Synthesis Project: Context Recovery Through Meta-Sensing

Gary Holness Roderic Grupen
Laboratory for Perceptual Robotics

Jack Wileden
Convergent Computing Systems Laboratory

Department of Computer Science
University of Massachusetts, Amherst, MA 01003 USA
{gholness, grupen, jack}@cs.umass.edu

ABSTRACT

The problem of multi-source integration of scientific data requires detailed knowledge of the representational formats for all data sources participating in the application. An instance of such a system is a sensori-motor network. In such a system, the position of a subject is maintained as it moves about an instrumented space. Visual sensory modes extract salient regions of interest as potential subjects to be tracked from pictures taken from their viewpoint of a scene. We must ensure that feature vectors computed from such regions of one sensor refer to the same object in the world viewed by another sensor as pose can affect a sensor's response. Across a multi-modal sensor system, data can differ in any combination of resolution, feature vector representation, and mode. I examine the matching problem in a multi-modal sensori-motor network. Using a Convergent Computing approach for integrating poly-lingual systems, I treat feature vectors as object oriented types. With meta-information on a feature vector's structure, equivalences between two different representations can be found through a series of transformations. As disparate sensory modes diverge in their information content and semantics, the integration problem becomes more difficult. In stochastic domains, feedback from the environment can improve matching.

Keywords

Meta Information, System Integration, Networked Sensory-Motor systems, Multi-Agent Systems

1. INTRODUCTION

As part of the Mobile Autonomous Robot Software (MARS) research program, the Laboratory for Perceptual Robotics, has constructed an infrastructure for presenting sensors and motors in an instrumented room as network accessible services. Using the well defined interfaces in the system, one may acquire information from sensors and issue commands to motors. This infrastructure is used to build structures called Containment Units (CU's).

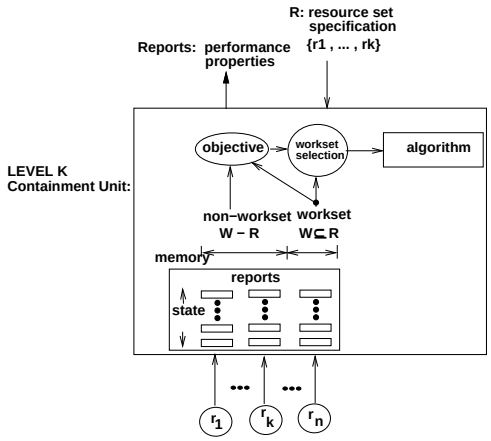

Figure 1: General Schematic of a Containment Unit

A CU (Figure 1) is a policy for maintaining a property while accomplishing some task. A CU is parameterized with motor resources through which it makes actions and sensor resources available to interact with its environment. By the nature of its closed loop control structure, a CU is able to suppress disturbances by adjusting its actions in order to maintain its property. A CU consists of sensory motor resources, memory resources, an objective function and an algorithm. The sensory motor resources produce reports describing sensory motor events. These reports, gathered in memory, form the CU's state. The algorithm has resource requirements necessary for accomplishing a task. This working set is required by the algorithm at all times in order to function properly. When parameterized with redundant resources, the containment unit may achieve fault-tolerance by changing the algorithm's working set while the algorithm is running. Such choices are made based on ratings computed

for each resource by an objective function. This results in a flexible policy for achieving a task which may adapt at run-time.

A CU is self-contained in that how its property is maintained is unknown to the application which employs it. In this sense, a CU is an object oriented construct in that it is a typed structure which encapsulates the details of its task. A CU may have other CUs in its resource set. This regular construction gives rise to a CU hierarchy. When given a set of resources, a CU is free to do with these resources as it sees fit.

