Decentralized motion planning for multiple mobile robots: The Cocktail Party Model*

V.J.Lumelsky and K.R.Harinarayan
University of Wisconsin-Madison
Madison, Wisconsin 53706, USA

Abstract

This paper presents an approach for decentralized real-time motion planning for multiple mobile robots operating in a common 2-dimensional environment with unknown stationary obstacles. In our model, a robot can see (sense) the surrounding objects. It knows its current and its target's position, is able to distinguish a robot from an obstacle, and can assess the instantaneous motion of another robot. Other than this, a robot has no knowledge about the scene or of the paths and objectives of other robots. There is no mutual communication among the robots; no constraints are imposed on the paths or shapes of robots and obstacles. Each robot plans its path toward its target dynamically, based on its current position and the sensory feedback; only the translation component is considered for the planning purposes. With this model, it is clear that no provable motion planning strategy can be designed (a simple example with a dead-lock is discussed); this naturally points to heuristic algorithms. The suggested strategy is based on maze-searching techniques. Computer simulation results are provided that demonstrate good performance and a remarkable robustness of the algorithm (meaning by this a virtual impossibility to create a dead-lock in a "random" scene).

1 Introduction

We address the problem of decentralized control and motion planning for multiple mobile robots operating in a common planar environment, perhaps among stationary obstacles. Robots and obstacles may be of arbitrary shape. The task of each robot is to reach its target position. A robot plans its motion based on the local information from its sensors (say, vision or range finders) and on its planning algorithm. There is no direct communication between the robots – effectively, each one is a moving obstacle for other robots. A robot has no knowledge about other objects in the scene until it sees them.

This is quite similar to the situation one faces in a crowded place, such as a cocktail party – hence the name *The Cocktail Party Model.* When a guest decides to talk to someone, he accomplishes

*Supported in part by the National Science Foundation Grant IRI-9220782.

this by maneuvering between tables, chairs, and other guests, planning his path "on the fly" and not consulting with other people about his or their intended motion. He assumes that other people mean well, and so as long as he somehow takes into account their movements, it is safe to move at a minimal distance from them. If, on the other hand, one of the guests does not fit this assumption ("Is he drunk?"), one will increase the safety margin distance when passing this person. Applications that fit this multi-agent model include mobile robots in large assembly plants and automated factories (e.g., automatic paper roll carriers in a paper mill), specialized assembly systems [1], military tasks, and intelligent highway control systems.

Two obvious approaches to motion planning in multi-agent systems are the *centralized* and *decentralized* (distributed) approach. Both have their pros and cons. The usual scheme for centralized control has its rationale in the factory floor tasks: in it, a central planner designs the motion plan for all robots based on full knowledge about the environment. Only after the complete paths have been computed, the actual motion takes place. The approach fits better purely computational problems rather than tasks that rely on real-time feedback control. Its obvious advantage is its conceptual simplicity: since everything is known, anything can be computed, including the optimal (shortest, smoothest etc.) trajectories for all agents. The price for this convenience is computational bottlenecks. The amount of computations quickly grows with the number of agents, and is likely to become unwieldy in a task with 4-6 robots. Since planning is done off-line, complete recalculation of paths is required if one of the robots' objectives are altered or if the environment changes.

Decentralized control has two obvious advantages: (i) It breaks the computational bottleneck of centralized control; in principle, computational complexity of a decentralized system can be made independent of the number of agents in it. (ii) It is inherently more stable and robust: it can tolerate changes and uncertainly; a failure of one or few agents does not kill the whole system. On the negative side, the decentralized control is inherently incapable of delivering optimal performance: since at any moment each agent is lacking some information, optimality is ruled out. Instead, a reasonable, acceptable performance is sought. The main question posed in this work is whether decentralized motion planning can deliver good performance in a reasonably complex system. The answer seems to be "yes".

Most of the literature on multiple robot motion planning is devoted to the centralized approach. Efforts tend to concentrate on decreasing the computational cost. This is typically achieved at the expense of completeness (which may be acceptable in some applications). In [2] the task is divided into two subtasks. First, each robot's path is determined taking into account only stationary obstacles. With the paths fixed, velocities of all the robots are then adjusted so as to avoid collisions. In [3] priorities are assigned to each robot and planning is done for one robot at a time: each robot's path is planned in the three dimensional space-time configuration space taking into account the stationary obstacles as well as the motion of the robots with a higher priority. A scheme based on priorities and attempting to maximize the number of robots traveling in a straight line has been considered in [4]. All these are heuristic algorithms, in the sense that they cannot guarantee the robot will find a path if one exists, or prove that there is no path if true.

Considerable research has been done in the area of provably correct algorithms. An $O(n^3)$ algorithm for planning the motion of two disks in a polygon-filled scene, and an $O(n^{13})$ algorithm for planning the motion of three disks have been presented in [5], where n is the number of sides of the polygonal obstacles in the environment. In general, such algorithms are polynomial in the complexity of the obstacles and exponential in the number of disks. For an enclosed space, using the idea of retraction, $O(n^2)$ and $O(n^3)$ algorithms have been devised for the motion of two and three disks respectively [6]. In [7] the problem of moving many disks among polygonal obstacles has been shown to be NP-hard. Coordinating the motion of an arbitrary number of rectangles which can only translate in a rectangular two-dimensional region has been shown to be PSPACE-hard [8].

The decentralized approach is based on the *model with incomplete information* and assumes no central planner. Each agent acts independently, planning its path based on its goal and on local and limited global information. The latter typically comes via sensory feedback (e.g.. from ultrasound sensors, a range finder, or a camera), and the path is planned dynamically in real time. An ideal decentralized strategy would require no direct communication between the robots, while ensuring collision avoidance and minimal interference of each robot with the purposeful motion of the other robots. The algorithmic methodology here makes use of maze-searching techniques [9, 10].

Examples of decentralized motion planning in multi-agent systems – say, with the crowded cocktail party above, or with automobiles on a highway – suggest that even when there is no direct communication, usually there is a kind of shared expectation of a "reasonable behavior" that agents use as a guideline in their planning strategies. For example, in [11] such shared information appears in the guise of separating lines between robots, to ensure collision avoidance.

We formulate the problem as one of maze searching, albeit in a dynamically changing "maze". The emphasis is on formal algorithmic issues of decentralized control, on a dynamically changing environment, and on objects of arbitrary shapes. One standard question in maze-searching algorithms is that of convergence: if a path between starting and target positions does exist, we would like the algorithm to guarantee finding one. Interestingly, while this is possible for one body moving among stationary objects, or for centralized planning, it is not feasible in the context of decentralized control (see Discussion).

To our knowledge, the seemingly natural idea of extending the methodology of sensor-based motion planning to decentralized control has not been explored in literature. Given this emphasis, some relevant issues are left out below:

(i) No connection is made to learning and collective behavior [12, 13]. Note that learning has little meaning for moving objects – they won't be there next time around. It could make sense for stationary objects though; note, however, that our model's allowing objects to be of arbitrary shapes, while adding power to the algorithm, makes learning problematic since storing arbitrary shapes requires, in principle, infinite memory.

(ii) We assume perfect sensing and precise knowledge of the robots' and their targets' positions (see a discussion about the issue of uncertainty in real sensory data in Section 6).

We make use of a simple mechanism of "reasonable behavior": when maneuvering to avoid a potential collision, each robot R_i assumes that other robots will try to avoid collisions as well. More specifically, when accounting for an approaching robot(s), R_i plans its next step so as not to cross an invisible boundary that separates the "safe areas" of the two robots. Two alternative mechanisms for such a boundary are suggested – the Voronoi diagram [14] and the perpendicular bisector to the line of minimum distance between R_i and the other robot. In other words, although the motion and sensing parameters (velocities, the step size, sensing range etc.) may differ widely from robot to robot, each robot can safely assume that the other robots operate under the same "civilized" strategy.

