Planning Dynamic Vehicle Motion using Move Trees

by

Andrew Adam

B.Sc., The University of Essex, 2002

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

August 22, 2007

Abstract

Generating skilled and well-planned behaviours for autonomous agents is a challenging problem common to both computer animation and robotics. This thesis presents a system that uses motion graphs for online motion planning, resulting in skilled driving behaviours for a dynamic model of a car in a constrained environment. The result reproduces skilled driving behaviors. It is a particular challenge to get the cars to produce skidding-into-turn behaviors when approaching sharp corners, which can achieve the fastest speeds around a track. The techniques explored in this thesis are potentially generalizable to other dynamic vehicle behaviours, in computer games or simulations.

We demonstrate that a well-formed move tree or motion graph, created from the output of a physics-based simulation can be used to produce realistic steering behaviours on a variety of tracks. We show that a finite-horizon A* search algorithm is well suited to this task. We have produced a number of smooth animations that demonstrate considerable anticipation and agility, be it through acceleration/deceleration around tricky obstacles, or a hard skidding turn into a corner after approaching at high speed. Finally, we offer a number of ways that we could speed up the algorithms for future work in this area.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank Michiel van de Panne for considerable help when constructing this work, and a continual outpouring of ideas. It would not have been possible to undertake nor complete this work without his guidance. I would also like to thank Robert Bridson for being my second reader.

I would also like to thank Marco Monster of the now unfindable Monstrous Software for providing the code that implements the underlying car physics, as it was incredibly useful, and it is a shame that I cannot cite him directly. I also thank everyone who I did cite for allowing me to get up speed in this area of research.

Finally I would like to thank my friends and family who have supported me throughout even when it was making me miserable, in particular Katayoon Kasaian who had to face the brunt of my difficulties. :)

Chapter 1

Introduction

The animation and simulation of autonomous vehicles is a challenging problem, and the demand for this to be done realistically has increased in recent years. The topic has foundations in animation, artificial intelligence and robotics. In particular computer games have pushed the need for real-time generated synthesized animation. Autonomous vehicles will not look or act realistically unless they are based on some sort of physics model, which directly impacts the motions observed for high velocity systems. Self-driving vehicles are of interest to generate realistic simulations of any kind of traffic, and for safer automated roads where the reduction of human error could save lives.

In this introduction, we discuss why we would study vehicle motion and present the major goals of our work, along with the contributions that we have made. An outline of the remainder of this thesis concludes the chapter.

1.1 Motivation

1.1.1 Motion Control

Any system that is controlled, either simulated or real, to produce a desired motion, can broadly be described as performing *motion control*. If we look at human and animal motion, we see incredibly complex systems that work with a natural grace that is very difficult to reproduce mechanically. We have barely reached the point where we can get a two legged robot to walk in a graceful fashion, or place an item on a shelf, let alone reproduce the motion of a cat as it jumps from wall to wall, or something equally complex. The neuro-muscular

systems in human and animals are researched in the field of biomechanics, and we can try and apply these to our robotic designs. However, we might be more interested in the results than the implementation when studying in the field of animation, and so we can attempt to reproduce motions without necessarily modeling the complex underlying details.

There are many applications of motion control. We see robot arms in industries such as car manufacture. We need robots that can navigate dangerous environments such as deep underwater, or traversing the surface of other planets for research. In simulated systems, we have motion capture technology that has allowed us to reproduce human motion by attaching markers to various points on the body and recording their position in 3D space, as shown in Figure 1.1.

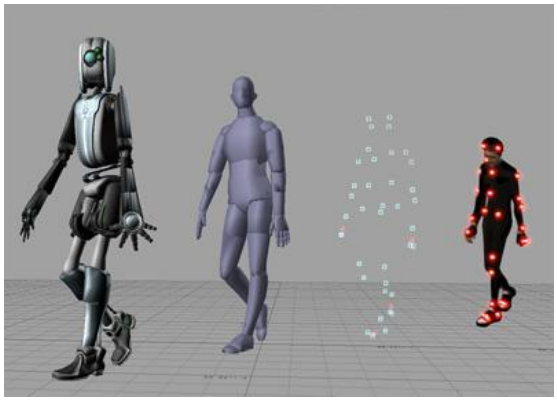

Figure 1.1: An actor has markers attached to his body, so that his movements can be recorded and then reproduced for animation. (Courtesy of Wikipedia.)

Motion capture data has allowed extremely lifelike animations to be produced in games, although is more difficult to use in dynamic environments, such as when two players collide in a sports game. Dynamic collisions such as the motion of bodies collapsing after death in a first-person shooter, are better modelled with ragdoll physics systems. Some recent systems have attempted to combine the data-driven approaches with physical simulation, as seen in Mandel[38].

1.2 Autonomous Vehicle Control

In this thesis our work is directly related to the field of *autonomous vehicle control*, for both real and simulated agents. The following two subsections look at vehicle research in both of these areas.

1.2.1 Real World Systems

Kanakia [26] writes that autonomous vehicles are being researched in the 2007 Defense Advanced Research Projects Agency (DARPA) Grand Challenge, where a car is sought that can navigate in a simulated urban environment for sixty miles in less than six hours, without human guidance. This has a $2 million top prize. For a previous challenge, a 132 mile route across the desert had to be navigated, with 5 finishers. This new challenge poses new problems, as now it will have to act reasonably in the face of other traffic. This could include a variety of new obstacles, such as curbs, holes, cars, bicycles and pedestrians, and entrants must also obey the traffic laws.

A car like this could save lives as it would have better reactions and senses as compared to human drivers, and it would never need to stop due to fatigue. It could drop you off at school or work and then park itself. It could also affect the speed of traffic, as cars could communicate with each other to maintain a faster group speed. This vision is a long way from reality at present, however cars are slowly becoming more autonomous, with systems such as anti-lock brakes, and we already have positioning systems such as GPS that can guide the driver.

Other recent automated vehicles include Roomba, a disc based robotic vacuum cleaner, and PackBot, a small tank-like battlefield robot that can climb stairs and recover when it falls over, as written by Clark [10]. PackBots have been used to explore enemy caves and to detect explosives. Clark [10] also writes about a current project called r-Gator, an unmanned jeep that can shuttle supplies to and from areas of combat.

It is not only on land where robots can perform tasks. They may even prove to be more useful for undersea operations, where humans find it difficult, and

at high depths, impossible, to penetrate and explore. [10] tells is about the *Autonomous Underwater Vehicles Laboratory* (AUV Lab), which has a vision of filling the void of the ocean with robots. They developed the Odyssey class of submarine-like vessels into AUVs that survey offshore oil fields and assist the U.S. Navy with mine warfare and battlespace preparation. The biggest challenge in this area is the provision of power, as most systems run on batteries.

In the air we have aircraft that are intelligent, and that can perform automated take off and landing. These can be more useful in areas where it is difficult for a land robot to enter, such as dense urban areas, or even simply over the sea. For many years guided missiles have effectively acted as autonomous airborne agents.

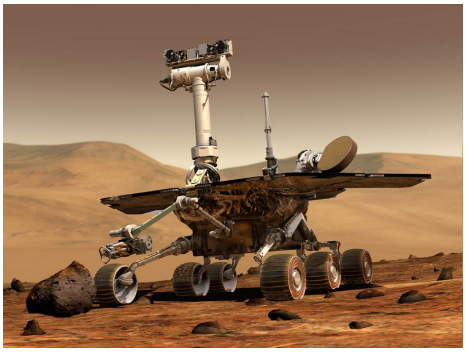

Figure 1.2: An artists impression of what one of the Mars Rovers could look like on the surface of the planet. (Image courtesy of NASA.)

An impressive ongoing project involves two robots on Mars (see Figure 1.2). The two rovers, named 'Spirit' and 'Opportunity' touched down in 2004 after having travelled 311 million miles. They have been functioning well for over 2 years, moving slowly across the difficult Martian terrain with directions sent from Earth. Interesting discoveries have been made as a result, such as the suggestion from chlorine and bromine deposits that Mars at one time had a salty sea.

1.2.2 Simulated Systems

Simulated systems can be seen in many different areas. Many computer games now exhibit considerable realism of human and vehicle motion, as seen in recent racing simulation GTR2(Figure 1.3), created by Simbin Development [1], which shows simulated vehicles. It is a realistic simulation of the FIA GT Series Championship and features many different cars and tracks. It can even be attached to MoTec telemetry software, which records real life car performances on race tracks, or Track-IR which can be attached to the head for panning around the cockpit. Game developers have been constantly trying to improve the physics model to make it as realistic as possible, and this version has improved tire physics at low speeds and better damage modelling. It also has AI routines that mean that that your opponents can be set to be tough or weak, conservative or aggressive, to tailor the game to your needs or ability level.

