An Autonomous Sensor-Based Path-Planner for Planetary Microrovers

S. L. Laubach
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

J. W. Burdick
Department of Mechanical Engineering
California Institute of Technology
Pasadena, CA 91125

Abstract

With the success of Mars Pathfinder's Sojourner rover, a new era of planetary exploration has opened, with demand for highly capable mobile robots. These robots must be able to traverse long distances over rough, unknown terrain autonomously, under severe resource constraints. Based on the authors' firsthand experience with the Mars Pathfinder mission, this paper reviews issues which are critical for successful autonomous navigation of planetary rovers. We next report on the "Wedgebug" algorithm for planetary rover navigation. This algorithm is complete, correct, requires minimal memory for storage of its world model, and uses only on-board sensors, which are guided by the algorithm to efficiently sense only the data needed for motion planning. The implementation of a version of Wedgebug on the Rocky7 Mars Rover prototype at the Jet Propulsion Laboratory (JPL) is described, and experimental results from operation in simulated martian terrain are presented.

1 Introduction

The recent Mars Pathfinder experience vividly illustrated the benefits of including a mobile robotic explorer on a planetary mission. Previous forays allowed scientists to explore planets remotely, via an orbiter, or were limited to a single site for study with a lander's instruments. However, the Sojourner rover, carried to Mars by the Pathfinder spacecraft, was able to roam and to place its instruments (a spectrometer and low-mounted cameras) directly on or near objects of interest. In all, the Sojourner rover ranged over an area roughly 20 meters square, conducted soil experiments in a variety of terrains, and sampled the spectra of 16 distinct targets [6]. Missions currently being planned call for new rovers to be sent to Mars at launch opportunities in 2001, 2003, and 2005, as well as a nanorover to be sent to the surface of an asteroid in 2003. Many of these missions require the rovers to operate for up to a year, compared with the 83 sols (martian days) of operation for the Sojourner rover. The rovers are also required to traverse vastly greater distances: up to 100 m/sol, as opposed to Sojourner's 104 m/83 sols. In addition, lessons learned from Mars Pathfinder indicate a need for significantly increased rover autonomy in order to meet mission criteria, within severe constraints including limited communication opportunities with Earth, power, and computational capacity.

1.1 Motion Planning on Mars

A key advance in functionality required for planetary rovers is greater navigational autonomy. Each rover will be working in unknown, rough terrain. (The resolution expected from Mars orbiters, for example, is roughly 300 meters/pixel, with only isolated "postage stamp" regions achieving the highest resolution of 1.4 m/pixel [7]. Orbiter camera pointing limitations prohibit attempting to use these highest-resolution images for rover navigation or localisation.) Given a distant (i.e., not immediately visible by the rover's sensors) goal designated by Earth-based operators, the rover must use its sensors to navigate safely and autonomously to that goal. Rather than address all of the issues which arise in this complex problem, this paper will focus on the aspects relevant to autonomous path planning.

Useful motion planners for planetary rovers have several key characteristics: they must assume no prior knowledge of the environment, must be sensor-based, robust, complete and correct. They must also operate under severe constraints of power, computational capacity, and the high cost of flight components, which translates into limited memory available on-board the rover. Due to dead reckoning errors, slippage on rough/loose substrate, nonholonomic fine-positioning constraints, and constraints on mission time available, using rover motion to augment sensing is costly. Simultaneously, limited memory, computational capacity, power and time available all argue for minimising the amount of data sensed and processed. Thus, a practical motion planner must utilise the available sensing array in a scheme which efficiently senses only the data needed for motion planning, requires minimal memory to store salient features of the environment, and conserves rover motion.

2 Relevant Work

Much of the body of work in motion planning can be divided into three major categories: "classical" path planners, heuristic planners, and "complete and correct" sensor-based motion planners. "Classical" planners assume full knowledge of the environment, and are complete. Heuristic planners, generally based upon a set of "behaviours," can be used in unknown environments but do not guarantee the goal will ever be reached, nor that the algorithm will halt. (A more detailed discussion is presented in [5].) The third category, which relies solely upon the rover's sensors and yet guarantees completeness, is most relevant to the problem of autonomous planetary motion planning.

Two distinct approaches to such planners have been explored, both of which adapt classical methods to a local sensed region. One set of methods incrementally builds "roadmaps" within the visible region, such as Choset's HGVG [2], Rimon's adaptation of Canny's OPP [1], and the "Tangent Bug" algorithm of Kamon, Rivlin, and Rimon [3]. The other approach springs from approximate cell decomposition, filling in a grid-based world model as more information is gathered, exemplified by Stentz' D* algorithm [9].

The above methods have each been developed to differing degrees in their application to real systems. For example, the sensor-based version of OPP is currently strictly theoretical, owing to the difficult-to-implement nature of the sensors required. The HGVG, on the other hand, has been implemented on a mobile robot using range sensors. Choset's planner produces paths which are maximally distant from obstacles, a plus for rover safety. However, it works best in contained environments with well-defined corridors; a description not applicable to the typical martian environment (Fig. 1).