Although the suggested approach makes use of a provably correct motion planning algorithm [9], it is heuristic in nature. This means, for example, that a robot may fail to find a path to its destination even if one exists (one such example is discussed in Section 6). It is important to note that this lack of guaranteed convergence is not the result of a weak algorithm: as long as the agents' decision-making processes are independent, provably correct algorithms are not feasible anymore. What is interesting is that the algorithm that emerges exhibits remarkable robustness in complex scenes. As used here, the term "robustness" means that, short of degenerate examples in which all robots' paths must be carefully coordinated in a centralized fashion, it is virtually impossible for a robot not to reach a reachable target under the suggested algorithm (Section 6).

The remainder of the paper is arranged as follows: the model of the robot and the environment are introduced in Section 2. Details of the approach are developed in Section 3. The final algorithm is presented in Section 4, followed by examples of its operation in Section 5 and a discussion of the performance issues in Section 6.

2 The model

The environment (the scene) is a plane; it is populated by *objects*. An object's *boundary* represents its shape and is a simple closed curve of finite length. No constraints are imposed on an object's shape. (What makes this happy generality possible is that the obstacle boundary needs not be represented for the planning purposes, and thus requires no representation scheme, since at any instance a robot will deal with only a tiny part of the boundary.) Objects can be of two types – they are either stationary *obstacles* or mobile *robots*, Figure 1a. The corresponding *configuration space (C-space)* of a robot is the space of the robot's translation variables x and y; it can be obtained

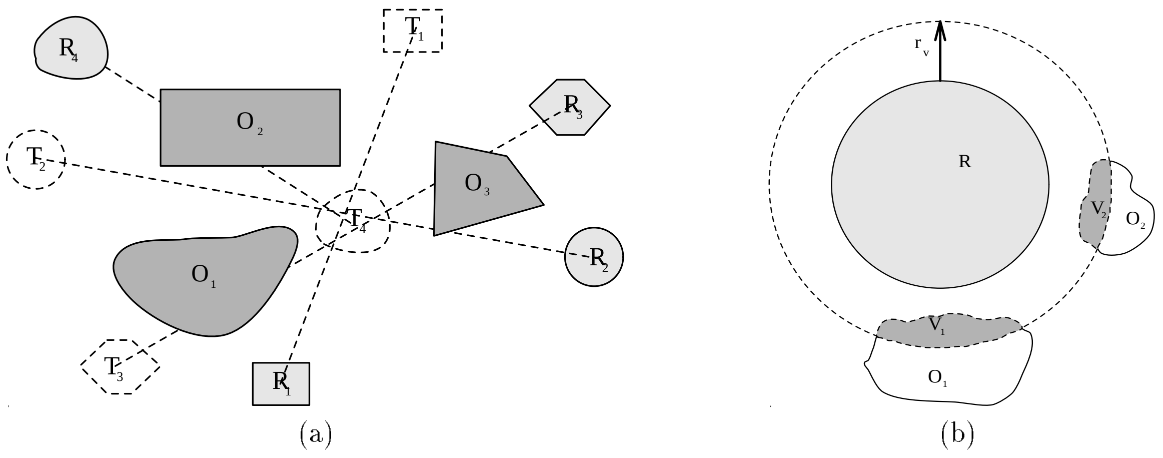

Figure 1: *(a) An environment with mobile robots* $R_1, ..., R_4$ *and stationary obstacles* $O_1, ..., O_3$. T_i *is the desired target location of robot* R_i. *(b) The scanning operation: robot* R *is equipped with range sensors which operate within the sensing radius* r_v. *The robot can see only portions* V_1 *and* V_2 *of objects* O_1 *and* O_2*;* V_i *thus forms the* visible objects *for* R.

by reducing the robot to a point and then growing the other objects in the scene accordingly.

Each robot has means for acquiring input information and planning its motion. For generality, we assume they those are specific for each robot (this may correspond to different motors, sensors, sampling rates). The motion control and sensing models are as follows.

Motion control. A robot is capable of *translation* only. The reason for this important assumption is simple – it reduces the planning problem to a two-dimensional, rather than three-dimensional, C-space. This may be unacceptable in some applications and acceptable in others. For example, in tasks where the robots can be considered roughly circular, or where passages between obstacles are wide enough, robot's steering (rotation) control would be independent of the global path planning algorithm, and so a two-dimensional model would be adequate. As is usual in computer controlled systems, planning and control are done in small steps defined by the robot's sampling rate, resulting in continuous motion. Typical sampling rates (e.g. in commercial mobile robots) are in the range 20 to 50 per second. Accordingly, each robot R_i is said to move in discrete steps of *step size* s_i. Step sizes may differ from robot to robot, $s_i \neq s_j$. A *step cycle* t_i of robot R_i is the (constant) time it takes R_i to perform sensing, planning and physical execution of a single step[1]. Step cycles may differ, $t_i = kt_j, i \neq j$, k – integer. For example, if the sampling rate of robot R_i is 20 (that is, $t_i = 50$ $msec$) and the robot's velocity is 1 m/sec, then it's step size $s_i = 5$ cm.

Given the Start (S) and Target (T) positions of robot R, its desirable path to T, called the main line or *M-line*, is defined as the straight line connecting S and T. In our algorithm the robot will move along its M-line towards its target until it is forced off the M-line due to a potential collision

[1]The sampling rate depends on the nature of sensing, computational resources, and the algorithm's computational complexity. The former two are functions of technology: today, even some vision-based systems (perhaps the most computation-taxing sensing medium) are already able to do at least simple planning in real time.

with an object. The latter may be a stationary obstacle, another robot, or a combination of both. The point where the robot abandons the M-line is called the *hit point, H*. A *local direction*, either left or right, determines the direction for passing around an object; it is decided upon beforehand. When a robot is passing around an object in a given local direction, it is said to be *following its boundary*. The object whose boundary the robot is following is called the *contact object*. The *leave condition* is the condition which, when satisfied, causes the robot to abandon the object whose boundary it has been following and resume its course towards its target. The distance between R and object O_j during the boundary following is chosen independent of the algorithm, based on such parameters as the robot's step size, the expected motion of O_j, and the desired safety margin.

Input information. Robot input information comes from its sensors. Robot R is said to *interact* with a visible object V_i if R can potentially collide with V_i within the current step cycle. Given our emphasis on formal algorithmic issues – the topological/geometric control scheme, convergence, completeness – we assume perfect sensing and accurate position information; no sensor inaccuracies are considered (a rather common liability inalgorithmic work, see e.g. [3, 5, 8], and also a discussion in Section 6).

Given the continuously changing world they live in, the robots will not attempt to build maps or maintain other large pieces of data that they acquire during their motion. Each robot's input information includes:

– its current coordinates and those of its target,
– the value of *step upper bound* s_{max} for all other robots,
– the current sensing information.

Current coordinates (current position) of the robot are the corresponding cartesian coordinates of one of its points – say, the center of mass. At a given instant t, C_t is the robot's current position.

Step upper bound, s^i_{max} (or simply s_{max}) is a predefined value known to each robot R_i beforehand: it is the upper bound of the maximum distance that any other robot $R_j, j \neq i$, can cover in any direction within the step cycle t_i. Note that s^i_{max} may differ from robot to robot, and that it may correspond to more than one step of robot R_j. The notion of the step upper bound represents the idea of a "reasonable behavior" mentioned above: in order to guarantee safe operation in an environment with other moving robots, each robot needs some expectation about the motion of the other robots it may encounter in the scene. In other words, whenever robot R_i sees another robot R_j, although it does not know R_j's objectives or step size, it assumes that R_j's displacement within the cycle time t_i will be no more than s^i_{max}.

Sensing. Each robot is equipped with *range sensors* which are capable of:

- Measuring distances to any objects within the robot's *range of sensing* r_v (stands for "radius of vision"); r_v may differ from robot to robot.

- Performing a *scanning operation* so as to construct an outline of the portions of the objects within its range of sensing and arriving at the set of *visible objects* V_j, see Figure 1b.
- Distinguishing between a robot and a (stationary) obstacle, and between two robots.
- Measuring the *instantaneous velocity* of any other robot within the robot's sensing range.