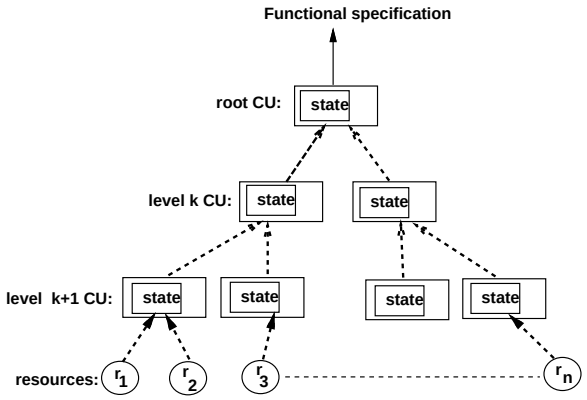

Figure 2: A Containment Unit Hierarchy

This includes initiating a hierarchy(figure 2) by instantiating a child containment unit with some resource subset. The structure of a CU hierarchy represents a functional decomposition of the task specified by the CU at the root. As indicated by a directed edge, information originates from the CU at the outbound end of an arrow and arrives as input to the CU at the inbound end of an arrow. This information describes events associated with property maintenance. A hierarchy grounds out with physical sensory-motor resources at the leaves which generate sensory-motor events. This hierarchical decomposition specifies a program of behaviors to achieve a task specified at the root.

Decisions made at each level in a CU hierarchy are based on local state information. At each level, k, this structure is robust to faulting conditions through some evaluation of information streaming in from the adjacent lower level, $k+1$. Previous research [10, 13] has focused on adaptation, hierarchical structure and typed interfaces for CU hierarchies that grounded out in physical resources that were very similar.

In this project, I consider the integration of dissimilar resources in a CU hierarchy for the purpose of cross-modal matching. Beginning with two visual modes whose feature vector representations differ, I will demonstrate how polylingual integration techniques can be used to find suitable cross-modal matches. Then, I extend the framework to deal with the general integration problem using statistical association. Such an extension also enables applications in stochastic domains by adding feedback from the environment.

1.1 Motivation, Visual Processing, and Background

The appropriate form and choice of a sensor greatly depends on the nature of the application and environment in which it will operate. Some environments lend themselves to certain types of sensory modes. For example, in an indoor environment inhabited by people, one could choose one or some combination of visual, thermal or acoustic mode. Moreover, for a particular sensory mode, the signal processing operators can respond to different environmental phenomena and return a number of feature vector representations.

In Computer Vision, it is the goal to endow machines with the ability to draw information from images necessary to describe and recognize control context. One can implement operators for various appearance based features such as color and texture or dynamic features such as optical flow. In my environment, the background is relatively stable, so I use dynamic background subtraction to extract foreground pixels from an image. By this approach, a window of observations for each pixel in the image is maintained. For each pixel, if variability across this window is sufficiently large, it is considered as a potential foreground pixel. Otherwise, with a low variability, the pixel is considered background noise and used to update a model maintained for the current background. Additionally, foreground pixels whose values failed a statistical significance test were rejected as noise.

Since more than one subject may be present in an image sequence, the next goal is to classify foreground pixels as belonging to a particular subject. I use connected components analysis in an 8-neighborhood to perform segmentation. As a pixel is labeled, a rectangle was stretched to accommodate the coordinates of the point. Iterating over an image, the resulting rectangles identify regions of interest or blobs. Spatial reasoning was employed to fold concentric and overlapping blobs. This approach is demonstrated in figure 3. The system draws rectangles around the blobs and drawn a circle at the center of mass for the set of blobs.

It is more feasible for a visual sensor to report a compact summary rather than the raw pixels of a blob. I consider appearance based features of gray scale color and texture in my system. For color I summarize with a histogram and texture I summarize with a concurrence matrix. A histogram describes the distribution of pixel values in a region. If two objects in the scene are sufficiently different, their pixels' distributions will differ. A texture is a measurement of pixel variation in a locale within a region. I consider a structural approach to texture, where the geometric relationship between a pixel and others in some neighborhood is measured.

1.2 Convergent Computing: Motivation and Background

Scientific systems such as sensori-motor networks produce and process large volumes of complex data sets. Such data is in service to some application such as tracking. The format and meaning of a feature vector are well defined as a response to an operator. The information presented by a feature vector may be queried by a number of methods to reveal some aspect of the original data. For example, in computer vision, a flow field can be consulted to estimate

Figure 3: Segmenting objects in a scene

which of a number of objects is closest. In object-oriented programming, an object collects together data along with methods which operate on the data. It has a structure and semantics defined by the public interface it exports. The data of a feature vector, when considered with queries made upon it, is exactly an object oriented type.

