LEARNING FROM ACCELEROMETER DATA ON A LEGGED ROBOT

Douglas Vail **Manuela Veloso** [1]

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213 USA
Email: {dvail2,mmv}@cs.cmu.edu

Abstract: Robot calibration for an environment is a tedious task that usually involves extensive, if not total, human intervention. However, robots have sensing mechanisms, in particular accelerometers, which could in principle be used to detect specific environmental states. In this paper, we contribute several approaches for robots to detect their state using accelerometer data. In particular, we use accelerometer data from a four-legged AIBO robot. We present a surface detector that identifies the surface under the robot as it walks using a decision tree. We present the features used for this classification. Additionally, since AIBO robots can easily become entangled on obstacles or other robots in multi-robot environments, such as robot soccer, we contribute results that show effective detection of robot state, again based on accelerometer data. Finally, we examine a third, more challenging problem: predicting gait velocity from accelerometer data. We use a k-nearest neighbor approach with a library of labeled accelerometer data for velocity prediction. We show that while this prediction is complex, accelerometer data can still be correlated with velocity. Our work, as reported in this paper, demonstrates the general use of robot accelerometer data for automatically detecting robot or environment state without tedious manual calibration.

Keywords: robotics, state detection, surface detection, velocity prediction, accelerometer, k-nearest-neighbors

1. INTRODUCTION

Robots often need to be calibrated for their specific operating environment in order to achieve peak performance. The mechanics of the calibrations vary widely, but typically they are time consuming and may require extensive human intervention. In this paper, we contribute learning approaches that use robot accelerometer data to model the state of a robot or the state of its environment. For our particular experiments, we have used the four-legged AIBO robot and the robot soccer domain.

Calibration is needed when designing gaits for the AIBO robot. In order to achieve a fast, stable walk, a human must spend several hours running the robot on the surface where it will be walking and adjusting the parameters that define the robot's gait. This adjustment is something between random guessing and a black art; experience

[1] This research was sponsored by Grant No. DABT63-99-1-0013, by generous support by Sony, Inc., and by a National Physical Science Consortium Fellowship with stipend support from HRL Laboratories. The content of this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

speeds the process, but even old hands at walk tuning are frequently surprised by the form the fastest walk takes.

In this paper we explore how we can train a surface detector from robot accelerometer data. Equipped with the learned model, a robot can detect the walking surface using its internal accelerometer. With this knowledge, a robot can either select the correct set of human-generated walk parameters or, if it can learn on its own, the robot can optimize its existing gait for the new surface automatically. Since generating a large number of parameters sets for various surfaces is impractical, ideally the robot should learn to optimize gaits on its own. Having a source of feedback for this learning is what motivates the third set of experiments on velocity prediction discussed in this paper.

Of course, state is not limited to the surface under the robot's feet. In a multi-robot environment, such as robot soccer, robots can get entangled on each other or become stuck against other objects, such as walls (while manipulating a target object, such as the ball). It is very important for the robot behavior selection mechanism to know its state, namely if the robot is free, entangled, or stuck. We show effective state detection using the robot's accelerometer.

As we mentioned, feedback is required for the robot to learn new walk parameters. We explore using accelerometer data to estimate the robot's velocity. This provides an internal source of feedback for learning. It is advantageous for the feedback to come from an internal source because it allows the robot to learn anywhere without additional calibration (e.g. vision, a motion model, and a map for localization). Localization is required to generate the initial, labeled training data, however.

In the remainder of the paper, we will describe the AIBO robot and its operating environment. We will examine what sort of data is returned by the accelerometer. Then, we will briefly review related work before providing a detailed description of our surface and state detection. Finally, we will examine using blocks of accelerometer data to estimate the robot's velocity.

