Proceedings of 2004 IEEE/RSJ International Conference on
Intelligent Robots and Systems
September 28 - October 2, 2004, Sendai, Japan

CAUTIOUSBUG : A Competitive Algorithm for Sensory-Based Robot Navigation

Evgeni Magid
Dept. of Mathematics
Technion – Israel Institute of Technology
Haifa, Israel
Email: evgenue@tx.technion.ac.il

Ehud Rivlin
Dept. of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel
Email: ehudr@cs.technion.ac.il

***Abstract*— *Bug* algorithms are a class of popular algorithms for autonomous robot navigation in unknown environments with local information. Very natural, with low memory requirements, *Bug* strategies do not yet allow any competitive analysis. The bound on the robot's path changes from scene to scene depending on the obstacles, even though a new obstacle may not alter the length of the shortest path. We propose a new competitive algorithm, *CautiousBug*, whose competitive factor has an order of $O(d^{m-1})$, where d is the length of the optimal path from starting point S to a target point T. $m = 2^{\#Min-1}$ and $\#Min$ denote the number of the distance function isolated local minima points in the given environment. Simulations were performed to study the average competitive factor of the algorithm.**

I. Introduction

Autonomous navigation of indoor mobile robots has received considerable attention in recent years. Work in this area was motivated by applications such as office cleaning, cargo delivery etc. In realistic settings, an autonomous robot cannot base its motion planning on complete a priori knowledge of the environment. The robot must use its sensors to perceive the environment and plan accordingly. The two main sensor-based approaches use either global planning or local planning.

In the global sensor-based planning approach, the mobile robot builds a global world model based on sensory information and uses it for path planning. This approach guarantees that the target will be reached or the robot will conclude that the goal is unreachable. However, the construction and maintenance of a global model based on sensory information imposes a heavy computational burden on the robot. Local path-planners use the local sensory information in a purely reactive fashion and do not guarantee global convergence.

A midway approach, originated by Lumelsky and Stepanov [12], combines local planning with global convergence: essentially reducing the reliance on a global model to loop detection. Purely reactive navigation decisions guarantee global convergence. Algorithms $Bug1$ and $Bug2$, presented in [12], use only position and contact sensors. These algorithms consist of two reactive modes of motion – moving directly towards the target and following an obstacle boundary – and transition condition for switching between them. When the robot hits an obstacle it switches from moving toward a target to following a boundary. It moves away from the obstacle boundary when a leaving condition, which ensures that the distance to the target decreases, holds. *Bug* algorithms can be divided into two groups according to their memory requirements. The first group inherits the main property of the original $Bug1$ and $Bug2$ – minimal memory requirements [8], [9], [11], [13], [14], [15]. The second group, originating from Sankaranarayanan's $Alg1$ and $Alg2$ [19],uses different data structures to store *Hit* and *Leaving* points together with some useful information about the followed path [4], [16]. Algorithms within one group differ mainly by their leaving conditions and the way they choose boundary following directions.

The *Bug* approach minimizes the computational burden on the robot while still guaranteeing global convergence to the target. However, the original *Bug* algorithms do not make the best use of the available sensory data to produce short paths. *VisBug* [11] uses range sensors only to find shortcuts to the path generated by $Bug2$. *DistBug* [9] uses range sensors to define a new leaving condition and to choose an initial boundary following direction with respect to local range data. *TangentBug* [8] expands on the existing *Bug* family algorithms, being specially designed for using range data and incorporating the notion of the locally shortest path into the general *Bug* paradigm. *TangentBug* uses a *local tangent graph* (LTG) for choosing the locally optimal direction while moving toward the target, for making local shortcuts and for testing a leaving condition while moving along an obstacle boundary.

Recently, one of the main requirements of the navigation strategy has been its *competitiveness* [2], [3], [5], [6], [18]. When analyzing its competitiveness, the robot's performance is measured with respect to the optimal path between start point S and target point T. The optimal path is a subgraph of the *connectivity graph* [10], representing the shortest path a robot would choose given complete information about the environment. The ratio of the robot's path length and the optimal path is called the *competitive factor* of the strategy. The competitive factor is a characteristic property of the navigation task alone, guaranteeing a good worst-case behavior. Often, the true competitive factor is considerably smaller than the upper bound we are able to prove.