Figure 1.3: An image from GTR2, a game for the PC, that demonstrates simulated vehicles. (Permission of image use gained from www.strategyinformer.com)

Crash testing can be an extremely expensive process. It cannot be the only method used, they can be simulated. Kirkpatrick[28] developed a crash simulation model for the Ford Crown Victoria. He conducted different simulated

tests such as the front bumper rigid pole impact test, front door rigid pole impact test and the vehicle frame rigid wall impact test. The vehicles' structural geometry were measured and non-structural components were removed from the vehicle. The measured surfaces were then fed into a vehicle mesh generation program called TrueGrid. Simulations of the crashes (and video of real crash tests), such as a frontal impact into a wall can be seen, and other similar work can be seen at Pennsylvania State University[55]. Simulation of this type is extremely useful, as the expense needed to run every type of test would be enormous, but this is offset by the fact that the simulation must be perfect for it to succeed.

Other simulations can include training simulations, for flying or driving. Combat situations for soldiers are also being simulated as a cheaper and easier alternative to combat training. The U.S. Army has funded development of its own 'game' using the Unreal game engine. It is described in Hachman[21] as 'an innovative, realistic computer game providing civilians with an inside perspective and a virtual role in today's premiere land force, the U.S. Army'.

Human motion can be seen in many games, movies and simulations, and there is every reason to believe that animated human motion will be in greater demand in the future, because they are an essential component of storytelling or simulating real life situations.

1.3 Goals and Scope

The long-term goal of this work is to create a system, based on motion graphs, for online motion planning, that produces skilled driving behaviours for a dynamic model of a car in a constrained environment. It is a particular challenge to get the cars to produce highly dynamic behaviours such as skidding-into-turn behaviours when approaching sharp corners, which can help achieve the fastest speeds around a track. The techniques developed could be applied to cars but also for other dynamic vehicle behaviours, in computer games or for intelligent autonomous vehicles in the future. We wish to gain insight into the key issues

in the creation of such behaviours, and to determine where we may need to improve upon and expand this work in the future. The success of our work will be measured on whether or not we can produce good solutions that can exploit dynamic behaviours such as skidding, while attempting to keep the computation time at a reasonable level.

1.4 Contributions

We demonstrate that a well formed move tree or, equivalently, a motion graph, can be used to produce realistic steering behaviours on a variety of tracks. We show that an A* search algorithm is well suited to this task, and offer a number of ways that we could speed up the generation and tree searching for future work in this area. We have produced a number of smooth animations where our agent chooses the fastest way to reach its goal, be it through acceleration/deceleration around tricky obstacles, or a hard skidding turn into a corner after approaching at speed.

1.5 Thesis Outline

In Chapter 2, we present related work in animation and motion planning. Chapter 3 formulates the problem that we wish to solve, and the mechanisms that we apply to solve it. Chapter 4 presents the results of this work along with a discussion of those results. Chapter 5 discusses how this work could be improved upon and extended in the future, including the automation of motion graph generation. Chapter 6 gives the conclusions that we can take from the work, and is followed by the bibliography and appendices.

Chapter 2

Related Work

This chapter reviews work on animation using motion graphs, and related topics.

2.1 Motion Graphs

Most human motions that we make can be described in terms of successive sequences of actions, many of which are similar in nature. Finding our way around a building could be seen as a list of different motions, such as walking forward, opening a door, climbing the stairs, etc. If we could collect each one of the motions needed to navigate and join them together, then we could create a stream of motion to navigate any building.

Motion graphs were formalised by Kovar et al.[29]. A commonly used solution to acquire motions, particularly human motion, is *motion capture*. Motion capture technologies include mechanical armatures, magnetic sensors and multi-camera systems. These systems can accurately reproduce any motion that a real actor can perform. This is a good way to reproduce motion, however it can be difficult to modify unless only minor changes are made, such as by *motion warping* in Witkin and Popovi[59]. If there is no captured motion that is similar enough to a needed new motion then the only choice is to collect more data which can be expensive and time consuming. The motion graph approach attempts to synthesize new streams of motion while retaining the quality of the original data. The motion graph itself is a data structure that shows where transitions can be made between various motion clips. It can be constructed from unstructured motion data by detecting where these transitions can safely be made. We can then select which paths we want to follow through the graph

to produce a new animation, which is called a *graph walk*.

A motion graph is a directed graph where all edges correspond to clips of motion. Each node can be considered to be a choice point connecting these clips.

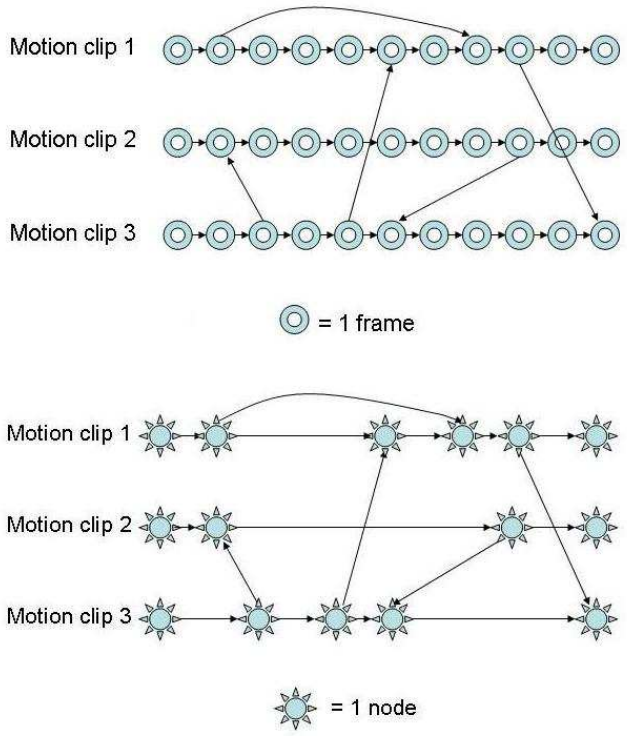

Figure 2.1: Two views of a motion graph.

In the top half of Figure 2.1 we can see 3 motion clips where each circle represents a frame in the clip. We can start with a trivial graph containing $2n$ nodes, one for the start and end of each motion clip. After computing where allowable transitions are, we can add nodes and arcs to the graph representing the points where we can transition from one clip to another. The transitions

can occur within the same starting clip, as shown by the curved arc at the top of Figure 2.1. The resulting graph, that has the remaining non-transition frames abstracted away from its graph edges is shown in the bottom half of Figure 2.1.

A motion graph requires a reasonable amount of connectivity between different nodes to provide compelling animation. It is also important to find transitions that will result in the highest quality motion. Smooth transitions between two motions typically require the use of *motion blending* as seen in Il Park et al.[39]. It is also important to find transitions that will result in the highest quality motion.

Motion graphs have become popular because they can automatically process a large motion database into the useful graph structure. However, the kinematic nature of motion graphs means they may lead to unrealistic behavior because this type of motion model does not take forces into account in dynamic environments.

2.1.1 Finding Transitions

Building a good motion graph requires finding an appropriate measure for when a transition can be reasonably made. This *distance metric* must ensure that both the position and velocity of any relevant DOF are close to each other. For human motion this would include the position of one of the joints, or in our vehicle model it would be the car angle, position and velocities. Velocities must be considered, as illustrated by the example of a man walking forwards and backwards. There will be positions in both walks where they have a similar pose but we would not want to transition between them, as we would have an abrupt and unrealistic change in direction. We also have to reorient the data to local frame coordinates. The data is recorded in global coordinates, but we need to capture the fact that a left turn while facing north is the same motion as a left turn while facing east, for example.

If we look at the methodology of Kovar et al.[29], if two frames A_i and B_j meet the threshold requirements for transition, then they can be blended by

using frames A_i to A_{i+k-1} with frames B_{j-k+1} to B_j, after they have been aligned with one another. In this example, k defines the number of frames over which to blend. However approaches such as these can violate constraints in the original motions, especially for the human motions, where feet may slide as a result of interpolation between two motions.

2.2 More Data Driven Animation Techniques

Motion editing is the process of making changes to motion data. Gleicher [18] showed individual clips of motion can be edited through *retargeting*, which maps motions between characters of different proportions on to a motion clip while retaining the core constraints. Bruderlin and Williams[8] use *signal processing* to reshape the data.

Motion synthesis can be achieved by taking a set of example motions and performing multi-target blends which results in a continuous space of parametized motion. This can be seen in Wiley and Hahn[57] which uses linear interpolation and Rose et al.[47] which uses radial basis functions. These methods are concerned with generating parameterizations of clips whereas our work is about creating a sequence of clips that can be used to control a vehicle with known environmental constraints. Gleicher et al.[17] created a system which built motion graphs with a simpler structure by automatically detecting frequently occurring poses, with transitions appearing at those points, reducing the number of nodes. Kim et al.[22] took rhythmic data and separated it by using beat analysis, then uses k-means to cluster and a Markov model to represent transition probabilities between clusters, meaning new motions could be synthesized that upheld the rhythmic structure. Lee and Lee[36] controlled motion synthesis in real time with dynamic programming, precomputing the optimal transition to take from any node given one of a finite set of goal configurations.

