Footstep planning of walking robots
Project Midway Report

Tianyu Chen
Mechanic1al Engineering
tianyuc@andrew.cmu.edu

Yankun Xi
Computer Biology
yankunxi@yeah.net

Lulu Wang
Civil and Environmental Engineering
chloewang160@gmail.com

Christopher Kaffine
Electrical and Computer Engineering
ckaffine@andrew.cmu.edu

1 Introduction

Snake monster is a hexapod walking robot invented by CMU Biorobotics Lab. The applications of this kind of robots like urban search and rescue requires footstep planning on some rough terrain.

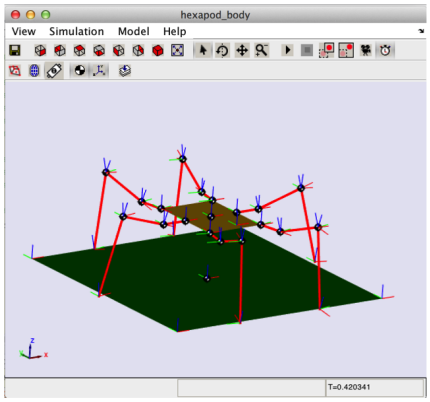


Figure 1: Hexapod Robot–Snake Monster

The goal of footstep planning is to develop an algorithm that can reliably navigate over rough, irregular terrain which requires active selection of footholds and robot posture to cross obstacles that are comparable in size to the length of the robots leg.
This report introduces works we made on locomotion of snake monster using several machine learning method. Firstly we capture data of real terrain or generate virtual terrain using models in Matlab. Then we form a cost function of points on the rough terrain which is a weighted sum of several features like height, angle and convexity. The vector of weight of each feature is initialized with a walker that is unstable. Then we use random forest and l1-regularized LR by supervised learning method to figure out how much each of these terms contributes to creating stable motion and then label the terrain by each points cost. Finally we find out points with low cost on this map and generate reference trajectories.

2 Related Work

A wide variety of work examines the task of locomotion over rough terrain. Early researches that focusing on planning legged robot footstep usually figure out a terrain reward map at first and give a recommend rank of each foothold. In order to map a rough terrain with rank of holding a feet of the robot well, a reward function as a weighted linear combination of several features is needed. But this method has some difficulty in practice. So Mrinal Kalakrishnan and Jonas Buchli (2011) introduced a novel method using terrain template, which is a discretized height map of the terrain in small area around the foothold. They choose some typical templates and give them each a reward value to build a template library. During the planning process, each candidate foothold is assigned the reward value of its closest matching template in the library by an off-the-shelf classifier. Then the robot will select the foothold with highest reward value for each step. Finally, a recommended path is generated.
Matt Zucker (2009), applied a similar approach called data-driven to perform a high-level planning. The basic idea is to develop a library that contains the data of previous planned actions and using it to do a quick selection. Matt use location cost to express relative difficulty of stepping on various terrain patches. To build a cost map for the terrain, a feature vector is extracted for each location.

3 Methods

We tried several methods like Logistic Regression by supervised learning, random forest and reinforcement learning. In this section, we will first introduce how we apply those methods to specific problems. Experiments are performed using these methods, and the results are given in next section.

Figure 2: Big Picture of the Project

3.1 L1-regularized LR by Supervised learning

By supervised learning, an expert gives his preferences on a set of training examples, and then the classifier learns a hypothesis H

$$H : R^{11\times 11} \rightarrow \pm 1$$

In the hypothesis H, we designed a cost function f which consists of 115 features of the ground. Every feature comes from a terrain template or terrain feature. 64 terrain templates represent the height and slope of the ground and 51 terrain templates represent the concavity/convexity of the ground. And each feature has a weight w_i, then the cost function can be written as

