disturbances. Their system performs a bidirectional search of the input torques needed to make a physics based character achieve some objectives defined in the input animation sequence. Since we aim to have realtime execution and controllability of a control running an expensive offline process is not feasible.

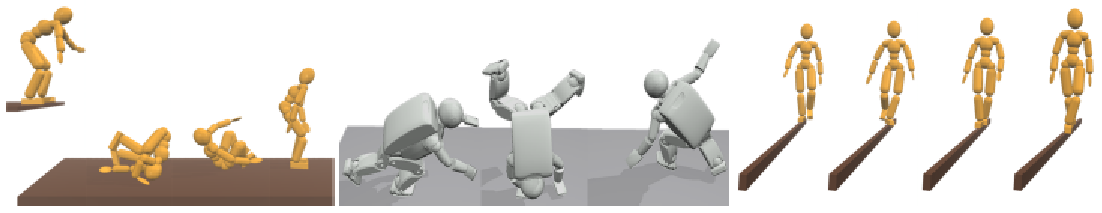


Figure 2.8: Image from Sampling-based Contact-rich Motion Control by Liu et al.

DaSilva et al. [12] explored adaptation of human styles from motion capture to Physics-based characters. The use of motion capture data is desirable because it guarantees realistic motions, but it also serves to broaden our understanding of the principles of how humans (and other animals) interact with the physical world.

2.4 Foot placement strategies

Human locomotion is difficult to simulate, mainly because the motion is typically not statically balanced, it is an inherently unstable system. Many researchers have investigated foot placement strategies in order to maintain dynamic balance while walking. Yin et al. [13] introduced simple feedback rules for the swing leg to produce robust walking behavior called SIMBICON. Our system is used to create a SIMBICON like controller.

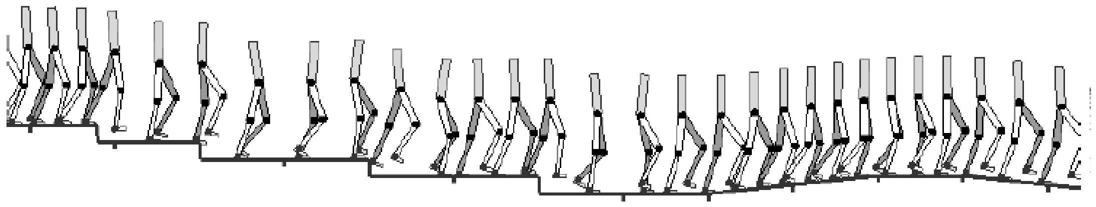


Figure 2.9: Image from SIMBICON by Yin et al.

Tsai et al. [14] used an Inverted Pendulum (IP) model for the swing foot placement in locomotion. Kajita et al. [15] used Zero Moment Point (ZMP) strategies to determine swing foot location. While usual IP and ZMP strategies assume that the contact surface is a plane, Pratt et al. [16] have explored a more general capture point method that can generalize to non flat surfaces. Using the ZMP and IP models for balance are promising but they require an analytical solution to a pretty complex problem. We need strategies such as these for staying balanced but we look at ways of automatically determine the control parameters.

Muico et al. [17] proposed a system that looks several steps ahead to determine contact states that can keep a character balanced.

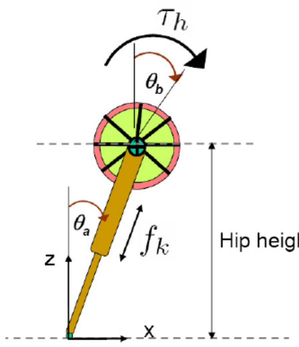


Figure 2.10: Image from Capture Point: A Step toward Humanoid Push Recovery by Pratt et al.

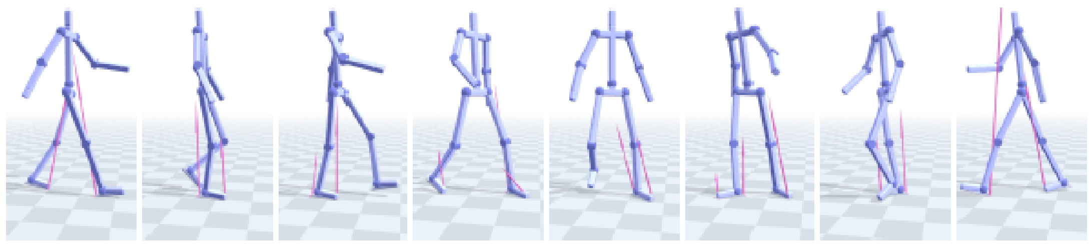


Figure 2.11: Image from Contact-aware Nonlinear Control of Dynamic Characters by Muico et al.

These foot placement control strategies work well for individual locomotion skills and they provide foundations for more complex systems of combined behaviors.

2.5 Composite controllers

With the development of many low level balance control schemes many researchers have investigated how to incorporate them into a more diverse skill set. Coros et al. [18] demonstrated robust bipeds performing various skills such as pushing and carrying objects, extending early work on the use of virtual forces. One of the goals of our work is to make a set of controllers that are created in a generalized way that can all be used on the same character.

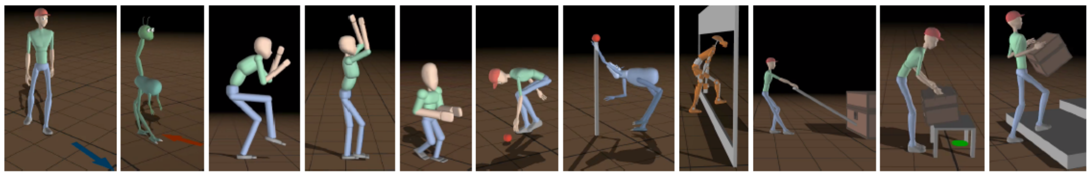


Figure 2.12: Image from Generalized Biped Walking Control by Coros et al.

Faloutsos et al. [19] determined a range of controller operations considering initial conditions (pre-conditions) using a Support Vector Machine to learn the range of post/pre-conditions that allows controllers to be concatenated together.Our work looks at determining the range of output states (post-conditions) that a controller can get the character to given an initial state (pre-condition).

Coros et al. [20] developed a task based control framework using a set of balance-aware locomotion controllers that can operate in complex environments [21]. Jain et al. [22] explored an optimization approach to interactively synthesize physics based motion in a dynamic environment. Coros et al. [23] developed a robust parameterizable control framework for a simulated quadruped.This is perhaps one of the related works that is closest to our own. They build a system

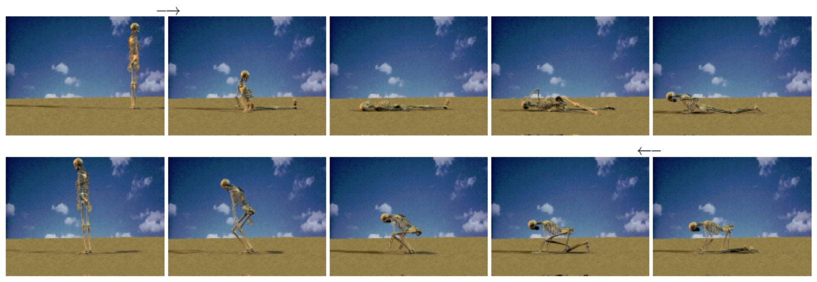


Figure 2.13: Image from Composable Controllers for Physics-Based Character Animation by Faloutsos et al.

so a user can have control of the input trajectories. However their system assumes that a functioning controller has been developed, our system allows a controller to be designed from the bottom up.

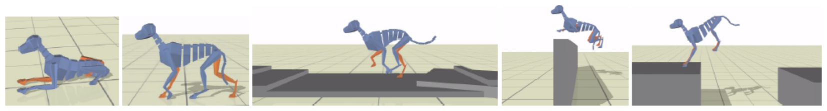


Figure 2.14: Image fromLocomotion Skills for Simulated Quadrupeds by Coros et al.

Finally, recent work by Liu et al. [24] have showed impressive results able to sequence controllers in order to perform parkour-style terrain crossing.

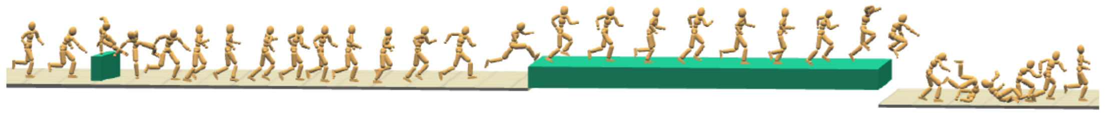


Figure 2.15: Image from Terrain Runner: Control, Parameterization, Composition, and Planning for Highly Dynamic Motions by Liu et al.

2.6 Summary of related work and contributions

Previous work in physics-based character animation has focused on the development of successful controllers for a number of tasks and situations. The primary goal of my thesis is to introduce a system that is able to expose the creation and exploration of such controllers to designers in an intuitive way. In doing so our system proposes two main contributions: 1) a set of data processing nodes to model controllers with graph-like connections able to form complete control feedback loops, and 2) a simple and effective sampling-based algorithmic approach to automatically achieve robustness and parameterization of designed controllers.

CHAPTER 3

Physics Based Animation

Here I give a brief overview of physics simulations in section 3.1 and describe the primary requirements. In section 3.2 we explore human balance strategies. Section 3.3 gives a brief overview of our system, and then we go into more depth concerning the physics module in section 3.4 and the animation module in section 3.5. Finally in section 3.6 we show how we combine these two modules together to build a unified control framework.

3.1 Simulation Environment

A fundamental requirement for physics based animation is to use a physics simulation. The simulation must be able to handle rigid body dynamics with collision detection. The primary constraints for characters are hinge joints, universal joints, ball and socket joints (with one,two and three DOF respectively) as shown in Figure 3.1. Additionally there are contact constraints (Figure 3.2) that are created at each frame between objects that are colliding. These contact constraints are essentially ball and socket joints that have zero resistance to tension. At each frame of the simulation the dynamics of the rigid bodies are updated. This means that, based on the external forces coupled with existing constraints, each rigid body is assigned a new position, velocity, orientation and rotational velocity. There are many possible physics simulation environments but for my research I primarily used Open Dynamics Engine(ODE) since it is fast, free and well documented.

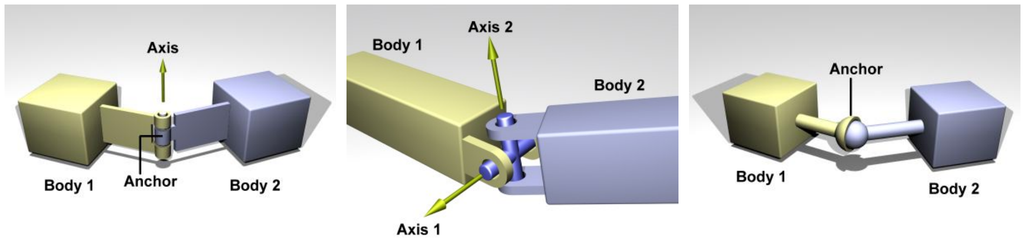


Figure 3.1: Three types of joint constraints. (Images from ODE wiki)

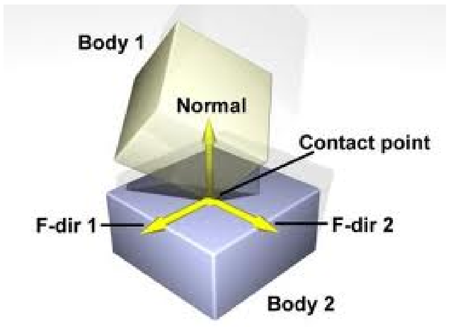


Figure 3.2: Contact constraint for collision handling. (image from ODE wiki)

3.2 Inspiration from Human Biomechanics

Generating realistic humanoid motions is an especially difficult problem since we are all experts in humanoid dynamics. From the moment we are born we start developing our sense of motion, both from self experience and watching others. That is not to say it is a trivial problem since it takes several years for a human to build the dexterity and strength to allow bipedal locomotion and many more years to learn to do more complex skills like gymnastics. A typical person may not be aware of the actual control system they develop in order to walk and stay balanced but for the most part the controls are very similar across individuals. To maintain static balance (slow motions) we modulate our Center Of Mass (COM) by moving our upper body to maintain balance. If we want to stay still we keep our COM directly above our feet, more specifically within the convex polygon defined by the outline of our feet called the support polygon Figure 3.3.

