Segmentation-Based Online Change Detection for Mobile Robots

Bradford Neuman

CMU-RI-TR-10-30

August 2010

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

As mobile robotics continues to advance, we are beginning to see intelligent robots with complex perception and planning systems onboard. These systems are often engineered for a specific task, or trained using machine learning to be adaptable and robust. Unfortunately, when faced with the complexities of the real world, almost every current robotic system will eventually encounter a situation which it has not been not trained or designed to handle. Unless a system operates in a highly structured or controlled environment, it will be impossible to determine all of the types of obstacles a robot may encounter. Instead of trying to make robots perfect perception and planning machines, we seek to enable robotic systems to detect situations in which the robot is unfamiliar. When a robot regularly visits an area more than once, we propose the use of a change detection algorithm to identify significant changes in that area over time and inform the robot about possible hazards. If a robot has traversed an area before, we can generally assume it to be safe, but if an unexpected change happens in the robots environment, this may represent a danger to the robot or the safety of humans nearby. We propose an algorithm to segment a 3D scene and use the segmentation as contextual information for an improved change detection algorithm. We will explain our proposed solution in detail and then provide data which show that the performance of change detection is increased by using segmentation. Finally, we will argue that these techniques can enhance the safety and reliability of mobile robots.

Contents

1 Introduction **1**
- 1.1 Detecting Changes in 3D Scenes . . . 2
- 1.2 Segmentation: Using Contextual Scene Information . . . 4

2 Related Work **5**

3 System Architecture **8**
- 3.1 Platform . . . 8
- 3.2 Perception System and Features . . . 9
- 3.3 Data Collection and Labeling . . . 12
- 3.4 Online Novelty Detection System . . . 12
- 3.5 Extension to Change Detection System . . . 15

4 Segmentation System **16**
- 4.1 Improvement to Change Detection System . . . 17
- 4.2 Segmentation Pipeline . . . 17
- 4.3 Similarity Classifier . . . 19
  - 4.3.1 Classifier Training . . . 20
  - 4.3.2 Boosting . . . 21
  - 4.3.3 SVM . . . 21
- 4.4 Difference Functions . . . 23
- 4.5 Seed Selection . . . 23
- 4.6 Global Segment Smoothness . . . 25
- 4.7 Segmentation Termination . . . 29
- 4.8 Parameter Optimization . . . 30

5 Results **31**
- 5.1 Evaluation Criteria . . . 31
- 5.2 Quantitative Results . . . 35
- 5.3 Analysis . . . 35
  - 5.3.1 Effects of System Parameters . . . 36
  - 5.3.2 Strengths and Weaknesses . . . 37
  - 5.3.3 Sources of Error . . . 39

6 Contribution **40**

7 Future Direction **41**

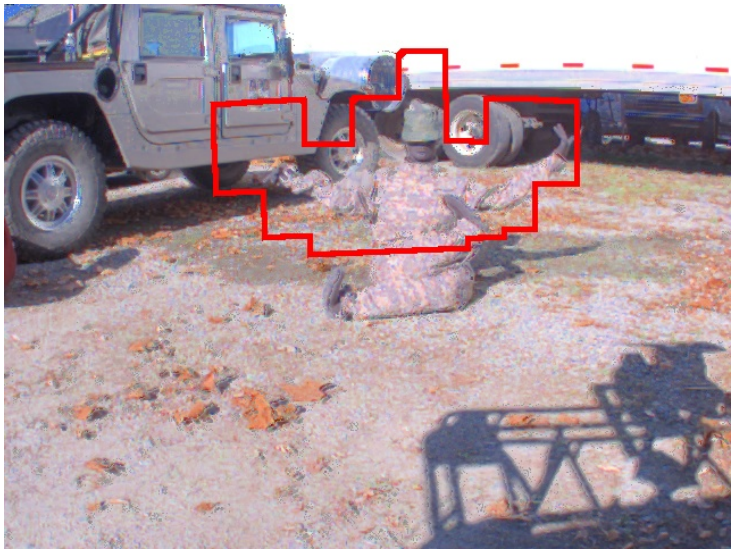


Figure 1: An example of the system detecting a manikin that was not previously present in this environment.

1 Introduction

As robotics continues to advance, mobile robots are becoming more and more impressive. In the past several years we have seen robots which can complete complex automated tasks robustly and without major incident. In many domains, such as urban [1] or outdoor [2, 3] navigation, one could say we have solved 95% of the problem of robustly and safely driving while obeying laws and safety constraints. However, the last 5% will be significantly harder to complete. As errors and potential pitfalls become more rare and complex, they become more dangerous because they are difficult to test and observe. Since these systems are heavily engineered for the environments they are intended to operate in, they can fail when encountering situations that are outside of their experience base.

Because of these limitations, robots which must operate in the real world must do so in very controlled environments where there is little uncertainty about the situations they will encounter. Robots like Crusher [4] are incredibly capable, but a human supervisor must still track the robot to monitor its performance and press an emergency-stop button if it is about to fall into trouble.

Due to the cost of failure of such systems, this lack of robustness to new situa-

tions is a primary reason why many autonomous systems in domains ranging from industrial and agricultural to military have yet to be fielded in the real world despite already impressive performance. Of particular concern is the issue of safety, since any harm to a human would be catastrophic to robotic applications.

This thesis leverages the fact that rather than developing an autonomy system that is 100% reliable, it is more practical to have a system that works well a huge majority of the time and is able to detect scenarios that are unusual and that the robot may be unable to handle safely. We can then detect potentially hazardous situations *before* they cause a serious failure.

In this thesis we explore change detection as a specific formulation of the problem of novelty detection. We examine the situation where robots will operate in an area that they had previously visited. Because a prime indicator of safety is the fact that the robot had successfully traversed a location at a previous time, detecting when something in the scene has changed is a good indicator of potential safety risks. When a change is detected, a human can be queried on the best way to deal with the situation or the robot can simply avoid the area entirely. This is common scenario in applications such as agriculture, warehouse management, and security patrolling.

The goal of this thesis is to improve on the functionality of an existing change detection system that compares each location in the scene to the same location as encountered previously. Just like when trying to classify an image using a fixed-scale sliding window, this approach discards potentially useful contextual information. By incorporating an online segmentation algorithm within the existing change detection pipeline, this thesis shows how considering an intermediate representation that captures more contextual knowledge about the environment can significantly improve the quality of online change detection for a mobile robot.

By enabling a mobile robot system to detect such situations, we hope to reduce the time it takes for such systems to be deployable in real-world applications.

1.1 Detecting Changes in 3D Scenes

Much of the theory behind working with 3D scenes comes from computer vision, where images are represented as 2D grids of pixels. In three dimensions, we image splitting up the world into boxes called *voxels*. Each voxel then can have a color specified in red, green, and blue channels (or some other representation). Other data, such as information about shape or other features, can also be associated with a given voxel.

Thinking of 3D scenes like analogs of 2D images does have several limitations. In a 2D image, there is no sense of occupancy; every pixel within a given range (the size of the image) has some color. In a 3D voxels, however, not all voxels will be occupied. There is a very important difference between a location occupied by a black voxel and a location that is not occupied at all. Because of this, it does not always make sense to think of edges between objects in a 3D point-cloud, since these objects may be separated by empty space. In images, we can consider a pixel to have either four neighbors (North, South, East, and West) or 8 neighbors, including the diagonals. In three dimensions this translates to 6, 16, or 26 neighbors.

Our goal in change detection is to detect change between two representations of the world: one historical and one current. We can represent each of these as 3D voxels and state out problem as:

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

In this problem, "significant change" is difficult to define. Our general approach in this thesis is to consider significant any change which may potentially represent a dangerous situation for the robot or any humans in the area. We also want the ability to flag any changes which may be of interest to human operators. This includes critical changes such as the presence of a human (perhaps in place of some other obstacle), but also 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 we strive to create a solution that is flexible in the definition of a change. This means we avoid heuristics about size or position of objects that might be significant (although we are limited on size because of the resolution of our perception system, explained in Section 3.2). In this way, different training data could be used to achieve different levels of sensitivity or to focus on different types of changes. Eventually we hope this system could be trained online to allow the user to specify the level of change sensitivity by driving around with the robot and showing it which changes are significant. For now, to simplify the problem, we are not considering this type of on-line training, but we will avoid approaches which would make it impossible. Since our primary motivation is to increase safety and reliability, our change detection system must run on-line and fast enough to notify the robot of changes in real time.

A straightforward way to approach the problem is to consider changes on a voxel-by-voxel basis. That is, consider both sets of voxels one voxel at a time and

compare each point to the corresponding voxel in the other log using some sort of classifier which works on pairs of voxels. This is the way the pre-existing change detection system worked. This approach was somewhat successful, but was very prone to false positives, like those seen in Figure 2(a). 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 will change the features created when the scene is binned into voxels. This means that even without sensor noise or small changes in the environment, the data will differ. Even in 2D image processing, considering two scenes one point at a time is rarely the best approach.

To improve the quality of this naïve voxel-by-voxel approach, we first consider a segmentation of the input scene which groups the voxels into distinct objects, and then use that to determine change. The problem of scene segmentation and its application to change detection is the focus of this thesis.

(a) Without Segmentation (b) With Segmentation

Figure 2: An example of change detection with and without segmentation

1.2 Segmentation: Using Contextual Scene Information

In order to provide context to algorithms which compare scenes, we consider the approach of scene segmentation. The idea of segmentation is to group each voxel in a scene into a distinct group. The goal is for these groups to semantically represent objects in the scene. For a given scene there are many logical ways to break things up into objects. For example, it is just as valid to segment a tree into

a trunk, branches, and a canopy, as it is to consider a tree as made up of hundreds of leaves and pieces of bark. What will differentiate these segmentations is their usefulness to our change detection problem.

We use segmentation as a way to inform change detection. When humans notice changes in their environment, they do not consider individual points one-by-one. Instead, they consider context and use a semantic understand of objects they see to determine change. For change detection, we consider important changes to be those which change an object, or introduce a new object into a scene. Therefore, we will look for changes on a segment-by-segment basis instead of voxel-by-voxel.

We can now update the statement of our problem from above to be:

> *Given voxels representing current and previous data at the same physical location, detect and flag any segments of the current scene which have changed significantly.*

This thesis will focus on the process of splitting the voxel data into segments. It is important to note that segmentation is not an end-result which we are seeking. Instead, we investigate segmentation as a means of improving change detection.

The segmentation problem can be looked at as learning a property between pairs of voxels in a scene. We want to look at neighboring voxels in 3D space and determine if they are in the same or different segments. This makes segmentation a binary classification problem. In order to apply standard machine learning techniques, we want a single data vector, $\mathbf{x}$, which represent a voxel pair, so we compute a *difference function* on the two neighboring voxels we are considering, which turns two voxels into a single feature vector. See Section 4.4 for a discussion of differences functions.

In this thesis we will present our methods and results for our segmentation based change detection system. We will explain how the system works and why it is a valid approach, and demonstrate experimental results that show that it is effective in improving the performance of change detection, as it did for the example in Figure 2(b).

2 Related Work

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 by being given many examples of correct actions or perceptions. Many other complex robotic systems have had large success with autonomous operation [2, 3, 5], but we have yet to see deployment of systems like these in the field.

One very common idea in robotics is that of a cost map, whereby the space of actions or paths the robot can take are given specific costs, and it is the goal of a planner to determine the optimal path as the path which minimizes the cost. Costs in mobile robotics often represent how much time or energy a robot will spend traversing a given area. They can also be used as hard constraints by assigning infinite cost to things like buildings or humans. Much of the research done to create robots like Boss is focused on how to derive these costs and how to select the best path given a constantly changing map of costs. Algorithms such as D* have been created, which can produce provably optimal paths given partial information, and update those paths as new information is discovered [6]. This means that robots can begin with an estimate of the world, and then update various costs as they move about.

