Evaluation of Different Fitness Functions for the Evolutionary Testing of an Autonomous Parking System

Joachim Wegener[1] and Oliver Bühler[2]

[1]DaimlerChrysler AG, Research and Technology, Alt-Moabit 96 a, D-10559 Berlin, Germany
Joachim.Wegener@DaimlerChrysler.com
[2]STZ Softwaretechnik, Im Gaugenmaier 20, D-73730 Esslingen-Zell, Germany
Oliver.Buehler@stz-softwaretechnik.de

Abstract. The method of evolutionary functional testing allows for the automation of testing by transforming test case design into an optimization problem. To this end it is necessary to define a suitable fitness function. In this paper two different fitness functions are compared for the testing of an autonomous parking system. The autonomous parking system is executed with the test scenarios generated, the fitness for each test scenario is calculated on the basis of an evaluation of the quality of the parking maneuver calculated by the autonomous parking system. A numerical analysis shows, that the proposed area criterion supports a faster convergence of the optimization compared to the proposed distance criterion and that the proposed area criterion describes an efficient method for finding functional errors in the system in an automated way.

1 Introduction

Electronic control units (ECUs) in cars are taking over increasingly complex tasks. New applications such as autonomous parking systems, intelligent cruise control systems, which track the distance to preceding vehicles, or emergency breaking systems rely on the sensor-based calculation of distances to other objects. For such applications, errors in the ECU's software can result in high costs. Therefore, the aim is to find as many errors as possible by testing the systems before they are released. In practice, dynamic testing is the analytical quality assurance method most commonly used. Usually, a complete test is infeasible because of the huge number of possible input situations. Therefore, test cases have to be selected according to test hypotheses, e.g. each requirement should be tested at least once or every program branch should be executed during the test.

In most cases, test case design is performed manually, requiring a considerable part of the project's resources. The evolutionary functional testing method facilitates the generation of test cases in order to detect functional errors during a directed search. The method transforms the test case design process into an optimization problem. Automated test result evaluation is a prerequisite for this process. The evaluation is carried out by means of the fitness function which assigns a numerical quality value to a test result.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 1400–1412, 2004.

This paper evaluates two different approaches to the definition of fitness functions for the functional testing of an autonomous parking system. The fitness functions defined represent a quality metric and can automatically evaluate a parking maneuver calculated by the parking system, i.e. they return a numerical value which describes the quality of the parking maneuver. Both approaches are compared by means of numerical experiments for a prototype implementation of an autonomous parking system. The results show that of both the criteria proposed in this paper, the area criterion can identify critical parking maneuvers better than the distance criterion introduced in [1]. The area criterion provides a more efficient method of error detection in the parking system.

The structure of the paper is as follows: after a brief introduction to evolutionary testing in the second section, the autonomous parking system is introduced and the application of evolutionary testing to its functional testing is explained in the third section. Section 4 describes the different fitness functions, whereas section 5 shows the experimental results achieved by applying these fitness functions. The paper closes with a short summary in the sixth section.

2 Evolutionary Testing

Testing aims to find errors in the system under test and create confidence in its correct behavior by executing the system with selected input situations. Of all the test activities, test case design is assigned decisive importance. Test case design determines the type, scope and thus the quality of the test [2]. If relevant test cases are forgotten, the probability of detecting errors in the system drops. Due to the central importance of test case design, a number of testing methods have been developed to help the tester with the selection of appropriate test data. One important weakness of the testing methods available is that they cannot be automated in a straightforward way. Manual test case design, however, is time-intensive and error-prone. The test quality depends on the performance of the tester. In order to increase the effectiveness and efficiency of the test and thus to reduce the overall development and maintenance costs for systems, a test should be systematic and extensively automatable. Both these objectives are addressed by the evolutionary testing method [3].

In order to transform a test aim into an optimization task a numerical representation of the test aim is necessary, from which a suitable fitness function for the evaluation of the test data generated can be derived. Depending on which test aim is pursued, different fitness functions emerge for test data evaluation. If, for example, the temporal behavior of an application is being tested, the fitness evaluation of the individuals created by evolutionary testing will be based on the execution times measured for the test data [3]. For safety tests, the fitness values are derived from pre- and post-conditions of modules [4], and for robustness tests on fault-tolerance mechanisms, the number of controlled errors can form the starting point for the fitness evaluation [5]. Applications of evolutionary testing to structural testing result in different fitness functions again [6], [7], [8], [9]. An overview of the different applications of evolutionary testing can be found in [10].

