Symmetry: A Basis for Sensorimotor Reconstruction

Thomas C. Henderson Hongchang Peng,
Christopher Sikorski
University of Utah
Nikhil Deshpande, Eddie Grant
North Carolina State University

UUCS-11-001

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

15 May 2011

Abstract

Given a set of unknown sensors and actuators, sensorimotor reconstruction is achieved by exploiting relations between the sensor data and the actuator control data to determine sets of similar sensors, sets of similar actuators, necessary relations between them, as well as sensorimotor relations to the environment. Several authors have addressed this problem, and we propose here a principled approach that exploits various symmetries and that achieves more efficient and robust results. A theoretical position is defined, the approach shown more efficient than previous work, and experimental results given.

1 Introduction

We propose to explore the thesis that symmetry theory provides key organizing principles for cognitive architectures. As described by Vernon et al. [29], cognition "can be viewed as a process by which the system achieves robust, adaptive, anticipatory, autonomous behavior, entailing embodied perception and action." Their survey considers two basic alternative approaches to cognition: *cognitivist* (physical symbol systems) and *emergent* (dynamical systems), where the cognitivist paradigm is more closely aligned with disembodied symbol manipulation and knowledge representation based on a priori models, and the emergent paradigm purports dynamic skill construction in response to perturbations to the embodiment. An important aspect of this discussion which concerns us here is that raised by Krichmar and Edelman [8]: "the system should be able to effect perceptual categorization: i.e. to organize unlabeled sensory signals of all modalities into categories without a priori knowledge or external instruction." We address this issue and propose that certain fundamental a priori knowledge about symmetries is vital to this function.

Vernon later took up Maturana and Varela's *enaction* conceptual framework for cognitive systems [28]. The goal there is to understand how to describe the role of development in making an agent act effectively and gain new skills. The five basic elements of enaction are: (1) autonomy, (2) embodiment, (3) emergence, (4) experience and (5) sense making. The last one is considered the most important: "emergent knowledge is generated by the system itself and it captures some regularity or lawfulness in the interactions of the system, i.e. its experience. However, the sense it makes is dependent on the way in which it can interact: its own actions and its perceptions of the environments actions on it."

This is the key issue addressed in this paper: it seems somewhat contradictory to say that "regularity or lawfulness" are captured "without a priori knowledge." How can a law or regularity be recognized without knowing the law or rule? Our claim is that symmetries help characterize these regularities.

Our goal is to advance the state of the art in embodied cognitive systems. The requirement for cognitive ability is ubiquitous, and its achievement is an essential step for autonomous mental development. At its root, a cognitive architecture is a structural commitment to processes and representations that permit adaptive control in an operating environment that cannot be modeled completely a priori. A cognitive agent optimizes its behavior to achieve an objective efficiently by finding models that resolve hidden state information and that help it to predict the future under a variety of real-world situations. These processes involve monitoring, exploration, logic, and communication with other agents. It is necessary to create new theories and realizations for cognitive organization in complex, real-time

systems that consist of interacting domain specific agents, each with rich internal state and complex actions in order to facilitate the construction of effectively organized cognitive infrastructure. The proposed technical basis for this is symmetry operators used in perception, representation and actuation.

Cognitive systems perceive, deliberate and act in unstructured environments, and the development of effective mental abilities is a longstanding goal of the AI and intelligent systems communities. The major approaches are the *cognitivist* (physical symbol systems) and *emergent* (dynamical systems) paradigms. For a detailed review of the relevant characteristics of cognitive systems and how these two approaches differ, see [29]. Basically, cognitivists maintain that patterns of symbol tokens are manipulated syntactically, and through percept-symbol associations perception is achieved as abstract symbol representations and actions are causal consequences of symbol manipulation. In contrast, emergent systems are concurrent, self-organizing networks with a global system state representation which is semantically grounded through skill construction where perception is a response to system perturbation and action is a perturbation of the environment by the system. The emergent approach searches the space of closed-loop controllers to build higher-level behavior sequences out of lower ones so as to allow a broader set of affordances in terms of the sensorimotor data stream. We propose to combine these approaches in order to exploit abstraction and specific signal processing domain theories to overcome that complexity. Our specific hypothesis is:

The Domain Theory Hypothesis: Semantic cognitive content may be effectively discovered by restricting controller solutions to be models of specific symmetry theories intrinsic to the cognitive architecture.

The *Domain Theory* predicates: (1) a representation of an innate theory and inference rules for the theory, (2) a perceptual mechanism to determine elements of a set and operators on the set, (3) a mechanism to determine that the set and its operators are a model of the innate theory, and (4) mechanisms to allow the exploitation of the model in learning and model construction.

As pointed out by Weng [31], a major research question in autonomous mental development is ”how a system develops mental capabilities through autonomous real-time interactions with its environment by using its sensors and effectors (controlled by an intrinsic development program coded in the genes or designed in by hand).” Thus, a representation is sought derived from sensorimotor signals as well as the grouping of such signals as processing takes place. Note that this assumes that no coordinate frames exist in this setting; see [27] for a discussion of coordinate frames in biological systems. Asada et al. [1] give a good account of the development of body representations in biological systems and maintain that

"motions deeply participate in the developmental process of sensing and perception." They review data ranging from spinal reflexes with fixed motor patterns, to motion assembly, to mixed motion combinations in the cerebrum. Lungarella [17] also has much to say on this issue, and of great interest here, states that "spontaneous activity in newborns are not mere random movements ... instead organized kicks, arm movements, short phase lags between joints ... may induce correlations between sensing and motor neurons."

Early on, Pierce [23] described an approach to learning a model of the sensor set of an autonomous agent. Features are defined in terms of raw sensor data, and feature operators are defined which map features to features. The goal is to construct a perceptual system for this structure. One of the fundamental feature operators is the *grouping operator* which assigns features to a group if they are similar. This work was extended to spatio-visual exploration in a series of papers [18, 19, 23]. For a detailed critique of Pierce's work, see [5]. Olsson extended this work in a number of papers [9, 10, 11, 12, 13, 20, 21]. He used information theoretic measures for sensorimotor reconstruction, and no innate knowledge of physical phenomena nor the sensors is assumed. Like Pierce, Olsson uses random movements to build the representation and learns the effect of actions on sensors to perform visually guided movements. The major contributions are the analysis of information theoretic measures and motion flow. O'Regan and Noë [22] use the term *sensorimotor contingencies* and give an algorithm which can determine the dimension of the space of the environment by "analyzing the laws that link motor outputs to sensor inputs"; their mathematical formulation is elegant.

2 Symmetry in Sensorimotor Reconstruction

Symmetry [32] plays a deep role in our understanding of the world in that it addresses key issues of invariance, and as noted by Viana [30]: "Symmetry provides a set of rules with which we may describe certain regularities among experimental objects." By determining operators which leave certain aspects of state invariant, it is possible to either identify similar objects or to maintain specific constraints while performing other operations (e.g., move forward while maintaining a constant distance from a wall). For an excellent introduction to symmetry in physics, see [3]. In computer vision, Michael Leyton has described the exploitation of symmetry [14] and the use of group theory as a basis for cognition [15]; we have shown how to use symmetry in range data analysis for grasping [4]. Popplestone and Liu showed the value of this approach in assembly planning [16], while Selig has provided a geometric basis for many aspects of advanced robotics using Lie algebras [25, 26]. Recently, Popplestone and Grupen [24] gave a formal description of general transfer functions (GTF's) and their symmetries.

A symmetry defines an invariant. The simplest invariant is identity. This can apply to an individual item, i.e., a thing is itself, or to a set of similar objects. In general, an invariant is defined by a transformation under which one object is mapped to another. Sensoriomotor reconstruction can be more effectively achieved by finding such symmetry operators on the sensor and actuator data (see also [2, 7]).

Invariants are very useful things to recognize, and we propose that various types of invariant operators provide a basis for cognitive functions, and that it is also useful to have processes that attempt to discover invariance relations among sensorimotor data and subsequently processed versions of that data.

2.1 Symmetry Detection in Signals

Assume a set of sensors, $\mathcal{S} = \{S_i, i = 1 \ldots n_{\mathcal{S}}\}$ each of which produces a finite sequence of indexed sense data values, S_{ij} where i gives the sensor index and j gives an ordinal temporal index, and a set of actuators, $\mathcal{A} = \{A_i, i = 1 \ldots n_{\mathcal{A}}\}$ each of which has a finite length associated control signal, A_{ij}, where i is the actuator index and j is a temporal ordinal index of the control values.

We are interested in determining the similarity of sensorimotor signals. Thus, the type of each sensor as well as the relation to motor control actions play a role. It is quite possible that knowledge of the physical phenomenon that stimulates a sensor may also be exploited to help determine the structure of the sensor system and its relation to motor action and the environment [6].

We suppose that certain 1D signal classes are important and are known a priori to the agent (i.e., that there are processes for identifying signals of these types). The basic signals are:

- *zero*: $y = 0$ (at all samples)
- *constant*: $y = a$ (for some fixed constant a)
- *binary:* y takes on either the value 1 or 0
- *linear*: $y = at + b$ (function of time index)
- *periodic*: has period P and the most significant Fourier coefficients C
- *Gaussian*: sample from Gaussian disctribution with mean μ and variance σ^2

Thus, a first level symmetry is one that characterizes a single signal as belonging to one of these categories. Of course, composite signals can be constructed from these as well, e.g., the impulse signal is a non-zero constant for one step, followed by the zero signal.

Next, pairwise signal symmetries can exist between signals in the same class:

- *linear*
  - same line: $a_1 = a_2, b_1 = b_2$
  - parallel: $a_1 = a_2, b_1 \neq b_2$
  - intersect in point: rotation symmetry about intersection point
- *periodic*
  - same period
  - same Fourier coefficients
- *Gaussian*
  - same mean
  - same variance

2.2 Sensorimotor Reconstruction

The sensorimotor reconstruction process consists of the following steps: (1) perform actuation command sequences, (2) record sensor data, (3) determine sensor equivalence classes, and (4) determine sensor-actuator relations. An additional criterion is to make this process as efficient as possible.

