Collision Detection in Legged Locomotion using Supervised Learning

Finale Doshi, Emma Brunskill, Alexander Shkolnik, Thomas Kollar, Khashayar Rohanimanesh,
Russ Tedrake, Nicholas Roy

Abstract— **We propose a fast approach for detecting collision-free swing-foot trajectories for legged locomotion over extreme terrains. Instead of simulating the swing trajectories and checking for collisions along them, our approach uses machine learning techniques to predict whether a swing trajectory is collision-free. Using a set of local terrain features, we apply supervised learning to train a classifier to predict collisions. Both in simulation and on a real quadruped platform, our results show that our classifiers can improve the accuracy of collision detection compared to a real-time geometric approach without significantly increasing the computation time.**

I. INTRODUCTION

Legged locomotion over rough terrain introduces a host of challenges beyond those typically addressed in the locomotion community. Unlike flat terrain—the focus of much of the existing literature—achieving kinematically-feasible, collision-free foot trajectories is non-trivial when walking on rough terrain. Some promising work on uneven terrain has been developed based on intelligent hardware design using compliant legs [1]. These approaches work well with elegant controllers over a variety of rough terrains, however they may not perform as well on terrain where accurate foot placement is crucial for successful and stable walking (e.g., walking along the edge of a cliff).

Our interests lie in quadruped locomotion over rough terrain. In such situations collisions with the terrain or self-collisions while executing leg motion are of significant concern: these collisions can cause the robot to lose stability and fall, potentially damaging itself, or severely hinder the robot's speed in reaching a goal location.

The major issue with computing collision-free trajectories is tractability. Testing for collisions between all moving components (such as the stepping leg) and the terrain requires a large number of demanding geometric tests. Leg trajectories must also be tested for self-collisions and kinematic feasibility. It is tempting to reduce the complexity of these tests with approximations such as testing only for collisions with the moving foot (instead of the full moving leg). However, Figure 1 shows a common failure case of such assumptions; here, the robot's calf collides with an obstacle and prevents the foot from landing on the opposite side. We note that even if the full geometric was tractable, such tests would require accurate kinematic and dynamic models of the robot that are hard to obtain as calibrations often shift with time.

F. Doshi, E. Brunskill, A. Shkolnik, T. Kollar, K. Rohanimanesh, N. Roy and R. Tedrake are members of MIT's Computer Science and Artificial Intelligence Laboratory. (finale,emmab,shkolnik, tkollar,khash,russt,nickroy)@mit.edu

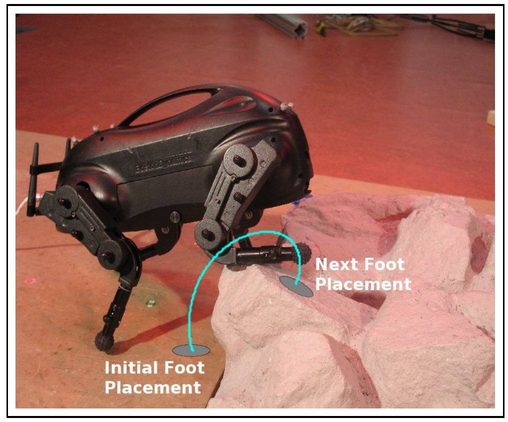

Fig. 1. An example where the swing foot trajectory is collision-free, however other parts of the swing leg (in this case the elbow) will be in collision with the terrain during the swing.

In this paper we present a fast approach to trajectory collision detection using learning. Instead of simulating and updating a geometric model of the robot during trajectory generation, we build classifiers to predict whether a trajectory is collision-free. The classifier takes in a set of input data about the current terrain and robot configuration and produces a label of whether a collision is predicted. We focus specifically on the problem of predicting collisions during the motion of a robot leg and foot during a stepping motion, and we refer to this motion as the "swing leg trajectory." We show that our learning approach produces a significant improvement over a simple heuristic without substantially increasing the computation time used for predicting collisions.

The rest of the paper is organized as follows. Sections II and III discuss related work and present the overall system. Sections IV-B and IV-C describe two different supervised learning techniques—AdaBoost and Support-Vector-Machines (SVMs)—for solving the problem. In Section V we present our empirical results in both simulation and a real quadruped platform. Finally, we summarize the paper in Section VI and describe some future directions.