Two comments on the sensing capabilities:

1. The physical devices that provide these sensing capabilities are not discussed here. For example, distinguishing between a stationary object and a robot may be tricky; this capability is necessary, though, and is common in nature: confusing a building with a truck that happen to stand still at the moment would be dangerous for a robot, as it would be for a human.

2. The need to measure instantaneous velocity of moving objects is quite basic: it comes from the nature of the problem rather than from the algorithm requirements. To pass around moving objects, one needs to assess their velocity. The reason this ability is not common in robotics is that until now roboticists rarely considered dynamically changing environments and decentralized planning. Just about all creatures in nature have this ability, and so do some technical systems (odometers, radars etc.).

3 The approach

Overview. Our algorithm will make use of maze-searching techniques. With those, a point robot can purposely move in an unknown environment with stationary obstacles. There is a number of such algorithms available (see, e.g., [9, 10]); in principle, any one can be used for our purpose. For specificity only, we use below the algorithm called Bug2 [9]. Define ST as the M-line; define the local direction (left or right); briefly, Bug2 works as follows:

1. Move along the M-line until one of the following happens:
   (a) T is reached. The procedure stops.
   (b) An obstacle is encountered. Go to Step 2.
2. Using the accepted local direction, follow the obstacle boundary until one of these occurs:
   (a) The robot meets the M-line at a point between H and T that satisfies the leave condition. Go to Step 1.
   (b) The condition of target non-reachability is satisfied. The procedure terminates.

In our multi-robot case, the procedure that each robot uses is similar to Bug2. Each robot moves towards its target along a straight line until it encounters an object (another robot or an obstacle). It then follows the boundary of the object in the local direction until a certain leave condition is satisfied, and then resumes its course towards the target. However, since the environment is dynamic in nature, the following additional considerations have to be taken into account:

- In order to ensure collision-free motion, a relationship between the robot's step size and its range of sensing needs be established. It is clear, for example, that the robot should not make a step that would take it outside its range of sensing. Details of this relationship are examined in the next section.

- When robots R_i and R_j meet, each one has no information about the objectives or the step size of the other. Some protocol is therefore needed that robots could count on when trying to pass around each other and avoid collision. One possibility could be a beforehand agreement on the direction, left or right, of passing each other during the encounter. When walking toward each other, two people could avoid collision by each stepping, say, to his left, and then continuing the path. Unfortunately, this mechanism loses consistency in the interaction of more than two robots. The use of hyperplanes between the robots [11] would be another alternative for the protocol, but it is applicable only to convex robots and pairwise interaction. Two possible mechanisms described in Section 3.3 make use of the perpendicular bisector and the Voronoi diagram [14], respectively, and allow many robots to interact simultaneously and pass around each other in a meaningful manner and collision-free.

- It will be shown in Section 3.4 that the leave condition used in the Bug2 algorithm – meeting the M-line between points H and T – is not adequate for a dynamic environment. Accordingly, a new condition based on a *dynamic M-line* is introduced, and a procedure for M-line modification is developed.

3.1 Constraint on the step size

In sensor-based motion planning, irrespective of the nature of the environment (static or dynamic), the only information available to a robot for planning its motion is the data coming from its sensors. This means that in order to guarantee its safety the robot has to confine its step to some limited area within which it has complete knowledge. There is a simple relationship between the sensing range, the step size of the robot, and the expected motion of other robots within a single step cycle, that should be taken into account when planning collision-free motion. Starting with the simple case of stationary obstacles, the step size s of the robot must satisfy the condition:

$$s \leq r_v \tag{1}$$

The constraint is apparent from the fact that is the robot makes a step outside of its sensing range, it risks a collision with an obstacle it currently cannot sense. It is easy to see that condition (1) is not adequate if the environment includes moving objects. A better condition for this case, which is independent of the motion planning algorithm used, is given by the following simple statement:

The constraint is apparent from the fact that the robot cannot plan for obstacles that it has not sensed: making a step outside of the robot's range of sensing could lead to a collision. It is easy to see that condition (1) is inadequate if the environment includes moving objects. A better condition for this case, which is independent of the motion planning algorithm used, is given by the following simple statement:

Lemma: *For robot R to guarantee collision-free motion in an environment with moving objects, it is necessary that the following inequality be satisfied:*

$$s \leq r_v - s_{max} \tag{2}$$

where s_{max} is the step upper bound on the other moving objects within the step cycle of robot R.

Proof. Assume for a moment that $s > r_v - s_{max}$, and assume a (minimum) distance, $d_{min} = s + s_{max}$, between robot R and some moving object whose step size is s_{max}. Since $d_{min} > r_v$, the robot will not sense the object and will not account for it while planning its next step. If the robot and the object were to take steps towards each other so as to minimize the distance between them at the end of the step, a collision could result. Similarly, note that if $s < r_v - s_{max}$, a collision with an object that is not visible at the robot's current position will never take place within a given step cycle. **Q.E.D.**

3.2 Boundary following

If robot R is interacting with a visible object V_i, and V_i, is a stationary obstacle, then following its boundary in a given direction is a trivial operation. If, however, V_i is a robot, then robot R has no information about its step size or intended motion, and so the boundary to be followed is not defined.

Although robot R cannot predict the motion of object V_i precisely, it can estimate all possible motions that V_i can make within R's one step cycle. The estimate is based on the fact that the maximum step size of any object is s_{max}. The area formed by taking into account all possible motions of object V_i defines its *collision front.* The collision front E_i of V_i represents the union of all possible moves V_i can make within a given step cycle. Given a robot R and a set of visible objects $\{V_i\}$ in its sensing range, the method of constructing the corresponding collision front E is as follows (see Figure 2):

1. With the collision front E_i obtained for each visible object V_i separately, the final collision front E is the union of all individual fronts, $E = \cup E_i$. Let d_{min} be the minimum distance between R and V_i.

2. If object V_i is an obstacle, then depending on d_{min} do one of the following:

   (a) If $d_{min} > s$, then $E_i = \phi$: that is, V_i is too far off to affect robot R, and is ignored.

