COMPUTER GRAPHICS *forum*

DOI: 10.1111/j.1467-8659.2009.01534.x

COMPUTER GRAPHICS *forum*
Volume 0 (2009), number 0 pp. 1–13

Tactics-Based Behavioural Planning for Goal-Driven Rigid Body Control

Stefan Zickler and Manuela Veloso

Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
{szickler, veloso}@cs.cmu.edu

Abstract

Controlling rigid body dynamic simulations can pose a difficult challenge when constraints exist on the bodies' goal states and the sequence of intermediate states in the resulting animation. Manually adjusting individual rigid body control actions (forces and torques) can become a very labour-intensive and non-trivial task, especially if the domain includes a large number of bodies or if it requires complicated chains of inter-body collisions to achieve the desired goal state. Furthermore, there are some interactive applications that rely on rigid body models where no control guidance by a human animator can be offered at runtime, such as video games.

In this work, we present techniques to automatically generate intelligent control actions for rigid body simulations. We introduce sampling-based motion planning methods that allow us to model goal-driven behaviour through the use of non-deterministic Tactics that consist of intelligent, sampling-based control-blocks, called Skills. We introduce and compare two variations of a Tactics-driven planning algorithm, namely behavioural Kinodynamic Rapidly Exploring Random Trees (BK-RRT) and Behavioural Kinodynamic Balanced Growth Trees (BK-BGT). We show how our planner can be applied to automatically compute the control sequences for challenging physics-based domains and that is scalable to solve control problems involving several hundred interacting bodies, each carrying unique goal constraints.

Keywords: rigid body, control, motion planning, behavioural, constraints, tactics, physics-based, animation

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Animation-Artificial Intelligence[I.2.8]: Plan execution, formation, and generation—Computer Graphics [I.3.5]: Physically based modelling

1. Introduction

Physics-based simulations of rigid body dynamics have become a popular tool for computer animation and video games. After defining initial states (positions, orientations and velocities) of a set of rigid bodies, a physics engine can automatically compute an animation based on the forward simulation of Newtonian Mechanics, robustly dealing with the complex inter-body dynamics such as collisions and friction.

A problem commonly occurs however, when the animation should reach a particular outcome that is not achievable by the pure automatic forward simulation of a physics engine. For example, the goal could be that some of the rigid bodies end up in a particular final state that is unlikely to occur naturally. Additionally, some of these rigid bodies might have to behave actively during the animation and expose some desired visual behaviour pattern. Finally, a rigid body might even need to purposefully manipulate other rigid bodies (through the means of collisions) in order to achieve the goal state for the animation.

In order to satisfy such custom goal constraints and behaviour patterns, intermediate control actions (consisting of forces and torques) need to be applied to some of the rigid-bodies. Manually finding a set of acceptable control actions, however, can pose a difficult and labour-intensive challenge. Oftentimes, an animator may not know how to adjust the available control actions to achieve a particular desired outcome of the simulation. Even with aggressive trial and error, many domains may simply be too complex or too highly constrained to be controlled manually. More importantly, there are several applications, which do not have the luxury of relying on the guidance of an animator. For example, in video games that feature rigid bodies, the automatic generation of intelligent control actions for computer-controlled opponent bodies might be needed. Because the initial state is subject to change (due to an evolving game-state and unpredictable player-input), an animator cannot manually adjust these controls in advance.

In this work, we present techniques to automatically generate intelligent control actions for rigid body simulations. We introduce and evaluate a sampling-based motion planning approach that uses high-level, goal-driven behavioural models to effectively reduce the control search space and thus provide an efficient way for automatically solving complex rigid body domains.

This article is organized as follows: In Section 2, we review related work, including approaches from the computer graphics, robotics and planning communities. In Section 3, we formally define rigid body control as a physics-based motion-planning problem. We then present a taxonomy of the different types of rigid bodies from a control perspective, and discuss some of the unique challenges associated with planning in domains involving these bodies. Next, in Section 4, we introduce sampling-based *Tactics* and *Skills* as a model for infusing goal-driven, high-level behaviours into a randomized motion planner. In Section 5, we introduce and compare two variations of our Tactics-driven, randomized planning algorithm, namely Behavioural Kinodynamic Rapidly Exploring Random Trees (BK-RRT) and Behavioural Kinodynamic Balanced Growth Trees (BK-BGT). In Section 6, we evaluate and compare these algorithms experimentally on several domains, and discuss their performance. Section 7 follows up with a discussion of possible applications, existing limitations, and directions for future work. Finally, we conclude with a summary of the contributions.

2. Related Work

The task of robustly and accurately simulating rigid body dynamics is well-understood [Bar01]. Given a set of rigid bodies with pre-specified positions and velocities, it is possible to apply a set of forces and then integrate the state forward in time, obtaining a new set of positions and velocities. Interbody-dynamics are similarly resolved during simulation by using collision and friction models.

There are multiple techniques for controlling rigid body simulations that focus on user-interaction as a means for solving difficult constraint problems. Popović et al. [PSE*00] present an interactive interface, allowing the user to manipulate objects at any point during the simulation. The system uses random sampling and gradient descent to support the user in the search for constraint-satisfying solutions. Twigg and James [TJ07] enhance this concept significantly by allowing the user to perform spatial queries to rapidly cut down the number of possible solutions. Their system was able to generate desired animations for scenes that involved complex multi-body dynamics. Another recent work by Moss et al. [MLM08] demonstrates how sampling-based planning techniques can be combined with user-provided constraints to solve the animation and navigation planning of deformable bodies.

The inherent downside of these semi-interactive techniques is clearly that some involvement by the animator is required to solve a particular control problem. Instead of relying on user-interaction, we are interested in using planning techniques to automatically solve difficult rigid body control problems. In order to automatically control rigid body control problems, some existing approaches relax the constraints of the problem by relying on the concept of 'physical plausibility' rather than physical correctness. These approaches introduce additional control parameters, such as the slight perturbation of collision normals [BHW96]. Chenney and Forsyth [CF00] took an innovative approach to this relaxation technique by using Markov Chain Monte Carlo in order to compute solutions to pre-specified constraint problems. However, such computations can take a long time to find acceptable solutions. Additionally, object manipulation problems have not been addressed.

A recent example of a planning approach for computer animation by Lau and Kuffner [LK05] utilizes a finite state machine (FSM) to describe the possible motion sequences of humanoid characters. This FSM is then searched using an A*-based algorithm to compute motion paths for animated humanoids navigating environments with moving obstacles. The fact that their behavioural FSM was comprised of a fixed set of pre-recorded motions, allowed them to use pre-computation to gain additional speed-ups [LK06]. While this approach is able to nicely solve the collision-free navigation problem, it does not scale to tasks, which require state-dependent decision making for their solutions, such as is the case with many physics-based games and manipulation domains.

Other recent work has covered the problem of animating human character motions when interacting in complex environments [SKF07], [YKH04]. In terms of object manipulation, however, these approaches again focus solely on navigation planning to/with/around objects, rather than the planning of goal-driven *collision sequences*, which would be required for solving various dynamic manipulation domains.

