Final Report

Eric Trautmann
COSC 134

June 2, 2009

1 Introduction

Typical obstacle-recognition systems on mobile robots use stereo-vision or laser ranging to perform classification. The Yeti Robot, a robot designed to perform medium range autonomous traverses in Antarctica and Greenland, is not currently equipped with vision or LIDAR, but has a proprioceptive sensor suite including 6-axis IMU, wheel encoders, wheel motor current, and GPS. The purpose of this project is to increase the mobility of the Yeti robot by enabling it to recognize obstacles in real time, so it can appropriately recover in the event that it becomes immobilized. Different discrete types of obstacles, like a step obstruction, steep slope, or deep snow, require different control strategies for optimal recovery. By identifying how the robot is immobilized, the robot can then apply an appropriate recovery strategy.

2 Obstacle Classification

There are several possible approaches one could take towards recognizing discrete obstacle classes using proprioceptive sensor data. The Yeti robot current samples its sensors at 20 Hz, providing relatively fine resolution over a time-history during an immobilization event. This project aims to prove the viability of obstacle recognition using time-series sensor data. There are several candidate methods to choose from, including Hidden Markov Models (HMMs), Linear Dynamic Systems (LDS), or a more classic system-identification or dynamic modeling approach that is more classic to robotics and engineering research.

The simplest approach is to use Hidden Markov Models, which model a sequence of observed data as being produced by a first-order hidden Markov process. HMMs are frequently used for different sequence classification tasks, including handwriting recognition, speech recognition, or gesture recognition [1, 2, 3]. The graph structure describing this model is shown in figure 1. This structure represents a *state space model*, familiar to roboticists and systems engineers.

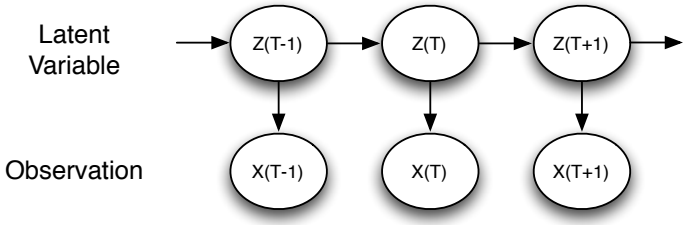

Figure 1: **Hidden Markov Model graph structure** An HMM models a sequence of observations X as being produced by hidden state variables Z.

The Hidden Markov Model can be used to model either discrete data or continuous data. Since we're attempting to perform sequence classification using our sensor readings, it makes sense to use a multivariate gaussian distribution or gaussian mixture model to represent the output data. HMMs assume that the hidden states are discrete, which is a gross simplification of the physical system we're trying to model. This is not a problem, however, since our main task is to perform sequence classification, and not recover parameters defining the dynamics of the robot and how it responds to inputs. If we wanted to model the latent variables as continuous, we could use Linear Dynamic Systems (LDS), though I've found that this is not necessary for the sequence classification task here.

A different approach is to apply system ID to recover the system inputs that create a given dynamic output. There are a number of algorithms appropriate for recovering model parameters from sets of input/output data, but this approach overcomplicates the task of classification and is unlikely to yield better results.

An HMM λ is represented by several parameters π, A, ϕ. π represents the initial state probability distribution, A represents the state transition probabilities, and ϕ represents the output distribution for each state. This

structure is shown graphically in figure 2

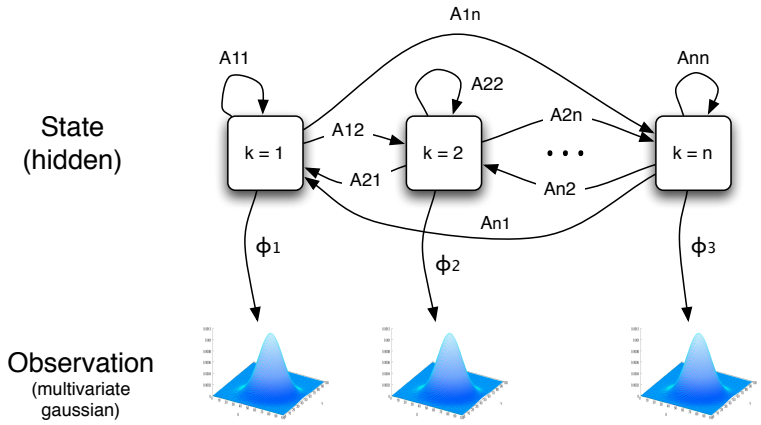

Figure 2: **Hidden Markov Model** States $k_1, k_2, \ldots, k_n$ have transition probabilities to successive states, and an output distribution ϕ that describes the expected distribution of output data for each state.

2.1 HMM Sequence Classification

There are three canonical problems associated with HMM's described in detail in Rabiner [2] and Dugad [4].