Many planning approaches rely on machine learning to infer traversal costs based on sensor data. For example, we can imagine a classifier for a large robot which learns that small green things (grass or leaves) are traversible, while larger brown things (tree trunks or people) are not traversible. In these kind of supervised machine learning approaches, the result is highly dependent on the data set used for training. Consider, for example, a robot which is trained to detect vegetation and obstacles like humans or rocks. If this robot were to encounter a chain link fence, it would not have any directly relevant training material. The fence might 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. This kind of failure is even more important when human safety is involved. If a robot sees a human lying in camouflage, it might consider it a low obstacle and attempt to traverse over the human. While it is possible to add samples of humans lying in camouflage to the training data, it is nearly impossible to come up with enough training examples to reliably cover everything a robot will see in an unstructured and uncontrollable environment.

Instead of trying to train a robot to know what to do for every type of object it encounters, one approach is to detect situations which are not representative in the training data. This approach is called *novelty detection* because it seeks to detect

objects which are novel compared to what has already been seen [7].

We consider change detection as the problem of novelty detection applied on a per-location basis. Basically, we consider things to be changes if the voxels are novel compared to the voxels that used to be in that location. Previous work on change detection has often focused on detecting changes in images, such as detecting land use changes from satellite imagery. These techniques all required accurate image registration, meaning that the images needed to be properly lined up before the algorithms could run. The simplest of the methods was to use pixel differencing or similar methods, and apply a threshold value to detect changed pixels. More advanced methods used some vision approaches to reduce artificial changes based on glare or lighting conditions. Some specifically targeted vegetation by looking at various bands of wavelengths in the images that correspond to vegetation. The images were also compared by running PCA to compare the principal components between images. These techniques produced 60%-70% accuracy in detecting land mass changes such as the conversion of non-urban areas to urban areas. Techniques for this type of land-mass change detection were also considered which relied on separate classifications of the two images into types of land (urban, non-urban, seat, etc.), and then comparison on the classifier result, but these were not effective [8].

The problem of change detection has also been extensively researched in the signal processing domain. There are many approaches to detecting change which rely on a temporal stream of signal data which can be compared against either a model for that data or the previous history of the signal [9]. These filtering approaches are not directly applicable to detecting 3D scene changes for several reasons. First, they generally assume more than two data points for a given signal. If we think of the scene a robot sees as a very complex signal or group of signals, in change detection we want to detect differences between the signal at one point and the signal at another, without seeing a continuous stream in-between. Additionally, it is hard to model the data the robot receives as individual signals. There is no way to directly compare sensor readings, since they are all co-dependent (the laser scans rely on localization, etc.). It is not clear how we could represent the post-processed feature-rich point clouds as continuous signals in order to apply filtering techniques.

3 System Architecture

This section describes the mobile robot platform used for this research and provides a high-level description of it's autonomy system and data flow.

3.1 Platform

Figure 3: The modified John Deere eGator platform

This research was performed using the John Deere eGator platform shown in Figure 3 from the rCommerce lab at Carnegie Mellon supported by the Robotics Collaborative Technology Alliance. The robot is equipped with a variety of computing and sensors to allow it to perceive the environment and to form and execute intelligent plans. Additionally, it can be driven manually or through remote control, facilitating data gathering efforts.

Sensing is performed by a SICK laser scanner and a PointGrey Grasshopper camera. The SICK is fixed to a motor that nods it vertically with a fixed frequency allowing it to survey the environment. The SICK returns 180 readings at about 50 Hz, continually building a cloud of laser impact points in front of the robot.

The camera is fixed facing forward on the robot, covering about a 60 degree horizontal field of view. Due to the variable lighting conditions the robot will encounter in outdoor environments, the camera makes use of a high-dynamic range

(HDR) mode where information from pairs of images of varying exposure times are combined to better enable consistent and reliable camera-based features. The exposure times of these two modes are regulated by a controller that constantly fluctuates the exposure to maximize the percentage of the laser points that can be adequately exposed (not over or under-saturated). This allows the robot to simultaneously deal with a shaded and a well-lit portion of the scene, even though no single image would expose them well. An example output of the HDR camera system can be seen in Figure 4. Because the transform between the laser and the camera at any time is known, locations in space determined by the laser can be tagged with color and other camera-based features by projecting the point into each image and taking the optimal values.

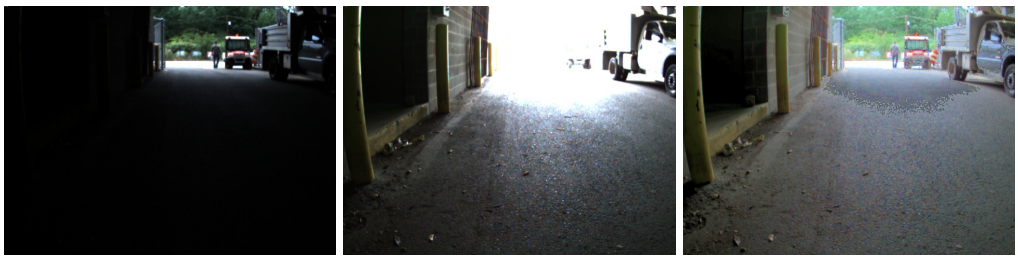


Figure 4: Example output from HDR camera system. An image of low exposure (left) is combined with an image of high exposure (center) to create a single image that is well-exposed for the entire scene (right).

The system estimates its position and motion at all times using a Kalman Filter aided by a GPS unit and an inertial measurement unit (IMU). This ensures that locally the position estimates are highly accurate even while the global position estimate can have errors of several meters.

The system has an onboard computer with a quad-core 3.2 GHz Pentium 4 CPU and 4 GB of memory that is shared between all onboard processes.

3.2 Perception System and Features

The eGator autonomy system is modeled closely on that of the Crusher vehicle from the UPI program [4] shown in Figure 6. Figure 5 outlines the high level data flow within the eGator autonomy system. The perception system assigns traversal costs by analyzing the color, position, density, and point cloud distributions of the environment [10, 11]. A large variety of engineered features that could be useful for this task are computed in real-time and the local environment is split into

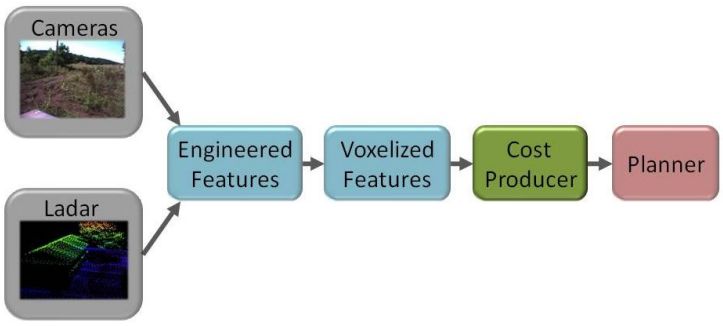


Figure 5: High-level structure of the eGator autonomy system.

Figure 6: The Crusher vehicle of the UPI program. A majority of the core autonomy system for our platform is derived from that of the Crusher autonomy system.

columns of voxels in order to capture all potentially relevant information. This vertical voxelization approach is effective for mobile robots since the presence of specific features at certain vertical positions is highly relevant to their impact on traversal cost. For example, solid objects at wheel height are likely to be small rocks while similar features higher off of the ground are more likely to be trees or man-made objects. Each voxel, tagged with its corresponding features, is passed to a cost producer. The system then interprets these features to create a final

traversal cost for that location in the world that can be used for path planning purposes.

(a) Image of scene

(b) Height-off-ground estimates

(c) Colorized laser data

(d) PCA spatial features

Figure 7: Example of features generated from the UGV's perception system. An HDR image for the environment is shown on the left. Laser data for the environment, colored using the onboard camera sensor is shown in the center image. Spatial features computed for the environment are shown on the right.

Each 20cm^3 voxel and contains up to 30 different features including color and texture-based features, spatial representation features, and camera-based features derived from image color and texture. Especially useful spatial features are computed by analyzing the eigenvectors of the distribution of points at each location [11]. Locations with one dominant eigenvalue (such as a pole) have a strong linear property, those with two dominant eigenvalues (such as the ground) have a

planar property, and those without a dominant eigenvalue have a more spherical distribution (such as bushes). An example of these color and spatial features is displayed in Figure 7. In Figure 7(d), the linear, planar, and spherical features are displayed in red, blue, and green respectively. You can observe that the vegetation has a much larger spherical feature while the ground is dominated by the planar feature.

An extensive log playback and processing infrastructure allows us to develop and optimize many of our system components off-line.

3.3 Data Collection and Labeling

We collected two types of data logs for this project in static outdoor environments by manually driving the vehicle through scenes consisting of both natural and man-made objects. First, we collected over 100 logs consisting of diverse scenes and objects. We labeled this output manually with 27 classes using a specially designed voxel labeling tool that utilized both image and laser data. This labeled data set was used by multiple modules in the system to perform supervised learning on various object categories. The labels consisted of many specific categories such as road, grass, big bush, small bush, tree trunk, car, pole, barrel, etc.

We also collected a second data set of 15 pairs of logs for testing the change detection system. In these change detection sets, we first took a baseline log and then made various changes to the environment and recorded changed logs. Some changes consisted of the additions of various man-made objects or rocks. Since human safety is a critical factor in robotics, many of these logs contained the appearances of manikins to represent a human that had wandered on a scene. The voxels that had changed in each scene were carefully labeled to quantitatively evaluate performance. There were logs collected from 7 unique scenes in total.

3.4 Online Novelty Detection System

This work builds upon the system's existing change detection system. We evaluate the impact of integrating a segmentation system into the change detection pipeline against this baseline system.

The system's change detection system is an extension of the novelty detection system described in [7]. The goal of novelty detection can be stated as follows: given a training set $D = \{\mathbf{x}\}_{1...N} \in \mathcal{X}$ where $\mathbf{x}_i = \{x_i^1, \ldots, x_i^k\}$, learn a function $f : \mathcal{X} \rightarrow \{novel, not\text{-}novel\}$. In the online scenario, each time step t provides an example $\mathbf{x}_t$ and a prediction $f_t(\mathbf{x}_t)$ is made.

The described system approaches the problem of novelty detection using a kernel machine where seen examples generate an influence of familiarity in feature space toward future examples (see Algorithm 1). All possible functions f are elements of a *reproducing kernel Hilbert space* $\mathcal{H}$ [12], meaning that all $f \in \mathcal{H}$ are linear combinations of kernel functions:

$$f_t(\mathbf{x}_t) = \sum_{i=1}^{t-1} \alpha_i k(\mathbf{x}_i, \mathbf{x}_t) \tag{1}$$

The novelty detection system therefore makes the assumption that proximity in feature space is directly related to similarity. Observed examples deemed as novel are therefore remembered and have an influence of familiarity on future examples through the kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$. A novelty threshold, γ, is specified and for each example $\mathbf{x}_t$, the algorithm accumulates the influence of all previously seen novel examples (line 5). If this sum does not exceed γ then the example is identified as novel and is remembered for future novelty prediction (line 7).