Modern programming languages such as Java, allow programmatic access to the structure of language elements themselves. Through such meta-information, one can craft programs which take action at run-time based upon structural information about types in a language. This is accomplished through a number of language run-time mechanisms. First, a means of reifying a language element's structure is necessary. This is done through a *reflective* API which presents a data structure which captures the necessary information. An interesting application of meta-information is in reflective middleware. In this research, components were reified as object graphs which grounded out with primitive objects. A quality of service monitor was attached at various point in the graph. If a component failed to meet a QoS goal, its implementation was modified by changing its object graph. This includes parameter tuning as well as replacing nodes in an object graph.

The added machinery to the language run-time for acquiring structural information adds overhead even to programs which do not make use of it. Another approach to reflective system was done in OpenJava. In this research, the Java language was augmented with a compile-time macro system. Since it had access to structural information, one had control over various parts of a Java class. For example, one can have multiple versions of a method body each of which are called when the method call count is high enough modulo some number.

Given a feature vector of one type, I employ structural information in aiding the transformation of the instance into another type. Structural information is not revealed directly through the public interface. This allows a feature vector to be used differently from how it was originally intended. Inter-type transformations are rarely exact. Thus, a method for measuring similarity is necessary.

It is usually the case that data from multiple sources must be gathered or integrated into an application. In the case of a CU hierarchy, the presence of a feature vector signals an event that some sensor or child CU has measured an aspect of its interaction with the environment. Part of a CU's job is to integrate feature vectors from multiple resource instances. If feature vectors are expressed as different types, with varying structure and semantics, we have exactly an instance of polylingual interoperability.

One approach to integration introduces a standardized data interchange format with its own type system. This approach requires the translation between concepts in a language's native format and the canonical language. A single common type system cannot possibly capture the syntax and semantics of many different languages. To achieve polylingual interoperability, a single type system approach would have to capture the least common denominator [12]. This means that matching expressions of two types means translating each to the common language instead of directly comparing them. Thus, meaning is lost because only native concepts expressible in the least common denominator can be translated. Lost information makes it harder to determine type compatibility. Research in polylingual interoperability [3] has defined an approach for modules written in a language to use their native type system rather than an external type system to access objects in another language. In this research, the authors define the dimensions of interoperability, namely location (naming), arbitration, communication and matching. By their approach, types are parsed into

the abstract form of a syntax tree. Given both interface and implementation for a type, they are able to generate equivalent types in another language by translation. The equivalent type is found by matching the abstract form generated from a type's native language against that of a type from a different language. An abstract syntax tree captures the structure of a program. As such it is a kind of meta information about a type. The use of structural information about types in the matching process allows comparison without strict regard to content.

In my system, meta-information about feature vectors is used in the matching process. Other research [18] defines a taxonomy and approach for matching types expressed in the ML language using signatures. They describe a prototype component browsing tool for use by someone wishing to write a new component in the programming language ML. The user inputs a query expressing their type of interest and is returned a list of equivalent types from a component library. I also draw inspiration from this framework and employ structural information from feature vectors to perform matching across disparate sensors using techniques from polylingual interoperability.

2. FEATURE TYPES

The two example feature vectors I consider in my framework are histogram and texture. A more formal definition describing their structure follows.

2.1 Histogram

Given univariate sample data, computing its frequency distribution allows us to make statements such as "the variance is a value k" about the behavior of the factor measured by the single variable. Such summaries facilitate the comparison of two factors. The variable values are drawn from a closed interval $(0, N)$. The space spanned by the variable is partitioned into a finite number k of bins of uniform width. Thus, for bin i, the sub-interval induced by the partition is a half open interval $((i-1)*k, i*k]$. Each sample contributes 1 to the count of samples whose values fall within the interval corresponding to one of the bins. This generalizes to multivariate sample data. In computer vision , *color*, is a multi-variate quantity where $N = 255$ and we have three dimensions, one for each axis of the color space. For this experiment, color images where converted to gray level in order to simplify processing by computing histograms in 1 dimension versus 3. I define a general histogram as a three tuple HIST:

$HIST = (R, k, B)$

of dimension, $d = |B|$, where

$$
\begin{aligned}
R &= \{r_0, r_1, \ldots, r_i, \ldots, r_{d-1}\} \quad \text{the range set} \\
r_i &= (min_i, max_i) \quad \text{the range along the } i^{th} \text{ dimension} \\
K &= \{k_0, k_1, \ldots, k_i, \ldots, k_{d-1}\} \\
k_i &\in \mathbb{N} \quad \text{the number of bins along dimension } i \\
B &= \{P_0, P_1, \ldots, P_i, \ldots, P_{d-1}\} \quad \text{the set of partitions} \\
P_i &= \{c_{i0}, c_{i1}, \ldots, c_{ij}, \ldots, c_{i\lceil \frac{max_i - min_i}{k_i} \rceil} - 1\} \quad \text{the} \\
&\quad \text{frequency distribution along dimension } i \text{ where} \\
c_{ij} &\in \mathbb{Z} \quad \text{frequency count for the interval} \\
z_{ij} &= \left(j * \lfloor \frac{max - min}{k_i} \rfloor, (j+1) * \lfloor \frac{max - min}{k_i} \rfloor\right] \\
&\quad \text{the } j^{th} \text{interval in partition } P_i
\end{aligned}
$$

A histogram is a feature vector which is reported for each region of interest. Additional information includes the coordinate and dimensions of the bounding box around the region summarized by the histogram. Structural information includes, the number of dimensions, the number of partitions along each dimension, and the interval endpoints for each partition.

2.2 Texture

A texture is a measurement of local patterns or variation within an image. Such a metric may take into account variations at multiple scales. The two main approaches to texture representations are structural and statistical. I have chosen a structural texture representation at a single scale. Pixels in the image plane vary locally and it is the goal of a textural representation to capture information about the pattern of local variation in the intensity surface of an image. A popular approach is a concurrence matrix. By this technique, an $n \times n$ matrix is computed for a set of spatial relationships such as $above, below, left$ and $right$. Pixels take on n unique values which need not correspond directly with pixel values. In the case where the pixel values are quantized to n levels, we have n equivalence classes.

A concurrence matrix begins with a frequency count matrix, F, which stores in position $F[i, j]$ the number of pixel pairs which satisfy some relationship. For example, $F_{\text{above}}(i, j) = 1$ means that there is a single pair in the image such that a pixel of value i is above a pixel of value j. In the case of quantized pixel intensity values, we use equivalence classes $[i]$ and $[j]$. The relationships I use, $\{above, below, left, right\}$, are isomorphic to one another. That is, $F_{above} = {F_{below}}^T$ and $F_{left} = {F_{right}}^T$. Isomorphic relations can be represented compactly as an exemplar along with the identity and transpose operators. Given a frequency count matrix, F, we compute a concurrence matrix by dividing it by $\sum_{i,j} F[i, j]$. Thus we have $C[i, j] = \frac{F[i,j]}{\sum_{i,j} F[i,j]}$. This gives us a joint probability distribution over pairs (i, j) of points satisfying the relationship. As a feature vector, I use $C_{horizontal}$, $\sum_{i,j} F_{horizontal}[i, j]$, $C_{vertical}$, and $\sum_{i,j} F_{vertical}[i, j]$ together to describe variation in four-neighborhoods within a region of interest. While I have chosen the $\{above, below, left, right\}$ relationships, this generalizes to arbitrary distances and orientations. My concurrence matrix is computed for gray level images, but generalizes to color representations. For brevity

in notation, I use $C_h, \sum_{i,j} F_h[i,j], C_v$ and $\sum_{i,j} F_h[i,j]$ in my specification. I define a texture as a three tuple TEXTURE.

$$TEXTURE = (D, R)$$

where

$$\begin{aligned} D &= n \quad \text{the matrix dimension} \\ R &= \{r_1, r_2, \ldots, r_l, \ldots, r_n\} \quad \text{relation set} \\ r_l &= \left(C_l, \sum_{i,j} F_l[i,j]\right) \quad \text{concurrence for relationship} r_l \\ C_l &= \{row_1^l, \ldots, row_i^l, \ldots, row_n^l\} \\ row_i^l &= \{p_1^l, \ldots, p_j^l, \ldots, p_n^l\} \\ 0 \leq p_j^l \leq 1 & \quad joint\ probabilities \\ \sum_{i,j} F_l[i,j] &= k \in \mathbb{N} \quad sum\ of\ frequencies \end{aligned}$$

