Heuristics for Projecting a Sensorimotor Mapping*

Karl F. MacDorman

Department of Systems and Human Science
Graduate School of Engineering Science, Osaka University
`http://robotics.me.es.osaka-u.ac.jp/~kfm`

Abstract: *Learning a sensorimotor model and projecting that model spatially and temporally are fundamental steps toward a robot that can create its own future. They enable a robot to predict the consequences of its actions and create complex sensorimotor plans to obtain goals in new environments. This paper proposes heuristics for projecting a sensorimotor model — to plan paths to potential affordances in a cluttered environment — and evaluates them in simulation. The robot represents the sensorimotor effects of chains of actions in a phase space, which it quantizes into regions, called cells, to avoid recomputing subpaths between regions. In forward search, the robot attempts actions "mentally," beginning from its starting position. It repeats this process among those cells it has examined according to their priority. We evaluate an iterative algorithm that optimizes a poor but quickly computed path and a noniterative algorithm that gives the highest priority to examining cells that are closest to the goal in distance and closest to the start in time. In our study the latter resulted in paths that are only 3.3 percent longer on average than the optimal method but take less than a seventh the time to compute. Finally, we discuss issues concerning the use of the algorithm in robots with onboard cameras.*

1 Background: Symbol Grounding

In 1980 Alan Newell clarified a central assumption in cognitive science, that the brain is a symbol system [16, 1]. The assumption seems plausible because the syntax of a representation can encode its role in inference. Although this has been known since Aristotle discovered the syllogism, its full impact came with the advent of computers. This is because computers automate the process of manipulating representations according to their syntax. Humans are able to ground deductive systems in their practical knowledge of the world, but this leaves a dilemma for machines. In robotics, a hybrid approach has developed that involves interfacing a symbol system, used for abstract reasoning, with a low-level system that recognizes and manipulates objects [11].

The trouble with this approach is that the symbol manipulation part functions according to internal relations only: the rules and representations of the symbol system. Yet evidence is mounting in various fields[1] that mind-to-world relations need to constrain even abstract reasoning [2, 5, 6, 9]. These relations are fundamentally sensorimotor, and they can change unexpectedly as either body or environment change. They determine an organism's opportunities for interaction — what J. J. Gibson called its affordances [3]

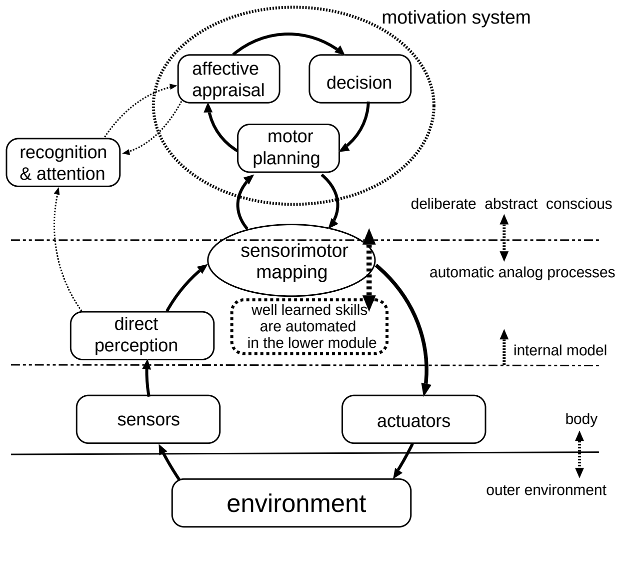

Figure 1: The design for Ψro (pronounced *sa'i·ro*, from the Greek *psy*che + *ro*bot), a robot that exploits affordances. This paper focuses on the issue of planning a sequence of actions to obtain a goal. We refer to this as the spatiotemporal projection of a learned sensorimotor mapping.

*A grant from CREST of JST helps support this research. Java is a trademark of Sun Microsystems. Lego is a trademark of the Lego Group. Pentium is a trademark of Intel Corporation.

[1]For example, linguistics [2], psychology [4, 5, 19], artificial intelligence and robotics [6, 9].

— and the kinds of invariance available to recognize them.

If sensorimotor relations must constrain abstract reasoning and if these relations are susceptible to bodily and environmental change, this leads naturally to the approach taken here. A mobile robot learns a sensorimotor model from experience so that it can make predictions about the consequences of its actions (§3.1). This involves learning to recognize affordances (§3.2) and the actions needed to exploit them. For a fairly low-level task like robot navigation, this may work out to knowing *what* is *where*. But notions of object and distance must be discovered by the robot itself based on how its motor signals transform both internal indicators of well-being (e.g., battery level) and the information that external objects project on to its camera image planes and other sensors [8]. Once a robot has learned to recognize affordances, it can project its sensorimotor model into the future to obtain goals (see Figure 1 and Figure 7 for a photo of Ψro the robot). This paper examines heuristics for projecting such a model.

