Proc. of IEEE Int'l Conf. on Robotics and Automation, San Diego, CA, May 1994.

Pixel-Based Range Processing for Autonomous Driving

M . Hebert

The Robotics Institute,
Carnegie Mellon University,
Pittsburgh, PA 15213

Abstract

We describe a pixel-based approach to range processing for obstacle detection and autonomous driving as an alternative to the traditional image- or map-based approaches. The pixel-based approach eliminates the delays due to image acquisition and map building and permits the integration of traversability evaluation and path generation into a single module without rhe latency involved in distributed systems. We describe the algorithm used for updating a local map using individual range pixels, for detecting obstacles on the fly, and for generating steering commands. We illustrate the performance of the algorithm using an implementation of a cross-count' driving system using a scanning laser range finder:

1 Introduction

A critical component of mobile robot systems is the ability to detect regions of the environment in which the vehicle should not travel. This is an especially important problem when a robot operates outdoor at relatively high continuous speed. In on-road driving, obstacles have to be detected a long range and very quickly to allow enough reaction time at high speeds. In cross-country driving, unsafe region of the terrain have also to be identified at long range with the added requirement that simple strategies such as elevation thresholds do not work because of the complexity of the terrain. We will refer to regions of the environment in which the vehicle is not allowed to drive as obstacle regions or as untraversable regions without making any distinction.

The basic approach to the problem is illustrated in Figure 1: A Cartesian elevation grid is built from range data and is transformed to a local map of untraversable regions which is used for generating admissible paths. Depending on the system under consideration, the implementation details of the general approach may be different. For example, the local map may be an explicit data structure [8] or it can be implicitly computed as the paths are evaluated over the grid [7][9]. Irrespective of the particular implementation, the main issue is to make the system as effective as possible in detecting obstacles as soon as they appear in the data and in using them in the path calculations as soon as possible after they are detected.

The main issue is that most system are image-based or map-based in that they have to wait until a full range image, or a full map, is processed before reporting any obstacles to the path generation module. In this paper, we propose a way to process range data pixel by pixel (Section 2), updating an internal representation on the fly for each pixel (Section 4) and correcting paths as soon as the obstacles are detected instead of waiting for a full image to be processed (Section 5). In addition, we will show how to interleave pixel processing, local map updating, and arc generation in a single module to eliminate the latency due to communication between modules and show timing results to validate our conclusions (Section 7). We have implemented a driving system based on our approach to pixel-based processing and tested it in a cross-country application. The system is an extension of our previous system for off-road navigation reported in [8]. The current system uses an imaging laser range finder, described in Section 3, as the source of range data.

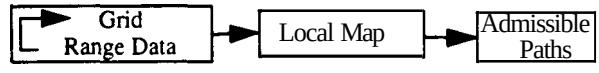

Figure 1: Perception system for autonomous navigation.

2 Pixel-Based Processing: Overview

The traditional approach is illustrated in Figure 2(a). A range image is first acquired from an active range finder or from passive stereo. Then, every range pixel in the image is transformed to local grid coordinates and stored in one cell of the grid. Finally, the grid is traversed and each cell is evaluated for traversability. Once the untraversable cells are identified, they ate used for planning an admissible path. It is useful to view these obstacle cells as being maintained in a local map. This view of the traditional approach corresponds most closely to our earlier implementation described in [8] and in the stereo based system described in [10] and [11]. We call this approach "image-based" processing.

Although this is a natural and convenient way to implement range image processing, it has many drawbacks. The first drawback is the latency in the navigation system due to the fact that the system has to wait for the processing time of a complete image before any result is produced from the perception system. The latency becomes a severe limitation with much faster acquisition rate from both passive stereo and laser range finders on the horizon.

A second problem is the fact that a system using image-based processing is typically highly sensor-dependent. In particular, it is usually tuned to a particular geometry of the imaging system. This lack of flexibility is particularly a problem when using sensors with variable fields of view in which the image structure may vary dynamically.