Olsson, Pierce and others produce sensor data by applying random values to the actuators for some preset amount of time, and record the sensor sequences, and then look for similarities in those sequences. This has several problems: (1) there is no guarantee that random movements will result in sensor data that characterizes similar sensors, (2) there is no known (predictable) relation between the actuation sequence and the sensor values, and (3) the simultaneous actuation of multiple actuators confuses the relationship between them and the sensors.

To better understand sensorimotor effects, a systems approach is helpful. That is, rather than giving random control sequences and trying to decipher what happens, it is more effective to hypothesize what the actuator is (given limited choices) and then provide control inputs for which the effects are known. Such hypotheses can be tested as part of the developmental process. The basic types of control that can be applied include: none, impulse, constant, step, linear, periodic, or other (e.g., random).

Next, consider sensors. Some may be time-dependent (e.g., energy level), while others may depend on the environment (e.g., range sensors). Thus, it may be possible to classify ideal (noiseless) sensors into time-dependent and time-independent by applying no actuation and looking to see which sensor signals are not constant (this assumes the spatial environment does not change). Therefore, it may be more useful to not actuate the system, and then classify sensors based on their variance properties. That is, in realistic (with noise) scenarios, it may be possible to group sensors without applying actuation at all.

Consider Pierce's sensorimotor reconstruction process. If realistic noise models are included, the four types of sensors in his experiments (range, broken range, bearing and energy) can all be correctly grouped with no motion at all. (This assumes some energy loss occurs to run the sensors.) All this can be determined just using the equals symmetry operator (identity) and the means and variances of the sensor data sequences.

2.3 Exploiting Actuation

Of course, actuation can help understand the structure of the sensorimotor system. For example, consider what can be determined by simply rotating a two-wheeled robot that has a set of 22 range sensors arranged equi-spaced on a circle. Assume that the control signal results in a slow rotation parallel to the plane of robot motion (i.e., each range sensor moves through a small angle to produce its next sample) and rotates more than 2π radians. Then each range sensor produces a data sequence that is a shifted version of each of the others – i.e., there is a translation symmetry (of periodic signals) between each pair. The general problem is then:

> **General Symmetry Transform Discovery Problem**: Given two sensors, S_1 and S_2, with data sequences T_1 and T_2, find a symmetry operator σ such that $T_2 = \sigma(T_1)$.

2.4 Symmetry-based Sensorimotor Reconstruction Algorithm

Using the symmetries described above, we propose the following algorithms.

Algorithm SBSG: Symmetry-based Sensor Grouping

```
1. Collect sensor data for given period
2. Classify Sensors as Basic Types
3. For all linear sensors
   a. Group if similar regression error
4. For all periodic sensors
  a. Group if similar P and C
5. For all Gaussian sensors
  a. Group if similar variance
```

This algorithm assumes that sensors have an associated noise. Note that this requires no actuation and assumes the environment does not change. Finally, the similarity test for the above algorithm depends on the agent embodiment.

Algorithm SBSR: Symmetry-based Sensorimotor Reconstruction

```
1. Run single actuator and
   collect sensor data for given period
2. For each set of sensors of same type
   a. For each pair
      i. If translation symmetry holds
         Determine shift value
         (in actuation units)
```

This determines the relative distance (in actuation units) between sensors. E.g., for a set of equi-spaced range sensors, this is the angular offset.

3 Comparison to Pierce's Work

3.1 Pierce's Simulation Experiment

A set of simulation experiments are described in Chapter 4 of Pierce's dissertation [23]. The first involves a mobile agent with a set of range sensors, a power level sensor, and four compass sensors. The sensors are grouped and then a structural layout in 2D is determined. The second experiment concerns an array of photoreceptors. Here we examine the first experiment, and in particular, the group generator.

3.2 Pierce's Experiment Definition

The basic setup involves a $6 \times 4\ m^2$ rectangular environment with a mobile robot defined as a point. The robot is equipped with 29 sensors all of which take values in the range from zero to one. Sensors 1 to 24 are range sensors which are arranged in an equi-spaced circle aiming outward from the robot. Range sensor 21 is defective and always returns the value 0.2. Sensor 25 gives the voltage level of the battery while sensors 26 to 29 give current compass headings for East, North, West and South, respectively. The value is 1 for the compass direction nearest the current heading and zero for the other compass sensors. There are two motors, a_0 and a_1, to drive the robot, and these can produce a maximum foward speed of 0.25 m/sec, and a maximum rotation speed of 100 degrees/sec. We assume that the values of the motors range from -1 to 1, where -1 produces a backward motion and 1 produces a forward motion (more specifically, assume the rotational axis of the tracks is aligned with the y-axis; then a positive rotation moves z into x and corresponds to a positive rotation about y in the coordinate frame).

Some details of the motion model are left unspecified; therefore we use the following model:

```
if a0>= 0 and a1>=0
then robot moves forward min(a0,a1)*0.25 m/sec
     robot rotates ((a0-a1)/2)*100 degrees/sec

elseif a0<=0 and a1<=0
then robot moves backward abs(max(a0,a1))*0.25 m/sec
     robot rotates ((a0-a1)/2)*100 degrees/sec
```

```
elseif a0>0 and a1<0
then robot rotates ((a0-a1)/2)*100 degrees/sec

elseif a0>0 and a1<0
then robot rotates ((a0-a1)/2)*100 degrees/sec

end
```

Moreover, if the robot attempts to move out of the rectangular environment, no translation occurs, but rotation does take place.

Two pairwaise metrics are defined (vector and PDF distances), and based on these the sensors are grouped pairwise. Then the transitive closure is taken on these. Pierce runs the simulation for 5 simulated minutes and reports results on the sample data generated from that run. Based on the samples generated from this run, the group generator produces seven groups:

```
Range: {1,2,3,4,5,6,7,8,9,10,11,12,13,
        14,15,16,17,18,19,20,22,23,24}
Defective range: {21}
Battery Voltage: {25}
Compass (East):  {26}
Compass (North): {27}
Compass (West):  {28}
Compass (South): {29}
```

It is not clear why range sensors are grouped, but compass sensors are not.

3.3 Symmetry-based Grouping Operator

Any simulation experiment should carefully state the questions to be answered by the experiment and attempt to set up a valid statistical framework. In addition, the sensitivity of the answer to essential parameters needs to be examined. Pierce does not explicitly formulate a question, nor name a value to be estimated, but it seems clear that some measure of the correctness of the sensor grouping would be appropriate. From the description in the

disertation, Pierce ran the experiment once for 5 minutes of simulated time, and obtained a perfect grouping solution.

From this we infer that the question to be answered is:

> **Grouping Correctness**: What is the correctness performance of the proposed grouping generator?

This requires a definition of correctness for performance and we propose the following (for more details, see [5]):

Correctness Measure: Given (1) a set of sensors, $\{S_i, i = 1 : n\}$ (2) a correct grouping matrix, G, where G is an n by n binary valued matrix with $G(i, j) = 1$ if sensors S_i and S_j are in the same group and $G(i, j) = 0$ otherwise, and (3) H an n by n binary matrix which is the result of the grouping generator, then the grouping correctness measure is:

$$\mu_G(G, H) = \sum_{i=1}^{n} \sum_{j=1}^{n} [(\delta_{i,j})/n^2]$$

$$\delta_{i,j} = 1$$

if G()==H(); 0 otherwise.

3.3.1 Sensor Grouping with Noise (No actuation)

Assume that the sensors each have a statistical noise model. The real-valued range sensors have Gaussian noise sampled from a $\mathcal{N}(0, 1)$ distribution (i.e., $v_{sample} = v_{true} + \omega$. The binary-valued bearing sensors have salt and pepper noise where the correct value is flipped $p\%$ of the time. Finally, the energy sensor has Gaussian noise also sampled from $\mathcal{N}(0, 1)$. (The broken range sensor returns a constant value.)

Based on this, the grouping correctness results are given in Figure 1. Sensor data sampling time was varied from 1 to 20 seconds for binary noise of 5%, 10% and 25%, and Gaussian variance values of 0.1, 1, and 10. Ten trials were run for each case and the means are shown in the figure. As can be seen, perfect sensor grouping is achieved after 20 seconds without any actuation cost. Previous methods required driving both wheels for a longer time and they cost about $30k_{a/s}$ more in energy than our method ($k_{a/s}$ is the actuation to sensing cost ratio).

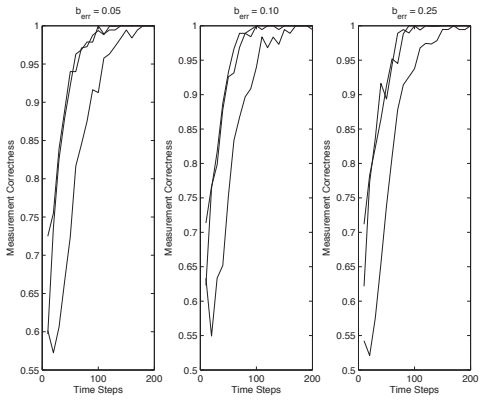

Figure 1: Grouping Correctness vs. Number of Samples; left to right are for binary salt and pepper noise of 5%, 10%, and 25%; curves for 0.1, 1.0, and 10.0 variance are given in each plot.

3.3.2 Sensor Grouping (Actuated)

Given a set of sensors that characterize the group operation nature of an actuator (in this case rotation), the sensors can be grouped based on the fact that similar sensors produce data that has a translation symmetry along the temporal axis. Figure 2 shows representative data for the range and compass sensors. The simple determination of a translaiton symmetry between signals allows both grouping (i.e., the signals match well at some time offset), and the angular difference between the sensors (given by the t_{offset} at which the symmetry occurs); t_{offset} is proportional to the angle between the the sensors in terms of actuation units. Figure 3 shows the perfect grouping result with noise of 1% in the compass sensor data and 0.1 variance in the range sensor data (the figure shows a 29x29 similarity matrix where white indicates sensors are in same group, and black indicates that are not).