Although the proposed design is too simple to explain the full range of human behavior, it is more complex than, for example, reinforcement learning. In reinforcement learning, action selection is based on the (expected future) reward in a perceived situation. Normally, the robot must be in the same situation many times to estimate this value. Thus, it is useful to have additional forms of representation so that new situations can be dealt with based on knowledge abstracted from past experience. This includes *(1)* sensorimotor maps that may be used to predict how motor signals transform sensory projections from the surfaces of individual objects; *(2)* representations of the sensorimotor invariance underlying various categories of affordances; and *(3)* representations of how interactions with different affordance categories tend to transform the robot's internal variables (e.g., battery level) [8, 9]. To generate interesting behavior, we may wish to set up the robot with variables that are behaviorally analogous to physiological and affective states in animals.[2]

A sensorimotor model offers predictions concerning how a robot's actions transform sensory information and recognized affordances. The next section introduces several algorithms for projecting a sensorimotor model in space and time to attain goals. It presents two heuristics for efficiently finding a sequence of motor signals that produce actions leading to a goal. The first heuristic involves the iterative optimization of a quickly calculated but suboptimal path, thus providing a locally optimal solution. The second heuristic involves balancing a search criterion that favors computational efficiency with one that favors global optimality. While the results are not guaranteed to be optimal, they are usually very good.

We explain the projection heuristics with the aid of a Java™ applet. The applet, a tutorial, and source code are available at http://robotics.me.es.osaka-u.ac.jp/∼kfm/tutorials. The applet assumes that information is available from an overhead camera, thus largely sidestepping issues concerning occlusion. It does not address issues concerning robot-mounted cameras, or how the robot may learn a sensorimotor model and determine an appropriate goal. These issues will be dealt with in later sections.

[2]It is not assumed that these variables would correspond to any real emotions or conscious states.

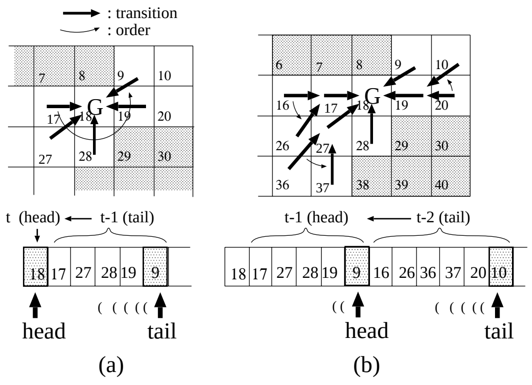

Figure 2: In previous work we developed a method of finding the shortest path to a goal in a cluttered environment, where a mobile robot's speed, orientation, and x, y position are described by a point in a phase space. This technique is closely related to dynamic programming because, once the robot has quantized the phase space, it finds the shortest path by calculating the optimal solution to all subproblems. In this example the robot reasons backwards: For any quantized motor signals (e.g., accelerate, turn sharply left), it calculates one path to the goal from each accessible cell that can reach the goal in one time step *(a)*, storing the results in a queue. The robot then calculates all paths to each of these cells in turn from *unexamined* cells that are one time step away *(b)*. (These cells are two time steps from the goal.) This process is repeated until the robot reaches the cell that contains its current state in the phase space.

2 Projection Heuristics

Once a robot has learned a sensorimotor model and can recognize a goal, a basic approach is to apply means-ends analysis [15]: the robot "mentally" takes whatever action will most reduce its physical distance to the goal.[3] This approach breaks down if there are obstacles because, instead of taking the shortest path to the goal, the robot will hug closely to the obstacles, and it will become stuck if they form a concave barrier on its path to the goal. Backtracking from this arrangement of obstacles is impossible because

[3]Distance here is not physical distance but distance in a phase space whose dimensions are determined by the robot's sensors and may include other nondistance metrics like speed and relative orientation.