- Problem 1 - Given a sequence of observations $O_1, O_2, \ldots, O_t$ and a trained HMM λ, find the probability the model produced the observation sequence. This problem can be solved using the forward-backward algorithm.
- Problem 2 - Given an observation sequence, find the optimal state sequence that produced these observations. This problem is solved using the Viterbi algorithm.
- Problem 3 - Given a series of observations, optimize model parameters $\{\pi, \lambda, \phi\}$ to maximize $p(O, \lambda)$. A solution to this problem can be calculated using the Baum-Welch algorithm.

Our task of classification reduces to finding solutions to problems 1 and 3 using an approach similar to that proposed by Rabiner [2] for speech recognition. Luckily, there are established algorithms that efficiently calculate the solutions to the three canonical HMM problems. These algorithms exploit the Markov properties of the model to efficiently calculate optimal solutions rather than exhaustively enumerating all possible solutions and selecting the best. The forward-backward algorithm, the Viterbi algorithm, and the Baum-Welch algorithm are discussed in [2] and interested readers should refer to this source for detailed description and implementation details.

Classification is performed by training an HMM on multiple sequences from a given obstacle class, and test sequences are evaluated by selecting the model with the highest probability of producing the given output sequence. The training and evaluation steps are:

1. Train HMM λ using multiple sequences of training data for each discrete obstacle class. This corresponds to finding a solution to problem 3 described above.

2. Evaluate the probability that each model produced the test observation sequence in question. This corresponds to solving problem 1 above.

3. Select the class corresponding to the model with the highest probability of producing the given observation sequence.

In reality, the number of datasets available for classification is extremely limited. Ideally, we would like to compare data from three main classes of obstacles that the robot faces in a polar environment: step obstructions (sastrugi features), steep slopes, and deep snow. Unfortunately, not enough data was available for deep snow to be able to perform meaningful classification, so the validity of this method must be tested using step obstructions and slopes only.

Collecting data is extremely expensive, as each run can take up to an hour to collect and process. Nine runs worth of data are available for each feature. Data collection is discussed in more detail in section 3. Due to the small amount of training data, classification is performed by holding out one run from a set of features. An HMM for this obstacle is trained using the other eight runs, and the model representing the other obstacle is trained using all nine available data runs. The probability that each model produced the held-out sequence is computed, and the sequence classified based on which

model is more probable. This process is repeated for all eighteen data runs to produce a final classification score.

It is important to note the differences between this procedure and a more classical model selection problem. In this case, the models used for classification are not re-used between sequences. This is not the same procedure as leave-one-out cross validation, in which cross validation is used to select the best model. In this case, we're not trying to optimize model performance as a function of parameters, but instead want to gauge classification accuracy using a limited set of data.

2.2 Tools

This project has been implemented using data collected with the Yeti robot. Data were collected during runs in which the robot was driven manually while sensor data was logged using a CR3000 Datalogger. The collection of runs was imported into matlab, and processed into datasets using the procedure outlined in section 3.

The two major components of this project are: 1) data acquisition and data set preparation and 2) data processing and classification. Matlab is used to import, filter, scale data, and process data into cell arrays that represent a complete set of data for all runs. The Baum-Welch algorithm, forward-backward algorithm, and Viterbi Algorithm are implemented using an open-source HMM toolbox available under the GPL[5].

3 Data

Data were collected for nine runs in which the robot became immobilized on different step obstructions, and nine runs in which the robot was immobilized after it failed to climb a slope. Data were collected on two different days and on different steps and slopes. Six of the slope obstacle runs and three of the step obstruction runs were collected on 2/25/09 on man-made obstacles at the rugby fields north of campus, and the remainder of the runs were collected on 3/25/09 at a different field further north.

Conditions on 2/25/09 were below freezing with densely packed snow that was several days old. On 3/25/09, the temperature was above freezing, and the snow was heavy, wet, and slushy. This made it difficult for the robot to gain traction, which is perfectly acceptable for these tests. Due to

the difference in surface conditions, we would expect the signature from the IMU, the distribution of motor currents, and other features included in the feature vector to vary significantly between these two days.

Data were imported into Matlab and inspected to ensure that each of the sensors was working properly during each run. Runs in which sensors were not working or outputting spurious data were culled before adding data to the data set. Each of the nine runs collected for each type of obstacle represents a 'clean' data run with no spurious observations. It is an interesting matter for further study to consider the impact of false sensor data on classification, but has not yet been considered in this work.

Each sequence from a data run is intended to capture the class of the obstacle using data from immediately before immobilization. For simplicity, we consider each data run to be approximately two seconds worth of data preceding immobilization. This corresponds to a sequence length of 41 data points in each run. The exact indices of which data to use were selected by hand using a simple threshold scheme to define full immobilization. The end of a data sequence was tagged at five samples past the point where each wheel exhibited wheel-slip above 90%. Wheels slip for each wheel is defined as:

$$S = \frac{(R\omega - V)}{R\omega} \tag{1}$$

where R is the wheel radius, V is the longitudinal velocity of the wheel center, approximated by the optical velocity sensor, and ω is the angular speed of the wheels as calculated using differential measurements from the wheel encoders.