Given this description, we may the frequency matrix by computing the product of the concurrence matrix and the frequency sum yielding
$F_l[i,j] = (\sum_{i,j} F_l[i,j]) \cdot C_l$. A texture is a feature vector which is reported for each region of interest. Additional information includes the coordinate and dimensions of the bounding box around the region. Structural information includes the number of dimensions and the endpoints which define the bounds of each cell in the matrix.

3. THE ARCHITECTURE

In my example system (figure 4), there are two fixed cameras, namely a primary camera C_D and a secondary camera C_R.

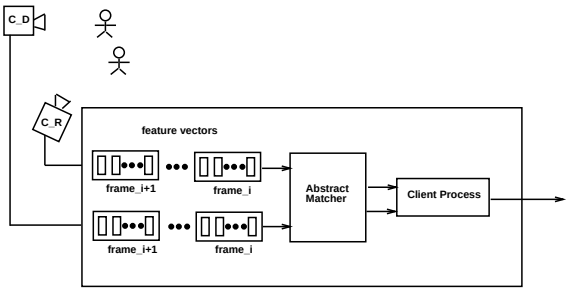

Figure 4: System Architecture

From each frame of image data, a histogram is computed from regions of interest from camera C_D and a texture is computed from camera C_R. A corresponding feature from C_R is found for a relevant feature from C_D. The relevant feature and its match are then input to a client process which operates on the corresponding pair. In a tracking example, the client process computes a Cartesian coordinate by triangulation using the heading to the corresponding feature vectors. In an example (figure 5) we have feature vectors reported for four regions of interest $\{r_1^D, r_2^D, r_3^D, r_4^D\}$ and $\{r_1^R, r_2^R, r_3^R, r_4^R\}$ from the primary and secondary cameras respectively. Meta information computed from these feature vectors are $M^D = \{m_1^D, m_2^D, m_3^D, m_4^D\}$ and $M^R = \{m_1^R, m_2^R, m_3^R, m_4^R\}$ respectively. Suppose region r_2^D at heading $\theta_1 = 40^\circ$ were interesting and the matcher queried M^R for a match to m_2^D and found m_3^R. Since m_3^R is associated with r_3^R at $\theta_2 = 32^\circ$, the matcher submits r_2^D and r_3^R to the triangulation algorithm which computes a position vector $\vec{x}$ from the pair of headings (θ_1, θ_2).

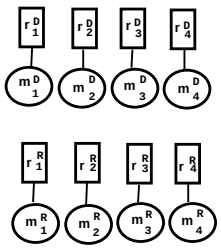

Figure 5: Meta Information for feature vectors

3.1 Matching Framework

Given

$$\begin{aligned} \mathbb{U} &\quad \text{set of populations M} \\ M', M \in \mathbb{U} & \\ q \in M' &\quad \text{a querry} \\ p \in \mathbb{P} &\quad \text{a predicate} \end{aligned}$$

a set of transformations ...

$$\begin{aligned} T^{M'} &= \{T_1, T_2, \ldots, T_i, \ldots, T_q\} \\ T^{M} &= \{T_1, T_2, \ldots, T_j, \ldots, T_m\} \end{aligned}$$

a set of relations ...

$$R = \{R_1, R_2, \ldots, R_k, \ldots, R_n\}$$

and a set of predicates ...

$$\mathbb{P} = \{(T^{M'})^* \, R \, (T^{M})^*\}$$

Define

$$\text{Match}(q, p, M): \qquad M' \times \mathbb{P} \times \mathbb{U} \longmapsto M \tag{1}$$

where

$$Match \equiv \exists_{M \in \mathbb{U}} \exists_{m \in M, q, k} \; p(q, m)$$

Given a set of transformations $T^{M'}$ on a query type, a set of transformations T^M on a population type, and a set of relations R of arity two, we define a set of predicates $\mathbb{P}$ consisting of zero or more compositions of transformations on a query type and population type under one of a number of relations. Each member $p \in \mathbb{P}$ is of arity two. $Match$ finds the member of the population which best relates a query to a member of the population each under zero or more transformations.

