Autonomous Robots, 2, 147–161 (1995)

Range Sensor Based Outdoor Vehicle Navigation, Collision Avoidance and Parallel Parking

DIRK LANGER AND CHARLES THORPE
The *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213*

Abstract, Detecting unexpected obstacles and avoiding collisions is an important task for any autonomous mobile system. This article describes GANESHA (Grid based Approach for Navigation by Evidence Storage and Histogram Analysis), a system using sonar that we implemented for the autonomous land vehicle Navlab. The general hardware configuration of the system is shown, followed by a description of how the system builds a local grid map of its environment. The information collected in the map can then be used for a variety of applications in vehicle navigation like collision avoidance, feature tracking and parking. An algorithm was implemented that can track a static feature such as a rail, wall or an array of parked cars and use this information to drive the vehicle. Methods for filtering the raw data and generating the steering commands are discussed and the implementation for collision avoidance, parallel parking and its integration with other vehicle systems is described.

Keywords: obstacle detection, collision avoidance, parallel parking, navigation, grid map

1 Introduction

The autonomous land vehicle Navlab has already successfully been driven on roads and cross country. Different sensors are used to perceive the structure of the environment and navigate the vehicle under a variety of conditions as described for example in *(Thorpe* et al., 1991). The sensors mainly employed so far were colour video cameras and the ERIM 3-D laser range finder. Detecting obstacles and taking an appropriate decision is an important task for any mobile system in order to navigate safely through its environment. Obstacle detection is possible using colour video images or ERIM range images. However, owing to the data aquisition process and the required intensive image processing, these types of perception systems are generally not very well suited for a quick reaction to unexpected obstacles. Especially in the case of collision avoidance, a sensor is needed that can supply the relevant information fast with little data processing overhead and interact with actuators at the level of the vehicle controller (see also (Amidi, 1990)). Sensors that satisfy these requirements are for example sonars, infrared sensors, pulsed 1-D laser range finders or microwave radar.

Compared to light based sensors, sonars have the advantage that they do not get confused by transparent or black surfaces. On the other hand, the wavelength of ultrasound is much larger than the wavelength of light, i.e. usually around 4 mm as compared to 550 nm for visible light. Therefore, unless the transducer faces the reflector surface in a normal direction, only rough surfaces or edges can reflect sound waves. However, most real world outdoor surfaces almost always have a type of surface roughness that enables a sonar to detect the object. We therefore decided to use sonar sensors for the collision avoidance system of the autonomous land vehicle Navlab. The following sections describe the sonar system and its performance in an outdoor environment. Some novel results were obtained in using the system for vehicle navigation by itself and by integrating it into other vehicle navigation systems. The system is configured in such a way that more sensors can be added easily in the future. These sensors do not necessarily have to be sonars but can be any other type of point range sensor. In the future we intend to integrate at least one other type of point range sensor into the system, most probably laser or radar based. For a proof of concept, the ERIM laser range finder has already been successfully integrated as part of a cross-country navigation system. In this case, however, obstacles can only be detected by first processing 3-D range images (refer to (Langer, Rosenblatt, & Hebert, 1994)).

2 Hardware Configuration

Sonar sensors have already been used successfully for indoor mobile robots as described in (Borenstein & Koren, 1990; Elfes, 1986; Moravec, 1988) and (Leonard & Durrant-Whyte, 1990). An outdoor environment, however, adds some additional constraints

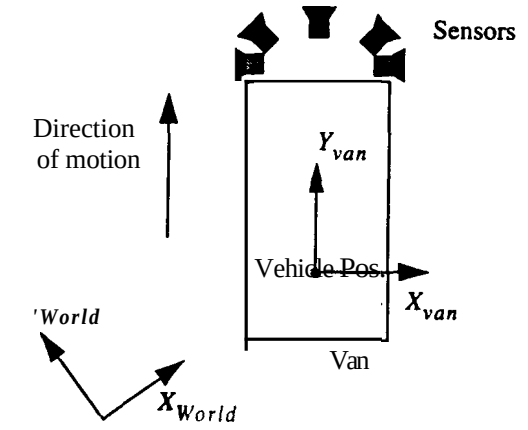


Fig. 1. **Sensor configuration.**

on the type of sensor that can be used. Specifically the sensor should meet the following requirements:

- Robust against moisture and dust particles.
- Robust against noise from vehicle engine and other sound sources.

Therefore an open type electrostatic transducer such as the Polaroid cannot be used. Instead a closed type piezoceramic transducer operating at a frequency of 80 kHz was selected. A detailed description of this sensor is given in (Magori &Walker, 1987). The high operating frequency makes this sensor fairly robust against acoustic noise, while still providing an operating range up to 6 m. The beam angle of the sensor is approximately 5°, i.e. at the 3 dB intensity fall off from the major axis. Based on these characteristics, a total of five sensors was chosen in order to provide a good area coverage in front of the vehicle with a reasonable spatial resolution. The sensors are mounted on a guide rail such that their position and orientation can be freely adjusted. A typical sensor arrangement is shown in Fig. 1. Certain sensor configurations or environments can lead to acoustic interference between individual sensors. Therefore the hardware provides the ability to choose the exact trigger time of each sensor. In most circumstances the sensors are mounted such that they point away from each other. In this case all sensors are triggered at the same time. At present a measurement rate of 9 Hz is used, which is based on the following calculations: For very good reflectors we can assume a maximum operating range of 8 m, which corresponds to a time of flight of sound in air of approximately 50 ms. Thus echoes are considered during a receiving period of $T_{rec} = 50$ ms after triggering the sensor. In order to avoid measurement errors due to multiple echoes, only the range of the first echo is measured. The sensors are retriggered after an additional wait period of $T_{wait} = 60$ ms, which ensures that all detectable echoes from previous pulses are attenuated below the detection threshold. Thus the total cycle time $T = T_{rec} + T_{wait} = 110$ ms.

Each sensor measurement is tagged with the position of the vehicle. At present the Navlab uses dead reckoning to estimate its position relative to some initial point. The distance travelled is provided by an optical encoder on the drive shaft and vehicle orientation in 3-D space is provided by the gyroscope of an inertial navigation system. The measurements are combined to give the x-y position and orientation of the vehicle with respect to world coordinates. For the position tag of a sonar measurement only the following three parameters are used: x, y and φ, where φ = yaw (Fig. 1).

The hardware of the sonar system consists of an interface module which triggers the sensors and measures the time of flight to the first echo returned. The interface module is accessed via VME bus from a 68020 based CPU. This processor runs as a slave of the vehicle controller processor under a real time operating system and takes care of data aquisition, conversion to range, position tagging and proper timings. The map building and tracking algorithms are presently implemented on a Sun SPARC station which communicates with the vehicle controller via ethernet. Ethernet communication time uncertainties can be neglected because of the comparatively long cycle time of the sonar system. The hardware configuration is shown in Fig. 2.

