The Dissertation Committee for Daniel Adam Stronger
certifies that this is the approved version of the following dissertation:

Autonomous Sensor and Action Model Learning for Mobile Robots

Committee:

Peter Stone, Supervisor

Dana Ballard

Benjamin Kuipers

Risto Miikkulainen

Nicholas Roy

Autonomous Sensor and Action Model Learning for Mobile Robots

by

Daniel Adam Stronger, B.A.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2008

Acknowledgments

I would like to acknowledge many people who have helped me over the course of my graduate career. I would especially like to thank my advisor, Peter Stone, for his generous time and commitment. Over the course of my doctoral work, he has taught me how to be a scientist, a researcher, a writer, and a member of the academic community.

I would also like to thank my committee members, Dana Ballard, Benjamin Kuipers, Risto Miikkulainen, and Nicholas Roy, for the extensive time, technical suggestions, and advice that they have given me. Additionally, I would like to extend thanks to the many professors, colleagues, and friends whose suggestions about my work have been truly invaluable: Mazda Ahmadi, Patrick Beeson, Craig Boutilier, Kurt Dresner, Gregory Dudek, Selim Erdogan, Ian Fasel, Peggy Fidelman, Dieter Fox, Todd Hester, Nicholas Jong, Nate Kohl, Gregory Kuhlmann, Daniel Lee, Juhyun Lee, Yaxin Liu, Tekin Mericli, Raymond Mooney, Aniket Murarka, Gregory Plaxton, Michael Quinlan, Mohan Sridharan, Ken Stanley, Jeremy Stober, Matthew Taylor, and Shimon Whiteson.

I would especially like to thank Patrick Beeson for helping me run the autonomous car to collect the data used in the experiments reported in Chapter 5. I am also very grateful to the entire Marvin team from Austin Robot Technology and the University of Texas at Austin for their work in developing the code base that enabled the car to drive autonomously and record its actions and sensations

over time. I would also like to thank the UT Austin Villa team for their work in developing the code base that is used in the experiments reported in Chapters 3 and 4.

Finally, I would like to thank my family for always believing in me and for surrounding me with computers since I was a toddler. I am especially grateful to my wife, Yia, for her unwaivering support and encouragement.

Daniel Stronger

The University of Texas at Austin
August 2008

Autonomous Sensor and Action Model Learning for Mobile Robots

Daniel Adam Stronger, Ph.D.
The University of Texas at Austin, 2008

Supervisor: Peter Stone

Autonomous mobile robots have the potential to be extremely beneficial to society due to their ability to perform tasks that are difficult or dangerous for humans. These robots will necessarily interact with their environment through the two fundamental processes of acting and sensing. Robots learn about the state of the world around them through their sensations, and they influence that state through their actions. However, in order to interact with their environment effectively, these robots must have accurate *models* of their sensors and actions: knowledge of what their sensations say about the state of the world and how their actions affect that state.

A mobile robot's action and sensor models are typically tuned manually, a brittle and laborious process. The robot's actions and sensors may change either

over time from wear or because of a novel environment's terrain or lighting. It is therefore valuable for the robot to be able to autonomously learn these models. This dissertation presents a methodology that enables mobile robots to learn their action and sensor models starting without an accurate estimate of either model.

This methodology is instantiated in three robotic scenarios. First, an algorithm is presented that enables an autonomous agent to learn its action and sensor models in a class of one-dimensional settings. Experimental tests are performed on a four-legged robot, the Sony Aibo ERS-7, walking forward and backward at different speeds while facing a fixed landmark. Second, a probabilistically motivated model learning algorithm is presented that operates on the same robot walking in two dimensions with arbitrary combinations of forward, sideways, and turning velocities. Finally, an algorithm is presented to learn the action and sensor models of a very different mobile robot, an autonomous car.

Contents

Chapter 1

Introduction

One long-term goal of artificial intelligence is the development of physically embodied agents that behave effectively in a wide range of environments without human supervision. The development of these *autonomous robots* has tremendous potential to improve people's lives in a wide range of contexts. Autonomous robots may be able to go places that are too dangerous or difficult for people to go, such as the surface of Mars, where they could collect data or perform experiments, or the rubble of a collapsed building, where they could help find people that are trapped. Additional potential applications include autonomous robots designed to assist people in their homes and autonomous vehicles that can transport people and cargo without human supervision.

As an autonomous robot interacts with its environment, it necessarily relies on two fundamental processes: sensing and acting. In sensing, the robot receives observations through its sensors that are designed to provide information about the *state of the world*: the relevant features of the current environment. In turn, the robot takes actions to bring about desired changes in that same world state. However, in order for the robot to use these sensors and actions effectively, it must have accurate *sensor and action models*: knowledge of how its observations provide in-

formation about the world and knowledge of how its actions correspond to changes in the world state. More precisely, a sensor model is a mapping from the world state to a probability distribution over observations from the sensor. The action model maps the current state of the world and action being executed onto a probability distribution over subsequent world states. These definitions are formalized in Chapter 2.

For a mobile robot, one important aspect of its world state is its pose (position and orientation). For example, one type of mobile robot, which we will discuss in detail in Chapter 5, is a driverless car. An autonomous car's action model maps its actions, such as the position of the throttle, brake, and steering wheel, to the components of its motion, such as its forward acceleration and angular velocity. At the same time, the car's sensors might include a stereo camera or laser range finder, and the sensor model describes the observations that are expected from these sensors as a function of the car's pose (in conjunction with the layout of the environment). These relationships are depicted in Figure 1.1.

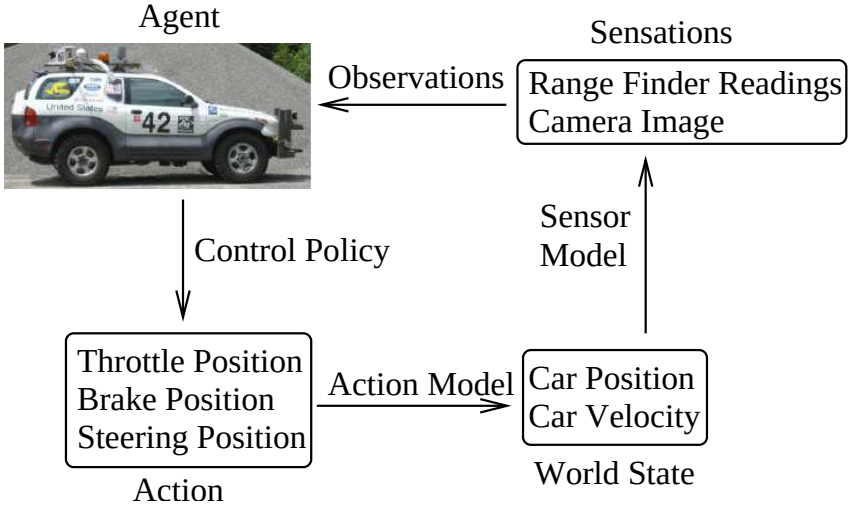


Figure 1.1: A mobile robot's actions influence its world state, while its sensors inform its knowledge of that state. In order for the robot's actions and sensors to be effective and informative, it must have accurate action and sensor models.

One way to construct a robot's action and sensor models is through manual calibration. This process involves systematically recording the robot's observations over the complete range of world states and the change in the world sate caused by each of its actions. Manual calibration can be laborious, inaccurate, and in particular, brittle, due to the possibility of the behavior of the robot's sensors and actuators changing over time because of wear or properties of a novel environment, such as its terrain or lighting conditions. Furthermore, previous work in model learning (discussed in detail in Chapter 6) has typically assumed the presence of an accurate action model to provide labeled training data for the sensor model or vice versa.

This dissertation considers the following question: Is it possible for autonomous mobile robots to learn their action and sensor models without either human supervision or an accurate initial estimate of either model? To demonstrate that achieving this goal is indeed possible, Chapters 3 through 5 present learning algorithms that autonomously learn action and sensor models in three different robotic domains. In each case, experimental results show that the learned models closely match the measured true properties of the actions and sensors.

In order to autonomously learn the robots' action and sensor models, despite having no source of accurate training data for them, the algorithms that are presented leverage redundancy in the combination of the robot's sensory input and its knowledge of its own actions, which we refer to collectively as *perceptual redundancy*. For example, if a robot were equipped with a stereo camera and a laser range finder with known models, they would at times provide redundant information regarding the distance of the robot to obstacles.

Perceptual redundancy in general is depicted in Figure 1.2. The arrows in the diagram represent the robot's *processing modules*, algorithms that convert the robot's raw sensory inputs and action knowledge into useful information about the

state of the world. These algorithms are dependent on the robot's knowledge of its corresponding sensor and action models. Although any one information source by itself may not determine the world state, when taken together, all of a robot's inputs can often provide highly redundant world state information. This redundancy affords the robot a valuable opportunity to learn its action and sensor models from each other.

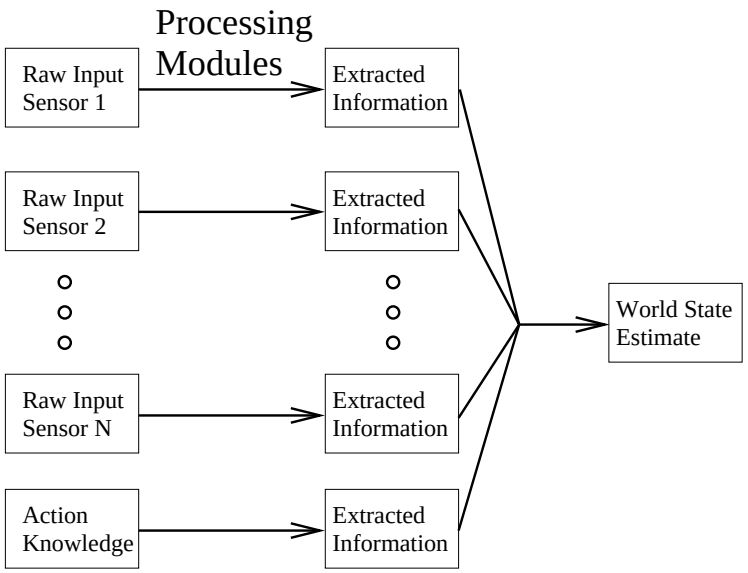


Figure 1.2: Perceptual redundancy provides the robot with multiple sources of information about its world state.

Because the world state can be estimated from redundant sources of information, the robot can use this information to learn its action and sensor models. The learning compares the input to each module to an expected output: an estimate of what that module should return, based on all of the other information sources. Figure 1.3 shows (in an example with one sensor) how the robot can use redundant information to learn about multiple processing modules. In this example, a sensor and the robot's knowledge of its own actions each provide enough information to deduce the state of the environment. Note that for the sensor model, it is the

inverse of the model that is used by the processing module to convert sensations into information about the state of the world. This distinction is discussed in more detail in Chapter 2.

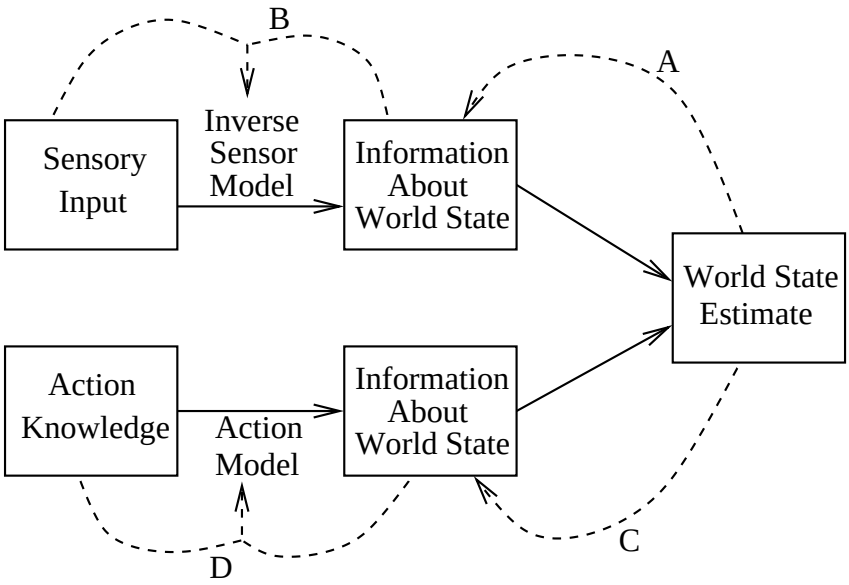


Figure 1.3: Dashed arrows A through D show how information can be propagated back from a redundantly informed world state estimate to learn components of two processing modules.

In this example, the robot learns its sensor model by first relaying a world state estimate based on the action knowledge back through arrow A to the "information about world state" from the sensor. This tells us what the output of the inverse sensor model should have been, assuming the world state estimate is perfectly accurate. When this data is combined with the sensory input through arrow B, the result is training data that can, for example, be used by a supervised learning method to learn components of the sensor model. Simultaneously, a world state estimate based on the sensory input can be relayed back through arrow C to the "information about world state" based on the action knowledge. This information can be combined with the actual action knowledge through arrow D to train the

action model.

The procedure described above represents a useful conceptual plan; however, realizing it algorithmically is not straightforward. Standard supervised learning methods require accurately labeled training data; in the absence of manually labeled data, the robot must combine information from its various sensors to bootstrap accurate processing modules. The resulting challenges are addressed in this dissertation.

Specifically, Chapter 3 introduces a class of settings in which the world state space is one-dimensional. In these settings, the agent has access to a sensor whose reading corresponds to the state of the world and a continuum of actions that correspond to the rate at which the state changes. These two correspondences are the sensor and action models that are learned by the algorithm presented in that chapter. This algorithm is empirically validated on a four-legged robot, the Sony Aibo ERS-7, walking forward and backward at different speeds while facing a fixed landmark.

The work presented in Chapter 3 demonstrates the possibility of autonomously learning a mobile robot's action and sensor models starting without an accurate estimate of either model. Specifically, it enables a robot in a one-dimensional domain to learn both models starting with no sensor model and only a linear approximation for the action model. (Section 3.3.3 discusses the sensitivity of the algorithm to this starting model.) However, mobile robots commonly traverse two-dimensional areas in which they can turn and sometimes move sideways, in addition to moving forward and backward. Such a domain raises many additional challenges for autonomous model learning. These challenges are addressed in Chapter 4. That section presents an adaptation of the EM algorithm [26] to learn parameters of a Kalman filter [44] that correspond to the robot's sensor and action models. The resulting algorithm is empirically validated both on the Sony Aibo walking on a field with known, fixed

landmarks, as well as in a simulation of that same scenario.

The methods presented in Chapters 3 and 4 essentially follow the procedure depicted in Figure 1.3. However, note that this procedure relies on the sensory input and action knowledge each being able to independently provide an estimate of the world state. In Chapter 5, a robotic scenario is examined in which this is not the case. In particular, an algorithm is presented that learns the action and sensor models of an autonomous self-driving car. In this work, the learning process starts with no knowledge of the map of the environment. Treating the map as a component of the world state, this means that the robot's action knowledge does not by itself contain enough information to determine the world state sufficiently to learn a sensor model. However, the sensor that is modeled is a high-bandwidth three-dimensional laser range finder. This sensor provides so much information that it contains a significant degree of redundancy by itself. This redundancy is used to first learn a sensor model, which is then used to learn an action model.

Chapter 6 discusses the wide range of previous work that is related to the contributions of this dissertation. Finally, Chapter 7 discusses these contributions and possibilities for future work.

Chapter 2

Background

This chapter presents background information that is needed for the algorithms and empirical evaluations presented in Chapters 3 through 5. First, recall from Chapter 1 that a robot's sensor and action models are described as mappings onto probability distributions. Since the domains discussed in this work have continuous state and observation spaces, these distributions are represented by probability density functions, which are described in Section 2.1. In Section 2.2, the sensor and action models are formally defined as conditional probability distributions. Normal distributions and Kalman filtering are discussed in Sections 2.3 and 2.4. Finally, Section 2.5 describes the robotic platforms used in the experiments reported in this thesis.

2.1 Probability Density Functions

A probability distribution over a continuous variable, $x \in X$ can usually be represented by a *probability density function* (PDF),[1] a mapping $P : X \mapsto [0, \infty)$. The

[1]Probability distributions that concentrate a non-zero amount of probability in a region with volume zero can not be represented by a PDF; however, any distribution can be arbitrarily closely approximated by distributions that do have PDFs.

key property of this mapping is that for any set $A \subseteq X$, the probability that x is in A is given by:

$$p(x \in A) = \int_A P(x)\, dx \tag{2.1}$$

where we use lowercase p for discrete probabilities, which range from 0 to 1. In particular, taking $A = X$ yields $\int_X P(x)\, dx = 1$. Note that if x is an n-dimensional vector, the integral in Equation 2.1 is used as a shorthand for:

$$\underbrace{\iint \cdots \int_A}_{n \text{ integrals}} P(x)\, dx_1 dx_2 \ldots dx_n \tag{2.2}$$

If the region of integration A is omitted, each integral is taken to be from $-\infty$ to ∞.

Note that if x represents a physical quantity such as a length, the magnitude of $P(x)$ depends on the units in which x is measured. For example, consider a length that is uniformly distributed between five and six centimeters. If x is that length in centimeters, its probability density is 1 for $5 < x < 6$ and 0 elsewhere. On the other hand, if x represents the length in meters, its density is 100 for $\frac{5}{100} < x < \frac{6}{100}$ and 0 elsewhere.

Additionally, a conditional probability density function can be defined in analogy with a discrete conditional probability:

$$P(x|y) = \frac{P(x, y)}{P(y)} \tag{2.3}$$

This conditional density satisfies:

$$p(x \in A|y) = \int_A P(x|y)\, dx \tag{2.4}$$

Again taking $A = X$ yields $\int_X P(x|y)\, dx = 1$.

In the following section, sensor and action models are defined as conditional probability density functions.

2.2 Sensor and Action Models

As mentioned in Chapter 1, a sensor model is a mapping from the world state to a probability distribution over observations reported by the sensor. Formally, denoting the world state and observation as w and o respectively, the sensor model is the conditional distribution $P(o|w)$. As desired, for any given world state the sensor model specifies a probability distribution over the space of possible observations.

To define the action model, note that in a deterministic continuous-time system, the rate of change of the state of the world is a function of the action being taken and the world state:

$$\frac{dw}{dt} = A(c(t), w) \tag{2.5}$$

where A is the action model and $c(t)$ is the action command being executed at time t. A can also be thought of as the process dynamics, holding the action constant. If the random noise in the process is taken into account, the result is a continuous-time stochastic process that varies based on the action command. In the work presented in this dissertation, however, time is treated as discrete. In a discrete-time stochastic process, the action model can be thought of as a mapping from each state-action pair to a probability distribution over possible next states, as mentioned in Chapter 1. The action model is therefore represented by the conditional probability density function $P(w_{t+1}|w_t, c_t)$, where w_t and c_t represent the world state and action command executed at time t.

2.3 Normal Distributions and Linear Regression

One commonly used probability distribution is the multivariate normal (Gaussian) distribution. In n dimensions, each such distribution is determined by a mean n-vector μ and an $n \times n$ positive-semidefinite covariance matrix Σ. An n-dimensional random variable x having this distribution is indicated by: $x \sim \mathcal{N}(\mu, \Sigma)$. If Σ is non-singular, this distribution can be represented by the following PDF:

$$P(x) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right) \tag{2.6}$$

This PDF can be used to prove the following properties of normal distributions, used in Chapter 4. If $x_1 \sim \mathcal{N}(\mu_1, \Sigma_1)$ and $x_2 \sim \mathcal{N}(\mu_2, \Sigma_2)$ are independent random n-vectors and A is an $m \times n$ matrix:

$$Ax_1 \sim \mathcal{N}(A\mu_1, A\Sigma_1 A^\top) \tag{2.7}$$

$$x_1 + x_2 \sim \mathcal{N}(\mu_1 + \mu_2, \Sigma_1 + \Sigma_2) \tag{2.8}$$

Real-world probability distributions can often be closely approximated as Gaussian. One reason for this is the central limit theorem: The distribution of a sum of N independent, identically-distributed random variables approaches a normal distribution as N approaches infinity. Additionally, Gaussian distributions are often used for their analytical tractability. For example, techniques that minimize a sum of square errors, such as linear regression, can be thought of as maximizing a likelihood where measurements are perturbed by Gaussian noise in the output variable.

Specifically, least squares regression, a technique used repeatedly in Sections 3 through 5, can be thought of as the solution to the following problem. An output m-vector, y, is generated from an input $m \times n$ matrix, X, by the equation $y = X\beta + \epsilon$, where the random noise $\epsilon \sim \mathcal{N}(0_m, \sigma^2 I_m)$ and β is an n-vector of un-

known coefficients. Given X and y, find the value of β that maximizes the likelihood of y, $P(y|X,\beta)$.

To solve this problem, note that given X and β, $y \sim \mathcal{N}(X\beta, \sigma^2 I_m)$. Since $|\sigma^2 I_m| = \sigma^{2m}$ and multiplying by $(\sigma^2 I_m)^{-1}$ is equivalent to dividing by σ^2,

$$P(y|X,\beta) = \frac{1}{(2\pi)^{m/2}\sigma^m} \exp\left(-\frac{1}{2\sigma^2}(y - X\beta)^\top (y - X\beta)\right) \tag{2.9}$$

$$= \frac{1}{(2\pi)^{m/2}\sigma^m} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{m}(y_i - (X\beta)_i)^2\right) \tag{2.10}$$

Maximizing this likelihood with respect to β is equivalent to minimizing $\sum_{i=1}^{m}(y_i - (X\beta)_i)^2$. Noting that $(X\beta)_i = \sum_{j=1}^{n} X_{i,j}\beta_j$, this is the standard formulation of least squares regression. If $(X^\top X)$ is invertible, the maximum likelihood value, $\hat{\beta}$, is given by [89]:

$$\hat{\beta} = (X^\top X)^{-1}(X^\top y) \tag{2.11}$$

In Chapters 3 through 5, the use of least squares regression can therefore be thought of as an implicit use of normal distributions. Weighted least-squares regression, used in Section 3.2.3, is motivated by replacing $\sigma^2 I_m$ above with a diagonal matrix of different variances (the reciprocals of the weights), one for each data point. Additionally, multivariate normal distributions are used explicitly to represent the world state uncertainty in Chapter 4. In a setting in which the initial state distribution, sensor model, and action model are represented by normal distributions, the *a posteriori* state distributions can be estimated by Kalman filtering, described in the following section.

2.4 Kalman Filtering

Consider a setting where the initial world state, w_0, is drawn from a known normal prior distribution and where a series of observations, o_t, and the process dynamics are linear but perturbed by Gaussian noise:

$$
\begin{aligned}
w_0 &\sim \mathcal{N}(\mu_0, \Sigma_0) && (2.12)\\
w_t &= Aw_{t-1} + q_{t-1} && (2.13)\\
o_t &= Hw_t + r_t && (2.14)\\
q_t &\sim \mathcal{N}(0, Q) && (2.15)\\
r_t &\sim \mathcal{N}(0, R) && (2.16)
\end{aligned}
$$

where μ_0 and Σ_0 define the prior distribution on the initial state, and A, H, Q, and R are matrices that specify the behavior of the system. Note that if a constant offset vector is added to the righthand side of Equation 2.13 or 2.14, it can be eliminated by appending it to the right edge of A or H and adding a final component to w that always equals 1.

In such a system, the *a posteriori* distribution of w_t can be determined by employing a Kalman filter [44, 99]. Specifically, if the prior distribution over w_t is normal with mean μ_t and covariance Σ_t, the posterior distribution that takes the observation, o_t, into account is also normal, with mean μ'_t and covariance Σ'_t given by the Kalman filter *measurement update*:

$$
\begin{aligned}
K_t &= \Sigma_t H^\top (H\Sigma_t H^\top + R)^{-1} && (2.17)\\
\mu'_t &= \mu_t + K_t(o_t - H\mu_t) && (2.18)\\
\Sigma'_t &= (I - K_t H)\Sigma_t && (2.19)
\end{aligned}
$$

Once μ'_t and Σ'_t are known, they can be used to determine the prior distribution at time $t+1$, represented by μ_{t+1} and Σ_{t+1} through the Kalman filter *time update*:

$$\mu_{t+1} = A\mu'_t \tag{2.20}$$

$$\Sigma_{t+1} = A\Sigma'_t A^\top \tag{2.21}$$

However, if the system is nonlinear or dependent on agent's action commands, c_t, Equations 2.13 and 2.14 must be modified accordingly:

$$w_t = a(w_{t-1}, c_{t-1}) + q_{t-1} \tag{2.22}$$

$$o_t = h(w_t) + r_t \tag{2.23}$$

where functions a and h now represent the system dynamics and random vectors q_t and r_t satisfy Equations 2.15 and 2.16 as before. In order to apply the Kalman filter to a nonlinear system, the *extended Kalman filter* (EKF) employs a first-order approximation to Equations 2.22 and 2.23:

$$w_t \approx a(\mu'_{t-1}, c_{t-1}) + A_{t-1}(w_{t-1} - \mu'_{t-1}) + q_{t-1} \tag{2.24}$$

$$o_t \approx h(\mu_t) + H_t(w_t - \mu_t) + r_t \tag{2.25}$$

where A_t and H_t are the Jacobian matrices:

$$A_t = \frac{\partial a}{\partial w}(\mu'_t, c_t) \tag{2.26}$$

$$H_t = \frac{\partial h}{\partial w}(\mu_t) \tag{2.27}$$

Note that the righthand sides of Equations 2.24 and 2.25 only differ from those of (2.13) and (2.14) by known offsets, namely $a(\mu'_{t-1}, c_{t-1}) - A_{t-1}\mu'_{t-1}$ and $h(\mu_t) - H_t\mu_t$ respectively. As discussed above, such offsets can be incorporated into A_t and H_t by adding a final component to w that is always 1. Equations 2.17 through 2.21 can thus be used with A_t and H_t to execute the EKF.