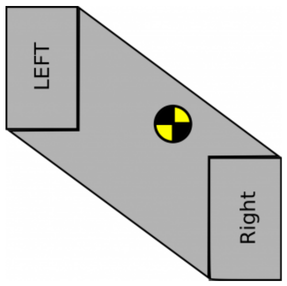


Figure 3.3: Support polygon during double support.

In the case where we are not able to stabilize our COM, because of limitations in our movement, we can change the contact polygon by taking a step in the right direction. A quick experiment can show how robust our control algorithm is. If you walk up to someone and push them slightly they will lean into your push. If you push them sufficiently hard they will be unable to compensate by leaning and have to take a step away from you to maintain balance. Additionally we can modulate our angular momentum to maintain balance, for example swinging your arms while you are falling can help rotate your body for an optimal landing orientation. For more energetic motions we maintain not the COM but the Zero Moment Point(ZMP) or Center of pressure within the support polygon since at high speeds the overall Kinetic energy of the system becomes important to balance.

3.3 System Overview

The core of our system has two main modules: an *animation module* and a *physics module*. The animation module generates target angles based on input trajectories. The physics module contains a tracking controller that produces the torques necessary to drive each joint of the physically simulated character towards the target angles specified by the animation module. Additionally, the physics module has a virtual force controller that adds additional torques to achieve higher level requirements such as staying balanced or achieving a global position of a body part in the joint hierarchy. See Figure 3.4 for an overview of the main parts of the system.

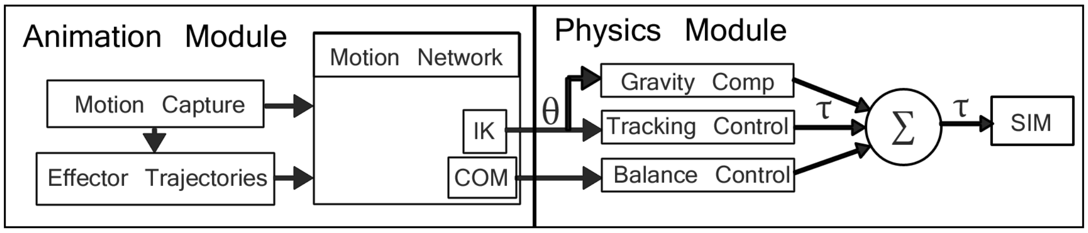


Figure 3.4: Data flow of the system.

The animation module operates at 60fps feeding a stream of postures to the physically simulated character. The output trajectories are specified in joint angles, or also as end-effector positions, which are then converted to joint angles using Inverse Kinematics (IK).

The physically simulated character is composed of a set of rigid bodies connected by hinge, universal and ball joints as shown in Figure 3.5. Each rigid body in the character is approximated by an oriented bounding box for fast collision handling. The character is simulated using ODE and is running at 1200 FPS. The reason for the high simulation frame rate is to handle high speed contacts, which is a potential problem with typical forward dynamic simulations.

3.4 Physics Module

The Physics Module has two main components: tracking control and virtual force control. The tracking control makes the character follow joint angles specified by the animation module. The virtual force control allows higher level goals such as maintaining balance, global effector goals and gravity compensation.

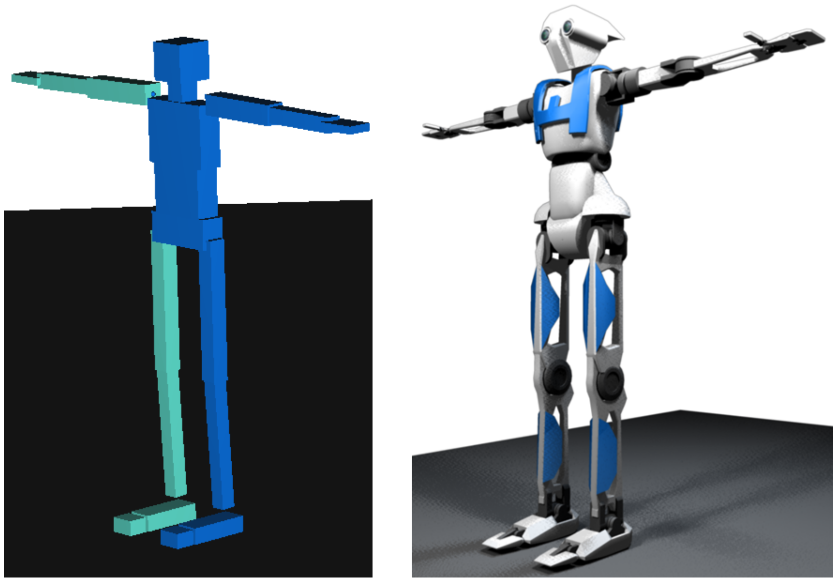


Figure 3.5: Simplified model for collision detection (left) and full character model (right).

3.4.1 Tracking control

A Proportional Derivative (PD) servo at each joint is responsible for tracking the target angular value θ_{goal} and rotational velocity v_{goal} specified from the animation module for each rigid body. We use PD gain coefficients of $k_p = 4kn/rad$ and $k_d = 2\sqrt{k_p}$.

The goal angle θ_{goal} for each joint can be either specified in local coordinates or in a *heading-based frame*. The heading-based frame is calculated by starting with the root joint global orientation then aligning it with the world up vector. Knowing the heading of the character is useful for encoding the notion of front, back, left and right of the character independent of its orientation and is critical for balance feedback. However, this approximation can be a problem if the character flips upside down since the heading will suddenly change. If there are inputs that cannot be achieved, as is common with PD tracking joint angles, it will approach the target as close as possible.

3.4.2 Virtual force control

Virtual Model Control (VMC) was introduced by Pratt et al.[6] for bipedal robots and has subsequently been used in many systems such as[18]. Considering the character is under actuated(the root is not controllable), it is desirable to control certain components with external forces. It would be a trivial matter to have a character stay balanced and track arbitrary motions by applying external forces to each rigid body that composes the character, however this would effectively defeat the purpose of physics simulation since the result would typically be unrealistic. Applying forces to arbitrary rigid bodies is commonly called the 'god force' since it essentially allows a character to violate the laws of physics and act as if there were strings attached to its body parts like a puppet. To approximate this level of global control we can imagine a virtual external force acting on a rigid body to achieve some goal, then convert this virtual force to internally realizable torques that span from the affected body up the joint hierarchy to a more stationary body.

For example, to control the position of a character's hand in global coordinates we calculate a virtual force that will move the hand towards a goal configuration (see Figure 3.6), and then convert this virtual force into a set of torques that can be applied to the arm joints to achieve the goal of precise hand placement. Ideally this chain of rigid bodies should span all the way to the foot in contact with the ground but in practice it only needs to go to the torso of the character.

Another use of VMC is to control the swing foot during a walk cycle, since it will rarely be at the same state as the input motion, this can be critical for preventing premature foot contact with the ground. To maintain static balance of the character we employ a similar virtual force on the COM to bring its floor projection to the center of the support polygon and then convert the virtual force to joint torques for the stance leg. It is also a simple way to control the velocity of the COM while walking.

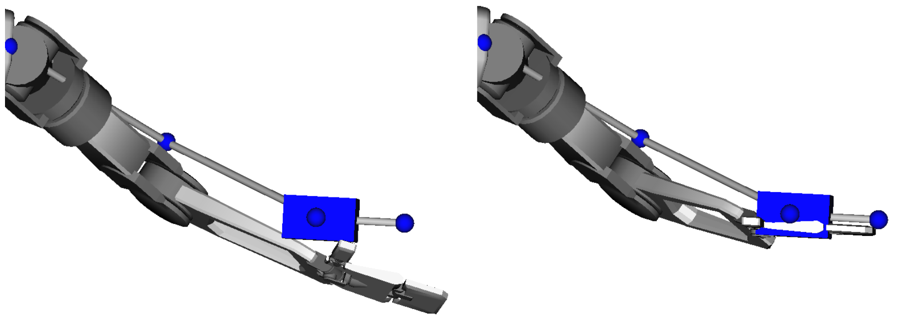


Figure 3.6: Global targets may be difficult to reach with joint control (left). Virtual forces can be effective (right).

Gravity compensation torques are also computed to allow lower gain(less stiff) tracking by proactively countering the effects of gravity. For each joint that is considered for gravity compensation the COM and mass of the descendant joints is calculated, then a torque is applied that would counter the moment this mass would create. Gravity compensation is typically only applied to the upper body but can also be used for the swing leg for walking.

An important component of our balance strategy is controlling the orientation of the torso independent of the contact configuration. Without considering the root orientation the character typically leans over and falls as soon as it lifts the swing leg, due to the sudden change in torque requirements for the stance leg. However, since the torso has no parent joints directly in contact with the ground it cannot directly be actuated, so instead a virtual torque is calculated for the root that must be distributed to the stance leg. For double support this torque is distributed to both legs.

With these components we are able to achieve full body posture control to maintain balance while completing complex global objectives, such as touching a characters toe while standing on one foot and holding the arm parallel to the ground, as shown in Figure 3.7.

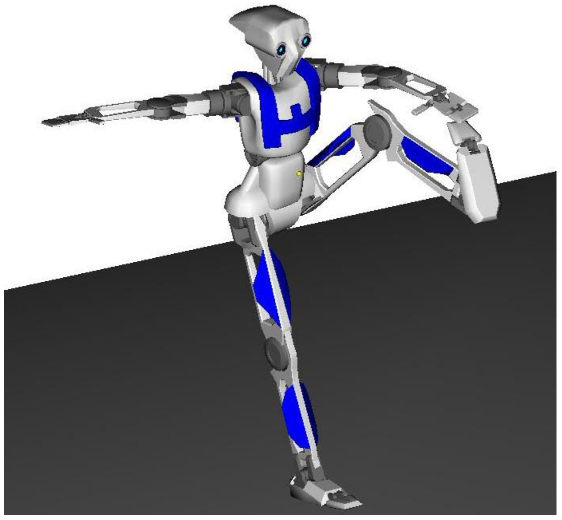


Figure 3.7: Character performing balance and reaching

3.5 Animation Module

The animation module creates kinematic motions that are used as input to the physics module. The motion can be described as a sequence of full body postures determined at each frame by spline trajectories controlling joint angles or effector trajectories (that are later converted to angles by IK). Trajectories can be encoded by a sparse list of control points and tangents or by a dense list of points that are linearly interpolated (typically the case of mocap data). Another common type of trajectory in our system is a boolean step function, where each control point is constrained to be either zero or one, representing true or false. These trajectories are useful for toggling the activity of different components of the system.

The output may be a combination of the above methods, for example the upper body may be driven by Euler angles derived from motion capture data while the lower body may be driven by feet IK trajectories. The methods described above can create simple motions that maintain balance, such as jumping to a known goal, but they fail when there are external disturbances or changes in the environment, or if the motion gets complicated with large sudden changes in contact. Typically, to control balance, a separate system of feedback controllers is layered over the animation module to override or modulate the reference motions that are produced. But we are interested in a composite/unified approach that brings feedback terms directly into the animation module.

