Structure and Synthesis of Robot Motion

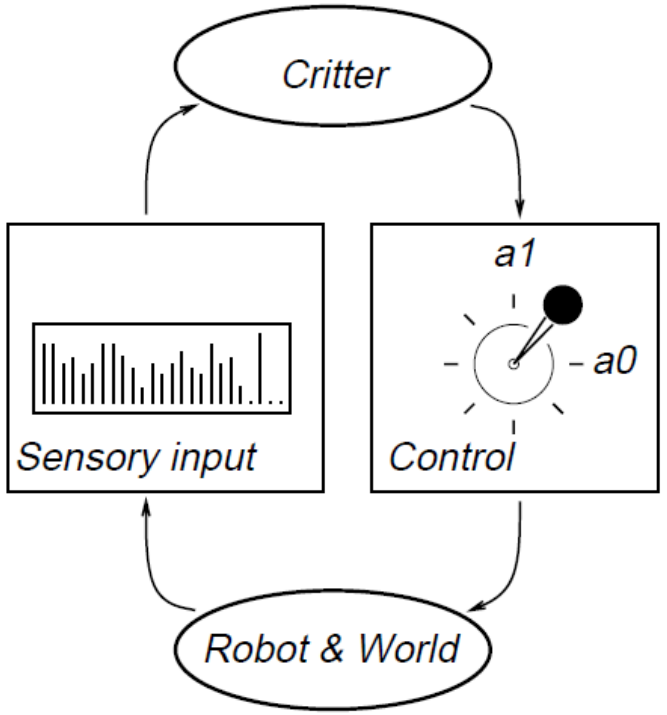
Making Sense of Sensorimotor Systems

Subramanian Ramamoorthy
School of Informatics

2 February, 2012

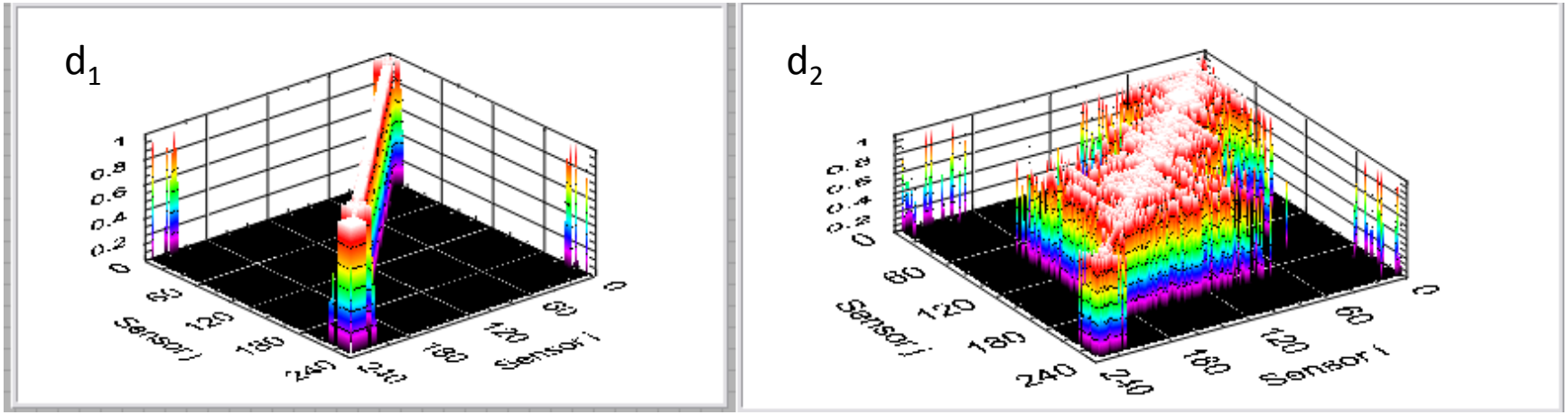
Recap: Bootstrap Learning Framework

- Learn models of robot and environment with no initial knowledge of what sensors and actuators are doing
- Many learning methods begin this way, e.g., RL, but the goal here is to construct a representation incrementally and continually as well



D. Pierce, B.J. Kuipers, Map learning with un-interpreted sensors and effectors, *Artificial Intelligence* 91:169-227, 1997.

Example Trace



$$i \approx_k j \quad \text{if } d_{k,ij} < \min\{\varepsilon_{k,i}, \varepsilon_{k,j}\}.$$

$$\varepsilon_{k,i} = 2 \min_j \{d_{k,ij}\}.$$

Extending the Group Notion

We can reason transitively about similarity:

$$i \sim j \quad \text{iff} \quad i \approx j \vee \exists k\colon\ (i \sim k) \wedge (k \sim j).$$

So, a wandering trace might yield something like this as groups:

(0 1 2 22 23) (0 1 2 3 23) (0 1 2 3 4) (1 2 3 4 5) (2 3 4 5 6) (3 4 5 6 7)
(4 5 6 7) (5 6 7 8 9) (7 8 9 10) (7 8 9 10 11) (8 9 10 11 12) (9 10 11 12 13)
(10 11 12 13 14) (11 12 13 14 15) (12 13 14 15 16) (13 14 15 16 17)
(14 15 16 17 18) (15 16 17 18 19) (16 17 18 19) (17 18 19 21) (20)
(19 21 22 23) (0 21 22 23) (0 1 21 22 23) (24) (25) (26) (27) (28).

After Transitive Closure

(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 21 22 23)

(20) *defective*

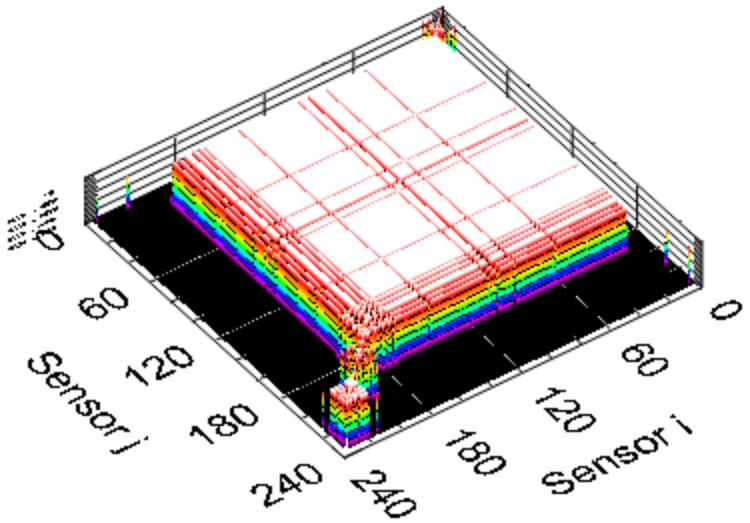
(24) *battery voltage*

(25) *east*

(26) *north*

(27) *west*

(28) *south*



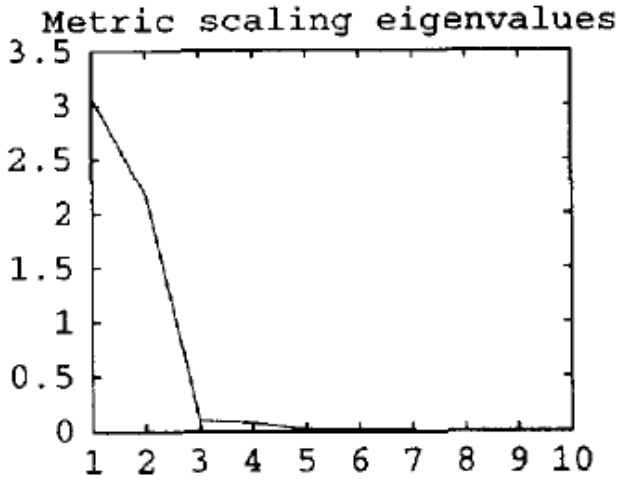
Getting at Structure of Array

- Task is to find an assignment of positions (in space) to elements that captures the structure of the array as reflected in distance metric d_1.
- Distance between positions in image ≈ distance between elements according to d_1.

