Learning Obstacle Avoidance Parameters from Operator Behavior

Bradley Hamner, Sanjiv Singh, and Sebastian Scherer
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
e-mail: bhamner@andrew.cmu.edu, ssingh@cmu.edu, basti@cs.cmu.edu

Received 30 March 2006; accepted 5 November 2006

This paper concerns an outdoor mobile robot that learns to avoid collisions by observing a human driver operate a vehicle equipped with sensors that continuously produce a map of the local environment. We have implemented steering control that models human behavior in trying to avoid obstacles while trying to follow a desired path. Here we present the formulation for this control system and its independent parameters and then show how these parameters can be automatically estimated by observing a human driver. We also present results from operation on an autonomous robot as well as in simulation, and compare the results from our method to another commonly used learning method. We find that the proposed method generalizes well and is capable of learning from a small number of samples. © 2007 Wiley Periodicals, Inc.

1. INTRODUCTION

We are interested in high speed operation of an outdoor mobile robot whose task is to follow a path nominally clear of obstacles, but not guaranteed to be so. Such a case is necessary for outdoor patrolling applications where a mobile robot must travel over potentially great distances without relying on structure such as beacons and lane markings. In addition to avoiding obstacles, it is important that the vehicle stays on the designated route as much as possible. While the problem of detecting obstacles is itself challenging, here we consider issues related to collision avoidance while following a designated path given that the robot can detect obstacles in front of the vehicle in sufficient time to react to them.

Steering between obstacles is a difficult task, because good paths defy description by simple geometric constructs. Carlike vehicles, with a non-holonomic constraint, are limited in their capability to change direction, especially at high speeds where vehicle dynamics are a factor. Many methods of vehicle control have been reported in the literature, most of which rely on proper tuning of a set of parameters to a control function. These parameters dictate the robot's behavior, such as when to start avoiding obstacles, how much clearance to give obstacles, and the relative level of desire to progress toward a goal while avoiding obstacles. The values of these parameters and their relation to each other are critical to the effectiveness of the algorithm. Insufficient repulsion from obstacles can lead to collisions. Too much obstacle repulsion can cause erratic behavior as the vehicle veers away from obstacles. These gains have typically been tuned by hand over a large number of trials. The programmer decides when the vehicle's behavior is good enough.

We have implemented a model of collision avoidance based on studies with human subjects avoiding obstacles (Fajen, Warren, Termizer & Kaebling, 2003; Fajen & Warren, 2003). The model proposed by Fajen and Warren is attractive because it embeds steering dynamics into the generation of commands and hence guarantees that the control produced can be executed by the robot. The use of an explicit steering model also provides a means to accurately predict the closed loop path of the system over a time horizon many multiples of the control step. Such prediction facilitates the use of an aggressive speed control because the vehicle can be close to obstacles and not have to slow down unless it approaches a configuration of obstacles, such as a cul de sac, that will confound the collision avoidance scheme. The downside of this control model has been that it requires a large number of parameters to be adjusted to obtain a balance between safe and aggressive maneuvering.

Here we present a method for automatically learning the parameters of the control model by closely approximating observed paths of a human driver. We show how we collected a training data set and applied our method to learn a set of parameters. We present results of our obstacle avoidance system using these parameters to drive a vehicle, and compare the performance of the system using the learned parameters to its performance with a set of hand-tuned parameters. In cases where the immediate environment is densely packed with obstacles, we find that our scheme is insufficient on its own. This is because the horizon required for cluttered environments is further than can be incorporated by the model. We show here that augmentation with a path planner, which suggests goals (not paths), can help in negotiating in cluttered environments. Finally, to emphasize how our learning method is effective with a small training data set, we compare it against a black box road-following method that is trained on similar data as the proposed method.

2. RELATED WORK

Collision avoidance for mobile robots is a fundamental technology, and a large number of methods have been developed for various applications. Most of the work has been indoors with vehicles that move at speeds where dynamics are not an important consideration. For example, Borenstein and Koren's vector field histogram (VFH) method transforms a local map into a one-dimensional discretized "polar obstacle density" function (Ulrich & Borenstein, 2000). The angle closest to the heading to the goal that has low obstacle density is chosen. Minguez and Montano present the nearness diagram method, in which the robot classifies situations based on safety and nearness of obstacles, choosing an action appropriately (Minguez & Montano, 2004).

Other notable, similar methods are the dynamic window approach (Fox, Burgard & Thrun, 1997) and the curvature-lane approach (Simmons, 1996), which perform a parameter search in space of steering and velocity commands. These ideas have been extended in outdoor systems in which the robot projects candidate paths ahead of itself and then chooses among the corresponding steering actions the one that makes the most progress toward the goal and is obstacle free (Feiten, Bauer & Lawitzky, 1994). If no collision-free steering angle can move the robot toward a goal location, a higher level planner is consulted. Further work in the same vein uses a global planner in conjunction with the local planner and has been implemented on robots for space exploration (Singh et al., 2000; Urmson & Dias, 2002). Both of these systems operate at low speeds (less than 1 m/s) where vehicle dynamics are not a factor. Brock and Khatib proposed the elastic strips framework, in which an existing path is deformed around obstacles to produce smooth paths (Brock & Khatib, 2002). Philippsen and Siegwart use a modification of the traditional wavefront algorithm by parametrizing a curve that sweeps outward from a goal point but deforms around obstacles (Philippsen & Siegwart, 2005). The paths produced by this algorithm are smooth, but not guaranteed to respect the dynamic constraints of a vehicle.

Tilove proposed a local obstacle avoidance method that does respect dynamic constraints by using artificial potential fields (Tilove, 1990). In this method, obstacles influence a vehicle's motion based on its velocity as well as its position. Lamiraux, Bonnafous, and Lefebvre presented a method in which a nominal path is deformed around obstacles in accordance with vehicle dynamics to minimize an optimization function (Lamiraux, Bonnafous & Lefebvre, 2004). This method was shown to be successful in planning paths for an indoor robot.

Several systems have demonstrated off-road navigation. The Demo III XUV drives off-road and reaches speeds up to 10 m/s. The speeds are high, but the testing environments are rolling meadows with few obstacles. Obstacles are given a clearance that is wider than the clearance afforded by extreme routes. When clearance is not available, the algorithm plans at slower speeds (Bellutta, Manduchi, Matthies, Owens & Rankin, 2000). Sandstorm, a robot developed for desert racing, has driven extreme routes at speeds up to 22 m/s, but makes an assumption that it is traveling on slowly varying roads. If an obstacle is encountered in the center of a road, the path cannot change rapidly enough to prevent collision (Urmson et al., 2004).

Fajen and Warren have proposed a model for obstacle avoidance by humans (Fajen et al., 2003; Fajen & Warren, 2003). While similar to the standard potential field approach, there are important differences. Rather than create a potential field over the state of the world, their model stipulates a potential over the rate of heading change of the vehicle. Hence obstacles repel as a function of their bearing to the vehicle's heading. Second, the control is stated in a way that allows for the incorporation of vehicle dynamics.

Recently Huang et al. have used a modified version of the model proposed by Fajen and Warren that is geared toward obstacle avoidance using a monocular camera (Huang, Fajen, Fink & Warren, 2006). Since range to obstacles cannot be measured directly, the width of obstacles (segmented in the image) is used instead of the distance. The authors report results with an indoor robot moving at 0.7 m/s. In contrast we would like our robot to drive outdoors at high speeds where it might encounter various configurations of obstacles, and, since we would like the robot to track a specific path, the goal will move continuously.

There has been quite a lot of attention to learning as applied to mobile robots. Typically "learning" applies to perception as in learning a map of the environment (Thrun, 1998) or learning a classification of the terrain given image or range data (Wellington & Stentz, 2003; Talukder, Manduchi, Castano, Owens & Matthies, 2002). In a few cases learning has been applied to the control of a vehicle. For example, several systems at Carnegie Mellon University were trained to follow roads based on simultaneous observation of human driving and imagery of the road ahead from onboard cameras (Pomerleau, 1995; Hancock & Thorpe, 1994).

Likewise, Ng et al. developed a system to teach a helicopter to hover and perform stunt maneuvers via reinforcement learning (Ng, Kim, Jordan & Sastry, 2004). D'Este, O'Sullivan, and Hannah present a series of experiments in which a human operator is observed driving a mobile robot with a joystick (D'Este, O'Sullivan & Hannah, 2003). Their system then built a decision tree to model the behavior. An autonomous controller used the decision tree to choose actions in new scenarios. All of these methods seek black box relationships between a description of the environment and a control output. Such methods require a large amount of training data, and future test cases must lie within previous examples shown. That is, such methods are suited to interpolation between examples, but not to extrapolation.

Our work is most related to the model proposed by Fajen and Warren. We have extended their formulation to deal with path tracking and arbitrarily shaped obstacles. While Fajen and Warren chose coefficients for their model by inspection, we propose here a method to learn these coefficients automatically from observation of a human driver operating a vehicle among obstacles. The method does this by inverting the model using data taken over short segments of driving in the vicinity of obstacles.

3. APPROACH

Here we present the basic model of collision avoidance based on the formulation of Fajen and Warren (Fajen & Warren, 2003). We also show our extensions to the model and the parameters to be learned by the system.

3.1. Control Model

Fajen and Warren's model uses a single goal point that attracts the vehicle's heading. This attraction increases as the distance to the goal decreases and as the angle to the goal increases, yielding the goal attraction function

$$attract_{FW}(g) = k_g(\phi - \psi_g)(e^{-c_1 d_g} + c_2).$$

$(\phi - \psi_g)$ is the angle to the goal. d_g is the distance to the goal. k_g, c_1, and c_2 are parameters that must be tuned.

Similarly, each obstacle repulses the agent's heading. The repulsion increases with decreasing angle and decreasing distance. Then, for each obstacle, there is a repulsion function:

$$repulse_{FW}(o) = k_o(\phi - \psi_o)(e^{c_3 d_o})(e^{-c_4|\phi-\psi_o|})$$

$(\phi-\psi_o)$ is the angle to the obstacle. d_o is the distance to the obstacle. k_o, c_3, and c_4 are parameters that must be tuned.

The goal attractions and obstacles repulsions are summed together, applying superposition, and damped with the current angular velocity to get an angular acceleration command. The result is a single control law:

$$\ddot{\phi}^*_{FW} = -b\dot{\phi} - attract_{FW}(g) + \sum_{o \in O} repulse_{FW}(o).$$

Note that superposition will not always yield good results. There are many situations in which attractions and repulsions can oppose each other in such a way as to cause the vehicle to not properly avoid obstacles. This is a limitation of the system. It is up to the user to find good parameter values to prevent these situations from occurring as much as possible.

