

Extracting Terrain Features from Range Images for Autonomous Random Stepfield Traversal

Raymond Sheh, M. Waleed Kadous, Claude Sammut
ARC Centre of Excellence for Autonomous Systems
School of Computer Science and Engineering
The University of New South Wales
Sydney NSW 2052 Australia
[rsheh|waleed|claudes]@cse.unsw.edu.au

Bernhard Hengst
NICTA
School of Computer Science and Engineering
The University of New South Wales
Sydney NSW 2052 Australia
bernhard.hengst@nicta.com.au

Abstract — One of the challenges of rescue robotics is to create robots that can autonomously traverse rough, unstructured terrain. Although mechanical engineering can produce very capable robots, mechanical engineering alone will not drive them. In this paper, we present a terrain feature extractor that can be taught to find significant features in range images of terrain around a robot. A terrain model is generated from the many points in the range sensor data. Techniques from the field of knowledge acquisition are then used to find patterns in the terrain model. A knowledge acquisition system can then be taught to drive a robot in unstructured terrain based on these features. We evaluate the performance of the initial stages of the feature extractor on a real robot, traversing NIST specification red stepfields.

I. INTRODUCTION

One of the challenges of rescue robotics is to create robots that can autonomously traverse rubble and rough terrain at a disaster site. Mechanical engineering can help in this regard and competitions like RoboCup Rescue see many very capable designs for the purpose of traversing rough terrain on a scale seen in search and rescue environments. However, mechanics alone won't drive a robot.

Much of the terrain that rescue robots can be expected to operate in consists of highly unstructured features that cannot be easily classified into traversable or non-traversable, even if they could be identified. At a human scale, this would be like scrambling over a rocky seawall where the traversability of the various rocks is neither obvious nor constant. This traversability would also depend more heavily on the ability of the human (height, flexibility, strength, etc.) than in the case of walking along a forest path.

Our aim is to create robots that are able to traverse these highly unstructured terrains. We use NIST Random Stepfields [2] as an analog for real rubble. Stepfields are designed to be easily reproduced, and yet behave in a similar way to real rubble. To standardise difficulty, NIST publishes a set of randomly generated patterns in various sizes. Our experiments are run on NIST Red stepfields which consist of blocks around 90mm per side and up to 400mm high. This paper describes the development of a set of features that can compactly represent the terrain around the robot, as sensed by the robot's range imager. These features are used by the decision making

processes on the robot in order to decide on the appropriate action to take in order to traverse the terrain.

A. Platform and Sensors

Our robot, CASTER3, is based on a Yujin Robotics Robhaz DT-3 EOD robot, and was our entry in the 2005 and 2006 RoboCup Rescue Robot League competitions where we came 3rd and reached the semi-finals respectively. CASTER3 was designed for stairclimbing and overcoming obstacles in an urban area, however its two pairs of tracks and articulated body give it reasonable performance on NIST Red stepfields.

Our selection of robot and terrain is such that one cannot drive the robot over the stepfield as if it were flat ground. Doing so would result in the robot being trapped. However, when driven with care, it is almost always possible to traverse the terrain. Thus it is a good test for autonomous agent systems that are to sense the terrain and drive carefully.

Our primary sensor for terrain characterisation is the CSEM SwissRanger SR-3100 range imager [4]. This camera-like device provides a 176x144 array of pixels, each pixel being a distance value with a field of view of approximately 40°. This 3D data is low in distortion, dense, lighting and surface texture invariant and enables us to focus on 3D feature extraction. An XSens MTi heading-attitude sensor allows us to rotate the 3D data to the horizontal and provides an estimate for yaw.

B. Related Work

Several groups have tackled the issue of 3D terrain representation for the purpose of robot behaviour although most have done so from the perspective of autonomous road vehicles.

Representations such as those used in [6] process pointclouds sensed at a particular instant in time and do not make use of an ongoing map. Obstacles are segmented based on their deviation from the driving surface, which need not be level or flat. The thresholding of these deviations is governed by the capabilities of the robot. Other approaches that attempt to segment out interesting features from raw pointclouds include [8] where a statistical approach was used to classify sections of pointclouds taken from an autonomous vehicle in a forest environment and used to detect such obstacles as trees and wires.