4 Physical Experiment

We have performed experiments with physical sensors to validate the proposed approach. Data was taken for both the static case (no actuation) and the actuated case (camera rotation).

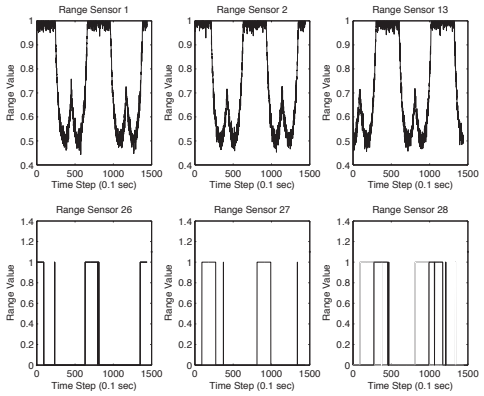

Figure 2: Sensor data showing translation symmetry: Row 1 shows sensors 1, 2, and 13; Row 2 shows compass sensors 27,28, and 29.

4.1 Unactuated Experiment

Two sensors were used in this experiment: a camera and a microphone. The camera was set up in an office and a sequence of 200 images was taken at a 10Hz rate. Figure 4 shows one of these images. The 25x25 center set of pixels from the image comprise a set of 625 pixel signals each of length 200. An example trace and its histogram are given in Figure 5. As can be seen, this is qualitatively a Gaussian sample. Figure 6 shows a 200 sequence signal of microphone data, and its histogram which also looks Gaussian.

The application of our symmetry detectors classified all pixel and microphone signals as Gaussian signals, and grouped the pixel signals separately from the microphone due to the difference in their variance properties.

4.2 Actuated Experiment

We also took a set of images by rotating the camera by one degree for 360 degrees. Domain translation symmetry allows the identification of all the pixel signals along a row as similar to each other (i.e., they are all in the plane of the rotation). Due to the translation amount, the offset between the signals is also discovered.

Figure 3: Grouping Matrix: 29×29 binary matrix; sensors 1-24 are range sensors (sensor 21 returns constant value); 25 is energy; 26-29 are compass sensors.

5 Conclusions and Future Work

We propose symmetry theory as a basis for sensorimotor reconstruction in embodied cognitive agents and have shown that this allows the identification of structure with simple and elegant algorithms which are very efficient. The exploitation of noise structure in the sensors allows unactuated grouping of the sensors, and a simple one actuator rotation permits the recovery of the spatial arrangement of the sensors. This method was shown to hold for physical sensors as well.

Several directions remain to be explored:

1. Consider rotational actuators; these can be seen to define a group in the following way: any specific rotation is an element of the group set, and application of rotation is the operator. Group properties can be seen to hold in that (i) the sequential application of two rotations is a rotation, (ii) the opposite rotation is the inverse element, (iii) the application of no actuation is the identity element, and (iv) associativity holds. [Note that rotation in just one sense forms a group, and various combinations of actuators may form larger groups - e.g., two wheels.]

   $\rightarrow$ The analysis of actuators as specific group operators requires study.

2. Higher-dimensional symmetries offer many opportunities for research. For example, the transformation from spatial image layout to log-polar form allows 1D symmetries to be sought which characterize object scaling and rotation.

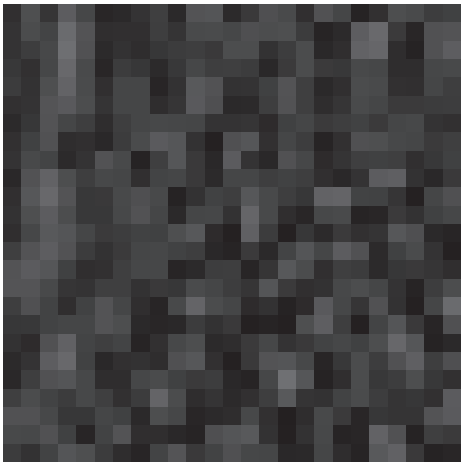

Figure 4: One of the 200 Static Images.

→ The analysis of higher-dimensional symmetries requires study.

3. Higher-level sensorimotor symmetries will allow the conceptualization of physical objects in terms of sensorimotor sequences characterized by some invariant (e.g., stand-off distance in circumlocuting the object).

   → The analysis of symmetries in sensormotor interactions with the environment requires study.

4. Finally, we are instrumenting a set of mobile robots with range and other sensors and a series of experiments will be conducted to study these broader issues.

   → Experimental studies in broader environmental interaction are required.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1021038.

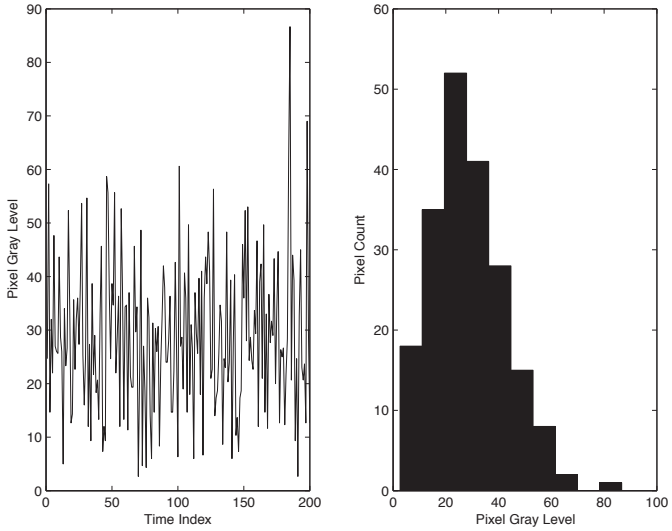

Figure 5: Trace and Histogram of the 200 Pixel Values of the Center Pixel of the Images.

A Basic Signal Classification

The determination of the similarity of signals is an important aspect of sensorimotor reconstruction. We propose that signals be classified into a small set of basic types, and then sets of similar signals can be found based on their types and parameters. The basic signal types are:

1. *constant*: every value is exactly the same.
2. *linear*: $y = ax + b + \omega$, where ω represents noise.
3. *periodic*: $\exists T \ni \forall t y(t) = y(t + T) + \omega$, where ω represents noise
4. *Gaussian*: y is a sample from $\mathcal{N}(\mu, \sigma^2)$, where $\mathcal{N}$ is the normal distribution with mean μ and variance σ^2.

A.1 Constant Signals

The main point about constant signals is that each signal value, $y(t)$, is exactly equal to every other signal value. The associated parameter of a constant signal is the value of the constant.

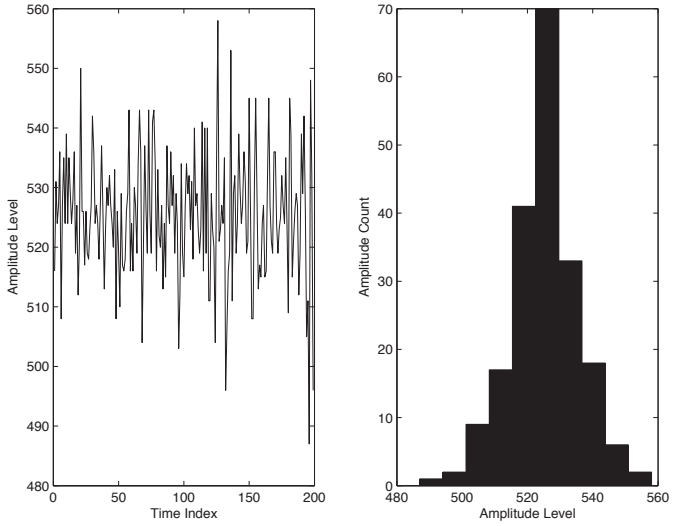

Figure 6: Trace and Histogram of the 200 Amplitude Values of the Microphone Data.

Algorithm: SYM_constant

Input: y (an n vector)
Output: b (Boolean): 1 if constant signal, else 0
c (float): value of constant signal

$c = y(1)$;
for $i \leftarrow 2 : n$
  if $c \neq y(i)$
    $b \leftarrow 0$;
    $return$;
  end
end

A.2 Linear Signals

Signal values are acquired sequentially in time, and each sample is assigned the next integer index, starting at 1. That is, the independent variable for a signal ranges through the whole numbers (i.e., $\{1, 2, 3, \ldots\}$), and therefore, no vertical lines are possible. A least squares fit

is made to the signal points:

$$\{(i, y(i)|i = 1, n\}$$

Next, the vertical distances of the signal points to the line are checked to see if they form a sample from a Gaussian distribution. If so, the signal is characterized as linear.

Algorithm: SYM_linear

Input: y (an n vector)
Output: e (float): error in linear fit
a,b (float): linear parameters ($y = ax + b$)
m,s (float): noise parameters ($\mathcal{N}(m, s)$)

```
[params,error] = polyfit([1:n],y,1);
a = params(1);
b = params(2);
e = error.normr
vals = polyfit(params,[1:n]);
diffs = y-vals;
m = mean(diffs);
s = var(diffs);
```

A.3 Periodic Signals

A periodic signal is characterized by the fact that there exists a value T such that $y(t) = y(t+T)$ for all T. Of course, noise and sampling effects disturb the equality. Our approach to the characterization of periodic signals involves a two-part analysis: (1) find maxima and minima to determine possible periods T, and (2) check $|y(t) - y(t + T)|$ for signal points up to $y(n - T)$. The likelihood that the signal is periodic depends on finding a suitable period, T, as well as the associated error in y values and t displacements of best matching values.

