Using Atomic Actions to Control Snake Robot Locomotion

Joseph C. Koo
Email: jckoo@stanford.edu

I. Introduction

LOCOMOTION problems are generally difficult because of the large amount of coordination required to make sure that desired movements occur, while undesired actions (such as falling over) are avoided. Snake locomotion [1] is no exception to the rule. In this project, we show that we can simplify the control of a snake robot so that it is computationally tractable, but yet can still produce interesting locomotion strategies. Our particular application is to have a snake robot navigate through complex terrains in order to reach a goal of traversing the greatest distance possible. This scenario could arise, for example, in applications where a robot needs to look through the rubble field of a building collapse in order to search for survivors. In such a situation, well-known strategies such as sidewinding may not work because these strategies are highly specialized for particular terrains. In the case of sidewinding, for example, acceptable movement is produced only when the terrain is flat enough that the contact between the snake and the surface occurs at regular intervals along the snake length. Unfortunately, this is hard to come by on a rubble field.

The snake robot being used for this project (shown in Figure 1) consists of $L = 10$ body segments connected in a chain. Consecutive body segments are connected by pairs of revolute joints, giving 2 degrees of freedom at each link. Thus, there are a total of 18 degrees of freedom in the entire snake, making precise control of snake movement very difficult. For reference, we set up a coordinate frame such that the length of the snake is considered the x-direction, sideways from the snake is the y-direction, and up and down constitute the z-direction.

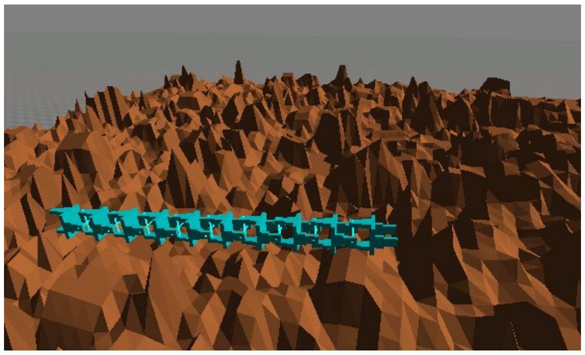

Fig. 1. Snake robot on a rubble field.

For the project, techniques and ideas were first run in simulation—in order to show proof of concept and to generate analyzable data—before being attempted on a physical robot. A model of a snake robot was built in the rigid-body dynamics simulator Open Dynamics Engine [2] for this purpose.

II. Atomic Actions

The first step in narrowing down the possible action space is to define a set of *atomic actions*. Each atomic action consists of a short sequence of commands for each of the joints in the snake, so that instead of carefully specifying a complete sequence of joint angles to move, one only needs to issue the command for a particular atomic action. This reduces the need for fine control of each angle while still allowing for fairly robust actions to take place. Then, when atomic actions are strung together in succession, the snake can perform complicated gaits with only a few quick commands.

The atomic actions approach requires that the actions be able to perform some useful function for the snake. We specify actions which are useful for lateral movement of the snake—so that we can accomplish a goal of maximimizing the lateral distance traveled. Some of the atomic actions were inspired by actions already proven to be able to lift a snake over steplike obstacles; these include the lifting of the head segment, then a follow-the-leader sequence to lift successive segments, and concluding with a tail swing up-and-over. Each of these components are then specified as an individual atomic action. Another inspiration for defining the set of atomic actions comes from snake sidewinding, where the snake is able to move sideways by simultaneously pushing on the ground and shifting its body position to make ground contact at a different location.

For this project, we created a set of $n = 35$ atomic actions, drawn from the activities mentioned. Most of these atomic actions are specified in terms of changes in angles, where the action gives the number of radians a joint should move from its current angle. There is also a *relaxation* atomic action, where the snake is told to move all its angles absolutely to a resting $0\,radians$ position, at a rate proportional to its current deviation from $0\,radians$.

III. Learning Good Sequences

Once the atomic actions are defined, the next major task is to determine good sequences of atomic actions to perform. This requires searching through possible sequences of actions in order to find something satisfactory.

A. Metrics and Reward Function

In order to perform good searching, we have to know what the goal of our search is. Thus, we must first devise an appropriate metric to measure the progress of our snake. For example, we could look at the total displacement of all of the snake bodies, or we could keep track of the smallest of the displacements that any of the bodies have moved from their starting position. We choose to include minimum displacement y_{min} in our metric, since it encourages all bodies to move from their initial positions, rather than rewarding a sequence of actions which can, for example, move the head very far while keeping the tail fixed at its starting position.