1.1 The AIBO Robot: Sensors, Software, and Environment

The commercially available Sony AIBO ERS210a is a small, quadruped robot that resembles a small dog or cat. The CPU is a MIPS processor running at approximately 400 mhz. The robot has 15 degrees of freedom - three for each of the legs and an additional three from the head. We do not count the tail, mouth, or LEDs in this tally since they are not directly relevant to walking. The head is heavy enough, in proportion to the body, to affect the robot's balance as it walks.

Table 1. Selected Walk Parameters

Per Leg Parameters	
Neutral Position	A point in space that the foot must pass through in the ground stroke.
Lift Velocity	The desired foot velocity when a foot leaves the ground.
Down Velocity	The desired foot velocity when a foot touches the ground.
Global Parameters	
Body Angle	The angle between the robot and ground along the x axis.
Body Height	The height of the body midpoint.
Walk Period	The time to complete a full cycle.

In addition to its camera, the robot also has an accelerometer. The accelerometer returns a real valued estimate of the robot's acceleration along the x, y, and z axes. The values are limited to the range of [-2,2] gravities for each axis and are sampled at 125 hertz. We define positive x as the direction that the robot is facing, positive y as directly away from the robot's left side, and positive z as up from the floor. Figure 1 shows sample accelerometer data.

During the experiments, the robot ran code built on top of our RoboCup Legged League team, CMPack'03. This allowed us to run the robot on an approximately 3 by 4 meter soccer field and take advantage of our existing code for vision [1] and localization [5]. CMPack'03 also provided a parameterized walk engine that allows us to specify different parameterized walks. The walk engine uses these walk parameters to calculate the ground path of each foot, how to move each foot through the air between steps, and the order in which to move the feet [2].

There are a total of 54 parameters that fully describe each walk. The parameters are divided into two groups: per leg parameters, which apply to every leg and global parameters that apply to the whole robot. Table 1 lists several of the most important parameters.

1.2 Related Work

Surface detection is important for legged robots. Sinha and Bajcsy argue that robots should use their legs to characterize surfaces as well as traverse them [7]. The material properties of a surface determine how the robot should move on that surface. In particular, they determine if a surface is penetrable, by using a probe; how conformant the surface is, by measuring the amount of force

required to deform it; and they detect slip using an accelerometer on the robot's foot.

In addition to detecting slippage, accelerometers have also been used to generate kinematic models for robotic arms. Canepa et al [4] describe how accelerometers were used in conjunction with joint encoders to generate a model of the segment lengths and relative orientations of the joints of a 7-DOF robot arm.

Moving further afield, Veltink et al have examined using Bayesian methods on accelerometer data to classify rehab patient movements as either dynamic activities, such as walking or climbing stairs, or as static activities, such as sitting or standing in place [8]. Cakmaki et al were able to determine when a person was checking their watch by using statistics gathered on accelerometer data with the goal of saving power by turning off the display when it was unneeded [3].

2. SURFACE DETECTION

We introduce our surface detection algorithm and discuss the tradeoffs in its design. We present experimental results showing the robots performance discriminating between a cement floor, the carpet in our laboratory, and the carpet on a RoboCup field. Figure 1 shows accelerometer data from the two carpeted surfaces as an example; the algorithm must distinguish between these two classes of data. We present additional results showing that our surface detection method may be applied to other problems. Specifically, we use it to detect if a robot is freely playing soccer, bumping into a wall, or entangled with another robot.

2.1 Algorithm Details

Surface detection is a classification problem. We would like the robot to identify the surface underneath it based on accelerometer data as it walks. Since we need to know the surfaces in advance to label them, we also require sample data from each surface so that we can use supervised learning to train our classifier. This is not an unreasonable assumption, especially if the goal of surface identification is to chose a pregenerated set of parameters for movement on each particular surface.

Since the data arrives as a stream, we window it into discrete chunks when creating the feature vector for the classifier. Larger data windows increase accuracy at the cost of increased latency when detecting change. We chose to use a one second sliding window.