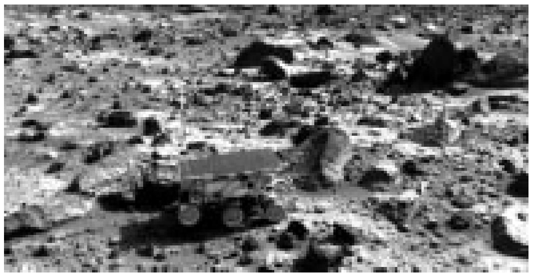

Figure 1: Typical terrain encountered on Mars by the Sojourner rover. The intrepid mobile explorer is 68cm long by 48cm wide, and stands 28cm tall.

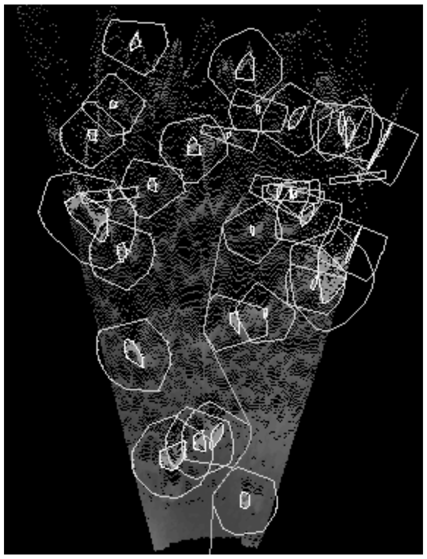

Figure 2: Rangemap of a single image from a stereo pair. This image also shows obstacles detected within the visible region, and a path generated by the implementation of the "RoverBug" algorithm on Rocky7 (see Section 5).

The D* algorithm and Tangent Bug both are useful in unbounded environments. In addition, they both produce "locally optimal" solutions, that is, the resultant paths are the shortest length possible given the use of solely local information. D* has in particular been implemented on a real world system (an autonomous HMMWV driven in a slag heap near Pittsburgh). However, the grid-based world model requires a significant amount of memory for storage, and the algorithm's completeness depends entirely upon the precision of its world model, which is determined by cell granularity.

Tangent Bug provides the motivation for the work presented here. Its world model is streamlined, consisting only of sensed obstacle boundary endpoints. The planner itself consists of two "modes"— motion-to-goal, and boundary following—which interact incrementally to ensure global convergence (if the goal is reachable), and which "fail gracefully" if the goal is found to be out of reach. Thus, the algorithm is memory-efficient, fairly robust, and conserves robot motion. However, some of its assumptions do not apply to the "rover problem" of navigating in planetary terrain. For example, Tangent Bug assumes that the robot is modelled as a point, and that obstacles block both motion and sensing. In addition, Tangent Bug assumes that the robot's sensor provides an omnidirectional view.

The current scenario for a rover sensing system consists of a stereo pair of cameras mounted on a pan-able mast. Typically, these cameras have a 30° to 45° field of view (FOV), and the "visible region" connected with these sensors sweeps out roughly a wedge, with limited downrange radius R due to both viewing angle (tilt) and feature resolving ability. (See Fig. 2 for an example of data from such a sensing array.) Camera pixels imaging features closer to the horizon (hence farther away) have a larger footprint than pixels imaging the foreground; simultaneously, obstacles further away are apparently smaller in relative size. These two properties combine to limit the range at which a stereo pair can resolve obstacles of a given height, for instance. From the discussion in Section 1.1, it is clear that it is important to not simply pan the sensor array and obtain an omnidirectional view at every step. Rather, the planner should be able to identify the minimal number of sensor scans needed—and which specific areas to scan—to proceed at each step, while avoiding unnecessary rover motion. Thus, we have developed the "Wedgebug" algorithm to address the shortcomings of Tangent Bug, as a step towards a more practical path planner for flight microrovers. Wedgebug is complete, correct, and relies solely upon the robot's sensors. The implementation discussed in Section 5 relaxes the assumption that the rover is a point robot. Perhaps most importantly, Wedgebug deals with the limited FOV of flight rovers in a manner which is efficient and minimises the need to sense and store data, using autonomous gaze control.

Section 3 presents the Wedgebug algorithm in some detail. Section 4 develops the proof of completeness for this motion planner. Section 5 describes briefly the current implementation of an extended Wedgebug on a prototype microrover at JPL, with experimental results. Section 6 contains concluding remarks.