The data sets were constructed using all possible data that *might* be used for HMM-based classification. The classification results presented here use only the first six features, including optical velocity, IMU X-axis acceleration, and four-wheel motor currents. The remaining features are included in the data set for completeness and potential future work.

3.1 Data Preparation

Data are sampled at 20Hz using a CR3000 data-logger from Campbell Scientific. The wheel motor currents and IMU signals are first fed through a signal conditioning board that filters the signals using simple first-order RC filter with a cutoff frequency of 32 Hz. Unfortunately this board was designed

without this application in mind, and the filter cutoff is above the Nyquist frequency of 10 Hz. We can compensate for this by filtering the data in software using Matlab's discrete-time filtering function.

Each input feature is filtered using a basic moving-average filter with a window of 5 samples. It is trivial to implement a more sophisticated digital filter such as an FIR filter if different specifications are required for future work. The original data set is filtered prior to selecting the 41-sample sequences used for model training and evaluation. This ensure that all data in a given run are filtered identically, and the data at the beginning of the sequence are calculated using data values from several points outside the range. This approach is perfectly valid whether we choose to perform sequence classification on or off-line since filtering only uses past data points, in contrast to 'smoothing,' which incorporates future data points.

After filtering the data, each sequence is standardized by subtracting the mean value for each feature from each value. The data are then normalized by feature-wise variance such that the resulting feature sequences have similar dynamic ranges. Experimental results suggest that this normalization is required to avoid numerical problems during training.

The un-normalized data sets are presented in figures 5-8. These figures are produced as images in Matlab, which makes it possible to visualize each sequence and the relationship between features in the data. Figure 3 provides a key for interpreting each image. The images use pixel values scaled relative to the data within each image, so the same color does not necessarily represent the same data value in two different images.

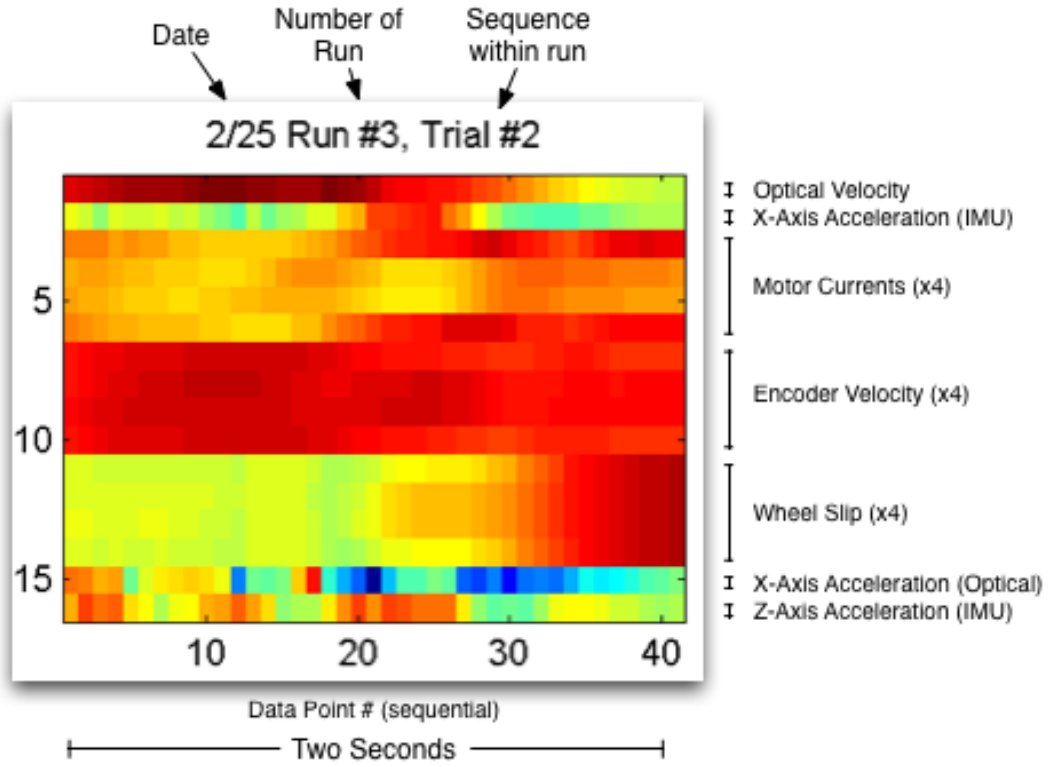

Figure 3: **Example Data image** - The purpose of imaging data this way is to observe the relationships between features for each type of obstacle. Colors are relative to each individual image and not representative of absolute data values.

