[9] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 500–505, 1985.

[10] J. J. Leonard and H. F. Durrant-Whyte. *Directed sonar sensing for mobile robots navigation*. Kluwer academic publishers, Boston, London, Dordrecht, 1992.

[11] Y. H. Liu and S. Arimoto. Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. *International Journal of Robotic Research*, 11(4):376–382, 1992.

[12] V. J. Lumelsky and T. Skewis. Incorporating range sensing in the robot navigation function. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(5):1058–1068, 1990.

[13] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst obstacles of arbitrary shape. *Algoritmica*, 2:403–430, 1987.

[14] V. J. Lumelsky and S. Tiwari. An algorithm for maze searching with azimuth input. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 111–116, 1994.

[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 and T. Yoshioka. An on-line and deadlock-free path-planning algorithm based on world topology. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems, IROS*, pages 1425–1430, 1993.

[17] P. Reignier. Molusc: an incremental approach of fuzzy learning. In *Proceedings of the International symposium on Intelligent Robotics Systems*, pages 178–186, 1994.

[18] W. D. Rencken. Concurrent localization and map building for mobile robots using ultrasonic sensors. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems, IROS*, pages 2192–2197, 1993.

[19] E. Rimon and J. F. Canny. Construction of c-space raodmaps from local sensory data. what should the sensors look for ? In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 117–123, 1994.

[20] A. Sankaranarayanan and M. Vidyasagar. A new path planning algorithm for a point object moving amidst unknown obstacles in a plane. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1930–1936, 1990.

[21] A. Stentz. Optimal and efficient path planning for partially known environments. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 3310–3317, 1994.

[22] A. Zelinsky. Using path transforms to guide the search for findpath in 2d. *The International Journal of Robotic Research*, 13(4):315–325, 1994.

is by modifying the expected path length from LTG nodes to the target.

The expected path length from LTG nodes to the target can be estimated by applying additional reasoning on the LTG, by using a global model of the environment, or by using more informative sensing modalities. Additional reasoning may distinguish between nodes which result from obstruction or from the sensor range limit, and nodes which are tangent points on real obstacles. The path length to the target from the first type of nodes is expected to be longer than the direct distance from such a node to the target. When a global model of the environment is available, it can be used to estimate the path length from the LTG nodes to the target, and thus affect the choice of locally optimal direction while moving towards the target, and the choice of boundary following direction. More informative sensing modalities, i.e. vision, can provide better estimations of the expected path length from LTG nodes to the target. For example, if the LTG node is recognized as a door opening it should have a shorter expected path length than a node which is recognized as a cupboard edge.

References

[1] R. C. Arkin. Motor schema based navigation for a mobile robot: an approach for programming by behavior. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 264–271, 1987.

[2] R. Bauer, W. Feiten, and G. Lawitzky. Steer angle field: an approach to robust maneuvering in cluttered, unknown environments. *Robotics and Autonomous Systems*, 12:209–212, 1994.

[3] J. Borenstein and Y. Koren. Real time obstacle avoidance for fast mobile robots in cluttered environments. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 572–577, 1990.

[4] H. Choset and J. W. Burdick. Sensor based planning, part ii: Incremental construction of the generalized voronoi graph. In *IEEE International Conference on Robotics and Automation*, Nagoya, Japan, May 1995.

[5] James L. Crowley and Yves Demazeau. Principles and techniques for sensor data fusion. *Signal Processing*, 32:5–27, 1993.

[6] G. Foux, M. Heymann, and A. Bruckstein. Two dimensional robot navigation among unknown stationary polygonal obstacles. *IEEE Transactions on Robotics and Automation*, 9(1):96–102, 1993.

[7] S. G. Goodridge and R. C. Luo. Fuzzy behavior fusion for reactive control of an autonomous mobile robot: Marge. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1622–1627, 1994.

[8] V. Guillemin and A. Pollack. *Differential Topology*. Prentice-Hall, Inc., New Jersey, 1974.

which result from obstruction, because the obstacle boundary left to O_1 is not visible to the robot, and thus expected path length to the target through O_1 is shorter than the expected path length to the target through O_2. When the robot reaches P_2, the node O_1 in the LTG is replaced by O_3, and the robot moves towards O_3. Only when the robot reaches P_3 it realizes that it is not possible to reach the target bypassing O_3 from the left, and moves towards O_1, from which it proceed to the target.

6 Discussion

We presented $TangentBug$, a new range-sensor based globally convergent navigation algorithm for mobile robots. We incorporated the idea of the locally shortest path, using the tangent graph, into the Bug paradigm, which guarantees reaching the target without building a global world model. We adjusted the structure of the tangent graph, which was defined for a completely known environment, and introduced a local range-data based version of it, termed the *local tangent graph*, or LTG. We re-formulated the basic behaviors of the Bug family, and defined new transition conditions for switching between them.

The $TangentBug$ algorithm uses the basic behaviors of motion towards the target and obstacle boundary following. Let $d(x,T)$ be the distance of the robot, located at a point x, from the target T. Then during the motion towards the target, $d(x,T)$ decreases monotonically. During the boundary following, the robot attempts to escape from a local minimum of $d(x,T)$. While moving towards the target, the robot chooses a *locally optimal direction*, which is the direction along the shortest path to the target according to the current LTG. The motion towards the target terminates when the robot detects that moving in the locally optimal direction would drive it into a local minimum. The obstacle boundary following is then invoked to drive the robot away from the local minimum. The robot first chooses a boundary following direction, then it moves along the boundary while continuously monitoring the LTG. The robot uses the LTG to make shortcuts along the obstacle boundary, but it may not leave the boundary before the following *leaving condition* is met. Let $d_{\min}(T)$ be the minimal distance of the robot from T observed along the path so far. Then the robot leaves the obstacle boundary when the shortest path to the target on the current LTG is guaranteed to achieve a distance $d(x,T)$ which is smaller than $d_{\min}(T)$.

The $TangentBug$ algorithm uses range data for choosing the locally optimal direction while moving towards the target, for choosing the boundary following direction when boundary following behavior is initiated, and for the leaving condition. The simulation results showed improvement (from 0.71 to 0.28 in the complex environment) in the path length as the sensor range increases. The results indicate a significant advantage of $TangentBug$ relative to the algorithm $VisBug$ from [12], in all the tested scenarios.

The $TangentBug$ algorithm is simple for implementation because it is purely reactive and does not require exact positioning along the path [6]. However, additional information can be easily incorporated to improve performance in practical scenarios. The LTG can be improved when accumulated sensory data, e.g. using occupancy grids, is used to increase both sensor reliability and the effective sensor range. Another way to improve performance

[6] Exact positioning is necessary only for detection of target reaching and completion of a loop around an obstacle

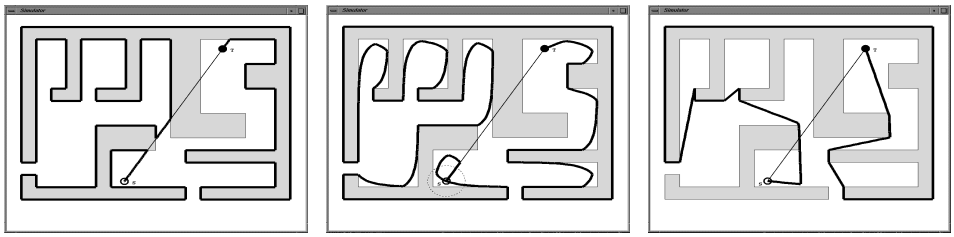


Figure 14: Simulation results of $VisBug$ in "world2" environment. Left - using contact sensors (path length 1.00). Middle - using limited range sensor 50 (path length 0.89). Right - using unlimited range sensor (path length 0.48).

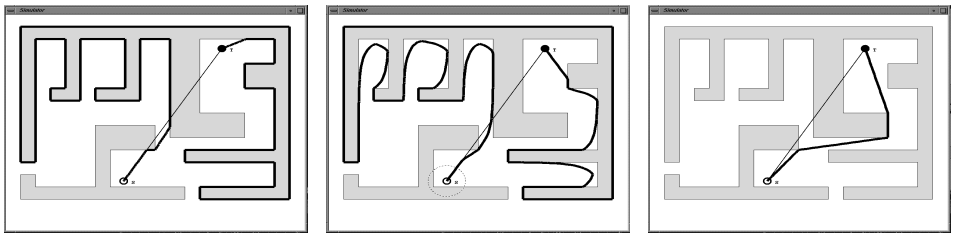


Figure 15: Simulation results of $TangentBug$ in "world2" environment. Left - using contact sensors (path length 0.79 relative to $Visbug$); Note that the boundary following direction is chosen based on local information. Middle - using limited range sensor 50 (path length 0.70). Right - using unlimited range sensor; The path length is 0.09 relative to $Visbug$ with contact sensors. The path was changed significantly, relative to the previous examples, using the additional range data.