To cut down on the amount of data passed to the learner, we used statistics to described the

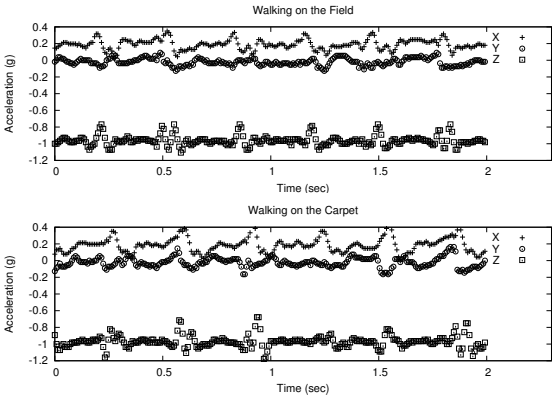

Fig. 1. Two seconds of accelerometer data gathered while the robot marched in place on the two carpeted surfaces. The acceleration along the x axis has a positive mean because the robot's stance leans forward slightly.

sensor reading distribution in each window instead of the data window itself as the feature vector. We used the variance in x, y, and z accelerations as well as the (x, y), (x, z), and (y, z) correlation coefficients over each window as the six features. The correlation coefficients were calculated as:

$$cor(x,y) = \frac{\sum_{i=1}^{n}(x_i-\overline{x})\cdot(y_i-\overline{y})}{\sqrt{\sigma(x)\cdot\sigma(y)}}$$

We chose to use decision trees for learning due to their simple representation and classification speed. Labeled training data was passed to the C4.5 decision tree learning program to generate the tree [6]. Every sensor frame when new accelerometer data becomes available, the sliding window is shifted and the decision tree is used to generate a new surface classification.

As an aside, it is possible to calculate each of the six features iteratively so that updates only require calculations involving the oldest accelerometer data point and the newest data point. Coupled with the speed of the decision tree this makes it practical to run the classifier at the sensor frame rate.

2.2 Experimental Results

To test our surface prediction algorithm, we gathered five minute segments of accelerometer readings while the robot marched in place on each surface. We created a decision tree using C4.5 and used 10-fold cross validation to quantify the accuracy of our final classifier. That is, we divided the data into ten 30 second segments, trained a decision tree using 9 of the segments and evaluated the tree on the remaining, unseen segment. We repeated this until each segment had been used to evaluate a decision tree trained on the other nine data segments.

Table 2. Surface Recognition Performance

Overall Performance	
Correct	84.9%
Incorrect	15.1%

Detailed Performance			
	Samples classified as:		
True Class	Cement	Carpet	Field
Cement	91.0%	3.1%	5.9%
Field	4.8%	81.1%	14.1%
Carpet	7.1%	11.7%	81.2%

Table 3. RoboCup Domain State Detection Performance

Overall Performance	
Correct	92.3%
Incorrect	7.7%

Detailed Performance			
	Samples classified as:		
True Class	Playing	Hooked	Wall
Playing	97.8%	1.8%	0.4%
Hooked	1.4%	92.4%	6.2%
Wall	0.3%	15.7%	84.0%

The evaluation results are presented in table 2. As we would expect, the classifier has more trouble disambiguating between the two carpeted surfaces as they are very similar and produce similar accelerometer data. See figure 1.

Table 3 shows the results when the classifier is trained to recognize different states that occur in a soccer game. The states considered are when the robot is freely playing, i.e. walking, kicking, and turning; when the robot is running into a wall; and when a robot is entangled with another robot. The classifier is less accurate when disambiguating between the two collision states than determining when the robot is freely playing.

3. VELOCITY PREDICTION

We trace the design of our velocity prediction algorithm from examining the initial data to choosing and evaluating a prediction algorithm.