From these data visualizations, we can see that clear patterns in the step and slope obstacles, suggesting at a qualitative level that sequence classification is possible. Figure 4 presents an interpretation of these patterns based on an intuitive understanding of the robot dynamics through the two types of immobilization events.

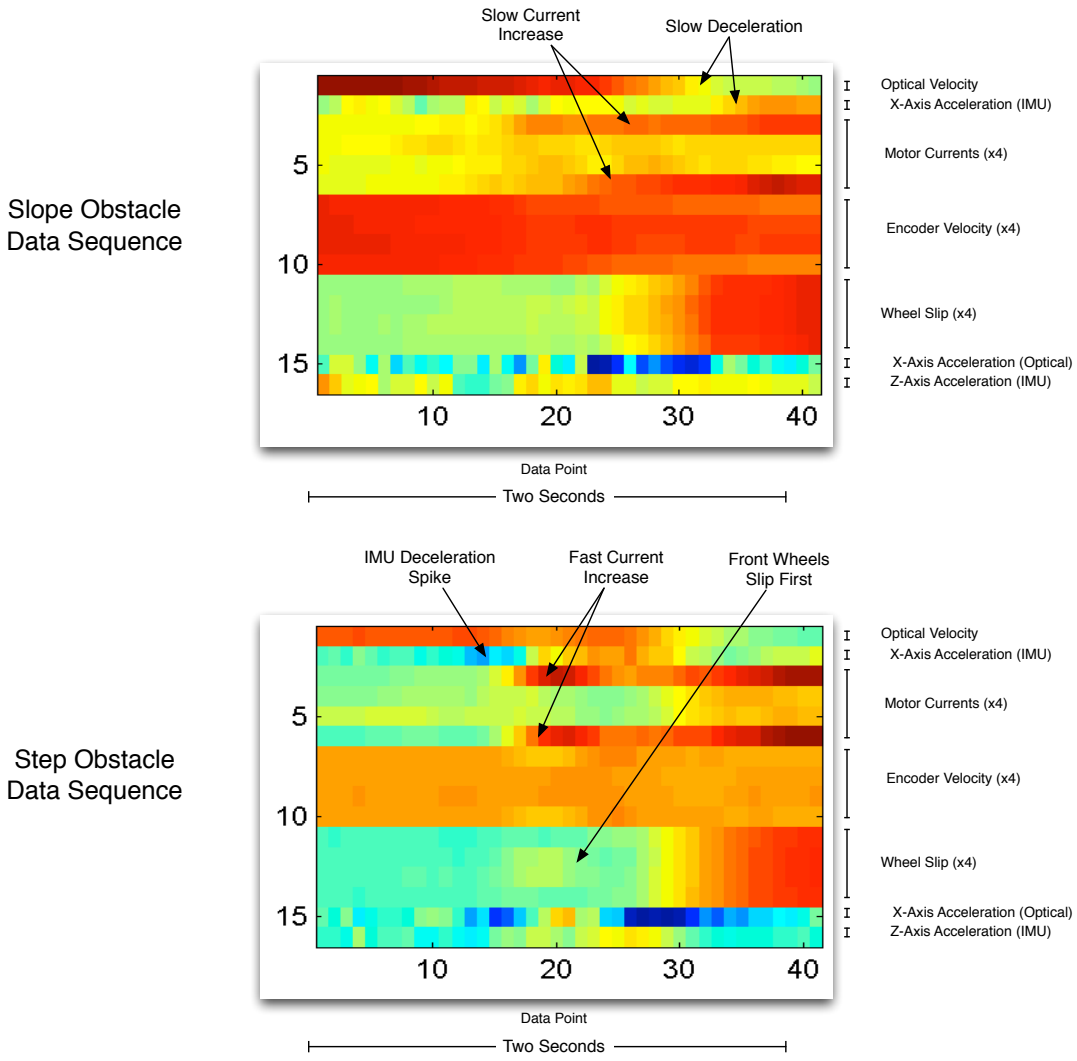

Figure 4: Intuitive visual pattern interpretation.

