Polly: A Vision-Based Artificial Agent

Ian Horswill
MIT AI Lab
545 Technology Square
Cambridge, MA 02139
ian@ai.mit.edu

Abstract

In this paper I will describe Polly, a low cost vision-based robot that gives primitive tours. The system is very simple, robust and efficient, and runs on a hardware platform which could be duplicated for less than \$10K US. The system was built to explore how knowledge about the structure the environment can be used in a principled way to simplify both visual and motor processing. I will argue that very simple and efficient visual mechanisms can often be used to solve real problems in real (unmodified) environments in a principled manner. I will give an overview of the robot, discuss the properties of its environment, show how they can be used to simplify the design of the system, and discuss what lessons can drawn for the design of other systems.[1]

Introduction

In this paper, I will describe Polly, a simple artificial agent that uses vision to give primitive tours of the 7th floor of the MIT AI lab (see figure 1). Polly is built from minimalist machinery that is matched to its task and environment. It is an example of Agre's principle of machinery parsimony [Agre, 1988], and is intended to demonstrate that very simple visual machinery can be used to solve real tasks in real, unmodified environments in a principled manner.

Polly roams the hallways of the laboratory looking for visitors. When someone approaches the robot, it stops and introduces itself and offers the vistor a tour, asking them to answer by waving their foot around (the robot can only see the person's legs and feet). When the person waves their foot, the robot leads them around the lab, recognizing and describing places as it comes to them.

[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.

Polly is very fast and simple. Its sensing and control systems run at 15Hz, so all percepts and motor commands are updated every 66ms. It also uses very little hardware (an equivalent robot could be built for approximately \$10K), and consists of less than a thousand lines of Scheme code.[2] All computation is done on-board on a low-cost digital signal processor (a TI C30 with 64KW of ram). Polly is also among the best tested mobile robots to date, having seen hundreds of hours of service, and has a large behavioral repertoire.

Polly falls within the task-based or active approach to vision [Horswill, 1988][Aloimonos, 1990][Ballard, 1991][Ikeuchi and Herbert, 1990][Blake and Yuille, 1992]. While the work descrived here cannot prove the efficacy this approach, it does give an example of a large system which performs an interesting high-level task using these sorts of techniques.

Polly's efficiency is due to its specialization to its task and environment. Many authors have argued that simple machinery is often sufficient for performing intelligent behavior because of the special organizing structures of the environment (see, for example, [Rosenschein and Kaelbling, 1986][Brooks, 1986][Agre, 1988]). If we are to use such structures in a routine manner to engineer intelligent systems, then we must be able to isolate individual structures or properties of the environment and explain their computational rammifications. In this work, I have used the technique of stepwise transformation to draw out the relationships between a system specialized to its environment and a more general system. We look for a series of transformations, each of which conditionally preserves behavior given some constraint on the environment, that will transform the general system into the specialized system. The resulting derivation of the specialized system from the general system makes the additional assumptions made by the specialized system explicit. It also makes their computational rammifications explicit by putting them in correspondence with particular transformations which simplify particular computational subproblems. In effect, we imagine that the general system has been run through a "compiler" that has

[2]Not including device drivers and data structures.

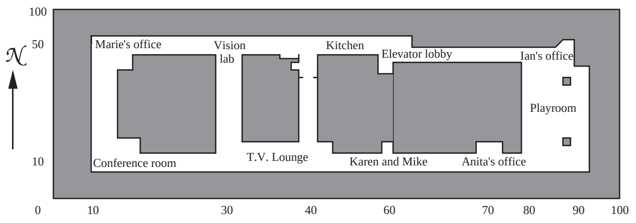

Figure 1: The approximate layout of Polly's environment (not to scale), and its coordinate system.

used declarations about the environment (constraints) to progressively optimize it until the specialized system is obtained. If we can "solve" for the declarations and associated optimizations which derive a system from a more general system, then we can reuse those optimizations in the design of future systems. Space precludes either a formal treatment of this approach or detailed analyses. See [Horswill, 1993a] or [Horswill, 1993b] for detailed discussions.

