Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2004)
R. Boulic, D. K. Pai (Editors)

Autonomous Behaviors for Interactive Vehicle Animations

Jared Go† Thuc Vu‡ James J. Kuffner§

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. USA.

Abstract

We present a method for synthesizing animations of autonomous space, water, and land-based vehicles in games or other interactive simulations. Controlling the motion of such vehicles to achieve a desirable behavior is difficult due to the constraints imposed by the system dynamics. We combine real-time path planning and a simplified physics model to automatically compute control actions to drive a vehicle from an input state to desirable output states based on a behavior cost function. Both offline trajectory preprocessing and online search are used to build an animation framework suitable for interactive vehicle simulations. We demonstrate synthesized animations of spacecraft performing a variety of autonomous behaviors, including Seek*,* Pursue*,* Avoid*,* Avoid Collision*, and* Flee*. We also explore several enhancements to the basic planning algorithm and examine the resulting tradeoffs in runtime performance and quality of the generated motion.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Interactive Animation

Keywords: Vehicle Motion, Path Planning, Steering Behaviors, Real-Time Animation, Online Search

1. Introduction

Realistically animating the behavior of autonomous vehicles in interactive simulations presents a challenging research problem with strong connections to research in the fields of robotics, computer graphics, and artificial intelligence. In particular, dramatically improved realism in the rendering of computer games during recent years has raised the level of expectation for synthesized animations. Reynolds has previously proposed several steering behaviors for controlling the motion of autonomous creatures in virtual environments [Rey99]. These behaviors produce compelling motions for autonomous entities and are generalized improvements over the original BOIDS model [Rey87]. However, it is difficult to directly apply them to animate vehicles such as fighter planes or spacecraft whose system dynamics typically involve complicated control models and sustained high velocities.

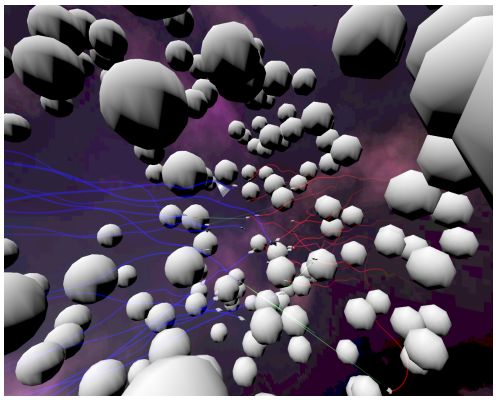

Figure 1: *Generated example spacecraft trajectories in a field of moving asteroid obstacles.*

Our research focuses on extending the steering behavior control model and combining it with online path planning techniques to yield more visually realistic synthesized vehicle animations. Our path planning framework is suitable for controlling the motion of not only space, water, and land-based vehicles, but potentially birds, bicycles, or a variety of simulated entities with significant dynamics. We utilize a physically based model of the available controls, offline

† jgo@cmu.edu
‡ tdv@andrew.cmu.edu
§ kuffner@cs.cmu.edu

trajectory preprocessing, and efficient online search to generate motion trajectories. Behaviors can be intuitively designed for planning and coordinating the motions of multiple autonomous entities. We applied this framework to the problem of generating motions for spacecraft and missiles, which were implemented and evaluated in an interactive game "Aeternalis". We also explored several enhancements to the real-time path planning algorithm and analyzed their effects on runtime performance and motion quality.

2. Background and Related Work

Our framework draws from research in both path planning and real-time local steering algorithms. The goal is to interactively generate high-quality paths for interactive dynamic environments in real-time applications. We adopted a hybrid approach to path planning, combining online search with the notion of steering behavior functions as the guiding heuristics of the search algorithm.

Offline path planning algorithms often emphasize accuracy and correctness at the cost of speed, and may take into account full dynamics and detailed robot control models when planning paths. With full dynamics, non-holonomic constraints can be simulated with high fidelity. However, for application domains such as interactive games, high accuracy is typically secondary to the speed of the algorithm. In these domains, the rapid generation of interesting and visually plausible motion is required.

One popular path planning method which has been used to plan motions for dynamic systems is the rapidly-exploring random tree (RRT) [LaV98, LK99]. RRT-based methods have also been used for real-time robot replanning systems as described in [BV02]. The emphasis is on generating goal-oriented paths for movement planning in both static and dynamic environments. In this paper, we consider the joint problem of generating paths for many types of behaviors, including those that are not goal-oriented, as well as generating animation trajectories which are visually pleasing.

In contrast to path planning algorithms which make use of global information, reactive steering behaviors provide a very different approach that uses local information [Rey99]. Offshoots of the original BOIDS model [Rey87] are ubiquitous in graphics, and similar methods using potential fields have been devised to modeling the actual flocking of animals [VSH*99]. Various heuristics can be defined to generate complex group behaviors such as flocking and following. The steering behavior approach uses a time-local approximation of steering forces which depend on the desired behaviors. Steering forces are typically very efficient to compute relative to global path planning. However, steering behavior methods are susceptible to becoming trapped in local minimum or other infinite behavior loops because they consider only local information.

The problem of applying group behaviors to systems with significant dynamics has been investigated by Brogan and Hodgins [BH97]. They consider controlling herds of hopping robots and bicycles [BH02]. Important issues considered include multi-agent control and group formations, where the perceptive limitations of agents influence the outcome of the simulation. These methods are still fundamentally local techniques. In our approach, global path planning occurs on top of a simulation that directly takes into account the system dynamics and obstacle motion, making the system much less prone to becoming trapped in local loops.

