Learning to Move Autonomously in a Hostile World

Leslie Ikemoto, Okan Arikan, David Forsyth*
University of California, Berkeley

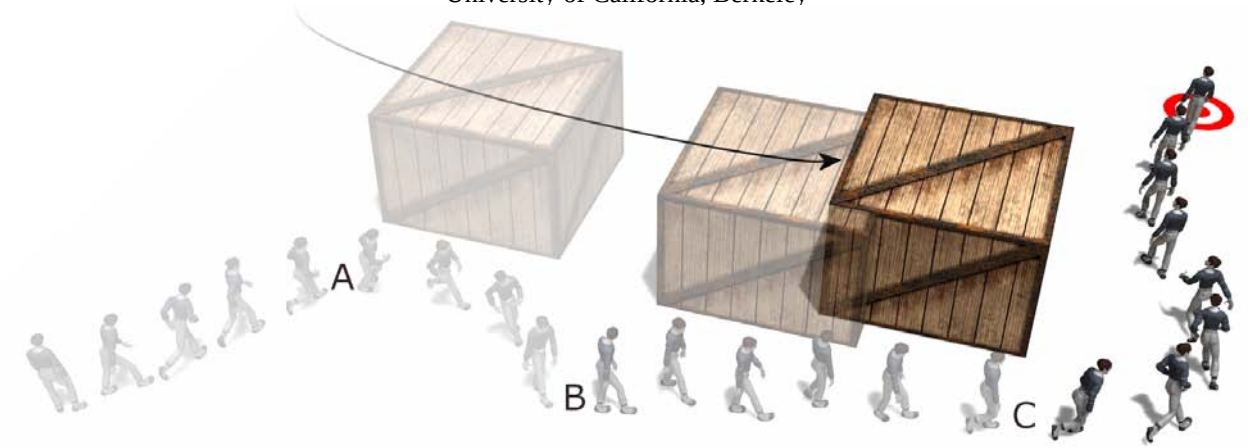

Figure 1: *This sequence was recorded from a live, interactive demo, in which the user controls the crate and can move it anywhere at anytime. A virtual agent is tasked with traveling from the left of the scene to the target on the right. While the agent is running towards the target, the user moves the crate into the position shown on the left, blocking the agent's intended path. The local motion planner selects frames that avoid hitting the object but still make progress toward the goal (A). The user again moves the object into the agent's path, and again the system successfully copes (B). Altogether, the user tries to block the agent by moving the crate 3 times, but the agent still dodges it and arrives at the target position seamlessly.*

As virtual worlds become richer and more complex, the agents that populate them face increasingly complicated control decisions. They must interact with the environment in a natural way and accomplish often complex objectives. This sketch describes a framework for controlling autonomous agents online in real-time.

In many current computer games, each agent is given a path to follow by a global path planner. The path planner computes an optimal (or near optimal) path through the environment, considering obstacles and the agent's objectives. The agent follows this path plan exactly, playing back pre-recorded motion sequences. For example, he may play back a motion captured walk cycle translated appropriately to follow the path plan. Unfortunately, forcing the agent to follow a path exactly results in awkward, discontinuous motion, because the path planner cannot ensure smoothness of the agent's motion. Considering smoothness makes the path plan calculation intractable. As the length of the motion sequence needed to meet the path plan increases, there is a combinatorial explosion in the number of possible sequences one can create from a collection of motion frames.

Our solution is to endow the agent with a *local motion planner*. The local motion planner is given the position and velocity of nearby objects and a path segment from the global path planner. At each frame interval, it must choose the next frame of motion to display. Sometimes, considerations of motion continuity make the choice unique; on other occasions, there are several frames to choose from. The local planner must choose a frame with the intention of producing a smooth motion that is one second long, lies close to the demanded path segment, and does not bump into any objects, even though some objects may move in ways that are difficult to predict. There are three major advantages to allowing the local planner to deviate slightly from the global path plan.

The first is that our system can handle dynamically changing environments, in which the agent has no knowledge of where objects will be in the future. It is expensive to update a global path plan, because the update considers all objects in the environment. In contrast, the local planner need only consider the objects immediately surrounding the agent, since these objects are (most likely) the only ones that can affect the agent's next second of motion.

The second advantage is that we can use a rough global path plan consisting of waypoints joined linearly. Linear path plans are usually faster and simpler to compute than smooth ones.

The third and most important advantage is that the motion sequence our system outputs is smooth, because the local motion planner knows which frames join continuously to the current frame. The local planner is not forced into using frames that join badly. The agent's frames are taken from a motion graph ([Kovar et al. 2002]), which encodes smooth transitions between motion clips.

From an implementation standpoint, the local motion planner's function is to pick which transitions in the motion graph to follow. Every time the local motion planner reaches a frame in the motion graph with multiple transitions, it chooses which transition leads the agent to best follow the global path plan and achieve objectives. In our system, the agent's objectives are to reach his target position while avoiding mobile objects and enemies.

One of the problems of using a local motion planner is that it suffers from the *short-horizon* problem inherent to local search. The local planner cannot account for the long-term consequences of choosing particular transitions in the motion graph. For example, a particular transition may lead the agent to follow the global path plan well, but every transition subsequently available from it may lead him to follow the plan poorly.

We use *reinforcement learning* techniques to estimate the long-term effects of decisions. We pre-compute the expected goodness of transitions in the motion graph, and use these estimates to adjust the local motion planner.

We demonstrate that it is possible to learn a simple yet effective local motion planner for a motion graph. As the accompanying video shows, we have created a real-time, online system that can adapt to a rapidly changing environment and drive the agent to achieve his goals.

References

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 473–482.

*e-mail: lesliei, okan, daf@eecs.berkeley.edu