High Speed Obstacle Avoidance using Monocular Vision and Reinforcement Learning

Jeff Michels JMICHELS@STANFORD.EDU
Ashutosh Saxena ASAXENA@STANFORD.EDU
Andrew Y. Ng ANG@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford, CA 94305 USA

Abstract

We consider the task of driving a remote control car at high speeds through unstructured outdoor environments. We present an approach in which supervised learning is first used to estimate depths from single monocular images. The learning algorithm can be trained either on real camera images labeled with ground-truth distances to the closest obstacles, or on a training set consisting of synthetic graphics images. The resulting algorithm is able to learn monocular vision cues that accurately estimate the relative depths of obstacles in a scene. Reinforcement learning/policy search is then applied within a simulator that renders synthetic scenes. This learns a control policy that selects a steering direction as a function of the vision system's output. We present results evaluating the predictive ability of the algorithm both on held out test data, and in actual autonomous driving experiments.

1. Introduction

In this paper, we consider the problem of high speed navigation and obstacle avoidance on a remote control car in *unstructured* outdoor environments. We present a novel approach to this problem that combines reinforcement learning (RL), computer graphics, and computer vision.

Most work on obstacle detection using vision has focused on binocular vision/steoreopsis. In this paper, we present a monocular vision obstacle detection algorithm based on supervised learning. Our motivation for this is two-fold: First, we believe that single-image monocular cues have not been effectively used in most obstacle detection systems; thus we consider it an important open problem to develop methods for obstacle detection that exploit these monocular cues. Second, the dynamics and inertia of high speed driving (5m/s on a small remote control car) means that obstacles must be perceived at a distance if we are to avoid them, and the distance at which standard binocular vision algorithms can perceive them is fundamentally limited by the "baseline" distance between the two cameras and the noise in the observations (Davies, 1997), and thus difficult to apply to our problem.[1]

To our knowledge, there is little work on depth estimation from monocular vision in rich, unstructured environments. We propose an algorithm that learns relative depth using only monocular visual cues on single images of outdoor environments. We collected a dataset of several thousand images, each correlated with a laser range scan that gives the distance to the nearest obstacle in each direction. After training on this dataset (using the laser range scans as the ground-truth target labels), a supervised learning algorithm is then able to accurately estimate the distances to the nearest obstacles in the scene. This becomes our basic vision system, the output of which is fed into a higher level controller trained using reinforcement learning.

An addition motivation for our work stems from the observation that the majority of successful robotic applications of RL (including autonomous helicopter flight, some quadruped/biped walking, snake robot locomotion, etc.) have relied on model-based RL (Kearns & Singh, 1999), in which an accurate model or simulator of the MDP is first built.[2] In control tasks in which the perception problem is non-trivial—such as when the input sensor is a camera—the quality of the simulator we build for the sensor will

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

[1] While it is probably possible, given sufficient effort, to build a stereo system for this task, the one commercial system we evaluated (because of vibrations and motion blur) was unable to reliably detect obstacles even at 1m range.

[2] Some exceptions to this include (Kim & Uther, 2003; Kohl & Stone, 2004).

be limited by the quality of the computer graphics we can implement. We were interested in asking: Does model-based reinforcement learning still make sense in these settings?

Since many of the cues that are useful for estimating depth can be re-created in synthetic images, we implemented a graphical driving simulator. We ran experiments in which the real images and laser scan data was replaced with synthetic images. We also used the graphical simulator to train a reinforcement learning algorithm, and ran experiments in which we systematically varied the level of graphical realism. We show that, surprisingly, even using low- to medium-quality synthetic images, it is often possible to learn depth estimates that give reasonable results on real camera test images. We also show that, by combining information learned from both the synthetic and the real datasets, the resulting vision system performs better than one trained on either one of the two. Similarly, the reinforcement learning controller trained in the graphical simulator also performs well in real world autonomous driving. Videos showing the system's performance in real world driving experiments (using the vision system built with supervised learning, and the controller learned using reinforcement learning) are available at `http://ai.stanford.edu/∼asaxena/rccar/`

2. Related Work

Broadly, there are three categories of cues that can be used for depth perception from two-dimensional images: monocular cues, stereopsis, and motion parallax (Kudo et al., 1999). By far the most commonly studied for this problem is stereo vision (Scharstein & Szeliski, 2002). Depth-from-motion or optical flow is based on motion parallax (Barron et al., 1994). Both methods require finding correspondence between points in multiple images separated over space (stereo) or time (optical flow). Assuming that accurate correspondences can be established, both methods can generate very accurate depth estimates. However, the process of searching for image correspondences is computationally expensive and error prone, which can dramatically degrade the algorithm's overall performance.