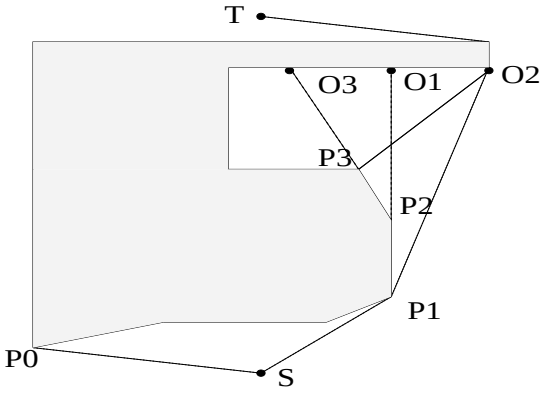


Figure 16: The incomplete knowledge of the robot leads to local decisions which may be different from the globally optimal ones. From P_1 the robot moves towards O_1, because the expected path length to the target through O_1 is shorter than through O_2. From P_2 the robot moves towards O_3, because of the same reason. Only when the robot reaches P_3 it realizes that it is not possible to reach the target bypassing O_3 from the left, and moves towards O_1, from which it proceed to the target.

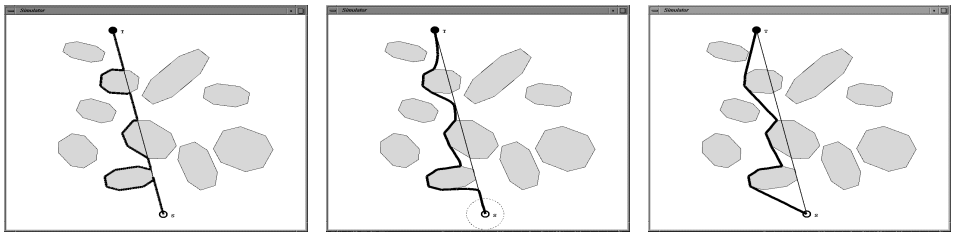


Figure 12: Simulation results of $VisBug$ in "world1" environment. Left - using contact sensors (path length 1.00). Middle - using limited range sensor 50 (path length 0.82). Right - using unlimited range sensor (path length 0.76).

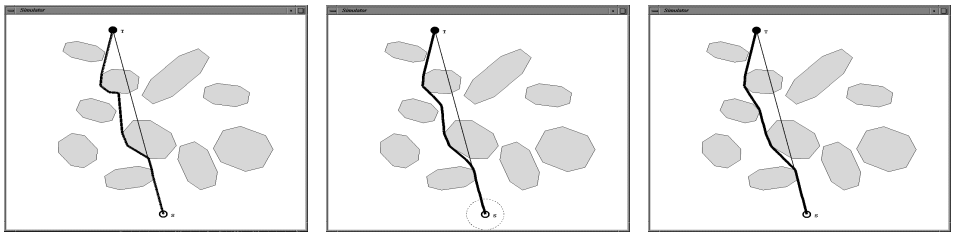


Figure 13: Simulation results of $TangentBug$ in "world1" environment. Left - using contact sensors (path length 0.63 relative to $Visbug$). Middle - using limited range sensor 50 (path length 0.61). Right - using unlimited range sensor (path length 0.60).

cases, implying that range data was used only for choosing the locally optimal direction. However, using this information decreased path length relative to $VisBug$. The minor improvement in the path length as the sensor range increases, from 0.77 to 0.73, suggests that in simple environments small sensor range is sufficient, because the local data usually leads to the globally correct decisions regarding the detour direction left/right.

In "world2", the effect of the range sensors is more apparent. Significant local shortcuts can be performed by scanning the boundaries of concave obstacles, using the range sensors, instead of actually following them. This advantage is used by both algorithms. The ability to choose the boundary following direction, based on local information, and the ability to leave an obstacle boundary before the line $[Start, Target]$ is visible, are the main advantages of $TangentBug$ in this environment. It is interesting to note that local range data is informative for choosing the boundary following direction even in this relatively complex environment.

dv

As the sensor range increases, edges of the global tangent graph become edges of the LTG, and the robot has a higher probability to move along the the globally shortest path. However, the incomplete knowledge of the robot leads to local decisions which may be different from the globally optimal ones. Our modeling approach considers the observed obstacles as thin walls which are the only obstacles in the environment, thus obstructed obstacles are not taken into consideration. The underestimation of the obstacles leads to non-optimal decisions, as shown in the example in figure 16. In this example, we consider a robot with unlimited sensor range. Starting from S, the robot moves to P_1 on an edge of the global tangent graph. From P_1, the robot moves towards the node O_1 of the LTG,

	world1		world2	
R	$VisBug$	$TangentBug$	$VisBug$	$TangentBug$
0	1.00	0.77	1.00	0.71
10	0.99	0.77	0.99	0.70
50	0.88	0.74	0.87	0.64
250	0.81	0.73	0.49	0.29
∞	0.81	0.73	0.49	0.28

Table 1: The performance of $TangentBug$ algorithm compared to $VisBug21$ algorithm. The algorithms were tested in two simulated environments. Five maximal sensor range values, ranging from 0 to ∞, were tested in each environment. The average path length, measured over 100 runs in each scenario, is presented relative to $VisBug21$ performance with contact sensors.

5 Simulation results

Simulations were performed to study the dependence of the resulting paths on the sensor range R. The simulations show that $TangentBug$ often generates paths that approach the shortest path as R increases. The simulations also compare $TangentBug$ with the classical $VisBug$ algorithm, showing that $TangentBug$ generates significantly shorter paths in congested office-like scenarios.

The algorithm was tested in two simulated environments. The simple environment "world1" consists of convex non-intersecting obstacles, while the complex one "world2" consists of concave obstacles with "office-like" shape. One hundred start/target points were used for in each environment. In "world1", the y coordinates were fixed [4] and the x coordinates were chosen randomly within a given range. In "world2", the start/target points were chosen randomly within the freespace.

We used the algorithm $VisBug21$ from [12] for comparison. This algorithm plans local shortcuts, based on range data [5], relative to the path that would be planned by algorithm $Bug2$ from [13] . Under $Bug2$, the robot moves directly towards the target until hitting an obstacle. It then follows the obstacle boundary, using a predefined direction (clockwise), until reaching the line $[Start, Target]$. The robot then leaves the obstacle boundary and moves directly towards the target again.

The results are summed in table 1. The paths produced by $TangentBug$ were shorter in all the scenarios. Increasing the maximal range of the sensors improved the performance of both algorithms. However, $TangentBug$ makes better use of the range data. Range data is used for choosing the locally optimal direction while moving towards the target, for choosing the boundary following direction when boundary following behavior is initiated, and for the leaving condition, which is not restricted to the line $[Start, Target]$ as in $VisBug$.

In "world1", the paths produced by $TangentBug$, using unlimited range sensors, were optimal in most cases. Only the moving-towards-the-target behavior was used in most

[4] We used the upper and lower sides of the environment in order to produce longer paths, so that the effect of the range sensors was more apparent.

[5] Using contact sensors, $VisBug21$ performance is identical to $Bug2$.

The total path length of the first type subsegments is bounded by $\frac{\|S,T\|}{Tresh}$, because the distance to the target decreases along them with the minimal rate $Tresh$.

Obviously, all the subsegments of the second type, in which $\frac{d}{ds}d(P(s),T) \geq -Tresh$, are *sliding* subsegments, in which the robot motion is influenced by blocking obstacles. To bound the path length of these subsegments we first show that the path length traversed by the focus point, which is the LTG node towards which the robot is moving, is bounded by $\sum_i P_i$. We then show that the number of these subsegments can be bounded by some constant K, and use Lemma 4.9 to argue that the addition to the path length of the robot in each subsegment, relative to the path length of the focus point, is bounded by R.

The focus point F can not traverse the same part of the boundary twice during a single sliding subsegment. Lemma 4.9 state this fact for the case when the detour direction and the blocking obstacle are fixed. The distance $d(F,T)$ strictly decreases either when the detour direction is changed, because this is explicitly required by the detour constrain, or when the blocking obstacle is changed, as shown in Lemma 4.8, thus the focus point can not traverse twice parts of the boundary which were traversed in previous subsegments.

We next show that the distance from the focus point to the target $d(F,T)$ decreases between moving-towards-the-target segments. When the robot switches into boundary-following behavior, in a switch point Sw_i, it registers a minimum point M_i which is a local minimum of $d(x,T)$. The focus point F_1 in Sw_i satisfies $d(F_1,T) \geq d(M_i,T)$. When the robot switches into moving-towards-the-target in leave point L_i, the leaving condition holds and therefore its focus point F_2 satisfies $d(M_i,T) > d(F_2,T)$. It follows that $d(F_1,T) > d(F_2,T)$, therefore the distance from the focus point to the target $d(F,T)$ decreases between moving-towards-the-target segments. It follows that the focus point F can not traverse the same part of the boundary twice during all moving-towards-the-target segments, and its path length can be bounded by the perimeter of the obstacles it touches $\sum_i P_i$.

We next show that the number of subsegments on which $\frac{d}{ds}d(P(s),T) \geq -Tresh$ can be bounded by a constant K. After the detour direction becomes fixed, it can be changed only when the blocking obstacle is changed or the moving-towards-the-target behavior is terminated. The blocking obstacle can be changed only in a finite number of tangent points on each obstacle i, $\sharp Tangent_i$, The moving-towards-the-target behavior can be terminated only in a finite number of local minima points on each obstacle i, $\sharp Minima_i$. We therefore have $K = \sum_i(\sharp Tangent_i + \sharp Minima_i)$.

