Dynamic Generation of Subgoals for Autonomous Mobile Robots Using Local Feedback Information

BRUCE H. KROGH, MEMBER, IEEE, AND DAI FENG, STUDENT MEMBER, IEEE

Abstract—**An algorithm is presented for using local feedback information to generate subgoals for driving an autonomous mobile robot (AMR) along a collision-free trajectory to a goal. The *subgoals selection algorithm* (SSA) updates subgoal positions while the AMR is moving so that continuous motion is achieved without stopping to replan a path when new sensor data become available. Assuming a finite number of polygonal obstacles (i.e., the internal representation of the local environment is in terms of a 2-D map with linear obstacle boundaries) and a dynamic steering control algorithm (SCA) capable of driving the AMR to *safe* subgoals, it is shown that the feedback algorithm for subgoal selection will direct the AMR along a collision-free trajectory to the final goal in finite time. Properties of the algorithm are illustrated by simulation examples.**

I. INTRODUCTION

A MAJOR objective of current robotics research is to develop autonomous mobile robots (AMR's) which can navigate in unstructured environments [1]–[4]. In this paper we present a feedback algorithm for selecting subgoals to steer an AMR to a goal along a collision-free path. The subgoals are chosen while the robot is moving using feedback information about the locally visible environment. The objective of the proposed subgoal selection algorithm is to close to the higher level feedback loop in real time so that collisions with unanticipated obstacles can be avoided without stopping to replan the path to the goal.

There has been a considerable amount of research on path planning for AMR's. Many of the proposed algorithms require a map of the entire region to be navigated and give no consideration to kinematic and dynamic constraints [5]–[9]. Application of these algorithms for autonomous navigation requires a "stop-look-and move" approach to incorporating new sensor information [2]. In such a scheme, the computer uses the most recent map of the environment to plan a collision-free path to the goal while the AMR is stopped. The AMR moves a short distance along this path and then stops to update the map. While the AMR is stopped, the computer modifies the original path, taking into account new information from the sensors. The robot then moves a short distance along the updated path and the cycle is repeated until the robot reaches the goal. This is essentially an open-loop feedback approach to the problem of autonomous navigation.

The algorithm proposed in the present paper offers an alternative to stop-look-and-move schemes for autonomous navigation in unstructured environments. The subgoal selection algorithm (SSA) uses the most recent sensor data to generate subgoals while the AMR is moving. The current subgoal is pursued by a steering control algorithm (SCA) which takes into account the kinematic and dynamic operating constraints for the AMR. The SCA is designed to drive the AMR toward the current subgoal along a collision-free path in the local obstacle-free space. The subgoal is updated whenever the next subgoal generated by the SSA can be safely pursued by the SCA (the concept of safe subgoals is defined in Section II).

The proposed algorithm is in the spirit of so-called *reflexive* control schemes for autonomous navigation in that the subgoals and resulting trajectory are generated "on the fly" as the local environment becomes visible. Most of the research in reflexive control for AMR's has focused on hardware and software architectures for sensor fusion and intelligent navigation using *a priori* information [10]–[13]. Although the integration of *a priori* information and higher level planning into an overall navigation system is essential for practical applications, the purpose of the present paper is to focus on the navigation problem in domains where no *a priori* information is available and to demonstrate rigorously that a collision-free, finite-time trajectory to the goal can be generated using only local feedback information.

Among the many papers which have appeared in the area of path planning, our problem formulation and basic objective are most similar to the work of Lumelsky [14], [15]. Lumelsky considers the navigation problem for a *point mobile automation* using only *tactile* sensing, that is, the sensors only indicate whether the system (AMR) is in contact with an obstacle. The most significant ways in which our problem formulation differs from Lumelsky's work are as follows: 1) we assume that proximity sensors provide a map of the locally visible obstacles (within, perhaps, a finite range); 2) the system dynamics are explicitly taken into account by the SCA and the notion of safe subgoals; and 3) obstacles are modeled as convex polygons. The latter assumption makes our algorithm less general than Lumelsky's with respect to the model of the environment. However, our assumptions about the AMR sensing capabilities and dynamics lead to trajectories which are more acceptable for actual AMR's, such as the autonomous land vehicle at Carnegie-Mellon University [16].

The basic approach of the SSA is as follows. If an obstacle lies between the AMR and the final goal, subgoals are chosen near the obstacles vertices as they become visible, guiding the AMR around the obstacle until it does not block the line-of-sight to the final goal. Further subgoals are generated when there is another obstacle between the AMR and the goal, or when a previously generated subgoal is not visible due to another obstacle. As subgoals are generated they are stored until they can be safely pursued by the SCA. The algorithm also saves information about the extreme edges of obstacles which have been circumvented to avoid cycling indefinitely around the final goal.

Two aspects of our approach contribute to the complexity of the convergence proof for the SSA. First, we make no assumptions about the particular path followed by the AMR. Thus, no assumptions can be made concerning the vantage points from which the local obstacle maps are generated. Second, the subgoals generated by the SSA are not necessarily reached by the AMR because the subgoal being pursued can be updated while the AMR is moving. Thus, although the subgoals are associated with obstacle vertices, the convergence proof cannot rely exclusively upon the topological properties of the obstacle surfaces. With respect to the general problem of autonomous navigation, the

Manuscript received July 2, 1987; revised June 7, 1988. Paper recommended by Past Associate Editor, S. S. Sastry. This work was supported in part by the National Science Foundation under Grants ECS-8404607 and DMC-8451493, by the Ben Franklin Partnership Program, Commonwealth of Pennsylvania, and by the Robotics Institute, Carnegie-Mellon University.

The authors are with the Department of Electrical and Computer Engineering, Carnegie-Mellon University, Pittsburgh, PA 15213.

IEEE Log Number 8926481.

0018-9286/89/0500-0483$01.00

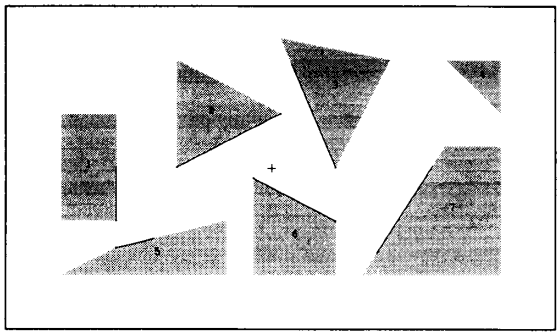


Fig. 1. Illustration of local obstacle map (bold lines) from the position $p_k(+)$.

present work addresses the problem of local guidance through a field of obstacles. Our objective is to explore the interface between navigation decisions and dynamic steering of an AMR. Consequently, we restrict our model of the environment to include only convex polygonal obstacles. Extending our algorithm to handle more complex environments is a topic for future research.

In the following section we formulate the subgoal selection problem and introduce terminology, notation, and assumptions used in the remainder of the paper. In Section III we present the SSA and describe each step in the algorithm. Section IV contains the analytical results which establish the convergence properties of the algorithm. The performance of the algorithm is demonstrated with simulations in Section V using a point mass model of the AMR and a previously proposed SCA based on artificial potential fields [17], [18]. In the concluding section we discuss directions for future research. As a convenience to the reader, the Appendix provides a summary of the notation, parameters, and functions defined in the text.

II. Problem Formulation

We model an AMR as a point on a plane which is to be steered along a continuous path from an initial position p_0 to p_g, the *final goal.* The space to be navigated contains a finite number of obstacles which are modeled as convex polygons in the plane. We assume that the minimum distance between points on any two obstacles is greater than a given parameter κ, which is greater than the physical width of the AMR. This assures that there exists a feasible path for the AMR from p_0 to p_g. In this paper we deal only with a point model of the AMR. The finite dimensions of the AMR are easily taken care of by appropriately "growing" the obstacles in the local maps to account for the AMR dimensions.

We assume that on-board sensors provide a local map of the environment which is *visible* from the AMR at discrete points in time. Letting p_k denote the AMR position at the kth iteration of the SSA, the faces of obstacles which can be detected from the position p_k are modeled by line segments in a two-dimensional *local map.* At the kth execution of the SSA, the local map consists of a set $O(p_k)$ of line segments representing obstacle faces which are visible from p_k. A local map is illustrated in Fig. 1 where the bold lines indicate the faces of obstacles which are visible from the AMR position p_k. We note that since the AMR is not necessarily stopped, the position vector p_k and the local map reflect the situation at the time the sensor data are taken. The processing of the sensor data will take a finite amount of time, which means the state of the AMR at the point when the SSA is executed is different from the time at which the local map was produced. As we shall show below, this time delay is accounted for by the evaluation of the safeness of subgoals.

In general, an AMR is a complex dynamic system. We denote its dynamic state including position at time t by $x(t)$ and assume that there is a dynamic SCA which can drive the AMR to specified points in the plane provided certain conditions are satisfied. The minimum requirement is that SCA be capable of driving the AMR along a finite-time, collision-free trajectory from rest at an initial point to stop at any point visible from the initial point. If this is all the SCA is capable of doing, our subgoal algorithm will generate a sequence of points at which the AMR will stop en route to the goal.

Our objective, however, is to generate subgoals for more sophisticated SCA's which can dynamically steer the system to visible subgoals without necessarily starting at rest. The purpose of the SSA is to provide the SCA with a temporary direction to pursue when the final goal p_g is not visible. Continuous motion is achieved by reevaluating and updating the subgoal while the AMR is moving. Each subgoal is pursued by the SCA until a new subgoal is provided by the SSA.