One common use for the extended Kalman filter is in *mobile robot localization* [12, 96]. The EKF's ability to estimate a probability distribution over robot poses is a critical component of the model learning technique described in Chapter 4.

2.5 Robotic Platforms

This section describes the two robotic platforms used in the experiments reported in this dissertation, the Sony Aibo ERS-7 and the autonomous car developed by Austin Robot Technology.

2.5.1 The Sony Aibo ERS-7

In Chapters 3 and 4, experiments are performed on a Sony Aibo ERS-7. The robot is roughly 280 mm tall and 320 mm long. It has 20 degrees of freedom: three in each of four legs, three in the neck, and five more in its ears, mouth, and tail. At the tip of its nose there is a CMOS color camera that captures images at 30 frames per second in YCbCr format. The images are 208×160 pixels giving the robot a field of view of 56.9° horizontally and 45.2° vertically. The robot is depicted in Figure 2.1. In the robot's coordinate system, the positive x-axis points to its right, positive y points

forward, and positive z points up. The origin of the Aibo's coordinate system is at the center of its body. The robot's world state is its pose in the world coordinate system, $w = (x, y, \theta)^\top$.

Figure 2.1: The Sony Aibo ERS-7 has four legs that each have three degrees of freedom, three additional degrees of freedom in its neck, and a color camera in its nose.

The Aibo's basic acting and sensing capabilities used in Chapters 3 and 4 were developed by the UT Austin Villa robot soccer team as part of the international RoboCup research initiative [82]. These capabilities and their underlying algorithms are detailed in the UT Austin Villa 2003-2006 technical reports [84, 85, 86, 87].[2]

Action Model

The different actions taken by the Aibo are walking motions that cause the robot to move in combinations of forward, sideways, and turning velocities. These walk-

[2]Modifications made in 2006 [87] were not used in the experiments presented in Chapter 3.

ing motions are parameterized by a set of *attempted velocities*, (a_x, a_y, a_θ), that are converted into the robot's leg motions through a "virtual wheel" technique that determines the sizes and directions of the steps taken by the robot's four legs [40]. These attempted velocities, however, are not exactly the same as the robot's corresponding actual forward, sideways, and turning velocity components, because of unmodeled properties of the robot's joints and friction with the ground. This correspondence between actions and resulting velocities is the aspect of the action model that is learned in this work.

Specifically, the action model function, A, maps each action c_t onto a set of actual velocities with respect to the robot's coordinate system, $A(c_t) = (v_x, v_y, v_\theta)^\top$. Each possible velocity vector corresponds to a conditional probability distribution over subsequent world states, as discussed in Section 2.2, as follows. If at time t the robot is at pose $w_t = (x_t, y_t, \theta_t)^\top$, then the pose at time $t+1$ is modeled as:

$$w_{t+1} \sim \mathcal{N}(w_t + R(\theta_t)(A(c_t)\Delta t), \Sigma_m) \tag{2.28}$$

where Δt is the amount of time between consecutive time steps, $R(\theta)$ represents a counterclockwise rotation through an angle of θ,[3] and Σ_m represents the covariance matrix of the zero-mean Gaussian white noise in the motion. Although Σ_m may be different for each action, these values are not learned in this work and a fixed, constant value is used for Σ_m, as discussed in Section 4.4.1.

In Chapter 4, the actual velocities $A(c_t)$ that result from each of a set of 40 discrete actions are learned. These results could additionally be interpolated for intermediate actions [28, 68]. In Chapter 3, only actions that walk directly forward or backward are considered ($a_x = a_\theta = 0$), and the action model function A maps the attempted forward velocity, a_y, to the actual one: $v_y = A(a_y)$.

[3] Because rotations are only applied to velocities and pose differences in this work, the rotation $R(\theta)$ does not affect the third (θ) component of the vector on which it acts.

Sensor Model

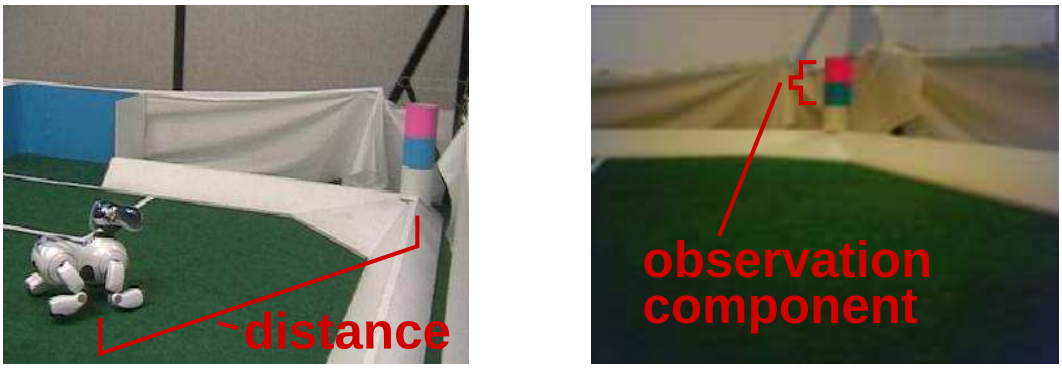
The Aibo's observations are based on its ability to recognize color-coded cylindrical landmarks in its camera's field of view, like the one shown in Figure 2.2. These landmarks are in fixed locations that are known by the robot. Each landmark observation has two components. The first, denoted as o_1, is the landmark's height in the image in pixels, as depicted in Figure 2.2b). The second component, o_2, is the robot's estimate of the landmark's horizontal angle from the robot, computed based on the landmark's horizontal position in the image and the pan angle of the robot's head. Recall from Section 2.2 that the robot's sensor model specifies a probability distribution over observations that depends on the world state. If the Aibo is at a distance d from the landmark, as depicted in Figure 2.2a), and a horizontal angle of α, the sensor model specifies that the corresponding observation distribution is:

$$o = \begin{pmatrix} o_1 \\ o_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} f(d) \\ \alpha \end{pmatrix}, \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix}\right) \tag{2.29}$$

where the function f and constants σ_1 and σ_2 are the unknown parameters of the sensor model.

The Aibo's camera specifications can be used to derive an estimate of $f(d)$, namely $f(d) = 38388.7/d$, where d is measured in millimeters. However, experimental results show that this estimate is not exactly correct (see Section 4.4). In this work, we take the approach of starting the learning process with no *a priori* knowledge of what functional form f might have.

Given a sequence of observations and actions, one way to estimate the world state over time is through Kalman filtering, as discussed in Section 2.4 and Chapter 4. However, in a scenario where any one observation provides enough information to estimate the world state accurately, another option is to use an *inverted sensor model*: a mapping from the observation to the world state. This is the case in



a) b)

Figure 2.2: a) The Aibo faces a color-coded, cylindrical landmark. Its distance to the landmark is an important aspect of the world state. b) The landmark is depicted from the robot's point of view. Its height in pixels in the image is a component of each landmark observation.

Chapter 3, where the robot's distance to a landmark is the entire world state. In this case, we learn a function $S = f^{-1}$ through polynomial regression (note that f is naturally monotonically decreasing and thus invertible). The condition that f is invertible restricts the *perceptual aliasing* that the domain can have, where two states yield the same observation, to be only that caused by random noise. This assumption is relaxed in the algorithm presented in Chapter 4, which does not rely on f being invertible.

2.5.2 The Autonomous Car

Chapter 5 addresses the challenge of learning a mobile robot's action and sensor models on a second robotic platform: a self-driving car. The autonomous car used in this work is a modified 1999 Isuzu VehiCross, shown in Figure 2.3. The car's software can control its throttle, brake, and steering wheel, and has access to sensors that enable the car to navigate and avoid obstacles. This section describes the car's action and sensor models and the parameters of these models that are learned in this work. The algorithms that control the car and store its observations are described

in the Austin Robot Technology technical reports [9, 16, 83].

Figure 2.3: The autonomous car.

The car's model learning process does not require prior knowledge of the map of the environment. This map is considered part of the state of the world. Additionally, the car's world state at time t is taken to include its forward velocity, v_t, as well as its position and orientation, p_t and θ_t, enabling the action model to specify the car's forward acceleration, as discussed in the following section.

Action Model

There is a wide range of previous work in modeling a car's motion as a function of the throttle, brake, and steering wheel positions. For example, Sayers and Han present a multibody dynamic model that predicts the car's motion [74, 73]. In this section, a highly simplified car model is presented, with six parameters that are the aspect of the car's action model learned in Chapter 5. Despite the model's simplicity, it is able to accurately model the car's motions (see Section 5.4 for experimental results). Learning a simple but accurate model is made possible by a judicious choice of the model features described below. An interesting area for future work, discussed in Chapter 7, is the automation of the feature selection process.

The car's action command at time t, c_t, is the combination of the positions of the throttle, g_t, the brake, b_t, and the steering wheel, s_t. The action model specifies the car's corresponding forward acceleration, a_t, and angular velocity, ω_t,

in accordance with the following simplifying assumptions.

First, if the throttle is held at a fixed position, the car will approach a corresponding steady state velocity, which is approximated as a linear function of the throttle position: $C_1 + C_2 g_t$. A further approximation is that the car's acceleration is proportional to the difference between its current velocity and this steady state velocity: $C_3(C_1 + C_2 g_t - v_t)$. However, this acceleration does not take into account the deceleration caused by braking, which is presumed to be proportional to the brake position and the car's velocity: $C_4 v_t b_t$ (when the velocity is zero, so is the deceleration from braking). In summary, we can write the acceleration model as:

$$a_t = C_5 + C_6 g_t + C_7 v_t + C_8 v_t b_t \tag{2.30}$$

where C_5 through C_8 are determined by C_1 through C_4 and vice versa. Furthermore, the car's angular velocity is modeled as proportional to its forward velocity and as a linear function of the steering wheel position:

$$\omega_t = C_9 v_t + C_{10} v_t s_t \tag{2.31}$$

The coefficients C_5 through C_{10} are the parameters of the action model that are learned in this work. Since a and ω are the outputs of a linear regression discussed in Section 5.3, the corresponding assumption is that zero-mean Gaussian white noise is added to the righthand side of Equations 2.30 and 2.31. Finally, the car's pose and forward velocity, x, y, θ, and v, satisfy:

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \end{pmatrix} + R(\theta_t) \begin{pmatrix} 0 \\ v_t \Delta t \end{pmatrix} \tag{2.32}$$

$$\theta_{t+1} = \theta_t + \omega_t \Delta t \tag{2.33}$$

$$v_{t+1} = v_t + a_t \Delta t \tag{2.34}$$

Sensor Model

The sensor modeled in this work is the Velodyne HDL-64E,[4] depicted in Figure 2.4a). The Velodyne consists of n lasers that each report the distance to the nearest obstacle along the direction it is facing at a frequency f. Each laser is positioned at a unique vertical angle, while the entire set of lasers rotates around a vertical axis at a known angular speed, ω, with respect to the Velodyne base, which is affixed to the top of the car.[5] Each laser is additionally pointed at a different horizontal angle inside the rotating unit, denoted as θ_l for $l = 0$ to $n - 1$. The lasers are modeled as all emanating from the same point at the center of the Velodyne; this approximation relies on the assumption that the sensors' reported distances to obstacles are much larger than any distances between lasers inside the Velodyne. An example of the data obtained from the sensor is shown in Figure 2.4b).

a) b)

Figure 2.4: a) The Velodyne HDL-64E. b) An example of the data reported by the sensor.

On the Velodyne, n is 64, f is 7800 Hz, and ω is 20π rad/s, corresponding to a rate of 600 rpm. The lower 32 lasers' vertical angles are evenly spaced from $-24.7°$ to $-8.9°$ (spaced every $0.512°$), and the upper 32 range from $-8.5°$ to $+2.0°$ (spaced every $0.340°$). The horizontal angles, on the other hand, can differ for different units,

[4] `www.velodyne.com/lidar/`

[5] Although the car undergoes moderate pitching and rolling while driving, these rotations are not modeled. That is, the Velodyne base is assumed to remain horizontal at all times.

and they are the aspect of the sensor model that is learned in this work, starting without any *a priori* knowledge about these horizontal angles.

We denote the lth laser's vertical angle from horizontal as ϕ_l and its horizontal angle as θ_l. The lasers are ordered by vertical angle so that the ϕ_l are strictly increasing. In each distance reading, the distance, d, is reported as well as the current angle of the Velodyne with respect to its base, θ_r. Finally, we denote the constant angle between the base and the car as θ_b. These horizontal angles are depicted in Figure 2.5. The car's coordinate system is centered at the Velodyne center and oriented so that positive x, y, and z point rightward, forward, and up respectively. Although the car may experience moderate pitching and rolling in the course of driving, these rotations are not included in the model. The coordinates of the point observed by the laser in the car's coordinate system are given by:

$$(x, y, z) = (d\cos(\phi_l)\cos(\theta_l + \theta_r + \theta_b), d\cos(\phi_l)\sin(\theta_l + \theta_r + \theta_b), d\sin(\phi_l)) \quad (2.35)$$

The algorithm does not rely on knowing θ_b, enabling the Velodyne base to be affixed to the roof of the car at an arbitrary and unknown angle. If it is not known originally, the quantity θ_b is fundamentally unknowable, as increasing θ_b and decreasing all of the θ_l by the same amount leaves the expression (2.35) unchanged. Therefore, our goal is to learn the n values of $\theta_l + \theta_b$ over the n values of l, which we denote as $\hat{\theta}_l$. Note that these values can be thought of as the angles of the lasers with respect to the car when the Velodyne reports a θ_r of 0.

This chapter presented mathematical preliminaries and robot models that define the problems that this dissertation aims to solve. The following three chapters apply the methodology proposed in Chapter 1, yielding algorithms that are able to learn action and sensor models, starting without an accurate estimate of either, in three different mobile robot scenarios. These algorithms are the technical

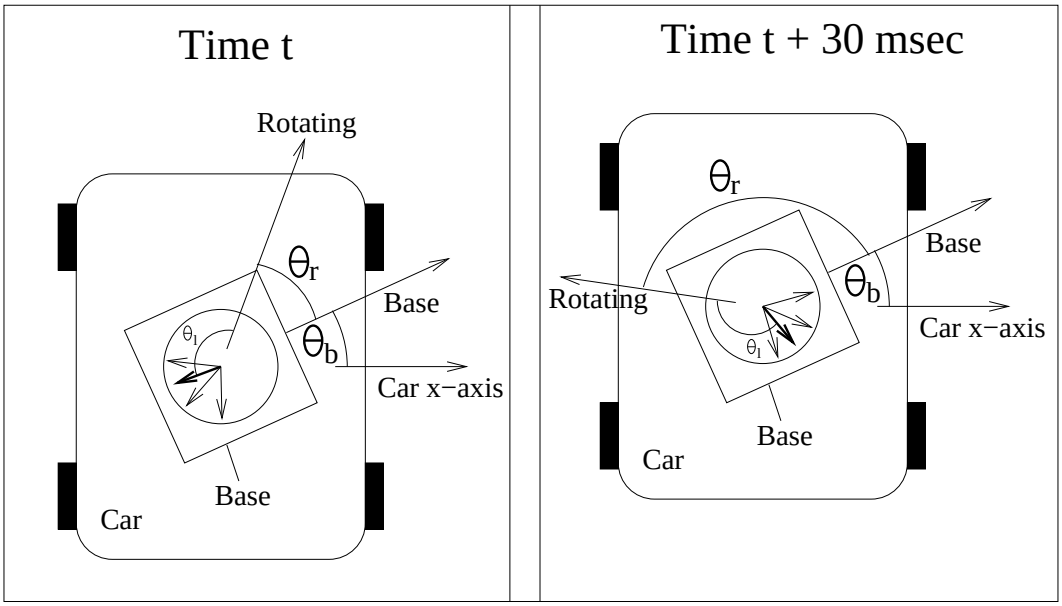


Figure 2.5: Top view of the relative horizontal angles between the laser, the Velodyne, and the car. As the car moves, θ_r, which is continually reported by the sensor, varies as the unit rotates. The angle of the base of the velodyne, θ_b, is constant but unknown, and the n values of θ_l are what the algorithm aims to learn.

contributions of the dissertation.

Chapter 3

Model Learning in One Dimension

Chapter 1 presented a general framework by which a mobile robot can learn its action and sensor models starting without an accurate estimate of either model. In this chapter, that framework is realized by a concrete algorithm on a class of one-dimensional autonomous agent settings [90]. This class of settings is introduced and discussed in Section 3.1. The algorithm presented in Section 3.2 serves as an initial concrete instantiation of the model-learning plan depicted in Figure 1.3. Experimental results described in Section 3.3 provide empirical validation of the idea that simultaneously training the action and sensor models *from each other* can yield accurate models. In contrast to the relatively limited class of the settings discussed in this chapter, a general probabilistic framework for model learning is presented in Chapter 4.

3.1 Setup

This chapter considers the class of autonomous agent settings that satisfy the following properties, depicted in Figure 3.1:

- The set of possible states of the world can be characterized as a one-dimensional, continuous state space.
- The agent has one sensor that converts the state of the world into numerical input data.
- The agent has access to a continuum of actions that it can take. Each action maps onto a single rate of change in the state of the world over time.
- The effects of the actions and the sensor readings are perturbed by zero-mean, random noise.

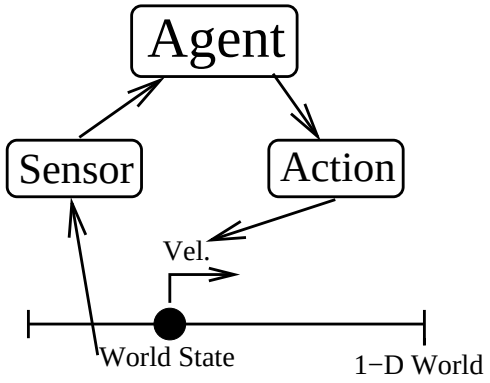


Figure 3.1: The agent interacts with a one-dimensional world state via a sensor and a continuum of actions.

Even in this somewhat restricted setting, there are many interesting domains that satisfy these conditions. One straightforward example is a robot on a one-dimensional track that has a global positioning sensor and takes actions that control its velocity along the track. Another example is a temperature regulator that can

sense the temperature in a room and set the rate at which the temperature changes. Yet another example is where the velocity of a vehicle is the state and the agent has access to a velocity sensor and a throttle whose settings correspond to accelerations. The algorithm presented in this chapter is validated on an approximation of the first setting (a robot on a track) in the empirical results presented in Section 3.3.

In the context of this class of settings, a model of the sensor consists of a function from the state of the world to the corresponding observations. The action model is learned as a function from the selected action to the rate of change of the state of the world. The agent learns both models at once by simultaneously performing the following three operations.

1. Exploring the state space of the world, covering the entire range of world states and state velocities.

2. Learning a function from action commands to actual state velocities, assuming the sensor model is accurately calibrated.

3. Learning a function from sensor readings to the actual state of the world, assuming the action model is accurately calibrated.

For operation 1, any action policy that allows the agent to experience the full range of possible combinations of states and velocities will suffice. An example of such a policy is described in Section 3.3. Section 3.2 describes in detail how to perform operations 2 and 3, first individually, and then simultaneously via a bootstrapping process.

As the agent interacts with its domain, it has two sources of information about its location along its axis of movement. For one, the agent receives a sequence of sensor observations, o_t at time t. Recall from Section 2.5.1 that the observations are modeled as $o_t \approx f(w_t)$, where the function f specifies a typical observation resulting from world state w_t and the actual observations are perturbed by zero-

mean random noise. In this chapter, we restrict our attention to situations in which f is invertible. As mentioned in Section 2.5.1, this restriction implies that the domain has no perceptual aliasing except for that caused by random noise, an assumption that is relaxed in Chapter 4. We denote the inverse function, f^{-1}, as S and refer to it as the sensor model, so that on average $w_t = S(o_t)$. This function S is one of the two functions that the agent is trying to learn. This relationship is depicted in Figure 3.2.

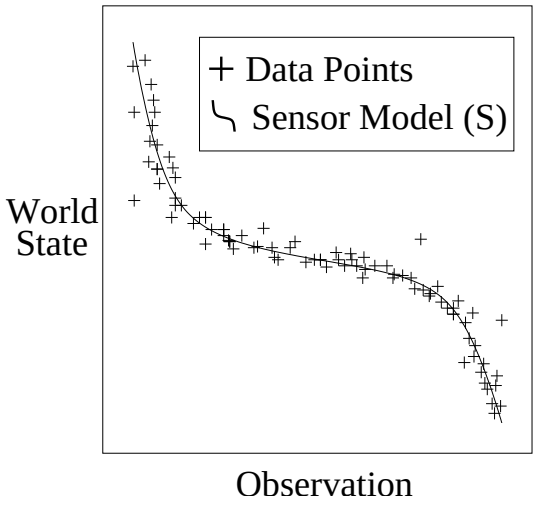


Figure 3.2: The sensor model S maps observations onto the corresponding world states: $w_t = S(o_t)$.

At the same time, the agent executes action command c_t at each time step t. Each action command changes the world state at a specific velocity, and the action model is represented by the function A, which maps action commands onto the mean velocity at which the robot moves when executing that command. This action model function A is the function that the agent learns along with the sensor model function S. The action model also provides an estimate of the state of the world: $w_t = w_0 + \sum_{i=0}^{t-1} A(c_i)\Delta t$, where Δt is the amount of time between time steps. For example, if the agent executes a piecewise constant series of actions, each action

c will cause the robot to move at a constant velocity, $A(c)$, over the corresponding period of time, so the world state will vary in a continuous, piecewise linear manner with respect to time. This scenario is depicted in Figure 3.3.

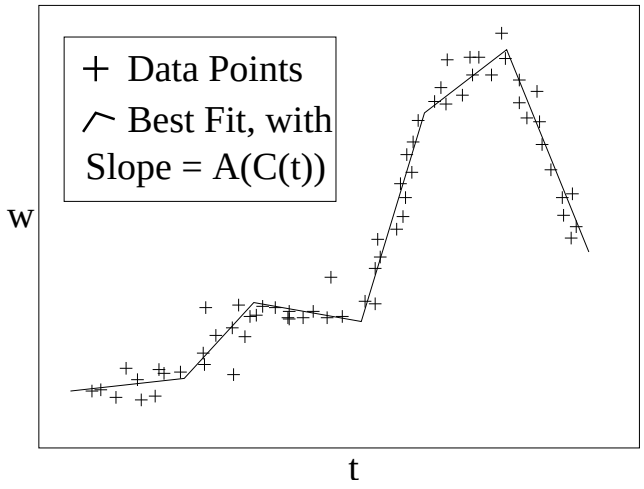


Figure 3.3: When the robot executes a piecewise constant series of actions, the rate of change of the world state is depicted by the slopes of the lines here. These slopes are equal to $A(c_t)$, where c_t is the action command being executed at time t and A is the action model. The data points here represent the agent's estimates of the world state over time.

Because the agent is trying to learn two arbitrary continuous functions, it must represent them with a function approximator. We use polynomial regression for both functions. From among the many function approximator systems that exist, we chose to use polynomial regression for both functions due to its robustness, versatility, and simplicity. Specifically, it can filter out random noise, closely approximate any continuous function, and incorporate new data points efficiently. Initially, the degrees of the polynomials for the sensor and action models are chosen manually, although Section 3.3.3 discusses the possibility of further automating the process by choosing the degree itself on-line.

The agent learns the sensor model by identifying the coefficients s_0 through s_d such that the polynomial $S(o_t) = \sum_{i=0}^{d} s_i o_t^i$ approximates the corresponding w_t

as closely as possible, where d is the degree of the polynomial being fit to the data. Similarly, the agent learns coefficients a_0 through a_d for the action model, with the goal of $A(c_t) = \sum_{i=0}^{d} a_i c_t^i$ approximating the corresponding velocities over the range of actions c_t.

