Visual Support for Navigation in the Polly System

Ian Horswill
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139
ian@ai.mit.edu

Abstract

In this paper I will discuss a system which uses vision to guide a mobile robot through corridors and freespace channels. The system runs in an unmodified office environment in the presence of both static and dynamic obstacles (e.g. people). The system is also among the simplest, most effective, and best tested systems for vision-based navigation to date. The performance of the system is dependent on an analysis of both the task and the environment in which it is performed. I will describe both of these and discuss how they simplify the computational problems facing the robot, as well as the performance of the resulting system. Finally, I will briefly discuss work in progress regarding place recognition and point-to-point navigation.[1]

Introduction

Navigation is one of the most basic problems in robotics. Since the ability to safely move about the world is a prerequisite of most other activities, navigation has received a great deal of attention in the AI, robotics, and computer vision communities. One of the limiting factors in the design of current navigation systems, as with many other robotic systems, has been the availability of reliable sensor data. Most systems have relied on the use of sonar data [7][9], or on vision [10][3][5][6][13][1]. In all cases, the unreliability of the available sensor data was a major concern in the research. Some researchers have even avoided the use of sensor data entirely in favor of precompiled maps [11].

In this paper, I will discuss a very simple vision-based corridor following system which is in day-to-day use on Polly, a low cost mobile robot. The system is notable in that it is very fast (14 frames per second in the current system), very well tested, and uses only cheap "off-the-shelf" hardware. A major source of this simplicity is an analysis of both the task and the niche of this simple agent. Such an analysis helps make clear the dependence of an agent on its environment and provides guidance for the design of future systems.

The Polly system

Polly is a low cost autonomous robot[2] intended for use as a platform for developing low cost, real-time vision systems for guiding situated action. The computational hardware on Polly consists of a 16 MIP digital signal processor (Texas Instruments TMS320C30) with 64K 32-bit words of high speed ram[3], a video frame buffer/grabber, a simple 8-bit microcontroller (M68HC11) for I/O tasks, and commercial microcontrollers for voice synthesis and motor control (see figure 1). Nearly all computation is done on the DSP. The implementation goal of the project is to develop an efficient visual system which will allow the robot to run unattended for extended periods (hours) and to give primitive "tours" of the MIT AI lab. Our general approach to design has been to determine what particular pieces of information are needed by the agent to perform its activities, and then to design a complete visual system for extracting each piece of information. Thus, the agent might have distinct systems for answering questions such as "am I about to hit something?" or "what is the axis of this corridor?" If these systems are simple enough, they can be run in parallel very efficiently. While one would eventually like to build a single system which can compute all of these pieces of information, it is useful as a research strategy to treat each piece of information as a separate computational problem.

[1]Support for this research was provided in part by the University Research Initiative under Office of Naval Research contract N00014-86-K-0685, and in part by the Advanced Research Projects Agency under Office of Naval Research contract N00014-85-K-0124. Thanks to Steve White for reading an early draft of this paper.

[2]Polly was built for $20K (parts cost in the U.S.), but today, a roughly comparable, or perhaps even faster, system could be bought for roughly $10K US.

[3]The DSP includes an additional 1Mb of low speed ram, which is not presently in use. At present, only approximately 10KW of RAM are in use.

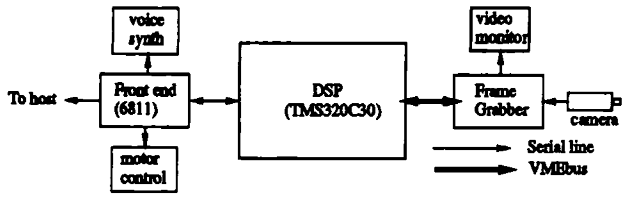

Figure 1: Computational hardware of the Polly robot.

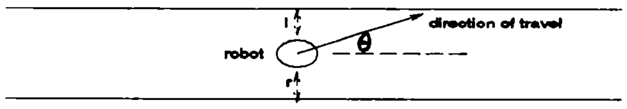

Figure 2: Corridor following.