If an appropriate fitness function can be defined, the evolutionary test proceeds as follows. The initial population of test data is usually generated at random. Each individual within the population represents a test datum with which the system under test is executed. For each test datum the execution is monitored and the fitness value is determined with respect to the test aim defined. Next, population members are selected with regard to their fitness and subjected to combination and mutation processes to generate new offspring. These are then also evaluated by executing the system under test with the corresponding test data. A new population is formed by combining offspring and parent individuals, according to the survival procedures laid down. From here on, the process repeats itself, starting with selection, until the test objective has been fulfilled or another given stopping condition has been reached.

3 The Evolutionary Testing of the Autonomous Parking System

As an automobile manufacturer, DaimlerChrysler is continuously developing new systems in order to improve vehicle safety, quality, and comfort. Within this context, prototypical vehicle systems have been developed which support autonomous vehicle parking – a function that is likely to be introduced onto the market in some years time. In order to describe the application of evolutionary testing for the functional testing of autonomous parking systems, we shall first provide a brief description of the functionality of an autonomous parking system. We shall then explain the application of the evolutionary test to the parking system. The test environment developed for testing the parking systems and the test data generator are also described.

3.1 The Autonomous Parking System

The autonomous parking systems dealt with in this paper are intended to automate parking lengthways into a parking space, as shown in Fig.1. For this purpose, the vehicle is equipped with environmental sensors which register objects surrounding the vehicle. When driving along, the system can recognize sufficiently large parking spaces and can signal to the driver that a parking space has been found. If the driver decides to park in the parking space detected, the vehicle can do this automatically.

Fig.2 shows the system environment and the internal structure of the autonomous parking system. The inputs are sensor data, containing information on the state of the vehicle, e.g. vehicle speed or steering position, and information from the environmental sensors which register objects on the left and right hand side of the vehicle. As regards output, the system possesses an interface to the vehicle actors, where the vehicle's velocity and steering angle are set.

The parking space detection component processes the data obtained from the environmental sensor systems and delivers the recognized geometry of a parking space if it has been identified as sufficiently large. The parking controller component

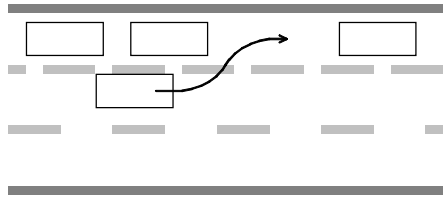

Fig. 1. Functionality of an Autonomous Parking System

uses the geometry data of the parking space together with the data from the vehicle sensors to steer the vehicle through the parking procedure. For this purpose, velocity and steering angle are set for the vehicle actors. The parking controller has to guarantee that the collision area is not entered by the vehicle in order to avoid a high probability of causing damage to adjacent parked vehicles or objects and the vehicle itself.

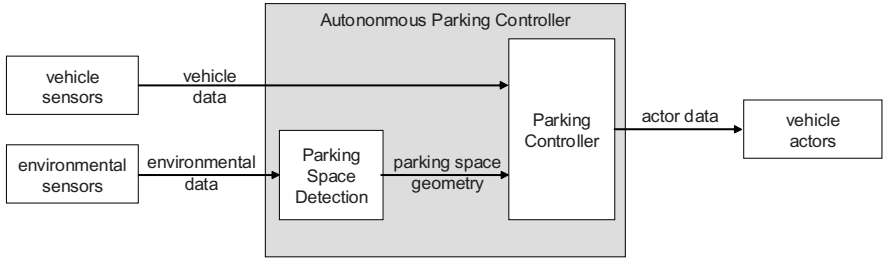

Fig. 2. System Environment and Sub-Components of the Autonomous Parking System

3.2 Applying Evolutionary Testing to the Autonomous Parking System

Comprehensive and efficient testing is essential before releasing a system such as the automatic parking system. As many tests as possible must be performed in a systematic and efficient way. Manual testing of the complete system is cost-intensive and time-consuming because every test case involves setting up a parking scenario with real cars and the manual driving of each maneuver. Furthermore, it is difficult to achieve an exact reproduction of the tests because the details of the test execution vary. In contrast, automated tests have the potential to perform a large number of test cases with less effort in a reproducible manner. Therefore, automated functional tests performed in a controlled simulation environment should be integrated into the quality assurance process of the autonomous parking system.