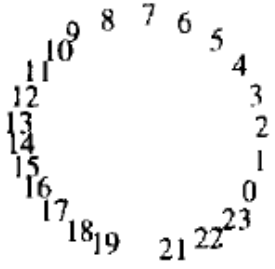
$$\| (\boldsymbol{pos}\ y_i) - (\boldsymbol{pos}\ y_j) \| = d_{1,ij},$$

- This is a constraint satisfaction problem: n sensor elements yield n(n-1)/2 constraints.
- Solve by metric scaling: $$E = \frac{1}{2} \sum_{ij} (\| (\boldsymbol{pos}\ y_i) - (\boldsymbol{pos}\ y_j) \| - d_{ij})^2.$$

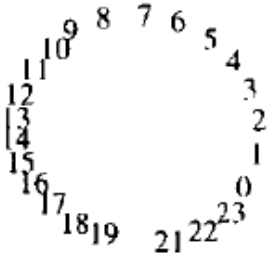
Structural Model of Distance Array



a



b



c

Key Idea: Language of Features

- We want a critter to develop an understanding of the world in order to act at the level of navigation, etc. How does it know what to do?
- The critter tries to develop an **abstract interface** – a sequence of increasingly sophisticated sensorimotor interfaces
- Key to this is the definition of features:
  - Sensory side: Re-describe sensor readings into higher level signals
  - Motor side: Define new control laws at these levels
- Based on these features, the critter adopts a **generate and test** approach

On Features

They provide:

- Change of representation: Although the initial vector of distance readings provides the same information, representing it as a 2D ring enables new control possibilities
- Focus of attention: If a lot of things are happening (information overload, noise) the critter can define a few salient features to be tracked, e.g., nearest distance reading
- Dimensionality reduction: In the interest of keeping computation tractable, it is key to limit the dimensionality to the effective dimensionality of the world/problem

Features enable Operators

- e.g., by defining a vector or matrix grouping, we get to define linear algebraic operations
- More interesting are image operators, e.g., motion detector

  First, recall our notion of an 'image'

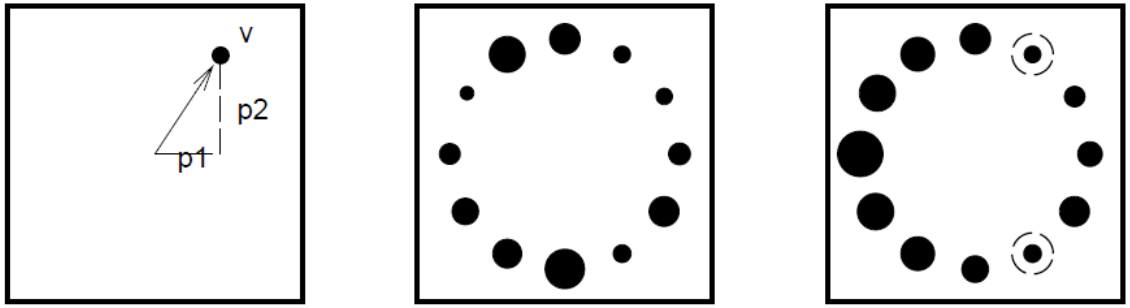


Figure 2.5: a) An example of a two-dimensional image element of type $(\mathcal{S}, \mathcal{S}^2)$. The position vector is represented by the arrow; the value is represented by the size of the disk. b) An example of an image of type $(\mathcal{S}, \mathcal{S}^2)^{12}$, a sequence of image-elements. c) An example of a focused image. The circled elements are local minima and have been assigned a strength of 1.

More Features

- In general, one could define more sophisticated features building on the elementary primitives
- Here is a field element that captures motion at each point in an image

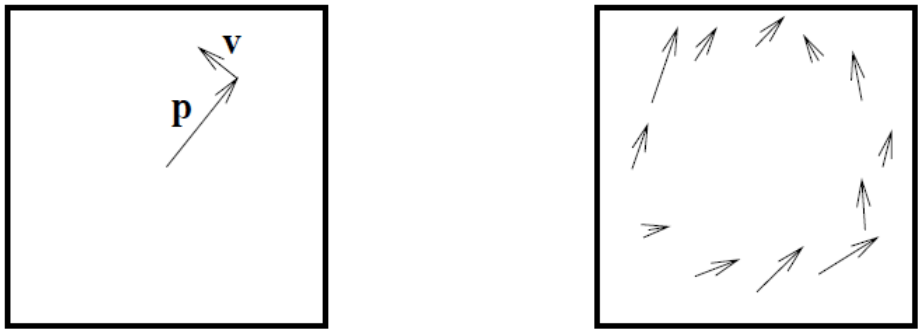


Figure 2.6: a) An example of a field element of type $(\mathcal{S}^2, \mathcal{S}^2)$. The position and value vectors are both represented by arrows. b) An example of a field, a sequence of field-elements, of type $(\mathcal{S}^2, \mathcal{S}^2)^{12}$.

A Motion Operator

- Detection of motion requires spatial and temporal information: both are available in the image features
- We can think of a sequence of images in terms of an intensity function E(p,t) – maps image positions to values over time
- This function has a spatial and time derivative
  - Changes in these define something about environment (e.g., edges)
- Based on this motivation, one can understand the optical flow operation in image processing:

$$\mathbf{v} = -\frac{E_t}{\|\vec{E}_\mathbf{p}\|}\frac{\vec{E}_\mathbf{p}}{\|\vec{E}_\mathbf{p}\|} = -\frac{E_t\vec{E}_\mathbf{p}}{\|\vec{E}_\mathbf{p}\|^2}$$

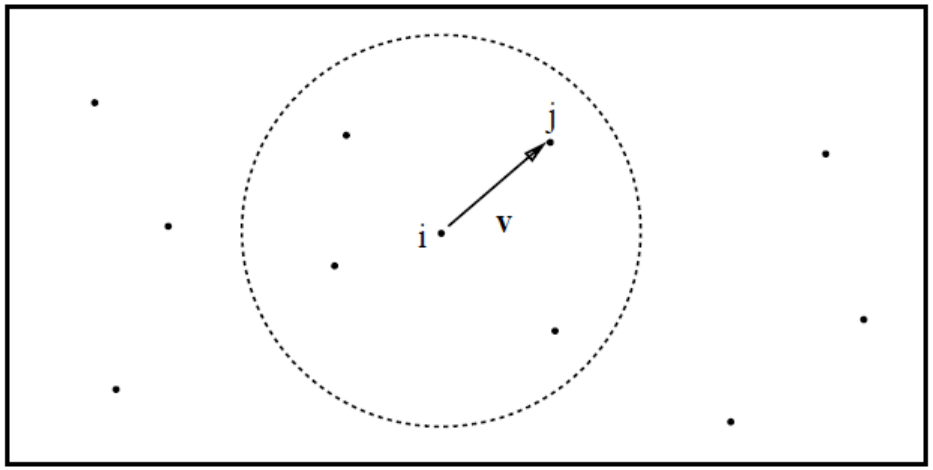
- Could also just pay attention to $\mathbf{v} = -E_t\,\vec{E}_\mathbf{p}$

Instantaneous Motion Vector Field

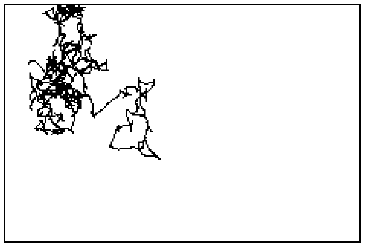
Building on the above, we can define such a field where we compute an averaged mvf for each element of the image

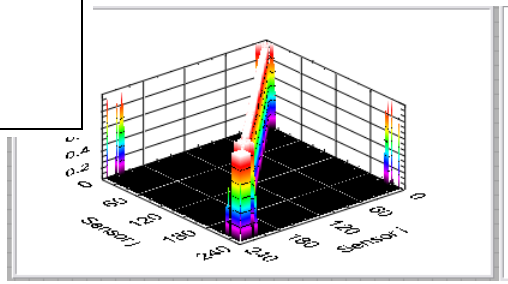
$$\mathbf{pos}(\mathbf{motion}\ x) \stackrel{\text{def}}{=} \mathbf{pos}\ x$$

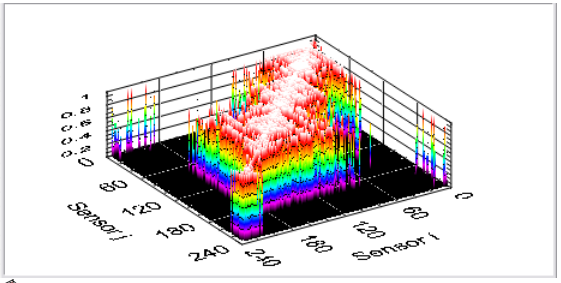
$$(\mathbf{val}(\mathbf{motion}\ x))_i \stackrel{\text{def}}{=} \sum_{j \in (\mathrm{nbrs}\ i)} \mathbf{v}_{ij}/\|\mathbf{p}_{ij}\|$$