Corridor following

Corridor following is a common navigation task in office environments. Office buildings tend to consist of long corridors lined with rooms on either side, thus much of the work of getting from one room to another consists of driving along a series of corridors. The corridor follower described here is intended to be used as one component among many that cooperate to allow the robot to participate in its projects.

Corridor following can be broken into the complementary problems of keeping aligned with the axis of the corridor and keeping away from the walls (see figure 2). This amounts to keeping the variable θ small, while simultaneously keeping l and r comfortably large. Since Polly can only move in the direction in which it's pointed, these variables are coupled. In particular, if the speed of the robot is s, then we have that:

$$\frac{dl}{dt} = -\frac{dr}{dt} = s \sin \theta$$

so moving away from a wall requires that Polly temporarily turn away from the axis of the corridor. Thus the problem for the control system amounts to controlling s and $\frac{d\theta}{dt}$ so as to keep l and r large and θ small and so the problem for the visual system amounts to determining when one of these conditions is violated.

Computational properties of office environments

In the general case, estimating distance and parsing the visual world into objects are both very difficult problems. For example, figure/ground separation can be arbitrarily difficult if we consider pathological situations such as camouflage or crypsis, which can require preditors to learn to recognize prey on a case by case basis [12, p. 260]. Fortunately, office environments, which form the habitat of the Polly system, are actively structured by both designers and inhabitants so as to facilitate their legibility. Thus computational problems can often be greatly simplified by taking into account these special properties, or *constraints*, which partially define the agent's habitat (see [4]).

One such property is that office environments have a flat ground plane, the floor, upon which most objects rest. For a given height and orientation of the camera, the distance of a point P from the camera will be a strictly increasing function of the height of P's projection in the image plane, and so image plane height can be used as a measure of the distance to objects resting on the floor[4]. While this is not a linear measure, and the exact correspondence between heights and distances cannot be known without first knowing the specifics of camera, it is a perfectly useful measure for determining which of two objects is closer, whether an object is closer than a certain threshold, or even as an (uncalibrated) measure of absolute distance. This property was referred to as the *ground plane constraint* in [4].

Another important property of office environments is that they are generally carpeted and their carpets generally have only fine-scale texture. That is to say that from a distance, the carpet will appear to have a uniform reflectance. If this is true, and if the carpet is uniformly illuminated, then the areas of the image which correspond to the floor should have uniform image brightness, and so any violation of this uniformity must be an object other than the floor. This property, called the *background texture constraint* in [4], can greatly simplify the computational problem of figure-ground separation.

The control system

At any given time, the corridor follower has to make a decision about how to turn and whether or not to brake. The forward velocity is based on a measure c' (estimated by the visual system, see the next section) of the distance to the nearest object in front of the robot:

$$s = \min\left(v_{max}, \max\left(0, \frac{v_{max}}{d_{safe} - d_{stop}}(c' - d_{stop})\right)\right)$$

Thus the robot drives at a rate of v_{max} for distances over d_{safe} and slows down linearly until it stops at a distance of d_{stop}.

The steering rate $\frac{d\theta}{dt}$ is controlled by four pieces of information: the orientation of the axis of the corridor, θ, a confidence measure for θ, and measures of the distances l' and r' to the nearest obstacles (including walls) on the left and right respectively. Again, all these are estimated by the visual system. The system drives the steering motor at a rotational velocity of

$$\frac{d\theta}{dt} = \alpha(l' - r') + \beta\theta$$

if it is confident of its measure of θ, otherwise with a velocity of $\alpha(l' - r')$. Here α and β are gains (constants) adjusted empirically for good results.

[4]This observation goes back at least to Euclid. See [2].

In practice, we have not measured the variable θ directly, but instead have used the image plane x coordinate of the projection of the axis of the corridor, which is more easily measured. The projection x is equal to $k \tan^{-1} \theta$, where k is determined by the focal length and resolution of the camera. Thus the actual control law used in our system is:

$$\frac{d\theta}{dt} = \alpha(l' - r') + \beta \tan^{-1} \theta$$

In our experience, this has lead to a stable control system. This seems reasonable, since $\theta \approx \tan^{-1} \theta$ for θ near 0, and since the visual system rejects values of θ outside the field of view of the camera.

