Subjective Mapping

by

Dana Wilkinson

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2007

Author's Declaration for Electronic Submission of a Thesis

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

There are a variety of domains where it is desirable to learn a representation of an environment defined by a stream of sensori-motor experience. This dissertation introduces and formalizes subjective mapping, a novel approach to this problem. A learned representation is subjective if it is constructed almost entirely from the experience stream, minimizing the requirement of additional domain-specific information (which is often not readily obtainable).

In many cases the observational data may be too plentiful to be feasibly stored. In these cases, a primary feature of a learned representation is that it be compact—summarizing information in a way that alleviates storage demands. Consequently, the first key insight of the subjective mapping approach is to phrase the problem as a variation of the well-studied problem of dimensionality reduction. The second insight is that knowing the effects of actions is critical to the usefulness of a representation. Therefore enforcing that actions have a consistent and succinct form in the learned representation is also a key requirement.

This dissertation presents a new framework, action respecting embedding (ARE), which builds on a recent effective dimensionality reduction algorithm called maximum variance unfolding, in order to solve the newly introduced subjective mapping problem. The resulting learned representations are shown to be useful for reasoning, planning and localization tasks. At the heart of the new algorithm lies a semidefinite program leading to questions about ARE's ability to handle sufficiently large input sizes. The final contribution of this dissertation is to provide a divide-and-conquer algorithm as a first step to addressing this issue.

Acknowledgements

Oh yes, the acknowledgements. I think not. I did it. I did it all, by myself.

Olin Shivers

Clearly not true in my case. Can I really thank all the people that made this five-year journey possible and (mostly) tolerable (sometimes even enjoyable)? As Olin says: I think not. Anyway, to all of you (some of you may even know who you are): Thanks.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

> There are only two mistakes one can make along the road to truth;
> not going all the way and not starting.
>
> Buddha

This dissertation examines the problem of learning a representation (or map) of an environment from a stream of actions and observations. In particular, it formalizes a novel approach—called subjective mapping—aimed at providing an agent with a low-dimensional representation of its environment in which the agent's actions have a sensible interpretation. This approach is based on the intuition that learning a representation of an environment can be phrased as a variation of the dimensionality reduction problem. In addition, this dissertation develops the action respecting embedding algorithm for solving the subjective mapping problem and demonstrates the usefulness of the algorithm for a variety of applications, as well as addressing some scaling issues. This chapter provides an overview of the domain and the problem, explicitly delineates the contributions of this dissertation and provides an outline of the remainder of the document.

1.1 Domain

Imagine the prototypical artificially intelligent agent existing in some environment (for an introduction see for example [RN03]), capable of taking actions and getting observations in return. The agent's data, a stream of actions and observations, is expressed in the form:

$$z_0, u_1, z_1, u_2, z_2, u_3, z_3, \ldots, u_{n-1}, z_{n-1}, u_n, z_n$$

where z_0 is the agent's first observation, u_1 is the first action the agent takes, z_1 is the observation the agent gets after taking the first action, and so forth. Such a sequence (where the agent has taken n actions and made $n + 1$ observations) is a typical data set from an agent which might benefit from learning a useful representation of its environment.

Although one could imagine a version of this model where the actions and observations are arbitrarily complex data structures it will be easier (and sufficiently powerful, for the most part) to restrict them. In this document, action labels are nominal variables represented by integers ($u_i \in \mathbb{Z}_{n_a}$, where n_a is the number of distinct possible actions) while observations are continuous D-dimensional vectors ($z_i \in \mathbb{R}^D$) typically high-dimensional.

For example, imagine a wheeled robot with a mounted camera moving around in a room. In this case the robot is the agent. The agent's actions are to move forward a certain distance or to turn right or left by a certain angle. The observations are the images received from the camera. The data in this example is the interleaved

sequence of action labels (u_i) and images from the camera (z_i). Take the same robot and replace the camera with a laser range-finder and the resulting data will still be in the same format—action labels (u_i) and distance vectors (z_i).

Another example of an agent is ImageBot: a square patch moving around in an arbitrary image. The agent's actions are to move the square in a direction or to rotate the square or to alter the scale of the image. The current observation is the portion of the image covered by the square. Again, the data is an interleaved sequence of action labels and observations. ImageBot is a synthetic domain inspired by robotics applications (complex observations and simply described actions) which will be used as the test domain in this dissertation. The ImageBot domain will be described in detail in Section 3.3.1.

1.2 Problem

The goal is for an agent (as described in Section 1.1) to be able to learn a useful representation of its environment, in other words a map[1] of its environment (or at least the portion it observes). While mapping is often considered a challenge specific to robotics, in its general form it is actually one of the fundamental challenges of artificial intelligence—learning a useful representation of an environment. Such a task is important in robotics, reinforcement learning, planning, multi-agent games, and other areas.

Ideally, an agent would learn a map that could be used by other agents (one

[1]The term map is used in the most common sense of the word, such as a road map or a map of the inside of a building.

that somehow corresponds to an ideal objective map), but in the most general case there may be no requirement that the representation matches a known objective representation (for example, a building floor plan)—it may be sufficient to learn a map that is only useful to the agent alone. In this case, it should be possible to learn a map wholly (or mostly) from the subjective stream of experience (the agent's stream of actions and observations). The use of additional objective information should not be required or, at least, be minimized.

Subjective mapping will be defined (in Chapter 3) by identifying three key properties of a useful representation. It should be a compact representation: distilling large quantities of information. Landmarks in the environment should be in one-to-one correspondence with points in the representation. Finally, actions associated with the environment should correspond to consistent transformations in the representation. Note that the first two properties are the goals of the well-known problem of dimensionality reduction, suggesting that good representations may be obtained from modified dimensionality reduction algorithms.

1.3 Contributions

The contributions of this dissertation are as follows:

1. The introduction and formal definition of a new approach to learning a representation of an environment, here called *subjective mapping*, motivated through the articulation of three key properties of good maps (two of which are also properties of the dimensionality reduction problem).

2. The action respecting embedding (ARE) algorithm for solving the new problem—the input to ARE is a stream of experience, the output a low-dimensional representation of the environment associated with that stream.

3. Results from running the algorithm in a complex robotics-inspired test domain which support the thesis that subjective mapping is a workable approach.

4. A method for learning operators corresponding to original actions in a representation learned by ARE by solving a constrained regression problem.

5. Demonstrations of the empirical usefulness of such representations and operators in both planning (which in turn leads to a quantitative evaluation of the learned representations) and localization tasks.

6. An effective way of scaling the ARE algorithm to handle larger input sizes using a divide-and-conquer technique.

(Items 2 and 3 are a joint contribution; an earlier version appears in Ali Ghodsi's PhD thesis [Gho06] though in a considerably different form.)

1.4 Outline

The subjective mapping problem introduced in Section 1.2 can be thought of as an extension of the problem of dimensionality reduction which consequently will be reviewed in Chapter 2. Chapter 3 expands on the key characteristics of maps, formally defines the problem of subjective mapping and introduces the test domain

of this dissertation. Chapter 4 introduces the Action Respecting Embedding algorithm for subjective mapping. Chapter 5 shows how operators corresponding to actions can be extracted from the learned map and used for planning. Chapter 6 demonstrates localization in a learned representation. Chapter 7 addresses some scaling issues with Action Respecting Embedding. Chapter 8 discusses alternate approaches to mapping (learning representations) and how they relate to the new subjective mapping approach presented in this document. Chapter 9 concludes.

Chapter 2

Background

> To be conscious that you are ignorant is a great step to knowledge.
>
> Benjamin Disraeli

This chapter summarizes a variety of dimensionality reduction concepts. This is necessary as the problem of subjective mapping, which will be developed and formally defined in Chapter 3, is essentially a variation of dimensionality reduction. Additional related work is discussed in Chapter 8, once the original contributions of the dissertation have been presented.

2.1 Dimensionality Reduction

Dimensionality reduction or manifold learning can be seen as the process of deriving a set of degrees of freedom which can be used to reproduce most of the variability of a data set. Consider, for example, a set of images produced by rotating a camera through a range of different angles around an object. Clearly only one degree of

Given a collection of high-dimensional inputs:

$$\{z_1, z_2, z_3, \ldots, z_n\} \quad : \quad z_i \in \mathbb{R}^D$$

compute outputs:

$$x_i \in \mathcal{R}^d \quad : \quad d \ll D$$

in one-to-one correspondence with the inputs.

Table 2.1: Formal definition of dimensionality reduction.

freedom is being altered so the images should lie along a one-dimensional continuous curve through the image space.

A formal definition of the dimensionality reduction problem is provided in Table 2.1. Solving the problem provides the d-dimensional coordinates of the input points on a d-dimensional manifold (or subspace) embedded in the original D-dimensional space. In the rotating camera example where the camera produces 256×256 RGB images (3 color values per pixel), the goal might be to go from $z_i \in \mathbb{R}^{256\times256\times3}$ to $x_i \in \mathbb{R}^1$. If the camera rotates all the way around the object, then two dimensions ($x_i \in \mathbb{R}^2$) may be required in order to capture a circular manifold. Figure 2.1 demonstrates a possible two-dimensional manifold resulting from dimensionality reduction applied to a collection of images of a teapot viewed from different angles.

2.2 Principal Components Analysis

One of the original algorithms for dimensionality reduction was principal components analysis or PCA. PCA provides a sequence of best linear approximations to

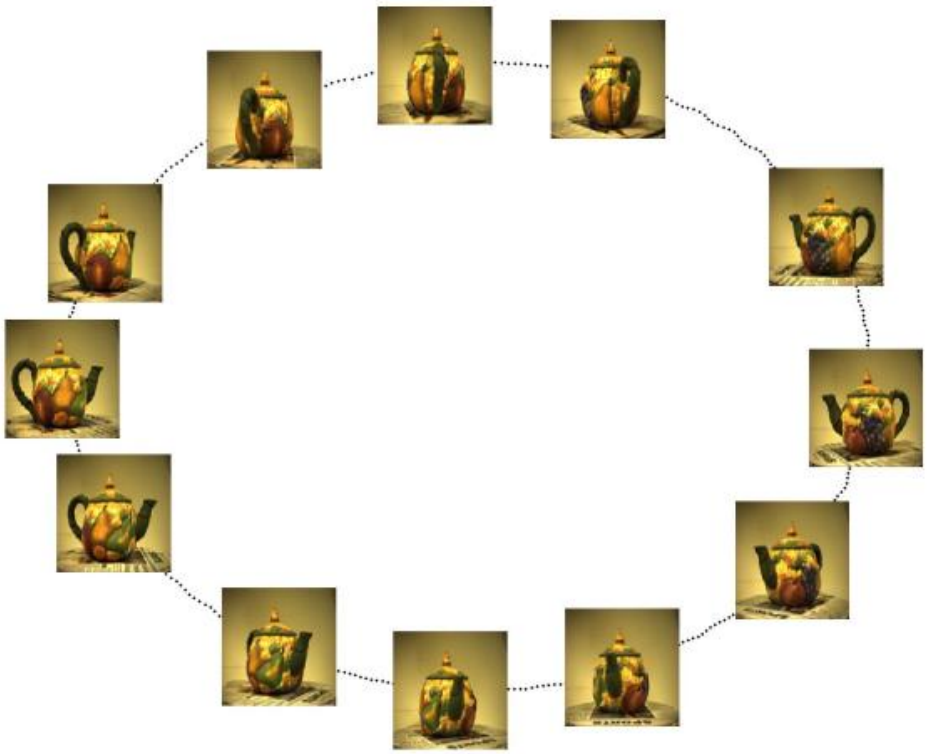


Figure 2.1: Images of a teapot on a 2-dimensional manifold (this example taken from [WS04]).

a given high-dimensional observation. Although popular, its effectiveness is limited by its global linearity.

The subspace output by PCA attempts to capture the maximum variability in the data. Given a data set of n-dimensional vectors, PCA can be seen as a two-step process. The first step is to to find a new orthonormal basis for the vectors where the first dimension captures the most variation, the second dimension captures the second most variation and so on. These new basis vectors are called *principal components* or *PCs*. The second step is to then choose the first d axes

(where preferably $d \ll n$) and project the high dimensional data onto this new lower-dimensional subspace. The linear subspace specified by these d principal components will hopefully capture either all of or the majority of the variation in the data.

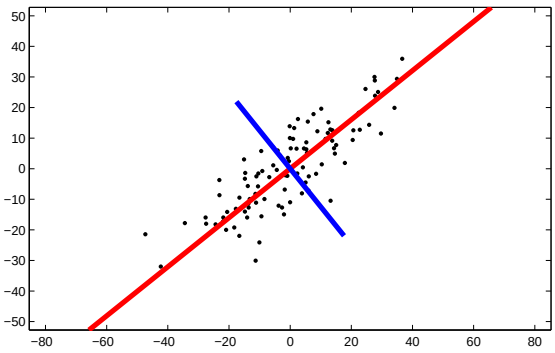


Figure 2.2: A set of 2-dimensional points and their principal components.

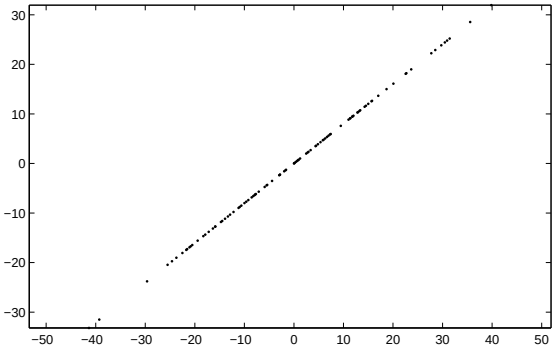


Figure 2.3: Points from Figure 2.2 projected onto the first principal component.

Figure 2.2 shows a collection of two dimensional data points as well as the new 2-dimensional orthonormal basis which maximizes the variance of the data points.

Figure 2.3 shows the result of projecting those 2-dimensional data points down onto the 1-dimensional subspace which corresponds to the first principal component.

There are a number of ways to derive PCA (see for example [Hot33]). Given an $D \times n$ matrix X of centred data $[x_i, i \in 1 \ldots n]$ where each of the n columns corresponds to a D dimensional data vector,[1] the principal components are the orthonormal vectors which maximize retained variance when the data vectors are projected onto them.

In order to capture as much variability as possible, choose the first principal component, U_1, to have maximum variance. This first principal component must be a linear combination of X weighted by coefficients $w = [w_1, \ldots, w_n]$:

$$U_1 = w^T X$$

$$var(U_1) = var(w^T X) = w^T S w$$

where S is the sample covariance matrix of X. Maximizing this with respect to w will yield a solution for U_1.

Since $var(U_1)$ could be made arbitrarily large by increasing the weights, a constrained optimization is necessary:

$$\max_w w^T S w \quad \text{subject to: } w^T w = 1$$

[1] A centred data matrix is one for which the columns have a mean of zero. Any collection of data vectors can be centred without affecting the covariance structure. It is often preferable to work with centred data as there is a straightforward relationship between inner products and the covariance matrix.

This can be reformulated as an unconstrained optimization in the standard way by using a Lagrange multiplier:

$$L(w, \lambda) = w^T S w - \lambda(w^T w - 1)$$

which can be differentiated with respect to w and set to 0 giving:

$$Sw = \lambda w.$$

Multiplying by w^T on the left gives:

$$w^T S w = w^T \lambda w = \lambda w^T w = \lambda,$$

since λ is scalar and $w^T w = 1$ by the constraints. Therefore $var(U_1)$ is maximized if λ is the largest eigenvalue of S.

The first principal component is the eigenvector of S corresponding to the largest eigenvalue of S. Similarly, the second principal component is the eigenvector corresponding to the second largest eigenvalue of S, and so forth. In other words, the first d principal components of a data set X are the d dominant eigenvectors of S, the sample covariance matrix of X.

Since the data vectors are centred, $S = XX^T$ and the eigenvectors (and eigenvalues) of S can be recovered through singular value decomposition of X:

$$X = U \Sigma V^T \tag{2.1}$$

where the columns of U contain the required eigenvectors of $XX^T = S$ and the diagonal of Σ contains the singular values of $XX^T = S$ which can be squared to get the eigenvalues.[2]

2.2.1 Dual Principal Components Analysis

Given n points with D dimensions, XX^T is a $D \times D$ matrix. Typically, $n \ll D$ (as might be the case with a collection of images) therefore there may be substantial savings in computation time when using dual PCA—a version of PCA which requires computation of the eigenvectors of of the $n \times n$ matrix X^TX instead of the $D \times D$ matrix XX^T.

In the singular value decomposition in Equation 2.1 the columns of U are the eigenvectors of XX^T while the rows of V are the eigenvectors of X^TX. Further, the top d eigenvectors in U will be in one-to-one correspondence with the first d eigenvectors in V in the following way. Since V is orthogonal, multiplying Equation 2.1 by V^T on the right yields:

$$XV = U\Sigma V^TV = U\Sigma$$

Since Σ is square and invertible (because it has nonzero entries on the diagonal) one can convert between the top d eigenvectors as follows:

$$U = XV\Sigma^{-1}.$$

[2]This is usually not necessary as the majority of the time the only interest in the eigenvalues is in their ordering which is preserved with the singular values.

This means that the principal components can also be found from the eigenvectors of $X^T X$. Since this dual version of PCA depends on the number of points not their dimensionality it is clearly preferable in situations where D is much larger than n.

2.2.2 Kernel Principal Components Analysis

Kernel PCA is a non-linear generalization of PCA. In kernel PCA, kernels are used to efficiently compute principal components in a high-dimensional feature space that is related to the original input space by some non-linear mapping. Where PCA finds an orthogonal transformation of the coordinate system in which the input data is described, kernel PCA finds a similar orthogonal transformation in a coordinate system which is a non-linear transformation of the space of the input data. A key observation is that dual PCA is formulated in terms of dot products between data points. This means that in a similar formulation of kernel PCA the dot product is replaced by the inner product of points in a Hilbert space. This is equivalent to performing PCA in a higher-dimensional space produced through some non-linear mapping—where hopefully the low-dimensional latent structure is globally linear with respect to this new feature space.

Figure 2.4 shows a collection of 2-dimensional points (marked with squares) with an obvious 1-dimensional manifold—starting from the inner, lighter-coloured points and spiralling out to the outer darker-coloured points. Clearly running PCA on these points will not capture that manifold—the points projected onto the first PC (marked with circles) are not in the desired order. If the points are projected into

the 3-dimensional feature space shown in Figure 2.5 (again, marked with squares) and PCA is run there, then the desired manifold is captured—the points projected onto the first PC (again, marked with circles) are in the same order as the original points along the spiral.

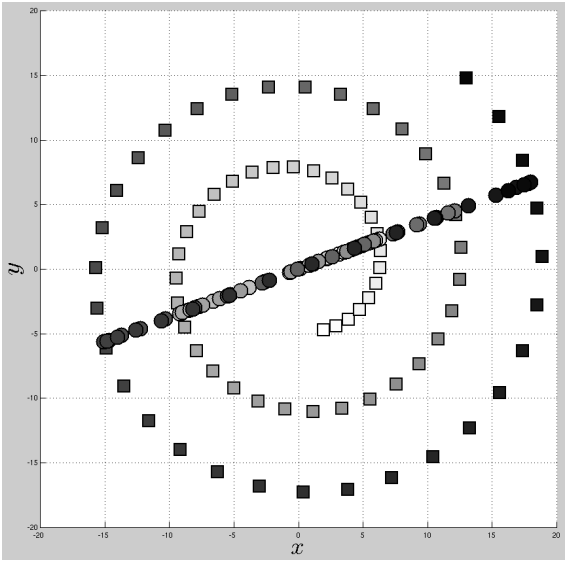


Figure 2.4: 2-dimensional points on a spiral manifold (squares) and their projections (circles) onto the first PC found by PCA.

Kernel PCA finds principal components by performing PCA in a feature space $\mathcal{H}$ where $\Phi : \mathbb{R}^D \rightarrow \mathcal{H}$ maps D-dimensional points x_i to $\mathcal{D}$-dimensional points $\Phi(x_i)$. As before, start with an $\mathcal{D} \times n$ matrix $\mathcal{X}$ of centred[3] data $[\Phi(x_i), i \in 1 \ldots n]$. An analysis similar to that of dual PCA follows. The only place where $\Phi(x)$ appears is in the singular value decomposition of $\mathcal{X}^T\mathcal{X}$ which is called the *kernel matrix* K.

All of the entries in K are of the form $K_{ij} = \Phi(x_i)^T\Phi(x_j)$. In other words it is

[3]Previously, it was argued that a non-centred X could always be converted into a centred X without affecting the structure of the points and hence the principal components. Although more difficult to show, this holds for kernel PCA as well.