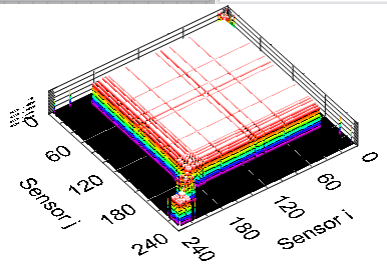


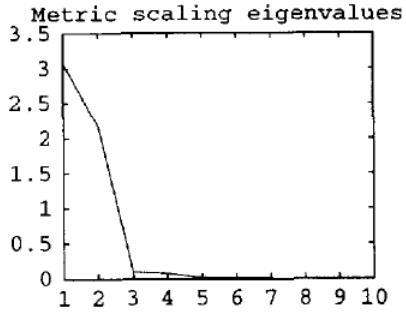
Sequence so far… on the sensor side



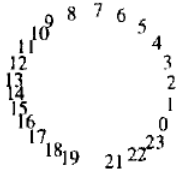




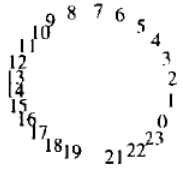




a



b



c

Learning on the Motor side

Summary of learning method:

- Discretize the space of motion control vectors
- Compute average motion vector fields
- Apply PCA
- Identify primitive actions
- Define a new abstract interface using these actions

Average Motion Vector Field

$$amvf_i = \mu\left((\textbf{motion}\ x) \mid (= \mathbf{u}\ \mathbf{u}^i)\right)$$

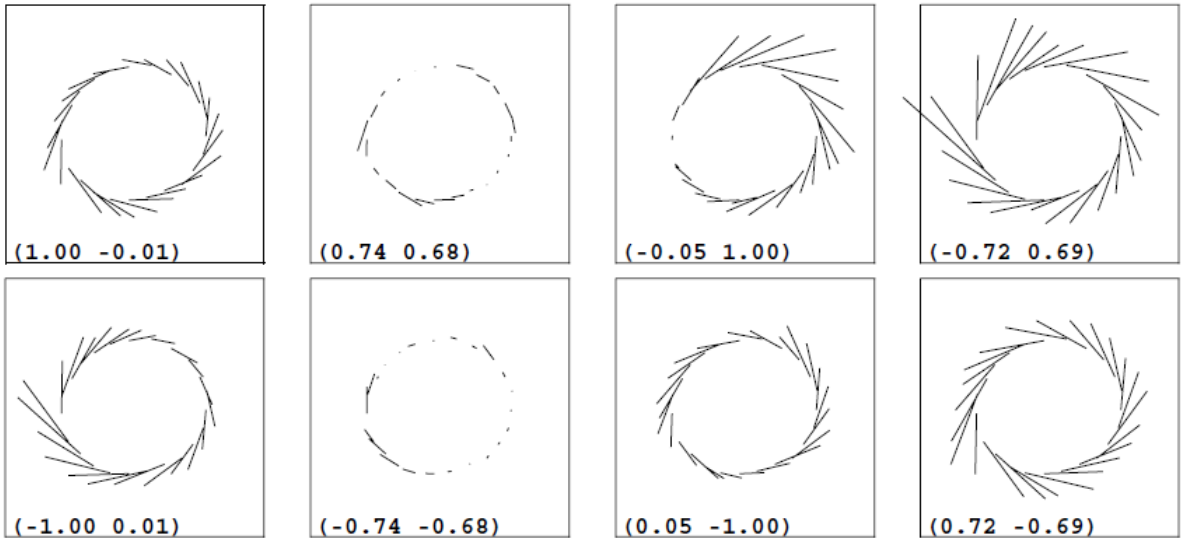


Figure 5.1: Examples of average motion vector fields (*amvf*'s) and their associated motor control vectors. An *amvf* associates an average local motion vector with each position in the image (see Figure 4.8). The examples in this section were all produced in an experiment involving the robot and environment described in Section 4.1.

PCA over AMVFs

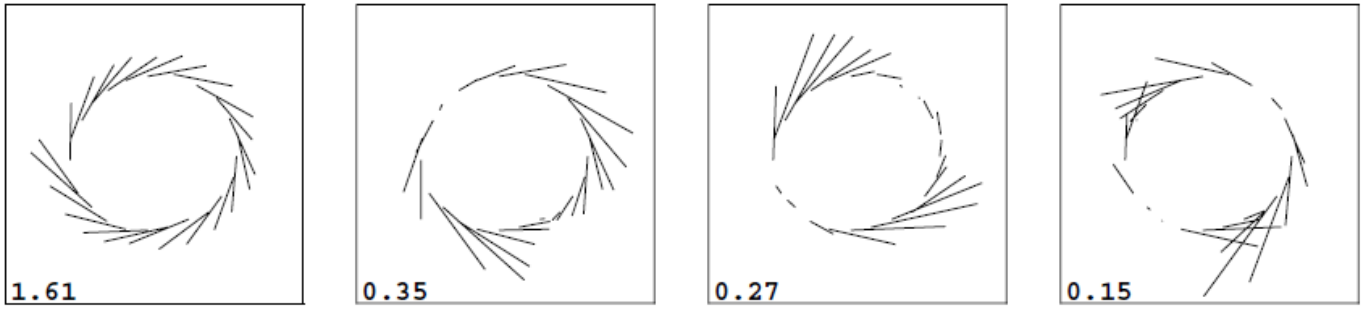


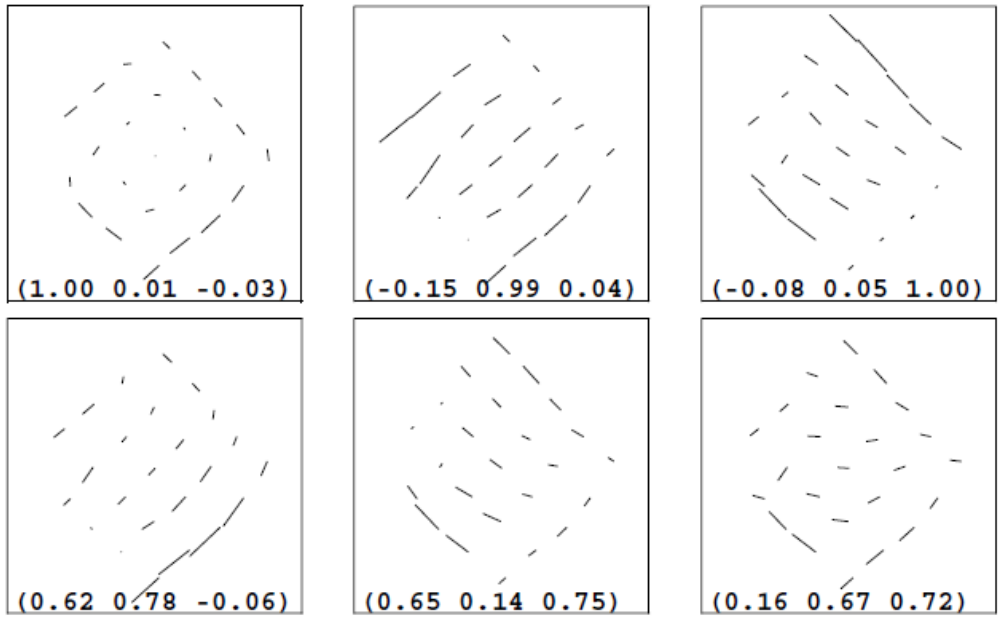
Figure 5.3: The first four eigenvectors and the standard deviations of the associated principal components for the space of average motion vector fields. The first corresponds to a pure rotation motion and the second corresponds to a forward translation motion. (The top-left elements in these diagrams are associated with the robot's front sensor s_0). The robot's motor apparatus can produce the first two effects directly using the motor control vectors shown.

Defining New Primitive Actions

- We now have two things:
  - AMVFs associated with actions that were tried
  - Eigenvectors that form a basis for these AMVFs
- We define primitive actions by looking for actions that can enforce motion along the eigenvectors
- If an eigenvector and AMVF have an angle of 0 between them, they use same actions (180^0 corresponds to the opposite)
- By finding AMVFs that are within 45^0, we can define $\mathbf{u}^{i+}$ and $\mathbf{u}^{i-}$ that are useful to enforce motion in that direction
- From this, we define an abstract interface (with turn & travel)

$$\mathbf{u} = u_0\mathbf{u}^0 + u_1\mathbf{u}^1$$

AMVF for the Roving Eye



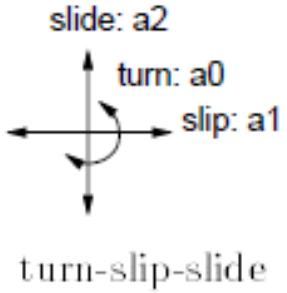


Figure 5.6: Examples of average motion vector fields and their associated motor control vectors for the roving eye.