3.6 Combining Animation and Physics Modules

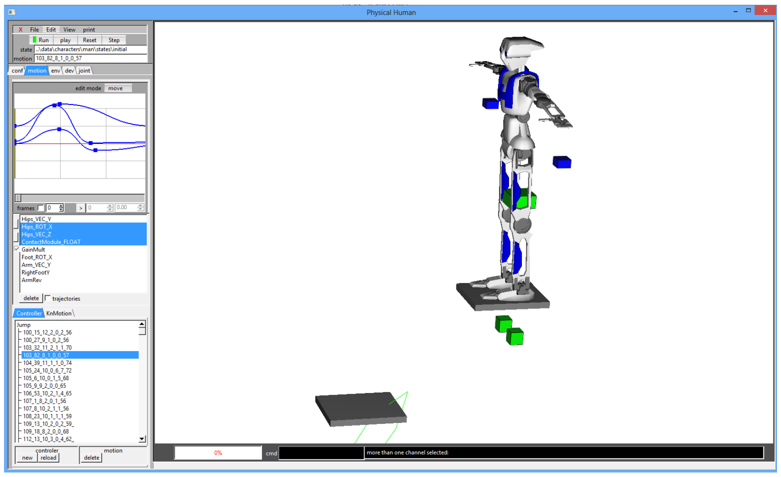
In addition to having control points (or frames) serving as input for the animation module, feedback variables become additional inputs to the animation system and gain parameters become additional outputs. Trajectories can thus control any parameter in the system and the results of the physics simulation can modify the trajectories. For example, for jumping, a trajectory is needed for representing a global gain multiplier for the tracking control so the character can become stiff before jumping, then less stiff during flight, then gradually stiffer to the default value for landing. Another trajectory can be used to change the gains on a virtual force that is computed for the swing foot that gradually transitions towards the end of a step.

By exposing the lowest level control parameters to the designer we raise the risk of them creating non-functioning controllers, but we also provide the potential of exploration and creation of endless possible controllers. In the next chapter we will describe our user interface for exposing these modules to a designer.

CHAPTER 4

A Graphical User Interface for Designing Physics Based Controllers

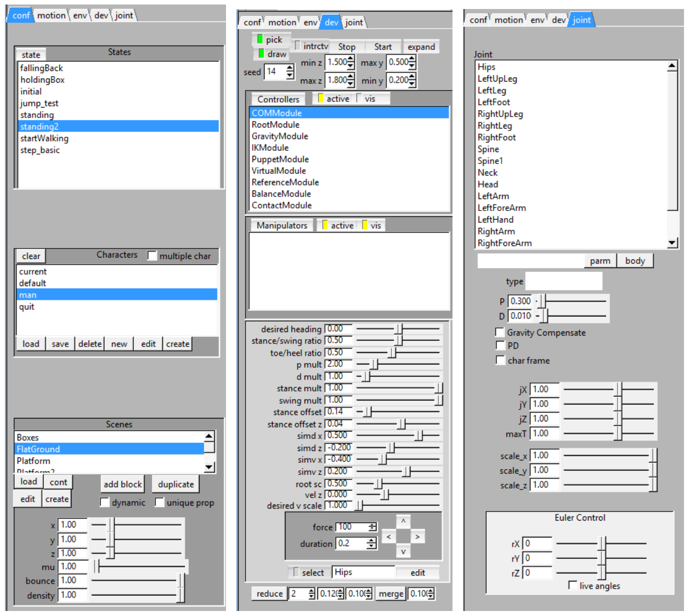
This chapter will describe our User Interface(UI) that allows a designer to build a functioning controller for a physics based character. Section 4.1 looks at the time-dependent directed acyclic graph, a fundamental data structure of our system. Then in section 4.2 we show how to modify and parameterize a controller by using a randomized manifold search algorithm. Section 4.3 lists a subset of the parameters that are used for a typical controller. Finally in section 4.4 we will step through the process of creating a SIMBICON(SIMple BIped CONtroller) like walking controller.



state ..\data\characters\man\states\initial
motion 103_82_8_1_0_0_57

more than one channel selected:

Figure 4.1: Snapshot of system UI



x 1.00
y 1.00
z 1.00
mu 1.00
bounce 1.00
density 1.00

seed 14
min z 1.500
max z 1.800
max y 0.500
min y 0.200

desired heading 0.00
stance/swing ratio 0.50
toe/heel ratio 0.50
p mult 2.00
d mult 1.00
stance mult 1.00
swing mult 1.00
stance offset 0.14
stance offset z 0.04
simd x 0.500
simd z -0.200
simv x -0.400
simv z 0.200
root sc 0.500
vel z 0.000
desired v scale 1.000
force 100
duration 0.2
select Hips
reduce 2 0.12 0.10 merge 0.10

P 0.300
D 0.010
jX 1.00
jY 1.00
jZ 1.00
maxT 1.00
scale_x 1.00
scale_y 1.00
scale_z 1.00
rX 0
rY 0
rZ 0

Figure 4.2: UI elements for configuring the simulation and character parameters.

4.1 Time-Dependent Directed Acyclic Graph

The physics module is able to maintain balance and achieve high level goals such as foot and hand global positions but the parameters are static. The animation engine generates motion but it has no notion of the physics. To interconnect these two modules we present a system called a Time-dependent Directed Acyclic Graph (T-DAG), inspired by the Directed Acyclic Graph in Autodesk Maya.

The T-DAG interconnects the animation module with the physics module. To foster the intuitive development of controllers we then propose a Graphical User Interface(GUI) to expose the parameters of the physics module and to connect them with appropriate channels from the animation module.

Any relevant parameter can be exposed as a *channel*. Channels can represent the orientation or position of an IK end-effector, an individual joint angle, boolean values, feedback parameters or gain parameters. Some examples of feedback parameters are: the pressure of the stance foot, the velocity or relative position of the COM, and many others which are described in more detail in Section 4.3.

The user interface allows the individual or group assignment of channels to any type of trajectory. For example with a forward jump, since the motion is typically left-right symmetric, we have one trajectory that specifies the rotation of the foot but we connect it to both feet.

4.1.1 Operation Nodes

To transform the motions several *operation nodes* are introduced. Each node in the control graph takes as input a set of channels (trajectories or feedback parameters), performs an operation, and then outputs the transformed value.

It is unclear what the optimal set of nodes would be to create a controller. Our approach was to start with a set of nodes that could achieve important common actions such as walking or jumping. The work initially started as a way to modulate input trajectories to parameterize a controller. So for example if a character wanted to make a step that was a bit higher off the ground it makes sense to simpy add a value to the IK foot trajectory. However simply adding a value would not produce the desired result since the entire step would be off the ground. Instead we can add the original foot trajectory to a spline to have only the middle of the step be higher off the ground. Scaling trajectories was a natural need to build feedback controllers to account for gain values. Switching and modulation nodes were developed since many motions are left right symetric, so the controller design can be more generic in terms of Stance side or Swing side. Then a controller can be created for the swing leg and this output can be routed to either the left or the right side.The operators described later in this section capture all these identified needs.

An interesting topic, but beyond the scope of this paper, is determining the optimal set of nodes that are needed and performing some analysis as to the coverage of the control space created by these four operators. Beyond that what are the results of adding different types of nodes to the system.

The animation module outputs trajectories based on editable splines or on feedback parameter that are used as input (see Figure 4.3). After a group of nodes is connected in a desired way, the T-DAG network can be saved as a template to be used for other controllers or duplicated for other channels. Then, by varying the input trajectories several goals can be achieved without changing the network connections.

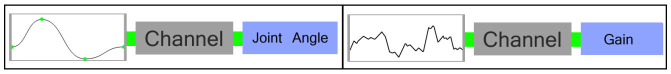


Figure 4.3: Animation nodes can be based on spline input (left) or feedback input (right).

Several key operations needed to model controllers are available as nodes to be interconnected and added to the T-DAG. These operations are illustrated in Figure 4.4, and are described below:

- Addition node: adds the input trajectories. It can take any number of inputs.
- Multiplication node: multiplies the values from a set of input trajectories.
- Modulation node: this node requires one step function and at least one trajectory input. For each high step in the control input (step function) the first input trajectory is scaled in time to fit within the step, and for each low step the second input trajectory, if there is one, is scaled in time and fit within the low step.
- Switch node: it also requires one step function and one or two input channels. If the step function is high it outputs the first input, and if it is low it outputs the second input (if there is one).

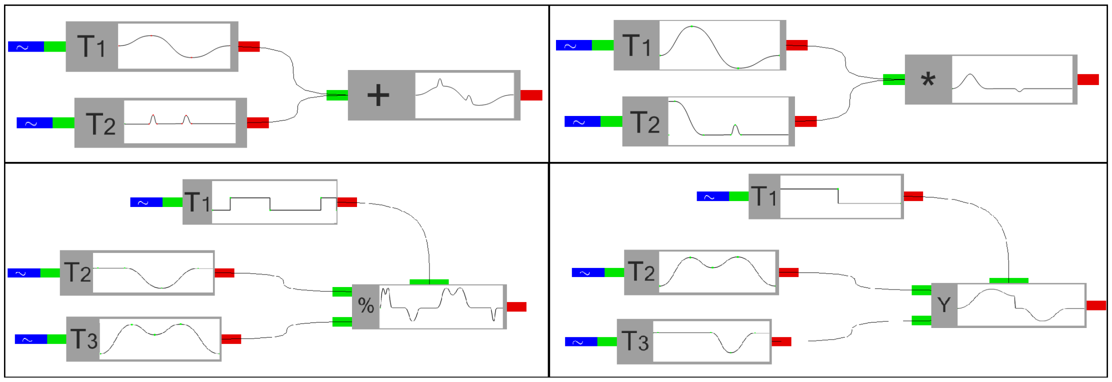


Figure 4.4: Operation nodes, in top-down left-right order: addition, multiplication, modulation and switch.

An intuitive graphical user interface was developed to allow designers to edit and explore T-DAGs. Figures 4.5- 4.11 are direct snapshots from the graphical input panel of our motion network editor. The accompanying video to this paper illustrates interactive sessions with our system[1].

4.2 Trajectory Randomization to discover controllers

Once a T-DAG is built trajectories can be connected in different ways. They can be designed by hand (using editable Splines) or they can come from motion capture. Editable offset trajectories can also be easily associated to motion capture trajectories by using an addition node, allowing different possibilities for customization. Any set of control points can then be exposed to an automatic sampling procedure that will explore successful variations of the controller in order to achieve parameterizations for real-time execution.

A *cost function* is selected as a network of feedback channels, for example jumping requires the character to be statically balanced at the end of the motion and to minimize the total torque of the tracking control. Walking requires the

[1]video available at http://graphics.ucmerced.edu/projects/12-mig-mnet/

character to be upright and have the COM moved by some minimum distance. Additionally, there is a boolean parameter provided by the physics module which looks for non-foot contacts with the environment which multiplies with the cost to ignore any motions that have *bad contacts*. There is also an *objective function* that is the goal of the simulation, the goal can be a certain target distance, or for example, a certain distance and height for a jump controller to achieve.

Once the objective and cost networks are constructed, a sampling process can be initiated. The trajectory control points that are exposed are randomly sampled within initial *sampling bounds* and the simulation is run along with the controller. If after n tries the controller does not achieve the objective, the sampling bounds are enlarged and the process re-starts. If the objective is satisfied then the control points are saved to a file along with *motion descriptors* of the outcome (the achieved jump distance, walk speed, etc). If a controller fails in some more global way, such as falling over, then it is discarded during the sampling phase. To assure that a controller will work, some representation of the environment and character initial state needs to be embedded into the controller.

After several motions are found that successfully complete several objectives, the successful motions are then used as starting points for a new round of iterations. We randomly choose new objectives and then use radial basis interpolation of the k-closest previous successful examples to find a set of trajectories which would ideally meet the objective. This typically does not work at first since there is no guarantee that interpolating successful controllers will give a new functional controller, but it works well as a starting point for the next round of sampling. The longer the sampling process runs, the better the interpolations become. When enough coverage of the desired variations is achieved, an effective parameterization of the objective space is achieved.

4.3 Parameters

Any parameter in the system can be exposed to the motion network to design a controller. Here we will explain a few of the parameters that are needed for the controllers in the paper.

4.3.1 Control Parameters

Root (Position and Rotation): the root of the character can be animated and at each frame the joint angles are determined based on IK effectors that are specified in global coordinates.