In the robotics literature, trim trajectories for helicopter control and the notion of composing them to form obstacle-avoiding paths has been investigated by Frazzoli [Fra01, FDF02]. This work uses a very detailed model of the control and helicopter dynamics, which is above the level required for interactive games. In addition, this work focuses primarily on goal-oriented motion, and does not provide a mechanism for clearly expressing varied types of behaviors. Bayazit, et. Al. use environment preprocessing to build a global roadmap of subgoals that can be combined with local behavior rules [BLA02]. In [LJC01], simulated crowds with leaders and followers were animated using decoupled planning and following behaviors. This research is most closely related to our work in that it combines global planning with reactive behaviors. These methods have not yet been applied to systems with significant dynamics.

Another closely related branch of research relates to Markov Decision Problems [GRD96]. In particular, the continuous and partially-observable versions of the general problem (CMDP, POMDP) can be used as one formulation of the planning problem. Extensive research aimed at solving MDPs efficiently has been developed [RP02, Sze01]. From this perspective, given the payoffs of choosing particular paths from a continuum defined by the possible control model inputs, solving for the optimal policy corresponds exactly to choosing the right sequence of control inputs that generates the best path. We explore in greater detail the links between our path planning problem and the associated techniques for solving MDPs.

3. Path Planning Framework

The path planning framework described in this paper is composed of several key components: (1) the steering behavior interface, (2) a visually-plausible control model, (3) pre-integration of the dynamics to enable real-time performance, and (4) a search algorithm designed to operate with limited time and information. Each of these components will be described in greater detail in the following sections.

3.1. Steering Behavior Interface

Steering behaviors, as described in the original paper by Reynolds [Rey99], present an intuitive method of controlling the motion of virtual agents. Each steering behavior is

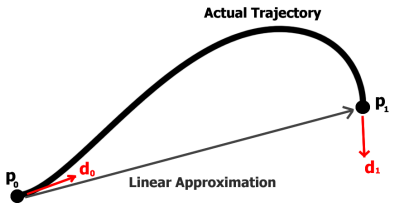

Figure 2: *Steering behavior parameterized model.*

denoted by the type of motion it produces, such as *Seek*, or *Wander*, and several can be combined simultaneously to produce highly complex behaviors. Formally, each steering behavior can be thought of as a mapping from world-state space to desired-velocity space. The steering behavior control algorithm then makes the proper adjustments so that the vehicle achieves the desired velocity.

One interesting property of steering behaviors is that they provide a universal method in which many different types of behaviors can be expressed. Regardless of the input, the final output of a steering behavior is simply a vector of desired velocity. Thus, behaviors do not need to be goal-oriented and can easily support motions such as wandering. The uniformity of output is also useful as it allows for the mathematical combination of steering behaviors. Steering behaviors also reduce the interdependence between behaviors and an entity's control model, as the mapping between desired-velocity and controls is orthogonal to the actual steering behavior functions.

Despite the numerous advantages, there are also drawbacks to using steering behaviors. First, the mathematical combination of different steering behaviors only has meaning in some cases, and thus combining behaviors often requires parameter tweaking by the designer. Furthermore, steering behaviors rely on highly-local approximations in both time and space and thus are poorly suited for non-holonomic vehicles in highly constrained environments.

In our framework, we make several modifications to steering behaviors in order to make them suitable for path planning. Under these modifications, we have implemented several steering behaviors from the Reynolds paper, including *Seek*, *Pursue*, *Avoid*, *Collision Avoidance*, and *Flee* [Rey99]. The primary modification to the original algorithm involves the input and output of the steering behavior functions. In our model, a steering behavior takes as input a path and returns a goodness value associated with the given path. Each path represents a possible trajectory and is composed of several parameters: a time offset which represents the starting time of the path relative to the present time; an initial and final position (which is used as a linear approximation of the true path), the final orientation given by a quaternion and the final facing direction given by a 3-vector. An example path and associated parameters is shown in Figure 2. The parameters for each path are passed as inputs to the steering behaviors. p_0 and p_1 are points in world space, and d_0 and d_1 specify the velocity of the craft at the beginning and end of the path. The full orientation quaternion is also stored for the beginning and end of the paths.

The goodness value returned as output from a steering behavior is a function of the input path and any auxiliary data required by the steering behavior. For example, a behavior which relates to collision detection may make use of a scene description with bounding spheres representing the objects in the scene, and execute ray-tests or do other geometric operations on the scene. A simple behavior such as seeking to a point may only take in a 3D point which represents the seek target. The steering behavior returns higher values for paths which are more desirable than others.

As an example, consider the *Seek* behavior. The auxiliary input to this behavior is a target point that the agent should seek towards, which is represented as a floating point 3-vector. If we denote the initial and final positions of the vehicle on the input path by p_0 and p_1, and the seek point by t, then the seek function can be expressed as:

$$g = \frac{1}{4}\left(1 + \frac{d_1}{\|d_1\|} \cdot \frac{t-p_1}{\|t-p_1\|}\right) * \left(1 + \frac{p_1-p_0}{\|p_1-p_0\|} \cdot \frac{t-p_1}{\|t-p_1\|}\right)$$

Note that paths where the vehicle faces towards the target are given higher values, and paths which contribute more to moving the ship closer to the target are also given higher values. The bias and scale is applied to ensure the function is positive and in the range $[0,1]$.