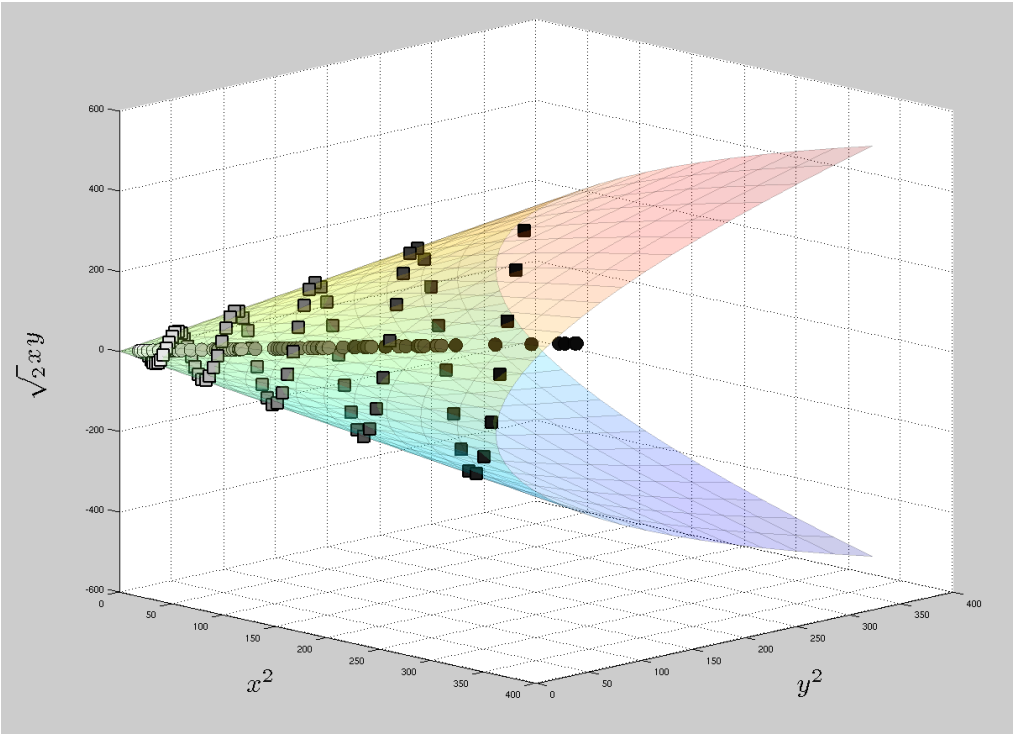


Figure 2.5: Points from Figure 2.4 mapped into 3 dimensional points (squares) and their projections (circles) onto the first PC found by PCA in this new space.

a matrix of inner products in the space $\mathcal{H}$. Note that K is an $n \times n$ matrix—the dimensionality $\mathcal{D}$ of the feature space is no longer a concern. In fact, as long as there is some way to compute the required inner products, $\Phi(x_i)^T\Phi(x_j)$, it may not be necessary to ever compute $\Phi(\cdot)$ itself, or even necessarily know what the mapping is. This is the standard "kernel trick" where any learning algorithm whose solution can be expressed entirely in terms of inner products can be *kernalized* as long as some way of determining corresponding inner products in feature space exists.

The example in Figures 2.4 and 2.5 uses a variation of a polynomial kernel

where:

$$\Phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1y_1 \end{bmatrix}$$

but $\Phi(x)$ never needs to be computed to run PCA. Instead only the inner product needs to be computed:

$$\begin{aligned} \langle \Phi(x), \Phi(y) \rangle &= \left\langle (x_1^2, x_2^2, \sqrt{2}x_1x_2), (y_1^2, y_2^2, \sqrt{2}y_1y_2) \right\rangle \\ &= x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 \\ &= (x_1y_1 + x_2y_2)^2 \\ &= \langle x, y \rangle^2 \end{aligned}$$

which is expressed in terms of the original points.

Kernel methods and their applications in a wide variety of areas have been well studied. A good starting point for more information would be [SS01] or [STC04].

2.3 Nearest Neighbour Methods

With dimensionality reduction, the goal is to map high-dimensional points to low dimensional points on a manifold. If the resulting manifold, when embedded in the higher dimensional space, is non-linear, it may be the case that points that are close in high dimensional space may be quite far apart on the manifold. A majority of dimensionality reduction techniques attempt to preserve some of the distances between high-dimensional points. If all such distances are preserved, then the results

will be similar to PCA: a linear subspace. In fact, there is a linear dimensionality reduction technique called multidimensional scaling (MDS [CC00]) which does exactly this—attempts to find a low-dimensional embedding while preserving all pairwise distances. Although the derivation of MDS is quite different from that of PCA, the resulting embeddings end up being the same. In order to learn non-linear manifolds, only a subset of these distances can be preserved.

Imagine that along with the high-dimensional input points a graph is also provided with a node for each input point and an edge connecting every pair of nodes whose distances should be preserved. This graph is a *neighbourhood graph*, η. Additionally, given the point z_i, let N_i be the set of other points which correspond to nodes connected to the z_i node by an edge; let n_{N_i} be the cardinality of that set; and let $N_i(j)$ be the jth point from that set (where $1 \leq j \leq n_{N_i}$).

There are a variety of methods for building η; the two most popular are:

1. k-nearest-neighbour graphs: each point is connected to the k closest points. Algorithm 1 shows the complete details.

2. r-radius-neighbour graphs: each point is connected to points within distance r.

The majority of methods that require such a neighbourhood graph η require that η be connected. This can be ensured by using appropriately high values for k or r.

Nearest neighbour techniques are prevalent in machine learning. A good source of further coverage would be [DHS01].

Algorithm 1: Building a k-nearest-neighbour graph

Input:
Data points: $Z = \{z_1, z_2, \ldots, z_n : z_i \in \mathbb{R}^d\}$
Number of neighbours: $k \in \mathbb{Z}$ where $k \geq 1$
Distance metric: **distance** $(x, y) \sim \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$
Output: k-nearest neighbour graph η

1 Build neighbour sets (N_i):
for $i = 1 \ldots n$ **do**
  $Z' = Z - z_i$
  Create sequence Y by sorting Z' according to **distance** $(\cdot, z_i)$
  $N_i = \{Y_1, \ldots, Y_k\}$

2 Create graph η with nodes $1 \ldots n$ and edges as follows:
for $i = 1 \ldots n$ **do**
  foreach $z \in N_i$ **do**
    add edge (z_i, z) to η

3 **return** η

2.3.1 Locally Linear Embedding

Locally linear embedding (LLE) [RS00] is an approach to non-linear dimensionality reduction which computes a low-dimensional, neighbourhood preserving embedding of high-dimensional data. A data set of dimensionality D, which is assumed to lie on or near a smooth nonlinear manifold of dimensionality $d \ll D$, is mapped into a single global coordinate system of lower dimensionality, d. The structure of this system is determined by locally linear best fits.

Consider n d-dimensional real-valued vectors x_i sampled from some underlying manifold. Assume each data point and its neighbours are close to a locally linear portion of the manifold. By a linear mapping, consisting of a translation, rotation and rescaling, the high-dimensional coordinates of each neighbourhood can be mapped to global internal coordinates on the manifold. Thus, the nonlinear struc-

ture of the data can be identified by first computing the locally linear patches, then computing the linear mapping to the coordinate system of the manifold.

The main goal here is to map the high-dimensional data points to the single global coordinate system of the manifold such that the relationships between neighbouring points are preserved. This is accomplished by the following three steps:

1. Identify neighbours of each data point x_i (determine η using Algorithm 1).

2. Compute the weights that best linearly reconstruct x_i from its neighbours.

3. Find the low-dimensional vector y_i best reconstructed by those weights.

Given an η in Step 1, Step 2 must compute a locally linear geometry from each local patch. This geometry is characterized by linear coefficients that reconstruct each data point from its neighbours. These weights are obtained by solving:

$$\min_{w} \sum_{i=1}^{n} \left\| x_i - \sum_{j=1}^{k} w_{ij} x_{N_i(j)} \right\|^2 \tag{2.2}$$

Low-dimensional vectors that preserve these weights are obtained by solving:

$$\min_{Y} \sum_{i=1}^{n} \left\| y_i - \sum_{j=1}^{k} w_{ij} y_{N_i(j)} \right\|^2 \tag{2.3}$$

which can be reformulated as

$$\min_{Y} Tr\left(Y^T Y L\right)$$

where

$$L = (I - W)^T (I - W).$$

The solution for Y can have an arbitrary origin and orientation. In order to make the problem well-posed, these two degrees of freedom must be removed. Adding a constraint that the coordinates to be centred ($\sum_i y_i = 0$) and a constraint enforcing unit covariance ($Y^T Y = I$) accomplishes this.

The cost function can be optimized initially with the second of these two constraints. Under this constraint, the cost is minimized when the columns of Y^T (rows of Y) are the eigenvectors associated with the lowest eigenvalues of L. Discarding the eigenvector associated with eigenvalue 0 then satisfies the first constraint.

2.3.2 Isomap

Isomap [Ten98, TdSL00] is a non-linear extension of MDS. The goal of the algorithm is to preserve some pair-wise distances and try to enforce that the remaining distances are close to the geodesic distances.

Isomap takes as input a matrix of distances, D, and first identifies a graph, η, of local distances by finding the k-nearest-neighbours (Algorithm 1). The distances between neighbours are kept the same but non-neighbours have their distances replaced with the length of the shortest path between those two points in the neighbourhood graph. This graph distance is used as an approximation of the true geodesic ("on the manifold") distance between points. In fact, as the density of the points increases, the difference between the actual geodesic and the graph distance provably converges. Once D has been updated, it is converted into a kernel matrix

K which can be used with kernel PCA to obtain an embedding.[4] Algorithm 2 shows the complete details.

Algorithm 2: Isomap

Input:
High-dimensional data points: $Z = \{z_1, z_2, \ldots, z_n : z_i \in \mathbb{R}^D\}$
Number of neighbours: $k \in \mathbb{Z}$ where $k \geq 1$
Distance metric: **distance** $(x, y) \sim \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$
Output: $X = \{x_1, x_2, \ldots, x_n : x_i \in \mathbb{R}^d\}$ in correspondence to Z where $d \ll D$

Create k-nearest-neighbour graph, η (see Algorithm 1). ;
Create geodesic distance matrix D:
if *i and j are neighbours in η* **then**
    $D_{i,j} =$ **distance** (z_i, z_j)
else
    $D_{i,j} = \infty$
for $k = 1 \ldots n$ **do**
    foreach i **do**
        foreach j **do**
            $D_{i,j} = \min\{D_{i,j}, D_{i,k} + D_{j,k}\}$
Create squared distance matrix S where $S_{i,j} = D_{i,j}^2$.
Convert D to kernel matrix $K = -HSH/2$ where H is the centring matrix.
Get X by running kernel PCA with the resulting kernel K.
return X

2.3.3 Maximum Variance Unfolding

The final algorithm to be summarized is maximum variance unfolding (MVU) [WS04, WSS04]. This is a variation of kernel PCA where the kernel matrix is learned from the data. Recall that kernel PCA requires a way of populating the

[4]Note that the K returned by Isomap may not necessarily be positive semidefinite and so may not be a valid kernel (as per Section 2.3.3). Nonetheless, Isomap tends to work well in practice.

entries (inner products) of the kernel matrix. It turns out that if K is positive semidefinite then its entries *must* be inner products of points in some Hilbert space (see for example [STC04]). MVU learns such a kernel matrix by solving a semidefinite program. As long as the semidefinite constraint is satisfied, K will be a valid kernel and PCA can be applied. In addition to the positive-semidefinite constraint, additional constraints are added to enforce that the kernel is centred and that distances and angles between points and their neighbours (according to a neighbourhood graph η) are preserved. In particular, if z_i and z_j are neighbours or share a common neighbour, then the distance between them is preserved:

$$\|\Phi(z_i) - \Phi(z_j)\|^2 = \|z_i - z_j\|^2$$

which can be written in terms of the kernel matrix as:

$$K_{ij} - 2K_{ij} + K_{jj} = \|z_i - z_j\|^2 .$$

The objective function maximizes the trace of K, which represents the variance of the data points in the learned feature space. This objective with the above constraints is MVU: a semidefinite program for learning K. Algorithm 3 shows the complete details.

An attraction of MVU is that it is explicitly phrased as an optimization problem. This means that the algorithm is readily modified by adding new constraints or changing existing constraints. This will be of great benefit in developing the new action respecting embedding algorithm in Chapter 4.

Algorithm 3: Maximum variance unfolding (MVU)

Input:
High-dimensional data points: $Z = \{z_1, z_2, \ldots, z_n : z_i \in \mathbb{R}^D\}$
Number of neighbours: $k \in \mathbb{Z}$ where $k \geq 1$
Distance metric: $\textbf{distance}\,(x, y) \sim \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$
Output: $X = \{x_1, x_2, \ldots, x_n : x_i \in \mathbb{R}^d\}$ in correspondence to Z where $d \ll D$

1 Create k-nearest-neighbour graph, η (see Algorithm 1).
2 Learn K by solving the following semidefinite optimization:

$$\begin{aligned} \max_{K} \quad & Tr\,(K) \\ \textbf{subject to:} \quad & K \succeq 0, \\ & \sum_{ij} K_{ij} = 0, \\ \textbf{and} \quad \forall ij \quad & \eta_{ij} > 0 \vee [\eta^T \eta]_{ij} > 0 \\ \Rightarrow \quad & K_{ii} - 2K_{ij} + K_{jj} = \textbf{distance}\,(z_i, z_j)\,. \end{aligned}$$

3 Get X by running kernel PCA with the resulting kernel K.
4 **return** X

Chapter 3

Problem: Subjective Mapping

> I have an existential map.
>
> It has "You are here" written all over it.
>
> Steven Wright

This chapter develops and formalizes a new problem: subjective mapping. The motivation for this new problem is the generic situation of an arbitrary agent (as described in Section 1.1) whose experience is represented as an interleaved sequence of observations and actions $\langle z_0, u_1, z_1, \ldots, u_n, z_n \rangle$ that the agent has received up through time n. At its essence, the subjective mapping problem is to take such a stream of experience and learn a low-dimensional representation of the environment in which actions have a sensible interpretation.

3.1 Qualities of Good Maps

The approach taken in this dissertation is based on three crucial observations:

1. Maps involve the compression of information.

2. Maps must contain points corresponding to landmarks in the environment.

3. Maps must be useful in the context of an agent's actions.

3.1.1 Compactness and Landmark Correspondence

Maps often provide a compact representation of the environment. In theory, the collection of raw observations itself is a representation of the environment. Practically, for all but the most trivial cases, using raw observations as a representation is often difficult for a variety of reasons—more importantly, trying to extract meaningful information "on the fly" from a collection of raw observations is not tractable in general. Intuitively, if one attempts to make a map of a house, one will wander around collating a massive stream of visual information, memories, etc. then produce a two-dimensional piece of paper which (hopefully) effectively and succinctly represents the house. In other words, the map summarizes a large quantity of information.

Additionally, there must be correspondence between observations (or recognizable landmarks) and points on the map. If it is not possible to associate observations with specific points on a map then it is not possible for the agent to determine its own location in the map. Such a map may serve other purposes, but its usefulness is limited.

These first two properties are exactly the properties of the dimensionality reduction problem reviewed in Chapter 2 and formally stated in Table 2.1: Given a set of n D-dimensional inputs $\{z_1, z_2, \ldots, z_n\}$, find a set of d-dimensional outputs

$\{x_1, x_2, \ldots, x_n\}$ in **one-to-one correspondence** with the inputs such that $\mathbf{d} \ll \mathbf{D}$. However, there is nothing inherent in dimensionality reduction that enforces the third property. In fact, actions do not even play a role in the dimensionality reduction problem description and so do not affect the resulting output.

3.1.2 Action Consistency

The final critical property of a good map is that the positions of the landmarks on the map relative to each other must be sensible in the context of the agent's actions for the map to be useful. To expand on this observation requires clarification of the term "useful" by examining the ways in which one commonly uses a map. Typically, a map is used to answer the following questions:

1. Where am I on the map?
2. Where have I been on the map?
3. Where am I going on the map?
4. How do I get from location A to location B?

The answer to each of these questions is dependent on the set of actions the agent is capable of taking in the environment being mapped. The question "Where have I been?" is really asking for the path on the map which corresponds to the stream of actions that has just been taken. The question "Where am I going?" is really asking where a single action or a particular list of actions lead on the map. Finally, the last question is a query for a list of actions (usually the shortest possible

such list) which, when performed, will result in movement from point A to point B. The answer to the first question may at first seem like a simple correspondence of a position on the map to an agent's position in the space being mapped, but this is intrinsically tied with the actions that the agent has already taken. For example, a road map is useful with respect to a collection of actions related to driving. If one looks at a road map the relevant actions (for example, driving north 50 km) relate in a clear intuitive manner to a corresponding transformation on the map.

In order to answer the above questions, a map must be made in the context of a particular set of actions—the actions in the environment must correspond to well-represented operators on the map. These operators must have the following properties (in order of importance)—they must be:

1. defined,

2. consistent, and

3. simple.

In order for a map to be useful, it must be possible to determine what the effects of an action will be regardless of where one is on the map. This means that the result of an action must be defined for all points on the map. In the road map example it is possible to determine, for any point on the map, the result of driving north 100 meters.[1]

[1]Note that it is actually sufficient for the actions to be defined on some reasonable subset of the map—it is not actually necessary, for example, for the "drive north" action to be defined on the area of a map corresponding to a lake.

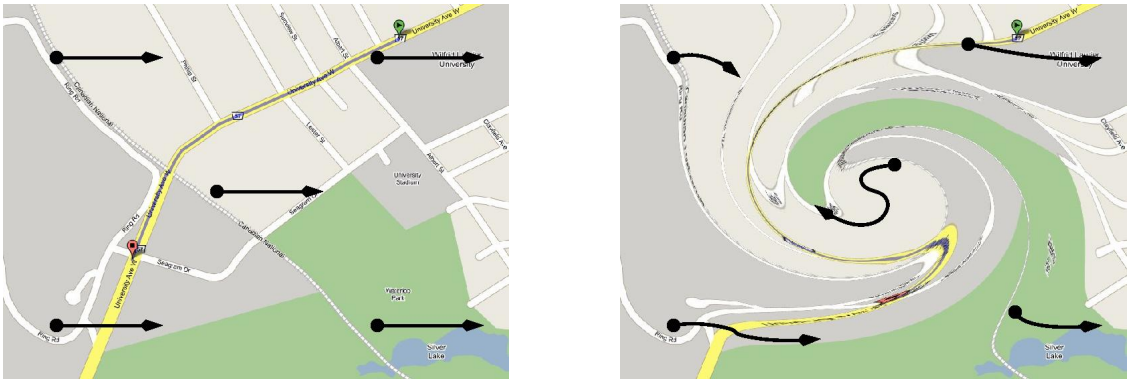


Figure 3.1: Two maps, the left one simpler than the right (original images taken from [Goo07]).

Note that the well-defined property exists for operators that are defined by a lookup table. Arguably, however, such operators are not as useful as it may not be possible to generalize the effects of operators in different parts of the map. It is more useful if the actions on the map are (roughly) the same regardless of where they take place. This is what is meant by the notion of consistency. A road map has this property—regardless of what point is chosen on a map, the "drive north" action corresponds to moving upwards on the map by a fixed amount.

Finally, the simpler the actions are on a map, the easier it is to use the map. For example, examine the two maps in Figure 3.1. On the map on the right, walking east in a straight line corresponds to a complex (but well-defined) spiral-like motion around a particular point. On the map on the left, walking east in a straight line corresponds to a straight translation to the left from the starting point. Arguably, the map with the simpler action is the better of the two. In general, then, maps are preferred where the representations of actions are as simple as possible.

Given sequences of high-dimensional inputs and action labels:

$$\langle z_0, z_1, z_2, \ldots, z_n \rangle \qquad \textbf{and} \qquad \langle u_1, u_2, u_3, \ldots, u_n \rangle$$

where:

$$z_i \in \mathbb{R}^D \qquad \textbf{and} \qquad u_i \in \mathbb{Z}_{n_a}$$

compute outputs:

$$x_i \in \mathbb{R}^d \quad : \quad d \ll D$$

in one-to-one correspondence with the inputs, ensuring that:

$$\forall a \in \mathbb{Z}_{n_a} \quad \exists f_a(\cdot) \in \mathcal{F} \quad : \quad u_i = a \quad \Rightarrow \quad f_a(x_{i-1}) \approx x_i,$$

where $\mathcal{F}$ is a class of functions chosen with some notion of simplicity.

Table 3.1: Formal definition of subjective mapping.

The first property (definition) is a necessity for a good map. If what the actions correspond to on the map is not known then the map is useless. The next two properties (consistency and simplicity) are not strictly necessary but are definitely desirable to some extent.

To summarize, actions in the world should correspond to a transformation on the map. Ideally this transformation should be both consistent, in other words the same, for all parts of the map, as well as simple, in other words the class of transformations should be easily represented and learned.

3.2 Formal Definition of Subjective Mapping

The problem of subjective mapping is formally stated in Table 3.1. The input is a *sequence* of $n + 1$ D-dimensional inputs $\langle z_0, z_1, z_2, \ldots, z_n \rangle$ and a *sequence* of

n action labels $\langle u_1, u_2, u_3, \ldots, u_n \rangle$ where $u_i \in A = \mathbb{Z}_{n_a}$ and n_a is the number of unique possible actions. u_i represents the action taken between observations z_{i-1} and z_i. The desired output is a sequence of d-dimensional points $\langle x_1, x_2, \ldots, x_n \rangle$ where $d \ll D$ *and* for every $a \in A$ there exists a simple function $f_a(\cdot)$ where $u_i = a \Rightarrow f_a(x_{i-1}) \approx x_i$. Note that there are a variety of possible definitions of simplicity which could be used to restrict the choice of f_a. The exact details of this (defined by the $\approx$ relation) are left to individual algorithms for solving this problem.

3.3 Evaluation

Imagine that there is an algorithm to solve the stated problem (to learn a map). Further, imagine that the output of that algorithm, the learned map, has a format similar to the output of standard dimensionality reduction algorithms (a number of low-dimensional points corresponding to a set of high-dimensional observations—the original observations of the agent). How can the learned map be evaluated? This problem is not unique to the subjective mapping situation; it arises with most non-linear dimensionality reduction algorithms. Given a manifold, how can its quality be evaluated? This section briefly addresses this issue.