In Lemma 4.9 we show that the addition to the path length of the robot in each subsegment with fixed detour direction, relative to the path length of the focus point, is bounded by $R\,W \leq W_f + R$. Therefore the total path length of subsegments on which $\frac{d}{ds}d(P(s),T) \geq -Tresh$ can be bounded by $\sum_i P_i + R \times \sum_i(\sharp Tangent_i + \sharp Minima_i)$.

□

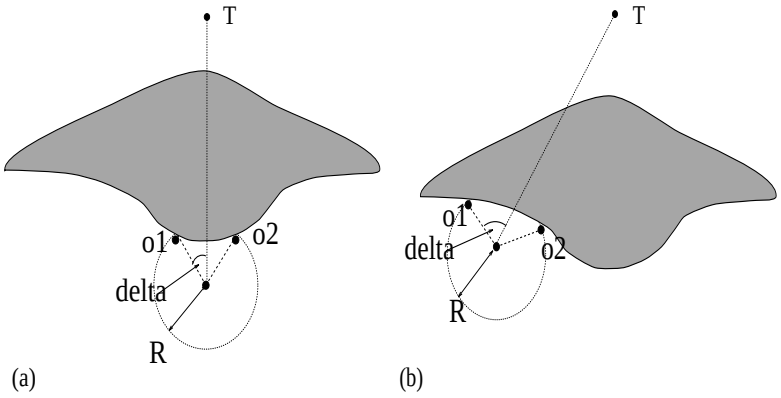


Figure 11: (a) Small detour angle implies that the distance to the target decreases rapidly. (b) Big detour angle implies that the distance to the target decreases slowly. When $delta \geq \Delta$ the detour direction becomes fixed.

To bound the length of the path produced by the algorithm, using range sensors, we introduce an additional constrain on changing the detour direction while moving-towards-the-target. The detour direction is allowed to change only as long as the robot's distance to the target decreases "fast enough"; the robot path can be defined as a parametric curve $P(s)$; the rate of decreasing the distance from the target is $\frac{d}{ds}d(P(s),T)$; the detour direction is allowed to change when $\frac{d}{ds}d(P(s),T) < -Tresh$, for some threshold $0 < Tresh \leq 1$. This constrain can be defined in terms of the detour angle δ: $\frac{d}{ds}d(P(s),T) < -Tresh \Leftrightarrow abs(\delta) < \Delta$, where $\Delta = arccos(Tresh)$ (see figure 11). When $abs(\delta) \geq \Delta$ the detour direction becomes fixed until the blocking obstacle is changed or the moving-towards-the-target behavior is terminated.

Proposition 4.14 *The Upper bound W_{max} for the path length produced by the algorithm $TangentBug$, using sensors with maximal range R, is:*

$$W_{max} = \frac{\|S,T\|}{Tresh} + \sum_i P_i + R \times K + \sum_i (P_i + R) \times \sharp Minima_i$$

where $\sum_i P_i$ refers to the perimeter of the obstacles intersecting the disc of radius $\|S,T\|$ centered at T, K is a constant, and $\sharp Minima_i$ is the number of local minima of $d(x,T)$ on the boundary of obstacle i.

Proof: The first three elements of the sum bound the path length of moving-towards-the-target segments, while the fourth element bounds the path length of boundary-following segments. The length of each boundary-following segment is bounded by $P_i + R$, as explained in corollary 4.11. The total length of these segments is therefore $\sum_i (P_i + R) \times \sharp Minima_i$, as explained in proposition 4.7. We now explain the first three elements of the upper bound.

The moving-towards-the-target segments can be divided into two types of subsegments: subsegments on which $\frac{d}{ds}d(P(s),T) < -Tresh$, and subsegments on which $\frac{d}{ds}d(P(s),T) \geq -Tresh$.

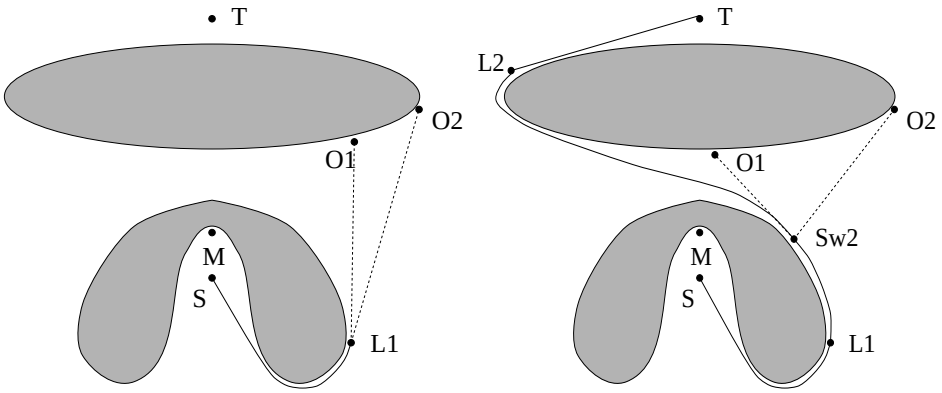


Figure 10: (a) The range based leaving condition enables the robot to leave an obstacle boundary from the leaving point L_1 s.t. $d(L_1) > d_{followed}(T) = d(M)$. The robot leaves the boundary because $d_{reach}(T) < d_{followed}(T)$, where $d_{reach}(T) = d(O_1, T)$. (b) The robot moves towards the node $O_1 \in$ LTG using moving-towards-the-target behavior until Sw_2, where O_1 reaches a local minimum. The robot then switches to boundary following and moves until L_2, where it leaves the obstacle and moves directly towards the target.

To insure that $d(M_j, T) \leq d_{reach}(T)$, and thus $d(M_j, T) < d(M_i, T)$, we want to guarantee that $d(M, T) \leq d_{reach}(T)$. During moving-towards-the-target behavior the point M is one of the endpoints of the blocking obstacle, because otherwise it is a local minimum and the behavior is terminated [3]. To guarantees that $d(M, T) \leq d_{reach}(T)$ during the motion towards the target, we use an additional constrain on the direction choosing mechanism. A node $v \in$ LTG is considered as a candidate for the locally optimal direction only if $d(v, T) \leq d_{reach}(T)$. The distance $d(v, T)$ can only decrease while moving towards v, because a local minimum is detected when $d(v, T)$ increases, and the motion towards the target is terminated.

The additional constrain is activated by transferring the parameter $d_{NextMin}(T)$, set to $d_{NextMin}(T) \leftarrow d_{reach}(T)$, from the boundary-following behavior to the moving-towards-the-target behavior. The constrain is used as long as the robot's distance from the target satisfies $d(x, T) > d_{reach}(T)$. When $d(x, T) \leq d_{reach}(T)$ the constrain is not necessary anymore because it is guaranteed that $d(M_j, T) \leq d(x, T)$.

□

Corollary 4.13 *Using range sensors, the algorithm reaches the target T in a finite path if it is reachable from S. Otherwise the algorithm terminates after a finite path.*

Proof: The proof follows the lines of Theorem 1 and Theorem 2 for the contact sensors case.

□

[3] Note that the leaving condition from obstacle O_F may hold in L_i, where the closest point M on the blocking obstacle O_B is a local minimum. The robot will immediately swith to boundary following $Sw_{i+1} = L_i$, where the new followed obstacle is O_B. This scenario may happen on non-smooth obstacles.

along the subsegment is towards F. The distance from the robot to F along the subsegment is not bigger than the sensor range $d(x, F) \leq R$, and this distance is non-increasing because F is not faster than the robot. It follows that the path length W of the robot is bounded by $W \leq W_f + R$.

□

Lemma 4.10 *The path length of a moving-towards-the-target segment, using range sensors, is finite.*

Proof: We consider a moving-towards-the-target segment starting from L_i and ending in M_j. The total length of *direct* subsegments is bounded by the distance from the starting point to the target $\|L_i, T\|$, as shown in Lemma 4.3.

The *sliding* subsegments are divided into separate parts, in which the blocking obstacle and the detour direction are fixed. The number of these parts is finite because the detour direction can be changed only a finite number of times, because of the detour constrain, and the blocking obstacle can be changed only in a finite number of point on each obstacle. The path length of each part is finite, as shown in Lemma 4.9, therefore the total path length of the sliding subsegments is finite.

□

Corollary 4.11 *The path length of a boundary-following segment, using range sensors, is finite.*

Proof: If the starting point Sw_i of the boundary-following segment is on an obstacle boundary, the path planned using the LTG is bounded by the perimeter of the obstacle. It may be shorter because it may contain local shortcuts relative to the obstacle boundary. If Sw_i is not on an obstacle boundary, its distance from the followed obstacle is not larger than the sensor range R. Therefore the bound for the path length is $R + P$, where P is the perimeter of the obstacle.

□

Corollary 4.12 *Using range sensors, the distance to the target decreases between successive minima points $d(M_i, T) > d(M_j, T)$ where $i < j$.*

Proof: Moving from M_i to M_j the robot must leave an obstacle in a leaving point L_i, in which the leaving condition holds $d_{reach}(T) < d_{followed}(T) \leq d(M_i, T)$. If a blocking obstacle is sensed from L_i then $d_{reach}(T)$ is the distance to the target from the closest point M on its boundary and $d(M, T) = d_{reach}(T)$. If no blocking obstacle is sensed from L_i then the robot moves from L_i directly towards the target until it senses a blocking obstacle (or reaches the target - which is not considered here). In this case the closest point to the target M on the blocking obstacle's boundary satisfies $d(M, T) < d_{reach}(T)$. The closest point to the target M is monitored during the moving-towards-the-target behavior, and it becomes the next minima point M_j when this behavior is terminated.