All algorithms of the *Bug* family produce a worst-case path length, which is expressed in the notion of perimeters of the obstacles intersecting the disk of radius $||S - T||$ centered at the target point. Hence, the main disadvantage of the *Bug* family algorithms is that they are not competitive. The boundary-following direction is chosen based on local information and may be "wrong" in the sense that it will result in a longer path. The mechanism of switching the boundary following direction, presented in [7], can solve this problem in some simple non-generic cases, but fails in more complicated scenario. Thus, by choosing a "wrong" direction, *Bug* algorithms will produce a catastrophically long path, while in reality there exists a much shorter path. This becomes a disaster in complex environments.

The name of our algorithm, *CautiousBug*, is used to show its main feature – it does not make daring decisions. While *Bug* algorithms take a risk choosing the boundary-following direction at the switch point, *CautiousBug* executes a conservative spiral search in both directions of the boundary.

II. Spiral Search

Suppose there is a very long wall with a door in it. At some point along the wall a mobile robot is located. Its task is to get to the other side of the wall. The robot does not know whether the door lies to the left or to the right of its start position S_w. The robot could pick one direction, left or right, and then walk along the wall in this direction forever. If it happens to guess the correct direction the door will be reached without having to detour. But if it chooses the wrong direction, the door will never be reached. To avoid this situation, the robot should alternate directions, and explore both the left and the right part of the wall in turn. It moves one step to the right, and returns. Then the length of the step is doubled and the robot steps in the opposite direction, and again returns to its start position (Figure 1). Suppose that p_a and p_b are the two concurrent paths of the search and the target is at the distance d units from the origin on p_a. If the distance d is slightly bigger than a power of 2, $d > 2^j$, then the path is of total length

$$2\sum_{i=0}^{j+1} 2^i + d = 8 \times 2^j - 2 + d < 9d,$$

exceeding the distance to the door only by a factor of 9. Baeza-Yates et al. in [1] showed that no smaller competitive factor than 9 can be achieved for this problem.

Theorem 2.1: The doubling strategy in [1] for finding a door in a wall is competitive with factor 9, and this is optimal.

Lemma 2.1: If a point robot follows the Spiral Search strategy in [1], the furthest distance walked by the robot in the wrong direction is 2(d-1) units.

In our strategy, the robot executes this spiral search along the edges of the LTG in the obstacle boundary following mode.

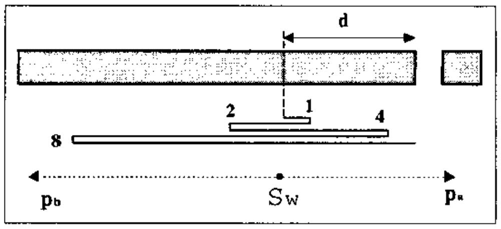

Fig. 1. Spiral search

III. CautiousBug Algorithm

The algorithm navigates a point robot in a planar unknown configuration space populated by stationary rectilinear obstacles[1]. We assume that all obstacles in the C-space have a minimal thickness, i.e. we can encapsulate a unit square into any obstacle or its part. The robot knows its precise coordinates at any point and the exact positions of the *Start* and *Target* points. We assume that there are no errors in the walked path estimation due to odometry errors, frequency etc. The sensory input consists of the robot's current position x and the distance from x to the obstacles within a detection range R. At each moment the robot can measure the distance $d(x,T)$ from its current location x to the target T. This distance function $d(x,T)$ determines the behavior of the *CautiousBug*.

A. The Algorithm Description

CautiousBug uses two basic motion modes: motion toward the target (decreases $d(x,T)$) and following an obstacle boundary (used to escape from the local minimum of $d(x,T)$). At every step the robot constructs the LTG, based on its current range readings. It uses the LTG to plan its next actions as follows. During motion toward the target, the robot moves in the *locally optimal direction*, which is the direction of the shortest path to the target according to a subgraph of the LTG. The motion toward the target terminates when the robot detects that moving in the locally optimal direction will bring it into a local minimum of the distance function $d(x,T)$. At this point the motion mode is switched to the obstacle-following behavior. The initial boundary-following direction is chosen and the robot starts the *spiral search* along the obstacle boundary, continuously monitoring the LTG until it finds a suitable leaving point, which satisfies the *leaving condition*. The leaving condition checks that $d(x,T)$ can be decreased relative to the shortest distance to the target observed along the path so far. Here is a summary of the algorithm:

1) Move toward T along the *locally optimal direction* on the current LTG subgraph, until one of the following events occurs:
   - The target is **reached**. Stop.
   - A local minimum is detected. Go to step 2.
2) Choose a boundary following direction. Move along the boundary with *spiral search* strategy using the

[1]The restriction of the obstacles to being rectilinear is needed for the competitiveness proof only.

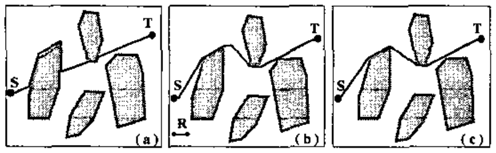

Fig. 2. The robot in a simple convex environment uses only moving-toward-the-target behavior. Unlimited sensors can change the path significantly, but do not guarantee coincidence with the optimal path. (a) contact sensors (b) limited sensors with range R (c) unlimited sensors

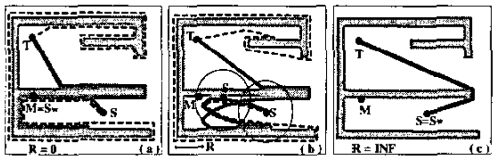

Fig. 3. In the concave environment the robot uses both moving toward-target and boundary-following behavior with $TangentBug$ (dashed line) and CautiousBug (solid line). (a) contact sensors (b) limited sensors with range R (c) unlimited sensors. In this particular scene the path produced by both algorithms coincides, but this does not happen in all scenes

LTG for shortcuts while recording $d_{min}(T)$, until one of the following events occurs:

- The target is **reached**. Stop.
- The leaving condition holds: $\exists$ V $\in$ LTG s.t. $d(V,T) \leq d_{min}(T)$. Go to step 1.
- The robot completes a loop around the obstacle. The target is **unreachable**. Stop.

Three kind of motions are considered: motion in free space between obstacles; sliding along an obstacle boundary; following an obstacle boundary in a spiral search. Moving-toward-the-target motion mode uses the first two kinds of motion – as long as the distance function $d(V,T)$ decreases.

B. Moving Toward the Target

While moving towards the target, the robot moves in a *greedy locally optimal direction*, which is the direction along the shortest path to the goal according to LTG (Fig. 2). The candidate motion directions are the directions toward LTG nodes. A node O_i is considered as a candidate only if it is closer to the target than the current robot location $d(O_i,T) < d(x,T)$. The expected path length to the target is calculated for each candidate direction, assuming that the sensed obstacles are thin walls and that they are the only obstacles in the environment. If there is no *blocking obstacle* between the robot and the target within the sensor range, the shortest path will be along the edge toward T_{node}; otherwise, the endpoints of the blocking obstacle are the candidates for the shortest path.

The motion-toward-target stops when the robot detects that moving in the locally optimal direction will drive it into a local minimum, created by the blocking obstacle. As the robot moves toward the target, it monitors the closest point to the target M on the boundary of the blocking obstacle. If the distance $d(M,T)$ is less than the distances $d(O_1,T)$ and $d(O_2,T)$ to the target from the endpoints O_1,O_2 of the blocking obstacle, the robot concludes that it is within the basin of the local minimum M and switches to boundary-following behavior (Fig. 3).

C. Following the Obstacle Boundary

The obstacle-boundary-following behavior is used to drive the robot away from a local minimum. The robot conducts the *spiral search*, moving around the obstacle until either the leaving condition holds or a loop around the obstacle is completed. The LTG is used to plan local shortcuts relative to the obstacle boundary and for testing the leaving condition.