In many situations, there may be some underlying generative manifold which describes how the agent is moving in the environment. For example, although the collection of images from a camera mounted on a track constitute very high-dimensional data, the actual process that generates that data has but a few parameters: the position on the track and the direction the camera is pointed. It may be

possible, when setting up an experiment, to track these parameters at the same time as the observations are taken. If the points on the learned manifold are structured similarly to those on this true underlying generative manifold, this provides qualitative evidence that the map is good. Alternately, if some clear relationship can be reasoned between the structures of the points on the learned manifold and those on the underlying generative manifold then again qualitatively the map is good. Even if neither of these is the case, it does not necessarily mean that the learned map is not good. Recall that the general problem is to learn a useful "subjective" representation. This means that the structure of the map may not represent anything that can be immediately related to the underlying generative manifold (if it does, then the learned representation would correspond to an objective map which would be good but is more than is required).

In general, these methods of evaluating a subjective mapping algorithm require an experimental framework where the underlying generative manifold is known. This largely motivates the choice of ImageBot as the primary test domain for this dissertation.

3.3.1 The ImageBot Test Domain

This section details ImageBot, the test domain used throughout this dissertation. Given an arbitrary image, imagine a virtual robot that can observe a small, square patch on that image and also take actions to move this observable patch around on the larger image. This "image robot", ImageBot, provides a test domain with ob-

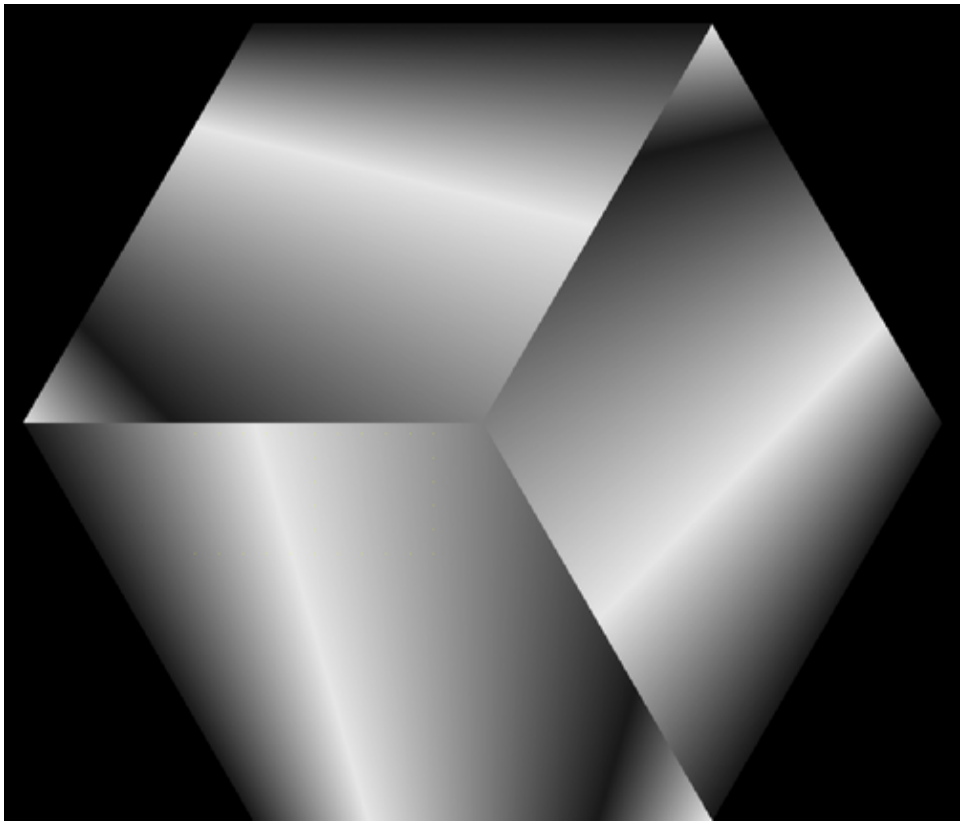


Figure 3.2: A synthetic ImageBot environment (1000×876 pixel image).

vious connections to robotic applications.[2] Compare ImageBot to a wheeled robot with a mounted camera. Both make complex high-dimensional image observations, and both have action sets that can be simply described.

Figures 3.2 and 3.3 show sample images that can be used as ImageBot's "world", one synthetic and one natural. For the experiments in this dissertation, ImageBot is always viewing a 201 by 201 square patch of an image. ImageBot is capable of eight distinct actions: four translation actions, two rotation actions,

[2]Note, ImageBot is similar to the "Roving Eye" domain from [PK97].

Figure 3.3: A natural ImageBot environment (2048×1536 pixel image).

and two zoom (global scaling) actions. The translation actions are "move forward", "move backward", "move left" and "move right" each by 25 pixels. The rotation actions are "turn left" and "turn right", each by 22 and 1/2 degrees ($\frac{1}{16}$ of a circle). The zoom actions are "zoom in" and "zoom out", changing the scale of the image by a factor of $\sqrt[8]{2}$ (in other words eight "zoom in" actions doubles the scale of the image). For the most part, these actions will be precise but the model can be extended to a noisy ImageBot by adding zero-mean Gaussian noise to the magnitude of change of the actions. Unless otherwise stated, the standard deviation of

the added noise will be one tenth of the overall change (that is for the translation actions the standard deviation of the added noise will be 2.5 pixels).

Figure 3.4: Sample experience stream from the environment of Figure 3.3.

Figure 3.4 shows a sample stream of experience that ImageBot might have on the image in Figure 3.3. The stream starts with image z_1, then ImageBot performs discrete action u_1 (in this case the action arbitrarily labelled 1), then its next observation is z_2 and so on. Note that action label 1 clearly corresponds to a shift to the right but the label itself is arbitrary as no semantic meaning should be attached to the action labels.

3.4 Conclusion

Section 3.1 first listed a number of properties of useful maps that could be leveraged for learning, then demonstrated the relationship between two of these properties and the well-studied machine learning problem of dimensionality reduction. Section 3.2 provided a formal definition of a new problem, *subjective mapping*, cast as a variation of dimensionality reduction. Section 3.3 briefly discussed some of the issues involved with evaluating possible solutions to the newly defined problem and introduced a plausible synthetic testbed for testing subjective mapping algorithms.

The remainder of the dissertation will be oriented towards showing that this newly presented problem is a useful paradigm. The first step in this direction is to create a new algorithm for the problem which is the focus of the next chapter.

Chapter 4

Solution: ARE

> The value of an idea lies in the using of it.
>
> Thomas Edison

This chapter introduces the action respecting embedding (ARE) algorithm for solving the problem of subjective mapping introduced in Chapter 3 and formally defined in Table 3.1. The ARE algorithm is an extension of MVU (Algorithm 3 from Section 2.3.3) with two key additions, the usage of a non-uniform neighbourhood graph and the addition of action respecting constraints. ARE is compared with MVU on a variety of test streams from the test domain introduced in Section 3.3.1 in order to demonstrate its effectiveness.

4.1 Action Respecting Embedding

Action respecting embedding takes a sequence of high-dimensional data $\langle z_0, \ldots, z_n \rangle$ along with a sequence of associated discrete actions $\langle u_1, \ldots, u_n \rangle$. Within the data

it is assumed that action u_i was taken between data points z_{i-1} and z_i. The final piece of input is a function **distance** $(\cdot, \cdot)$ defining a distance metric over the high-dimensional data points. Often Euclidean distance is sufficient, but other similarities or metrics could be used.

The overall structure of the ARE algorithm follows that of the MVU algorithm reviewed in Section 2.3.3 and presented in full in Algorithm 3:

1. construct a neighbourhood graph, η
2. solve a semidefinite program to find the maximum variance embedding subject to constraints
3. extract the final embedding from dominant eigenvectors of the learned kernel

ARE exploits the additional information provided by the action labels in two key ways. First, the assumption that two high-dimensional points connected by an action must somehow be near one another allows construction of non-uniform neighbourhood graphs in step one. Second, additional action-respecting constraints are added to the semi-definite program which enforce the actions to be distance-preserving in the resulting manifold.

4.1.1 Non-Uniform Neighbourhoods

As seen before, many current non-linear manifold-learning techniques attempt to preserve local properties of the original data where the notion of locality is defined according to a neighbourhood graph. In most cases (including that of MVU) some variation of a k-nearest neighbour graph is used—the neighbourhood of a point is

determined by the k closest points in the input data set. Since the neighbourhood graph must be connected for MVU to have a bounded solution, the choice of k may be forced to be quite large and may consequently over-constrain the learned manifold. This problem may still arise with the common variation of neighbourhood-graph creation which considers all points within a specified distance threshold r to be neighbours. A key drawback with these techniques is that they require a globally uniform k (or r). However, in the subjective mapping case additional information is given relating the points in the input sequence: certain pairs of data points are connected by an action. This fact can be used to build a more intuitive, non-uniform neighbourhood graph.

The idea is based on the assumption that data points connected by an action are nearby and therefore can be considered neighbours. This definition of closeness is not global, as the distance between consecutive points in one portion of the input stream may differ from the distance between consecutive points in another portion. These assumed neighbours define a "neighbourhood ball" around each data point. The radius of this ball is defined to be just large enough to encompass all other data points explicitly connected by an action. In other words, to begin with the radius of the ball around point z_i is:

$$\max\left(\textbf{distance}\left(z_{i-1}, z\right), \textbf{distance}\left(z, z_{i+1}\right)\right).$$

Now, for any pair of data points, if they are both in each other's neighbourhood ball then add an edge between them in the neighbourhood graph. If desired, the connection density of this neighbourhood graph can be increased by enlarging the

size of the action window: in other words define the radius of a neighbourhood ball to be large enough to encompass all data points connected by a path in the stream of length T. Algorithm 4 has the complete details. Figure 4.1 shows two-dimensional points connected by actions and sample neighbourhood balls for $T = 1$.

Algorithm 4: Building a non-uniform-nearest-neighbour graph

Input:
Stream of experience:
$Z = \langle z_0, z_1, z_2, \ldots, z_n : z_i \in \mathbb{R}^D \rangle$
$A = \{1, 2, \ldots, n_a\}$ where n_a is the number of unique action labels
$U = \langle u_1, u_2, \ldots, u_n : u_i \in A \rangle$
Action window size: $T \in \mathbb{Z} : T \geq 1$
Distance metric: $\textbf{distance}\,(x, y) \sim \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$
Output: non-uniform neighbour graph η

1 Create graph η with nodes $0, 1 \ldots n$;
2 Fill in the adjacency matrix of η as follows:

$$\begin{aligned} \eta_{ij} = 1 \quad \Longleftrightarrow \quad & \exists k, l \text{ such that} \\ & |k - i| < T, |l - j| < T, \\ & \textbf{distance}\,(z_i, z_k) > \textbf{distance}\,(z_i, z_j) \quad \text{and} \\ & \textbf{distance}\,(z_j, z_l) > \textbf{distance}\,(z_i, z_j)\,. \end{aligned}$$

3 **return** η

Note that since the input data is a full sequence of actions, the neighbourhood graph when $T \geq 1$ must be connected. This satisfies a critical requirement that the semidefinite optimization be bounded (or a solution may not exist).

4.1.2 Action-Respecting Constraints

The second contribution of ARE is the addition of action-respecting constraints. Action labels, even without interpretation or implied meaning, should provide some

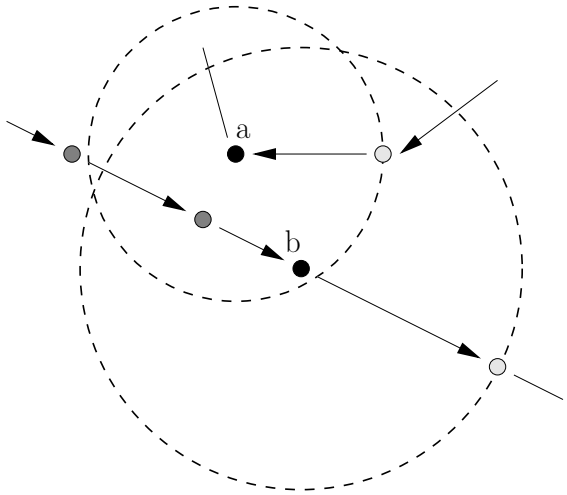


Figure 4.1: The neighbourhood balls for 2-dimensional points a and b. The arrows indicate actions in the original stream of experience. The white points are those which determine the radii of the neighbourhood balls, note that the radii are different (non-uniform). Since a is in b's ball and b is in a's ball, a and b will be connected in the final non-uniform neighbourhood graph.

information about the underlying generation of the data. If there is some underlying low-dimensional space which is generating the data, it is natural to expect that the actions correspond to some simple operator on this generative space's own degrees of freedom. For example, a camera that is being panned left and then right has actions that correspond to simple translations in the camera's actuator space. The goal is to constrain the learned representation so that the labelled actions correspond to simple operators in it.

One possibility would be to enforce actions to correspond to linear transformations in the learned embedding. However, when scaling effects are allowed it can lead to non-intuitive transformations like those on the right in Figure 3.1. Consequently, it may be desirable to enforce the actions to also be non-expansive and

non-contractive. In particular, constrain each unique action label to correspond to a rotation-plus-translation[1] in the low-dimensional representation.

This constraint is formalized by first observing that rotation-plus-translations are the set of *distance-preserving* transformations. A transformation f is distance-preserving (and thus a rotation-plus-translation), if and only if:

$$\forall x, x' \quad \|f(x) - f(x')\| = \|x - x'\|$$

Consider this in the context of an action-labelled data sequence. All actions must be distance-preserving transformations in the learned representation. Therefore, for any two data points z_i and z_j, the same action taken at each data point must preserve the distance between them. Let $\phi(z_i)$ denote data point z_i in the learned subspace, then action u's corresponding low-dimensional operator or transformation f_u, must satisfy:

$$\forall i, j \quad \|f_u(\phi(z_i)) - f_u(\phi(z_j))\| = \|\phi(z_i) - \phi(z_j)\| \tag{4.1}$$

Now, let $u = u_i$ and consider the case where $u_j = u_i$. Then, $f_u(\phi(z_i)) = \phi(z_{i+1})$ and $f_u(\phi(z_j)) = \phi(z_{j+1})$, so Constraint 4.1 becomes:

$$\|\phi(z_{i+1}) - \phi(z_{j+1})\| = \|\phi(z_i) - \phi(z_j)\| \tag{4.2}$$

This constraint on distances can be converted to a constraint on inner products

[1]Rotations-plus-translations are the subset of linear transformations that do not involve scaling.

(in other words a constraint on the learned kernel matrix K) as follows. Squaring both sides of Equation 4.2 gives:

$$\begin{aligned} & \phi(z_{i+1})^T\phi(z_{i+1}) - 2\phi(z_{i+1})^T\phi(z_{j+1}) + \phi(z_{j+1})^T\phi(z_{j+1}) \\ = \;& \phi(z_i)^T\phi(z_i) - 2\phi(z_i)^T\phi(z_j) + \phi(z_j)^T\phi(z_j) \end{aligned}$$

Since $\phi(z_i)^T\phi(z_j) = K_{ij}$ this can be rewritten in terms of K to get the following constraints:

$$\forall i, j \quad u_i = u_j \Rightarrow K_{(i+1)(i+1)} - 2K_{(i+1)(j+1)} + K_{(j+1)(j+1)} = K_{ii} - 2K_{ij} + K_{jj} \quad (4.3)$$

Constraint 4.3 can be added to MVU's usual constraints to arrive at the optimization and algorithm shown in Algorithm 5. There is a slight modification to MVU's usual neighbour constraint, changing strict equality into an upper bound. This ensures that the constraints are feasible and that there will always be a solution to the optimization by allowing the zero matrix to be a feasible solution. Notice that the additional action-respecting constraints are still linear in the optimization variables K_{ij} and so the optimization remains a semidefinite program. Since the neighbourhood graph η_{ij} is connected, the optimization is bounded, convex and feasible, and therefore can be solved efficiently with any one of the various general-purpose toolboxes available. The results in this dissertation are obtained using CSDP in MATLAB [Bor99]. Additionally, these results use a variation of Algorithm 5 with highly-penalized slack variables in MVU's neighbourhood constraint to help improve solution stability, as recommended in the original MVU

paper [WS04].

Algorithm 5: Action respecting embedding (ARE)

Input:
Stream of experience:
$Z = \langle z_0, z_1, z_2, \ldots, z_n : z_i \in \mathbb{R}^D \rangle$
$A = \{1, 2, \ldots, n_a\}$ where n_a is the number of unique action labels
$U = \langle u_1, u_2, \ldots, u_n : u_i \in A \rangle$
Action window size: $T \in \mathbb{Z} : T \geq 1$
Distance metric: $\textbf{distance}\,(x, y) \sim \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$
Output: $X = \langle x_0, x_1, x_2, \ldots, x_n : x_i \in \mathbb{R}^d \rangle$ in correspondence to Z where $d \ll D$

1 Create non-uniform-neighbour graph, η, using Algorithm 4.
2 Learn K by solving the following semidefinite optimization:

$$
\begin{array}{lll}
\max\limits_{K} \quad Tr\,(K) & & \\
\textbf{subject to:} & K \succeq 0, & \\
& \sum\limits_{ij} K_{ij} = 0, & \\
\forall ij & \eta_{ij} > 0 \vee [\eta^T \eta]_{ij} > 0 & \\
\Rightarrow & K_{ii} - 2K_{ij} + K_{jj} \leq \textbf{distance}\,(z_i, z_j)\,, & \\
\textbf{and} \quad \forall ij & u_i = u_j & \\
\Rightarrow & K_{(i+1)(i+1)} - 2K_{(i+1)(j+1)} + K_{(j+1)(j+1)} & \\
= & K_{ii} - 2K_{ij} + K_{jj}. &
\end{array}
$$

3 Get X by running kernel PCA with the resulting kernel K.
4 **return** X.

4.2 Choosing Manifold Dimensionality

The output of the first two steps of ARE (and of MVU) is a kernel matrix. This matrix is then used with the kernel PCA algorithm to generate a low-dimensional

subspace. This subspace has dimensionality equal to the number of data points $n+1$ (n in the case of MVU). If the number of data points is less than the dimensionality of the input observations, then technically dimensionality-reduction has been performed. It may be that this subspace itself will be useful as a map. In general, however, PCA assumes that the data is noisy and that the basis vectors corresponding to the smaller eigenvalues are really only capturing variation according to this noise. Consequently, the final step involves choosing some number of the top eigenvalues and taking the subspace defined by the corresponding principal components (eigenvectors) as the final result. This admits the obvious question of how to choose how many top eigenvalues to take? In other words, how does one choose the dimensionality of the final representation? Two approaches are briefly discussed here.

In some domains, prior information may exist to help answer this question. The existence of a true underlying generative manifold (such as the actual positions after each action of a robot with a mounted camera) may be hypothesized or known and the dimensionality of that manifold may in turn be estimated or known. If the actions in the underlying generative manifold are themselves distance preserving then one would expect to capture them (or a linear transformation of them) with ARE so the estimated dimensionality of the underlying generative manifold is a good choice for the dimensionality of the final learned representation. Even if the actions in the underlying generative manifold are not distance preserving it may still be possible to reason about a corresponding space with only distance-preserving operators that captures the same properties. If so, the dimensionality

of this hypothesized space is a reasonable choice for the dimensionality of the final learned representation.

If no such prior knowledge exists, then another approach, common to a variety of eigenvalue/eigenvector based techniques, is to choose eigenvalues larger than a certain threshold or choose the eigenvalues that are relatively larger than most others. Another method is to sort the eigenvalues from largest to smallest, then select eigenvalues from the start of this list until the sum of the selected eigenvalues is larger than a previously-selected percentage of the total sum. One of the advantages of MVU (and so, presumably, of ARE) is that it tends to have much more distinct drops in relative eigenvalues compared to other methods (such as LLE or Isomap) meaning that this approach is empirically acceptable. More details on the eigenvalue falloff of MVU compared to other methods can be found in the original MVU paper [WS04].

4.3 Results: Learned Representations

Figure 4.2: An ImageBot trajectory—moving 40 steps right, then 20 steps left.

Both MVU and ARE were applied to the ImageBot data from the trajectory in Figure 4.2. As might be expected, the resulting manifold for both algorithms is not surprising—essentially one-dimensional as the first eigenvalue of the resulting kernel dominates the others. Of interest, however, is a plot of the trajectory on

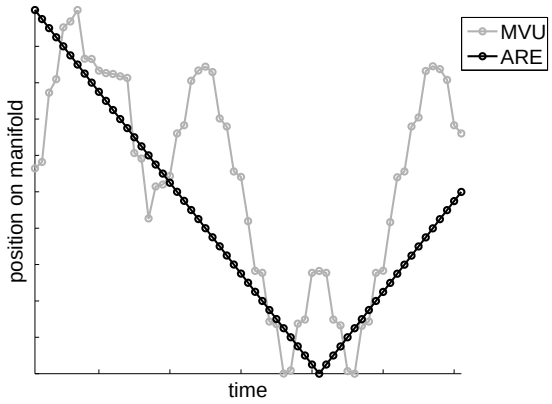


Figure 4.3: A manifold learned from the trajectory in Figure 4.2.