In our situation, a desired path is given. We set the goal point to be a large distance along the path from the vehicle's current location. Since the goal is always nearly the same distance from the vehicle, we do not need a distance term in our control law (called MFW for modified Fajen/Warren):

$$attract_{MFW}(g) = k_g(\phi - \psi_g).$$

Also, we found that FW works well for individual point obstacles, but in dense obstacle configurations more information is needed. In preliminary tests, we frequently observed the vehicle oversteering on curves due to repulsion from obstacles that were directly in front of the vehicle but far from its intended path. In these situations, the attraction of the goal is enough to assure the vehicle will avoid the obstacles. Using FW, though, the extra repulsion could even cause the vehicle to steer toward a small obstacle on its intended path. The obstacle on the path should have more weight than those off the path. Rather than decrease the weight of any obstacles, we add a term to the repulsion equation to put additional weight on obstacles between the vehicle and the goal:

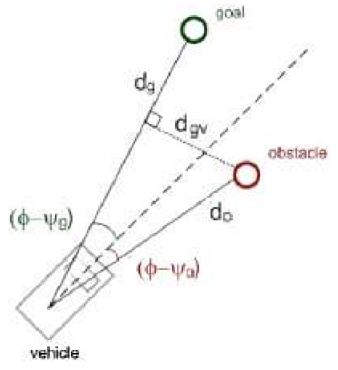

Figure 1. Distance and angle terms used in the MFW control law. We consider the vehicle's position to be the center of the rear axle, so all distances are measured from that point.

$$repulse_{MFW}(o) = k_o(\phi - \psi_o)(e^{-c_3 d_o})(e^{-c_4|\phi-\psi_o|}) \times (1 + c_5(d_{max} - \min(d_{max}, d_{gv})^2)).$$

For the last term, we draw a vector from the vehicle to the goal point. An obstacle's repulsion is increased in proportion to its distance, d_{gv}, to that vector. If the distance is greater than d_{max}, then no extra weight is applied. This maximum distance is based on the maximum distance the vehicle is allowed to be off the path. We use the goal vector term as an approximation of the obstacle's distance to the desired path, as it is simpler and quicker to calculate than the distance to the path. The approximation loses effectiveness when the path curves very sharply. However, in practical situations, we have found these cases to be rare.

Finally, Fajen and Warren's experiments were on humans, so they added the damping term and used a second-order command model to prevent sudden steering changes. Our vehicle has its own second-order system built in to the steering. Furthermore, the low level control of the vehicle is abstracted from our system; we can only send angular velocity commands to the vehicle. This being the case, we remove the damping term and define the control law to command angular velocity instead. This improves the

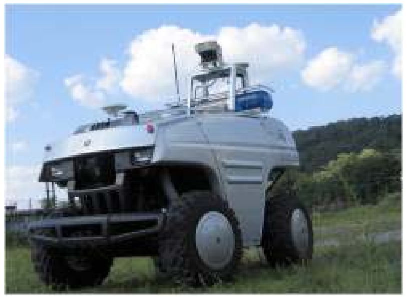

Figure 2. Our test platform is a modified all-terrain vehicle. The panning laser is mounted between the headlights. The fixed laser is mounted on top of the frame in the rear. The vehicle also carries two GPS antennas plus a differential antenna and an inertial measurement unit.

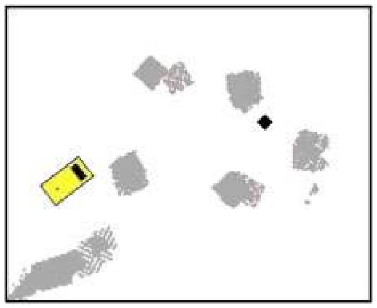

Figure 3. A capture of the screen displayed to the driver during training. The small, black diamond on the right represents a moving goal point that slides along the desired path. The vehicle is drawn large to make its heading clear to the driver, but each obstacle is conservatively expanded in each direction by the length of the vehicle to ensure collision-free operation. A collision only occurs when the representative point of the vehicle intersects the "grown" obstacles.

reaction time of the vehicle, but does allow oscillation to occur more readily:

$$\dot{\phi}^{*}_{MFW} = -k_g(\phi - \psi_g) + \sum_{o \in O} (k_o(\phi - \psi_o)(e^{-c_3 d_o})(e^{-c_4|\phi-\psi_o|}) \times (1 + c_5(d_{max} - \min(d_{max}, d_{gv})^2)).$$

We are left with five constant terms in the control law, which can be expressed in a 5-tuple $\bar{u} = (k_g, k_o, c_3, c_4, c_5)$. See Figure 1 for an illustration of the terms used in the MFW control law.

Fajen and Warren found that most subjects walked at a constant pace, so they did not explore speed control. We constructed a speed control function based on the obstacle's distance and angle:

$$v = \min_{o \in O}\left[\frac{d_o}{2\cos(|\phi - \psi_o|)}\right].$$

This slows down the vehicle as obstacles get closer, which allows sharper turning and more time for the

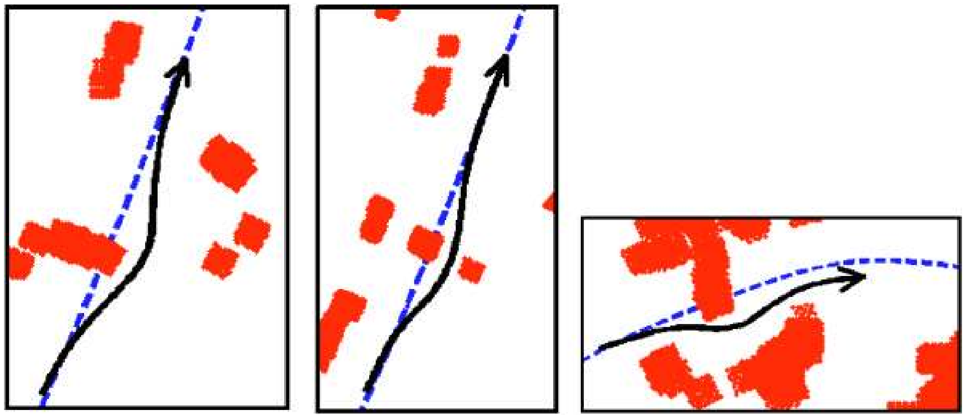

Figure 4. A few examples of the data segments collected from the experiments with the operator. The dashed line portrays the desired path if there were no obstacles. The operator deviates from the path to avoid obstacles and returns when the path is clear. The parameters presented in Section 5 were learned using only these three segments as training data.

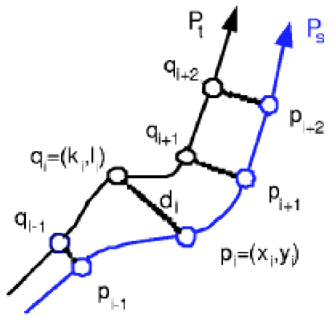

Figure 5. The difference between two path segments is used to optimize the parameter set u. P_s is a recorded path segment while P_t was generated from simulating the system's driving using a parameter set u.

system to detect additional nearby obstacles. In addition, the system predicts the vehicle's course using the MFW control law for the next few seconds. If the vehicle is forced to stop for an obstacle in the next few seconds, the system slows it down immediately. For quick computation, this is only a kinematic simulation. It assumes all steering commands are instantly achieved with the vehicle traveling in perfect arcs for half-second intervals. For more detail of our control system, please see Roth, Hamner, Singh & Hwangbo (2005).

3.2. Vehicle Characteristics

All data collection and testing were performed on a modified all-terrain vehicle (ATV), as shown in Figure 2. Obstacles were detected using data fused from two laser range finders, one fixed horizontally and the other arranged vertically panning back and forth to cover the road. Ground truth was provided by a pose estimation system, which uses the global position system (GPS) combined with motion data from an onboard inertial measurement unit (IMU) to produce positioning accurate to better than 5 cm.

To provide a better model of vehicle motion and increase the accuracy of vehicle simulations for the training and testing of parameters, we derived the vehicle's steering dynamics. The vehicle's steering actuator in contact with the ground behaves like a second-order system with delay. The identified parameters of the unit-mass spring-damper system are

$$\ddot{\theta} = -b\dot{\theta} - k(\theta - \theta_d),$$

$$b = 6.5789,$$

$$k = 258.7,$$

$$t_{delay} = \phi.2 \text{ s}.$$

These values were used to generate vehicle paths as described in the following section.

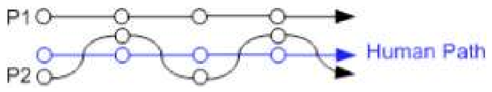

Figure 6. Path 2 stays closer to the human path than path 1, but is less desirable, due to its high frequency oscillations.

Table I. The parameter sets.

	k_g	k_o	c_3	c_4	c_5
Hand-tuned	0.767	0.060	0.340	2.000	0.250
Learned random	0.8976	7.5537	0.9082	9.0856	0.5688
Learned genetic	0.6445	8.2175	1.6119	13.018	6.0817
Learned SA	6.4328	8.7506	1.4971	5.0594	5.5838

4. SEARCH FOR PARAMETERS

Tuning the set of parameters $\bar{u}$ by hand using intuition is tedious and difficult. In this section we outline

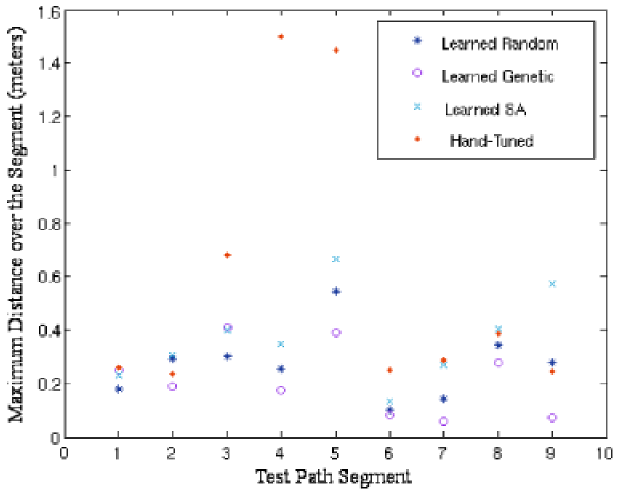

Figure 7. The maximum distance error of the hand-tuned and learned parameters for each test segment.

