In *IEEE International Conference on Robotics and Automation* (ICRA 05),
Barcelona, April 2005.

Simultaneous Calibration of Action and Sensor Models on a Mobile Robot

Daniel Stronger and Peter Stone
Department of Computer Sciences, The University of Texas at Austin
1 University Station C0500, Austin, Texas 78712-1188
{stronger,pstone}@cs.utexas.edu
http://www.cs.utexas.edu/~{stronger,pstone}

***Abstract*— This paper presents a technique for the Simultaneous Calibration of Action and Sensor Models (SCASM) on a mobile robot. While previous approaches to calibration make use of an independent source of feedback, SCASM is unsupervised, in that it does not receive any well-calibrated feedback about its location. Starting with only an inaccurate action model, it learns accurate relative action and sensor models. Furthermore, SCASM is fully autonomous, in that it operates with no human supervision. SCASM is fully implemented and tested on a Sony Aibo ERS-7 robot.**

I. Introduction

Mobile robots rely on models of their actions and sensors in order to interact with their environment. For example, they may select actions based on their *anticipated* effects and deduce their *actual* effects based on their subsequent sensations. These sensor and action models are typically calibrated manually, a fairly laborious and often brittle process. Furthermore, if the robot is placed in a novel environment and/or its sensors and actuators change over time, the pre-calibrated models can quickly become error-prone. Thus it is desirable for this calibration process to be automated, so that the robot can learn the effects of its actions and the meanings of its observations without human supervision.

This paper presents a technique, called SCASM,[1] for a mobile robot to simultaneously learn two calibration functions. One maps the various readings of a visual sensor to relative distances from a fixed landmark, and the other maps a range of action commands to the velocities of the corresponding movements. SCASM is completely autonomous, and it is unsupervised, in that the robot never receives any feedback as to its actual location or velocity. SCASM's goal is for the robot to learn action and sensor models that accurately reflect its distances and velocities.

SCASM involves the robot performing the following three tasks simultaneously.

- Walking forwards and backwards while its visual sensor is faced at a fixed target, covering the entire range of relevant distances and velocities.
- Learning a function from action commands to actual velocities, assuming the distance calibration for the visual sensor is accurate.
- Learning a function from distance observation data to its distances from the target, assuming the robot has an accurate sense of its velocities.

This process successfully learns action and sensor models that closely approximate measurements made manually with a stopwatch and a tape measure.

II. Experimental setup

SCASM is implemented and tested on a commercially available robot platform, namely the Sony Aibo ERS-7.[2] The results reported in this paper make use of walking and vision processing modules that we created earlier as part of a larger project [1]. The walk is defined by a number of parameters that specify the attempted trajectories of the Aibo's feet. To move forwards and backwards at different speeds, the robot interpolates between parameters for an idle walk $\mathbf{p_i}$ that steps in place, a fast forwards walk $\mathbf{p_f}$ that goes at a speed of v_{max} (335 mm/s), and a fast backwards walk $\mathbf{p_b}$ that goes at a speed of v_{min} (-280 mm/s). The action commands are labeled by a desired velocity r and have parameters given by:

$$\mathbf{p_r} = \begin{cases} \mathbf{p_i} + \frac{r}{v_{max}}(\mathbf{p_f} - \mathbf{p_i}) & \text{if } r \geq 0 \\ \mathbf{p_i} + \frac{r}{v_{min}}(\mathbf{p_b} - \mathbf{p_i}) & \text{if } r < 0 \end{cases} \quad (1)$$

Note that Equation (1) is based on the assumption that the Aibo's velocity is linear in its walking parameters. However, it turns out not to be linear at all, as the experimental results will bear out. The robot's actual velocity varies unpredictably with respect to the desired velocity. SCASM learns the function from the robot's action commands to their corresponding velocities.

The Aibo's visual sensor is based on its camera, which we have previously trained to recognize objects in the robot's environment. One of these objects is a colored cylindrical beacon that the robot can use to help it localize while on a playing field. The height of the beacon in the robot's image plane decreases with the robot's distance from the beacon; this observed height (in pixels) is the visual sensor reading used for the experiments reported in this paper. The Aibo and the beacon are shown in Figure 1, along with a view of the beacon taken through the Aibo's camera. Detailed descriptions of the walking and vision modules used here are given in [1].