End-Effectors (Position and Rotation): the character frame based position and rotation solved with analytical Inverse Kinematics.

Joint Offsets (Rotation): added to the reference joint angle before tracking torque is calculated.

Joint Angle (Rotation): for non IK joints the desired rotation can be specified directly.

Desired COM Velocity (Vector): the desired velocity of the COM which is used as input into the Balance controller.

Toe Heel Ratio (Scalar): specifies how much the COM should shift to the front or back of the foot, a value of one puts the COM at the toes, 0.5 is midway between the toe and the heels.

Stance Swing Ratio (Scalar): specifies how much the center of mass should be above the stance foot(determined by stance state) a value of 1 puts the COM on the outside edge of the stance foot.

Stiffness Multiplier (Scalar): global value that applies to all joints or per joint to change the overall stiffness of the tracking controller.

Stance State (Boolean): specifies which foot is the stance state, if Stance State is true the left foot is the stance foot otherwise it is the right foot.

Character Frame (Boolean): specifies if the tracking controller should calculate torques relative to the parent joint or the character frame.

4.3.2 Feedback Parameters

COM Position (Vector): distance between the COM and stance foot in character frame coordinates.

COM Velocity (Vector): velocity of the COM in the character frame coordinates.

Total Torque (Scalar): the sum of all the torques from the tracking control on the previous frame.

4.4 Case Study

In this section we summarize in more detail the steps required for creating a walking controller including the balance feedback terms. The gait period of the walk is defined by the step function in Figure 4.5. The step function is defined by two control points and since we desire a symmetric gate the middle control point is half the duration of the trajectory.

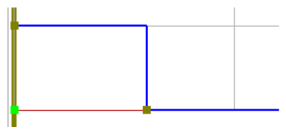


Figure 4.5: The stance state.

Figure 4.6 shows the trajectory that controls the vertical position of the foot. The trajectory is routed to the foot in Figure 4.7 modulated by the step function defined in Figure 4.5.

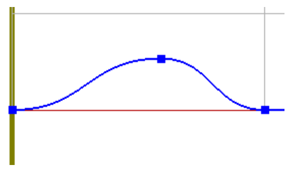


Figure 4.6: Swing Y trajectory.

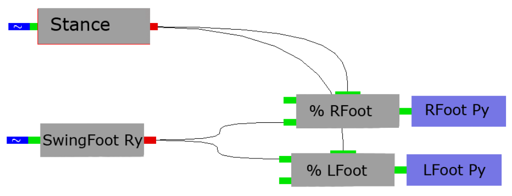


Figure 4.7: The swing Y network.

Similarly the Z position (Figure 4.8) of the foot and the X rotation (Figure 4.9) are modulated based on the step function in Figure 4.5.

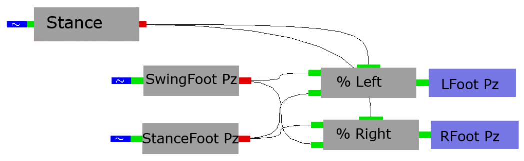


Figure 4.8: The Z position of the foot.

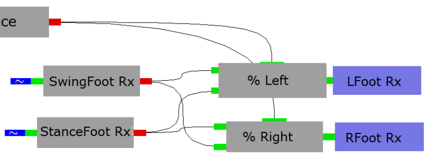


Figure 4.9: The X rotation of the foot.

The arm rotation is defined by two trajectories (Figure 4.10): one for the stance (bottom) and one for the swing (top). They are routed to each arm in Figure 4.11 and modulated by the step function in Figure 4.5. The angle is inverted for the right arm. The value for each arm receives a further offset and is then added to the forearm.

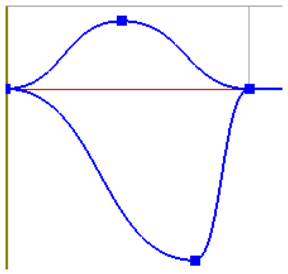


Figure 4.10: Arm Trajectories.

Up till this point we have simply built a kinematic walking controller that can be parameterized by editing the control points of the input trajectories. What we need to do next is define the feedback control that will allow the character to maintain balance while walking under physics. The first thing we define are several constant value parameters (Figure 4.12) that are needed by the *virtual*

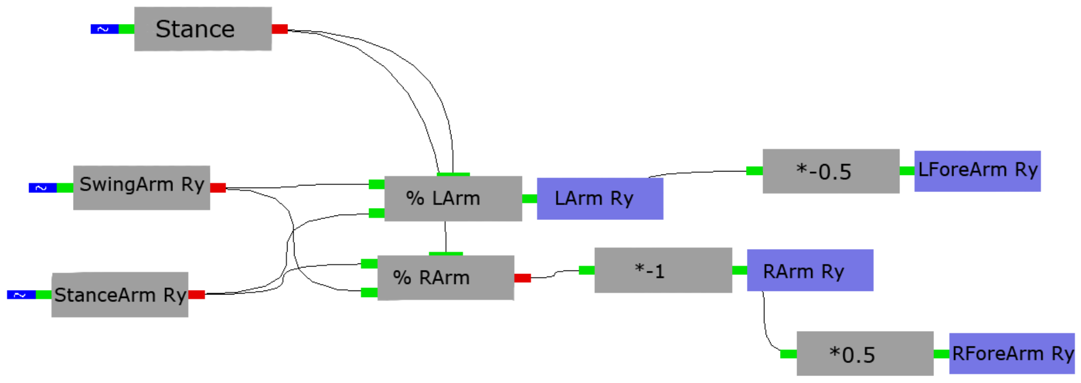


Figure 4.11: Trajectories to control arm swing.

force controller. These include the *stance swing ratio* and the *toe heel ratio*, which define the desired contact state of the character. The desired *forward velocity* is routed to the *balance controller* and the X rotation of the torso gives the character an initial lean in the forward direction.

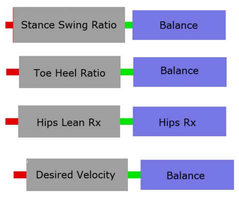


Figure 4.12: Constant values for the virtual force controller.

To generate the Simbicon-like [13] feedback rules we first determine the sagittal and coronal offset angles based on the current velocity and the offset of the COM to the stance location. These values are multiplied by gain parameters and routed to one node for the sagittal plane and to another node for the coronal plane. Figure 4.13 shows the feedback network for the sagittal plane.

The feedback value is sent to either the right or left leg in Figure 4.14 (the sagittal plane) depending on the step function in Figure 4.5. The same value is then scaled and added to the torso orientation.

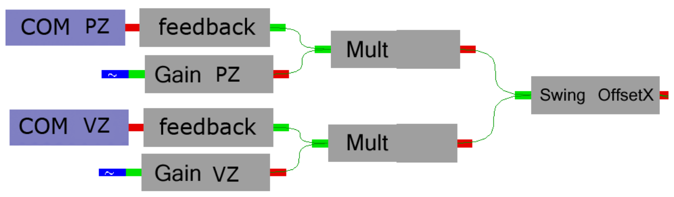


Figure 4.13: Feedback Terms.

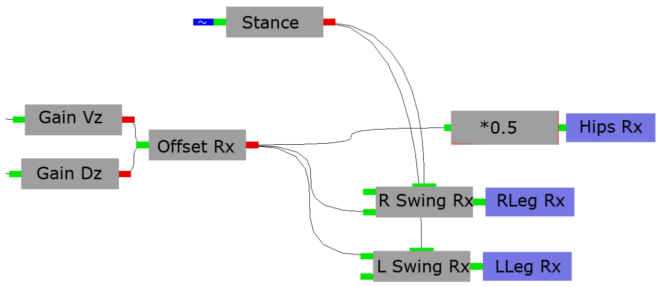


Figure 4.14: Sagittal Control.

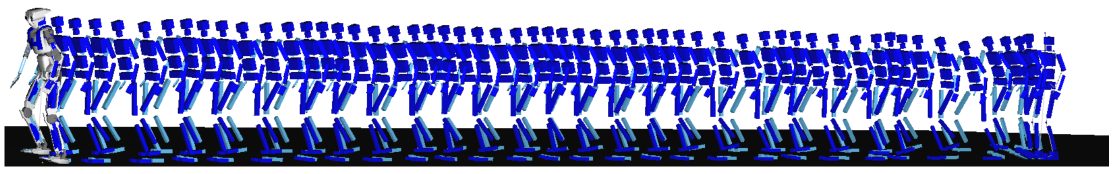


Figure 4.15: Example walk from completed network.

Figures 4.5- 4.14 demonstrate the typical operations needed in order to design controllers involving walking. While the presented case study is specific for walking with disturbances, the same operations can be extended for different new styles of walking or other forms of physics based motion. The resulted network can then be integrated into a higher level control framework in order to make sequences of controllers to have more complex behavior. When the entire network presented in Figures 4.5- 4.14 is put together the resulting graph is shown in Figure 4.16 and the resulting motion is shown in Figure 4.15.

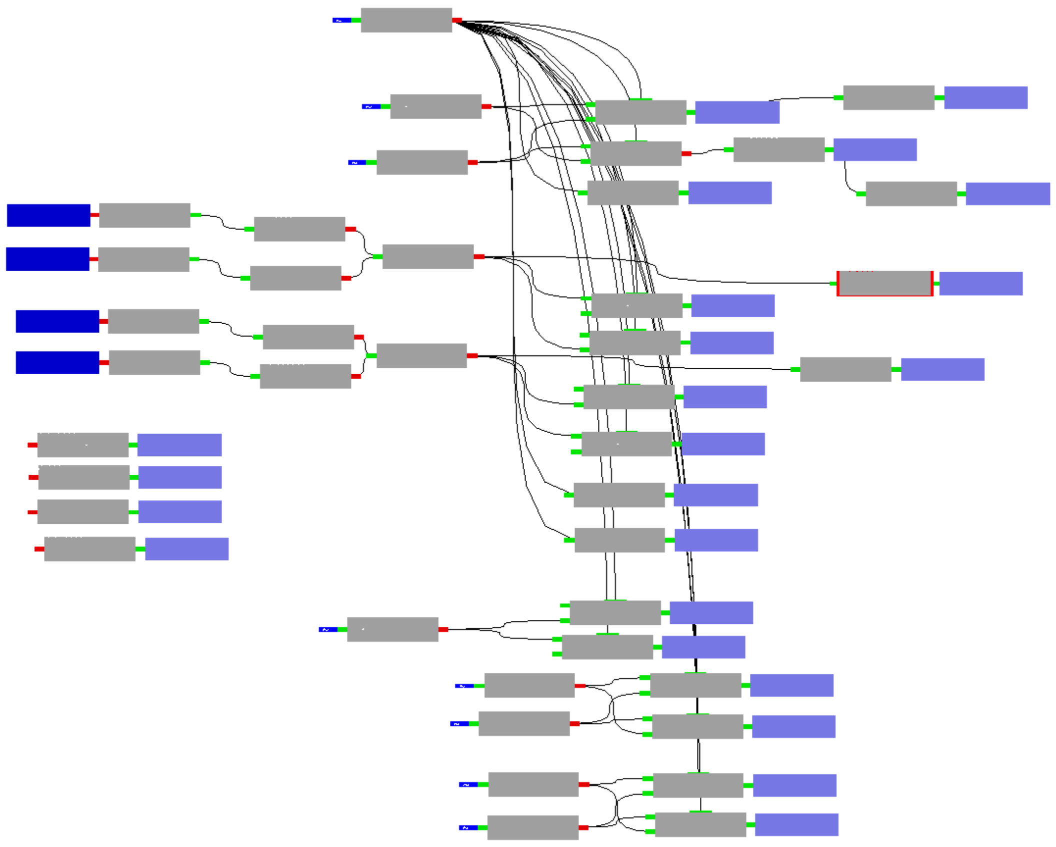


Figure 4.16: Topology of the complete network.

