Learning Long-Range Vision for Autonomous Off-Road Driving

Raia Hadsell[1] **Pierre Sermanet**[1,2] **Jan Ben**[2] **Ayse Erkan**[1]
Marco Scoffier[1,2] **Koray Kavukcuoglu**[1] **Urs Muller**[2] **Yann LeCun**[1]

(1) Courant Institute of Mathematical Sciences
New York University
New York, NY USA

(2) Net-Scale Technologies
Morganville, NJ USA

Abstract

Most vision-based approaches to mobile robotics suffer from the limitations imposed by stereo obstacle detection, which is short-range and prone to failure. We present a self-supervised learning process for long-range vision that is able to accurately classify complex terrain at distances up to the horizon, thus allowing superior strategic planning. The success of the learning process is due to the self-supervised training data that is generated on every frame: robust, visually consistent labels from a stereo module, normalized wide-context input windows, and a discriminative and concise feature representation. A deep hierarchical network is trained to extract informative and meaningful features from an input image, and the features are used to train a realtime classifier to predict traversability. The trained classifier sees obstacles and paths from 5 to over 100 meters, far beyond the maximum stereo range of 12 meters, and adapts very quickly to new environments. The process was developed and tested on the LAGR mobile robot. Results from a ground truth dataset are given as well as field test results.

1 Introduction

Humans navigate effortlessly through most outdoor environments, detecting and planning around distant obstacles even in new, never-seen terrain. Shadows, hills, groundcover variation - none of these affect our ability to make strategic planning decisions based purely on visual information. Human visual performance is not due to better stereo perception; in fact, humans are excellent at locating pathways and obstacles in monocular images (see Fig. 1 right). These tasks, however, are extremely challenging for a vision-based mobile robot. Current robotics research has recently begun to develop vision-based systems that can navigate through offroad environments; however, existing approaches generally rely on stereo algorithms, which produce short-range, sparse, and noisy costmaps that are inadequate for long-range strategic navigation.

The method of choice for vision-based driving in offroad mobile robots is to construct a traversability map of the environment using stereo vision. In the most common approach, a stereo matching

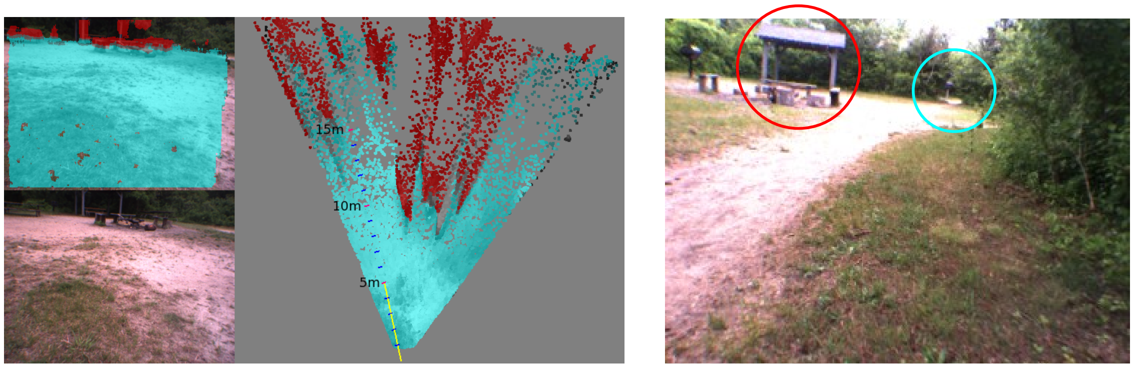


Figure 1: *Left:* Top view of a map generated from stereo (stereo is run at 320x240 resolution). The map is ”smeared out” and sparse at long range because range estimates from stereo become inaccurate above 10 to 12 meters.
Right: Examples of human ability to understand monocular images. The obstacles in the mid-range are obvious to a human, as is the distant pathway through the trees.

algorithm, applied to images from a pair of stereo cameras, produces a *point-cloud*, in which the most visible pixels are given an XYZ position relative to the robot. A traversability map can then be derived using various heuristics, such as counting the number of points that are above the ground plane in a given map cell. Maps from multiple frames are assembled in a global map in which path finding algorithms are run (Kelly and Stentz, 1998; Kriegman et al., 1989; Goldberg et al., 2002). The performance of such stereo-based methods is limited, because stereo-based distance estimation is unreliable above 10 or 12 meters (for typical camera configurations and resolutions). This may cause the system to drive as if in a self-imposed “fog”, driving into dead-ends and taking time to discover distant pathways that are obvious to a human observer (see Fig. 1 left). Stereo algorithms also fail when confronted by repeating or overly-smooth patterns, such as tall grass, dry scrub, or smooth pavement.

Recent learning-based research has focused on increasing the range of vision by classifying terrain in the far field according to the color of nearby ground and obstacles. This type of near-to-far color-based classification is quite limited, however. Although it gives a larger range of vision, the classifier has low accuracy and can easily be fooled by shadows, monochromatic terrain, and complex obstacles or ground types.

The primary contribution of this work is a *long-range vision system that uses self-supervised learning to train a classifier in realtime*. This approach poses several challenges which are addressed and resolved by our learning-based system. The first challenge is the choice of a feature representation that is robust to irrelevant transformations of the input, such as lighting and viewpoint. At the same time, the feature representation must be informative enough to recognize high-level terrain structures such as paths and obstacles. One way to successfully recognize a complex environment is to train a classifier with *discriminative features extracted from large image patches*. Training with large image patches allows for high-level recognition of obstacles, paths, groundtypes, and other natural features. Color histograms or texture gradients cannot replace the contextual information in actual image patches. Another challenge lies in classifier self-supervision. Training labels can be automatically generated by processing the output of a separate (short-range) sensor module, but *visually consistent categories* are important for successful learning. The classifier is trained using labels generated by stereo processing. If the label categories are inconsistent or extremely noisy, the learning will fail. Therefore, our stereo-based supervisor module uses 5 categories that are visually distinct: *super-ground*, *ground*, *footline*, *obstacle*, and *super-obstacle*. The supervisor module is designed to limit incorrect or misaligned labels; multiple ground planes are estimated to threshold the stereo points, and false obstacles are removed through analysis of plane distance statistics. A third challenge to this approach is the need for the classifier to generalize from near to far field. The visual size of features scales inversely with distance from the camera, and the training samples are all close-range. This difficulty is masked in systems that use simple color-based features, but becomes a challenge in our approach because of the wide-context feature representation. Our solution explicitly scales the training samples to make them *normalized with respect to scale and distance*. We normalize the image by constructing a horizon-leveled input pyramid in which similar obstacles are a similar height, regardless of their distance from the camera.

The long-range vision classifier was developed and tested as part of a full navigation system. The outputs from the classifier populate a hyperbolic polar coordinate costmap, and planning algorithms are run on the map to decide trajectories and wheel commands at each step. Since the classifier is trained online, its outputs vary over time as the inputs and training labels change. To accommodate this uncertainty, we use histograms to accumulate the classifier probabilities over time. Histograms allow us to accumulate evidence for a particular label in the face of changing classifier outputs. Similarly, the geometry of the hyperbolic polar map is designed to accommodate the range uncertainties that are intrinsic to image-space obstacle labeling, while also providing a vehicle-centered world representation with an infinite radius using a finite number of cells. Details of the full navigation system are given in (Sermanet et al., 2008b; Sermanet et al., 2008a).

The long-range classifier has been thoroughly tested. The accuracy of the classifier was assessed with hand-labeled, ground truth data as well as with its own supervisory labels. Experiments were also conducted to assess the impact of the long-range vision on the navigation performance of the robot. We show that on multiple test courses, the long-range vision yields driving that is smoother and strategically far-sighted. We also show examples of the accuracy of the classifier on diverse images.

1.1 The LAGR Platform

The vision system described here was developed on the LAGR platform (see Fig. 2). LAGR (Learning Applied to Ground Robots) is a DARPA program in which competing research labs must develop learning and vision algorithms for an outdoor, offroad autonomous vehicle. The participants are tested monthly by a DARPA testing team in new, unseen terrain (examples of terrain from 6 DARPA tests are shown in Fig. 2). The LAGR robot has 4 onboard computers, 2 stereo camera pairs, a GPS receiver, and an IMU (inertia measurement unit). It has a maximum speed of 1.2 meters/second. The hardware and the baseline navigation software were developed by Carnegie Mellon University and the National Robotics Engineering Center (NREC). The platform and the program have been thoroughly documented in previous publications (Jackel, 2005; Jackel et al., 2006).

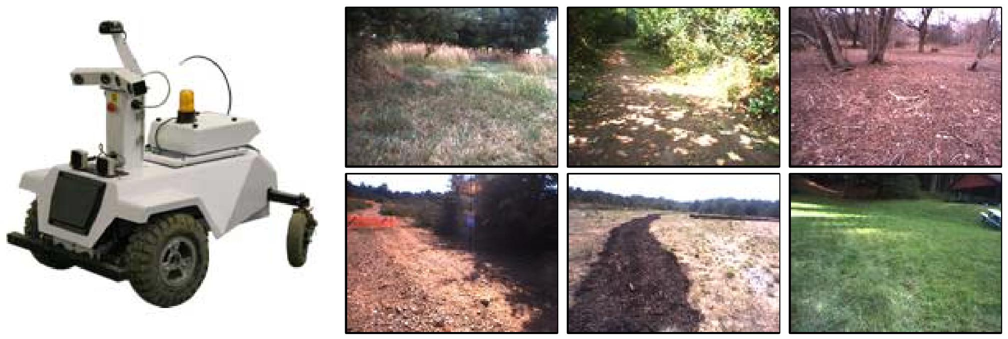


Figure 2: *Left*: The LAGR mobile robotic vehicle, developed by Carnegie Mellon University's National Robotics Engineering Center. Its sensors consist of 2 stereo camera pairs, a GPS receiver, and a front bumper. *Right*: Images from 6 different official LAGR test courses.

2 Related Work

Many methods for vision-based navigation rely on stereo-based obstacle detection (Kelly and Stentz, 1998; Kriegman et al., 1989; Goldberg et al., 2002). A stereo algorithm finds pixel disparities between two aligned images, producing a 3d point cloud. By applying heuristics to the statistics of groups of points, obstacles and ground are identified. However, the resulting costmaps are often sparse and short-range.

Recent approaches to vision-based navigation use learning algorithms to map traversability information to color histograms or geometric (point cloud) data. This is especially useful for road-following vehicles (Dahlkamp et al., 2006; Leib et al., 2005; Hong et al., 2002); the ground immediately in front of the vehicle is assumed to be traversable, and the rest of the image is then filtered to find similarly colored or textured pixels. Although this approach helped to win the 2005 DARPA Grand Challenge (Thrun et al., 2006), its utility was limited by the inherent fragility of color-based methods, and the online visual classifier was only used to slightly modulate the speed of the autonomous car, rather than used for planning of any sort.

Other, non-vision-based systems have used the near-to-far learning paradigm to classify distant sensor data based on self-supervision from a reliable, close-range sensor. Stavens and Thrun used self-supervision to train a classifier to predict surface roughness (Stavens and Thrun, 2006). A self-supervised classifier was trained on satellite imagery and ladar sensor data for the Spinner vehicle's navigation system (Sofman et al., 2006). An online self-supervised classifier for a ladar-based navigation system was trained to predict load-bearing surfaces in the presence of vegetation (Wellington and Stentz, 2004).

Not surprisingly, the greatest similarity to our proposed method can be found in the research of other LAGR participants. Since the LAGR program specifically focused on learning and vision algorithms that could be applied in new, never-seen terrain, using near-to-far self-supervised learning was a natural choice. Angelova et al. use self-supervised learning to train a feature extractor and classifier to discriminate terrain types such as gravel, asphalt, and soil. The visual representation is a 15 bin histogram of "texton" matches (Angelova et al., 2007). The SRI LAGR system used fast stereo and color-based online learning (Konolige et al., 2008). Staying within the realm of simple color-based near-to-far learning, Grudic and Mulligan explore the use of distance metrics for clustering traversable pixels in (Grudic and Mulligan, 2006). The Georgia Tech LAGR team built a self-supervised terrain classifier that uses traversability cues from close-range sensors (IMU, bumper switch) to train a classifier on stereo point cloud features (Kim et al., 2006). In a variation on the basic near-to-far strategy, Happold et al. use supervised learning to map stereo geometry to traversability costs; they also use self-supervised learning to map color features to stereo geometry. Thus their obstacle detection algorithm is in two phases: first color information is extracted and stereo geometry is predicted, then the predicted geometry is mapped to a traversability cost (Happold et al., 2006).