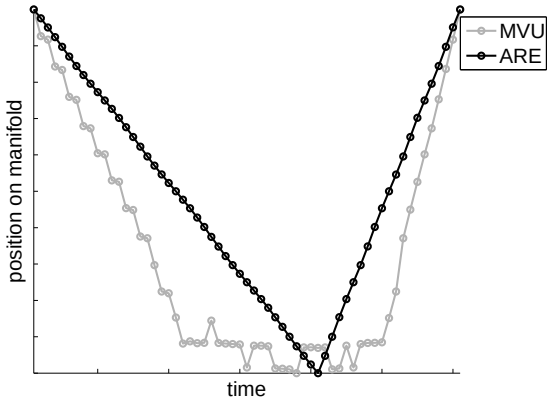


Figure 4.4: A manifold learned from a trajectory similar to Figure 4.2 but with "left" actions twice the magnitude of "right" actions.

this manifold over time, which is shown in Figure 4.3. Note that the result from MVU indicates that ImageBot moved in a jerky fashion and doubled back on itself seven times. The trajectory on the manifold learned by ARE is markedly smoother and corresponds almost exactly to ImageBot's actual trajectory in the underlying 1-dimensional generative manifold. Despite not having any meaning attached to the actions, ARE has clearly managed to learn a representation which captures the essential properties of and relationships between ImageBot's actions—in particular, that the two actions are opposites of each other in terms of direction and have the same magnitude. The actions which generate the data can be subtly changed so that the "backward" action moves twice as far as the "forward" action. The plot in Figure 4.4 demonstrates that ARE is capable of learning a manifold that can capture this property as well.

ARE can correctly handle periodic actions, such as rotation, as well. Figure 4.5

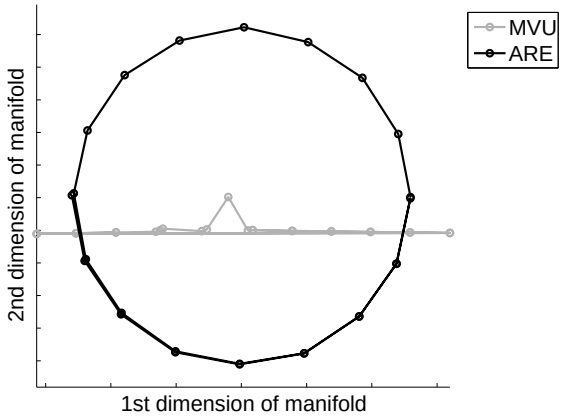


Figure 4.5: A manifold learned from an ImageBot rotation trajectory.

shows the first two dimensions of a manifold corresponding to a trajectory consisting of sixteen "rotate right" and eight "rotate left" actions. ARE essentially discovers the representation $(\sin(\theta), \cos(\theta))$ and also captures the fact that the actions are opposites and periodic.

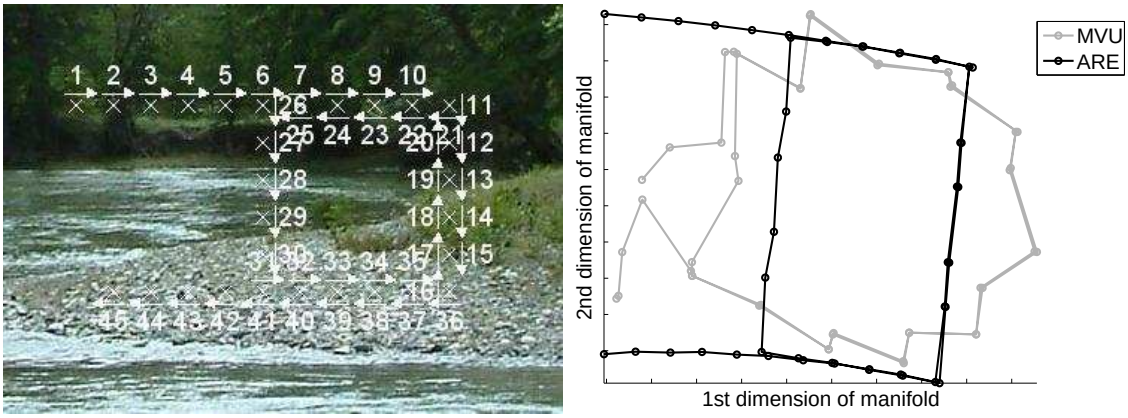


Figure 4.6: A more complex "A"-shaped trajectory and the resulting manifolds.

ARE continues to yield good results in the face of more complicated collections of transformations. ARE and MVU were both run with the more complex trajec-

tory shown on the left in Figure 4.6. The resulting manifolds are displayed on the right in Figure 4.6. MVU, as with the previous example, fails to generate a manifold in which the actions have a simple interpretation. Notice that again, ARE's manifold has a strong correspondence with ImageBot's actual trajectory. ARE again captures the relationships between the "forward" and "back" actions as well as the "right" and "left" actions. More impressive still, the manifold captures the independence and orthogonality of the "forward" and "back" pair of actions with the "right" and "left" pair of actions. Again this is possible despite the fact that none of this information was explicitly coded in the problem input.

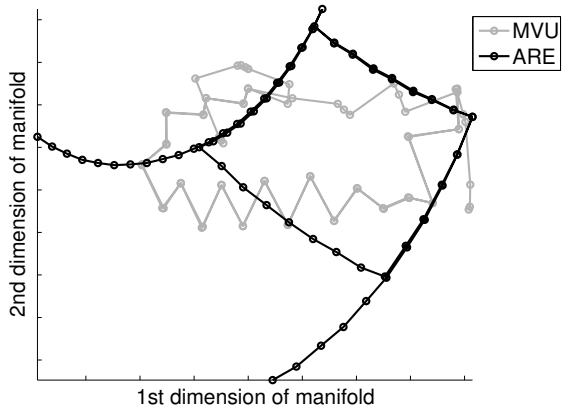


Figure 4.7: A manifold learned from an ImageBot trajectory with zoom actions.

In the example in Figure 4.7, ImageBot follows a variation of the "A"-shaped trajectory. Instead of the "forward" and "back" actions, ImageBot uses the "zoom in" and "zoom out" and the number of steps while zoomed in is doubled. In this case it is no longer true that the two pairs of actions— "right/left" and "zoom in/zoom out"— are independent, as the distance ImageBot moves when implementing the first pair is dependent on ImageBot's zoom level. Nonetheless, as

Figure 4.7 demonstrates, ARE again learns a manifold that captures this relationship. The left leg of the "A" corresponds to observations gathered after ImageBot zoomed in while the right leg of the "A" corresponds to observations gathered while ImageBot was zoomed out. In particular, note that the distance between consecutive points is less on the left leg than on the right, successfully capturing the radial relationship between the two sets of actions.

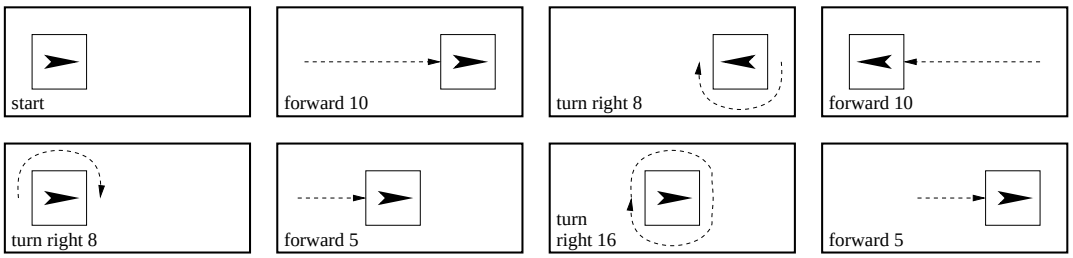


Figure 4.8: An ImageBot trajectory with movement and rotation.

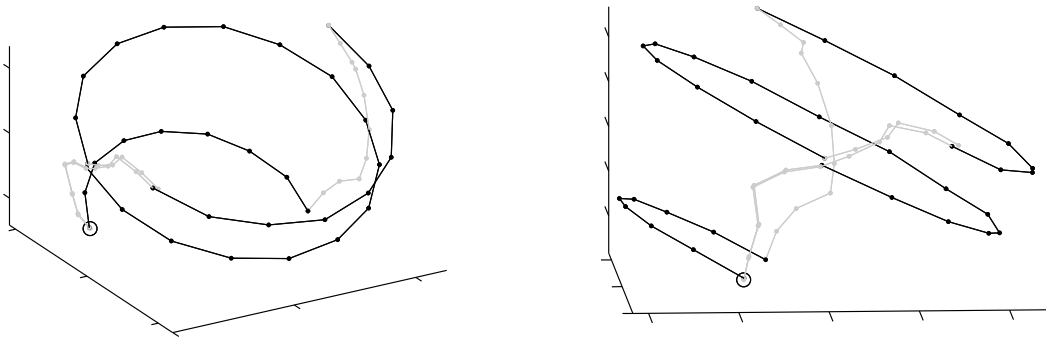


Figure 4.9: Two views of a manifold learned from the trajectory in Figure 4.8. The black line corresponds to the "turn right" actions while the grey line corresponds to the "move forward" actions.

Figure 4.8 shows an even more complicated trajectory. In this case, ImageBot only moves forward and right. From the starting point, ImageBot moves forward

ten steps, then turns right eight times (an about face), then moves forward another ten steps, then turns right eight times (back to the initial position), then moves forward five steps, turns right sixteen times (in a full circle), then moves forward five steps. Figure 4.9 shows the results of ARE on this new trajectory. With the previous examples, it is clear that the manifolds learned by ARE are similar in structure to the underlying trajectory that ImageBot followed. This is no longer the case; however, it is possible to determine the relationship between the learned manifold and the original trajectory (as discussed in Section 3.3). Figure 4.9 shows a "top" view and a "side" view of the three-dimensional manifold learned from the trajectory in Figure 4.8. Here, the portion of the manifold corresponding to rotating to the right is a black line while the portion corresponding to moving forward is a light-gray line. The first observation (the starting point) is circled. The manifold distinctly captures the structure of the original path, with two dimensions capturing the rotation action and a third capturing the forward action. In Figure 4.6, the original domain was one in which the actions were distance preserving so it is unsurprising that the exact structure was, indeed, extracted. For the trajectory in Figure 4.8 it is not immediately obvious what manifold will be learned which makes the results in Figure 4.9 all the more impressive.

All of the results shown here were generated using Euclidean distance as the distance metric, but recall that any metric could be used. It should be easy to obtain similar, or possibly better, results using, for example, an image-specific colour-histogram-based metric.

Up until now, the results have demonstrated that ARE can learn a qualita-

tively good representation given a stream of experience. It remains to demonstrate the robustness of the ARE algorithm. Figures 4.10 and 4.11 show representations learned by ARE on a number of "A"-shaped trajectories (similar to that in Figure 4.6) which differ only in starting position. In Figure 4.10, ImageBot was run with the image from Figure 3.3. In Figure 4.11, ImageBot was run with the image from Figure 3.2. In both cases, roughly half of the learned representations clearly correspond to the underlying "A"-shaped trajectory.

Interestingly, in almost all situations where ARE fails to capture a reasonable manifold the neighbourhood graph is broken. The two obvious ways that this can occur are when points are declared neighbours when they should not be or when points that should be neighbours are not. The former case is perhaps reflective of the problem of perceptual aliasing. For example, consider the environment in Figure 3.3: any observation of the sky alone is likely to be quite close in Euclidean distance to any other observation, even if the corresponding ImageBot positions are far apart. Trying to learn a representation where such points are superimposed seems doomed to failure. This problem is generally quite difficult and while it poses interesting research questions it is not addressed further in this dissertation. The latter case, while potentially damaging, is not necessarily as dire. If enough structure is somehow captured in another part of the neighbourhood graph there may still be enough to overcome the lack of structural information in the broken part of the graph. Intuitively, the first case involves disinformation about the environment while the second represents missing information. Missing constraints, while not ideal, is potentially much less damaging than incorrect constraints.

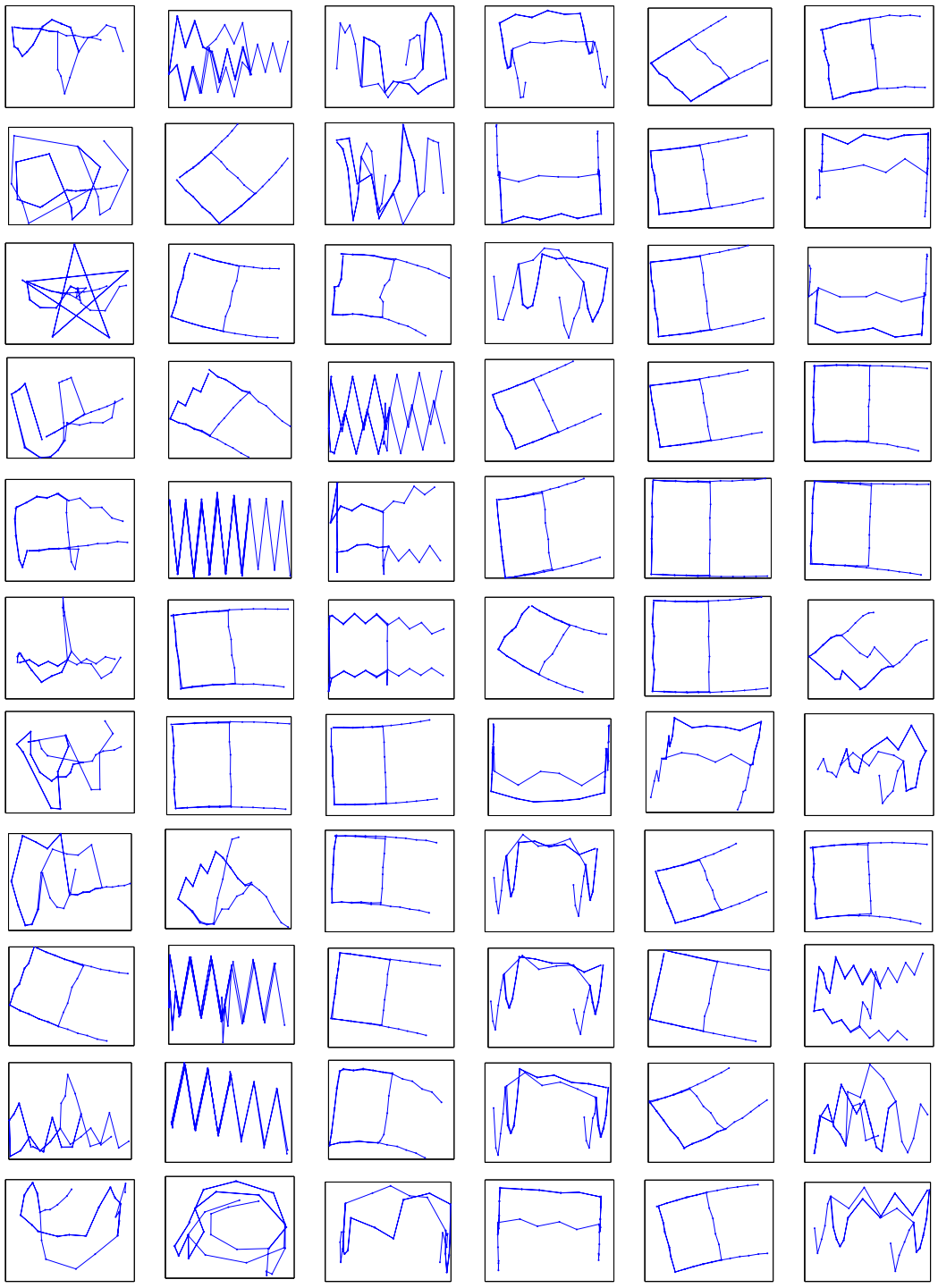


Figure 4.10: Multiple results in the non-synthetic environment of Figure 3.3.

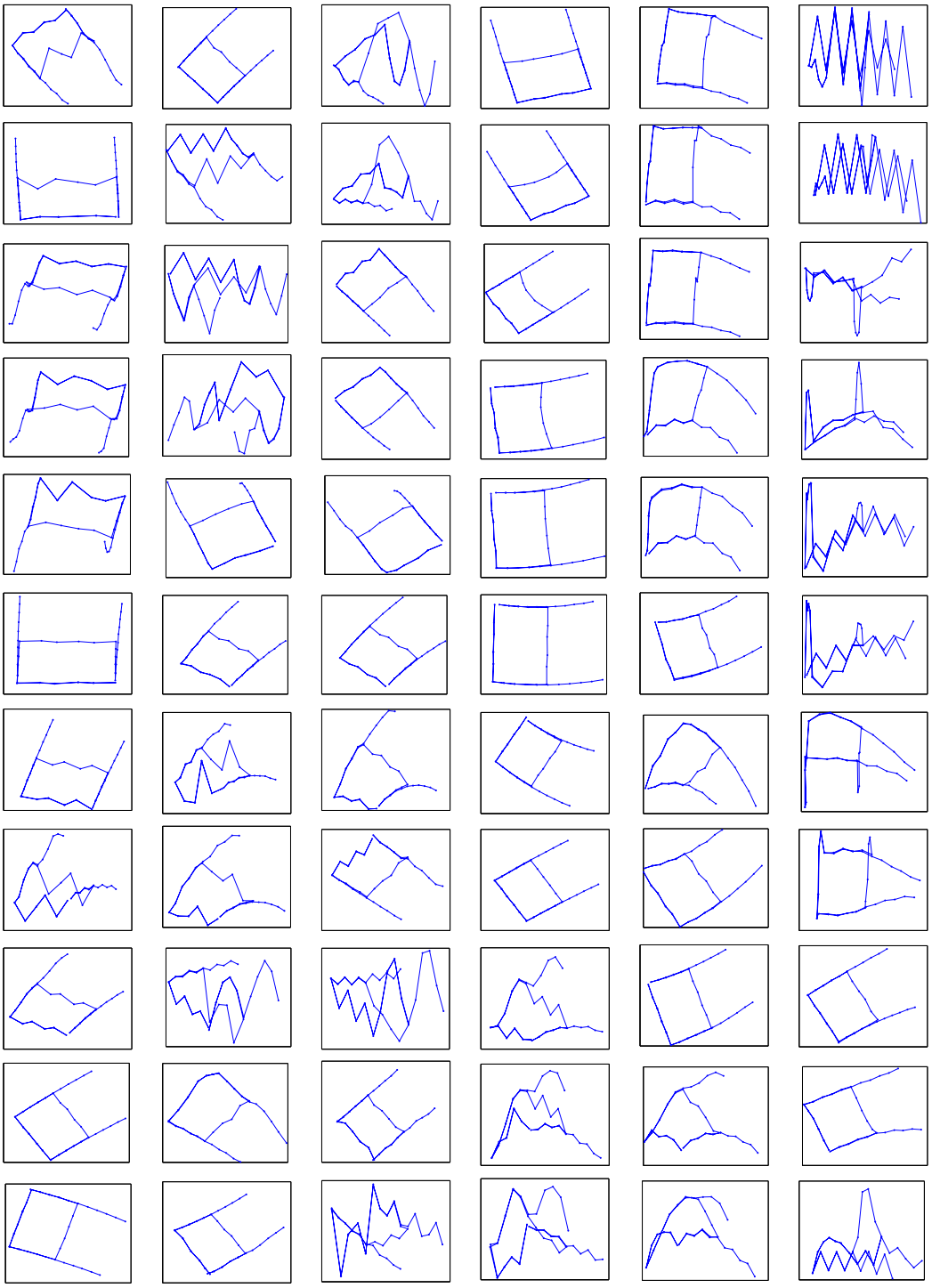


Figure 4.11: Multiple results in the synthetic environment of Figure 3.2.

4.4 Conclusion

Section 4.1 carefully developed an algorithm called action respecting embedding (ARE) for the problem of subjective mapping introduced in Chapter 3 and formalized in Section 3.2. ARE builds on the MVU algorithm for dimensionality reduction summarized in Section 2.3.3 with two important distinctions. The first is the use of the non-uniform neighbour graph, introduced in Section 4.1.1, instead of the traditional nearest-neighbour method used by MVU and described in Section 2.3. The second is the critical addition of the constraints developed in Section 4.1.2 which enforce that the resulting low-dimensional manifold is a space in which the actions correspond to distance-preserving transformations. Section 4.2 briefly discussed the issue of choosing the dimensionality of the final result. Section 4.3 demonstrated the results of ARE on a variety of different streams of experience from the test domain described in Section 3.3.1.

This chapter demonstrated the effectiveness of ARE in learning representations in the ImageBot domain that were similar to the underlying generative manifolds. It is important to note that ARE is able to extract properties of the underlying actions that were not explicitly coded in the input. The results provide solid qualitative evidence that subjective mapping is a useful approach to learning a representation of an environment. Further, ARE greatly outperformed MVU in learning the representations, supporting the key intuition that while dimensionality reduction is necessary, alone it is insufficient for capturing environmental dynamics. The next step in demonstrating the usefulness of the representations learned by the ARE algorithm is to determine the effects of actions in those representations.

Chapter 5

Actions and Planning

> Adventure is just bad planning.
>
> Roald Amundsen

This chapter details the recovery of operators in the representations learned in Chapter 4 corresponding to the original actions in the environment. Once these operators are found they can be used with a representation to plan by finding paths between landmarks. The effectiveness of a learned representation and operators for planning provides a method for quantitative evaluation of subjective mapping solutions.

5.1 Recovering Distance-Preserving Actions

ARE learns a representation with explicit constraints that the actions correspond to distance-preserving transformations in that representation. The algorithm does not, however, give those transformations; it only ensures that they exist. To increase

the usefulness of the representation, these transformations need to be extracted.

For each unique action a there is a collection of data point pairs (x_t, x_{t+1}) which are connected by that action. Put another way: there is a function f_a where $f_a(x_t) \approx x_{t+1}$, and such a function needs to be learned for each action. Because of the distance-preserving constraints, f_a can be represented as:

$$f_a(x_t) = A_a x_t + b_a = x_{t+1}$$

Recall that transformations of the above form encode translation in the b_a vector, and rotation and scaling in the A_a matrix. A_a and b_a could be learned using simple linear regression but scaling is not distance preserving so there is the additional constraint that A_a does not scale, in other words $A_a^T A_a = I$. It turns out that this is similar to the extended orthonormal Procrustes problem [SC70], but without allowing for a global scaling constant. This section derives the solution to the regression problem.