One condition which must be satisfied before a new subgoal can be introduced is directly related to the capabilities of the SCA, as characterized by the following definition.

Definition: A candidate subgoal g^* is said to be *safe* at time t with respect to a local map $O(p_k)$ and the current dynamic state $x(t)$ if the SCA can bring the AMR to rest at g^* along a collision-free trajectory in the obstacle-free space visible in $O(p_k)$.

The safeness of a subgoal is evaluated with respect to the AMR state at the current time t, which is different from the time at which the sensor data for local map $O(p_k)$ were taken. This reflects the fact that the time required to generate the local map and the subgoal is not negligible.

We assume there exists an algorithm for evaluating whether a candidate subgoal is safe. Any SCA can be used which satisfies these assumptions. For example, in Section V we use an SCA developed by Feng based on potential fields in the local obstacle-free space [18]. Other possible SCA's can be found in the literature on so-called *avoidance control* [19]–[22].

We can now state the problem solved by the subgoal selection algorithm as follows. Given an AMR modeled as a point in the plane and a finite number of convex obstacles, use the local map of visible obstacles at discrete points in time to generate safe subgoals to be pursued by an SCA such that AMR arrives at the final goal in finite time.

III. Subgoal Selection Algorithm (SSA)

In this section we describe and illustrate the steps in the SSA. The purpose of the subgoal is to provide the steering algorithm a temporary direction to pursue when the final goal is not in sight. The subgoal is chosen to direct the AMR towards the final goal while avoiding collisions with the obstacles. To achieve it, we determine the subgoals based on the data (local map) collected by the sensors on board the AMR. As new sensor data become available, the SSA determines when to update the subgoal.

To present the algorithm in a clear, concise manner, we use functions and data structures in a pseudoprogramming language. The following notation is used: small bold-faced letters represent points in the plane, large bold-faced letters represent other data structures, and small capitals are used for function names with brackets $\langle,\rangle$ delimiting function arguments. Before presenting the algorithm we define some of the variables, data structures, and functions.

A straight line (unbounded) containing two points a and b is denoted by ab. The open or closed termination of a line segment is indicated by a parenthesis or bracket, respectively. For example, $(ab]$ denotes the line segment from a to b which contains b but does not contain a, and $[ab$ denotes the ray beginning at and including a passing through b. The function INT $\langle ab, cd\rangle$ equals the point at which the two lines ab and cd intersect. If the arguments of INT $\langle .,.\rangle$ do not intersect, the function returns the symbol NIL, which is used throughout the algorithm definition to indicate empty sets or vacuous conditions.

At the kth execution of the SSA, the local map consists of a set $O(p_k)$ of line segments representing obstacle faces which are visible from p_k. Connected lines in $O(p_k)$ are faces of the same obstacle. Given two points a and b, the function OBS $\langle a, b\rangle$ is defined as the line segment (obstacle face) in $O(a)$ which intersects the line segment (ab), along with all other faces in $O(a)$ connected to the intersected face. When point b is visible from point a, OBS $\langle A, B\rangle$ equals NIL.

Connected line segments in $O(p_k)$ are faces of a single obstacle, where the points of connection are vertices of the obstacle. Each set of connected line segments will have two *extreme points*; that is, two ends not connected to any other line segments in $O(p_k)$. We refer to these extreme points as *edges* and denote the set of edges at iteration k by $E(p_k)$. Note that if some faces of obstacle A are only partially visible from p_k, due to the presence of another obstacle, B, then at least one extreme point on obstacle A is not actually a vertex. To distinguish edges which are actually vertices of the obstacle models from extreme points arising from obstructions, we let $E'(p_k)$ denote the set of edges which are actually vertices on the visible obstacles in $O(p_k)$. Given two points a and b, if OBS $\langle a, b\rangle \neq$ NIL, then the two extreme points on the connected lines in OBS $\langle a, b\rangle$ correspond to two edges in $E(p_k)$, but not necessarily in $E'(p_k)$. From the perspective of p_k, we distinguish the two edges on OBS $\langle p_k, b\rangle$ as "right edge" and "left edge," with the obvious meaning. Note that a particular physical obstacle can lead to more than one set of connected faces in $O(p_k)$ when it is partially obstructed from view by other obstacles, but it can lead to no more than two visible edges in $E'(p_k)$.

Subgoals are always associated with obstacle edges or vertices in a local map. A subgoal g and its associated obstacle edge (or vertex) e make up the fundamental data structure referred to as a subgoal–edge pair, denoted by (g, e), the associated subgoal is defined as a point g a distance $\epsilon \geq 0$ from e. The precise locations of subgoals g with respect to associate edges e are given below in the descriptions of the functions NEXT_VERTEX, CHOOSE_EDGE, OTHER_EDGE, EXTEND_FACE, OTHER_VERTEX, and SUBGOAL. We choose $\epsilon \leq \kappa/2$ to assure that the subgoal for an edge is not on another obstacle (which must be at least a distance κ from e). The purpose of setting the subgoal a certain distance from the edge is to guide the AMR around a vertex to a vantage point from which it is guaranteed that further subgoals will be generated.

Subgoal–edge pairs generated in the SSA are stored on a dynamic stack S which is operated on using the standard functions POP $\langle S\rangle$ and PUSH $\langle (g, e), S\rangle$. In the remainder of the paper (g, e) denotes the subgoal–edge pair currently being pursued by the SCA, and (g^T, e^T) denotes the top of the stack S, that is, (g^T, e^T) = POP $\langle S\rangle$. The function HEIGHT $\langle S\rangle$ equals the number of subgoal–edge pairs in the stack S.

$C(a_1, \cdots, a_n)$ denotes the open convex hull of a set of n points in the plane $\{a_1, \cdots, a_n\}$. L is a particular subset of subgoals generated by the SSA. The meaning of the points in L is described below in the discussion of Step 4 of the SSA.

The SSA is given in Fig. 2. In the remainder of this section we discuss each step in turn, providing further definitions of variables, data structures, and functions as needed.

The algorithm is initialized in Step 1 when $k = 0$ and the AMR is at rest at p_0. If there are no obstacles between the AMR and the final goal p_g, the subgoal g is set equal to p_g and the SSA terminates. It is assumed that the SCA can drive the AMR from rest to any visible point. Thus, p_g is a safe subgoal at this stage and no other subgoals are needed. When p_g is not visible initially, it is pushed on the stack (with a default value of p_g for the edge), L and h (defined below) are initialized, and the algorithm is continued at Step 5. It will be shown in the following section that in this case more subgoals will be put on the stack in Step 6 and a safe subgoal–edge pair will be generated.

The SSA terminates at Step 2 when the final goal is already being pursued (no further subgoals are needed), or when the ray from the goal on the top of the stack g^T through p_k does not

```
step 1: If k=0 then
            If (OBS<p0,pg> = NIL) then
                (g,e) ← (pg,pg)
                exit
            else
                PUSH<(pg,pg),S>
                L ← NIL
                h ← p0
                (g', e') ← CHOOSE_EDGE<OBS<p0, pg>>
                goto step 5
step 2: If ( g=pg or INT<[gTpk, (eg> = NIL) then
            exit
step 3: SAVE<S,(g,e),[ab],L,h>
        If (HEIGHT<S> ≥ 3) then
            goto step 6
step 4: If (HEIGHT<S> = 1) then
            If (OBS<pk,pg> = NIL) then
                goto step 7
            If (O(pk) ∩ C(pk, g, e) ≠NIL) then
                goto step 8
            (g', e') ← NEXT_VERTEX < O(pk), (g, e)>
            If (INT < (ge'), (epg) > ≠ NIL or e' =e) then
                If (OBS<pk,pg> ∩ C(g,e,pg)=NIL) then
                    goto step 8
                else
                    (g', e') ← CHOOSE_EDGE<OBS<pk, pg>>
                    L ← L U {h}
                    h ← g
            If (INT<(hg')],(pgx)>≠NIL and INT<(hg')],(pgx)>=NIL for some x∈ L) then
                (g',e') ← OTHER_EDGE<O(pk),(g',e')>
        else
            If (O(pk) ∩ C(pk,gT,eT) ≠ NIL) then
                goto step 6
            (g', e') ← EXTEND_FACE < O(pk), (gT, eT)>
            If (INT<(hg')],(pgx)>≠NIL and INT<(hg')],(pgx)>=NIL for some x∈ L) then
                (g',e') ← OTHER_VERTEX<O(pk),(g',e')>
            POP<S>
step 5: [ab] ← [pke']
        (g, e) ← (pk, pk)
        PUSH < (g', e'), S >
step 6: while (O(pk) ∩ C(pk, gT, eT) ≠ NIL or OBS<pk,gT> ≠ NIL)
            (g',e') ← CREATEG<pk,(gT,eT),(g,e),[ab]>
            If ((g',e') = NIL) then
                goto step 8
            else
                PUSH<(g',e'),S>
step 7: (g,e) ← POP<S>
        If (SAFE<g,x(t), O(pk)> then
            exit
step 8: RESTORE<S,(g,e),[ab],L,h>
        end
```

Fig. 2. Subgoal selection algorithm (SSA).

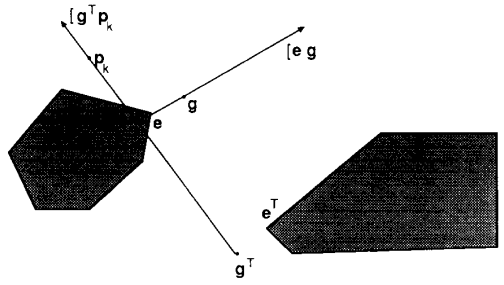


Fig. 3. Condition tested in Step 2: p_k must be past current edge e before the visibility of subgoal g^T is evaluated.

intersect the ray from the current edge e through the goal g. As illustrated in Fig. 3, this second condition assures that a new subgoal is sought only after the AMR has gone beyond the current edge e so that visibility of the subgoal g^T on the top of the stack is not blocked by the obstacle face that produced e.