The algorithm presented in this chapter works by implicitly performing a continual comparison between the information from the action and sensor models. Specifically, since they are used to calculate the same location function w_t, we can set them equal, yielding:

$$S(o_t) = w_0 + \sum_{i=0}^{t-1} A(c_i)\Delta t \tag{3.1}$$

Although both sides of the equation are perturbed by random noise, because the noise is unbiased, the equation is true *in expectation*, and can therefore be used to learn the action and sensor models. The agent knows the values of o_t and c_t at each time, and its task is to learn the functions A and S.

The algorithm presented in Sections 3.2.1-3.2.3 learns the action and sensor models *from each other*, without requiring as input any ground truth knowledge of the actual world state or its rate of change. Therefore, it does not learn the two models in any particular units. The sensor model maps observations onto points on a linear axis, but it makes no claims as to what particular state corresponds to the number zero, or what magnitude in the domain's single dimension corresponds to the model's unit. For instance, if the domain is a temperature, the model's unit could correspond to any fixed number of degrees. Similarly, the action model is also learned in arbitrarily units, although here the number zero is constrained to correspond to a rate of change of zero.

Despite this lack of grounding to actual units, the learned action and sensor models are consistent with one another. That is, the ratio between the actual velocities and the corresponding learned velocities is considered the units of length

in which the action model was learned. These are the same units in which the sensor model represents distances. Note that this property is sufficient for the agent to perform domain-specific planning, for instance by predicting the amount of time a specific action command will take to yield a certain visual sensor reading. In other words, we enable the agent to learn models in its own natural frame of reference, rather than imposing upon it human units such as meters or meters per second.

3.2 Methods

This section presents the algorithm that is used to learn the action and sensor models starting without an accurate estimate of either model. Specifically, Sections 3.2.1 and 3.2.2 show how each model can be learned by assuming the other one is already accurate. The remaining question of how the agent can learn both models simultaneously is addressed in Section 3.2.3.

3.2.1 Learning the Sensor Model

First we demonstrate that it is possible to learn a relative sensor model given any *constant* action, even in the absence of any knowledge about that action. We will then generalize to the case of varying actions, assuming access to an accurate action model. Section 3.2.3 shows how this ability can be incorporated into a process that can learn both models from scratch.

Consider the situation in which the agent executes a constant action. Recall that the sensor model is a function from the observations, o_t, to the corresponding world states, w_t. Furthermore, while the agent executes a constant action command, c, the state changes at a constant rate, $A(c)$. Thus if this command is executed continuously starting at time 0, the state of the world at time t_k will be given approximately by $w_t = w_0 + tA(c)$. As long as the constant $A(c)$ is not zero, solving for t yields $t = (w_t - w_0)/A(c)$. This expression represents a shifted and scaled

version of the world state at time t. Furthermore, since our goal is only to learn a sensor model up to shifting and scaling, it suffices to learn a function directly from o_t to t. Such a function will represent a relative sensor model. Thus a satisfactory sensor model can be learned even without knowing the constant rate of change $A(c)$. The agent learns the function by performing polynomial regression on the pairs (o_t, t).

Polynomial regression is implemented as a special case of multivariate linear regression, discussed in Section 2.3. Specifically, to fit a d-degree polynomial to points (x_i, y_i), Equation 2.11 is employed where the $d+1$ columns of X correspond to the powers of x_i: $X_{i,j} = x_i^{j-1}$. Notably, although X and y grow in size as data points are received, the regression can be performed in constant space, taking constant time to incorporate each new data point by instead storing $X^\top X$ and $X^\top y$, which suffice to compute β as in Equation 2.11. This is possible because $(X^\top X)_{i,j} = \sum_k X_{k,i} X_{k,j}$ and $(X^\top y)_i = \sum_k X_{k,i} y_k$, so these entries can simply be incremented as needed for each new data point. This computation, when applied to the data (o_t, t), identifies a suitable sensor model under the restrictive assumption that the agent is executing a constant action command.

With access to an accurate action model, it is possible to use a very similar process to learn a sensor model while the agent performs an arbitrary series of actions. Specifically, since the sensor model is a function from the observations to the world state, if the agent can compute a world state estimate from its actions, it can perform polynomial regression on its observations and those state estimates to learn a sensor model. Given an action model A, the agent can use its knowledge of the state velocities to compute the world state as a function of time. As discussed in Section 3.1, the world state w_t is approximated by $w_0 + \sum_{i=0}^{t-1} A(c_i)\Delta t$, which we denote as $w_{a,t}$, the estimate of the world state at time t based on the action model. Since the algorithm is learning relative distances, it suffices to assume that

$w_0 = 0$. Thus the agent can maintain an estimate for w_t by initializing it to be 0 at time 0 and incrementing it by $A(c_t)\Delta t$ at each time step. The agent then learns a sensor model from the action model by performing polynomial regression on the pairs $(o_t, w_{a,t})$.

3.2.2 Learning the Action Model

In the previous section, we showed how to learn a sensor model when given an accurate action model. In this section, we show that the reverse is also possible: we assume that the agent has an accurate sensor model and show how the agent can use it to learn an action model. This learning uses the sensor model to provide an estimate of the state of the world from each observation. We denote this estimate by $w_{s,t}$, and it is given by $S(o_t)$. For learning the action model, the training data consists of the state estimates $w_{s,t}$, combined with the knowledge of the action selections c_t. Since the action model maps the action selections to the *rate of change* of the world state, rather than directly to the state itself, the training data does not allow for a direct polynomial regression as in the previous section. Intuitively, the task is to learn a function from the action command to the rate of change of the world state.

More precisely, the agent's goal is to learn the function $A(c_t) = \sum_{i=0}^{d} a_i c_t^i$ that causes the values of w based on A and S, $w_{a,t}$ and $w_{s,t}$, to match each other as closely as possible. That is, the agent computes the coefficients a_i that minimize the error defined by:

$$
\begin{aligned}
E &= \sum_{t=1}^{T} \left[w_{s,t} - \left(w_0 + \sum_{i=0}^{t-1} \left(\Delta t \sum_{j=0}^{d} a_j c_i^j \right) \right) \right]^2 \\
&= \sum_{t=1}^{T} \left[w_{s,t} - \left(w_0 + \sum_{j=0}^{d} a_j \left(\Delta t \sum_{i=0}^{t-1} c_i^j \right) \right) \right]^2 \qquad (3.2)
\end{aligned}
$$

where the agent knows the values o_t, $w_{s,t}$, and the values of c_t. This problem is an instance of a multivariate linear regression, with $d+2$ coefficients being learned, w_0 and a_0 through a_d, where the input data is 1 for w_0 and $\Delta t \sum_{i=0}^{t-1} c_i^j$ for a_j and the output data is $w_{s,t}$. This regression has the effect of identifying the curve that fits the data $(t, w_{s,t})$ as closely as possible, provided that the slope of the line at any time t is a constant d-degree function of c_t. Figure 3.3 shows what this curve might look like in the case where the executed actions are piecewise constant over time.

3.2.3 Learning Both Models Simultaneously

The previous sections presented algorithms that enable the agent to learn the sensor model from the action model and vice versa. Making use of both of these capabilities, this section shows how the agent can simultaneously learn both models, even when it is given very little useful starting information. This learning is possible because, even though an action model learned from an inaccurate sensor model (and vice versa) will be inaccurate, it will often be an improvement. As each model grows more accurate, its ability to help the other model improve grows. As this bootstrapping process continues, the two models converge to functions that accurately reflect what they are trying to model.

Because both models grow in accuracy as time goes on, the regressions should give more weight to the more recent data points. Thus a weighted regression is used, where each data point has a weight that decreases over time. Note that for both

learning directions, there is one regression data point for each observation. We define the weight of each data point to start at one and decrease by a constant factor $\gamma < 1$ every time a new observation is taken. Thus if there have been n observations so far, the weight of the data points corresponding to the ith one is γ^{n-i}.

To compute the solution to the weighted regression, the diagonal $n \times n$ weight matrix D is defined by $D_{i,i} = \gamma^{n-i}$. The coefficients are then given by a weighted version of Equation 2.11 [98]:

$$\beta = (X^\top DX)^{-1}(X^\top Dy) \tag{3.3}$$

where X and y are the input matrix and output vector defined in Section 2.3.

As discussed in Section 3.2.1 for unweighted regression, these quantities can be represented in terms of sums that are maintained incrementally. For example, $(X^\top DX)_{i,j} = \sum_k \gamma^{n-k} X_{k,i} X_{k,j}$. For each new data point, such a sum is updated by taking advantage of the fact that $\sum_{k=1}^{n+1} \gamma^{(n+1)-k} z_k = \gamma(\sum_{k=1}^{n} \gamma^{n-k} z_k) + z_{n+1}$ for any sequence z_k.

At time t, the agent makes use of its best estimates thus far of the action and sensor models, denoted as A_t and S_t. These model estimates are continually updated in accordance with world state estimates $w_{s,t}$ and $w_{a,t}$, with each model being updated by the location estimate based on the other model. These incremental updates comprise the weighted polynomial regressions that give the best fit estimates of S and A, as described above. In turn, these world state estimates are updated based on the current models. Specifically, after observation o_t, $w_{s,t}$ is given by $S_t(o_t)$. At the same time, $w_{a,t}$ is maintained by continually incrementing it by $A_t(c_t)\Delta t$, $A_t(c_t)$ being the current estimate of the state velocity.

The update of $w_{a,t}$ ensures that its rate of change is the agent's best estimation of the state velocity. However, by itself this constraint allows for the possibility that there is a large, persistent difference between the estimates $w_{a,t}$ and $w_{s,t}$. It is

necessary to avoid such a difference because it would have the following adverse effect. Because the sensor model S is trained based on the values of $w_{a,t}$, the estimates $w_{s,t}$ would gradually drift toward these $w_{a,t}$ values. Then, when the action model A was learned from the estimates $w_{s,t}$, the drift would be interpreted as state velocity, which would in turn cause the values of $w_{a,t}$ to drift in the same direction. This would cause the difference between the estimates to persist while both drifted in the same direction continually. With both state estimates drifting, A_t and S_t would be unable to converge on accurate estimates of the action and sensor models.[1]

To avoid this problem, a mechanism is included that constrains $w_{a,t}$ and $w_{s,t}$ to come into closer agreement with each other over the course of the learning. The mechanism adjusts $w_{a,t}$ toward $w_{s,t}$ every time an observation is taken. The adjustment is implemented by the assignment $w_{a,t} \leftarrow (1-\alpha)w_{a,t} + \alpha w_{s,t}$, where α is a constant that determines the strength with which $w_{a,t}$ is pulled toward $w_{s,t}$. Figure 3.4a) depicts the overall flow of information.

At the start of the training, there is no data to ground either the action model or the sensor model, raising the challenge of how to get the learning process started. If regression is performed on too little data, the resulting models may have extreme inaccuracies that lead the process to diverge. To address this challenge, for a period of time at the beginning, the agent uses a fixed, pre-set action model, A_0, instead of A_t. The function used for A_0 can be a very rough approximation of the true action model, discussed in Section 3.3.3. During this time, the sensor model is learned based on A_0, but the action model is not learned yet, because the sensor model is based on too few data points. The amount of time taken to initialize a model is denoted as t_{start}. After this time has passed, the sensor model can be used to start learning an action model. However, until another period of time has passed, this new action model is not based on enough data points for it to be used

[1] In preliminary experiments, such a divergence was indeed observed.

for learning. We set this second period of time to be the same length as the first for the sake of parsimony. After these two periods of length t_{start}, the action and sensor models can learn from each other. This process is depicted in Figure 3.4b).

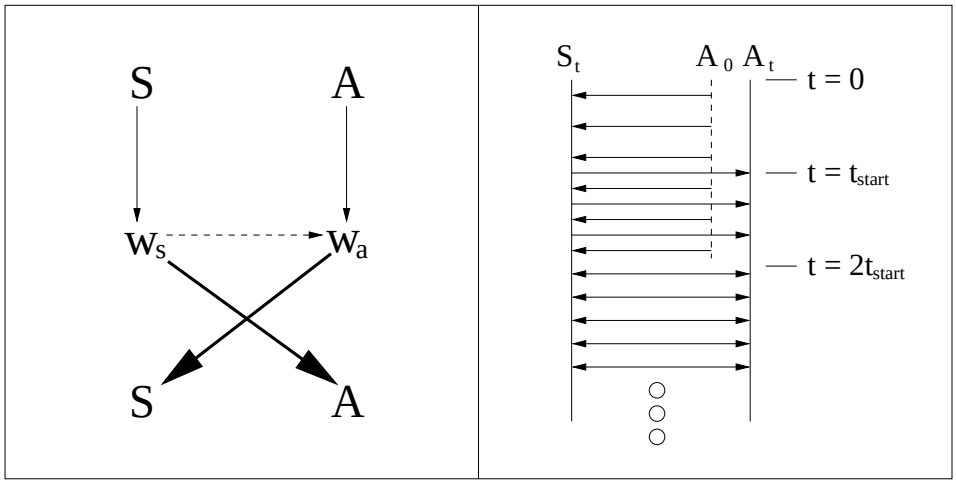


Figure 3.4: a) The flow of information. The thick arrows represent incorporating a data point into the weighted regression for a model. The thin arrows indicate that each model is used to construct the corresponding estimate of the world state. The dashed arrow signifies the influence of w_s on the estimate w_a as described above. b) The ramping up process. The arrows indicate one model being learned based on another. Note that aside from A_0, a model is not learned from until it has been learned for a sufficient amount of time.

Pseudocode for the entire process is given in Algorithm 1. The routine UPDATE incorporates one new data point into the weighted regression for the model being updated. The goal of the algorithm is for the models S and A to grow in accuracy over time, so that their estimates of the world state and velocity eventually come to closely match the true state of the world and its rate of change. The following section presents empirical evidence that these goals are met in a test domain.

Algorithm 1 Autonomous Sensor and Action Model Learning.

$w_{a,t} \leftarrow 0$
for each time step **do**
  if $t < 2t_{start}$ **then**
    $w_{a,t} \leftarrow w_{a,t} + A_0(c_t)\Delta t$
  else
    $w_{a,t} \leftarrow w_{a,t} + A_t(c_t)\Delta t$
  end if
  if an observation o_t is made **then**
    if $t > t_{start}$ **then**
      $w_{s,t} \leftarrow S_t(o_t)$
      UPDATE A_t based on $w_{s,t}$
      $w_{a,t} \leftarrow (1-\alpha)w_{a,t} + \alpha w_{s,t}$
    end if
    UPDATE S_t based $w_{a,t}$
  end if
end for

3.3 Empirical Validation

This section presents an empirical validation of the algorithm presented in this chapter on the Sony Aibo ERS-7 robot described in Section 2.5.1. The test domain for the robot is designed to meet the four characteristics described in Section 3.1. Specifically, the robot navigates through a one-dimensional state space, its sensor readings map onto possible states, its actions map onto rates of change in that same state space, and its sensations and actions are corrupted by zero-mean random noise. To realize these characteristics, the robot walks forward and backward at different speeds while facing a visible landmark. Experimental results bear out that the robot successfully learns accurate models of its sensor and actions in this domain.

As discussed in Section 3.1, the robot simultaneously performs three operations. Operations 2 and 3 consist of learning the action and sensor models. Operation 1 is for the agent to act so that it experiences the full range of action commands and observations. To achieve this goal, the Aibo walks alternatingly forward and backward across a pre-set range of distances from the beacon. As dis-

cussed in Section 2.5.1, the robot's walking action commands are parameterized by the attempted forward velocity, a_y. The value of a_y ranges from -300 to 300, specified in mm/s. The robot chooses a random action command in the range $[0, 300]$ while going forward and from $[-300, 0]$ during the backward phase. It continues to execute each action for three seconds before choosing a new one. It switches between walking forward and backward when the beacon height in the image gets too big or too small. These size thresholds are chosen manually so as to keep the robot in its field of operation. This behavior covers the full range of distances and velocities, as desired.

Although the action commands being executed only attempt to move forward and backward, random drift would cause the Aibo to slowly get off course. To counteract this effect, the walking controller is set to constantly turn toward the beacon with an angular velocity proportional to the beacon's horizontal angular distance from straight ahead. This small angular velocity has a negligible impact on the robot's forward or backward velocity.

Section 3.3.1 demonstrates the robot's ability to learn one model from the other developed in Sections 3.2.1 and 3.2.2. Section 3.3.2 shows that the robot is able to learn both models simultaneously, as described in Section 3.2.3. Finally, Section 3.3.3 presents some additional experimental results that illustrate the algorithm's robustness and flexibility.

3.3.1 Learning One Model at a Time

This section demonstrates that the robot can learn the sensor and action models, as described in Sections 3.2.1 and 3.2.2, in the test domain. The experiments in this section show qualitatively how the different components of the model learning work, in isolation, on the robotic platform. The following section demonstrates the robot's ability to learn both models simultaneously and provides quantitative experimental

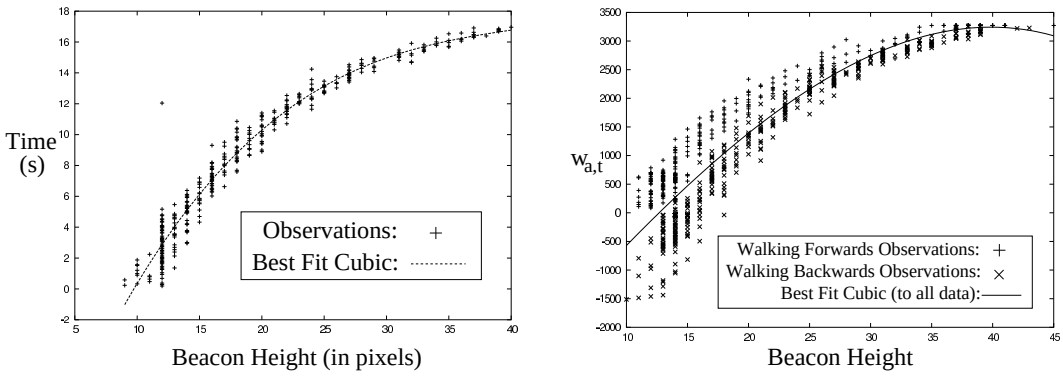
results.

The sensor and action models are learned as polynomials of degree three and four respectively, based on the estimation (without detailed experimentation) that these are roughly the polynomial degrees necessary to capture the complexity of the functions being modeled. The possibility of having the agent autonomously select the polynomial degrees is discussed in Section 3.3.3.

First we show that the robot can learn a sensor model while executing a constant action. As discussed in Section 3.2.1, this learning entails applying polynomial regression to the pairs (o_t, t) with $d = 3$ while a constant action command is being executed. When this process is performed, the cubic learned is typically quite an accurate fit to the data, as shown in Figure 3.5a).

Second, the robot must also be able to learn a sensor model while executing any control policy, such as the randomized one described above. In this case, we assume that the robot already has access to an accurate action model. Then it can use the action model to compute its velocity at any time, and use its velocity to accumulate an estimate of its location: $w_{a,t} = w_0 + \sum_{i=0}^{t-1} A(c_i)\Delta t$. To test this hypothesis, we provide the robot with a starting action model estimate, and train the sensor model based on the resultant world state estimates, using a third-degree polynomial regression. The result of such a regression is shown in Figure 3.5b). Note that because the action model used here is actually not perfectly accurate, the estimates taken while walking forward and backward are not well aligned with one another. Nonetheless, the learned sensor model is still a qualitatively reasonable one, in that as the beacon height increases, the rate of change of the corresponding location decreases, as would be expected.

Recall from Section 3.2.2 that the robot can also learn an action model if given an accurate sensor model. To verify this ability in our test domain, the robot first learns a rough sensor model using the constant action method described



a) b)

Figure 3.5: a) After walking forward via a constant action, these are the observed data points (+), mapped against time. The dashed curve is the best fit cubic to these points. The variation in beacon height at any given time is due to inherent noise in vision. b) The plotted points are $(o_t, w_{a,t})$ as the robot performs one full cycle of walking toward the beacon and backing away from it. The +'s are the observations while walking forward and the ×'s are while walking backward. The polynomial is fit to all the points.

above. It then uses that sensor model to produce a running estimate of the robot's location, $w_{s,t}$. As discussed in Section 3.2.2, this data can be combined with the knowledge of the executed actions to identify the action model that minimizes the disagreement between the location estimates based on the sensor and action models. Equation 3.2 is used to convert this minimization problem into a multivariate linear regression. Performing the regression yields the action model estimate. The result of this process is shown in Figure 3.6.

3.3.2 Learning Both Models Simultaneously

This section demonstrates the robot's ability to learn the sensor and action models simultaneously using the technique described in Section 3.2.3. The learned sensor and action models were evaluated by comparing them to models measured manually.

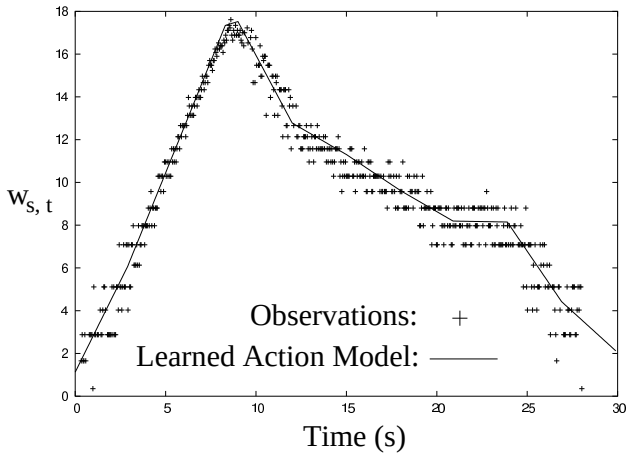


Figure 3.6: The plotted points are $(t, w_{s,t})$ as the robot performs one full cycle of walking toward the beacon and backing away from it. The learned action model is applied to the executed action commands to yield the piecewise linear location estimate shown here. Note that the units in the vertical axis of this graph are arbitrary, since it is based on the relative learned sensor model.

These comparisons found that the learned models closely matched the measured models.

Implementing Algorithm 1 on a specific platform requires finding suitable values for a few algorithmic constants. Finding these values did not require any extensive tuning. The discount factor used for the regression weights, γ, was 0.999. The strength of the pull of $w_{a,t}$ toward $w_{s,t}$, α, was 1/30. These values were the first ones that were tried for γ and α. The starting phase time, t_{start}, was 20 seconds. We tried 10 seconds first but that was too short. The initial action model, A_0, was the assumption that the attempted velocities are correct: $A_0(c) = c$. Section 3.3.3 discusses the sensitivity of the algorithm to this initial action model.

When the models S and A are learned simultaneously, Figure 3.7 depicts how $w_{s,t}$ and $w_{a,t}$ vary over time. Note that both oscillate as the robot walks toward and away from the beacon. As A and S grow more accurate, their corresponding

estimates of the location come into stronger agreement.

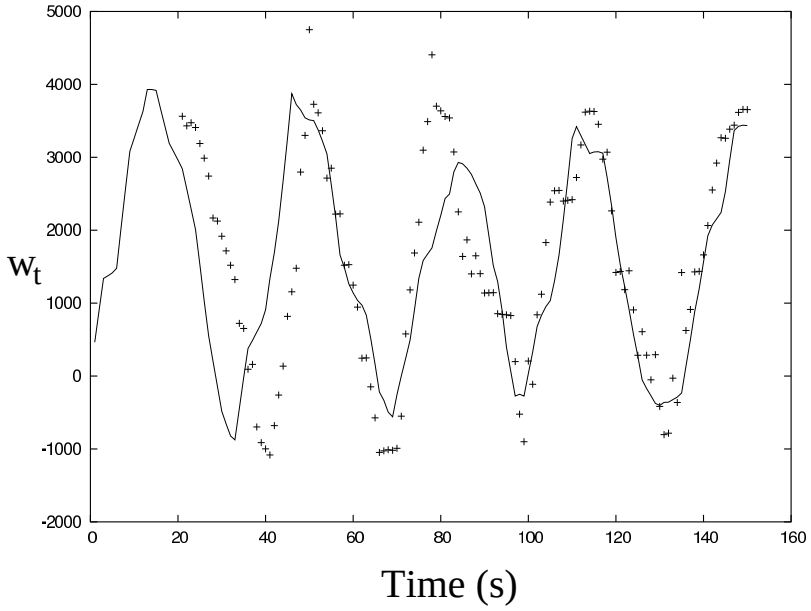


Figure 3.7: This figure shows how $w_{a,t}$, and $w_{s,t}$ vary over time. In this example run, the +'s are values of $w_{s,t}$, and the curve depicts $w_{a,t}$. Over time, each model learns how to keep its estimate of the location close to the other model's estimate.

The model learning process consists of two regression updates every time the robot processes an image. The images were processed as they were received, at a frequency of roughly 20 Hz. The model learning was performed concurrently with all of the robot's other real-time computation, including vision and motion processing, all on-board on a single 576 MHz processor.

After the learning algorithm was performed for a pre-set amount of time (two and a half minutes), its estimates of A and S at that point were considered to be the models that it had learned. In order to evaluate the learned models, the robot's actual action and sensor models were measured manually. These measurements were performed with a stopwatch and a tape measure.