We also want to keep the snake fairly straight, if possible, as it moves across the terrain. This seems reasonable since the snake will be unable to create much movement when it is wound up tightly (unless the snake is curled together in a wheel and rolling around, but that particular configuration is undesirable for physical reasons). Thus, we introduce a pairwise angle cost $c_{pairwise}$ given by (for pitch angle θ_i and yaw angle α_i connecting segment $i-1$ with segment i)

$$c_{pairwise} = \sum_{i=1}^{L-1} (|\theta_i| + |\alpha_i|). \tag{1}$$

If $c_{pairwise}$ is small, then it means that all the joint angles are close to 0.

Combining these two criterion, we formulate the reward function which we wish to maximize as

$$R = y_{min} \cdot e^{(-|c_{pairwise} - c_{pairwise,des}|)}, \tag{2}$$

where $c_{pairwise,des}$ is a desired value corresponding to a fairly straight snake.

B. Search Techniques

1) Exhaustive Search: The first method tried was exhaustive search, where all possible sequences of depth d were enumerated, and the results of the simulations compared. We then chose the sequence which maximized our reward. Because of its exhaustiveness, we could guarantee that we had found the best sequence, but at great computational cost. For a sequence of d actions, the enumeration of $n^d = 35^d$ different branches was required, meaning that we could not expand past a level of about $d = 3$ within a reasonable amount of time.

2) Greedy Best First Search and Modified Greedy Best First Search: The opposite extreme in terms of search complexity is best first search. At each depth, by expanding only the node with the highest current reward, we can reduce the number of branches to search down to $nd = 35d$. However, it was noticed that if at each step in the sequence one always chose the myopic best next action, it would preclude choosing actions which might at first appear to be bad but actually lead to good actions later on. Thus, we modified the greedy best first search so that it searched in tiers. Toward this end, we specified tiers of depth $d_{tier} = 3$, performed exhaustive search on all sequences of actions of depth d_{tier}, and then chose the best sequence from this tier as our best action sequence. Then, using this found sequence as a node, we would expand another set of d_{tier} actions and find the best one out of those. In this way, we performed greedy best first search not on individual atomic actions, but on subsequences of depth d_{tier}. This increased the range of behaviors allowed by our actions, without having to resort to exhaustive search. For a search of depth d, modified greedy best first search enumerates $\frac{d}{d_{tier}} \cdot n^{d_{tier}}$ sequences, which can be much less than the n^d sequences of exhaustive search.

C. Speeding Up Search

1) Pruning Branches: In order to speed up our searches, we would like to avoid expanding nodes which seem like they might not lead to good configurations later on. If we can prune these branches early, then we can reduce the number of branches we will need to search through without affecting our final solution. Some simple heuristics we used initially for pruning branches are the following:

- If the snake has moved significantly in the direction opposite from our goal, we would eliminate that sequence of actions from our search space. It would be highly unlikely that an optimal sequence would require much backtracking, since the snake would then have to recover the lost distance before continuing on its path. (Of course, if there were a very large obstacle directly next to the snake, backtracking might be beneficial, but currently we are not considering such large obstacles. And in fact, our current method enables the snake robot to navigate over some fairly sizeable obstacles already.)
- As mentioned before, we imposed a desire for straightness into our reward function. This is because, for a snake that has coiled together, it will probably need to uncoil itself before being able to do anything else useful. Thus, we prune out sequences of actions which lead to the snake being tightly wrapped around itself.
- It was also discovered that there are particular joint configurations which were infeasible for the snake. For example, if both the pitch θ_i and yaw α_i angles are moved to their extreme limits (i.e., both angles near $90°$), then the brackets attached to these joints would collide (and in fact, intersect) with each other, which is unphysical. As a result, sequences which led to these configurations were also pruned.

2) Using SVM's: Classifying or ranking actions may offer insight into which actions are better than others. If we can reduce our set of actions to those which are predicted to lead to higher reward, we can prune off those branches which do not satisfy such a requirement. In the following section, we discuss how we use support vector machines [3] in order to learn better searching strategies.

IV. Support Vector Machines for Enhancing Search Performance

Support vector machines were implemented (using the software package SVM-Light [4]) in order to classify actions based on the current configuration of the snake and terrain. The purpose of using SVM's is that if we can classify an action to improve the reward function, then we would most likely want to expand our search tree in that direction. Using this information, we can utilize beam search to focus our search down the most promising paths.

A. Feature Selection

Our input feature set takes into account descriptions of the snake shape, its relative orientation, and its relationship to the terrain immediately surrounding it.