In robotics, such dynamic control problems are frequently tackled by reactive control architectures [D'A05], [LR04],

[BR01]. One particularly successful approach that has been tested in several real-time adversarial robot domains is the *Skills, Tactics and Plays* architecture (STP) [BBBV05] which uses multiple layers of intelligent reactive building blocks to solve difficult real-time control problems.

Our work aims to combine the concept of intelligent Tactics and Skills [BBBV05] with modern Sampling-Based Motion Planning Techniques. Introduced by LaValle [LaV98], Rapidly Exploring Random Trees (RRT) is a planning technique that rapidly growth a search-tree through a continuous space. The algorithm has been refined by Kuffner and LaValle [KJL00] who have shown that RRT can be adapted to work under Kinematic and Dynamic planning constraints [LKJ01]. One of the most prominent applications of RRT is robot navigation planning [VSA*08], [MKS07]. Further work has focused on increasing RRTs replanning performance to alleviate uncertainty in dynamic robot environments [ZKB07], [FKS06], [BV06].

3. Physics-Based Planning

We can treat automated rigid body control as a general motion planning problem: given a state space X, an initial state $x_{\text{init}} \in X$, and a set of goal states $X_{\text{goal}} \subset X$, a motion planner searches for a sequence of actions $a_1, \ldots, a_n$, which, when executed from x_{init}, ends in a goal state $x_{\text{goal}} \in X_{\text{goal}}$. Additional constraints can be imposed on all the intermediate states of the action sequence by defining only a subset of the state-space to be valid ($X_{\text{valid}} \subseteq X$) and requiring that all states of the solution sequence $x_{\text{init}}, x_1, x_2, \ldots, x_{\text{goal}}$ are elements of X_{valid}.

We use the term *Physics-Based Planning* for action models that aim to reflect the inherent physical properties of the real world. The *Rigid Body Dynamics* model [Bar01] provides a computationally feasible approximation of basic Newtonian physics, and allows the simulation of the physical interactions between multiple mass-based bodies, under the assumption that such bodies are non-deformable. The term *Dynamics* implies that rigid body simulators are second order systems, able to simulate physical properties over time, such as momentum and force-based inter-body collisions. Physics-Based Planning is an extension to *kinodynamic planning* [DXCR93], adding the simulation of rigid body interactions to traditional second order navigation planning.

More formally, the State Space X describes the entire variable space of the physical domain, containing n rigid bodies. A state $x \in X$ is defined as $x = \langle t,\ r_0, \ldots, r_n \rangle$, where t represents time and r_i is the second order state of the i-th rigid body, described by its position, rotation and their derivatives: $r = \langle \alpha, \beta, \gamma, \omega \rangle$ where

α : position ($3D - vector$);
β : rotation ($unit\ quaternion\ or\ rotation\ matrix$);
γ : linear velocity ($3D - vector$);
ω : angular velocity ($3D - vector$).

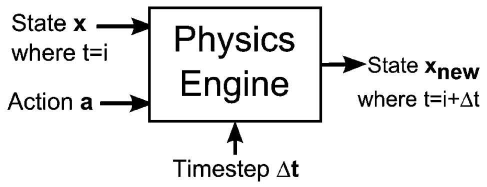

Figure 1: *A Physics Engine computes state transitions.*

The action space A is the set of the applicable controls that the physics-based planner can search over. An action $a \in A$ is defined as a vector of subactions $\langle a_{r_1}, \ldots, a_{r_n} \rangle$, where a_{r_i} represents a pair of 3D force and torque vectors applicable to a corresponding rigid body r_i.

A physics-based planner chooses actions by reasoning about the states resulting from the actuation of possible actions. The state computations are done by simulation of the rigid body dynamics. There are several robust rigid body simulation frameworks freely available, such as the Open Dynamics Engine (ODE), Newton Dynamics and NVIDIA PhysX. Frequently referred to as *physics engines*, these simulators are then used as a 'black box' by the planner to simulate state transitions in the physics space (see Figure 1). Given a current physics state of the world x, in combination with a control action vector a, the physics engine is able to simulate the rigid body dynamics forward in time by a fixed timestep Δt, delivering a complete new planning state x_{new}.

3.1. Rigid body types

Planning for a solution sequence of physical actions is clearly related to the types of bodies present in the domain. We classify the types of bodies in the domains of the physics-based planner, using a hierarchy as shown in Figure 2.

Every body is by definition a *rigid body*. There are *static* rigid bodies that do not move, even when a collision occurs, which are often used to model the ground plane and all non-movable bodies, such as walls and heavy objects. All other bodies are *manipulatable*, meaning that they react to collision forces exerted upon them. Among these, the planner can

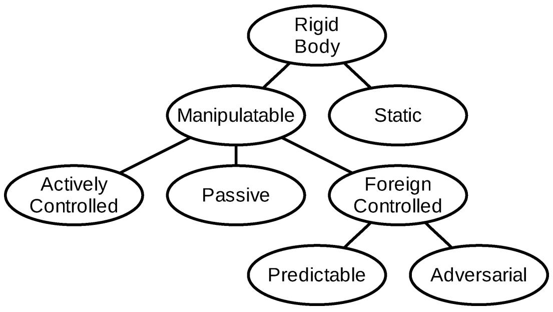

Figure 2: *Rigid Body classes.*

directly control the *actively controlled* bodies, i.e., the planner has available actions directly applicable to these bodies.

Interestingly, there are two different types of bodies that are manipulatable but not actively controlled, namely the *passive* and *foreign controlled* bodies. *Passive* bodies can only be actuated by external influences and interactions, such as being carried or pushed. The *foreign controlled* bodies are actively actuated, but by external control to our planner. Such *foreign controlled* bodies can have *predictable* motion, such as an escalator or a windmill, or be *adversarial*, such as an opponent player to our actively controlled body in a physics-based game.

4. Tactics and Skills

Automatically finding control sequences for animations involving interactions between multiple rigid-bodies can be a very challenging problem, especially if the goal state is difficult to reach and the set of valid intermediate control states is additionally constrained. To efficiently search through the vast physics-based control space, we use a high-level behavioural control model in a novel way to reduce the search space.

Among the many reactive behavioural control architectures, e.g., [D'A05], [LR04], [BR01], we choose the *Skills, Tactics and Plays (STP)* architecture, as it has effectively been used in real-time adversarial robot domains [BBBV05]. In our work, each controllable rigid body is synonymous to an agent in STP. In STP, a *Play* captures the behaviour assignments of a group of multiple agents. A single agent's behaviour is modeled as a reactive *Tactic*, representing a finite state machine (FSM) of lower-level *Skills*, which act as pre-programmed, reactive control blocks. Our work assumes fixed role assignments for all the controlled bodies in the domain, and therefore does not use the multi-agent play-selection component of STP. Traditionally, STP runs online, as a policy-based agent controller, without any physics-aware planning at the Tactics level [BZLV07].

In our work however, a planner uses Tactics and Skills as an action sampling model. Instead of being a reactive controller as in traditional STP, the Tactics' and Skills' new role is to guide the planner's search by imposing constraints on the searchable action space. For this purpose we introduce and formalize a new, probabilistic version of Tactics and Skills that uses random sampling to choose from a set of possible executions.