We can also construct statistical models. Pullen and Bregler[42] use kernel-based probability distributions to synthesize new motion based on the properties of the example motion. Bowden[5], Galata et al.[16] and Brand and

Hertzmann[6] all construct abstract states which represent sets of poses. Transition probabilities between states are used to drive motion synthesis. However these techniques can lose important details and end up with an unrealistic average of different motions. The problem with using statistical models in this case is that it can be very difficult to truly map the statistics of human motion. Tanco and Hilton[53] use a Hidden Markov Model (HMM) and k-means to clusters of motion segments. A maximum likelihood algorithm is used to find a sequence of connecting motion segments between a start and end pose. Pullen and Bregler[43] break motion capture animations into smaller pieces and use these to create new "fill-in" motions.

Other graph based work includes Lamouret and van de Panne[31], which discussed characters with a low number of degrees of freedom (DOF). Schodl and Essa[52] apply motion graphs to video images. James and Fatahalian[25] use physical simulation within deformable objects. Perlin[40] and Perlin and Goldberg[41] uses rules and blending to connect procedurally generated pieces of motion into usable streams. Faloutsos et al.[13] use support vector machines to create motion sequences as compositions of actions generated from a set of physically based controllers. These were mainly concerned with creating individual transitions whereas we are planning long sequences of motion.

In the context of games, motion graphs are often referred to as *move trees*. The primary difference is that these are generated manually rather than automatically, which is also seen in Perlin[40] and Rose et al.[46]. This thesis uses manually constructed move trees. Their construction could theoretically be automated, but our focus was their use in efficiently planning constrained motions rather than automatic graph construction.

We need to create smooth transitions between motions at transition points. Perlin[40] interpolates between clips to create a *blend*. Lee and Shin[37] define orientation filters that allow blending operations on rotational data. Rose et al.[47] preserve kinematic constraints and dynamic properties when creating transitions. Arikan and Forsyth[4] and Lee et al.[35] identify transition locations based on a distance metric. Perlin and Goldberg[41] use a linear blending

method. Arikan and Forsyth[4], Pullen and Bregler[42] and Lee et al.[35] use *displacement maps* to achieve smooth transitions. Rose et al.[47] use a torque minimization algorithm. Smooth animations are created in this work by only using frames that are extremely close for comparison purposes, and do not require any complex blending procedures.

2.3 Autonomous Behaviours and Planning Algorithms

The behaviour of flocks, herds and schools are discussed in Reynolds[45], as the BOIDS model. This allows the programming of large animal groups to act in support of a group goal, while also demonstrating realistic individual variability. Scripting the path of each individual member of the group is time consuming and in many cases downright tedious. The work of Reynolds[45] applies three local behavioural rules to each member of the group: collision avoidance, velocity matching and flock centering (staying close to nearby flockmates). Some of the most interesting motion occurs when the flock is forced to interact with and avoid objects in its environment.

As an extension of BOIDS, Go et al.[19] tackle the problem of animating the behaviour of vehicles within interactive simulations. However individual vehicles generally do not have the same mentalities as flocks, and the dynamics mean that they have complicated control models and high velocities. They combine online search with *steering behaviours*, seen in [44], to guide the search function. For interactive games, speed is generally preferred to high accuracy, and so the motion must be generated rapidly. They focus on extending the steering behaviour control model and combine it with online path planning techniques, in an attempt to create more goal-driven synthesized vehicle animations. It is claimed to be suitable for not only space, water and land based vehicles, but also birds, bicycles or any other simulated entity with significant dynamics. Behaviours are designed in a manner that means they can apply it to the motions

of multiple entities. The key components of this work are the interface for steering behaviours, a visually plausible control model, the pre-integration of dynamic trajectories to enable real time performance, and a search algorithm designed to operate with limited time and information.

Steering behaviours are denoted by the type of motion it produces, such as seeking or wandering. A major advantage is that no matter what input is given, the output is always a vector of desired velocity. These behaviors can be mathematically combined and mean that there is a weak dependence between the control model and the behavior. However combining behaviours can result in nonsense behaviour. They are also not well suited to nonholonomic (wheeled) vehicles with inherent drift in their system dynamics, as computed paths may not be representative of the actual path taken.

Most planning methods currently assume full knowledge of the environment. A popular path planning method was developed by LaValle[33][34], called *Rapidly-exploring Random Trees* (RRTs), that can be used for many types of dynamic system. This technique is successfully applied by Yamane et al.[60], who synthesized animations of human manipulation tasks, such as placing a box on a shelf, and also by Bruce and Veloso[7] who implement a real-time robot replanning system.

Reynolds[44] uses a different approach that uses local information. Steering behaviours use a time-local approximation of steering forces that depend on desired behaviors. Steering forces are efficient to compute, but behaviours can get stuck in local minima or behaviour loops because they only consider local information.

Witkin and Kass[58] globally optimize a set of control inputs subject to specified constraints and defined optimization criteria. van de Panne et al.[56] compute an optimal control policy for small environments and for objects with simple dynamics using state space controllers. Grzeszczuk and Terzopoulos[20] use simulated dynamic entities control spaces to create local behaviour controllers.

Frazzoli et al.[15] and Feron et al.[14] use *trim trajectories* to control a he-

licopter. This is a similar problem to our work in terms of vehicle control, but has further complexity from being in a dynamic environment, so we will discuss this work in more depth. Trim trajectories are predefined manoeuvres that are invariant with respect to some state variables. For example, they are useful because they allow a high level planner to efficiently reason about local state reachability. The operation of an autonomous vehicle in an unknown, dynamic and hostile environment is an extremely complex problem when the vehicle is required to use its full maneuvering capabilities. To deal with such a system they decompose activities into a hierarchy and introduce control and decision layers. Some systems are continuous while decision making systems are likely to be discrete, which means this system is referred to as a *hybrid system*. Such a controller is responsible for both the generation and execution of a flight plan. To reduce complexity they base planning upon a quantization of the system dynamics, restricting the feasible system trajectories to a family of time-parametrized curves, which as a whole constitute a *maneuver library*. This reduces the search space for solution, now not a continuous space. Their system tries to capture the relevant characteristics of the vehicle dynamics and allow the creation of complex behaviours for the interconnection of primitives, to produce good approximations of optimal solutions. This piecewise assembly of trajectories is similar in essence to the system we propose. However they track the actual trajectories that occur given the input trajectories primitives, to ensure that they are close to each other based on the outside factors such as noise and unmodelled dynamics, whereas we directly assemble and play our stored animations.

Our work is directly related to the field of vehicle control, for both real and simulated agents. Latombe[32] describes motion planning methods, that use a trajectory-planning stage and trajectory-tracking scheme. The scheme is meant to avoid obstacles and enforce the non-holonomic constraints of the vehicle. Bullo and Murray[9] evaluate various linear and non-linear designs for trajectory tracking. Divelbiss and Wen[12] looked at the problem of parallel parking non-holonomic vehicles that had between zero and two trailers. Trajec-

tories are planned offline and a linear quadratic regulator tracks the trajectory. Saffiotti[50] focuses on the control of autonomous robots and the presence of uncertainty in real-world environments. Rosenblatt[48] proposes a distributed architecture which combines information from different sources each voting on the best way to complete there personal goals. Santana[51] aims to ensure safe local navigation for a robot. Kelly and Nagy[27] determine a control input in real-time to bring a vehicle from a start point to a goal. Most of these are dependent upon knowing a world model.

Other work is based upon *policy search* methods. For the truck backing-up problem, Koza[30] assumes knowledge of the global state of the truck and used genetic programming. Hougen et al.[23] learn a controller on a real tractor-trailer robot, and give the self organising neural network controller access to the angle of the last trailer with respect to the goal state. Hougen et al.[24] continue from this to look at the more difficult problem of backing up a double trailer truck in simulation mode.

Temporal Difference (TD) learning can be seen in Coulom[11], which created a policy in the now defunct Robot Automobile Racing Simulation (RARS), which was a race between programmers to produce the best car controller. Actions are learned based upon the cars position and velocity in relation the local track features, and also the curvilinear distance from the start line. A newer version of the RARS championship can be seen in The Open Racing Car Simulator (TORCS)[54].

In Alton[2] and Alton and van de Panne[3], the motion of the vehicle is controlled by reinforcement learning and policy search. Two non-holonomic vehicle types are tested, a car, and truck with a trailer, with different control problems involving going forwards and backwards. They do not use a physics system to model car sliding as in our work. A policy search framework is used to solve the motion control problem. The control policies use a nearest-neighbor function approximator (NNFA) to determine which policy will be chosen at any particular time point. A particular challenge is the design of the reward function, where it is hard to work out what weights to give to various factors such as

energy efficiency, time efficiency, closeness to goal. Other notions of optimality, such as the smoothness of trajectories, are rewarded as those trajectories will tend to be the ones that make maximum progress.

Chapter 3

Vehicle Steering

In this chapter, we develop a technique for efficiently planning the dynamic motion of a physics-based car simulation around highly constrained dynamic environments.

3.1 Vehicle Model

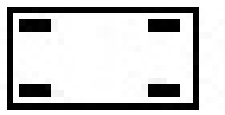

Figure 3.1: The car.