In next section, I will discuss some of the useful properties of Polly's environment and allude to the transformations which they allow. Then I will discuss the actual design of the system, including abbreviated forms of the constraint derivations for parts of the visual system. Finally, I will discusses the performance and failure modes of the system in some detail and close with conclusions.

Computational properties of office environments

Office buildings are actively structured to make navigation easier [Passini, 1984]. The fact that they are structured as open spaces connected by networks of corridors means that much of the navigation problem can be solved by corridor following. In particular, we can reduce the problem of finding paths in space to finding paths in the graph of corridors. The AI lab is even simpler because the corridor graph has a grid structure, and so we can attach coordinates to the verticies of the graph and use difference reduction to get from one pair of coordinates to another.

Determining one's position in the grid is also made easier by special properties of office environments: the lighting of offices is generally controlled; the very narrowness of their corridors constrains the possible viewpoints from which an agent can see a landmark within the corridors. I will refer to this latter property as the *constant viewpoint constraint*: that the configuration space of the robot is restricted so that a landmark can only be viewed from a small number of directions. These properties make the recognition of landmarks in a corridor an easier problem than the fully general recognition problem. Thus very simple mechanisms often suffice.

Another useful property of office buildings is that they are generally carpeted and their carpets tend to

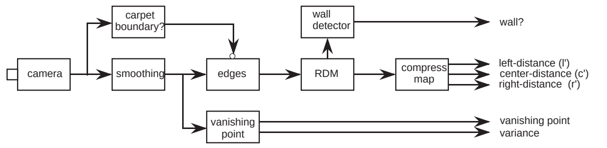

Figure 2: Portion of visual system devoted to navigation.

be either regularly textured or not textured at all. The predictability of the texturing of the carpet means that any region of the image which isn't textured like the carpet is likely an object resting on the ground (or an object resting on an object resting on the ground). Thus *obstacle detection* can be reduced to *carpet detection*, which may be a simpler problem admitting simpler solutions. In the case of the MIT AI lab, the carpet has no texture and so a texture dectector suffices for finding obstacles. We will refer to this as the *background-texture constraint* (see [Horswill, 1993a] for a more detailed discussion).

Finally, office buildings have the useful property that they have flat floors and so objects which are farther away will appear higher in the image, provided that the objects rest on the floor. This provides a very simple way of determining the rough depth of such an object. We will refer to this as the *ground-plane constraint*: that all obstacles rest on a flat floor (see [Horswill, 1993a]).

Architecture

Conceptually, Polly consists of a set of parallel processes connected with fixed links (see [Brooks, 1986][Agre and Chapman, 1987][Rosenschein and Kaelbling, 1986] for examples of this type of methodology)). The actual implementation is a set of Scheme procedures, roughly one per process, with variables used to simulate wires. On each clock tick (66ms), the robot grabs a new image, runs each process in sequence to recompute all visual system outputs, and computes a new motor command which is fed to the base computer.

Physically, the robot consists of an RWI B12 robot base which houses the motors and motor control logic, a voice synthesizer, a front panel for control, a serial port for downloading, a TMS320C30-based DSP board (the main computer), a frame grabber, and a microprocessor for servicing peripherals. All computation is done on board.

Visual system

The visual system processes a 64×48 image every 66ms and generates a large number "percepts" from it (see figure 3) which are updated continuously. Most of these are related to navigation, although some are devoted to person detection or sanity checking of the image. Because of space limitations, we will restrict ourselves to the major parts of the navigation section.

`open-left?`	`open-region?`	`person-direction`
`open-right?`	`blind?`	`wall-ahead?`
`blocked?`	`light-floor?`	`wall-far-ahead?`
`left-turn?`	`dark-floor?`	`vanishing-point`
`right-turn?`	`person-ahead?`	`farthest-direction`

Figure 3: Partial list of percepts generated by the visual system.

Constraint	Computational problem
Ground plane	Depth perception
Background-texture	Figure/ground separation
Long corridor edges	Vanishing point
Strong corridor edges	Edge detection
Known camera tilt	Vanishing point
Uniform NC intersctns	Vanishing point

Figure 4: Constraints assumed by the visual system and the problems they helped to simplify. Note that "known camera tilt" is more a constraint on the agent, than on the habitat.