3 The Wedgebug Algorithm

The basic assumptions of the Wedgebug algorithm are as follows: The rover is modelled as a point robot in a 2D binary environment (i.e., every point in the environment is either contained within an impassable obstacle, or lies in freespace). Obstacle boundaries block sensing as well as motion. (In Section 5, we discuss how the implementation deals with the fact that the real robot is not a point robot, and that obstacles do not necessarily block sensing.) The rover's sensors, from position x, detect ranges within a wedge $W(x,\vec{v})$

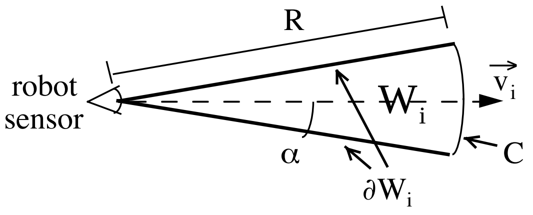

Figure 3: Anatomy of a wedge.

of radius R, which sweeps out an angle 2α (> 0) and is centered on the direction $\vec{v}$. Define C as the arc boundary of $W(x,\vec{v})$ at radius R, and $\partial W(x,\vec{v})$ as the union of the two bounding rays of $W(x,\vec{v})$ (Fig. 3). We further define the "interior" of $W(x,\vec{v})$ as $\text{int}(W(x,\vec{v})) = \overline{W(x,\vec{v})} - \partial W(x,\vec{v})$ (N.B., an "interior" point may lie on C). Let $\mathrm{d}(a,b)$ be the Euclidean distance between points a and b.

Wedgebug, like Tangent Bug, is based upon two modes which interact to ensure global convergence: *motion-to-goal* (*MtG*) and *boundary following* (*BF*). However, each mode is more finely divided into components that improve efficiency and handle the limited FOV. A high-level sketch of the operation of the Wedgebug algorithm follows: At the beginning of the path sequence, an initialisation step records the parameter $\mathrm{d}_{\text{LEAVE}} = \mathrm{d}(A,T)$, where A is the robot's initial position, and T is the goal. This parameter marks the largest distance the robot can stray from T during an *MtG* segment.

MtG is typically the dominant behaviour. It basically directs the robot to move towards the goal using a local version of the tangent graph, restricted to the visible region (Fig.4). The tangent graph (also known as the "reduced visibility graph") consists of all line segments in freespace connecting the initial position, the goal, and all obstacle vertices, such that the segments are tangent to any obstacles they encounter [4]. Let LTG(S) be the *local tangent graph* within the set S, defined as the tangent graph restricted to S. In our case, obstacle boundaries appear as continuous contours in the range data; the endpoints of these contours are called the "obstacle vertices". Each endpoint e corresponds to a discontinuity in the range data or to an intersection of a contour with ∂W or C.

MtG works roughly as follows: The robot (at position x) first senses a wedge, $W_0 = W(x,\vec{v}_0)$, where $\vec{v}_0 = \overrightarrow{xT}$ is the vector from x to the goal. (All wedges in the subsequent discussion are assumed to subsume a half-angle α.) The planner constructs LTG(W_0). If there are no visible obstacles intersecting the ray $\vec{xT}$, the planner adds a node T_g to LTG(W_0) at a distance R from x along $\vec{xT}$, so LTG(W_0) contains a

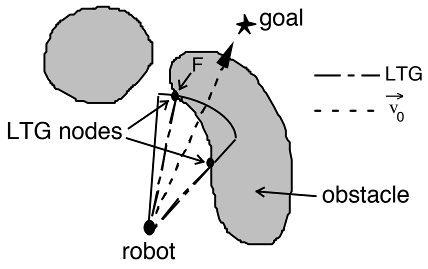

Figure 4: LTG calculated within $W(x, \vec{v}_0)$.

path directly towards T. The planner then searches a subgraph, $G1(W_0) = \{V \in \mathrm{LTG}(W_0) | \, \mathrm{d}(V,T) \leq \min(\mathrm{d}(x,T), \mathrm{d}_{\text{LEAVE}})\}$, for the optimal local subpath. Using the criterion discussed in Section 3.1, the rover may scan additional wedges as needed, and constructs the LTG in the conglomerate wedge $\overline{W}(x)$ (see Section 3.1). After executing this subpath, *MtG* begins anew. This behaviour is continued until either the goal is reached, T is deemed unreachable, or the robot encounters a local minimum in $\mathrm{d}(\cdot, T)$. In the latter case, the planner switches to *BF*. The objective of this mode is to skirt the boundary of the *blocking obstacle* (the obstacle whose boundary contains the local minimum), still calculating $\mathrm{LTG}(W_0)$, until one of two events occur: either the robot completes a loop, in which case the goal is unreachable and the algorithm halts; or $\mathrm{LTG}(W_0)$ contains a new subpath toward T. The next two sections describe the *MtG* and *BF* modes in more detail. (Of note, no information, other than explicitly recorded points and parameters, are passed between steps.)

3.1 Motion-to-Goal

The basic idea of *MtG* is for the robot to progress toward the goal in such a manner that its distance to T is nonincreasing. During this type of behaviour, there are two situations: either the robot can move directly through freespace toward the goal ("direct mode"), or it must skirt the boundary of a *blocking obstacle* while still decreasing its distance from T ("sliding mode"). Further, "sliding mode" contains another submode, "virtual *MtG*", in order to improve efficiency. That is, during normal *MtG*, the planner scans a single wedge view toward the goal to determine whether a path exists. If it is apparent that more information could lead to a shortcut around an obstacle boundary, "virtual *MtG*" scans additional wedges in order to determine the appropriate shortcut, and therefore improve efficiency.