While the robot learns the action and sensor models, its behavior must allow it to experience the full range of relevant action commands and observations. To achieve this goal, the Aibo walks alternatingly forwards and backwards across a preset range of distances from the beacon. For the experiments reported in this paper, the robot's goal is to learn about the action commands in the range $[-300, 300]$. Hence, the

[1]Simultaneous Calibration of Action and Sensor Models

[2]http://www.aibo.com

robot chooses a random action command in the range $[0, 300]$ while going forwards and from $[-300, 0]$ during the backwards phase. It continues to execute each action for three seconds before choosing a new one. It switches between walking forwards and backwards 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 relevant distances and velocities, as desired.

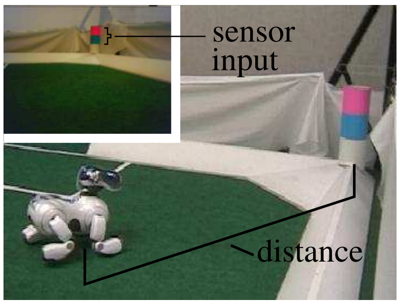


Fig. 1. The Aibo and the beacon. The inset is a picture of the beacon taken through the Aibo's camera.

Although the action commands being executed only attempt to move forwards and backwards, random drift would cause the Aibo to slowly get off course. To counteract this effect, the walking controller is set to constantly turn towards 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 forwards or backwards velocity. A video of the Aibo performing the training behavior described here is available online.[3]

III. Action and sensor models

As the robot moves towards and away from the beacon, we denote its (actual) distance from the beacon at time t as $x(t)$. During this process, the robot has two sources of information about its location along its axis of movement. For one, the robot receives a sequence of visual sensor observations, the kth one denoted by obs_k and occurring at time t_k. Each value reported by the visual sensor corresponds to a specific distance. We call this function the *sensor model*, and denote it by S, so that $x(t_k) = S(obs_k) + \omega_k$, where ω_k is a zero-mean random offset due to the inherent noise in the visual sensor. This function S is one of the two functions that the robot is trying to learn.

At the same time, the robot continuously executes an action command, $C(t)$, that varies with time. Each action command moves the robot at a specific velocity, and we denote the function from command to velocity by A. This function is the *action model* that the robot learns along with the sensor model S. The action model also provides information about the robot's location: $x(t) = x(0) + \int_0^t A(C(s))\, ds$. SCASM works by implicitly performing a continual comparison of these two sources of information. The robot knows the values of obs_k, t_k, and $C(t)$, and its task is to learn the functions A and S.

[3]http://www.cs.utexas.edu/~AustinVilla/?p=research/simultaneous_calibration

Because the robot is trying to learn two arbitrary continuous functions, it must represent them with a function approximator. Polynomial regression is used for both functions. That is, for the sensor model, the robot's goal is to learn coefficients s_0 through s_d such that the polynomial $\sum_{i=0}^{d} s_i obs^i$ approximates the actual $S(obs)$ as closely as possible, where d is the degree of the polynomial being fitted to the data. Similarly, the robot will learn coefficients a_0 through a_d for the action model, with the goal of $\sum_{i=0}^{d} a_i c^i \approx A(c)$ over the range of commands c. 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.

SCASM learns the action and sensor models *from each other* in that it is not given any ground truth as to the robot's distance to the beacon or its speed. Therefore, it cannot learn the two models in any particular units. For example, the sensor model maps observations onto points on a linear axis, but it makes no claims as to what physical distance from the beacon corresponds to the number zero, or what length corresponds to the model's units. Similarly, the action model is learned in arbitrarily units, although here the number zero is constrained to correspond to a speed of zero. However, the learned action and sensor models will be consistent with each other. That is, since the robot knows how long a second is, whatever distance turns out to be the unit for the learned distances, that distance per second is the unit for the learned velocities. Note that this property is sufficient for it to perform domain-centric tasks, such as predicting the amount of time a specific action command will take to yield a certain visual sensor reading.

IV. Learning the sensor model

First we demonstrate that it is possible to learn a relative sensor model given any constant action. Note that while the robot executes a constant action command, c, it is moving at a constant velocity, $A(c)$. Thus if this command is executed continuously starting at time 0, the robot's location at time t will be given by $x(t) = x(0) + t \cdot A(c)$. Thus there is enough information to learn the sensor model, even in the absence of knowledge of the value of $A(c)$.

In particular, in this situation it suffices to learn a function from obs_k to t_k. If $\hat{S}(obs_k) = t_k$, then since $t_k = (x(t_k) - x(0))/A(c)$, $\hat{S}(obs_k) = (x(t_k) - x(0))/A(c)$. This expression represents a shifted and scaled version of the robot's location, and since SCASM is only trying to learn a sensor model up to shifting and scaling, $\hat{S}$ is a satisfactory sensor model. The robot learns the function by performing polynomial regression on the pairs (obs_k, t_k).

Given n data points (obs_k, t_k), SCASM computes the coefficients of the best fit d-degree polynomial, $P(obs) = \alpha + \sum_{i=1}^{d} \beta_i obs^i$, which minimizes the total squared error between $P(obs_k)$ and the corresponding t_k over all k from 1 to n. To compute α and $\beta = (\beta_1, \cdots, \beta_d)^\top$, we reformulate the problem as a multivariable linear regression by representing each of the powers of obs with its own variable $V_j = obs^j$. The input data is then an $n \times d$ matrix V given by $V_{i,j} = obs_i^j$

and we denote the n-dimensional output vector by y where $y_i = t_i$. To perform this regression, first we define M and Y to be versions of V and y where the variables are centered around zero. That is, their means are subtracted from their values: $M_{i,j} = V_{i,j} - \overline{V_j}$ and $Y_i = y_i - \overline{y}$. Then α and β are given by [2], [3]

$$\beta = (M^\top M)^{-1}(M^\top Y) \qquad \text{and} \qquad \alpha = \frac{1}{n}\sum_{i=1}^{n}(y - V\beta)_i \tag{2}$$

Fortunately, it is not necessary to store V and y explicitly and compute these quantities from them each time. To save space and time as arbitrarily many data points come in, the robot incrementally maintains a number of sums that require constant storage space in the number of data points. For example, $(M^\top M)_{i,j}$ evaluates to

$$\sum_k V_{k,i}V_{k,j} - \frac{1}{n}(\sum_k V_{k,i})(\sum_k V_{k,j}) \tag{3}$$

The robot maintains these sums (for all i and j from 1 to d) incrementally, along with $\sum_k V_{k,i}y_k$ and $\sum_k y_k$. They enable it to compute $M^\top M$ as above, $M^\top Y$, and then β and α.

When this process is applied to the pairs (obs_k, t_k) with $d = 3$ while a constant action command is being executed, the cubic learned is typically quite an accurate fit to the data, as shown in Figure 2a).

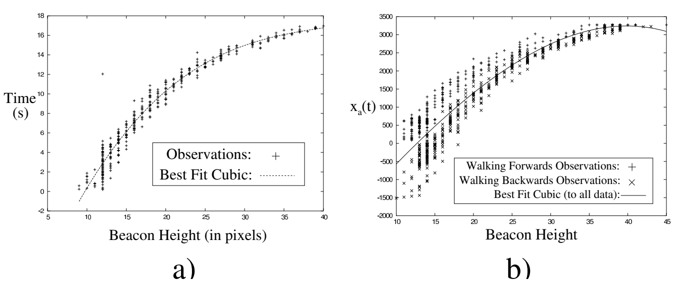


Fig. 2. a) After walking forwards via a constant action, these are the observed data points (+), mapped against time. The dashed curve is the best fit cubic to these points. The variation in beacon height at any given time is due to inherent noise in vision. b) The plotted points are $(obs_k, x_a(t_k))$ as the robot performs one full cycle of walking towards the beacon and backing away from it. The $+$'s are the observations while walking forwards and the $\times$'s are while walking backwards. The polynomial is fitted to all the points.

It is also desirable for the robot to be able to learn a sensor model while it performs a series of various actions, such as in the randomized behavior described in Section II. This process relies on the robot having access to an accurate action model. Although it does not have one initially, Section VI shows how this ability can be incorporated into a process that can learn both models from scratch.

Given an action model A, the robot can use dead reckoning to compute its location as a function of time. As mentioned in Section III, the robot's location $x(t)$ is given by $x(0) + \int_0^t A(C(s))\,ds$, which we denote by $x_a(t)$. It suffices to assume that $x(0) = 0$, since it is learning relative distances. Thus the robot can accumulate an estimate for $x(t)$ by initializing x to be 0 at time 0 and continually incrementing it by $A(C(t))\Delta t$, where Δt is the amount of time between increments. Then, by performing cubic regression on the pairs $(obs_k, x_a(t_k))$, the robot effectively learns a sensor model from the action model. The result of such a regression is shown in Figure 2b). Note that because the action model used here is inaccurate, the estimates taken while walking forwards and backwards are not well aligned with each other. 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.

V. Learning the Action Model

In this section, we assume that the robot has an accurate sensor model and show how the robot can use it to learn an action model. This learning uses the sensor model to give the robot an estimate of its location from each observation. We denote this estimate by $x_s(t_k)$, and it is given by $S(obs_k)$. The robot's goal is to learn the function $A(c) = \sum_{i=0}^{d} a_i c^i$ that causes the values of $x(t_k)$ based on A to match those based on S as closely as possible. That is, the robot computes the coefficients a_i that minimize the error defined by

$$\begin{aligned} E &= \sum_{k=1}^{n}\left[x_s(t_k) - \left(x(0) + \int_0^{t_k}\sum_{i=0}^{d} a_i C(s)^i\,ds\right)\right]^2 \\ &= \sum_{k=1}^{n}\left[x_s(t_k) - \left(x(0) + \sum_{i=0}^{d} a_i \int_0^{t_k} C(s)^i\,ds\right)\right]^2, \end{aligned} \tag{4}$$

where the robot knows the values obs_k, $x_s(t_k)$, and the values of $C(s)$. This problem is an instance of a multivariable linear regression, with $d+1$ variables V_1 through V_{d+1} defined as $V_j = \int_0^t C(s)^{j-1}\,ds$ and output $y_k = x_s(t_k)$. The regression computes the weights a_i (and a value for $x(0)$) that minimize the error. Since $C(s)$ changes every three seconds, the value for $x(t)$ suggested by an estimate for $x(0)$ and the weights a_i varies in a piecewise linear manner with respect to time. The regression being performed has the effect of finding the piecewise linear curve that fits the data $(t_k, x_s(t_k))$ as closely as possible (as shown in Figure 3), provided that the slope of the line at any time t is a constant quartic function of $(C(t))$.

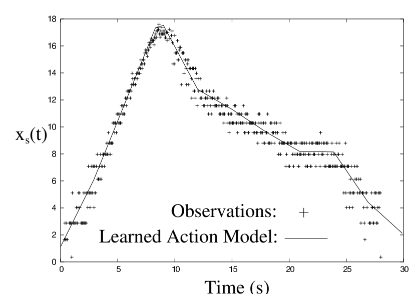


Fig. 3. The plotted points are $(t_k, x_s(t_k))$ as the robot performs one full cycle of walking towards the beacon and backing away from it. The learned action model is applied to the executed action commands to yield the piecewise linear location estimate shown here.

To learn an action model, the robot first learns a rough sensor model using the constant action method at the top of Section IV. Then it uses that sensor model to execute the process described in this section. The result of this process is shown in Figure 3.

VI. Learning both simultaneously

We have so far demonstrated the ability for the robot to learn the sensor model from the action model and vice versa. Making use of both of these capabilities, this section shows how the robot can simultaneously learn both models, even when it is given very little useful starting information. This learning is possible because, even though the action (sensor) model learned from an inaccurate sensor (action) model will be inaccurate, it will 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 visual sensor observation. Thus the weight of each data point starts at one and decreases 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, we define W as an $n \times n$ diagonal matrix with $W_{i,i} = \gamma^{n-i}$ and N as the sum of the weights $\sum_{i=1}^{n} \gamma^{n-i}$. Then we can use a weighted version of Equation (2) [3]:

$$\beta = (M^\top W M)^{-1}(M^\top W Y) \quad \text{and} \quad \alpha = \frac{1}{N}\sum_{i=1}^{n} \gamma^{n-i}(y - V\beta)_i \tag{5}$$