A number of researchers have studied how humans use monocular cues for depth estimation (Loomis, 2001; Wu et al., 2004; Blthoff et al., 1998). Also, (Kardas, 2005) presents experiments charaterizing some of the the different cues' effects. Such studies done both on humans and on animals show that cues like texture, texture gradient, linear perspective, occlusion, haze, defocus, and known object size provide information to estimate depth.

Gini (Gini & Marchi, 2002) used single camera vision to drive a indoor robot, but relied heavily on the known color and texture of the ground, and hence does not generalize well and will fail in unstructured outdoor environments. In (Pomerleau, 1989), monocular vision and apprenticeship learning (also called imitation learning) was used to drive a car, but only on highly structured roads and highways with clear lane markings, in which the perception problem is much easier. (LeCun, 2003) also successfully applied imitation learning to driving in richer environments, but relied on stereo vision. Depth from defocus (Jahne & Geissler, 1994; Honig et al., 1996; Klarquist et al., 1995) is another method to obtain depth estimates, but requires high-quality images, objects with sharp boundaries, and known camera parameters (including camera aperture model and modulation transfer function). Nagai (Nagai et al., 2002) built an HMM model of known face and hand images to recover depth from single images. Shao (Shao et al., 1988) used shape from shading to reconstruct depth for objects having relatively uniform color and texture.

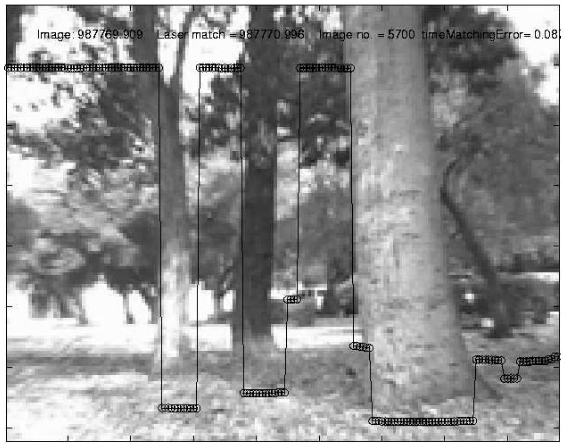
Image 987769.909 Laser match = 987770.996 Image no. = 5700 timeMatchingError= 0.08

Figure 1. Laser range scans overlaid on the image. Laser scans give one range estimate per degree.

3. Vision System

We formulate the vision problem as one of depth estimation over stripes of the image. The output of this system will be fed as input to a higher level control algorithm.

In detail, we divide each image into vertical stripes. These stripes can informally be thought of as corresponding to different steering directions. In order to learn to estimate distances to obstacles in outdoor scenes, we collected a dataset of several thousand outdoor images (Fig. 1). Each image is correlated with a laser range scan (using a SICK laser range finder, along with a mounted webcam of resolution 352x288, Fig. 2), that gives the distance to the nearest obsta-

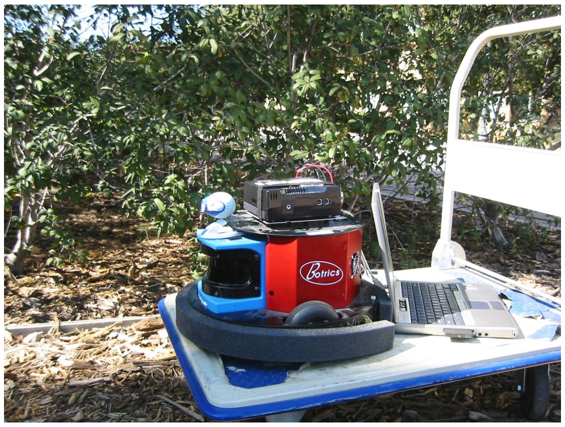

Figure 2. Rig for collecting correlated laser range scans and real camera images.

cle in each stripe of the image. We create a spatially arranged vector of local features capturing monocular depth cues. To capture more of the global context, the feature vector of each stripe is augmented with the features of the stripe to the left and right. We use linear regression on these features to learn to predict the relative distance to the nearest obstacle in each of the vertical stripes.