When obstacle-boundary-following behavior is initiated, in a *switch point* Sw, the algorithm defines the followed obstacle as the current blocking obstacle, that is, the one which caused the local minimum. During boundary-following, the followed obstacle may differ from the blocking obstacle, when the followed obstacle is not the one that blocks the target direction. Initially, a *minimum point* M is chosen to be a local minimum point that will cause the switch to the boundary-following mode. Initial boundary-following direction dir_1 is chosen using the local properties of the boundary of the followed obstacle [7] . At each step the robot constructs the LTG, locates the followed obstacle in it, and focuses on those nodes of the LTG that lie on the followed obstacle. The motion direction during the boundary-following is toward the left/right endpoint of the followed obstacle. The robot executes the spiral search, using the switch point Sw as an origin of the *search rays*, which are the edges of the LTG. The length of the path walked by the robot from the origin of the search rays Sw to the current point x is registered in d_{new}. Variable d_{prev} stores the length of the previous step of the spiral search, performed in the opposite direction. After the initial boundary-following direction dir_1 is chosen, d_{prev} is initialized with the distance to the farthest point on the obstacle boundary seen within the sensor range in the opposite direction dir_2. In the case of a contact sensor, d_{prev} is initialized with some very small value ϵ. When d_{new} exceeds $2d_{prev}$, the boundary-following direction is switched to the opposite direction and $d_{prev} \leftarrow d_{new}$.

As the robot moves around the followed obstacle it updates two variables, which register the minimal distance observed along the path: the shortest distance from the target **on the followed obstacle's boundary** $d_{followed}(T)$, initialized with $d(M,T)$, and the shortest distance from target **within the visible environment** $d_{reach}(T)$. To ensure that the robot stays on the locally optimal paths, $d_{reach}(T)$ is updated by the shortest distance from the target of the blocking obstacle's boundary or, when there is no blocking obstacle, by $d(T_{node},T)$. The robot leaves the obstacle boundary when it can reach, via free space, a point that is closer to the target than the distance $d_{followed}(T)$. The test guarantees that the robot will not be trapped again in the local minima, which it passed during the boundary-following motion. At each step the leaving condition $d_{reach}(T) < d_{followed}(T)$ is tested; when it is satisfied, the robot leaves the obstacle boundary.

IV. Analysis

We denote the L_2 distance from the start point S to the target point T with ST and the shortest path from S to T in the given environment with D. Suppose that the length of the path D, which is a part of a *connectivity graph*, is d. Being interested in the competitiveness of the algorithm, we analyze the produced path length in terms of d. The distance along the obstacle boundary walked by the robot from point A to point B, both lying on the boundary of the same obstacle, we denote with $db(A, B)$. We first explain some geometric properties, which will be useful in the competitive analysis of the strategy.

Lemma 4.1: The maximum possible perimeter of a rectilinear polygon inscribed in an $m \times n$ grid is mn.

Lemma 4.2: Among figures with the same boundary perimeter, the figure with a maximal square is a disk.

Lemma 4.3: The number of isolated local minima points of d(x,T) over the free configuration space is finite. [7]

In our analysis we use a technique similar to one in Datta and Soundaralakshmi [3]. We assume an *adversary* who knows our strategy and designs an environment in such a way that the robot will be forced to cover as much distance as possible following the strategy, that is, with the minimal presence of motion toward target, but maximal boundary-following.

Lemma 4.4: The adversary will design the obstacles in the environment in a way such that the robot reaches the maximum number of Hit and Leaving points in the scene.

Consider hit point H_j. When the robot executes the spiral search for the next leaving point L_j, with H_j as the origin, there are two paths along the boundary: p_{j_a} and p_{j_b}. We assume w.l.o.g. that path p_{j_b}, if chosen, will finally bring the robot to T or to a better alternative of a *leaving* point AL_j, satisfying the relation $d(AL_j, T) \leq d(L_j, T)$, while p_{j_a} leads to L_j.[2] For each pair of points [L_{j-1},H_j], we further define the j-th *bounding box*, which will help to bound the path length $db(H_j, L_{j-1})$ along the boundary back from H_j to L_{j-1}. The adversary will force the robot to walk outside the bounding boxes as much as possible, since any distance walked inside each bounding box can be bounded with Lemma 4.1. In our strategy one of the two paths (p_{j_b}) is maintained inside the j-th *bounding box*.

Lemma 4.5: At least one of the two paths p_{j_a} and p_{j_b} lies inside the j-th *bounding box* until the robot reaches the next *leaving* point L_j that is closer to the target T or it reaches T.