II. RELATED WORK

The graphics community has extensively studied the problem of collision detection (e.g., [2], [3]). Drawing from this work, we implemented a baseline predictor that checks the entire step leg for collisions with every other part of the terrain at some resolution δt for the entire trajectory. A number of methods improve upon the basic algorithm, looking for roots of functions over time of convex polyhedra [4] or using *space-time* bounds to compute the time of possible intersections [5]. Hubbard [5] also approximates the surface with a tree of successively finer resolution balls.

In the area of legged locomotion, swing-trajectory collision detection typically uses spatial partitioning or bounding volumes hierarchies. For example, Kuffner [6] applies computational geometry to detecting collisions in humanoid robots. After surrounding objects with a 'protective' convex hull they bound the minimum separation distance between objects using object velocities and apply a Voronoi algorithm to find collisions. With clever pruning, a trajectory can be checked in 10-30 ms; however, this approach does not scale to checking many candidate swing trajectories in real time.

Rapidly-exploring Random Trees (RRTs) [7] has also been a popular method for generating collision-free trajectories, however this approach requires a collision-detection at each intermediate point during the trajectory. Thus, RRTs for generating swing foot trajectory may fail to find a smooth trajectory in real time [8] given the tight time constraints for achieving a desired locomotion speed, as it is required in the LittleDog locomotion problem. Our approach checks the entire trajectory at once.

Little work exists on using machine learning for collision detection. Quinlan et al. ([9], [10]) examined collisions in the AIBO RoboCup domain, where collisions primarily occur between legs and between other robots. They first flagged outliers using statistics on successful and colliding joint angles for a variety of scenarios [10]. They next reduced memory requirements by training an SVM on the data [9]. Using features such as leg joint angles and various gait parameters, and parameter tuning, they produced a classifier with similar accuracy to the first approach. In contrast, we investigate this problem in extreme terrains where collisions primarily occur between the robot's legs and terrain rubble.

III. Quadruped Locomotion

The DARPA Learning Locomotion project is focused on research on learning locomotion controllers to allow a quadruped robot (LittleDog) to traverse rough terrain (see Figure 1). LittleDog weighs 3.0 kg with dimensions: $0.338 \times 0.178 \times 0.142$ meters, and total leg length of 0.180 meters. A high resolution motion capture system (MoCap), the Vicon MX system, reports the accurate positions of the robot and the terrain. The average rock size on the terrain is 0.10×0.15 meters, and the largest obstacle is 0.25×0.18 meters.

A. Control Architecture

LittleDog contains 18 DOF, with six unactuated degrees specifying body position and orientation, a two DOF universal joint at each hip, and a hinge joint at each knee. Each joint contains a position sensing encoder. To achieve a basic gait, we independently control the robot whole-body center of mass and the position of a foot, reducing the control problem to six degrees of freedom for the center of mass.

We rely on a hierarchical decomposition of the control problem consisting of two main components:

1) Foot placement planner: Computes a center of body (COB) trajectory to the goal; searches for the best possible foot placement for the next step.
2) Joint trajectory controller: Combines plan with a learned stability function to achieve the desired foot placements.

To choose the best foot and leg pose for the next step, the foot placement planner randomly samples hundreds of candidate COB and foot placement pairs around the current position of the swing foot and scores them based on costs such as predicted foot slippage, violation of static stability, and whether the swing trajectory generated based on a sample foot placement leads to a collision. The best scoring candidate from the set is executed as the next foot step.

For real-time performance, we have a fast deterministic algorithm for generating a swing-foot trajectory given the current and next foot-placement. Our algorithm first computes the convex hull of the terrain strip connecting the current and the next foot placement. The points in the convex hull set are translated based on a deflective forcefield on the terrain surface to compute a clearance distance. Finally, we take the new set of points and fit a cubic spline which guarantees a collision-free trajectory for the foot of the robot. Although the foot trajectory is collision-free, other parts of the leg such as the knee or hip may collide with the terrain while following this trajectory.

For comparison, we have also implemented a simple collision detection system using *line* and *cylinder* models. The first method (the line model) models each link of the swing leg as a line tangent to the frontal surface of the swing leg. The second method (the cylinder model) fits an axis aligned bounding cylinder along each link of the swing leg. The swing leg is in collision with terrain if any points of the object modeling the swing leg penetrates the terrain. The line model is much faster to compute than the cylinder model, however, it fails to detect collisions that may happen to the sides and back of the swing leg.