Let X_a be the $d \times n$ matrix whose columns are x_t for all t such that $a_t = a$, and let Y_a be the $d \times n$ matrix whose columns are x_{t+1} for the same t. The goal is to learn a rotation matrix A_a and a translation vector b_a which maps X_a to Y_a. Formally, the following optimization problem needs to be solved:

$$\begin{aligned} \min_{A_a, b_a} \quad & \|A_a X_a + b_a e^T - Y_a\| \\ \textbf{subject to:} \quad & A_a^T A_a = I \end{aligned}$$

where e is a column vector of n ones.

Recall that: $\|A\|_F = \sqrt{Tr\,(AA^T)}$: but since A must be square $A^TA = AA^T$ so the optimization can be rewritten as:

$$\begin{aligned} \min_{A_a,b_a} \quad & Tr\left((A_aX_a + b_ae^T - Y_a)^T(A_aX_a + b_ae^T - Y_a)\right) \\ \textbf{subject to:} \quad & A_a^TA_a = I. \end{aligned}$$

The corresponding dual function is:

$$D(L) = \min_{A_a,b_a} \quad L(A_a, b_a, \Lambda) \tag{5.1}$$

where L is the Lagrangian function:

$$L(A_a, b_a, \Lambda) = Tr\left((A_aX_a + b_ae^T - Y_a)^T(A_aX_a + b_ae^T - Y_a)\right) + Tr\left(\Lambda(A_a^TA_a - I)\right)$$

and Λ is a matrix of Lagrangian multipliers. The first part expands as follows:

$$\begin{aligned} & Tr\left((AX + be^T - Y)^T(AX + be^T - Y)\right) \\ = \;& Tr\left((X^TA^T + eb^T - Y^T)(AX + be^T - Y)\right) \\ = \;& Tr(X^TA^TAX + X^TA^Tbe^T - X^TA^TY + \\ & \quad eb^TAX + eb^Tbe^T - eb^TY + \\ & \quad\quad -Y^TAX - Y^Tbe^T + Y^TY) \\ = \;& Tr\left(X^TA^TAX\right) + \underline{Tr\left(X^TA^Tbe^T\right)} - \underline{Tr\left(X^TA^TY\right)} \\ & \quad +Tr\left(eb^TAX\right) + Tr\left(eb^Tbe^T\right) - Tr\left(eb^TY\right) \\ & \quad\quad +Tr\left(-Y^TAX\right) - \underline{Tr\left(Y^Tbe^T\right)} + Tr\left(Y^TY\right) \end{aligned}$$

Since $Tr\left(A\right) = Tr\left(A^T\right)$ some terms (underlined) can be rewritten:

$$\begin{aligned} & Tr\left(X^T A^T AX\right) + \underline{Tr\left(eb^T AX\right)} - \underline{Tr\left(Y^T AX\right)} \\ & +Tr\left(eb^T AX\right) + Tr\left(eb^T be^T\right) - Tr\left(eb^T Y\right) \\ & +Tr\left(-Y^T AX\right) - \underline{Tr\left(eb^T Y\right)} + Tr\left(Y^T Y\right) \end{aligned}$$

Note that $Tr\left(ebTbeT\right) = nbTb$ so the simplified Lagrangian is:

$$\begin{aligned} L(A_a, b_a, \Lambda) &= Tr\left(Y_a^T Y_a\right) + Tr\left(X_a^T A_a^T A_a X_a\right) + nb_a^T b_a \\ & -2Tr\left(Y_a^T A_a X_a\right) - 2Tr\left(eb_a^T Y_a\right) + 2Tr\left(eb_a^T A_a X_a\right) \\ & +Tr\left(\Lambda(A_a^T A_a - I)\right) \end{aligned}$$

The solution to the dual function in Equation 5.1 can be found by taking the derivatives of L with respect to the unknowns and setting them to zero:

$$\frac{\delta L}{\delta A_a} = 2A_a X_a^T X_a - 2Y_a X_a^T + 2b_a e^T X_a^T + A_a(\Lambda + \Lambda^T) = 0 \tag{5.2}$$

$$\frac{\delta L}{\delta b_a} = 2nb_a - 2Y_a e + 2A_a X_a e = 0 \tag{5.3}$$

Solving Equation 5.3 for the translation vector gives:

$$b_a = \frac{Y_a e - A_a X_a e}{n} \tag{5.4}$$

Reordering Equation 5.2 and multiplying by $\frac{A_a^T}{2}$ on the right gives:

$$\begin{aligned}\left(2A_aX_a^TX_a + A_a(\Lambda+\Lambda^T)\right)\frac{A_a^T}{2} &= \left(2Y_aX_a^T - 2b_ae^TX_a^T\right)\frac{A_a^T}{2}\\ A_aX_a^TX_aA_a^T + A_a(\Lambda+\Lambda^T)\frac{A_a^T}{2} &= Y_aX_a^TA_a^T - b_ae^TX_a^TA_a^T\end{aligned}$$

Since the left hand side is symmetric, the right hand side must also be symmetric. Substituting in Equation 5.4, the right hand side can be written as:

$$\begin{aligned}& Y_aX_a^TA_a^T - \left(\frac{Y_ae - A_aX_ae}{n}\right)e^TX_a^TA_a^T\\ =\ & Y_aX_a^TA_a^T - \left(\frac{Y_aee^TX_a^TA_a^T - A_aX_aee^TX_a^TA_a^T}{n}\right)\\ =\ & Y_aX_a^TA_a^T - \frac{Y_aee^TX_a^TA_a^T}{n} + \frac{A_aX_aee^TX_a^TA_a^T}{n}\\ =\ & Y_aX_a^TA_a^T - Y_a\left(\frac{ee^T}{n}\right)X_a^TA_a^T + A_aX_a\left(\frac{ee^T}{n}\right)X_a^TA_a^T\end{aligned}$$

where the last term is symmetric. Thus the rest of the expression must also be symmetric. This can be simplified as:

$$\left(Y_a\left(I - \frac{ee^T}{n}\right)X_a^T\right)A_a^T \tag{5.5}$$

Since Equation 5.5 is symmetric, it should be equivalent to its transpose:

$$\left(Y_a\left(I - \frac{ee^T}{n}\right)X_a^T\right)A_a^T = A_a\left(Y_a\left(I - \frac{ee^T}{n}\right)X_a^T\right)^T \tag{5.6}$$

Now find the singular value decomposition of the second term on the right:

$$VSW^T = Y_a \left(I - \frac{ee^T}{n} \right) X_a^T$$

and set the rotation matrix to be:

$$A_a = VW^T \tag{5.7}$$

Note that $A_a^T A_a = I$ as required. Equation 5.6 can then be rewritten as:

$$VSW^T WV^T = VW^T WS^T V^T$$

which is clearly true because $W^T W = I$ (since W is orthonormal) and $S = S^T$ (since S is diagonal). Therefore Equations 5.4 and 5.7 are a solution to the dual function in Equation 5.1. The dual optimization problem now is:

$$\max_{\Lambda} D(L)$$

but $D(L)$ is constant with respect to Λ so the solution, $D^\star$, is easily computed by plugging the derived A_a and b_a into L. Now compute P by plugging the derived A_a and b_a into the primal objective. Clearly $P = D^\star$ (the extra term in L is zero for that A_a and b_a) and P is feasible (since the constraint on A_a is satisfied). By weak duality $P = D^\star \leq P^\star$ where $P^\star$ is the optimal solution to the original optimization therefore P must be an optimal solution. The derived A_a and b_a combine to provide the operator on the learned representation corresponding to the original action a.

5.1.1 Results: Learned Transformations

Figure 5.1 shows the 2-dimensional representation generated by ARE in Figure 4.6. The bold arrows show the result of applying each action from an arbitrary point on the manifold. Clearly, the operators are capturing the essence of the actions used to generate the data.

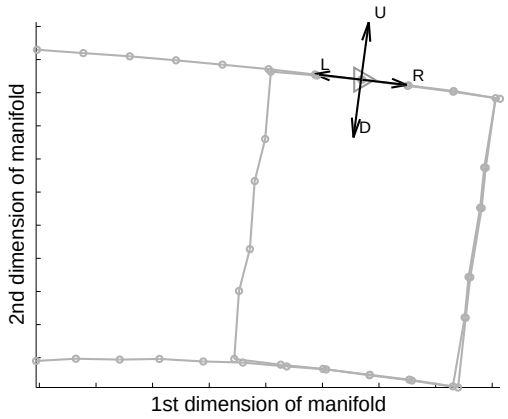


Figure 5.1: Learned operators plotted on the manifold from Figure 4.6.

Figure 5.2 shows the representation from Figure 4.7 where zooming actions replaced the forward and backward actions. Again, the bold arrows show the result of applying each action from a point on the manifold. Again the operators are capturing the dynamics of the environment; note in particular that the scale of the left and right actions is now clearly dependent on the zoom level.

5.2 Planning

Now that a low-dimensional representation and operators in that representation have been learned, all the pieces are in place to actually use the representation for

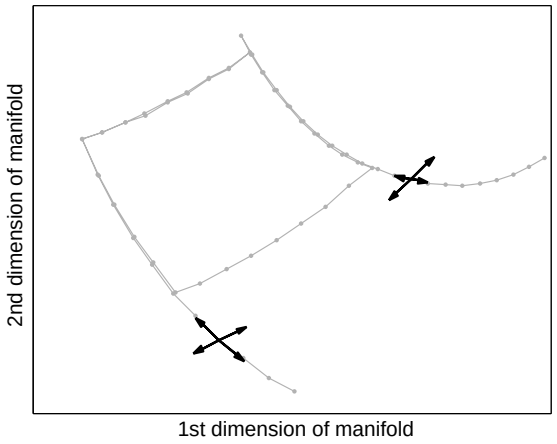


Figure 5.2: Learned operators plotted on the manifold from Figure 4.7.

planning. The points on the learned representation are states or locations on the map, and the operators recovered by the method of Section 5.1 define transitions between those states or locations. Furthermore, this new domain has two advantages over the original data set. First, the dimensionality has been reduced drastically (from $121,203$ to 2). Second, the functions learned in the new domain are simpler than the corresponding functions in the high-dimensional domain. Learning such functions in the original input space would most likely be intractable, or at the very least require knowledge specific to ImageBot and its underlying image.

To begin with, note that some of the questions that maps are traditionally used for can now be answered, namely: "Where will I be if take a particular action" (Figure 5.1) or a particular set of actions (determined by successive application of the appropriate operators). Further, while there is no semantic meaning given with the input action labels, such meaning can now be derived. It can now be quantitatively tested whether a pair of actions are opposites, such as the "forward"

and "backward" actions or the "left" and "right" actions. Also, any two actions can be tested for orthogonality or independence (such as the "forward" action and the "right" action). If the learned representation captured some underlying structure within the data, the learned operators will maintain that structure and therefore relationships can be successfully hypothesized.

Further, given any two high-dimensional data points from the input data set (images, in the case of ImageBot), the shortest sequence of actions can be found that will get from the state corresponding to the first image to the state corresponding to the second. First, find the corresponding points in the low-dimensional representation. Next, find the shortest path between them using traditional search methods and the set of learned operators. Since each operator in the low-dimensional space corresponds to an action label in the original input data, the list of action labels corresponding to the desired path can be returned. For the results in Sections 5.2.1 and 5.3, iterative-deepening depth-first search was used and the path whose final point was closest to the desired goal was returned The quality of the path can be demonstrated by starting ImageBot at the initial state and applying the sequence of actions.

5.2.1 Results: Planning

Figure 5.3 shows an example of a path found by the described algorithm. The initial state is indicated by a triangle pointing to the right and the desired goal state is indicated by a triangle pointing to the left. The list of applied operators corresponds exactly to an actual list of actions that would bring ImageBot from

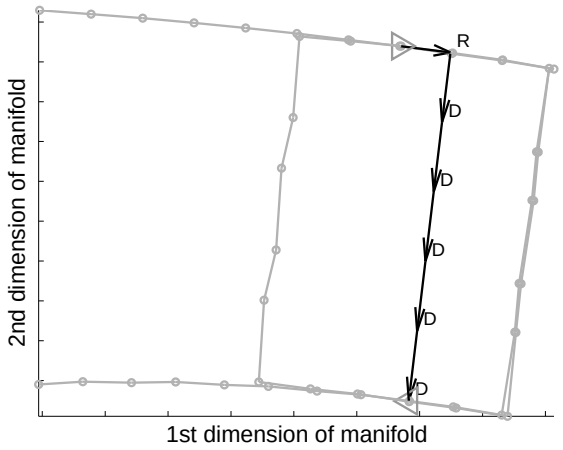


Figure 5.3: A plan on the manifold from Figure 4.6.

the initial state to the desired goal state. Even more impressive is that the path was successfully found even though it involves moving through portions of the space that ImageBot never actually observed—the learned map has captured enough of the structure of ImageBot's interaction with its environment to generalize the effects of ImageBot's actions throughout the space. The applications of successfully being able to plan in this manner are numerous. At the very least, the final common map question, "How do I get from location A to location B?", has been answered.

5.3 Evaluation of Learned Representations

The goal of subjective mapping is to learn a *useful* representation, and until now the method of evaluation has been solely qualitative. Now that the framework for planning is in place, it can be used to provide a quantitative evaluation of a learned representation.

	Mean Distance	Mean Goal Difference	% Exact
ARE	133.5	1.5	94
MVU-d	133.5	85.0	16
MVU-l	133.5	105.0	2

Table 5.1: Quantitative evaluation based on planning performance.

Any potential plan can be tested by running another instance of ImageBot with the plan's action sequence. Ideally, ImageBot's position after running the plan should be the same as the position corresponding to the goal state. The distance between the goal and result positions can be used as a metric of how successful the plan was. Repeated measures of this metric for a collection of random path-finding problems provides a quantitative measure of the usefulness or goodness of a learned representation.

For the learned representation from Figure 4.6, 50 random pairs of points were chosen and a shortest path between those points was learned as in Section 5.2. For the purposes of comparison, the same method was used to find the shortest path in the representation learned by MVU (using best-fit linear transformations as operators (MVU-l) and best-fit distance-preserving transformations as operators (MVU-d)). In each case, the resulting path was then run on ImageBot. The distance between the predicted goal and the actual goal in the underlying generative manifold is recorded along with the length of the path and whether the prediction is exactly correct. The results are presented in Table 5.1. Unsurprisingly, the performance of this task is much better when using representations learned by ARE as opposed to those learned by MVU, providing quantitative evidence of success to accompany the qualitatively good visualizations provided so far.

5.4 Conclusion

Section 5.1 provided a method to recover a transformation for each possible action from a representation learned by the ARE algorithm introduced in Chapter 4. Further, Section 5.1.1 showed the application of this method to the specific representations learned in Section 4.3. Section 5.2 demonstrated how the learned representations and the recovered transformations can be used for a planning task. Section 5.3 showed how the effectiveness of the resulting plans can be used to provide a quantitative evaluation of a learned representation.

This chapter demonstrates how the representations learned in Chapter 4 can be used for planning. Operators which capture the dynamics of the actions can be recovered in the new space. These operators combined with the learned representations can be used with simple search procedures to generate sequences of operators corresponding to sequences of actions which achieve goals in the original environment. It is important to note that the resulting plans have high accuracy, even though they can involve sequences of actions passing through unobserved portions of the environment.

Chapter 6

Subjective Localization

> We're not lost.
>
> We're locationally challenged.
>
> John M. Ford

This chapter demonstrates subjective localization with the representations learned by ARE in Chapter 4 and the operators extracted in Chapter 5. In the mobile robotics domain (a main motivation of this work) difficulties do not end with the construction of a map. An agent must also be able to accurately and consistently estimate its position on the map: the problem of localization. A general and effective solution to this is Monte Carlo Localization (MCL), which will be reviewed in Section 6.1.

The remainder of this chapter will focus on performing MCL in the newly learned representation. ARE provides both a coordinate system and the actual d-dimensional embedded points, $x_0, \ldots, x_n$, that correspond to the trajectory the robot followed in the data-gathering (map making) phase. This trajectory—along

with the robot's actual observations, $z_0, \ldots, z_n$, and actions, $u_1, \ldots, u_{n_1}$—can be used to learn the required motion and sensor models for MCL from the data. Both motion and sensor models will be learned in a similar fashion: first estimating the expectation of a model, then using errors to estimate the noise.

6.1 Background: Monte Carlo Localization

Monte Carlo localization (MCL [FBDT99]) is a method for local position tracking which estimates the posterior distribution of a robot's pose conditioned on the actions and sensor readings. It relies on the Markovian assumption that the past and future are conditionally independent given the present. MCL is an implementation of a recursive Bayes filter. If x_t is the location at time t, z_t is the sensor data at time t, and u_t is the motion data at time t then the posterior distribution becomes:

$$\texttt{Bel}(x_t) = p(x_t|z_T, u_T) \tag{6.1}$$

where $z_T = z_0, \ldots, z_t$ and $u_T = u_1, \ldots, u_t$. For objective localization the sensor data is usually in the form of range data, such as laser range-finder readings; however, any type of sensor for which the proper kind of model exists is admissible. The motion data is usually obtained from the robot's odometers, but again, any data with an appropriate model will satisfy the equation.

For a recursive Bayes' filter, a recursive formula is necessary so Equation 6.1 is

converted by Bayes' rule to:

$$\texttt{Bel}(x_t) = \frac{p(z_t|x_t, z_{T-1}, u_T)p(x_t|z_{T-1}, u_T)}{p(z_t|z_{T-1}, u_T)}.$$

Since the denominator is constant relative to x_t it can be replaced by a normalizing constant Z. Also, by the Markovian assumption $p(z_t|x_t, z_{T-1}, u_T) = p(z_t|x_t)$ so the belief can be rewritten as:

$$\texttt{Bel}(x_t) = \frac{1}{Z}p(z_t|x_t)p(x_t|z_{T-1}, u_T).$$

The last term can be integrated over all states at time $t-1$:

$$p(x_t|z_{T-1}, u_T) = \int p(x_t|x_{t-1}, z_{T-1}, u_T)p(x_{t-1}|z_{T-1}, u_{T-1})dx_{t-1}.$$

Again by the Markovian assumption $p(x_t|x_{t-1}, z_{T-1}, u_T) = p(x_t|x_{t-1}, u_t)$ while the last term in the integral can be substituted with Equation 6.1. The final recursive formula is:

$$\texttt{Bel}(x_t) = \frac{1}{Z}p(z_t|x_t) \int p(x_t|x_{t-1}, u_t)\texttt{Bel}(x_{t-1})dx_{t-1}. \tag{6.2}$$

$p(x_t|u_t, x_{t-1})$ is called the motion model, the probability of a resulting pose given a starting pose and an action. $p(z_t|x_t)$ is called the sensor model, the probability of receiving a particular sensor reading given the robot's pose. If these two models exist and the integral can be computed then Bayes' filtering can be performed.

Since virtually all robots operate in a continuous space, the integral in Equation 6.2 is often impossible to compute directly. Instead MCL approximates the

continuous space with a finite set of samples called "particles". At each time-step the set of samples is moved probabilistically according to the motion model, then annotated with a weight determined by the sensor model (the weight is the probability of receiving the observed sensor reading given that the robot is at the location represented by the particle). Finally, the particles are resampled according to this weight. Re-sampling generates a new set of samples by choosing a particle with probability proportional to its weight with replacement. Although MCL is only provably correct as the number of samples approaches infinity, it is often accurate for a relatively small number of samples. See [TFBD00] for a more detailed derivation of MCL.

6.2 Motion Model

The motion model is the posterior distribution $p(x_t|u_t, x_{t_1})$. Since this model will be used in a particle filter, it is only necessary to draw a sample, $\hat{x}$, from the model, given a u_t and x_{t-1}, i.e.: $\hat{x} \sim p(x_t|u_t, x_{t-1})$. First, separate the model into an expectation plus a noise component:

$$\hat{x} \sim E(x_t|u_t, x_{t-1}) + \eta(x_t|u_t, x_{t-1}). \tag{6.3}$$

Now make the simplifying assumption that the noise depends only on the action and not on the previous pose. This gives the form:

$$\hat{x} \sim E(x_t|u_t, x_{t-1}) + \eta(x_t|u_t).$$

A model can now be constructed by learning the expectation component, then using the sample errors to estimate the noise component.

Consider some action u. Every t where $u_t = u$ gives one sample, x_t and x_{t-1}, from the distribution $p(x_t|u, x_{t-1})$. Using these sample points, a function over x_{t-1} is desired which provides a close estimate of x_t. ARE explicitly includes constraints that ensure such a function exists and is a simple rotation plus a translation in the learned representation. These functions can be recovered as described in Section 5.1 and therefore the expected motion can be defined as:

$$E(x_t|u_t, x_{t-1}) = A_{u_t} x_{t-1} + b_{u_t} \tag{6.4}$$

Since only the top d principal components of the output of ARE were included, this model of the expected motion will not be exact. The errors in the learned transformation can be used to build a model of the motion noise. Again consider some action u, let ξ_t^u be the residual error for action u on x_t:

$$\xi_t^u = A_u x_{t-1} + b_u - x_t \quad \texttt{where} \quad \sum_{t:u_t=u} \xi_t^u = 0.$$

It is expedient to model the motion noise as a zero-mean multivariate Gaussian, where the covariance matrix can be estimated directly from the samples ξ_t^u. Formally:

$$\eta(x_t|u_t) \sim N(0, \Sigma_{u_t}), \quad \texttt{where} \quad \Sigma_{u_t}(i,j) = \sum_{t:u_t=u} \xi_t^u(i)\xi_t^u(j). \tag{6.5}$$

Combining Equations 6.3, 6.4 and 6.5 gives the complete motion model.

6.3 Sensor Model

The sensor model is the probability distribution $p(z_t|x_t)$. In the context of a particle filter, the density of the distribution at z_t must be provided for a given x_t. In estimating this model from the data a few assumptions must be made. Notice that ARE does not take images directly as its input, but rather uses an image's distance to every other image as a kind of feature representation. The same representation will be used for new observations, computing a feature vector:

$$\bar{z}_t(i) = \|z_t - z_i\| \quad \forall i = 1 \ldots n.$$

The best way to view this feature vector is that it provides a crude estimate of the "distance" of the robot's pose to the previous poses, $z_0, \ldots, z_n$. An additional assumption is required that each of the components of the feature vector are independently distributed.[1] That is, each component is an independent estimate of the "distance" to a past pose. The final assumption is that this probability only depends upon the *distance* to the specific past pose in the subjective representation,[2] i.e., $\|x_t - x_i\|$. These assumptions combine to give the following form for the model. Let $d_{ti} = \|x_t - x_i\|$:

$$p\left(z_t|x_t\right) = p\left(\bar{z}_t|x_t\right) = \prod_{i=1}^{n} p\left(\bar{z}_t(i)|x_t\right) = \prod_{i=1}^{n} p\left(\bar{z}_t(i)|d_{ti}\right). \tag{6.6}$$

[1]This assumption, while almost certainly incorrect, is similar to the common MCL assumption (often necessary for tractability) that sensor readings are independent.

[2]This is not an unreasonable assumption, since ARE explicitly constrains distances in the subjective representation $\|x_i - x_j\|$ by observed image distances $\|z_i - z_j\|$.

Now it is necessary to estimate a model for the conditional random variable $\bar{z}_t(i)|d_{ti}$. Consider again the training trajectory, each x_t gives one sample for this joint distribution: $\bar{z}_t(i)$ and d_{ti}. To build a Gaussian model, for each previous observation i use regression to fit a low-degree polynomial determining the distribution mean as a function of distance ($\mu_i(d_{ti})$).[3] Then take the mean of the squared errors to estimate distribution variance (σ_i^2). This gives the following Gaussian density function:

$$\bar{z}_t(i)|d_{ti} \sim N\left(\mu_i(d_{ti}), \sigma_i^2\right). \tag{6.7}$$

Combining Equations 6.6 and 6.7 gives the sensor model.

6.4 Results: Localization

The final step of the technique is to use the motion and sensor models with Monte Carlo localization to track the robot's position in the learned subjective space. The only detail left to be addressed is the initial distribution for localization. Since test data is processed after a single training run, the exact position in the subjective representation, x_n, is known. All the samples in MCL are initialized to this point.

In the end, the subjective localization procedure has three configurable parameters: the dimensionality of the subjective representation, d, the degree of the polynomial used in the sensor model, and the number of particles used by MCL. Overall, the procedure has a small number of parameters and can successfully localize in a number of different situations with a variety of parameter settings.

[3] This is very similar to the sensor model construction by Stronger and Stone [SS05].

In all experiments, a dataset is gathered by executing a sequence of actions and receiving the associated sequence of sensor readings. After each action, measurement of objective location is taken—this is only used afterwards to evaluate the localization accuracy. The sequence is split into two sets, training and test. The training set is used by ARE to extract a subjective representation and associated trajectory. Motion and sensor models are learned as described previously. Finally, the models are used in MCL to localize given the test set. The mean of the particles after every given action and observation is used as the estimated position in the subjective frame of reference.

In order to extract a model of noise, the training data needs to contain examples of executing the same action from approximately the same location. Since the points after taking this action will be in various locations, the noise of the motion model can then be reconstructed. Therefore, each dataset begins by taking repeated short sequences of actions such as going forward three steps then backward three steps, ensuring the training data includes a representation of noise in the robot's actions.

Figure 6.1 shows an example "A" shaped path in objective coordinates. The dotted line shows the training data (with the trajectory starting in the top left. The solid line shows the test data, a reversed "A" continuing on from the last point in the training data (at the bottom left). Note that this data has been generated with a noisy version of ImageBot so the paths do not exactly line up.

Figure 6.2 shows results of using the data associated with the trajectory in Figure 6.1 with the described subjective localization technique. The dotted line shows the trajectory that resulted from running ARE on the training data in the learned

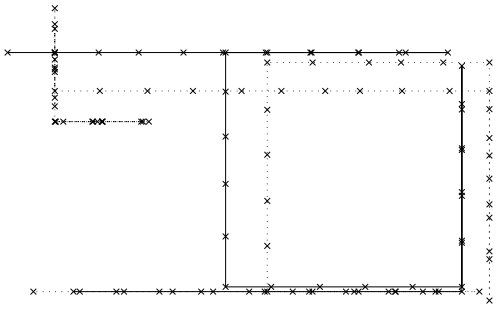


Figure 6.1: ImageBot's "A" shaped trajectory in objective coordinates. The dotted line is the training data, the solid line is the test data.

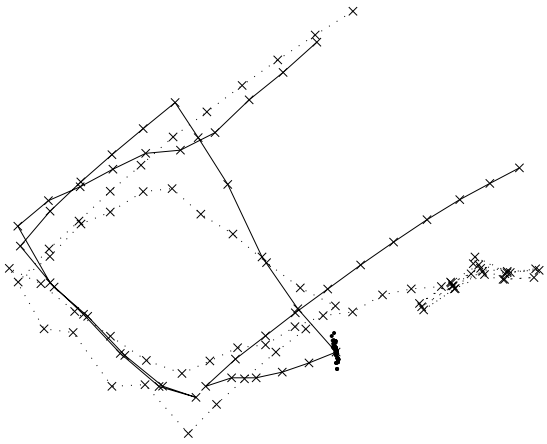


Figure 6.2: Subjective localization on the trajectory from Figure 6.1. The dotted line is ARE's learned representation, the solid line is predicted locations of the test data using MCL.

frame of reference. The solid line is the predicted points from MCL (while receiving images and actions from the test data). The cloud of points at bottom centre shows the set of 100 particles in MCL at that point in the trajectory. The learned trajectory corresponds strongly with the objective trajectory, and the localized trajectory follows appropriately.

Preliminary results have also been generated from an actual robot following a trajectory similar to that in Figure 4.2 (i.e. forward and backward only). Experiments were performed on an ActivMedia Pioneer 3 DX8 robot equipped with an ordinary web camera which returned 160×120 pixel images. Predefined discrete actions were used to move the robot up and down a corridor with the image from the camera being collected after each action was performed.

The observations from this trajectory are shown in Figures 6.3, 6.4 and 6.5.

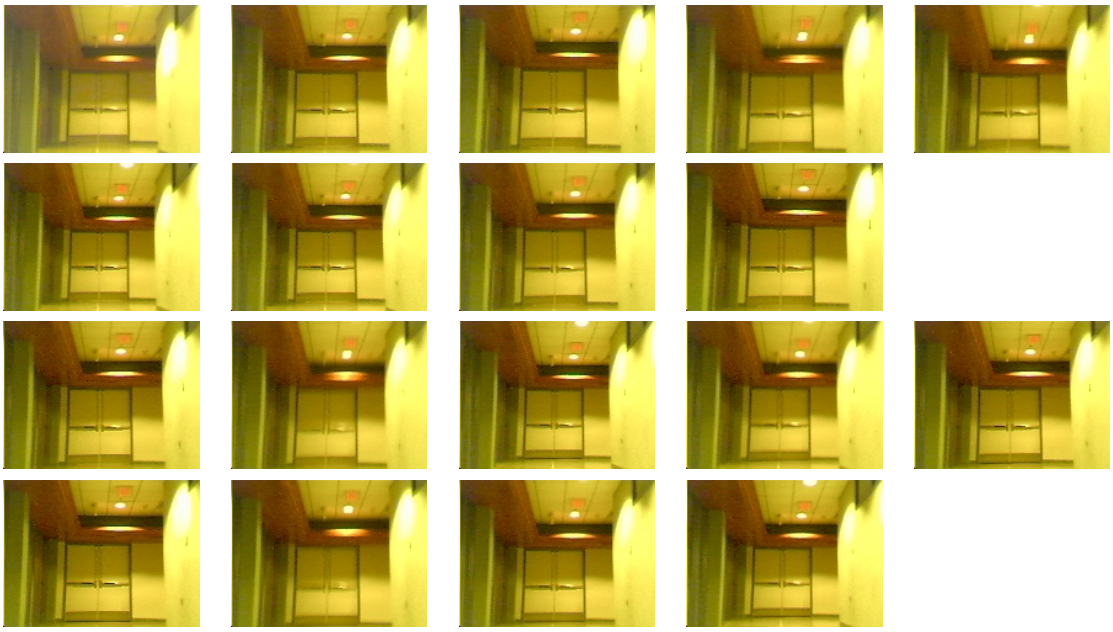


Figure 6.3: The first half of a training set of observations from a robot.

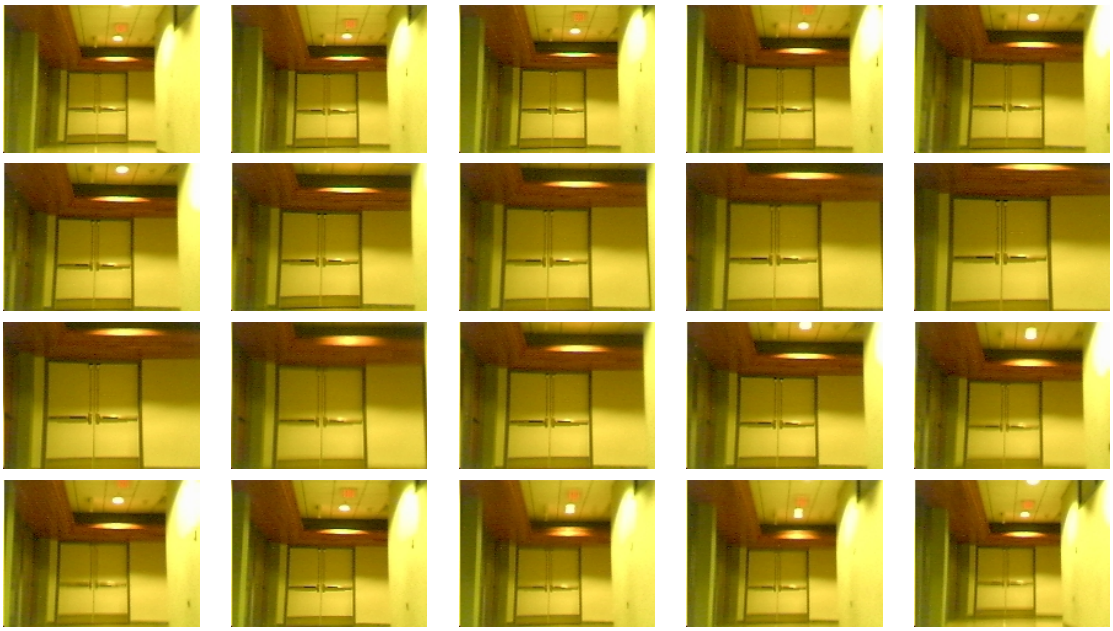


Figure 6.4: The second half of a training set of observations from a robot.

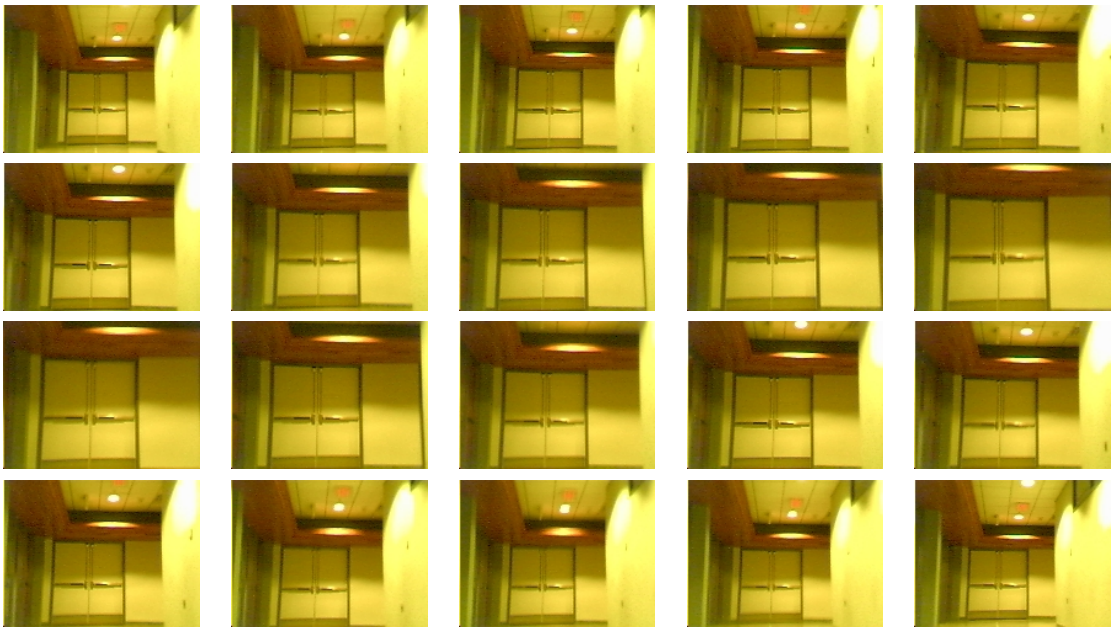


Figure 6.5: The test set of observations from a robot.

Again, to build a model of noise the robot moves forwards and backwards a short distance a couple of times to start with (Figure 6.3) before moving ten steps forward, then ten steps back (Figure 6.4)—these observations are used to learn a representation as described in Chapter 4. Finally, the robot again moved ten steps forward, then ten steps back to generate a test set of observations to be used to test the localization procedure described in this chapter.

ARE was run on the observations in Figures 6.3 and 6.4 and the associated action labels. The grey line in Figure 6.6 plots the trajectory of the manifold over time. Note that, as in Figures 4.3 and 4.4, the learned manifold is similar to the underlying generative manifold. The black line in Figure 6.6 shows the localization predictions (in other words the mean of the relevant particles) of the observations

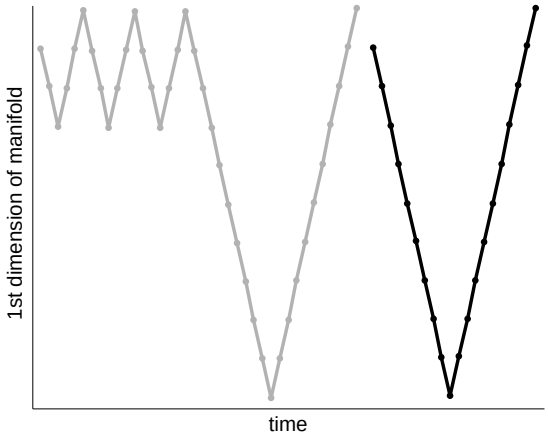


Figure 6.6: The grey line is a manifold learned from a robot moving forwards and backwards (generating the observations in Figures 6.3 and 6.4). The black line is the localization predictions of the observations in Figure 6.5.

from the training set (Figure 6.5) over time. Note that the localization procedure is predicting positions on the manifold for the test observations that are very close to where one would expect.

6.5 Conclusion

Section 6.1 provided a brief overview of the Monte Carlo localization method (MCL). The two critical requirements of MCL are a motion model and a sensor model. A generic motion model which could be used for a representation learned by ARE was developed in Section 6.2 using the transformations recovered in Section 5.1. Similarly, a generic sensor model was developed in Section 6.3. Section 6.4 showed some results of running MCL in a learned representation, both in the ImageBot test domain and on an actual robot with a mounted camera.

This chapter has demonstrated how MCL can be applied to the representations learned by the ARE algorithm of Chapter 4. If the representation is successful in capturing the dynamics of the environment, then the generic motion and sensor models should be sufficient for using MCL. The results bear this out, showing that MCL can be used to successfully localize an agent in a representation after it has been learned. This could be useful in the robotics domain where custom motion and sensor models are normally required for localization tasks—replacing these typically expensive custom models with generic ones that can be learned from the data is highly desirable.

Chapter 7

Scaling: Manifold Merging

> Divide each difficulty into as many parts
> as is feasible and necessary to resolve it.
>
> Rene Descartes

This chapter addresses scaling issues with the Action Respecting Embedding algorithm presented in Chapter 4. Chapters 5 and 6 demonstrated the usefulness of ARE-produced representations given a short stream of experience. Learning representations for longer streams is desirable but as ARE requires the solving of a semidefinite program, it may not scale well with the length of the stream—consider that a rough bound on the time complexity for solving a semidefinite program is $O(m^3)$ [Bor99] where m is the number of constraints and that ARE potentially adds $O(n^2)$ constraints where n is the number of observations. This chapter provides a divide-and-conquer algorithm which mitigates this issue.

7.1 Divide-and-Conquer

One paradigm for dealing with any large problem is divide-and-conquer: split a problem into smaller independent subproblems, solve the subproblems, and then merge the subproblem solutions into the final answer. An obvious application of this approach is to divide a long input stream $\langle z_1, u_2, z_2, \ldots, u_n, z_n \rangle$ into tractable sized substreams $\langle z_1, u_2, \ldots, u_m, z_m \rangle$, $\langle z_m, u_{m+1}, z_{m+1}, \ldots u_{2m}, z_{2m} \rangle$, and so on, then learn a representation for each substream with ARE. It only remains to describe how to combine the individual representations into one global representation.

Figure 7.1 shows an "8"-shaped trajectory—clearly tractable for ARE. However, if this trajectory is repeated 9 times (effectively creating a stack of 9 "8"-shaped sequences), the resulting stream is too long for ARE to handle. The divide-and-conquer approach begins by first breaking up the stream into tractable sized substreams; in this case, taking each "8"-shaped sequence as a separate substream. The only constraint is that the final point of a substream is the first point of the next substream so there is one point of overlap between the substreams. ARE is then used to learn a representation for each substream independently. With this approach, the dimensionality, d, of all representations to be merged must be the same. For this example, $d = 2$.

Figure 7.2 shows the representation learned by ARE for the first substream. It quite clearly has captured the underlying structure of its actions. Figure 7.3a shows the representation learned for the second substream, again reflective of the underlying model.[1] The first challenge is to combine these two representations into

[1]Note the representation in Figure 7.3b was clearly not as successful as the others. This will

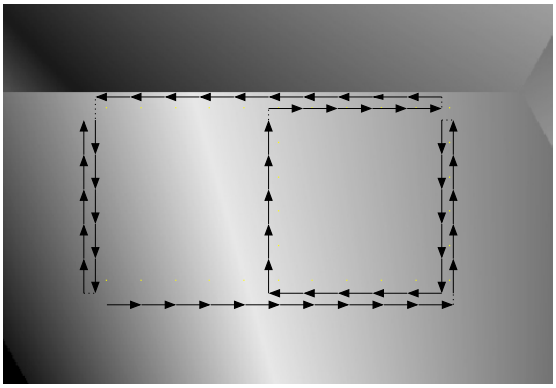


Figure 7.1: An "8"-shaped ImageBot trajectory.

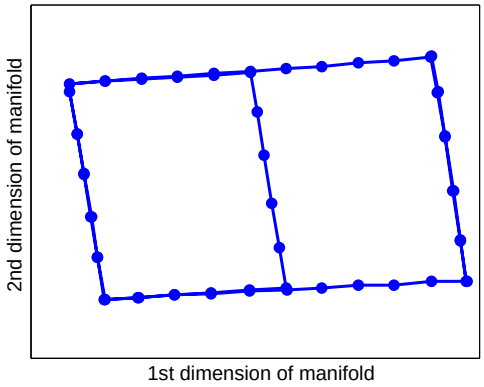
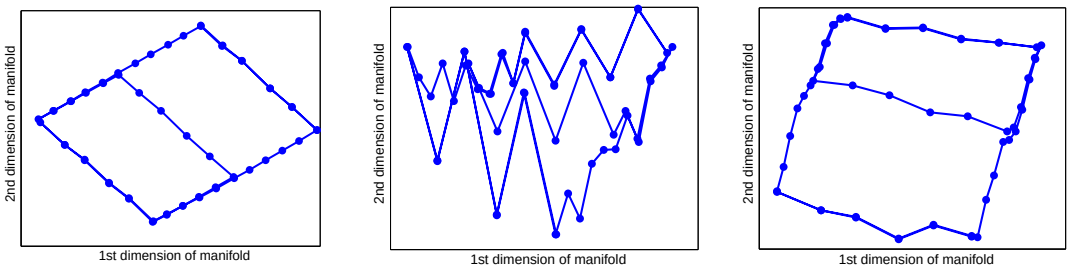


Figure 7.2: A manifold learned from the trajectory in Figure 7.1.



a b c

Figure 7.3: Learned representations for the 2nd, 3rd and 4th "Figure 8"s.

a single joint representation.

7.2 Combining Representations

Given two separate manifolds $\langle x_1^1, \ldots, x_m^1 \rangle$ and $\langle x_1^2, \ldots, x_m^2 \rangle$ learned by ARE, this section describes how to combine them into a single manifold. If there was a mapping from one manifold to another, it could be used to map all the points

be addressed later.

from one manifold into the other to create a joint manifold. Such a mapping between manifolds should be linear in order to preserve the actions as scale-free linear transformations.

One way to find this linear mapping is from a correspondence set: a set of points which appear in both manifolds. For example, suppose two sets of points are given: $\{y_1^1, \ldots, y_k^1\}$ and $\{y_1^2, \ldots, y_k^2\}$, where y_i^1 is in the first manifold and corresponds with y_i^2 in the second, giving k points of correspondence. The goal is to find a linear mapping T that minimizes $\sum_{i=1}^{k} \|y_i^1 - Ty_i^2\|^2$ (the sum squared error in mapping the points of correspondence from the second manifold into the first). This linear mapping can be found using simple linear regression. Once T has been obtained, let $x_{m+i}^1 = Tx_i^2$ for $i = 1 \ldots m$, then $\langle x_1^1, \ldots, x_{2m}^1 \rangle$ is the joint manifold. The final step of the merging is to recompute the action transformations (as shown in Section 5.1) in the newly merged representation.