Some other representations consist of occupancy grid based techniques. In [9], a 3D scanning laser was used to detect solid obstacles hidden amongst sparse vegetation by maintaining an occupancy grid. The use of multiple hits/non-hits per cell, taken from several viewpoints as the robot moves around, allows solid obstacles (which are always hits) to be distinguished from sparse obstacles (which are often non-hits). This grid is then used to predict the behaviour of the platform (a large tractor) and for planning an appropriate path. [5] also use an occupancy grid based technique, with stereo vision on a planetary rover used to build an occupancy grid map of the terrain around the robot. This is then analysed for terrain “goodness” by observing the roll, pitch and roughness of patches of ground comparable in size to the robot.

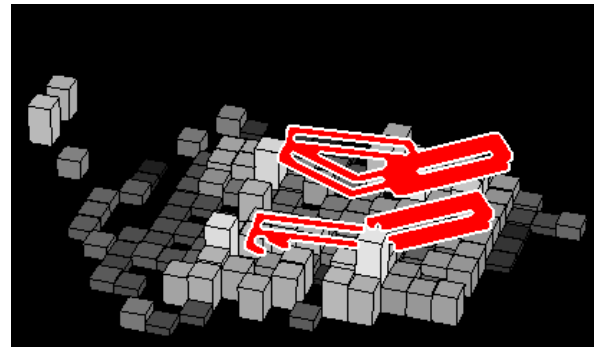
II. FEATURE CHARACTERISTICS

Our prior work on representing unstructured terrain [3] started in consultation with expert robot drivers. We asked them what types of characteristics were considered when driving the robot. This was backed up by eyetracking experiments to determine what aspects of the terrain were actually focused on during driving.

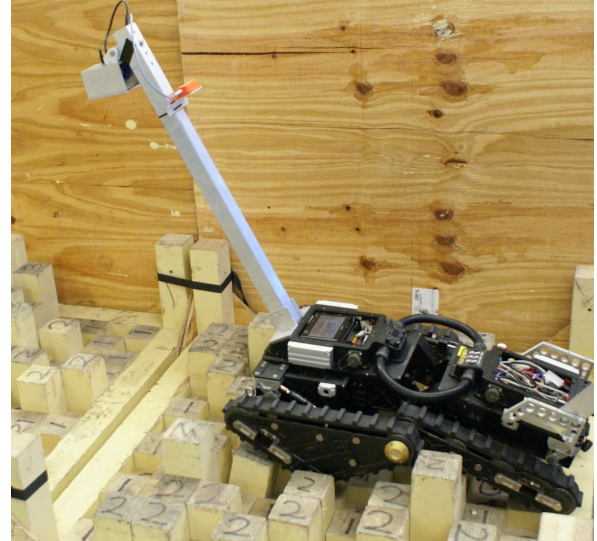
As a result of this, we developed features that capture the overall “lay of the land” and major deviations from this, such as holes or protrusions. The former was characterised by planes that were fit through the terrain in various regions around the robot, to the distance of around one robot length. The latter was characterised by finding large deviations from these planes. These features were recorded along with the actions that an expert driver took when the features were recorded and the result fed into a bagged decision tree generator in order to produce rules that could control the robot.

Although we had some success with this representation in traversing stepfields, there were terrain characteristics that were important to operators but were lost in the feature representation. When an expert drives a robot over stepfields, specific features such as ruts and ridges in the terrain can be just as important as protrusions and overall “lay of the land”. However, our previous representation often missed these features. The operator was also not able to specify particular regions of interest or particular features to look for and thus a channel for providing additional information to the system was not exploited. Our new representation seeks to address this issue and also enable a much richer set of features to be developed.