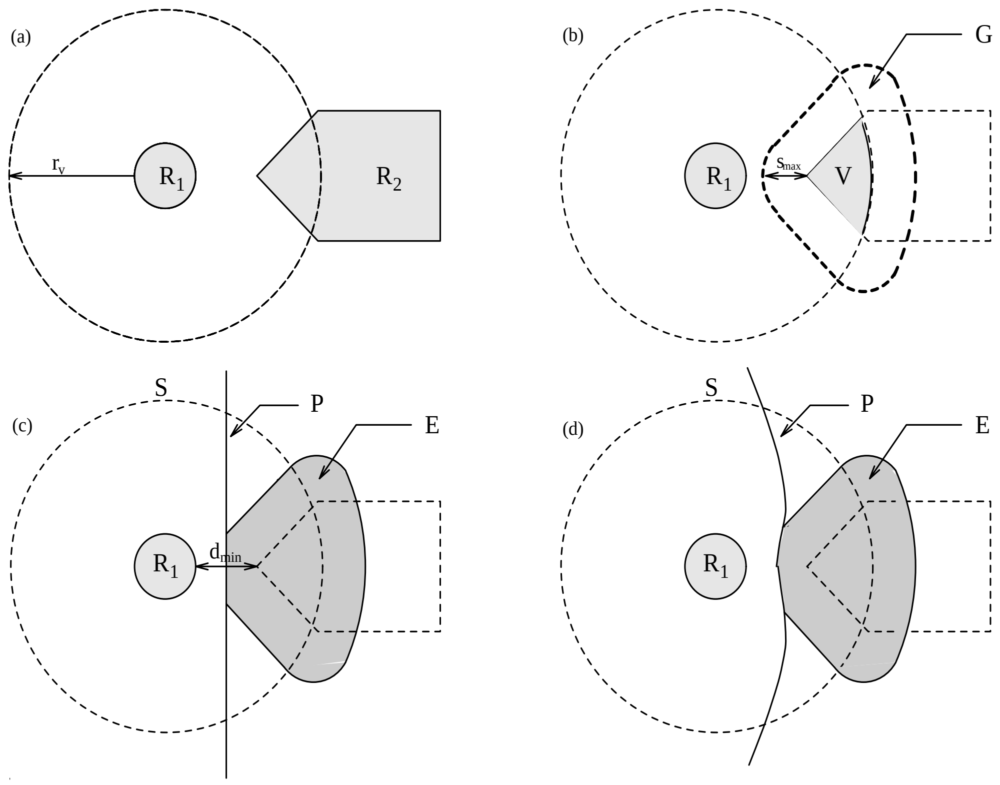

Figure 2: *Construction of a collision front. (a) Robot R_1, whose radius of vision is r_v, detects R_2. (b) V is the portion of R_2 that is within the range of R_1. G is V grown by s_{max} in all directions. (c) Perpendicular bisector method: P is the perpendicular bisector drawn to the line of minimum distance d_{min}. S is the robot's side of the semi-plane formed by P. The collision front (shaded) is $E = G - (G \cap S)$. (d) Voronoi diagram method: P represents the Voronoi curve between R_1 and V (Any point on P is equidistant to R and V). S is the robot's side of the semi-plane formed by P. The collision front (shaded) is $E = G - (G \cap S)$.*

(b) If $d_{min} \leq s$, then $E_i = V_i$.

3. If object V_i is a robot then depending on d_{min} do one of the following:

(a) If $d_{min} > s + s_{max}$, then $E_i = \phi$: V_i is ignored as it is too far off to affect the robot.

(b) If $d_{min} \leq s + s_{max}$, then grow V_i by s_{max} in all directions, to obtain the "grown" region G_i, see Figure 2b. Region G_i thus represents the worst case of all possible motions V_i can make. However, if what we called a "reasonable behavior" and a corresponding protocol is assumed, region G_i can be further reduced so as to give R more latitude for motion. Note that such a protocol assumes a prior agreement and requires no explicit communication between the robots. Two procedures for reducing area G_i are considered:

The perpendicular bisector method: Let P be the perpendicular bisector of the line of minimum distance between R and V_i dividing the workspace into two semi-planes. Let S be the robot's side semi-plane of the bisector P. The portion of G_i in S,

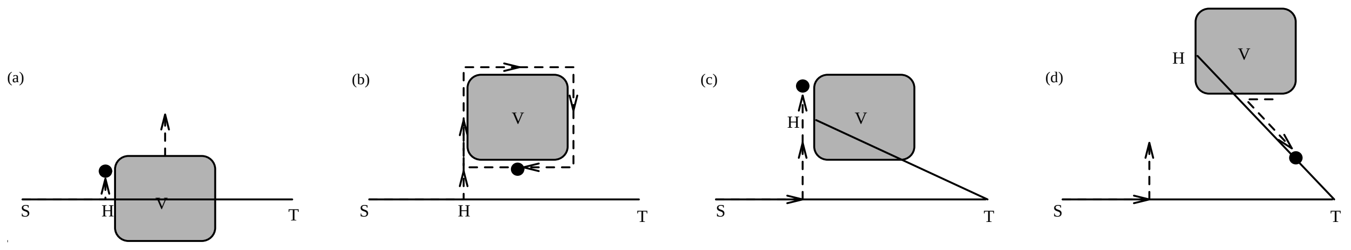

Figure 3: *The argument for a dynamic M-line. (a) While moving from S to T, a point robot R (shown as a small black disc) encounters object V. After defining the hit point H, R starts moving around V using left as the local direction. (b) While R is going around V, V moves to the new position. Under the Bug2 algorithm, this would cause R to go around V indefinitely without ever meeting the M-Line. Dynamic M-line: (c) At each step, point H (and thus the M-line HT), is shifted by the corresponding vector traversed by V within this step. (d) Eventually R meets the current line HT and proceeds towards T. (The entire path is not shown). For the sake of simplicity, direct contact between R and V is shown.*

$G_i \cap S$, is then excluded from G_i to obtain the collision front E_i, $E_i = G_i - (G_i \cap S)$, see Figure 2c. (For nonconvex objects, a more complex recursive procedure would be needed).

The Voronoi diagram method: Given a robot R and a visible object V_i, the Voronoi diagram presents a skeleton P, a curve defined by the locus of points equidistant to R and V_i, see Figure 2d. As in the perpendicular bisector method, the Voronoi curve P divides the workspace into two generalized semi-planes, and the portion of G_i on the robot's side of P is excluded from G_i to obtain the collision front E_i, $E_i = G_i - (G_i \cap S)$. The Voronoi diagram of the set (R, V_i) is unique [14].

Due to its simplicity, the calculation of the collision front carries low computational burden and can be easily done in real time. In applications the procedure can be simplified even further: at every step only the small area of the collision front around the robot's position is contemplated for the next step. In the algorithm, after defining the collision front E, the robot will follow the boundary of E. Since the described method for constructing the collision front guarantees that no collisions take place (as no candidate locations for two robots can overlap), this assures collision-free motion. As the environment is dynamic, the collision front computation is done at every step.

3.3 The leave condition

The leave condition for the Bug2 algorithm requires the robot to meet its M-line at some point between H and T [9]. In a dynamic environment, however, this leave condition in general will not work. While following the boundary that is shifting in time, the robot may be "dragged" away from the M-line and never meet it again, causing the robot to go around the boundary indefinitely,

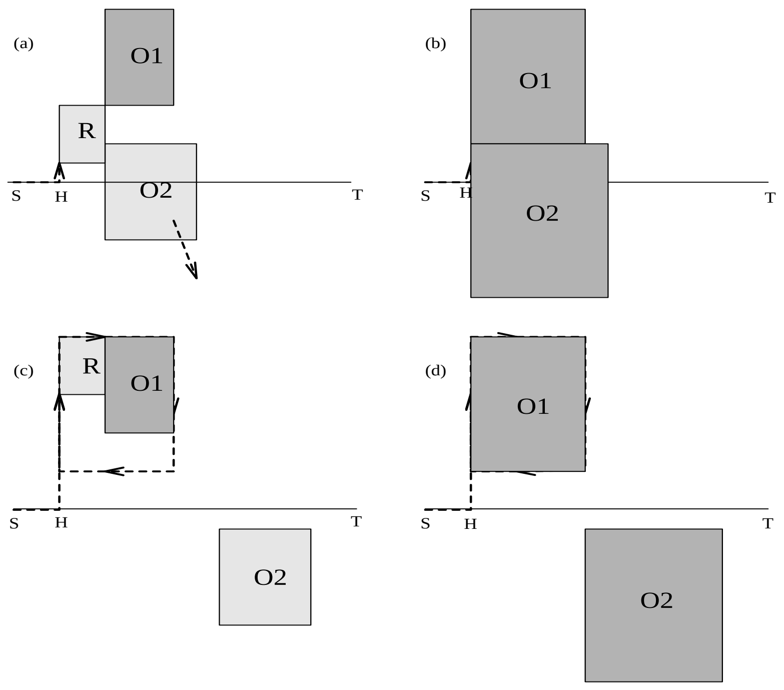

Figure 4: *In this example, an attempt to follow a fixed M-line would result in an infinite loop. (a) In the workspace, robot R tries to reach its target T from S. Object $O1$ is a stationary obstacle, object $O2$ is a mobile robot. After encountering $O1$ and $O2$, R uses the local direction left to negotiate around them. (b) The C-space representation of $O1$ and $O2$; here, robot R becomes a point, and $O1, O2$ grow accordingly and are 'fused' together, appearing as a single obstacle. (c) Once R loses contact with $O2$, the latter moves away thus 'splitting' the C-space obstacle into two. As a result, R goes around $O1$ indefinitely as it is unable to reach the M-line. (d) The C-space representation of R continually looping. Again, direct contact between R and obstacles is shown for simplicity.*

see Figure 3a,b. In order to overcome this problem of losing the M-line, we introduce the notion of a *dynamic M-line*. Namely, when a hit point H is defined at a contact object V, the M-line is re-defined as the line HT. Since the robot is capable of measuring the instantaneous velocity of the object V, at every step the robot compensates for the motion of V by shifting (the invisible but known) H by the corresponding distance traversed by V, see Figure 3c. The hit point H can be thought of as being "attached" to the object V at the first point of contact. The M-line is re-defined as HT at every step.