Before we explain in detail in the next section the use of the Tactics in our planning algorithm, we now further define the non-deterministic Tactics and Skills.

We define a Tactic $\tau = \langle S, \Pi, \sigma_{\text{init}} \rangle$ where

S	is a set of k Skills $\{s_1, \dots, s_k\}$,
Π	is the function $\Pi : \langle \sigma, \sigma', x \rangle \to p$, computing p, thetransitionprobabilityfrom s_σ to $s_{\sigma'}$ depending on the world state x,
σ_{init}	is the index of the initially active Skill$s_{\sigma_{\text{init}}} \in S$,

A Tactic is composed of a set of Skills and a set of possible transitions between these Skills. This model is similar to a classic Nondeterministic Finite State-Machine (with our Skills corresponding to FSM-States), except that in our case, any transition probability p can change dynamically during the life of the Tactic because it is generated by the function $\Pi : x \to p$, thus depending on the world state x.

To perform the actual non-deterministic state transitions between Skills, using the probabilities generated by Π, we define the transition function

$$G : \langle S, \Pi, \sigma, x \rangle \to \sigma',$$

taking a Tactic's set of Skills S, its probability function Π, an index σ pointing to a Skill $s_\sigma \in S$ and a state of the world x to produce a new index σ' pointing to a Skill $s_{\sigma'}$. Algorithm 1 shows how G computes these state transitions probabilistically.

Algorithm 1: Transition Function G

```
Input: $S = \{s_1, \dots, s_k\}, \Pi, \sigma, x$
probSum $\leftarrow 0$;
for $j \leftarrow 1$ to $k$ do
    probSum $\leftarrow$ probSum $+ \Pi(\sigma, j, x)$;
if probSum $> 0$ then
    rndVal $\leftarrow$ sample(0,max(1,probSum));
    tempSum $\leftarrow 0$;
    for $j \leftarrow 1$ to $k$ do
        tempSum $\leftarrow$ tempSum $+ \Pi(\sigma, j, x)$;
        if tempSum $>$ rndVal then
            return $j$;
return $\sigma$;
```

Figure 3 shows an example diagram of a Tactic. Transitions between these Skills can be deterministic (such as between s_1 and s_2), they can be modeled non-deterministically by letting Π return predefined transition probabilities (such as the outgoing transitions of s_2), or they can programmatically depend on any desired property of the current state of the world

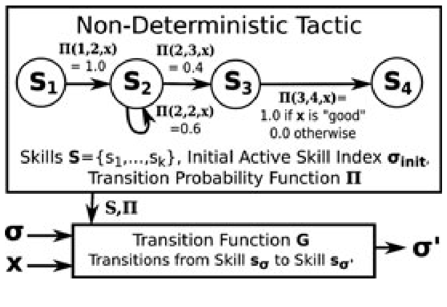
Non-Deterministic Tactic
Π(1,2,x) = 1.0
Π(2,3,x) = 0.4
S1
S2
S3
S4
Π(3,4,x)= 1.0 if x is "good" 0.0 otherwise
Π(2,2,x) =0.6
Skills S={s1,...,sk}, Initial Active Skill Index σinit,
Transition Probability Function Π
S,Π
σ
x
Transition Function G
Transitions from Skill sσ to Skill sσ'
σ'

Figure 3: *Example of a non-deterministic Tactic.*

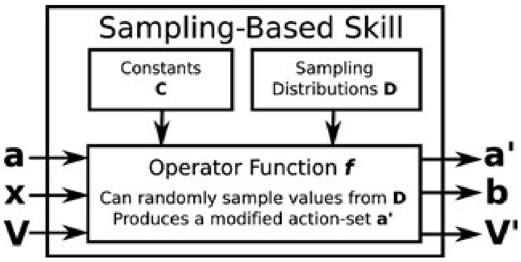

Figure 4: *Structure of a sampling-based Skill.*

x (such as the outgoing transition of s_3). By being modeled in this non-deterministic fashion, each execution of the Tactic can result in a different sequence of state-transitions, thus covering different parts of the search space.

The purpose of the Tactic's individual Skills is to act as non-deterministic controllers that generate actions through the means of random sampling in combination with some domain knowledge.

We define a Skill $s = \langle C, D, f \rangle$ where:

C is a set of constants,
D is a set of sampling distributions,
f is the function $f : \langle C, D, V, x, a \rangle \rightarrow \langle a', V', b \rangle$
where V is a set of variables and
$b$is a boolean.

The Skill's operator function f is the part of the Skill that generates the control actions (see Figure 4). Part of f's input are the state of the world x and a set of actions a that it can modify (by default, a is a zero-filled vector when initially passed into f). f produces the modified set of control actions a' and the boolean 'busy-flag' b (to be explained in more detail in Section 5). C and D are predefined parameters that are part of the Skill itself. The set of constants C may contain any configuration parameters that define how the Skill should operate (for example, the index i of the rigid body r_i that this Skill should generate control actions for). The set of variables V is similar to C, except that it is not stored as part of the Skill and that the operator f is allowed to change its contents by producing V'. As will be shown later in Section 5, V can be used to communicate information between multiple successive calls of the same Skill. Finally, the set of sampling distributions D is the component that makes the Skill non-deterministic because the operator can randomly sample values from these distributions and use them for the computation of control actions. This means that, similarly to the non-deterministic Tactic, multiple calls to f with the same input arguments can each produce a slightly different set of actions a' where the degree of random perturbation depends on the distributions D and on f itself.

The Skill's operator function f, its parameters D, C and the initial value of V are all predefined and assume a certain higher level knowledge about the domain's goal and its physical properties, such as the reasonable ranges of applicable forces and torques. Typically, a single Skill implements a particular control objective such as 'drive toward a sampling-based location,' 'push some target rigid body toward a sampling-based target region,' or 'turn a sampling-based amount.' Designing these actual Skills does take some programmatic effort and it can be argued that some of the Skills should be considered domain-dependent. However, the STP architecture exactly provides for Skills to be used as template-like 'building blocks' that apply to a variety of domains. Given a library of Skills, it is feasible to rapidly create an intelligent tactical model that is well-suited for solving a particular domain. The effectiveness of our approach arises from the fact that non-deterministic Tactics and Skills can encode as much or as little domain-dependent information as desired.

It should be noted that our Tactic and Skill model is a superset that contains their traditional deterministic versions. In our formal model, we can define a *deterministic Tactic* as a Tactic which cannot have multiple outgoing state transitions from any s_σ with probability greater than 0 for any x. Similarly, we define a *deterministic Skill* as one where there is a single deterministic mapping from states to actions. Therefore, in a deterministic Skill, $D = \emptyset$ as no sampling distributions are needed for deterministic decision making.

5. Tactics-Based Planning Algorithm

We are now ready to introduce our planning algorithm. We present two variations of the algorithm that share the same fundamental forward-planning loop, but differ in the way they control the growth of the search tree. We name these two variations as BK-RRT and BK-BGT.