Many features, like ruts and ridges, have dominant directions. These features are important because they can heavily influence the behaviour of the robot. Tracks and wheels can become stuck or the robot “railroaded” by ruts and ridges that lie close to the direction of travel. Those same ruts and ridges are less dangerous when they lie across the direction of travel. Holes and tall obstacles can also cause ground clearance problems. Bolts, braces and other fittings on the sides of the robot can get caught on terrain features, thus protrusions in the terrain that brush along the sides of the robot must also be detected.



(a)



(b)

Fig. 1. (a) An example of a terrain model with the position of the robot shown as a wireframe. By including the terrain models generated earlier in the traversal, we automatically and in real-time reconstruct the terrain underneath the robot. (b) A photograph of the actual stepfield with the robot in a similar location to the visualisation above. The rear of the robot (to the right) appears different as the articulated suspension is not modeled in the visualisation.

Our feature representation takes a two level approach. First, a terrain model is formed. This consists of discrete blocks in a fixed sized grid covering a particular area around the robot. In order to find a grid that best fits the important features in the terrain, the grid should be aligned with the dominant features in the terrain. This avoids problems where dominant features, such as holes, become “smudged” between grid cells and maximises the amount of information that can be conveyed without resorting to a finer grid or multi-resolution representations, both of which increase complexity and training time. As a feature, the pattern of blocks in the terrain model are reasonably robust and unique and thus can also be used to estimate the robot’s motion, as demonstrated in Figure 1.

This terrain model is then used as input into a set of block feature detectors. These detectors flag the presence or absence of a set of predefined features such as “Gap smaller than the robot in front”, “Large blocks about to hit the side of the robot” and so-on. Thus, block features are patterns found in the blocks

of the terrain model. These block features, combined with data from the heading-attitude sensor, are used as input into a knowledge acquisition system that outputs the appropriate driving instruction.

Although we have taken advantage of the fact that the stepfields have features biased towards two highly dominant directions in finding our terrain model, this representation is applicable in more general terrain where often there is still a dominant direction in a local area. In the complete absence of any dominant direction, the terrain model becomes a standard discrete grid that can still detect patterns, holes and protrusions.

III. IMPLEMENTATION

The implementation of our feature extractor is split into three sections. First the raw range and attitude data is processed and a pointcloud and heightfield generated. The dominant directions of terrain features are then extracted, the terrain model formed and successive sets of terrain models combined to provide a map that includes portions of the terrain that are no longer visible, in a process similar to Simultaneous Localisation and Mapping. Finally, a set of block feature detectors is applied to the terrain model to produce the final set of features.

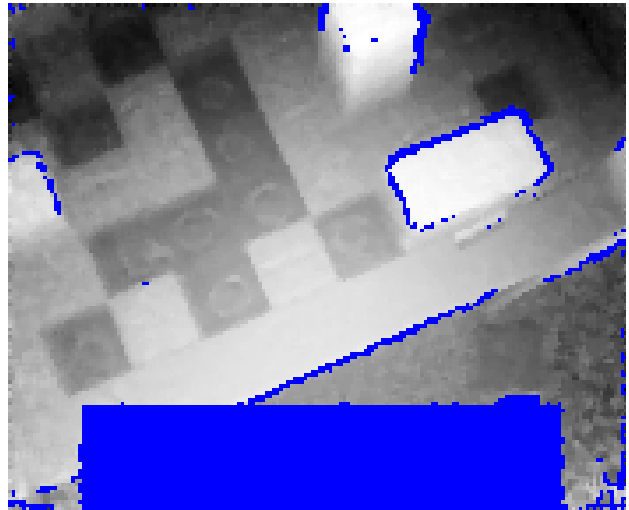
A. Processing Range Data

The SR-3100 provides a 176x144 pixel range image. The front of the robot body, visible in the range image, is masked out and sharp edges filtered to remove edge artifacts. The result is smoothed slightly to reduce noise. The resulting data is projected into a pointcloud, which is then rotated by the pitch and roll of the robot as detected by the heading-attitude sensor. This process is shown in Figure 2.