The visual system

The visual system estimates the axis of the corridor and three distance measures from 64 × 48 pixel grey-scale images covering a field of view of 110 degrees (1.9 radians). As was mentioned above, the axis of the corridor is represented by the x coordinate of its image-plane projection. This can be estimated by finding the vanishing point of the parallel lines forming the edges of the corridor. This can, in turn, be computed by extending to infinity the perpendicular to the intensity gradient of each pixel with significant intensity variation and searching for intersections. For a point (x, y) in the image, this perpendicular line is given by

$$\{(x', y') : x' = x + s\frac{dI}{dy}, y' = y - s\frac{dI}{dx}, \text{for some } s\}$$

For points lying on edges, this perpendicular will be the local tangent line of the edge at that point.

The resulting lines could then be clustered to find their intersections and the strongest one chosen as corridor's vanishing point. This is the approach used by Bellutta, et. al. [1], which was an inspiration for this work. We can optimize this system by using knowledge of the camera's geometry to assume the y-coordinate (height) of the vanishing point. In the case of Polly, the vanishing point is roughly at y=0. This allows us to reduce the estimation problem from two parameters to only one. The x-coordinate of the vanishing point can then be estimated by intersecting each line with the line y=0 to find the x-coordinate of the intersection:

$$x_{vanishing-point} = x + y\frac{dI/dy}{dI/dx}$$

The system then finds the mean and variance of the estimated x coordinates. The mean is used as the estimate of the axis of the corridor and the variance as the confidence measure.

There are two problems involved in estimating the left, right, and center distances: figure/ground separation to find the walls in the image, and depth estimation to determine the distances to them. As was discussed in section , these can be greatly simplified by the background texture and ground-plane constraints, respectively. The system finds boundaries between the floor and the walls or other objects by running an edge detector over the image. It then constructs a radial depth map[5], D, indexed by x coordinate:

$$D(x) = \min\{y : \text{the image has an edge at } (x, y)\}$$

Thus $D(x)$ is the y coordinate of the lowest edge found in the xth column of the image, and hence is a measure of the distance to the nearest object in that direction. The left, right, and center distances, l', r', and c', can then be computed by taking the minimum over all columns in the left, right and center regions respectively.

A number of things are worth noting here. First of all, l' and r' are not necessarily the distances to the walls. They are simply the distances to the nearest non-floor objects on the left and right sides of the image. Fortunately, this is not a problem, since if there are other objects in the way it will simply cause the robot to steer around them, thus conferring on the robot a limited object avoidance capability. If there are no such objects, then there is no difference anyhow. Thus having the system make no distinctions between walls and other obstacles is actually advantageous in this situation. The second thing worth noting is that the distance measures are nonlinear functions of the actual distances[6]. For some applications this might be unacceptable, but for this application we are mostly concerned with whether a given object is too close or whether the left side or the right side is closer, for which purposes these nonlinear measures are quite adequate. Finally, since no camera has a 180 degree field of view, l' and r' are not even measures of the perpendicular distances to the walls, l and r, but rather are measure of the distance to the closest point *in view*. Again, this is not a problem in practice, partly because our camera has a relatively wide field of view, and partly because for a given orientation of the robot, the perpendicular distance is another monotonic and strictly increasing function of the measured distance, and *vice versa*.

Evaluation

The corridor follower has been running for six months and is quite stable. It has seen at least 150 hours of service with several continuous runs of one hour or more. This makes it one of the most extensively tested and reliable visual navigation systems to date. We have been able to run the system as fast as our robot base could run without shaking itself apart (approximately 1 m/s). While there are cases which will fool the braking system (see below), we have found the system to be quite reliable in general.

[5]The map is called "radial", because its entries correspond to distinct directions along the ground plane.

[6]The actual function is a quotient of linear equations whose coefficients are determined by the camera parameters.