In Step 3 the function SAVE stores the current *context* which consists of the stack S, the current subgoal–edge pair (g, e), line segment $[ab]$, a set of previous subgoals L, and vector h ($[ab]$, L, and h are defined below). After the context is saved in Step 3, the algorithm jumps to Step 6 if there are three or more subgoal–edge pairs in the stack S. If it is determined in Steps 4–7 that a new subgoal should not be selected on the current iteration, the context stored in Step 3 is retrieved by the function RESTORE in Step 8.

Step 4 is executed for two mutually exclusive cases, namely,

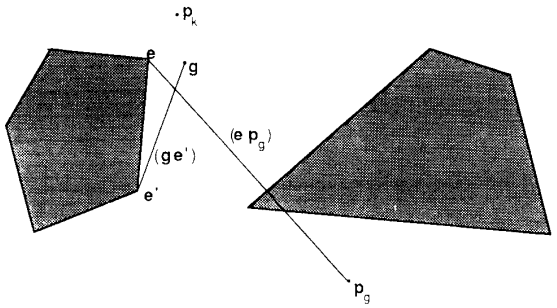


Fig. 4. Condition tested in Step 4: if vertex e' is not a face blocking the view of p_g, a new obstacle is used to generate a subgoal.

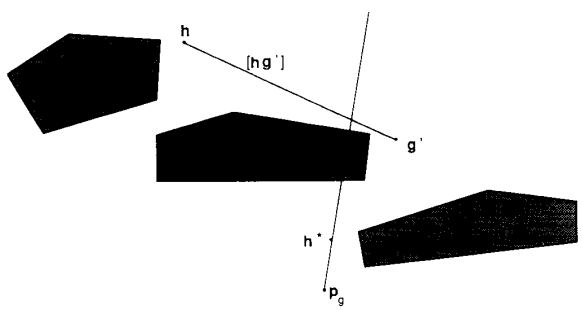


Fig. 5. Example of candidate subgoal g' taking the line $[hg']$ "behind" a previous subgoal h^* in L in Step 4.

when only the final goal p_g remains in the stack (HEIGHT $\langle S \rangle = 1$) or when the stack contains one subgoal in addition to the final goal (HEIGHT $\langle S \rangle = 2$). In the first case, if the final goal is visible from p_k, the algorithm jumps to Step 7 to evaluate whether the final goal is safe. If the final goal is not visible, the next condition in Step 4 assures that the line segment (e, g) is visible. If it is not visible, the SSA jumps to Step 8 to restore the original context. Otherwise, Step 4 continues and the function NEXT_VERTEX generates a subgoal-edge pair (g', e') where e' is the next visible vertex on the obstacle containing the current edge e in the direction of g. The associated subgoal g' is on the extension of the face containing e', a distance ϵ from e'. If the current edge e is an extreme point in $O(p_k)$, then NEXT_VERTEX returns $(g', e') = (g, e)$, the current subgoal-edge pair.

The next **if** condition in Step 4 determines whether the edge e' is on a face between the current edge and the goal. This condition is illustrated in Fig. 4. If INT $\langle (ge'), (ep_g) \rangle \neq$ NIL, or $e' = e$, the current edge is the last one that must be circumvented on that obstacle. In this case, before choosing an edge on another obstacle, a test is performed to see if the obstacle between p_k and p_g intersects the region $C(g, e, p_g)$. The purpose of this test is to assure that the next obstacle lies between the current subgoal and p_g. If OBS $\langle p_k, p_g \rangle \cap C(g, e, p_g) =$ NIL, the SSA jumps to Step 8 and terminates. If OBS $\langle p_k, p_g \rangle \cap C(g, e, p_g) \neq$ NIL, a subgoal-edge pair is generated by the function CHOOSE_EDGE from one of the two edges for OBS $\langle P_k, p_g \rangle$. Since only local information is being used in the SSA, there is no "optimal" choice between these two edges. In our implementation of CHOOSE_EDGE we select the edge closest to p_g. We show in the following section that the SSA steers the AMR to the final goal no matter which edge is selected by CHOOSE_EDGE.

When a new obstacle is used to generate the potential subgoal-edge pair, h is added to the set L. The variable h is then set equal to the current subgoal g. Thus, h is the final subgoal generated in Step 4 for a particular obstacle before switching to a new obstacle and L is the set of previous values of h. The set of subgoals L serves as a "memory" of where the AMR has been so that it does not cycle around the final goal indefinitely. This is assured by the next condition tested in Step 4 which, in words, checks to see if the subgoal g' will take the AMR "behind" a previous subgoal in L. This condition is illustrated in Fig. 5. If it occurs, the function OTHER_EDGE generates a subgoal-edge pair from the other edge on the obstacle containing e'. This concludes Step 4 when HEIGHT $\langle S \rangle = 1$.

In the second case of Step 4 (when HEIGHT $\langle S \rangle = 2$), if the line segment $(g^T e^T)$ is not completely visible from p_k [which is the case when there are obstacles in $C(p_k, g^T, e^T)$], the SSA jumps to Step 6. Otherwise, function EXTEND_FACE chooses e' as the most extreme visible point on the face collinear with $(g^T e^T)$ in the direction of g^T from e^T. The subgoal g' is generated as the extension of this face a distance ϵ from e'. Before going to Step 5 to push this new subgoal-edge pair on the stack, the same test is applied as described above for case one of Step 4 to make sure the

```
CREATEG ⟨p_k, (g^T, e^T), (g, e), [ab], E'(p_k)⟩
    A ← E'(p_k) ∩ {C(e, g, e^T, g^T) U [g, g^T]}
    if (A = NIL) then
        (g', e') ← NIL
    else
        e' ← MIN_DIST ⟨A, [ab]⟩
        g' ← SUBGOAL ⟨e', [ab]⟩
    return (g', e')
```

Fig. 6. Function CREATEG.

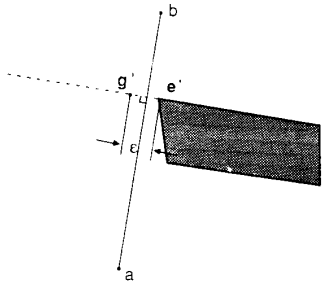


Fig. 7. Location of subgoal g' generated by function SUBGOAL for edge e'.

subgoal generated by EXTEND_FACE is not taking the AMR behind a previous subgoal in L. If it is, the other vertex of the face is chosen by OTHER_VERTEX to generate the subgoal-edge pair (g', e'). This subgoal-edge pair will replace the subgoal-edge pair on top of the stack (g^T, e^T) which is "popped" and discarded.

Step 5 updates the appropriate variables with the subgoal-edge pair (g', e') generated in Step 4. The line $[ab]$ is set equal to line segment $[p_k e']$, which is guaranteed not to intersect the interior of any obstacles. The current edge e is set equal to p_k for use in the function CREATEG in Step 6. The subgoal-edge pair (g', e') is then pushed on the stack. We shall see that the line segment $[ab]$ provides the "memory" required to assure that if any other subgoal-edge pairs are pushed on top of (g', e'), they will guide the AMR to a point from which the subgoal g' is visible.

In Step 6, the function CREATEG generates subgoal-edge pairs when the stack already contains a subgoal-edge pair generated previously in Step 4. The function CREATEG is defined in Fig. 6. An edge selected by CREATEG must satisfy conditions related to its proximity to the current edge e, the edge e^T on top of the stack, and the line $[ab]$. If no such edges exists, NIL is returned. The function CREATEG chooses the edge e' in $E'(p_k)$ (the set of visible edges) which is closest to the line $[ab]$ while being in the region $C(g, e, g^T, e)$. The significance of these conditions is explained in the following section where we prove the trajectory converges to the final goal in finite time.

As illustrated in Fig. 7, function SUBGOAL generates a subgoal g' a distance ϵ from e' on a line perpendicular to, and in the

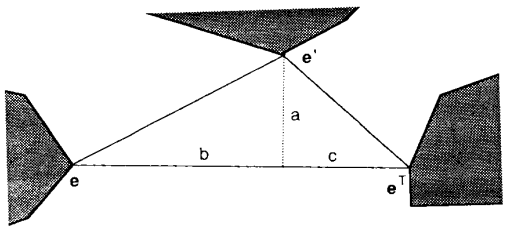


Fig. 8. Triangle with vertices e', e, e^T and line segments a, b, c, used in proof of Lemma 1.

direction of, the line segment $[ab]$. Subgoal-edge pairs are generated by CREATEG and pushed on the stack until the line segment $[g^T e^T)$ is visible. If CREATEG returns NIL, the algorithm jumps to Step 8 and the original context is restored. In Step 7 the top of the stack is popped and the (visible) subgoal is evaluated by the function SAFE which returns the logical condition TRUE if the SCA can drive the AMR to g along a collision-free path given the current system state $x(t)$ and the local map $O(p_k)$. If g is a safe subgoal, the algorithm terminates and this new subgoal is pursued by the SCA. If g is not safe, Step 8 is executed and the original context is restored.