Algorithm 1 Online novelty detection algorithm from [7]

1: **given:** A sequence of features $S = (\mathbf{x}_i)_{1...T}$; a novelty threshold γ
2: **outputs:** A sequence of hypotheses $\mathbf{f} = (f_1(\mathbf{x}_1), f_2(\mathbf{x}_2), \ldots)$
3: **initialize:** $t \leftarrow 1$
4: **loop**
5: $\quad f_t(\mathbf{x}_t) \leftarrow \sum_{i=1}^{t-1} k(\mathbf{x}_i, \mathbf{x}_t)$
6: $\quad$ **if** $f_t(\mathbf{x}_t) < \gamma$ **then**
7: $\quad\quad \alpha_t \leftarrow 1$
8: $\quad$ **else**
9: $\quad\quad \alpha_t \leftarrow 0$
10: $\quad$ **end if**
11: $\quad t \leftarrow t + 1$
12: **end loop**

This approach found that especially if the number of features is large, it may first be necessary to project the high-dimensional input $\mathbf{x}_t$ into a lower-dimensional subspace more suitable for novelty detection using distance metrics. To deal with issues caused by features that are redundant, noisy or are dominated disproportionally by a subset of the training set, this system performs dimensionality reduction using *discriminant analysis*, which seeks transformations that are efficient

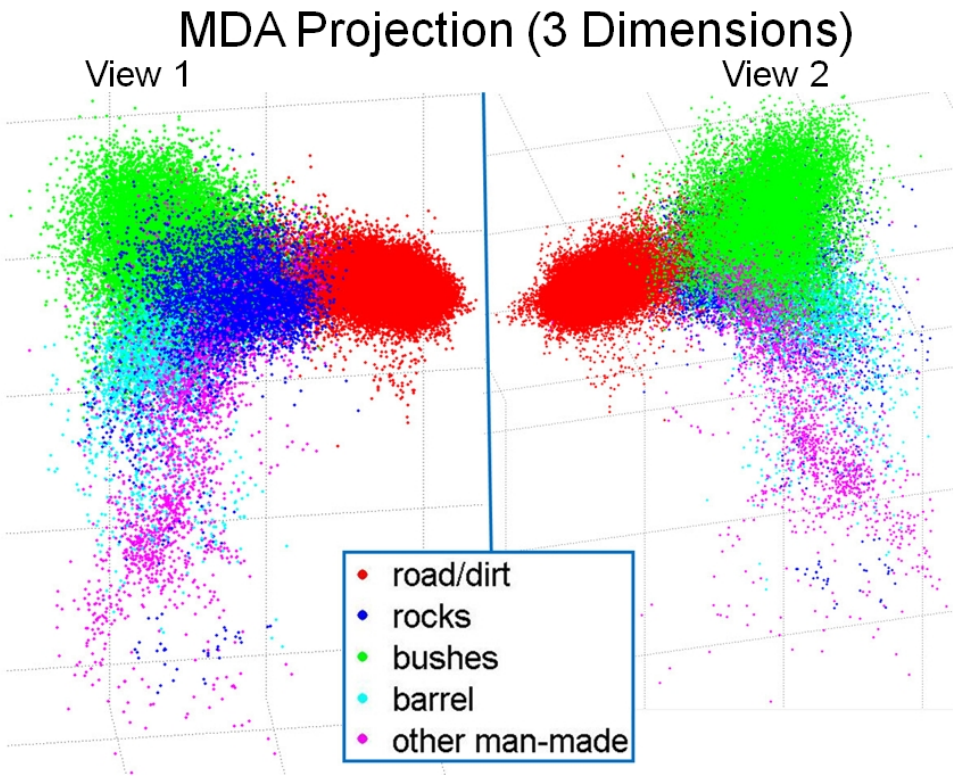


Figure 8: Labeled examples projected onto the subspace defined by the first three basis vectors computed by MDA. Only the first four classes were used to construct the subspaces ('other man-made' class was withheld as a test class). There is significant separation between the new man-made class and the known classes, implying a suitable subspace for novelty detection.

for discriminating between different classes within the data. Multiple Discriminant Analysis (MDA), a generalization of Fischer's linear discriminant for more than two classes, computes the linear transformation that maximizes the separation between the class means while keeping the class distributions themselves compact, making it useful for classification tasks [13]. Using MDA to construct a lower dimensional subspace using labeled classes not only optimizes for known class separability but likely leads to separability between known classes and novel classes.

Novelty detection is about encountering new classes, so by using discriminating ability as the metric for constructing a subspace, one can capture the com-

binations of features that make known classes novel *with respect to each other* and likely generalize to previously unseen environments, in effect capturing *what makes things novel*. An example of this dimensionality reduction technique can be seen in Figure 8.

3.5 Extension to Change Detection System

Change detection can be viewed as a location-specific version of novelty detection. Rather than trying to identify situations that are novel with respect to everything seen previously, a change detection system needs to identify when a situation is novel with respect to how the situation looked at an earlier time (from a stored log for example).

Algorithm 2 Online change detection algorithm (modification of Algorithm 1)

given: A set of voxel features $(\mathbf{x}_i)_{1...N}$ from the current navigation; a sequence of corresponding voxel features from the nearby locations from the previous run $S = (\tilde{\mathbf{x}}_i)_{1...M}$; a novelty threshold γ
outputs: A sequence of hypotheses $\mathbf{f} = (f_1(\mathbf{x}_1), f_2(\mathbf{x}_2), \ldots, f_N(\mathbf{x}_N))$ where $f_i(\mathbf{x}_i)$ specifies if voxel $\mathbf{x}_i$ has changed from the previous iteration
initialize: $i \leftarrow 1$
loop
  initialize: $y \leftarrow 0$
  $y \leftarrow \sum_{j=1}^{M} k(\mathbf{x}_i, \tilde{\mathbf{x}}_j)$
  if $y < \gamma$ **then**
    $f_i(\mathbf{x}_i) \leftarrow$ true
  else
    $f_i(\mathbf{x}_i) \leftarrow$ false
  end if
  $i \leftarrow i + 1$
end loop

Algorithm 2 shows a slight modification of the novelty detection algorithm shown in Algorithm 1 for use in change detection. Assume a log of all seen voxel features is available from a previous navigation of an environment (only the compressed representation is necessary since comparisons happen in the MDA-defined space rather than the raw feature space). For each seen voxel, the kernel function is used to sum the similarity between that voxel and all voxels near that

location from the previous navigation (line 6). If the example is novel with respect to those sets of voxels, then the system specifies that the voxel has changed from before.

This approach to change detection parallels a sliding window approach to image classification. Evaluating one voxel at a time works for many situations, but losing the ability to reason contextually about the scene leads to many false positives in the same way that classifying objects in an image is difficult when only considering one pixel at a time.

In the next section we will consider an expansion of this change detection algorithm which takes into account contextual information in the form of scene segmentations.

4 Segmentation System

The problem of segmenting data into meaningful regions has been extensively researched in the computer vision domain [14, 15, 16, 17], but previously existing 3D segmentation approaches are not applicable to change detection. One approach was to use graphical models to create segmentations based entirely on the locations of points in 3D space. This approach is limited in that it considers only spatial dimensions (not color or other features). It also requires knowledge of a "foreground" point known to be in the segment, which must be the point closest to the camera which is contained within the segment. Worse, it has a parameter that effects the radius of the captured segment, which is hard to adjust automatically [18]. Other work has been done to create basic classifiers from feature-rich 3D scenes. This work produced a method which could determine differences between vegetation, ground, and non-traversible obstacles. It performed well on the types of data with which it was trained, but is not effective for unknown environments [19]. More Recently, work has been done to automatically segment 3D point-clouds directly into objects. Some approaches [20, 21] try to directly detect known objects in the scene, but this is inappropriate for change detection wince our assumption is that the object is unknown. Other work tries to detect and analyze the shapes inside the point clouds [22, 23], which is not likely to generalize well to completely unknown objects. There has been work done to create a discriminative 3D segmentation technique which uses a subclass of Markov Random Fields that provided impressive experimental results [24]. Their approach is not entirely applicable to change detection, however. They assume that they having training data which contains representative objects, and then they use segmen-

tation to detect those same type of objects in point clouds. This approach will eventually fail in change detection when the robot encounters a new type of object which it has not previously seen. Our approach will be to build a classifier which can determine the similarity of objects, regardless of their class. This approach also uses only location information, while we will consider advanced features like color and shape.

4.1 Improvement to Change Detection System

We propose creating a new segmentation based change detection module which will first perform a segmentation of the scene and then use that information along with the voxel-based change detector described in 3.5 The segmentation module will add context information by looking at the perceived scene and splitting it into regions which represent entire objects.

In this section we will discuss the details of the segmentation pipeline and how it is integrated into the system. We will also discuss relevant parameters and how they have been optimized. See figure 9 for a graphical overview of this system.

4.2 Segmentation Pipeline

Segmentation-based change detection uses the voxel-based change detection algorithm (algorithm 2) currently present in the system in order to classify the individual voxels within a segment. Instead of performing the full scene segmentation simultaneously, we present an iterative approach. Since we do not want to train classifiers which are specific to types of objects, we seek methods which can train on the general differences between objects. Because of this, and some important optimizations described in section 4.6, we look at the problem one segment at a time. When finding a single segment, the problem is a two-class classification problem where one class represents voxels within the segment while the other represents voxels outside the segment.

Algorithm 3 is an overview of the entire pipeline for detecting changes using segmentation. Each major step is highlighted in a subsection below. The basic approach for each segment is to select a representative seed voxel and then optimize between the similarity of that voxel and other voxels in the scene and the constraint that segments are made up of neighboring voxels in a scene.

Algorithm 3 Segmentation-based change detection algorithm

given: A set of voxel features $(\mathbf{x}_i)_{1...N}$ from the current navigation; a sequence of corresponding voxel features from the nearby locations from the previous run $S = (\tilde{\mathbf{x}}_i)_{1...M}$; a novelty threshold γ; a segment change threshold η
outputs: A sequence of hypotheses $\mathbf{f} = (f_1(\mathbf{x}_1), f_2(\mathbf{x}_2), \ldots, f_N(\mathbf{x}_N))$ where $f_i(\mathbf{x}_i)$ specifies if voxel $\mathbf{x}_i$ has changed from the previous iteration
initialize: $\mathcal{S} \leftarrow \{\}$ (NOTE: $\mathcal{S}$ is a set which will hold sets representing indices of segment voxels)
loop
  $s \leftarrow \text{seed}(\mathbf{x})$ The result from Algorithm 4 in described Section 4.5
  if $s = -1$ or $|\mathbf{x}| = 0$ **then**
    $\mathbf{g} \leftarrow \Delta(\mathbf{x}, S, \gamma)$ Where Δ is the voxel based change detector in algorithm 2 described in section 3.5
    for each $\mathbf{z}$ in (S) **do**
      if $\frac{|\{g_i = 1 \ \ \forall i \in \mathbf{z}\}|}{|\mathbf{z}|} > \eta$ **then**
        $f(i) \leftarrow 1 \ \forall i \in \mathbf{z}$
      else
        $f(i) \leftarrow 0 \ \forall i \in \mathbf{z}$
      end if
    end for
  end if
  Swap x_1 and x_s
  Run the similarity classifier on the voxles in $\mathbf{x}$ as described in section 4.3
  Create the graph representing the MRF described in section 4.6
  Solve for $\mathbf{y}$, the labeling which minimizes the MRF energy function in equation 4 in section 4.6
  $\mathbf{x} \leftarrow \{x_i \ \ \forall i : y_i = 0\}$
  append $\{\{i \ \ \forall i : y_i = 1\}\}$ to $\mathcal{S}$
end loop

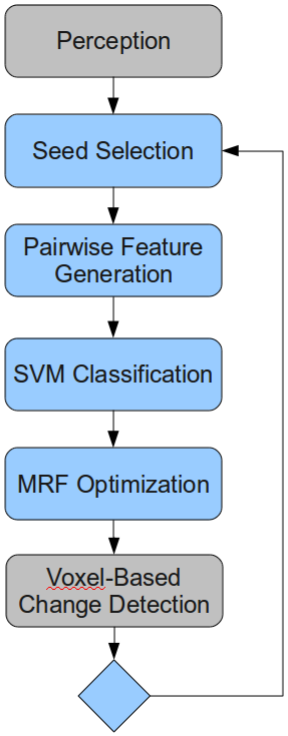


Figure 9: The control flow through the segmentation-based change detection system. The blue nodes are the steps added by this thesis

4.3 Similarity Classifier

We believe that voxels in the same segment should be noticeably more similar than those in different segments. In order to determine constraints on similarity between voxels we need some sort of classifier which learns what features make voxels similar or different. Since we are not seeking to build a specific object classifier, we need a general approach. In many machine learning problems, it is possible to train on exactly the same classes that the classifier is trying to determine. However, since the focus of change detection is to detect new and unprecedented situations, we cannot assume that we will provide a close example for everything the robot will encounter. Therefore, we use a radial basis function in a kernel method. Radial basis functions are homogeneous kernel functions, which means that they depend only on the difference of feature vectors they are comparing [25]. This means that we will learn based on the difference between a pair

of voxels which is similar and a pair which is dissimilar (remember that feature vectors for our classifier are results of a difference function computed on a pair of voxels, see Section 4.4).