3 Local Grid Map

This sensor system can now be used to build a local grid map. The grid map is local because it contains only information about the immediate surroundings of the vehicle. The vehicle position is kept at a fixed point in the map. As the vehicle moves, objects in the map are moved from cell to cell relative to vehicle position. Once an object falls outside the map boundary it is discarded and the information is lost. Using just a local map has the advantage that error accumulation owing to dead reckoning is kept small, since only relative movements are considered. On the other hand the disadvantage is that information is lost and thus no global information is available. However, if desired, sequences of the output from the local map could be combined and included in a larger global map. At present

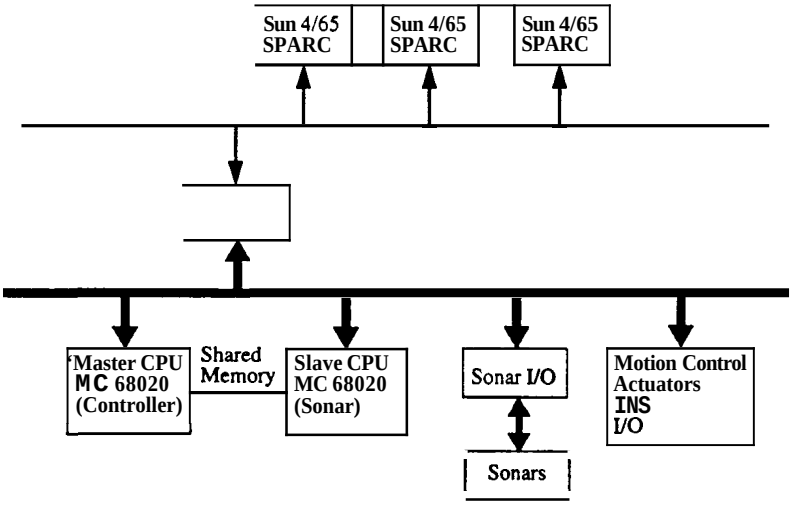


Fig. 2. **Hardware architecture.**

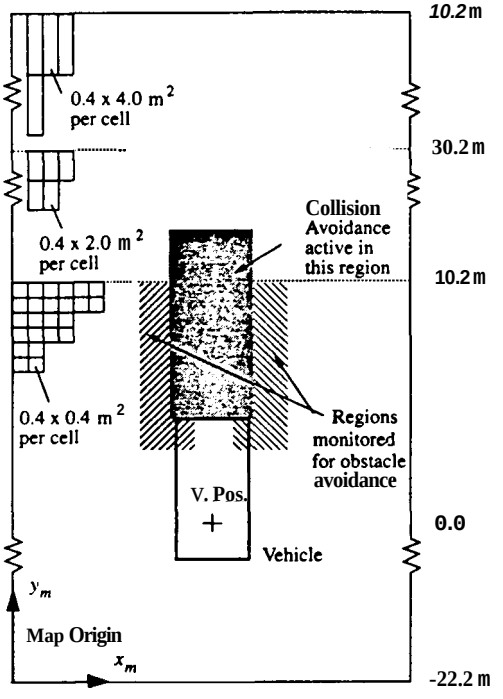


Fig. 3. **Local grid map**

the area covered by the local map is from **−8.2** m to **8.2** m along the x-axis and from **−22.2** m to **70.2** m along the y-axis. Each grid cell has a resolution of 0.4 m along the x-axis, and a variable resolution of 0.4 m, **2** m or 4 m along the y-axis, depending upon distance from the vehicle. Hence the map consists of 41 **x** 101 cells (see Fig. 3). The reason for taking such a coarse resolution for each cell is that most applications of the system do not require a high accuracy and the size of the highest resolution grid cell is small with respect to the size of the vehicle. As objects get further away from the vehicle, a coarser map resolution is adequate. In general, sensor readings become less accurate as the object is further away. However, coarse resolution map information is sufficient to plan initial navigation maneuvres. As the object moves closer, navigation needs to be more precise and therefore map resolution is increased.

3.1 Grid Representation

This grid map representation can be used with any obstacle detection sensor such as sonar, laser (single spot or scanned), radar or stereo. Currently only an array of sonar sensors is used as described in section **2.** This sensor does not always provide an accurate measurement. For the particular sonar sensors chosen for example, the measurement accuracy is about **±1** cm. However, depending on the environment, the sensor may also deliver noisy results. The reason lies in the poor angular resolution of the sensor. Echoes may be detected **from** an object at a far range in the main lobe or from **a** good reflector at closer range in a side lobe. Depending on the relative signal strengths and movement of the sensor, the range reading may oscillate. A similar effect can also happen when an echo reflected multiple times is received. The sonars have a fairly small maximum range. However, in anticipation of future additions of longer range sensors based on

radar or laser, the maximum forward coverage of the map is large at multiple resolutions.

Each cell has a set of parameters or annotations associated with it, which are described below:

1. ***Object Type.*** This parameter is used to indicate if the object in that cell was seen at the current sensor reading, or if it was seen at a previous reading. If it was seen only at a previous reading, then the object type indicates that it must have moved to that particular cell due to vehicle motion only.
   The parameter is also used to denote which type of sensor detected that particular object if several different types of sensors are connected (e.g. sonar or radar).
2. ***Position.*** Indicates the x-y position of the object with respect to vehicle position.
3. ***History.*** This parameter counts the number of times an object was detected in a particular cell.
4. ***Curvature Vector:*** Is precomputed for each cell and denotes the range of steering arcs that would avoid a collision between the vehicle and an object in that cell.

The resolution of the grid is fairly coarse and hence a position parameter (X_{obj}, Y_{obj}) is kept to avoid gross error accumulation when objects are transformed in the map. Only one object is kept per grid cell.

Measurement uncertainty is part of the grid cell representation and any object detected within an area covered by a particular cell is taken to belong to the same object.

3.2 *Grid Transformation*

Following, a short description of object transformation within the map is given: Vehicle position and orientation are kept constant within the map. Therefore objects in the map move with respect to the vehicle. The vehicle's positioning system returns vehicle position and orientation with respect to a global frame of reference (x, y) that is determined at the time of initialization of the system. Since we are interested only in the movement of objects with respect to the vehicle's coordinate system (x_m, y_m), the appropriate transformation is obtained by using the position increments along the axis $(\delta x, \delta y)$ and orientation $\delta\varphi$. The total distance traveled is given by $\sum \delta s$ (Eq. (7)). As the map remains local, errors due to dead reckoning are reduced and the positioning system can be initialized independently by using this method. Hence if the vehicle moves from a position in a plane (x_1, y_1, φ_1)

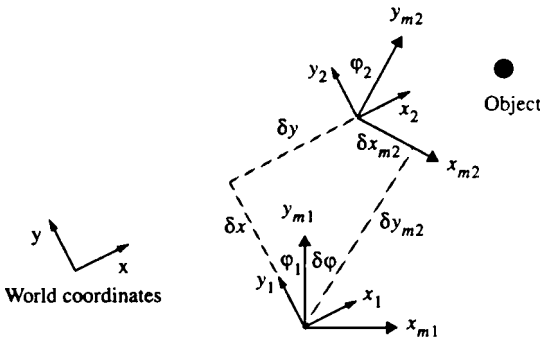


Fig. 4. **Object transformation in local grid map**

to a new position (x_2, y_2, φ_2), then an object in the map at position (x_{m1}, y_{m1}) is transformed to position (x_{m2}, y_{m2}) as follows (refer to Fig. 4):

Position and orientation increment:

$$\begin{aligned} \delta x &= x_2 - x_1, \qquad \delta y = y_2 - y_1, \\ \delta\varphi &= \varphi_2 - \varphi_1, \qquad \delta s = \sqrt{\delta x^2 + \delta y^2} \\ \delta x_{m2} &= \delta x \cos\varphi_2 + \delta y \sin\varphi_2 \\ \delta y_{m2} &= -\delta x \sin\varphi_2 + \delta y \cos\varphi_2 \end{aligned} \tag{1}$$

New object position (vehicle moving forward):

$$\begin{aligned} x_{m2} &= x_{m1} \cos\delta\varphi - y_{m1} \sin\delta\varphi - \delta x_{m2} \\ y_{m2} &= x_{m1} \sin\delta\varphi + y_{m1} \cos\delta\varphi - \delta y_{m2} \end{aligned} \tag{2}$$

After the position of objects in the map is updated, new objects detected by the sensors are added. A sensor measures the range R to an object. Position and orientation of each sensor on the vehicle are known. Hence, using the transformation 'sensor position → vehicle position → map', a new object is placed in a cell in the map (refer to Fig. 5). If that particular cell is already occupied, then only the cell parameter 'History' is updated as described in 3.3, otherwise all cell parameters are updated.