In order to use Tactics and Skills as a sampling model for our planner, we need to store their variable parameters as part of the state space. More formally, a state $x \in X$ is now defined as

$$x = \langle\, t,\ \ r_1, \ldots, r_n,\ \ \sigma_1, \ldots, \sigma_h, \\ V_{00}, \ldots, V_{hk_h},\ \ b_\wedge,\ \ b_1, \ldots, b_h \,\rangle,$$

including the indices to the currently active Skill for each Tactic ($\sigma_1, \ldots, \sigma_h$), as well as the internal variable state V of all k Skills among all h Tactics. Finally, $b_\wedge$ and $b_1, \ldots, b_h$ are all boolean values, indicating whether the entire state x and/or any of its active Skills s_{σ_i} are currently 'busy', to be further explained later in this Section. The number of Tactics h is dependent on the number and types of bodies within the domain. Each actively controlled body has a corresponding non-deterministic Tactic consisting of sampling-based Skills from where its actions are sampled. Additionally, we construct reactive, deterministic Tactics with deterministic Skills to act as prediction models for each foreign-controlled, and especially also to approximate the model of each adversarial body, if existing. Even if an adversary body's exact Tactic is not known, it may be useful to still model its roughly expected behaviour rather than assuming it is static. Finally, passive bodies in the domain are non-actuated and do not require a Tactic.

Algorithm 2 shows BK-RRT. We initialize the search with a tree T containing an initial state $x_{init} \in X$, and set the boolean variable busy to *false*. We then enter the main planning loop, which runs for a predefined domain-dependent maximum number of search iterations z, if no solution is found earlier. On each iteration, assuming busy is *false* (which is the case initially), the algorithm selects a node x from the existing tree T which it will expand from. The difference between BK-RRT and BK-BGT lies precisely on this node selection function. BK-RRT selects nodes similarly to the Rapidly Exploring Random Trees (RRT) [LaV98] search.

The function `SampleRandomState` uses an internal probability distribution to provide a sample y taken from the sampling space Y that is some predefined subspace of X. The function `NN` then finds the nearest neighbour to y among all 'non-busy' nodes in the tree T, according to some predefined distance function. As with traditional RRT, it is important that the sampling space Y, the underlying probability distribution and especially the distance function are all carefully chosen to match the domain. For our domains, we use a simple acceleration-based motion model to compute the minimal estimated time for the controlled rigid body in $x \in T$ to reach its target position and orientation in y.

After selection of the source node x, the algorithm initializes x_{new} to be a copy of x and creates an empty action a. Next, it iterates over each Tactic and, assuming the Tactic's active Skill was not marked as 'busy' ($x.b_i = false$), performs its state transition by calling G. If the Skill was marked as 'busy' then no state transition is performed and its variables are obtained from its previous state. After storing the new Tactics state σ' into x_{new}, the algorithm then calls the active Skill's operator function f. Note, that the contents of the variables V passed into f includes the random RRT sample y because it was added earlier ($V \leftarrow \{y\}$), thus allowing the Skill to make use of y. This in fact implies, that the traditional RRT algorithm is actually a subset of the introduced BK-RRT. If we imagine a Tactic containing a single Skill that only implements the typical RRT 'extend toward y' action, then this BK-RRT search would behave algorithmically identical to standard RRT.

Each call to the operator f returns a tuple including the boolean 'busy-flag' value b_i. The purpose of this boolean flag is to let a Skill report that it is currently executing a linear control task that does not require any branching because its action selection is currently deterministic. Furthermore, because b_i is stored into x_{new}, it signals to the BK-RRT algorithm to not perform any state transitions on the Tactic τ_i that has a currently busy Skill (by not calling G if $x.b_i = false$).

Algorithm 2: BK-RRT

```
Input: Initial state: $x_{init}$, set of goal states: $X_{goal}$, set of $h$
       tactics: $\langle \tau_1, \dots, \tau_h \rangle$, RRT sampling space: $Y$, set of
       valid states: $X_{valid}$, timestep: $\Delta t$, max iterations: $z$.
$T \leftarrow$ NewEmptyTree();
T. AddVertex($x_{init}$);
busy $\leftarrow$ false;
for iter $\leftarrow 1$ to z do
    if busy = true then
        $x \leftarrow x_{new}$;
    else
        $y \leftarrow$ SampleRandomState(Y);
        $x \leftarrow$ NN ($\forall x \in T : x.b_\wedge = false$, y);
    $x_{new} \leftarrow x$;
    $a \leftarrow [0, \dots, 0]$;
    for i $\leftarrow 1$ to h do
        $\langle S = \{s_1, \dots, s_k\}, \Pi, \sigma_{init} \rangle \leftarrow \tau_i$;
        if $x.b_i = false$ then
            $\sigma' \leftarrow G(S, \Pi, x.\sigma_i, x)$;
            $V \leftarrow \{y\}$;
        else
            $\sigma' \leftarrow x.\sigma_i$;
            $V \leftarrow x.V_{i\sigma'}$;
        $\langle C, D, f \rangle \leftarrow s_{\sigma'}$;
        $\langle a, V, b_i \rangle \leftarrow f(C, D, V, x, a)$;
        $x_{new}.\sigma_i \leftarrow \sigma'$;
        $x_{new}.V_{i\sigma'} \leftarrow V$;
        $x_{new}.b_i \leftarrow b_i$;
    busy $\leftarrow b_1 \wedge b_2 \wedge \dots \wedge b_h$;
    $x_{new}.b_\wedge \leftarrow$ busy;
    $x_{new}.\langle r_1, \dots, r_n \rangle \leftarrow$ Sim($x.\langle r_1, \dots, rb_n \rangle$, a, $\Delta t$);
    $x_{new}.t \leftarrow x.t + \Delta t$;
    if $x_{new} \in X_{valid}$ then
        T.AddVertex($x_{new}$);
        T.AddEdge(x,$x_{new}$,a);
        if $x_{new} \in X_{goal}$ then
            return $\langle x_{new}, T \rangle$;
    else
        busy $\leftarrow$ false;
return Failed;
```

Note, that if we have multiple Skills, then the global flag busy is set to the logical *and* of all the individual Skill's b_i flags. If busy is *true* then the algorithm does two important things that can help increase overall planning efficiency. First, during the next node selection iteration, the planner will extend from x_{new} instead of sampling x, thus forcing a continued, greedy single branch expansion for as long as all Skills report that they are 'busy'. Second, the planner marks the entire node x_{new} as 'busy' (by setting $x_{new}.b_\wedge$), thus preventing any future planning iterations to select this node as a source node to grow additional branches from. The algorithm's use of these 'busy' flag(s) can dramatically improve planning efficiency because it prevents the creation of unnecessary branches in the search tree by only allowing the growth of multiple child-branches from states that truly are non-deterministic decision points.

The algorithm is now ready to call the physics-engine through the function `Sim`, using as input the rigid body states $r_1, \ldots, r_n$ contained in the source state x, and the forces and torques defined by a, to simulate the physics of the system forward in time by Δt. The physics-engine returns a new set of rigid body states that are stored into the new state x_{new}. The algorithm then checks whether the resulting state is a valid state ($x_{new} \in X_{valid}$) to ensure that the simulation from x to x_{new} did not violate any constraints that may be required for the domain. This additional validity check is optional and for many domains $X_{valid} = X$, thus $x_{new} \in X_{valid}$ is always true. If accepted, the algorithm adds x_{new} to the search tree T as a child of the chosen node x. If the busy flag was set to true during this iteration, then x_{new} will become the new x during the next iteration.