The main problem we are trying to solve is that of driving a car around a track at speed using clips from a motion graph. This agent has a 6 dimensional state space:

$$\mathrm{S} = [x,\ y,\ \theta,\ \dot{\mathrm{x}},\ \dot{\mathrm{y}},\ \dot{\theta}]$$

where x, y are coordinates and θ is the angle the agent is facing, and $\dot{\mathrm{x}}$, $\dot{\mathrm{y}}$ and $\dot{\theta}$ are the velocities of these. The agent has a steering angle as input, represented graphically by the direction the tires are pointing, which we will call s. The agent itself has a physics based model for movement, which is affected by the weight of

the agent, the size of the agent, etc. This takes into account a large number of variables in order to work out the position of the agent for the next frame, and constants such as *drag*, *resistance* and *friction*. The agent itself is described by several constant parameters including front/rear axle distance, and also mass (in kg) and inertia (in kg per meter). In order to compute the position of the agent for the next frame, the velocity in world coordinates is transformed to the agent reference frame. Next the lateral force on the wheels is calculated. Forces and torque are applied to compute the current acceleration. Finally, the accelerations are integrated to obtain an updated velocity and position. The full implementation of the sliding physics is given in Appendix A.

3.2 Pseudocode Planning Algorithm

Our overall algorithm, which runs for each frame, is shown below. We discuss it in detail in the remainder of the chapter.

```
for (each frame) {
    if (current frame is a transition point) {
        initialize route tree and add unexpanded start node to list
        while (there are unexpanded nodes and node limit has not been reached) {
            select next node
            if (path between current node and next node does not collide with wall) {
                expand node and add new unexpanded nodes to list
            }
        }
        create a list of all possible paths from end points
        find the best path by heuristic
        if (best path is not current path) {
            set current path to best path
}   }   }
```

3.3 Data Generation

In order to develop a motion graph (or move tree), we first need to collect relevant motion data. A 'Logitech Dual Action' (i.e. PS2 style) control pad is used to interactively control turns of different curvature and speeds. The steering angle is mapped to the left analog control stick, and acceleration and braking are mapped to buttons[1]. These turns are all made using the physics model but there is no interaction with scene objects, i.e. no bouncing off walls, apart from direct collisions with walls which reduce velocities to zero. Many clips of animation were recorded of a user driving around tracks or with no track, to experiment with making soft and hard turns, and the best of these were selected for use to try and represent as many of the most useful types of movement and cornering as possible. Only clips that did not contain collisions were selected. Clips were not doubled up using symmetry, each clip in use is of an actual left or right turn.

3.4 Finding Similar Frames

In order to construct a motion graph from the data, we need to identify potential transition points, either manually or automatically. The automatic identification of transition points requires finding similar frames. Before we can compare two frames *f1* and *f2*, we must rotate them so they are in the same coordinate system. We compare the velocities in the local coordinate frame of the car rather than in world coordinates, because the global position and global orientation are irrelevant.

Next, we need to find a metric for frame comparison. There are various ways we could go about this, but the final goal is come up with a single value that measures the similarity of the state or 'situation' of the car (Figure 3.2).

We use u^f to describe velocity of frame f in x, v^f to describe velocity of

[1] the right analog stick, or the up down axis of the left analog stick would have been viable alternatives.

frame f in y, and ω^f to describe angular velocity of frame f, which gives the following function:

$$d = |u^1 - u^2| + |v^1 - v^2| + |\omega^1 - \omega^2|$$

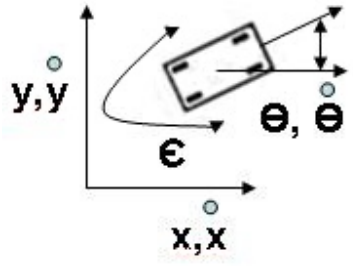

Figure 3.2: The car velocity comparison.

We then use a threshold of $d < \epsilon$ to detect frames that we can mark as being sufficiently similar to support a motion transition.

Using similar frames we can now begin construction of a *Route Tree* that gives possible paths for the agent to take. A route tree is a map of routes that the car can choose from based on its current location. This can be expanded to various search depths to give a richer selection of paths. A route tree represents a specific instantiation of the move tree or motion graph that enumerates all the paths forward from the current state.

3.5 Creating and Expanding Routes

This section explains how the route tree in this work is created and expanded. A route tree is created when there are multiple choices available for the next motion clip at the current frame. The next node to be expanded is chosen via an algorithm, which after some experimentation was chosen to be *A* search*. We use a finite search horizon expressed as a fixed planning time duration. Also, a

memory bounded value is given to limit the total number of nodes searched for larger expansions.

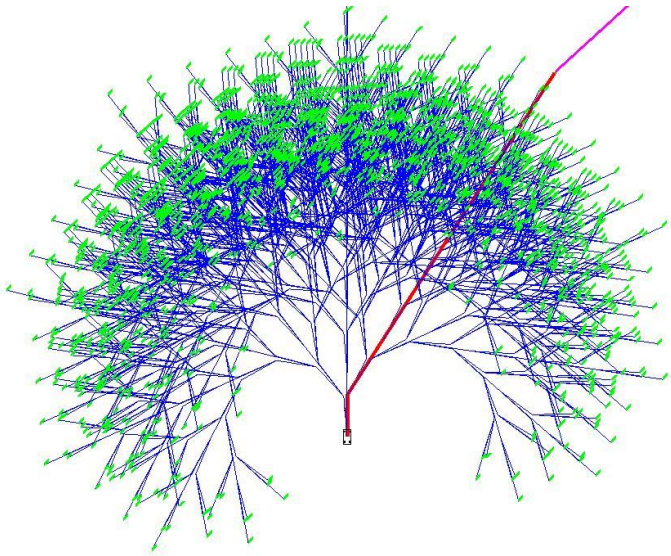

Figure 3.3: A route tree enumerates possible future paths from a particular state. In this example, there are three choices at each decision point. The magenta line at the top right is a line from the selected best node to the current goal. The best path is shown in red.

3.5.1 A* Search

A* search is a widely used search algorithm, which is based upon *best-first search*, as seen in Russell and Norvig[49]. Best first search incorporates strategies that choose to expand the current 'best' node, where the metric used to define 'best' is given by a *heuristic*. A common example of a cost-to-go heuristic for motion planning is the *straight line distance*, where the next node that is expanded is chosen to be the one that is currently closest to the goal. Just using cost-to-go can be inefficient, given that it ignores the ability to cull solution paths based on the cost-to-date.

A better approach combines the straight line distance *h(n)* with the cost to reach the node *g(n)*.

$$f(n) = g(n) + h(n)$$

This sum gives us an estimate of the cheapest solution that goes through node n. The straight-line distance-to-goal never underestimates the cost to reach the goal and therefore defines an *admissible heuristic*. The A* search therefore finds the optimal solution if run to completion. In order to optionally further constrain the search we can bound each search to a maximum number of nodes, meaning that for larger searches we may not achieve an optimal solution, but smaller searches can be performed in real time. The number of nodes expanded for each search is discussed further in the results section.

In order to implement A* search efficiently, a data structure must be chosen to place the items in a priority queue in a much faster manner. A *heap* is ideally suited to this task, in this case a *min-heap*. A min-heap is one which is partially ordered with the lowest value at the top. This allows us to store values in an order and release the highest priority (lowest value) node when needed.

3.6 Route Selection and Clip Alignment

When the agent reaches a decision point in the route tree, it finds which one of the corresponding branches is on the best path to the goal, as determined by the A* search and sets its new start frame accordingly. This frame must be translated and rotated into the agent's current coordinate frame. Given a current agent coordinate frame of A, the agent coordinate frame at the beginning of the current clip as $A\acute{}$, the new playback clip coordinate frame $B\acute{}$, and the current frame in that playback clip of B, we can compute the agents position as driven by the transformed motion clip. This is shown in Figure 3.4.

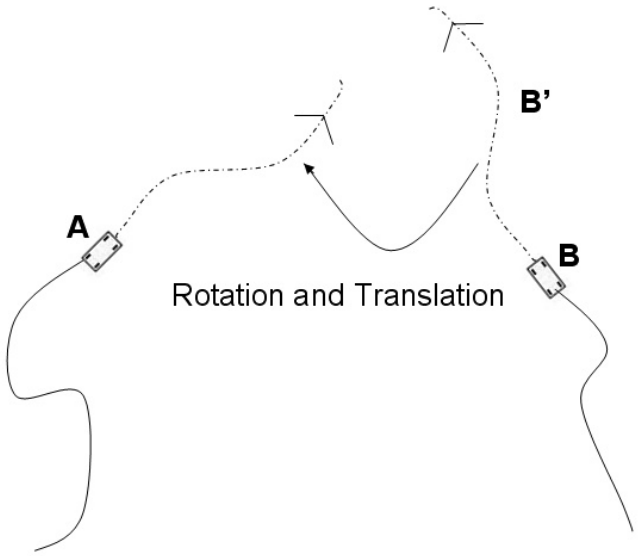

Figure 3.4: Motion clip B' is rotated into place in order to provide the next frames of motion for car A.