Evolutionary functional testing provides a way of automating functional tests as a complete process. Instead of selecting the test cases manually, a search for interesting test cases is performed automatically. This is done by translating test case selection into an optimization problem. The possible input situations of the system under test are mapped to the search space. On the one hand, the mapping should keep the size of the search space as small as possible, and on the other hand, the mapping should be

able to produce all possible input data for the system. If one considers the whole input range during the design of the test data generator, it does not mean that all the test cases in this range will actually be tested but it does provide the possibility of generating any test data required. An appropriate model has to be designed for this purpose.

Both components of the autonomous parking system have to be tested thoroughly. In the case of the parking space detection component, we must make sure that suitable parking spaces are identified precisely whereas parking spaces too small for a parking maneuver have to be rejected. Usually, the parking space detection has to be tested in natural environments since reliable simulation models for the environmental sensors are not available. Therefore, it is not considered in the subsequent sections of this paper. Nevertheless, we plan to test the parking space detection in the future by generating environmental data.

For the testing of the parking controller, test cases describing different parking scenarios are generated. The evaluation of the test cases is carried out by the fitness function. In order to test the automatic parking system, the fitness function calculates a numerical fitness value for each parking scenario generated. This fitness value represents the quality of the corresponding test case and aims to lead the evolutionary search into a direction of input situations which result in a parking controller error. Therefore, the fitness function is designed to assign good fitness values to parking scenarios which cause the system to enter the collision area or end up in an inadequate parking situation. Bad fitness values are assigned to scenarios which result in a good parking position with enough distance to the collision area.

3.3 Test Environment

The test environment of the automatic parking system (Fig. 3) comprises the simulation environment, an evolutionary algorithm toolbox, an implementation of the fitness function and the test data generator which translates individuals into actual parking scenarios. The test object is the control unit of the vehicle with the implementation of the automated parking system inside.

The GEA toolbox for Matlab (www.geatbx.com) was used to implement the evolutionary algorithms used to test the autonomous parking system. The simulation environment (built up on a Matlab R12.1 platform) simulates the properties of the vehicle as well as the surrounding environment. It runs with the "in-the-loop" control unit, meaning that the simulation environment calculates the sensor data for the vehicle and presents it to the parking controller inside the control unit. The control unit processes this sensor data and reacts to it with control data for the simulation environment. This loop simulates a complete parking maneuver for the parking space scenario generated. The parameters describing a parking space scenario for the simulation, such as the position of the car and the size of the parking space, are outputs of the test data generator (see 3.4.). After the simulation of a parking maneuver the fitness value is calculated, using the fitness functions described in the subsequent section, and is then assigned to the individual generated.

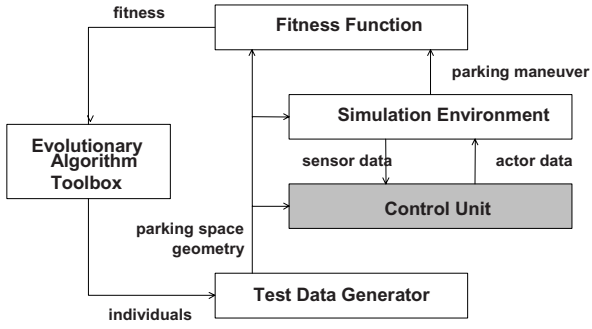

Fig. 3. Design of the Test Environment

3.4 Test Data Generator

The geometric data which characterizes a parking space comprises six points P0 to P5, and is referred to as parking space geometry. The points define the border between the drivable and impassable area of the parking situation. The model for the generation of this parking space geometry is shown in Fig.4. It is a simplified model because the borders of the parking space are always rectangular. The shape of the parking space can only vary in length and depth.

This model takes the values of five independent variables and calculates the parking space geometry from them. The independent variables define *length* and *width* of the parking space. In addition, the starting position with the *distance* of the car to the parking space (*dist2space*), the angle *psi* and the *gap* between the vehicle and the collision area on the right hand side of the car are part of the parking space scenarios generated.