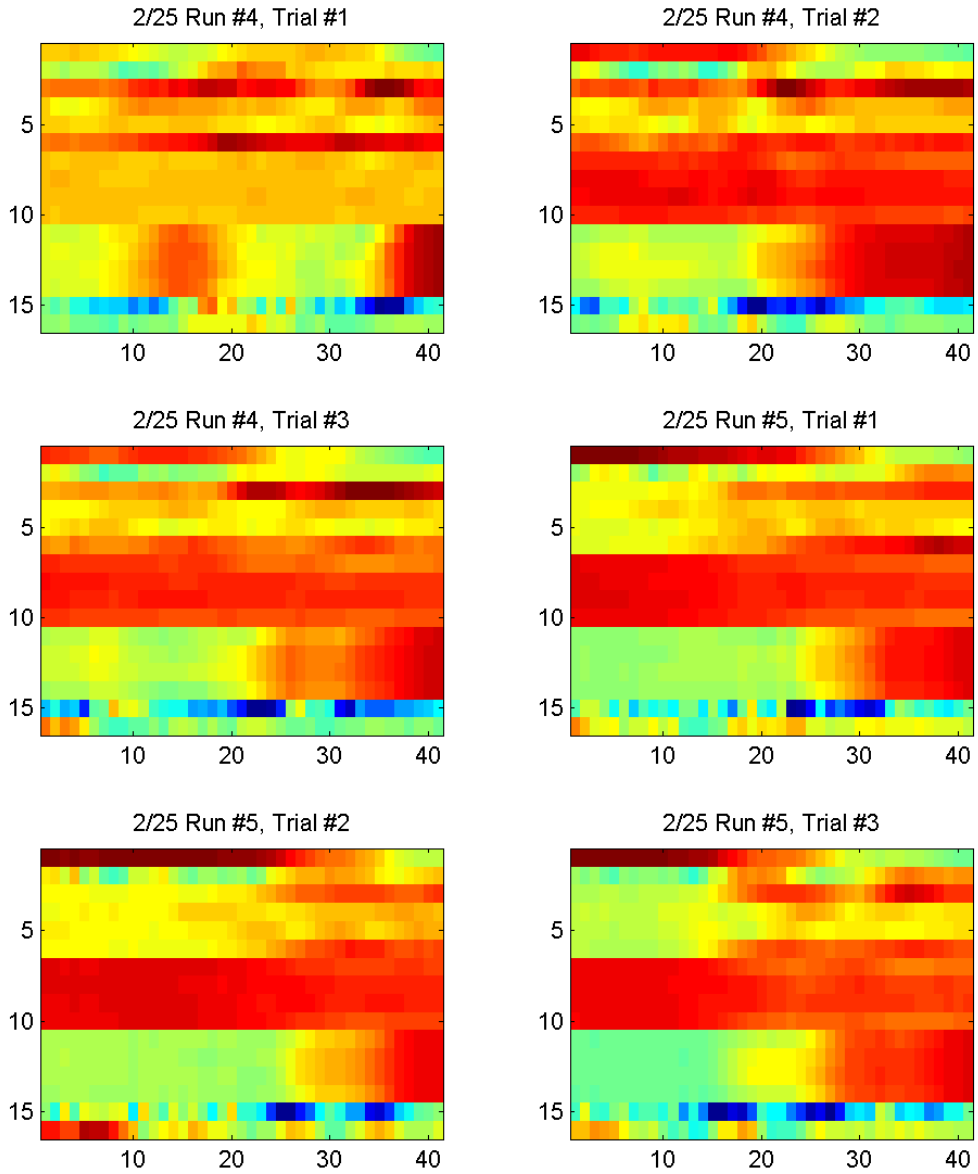

Figure 5: Immobilization events recorded on slope obstacles from 2/25/09.

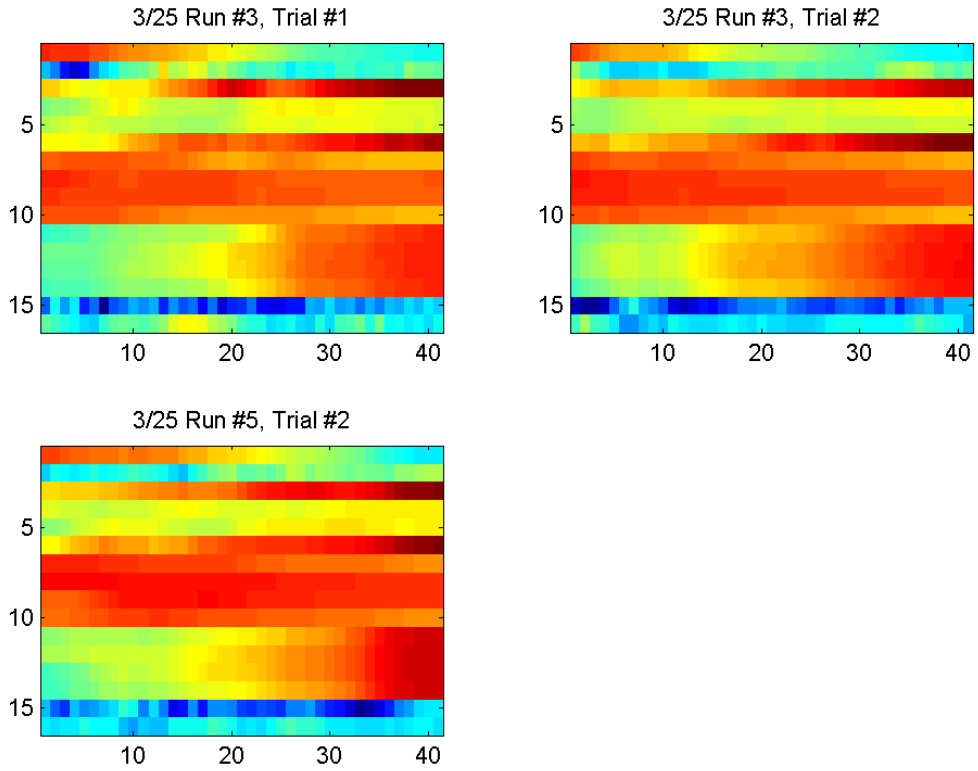

Figure 6: Immobilization events recorded on slope obstacles from 3/25/09.