Lemma 4.6: The robot walks a maximum distance in the scene from S to T if for every pair $[H_j, L_j]$ the path p_{j_a} connecting H_j and L_j along the obstacle boundary lies completely outside the j-th *local bounding box*. Moreover, the other path p_{j_b} from H_j passes through all previously defined leaving and hit points before AL_j can be reached.

[2]The scene is designed by the *adversary* in a such way that the robot, conducting a spiral search, will switch to the opposite direction just before reaching AL_j or T and will always choose L_j as a leaving point.

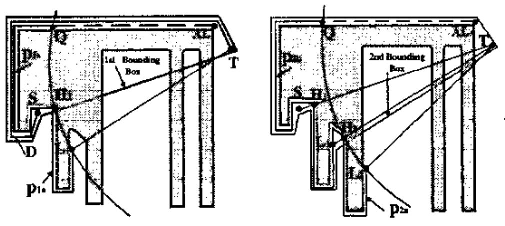

Fig. 4. Illustration for the definitions of the CautiousBug Algorithm

Thus, in the worst-case scenario the robot will travel along the single obstacle boundary, traversing the same parts of the boundary many times. Now we inductively define the *bounding boxes* and find the upper bound on the paths from hit point H_j to the next leaving point L_j along the boundary. Starting at point S (L_0), the robot walks on the straight segment [S,T] toward T until point H_1 is reached and the mode switches to boundary-following, starting the spiral search for L_1.

Lemma 4.7: Path p_{1_b} lies entirely inside the figure with the boundary formed by optimal path D and the straight segment [S,T]. The maximal path length of p_{1_b} is less than d^2.

We call this figure the 1-*st bounding box*. Lemma 4.1 and 4.2 determine an upper bound d^2 on the path length p_{1_b}. As soon as the robot comes closer to T than the current minimal distance to the target $d(H_1, T)$, d_{min} will be updated or even a better leaving point AL_1 will be defined. Since our *advisory* will try to escape this situation as long as possible, the actual bound on the maximal path length of p_{1_b} is even less then d^2. Consider hit point H_1 and leaving point L_1. The proof of the following lemma follows from Theorem 2.1 and Lemma 2.1.

Lemma 4.8: The adversary can place L_1 at a traveling distance $p_{1_a} \leq 2p_{1_b}$ from H_1 such that the robot reaches L_1 just before it reaches AL_1 or T. The robot traverses a distance $W_1 \leq 9p_{1_b}$ along the obstacle boundary before it reaches L_1 from H_1. Here, p_{1_b} is a part of the followed obstacle boundary completely inside the 1-*st bounding box*.

Consider hit point H_j and leaving point L_j. Then, the j-th *bounding box* is defined as following:

Lemma 4.9: Path p_{j_b} lies entirely inside the figure with boundary, formed by optimal path D, parts of obstacle boundary $[L_1, H_1]$, $[L_2, H_2]$, ... $[L_{j-1}, H_{j-1}]$, straight segments $[L_0, H_1]$, $[L_1, H_2]$, ... $[L_{j-1}, H_j]$ and $[H_j, T]$. Both the maximal path length of p_{j_b} and the maximal path length of the spiral search, W_j, have an order $O(d^{2^j})$.

We give a sketch of the inductive proof. From Lemmas 4.7 and 4.8, $p_{1_b} = O(d^2)$ and $W_1 = O(d^2)$. Note that, while executing the spiral search at H_2, p_{2_b} is restricted to stay inside the figure with boundary $d + d(S, H_1) + db(H_1, L_1) + d(L_1, H_2) + d(H_2, T) < d + ST + d^2 = O(d^2)$. Thus, $p_{2_b} = O(d^4)$ and $W_2 = O(d^4)$. We assume that the lemma holds for $p_{(j-1)_b}$ and W_{j-1}. The boundary of the j-th *bounding box* is formed by

$d + d(S,H_1) + db(H_1,L_1) + d(L_1,H_2) + db(H_2,L_2) + ... + db(H_{j-1},L_{j-1}) + d(L_{j-1},H_j) + d(H_j,T) \leq d + ST + p_{1_a} + p_{2_a} + ... + p_{(j-1)_a} = O(d) + O(d^2) + ... + O(d^{2^{j-1}}) = O(d^{2^{j-1}})$. By Lemma 4.1 and 4.2, the maximal path length of p_{j_b} inside the j-th *bounding box* has the order of $O(d^{2^j})$. Applying Theorem 2.1 and Lemma 2.1, $W_j = O(d^{2^j})$.