3.1 Generating Velocity Data and Choosing KNN

Initially, we logged data from uniformly generated parameter sets. Since there are many more slow walks than fast ones, the median velocity of these random walks was biased toward lower velocities. To create a more uniform distribution of velocities, to aid learning, we manually created ten parameter sets with a widely spaced set of velocities. We added Gaussian noise to each parameter to generate walks for our trials. Figure 2 shows cumulative distributions of walk velocities

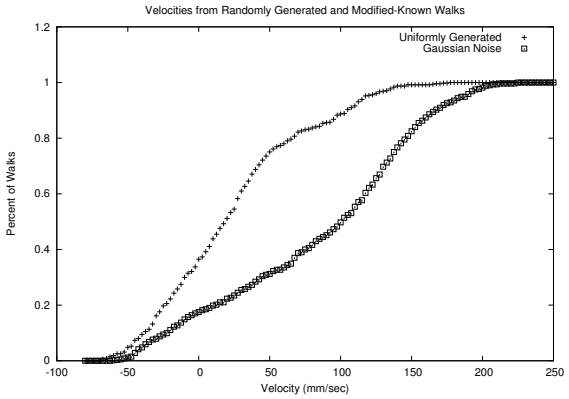

Fig. 2. Cumulative distributions of velocities from walks defined by randomly chosen parameters and walks created by perturbing a set of ten existing walks with Gaussian noise. Notice that the walks derived from the existing parameters are more uniformly spaced over the range of velocities.

for the purely random walks versus the velocities for the walks generated by adding Gaussian noise to ten human designed walks; the ten walks with added noise produce a more uniform set of output velocities.

While choosing a method for velocity prediction, we examined several statistics describing the distribution of sensor readins for each walk. These statistics are displayed in figure 3. There are few clear trends in this data. The mean x-axis acceleration increases with velocity, but it also has a high variance that limits its usefulness as a predictor. There is also a correlation between high variance in the sensor readings and low velocity, but some of the walks show low variance no matter what velocity they produce.

Due to the lack of clearly visible trends in the distribution of accelerometer data, we chose to use k-nearest neighbors for velocity prediction. The intuition is that since it is difficult to create a model to represent the data, we will use the data to model itself.

3.2 Applying K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a non-parametric method for function approximation. It depends on having a database of inputs along with the correct output for each of those inputs. When a query occurs, KNN finds the closest k input vectors in its database to the query input. It then merges the labels of these k closest points from the database to predict the correct output value.

KNN requires a few building blocks. First, it requires an input vector. In our case, we use a 63 dimensional input vector that consisted of

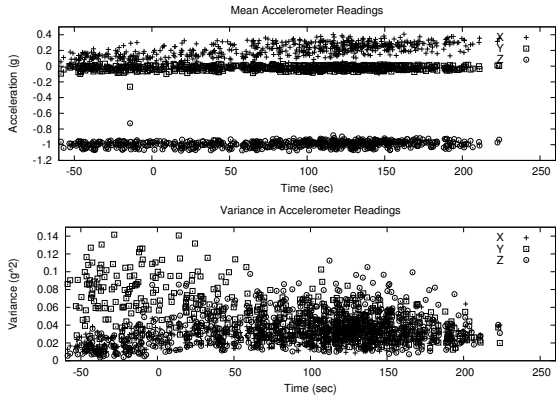

Fig. 3. Accelerometer data statistics for many different walks. The mean x acceleration shows a positive correlation with velocity but is very noisy. High variance is associated with low speeds.

the means, variances, and covariances of the accelerometer data for a given window while a walk was being tested. We added the 54 walk parameters to these statistics to form the complete vector. In other words, an input vector, $\boldsymbol{a}$ takes the form:

$$\boldsymbol{a} = \{a_1...a_9\}$$
$$\{a_1...a_3\} = \{\overline{x}, \overline{y}, \overline{z}\}$$
$$\{a_4...a_6\} = \{\sigma_x^2, \sigma_y^2, \sigma_z^2\}$$
$$\{a_7...a_9\} = \{\sigma_{xy}^2, \sigma_{xz}^2, \sigma_{yz}^2\}$$
$$\{a_{11}...a_{63}\} = \{54\ Walk\ Parameters\}$$

$\overline{x}, \overline{y}$, and $\overline{z}$ are the mean values of the data window. Similarly, σ_x^2, σ_y^2, and σ_z^2 are the variances of the accelerations. Finally, $\sigma_{xy}^2, \sigma_{xz}^2$, and σ_{yz}^2 represent the covariances between each component of the acceleration vector.