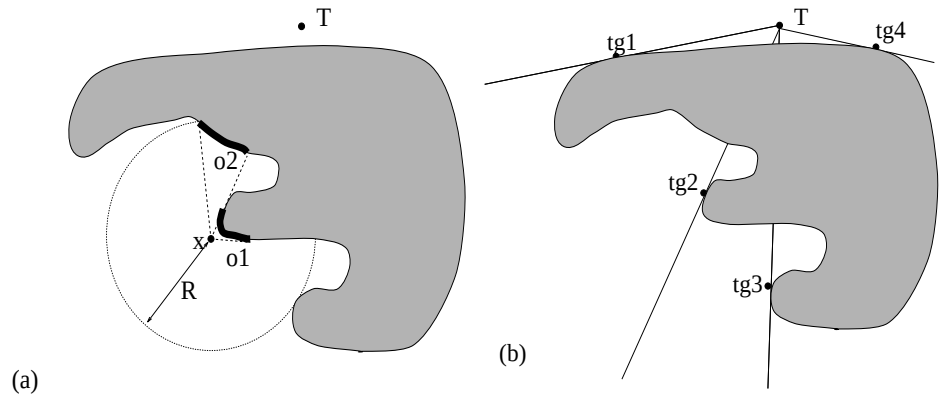


Figure 9: (a) A change in the blocking obstacle takes place when the robot is heading directly towards the target and there is a discontinuity of the distance in this direction. In the example the blocking obstacle $o1$ will be replaced by $o2$. (b) A change in the blocking obstacle may happen only in a finite number of tangent points $tg1, tg2, tg3, tg4$.

$d(O_2, T) - d(O_1, T) > Step$ for some constant $Step > 0$. The detour constrain guarantees that the detour direction will be changed only a finite number of times along the path.

Corollary 4.8 *The blocking obstacle may be changed, during moving-towards-the-target segment, only in a finite number of tangent points on each obstacle.*

Proof: A change in the blocking obstacle takes place when the robot is heading directly towards the target and there is a discontinuity of the distance in this direction (see figure 9). This situation may happen only when the focus point is located in a tangent point with a line that pass through the target. The number of these tangent point on each obstacle i is finite $\sharp Tangent_i$. When the blocking obstacle changes from O_1 to O_2, the focus point $F_2 \in O_2$ is closer to the target than the focus point $F_1 \in O_1$, $d(F_1, T) > d(F_2, T)$.

□

Lemma 4.9 *The path length of a sliding subsegment, along which the detour direction and the blocking obstacle are fixed, is finite.*

Proof: We consider a *sliding* subsegment, which is a part of a moving-towards-the-target segment, along which the detour direction and the blocking obstacle are fixed. The robot moves towards the focus point F which slides along the boundary of a single obstacle. We first show that the path length W_f traversed by the focus point is finite, and then show that the path length W traversed by the robot is bounded by $W \leq W_f + R$.

The sliding direction of F on the boundary is fixed. It can not complete a loop around the obstacle boundary, because the blocking obstacle is changed when the focus point reaches a tangent point with a line that pass through the target. It follows that the path length of F along the boundary is finite. The robot heading

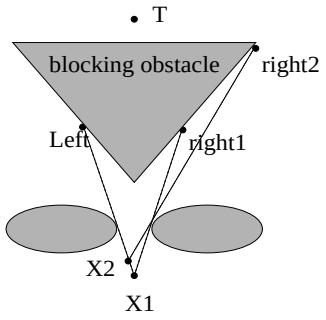


Figure 8: An unstable behavior may happen when the the robot motion changes the location of the sensed endpoints of the blocking obstacle. The detour direction in X_1 is left. Moving from X_1 to X_2 makes the right obstacle endpoint *right*2 closer to the target, and the detour direction will be changed from left to right.

sensors case, some distinguished points along the path. We show that the path length of each motion segment is finite, prove completeness of the algorithm for the range sensors case and derive an upper bound for the path length in this case.

In the following, we consider sensors with maximal range R, and use the notion of the *blocking obstacle*, which is the obstacle positioned between the robot location and the target, within the sensor range R. A hit point H_i is a point where the robot first senses the blocking obstacle; a departure point De_i is a point where the blocking obstacle changes during moving-towards-the-target behavior; a switch point Sw_i is a point where the robot switches from moving-towards-the-target to boundary-following; each switch point has a corresponding minimum point M_i, which is a local minimum of the function $d(x,T)$, visible from Sw_i within the sensor range R; a leave point L_i is a point where the the leaving condition holds; the robot leaves an obstacle boundary and switches to moving-towards-the-target. We also define the *focus point* F as the location of the LTG node towards which the robot is heading.

Using range sensors usually reduces path length. Moving from a hit point H_i, the robot moves in the locally optimal direction towards the focus point F, which is an endpoint of the blocking obstacle. Moving towards F can be viewed as a local shortcut relative to the path that would be planned using contact sensors, in which the robot moves towards the target until touching the obstacle and then follows the obstacle boundary until reaching F. However, the robot's policy to change its detour direction right/left, based on locally optimal decisions, can cause unstable behavior and increase the path length.

Unstable behavior may happen when the robot motion changes the location of the sensed endpoints of the blocking obstacle, and thus affects the choice of the locally optimal direction (see figure 8). The problem is more apparent when the endpoints of the blocking obstacle are defined by the limited sensor range or by obstruction. However, this problem can also be observed while moving amidst smooth obstacles, when the endpoints are defined by the tangents to the real obstacle.

To avoid changing the detour direction due to infinitesimal improvements in the expected path length to the target (as may happen in the example presented in figure 8), we introduce the *detour constrain*. The detour direction is changed from moving towards O_1 to moving towards O_2, where O_1, O_2 are the endpoints of the blocking obstacle, only if

□

Proposition 4.7 *The Upper bound W_{max} for the path length produced by the algorithm TangentBug, using contact sensors, is:*

$$W_{max} = \|S,T\| + \sum_i P_i + \sum_i P_i \times \sharp Minima_i$$

where $\sum_i P_i$ refers to the perimeter of the obstacles intersecting the disc of radius $\|S,T\|$ centered at T, and $\sharp Minima_i$ is the number of local minima of $d(x,T)$ on the boundary of obstacle i.

Proof: The first element of the sum $\|S,T\|$ bounds the path length of *direct* subsegments of the moving-towards-the-target behavior. These are straight subsegments which satisfy the following properties: the endpoint of each subsegment is closer to the target than the starting point of the same subsegment and the starting point of each subsegment is closer to the target than the endpoint of the previous subsegment (see Lemma 4.3).

The second element of the sum $\sum_i P_i$ bounds the path length of *sliding* subsegments of the moving-towards-the-target behavior. The robot cannot traverse the same part of the boundary twice during a single segment because the distance to the target decreases during motion towards the target. Successive segments are non-overlapping, because the leaving condition guarantees $d(L_i,T) > d(H_{i+1})$. It follows that the total length of the *sliding* subsegments is bounded by the perimeter of the obstacles which the robot hits on its way. The distance from hit points to the target decreases along the path, therefore $d(H_i,T) < d(S,T)$ holds for every i. This property implies that the robot may hit only obstacles that intersect the disc of radius $\|S,T\|$ centered at T, thus $\sum_i P_i$ bounds the length of the sliding subsegments.

The third element of the sum $\sum_i P_i \times \sharp Minima_i$ bounds the path length of the boundary-following segments. To bound the number of these segments, we use the fact that the robot switches to boundary-following only in local minim points of $d(x,T)$ on the boundary of obstacles which the robot hits, and that the robot may switch into boundary-following only once in every local minimum. The path length of a single boundary-following segment is bounded by the perimeter of the followed obstacle. The bound for the path length of boundary following segments on a single obstacle i is the obstacle perimeter multiplied by the number of local minima on the obstacle's boundary $\sharp Minima_i$. This bound must be summed over all the obstacles that the robot may hit along its way $\sum_i P_i \times \sharp Minima_i$.

□

4.2 Using range sensors

Using range sensors, the robot's path becomes more complex; moving between obstacles is not limited to the direction towards the target, and following obstacle boundaries is performed while the robot senses the obstacles but not necessarily touches them. Consequently, bounding the path length becomes more difficult. We next re-define, for the range

Proof: Follows directly from Lemma 4.4. If the target is reachable then the robot will leave any obstacle before completing a loop around it, thus completing a loop around an obstacle means that the target is unreachable.

□

Corollary 4.6 *The distance to the target decreases between successive minima points* $d(M_i, T) > d(M_j, T)$ *where* $i < j$.

Proof: Moving from M_i to M_j, the robot follows an obstacle boundary until reaching a leave point L_i, which satisfies $d(M_i, T) \geq d(L_i, T)$. The robot then switches to moving-towards-the-target behavior. The leaving condition guarantees $d(L_i, T) > d(H_{i+1}, T)$. The distance $d(x, T)$ decreases during the motion towards the target, thus $d(H_{i+1}) \geq d(M_j)$. It follows that $d(M_i, T) > d(M_j, T)$.