3.7 Collision Detection

The agent needs knowledge of its environment in order to plan its motion. This is done by intersecting the potential paths, as defined by the route tree, with the environment. Collision detection with 2D track walls is made by comparing the agents sides to every track or car obstacle line at multiple points within each motion clip. For a straight clip the lines between the start frame and the final frame are the only ones tested, but for a sharp turn up to three separate in-between frames are tested, as shown in Figure 3.5.

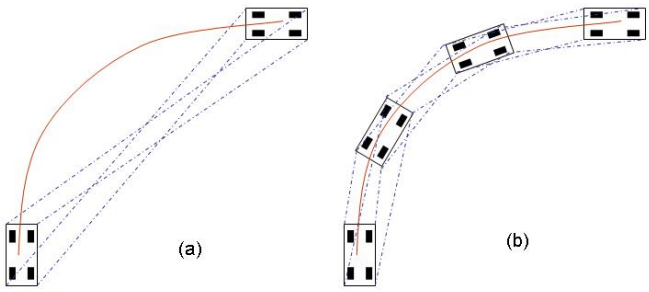

Figure 3.5: Collision detection tests, which is much more accurate given more frames. (a) shows detection between the two end points of a curve, which is less accurate than (b) given 2 extra frames.

Chapter 4

Results

This chapter describes the results of our work.

First a move tree with a 5-way branching factor is constructed that guarantees continuity of position. This move tree does not necessarily have smooth transitions in terms of forward velocity. Second, we develop a large move tree with smoother and more realistic transitions between these. Finally we discuss the implications of these results.

We use a goal location of *x=0,y=100000*, as measured in metres, where the xy plane represents the driving surface. It would also be possible to implement a dynamic goal to allow the agent to drive around circular style tracks. We use the cost metric developed in Chapter 3.

4.1 Move Tree

Two small hand constructed move trees are created from data recorded using the joypad. Figure 4.1 shows schematic views of the two trees. Roughly speaking, the trees consist of varying degrees of left and right turns. For simplicity of construction, velocities are not enforced to be perfectly continuous between successive animation clips. Lateral components of the velocities are very close by design and because of the car physics. The agent will thus not suddenly lurch sideways when entering a new animation. However, it may speed up or slow down quicker than true physics would allow.

The larger move tree is the same as the first, except it has two more branches allowing for slightly sharper turns. Each one of the clips could be used after the end of any previous clip, and they were all 25 frames long (1 second of the

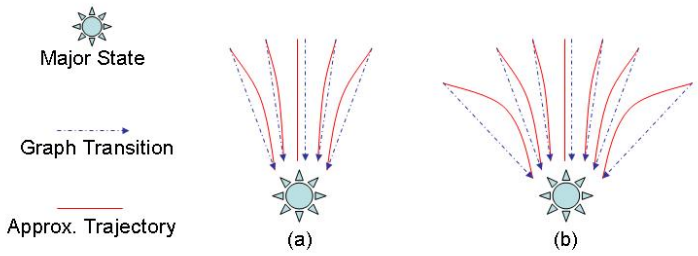

Figure 4.1: This is a figure of the small move trees, with a) 5 and b) 7 branches, showing the possible transitions and approximate trajectories from the only possible state. Each clip is 25 frames in length.

final movie). The 5 and 7 branch move trees were tested against each other to see how they performed on a variety of tracks. We now describe the results on a variety of environments.

4.1.1 Long Straight Track with Other Cars

The straight track shown in Figure 4.2 is designed to test the ability to avoid other moving cars on a long straight road, using various search depths. The test is conducted with the 5 branch tree, which does not offer many choices for movement, but is useful on this style of track where most movements will be straight. The performance for various levels of 'look ahead' or depth were evaluated.

The results[1] of these tests shown in Table 4.1 show the benefit of longer time-horizon planning. Looking ahead one or two clips results in both cars avoiding obstacles at the last possible moment and both collide quickly. Looking three or four clips ahead fared better but both still crashed, although they showed a greater degree of anticipation of future car movements. Looking ahead five clips (124 frames) allowed the car to not crash during two minutes of driving.

[1] See www.cs.ubc.ca/∼andyadam for videos.

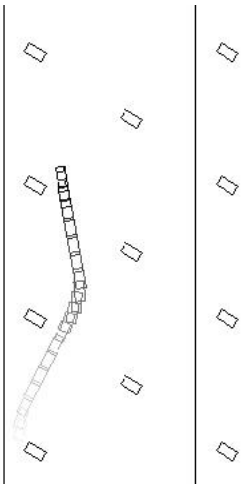

Figure 4.2: Long straight track. The equally-spaced tilted rectangles on track represent cars moving at constant speed that act as dynamic obstacles.

Whenever the agent does decide to turn, it makes only slight movements that allow it to get through the gaps among the cars at maximum speed. This does not prove that this planning depth is sufficient for all possible tracks and obstacle placements.

4.1.2 Cornered Track

The cornered track, as shown in Figure 4.3, is designed to test the ability to travel around a more interesting track with a variety of corner angles.

This produced a number of interesting behaviours. Firstly, the test that plans only 2 clips moves forward quickly, but only reacts when it reaches a wall. This is caused by only having a few different options for movement, with no fast yet slight turn available.

The animation using a 4 clip planning horizon performs better. However the search may result in a slower time to the goal despite searching further ahead as a result of our choice to bound the node search to the first 15000 nodes encountered. This phenomenon repeats itself with the two deeper searches using

depth(*frames*/*nodes*)	*times* (*secs*)
24/1	10 sec crash
49/2	17 sec crash
74/3	57 sec crash
99/4	110 sec crash
124/5	no crash after 2 mins

Table 4.1: The results for the small 5 branch move tree on the long track with other cars.

depth(*frames*/*nodes*)	*branch*	*node limit*	*time to finish line* (*secs*)
149/6	5	15000	33
299/12	5	45000	39
49/2	7	15000	35.5
99/4	7	15000	31.5
149/6	7	15000	35

Table 4.2: The results for the 5 and 7 branch move trees on the cornered track.

the 5 branch tree, where the search of 6 clips and 15000 nodes performs much better than the search of 12 clips of 45000 nodes.

4.1.3 Cornered Track with Other Cars

The same track as above is used but with a collection of other cars that serve as moving obstacles. The majority of tests are conducted here as this problem is the most interesting and difficult.

A variety of tests were run with the 5 branch tree, but all of them end in collision, most of them in the same tricky area of track. The 7 branch tree with a planning horizon of 2 clips cannot avoid the cars in time and make it around the corner, and collides with a wall. As the planning depth increases, the agent manages to improve its performance, i.e. to get to the goal faster.

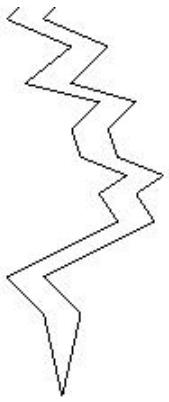

Figure 4.3: Cornered track. The car begins at the bottom and its goal is to drive towards the top.

Two tests are run at a depth of 6 clips, one with each planning step bounded to 15000 nodes and another bounded to 45000 nodes. Surprisingly the 45000 node version was slower. The only reason for this can be that a more optimal (faster) decision earlier in the track results in it having to follow a slightly slower path later, perhaps getting held up in 'traffic'.

4.2 Large Move Tree

A more expansive move tree is constructed in order to produce animations with more realistic transitions while avoiding the need for blending. This is shown in Figure 4.4.

The tree was given three main states from which a number of different turns could be performed - a *Stop* state, a *Medium* state and a *Fast* state. In order to make it easier to maintain a constant speed and make decisions about going fast or slow at any point, two in-between states were added between each main state. This allows the agent to speed up and slow down without having to go all the way up to commit to moving to Fast and then back to Medium, needing a

depth(*frames*/*nodes*)	*branch*	*node limit*	*times* (*secs*)
149/6	5	45000	crash
199/8	5	45000	crash
299/12	5	15000	crash
299/12	5	45000	crash
49/2	7	15000	crash
99/4	7	15000	41
149/6	7	15000	38
299/12	7	15000	33.5
299/12	7	45000	35

Table 4.3: The results for the small move trees on the cornered track with other cars.

large section of straight track. The car can speed up to *mf1* and then slow down to Medium again, gaining speed but not being stuck in one long acceleration or deceleration animation.

4.2.1 Long Straight Track with Other Cars

depth(*frames*)	*node limit*	*times*
299	45000	still going 2 min

Table 4.4: Results for the large move tree on the long track with other cars.

A test was run on the long track with other cars moving as obstacles. The agent tends to hug the wall, making slight speed adjustments, and very slight turns, to keep it aligned with the gaps between the cars. It also shows that the agent is always attempting to go straight and not randomly choosing to turn for an unnecessary purpose.