Algorithm: SYM_periodic

Input: y (an n vector)

Output: e (float): error in periodic fit
T (float): period estimate so y(t) = y(t+T)

```
    yc = low_pass_filter(y);
  T_set = find_best_estimates_for_T(yc);
  T = find_best_T_from_distributions_of_error(T_set,yc);
```

A.4 Gaussian Signals

Samples from a Gaussian distribution (called a Gaussian signal) are characterized by the fact that most of the power in the signal autocorrelation is concentrated in the 0 displacement component.

Algorithm: SYM_Gaussian

Input: y (an n vector)
Output: mu (float): mean of signal
sigma2 (float): variance of signal

```
  yc = autocorrelation(y);
  if most_magnitude_in_0_component(yc)
    mu = mean(y);
    sigma2 = variance(y);
  end
```

These basic signal classification algorithms were tested on 128 1-D signals of various types (including 2 acoustic recordings - one a periodic tone, the other background Gaussian noise), and the resulting confusion matrix was:

Type/Type	*constant*	*linear*	*periodic*	*Gaussian*
constant	2	0	0	0
linear	0	42	0	0
periodic	0	1	47	5
Gaussian	0	0	0	31

The errors on the periodic signals are mainyly due to the high amount of Gaussian noise in the periodic signals. The periodic nature of these signals is difficult to ascertain.

A test was also performed on two sequences of 200 25x25 images. One set of images, $D(r, c, t)$, where r is the row, c is the column, and t is the time, was produced by taking images of a dark scene at 0.1 second intervals. Signals were extracted at each pixel as $S_{r,c} = \{D(r, c, 1 : 200)\}$. The second set of images was obtained by rotating the camera about the z-axis 4π degrees, thus, producing at each pixel a periodic signal. The test results on these pixel signals were (1) 621 of 625 (99%) were correctly classified as Gaussian and 4 were mis-classified periodic), and (2) 442/625 (71%) of the periodic signals were classified periodic. Figure 7 shows the signal generated by the central pixel (13,13) of the Gaussian images, and Figure 8 shows the central pixel (13,13) of the set of periodic images.

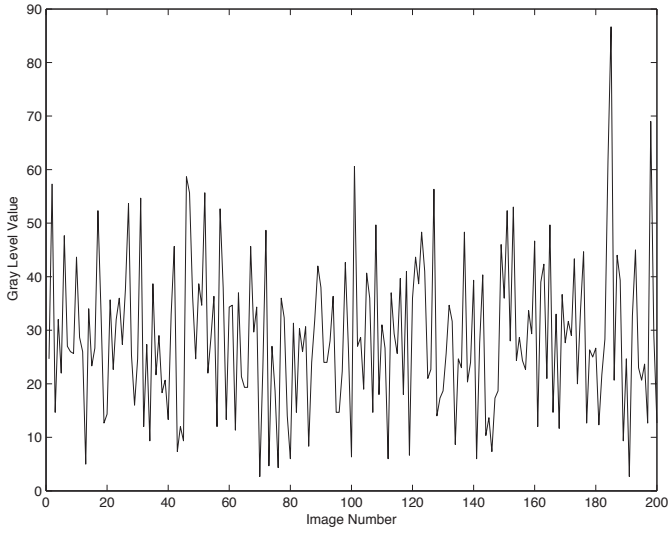

Figure 7: Center Pixel Signal of Gaussian Image Sequence.

B Matlab Code

```
%-------------------------------------------------------------
function [best_T,best_expected_error,T_expected_error] = SYM_best_T(...
    T_dist_hist_t,T_dist_hist_y,y)
% SYM_best_T - determine best value T for period
% On input:
%     T_dist_hist_t (structure): has histograms of difference between
```