□

Theorem 1 *The algorithm terminates after a finite path.*

Proof: Using contact sensors, swithching from moving-towards-the-target to boundary-following takes place in minima points M_i, which are local minima of the distance from the target function $d(x, T)$. Corollary 4.6 states that the distance to the target decreases between successive minima points along the path, thus switching to boundary-following can happen only once in each local minimum of $d(x, T)$. Lemma 4.1 states that the number of local minima points of $d(x, T)$ in the free configuration space is finite. It follows directly that the number of motion segments along the path is finite. Lemma 4.3 and Lemma 4.4 guarantee that the path length for each motion segment is finite, thus the total path length is finite.

□

Theorem 2 *If the target T is reachable from the starting point S then the robot will reach it in a finite path.*

Proof: Corollary 4.2 determines that the every moving-towards-the-target segment terminates either in the target or in a local minimum. When a local minimum M_i is reached, the robot switches to boundary-following. If T is reachable from S then Lemma 4.4 guarantees that every boundary-following segment will be terminated in a leave point L_i and followed by a moving-towards-the-target segment. We will show that the robot will eventually reach the target, because the number of boundary following segments is finite, and consequently there will be the **last** moving-towards-the-target segment, which terminates in the target.

We denote by N the number of local minima points of $d(x, T)$. Corollary 4.6 implies that switching to boundary-following can happen only once in each local minimum of $d(x, T)$. If the robot does not reach the target after leaving $N - 1$ obstacles, it must reach the N^{th} minimum point M_N. Lemma 4.4 guarantees that the robot will leave the N^{th} obstacle. The $N + 1^{th}$ moving-towards-the-target segment cannot be terminated in a local minimum M_{N+1}, because the robot have already visited all the local minima in the free space, thus it must be terminated in T.

it into a local minimum. Using contact sensors, a local minimum can be detected only when the robot reaches it, therefore the segment must terminate in a minimum point.

□

Lemma 4.3 *The path length of a moving-towards-the-target segment, using contact sensors, is finite.*

Proof: We consider a segment starting from L_i, ending in M_j and including several hit/departure points $L_i, H_{i+1}, De_{i+1}, ..., H_{j-1}, De_{j-1}, H_j, M_j$. [2]

The total length of the direct subsegments is bounded by the distance from the starting point to the target
$\|L_i, H_{i+1}\| + \|De_{i+1}, H_{i+2}\| + ... + \|De_{j-1}, h_j\| \leq \|L_i, T\|$.
This property holds because the endpoint of each subsegment H_{k+1} is closer to the target than the starting point of the same subsegment $d(L_i, T) > d(H_{i+1}, T)$, $d(De_k, T) > d(H_{k+1}, T)$, and the starting point of each subsegment De_k is closer to the target than the endpoint of the previous subsegment $d(H_k, T) \geq d(De_k, T)$.

The total length of the sliding subsegments is bounded by the perimeter of the obstacles which the robot hits on its way. The robot cannot traverse the same part of the boundary twice during a single segment because the distance to the target decreases along the segment.

□

Lemma 4.4 *If the target T is reachable from a minimum point M_i then the leaving condition will enable the robot to leave the obstacle boundary after a finite path.*

Proof: Consider a minimum point M_i, where the robot touches the obstacle O. The robot can reach any point on O's boundary using the boundary-following behavior. If the target is reachable from M_i, then there exist at least one point on the boundary of O from which the leaving condition holds. In particular we can look at a point C on the boundary of O which is closest to the target. When the robot reaches C, the minimal distance to the target on the followed obstacle is updated to $d_{followed}(T) \leftarrow d(C, T)$. The target T is reachable from C because it is reachable from M_i, therefore it is possible to move from C directly towards T. In this case $T_{node} \in$ LTG and $d(T_{node}, T) < d(C, T)$. The minimal distance to the target within the visible ϵ environment is updated to $d_{reach}(T) \leftarrow d(T_{node}, T)$. The leaving condition holds $d_{reach}(T) < d_{followed}(T)$. The robot path from M to C is finite because we assumed that the perimeter of every obstacle is finite.

□

Lemma 4.5 *If the robot completes a loop around an obstacle then the target is unreachable.*

[2] In the general case the number of $[H_k, De_k]$ pairs may be zero. If $M_j = T$ then the last hit point H_j is eliminated.

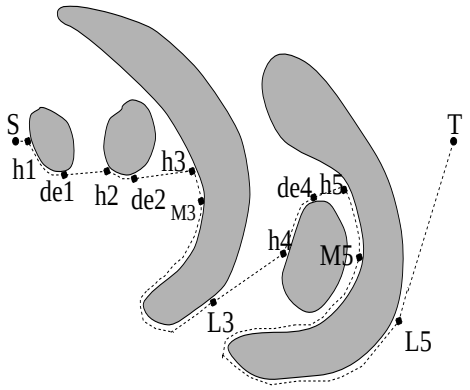


Figure 7: The segments of *TangentBug* path. The first moving towards the target segment starts from S, hits and departures from two obstacles $[h_1, de_1, h_2, de_2]$, hits the third obstacle at h_3 and ends at the minimum point M_3. The direct subsegments are $[S, h_1], [de_1, h_2], [de_2, h_3]$, and the sliding subsegments are $[h_1, de_1], [h_2, de_2], [h_3, M_3]$. The robot switches to boundary-following in M_3 and follows the obstacle boundary until the leave point L_3. The next moving-towards-the-target segment is $[L_3, h_4, de_4, h_5, M_5]$. The robot follows the obstacle boundary from M_5 to L_5 and then leaves the obstacle and moves directly towards the target T.

4.1 Using contact sensors

We first define several distinguished points and segments along the robot path: a hit point H_i is a point where the robot first touches an obstacle; a departure point De_i is a point where the moving-towards-the-target behavior drives the robot away from an obstacle boundary; a switch point Sw_i is a point where the robot switches from moving-towards-the-target to boundary-following; using contact sensors, each switch point is also a minimum point M_i, which is a local minimum of the function $d(x,T)$; a leave point L_i is a point where the leaving condition holds; the robot leaves an obstacle boundary and switches to moving-towards-the-target. We define the starting point S as L_0, and the target T as M_k for some $k > 0$ (because T is the global minimum of $d(x,T)$).

The robot path consists of several (1 to k) moving-towards-the-target segments, each starting from a leave point L_i and ending at a minimum point M_j. Each segment is divided into two types of subsegments: *direct* subsegments $[L_i, H_{i+1}], [De_k, H_{k+1}]$ in which the robot moves in straight line directly towards the target, and *sliding* subsegments $[H_k, De_k]$, in which the robot slides along the boundary of a blocking obstacle. The moving-towards-the-target segments are interleaved with boundary-following segments, in which the robot moves from a minimum point M_j to its corresponding leave point L_j (see figure 7). Note that the robot's distance from the target $d(x,T)$ decreases along sliding subsegments, in contrast to boundary-following segments in which this distance may increase.

Corollary 4.2 *A moving-towards-the-target segment terminates in a minimum point M_i, which is either the target or a local minimum of $d(x,T)$.*

Proof: The moving-towards-the-target behavior is terminated when the robot either reaches the target or detects that moving in the locally optimal direction will drive

figure (b) with dashed line, the resulting path is completely different. The LTG created at the starting point S consist of two obstacle endpoints $O1, O2$. The locally optimal direction is towards the endpoint $O2$, and the robot uses the moving towards the target behavior until the target is reached.

4 Convergence proof

The convergence of *TangentBug* algorithm is based on the following main ideas: using the moving-towards-the-target behavior, the robot's distance from the target is reduced and the path length is guaranteed to be finite. The robot may switch into boundary-following behavior only in a finite number of local minima points of the distance from the target function. The algorithm terminates after a finite time because switching to boundary-following may happen only once in each minimum point. Reaching the target is guaranteed because the leaving condition enables the robot to leave an obstacle boundary if the target is reachable. We will next present some assumptions and definitions, describe the convergence proof for the contact sensors case, and then consider range sensors.

We consider a point robot in a planar unknown configuration space populated with arbitrary obstacles. Assuming that the workspace is bounded, it follows directly that the perimeter of any obstacle is finite, and that the number of obstacles is finite. We define the function $d(x, T)$ over the free configuration space to be the Euclidean distance from x to the target T.

Lemma 4.1 *There is a finite number of isolated local minima points of $d(x, T)$ over the free configuration space.*

Proof: In the following, we will need the basic topological fact that any set of isolated points in a compact space (i.e. closed and bounded) is finite. Further, in the following we will assume that the obstacle boundaries are smooth curves (the result can be extended to piecewise smooth boundaries). First, d(x,T) clearly can have critical points only on the obstacle boundaries, never in the interior of the free space. The restriction of d(x,T) to the i'th boundary is a smooth function. The critical points of d(x,T) on the boundary are precisely those points where the vector (x-T) is orthogonal to the boundary. Recall now the definition of Morse function. It is a smooth function whose second derivative matrix is non-singular at all the critical points of the function. It is known that almost all smooth functions on a given smooth manifold (the i'th obstacle boundary in our case) are Morse [8]. Moreover, the critical points of a Morse function are always isolated. Hence d(x,T) has finitely many local minima on almost any obstacle boundary. In our case the obstacle boundaries on which d(x,T) is non-Morse can be characterized as follows. These are the obstacle boundary which have a critical point of d(x,T) at which the circle of curvature of the boundary has the target T at its center. This is clearly a non-generic situation. Moreover, it seems that even then the only way by which the critical point can be non-isolated is when the boundary contains a circular segment centered at T.