IV. Proof of Convergence

In this section we prove that the SSA generates a sequence of subgoals which guide the AMR to the final goal p_g in finite time, that is, we show that p_g eventually becomes a safe subgoal for the SCA under the assumptions delineated in Section II. We prove this by establishing certain properties of the sequence of subgoals which are generated by consecutive iterations of the SSA. In particular, we show that the height of the stack S is finite bounded, that any goal generated by CREATEG eventually becomes a safe subgoal, and that the height of the stack is always reduced to one in a finite number of iterations. Finally, we demonstrate that the SSA can generate only a finite number of subgoal-edge pairs in Step 4 when the stack height is one before the final goal is visible and safe.

We begin with three lemmas on the properties of subgoal-edge pairs generated by the function CREATEG in Fig. 6. The first lemma establishes a bound on the distances between an edge generated by CREATEG and the two edges passed to the function as arguments. Lemmas 2 and 3 are existence results which state that under certain conditions a new subgoal-edge pair will always be generated by CREATEG.

Lemma 1: Suppose that at iteration k of the SSA, (g', e'), is a subgoal-edge pair generated in Step 6 by CREATEG for given values of p_k, (g^T, e^T), (g, e), $[ab]$, and $E'(p_k)$. If e', e, and e^T are on three different obstacles, then

$$\|e'-e\|^2 \le \|e-e^T\|^2 - 8\epsilon^2, \text{ and } \|e'-e^T\|^2 \le \|e-e^T\|^2 - 8\epsilon^2.$$

Proof: The first condition in the definition of CREATEG guarantees that the distance from e' to the line $[ee^T]$ is less than ϵ, since g and g^T are a distance ϵ from e and e^T, respectively. Also, the pairwise distances between e', e, and e^T are all more than $\kappa \ge 2\epsilon$ since they are on different obstacles. The geometry of this situation is illustrated by the triangle with vertices e', e, and e^T in Fig. 8. From the figure we have

$$b^2 = \|e'-e\|^2 - a^2 \ge 3\epsilon^2 \text{ and } c^2 = \|e'-e^T\|^2 - a^2 \ge 3\epsilon^2.$$

Since $\|e - e^T\|^2 = (b + c)^2$, we have from the above inequalities for b and c

$$\|e-e^T\|^2 \ge b^2 + c^2 + 6\epsilon^2. \quad (1)$$

Substituting the above equation for b^2 and inequality for c^2 in inequality (1) and noting $a \le \epsilon$, we have

$$\|e-e^T\|^2 \ge \|e'-e\|^2 + 8\epsilon^2.$$

Alternatively, substituting the above inequality for b^2 and equation for c^2 in inequality (1) gives

$$\|e-e^T\|^2 \ge \|e'-e^T\|^2 + 8\epsilon^2.$$

These last two inequalities prove the lemma. △

The bounds in Lemma 1 will be used to prove that only a finite number of edges can be generated in one iteration of the SSA, or added to the stack in multiple iterations of the SSA. In the following lemma we show that if the AMR has arrived at the current goal at some iteration m, that is, $p_m = g$, and CREATEG is executed, then it is guaranteed to generate a subgoal-edge pair.

Lemma 2: Suppose the current subgoal-edge pair (g, e) at iteration m of the SSA was generated by CREATEG at some previous iteration $k < m$. If $O(g) \cap C(g, g^T, e^T) \ne$ NIL, then

$$\text{CREATEG}\langle g, (g^T, e^T), (g, e), [ab], E'(g)\rangle \ne \text{NIL}.$$

Proof: It must be shown that under the hypotheses there is always an edge e' in $E'(g)$ which is in the region $C(e, g, e^T, g^T)$, since this will give $A \ne$ NIL in CREATEG. Since $O(g) \cap C(g, g^T, e^T) \ne$ NIL at least one obstacle intersects the region $R = C(e, g, e^T, g^T)$. Suppose there are no visible edges [elements of $E'(g)$] in R. This would imply that any line segment which intersects both $[eg]$ and $[e^T g^T]$ necessarily intersects an obstacle in R. We now show that $[ab]$ interests both $[eg]$ and $[e^T g^T]$, which is a contradiction since $[ab]$ does not intersect any obstacles except possibly at b.

Consider the first time CREATEG is executed in Step 6 following the definition of $[ab]$ in Step 5. In this case $a = p_k = g$ and $b = e^T$ and the assertion is trivially true, that is, $[ab]$ intersects $[eg]$ and $[e^T g^T]$. The other case is that (e, g) and possibly (e^T, g^T) were generated by CREATEG. If, however, CREATEG generates the subgoal-edge pair (g', e') from two subgoal-edge pairs (e, g) and (e^T, g^T) and $[ab]$, where $[ab]$ intersects $[eg]$ and $[e^T g^T]$, then $[ab]$ must intersect $[g'e']$. This follows from the fact that e' is within ϵ of $[ab]$ and $g' =$ SUBGOAL $\langle e', [ab]\rangle$, which puts g' a distance ϵ from e' on a line perpendicular to, and in the direction of, $[ab]$. Thus, it can be shown by induction that $[ab]$ intersects $[eg]$ and $[e^T g]$ in all cases and the lemma is proved. △

Lemma 2 applies to the first time CREATEG is invoked in Step 6 if the AMR has arrived at the current goal. In the following lemma, we show that if CREATEG is executed multiple times in Step 6 when the AMR is at g, a new subgoal-edge pair is always generated.

Lemma 3: Suppose for two subgoal-edge pairs (g, e), (g^T, e^T) there exists a subgoal-edge pair (g_1, e_1) such that $(g_1, e_1) =$ CREATEG $\langle g, (g^T, e^T), (g, e), [ab], E'(g)\rangle$. If $O(g) \cap C(g, g_1, e_1) \ne$ NIL, then

$$\text{CREATEG}\langle g, (g^1, e^1), (g, e), [ab], E'(g)\rangle \ne \text{NIL}.$$

Proof: As shown in the proof of Lemma 2, for any subgoal-edge pair (g_1, e_1) generated by CREATEG in Step 6 we have INT $\langle [ab], [g_1 e_1]\rangle \ne$ NIL. Therefore, since $O(g) \cap C(g, g_1, e_1) \ne$ NIL, there must be a visible edge in $C(e, g, e_1, g_1)$ by the proof of Lemma 2, *mutatis mutandis.* △

We now use these results to prove the following three propositions which characterize the sequences of subgoals generated by the SSA when there are more than two subgoal-edge pairs on the stack. First, we show in the following proposition that the SSA never gets "stuck" in an infinite loop.

Proposition 1: On any given iteration, the SSA always terminates in a finite number of steps.

Proof: Since Step 6 is the only point at which there exists the possibility of an infinite loop, it is sufficient to show that whenever Step 6 is executed, the function CREATEG eventually

returns NIL or it returns a subgoal-edge pair (g', e') for which $O(p_k) \cap C(p_k, g', e') = \text{NIL}$ and $\text{OBS}\langle p_k, g^T\rangle = \text{NIL}$. In fact, we shall show that on a given execution of Step 6, no more than $1 + \|e - e_0\|^2/8\epsilon^2$ subgoal-edge pairs can be pushed on the stack S, where e is the current edge and e_0 is the edge on top of S when Step 6 is initiated.

Let e_j be the edge returned on the jth call to CREATEG in a given execution of Step 6. From Lemma 1 we have for $j > 1$

$$\|e_j - e\|^2 \geq \|e - e_{j-1}\|^2 - 8\epsilon^2$$

where e_{j-1} is the edge on top of S when e_j is generated. Therefore, if e_j returned by CREATEG, it satisfies $\|e_j - e\|^2 \leq \|e - e_0\|^2 - (j-1)8\epsilon^2$ which implies $j \leq 1 + \|e - e_0\|^2/8\epsilon^2$, proving the lemma. △

The next proposition states that when there are more than two subgoal-edge pairs on the stack, the SSA always generates a new safe subgoal to pursue after a finite number of iterations. In other words, the current subgoal is eventually replaced with a new subgoal to pursue.

Proposition 2: If (g, e) is the current subgoal-edge pair and $\text{HEIGHT}\langle S\rangle \geq 3$ following iteration k_1 of the SSA, then for some $k > k_1$, the SSA will terminate with a new subgoal-edge pair (g^*, e^*) where g^* is safe with respect to the state x_k and local map $O(p_k)$.

Proof: If $\text{HEIGHT}\langle S\rangle \geq 3$, the SSA terminates either at Step 2, Step 7, or Step 8. If it terminates at Step 7, a new visible, safe subgoal has been found. It must be shown that on consecutive iterations the SSA cannot terminate indefinitely at Steps 2 or 8 while the SCA continues to pursue the same subgoal-edge pair (g, e). In the absence of a new subgoal from the SSA, we have assumed that the SCA will bring the AMR to rest at the current subgoal g in finite time. Moreover, we have assumed that when the AMR is stopped, the SCA can drive the AMR safely to any visible point. Thus, it suffices to show that if at some iteration $k > k_1$, the AMR is at rest with $p_k = g$, then the SSA cannot terminate at Step 2 to Step 8.

Under these assumptions neither condition in Step 2 can be satisfied. The first condition $g = p_g$ occurs only when p_g has become a visible, safe subgoal, in which case the stack is empty, but we have assumed $\text{HEIGHT}(S) \geq 3$. The second condition for terminating at Step 2 cannot be satisfied since $p_k = g$ implies $\text{INT}\langle [g^T p_k, (eg\rangle \neq \text{NIL}$.