Eigenvectors for the Roving Eye

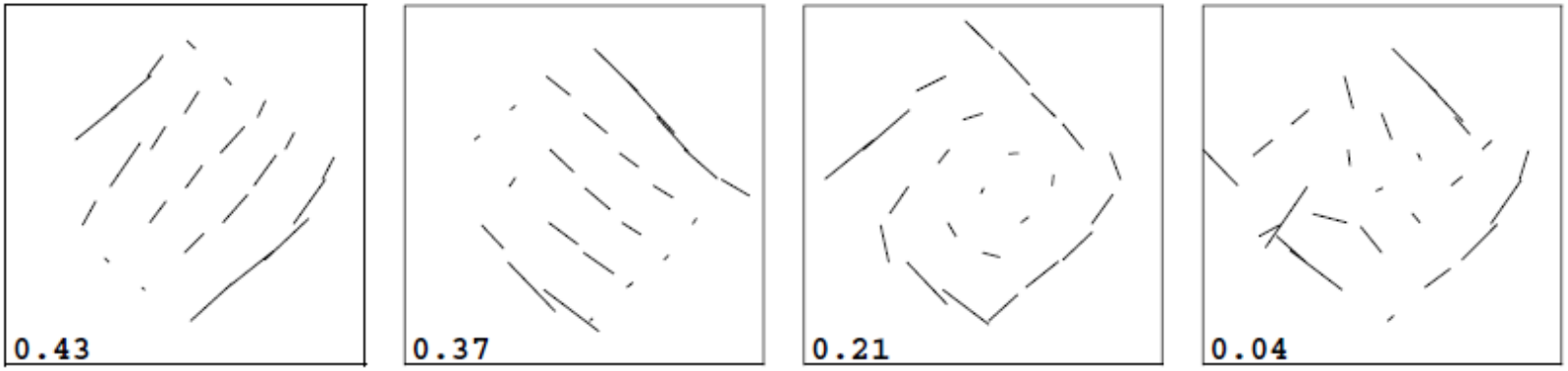


Figure 5.7: The first four eigenvectors and standard deviations for the roving eye.

Moving on, our game plan

Knowing how to abstract on both the sensor and motor side, our game plan will be to generate candidate states and features – looking for regularities that can be explained using these generated entities.

For instance, we can define a local state variable as:

Let $\hat{\mathbf{u}}$ be the vector of control signals u_j. A scalar feature y_i is a *local state variable* if the effect of the control signals on y_i can be approximated locally by

$$\dot{y}_i = \mathbf{m}_i \cdot \hat{\mathbf{u}} \quad (= \sum_j m_{ij}\, u_j) \tag{6.1}$$

where $\mathbf{m}_i$ is nonzero.

Hierarchy of Features and Generators

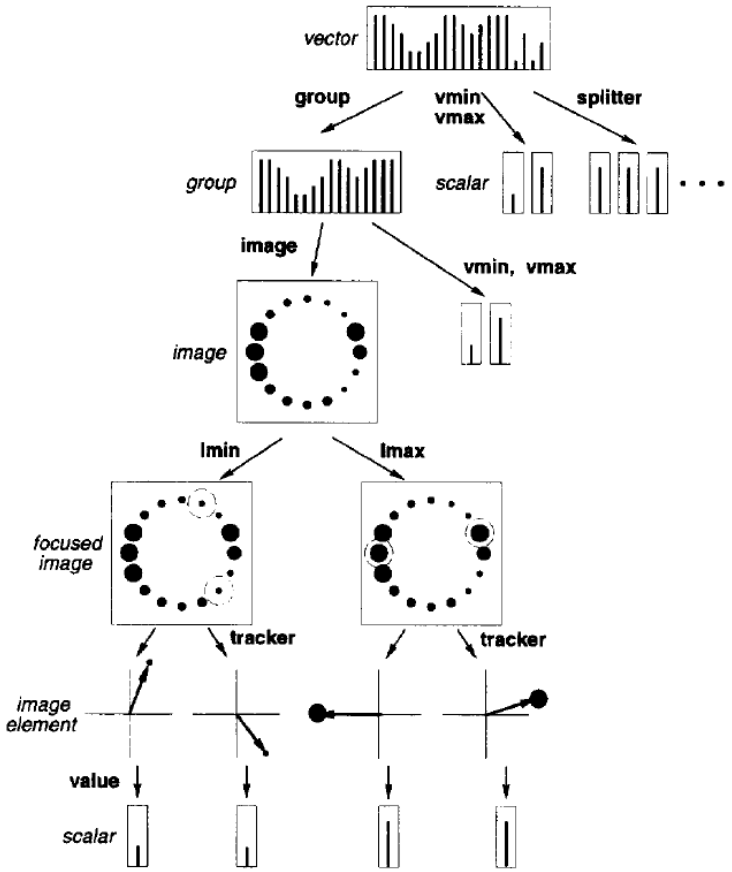
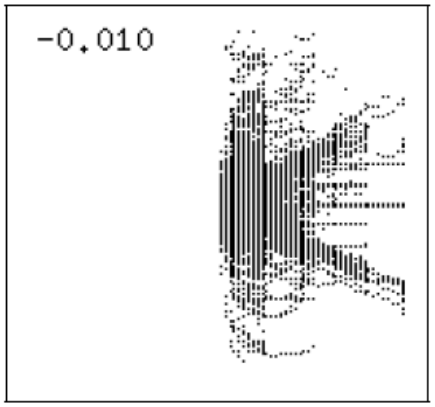
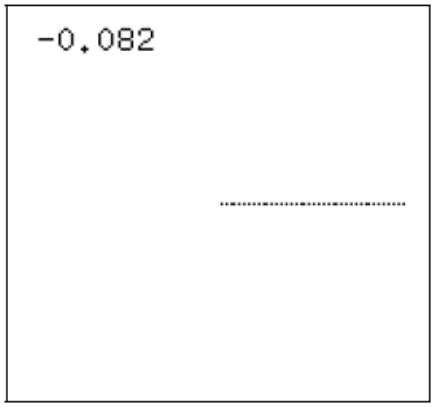


Fig. 9. The complete hierarchy of features and generators in the learning agent's feature-learning process used to produce candidate local state variables. The feature generators are shown in bold face; the feature types are shown in italics.

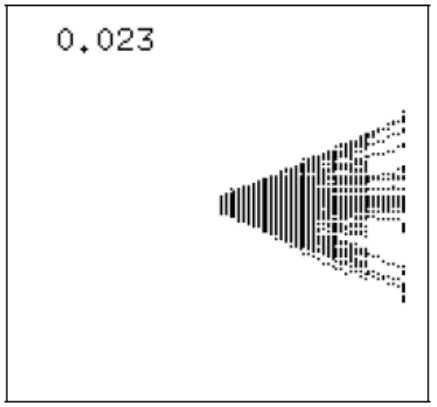
Looking for Action Models - Action Independent



Δs_0 vs. Δt



Δs_{24} vs. Δt



lmin vs. Δt

Figure 6.2: Plots of Δy_i vs. Δt for three features. Whenever a new motor control vector is used, Δy_i and Δt are reset to 0. These are used to see if it is possible to predict the behavior of the feature independently of the motor control vector. Here, *lmin* is short for *s-g0-im-lmin-tr-val*. The numbers shown are the correlations between Δy_i and Δt.

Looking for Action Models - Action Dependent

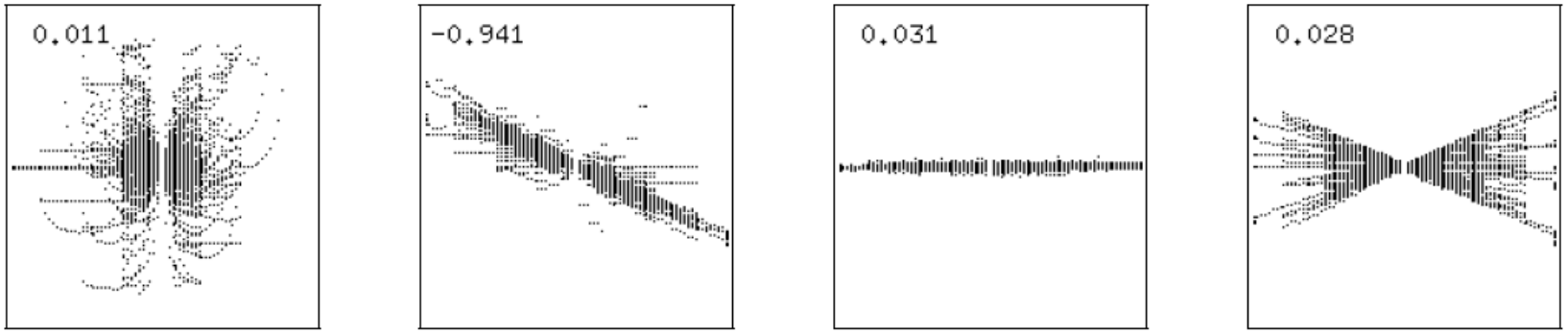


Figure 6.3: Plots of Δy_i vs. $u_j \Delta t$ for two features and two primitive actions. These are used to see if it is possible to predict the behavior of the feature as a function of the motor control vector.