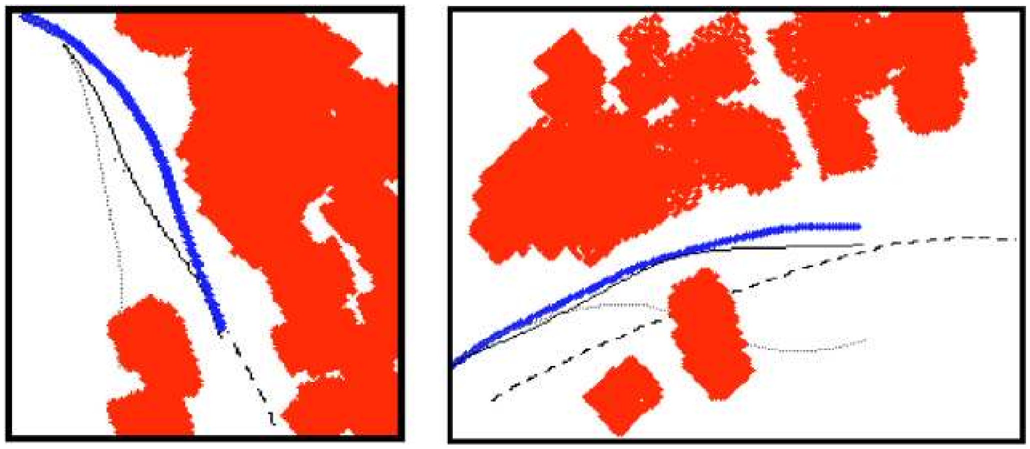

Figure 8. The system's performance on test paths 4 and 5 with the two sets of parameters. The hand-tuned parameters (dotted line) cannot deal with the large obstacles on both sides of the vehicle. The learned parameters (solid line) drive closer to the human's path (heavy line). The desired path is indicated by the dashed black line.

a method to automatically determine the parameters based on a driver's behavior steering the robot.

4.1. Data Collection

A human subject drives the robot and tries to follow a path while avoiding obstacles. The path is replaced by a virtual goal point at a distance ahead of the current position as mentioned in Section 3.1. Data about the goal point, the obstacles, and the driven path are recorded and used to determine the unknowns of the described control model. First we segment the path, and then optimize the parameters to match the subject's path.

The input to the control model and human subject at any point in time is a goal point p_g and a set of obstacles O defined as points in the x-y plane of the vehicle. A subject sat on the robot and drove while

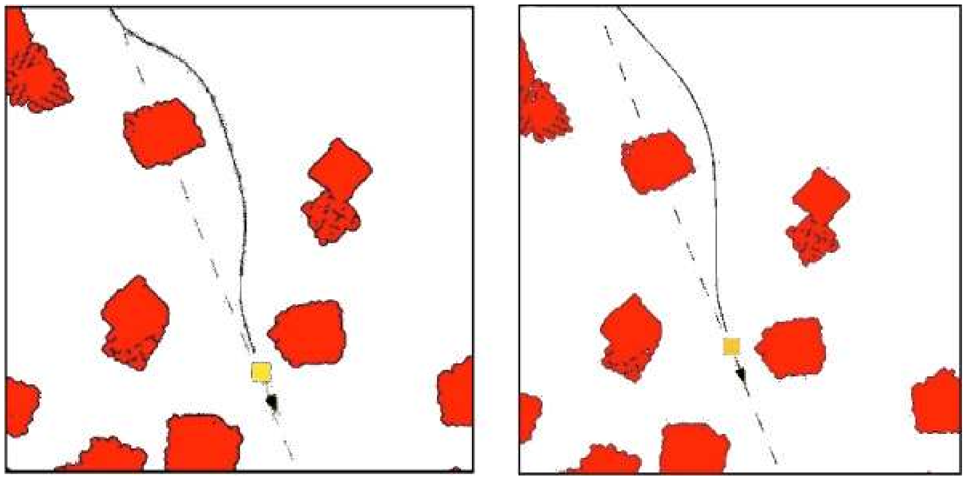

Figure 9. At left, the path driven by the system using the parameters learned from the genetic algorithm. At right, the path driven by the system using the parameters learned by random guessing. They both avoid obstacles, but the genetic algorithm parameters make harder turns.

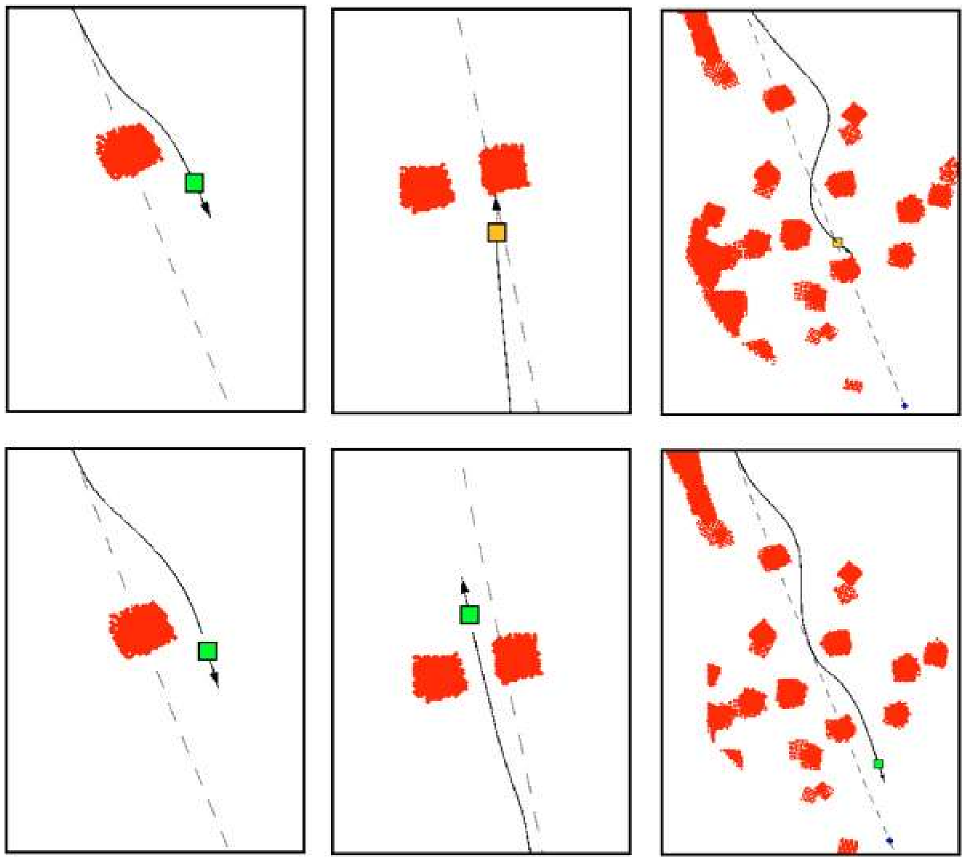

Figure 10. We also tested the system in new situations at 4 m/s in simulation with recorded obstacles. On the top, the hand-tuned parameter set can avoid individual obstacles, but performs poorly in more complex situations. The system must stop the vehicle before a collision occurs. On the bottom, the system avoids all obstacles when using the randomly learned parameters.

only looking at a monitor that displays the virtual goal point and the relevant obstacles, not looking up to see the true location of obstacles ahead. This setup ensures that the operator only knows of the obstacles the system has detected. It does hinder the operator's driving ability, as he could drive more easily using his own vision to detect obstacles. However, since we would be training the autonomous system to follow the same paths as the operator, it was imperative for our learning experiments that the operator and the autonomous system have the same information about the goal and obstacle points. It was also imperative that the operator be skilled in driving the vehicle, though. Before the experiments began, the driver was allowed to practice driving the vehicle while staring at the monitor until he felt comfortable operating the vehicle with this impediment. Figure 3 shows the only information available to the operator.

Using the goal and obstacle information, the operator drove the vehicle through a variety of paths and obstacle configurations. During operation, the pose estimation system collected information about the driven path at a rate of 100 Hz. A pose data point contains a timestamp, an X-Y position, heading, velocity, and angular velocity. We also recorded the goal points given to the driver, as well as the locations of obstacles and when they were detected.

Often the paths were obstacle-free, where the driver simply followed the goal point. Of course, these situations provide no information with which to learn the obstacle parameters. We therefore ex-

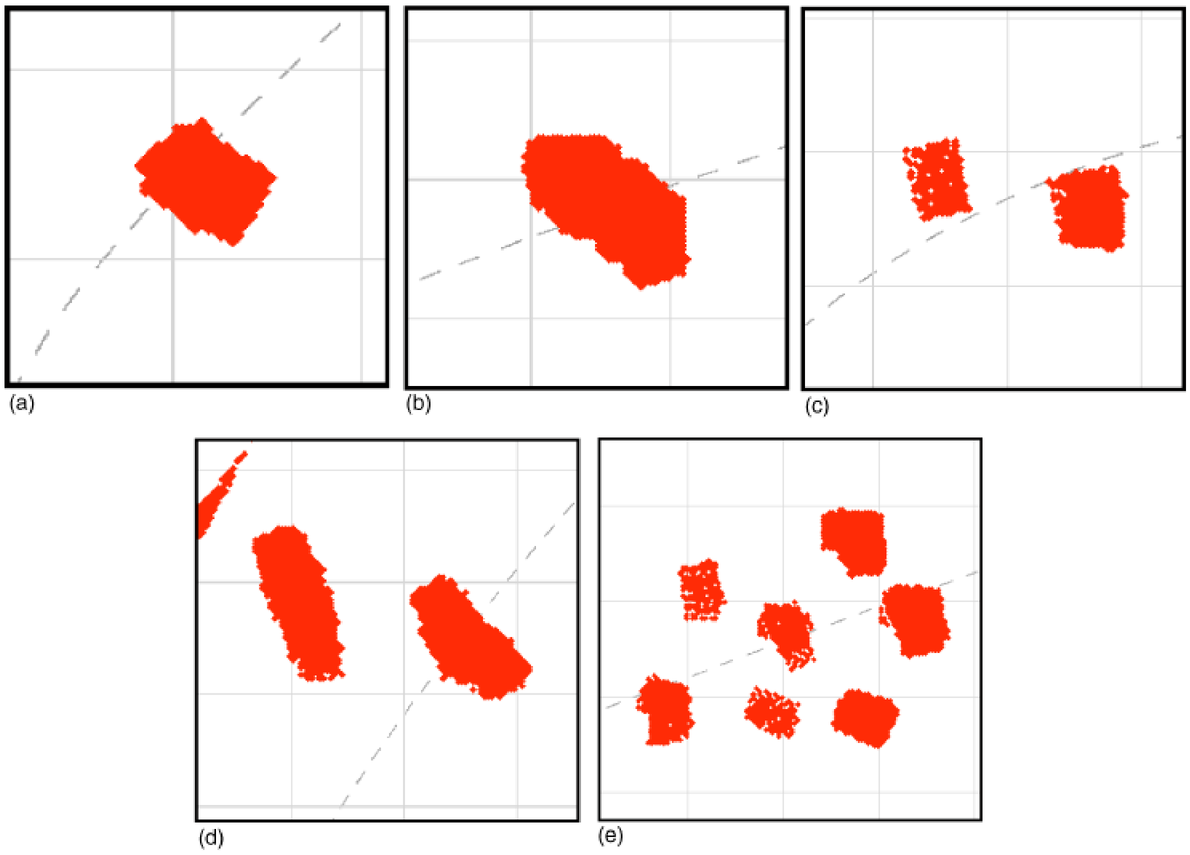

Figure 11. Example obstacle scenarios from each of the five categories: (a) single small obstacles less than 1 m wide, (b) single obstacles wider than 1 m, (c) two to three small obstacles, (d) two to three wide obstacles, and (e) clutter (more than four obstacles). The dashed line is the desired path. The cells in the underlay grid are 3×3 m^2.

tracted only the relevant cases, breaking the data into short path segments consisting of cases where the driver steered the vehicle around the obstacles and returned to the path. A few of these segments are shown in Figure 4.