We now consider the ways in which the SSA can terminate at Step 8. First note that if $\text{HEIGHT}(S) \geq 3$, Step 4 is not executed, which means that the SSA can terminate at Step 8 only if CREATEG returns NIL in Step 6, or a visible subgoal generated by CREATEG in Step 6 is not safe. Since we have assumed that the AMR is at rest at $p_k = g$, any visible subgoal will be safe. Hence, it suffices to show that CREATEG will generate a visible subgoal in Step 6 in this case. From Lemmas 1 and 2 it follows by induction that CREATEG cannot return NIL when $p_k = g$. Moreover, by Proposition 1, CREATEG eventually returns a subgoal-edge pair (g^*, e^*) for which $O(g) \cap C(g, g^*, e^*) = \text{NIL}$ and $\text{OBS}\langle p_k, g^T\rangle = \text{NIL}$. Since the subgoal g^* is visible, and hence safe, the proposition is proved. △

The implication of Proposition 2 is that no subgoal g generated by CREATEG will be persued indefinitely by the SCA since, in the worst case, the SCA eventually brings the AMR to rest at g from which it is guaranteed that a new safe subgoal will be generated. We now show in the next proposition that any subgoal generated by CREATEG and pushed on the stack will eventually become a visible, safe subgoal.

Proposition 3: If (g^*, e^*) is a subgoal-edge pair generated by CREATEG and pushed on the stack S during iteration k of the SSA, then at some iteration $m \geq k$, g^* will become the current subgoal.

Proof: Let e be the current edge at iteration k when (g^*, e^*) was pushed on the stack. Noting that all edges generated by CREATEG are obstacle vertices, we consider the following cases.

Case 1: e and e^* are vertices on the same physical obstacle O_A. In this case any further edges generated by CREATEG and pushed on the stack above g^* are necessarily on the same obstacle as e and e^* since edges on any other obstacles must be more than 2ϵ from O_A. Since any new edge (vertex) will be closer to e^* than e and there are a finite number of possible edges (vertices) on an obstacle, e^* must eventually become the top of the stack and visible from the current subgoal by Propositions 1 and 2. Moreover, it will eventually become safe since the SCA ultimately brings the AMR to rest at the current subgoal.

Case 2: e and e^* are on the different physical obstacles O_A and O_B. If any more subgoals are pushed on the stack they are either on O_A, O_B, or on yet another obstacle. If an edge e_B is generated on O_B and pushed on the stack, we have by the same argument as in Case 1 that there can be no edges between e^* and e_B on the stack which are not on O_B. Therefore, if e_B ever becomes the current edge, e^* will eventually become the current edge by Case 1. If, on the other hand, an edge e_A on O_A is generated and pushed on the stack, any edges pushed on top of e_A must also be on O_A, and by Case 1, e_A will eventually become the current edge. Moreover, only a finite number of edges on O_A can be pursued before the top of the stack is on another obstacle.

Finally, if an edge e' is not on O_A or O_B, then by Lemma 1 we have $\|e' - e\|^2 \leq \|e - e^*\|^2 - 8\epsilon^2$, that is, when the edges are on different obstacles the distance between the current edge and the new edge is always less than the distance between the current edge and the top of the stack by a finite amount. Therefore, only a finite number of different obstacles can lead to edges which are pushed on top of e^*. Since there are a finite number of obstacles, g^* eventually becomes visible and safe. △

In summary, the above propositions demonstrate that any subgoals generated by CREATEG which get put on the stack are eventually removed from the stack and used as the current subgoal for the SCA. This implies that if the height of the stack ever exceeds two, it will always return to that height. We now deal with the subgoals generated in Step 4 which is executed when only the stack has less than three subgoal-edge pairs.

We now consider the subgoal-edge pairs generated in Step 4 when $\text{HEIGHT}\langle S\rangle = 1$ or 2. Step 4 generates subgoal-edge pairs for visible obstacles which are between the AMR and the final goal. As described in the previous section, when only (p_g, p_g) is in the stack and there is an obstacle O between p_k and p_g, Step 4 is executed and a subgoal-edge pair is generated from O and pushed on the stack. In future iterations, whenever the stack height is 2, new subgoal-edge pairs are generated on the same face of the obstacle O until a real vertex is used as the edge. Subsequent vertices on the obstacle are used as edges until the final goal becomes visible and safe, or until subgoals can be generated for another obstacle which lies between the AMR and p_g. To analyze the sequence of subgoals generated in Step 4 we make the following definition.

Definition: A vertex v on an obstacle O is said to be an *extreme vertex* (with respect to p_g) if the ray $[p_g v$ does not intersect the interior of O and if a face F of O is contained in $[p_g v$, then v is the vertex of F which is the greatest distance from p_g.

Clearly every obstacle has exactly two extreme vertices. The objective of Step 4 is to generate a sequence of subgoal-edge pairs on an obstacle until one of the extreme vertices is used as an edge. At that point either p_g is visible or another obstacle is used to generate subgoal-edge pairs in Step 4. A subgoal for an extreme vertex on an obstacle, referred to as an *extreme subgoal,* is saved as the variable h in Step 4, and then placed in the set L when an extreme vertex of the next obstacle has been reached.

When a candidate subgoal g' is generated in Step 4, it is tested to assure the line from h (the last extreme subgoal) to g' does not go behind one of the previous extreme subgoals in L, as described in the previous section. The following lemma implies that if the line segment $[hg']$ does go behind a subgoal in L, then the sequence of subgoals which will be generated in the other direction on the obstacle (starting with the subgoal generated by

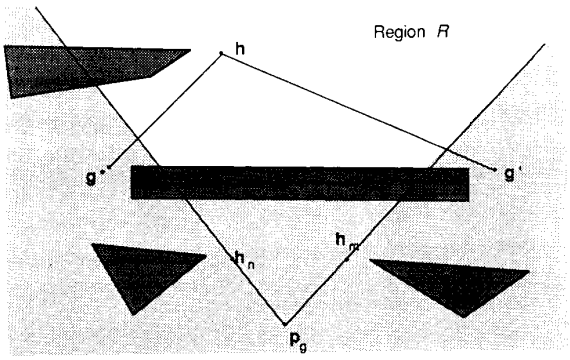


Fig. 9. Situation used to arrive at contradiction in the proof of Lemma 4.

OTHER_EDGE or OTHER_VERTEX) will not lead the AMR behind any other subgoals in L.

Lemma 4: Suppose Step 4 is executed during an iteration of the SSA and (g', e') is a subgoal-edge pair generated for an obstacle O by the function NEXT_VERTEX, CHOOSE_EDGE, or EXTEND_FACE. Let g^* be the extreme vertex of O in the opposite direction of e' as g'. If the line segment $[hg']$ goes behind one of the subgoals in L, then the line segment $[hg^*]$ will not go behind any subgoals in L.

Proof: Let $L = \{h_1, \cdots, h_j\}$, where j is the number of extreme subgoals in L and the index indicates the order in which the extreme subgoals have been added to L. For notational convenience we assign the index $j + 1$ to the most recent extreme subgoal h, that is, $h_{j+1} = h$. Note that for any $h_k \in L$ the line segment $[h_k h_{k-1}]$ cannot go behind any of the subgoals $h_1, \cdots, h_{k-2}$.

We now prove the lemma by contradiction. Suppose at some iteration of the SSA a candidate subgoal g' is generated in Step 4 for which the line segment $[hg']$ goes behind $h_m \in L$ and the line segment $[hg^*]$ goes behind $h_n \in L$, where g^* is the extreme subgoal on the obstacle as defined in the statement of the lemma. This situation is illustrated in Fig. 9. As illustrated in the figure, the most recent extreme subgoal h_{j+1} is the region R behind the obstacle O (with respect to p_g) and between the rays $[p_g h_m$ and $[p_g h_n$. Without loss of generality we assume $m < n$. Since $h_{j+1} \in R$ and $h_n \notin R$, there is an index i where $n < i \leq j + 1$ such that $\{h_i, \cdots, h_{j+1}\} \subset R$ and $h_{i-1} \notin R$. By the property of the extreme subgoals in L, the line segment $[h_{i-1} h_i]$ cannot pass behind h_m or h_n. Therefore, the line segment $[h_{j-1} h_i]$ must go through the obstacle O. However, since h_i must be an extreme subgoal generated in Step 4 for an obstacle encountered while pursuing the subgoal h_{i+1}, this situation contradicts the assumption that the obstacles are convex, which proves the lemma. △

The implication of Lemma 4 is that the direction in which consecutive vertices of a given obstacle are selected as edges in Step 4 can change at most once. Thus, an extreme vertex will always be reached in a finite number of steps. This result is stated as the following proposition.

Proposition 5: If at some iteration k_0 of the SSA, h is the most recent extreme subgoal saved in Step 4 and the subgoal-edge pair (g_0, e_0) is generated from obstacle O in Step 4 and pushed on the stack in Step 5, then at some iteration $k^* \geq k_0$, the SSA will terminate with $(g, e) = (g^*, e^*)$ where g^* is a visible, safe subgoal, and with $g^* = p_g$, the final goal, or e^* is an extreme edge of the obstacle O.

Proof: First note that whenever Step 5 is executed, (p_g, p_g) is the only subgoal-edge pair in the stack. Thus, (g_0, e_0) is the second entry in the stack. Also, the line segment $[gh_0]$ does not go behind any of the extreme subgoals in the set L. This follows from the condition tested in Step 4 for subgoals generated by either NEXT_VERTEX, CHOOSE_EDGE, or EXTEND_FACE. If the subgoal initially generated by one of these functions went behind a subgoal in L, the function OTHER_EDGE or OTHER_VERTEX would generate the subgoal g_0 which, by Lemma 4, cannot take the line segment $[hg_0]$ behind a subgoal in L.