Context Dependent Models

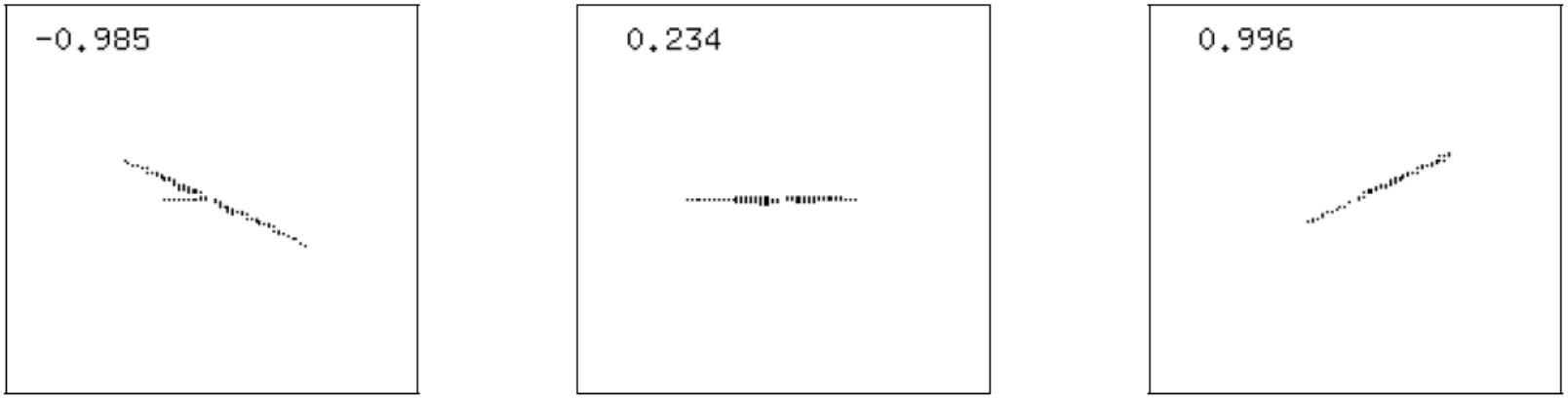


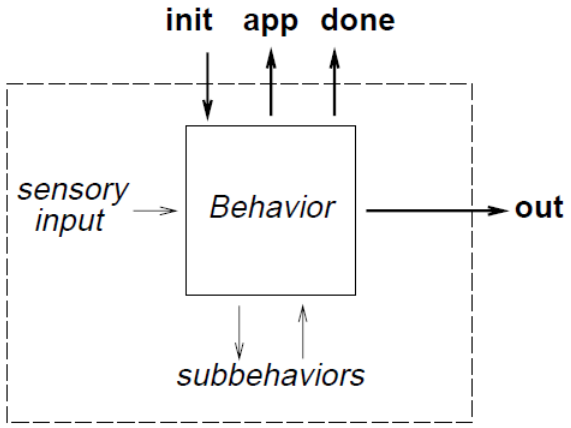
Figure 6.4: Example plots of Δy_i vs. $u_1 \Delta t$ for the *s-g0-im-lmin-tr-val* feature for three different contexts. These are used to see if it is possible to predict the behavior of the feature as a function of the motor control vector and the current context.

$$\dot{y}_i = m_{ijk}\, u_j, \text{ if } z_{ij} = k$$

After all this, we finally have a language for talking to a mobile robot!

Defining Behaviours

Consider the following interface to define a behaviour – including **init**ialization signal and **app**licability readout.



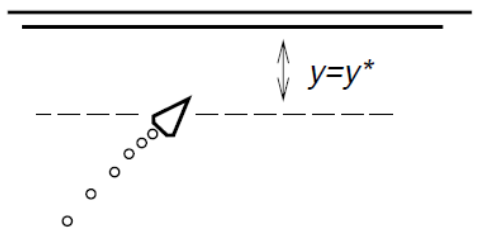
Example: homing behaviour

For each context $z_{ij} = k$,

$$
\begin{aligned}
app(k) &= \max\{0,\ 2|r_{ijk}| - 1\} \\
output(k) &= u_{ijk}\,\mathbf{u}^j \\
done &\equiv \frac{|y_i^* - y_i|}{y_i^*} < 0.1
\end{aligned}
$$

where

$$
\begin{aligned}
u_{ijk} &= \frac{2\zeta\omega}{m_{ijk}}\,e_i + \frac{\omega^2}{m_{ijk}}\int e_i\,dt \\
e_i &= y_i^* - y_i.
\end{aligned}
$$



Open-loop Path Following Behaviour

$$
\begin{aligned}
app &\equiv \frac{|y_i^* - y_i|}{y_i^*} < 0.1 \ \wedge \ z_{ij} = k \\
output &= \mathbf{u}^{\beta} + \sum_{\delta \neq j} u_{\delta} \mathbf{u}^{\delta} \\
done &\equiv \frac{|y_i^* - y_i|}{y_i^*} > 0.4 \\
&\quad \vee \text{(a new behavior becomes applicable)}
\end{aligned}
$$

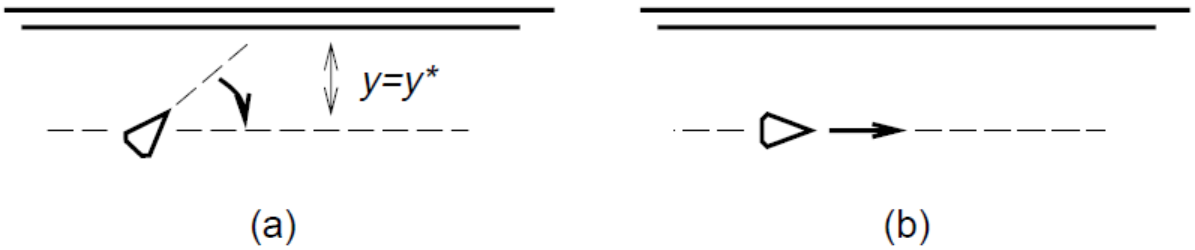
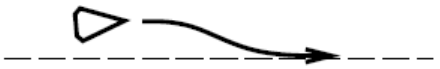


Figure 7.6: Two examples of open-loop path-following behaviors. (a) A behavior based on $\mathbf{u}^0$ (for turning) and constraint $y_i = y_i^*$ is applicable whenever $y_i = y_i^*$ since $\mathbf{u}^0$ never changes the value of y_i. (b) A behavior based on primitive action $\mathbf{u}^1$ (advancing) and constraint $y_i = y_i^*$ is applicable whenever $y_i = y_i^*$ and the robot's heading is parallel to the wall on its left (i.e., $z_{ij} = 18$) since in this context $\mathbf{u}^1$ keeps the error $e = y_i^* - y_i$ near zero.

Closed-loop Path Following



$$
\begin{aligned}
app &\equiv \forall y_i \in \mathbf{y} : \left(\frac{|y_i^* - y_i|}{y_i^*} < 0.1 \right) \wedge \forall z_{ij} \in \mathbf{z} : (z_{ij} = k_i) \\
output &= \mathbf{u}^\beta + \sum_{\delta \neq j} u_\delta \, \mathbf{u}^\delta \\
done &= \exists y_i \in \mathbf{y} : \left(\frac{|y_i^* - y_i|}{y_i^*} > 0.4 \right) \\
&\quad \vee \text{(a new behavior becomes applicable)}
\end{aligned}
$$

where

$$
\begin{aligned}
u_\delta &= \sum_{y_i \in \mathbf{y}} u_{\delta i} \\
u_{\delta i} &= \frac{2\zeta\omega}{m_{ijk\delta 1}} e_i + \frac{\omega^2}{m_{ijk\delta 1}} \int e_i \, dt \qquad \text{if } |r_{ijk\delta 1}| \geq |r_{ijk\delta 2}|, 0.6 \\
u_{\delta i} &= \frac{\omega^2}{m_{ijk\delta 2}} e_i + \frac{2\zeta\omega}{m_{ijk\delta 2}} \dot{e}_i \qquad \text{if } |r_{ijk\delta 2}| > |r_{ijk\delta 1}|, 0.6 \\
u_{\delta i} &= 0, \quad \text{otherwise} \\
e_i &= y_i^* - y_i .
\end{aligned}
$$

What does all this enable?