Since many of the cues that we used to estimate depth can be re-created in synthetic images, we also developed a custom driving simulator with a variable level of graphical realism. By repeating the above learning method on a second data set consisting of synthetic images, we learn depth estimates from synthetic images alone. Additionally, we combine information learned from the synthetic dataset with that learned from real images, to produce a vision system that performs better than either does individually.

3.1. Synthetic Graphics Data

We created graphics datasets of synthetic images of typical scenes that we expect to see while driving the car. Graphics data with various levels of detail were created, ranging from simple two-color trees of uniform height without texture; through complex scenes with five different kind of trees of varying heights, along with texture, haze and shadows (Fig. 3). The scenes were created by placing trees randomly throughout the environment (by sampling tree locations from a uniform distribution). The width and height of each tree was again chosen uniformly at random between a minimum and maximum value.

There are two reasons why it is useful to supplement real images with synthetic ones. First, a very large amount of synthetica data can be inexpensively created, spanning a variety of environments and scenarios. Comparably large and diverse amounts of real

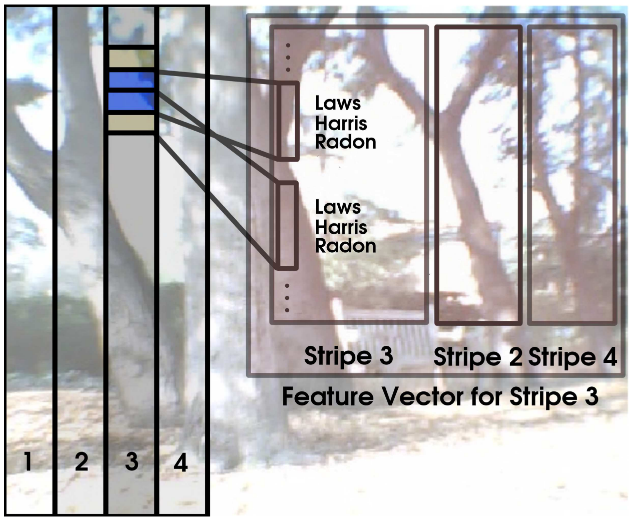

Figure 4. For each overlapping window W_{sr}, statistical coefficients (Law's texture energy, Harris angle distribution, Radon) are calculated. The feature vector for a stripe consists of the coefficients for itself, its left column and right column.

world data would have been much harder to collect. Second, in the synthetic data, there is no noise in the labels of the ground truth distances to obstacles.

3.2. Feature Vector

Each image is divided into 16 stripes, each one labeled with the distance to the nearest obstacle. In order to emphasize multiplicative rather than additive errors, we converted each distance to a log scale. Early experiments training with linear distances gave poor results (details omitted due to space constraints).

Each stripe is divided into 11 vertically overlapping windows (Fig. 4). We denote each window as W_{sr} for $s = 1...16$ stripes and $r = 1...11$ windows per stripe. For each window, coefficients representing texture energies and texture gradients are calculated as described below. Ultimately, the feature vector for a stripe consists of coefficients for that stripe, the stripe to its left, and the stripe to its right. Thus, the spatial arrangement in the feature vector for the windows allows some measure of global structure to be encoded in it.

3.2.1. Texture Energies

First, the image is transformed from RGB into the YCbCr color space, where Y represents the intensity channel, and Cb and Cr are the color channels. Information about textures is contained mostly in the variation of intensity (Davies, 1997). For each window, we apply Laws' masks (Davies, 1997) to measure texture energies. The nine Laws masks $M_1, \ldots, M_9$ are obtained by multiplying together pairs of three 1x3