Any subgoals pushed on top of g_0 in the stack are necessarily generated by CREATEG in Step 6. By Proposition 3, any such subgoals eventually become visible and safe. Therefore, in a finite number of iterations the SSA terminates with HEIGHT $\langle S \rangle = 2$ and (g_0, e_0) on top of the stack. On the next iteration, if the entire line segment $(g_0 e_0)$ is not visible, Step 6 is executed and more subgoal-edge pairs are pushed on the stack. By the same argument as used in the proof of Proposition 3, this can occur for only a finite number of iterations before (g_0, e_0) is the top of the stack and the entire line segment $(g_0 e_0)$ is visible. Suppose this occurs at iteration $k_1 \geq k_0$. At this point Step 4 is executed and (g_0, e_0) is replaced by a new subgoal-edge pair (g_1, e_1) where e_1 is one of the two visible extreme points of the face containing e_0 (generated by either EXTEND_FACE or OTHER_VERTEX). Again, by Lemma 4, the line segment $[hg_1]$ does not go behind any subgoal in L.

Applying the above argument again, (g_1, e_1) becomes the top of the stack in a finite number of iterations and is replaced by a new subgoal-edge pair (g_2, e_2) with e_2 on the same obstacle face as e_0. This cycle continues until at some iteration k_{n1} a subgoal-edge pair (g_{n1}, e_{n1}) is generated in Step 4 where e_{n1} is an actual vertex of the obstacle face containing e_0 and the line segment $[hg_{n1}]$ does not go behind any subgoals in L. This follows from Lemma 4 since at least one of the vertices of the obstacle face cannot take the line from h behind any subgoals in L. By the same arguments as in the proof of Proposition 3, g_{n1} eventually becomes a visible, safe subgoal.

If e_{n1} is an extreme edge of O, the theorem is proved. Otherwise, in the next iteration Step 4 is executed with HEIGHT $\langle S \rangle = 1$ at which point there are several possibilities. If the final goal is visible and safe, the SSA returns $g = g_g$ and the theorem is proved. If p_g is visible but not a safe subgoal, a new subgoal is not generated and the former context is restored. If p_g remains visible on future iterations, then it eventually becomes a safe subgoal since the SCA will stop the AMR at the current subgoals, again proving the proposition.

If p_g is not visible when Step 4 is executed with HEIGHT $\langle S \rangle = 1$, the SSA first checks to see if the region $C(p_k, g, e)$ is free of obstacles. If it is not, the SSA terminates and the former context is restored. This condition must eventually be satisfied, however, since the SCA drive the AMR to g. Therefore, if p_g does not become a safe subgoal while pursuing g_{n1}, the function NEXT_VERTEX is executed in Step 4 after a finite number of iterations and a subgoal-edge pair (g', e') is generated. Since we have assumed that e_{n1} is not an extreme edge, the edge e' is either the same as e_{n1}, or e' is on a different face on O. In the former case, p_k is not a position from which the next face on O is visible and the SSA terminates after restoring the context. This can only happen a finite number of times before the SCA drives the AMR to a position from which the next face is visible. In the latter case, e' is retained if $[hg']$ does not go behind a subgoal in L, otherwise an alternative subgoal-edge pair is generated in the other direction by OTHER_EDGE which is guaranteed not to go behind a subgoal in L by Lemma 4. The subgoal-edge pair generated in Step 4 is pushed on the stack in Step 5.

By the discussion above, after a finite number of iterations there is some iteration $k_{n2} \geq k_{n1}$ such that a subgoal-edge pair (g_{n2}, e_{n2}) is generated in Step 4 with e_{n2} the next vertex on the same obstacle face as e_{n1}. This cycle will continue, generating a sequence of edges $e_{n1}, e_{n2}, \cdots$ which are consecutive vertices of obstacle O. By Lemma 4 and the assumption that O has a finite number of vertices, this sequence must eventually terminate with a subgoal-edge pair (g^*, e^*) where e^* is an extreme edge of O. Moreover, there will be an iteration k^* for which g^* is returned by the SSA as a visible, safe subgoal and HEIGHT $\langle S \rangle = 1$. △

The previous proposition implies that the height of the stack will always be reduced to one in a finite number of iterations. The

following final lemma states that the SSA returns a valid subgoal on the first iteration when $k = 0$.

Lemma 5: When $k = 0$ with the AMR stopped at p_0, the SSA terminates with a safe, visible subgoal.

Proof: If p_g is visible from p_0, the subgoal $g = p_g$ returned by the SSA in Step 1 is safe by the assumption that the SCA can steer the AMR to any visible point when it is initially at rest. If p_g is not visible, (p_g, p_g) is pushed on the stack and a subgoal-edge pair (g', e') is generated by CHOOSE_EDGE from the obstacle between p_0 and p_g. The SSA skips to Step 5 where the line segment $[ab]$ is set equal to $[p_0e']$, the current subgoal-edge pair (g, e) is set equal to (p_0, p_0), and (g', e') is pushed on the stack (which now has height 2). Step 6 is then executed. By Lemmas 2 and 3, CREATEG will not return NIL because $p_0 = g$. Moreover, by Proposition 1, Step 6 will terminate in a finite number of steps at which point a visible subgoal will be on top of the stack and Step 7 will be executed. The visible subgoal on top of the stack is necessarily safe since the AMR is at rest, which proves the lemma. △

We now use the previous results to prove the fundamental property of the SSA which is stated as the following theorem.

Theorem: Given an AMR modeled as a point in two-dimensions, initially at rest at position p_0, a finite number of polygonal convex obstacles with a finite number of vertices separated by a distance of at least 2ϵ, and an SCA (steering control algorithm) capable of driving the AMR from rest to stop at any point visible from the initial point. If the final goal position p_g is not contained in an obstacle, then the SSA (subgoal selection algorithm) will always return a visible, safe subgoal and will return p_g as a visible safe subgoal after a finite number of iterations.

Proof: By Lemma 5 the SSA returns a visible, safe subgoal when $k = 0$. If $g = p_g$, then the theorem is proved. If $g \neq p_g$, the stack will contain at least one subgoal-edge pair, namely, (p_g, p_g). By Proposition 1 the SSA always terminates in subsequent iterations with a visible, safe subgoal. If after any iteration there is more than one subgoal-edge pair in the stack, Proposition 5 states that after a finite number of iterations either p_g will be returned as a visible, safe subgoal, or HEIGHT $\langle S \rangle = 1$ and the visible, safe subgoal will be an extreme subgoal on an obstacle. If the former case occurs, the theorem is proved. The latter case can occur only a finite number of times since there are a finite number of extreme subgoals and, by Lemma 4, no extreme subgoal can become the current subgoal when HEIGHT $\langle S \rangle = 1$ more than once. Therefore, after a finite number of iterations, an extreme subgoal will become the current subgoal g with HEIGHT $\langle S \rangle = 1$ and, since the SCA will eventually bring the AMR to rest at g, p_g will become a visible, safe subgoal. △

V. Simulation Examples

To illustrate the properties of the SSA we have implemented a simulation program using a simple point mass model of an AMR in a two-dimensional planar region containing polygonal obstacles. The dynamics of the point mass are given by

$$\ddot{p} = \dot{v} = u \tag{2}$$

where $p(t)$ and $v(t)$ are the AMR position and velocity, respectively, and $u(t)$, the acceleration vector, is the control input for the dynamic steering control algorithm (SCA). We assume the magnitude of the acceleration is constrained as

$$\|u(t)\| \leq \alpha. \tag{3}$$

The control problem is to transfer the point mass with dynamics (2) along a collision-free trajectory in the plane from an initial position $p(0) = p_0$ to a final goal position $p(t_f) = p_g$ with $v(0) = v(t_f) = O$, subject to the control constraint (3). The final time t_f is unspecified.

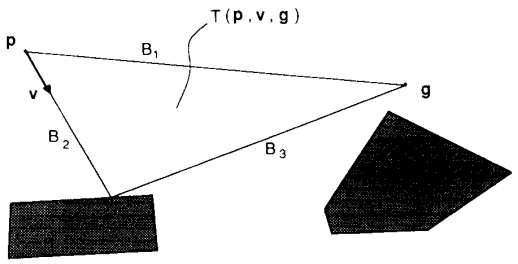


Fig. 10. Boundaries B_1, B_2, B_3 of region $T(p, v, g)$ used to generate potential fields in the PFA.

Note that even if the obstacle locations are known *a priori*, this problem would be difficult to solve using optimal control theory because the obstacles impose complex state-variable constraints. Obtaining a feedback solution using only the information about obstacles visible from $p(t)$ would be even more difficult. Our approach is to use the SSA to generate subgoals for an SCA which steers the point mass with the local obstacle-free space. The particular SCA used in the simulation examples is a dynamic feedback scheme based on artificial potential fields which "attract" the point mass to the subgoal while "repelling" it from a set of boundaries defining an obstacle-free region in the most recent local map. Since our purpose here is to illustrate the properties of the SSA, we only briefly describe the basic elements of the SCA which we shall refer to as the potential field algorithm (PFA). General properties of the potential field approach are discussed by Krogh in [17], and details of the PFA used in the simulation examples are given in Feng's Master's Thesis [18]. The objective of the PFA to choose the acceleration at each instant based on the available local information so as to guarantee the resulting trajectory is acceptable (but not necessarily "optimal").