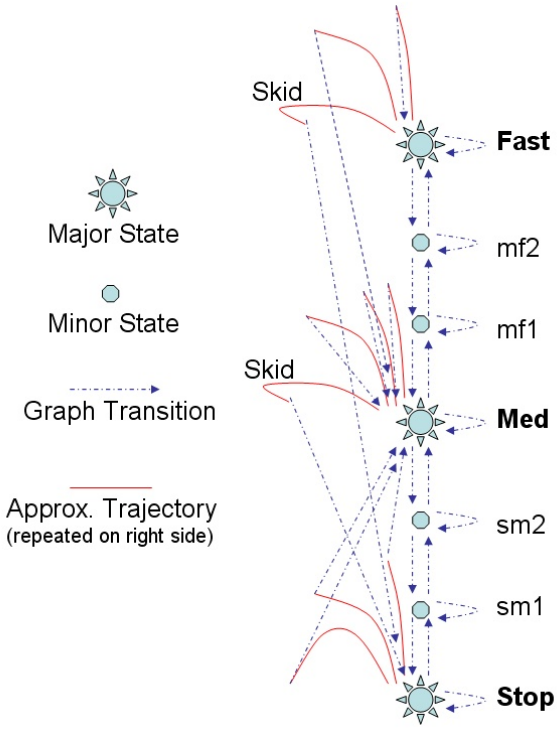

Figure 4.4: A large move tree showing a total of seven speeds.

4.2.2 Cornered Track

depth(frames)	*node limit*	*time to finish line (secs)*
49	45000	44

Table 4.5: Results for the large move tree on the cornered track.

One test was run on the cornered track to ensure it produced a good animation. The animation, though being expectedly slower than those with the smaller tree, which was ignoring physics, was much smoother than those animations and also intelligently anticipated upcoming corners.

4.2.3 Cornered Track with Other Cars

depth(*frames*)	*node limit*	*time to finish line* (*secs*)
400	15000	45.5
400	45000	50
400	250000	46
700	250000	45.5

Table 4.6: Results for the large move tree on the cornered track with other cars.

Tests were run on the cornered track with other cars. With a 15000 node planning horizon a reasonable time and smooth animation is produced, but at 45000 we see a slower performing result. To show that this is not the result of a potential coding error, an additional test was run with a 250000 node planning horizon. This produced a similar time to the first test. It seemed that the agent had to sit behind other cars occasionally, probably because it had deduced that it could not speed up and slow down in time to get around the car and also the following corner. All of the animations were realistic and smooth.

4.2.4 Sharp Cornered Track

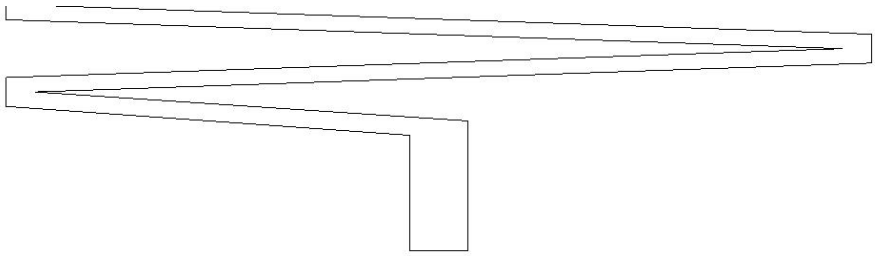

Figure 4.5: Sharp cornered track.

Despite producing smooth animations on the track above, we have not provided the system any need to use the hard slides provided within the move tree.

In order to do this a track (Figure 4.5) was created with long straight sections and sharp hard turns, which should ensure that the fastest (if not only) way to get around the corner would be to skid into it, as shown in Figure 4.7)

$depth(frames)$	*node limit*
200	15000
400	15000

Table 4.7: The results for the large move tree on the sharp cornered track.

The results proved successful. In all cases the agent used the skid to get around the corner. For the depth 200 version the particular instantiation of the skid causes it to leave the corner slowly. This could perhaps be rectified by having hard skids that leave the agent at different angles at the conclusion. However all of the skids allow the agent to reach high speed going into the corner and escape effectively.

4.2.5 Final Track

The final track (Figure 4.6) was produced to show all of the above elements. It contains normal turns, a sharp turn, and many obstacle cars driving in the large center area. Figure 4.7 shows the skid. For a better representation of the results see the online videos.

depth	*node limit*	*time to finish line* $(secs)$
200	5000	48.5
400	250000	51

Table 4.8: The results for the large move tree on the final track.

A small lookahead was used with a low node limit, and the animation was created in around 3-4 minutes, which results in around 5 seconds of computation

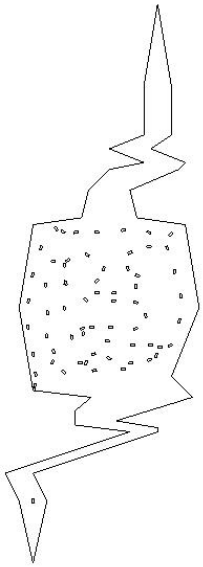

Figure 4.6: Final track.

per frame. Looking further ahead and taking much longer to process did not produce significantly better results in this case, but the number and closeness of opposition cars was increased. The low node limit could not find a path through, but the larger search produced a smooth animation and path through the track, making it look easy. These final movies show the culmination of all of the above elements, to show that they all work as one.

4.3 Discussion

We have learned various things of interest from these tests. First, interesting local minima exist. Simply increasing the number of nodes we search does not automatically mean that we are going to achieve a better performing result. This is because a slightly faster earlier route may yet slow the agent down at a later point. Despite this we can still see from other examples that longer planning horizons are more likely to produce a good animation and in all cases

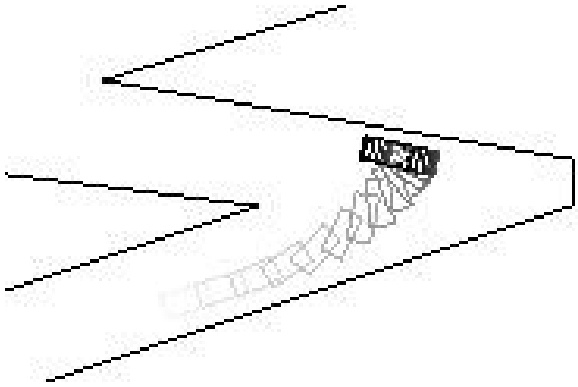

Figure 4.7: A resulting skid on the final track.

reduce the possibility of a collision. There is also clear evidence that searching too far ahead while not searching enough nodes gives us poor results.

The way we structure our motion tree greatly affects the quality of animation we can produce, and can be tailored to individual tracks. The 5 branch graph worked well on the straight track at avoiding other agents, however it performs poorly on the curved track that forced it to negotiate tight corners at exact angles. Adding just two more turns to the tree allows it to get around the track, and planning further ahead allows it to negotiate the track in a much faster time.

While the 5 and 7 branch trees prove useful for testing aspects of move trees, the animations they produced are not smooth and do not conform to the physics constraints. Producing a move tree that ensures transitions are very close to each other in terms of velocities allows us to produce smoother animations and come closer to meeting the goals of this work, namely to produce animations conforming to a physical system. However the computation time for these smaller trees is real time in some cases, as opposed to hours for the largest tests conducted. Most mid range tests, which produce almost as good or occasionally better results than the really large tests, take around 10-20 minutes when using the large move tree, depending upon the number of track

obstacles, and whether the whole tree could be expanded and each step. Tight corridors resulted in much shorter computation times as large sections of the motion become highly constrained. These computation times could be improved significantly by the implementation of some of the suggestions made in the next chapter.

When comparing the results produced to those produced by a human using a joypad, with practice it was possible for a human to get around all tracks, but it is very difficult to reproduce the dynamic skidding behaviours for tight corners, with virtually every attempt at high speed skids resulting in crashes into the wall, or ending up short of the corner. Similar problems occur when trying to avoid car obstacles, resulting in crashes or slow routes taken after twitch based movements to avoid cars at the last seconds. In particular they could not reproduce the long sections of straight driving seen when the agent is driving on the straight track against other cars. The agent could be much more exact about its movements, whereas a person may get lucky on a tricky track but will usually require multiple attempts to even get through tricky areas at a decent speed. Overall human control produced motions that were less smooth and were often unsuccessful, although perhaps they could come close with a refined control system and sufficient practice. Two instances can be given where human control resulted in better animations. Firstly, this occurs when very simple tracks were given, and much practice made at cornering exactly to limits, and secondly when the agent had a lot of scope for a path to take and ended up in a suboptimal area later in the track, it is possible that a human could realise this mistake and choose a better route based upon their past experience. In the first case, the planning result depends on how rich or how suited the move tree we give is to that particular track and set of obstacles.

The overall conclusion is that the agent based animations are much more likely to succeed and produce nice smooth driving animations. By using the methods in this thesis we can get the car to produce highly dynamic manoeuvres such as skidding behaviours in order to achieve the goal of navigating a track in minimal time.

Chapter 5

Discussion

In this chapter we discuss various limitations of our work and how the methods and results might be improved.

5.1 Graph Transition Selection

The move trees used to obtain our results were hand constructed. Developing good move trees is a time consuming and painstaking process. Motion graphs can be used to automate this process. However the use of motion graphs is potentially problematic for our problem in several ways. If we identify suitable transitions by comparing every frame with every other frame, we can end up with an extremely dense motion graph. This can happen whenever two similar motions occur in a clip. Figure 5.1 shows where connections are made for two frames n and m, but connections are also made at *n+1* and *m+1*, *n+2* and *m+2*, etc. Many of these connections can be removed without any significant loss of functionality. One choice is to not compare the frame with the next k_c frames that occur after it. This figure can be adjusted by the user for each set of motion graphs to give optimal results.