Lemma 4.10: Starting from S, the total distance walked by the robot before T is reached has an order of $O(d^{2^k})$, where $k = \#Min - 1$. $\#Min$ is the total number of isolated local minima points of the distance function in the scene.

Proof: We distinguish three components of the path: straight-to-target motion on the straight segments $[L_{j-1}, H_j]$, sliding along the obstacle boundary from H_j to D_j and boundary-following from H_j to L_j.

Moving in the free space between obstacles: The summarized path length over all straight segments is at most ST, i.e. $O(d)$.

Sliding on the obstacle boundaries: The distance to the target d(x,T) decreases along sliding segment $[H_j, D_j]$ with regard to d(H_j, T). D_j represents a *departure point* – the point where the robot leaves the obstacles, switching from sliding along the blocking obstacles' boundary to moving in the free space between obstacles[8]. The endpoint of each segment H_{j+1} is closer to T than the starting point of the same segment D_j, which is closer to T than the previous hit point H_j. In the scene with maximal sliding the robot has to move by a spiral starting from S and ending at T. Considering our restrictions on the environment, the spiral can be approximated with ST/2 distorted circles with a small perforation in each. The outer circle's boundary length is bounded with $2\pi \cdot ST$. The shortest path goes through the perforations in the circles: $SlidingPath \leq 2\pi \cdot ST \cdot ST/2 \leq \pi \cdot d^2 = O(d^2)$

Boundary following: The distance walked along the obstacles' boundaries is $\sum_{i=1}^{k} W_i$, where $k = \#Min - 1$. $\#Min$ is the total number of the isolated local minimum points of the distance function in the scene. The last minimum point of the distance function is the target. Thus, $\sum_{i=1}^{k} W_i = \sum_{i=1}^{k} O(d^{2^i}) = O(d^{2^k})$.

Hence, the total distance walked by the robot to reach the target has an order of $O(d^{2^k})$.

Lemma 4.11: The competitive ratio of $CautiousBug$ has an order of $O(d^m)$, where $m = 2^{\#Min-1} - 1$.

The worst-case construction of an environment by an *adversary* is shown in Fig. 4. For example, after a short walk along the straight segment $[S, H_1]$, the robot executes the spiral search with H_1 as the origin. The adversary adjusts the length p_{1_a} such that the robot reaches L_1 just before it reaches obstacle boundary point Q, where $d(Q,T) \leq d(H_1,T)$. The adversary designs the other parts of the obstacle in a similar way. Our proof does not consider the sensors' range, since our *adversary* can rebuild the environment so that there will be no significant benefit from the range sensor. However, in a real-world scenario, range sensor will only decrease the path length.

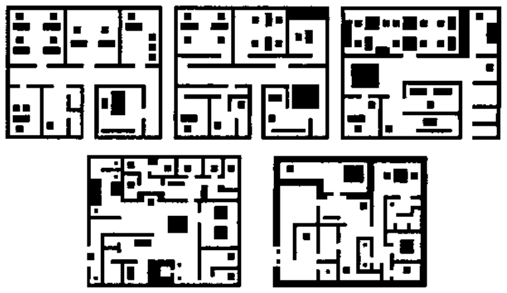
Fig. 5. The office-like simulated environment

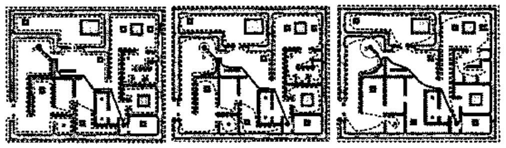
Fig. 6. The simulation of the $TangentBug$ algorithm (dashed line) and the $CautiousBug$ algorithm (solid line) in scene 5. The circle around Start shows the range of the sensor. (a) contact sensors: Competitive ratio of $TangentBug$ is 17.55 vis. 3.47 for $CautiousBug$ (b) sensors range R = 25: 16.16 vis. 1.53, correspondingly (c) sensors range R = 50: 12.33 vis. 1.51, correspondingly