Because of their efficiency, expressive power, and the applicability of kernelization, we chose Support Vector Machines as our similarity classifier (although we test another popular method in section 4.3.2). A support vector machine is a margin-based linear classifier which creates a hyper-plane in a high dimensional space to separate vectors of two classes. There are more general forms of SVM for regression or multi-class problems, but the segmentation approach only uses the traditional two-class version. The thing that is unique about SVMs is that they find a hyperplane which classifies the training data as correct *within a margin*. This means that there is a region around the hyper-plane which contains no datapoints. When an SVM is given a test datapoint, it simply determines which side of the decision hyperplane the point lies within. The SVM also gives us a confidence value which represents the distance from the hyperplane. If the test point is very near to the boundary, the SVM is likely to be less accurate than if the point is far away. This is because, during training, potential hyperplanes are penalized based on how far they incorrectly classify points past the hyperplane. This confidence value has units of distance in the higher dimensional space used by the SVM, so it cannot be directly interpreted as a probability [25].

The SVM approach can be extended with the use of kernel functions which allow much more powerful classifiers by extending the space of the feature vectors. In traditional SVMs, the hyperplane lies within the same space the feature vectors lie in. For example, if we were training on 2D data, the hyperplane would be a line. Using kernels, however, would allow the same 2D training data to create a hyperplane in a higher dimensional space, which would appear nonlinear when projected back down onto the original space, creating complex decision boundaries [12]. Our use of kernels allows us to learn complicated relationships between our feature rich data vectors which may have many dependent and redundant features.

4.3.1 Classifier Training

Our training data comes from logs which were labeled with categories, explained in Section 3.3. The data are labeled with 27 object categories, however, there is significant overlap between many of the categories such as large and small bushes. To deal with these, we merge several of these classes to end with 20 training categories such as paved roads, trees, bushes, barrels, buildings, etc.. We use

these categories to determine similarity, we are not seeking to develop a category classifier for voxels.

Because we use combinations of two voxels to create learning features, the number of training feature examples is exponential in the number of voxels in the labeled data. However, it is important to see that we want a good distribution of feature vectors, which are the output of some function $f(x, y)$, so the number of examples we need will depend on the dimensionality of the output of that function. To prevent over-fitting the test data, we reserve 20% of the original labeled voxels to be used for testing only, and all training examples come from difference functions computed on only the training voxels while test data comes only from the test voxels. The similarity classifier test accuracies reported throughout this section are the percentage of times the classifier correctly determined if a pair of points from the test data had the same or different labels.

The labels are created such that there are an equal number of sampled similar pairs as non-similar pairs. Similar pairs are defined as those which appear in the same log as neighbors and are tagged with the same category label. Non-similar pairs consist of two non-adjacent voxels labeled with different categories.

4.3.2 Boosting

Because this is a complex non-linear classification problem, we first attempted using the boosting technique. In boosting, we consider a weak classifier which is only required to have an accuracy greater than 50%. In our case we use decision trees of depth one (often called decisions stumps) as the weak classifier. In each iteration of boosting, results from multiple instances of the weak classifier are combined to create a stronger overall classifier. After 120 iterations of the adaboost[26] algorithm, we achieved 73.15% test accuracy.

In order to verify the convergence properties of the algorithm, we plotted the test and training error based on the number of iterations in Figure 10. This shows that the algorithm is working properly and that we have converged on a stable solution.

4.3.3 SVM

We had the most success using an an SVM to classify similarity, achieving 95% test accuracy for the similarity classifier. We use the Radial Basis Function kernel:

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

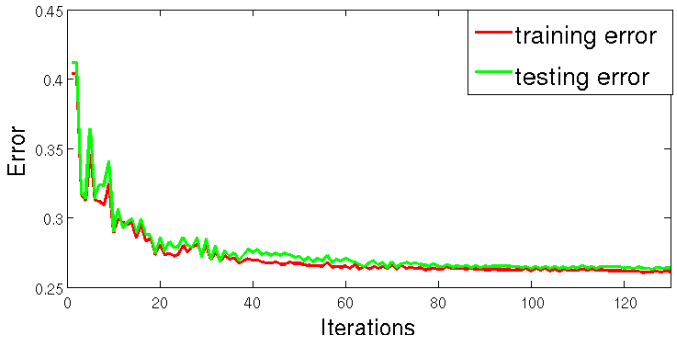
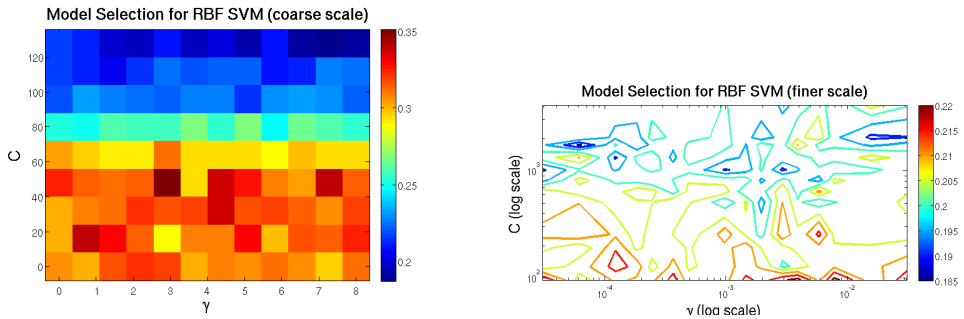


Figure 10: Adaboost error, zoomed in on y-axis

We considered other kernel options, but settled on RBF because of its performance and homogeneous properties (see Section 4.3).

This method has two parameters: The slack variable, C, and the kernel bandwidth, γ.

In order to tune these parameters we performed a grid search over values of C and γ and focused the search manually over three iterations to settle on the values of $C = 541.4$ and $\gamma = 0.27724$ which minimized test error to 5%. we 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 11(a)) then and intermediate grid, and finally the finer plot (Figure 11(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 11: SVM parameter tuning

For this work we use the kernelized SVM implementation provided by libsvm[27].

4.4 Difference Functions

Difference function represent differences between voxels. They are what allow us to train and evaluate the similarity classifier pairs of voxels by creating a single feature vector from a pair of voxels. Because ordering of voxels is arbitrary, e want to maintain the property that difference functions are symmetric. Difference functions can be traditional distance functions such as Euclidean distance between voxels in the feature space, or they can be more complicated vector functions which might preserve more of the original data. Using a scalar function like Euclidean distance basically reduces the problem to that of finding a difference threshold. Using a higher dimensional difference function allows us to use more complicated classifiers which can learn non-linear boundaries in the space of the feature vectors returned from the difference function. The difference functions we considered and the accuracies of similarity classifiers trained using them are reported in Table 1. The most naive of these functions is the absolute value difference, which simply subtracts the feature vectors of the two input voxels. This approach, however, loses information about the locations of the original voxels in feature space. For example, it might be the case that bright red voxels represent noise in the camera. If we take the absolute value of the difference in the red channel between a noise voxel and some other voxel, the number we get does not tell us about the original magnitude of the red feature. If we also store the sum of the two features, however, we have the ability to learn different properties of change for different regions of the feature space. We could learn that anything with a very high combined red channel feature is likely noise, while the same difference in redness for two more green objects does in fact represent a change.

4.5 Seed Selection

Because our voxel classifier works with pairs of voxels, we need a method by which to determine which pairs in the scene to evaluate. It is not computationally feasible to tun the classifier against all n^2 pairs in a log, so we approach the problem as one of picking a relevant seed and comparing all other voxels to that seed. When we come up with a segment based on this seed, we will remove the segment and choose a new seed from the remaining voxels.

In order to determine the quality of a seed, we compute a scoring function by evaluation the classifier on each pair in the clique of the k voxels we randomly

Table 1: Difference functions. The input dimension is n ($n \approx 30$ in our system) and the output dimensions listed are the dimensions of the features used in training. Since the scalar difference functions performed so poorly we did not try to optimize them, so it may be possible to achieve better results with those functions.

Function $f(x, y) = z$	$\dim(z)$	Comment	SVM Test Accuracy
$\lvert x - y \rvert$	n	Absolute value	91.97%
$\{\lvert x - y \rvert, x + y\}$	$2n$	Absolute value concatenated with sum	95%
$\lVert (x - y) \rVert$	1	Euclidean distance in the raw feature space	62%
$\lVert \mathtt{proj}_{\mathtt{PCA}_4}(\mathtt{x}) - \mathtt{proj}_{\mathtt{PCA}_4}(\mathtt{y}) \rVert$	1	Distance in first 4 PCA dimensions	68%
$\lVert \mathtt{proj}_{\mathtt{MDA}_4}(\mathtt{x}) - \mathtt{proj}_{\mathtt{MDA}_4}(\mathtt{y}) \rVert$	1	Distance in first 4 MDA dimensions	65%
$\frac{x \cdot y}{\lvert\lvert x \rvert\rvert \, \lvert\lvert y \rvert\rvert}$	1	"cosine distance"	54%

select as seed candidates. If the classifier confidence is positive, this provides evidence that the two seed candidates are more likely similar, while a negative confidence provides evidence they are different. Let x_1 be the seed of chosen seed candidates, and let $c(x_i, x_j)$ be the confidence of the result of running the voxel classifier described in section 4.3 on the pair of voxels x_i, x_j. One simple scoring function would be

$$s(x_i) = \sum_{x_j \in \mathbf{x}, i \neq j} c(x_i, x_j)$$

This function does not work well, however, because it penalizes seeds for being different. While we seek a representative seed, we don't want a seed which is the average of all the remaining voxels. Instead, we prefer a seed which is most clearly associated with one of the remaining unknown segments. Consider a case in which we have two strong segments remaining and several noise voxels. Our classifier may not be very confident about the relationship between a noise voxel and a regular voxel, so the score of the noise voxel may be a sum of small numbers. This score might then be the same as the score for one of the good seed voxels since that score would be the sum of high positive confidence terms for the voxels in the same segment and large negative confidence terms for the voxels in the other segment.

Instead of penalizing a seed candidate for being clearly different from other seed candidates, we simply ignore negative confidences to select the seed which is the most correlated with all of the voxels which the classifier thinks are in the same segment. Thus, our scoring function becomes

$$s(x_i) = \sum_{x_j \in \mathbf{x}, i \neq j} \max\{0, c(x_i, x_j)\} \tag{2}$$

Note that since we are randomly sampling to find seeds, the number of seeds we sample, k, has a large effect on performance.

The seed selection method is laid on in algorithm 4.