Our prototype system is not yet ready to be used by novice users. With expert knowledge, the development of a working jump controller can be completed in under 20 minutes. The walking controller described in this section takes about 35 minutes. In addition to developing controllers the system has showed to be very useful for understanding and visualizing the effects of all terms of a controller, what indicates great potential for educational use. As future work, we intend to develop a comprehensive user study in order to better understand the bottlenecks in developing controllers with the proposed operations.

CHAPTER 5

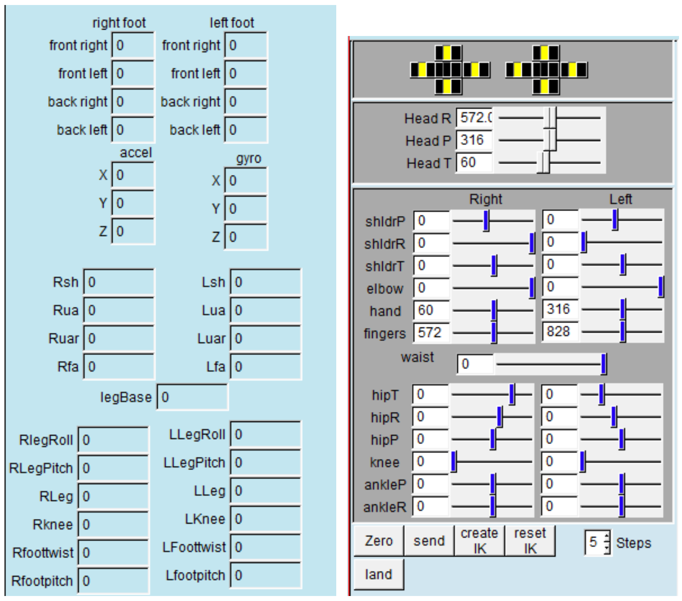
Application to Robotics

Research in the field of physics based character animation is closely related to humanoid robotics. While there are discrepancies between a simulation environment and execution on a real robot the underlying control strategies are similar. However there are some difficulties in transferring the knowledge between the domains. First of all, humanoid robots are typically very expensive and often delicate so it is difficult to learn new skills directly on the robot without the risk of damaging it. Additionally the robot is limited on its ability to sense the environment and its own configuration to guide it's control policy.

To explore the feasibility of humanoid robot control I built a system to control the Fujitsu HOAP-3 humanoid robot illustrated in Figure 5.2. This work was done in collaboration with Yazou Huang. I made two primary contribution to this work.

My first contribution was building the robot model and simulation environment. The original model that came with the robot was based on a messy conversion from a CAD file and had a very large number of polygons, even polygons for internal components that were not needed. So I used this model as a reference in Maya to sculpt a much lower resolution mesh that was much more optimal for out needs. I also modeled several objects representing real world obstacles that we wanted to track and put in the simulation.

My second contribution to this work was developing the system architecture for controlling the robot. This consisted of 3 separate threads that had to be


right foot
front right 0
front left 0
back right 0
back left 0

left foot
front right 0
front left 0
back right 0
back left 0

accel
X 0
Y 0
Z 0

gyro
X 0
Y 0
Z 0

Rsh 0
Rua 0
Ruar 0
Rfa 0

Lsh 0
Lua 0
Luar 0
Lfa 0

legBase 0

RlegRoll 0
RLegPitch 0
RLeg 0
Rknee 0
Rfoottwist 0
Rfootpitch 0

LLegRoll 0
LLegPitch 0
LLeg 0
LKnee 0
LFoottwist 0
Lfootpitch 0

Head R 572.(
Head P 316
Head T 60

	Right	Left
shldrP	0	0
shldrR	0	0
shldrT	0	0
elbow	0	0
hand	60	316
fingers	572	828
waist	0	
hipT	0	0
hipR	0	0
hipP	0	0
knee	0	0
ankleP	0	0
ankleR	0	0

5 Steps

Figure 5.1: GUI components for getting sensor data(left) and sending individual DOF commands(right).

in constant network communication. There was the Real-time Linux layer that directly actuated the robot motors based on a queue of postures. The next layer up was a nonreal-time thread that would listen for tcp packages containing robot commands and hand them to the real-time thread. This thread would also gather sensor data from the real-time layer. The final layer was the user interface layer that was compiled as a library for easy integration into various research projects. With the source code for the interface layer was a comprehensive GUI(Figure 5.1) developed with FLTK to aid in the connection and debugging of the different components.

This was integrated into Yazou's footstep pattern generator. I also developed a footstep pattern generator(Figure 5.6) that was able to find paths with clearance(Figure 5.3) but in the end we used his system that incorporated online adjustments based on the pressure readings from the foot sensors. The result was able to determine an environment model and robot position using our Vicon motion tracking system as show in Figure 5.5.

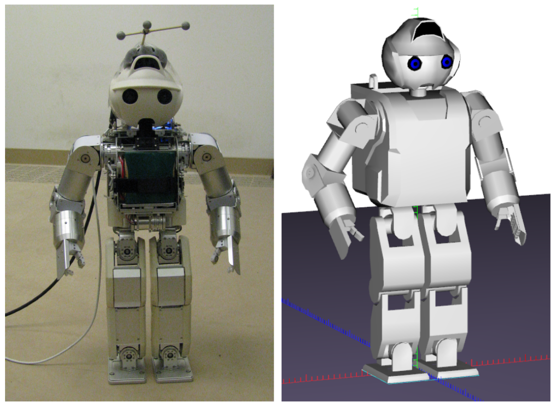


Figure 5.2: The HOAP-3 humanoid robot (left) and its virtual model (right).

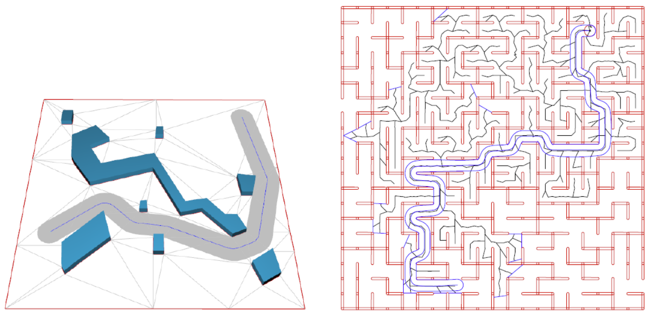


Figure 5.3: The left image illustrates a computed collision-free path and channel with given clearance radius. The right image shows the expanded nodes of the A* search during the path computation.

5.1 System Architecture

There are three components to the network interface controlling the robot: a user interface client application, a server application running on the robots on board computer, and a real-time module that has dedicated access to the motor control board. The user interface is running on a dedicated computer, and is where the motion is generated. For each frame of motion the posture of the robot is converted

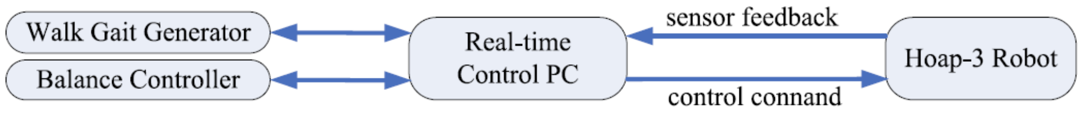


Figure 5.4: Overall system architecture for controlling the robot.

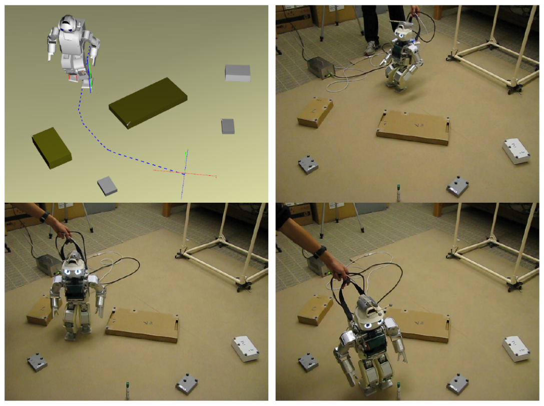
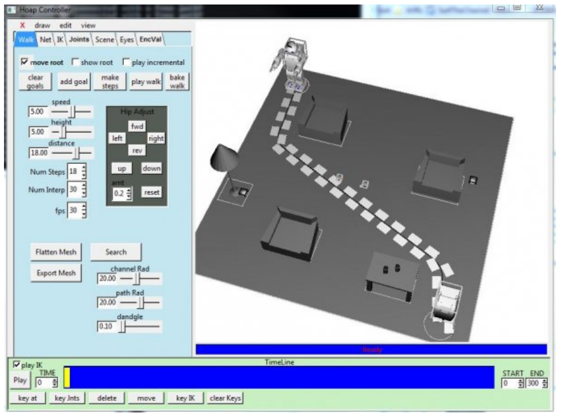


Figure 5.5: A path planning and execution example.



☑ move root ☐ show root ☐ play incremental
speed 5.00
height 5.00
distance 18.00
Num Steps 18
Num Interp 30
fps 30
amt 0.2
channel Rad 20.00
path Rad 20.00
dandgle 0.10
☑ play IK
TIME 0
START 0 END 300

Figure 5.6: Footstep generator using the path planning funnel algorithm.

from joint angles to encoder values and is put into a message struct along with the step size for the real-time module. The main method to control the overall speed of the robot is by adjusting the step size for each posture sent. The message is then sent over the network to the server on the robot's on-board computer. When the server receives a posture message from the UI it relays it directly to the real-time module. When the real-time module receives a message it places it into a circular queue. The real-time module starts interpolating the postures and sends each increment to the motor control board. For each step the real-time module pulls a posture from the queue along with the step size. The number of interpolation steps is determined by the largest difference in motor positions between the current and next posture divided by the step size. The increment for each motor is given by the difference in position divided by the number of steps. This ensures that for each move all of the motors start and end at the same time and the overall motion is continuous. Each time a posture is taken from the queue a data log is created that includes the pressure sensors in the feet,accelerometers,gyros and the corresponding encoder values. This information is then sent back through the server to the user interface for display and optimization. There is also a separate computer which is dedicated to tracking the robot and any obstacles. A separate server is set up on this computer which enables retrieval of the global position and orientation of the robot and the obstacles any time that it is queried. With the combination of the sensor data log from the robot and the global position and orientation from the Tracking system an accurate depiction of the robot and its environment can be made in real-time.

5.2 Results

We were successful at developing a walk generator for the HOAP robot and we learned a great deal about some of the difficulties. First of all the method used a projection of the COM to maintain static balance but this limited the maximum speed of motion to be rather slow. Additionally we found that the surface of the floor had a serious impact on execution. On carpet the robot had a hard time balancing since the compliance of the ground allowed the robot to lean over and fall even with a flat foot. Using a stiffer surface helped the balance but the surface tended to allow the feet to slide so that there would be deviation from the footstep plan. Finally there was the overall safety of the robot. Since it is a rather expensive piece of equipment we were hesitant to let the robot walk on its own so it required someone to always be holding the safety strap. This has motivated me to develop a physics simulation for the robot and persuaded me to focus more on simulation, where a robot model could fall over countless times without causing any damage.

CHAPTER 6

Learning Hand Configurations Through Back-propagation of an Artificial Neural Network

Here we explore a method to determine hand configurations using the depth image provided from an Xbox Kinect and the sensor outputs from a 5DT data glove by training an artificial neural network with back-propogation. The motivation is to allow a user a multi dimensional interface to control a physics based character. Since a humanoid character has so many different DOF using hand gestures would allow a user to have a greater degree of control than a more standard interface device like a mouse or keyboard. We are curious about different mappings between hand configurations and character configurations and are further motivated by the possibility of using the system for hand therapy. To interpret hand configurations using the Kinect a comprehensive pipeline was developed to capture,store data to train a neural network machine learning algorithm. The system is shown to have promising results and is demonstrated in a real-time application.

6.1 Introduction

A database of depth images was collected from an XBox Kinect along with joint angle measurements from a data glove to train a two layer artificial neural network. This project analyzed different parameters of the neural network and image capture and processing to find the optimal parameters to minimize the root mean square error of a testing set while maintaining real-time performance of the fully trained system.