The measured action model was obtained by measuring the velocity of each action command that is a multiple of 20 from −300 to 300. These velocities were measured by timing the robot walking across a distance of 4.2 meters five times.[2] The standard deviation of the velocity measurement for a given action command across the five timings never exceeded 7 mm/s. The measured action model is shown in Figure 3.8a).

Similarly, the accuracy of the learned sensor model was gauged by comparing it to a measured sensor model. The sensor model was measured by having the Aibo stand at measured distances from the beacon. The distances used were the multiples of 20 cm from 120 cm to 360 cm. At each distance, the robot looked at the beacon until it had collected 100 beacon height measurements. The average of these measurements was used as a data point for the sensor model, and their standard deviation did not exceed 1.1 pixels at any distance. The measured sensor model is shown in Figure 3.8b).

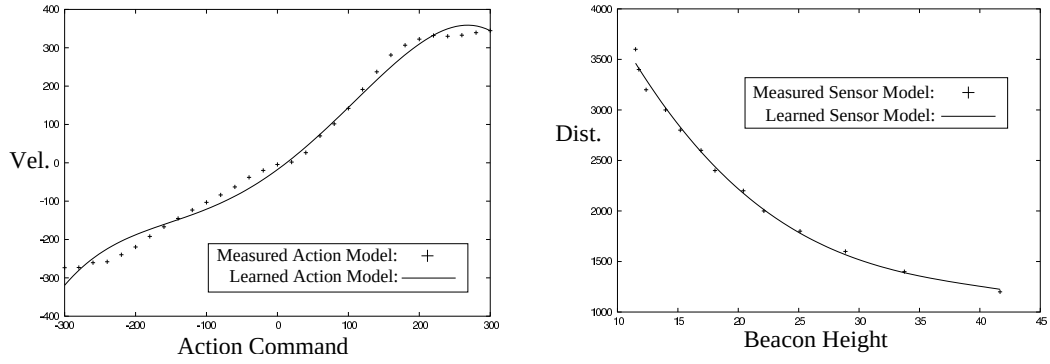


Figure 3.8: A learned action and sensor model

The learning process was executed 15 times, with each trial lasting for two and a half minutes. Figure 3.8a) shows a typical learned action model, compared to

[2]For a few very slow action commands, shorter distances were used.

the measured action model data. Note that since the action model is not learned in any specific units, in order to compare the learned model to the measured one, we must first determine the appropriate (linear) y-axis scaling factor. This evaluation is done by calculating the scaling factor that minimizes the mean squared error. On average, the root mean square error between the scaled learned action model and the measured action model was 29.6 ± 12.4 mm/s. Compared to the velocity range of 600 mm/s, this average error is 4.9 percent. The best fit possible by a fourth degree polynomial to the measured action model has an error of 17.2 mm/s. By contrast, when the the initial action model, A_0, is evaluated in the same manner, the error is 43.0 mm/s.

Figure 3.8b) shows a typical learned sensor model with the measured sensor model. The learned model S maps observations to relative distances, $S(o)$, which are intended to model the actual distances from the beacon. These actual distances are given by $a + bS(o)$, where a and b are two constants that are not learned. Thus in order to evaluate a learned sensor model, we compute the values of a and b that minimize the mean squared error between $a + bS(o)$ and the measured sensor model. This minimization is done with a linear regression on the points $(S(o_i), S_m(o_i))$, where the o_i are the sensor readings corresponding to the measured distances $S_m(o_i)$. Our evaluation of a learned sensor model is the root mean square error between it and the measured model, once this process has been applied. This value was, on average, 70.4 ± 13.9 mm. Compared to the distance range of 2400 mm, this average error is 2.9 percent. The best fit possible by a cubic to the measured sensor model has an error 48.8 mm.

Over the course of a trial, both models get progressively more accurate. The learning curves are depicted in Figure 3.9. Both models' errors are shown, compared to the best possible error for the measured model and the degree of the polynomial being learned. The data is averaged over all 15 trials.

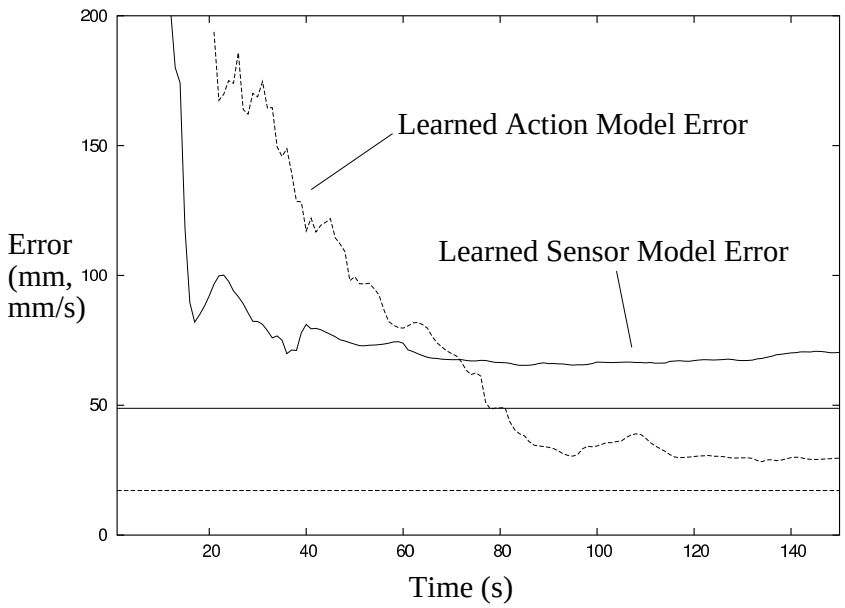


Figure 3.9: This figure depicts the average error in the learned models as a function of time. The error for the action model is in mm/s, and for the sensor model in mm. The horizontal lines are at the minimum possible error to the measured models for a polynomial of the appropriate degree.

Although the action and sensor models were not learned to any particular scale, since they were learned from each other they should be to the same scale. This property was tested by comparing the scaling constants used to give the best fits to the measured models, the scaling constant for the action model and b for the sensor model. These two values should be equal to each other in absolute value. The degree of equality was evaluated by computing the average distance between the absolute value of the ratio between the two scaling constants and 1. The average distance was 0.08 ± 0.06. This result shows that the two learned models were consistent with each other.

3.3.3 Additional Results

This section examines the impacts of two assumptions used to this point: that there is a reasonable initial action model, and that the degrees of the polynomials used for regression are chosen manually. For the initial action model, I show that it can convey very little information and still be sufficient to get the learning started. In fact, even when the initial action model function is a constant, conveying no information about c_t, the robot can frequently learn accurate action and sensor models. Additionally, I discuss the challenge of choosing the polynomial degree autonomously and present some preliminary results in this regard.

Initial Action Model

Recall from Section 3.2.3 that an initial action model A_0 is used for an amount of time t_{start} to seed the learning. This approximate model is the only information that the robot starts with about either the action or sensor model. The results described in Section 3.3.2 use a linear initial action model, $A(c) = c$, that is somewhat similar to the measured action model (shown in Figure 3.8). To examine the reliance of the learning algorithm on this initial action model, we performed two tests with more impoverished starting points. First, we used a piecewise constant model equal to Sgn(c): 1 for positive action commands and -1 for negative ones. This model conveys only the direction of the action but no information about its speed. In 15 runs, the robot achieved an average error of 85.3 ± 24.5 mm in its learned sensor model and 31.3 ± 9.2 mm/s in the action model after two and a half minutes. These errors are comparable to those attained with the linear model (70.4 ± 13.9 mm and 29.6 ± 12.4 mm/s for the sensor and action models respectively).

Even with a starting model of $A(c) = 1$, which imparts no information about the action model, on 10 out of 15 trials the robot was able to achieve an average performance of 88.6 ± 11.5 mm error in the sensor model and 27.3 ± 6.2 mm/s in

the action model after five minutes. The remaining trials diverged, presumably due to initially learning a pair of models that were so inaccurate that no useful information could be recovered from them. The results from the three different starting conditions are presented in Table 3.1. Note that the errors achieved with these more impoverished models are similar to those achieved with the linear model, indicating that our results are not particularly sensitive to the choice of the starting action model.

$\mathbf{A_0}$	**Sensor Model (mm)**	**Action Model (mm/s)**	**Success Rate**
$A_0(c) = c$	70.4 ± 13.9	29.6 ± 12.4	15/15
$A_0(c) = \mathrm{Sgn}(c)$	85.3 ± 24.5	31.3 ± 9.2	15/15
$A_0(c) = 1$	88.6 ± 11.5	27.3 ± 6.2	10/15

Table 3.1: For each of three initial action models tried, this table shows the average fidelity of the learned sensor and action models. The last column shows how many of the 15 trials resulted in the sensor and action models converging. The top row replicates the results presented in Section 3.3.2.

Polynomial Degree Selection

A potential enhancement to the model learning algorithm presented in this chapter would be to enable it to choose the degrees for the polynomial regressions automatically. The remainder of this section presents some results toward that goal based on a method that explicitly distinguishes random noise from model error [92].

Choosing the degree of the polynomial is a type of model selection, the problem of identifying the best parameters for a function approximator. There are many popular model selection techniques, such as the Akaike Information Criterion [2], the Bayes Information Criterion [77], and cross-validation [38]. For choosing the degree of a polynomial for regression in the context of robotic model learning, we tried a method that takes advantage of the increasing expressive power of the successive possible polynomial degrees and the wealth of data typically available in a robotic

setting.

To choose the degree, we start by fitting a first degree (linear) polynomial and continually monitoring the fit to see if the degree needs to be increased. If so, the regression is restarted with the degree incremented by one. The degree continues to be incremented until a satisfactory fit is found. In order to determine whether or not a particular degree is satisfactory, the robot compares a *global prediction error* and a *local noise estimate*, two values that are continually maintained. A high global prediction error indicates a poor fit, suggesting that the polynomial degree should perhaps be increased. However, such an error might also be accounted for by a large amount of random noise in the observed data, in which case increasing the degree will not help. This comparison finds the lowest degree polynomial that achieves a satisfactory fit, which has the effect of implicitly balancing the higher computational costs of higher degrees against their improved accuracy.

Because of the added complications in learning an action model (see Section 3.2.2), we tested this method on only learning a sensor model. While the robot executed the same randomized behavior described in Section 3.3.2, it obtained training data for the sensor model by using a fixed action model that is the best fit fourth degree polynomial to the measured data. The estimates of the robot's location based on this action model were used as training data for the sensor model learning with automated degree finding.

Fifteen trials were run, each lasting five minutes. In each trial, the degree stabilized within two and a half minutes and did not increase after that time. The fact that the robot was able to settle on a degree every time demonstrates the method's stability. Nevertheless, randomness in the training data caused some variation in the final polynomial degree. The average degree chosen was 3.33, with a standard deviation of 1.29. This degree corroborates our earlier estimate that a third degree polynomial was roughly the amount of complexity needed to learn the sensor model.

The learned sensor models were evaluated as described in Section 3.3.2. The average errors for the learned sensor models was 101 ± 34 mm. Compared to the distance range of 2400 mm, the average error is 4.2 percent. For comparison, with the manually chosen fixed degree, the average sensor model in Section 3.3.2 was 70.4 ± 13.9 mm. This experiment demonstrates the potential feasibility of enabling the robot to autonomously choose the degrees for its polynomial regressions during the model learning process.

This chapter demonstrated the possibility of an autonomous agent learning its action and sensor models starting without an accurate estimate of either model. The algorithm presented achieves this goal in a class of one-dimensional settings that includes a robot walking forward and backward while facing a fixed landmark. In the following chapter, the challenges that arise from extending this framework to a robot walking in a two-dimensional domain are addressed.

Chapter 4

Model Learning in Two Dimensions

In this chapter, we consider the scenario of an autonomous legged robot walking on a two-dimensional surface with observations that correspond to the robot's distance and angle to landmarks. The robot used in the experiments, the Aibo ERS-7, and the robot model are described in Section 2.5.1. Section 4.1 summarizes this model and describes the maximum likelihood framework used to learn the action and sensor models, namely the Expectation-Maximization (EM) algorithm [26]. The adaptation of the E-step, described in Section 4.2, is partly achieved by an extended Kalman filter and smoother (EKFS) [31]. The remainder of the E-step and the adaptation of the M-step to this domain, presented in Section 4.3, are primary contributions of this work [91]. Finally, Section 4.4 presents experimental results.

4.1 Setup

Recall from Section 2.5.1 that the robot's motions lead to combinations of forward, rightward, and turning velocities. Its observations consist of the vertical size of an

observed landmark in the robot's camera image and the horizontal angle at which the landmark is observed. These actions and observations are depicted in Figure 4.1.

Figure 4.1: a) The robot moves in arbitrary combinations of forward, rightward, and turning velocities. b) The robot's observation components are a function of its distance to the landmark and the landmark's horizontal angle from the robot.

The components of the sensor model that are learned in this chapter are the function f from landmark distance to observation height (see Figure 2.2) and the variances of the random noise in the observation components, σ_1^2, and σ_2^2. The function f is approximated as a third-degree polynomial. As discussed in Chapter 3, this degree was chosen manually based on the estimated complexity of the function being modeled. The component of the action model that is learned is the action model function A from action commands to the corresponding velocity combinations. This function is learned as a table of function values, namely the velocities that correspond to a suite of 40 actions that are designed to cover the full range of combinations of forward, rightward, and turning velocities, described in Section 4.4. Previous work has shown that these velocities can be interpolated for intermediate actions [28, 68].

As in the previous chapter, the action and sensor models are learned starting without an accurate estimate of either model. Specifically, here the starting sen-

sor model consisted of an inaccurate linear polynomial, combined with inaccurate estimates of σ_1 and σ_2. These values are presented along with the experimental results in Section 4.4.1. Additionally, the starting action model corresponded to the assumption that each action has no effect.

The action and sensor models, referred to collectively as λ,[1] are learned by identifying those models that have the maximum likelihood, $P(O|C,\lambda)$, where O and C represent the set of all of the robot's observations and action commands respectively. This likelihood is marginalized over the possibilities for W, the series of world states. The EM algorithm is a method for identifying a maximum likelihood parameter set given a set of observations in the context of *hidden data*. In this work, the hidden data is the world state (pose) of the robot over time, $w_t = (x_t, y_t, \theta_t)$. The learning process therefore operates in batch on a series of actions and observations recorded by the robot over a period of time.

The EM algorithm starts with an initial guess for the parameters, λ_0, and repeatedly revises it by a sequence of two steps, expectation (E) and maximization (M). The E-step assumes the current parameter estimate is correct and identifies the probability distribution over the hidden variables, denoted as $\hat{P}$, given all of the actions and observations, $\hat{P}(W) = P(W|O,C,\lambda_i)$.

The M-step uses the distribution $\hat{P}$ computed in the E-step to compute a new set of parameter estimates, λ_{i+1}. The new parameters are defined to be those that maximize the expected log likelihood of λ leading to the observed and hidden variables, where the expectation is taken with respect to $\hat{P}(W)$:

$$\lambda_{i+1} = \text{argmax}_{\lambda'} \left(\text{E}_{\hat{P}}[\log P(O,W|C,\lambda')] \right) \tag{4.1}$$

$$= \text{argmax}_{\lambda'} \left(\int \hat{P}(W) \log P(O,W|C,\lambda')\, dW \right) \tag{4.2}$$

[1] A prior distribution over the initial state, denoted as $\pi(w_0)$, is also included in λ.

where the integral is taken over the space of all possible sequences of world states W. Alternating between the E-step and M-step causes the parameter estimate to converge to a local maximum likelihood. However, under the approximations made in the following two sections, convergence is not guaranteed.

The following section explains the adaptation of the E-step to the problem of learning the robot's action and sensor models. The adaptation of the M-step is presented in Section 4.3.

4.2 Adapting the E-step

As described above, the E-step aims to find the posterior distribution over the sequence of world states given the sequence of actions and observations and the previous parameter estimate, $\hat{P}(W) = P(W|O, C, \lambda_i)$. The M-step described in the following section relies specifically on knowledge of $\hat{P}$ viewed as a distribution over the world state at one time step, $P(w_t|O, C, \lambda_i)$, or over two consecutive time steps (in Section 4.3.1), $P(w_t, w_{t+1}|O, C, \lambda_i)$. These distributions are marginalized over the world state at all of the time steps except for the one or two in question. This section describes how they are estimated.

First, consider $P(w_t|O, C, \lambda_i)$, which we denote as $\gamma_t(w_t)$. Section 2.4 described how an extended Kalman filter can be used to estimate a distribution over w_t given all of the observations and actions up until that point. However, we are interested in the distribution over w_t that also incorporates the knowledge of the *future* observations and actions, o_{t+1} through o_T and c_t through c_{T-1}. This information can be incorporated through the EKFS, which can be implemented as a forward-backward smoother [31].

The forward-backward smoother can be thought of as an instance of the forward-backward algorithm for hidden Markov models [69]. That is, it can be understood in terms of the functions α_t and β_t, defined as follows:

$$\alpha_t(w_t) = P(o_1 \dots o_t, w_t | c_0 \dots c_{t-1}, \lambda_i) \tag{4.3}$$

$$\beta_t(w_t) = P(o_{t+1} \dots o_T | w_t, c_t \dots c_{T-1}, \lambda_i) \tag{4.4}$$

To compute γ_t, we apply Bayes' theorem to w_t and $o_{t+1} \dots o_T$:

$$P(w_t | O, C, \lambda_i) = \frac{P(w_t | o_1 \dots o_t, C, \lambda_i) P(o_{t+1} \dots o_T | w_t, o_1 \dots o_t, C, \lambda_i)}{P(o_{t+1} \dots o_T | o_1 \dots o_t, C, \lambda_i)} \tag{4.5}$$

Removing dependencies that have no effect, we get:

$$P(w_t | O, C, \lambda_i) = \frac{P(w_t | o_1 \dots o_t, c_0 \dots c_{t-1}, \lambda_i) P(o_{t+1} \dots o_T | w_t, c_t \dots c_{T-1}, \lambda_i)}{P(o_{t+1} \dots o_T | o_1 \dots o_t, C, \lambda_i)} \tag{4.6}$$

Multiplying the numerator and denominator by $P(o_1 \dots o_t | C, \lambda_i)$ yields:

$$\begin{aligned} P(w_t | O, C, \lambda_i) &= \frac{P(o_1 \dots o_t, w_t | c_0 \dots c_{t-1}, \lambda_i) P(o_{t+1} \dots o_T | w_t, c_t \dots c_{T-1}, \lambda_i)}{P(O | C, \lambda_i)} \\ \gamma_t(w_t) &= \frac{\alpha_t(w_t) \beta_t(w_t)}{P(O | C, \lambda_i)} \end{aligned} \tag{4.7}$$

Note that $\alpha_t(w_t)$ is equal to the constant $P(o_1 \dots o_t | c_0 \dots c_{t-1}, \lambda_i)$ multiplied by $P(w_t | o_1 \dots o_t, c_0 \dots c_{t-1}, \lambda_i)$, the density function computed by the EKF in Section 2.4. The distribution's mean and covariance, μ'_t and Σ'_t are denoted here as $\mu_{\alpha,t}$ and $\Sigma_{\alpha,t}$.

On the other hand, the function $\beta_t(w_t)$ is a likelihood function of w_t, and it is not guaranteed to be proportional to a probability distribution. That is, the integral $\int \beta_t(w_t)\, dw$ is not guaranteed to converge for all t (e.g., $\beta_T(w_T) \equiv 1$). However, for

most values of t, the integral does converge and β_t can be considered as a distribution by dividing it by this constant. The resulting distribution's mean and covariance matrix are denoted as $\mu_{\beta,t}$ and $\Sigma_{\beta,t}$. They can be estimated by running a separate EKF backward in time, starting from a mean and covariance matrix of 0 and ∞.[2]

The means and covariances of α and β are combined to find those of γ by applying the definition (2.6) to Equation 4.7:

$$\begin{aligned}\gamma_t(w_t) &\propto \exp\left(-\frac{1}{2}((w_t-\mu_{\alpha,t})^\top\Sigma_{\alpha,t}^{-1}(w_t-\mu_{\alpha,t})+(w_t-\mu_{\beta,t})^\top\Sigma_{\beta,t}^{-1}(w_t-\mu_{\beta,t}))\right)\\ &\propto \exp\left(-\frac{1}{2}((w_t-\mu_{\gamma,t})^\top\Sigma_{\gamma,t}^{-1}(w_t-\mu_{\gamma,t}))\right) \qquad (4.8)\end{aligned}$$

where $\mu_{\gamma,t}$ and $\Sigma_{\gamma,t}$ are defined by:

$$\mu_{\gamma,t} = (\Sigma_{\alpha,t}^{-1}+\Sigma_{\beta,t}^{-1})^{-1}(\Sigma_{\alpha,t}^{-1}\mu_{\alpha,t}+\Sigma_{\beta,t}^{-1}\mu_{\beta,t}) \qquad (4.9)$$

$$\Sigma_{\gamma,t} = (\Sigma_{\alpha,t}^{-1}+\Sigma_{\beta,t}^{-1})^{-1} \qquad (4.10)$$

Expanding Expression 4.8 accordingly yields the exponent in the preceding expression plus a constant. Hence Equations 4.9 and 4.10 determine the mean and covariance of $\gamma_t(w_t) = P(w_t|O,C,\lambda_i)$, as desired.

Next, we wish to find the mean and covariance of the joint distribution over w_t and w_{t+1}, which we refer to collectively as $w_{t,t+1}$, given O, C, and λ_i. We denote this distribution, $P(w_{t,t+1}|O,C,\lambda_i)$, as $\xi_t(w_{t,t+1})$. It can be factored as [69]:

$$\xi_t(w_{t,t+1}) = \frac{\alpha_t(w_t)\beta_{t+1}(w_{t+1})P(o_{t+1}|w_{t+1},\lambda_i)P(w_{t+1}|w_t,c_t,\lambda_i)}{P(O|C,\lambda_i)} \qquad (4.11)$$

[2]In practice, a very large diagonal matrix can be used for $\Sigma_{\beta,T}$.

Section 4.3.1 relies on knowledge of the mean and covariance of the distribution ξ_t. To compute these quantities, first note that multiplying $\beta_{t+1}(w_{t+1})$ by $P(o_{t+1}|w_{t+1}, \lambda_i)$ is, up to a constant, the operation that is considered by the Kalman filter measurement update (2.17)-(2.19). The mean and covariance of this quantity (when normalized) are therefore computed in the backward sweep of the forward-backward smoothing; we denote them as $\mu_{\delta,t+1}$ and $\Sigma_{\delta,t+1}$. Next we combine these values with our knowledge of α_t, specifically $\mu_{\alpha,t}$ and $\Sigma_{\alpha,t}$, which are generated in the forward EKF sweep. The mean and covariance of the product $\alpha_t(w_t)\beta_{t+1}(w_{t+1})P(o_{t+1}|w_{t+1}, \lambda_i)$ (when normalized) are denoted as $\mu_{\zeta,t}$ and $\Sigma_{\zeta,t}$ and given by:

$$\mu_{\zeta,t} = \begin{pmatrix} \mu_{\alpha,t} \\ \mu_{\delta,t+1} \end{pmatrix} \quad \text{and} \quad \Sigma_{\zeta,t} = \begin{pmatrix} \Sigma_{\alpha,t} & 0 \\ 0 & \Sigma_{\delta,t+1} \end{pmatrix} \tag{4.12}$$

The final factor of Equation 4.11 that must be incorporated is the action likelihood $P(w_{t+1}|w_t, c_t, \lambda_i)$. We accordingly define $D(w_{t,t+1})$ to be the relative displacement between the two states, specifically $R(-\theta_t)(w_{t+1} - w_t)$.[3] Note from Equation 2.28 that $P(w_{t+1}|w_t, c_t, \lambda_i)$ has the value of a normal distribution with mean $A(c_t)\Delta t$, which we denote as μ_{c_t}, and covariance Σ_m, evaluated at $D(w_{t,t+1})$, where A is the action model function according to λ_i. This multiplication by the PDF of a normal distribution over a function of the state is again the operation performed by the extended Kalman filter measurement update: the state, observation, and function h in Equation 2.23 correspond to $w_{t,t+1}$, μ_{c_t}, and D, and the role of R in Equation 2.16 is played by Σ_m, the motion uncertainty. Applying the EKF measurement update as described in Section 2.4, linearizing around $\mu_{\zeta,t}$, yields our estimate of the mean and covariance of ξ_t, denoted as $\mu_{\xi,t}$ and $\Sigma_{\xi,t}$. These quantities, along with $\mu_{\gamma,t}$ and $\Sigma_{\gamma,t}$ derived above, are the properties of the distribution

[3]Whenever angles are subtracted, such as in the third component of w here, the result is normalized to be within the interval $[-\pi, \pi)$ throughout this dissertation.

$\hat{P}(W) = P(W|O, C, \lambda_i)$ that are used in the following section to determine the next iteration of action and sensor models, λ_{i+1}.