This primary problem can also occur when we have sections of very similar data. For example, if there are portions of the motion data that have the agent driving straight at a constant speed, we can end up with an huge increase in the number of connections between all the similar frames of the agent going straight (Figure 5.2). This can be tackled by not comparing against the next k_s frames that occur after the last similar frame that we have found. Once again, this value can be adjusted by the user to give the best results for the data they are

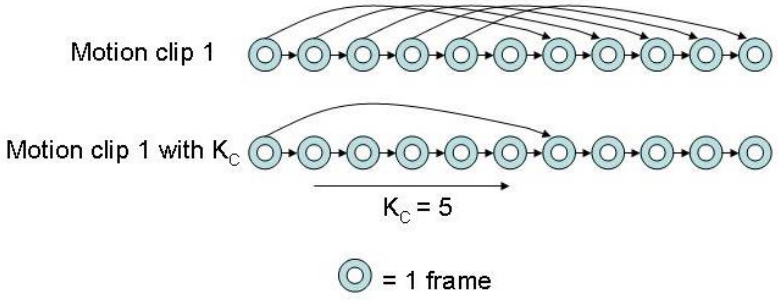

Figure 5.1: A dense, degenerate motion graph can arise from the occurrence of highly similar motions.

using.

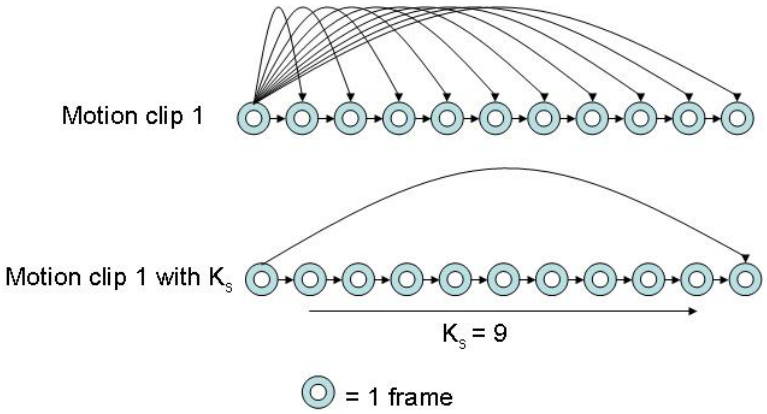

Figure 5.2: Another type of degeneracy in motion graphs.

5.2 Graph Pruning

To avoid creating a motion graph that is overly dense, we need to able to strip away motions that provide us with minimal or repeated animations. A good

example of this would be where we end up many repeated connections where the agent is simply driving straight. It may be possible to match all of the closest frames to each other, but apply some sort of algorithm that removes any animation clips that add little to the overall possibilities of movement. Edges can also be eliminated when they are not part of the strongly connected components of the graph.

5.3 Improving A*

The heuristic for expanding nodes using A* could be improved by weighting in factors such as the agents current speed and angle. It would be possible to work out an approximate figure for the time the agent needs to turn, face the goal, and reach maximum velocity from any orientation/velocity combination. This may improve the order that nodes are expanded, and reduce the number of nodes that we need to expand in order to find an optimal or close to optimal solution.

5.4 Simplistic Goal Selection

At the moment we employ a goal which simply tells the agent to maximize its y value at any point. This does not allow us to traverse looped tracks. It would be easy to implement a system that updates the goal based on the current corner the agent is at, but a more fluid system would allow the agent to set its own goal at any point.

5.5 Track Generation

All of the tracks used in this project were hand constructed which was time consuming. An automation of this process would be preferable as long as it could be done in a way that gave the user a reasonable level of control over

what is produced, in terms of length, number of corners, sharpness of corners etc.

5.6 Collision Detection

The collision detection algorithm could be significantly improved. The trajectory of the car is represented using piecewise line segments, and every such segment is tested against every single line segment representing the environment. This can be made significantly more efficient.

5.7 Additional Mobility

An interesting further application of this work would be to provide the vehicle with the ability to reverse. This would allow the agent to move through extremely tight sections of track and should greatly reduce the chance of the agent becoming trapped.

Chapter 6

Conclusions

We have demonstrated that a well formed move tree based on motion clips derived from a physics-based system can be used to perform motion planning that exhibits considerable anticipation. The move tree is expanded and searched using A* search to produce realistic steering behaviours. We have provided results on various tracks using different search trees, node limits and planning horizons. We have produced a number of smooth animations where our agent demonstrates skilled dynamic behavior in reaching its goal, be it through acceleration/deceleration around tricky obstacles, or a hard skidding turn into a corner after approaching at speed.

The work can be repeated on general tracks and obstacles other than those tested upon here, and with any simple or complex, hand-constructed or automatically derived move tree. It may also be useful for other physical systems that could base their movements upon motion graphs. It is not as suitable for environments with rugged terrain, where movement may be affected by the surface of movement, but in a smooth open environment, including space/air/water, the ideas here should be just as applicable, and could be extended into three dimensions. As long as the move tree given to the agent is suited to the track, or rich in content, then on average the animations produced will be more skilled or better performing than those produced by human control.

The most notable success of this work is the production of highly dynamic planned motions such as skidding behaviours, which to our knowledge have not been demonstrated in equivalent path finding procedures.

Planning and control of highly dynamic motions very much remains an open

area of research in animation and robotics.

Appendix A

Physics Implementation

This is the implementation of the sliding physics[1].

```
// global constants
// These constants are arbitrary values, not realistic ones.
DRAG = 4.0 // factor for air resistance (drag)
RESISTANCE = 20.0 // factor for rolling resistance
CAR = -4.20 // cornering stiffness
CAF = -4.0 // cornering stiffness
MAXGRIP = 3.0 // maximum (normalised) friction force, =diameter of friction circle
carb = 1; // distance from CG to front axle in metres
carc = 1; // idem to rear axle in metres
carwb = carb + carc; // car wheelbase in metres
carm = 1500; // car mass in kg
cari = 1500; // car inertia in kg per metre

// global variables
gX, gY, gA; // car position and angle
gvX, gvY, gvA; // velocities of car
cars; // steering angle
carthr, carbra; // car throttle and brakes (throttle used but brakes not)
dt; // delta t, time interval
```

[1] Originally developed by Marco Monster of Monstrous Software - unfortunately since it was found, all references to it have disappeared and the website has shut down.

```
// local variables
fvX, vY; // velocity
fawcX, awcY; // acceleration in world coordinates
ffX, fY; // force
frsX, rsY; // resistance
faX, aY; // acceleration
fftX, ftY; // traction
fflatfX, flatfY, flatrX, flatrY; // flat
fwheelbase; // wheelbase
ra; // rotation angle
ss; // sideslip
saf, sar; // slip angle front and rear
t; // torque
aa; // angular acceleration
sn, cs; // sine and cosine, of car angle
y; // yawspeed
w; // weight

sn = sin( gA )
cs = cos( gA )
// x is to the front of the car, y is to the right, z is down
// transform velocity in world reference frame to velocity in car reference frame
vX = cs * gvY + sn * gvX
vY = -sn * gvY + cs * gvX

// Lateral force on wheels
    // Resulting velocity of the wheels as result of the yaw rate of the car body
    // v = yawrate * r where r is distance of wheel to CG (approx. half wheel base)
    // yawrate (ang.velocity) must be in rad/s
    yawspeed = wheelbase * 0.5 * gvA
```

```
    if( vX = 0 ) ra = 0
    else ra = atan2( ys, vX )
    // Calculate the side slip angle of the car (a.k.a. beta)
    if( vX = 0 ) ss = 0
    else ss = atan2( vY, vX )
    // Calculate slip angles for front and rear wheels (a.k.a. alpha)
    saf = ss + ra - cars
    sar = ss - ra
    // weight per axle = half car mass times 1G (=9.8m/s2̂)
    w = carm * 9.8 * 0.5
    // lateral force on front wheels = (Ca * slip angle) capped to friction circle * load
    flatfX = 0
    flatfY = CAF * saf
    flatfY = lower value of ( MAXGRIP, flatfY )
    flatfY = higher value of ( -MAXGRIP, flatfY )
    flatfY = flatfY* w
    // lateral force on rear wheels
    flatrX = 0
    flatrY = CAR * sar
    flatrY = lower value of ( MAXGRIP and flatrY )
    flatrY = higher value of ( -MAXGRIP and flatrY )
    flatrY = flatrY * w
    // longtitudinal force on rear wheels - very simple traction model
    ftX = 100 * ( cartho - carbra * ([vX]/[vX]) ) )
    ftY = 0

// Forces and torque on body
    // drag and rolling resistance
    rsX = -( RESISTANCE*vX + DRAG*vX*[vX] )
    rsY = -( RESISTANCE*vY + DRAG*vY*[vY] )
```

```
    // sum forces
    fX = ftX + sin( cars ) * flatfX + flatrX + rsX
    fY = ftY + cos( cars ) * flatfY + flatrY + rsY
    // torque on body from lateral forces
    t = carb * flatfY - carc * flatrY

// Acceleration
    // Newton F = m.a, therefore a = F/m
    aX = fX / carm
    aY = fY / carm
    aa = t / cari

// Velocity and position
    // transform acceleration from car reference frame to world reference frame
    awcX = cs * aY + sn * aX
    awcY = -sn * aY + cs * aX
    // velocity is integrated acceleration
    gvX = gvX + ( dt * awcX )
    gvY = gvY + ( dt * awcY )
    // position is integrated velocity
    gX = gX + ( dt * gvX )
    gY = gY + ( dt * gvY )

// Angular velocity and heading
    // integrate angular acceleration to get angular velocity
    gvA = gvA + ( dt * aa )
    // integrate angular velocity to get angular orientation
    gA = gA + ( dt * gvA )
```

Bibliography

[1] Simbin Development Team AB. Gtr2 game. http://gtr-game.10tacle.com/index.php?id=246&L=1.