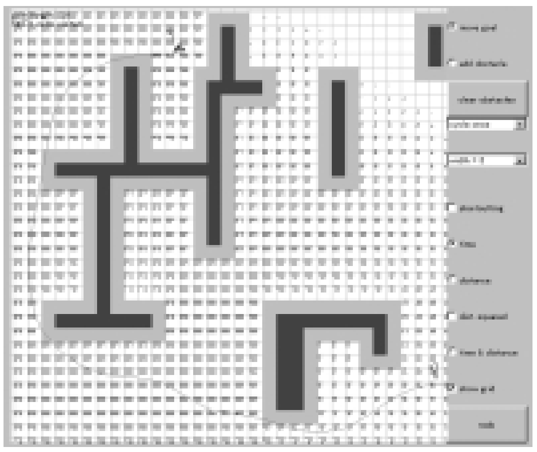

Figure 3: This simulation illustrates an application of the spatiotemporal projection algorithm to planning a mobile robot's path to a goal. In the simulation, planning begins from the goal and works back to the robot's initial state. A priority queue is implemented as a heap, thus allowing different criteria for the search priority. In this figure, the search priority is the *time* it takes to reach the goal, so the results are similar to those obtained by using a simple queue. Obstacles appear in black. Since the robot's state is represented by a point, regions around obstacles, which appear in gray, are declared off limits to prevent collision with obstacles owing to the robot's width. The white flag marks the robot's starting position and the gray flag marks its goal. The grid show the subdivision of the phase space along the x, y axes. The subdivision according to speed and orientation is given indirectly by the appearance of black dots within each square. These dots only occur when a cell for a given range of positions, speeds, and orientations has been examined during search. The vertical position of the dot within the square denotes one of three speed ranges and the horizontal position denotes one of 36 orientation ranges. 54,571 cells were visited to calculate an optimal path of length 39.

priority is given to reducing the distance to the goal, but to reach the goal the robot must at first move away from it to clear the obstacles. At the other extreme, there is exhaustive search. This will find an optimal path to the goal for a given coarseness in the quantization of the motor signals and time interval. However, the approach is only practical for very small problems because the search grows exponentially (a^l) with the length of the path (l) and the number of possible actions at any given moment (a). In other words, exhaustive search suffers from the curse of dimensionality.

The approach taken here is quite general, and it may be applied to other problems, besides robot navigation, that involve sensorimotor planning — for example, performing a gymnastic movement like the kip [14]. The robot's phase space (i.e., its space of possible states, which in this simple simulation is determined by its position, speed and orientation) is quantized into hypercubes called cells. The robot begins planning from its starting cell and "mentally" tries actions that may lead to other cells (see Listing 1). In the simulation we took the alternative approach of working back from the goal. The robot begins by calculating all paths originating in distinct cells that reach the goal in one time step. These cells are placed in a priority queue. If the queue were simply a linked list and the robot tried actions from the state in the cell at the head of the queue and added cells it had examined to the tail, the projection algorithm would be closely related to the application of dynamic programming[4] to continuous phase spaces [12, 7, 8] (see Figure 2).[5] In this simulation, however, the priority queue is implemented as a heap [20] so that an arbitrary priority function may be used for guiding search (see Figure 3).[6]

Each cell records the time required to reach it, the cell from which it was reached, and the action taken at that cell. It also records the (floating point) location in the phase space of the point within the cell, so that errors in prediction do not accumulate owing to the quantization. Cells that have already been visited are never revisited. It is in this sense that the projection algorithm exploits the power of dynamic programming, which lies in the fact that optimal partial solutions are stored away and looked up when needed to avoid recomputation. Exhaustive search, by contrast, may compute many paths between nearby regions of a phase space.

Since a continuous phase space is being quantized, we can only discuss optimality at a given coarseness of quantization. In the limiting case, as the quantization becomes ever more fine, the algorithm approaches the theoretical optimum. But these marginal gains come at the expense of exponentially more computation.

As an alternative to forward search, we mentioned that in the simulation the robot takes the somewhat unnatural approach of "mentally" attempting actions in reverse, starting from the goal. Simultaneously calculating a forward and backward solution and stopping when the two paths meet also provides an optimal solution. Since there are usually more paths to consider further from the start or goal, this technique can greatly reduce the number of cells visited by eliminating the need to consider these paths. (The usefulness of this technique is less certain if the search is guided by a priority other than time-to-goal or time-from-start because the two paths may only meet near the starting state or goal instead of meeting midway.)

[4]Dynamic programming is a method applied to discrete problem where the only method of finding the optimal solution involves solving all unique subproblems [20].

[5]I used this technique in controlling a remote-controlled car [7]. The best solution, a three-point turn, required the car to move at first away from the goal, even though there are no obstacles in its line of sight to the goal.

[6]An insertion into a heap requires on the order of $\log_2 H$ operations where H is the size of the heap. Removing an element from the top of the heap is also order $\log_2 H$.