The central pipeline in figure 2 ("smoothing" ... "compress map") computes depth information. The major representation used here is a *radial depth map*, that is, a map from direction to distance, similar to the output of a sonar ring. Computing depth is a notoriously difficult problem. The problem is greatly simplified for Polly by the use of domain knowledge. By the ground plane constraint, we can use height in the image plane as a measure of distance. Thus the system[3]

$$\begin{tikzcd} {} \arrow[r, Rightarrow] & \boxed{\text{F/G}} \arrow[r] & \boxed{RDM} \arrow[r] & {} \end{tikzcd}$$

can be substituted for any system which computes a radial depth map, where F/G is any system which does figure/ground separation (labeling of each pixel as figure or background), and *RDM* is a transformation from a bitmap to a vector defined by

$$RDM(x) = \min\{y | \text{the point } (x, y) \text{ isn't floor}\}$$

The effect of this is to reduce the problem of depth recovery to the figure-ground problem. The figure-ground problem is, if anything, more difficult than the depth-recovery problem in the general case so one might expect this to be a bad move. However, by the background-texture constraint, we can use any operator which responds to the presence of texture. Polly presently uses a simple edge detector (thresholded magnitude of the intensity gradient):

$$\begin{tikzcd} {} \arrow[r, Rightarrow] & \boxed{|\nabla I|} \arrow[r] & \boxed{RDM} \arrow[r] & {} \end{tikzcd}$$

The visual system then compresses the depth map into three values, `left-distance`, `right-distance`, and `center-distance`, which give the closest distance on the left side of the image, right side, and the center third, respectively. Other values are then derived from these. For example, `open-left?` and `open-right?` are true when the corresponding distance is over threshold. `left-turn?` and `right-turn?` are true when the depth map is open on the correct side and the robot is aligned with the corridor.

The visual system also generates the vanishing point of the corridor. Bellutta *et al* [Bellutta *et al.*, 1989] describe a system which extracts vanishing points by running an edge finder, extracting straight line segments, and performing 2D clustering on the pairwise intersections of the edge segments. We can represent it schematically as:

$$\begin{tikzcd} {} \arrow[r, Rightarrow] & \boxed{\text{edges}} \arrow[r] & \boxed{\text{lines}} \arrow[r] & \boxed{\text{intersect}} \arrow[r] & \boxed{\text{cluster}} \arrow[r] & {} \end{tikzcd}$$

We can simplify the system by making stronger assumptions about the environment. We can remove the step of grouping edge pixels into segments by treating each edge pixel as its own tiny segment. This will weight longer lines more strongly, so the lines of the corridor must dominate the scene for this to work properly. If the edges are strong, then a simple edge detector will suffice. Polly uses a gradient threshold detector. If the tilt-angle of the camera is held constant by the camera mount, then the vanishing point will always have the same y coordinate, so we can reduce the clustering to a 1D problem. Finally, if we assume that the positions and orientations of the non-corridor edges are uniformly distributed, then we can replace the clustering operation, which looks for modes, the mean. After all these optimizations, we have the following system:

$$\begin{tikzcd} {} \arrow[r, Rightarrow] & \boxed{|\nabla|} \arrow[r] & \boxed{y \text{ intersect}} \arrow[r] & \boxed{\bar{x}} \arrow[r] & {} \end{tikzcd}$$

The system first computes the gradient threshold edges, then intersects the tangent line of each edge pixel with the horizontal line in which the vanishing point is known to lie, then computes the mean of the x coordinate of the intersections. The variance is also reported as a confidence measure.

The constraints assumed by these systems are summarized in figure 4. The discussion here has been necessarily brief. For a more detailed derivation, see [Horswill, 1993a].

Low-level navigation

The robot's motion is controlled by three parallel systems. The distance control system drives forward with a velocity of $\alpha($`center-distance`$-d_{stop})$, where d_{stop} is the threshold distance for braking and α is a gain parameter. Note that it will back up if it overshoots or if it is aggressively approached. The corridor follower drives the turning motor so as to keep the vanishing point in the middle of the screen and keep `left-distance` and

[3]Here the $\Rightarrow$ symbol is used to denote input from the sensors, while $\rightarrow$ denotes signals moving within the system.