A more fundamental limitation of image-based processing is its dependence on a particular data format, the image w-ray. This is appropriate for many sensors such as traditional imaging laser range finders, but it is not appropriate for oth-

. This research was partly sponsored by DARPA, under contracts "Perception for Outdoor Navigation" (contract number DACA76-89-C-0014, monitored by the US Army Topographic Engineering Center) and "Unmanned Ground Vehicle System" (contract number DAAE07-90-C-R059, monitored by TACOM). The views and conclusions expressed in this document are those of the authors and should not be interpreted as representing the official policies of ARPA or the US government.

er sensor modalities. For example, very fast single-line sensors have been proposed as a means to perform obstacle detection using either passive techniques [3][4], or laser ranging. In these sensors, the trace of the data is a single line. In this case, the basic unit is a single scanline which cannot be treated in the same manner as a dense 2-D image. The interest in these sensors stems from their low-cost and speed which makes them ideal for automated driving applications (e.g., [2]). Even faster laser range finders recently developed use a non-standard scanning geometry in which range pixels measured on the ground may follow concentric circles around the vehicle or may follow helicoidal patterns [13]. In those cases, the concept of image, or even scanline loses its meaning.

The alternative, pixel-based range processing, is illustrated in Figure 2(b). Here, each pixel is processed individually, independently of the way it was actually acquired, through an image or through another type of scanning pattern. Every time a new range pixel is available, it is converted to a 3-D point in the coordinate system of a grid. Each cell of the grid has a current state which is updated whenever a pixel falls in the cell. Every time a cell is updated, its state is used to evaluate the traversability of the cell. The cell is placed in a separate local map if it is found to be untraversable. In the rest of the paper, we will distinguish between the ***grid*** which is a discrete array of cells that is updated every time a new data point is processed, and the ***local map*** which is another array of cells expressed in a different coordinate system. Grid cells contain a state that is used to determine traversability, while local map cells contain only a flag that indicate whether the cell is an obstacle. The distinction is made because the local map maintains information gathered over many scans but updates it at relatively low frequency, while the grid maintains information gathered over a few scans and is frequently updated.

There are many advantages in using pixel-based processing. First, it does not make any assumptions about the scanning geometry of the sensor. As a result, pixel-based processing can still be used with imaging sensors but can now be used with arbitrary scanning geometry sensors. In particular, this approach provides a simple, common framework for a number of proposed single-line sensor for fast obstacle detection. A second advantage of pixel-based processing, is the ability to have better control over scheduling issues. Specifically, the generation of admissible arcs can be interleaved with the range processing by checking after each pixel is processed whether it is time to send a new steering command to the vehicle. This approach to resolving scheduling between perception and planning enables for guaranteed command update rate.

3 Range Date Acquisition

In order to test and demonstrate our approach to pixel-based processing, we have used the Erim laser range finder as the primary source of range data. This sensor is an imaging range finder that produces a 64 by 256 range image at 2Hz. The characteristics of the sensor and its used are described elsewhere [5]. Although the sensor is an imaging sensor, we are simulating a single scanner by processing each line independently and by tagging each scanline by the position of the vehicle at the time the scanline was acquired. Although all the results presented in this paper use Erim images, the implementation is independent of the image nature of the underlying data.

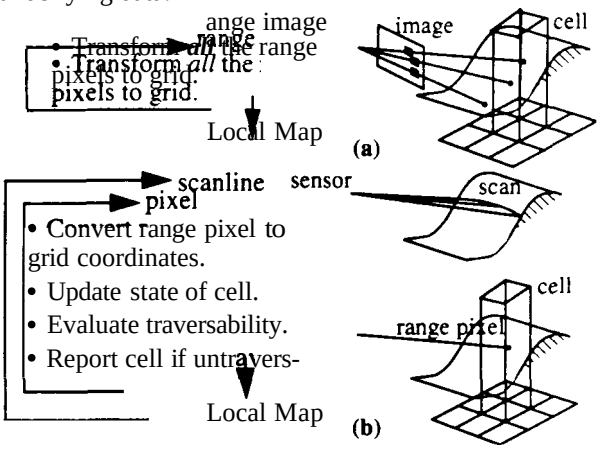

Figure 2: (a) Traditional image/grid processing: (b) Pixel-Based Processing.

