Learning-based Change Detection for Mobile Robots

Bradford Neuman

Note: This document is an extended abstract for my thesis. It contains the general idea used in the work, but not all of the data and results, which are still in progress.

1 Introduction

Mobile Robotics has advanced to a stage where robotic vehicles are beginning to navigate autonomously and make decisions about the traversibility of obstacles based on rich sensor data and machine learning and planning techniques. Unfortunately, these systems can still fail, especially in circumstances on which they were not trained. My goal is to create a technique which should improve robot safety and reliability by allowing robots to detect important changes in their environment. If a robot sees a given scene more than once and there is a significant change, such as a human being present or a tree falling, the robot should be able to detect the change and avoid it or alert a human. I am investigating machine learning techniques to perform change detection on a set of rich sensor data collected by a mobile robot.

In this paper I will be considering data from a feature-rich 3D perception system based on LIDAR (3D laser scans) and color camera images which are coordinated with a vehicle position. Change detection is thus the problem of detecting changes between the data from two runs of the robot through the same physical space. A significant change is hard to define precisely, but intuitively it would correspond to a real difference in the environment of the robot as opposed to the artificial differences in the data due to sensor noise, localization error, and other differences in sensing.

2 Motivation

Change detection is an important problem because it has the potential to dramatically increase the field-ability of a mobile robot. Robotics has developed to a point that it is possible to create autonomous vehicles which can successfully navigate in complex environments. For example, Boss, the CMU robotic entry to the 2007 DARPA Urban Challenge, utilized complex algorithms to identify and avoid obstacles [1]. This robot worked well enough to avoid failures (such as hitting an obstacle or disobeying a traffic law), but there is still too high a chance of failure. During development of Boss and similar robots, the autonomy systems are trained to detect a huge number of objects, but no amount of training can ever fully prepare a system for what it may experience in the real world, and the systems can still fail when they encounter something new and unexpected.

Consider, for example, a large robot which operates in a natural outdoor environment. The robot can handle driving through small twigs, leaves, and branches, so the autonomy system will learn that things that like look like small branches are not obstacles. This approach has worked well for avoiding obstacles like rocks and tree trunks while passing through light brush, but consider what might happen if the robot were to encounter a chain link fence. Assuming the system has never seen such a fence, it would appear to be made of small thin objects, much like twigs. The robot may thus decide to try to drive through the fence, which is unlikely to succeed. Detecting these kind of problems is known as novelty detection.

The idea behind novelty detection is that novel objects like the chain link fence should have some noticeably different characteristics from other objects seen. For example, the color and texture should be different from twigs, branches and leaves. If the robotic system can detect novel objects,

Figure 1: The modified John Deere eGator platform

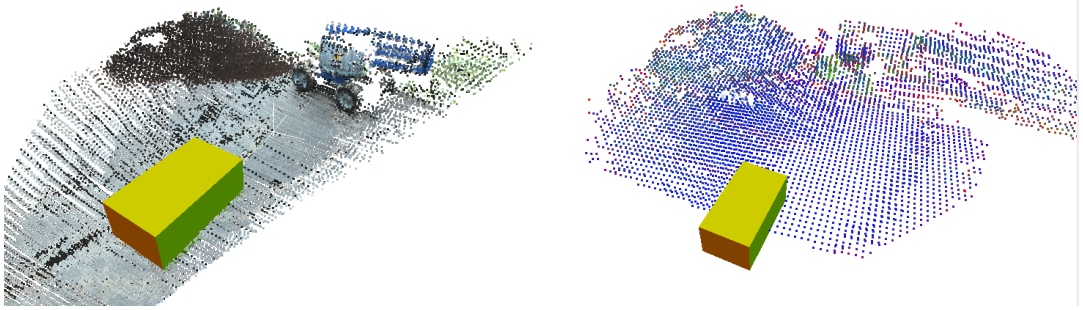
that is a sign that the system may not be well equipped to respond to them, and the robot should perhaps avoid the situation or ask a human for help.