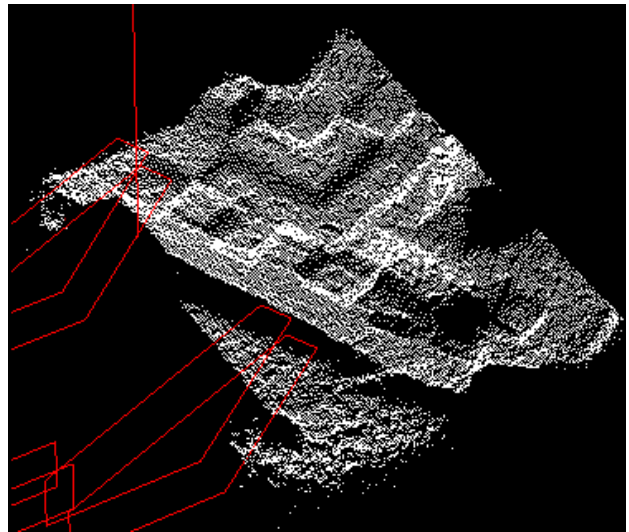
A heightfield, or vertical occupancy grid, is produced from this pointcloud. For our application a cell size of 6mm per side was chosen. A Sobel edge detector is then applied to the heightfield to detect the edges of features. The edge detector ignores portions of the heightfield that were not observed, in order to avoid detecting edges that are artifacts of occlusion. This process is shown in Figure 3.

B. Generating the Terrain Model

The edges present in the heightfield are searched for a dominant grid pattern. A modified Radon transform (similar to a Hough transform) [7] is used. This transform finds dominant instances of arbitrary, parameterised patterns in an edge image. Each edge pixel is projected into all possible instances of the desired patterns in parameter space that could contain it. By discretising the parameter space and storing the number of “hits” each instance receives, it is possible to find instances of the pattern in parameter space that happen to lie on many edge pixels – these instances correspond to patterns that are actually present. Each entry in parameter space also stores a list of pixels that contributed to it, allowing the exact locations of contributing pixels to be determined.



(a)



(b)



(c)

Fig. 2. (a) The range image with the robot and smeared points (shown in red) filtered out. Darker portions are further from the camera. (b) The resulting pointcloud with the robot's position shown in wireframe. (c) A photograph of the robot and stepfield taken from the same angle.

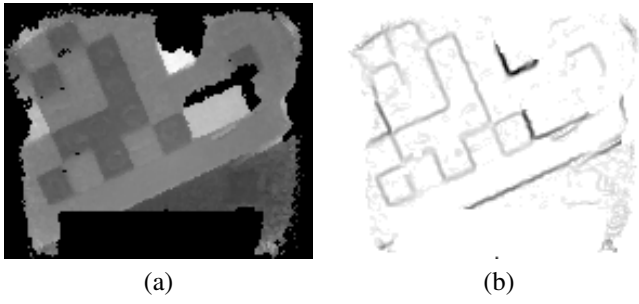


Fig. 3. (a) The heightfield produced from the pointcloud in Figure 2 where lighter colours are higher. (b) The result of treating this as an image and applying a modified Sobel edge detector that ignores areas with no data.

We treat the heightfield as an image¹ and the shape we wish to find is that of a 2D square grid of a set cell size and infinite extent but with unknown position (X_{grid} , Y_{grid}) and orientation θ . These three parameters are bounded by the cell size for X_{grid} and Y_{grid} (moving an infinite grid by one gridcell is equivalent to no motion) and $[0, \pi/2]$ for θ . We discretise position into steps of 6mm to match the heightfield, and rotation to 1° . We use grid cells of 90mm (15 steps) per side to match the size of blocks on the Random Stepfields. This provides us with a three dimensional parameter space accumulator of size 90x15x15 entries.