3.3 *Filtering*

The map parameter **History** is used for differentiating between moving and stationary objects and for filtering the data. **History** can also be used to evaluate the confidence that a particular cell is occupied by an object. A higher value of **History** indicates a higher confidence. In the case of collision avoidance for example it is desirable to slow down the vehicle if there is an obstacle in front and to resume driving if the obstacle moves away, like a car or person. Hence objects

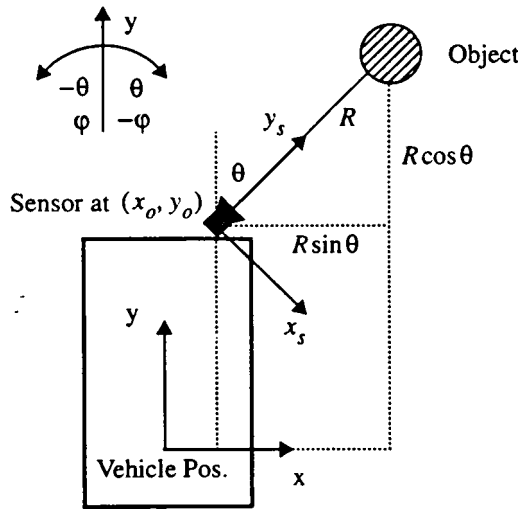


Fig. 5. Sensor to object transformation.

in the map in a sensor's field of view are deleted if a sensor does not detect them anymore. However, an object will not be deleted immediately, but only after it was not seen by the sensor for a certain time period. This time period is defined by a parameter called **Life** Time, which is given in terms of the number of cycles of the update sequence (Fig. **7).** We can assume that the cycle period is approximately constant. The parameter **History** is therefore updated at time t as follows:

- If an object is detected, then
  1. If cell is empty, History, = Life Time
  2. Otherwise, History, $= \text{History}_{t-1} + 1$
  3. Decay Amplitude = 0
- If no object is detected by the sensor and History is not equal to zero, then
  1. Calculate Decay Amplitude initially, i.e. if it is zero:

     Decay Amplitude $= \text{History}_t/\text{Life Time}$

     Decay Amplitude is calculated only once at the beginning of a decay sequence. It ensures that each object disappears after the same amount of time, i.e. Life Time, has passed.
  2. History, $= \text{History}_{t-1}$ − Decay Amplitude

Since different types of sensors may have different field of views, a field of view constraint is applied depending on which type of sensor detected a particular object. Each map grid cell is labeled as to whether it

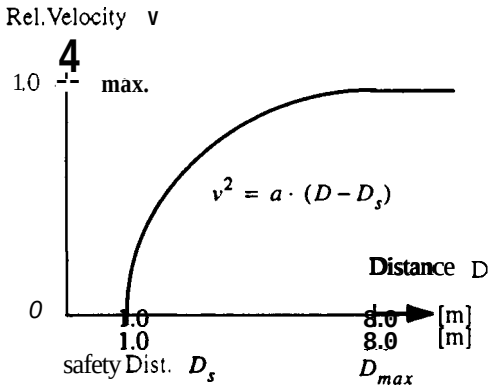


Fig. 6. Velocity control.

is within the field of view of a particular sensor. In the case of sonar sensors, it is not very useful to consider the field of view of side looking sensors. Because of vehicle speed and driving direction, objects detected in these areas are seen only for a short time and it would not be possible to apply a proper decay. Therefore here the decay algorithm is applied only to objects appearing in the area directly in front of the vehicle. Moving objects appearing in this area are also of most concern for safe vehicle navigation. Therefore the velocity of the vehicle is reduced as a function of range to the closest object detected in this area. The velocity is set to zero if the closest range is less than a certain minimum safety distance. The decay algorithm ensures that the vehicle resumes driving when the obstacle moves away. Figure 6 shows a plot of the percentage of vehicle velocity set versus closest range. $D_{\max}$ corresponds to the maximum sensor operating range. The gain a is calculated as

$$a = v_{\max}^2/(D_{\max} - D_s) \tag{3}$$

The parameter **History** can also be used to eliminate spurious echoes such as the ones returned by rain droplets. In this case an object is supposed to be actually present only if it has been seen consecutively for a certain number of cycle times. In practice, a threshold of Life Time + 2 has worked well. Since rain droplets return echoes at random ranges as time progresses, these returns can then be filtered out and will not appear as ghost objects in the map. Of course this procedure does not work in a heavy downpour since the number of rain drops in the environment just becomes too large.

The parameter **Object Type** is used to denote the sensor type that detected the object, i.e. for example sonar or laser if these sensor types are connected. In addition, it also indicates the type of operation that

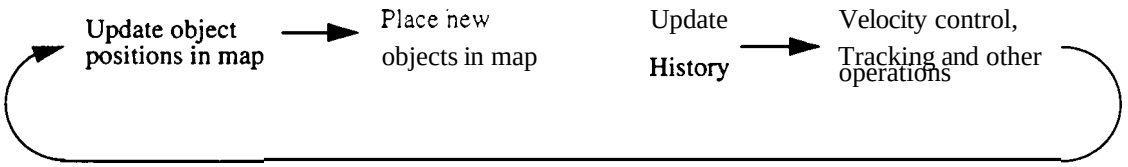


Fig. 7. Update sequence for local grid map

was performed on an object in a particular grid cell. These operations can be transformations as mentioned before, or filtering operations like the ones described in the following sections. **Object Type** is also used to tag certain objects for tracking as described in section 6.

Figure 7 shows the cycle for map operations.

At present, each grid cell is associated with only four parameters. The map structure allows an extension to other annotations in the future, such as object characteristics or triggers. In this way the local grid map could easily be integrated into an Annotated *Map* such as the one described in (Thorpe & Gowdy, 1990).

Using a histogram based method provides fast map building for real time outdoor mobile robot navigation. This has also been shown in a similar approach for sonars in (Borenstein & Koren, 1990). Probability based methods such as described in (Moravec, 1988) and (Elfes, 1986) provide more accurate and detailed map data. This is achieved by using many data samples and a high resolution grid. However, it is computationally more expensive and therefore not as well suited for fast navigation. If only few data samples are available as in our case, the performance of a probability based method will degrade correspondingly and a histogram method will be more efficient.

In general, for the environment considered, a low resolution map and the ability to detect objects are sufficient.

4 Obstacle Avoidance

The data collected by the grid map can now be used for autonomous vehicle navigation functions. A basic navigation function is the avoidance of obstacles in the environment of the vehicle.

The problem here consists of finding a range of arcs that the vehicle can safely travel on without colliding with an obstacle. In order to represent arcs, curvature is used instead of radius since curvature space is continous whereas radius space is not. For each cell in the grid map as described in section 3, a range of arcs can be determined that would make the vehicle collide with an object in that cell. Computing this range of arcs for each grid cell and merging the results from the entire map will determine the final range of arcs that the vehicle can safely drive on and those that would lead to a collision. In order to simplify the merging of results from each individual grid cell, the total range of steering arcs is linearly discretized into a set of 31 arcs, i.e. from maximum curvature left to maximum curvature right. The number of arcs was chosen to be 31 as it can be easily represented as a 32 bit integer number for efficient computations.

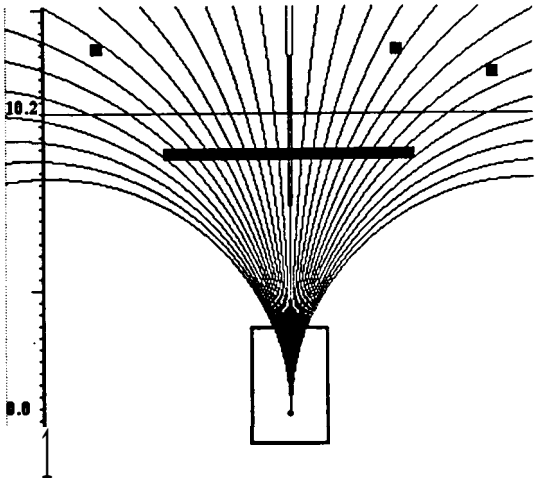


Fig. 8. Arc evaluation for collision avoidance.