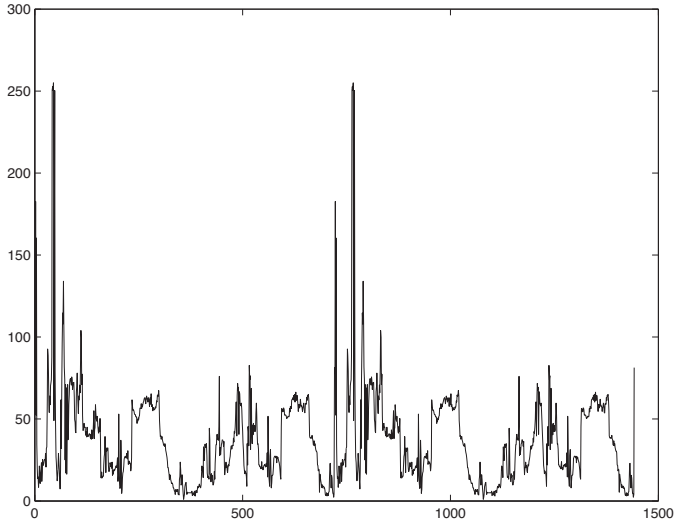

Figure 8: Central Pixel Signal of Periodic Image Sequence.

```
%                   proposed T and actual spacing
%     T_dist_hist_y (structure): has histograms of y value errors at T
%     y (vector): input signal
% On output:
%     best_T (float): best estimate of period T
%     best_val (float): expected value of error from T for best T
%     T_expectde_error (vector): expected values of error for all T's
% Call:
%     [bT,Bv,Tv] = SYM_best_T(T_t,T_y,y);
% Author:
%     T. Henderson
%     UU
%     Spring 2011
%

THRESH = 4;

num_pts = length(y);
half_pts = ceil(num_pts/2);
best_T = 0;
best_expected_error = 0;
T_expected_error = [];
if isempty(T_dist_hist_t)
    return
```

```
end

num_T = length(T_dist_hist_t);
T_expected_error = zeros(1,num_T);
T_candidates = zeros(1,num_T);
for t = 1:num_T
    T_expected_error(t) = dot(T_dist_hist_t(t).htv,...
        (T_dist_hist_t(t).hth/sum(T_dist_hist_t(t).hth)));
    T_candidates(t) = T_dist_hist_t(t).T;
end

[T_sorted_expected_error_vals,T_sorted_expected_error_indexes] = ...
    sort(T_expected_error);
best_T = T_dist_hist_t(T_sorted_expected_error_indexes(1)).T;
best_expected_error = T_sorted_expected_error_vals(1);
best_T_index = T_sorted_expected_error_indexes(1);
[T_sorted_candidates_vals,T_sorted_candidates_indexes] = ...
    sort(T_candidates);
even_better_T = Inf;
even_better_expected_error = 0;
found = 0;
for t = 1:num_T
    if (t~=best_T_index)&&(T_sorted_candidates_vals(t)<best_T)&&...
            SYM_close_mult(T_sorted_candidates_vals(t),best_T)&&...
            abs(T_expected_error(t)-best_expected_error)<THRESH
        if (found==0)||T_candidates(t)<even_better_T
            even_better_T = T_candidates(t);
            even_better_expected_error = T_expected_error(t);
            even_better_index = t;
            found = 1;
        end
    end
end
if found==1
    best_T = even_better_T;
    best_val = even_better_expected_error;
end

if half_pts+2<best_T
    best_T = 0;
    best_expected_error = 0;
```

```
    T_expected_error = [];
end

%-------------------------------------------------------------
function b = SYM_close_mult(v,w)
% SYM_close_mult - determine if there exists an n so that nv~w
% On intput:
%     v (float): smaller number
%     w (float): larger number
% On output:
%     b (Boolean): 1 if v is close mult of w, else 0
% Call:
%     b = SYM_close_ult(23.3,45);
% Author:
%     T. Henderson
%     UU
%     Spring 2011
%

b = 0;
done = 0;
n = 0;
while done==0
    n = n + 1;
    if abs(v*n-w)<v/10
        b = 1;
        return
    end
    if v*(n-1)>w
        done = 1;
    end
end

%-------------------------------------------------------------
function pt = SYM_closest_int_pt(y_val,t_val,y)
% SYM_closest_int_pt - t value for closest y(t) equal to y_val
% On input:
%     y_val (float): required value of y
%     t_val (float): t value
%     y (vector): signal
% On output:
```

```
%     pt (2x1 vector): closest y point with y value equal y_val
% Call:
%     p1 = SYM_closest_int_pt(2.2,3,y);
% Author:
%     T. Henderson
%     UU
%     Spring 2011
%

pt = [];
int_pts = SYM_line_sig_int(y_val,y);
if isempty(int_pts)
    return
end
[num_pts,dummy] = size(int_pts); best_dist = Inf;
for p = 1:num_pts
    d = abs(t_val-int_pts(p,1));
    if d<best_dist
        pt(1) = int_pts(p,1);
        pt(2) = y_val;
        best_dist = d;
    end
end
pt = pt';

%------------------------------------------------------------
function result = SYM_constant(y)
% SYM_constant - recognize perfectly (exact) constant signal
% On input:
%     y (n vector): vector of length n
% On output:
%     result (structure):
%        .type = 0 (indicates perfectly constant signal)
%        .p (float): likelihood signal is constant (0 or 1)
%        .c (float): constant value of signal
%        .G_power (float): unused - set to 0
% Call:
%     r1 = SYM_constant(ones(1,100));
% Author:
%     T. Henderson
%     UU
```

```
%     Spring 2011
%

c = y(1);
p = 1;
result.type = 0;
result.c = c;
result.p = 0;
result.G_power = 0;

n = length(y);
for i = 2:n
    if c ~= y(i)
        return
    end
end

result.p = 1;

%-------------------------------------------------------
function [T_dist_hist_t,T_dist_hist_y] = SYM_dist_hist(T_set,y)
% SYM_dist_hist - produce distance histogram info for period analysis
% On input:
%     T_set (vector): possible period values
%     y (n vector): input signal
% On output:
%     T_dist_hist_t (structure): has independent variable distance info
%        (i).htv (vector): time variable histogram x-axis (from hist)
%           .hth (vector): histogram info (from hist)
%     T_dist_hist_y (structure): has y value distance info
%        (i).hyv (vector): y variable histogram x-axis (from hist)
%           .hyh (vector): histogram info (from hist)
% Call:
%     [Tdt,Tdy] = SYM_dist_hist(Ts,y);
% Author:
%     T. Henderson
%     UU
%     Spring 2011
%

T_dist_hist_t = [];
```

```
T_dist_hist_y = [];
if isempty(T_set)
    return
end
num_T = length(T_set);
num_samps = length(y);
for t = 1:num_T
    [t,num_T]
    T = T_set(t);
    n = num_samps - ceil(T);
    dists_t = zeros(n,1);
    dists_y = zeros(n,1);
    pt_y_p_dists = zeros(1,n);
    for p = 1:n
        [p,n];
        y_p = y(p);
        t_T = p + T;
        y_T = SYM_interpolate_sig(t_T,y);
        pt_t = [p;y_p];
        pt_T = [t_T;y_T];
        pt_y_p = SYM_closest_int_pt(y_p,t_T,y);
        if ~isempty(pt_y_p)
            pt_y_p_dists(p) = norm(pt_y_p-pt_t);
        end
        if ~isempty(pt_y_p)
            dists_t(p) = abs(pt_y_p(1)-pt_T(1));
            dists_y(p) = abs(pt_t(2)-pt_T(2));
        end
    end
    [hth,htv] = hist(dists_t);
    [hyh,hyv] = hist(dists_y);
    T_dist_hist_t(t).T = T;
    T_dist_hist_t(t).htv = htv;
    T_dist_hist_t(t).hth = hth;
    T_dist_hist_y(t).T = T;
    T_dist_hist_y(t).hyv = hyv;
    T_dist_hist_y(t).hyh = hyh;
end

%-------------------------------------------------------
function result = SYM_Gaussian(y)
```

```
%SYM_Gaussian - Gaussian if autocorrelation has specific form
% On input:
%     y (float vector): function samples
% On output:
%     result (structure):
%       .type (int): 3 (indicates Gaussian)
%       .p (float in [0,1]): likelihood that y is Gaussian sample
%       .mu (float): mean of y
%       .sigma2 (float): variance of y
%       .G_power (float): autocorrelation
% Call:
%     res = SYM_Gaussian(0.001*randn(1000,1));
% Author:
%     T. Henderson
%     Spring 2011
%     UU
%

T1 = 0.10;
MAX_VALS = 2;
MAX_RATIO = 1/3;

result.type = 3;
result.p = 0;
result.mu = 0;
result.sigma2 = 1;
result.G_power = 0;

h = hist(y);     % check if binary signal
indexes = find(h>0);
if length(indexes)<3
    return
end

ym = mean(y);
y0 = y-ym;

yc = xcorr(y0,y0);
max_value = max(yc);
indexes = find(yc>max_value*MAX_RATIO);
num_indexes = length(indexes);
```

```
if num_indexes<MAX_VALS
    [mv,mi] = max(yc);
    result.mu = ym;
    result.sigma2 = var(y);
    result.p = 1 - max([yc(1:mi-1),yc(mi+1:end)])/mv;
    result.G_power = yc;
end

%-------------------------------------------------------------
function statistics = SYM_gen_statistics(signals)
% SYM_gen_statistics - generate statistics for signal classification
% On input:
%     signals (sturcture stored in file):
%       (i).type (int):
%           0: constant
%           1: linear
%           2: periodic
%           3: Gaussian
%       (i).y (num_samps vector): signal values
%       (i).parameters (depends on type):
%          type 0: c (float): constant value
%          type 1: y = ax + b + N(mu,sigma^2)
%                a (float): x coefficient in equation
%                b (float): y intercept in equation
%                m (float): mean noise value in signal
%                s (float): variance in noise in signal
%          type 2: y(t) = y(t+T) + N(mu,sigma^2)
%                T (float): period of signal
%                m (float): mean noise value in signal
%                s (float): variance in noise in signal
%          type 3: y(t) sampled from N(mu,sigma^2)
%                m (float): mean noise value
%                s (float): variance of noise in signal
% On output:
%     statistics (structure):
%        .confusion_matrix (4x4 array): classifications made
%            row 1: constant signals
%            row 2: linear signals
%            row 3: periodic signals
%            row 4: Gaussian signals
```

```
%        .constant_c_mean (float): mean in constant value error
%        .constant_c_var (float): variance in constant value error
%        .linear_a_mean (float): mean in linear a value error
%        .linear_a_var (float): variance in linear a value error
%        .linear_b_mean (float): mean in linear b value error
%        .linear_b_var (float): variance in linear b value error
%        .periodic_T_mean (float): mean in periodic T value error
%        .periodic_T_var (float): variance in periodic T error
%        .Gaussian_mu_mean (float): mean in Gaussian mu error
%        .Gaussian_mu_var (float): variance in Gaussian error
%        .Gaussian_sigma2_mean (float): mean in Gaussian error
%        .Gaussian_sigma2_var (float): var in Gaussian error
%        .likelihoods (vector): likelihoods produced classifiers
%        .res_constants (structure): output from SYM_test_function
%        .res_linear (structure): output from SYM_test_function
%        .res_periodic (structure): output from SYM_test_function
%        .res_Gaussian (structure): output from SYM_test_function
% Call:
%     sig_1 = SYM_gen_test_signals(100,1,1);
%     s1 = SYM_gen_statistics(sig_1);
% Author:
%     T. Henderson
%     UU
%     Spring 2011
%

statistics.confusion_matrix = zeros(4,4);
statistics.constant_c_mean = 0;
statistics.linear_a_mean = 0;
statistics.linear_b_mean = 0;
statistics.periodic_T_mean = 0;
statistics.Gaussian_mu_mean = 0;
statistics.Gaussian_sigma2_mean = 0;
statistics.constant_c_var = 0;
statistics.linear_a_var = 0;
statistics.linear_b_var = 0;
statistics.periodic_T_var = 0;
statistics.Gaussian_mu_var = 0;
statistics.Gaussian_sigma2_var = 0;

num_signals = length(signals);
```

```
res_constant = SYM_test_function('SYM_constant',signals);
res_linear = SYM_test_function('SYM_linear',signals);
res_periodic = SYM_test_function('SYM_periodic',signals);
res_Gaussian = SYM_test_function('SYM_Gaussian',signals);
statistics.res_constant = res_constant;
statistics.res_linear = res_linear;
statistics.res_periodic = res_periodic;
statistics.res_Gaussian = res_Gaussian;

c_err = [];
a_err = [];
b_err = [];
T_err = [];
mu_err = [];
sigma2_err = [];
likelihoods = zeros(num_signals,5);

p = zeros(1,4);
for s = 1:num_signals
    s_type = signals(s).type;
    p(1) = res_constant(s).s_p;
    p(2) = res_linear(s).s_p;
    p(3) = res_periodic(s).s_p;
    p(4) = res_Gaussian(s).s_p;
    likelihoods(s,1) = s_type;
    likelihoods(s,2:5) = p;
    [max_p_val,max_p_index] = max(p);
    statistics.confusion_matrix(s_type+1,max_p_index) = ...
        statistics.confusion_matrix(s_type+1,max_p_index) + 1;
    switch max_p_index
        case 1   % constant signal
            if signals(s).type==0
                c_err = [c_err,...
                    abs(res_constant(s).s_c-signals(s).c)];
            end
        case 2   % linear signal
            if signals(s).type==1
                a_err = [a_err,abs(res_linear(s).s_a-signals(s).a)];
                b_err = [b_err,abs(res_linear(s).s_b-signals(s).b)];
            end
```

```
        case 3   % periodic signal
            if signals(s).type==2
                T_err = [T_err,...
                 abs(res_periodic(s).s_T-signals(s).T)];