The *Flee*, *Pursue*, and *Avoid* behaviors are similarly implemented. The basic *Seek* and *Flee* behaviors simply seek towards or escape from a single point, and thus have no need for further information. The *Pursue* and *Avoid* behaviors take as input a target vehicle, and operate similarly but instead use a simple linear velocity prediction to determine the future position of the target at a given time in the future.

On the other hand, a behavior such as *Collision Avoidance* makes use of significantly more information about the scene. Our implementation of the function maintains a multi-resolution hash space which contains bounding spheres for the objects in the scene. These spheres are inflated by the radius of the steering vehicle, and input paths are evaluated by ray-test queries. A goodness of one is returned in the case of no collision, while a value of zero is returned in the case of a collision. This binary collision function has some benefits and drawbacks as compared to the original *Collision Avoidance* function, which will be considered in Section 5.

Our implementation of steering behaviors uses an object-oriented framework in which steering behaviors have a clearly defined interface and thus can be easily combined and composed in different ways. The *Composite* behavior represents a multiplicative combination of different steering behaviors. In our implementation, the designer can define the behavior of a vehicle with a few simple statements:

```
s1.Behavior = new Seek(0, 0, 100);
s2.Behavior = new Composite(
     new Pursue(s1), new AvoidCollision());
```

Considering the issue of numerical bounding of combined steering behaviors, we require that each steering behavior returns a value in the range $[0,1]$. The need for this restriction is discussed further in Section 3.3. An infinite value steering behavior $h(x) \mapsto [0,\infty)$ can be converted to a bounded form $f(x) \mapsto [0,1]$ by simply using a formula such as:

$$f(x) = 1 - \frac{1}{h(x)+1}$$

We do not automate the modification of weights in this framework, and give the designer full control over the weights and gains on different steering behaviors. Thus, it remains the responsibility of the designer to determine appropriate weights and gains depending on the desired overall behavior.

3.2. Control Model and Pre-Integration

The path planning framework places no constraints on the complexity or simplicity of the control model used. The only requirements for any control model are an input space which is parameterizable and the ability to forward integrate and generate vehicle trajectories under different control inputs. For complex control models, integration is expensive and unsuitable for online use. One solution is to attempt to pre-integrate a collection of representative trajectories offline for later use at runtime. The primary drawback is the potentially tremendous storage costs - a full dynamics control model involving rotational and linear velocity as well as variable thrust, pitch, and yaw controls would require immense storage in order to cover an adequate sampling of the space of available controls. Moreover, because rotational and linear velocity inputs may be unbounded, the difficulty of precomputing trajectories is exacerbated.

In this paper, we adopted a simplified control model which produces visually pleasing vehicle animations while remaining efficient enough to be stored compactly for real-time use. The state space of a controlled vehicle is given by the following parameters:

$$S = \{X, v, \theta, \dot{\theta}\}$$

X is a 3-vector which represents the current position of the vehicle in world space. v is a scalar, and denotes the forward velocity of the vehicle. The angular elements θ and $\dot{\theta}$ represent the orientation and angular velocity of the craft. The orientation is stored as a quaternion in all implementations, while the angular velocity is separated into yaw, pitch, and roll components depending on the domain and the set of available controls. For the 2D case, only the yaw angular velocity is used, while the 3D case requires at least both the yaw and pitch angular velocities. The roll angular velocity can be added to allow for the generation of more complex motions, but the resulting model requires more storage and increased running time. The model also assumes that the vehicle maintains constant forward velocity v in the direction that it instantaneously faces. In some cases, we restrict the ability to accelerate or decelerate. The range of angular velocities available to the craft along each rotational axis can be bounded by arbitrary limits in order to simulate vehicles of varying maneuverability.

Prior to runtime use, a fixed time interval is selected which determines the frequency of control input changes. This interval should be selected by the designer and depends on the complexity of the environment and the degree to which the steering behavior functions vary over time and space. In our example domain, we used a value of one second. During execution, the steering behavior selects a new set of control inputs after every interval. In our model of the vehicle, the control inputs are simply target yaw, pitch, and roll angular velocities. Once the appropriate target velocities are selected, the craft's actual angular velocity is linearly interpolated over the time interval from the angular velocity at the start of the interval to the selected values.

This simplified model differs from more realistic, yet computationally intensive models of vehicle dynamics. First, although the vehicle's angular acceleration is not explicitly bounded along any axis, the bounds on angular velocities and the time interval between selecting new angular velocities places implicit bounds on the vehicle's angular acceleration. Second, the simplified control model greatly reduces the number of necessary pre-integrated trim trajectories as compared to a full-blown dynamic simulation. In our case, because the velocity is forward-aligned, the trajectories are dynamics-state invariant and can be easily applied to any position and orientation of the craft via simple linear transforms. Finally, we note that the linear interpolation of angular velocities results in generated trajectories that are C_1 continuous but not C_2 continuous. However, the motions still appear smooth and pleasing to the human eye.