Finally, the experimental results reported in Section 4.4 incorporate knowledge of the overall likelihood of the current parameter set, $P(O|C, \lambda_i)$. During the E-step, this likelihood can be determined by factoring it as:

$$P(O|C, \lambda_i) = \prod_{t=1}^{T} P(o_t|o_1 \ldots o_{t-1}, c_0 \ldots c_{t-1}, \lambda_i) \quad (4.13)$$

$$= \prod_{t=1}^{T} \int P(o_t|w_t, \lambda_i) P(w_t|o_1 \ldots o_{t-1}, c_0 \ldots c_{t-1}, \lambda_i) dw_t \quad (4.14)$$

The second factor in this integral is the state distribution computed in the forward sweep of the EKF at time t, with mean and covariance $\mu_{\alpha,t}$ and $\Sigma_{\alpha,t}$. Linearizing the sensation function as described in Equation 2.27 yields the approximation $o_t = Hw_t + r_t$ where $r_t \sim \mathcal{N}(0, R)$ (Equations 2.14 and 2.16). Hence the probability distribution over o_t in Equation 4.13 is given by $o_t \sim \mathcal{N}(H\mu_{\alpha,t}, H\Sigma_{\alpha,t}H^\top + R)$ from Equations 2.7 and 2.8. The likelihood density of the observed values of o_t at each time step according to this distribution are multiplied into a running overall parameter likelihood.[4]

4.3 Adapting the M-step

As discussed in Section 4.1, the M-step of the EM algorithm is to determine the next parameter set λ_{i+1}, defined by Equation 4.1 to be the value of λ' that maximizes $\mathrm{E}_{\hat{P}}[\log P(O, W|C, \lambda')]$, where $\hat{P}(W)$ is the distribution based on λ_i that was analyzed in the previous section. This quantity can be decomposed as follows:

[4] To avoid overflow or underflow, the log likelihoods are added to a cumulative overall log likelihood.

$$
\begin{aligned}
& \mathrm{E}_{\hat{P}}[\log P(O, W|C, \lambda')] \\
= & \int \hat{P}(W) \log P(O, W|C, \lambda')\, dW \\
= & \int \hat{P}(W) \log \left[\pi'(w_0) \prod_{t=1}^{T} P(w_t|w_{t-1}, c_{t-1}, \lambda') P(o_t|w_t, \lambda') \right] dW \\
= & \int \hat{P}(W) \left[\log \pi'(w_0) + \sum_{t=1}^{T} \left(\log P(w_t|w_{t-1}, c_{t-1}, \lambda') + \log P(o_t|w_t, \lambda') \right) \right] dW \\
= & \int \hat{P}(W) \log \pi'(w_0)\, dW + \sum_{t=1}^{T} \int \hat{P}(W) \log P(w_t|w_{t-1}, c_{t-1}, \lambda')\, dW \\
& + \sum_{t=1}^{T} \int \hat{P}(W) \log P(o_t|w_t, \lambda')\, dW \qquad (4.15)
\end{aligned}
$$

where π' is the prior distribution over w_0 according to λ'.

This expression decomposes the expected log likelihood into a sum of three terms that are functions of the three components of λ': π', the action model, $P(w_t|w_{t-1}, c_{t-1}, \lambda')$, and the sensor model, $P(o_t|w_t, \lambda')$. Maximizing this expression with respect to λ' consists of maximizing each term with respect to the corresponding component. The distribution π' that maximizes the first term is equal to the distribution of w_0 according to $\hat{P}$, namely $\gamma_0(w_0)$ [69]. Maximizing the other two terms corresponds to learning the action and sensor model, discussed in the next two sections.

4.3.1 Learning the Action Model

The revised action model in each iteration of the EM algorithm is the one that maximizes the second term in (4.15). For each action a, let $Q(a) = \{t : c_t = a\}$, the set of time steps at which action a occurred. Then, the portion of the second term in (4.15) affected by action a is given by:

$$
\begin{aligned}
& \sum_{t \in Q(a)} \int \hat{P}(W) \log P(w_{t+1}|w_t, c_t, \lambda') \, dW \\
= & \sum_{t \in Q(a)} \int \hat{P}(w_{t,t+1}) \log P(w_{t+1}|w_t, a) \, dw_{t,t+1} \\
= & \sum_{t \in Q(a)} \int \xi_t(w_{t,t+1}) \left[C - \frac{1}{2} (D(w_{t,t+1}) - \mu_a)^\top \Sigma_m^{-1} (D(w_{t,t+1}) - \mu_a) \right] dw_{t,t+1}
\end{aligned}
$$

where D and ξ are defined in Section 4.2 as the relative displacement function and the state distribution $\hat{P}$ over two consecutive time steps, respectively, and C is a constant with respect to μ_a. The last step follows from Equations 2.28 and 2.6.

Completing the M-step requires finding the displacement resulting from action a, μ_a, that maximizes the above expression. Equivalently, the new value of μ_a is the one that minimizes:

$$
\sum_{t \in Q(a)} \int \xi_t(w_{t,t+1}) (D(w_{t,t+1}) - \mu_a)^\top \Sigma_m^{-1} (D(w_{t,t+1}) - \mu_a) \, dw_{t,t+1} \tag{4.16}
$$

Taking the gradient with respect to μ_a and setting it equal to zero, we get:

$$
\begin{aligned}
\sum_{t \in Q(a)} \int \xi_t(w_{t,t+1}) (2\Sigma_m^{-1} \mu_a - 2\Sigma_m^{-1} D(w_{t,t+1})) \, dw_{t,t+1} &= 0 \\
2\Sigma_m^{-1} \left(\sum_{t \in Q(a)} \int \xi_t(w_{t,t+1}) \mu_a \, dw_{t,t+1} - \sum_{t \in Q(a)} \int \xi_t(w_{t,t+1}) D(w_{t,t+1}) \, dw_{t,t+1} \right) &= 0 \\
\sum_{t \in Q(a)} \int \xi_t(w_{t,t+1}) \mu_a \, dw_{t,t+1} = \sum_{t \in Q(a)} \int \xi_t(w_{t,t+1}) D(w_{t,t+1}) \, dw_{t,t+1} &
\end{aligned}
$$

Using the fact that $\int \xi_t(w_{t,t+1}) dw_{t,t+1} = 1$, we get:

$$\sum_{t \in Q(a)} \mu_a = \sum_{t \in Q(a)} \int \xi_t(w_{t,t+1}) D(w_{t,t+1})\, dw_{t,t+1} \tag{4.17}$$

$$\mu_a = \frac{1}{|Q(A)|} \sum_{t \in Q(a)} \int \xi_t(w_{t,t+1}) D(w_{t,t+1})\, dw_{t,t+1} \tag{4.18}$$

In order to complete the computation of μ_a, we must compute $\int \xi_t(w_{t,t+1})$ $D(w_{t,t+1}) dw_{t,t+1}$ for each t in $Q(a)$ and average the results. This integral can be thought of as the expected value of $D(w_{t,t+1})$ with respect to the probability distribution ξ_t. To evaluate this expected value, $E_{\xi_t}[D(w_{t,t+1})]$, note that in general if $E[x] = \mu$, $E[Ax] = A\mu$. Therefore, approximating D as linear, we get that the expected value is $D(\mu_{\xi,t})$, where $\mu_{\xi,t}$ is the (six-dimensional) mean of the ξ_t distribution computed in Section 4.2. Averaging this quantity over the $|Q(a)|$ relevant time steps yields the new estimate for the mean displacement μ_a corresponding to action a. This mean displacement is divided by Δt to yield the corresponding learned velocity components.

4.3.2 Learning the Sensor Model

Recall from Section 2.5.1 that the robot's observations correspond to sightings of the landmarks in the environment, which are assumed to be visually distinguishable. In world state w_t, the distribution over observations is given by:

$$o_t = \begin{pmatrix} o_{t,1} \\ o_{t,2} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} f(\text{dist}(w_t)) \\ \text{ang}(w_t) \end{pmatrix}, \begin{pmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{pmatrix} \right) \tag{4.19}$$

where the functions $\text{dist}(w_t)$ and $\text{ang}(w_t)$ represent the distance and horizontal angle respectively from a robot at state w_t to the landmark that is observed. The sensor model that is learned consists of the function f plus the standard deviations σ_1 and σ_2.

To learn this sensor model, we maximize the third term in (4.15) with respect to f, σ_1, and σ_2:

$$\sum_{t=1}^{T} \int \hat{P}(W) \log P(o_t|w_t)\, dW \tag{4.20}$$

$$= \sum_{t=1}^{T} \int \hat{P}(w_t) \log P(o_t|w_t)\, dw_t \tag{4.21}$$

$$= \sum_{t=1}^{T} \int \gamma_t(w_t) \left[C - \frac{1}{2} \left(\log \sigma_1^2 \sigma_2^2 + \left(\frac{f(\text{dist}(w_t)) - o_{t,1}}{\sigma_1} \right)^2 + \left(\frac{\text{ang}(w_t) - o_{t,2}}{\sigma_2} \right)^2 \right) \right] dw_t \tag{4.22}$$

where $\gamma_t(w_t)$ is the probability of w_t according to $\hat{P}$ discussed in Section 4.2 and $\sigma_1^2\sigma_2^2$ is the determinant $|\Sigma|$ in Equation 2.6.

Maximizing the above expression is equivalent to minimizing both

$$\sum_{t=1}^{T} \int \gamma_t(w_t) \left(\log \sigma_1^2 + \left(\frac{f(\text{dist}(w_t)) - o_{t,1}}{\sigma_1} \right)^2 \right) dw_t \tag{4.23}$$

$$\text{and} \quad \sum_{t=1}^{T} \int \gamma_t(w_t) \left(\log \sigma_2^2 + \left(\frac{\text{ang}(w_t) - o_{t,2}}{\sigma_2} \right)^2 \right) dw_t \tag{4.24}$$

First, note that Expression 4.23 can be minimized with respect to f, independent of σ_1 and σ_2, by minimizing $\sum_t \int \gamma_t(w_t)(f(\text{dist}(w_t)) - o_{t,1})^2 dw_t$, which we approximate as:

$$\sum_{t=1}^{T} \sum_{i=1}^{N_s} (f(\text{dist}(w_t^{(i)})) - o_{t,1})^2 dw_t \tag{4.25}$$

where the $w_t^{(i)}$ are N_s samples of w_t that are drawn from the γ_t distribution. Samples are drawn from this distribution, $\mathcal{N}(\mu_{\gamma,t}, \Sigma_{\gamma,t})$, by taking the Cholesky decomposi-

tion $\Sigma_{\gamma,t} = LL^\top$, generating samples of $z^{(i)} \sim \mathcal{N}(0, I)$ using the Box-Muller transform, and choosing the sample $w_t^{(i)}$ to be $\mu_{\gamma,t} + Lz^{(i)}$. The resulting desired variance of $LL^\top$ follows from Equation 2.7. The function f is represented as a polynomial and thus learned by polynomial regression applied to the pairs $(\text{dist}(w_t^{(i)}), o_{t,1})$. In the experiments reported in the following section, the number of samples used from each frame was one.[5]

The resulting polynomial f minimizes Expression 4.23 for any choice of σ_1. To find the minimizing value of σ_1, we differentiate the expression with respect to σ_1 and set the result equal to 0, using the new value of f. The result is:

$$\sum_{t=1}^{T} \int \gamma_t(w_t) \left(\frac{2}{\sigma_1} - \frac{2(f(\text{dist}(w_t)) - o_{t,1})^2}{\sigma_1^3} \right) dw_t = 0$$

$$\sum_{t=1}^{T} \int \gamma_t(w_t) \frac{2}{\sigma_1} \, dw_t = \sum_{t=1}^{T} \int \gamma_t(w_t) \frac{2(f(\text{dist}(w_t)) - o_{t,1})^2}{\sigma_1^3} \, dw_t$$

Because $\int \gamma_t(w_t) dw_t = 1$:

$$\frac{2T}{\sigma_1} = \sum_{t=1}^{T} \int \gamma_t(w_t) \frac{2(f(\text{dist}(w_t)) - o_{t,1})^2}{\sigma_1^3} \, dw_t \tag{4.26}$$

$$\sigma_1 = \sqrt{\frac{\sum_{t=1}^{T} \int \gamma_t(w_t)(f(\text{dist}(w_t)) - o_{t,1})^2 \, dw_t}{T}} \tag{4.27}$$

This weighted RMS error is estimated by again using the N_s samples drawn from each γ_t distribution and taking the RMS average of the corresponding errors, namely $f(\text{dist}(w_t^{(i)})) - o_{t,1}$.

Similarly, minimizing (4.24) with respect to σ_2 yields:

[5] Because of the large number of frames in the data set, it was not necessary to draw many samples from each one.

$$\sigma_2 = \sqrt{\frac{\sum_t \int \gamma_t(w_t)(\text{ang}(w_t) - o_{t,2})^2}{T}} \tag{4.28}$$

This value is also estimated based on the samples of γ_t as the RMS average of the corresponding errors: $\text{ang}(w_t^{(i)}) - o_{t,2}$. This computation completes the reestimation of λ in the M-step.

As mentioned in Section 4.1, the EM algorithm alternates between the E-step and M-step until convergence is reached. In general, this process is guaranteed to converge to a local maximum likelihood parameter set. However, a number of approximations have been made throughout the algorithm. Primarily, the EKFS used in the E-step approximates the system dynamics and observations as being linear and as having Gaussian noise. These approximations, along with the distribution sampling in the M-step, mean that the learning process is in fact an approximation of the EM algorithm, and is therefore not guaranteed to converge. In practice, informal experiments suggested that the nonlinearity and non-Gaussianity of the observations can potentially cause persistent severe inaccuracies in the EKFS, which in turn keeps the entire EM process from converging, especially if observation outliers are not explicitly rejected, or if a sufficiently poor starting model is used. However, even starting with no action model and a very inaccurate sensor model, this approximation to EM is able to learn accurate action and sensor models for the robot, as shown in the following section.

4.4 Empirical Validation

The technique described in this chapter was validated both on a physical mobile robot and in simulation. In both cases, accurate sensor and action models were learned, starting with no action model and a very poor sensor model. On the real robot, the learned translational velocities were not evaluated due to the difficulty

in measuring the ground truth for these velocities. All of the other aspects of the action and sensor models were measured and compared to the learned models. In simulation, ground truth is known, and all components of the learned models were evaluated. Additionally, Section 4.4.3 provides two demonstrations of the versatility of the algorithm presented in this chapter. First, it shows that the technique presented in this chapter can be easily adapted to the class of one-dimensional domains considered in Chapter 3. Second, I demonstrate the possibility of relearning the action and sensor models after one of them has changed.

4.4.1 Real Robot Results

The robot used in the experiments was a Sony Aibo ERS-7, depicted in Figure 2.1. The Aibo's field of operation, shown in Figure 4.1, measures 5.4m $\times$ 3.6m. The landmarks on the field are four distinct cylindrical beacons in fixed, known locations. Recall from Section 2.5.1 that the robot's action commands correspond to *attempted* velocities, a_x, a_y and a_θ. These attempted velocities determine the robot's step sizes and directions. However, they are often significantly inaccurate because of inaccuracies in the robot's joint movement and its feet slipping against the ground. The action model learned by the robot maps these action commands onto actual velocity combinations. A set of 40 action commands was used, determined as follows.

The attempted velocity combinations, (a_x, a_y, a_θ), were all chosen to walk as fast as possible in a given direction (including turning as a component of the direction). With each component normalized to be in the range $[-1, 1]$, such combinations satisfy the equation $a_x^2 + a_y^2 + a_\theta^2 = 1$. The velocities are uniquely determined by the combination of this equation, an angular velocity (specified as a fraction of the maximum), $\omega \in \{-\frac{1}{2}, -\frac{1}{6}, 0, \frac{1}{6}, \frac{1}{2}\}$, and the direction of (a_x, a_y), $\psi \in \{0, \pm\frac{\pi}{4}, \pm\frac{\pi}{2}, \pm\frac{3\pi}{4}, \pi\}$. Specifically:

$$a_\theta = \omega \tag{4.29}$$

$$a_x = \cos(\psi)\sqrt{1 - a_\theta^2} \tag{4.30}$$

$$a_y = \sin(\psi)\sqrt{1 - a_\theta^2} \tag{4.31}$$

This set of motions was designed to cover the range of possible motions, excluding ones that have a very high angular velocity, which all effectively just cause the robot to spin in place. This parameterization is based on the one used by Duffert and Hoffman [28].[6]

The constant value that was used for the motion variance, Σ_m, was chosen to be large enough to accommodate the inaccuracy of all the starting velocity estimates being zero. Specifically, it corresponded to standard deviations of 10 mm in each direction and 0.1 radians, at each time step (30 Hz). In preliminary experiments, the algorithm was robust to these values being at least doubled or halved. Note that the different actions each have their own true variances that were not learned in this work. Because these action covariance matrices each have six degrees of freedom, the space of sets of action covariances is a high-dimensional space, making it inherently difficult to search in. These action covariance matrices are nevertheless an important aspect of the action model, and adapting the method for learning the action model presented in Section 4.3.1 to also learn these variances is an important area for future work.

As the robot walked, it scanned its head from side to side to see as many beacons as possible. At each time step, the action command executed and any observation made was recorded. In some frames, no observation was made. When the algorithm processed these frames no EKF observation updates are made, and

[6] Duffert and Hoffman [28] used a parameterization that includes an additional parameter for overall speed.

those frames were omitted in the sensor model reestimation. To attenuate the effect of false positives in object recognition, outlier observations were pruned in the first iteration of EM by discarding observations that represented too large of an innovation in the forward Kalman filter. Specifically, the observation is discarded if $\text{abs}(o_{t,2} - h(w_t)_2) > 0.6k$ radians, where k is the number of time steps that have elapsed since the previous observation was made. The robot ran at its native frame rate of 30 time steps per second.

The training run lasted for 15 minutes, with each of the 40 actions being executed roughly four times for five seconds at a time. As in the previous chapter, a control policy is needed that enables the robot to explore the full range of actions and states. To meet this constraint, after every five seconds a new action was selected randomly, with priorities being placed on staying on the field and on distributing the executed actions evenly. As a matter of convenience, a previously developed accurate localization module was used to help choose actions that keep the robot on the field. This measure would not be necessary if a larger field were used.

The parameter estimation algorithm was run on the resulting data set until convergence, defined as follows. In an ideal setting, EM is guaranteed to converge with the overall likelihood increasing with every iteration. However, approximations made, such as the linearization for the EKF, the assumption of Gaussianity, and drawing samples from a distribution, cause the overall likelihood, $P(O|C, \lambda)$, to fluctuate. The learning is considered finished when 50 iterations pass without a new highest likelihood, indicating that these fluctuations have started to overshadow the learning. On the data collected by the robot, the algorithm took 152 iterations to converge. The learning curve is shown in Figure 4.2, with the log likelihood computed as described in Section 4.2.[7] The entire process took 15 minutes of data-

[7]Recall from Section 2.1 that the magnitudes of these likelihood densities depend on the units in which the random vector is measured. The 5639 observations that comprise O each have one component measured in pixels and one measured in radians.

collection plus about 10 minutes of offline processing on a 2.79 GHz Pentium 4 processor.

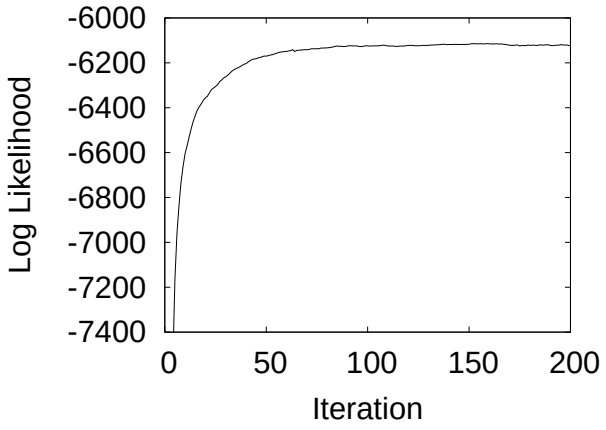


Figure 4.2: This learning curve shows the computed total log likelihood, $P(O|C, \lambda_i)$ during each iteration of EM. The log likelihood of the starting parameter estimates was -23000.

The sensor model consists of a polynomial function from distances to beacon heights, a variance for those beacon heights, and a variance for the observed landmarks' horizontal angles. For each quantity, a starting value was used that was very inaccurate and the learned value was compared to the measured actual value. Specifically, the starting polynomial was a very poor linear model, shown in Figure 4.3. Preliminary experiments suggested that a significantly less accurate starting sensor model would cause the learning to diverge.

The actual sensor model was measured as follows. The robot was placed at distances from the beacon every 100 mm from 1275 to 4175 mm, the range of distances at which the robot can recognize the beacon. Distances were measured from the center of the robot's body, because this is the point from which the horizontal angle observations are computed. At each distance, 100 observations were made, and the mean and variance of the beacon heights and angles were computed. During this time the robot was stepping in place and scanning its head from side

to side, to replicate the conditions during the learning. The actual beacon height means are shown as the measured sensor model and compared to the learned model in Figure 4.3.

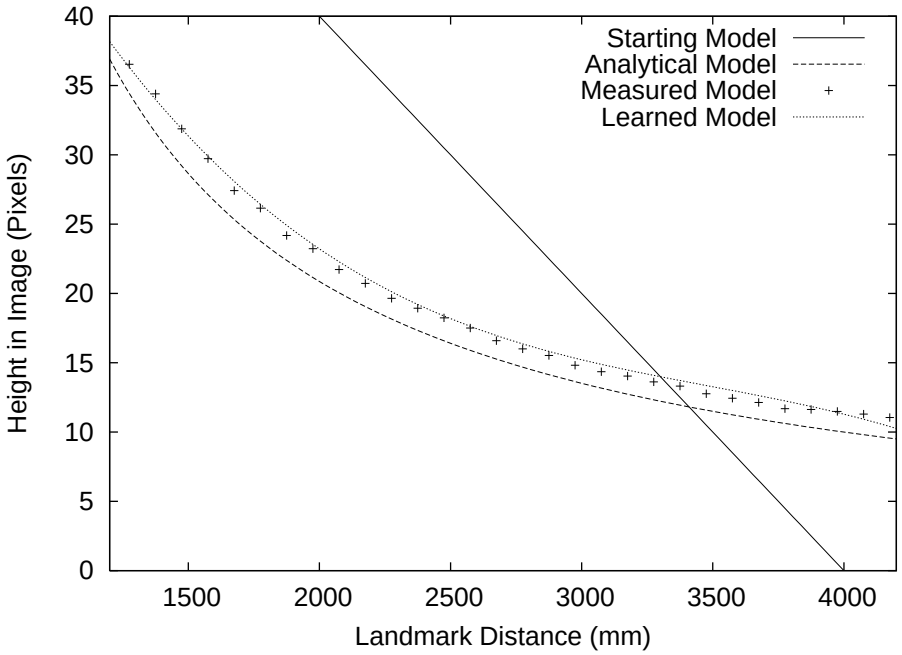


Figure 4.3: The starting, learned, and measured sensor models. For comparison, an analytical sensor model is also shown, derived from the specifications of the camera. The learned model successfully approximates the measured one.

To obtain the measured beacon height and angle variances, the variances at each distance were averaged, each weighted by the frequency with which that beacon distance occurred during the learning. The starting, measured, and learned standard deviations for the beacon height were 10, 1.59, and 1.69 pixels respectively, an error of 6.3% in the learned value. The starting, measured, and learned angle standard deviations were 0.2, 0.0267, and 0.0116 radians, an error of a factor of 2.3. This error was likely due to the very small magnitude of the actual angle variances being dwarfed by the uncertainty in the robot's orientation at any time. This hypothesis is supported by the results in the following section, in which a much

larger simulated angle variance was learned accurately.

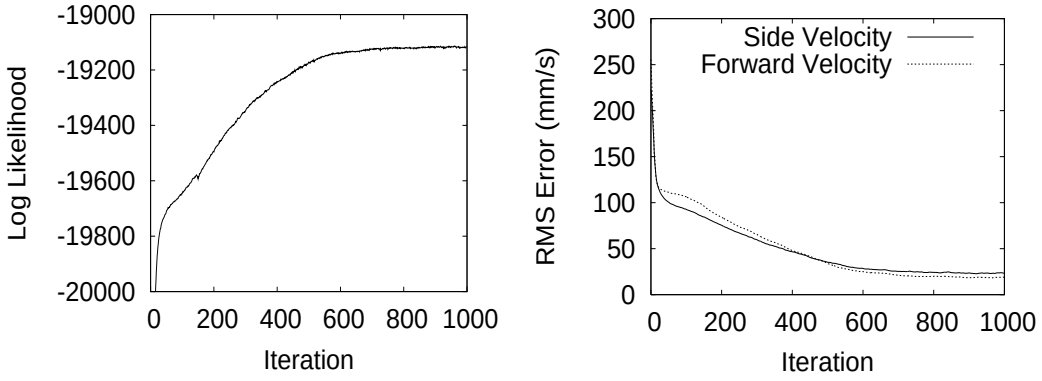
For the action model, the starting velocity estimates were all zero. Rotational velocities were measured for each of the 40 actions. Each one was measured by allowing the robot to execute that action for 30 seconds and measuring the total angular change. The RMS difference between the measured and learned angular velocities was 0.135 rad/s, a relative error of 3.2% compared to the the measured range of angular velocities, 4.21 rad/s. By contrast, the original "attempted" angular velocities, which were manually calibrated (and not used by the robot), had an RMS error of 0.331 rad/s, a relative error of 7.9%.