4 Updating Cell State

We now describe the state information of a grid cell and the algorithm used for updating it when a new data point is added. The state of a grid cell is described by the following fields:

- Number of data points: N_p. N_p is the number of data points that fall in the cell.
- Average coordinate and uncertainty: p, R. p is the vector of average coordinates of the set of data points in the cell. R is the uncertainty matrix computed from the set of coordinates.
- Min and max elevation: z_{min}, z_{max}. z_{min} and z_{max} characterize the variation of elevation within a cell.
- Average slope and uncertainty: a, S; used for evaluating traversability.
- Number of scanlines: N_s; N_s is used for reducing the number of cell evaluations.
- Traversability status: T_s. T_s is true if the cell has already been evaluated for traversability and has been determined to be an obstacle cell. T_s is used for reducing the number of cell evaluations.

The average elevation and slope can be estimated recursively every time a new data point is added to a cell: If the surface is represented locally by $z = ux+vy+w$, where z is the vertical axis, then the parameter vector $a = [u\ v\ w]^t$ can be updated by the recursive equations: $a_n = a_{n-1} + K(z_n - Mu_{n-1})$, where a_{n-1} is the current estimate of the cell parameters, a_n is the new estimate after adding data point $x_n = [x_n, y_n, z_n]^t$, M is the vector $[x_n\ y_n\ 1]$, and K is a gain computed from the uncertainty matrices of a_{n-1} and of x_n. The uncertainty matrix S_n of a_n is also updated recursively by: $S_n^{-1} = S_{n-1}^{-1} + H^t C_n H$, where C_n is the uncertainty of x_n (a 3x3 matrix), and H is the 3x3 Jacobian matrix: $[u_{n-1}\ v_{n-1}\ -1]^t[x_n\ y_n\ 1]$. The vector of parameters a is converted to a slope vector and an

average elevation value in the cell. This approach has the advantage of using simple computations to update slope without having to recompute all the coordinate moments every time. **Also,** this gives a simple way to estimate the uncertainty on elevation and **slope** at each point. The estimation of p and R is conducted in a similar way. **Although** it may seem that this formulation would require a large amount of computation, we will see in Section 7 that it is very efficient in practice owing to the simple nature of the matrices involved. In fact, this approach coupled with the efficient approach to scrolling of Section 6 makes it more efficient than the traditional full image approach.

Several tests are performed every time a cell is updated to determine whether the cell is traversable or is an obstacle cell. Two tests are currently performed. The first one checks the variation of elevation in the cell according to the min, max, and average values by comparing it to a threshold. The second one uses the direction of slope. These two tests correspond to two traversability conditions introduced in [9], that is, undercarriage clearance and vehicle tilt.

Figure 4 shows an example of updating the state of a grid cell by adding pixels. Figure 3(a) is a plot of the vertical component of slope and its uncertainty as function of the number of pixels added to the cell and Figure 3(b) is a plot of mean, min, and max elevation also as function of the number of pixels. In this example, the cell was located approximately ten meters from the sensor. The pixels appear from the bottom to the top of the surface patch contained is the cell. The slope does not vary monotonically because the variation in elevation is small enough until more than ten points are added to the cell that it appears as if the terrain were flat. The cell is correctly classified as untraversable after 13 points. The example was generated from real data by plotting the state of one of the cells of Figure 5.

A cell is sent to the local map if it is found to be untraversable by computing its coordinates with respect to the local map and by marking the corresponding cell in the local map. We have used a scanline-based outer loop in the actual implementation in that the obstacle cells are sent to the local map only after the entire scanline is processed. However, as noted earlier, the notion of scanline is used only for convenience, because the data from the ranging sensors available to us is naturally organized in scanlines, and because we could tolerate the scanline delay in our navigation scenario.

Figure 5 shows an example of processing on a typical outdoor scene. The top part of the figure shows a video image of an outdoor scene and the corresponding range image. The horizontal lines overlaid on the range image show the four scanlines included in this figure, although all the scanlines are actually processed

Four instances of the grid are displayed in this figure, corresponding to the four scanlines displayed on the range image. Each display is an overhead view of the grid in which the rectangular icon indicating the position of the vehicle and numbered scale on the left indicate distance from the vehicle in meters. The footprints of the scanlines are displayed in each of the grids. The obstacle cells detected as the scanlines are processed are displayed as black dots. The grid displays include all the obstacle cells detected from the scanlines that fall inside the grid for this position of the grid reference frame. In this example, the grid is 60 meters wide and 40 meters deep with a cell resolution of 40cm.

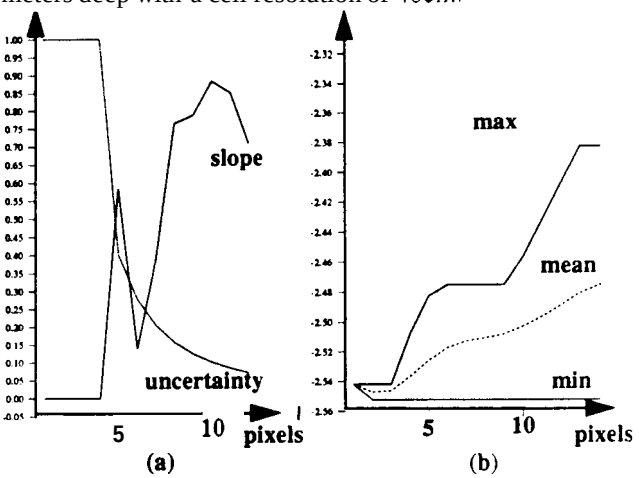

Figure 3: Updating the state of a cell; (a) slope and confidence as functions of the number of pixels; (b) estimated elevation in cell as function of number of pixels.

The scanlines are processed from the closest one to the vehicle to the further away. The order in which obstacle cells are found illustrates why the pixel-based processing allows for lower latency and better system performance: The obstacles closer to the vehicle detected in line 1 are immediately reported to the local map. In general, the algorithm has the desired behavior of reporting the closest objects first. By contrast, a more traditional approach would require a relatively large map to be built before any obstacle cells are reported.

Each scanline is converted to the current grid coordinate system by using the vehicle pose at the time the line was acquired. Since the grid has a finite size, the running scanline will eventually fall outside of the grid. In the example of Figure 5 this happens just after line 4, at which the reference coordinate system of the grid is modified to keep the scanlines in view.

In order to avoid unnecessary cell evaluations and to avoid erroneous classifications, it is important to make sure that a cell contains sufficient information before performing the tests. A cell is evaluated only if $T_s = 0$, that is, it has not been classified through an earlier pass, and if $N_s > 1$. The second condition is to avoid situations in which data from a single scanline is included in a cell which lead to a numerically unstable estimation of the slope. Currently, the traversability tests are applied to a cell only if the number of points in the cell is large enough, five in the current implementation, and if the uncertainty on the parameters is low enough. The effect of using the uncertainty is to require proportionally more data points in a distant cell than in a close for traversability evaluation. It is a desirable behavior because measurement at long range at less accurate than at short range and therefore require more evidence.

5 Local Map and Arc Generation

The local map is an array of cells with a simpler structure than the grid. Local map cells contain only a binary flag indicating whether or not the cell is an obstacle, and, if it is and obstacle, it contains also the coordinates of a **3-D** point inside the obstacle. Storing the coordinates of the point is necessary to avoid aliasing problems when transforming the local map **as** we will show in Section *6*.

The underlying structure of the local map is a two-dimensional array but only the obstacle cells are used. Since the obstacle cells occupy a small fraction of the map (typically less than 10%), it is advantageous to maintain a separate list Lof obstacle cells, each element of $\mathcal{L}$ contains a pointer to a cell location in the **grid**.

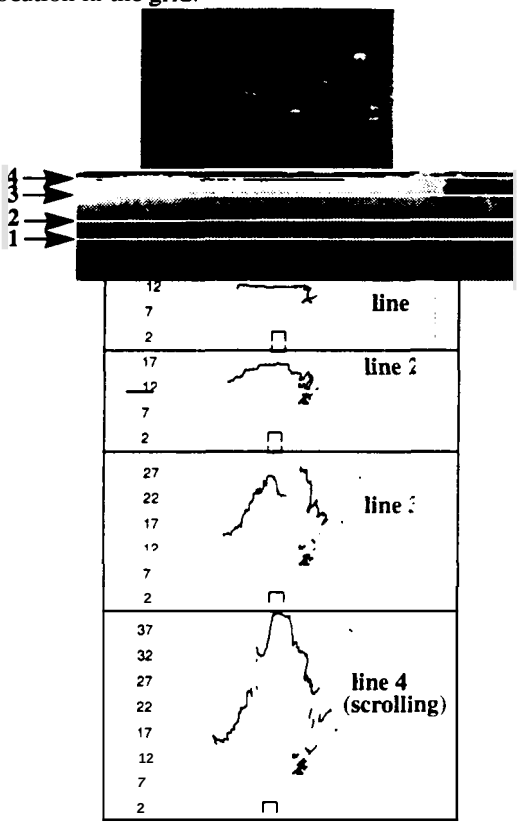

Figure **4:** Scanline-based processing from a **range** image.

The local map is used for generating admissible steering commands. The approach used here was developed by Julio Rosenblatt [6]. Consequently, we give here only a brief description of the approach and refer the reader to *[6]* for a detailed description of the planning architecture. The basic structure used in the arc generation is a discrete array of steering radii. A vote between **-1** and **+1** is associated to each arc, -1 meaning that the arc is forbidden, **+1** meaning that the arc is admissible. Each steering radii corresponds to a segment of arc in the local map with its at the origin of the local map. Each possible arc is evaluated by computing the distance between every cell of Land the arc. An arc receive a vote of -1 if it intersects an obstacle cell and it receives a vote varying monotonically between -1 and **+1** with the distance to the nearest obstacle cell. The fucntion used for mapping cell distances to votes is described in [8]. **A** set of 39 possible arcs is used in the current implementation. The arc radius ranges from **-8** m to **+8** m using an approximately linear variation in curvature. After the vote for each individual arc is computed, the entire array of votes is sent to an arbiter module *[6]* which generates an actual steering command that is sent to the vehicle.

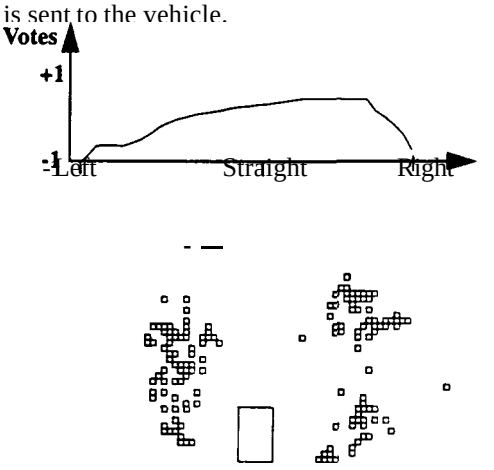

Figure **5: A local** map (bottom) and the corresponding array of votes (top).

Figure *6* shows an example of arc generation from a local map. The bottom part of the figure shows the location of the obstacle cells, displayed **as** squares, in the local map. For reference, the vehicle is shown **as** a rectangular icon at the origin of the local map. In this example, the terrain is another part of the comdor shown in Figure *5* with untraversable region on the both sides of the vehicle and with a traversable corridor curving slightly to the right. The top part of the figure shows the corresponding distribution of votes from a minimum turning radius of **-8** meters to a maximum of +8 meters. The curves show the maximum votes are for moderate right turns of the vehicle and are close to -1 for left and hard right turns.

6 Maintaining Transformations and Scrolling

One of the main difficulty in this type of system is to maintain a consistent representation of pieces of data acquired at different times corresponding to different locations of the vehicle. We use vehicle pose **as** a tagging mechanism. Specifically, every piece of information is tagged by the pose of the vehicle with respect to a fixed reference frame at the time the information was computed. In the remainder of this section we will simply say "the pose of the data", e.g., the pose of a scanline, to mean "the vehicle pose associated with the data". We describe the various poses used in the system in the next section; we describe the scrolling used for updating the **grid** and the local map as vehicle pose changes in Section **6.2.**

6.1 Transformations

In general, vehicle pose is the position in x, y, and z and the orientation in roll, pitch, and yaw of the vehicle with respect to a fixed reference frame. Although poses have in general six degrees of freedom, we limit ourselves in this paper to the three degrees of x,y position and heading because of cur-

Pose	Description	Updated When	Data
V_l	Pose at the time the scanline is acauired.	New scanline is being processed.	Scanline
V_b	Pose at the beginnine of the image.	New image is read.	Image
Ve	Pose at the end of the image.	End of image is reached.	Image
V_g	Pose with respect to which the grid is expressed.	Scanline is outside of the grid.	**Grid**
V_m	Pose with respect to which the local map isexpressed.	Arcs are sent to the external planner.	Local Map

Figure 6: Transformations and their inter-relations.

The first pose V_l is computed every time a new scanline is read from the sensor and is attached to the scanline. V_l is read directly from the controller in the case of a one-line sensor; it is interpolated between beginning and end of image in the case of an imaging sensor as described below. As noted earlier, the concept of scanline is used for convenience only and can be replaced by V_r, the vehicle pose for a single range pixel, for an arbitrary scanning geometry. Such pose data is available from scanners such as [13].

The next two poses, V_b and V_e, are meaningful only when an imaging sensor is used. They are used for correcting the smearing of the image due to vehicle motion by interpolating pose of any scanline in the image between V_b and V_e. Currently, the interpolation is a simple linear interpolation which performs well for moderate speeds and rates of turn.

The next pose, V_g indicates the origin of the grid in which the coordinate conversion and obstacle classification are taking place. Whenever a new scanline of pose V_l is processed, its data points are converted from local sensor coordinate system to grid coordinate system by $V_g.V_l^{-1}$. Unlike the previous poses, V_g is not read from the controller but is computed from the other poses in order to ensure that the grid covers an area large enough in front of the vehicle. This is the object of the ***grid scrolling.*** The first time data is processed, V_g is set to the pose of the first scanline V_l^0, the origin of the grid is placed at V_g and all the computations are done with respect to this origin. As the vehicle travels, it reaches a point a which a new scanline, of pose V_l^1 is partially outside of the bounds of the grid. When this condition occurs, V_g is set to V_l^1 and all the cells in the grid are transformed to this new reference frame. This procedure is repeated whenever a new scanline falls outside of the grid.

The last pose, V_m, is the pose with respect to which the elements of the local map are expressed. A new obstacle cell is added to the map by transforming its coordinates in the grid to the local map coordinates using the transformation $V_m.V_g^{-1}$. A new V_m is read from the controller every time the system determines that it is time to send new arcs to the planning system. When this condition occurs, the system performs a local map scrolling operation in which all the elements are converted from their coordinates in the previous V_m to the new V_m.

6.2 Scrolling Algorithms

Scrolling grid or local map involves applying the appropriate transformations to the cells and re-arranging the cells according to the new coordinates. The implementation of both grid and local map require a efficient scrolling algorithm, for the scrolling is potentially expensive and may introduce undesirable delays. A straightforward implementation would be to initialize a new array, to transform all the cells of the old array into the new one, and to copy the new array into the old location. This simple algorithm is potentially expensive because it requires initializing and traversing a potentially large array. This algorithm is inefficient because the array is typically very sparse, especially in the case of the local map in which less than 10% of the cells are used at any given time. From the point of view of navigation, this algorithm has the undesirable side effect that it takes the same amount of time irrespective of the number of active cells in the array.