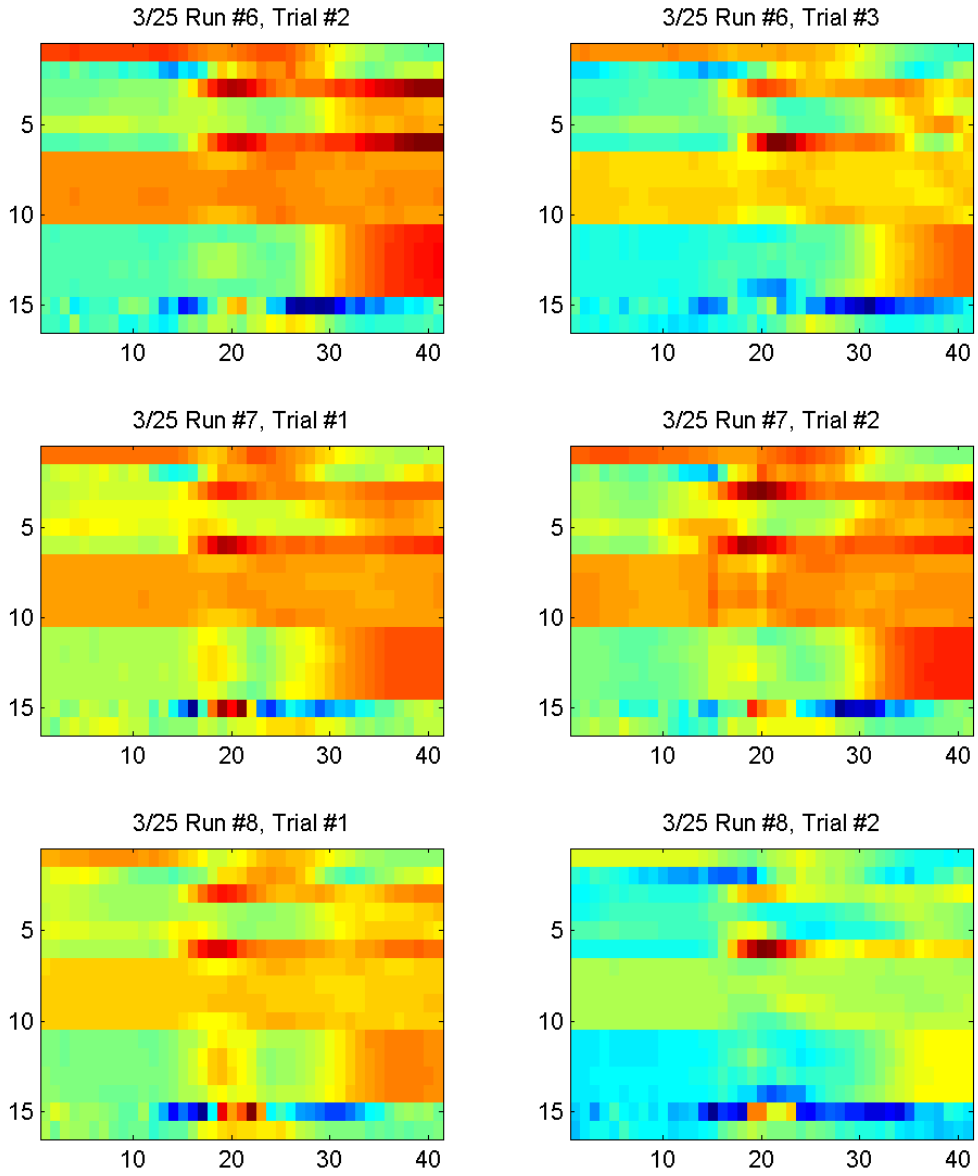

Figure 7: Immobilization events recorded on step obstacles from 3/25/09.