The first actions taken in a new *MtG* segment are to scan W_0, construct $\mathrm{LTG}(W_0)$, and search $G1(W_0)$. It can be shown that the shortest possible path will pass through either T_g (if $T_g \in G1(W_0)$), or an endpoint e of a *blocking obstacle* (which intersects the ray $\overrightarrow{xT}$); call the point through which the shortest path passes the *focus point*, F (Fig 4). The focus point (fixed for each step) serves as the goal for each *MtG* step—the subpath for the current *MtG* step is precisely $\overline{xF}$. Its position within the robot's FOV also determines whether additional wedge views are needed.

If $F = T_g$, the robot simply executes the subpath to F, and starts the next *MtG* step. We call this case a *direct MtG* segment, since the robot proceeds through freespace directly towards the goal.

Otherwise, the robot has encountered a blocking obstacle, O, which it must skirt in order to continue towards the goal, in which case *MtG* enters "sliding mode". To ensure that the robot does not backtrack, the planner establishes a traversal direction—call it *positive* (ρ^+)—upon first sensing O. Thereafter, the robot must satisfy the *sliding condition*: it must traverse in the positive direction around O, and may not change direction while following a single (sensed) obstacle's boundary. In subsequent *MtG* steps, after determining the shortest path in $G1 \cup T$, F is chosen as follows: If $F \notin \partial O$, or if F is the sensed endpoint of ∂O in the positive direction (e^+), F is unchanged. Otherwise, F is changed to e^+. At the start of "sliding mode", the planner also records V_{loop}, the sensed endpoint of ∂O in the *negative* direction (e^-). "Sliding mode" ends when (1) the blocking obstacle is changed (including the case when $F = T_g$), (2) the planner switches to *BF*, or (3) the robot detects that it has circumnavigated O. Below, we delineate the actions of "sliding mode", as determined by the position of F within the visible region:

Case 1: If $F \in \mathrm{int}(W(x, \vec{v}_0))$, there are two cases to consider: either the ray $\overrightarrow{xF}$ is tangent to O, or another obstacle obscures the blocking obstacle's boundary. In the first case, after recording ρ^+ and V_{loop}, the robot simply executes the subpath to F, and starts the next *MtG* step. This is the case illustrated in Fig. 4. (For purposes of the proof to be given later, the robot never lies directly on ∂O, but rather remains a distance ε away.) In the second case, it is possible that the second obstacle is hiding a local minimum. We will return to this case later.

Case 2: If, on the other hand, $F \in \partial W(x, \vec{v}_0)$, the planner must inspect the tangent to ∂O at F, $\vec{t}_F$, to determine if the robot will be "sliding around" the blocking obstacle, or if it has possibly encountered a local minimum in $\mathrm{d}(\cdot, T)$. If $\vec{t}_F \cdot \overrightarrow{xT} \leq 0$, the robot would need to increase its distance from the goal to

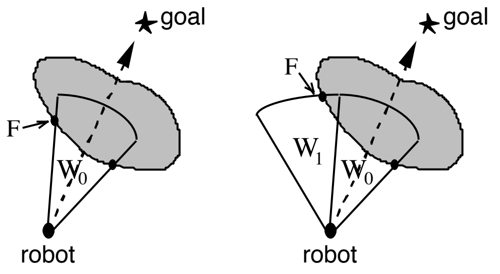

Figure 5: "Virtual *MtG*." The figure on the left depicts the first part of an *MtG* step. The nodes of LTG(W_0) are marked. F satisfies the conditions for "virtual *MtG*," so the robot scans W_1 (right). Now, $F \in \mathrm{int}(W_0 \cup W_1)$, so "virtual *MtG*" ends.

skirt the obstacle on the subsequent step. So, if allowed, the planner changes F to the opposite sensed endpoint of ∂O, and tests the new F'. Changing F is not allowed if (1) F has already been changed once at x, or (2) the change would violate the sliding condition. If then $F' \in \partial W(x, \vec{v}_0)$ and $\vec{t}_{F'} \cdot \overrightarrow{xT} \leq 0$ (or F cannot be changed), the robot has encountered a local minimum in $\mathrm{d}(\cdot, T)$. Thus, the planner switches to *BF* (described in Sect. 3.2).

In the case that $F \in \partial W(x, \overrightarrow{xT})$, but $\vec{t}_F \cdot \overrightarrow{xT} > 0$, the robot must "slide around" the obstacle while progressing toward T. Unfortunately, being close to an obstacle restricts the robot's already-limited view and can result in tiny incremental steps. Thus, in order to efficiently acquire data from the robot's current position and to avoid as much inefficient motion as possible, we add a submode of *MtG*, called "virtual *MtG*". The object of "virtual *MtG*" is to sense additional wedges in the direction the robot will "slide around" the obstacle, and to generate a local shortcut in the robot path.