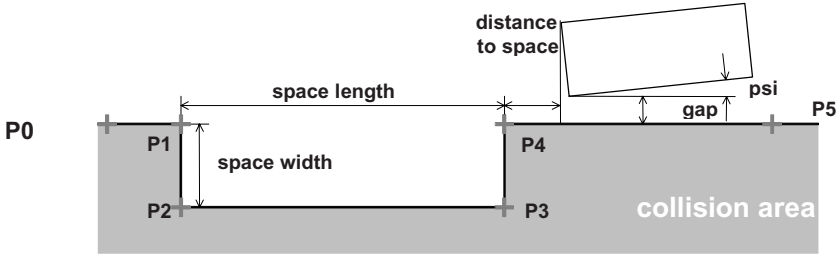

Fig. 4. Model for the Generation of Parking Space Geometry

4 Definition of Fitness Functions

This section describes the definition of two different strategies for the evaluation of the fitness of a parking scenario. One strategy uses the distance between the vehicle and the collision area as a measure of the evaluation of fitness [1], the other strategy

works with the area between the vehicle and the collision area. Both strategies separate the parking space into two parts: (1) collision with the preceding vehicle and (2) collision at the parking side. While the vehicle is pulling into the parking space its freedom of movement is limited and thus, if a collision with the preceding vehicle or the parking side occurs, either the right rear edge or the right front edge of the car will be involved. Depending on the position of the collision areas the lines between P3-P4 and P5-P4 have to be observed in order to identify a collision with the front vehicle. To identify a collision at the side, the line P2-P3 has to be observed. The definition distinguishes between the observation of a corner, defined by three points, and the observation of an edge, defined by two points.

The fitness function is intended to assess a parking scenario and to assign an adequate fitness value to it. The fitness value should correspond to the quality of the parking scenario. From the testing perspective it is a good parking scenario if a collision with other parked cars occurs or the controlled vehicle touches the parking side. Since we are minimizing the fitness values during the search process, the fitness value assigned to a parking scenario decreases with the quality of the scenario. An interesting parking scenario achieves a smaller value than a parking scenario for which the automatic parking system performs well. The fitness becomes negative when a collision occurs.

4.1 The Distance Criterion Fitness Function

The distance criterion considers the closest distance between a vehicle edge and the collision border during the parking maneuver. In separate evaluations the smallest distance of the collision corner P3-P4-P5 and the smallest distance from the collision side P2-P3 are calculated. The following subsections describe, how the evaluation of the collision corner and the collision side is carried out.

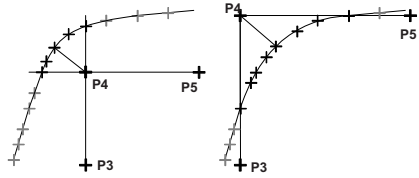

Fig. 5. Selection of Smallest Distance

Evaluation of a Collision Corner

The collision corner is defined by three points P3-P4-P5. All distances are calculated as polar coordinates with P4 as their origin (Fig. 5). The evaluation of the collision corner observes the section defined by P3-P4-P5 and the section diagonally opposite. Only the points within these sections are considered. As a quality measure for the evaluation of the parking maneuver, the smallest distance between the vehicle path positions and point P4 is taken. The value of the distance is positive, if the path is

outside the collision corner. The value is set to zero, if the path crosses P4. The value is measured as a negative value, if the path runs through the collision corner (Fig. 6).

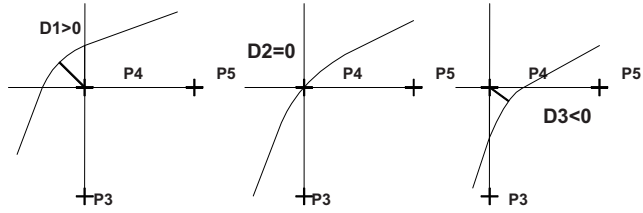

Fig. 6. Signed Distance Values

This strategy aims to ensure, that the closer the path gets to the collision corner, the lower the assigned fitness value becomes. For different paths which continuously move into the collision corner, the corresponding fitness values become continuously lower, as shown in Fig.7, where D1 > D2 > D3.

Evaluation of a Collision Side

The collision side is defined by the straight line between P2-P3. The distances calculated are between this line and the path positions of the car. In this calculation, only path points whose x-values are within the range of P2 and P3 are taken into account. The selection is carried out by comparing the x-coordinates of the path points with P2 and P3.