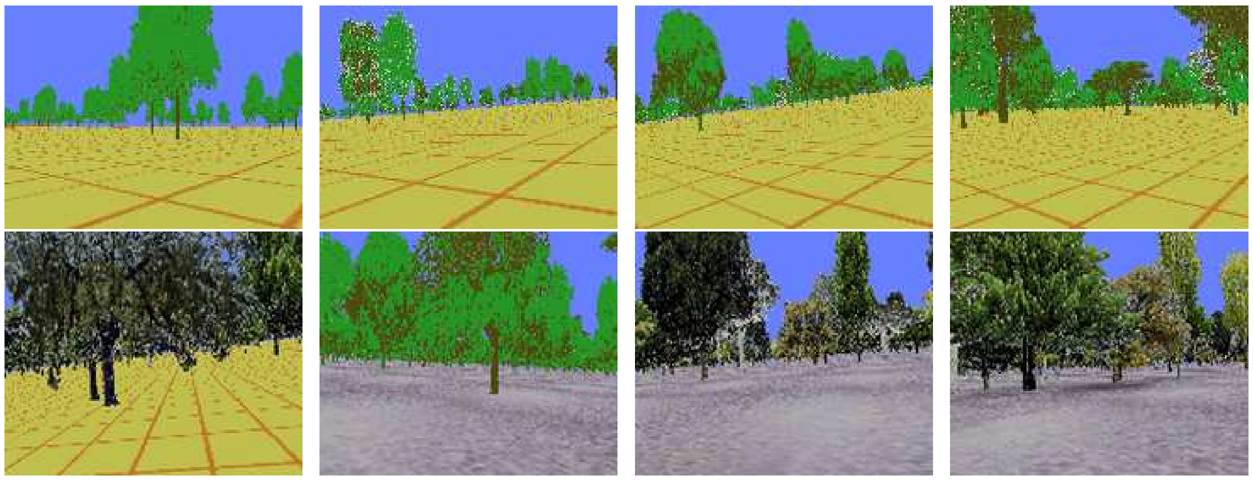

Figure 3. Graphics images in order of increasing level of detail. (In color, where available.) (a) Uniform trees, (b) Different types of trees of uniform size, (c) Trees of varying size and type, (d) Increased density of trees, (e) Texture on trees only, (f) Texture on ground only, (g) Texture on both, (h) Texture and Shadows.

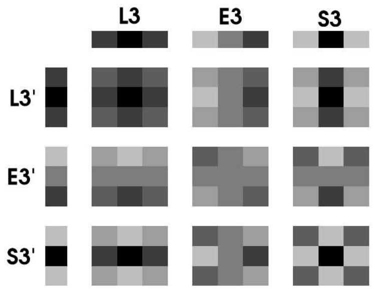

Figure 5. Law's masks for Texture Energy. The 1x3 masks (local averaging L_3, edge detection E_3 and spot detection S_3) are used to obtain nine 3x3 masks M_n

masks (Fig. 5). We apply these masks to the intensity channel image I_Y and to the color channel images I_{Cb}, I_{Cr} to obtain

$$F_n = I_Y * M_n, \quad n = 1, 2, ..., 9 \quad (1)$$

$$F_{10} = I_{Cb} * M_1 \quad (2)$$

$$F_{11} = I_{Cr} * M_1 \quad (3)$$

For color channels Cb and Cr, we calculate the local averaging mask M_1 only; this gives a total of 11 texture energy coefficients for each window. Lastly, we estimate the texture energy E_n for the r^{th} window in the s^{th} stripe (denoted as W_{sr}) as

$$E_n(s,r) = \sum_{x,y \in W_{sr}} |F_n(x,y)| \quad (4)$$

3.2.2. Texture Gradient

Studies on monocular vision in humans strongly indicate that texture gradients are an important cue in depth estimation. (Wu et al., 2004) When well-defined edges exist (e.g., scenes having regular structure like buildings and roads, or indoor scenes), vanishing points can be calculated with a Hough transform to get a sense of distance. However, outdoor scenes are highly irregular.

In order to calculate a texture gradient that is robust to noise in the image and that can capture a greater variety of textures, we use a variant of the Radon transform[3] and a variant of the Harris corner detector.[4] These features measure how the directions of edges in each window are distributed.

3.3. Training

Using the labeled data and the feature vector for each column as described above, we trained linear models to estimate the log distance to the nearest obstacle in a stripe of the image. In the simplest case, standard

[3]The Radon transform is a standard method for estimating the density of edges at various orientations. (Davies, 1997) We found the distribution of edges for each window W_{sr} using a variant of the this method. The Radon transform maps an image $I(x,y)$ into a new (θ, ρ) coordinate system, where θ corresponds to possible edge orientations. Thus the image is now represented as $I_{\text{radon}}(\theta,\rho)$ instead of $I(x,y)$. For each of 15 discretized values of θ, we pick the highest two values of $I_{\text{radon}}(\theta,\rho)$ by varying ρ, i.e., $R(\theta) = \text{toptwo}_\rho(g(\theta,\rho))$