Figure 8 shows the set of discretized arcs that are evaluated for collision avoidance.

The range of inhibited arcs for each grid cell can be precalculated as follows: The centre of the obstacle is taken to be at the mid point of the cell. Since it is undesirable to pass an obstacle too closely, the obstacle is assumed to fill the entire grid cell and in addition is expanded by a safety margin M_s. A particular expanded object blocks not just any arc that it touches, but often a range of nearby arcs as well, due to vehicle size. We therefore calculate the boundaries of the range of arcs that would make the vehicle collide with the object.

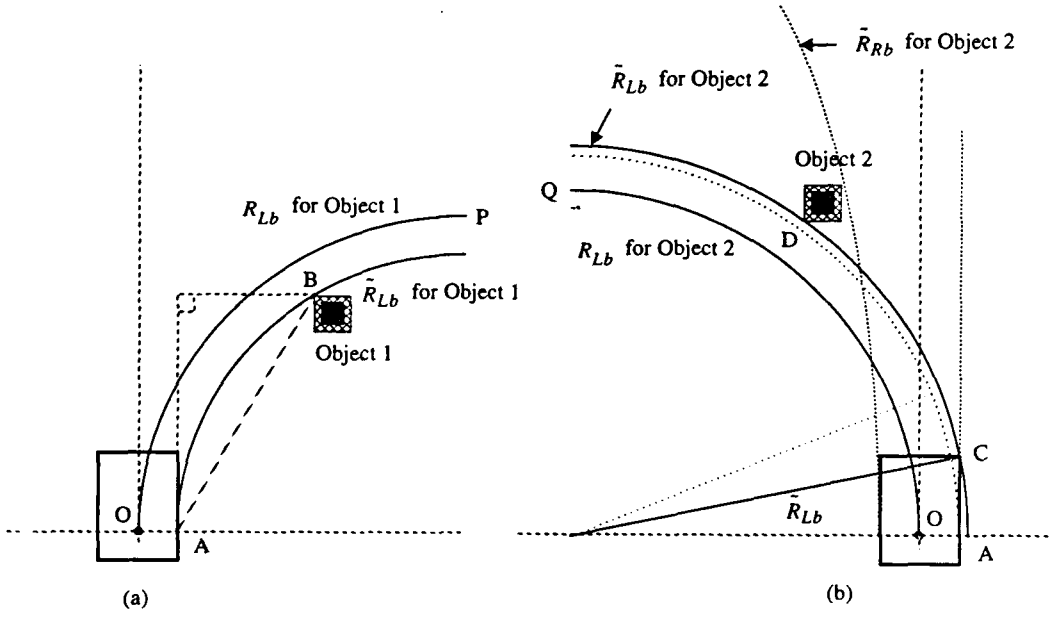


Fig. 9. **Precalculating arcs**

If the expanded object is inside the arc, the rear inside wheel tracks the innermost path and this defines the widest part that needs to be checked for collision (Fig. 9(a)). If the object is outside the arc, the outside front corner swings widest and sets the collision limit (Fig. 9(b)).

Hence, in the first case the left bound radius of the inhibited steering arcs R_{Lb} is given by the arc OP with respect to Object 1 as shown in Fig. 9(a). We therefore use the geometry shown for Object 1 and obtain (see also Appendix **A),**

$$\tilde{R}_{Lb} = (x_L^2 + y_L^2)/(2x_L) \qquad x_L = x_{\text{obj}} - M_s$$
$$y_L = y_{\text{obj}} + M_s$$
$$\therefore \quad R_{Lb} = \tilde{R}_{Lb} + 0.5V_w \qquad \text{Curv}_{Lb} = 1/R_{Lb} \quad (4)$$

where $(x_{\text{obj}}, y_{\text{obj}})$ are the coordinates of the centre of the object and V_w is the width of the vehicle.

In the second case the outside front comer point will hit an obstacle first in a collision and the geometry shown for Object *2* in Fig. 9(b) is used. $\tilde{R}_{Lb}$ is then given by the circle that passes through points C and D with its centre $(-x_o, 0)$ on the x-axis. If the distance from vehicle position to the front of the vehicle **is** V_F, then the coordinates of point C are $(0.5V_w, V_F)$. The coordinates of point D are $x_D = x_{\text{obj}} - M_s$ and $y_D = y_{\text{obj}} - M_s$. Using the equation of the circle, we then obtain,

$$\tilde{R}_{Lb} = \sqrt{(x_D + x_o)^2 + y_D^2} = \sqrt{(x_C + x_o)^2 + y_C^2}$$
$$\Rightarrow x_o = \frac{x_D^2 - (0.5V_w)^2 + y_D^2 - V_F^2}{2(0.5V_w - x_D)} \quad (5)$$

The left bound of the inhibited steering arcs R_{Lb} is then given by,

$$R_{Lb} = -\left(\sqrt{\tilde{R}_{Lb}^2 - V_F^2} - 0.5V_w\right) \quad (6)$$

Note that in the above equation R_{Lb} is negative for the given geometry as **it** indicates a left turn. Radius is then again converted to curvature.

In a similar way the right bound radius (curvature) of the inhibited steering arcs $R_{Rb}(Curv_{Rb})$ can be calculated (see Fig. **9** (b)).

For each grid cell we now have the bounds on the inhibited steering arcs $Curv_{Lb}$ and $Curv_{Rb}$. The set of 31 discrete steering arcs can be represented using a **32** bit number or bitvector. Each bit represents a steering arc and a value of **1** indicates permission to drive on the respective arc; a bit value of 0 indicates that the respective arc is inhibited. Bit 0 (LSB) is always ignored here since only **31** bits are needed. Given the bounds on the inhibited steering arcs, the bitvector representing permitted and inhibited arcs for each grid cell can

be calculated and stored. An example of a bitvector is given in **Eq.** (7). Here the zero curvature direction (straight ahead) and the three extreme left curvatures are inhibited. All other directions are permitted.

$$\vec{p} = 00011111111111101111111111111111 \quad (7)$$

During actual program runtime, only the bitvectors for each obstacle cell need to be ANDed and the resultant bitvector indicates permitted and inhibited arcs for all obstacles in the vehicles environment, i.e.

Initially:

$$\vec{P}_{\text{Result}} = 11111111111111111111111111111111$$

For each cell i:

$$\vec{P}_{\text{Result}} = \vec{P}_{\text{Result}} \quad \text{AND} \quad \vec{P}_i$$

If the vehicle moves through an obstacle field along a desired direction such as one given by a feature tracker (see section **5),** an unobstructed direction closest to the desired direction is selected from the resultant bitvector. In certain circumstances this procedure can lead to oscillations in the vehicle steering, especially if the spatial distribution of obstacles changes rapidly or when approaching a wall (Fig. **8).** To prevent this from happening, a mechanism is used that keeps the vehicle turning in one direction, either left or right, once a direction has been selected in order to avoid an obstacle. The mechanism resets when no more obstacles are present or **if** the difference between desired steering direction and the new selected steering direction is less than 10% of the number of discrete arcs, i.e. 3 arcs here. The condition for the latter case indicates that there is little danger of oscillations as desired and selected steering direction are very close.

In addition the vehicle is prevented from switching from extreme left turns to extreme right turns and vice versa unless **an** intermediate turn is selected in between.

These mechanisms were implemented as a result of observations of the behaviour of the algorithm in simulation and in practice on the real vehicle.

5 Object Tracking

Another basic navigation function is the tracking of features in the environment and using this information to determine a path that the vehicle can drive. The following paragraphs describe a method by which the vehicle uses its sonar sensors to drive on a path parallel to a feature such as a wall, railroad track or parked cars.