The "virtual *MtG*" mode directs the sensor to pan towards F (defining this direction of rotation *positive*, if not already defined), and to sense the wedge $W_1 = W(x, \vec{v}_1)$, where $\angle(\overrightarrow{xT}, \vec{v}_k) = 2k\alpha$ (that is, W_1 abuts W_0 at F). Let the conglomerate wedge $\overline{W} = W_0 \cup W_1$ (in general, at each position x, $\overline{W} = \bigcup^{sensed} W_k(x)$). The planner computes $G1(\overline{W})$, and finds the new focus point F (according to whether the planner is in "sliding mode"). Let $\partial\overline{W}^+$ be the bounding ray $\vec{r}$ such that $\angle(\overrightarrow{xT}, \vec{r}) > 0$ (i.e., the edge of $\overline{W}$ in the positive direction). If $F \in \partial\overline{W}^+$, "virtual *MtG*" is repeated. This mode ends if one of three conditions is met:

1. $F \in \mathrm{int}(\overline{W})$, in which case the robot has found a suitable shortcut. The robot records ρ^+ and V_{loop} (if needed), executes the subpath to F, and begins a new *MtG* iteration.
2. $\angle(\overline{xT}, \partial\overline{W}^+) \geq \pi/2$, which means that the rover is sensing part of a region not useful for *MtG*, since G1 contains only nodes closer to T than the robot's current position.
3. $\vec{t}_F \cdot \overrightarrow{xT} \leq 0$, which indicates that the obstacle boundary is curving back toward x, that is, the robot can no longer "virtually slide" in this direction without losing ground.

In fact, (2) $\Longrightarrow$ (3). In these cases, if allowed, the robot changes F, and (if F' meets the conditions), attempts "virtual *MtG*" again. If the second attempt fails, the robot has encountered a local minimum in $\mathrm{d}(\cdot, T)$, and the planner switches to *BF*.

We now return to the case $F \in \mathrm{int}(W(x, \vec{v}_0))$, where a second obstacle obscures the blocking obstacle boundary. As noted, the obscuring obstacle may be hiding a local minimum. Thus, as in the case when $F \in \partial W(x, \overrightarrow{xT})$, we check the tangent to ∂O at F. If $\vec{t}_F \cdot \overrightarrow{xT} > 0$, "virtual *MtG*" can't help, so the planner records ρ^+ and V_{loop}, executes the subpath, and begins another *MtG* iteration. If $\vec{t}_F \cdot \overrightarrow{xT} \leq 0$, we change F (if allowed), and check the new F'. If on the second time around, $F' \in \partial W(x, \vec{v}_0)$ or the boundary at F' is similarly obscured, and $\vec{t}_{F'} \cdot \overrightarrow{xT} \leq 0$ (or F cannot be changed), the robot has encountered a local minimum and the planner switches to *BF*.

3.2 Boundary Following

The basic idea of *BF* is to skirt the *blocking obstacle* until progress can be made once more toward the goal. As with *MtG*, *BF* is split into two submodes. "Normal *BF*" uses two wedge views, one toward the goal and one in the direction of travel around the obstacle boundary, to determine whether a clear path towards the goal exists while the robot circumnavigates the obstacle. Immediately after a switch from *MtG* to *BF*, however, the robot must determine its direction of travel around O, the blocking obstacle. If ρ^+ is not defined, "virtual *BF*" is used to take full advantage of the information which can be gleaned at the current distance from the obstacle (arguably more than from a closer range), to choose this direction efficiently. (The primary motivation for "virtual *BF*" is the idea that it is less costly for the robot to swivel its sensors than for the robot to actually move.) In essence, the robot will swing its sensor array back and forth in a prescribed manner, to search for the "best" place to move and begin "normal *BF*".

More precisely, the robot initially scans the wedge $W_1 = W(x, \vec{v}_1)$, where in this case the positive direction is chosen by comparing the tangents to ∂O at

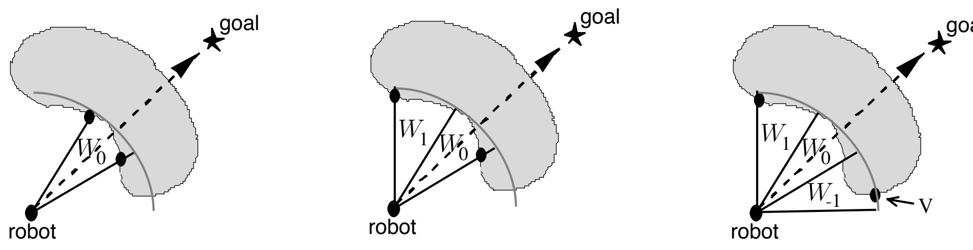

Figure 6: "Virtual *BF*." The figure on the left depicts the first part of a "virtual *BF*" step. The nodes of $\mathrm{LTG}(W_0)$ are marked with black circles. Since $\nexists V \in \mathrm{int}(\mathrm{LTG}(W_0))$, the robot scans W_1 (center). Again, $\nexists V \in \mathrm{int}(\mathrm{LTG}(W_0 \cup W_1))$, so the robot scans W_{-1} (right). Now, $V \in \mathrm{int}(W_0 \cup W_1 \cup W_{-1})$, so "virtual *BF*" ends.