The database is a list of these feature vectors with a label. The label is the average velocity of the robot during the time window in which the statistics of the feature vector were calculated. In other words, $L = \{(\boldsymbol{a_1}, v_1), (\boldsymbol{a_2}, v_2), ..., (\boldsymbol{a_n}, v_n)\}$. The $\boldsymbol{a}$'s represent input vectors and the v_i's are the velocities associated with each input vector. n is the size of the library. Labels were calculated from the robot's localization while the training data was gathered.

With an input vector format and database defined, the next task is to define a distance function that maps from two input vectors to a real-valued heuristic distance between the two inputs. We chose to use the Euclidean distance between the vectors where the individual differences in the components are first divided by the standard deviation of that particular component in our database. Dividing each delta by the standard deviation of that component normalizes the components so that they are all weighted [approximately] equally in the final distance function. Writing out the equations, we have:

$$dist(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{n} \frac{(a_i - b_i)^2}{\sigma_{L_i}^2}}$$

$\boldsymbol{a}$ and $\boldsymbol{b}$ are the two input vectors being compared. For each component, we find the difference in that component and then divide by the standard deviation of that component (σ_{L_i}) in the library. Note that since we square the difference this is equivalent to dividing the squared difference by the variance of the component.

The final component of our KNN algorithm is a method to combine the k closest saved values into a single prediction. We used a Gaussian kernel to perform the combination:

$$w_i = \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot e^{(\frac{-d_i^2}{2 \cdot \sigma^2})}, i \in [1...k]$$
$$v_{pred} = \sum_{i=1}^{k} (\frac{v_i \cdot w_i}{\sum_{j=1}^{k} w_j})$$

Once the algorithm has been defined, the only thing left is to pick a value for k. Assuming that the value of k that performed best at predicting the velocities of walks in the database will also work well for predicting the velocity of unseen walks, we examined the median absolute error of predictions as k was varied. We picked the value of k that minimized this error metric after an exhaustive trial of values. For our particular experiments, we chose a value of 6.

3.3 Cross-Validation for Evaluation

Leave one out cross-validation is trivial to implement with KNN; simply remove the query point from the database before making each query. Figure 4 shows the results of evaluating the trial data using cross-validation. The data points are sorted according to their true velocity, as determined by localization when the data was gathered. The predicted velocity returned by the KNN algorithm is shown for each trial. The mean absolute error of these predictions is 26.4 mm/sec. The RMS error was 35.2 mm/sec. Given that the median true velocity of the trials was 100.4 mm/sec, the predictor shows a large amount of error.

Figure 5 shows a histogram of the absolute errors during evaluation. The majority of the predictions are within 20 mm/sec of the true value. So while the predictions would be a noisy source of feedback for walk optimization, they do provide useful information. Additionally, increasing the size of the library of known points may increase the accuracy of predictions. It took approximately 30 seconds for the robot to generate each point. Since it is possible to run several robots at once, it is possible to gather a large amount of data fairly quickly.

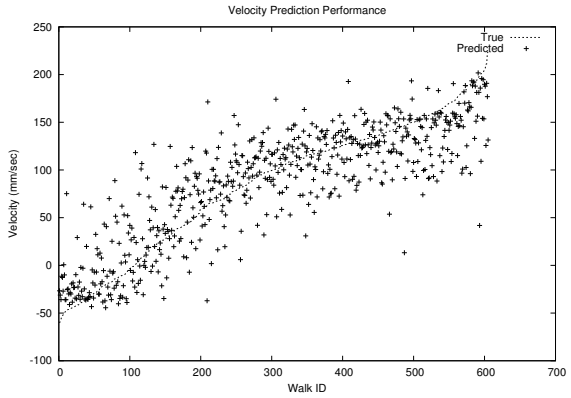

Fig. 4. A comparison of true walk velocity and predicted walk velocity, sorted by the true velocities. Negative velocities occurred when the robot dug its front feet into the carpet and pushed itself backwards while moving its feet forward.