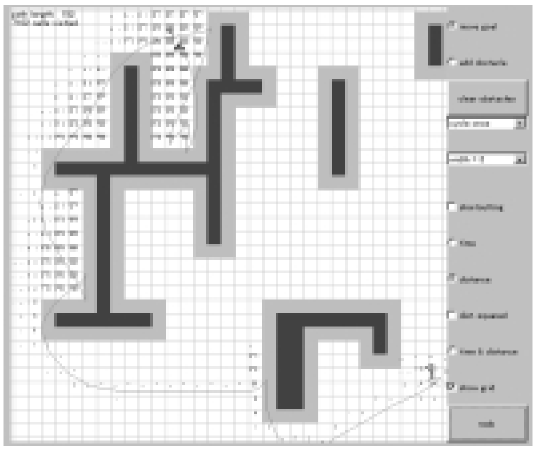

Figure 4: In this simulation each cell's Euclidean *distance* to the robot's starting position determines its priority during search. Since cells are stored in the heap according to their priority, the cell at the top of the heap is the closest among them to the start. All unexamined cells that can reach this cell in one time step are stored in the heap, ordered according to their distance from the start. The path is circuitous because the robot tries to work its way from the goal back to its starting state but encounters a convex obstacle. Unlike with simple hill climbing algorithms, the robot can escape from this local minimum because the algorithm will not revisit the same cell. 7102 cells were examined to calculate a path of length 152.

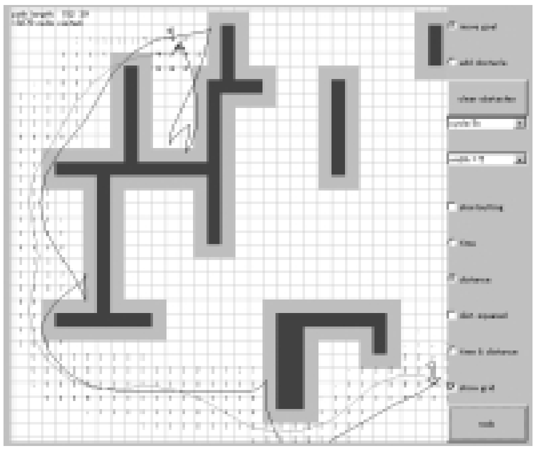

Figure 5: Using time as the search priority resulted in the examination of 54571 cells and an optimal path length of 39 time steps. Using distance resulted in the examination of 7102 cells (13%) but a path length of 152 cells — almost four times the optimum. However, we can improve on this bad path by marking all cells within a 5x5 region around it and using time as a priority to find an optimal path within this marked region only. Although this method is not guaranteed to find a globally optimal solution, it does find one in this example. The final path length (shown in gray) is 39 time steps; however, the algorithm has only examined 18279 cells in total — one third the number examined when just using time as the priority.

Although dynamic programming may postpone the curse of dimensionality, finding the optimal solution to *all* subproblems at a given coarseness of quantization may still be too costly for real-time application. Furthermore, an optimal solution may not be needed. This is why it is useful to organize cells in the heap according to a priority function other than *time* (i.e., time to the goal for forward searches or time to the robot's starting state for backward searches). This means applying some heuristic — a method that is not guaranteed to give an optimal solution but usually gives a reasonable solution. Using *distance* as the priority function enables the rapid computation of a path (see Figure 4). (In a forward search, this is the distance to the goal. In a backward search, it is the distance from the start.) But the path suffers from the same obstacle hugging found in means-ends solutions (except that the robot will not get stuck because, once visited, a cell will not be revisited unless it can be reached in less time).

An iterative and noniterative heuristic. There are two good solutions to this problem. One involves improving on a poor path (e.g., one computed with distance as the priority, as in Figure 4) by (repeatedly) marking the cells in the path's vicinity and then applying the projection algorithm using the time priority in the marked region only [7, 8]. This method leads to a locally optimal solution in the sense that it is the best obtainable path given an initial path, but a better path may be obtainable from a different initial path (see Figure 5).

Another good method based on the well-known $A^\star$ algorithm involves using a weighted *time and distance* priority (see Figure 6 and Listing 1). In a forward search, the first cells to be visited in the heap are those that are closest to the start in terms of the time required to reach them and closest to the goal in terms of Cartesian distance in the robot's sensory subspace. (In a backward search, the first cells to be visited are those that are closest to the goal in time and closest to the start in distance.) If distance is weighted more heavily in setting the priority, the path is longer but calculated more quickly. As we weight time more heavily than distance, the path approaches the optimum but at the cost of more and more computation.

Shortcutting. To improve the search heuristics, *short cutting* was introduced: Shortcutting allows a cell to be reexamined and updated if its a new path to the cell reaches it in less time than the previously recorded path. Shortcutting never occurs if the priority is only time-to-goal (or time-to-start, for forward searches).