In our implementation, when pre-integrating the trim trajectories, we note that a different trajectory results for every pair of initial and final angular velocities. Thus, a model with yaw and pitch requires $O(n^4)$ storage, where n is the number of samples along each control input axis. For a uniform sampling of the control model, bounds and increments are selected for each axis and the ship is forward integrated to determine the resulting set of trajectory samples. For each trajectory, we store only the relative ending position, facing vector, and orientation quaternion as if the ship were initially placed at the origin with the identity rotation. Stored as single-precision floating point values, this requires $12+12+16=40$ bytes per path. The control models that we experimented with contained between 300 and 80000 paths, resulting in modest precomputation sizes ranging from 12 KB to 3.2 MB. The precomputation can be reused for all vehicles using the same control model, as long as the angu-

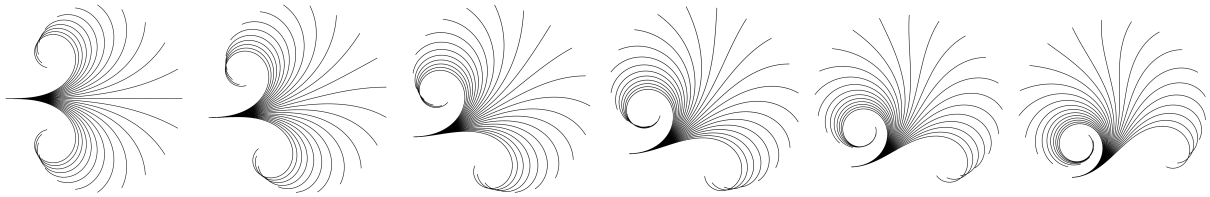

Figure 3: *Trajectory traces for an autonomous animated spacecraft in 2D with only yaw control inputs. From left to right, trim trajectories with initial yaw velocity 0.0 to 6.0 in increments of 1.0. In each image the paths result from choosing different end angular yaw velocities of -12.0 to 12.0 in increments of 1.0 radians per second.*

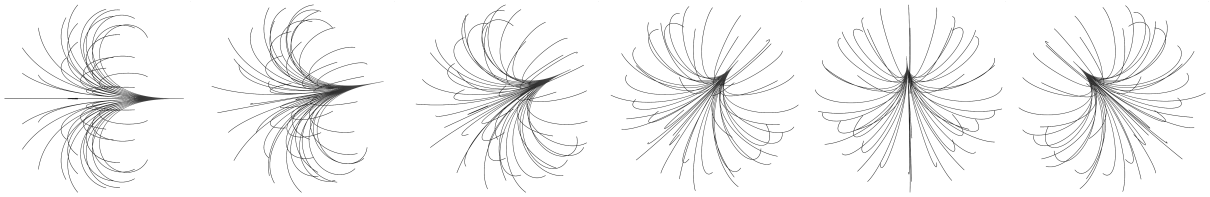

Figure 4: *Trajectory traces for a spacecraft in 3D, with yaw and pitch control inputs. From left to right, the camera rotates around the trajectory traces for clarity. In each image the paths result from choosing different end angular pitch and yaw velocities of -6.0 to 6.0 in increments of 2.0 radians per second.*

lar velocity bounds for those vehicles are less than or equal to the bounds used in precomputation. Each vehicle can also have a different velocity v, provided that it is constant during the vehicle motion. This is possible because differences in velocity simply result in trajectories that are scaled in length.

Symmetry can potentially reduce storage costs by reducing the number of initial-final state configurations which need to be precomputed, but we do not take advantage of these factors in our current implementation. One difficulty is that the actual control model as well as the integration process determines whether or not symmetry is applicable along different control input axes.

An important issue that must be dealt with in any implementation is ensuring a high correlation between the offline precomputed trajectories and the actual online integrated trajectories of the vehicle. We use simple Euler integration with a fixed timestep in both cases, which ensures that our paths match exactly. Systems using variable simulation timesteps with low-order integration may choose to either warp or interpolate to ensure that the precomputation matches with the simulation.

3.3. Online Search

The online search component of our framework is responsible for selecting the sequence of control inputs to be used at the start of each time interval. Given the current state of the ship, the possible future states are generated using the precomputed trim trajectories. Ideally, the search would select a sequence of control inputs that maximizes the sum of the goodness returned by the steering behaviors for the infinite sequence of path segments generated by the selection of inputs.

While the original control input steering problem is continuous in both time and space, we approach the problem by sampling and discretizing, both of which convert the continuous decision problem into a discrete decision problem. As control inputs are only selected at regular intervals, time discretization is already inherent to our control model. Furthermore, as we uniformly sample the control input space, we also use this sampling as the basis for discretizing the set of possible actions. Note that this decision is not forced on the online search as it is still possible to use the preintegration to generate values for continuous inputs via interpolation. We discuss further effects of sampling in Section 5.

If we define a path by a sequence of discrete path segments $p_0, p_1, ...$, and a function g to be the value returned by the steering behaviors when considering some path segment, then the goodness of any path can be approximated by:

$$\sum_{k=0}^{\infty} g(p_k)$$

For the purposes of planning, the path sequences are infinite and thus to obtain a reasonable comparison of infinite-valued paths, we must apply a discount factor to future goodness values. This practice is standard when solving for the optimal policy in Markov decision problems. The modified eval-

uation formula for a path is:

$$\sum_{k=0}^{\infty} g(p_k) \cdot d^k$$

In this case, d is a discount factor that is in the range $(0,1)$. If a bound is placed on goodness values, then we can naturally place a bound on the sum. In our case, each steering behavior returns a value in $[0,1]$ and the result is combined by multiplication. Thus, each term is bounded, and for some path p of length l we can bound the total value of all paths beginning with p by:

$$\sum_{k=0}^{l-1} g(p_k) \cdot d^k \pm \frac{d^l}{1-d}$$

In our search, we select a depth that corresponds to a particular maximum error in the steering behaviors and simply use the sum component of the formula. This selected depth acts as a *limited horizon* in the search. In practice, due to the breakdown of local approximations as we search further into the future, adopting limits of about 15 to 20 states worked reasonably well in our implementation. Further discussion of the effects of local approximations on the search is given in Section 5.