Kitchen	
Position	(50, 40)
Direction	west
Veer	0
Image	...

Elevator lobby	
Position	(60, 40)
Direction	east
Veer	45
Features	right, wall

Figure 5: Example place frames.

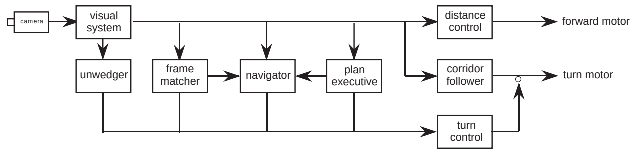

Figure 6: Architecture of the complete navigation system.

`right-distance` equal. The corridor follower switches into wall-following mode (keeping the wall at a constant distance) when only sees one wall. Finally, the turn unit drives the base in open-loop turns when instructed to by higher-level systems. The corridor follower is overridden during open-loop turns. During large turns, the distance control system inhibits forward motion so as to avoid suddenly turning into a wall. For more detailed of the low-level navigation system discussion, see [Horswill, 1993b].

Place recognition

Polly generally runs in the corridors and open spaces of the 7th floor of the AI lab at MIT. It keeps track of its position by recognizing landmarks and larger-scale "districts," which are given to it in advance. The lab, and some of its landmarks are shown in figure 1.

The corridors of the lab provide a great deal of natural constraint on the recognition of landmarks. Since corridors run in only two perpendicular directions, which we will arbitrarily designate as north-south (ns) and east-west (ew), they form natural coordinate axes for representing position. The robot's base provides rough rotational odometry which is good enough for the robot to distinguish which of four directions it is moving in, and so, in what type of corridor it must be.

Each distinctive place in the lab is identified by a pair of *qualitative coordinates*, shown in the figure. These coordinates are not metrically accurate, but they do preserve the ordering of places along each of the axes. Information about places is stored in an associative memory which is exhaustively searched on every clock tick (66ms). The memory consists of a set of frame-like structures, one per possible view of each landmark (see figure 5). Each frame gives the expected appearance of a place from a particular direction (north/south/east/west). Frames contain a place name, qualitative coordinates, and a direction and some specification of the landmark's appearance: either a 16×12 grey-scale image or a set of qualitative features (left-turn, right-turn, wall, dark-floor, light-floor). No explicit connectivity information is represented. Frames can also be tagged with a speech to give during a tour or an open-loop turn to perform.

While at first glance, this may seem to be an inefficient mechanism, it is in fact quite compact. The complete set of frames for the 7th floor requires approximately 1KByte of storage. The system can scan all the frames and find the best match at 15Hz using only a fraction of the CPU.

The system can also recognize large-scale "districts" and correct its position estimate even if it cannot determine exactly where it is. There is evidence that humans use such information (see [Lynch, 1960]). The robot presently recognizes the two long east/west corridors as districts. For example, when the robot is driving west and sees a left turn, it can only be in the southern ew corridor, so its y coordinate must be 10, regardless of its x position. This allows to robot to quickly recover from getting lost. At present, the recognition of districts is implemented as a separate computation, but I intend to fold it into the frame system.

High-level navigation

By default, the corridor follower is in control of the robot at all times. The corridor follower will always attempt to go forward and avoid obstacles unless it is overridden. Higher-level navigation is implemented by a set of independent processes which are parasitic upon the corridor follower. These processes control the robot by enabling or disabling motion, and by forcing open-loop turns. The navigator unit chooses corridors by performing difference reduction of the (qualitative) goal coordinates and the present coordinates. When there is a positive error in the y coordinate, it will attempt to drive south, or north for negative error, and so on. This technique has the advantages of being very simple to implement and very tolerant of place recognition errors. If a landmark is missed, the system need not replan its path. When the next landmark in the corridor is noticed, it will automatically return to the missed landmark. The "unwedger" unit forces the robot to turn when the corridor follower is unable to move for a long period of time (2 seconds). Finally, a set of action-sequencers (roughly plan executives for hand-written plans) are used to implement tour giving and operations such as docking. The sequencers execute a language of high level commands such as "go to place" which are implemented by sending commands to lower-level modules such as the navigator.