Algorithm 4 Seed Selection Algorithm

given: A set of unsegmented voxels **x**; a maximum number of seed candidates, k
outputs: The index of chosen seed voxel in **x**
initialize: $i \leftarrow 1$
x' $\leftarrow \text{shuffle}(\mathbf{x})$
for $i = 1$ to k **do**
    $S_{i,i} \leftarrow 0$
    for $j = 1$ to k, $i \neq j$ **do**
        $S_{i,j} \leftarrow \sum_{x_j \in \mathbf{x'}, i \neq j} \max\{0, c(x_i, x_j)\}$ (the score from Equation 2)
    end for
end for
Let $S_i = \sum_{j=1}^{k} S_{i,j} \quad \forall i \in [1:k]$
$s \leftarrow \text{argmax}_i \, S_i$
if $S_s > 0$ **then**
    return: $\text{indexOf}\{\mathbf{x'}_s \text{ in } \mathbf{x}\}$
else
    return: -1
end if

4.6 Global Segment Smoothness

Our similarity classifier gives us segmentation solutions which consider only two voxels at a time. For each pass of the classifier, we have a result representing the similarity between each voxel in the scene and the chosen seed voxel (See Figure 12)

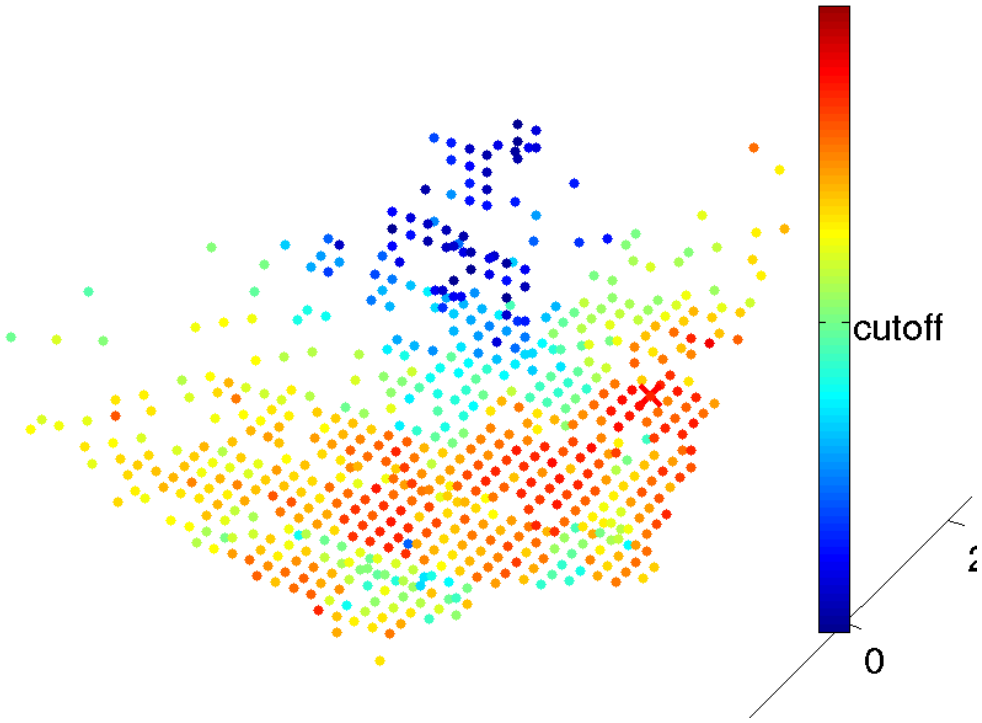


Figure 12: This image represents one pass of the similarity classifier on an outdoor scene. The seed voxel is represented by a red "x" near the right of the image. The seed is a ground voxel in this case. The color of each voxel represents its level of confidence. Voxels more red are more likely similar while more blue voxels are more likely different. The blue area in the center contains a bench and a tree, so they look very different from the ground. Notice the single blue voxel near the bottom, which is a ground point incorrectly labeled as not similar.

We believe this can be improved by using a graphical model to take into account the 3D structure of a scene. In general, changed objects contain many voxels, so changed voxels are likely to be in close proximity to other changed voxels. Adding such a neighborhood consistency assumption to our similarity classifier allows us to remove some of the noise inherent to the classifier. This closely parallels the smoothing assumption used to continually adjust the ground height estimate within the perception system to take into account both evidence from laser data as well as maintaining a degree of smoothness due to the continuity of

the ground surface[28].

It is common in machine learning to represent structure in data through a graphical model. A Markov random field (MRF) is a type of graphical model in which nodes represent some state variable and undirected edges represent the dependence of those two variables. If two nodes are connected with an edge, those variables are not independent. However, two isolated nodes (with no connected path) are independent. We can express conditional independence between two variables if we know the values of each node that connects the two variables using methods from graph theory. [25]

In general, such MRF problems are solved by minimizing an *energy function* which consists of two terms:

$$E(l) = E_{data}(l) + E_{smooth}(l) \tag{3}$$

Where $E(l)$ is the energy of the proposed labeling l. $E_{data}(l)$ represents the extent to which the data disagrees with the proposed labeling, while $E_{smooth}(l)$ measures the degree of violation of the smoothness constraint.

In our scenario, the energy function for a set of segmentation labels $\mathbf{y}_{1...N}$ for an MRF over the N voxels with features $\mathbf{x}_{1...N}$ where voxel 1 is assumed to be the current seed voxel can written as:

$$E(\mathbf{y}_{1...N}) = \sum_{i=2}^{N} E_{classifier}(\mathbf{x}_i, \mathbf{x}_j) + \lambda \sum_{\{i,j\} \in \mathcal{N}} \delta(y_i y_j < 0) + \infty \delta(y_1 \neq 1) \tag{4}$$

$$E_{classifier}(\mathbf{x}_i, \mathbf{x}_j) = \left\{ \begin{array}{ll} |c(\mathbf{x}_i, \mathbf{x}_j)| & \text{if } sign(c(\mathbf{x}_i, \mathbf{x}_j)) \neq (y_i y_j) \\ 0 & \text{otherwise} \end{array} \right\} \tag{5}$$

Where:

- $y_i = 1$ means voxel i is in the same segment as the seed voxel and $y_i \neq 1$ means voxel i is not in the same segment as the seed voxel. While the algorithm is running the y variables represent our current guess at the labeling.
- $c(\mathbf{x}_i, \mathbf{x}_j)$ is the output of the similarity classifier described in Section 4.3
- λ is the neighbor smoothness weight

The first term in Equation 4 represents the degree to which the segmentation agrees with the evaluations of the classifier (between the seed voxel and the rest

of the voxels). If the sign of the classifier (positive if same, negative if different) does not match the assignment of the labels, a penalty equal to the magnitude (confidence) of the classifier is applied. The second term ensures a neighborhood constraint by penalizing mismatching neighbors by the smoothness parameter λ. Finally, we ensure that the seed voxel $\mathbf{x}_1$ is in segment 1 by applying an infinite penalty if this constraint is not met.

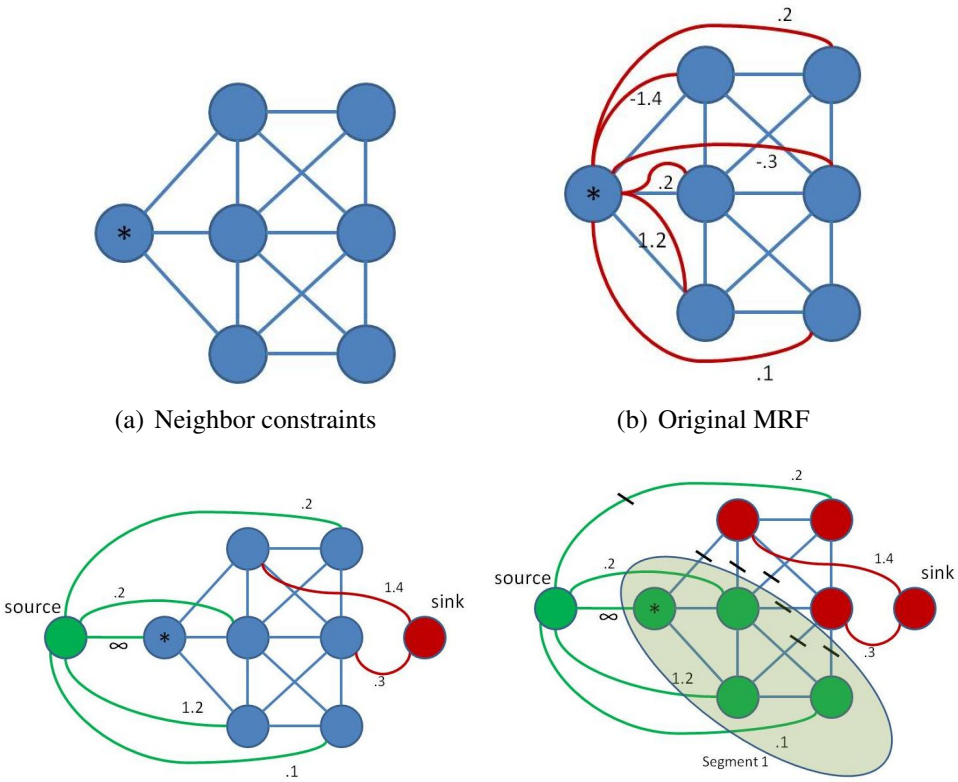
In general, minimizing the energy of a k-class MRFs is NP-hard, but the 2-class problem is a special case that can be solved optimally in polynomial time using the graph-cut algorithm if the energy of the distribution of each pair of binary variables is submodular[29, 30].

To solve our MRF represented by the energy function in Equation 4 using the min-cut algorithm, we construct a graph as follows. We create two additional virtual nodes to function as the source and the sink nodes in the graph. These nodes represent the two segments to be computed by this iteration of the segmentation process where the seed voxel will below to the source. For each classifier evaluation, if the classifier says the voxel is the same as the seed, that voxel is connected to the source node with an edge with weight equal to the magnitude classifier's output (the confidence of the prediction). If the classifier predicts that the voxel is different from the seed, that voxel is instead connected to the sink by the magnitude of the classifier's output. Neighborhood constraints are represented simply as connections between those pairs of voxels with a weight equal to the smoothing parameter, λ . Finally, the seed node is connected to the source node by an edge with in finite weight, ensuring that it will be part of that class. Because all edge weights are positive, this graph meets the submodularity requirement (having all positive weights meets this criteria).

After the graph-cut optimization is performed, any voxel connected to the source node is part of the current segment while any voxel connected to the sink node is part of another segment. Any edge that is broken in the optimal segmentation is a computed constraint that was violated.

An example of this approach can be seen in Figure 13. The neighbor constraints for the voxel set is shown in Figure 13(a). The full MRF with classifier constraints representing the energy function we are trying to minimize is shown in Figure 13(b) along with classifier edge weights. The graph-cut representation of this two-class MRF is shown in Figure 13(c). The final segmentation computed by optimizing the graph-cut representation of the MRF is shown in Figure 13(d). Broken edges resulting in energy penalties are marked with a slash through them. Notice that as a result of the neighborhood constraints in this example, the top-right node is not selected as part of the seed voxel's segment even though the

classifier gave a weak recommendation to do so.



(a) Neighbor constraints

(b) Original MRF

(c) Graph-cut representation of MRF

(d) Final segmented MRF

Figure 13: MRF optimization process for a sample problem.

4.7 Segmentation Termination

Deciding when to terminate segmentation is not trivial. It is possible to consider adding segments until every voxel has been assigned a segment, but this often creates tiny fractured segments which contain only a very small number of voxels. These small segments are not useful to change detection because they do not provide contextual information and the problem essentially reverts back to one of point-by-point change detection. It is important to remember that segmentation is not our end-goal, so we are only interested in segmentations that will be useful for

change detection.