4.4.2 Simulation Results

The above experiment was also run in the exact same way in simulation, using the same starting action and sensor models. The simulation engine models the robot's pose and observation vectors, but not its physical joint angles or camera image. This experiment verified that the method was able to learn accurate action and sensor models, including the translational velocities of the action model. The simulator's time steps represented 0.05 seconds. The observations were computed by applying a simulated "actual" sensor model to the distances and adding gaussian noise to yield beacon height and angle observations. Gaussian noise was also added to the robot's motion at each time step.

The algorithm converged on the simulated data after 959 iterations, with the learning taking roughly one hour. The learning curve is shown in Figure 4.4a). Each action's learned velocities were compared to the ground truth. The final RMS errors in v_x, v_y, and v_θ were 23.06 mm/s, 18.34 mm/s, and 0.086 rad/s, relative errors of 3.2%, 2.2%, and 2.7%, respectively, with respect to the total range of velocities in those three directions, namely 710 mm/s, 840 mm/s and π rad/s. The x and y velocity RMS errors are shown decreasing over the course of the EM itera-

tions in Figure 4.4b). Note that this RMS error improvements mirror the overall log likelihood improvements shown in Figure 4.4a). This similarity provides confirmation that the log likelihood is a useful measure of the accuracy of the models being learned.



a) b)

Figure 4.4: a) The log likelihood of the 8659 observations improves over the course of 1000 iterations of EM. The starting model had a log likelihood of -50881. b) As the action model estimates converge, the average velocity errors decrease. The angular velocities (not pictured) converge within the first 100 iterations.

Additionally, the standard deviations of the Gaussian noise added to the two observation components in the simulator were 1 pixel for the beacon height and 0.5 radians for the observed horizontal angles. The learned values of these standard deviations were 0.980 pixels and 0.474 radians, errors of 2.0% and 5.3% respectively. The learned, starting, and actual sensor model functions are shown in Figure 4.5.

4.4.3 Additional Results

This section presents additional results that illustrate two important capabilities of the algorithm presented in this chapter. First, I show that the EM-based approach can be adapted to learn models in the one-dimensional domain considered in Chapter 3. Second, I demonstrate the possibility of periodically relearning the action and

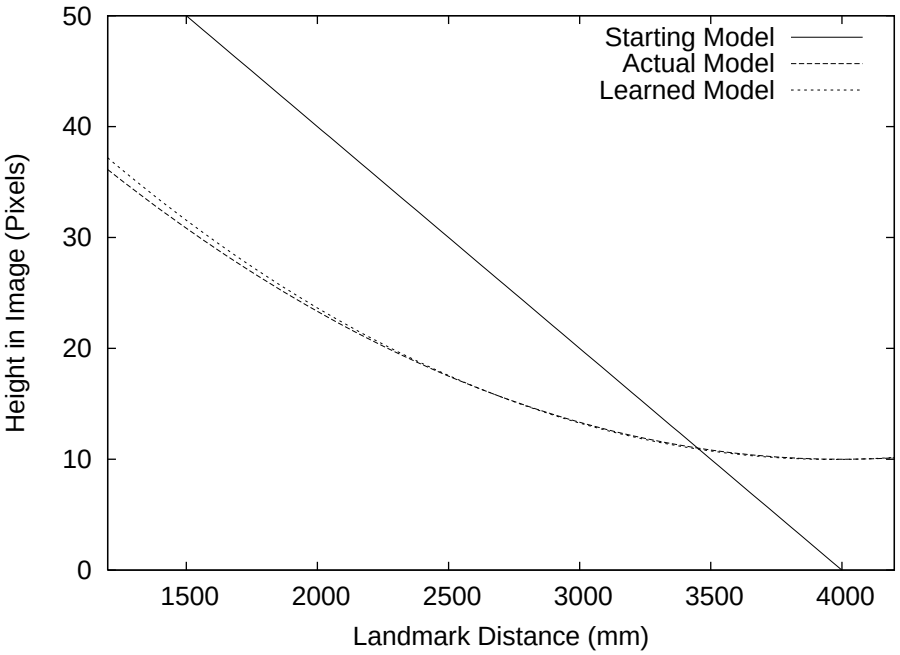


Figure 4.5: The learned sensor model closely matches the actual one.

sensor models to accommodate the models changing over time.

Expectation-Maximization in One Dimension

Compared to the algorithm presented in Chapter 3, the model learning framework presented in this chapter was motivated by the additional challenges raised by a more complex two-dimensional domain. Specifically, the algorithm presented in Sections 4.2 and 4.3 is designed to accommodate a domain with perceptual aliasing, especially that which is caused by the world state space having a higher dimensionality than the observations. This section tests the hypothesis that the EM-based model learning technique presented in this chapter can also be effective in a domain without these complications, specifically the one-dimensional domain used in Chapter 3.

Adapting the EM-based technique to a one-dimensional world state does

not require any significant modifications to the algorithm described in Sections 4.2 and 4.3. The learned action model maps each of a suite of 21 actions onto its corresponding velocity. The robot used actions whose normalized attempted forward velocities, a_y, were the multiples of 0.1 from -1 to 1. These actions were executed with the attempted rightward and turning velocities, a_x and a_θ, set to zero. The action standard deviation used in the learning was 10 mm. The learned sensor model was the polynomial function f that maps the robot's distance to the landmark onto the landmark's height in pixels in the camera image.

The experimental setup used to collect the data was the same as the one described in Section 3.3.2, with alternating phases of forward actions and backward actions. However, the learning was performed offline in batch, as needed for the EM-based approach. The initial sensor and action models used were the same as in Sections 4.4.1 and 4.4.2, namely a poor linear model for the starting sensor model and the assumption of no motion for the starting action model.

Data was collected over the course of eight minutes of the robot walking forward and backward at different speeds. EM converged after 639 iterations, taking just 2 minutes and 14 seconds. As in Chapter 3, the experimental setup only provides information about relative distances and velocities, not absolute ones measured in meters or meters per second. The robot therefore learns the models in its own arbitrary units, although the action and sensor models should be learned in units that are consistent with each other. In order to evaluate the learned models, they were optimally scaled when compared to the measured models, as in Section 3.3.2. The learned and measured action and sensor models are depicted in Figure 4.6.

The RMS error in the learned sensor model shown in Figure 4.6a) was 0.83 pixels, an error of 2.52% compared to the total range of observed landmark heights. Note that this error is not directly comparable to the sensor model error learned in Chapter 3, because there the inverted sensor model was learned. Its average error

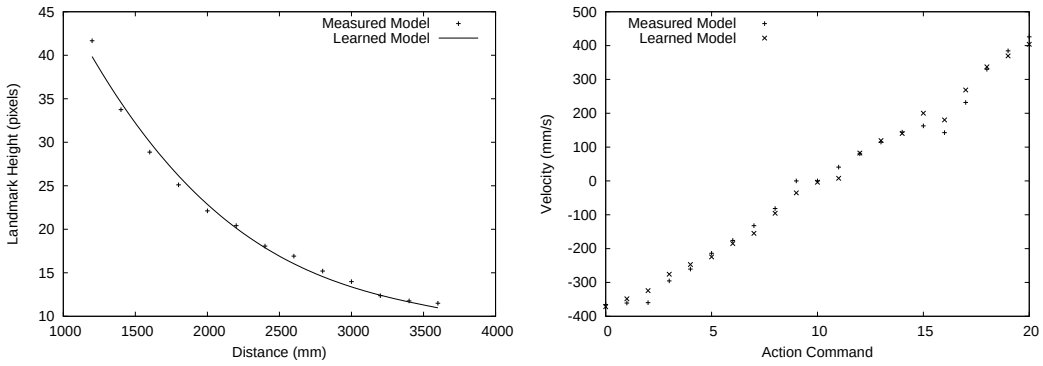


Figure 4.6: a) The learned polynomial sensor model closely matches the measured ground truth observations. b) Similarly, the learned table-based action model closely matches the measured one.

was 70.4 mm, or 2.9% of the range of distances. For the action model shown in Figure 4.6b), the RMS error was 22.1 mm/s, an error of 2.80% of the total range of velocities. By comparison, the average action model error observed in Chapter 3 was 29.6 mm/s. Note however that the table-based action model function used here has more expressive power in its ability to represent different action models than the fourth degree polynomial used in Chapter 3. The ratio between the scaling factors that were determined for the action and sensor models was 0.91, showing that the two learned models were relatively consistent with each other.

Learning a Changing Model

This section describes an experiment in which the robot's models change over time. The possibility of such a change, for example because of a new environment's terrain or lighting conditions, or from the robot's parts wearing down over time, is a primary motivation for the work presented in this dissertation. By having the robot periodically record a series of actions and observations and apply the learning algorithm to that data series, it can be ensured that after a change the models do

not stay miscalibrated for very long. Specifically, when a change occurs, the next complete data series that is recorded and analyzed can be expected to yield the newly accurate action and sensor models.

This experiment is performed in the model learning setting described above, where the EM-based algorithm is adapted to a one-dimensional domain. Sixteen minutes of data were collected from the robot walking forward and backward at different speeds. At the halfway point, the robot's underlying motions were changed, simulating a degradation of the robot's motors. This change was accomplished by multiplying the attempted velocities by 3/4 before they were converted into low-level motions. The eight minutes of data from before the change are those used in the experimental results reported above and shown in Figure 4.6. On that data, the EM-based algorithm was able to learn accurate one-dimensional action and sensor models, as described above.

The eight minutes of data from after the change were treated as a separate batch of data and used to learn a second action and sensor model. The learning process converged on this data in 2906 iterations of EM, taking 9 minutes and 17 seconds. The new learned action model is compared to the new ground truth in Figure 4.7.

The RMS error of the learned velocities after the change was 16.53 mm/s, 2.78% of the range of observed modified velocities. At the same time, the sensor model was also learned accurately, with an error of 0.72 pixels or 2.18%. In this case the ratio between the two scaling factors was 1.03. These average errors were all smaller than those reported above for the models learned from the data recorded before the actions changed.

To illustrate the improvement provided by relearning the models, we also evaluated the model learned before the change as an estimate of the ground truth after the change. The resulting average action model error is 33.9 mm/s, over

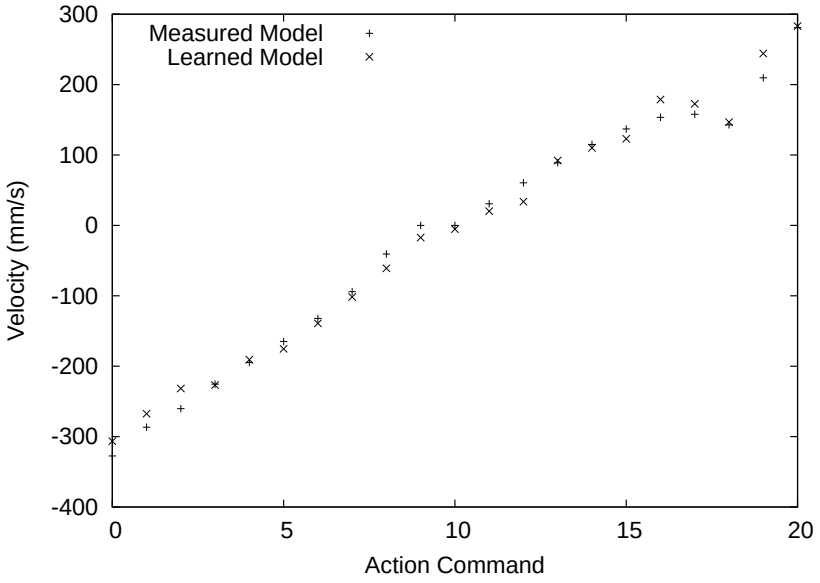


Figure 4.7: After the action model changed it was accurately relearned.

1.5 times the two action model errors reported above. Additionally, in this case the ratio between the scaling factors was 0.65. The difference between this ratio and 1 indicates a significant inconsistency between the learned model combination before the change and the ground truth combination after the change. Conversely, evaluating the models learned after the change using the ground truth from before it, we get an average action model error of 43.3 mm/s, or 5.48%, and a scaling factor ratio of 1.42, again indicating a significant inconsistency.

This chapter presented an algorithm that enables a legged robot walking in two dimensions to learn accurate models of its action and sensor models, starting without an accurate estimate of either model. The following chapter addresses the challenges that arise in adapting the model-learning methodology presented in Chapter 1 to a different autonomous mobile robot: a self-driving car with no prior knowledge of the map of its environment.

Chapter 5

Model Learning on an Autonomous Car

In the previous two Chapters, techniques were presented that are able to autonomously learn the action and sensor models of a small legged robot in a known environment. In this chapter, the methodology presented in Chapter 1 is adapted to learn the models of a large wheeled robot in an unknown environment, specifically an autonomous car. Recently, research in autonomous cars has been spurred on by the DARPA Grand Challenge, a competition for driverless cars [9, 55, 63]. The experiments reported in this chapter have been performed on an autonomous car that was developed for the DARPA Urban Challenge by the University of Texas at Austin and Austin Robot Technology. The car's hardware and its control algorithms are described in the Austin Robot Technology technical reports [9, 16, 83], and the model of the car learned in this chapter was presented in Section 2.5.2.

Recall from that section that for the car the structure of the environment is treated as an unknown aspect of the world state. Thus applying the EM-based technique presented in the previous chapter would require reasoning explicitly about probability distributions over the shape of the environment. One possibility might

be to use three-dimensional occupancy grids for this purpose. However, preliminary experiments suggested that such a structure would be prohibitively computationally expensive at the resolution needed to learn accurate models.

Additionally, recall from Chapter 1 that the model-learning techniques presented in this dissertation are made possible by the concept of perceptual redundancy, depicted in Figures 1.2 and 1.3. The mobile robot scenario discussed in this chapter, however, differs from those presented in the previous two chapters in two significant ways. First, as mentioned above, the map of the environment is treated as an unknown aspect of the world state. Second, as described in Section 2.5.2, the car's Velodyne sensor reports data at a high bandwidth: 64 lasers, each at a rate of over 7800 Hz. These differences alter the "flow" of perceptual redundancy, as depicted in Figure 5.1.

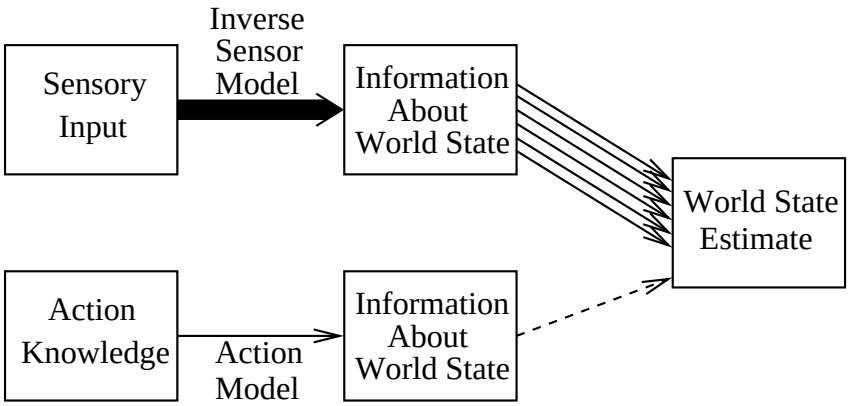


Figure 5.1: For the autonomous car, the information coming in from the sensor is highly redundant in and of itself. The action knowledge, on the other hand, is insufficient to yield complete estimates of the world state, which includes the map of the environment.

Because the action knowledge is insufficient to estimate the world state by itself, it cannot be used directly to provide training data for the sensor model. At the same time, the Velodyne by itself provides redundant information about the combination of the robot's pose and the state of the environment. This situation

suggests the following general plan for learning the car's models. First, the sensor model is learned, based on just the Velodyne data. Second, the car's motion over time is determined from the resulting sensory information. Finally, these motions are combined with the car's actions to learn the action model. The following three sections present algorithms that accomplish these three steps. Section 5.4 demonstrates empirically that these algorithms are able to learn the car's action and sensor models, starting without *any* initial estimate of either the action or sensor model. The model learning process operates in batch on a data set consisting of actions and sensations that the car recorded while driving for a short period of time.

5.1 Learning the Sensor Model

Recall from Section 2.5.2 that the aspect of the sensor model that is learned is the set of horizontal angles of the Velodyne lasers, $\hat{\theta}_l$ (defined as $\theta_l + \theta_b$, the angle of the Velodyne base), where l goes from 0 to $n-1$. The first step in learning these angles is to determine the $n-1$ differences between the horizontal angles of vertically adjacent lasers, $\hat{\theta}_l - \hat{\theta}_{l-1}$, as follows.

The velodyne rotates at a known constant speed of ω so that θ_r, the angle of rotation reported by the Velodyne, is equal to $\omega t \pmod{2\pi}$.[1] (Time 0 can be chosen to be one at which θ_r is 0.) Therefore at time t the lth laser points in direction $\hat{\theta}_l + \omega t$. Additionally, recall that the n lasers are ordered by their vertical angles, so that vertically consecutive lasers have consecutive indices.

The sensor model is learned based on the principle that if two observed points are close to each other in their vertical and horizontal angles from the Velodyne center, they should have similar distances as well. Specifically, consider a pair of vertically consecutive lasers, $l-1$ and l. At any one time, the two lasers' horizontal angles differ by $\hat{\theta}_l - \hat{\theta}_{l-1}$. However, readings from the two lasers at times that differ by

[1] The notation (mod 2π) shall be omitted from this point on.

$(\hat{\theta}_l - \hat{\theta}_{l-1})/\omega$, denoted as δ, will be pointing at the same horizontal angle. Since the car will not have moved much during that time and the vertically consecutive lasers have close vertical angles, the distances from those two readings can be expected to be relatively close to each other. One natural measure of this closeness is the normalized cross-correlation, a function of δ:

$$K_l(\delta) = \text{corr}(d_{l-1}(t), d_l(t - \delta)) \tag{5.1}$$

where $d_l(t)$ denotes the distance reading from laser l at time t and the correlation is taken over all times t for which laser $l-1$ reports a valid distance reading at time t and laser l does so at time $t - \delta$.[2]

For each l, the normalized cross-correlation is evaluated for each value of δ ranging from $-\pi/\omega$ to π/ω over the integer multiples of $1/f$, the temporal resolution of the sensor. The values of δ that maximize $K_l(\delta)$ are multiplied by ω to yield the learned differences $\hat{\theta}_l - \hat{\theta}_{l-1}$.

These $n-1$ differences determine the angles themselves, up to a constant offset. That is, defining C to be $\hat{\theta}_0$, $\hat{\theta}_1 - C$ through $\hat{\theta}_{n-1} - C$ can be determined by accumulating the angle differences between consecutive lasers: $\hat{\theta}_k - C = \sum_{l=1}^{k} (\hat{\theta}_l - \hat{\theta}_{l-1})$. This offset, C, corresponds to the angle of the entire set of lasers with respect to the car. A method for determining C is presented at the end of the following section based on knowledge of the car's motion over time.

5.2 Constructing the Odometry

The previous section presented a method for determining the horizontal angles of the Velodyne lasers up to a constant offset C: $\hat{\theta}_l - C$, which we denote as θ'_l. Now, consider a coordinate system, denoted as (x_C, y_C), that is centered at the Velodyne

[2]Invalid distance readings arise from sensor noise or from the laser not hitting any objects within its range.

center but oriented an angle C counterclockwise from the coordinate system of the car (in which the y-axis points forward). In this coordinate system, a laser reading with distance d and unit rotation θ_r corresponds to an object point at distance d and known horizontal angle $\theta_r + \theta'_l$. By comparing scans at nearby times, the motion of the car with respect to (x_C, y_C) is determined. This information is used to determine both C and the action model.

The problem of inferring a robot's motion from range scans is a well studied one. One common approach is map matching [75, 79], in which each scan is compared to a map of the environment that has either been learned or provided to the robot. Alternatively, the approach of scan matching [5, 35, 51] first determines the relative motion of the environment with respect to the robot and inverts it to yield the robot's motion. Identifying the transformation between the two sets of points is done primarily through variants of the Iterative Closest Point (ICP) method [10, 105], described below. In the experiments reported in Section 5.4, the robot's motion is learned through scan matching with ICP.

Although there are a number of variants of ICP that improve its robustness (e.g. [21, 65]), these methods are still sensitive to outliers in the data (albeit to varying extents), and they require an initial guess that is relatively close to the correct transformation. In this work, a relatively straightforward implementation of ICP is used, illustrating the robustness of the technique presented in this chapter. For the initial guess, at first the identity transformation is used, corresponding to a starting estimate of no car motion. The following section describes an iterative process by which, once an action model is learned, it can be used to provide improved starting guesses for ICP, which can in turn be used to learn a more accurate action model.

Although ICP can be applied to learn transformations between sets of points in either two or three dimensions, certain difficulties arise in applying ICP directly

to the three-dimensional sets of points produced by the Velodyne. The primary difficulty arises from the way in which the Velodyne lasers' rays intersect the ground plane. As seen in Figure 2.4b), these intersections form circular patterns on the ground. Furthermore, as the car moves, the ground typically remains invariant with respect to its motions, causing these circular patterns to remain stationary with respect to the car, leading to incorrectly learned non-motion. To circumvent this problem, points that are over a meter below the height of the Velodyne are discarded from the scans and the car's motion is learned in two dimensions.

The set of points observed by the laser as it rotates through an angle of 2π are considered as a single scan. At times t that are a multiple of a fixed interval, Δt, scans are recorded. Scans from consecutive times are matched to each other to determine the car's motion over the corresponding time. An example of two consecutive scans is shown in Figure 5.2.

Given two consecutive scans, we define the one with fewer points to be D_1 and the other one to be D_2. The goal of ICP is to identify the matching function $\mu : D_1 \mapsto D_2$ and the rigid transformation T that minimizes the RMS error:

$$\sqrt{\frac{1}{|D_1|} \sum_{p \in D_1} ||T(p) - \mu(p)||^2} \tag{5.2}$$

The basic ICP algorithm is presented in Algorithm 2.

Algorithm 2 The Iterative Closest Point algorithm.

1: $T \leftarrow T_{init}$
2: **repeat**
3: **for** each $p \in D_1$ **do**
4: define $\mu(p)$ to be the point in D_2 that minimizes $||T(p) - \mu(p)||$.
5: **end for**
6: Identify the transformation T that minimizes Expression (5.2).
7: **until** convergence

Step 4 can be done in $O(\log|D_2|)$ time by representing the scans with kd-

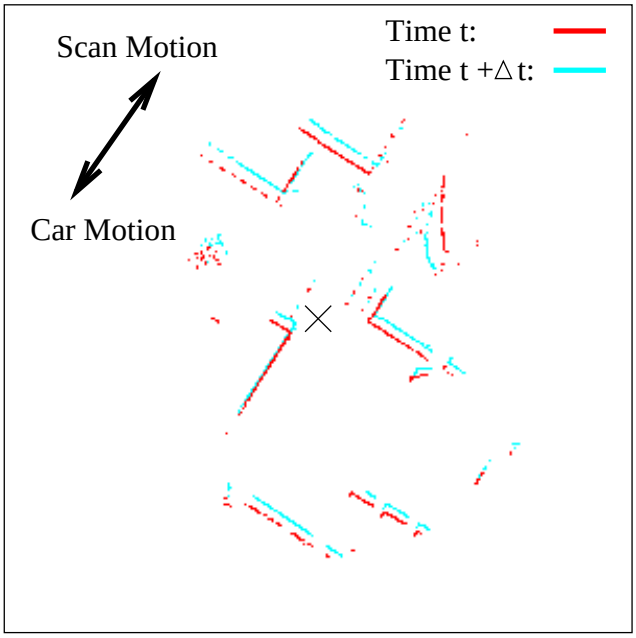


Figure 5.2: The red and blue scans are sets of points observed by the Velodyne at times that differ by Δt. The black $\times$ represents the location of the sensor. The transformation between the two scans is inverted to determine the motion of the car. The area depicted here has a sidelength of 300 meters.

trees [32]. Step 6 can be solved exactly in $O(|D_1|)$ time by using rotation matrices and the singular value decomposition [37]. In the results presented in this paper, the transformation is approximated via a least squares regression based on treating T as a function of d_x, d_y, and d_θ, the x, y, and θ components of the displacement between the scans: $T(p) = R(d_\theta)p + (d_x, d_y)^\top$. This function is linearized around the value of T from the previous iteration, enabling the parameters d_x, d_y, and d_θ to be estimated by least squares regression. Additionally, points p for which the distance $||T(p) - \mu(p)||$ is above a fixed threshold[3] are discarded in this computation of T in step 6.