the intersection with ∂W_0; that is, if $\vec{t_l}, \vec{t_r}$ are the two tangents (at intersection points e_l (left) and e_r (right), respectively), then if $\vec{t_l} \cdot \vec{v}_0 \geq \vec{t_r} \cdot \vec{v}_0$, then $\angle(\vec{v}_0, \overrightarrow{xe_l}) > 0$. As before, let $\overline{W} = \bigcup^{sensed} W_k(x)$. The planner computes $\mathrm{LTG}(\overline{W})$. If $\exists$ a node $V \in \mathrm{LTG}(\overline{W}) \cap \partial O$ such that $V \in \mathrm{int}(\overline{W})$, the robot moves to V and begins "normal *BF*", first recording two features: d_{reach}, the closest distance to T encountered so far on ∂O, and $V_{loop} = \partial \overline{W}^- \cap \partial O$ (as well as ρ^+). If there is no such node V, the planner directs the sensor to scan $W_{-1} = W(x, \vec{v}_{-1})$, constructs $\overline{W} = W_0 \cup W_1 \cup W_{-1}$, and searches the freshly expanded $\mathrm{LTG}(\overline{W})$. In this manner, the robot scans back and forth until a suitable node is found, then travels there to begin "normal *BF*."

"Virtual *BF*" ends when one of three events are detected:

1. $\exists V \in \mathrm{LTG}(\overline{W}) \cap \partial O$ such that $V \in \mathrm{int}(\overline{W})$. The robot moves to V, and begins normal *BF*.

2. The latest scanned wedge overlaps a previously scanned region (i.e., $|\angle(\vec{v}_0, \vec{v}_{last})| > \pi$). In this case, the robot is trapped by an encircling obstacle, and the algorithm halts.

3. $\exists V \in \mathrm{LTG}(\overline{W})$ with $V \in \mathrm{int}(\overline{W})$, but $V \notin \partial O$. In this case, an obstacle obscures the blocking obstacle boundary. We call V a "framing point," since it "frames" the sensed extent of ∂O. If only one "framing point" is in view, the robot scans once more in the opposing direction, and then no matter the outcome, "virtual *BF*" ends. If a node as in item (1) is found, the robot moves there and begins normal *BF*. Otherwise, the rover moves to the point on ∂O just before V. If there are two "framing points", V_l and V_r, the rover moves to ∂O near V_l iff $|\angle(\vec{v}_0, \overrightarrow{xV_l})| > |\angle(\vec{v}_0, \overrightarrow{xV_r})|$). At this point, the rover begins normal *BF*.

In normal *BF*, at the start of each step, the robot senses W_0, and searches $G1(W_0)$. *BF* exits here if: (1) $T \in W_0$, in which case the robot moves to T and the algorithm halts, or (2) $\exists V \in G1(W_0)$ such that $\mathrm{d}(V,T) < \mathrm{d}_{reach}$, the *leaving condition*, in which case the planner resets $\mathrm{d}_{\text{LEAVE}}$ to $\mathrm{d}(V,T)$, and begins a new *MtG* segment. If neither of these conditions hold, the planner computes $\vec{t_x}$, the tangent to ∂O at x, and directs the sensor to scan $W(x, \vec{t_x})$. If $V_{loop} \in W(x, \vec{t_x})$, and $V_{loop} \in$ the connected portion of ∂O containing x, the robot has executed a loop—therefore, the goal is unreachable, and the algorithm halts. Otherwise, the planner computes $V \in \partial O \cap \mathrm{LTG}(W(x, \vec{t_x}))$ such that $\mathrm{d}(x,V) > \mathrm{d}(x,V')\ \forall V' \in \partial O \cap \mathrm{LTG}(W(x, \vec{t_x}))$. The robot records d_{reach}, executes this subpath, then begins a new *BF* step.

The Wedgebug algorithm thus deals with the limited FOV of the robot in an efficient manner. The "virtual" submodes both take advantage of the lower cost of panning the sensor array over actual motion, while minimising the number of views required at each step.

4 Sketch of Proof of Convergence

The proof of convergence of the Wedgebug algorithm is analagous to the Tangent Bug convergence proof [3]. The sketch of the proof is as follows: Each robot motion can be characterised as a particular type of motion segment. In turn, each type of segment can be shown to have finite length. Following Kamon, Rivlin, and Rimon, it can be shown that there are a finite number of each type of segment, and thus the path terminates after finite length. Due to space limitations, we will detail here only the proof that *BF* segments have finite length. The proofs for the other types of motion segments are analagous.