            end
        case 4   % Gaussian signal
            if signals(s).type==3
                mu_err = [mu_err,...
                  abs(res_Gaussian(s).s_mu-signals(s).mu)];
                sigma2_err = [sigma2_err,...
                    abs(res_Gaussian(s).s_sigma2-signals(s).sigma2)];
            end
    end
end
if ~isempty(c_err)
    statistics.constant_c_mean = mean(c_err);
    statistics.constant_c_var = var(c_err);
end
if ~isempty(a_err)
    statistics.linear_a_mean = mean(a_err);
    statistics.linear_a_var = var(a_err);
    statistics.linear_b_mean = mean(b_err);
    statistics.linear_b_var = var(b_err);
end
if ~isempty(T_err)
    statistics.periodic_T_mean = mean(T_err);
    statistics.periodic_T_var = var(T_err);
end
if ~isempty(mu_err)
    statistics.Gaussian_mu_mean = mean(mu_err);
    statistics.Gaussian_mu_var = var(mu_err);
    statistics.Gaussian_sigma2_mean = mean(sigma2_err);
    statistics.Gaussian_sigma2_var = var(sigma2_err);
end
statistics.likelihoods = likelihoods;

%-------------------------------------------------------------
function statistics = SYM_gen_statistics_one_function(fname,signals)
% SYM_gen_statistics - generate statistics for signal classification
% On input:
%     fname (string): name of classification function
```

```
%     signals (sturcture stored in file):
%       (i).type (int):
%           0: constant
%           1: linear
%           2: periodic
%           3: Gaussian
%       (i).y (num_samps vector): signal values
%       (i).parameters (depends on type):
%          type 0: c (float): constant value
%          type 1: y = ax + b + N(mu,sigma^2)
%                a (float): x coefficient in equation
%                b (float): y intercept in equation
%                m (float): mean noise value in signal
%                s (float): variance in noise in signal
%          type 2: y(t) = y(t+T) + N(mu,sigma^2)
%                T (float): period of signal
%                m (float): mean noise value in signal
%                s (float): variance in noise in signal
%          type 3: y(t) sampled from N(mu,sigma^2)
%                m (float): mean noise value
%                s (float): variance of noise in signal
% On output:
%     statistics (structure):
%        .confusion_matrix (4x4 array): classifications made
%             row 1: constant signals
%             row 2: linear signals
%             row 3: periodic signals
%             row 4: Gaussian signals
%        .constant_c_mean (float): mean in constant value error
%        .constant_c_var (float): variance in constant value error
%        .linear_a_mean (float): mean in linear a value error
%        .linear_a_var (float): variance in linear a value error
%        .linear_b_mean (float): mean in linear b value error
%        .linear_b_var (float): variance in linear b value error
%        .periodic_T_mean (float): mean in periodic T value error
%        .periodic_T_var (float): variance in periodic T value error
%        .Gaussian_mu_mean (float): mean in Gaussian mu value error
%        .Gaussian_mu_var (float): variance in Gaussian error
%        .Gaussian_sigma2_mean (float): mean in Gaussian error
%        .Gaussian_sigma2_var (float): var in Gaussian error
%        .likelihoods (vector): likelihoods produced by classifiers
```

```
%        .res (structure): output from SYM_test_function
% Call:
%     sig_1 = SYM_gen_test_signals(100,1,1);
%     s1 = SYM_gen_statistics_one_function('SYM_Gaussian',sig_1);
% Author:
%     T. Henderson
%     UU
%     Spring 2011
%

statistics.confusion_matrix = zeros(1,4);
statistics.constant_c_mean = 0;
statistics.linear_a_mean = 0;
statistics.linear_b_mean = 0;
statistics.periodic_T_mean = 0;
statistics.Gaussian_mu_mean = 0;
statistics.Gaussian_sigma2_mean = 0;
statistics.constant_c_var = 0;
statistics.linear_a_var = 0;
statistics.linear_b_var = 0;
statistics.periodic_T_var = 0;
statistics.Gaussian_mu_var = 0;
statistics.Gaussian_sigma2_var = 0;

num_signals = length(signals);

res = feval(f_handle,y);
statistics.res = res;

c_err = [];
a_err = [];
b_err = [];
T_err = [];
mu_err = [];
sigma2_err = [];
likelihoods = zeros(num_signals,3);
f_type = res.type;
likelihoods(:,1) = f_type;

for s = 1:num_signals
    s_type = signals(s).type;
```

```
    p = res(s).s_p;
    likelihoods(s,2) = s_type;
    likelihoods(s,3) = p;
    switch f_type
        case 1   % constant signal
            if signals(s).type==0
                c_err = [c_err,...
                    abs(res_constant(s).s_c-signals(s).c)];
            end
        case 2   % linear signal
            if signals(s).type==1
                a_err = [a_err,abs(res_linear(s).s_a-signals(s).a)];
                b_err = [b_err,abs(res_linear(s).s_b-signals(s).b)];
            end
        case 3   % periodic signal
            if signals(s).type==2
                T_err = [T_err,...
                   abs(res_periodic(s).s_T-signals(s).T)];
            end
        case 4   % Gaussian signal
            if signals(s).type==3
                mu_err = [mu_err,...
                   abs(res_Gaussian(s).s_mu-signals(s).mu)];
                sigma2_err = [sigma2_err,...
                    abs(res_Gaussian(s).s_sigma2-signals(s).sigma2)];
            end
    end
end
if ~isempty(c_err)
    statistics.constant_c_mean = mean(c_err);
    statistics.constant_c_var = var(c_err);
end
if ~isempty(a_err)
    statistics.linear_a_mean = mean(a_err);
    statistics.linear_a_var = var(a_err);
    statistics.linear_b_mean = mean(b_err);
    statistics.linear_b_var = var(b_err);
end
if ~isempty(T_err)
    statistics.periodic_T_mean = mean(T_err);
    statistics.periodic_T_var = var(T_err);
```

```
end
if ~isempty(mu_err)
    statistics.Gaussian_mu_mean = mean(mu_err);
    statistics.Gaussian_mu_var = var(mu_err);
    statistics.Gaussian_sigma2_mean = mean(sigma2_err);
    statistics.Gaussian_sigma2_var = var(sigma2_err);
end
statistics.likelihoods = likelihoods;

%-------------------------------------------------------
function signals =SYM_gen_test_signals2(num_samps,num_trials,default)
% SYM_gen_test_signals2 - generate set of signals for basic type test
% On Input:
%     num_samps (int): number of samples per test signal
%     num_trials (int): number of samples from given distribution
%     default (Boolean): if 1 use the random default stream, else not
% On output:
%     signals (structure stored in file):
%       (i).type (int):
%           0: constant
%           1: linear
%           2: periodic
%           3: Gaussian
%       (i).y (num_samps vector): signal values
%       (i).parameters (depends on type):
%          type 0: c (float): constant value
%          type 1: y = ax + b + N(mu,sigma^2)
%                a (float): x coefficient in equation
%                b (float): y intercept in equation
%                mu (float): mean noise value in signal
%                sigma2 (float): variance in noise in signal
%          type 2: y(t) = y(t+T) + N(mu,sigma^2)
%                T (float): period of signal
%                mu (float): mean noise value in signal
%                sigma2 (float): variance in noise in signal
%          type 3: y(t) sampled from N(mu,sigma^2)
%                mu (float): mean noise value
%                sigma2 (float): variance of noise in signal
% Call:
%     SYM_gen_test_signals2(200,5,1);
% Author:
```

```
%     T. Henderson
%     UU
%     Spring 2011
%

if default==1
    randn('state',1);
end

% Signal 1: Constant with y = 0
y = zeros(1,num_samps);
signals(1).type = 0;
signals(1).y = y;
signals(1).c = 0;

% Signal 2: Constant with y = 2.5
y = 2.5*ones(1,num_samps);
signals(2).type = 0;
signals(2).y = y;
signals(2).c = 2.5;

% Signal 3: Linear with y = x   (no noise)
x = -5:10/num_samps:5;
x = x(1:num_samps);
y = x;
signals(3).type = 1;
signals(3).a = 1;
signals(3).b = 0;
signals(3).mu = 0;
signals(3).sigma2 = 0;
signals(3).y = y;

index = 3;
% Signals 4: Linear with y = x + N(0,.1)
for t = 1:num_trials
    index = index + 1;
    y1 = y + 0.1*randn(1,num_samps);
    signals(index).type = 1;
    signals(index).a = 1;
    signals(index).b = 0;
    signals(index).mu = 0;
```

```
    signals(index).sigma2 = 0.1;
    signals(index).y = y1;
end

% Signals 5: Linear with y = x + N(0,1)
for t = 1:num_trials
    index = index + 1;
    y1 = y + randn(1,num_samps);
    signals(index).type = 1;
    signals(index).a = 1;
    signals(index).b = 0;
    signals(index).mu = 0;
    signals(index).sigma2 = 1;
    signals(index).y = y1;
end

index = index + 1;
% Signal 6: Linear with y = 2x+1  (no noise)
x = -5:10/num_samps:5;
x = x(1:num_samps);
y = 2*x+1;
signals(index).type = 1;
signals(index).a = 2;
signals(index).b = 1;
signals(index).mu = 0;
signals(index).sigma2 = 0;
signals(index).y = y;

% Signals 7: Linear with y = 2x + 1 + N(0,.1)
for t = 1:num_trials
    index = index + 1;
    y1 = y + 0.1*randn(1,num_samps);
    signals(index).type = 1;
    signals(index).a = 2;
    signals(index).b = 1;
    signals(index).mu = 0;
    signals(index).sigma2 = 0.1;
    signals(index).y = y1;
end

% Signals 8: Linear with y = 2x + 1 + N(0,1)
```

```
for t = 1:num_trials
    index = index + 1;
    y1 = y + randn(1,num_samps);
    signals(index).type = 1;
    signals(index).a = 2;
    signals(index).b = 1;
    signals(index).mu = 0;
    signals(index).sigma2 = 1;
    signals(index).y = y1;
end

index = index + 1;
% Signals 9: Periodic with y = sin(x) with no noise
dx = 4*2*pi/num_samps;
x = 0:4*2*pi/num_samps:4*2*pi;
x = x(1:num_samps);
T = (2*pi)/dx;
y = sin(x);
signals(index).type = 2;
signals(index).T = T;
signals(index).mu = 0;
signals(index).sigma2 = 0;
signals(index).y = y;

% Signals 10: Periodic with y = sin(x) + N(0,0.1)
for t = 1:num_trials
    index = index + 1;
    y1 = y + 0.1*randn(1,num_samps);
    signals(index).type = 2;
    signals(index).T = T;
    signals(index).mu = 0;
    signals(index).sigma2 = 0.1;
    signals(index).y = y1;
end

% Signals 11: Periodic with y = sin(x) + N(0,1)
for t = 1:num_trials
    index = index + 1;
    y1 = y + randn(1,num_samps);
    signals(index).type = 2;
    signals(index).T = T;
```

```
    signals(index).mu = 0;
    signals(index).sigma2 = 1;
    signals(index).y = y1;
end

index = index + 1;
% Signals 12: Periodic with y = sin(x) + sin(3x/2) with no noise
dx = 4*2*pi/num_samps;
x = 0:4*2*pi/num_samps:4*2*pi;
x = x(1:num_samps);
T = 4.3*pi/dx;
y = sin(x) + sin(3*x/2);
signals(index).type = 2;
signals(index).T = T;
signals(index).mu = 0;
signals(index).sigma2 = 0;
signals(index).y = y;

% Signals 13: Periodic with y = sin(x) + sin(3x/2) + N(0,0.1)
for t = 1:num_trials
    index = index + 1;
    y1 = y + 0.1*randn(1,num_samps);
    signals(index).type = 2;
    signals(index).T = T;
    signals(index).mu = 0;
    signals(index).sigma2 = 0.1;
    signals(index).y = y1;
end

% Signals 14: Periodic with y = sin(x) + sin(3x/2) + N(0,1)
for t = 1:num_trials
    index = index + 1;
    y1 = y + randn(1,num_samps);
    signals(index).type = 2;
    signals(index).T = T;
    signals(index).mu = 0;
    signals(index).sigma2 = 1;
    signals(index).y = y1;
end

% Signals 15: Periodic signal comprised of repeated random sample
```

```
for t = 1:num_trials
    index = index + 1;
    sn = max(1,floor(num_samps/3));
    s = randn(1,sn);
    y = [s,s,s,s];
    y = y(1:num_samps);
    signals(index).type = 2;
    signals(index).T = sn;
    signals(index).mu = 0;
    signals(index).sigma2 = 0;
    signals(index).y = y;
end

% Signals 16: Gaussian samples from N(0,0.01)
for t = 1:num_trials
    index = index + 1;
    y = 0.01*randn(1,num_samps);
    signals(index).type = 3;