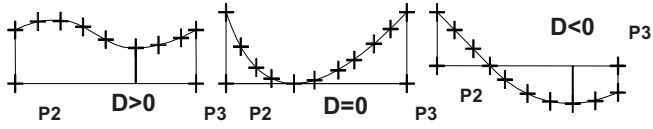

Fig. 7. Distance from Line with Positive, Zero and Negative Value

The distance is calculated positive, when the path is above the line, and is set to zero, when the path touches the line. The distance is calculated as a negative value, when the path runs through the collision area. From all the distance values calculated, the minimum is taken as the fitness value for the parking space scenario generated.

4.2 The Area Criterion Fitness Function

The area criterion fitness function considers the area between the path of the vehicle and the parking geometry. Here, the areas included in the collision corner and the collision side are calculated in separate evaluations. The following subsections describe how the evaluation of the collision corner and the collision side is carried out with the area criterion.

Evaluation of a Collision Corner

The area included between the corner lines and the vehicle path is taken as a measure of the evaluation of the parking scenario. In order to calculate this area, it is separated into smaller segments, appropriate to the points of the vehicle's path through the corner, as shown in Fig.8. The overall area A included in the collision corner is the sum of all the segments together.

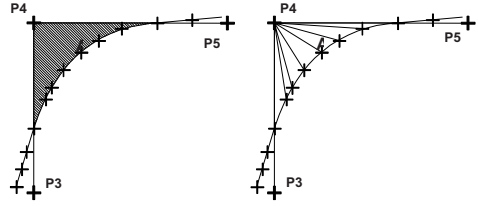

Fig. 8. Separation of Included Area into Small Segments

For an effective and fast calculation of the fitness value, an approximate description of the area of each segment can be provided by a triangle. When the distances between two points of the path T_n and T_{n+1} is significantly smaller than the distance to P4 the angle $alpha_n$ is very small.

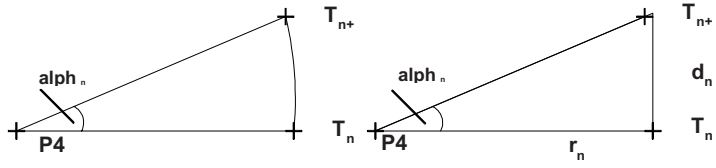

Fig. 9. Approximation of Segment by a Triangle

For small angles of $alpha_n$, the ratio d_n to r_n is approximately the $angle_n$ in radians. The area of one segment which is approximately described by a triangle is:

$$A_n = \frac{1}{2} \times r_n \times d_n \text{ and with } d_n \approx r_n \times \left(th_{n+1} - th_n\right) \tag{1}$$

the area of a segment can be calculated thus:

$$A_n \approx \frac{1}{2} \times {r_n}^2 \times \left(th_{n+1} - th_n\right). \tag{2}$$

The angle th_n and the radius r_n for each path point T_n can be easily obtained, by transferring the path positions from Cartesian coordinates into polar coordinates with P4 as the origin of the coordinate system. The overall area of the corner is the sum of all the segments within the corner

$$A_{corner} = \sum A_n \ . \tag{3}$$

To lead the optimization towards a collision, the corner P3-P4-P5 symmetrical to point P4 is also taken into consideration. The path of the vehicle has to pass the point P4 and the aim of the optimization is to bring that path into the collision area. The idea is to rate an area included in the opposite corner as a positive value and an area included in the collision corner as a negative value. When the vehicle path crosses through the corner point P4 the corresponding value is set to zero. Thereby, the areas shown in Fig.10 become A1 > A2 > A3 when the vehicle path keeps on shifting into the collision corner. This leads to a gradual improvement of the fitness value, depending on how near the vehicle path passes by the collision corner or crosses into it.

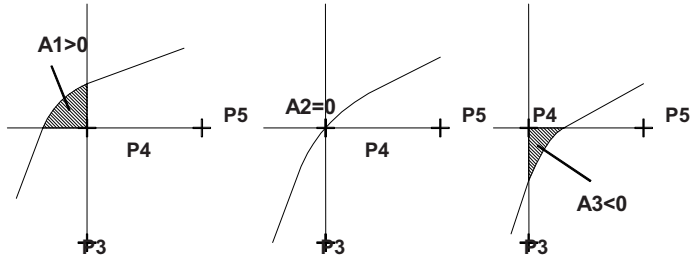

Fig. 10. Signed Area Values

Evaluation of a Collision Side

The evaluation of a collision side takes the area included between the vehicle path and the straight line P2-P3 as a measure. The calculation of the area takes only those points into consideration which have their x-coordinate in the range between P2 and P3. The calculation is carried out using an approximation, with the rectangles defined by the path positions. The area of each rectangle can be easily calculated by Δx and the distance between path and line Δy. With a sufficient number of path points in the range between P2 and P3, a good approximation of the area included can be achieved.