Change detection also seeks to detect when there is a new object which a robot may not know how to handle, but it uses additional information based on data previously collected at the same location. The hope is that it will be possible to detect when something changes in the environment that may signify a dangerous situation. This approach could eventually allow a human to train the robot by leading it through an environment a small number of times to show it what is "safe." Then the robot can avoid any significant changes from the known safe environment.

3 Dataset

This data comes from a modified John Deere eGator in the rCommerce lab pictured in Figure 1. The robot is equipped with a SICK laser scanner and an HDR camera. The laser provides a 3D point cloud in front of the robot while the HDR (high dynamic range) camera produces output which is combined from images taken with different exposures in order to best illuminate the world. The system has an onboard computer and can be driven manually, remotely, or fully autonomously. The robot uses an advanced perception system which combines the sensor data into 3D point clouds. Each voxel is $20cm^3$ and can contain 30 different features including color as shown in Figure 2(a), spatial coordinates, and computed features like the texture and relative shape of nearby voxels. Useful features are computed using PCA on the spatial dimensions to determine shape. If points fit well along the first dimension of PCA they have a linear shape, if they fit within the first two dimensions the shape is planar and if it takes all three then the area is spherical (has points throughout the region). One set of these PCA features are displayed in Figure 2(b) where the shape is displayed by using red for the degree to which the region is linear, blue for planar, and green for spherical. The two images are of the same scene and you can observe that the wheels of the machine, for example, appear as mostly spherical while the ground is almost entirely planar.

The robot collected data for this project in static outdoor environments. The vehicle was manually driven through scenes consisting of both natural and man-made objects. I labeled this output with 27 classes wherever I thought I saw cleanly identifiable objects (those with no parallax issues and no bad noise issues). It is important to note that is not my intent to create a classifier capable of distinguishing those 27 classes. Instead, I use the labeled data to create examples of voxels in the same segment (those in the same class and near each other) vs. those in different segments (voxels in different classes).



(a) colorized points (b) PCA colored points

Figure 2: pointcloud visualizations

The initial data collected and labeled will be used to train the voxel-based classifier offline. We also collected a set of data in groups where the robot was driven over the same course multiple times with various changes made between the runs. This is the data that will be used to train and evaluate the entire change detection pipeline.

4 Problem Representation

The problem is:

> *Given the data from two runs of the same physical location, detect and flag any significant changes in the environment.*

In this problem, "significant change" is very difficult to define. Our general approach in this paper is to consider significant any change which may represent a case which requires human involvement. This includes critical changes such as the presence of a human (perhaps in place of some other obstacle), but also includes other changes in the environment. For example, in a factory setting, a change in the position or orientation of large factory equipment may represent a safety hazard. In general with this work we strive to create a solution that is flexible in the definition of a change. In this way, different training data could be used to achieve different levels of sensitivity. In the final version of this change detection system we plan to have a process where a human can lead a robot through an environment a small number of times and tell it which changes are acceptable. Each time the robot would discover a change, a human operator could actively tell the robot if that was significant or not and the robot could update its models. For now, to simplify the problem, we are not allowing this on-line training, but we will avoid approaches which would make it impossible.

This problem can be defined and broken down in many ways, and a large part of my work involves determining how to frame the problem. The naive way to simplify the problem is to consider each log on a voxel-by-voxel basis. That is, consider both logs one voxel at a time and compare each voxel to the corresponding voxel in the other log using some sort of classifier which works on pairs of voxels. This approach was tested, but we found that it gave unreliable results. This is not surprising because there is a lot of noise at an individual voxel level. It is also very hard to match up the voxels in a significant way. For example, if the robot is driving at slightly different speeds the density of points it scans as the laser sweeps across the ground will differ. This means that even without sensor noise or small changes in the environment, the data will differ.