For a given position p, velocity v, and subgoal g, the PFA computes the acceleration vector u to maximize the projection of u onto the gradient of a potential function $P(p, v, g)$ given by

$$P(p, v, g) = P_a(p, v, g) + P_r(p, v, g) \tag{4}$$

where $P_a(p, v, g)$ is computed to attract the point mass to the subgoal g and $P_r(p, v, g)$ is computed to repulse the point mass from the obstacles. The dependence of P_a and P_r on the system velocity as well as position takes into account the dynamic aspects of the trajectory and control constraints.

$P_r(p, v, g)$ is the sum of a set of repulsive potential functions $P_r(p, v, B_i)$ for three linear boundaries B_1, B_2, B_3 which define a triangular obstacle-free region $T(p, v, g)$ in the local map $O(p)$. As illustrated in Fig. 10, $B_1 = [pg]$, B_2 extends from p in the direction of v, and B_3 extends from g to $B2$ so as to maximize the area of $T(p, b, g)$ while keeping the angle at g less than or equal to $\pi/2$. For this SCA we define a subgoal to be *safe* when there is sufficient deceleration capability in the direction of v to bring the point mass to a stop before crossing boundary B_3.

The repulsive potential field $P_r(p, v, B_i)$ is defined as $(\tau_M(B_i) - \tau_m(B_i))^{-1}$, where $\tau_m(B_i)$ is the minimum time in which the velocity toward boundary B_i can be brought to zero using maximum deceleration, and $\tau_M(B_i)$ is the maximum time in which the velocity toward the obstacle can be brought to zero under constant deceleration without crossing boundary B_i. This gives

$$P_r(p, v, B_i) = v_i(v_i^2 - 2\alpha p_i)^{-1}$$

where $-v_i$ is the speed at which the AMR is moving toward B_i and p_i is the distance to B_i. The essential property of $P_r(p, v, B_i)$ is that it grows to infinity as the capability to avoid crossing the boundary goes to zero.

The goal attraction potential function $P_a(p, v, g)$ is computed so that the gradient results in an acceleration which will eventually

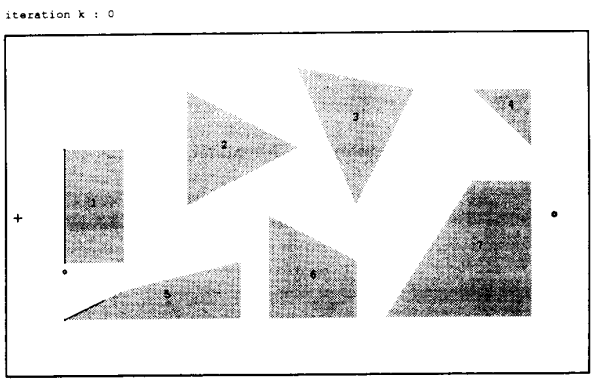


Fig. 11. Initial position (+) and local map for simulation example 1.

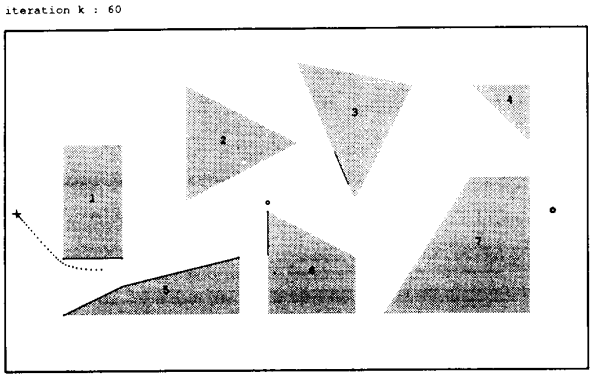


Fig. 12. Trajectory after $k = 60$ iterations for simulation example 1.

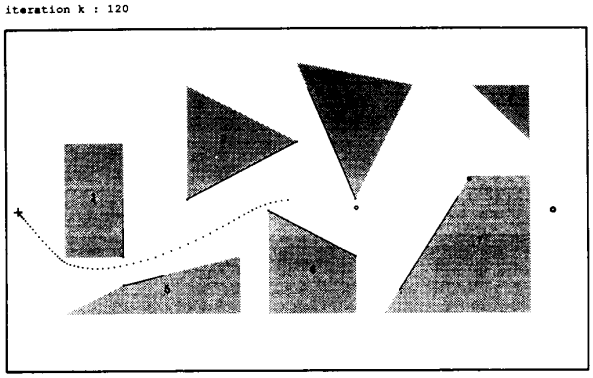


Fig. 13. Trajectory after $k = 120$ iterations for simulation example 1.

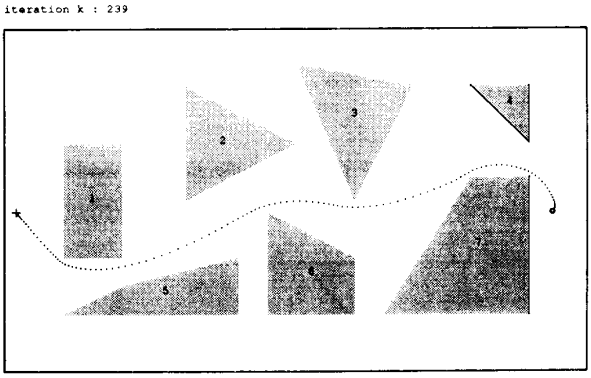


Fig. 14. Complete trajectory (242 iterations) for simulation example 1.

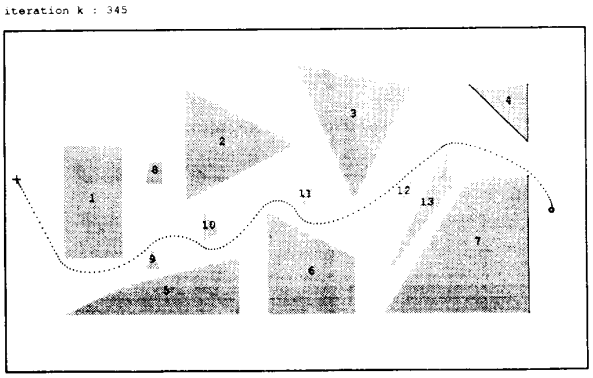


Fig. 15. Complete trajectory (345 iterations) for simulation example 2.

bring the system to rest at p_g. In the simulation examples we use a heuristic proposed by Krogh to compute orthogonal components of the goal attracting potential gradient based on the time required to reach the goal using the maximum acceleration available in each orthogonal direction in the absence of obstacles [17].

We implement the PFA as a sampled-data control scheme in which the acceleration u at each sampling instant is chosen to maximize the rate of decrease in the function $P(p, v, g)$, subject to the constraints 3. Additional constraints are imposed on u to guarantee the subgoal will be safe at the next control instant. In the simulations presented below the sample frequency for the PFA is the same as the frequency for the SSA iterations.

Figs. 11–14 show the evolution of the trajectory generated by the SSA and SCA for a particular configuration of obstacles. Points along the trajectory indicate the AMR position at constant time intervals. In each frame, bold lines indicate the obstacle faces which are visible from the current position, + indicates the initial position p_0, and small circles indicate the locations of the current subgoals in the stack. For example, in Fig. 13, there is one intermediate subgoal in addition to the final goal which is not visible from the current position. Note that the resulting trajectory provides smooth, continuous motion and the system is brought to rest at the final goal without overshoot.

The second case shown in Fig. 15 demonstrates that the SSA guides the point mass to the goal even in a very cluttered environment with virtually the same amount of computational effort as required for the previous case. The significance of these simulation results is that smooth, acceptable dynamic trajectories are being generated using feedback information without path planning.

VI. Conclusions

In this paper we have presented a feedback algorithm for selecting subgoals for guiding an autonomous mobile robot (AMR) using only local information about the locations of visible obstacles. We model the AMR as a point in a planar environment with convex, polygonal obstacles. The subgoal selection algorithm (SSA) generates a sequence of visible, safe subgoals which are pursued by a dynamic steering control algorithm (SCA). We have shown analytically that the subgoals generated by the SSA guide the AMR to the goal along a finite-time trajectory. Properties and performance of the algorithm were demonstrated by simulation examples using a point-mass model of an AMR with limited acceleration and an SCA based on artificial potential fields.

From the perspective of control theory, the fundamental contribution of this paper is a feedback algorithm for steering a

dynamic system subject to state-variable constraints (the obstacles) which are unknown *a priori*. The proposed algorithm is an alternative to the standard methods of dynamic programming or optimal control which become computationally intractable for even trivial examples. In contrast, the SSA can be implemented in real time to generate an acceptable sequence of subgoals whenever a feasible path to the final goal exists.

To integrate the SSA into a complete guidance system for an AMR, we are currently investigating the following issues.

- Uncertainty in the local map and AMR position. The present version of the SSA assumes perfect information. However, noise in sensor data is inevitable in AMR applications. One approach for dealing with uncertainty is to use very conservative estimates of the obstacle positions in the local map.
- Development of real-time steering control algorithms. The SCA requires an underlying dynamic control algorithm to drive the AMR to the subgoals. Such algorithms must be designed for the specific kinematic and dynamic configuration of the ARM.
- Integration of high-level planning. Since the SSA is intended for local navigation, higher level planning will be necessary for navigating in complex environments. One approach to integrating high-level path planning with the SSA has been proposed by Krogh and Thorpe [23].
- Use of *a priori* knowledge in the SSA. Prior knowledge about the environment might be used to direct the selection of obstacle edges in Step 4 of the SSA. Currently we use the heuristic of selecting the edge closest to the final goal. We note that the convergence results of Section IV are independent of the rule used for selecting edges.