The complete loop is repeated until the algorithm either reaches the goal, or until it reaches the maximum allowed number of iterations z, at which point the search returns failure for this state. Once the goal is reached, the algorithm simply returns x_{goal} which is then traversed back to the root of T to provide the solution sequence.

5.1. Balanced growth trees

RRTs combination of node-selection and extension has a clear advantage of providing a relatively efficient and probabilistically uniform coverage of the work-space [LaV98]. This uniform expansion however, comes with the tradeoff of computational time. Because BK-RRT requires a nearest neighbour distance lookup through all existing nodes on the tree on every iteration, the computational time grows quadratically as the tree size increases. Also note that, unlike in non-dynamic motion planning, this nearest neighbour lookup process cannot always be trivially sped up through the use of pre-sorting the tree in a data-structure (such as KD-Trees), due to the potential non-linearity and non-symmetry of the time-based distance function that is required in dynamic environments. A related concern is that the user might have too many proverbial knobs to tweak to adapt RRT for a particular domain. Wise decisions need to be made about how to define the distance function and which dimensions of the state-space X are to constitute the sampling-space Y. Once defined, the user needs to choose a well-working sampling-distribution.

Considering all these programmatic and computational challenges, it is questionable whether the advantages of RRT, such as its probabilistically uniform and rapid growth through space, are actually a significant requirement for the tactically constrained domains that we encounter. This is particularly true if none of the Tactic's Skills even make use of the random sample y, and thus in no way implement the traditional RRT 'extend toward y' action. For many tactically constrained models, it might in fact make more sense to abandon any attempts in modelling complicated distance functions that involve knowledge about body kinematics and the state-space, and instead focus on a fast and random growth of the search tree itself.

To test this hypothesis, we introduce a less informed approach that has the only objective to expand the search through our Tactics space in a well-balanced fashion. We call this approach Balanced Growth Trees (BGT). The basic

Algorithm 3: BK-BGT Node Selection

```
if ($\frac{\texttt{AvgLeafDepth(T)}}{\texttt{AvgBranchingFactor(T)}}$) > $\mu$ then
    x ← RandomNonLeaf ($\forall x \in T : x.b_\wedge = false$);
else
    x ← RandomLeaf ($\forall x \in T : x.b_\wedge = false$);
```

planning loop of this algorithm is identical to BK-RRT. The significant difference however, is the method of the selection of the node x. Algorithm 3 shows this new BGT node selection method. Where BK-RRT required a sampling distribution and a nearest neighbour lookup, the only parameter of the BGT node selection is a single constant μ which represents the desired ratio between average leaf depth and average tree branching factor. A large value of μ leads the algorithm to expand further into the future, but creates a 'thinner' tree, whereas a smaller value of μ focuses on a more dense expansion, but with a more limited average time horizon. As it is possible to keep running values of the average branching factor and leaf depth as the tree grows, this node selection scheme is able to run in constant time per node-selection, thus resulting in linear time execution of the entire planning algorithm.

5.2. Many-worlds browsing

Both algorithms, as presented, will stop as soon as a solution has been found. Because we are using a sampling-based planning scheme, there are no immediate guarantees about the optimality of the solution. Although the algorithm does guarantee that the solution adheres to all constraints, it cannot guarantee that there does not exist another, better (by some metric) control sequence that also solves the problem. A common solution to this well-known property of sampling-based planning, is to let the algorithm keep planning even after a valid goal state has been found, thus potentially generating additional valid, but different, control solutions. An animator could then manually inspect and select one of these 'many-worlds' [TJ07]. More interestingly, various metrics, such as plan length or curvature, could be used to automatically compare multiple solutions and let the planner select the best one. A similar selection scheme might be desirable when no complete solution can be found in the alloted time. In this case, the planner can simply select the best intermediate solution state from the tree by using some quality metric.

6. Results

We tested our algorithm in a variety of domains. We implemented the planner in C++, we chose NVIDIA PhysX as the underlying physics engine, and the results were computed on a Pentium 4. We used the Mersenne Twister pseudo-random number generator [MN98] for all non-deterministic decisions and sampling functions throughout our algorithms. We used an action timestep Δt of 1/60th of a second. In this Section, we first introduce the domains with their respective Tactics models and examples of visual results. We then present and analyse the planning performance of our BK-RRT and BK-BGT algorithms for all of these domains. A video with animations of several experiments can be found at `http://www.cs.cmu.edu/~szickler/tactics/`. The same website also contains an Annex to this paper, showing the internal algorithms of several selected Skills in more detail.

6.1. Minigolf domain

In the 'Minigolf' domain, a robot rigid body has the objective to kick a ball through a course with a moving obstacle. Figure 5 shows the Tactics model, allowing the robot-body to wait for a sampling-based amount of time, position itself at a sampling-based location, and then drive into and kick the ball toward a sampling-based target with a sampling-based speed. Note that the 'Wait Sampled Time'-Skill, plays a significant role since the obstacles in the domain are moving. Figure 6 shows one particular solution from this domain.

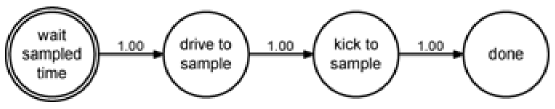

Figure 5: *The Tactic used in the 'Minigolf' domain.*

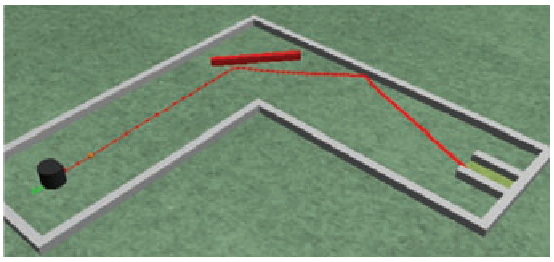

Figure 6: *The 'Minigolf' domain. The bar-shaped obstacle at the centre of the course is rotating at constant velocity and thus represents a predictable, foreign-controlled body. The path shows an example of a legal solution found by the planner for the controllable body, the robot (dark cube, left): the robot-body waits an appropriate amount of time and then accurately manipulates the ball to use the rotating obstacle as a bounce-platform, leading it into the goal position (bottom right).*

6.2. Soccer domain

The 'Soccer' domain (a single soccer attacking situation) significantly enhances the concept of goal-driven manipulation of a passive body and truly demonstrates the unique tactical planning abilities of our planner. In this domain, the controlled body features a rather complex tactical model as shown in Figure 7. This Tactic allows a robot-body to dribble the ball to a sampling-based location, drive toward the ball, drive with the ball ('dribble'), kick or chip the ball toward a sampling-based target near the robot ('minikick' and

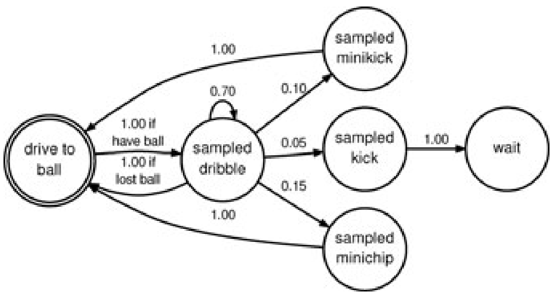

Figure 7: *The Tactic used in the 'Soccer' domain.*

'minichip'), or toward a sampling-based point in the goal ('kick').

Additionally, the domain contains three dynamic adversary rigid bodies which were running deterministic adversarial Tactics to block the ball from the goal. Note that from a planning perspective, this domain represents a very difficult problem. Not only does the robot need to navigate around moving bodies, but it also needs to exert accurate control on the ball to achieve the scoring of a goal in an adversarial environment. This domain generated various interesting solutions, one of which is shown in Figure 8.

6.3. Pool table domain

The 'Pool Table' domain displayed in Figure 9 demonstrates the ability to chain multiple Tactics, as shown in Figure 10. Here, the goal is to hit the blue and yellow balls simultaneously which are then supposed to roll into the purple and red balls respectively to deliver them into their pockets, all without touching any striped balls. Note that in this domain, the pool-balls are treated as active bodies, but with limited

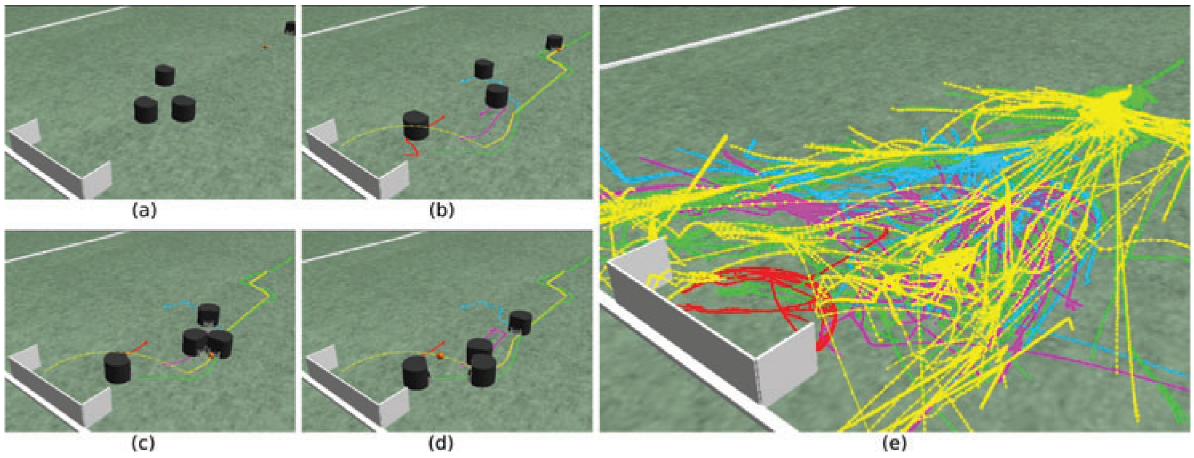

Figure 8: *An example of the 'Soccer' domain. (a) The initial configuration of the bodies. The controllable robot-body is initially at the top right, and the planning goal is to deliver the ball to the goal while avoiding the defenders and goalie robot bodies;(b)-(d) Snapshots of one solution found by the planner; (e) The entire search tree representing the positions of all rigid bodies, including ball (yellow nodes), controlled body (green nodes) and defenders (other colours).*

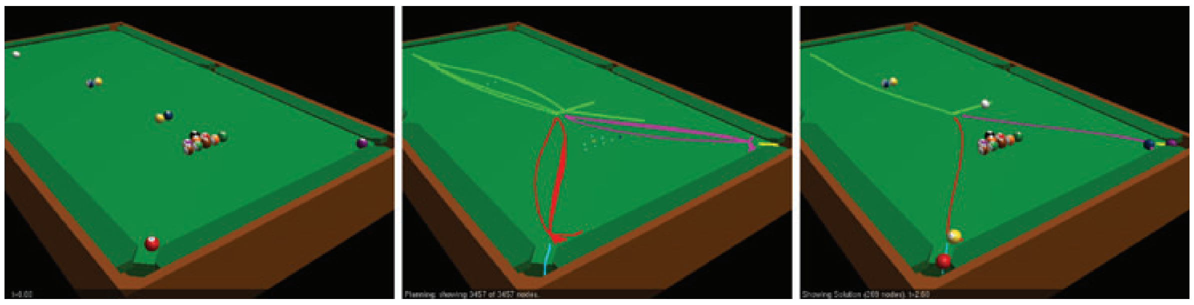

Figure 9: *An example of chaining multiple behavioural models in the 'Pool Table' domain. Given the initial state in the left image, the objective is to use the cue-ball in order to deliver both the red and the purple balls in their closest respective corner pockets without touching any of the striped balls. Our chained behavioural planner finds several acceptable solutions (centre image) and automatically selects one for execution (right image).*

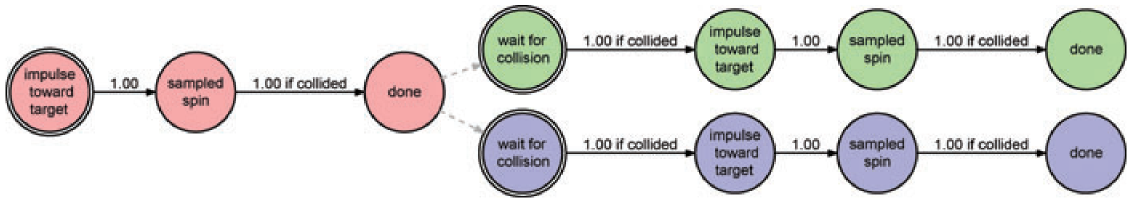

Figure 10: *The chained Tactics used for the 'Pool Table' domain. The cue-ball used the Tactic labelled in red, whereas the two intermediate balls used the green and blue Tactics respectively.*

actuation abilities. While the initial force is applied directly to the cue-ball, the following balls obtain their force passed on through the collision. The balls' Tactics, and in particular the 'sampled spin' Skill is able to slightly perturb their natural path by applying a sampling-based spin toward the target. The ability of the planner to sample from different degrees of these perturbations (thanks to the sampling-based Skill) allows the automatic selection of a solution, which will not collide with any striped balls, and yet requires a minimum of perturbation from their natural passive behaviour.

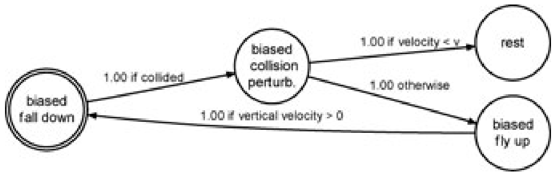

Figure 11: *The Tactic used for the 'Many-Dice' domain. Each die carries its own instance of this Tactic.*

6.4. Many-dice domain

Finally, the 'Many-Dice' domain represents a more traditional computer animation task and aims to demonstrate the ability of our planner to run several hundred Tactical instances concurrently. The objective in this domain is to control the fall of 400 dice, each initialized to a random position. While their fall and collisions are supposed to look physically plausible, they also have the target objective of forming the Eurographics logo. This is achieved by providing each die with a separate instance of the Tactic shown in Figure 11. This domain is tricky because while we are able to actuate the individual dice, we are interested in achieving the illusion of a natural looking free-fall. This is very tough when the dice still have to passively react to various collisions with spherical objects and with other dice. In order to do so, this Tactical models carries multiple Skills, each representing a different state of the fall based on a die's current position and velocity. These Skills are again sampling-based, but probabilistically biased to create motions toward the die's final target position (which is determined at initialization). Figure 12 shows a visual result of this domain.