4.2. Problem Setup

We use the path segments with obstacle avoidance behavior to train the parameters of our control model. Given a set $\bar{u}$ of parameters, we generate a path $P_t=\{q_i=(k_i,l_i)|i=1,\ldots,n\}$ with the same n number of points as the training path segment P_s, which contains regularly sampled points in the plane. $P_s=\{p_i=(x_i,y_i)|i=1,\ldots,n\}$.

One measure of error is the Euclidean distance between each point pair as shown in Figure 5: $d(\bar{u})_i=\sqrt{(k_i-x_i)^2-(l_i-y_i)^2}$. However, we are also concerned with driving smoothly. Minimizing distance error alone could allow high frequency oscillations, illustrated in Figure 6. Consequently, we add a term to penalize accelerations in the steering angle used by the autonomous system. The total error minimized between two path segments is $D(\bar{u})=\Sigma_{i=1}^{n}[d(\bar{u})_i+abs(\ddot{\rho}_i)]$, where ρ is the commanded steering angle. We minimize the error of the parameter set for a number of path segments. Over m segments, the optimization procedure minimizes the combined error term $\min_{\bar{u}}\delta(\bar{u})=\Sigma_{j=1}^{m}D(\bar{u})_j$.

The path P_t is generated from a forward simulation of the steering behavior of the robot. This simulation is intended to be very accurate, using the steering model and its derived parameters presented in the previous section. Since the length of the path

Table II. Results of on-vehicle tests.

	No. of scenarios	Collisions	Stuck states	Success rate (%)	Recoveries	Recovery rate (%)
Single small (Hand-tuned)	26	2	1	**88.4**	1	100.0
Single small (Learned random)	26	1	0	**96.1**	···	···
Single large (Hand-tuned)	17	0	6	**64.7**	6	100.0
Single large (Learned random)	17	0	2	**88.2**	2	100.0
Multiple small (Hand-tuned)	27	1	12	**51.8**	7	58.3
Multiple small (Learned random)	27	0	1	**96.3**	1	100.0
Multiple large (Hand-tuned)	18	1	13	**22.2**	2	14.3
Multiple large (Learned random)	18	0	1	**94.4**	1	100.0
Clutter (Hand-tuned)	15	0	6	**60.0**	2	33.0
Clutter (Learned random)	15	0	3	**80.0**	0	0.0
Overall (Hand-tuned)	103	4	32	**65.0**	18	56.2
Overall (Learned random)	103	1	7	**92.2**	4	57.1

and the velocities are not controlled in this model, we use the recorded speeds to ensure P_t has the same length as the training path P_s. The simulated path is generated as follows:

For a control iteration, the commanded angular velocity is

$$\dot{\phi}^*_{MFW} = -\,attract_{MFW}(g) + \sum_{o \in O} repulse_{MFW}(o).$$

The commanded steering angle given the translational velocity v and the vehicle length l is

$$\theta_d = a\tan\left(l\frac{\dot{\phi}^*_{MFW}}{v}\right).$$

The system retains a notion of the vehicle's current steering angle. The commanded steering angle is integrated with the current steering angle using the second-order model presented in Section 3.2:

$$\ddot{\theta} = -\,b\dot{\theta} - k(\theta - \theta_d).$$

The differential equations are integrated using the first-order Newton-Euler method to produce a new steering angle. The system then calculates the curvature of the arc the vehicle would travel on until the next control iteration:

$$\kappa = \frac{\tan(\theta)}{l}.$$

Finally, the vehicle's position is updated to be at the end of that arc:

$$(x_{next}, y_{next}, \theta_{next}) = updateArc(x_{prev}, y_{prev}, \theta_{prev}, \kappa, v).$$

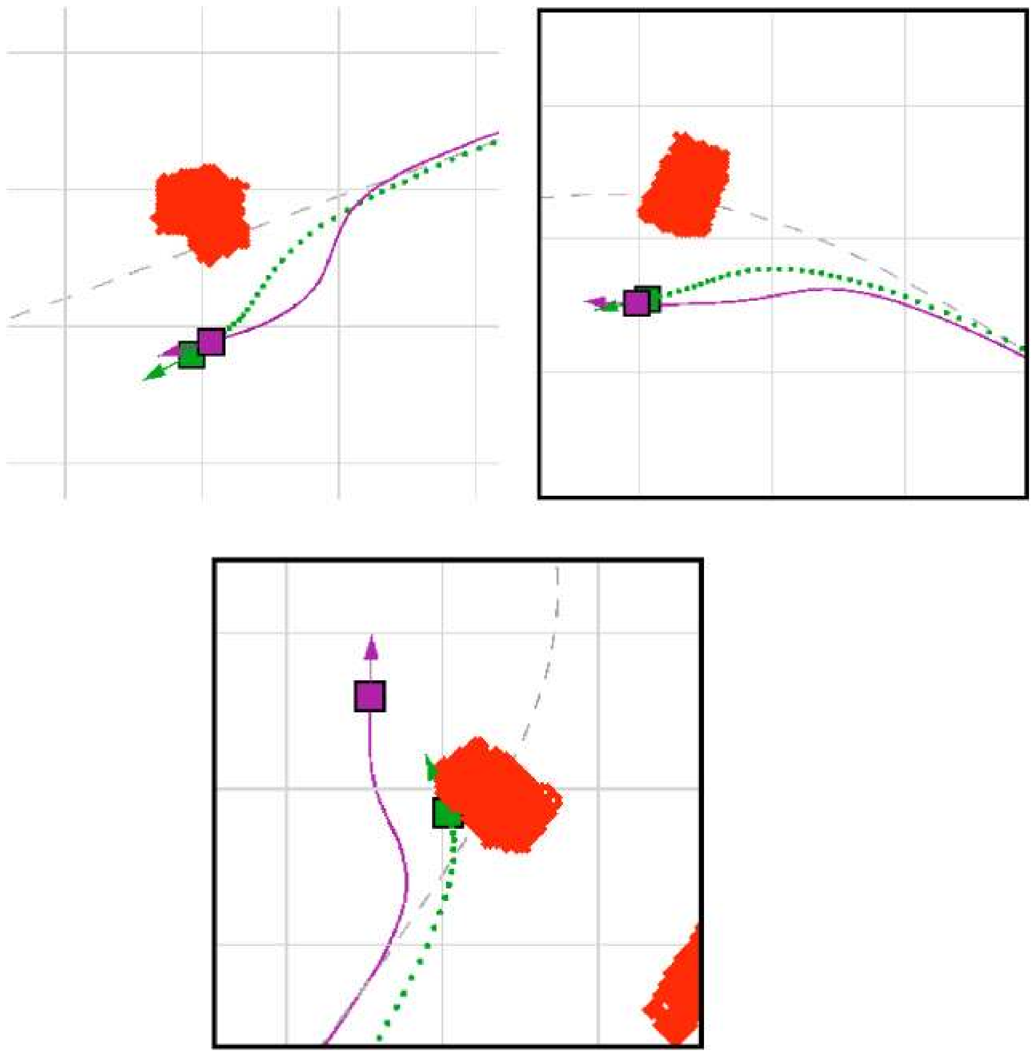

Figure 12. Sample scenarios from category A, single small obstacles, in the on-vehicle tests. In each figure, the dashed line is the desired path, the solid line is the path of the vehicle using the learned parameter set, and the dotted line is the path of the vehicle using the hand-tuned parameter set. The cells in the grid underlay are 3×3 m^2.

4.3. Learning Process

The relationship between the set of parameters $\bar{u}$ and the resulting paths is nonlinear with many local minima. Normal gradient descent techniques are insufficient to find the global optimum, since they get trapped in the local minima. We address this problem using a two-step optimization process; first a nondeterministic algorithm to identify good candidates, followed by an optimization step.

We tried three different approaches for the nondeterministic step. The first was simply to randomly choose sets of parameters. We set the range of each parameter to be uniformly distributed between 0 and 10. With these ranges the system randomly picked 2500 sets of parameters. We kept the ten sets with the lowest residuals.

Another approach was a genetic learning algorithm modeled on the theory of survival of the fittest. The system starts with a population of 25 randomly chosen sets of parameters, each parameter having a value between 0 and 10. Every iteration, 25 new sets are formed through combination and mu-

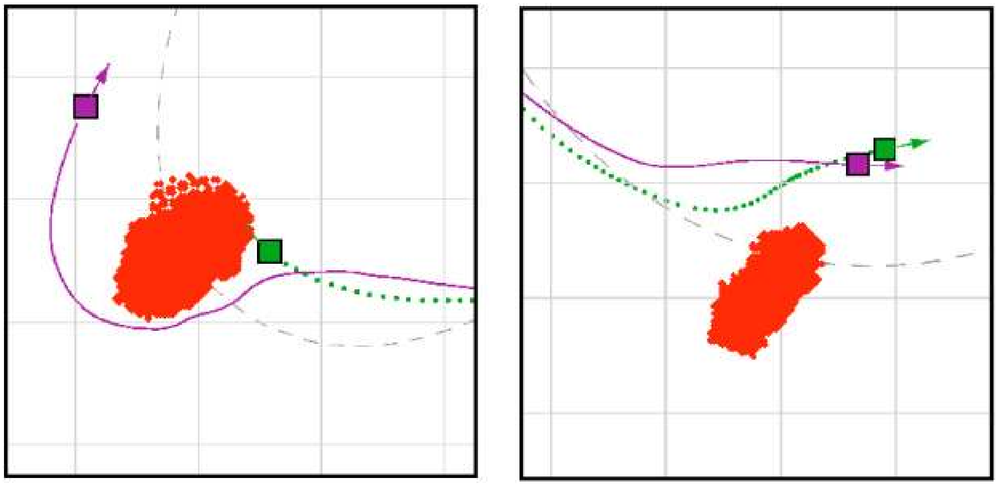

Figure 13. Sample scenarios from category B, single wide obstacles, in the on-vehicle tests. In each figure, the dashed line is the desired path, the solid line is the path of the vehicle using the learned parameter set, and the dotted line is the path of the vehicle using the hand-tuned parameter set. The cells in the grid underlay are 3×3 m^2.

tation. In combination, the system selects two parameter sets to form a new one, where each of the five new parameters is taken independently from one of the two parent sets. In mutation, the system selects one set and randomly mutates a number of its parameters to other values. The sets to be combined and mutated are drawn randomly with a distribution favoring those with the lowest residual, δ. The residuals of the population decrease over time as better sets are found. After 100 iterations of the genetic algorithm (totaling 2500 error computations for a population of 25) we kept the ten sets with the lowest residuals.