Even the line model can be slow if we need to check many points along the trajectory, and prior to this work, we used a simple heuristic we title "Pre/Post." Pre/Post is a simple classifier that checks whether the initial and final step leg configuration intersect with any part of the terrain using the line model. This heuristic is very fast to compute, but it completely misses mid-trajectory failures.

IV. Learning to predict collisions

Since hundreds of potential foot placement and COB pairs are sampled for each foot step that is executed, any collision checking algorithm must be fast and scale well with the number of potential foot step trajectories. The geometric approaches discussed earlier are very accurate, but they require a large computational cost and are not suitable for real time planning. Our approach to alleviate this problem is to use machine learning techniques. We employ supervised learning to train classifiers that can predict in real time whether due to geometric constraints trajectories are likely to cause a collision.

Supervised learning methods use a training set of (data, label) pairs to discover patterns that will allow the classifier to predict the labels of new data. Here, the data is a vector

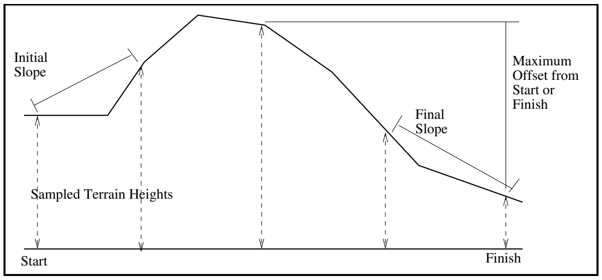

Fig. 2. Cartoon of basic features: we sample terrain heights between the start and end foot positions using the motion capture system; these values are used to compute more complex features.

of information about a trajectory, such as the terrain heights and the relative position of the robot's legs, and the label is whether that trajectory resulted in a collision with the terrain. To run in real-time, all the elements of input data (features) given to the classifier must be efficient to compute.

In simulation, the "ground truth" label of whether a collision occurred was determined by computing the full cylinder model of the robot's leg; in live experiments with a real robot the labels were determined by a human watching the robot's leg for collisions. Once the classifier was trained using the label data, we tested the quality of the classifier on a test set of data that the classifier had not encountered. The classifier prediction based on the test data is compared with a ground truth label to measure the classifier accuracy.

A. Feature Selection

A key question is what aspects of a step leg trajectory are both fast to compute and provide information about whether the trajectory will collide—this is the question of feature selection. Given an initial and final foot placement, we uniformly sample the terrain points on the line connecting the foot placements and record the terrain heights in the robot's local coordinate frame (see Figure 2). Transforming the terrain features into the robot's local coordinate frame makes the classifier invariant to the global position and orientation of the robot on the terrain. Thus, we can generalize across a large set of situations which share similar local terrain features and similar robot poses relative to the local terrain. We also include information about the robot's current and desired pose, the robot's current position, and the identity of the swing leg as basic features.

The features discussed above can be quickly extracted from a planned trajectory. We can ease the burden on the classifier by also providing features likely to be relevant to the problem. For example, the slope of the terrain or the height of an obstacle (both which can be computed efficiently from the vector of terrain heights), might be strong indicators of whether the trajectory will collide. These additional quantities, computed from the basic trajectory information, are used as additional features in our classifier. Sections IV-B and IV-C will describe the derived features used to train the respective classifiers.

B. AdaBoost

In our initial exploration of what derived features would be strong indicators of a collision, we discovered that no

TABLE I
THE BASIC ADABOOST ALGORITHM [11].

ADABOOST
- Initialize all sample weights to be uniform
- For m = 1..M:
  - Choose weak classifier with the minimum weighted error e; abort if the error is greater than 0.5.
  - $\beta_m = \frac{e}{1-e}$.
  - Update the weight distribution. For sample x^i:
    - $w_i = w_i$ if the weak classifier picked the wrong label.
    - $w_i = \beta w_i$ if the weak classifier picked the correct label.
- Output the label c that maximizes the $\sum_{m \in c} \log \frac{1}{\beta_m}$

single feature (or even feature set) was strongly correlated with the collisions. However, several features were weak indicators: for example, although the robot suffered from collisions on all kinds of terrain, more collisions occurred when the robot tried to go down a steep slope with its front leg. Used individually, each of these features had an errors of 30 to 40 percent, too high for any real application.