We also believe that an ideal segmentation should have a hard time segmenting noise voxels, since they will not appear to be similar to any other voxels in the scene. If these noise voxels appear inside a segment, the MRF smoothing parameter should make them seem as part of the segment, but if the voxels are spatially isolated, they will not become part of any segment. This is possible in cases where rain or dust causes spurious laser reflections and creates stray voxels. The system of binning the pointclouds into larger voxels helps to eliminate some of this noise, but it can still be present. Instead of adding segments for each of these noise voxels, we seek to terminate segmentation when no more good segments exist.

To achieve termination, we segment until either all voxels have been segmented, or we cannot find a valid seed. If there are no seeds with nonzero score (see Equation 2 in Section 4.5), this means we cannot find a voxel which seems to be in the same object as any other voxel we have sampled, and our seed selection algorithm returns -1 This means that none of the remaining voxels appear to be in the same segment, so there are no more good segments to be found and we terminate.

We can see that segmentation will always terminate because voxels are always being eliminated from the working set $\mathbf{x}$. Since seed selection is a randomized algorithm, we cannot be sure if it will or will not return a valid seed. However, if it does not return a valid seed, segmentation will terminate because the output of algorithm 3 gets set at line 10 or 12. If there are valid seeds, we will optimize an MRF as described in section 4.6. Since there is an infinite weight between the seed voxel and the source of the graph, the seed voxel will always be in the segment. This means that regardless of the neighbor constraints or classifier weights, the segment generated from an MRF optimization must have at least one voxel. This voxel will then be removed from the working set in line 21. Since each step either terminates or reduces the size of $\mathbf{x}$ we will eventually fall into the true branch of line 6 and terminate the algorithm.

4.8 Parameter Optimization

Because of the complexity of this pipeline, thee are many important system parameters that have great impact on performance. In order to determine the final values for these parameters we ran through our change detection data set (see section 3.3) beginning with our best guess for reasonable values. Then we performed a manual coordinate descent, by varying one parameter at a time to determine its affect on the data and optimizing iteratively. Additionally, several parameters and

Name	Description	Value
k	The number of seed candidates in seed selection from algorithm 4	12
λ	The MRF smoothing parameter from equation 4	0.25
η	The percentage of a segment which must be changed for a changed segment from algorithm 3	0.45

Table 2: A summary of the relevant parameters of segmentation and change detection

options were ruled out based on smaller runs or using visualization. See results in section 5.3.1 for explanations of the effects of the various parameters.

5 Results

Overall, we have found that our approach to change detection is effective. While there are still circumstances that cause our algorithm to perform poorly, it does a good job classifying the majority of logs and is a strong step in the direction of a fully robust and performant change detector for feature-rich 3D scenes. We have also found that by treating voxels as members of segments, we can improve change detection. Since the ROC curve for change detection with segmentation is uniformly greater than the curve for change detection without segmentation, we can conclude that segmentation is a valid approach for improving change detection. An example segmentation is shown in Figure 15 and the corresponding change detection in Figure 14

5.1 Evaluation Criteria

One of the most difficult aspects of segmentation-based algorithms is determining what is a good segmentation. Early in the project, we relied heavily on visualization methods to look at the qualitative results of segmentation. It is very difficult even for humans to determine what is a good segmentation. Consider, for example, a building. Should the entire building be one segment? Or should each window and door be its own segment? It could also be valid for each brick and window pane to be in its own segment.

Even if we could come up with a standard to determine which segmentations were best, labeling a full segmentation training set would be prohibitively diffi-

Figure 14: A visualization of change detection without segmentation (left) and with segmentation (right). The red outline is the outline of the changed voxels. The result with segmentation has notably fewer false positives

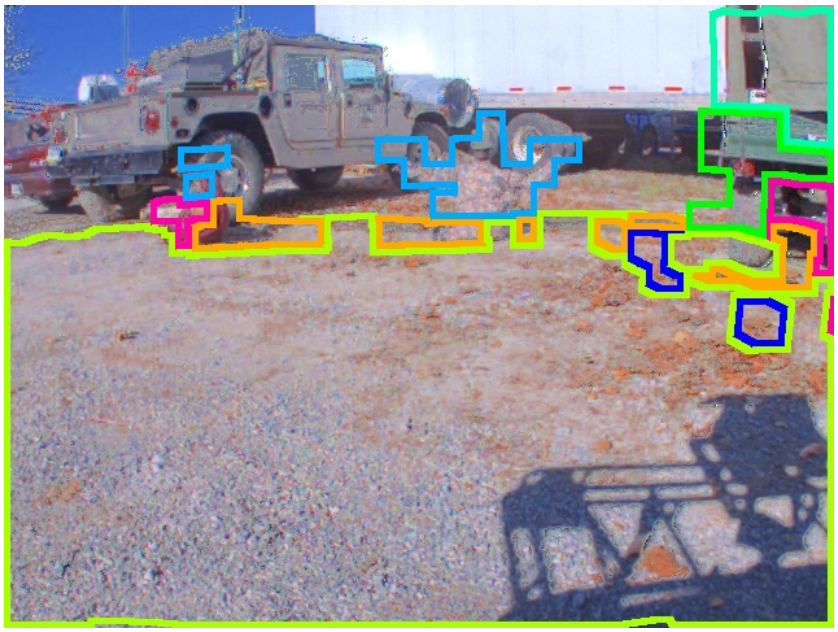


Figure 15: A visualization of segmentation. Note that not all points seen by the camera represent valid voxels within laser range

cult. When we labeled data points for the voxel classifier (see Section 3.3) we labeled only the points that clearly belonged to a certain class. This allowed us to skip edge voxels and voxels that were hard to see. Since the data are binned into 20 cm^3 voxels, it is almost impossible even for humans to determine exactly

where the edges are. Any human-labeled segmentation also runs the risk of being inconsistent and punishing a classifier for creating a segmentation that would be valid.

Since segmentation is coupled with change detection, We take the approach of classifying the quality of a segmentation based on how useful it is for the change detection problem. There is not much use for a segmentation by itself, so it makes sense to look at how the segments are being used and determine quality of the segmentation with a measure of the amount of improvement in change detection.

Change detection is evaluated based on human-labeled change data (see Section 3.3). We run change detection both with and without segmentation for a given parameter set and compare the output to the hand-labeled data, reporting the rate of true positives (voxels both our classifier and the labeled data mark as changed) vs. the rate of false positives (voxels our classifier marks as changed which are not marked as changed in the human-labeled data). Since the parameter γ which controls the sensitivity of the voxel-based change detection algorithm (algorithm 2 described in section 3.5) creates a trade-off between true positives and false positives, we consider varying that parameter while holding all other parameters constant. This produces a Relative Operating Characteristic (ROC) curve which is a plot of true vs. false positives. The top left of an ROC curve represents the ideal of 100% true positives and 0% false positives. There is no need for a similar ROC curve with true negatives vs. false negatives since these plots would be symmetrical. The true negative rate is just 1 minus the false positive rate, and similarly for false negatives. The line $y = x$ represents the cases when classifier gets an equal portion of true and false positives, meaning that it gets as many classifications correct as it gets incorrect, and is therefore a worthless classifier.

More formally, let $\phi(v)$ be the ground-truth about voxel v, 1 for a change and 0 for a non-change. Let $\hat{\phi}(v)$ be the result of our change detection algorithm. For a given parameter set and γ, the location on the ROC plot is:

$$(x, y) = \left(\frac{\sum_{v \in V} f(v)}{\sum_{v \in V} \phi(v)}, \quad \frac{\sum_{v \in V} t(v)}{\sum_{v \in V} 1 - \phi(v)} \right)$$

Where true positives are:

$$t(v) = \begin{cases} 1 & \hat{\phi}(v) = \phi(v) = 1 \\ 0 & \text{otherwise} \end{cases}$$

and false positives are:

$$f(v) = \begin{cases} 1 & \hat{\phi}(v) = 1 \cap \phi(v) = 0 \\ 0 & \text{otherwise} \end{cases}$$

Note that the plot is scaled by the number of true positives and true negatives so that both axes represent a rate of prediction.

Once an ROC curve is generated with a parameter set, we can find an optimal threshold. In the simplest form the optimal threshold is just the threshold which maximizes the score function $h(x)$

$$h(v) = t(v) + (1 - f(v)) \tag{6}$$

This means we weight true positives and false positives equally. In other words, we would not care if we got one more true positive at the expense of one additional false positive. It is possible to modify equation 6 to increase true positives or decrease false positives. For change detection, it may make sense to prefer false positives so the robots are overly cautious. We can therefore add a risk parameter, α, to equation 6 to get

$$h(v) = \alpha t(v) + (1 - f(v)) \tag{7}$$

An α of 1 represents the standard scoring function, while higher values prefer true positives (allowing more false positives) and lower values are stricter, allowing fewer false positives. For the parameter optimization in this thesis we consider only $\alpha = 1$, but it would be straightforward to extend our analysis for different values.

When comparing parameter sets we look at pairs of ROC logs and determine which optimal threshold is better (has a higher score in equation 7). Even though we vary γ for the ROC generation, γ is actually a fixed parameter of the system, so an ideal segmentation is one that has a better score for all values of γ. This would cause the segmentation ROC curve to be uniformly above and to the left of the voxel-based change detection curve. Additionally, we do not want to limit out solution to only being better than voxel-based change detection for certain values of α so we want an ROC curve which is uniformly better regardless of α and γ. This way, if there are some requirements about minimum true positive rates and maximum false positive rates, they can generally be imposed without changing the parameters of the system. However, if strong demands were places on these cutoffs, changing the parameters might improve the performance of the system,

but as long as the curve for change detection with segmentation is to the upper-left of the curve without segmentation, it will always be better (on average) to use segmentation.

For comparison purposes we also consider a very naïve occupancy-based approach to the change detection problem. Here we consider as changed every new voxel which appears in the current scene but wasn't occupied historically. This algorithm can only detect changes that add new objects to the scene, which make up the majority of our dataset. This occupancy-based algorithm has no parameters and is thus represented as a single point in our ROC analysis.

5.2 Quantitative Results

Our optimal total performance for change detection with segmentation was 80.3% true positives with 12.96% false positives. To achieve the same true positive rate using change detection without segmentation, we would have to incur a 24% false positive rate. This was achieved with a threshold of 0.1245. Segmentation outperforms results without segmentation for almost every log we recorded. Our baseline occupancy based algorithm performed much worse, resulting in a 62.27% true positive rate while incurring 37.35% false positives. See Figure 16 for a total ROC plot of our system run against all labeled logs.

These results were generated with a pre-scanned baseline and a replayed log, which means it should be feasible to run this algorithm online. For these ROC results, we slowed down playback to 20% of real-time, but there are many performance optimizations that could be made to this algorithm. Some are described in Section 7. In the interest of time during the parameter selection process, we pre-scan the baseline logs instead of running them through the perception system, although the system does have this capability.