Unfortunately, shortcutting complicates the handling of the heap. What happens if the revisited cell is simply inserted again into the heap according to its new priority without removing the old pointer from

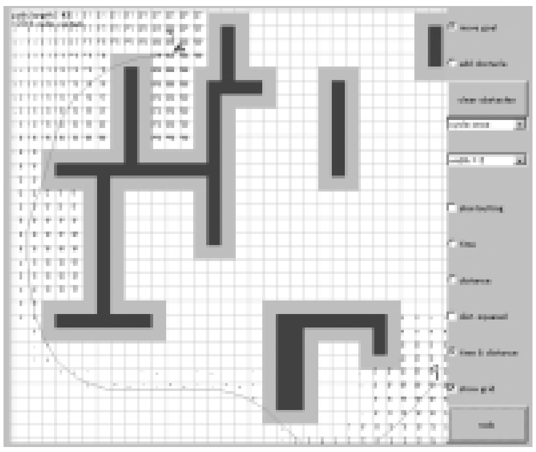

Figure 6: This simulation set the search priority according to time and distance. (The time to reach the goal is simply subtracted from the distance to the starting state; a lower value has a higher priority.) The path length is 43, which is almost as short as the optimal 39 (91%) and the algorithm only examines 10756 cells, or less than a fifth the number using time alone as the priority.

the heap's interior? Depending on certain implementation details, the old pointer may either be in the wrong position in the heap given the priority of the cell it points at or it may in the right position but pointing at a cell that is obsolete.

To remove the old pointer efficiently, each cell in the heap must record its position therein. Furthermore, any cells that are reached from the removed cell later in the search are also obsolete. They should also be removed or at least blocked from future consideration. This requires many extra pointers, since a removed cell must point at all its children so that they too can be found and removed.

Although shortcutting often resulted in shorter paths when distance was used as the search priority, this came the cost of much more computation. Other heuristics generally offered shorter paths with less computation than shortcutting.

2.1 Simulation Results

The table below lists the average path length, average computation time in milliseconds, and the average number of cells examined for five different search priorities based on 10 trials with 10 different mazes or arrangements of obstacles.

priority	*ave. len.*	*ave. time*	*ave. cells*
time	26.6	5197 ms	59039
distance	78.7	127 ms	4699
iterative	28.7	1603 ms	14325
time & dist.	27.5	717 ms	7512
30% – 70%	32.1	518 ms	5923

```
procedure fwd_project (start, goal, heap, space) {
  curr ← start;
  do {
    $\forall \vec{a} \in \{\vec{a}_1, \ldots \vec{a}_n\}$ {
      next ← predict(curr, $\vec{a}$);
      if (¬out-of-bounds-or-revisiting(next)) {
        next.$\vec{a}$ ← $\vec{a}$;
        next.time ← time-from-start;
        next.priority ← time-from-start + dist.-to-goal;
        next.origin ← curr;
        space[quantize(next)] ← next;
        if (reached(goal)) finished;
        insert(next, heap);
      }
    }
    curr ← pop(heap);
  } while (¬empty(heap));
}
```

Listing 1: The forward projection algorithm. The main difference between the various simulations is the criterion used for calculating the priority. The current cell is set to the starting cell. The algorithm tries in turn every combination of motor signals $\vec{a}$. It estimates the likely next cell from the robot's experience given the current cell and motor signals. It does not consider the cell further if it is out of bounds or occupied by an obstacle, or if it has already been examined. The motor signals and time required to reach the next cell are recorded as are the priority and a pointer to the current cell. The next cell is then indexed by the phase space and heap data structure, and if the goal has been reached, the procedure exits. Once all cells that are reachable from the current cell in one time step have been examined, the cell with highest priority in the heap is removed, and that cell becomes the current cell.

The simulation results show that a search priority based on time-to-goal and distance-from-start for backward searches results in a good balance between computational efficiency and short path length. The path length is only 3.3% longer than when using the optimal method, but the computation time is just 13.8% of the computation time using the optimal method; and the number of cells examined is just 12.7%.

The computing time may be shaved further, at the expense of a slightly longer path, by giving a 30% weighting to the time criteria and a 70% weighting to distance rather than weighting them equally. Diminishing returns set in for more than a 70% weighting for distance. A 200 MHz Pentium™ PC was used in the simulations. The reader may experiment with the simulation at http://robotics.me.es.osaka-u.ac.jp/~kfm/tutorials.

2.2 A Note on RoboCup

In a robot competition Koji Tatani applied this algorithm to controlling a manipulator.[7] A member of the audience was told to set up a maze with a starting position and goal by attaching Lego® bricks to a board. An overhead camera then captured an image of the board, which was segmented by color and fed to the projection algorithm. The algorithm then planned a route from start to finish. The manipulator followed the route by pulling a toy car across the board.