5.1 Feature Selection and Path Determination

Figure 10 shows data collected in the local grid map when tracking cars parked on the right hand side of the road. As can be seen in the figure, the side of the cars facing the road is fairly well detected. Usually sonar does not detect smooth surfaces very well because of specular reflections. However, in most real

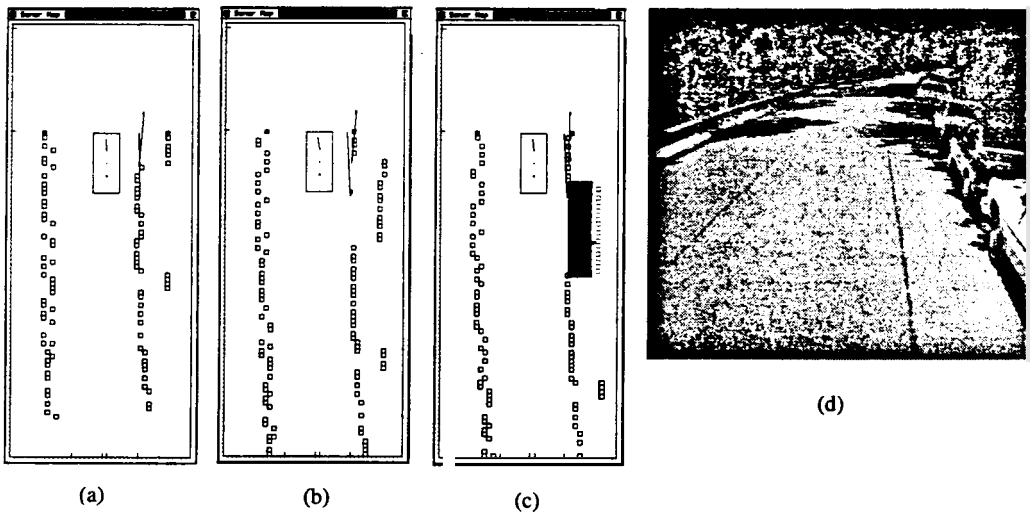


Fig. 10. **Searching** for **a parking space: (a) Approaching gap, (b) Detecting gap, (c) Preparing vehicle** for **parking** maneuvre, **(d) Typical** street **scene.**

world environments such as this one, there are no perfectly smooth surfaces. In this case the sonar receives echo returns from comers and projections like door handles, mirrors, wheels, etc.

The vehicle is to drive on a path parallel to the curve formed by the parked cars, keeping a constant distance from the cars (usually around 2.5 m). Therefore the parameters of that curve have to be calculated first, using the data from the local grid map. For reasons of computational simplicity and decreased noise sensitivity a least square fit of a straight line was chosen. All computations are performed with respect to the vehicle origin which is at a fixed position in the grid map. Data points for the line fit are selected by choosing only data points that appear in a specific area in the grid map. Thus for the environment represented in Fig. 10, only the right half of the map is searched for data points. The Position parameter gives a data point in the map in terms of vehicle coordinates. Since the direction of vehicle motion is along the y-axis, a line parallel to the vehicle would have a slope of $m = \infty$ (Fig. 3; Note that this coordinate system is defined by the vehicle's position system). To avoid this inconvenience, the vehicle coordinate system is rotated anticlockwise by 90". All following computations are now performed with the transformed coordinates $y = -x_{old}$ and $x = y_{old}$. Therefore the selected data points can now be represented as a discrete function $y_i = f(x_i)$. The sonar sensors sometimes return a spurious echo. These outliers generally degrade the performance of a least square fit. Hence a median filter is applied first along x_i on the data points given by $y_i = f(x_i)$. The filter is applied twice, using window sizes three and five points. The two parameters of the straight line $y = mx + c$ can now be found by using standard formulae for a least square fit for n data points (x_i, y_i):

$$m = \frac{n\sum x_i y_i - \left(\sum x_i\right)\left(\sum y_i\right)}{n\sum x_i^2 - \left(\sum x_i\right)^2}$$

$$c = \frac{\sum y_i}{n} - m\frac{\sum x_i}{n} \qquad (8)$$

Also, the standard error of estimate s_0 provides a measure of how well the data points could be fitted and is given by:

$$s_0 = \sqrt{\frac{\sum [y_i - (mx_i + c)]^2}{n - 2}} \qquad (9)$$

The data trajectory parameters m and c are stored and updated during each system cycle (Fig. 7). The line

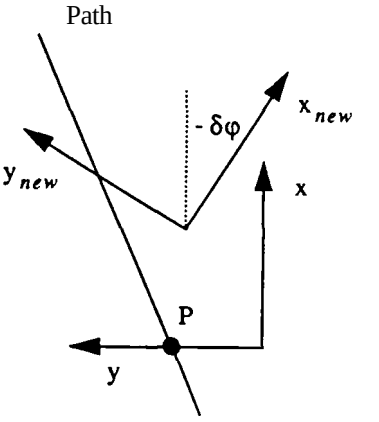


Fig. 11. Path parameter transformation.

can now be used by the controller to steer the vehicle (section 5.3). To ensure a consistent steering response, the line fit also has to be checked for validity. It may happen that the sensors do not detect any features for some driving distance. One reason in the case of parked cars for example may be a gap between cars and no reflectors on the road edge. Here the line fit will produce no valid output. Therefore if less than four data points are available, the output of the line fit is ignored. In this case the old path parameters are used and the vehicle will continue driving in the previously calculated direction until it encounters features again. Since the old path parameters are referenced to the vehicle position they still have to be updated to compensate for vehicle movement during one system cycle. The position and orientation increment during one system cycle is known from Eq. (1). Hence m is transformed as follows:

$$\varphi = \text{atan}(m), \qquad m_{new} = \tan(\varphi - \delta\varphi) \qquad (10)$$

In order to obtain c_{new}, point $P(x_1, y_1)$ is selected on the path where it intersects the y-axis (for convenience) as shown in Fig. 11. Using Eqs. (1) and (2) the coordinates of P in the new coordinate system (x_{1new}, y_{1new}) can be calculated. Hence c_{new} can be calculated by

$$c_{new} = y_{1new} - m_{new}\, x_{1new} \qquad (11)$$

5.2 *Path Filtering*

The method described above worked well in an environment that provided good reflectors and continuous smooth features. However, especially in the case of parked cars, problems arise when gaps are encountered or reflectors that do not belong to the feature being tracked are nearby. A typical situation is shown in

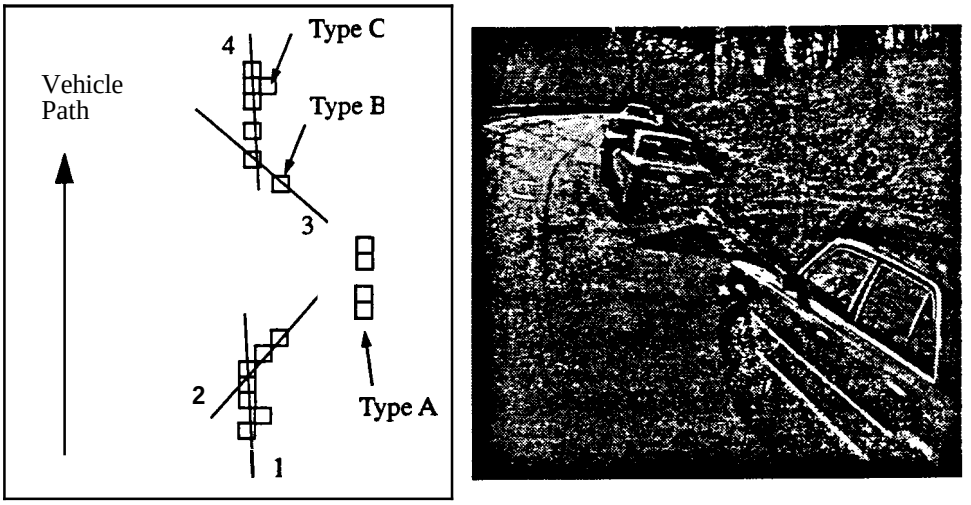


Fig. 12. **Removing outliers from least square fits: (a) Map display, (b) Typical corresponding scene.**

Fig. **12.** In the least square fit sequence **1 4 ,** only line fits **1** and **4** track the feature. Type ***A*** data does not belong to the feature being tracked, type B data is due to corner effects at a gap and type C data is a noisy range measurement. Type *B* data can lead to an undesired least square fit as shown by line fits **2** and 3. In order to reduce these errors, obtain a smoother steering response and make the system more robust against outliers, the values obtained from the least square fit are filtered and path parameters are updated by merging new information with past values.