The last approach was simulated annealing. The system starts with one randomly chosen set of parameters. Each iteration, the system either performs gradient descent on the current parameters or randomly selects new parameters. The probability of performing gradient descent increases over time, approaching one. We ran 2500 iterations of simulated annealing. From the 2500 sets whose residuals were calculated, we kept the ten with the lowest residuals.

Following the nondeterministic step, we applied a nonlinear least squares procedure with the previous best parameter sets as the initial guesses, producing ten optimized parameter sets for each approach. Then for each approach the optimized parameter set with the lowest residual, δ, was chosen as the best parameter set.

5. RESULTS

The method described above learned optimal parameter sets with only three training path segments, covering 30 m of driving. We left the rest of our segments for test data. We now compare the performance of our control system using the learned parameters against hand-tuned parameters, and against a system learned using principal component analysis.

5.1. Comparison with Hand-Tuned Parameters

Previous to this work, the system had used a parameter set that we had hand-tuned over the course of a year. The set was tuned in a manner similar to gradient descent, starting with an initial guess and adjusting individual parameters slightly until we thought we had achieved the best vehicle behavior. The four sets are shown in Table I. Notice they are very dissimilar. We compare them on our remaining nine path segments, excluding the three training segments, in Figure 7. This figure displays the maximum distance away from the human-driven path on each data segment. The hand-tuned parameters perform nearly as well as the learned ones in most segments, but are drastically far away in a few situations. A few of the test paths are shown in Figure 8. The parameters learned with simulated annealing perform better than the hand-tuned parameters, but

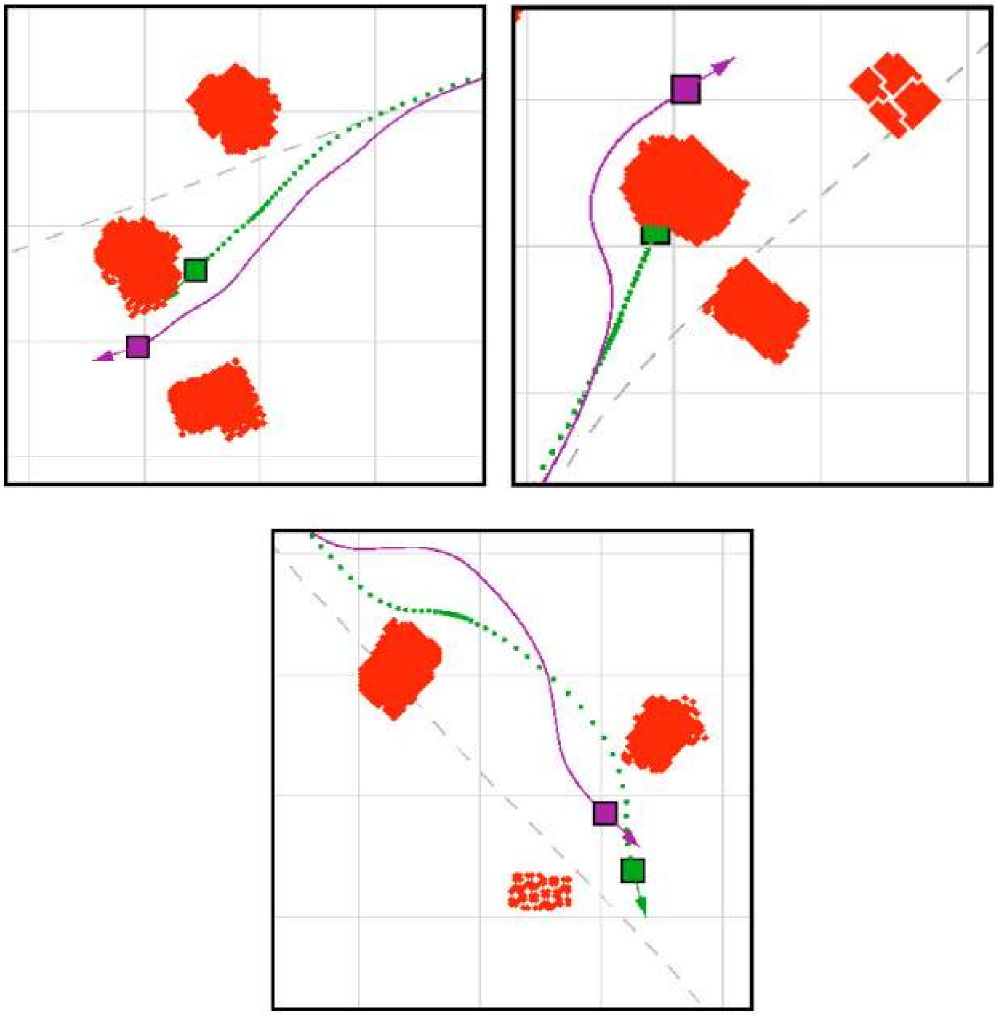

Figure 14. Sample scenarios from category C, multiple small obstacles, in the on-vehicle tests. In each figure, the dashed line is the desired path, the solid line is the path of the vehicle using the learned parameter set, and the dotted line is the path of the vehicle using the hand-tuned parameter set. The cells in the grid underlay are 3×3 m^2.

not as well as the other learned parameter sets. The parameters learned with the genetic algorithm perform only slightly better than the randomly learned parameters on the distance metric, averaging 6 cm closer to the human path, despite the difference in the learned parameters. The two have closer overall residuals due to more oscillation in the parameters from the genetic algorithm. The mean test residuals are 0.3 for the hand-tuned set, 0.28 for the simulated annealing set, 0.18 for the randomly learned set, and 0.14 for the genetically learned set. We chose the randomly learned parameter set to use for additional testing, since it drives the vehicle smoother than the genetically learned set, as shown in Figure 9. Neither method gives high frequency oscillations, but the set learned by the genetic algorithm makes harder turns. This suggests that the terms we included in the error metric to ensure smooth steering need to be improved. Either smooth steering needs to be weighted higher, or we need to design a new metric to prevent those hard turns. All further results were obtained with the system using the set learned through random guesses.

Of course, our recorded data segments are only

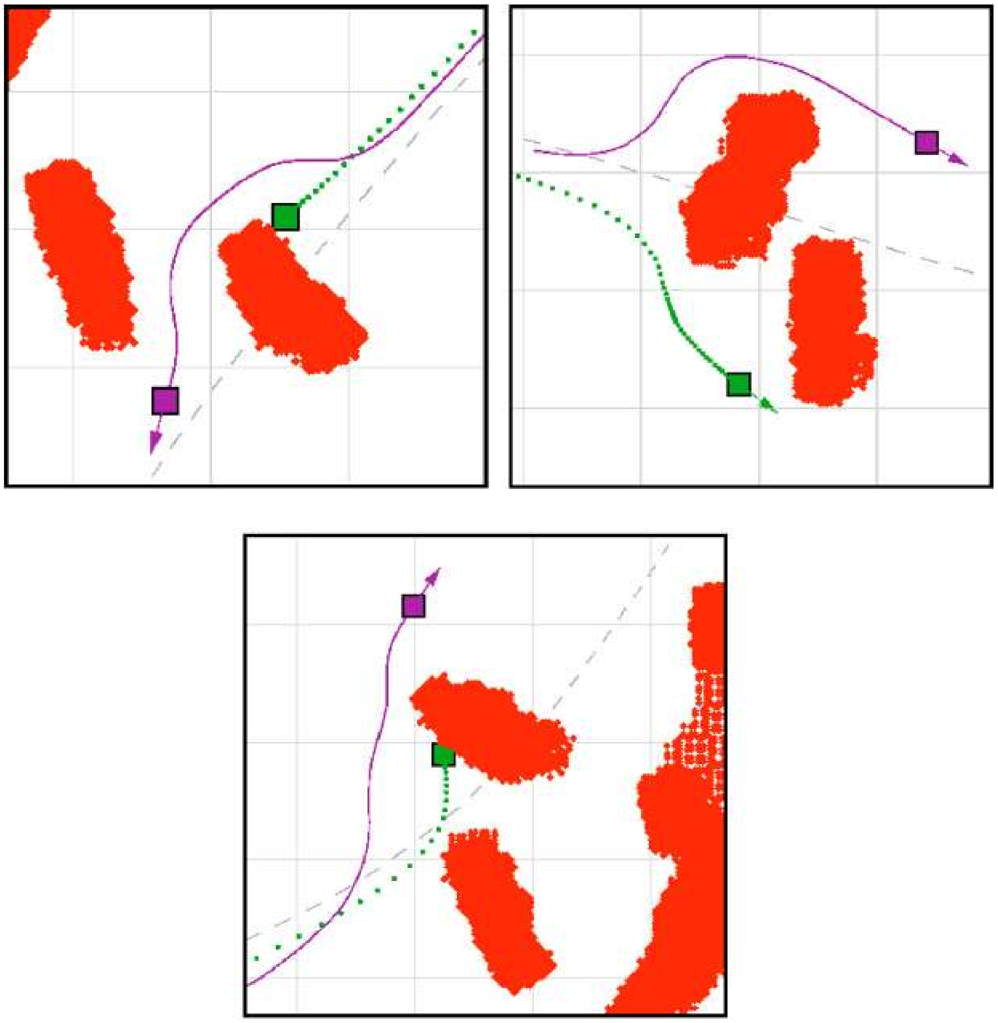

Figure 15. Sample scenarios from category D, multiple wide obstacles, in the on-vehicle tests. In each figure, the dashed line is the desired path, the solid line is the path of the vehicle using the learned parameter set, and the dotted line is the path of the vehicle using the hand-tuned parameter set. The cells in the grid underlay are 3×3 m^2.

a small sample of what the vehicle may encounter. To ensure that the learned parameter set is better than the hand-tuned set, we first ran the system with the two sets in a vehicle simulator at 4 m/s, using new recorded obstacles from data files. Obstacle detection is simulated by an obstacle replay program that reads the vehicle's location from shared memory and serves to the simulator all obstacles within a fixed distance from the vehicle. The system using the hand-tuned set frequently got the vehicle "stuck," where the vehicle has not avoided an obstacle and must instead stop before colliding with it. The system using the learned parameters got stuck much less often. A few of these cases are shown in Figure 10. In simulation, we did not encounter a case where the learned parameters get the vehicle stuck and the hand-tuned parameters do not.