If the scan D_1 preceded D_2, the transformation T is inverted to yield the car's motion, M. If instead D_2 was the earlier scan, M is equal to T. The rightward,

[3]The value used for this threshold was 7 meters.

forward, and rotational components of M, denoted as m_x, m_y, and m_θ, yield the learned rightward, forward, and angular velocities of the car at each time, with respect to coordinate system (x_C, y_C): $m_x/\Delta t$, $m_y/\Delta t$, and $m_\theta/\Delta t$. Since the vector (m_x, m_y) is originally in the reference frame of the earlier scan, it is first adjusted by a rotation through an angle of $-m_\theta/2$ (which never exceeded three degrees) to more accurately reflect the average translational motion over the time step.

As mentioned above, the car's motion can now be used to determine the sensor model's constant offset, C. This constant is determined based on the following straightforward definition: Forward is the direction in which the car moves. Even though the car's motion has a rotational component when it turns, it should not have a significant average sideways component. Since (m_x, m_y) represents the car's motion with respect to a coordinate system that is rotated counterclockwise through an angle of C, the vector (m_x, m_y) will point in an angle of $\pi/2 - C$ when the car moves forward along its y-axis. Therefore, C can be computed by taking an average of the angles of the different (m_x, m_y) vectors and subtracting the result from $\pi/2$.

In order to eliminate error caused by outliers in the values of (m_x, m_y), first those values that correspond to a speed below a certain threshold[4] are discarded. Then, to further reduce the effect of outliers, an *angle median* is used: the angle with the minimum total absolute distance from the angles of all of the remaining (m_x, m_y). This median angle is subtracted from $\pi/2$ to yield our estimate of C. This value of C is then added to the relative horizontal laser angles θ'_l to yield the correct absolute angles, $\hat{\theta}_l$.

[4]The value used for this threshold was 2 m/s.

5.3 Learning the Action Model

Recall from Section 2.5.2 that the car's acceleration and angular velocity are modeled as function of the action command, $c_t = (g_t, b_t, s_t)$, and the forward velocity:

$$a_t = C_5 + C_6 g_t + C_7 v_t + C_8 v_t b_t \tag{5.3}$$

$$\omega_t = C_9 v_t + C_{10} v_t s_t \tag{5.4}$$

$$v_t = v_{t-1} + a_{t-1}\Delta t \tag{5.5}$$

The aspect of the action model that we wish to learn is the constants C_5 through C_{10}. At each time step, the procedure discussed in the previous section yields values of $\omega_t = m_\theta/\Delta t$, v_t: the component of (m_x, m_y) in the direction of $\pi/2 - C$, and $a_t = (v_{t+1} - v_t)/\Delta t$.

These values of v_t, a_t, and ω_t, in conjunction with the recorded action commands c_t, comprise the training data that is used to learn the action model. Linear regression is performed on Equation 5.3, yielding the values of C_5 through C_8 that minimize the mean squared error in that equation over t. Similarly, linear regression is performed on Equation 5.4, yielding the learned values of C_9 and C_{10}.

The learned action model can be used to predict the car's motion by applying Equation 5.5 first in each time step to determine v_t, and then using this value in Equations 5.3 and 5.4 to estimate a_t and ω_t. The derived values of v_t and ω_t can now be used to determine more accurate starting guesses, T_{init}, for the ICP procedure discussed in the previous section: $m_x = 0$, $m_y = v_t \Delta t$, and $m_\theta = \omega_t \Delta t$. Since these values of T_{init} are more accurate than the original values, namely the identity transformation, the resulting motions suggested by ICP should also be more accurate. In turn, these motions can be used to learn the action model again, which should again be more accurate than the first one. This process is iterated until convergence.

Algorithm 3 summarizes the process by which the car's action and sensor models are learned.

Algorithm 3 Model learning on an autonomous car.

1: Learn the relative horizontal Velodyne angles, $\hat{\theta}_l - C$, as described in Section 5.1.
2: **repeat**
3: Use scan matching to estimate the car's motion at each time step, using the action model to generate starting guesses, if available.
4: Use the estimated motion to determine C.
5: Use the estimated motion to determine the action model parameters: C_5 through C_{10}.
6: **until** convergence

5.4 Empirical Validation

The method described above was empirically validated on the Austin Robot Technology autonomous car, described in Section 2.5.2. Over the course of 200 seconds of driving autonomously during which the car made three left turns, the car recorded all of its action commands and Velodyne observations.

In the execution of the model learning, the value of Δt used was 0.5 seconds. Because actions were recorded at a higher frequency than this (roughly 20 Hz), the recorded actions were averaged over each Δt to yield the corresponding values of $c_t = (g_t, b_t, s_t)$. The iterative learning process converged after four iterations. The model learning was performed offline on a 2.79 GHz Pentium 4 processor.

For validation purposes, the car was additionally equipped with an Applanix Pos LV system, a sensor that combines GPS and inertial motion data [102] to provide pose estimates with accuracies of under one meter and 0.02° [83]. The difference between the pose reported by the Applanix at times that differ by Δt are used to construct the ground truth estimates of the car's forward and angular velocities. These velocities are used to evaluate the learned action models, as described below.

To evaluate the learned sensor model, it was compared to ground truth values for the lasers' horizontal angles that were manually calibrated by Velodyne. These values are interpreted as the raw θ_l described in Section 2.5.2. Recall from that section that the learned angles, $\hat{\theta}_l$, are defined as $\theta_l + \theta_b$, the angle at which the Velodyne is affixed to the car. The Velodyne and Applanix are both affixed and aligned to a frame that is in turn affixed to the top of the car. However, data from the Applanix suggests that this frame is not exactly aligned with the direction of the car's motion. That is, while the car was moving forward, the median angle of the motion observed by the Applanix was $0.9°$, suggesting that the Velodyne and Applanix are oriented at an angle of $-0.9°$ with respect to the car's motion, or $89.1°$ with respect to the car's x-axis. Accordingly, the value of θ_b used was $89.1°$. The resulting ground truth angles are compared to the learned ones in Figure 5.3. The RMS angle error taken over the 64 lasers is $0.544°$. This error is 3.0% of the range of true horizontal angles, $18°$.

To evaluate the action model, estimates of the car's accelerations, forward velocities, and angular velocities were generated by applying the model to the actions taken by the car. The resulting quantities were compared to the ground truth reported by the Applanix. In Figures 5.4 and 5.5, this comparison is depicted over the course of 25 seconds in which the car stops and accelerates through a left turn. Note that although the errors in the acceleration are relatively small, they accumulate over time into moderate errors in the forward velocity.

The RMS errors between the learned and measured values over the entire 400 time steps are shown in Table 5.1, as well as those errors as percentages of the total range of observed accelerations, velocities, or angular velocities, namely 5.79 m/s^2, 6.30 m/s, and 0.202 rad/s. Additionally, for comparison the errors obtained by fitting the action model directly to the ground truth motions is shown in the column labeled From-Truth.

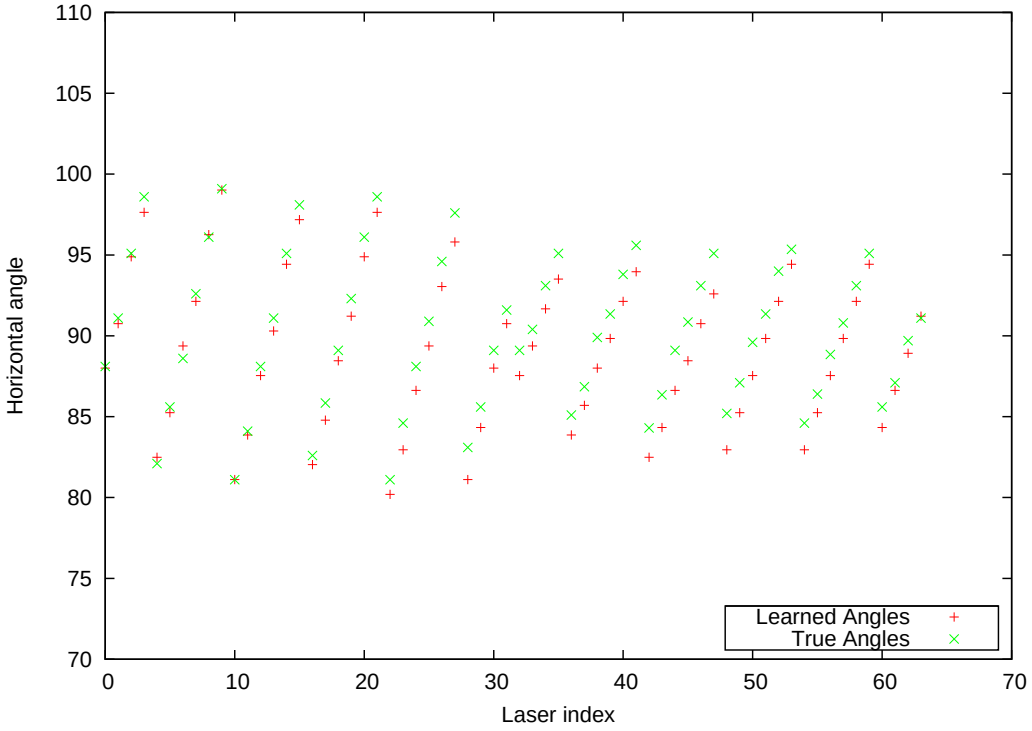
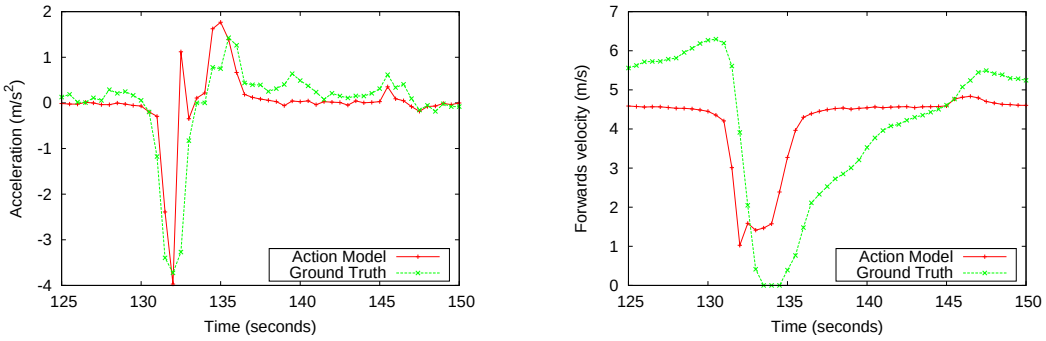


Figure 5.3: The learned sensor model closely matches the ground truth laser angles.

The entire learning process took 3 hours and 40 minutes, broken down as follows. The first stage, learning the relative horizontal angles, took 83 minutes, although this time can be reduced by leveraging knowledge of a maximum absolute horizontal angle difference between consecutive lasers. For example, using a maximum difference of 23° leads to this stage taking 10.7 minutes. The remainder of the learning took 2 hours and 17 minutes, 34 minutes for each of the four iterations.

In Chapter 1, I presented a methodology for learning a mobile robot's action and sensor models, starting without an accurate estimate of either model. The algorithms presented in Chapters 3, 4, and 5 have accomplished this goal in three different robotic settings. In the remainder of the dissertation, the following chapter discusses previous related work and Chapter 7 discusses the contributions of this work and possibilities for future research.



a) b)

Figure 5.4: The learned action model is compared to the ground truth accelerations (a) and forward velocities (b).

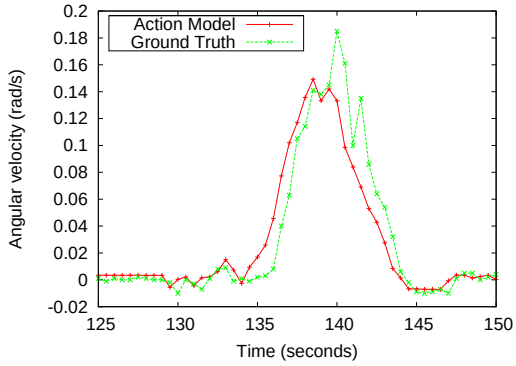


Figure 5.5: The action model yields forward angular velocities that closely match the measured ones.

	RMS Error	% Error	From-Truth
Acceleration	0.392 m/s^2	6.77%	0.254 m/s^2
Fwd. Vel.	1.04 m/s	16.5%	0.64 m/s
Ang. Vel.	0.0129 rad/s	6.39%	0.013 m/s

Table 5.1: The learned action model is evaluated by comparing the resulting acceleration and velocities to the ground truth values.

Chapter 6

Related Work

This dissertation introduces a methodology for enabling mobile robots to learn models of their actions and sensors, starting without an accurate estimate of either model. This chapter discusses the wide range of previous related work. First, Section 6.1 situates this work within the burgeoning field of developmental robotics. Section 6.2 describes previous work in learning sensor and action models individually. Finally, Section 6.3 discusses previous work related to the learning paradigm used in Chapter 4, namely the dual estimation of the parameters of an extended Kalman filter.

6.1 Developmental Robotics

The work presented in this dissertation is motivated in part by the ideals of developmental robotics [11, 52, 100], an approach to artificial intelligence in which a robot autonomously learns about itself and its environment via a general exploration process, as opposed to a task-specific learning algorithm. In order to enable such a general exploration, one possibility is to endow the robot with a *curiosity* mechanism whereby the robot seeks out novel areas of the state space [62, 61]. In the

context of reinforcement learning, where an agent learns to maximize an external reward, an *intrinsic motivation* can be employed to enable the agent to learn a set of task-independent competencies in its environment [80, 81]. Other work in this area has focused on specific challenges that are crucial to the success of a developing embodied agent, such as acquiring a language [48] or learning about the properties of environmental objects [54, 57, 76, 88].

One common goal of developmental learning processes is for the agent to learn about itself and its environment starting from as little innate knowledge as possible. For example, Philipona et al. suggest using sensorimotor dependencies to learn "the structure of reality," including such things as the number of dimensions of the external physical world [64]. Dangauthier et al. show how statistical regularities in the robot's sensorimotor information can be used to acquire new skills in a Bayesian framework [24]. Alternatively, Bowling et al. apply an action-respecting dimensionality reduction to the raw sensory input that yields an ability to predict how actions will affect the agent's sensations [14]. This idea of representing state purely in terms of action-sensation predictions is also employed in a setting with a discrete state space by a predictive state representation (PSR) [49].

Additionally, starting with only uninterpreted sensorimotor information, Pierce and Kuipers enable a robot to learn a tiered representation of its sensations, actions, and environment known as the spatial semantic hierarchy (SSH) [46], starting by learning the relative orientations of a set of range finder elements [66]. Provost et al. additionally show how to combine the SSH with a self-organizing map that learns a set of actions and their effects with respect to salient locations in the environment [67]. On a Sony Aibo, where the primary sensor is a camera, Olsson et al. show how the robot can learn an *informational map* of its camera pixels that corresponds to their physical configuration [60]. The robot then uses this map to learn about the effects of certain actions on its sensations, resulting in the robot

being able to perform visually guided movement.

The model learning processes presented in this dissertation start with *relatively* little innate knowledge. The robot initially has no accurate knowledge of how its actions and sensations correspond with the external world, and the only inputs to the learning are the robot's sensations and action selections; there is no externally generated training data. However, in contrast to some of the work described above, the algorithms presented in Chapters 3-5 do rely on some implicit innate knowledge about the structure of the robot's body and the state space of its world. This structure is inherent in the formulation of the models being learned (i.e. Section 2.5). The algorithms presented here learn model functions and parameters that specify how its sensory input and actions correspond with the state of the world and its changes. These processes rely on less innate knowledge than previous work in model learning, as discussed in the following section.

6.2 Sensor and Action Modeling

In contrast to the work presented in this dissertation, previous work in action and sensor modeling typically assumes the presence of reliable training data. For example, one common type of sensor modeling is camera calibration. Cameras have both intrinsic parameters, such as focal length and distortion factors, and extrinsic parameters, such as the camera's location and orientation. Tsai presents a general method for calibrating all of these parameters by a combination of geometric analysis, linear regression, and nonlinear optimization [97]. This method relies on labeled training data in the form of coordinates of points in the world and their corresponding coordinates in the image plane.

On a mobile robot, training data for the sensor model can be generated based on the robot's action model if it is already known. Different techniques have been used to calibrate models of different types of sensors. In the context of constructing

occupancy grids, Moravec and Blackwell present a method to learn a model of a sonar range finder [56], while Thrun suggests a method for the robot to learn the inverse sensor model of its laser range finder [94, 96]. To model the range of RFID tags, Hähnel et al. represent the sensor model as a piecewise constant probability distribution [36]. For a robot with multiple sensors, one important challenge is to calibrate the relative poses of the different sensors. A range of approaches have been proposed to address this challenge on a robot learning different sets of calibration parameters between a camera and a laser range finder [6, 70, 104]. All of these sensor calibration methods rely on the robot's knowledge of its pose over time, which is based in part on its action model being known and accurate.

Another common type of sensor modeling involves calibrating networks of stationary sensors. For example, Bychkovskiy et al. present an algorithm for calibrating the biases of a network of light intensity sensors [18]. For a network of temperature sensors that suffer from occasional drift, Hines et al. show how an autoassociative neural network can be used to learn to identify and correct this sensor drift [41]. One challenge for a sensor network is to determine their respective locations and orientations. Toward this end, Ihler et al. present an efficient belief propagation technique [42] and Savvides et al. introduce a set of distributed algorithms for precise sensor localization [72]. Additionally, Cevher and McClellan show how an extended Kalman filter can be used to calibrate the orientations of an array of directional sensors [20].

If the robot's pose can be known reliably, this information can be used to provide training data for either model. This approach is taken by Borenstein and Feng, using manually measured robot poses to identify systematic odometry errors on a wheeled robot [13]. Ojeda et al. show how to use a global positioning sensor to train an accurate model of wheel slippage as a function of the wheels' motor currents [59]. Burlet et al. use an overhead camera that can accurately localize the

robot to calibrate both its action and sensor models [17].

If the robot has prior knowledge of an accurate sensor model, its pose can be determined by localization. On an Aibo, Quinlan et al. discuss such a method for calibrating the odometry based on the robot's vision-based localization [68]. Düffert and Hoffmann extend this approach by using an enhanced landmark set that ensures accurate localization [28]. On wheeled robots, other approaches to learning an action model have included using an augmented Kalman filter to estimate odometry errors [47, 53], as well as adaptations of the EM algorithm [8, 29, 30, 71]. These applications of EM are specific to learning an action model. EM has also been used on a mobile robot for Simultaneous Location and Mapping (SLAM) [4, 95]. In that work, the parameters being learned are a map of the environment, not the robot's action and sensor models, which are presumed to already be accurately known.

In order to learn a mobile robot's action or sensor models, all of the above-mentioned work relies on either prior knowledge of the robot's sensor or action models (respectively) or a reliable method of determining the robot's pose over time. However, as an autonomous robot explores novel environments, it cannot necessarily expect to have reliable access to any of this information. The techniques presented in this dissertation address this challenge, enabling a robot to learn its action and sensor models starting without knowledge of its pose over time or an accurate estimate of either model.

Some previous work has learned a small number of model parameters on a mobile robot starting without either model being known. Kaboli et al. use a Markov chain Monte Carlo method to learn five variance parameters of a probabilistic action and sensor model [43]. Censi et al. show how to simultaneously learn three parameters of a wheeled robot's action model and three parameters that specify the relative pose of a laser range finder [19]. Yap and Shelton have suggested using two nested applications of the EM algorithm to learn twelve motion model parameters

and six sensor model parameters for a wheeled robot [103]. However, the parameter values learned in that work are not evaluated against the ground truth properties of the robot. Finally, in a discrete state space, Koenig and Simmons [45] use the Baum-Welch algorithm [7, 69], an adaptation of EM for an HMM, to learn a map as well as a sensor and action model for a simulated robot.

By contrast, the work presented in Chapters 3 and 4 enable a mobile robot to learn aspects of its action and sensor models that are treated as arbitrary functions and learned via a function approximator, specifically polynomial regression. In Chapter 5, the linear coefficients of an action model are learned along with 64 parameters of the robot's three-dimensional laser range finder. Overall, the techniques presented in this dissertation are unique in both the complexity of the models learned and the paucity of the knowledge with which the robots start.

6.3 Dual Estimation of a Kalman Filter

The model learning technique presented in Chapter 4 frames the problem of learning a robot's action and sensor model functions in terms of an extended Kalman filter. In this context, the robot's models correspond to the system dynamics. Dual estimation is the problem of estimating these system dynamics in addition to the unknown world state over time. A number of approaches have been applied to the problem of dual estimation for nonlinear dynamical systems. One such approach is the dual extended Kalman filter, in which two EKFs run in parallel, one that estimates the hidden state and one that learns the system model [22, 58]. Another possible approach is the joint extended Kalman filter, in which the state vector and unknown parameters are combined into an augmented state [23, 50]. Additionally, Abbeel et al. [1] describe a discriminative training method that learns the noise parameters of an EKF.

In Chapter 4, the EM algorithm is used to estimate the EKF parameters, namely the robot's action and sensor models. Ghahramani and Roweis discuss a

number of advantages of using the EM algorithm for dual nonlinear estimation over the joint and dual EKF methods [34]. In particular, EM generalizes well to learning complex models or parameter combinations. This property makes it well-suited to learning the action and sensor model functions for a mobile robot.

When the EM algorithm is applied to dual estimation in a linear system, the E-step is an optimal smoother such as forward-backward smoothing [31]. The M-step yields new parameter settings that can be computed from summary statistics of the E-step distributions [27, 33, 78]. In a nonlinear system, Ghahramani and Roweis use an EKFS for the E-step, and restrict the nonlinear function to a radial basis function with the nonlinear parameters held fixed [34]. Briegel and Tresp compare a number of possibilities for the E-step, including an EKFS, Fisher scoring, and a mixture of Gaussians [15]. For the M-step, they assume the nonlinearities are represented by a neural network, and compute the gradient for a generalized M-step based on samples drawn from the E-step distribution. Assuming that the state features are the weights of a neural network and the process dynamics are linear, de Freitas et al. use an EKFS for the E-step and compute closed form expressions for the result of the M-step in this context [25].

The algorithm presented in Chapter 4 contributes an extension and adaptation of the above approaches that enables learning a robot's action and sensor model functions. The resulting method uses an EKFS for the E-step, and a combination of methods for the M-step. For the sensor model, a novel combination of polynomial regression and drawing samples from the E-step distribution is presented. For the action model, Section 4.3.1 derives a closed form expression for the mean relative displacement corresponding to each of a set of actions.

Chapter 7

Discussion and Future Work

This dissertation considers the following question: How can a mobile robot learn models of its actions and sensations without relying on human supervision or prior knowledge of either model? In this context, the robot has no reliable source of information about its pose over time, and therefore no reliable source of training data for the models being learned. I have presented a model-learning methodology that is algorithmically instantiated to address these challenges in three different robotic scenarios.

First I consider a legged robot, the Sony Aibo ERS-7, in a one-dimensional domain where it walks forward and backward at different speeds while facing a fixed landmark. Second, the same robot's action and sensor models are learned in a richer, two-dimensional domain. In this domain, the robot walks with arbitrary combinations of forward, sideways, and turning velocities, and observations provide the robot with indirect noisy information about its distance and angle to landmarks. Finally, I consider a large wheeled robot: an autonomous car. The method presented for this robotic platform learns a model of the car's acceleration and angular velocity as a linear function of its steering, throttle, and brake positions, as well as a model of the internal horizontal angles of a high-bandwidth three-dimensional laser range

finder. Furthermore, the car is not presumed to have any prior knowledge about landmarks or a map of its environment.

The differences between these three scenarios lead to differences in the corresponding learning algorithms that are introduced. In Chapter 3, where the robot's world state is its one-dimensional pose, the robot's sensations and actions each provide enough information alone to yield a useful estimate of this world state. The method presented therefore simultaneously maintains two estimates of the robot's pose over time, one based on each model, and each is used to provide training data for the other model. The two models are treated as arbitrary functions and approximated through polynomial regression. Because neither model is known accurately initially, a bootstrapping process is introduced that enables both models to gradually grow in accuracy until they both converge to correct estimates of the robot's true models.

The two-dimensional setting considered in Chapter 4 presents the robot with additional challenges. In this domain, the robot's pose is a three-dimensional world state that requires care to estimate even if the robot's models are known accurately. One effective method to accomplish this mobile robot localization is an extended Kalman filter. In this context, the models being learned correspond to the system dynamics of the Kalman filter. These models are learned in a probabilistic framework, with the learning process identifying the models with the maximum likelihood of having produced the data that was observed. This likelihood maximization is achieved by an adaptation of the Expectation-Maximization algorithm. A novel adaptation of the M-step enables the robot to learn a sensor model that combines a polynomial function with variances of the random noise components that are added to the observations. Additionally, a closed form expression is derived for the M-step's estimate of the action model, specified as the velocity combinations corresponding to a discrete set of 40 actions. Because this probabilistically motivated framework

is designed to address the greater challenges posed by the two-dimensional domain, we would expect that the same framework could also be used to learn the robot's models in the one-dimensional setting discussed in Chapter 3. This expectation is confirmed in Section 4.4.3.