As before, these quantities are expressed in terms of sums. For example $(M^\top W M)_{i,j}$ is given by

$$\sum_k \gamma^{n-k} V_{k,i} V_{k,j} - \frac{1}{N}(\sum_k \gamma^{n-k} V_{k,i})(\sum_k \gamma^{n-k} V_{k,j}) \tag{6}$$

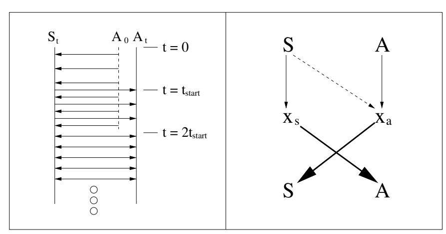
These sums can also be maintained incrementally, because $\sum_{k=1}^{n+1} \gamma^{(n+1)-k} z_k = \gamma(\sum_{k=1}^{n} \gamma^{n-k} z_k) + z_{n+1}$ for any sequence z_k.

Pseudocode for the entire algorithm is given in Figure 4. At time t, the robot makes use of its best estimates thus far as to the action and sensor models, A_t and S_t. Throughout the learning, the robot maintains two estimates of its location, one based on its current sensor model, $x_s(t)$, and the other based primarily on its action model, $x_a(t)$. After any observation obs_k at time t_k, $x_s(t_k)$ is given by $S_t(obs_k)$. At the same time, $x_a(t)$ is maintained by continually incrementing it by $A_t(C(t))\Delta t$, where Δt is the amount of time between increments and $A_t(C(t))$ is the robot's current estimate of its velocity. Unfortunately, it is not sufficient for $x_a(t)$'s derivative to be an accurate estimate of the robot's velocity. This constraint still allows for the possibility that $x_a(t)$ is a constant displacement away from $x_s(t)$. When this approach was tried on the robot, it occasionally happened that the estimates diverged, both increasing continually, in which case neither model can be learned accurately. To prevent this problem, $x_a(t)$ is adjusted towards $x_s(t)$ every time an observation is taken. The adjustment is implemented by the assignment $x_a(t_k) \leftarrow (1-\lambda)x_a(t_k) + \lambda x_s(t_k)$, where λ is a constant that determines the strength with which $x_a(t)$ is pulled towards $x_s(t)$.

The model estimates S_t and A_t are continually updated in accordance with the location estimates $x_a(t)$ and $x_s(t)$, with each model being updated by the location estimate based on the other model. These updates consist of the incremental updates that comprise the weighted polynomial regressions that give the best fit estimates of S and A, as described above. The flow of information is depicted in Figure 5b). Note that because the regressions can be computed incrementally, they can be calculated every time the robot processes an image, corresponding to about 20 Hertz. This process happens 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.

$x_a(t) \leftarrow 0$
for each time step **do**
  if $t < 2t_{start}$ **then**
    $x_a(t) \leftarrow x_a(t) + A_0(C(t))\Delta t$
  else
    $x_a(t) \leftarrow x_a(t) + A_t(C(t))\Delta t$
  end if
  if an observation obs_k is made **then**
    if $t > t_{start}$ **then**
      $x_s(t) \leftarrow S_t(obs_k)$
      UPDATE A_t with $(t, x_s(t))$
      $x_a(t) \leftarrow (1-\lambda)x_a(t) + \lambda x_s(t)$
    end if
    UPDATE S_t with $(t, x_a(t))$
  end if
end for

Fig. 4. Algorithm for simultaneous action and sensor model learning. The routine UPDATE incorporates one new data point into the weighted regression for the model being updated.



a) b)

Fig. 5. a) 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. b) 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 robot's location. The dashed arrow signifies S's influence on the estimate x_a.

At the start of the training, there is no data to motivate either the action model or the sensor model to get the learning process started. For a period of time at the beginning, t_{start}, the robot uses a fixed, pre-set action model, A_0, instead of A_t. The function used for A_0 is the identity function, so that $A_0(c) = c$. During this time, the sensor model is learned based on A_0, but the action model is not being learned yet, because the sensor model is based on too few data points. After time t_{start} has passed, the sensor model can be used to start learning an action model. However, until another period of time length t_{start} has passed, this new action model is not based on enough data points to be used for learning. From time $2t_{start}$ into the learning on, the action and sensor models can learn from each other. This process is depicted in Figure 5a).