Another problem that arises due to the dynamic nature of the environment is that two objects that are simultaneously in contact with robot R may split later, causing the robot to go into an infinite loop, as illustrated in the example in Figure 4. In this case the C-space representation of the obstacle that the robot is following is altered without the robot being aware of it. To overcome this difficulty, the robot defines a new hit point every time it meets a new contact object (which it recognizes using its ability to distinguish between a robot and a stationary obstacle, and between two robots). These rules for modifying hit points are summarized in the next section.

3.4 Modification of the hit point

The rules for modifying the hit point H apply only to the case when the contact object(s) (or, rather, the collision front) whose boundary robot R is following at the time involves at least one moving object, that is another robot. No modification is needed otherwise. At a given step, the rules are as follows:

1. When passing around a robot R_i, move H by the distance traversed by R_i during one step; define the M-line as M-line=HT.
2. When switching contact from a robot to a stationary obstacle or from a stationary obstacle to a robot, use the current position C to define the new hit point $H = C$, and define the M-line as M-line=HT.

4 The algorithm

We are now ready to formulate the final algorithm for sensor-based motion planning in an environment with multiple mobile robots. The algorithm consists of two procedures, one covering the motion along the M-line (Step 1), and the other for moving off the M-line (Step 2). All robots execute the algorithm in parallel, independent of each other. Assume that initially robot R is at point S, $C = S$ (Current position = Start), and it attempts to reach point T. Set the M-line of R as CT.

1. Move along the M-line until one of the following occurs:
   (a) T is reached. The procedure stops.
   (b) Object(s) O_i appears within the robot's range of sensing. If the collision front is $E = \phi$ or if the next intended position of robot R is outside of E, iterate this step. Otherwise, turn in the local direction to follow the object's boundary, and go to Step 2.
2. At every step, follow the boundary of the collision front, while modifying the hit point H and the M-line according to the rules of Section 3.5, until one of the following occurs:
   (a) T is reached. The procedure stops.
   (b) The robot meets its M-line within the segment HT satisfying the leave condition (Section 3.4). Go to Step 1.
   (c) The robot loses contact with the boundary, $E = \phi$. Set the M-line = CT. Go to Step 1.

5 Examples

The computer simulations shown below have been carried out in real time. That is, after one creates the robots, the scene (obstacles), and indicates the intended target positions, one presses the button, and the whole animation of the motion unfolds in front of one's eyes. Given the complexity of these examples, the feasibility of real-time computation is remarkable. The source of this high performance is in the algorithm's exploiting the advantage of the decentralized control – only local data processing is done.

Given the nature of computer simulation, the task and robot parameters defined in Section 2 – step sizes s_i and s_{max}, radius of vision r_i, step cycle time t_i, object dimensions, distances in the scene – are relative values. For "myopic" sensing (e.g. Figure 5), the step size has been such as to fit 2-3 steps into r_v; in the case of better sensing (see e.g. the relative size of the sensing range in Figure 6), it corresponded to 10-20 steps per r_v. The sensing ranges tested varied widely; those shown seem to be realistic enough – in Figure 5 it would correspond to a short-range infrared proximity sensor, in Figure 6 – to a sonar or laser range sensor. The cycle time varied widely as well, including more realistic 10 to 30 cycle/sec rates.

In the first example, Figure 5, two robots operate in an environment with two stationary obstacles. Both robots have a very short sensing range. Figure 5a shows the robots in their starting positions. The line connecting Si and Ti is the desired path (the initial M-line) of robot Ri. Right before the situation shown in Figure 5b, the robots see each other (not necessarily simultaneously) and attempt to pass around each other using left as the local direction. This turns out to be impossible because of the stationary obstacles $O1, O2$, and both robots embark upon rather long detours, Figure 5c. [Clearly, some coordination between the robots would help here]. The resulting paths are shown in Figure 5d.

Figure 6 shows what happens in the same environment when the robots are enabled with better sensing. The sensing ranges for $R1, R2$ are shown by the two bars on the left, 1 and 2, respectively. The situation becomes more complex: both robots see each other much in advance, and for a while do not see a need to change their paths. Then, sometime before the situation shown in Figure 6b, $R1$ infers that $R2$ will be blocking its path to $T1$, and embarks upon a detour around $O1$; this clears way for $R2$ which continues along its path, Figure 6c. Note that the resulting paths, Figure 6, are smoother, shorter, and quite different from those in Figures 5.

Examples shown in Figures 7 and 8 are more complex. Five robots, $R1, ..., R5$, operate in an environment with three stationary obstacles $O1, O2, O3$. In Figure 7 all robots have very short sensing range; their velocities differ from each other; robots $R1, ..., R4$ are of simple geometric shape, $R5$ has a more complex nonconvex shape. Each robot has no information about other robots and obstacles until it senses them. Robots do not store their paths or outlines of other objects that they encounter – only very limited information, such as recent intersection points between the paths and objects is stored (see Section 3).

Given the multiplicity of objects in the scene, intended paths, and robots' inferior sensing, Figure 7a, mutual interference involving combinations of obstacles and robots is likely. Indeed, intersections between the intended paths lead to a temporary crowding in the middle of the scene, Figure 7b. Robots $R2$ and $R3$ are first to finish their motion, Figure 7c. Figure 7d shows the complete paths, with the robots in their final destinations.

Figure 8 relates to the same set of objects and tasks, except the robots have better sensing: the sensing ranges for robots $R1, ..., R5$ are shown by the bars on the left, 1 through 5, respectively. With the bigger radius of sensing here, each robot has more information in advance, and so the likelihood of crowding is reduced. Paths become smoother, and also more complex, in the sense that it is often hard to understand the "rationale" behind the planning decisions. Note that as long as some information is still missing, although better sensing does on the average result in better paths, it cannot guarantee it: for example, the path of robot $R4$ turned out to be shorter when it had "myopic" sensing, Figure 7d, than when it had a wider sensing range, Figure 8d.

6 Discussion

The problem of multi-robot decentralized motion planning is formulated in this work as a maze-searching problem, albeit in a dynamically changing "maze". There is no communication between the robots; each of them knows about the other(s) only when it see it. The main emphasis is on the algorithmic issues of decentralized decision-making. Consequently, as mentioned before, a number of multi-agent control issues, such as learning or collective behavior, are not being addressed; also ignored are real life uncertainties of input information – we assume precise knowledge of robots' and their target' positions, and perfect sensing.

Real physical sensors being what they are, they are of course a source of various errors. Some of those depend on the technology used; some may accumulate with time – as e.g. in registration systems based on dead reckoning; some may depend on the robot's surroundings and position in space – e.g., compass readings get worse near iron masses. Proper handling of those errors in a real system is very important. Note, however, that the necessary measures are likely to be independent of and can be separated from the planning algorithm design process. For example, if the error in distance reading is within 3 robot steps, it would be logical to use a safety margin that is at least 3 steps wide. A more complex example with a real physical system can be found in [15].

The two examples in Section 5 have been simulated on one computer, in real time, producing an animated "movie". Given the decentralized character and independent decision-making, the same software could be used to simulate the operation on multiple computers, one per simulated robot, or to implement motion planning on real robots. Given this parallelism, although from our standpoint the second example is more complex than the first, from the standpoint of each robot the computational complexity in both examples is the same.