Chapter 5 presents an algorithm that is able to learn the action and sensor models of an autonomous car. This robotic scenario has a number of properties that differ from those discussed in Chapters 3 and 4. First, because the car drives through arbitrary environments, the learning algorithm is not able to rely on prior knowledge of the map or structure of the environment. Since the map of the environment is a component of the world state, the action model is not sufficient by itself to estimate this state, and therefore cannot be used to provide training data for the sensor model. At the same time, the car's primary sensor, a three-dimensional laser range finder, provides highly redundant information about the car's environment. Therefore, the sensor model is learned first, based only on the range finder data. Once this learning is accomplished, the sensor data can be used to estimate the car's motion over time, which is used in turn to train the action model. Finally, an iterative procedure is introduced by which the learned action model is repeatedly used to reestimate the car's motions, which are then used to improve the action model's accuracy.

In all three robotic domains, the algorithms presented are validated by comparing the learned models to direct measurements of the properties of the robot's actions and sensations. On the Aibo, the mean and variance of the robot's observations are recorded over a range of distances between the robot and the landmark. Its velocities while executing the different action commands are also measured manually. Chapter 4 additionally presents results learning in a simulation where the ground truth is known. On the autonomous car, an additional accurate global positioning sensor is used to generate ground truth data for the car's motion, while the ground truth sensor model was provided to us by the sensor manufacturer. In

all three settings, the learned models are shown to closely approximate the true properties of the robot's motions and sensations, as desired.

The methods presented in Chapters 3-5 all address the same challenge, learning the action and sensor models without human supervision or prior knowledge of either model. Although the different robotic settings lead to the use of different algorithms, these methods share a number of similarities. First, the methods reason explicitly about *functional* relationships rather than simply sets of parameters. Although many functional relationships, including the ones used here, can be represented in terms of a constant number of parameters, treating the relationships as arbitrary functions opens up the possibility of replacing the function approximators used with qualitatively different ones, such as backpropagation of a neural network [39] or CMACs [3].

Another similarity among the learning processes is the theme of having two intertwined processes each relying on data from the other. In Chapter 3, the two processes are the action model learning and sensor model learning, each of which requires feedback from the other to be effective. In Chapter 4, the action and sensor model learning processes rely on each other indirectly, mediated through the world state distribution by the EM algorithm. On the autonomous car in Chapter 5, the two processes that are intertwined are the scan matching used to determine the car's motion and the action model learning. Specifically, the car's estimated motion and its action model are alternately based on each other in an iterative procedure. These pairs of intertwined processes have proven to be an effective tool for learning multiple models starting with minimal innate knowledge.

Endowing mobile robots with the ability to autonomously learn their action and sensor models can be beneficial in three qualitatively different ways. First, this ability has the potential to be useful for applications in which a robot may need to operate effectively in an unfamiliar environment without human supervision. Some

examples of such areas are space exploration, rescue robots, and robots that assist people in their homes. Second, even if a robot stays in one environment, its parts may wear down over time, causing the properties of its actions and sensations to change over time. Section 4.4.3 demonstrates that the model learning process can be executed repeatedly, learning new correct models after such a change occurs. Third, research in autonomous robotic model learning can benefit manufacturers that produce a large number of robots with the same qualitative structure, but whose action and sensor models differ for each individual robot because of variation in the production process. If each robot can autonomously learn its own action and sensor models, it can eliminate the need for each robot to be manually calibrated.

Looking forward, the work presented in this dissertation represents a starting point for a wide range of potential directions for future research. One important area for future work is to adapt the methodology presented here to other robotic platforms. At this time, enabling each new robotic platform to autonomously learn its action and sensor models presents many new challenges. However, as research along these lines progresses, the techniques that are developed to accommodate various robotic features can be viewed as a suite of tools that may each be applicable to a wide range of robotic platforms that share those features. These robotic features may include aspects of the robot's physical structure, its actuators and sensors, or other properties. Given such a tool set, each new robotic model learning problem can be approached more easily by incorporating the tools that apply to that platform.

The techniques presented in Chapters 3-5 can be thought of as initial contributions to this tool set. For example, Section 5.1 describes a technique for learning the horizontal angles of a specific type of three-dimensional laser range finder. Although this technique is implemented and tested on a specific sensor and robotic platform, the Velodyne HDL-64 on an autonomous car, the cross-correlation technique discussed in that section could be applied to any three-dimensional range

finder with a similar structure to the Velodyne, on any mobile robot.

Ultimately, one long-term goal for robotic model learning is the development of an autonomous model learning algorithm that is not specific to a particular robotic platform. That is, the algorithm would enable a general robot to autonomously learn its action and sensor models, starting with only a description of the robot's physical structure, actuators, and sensors. Such an algorithm would automatically combine the known model learning techniques that correspond to the features of the given robot, while generating the required robot description would only need to be done once for each robotic configuration that is developed.

Another potential area for future work is to reduce even further the method's reliance on human-supplied knowledge. For instance, the work presented in Chapters 3-5 rely on the inputs to the models being correctly identified. One possibility would be to incorporate an algorithm for automatically choosing the features that are the most effective inputs to a function approximator, such as FS-NEAT [101].

Additionally, in this dissertation the primary aspect of the world state considered by the robots was their position and orientation in the environment. Another important component of a robot's world state is the positions and orientations of objects in its environment. Including these object poses in the world state leads to an important area for future work: enabling robots to learn about the geometrical shapes, affordances, and appearances of objects in the environment as they are grasped, pushed, and viewed from multiple angles. This future work may benefit from combining ideas from this dissertation with those from the ongoing research in developmental object learning [54, 88], mentioned in Section 6.1.

Furthermore, this dissertation has focused on the problem of enabling a mobile robot to better understand what its sensations say about the state of the world and how its actions affect that world state. However, another crucial challenge for such an autonomous agent is that of choosing actions that achieve a desired effect.

For example, one natural use of the learned action and sensor models would be as components of a planning algorithm. Such planning could be used to bring the robot to a desired pose or carry out a complex task. Another approach to action selection is reinforcement learning, where the agent learns through repeated trials which actions lead to an external reward [93]. In this context, action and sensor models such as those learned in this work could be used to plan a path to the rewarding states.

Finally, as a developing robot explores its environment, one important question is how it can choose actions that enable it to learn about new things. A number of mechanisms described as intrinsic motivation or adaptive curiosity have been proposed to maximize the robot's learning progress in this way [62, 61], as mentioned in Section 6.1. Given a mobile robot that is able to learn its action and sensor models, the ability to autonomously seek out "interesting" areas of the state space may be able to dramatically expedite this model learning process.

The work presented in this dissertation addresses the challenge of enabling a mobile robot to reliably and accurately know what its sensations are saying about the state of the world, and how its actions influence that state. Compared to previous work, the algorithms presented in Chapters 3-5 expand the complexity of the action and sensor models that the robot can learn, while reducing the amount of innate knowledge that is required by the learning algorithms. As the field of intelligent autonomous robots progresses, the continued development of learning algorithms that learn as much as possible, starting with as little knowledge as possible, promises to continue to improve the intelligence, autonomy, and overall effectiveness of such robots.

Bibliography

[1] P Abbeel, A Coates, M Montemerlo, A Ng, and S Thrun. Discriminative training of Kalman filters. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[2] Hirotugu Akaike. Information theory and an extension of the maximum likelihood principle. In *Proc. Second International Symposium on Information Theory*, Budapest, 1973.

[3] J. S. Albus. *Brains, Behavior, and Robotics*. Byte Books, Peterborough, NH, 1981.

[4] W Burgard an D Fox, H Jans, C Matenar, and S Thrun. Sonar-based mapping with mobile robots using EM. In *Proc. of the International Conference on Machine Learning*, 1999.

[5] K Arras and S Vestli. Hybrid, high-precision localisation for the mail distributing mobile robot system MOPS. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 1998.

[6] C Barat, J Triboulet, Y Chekhar, and E Colle. Modelling of a camera-3d range finder system. *Robotica*, 15:225–231, 1997.

[7] L. Baum, T Petrie, G Soules, and N Weiss. A maximization technique oc-

curring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41:164–171, 1970.

[8] P Beeson, A Murarka, and B Kuipers. Adapting proposal distributions for accurate, efficient mobile robot localization. In *IEEE International Conference on Robotics and Automation (ICRA-06)*, 2006.

[9] P Beeson, J O'Quin, B Gillan, T Nimmagadda, M Ristroph, D Li, and P Stone. Multiagent interactions in urban driving. *Journal of Physical Agents: Multi-Robot Systems*, 2(1), March 2008.

[10] P J Besl and N D McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[11] D Blank, D Kumar, and L Meeden. A developmental approach to intelligence. In Sumali J. Conlon, editor, *Proceedings of the Thirteenth Annual Midwest Artificial Intelligence and Cognitive Society Conference*, 2002.

[12] J Borenstein, B Everett, and L Feng. *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., 1996.

[13] J Borenstein and L Feng. Correction of systematic odometry errors in mobile robots. In *Proceedings of the 1995 International Conference on Intelligent Robots and Systems (IROS '95)*, pages 569–574, August 1995.

[14] M Bowling, A Ghodsi, and D Wilkinson. Action respecting embedding. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 65–72, 2005.

[15] T Briegel and V Tresp. Fisher scoring and a mixture of modes approach for approximate inference and learning in nonlinear state space models. In

M Kearns, S Solla, and D Cohn, editors, *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1999. MIT Press.

[16] J Brogdon, R Dunlap, P Dyson, A Martin de Nicolas, J Martin de Nicolas, J Martin de Nicolas, D McCauley, D Miner, J O'Quin, S Polkowski, and J Roever. Austin Robot Technology, Inc. Technical Paper, 2005. `www.darpa.mil/grandchallenge05/TechPapers/Austin_Robot_Technology.pdf`, DARPA Grand Challenge, 2005.

[17] J Burlet, O Aycar, and T Fraichard. Robust navigation using Markov models. In *International Conference on Intelligent Robots and Systems*, August 2005.

[18] V Bychkovskiy, S Megerian, D Estrin, and M Potkonjak. A collaborative approach to in-place sensor calibration. In *Information Processing in Sensor Networks*, pages 301–316. Springer, 2003.

[19] A Censi, L Marchionni, and G Oriolo. Simultaneous maximum-likelihood calibration of robot and sensor parameters. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2008.

[20] V Cevher and J McClellan. Sensor array calibration via tracking with the extended Kalman filter. In *Proceedings of the Fifth Annual Federated Laboratory Symposium on Advanced Sensors*, pages 51–56, March 2001.

[21] D Chetverikov, D Stepanov, and P Krsek. Robust Euclidean alignment of 3d point sets: the trimmed iterative closest point algorithm. *Im. and Vis. Comp.*, 23(3):299.

[22] J T Connor, R D Martin, and L E Atlas. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5(2):240–254, 1994.

[23] H Cox. On the estimation of state variables and parameters for noisy dynamic systems. *IEEE Transactions on Auromatic Control*, 9:5–12, 1964.

[24] P Dangauthier, P Bessiere, and A Spalanzani. Auto-supervised learning in the Bayesian programming framework. In *The AAAI Spring Symposium on Developmental Robotics*, March 2005.

[25] J de Freitas, M Niranjan, and A Gee. Nonlinear state space estimation with neural networks and the EM algorithm. Technical report, Cambridge University Engineering Department, 1999.

[26] A P Dempster, N M Laird, and D B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[27] V Digalakis, J Rohlicek, and M Ostendorf. ML estimation of a stochastic linear sytem with the EM algorithm and its application to speech recognition. *IEEE Transactions on Speech and Audio Processing*, 1(4), October 1993.

[28] Uwe Duffert and Jan Hoffmann. Reliable and precise gait modeling for a quadruped robot. In *RoboCup Symposium*, 2005.

[29] A Eliazar and R Parr. Learning probabilistic motion models for mobile robots. In *Internation Conference on Machine Learning (ICML-04)*, 2004.

[30] M Fox, M Ghallab, G Infantes, and D Long. Robot introspection through learned Hidden Markov Models. *Artificial Intelligence*, 170(2):59–113, 2006.

[31] D Fraser and J Potter. The optimum linear smoother as a combination of two optimum linear filters. *IEEE Transactions on Automatic Control*, 14(4):387–390, August 1969.

[32] J Friedman, J Bentley, and R Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. on Mathematical Software*, 3(3):209–226, 1977.

[33] Z Ghahramani and G Hinton. Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, University of Toronto Department of Computer Science, 1996.

[34] Z Ghahramani and S Roweis. Learning nonlinear dynamical systems using an EM algorithm. In M Kearns, S Solla, and D Cohn, editors, *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1999. MIT Press.

[35] J Gutmann and C Schlegel. AMOS: Comparison of scan matching approaches for self-localization in indoor environments. In *Proc. of the 1st Euromicro Workshop on Advanced Mobile Robots*. IEEE Computer Society Press, 1996.

[36] D Hahnel, W Burgard, D Fox, K Fishkin, and M Philipose. Mapping and localization with RFID technology. In *International Conference on Robotics and Automation (ICRA-04)*, 2004.

[37] R Hanson and M Norris. Analysis of measurements based on the singular value decimposition. *SIAM Journal of Scientific and Statistical Computing*, 27(3):363–373, 1981.

[38] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer-Verlag, New York, 2001.

[39] Simon Haykin. *Neural Networks*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1999.

[40] B Hengst, D Ibbotson, S Pham, and C Sammut. Omnidirectional motion for quadruped robots. In *RoboCup International Symposium*, August 2001.

[41] J Hines, D Wrest, and R Uhrig. Plant wide sensor calibration monitoring. In *IEEE Symposium on Intelligent Control*, September 1996.

[42] Alexander T. Ihler, John W. Fisher, Randolph L. Moses, and Alan S. Willsky. Nonparametric belief propagation for self-calibration in sensor networks. In *Proceedings of the third international symposium on Information processing in sensor networks*, Berkeley, CA, April 2004.

[43] A Kaboli, M Bowling, and P Musilek. Bayesian calibration for Monte Carlo localization. In *Twenty-First National Conference on Artificial Intelligence (AAAI)*, pages 964–969, 2006.

[44] Rudolph Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82:35–45, 1960.

[45] S Koenig and R Simmons. Unsupervised learning of probabilistic models for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1996.

[46] B Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119:191–233, 2000.

[47] T. D. Larsen, M. Bak, N.A. Andersen, and O. Ravn. Location estimation for an autonomously guided vehicle using an augmented Kalman filter to auto-calibrate the odometry. In *FUSION98 Spie Conference*, Las Vegas, NV, July 1998.

[48] S Levinson, K Squire, R Lin, and M McClain. Automatic language acquisition by an autonomous robot. In *The AAAI Spring Symposium on Developmental Robotics*, March 2005.

[49] M Littman, R Sutton, and S Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems 14*, pages 1555–1561, 2002.

[50] L Ljung. Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems. *IEEE Transactions on Automatic Control*, 24:36–50, 1979.

[51] F Lu and E Milios. Robot pose estimation in unknown environments by matching 2d range scans. In *IEEE Computer Vision and Pattern Recognition Conference (CVPR)*, 1994.

[52] M Lungarella, G Metta, R Pfeifer, and G Sandini. Developmental robotics: a survey. *Connection Science*, 15(4):151–190, December 2003.

[53] Agostino Martinelli, Nicola Tomatis, Adriana Tapus, and Roland Siegwart. Simultaneous localization and odometry calibration for mobile robot. In *Proceedings of the 2003 International Confrerence on Intelligent Robots and Systems*, Las Vegas, NV, October 2003.

[54] J Modayil and B Kuipers. Bootstrap learning for object discovery. In *IEEE International Conference on Intelligent Robots and Systems (IROS-04)*, pages 742–747, 2004.

[55] M Montemerlo, S Thrun, H Dahlkamp, D Stavens, and S Strohband. Winning the DARPA Grand Challenge with an AI robot. In *Proceedings on the AAAI National Conference on Artificial Intelligence*, 2006.

[56] H Moravec and M Blackwell. Learning sensor models for evidence grids. *CMU Robotics Institure 1991 Annual Research Review*, pages 8–15, 1993.

[57] L Natale, G Metta, and G Sandini. A developmental approach to grasping. In *The AAAI Spring Symposium on Developmental Robotics*, March 2005.

[58] L W Nelson and E Stear. The simultaneous on-line estimation of parameters and states in linear systems. *IEEE Transactions on Automatic Control*, AC-12:438–442, 1967.

[59] L Ojeda, D Cruz, G Reina, and J Borenstein. Current-based slippage detection and odometry correction for mobile robots and planetary rovers. *IEEE Transactions on Robotics*, 22(2), April 2006.

[60] L Olsson, C Nehaniv, and D Polani. From unknown sensors and actuators to actions grounded in sensorimotor perceptions. *Connection Science*, 18(2), 2006.

[61] P Oudeyer, F Kaplan, and V Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

[62] P Oudeyer, F Kaplan, V Hafner, and A Whyte. The playground environment: task-independent development of a curious robot. In *The AAAI Spring Symposium on Developmental Robotics*, March 2005.

[63] U Ozguner, K A Redmill, and A Broggi. Team TerraMax and the DARPA grand challenge: a general overview. In *Intelligent Vehicles Symposium*, pages 232–237, June 2004.

[64] D Philipona, J O'Regan, and J.-P. Nadal. Is there something out there? Inferring space from sensorimotor dependencies. *Neural Computation*, 15(9), 2003.

[65] J M Phillips, R Liu, and C Tomasi. Outlier robust ICP for minimizing fractional RMSD. In *6th International Conference on 3-D Digital Imaging and Modeling*, pages 427–434, August 2007.

[66] David Pierce and Benjamin Kuipers. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, 92:169–229, 1997.

[67] J Provost, B Kuipers, and R Miikkulainen. Developing navigation behavior through self-organizing distinctive-state abstraction. *Connection Science*, 18(2):159–172, 2006.

[68] M Quinlan, C Murch, T Moore, R Middleton, L Li, R King, and S Chalup. The 2004 NUbots team report, 2004. `http://robots.newcastle.edu.au/publications/NUbotFinalReport2004.pdf`.

[69] L Rabiner and B Juang. An introduction to hidden markov models. *ASSP Magazine*, 3(1):4–16, 1986.

[70] I Reid. Projective calibration of a laser-stripe range finder. *Image and Vision Computing*, 14:659–666, 1996.

[71] Nicholas Roy and Sebastian Thrun. Online self-calibration for mobile robots. In *Proceeding of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2292–2297, Detroit, MI, May 1999. IEEE Computer Society Press.

[72] A Savvides, C C Han, and M B Strivastava. Dynamic fine-grained localization in ad-hoc wireless sensor networks. In *Proceedings of the International Conference on Mobile Computing and Networking*, July 2001.

[73] M Sayers. Vehicle models for RTS applications. *Vehicle System Dynamics*, 32(4–5):421–438, 1999.

[74] M Sayers and D Han. A generic multibody vehicle model for simulating handling and braking. *Vehicle System Dynamics*, 25:599–613, 1996.

[75] B Schiele and J Crowley. A comparison of position estimation techniques using occupancy grids. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 128–134, 1994.

[76] M Schlesinger. Decomposing infants' object representations: a dual-route processing account. 18(2):207–216, June 2006.

[77] Gideon Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.

[78] R H Shumway and D S Stoffer. An approach to time series smoothing and forecasting using the EM algorithm. *Journal of time series analysis*, 3:253–264, 1982.

[79] R Simmons, S Thrun, C Athanassiou, J Cheng, L Chrisman, R Goodwin, G Hsu, and H Wan. ODYSSEUS: an autonomous mobile robot. *AI Magazine*, 1992.

[80] O Simsek and A Barto. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML '06)*, June 2006.

[81] S Singh, A Barto, and N Chentanez. Intrinsically motivated reinforcement learning. In *18th Annual Conference on Neural Information Processing Systems (NIPS)*, December 2004.

[82] Peter Stone, Tucker Balch, and Gerhard Kraetzschmar, editors. *RoboCup-2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, 2001.

[83] Peter Stone, Patrick Beeson, Tekin Mericli, and Ryan Madigan. DARPA urban challenge technical report: Austin Robot Technology, June 2007. Available from `http://www.darpa.mil/grandchallenge/rules.asp`.

[84] Peter Stone, Kurt Dresner, Selim T. Erdoğan, Peggy Fidelman, Nicholas K. Jong, Nate Kohl, Gregory Kuhlmann, Ellie Lin, Mohan Sridharan, Daniel

Stronger, and Gurushyam Hariharan. The UT Austin Villa 2003 four-legged team. In Daniel Polani, Brett Browning, Andrea Bonarini, and Kazuo Yoshida, editors, *RoboCup-2003: Robot Soccer World Cup VII*. Springer Verlag, Berlin, 2004.

[85] Peter Stone, Kurt Dresner, Peggy Fidelman, Nicholas K. Jong, Nate Kohl, Gregory Kuhlmann, Mohan Sridharan, and Daniel Stronger. The UT Austin Villa 2004 RoboCup four-legged team: Coming of age. Technical Report UT-AI-TR-04-313, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, October 2004.

[86] Peter Stone, Kurt Dresner, Peggy Fidelman, Nate Kohl, Gregory Kuhlmann, Mohan Sridharan, and Daniel Stronger. The UT Austin Villa 2005 RoboCup four-legged team. Technical Report UT-AI-TR-05-325, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, November 2005.

[87] Peter Stone, Peggy Fidelman, Nate Kohl, Gregory Kuhlmann, Tekin Mericli, Mohan Sridharan, and Shao en Yu. The UT Austin Villa 2006 RoboCup four-legged team. Technical Report UT-AI-TR-06-337, The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, December 2006.

[88] A Stoytchev. Learning the affordances of tools using a behavior-grounded approach. In E Rome, J Hertzberg, and G Dorffner, editors, *Affordance-Based Robot Control*, pages 140–158. Springer Lecture Notes in Artificial Intelligence, 2008.

[89] G. Strang. *Linear Algebra and its Applications*. Brooks Cole, 1998.

[90] Daniel Stronger and Peter Stone. Towards autonomous sensor and actuator

model induction on a mobile robot. *Connection Science*, 18(2):97–119, 2006. Special Issue on Developmental Robotics.

[91] Daniel Stronger and Peter Stone. Maximum likelihood estimation of sensor and action model functions on a mobile robot. In *IEEE International Conference on Robotics and Automation*, May 2008.

[92] Daniel Stronger and Peter Stone. Polynomial regression with automated degree: A function approximator for autonomous agents. *International Journal on Artificial Intelligence Tools*, 17(1):159–174, February 2008.

[93] R Sutton and A Barto. *Reinforcement learning: an introduction*. MIT Press, 1998.

[94] S Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99:21–71, 1998.

[95] S Thrun, W Burgard, and D Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31(1–3):29–53, 1998.

[96] S Thrun, W Burgard, and D Fox. *Probabilistic Robotics*. MIT Press, 2005.

[97] R Tsai. An efficient and accurate camera calibration technique for 3d machine vision. In *IEEE CVPR 1986*, 1986.

[98] Sanford Weisberg. *Applied Linear Regression*. John Wiley & Sons, Inc., New York, 1980.

[99] Greg Welch and Gary Bishop. An introduction to the Kalman filter. Technical Report 95-041, University of North Carolina at Chapel Hill, Department of Computer Science, 2004.

[100] J Weng, J McClelland, A Pentland, O Sporns, I Stockman, M Sur, and E Thelen. Autonomous mental development by robots and animals. *Science*, 291:599–600, 2001.

[101] Shimon Whiteson, Peter Stone, Kenneth O. Stanley, Risto Miikkulainen, and Nate Kohl. Automatic feature selection via neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, June 2005.

[102] W Whittaker and L Nastro. Utilization of position and orientation data for pre-planning and real time autonomous vehicle navigation. In *Proceedings of IEEE/ION Position Location and Navigation Symposium*, 2006.

[103] T Yap and C Shelton. Simultaneous learning of motion and sensor model parameters for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2008.

[104] Q Zhang and R Pless. Extrinsic calibration of a camera and laser range finder. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2004.

[105] Z Zhang. Iterative point matching for registration of free-form curves. *International Journal of Computer Vision*, 13(2):119–152, 1994.

Vita

Daniel Stronger began his doctoral work in the Department of Computer Sciences at the University of Texas at Austin in August 2001. As an undergraduate, he attended Harvard University from 1997 to 2001, receiving a Bachelor of Arts in Physics *cum laude*. Growing up in New York City, Daniel attended Stuyvesant High School and participated in mathematics competitions, where he achieved second place in the USA Mathematical Olympiad. He subsequently represented the U.S. team in the International Mathematical Olympiad in 1997, where he was awarded a Silver Medal.

Permanent Email Address: stronger@post.harvard.edu

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab.