Playing Creative Soccer: Randomized Behavioral Kinodynamic Planning of Robot Tactics

Stefan Zickler and Manuela Veloso

Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
{szickler,veloso}@cs.cmu.edu

Abstract. Modern robot soccer control architectures tend to separate higher level tactics and lower level navigation control. This can lead to tactics which do not fully utilize the robot's dynamic actuation abilities. It can furthermore create the problem of the navigational code breaking the constraints of the higher level tactical goals when avoiding obstacles. We aim to improve such control architectures by modeling tactics as sampling-based behaviors which exist inside of a probabilistic kinodynamic planner, thus treating tactics and navigation as a unified dynamics problem. We present a behavioral version of Kinodynamic Rapidly-Exploring Random Trees and show that this algorithm can be used to automatically improvise new ball-manipulation strategies in a simulated robot soccer domain. We furthermore show how opponent-models can be seamlessly integrated into the planner, thus allowing the robot to anticipate and outperform the opponent's motions in physics-space.

1 Introduction

Multi-agent control architectures as encountered in robot soccer tend to follow a common paradigm: controls are layered in a hierarchical structure with higher level decision-making controls located at the top, flowing to more concrete, lower level controls as we move towards the bottom. One particularly successful approach is the "Skills, Tactics, and Plays" (STP) model [1]. Here, a multi-agent *playbook* performs role-assignment at the top of the hierarchy. These roles are then executed by the individual agents and modeled in the form of *tactics* which again consist of a state machine that can invoke even lower level *skills*. Navigational controls are modeled on the skill level and can range from very simplistic reactive controls to more elaborate motion planning techniques.

While this STP approach has proved very successfully in several real-world robot domains, there exists at least one remaining limitation. The lower level motion planning code is generally unaware of the higher level strategic goals of the game. Similarly, the higher level strategy is mostly unaware of the inherent dynamics of the physical world that the robot is acting within. While the navigational control might for example perform obstacle-avoidance to prevent collisions, it might at the same time also violate the assumptions of the higher

level tactic, such as being able to dribble the ball successfully towards the goal. A related issue is the problem of predictability. There might exist several tactics that the agent can chose from, but each of them will typically perform its task deterministically, given the current state of the world. There is little to none creative variation in the motions that the robot executes, which can ultimately not only lead to missed opportunities, but also to the opponent team quickly adapting to one's strategy.

Our work aims to solve these problems by embedding the skills and tactics components of STP into a kinodynamic planning framework. By treating soccer as a physics-based planning problem, we are able to automatically generate control sequences which achieve higher level tactical goals while adhering to any desired kinodynamic navigation constraints. To generate such control sequences, we introduce Behavioral Kinodynamic Rapidly-Exploring Random Trees: a randomized planning algorithm that allows us to define tactics as non-deterministic state machines of sampling-based skills. Finally, we show that our approach is able to creatively improvise new ball-manipulation strategies in a simulated soccer domain and compare its performance with traditional static tactic approaches.

1.1 Related Work

Several multi-agent control architectures exist that are applicable to robot soccer [1–5]. Our presented approach aims to improve the single-robot control components of such layered approaches and is able to work underneath any higher level multi-agent role assignment methodology, such as playbooks [1], state machines, or even market-based methods [6]. In terms of single agent control, our work adopts the concept of tactics as a state machine of skills as presented in [1]. However, in our case, both the tactics and skills are modeled inherently probabilistically and act as search axioms inside of a kinodynamic planner. We are currently unaware of any other approach that has unified planning, strategy, and lower-level dynamics in such a way.

Our approach builds heavily upon kinodynamic randomized planning. Introduced by LaValle [7], Rapidly-Exploring Random Trees (RRT) are a planning technique which allows rapid growth of a search tree through a continuous space. The algorithm has been refined by Kuffner and LaValle [8] and has been used for collision free motion planning in many robotics applications, such as in the RoboCup Small Size team by Bruce and Veloso [9, 10]. LaValle and Kuffner [11] successfully showed that RRT can be adapted to work under Kinematic and Dynamic planning constraints.

Internally, our planning algorithm uses a rigid-body simulator to perform its state transitions. The task of robustly and accurately simulating rigid body dynamics is well-understood [12] and there exist several free and commercial simulators, such as the Open Dynamics Engine (ODE), Newton Dynamics, and Ageia PhysX.

2 Modeling Soccer as a Kinodynamic Planning Problem

```
T.AddVertex(x_init);
for k ← 1 to K do
    x_sample ← SampleRndState();
    x_source ← NNeighbor(T,x_sample);
    x_new ← Extend(x_source,x_sample);
    if IsValidExtension(x_new) then
        T.AddVertex(x_new);
        T.AddEdge(x_source,x_new);
        if x_new ∈ X_goal then
            return x_new ;
        end
    end
end
return Failed;
```

Algorithm 1: Standard RRT

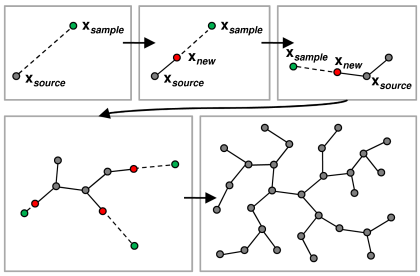

Fig. 1. A visual demonstration of the standard RRT algorithm.

Instead of artificially separating our control model into higher level tactics and lower level navigation, we treat soccer as a general control planning problem. Given an initial state x_{init} in the state-space $\mathbf{X}$, our algorithm is to find an acceptable sequence of control parameters which lead us to a final state x_{final} located within a predefined goal region $X_{goal} \subseteq \mathbf{X}$. Additional constraints might exist on the intermediate states of our action sequence, such as collision avoidance or velocity limits. A very common tool to solve such problems is randomized planning.

Randomized planning techniques are search algorithms which apply probabilistic decision making. One advantage of randomized approaches is that we are able to shape their behavior by modifying the underlying sampling distributions. More importantly, randomized techniques tend to work well in continuous environments as they are less likely to "get stuck" due to some predefined discretization of the set of possible actions. Rapidly-Exploring Random Trees (RRT) is a popular example of modern randomized planning techniques. Standard RRT as introduced by LaValle [7] is outlined in algorithm 1. We start with a tree T containing a single node x_{init} representing our initial state. We then continuously sample random points x_{sample} located anywhere within our domain. We locate the node x_{source} in T that is closest to x_{sample} using a simple nearest neighbor lookup. We extend node x_{source} towards x_{sample}, using a predefined constant distance, thus giving us a new node, x_{new}. Finally, we add x_{new} to T with x_{source} being its parent. This process is repeated until we either reach the goal or give up. A visual example of a typical RRT search is given in figure 1.

Since robot soccer is a problem which can contain kinematic and dynamic constraints, we want our planner to operate in second order time-space. In the planning literature this is commonly referred to as kinodynamic planning [13].

RRT can be adapted to work in a kinodynamic environment as shown by LaValle and Kuffner [11].

Finally, it is important to point out that planning robot soccer tactics is significantly more challenging than a pure path planning problem. The crucial difference is that in robot soccer, our agent's motions and the higher level planning goal are actually allowed to be disjoint subsets of our state space. For example, our planning goal is typically defined in terms of the ball being delivered to a particular target, whereas our actions can only be applied to the robot, but not the ball itself. Thus, the only way to manipulate the ball, is through the interactions defined by our physics simulator, such as rigid body dynamics. This prevents us from using a simple goal-heuristic such as modifying RRT's sampling distribution. Without the bias of a useful goal heuristic, our search would become infeasible.

2.1 Behavioral Kinodynamic Planning for Robot Tactics

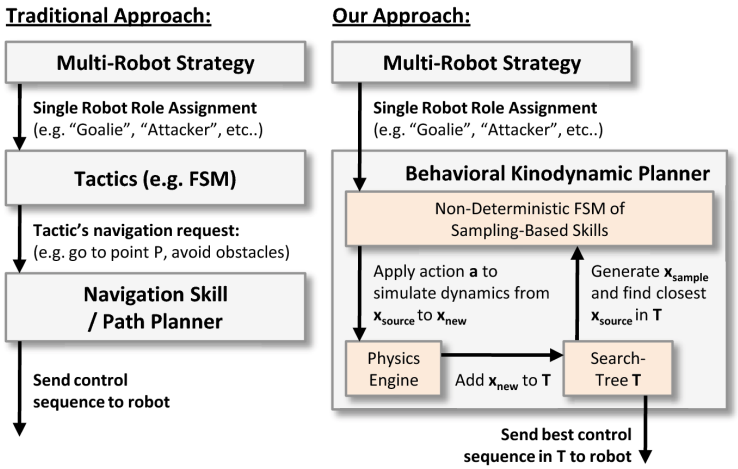

Fig. 2. A diagram of a traditional STP-style approach (left) in comparison to our behavioral kinodynamic planning approach (right).

To guide our search through dynamics space, we embrace and extend the concept of tactics and skills as presented by Browning and colleagues in [1]. For our work, tactics and skills still follow their traditional role to act as informed controllers of our robot by processing the input of a particular world state and providing an action for our robot to execute. In other words, they act as an implementation of a robot's behavior (such as "Attacker" or "Goalie") and contain some kind of heuristic intelligence that will attempt to reach a particular goal state.

However, the significant difference is that in our case, both the tactic and its skills reside within a kinodynamic planner. That is, instead of greedily executing a single tactic online, we use planning to simulate the outcome of many different variations of a tactic in kinodynamic space. Our agent then selects one particularly good solution for execution. To achieve this variation in tactic execution, we modify the traditional definition of a tactic to be modeled probabilistically as a Nondeterministic Finite State Machine. Similarly, skills are sampling-based, and thus deliver nondeterministic actions. A diagram showing this structural difference is shown in figure 2.

In the nondetermininistic FSM, transitions are modeled probabilistically. The probabilities are defined by the programmer and act as a behavioral guideline for how likely the tactic is to chose a particular sequence of state-transitions. By being modeled in this nondeterministic fashion, each execution of the tactic can result in a different sequence of state-transitions, thus reducing predictability of our robot's actions. However, we are in full control over how much "creative freedom" we want to provide to the tactic by adjusting the transition probabilities.

To further increase the freedom of choice in our agent's behavior, we also modify our Skills to act non-deterministically. In traditional STP, a skill would normally perform an analytic reactive control given the current state of the world. For example, a traditional "shoot-on-goal" skill would aim for the center of the largest opening in the goal, and take a shot. This not only leads to very high predictability, but it also ignores the fact that there might exist other shots that could be more likely to succeed, given a particular dynamics state of the world. In our new non-deterministic version of a "shoot-on-goal" skill, we instead use sampling to select randomly from a set of target points throughout the goal area. We then rely on the fact that our planner will execute this skill many times, always selecting a different sample point and simulating the skill's outcome in kinodynamic space.

Given this non-deterministic model of tactics and skills, we can now use search to cover many different simulated executions of the tactic in our kinodynamic search space. The goal of this methodology is to be able to search over a variety of executions and find one particular instance that leads us successfully to the goal state. The inherent creativity of this approach arises from the fact that skills no longer execute deterministic motions based on the state of the world, but instead can chose randomly from different perturbations of their typical behavior. The way that these skills are then chained together into a control sequence is again a probabilistic process, thus allowing the planner to come up with new, and unique solutions to the problem. Since all the planning occurs inside of a kinodynamic framework, we can guarantee that the solutions will satisfy any physics-based constraints that we might have (such as collision-free navigation).

We are now ready to formalize the concept of Behavioral Kinodynamic RRT. The State Space $\mathbf{X}$ describes the entire modifiable space of our physical domain. More concretely, a state $x \in \mathbf{X}$ is defined as $x = [t, fsm, r_0, \ldots, r_n]$ where t represents time, *fsm* represents the agent's internal behavioral FSM-state, and

r_i represents the state of the i-th rigid body in our domain. A rigid body state in a second order system is described by its position, rotation, their derivatives, and any additional state. That is, $r_i = [p, q, v, \omega, o]^T$ where

p : position (3D-vector)
q : rotation (unit quaternion or rotation matrix)
v : linear velocity (3D-vector)
ω : angular velocity (3D-vector).
o : optional additional state-variables (e.g. robot's actuators).

It is important to point out that r may include both active and passive rigid bodies. Active rigid bodies are the ones which we can directly apply forces to as an execution of our planning solution, such as robots. All other bodies which can solely be manipulated through rigid body dynamics (e.g. the ball) are considered passive. For simplicity, we will assume that the total number of rigid bodies in our domain stays constant.

The Action Space **A** (also known as control space) is defined by the set of possible actions of our agent. An action $a \in \mathbf{A}$ of an agent consists of force, torque, and possibly actuation commands, such as kick or dribble. Robot-dependent kinematic constraints are modeled by the possible torques and forces that can be applied to the robot.

The Sampling Space **S** is what we draw our random samplings from. It is a subset of the state-space, i.e. $\mathbf{S} \subseteq \mathbf{X}$.

```
T.AddVertex(x_init);
for k ← 1 to K do
    s_random ← SampleRandomState();
    x_source ← KinematicNearestNeighbor(T,s_random);
    fsm_state ← FsmTransition(x_source.fsm);
    if IsValidFSMstate(fsm_state) then
        a ← ApplyFsmBehavior(fsm_state,x_source,s_random);
        x_new ← Simulate(x_source,a,Δt);
        x_new.t ← x_source.t + Δt;
        x_new.fsm ← fsm_state ;
        if IsValidPhysicsState(x_new) then
            T.AddVertex(x_new);
            T.AddEdge(x_source,x_new,a);
            if x_new ∈ X_goal then
                return x_new ;
            end
        end
    end
end
return Failed;
```

Algorithm 2: Behavioral Kinodynamic RRT

Using this terminology, we can now define Behavioral Kinodynamic RRT as shown in algorithm 2. We start with a tree T containing an initial state $x_{init} \in \mathbf{X}$. We then enter the main RRT loop. The function `SampleRandomState` uses an internal probability distribution to provide us with a sample s_{random} taken from the sampling space $\mathbf{S}$. It is important that the sampling space $\mathbf{S}$ and the probability distribution are carefully chosen to match the particular robot domain. For a typical omni-directional robot movement, the sampling space $\mathbf{S}$ might consist of three dimensions, representing a point and orientation in space. Similarly, the sampling distribution might be spread uniformly throughout the confines of the domain. The `KinematicNearestNeighbor` function defines a distance metric between a sample s_{random} and an existing state x from our tree T.

The definition of a consistent distance metric is crucial for a correct operation of the algorithm. Since we are planning in second order timespace, using an Euclidean nearest neighbor distance approach will fail. This is because the Euclidean distance between two nodes' positions ignores the fact that the nodes have velocities attached to them. Thus, distance is not a good indicator of how much time it will take to reach a particular node. A much more reliable approach is to define a metric based on estimated time to get from $x_i \in T$ to s_{random}. This can be achieved by using a simple acceleration-based motion model to compute the minimal estimated time for the robot to reach its target position and orientation. While this certainly will not always constitute an accurate prediction of the time traveled (in particular because we are not taking potential obstacles into account), it is still a good heuristic for the nearest neighbor lookup.

Once we have located the nearest neighbor x_{source} towards s_{random}, we now call `FsmTransition` to perform a nondeterministic FSM state transition giving us the new internal FSM state `fsm_state`. We then make sure that this state is valid by ensuring that the FSM has not reached any end-condition. Finally, we apply the behavior associated with `fsm_state` by calling `ApplyFsmBehavior`. The actual FSM behavior can be any skill, ranging from a simple "full stop" to much more elaborate controls. It is important to note, that each FSM behavior is able to make full use of the sample s_{random}. For example, we might imagine a skill called "move towards point" which will accelerate our robot towards the sample s_{random}. This in fact implies, that the traditional kinodynamic RRT algorithm is actually a subset of the introduced Behavioral Kinodynamic RRT. If we imagine a behavioral FSM with a single behavior that implements the typical RRT extend-operator then this behavioral RRT would behave algorithmically identical to standard RRT.

The result of `ApplyFsmBehavior` will be an action a describing a vector in the action-space $\mathbf{A}$. We can now access our physics-engine, load the initial state x_{source}, apply the forces and torques defined by a, and simulate Δt forward in time which will provide us with a new state x_{new}. We then query the physics engine to ensure that the simulation from x_{source} to x_{new} did not violate any of our dynamics constraints, such as collisions. If accepted, x_{new} is added to T. This entire process is repeated until we either reach the goal, or give up. Once

the goal has been reached, we simply backtrack until we reach the root of T and reverse that particular action sequence.

The presented pseudocode will stop as soon as a solution has been found. Because we are using a randomized planning scheme, there are no immediate guarantees about the optimality of the solution. However, one could easily enhance the algorithm to keep planning until a solution of a desired quality has been found. Various metrics, such as plan length or curvature could be used to compare multiple solutions.

2.2 Domain and Robot Modeling

To plan in dynamics space, an accurate physics model of the domain is required. Modern physics simulators allow the construction of any thinkable rigid-body shapes and are sufficiently reliable in simulating motions and collisions based on friction and restitution models. In our approach, this domain description is treated as a module and can be interchanged independently of the planner, and its tactics and skills. Skills are similarly designed with platform independence in mind, allowing an abstracted description of motion, sensing, and actuation which will then be translated into physics-forces defined by a modular platform-dependent robot-model.

2.3 Opponent Modeling

Since we are dealing with a robot soccer domain, we need to assume that we are planning against an adversary. Note, that our planner stores the entire physics state x for any node when performing its search through our dynamics time-space. This conveniently allows us to integrate an opponent model into the search-phase. Similar to our own behavior, such opponent model can be reactive to the given state of the world as defined in a particular x_{source}. For example, we can model an opponent which will attempt to steal the ball from us, by defining a simple deterministic skill. Using this model, our planner will anticipate and integrate the opponent's motions during the planning phase and only deliver solutions which will out-maneuver the predicted opponent's motions. In algorithm 2, this opponent model would apply its actions for any given plan-state transition, similar to `ApplyFsmBehavior`. The `Simulate` function will then not only simulate the actions of our own agent, but also the ones of the opponent. Obtaining an accurate opponent model is certainly a completely different challenge that we will not cover in this paper. Nevertheless, even simple assumptions, such as extrapolating the opponent's linear motion, or assuming a simple "drive to ball" strategy, should be significantly preferable over the assumption of a static opponent.

3 Experimental Results

We present results in a simulated environment that resembles the RoboCup Small-Size League. The simulated robot models represent our actual Small-Size

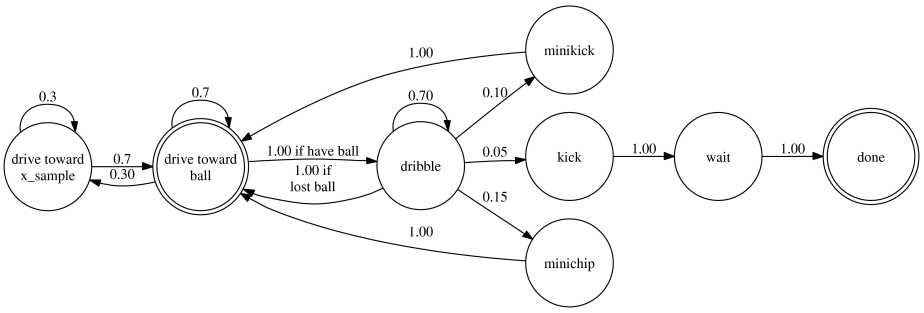

Fig. 3. A behavioral nondeterministic finite state machine used for generating dribbling sequences.

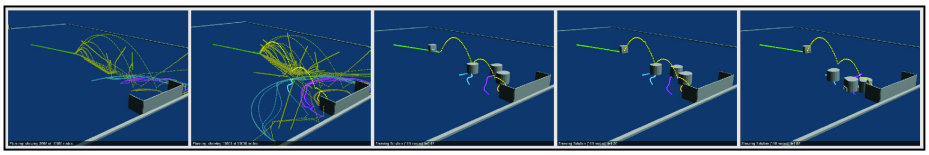

Fig. 4. An example of planning an attacker-tactic with chip and flat kicks. The left two images show the growth of the Behavioral Kinodynamic RRT through the domain (robots and ball are not displayed). Yellow nodes represent the position of the ball. Green nodes represent our agent, any other nodes represent the three defenders respectively. The right three images highlight the selected solution and show the attacker executing it.

RoboCup robots. Our simulated model has been proved successfully for developing RoboCup code and models the motion, sensing, and ball-manipulation capabilities of our robots with good accuracy. The planning framework was implemented in C++. Ageia PhysX was chosen as the underlying physics engine. The results were computed on a Pentium 4 processor running Linux. The action timestep Δt used in all tests was 1/60th of a second.

The first experiment used a probabilistic "Attacker" tactic to shoot a goal against an opponent defense. The skills within this tactic were a sampling-based flat kick and a sampling-based chip-kick. Kick-strength and kick-aiming for both of those skills were modeled as free planning variables using uniform sampling distributions. The decision between the two skills was also a free planning variable because it was modeled using the nondeterministic state machine. The opponent defense was modeled using a simple RoboCup defender code, which would try to block the ball from going into the goal. Visual results of planning an attacker strategy can be seen in figures 4 and 5.

To evaluate the qualitative performance of our planning approach to traditional control methods, we ran samples of a linear execution of a deterministic version of tactics and skills. We then compared the success rate (trials resulting in a goal) of the linear execution with our approach using different maximum

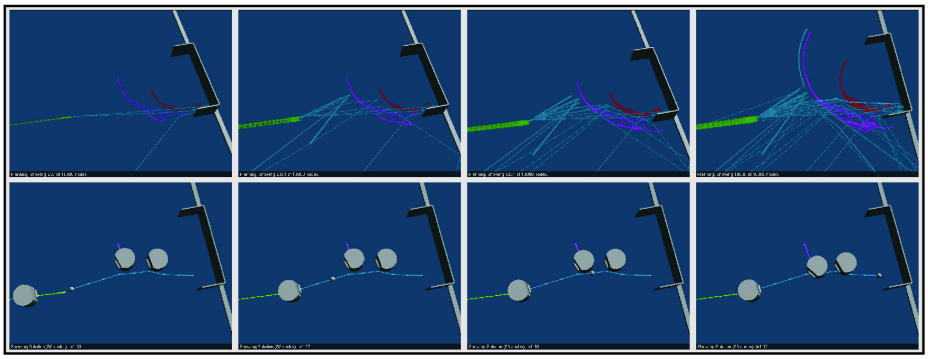

Fig. 5. An example of planning an attacker-tactic which has been limited to flat kicks only. The top row shows the growth of the Behavioral Kinodynamic RRT through the domain (robots and ball are not displayed). Blue nodes represent the position of the ball. Green, purple, and red represent positions of attacker, defender, and goalie respectively. The bottom row highlights the selected solution and shows the attacker executing it.

search tree sizes. The results are shown in figure 6. We can see that with growing search size, the planner also ends up finding successful solutions more frequently. In fact, even when limited to relatively small search tree sizes (such as 2500 nodes), we obtain a greater average success rate than linear executions.

Planning however, does come with the trade-off of computational time. A timing analysis of our planner can be seen in figure 7. The total time spent on physics and tactics computations scales linearly with tree size, which is to be expected because they are constant time operations applied per node. For larger tree sizes, the significant slowdown is the RRT nearest neighbor lookup which scales exponentially as the tree size increases. However, for smaller trees, such as 5000 nodes, the physics engine is still the major bottleneck. This is good news in a way, as processing power will undoubtedly increase and dedicated physics acceleration hardware is becoming a commodity.

Our second experiment was designed to emphasize the creative power of our approach. We created an opponent model which attempts to steal the ball from our robot, whereas our robot attempts to out-dribble the opponent and score a goal. The tactic used a state machine similar to the one outlined in figure 3. Note, that all of the ball-manipulation skills within this state machine contain free sampling-based planning variables to increase the variety of created solutions. It is also interesting to note that some of the FSM transition probabilities actually depend on the current state x_{source}. Some visual results from this experiment can be seen in figure 8. Note that additional videos are available at `http://www.cs.cmu.edu/~szickler/papers/robocup2008/`.

Method	Tree Size	% Success
Linear Tactic	n/a	30%
BK-RRT	1000	25%
BK-RRT	2500	40%
BK-RRT	5000	50%
BK-RRT	10000	65%

Fig. 6. Average success rate of a simulated "attacker vs. two defenders" scenario. Each row was computed using 20 randomly initialized trials.

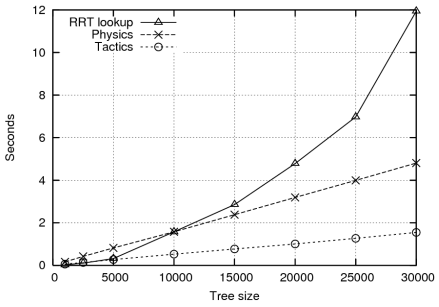

Fig. 7. A performance analysis of the different planning components throughout growing tree sizes.

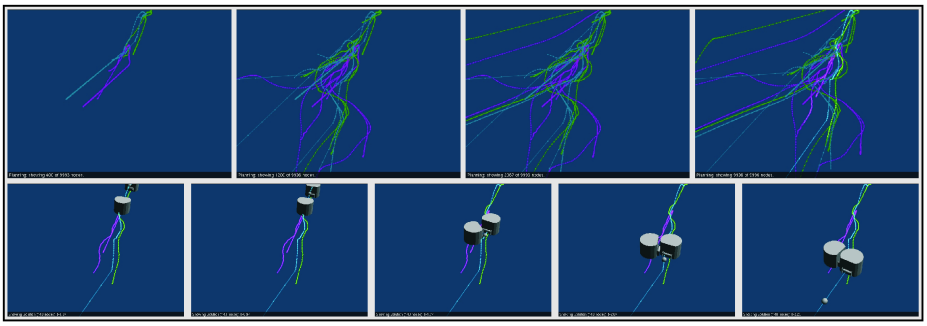

Fig. 8. An example of planning a tactic to out-dribble an opponent and score a goal. The top row shows the growth of the Behavioral Kinodynamic RRT through the domain (robots and ball are not displayed). Blue nodes represent the position of the ball. Green and purple nodes represent the positions of the attacker and opponent respectively. The bottom row highlights the selected solution and shows the attacker executing it.

4 Conclusion and Future Work

We presented a randomized behavioral kinodynamic planning framework for robot control generation. In particular, we introduced the Behavioral Kinodynamic RRT algorithm and demonstrated how it can be used to effectively search the dynamics space of a robot-soccer domain. We demonstrated that our planning-based approach is able to execute attacker-tactics with a greater success rate than linear approaches, due to its ability to sample from different possible control-sequences.

One obvious goal of future work will be to apply this approach to real robotic hardware, bringing along the challenges of model accuracy and computational performance. Another aspect that should be addressed in the future is the ability of re-planning. As the environment tends to change very quickly in a domain such as the RoboCup Small Size league, it might be a better strategy to generate

shorter, but more densely sampled plans, instead of predicting the environment too far into the future. Furthermore, the question how to re-use the previously computed result during re-planning needs to be answered. Finally, it would also be interesting to see how to integrate multi-agent control processes into the planning framework. Currently, the presented framework computes the control strategy for each robot independently, possibly using serial prioritized planning, which can be globally non-optimal. Allowing the planner to devise a joint control policy for sets of robots will be an interesting research problem, as it increases the dimensionality of the planning problem significantly.

References

1. 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
2. Parker, L.: ALLIANCE: an architecture for fault tolerant multirobot cooperation. Robotics and Automation, IEEE Transactions on **14**(2) (1998) 220–240
3. Behnke, S., Rojas, R.: A Hierarchy of Reactive Behaviors Handles Complexity. Balancing Reactivity and Social Deliberation in Multi-Agent Systems: From Robocup to Real-World Applications (2001)
4. D'Andrea, R.: The Cornell RoboCup Robot Soccer Team: 1999-2003. New York, NY: Birkhauser Boston, Inc, 2005. (2005) 793–804
5. Laue, T., Rofer, T.: A Behavior Architecture for Autonomous Mobile Robots Based on Potential Fields. 8th International Workshop on RoboCup (2004)
6. Bernadine Dias, M., Zlot, R., Kalra, N., Stentz, A.: Market-based multirobot coordination: A survey and analysis. Proceedings of the IEEE **94**(7) (2006) 1257–1270
7. LaValle, S.: Rapidly-exploring random trees: A new tool for path planning. Computer Science Dept, Iowa State University, Tech. Rep. TR (1998) 98–11
8. Kuffner Jr, J., LaValle, S.: RRT-connect: An efficient approach to single-query path planning. Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on **2** (2000)
9. Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation. In: Proceedings of IROS-2002, Switzerland (October 2002)
10. James Bruce and Manuela Veloso: Safe Multi-Robot Navigation within Dynamics Constraints. Proceedings of the IEEE, Special Issue on Multi-Robot Systems (2006)
11. LaValle, S., Kuffner Jr, J.: Randomized Kinodynamic Planning. The International Journal of Robotics Research **20**(5) (2001) 378
12. Baraff, D.: Physically Based Modeling: Rigid Body Simulation. SIGGRAPH Course Notes, ACM SIGGRAPH (2001)
13. Donald, B., Xavier, P., Canny, J., Reif, J.: Kinodynamic motion planning. Journal of the ACM (JACM) **40**(5) (1993) 1048–1066