An angle, θ , and edge pixel are first chosen. The closest distance y between a line at the desired angle passing through the edge pixel and the origin at the lower left corner is found. The corresponding column in the accumulator is given by $Y_{grid} = (y \bmod cellsize)$ and all entries in this column for the selected angle are incremented. This is shown in Figure 4(a). Next, the closest distance x between the origin and a line perpendicular to the desired angle is found. The corresponding row in the accumulator is given by $X_{grid} = (x \bmod cellsize)$ and all entries in this row for the selected angle are incremented. This is shown in Figure 4(b). The process repeats for each combination of angle and edge pixel. Figures 4(c) and (d) describe the actual grid found. The resulting 3D parameter space accumulator can be visualised as 90 two dimensional arrays as shown in Figure 5.

Given the dominant grid orientation and position, edge pixels that lie on the grid can be found. The area of the heightfield that contributed to that grid can then be segmented. This enables the segmentation of multiple features that are not standard gridcells apart. For each area found, the corresponding grid is overlaid onto the original heightfield and a weighted average of the points in the heightfield within each cell computed to find the estimated height of that cell. The weighting is biased towards the center of each cell to reduce the effects of slight alignment issues at the edges due to skew grids, imperfect prerotation in pitch and roll and the like. Grid cells that fail to cover a sufficient number of points in the heightfield or contain too much variation are unlikely to be clean blocks and are removed. The resulting terrain model is

¹By using the heightfield instead of the raw range image we avoid problems with variations in angle and height of the range imager.

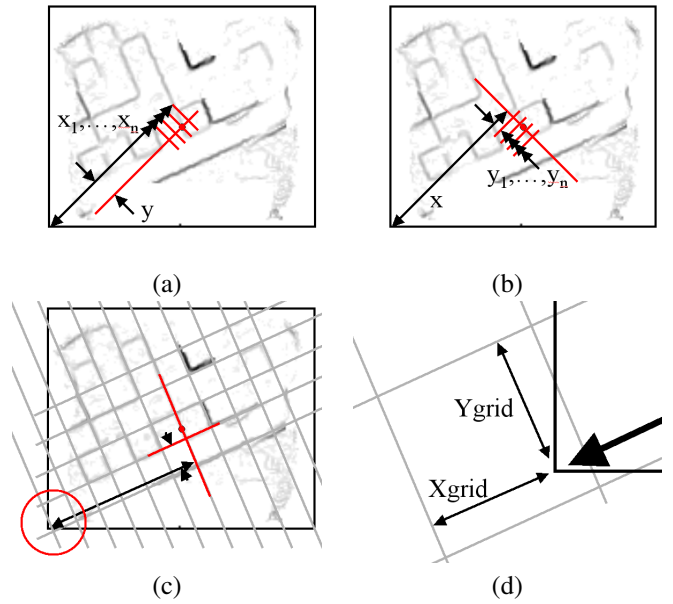


Fig. 4. The process of finding the dominant grid pattern through a set of edge pixels. See text for a full explanation.

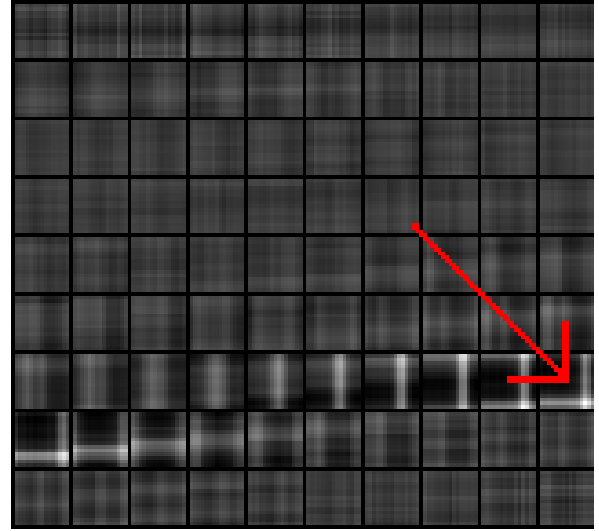


Fig. 5. The 3D parameter space accumulator, sliced along the θ axis into 90 separate 2D accumulators where the horizontal and vertical axes are X_{grid} and Y_{grid} . The maximum (arrow) corresponds to $\theta = 25^\circ$ and a grid offset of 72mmx78mm.

shown in Figure 6 and consists of a discrete set of blocks of known height, orientation and position relative to the robot.

Once the robot has created several terrain models over a period of time, they can be used to determine the movement of the robot and build a terrain map. Two terrain models are matched by inspecting all possible matches and calculating a match quality for each. For a given match, the heights of the blocks in the two terrain models are shifted by the centroids of the matching blocks to remove the effect of the robot changing height relative to the terrain (perhaps caused by the robot rolling or pitching). The average difference between the heights of corresponding blocks in the two terrain models are then calculated and used as the matching quality.

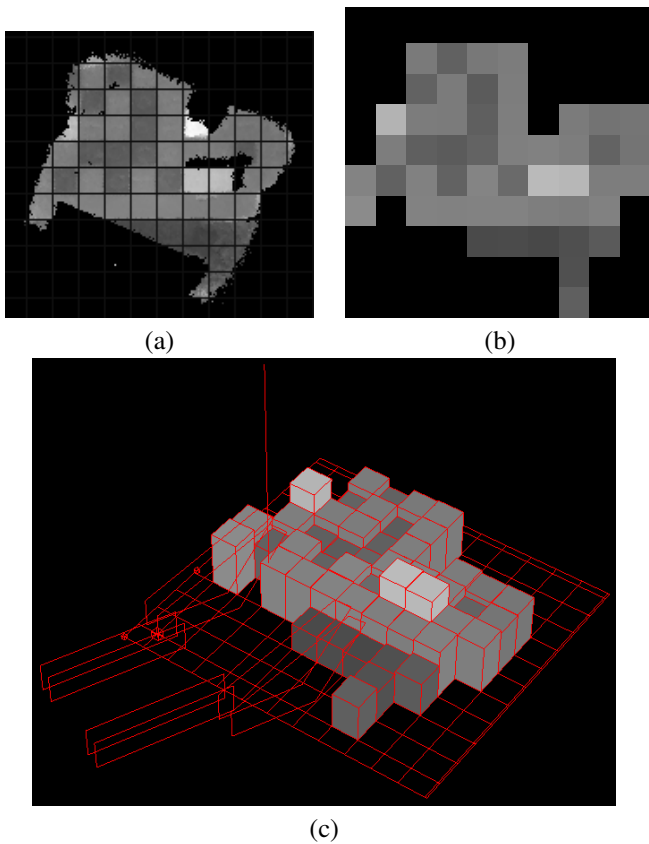


Fig. 6. (a) The grid corresponding to the maxima entry in the parameter space shown in Figure 5 (and the heightfield rotated by this grid). (b) The weighted average of points within each gridcell found and filtered to form the blocks of the terrain model, shown in bitmap form. (c) The terrain model rendered from the same perspective as the pointcloud in Figure 2.

The size of the terrain model grid is small, generally no more than 20x20, as it is made up of relatively coarse gridcells. Therefore the match can be performed for all possible matches in a brute force fashion. This match quality is weighted by the amount of motion required in the robot for that match to occur. This could be replaced by a proper motion model if available.

Although the matching is performed at a coarse level, because the location of the robot relative to the terrain model is continuous, the error in the match is the error in the grid position and orientation detection process. These terrain models are then merged, providing the block feature detectors with information about the terrain underneath and around the robot. Figure 1 demonstrates the result of merging a number of terrain models.

C. Block Feature Detectors

It is possible to supply the blocks of the terrain model themselves as input into a machine learning program. However, the state space that would result would be so large that the amount of training examples required would be excessive, especially since training examples for these types of learning problems need to be recorded from real robot runs.

Instead, we develop a set of block feature detectors, each one designed to report the presence or absence of a given

feature in the terrain model. The resulting vector of true and false values corresponding to the presence or absence of each of the features can be used in a machine learning program. It is possible to explicitly program detectors for each type of feature we wish to detect. However, we wish to have a stronger grounding in reality. We use a technique from Ripple-Down Rules (RDRs), a form of Knowledge Acquisition [1].