$$f = \mathbf{w}^T \mathbf{x}$$

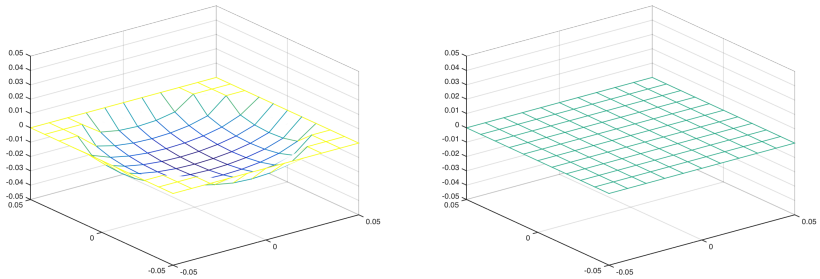
where **w** is the weight vector of every terrain feature and **x** is the terrain feature vector. Our goal is to learn an optimal **w** so that this linear separator can tell whether a ground is good or not correctly. Coming up with broad statements relating terrain shape to cost is easy because we know steep slopes are awful and dome-shaped are also terrible due to an easy slipping off. By contrast, small indentations are preferred as their shape is similar to the feet of the robot, large indentations are bad since the feet can get stuck.
In practice, it is difficult to design a precise weighting of those features discussed above. According to one of the related work—the LittleDog project, this also becomes a big problem[1]. But the combination of terrain features and terrain templates turns out to be a good choice in this issue[2]. Hence, both terrain feature and terrain templates are used in our project and introduced in the following paragraphs.

Terrain Features
There are several terrain features we can think of like:

1.height of the ground (or contact point)
2.slope and angle of the ground (or contact point)
3.concavity/convexity of the ground

The height of every footstep of the robot should be within a reasonable range since climbing a mountain or sliding down a hill is more dangerous than walking on the ground with same height. In other words, changing the height of footstep is penalized.
The slope and angle of the ground or contact point need to be as small as possible. This is easy to understand intuitively because the inclined ground causes sliding. As for concavity/convexity of a given ground, we apply terrain template method to represent concavity/convexity.

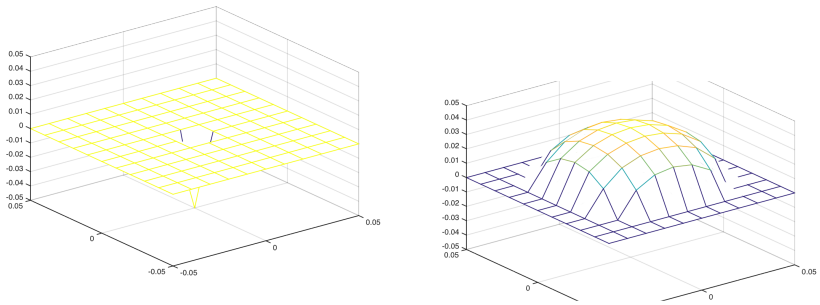
Terrain Template
Basically, we have 51 terrain templates with different widths and depths(or heights). The following figures are a few examples of terrain templates.



(a) template2 (b) template2

Figure 3: Terrain Template

Figure 3(a) is perfect because it is not likely to slip off if the robot puts its foot in this ground. Figure 3(b) is a flat ground and a good footstep choice. Figure 4(a) is a fair ground since a small hole exists in this ground. Figure 4(b) is bad since this is a dome-shaped ground. Figure 5 is also bad because the foot can get stuck in this kind of terrain.



(a) template3 (b) template4

Figure 4: Terrain Template

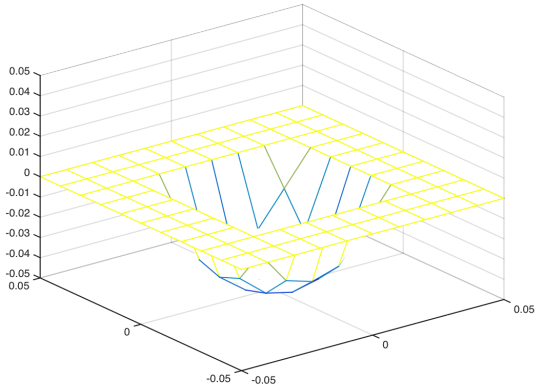


Figure 5: template5

Given a terrain height sample, each template returns a value that indicates the similarity of this given terrain and the template. That value is used in the cost function and defined as $x = exp(-h\sum_{i=1}^{n}(t_i - c_i)^2)$ where n is the number of points in the training sample, t_i is the height of the terran template, c_i is the height of the training sample, $h = 1$ is a fixed kernel bandwidth.
Further, we use l1-regularized LR to learn that optimal weight vector by optimizing J by sequential dynamic programming, J is defined as:

$$J = \sum_{i=1}^{m} -log(\frac{1}{1 + exp(-w^T x_i y_i)}) + \lambda||w||_1 \tag{1}$$

i means the i-th training data and $\lambda = 1$ is the regularization parameter and the use of an l_1 regularization term is aimed at producing sparse solutions.
since $-w^T x_i = f$, we have