To complete this procedure requires a set of points in correspondence. The manifolds were constructed to have one known point in correspondence: x_m^1 and x_1^2. Let $y_1^1 = x_m^1$ and $y_1^2 = x_1^2$. This single point of correspondence along with the action transformations will generate the rest of the set. For $i = 2 \ldots m$

$$y_i^1 = f_{u_{i+1}^2}^1(y_{i-1}^1) \qquad y_i^2 = x_i^2,$$

where f_a^1 is the transformation associated with action a in the first manifold. In other words, project (based on the action transformations) a sequence of hypothetical points where the second manifold will map into the first, then place these hypothetical points in correspondence with the actual points in the second manifold

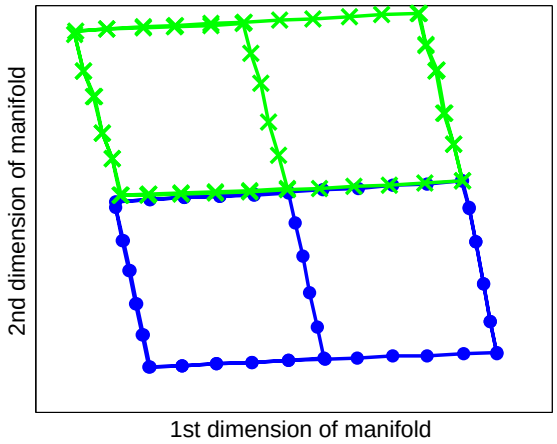


Figure 7.4: The second manifold added to first.

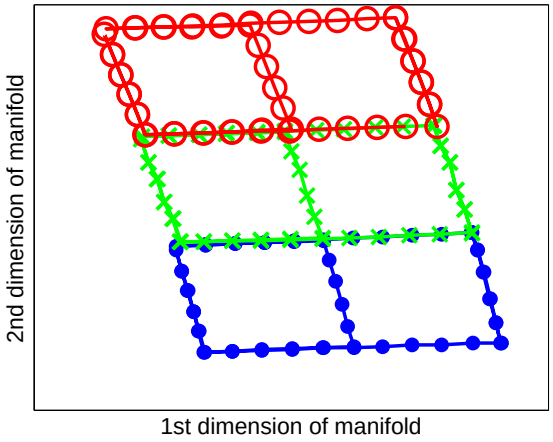


Figure 7.5: The third manifold added by projection.

to learn the linear transformation.

Returning to the concrete example, Figure 7.4 shows the results of applying this procedure to combine the first and second manifolds from Figures 7.2 and 7.3a. The representations returned by ARE for the first two substreams successfully captured the underlying model. However, if an attempt is now made to merge in the third substream, shown in Figure 7.3b, there is an apparent problem. It appears that ARE has failed to learn a sensible representation for this portion of the trajectory. Combining this representation with the others will likely only confuse the resulting map. Rather than blindly merging the substreams, it is necessary to identify instances where ARE has failed to produce a reasonable representation. A quantitative measure for evaluating a manifold is required.

7.3 Another Evaluation Technique

Given a manifold $\langle x_1, \ldots, x_m \rangle$, a quantitative measure of the quality of the manifold needs to be computed. The projection accuracy of the action transformations is one such measure. For every action u_i in the original input there is a corresponding x_{i-1} and x_i. The projection error for i is $\|x_i - f_{u_i}(x_{i-1})\|$—the distance between the projected ith point using the action transformation and the actual ith point. This error is scaled by the actual distance travelled. Taking the average over all actions provides the following quantitative evaluation of a manifold,

$$\textsc{ProjErr} = \frac{1}{n-1} \sum_{i=2}^{n} \frac{\|x_i - f_{u_i}(x_{i-1})\|}{\|x_i - x_{i-1}\|}.$$

ProjErr has a natural interpretation as the average error in the action transformations as a percentage of the total distance travelled. The ProjErr values for the manifolds in Figures 7.2–7.3c are 0.05, 0.1, 0.51, and 0.22, suggesting that this quantity can be used to identify poor representations that should not be merged. Limited experiments suggest a threshold of $\tau = 0.15$ so for the following results only manifolds with at most that ProjErr are merged.

If ProjErr is higher than the threshold τ, the corresponding manifold should not be combined into the joint manifold, yet something still needs to be included in the joint representation for this substream. As shown before, actions can be projected ahead to create a hypothetical sequence of points for the correspondence set. In the absence of a good manifold, these projected points can be used directly as position estimates for the problematic substream. Figure 7.5 shows the result

of projecting a hypothetical sequence instead of using the high ProjErr manifold from Figure 7.3b.

7.4 Results: Merging

Here is the final divide-and-conquer algorithm: Split the large stream into many tractable substreams, run ARE on each substream, then compute ProjErr for each manifold generated by ARE. Next, combine the manifolds by checking if ProjErr for the next manifold in the sequence is less than τ. If so, use projection to generate points of correspondence, use linear regression to find a linear map, then map the points from this manifold into the joint manifold. If ProjErr is too high, project a hypothetical sequence for this substream using the action transformations.

Before discussing planning, first two of the resulting manifolds are shown. Figure 7.6 shows the final result of the algorithm applied to the stacked "8"-shaped stream on the synthetic image (the running example throughout Section 7.2). Figure 7.7 shows the result of the algorithm on a stream of experience generated by a random walk, also on the synthetic image. In both cases, the representation appears to capture, to varying degrees, the underlying structure of the domain (in particular that the four actions consist of two orthogonal pairs of opposing actions). This is an impressive result due to the length of input streams. More importantly, the resulting representations can be used for planning as shown in the next section.

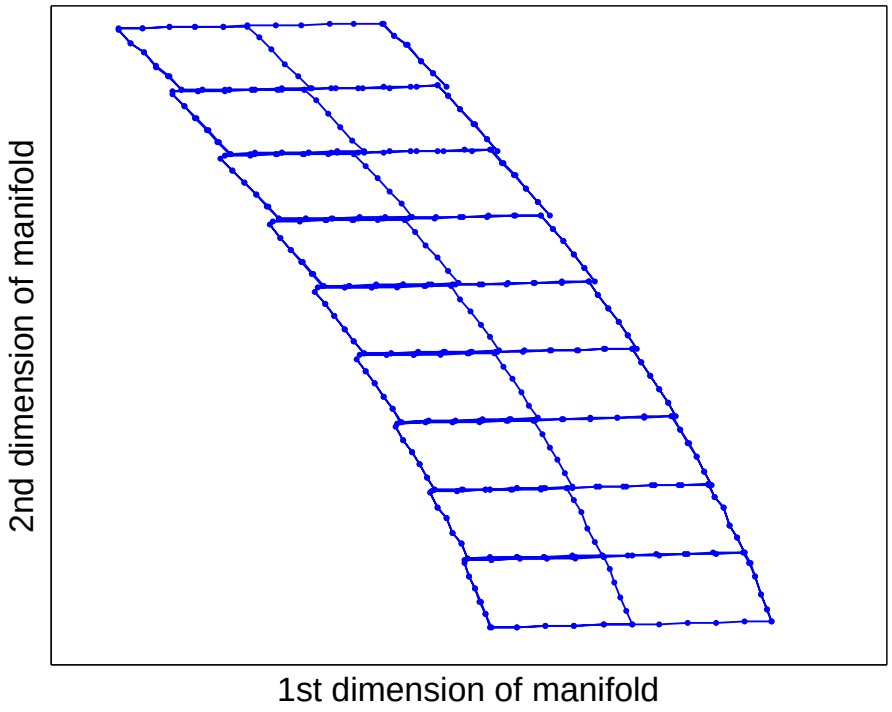


Figure 7.6: A merged representation for the "stacked 8's" trajectory.

7.5 Results: Evaluation

This section evaluates the approach in a manner similar to that of Section 5.3. The goal of subjective mapping is a *useful* representation, so again, the evaluation will focus on using the representations for planning.

Experiments consist of two streams using the synthetic image from Figure 3.2 and one stream using the natural image from Figure 3.3 (used in Chapters 4 and 5). For the synthetic image, one stream involved the nine stacked "8"-shaped trajectories and the other stream involved a random walk. For the natural image the stream involved the nine stacked "8"-shaped trajectories. Each of the three

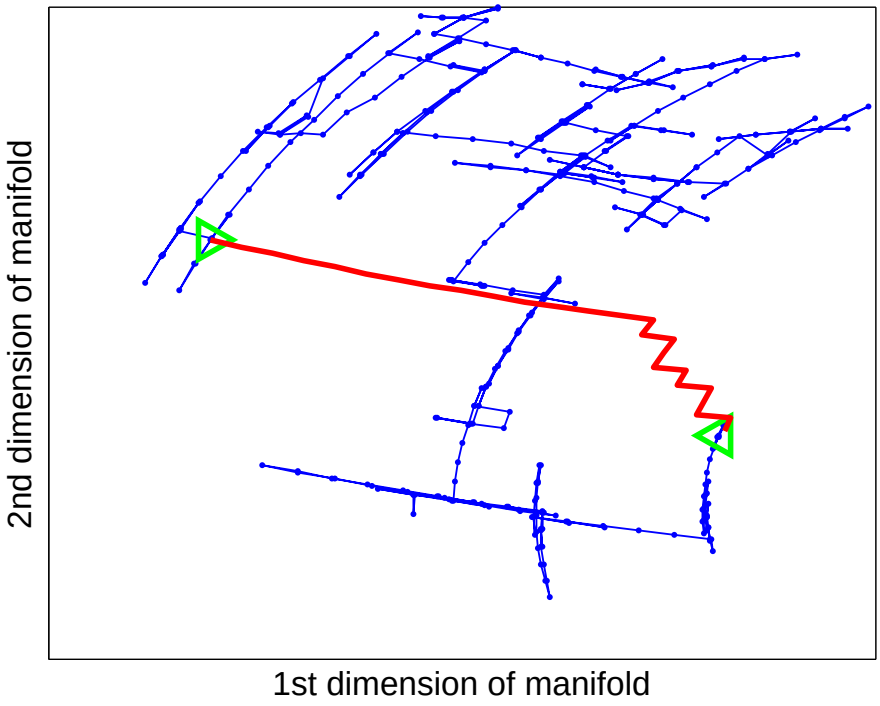


Figure 7.7: A merged representation for a long random trajectory.

streams is nearly an order of magnitude longer than those presented in Chapter 4.

For each of the test streams fifty planning problems were generated by selecting random start and goal points from the original input stream. For each problem, best-first search was used with Manhattan distance as the heuristic to find a sequence of actions from start point to goal point, using the action transformations as models. Figure 7.7 shows one example of start and goal points (labelled with triangles) along with the resulting planned path overlaid on top of the learned manifold. Next, the plan was executed in the ImageBot domain and the distance (in pixels) between the start and goal point as well as the distance (in pixels) between where the plan ended up and the goal point (called the "goal difference") were

Stream	Mean Distance		Mean Goal Difference		% Exact	
	D&C	ARE	D&C	ARE	D&C	ARE
Stacked "8" (Synthetic)	340.88	166	33	32.5	32%	62%
Random (Synthetic)	393.16	139.17	76	59.17	16%	40%
Stacked "8" (Natural)	439.22	152	48	72.5	32%	36%

Table 7.1: Quantitative evaluation of merged representations based on path planning.

recorded. The number of times the predicted goal exactly matched the actual goal was also recorded. The results for all streams are shown in Table 7.1 under the heading "D&C". The mean distance is the average distance between start and goal points, the mean goal difference is the average distance between plan result and actual goal and the percentage exact is the percentage of time that the plan result and the actual goal are the same. Note that the agent's actions adjust its position by 25 pixels, so a 50 pixel mean goal difference means that on average the plan ended two actions away from the goal. To provide some comparison, an additional 50 planning problems were generated where the start and goal points were within the same substream. The planning task was then evaluated in the ARE manifold for just that substream. These results, shown under the heading "ARE", give an idea of how useful typical ARE representations are when used for planning.

The planning results for the natural and synthetic stacked "8" streams are about the same. The start and goal points were on average 14–18 actions away and the plans get within two actions of the goal, or approximately 90% of the total path. One third of the time it reaches the goal point exactly. The ARE plans for these streams, although shorter problems, performed similarly, albeit with a higher probability of reaching the goal point exactly. Planning with the representations

learned from the random walk streams was more difficult. The problems are about the same length, but the plans reach the goal half as often and end on average three actions from the goal. This quantitative difference is not surprising, as the stacked "8"-shaped stream ensures that each substream sees enough of each action with enough overlap for ARE to be able to identify the underlying structure. This will not be true for random streams. As a result more of the substream manifolds may fall below the ProjErr threshold, resulting in a less accurate global manifold. In general, the divide-and-conquer technique produces manifolds an order of magnitude larger than is possible with ARE alone, while being roughly as useful for planning as a single ARE manifold.

7.6 Conclusion

Section 7.1 illustrated the requirements for extending the ARE algorithm from Chapter 4 using a divide-and-conquer approach. Section 7.2 demonstrated a way to combine two representations learned by ARE requiring only one point of correspondence between them. A shortcoming of the divide-and-conquer method is that it is not always desirable to merge a particular learned representation as ARE might not have worked sufficiently well for that particular substream. Consequently, Section 7.3 introduced another way of evaluating a particular representation when it is not possible to test plans as was proposed in Section 3.3. The ability to merge representations combined with the ability to choose when a representation is acceptable for merging yielded the entire approach. Sections 7.4 and 7.5 show the results of applying the new divide-and-conquer method to streams much longer

than those from Chapter 4.

The divide-and-conquer algorithm presented in this chapter makes it possible to learn representations with input sizes an order of magnitude larger than previously shown in Chapter 4. Despite the much-larger resulting representations, the random planning task introduced in Chapter 5 is still relatively successful. This demonstrates that ARE has the potential to be used in a wider range of possible applications, with an extended range of possible input sizes.

Chapter 8

Related Work

> Knowledge is of two kinds.
>
> We know a subject ourselves, or we know where we can find information on it.
>
> Samuel Johnson

This chapter summarizes some other approaches to learning representations and how they relate to the new subjective mapping approach introduced in Chapter 3. In all of these cases there is a trade-off between the power of the representation and the computational cost of learning or using the representation. The methods will be presented from most powerful (and most expensive) to least.

Section 8.1 briefly reviews work on modelling dynamical systems—equivalent to learning an effective representation for such a system. These models can be arbitrarily powerful but often require the use of very long action/observation sequences in order to learn them and the learning may be computationally intractable even when given enough data.

In many situations, it is desirable to sacrifice some of this power in order to be able to learn more quickly. Section 8.2 is a review of some work on learning maps

from uninterpreted sensor information as well as some work on extracting representations from multivariate time-series data. In both situations, prior knowledge is used to reduce the amount of required training data but a large training curve still remains.

Section 8.3 introduces probabilistic mapping methods for robotics including simultaneous localization and mapping (SLAM), which simultaneously localizes and builds maps using Monte Carlo methods. This method requires much less input data in order to build a map, but, in turn, is very specific. The motion and sensor models involved need to be customized for every actuator and sensor. Another way of looking at this is that a large portion of the work, learning and calibrating the sensor and motion models, is done before the agent is immersed in its environment.[1]

8.1 Dynamical Systems

There has been a great deal of work in modelling dynamical systems, which has many similarities to the task of learning a representation of a system. The general definition of a dynamical system is quite similar to the description of an agent in Section 1.1. Such a system generates a sequence of observations O that is perceived by an agent. If the agent is able to interact with the system it is called a controlled system as opposed to an uncontrolled system [Jae98]. In the case of a controlled system the agent affects the system through a collection of possible actions A. The dynamical system is in a particular state and this state can change over time,

[1]This is not entirely true: the models are environment dependent so they must be developed in a similar environment. For example, motion models constructed for a wheeled robot on carpet may not be effective when used for a wheeled robot on asphalt.

possibly dependent on what actions are taken. The current state of a system can affect the effects of the actions as well as the probability of the next observation. It will be necessary to differentiate between two notions of state: *underlying state* is the current full description of the environment and everything in it, while *state* is an agent's representation of the current situation. From a mathematical point of view, the state of a system can be represented by any sufficient statistic for predicting the future of the system. This can be the underlying state or some other sufficient statistic [LSS02].

The timescale associated with the system, as well as the observations and actions, can all be discrete or continuous. Further distinctions can be made between systems that are completely observable as opposed to partially observable, and systems which are deterministic as opposed to stochastic. Models with the latter capabilities in each case are more powerful but also more computationally expensive to learn.

Often, there is some reward function associated with the states of such a system. If there is such a function then a natural goal is to try to find an optimal policy for the system which is a distribution over actions given state (essentially determining how the agent interacts with the system) that maximizes expected reward. If a problem can be modelled in such a way it belongs in the reinforcement learning paradigm (see for example [SB98]).

Markov chains (MCs) are models which represent uncontrolled, fully observable dynamical systems [RN03]. They incorporate an independence assumption which implies that a state is dependent only on the previous observation and the passing

of time is modelled through a function which maps each state to a probability distribution over all possible states. Markov decision processes (MDPs) extend Markov chains to the controlled case [Put94]. The difference is that the function now maps state-action pairs to a probability distribution over all possible states. Both MCs and MDPs can be generalized to k-order Markov methods, where the state is dependent not on the previous state but on the previous k states [RN03].

MCs and MDPs can clearly be viewed as models of their associated environment, as their states correspond directly to the underlying states in the environment but construction of these models requires explicit knowledge of these underlying environmental states. This is not applicable to subjective mapping due to the desire to specifically avoid the necessity of such knowledge, instead building representations from streams of observations (and actions). Arguably, if the underlying state of the world is clearly and directly encoded in the observations then building a representation of the environment becomes a simpler task.

To this end, Markovian processes can be modified to handle partially observable systems. Hidden Markov Models (HMMS [Rab89]) are extensions of Markov chains and Partially Observable Markov Decision Processes (POMDPS, [Ast65, SS73, Son78]) are extensions of MDPs. These models postulate a set of underlying states and assume that the system is in one of these states but that the observations are not enough to determine which state the system is currently in. Instead, the agent maintains a probability distribution over these hidden states. This distribution is often called a belief state. Even if the model is known, learning these belief states can be quite difficult (see for example [Mur00]) and in general there is

no reason to believe that the model is available. There are a variety of model-free approaches (see for example [Abe03]) as well as expectation-maximization (EM) algorithms for simultaneously learning structure and transition probabilities ([Chr92]) but these only add to the computational expense of learning POMDPs.

Finally, there are newer representations of state such as predictive state representations (PSRs [SJR04]), which can be further extended into Memory-PSRs (mPSRs [JWS05]); Observable Operator model (OOMs [Jae98]); and Temporal-Difference networks (TD nets [TS05]). The main idea in these cases is to model the state of the system not only with histories but also with predictions over future states. In general, these models can be arbitrarily powerful and consequently tend to require large amounts of training data in order to learn them.

The definition of the subjective mapping problem from Table 3.1 is similar to that of learning an MDP with three differences:

1. The requirement of mapping into $\mathbb{R}^d$ must be changed to mapping into $\mathbb{S}$ (an arbitrary state space).

2. The line:

$$\forall a \in \mathbb{Z}_{n_a} \quad \exists f_a(\cdot) \in \mathcal{F} \quad : \quad u_i = a \quad \Rightarrow \quad f_a(x_{i-1}) \approx x_i,$$

must be altered to:

$$\forall a \in \mathbb{Z}_{n_a} \quad \exists f_a(\cdot) \in \mathcal{F} \quad : \quad u_i = a \quad \Rightarrow \quad f_a(x_{i-1}) = g(x),$$

where $g(x)$ is a probability distribution function over $\mathbb{R}^d$ parameterized in

some way by x. This allows for probabilistic actions, required for correspondence to a probabilistic transition function.

3. The notion of "simplicity" must allow $\mathcal{F}$ to be the set of lookup tables (equivalent to MDP transition functions).

This variation of subjective mapping is actually equivalent to the problem of learning a general MDP. It can almost be extended to learning a general POMDP but the details of learning belief states (distributions over states given observations) are not trivially intertwined with the dimensionality reduction step. Still, this suggests a number of workable approaches to scaling or modifying current (PO)MDP techniques. Changing the degree of relaxation of each of the above constraints should restrict the class of (PO)MDPs that can be learned. It would be interesting to study the resulting tradeoffs in power and complexity.

8.2 Mapping From Raw Sensor Data

This section outlines two approaches towards learning representations that are more restrictive than the methods for learning a general dynamical system. The first assumes the representation is specifically of a spatial environment while the second attempts to extract information from data using the assumption of temporal contiguity.

8.2.1 Mapping Spatial Environments

One approach to the problem is to explicitly design general models for a more specific task—that of building a cognitive map of a spatial environment. This is the approach suggested by [Kui00] who proposes the spatial semantic hierarchy, consisting of sensorimotor, control, procedural, topological and metrical levels. The sensorimotor level details an agent's interface with the environment via raw sense and control vectors and must further extract important features from the raw observed data. At the control level, action models must be learned to predict the effects of the motor control vectors on those features. At the procedural or causal level the continuous environment is abstracted into a model of discrete views and discrete actions and the relationships between them. At the topological level a finite state machine is constructed based on these sets and relationships, then at the metrical level this state machine is annotated with distances to construct a geometrically accurate map of the environment. The full hierarchy is hypothesized not only as a useful model for robotic agents, but also a descriptive model of how human agents model spatial environments.