AdaBoost [11] is an algorithm that combines a collection of weak classifiers—trained on different sets of features—to get better prediction (see Table IV-B for an overview of the algorithm). The AdaBoost training process consists of several rounds. At the beginning of the training process, each training sample is assigned a uniform weight. During each round, AdaBoost first trains a weak classifier that minimizes the weighted error using the current weights. The weights of correctly classified samples are then reduced; in each future round, AdaBoost generates a new weak classifier that performs well on the misclassified samples. We also maintain a weight on each selected weak classifier that describes how successful it was on the training data. Once AdaBoost is trained, each classifier votes for the label on new instances to be labeled proportionally to the weight of the classifier.

The quality of the final prediction ultimately depends on the quality of the weak classifiers. We tested linear, logistic, and quadratic multivariate regression models before choosing decision trees. Among the weak classifiers, decision trees had the best performance because they were able to handle highly nonlinear and disconnected decision boundaries. For example, the relevance of many features depended on which of the robot's legs was moving; decision trees handled these situations by branching on the front and back legs. Decision trees could also robustly combine discrete and continuous data of different scales. We used a standard decision tree implementation [12] to create the decision trees.

Section IV-A described how we might create a large vector of basic and derived features as input data for the classifier. We chose additional features by evaluating potential features that we believed might provide predictive power. The learning algorithm will identify the relevant features from the large and potentially redundant initial feature set, and therefore initially providing potentially superfluous features will not harm performance. The features we used included the overall slope of the terrain, the height of any obstacle the robot was trying to step over (0 if there was no obstacle), and the initial and final slope of the terrain.

TABLE II

FEATURE SETS USED; STARRED SETS WERE MOST DISCRIMINATIVE.

Set	Features (all sets included the step leg, Pre/Post value)
1	overall slope
2	height of obstacle to be stepped over
3	initial slope
4	final slope
5	initial joint angles
6	final joint angles
7*	step length scale, vertical terrain heights
8	(planar) step length
9	maximum terrain slope
10*	center of body, initial joint angles, initial slope
11*	center of body, final joint angles, final slope
12	max height - min height
13	max(depth of dip, height of obstacle) to be stepped over
14	step leg, Pre/Post only

Each of the derived quantities weakly helped predict a collision and was the basis for a weak classifier—a decision tree—that AdaBoost would later combine into a strong classifier. All together, we used fourteen combinations of basic and derived features to train fourteen decision trees as weak classifiers (training on the entire feature vector, instead of parts of it, was too complicated for one decision tree to handle). Each feature set consisted of a vector containing the step leg (which was so important we included it in all of our tests), whether our Pre/Post heuristic predicted a collision, and some other derived data (for example, the initial slope of the terrain or a vector of terrain heights). Thus, a decision tree might first split on the step leg (front or back) and then split on particular values of the remaining elements of the vectors. Table II lists all of the feature sets we used. The starred features turned out to be the most relevant decision trees (those picked first by AdaBoost).

The collision detection problem is asymmetric; it is much worse to classify a collision as safe than to classify a safe trajectory as a collision (we can always sample more step trajectories). Thus, we wish to bias the classifier to penalize missed collisions more severely than missed safe trajectories. In order to do this, within the training loop, we both reduced the weight of samples that were correctly classified and samples that were misclassified as false alarms. The missed collision samples (false negatives) remained with the highest weight, to force the learning to expend more effort in classifying those training instances correctly in future iterations. Section V-A, we show that this bias significantly reduced the rate of missed collisions.

C. *Support Vector Machines*

In addition to AdaBoost there exist a wealth of other supervised classification techniques in the machine learning literature. One particularly popular approach is Support Vector Machines (SVMs). Both SVMs and AdaBoost are well designed to handle binary classification tasks where the input space cannot be linearly separable into the two desired classes. While AdaBoost collects weak classifiers tailored to particular examples, SVMs instead project the input space to a higher dimensional space where the data is more likely to be linearly separable. The simplest example of the benefit of this approach is the XOR function which is not linearly separable in the input space x_1, x_2, but if the data is projected into a polynomial of the input space $(x_1x_2, x_1^2 ...)$ then it becomes linearly separable. More formally, the data is mapped into a new space using a kernel function and the classifier is chosen which maximizes the margin: the distance between positive and negative examples in the feature space. A small modification to the optimization performed by SVMs allows different weight to be placed on correctly classifying positive and negative examples, providing a tool to more heavily penalize misclassifying colliding trajectories as safe than classifying safe trajectories as collisions.