$$J = \sum_{i=1}^{m} -log(\frac{1}{1 + exp(f_i y_i)}) + \lambda||w||_1$$

So the predicted label y_{test} for an input x_{test} is obtained by using

$$y_{test} = sgn(w^T x_{test})$$

3.2 Random Forest

Random forest is an unsupervised ensemble learning method for both classification and regression. It works by constructing a number of decision trees based on the training data and outputting the mean predictions of all decision trees. A set of ntree unpruned regression trees are generated based on bootstrap selecting from the original training data. We considered ntree = 1000. For each node, a random set of mtree points selected from the total points (M) is used for fitting a regression tree based on the bootstrap sample. We considered mtree = M/3. Since the contribution of every point for predicting the terrain cost is unknown, and the fraction of the points contributing more can be small, we considered a large m_{tree} to avoid missing the important points during the randomized point selection process.
Using the randomized feature selection process, we fit the tree based on the bootstrap sample (X1,Y1),...,(Xn,Yn) generated from the training data. During the tree generation process, a node with less than n_{size} training samples is not partitioned any further. We selected n_{size} = 1.
Because of the essential of random forest, it?s not necessary to know much of the physics foundations. We can use random forest to generate terrain cost function straight from 10 * 10 terrain matrix instead of terrain features. The challenge is that we need to generate a good training set including significant templates and corresponding cost values.
The same terrain template as in supervised learning part are used as training data for random forest so far.

3.3 Reinforcement Learning

Reinforcement learning is a form of unsupervised learning in which the algorithm learns by interacting with some environment and receiving rewards based on its actions. Through this process, the algorithm learns a value function which represents how valuable it is to be in a certain state. While the algorithm is running, it keeps track of the return for each state it visits, where the return is some function of the rewards received after visiting that state. The returns for each state are then used to update the estimate of the value function, which can be done by using the returns as inputs in a function approximation algorithm, the output of which is used as the new value function.

In our project, we attempt to learn a terrain cost function by simulating a robot walking over rough terrain. The rewards the robot receives are based on the simulated stability of the robot, and the terrain cost function can be extracted from the value function. The algorithm is split into episodes, with each episode proceeding as follows: the robot begins with some estimate of the terrain cost function, and starts in some initial state on a randomly generated terrain map. At each time step the robot has a finite set of footsteps it could choose to take next, each sending it to a new state. To decide which step to take next, the robot evaluates the value function for each of the possible next states and chooses the footstep which puts it in the best state. Next, we simulate the robot taking this step and get a measure of how stable the robot is in its new state. The robot is given a reward based on this value, which is used to update the returns for all states visited so far. The robot continues to do this until some end condition is met, at which point the current values for the returns are used as the inputs into a gradient descent algorithm which generates the next estimate of the value function. We run this algorithm for some set number of episodes, and the terrain cost function extracted from the final estimate of the value function is the output of the learner.

Episode Structure

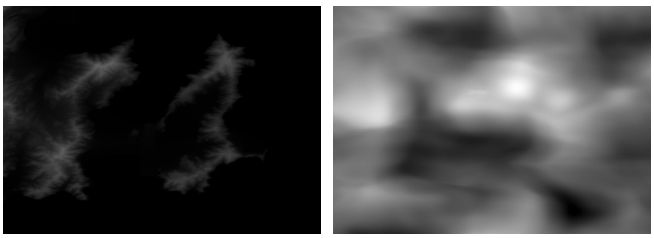
There are two slightly different ways of structuring each episode that we will try. One option is to set a goal for the robot in each episode which it attempts to approach. In this case, the value function is a linear combination of the robots closeness to the goal and the terrain cost function evaluated at each of the robots feet. The reward function in this setup is equal to the robots stability plus a time penalty to encourage shorter paths, with a bonus when the robot reaches the goal and an extra penalty if the robot falls over, and the return for each visited state is just the sum of the rewards received after being in that state. The episode ends when the robot either reaches the goal or falls over. The other way of structuring the episode is to allow the robot to simply wander around the terrain until it falls over. The value function in this case is just the sum of the values of the terrain cost function evaluated under each of the robots feet, and the reward function is the sum of the robots stability, a time bonus to reward long runs, and a penalty for falling over. Because of how the episode is structured it doesnt make sense for states visited at the very beginning of the run to be rewarded for something that happens towards the end of the run. To address this complication, we compute the return for a state as a weighted sum of rewards seen after visiting that state, with the weights.

4 Experiments and Results

Before midway of the project, we built the robot model and rough terrain model, then combine them together in Matlab. Then we got a rough cost function with a decent weights but not the best weights.