With our method showing promise in simulation, we then performed experiments on a test vehicle. For these tests, we used a different test vehicle. This vehicle is of approximately the same scale, 2.5 m long and 1.5 m wide, and had the same sensor setup but different steering dynamics. We ran the vehicle using the new learned parameters and the

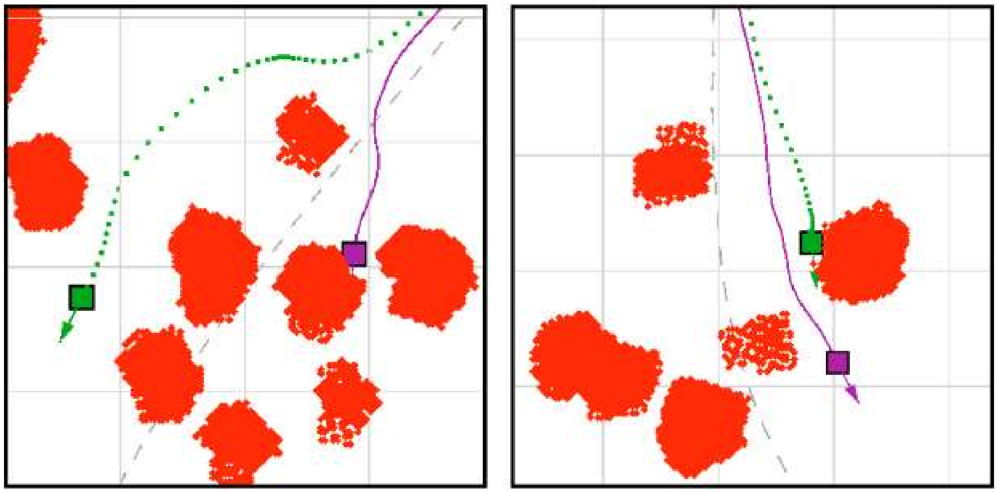

Figure 16. Sample scenarios from category E, clutter, in the on-vehicle tests. In each figure, the dashed line is the desired path, the solid line is the path of the vehicle using the learned parameter set, and the dotted line is the path of the vehicle using the hand-tuned parameter set. The cells in the grid underlay are 3×3 m^2.

hand-tuned parameters on the same test paths through over 100 obstacle avoidance scenarios. An obstacle avoidance scenario is defined as an instance in which an obstacle or a cluster of obstacles lies closer than 1 m to the desired path (that is, slightly larger than half the vehicle width), forcing the vehicle to divert from the path. A scenario is completed when the vehicle returns to the path clear of ob-

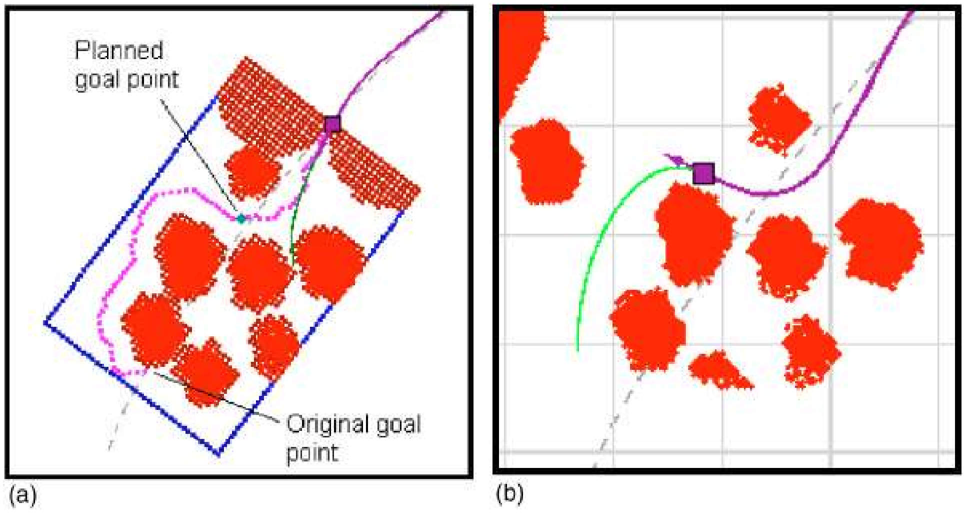

Figure 17. In (a), the predicted path using the learned parameters (solid line extending forward from the vehicle point) anticipates the vehicle coming to a stop in a cul de sac. The system builds a planning map (dotted rectangle) that surrounds the area. It creates a plan around the obstacles (dotted line extending forward from the vehicle) and selects a new goal point (dot on the plan). The system then runs the reactive obstacle avoidance system with the new goal. In (b), the prediction no longer shows a collision, so planning is no longer necessary.

stacles. To better analyze the data, we grouped the obstacle scenarios into five categories (A–E), as described in Figure 11.

In our experiments, we consider that the vehicle has successfully navigated a scenario when it leaves the path and returns to it without colliding with an obstacle or getting stuck (that is, coming to a complete stop before an obstacle to avoid a collision). In these experiments, when the vehicle did get stuck, we gave it a chance to recover by manually backing up the vehicle by 2 m and starting the system again from a stop. This tells us if the system could have successfully navigated the scenario with better speed modulation. Even if the vehicle did recover, though, as a data point for our experiments, this type of situation was declared a failure.

Since our obstacle detection system requires mechanical scanning, two identical runs in an identical environment can be different based on when obstacles are seen. To ensure that the parameters sets being compared (hand-tuned versus learned) have the same input, we prerecorded obstacles in the environment and made available a map of the local environment to the robot as it navigated. That is, any obstacle within 10 m becomes known to the obstacle avoidance system. This horizon is consistent with online sensing.

We conducted 103 tests total with the categories A–E with the vehicle being commanded a maximum speed of 4.0 m/s. Results of these tests are shown in Table II. The learned parameter set had a higher success rate than the hand-tuned parameter set in every obstacle category. Both parameter sets performed well in single, small obstacle scenarios (category A). The hand-tuned parameters got the vehicle stuck once, when the obstacle was located on a curve in the desired path. See Figure 12 for a few examples of the vehicle's paths in scenarios from category A.

In single, wide obstacle scenarios (category B), the learned parameters outperformed the hand-tuned. However, the learned parameters did have problems with wide obstacles along curves in the desired path, sometimes resulting in stuck states (see Figure 13). We believe we could improve the vehicle's behavior in these scenarios by including more of them in the training data. The hand-tuned set performed particularly poorly when avoiding multiple obstacles (categories C and D). Since the learned parameters were trained on multiple obstacles, both small and wide, their usage allowed the system to maintain a high success rate in those categories. See Figures 14 and 15 for examples of these scenarios.

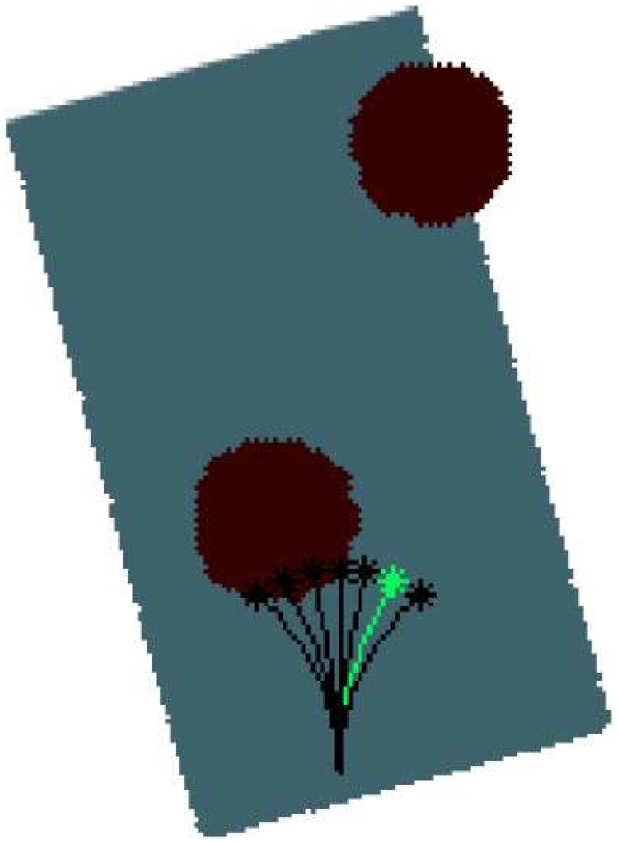

Figure 18. To overcome limited training data, we created random situations. Here, the user is shown a top down view of a set of obstacles (large dots) and a road (wide gray strip). He is also shown a set of potential steering arcs for the vehicle (black arcs). The user chooses which arc the vehicle should drive on (highlighted) to best avoid the obstacle while following the road.

Finally, neither parameter set performed very well in category E, cluttered obstacles. The learned parameter set got the vehicle stuck three times out of 15 tests. All of these were produced by the system attempting to drive the vehicle through narrow spaces between two obstacles, as shown in Figure 16. In these cases it would have been easier for the vehicle to drive around all the obstacles.

In categories A–D, the system was able to recover from stuck states 100% of the time using the learned parameters. This provides further support to the suggestion that the vehicle should slow down