This advantage of decentralized motion planning is also the source of its drawback – in spite of the fact that the underlying maze-searching algorithm does guarantee convergence, it cannot be guaranteed anymore for our multi-agent algorithm. As mentioned before, the loss of convergence is not a matter of a good or bad algorithm – it is due to the decentralized control model. This is easy to show: in the example in Figure 10 (taken from [5]) each of the robots R_1, R_2 is required to reach its respective target T_1, T_2. The task is clearly impossible unless the motion of both robots is closely coordinated in a centralized manner.

To understand the frequency of such unfortunate situations, consider a notion of algorithm robustness, defined, say, by the frequency of cases when because of the "jams" one or more robots do not reach their (otherwise reachable) targets in a randomly generated environment. Since today there are no accepted ways for defining statistical distributions of geometric shapes [16], setting up a rigorous statistical test here is difficult. Simple statistical tests with "reasonable" shape/position distributions show, however, that in randomly generated complex scenes the described algorithm is remarkably robust [17]. Note also that the complexity of generated motion in the example in Figure 8 seems already to be beyond the human ability for space reasoning and centralized control (with complete or with incomplete information). One would also find it difficult to contemplate such control when observing this motion in real-time animation.

References

[1] V. Scheinman, Robotworld: A multiple robot vision guided assembly system. *Proceedings of the International Sysmposium on Robotics Research*, 1987.

[2] K. Kant, S. Zucker, Towards efficient planning: the path-velocity decomposition. *International Journal of Robotics Research*, 5: 72-89, 1986.

[3] M. Erdmann, T. Lozano-Perez, On multiple moving objects. *Algorithmica*, 2: 477-522, 1987.

[4] S. Buckley, Fast motion planning for multiple moving robots. *Proceedings of the IEEE International Conference on Robotics and Automation*, 322-326, 1989.

[5] J. Schwartz, M. Sharir, On the Piano Movers problem - III. Coordinating the motion of several independent bodies, *International Journal of Robotics Research* 3, 1983.

[6] C. Yap, Coordinating the motion of several discs. TR-105, Courant Institute of Mathematical Sciences, New York University, New York, 1984.

[7] P. Spirakis, C. Yap, Strong NP-hardness of moving many discs.*Information Processing Letters*, 19: 55-59, 1984.

[8] J. Hopcroft, J. Schwartz, M. Sharir, On the complexity of motion planning for multiple independent objects: PSPACE hardness of the 'warehouseman's problem'. *International Journal of Robotics Research 3*, 3(4), 76-88, 1984.

[9] V. Lumelsky, A. Stepanov, Path planning strategies for a point mobile automaton moving amongst unknown obstacles of arbitrary shape, *Algorithmica*, 3(4): 403-430, 1987.

[10] H.Abelson, A.diSessa, *Turtle geometry.* MIT Press, Cambridge, MA, 1981.

[11] P. Tournassoud, A strategy for obstacle avoidance and its application to multi-robot systems. *Proceedings of the IEEE International Conference on Robotics and Automation*, 1224-1229, 1986.

[12] S. Mahadevan, J. Connell, Automatic Programming of Behavior-based Robots using Reinforcement Learning, *Artificial Intelligence*, vol. 55, Nos. 2-3, pp. 311-365, 1992 .

[13] M. Mataric, Interaction and Intelligent Behavior, PhD Thesis, MIT, Dept of Electrical Engineering and Computer Science, AITR-1495, 1994.

[14] F. Preparata and M. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.

[15] E. Cheung, V.J. Lumelsky, Proximity Sensing in Robot Manipulator Motion Planning: System and Implementation Issues, *IEEE Journal of Robotics and Automation*, Vol. 5, No.6, 740-751, 1989.

[16] *Stochastic Geometry*, (E. Harding and D. Kendall, Eds), John Wiley and Sons, London-New York, 1974.

[17] V. Yegorov, Statistical Analysis of Sensor-Based Motion Planning Algorithms, MS Thesis, Dept of Computer Science, University of Wisconsin-Madison, 1996.

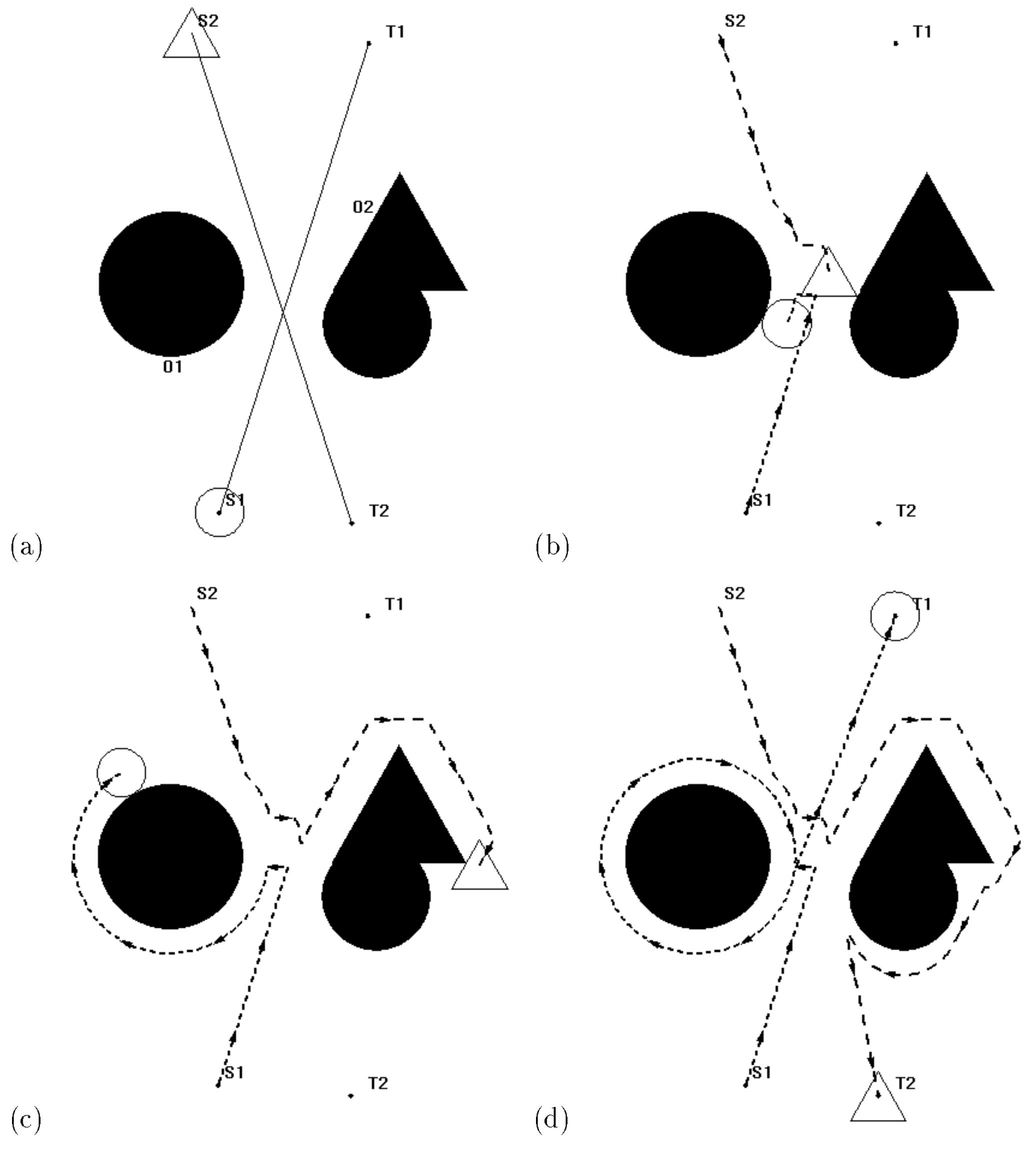

Figure 5: *Example of the algorithm performance. While attempting to reach their target positions T_1, T_2, robots $R1$ and $R2$ have to negotiate their paths around each other, and possibly around stationary obstacles $O1$ and $O2$. Each robot has very "myopic", short range sensing: (a) starting configurations and intended paths, (b),(c) intermediate positions, (d) final positions.*