1) Snake Configuration: One set of features which is important in identifying the snake's state is its current configuration. That is, using the coordinate frame of one of the bodies (e.g., the tail body) as a reference frame, we find the position and rotation of all of the other bodies with respect to this reference frame, so that we obtain a full description of the shape of the entire snake. Position is described using Cartesian coordinates (3 dimensions) and rotation is written in quaternion form (4 dimensions) [5], so that for each body, we have 7 features. Thus, the snake configuration is described by a 70-dimensional feature vector.

2) Overall Snake Orientation: Because the snake is not always upright (i.e., with the up-side always up), we include in our feature vector elements which describe the rotation of the snake with respect to the world frame (i.e., the reference frame is the coordinate frame of the terrain). Again, rotation is represented in quaternion form, so there are 4 dimensions. By combining this information with the information on the snake configuration (described previously), we know the orientation of all of the bodies with respect to the world frame.

3) Local Terrain: Since we are traversing an irregular terrain, it is intuitive that we should also include some measure of our terrain knowledge within our feature set. That is, what happens to the snake depends a lot on what the nearby terrain looks like. We include the following terrain features:

- Height of each snake body above the terrain.
- Height of each snake body in relation to the terrain height one body width laterally forward (i.e., in +y direction).

Recalling that our goal is to maximize the lateral distance traveled (i.e., travel as far in the +y direction as possible), we would like to know the features laterally forward since we need to know what obstacles we will need to overcome soon. We choose not to look at terrain features too far away (i.e., no more than one body width surrounding the snake) because we are learning for limited lookahead, where each action does not propel the snake very far, so it is not necessary to know what the terrain looks like, say, 10 body widths away.

4) Normalization: The entries in our feature set differ widely both in their scales as well as their average variations. This ends up affecting our learning performance since we will be using linear kernels. To fix this, we normalize our data so that each entry has zero mean and unit variance. In a sense, then, the variation in each entry will be weighted equally. We can see the improvement in training performance due to normalization, by looking at the increase in the true positive rate between the first and second entries in Table I.

B. Target Values

The target values for our learning algorithm come from the reward function defined in (2). Thus, we would like to maximize the predicted lateral distance traveled, while keeping the pairwise angle cost from becoming too large. When using SVM classification, only the sign of the target values is utilized, so the learning algorithm will only take into account positive versus negative progress of the snake—without bothering about the correction term introduced by the pairwise angle cost.

TABLE I
COMPARISON OF TRAINING DATA PREPROCESSING METHODS

Training Set Size	Normalized?	True Positive Rate	False Positive Rate
1000 samples per action	No	0.67305	0.09854
1000 samples per action	Yes	0.81655	0.08621
Full training set	Yes	0.83682	0.09667

C. Training Data Collection

Training data for our SVM was obtained by running short (i.e., of depth $d = 6$) modified greedy best first searches on randomly-generated rubble fields. For each training run, data was collected on the current snake state and relation to terrain (i.e., the feature vector), the action which led to this state, and the action which is taken next. We also recorded the reward metric that arose from taking this next action.

We were able to reduce training time by taking random samples of the training set and only training over these samples. This was acceptable because the training data had low variance; training errors were similar for training on the entire set versus training on a pared-down training set, as can be seen in Table I.

D. Learning to Choose Good Actions

The training data was then split into $n = 35$ data sets, according to the particular next action taken in the training run. Within each of these data sets, the features were normalized as described previously, and then an SVM with linear kernel was trained for that set. Thus, we constructed $n = 35$ SVM's—one for each of the individual data sets—where each SVM corresponds to taking a particular next action.

With these SVM's available, we would then run our searches over extended depths (say, $d = 60$) on the particular rubble field being traversed. Whenever an atomic action needed to be chosen, the controller would first determine the feature vector associated with the current state of the snake and terrain. Then, for each of the $n = 35$ atomic action SVM's, a predicted reward value (or, in the classification case, the SVM margin) would be computed. After ranking these values, the controller would perform beam search and only expand the nodes associated with the top $n_{reduced}$ actions, thereby avoiding having to look through any actions not predicted to have decent outcomes.