5.3 Analysis

Overall, we are pleased with the performance of our change detection system using scene segmentation. We believe that the segmentation step improves change detection, but we also think there is still some room for improvement. We believe that our results make sense. Occupancy does very poorly because it is a very naïve method which cannot deal with any mis-registration in the data, displayed in Figure 17. We also see that segmentation improves the ROC curve uniformly across the threshold parameters, which is what we expect. Individually, ROC plots such as those in Figure 20 generally have better performance than the corresponding

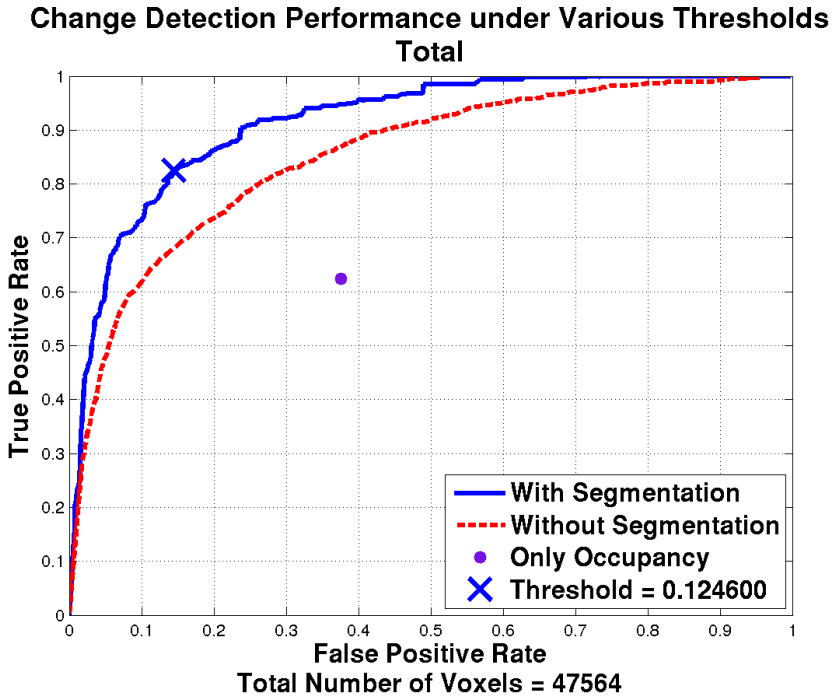


Figure 16: Performance of the Change Detection system with and without segmentation

ROC for change detection without segmentation. We also see that the segmentation performance is somewhat discrete, which is also expected. This happens because we are marking changes on a segment-by-segment basis, so performance will remain constant until another segment meets the thresholds, at which point there will be a jump, possibly in both true and false positives, which is exactly what we see.

5.3.1 Effects of System Parameters

The weight on the neighborhood constrain in the MRF, λ (see Equation 4) essentially encodes how smooth we expect our segments to be. If the smoothing parameter is very low we do very little smoothing and rely almost entirely on the voxel-based classifier. This leads to increased false positives because noise doesn't get canceled out by the surrounding voxels. This does, however, help in detecting smaller objects. If the parameter is very high, we get very smooth segments and very few false positives, but we are too likely to miss important seg-

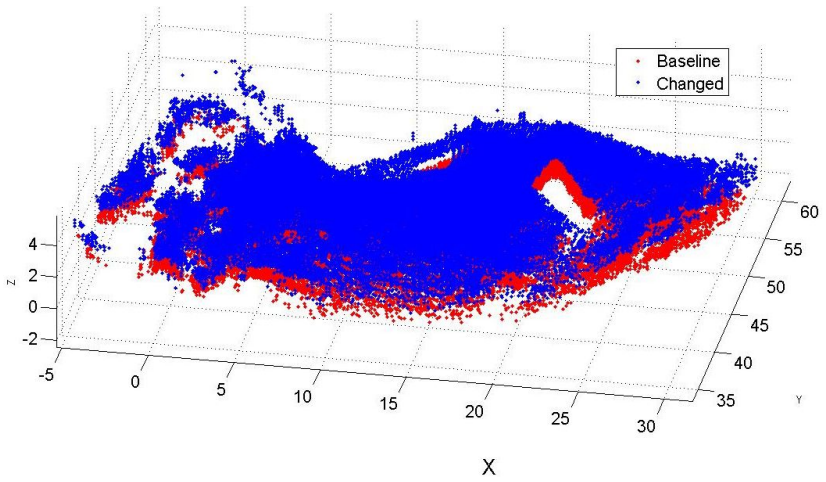


Figure 17: This plot represents the misalignment between subsequent runs of data collection that our algorithm can handle

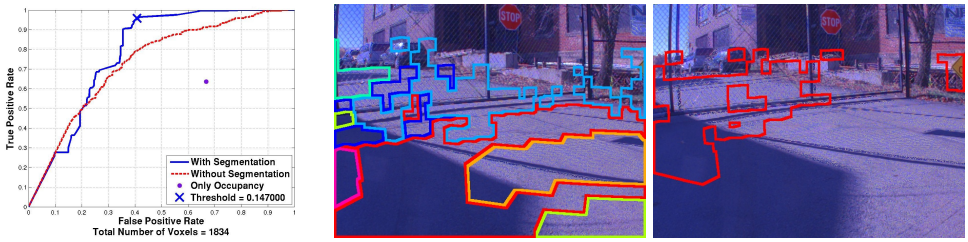
ments or merge multiple segments which should have been kept distinct. Figure 18 shows the effect of extreme values of the smoothing parameter on segmentation and the resulting change detection.

The segment change threshold, η is the percentage of a segment which must be novel in order for the entire segment to be considered novel. Low values of this parameter cause too many false positives because noise in a segment may cause it to become changed. High values here have a risk of missing changes, especially when the segmentation quality is poor. If the segments are too large, and encompass both a change and a non-change, too high a change threshold will cause the change to be missed. We generally consider it better to mark a larger area as changed than to miss a change that happens within a large segment. See Figure 19

This parameter is somewhat coupled to the smoothness because an optimal smoothness parameter will give us well sized segments. If the smoothing parameter is high, we tend to get larger segments, and so need a smaller segment change threshold in order to capture changes, while if the smoothing constant is low, we get many smaller segments and so can use a higher threshold.

5.3.2 Strengths and Weaknesses

There are several factors that we believe influence the performance of segmentation, and thus change detection. First, although we can handle error in localiza-

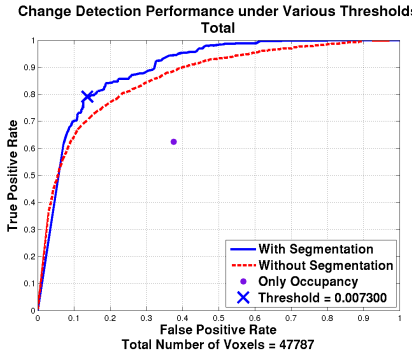


(a) $\lambda = 0.05$ creates too many segments, causing the detected changed to be less accurate

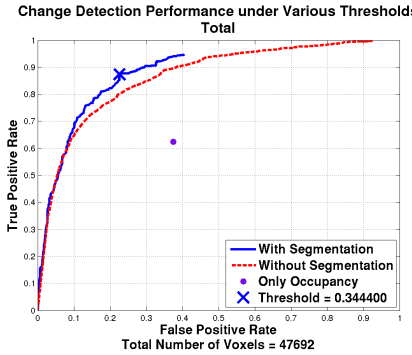
(b) $\lambda = 5.0$ creates too large of a segment and misses most of the change. Notice the increase in false positives due to oversized segments

Figure 18: Side-by-side segmentations with varying smoothness parameters. See figure 20(d) for optimal parameters

tion, the better a log is registered the better out results are likely to be. Since we have no a priori assumptions about things like number or size of segments, some logs prove very difficult to segment. For example, the log depicting a manikin on a pile of logs in Figure 20(c) is very difficult to segment and comes out looking fractured and difficult to interpret. Additionally, the ROC curve is pushed towards the false positives. We believe that this may be due to the complexity of the scene. The logs create many disjoint segments. This can cause segmentation to perform worse because it will have a harder time finding valid voxels which represent a large volume of the voxels in the scene. Additionally, any noise on the edges of real segments will be amplified by the number of segments. Logs like the barrel and manikin on concrete slabs in figure 20(a) perform extremely well, possibly due to the distinctness of the changes compared to the baseline. Although we never trained the similarity classifier with anything resembling the shape or color of the manikin sitting on the concrete slab (we didn't even use any manikins for the similarity classifier), the segmentation module has an easy time identifying it. Logs like this might be "easier" because the occupancy only approach seems to



(a) $\eta = 0.1$ causes too many false positives



(b) $\eta = 9.9$ makes it too so difficult to get true positives that the algorithm cannot achieve better than about 95% for any threshold

Figure 19: ROC curves for high and low values of the percent segment change threshold

do better on this log than on logs like the one in figure 20(d). The closing gate log is a great example of this algorithms success because something like a gate can be very difficult for a robot to understand the traversibility cost of because it is made up of so many small and isolated pieces.

5.3.3 Sources of Error

There are several possible causes of error that may have partially degraded our results. We believe that a large source of error may come from poor human labeling of the change detection pair data logs explained in Section 3.3. It was very difficult to determine exactly which voxels should be labeled, especially around the borders of objects. This is especially difficult because of the voxelization. If we could determine the true edge of a change, it would occur somewhere within a 20cm^3 voxel. This means that on average, half of the information inside the voxel we labeled is incorrect. In logs where changes were partially occluded by vegetation, it was almost impossible to differentiate between vegetation voxels and the new object's voxels, so there are almost certainly incorrectly labeled voxels in those logs.

Another source of error is the effect of parallax between the camera and laser scanner on the robot. The perception system combines data from these two sources to assign colors to each voxel, but the two sensors are not at the same location.

If there are objects near the robot, there will be a region of voxels which the laser scanner can see but the camera cannot, even though the voxels are within the range of the projected camera image (they are "behind" something the camera sees). This causes some voxels to be incorrectly colored.

We also believe that better registration between scenes would improve results. While we do not require perfect registration, a better registration means that more of voxels from the current segment will correspond to the ones they are projected upon in the historical data. Registration is a serious issue with the data we collected. Due to GPS error, it is not uncommon to have logs taken subsequently be separated by 2 meters (see Figure 17), and we regularly deal with at least half a meter of error.

6 Contribution

In this thesis we have presented a new approach to change detection which takes into account contextual information based on 3D scene segmentations. We have demonstrated experimentally that this approach provides an advantage over voxel-by-voxel change detection. Our algorithm runs on-line and was evaluated using real data collected from a robot so we know that our technique is robust to noise in sensor data and poor localization.

This work is a step towards the goal of increasing the fieldability of autonomous mobile robots. We have demonstrated that it is possible to detect change which could represent a danger to the robot or its environment. We believe that a system like the online segmentation-based change detection algorithm presented in this thesis can improve robotics by alleviating the pressure on learning systems. Amazing work has gone into complex systems which can classify obstacles and potential dangers with great accuracy. We believe it to be incredibly difficult if not impossible to improve those systems to 100% accuracy, so our approach is to attempt to detect when a complex autonomy system might fail. Change detection is useful because it can reduce the amount of time it takes to bring a robot into use in the field because it allows humans to monitor robots and check in on then if they encounter a new situation which they may not be able to handle. It also allows fewer humans to oversee more robots since the robots can flag circumstances in which they may need human help. This allows the humans to focus their time on the more important tasks which may be difficult for robots. Change detection is also a useful end result for applications like security patrolling or inspection where the goal is to notice if any changes are going on which might breach security or

safety.

We believe that even further improvements can be made to the method of change detection and that this idea of detecting changes will be the best way to bring advanced robotics systems to use.

7 Future Direction

We would like to gather more data or find ways of using existing data sets to further validate our algorithm.

We believe that we could improve change detection results by re-working the perception system to be more friendly to segmentation and change detection. The perception system was created primarily to discover obstacles that may obstruct the robot. Since the robot is large enough to traverse over small obstacles, 20 cm^3 was a logical choice of voxel size. However, given finer resolution of voxels we believe we may be able to improve segmentation since we could find better boundaries between objects. We think it might even be best to segment based on the original point-cloud data before voxelization to get the most information, which is what other 3D segmentation methods use. This would create more computational difficulties and may introduce noise, but it would also be possible to get much finer grained segmentations.