Plausible procedures for the first three levels are detailed in [PK97]. Given uninterpreted sense and control vectors (s and u), structure is identified in the environment (defined exclusively by streams of these vectors) suitable for prediction and navigation. This is done through three distinct steps:

1. Building a model of the sensors

2. Building a model of the motor apparatus

3. Learning an abstracted (and discrete) set of behaviours

The first step is done by learning low level features (such as scalar, vector, matrix, image and field features) from the raw sense vector, then applying feature generators (using minimum, maximum and differentiation) to learn new features from existing ones. For example, if a sense vector contains, among other things, a collection of pixel intensities, then a group feature generator will attempt to identify that that collection of features belongs together. Further, an image feature generator would then attempt to determine positional relationships between features within an identified group (for example, that one pixel is above another). The group feature generator requires distance metrics that can be used to recognize similar sensors in different ways. The image feature generator also requires these metrics and uses metric scaling to construct the image features.

The second task is done by characterizing the effects of the primitive actions u on s using motion feature generators which relates actions to spatial and temporal derivatives within groups of features—predicting the effects of motor control vectors on features. This is done by:

- discretizing the continuous control vectors into representative motor control vectors
- randomly executing these representative vectors while maintaining average motion vector fields (AMVFs)
- using principal component analysis to extract the most important motion effects

- matching AMVFs with these motion effects to identify primitive actions

For the final step, an attempt is made to determine local state variables (for example, the closest wall to a robot may be determined by the minimum distance reading obtained from a group of distance sensors). Next, linear regression is used to learn a static action model which predicts the effect of the learned primitive actions on these local state variables. Now behaviours can be learned by relating local state variables to arbitrarily chosen target values. Homing behaviours can be learned that minimize an error signal while path-following behaviours can be obtained by learning the correct feedback model for error correction. Finally, the original environment can be discretized into actions (the finite set of homing and path-following behaviours) and views (collections of similar sense vectors). Since both actions and views are discrete, the agent can try all possible successive state-action pairs to learn a complete graph of its state space. Note that if the number of states and/or actions is large this step could be very time consuming.

Once the environment has been discretized into actions and views (similar to the states in Section 8.1) the next step is to build a graph of the relationships between various states (nodes) according to possible actions (edges). This occurs at the fourth level of the hierarchy called the topological level. Finally, at the fifth level (the metrical level) it may be desirable to annotate the edges of the resulting topological map with distances to obtain a metric map. Approaches to these remaining levels in the spatial semantic hierarchy can be found in, for example, [KB88]. However, already it can be seen that the first three levels of the hierarchy require a fair amount of training data in the form of observation and action

sequences in order to learn the required models.

The ARE algorithm from Chapter 4 is not necessarily orthogonal to this approach. In particular, one could view the building of the non-uniform neighbour graph from the stream of data as constructing a topological map of an environment. Correspondingly, the solving of the semidefinite program could be classified as being in the metrical level of the spatial-semantic hierarchy (although that may be a bit of an over-simplification). Indeed, it may be useful to look at whether tools associated with various levels of the hierarchy can be used to improve upon the subjective mapping solutions presented in this document (for example, to build a better neighbour graph).

8.2.2 Learning Representations From Time-Series Data

Another approach relies on the assumption that the stream of observations (where again each observation is a vector of real values) are close temporally and that consecutive observations must therefore be close to each other. It is possible to learn general structural abstractions from such time-series data to find meaningful episodic structures. For example, imagine that the observation vectors are taken from a distance-finder mounted on a robot that measures distances to objects within a 180° frontal arc. The time series that corresponds to moving past an object has a slow increase in values (as the object is approached) followed by a sharp drop to a constant (after the object has been passed). If these characteristic patterns in a multivariate time series can be grouped into similar experiences then a taxonomy of robot experience can be learned.

The clustering by dynamics (CBD [OSC00b]) method does this by first segmenting a multivariate time-series (each segment will hopefully represent a particular episode, for example, moving towards an object). Next, every possible pair of episodes is compared and similarity between them is measured after finding the best temporal fit with respect to time warping (stretching and/or compressing the temporal axis). These similarities can then be used to cluster the segments into experience groups. Within each cluster a cluster prototype is found and used as the generic representation of the episode associated with that cluster.

These prototypes can be used as planning operators (see [SOC00]) by using a decision-tree induction algorithm to learn rules which map state-action pairs to resulting states probabilistically. The states here correspond to observations and features of these observations are used as decision variables.

There is some evidence that the learned episode prototypes correlate well with prototypes chosen by human judges [OSC00a]. This implies that such methods may be useful as descriptive models for human agents as well as for learning models for artificial agents.

Finally, CBD can be used on raw speech data to identify individual words [Oat02]. This leads to the interesting idea of combining co-occurrent streams of differing sensory modality in order to extract further meaning from raw sensory data. It may be possible to learn labels for raw sensor vectors from a corresponding stream of natural language (for example, the identification of a cup may be learned from a stream of visual data from a camera focused on a cup and a stream of raw audio data saying "there is a cup")[COB02].

The subjective mapping problem shares a critical assumption with CBD: that observations that are temporally close must be spatially close. Beyond that, the two problems are quite different. CBD requires data with enough instances of repetition to make time-series analysis viable and is not as reliant on the presence of actions.

8.3 Probabilistic Mapping in Robotics

Mapping is a key problem in mobile robotics where it is intimately tied to the problem of localization: estimating a robot's position while it moves and senses in the world. Knowledge of a robot's position in its environment is one of the most basic requirements for many autonomous tasks. One of the most successful approaches to objective localization uses probabilities to model all aspects of a robot's uncertainty, including the current pose estimate, the effect of actions, and the information provided by sensors. Probabilistic inference can then be applied in a straightforward fashion to maintain an estimate of the robot's location. Approaches of this type often restrict the form of the models (for example, Gaussian distributions in Kalman filters [Kal60]) or use various approximation techniques (such as the Monte Carlo localization procedure reviewed in Section 6.1), to allow inference and localization to be computationally feasible.

A prerequisite for all probabilistic approaches are models of the uncertainty in the robot's motion and sensors. Classical kinematics defines the expected global motion of the robot when a particular control is applied to it. But kinematics requires many assumptions in its deterministic calculations (for example, infinite friction) that do not hold in practice. Hence, robot motion is uncertain. Likewise, there are

many uncontrollable and unpredictable factors (for example, acoustic reflectance of a surface with sonar, or ambient lighting with vision) that affect readings from sensors. Hence, robot sensing is also uncertain. Probabilistic models of these uncertainties form the basis for inference (which drives the localization). Unfortunately, these models are often not easy to build. They can require extensive knowledge of the robot's kinematics or sensors, which may not be known or easily described. They may require time-consuming manual measurements to estimate characteristics of noise or to build a map of sensor readings over the environment. Finally, by definition, a well constructed model must be specific to the particular hardware used. Modifying the robot platform invalidates these laboriously constructed models and new models must be created. For example, changing from a wheeled robot to a legged robot obviously invalidates the motion model. Changing from a sonar to a laser, or from a laser to a camera will require replacement of the sensor model. Even minor changes, such as inflating the tires on the robot, or replacing its camera with one of a different model, will require expert modifications to the various models. Recent work has examined techniques for automatically calibrating some of these models (see for example [RT99], [MTKW02], [EP04], [SS05], [KBM06]), but no current method exits to calibrate these models for objective localization without considerable expert knowledge.

8.3.1 Simultaneous Localization and Mapping

Simultaneous localization and mapping (SLAM) is an extension of Monte Carlo localization which tracks both uncertainty in the robot's pose as well as uncertainty

in the location of obstacles and landmarks [MTKW02]. It is the quintessential example of objective mapping where the robot's state representation is given, along with partially specified models; in other words, the entire motion model and the sensor's noise model is commonly considered known. In many cases, where the robot and environment is well studied and predictable, this has proven to be a very effective approach. If the robot's locomotion or sensing apparatus, or even aspects of the environment, are not known a priori there can be no partially specified models. Simultaneous localization and mapping with a completely unknown robot is perhaps the "Grand SLAM" challenge, and it is not clear how current objective mapping approaches could handle the complete absence of models and remain objective. As the models are providing the critical connection between subjective and objective quantities it may, in fact, be impossible.

Of the related work examined in this chapter, SLAM is perhaps closest related to the subjective mapping problem of Chapter 3, unsurprising as the motivations for this dissertation are rooted in the same motivations as SLAM. A fundamental difference in the two approaches is that SLAM explicitly seeks to learn an objective map, making its dependence on objective motion and sensor models a necessity. SLAM has been an extremely successful approach to robotic mapping and in situations where models are available or where an objective map is necessary (to be communicated to other agents or to human observers, for example) it is clearly the correct approach. In cases where such models are not easily obtainable or where a subjective map is sufficient, the subjective mapping paradigm introduced in this dissertation provides an alternate and possibly better approach.

Chapter 9

Conclusion

> In the end, everything is a gag.
>
> Charlie Chaplin

Chapter 3 introduced and formalized the subjective mapping problem: to learn a useful representation by extending dimensionality reduction techniques. Chapter 4 developed the action respecting embedding algorithm—a solution to the new problem—and provided qualitative evidence for the value of the subjective mapping approach. Chapter 5 presented a method for extracting the effects of actions in learned representations and showed how these actions could then be used to effectively find shortest paths. Experimental results in a random path-planning task provided further quantitative evidence that subjective mapping and ARE can be effective tools. Chapter 6 demonstrated that the learned representations could be used for localization, demonstrating that subjective mapping and ARE can be practically applicable in the domain of probabilistic mapping for robotics. Finally, Chapter 7 introduced a divide-and-conquer procedure for merging representations,

providing a way of overcoming the complexity issues that arise due to the semidefinite program at the heart of the action respecting embedding algorithm. This chapter briefly discusses some avenues of future research and provides a summary of other sources of information on subjective mapping and ARE.

9.1 Future work

9.1.1 Parameterized Actions

While there are a variety of domains where discrete action labels are sufficient for an agent's purposes, there are some situations where it may be highly desirable to allow the use of continuous or parameterized actions. For example, imagine a situation with a robot whose action set includes not merely "move forward" but instead "move forward distance x". The structured input of this variation of the problem becomes:

$$z_0, u_1(p_1), z_1, u_2(p_2)..., u_n(p_n), z_n$$

where p_i is the parameter of action u_i.

There are some features of the current ARE algorithm that provide hope that it may be modified to handle this new problem. Recall that in the learned representation it is possible to learn operators o_i that corresponded to the original actions u_i. These operators are all of the form (A_i, b_i) and when they are applied to a point x on the manifold they yield a new point $x' = A_i x = b_i$. For the sake of simplicity, examine an operator with no translation component (in other words $b_i = \mathbf{0}$). In this particular case, one can easily answer the question "what happens if this

operator is applied twice": $x' = A_i(A_i x) = A_i^2 x$. This allows for the definition of a new operator $o_i(2)$. Similarly, this can be extended to define any operator $o_i(r)$ for some $r \in \mathbb{R}$.

This is promising as it implies that the underlying method for representing transformations on the learned manifold is already, in some sense, continuous. Transformations correspond to discrete action labels but they are easily manipulated with continuous variable multipliers. This suggests that parameterized actions are plausible.

The argument above clearly applies to operators with only a translation component (in other words $A_i = \mathbf{0}$). Problems may arise in parameterizing those operators with both translation and rotation components, application of operator o_i twice yields $A_i(A_i x + b_i) + b_i = A_i^2 x + (Ab + b)$, repeated application clearly continues to yield transformations of the correct form but the nested translation term may be cumbersome to compute.

There is also an issue with learning a manifold from a collection of parameterized actions. A first naive attempt would involve finding the lowest common denominator, l, of all the parameters for a particular action label, then re-expressing the input as streams of discrete action labels, each corresponding to a parameterized action with parameter l. This is almost a situation which allows for direct application of the ARE algorithm. The operators corresponding to the new action labels would also correspond to the parameterized actions where the parameters are applied as described above. To do this still requires that the ARE algorithm be extended to deal with the situation where the input stream is not perfectly interleaved, in other

words $z_1, u_1, u_2, u_3, u_4, z_4, \ldots$. This proposed method also has the obvious problem that it will greatly blow up the number of actions which may be an issue despite the promise of scalability delivered in Chapter 7.

9.1.2 Extending the Applications

Better planning

The planning results that have already been generated are impressive. One problem, however, is that iterative deepening was used in searching the learned space. This becomes problematic in cases where the desired plan is very long or in cases where there are a large number of actions (resulting in a very high branching factor in the search tree). Ideally, the search tree should be pruned by using something like an $A\star$ algorithm. This raises the interesting question of what an admissible heuristic would be in a space where the actions are distance preserving. An obvious first step in this direction would be to design an admissible heuristic for a space where the actions are simple translations (although even this seems not to be at trivial task), then somehow extend it to a heuristic for general embeddings.

Better localization

The current results for localization, while quite good, have some potential for improvement. In general, there is uncertainty as to whether the gains are coming from the choice of generic motion model or generic sensor model. This question naturally leads to future work where the experiments are repeated with a baseline sensor model, and then with a baseline motion model in order to separately evaluate

the impact of the models. Additionally, it could be beneficial to try to formulate a variety of other generic motion and sensor models to see if the technique can be improved in some or all cases. In particular, while the motion model used seems intuitively to be appropriate, it is not so clear that this is the case for the chosen sensor model. A first step would be to ensure that the sensor model is, in fact, doing something and, if not, replace it with a better one.

Embedding new points

One restriction on the planning algorithm so far is that it is unable to plan from or to a new point. This is because of a problem that ARE has in common with most other non-linear dimensionality-reduction techniques—it is unclear how to embed new points. Given a new high-dimensional point, what is the corresponding point on the learned representation? The analogous question in the subjective mapping case is: Given a new observation (not seen before), what is the corresponding point on the map? Clearly being able to answer this question would be tremendously useful, and not just for planning.

If new points could be embedded then the learned representation could be used for reinforcement learning. A representation could be learned and locations on that representation could be annotated with the rewards associated with the corresponding high-dimensional observations. This would allow a policy to be learned in the representation. Without the ability to embed new points, however, there is no way that a policy could be used in the original observation space.

9.2 More Information

An overall summary of the subjective mapping problem and the ARE algorithm can be found in the Nectar track of the 2006 AAAI proceedings [BWG06]. The material in Chapter 4 appears in the 2005 ICML proceedings [BGW05]. An earlier version appears in Ali Ghodsi's PhD thesis [Gho06] in substantially different form but it is joint work. Part of the material in Chapter 5 appears in the 2005 IJCAI proceedings [WBG05]. The material in Chapter 6 appears in the 2005 ISRR proceedings [BWGM05].

Bibliography

[Abe03] Douglas Aberdeen. A (revised) survey of approximate methods for solving partially observable Markov decision processes. Technical report, National ICT Australia, 2003.

[Ast65] Karl J. Astrom. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10:174–205, February 1965.

[BGW05] Michael Bowling, Ali Ghodsi, and Dana Wilkinson. Action respecting embedding. In *The 22nd International Conference on Machine Learning*, pages 65–72. (ICML 2005), 2005.

[Bor99] Brian Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11:613–623, 1999.

[BWG06] Michael Bowling, Dana Wilkinson, and Ali Ghodsi. Subjective mapping. In *New Scientific and Technical Advances in Research (NECTAR) at the 21st National Conference on Artificial Intelligence*, pages 1569–1572. (AAAI 2006), 2006.

[BWGM05] Michael Bowling, Dana Wilkinson, Ali Ghodsi, and Adam Milstein. Subjective localization with action respecting embedding. In *The 12th International Symposium of Robotics Research*. (ISRR 2005), 2005.

[CC00] Trevor F. Cox and Michael A. A. Cox. *Multidimensional scaling*. Chapman & Hall/CRC, New York, NY, 2nd edition, September 2000.

[Chr92] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *The 10th National Conference on Artificial Intelligence*, pages 183–188. (AAAI 1992), 1992.

[COB02] Paul Cohen, Tim Oates, and Carole Beal. Robots that learn meanings. In *The First Joint Conference of Autonomous Agents and Multiagent Systems*. (AAMAS 2002), 2002.

[DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern classification*. John Wiley and Sons, New York, NY, 2nd edition, 2001.

[EP04] Austin I. Eliazar and Ronald Parr. Learning probabilistic motion models for mobile robots. In *The 21st International Conference on Machine Learning*. (ICML 2004), 2004.

[FBDT99] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte Carlo localization: Efficient position estimation for mobile robots. In *The 16th National Conference on Artificial Intelligence*. (AAAI 1999), 1999.

[Gho06] Ali Ghodsiboushehri. *Nonlinear dimensionality reduction with side information*. PhD thesis, School of Computer Science, Faculty of Mathematics, University of Waterloo, 2006.

[Goo07] Google. Google maps website. http://maps.google.com/, 2007.

[Hot33] Harold Hotelling. Analysis of a complex of statistical variables into components. *Journal of Educational Psychology*, 24:417–441, 1933.

[Jae98] Herbert Jaeger. Discrete-time, discrete-valued observable operator models: A tutorial. Technical report, German National Research Center for Information Technology, 1998.

[JWS05] Michael R. James, Britton Wolfe, and Satinder P. Singh. Combining memory and landmarks with predictive state representations. In *The 19th International Joint Conference on Artificial Intelligence*, pages 734–739. (IJCAI 2005), 2005.

[Kal60] Rudolf E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basis Engineering*, 82:35–45, 1960.

[KB88] Benjamin J. Kuipers and Yung-Tai Byun. A robust qualitative method for robot spatial learning. In *The Seventh National Conference on Artificial Intelligence*, pages 774–779. (AAAI 1988), 1988.

[KBM06] Armita Kaboli, Michael Bowling, and Petr Musílek. Bayesian calibration for monte carlo localization. In *The 21st National Conference on Artificial Intelligence*. (AAAI 2006), 2006.

[Kui00] Benjamin J. Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119(1–2):191–233, 2000.

[LSS02] Michael Littman, Richard S. Sutton, and Satinder Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems 14*, pages 1555–1561. (NIPS 2001), 2002.

[MTKW02] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *The 18th National Conference on Artificial Intelligence*, pages 593–598. (AAAI 2002), 2002.

[Mur00] Kevin Murphy. A survey of POMDP solution techniques. Technical report, U.C. Berkeley, 2000.

[Oat02] Tim Oates. PERUSE: An unsupervised algorithm for finding recurring patterns in time series. In *The 2002 IEEE International Conference on Data Mining*, pages 330–337. (ICDM 2002), 2002.

[OSC00a] Tim Oates, Matthew D. Schmill, and Paul R. Cohen. Identifying qualitatively different outcomes of actions: Gaining autonomy through learning. In *The Fourth International Conference on Autonomous Agents*, pages 110–111. (Agents 2000), 2000.

[OSC00b] Tim Oates, Matthew D. Schmill, and Paul R. Cohen. A method for clustering the experiences of a mobile robot that accords with human

judgments. In *The 17th National Conference on Artificial Intelligence*, pages 846–851. (AAAI 2000), 2000.

[PK97] David Pierce and Benjamin J. Kuipers. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, 92(1–2):169–227, 1997.

[Put94] Martin L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Incorporated, New York, NY, 1994.

[Rab89] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[RN03] Stuart Russell and Peter Norvig. *Artificial intelligence: A modern approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.

[RS00] Sam Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[RT99] Nicholas Roy and Sebastian Thrun. Online self-calibration for mobile robots. In *The 1999 IEEE International Conference on Robotics and Automation*. (ICRA 1999), 1999.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, Cambridge, MA, 1998.

[SC70] Peter H. Schoenemann and Robert M. Carroll. Fitting one matrix to another choice of a central dilation and a rigid motion. *Psychometrika*, 35(2):245–255, 1970.

[SJR04] Satinder Singh, Michael R. James, and Matthew R. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *The 20th Conference on Uncertainty in Artificial Intelligence*, pages 512–519. (UAI 2004), 2004.

[SOC00] Matthew D. Schmill, Tim Oates, and Paul R. Cohen. Learning planning operators in real-world, partially observable environments. In *The Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems*, pages 246–253. (AIPS 2000), 2000.

[Son78] Edward J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, March 1978.

[SS73] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088, September 1973.

[SS01] Bernhard Scholkopf and Alexander J. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond.* MIT Press, Cambridge, MA, 2001.

[SS05] Daniel Stronger and Peter Stone. Simultaneous calibration of action

and sensor models on a mobile robot. In *The 2005 IEEE International Conference on Robotics and Automation*. (ICRA 2005), 2005.

[STC04] John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, Cambridge, MA, June 2004.

[TdSL00] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[Ten98] Joshua B. Tenenbaum. Mapping a manifold of perceptual observations. In *Advances in Neural Information Processing Systems 10*. (NIPS 1997), 1998.

[TFBD00] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1–2):99–141, 2000.

[TS05] Brian Tanner and Richard S. Sutton. TD(λ) networks: Temporal-difference networks with eligibility traces. In *The 22nd International Conference on Machine Learning*, pages 888–895. (ICML 2005), 2005.

[WBG05] Dana Wilkinson, Michael Bowling, and Ali Ghodsi. Learning subjective representations for planning. In *The 19th International Joint Conference on Artificial Intelligence*. (IJCAI 2005), 2005.

[WS04] Killian Q. Weinberger and Lawrence K. Saul. Unsupervised learning of image manifolds by semidefinite programing. In *The 2004 IEEE*

Computer Society Conference on Computer Vision and Pattern Recognition, pages 988–995. (CVPR 2004), 2004.

[WSS04] Killian Q. Weinberger, Fei Sha, and Lawrence K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *The 21st International Conference on Machine Learning*, pages 839–846. (ICML 2004), 2004.