V. RESULTS

A. Simulation Results

We show selected frames from a sample simulation run of the robot traversing a rubble field in Figure 2. This simulation

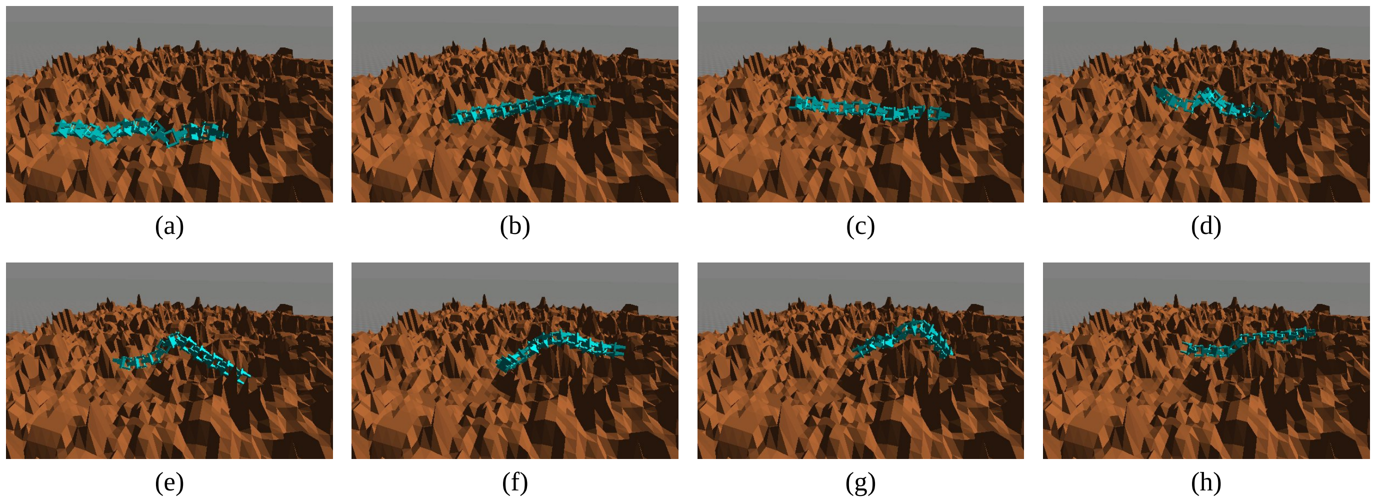

Fig. 2. Sequence of still frames showing the snake robot traversing a rubble field. (a) The snake sidewinds across a small flat patch of land. (b), (c) A tall but narrow obstacle is overcome by a lifting of the tail over the obstacle (right side of frame (b)). (d), (e) The snake prepares to traverse a larger obstacle by raising its center segments. (f) After lifting, there is a brief pause on top of the obstacle. (g) The big rock is cleared. (h) The snake relaxes after a job well done.

was generated using a search of depth $d = 60$ and beam-width $n_{reduced} = 5$. Training and computation of this path required approximately 3 hours. More videos of the robot traversing this and other rubble fields can be found online at `http://ai.stanford.edu/~jckoo/snake`

Some significant aspects of the snake gait can be observed from this simulation. In frame (a), we see the snake in the middle of a sidewinding motion, which works here since it is in a relatively flat patch of land. Frame (b) shows the snake getting past a narrow high point at its tail end by lifting and then resting on top of it, before nudging itself off the other side in frame (c). Frames (d)–(g) show the snake in a series of actions to climb over a fairly large obstacle. It starts to push itself together so that its center portion is raised (in frames (d) and (e)), allowing it to rest these bodies on top of the rock in frame (f). Another few actions finds the snake going over the rock in a jump-rope fashion (g), before straightening out and coming to a rest on the other side (h).

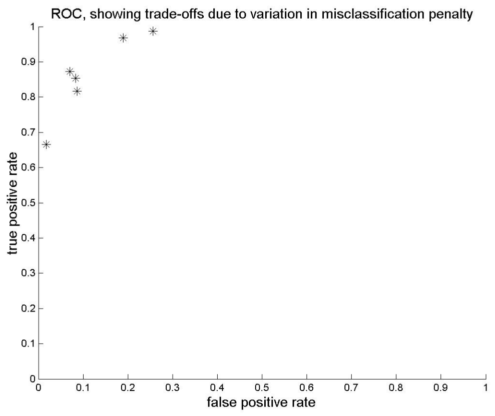

Fig. 3. Receiver operating characteristic for training data. Points are generated by varying the misclassification penalty C in the SVM formulation. Also included is the use of different penalties C_{fp} and C_{fn} for false positives and false negatives, respectively.

B. *Empirical Analysis*

1) Training Data Errors: Training with our particular feature set and target values turned out to perform very well. Looking at points on the ROC curve (Figure 3), we see that they are concentrated toward the upper left corner, meaning that we have good separation between positive and negative examples within our feature space. As far as the training data goes, classification performs reasonably well. Although the same analysis was not undertaken for the test data, we present a much more meaningful analysis of the test data in the next section by looking at how the ranking of actions with and without learning compare.