    signals(index).mu = 0;
    signals(index).sigma2 = 0.01;
    signals(index).y = y;
end

% Signals 17: Gaussian samples from N(0,0.1)
for t = 1:num_trials
    index = index + 1;
    y = 0.1*randn(1,num_samps);
    signals(index).type = 3;
    signals(index).mu = 0;
    signals(index).sigma2 = 0.1;
    signals(index).y = y;
end

% Signals 18: Gaussian samples from N(0,1)
for t = 1:num_trials
    index = index + 1;
    y = randn(1,num_samps);
    signals(index).type = 3;
    signals(index).mu = 0;
    signals(index).sigma2 = 1;
    signals(index).y = y;
```

```
end

index = index + 1;
% Signal 19: Actual mono tone recorded signal
load micro_data
signals(index).type = 2;
signals(index).T = 7.3;
signals(index).mu = 0;
signals(index).sigma2 = 0.01;
signals(index).y = micro_tone;

index = index + 1;
% Signal 20: Actual background noise recorded signal
signals(index).type = 3;
signals(index).mu = mean(micro_static);
signals(index).sigma2 = var(micro_static);
signals(index).y = micro_static;

%-------------------------------------------------------
function v = SYM_interpolate_sig(t,y)
% SYM_interpolate_sig - linear interpolation of signal
% On input:
%     t (float): independent variable
%     y (vector): signal
% On output:
%     v (float): interpolated value y(t)
% Call:
%     x = 0:0.1:2*pi;
%     ys = sin(x);
%     v = SYM_interpolate_sig(3.2,ys);
% Author:
%     T. Henderson
%     UU
%     Spring 2011
%

v = NaN;
num_samps = length(y);
if (t<1)||(t>num_samps)
    return
end
```

```
if t==1
    v = y(1);
    return
end
if t==num_samps
    v = y(num_samps);
    return
end

s1 = floor(t);
s2 = s1 + 1;
frac = t - s1;
v = (1-frac)*y(s1) + frac*y(s2);

%-------------------------------------------------------------
function result = SYM_linear(y)
% SYM_linear - classify linear signals
% On input:
%     y (n vector): input signal
% On output:
%     result (structure)
%        .type (int): set to 1 (indicates linear)
%        .p (float): likelihood signal is linear
%        .a (float): slope of line
%        .b (float): y intercept of line
%        .err (float): error in fit of line
%        .mu (float): mean of signal noise
%        .sigma2 (float): variance of signal noise
%        .G_power (vector): autocorrelation of error values
% Call:
%     r_lin = SYM_linear([1:20]);
% Author:
%     T. Henderson
%     UU
%     Spring 2011
%

MIN_PROB = 0.5;
GAUSS_THRESH = 0.4;

result.type = 1;
```

```
result.p = 0;
result.a = 0;
result.b = 0;
result.err = Inf;
result.mu = 0;
result.sigma2 = 0;
result.G_power = [];

num_samps = length(y);
x = [1:num_samps];
[pars,err] = polyfit(x,y,1);
a = pars(1);
b = pars(2);
vals = polyval(pars,x);
diffs = y-vals;
r_Gau = SYM_Gaussian(diffs);
m = mean(diffs);
s = var(diffs);
e = err.normr;

result.a = a;
result.b = b;
spread = max(y) - min(y);
if spread<eps
    result.p = 1;
else
    result.p = 1 - (max(abs(diffs))/(max(y)-min(y)));
end
int_pts = SYM_line_sig_int(mean(diffs),diffs);
if ~isempty(int_pts)
    int_p = length(int_pts(:,1))/num_samps;
    if int_p<GAUSS_THRESH
        result.p = int_p*result.p;
    end
end
result.err = e;
result.mu = m;
result.sigma2 = s;
result.G_power = r_Gau.G_power;

%-------------------------------------------------------------
```

```
function int_pts = SYM_line_sig_int(c,y)
% SYM_line_sig_int - intersection points of line with signal
% On input:
%     c (float): y-value for horizontal line
%     y (1xm vector): 1D signal values
% On output:
%     int_pts (nx2): intersection points of line with signal
% Call:
%     ip = SYM_line_sig_int(0.2,y);
% Author:
%     T. Henderson
%     UU
%     Spring 2011
%

LINE_DIST_THRESH = 0.00001;

num_samps = length(y);
int_pts = [];
line_c = [1,c,0;2,c,0];
for p = 1:num_samps-1
    line_p = [p, y(p), 0; p+1, y(p+1), 0];
    [p_int_pt, b_p] = cv_int_line_line(line_c, ...
         line_p, LINE_DIST_THRESH);
    if abs(b_p)==1
        min_y = min(y(p),y(p+1));
        max_y = max(y(p),y(p+1));
        in_range = (min_y<=p_int_pt(2))&&(p_int_pt(2)<=max_y);
        if in_range
            int_pts = [int_pts;p_int_pt(1),p_int_pt(2)];
        end
    end
end

%-------------------------------------------------------------
function y_local_max_interp = SYM_local_max_periodic_interp(y)
% SYM_local_max_periodic_interp - find local maxima by interpolation
% On input:
%     y (n vector): input signal
% On output:
%     y_local_max (kx2 array): local max array
```

```
% Call:
%     lm = SYM_local_max_periodic_interp(y);
% Author:
%     T. Henderson
%     UU
%     Spring 2011
%

k = 3;
num_pts = length(y);
y_local_max = [];
y_local_max_indexes = [];

for p = 1:num_pts
    p_min = max(p-k,1);
    p_max = min(p+k,num_pts);
    vals = y(p_min:p_max);
    if y(p)>=max(vals)&&y(p)>min(vals)
        y_local_max = [y_local_max; p, y(p)];
        y_local_max_indexes = [y_local_max_indexes, p];
    end
end

num_local_max = length(y_local_max_indexes);
y_local_max_interp = y_local_max;
for n = 1:num_local_max
    n_ind = y_local_max_indexes(n);
    if (n_ind>1)&&(n_ind<num_pts)
        [p,s] = polyfit([n_ind-1:n_ind+1],[y(n_ind-1:n_ind+1)],2);
        x_max = -p(2)/(2*p(1));
        y_max = polyval(p,x_max);
        y_local_max_interp(n,1) = x_max;
        y_local_max_interp(n,2) = y_max;
    end
end

%-------------------------------------------------------------
function result = SYM_periodic(y)
% SYM_periodic - classify signal as periodic and give parameters
% On input:
%     y (vector): input signal
```

```
% On output:
%     result (structure):
%       .type (int): 2 (indicates periodic)
%       .p (float in [0,1]): likelihood that y is periodic signal
%       .T (float): period of y
%       .err (float): error in periodic nature
%       .yc (vector): extracted periodic signal
%       .mu (float): mean of noise in y
%       .sigma2 (float): variance of noise y
%       .G_power (float): autocorrelation of noise in y
% Call:
%   res = SYM_periodic([[1:10],[9:-1:1],[2:10],[9:-1:1]1,[9:-1:1]]);
% Author:
%     T. Henderson
%     Spring 2011
%     UU
%

T1 = 0.2;

PERIOD_THRESH = 1;
warning off

result.type = 2;
result.p = 0;
result.T = 0;
result.err = 0;
result.mu = 0;
result.sigma2 = 0;
result.G_power = 0;

num_samps = length(y);
p_y = 0;
p_t = 0;
T = 0;
yc = y;

indexes = find(y(1)==y);  % check exact repeated sequence
if 1<length(indexes)
    T_candidates = indexes(2:end)-1;
    half_samps = floor(num_samps/2);
```

```
    num_candidates = length(T_candidates);
    for c = 1:num_candidates
        Tc = T_candidates(c);
        found = 1;
        for p = 1:half_samps
            [p,half_samps]
            if (p+Tc<=num_samps)&&(y(p)~=y(p+Tc))
                found = 0;
                break
            end
        end
        if found==1
            result.p = 1;
            result.T = Tc;
            return
        end
    end
end

Y = fft(y);
m = abs(Y);
locs = find(m>(max(m)*T1));
YP = 0*Y;
num_locs = length(locs);
C = zeros(num_locs,2);
for p = 1:num_locs
    YP(locs(p)) = Y(locs(p));
    C(p,1) = locs(p);
    C(p,2) = Y(locs(p));
end
yc = ifft(YP);

y = yc;
y_local_max = SYM_local_max_periodic_interp(y);
T_set = SYM_T_set(y_local_max,y);
[T_dist_hist_t,T_dist_hist_y] = SYM_dist_hist(T_set,y);
[T,T_best_expected_error,T_expected_error] = ...
    SYM_best_T(T_dist_hist_t,T_dist_hist_y,y);
if T<=0
    return
end
```

```
p_t = 1-T_best_expected_error/T;

yc_err = 0;  % include the error to periodic y values in error
num_pts = length(yc);
max_pt = num_pts - ceil(T);
for p = 1:max_pt
    yc_int = SYM_interpolate_sig(p+T,yc);
    yc_err = yc_err + abs(yc_int-yc(p));
end
yc_err_avg = yc_err/max_pt;

result.p = p_t*(1-yc_err_avg/(max(yc)-min(yc)));
result.T = T;
result.yc = yc;
result.err = T_best_expected_error/T;

warning on

%-------------------------------------------------------
function result = SYM_test_function(f_handle, sigs);
%
% SYM_test_function: test basic classifier on set of signals
% On Input:
%     f_handle (string): function name
%     sigs (structure): signal set structure
%       (i).type (int):
%           0: constant
%           1: linear
%           2: periodic
%           3: Gaussian
%       (i).y (num_samps vector): signal values
%       (i).<param> (depends on type):
%          type 0: .c (float): constant value
%          type 1: y = ax + b + N(mu,sigma^2)
%                .a (float): x coefficient in equation
%                .b (float): y intercept in equation
%                .mu (float): mean noise value in signal
%                .sigma2 (float): variance in noise in signal
%          type 2: y(t) = y(t+T) + N(mu,sigma^2)
%                .T (float): period of signal
%                .mu (float): mean noise value in signal
```

```
%                .sigma2 (float): variance in noise in signal
%          type 3: y(t) sampled from N(mu,sigma^2)
%                .mu (float): mean noise value
%                .sigma2 (float): variance of noise in signal
% On Output:
%     result (structure): results
%        like signals structure with following fields for each sig:
%        .s_type (int): signal classified as this
%        .s_p (float): likelihood signal is this type
%        .s_<params>: value for paramter of this type signal
%        .s_err (float): measure signal fit to parameterized signal
%        .s_mu (float): estimate of noise mean
%        .s_sigma2 (float): estimate of noise variance
%        .s_DC (float): percentage of power at 0 in autocorrelation
% Call:
%     res = SYM_test_function('SYM_Gaussian','t1',[0.1],[0.01],10);
% Author:
%     H. Peng and T. Henderson
%     UU
%     Spring 2011
%

num_sigs = length(sigs);

result = sigs;

for s = 1:num_sigs
    y = sigs(s).y;
    res = feval(f_handle,y);
    switch res.type
        case 0
            result(s).s_type = 0;
            result(s).s_p = res.p;
            result(s).s_c = res.c;
            result(s).s_err = 0;
            result(s).s_mu = 0;
            result(s).s_sigma2 = 0;
            result(s).s_DC = res.G_power;
        case 1
            result(s).s_type = 1;
            result(s).s_p = res.p;
```

```
            result(s).s_a = res.a;
            result(s).s_b = res.b;
            result(s).s_err = res.err;
            result(s).s_mu = res.mu;
            result(s).s_sigma2 = res.sigma2;
            result(s).s_DC = res.G_power;
        case 2
            result(s).s_type = 2;
            result(s).s_p = res.p;
            result(s).s_T = res.T;
            result(s).s_err = res.err;
            result(s).s_mu = res.mu;
            result(s).s_sigma2 = res.sigma2;
            result(s).s_DC = res.G_power;
        case 3
            result(s).s_type = 3;
            result(s).s_p = res.p;
            result(s).s_mu = res.mu;
            result(s).s_sigma2 = res.sigma2;
            result(s).s_DC = res.G_power;
    end
end

%-------------------------------------------------------
function T_set = SYM_T_set(y_local_max,y)
% SYM_T_set - get set of possible period values
% On input:
%     y_local_max (kx2 array): local signal maxima as points
%     y (n vector): input signal
% On output:
%     T_set (vector): list of possible period values
% Call:
%     Ts = SYM_T_set(lm,y);
% Author:
%     T. Henderson
%     UU
%     Spring 2011
%

T_set = [];
num_max = length(y_local_max);
```

```
if num_max<2
    return
end
num_samps = length(y);
half_samps = ceil(num_samps/2);

for p = 2:num_max
%    if y_local_max(p,1)<=half_samps
        T_set = [T_set, y_local_max(p) - y_local_max(1)];
%    end
end

if num_max<3
    return
end

for p = 3:num_max
    if y_local_max(p,1)<=half_samps
        T_set = [T_set, y_local_max(p) - y_local_max(2)];
    end
end
```