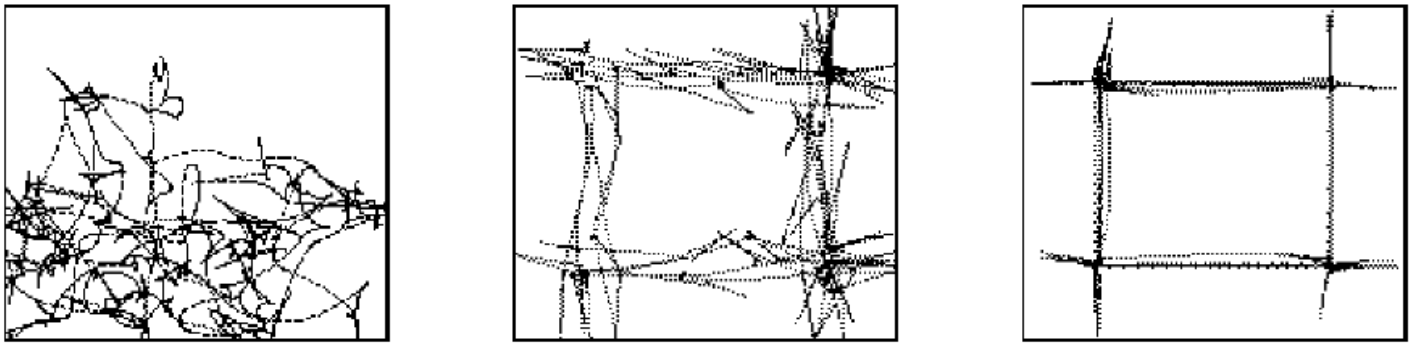
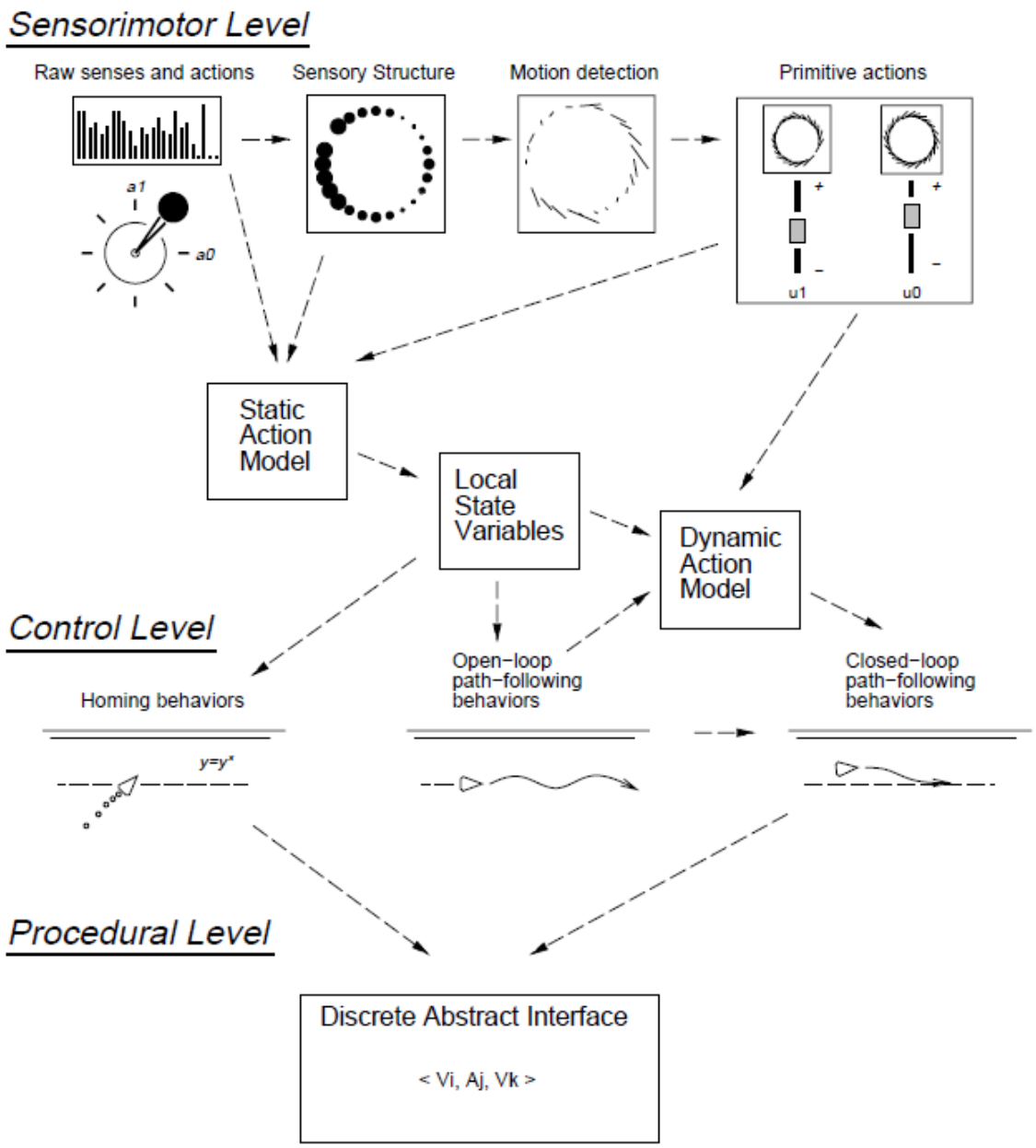


Figure 7.10: Exploring a simple world at three levels of competence. (a) The robot wanders randomly while learning a model of its sensorimotor apparatus. (b) The robot explores by randomly choosing applicable homing and open-loop path-following behaviors based on the static action model while learning the dynamic action model. (c) The robot explores by randomly choosing applicable homing and closed-loop path-following behaviors based on the dynamic action model.

Original motivation: Get from uninterpreted sensors and effectors to a **finite** state world where FSA learning, etc. is possible


Sensorimotor Level
Raw senses and actions
Sensory Structure
Motion detection
Primitive actions
a1
a0
u1
u0
Static Action Model
Local State Variables
Dynamic Action Model
Control Level
Homing behaviors
Open−loop path−following behaviors
Closed−loop path−following behaviors
y=y*
Procedural Level
Discrete Abstract Interface
< Vi, Aj, Vk >
02/02/2012
30

Can this work for more complex worlds?

Consider the case where we have more elaborate sensor arrays

- Image a sensorized Stanford bunny!

Would d_1 and d_2 suffice? Why?

Useful distance: $\text{corr}(z_*^i, z_*^j) \propto \exp(-||e(i) - e(j)||^2)$

Sensor Correlation Distance Embedding

Three steps*:

- The sensor correlation distance is computed between pairs of sensors from a set of sensor observations.
- Using the sensor correlation distance, the k-nearest neighbors for each sensor are selected to form a sparse set of distance constraints.
- The fast maximum variance unfolding algorithm is applied to the sparse constraints to generate a spatial embedding of the sensors that approximately preserves the selected sensor correlation distances.

* J. Modayil, **Discovering Sensor Space: Constructing Spatial Embeddings That Explain Sensor Correlations**, In Proc. Intl. Conf. Development and Learning 2010

Setting up the SCD

- Assuming *m* unit-normalized readings z^i_t,
- Model covariance between two channels as a Gaussian Process model:

$$\mathrm{cov}_E(Z^i_t, Z^j_u) = \sigma_f^2 \exp\left(-d_S(i,j)^2 - d_T(t,u)^2\right) + \delta_{i,j}\delta_{t,u}\sigma_n^2.$$

Systematic variance

Observation noise

- Some useful facts about correlation and covariance:

$$\mathrm{corr}(z^i_*, z^j_*) = \frac{\mathrm{cov}_S(z^i_*, z^j_*)}{\sqrt{\mathrm{cov}_S(z^i_*, z^i_*)\mathrm{cov}_S(z^j_*, z^j_*)}}$$

$$\mathrm{corr}(z^i_*, z^j_*) = < z^i_*, z^j_* >$$

Setting up the SCD

Focussing only on covariance between observations at the same time instant, ignoring observation noise,

$$\mathrm{cov}_E(Z_t^i, Z_t^j) = \sigma_f^2 \exp(-d_S(i,j)^2)$$

$$d_S(i,j) = \sqrt{-\ln(\mathrm{cov}_E(Z_t^i, Z_t^j)/\sigma_f^2)}.$$

Using the previous identities to simply and rewrite:

$$SCD(i,j) = \sqrt{-\ln(\frac{1}{T}\sum_{t=1}^{T} z_t^i z_t^j)}.$$

Isometric Embedding

- Use a technique called fast maximum variance unfolding*
- Objective of embedding is to find a redescription that preserves a local distance of the form $||x_i - x_j||^2 = d_{ij}^2$.
- By using the inner product, $X_{ij} = < x_i, x_j >$, this can be rewritten:

$$|X_{ii} - 2X_{ij} + X_{jj} - d_{ij}^2| = 0,$$

- By softening constraints, this gives an optimization problem:

$$\begin{array}{ll} \text{Maximize} & \text{tr}(X) - \nu \sum_{i \sim j} (X_{ii} - 2X_{ij} + X_{jj} - d_{ij}^2)^2 \\ \text{subject to} & (i) \sum_{ij} X_{ij} = 0 \text{ and } (ii) X \succeq 0 \end{array}$$

* K. Weinberger, F. Sha, Q. Zhu, and L. Saul, **Graph Laplacian regularization for large-scale semidefinite programming**, Neural Information Processing Systems 19, 2007.