We applied SVM classification to attempt to predict leg collisions. After exploring the feature space, we elected to use a feature space consisting of the following features: (x, y, z) coordinates of the 10 points marking the leg trajectory, whether the spline was considered kinematically feasible to execute, which leg was stepping, the 6 coordinates describing the final center of the body, and the start and end values for roll, pitch and yaw of the robot's leg.

V. EMPIRICAL RESULTS

We tested the classifiers both in simulation and on an actual quadruped robot. The real world terrain was machined from the same specifications used in the simulation, so the terrains in both experiments were nearly identical.

The best SVM classifier rarely predicted collisions as non-collisions (error rate 1.3%), but at the expense of high overall error rate of 48%. Since the classifier was discarding a very large percentage of good samples (i.e., it had a very high false positive rate), we elected to focus our attention on the AdaBoost classifier which showed superior performance on our dataset. The results presented below are based on the AdaBoost classifier unless otherwise noted.

A. *Simulation Results*

As an initial validation of our model, we first tested the collision learning in simulation. This also allowed us to establish a reliable "ground truth" for our collision detection by using a cylinder model of the moving leg and foot to determine if a collision would occur. Experience on the robot had shown that this geometric model was reliable but slow.

Figure 3 shows how AdaBoost's performance (based on 5-fold cross validation) varies with the number of training samples. Approximately 10,000 sample trajectories were collected. The error rate drops quickly even with a relatively small number of training samples.

Figure 4 shows the response of the AdaBoost classifier described in Section IV-B as we tuned it. The classifier does well—false alarm rates of less than 10% and, importantly, missed collision rates of less than 5%—for many choices of the parameters. Within the weak classifiers, we found that penalizing false-alarms at one-third of the penalty for missed collisions produced the best performance. Within the AdaBoost classifier, we reduced the weights of correctly classified samples and false-alarm samples equally.

Table III shows how long each classifier required on the machine that was used to run the robot. The times

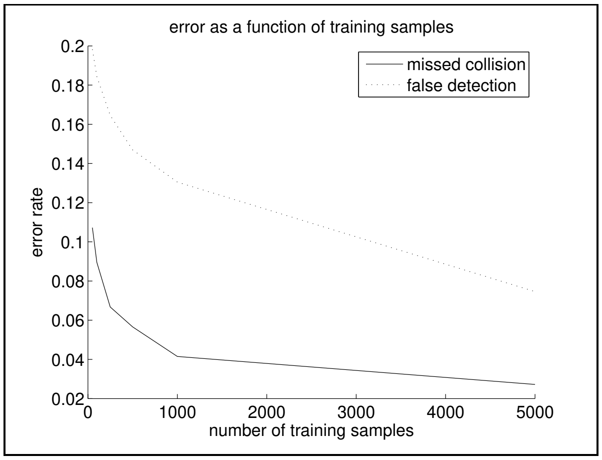

Fig. 3. AdaBoost error rate as the number of training samples was varied.

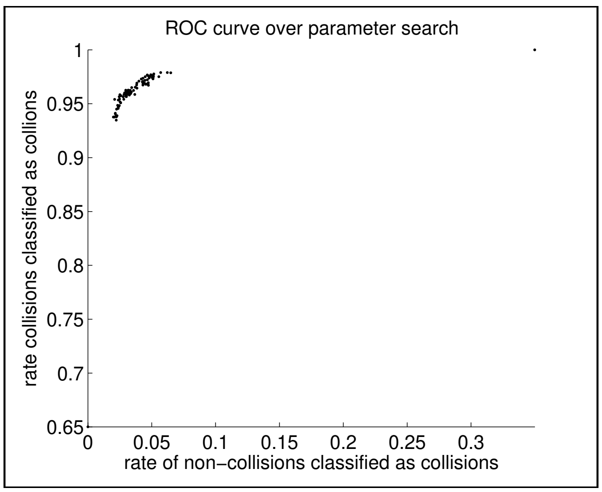

Fig. 4. Operating characteristic as AdaBoost parameters were varied.

are the average time required for computing collisions on each step, averaged over 42 simulated steps. On each step, approximately 800 trajectories were sampled and evaluated for collisions. The Pre/Post heuristic described in Section III was the baseline for our tests. Recall this heuristic only tested for collisions at the beginning and end of step and therefore often missed collisions that occurred in mid-trajectory. As a result, although it was not a very accurate collision detector, it was the fastest. Using the full geometric cylinder model was not an option for real-time performance.