The search strategy we implemented is based on a form of greedy search or *best-first search*. A new search is conducted from the current state of the ship at the beginning of each time interval when the new control inputs must be selected. The pseudocode for the search is given in Algorithm 1. The current state of the ship is denoted by s_0 and is assigned a depth of 0. The search space consists of two sets: V, the set of visited states; and E, the set of expanded states. A state s is considered visited once the path from s_0 to s has been evaluated. The state is considered expanded once the control model has been sampled at s and trajectories leading to successor states have been generated. V initially contains only s_0 while E begins as an empty set.

During the search routine, we repeatedly expand the state with the best evaluation among the states visited but not yet expanded until we have expanded all the visited states or have run out of time. The process of expanding a state into successors is accelerated by using the offline preintegration described in Section 3.2. The end position for each path can be computed with a matrix transformation. All or only a portion of the precomputed paths are evaluated to future states, depending on the specific search algorithm in use.

For each state that we visit, we compute a goodness metric using the path between the state and its parent and pass it into the steering behavior evaluation function. The resulting value is then multiplied by the discount factor as mentioned above, and added to the parent's value in order to obtain the goodness of the path. When the node is inserted into the expanded set E, the value used is this value with an added optimism factor f in the range of $[-1,1]$ multiplied by the remaining maximum bounds. Thus, if we let k be the depth

Algorithm 1: Pseudocode for online search.

```
V - set of visited states;
E - set of expanded states;
V = {s_0};
E = { };
depth(s_0) ← 0;
t ← 0;
while t < t_max or V = E do
    s_m = argmax_{s_i ∈ V−E} {evaluation of s_i};
    if depth (s_m) ≤ D_max then
        S_succ ={s_i | s_i = apply (control_i, s_m)};
        depth (s_i) = depth (s_m) + 1 ∀s_i ∈ S;
    end
    E = E + {s_m};
    V = V + S_succ;
end
s_best = argmax_{s_i ∈ V} {evaluation of s_i};
path = Retrace: control sequences s_0 → s_best;
return path;
```

of a state s, and s' be the parent of s, then the value $v(s)$ and effective value $ev(s)$ of the state s are given by:

$$\begin{aligned} v(s) &= v(s') + g(p_{s \to s'}) * d^{k-1} \\ ev(s) &= v(s) + \frac{d^k}{1-d} * f \end{aligned}$$

This formulation allows paths to be compared on the basis of the infinite paths that go through the finitely explored path segments. Thus, it is possible to compare paths of varying length in a common format because of the bounding on possible path values. The optimism factor f represents an estimation of the average steering behavior values for paths as they grow to infinity. Although the effects of the optimism factor on the search have not been fully explored, lower optimism factors tend to cause the tree to be biased towards searching deeper, while larger optimism factors tend to bias exploration among shallower nodes.

Once the search has terminated, we find the leaf state s_{best} which has the highest evaluation among the visited states. We can then trace back to s_0 to find the sequence of paths and control inputs to get from s_0 to s_{best}. The first control input in the sequence is returned as the selected input, and the current implementation discards the other inputs as they can potentially be invalidated by the time the vehicle arrives at the end state and initiates the subsequent search.

The selection of s_{best} is not trivial, and different ways of selecting the best state can yield different search outcomes. One restriction that we place on the selection is that we only consider the leaf states with depth greater than or equal to some minimum value. This restriction helps to avoid the situation in which a shallow state is selected on the basis that it has a fairly high value but it has not been extensively ex-

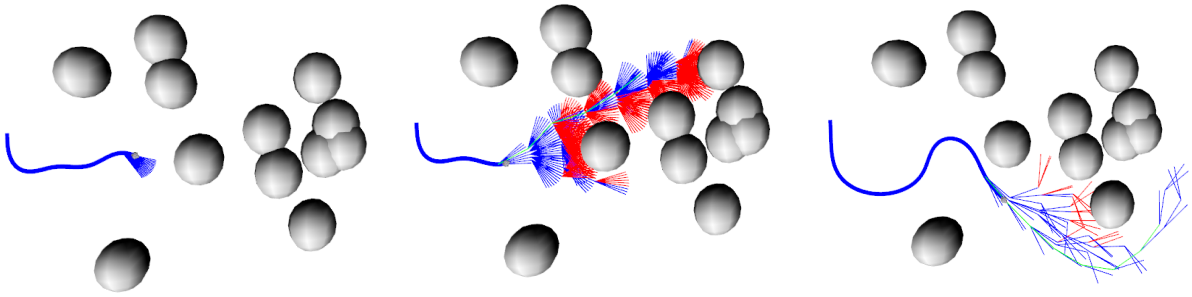

Figure 5: *Example search patterns for single-step, uniform sampling, and adaptive sampling. In the single-step case* (left)*, a small fan represents the trajectories being considered. In the uniform sampling case* (center)*, the trajectory candidate tree is extremely dense. The adaptive sampling case* (right) *represents a tradeoff between performance and path quality.*

panded yet so the future paths are unknown. Depending upon the severity of the nonholonomic constraints, if such a path were to be selected, a future collision could be unavoidable due to insufficient time and free space to escape. Selecting a state with at least some minimum depth means that there is some guarantee of goodness up to some time in the future.