Table 1: *Numerical performance comparison of BK-RRT and BK-BGT for different domains. Each value represents an average over several experiments.*

	BK-RRT		**BK-BGT**[1]	
Domain	Time	Nodes	Time	Nodes
Soccer	9.6s	9752	11.1s	11088
Minigolf	5.3s	13042	2.5s	8094
Pool Table	2.7s	451	0.4s	191
Many-Dice	n/a[2]	n/a[2]	204s	preset[2]

[1] $\mu = 100$ for Soccer, $\mu = 1000$ otherwise.
[2] Many-Dice was tested with BK-BGT only, using a preset tree size.

6.5. Performance

Table 1 shows average planning times and tree sizes for all of the above domains. Our planner was able to deliver many viable solutions for all of these domains. The average planning times and tree sizes suggest that, for tactically constrained domains, it is in fact viable to use a more scalable, less informed algorithm such as BK-BGT over BK-RRT. Figure 13

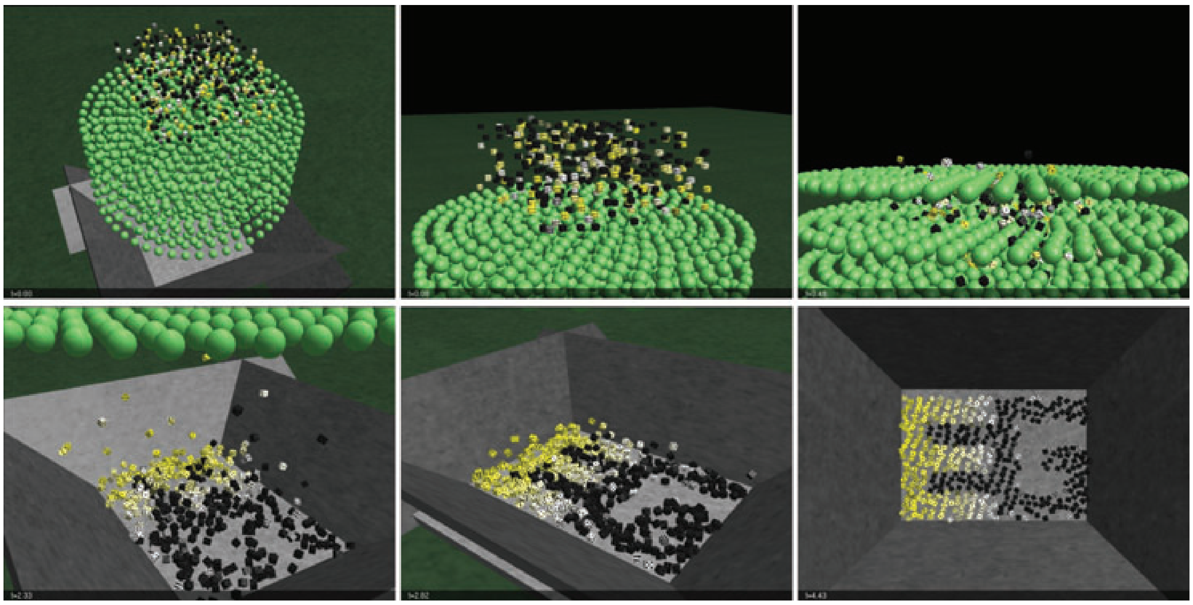

Figure 12: *A solution sequence of the 'Many-Dice' domain. Four hundred randomly initialized dice fall toward a grid of spherical obstacles. After bouncing through the grid, they fall into a bucket where they come to a rest. Our planner successfully found a set of control actions which led the dice to their goal state of forming the Eurographics logo, while at the same time appearing physically plausible during their fall.*

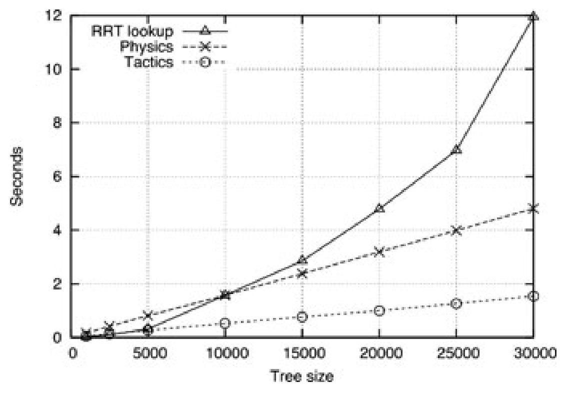

Figure 13: *BK-RRT analysis of accumulated time spent in node selection, physics and tactics computations respectively over increasing tree sizes.*

shows a timing analysis of the BK-RRT algorithm, breaking down the total time spent during search into RRT node selection, Tactics and Skills execution and physics engine simulations. As expected, RRTs node selection quickly becomes the bottleneck for larger search trees, due to its quadratic scalability.

7. Discussion

We now discuss the potential real-world applications of our approach. We also elaborate on some of its remaining limitations and trade-offs. Finally, we present several interesting directions for future work.

7.1. Applications

There are several real-world applications where our introduced approach should be considered beneficial. The first one is computer animation of scenes involving rigid body interactions. The advantage of our automated planning approach becomes clear for domains like 'Many-Dice' where it is simply not feasible for an animator to manually adjust the physics interactions of all the bodies involved. Instead, it makes much more sense to define a simple Tactic and Skill model that can be replicated for all the dice. Furthermore, this very same model is likely to be applicable for many other problem instances in the same domain. In the case of 'Many-Dice', for example, our Tactics model would still work, even if we changed the valid goal state from the pattern 'EG' to some other shape, whereas a manual animator would have to start over from scratch to re-adjust the paths of all the dice involved.

Beyond traditional animation, there are types of applications where requiring manual animation is not only inefficient, but simply impossible. These are any applications where the initial configuration and/or goal states are not known at the time of development (where an animator would be available), but only at runtime. This is the case in video games and robot control software. Given the demonstrated planning performance, we believe that our approach could already be applied as an opponent A.I. in turn-based physics games, such as Pool or Golf. In Section 7.3, we will furthermore discuss potential performance improvements and algorithmic optimizations which could allow this technique to be applied in interactive real-time applications in the near future.

7.2. Limitations, and trade-offs

In terms of limitations, our approach is certainly not immune against the general curse of dimensionality in planning. While intelligent Skills and Tactics are able to rapidly increase our planner's ability to solve domains involving rigid body interactions, it is likely that this approach will require long planning times when each controllable body has an excessive amount of controllable degrees of freedom. Additionally, the more complex the body's control model, the more complex is the required design of the Tactic and Skills which make intelligent use of this control.