□

following behavior is not strictly reactive; the followed obstacle is determined initially and then traced in the successive steps, considering the displacement of the robot between steps. Note that, during boundary following, the followed obstacle may differ from the blocking obstacle, when the followed obstacle does not block the direction towards the target.

Another initial action is choosing a *minimum point* M, which is a closest point to the target on the observed boundary of the followed obstacle. The minimum point is used for detection of loop completion around the followed obstacle.

The boundary following direction (clockwise/counterclockwise) is chosen using the detour direction, defined by the moving towards the target behavior, in the switch point Sw. For example, if the detour direction is left then the boundary following direction is clockwise.

At each step the robot constructs the LTG, locates the followed obstacle in it, and focuses on those nodes of the LTG which lie on the followed obstacle. The motion direction during boundary following is towards the left/right endpoint of the followed obstacle, in the defined following direction.

As the robot moves around the followed obstacle it updates two variables, which register minimal distances observed along the path. The shortest distance from the target **of the followed obstacle's boundary** is registered in $d_{followed}(T)$, which is initialized to $d_{followed}(T) \leftarrow d(M,T)$. The shortest distance from the target **within the visible environment** is registered in $d_{reach}(T)$. To insure that the robot stays on locally optimal paths, we constrain the update of $d_{reach}(T)$ in the following way. When there is a blocking obstacle, $d_{reach}(T)$ is updated by the shortest distance from the target of the blocking obstacle's boundary. When there is no blocking obstacle, $d_{reach}(T)$ is updated by $d(T_{node},T)$.

The robot leaves an obstacle boundary when it can reach, via free space, a point which is closer to the target than the distance $d_{followed}(T)$. This test guarantees that the robot will not be trapped again in local minima which it passed during the boundary following motion. The following leaving condition is tested is every step:
if $d_{reach}(T) < d_{followed}(T)$ then leave the obstacle boundary.

The parameter $d_{NextMin}(T) \leftarrow d_{reach}(T)$ is transfered to the motion towards the target behavior (the role of this parameter will be described in corollary 4.12).

3.4 An example

An example of $TangentBug$ algorithm, compared to $Bug2$ algorithm from [13], is presented below (see figure 1). The path planned by $Bug2$ using contact sensors is presented in figure (a) with solid line. The algorithm $VisBug21$ from [12], with unlimited sensor range, plans local shortcuts relative to $Bug2$ path, as presented in figure (a) with dashed line. Using $VisBug21$ the robot leaves the obstacle boundary in $L2$, as soon as the straight line $[S,T]$ is visible to the robot.

The path planned by $TangentBug$ using contact sensors is presented in figure (b) with solid line. The robot switches from motion towards the target to boundary following at point $Switch$, where it reaches a local minimum of the distance from the target. It leaves the first obstacle at the point $L1$ and moves towards the target, going around the second obstacle, until the target is reached. When unlimited sensor range is used, as presented in

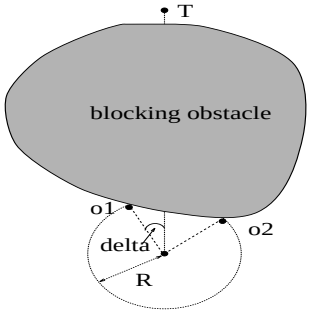


Figure 6: An example of moving towards the target behavior. There is a blocking obstacle, and the locally optimal direction is towards node O_1 of the LTG. The detour angle is *delta*, and the detour direction is left.

between the motion direction and the direction towards the target. We define the *detour direction*, right/straight/left, as $sign(\delta)$.

Lemma 3.1 *If $T_{node} \in LTG$, meaning that there is no blocking obstacle, then the shortest path will be along the edge towards T_{node}. Otherwise the endpoints of the blocking obstacle are the only candidates for the shortest path.*

The motion towards the target terminates when the robot detects that moving in the locally optimal direction will drive it into a local minimum. This situation happens when the blocking obstacle creates a local minimum in the distance from the target function. As the robot moves towards the target, it monitors the closest point to the target M on the boundary of the blocking obstacle. If the distances from the endpoints O_1, O_2 to the target satisfy $d(O_1,T) > d(M,T)$ and $d(O_2,T) > d(M,T)$ then the robot concludes that it is within the basin of attraction of the local minimum M and switches to boundary following behavior.

An additional constrain to the direction choosing mechanism, which is necessary for preserving convergent properties when the robot switches from boundary following to moving towards the target, is presented below (the role of this constrain will be described in corollary 4.12). Given the parameter $d_{NextMin}(T)$ and as long as $d(x,T) > d_{NextMin}(T)$, a node $v \in$ LTG is considered as a candidate for the locally optimal direction only if $d(v,T) \leq d_{NextMin}(T)$.

3.3 Following an obstacle boundary

The obstacle boundary following behavior is used to drive the robot away from a local minimum. A boundary following direction is first chosen, and the robot moves 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. We will describe the initial actions performed when this behavior is activated, present the motion planning mechanism, and define the leaving condition.

When obstacle boundary following behavior is initiated, in a *switch point* Sw, the algorithm first defines the *followed obstacle* as the current blocking obstacle. It can be verified that this obstacle is the one which caused the local minimum. The boundary

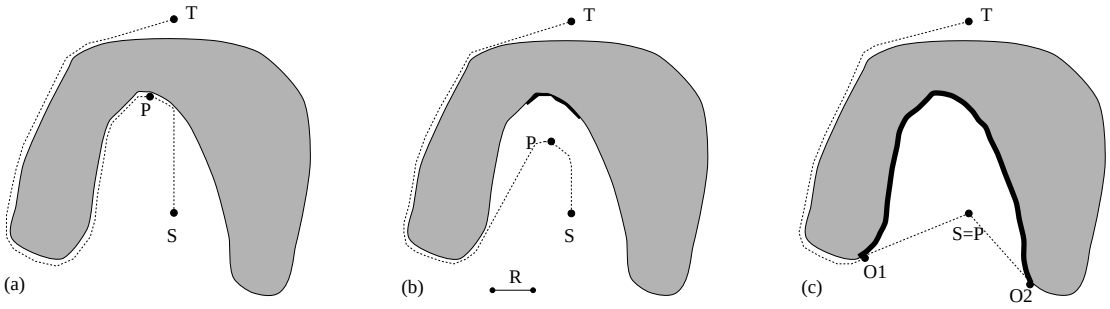


Figure 5: The robot motion near a concave obstacle. The motion towards the target terminates at P, where the robot detects that moving in the locally optimal direction will drive it into a local minimum. The robot then switches to boundary following. (a) using contact sensors; (b) using limited sensor range R. The sensed boundary from location P is marked with a bold line; (c) using unlimited sensor range. The robot switches into boundary following mode at the starting location S.

2. Choose a boundary following direction. Move along the boundary using the current LTG while recording $d_{followed}(T)$, the minimal distance from the target along the followed obstacle's boundary and $d_{reach}(T)$, the minimal distance from the target within the visible environment, until one of the following events occurs:

   - The target is **reached**. Stop.
   - The "leaving condition" holds: $d_{reach}(T) < d_{followed}(T)$. Go to step 1.
   - The robot completed a loop around the obstacle. The target is **unreachable**. Stop.

Note that the motion toward the target (step 1) includes both motion between the obstacles and sliding along obstacle boundaries. This motion mode persists as long as the distance function $d(x,T)$ decreases and the robot is not trapped in the basin of attraction of a local minimum.

3.2 Moving towards 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 target according to the LTG. The candidate motion directions are the directions towards 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 [1].

If an obstacle is detected, within the sensor range, between the robot and the target, we term it the *blocking obstacle*. We define the *detour angle* $\delta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ as the angle

[1]The expected path length from an obstacle endpoint O_i to the target is $d(O_i,T)$ if the straight line $[O_i,T]$ does not intersect any sensed obstacle. If there is an intersection then the shortest path length to the target must be calculated considering the geometry of the sensed boundary.

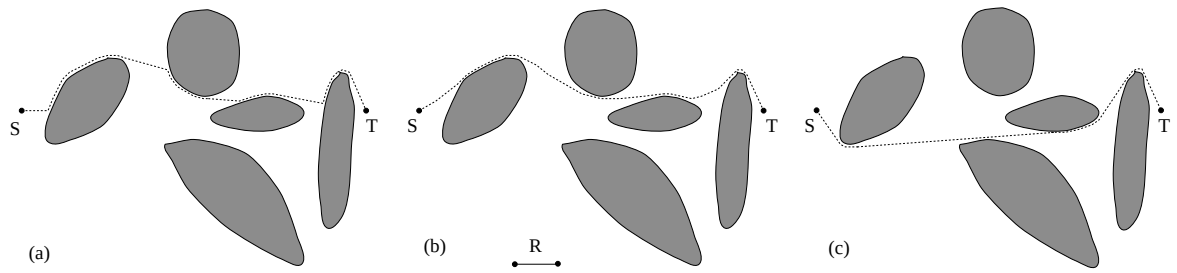


Figure 4: The robot path in a simple environment using the moving towards the target behavior. (a) using contact sensors; (b) using limited sensor range R; (c) using unlimited sensor range. Note the global change in the path caused by the increase of the sensor range.