An initial noise reduction is achieved by selecting data points only from a small window in the grid map: As we have some knowledge about the environment the vehicle is driving in, we can predict up to a certain degree where we should look for valid data in the map. This means in practice that only data points within a certain distance from the data trajectory are taken. This procedure removes outliers of type ***A*** as shown in Fig. **12.** Furthermore a point is selected only if it is within a certain distance and direction from previously selected adjacent points. This method removes outliers of type *C* and ensures that most points are already grouped close to a line segment.

Again a straight line is fitted using Eq. **(8).** For each distance δs_i travelled, we obtain a new value of gradient m_i. The change in gradient is then given by

$$\delta m_i = m; - m_{i-1} \tag{12}$$

and $\delta\varphi_i$ is the corresponding change in vehicle orientation during that interval. Since not all type B data is removed, there still may remain a problem with sudden large changes in orientation **as** shown by the least square fit sequence **1–4** in Fig. **12.**

These sudden changes in orientation basically appear **as** median noise. They can be filtered out by putting current and past readings of δm into a buffer of N values, passing a median filter across and then averaging over N values. The buffer is implemented as an ordered set,

$$m_{\text{Buffer}} = \{\delta m_i, \delta m_{i-1}, \delta m_{i-2}, \ldots, \delta m_N\} \tag{13}$$

Initially all buffer elements are set to zero, which means that it is assumed that the vehicle so far has driven exactly parallel to the data trajectory. During each update sequence, all elements are shifted right by one, the last element being discarded and the current result replacing the first element. Median filtering is achieved by replacing element δm_{i-k} with a median value, where **2k + 1** is the maximum possible window size of the filter. Since values towards the right in the buffer represent increasingly earlier points in timddistance, successive applications of a median filter can be achieved by replacing elements δm_{i-k-t} with the filtered value, where t is the discrete shift in timddistance. The new value $m_{\text{Path}}^{\{\text{new}\}}$ of the path parameter is then computed, compensating also for change in vehicle orientation during the averaging interval,

$$m_{\text{Path}}^{\{\text{new}\}} = \frac{1}{N} \cdot \sum_{k=1}^{N} \tan\left(\text{atan}(\delta m_k) - \sum_{i=1}^{k} \delta\varphi_i\right) \tag{14}$$

m_{Path} is then updated by merging the current and the new value for m_{Path} using a weighted average,

$$m_{\text{Path}}^{\{t+1\}} = \frac{m_{\text{Path}}^{\{\text{new}\}} + w \cdot m_{\text{Path}}^{\{t\}}}{1+w}, \quad \text{where } w \geq 0 \quad (15)$$

In a similar fashion as described above, the path parameter c_{Path} is updated. The weight w and the number of values N control how close the path parameters m_{Path} and c_{Path} should follow the actual data points. If a lot of noise is present in the data, the path parameters should be influenced only slightly, whereas if little noise is present, the data should be followed closely. An estimate of the noise is given by the standard error s_0 from Eq. (9) and w and N are adjusted accordingly; i.e. small w and N for a small standard error and large w and N for a large standard error. The actual values were determined empirically. As a result the steering of the vehicle is now much smoother. From m_{Path} we can also obtain an estimate of the road curvature Υ over a certain distance Δs travelled, using

$$\Upsilon = \frac{\Delta\phi}{\Delta s}, \quad \text{where } \Delta s = \sum \delta s_i,$$
$$\Delta\phi = \text{atan}\left(m_{\text{Path}}^{\{t+1\}}\right) - \text{atan}\left(m_{\text{Path}}^{\{t\}}\right) + \sum_{N} \delta\varphi_i \quad (16)$$

A drawback of the method is its slow reaction to a relatively sudden change in road direction. This effect is also due to the short range of the ultrasonic sensors and the fact that almost no echoes are received at large angles of incidence. In the case of features being tracked on the right side, the system is able to handle curves to the right since data points slowly move away from the vehicle and the vehicle can follow with a slight delay. On the other hand a curve to the left poses a problem because by the time the vehicle recognizes that it should change direction, it has usually come already too close to the feature and has no space left to make a sharp left turn anymore.

For this reason a monitor is added which monitors a particular area for objects towards the right hand side of the vehicle as indicated in Fig. 3. If objects are detected in this area, then the bitvector representing inhibited arcs is checked for each occupied grid cell as described in section 4. If the resultant bitvector does not permit an arc as commanded by the path tracker, the obstacle avoidance procedure overides the tracker. In this case, the closest permitted left turning radius is selected. This ensures that any part of a feature being tracked to the right of the vehicle will be avoided. In general therefore, the following condition has to be true for all occupied grid cells i in the monitored area, using curvature:

$$Curv_{\text{tracker}} < \text{Min}_i(Curv_{Lb})_i$$

Altogether, the system performs well when tracking a wall or a feature that changes curvature smoothly. When tracking parked cars it does not perform as well in curves and fails when the road curvature becomes sharp. The reason here is that often parked cars are not very well aligned and even on straight road stretches parked at different angles to each other. The maximum range of the sensors is simply too short to detect these configurations well enough. Figure 10 shows the vehicle tracking parked cars, detecting a gap and preparing for reverse parking.

5.3 *Path* Tracking

The actual path tracking procedure used by the system is based on the ***pure pursuit*** tracking described in (Amidi, 1990). The goal to be achieved is to drive the vehicle on a path parallel at a constant distance D_{ref} to the data trajectory given by m and c. Since the Navlab cannot move in any arbitrary direction from a given position, a point on the vehicle has to be selected that will be kept at distance D_{ref} from the data trajectory. As the vehicle uses Ackerman steering, the front part of the vehicle experiences the largest displacement when the steering angle is changed. Therefore we do not use the vehicle position which is at the centre of the rear axle, but the position of the front part of the vehicle to be kept at D_{ref}. Using data trajectory parameters m and c, the spatial displacement ε and angular displacement β for points P_0 or P_0^* from the desired path can be calculated (Fig. 13):

$$\beta = \text{atan}(m) \quad (17)$$

since vehicle orientation is always fixed in the map.

$$\varepsilon = D_{\text{act}} - D_{\text{ref}}$$
$$D_{\text{act}} = y_F \cdot \cos\beta, \quad y_F = \text{m} \cdot x_F + \text{c} \quad (18)$$

where x_F is the distance between vehicle position and the point selected at the front part of the vehicle as reference for path tracking.