Using the described search framework, we implemented three different search algorithms which differ in the number and the maximum depth of states expanded. The search strategies also differ in the method used to select the successor states for each state that is being expanded. Figure 5 shows snapshots of the various search trees at runtime.

Single-Step Search: This type of search is used only for comparison with the other methods. In this case, we only expand the current state of the ship and visit all of its successors. Thus the number of states expanded and the maximum depth of the search tree are equal to one. This search is the most computationally inexpensive but suffers dramatically in terms of generated path quality, much like local steering methods.

Uniform Sampling Search: This strategy only visits a portion of the successor states for each expansion through regular sampling of the control actions. Let k_{skip} be the number of control actions that will be skipped between samples, and N_{cr} be the total number of control actions. The number of states visited for every expansion is thus:

$$N_v = \frac{N_{cr}}{k_{skip}}$$

The maximum depth for a search state is restricted to a finite number D_{max}. Thus the total number of states visited for every search is bounded the sum of all possible visited states at each depth d:

$$\sum_{d=0}^{D_{max}} (N_v)^d = \frac{(N_v)^{D_{max}+1} - 1}{N_v - 1}$$

Even with a rudimentary control model, the branching factor is enormous with this type of search. The skip factor helps to make this algorithm somewhat tractable.

Adaptive Sampling Search: The Adaptive Sampling search algorithm is a variant of the Uniform Sampling algorithm that reduces the states considered while maintaining reasonably high quality paths. In this algorithm, instead of simply expanding all paths for every node regardless of depth, the number of paths expanded grows inversely proportional with the depth of the state under expansion.

The number of states to expand at a particular depth is given by a geometric sequence. In particular, the collapse factor c is a parameter in the range $(0,1)$. At each state of depth d, the number of paths that should be expanded is given by $N_v * c^d$. The general effect of adaptive sampling is that the branches of the search tree grow more sparse as the depth increases, significantly reducing the total number of states being searched. The total number of states visited is given by:

$$\sum_{d=0}^{D_{max}} (N_v)^d \cdot c^{T_d}$$

where T_d represents the d-th triangular number. The growth rate in this case is asymptotically smaller than that of Uniform Sampling search. As we only consider a subset of the paths at each node, we select paths using a Monte Carlo procedure. A circular permutation array is generated at the start of execution, and paths are looked up by indexing the permutation array to find the true index of the path to sample.

4. Results

In order to determine the effectiveness of various online path planning strategies, we conducted numerous experiments under various conditions representative of typical usage scenarios. The first test measured the quality of paths generated by each search algorithm across a range of different environments. The numerical results are given in Figure 6, and

each environment shown in the figures is described in detail in subsequent paragraphs. For each configuration, the various algorithms were run sixty times with randomized initial obstacle positions. The performance of a run was measured as the average value of the goodness of each path segment selected during the run.

The 2D seek configuration consists of a distribution of asteroid obstacles within a rectangular region of space, with a single vehicle placed to one side of the asteroid field. A *Seek* steering behavior is attached to the vehicle, with the seek target being a point on the far side of the field. In addition, a *Collision Avoidance* behavior is attached to encourage the ship to steer around the asteroids. The 3D case is similar, but the asteroids are contained within a rectangular volume and the control model involves both pitch and yaw. In terms of path quality, the single-step algorithm performed poorly and the uniform search performed best due to its high sampling density. The adaptive search performed only moderately worse than the uniform search. Several of the test configurations are illustrated in Figure 8.

The 2D dynamic pursuit configuration involves a moving asteroid field. The asteroids are spaced further apart and are given an initial random linear velocity. In this case, multiple ships were simulated, one seeking to the far side of the field as before, while five others pursue the first. The seeker is denoted in the data table by S while the goodness of the pursuers' paths were averaged together and are listed as P. The generated paths are visually smooth and take the motion of the asteroids into account during the search. Some frames from this animation are shown in Figure 9.

The 3D pursuit configuration is similar, but only has one pursuer. The asteroids also do not move, but the density in this case is roughly five times that used for the 3D seek configuration. Example images from the 3D scenario animations are shown in Figure 8. In the 3D meet configuration, two ships start on opposite sides of a field and both pursue each other. The typical resulting trajectories in this case involve the two ships meeting somewhere inside the field and orbiting around each other. The two ships are denoted by A and B in the data table.

Averaging over all runs and configurations, the single-step algorithm generates very poor paths when compared to uniform and adaptive search. The difference between the results obtained by uniform and adaptive search is rather small, indicating that the adaptive algorithm is still able to generate high quality paths while searching far fewer states than the uniform algorithm.