Understanding the Optimization Problem

$$\begin{array}{ll} \text{Maximize} & \text{tr}(X) - \nu \sum_{i \sim j} (X_{ii} - 2X_{ij} + X_{jj} - d_{ij}^2)^2 \\ \text{subject to} & (i) \sum_{ij} X_{ij} = 0 \text{ and } (ii) X \succeq 0 \end{array}$$

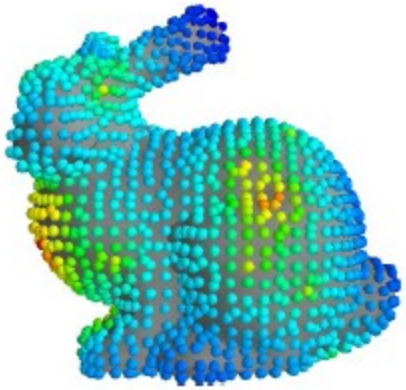
- Maximize the spread of the embedded points while minimizing violations of distance constraints
- Of the other constraints, the first one centers data points

- [Technical] Last constraint makes sure we have semidefinite or Gramian matrix – related to inner product properties
- [Technical] There are a lot of constraints (as before) but one can simplify the problem using Laplacian eigenvector approximation of the sparse connectivity graph

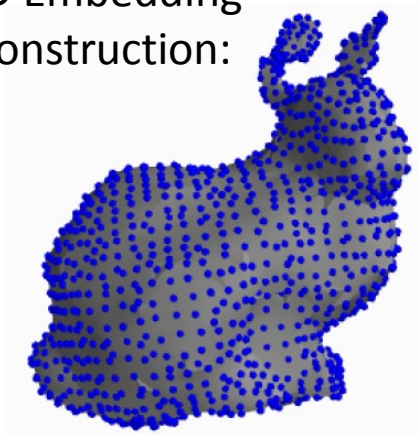
$$x_i \approx \sum_{\alpha=1}^{b} Q_{i\alpha} y_\alpha.$$

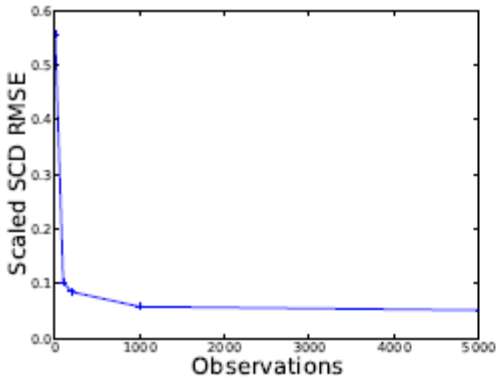
SCD Embedding Algorithm in Action

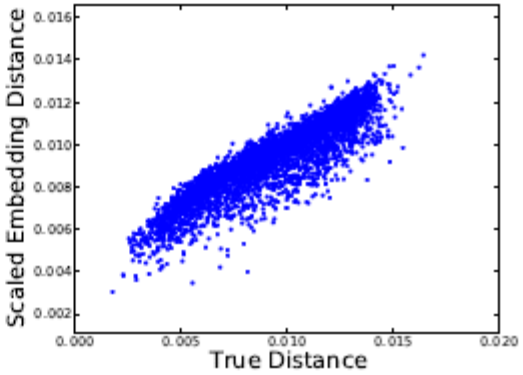
Typical image:



SCD Embedding reconstruction:

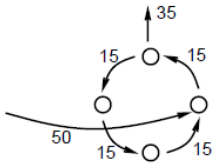




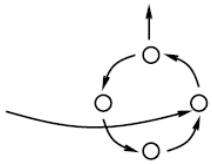


Multiple Levels of Knowledge

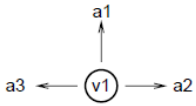
Metrical Level



Topological Level

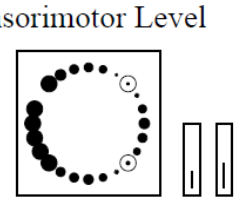


Procedural Level

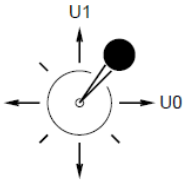


Control Level

Sensorimotor Level



Local state variables



primitive actions

Spatial Semantic Hierarchy

- A model of knowledge of large scale space consisting of multiple interacting representations
  - Inspired by the human cognitive map, a model of the same
  - Method for robot exploration and map building
- Separates and utilizes everything from control level to topology level
- Key idea:

  Quantitative knowledge at the control, causal and topological levels supports a “patchwork map” of local geometric frames of reference linked by causal and topological connections.

A Lattice for Spatial Knowledge

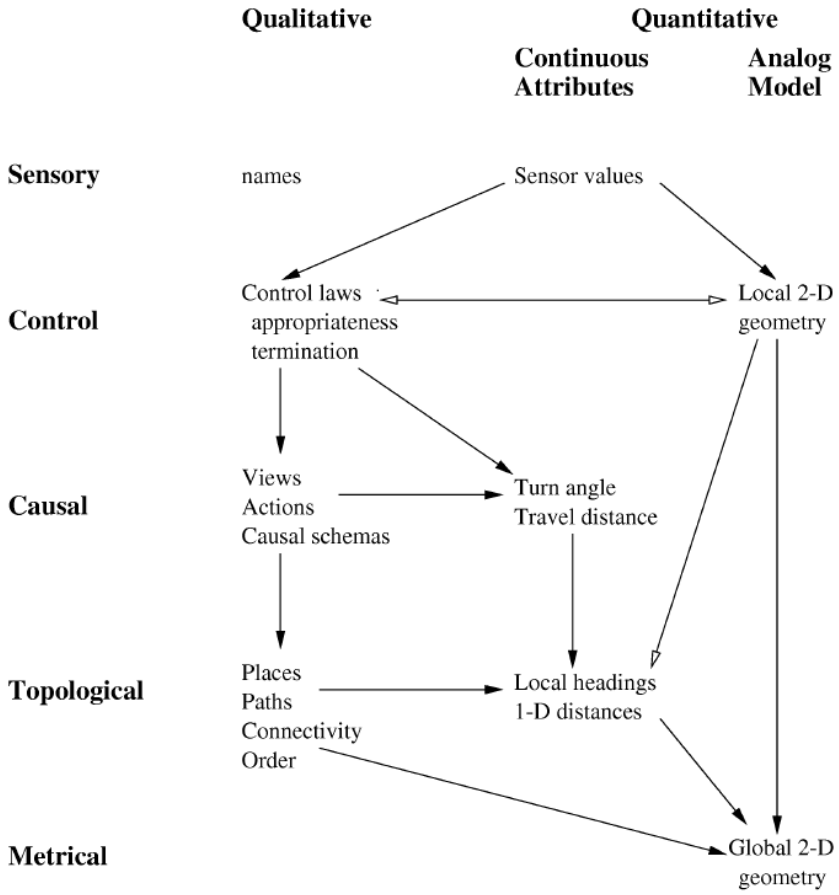
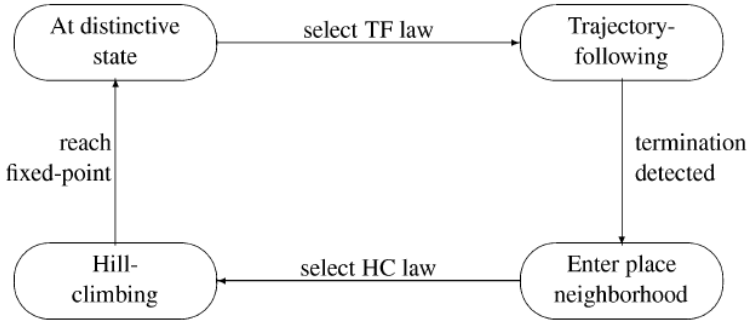
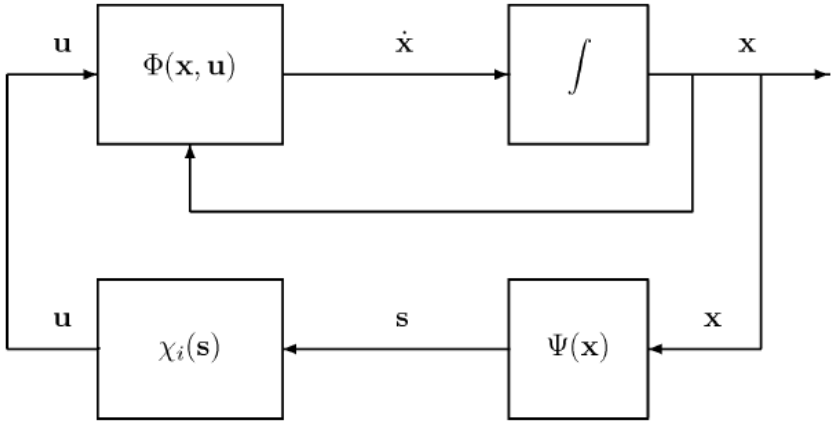


Fig. 1. The distinct representations of the Spatial Semantic Hierarchy. Closed-headed arrows represent dependencies; open-headed arrows represent potential information flow without dependency.