Using AdaBoost with the Pre/Post heuristic as an input feature took three times as long as the Pre/Post heuristic alone—this roughly corresponded to computing the Pre/Post heuristic and then spending twice that time to do the AdaBoost computation. This was fast enough to use in real time. The AdaBoost classifier itself required only minutes to train and cross-validate, even with close to 10,000 samples. Thus, we were able to optimize a model through a fairly large parameter search over the course of a few hours.

Interestingly enough, the SVM ran slower than the full geometric model on our machine. Since it also took a long time to train, we did not test the SVM further outside of simulation. However, all of these timing numbers are based on un-optimized Matlab implementations of the collision detectors, and they could be reduced significantly by implementing them in a faster language.

TABLE III

CLASSIFICATION TIME FOR COLLISION DETECTION TECHNIQUES.

	Time for 800 samples (s)
Pre/Post with Cylinder Model	3.82
Pre/Post with Line Model	0.15
AdaBoost	0.33
SVM	6.89

B. Results on the Real Robot

As a training data set for the physical robot, we hand-labeled a data set of 341 steps.[1] No attempt was made to avoid collisions as the robot crossed the rugged terrain. The same people labeled the collisions for consistency.

Table IV shows errors from a 5-fold cross validation from the different classifiers. Since the robot was started in slightly different positions during each run, and the planner uses randomized heuristics, both the training and test sets contained some variability that suggests that the AdaBoost classifier may generalize to other similar terrains.

As described above, we used the prediction from the Pre/Post heuristic as an additional input into the classifier. The resulting classifier was slower than using the Pre/Post heuristic by itself, but as table IV shows, we were able to achieve more accurate collision predictions with the Pre/Post heuristic as input features. The most important predictors were the initial and final slopes of the terrain (coupled with the step leg), and the overall terrain trajectory.

TABLE IV

PREDICTIVE PERFORMANCE OF VARIOUS CLASSIFIERS ON ROBOT DATA.

	Missed Collision Detection	False Collision Detection
Pre/Post Only	.1994	.0674
AdaBoost Only	.1614	.1260
AdaBoost with Pre/Post	.1526	.1434

To test the performance of the planner, we collected additional data in which the foot planner actually used input from a classifier when choosing what steps to take. We interleaved the trials using only the Pre/Post heuristic with trials using the AdaBoost classifier to avoid bias from changing properties of the robot (e.g., encoder slippage, changing calibration parameters, etc.). In total, we collected 23 trials (626 steps) of performance data using the AdaBoost classifier and 26 trials (679 steps) using only the Pre/Post heuristic. Table V shows statistics from these trials. Statistics from trials where there was no collision prediction were used to train AdaBoost and are included for reference. We found that with the AdaBoost classifier, we had about 3 fewer collisions per run (unpooled t-test, $p = 0.0004$) and more successful trials (2-proportion z-test, $p = 0.03$).

[1]Simulator results were not used to train the robot since simulator collisions were qualitatively different from real world collisions.

TABLE V

PERFORMANCE OF TRAINED CLASSIFIERS ON ACTUAL ROBOT.

	Success Rate	Mean Collisions/ Run	Missed Detection Rate	Mean Steps/ Run
No Avoidance	0.31	6.50	NA	34.3
Pre/Post Only	0.65	5.88	0.18	31.9
AdaBoost & Pre/Post	0.87	3.15	0.11	30.9

We collected the data described above in two stages. Human-labeling of collisions is inherently subjective, so in the next set of trials we differentiated between serious collisions (made a noise or did not allow the robot to complete its desired step) from minor grazing of the terrain. Table VI shows the same statistics for only major collisions. This subset of the complete data set included 16 trials, 413 steps for Pre/Post and 14 trials, 384 steps for AdaBoost. In this subset the classifier missed even fewer collisions.

TABLE VI

PERFORMANCE ON ROBOT, PENALIZING MAJOR COLLISIONS ONLY.