Our approach gives a user the flexibility to infuse as much Tactical knowledge into the planning stage as desired. While the definition of such knowledge does involve some work by designing the actual Tactics and Skills, it will also likely result in better planning performance. It should also be noted, that many of the Skills can be considered template-like 'building blocks' that apply to a variety of domains and therefore can be re-used. Since our model allows arbitrary complexity of Tactics and Skills (with the extreme case being a dead-simple Tactic that implements standard RRT), we feel that our approach should be treated as a valuable extension to traditional planning models.

7.3. Future work

Improving the planning performance should be one of the main focus points of future work. On the computational side, there are several promising approaches, such as performing the algorithm's physics-computations on the GPU. Additionally, it might be interesting to parallelize the entire planning algorithm to be run on modern multi-core processors. One way to achieve this would be by designating different branches of the search tree to different processing cores.

On the algorithmic side, it would be interesting to look into the concept of finite horizon planning. In particular, if used in real-time video games, our planner needs to be able to operate within a fixed amount of time and possibly under uncertain, unpredictable future input. Limiting the depth of the search tree in combination with frequent re-planning is a promising approach to achieve this.

Finally, it might also be worthwhile to investigate the use of supervised machine learning techniques to automatically 'train' a particular tactical model by optimizing its internal transition probabilities and sampling distributions, thus creating the strongest and most efficient planner possible for a given domain.

8. Conclusion

We presented the rigid body control planning problem and its challenges. We introduced and formalized a non-deterministic version of *Tactics and Skills* as a model for infusing goal-driven, high level behaviours into a sampling-based motion planner, thus reducing its action space and making search feasible. We introduced a Tactics-based planning algorithm with two different techniques for the search tree expansion, one based on RRT, and one based on the novel 'Balanced Growth Trees'. We experimentally demonstrated the effectiveness of our approach in challenging domains. Finally, we discussed these results, potential applications, as well as directions for future work.

References

[Bar01] BARAFF D.: Physically based modeling: Rigid body simulation. *SIGGRAPH Course Notes, ACM SIGGRAPH* (2001).

[BBBV05] BROWNING B., BRUCE J., BOWLING M., VELOSO M.: Stp: Skills, tactics and plays for multi-robot control in adversarial environments. *IEEE Journal of Control and Systems Engineering 219* (2005), 33–52.

[BHW96] BARZEL R., HUGHES J., WOOD D.: Plausible motion simulation for computer graphics animation. *Computer Animation and Simulation* (1996), 184–197.

[BR01] BEHNKE S., ROJAS R.: A hierarchy of reactive behaviours handles complexity. *Balancing Reactivity and Social Deliberation in Multi-Agent Systems: From Robocup to Real-World Applications* (2001).

[BV06] BRUCE J., VELOSO M.: Safe multi-robot navigation within dynamics constraints. *Proceedings of the IEEE 94*, 7 (2006), pp. 1398–1411.

[BZLV07] BRUCE J., ZICKLER S., LICITRA M., VELOSO M.: *CMDragons 2007 Team Description*. Tech. rep., Tech Report CMU-CS-07-173, Carnegie Mellon University, School of Computer Science, 2007.

[CF00] CHENNEY S., FORSYTH D.: Sampling plausible solutions to multi-body constraint problems. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New Orleans, USA, 2000), pp. 219–228.

[D'A05] D'ANDREA R.: The Cornell RoboCup Robot Soccer Team: 1999–2003. New York, NY: Birkhauser Boston, Inc, pp. 793–804, 2005.

[DXCR93] DONALD B., XAVIER P., CANNY J., REIF J.: Kinodynamic motion planning. *Journal of the ACM (JACM) 40*, 5 (1993), 1048–1066.

[FKS06] FERGUSON D., KALRA N., STENTZ A.: Replanning with RRTs. In *Robotics and Automation. ICRA'06: Proceedings of the IEEE International Conference* (Orlando, FL, USA, 2006), pp. 1243–1248.

[KJL00] KUFFNER Jr J., LAVALLE S.: RRT-connect: An efficient approach to single-query path planning. In ICRA'00: *Robotics and Automation, Proceedings of the IEEE International Conference on 2* (San Francisco, USA, 2000).

[LaV98] LAVALLE S.: *Rapidly Exploring Random Trees: A New Tool for Path Planning.* Tech. Rep. TR, Computer Science Department, Iowa State University, 1998, pp. 98–11.

[LK05] LAU M., KUFFNER J.: Behavior planning for character animation. In *Proceedings of the ACM SIGGRAPH'05/Eurographics symposium on Computer animation* (Los Angeles, USA, 2005), pp. 271–280.

[LK06] LAU M., KUFFNER J.: Precomputed search trees: planning for interactive goal-driven animation. In *Proceedings of the ACM SIGGRAPH'06/Eurographics symposium on Computer animation* (Boston, USA, 2006), pp. 299–308.

[LKJ01] LAVALLE S., KUFFNER JR J.: Randomized kinodynamic planning. *The International Journal of Robotics Research 20*, 5 (2001), 378.

[LR04] LAUE T., ROFER T.: A behavior architecture for autonomous mobile Robots based on potential fields. *Proccedings of the 8th International Workshop on RoboCup* (Lisboa, Portugal, 2004), pp. 122–133.

[MKS07] MELCHIOR N., KWAK J., SIMMONS R.: Particle RRT for Path Planning in very rough terrain. In *NASA Science Technology Conference (NSTC'07)* (Maryland, USA, 2007).

[MLM08] MOSS W., LIN M. C., MANOCHA D.: Constraint-based Motion Synthesis for Deformable Models. *Computer Animation and Virtual Worlds 19* (July 2008), 421–431.

[MN98] MATSUMOTO M., NISHIMURA T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS) 8*, 1 (1998), 3–30.

[PSE*00] Popović J., Seitz S., Erdmann M., Popović Z., Witkin A.: Interactive manipulation of rigid body simulations. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New Orleans, USA, 2000), pp. 209–217.

[SKF07] Shapiro A., Kallmann M., Faloutsos P.: Interactive motion correction and object manipulation. In *I3D'07: Proceedings of the Symposium on Interactive 3D graphics and games* (New York, NY, USA, 2007), ACM, pp. 137–144.

[TJ07] Twigg C., James D.: Many-worlds browsing for control of multibody dynamics. *ACM Transactions on Graphics (TOG) 26*, 3 (2007).

[VSA*08] Vahrenkamp N., Scheurer C., Asfour T., Kuffner J., Dillmann R.: Adaptive motion planning for humanoid robots. In *Intelligent Robots and Systems, IROS'08. IEEE/RSJ International Conference* (Nice, France, 2008), pp. 2127–2132.

[YKH04] Yamane K., Kuffner J., Hodgins J.: Synthesizing animations of human manipulation tasks. *ACM Transactions on Graphics (TOG) 23*, 3 (2004), 532–539.

[ZKB07] Zucker M., Kuffner J., Branicky M.: Multipartite RRTs for rapid replanning in dynamic environments. In *Proc. IEEE Int. Conf. on Robotics and Automation* (Roma, Italy, 2007), pp. 1603–1609.