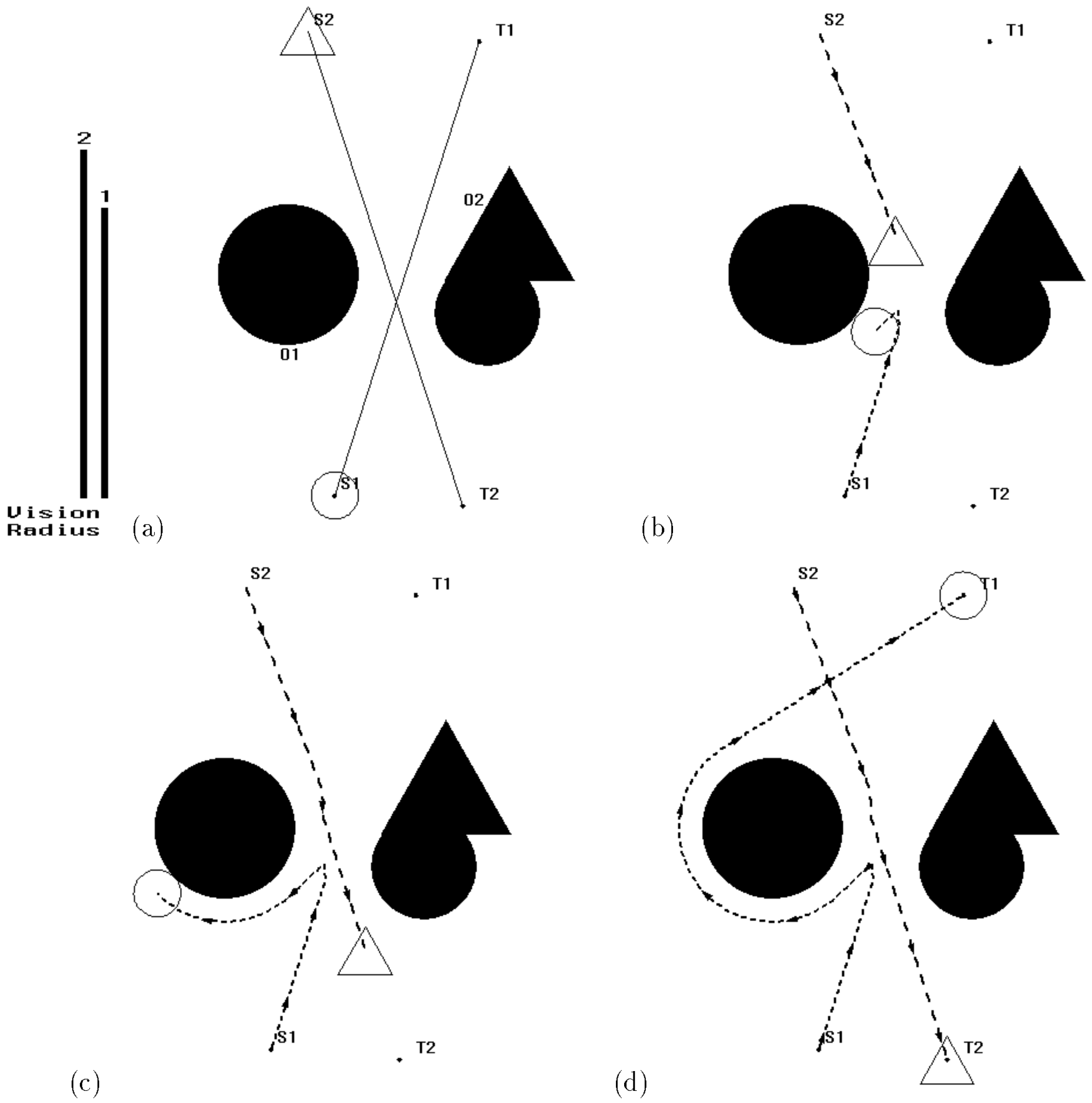

Figure 6: *Same example as in Figure 5, except both robots have a better sensing; the sensing ranges for $R1, R2$ are shown by the two bars, 1 and 2, respectively. Note the improvement in the path length performance compared to Figure 5.*

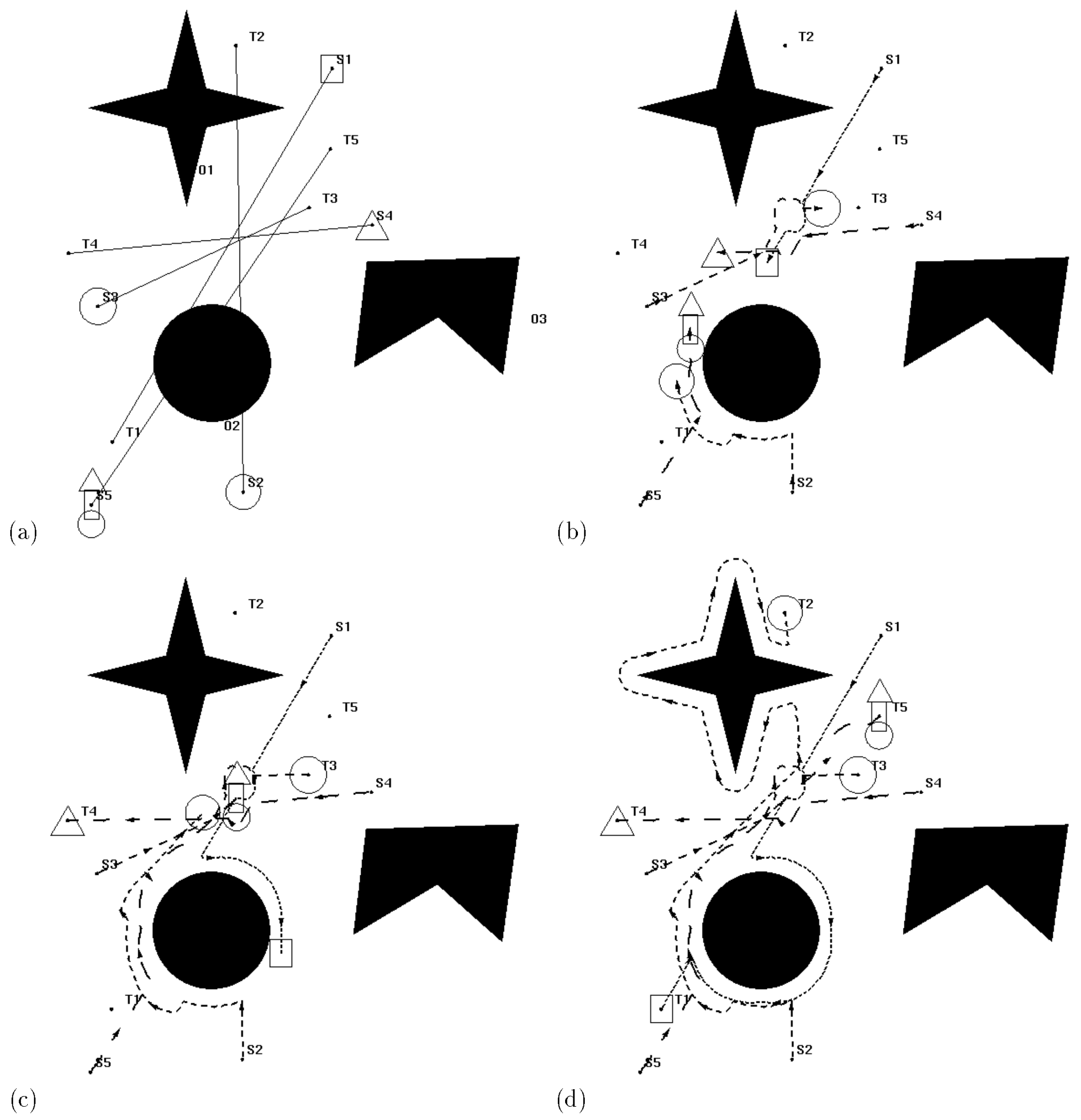

Figure 7: *An example with five robots, R1, ..., R5, operating in an environment with three stationary obstacles, O1, O2, O3. The robots are capable of only simple very short range sensing. Si and Ti are the robot Ri start and target positions. (a) Starting configuration and the intended paths; (b),(c) intermediate positions; (d) final configuration.*