V. Simulation results

Simulations were performed to study the average competitive ratio of $CautiousBug$ and $TangentBug$ algorithms and their dependence on the sensor range R. As an algorithm against which to test our $CautiousBug$, we chose $TangentBug$, the best known algorithm within the same minimal memory requirements group. To compare the efficiency of different algorithms, Nogami et al. in [17] chose all possible start and target points in the scene and used the average path lengths. Unfortunately, this reasonable and fair criterion is limited by the hardware resources since in the large and relatively complicated scenario the number of (S,T)-pairs becomes too big. In this paper we propose to use the pure **efficiency** criterion of the algorithm - the competitive ratio, which can be reasonably used to estimate the efficiency of even a non-competitive algorithm.

The algorithms were tested in office-like environments, consisting of one large concave obstacle, simulating the outer walls, and small concave and convex obstacles, simulating inner walls and office furniture space(Fig.5). In five scenes of size 800 $\times$ 700 pixels, all pairs of start and target points were chosen on the 5-pixel grid in the free space. Then a random number generator, initialized with the current PC-time, chose, from among these 620,000-760,000 combinations, 0.03-0.05 % of (S,T)-pairs, which were used for the simulations. Only small realistic range sensors were used, which covered at most 1.4 % of the scene.

The results are summarizeded in Table I. The average competitive ratio of $CautiousBug$ was smaller than that of non-competitive $TangentBug$ with all sensor ranges in all scenes. Increasing the maximal range of the sensors

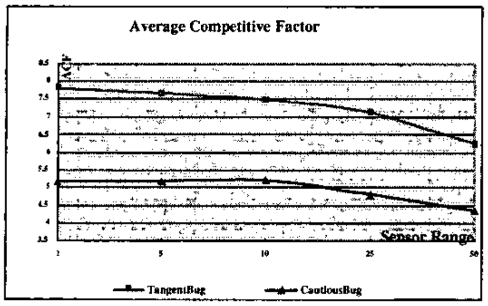

Fig. 7. The average competitive factors of the *TangentBug* and *CautiousBug* algorithms

improved the performance of both algorithms. In almost all scenes *CautiousBug*'s average competitive ratio with a contact sensor is even less than the competitive ratio of *TangentBug* with the 50-pixel range sensors!

TABLE I

CautiousBug VS. *TangentBug* – AVERAGE COMPETITIVE FACTOR

	Scene1		Scene2		Scene3		Scene4		Scene5	
R	TB	CB	TB	CB	TB	CB	TB	CB	TB	CB
1	6.9	4.6	8.5	5.4	7.1	5.3	9.6	5.3	6.7	4.9
5	6.8	4.9	8.1	5.1	7.2	5.4	9.4	5.7	6.4	4.6
10	6.7	5.4	7.7	5.1	7.2	5.2	9.5	5.7	6.1	4.7
25	6.3	4.4	7.3	5.0	6.9	5.2	9.3	4.8	5.5	4.5
50	4.4	4.0	7.0	4.4	6.9	4.4	7.4	4.4	4.9	4.2

As the sensor range increases, the edges of the global tangent graph become the edges of the LTG, and the robot has a higher probability to move along the globally shortest path. However, the incomplete knowledge of the robot leads to local decisions under the *TangentBug* algorithm, which may be different from the globally optimal ones. At the same time *CautiousBug* does not risk choosing a "wrong" boundary-following direction, but checks both directions. In some cases this cautious behavior incurs a lot of additional path being walked, but in other cases it saves the robot from traveling along the entire boundary of the followed obstacle. This behavior is more efficient on average (Fig.7).