A RDR system starts by always reporting a default action, such as “drive forward”, regardless of the terrain. To train the RDR, an operator monitors its reported actions and, when it makes a mistake, stops it. The operator enters the correct action and then creates a rule that allows the RDR to decide when it should take this new action in the future. For instance, the first time the robot sees an obstacle it will try to drive forward into it. The operator, seeing this, will stop the robot, tell the RDR that the robot should instead turn left, and generates a rule, based on the terrain model, that can detect the obstacle. The RDR stores the terrain model so that in the future, if the operator decides that this rule has made an incorrect choice, for instance because now there are obstacles in front and to the left, the RDR can present the old and new terrain models and ask the operator for a new rule that can tell the two apart and the appropriate action to take. Internally, the RDR is represented as a sequence of statements, similar to IF-THEN-ELSE but where any given IF clause can be overridden by an EXCEPT clause.

After further training, this might produce an RDR that normally drives forward, prefers to turn left when it sees an obstacle but turns right when it can’t turn left and reverses if it sees a hole (but no obstacle) in front of it. The internal representation would look like this.

```

IF normal case THEN go forward
EXCEPT
  IF obstacle in front THEN go left
  EXCEPT
    IF obstacle at left THEN go right
  ELSE
    IF hole in front THEN reverse

```

A particular feature of RDRs is that at each IF line, only a small number of features are required. Thus it is possible to have a large library of feature detectors available and yet not need to use all of them all of the time. In the above example, the “hole in front” feature detector would not be used if there was an obstacle detected in front of the robot. Also, during training, examples are shown to the operator, who can explicitly define feature detectors based on those examples.

To create feature detectors, first we drive the robot over the stepfield by teleoperation and record data from the range imager and heading-attitude sensor. In order to keep the relationship between the observed situation and the resulting action consistent, we drive the robot by taking a given action for one second, then stopping and recording a set of data from the robot’s sensors, in preparation for the next action. The robot was driven in a similar way for our previous sets of experiments with considerable success and doing so also removes the need to account for the robot’s momentum.

Each set of data is then processed to form a terrain model

TABLE I
ERROR RATES IN DETECTING ACTUAL STEPFIELD BLOCKS

Set	Blocks	Correct	50mm classified as 100mm	Other error
1	43	100%	0%	0%
2	71	91.5%	8.5%	0%
3	69	100%	0%	0%
4	65	95.4%	4.6%	0%
5	59	94.9%	5.1%	0%
6	43	100%	0%	0%
7	32	96.9%	3.1%	0%
8	37	75.3%	22%	2.7%
\sum	419	94.8%	5%	0.2%

which is then shown to an experienced operator. Through the user interface, the operator marks out areas around the robot in which to look for block features and defines the pattern of blocks that form those features. For example, to define an “obstacle in front” feature detector, the operator might mark out the area in front of the robot and specify a pattern that involves a tall block surrounded by lower blocks. This pattern is significant as it can cause ground clearance problems. The operator also specifies the correct action to take and thus generates a rule. This process repeats for each pair of range image and heading-attitude sensor measurement obtained during the test runs. This process continues during early autonomous runs where at each step, the RDR proposes an action to take based on its training and the operator only creates a new rule when an error is made. Although generated in the context of RDRs, these feature detectors can be used more generally. They could be used as binary features within a decision tree for behavioural cloning, for instance [3].

IV. RESULTS

Our feature extraction was tested in two stages. The accuracy of the terrain model generator can be easily evaluated by traversing the stepfield and assessing the error rate of the detector and we present these results for this stage.