[2] Ken Alton. Shaping and policy search for nearest-neighbour control policies with applications to vehicle steering. Master's thesis, UBC Computer Science 2004, 2004.

[3] Ken Alton and Michiel van de Panne. Learning to steer on winding tracks using semi-parametric control policies. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, Vol., Iss., 18-22*, 2005.

[4] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 483–490, New York, NY, USA, 2002. ACM Press.

[5] R. Bowden. Learning statistical models of human motion. In *IEEE Workshop on Human Modeling, Analysis and Synthesis, CVPR, 2000.*, 2000.

[6] Matthew Brand and Aaron Hertzmann. Style machines. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 183–192. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[7] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of IROS-2002*, 2002.

[8] Armin Bruderlin and Lance Williams. Motion signal processing. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 97–104, New York, NY, USA, 1995. ACM Press.

[9] F. Bullo and R. Murray. Experimental comparison of trajectory trackers for a car with trailers. In *IFAC World Conference, pages 407–412, 1996.*, 1996.

[10] Lauren J. Clark. Robots serve humans on land, in sea and air. http://web.mit.edu/newsoffice/2005/roboticvehicles-0302.html.

[11] Rémi Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[12] A. Divelbiss and J. T. Wen. Trajectory tracking control of a car trailer system. In *IEEE Transactions on Control Systems Technology, 5(3):269-278*, 1997.

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

[14] E. Feron, E. Frazzoli, and M. Dahleh. Real-time motion planning for agile autonomous vehicles. In *AIAA Conf. on Guidance, Navigation and Control,Denver, August 2000.*, 2000.

[15] E. Frazzoli, M. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicles motion planning. In *Technical report, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1999. Technical report LIDS-P-2468.*, 1999.

[16] Aphrodite Galata, Neil Johnson, and David Hogg. Learning variable length markov models of behaviour. In *Computer Vision and Image Understanding: CVIU*, volume 81, pages 398–413, 2001.

[17] M. Gleicher, H. Shin, L. Kovar, and A. Jepsen. Snap-together motion: Assembling run-time animation. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics, April 2003.*, 2003.

[18] Michael Gleicher. Retargetting motion to new characters. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 33–42, New York, NY, USA, 1998. ACM Press.

[19] Jared Go, Thuc D. Vu, and James J. Kuffner. Autonomous behaviors for interactive vehicle animations. *Graph. Models*, 68(2):90–112, 2006.

[20] Radek Grzeszczuk and Demetri Terzopoulos. Automated learning of muscle-actuated locomotion through control abstraction. *Computer Graphics*, 29(Annual Conference Series):63–70, 1995.

[21] Mark Hachman. U.s. army develops free combat simulation. http://findarticles.com/p/articles/mi_zdext/is_200205/ai_n9517793.

[22] Tae hoon Kim, Sang Il Park, and Sung Yong Shin. Rhythmic-motion synthesis based on motion-beat analysis. *ACM Trans. Graph.*, 22(3):392–401, 2003.

[23] D. Hougen, J. Fischer, M. Gini, and J. Slagle. Fast connectionist learning for trailer backing using a real robot. In *IEEE Int'l Conf. on Robotics and Automation, pages 1917–1922, 1996.*

[24] D. Hougen, M. Gini, and J. Slagle. Rapid, unsupervised connectionist learning for backing a robot with two trailers. In *Proceedings of the IEEE International Conference on Robotics and Automation. pp. 2950-2955.*

[25] Doug L. James and Kayvon Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Trans. Graph.*, 22(3):879–887, 2003.

[26] Rahul Kanakia. Robot car to tackle city streets. http://daily.stanford.edu/article/2006/10/11/robotCarToTackleCityStreets.

[27] Alonzo Kelly and Bryan Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. In *International Journal of Robotics Research, 22(7):583-601*, 2003.

[28] Dr. Steven Kirkpatrick. Ford crown victoria crash simulation. http://www.arasvo.com/crown_victoria/crown_vic.htm.

[29] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings of ACM SIGGRAPH 02, 2002.*, 2002.

[30] John R. Koza. A genetic approach to finding a controller to back up a tractor-trailer truck. In *Proceedings of the 1992 American Control Conference*, volume III, pages 2307–2311, Evanston, IL, USA, 1992. American Automatic Control Council.

[31] Alexis Lamouret and Michiel van de Panne. Motion synthesis by example. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pages 199–212, New York, NY, USA, 1996. Springer-Verlag New York, Inc.

[32] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[33] S. LaValle. Rapidly-exploring random trees: A new tool for path planning. In *Computer Science Dept., Iowa State University., Oct. 1998.*, 1998.

[34] S. LaValle and J. Ku. Randomized kinodynamic planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 1999.

[35] J. Lee, J. Chai, P. Reitsma, J. Hodgins, and N. Pollard. Interactive control of avatars animated with human motion data. In *Proc. SIGGRAPH 2002.*, 2002.

[36] Jehee Lee and Kang Hoon Lee. Precomputing avatar behavior from human motion data. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 79–87, New York, NY, USA, 2004. ACM Press.

[37] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 39–48, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[38] Michael J. Mandel. Versatile and interactive virtual humans: Hybrid use of data-driven and dynamics-based motion synthesis. Master's thesis, 2004.

[39] Sang Il Park, Hyun Joon Shin, Tae Hoon Kim, and Sung Yong Shin. Online motion blending for real-time locomotion generation: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):125–138, 2004.

[40] Ken Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, 1995.

[41] Ken Perlin and Athomas Goldberg. Improv: A system for scripting interactive actors in virtual worlds. *Computer Graphics*, 30(Annual Conference Series):205–216, 1996.

[42] Katherine Pullen and Christoph Bregler. Animating by multi-level sampling. In *CA '00: Proceedings of the Computer Animation*, page 36, Washington, DC, USA, 2000. IEEE Computer Society.

[43] Katherine Pullen and Christoph Bregler. Motion capture assisted animation: texturing and synthesis. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 501–508, New York, NY, USA, 2002. ACM Press.

[44] Craig Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference 1999*, 1999.

[45] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[46] Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–41, /1998.

[47] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 147–154, New York, NY, USA, 1996. ACM Press.

[48] J. K. Rosenblatt. DAMN: A distributed architecture for mobile navigation. In *Proc. of the AAAI Spring Symp. on Lessons Learned from Implememted Software Architectures for Physical Agents*, Stanford, CA, 1995.

[49] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[50] Alessandro Saffiotti. Handling uncertainty in control of autonomous robots. *Lecture Notes in Computer Science*, 1600:381, 1999.

[51] Pedro Santana. Survival kit: A bottom layer for robot navigation. citeseer.ist.psu.edu/article/santana05survival.html.

[52] Arno Schodl and Irfan A. Essa. Controlled animation of video sprites. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 121–127, New York, NY, USA, 2002. ACM Press.

[53] L. M. Tanco and A. Hilton. Realistic synthesis of novel human movements from a database of motion capture examples. In *HUMO '00: Proceedings of the Workshop on Human Motion (HUMO'00)*, page 137, Washington, DC, USA, 2000. IEEE Computer Society.

[54] TORCS. http://torcs.sourceforge.net.

[55] Pennsylvania State University. Virtual crash and tire simulation laboratory. http://www.arl.psu.edu/capabilities/mm_vehicledynmcs_crashlab.html.

[56] Michiel van de Panne, Eugene Fiume, and Zvonko Vranesic. Reusable motion synthesis using state-space controllers. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 225–234, New York, NY, USA, 1990. ACM Press.

[57] Douglas J. Wiley and James K. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications*, 17(6):39–45, / 1997.

[58] Andrew Witkin and Michael Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, 1988.

[59] Andrew Witkin and Zoran Popović. Motion warping. *Computer Graphics*, 29(Annual Conference Series):105–108, 1995.

[60] Katsu Yamane, James J. Kuffner, and Jessica K. Hodgins. Synthesizing animations of human manipulation tasks. *ACM Trans. Graph.*, 23(3):532–539, 2004.