In order to examine the speed of the algorithms, a single ship was considered in the 2D and 3D seek configurations. The control model in the 3D case has 121 sampled control inputs at each state as opposed to 17 inputs for the 2D case. Figure 7 summarizes the average performance of the various search strategies in the two environments. While the complexity of the control model alone increases by a factor of 7

	Single-Step	Uniform	Adaptive
2D Seek	0.400	0.880	0.827
3D Seek	0.704	0.959	0.922
2D Dyn Pursue(P)	0.748	0.845	0.876
2D Dyn Pursue(S)	0.905	0.913	0.862
3D Pursue(P)	0.804	0.880	0.820
3D Pursue(S)	0.789	0.792	0.929
3D Meet(A)	0.712	0.765	0.786
3D Meet(B)	0.663	0.699	0.703
Average	**0.715**	**0.841**	**0.840**

Figure 6: *Average goodness of selected paths for various search algorithms in different test configurations, and the average across all configurations.*

	Single-Step	Uniform	Adaptive
2D	2.48 ms	48.33 ms	16.8 ms
3D	24.17 ms	437.33 ms	68.0 ms
Ratio $(\frac{3D}{2D})$	9.74	9.04	4.07

Figure 7: *Average running times per search in the 2D and 3D asteroid environments.*

when going from 2D to 3D, we note that the adaptive sampling search grows by a smaller factor of 4.07. On the other hand, both the single-step and uniform search algorithms grow by a factor of 9, most likely due to the increased number of asteroids and thus requiring additional expensive intersection testing. The adaptive sampling algorithm requires 65% less running time than the uniform sampling algorithm in the 2D case, and the advantage grows to 84% when the control model becomes more complex.

Many other configurations were also experimented with, although the performance was not analyzed directly. One interesting scenario was a fleet battle between two groups of ships, with 15 ships on each side. Each ship was randomly assigned to pursue another ship on the opposing side. The simulation was able to run at interactive rates, and an example frame is shown in Figure 8.

5. Limitations and Comparison

One difference between our current implementation of collision avoidance and the *Obstacle Avoidance* steering behavior from [Rey99], is that our algorithm uses direct intersection testing instead of a local approximation strategy. The local approximation involves building a bounding box around

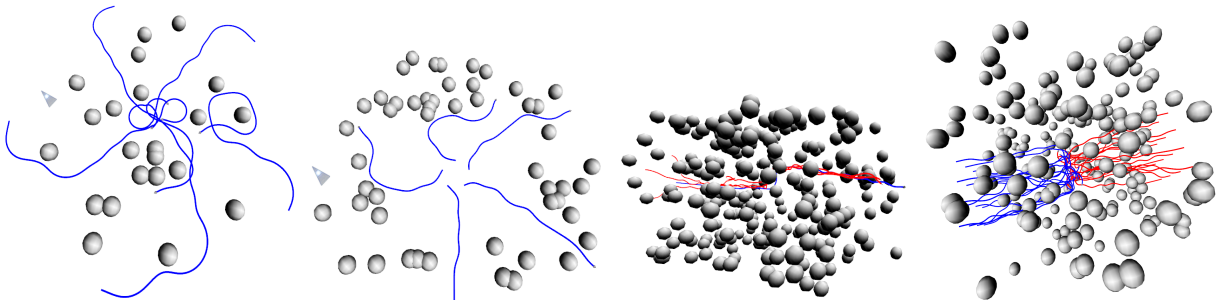

Figure 8: *From left to right, trajectory traces for: 2D seeking for multiple vehicles, 2D fleeing for multiple vehicles, 3D pursuit, and a large fleet combat simulation.*

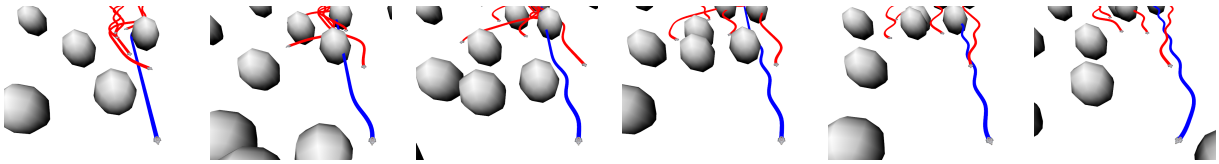

Figure 9: *Frames from the 2D dynamic pursuit scenario. The red trails are left by the pursuers, and the blue trail is left by the seeking ship. All ships interactively plan trajectories toward their target while simultaneously avoiding the moving asteroids.*

the vehicle and extending the box forward in the direction of the vehicle's current velocity, and adding a steering force depending on where the box intersects obstacles in the scene. This type of avoidance strategy produces smoother steering behavior output than the avoidance method described in this paper. The direct intersection testing implemented in this paper causes high-frequency steering behavior output due to the binary nature of the returned value (0 on collision, 1 on no collision). In the case of complex environments, however, we have observed that this allows for paths with improved trajectories. Environments such as those consisting of narrow corridors can also be difficult to navigate in the original steering behaviors implementation without a designer-specified path to follow.

Another limitation of our collision steering behavior is that it does not take into account the curvature of the actual selected path during intersection testing and instead simply tests intersections with a ray from the initial path position to the final path position. The curvature is instead taken into account by virtually expanding the radius of the objects in the scene; however, this expansion many not take into account the maximum possible path curvature.

The effect of discretization in both time and space on the quality of generated paths is also an important issue. One type of artifact observed involves the slight oscillation of the controlled vehicle during near straight-line trajectories. We term this artifact *path aliasing*, as it essentially corresponds to undersampling of the control model during discretization. Path aliasing occurs when the vehicle is unable to directly select the control inputs that would yield a perfectly straight trajectory, and instead must continuously select between the two nearest discrete values on either side of the ideal control input.