	Success Rate	Mean Collisions/ Run	Missed Detection Rate	Mean Steps/ Run
No Avoidance	0.31	6.50	NA	34.3
Pre/Post Only	0.69	3.63	0.11	30.6
AdaBoost & Pre/Post	0.93	0.54	0.02	29.0

Anecdotally, the learned collision prediction appeared to lead to qualitatively different trajectories. The planner appeared to pick trajectories that not only provided better footholds, but also reduced the likelihood of a collision. Many times the robot appeared to take footsteps that were further away from all obstacles than it would otherwise do compared to using the Pre/Post heuristic alone to predict collisions. The results in both the success rate and the collision rate of the robot support this qualitative difference.

VI. DISCUSSION AND CONCLUSION

We have demonstrated the application of a machine learning classification technique, AdaBoost, to increase the accuracy of predicting collision-free foot step trajectories for robotic quadruped locomotion. By combining an AdaBoost classifier with a simple collision check at the initial and desired foot locations, we significantly reduced the number of collisions that occurred during experiments run on a real quadruped robot platform compared to only using the simple start and end classifier. The resulting classifier only added an additional 0.0004s of overhead to evaluate each potential foot trajectory for a step, and is fast enough to be used as part of a real time locomotion planner which samples thousands of potential steps before selecting one to execute.

In contrast to previous machine learning work [9], our work focuses on detecting collisions over rough, varying terrain. Detecting collisions in this domain is critical to ensuring that the robot can successfully reach its goal. We also demonstrated that in addition to reducing the number of collisions that occurred during a single terrain traversal to a goal location, the robot also reached the goal statistically significantly more often when using the new collision classifier. While our experimental results are fairly small due to the temporal expense of running trials on the real robot, the results demonstrate the promise and utility of this approach. Future work should expand upon features and labels are most useful to the classifier, and compare this machine learning approach to other geometry-based approaches.

For increased computational efficiency and robustness, future work could also consider a sequential version of AdaBoost, where a more complex classification scheme is used only if a simpler classifier is unsure about the safety of a foot trajectory. Alternatively, a simple classifier could be used to separate trajectories into sub-categories such as safe, with potentially beneficial collisions (that might stabilize the robot) and with potentially harmful collisions. We exhibit some generalization because the robot walks over a different part of the terrain in each trial, but we wish to explore how the collision checking transfers to new terrain boards.

Finally, our approach is tailored to locomotion over rough terrain, but it may apply to other robotic applications where a large number of checks must be computed quickly, such as with multi-agent systems.

VII. ACKNOWLEDGMENTS

The DARPA Learning Locomotion project (AFRL contract #FA8650-05-C-7262) and the National Science Foundation Graduate Fellowship provided financial support.

REFERENCES

[1] R. Altendorfer, N. Moore, H. Komsuoglu, M. Buehler, H. B. Jr., D. McMordie, U. Saranli, R. Full, and D. Koditschek, "RHex: A biologically inspired hexapod runner," 2001.

[2] M. Lin and D. Manocha, "Collision and proximity queries. in handbook of discrete and computational geometry," 2004.

[3] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino, "Collision detection for deformable objects," 2004.

[4] J. Canny, "Collision detection for moving polyhedra," *MIT Artificial Intelligence Laboratory Memo*, no. 806, 1984.

[5] P. M. Hubbard, "Collision detection for interactive graphics applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 3, pp. 218–230, 1995. [Online]. Available: citeseer.ist.psu.edu/hubbard95collision.html

[6] J. Kuffner, "Self-collision detection and prevention for humanoid robots," 2002. [Online]. Available: citeseer.ist.psu.edu/kuffner02selfcollision.html

[7] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, U.K., Chapter 5, 2006.

[8] H. Lee, Y. Shen, C. Yu, G. Singh, and A. Ng, "Quadruped robot obstacle negotiation via reinforcement learning," in *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[9] M. Quinlan, S. Chalup, and R. Middleton, "Application of SVMs for colour classification and collision detection with AIBO robots," 2003. [Online]. Available: citeseer.ist.psu.edu/quinlan03application.html

[10] M. J. Quinlan, C. L. Murch, R. H. Middleton, and S. K. Chalup, "Traction monitoring for collision detection with legged robots." in *RoboCup*, ser. Lecture Notes in Computer Science, D. Polani, B. Browning, A. Bonarini, and K. Yoshida, Eds., vol. 3020. Springer, 2003, pp. 374–384.

[11] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *European Conference on Computational Learning Theory*, 1995, pp. 23–37. [Online]. Available: citeseer.ist.psu.edu/freund95decisiontheoretic.html

[12] "Matlab r2006a," 2006.