Figure 6 depicts how $x_s(t)$ and $x_a(t)$ vary over time when S and A are being learned simultaneously. Note that both oscillate with the robot's walking towards and away from the beacon. As A and S grow more accurate, their corresponding estimates of the location come into stronger agreement.

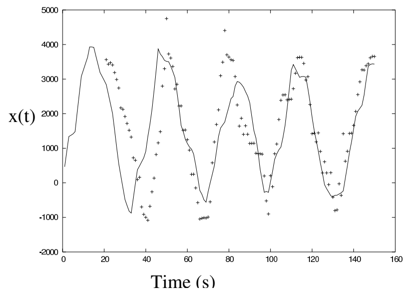
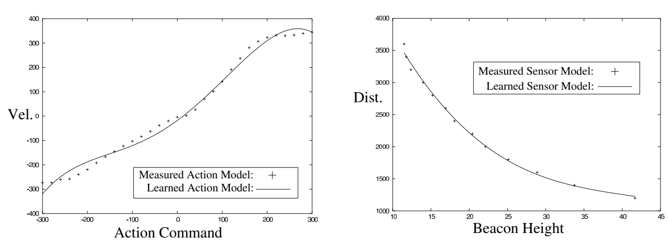


Fig. 6. This figure shows how $x_a(t)$, and $x_s(t)$ vary over time. In this example run, the $+$'s are values of $x_s(t)$, and the curve depicts $x_a(t)$. Over time, each model learns how to keep its estimate of the location close to the other model's estimate.

The algorithm described above makes use of a few constants that did not require any extensive tuning. The discount factor for the regression weights, γ, is 0.999. The strength of the pull of x_a towards x_s, λ, is $1/30$. These values were the first ones that were tried for γ and λ. The starting phase time, t_{start}, is 20 seconds. We tried 10 seconds first but that was too short.

VII. Experimental results

After SCASM has run for a pre-set amount of time (two and a half minutes), we consider its best estimates for A and S to be the models that it has learned at that point. The success of SCASM is evaluated by comparing the learned action and sensor models to those measured with a stopwatch and a tape measure. The measured action model is obtained by measuring the velocity of each action command that is a multiple of 20 from -300 to 300. We measure the velocity of an action command by timing it across an appropriate distance five times. 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 7a).



a) b)

Fig. 7. A learned action and sensor model

Similarly, the accuracy of the learned sensor model is gauged by comparing it to a measured sensor model. The sensor model is 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 7b).

The learning process was executed 15 times, with each trial lasting for two and a half minutes. Figure 7a) shows a typical learned action model, compared to 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 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, the 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 7b) shows a typical learned sensor model with the measured sensor model. The learned model S maps observations to relative distances, $S(obs)$, which are intended to model the actual distances from the beacon. These actual distances are given by $a + bS(obs)$, 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 $a + bS(obs)$ and the measured sensor model. This minimization is done with a linear regression on the points $(S(obs_i), S_m(obs_i))$, where the obs_i are the sensor readings corresponding to the measured distances $S_m(obs_i)$. 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, the 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 8. 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.

Although the action and sensor models are not learned to any particular scale, since they are learned from each other they should be to the same scale. This property is 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. We evaluate the degree of equality by computing the average distance between the absolute value of the ratio between the two scaling constants and 1. The average distance is 0.08 ± 0.06. This result shows that the two learned models are consistent with each other.

The amount of time taken by SCASM to accurately learn its action and sensor models is two and a half minutes. Note that for a fixed environment, SCASM only needs to be executed once. Given that each time the robot is booted up, it takes about 27 seconds to initialize, we consider SCASM's one-time execution time to be qualitatively quite short. Certainly, it is within the bounds of what can be reasonably executed on-line upon insertion into a new environment.

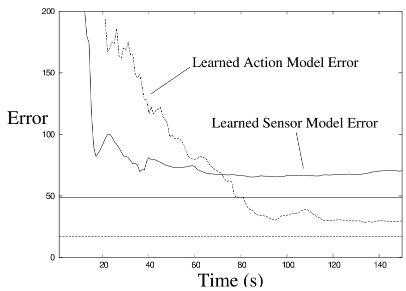


Fig. 8. This figure depicts the average error in the learned models as a function of time. The error for the action model is in mm/s, and for the sensor model in mm. The horizontal lines are at the minimum possible error to the measured models for a polynomial of the appropriate degree.