A better approach is to maintain a marker M for each cell. The marker is used to indicate whether the cell has been visited since the last time the array was scrolled. In addition, a list L of active cells is maintained. In the case of the grid, the active cells are those cells to which more than one data point contributes from the current grid pose. In the case of the local map, the active cells are the current obstacle cells. With these definitions, the scrolling proceeds as follows: First, the current marker value, M_0, is incremented, then the coordinates of each cell pointed to by the elements of L are computed, the corresponding pointer is updated and the marker of the cell is set to the new value M_0. Two cases may occur after the transformed cell C' of a cell C of L is computed. If the marker of $M(C')$ is different than the current value M_0, then C' should be considered as a new cell and should be initialized, otherwise, its current value is updated by adding the data from C'. The way the update is performed depends on the array being scrolled. In the case of the local map, no computation is needed because C' is already listed as an obstacle and does not need to be updated further. In the case of the grid, the data in C', such as mean elevation, slope, etc. is updated by merging the data from C. The updates of slope and mean elevation are performed using the algorithm of Section 2. After all the cells in L have been updated, L is traversed one more time to delete entries C such that $M(C) \neq M_0$. Such entries are cell locations that are not used in the scrolled array.

We described the use of markers in the context of scrolling but the same mechanism is used for adding new cells to the grid or to the local map. Let C be the cell in which the new

data is to be added, if $M(C)$ is different from M_n then C is initialized, the new data is copied in C, and a pointer to C is added to $\mathcal{L}$; otherwise, C is updated with the new data. This algorithm allows for efficient scrolling of the grid and local map arrays by initializing and updating only the cells that are needed. In particular, it has the desired behavior of being faster when no active cells are present in the array.

7 Performance

In the current implementation, the driving system is implemented as shown in Figure 3: The main loop acquires and processes one scanline at a time, updating grid cells as described is Section 4 and scrolling the grid when necessary as described in Section 6. In addition, the scrolling local map and arc generation of sections 5 and 6 are activated at regular intervals. The current interval is 100ms although the actual frequency of map scrolling and arc generation must be computed based on the desired speed of the vehicle. The time performance of the system is shown in Figure 8.

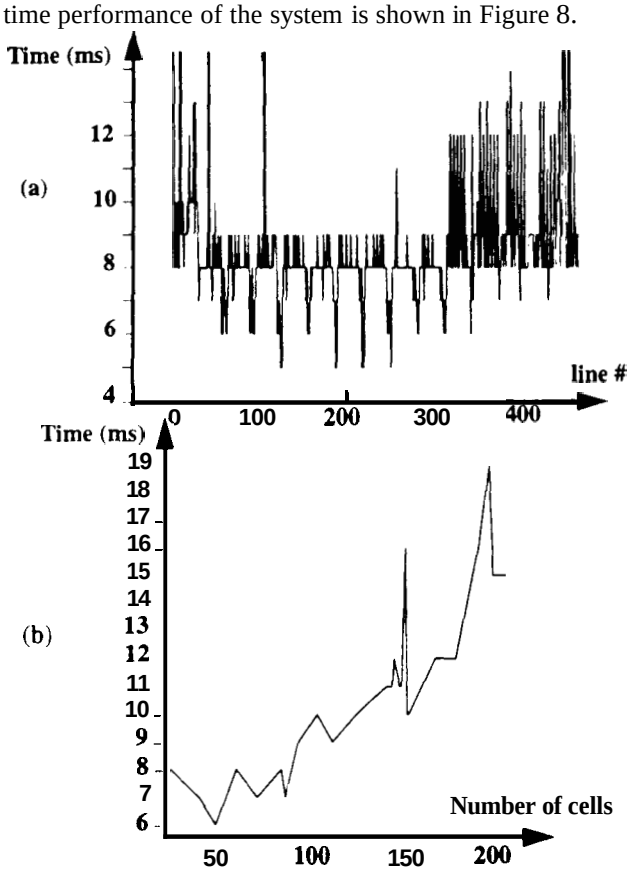

Figure 7: (a) Grid processing time per scanline; (b) local map processing time as function of the number of obstacle cells.