4.1 Supervised learning

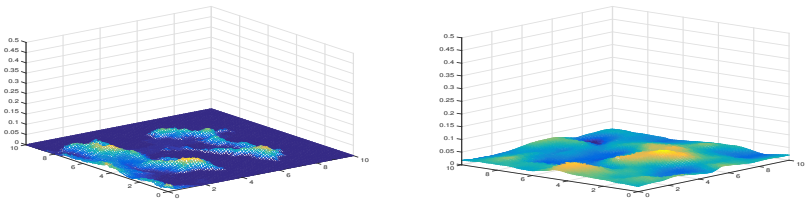
Training Data
There are two kind of training datas. We first label the terrain templates as good(1) or bad(-1). This set of training data is called template training data. In Table 1, we also compare the performance of the 51 terrain templates and 115 terrain templates.
The other set of training data comes from the incoming training data. Incoming training data is a group of height matrixes which we generate in Matlab by the terrain height maps from internet. Two terrain height maps are shown in Figure 6.



(a) Easy Test (b) Gentle

Figure 6: Height Maps

Every pixel of the terrain height map represents a point in a real terrain. The whiter the point is, the higher the terrain is. Then these two terrain height maps are processed in Matlab so that we get two 100 by 100 matrixes. Figure 7(a) and (b) are the mesh of that, and the height of each terrain data is between [0, 0.05]m.
We randomly extract 50 matrixes with size 11 by 11 from the above terrains and use these 50 small blocks as our training data. This set of training data is called incoming training examples. To use supervised learning, we also write an expert system to train the classifier in Figure 9.



(a) terrain2 (b) terrain2

Figure 7: Terrain Template

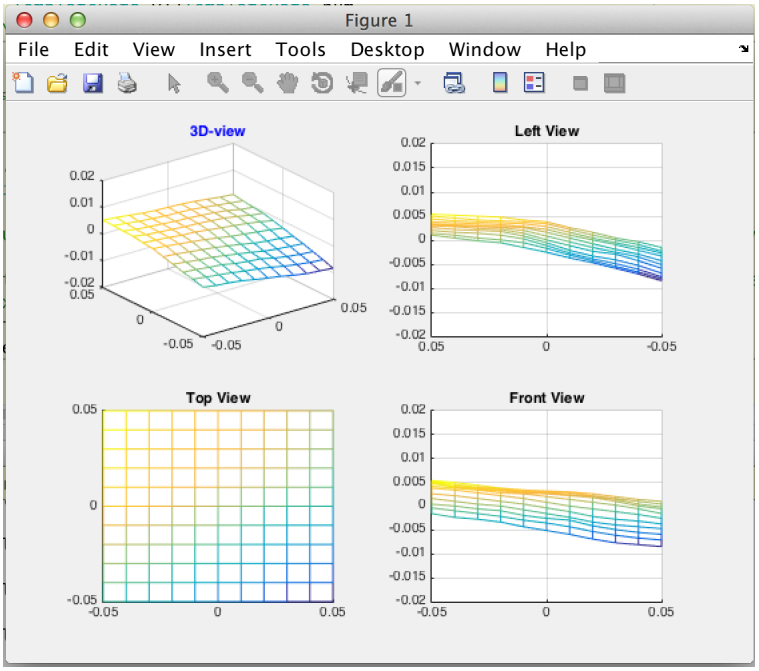


Figure 8: Views of Terrain Templates Using in Expert System

In Figure 8, a small blocks are shown to the expert by our program and then the expert gives his preference over these small blocks. Every Training Data will be labeled as good (1) or bad (-1). Next we calculate the 115 feature values of the given training data. Based on Formula (1), we use Matlab function fmincon to solve this optimization problem. The kernel bandwidth $h = 50$ and regularization parameter $\lambda = 0.05$. Test Data We also use this expert system to generate our test data, that is, select 20 small blocks with size 11 by 11 in the whole terrain map, and then the expert gives preferences to the 20 test data, next the test data is stored in a local directory and never used to train the classifier. There is some probability that several test data will be selected in the training data by accident. However, this probability is very small because only 50 training samples are used in our method, while every terrain map has 90^2 different small blocks.