The results described above start from a linear action model that is somewhat similar to the measured action model (shown in Figure 7). To examine the reliance of our approach on the starting action model, we performed two tests with more impoverished starting points. First, we used a piecewise constant model equal to 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 was able to achieve 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. Even with a starting model of $A(x) = 1$, 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 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. Note that the errors achieved with these more impoverished models 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), indicating that these results are not particularly sensitive to the starting action model.

VIII. Related work

Some previous work has focused on mobile robots calibrating their odometry models automatically based on their sensors. For example, Roy and Thrun [4] calibrate the odometry on a wheeled robot using an incremental maximum likelihood method, while Martinelli et al. [5] and Larsen et al. [6] use an augmented Kalman Filter to estimate odometry errors. There has also been work on calibrating networks of sensors. However, this work (e.g., [7], [8]) typically focuses on networks with large numbers of sensors and calibrating their respective locations and orientations. We know of no previous work calibrating a sensor based on an action model. Furthermore, to the best of our knowledge, SCASM differs from all previous work along these lines in the following significant way. SCASM learns models of its actions and sensors starting without an accurate model of either. Previous approaches to calibration rely either on accurate training data or on sensors that are already well calibrated (as in [4]).

IX. Conclusion and future work

This paper presents a technique by which a mobile robot can learn an action model and a sensor model from each other simultaneously. Starting with only a very simplistic action model estimate, the robot learns highly accurate approximations to its true action and sensor models. The learning process is completely autonomous and unsupervised, so that no human oversight or feedback is necessary. The technique is successfully implemented on a Sony Aibo ERS-7, which calibrates its action commands to the resultant velocities and its visual sensor readings to the corresponding distances, all in two and a half minutes of autonomous behavior.

One direction for future work is to explore potential synergies between SCASM and particle filtering methods that integrate sensor and action models into a position estimate (e.g., [9]). The work presented here represents an exciting start towards the long-term challenge of enabling fully autonomous calibration of complex, multi-modal sensor and action models on mobile robots.

Acknowledgments

We would like to thank Ben Kuipers for helpful discussions. Thanks also to the members of the UT Austin Villa team for their efforts in developing the software used as a basis for the work reported in this paper. This research was supported in part by NSF CAREER award IIS-0237699, ONR YIP award N00014-04-1-0545, and DARPA grant HR0011-04-1-0035.

References

[1] P. Stone, K. Dresner, P. Fidelman, N. K. Jong, N. Kohl, G. Kuhlmann, M. Sridharan, and D. Stronger, "The UT Austin Villa 2004 RoboCup four-legged team: Coming of age," The University of Texas at Austin, Department of Computer Sciences, AI Laboratory, Tech. Rep. UT-AI-TR-04-313, October 2004.

[2] R. F. Gunst and R. L. Mason, *Regression Analysis and its Application*. New York: Marcel Dekker, Inc., 1980.

[3] S. Weisberg, *Applied Linear Regression*. New York: John Wiley & Sons, Inc., 1980.

[4] N. Roy and S. Thrun, "Online self-calibration for mobile robots," in *Proceeding of the IEEE International Conference on Robotics and Automation*, vol. 3. Detroit, MI: IEEE Computer Society Press, May 1999, pp. 2292–2297.

[5] A. Martinelli, N. Tomatis, A. Tapus, and R. 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.

[6] T. D. Larsen, M. Bak, N. Andersen, and O. Ravn, "Location estimation for an autonomously guided vehicle using an augmented Kalman filter to autocalibrate the odometry," in *FUSION98 Spie Conference*, Las Vegas, NV, July 1998.

[7] A. T. Ihler, J. W. Fisher, R. L. Moses, and A. 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.

[8] R. Moses and R. Patterson, "Self-calibration of sensor networks," in *SPIE vol. 4743: Unattended Ground Sensor Technologies and Applications IV*, 2002.

[9] C. Kwok, D. Fox, and M. Meila, "Adaptive real-time particle filters for robot localization," in *Proc. of the IEEE International Conference on Robotics & Automation*, 2003.