Efficiency

The system is very efficient computationally. The present implementation runs at 14 frames per second on the DSP. This implementation is heavily I/O bound however, and so it spends much of its time waiting for the serial port and doing transfers over the VMEbus to the frame grabber and display. We expect that performance would be noticeably better on a system with a more tightly coupled DSP and frame grabber. The efficiency of the system allows it to be implemented with a relatively simple and inexpensive computer such as a DSP. The modest power requirements of the computer allow the entire computer system to run off of a single motorcycle battery for six hours or more.

The simplicity and efficiency of the system make it quite inexpensive compared to other real-time vision systems. C30 DSP boards are now available for personal computers for approximately $1-2K US and frame grabbers can be obtained for as little as $400 US. Thus the corridor follower would be quite cheap to install on an existing system. We are also working on a very inexpensive hardware platform for the system which we hope will cost less than $200 US.

Failure modes

The system runs on all floors of the AI lab building on which it has been tested (floors 3-9), except for the 9th floor, which has very shiny floors. There the system brakes for the reflections of the overhead lights in the floor. We expect the system to have similar difficulties in the basement. The present system also has no memory (e.g. a local map) and so cannot brake for an object unless it is actually in its field of view. This sometimes causes problems. The system also cannot brake for an object unless it can detect an edge on or around it, but this can more or less be expected of all vision systems. The system's major failure mode is braking for shadows. If shadows are sufficiently strong they will cause the robot to brake when there is in fact no obstacle. This is less of a problem than one would expect because shadows are generally quite diffuse and so will not necessarily trigger the edge detector. Nevertheless, the edge detector is biased to prefer vertical edges to horizontal ones since shadows in this environment tend to be horizontal (see section). Finally, the 7th floor of the lab, where the robot spends most of its time, does not have a single carpet, but several carpets, each with a different color. The boundaries between these carpets can thus be mistaken for obstacles. Fortunately for us however, the carpets boundaries always appear horizontal in the image and their changes in grey-scale intensity are small enough so that the do not actually cause the edge detector to fire.

Performance outside corridors

While the system was designed to navigate corridors, it is also capable of moving through more complicated spaces. Its major deficiency in this situation is that

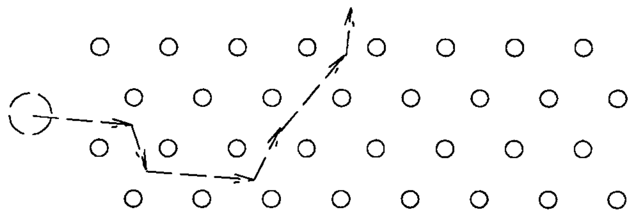
Figure 3: A forest environment.

there is no way of specifying a desired destination to the system. Effectively, the system acts in a "point and shoot" mode: it moves forward as far as possible, veering away from obstacles, and continuing until it reaches a dead end or is stopped externally. The system is also non-deterministic in these situations. When the robot is blocked by an object, it will turn either left or right depending on the exact position and orientation of the robot and object. Since these are never exactly repeatable, the robot is effectively non-deterministic. Thus in a "forest environment" such as figure 3, the robot could emerge at any point or even get turned around completely. The system's performance is good enough however that a higher-level system can lead it though a series of corridors and junctions by forcing the robot to make a small open-loop turn when the higher-level system wants to take a new corridor at a junction. The corridor follower then realigns with the new corridor an continues on its way.

Extensions

A number of minor modifications to the algorithm described above are worthwhile.

Vertical biasing

As discussed above, shadows and bright lights radiating from office doors can sometimes be sufficiently intense to trigger the edge detector. Since these shadows always radiate perpendicular to the wall, they appear horizontal in the image when the robot is successfully aligned with the corridor. By biasing the edge detector toward vertical lines, we can make it less sensitive to these shadows. The present system weights vertical lines twice as heavily as horizontal lines, thus a horizontal line must be twice as strong as a vertical line to trigger the edge detector.