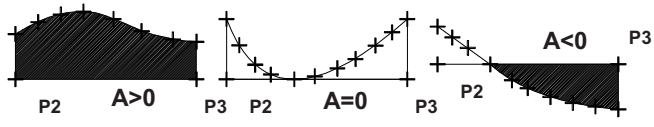

Fig. 11. Area with Positive, Zero and Negative Value

Fig.11. (1) shows when the path is above the line, (2) the path touches the line or (3) the path crosses the straight line into the collision area. In the first case, the area included has a positive value, in the second case the value for the area is set to zero. In the third case the area below the line has a negative value. The distinction drawn between the three cases prevents a larger area above the line from compensating for small areas below the line. It also prevents a situation when the line is touched from being concealed by adjacent areas so that it cannot be observed.

5 Experiments

The aim of the experiments is to analyze and compare the suitability of the fitness functions described in the previous section for the testing of autonomous parking systems. A number of experiments were performed to analyze the fitness landscapes for both fitness functions. For each experiment, two variables from the test data generation input vector were varied within a defined range and a defined number of samples. The remaining three variables were kept constant resulting in two-dimensional variation plots of the fitness landscapes. In order to calculate the fitness value for each parking scenario, a parking maneuver carried out by the autonomous parking system was simulated as described in section 3.2.

Fig. 12 shows the results for the experiment in which the variables *dist2space* and *psi* were varied, and in which the *length* and *width* of the parking space as well as the *gap* to the right of the vehicle are kept constant: *length* was set to 8.0 m, *width* to 2.5 m and the *gap* to 0.7 m. The axis to the right shows the *distance* to the parking space in the range of 0.0 m to 7.0 m, with a resolution of 70 points. The axis at the bottom shows the angle *psi* from +10° to –10°, with a resolution of 40 points. Both functions return negative fitness values when angle *psi* reaches +10° and *dist2space* goes towards 7 m. A considerable difference between the shapes of both surfaces is that for fitness values greater than zero the distance criterion returns constant small values where the area criterion lowers its values towards the border to zero. This appears as a flat plateau in the distance criterion surface (on the right hand side of Fig. 12) where the area criterion surface slopes continuously (left hand side of Fig. 12).

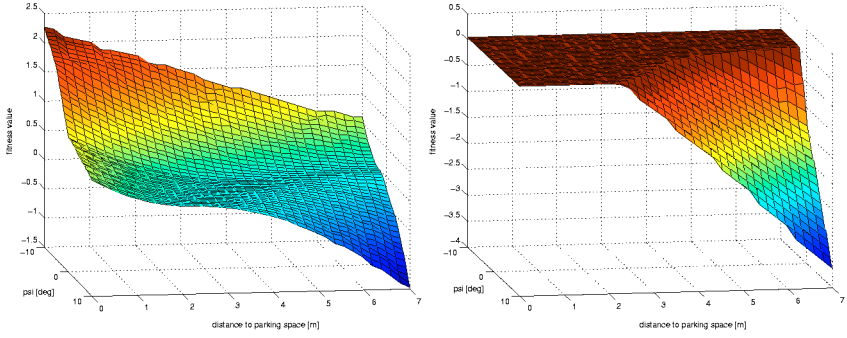

Fig. 12. Fitness Landscapes for Area Criterion (left) and Distance Criterion (right) as a Function of Angle psi and Distance to Parking Space

Comparable results were found in most experiments, e.g. Fig. 13 shows the plots for the variation of *dist2space* and parking space *length*. The range of the *length*, shown on the axis to the right, is between 5 and 10 m. *Dist2space* is shown on the axis to the left and ranges from 0 to 7 m. The remaining variables are kept constant: *width* of parking space was set to 2.5 m, the *gap* to the right of the vehicle was set to 0.7 m and the angle *psi* was set to 0°. As in the preceding example, the surfaces for

the area criterion and the distance criterion differ. The distance criterion surface has a flat plateau of equal fitness values for small distances to the parking space and longer parking spaces. In contrast, the area criterion shows a sloping characteristic for that domain.