3.1 Algorithm description

The $TangentBug$ algorithm plans the motion of a point robot in a planar unknown configuration space populated with arbitrary obstacles. The algorithm is based on two basic behaviors, which are governed by the function $d(x,T)$, which measures the distance of the robot, located at the point x, from the target. The two behaviors, or modes of motion, are moving towards the target and following an obstacle boundary. The motion towards the target decreases $d(x,T)$, while the boundary following motion attempts to escape from a local minimum of $d(x,T)$. In every step the robot creates a local tangent graph, LTG, based on its immediate range readings. It uses the LTG to plan the next action in the following way. While moving towards the target, the robot chooses a greedy *locally optimal direction*, which is the direction along the shortest path to the target according to the LTG. In simple scenarios, where the local decisions of turning right/left are correct from a global point of view, this behavior will drive the robot in the optimal path considering the robot's incomplete information (see figure 4). The motion towards the target terminates when the robot detects that moving in the locally optimal direction will drive it into a local minimum. This situation happens when an obstacle, placed between the robot and the target, creates a local minimum in the distance from the target function (see figure 5). The obstacle boundary following behavior is then used to drive the robot away from the local minima in the following way. A boundary following direction is chosen, then the robot moves along the obstacle boundary while continuously monitoring the LTG. The LTG is used to plan local shortcuts relative to the obstacle boundary, but the robot may not leave the boundary before the following *leaving condition* is met. 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. We now describe the $TangentBug$ algorithm in detail, and provide several examples of its operation.

1. Move towards the target along the *locally optimal* direction, until one of the following occurs:

   - The target is **reached**. Stop.
   - Moving in the locally optimal direction will drive the robot into a local minimum. Go to step 2.

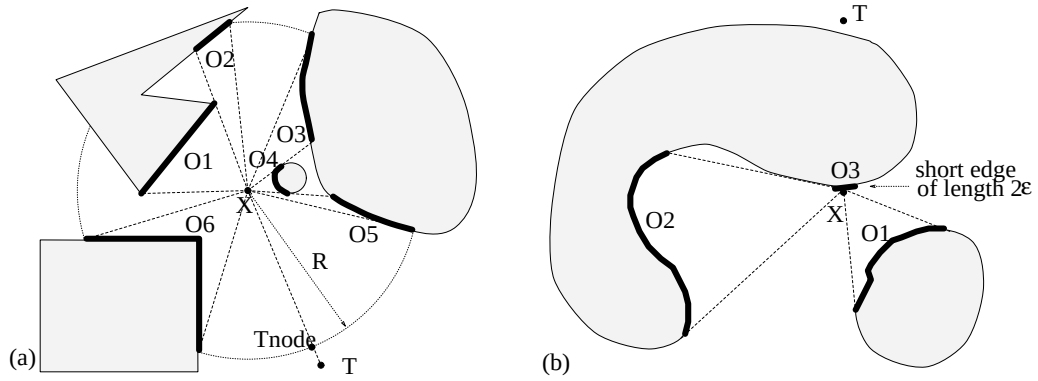


Figure 3: (a) The LTG using detection range R. The sensed obstacles $\mathcal{O}_1, ..., \mathcal{O}_6$ are modeled as thin walls. The LTG edges connect the robot location x with the endpoints of the sensed obstacles and T_{node}. (Bi-tangent edges are not shown.) (b) The LTG when the robot touches a convex obstacle boundary, using unlimited sensor range. The endpoints of the touched boundary are represented as points at distance ϵ from x. Note that $\mathcal{O}_2$ and $\mathcal{O}_3$ are distinct sensed obstacles in this case.

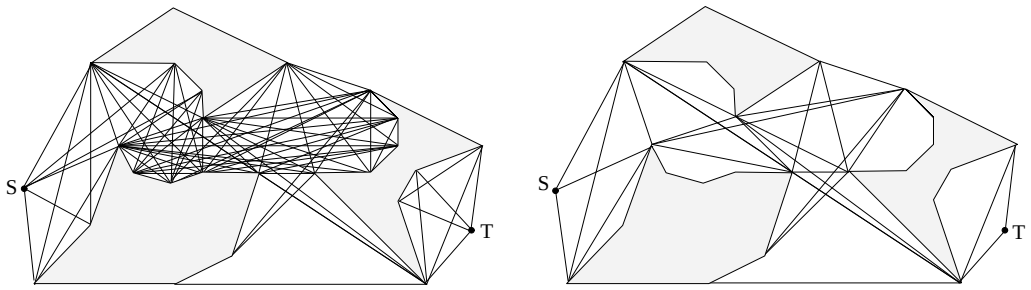
contact point. We represent the boundary by a short edge of length 2ϵ tangent to the boundary, as shown in Figure 3(b).

The nodes of the LTG are the current robot location x, the endpoints of the sensed obstacles, and optionally an additional node in the direction of the target called T_{node}. If the line segment from x to T is not blocked by an obstacle, we set T_{node} at the furthest point on this line within the visible set. In particular, T_{node} is set to T if the target lies in the visible set. The edges of the LTG connect the robot location x with all the other nodes of the LTG. The LTG additionally contains edges which are bi-tangent to the sensed obstacles, but the algorithm never uses these edges and they are not explicitly constructed. Two examples of the LTG are shown in Figure 3.

Contact sensors require a special treatment to fit the above definitions. We assume that the contact sensors can determine if there is free space in the direction of the target, and can recover the local orientation of a touched boundary. These two types of information are modeled as distance readings in a small range $R = \epsilon$, where $\epsilon > 0$ is a small scalar. If there is free space in the direction of T, it is represented by placing T_{node} at a distance ϵ from x in the direction of T. The boundary touched by the robot is represented by a short edge of length 2ϵ tangent to the boundary, and the endpoints of this edge become vertices of the LTG.

3 The *TangentBug* algorithm

First we describe the global structure of the algorithm, and then present the details of the motion behaviors and the transition conditions.



(a) (b)

Figure 2: (a) The visibility graph compared to (b) the tangent graph

$\|E_t\| \leq 2M^2 + N$. Since M is typically much smaller than N, we see that $\|E_t\|$ is $O(N)$ while $\|E_v\|$ is $O(N^2)$.

The tangent graph was also generalized in [11] to curved obstacles. In this case the boundary of each obstacle is partitioned into convex and concave arcs. The edges of TG are the convex boundary arcs, and the edges which are bi-tangent to convex boundary arcs. In that case the number of edges in TG is bounded by $\|E_t\| \leq 2M^2 + N$, where M is the number of convex obstacles and N is the number of distinct convex arcs on the obstacle boundaries.

2.2 The Local Tangent Graph

We assume a range finder which provides readings 360° around the robot. The range finder provides perfect readings of the distance to the obstacles within the *visible set*, which is the star-shaped set centered at the robot whose maximal radius is R. The *local tangent graph*, or LTG, is a tangent graph that includes only the portion of the obstacles which lie in the visible set. The LTG is constructed as if the local range information represents all the obstacles in the environment. This assumption allows the robot to base its local decisions only on the currently observable obstacles, thereby greatly simplifying the data processing. The local range data is first divided into distinct *sensed obstacles* by a process described below. Each sensed obstacle is then modeled as a thin wall in the real world. This assumption would always underestimate the obstacles' size. However a more accurate model would require sophisticated and computationally expensive modeling techniques.

Next we describe the process of partitioning the range data into sensed obstacles. The range information can be represented as a function $r(\theta)$, defined on the interval $[0, 2\pi)$, where θ is the angle relative to a predefined direction and $r(\theta)$ is the distance, in direction θ, from x to the nearest point on an observed boundary. The range data is divided into distinct sensed obstacles based on the fact that the range changes continuously along the boundary of a single visible part of an obstacle. The function $r(\theta)$ has a finite number of discontinuity points, since discontinuities occur only at angles where an obstacle is obstructed. A continuous ranges interval $(r(\alpha), r(\beta))$, whose endpoints are either discontinuity points or points where $r(\theta) = R$, is considered a distinct sensed obstacle. Note that a single real obstacle can give rise to several sensed obstacles, as shown in Figure 3(a). A special sensed obstacle occurs when the robot touches an obstacle boundary at a convex point. In that case the visible set locally intersects the boundary only at the

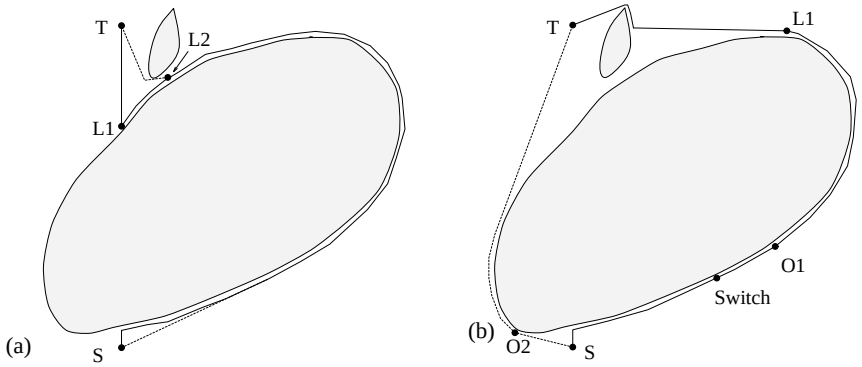


Figure 1: An example comparing $TangentBug$ with $Bug2$ and $VisBug$. (a) The path generated by $Bug2$ using contact sensors (solid line) is modified and shortened by $VisBug$ with unlimited sensor range (dashed line). (b) The path generated by $TangentBug$ using contact sensors (solid line), and the path generated by $TangentBug$ using unlimited sensor range (dashed line). This example is discussed in detail in Section 3.