Performance

At present the system runs at 15Hz, but is I/O bound. The navigation system can safely run the robot at up to 1.5m/s, however, the base becomes unstable at that speed. The system is very robust, particularly the low-level locomotion system, which has seen hundreds of hours of service. The place recognition and navigation systems are newer and so less well tested. The failure modes of the component systems are summarized below.

Low-level navigation

All locomotion problems were obstacle detection problems. The corridor follower 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. The present system also has no memory and so cannot brake for an object unless it is actually in its field of view. 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. Finally, several floors have multiple carpets, each with a different color. The boundaries between these carpets can thus be mistaken for obstacles. This problem was dealt with by explicitly recognizing the boundary when directly approaching it and informing the edge detector to ignore horizontal lines.

The system has also been tested recently at Brown university. There the robot had serious problems with light levels, but performed well where there was sufficient light for the camera. In some areas the robot had problems because the boundary between the floor and the walls was too weak to be picked up by the edge detector. We would expect any vision system to have problems in these cases however.

Place recognition

Place recognition is the weakest part of the system. While recognition by matching images is quite general, it is fragile. It is particularly sensitive to changes in the world. If a chair is in view when a landmark template is photographed, then it must continue to be in view, and in the same place and orientation, forever. If the chair moves, then the landmark becomes unrecognizable until a new view is taken. Another problem is that the robot's camera is pointed at the floor and there isn't very much interesting to be seen there. For these reasons, place recognition is restricted corridor intersections represented by feature frames, since they are more stable over time. The one exception is the kitchen which is recognized using images. In ten trials, the robot recognized the kitchen eight times while going west, and ten times while going east. Westward recognition of the kitchen fails completely when the water bottles in the kitchen doorway are moved however.

Both methods consistently miss landmarks when there is a person standing the the way. This often leads it to miss a landmark immediately after picking up a visitor. They also fail if the robot is in the process of readjusting its course after driving around an obstacle or if the corridor is very wide and has a large amount of junk in it. Both these conditions cause the constant-viewpoint constraint to fail. The former can sometimes cause the robot to hallucinate a turn because one of the walls is invisible, although this is rare.

Recognition of districts is very reliable, although it can sometimes become confused if the robot is driven in a cluttered open space rather than a corridor.

High-level navigation

High-level navigation performance is determined by the accuracy of place recognition. In general, the system works flawlessly unless the robot gets lost. For example, the robot has often run laps (implemented by alternately giving opposite corners as goals to the navigator) for over an hour without any navigation faults. When the robot gets lost, the navigator will generally overshoot and turn around. If the robot gets severely lost, the navigator will flail around until the place recognition system gets reoriented. The worst failure mode is when the place recognition system thinks that it is east of its goal when it is actually at the western edge of the building. In this case, the navigator unit and the unwedger fight each other, making opposite course corrections. The place recognition system should probably be modified to notice that it is lost in such situations so that the navigator will stop making course corrections until the place recognition system relocks. This has not yet been implemented however.

Getting lost is a more serious problem for the action sequencers, since they are equivalent to plans but there is no mechanism for replanning which a plan step fails. This can be mitigated by using the navigator to execute individual plan steps, which amounts to shifting responsibility from plan-time to run-time.

Conclusions

Many vision-based mobile robots have been developed in the past (see for example [Kosaka and Kak, 1992][Kriegman *et al.*, 87] [Crisman, 1992][Turk *et al.*, 1987]). The unusual aspects of Polly are its relatively large behavioral repertoire, simple design, and principled use of special properties of its environment.

Polly's efficiency and reliability are due to a number of factors. Specialization to a task allows the robot to compute only the information it needs. Specialization to the environment allows the robot to substitute simple computations for more expensive ones. The use of multiple strategies in parallel reduces the likelihood of catastrophic failure (see [Horswill and Brooks, 1988]). Thus if the vanishing point computation generates bad data, the depth-balancing strategy will compensate for it and the distance control system will prevent collisions until the vanishing point is corrected. Finally, the speed of its perception/control allows it to rapidly recover from errors. This relaxes the need for perfect perception and allows simpler perceptual and control strategies to be used.