6.1.1 Motivation

The UC Davis Medical Center has a pediatric unit that specializes in children with burns on their hands. The children are typically between 6 and 12 years old and the severity of the burns are varied. Some have additional complications such as missing fingers, heavy bandages or other ailments. The doctors in this clinic have a set of protocols (hand motions) that they encourage the children to exercise during physical therapy. These protocols have multiple functions for both treatment and diagnosis. The protocols are designed to stretch the skin to help restore the flexibility that has been lost due to the burns and are also a measure for the progress the patient is making towards recovery. During the therapy the doctors make note of the progress by scoring each protocol from 1-10 depending on how far into the protocol the patient is able to go. This measurement is done by eye and the readings are taken at intervals of several days during the therapy sessions. Since the measurements are taken by eye there is bound to be error which the doctors desire to reduce by using some sort of measurement device. However, more standard approaches to measure the hand configuration are not feasible since the burns prevent anything from being attached to the hands. Another issue with the current method of physical therapy is that the protocols themselves cause pain in the patients. Since the children are so young, often times they are unwilling to perform the exercise since it only causes pain with no reward. To address this issue we want to incorporate a video game into the therapy that is driven by the progress for each protocol. The theory is that making a reward for achieving a protocol will distract the children from the pain and expedite the treatment. Additionally we would like this system to be cheap enough so they can take it home with them to perform the exercises out of a clinical setting.

6.2 Related Work

Since this topic is a bit of a tangent to my research in physics based characters we will look separately at the related research.Tracking hand motions is a well researched field, and many approaches have been taken. There are several commercial devices that can easily measure joint angles such as the data glove that is used for this project [25] however this cannot be used for the final system since we cant put a glove on the childs hand. There are also several devices that are still under development such as the finger tracker prototype by Microsoft Research [26] but this also requires something to be attached to the hands. An interesting product called Leap Motion is soon to be released [27] but it is unclear how it works and whether it will offer the resolution that we desire.

Researchers in academia are also trying to solve this problem such as [28] where they use a colored glove and a standard webcam to reconstruct the joint angles of the fingers. Argos et al. [29] show impressive results using a Kinect to reconstruct hand configurations, but this system does not generalize well and requires calibration for each user. Additionally, the system requires the hand to be in an initial configuration and performs a computationally expensive continuous optimization procedure that requires the hand to move relatively slow and always stay within the capture region. It is also prone to local minima when occlusions occur.

Perhaps the closest system to what we desire is Three Gear [30] a system that uses two Kinect cameras to reconstruct hand postures; however, this requires a lengthy calibration phase and is limited to the small set of postures that are provided within the system. Additionally, since two Kinects are used, the system tends to overwhelm the operating system, requiring an extremely fast computer and still preventing other processes from performing well. We are interested in a customizable contact-free hand measurement device that can generalize through different hand shapes and sizes. However, we are more flexible in the overall generality of the system and favor better precision and performance over things such as global hand position and orientation tracking.

6.3 System Overview

The system developed for this project has three main operating modes: Data Collection, Training and Operation. The data capture mode is where the training/testing data is generated and formatted for the learning algorithm. The training mode allows a user to generate a neural network, adjust the parameters and train the system. The operation mode allows the results to be verified by using the fully trained neural network in a real-time application. The physical setup of the Kinect is show in Figure 6.1 The Kinect is mounted to a tripod looking down at a chair where the user will sit. All user interfaces are built using Fast Light Tool Kit (FLTK) [31].

6.3.1 Data Capture

Since the data for this project is to be collected,instead of relying on a preexisting dataset, a fair amount of time was spent developing the pipeline to capture, process and store the data. A user interface was developed to facilitate this process shown in Figure 6.2

When the Kinect is first initialized and the point cloud is generated, the entire field of view has a mesh generated including the background by connecting neighboring pixels in the depth image with 3D triangles where we use the camera calibration to project the 2D pixels with their depth values to 3D points in space. Using a depth sensor allows the background/foreground to be segmented easily by simply thresholding the depth values and only admitting points within a certain range. The procedure is rather simple; the user places their hand in front of

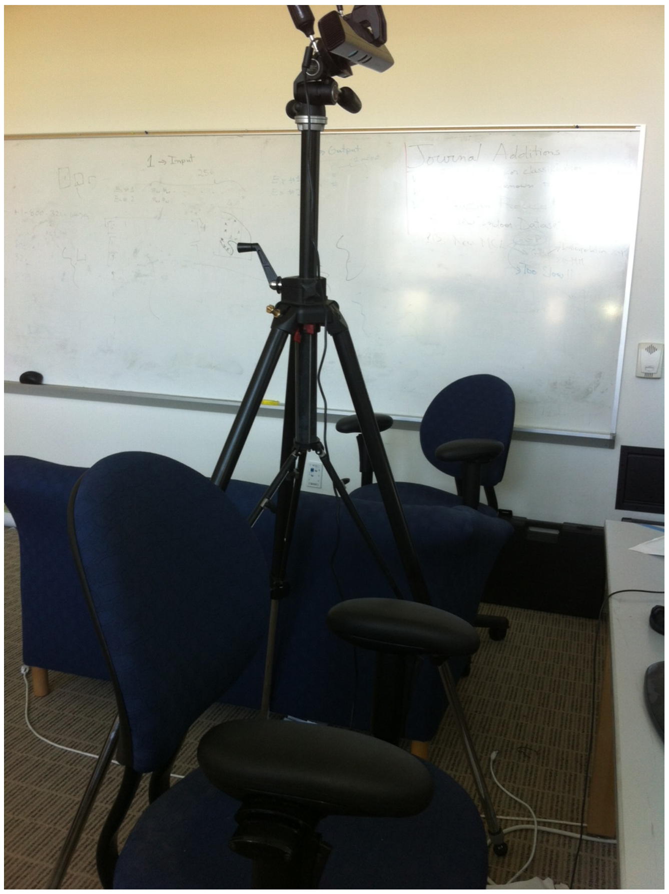
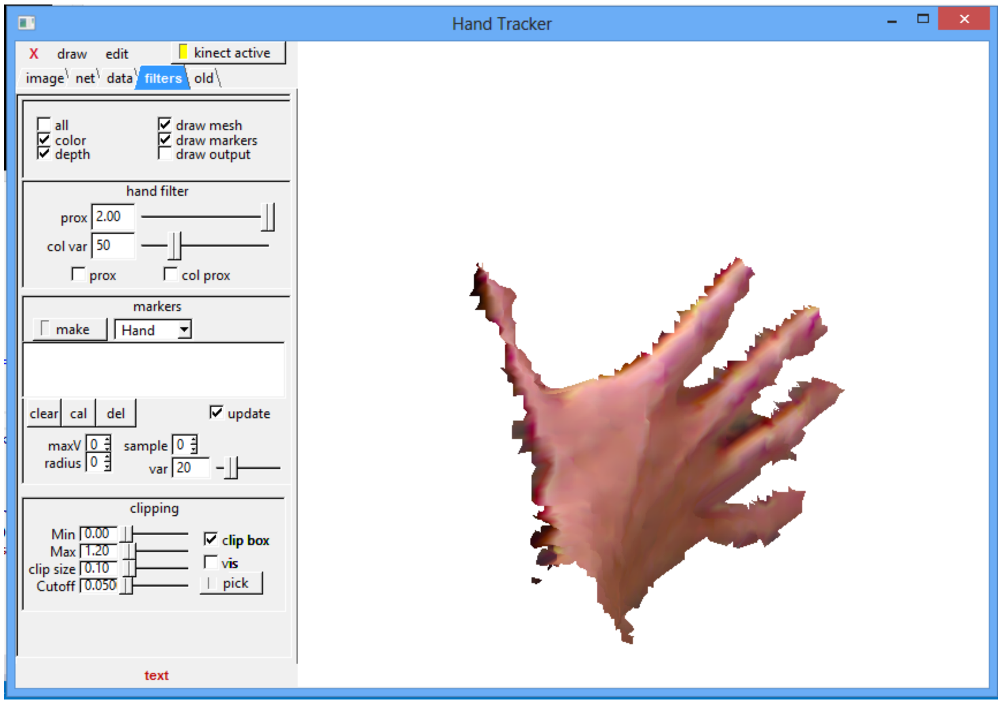


Figure 6.1: Kinect mounted to tripod next to user



☐ all
☑ color
☑ depth
☑ draw mesh
☑ draw markers
☐ draw output
prox 2.00
col var 50
☐ prox
☐ col prox
☐ make
☑ update
maxV 0
radius 0
sample 0
var 20
Min 0.00
Max 1.20
clip size 0.10
Cutoff 0.050
☑ clip box
☐ vis

Figure 6.2: Pointcloud converted to polygon mesh

the camera in a comfortable position then they enter a bounding region selection mode by pressing a button. They click the viewer on the hand and this casts a ray intersecting the geometry and detecting the intersection point. This is then the center for the cubic capture volume. Then the size of the capture volume can be adjusted with a slider. After this has been setup the system performance is greatly increased since only points within this volume are considered.

After the points have been segmented they are projected back to a 2D image(see Figure 6.3) since this is a compact way of storing the information. Projecting the point cloud back to 2D like this does not lose any information since the initial capture is only the camera facing shell of the actual object. Each image is cropped to fit the maximum horizontal and vertical dimension of the hand.

To get the ground truth hand configuration a data glove is used to measure the joint angles of the hand. Several methods of storing these hand configurations were explored: The initial approach was to just save all the joint angles; however,