2 The Tangent Graph

This section reviews the notion of tangent graph, a structure which requires global knowledge of the environment. Then the sensor-based local tangent graph is introduced.

2.1 Review of the Global Tangent Graph

First recall the definition of the *visibility graph*. Consider a polygonal environment in which a starting point S and a target T are specified. The visibility graph, denoted $VG(V_v, E_v)$, is the undirected graph whose vertices V_v are the obstacle vertices and the points S and T, and whose edges E_v are the collision-free line segments which connect the graph vertices (Figure 2(a)). It is known that the shortest collision-free path from S to T always lies on the visibility graph. Thus, an optimal path from S to T in the environment can always be found by limiting the search to the visibility graph. However, the visibility graph contains many edges which never participate in the shortest path. For purpose of comparison with the tangent graph, if there are N obstacle vertices, the number of edges in the visibility graph is bounded by $\|E_v\| \leq N^2$.

The tangent graph, denoted $TG(V_t, E_t)$, was introduced in [11]. It is the subgraph of VG obtained by retaining only the convex obstacle vertices and the edges which are bi-tangent to convex obstacle vertices (Figure 2(b)). Like the visibility graph, the tangent graph contains the shortest path in the environment. But the tangent graph is the minimal such graph, as it can be shown that the removal of any of its edges would destroy its optimality. The tangent graph has a significantly smaller number of nodes and edges than the visibility graph, as Figure 2 shows. Thus searching for the shortest path is more efficient on the tangent graph. To estimate the size of TG, assume that the obstacles are comprised of union of M convex polyhedra, such that the boundary curves of any two overlapping convex polyhedra intersect at two points. (This condition can always be met by suitable subdivision of the polyhedra.) It is known that any two convex polyhedra can have at most 4 bi-tangent edges. Hence the number of edges in TG is bounded by

VisBug [12]. Several versions of leaving conditions and internal representations appear in [14, 15, 16, 20].

The midway approach of the *Bug* algorithms reduces the reliance on a global model to the essential minimum of loop detection, while augmenting the purely reactive navigation decisions with a globally convergent criterion. This approach thus minimizes the computational burden on the planner while still guaranteeing global convergence to the target. However, the *Bug* algorithms do not make the best use of the available local information to produce short paths. These algorithms use mainly contact sensors, and range data is used in *VisBug* only to find shortcuts to the path generated by *Bug*2 (Figure 1(a)). This paper presents a new *Bug* algorithm which specifically exploits range-data. The new algorithm, termed *TangentBug*, uses the recently introduced notion of *tangent graph* to produce paths which often resemble the shortest path to the goal (Figure 1(b)). The tangent graph of a planar configuration space is the smallest subgraph of the visibility graph which still contains the shortest path [11]. Let us briefly outline the algorithm.

The *TangentBug* algorithm uses the basic behaviors of motion towards the target and obstacle boundary following. We introduce a local range-data based version of the tangent graph, termed the *local tangent graph*, or LTG. In every step the robot constructs the LTG based on the local range-data, and uses it to plan the next action as follows. Let $d(x,T)$ be the distance of the robot, located at a point x, from the target T. Then during the motion towards the target, $d(x,T)$ decreases monotonically. During the boundary following, the robot attempts to escape from a local minimum of $d(x,T)$. While moving towards the target, the robot chooses a *locally optimal direction*, which is the direction along the shortest path to the target according to the current LTG. The motion towards the target terminates when the robot detects that moving in the locally optimal direction would drive it into a local minimum. The obstacle boundary following is then invoked to drive the robot away from the local minimum. The robot first chooses a boundary following direction, then it moves along the boundary while continuously monitoring the LTG. The robot uses the LTG to make shortcuts along the obstacle boundary, but it may not leave the boundary before the following *leaving condition* is met. Let $d_{followed}(T)$ be the minimal distance from T observed along the followed obstacle so far. Then the robot leaves the obstacle boundary when it can reach a point P, within its visible environment, such that $d(P,T) < d_{followed}(T)$.

The paper is organized as follows. Section 2 reviews the tangent graph and introduces the LTG. Section 3 presents the *TangentBug* algorithm. In Section 4 we show that *TangentBug* is globally convergent, and discuss general bounds on the performance of the algorithm. We will present *TangentBug* as a 1-parameter family of algorithms, parameterized by the range-sensor detection distance R, where $0 \leq R \leq \infty$. In Section 5 we describe simulation results, where we study the dependence of the resulting paths on the parameter R. The simulations show that *TangentBug* often generates paths that approach the shortest path as R increases. The simulations also compare *TangentBug* with the classical *VisBug* algorithm, showing that *TangentBug* generates significantly shorter paths in congested office-like scenarios.

1 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 the robot cannot base its motion planning on complete apriori knowledge of the environment. The robot must rather use its sensors to perceive the environment and plan accordingly. The two main sensor-based motion planning approaches use either global planning or local planning. Let us briefly describe these approaches and point out their limitations.

In the global sensor-based planning approach, the mobile robot builds a global world model based on sensory information and use it for path planning [6, 21, 22]. This approach guarantees that either the target will be reached, or the robot will map its entire accessible environment and 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. Moreover, the reliance on a global model for the navigation requires frequent localization of the robot relative to the model, a process which is difficult to attain due to the inherent uncertainties of practical sensors [5, 10, 18]. Recent works also use the global approach to achieve sensor-based navigation of general robots, not just mobile robots [4, 19]. These works propose algorithms for the incremental construction of a roadmap for the robot configuration space as the global model. However several implementation issues of these algorithms remain unsolved.

In contrast, local path-planners use the local sensory information in a purely reactive fashion. In every control cycle the robot uses its sensors to locate the nearby obstacles, and then it plans and performs the next action. Local planners are usually much simpler to implement than the global ones, since they typically use navigation vector-fields which directly map the sensor readings to actions. Different methods are used to choose or learn these vector-fields, including the potential-field method and its variations [1, 9], fuzzy logic approaches [7, 17], and methods which construct specialized data structures to make more reliable decisions [2, 3]. However, while the local approaches are simple to implement, they do not guarantee global convergence to the target. A local planner may get trapped in a local minimum, and subsequently follow a diverging path or a loop while attempting to escape from the local minimum. Once trapped in a loop, the robot has no way to escape it since the sensory information is inherently local, and the same action will always be taken for the same local situation.

Thus, the global approaches suffer from practical implementation problems, while the local ones lack a global convergence guarantee. This paper focuses on a midway approach, originated by Lumelsky and Stepanov [13], which combines local planning with global information. This approach consists of two reactive modes of motion termed *behaviors*, and transition conditions for switching between them. The two behaviors are moving directly towards the target and following an obstacle boundary. When the robot hits an obstacle it switches from moving towards the target to boundary following. It leaves the obstacle boundary when a *leaving condition*, which monitors a globally convergent criterion, holds. The leaving condition of [13] ensures that the distance to the target decreases monotonically at successive hit points, thus guaranteeing global convergence. However, the algorithms of [13], termed $Bug1$ and $Bug2$, use only position and contact sensors. Range data was incorporated only at a later stage, in an algorithm termed

Abstract

We present TangentBug, a new range-sensor based globally convergent navigation algorithm for mobile robots. We incorporate the idea of the locally shortest path, using the tangent graph, into the decisions of a local planner which exploits range-data to produce short paths. We adjust the structure of the tangent graph, which was defined for a completely known environment, and introduce a local range-data based version of it, termed the local tangent graph*, or LTG.*

Our algorithm belongs to the Bug family from [12], which combines local planning with global information that guarantees convergence. We re-formulate the basic behaviors of the Bug family, and define new transition conditions for switching between them. The TangentBug algorithm uses the basic behaviors of motion towards the target and obstacle boundary following. Let $d(x,T)$ be the distance of the robot, located at a point x, from the target T. Then during the motion towards the target, $d(x,T)$ decreases monotonically. During the boundary following, the robot attempts to escape from a local minimum of $d(x,T)$.

The TangentBug algorithm specifically exploits range-data, and uses it for choosing the locally optimal direction while moving towards the target, for choosing the boundary following direction when boundary following behavior is initiated, and for the transition condition which terminates boundary following. The simulation results show improvement in the path length as the sensor range increases. The results indicate a significant advantage of TangentBug relative to the algorithm VisBug from [12], in all the tested scenarios.

A New Range-Sensor Based Globally Convergent Navigation Algorithm for Mobile Robots

Ishay Kamon
Dept. of Computer Science
Technion

Elon Rimon
Dept. of Mechanical Engineering
Technion

Ehud Rivlin
Dept. of Computer Science
Technion

September 5, 1995