Figure 8(a) is a plot of computation time per scanline. The vertical axis is the time in milliseconds spent processing each scanline and the horizontal axis is the scanline number. There are on average 256 points per scanline. The graph shows that the time per scanline is 8ms on average with the periodic drops being due to the fact that not all pixels of a scanline are processed at far range. Figure 8(b) is a plot of the computation time used for scrolling the map and generating the arcs as a function of the number of obstacle cells. Although the absolute time values are clearly dependent on the platform used, in this case a Sparc II station, the graphs show that our proposed architecture for scanline- and pixel-based processing allows high performance and low latency using conventional hardware. In particular, an obstacle is taken into account in the arc calculation at most 100ms after it is detected with very little additional delay (19ms is the worst case in this example). This data was collected on the course used in [8].

8 Conclusion

We have developed an efficient approach to range data processing for autonomous driving. The approach uses the concept of pixel-based processing in order to eliminate latency in the system and to allow for efficient merging of range data processing, obstacle detection, and arc generation. We have demonstrated the system using the Erim imaging laser range finder to simulate a single scanline sensor.

This work improves on our earlier system for autonomous navigation [8] by eliminating the delays in image processing and by eliminating variable communication delays through tighter integration with predictable cycle time. Our current work concentrates on demonstrating the system on long-distance missions such as in [8]. Also, we will demonstrate the flexibility of the system by using it with other sensors, such a fast single scanline sensor and passive stereo. Finally, we are planning on porting the system to a real-time operating system in order to eliminate the remaining spikes in computation time profile due to resource contention between processes.

References

[1] O. Amidi. *Integrated Mobile Robot Control.* CMU-RI-TR-90-17, The Robotics Institute, Carnegie Mellon, 1990.

[2] J.L. Bruyelle, J.G. Postaire, "Direct Range Measurement by Linear Stereovision for Real-Time Obstacle Detection in Road Traffic", Proc. IAS-3, Pittsburgh. 1993.

[3] S.X. Godber, M. Robinson, "Three-Dimensional Machine Vision Using Line-Scan Sensors". SPIE Vol. 1823. 1992.

[4] S.X. Godber, M. Robinson, P. Evans, "Stereoscopic Line-Scan Systems for Robotics Control". Proc. IAS-3, Pittsburgh. 1993.

[5] M. Hebert, E. Krotkov. 3D Measurements from Imaging Laser Radars. *Image and Vision Computing 10(3)*, April 1992.

[6] Keirsey, D.M., Payton, D.W. and Rosenblatt. J.K.. "Autonomous Navigation in Cross-country Terrain," in Image Understanding Workshop, Cambridge, MA, April. 1988.

[7] A. Kelly, T. Stentz, M. Hebert. Terrain Map Building for Fast Navigat ion on Rugged Outdoor Terrain. In *Proc. of the SPIE Conference on Mobile Robots,* 1992.

[8] D. Langer, J.K. Rosenblatt, M. Hebert, "An Integrated System for Off-Road Navigation", Proc. IEEE International Conference on Robotics and Automation, San Diego 1993.

[9] K. Olin, D.Y. Tseng. "Autonomous Cross-Country Navigation". *IEEE Expert,* 6(4), August 1991.

[10] L. H. Matthies. Stereo Vision for Planetary Rovers: Stochastic Modeling to Near Real-Time implementation. IJCV, 8:1, 1992.

[11] L. H. Matthies, P. Grandjean. *Stochastic Performance Modeling and Evaluation of Obstacle Detectability With Imaging Range Sensors.* JPL-93-11, JPL, 1993.

[12] C. Thorpe, O. Amidi, J. Gowdy, M. Hebert. D. Pomerleau.. Integrating Position Measurement and Image understanding for Autonomous Vehicle Navigation. Proc. *Workshop on High Precision Navigation,* Springer-Verlag Publisher, 1991.

[13] H. Verdun, G. Stevenson "1.54μm Solid State Laser Radar for Obstacle Avoidance", Tech. Report, Fibertek Inc., 1992.