Another place for potential improvement is in the training of the similarity classifier. Although we achieved great test accuracy in our SVM classifier (95%), the data we used were not actual segmentations, but rather labels which determined similarity. It may be possible to label entire segmentations and train a classifier directly on the segmentation data-set. We may also be able to improve the way we use the similarity classifier to optimize global smoothness. Our SVM approach gives us an unbounded confidence, but there are modifications to SVM which use Bayesian methods to return actual probabilities for each classification [27]. Using this method requires more computation but may produce better results.

There are many possible ways to improve the runtime and memory usage efficiency of the change detection pipeline. We are using a very large feature space (60 dimensions for feature vectors used in the voxel classifier), so we spend a huge amount of time computing kernel functions on these large vectors. Decreasing the size of the space would have some impact on change detection performance, but could drastically improve runtime. We also have not optimized memory efficiency at all and removing some of the duplicated data in the pipeline would improve

both space and time efficiency. Also, we could use random sampling methods to create the MRF constraints instead of comparing the seed to every single other voxel. Also, since segments represent disjoint regions of the scene, we could process change detection on each segment in parallel. It may also be possible to parallelize much of the segmentation pipeline. Optimization for runtime would be important if this algorithm were going to be used in the field.

One promising idea we hope to explore is using multiple segmentations and combining the results. If we could rank how confident an entire segment was as to whether or not it had changed, we could weigh votes between multiple segmentations with different parameters and seeds. The idea is that some of the segments in some of the segmentations will be very good, and those will have a higher weight, making the overall change detection result more accurate. This idea is being applied in computer vision to determine properties of objects [31, 32, 33].

There are also recent techniques in graphical models which allow the training of a graph much like our MRF but that maximizes a margin. This approach has shown promise in 3D segmentation where the classes are known ahead of time [24, 34].

We would also like to begin to consider what happens when the robot does encounter change. We would like to explore ways to notify humans when potentially important changes are discovered, while optimizing for the humans time. One possibility is to ask humans when they are available and simply avoid changed areas when all humans are busy. We would then provide a method for humans to mark changes as safe, which would update the online change detection system to mark those things as non-changes the next time they are seen.

We are also interested in how an algorithm like this could work with a team of robots. By sharing information it may be possible for groups of robots to detect change more robustly. This is especially interesting because one of the ideal uses of change detection is in a setting where humans are working with robot teams and the team of robots is trying to optimize the humans time.

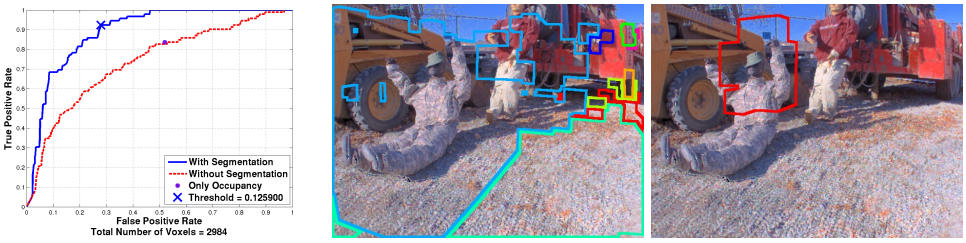
Acknowledgments

I would like to start by thanking my senior thesis advisor, Tony Stentz, for his support and guidance and for providing me the resources without which I could have never completed this work. I would like to thank Boris Sofman for his tremendous impact on my research. He has helped me develop and express my ideas and without his strong collaboration none of this research would have been possible.

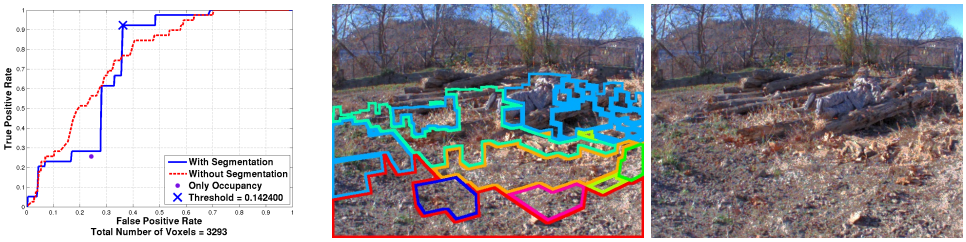
I would also like to thank Dominic Jonak, Balajee Kannan, Freddie Dias, Nisarg Kothari, Daveid Silver and everyone else who is or was a part of the CTA project. These people provided me great support and an incredible system upon which I could carry out this research.

This work was partially sponsored by the U.S. Army Research Laboratory under contract Robotics Collaborative Technology Alliance (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

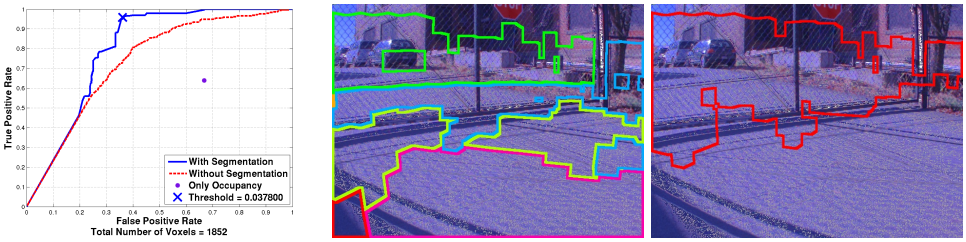
(a) Despite never having trained on an object this shape or color, the system has no problem identifying the manikin as a change



(b) Despite good ROC performance, this segmentation misses the second manikin entirely



(c) The segmentation is so bad that none of the segments surpass the segment change threshold η and we see no detected changes at all



(d) The gate is somewhat difficult to segment, but segmentation is still helpful. The baseline for this log is the same scene with the gate opened

Figure 20: Performance on selected individual data logs. The ROC performance, scene segmentation, and detected changes are shown from left to right.

References

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

[2] A. Kelly, A. Stentz, O. Amidi, M. Bode, D. M. Bradley, A. Diaz-Calderon, M. Happold, H. Herman, R. Mandelbaum, T. Pilarski, P. Rander, S. Thayer, N. Vallidis, and R. Warner, "Toward reliable off road autonomous vehicles operating in challenging environments," *I. J. Robotic Res*, vol. 25, no. 5-6, pp. 449–483, 2006. [Online]. Available: http://ijr.sagepub.com/cgi/content/abstract/25/5-6/449

[3] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. M. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. R. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. V. Nefian, and P. Mahoney, "Stanley: The robot that won the DARPA grand challenge," *J. Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006. [Online]. Available: http://dx.doi.org/10.1002/rob.20147

[4] A. Stentz, J. Bares, T. Pilarski, and D. Stager, "The crusher system for autonomous navigation," in *AUVSIs Unmanned Systems North America*, 2007.

[5] A. Stentz, C. Dima, C. Wellington, H. Herman, and D. Stager, "A system for semi-autonomous tractor operations," *Auton. Robots*, vol. 13, no. 1, pp. 87–104, 2002.

[6] A. Stentz, "The focussed d* algorithm for real-time replanning," in *In Proceedings of the International Joint Conference on Artificial Intelligence*, 1995, pp. 1652–1659.

[7] B. Sofman, J. A. D. Bagnell, and A. T. Stentz, "Anytime online novelty detection for vehicle safeguarding," in *IEEE International Conference on Robotics and Automation*, May 2010.

[8] A. Singh, "Review article digital change detection techniques using remotely-sensed data," *International Journal of Remote Sensing*, vol. 10, no. 6, pp. 989–1003, 1989.

[9] F. Gustafsson, *Adaptive Filtering and Change Detection*. John Wiley & Sons, LTD, 2000.

[10] D. M. Bradley, R. Unnikrishnan, and J. Bagnell, "Vegetation detection for driving in complex environments," in *ICRA*. IEEE, 2007, pp. 503–508. [Online]. Available: http://dx.doi.org/10.1109/ROBOT.2007.363836

[11] J.-F. Lalonde, N. Vandapel, D. Huber, and M. Hebert, "Natural terrain classification using three-dimensional ladar data for ground robot mobility," *Journal of Field Robotics*, vol. 23, no. 1, pp. 839 – 861, November 2006.

[12] B. Schölkopf and A. J. Smola, *Learning with Kernels*. Cambridge, MA: The MIT Press, 2002.

[13] R. O. Duda and P. E. Hart, *Pattern Classification*. John Wiley and Sons, 2000.

[14] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, Sept. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000022288.19776.77

[15] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Conf. Computer Vision and Pattern Recognition*, June 1997.

[16] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 24, no. 5, pp. 603–619, 2002. [Online]. Available: http://computer.org/tpami/tp2002/i0603abs.htm

[17] L. G. Shapiro and G. C. Stockman, *Computer Vision*. Prentice-Hall, 2001.

[18] A. Golovinskiy and T. Funkhouser, "Min-cut based segmentation of point clouds," princeton University.

[19] D. Munoz, N. Vandapel, and M. Hebert, "Onboard contextual classification of 3-d point clouds with learned high-order markov random fields," the Robotics Institute, Carnegie Mellon University.

[20] D. Huber, A. Kapuria, R. R. Donamukkala, and M. Hebert, "Parts-based 3d object classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 04)*, June 2004.

[21] A. E. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3D scenes," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 433–449, May 1999. [Online]. Available: http://ieeexplore.ieee.org:80/xpls/abs_all.jsp?isNumber=16576&prod=JNL&arnumber=765655&arSt=+433&ared=+449&arNumber=765655

[22] S. Ruiz-Correa, L. G. Shapiro, M. Meila, and G. Berson, "Discriminating deformable shape classes," in *NIPS*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds. MIT Press, 2003. [Online]. Available: http://books.nips.cc/papers/files/nips16/NIPS2003_VM01.pdf

[23] G. Mori, S. J. Belongie, and J. Malik, "Shape contexts enable efficient retrieval of similar shapes," in *CVPR*, 2001, pp. I:723–730. [Online]. Available: http://ieeexplore.ieee.org:80/xpls/abs_all.jsp?isNumber=21353&prod=CNF&arnumber=990547&arSt=+723&ared=+730&arNumber=990547

[24] D. Anguelov, B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng, "Discriminative learning of markov random fields for segmentation of 3D scan data," in *CVPR*, 2005, pp. II: 169–176. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2005.133

[25] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

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

[27] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[28] C. Wellington, A. Courville, and A. Stentz, "Interacting markov random fields for simultaneous terrain modeling and obstacle detection," in *Proc. of Robotics Science and Systems*, June 2005.

[29] R. Zabih, O. Veksler, and Y. Y. Boykov, "Fast approximate energy minimization via graph cuts," in *ICCV*, 1999, pp. 377–384. [Online]. Available: http://dx.doi.org/10.1109/ICCV.1999.791245

[30] Kolmogorov and Zabih, "What energy functions can be minimized via graph cuts," *IEEETPAMI: IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, 2004.

[31] T. Malisiewicz and A. A. Efros, "Improving spatial support for objects via multiple segmentations," in *British Machine Vision Conference (BMVC)*, September 2007.

[32] D. Hoiem, A. A. Efros, and M. Hebert, "Geometric context from a single image," in *ICCV*, 2005, pp. I: 654–661. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2005.107

[33] B. C. Russell, W. T. Freeman, A. A. Efros, J. Sivic, and A. Zisserman, "Using multiple segmentations to discover objects and their extent in image collections," in *CVPR*, 2006, pp. II: 1605–1614. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2006.326

[34] B. Taskar, V. Chatalbashev, and D. Koller, "Learning associative markov networks," 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.71.8512; http://kingman.cs.ualberta.ca/_banff04/icml/pages/papers/394.ps