In like manner, the algorithm can be easily adapted to soccer playing robots at RoboCup.[8] Most of the teams in the medium-sized league use an overhead camera and color image segmentation. It is easy to find the location of the ball and goal based on their color. So a simple control program would find a point in the phase space that places the robot on the other side of the ball, facing the goal. This point would be given to the projection algorithm as the robot's goal position. The algorithm would then plot the robot's course to the point, treating other robots as obstacles. Once the robot reaches that point, it simply kicks the ball toward the goal. Of course, if the ball or other robots move unexpectedly, the robot must make a new plan.

If the movement of the other robots can be roughly predicted, then that information can be entered into the phase space, so that certain cells will be marked as being off-limits during future time intervals. Possible methods for predicting the movement of other robots based on their past activity range from canonical variate analysis to hidden Markov models.

Another issue concerns how to coordinate team activity. All the players on one team should not rush for the ball at the same time, possibly colliding with each other. And if the ball is far from the goal, teammates may need to set up their positions so they can make passes. This could be handled by only sending the closest teammate to the ball. If the ball is too far from the goal to be kicked there directly, other teammates could be sent to positions between the ball and goal. (Note: These suggestions for applying spatiotemporal projection heuristics to soccer-playing robots are not meant to address the broader philosophical issues that concern my research, such as how robots could develop grounded representation from experience.)

Is the algorithm real-time? In a dynamic environment, a plan is only useful if it can be made quickly. Otherwise, the plan will be out-of-date before it has been completed. In the Java simulation, the projection heuristics generally are able to plan a path to the goal in a fraction of a second. Since Java is an interpreted language and since, even with a just-in-time compiler, Java programs run a third to a tenth the speed of C programs, better performance is obtainable in a dedicated system. Of course, in a robotic system, CPU cycles may be required for other tasks like image processing and motor control. It is also important that sensorimotor movements can be predicted rapidly since planning continuously relies on this information (§3.1).

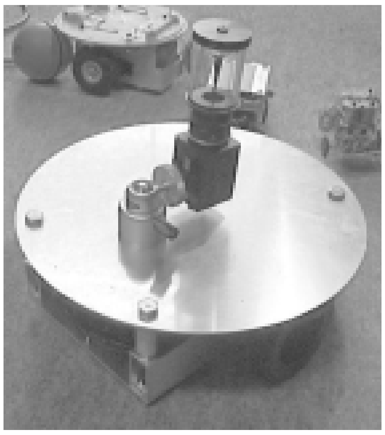
Figure 7: Ψro the robot.

The algorithm is fast enough given the current performance of medium-sized soccer robots. And even if it took several seconds, for example, to plan backward from the goal, the robot can be guided by reactive behaviors during this time. Its moving position can be continuously updated in the phase space until the projection algorithm reaches the corresponding cell. However, computation time becomes a more serious concern in higher dimensional phase spaces. A humanoid robot, for example, may have at least 28 degrees of freedom. Given that the number of cells in the phase spaces grows exponentially with its dimensionality, such complex phase spaces probably justify the added overhead of a variable resolution quantization [12, 13].

3 Mobile Robot Application

As we mentioned in the last section, the projection algorithm is very general and may be applied to many kinds of motor planning tasks. We now consider a specific application to mobile robot navigation using Ψro, our homemade robot (see Figure 7). In building Ψro, we took the "remote brain" approach, so that we could keep the robot small while making use of a sophisticated image processing board. An image from an omnidirectional camera is transmitted to the Hitachi IP5005 board via microwave (1.2 GHz). A 500 MHz Pentium III PC performs such tasks as recognition, decision making, and planning. A wireless modem then transmits motor commands to the robot's controller board.

It is clear from Figure 1 that motor planning (i.e., the projection algorithm) depends on a sensorimotor map. The algorithm also needs to know where the goal is and, therefore, depends on processes involved in recognizing affordances and deciding which affordance to exploit. We briefly summarize the algorithms used

[7]Mitsubishi Electric MOVEMASTER RV-M1.

[8]http://www.robocup.org

for learning a sensorimotor map and for recognizing affordances in §§3.1 and 3.2.

3.1 Learning a Sensorimotor Mapping

A robot may need to respond immediately to unexpected changes in its sensorimotor relations. Unlike neural networks, closest point methods are capable of learning immediately from single instances. However, they may give slower predictions and poorer generalizations, especially in high dimensional phase spaces. Since the projection algorithm may require thousands of predictions to make a single plan and prediction errors can easily accumulate, we need an algorithm that can combine the strengths of closest point algorithms and neural networks. *Partition nets* [10] are an efficient on-line learning algorithm that can make fast predictions about well-practised movements while quickly adapting to changes in sensorimotor relations.

The algorithm is interesting from a cognitive standpoint because it mirrors certain aspects of how humans learn. For example, in learning to ride a bike, we progress from reasoning from memories of particular events or instructions to a kind of automatic response that demands little conscious effort or attention.

For simplicity we may view sensorimotor relations as a function from an input state $\vec{\imath}_t$ at time t to an output state $\vec{o}_{t+1}$ at time $t+1$:

$$\vec{o}_{t+1} = f(\vec{\imath}_t)$$

where the robot's perceived state $\vec{s}_t$ and motor signals $\vec{a}_t$ at time t determine $\vec{\imath}_t$, and its perceived outcome state determines $\vec{o}_{t+1}$. To give a concrete example, the robot Ψro sees an object at position (x_t, y_t) on its camera image planes, and it needs to predict the next location of the object (x_{t+1}, y_{t+1}) after it has turned its right and left wheel by (l, r) (see Figure 7). Therefore, it needs to learn a mapping $g(\vec{\imath})$ that approximates f.

As Ψro move about, partition nets memorize input-output pairs $(\vec{\imath}_t, \vec{o}_{t+1})$, unless they are able to predict $\vec{o}_{t+1}$ from $\vec{\imath}_t$ with enough accuracy. At first, a prediction about the consequence of an action in some new state is based on a Gaussian blending of range values for pairs that are near to the vector $\vec{\imath}_t$ in the domain. Meanwhile, a neural network is being trained during spare CPU cycles using the backpropagation learning algorithm [18]. When the predictive accuracy of the network eclipses that of the blending functions, partition nets switch over to using the network for prediction.

If the number of memorized input-output pairs reaches a threshold, the input space is partitioned and two networks are formed — one whose receptive field covers the subspace on the low side of the partition, and the other whose receptive field covers the subspace on the high side. The input-output pairs are also partitioned into two matching subspaces. This partitioning concentrates network weights in more complex or often explored areas of the mapping where they are needed most. It also facilitates the rapid look-up of nearby points when using Gaussian blending [17, 21].

3.2 Learning to Recognize Affordances

Segmentation and tracking. Ψro's goal is survival. It must intercept tasty robots and avoid poisonous and dangerous robots in a cluttered dynamic environment. Ψro uses motion information to segment and track potential sources of invariance.

Image preprocessing. Once the robot has segmented the potential source of invariance, it converts it to a canonical form. This highlights invariance and facilitates comparison between different segmented images. The process involves *(1)* removing the background, *(2)* scaling the segmented image to fit on a 64-by-64 grid, *(3)* recoding color information in terms of an intensity, red-versus-green, and blue-versus-yellow channel, *(4)* decomposing the recoded image into a set of wavelet coefficients, and *(5)* quantizing the coefficients, retaining only the largest in absolute magnitude, to form a compact signature for each segmented image. The wavelet transform and other multiresolution techniques are useful because, at any given scale, it is often hard to find invariant features.

Learning categorical representations. While Ψro is tracking a potential source of invariance, it is calculating and accumulating image signatures. Internal feedback gives Ψro the affordance when it makes contact with it. The robot then creates a *categorical representation* [4] by statistically filtering out all signature values except those that tend not to vary among signatures of the same affordance category but vary among signatures of different affordance categories. Once Ψro has learned some categorical representations, it predicts the affordance from the representation that best matches the image signatures. If Ψro miscategorizes, it refines its categorical representations accordingly and may learn several representations in order to discriminate the same affordance.[9]

3.3 Projection Heuristics Revisited

Relying exclusively on sensors that are attached to the robot complicates both the projection algorithm and the algorithms that support it. If an ordinary camera is placed on the robot, nearby objects may easily disappear from view as the robot advances or turns (unless several camera are used). In this case, the robot should maintain some prediction about the relative locations of unseen and possibly moving objects. If the robot can only respond to what appears in a narrow angle of sight, planning is seldom much better than simply moving toward the goal.

Instead of developing an algorithm to second-guess the positions of objects that are out of view, we chose to use an omnidirectional camera. However, occlusion is a more serious problem for an omnidirectional camera than it is for a camera mounted overhead, and it is difficult to recognize distant objects because they appear smaller. Thus, the system works better if the robot can see over obstacles and if potential goals are reasonably large.

[9]This subsection is taken from [9].

From the standpoint of the projection algorithm, the main issue with using a robot-mounted camera is how to quantize the phase space. When the robot moves, distant objects generate much shorter flow vectors than nearby ones. The important point about the quantization is to avoid reexamining nearby areas in the phase space. But since it is basically just blocking out areas from future consideration, the partitioning of the phase space need not be exact; quantized regions may overlap.