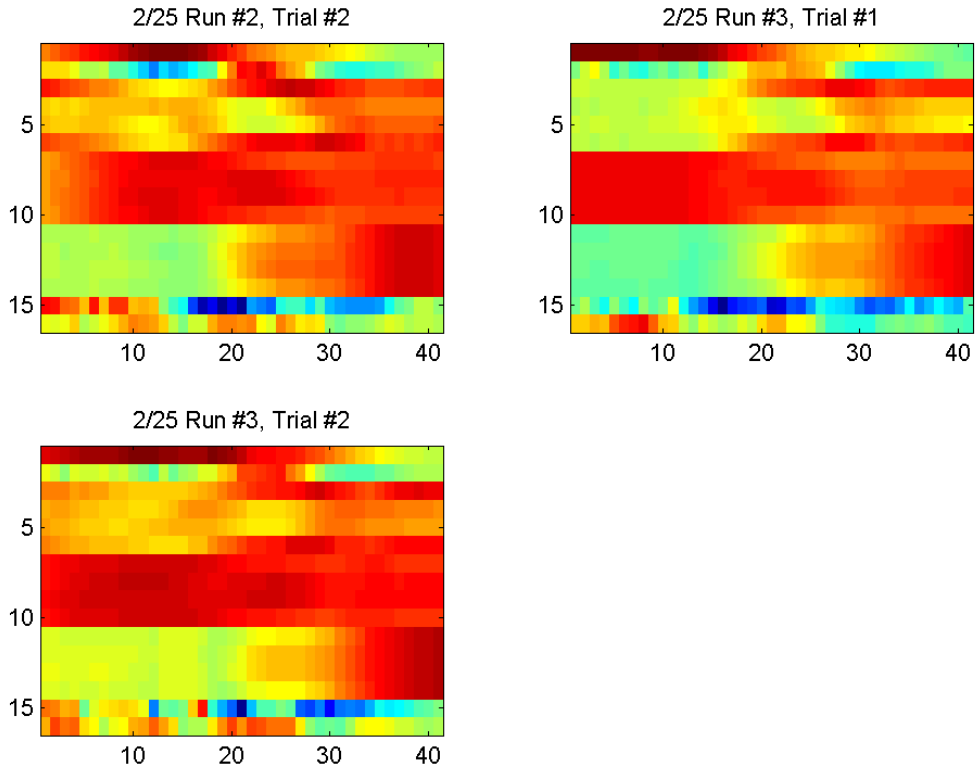

Figure 8: Immobilization events recorded on slope obstacles from 2/25/09.

4 HMM Sequence Classification

Several test were conducted using different combinations of data runs, feature vectors (subsets of the data presented above), and model parameters.

4.1 Classification Results

The main test was performed using a data set consisting of all nine slope obstacle trials and all nine step obstacles collected on 2/25 and 3/25/09. Each run sequence was classified by constructing an HMM using all other trials for that obstacle, while a second model was created using all of the available data for the opposite class. HMM's were constructed using six hidden states, though we have little intuition to guide our interpretation of what these hidden states physically represent.

Each held-out data sequence was classified according to the model that yielded the highest probability $p(O|\lambda)$. This process was then repeated for each of the 18 trials in the test. Only six features were used, including optical velocity, IMU X-axis acceleration, and four-wheel motor currents.

These six features likely contain the most informative signal and information useful for classifying obstacles. For example, the robot experiences different levels of deceleration as it contacts a step vs. a slope, and once immobilized, the robot's weight distribution and contact angle leads to differences in the distribution of motor currents among the four wheels for a given obstacle. The wheel slip measurements are slightly different for the two cases, but yield a smaller signal. The results here suggest that it is not necessary to include these features, though future work with a larger multi-class obstacle recognition problem may benefit by including these features.

This limited feature set yielded excellent classification results, shown in Table 1. All 18 sequences were correctly classified using the HMMs and overall classification accuracy was 100%.

Table 1: Test 1 - Confusion Matrix

Predictions\Targets	Step	Slope
Step	9	0
Slope	0	9

The same experiment was performed while varying different parameters, including the number of hidden states and the features used to determine the simplest model that accurately represents the data. Unfortunately, due to the limited number of training sequences, the granularity in the accuracy makes it difficult to evaluate different parameters.

I found that classification performance began to decrease as fewer than six features were used. I tried training models using just the two left wheels of the robot, ignoring the right wheels, and including velocity and IMU acceleration features. In this case, one of the 18 sequences was misclassified. Dropping more features from the set further decreased performance as expected. I also found that with fewer than six states, the classification accuracy decreased as one or two of the sequences were misclassified. More than six states does not hurt performance, but increase model complexity and is thus undesirable.

5 State Sequence Prediction

It is not necessary for sequence classification to calculate the predicted hidden state sequence for a given model and series of observations, but it is an interesting question to gain greater insight into the problem. Calculating an optimal state sequence requires us to define what we mean by optimal. In this case, it makes most sense to find a state sequence I that maximizes $p(I|O, \lambda)$. We can do this using the Viterbi algorithm, which is described in detail in [2].

Figure 9 shows the predicted state sequences for an HMM trained on step-obstruction data. We would expect that this model better represents the step-obstruction data, and that the predicted state sequences for step-obstruction data will more closely resemble each other than those sequences predicted for slope data. This is, in fact, what we see.

It's obvious from these plots that we placed no constraints on the form of the learned HMM. It would be possible to constrain the learned state transition matrix A such that the model learns a left-to-right structure in which states must progress sequentially. Many other alternative graph structures exist, but for this work, we start with the simplest possible model that meets our desired classification performance.