Control Level

Provides local hill-climbing and trajectory following: allowing a robot to get to and hold a distinctive state





Lifting to the Causal Level

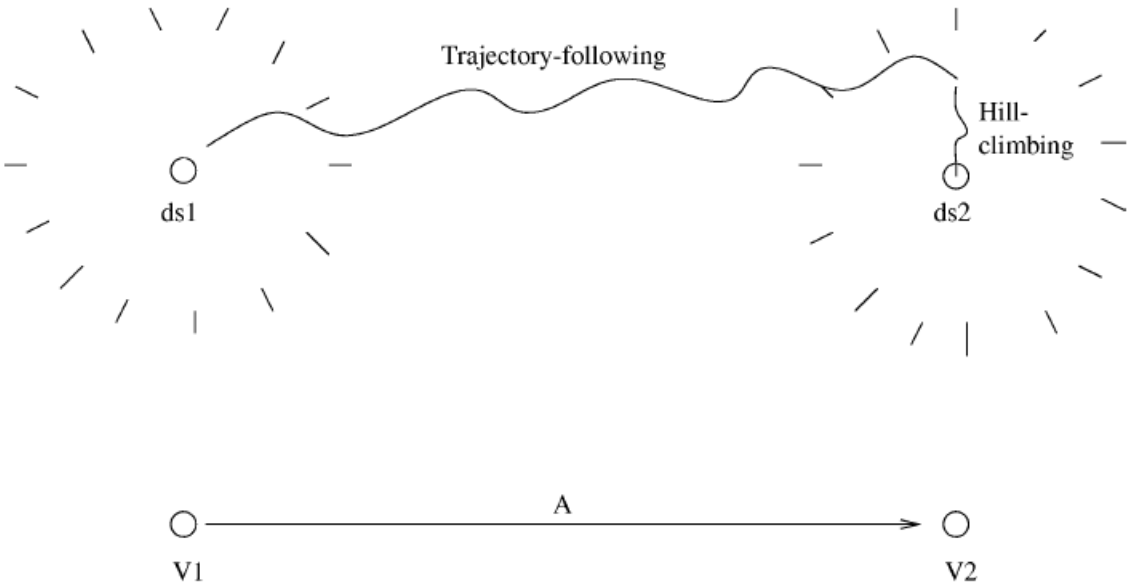
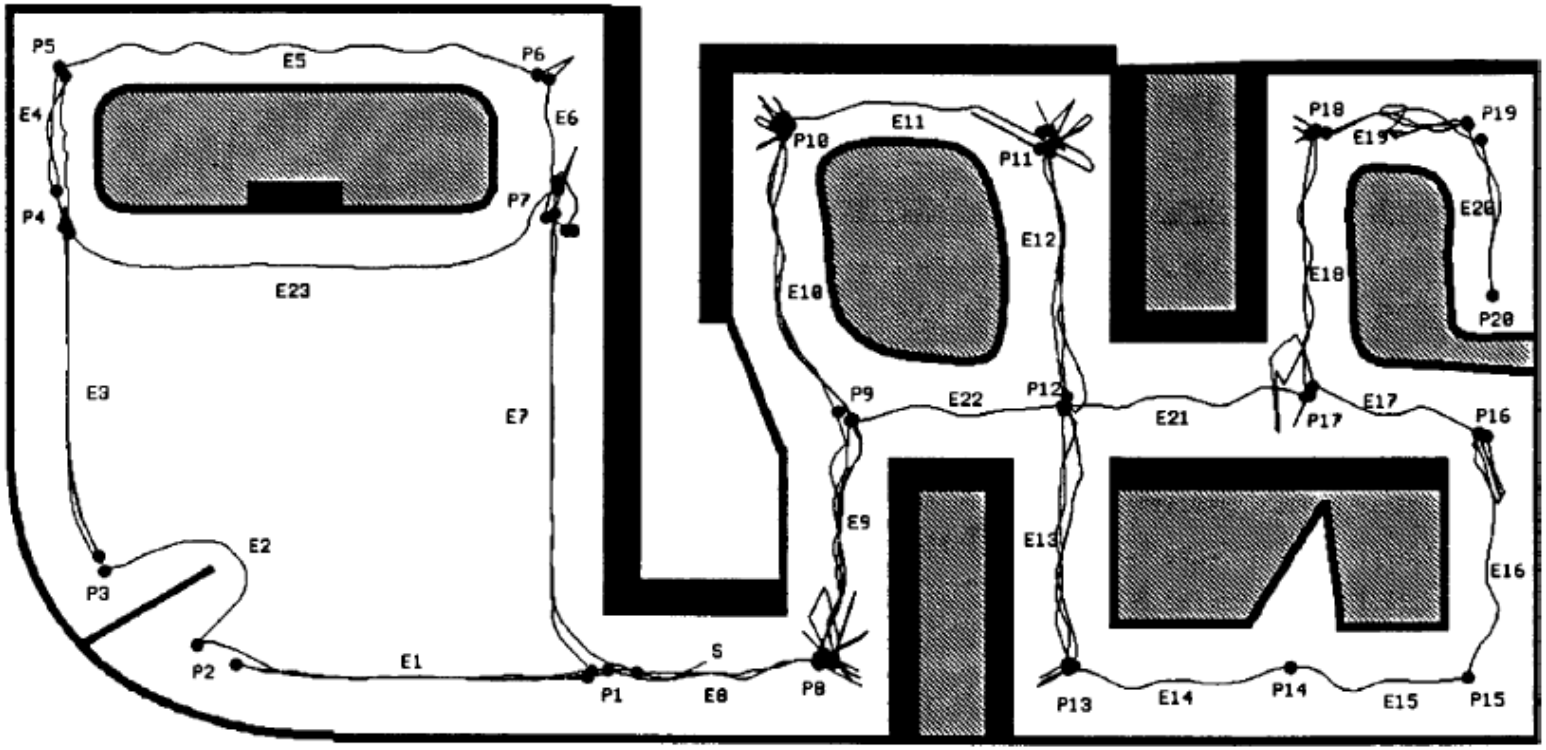


Fig. 4. Abstraction from controlled behavior to causal link $\langle V_1, A, V_2 \rangle$.

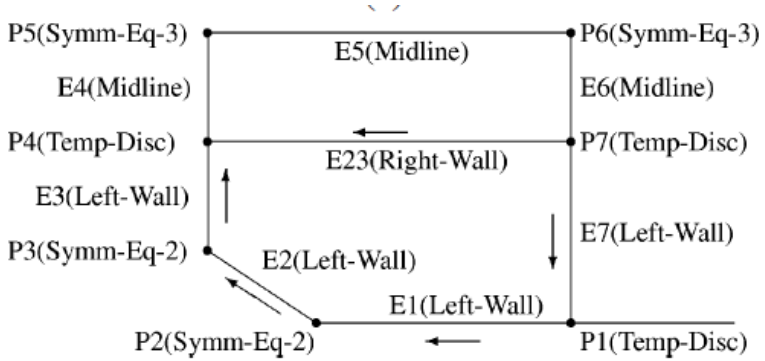
declarative: $holds(V, s_0) \wedge holds\big(V', result(A, s_0)\big)$

procedural: $holds(V, now) \Rightarrow do(A, now)$.

Exploring and Representing an Office



Topology Level – in terms of relations



at(*view*, *place*)	*view* is seen at *place*
along(*view*, *path*, *dir*)	*view* is seen along *path* in direction *dir*
on(*place*, *path*)	*place* is on *path*
order(*path*, *place1*, *place2*, *dir*)	the order on *path* from *place1* to *place2* is *dir*.
right_of(*path*, *dir*, *region*)	*path*, facing direction *dir*, has *region*
left_of(*path*, *dir*, *region*)	on its right (respectively left)
in(*place*, *region*)	*place* is in *region*.

Some Open Questions – Discuss!

- The SCD distance idea shows that sensory features can be extended using state of the art machine learning algorithms – how does one extend control actions in a similar sense?

- Space is a foundational part of our common sense and many aspects of the abstraction in SSH utilize the naturalness of it. What happens when we pass to more complex “spaces”?

- To what extent can this kind of learning be achieved in a continually changing world?