Therefore, to determine whether a region has already been examined, the algorithm searches among the points in the phase space that have been examined so far, and it finds the point among them that is closest to the point under consideration. (This search is performed in $\log_2 n$ time [21].) An elliptical region around this point is calculated based on the length of flow vectors associated with nearby points in the phase space. If the query point lies within that region, it is not considered further.

4 Conclusion

The simulation demonstrates the usefulness of using a heuristic that balances the robot's time-to-goal and distance-from-start in setting the search priority for backward searches. On average the algorithm obtained more than a seven fold gain in computational efficiency at the expense of only a 3.3 percent increase in path length.

In a robotic system the weighting between time and distance can be adjusted according to the computing time available. If the robot has very little time to plan, its path somewhat resembles one calculated by "hill climbing." If the robot has a little more time, the path becomes smoother. Given more time, the path approaches a coarsely optimal solution, even though its calculation takes just a fraction of the time of required by dynamic programming.

References

[1] Fodor, J. A. & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition, 28*(1-2), 3-71.

[2] Fodor, J. A. (1994). J. A. Fodor. In S. Guttenplan (Ed.), *A companion to the philosophy of mind.* Oxford: Blackwell.

[3] Gibson, J. J. (1979). *The ecological approach to visual perception.* Boston, MA: Houghton Mifflin.

[4] Harnad, S. (1987). Category induction and representation. In S. Harnad (Ed.), *Categorical perception: The groundwork of cognition.* Cambridge: Cambridge University Press.

[5] Harnad, S. (1990a). The symbol grounding problem. *Physica D, 42*(1-3), 335-346.

[6] Harnad, S. (1993). Problems, problems: The frame problem as a symptom of the symbol grounding problem. *Psycholoquy, 4*(34). frame-problem.11.

[7] MacDorman, K. F. (1992). *First year report and thesis proposal.* Computer Laboratory, Cambridge.

[8] MacDorman, K. F. (1997). *Symbol grounding: Learning categorical and sensorimotor predictions for coordination in autonomous robots.* Technical Report No. 423. Computer Laboratory, Cambridge (e-mail librarian@cl.cam.ac.uk for a copy).

[9] MacDorman, K. F. (1999). Grounding symbols through sensorimotor integration. *Journal of the Robotics Society of Japan, 17*(1), 20-24.

[10] MacDorman, K. F. (1999). Partition Nets: An efficient on-line learning algorithm. ICAR *99: Ninth International Conference on Advanced Robotics*, Tokyo.

[11] Malcolm, C. M. (1995). The SOMASS system: A hybrid symbolic and behaviour-based system to plan and execute assemblies by robot. In J. Hallam, et al. (Eds.), *Hybrid Problems, Hybrid Solutions*, pp. 157-168. Oxford: ISO Press.

[12] Moore, A. W. (1991). Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In L. Birnbaum & G. Collins (Eds.), *Machine Learning: Proceedings of the Eighth International Workshop.* San Mateo, CA: Morgan Kaufmann.

[13] Moore, A. W. & Atkeson, C. G. (1995). The partigame algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning, 21*(3), 199-233.

[14] Nakawaki, D., Cisek, R., MacDorman, K. F., Joo, S., & Miyazaki, F. (1998). Coaching information determined from dynamic modeling based on a total energy analysis. *16th Annual Conference of the Robotics Society of Japan* (vol. 1), September 18-20, 1998, Hokkaido University, pp. 45-6.

[15] Newell, A. & Simon, H. A. (1972). Human problem solving. Englewood Cliffs, NJ: Prentice-Hall.

[16] Newell, A. (1980). Physical symbol systems. *Cognitive Science, 4*, 135-183.

[17] Omohundro, S. M. (1991). Bumptrees for efficient function, constraint, and classification learning. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, *Advances in Neural Information Processing Systems 3.* San Mateo, CA: Morgan Kaufmann.

[18] Rummelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representation by error propagation. In D. E. Rummelhart and J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition*, Vol. 1, Ch. 8. Cambridge, MA: MIT Press.

[19] Schyns, P.G., Goldstone, R.L., & Thibaut, J.-P. (1998). The development of features in object concepts. *Behavioral and Brain Sciences, 21*(1), 1-17.

[20] Sidgwick, R. (1998). *Algorithms in C* (3rd ed.). Reading, MA: Addison-Wesley.

[21] Sproull, R.F. (1991). Refinements to nearest-neighbor searching in KD trees. *Algorithmica, 6*(4), 579-589.