Wall following

When the system described above reaches a large open space, the single wall which is in view will act as a repulsive force, causing the robot to turn away from it until there is nothing in view whatsoever. Thus it naturally tends toward situations in which it is effectively blind. While this is sufficient for following corridors, and is in fact a very good way of avoiding obstacles, it is a very bad way of actually getting to a destination unless the world consists only of nice straight corridors. This problem can be fixed by modifying the steering control so that when only a single wall is in view, the robot will try to keep the other wall at a constant distance. Thus, in the case where only the left wall is in view, the control law becomes:

$$\frac{d\theta}{dt} = \alpha(l' - d)$$

Where d is the desired distance for the wall.

Stop acceleration

When the robot is forced to stop, such as when it reaches a corner, it can take a relatively long period of time for it to turn far enough to enable it to move forward again. This is easily fixed by forcing the base to turn at a fixed (large) speed, but in the same direction it would otherwise turn, whenever the base is stopped.

Work in progress

The corridor follower is intended to be one component of a more general navigation system[7] presently under development. To navigate in its office environment, the robot needs to be able to detect important features of the environment such as the junctions of corridors, to keep track of its location, and to determine on each clock tick whether to continue following the present corridor or to initiate a turn. To date, I have implemented prototype versions of the first two of these.

In addition to the information discussed above, the robot's visual system presently delivers the following signals to the central (control) system. They are recomputed from a new image on every clock tick.

- **`blocked?`** True if the robot is stopped or is about to stop because there is an obstacle too near it. Computed from c'.
- **`in-corridor?`** True if there is a large open space in front of the robot. Computed from c'
- **`open-left?`, `open-right?`** True if there is a large open space to the left/right into which the robot might be able to turn. Computed from r' and l'.
- **`left-turn?`, `right-turn?`** True if open-left/open-right and the robot is in a corridor and aligned with it's axis, that is, if the robot is at a junction.
- **`open-region?`** True when open-left and open-right.
- **`dark-floor?`** True when the pixel on the bottom and in the middle of the screen is dark (grey level less than 80).
- **`light-floor?`** True when the pixel on the bottom and in the middle on the screen is light (grey level greater than 110).
- **`edge-count`** The number of edge pixels found by the edge detector.
- **`blind?`** True when fewer than 40 edge pixels have been found. (This is useful for determining if the robot may be pointing toward a wall, and thus unable to see the floor).

[7]To be specific, the navigation system is intended to be able to move to specified locations within the corridors and open areas of the lab. It is not intended to navigate within the offices of the lab.

In addition, the motor system provides the following signals:

- **`turn-rate`** The current velocity being sent to the robot's turn motor.
- **`turning?`** True if abs(**`turn-rate`** > threshold.
- **`aligned?`** True when not turning, i.e. when the robot is aligned with the corridor.
- **`aligned-long-time?`** True when **`aligned?`** has been true for at least three clock ticks.

This signals have proved sufficiently stable to implement a simple plan executive, which has been useful for debugging and gathering data for the place recognition system.

The current place recognition system is based on the observation that the corridor follower keeps the robot at a deterministic distance and orientation relative to the walls. Thus the robot's view of any particular scene within a corridor will be very nearly invariant. If this is true, then a very simple technique, such as template matching, ought to suffice to distingush places. In practice, we've used a system which is somewhat more complicated, based on a structured associative memory similar to a frame system [8]. All place frames in this system have identical structure, thus making them relatively simple to match. At present, place frames contain a low resolution (16×12), unnormalized greyscale image of the scene, the direction (north, south, east, west) in which the robot was facing when it took the image, the approximate position of the place[8], and a text string describing the place. On each clock tick, the frame system compares the current low-res image obtained from the frame grabber, together with the last known position, to each of the frames, so as to obtain a matching score:

$$||I - F_I||^2 + \alpha(\text{abs}(x - F_x) + \text{abs}(y - F_y))$$

Where I is the current image, x and y are the position of the last matched frame, and F is the frame being matched. If the score of the frame with the lowest score is below threshold, then that frame is taken as the current place, and the current position is updated.