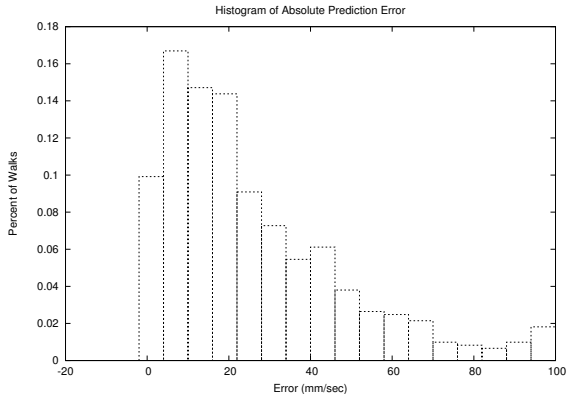

Fig. 5. A histogram of the absolute error when leave one out cross-validation is used to evaluated 605 sets of walk parameters.

4. CONCLUSION

We have shown that a legged robot can identify the type of surface that it is walking on by using accelerometer data. The classifier uses a set of six statistics to summarize windows of sensor data and passes these statistics as features to a decision tree. Due to the online calculation of the feature vector and speed of the classifier, it is possible to run our algorithm at the full sensor frame rate of the robot using a minimal amount of CPU cycles.

We also show that our algorithm for surface detection can be generalized to detect other state information. Specifically, we presented an example from the domain of robot soccer, where the state was one of playing freely, entangled with another robot, or bumping into a wall.

Finally, we presented preliminary results showing how accelerometer data can be used for velocity prediction using a k-nearest neighbors approach. While the velocity prediction were noisy, future work using a larger library of examples or a different weighting of features in the distance metric may help overcome this noise.

ACKNOWLEDGMENTS

The authors would like to thank Sonia Chernova, James Bruce, and Scott Lenser for sharing their AIBO expertise and good advice.

REFERENCES

[1] James Bruce, Tucker Balch, and Manuela Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of IROS-2000*, Japan, October 2000.

[2] James Bruce, Scott Lenser, and Manuela Veloso. Fast parametric transitions for smooth quadrupedal motion. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.

[3] Ozan Cakmakci, Joelle Coutaz, Kristof Van Laerhoven, and Hans-Werner Gellersen. Context awareness in systems with limited resources. In *Proceedings of AIMS-2002, Artificial Intelligence in Mobile Systems*, 2002.

[4] G. Canepa, J.M. Hollerbach, and A.J.M Boelen. Kinematic calibration by means of a triaxial accelerometer. In *Proceedings of ICRA-1994, the Internationall Conference on Robotics and Automation*, 1994.

[5] Scott Lenser and Manuela Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of ICRA-2000, the International Conference on Robotics and Automation*, April 2000.

[6] J. R. Quinlan. *C4.5 – Programs for Machine Learning*. The Morgan Kaufmann series in machine learning. Morgan Kaufman Publishers, 1993.

[7] Pramath R. Sinha and Ruzena K. Bajcsy. Robotic exploration of surfaces and its application to legged locomotion. In *Proceedings of ICRA-1992, the Internationall Conference on Robotics and Automation*, 1992.

[8] Peter H. Veltink, Hans B.J. Bussmann, Wiebe de Vries, Wim L.J. Martens, and Rob C. Van Lummel. Detection of static and dynamic activities using uniaxial accelerometers. In *IEEE Transactions on Rehabilitation Engineering*, volume 4, 1996.