There are several possible ways to improve upon the naive voxel-by-voxel approach. One would be to use the voxel based approach in a graphical model. This may help to smooth out some of the noise but may not be sufficient at determining meaningful change on a larger scale. The approach we consider is based on the idea that the things which change in a scene are objects. It is probably not important if a few voxels randomly look different, but if entire objects have voxels which

look different, that likely represents a change. Our approach is to first split the current scene into segments, and then compare each of those segments with what the robot has previously seen.

4.1 Sub-problem: Scene Segmentation

Since we will compare voxels segments at a time, the way the scene is segmented greatly impacts the final result. Thus, a significant portion of my effort was spent developing the segmentation methodology. The basic approach is to consider pairs of voxels in a scene and use a classifier to determine if the pairs lie in the same or two distinct segments.

The sub-problem is:

> *Given the 'distance' between two voxels which are neighbors in space, are they in the same object or not?*

This is a binary classification problem. Also, since the result should be the same for points (a, b) and (b, a) the distance function should either be symmetric or the learning algorithm should be able to classify points into "near zero" and "far from zero" classes. The distance between two voxels is the output of some difference function $f : (x, y) \rightarrow z$, which takes the vectors x and y as input and outputs a difference vector z which represents the distance between x and y in some metric. This output vector can have any dimension with regard to the input vectors and can even be a scalar in the case of functions like euclidean distance.

Because we use combinations of two voxels to create learning features, there are a large number of possible datapoints. To prevent over-fitting the test data, we reserve 20% of the voxels to be used for testing only, and all training examples come from difference functions computed on only the training voxels. The test accuracies reported throughout this paper are the percentage of times the classifier correctly determined if a pair of points from the test data had the same or different labels.

5 Methods

The following is the pipeline we experimented with for change detection. Note that some of these steps may not be helpful and may be left out in the final algorithm. Each of the steps will be described in more detail in the full paper.

Training:

- Read logs labeled with classes for voxels.
- Create pairs of voxels which are in the same or different classes.
- Run a difference function (see Table 1) to create a feature vector for the pair.
- Run a classifier with the feature vectors labeled with whether the pair was part of the same class or different classes.

Change Detection:

- If running from a log, replay it in real (or slowed down) time
- Collect voxels for a timestep (0.5 seconds)
- Iterate:
  - Select a seed voxel by choosing random voxels and running the voxel classifier trained above, selecting as the seed the voxel that is most "same" compared to the other voxels using classifier confidence values.
  - If the sameness score is below a threshold, the segmentation is complete for the scene and all remaining unsegmented voxels are just considered noise which is not part of any segment.
  - Randomly sample points to compare to the seed
  - Run the classifier on the difference function used in training applied to the seed and the sampled points

- Create a Markov Random Field that represents a neighbor constraint (neighboring voxels are likely to be "same") and a classifier constraint (from the result of the voxel-based classifier)
- Solve the MRF with smoothing paramater γ.
- Flood fill the resulting binary classification starting from the seed voxel to create a segment.

- For each segment, compare the voxels to the voxels previously seen at the same location
- If the difference is above a threshold, mark the entire segment as a change

Table 1: Difference functions

Function	Comment	SVM Test Accuracy
$\lvert x-y\rvert$	Absolute value	
$\{\lvert x-y\rvert, x+y\}$	Absolute value concatenated with sum	
$\lvert\lvert(x-y)\rvert\rvert$	Scalar euclidean distance in the raw feature space	
$\lvert\lvert\texttt{proj}_{\texttt{PCA}_4}(\texttt{x}) - \texttt{proj}_{\texttt{PCA}_4}(\texttt{y})\rvert\rvert$	Scalar distance in first 4 PCA dimensions	
$\lvert\lvert\texttt{proj}_{\texttt{MDA}_4}(\texttt{x}) - \texttt{proj}_{\texttt{MDA}_4}(\texttt{y})\rvert\rvert$	Scalar distance in first 4 MDA dimensions	
$\frac{x \cdot y}{\lvert\lvert x\rvert\rvert\ \lvert\lvert y\rvert\rvert}$	"cosine distance" scalar	

5.1 Voxel-Based Classifiers

Note to reader: The numbers below are out of date. The actual best SVM classifier now achieves over 93% accuracy, but the text and plots below have not yet been updated with the new numbers.