[4]A second approach to finding distributions of directional edges over windows in the image is to use a corner detector. (Davies, 1997) More formally, given an image patch p (in our case 5x5 pixels), the Harris corner detector first computes the 2x2 matrix M of intensity gradient covariances, and then compares the relative magnitude of the matrix's two eigenvalues. (Davies, 1997) We use a slightly different variant of this algorithm, in which rather than thresholding on the smaller eigenvalue, we first calculate the angle represented by each eigenvector and put the eigenvalues into bins for each of 15 angles. We then sum up all of the eigenvalues in each bin over a window and use the sums as the input features to the learning algorithm.

linear regression was used in order to find weights w for N total images and S stripes per image

$$w = \arg\min_{w} \sum_{i=1}^{N} \sum_{s=1}^{S} \left(w^T x_{is} - \ln(d_i(s))\right)^2 \qquad (5)$$

where, x_{is} is the feature vector for stripe s of the i^{th} image, and $d_i(s)$ is the distance to the nearest obstacle in that stripe.

Additionally, we experimented with outlier rejection methods such as robust regression (iteratively reweighted using the "fair response" function, Huber, 1981) and support vector regression (Criminisi et al., 2000). Our this task, these methods did not provide significant improvements over linear regression, and simple minimization of the sum of squared errors produced nearly identical results to the more complex methods. All of the results below are given for linear regression.

3.4. Error Metrics

Since the ultimate goal of these experiments is to be able to drive a remote control car autonomously, the estimation of distance to nearest obstacle is really only a proxy for true goal of choosing the best steering direction. Successful distance estimation allows a higher level controller to navigate in a dense forest of trees by simply steering in the direction that is the farthest away. The real error metric to optimize in this case should be the mean time to crash. However, since the vehicle will be driving in unstructured terrain, experiments in this domain are not easily repeatable.

In the i^{th} image, let α be a possible steering direction (with each direction corresponding to one of the vertical stripes in the image), let α_{chosen} be steering direction chosen by the vision system (chosen by picking the direction cooresponding to the farthest predicted distance), let $d_i(\alpha)$ be the actual distance to the obstacle in direction α, and let $\hat{d}_i(\alpha)$ be the distance predicted by the learning algorithm. We use the following error metrics:

Depth. The mean error in log-distance estimates of the stripes is defined as

$$E_{depth} = \frac{1}{N}\frac{1}{S} \sum_{i=1}^{N} \sum_{s=1}^{S} \left|\ln(d_i(s)) - \ln(\hat{d}_i(s))\right| \qquad (6)$$

Relative Depth To calculate the relative depth error, we remove the mean from the true and estimated log-distances for each image. This gives a free-scaling constraint to the depth estimates, reducing the penalty for errors in estimating the scale of the scene as a whole.

Choosen Distance Error. When several steering directions α have nearly identical $d_i(\alpha)$, it is it is unreasonable to expect the learning algorithm to reliably pick out the single best direction. Instead, we might wish only to ensure that $d_i(\alpha_{chosen})$ is nearly as good as the best possible direction. To measure the degree to which this holds, we define the error metric

$$E_\alpha = \frac{1}{N} \sum_{i=1}^{N} \left|\ln(\max_s(d_i(s)) - \ln(d_i(\alpha_{chosen}))\right| \qquad (7)$$

This gives us the difference between the true distance in the chosen direction and the true distance in the actual best direction.

Hazard Rate. Because the car is driving at a high speed (about 5m/s), it will crash into an obstacle when the vision system chooses a column containing obstacles less than about 5m away. Letting d_{Hazard}=5m denote the distance at which an obstacle becomes a hazard (and writing $1\{\cdot\}$ to denote the indicator function), we define the hazard-rate as

$$\text{HazardRate} = \frac{1}{N} \sum_{i=1}^{N} 1\{d_i(\alpha_{chosen}) < d_{Hazard}\}, \qquad (8)$$

3.5. Combined Vision System

We combined the system trained on synthetic data with the one trained on real images in order to reduce the hazard rate error. The system trained on synthetic images alone had a hazard rate of 11.0%, while the best performing real-image system had a hazard rate of 2.69%. Training on a combined dataset of real and synthetic images did not produce any improvement over the real-image system, even though the two separately trained algorithms make the same mistake only 1.01% of the time. Thus, we chose a simple voting heuristic to improve the accuracy of the system. Specifically, we ask each system output its top two steering directions, and a vote is taken among these four outputs to select the best steering direction (breaking ties by defaulting to the algorithm trained on real images). This results in a reduction of the hazard rate to 2.04% (Fig. 7).[5]

4. Control

4.1. Reinforcement Learning

In order to convert the output of the vision system into actual steering commands for the remote control car, a control policy must be developed. The policy controls how aggressively to steer, what to do if the vision system predicts a very near distance for all of the possible steering directions in the camera-view, when to slow down, etc. We model the RC car control problem as a Markov decision process (MDP) (Sutton & Barto,

[5]The other error metrics are not directly applicable to this combined output.