As of this writing, the system appears to reliably recognize eight places in the lab, provided that there are

[8]The position is not intended to be metrically accurate, it is mostly intended to allow the system to determine whether one place is north, south, east, or west of another.

no people or other obstacles in the scene, but since the system has only been operational for a week, we have only been able to perform a few dozen test runs. A more substantive appraisal will have to wait for the rest of the navigation system to be completed. As is to be expected, the system fails to match places which are not in corridorssince the viewpoint constraint no longer holds in these situations. Similarly, it cannot match when it is steering to avoid an obstacle, or when when the appearance of the world has changed. Finally, this implementation is limited in that the camera faces downward for a clear view of the floor, which makes it difficult or impossible to distinguish many places. Here the ability to steer the camera would be extremely useful.

Conclusions

Curiously, the most significant things about the system are the things which it does not do. It does not build or use detailed models of its environment, it does not use carefully calibrated depth data, it does not use high resolution imagery, and it is not designed to run in arbitrary environments. Indeed, much of its power comes from its specialization.

One may be tempted to object that this system is too domain specific and that more complicated techniques are necessary to build practical systems for use in the real world. I think that this is misguided however. To begin with, even if one had a truly general navigation system, its very generality may well make it much slower than the system discussed here. The general system may also require allocating scarce cognitive or attentive resources which would be better used for other concurrent tasks. In either case, a more intelligent approach might be to use *both* systems: the simple system in simple situations and the more cumbersome system in others. This makes a reconfigurable architecture, such as Ullman's Visual Routine Processor [14], quite attractive since it can be quickly configured by the central system to perform the most efficient algorithm which is compatible with the immediate task and environment.

One should also remember that very few systems are truely general. Stereo-based systems contain hidden assumptions about the presence of texture, the inverse of the background texture assumption decribed here. Other systems may assume a piecewise-planar world. In general, we cannot trust claims of generality unless they are accompanied by a detailed analysis and empirical tests in a variety of domains.

While the approach used in this paper may not work for all tasks, it does suggest that it is possible to build simple, inexpensive vision systems which perform useful tasks, and that the solutions to vision problems do not always involve buying better cameras or bigger computers. Furthermore, a detailed understanding of an agent's niche can bring to light special properties which can greatly simplify the computational problems facing the agent and/or its designer. Such analyses can even be recycled, allowing the insight gained in the design or analysis of one system to guide the design of another.

References

[1] P. Bellutta, G. Collini, A. Verri, and V. Torre. Navigation by tracking vanishing points. In *AAAI Spring Symposium on Robot Navigation*, pages 6–10, Stanford University, March 1989. AAAI.

[2] James E. Cutting. *Perception with an Eye for Motion.* MIT Press, 1986.

[3] Y. Goto and A. Stenz. The cmu system for mobile robot navigation. In *1987 IEEE Internation Conference on Robotics and Automation*, pages 99–105. IEEE, March 87.

[4] Ian Horswill. Characterizing adaptation by constraint. In *Proceedings of the First Annual European Conference on Artificial Life*, 1991.

[5] Ian D. Horswill. Reactive navigation for mobile robots. Master's thesis, Massachusetts Institute of Technology, June 1988.

[6] A. Kosaka and A. C. Kak. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. *To appear in Computer Vision, Graphics, and Image Processing*, 56(2), September 1992.

[7] Maja Mataric. A distributed model for mobile robot environment-learning and navigation. Technical Report 1228, Massachusetts Institute of Technology, Artificial Intelligence Lab, May 1990.

[8] Marvin Minsky. *The Society of Mind.* Simon and Schuster, New York, NY, 1986.

[9] Hans P. Moravec. Certainty grids for mobile robots. Unpublished memo.

[10] Hans P. Moravec. The stanford cart and cmu rover. Technical report, Robotics Institute, Carnegie-Mellon University, February 1983.

[11] ed. Nils J. Nilsson. Shakey the robot. Technical Report 323, SRI International, April 1984.

[12] Herbert L. Roitblat. *Introduction to Comparitive Cognition.* W. H. Freeman and Company, 1987.

[13] K. Storjohann, T. Zeilke, H. A. Mallot, and W. von Seelen. Visual obstacle detection for automatically guided vehicles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 761–766, May 1990.

[14] Shimon Ullman. Visual routines. *Cognition*, 18:97–159, 1984.