Our approach is distinguished from these by our careful examination of feature extraction methods, and our use of large image patches rather than color histograms or texture gradients or geometry statistics from stereo. Our system classifies the traversability of the image out to the horizon, unlike other, shorter-range approaches. Feature extraction is a critical component of a system that classifies large, distant image patches. If the feature representation can be found that is invariant to irrelevant transformations of the input while remaining discriminative to terrain and obstacle types, the classifier's work is greatly reduced.

3 Overview of Navigation and Long Range Vision

The LAGR platform is accompanied by navigation software developed by NREC and Carnegie Mellon. The navigation system uses a stereo-based obstacle detection module to populate a Cartesian coordinate map on every frame. The *D-star* algorithm is run on the global cost map to plan paths and generate driving commands. Our navigation system uses no part of this baseline software. Instead, our system uses multiple levels of perception and planning. At the lowest level, perception is simple and short-range, using very low resolution images. This allows very fast response times and robust obstacle avoidance, even allowing the vehicle to dodge moving obstacles. At the highest level, the perception and planning processes are much more sophisticated and use higher resolution images. The frequency and latency of these processes is much slower, but, because they are responsible for long-range vision and planning, a slower response time is acceptable. This multi-resolution architecture is fully described in (Sermanet et al., 2008b). The contribution of this paper is a full description and analysis of the long-range vision system.

The long-range vision system is a self-supervised, realtime learning process (see Fig. 3). It continuously receives images, generates supervisory labels, trains a classifier, and classifies the long-range portion of the images, completing one full training and classification cycle every half second. The only inputs are a pair of stereo-aligned images and the current position of the vehicle, and the output is a set of points in vehicle-relative coordinates where each point is labeled with a vector of 5 probabilities that correspond to 5 possible categories. The points and their energy vectors are used to populate a vehicle-relative polar-coordinate map which combines constant radius cells for the first 15 meters and hyperbolically increasing radius cells for cells from 15 meters to infinity. Path planning algorithms are run on the polar map, producing path candidates which interact with the short-range obstacle avoidance module to produce driving commands. The primary components of the learning process are briefly listed.

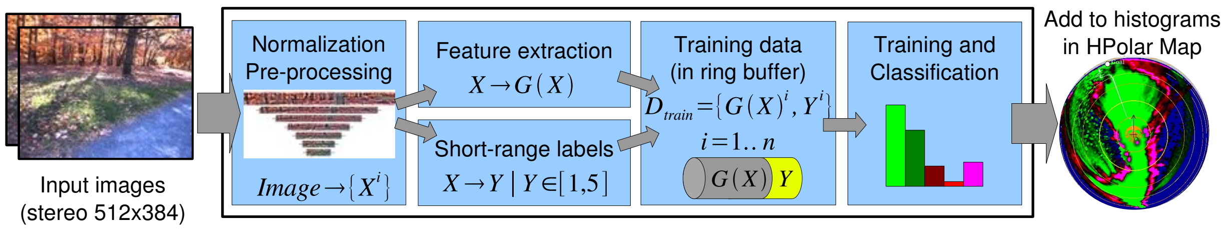


Figure 3: The input to the long-range vision module is a pair of stereo-aligned images and the current position and bearing of the robot. The images are normalized and features and labels are extracted, then the classifier is trained and immediately used to classify the entire image. The classifier outputs are accumulated in histograms in a hyperbolic polar map according to their vehicle-relative coordinates.

- **Pre-processing and Normalization.** Pre-processing is done to level the horizon and to normalize the height of objects such that their pixel height is independent of their distance from the camera.
- **Stereo Supervisor Module.** The stereo supervisor assigns class labels to close-range windows in the normalized input. This is a complicated process involving multiple ground plane estimation, heuristics, and statistical false obstacle filtering in order to generate training labels with as little noise as possible.
- **Feature Extraction.** Features are extracted from input windows in order to reduce dimensionality and gain a more discriminative, concise representation. Experiments with several different feature representations show that a multi-layer convolutional network, initialized with deep belief net training, is most effective. The filters are trained offline.
- **Training and Classification.** The classifier is trained on every frame for fast adaptability. We use stochastic gradient descent and a cross entropy loss function.

4 Horizon-Leveling and Normalization

We are strongly motivated to use large image patches (large enough to fully capture a natural scene element such as a tree or path) because larger context and greater information yields better learning and recognition. However, the problem of generalizing from nearby objects to far objects is daunting, since apparent size scales inversely with distance: Size $\propto \frac{1}{\text{Distance}}$. Our solution is to create a normalized "pyramid" of 7 sub-images which are extracted at geometrically progressing distances from the camera. Each sub-image is subsampled according to its estimated distance, yielding a set of images in which similar objects have a similar pixel height, regardless of their distance from the vehicle (see Fig. 4). The closest pyramid row has a target range of 4 to 11 meters distance and is subsampled with a scaling factor of 6.7. The furthest pyramid row has a range from 112 meters to infinity (above the horizon) and has a scaling ratio of 1 (no subsampling). The image row number to distance correlation is obtained by estimating the ground plane position; this estimate is calculated on each frame through a process described in Section 6.2.

A bias in the roll of the cameras, plus the natural bumps and grading in the terrain, means that the horizon is usually skewed in the input image. We normalize the horizon position in the pyramid

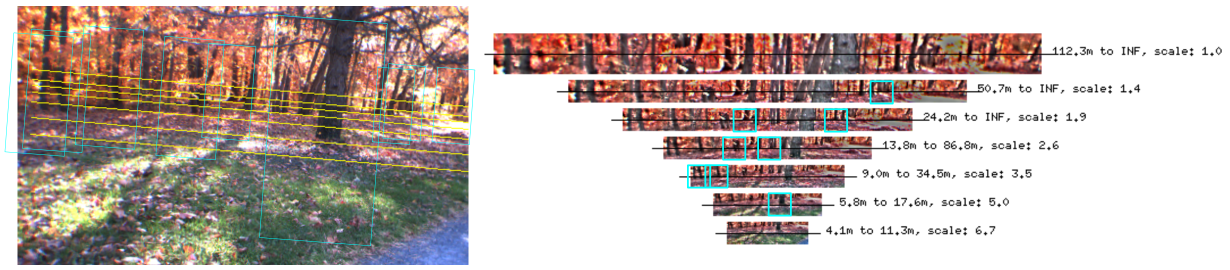

112.3m to INF, scale: 1.0
50.7m to INF, scale: 1.4
24.2m to INF, scale: 1.9
13.8m to 86.8m, scale: 2.6
9.0m to 34.5m, scale: 3.5
5.8m to 17.6m, scale: 5.0
4.1m to 11.3m, scale: 6.7

Figure 4: The input image at left has been systematically cropped, leveled and subsampled to yield each pyramid row seen to the right. The bounding boxes demonstrate the effectiveness of the normalization: trees that are different scales in the input image are similarly scaled in the pyramid.

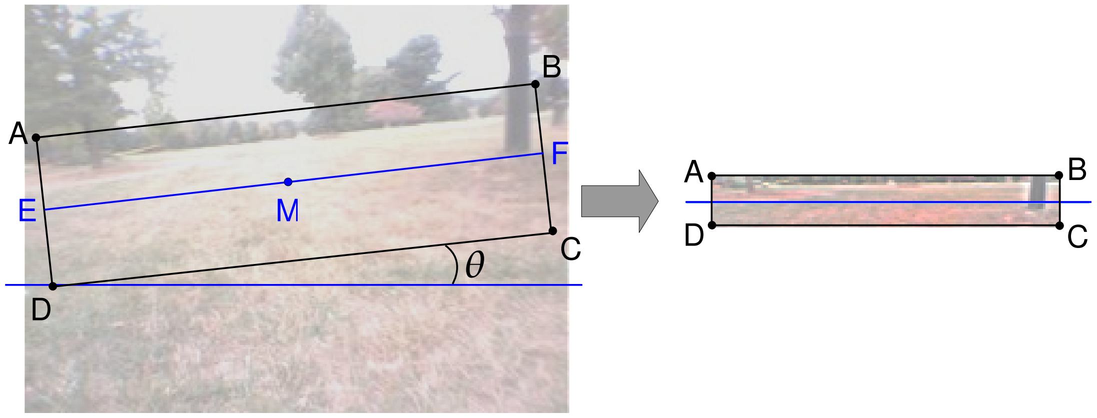


Figure 5: Each row in the normalized, horizon-leveled pyramid is created by identifying the 4 corners of the target sub-image, which must be aligned with the ground plane and scaled according to the target distance, and then warping to a re-sized rectangular region.

by explicitly estimating the horizon's location and then warping the image in relation to it. First we estimate the groundplane $P = (a, b, c, d)$ using a Hough transform on the stereo point cloud, then refine that estimate using a PCA robust refit (see Section 6.2 for details). P is converted to (p_r, p_c, p_d, p_o) format (p_r = *row*, p_c = *column*, p_d = *disparity*, p_o = *offset*), and the horizon can be leveled by computing the four corners of the target sub-image (points A, B, C,and D in Fig. 5) and transforming that sub-image to a scaled target rectangle using an affine warp. For a row in the pyramid, we first compute the location of $\overline{\mathrm{EF}}$ that lies on the ground plane at a distance with stereo disparity of d pixels. The endpoints of $\overline{\mathrm{EF}}$ are found by computing the center of the line (M) by plane intersection, then finding the endpoints (E,F) by θ rotation:

$$\mathrm{M}_y = \frac{p_c\,\mathrm{M}_x + p_d\,d + p_o}{-p_r} \tag{1}$$

$$\mathrm{E} = (\mathrm{M}_x - \mathrm{M}_x \cos\theta\ ,\ \mathrm{M}_y - \mathrm{M}_x \sin\theta) \tag{2}$$

$$\mathrm{F} = (\mathrm{M}_x + \mathrm{M}_x \cos\theta\ ,\ \mathrm{M}_y + \mathrm{M}_x \sin\theta) \tag{3}$$

where M_x is the horizontal center of the image, and θ is found by projecting the left and right columns of the image:

$$\theta = \left(\frac{\mathrm{w}p_c + p_d + p_o}{-p_r} - \frac{p_d + p_o}{-p_r} \right) /\mathrm{w}, \tag{4}$$

where W is the width of the input image. Points A, B, C, and D can be found by rotating and scaling E and F by the scaling value (α) for the pyramid row:

$$\mathrm{A} = (\mathrm{E}_x + \alpha\sin\theta,\ \mathrm{E}_y - \alpha\cos\theta) \tag{5}$$

$$\mathrm{B} = (\mathrm{F}_x + \alpha\sin\theta,\ \mathrm{F}_y - \alpha\cos\theta) \tag{6}$$

$$\mathrm{C} = (\mathrm{F}_x - \alpha\sin\theta,\ \mathrm{F}_y + \alpha\cos\theta) \tag{7}$$

$$\mathrm{D} = (\mathrm{E}_x - \alpha\sin\theta,\ \mathrm{E}_y + \alpha\cos\theta) \tag{8}$$

The input is also converted from RGB to YUV, and the Y (luminance) channel is contrast normalized to alleviate the effects of hard shadow and saturation. The contrast normalization performs a smooth neighborhood normalization on each y in Y by normalizing by the linear sum of a smooth 16x16 kernel and a 16x16 neighborhood of Y (centered on y). Pixel x in image I is normalized by the values in a soft window centered on x:

$$x = \frac{x}{\sum_{y \in I^W, k \in K} yk + 1}, \tag{9}$$

where I^W is a 16x16 window in I, and K is a smooth, normalized 16x16 kernel.

5 Feature Learning

Normalized overlapping windows (3@25x12 pixels) from the pyramid rows provide a basis for strong near-to-far learning, but the high dimensionality makes it infeasible to directly train a classifier on the YUV windows. Feature extraction lowers the dimensionality while increasing the generalization potential of the classifier. There are many ways that feature extraction may be done, from hand-tuned feature lists, to quantizations of the input, to learned features. We prefer to use learned features, because they can capture patterns directly from the data. A trained feature representation can encode most of the information in the input if it is trained to have a low reconstruction error, and a supervised learning algorithm can result in a highly discriminative feature representation.

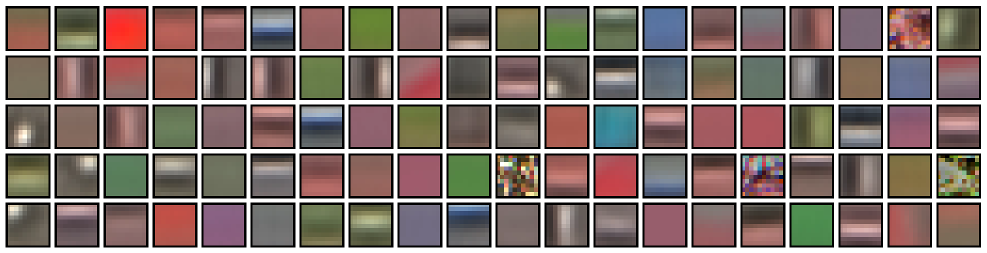


Figure 6: The 100 radial basis function centers used for feature extraction. The centers were learned through unsupervised k-means clustering on a set of 500,000 diverse patches from log files.

The convolutional network is the only component of our system that is trained offline rather than online. The offline data set consists of samples, either labeled or unlabeled, taken randomly from 130 diverse logfiles. Image pre-processing and stereo labeling of these samples was identical to the online process described in other sections of this paper (see Sections 4 and 6).

We have experimented with 4 learned feature extractors. In each case, the feature extractor was trained offline using data samples taken from a set of diverse logfiles. The trained feature extractor is then used online to produce a low-dimensional feature vector for each input window. Two of the methods are based on unsupervised learning, and two involve supervision. Each method is computationally equivalent, as compared in the number of multiply-add operations and also as bounded by the same realtime processing limitations. Performance of the 4 feature extractors is compared in Section 8.

5.1 Radial Basis Functions

Our first approach for feature learning is to learn a set of radial basis functions from data. For each 10x10 YUV input window in the normalized image pyramid, a feature vector is constructed by taking Euclidean distances between the input window and each of 100 fixed RBF centers. For an input window X and a set of n radial basis centers $\mathcal{K} = \{K^i \mid i = i..n\}$, the feature vector D has n components where:

$$D_j = \exp(-\beta^i ||X - K^i||_2^2) \tag{10}$$

where β^i is the inverse variance of RBF center K^i. The centers $\mathcal{K}$ are learned separately with k-means unsupervised learning, using a wide spectrum of logfiles from different environments (see figure 6). The width of RBF K^i, β^i, is the inverse variance of the points that clustered to it during k-means learning.

5.2 Convolutional Neural Network

A convolutional network (LeCun and Bengio, 1995) contains local receptive fields that are trained to extract local features and patterns. Convolutional nets use weight-sharing over the input area, which naturally produces invariance to translation and some rotation. Subsampling, or pooling, between convolutional layers increases the shift and scale invariance while reducing computational complexity, and the hierarchical architecture creates complex features. This architecture makes

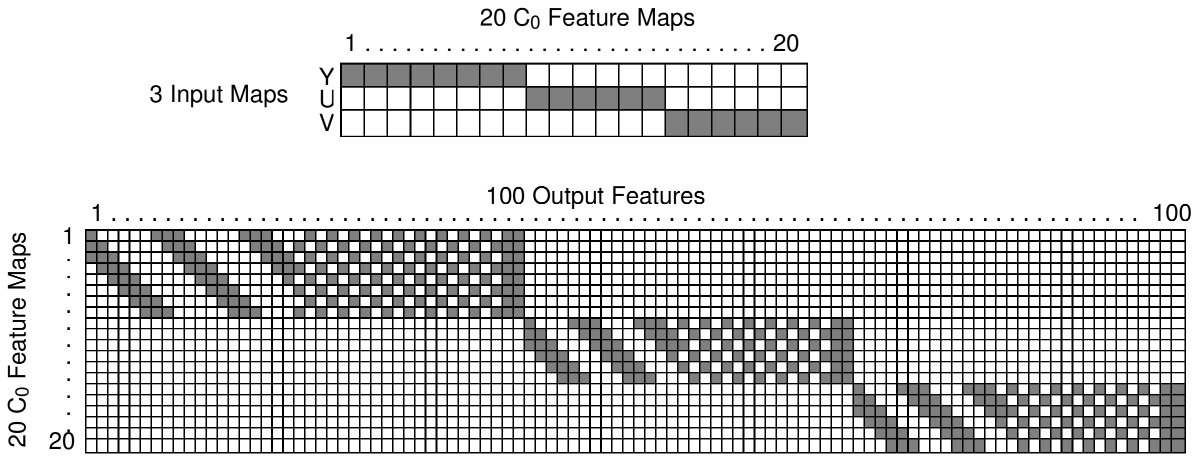


Figure 7: The top table shows the connections between the YUV input layers and the 20 feature maps in the first layer. Note that this is not a fully connected network; 8 filters are connected to the Y channel and 6 features are connected to each of the U and V channels. The second layer connections are also shown (*bottom*). The table shows connections between the 20 first layer feature maps and the 100 output features. As in the first layer, connections between Y, U, and V feature maps were separated.

convolutional nets naturally shift and scale invariant, and therefore desirable for learning robust, discriminative visual features.

This particular network has two convolutional layers and one subsampling layer. The first convolutional layer has 20 7x6 filters and the second layer has 369 6x5 filters. The layers are not fully connected; in particular, Y, U, and V filters are kept separate throughout the 2 layers. The connections between the input and the first layer, and from the first layer to the second, are shown in Fig. 7. For the purposes of training, a final 100 component, fully-connected layer is trained with 5 outputs. After the network is trained, the fully connected layer is removed such that the *online* output of the network is a 100 component feature vector.

The network was initialized with random values and trained for 30 epochs using stochastic gradient descent and $L2$ regularization. The training set contained 450,000 labeled image patches and the test set contained 50,000 labeled patches.

5.3 Convolutional Auto-Encoder

The next feature extraction approach uses some of the principles of *deep belief network training* (Hinton et al., 2006; Ranzato et al., 2007). The basic idea behind deep belief net training is to pre-train each layer independently and sequentially in unsupervised mode using a reconstruction criterion to drive the training, then retrain the same filters using supervision and labeled data to "fine-tune" the network. The deep belief net trained for the long-range vision system consists of 3 stacked modules. The first and third modules are convolutional layers, and the second layer is a

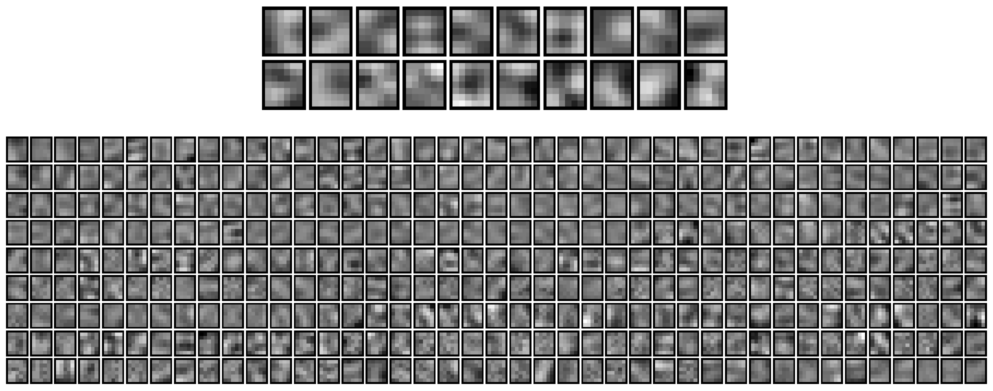


Figure 8: Trained filters from both layers of the trained convolutional auto-encoder. *Top:* the first convolutional layer has 20 7x6 filters. *Bottom:* the second convolutional layer has 369 6x5 filters.

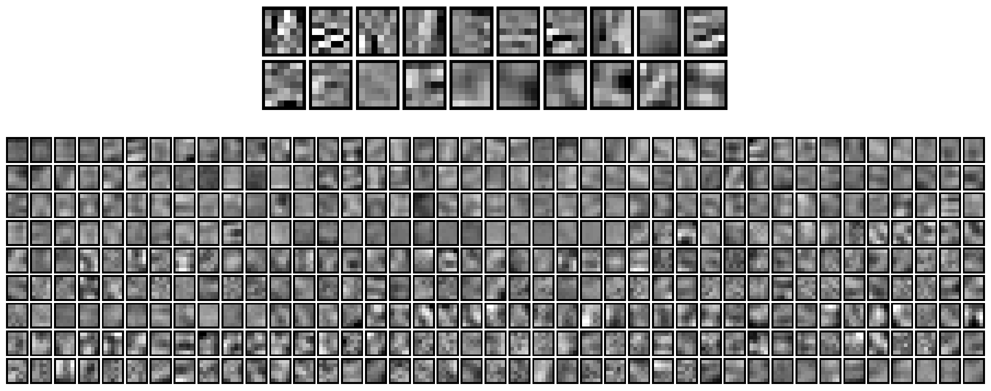


Figure 9: These features were trained using layer-by-layer unsupervised training with an auto-encoder, followed by top-down supervised fine tuning. Note the difference between these filters and the purely unsupervised ones in Figure 8 (which were used to initialize these filters before supervised training): the supervised training has altered the filters, but not substantially. In fact, we hypothesize that the new patterns are economic additions to the unsupervised filters, augmenting them with precisely the detectors needed to recognize footlines and key visual features.

max-pooling unit. Each convolutional layer can be understood as an *encoder* $F_{enc}(X)$ that creates a set of features from the given input by applying a sequence of convolutional filters. A *decoder* $F_{dec}(Y)$ tries to recreate the input from the feature vector output. The encoder and decoder are trained simultaneously by minimizing the reconstruction error, i.e., minimizing the mean square loss between the input and the encoded and decoded reconstruction:

$$\mathcal{L}(S) = \frac{1}{P}\sum_{i=1}^{P} ||X^i - F_{dec}(F_{enc}(X^i))||_2^2, \tag{11}$$

where S is a dataset with P training samples.

In each convolutional layer, the function computed for an input layer x and filter f and output feature map z is

$$z_j = \tanh(c_j(\sum_i x_i * f_{ij}) + b_j), \tag{12}$$

where $*$ denotes the convolution operator, i indexes the input layer, j indexes the output feature map, and c_j and b_j are multiplicative and additive constants. The max-pooling layer is used to reduce computational complexity and to pool features, creating translation invariance. The max-pooling operation, for input layer x and output map z, is

$$z_i = \max_{i \in N_i}(x), \tag{13}$$

where N_i is the spatial neighborhood for the max operator.

The first layer of the auto-encoder has 20 7x6 filters, and 20 feature maps are output from the layer. After max-pooling with a kernel size of 1x4, the pooled feature maps are input to the second convolutional layer, which has 300 6x5 filters and produces 100 feature maps. Each overlapping window in the input has thus been reduced to a 100x1x1 feature vector. The feature maps for a sample input (a row in the normalized pyramid) are shown in Fig. 10.

The feature extractor was trained until convergence with the reconstruction described above. The training set was composed of images from 150 diverse outdoor settings, comprising 10,000 images in total, with each image further scaled to different resolutions. The convolutional filters are shown in Fig. 8. The filters were then “fine-tuned” with supervision from a labeled dataset. Some of the filters remain very similar, while others are substantially rewritten (see Figure 9). This learning approach can be very beneficial if there is not enough labeled training data to adequately learn the best feature representation for the target domain.

6 Stereo Supervision

The supervision that the long-range classifier receives from the stereo module is critically important. The realtime training can be dramatically altered if the data and labels are changed in small ways, or if the labeling becomes noisy. Therefore, the goal of the supervisor module is to provide data samples and labels that are *visually consistent*, *error-free*, and *well-distributed*. The basic approach begins with a disparity point cloud generated by a stereo algorithm, and has 4 steps:

1. A **Stereo Algorithm** produces a 3D point cloud.

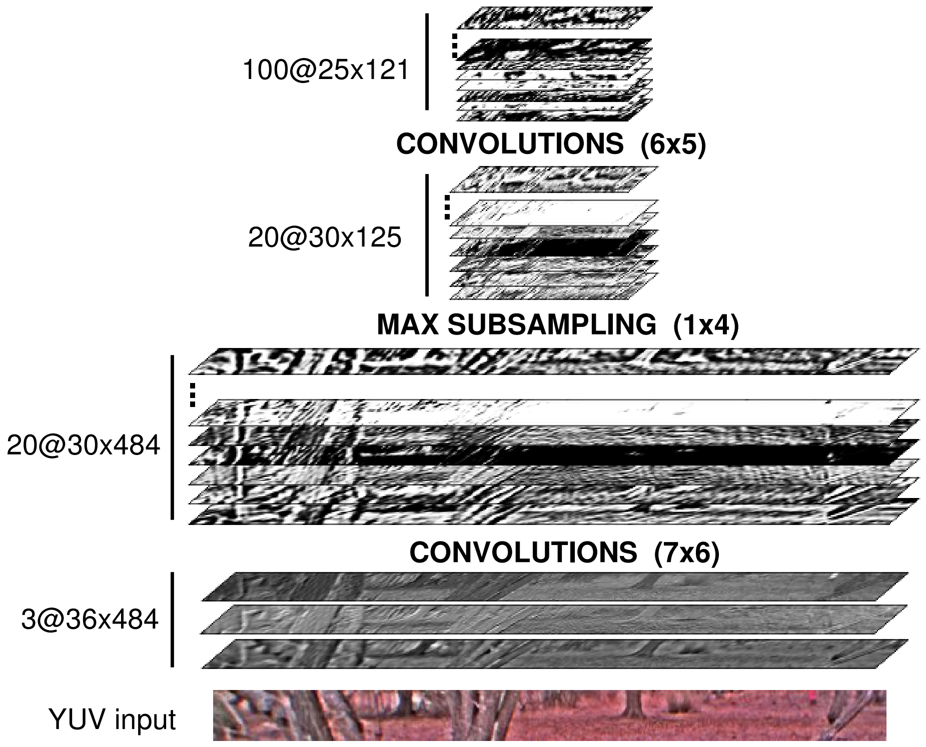


Figure 10: The feature maps are shown for a sample input. The input to the network is a variable width, variable height layer from the normalized pyramid. The output from the first convolutional layer is a set of 20 feature maps, the output from the max-pooling layer is a set of 20 feature maps with width scaled by a factor of 4 through pooling, the output from the second convolutional layer is a set of 100 feature maps. A single 3x12x25 window in the input corresponds to a single 100x1x1 feature vector.

2. **Estimation of the ground plane** within the point cloud is done, allowing separation of the points into ground and obstacle classes.
3. **Projection** Next, the obstacle points are projected onto the ground plane to locate the feet of obstacles.
4. **Labeling**. Third, overlapping regions of points are considered and heuristics are used to assign each region to one of five categories.

The results from this basic method have several sources of error, however. Since off-road terrain is rarely perfectly flat, areas of traversable ground can stick up above the ground plane and be misclassified as obstacle. Also, tufts of grass or leaves can look like obstacles when a simple plane distance threshold is used to detect obstacles. These sorts of error are potentially disastrous to the classifier, so two strategies, multi-groundplane estimation and moments filtering, are employed to avoid them. This section describes the stereo algorithm and the supervision process in detail.

6.1 The Stereo Algorithm: Triclops

The LAGR vehicle is equipped with 2 sets of Bumblebee stereo cameras, one for the left field and one for the right field, which together comprise a field of view of 120°. The Triclops SDK, from Point Grey Research, provides image rectification and stereo processing on each pair of cameras (Inc., 2003). Rectification is done to remove lens distortions and misalignment. Stereo processing produces a range estimate for each valid pixel in the field of view, and is limited by the resolution of the input images; our system uses 384x512 images, which provides a range of 12 to 15 meters with the Triclops algorithm. The Triclops algorithm works by triangulating between 2 slightly offset cameras: correspondences between pixels in the two images are found and distances are calculated based on the geometry of the cameras. The correlation is made with a sum of absolute differences over band-passed images. The algorithm fails if sufficient texture and contrast is not present in the images, and can be fooled by repeating visual elements (fenceposts, tall grass) and visual artifacts (sun glare, specularities).

6.2 Ground Plane Estimation

It is necessary to locate the ground plane in order to distinguish ground from obstacle points within a stereo point cloud. However, the assumption that there is a single perfect ground plane is rarely correct in natural terrain. To relax this assumption, we find multiple planes in each input image and use their combined information to divide the points into ground and obstacle clouds. First we describe how to fit a single ground plane model to a point cloud, where the point cloud is a set of 6-tuples: $\mathcal{S} = \{(x^i, y^i, z^i) \mid i = 1..n\}$ where x^i, y^i, z^i defines the position of the point relative to the robot's center. Color components (r^i, g^i, b^i) and image relative coordinates $(\text{row}^i, \text{column}^i, \text{disparity}^i)$ are also associated with each point in $\mathcal{S}$.

Ground plane estimation is done in two stages: an initial estimate is found using a Hough transform, and principal component analysis is used to refine the estimate. A Hough transform (Duda and Hart, 1972) is a voting procedure that is used to select a shape from within a parameterized class of shapes (in this case, the class of planes). A quantized parameter space defines the set of possible planes, and points in the cloud "vote" for a single plane. The winning plane is defined

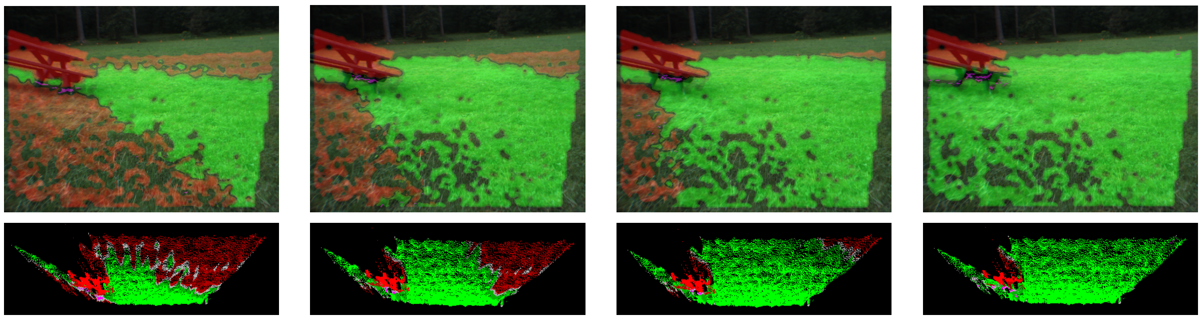


Figure 11: Up to 3 ground planes are found in the stereo point cloud of input images. After a plane is found, points close to the plane are removed from the point cloud and the process is repeated (a, b, c). After ground plane estimation, statistical analysis of the plane distances of the points is done to filter out remaining false obstacles (d).

by the parameter vector that accumulates the most votes. The ground plane parameter space is constrained in pitch, roll, and offset, and quantized into 64 bins for each parameter. The bounding of each parameter was governed by consideration of the maximum slope on which the robot could reasonably drive.

The exact voting process begins by randomly selecting a subset of points from $\mathcal{S}$. As the algorithm iterates each of these points, a point (x^i, y^i, z^i) in $\mathcal{S}$ votes for each of the candidate planes that it intersects. This can be done efficiently: we pre-compute all possible normals $[abc]$, then increment the tally for the candidate with parameters $a, b, c, (x^i a + y^i b + z^i c)$ if all the parameters are within the defined bounds. The yaw parameter c is fixed at 1.0, so the space exploration is only in the 3 dimensions of pitch, roll and offset. After the voting process is done, a plane is selected by finding the maximum within the voting space:

$$X = P_{ijk} \;\; | \; i, j, k = \text{argmax}_{i,j,k} \left(V_{ijk} \right), \tag{14}$$

where X is the new plane estimate, V is a tensor that accumulates the votes, and P is a tensor that records the plane parameter space.

The Hough estimate of the ground plane is then refit using principal component analysis. The PCA refit begins with the set of *inlier* points that are within a threshold of the Hough plane, then computes the eigenvalue decomposition of the covariance matrix of the points $X^{1..n}$:

$$\frac{1}{n} \sum_{1}^{n} X^i X^{i\prime} = Q \Lambda Q' \tag{15}$$

The eigenvector in Q corresponding to the smallest eigenvalue in Λ is the new plane normal, and the offset of the plane is set to the mean offset of the inlier point set.

The process of plane fitting that has been described is fast and robust, fitting a correct plane even in challenging terrain such as narrow paths where very few points on the ground can be seen. Restricting the Hough transform parameter space is necessary to prevent the plane being fit to table tops or vertical planes of buildings or cars. However, as has already been mentioned, natural terrain is often not planar. It can be non-continuous or disjoint (a street and a sidewalk, for instance) or curving (undulations of the ground, hills, and valleys). Since fitting a single plane to a non-planar surface can produce disastrous results (a hill or a valley will appear to be an obstacle if the points are far enough from the estimated plane) both for driving and for training the classifier, we fit multiple planes to each frame in the hopes of capturing the dominant facets of the terrain. The multi-plane fitting process is iterative in nature: after the first ground plane is fit to the point cloud, all points that are within a tight threshold of the plane are removed and a new plane is fit to the remaining points. The process continues until a stopping criterion is met: either (1) no valid plane can be found, (2) there are not enough points left in the point cloud, or (3) a maximum of 3 planes have been fit to the data.

More formally, we distinguish the set of "ground" points as a subset of the full point cloud: $\mathcal{S}^G \subseteq \mathcal{S}$, where $\mathcal{S}^G$ denotes the set of ground points. Given a set of m planes, $\mathcal{P} = \{P^i \mid i = 1..m\}$, points can be assigned to $\mathcal{S}^G$ based on their normal distance from the planes and a threshold α:

$$\mathcal{S}^G = \{(x^i, y^i, z^i) \mid D(X^i, P^j) \leq \alpha\}, \tag{16}$$

where D computes the Euclidean distance between point X^i and its projection on plane P^j:

$$D(X^i, P^j) = |x^i a^j + y^i b^j + z^i c^j + d^j| \tag{17}$$

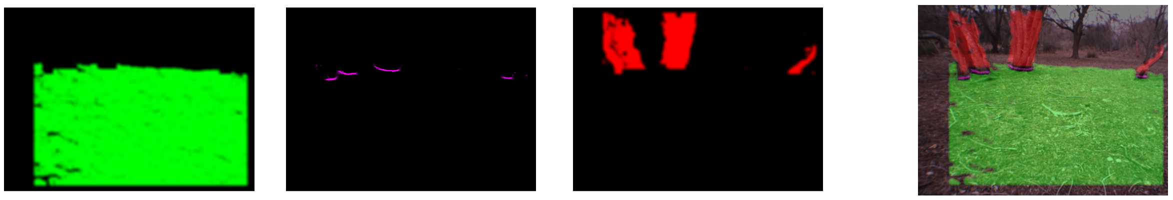
Fig. 6.2 shows the results of multiple plane fitting on a single point cloud. Frame *1* shows the point cloud in image space (top) and in 3d (bottom) with a single plane fit to the scene. Frame *2* shows the scene with a second plane fit, and Frame *3* shows the scene with 3 planes. The fourth panel shows the final classification of the points, after filtering by moments has been done. We discuss moments filtering now.

Even multiple ground planes cannot remove all error from the stereo labeling process. Tufts of grass or disparity mismatches (common with repeating textures such as tall grass, dry brush, or fences) can create false obstacles that cause poor driving and training. Therefore, our strategy is to consider the first and second moments of the plane distances of points and use the statistics to reject false obstacles. We use the following heuristics: if the mean plane distance is not too high and the variance of the plane distance is very low, then the region is traversable (probably a traversable hillside). Conversely, if the mean plane distance is very low but the variance is higher, then that region is traversable (possibly tall grass). These heuristics are simple, but they reduce errors in the training data effectively. The process is described in more detail.

The *plane distance* of each point in $\mathcal{S}$ is computed by projecting the point onto each plane and recording the minimum distance: $pd(X^i, \mathcal{P}) = min_{P \in \mathcal{P}}(x^i a + y^i b + z^i c + d)$, where $X^i = (x^i, y^i, z^i)$ is a point in $\mathcal{S}$ and $P = (a, b, c, d)$ defines a plane in $\mathcal{P}$. We next compute the mean and variance of non-overlapping regions of point plane distances, where the regions are defined in image space. Since each point is associated with its *(row, column)* indices, finding the points that correspond to a particular region is straightforward. For a region A centered on (r, c) (dimension was 10x10 pixels), the mean and variance follow:

$$\mu = \overline{A_{rc}} = \frac{1}{|A_{rc}|} \sum_{X^i \in A_{rc}} pd(X^i, \mathcal{P}) \tag{18}$$

$$\nu = \mathrm{Var}(A_{rc}) = \frac{1}{|A_{rc}|} \sum_{X^i \in A_{rc}} [pd(X^i, \mathcal{P}) - \mu]^2. \tag{19}$$



a. Ground map (**G**) b. Footline map (**F**) c. Obstacle map (**O**) d. Composite, labeled image

Figure 12: Our process for correct labeling of an image is dependent on robustly separating the stereo points into 3 subsets: ground points, footline points, and obstacle points. After these subsets are identified, final labels can be assigned, yielding the 5-category labeled image.

Proceeding according to the heuristics described previously, a low and high threshold was established for both mean and variance and each region was analyzed. If ($\mu <$ low-mean $\wedge$ $\nu <$ high-var) or ($\mu <$ high-mean $\wedge$ $\nu <$ low-var), then the points within that region are moved to the traversable point cloud ($\mathcal{S}^G$). This false-obstacle filtering reduces the noise in the stereo supervision substantially. It is useful when the curvature of the ground exceeds the modeling capacity of the 3 ground planes, or when tufts of grass or leaves poke above an otherwise flat plane. Frame 4 of Fig. 6.2 shows the effect of moments filtering on a single frame of difficult, hilly terrain.

6.3 Footline Projection

Identifying the footlines of obstacles is critical for the success of the long-range vision classifier. Footlines are not only visually distinctive and thus relatively easy to model, they are also, by definition, at ground level, and thus we have more confidence in their exact location when they are mapped into 3d coordinates. There is a fundamental uncertainty about the exact distance of points that are beyond the range of stereo, and points that belong to obstacles are especially uncertain, because their height above the ground gets projected into false distance. Footlines have less uncertainty, because we know that a footline point has a height of 0. Footlines also define the border between traversable and non-traversable regions, so they are very significant for planning purposes.

Accordingly, we are not satisfied with robustly separating ground and obstacle point clouds ($\mathcal{S}^G$ and $\mathcal{S} - \mathcal{S}^G$). We find footlines by projecting obstacle points onto the ground planes. Concentrations of projected points are recognized as footlines. Any roughly vertical obstacle will project enough points onto the ground to be recognized as a footline. A gradually rising obstacle, however, such as a gentle transition from grass to scrub to bushes to trees, will probably not project enough points to form a footline. This is acceptable; often gradually transitioning terrain does not have a visually identifiable footline and thus makes poor training examples. For these areas, we can rely on our recognition of obstacles and forgo footline classification. To find footline points, each point in $\mathcal{S} - \mathcal{S}^G$ is projected onto the nearest ground plane in $\mathcal{P}$ and its *(row, column)* image space coordinates are recorded.

After ground plane estimation and footline projection, we are left with 3 sets of points: ground, obstacle, and footline. Each of these sets is mapped into the image plane, yielding 3 labeled "maps", which we denote **G** (ground-map), **F** (footline-map), and **O** (obstacle-map) (see Fig. 12). The final step toward stereo supervision considers the distribution of points within each overlapping window in the 3 maps, where the kernel size is the same as the input window size of the classifier. In order

Category	Criteria for Label
Super-Traversable	*high number of ground points, very few obstacle and footline points.* **if** $(\mathbf{G}_{ij} > 0.8)$ **and** $(\mathbf{O}_{ij} < 0.1)$ **and** $(\mathbf{F}_{ij} < 0.01)$ **then** $Y_1 = 1$ **end if**
Traversable	*some ground points, few footline points.* **if** $(\mathbf{G}_{ij} > 0.4)$ **and** $(\mathbf{F}_{ij} < 0.1)$ **then** $Y_2 = \mathbf{G}_{ij}$ **end if**
Footline	*high number of footline points, some ground points (this avoids labeling hidden footlines).* **if** $(\mathbf{F}_{ij} > 0.6)$ **and** $(\mathbf{G}_{ij} > 0.1)$ **then** $Y_3 = 1$ **end if**
Obstacle	*some obstacle points, few footline points.* **if** $(\mathbf{O}_{ij} > 0.4)$ **and** $(\mathbf{F}_{ij} < 0.1)$ **then** $Y_4 = \mathbf{O}_{ij}$ **end if**
Super-Obstacle	*high number of obstacle points, very few ground and footline points.* **if** $(\mathbf{O}_{ij} > 0.8)$ **and** $(\mathbf{G}_{ij} < 0.1)$ **and** $(\mathbf{F}_{ij} < 0.01)$ **then** $Y_5 = 1$ **end if**

Table 1: The rules for assigning a label to a window centered at (i, j) are given, based on the weighted averages in three point maps: **G** (ground-map), **F** (footline-map), and **O** (obstacle-map).

to take the context of the window's full field of view into consideration while emphasizing the content at the center of the window, weighted averages of the points in **G**, **F**, and **O** are computed, where the averages are peaked at the center of the window. This is efficiently done by convolving a separable Gaussian kernel over the 3 maps. Given the weighted average of ground, obstacle, and footline points present in a window whose center is at (i, j), a series of heuristics, summarized in Table 1, is applied that assigns a label to that window.

6.4 Visual Categories

Most classifiers that attempt to learn terrain traversability are binary; they only learn ground vs. obstacle. However, our classifier uses 5 categories: *super-ground*, *ground*, *footline*, *obstacle*, and *super-obstacle*. *Super-ground* and *super-obstacle* refer to input windows in which *only* ground or obstacle points are seen, and our confidence is very high that these labels are correct. The weaker *ground* and *obstacle* categories are used when mixed types of points are seen in the window, or when the confidence is lower that the label is correct. *Footline* is the label for input windows that have the footline points centered in the window. Obstacle feet are visually distinctive, and

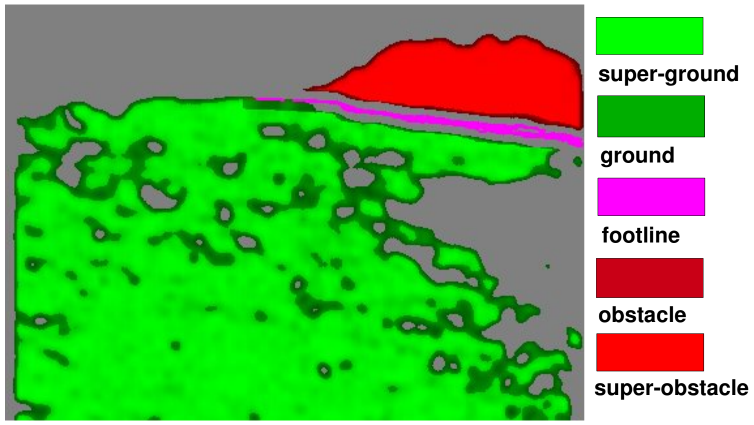


Figure 13: This shows the 5 category labeling of a full image.

it is important to put these samples in a distinct category. Fig. 13 and Fig. 14 show examples of the 5 categories. Although the examples in each of these categories are still very diverse - pavement, leaves, and hard shadows are all found in the ground class, and tree trunks, buildings, and leafy bushes are all found in the obstacle class - they are more consistent than the broad categories of a binary classifier. Also, as mentioned before, obstacle footlines are a very important category because we can have higher confidence about their projection into Cartesian coordinates. However, if a binary classification scheme is used, then obstacle footlines must be interpreted as the “unknown” output of the classifier, since the footline necessarily inhabits the threshold between the 2 categories of a binary classifier.

7 Realtime Terrain Classification

The online learning framework takes the feature vectors and supervisory labels and trains 5 binary classifiers. Since the number of labeled training samples in each category can vary widely (open lawn vs. dense forest, for example), we use 5 ring buffers to accumulate up to 1000 training samples of each category. This not only balances the training between the multiple classes, but also acts as a rudimentary short-term memory: we train on several frames worth of data at each timestep, so we “remember” obstacles and groundtypes for at least 2 timesteps, and as many as 30 timesteps, after our last direct sighting of them. Data is removed from its ring buffer if it persists for more than 30 timesteps; this prevents overtraining.

Since the online classifier trains from and then classifies every frame that it receives, it must be simple and efficient. A separate logistic regression is trained on each of the 5 categories, using a modified one-against-the-rest training strategy in which overlapping categories are not trained discriminatively against each other - i.e., super-traversable and traversable samples are not presented to the same classifier as positive and negative examples, nor are super-obstacle and obstacle. For a feature vector **x**, we compute the output of each regression i through its weight vector, bias, and a logistic sigmoid function:

$$q_i = f(\mathbf{w_i}\mathbf{x} + b), \quad \text{where} \quad f(z) = \frac{1}{1+e^{-z}} \tag{20}$$

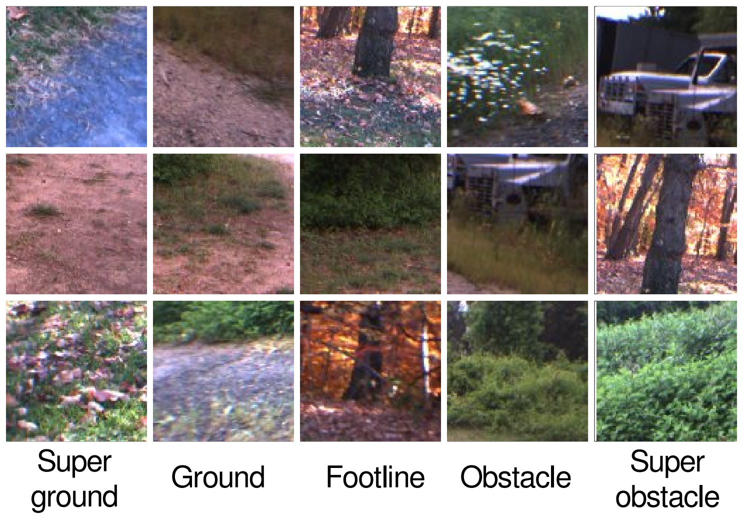


Figure 14: Examples of the 5 categories. Although the classes include very diverse training examples, there is still benefit to using more than 2 classes. *Super-ground*: only ground is seen in the window; high confidence, *ground*: ground and obstacle may be seen in window; lower confidence, *footline*: obstacle foot is centered in window, *obstacle*: obstacle is seen but does not fill window; lower confidence, and *super-obstacle*: obstacle fills window; high confidence.

The labeled feature vectors can be used to train the regressions by minimizing a loss function. The loss function that is minimized is the Kullback-Liebler divergence or relative entropy:

$$D_{KL}(P||Q) = \sum_{i=1}^{K} p_i \log p_i - \sum_{i=1}^{K} p_i \log q_i, \tag{21}$$

where p_i is the probability that the sample belongs to class i, as given by the stereo supervisor labels. q_i is the classifier's output for the probability that the sample belongs to class i. The loss for each binomial regression is

$$Loss_i = -p_i \log q_i - (1 - p_i) \log(1 - q_i) \tag{22}$$

The weights of each regression are updated using stochastic gradient descent, since gradient descent provides strong regularization over successive frames and training iterations. The gradient update for the ith regression with training sample $\mathbf{x}$ is

$$\Delta \mathbf{w_i} = -\eta \frac{\partial Loss}{\partial \mathbf{w_i}} = -\eta \left(p_i - q_i\right) \mathbf{x} \tag{23}$$

The regressions are trained, using all the samples in the ring buffer, for one epoch on each timestep. After training, all inputs from the current frame are labeled by all 5 regressions, yielding a 5 component likelihood vector for each input. Even the stereo-labeled inputs are labeled with the trained regressions, since the resulting classifications are often smoother and more accurate than the training labels. The output of the long-range module, after training and classification, consists of a set of points given in vehicle-relative coordinates and the associated likelihood vectors.

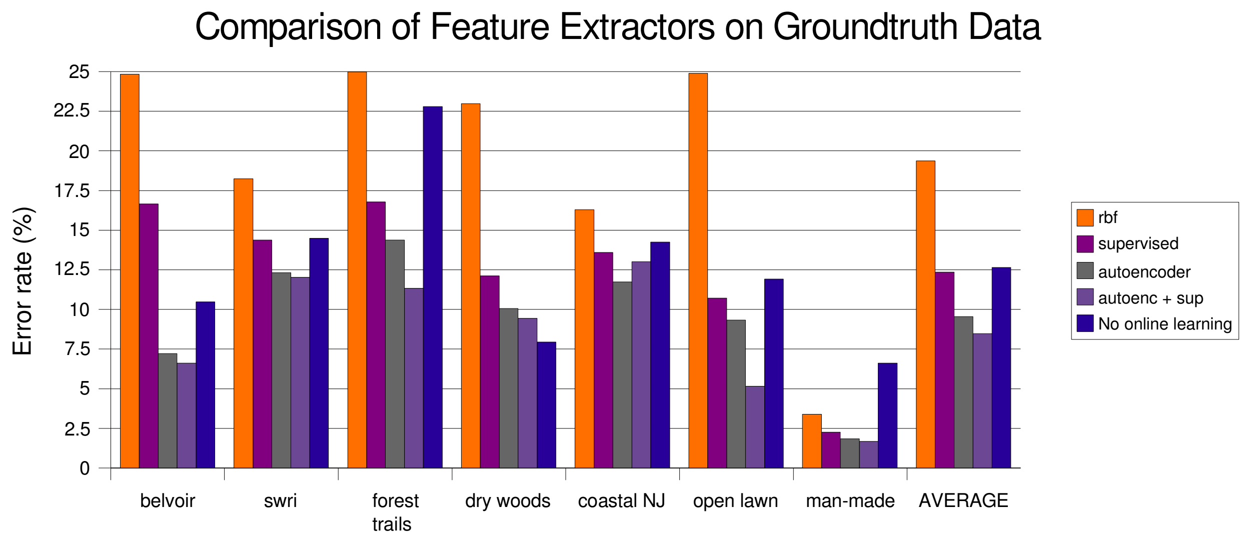


Figure 15: Error rates are given for a groundtruth data set. The logfiles in the data set are grouped into 7 sets by their terrain and geographical location. For each set, 4 different feature extractors are compared: the radial basis function network (rbf), the supervised convolutional net (supervised), the unsupervised auto-encoder (auto-encoder), and the convolutional unsupervised/supervised auto-encoder network (autoenc + sup) (see Section 5.3). The last bar shows the result of using the convolutional auto-encoder features, but not updating the logistic regression weights during the groundtruth evaluation. This comparison is included to demonstrate the effectiveness of online learning.

8 Results and Discussion

We have tested the long-range vision classifier independently as well as testing its effect on the full navigation system. Independent testing of the classifier is difficult, because the stereo supervision labels that would normally be used to judge whether the classifier is well-trained are extremely short-range and often noisy. In order to give a truer estimate of the accuracy of the classifier, we created a groundtruth data set containing 160 hand-labeled frames over 25 logfiles.

8.1 Ground Truth and Stereo Error

Fig. 15 shows the groundtruth error of the long-range classifier. The 25 groundtruth logfiles are divided into 7 groups according to the terrain and geography of the logfiles. At 12.36% error, the supervised network does not perform well on the groundtruth data, probably because of the wider data distribution found in the groundtruth dataset. The 2 purely unsupervised representations, auto-encoder and RBF, perform quite differently. The RBF is slower to adapt in general and has poor accuracy on visually complex terrain (19.37% average error). The auto-encoder does very well, but is outperformed by the hybrid version with supervised fine-tuning (9.55% vs. 8.46%). We believe that the supervision added precisely the patterns that the unsupervised features were lacking, allowing a stronger classification rate while retaining the generality of the unsupervised features. This is a significant result, demonstrating the utility of a hybrid unsupervised/supervised approach for real world problems.

The last bar in the figure shows the performance of the hybrid auto-encoder, but tested without training the classifier online. Instead the classifier weights were kept fixed on a set of general "default" weights. Not surprisingly, the no-learning classifier has difficulty on all the logfiles and at 12.64% has the second worst average error.

To quantify the accuracy of the stereo module, it was tested against the groundtruth set and found to be quite erroneous. In fact, it was less accurate, overall, than the classifier performance - surprising, since the classifier relies on the stereo module as its only source of training data. The difference in error rates for stereo module vs. classifier are shown in Figure 16. The positive data points represent frames where the classifier had a lower error rate than the training data. The classifier's improvement over the training data implies that there is noise in the training labels that is being smoothed, or regularized, by the classifier. We can also conclude that the groundtruth labels accord well with visual cues in the image. This is not surprising, of course, since the human groundtruth labeler has nothing *but* visual cues on which to base her labels.

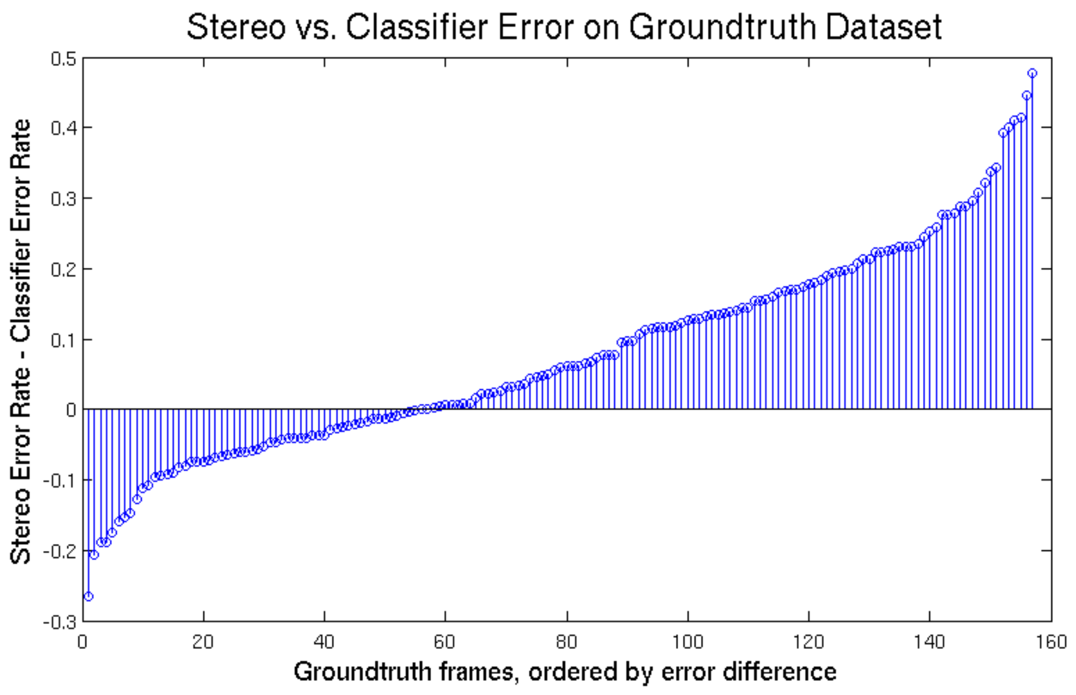


Figure 16: The difference between stereo error and classifier error is plotted, showing that the online classifier has higher accuracy than its own training data on a majority of the groundtruth frames. The positive data points represent frames where the classifier had a lower error rate than the training data.

Fig. 17 shows seven examples of long-range classifications in very different terrain. The input image, the stereo labels, and the classifier outputs are shown in each case. Note that generally the classifier output gives a better labeling even for the part of the image that can be labeled by stereo. The far-range image portion is smoothly labeled as well. In contrast to a color-based classifier, this classifier is able to recognize many different complex objects or groundtypes in the same scene.

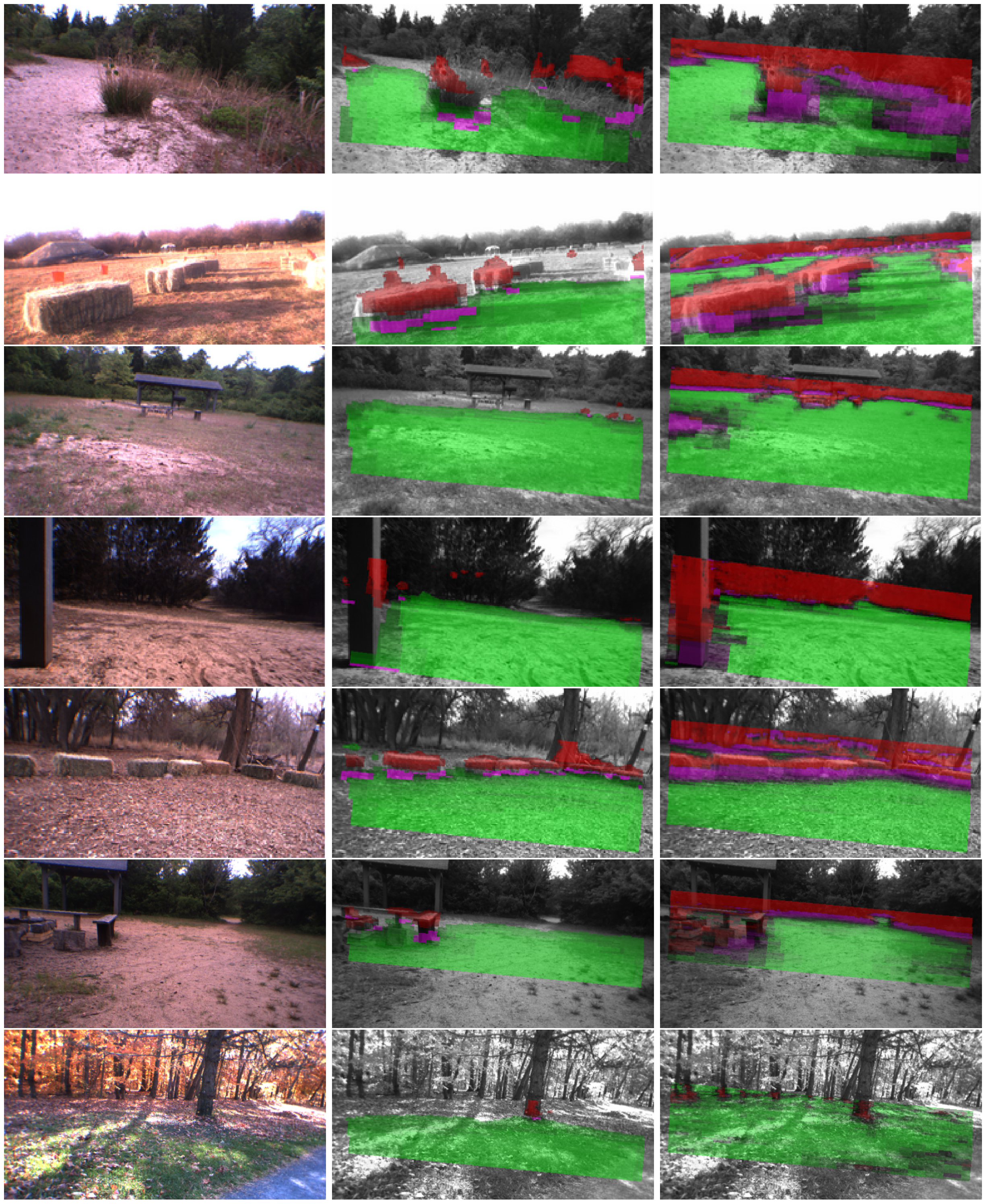


Figure 17: Qualitative examples of the success of the long-range classifier in different terrain. **Left**: RGB input; **middle**: training labels; **right**: classifier output. Green is traversable, red is obstacle, and pink is footline.

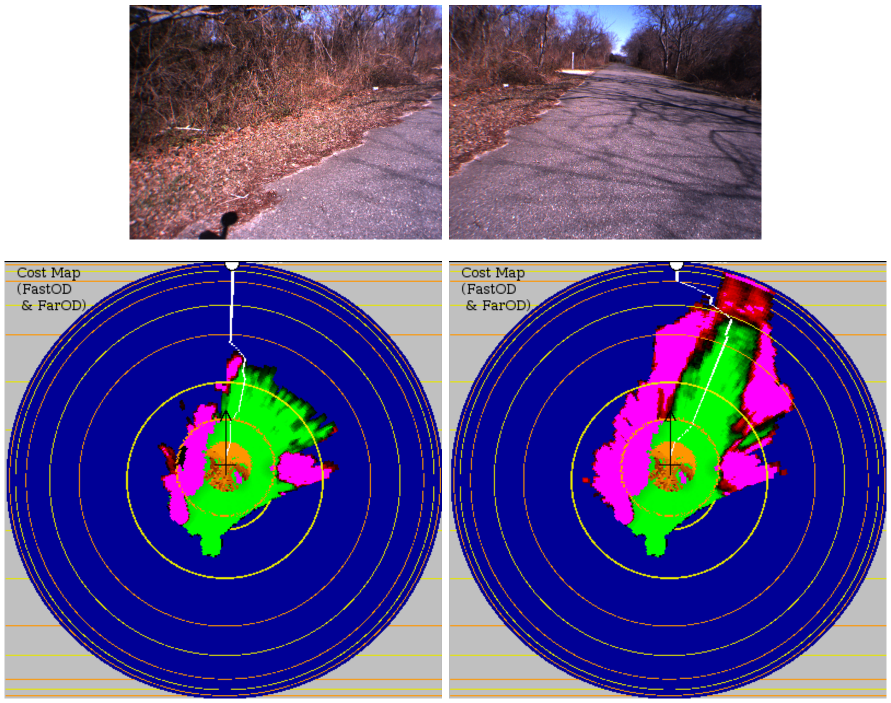


Figure 18: *Left: short-range; Right: long-range.*
On this course, the short-range system leaves the road and enters a long cul-de-sac to the left, while the long-range system proceeds down the road to the goal.

Figures 18, 19, 20, 21, 22, and 23 show examples of long-range mapping with a hyperbolic polar map, as described in (Sermanet et al., 2008a; Sermanet et al., 2008b). Each example shows the left and right input frames, the output map with short-range (10 meter) stereo vision, and the output map with long-range classifier outputs. The planned long-range route to the goal can be seen in each of the examples; the route and the goal are both white. These figures demonstrate that recognition of paths and obstacles in images can be successfully used for improved strategic navigation. They also serve as evidence for the long-range vision capabilities of the system, frequently showing successful recognition at distances of over 50 meters.

8.2 Full System Field Experiments

The long-range vision system has been used extensively in the full navigation system built on the LAGR platform. It runs at 1-2 Hz, which is too slow to maintain good close-range obstacle avoidance, so the system architecture runs 2 processes simultaneously: a fast, low-resolution stereo-based obstacle avoidance module and planner run at 8-10 Hz and allow the robot to nimbly avoid obstacles within a 5 meter radius. Another process runs the long-range vision and long-range planner at 1-2 Hz, producing strategic navigation and planning from 5 meters to the goal. Details of the full system, including architecture, short-range perception and planning, and long-range planning, are given in (Sermanet et al., 2008a).

We present experimental results obtained by running the robot on 2 courses with the long-range vision turned on and turned off. With the long-range vision turned off, the robot relies on its fast

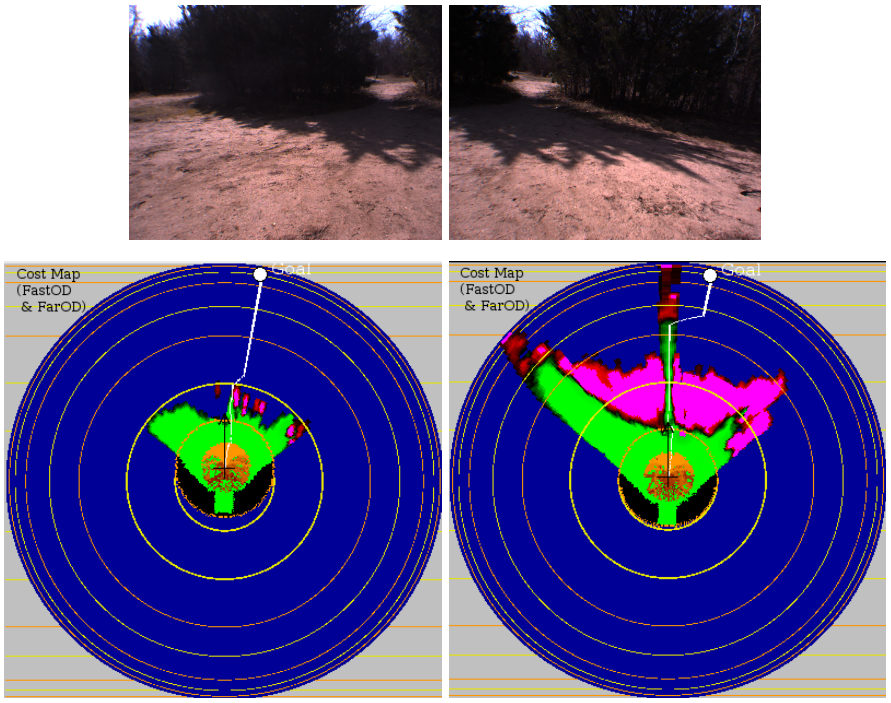


Figure 19: *Left: short-range; Right: long-range.*
Although both the short- and long-range systems find the goal, the long-range system is more robust to error and stays on track better than the short-range.

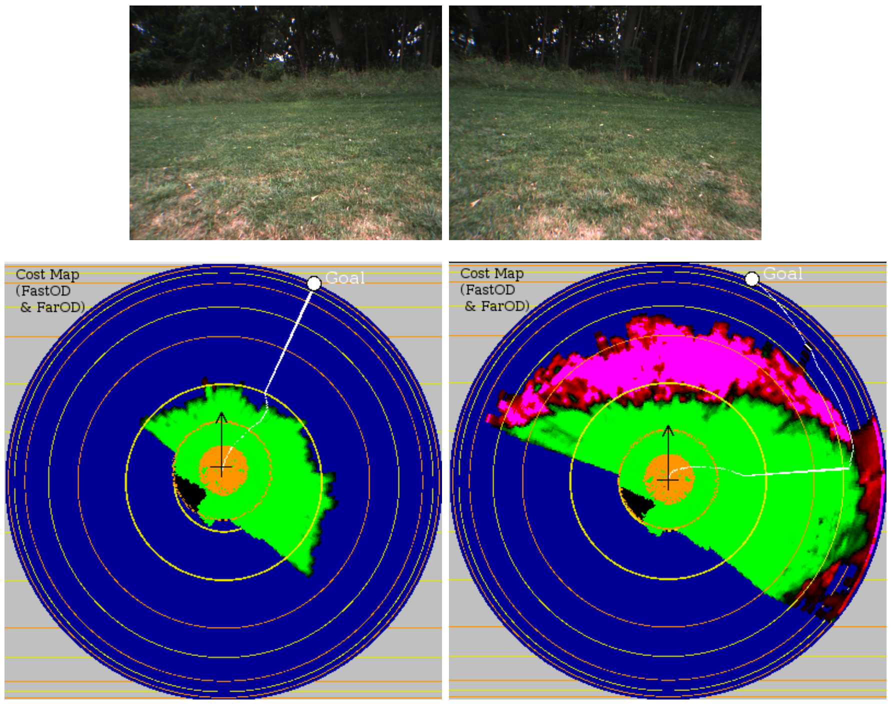


Figure 20: *Left: short-range; Right: long-range.*
This shot is looking at a very long, dense wall of trees. There is a long path around the forest to the goal, but the short-range does not see it.

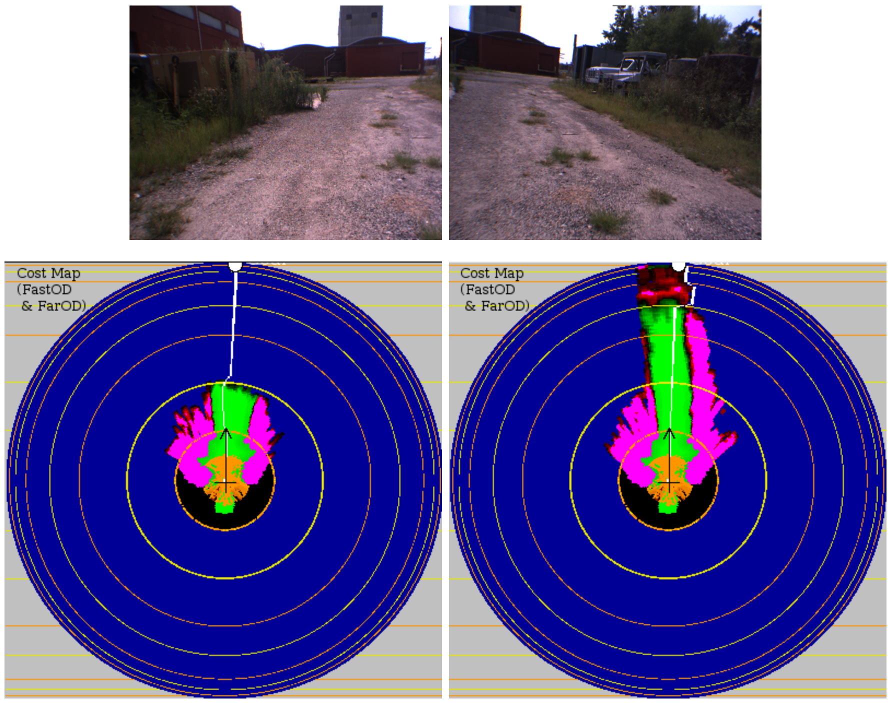


Figure 21: *Left: short-range; Right: long-range.*
This is the beginning of a long road leading around some trucks and buildings. The long-range classifier sees the route around the building from the beginning.

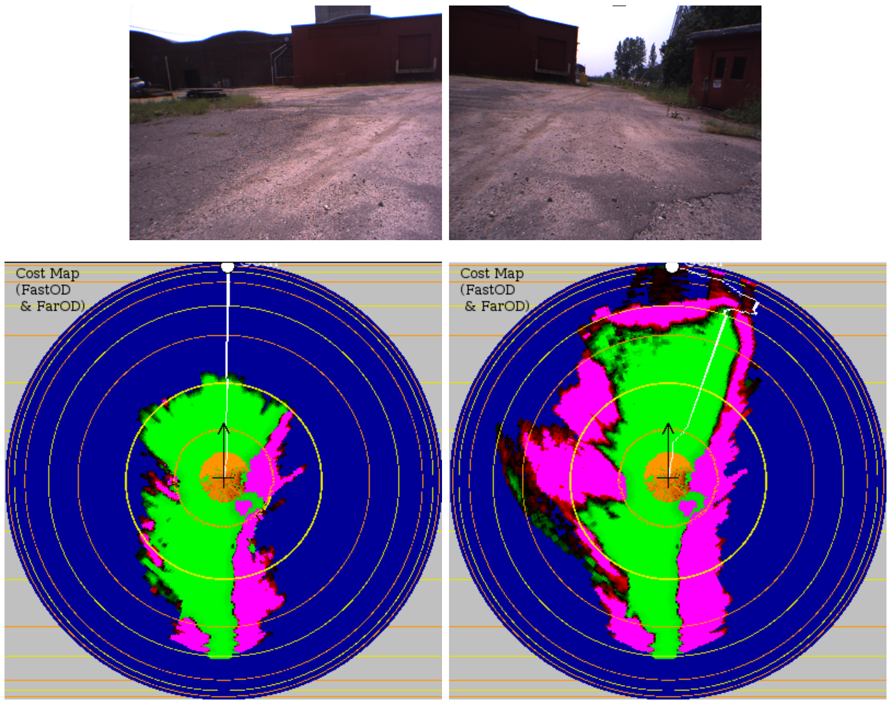


Figure 22: *Left: short-range; Right: long-range.*
This is the same course as the previous one, but closer to the buildings. The long-range system still wants to navigate around the building, and the short-range still does not see any obstacle.

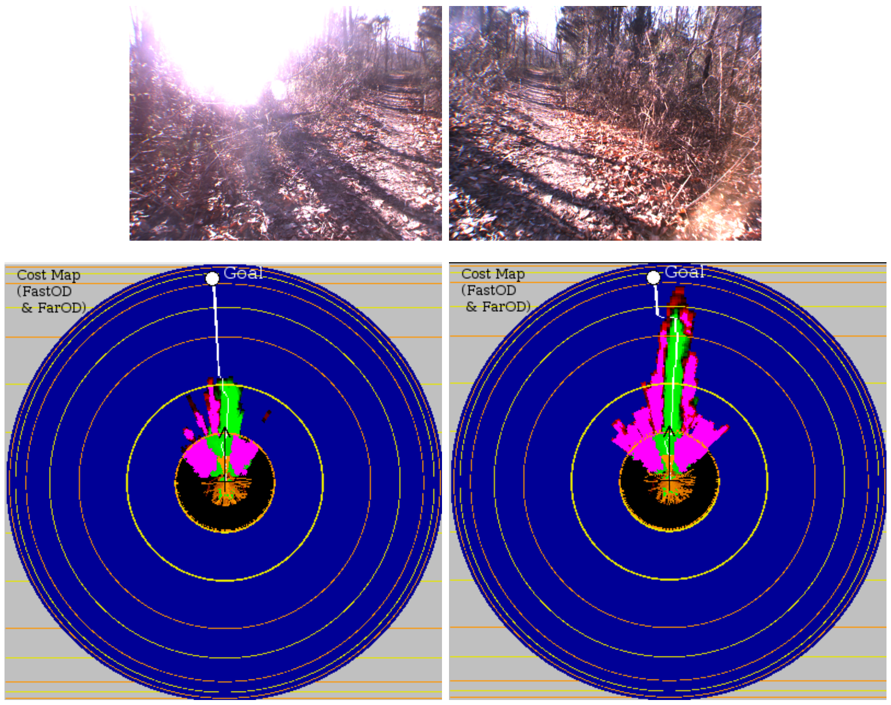


Figure 23: *Left: short-range; Right: long-range*
This is a wooded path, overgrown and difficult for the stereo supervisor (sun flares, sparse branches). Without the long-range, the system tends to leave the path and get stuck.

planning process and can only detect obstacles within 5 meters. The long-range configuration uses the auto-encoder features described in Section 5.3. Course 1 (see Fig. 24 and Table 2) is a narrow wooded path that proved very difficult for the robot with long-range vision off, since the dry scrub bordering the path was difficult to see with stereo alone. The robot had to be rescued repeatedly from entanglements off the path. With long-range vision on, the robot saw the scrub and path clearly and drove cleanly down the path to the goal. Course 2 (see Fig. 25 and Table 2) was a long wide path with a clearing to the north that had no outlet - a large natural cul-de-sac. Driving with long-range vision on, the robot saw the long path and drove straight down it to the goal without being tempted by the cul-de-sac. Driving without long-range vision, the robot immediately turned into the cul-de-sac and became stuck in scrub, needing to be manually driven out of the cul-de-sac and restarted in order to reach the goal. Course 3 (see Fig. 26 and Table 2) was a short but complex path from one clearing in the scrub to another such clearing. Although the paths taken by the short-range and long-range systems are similar, the path of the long-range system is smoother and avoids detouring down the false path.

9 Conclusions

We have described, in detail, a self-supervised learning approach to long-range vision in off-road terrain. The classifier is able to see smoothly and accurately to the horizon, identifying trees, paths, man-made obstacles, and ground at distances far beyond the 10 meters afforded by the stereo supervisor. Complex scenes can be classified by our system. We believe that the success

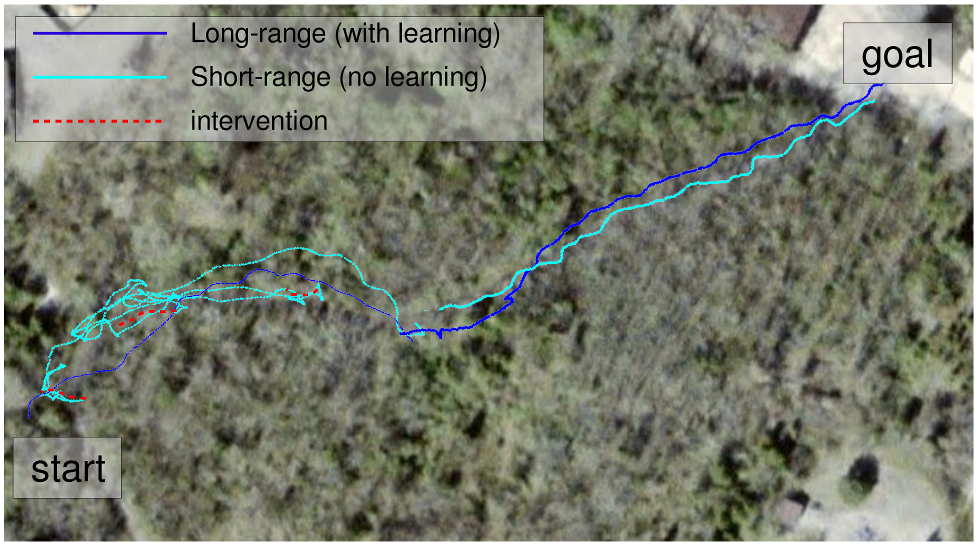


Figure 24: This was a long course through a scrubby, dense wood, following a narrow rutted path. Without long-range learning, the robot had a very difficult time staying on the path and had to be rescued from the bushes 3 times.

of the classifier is due to the use of large context-rich image windows as training data, and to the use of a deep belief network for learned feature extraction. The accuracy of the classifier has been shown through systemic field testing as well as through comparison with a hand-labeled groundtruth dataset.

Acknowledgments

The authors wish to thank Larry Jackel, Dan D. Lee, and Martial Hébert for helpful discussions. This work was supported by DARPA under the Learning Applied to Ground Robots program.

References

Angelova, A., Matthies, L., Helmick, D., and Perona, P. (2007). Dimensionality reduction using automatic supervision for vision-based terrain learning. In *Proc. of Robotics: Science and Systems (RSS)*.

Dahlkamp, H., Kaehler, A., Stavens, D., Thrun, S., and Bradski, G. (2006). Self-supervised monocular road detection in desert terrain. In *Proc. of Robotics: Science and Systems (RSS)*.

Duda, R. O. and Hart, P. E. (1972). Use of the Hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15.

Goldberg, S. B., Maimone, M., and Matthies, L. (2002). Stereo vision and robot navigation software for planetary exploration. In *IEEE Aerospace Conf.*

Grudic, G. and Mulligan, J. (2006). Outdoor path labeling using polynomial mahalanobis distance. In *Proc. of Robotics: Science and Systems (RSS)*.

Happold, M., Ollis, M., and Johnson, N. (2006). Enhancing supervised terrain classification with predictive unsupervised learning. In *Proc. of Robotics: Science and Systems (RSS)*.

Hinton, G. E., Osindero, S., and Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554.

Hong, T., Chang, T., Rasmussen, C., and Shneier, M. (2002). Road detection and tracking for autonomous mobile robots. In *Proc. of SPIE Aeroscience Conference*.

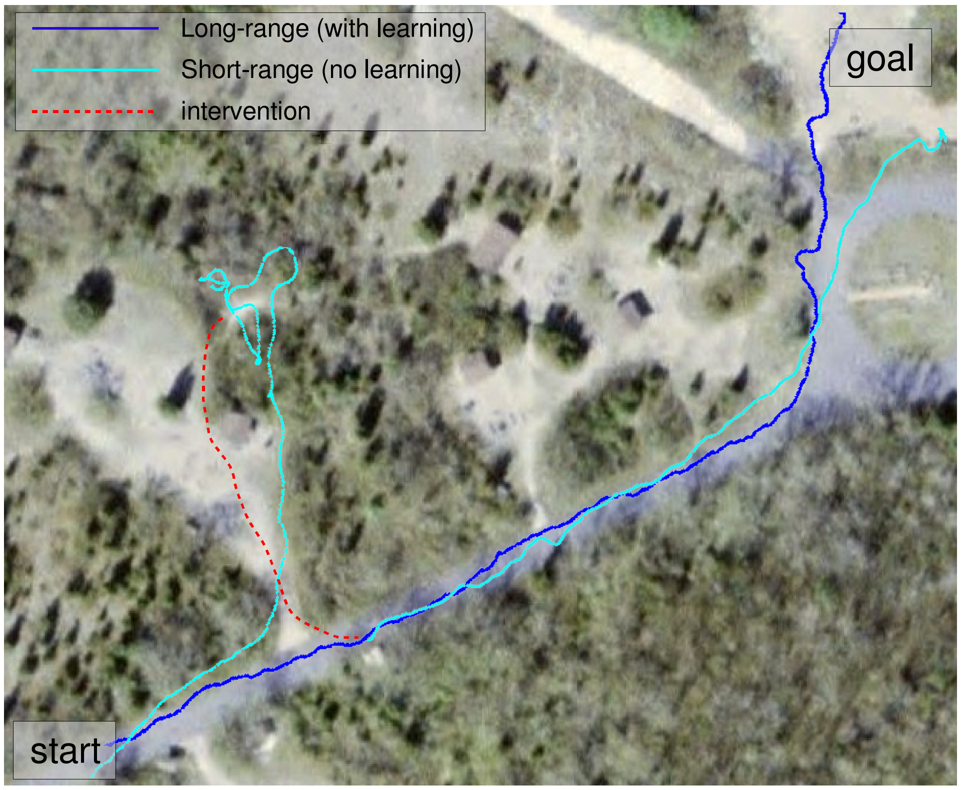


Figure 25: This course led down a wide paved road, with a natural cul-de-sac. The non-learning system took the turn into the cul-de-sac and had to be rescued. The long-range system could easily see the road and did not enter the cul-de-sac.

Course 1	Total Time	Total Distance	Inter-ventions
Short-Range	321 sec	271.9 m	3
Long-Range	155.5 sec	166.8 m	0
Course 2			
Short-Range	196.1 sec	207.5 m	1
Long-Range	142.2 sec	165.1 m	0
Course 3			
Short-Range	123.7 sec	122.9 m	0
Long-Range	108.7 sec	113.8 m	0
Average improvement of Long over Short	**157%**	**135%**	**1.33/0**

Table 2: Time-to-goal and Distance-to-Goal metrics for 3 offroad courses.

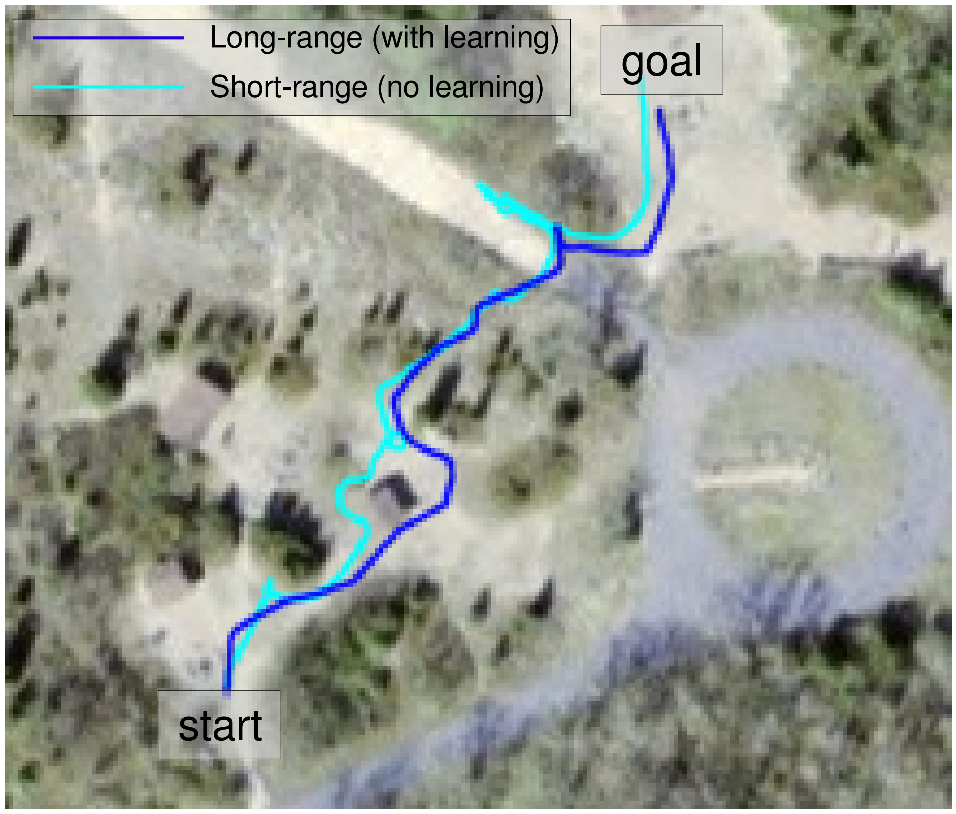


Figure 26: The third course zigzagged through three picnic areas. The paths taken by the different systems are similar, but the long-range vision path is smoother and optimized.

Inc., P. G. R. (2003). *Triclops Stereo Vision Software Development Kit User's guide and command reference*. Vancouver, B.C. Canada, version 3.1 edition.

Jackel, L. D. (2005). Learning applied to ground robots (LAGR).

Jackel, L. D., Krotkov, E., Perschbacher, M., Pippine, J., and Sullivan, C. (2006). The DARPA LAGR program: Goals, challenges, methodology, and phase I results. *Journal of Field Robotics*, 23(11-12):945–973.

Kelly, A. and Stentz, A. (1998). Stereo vision enhancements for low-cost outdoor autonomous vehicles. In *Int'l Conf. on Robotics and Automation, Workshop WS-7, Navigation of Outdoor Autonomous Vehicles*.

Kim, D., Sun, J., Oh, S. M., Rehg, J. M., and Bobick, A. F. (2006). Traversibility classification using unsupervised on-line visual learning for outdoor robot navigation. In *Proc. of Int'l Conf. on Robotics and Automation (ICRA)*. IEEE.

Konolige, K., Agrawal, M., Bolles, R. C., Cowan, C., Fischler, M., and Gerkey, B. (2008). *Outdoor Mapping and Navigation using Stereo Vision*, pages 179–190. Springer Tracts in Advanced Robotics. Springer Berlin / Heidelberg.

Kriegman, D. J., Triendl, E., and Binford, T. O. (1989). Stereo vision and navigation in buildings for mobile robots. *Trans. Robotics and Automation*, 5(6):792–803.

LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time-series. In *The Handbook of Brain Theory and Neural Networks*. MIT Press.

Leib, D., Lookingbill, A., and Thrun, S. (2005). Adaptive road following using self-supervised learning and reverse optical flow. In *Proc. of Robotics: Science and Systems (RSS)*.

Ranzato, M., Huang, F. J., Boreau, Y., and LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. of Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Sermanet, P., Hadsell, R., Scoffier, M., Grimes, M., Ben, J., Erkan, A., an U. Muller, C. C., and LeCun, Y. (2008a). A multi-range architecture for collision-free off-road robot navigation. Wiley Periodicals.

Sermanet, P., Hadsell, R., Scoffier, M., Muller, U., and LeCun, Y. (2008b). Mapping and planning under uncertainty in mobile robots with long-range perception. In *Proc. of Int'l Conf on Intelligent Robots and Systems (IROS)*. IEEE.

Sofman, B., Lin, E., Bagnell, J., Vandapel, N., and Stentz, A. (2006). Improving robot navigation through self-supervised online learning. In *Proc. of Robotics: Science and Systems (RSS)*.

Stavens, D. and Thrun, S. (2006). A self-supervised terrain roughness estimator for off-road autonomous driving. In *Proc. of Conf. on Uncertainty in AI (UAI)*.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., and Stang, P. (2006). Stanley: The robot that won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9):661–692.

Wellington, C. and Stentz, A. (2004). Online adaptive rough-terrain navigation in vegetation. In *Proc. of Int'l Conf. on Robotics and Automation (ICRA)*. IEEE.