Define the points S_i to be the points where the planner switches from *MtG* to *BF*; L_i the point where the *BF* leaving condition is met on obstacle i (switch point from *BF* to *MtG*); and finally Q_i, the point where a loop is detected on obstacle i (Q_i is within distance R of V_{loop}, where R is the sensor range). Then, there are two types of *BF* segments: $[S_i, L_i]$, and $[S_i, Q_i]$. Let P_i be the perimeter of obstacle O_i.

Lemma. *BF segments are finite length.*

Proof. Two cases: a) $[S_i, L_i]$. Let N designate the point where the robot actually touches ∂O_i during this *BF* segment. The path, then, consists of two pieces: $[S_i, N]$, and $[N, L_i]$. Since N, and L_i both lie on ∂Oi, the robot is traversing O_i in a fixed direction, and the robot has not detected a loop, we have $\text{length}([N, L_i]) \le \text{length}([N, Q_i]) \le \text{length}([Q_i, V_{loop}])$. Further, since V_{loop} is defined as the sensed endpoint of O_i in the opposite direction from traversal, we know that the segments $\overline{NV_{loop}}$ and $\overline{V_{loop}N}$ do not overlap. Therefore, since the obstacle perimeter, P_i, is finite, we have $\text{length}([N, L_i]) \le \text{length}([N, V_{loop}]) \le P_i < \infty$. We also have that $\text{d}(S_i, N) \le R$, and $R < \infty$ by definition. Thus, this segment is bounded by $\text{length}([S_i, L_i]) \le \text{d}(S_i, N) + \text{length}([N, L_i]) \le R + P_i < \infty$.

b) $[S_i, Q_i]$. Similarly, $\text{length}([S_i, Q_i]) \le \text{d}(S_i, N) + P_i \le R + P_i < \infty$. □

5 Implementation and Results

An extended version of the Wedgebug algorithm, called "RoverBug," has been implemented on the JPL Rocky7 prototype microrover (Fig 7), a research vehicle designed to test technologies for future missions [10]. The vehicle is roughly the same size as the Sojourner rover, with a few important differences which will come into play in future rovers. (Refer to [5], [10] for a fuller description.) Like Sojourner, Rocky7's mobility system is a rocker-bogie suspension, capable of surmounting obstacles $1\frac{1}{2}$ wheel diameters tall. However, Rocky7 boasts three stereo pairs of cameras for navigation (two body mounted, and one on a deployable 1.2m mast) as opposed to Sojourner's body-mounted laser striping system. In addition, the rover software features a recently-developed localisation algorithm utilising mast imagery to aid in dead-reckoning [8].

Although the Wedgebug algorithm is an important step, it still does not quite capture the complexities of the real world. For instance, the rover is not a point robot; a problem addressed in the "RoverBug" implementation by calculating the obstacles' "silhouettes": the smallest polygon bounding the projection of each $SE(2)$ obstacle onto $\Re^2$. Another issue is the fact that the mast imagery can "see over" many obstacles: the resulting visibility polygon is not a star-shaped set, and the LTG is much richer than in the development in Section 3. Also, the mast is limited in its ability to sense obstacles within 1m of the rover, since the obstacle detection algorithm searches for steps in elevation, not easy to detect while looking straight down on the tops of rocks. Thus, care must be taken while executing the subpaths.

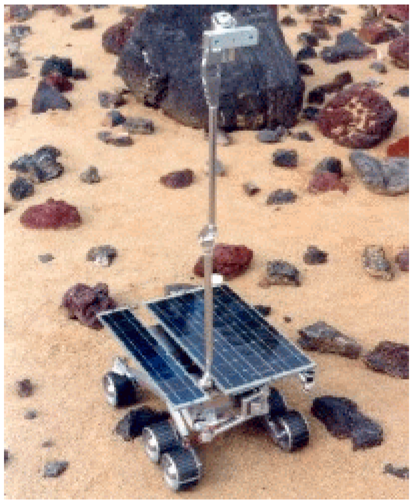
Figure 7: The Rocky7 Prototype Microrover, developed at JPL to test technologies for future missions. It is pictured here in the JPL MarsYard, an outdoor testing arena featuring simulated martian terrain.

In brief, the scenario is as follows: The rover is situated in unknown, rough terrain. The remote human operator designates a goal, which sets in motion the autonomous planner. The planner begins by directing the mast to image towards the goal. Software on-board produces a rangemap, detects obstacles, and computes the obstacles' convex hulls. The planner, which uses a version of the theory described above, then computes the obstacles' silhouettes, and searches the resulting LTG to produce the first subpath. The planner directs the mast to look in the appropriate direction(s), and incrementally builds and executes each subpath until the goal is reached.