Figure 9 shows that the model learned a roughly left-to-right structure without an explicit constraint. While the hidden states don't correspond to anything that we can physically interpret, intuitively it makes sense that the

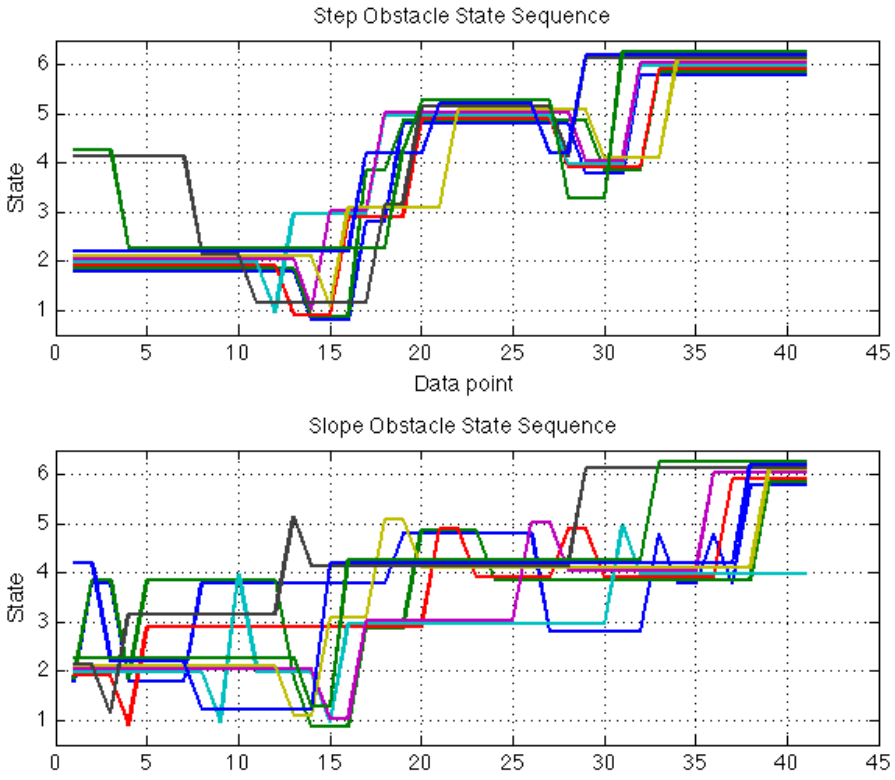

Figure 9: State sequence predictions for HMM trained on step obstruction data.

model would learn this type of structure since our data is a time-series from a physical system. We expect that the hidden state represents some aspect of the robot's position or internal state at a given point in time during the sequence of immobilization. In the case of gene sequence identification using HMM's, we wouldn't necessarily expect the same structure since the model doesn't represent a physical system as it evolves over time.

6 Discussion

These results, while based on relatively small data sets, strongly support the idea that HMMs can be used to perform obstacle recognition on a robot using proprioceptive sensor data. A system intended to be used online in a field-deployed robot must be trained on a library of more than two obstacles, and using more sequences than we had available for this work. These preliminary results, however, strongly suggest that this method is effective and can be generalized to other types of obstacles.

Unfortunately, the limited amount of training data means that classification accuracy is quantized into coarse levels of 5.56%, making it difficult to tune model parameters to improve classification accuracy. In this case, we have perfect classification performance, so there is little information we can use to improve our model or approach by constraining the state transition matrix, choosing more states, or using different features. Once more data becomes available, we can use standard cross-validation techniques to select an optimal number of hidden states or select fewer features.

It may be possible to use similar methods for other questions of interest to the robotics community, such as terrain classification. Current terrain classification methods typically use visual classifiers or calculations of wheel force vs. wheel slip to calculate terrain parameters. Using HMMs for terrain classification would require sensitivity to higher-frequencies, and a higher data sampling rate.

The next steps for future research are to collect a large set of examples of immobilization data from other obstacle types, including deep snow, low-cohesion 'corn' snow, wind-swept vertical ridges, and patches of ice. The results here suggest that this method can easily generalize to a large number of additional obstacles, but this needs to be verified in field testing. In addition, the algorithms currently implemented using Matlab need to be ported to Python or C for implementation in a real-time system.

References

[1] F Jelinek. Statistical methods for speech recognition. *books.google.com*, Jan 1998.

[2] L Rabiner and B Juang. An introduction to hidden markov models. *ieee assp magazine*, Jan 1986.

[3] J Bilmes. What hmms can do. *IEICE TRANSACTIONS on Information and Systems*, Jan 2006.

[4] R Dugad and U Desai. A tutorial on hidden markov models. *citeseerx.ist.psu.edu*, Jan 1996.

[5] Kevin Murphy. Hidden markov model (hmm) toolbox for matlab, 1998.