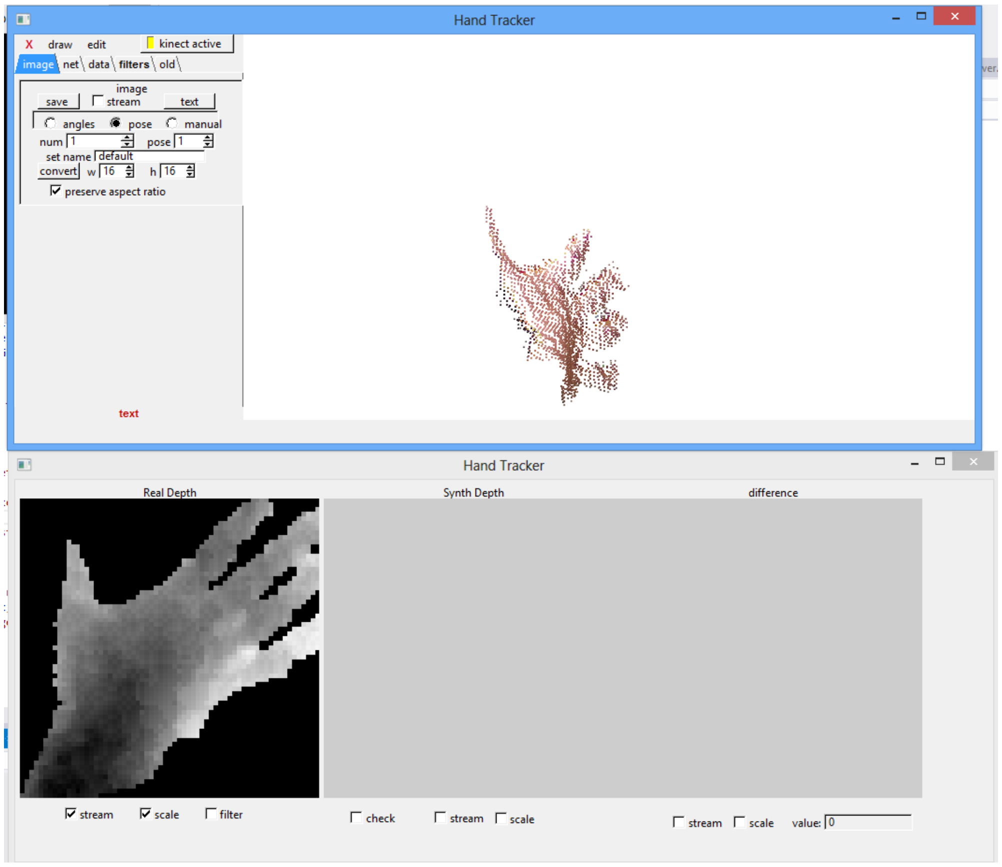


Figure 6.3: Image conversion UI

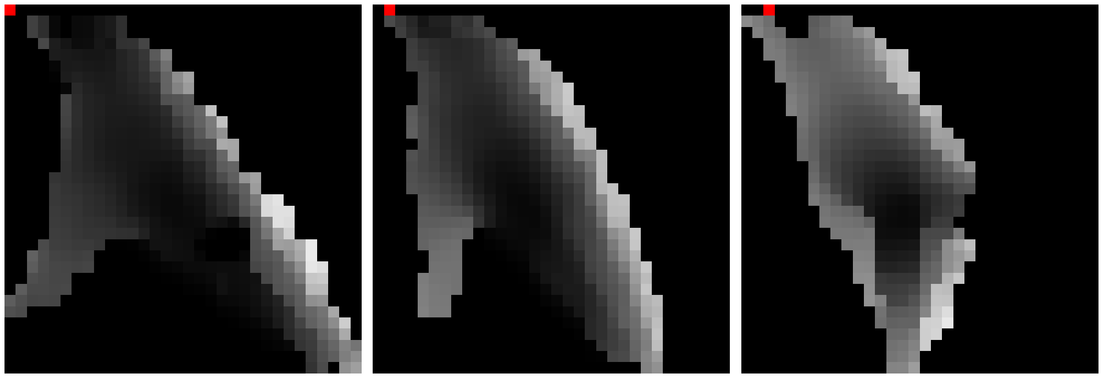


Figure 6.4: Example output images from data capture. Red dots on top row represent the output vector.

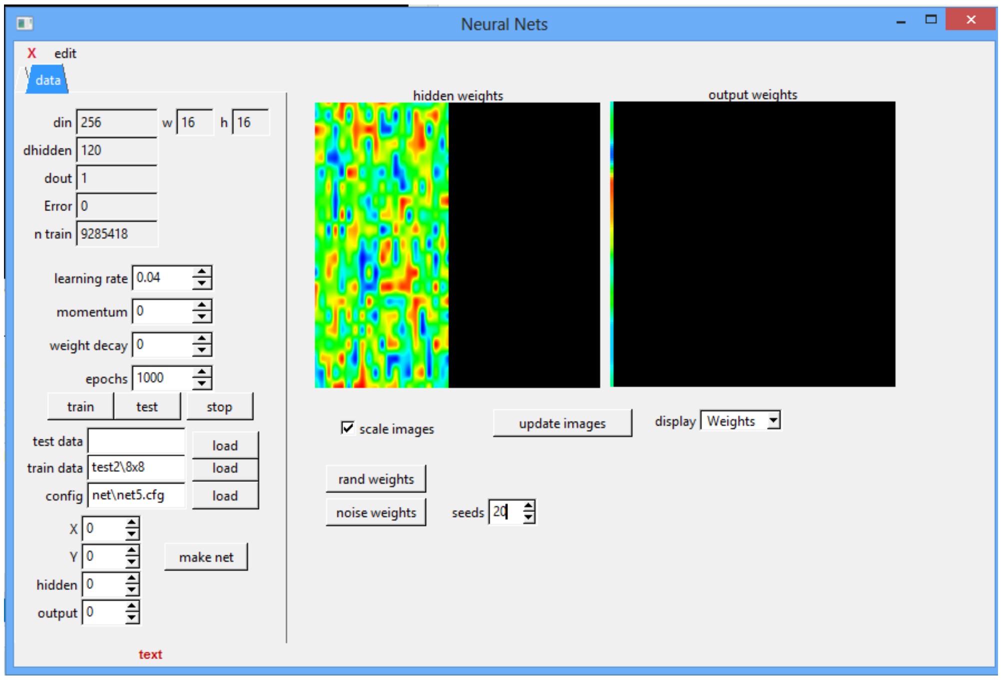
this was determined to not be optimal since the large number of degrees of freedom (DOF) prevented the neural network from converging well. An extension of this was developed where a user would show two hand postures which represent the extremes of the motion. For example: hand fully opened and fully closed. Then at capture time these two postures were used as an interpolation goal to output a single floating point number which represent relative progress through the motion. In our example a value of zero was hand fully opened and one was hand fully closed. This was also found to be less than optimal since it was not clear that we would capture a uniform distributuion of the hand postures. The final encoding method for the joint angles was to segment the output value into a discrete set of classes and create a binary vector representing the motion. For example, with 5 bins a fully open hand would be [1,0,0,0,0] and fully closed would be [0,0,0,0,1]. For convenience's sake the output values were stored along with the depth image captured at the same instant. The output value is stored as the top row of red pixels in the image Figure 6.4.

6.3.2 Image Processing

When the input image stream is set up the images still need to be formatted so they all have the same dimensionality. Additionally the images are reduced in size so as to minimize the dimensionality of the input vector. To do this, first the user specifies the desired dimension for the images (for example 32x32) Then each image in the captured set is scaled to fit that region maintaining the correct aspect ratio. Before down-sampling, an edge preserving bilateral filter [32] is applied to the image to approximate the average value of the pixels depth value while maintaining the edges for each sub-sampled pixel.

6.4 Neural Net

The process of creating and tuning the neural network is aided by an additional UI(Figure 6.5). Here the user can specify the input dimensions, the number of hidden units, and the output dimension. Additionally, the user can specify the learning parameters such as the learning rate, momentum, number of epochs and weight decay. Another feature was explored to initialize the weights of the perceptrons with either a typical noise function or something I discuss more later: a Perlin noise function [33].



din 256 w 16 h 16
dhidden 120
dout 1
Error 0
n train 9285418
learning rate 0.04
momentum 0
weight decay 0
epochs 1000
test data
train data test2\8x8
config net\net5.cfg
X 0
Y 0
hidden 0
output 0
☑ scale images
display Weights
seeds 20

Figure 6.5: Neural Net user interface.

6.4.1 Training

After an adequate database of training and testing images is collected and the neural network is created, the training phase can begin. Initially, all of the network weights are initialized based on a randomization function and user specified bounds. Each pattern from the training set is presented to the input nodes of the neural network and then propagated through the network to give a specified output. Then the error between the network output and the output specified by the data glove is used as the driving force for the back propagation weight update using the generalized delta rule from chapter 5 of [34]. Each epoch randomizes the order of the patterns and the updates are updated after each pattern. The system is coded with C++ and a typical training of a neural network with 1200 training patterns and 300 testing patterns using a1024 input vector(32x32 pixel image) 200 hidden units and 7 output units takes about 30 min to converge with 30 epochs.

6.4.2 Error Measurement

To measure the performance of the system the RMS error of the test set is used. From experimentation it seems that the training RMS error always converges very fast but the test set error seems to converge slower. Once the training error has reached close to zero the system is no longer able to learn, since it is this error that drives the generalized delta rule weight updates. Using this measure, several of the parameters were modified to see how this affects the system performance. For all the following examples an image of 32x32 pixels was used with 7 output units.

Many system configurations were tried with variations in the number of hidden units and different training parameters, Figure 6.6 shows a typical convergence rate of the training. After several trial configurations were examined, a system with 200 hidden units was chosen. Since run-time performance is desired a small increase in the error is justified by the fewer nodes to evaluate.

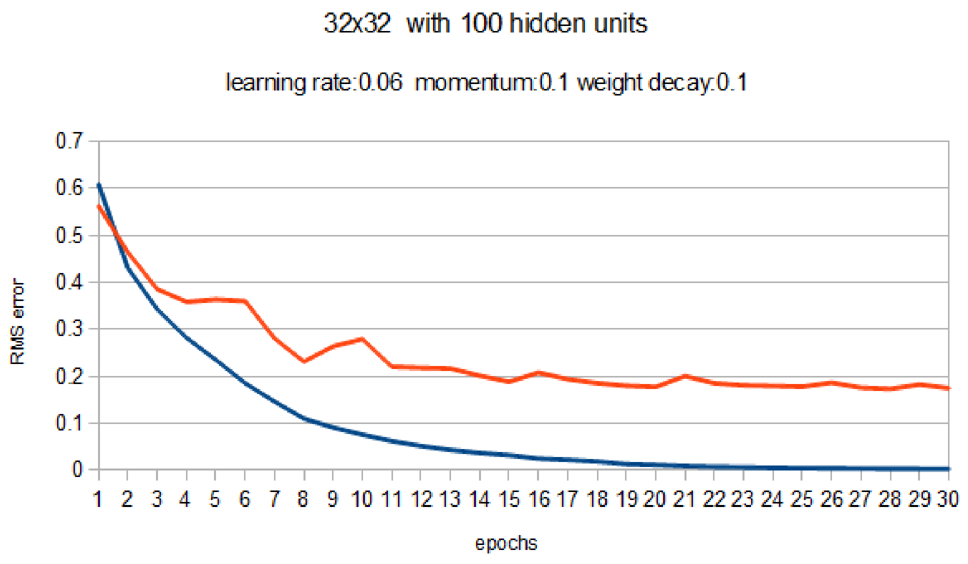


Figure 6.6: A typical plot of the convergence rate of the training.

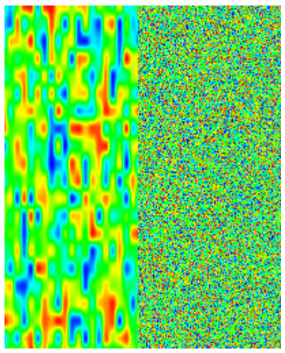


Figure 6.7: Perlin noise(Left)vs random noise(Right).

6.4.3 Effect of changing noise function

The typical approach to seeding a neural net is by using a random number generator bounded by some min/max value. But out of curiosity a Perlin noise function was used to seed the initial weights of the network(Figure 6.7). The justification for seeding the network in this manner was that the Perlin noise still uses a uniform distribution of random values but it does so where the random numbers are correlated with the neighboring pixels. That is because the value of each unit is a function of the row and column of the weight matrix for each layer. An additional parameter is used to specify the number of seeds or nodes in the noise function. This function was applied to both the hidden layer and the output layer and 10 seed nodes were specified for the noise function. The hope was that this would be just an interesting way to visualize the changes in weights(Figure 6.8), but it was shown to affect the performance as well which is explored next.

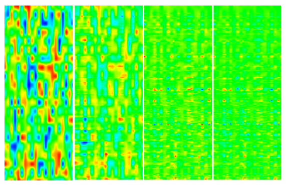


Figure 6.8: Evolution of weight values using Perlin noise.

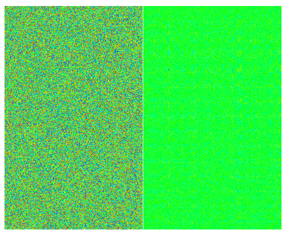


Figure 6.9: Evolution of weight values using random noise.

6.4.4 Performance of different noise function

To test the effect of using this noise function, two identical tests were performed changing only the weight initialization.One network uses Perlin noise(Figure 6.8) and the other uses random noise (Figure 6.9) Using a network with 300 hidden units momentum 0.1 and decay 0.1 the only thing that was changed was the seeding function in this case with 20 nodes for the Perlin seed. As can be seen below (Figure 6.10 and Figure 6.11) there is actually a rather dramatic change by simply changing the noise function. The system converges much faster and results in a final RMS error of 0.14 for the Perlin noise and 0.23 for the randomized noise. Further test could be done to explore this affect but maybe clustering the weights in the manner would create distributed processing units within the network that are targeting certain parts of the image allowing more generalization, instead of single points spread throughout the image.