The implementation so far has been tested extensively in the JPL MarsYard, as well as in natural arroyo terrain. Fig. 8 shows the results of one typical run in the MarsYard. The goal was approximately 21m distant from the initial position, and R for each wedge was 5m. As in Fig. 2, the convex hulls and silhouettes are computed within each wedge view, and a subpath generated, which is executed before the next wedge view is taken. The resultant multi-step path runs from lower right to upper left.

6 Summary and Conclusions

The requirements for autonomous flight rovers for planetary exploration provide compelling motivation for work in streamlined sensor-based motion planning. This paper continues the work begun in [5] to develop, implement, and test a robust, practical path planner for the Rocky7 prototype microrover. The Wedgebug algorithm is described, along with a sketch of its proof of convergence. A companion paper will describe in more detail the "RoverBug" planner, the Wedgebug extension implemented on Rocky7. These planners significantly augment microrovers' autonomous navigation ability, which in turn will aid in producing successful mobile robot missions.

Acknowledgments

The work described here was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. We would like to acknowledge the Long Range Science Rover team and the Mars Pathfinder Microrover Flight Experiment team, for help, inspiration, and flight experience with a rover. The authors would particularly like to thank Samad Hayati, Andrew Mishkin, Clark Olson, Rich Petras, and Todd Litwin for their invaluable assistance.

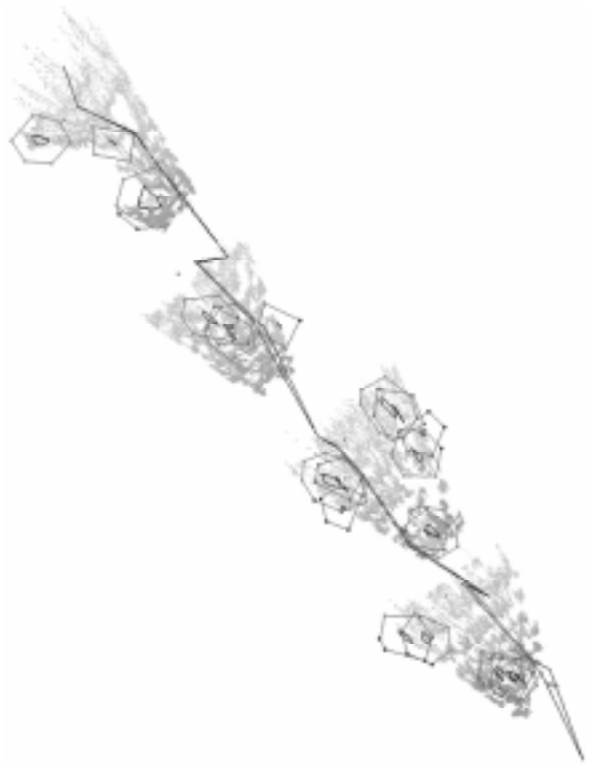

Figure 8: Results from a multi-step run in the JPL MarsYard. The path begins in the lower right corner of the image, toward a goal approx. 21m distant in the upper left. Each wedge depicts a rangemap produced from mast imagery, and extends roughly 5m from the imaging position. The obstacles are marked by a black convex hull, and a grey silhouette. Each subpath ends with an apparent "jag" in the path; these are not in fact motions, but rather the result of the localisation procedure run at the conclusion of each step. The second line echoing the path is the rover's telemetry for the run.

References

[1] E. Rimon and J. Canny, "Construction of C-space Roadmaps From Local Sensory Data: What Should the Sensors Look For?" in *Proc. IEEE Conf. Robotics Automat.*, 1994.

[2] H. Choset, *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph.* Ph.D. thesis, California Inst. of Tech., 1996.

[3] I. Kamon, E. Rimon, and E. Rivlin, "A New Range-Sensor Based Globally Convergent Navigation Algorithm for Mobile Robots," CIS–Center of Intelligent Systems 9517, Computer Science Dept., Technion, Israel, 1995.

[4] J.-C. Latombe, *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[5] S. L. Laubach, J. W. Burdick, and L. H. Matthies, "An Autonomous Path-Planner Implemented on the Rocky7 Prototype Microrover," in *Proc. IEEE Conf. Robotics Automat.*, 1998.

[6] A. Mishkin, J. Morrison, T. Nguyen, H. Stone, and B. Cooper, "Operations and Autonomy of the Mars Pathfinder Microrover," in *Proc. IEEE Aerospace Conf.*, 1998.

[7] *MGS Investigation Description and Science Requirement Document*, JPL Document D-12487, February 1995.

[8] C. Olson and L. Matthies, "Maximum Likelihood Rover Localisation by Matching Range Maps," in *Proc. IEEE Conf. Robotics Automat.*, 1998.

[9] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments," in *Proc. IEEE Conf. Robotics Automat.*, 1994.

[10] R. Volpe, J. Balaram, T. Ohm, and R. Ivlev, "The Rocky7 Mars Rover Prototype," in *Proc. IEEE/RSJ Conf. Intelligent Robots and Sys.*, 1996.