5.1.1 Boosting

I tried using boosting with decision stumps on this data because it can handle non linearity in the data. I tired several of the difference functions including a standard difference (no absolute value). After 120 iterations of the adaboost[2] algorithm, the standard difference function achieved 73.15% and the absolute value difference concatenated with the sum achieved 66.56%. Other difference functions did not produce significant results.

In order to verify the convergence properties of the algorithm I plotted the test and training error based on the number of iterations in Figure 3.

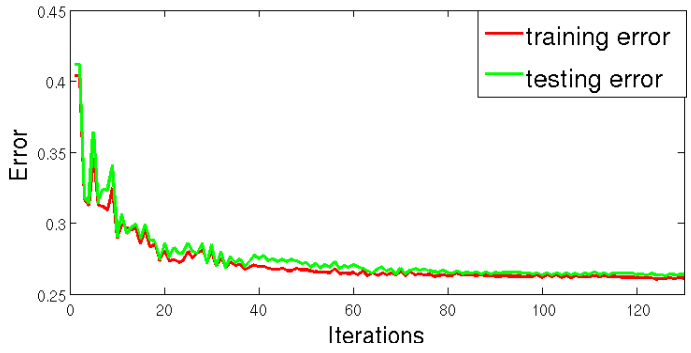


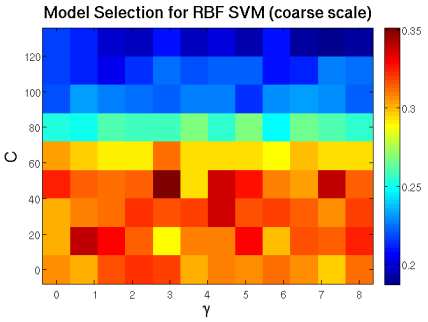
Figure 3: Adaboost error

5.1.2 SVM

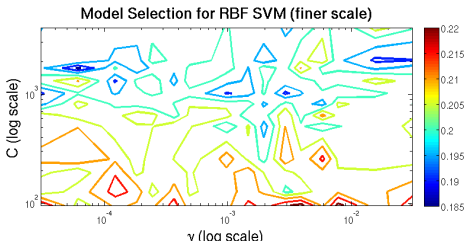
I had the most overall success using an SVM to classify differences. I tried using several kernels and found that the RBF kernel has the best performance. This method has two parameters: C, the slack variable, and γ, the kernel width:

$$k(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$$

In order to tune these parameters I ran a grid search over values of C and γ and zoomed in the search manually three times to settle on the values of $C = 212$ and $\gamma = 0.0029$ which minimized test error. This is the value which minimized the error for each of the best four difference functions. I generated the errors by running the classifier with separate test and training data 3 times at each zoom level, first generating the coarse plot (Figure 4(a)) then and intermediate grid, and finally the finer plot (Figure 4(b)).



(a) Coarse scale parameters. The C values here are divided by 10 and the γ values on the x-axis are a negative log scale where 0 corresponds to 10^{-8} and 8 corresponds to 10.



(b) Fine scale parameters plotted on a log scale

Figure 4: SVM paramater tuning

The best accuracy from the SVM methods was using the above parameters with the absolute value difference and sum difference function, which achieved 83.95% accuracy. The plain absolute value difference produced 80.61% while the norm-ordered concatenated pairs got 74.52% and the norm-ordered difference gave 78.55%. The other difference functions produced poor results with lower than 60% accuracy.

Note: This references section only covers some of the things written about in this extended abstract, and is not a complete list of references for the project.

References

[1] Chris Urmson et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 2008.

[2] Robert E. Schapire. *The Boosting Approach to Machine Learning: An Overview*, 2001. AT&T Labs - Research, Shannon Laboratory.

[3] Jyrki Kivinen, Alexander J. Smola, and Robert C. Williamson. Online learning with kernels. *IEEE Transactions on Signal Processing*, 2010.