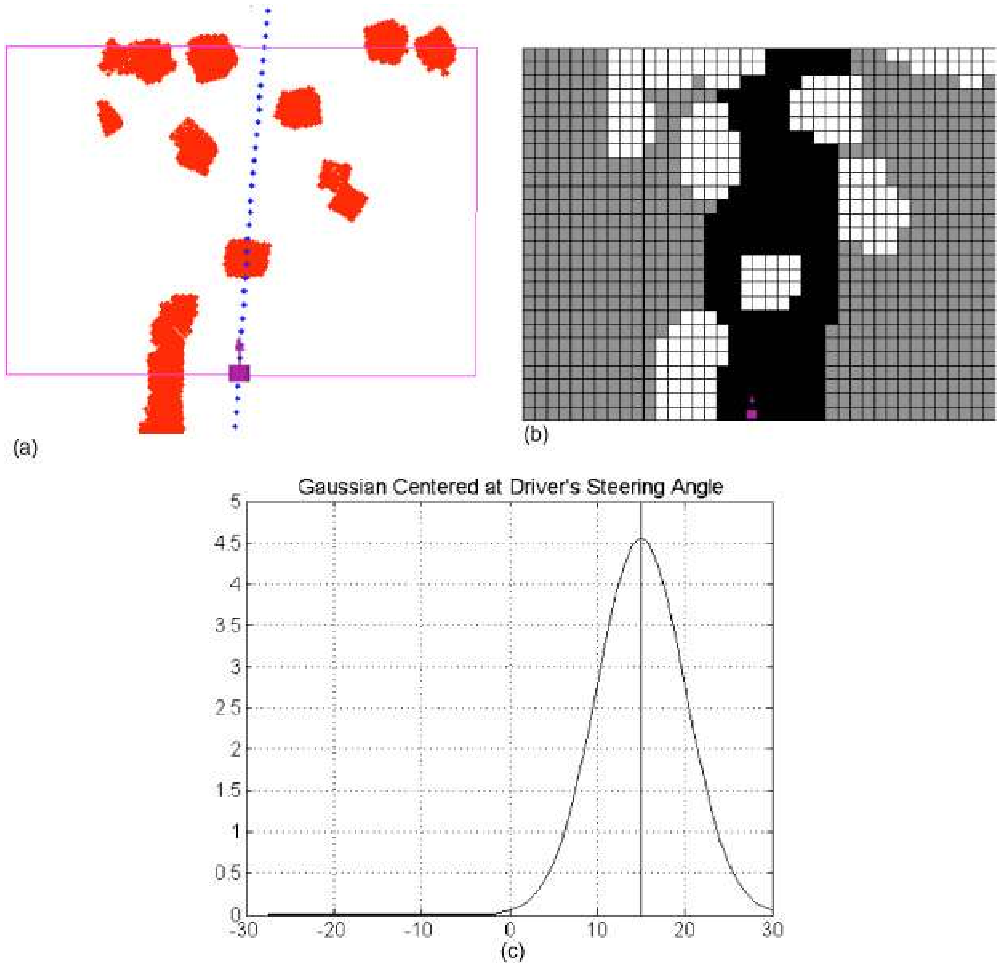

Figure 19. This is an illustration of the creation of a training vector. In (a), the vehicle is on the desired path (dotted line) in front of a few obstacles. In (b), the input image contains everything inside the rectangle from (a). The desired path was expanded to make a road, which are the black cells, value 1. The obstacles are the white cells, value −1. Everything else is gray, value 0. In this situation, the driver commanded a steering angle of 15 deg. In (c), a steering vote vector is created. The input image (b) and steering vote vector (c) combine to form a training vector.

more as it approaches obstacles. However, using the learned parameters, the slower speed did not help the system to recover in the cluttered scenarios (category E). This points to a limitation of the proposed method and reactive methods in general, in that they do not look ahead in time far enough to make decisions for the next step. Even if more cluttered scenarios are included in the training data, the system would likely still get stuck in the presence of high clutter.

Since it is important that our system be able to effectively deal with any amount of clutter that is visible to it, we have used an augmentation to the proposed method in earlier work (Roth et al., 2005). Recall from Section 3.1 that the system predicts the vehicle's path 4 s ahead. Using this prediction the system can check if the vehicle will enter a stuck state. If this is the case, the system performs an A^* search on a local map, selects a goal point along the planned path, and runs the reactive controller again with the new goal point. In essence, the planner is picking a general direction for the reactive controller. The reactive obstacle avoidance controller still runs to ensure feasibility of the plan, smoothing out sharp

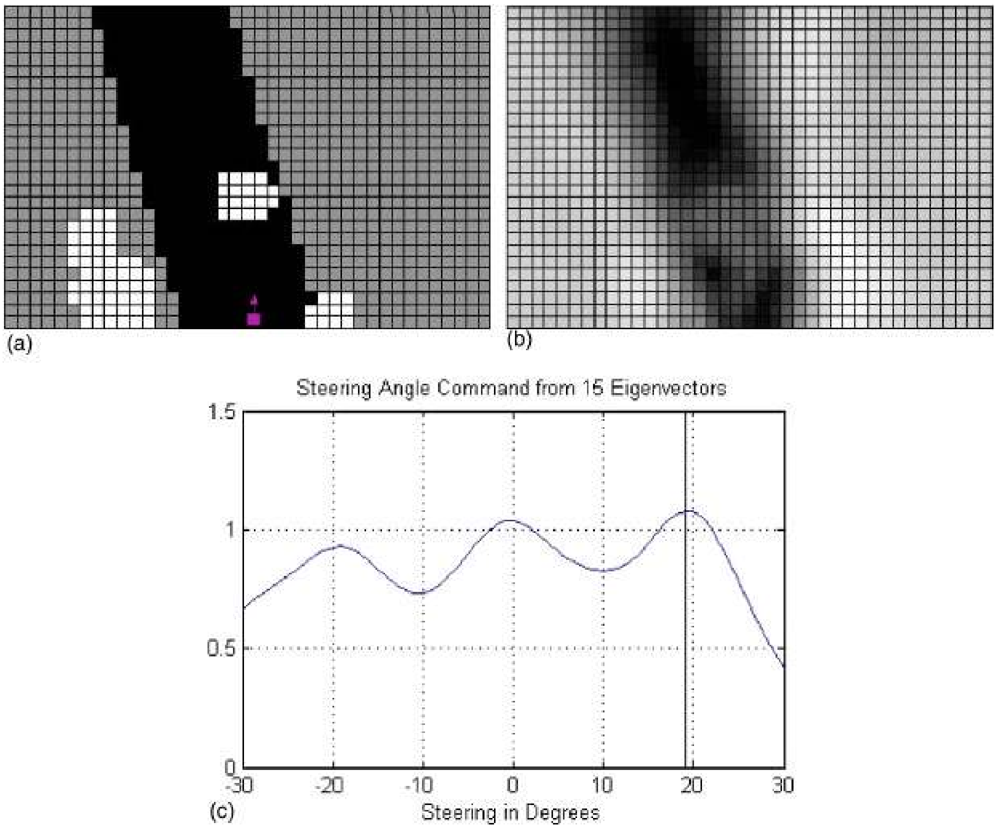

Figure 20. In (a), the input image shows a road veering to the vehicle's left, with an obstacle in front of the vehicle and slightly to its left. The black road pixels have value 1, the white obstacle pixels have value −1, and the gray pixels have value 0. In (b), the reconstruction of the input image after projection onto the 15 principal eigenvectors. The obstacle that was in the road now appears as a slightly lighter spot in the center of the road. In (c), there are three peaks in the steering angle voting: steer to the left around the obstacle, steer straight down the road, or steer right around the obstacle. The highest vote is for steering 19 deg to the right, so that is the command given to the vehicle.

turns and maintaining a good distance from obstacles. The plan and new goal point are recreated each iteration until the prediction shows no impending stuck states. Figure 17 shows how our planning augmentation leads the vehicle around the clutter case where it had previously got stuck using just the reactive controller with the learned parameters.

5.2. Comparison with Principal Component Analysis

Our proposed method uses a strong model of collision avoidance to learn parameters from observation of human driving. Here we compare performance of this method with a more black box approach in which the system directly learns a mapping between the environment and the steering function. For comparison we implemented eigenvector projection using principal component analysis, a method that has been used for road following using video imagery (Hancock & Thorpe, 1994). This method is similar to road following using a neural network (Pomerleau, 1995) but assumes a linear relationship between the input and the output.

Initially we used all of our collected path segments as the training set, plus a copy of the set mirrored from left to right to remove the possibility of directional steering bias. However, since a competent driver provided training data, there are no instances of the vehicle getting too close to the ob-

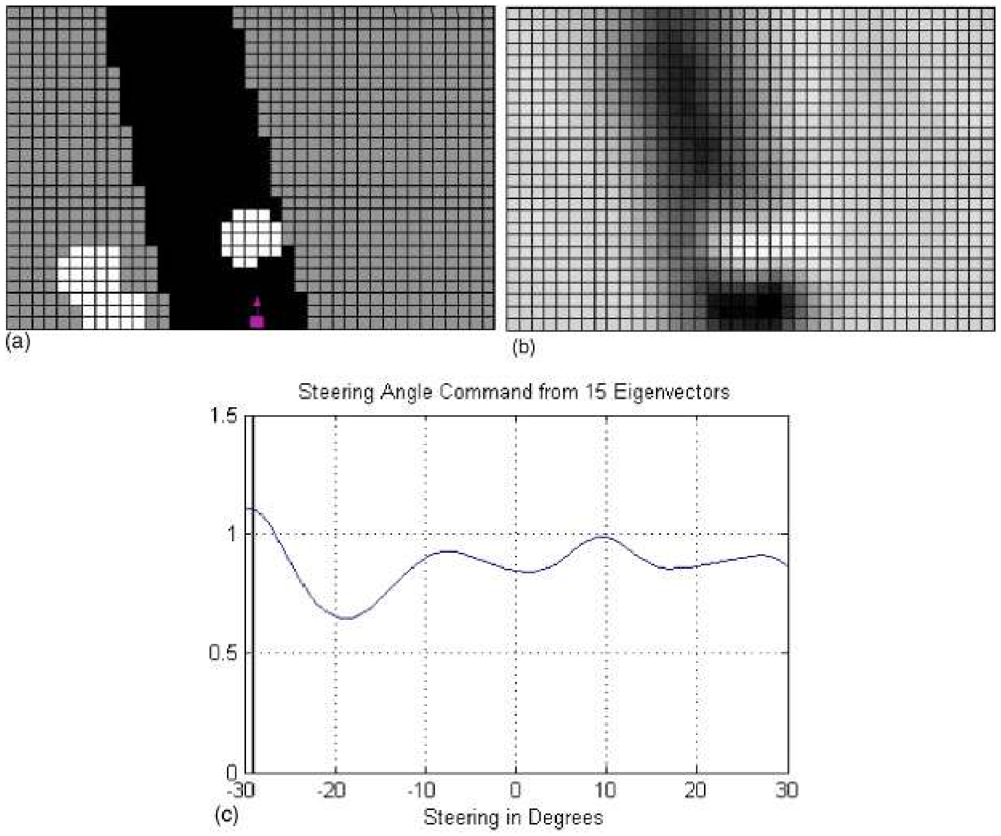

Figure 21. This figure presents the situation continued from Figure 20. In (a), in the input image, the obstacle is closer to the vehicle. In (b), the obstacle now shows up in the reconstructed image as a bright spot in front of the vehicle. In (c), the steering choices are now to steer hard to the left to avoid the obstacle, slightly to the left to follow the road, or hard to the right to avoid the obstacle. The highest vote is to steer 28 deg to the left.

stacles. In the absence of such data, a black box method is unable to generate reasonable answers when faced with situations that it has not encountered before. Therefore, to create a more complete training set, we developed a program to generate such cases. Upon the generation of a path and obstacle configuration, a user was asked to select an appropriate steering angle to follow the path and avoid the obstacles. An illustration of one such situation is shown in Figure 18. In this screen-shot, the user is shown a path, an obstacle scenario, and it is given a set of arcs the vehicle could possibly drive. The user has selected the arc second from the right as the one to drive on. Over 500 simulated cases were collected in this manner and added to the training set, also with a mirrored copy.