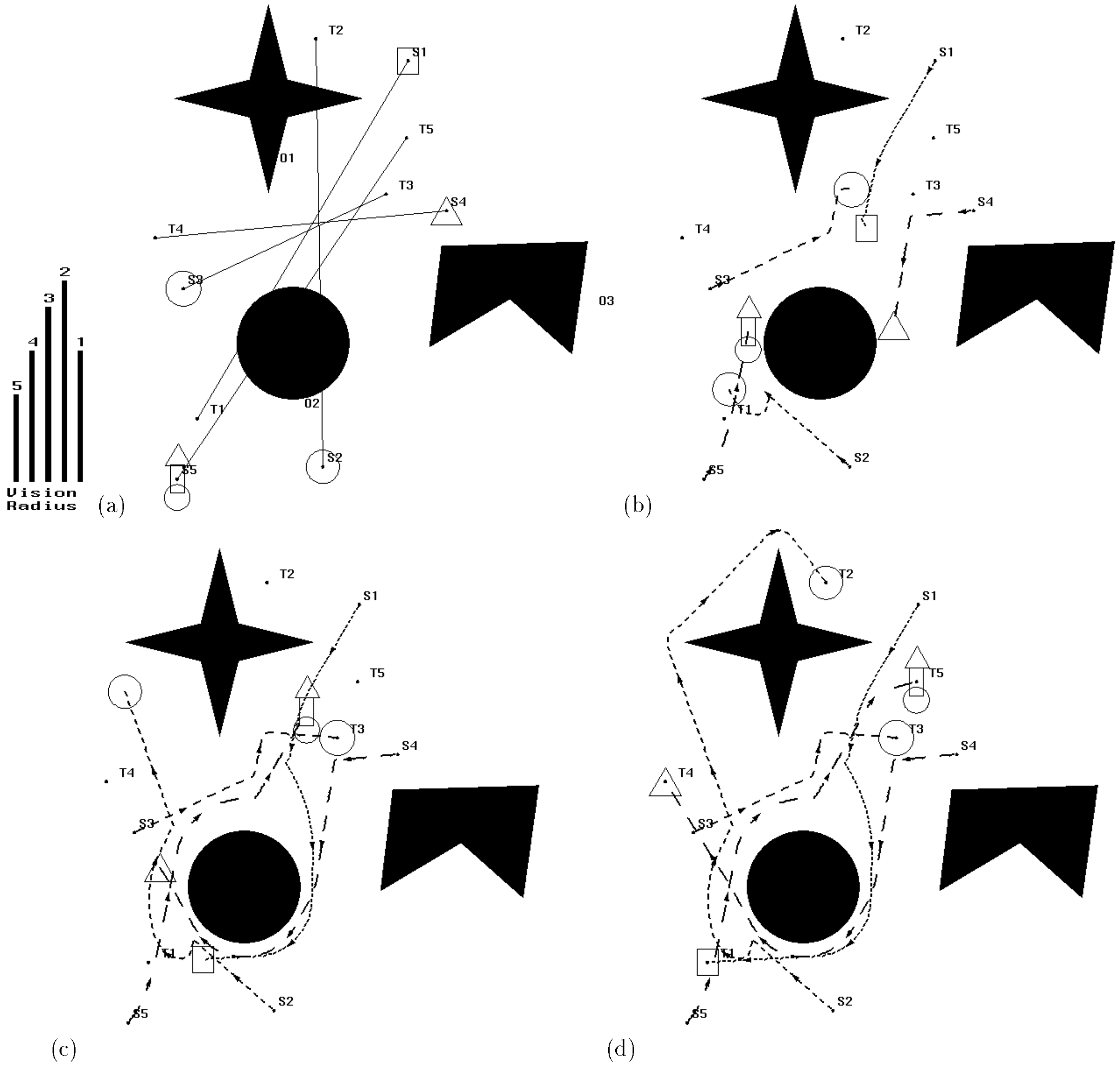

Figure 8: *Same example as in Figure 7, except all five robots are provided with better sensing; their sensing ranges are shown by the bars on the left, 1 through 5, respectively. Note the complexity of the interaction in this decentralized system.*

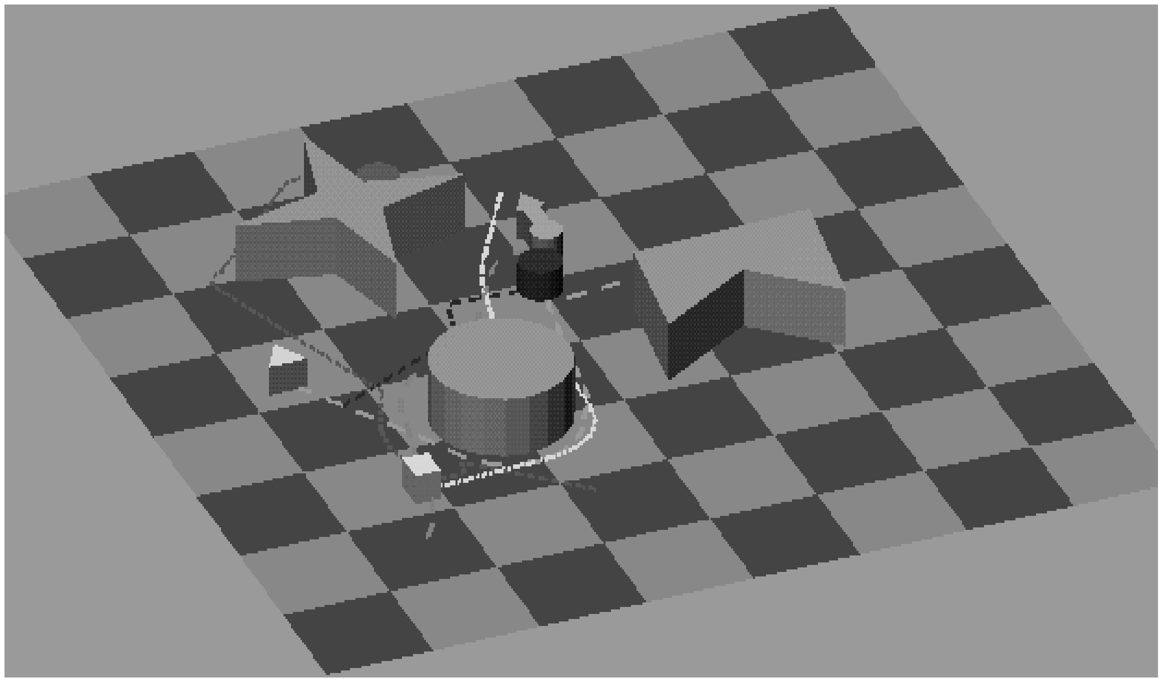
Figure 9: *A pseudo three-dimensional presentation of the situation shown in Figure 8d*

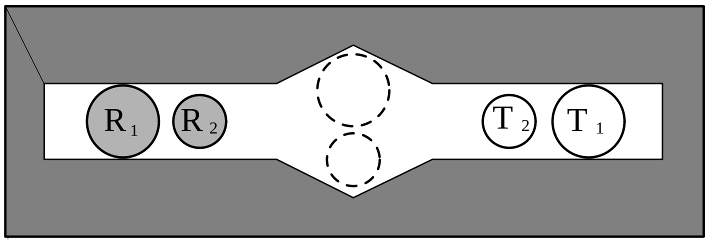

Figure 10: *Here, in order for the circular robots R_1 and R_2 to reach their respective targets, their relative positions need to be switched. The only way to do so is to move R_2 into one of the 'wedges' and then move R_1 through the other wedge. With decentralized motion planning this is clearly a futile task, as advance information about the wedges and close coordination are necessary to execute the task.*