References

[1] M. Asada, K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, Y. Yoshikawa, M. Ogino, and C. Yoshida. Cognitive Developmental Robotics: A Survey. *IEEE Transactions on Autonomous Mental Development*, 1(1):12–34, 2009.

[2] F. Bullo and R. M. Murray. Proportional Derivative (PD) Control On The Euclidean Group. In *Proceeding of the European Control Conference*, pages 1091–1097, 1995.

[3] J.P. Elliott. *Symmetry in Physics, Vol. I: Principles and Simple Applications*. Oxford University Press, NY, NY, 1979.

[4] R. Grupen and T.C. Henderson. Apparent Symmetries in Range Data. *Pattern Recognition Letters*, 7:107–111, 1988.

[5] T. Henderson and H. Peng. A study of pierce's group generator. Technical Report UUCS-10-001, The University of Utah, December 1984.

[6] T.C. Henderson. *Computational Sensor Networks*. Springer Verlag, New York, NY, 2009.

[7] J.C. Kinsey and L.L. Whitcomb. Adaptive Identification on the Group of Rigid Body Rotations. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3256–3261. IEEE Press, 2005.

[8] J.L. Krichmar and G. Edelman. Principles Underlying the Construction of Brain-Based Devices. In *Proceedings*, pages 37–42, Bristol, UK, 2006. Society for the Study of Artificial Intelligence and the Simulation of Behaviour.

[9] C.L. Nehaniv L. Olsson and D. Polani. Sensory Channel Grouping and Structure from Uninterpreted Sensor Data. In *Proceedings of NASA/DoD Conference on Evolvable Hardware*, pages 153–160, Seattle, WA, 2004. IEEE Computer Society Press.

[10] C.L. Nehaniv L. Olsson and D. Polani. The Effects on Visual Information in a Robot in Environments with Oriented Contours. In L. Berthouze, H. Kozima, C.G. Prince, G. Sandini, G. Stojanov, and G. Metta, editors, *Proceedings of the Fourth International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, pages 83–88, Genoa, Italy, 2004. Lund University Cognitive Studies.

[11] C.L. Nehaniv L. Olsson and D. Polani. Discovering Motion Flow by Temporal-Informational Correlations in Sensors. In L. Berthouze, H. Kozima, C.G. Prince, G. Sandini, G. Stojanov, G. Metta, and C. Balkenius, editors, *Proceedings of the Fifth International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, pages 117–120, Nara, Japan, 2005. Lund University Cognitive Studies.

[12] C.L. Nehaniv L. Olsson and D. Polani. From Unknown Sensors and Actuators to Visually Guided Movement. In *Proceedings of the International Conference on Development and Learning*, pages 1–6, Osaka, Japan, 2005. IEEE Computer Society Press.

[13] C.L. Nehaniv L. Olsson and D. Polani. Sensor Adaptation and Development in Robots by Entropy Maximization of Sensory Data. In *Proceedings of the International Symposium on Computational Intelligence in Robotics and Automation*, pages 587–592, Espoo, Finland, 2005. IEEE Computer Society Press.

[14] M. Leyton. *Symmetry, Causality, Mind*. MIT Press, Cambridge, MA, 1992.

[15] M. Leyton. *A Generative Theory of Shape*. Springer, Berlin, 2001.

[16] Y. Liu and R. Popplestone. A Group Theoretic Formalization of Surface Contact. *International Journal of Robotics Research*, 13(2):148–161, 1994.

[17] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini. Developmental Robotics: A Survey. *Connection Science*, 15(4):151–190, 2003.

[18] J. Modayil and B. Kuipers. Autonomous Development of a Grounded Object Ontology by a Learning Robot. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence*, Vancouver, Canada, 2007. AAAI.

[19] J. Modayil and B. Kuipers. The Initial Development of Object Knowledge by a Learning Robot. *Robotics and Autonomous Systems*, 56(11):879–890, 2008.

[20] L. Olsson, C.L. Nehaniv, and D. Polani. Measuring Informational Distances between Sensors and Sensor Integration. Technical report 431, University of Hertfordshire, Hertfordshire, England, 2005.

[21] L. Olsson, C.L. Nehaniv, and D. Polani. From Unknown Sensors and Actuators to Actions Grounded in Sensorimotor Perceptions. *Connection Science*, 18:121–144, 2006.

[22] J.K. O'Regan and A. Noë. A Sensorimotor Account of Vision and Visual Consciousness. *Behavioral and Brain Sciences*, 24:939–1031, 2001.

[23] D.M. Pierce. *Map Learning with Uninterpreted Sensors and Effectors*. PhD thesis, Austin, Texas, May 1995.

[24] R. Popplestone and R. Grupen. Symmetries in World Geometry and Adaptive Behaviour. In *Proceedings of the Workshop on Algebraic Frames for the Perception Action Cycle*, pages 269–283, Kiel, Germany, 2000.

[25] J.M. Selig. Lie Groups and Lie Algebras in Robotics. In J. Byrnes, editor, *Proceedings of the NATO Advanced Study Institute on Computational Noncommutative Algebra and Applications*, Il Ciocco, Italy, 2004. Kluwer.

[26] J.M. Selig. *Geometric Fundamentals of Robotics*. Springer, Berlin, 2005.

[27] J.F. Soechting and M. Flanders. Moving in Three-Dimensional Space: Frames of Reference, Vectors, and Coordinate Systems. *Annual Reviews on Neuroscience*, 15:167–191, 1992.

[28] D. Vernon. Enaction as a Conceptual Framework for Developmental Cognitive Robotics. *Journal of Behavioral Robotics*, 1(2):89–98, 2010.

[29] D. Vernon, G. Metta, and G. Sandini. A Survey of Artificial Cognitive Systems: Implications for the Autonomous Development of Mental Capabilities in Computational Agents. *IEEE Transactions on Evolutionary Computation*, 11(2):151–180, 2007.

[30] M.A.G. Viana. *Symmetry Studies*. Cambridge University Press, Cambridge, UK, 2008.

[31] J. Weng and I. Stockman. Autonomous Mental Development: Workshop on Development and Learning. *AI Magazine*, 23(2), 2002.

[32] H. Weyl. *Symmetry*. Princeton University Press, Princeton, NJ, 1952.