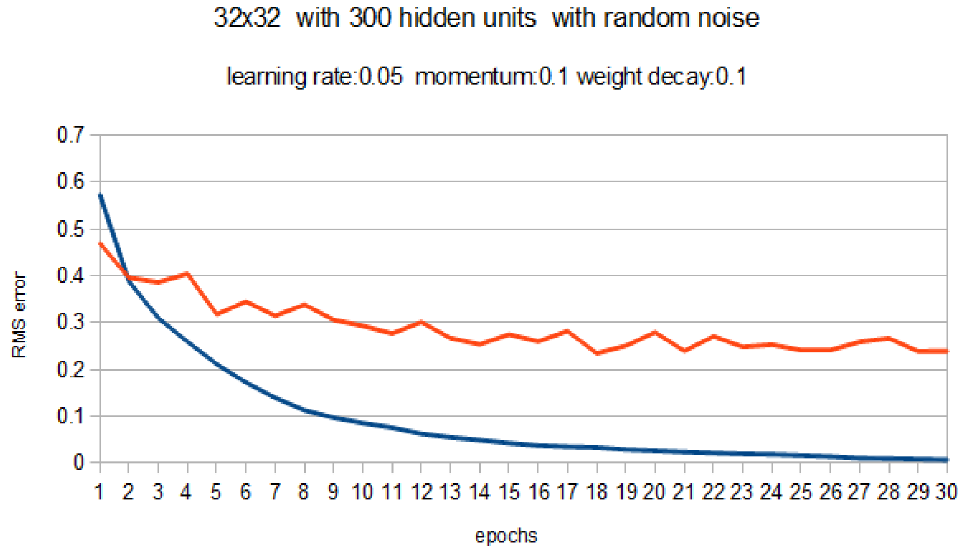


Figure 6.10: Error convergence of random noise.

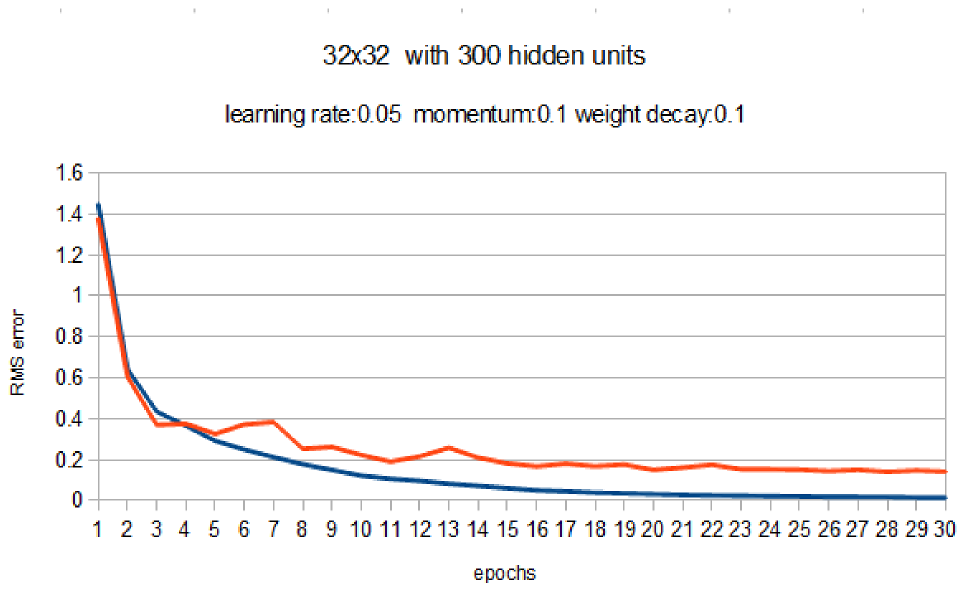


Figure 6.11: Error convergence of Perlin noise.

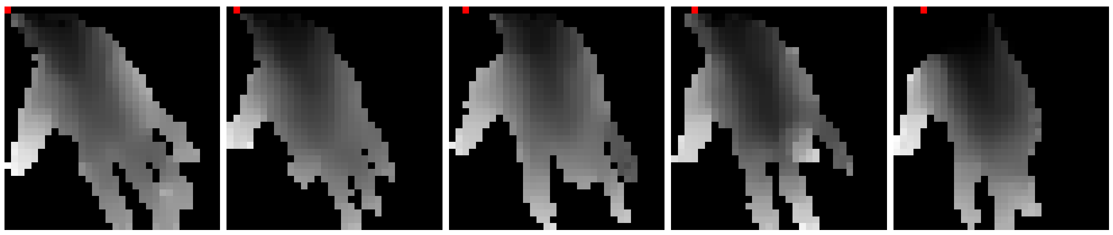
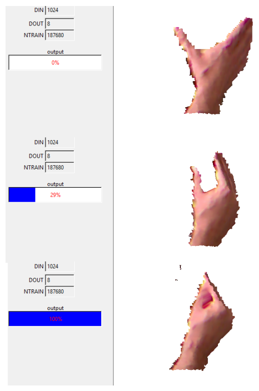


Figure 6.12: Each class represents a different finger.

6.4.5 Additional test

To test the generality of the system an additional protocol was tested but with less extensive analysis. In this case(Figure 6.12) there are five classes of hand configurations that are the outputs; one for all fingers open and the then each additional one has a single finger bent. This would not really be sufficient for the protocols but it is an interesting test of the robustness. In the end the test set error didnt converged below 30 percent error but perhaps additional methods could be used to augment the learning process.

6.5 Operation


DIN 1024
DOUT 8
NTRAIN 187680
output
0%

DIN 1024
DOUT 8
NTRAIN 187680
output
29%

DIN 1024
DOUT 8
NTRAIN 187680
output
100%

Figure 6.13: Trained network running in a realtime environment.

Using the RMS error is useful in terms of training convergence but to get a real sense of the functionality, the fully trained system was used for real-time evaluations using the Kinect. In the Image (Figure 6.13) the blue output bar represents the predicted protocol value from the neural network and the image on the right is the captured hand image. The system is far from perfect and there is still a good amount of noise but as a proof of concept it seems to work.

6.6 Future Work

There are many interesting direction that this project could be explored. One is using a more sophisticated method to encode the depth image instead of simply using the intensity values of the pixels. A Histogram of Oriented Gradients [35] has been used to detect faces in images but the method basically stores the gradient information at each pixel, using the gradient would seem to make sense since it is then capturing the geometry information and would expose correlations in the neighboring pixels. Additionally there are many feature extraction methods such as SIFT [36] that could potentially reduce the dimensionality of the image. Since the output of the system is still rather noisy the output value could be oversampled or put through a sort of low pass filter to make smoother results. Another interesting thing to explore would be how to encode the output values perhaps instead of using a binary vector for the output some sort of Gaussian distribution can represent the target values which might reduce the output noise further. Another interesting thing to explore would be to use multiple Kinects to get a better image of the hand, however this would likely reduce the performance substantially and would likely introduce other alignment issues.

6.7 Conclusion

The system demonstrates hand gesture recognition using an Xbox Kinect. The system was able to achieve a convergence of 14% RMSE on the testing set and was verified in a real-time application.

CHAPTER 7

Conclusion

I have presented in this thesis a system that allows users to create controllers for physically simulated characters without low-level programming. This system introduces a new methodology to explore, edit and create parameterized physics based feedback controllers that can be later used in real-time applications. This system was shown to be general enough to work for both walking and jumping but there is nothing that would keep it from working for various other skills of interest. A comprehensive GUI was developed that allows a user to design these controllers by simply dragging and dropping nodes onto a window then dragging between the nodes to create connections defining the functionality.

The presented system has the potential to be used for the rapid prototyping and customization of physics behaviors designed to improve game-based therapeutic applications integrated with the XBox Kinect and the results obtained so far are promising.

Acknowledgments The work from Chapter 6 was partially supported by CITRIS grant 128.

References

[1] Sergey Levine, Jack M. Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 31(4):28, 2012.

[2] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. Feature-based locomotion controllers. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 29(3), 2010.

[3] Zoran Popovic and Andrew P. Witkin. Physically based motion transformation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 11–20, 1999.

[4] Jessica K. Hodgins and Nancy S. Pollard. Adapting simulated behaviors for new characters. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 153–162, 1997.

[5] Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. Limit cycle control and its application to the animation of balancing and walking. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 155–162, New York, NY, USA, 1996. ACM.

[6] Jerry Pratt, Chee-Meng Chew, Ann Torres, Peter Dilworth, and Gill Pratt. Virtual model control: An intuitive approach for bipedal locomotion. *The International Journal of Robotics Research*, 20(2):129–143, 2001.

[7] Jerry E. Pratt and Russ Tedrake. Velocity-based stability margins for fast bipedal walking. In *In Fast Motions in Biomechanics and Robotics*, pages 299–324. Springer, 2006.

[8] Adriano Macchietto, Victor Zordan, and Christian R. Shelton. Momentum control for balance. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 28(3), 2009.

[9] Victor Zordan, Adriano Macchietto, Jose Medina, Marc Soriano, and Chun chih Wu. Interactive dynamic response for games. In *Proceedings of the ACM SIGGRAPH symposium on video games*, 2007.

[10] Sungeun Kim Yoonsang Lee and Jehee Lee. Data-driven biped control. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 29(4), 2010.

[11] Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. Sampling-based contact-rich motion control. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 29(4), 2010.

[12] Marco da Silva, Yeuhi Abe, and Jovan Popović. Interactive simulation of stylized human locomotion. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 82:1–82:10, New York, NY, USA, 2008. ACM.

[13] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: Simple biped locomotion control. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 26(3), 2007.

[14] Yao-Yang Tsai, Wen-Chieh Lin, Kuangyou B. Cheng, Jehee Lee, and Tong-Yee Lee. Real-time physics-based 3d biped character animation using an inverted pendulum model. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):325–337, March 2010.

[15] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *ICRA*, pages 1620–1626, 2003.

[16] Jerry E. Pratt, John Carff, Sergey V. Drakunov, and Ambarish Goswami. Capture point: A step toward humanoid push recovery. In *Humanoids*, pages 200–207, 2006.

[17] Uldarico Muico, Yongjoon Lee, Jovan Popović, and Zoran Popović. Contact-aware nonlinear control of dynamic characters. *ACM Transactions on Graphics*, 28(3), 2009.

[18] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Generalized biped walking control. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 29(4), 2010.

[19] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 251–260, New York, NY, USA, 2001. ACM.

[20] Stelian Coros, Philippe Beaudoin, KangKang Yin, and Michiel van de Panne. Synthesis of constrained walking skills. *ACM Trans. Graph. (Proc. Siggraph Asia)*, 27(5), 2008.

[21] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Robust task-based control policies for physics-based characters. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 28(5), 2009.

[22] Sumit Jain, Yuting Ye, and C. Karen Liu. Optimization-based interactive motion synthesis. *ACM Transaction on Graphics*, 28(1):1–10, 2009.

[23] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics*, 30(4):Article TBD, 2011.

[24] Libin Liu, KangKang Yin, Michiel van de Panne, and Baining Guo. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Trans. Graph.*, 31(6):154:1–154:10, November 2012.

[25] 5dt data glove, 2012.

[26] Microsoft hand tracker, 2012.

[27] Leap motion, 2012.

[28] Robert Y. Wang and Jovan Popović. Real-time hand-tracking with a color glove. *ACM Transactions on Graphics*, 28(3), 2009.

[29] I. Oikonomidis, N. Kyriazis, and A. Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC 2011*. BMVA, 2011.

[30] Kin Wang, Twigg. Three gear kinect tracking system, 2012.

[31] Fast light toolkit, 2012.

[32] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 839–, Washington, DC, USA, 1998. IEEE Computer Society.

[33] Perlin noise wiki, 2012.

[34] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[35] Mario Rojas Quiñones, David Masip, and Jordi Vitrià. Automatic detection of facial feature points via hogs and geometric prior models. In *Proceedings of the 5th Iberian conference on Pattern recognition and image analysis*, IbPRIA'11, pages 371–378, Berlin, Heidelberg, 2011. Springer-Verlag.

[36] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.