θ_1: σ of the Gaussian used for spatial smoothing of the predicted distances
θ_2: if $\hat{d}_i(\alpha_{chosen}) < \theta_2$, take evasive action rather than steering towards α_{chosen}
θ_3: the maximum change in steering angle at any given time step
θ_4, θ_5: parameters used to choose which direction to turn if no location in the image is a good steering direction (using the current steering direction and the predicted distances of the left-most and right-most stripes of the image).
θ_6: the percent of max throttle to use during an evasive turn

Figure 6. Parameters learned by RL

1998). We then used the PEGASUS policy search algorithm (Ng & Jordan, 2000) (with locally greedy search) to learn the the parameters of our control policy. Due to space constraints, instead of a complete description of the policy class used, we give only a short description of each of the six learned parameters[6] is given in Fig. 6.

The reward function was given as

$$R(s) = -|v_{desired} - v_{actual}| - K \cdot \text{Crashed} \quad (9)$$

where $v_{desired}$ and v_{actual} are the desired and actual speeds of the car, Crashed is a binary variable stating whether or not the car has crashed in that time step. In our experiments, we used $K = 1000$. Thus, the vehicle attempts to maintain the desired forward speed while minimizing contact with obstacles.

Each trial began with the car initialized at (0,0) in a randomly generated environment and ran for a fixed time horizon. The learning algorithm converged after 1674 iterations of policy search.

4.2. Experimental Setup

Our test vehicle is based on an off-the-shelf remote control car (the Traxxas Stampede measuring 12”x16”x9” with wheel clearance of about 2”). We mounted on it a DragonFly spy camera (from PointGrey Research, 320x240 pixel resolution, 4mm focal length). The camera transmits images at up to 20 frames per second to a receiver attached to a laptop running a 1.6GHz Pentium. Steering and throttle commands are sent back to the RC transmitter from the laptop via a custom-built serial interface. Finally, the RC transmitter sends the commands directly to the car.

[6]Since the simulator did not accurately represent complex ground contact forces that act on the car when driving outdoors, two additional parameters were set manually: the maximum throttle, and a turning scale parameter

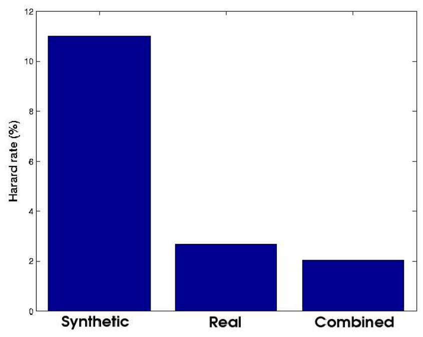

Figure 7. Reduction in hazard rate by combining systems trained on synthetic and real data.

5. Experimental Results

Table 2 shows the test-set error rates obtained using different feature vectors, when training on real camera images. As a baseline for comparing the performance of different features, the first row in the table shows the error obtained by using no feature at all (i.e., always predict the mean distance and steer randomly). The results indicate that texture energy and texture gradient have complementary information, with their combination giving better performance then either alone. Further, Harris and Radon features give comparable performance, with Harris performing slightly better. However, combining them does not significantly reduce the error. Table 3 shows the error rates for synthetic images at various degrees of detail. The performance of the vision system trained on synthetic images alone first goes up as we add texture in the images, and add variety in the images, by having different types and sizes of trees. However, it drops when we add shadows and haze. We attribute this to the fact that real shadows and haze are very different from the ones in synthetic images, and the learning algorithm significantly over-fits these images (showing virtually no error at all on synthetic test images).

Figure 7 shows hazard rates for the vision system relying on graphics-trained weights alone, real images trained weights alone, and improvement after combining the two systems.

We drove the car in four different locations and under different weather conditions. The learned controller parameters were directly ported from the simulator to the real world (rescaled to fit in the range of steering and throttle commands allowed by the real vehicle). For each location, the mean time[7] before crash is given in Table 1. All the terrain types had man-made structures (like cars and buildings) in the background.

[7]Some of the test locations were had nearby roads, and it was unsafe to allow the car to continue driving outside the bounds of our testing area. In these cases, the car was given a restart and the time was accumulated.

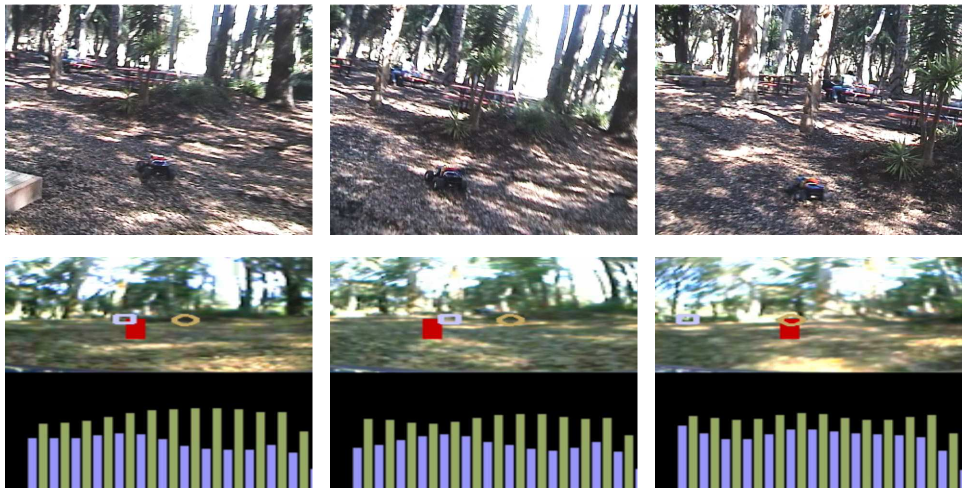

Figure 8. An image sequence showing the car avoiding a small bush, a difficult obstacle to detect in presence of trees which are larger visually yet more distant. (a) First row shows the car driving to avoid a series of obstacles (b) Second row shows the car view (c) Third row shows the predicted distances by the vision system. (Best viewed in color)

Table 1. Real driving tests under different terrains

Terrain	Obstacle Density	Obstacle Type	Ground	Mean Time (sec)	Max Time (sec)	Average Speed (m/s)
1	High	Sculptures, Trees, Bushes, Rocks	Uneven Ground Rocks, Leaves	19	24	4
2	Low	Trees, man-made benches	Even, with Leaves	40	80	6
3	Medium	Trees	Rough with Leaves	80	>200	2
4	Medium	Uniform Trees, man-made benches	tiled concrete	40	70	5

The unstructured testing sites were limited to areas where no training or development images were taken. Videos of the learned controller driving autonomously are available on the web at the URL given in Section 1. Performance while driving in the simulator was comparable to the real driving tests.

6. Conclusions and Discussion

We have shown that supervised learning can used to learn (1D) monocular depth estimates in unstructured outdoor environments, and demonstrated that the resulting depth estimates are sufficiently reliable to drive an RC car at high speeds through the environment. Further, the experiments with the graphical simulator show that model-based RL holds great promise even in settings involving complex environments and complex perception. In particular, a vision system trained on computer graphics was able to give reasonable depth estimate on real image data, and a control policy trained in a graphical simulator worked well on real autonomous driving. Some remaining important open problems are to characterize when graphics-trained algorithms will work in real life; and to develop further algorithms for exploiting monocular vision cues.

Acknowledgments

We give warm thanks to Vijay Pradeep for help setting up the RC Car, and Brian Gerkey for help collecting correlated laser and image data. We also thank Larry Jackel, Khian Lim, Wenmiao Lu, and Sebastian Thrun for helpful discussions. This work was supported by the DARPA LAGR program under contract number FA8650-04-C-7134.

References

Barron, J., Fleet, D., & Beauchemin, S. (1994). Performance of optical flow techniques. *Int'l Journal of Computer Vision*, *12*, 43–77.

Blthoff, I., Blthoff, H., & Sinha, P. (1998). Top-down influences on stereoscopic depth-perception. *Nature Neuroscience*, *1*, 254 – 257.

Criminisi, A., Reid, I., & Zisserman, A. (2000). Single

Table 2. Test error on 3124 images, obtained on different feature vectors after training data on 1561 images. For the case of "None" (without any features), the system predicts the same distance for all stripes, so relative depth error has no meaning.

Feature	E_{depth}	Rel Depth	E_θ	Hazard Rate
None	.900	-	1.36	23.8%
Y (intensity)	.748	.578	.792	9.58%
Laws only	.648	.527	.630	2.82%
Laws	.640	.520	.594	2.75%
Radon	.785	.617	.830	6.47%
Harris	.687	.553	.713	4.55%
Law+Harris	.612	.508	.566	2.69%
Laws+Radon	.626	.519	.581	2.88%
Harris+Radon	.672	.549	.649	3.20%
Law+Har+Rad	.604	.508	.546	2.69%

Table 3. Test error on 1561 real images, after training on 667 graphics images, with different levels of graphics realism using Laws features.

Train	E_{depth}	Rel Depth	E_θ	Hazard Rate
None	.900	-	1.36	23.8%
Laser Best	.604	.508	.546	2.69%
Graphics-8	.925	.702	1.23	15.6%
Graphics-7	.998	.736	1.10	12.8%
Graphics-6	.944	.714	1.04	14.7%
Graphics-5	.880	.673	.984	11.0%
Graphics-4	1.85	1.33	1.63	34.4%
Graphics-3	.900	.694	1.78	38.2%
Graphics-2	.927	.731	1.72	36.4%
Graphics-1	1.27	1.00	1.56	30.3%
G-5 (L+Harris)	.929	.713	1.11	14.5%

view metrology. *Int'l Journal of Computer Vision*, *40*, 123–148.

Davies, E. (1997). *Machine vision: Theory, algorithms, practicalities 2nd ed.* Academic Press.

Gini, G., & Marchi, A. (2002). Indoor robot navigation with single camera vision. *Proc. Pattern Recognition in Information Systems, PRIS, Spain.*

Honig, J., Heit, B., & Bremont, J. (1996). Visual depth perception based on optical blur. *Proc. of Int'l Conf. on Image Processing* (pp. 721–724).

Huber, P. (1981). *Robust statistics.* New York: Wiley.

Jahne, B., & Geissler, P. (1994). Depth from focus with one image. *Proc. IEEE Conf. on Computer Vision and Pattern Recognition CVPR* (pp. 713–717).

Kardas, E. (2005). Monocular cues in depth perception. *[Online]: http://peace.saumag.edu/faculty/Kardas/ Courses/GPWeiten/C4SandP/MonoCues.html.*

Kearns, M., & Singh, S. (1999). Finite-sample rates of convergence for q-learning and indirect methods. *NIPS 11* (pp. 996–1002). The MIT Press.

Kim, M., & Uther, W. (2003). Automatic gait optimisation for quadruped robots. *Proc. Australasian Conf. on Robotics and Automation* (pp. 1–9).

Klarquist, W., Geisler, W., & Bovik, A. (1995). Maximum-likelihood depth-from-defocus for active vision. *Proc. Int'l Conf. on Intelligent Robots and Systems* (pp. 374–379).

Kohl, N., & Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. *Proc. IEEE Int'l Conf. Robotics and Automation.*

Kudo, H., Saito, M., Yamamura, T., & Ohnishi, N. (1999). Measurement of the ability in monocular depth perception during gazing at near visual target-effect of the ocular parallax cue. *Proc. IEEE Int'l Conf. Systems, Man & Cybernetics* (pp. 34–37).

LeCun, Y. (2003). Presentation at Navigation, Locomotion and Articulation workshop. Washington DC.

Loomis, J. M. (2001). Looking down is looking up. *Nature News and Views*, *414*, 155–156.

Nagai, T., Naruse, T., Ikehara, M., & Kurematsu, A. (2002). Hmm-based surface reconstruction from single images. *Proc. IEEE Int'l Conf. on Image Processing* (pp. II–561 – II–564).

Ng, A. Y., & Jordan, M. (2000). Pegasus: A policy search method for large mdps and pomdps. *Proc. 16th Conf. UAI.*

Pomerleau, D. (1989). An autonomous land vehicle in a neural network. *NIPS 1.* Morgan Kaufmann.

Scharstein, D., & Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int'l Journal of Computer Vision*, *47*, 7–42.

Shao, M., Simchony, T., & Chellappa, R. (1988). New algorithms from reconstruction of a 3-d depth map from one or more images. *Proc. IEEE Conf. on Computer Vision and Pattern Recognition CVPR* (pp. 530–535).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning.* MIT Press.

Wu, B., Ooi, T. L., & He, Z. J. (2004). Perceiving distance accurately by a directional process of integrating ground information. *Letters to Nature*, *428*, 73–77.