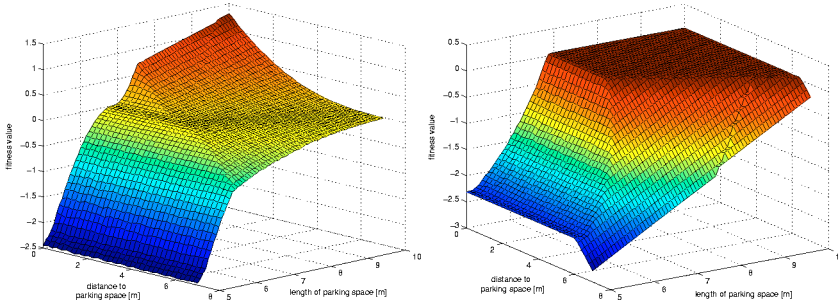

Fig. 13. Fitness Landscapes for Area Criterion (left) and Distance Criterion (right) as a Function of Distance to and Length of the Parking Space

6 Conclusion

The experiments have shown that both fitness functions have a sloping characteristic for collision maneuvers, where their fitness value is less than zero. The slope of the distance criterion function is steeper in that domain than that of the area criterion function. Furthermore, the distance criterion function shows an angle at the point where the values enter the negative range whereas the area criterion function has a smooth transition for fitness values around zero. Both fitness functions guide the search towards scenarios in which a collision occurs. The area criterion function provides a sloping characteristic for maneuvers with positive fitness values, in contrast to the distance criterion function, which returns constant fitness values resulting in plateaus of equal fitness in the search space for scenarios without a clash. It does not differentiate between good and better maneuvers. As a consequence, the area criterion function is better suited to directing the search towards collision maneuvers than the distance criterion function. It helps us to find test cases more quickly for which the autonomous parking system does not react correctly. Therefore, the area criterion will be used for the evolutionary testing of autonomous parking systems in the future.

Acknowledgments. The work described was carried out as part of the SysTest project., funded by the EC under the 5th framework programme (GROWTH, project ref. G1RD-CT-2002-00683).

References

1. Buehler, O., Wegener, J.: Evolutionary Functional Testing of an Automated Parking System. Proceedings of the International Conference on Computer, Communication and Control Technologies (CCCT '03) and the 9th. International Conference on Information Systems Analysis and Synthesis (ISAS '03), Florida, USA (2003).
2. Grochtmann, M., Grimm, K.: Classification-Trees for Partition Testing. Software Testing, Verification & Reliability, vol. 3, no. 2, pp. 63-82 (1993).
3. Wegener, J., Grochtmann, M.: Verifying Timing Constraints of Real-Time Systems by Means of Evolutionary Testing. Real-Time Systems, vol. 15, no. 3, pp. 275-298 (1998).
4. Tracey, N., Clark, J., Mander, K.: The Way Forward for Unifying Dynamic Test Case Generation: The Optimisation-Based Approach. Proceedings of the IFIP International Workshop on Dependable Computing and Its Applications, South Africa, pp. 169-180 (1998).
5. Schultz, A., Grefenstette, J., Jong, K.: Test and Evaluation by Genetic Algorithms. IEEE Expert, vol. 8, no. 5, pp. 9-14 (1993).
6. Jones, B., Sthamer, H., Eyres, D.: Automatic Structural Testing Using Genetic Algorithms. Software Engineering Journal, vol. 11, no. 5, pp. 299-306 (1996).
7. Pargas, R., Harrold, M., Peck, R.: Test-Data Generation Using Genetic Algorithms. Software Testing, Verification & Reliability, vol. 9, no. 4, pp. 263-282 (1999).
8. Michael, C., McGraw, G., Schatz, M.: Generating Software Test Data by Evolution. IEEE Transactions on Software Engineering, vol. 27, no. 12, pp. 1085-1110 (2001).
9. Tracey, N., Clark, J., Mander, K., McDermid, J.: An Automated Framework for Structural Test-Data Generation. Proceedings of the 13th IEEE Conference on Automated Software Engineering, Hawaii, USA (1998).
10. McMinn, P.: Search-based Software Test Data Generation: A Survey. To appear in Software Testing, Verification & Reliability & Reliability (2004).