If the vehicle is now displaced by ε from the desired path, the ***pure pursuit*** method is used to drive the vehicle back onto the desired path. In this method, an initial goal point is selected on the desired path by the look ahead distance L (Fig. *14).* An arc is fitted between

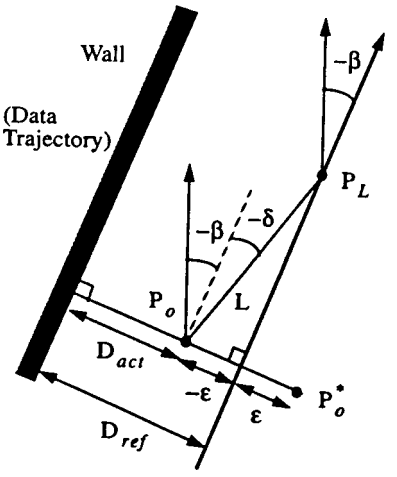


Fig. 13. **Path tracking parameters.**

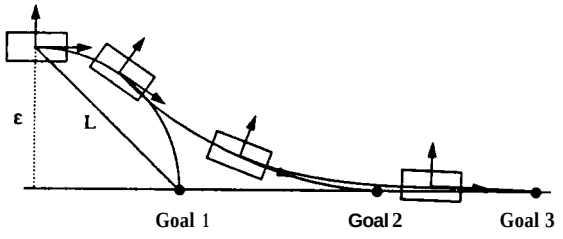


Fig. 14. **Pure pursuit.**

the current vehicle position and this goal point. The vehicle now moves a small distance during one system cycle and then a new goal point is selected. If the cycle time is small enough, a smooth path is executed that drives the vehicle back onto the desired path as shown in Fig. 14. The path tracking procedure works for data trajectories on either side of the vehicle. Only D_{ref} changes in sign.

The steering radius R during each cycle time can then be calculated by:

$$R = -\frac{L}{2\sin\phi}$$

$$\phi = \beta + \delta, \quad \delta = \operatorname{asin}\frac{\varepsilon}{L} \tag{19}$$

If the vehicle drives on a narrow road with objects on both sides, keeping D_{ref} at a fixed value may not be appropriate as the vehicle may come too close to the other side. In this case we obtain D_{actR} and D_{actL} from the right and left side respectively and can use this information to calculate a desired path that lies in the center between objects on both sides.

6 Parallel Parking

With the help of the basic navigation functions described in sections **4** and **5**, the vehicle can now search for a parking gap in a row of parallel parked cars and park itself autonomously by using the procedures described in this section.

The vehicle starts out in tracking mode, driving parallel at a constant distance to a row of parked cars (refer to section 5.1) and searches for a parking gap as shown in Fig. 10. If the sideways looking sensors do not see any new objects that can be tracked, it is assumed that the vehicle has reached the edge of a potential gap. The procedure for parking the vehicle is then as follows (Fig. 15):

Since the current'sensors cannot see the curb or the end of the gap, the vehicle's driving direction is predicted from the orientations of the parked cars passed previously (Fig. 15(a), Position B-C). When the vehicle drives past the next car, it detects the end point of the gap. Both gap endpoints are tagged as shown in Fig. **15** and the algorithm checks now whether there are any obstacles present within the gap and whether Gap length L_G is sufficient.

Since parking is done via a fixed path trajectory shape, the minimum length of the gap depends on several constant parameters and is given by Eq. *(20).* The vehicle is then aligned in parallel to the car in front of the gap (Position C–D).

$$L_G = 2R_{\min}\sin\alpha + \frac{D_{ref} + 0.5W_G - 2R_{\min}(1-\cos\alpha)}{\tan\alpha} - Y_P \tag{20}$$

where W_G is the width of gap (average width of a car), $R_{\min}$ is the minimum turning radius of the vehicle and α is a given constant. Here $\alpha = 30°$. The values selected for Y_P and α ensure that the side of the vehicle will not hit the car parked in front of the gap. Refer to Appendix A and Eq. *(22)* for a derivation of the above equation.

If there is a lateral offset between the distance D_{ref} that the tracker kept from parked cars and the current lateral distance of the vehicle with respect to the car in front of the gap, then this offset is corrected by moving the vehicle an equal distance through two opposite equal arcs (Fig. 15(a), D–E). This means the vehicle changes its heading by an angle 8 on the first arc and by an angle $-\theta$ on the second arc. Hence vehicle orientation is the same at position D and E. The angle depends

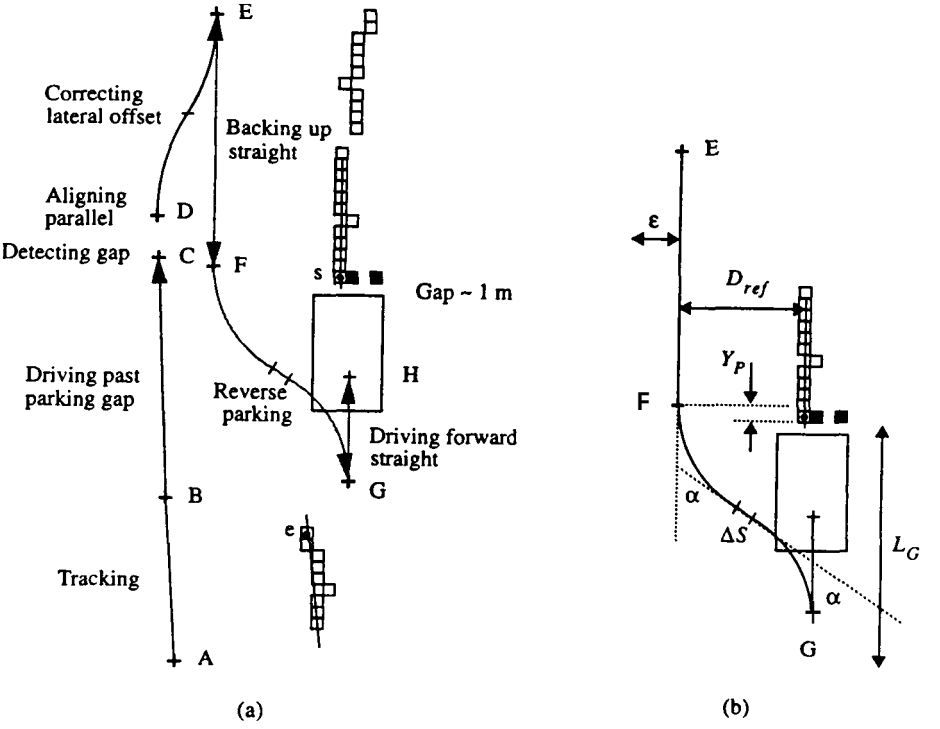


Fig. 15. Parallel parking.

on the lateral offset ε and can be calculated by,

$$\theta = \operatorname{acos}\left(1 - \frac{\varepsilon}{2R}\right) \tag{21}$$

(see also Appendix A, substituting θ for α).

The vehicle stops now and then drives straight in reverse until the vehicle position is within distance Y_P of the gap edge labelled 's' (E–F, see Fig. 15(b)).

For reverse parking, first the minimum turn radius $R_{\min}$ to the right is commanded until the vehicle heading has turned through an angle $-\alpha$. Then the vehicle drives straight again in reverse for adistance **AS.** Since α is a given constant for reasons explained previously, **AS** determines how close the vehicle is parked to the curb and is given by (see Appendix A),

$$\mathbf{AS} = \frac{|D_{\text{ref}}| + 0.5W_G - 2R_{\min}(1 - \cos\alpha)}{\sin\alpha} \tag{22}$$

Currently, the curb cannot be sensed and therefore the width of the gap W_G is assumed to be an average of the width of a typical car. Following, the minimum turn radius $-R_{\min}$ to the left is commanded until the vehicle heading has turned through an angle α (Position F-G).

The vehicle now reverses its direction and in the last step, moves forward straight again until the gap between the vehicle and the car in front is approximately one metre (G–H). The vehicle has now parallel parked.

During the reverse driving part of the parking procedure the sensors are not used and the vehicle path is computed from the data points previously collected in the map. In the last part of the parking procedure (G–H), the sensors are used again in order to detect the vehicle parked in front of the gap.