Appendix

Summary of Notation and Function Definitions

Notation

NIL	Empty set or vacuous result.
ϵ	Distance between a subgoal g and its associated edge e.
ab	Straight line (unbounded) through points a and b.
$[ab$	Ray from point a through point b containing point a.
$(ab]$	Line segment from point a to point b containing b but not a.
κ	Minimum distance between obstacles.
(g, e)	Subgoal-edge pair currently being pursued.
(g^T, e^T)	Subgoal-edge pair on top of stack S.
h	Most recent extreme subgoal generated in Step 4.
p_0	Initial position.
p_g	Final goal position.
p_k	Position at beginning of the kth execution of the SSA.
x_k	Dynamic state of the AMR at beginning of the kth iteration.
$C(a_1 \cdots, a_n)$	Open convex hull of points $a_1, \cdots, a_n$.
$E(a)$	Set of obstacle edges in $O(a)$, that is, extreme points of the connected obstacle faces in $O(a)$.
$E'(a)$	Set of visible edges in $E(a)$.
L	Set of extreme subgoals generated in Step 4 of SSA.
$O(a)$	Local map: set of obstacle faces (line segments) visible from point a.
S	Stack of subgoal-edge pairs maintained by SSA.

Functions

CHOOSE_EDGE $\langle$OBS $\langle p_k, p_g\rangle\rangle$	Returns subgoal-edge pair (g', e') with e' one of the two edges in $E(p_k)$ on OBS $\langle p_k, p_g\rangle$ and g' a distance E from e' on extension of the fact terminating at e'.
CREATEG $\langle p_k, (g^T, e^T), (g, e), [ab], E'(p_k)\rangle$	Generates subgoal-edge pairs in Step 6, as defined in Section III.
EXTEND_FACE$\langle O(p_k), (g, e)\rangle$	Returns subgoal-edge pair (g', e') where e' is the vertex (extreme point) of the face in $O(p_k)$ containing e in direction of g, and g' is defined as in CHOOSE_EDGE.
HEIGHT $\langle S\rangle$	Number of subgoal-edge pairs in stack S.
MIN_DIST $\langle A, [ab]\rangle$	Chooses one point in set A closest to line $[ab]$.
NEXT_VERTEX $\langle O(p_k), (g, e)\rangle$	Returns subgoal-edge pair (g', e') with e' as the next vertex on obstacle in $O(p_k)$ containing e in the direction of g and g' a distance ϵ from e' on extension of face $[ee']$.
OBS $\langle a, b\rangle$	Set of obstacle faces in local map $O(a)$ connected to and including a face intersected by line segment (ab).
OTHER_EDGE $\langle O(p_k), (g, e)\rangle$	Returns subgoal-edge pair (g', e') where e' is the other edge in $E(p_k)$ on the obstacles in $O(p_k)$ with edge e', and g' is defined as in CHOOSE_EDGE.
OTHER_VERTEX $\langle O(p_k), (g, e)\rangle$	Returns subgoal-edge pair (g', e') where e' is the other vertex (extreme point) for the face in $O(p)$ containing e, and g' is defined as in CHOOSE_EDGE.
POP $\langle S\rangle$	Removes and returns top subgoal-edge pair from stack S.
PUSH $\langle g, e), S\rangle$	Puts subgoal-edge pair (g, e) on stack S.
RESTORE $\langle S, (ge), [ab], L, h\rangle$	Sets arguments equal to values store by SAVE.
SAFE $\langle g, x(t), O(p_k)\rangle$	Returns logical TRUE if the SCA can bring the AMR to a strop at g from state $x(t)$ along a path in the visible obstacle-free space in $O(p_k)$.
SAVE $\langle S, (g, e), [ab], L, h\rangle$	Stores context of SSA.
SUBGOAL $\langle e, [ab]\rangle$	Generates a subgoal g a distance ϵ from e on the line perpendicular to, and in the direction of, line segment $[ab]$.

References

[1] G. Giralt, R. Chatila, and M. Vaisset, "An integrated and motion control system for autonomous multisensory mobile robots," presented at the 1st Int. Symp. Robotic Research, Bretton Woods, NH, Sept. 1983.

[2] H. P. Moravec, "The Stanford cart and the CMU rover," *Proc. IEEE*, vol. 71, no. 7, July 1983.

[3] A. M. Parodi, J. J. Nitao, and L. S. McTamaney, "An intelligent system for an autonomous vehicle," in *Proc. 1986 IEEE Int. Conf. Robotics Automat.*, San Francisco, CA, Apr. 1986, pp. 1657–1663.

[4] R. Wallace *et al.*, "Progress in robot road-following," *IEEE Int. Conf. Robotics Automat.*, San Francisco, CA, Apr. 1986, pp. 1615–1621.

[5] S. K. Kambhampati and L. S. Davis, "Multiresolution path planning for mobile robots," *IEEE J. Robotics Automat.*, vol. RA-2, pp. 135–145, Sept. 1986.

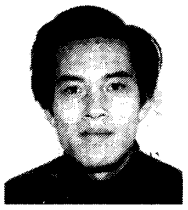
[6] C. E. Thorpe, "Path relaxation: Path planning for a mobile robot," in *Proc. AAAI Conf.*, Aug. 1984.

[7] E. Koch *et al.*, "Simulation of path planning for a system with vision and map updating," in *Proc. 1985 IEEE Int. Conf. on Robotics Automat.*, St. Louis, MO, Mar. 1985.
[8] C. K. Yap, *Algorithmic and Geometric Aspects of Robotics*. Hillsdale, NJ: Lawrence Erlbaum Associates, Advances in Robotics, vol. 1, ch. Algorithmic Motion Planning, 1987.
[9] B. J. Oommen *et al.*, "Robot navigation in unknown terrains using learned visibility graphs. Part I: The disjoint convex obstacle case," *IEEE J. Robotics Automat.*, vol. RA-3, pp. 672–681, Dec. 1987.
[10] D. W. Payton, "An architecture for reflexive autonomous vehicle control," in *Proc. 1986 IEEE Int. Conf. Robotics Automat.*, San Francisco, CA, Apr. 1986, pp. 1838–1845.
[11] J. J. Nitao and A. M. Parodi, "A real-time reflexive pilot for an autonomous land vehicle," *IEEE Contr. Syst. Mag.*, vol. 6, pp. 14–23, Feb. 1986.
[12] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robotics Automat.*, vol. RA-2, pp. 14–23, Mar. 1986.
[13] A. M. Waxman *et al.*, "A visual navigation system for autonomous land vehicles," *IEEE J. Robotics Automat.*, vol. RA-3, pp. 124–141, Apr. 1987.
[14] V. J. Lumelsky and A. A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment," *IEEE Trans Automat. Contr.*, vol. AC-31, pp. 1058–1063, Nov. 1986.
[15] V. J. Lumelsky and A. A. Stepanov, "Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, 1987.
[16] T. Kanade, C. Thorpe, and W. Whittaker, "Autonomous land vehicles project at CMU," in *Proc. 1986 ACM Comput. Conf.*, Cincinnati, OH, Feb. 1986.
[17] B. H. Krogh, "A generalized potential field approach to obstacle avoidance control," in *Proc. Robotics Int. Robotics Research Conf.*, Bethlehem, PA, Aug. 1984.
[18] D. Feng, "Dynamic steering control," Master's thesis, Dep. Elec. Comput. Eng., Carnegie-Mellon Univ., Pittsburgh, PA, Mar. 1986.
[19] R. Aggarwal and G. Leitmann, "Avoidance control," *ASME J. Dynam. Syst. Meas. Contr.*, pp. 152–154, June 1972.
[20] B. R. Barmish, W. E. Schmitendorf, and G. Leitmann, "A note on avoidance control," *ASME J. Dynam. Syst., Meas. Contr.*, pp. 69–70, Mar. 1981.
[21] W. E. Schmitendorf, B. R. Barmish, and B. S. Elenbogen, "Guaranteed avoidance control and holding control," *ASME J. Dynam. Syst., Meas. Contr.*, pp. 166–172, June 1982.
[22] G. Leitmann, "Guaranteed avoidance strategies," *J. Optimiz. Theory Appl.*, vol. 32, Dec. 1980.
[23] B. H. Krogh and C. E. Thorpe, "Integrated path planning and dynamic steering control for autonomous vehicles," in *Proc. IEEE Int. Conf. Robot. Automat.*, San Francisco, CA, Apr. 1986, pp. 1664–1669.

Bruce H. Krogh (S'82–M'82) received the B.S. degree in mathematics and physics from Wheaton College, Wheaton, IL, in 1975, and the M.S. and Ph.D. degrees in electrical engineering from the University of Illinois, Urbana, in 1978 and 1983, respectively.

He is currently an Associate Professor in the Department of Electrical and Computer Engineering, Carnegie-Mellon University, Pittsburgh, PA, where he conducts research in feedback algorithms for supervisory control of robotic systems and synthesis of control logic for discrete-event systems.

Dr. Krogh is an Associate Editor of the IEEE TRANSACTIONS ON AUTOMATIC CONTROL and received the Presidential Young Investigator Award from the National Science Foundation in 1985.



Dai Feng (S'83) was born in Beijing, China, in 1961. He received the B.S. degree in electrical engineering from Grove City College, Grove City, PA, in 1984 and the M.S. degree in electrical engineering from Carnegie-Mellon University, Pittsburgh, PA, in 1986.

He is currently working towards the Ph.D. degree at Carnegie-Mellon University. Since 1984 he has been a Research Assistant in the Department of Electrical and Computer Engineering and the Robotics Institute, Carnegie-Mellon University.