Scalability is a major worry for all approaches to AI. We don't know whether an approach will scale until we try to scale it and so the field largely runs on existence proofs. Polly is an existence proof that a robust system with a large behavioral repertoire can be built using simple components which are specialized to their task and environment, but it does not show how far we can extend the approach.

One of the benefits of making constraints explicit and putting them in correspondence with transformations is that it gives us some degree of leverage in generalizing our designs. Although space precluded a detailed analysis of Polly's systems, we can see from the brief analysis of the low level navigation system that the role of the background texture constraint was to simplify the figure ground problem by allowing the substitution of a edge detector. This tells us several useful things. First, any linear filter restricted to the right band will do (see [Horswill, 1993a]). Second, if the environment does not satisfy the BTC, then any other transformation which simplifies figure ground will also do. We can even use multiple figure/ground systems and switch between them depending on the properties of the environment. Another possibility is two implement both the general system and a specialized system and switch at the behavioral level. This effectively moves the optimization from compile-time to run-time and makes the specialized system a sort of a hardware accelerator on a par with a cache memory.

Thus specialized systems need not simply be hacks. We can learn things from the design of one specialized system which are applicable to the designs of other systems, even traditional systems.

References

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, 1987.

[Agre, 1988] Philip E. Agre. The dynamic structure of everyday life. Technical Report 1085, October 1988.

[Aloimonos, 1990] John Aloimonos. Purposive and qualitative active vision. In *DARPA Image Understanding Workshop*, 1990.

[Ballard, 1991] Dana H. Ballard. Animate vision. *Artificial Intelligence*, 48(1):57–86, 1991.

[Bellutta *et al.*, 1989] 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.

[Blake and Yuille, 1992] Andrew Blake and Alan Yuille, editors. *Active Vision*. MIT Press, Cambridge, MA, 1992.

[Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automoation*, 2(1):14–23, March 1986.

[Crisman, 1992] Jill D. Crisman. *Color Region Tracking for Vehicle Guidance*, chapter 7. In Blake and Yuille [1992], 1992.

[Horswill and Brooks, 1988] Ian Horswill and Rodney Brooks. Situated vision in a dynamic environment: Chasing objects. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, August 1988.

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

[Horswill, 1993a] Ian Horswill. Analysis of adaptation and environment. In submission, 1993.

[Horswill, 1993b] Ian Horswill. *Specialization of perceptual processes.* PhD thesis, Massachusetts Institute of Technology, Cambridge, 1993. forthcoming.

[Ikeuchi and Herbert, 1990] Katsushi Ikeuchi and Martial Herbert. Task oriented vision. In *DARPA Image Understanding Workshop*, 1990.

[Kosaka and Kak, 1992] A. Kosaka and A. C. Kak. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. *Computer Vision, Graphics, and Image Processing*, 56(3), September 1992.

[Kriegman *et al.*, 87] David J. Kriegman, Ernst Triendl, and Tomas O. Binford. A mobile robot: Sensing, planning and locomotion. In *1987 IEEE Internation Conference on Robotics and Automation*, pages 402–408. IEEE, March 87.

[Lynch, 1960] Kevin Lynch. *The Image of the City.* MIT Press, 1960.

[Passini, 1984] Romedi Passini. *Wayfinding in Architecture*, volume 4 of *Environmental Design Series.* Van Norstrand Reinhold, New York, 1984.

[Rosenschein and Kaelbling, 1986] Stanley J. Rosenschein and Leslie Pack Kaelbling. The synthesis of machines with provable epistemic properties. In Joseph Halpern, editor, *Proc. Conf. on Theoretical Aspects of Reasoning about Knowledge*, pages 83–98. Morgan Kaufmann, 1986.

[Turk *et al.*, 1987] Matthew A. Turk, David G. Morgenthaler, Keith Gremban, and Martin Marra. Video road following for the autonomous land vehicle. In *1987 IEEE Internation Conference on Robotics and Automation*, pages 273–280. IEEE, March 1987.