7 Results and Conclusions

Parking or docking maneuvres are important autonomous vehicle **tasks.** The system described has been successfully used to drive the Navlab parallel to parked cars in a city street, detect a parking space and autonomously park the vehicle. The sonar system also drove the Navlab on a dirt road next to a railroad track, using the railroad to guide the vehicle. The sonar is successfully integrated into two other systems that drive the Navlab. The first one is **YARF** (Kluge & Thorpe, 1990), which drives the robot on city streets. YARF uses colour vision for road following but cannot detect if obstacles obstruct the vehicle's path and thus cannot adjust the velocity accordingly. The sonar system

takes over that task and sends velocity commands to YARF via the TCX toolkit (Fedor, 1993), using TCP/IP. The sonar is also integrated into the new architecture DAMN (Distributed Architecture for Mobile Robot Navigation) of the Navlab. The outputs of several modules that can drive the vehicle are used here to decide on an optimum path for the vehicle (Payton, Rosenblatt, & Keirsey, 1990). The sonar module sends votes for a number of discrete steering arcs, voting against arcs that would result in a collision with an obstacle and voting for obstacle free arcs. The system then selects an obstacle free path to a goal point. Communication is again facilitated via the TCX toolkit.

The sonar system proved to work reliably in a variety of different situations. There were no major problems with false returns or sensor noise that could not be dealt with. One reason is most probably that an outdoor environment like a road is generally less cluttered than most indoor environments where autonomous vehicles are used. Outdoor objects tend to be large, having usually enough comers and projections that reflect ultrasound well. Care has to be taken in avoiding reflections from the ground. This problem can be solved in most cases by mounting the sensor high enough above the ground and pointing it slightly upward. The system can also be easily integrated with other vehicle navigation systems or adapted to other vehicles.

A drawback of using ultrasound in air is the limitation of range and data update due to high attenuation and low speed of sound. At low vehicle speeds this fact does not matter that much and the system works very well for slow speed maneuvres and parallel parking. However, the current system design is not limited to using only ultrasonic sensors.

We have successfully integrated a laser scanner (ERIM) into the system. Here obstacles are detected from 3-D laser range images and stored in the grid map. Votes for discrete steering arcs are then computed and sent to the arbitrator in DAMN. This system has been used in cross country navigation, driving the Navlab at speeds up to 3 m/s over a distance of 1 km (Langer, Rosenblatt, & Hebert, 1994).

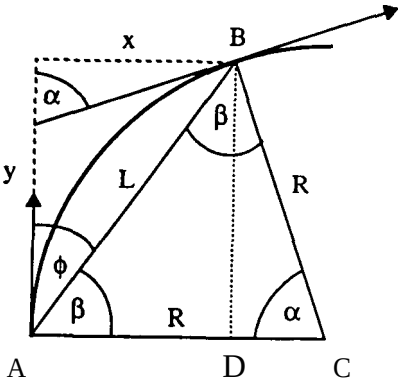
Currently, we are planning to integrate a millimeter wave radar system with a range upto 200 metres for driving on highways and in city traffic.

Acknowledgments

This research is partly supported by contracts from DARPA, titled "Robot System Development and Testing" (monitored by TACOM) and "Perception for Outdoor Navigation" (monitored by ETL), and partly supported by a grant from NSF titled "Annotated Maps for Autonomous Underwater Vehicles". The authors would like to thank Valentin Magori and Siemens AG for providing some of the sensors and helpful information. Many thanks to Martial Hebert for his advice and comments.

Appendix A

Calculation of parameters for pure pursuit and obstacle avoidance:



Triangle ABC is equilateral, since $\overline{AC} = \overline{BC} = R$

$$\Rightarrow \quad \alpha = 2\phi \tag{1}$$

$$\sin\alpha = \frac{Y}{R} \qquad \cos\phi = \frac{Y}{L}$$

$$\therefore \quad R\sin\alpha = L\cos\phi$$

$$\Rightarrow \quad R = L \cdot \frac{\cos\phi}{\sin 2\phi} = \frac{L}{2\sin\phi} \tag{2}$$

$$\sin\phi = \frac{x}{L} \qquad L = \sqrt{x^2 + y^2}$$

From (2),

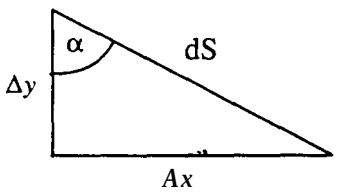
$$\Rightarrow \quad R = \frac{L^2}{2x} = \frac{x^2 + y^2}{2x} \tag{3}$$

Additional calculation of parameters for parallel parking (Eqs. (20), (21) and (22) in section 6):

From triangle BDC,

$$\cos\alpha = \frac{R - x}{R} \quad \Rightarrow \quad x = R(1 - \cos\alpha) \tag{4}$$

Calculation of dS in sequence F-G and part of gap length in Eq. (20), section 6:



$$dS = \frac{Ax}{\sin \alpha} \qquad \Delta y = \frac{Ax}{\tan \alpha} \tag{5}$$

References

Amidi. Omead 1990. ***Integrated Mobile Robot Control.*** Technical Repon, Robotics Institute, Carnegie Mellon University.

Borenstein. J. and Koren, **Y.** 1990. Real-time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments. In Proc. ***IEEE Conference on Robotics and Automation,*** Cincinnati, pp. 572–577.

Elfes. A. 1986. A Sonar-Based Mapping and Navigation System. In ***Proc. IEEE Conference on Robotics and Automation.***

Fedor. C. 1993. TCX, Task Communications User's Manual. Internal Report, The Robotics Institute, Carnegie Mellon.

Kluge, Karl and **Thorpe** Charles. 1990. Explicit Models for Robot Road Following. ***Vision and Navigation: The Carnegie Mellon*** Navlab. Kluwer Academic Publishers. Chapter 3.

Langer. D., Rosenblatt, J.K., and Hebert, M. 1994. A Reactive System For Off-Road Navigation. In ***Proc. IEEE Conference on Robotics and Automation.***

Leonard, J. and Durrant-Whyte, H. 1990. Application of Multi-Target Tracking to Sonar-based Mobile Robot Navigation. In Proc. IEEE ***Conference on Decision and Control.***

Magori, V. and Walker, H. 1987. Ultrasonic Presence Sensors with Wide Range and High Local Resolution. In ***IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control,*** UFFC-34(2).

Moravec, H.P. 1988. Sensor Fusion in Certainty Grids for Mobile Robots. ***AI Magazine***, 9(2):61–77.

Payton, David W., Rosenblatt Kenneth, and Keirsey, David M. 1990. Plan Guided Reaction. ***IEEE Journal on System, Man and Cybernetics.***

Thorpe, C. and Gowdy. J. 1990. Annotated Maps for Autonomous Land Vehicles. In ***Proceedings of DARPA Image Understanding*** *Workshop*, Pittsburgh PA.

Thorpe. Charles, Heben, Martial. Kanade. Takeo, and Shafer, Steven. 1991. Toward Autonomous Driving: The CMU Navlab. Part I — Perception. In IEEE *Expert.*

Dirk **Langer** received the Dipl.-Ing. degree in Electrical Engineering in 1989 from the Technical University in Munich, Germany. From 1989 to 1992 he was a Visiting Scholar at Camegie Mellon University, working on ultrasonic sensors for robot navigation and side scan sonar for underwater mapping. Currently he is a PhD student in Robotics **at** Camegie Mellon University.

His current research focuses on the design and development of a radar sensor for automated on-road driving.

Other research interests include robot navigation, perception and vehicle control.

Charles Thorpe is **currently** a Senior Research Scientist at the Robotics Institute of CMU. His interests **are** in computer vision, planning, and control of robot vehicles operating in unstructured outdoor environments. He directs the research on the Navlab project, which is developing mobile robots that use perception and mapping for outdoor autonomous driving. The Navlab project has long been supported by ARPA, for developing techniques for driving in hazardous environments. Recently, the Navlab project has expanded to include work for the US Department of Transportation, for studies of crash avoidance technology for highway driving. and for building the Automated Highway System.

Dr. Thorpe is also involved with robots for planetary exploration and for underwater mapping. He received his PhD in Computer Science from CMU, in 1984. and his BA in Natural Science from North Park College in Chicago.