The input images to this method consist of a map grid of the local area in front of the vehicle with pixel values of −1, 0, and 1 indicating obstacle, free space, and road, respectively. To make the road we expand the desired path by 2 m to either side. Each training data point consists of a vector containing the input image and a list of steering angle votes, which are Gaussian with the operator's steering angle as the mean (see Figure 19). Then we determine the eigenvectors as follows:

First we calculate the average of all training vectors, a. We subtract each training vector by a:

$$\Delta_i = v_i - a .$$

With each vector we form the matrix $A = [\Delta_1 ; \Delta_2 ; \cdots]$ and calculate the covariance matrix C

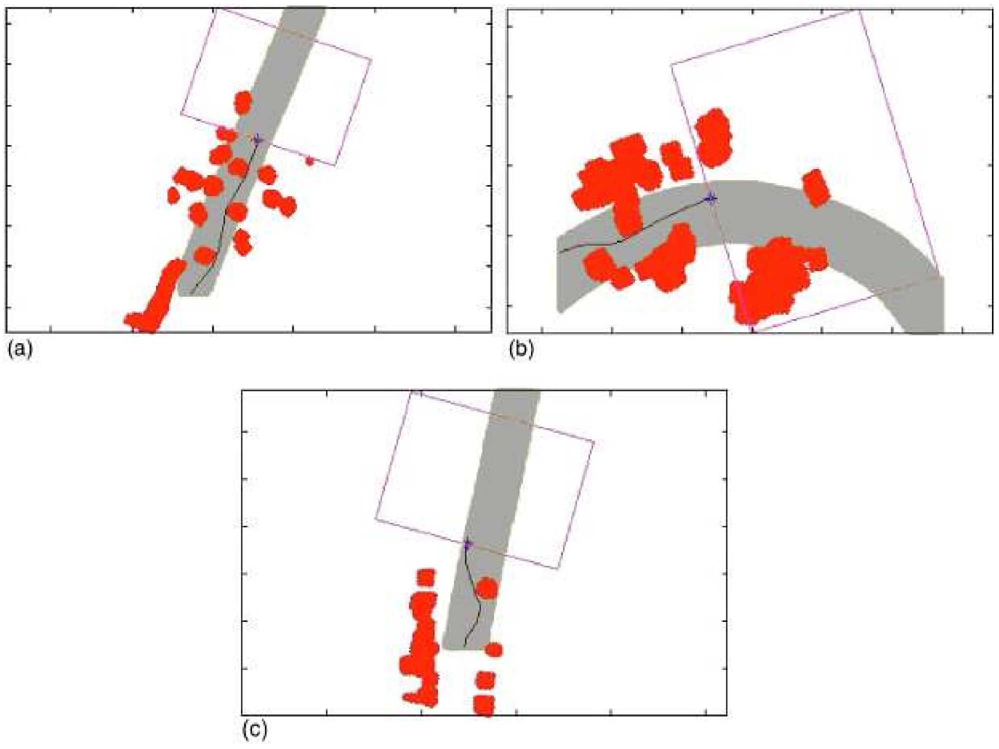

Figure 22. Paths driven by the simulated vehicle using PCA reduction. The system usually steers the vehicle in the right direction, but not enough to avoid the obstacles. Path (c) is the result from Figures 20 and 21.

$=AA^T$. Finally, the matrix C is very large, but we save computation by computing the eigenvectors of $U=A^TA$ and multiplying those vectors by A to get eigenvectors of C. See Hancock & Thorpe (1994) for more details.

Driving using the system consists of the following: an image based on the local map ahead of the vehicle is created. It is projected onto the image portion of the principal eigenvectors (in this case, the top 15 eigenvectors):

$$v = a + \sum_{i=1}^{15} ((x - image(a)) \cdot image(e_i))e_i,$$

where x is the input image, $image(a)$ is the image part of the average vector, and e_i is each of the principal eigenvectors. The steering angle portion of v provides a set of steering angle votes. Finally, the maximum vote is taken as the commanded steering angle. Examples of input images, projected images, and steering angle votes are shown in Figures 20 and 21.

We ran the PCA driving system in simulation on the path segments from its own training data. In general, the system steered the vehicle in the right direction, but consistently understeered. Three resulting paths are shown in Figure 22. A major problem is that there is almost always a steering vote spike to follow the road, regardless of obstacles, as shown in the example figures. Sometimes the spike is large enough to supercede the other possible steering angles, causing the vehicle to not avoid the obstacle. Also, in the eigenvector projection an obstacle can blur or shift slightly, which can allow the system to choose bad steering angles. Lastly, since PCA is a function approximator, it does not actually develop a concept of what an obstacle is. It simply learns what to do when an obstacle is found. For example, frequently in our training data an obstacle appears at a distance in front of the vehicle, and the operator chooses to cross in front of the obstacle to follow the road. In this case the system learns to drive *toward* an obstacle in the distance, which causes the convoluted vehicle path in Figure 22(c).

6. CONCLUSIONS

We have reported an obstacle avoidance control law, originally modeled on pedestrian behavior, for a robot following paths in the presence of previously unknown obstacles. We have presented a method to automatically learn parameters for the control law based on observation of a human driver. We used this method to learn a new set of parameters for our obstacle avoidance system, which previously used a set hand-tuned by us. These new parameters outperformed the hand-tuned set both in simulation and in tests on a vehicle. Using the learned parameters, we were able to decrease the number of vehicle stoppages, improving the overall success rate by over 25%.

Since we only needed to tune the parameters of the control law, rather than learn a complete model of vehicle behavior, we were able to successfully teach the system how to drive using a small amount of data. Function approximators like principal component analysis need to be trained with a very large set of data to learn how to drive the vehicle. They can also be misled by certain training examples where it appears the driver heads toward an obstacle. It is unclear whether, even with additional training data, the PCA system could learn to drive better than our system.

While the vehicle tests are encouraging, a few problems still remain. The vehicle did collide with obstacles in a few instances. Our analysis shows that, in all cases, the vehicle was in the process of slowing down before the collision and suggests that the speed controller needs to react sooner. In some of the cases, where we used large obstacles, the result was a path far from the obstacles. This suggests that we need to normalize the effect of large obstacles and also slow the vehicle down if the prediction indicates a future path that is far from the nominal path. Finally, we noticed oscillations in the path in cluttered environments as the effect of obstacles waned and grew quickly. In future work we will explore the use of a damping term in the steering dynamics, sacrificing reaction time to get a smoother response.

REFERENCES

Bellutta, P., Manduchi, R., Matthies, L., Owens, K., & Rankin, A. (2000, October). Terrain perception for Demo III. In Proceedings of the IEEE Intelligent Vehicles Symposium 2000, Dearborn, MI.

Brock, O., & Khatib, O. (2002). Elastic strips: A framework for motion generation in human environments. The International Journal of Robotics Research, 21(12), 1031–1052.

D'Este, C., O'Sullivan, M., & Hannah, N. (2003, June). Behavioural cloning and robot control. In the IASTED International Conference on Robotics and Applications, Salzburg, Austria (pp. 179–182).

Fajen, B., & Warren, W. (2003). Behavioral dynamics of steering, obstacle avoidance, and route selection. Journal of Experimental Psychology: Human Perception and Performance, 29(2), 343–362.

Fajen, B., Warren, W., Termizer, S., & Kaebling, L. (2003). A dynamical model of steering, obstacle avoidance, and route selection. International Journal of Computer Vision 54(1), 13–34.

Feiten, W., Bauer, R., & Lawitzky, G. (1994, May). Robust obstacle avoidance in unknown and cramped environments. In IEEE International Conference on Robotics and Automation, San Francisco, CA.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. IEEE Robotics and Automation Magazine 4(1), 23–33.

Hancock, J., & Thorpe, C. (1994). ELVIS: Eigenvectors for land vehicle image system (Tech. Rep. CMU-RI-TR-94-43). Pittsburgh, PA: Carnegie Mellon University, Robotics Institute.

Huang, W.H., Fajen, B.R., Fink, J.R., & Warren, W.H. (2006). Visual navigation and obstacle avoidance using a steering potential function. Robotics and Autonomous Systems, 54(4), 288–299.

Lamiraux, F., Bonnafous, D., & Lefebvre, O. (2004). Reactive path deformation for nonholonomic mobile robots. IEEE Transactions on Robotics, 20(6), 967–977.

Minguez, J., & Montano, L. (2004). Nearness diagram (ND) navigation: collision avoidance in troublesome scenarios. IEEE Transactions on Robotics and Automation, 20(1), 45–59.

Ng, A., Kim, H., Jordan, M., & Sastry, S. (2004). Autonomous helicopter flight via reinforcement learning. In S. Thrun, L. Saul, & B. Schölkopf (Eds.), Advances in neural information processing systems 16. Cambridge, MA: MIT Press.

Philippsen, R., & Siegwart, R. (2005, April). An interpolated dynamic navigation function. In Proceedings of the IEEE International Conference on Robotics and Automation, Barcelona, Spain (pp. 3782–3789).

Pomerleau, D. (1995). Neural network vision for robot driving. In M. Arbib (Ed.), Handbook of brain theory and neural networks. Cambridge, MA: MIT Press.

Roth, S., Hamner, B., Singh, S., & Hwangbo, M. (2005, July). Results in combined route traversal and collision avoidance. In Proceedings, International Conference on Field and Service Robotics, Port Douglas, Australia.

Simmons, R. (1996, April). The curvature-velocity method for local obstacle avoidance. In IEEE International Conference on Robotics and Automation, Minneapolis, MN.

Singh, S., Simmons, R., Smith, T., Stentz, A., Verma, V., Yahja, A., & Schwehr, K. (2000). Recent progress in local and global traversability for planetary rovers. In Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, April 24–28, 2000. New York: IEEE.

Talukder, A., Manduchi, R., Castano, R., Owens, K., & Matthies, L. (2002, September–October). Autonomous terrain characterisation and modelling for dynamic control of unmanned vehicles. In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland.

Thrun, S. (1998). Bayesian landmark learning for mobile robot localization. Machine Learning, 33(1), 41–76.

Tilove, R. (1990, May). Local obstacle avoidance for mobile robots based on the method of artificial potentials. In Proceedings of the IEEE International Conference on Robotics and Automation, Tsukuba, Japan (pp. 566–571).

Ulrich, I., & Borenstein, J. (2000, April). VFH*: Local obstacle avoidance with look-ahead verification. In IEEE International Conference on Robotics and Automation, San Francisco, CA.

Urmson, C., Anhalt, J., Clark, M., Galatali, T., Gonzalez, J.P., Gowdy, J., et al. (2004). High speed navigation of unrehearsed terrain: Red team technology for Grand Challenge 2004 (Tech. Rep. CMU-RI-TR-04-37). Pittsburgh, PA: Carnegie Mellon University, Robotics Institute.

Urmson, C., & Dias, M. (2002). Stereo vision based navigation for sun-synchronous exploration. In Proceedings of the International Conference on Robotics and Automation, Washington, DC, May 11–15, 2002. New York: IEEE.

Wellington, C., & Stentz, T. (2003, July). Learning predictions of the load-bearing surface for autonomous rough-terrain navigation in vegetation. In Proceedings of the 4th International Conference on Field and Service Robotics, Lake Yamanaka, Japan.