The robot was driven over the stepfield and 8 terrain models generated from 8 sets of sensor data. These sets contained a total of 419 blocks. The blocks were classified into one of the five discrete Red Stepfield block heights (50mm, 100mm, 200mm, 300mm, 400mm) based on the closest matching cluster and the resulting pattern compared to the groundtruth. The results for each terrain model collected are shown in Table I. Misclassifications of 50mm blocks as 100mm blocks is listed separately as they are closer together and thus more easily misclassified. There are two main sources of error. An error in the attitude measurements or distortion in the depth sensing results in a detected block appearing higher or lower than it actually is, this was the main cause of the errors in set 5. An error in the position or orientation of the detected grid results in grid cells containing points from two or more blocks. Generally the error is minor so only a few points from adjacent blocks appear in each cell but an excessive number will skew the averaging and result in an error. Most of the errors seen were due to this.

The evaluation of the block feature detectors is more difficult as it depends on the skill of the operator and the ultimate

sensitivity of the RDR to the quality of these features. As this work is still preliminary, we have no numerical results. However, during our initial experiments we have found that it is relatively easy for the operator to define features such as “block on side of robot”, “obstacle in front of robot” and so-on based on training examples, and have those feature detectors find qualitatively similar block patterns in subsequent terrain models.

V. CONCLUSIONS AND FUTURE WORK

We have described and demonstrated an effective terrain model generator for range images of stepfields, taken as a robot traverses them. We have also described our method for extracting higher level block features from terrain model using methods taken from Knowledge Acquisition. These features encode concepts like “tall block in front of robot”, “ridge running along robot” and so-on, features that are important to detect in order to effectively drive a robot over unstructured terrain.

The next step is to train the RDR to detect a variety of features and drive the robot appropriately. We predict that these features will not depend on the terrain being a stepfield as features with dominant edge directions must still be detected. Further experiments will be undertaken to test this.

ACKNOWLEDGMENTS

The ARC Centre of Excellence for Autonomous Systems is funded by the Australian Research Council. NICTA is funded by the Australian Government’s Backing Australia’s Ability initiative, in part through the Australian Research Council.

REFERENCES

- [1] Paul J. Compton and R. Jansen. A philosophical basis for knowledge acquisition. *Knowledge Acquisition*, 2:241–257, 1990.
- [2] Adam Jacoff, Brian Weiss, and Elena Messina. *Reference Test Arenas for Urban Search and Rescue Robots*. National Institute for Standards and Technology, 2005.
- [3] M. Waleed Kadous, Claude Sammut, and Raymond Sheh. Autonomous traversal of rough terrain using behavioural cloning. In *Proceedings of the 3rd International Conference on Autonomous Robots and Agents*, 2006.
- [4] T. Oggier, M. Lehmann, R. Kaufmann, M. Schweizer, M. Richter, P. Metzler, G. Lang, F. Lustenberger, and N. Blanc. An all-solid-state optical range camera for 3D real-time imaging with sub-centimeter depth resolution (SwissRanger). In *Optical Design and Engineering. Proc. of the SPIE*, volume 5249, pages 534–545, February 2004.
- [5] Sanjiv Singh, Reid Simmons, Trey Smith, Anthony Stentz, Vandia Verma, Alex Yahja, and Kurt Schwehr. “Recent progress in local and global traversability for planetary rovers”. In *Proc. 2000 IEEE Intl Conf on Robotics and Automation*, 2000.
- [6] A. Talukder, R. Manduchi, A. Rankin, and L. Matthies. “Fast and Reliable Obstacle Detection and Segmentation for Cross-country Navigation”. In *Intelligent Vehicles*. IEEE, June 2002.
- [7] M. van Ginkel, C.L. Luengo Hendriks, and L.J. van Vliet. A short introduction to the radon and hough transforms and how they relate to each other. Technical Report QI-2004-01, Imaging Science and Technology Department, Faculty of Applied Science, Delft University of Technology, 2004.
- [8] Nicolas Vandapel, Daniel Huber, Anuj Kapuria, and Martial Hebert. “Natural terrain classification using 3-d lidar data”. In *IEEE International Conference on Robotics and Automation*, April 2004.
- [9] Carl Wellington and Anthony (Tony) Stentz. “Online adaptive rough-terrain navigation in vegetation”. In *Proc IEEE Intl Conf on Robotics and Automation*, April 2004.