VI. Discussion

We presented *CautiousBug*, a new range-sensor-based competitive algorithm for mobile robots. We united the idea of the locally shortest path, using the LTG, and the best strategy for the search of the goal in alternate directions. The LTG is utilized for planning the most optimal path with the local information available. The *Spiral Search* fixes the problem of choosing a wrong boundary-following direction when switching to boundary-following mode. Thus, *CautiousBug*, combining the properties of *TangentBug* and *Spiral Search*, becomes a competitive algorithm. In the worst-case scene, the competitive factor of *CautiousBug* has an order of $O(d^m - 1)$, where d is the length of the optimal path from the starting point S to the target point T, $m = 2^{\#Min-1}$ and $\#Min$ denote the number of the distance function local minimum points in the environment. The simulations showed that the average competitive factor of *CautiousBug* is significantly less than the one we succeeded in proving.

Acknowledgment

The authors would like to thank Professor Shin'ichi Yuta and all members of the Roboken Laboratory, University of Tsukuba, Japan, where a part of this research was conducted under the support of AIEJ. The authors want to express special thanks to Professor Elon Rimon for his encouragement and helpful comments.

References

[1] R. Baeza-Yates, J. Culberson, G. Rawlins. Searching in the Plane. Information and Computation, 106(2):234-252, 1993

[2] A. Blum, P. Raghavan, B. Schieber. Navigating in Unfamillar Geometric Terrain. SIAM Journal on Computing, 26(1):110-137, 1997.

[3] A. Datta, S. Soundaralakshmi. Motion Planning in an Unknown Polygonal Environment with Bounded Performance Guarantee. In Proc. IEEE ICRA'99, pp. 1032-1037.

[4] Y. Horiuchi and H. Noborio. Evaluation of Path Length Made in Sensor-based Path-planning with the Alternative Following. In Proc. IEEE ICRA'01, pp. 909-916.

[5] C. Icking, R. Klein. Competitive Strategies for Autonomous Systems. Modelling and Planning for Sensor Based Intelligent Robot Systems, pp. 23-40, 1995.

[6] C. Icking, T. Kamphans, R. Klein, E. Langetepe. On the Competitive Complexity of Navigation Tasks. Sensor Based Intelligent Robots, pp. 245-258, 2002.

[7] I. Kamon. Locally Optimal Sensor Based Robot Navigation. Ph.D. Thesis. Technion, 1997.

[8] I. Kamon, E. Rivlin, E. Rimon. A New Range-sensor Based Globally Convergent Navigation Algorithm for Mobile Robots. In Proc. IEEE ICRA'96.

[9] I. Kamon, E. Rivlin. Sensory Based Motion Planning with Global Proofs. IROS'95, pp. 435-440.

[10] J. C. Latombe. Robot Motion Planning. Kluwer Academic Publishers, USA, 1991.

[11] V. J. Lumelsky, T. Skewis. Incorporating Range Sensing in the Robot Navigation Function. In IEEE Trans. Sys. Man. Cybernet., 20(5):1058-1068, 1990.

[12] V. J. Lumelsky, A. A. Stepanov. Path Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape. Algorithmica, 2:403-430, 1987.

[13] V. J. Lumelsky, S. Tiwari. An Algorithm for Maze Searching with Azimuth Input. In Proc. IEEE ICRA'94, pp. 111-116.

[14] H. Noborio. A Path-planning Algorithm for Generation of an Intuitively Reasonable Path in an Uncertain 2D Workspace. Proc. of the Japan-USA Symposium on Flexible Automation, 2:477-480, 1990.

[15] H. Noborio. A Sufficient Condition for Designing a Family of Sensor Based Deadlock Free Path Planning Algorithms. Advanced Robotics, 7(5):413-433, 1993.

[16] H. Noborio, T. Yoshioka, S. Tominaga. On the Sensor-based Navigation by Changing a Direction to Follow an Encountered Obstacle. In IROS'97, pp. 510-517.

[17] R. Nogami, S. Hirao, H. Noborio. On the Average Path Lengths of Typical Sensor-based Path-planning Algorithms by Uncertain Random Mazes. CIRA'03, pp.471-478.

[18] C. H. Papadimitriou, M. Yannakakis. Shortest Path Without a Map. Theoretical Computer Science, 84(1):127-150, 1991.

[19] A. Sankaranarayanan, M. Vidyasagar. Path Planning Algorithm for a Moving Point Object Amidst Unknown Obstacles in a Plane: the Universal Lower Bound on Worst Case Path Lengthes and a Classification of Algorithms. In Proc. IEEE ICRA'91, pp. 1734-1741.