Optimization

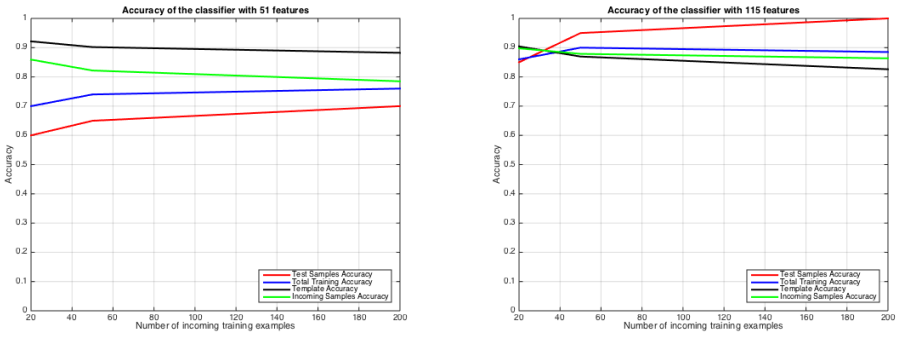
Finally, a cost function with weights w is learned and we randomly select 20 matrix with size 11 by 11 as the test data to test whether this cost function is good or not. The number of Test data is also 50. The cost function, training error and test error are introduced in the result part.

Results

Cost function:
The weight of the cost function calculated from fmincon satisfies our expectation from l1-regularization Logistic Regression because most of the weights are zeros or small values. Some features dominate the cost function because they have much larger weights over other features. That is because these dominant features can classify training data terrain efficiently. Training Error

Trial	Number of Temp	Ave Accu of Temp	Num of Incoming Training data	Ave Accu of Incoming Training data	Total Training Accu	Test Set	Ave Test Accu
1	51	0.9216	20	0.7000	0.8592	Test 1	0.60
2	51	0.9020	50	0.7400	0.8218	Test 1	0.65
3	51	0.8824	200	0.7600	0.7849	Test 1	0.70
4	115	0.9043	20	0.8600	0.8977	Test 1	0.85
5	115	0.8824	50	0.9000	0.8788	Test 1	0.95
6	115	0.8261	200	0.8850	0.8635	Test 1	1.00

Table 1 shows the experiment result by l1-regularized LR.
With the increasing number of Training examples, we compare the accuracy of Test Data, Total Training Data, Terrain Template and Incoming sample in Figure 5.1 and Figure 5.2



(a) 51 features (b) 115 features

Figure 9: Results Comparasion

According to Figure 9 (a) and (b), an increasing number of Incoming Training samples leads to lower Terrain Template accuracy, higher incoming samples accuracy. This is because the optimizer tends to fit as much training data as possible. However, we get lower Total training accuracy but higher test data accuracy. This may indicate that more incoming training data can make the classifier better. If we compare Figure 9 (a) and (b), it is obvious to find that classifier with 115 features is better than with 51 features. Since classifier with 115 features contains terrain feature slope and concavity/convexity, this classifier has stronger capability to handle more details of the terrain.

4.2 Random Forest

Training data is a group of height matrixes and the corresponding cost. It?s the same as the terrain templates in supervised learning.
Then we use the training data to generate a group of decision trees trying to predict cost given height matrix.
For a new terrain, the cost is given by mean predictions of the group of decision trees.
Now, because the limit of incomplete training data, the cost is relatively higher than we expect. The terrain having higher cost looks harder to stand on than a one having less cost.
The test data set used in this method is the same as in supervised learning.

Trial	N_{trees}	N_{size}	Num of Training data	Ave Accu of Training data	Test Set	Ave Test Accu
1	1000	5	51	0.9804	Test 1	0.65
2	1000	5	115	0.9043	Test 1	0.85
3	1000	1	115	0.9652	Test 1	0.90
4	1000	1	115	0.9565	Test 1	0.90

It shows that the number of training data (and also the significance of terrain they represent) is very important in this algorithm. N_size also affects accuracy of both training data and test data.

5 Summary

- As a milestone, we have set up all the training environments(models, terrain visualization, cost function design).
- LR by supervised learning is used to train a good w. Several experiments are performed and the results are presented in the midway report. We are going to do more experiments and improve this method by adding more templates in the final report.
- Random forest is applied to our project, experiments are performed by the same test data as LR.
- Reinforcement learning method is discussed in this midway report and there will be experiments in the final report.

References

[1] P. Abbeel and A.Y. Ng. Apprenticeship learning via inverse reinforcement learning. In Proceedings of the twenty-first international conference on Machine learning. ACM New York, NY, USA, 2004.

[2] Matt Zucker A Data-Driven Approach to High Level Planning. PhD thesis, Carnegie Mellon University, 2009.