2) Rank Analysis: One aspect of our learning algorithm that we are concerned with is how often the actual best action is searched, given that we have a particular beam-width. Data from test runs (see Figure 4) indicate that even when we only expand a small subset of actions, the search algorithm will still perform fairly well because the best action will still be highly likely to be chosen. For example, if we always look at the top 10 actions from our SVM's, we will explore the best action approximately 70% of the time. Also, not always picking the best action might not be too detrimental to the progress of the snake robot, since the best action will be taken often. That is, we may be able to recover from a current sub-optimal selection by taking good actions later (since good actions will be expanded quite often). In addition, we should not underestimate actions which are not ranked as the best one to take, since they might still perform well.

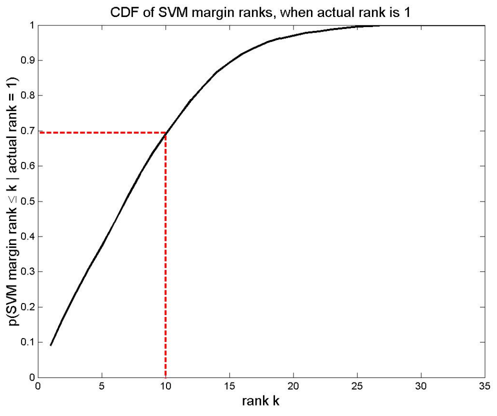

Fig. 4. Distribution of SVM margin ranks, for best action (taken from several test runs). Best action means that the actual metric for the best action is higher than the actual metric for any of the other actions. SVM margins are then computed for all of the actions and the actions ranked again. This plot shows where the best action stands amongst all the other actions in terms of its SVM margin ranking.

VI. Conclusions and Future Work

The results indicate that our feature set includes the salient features necessary for rubble field obstacle navigation. Our learning algorithm is able to choose good atomic actions, without sacrificing much in terms of performance. These atomic actions can be chained together to create novel snake gaits which move the snake considerable distances across complex terrains. We show that we can significantly reduce the time it takes to compute such paths and still achieve some certainty as to the goodness of the gait, so that automatic rubble field traversal by a real snake robot can be made feasible.

There are still many potential avenues for further investigation. A deeper analysis into the optimality of the beam search should be undertaken, so that perhaps the beam-width may be varied in different situations. One method of doing so would be by considering a more descriptive measure of the goodness of a particular atomic action besides just its rank. Such a description may involve using the computed margins in a way that indicates the probability of an atomic action being a good action for that scenario.

We might also wish to apply the atomic actions approach toward other terrain types, perhaps requiring the generation of an additional set of atomic actions. This way, we will have a more robust snake robot capable of traversing an even larger variety of terrains (besides the rubble fields considered). An idea worth pursuing is the ability to learn good atomic actions (as opposed to learning good *sequences* of atomic actions).

Robustness can also be achieved by implementing higher-level planning. Our current learning algorithm performs well when multiple short-sighted good actions sequenced together can lead to an overall good set of actions (i.e., when we can traverse the rubble field by moving laterally all of the time), but if there is no good straight-line path then our algorithm may return an unsatisfactory solution. A higher-level motion planner would be able to chart a course around an obstacle, rather than one that just goes over or slightly sideways of the obstacle. We may need to apply reinforcement learning, in order to take a more far-sighted approach to snake planning.

A major future goal is to be able to implement our algorithm on a physical snake robot. We can then judge real-world performance of the atomic actions approach.

VII. Acknowledgements

I would like to thank Prof. Andrew Ng and Zico Kolter for many helpful ideas and thoughtful discussions throughout the course of this project.

References

[1] S. Hirose, *Biologically Inspired Robots: Snake-like Locomotors and Manipulators*. Oxford: Oxford University Press, 1993, translated by P. Cave and C. Goulden.

[2] R. Smith. Open Dynamics Engine. [Online]. Available: http://www.ode.org

[3] V. N. Vapnik, *Statistical Learning Theory*. New York: John Wiley & Sons, 1998.

[4] T. Joachims, “Making large–scale SVM learning practical,” in *Advances in Kernel Methods — Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. MIT Press, 1999, ch. 11, pp. 41–56. [Online]. Available: http://svmlight.joachims.org

[5] K. Shoemake, “Animating rotation with quaternion curves,” *ACM SIGGRAPH Computer Graphics*, vol. 19, no. 3, pp. 245–254, July 1985.