We reduce path aliasing through the use of several different techniques. The first strategy simply ensures that a straight line trajectory is always considered at every state. In many cases, this modification was enough to allow the oscillation to stabilize. In other situations, we implemented a damping factor on the steering behavior which gave a slight bias to paths that were straighter. Another potential solution would be to use a non-uniform sampling of the control space, biased towards paths nearer to straight line trajectories; however, this solution was not explored in the current implementation. In our experiments, the use of the first two solutions mentioned above practically eliminated path aliasing completely.

Finally, another important issue concerns the effect of using local approximations during searches of relatively high depth. It is important to keep in mind that while the search may be inaccurate at high depths, the short time interval between replanning compensates for the potential inaccuracies of individual searches. This issue is most important with mutually dependent steering behaviors. In our framework, we do not compute globally optimal paths for multiple vehicles with mutually dependent steering behaviors as the computational cost would be prohibitive. Instead, the motion of other vehicles is estimated using an approximate straight line velocity and trajectories are determined for each ship individually. In practice, these approximations appear to be sufficient

for generating satisfying animations of relatively complex group behaviors.

6. Discussion and Future Work

In this paper, we have presented a framework for combining steering behaviors with online path planning and demonstrated its ability to simulate complex group behaviors interactively. A modified steering behavior control interface is adopted that enables intuitive control over the trajectories of simulated vehicles as well as a high degree of flexibility in manipulating the effects of each steering behavior. We also formalize some restrictions on steering behavior values and examine the effects of these restrictions on our ability to search and solve the planning problem as a continuous Markov decision problem. The proposed framework also offers a simple control model that yields visual pleasing trajectories that is efficient both in terms of performance and storage requirements. We presented results from our implementation of this framework for simulated spacecraft in 2D and 3D planning domains and examined the performance of various search techniques and the quality of the generated paths.

For future work, we intend to examine several issues in greater detail. The first of these issues involves the tradeoffs between control space complexity and path planning frequency under different domains. We note that planning with real-time constraints prevents us from having both high control model complexity and high path planning frequency. We hypothesize that different domains have different optimal path planning frequency and control model complexity. These differences are due in part to the rate of change of the steering behaviors as well as the density of objects in the domain. We also intend to examine strategies for compressing the offline control model which would allow for higher sampling density and support more complex control models. Currently, the uncompressed storage grows at a rapid rate with respect to the sampling density and degrees of freedom of the control model, especially considering the smoothing process which makes each trim trajectory dependent on the starting and ending control inputs for the specified time interval. The ability to preintegrate for higher sampling densities and more complex control models would be very beneficial in this framework. Other issues we are examining include adaptive planning horizons and adaptive path selection. Implementing these techniques would allow agents to make more frequent decisions when in dense spaces, and conserve CPU cycles by reducing the frequency of planning when in low-constrained spaces.

We are also investigating methods for improving the online search algorithm. We are currently developing a statistical estimator that we can use to build bounds on the goodness of particular paths. This has the potential to considerably reduce the workload of the online search algorithm. In addition, we are also exploring the use of analytic algorithms that do not require a discrete sampling of the continuous state space. Such algorithms would significantly reduce path aliasing by allowing for the selection of precise control input parameters as opposed to selecting only from preexisting sample points in the control space.

References

[BH97] BROGAN D., HODGINS J.: Group behaviors for systems with significant dynamics. In *Autonomous Robots* (1997). 2

[BH02] BROGAN D., HODGINS J.: Simulation level of detail for multiagent control. In *Proc. Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS)* (2002), pp. 199–206. 2

[BLA02] BAYAZIT O. B., LIEN J.-M., AMATO N. M.: Better group behaviors using rule-based roadmaps. In *Proc. Int. Wkshp. on Alg. Found. of Rob. (WAFR)* (Nice, France, 2002). 2

[BV02] BRUCE J., VELOSO M.: Real-time randomized path planning for robot navigation. In *Proc. Int. Conf. Intelligent Robots and Systems (IROS)* (2002). 2

[FDF02] FRAZZOLI E., DAHLEH M., FERON E.: Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control, and Dynamics 25*, 1 (2002), 116–129. 2

[Fra01] FRAZZOLI E.: *Robust Hybrid Control for Autonomous Vehicle Motion Planning*. PhD thesis, MIT, 2001. 2

[GRD96] GILKS W., RICHARDSON S., D.SPIEGELHALTER: *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996. 2

[LaV98] LAVALLE S. M.: Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State Univ., Oct. 1998. 2

[LJC01] LI T., JENG J., CHANG S.: Simulating virtual human crowds with a leader-follower model. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA)* (2001). 2

[LK99] LAVALLE S., KUFFNER J.: Randomized kinodynamic planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'99)* (Detroit, MI, May 1999). 2

[Rey87] REYNOLDS C.: Flocks, herds, and schools: A distributed behavioral model. In *Siggraph* (1987). 1, 2

[Rey99] REYNOLDS C.: Steering behaviors for autonomous characters. In *Game Developers Conference 1999* (1999). 1, 2, 3, 8

[RP02] RATITCH B., PRECUP D.: Characterizing markov decision processes. In *Proc. ECML* (2002). 2

[Sze01] SZEPESVARI C.: Efficient approximate planning in continuous space markovian decision problems. *AI Communications 13*, 3 (2001), 163–176. 2

[VSH*99] VAUGHAN R., SUMPTER N., HENDERSON J., FROST A., CAMERON S.: Experiments in automatic flock control. In *Proc. RAS* (1999). 2