The possible transformations fall into two major categories, volatile and non-volatile. A volatile transformation modifies the structure of the element on which it is performed while a non-volatile transformation is structure preserving.

Beginning with the non-volatile transformations, we define T_{Hpic} to be a transformation which creates a pseudo picture from a histogram. We define a pseudo picture as an image representation of the information captured by a feature vector. Because a histogram throws away position information of pixels in the image, the pseudo picture resulting from this

transformation is not intended for display. It is useful, however, for recomputing the distributions of varying bin width. Under the assumption that the points recorded by the frequency count in a particular bin are drawn from a normal distribution within its interval endpoint, we sample from a normal distribution and weight the points along the interval. While this approach is prone to error, it is our best estimate of the original picture. As the central limit theorem tells us, the number of points affects the accuracy of this estimate. A bin with many more sample points will better approximate a normal distribution. A high frequency bin contains more mass of the distribution and thus is more relevant in the summary of the original image region. While a bin with few sample points is a coarser approximation, it contains a small amount of the distribution's mass and is less relevant. Thus, a pseudo picture is more accurate where it is important.

Likewise, we define T_{Tpic} to be a transformation which creates a pseudo picture from a texture. Since a texture captures some relationship between pixel pairs in an image, a pseudo picture resulting from this transformation can maintain the pixel relationships, but not necessarily the original image. For example, suppose we have an $n \times n$ co-occurrence matrix where each position $C[i,j] = \frac{1}{n^2}$ for the *above* relationship. This relationship is preserved by constructing a pseudo picture of size $2 \times n^2$ to represent the single instance of a pixel of value $[i]$ above a pixel of value $[j]$ for all i and j. Given that some relationships are isomorphic such as $above = below^T$, preserving such spatial relationships in a pseudo picture are also useful for transformation to a texture which captures an isomorphic relationship.

The volatile transformation are those which change a feature vector's structure. We define T_{Hbin+} to be a transformation creates a histogram with $k+1$ bins from a histogram with k bins. We also define T_{Hbin-} to be a transformation which creates a histogram with $k-1$ bins from a histogram with k bins. Likewise, we define $T_{Tclass+}$ to be a transformation which creates a texture with $k+1$ equivalence classes from a texture with k classes. We also define $T_{Tclass-}$ to be a transformation which creates a texture with $k-1$ classes from a texture with k classes.

We define an interoperability transform $T_{T \to H}$ which produces an instance of a histogram given an instance of a texture. This transform is one-way, meaning that it does not have an inverse because a histogram discards all spatial information while a texture maintains some of it. Given an instance of a texture, we can always throw information away to get a histogram, but once information is discarded, there is no easy way to recover the appropriate texture representation. For example, suppose we are given a histogram with k bins computed from an $m \times n$ pixel region of interest and we wish to transform it into a texture with k equivalence classes for some relationship. Since, through structural transformations, we can make the number of bins equal to the number of equivalence classes, we consider them equal. The histogram records the fequency of mn pixels into k bins. For each pixel, p, in bin i there are k ways that a pixel in bin i satisfies some relationship with a pixel in bin j. Over mn total pixels, this means we have k^{mn} total ways to choose to associate pixels for some relationship.

If there are $m \times n$ pixels in the region of interest from which the histogram was computed, we know there are on order $O(mn)$ possible pairwise spatial relationships associated with a texture representation. Given that one of these synthesized textures is the appropriate one, we have $Pr\{\text{synthesizing appropriate texture}\} = \frac{1}{mnk^{mn}}$. This probability is exponentially small in the number of pixels in the region of interest used to compute the feature vector. Thus, synthesizing the appropriate texture from a histogram is very unlikely to occur. This highly unlikely event is attributed to the fact that information discarded by the histogram, namely the pixel pairings and spatial relationship, must me guessed for each pixel. The chance of guessing incorrectly multiplies over all $m \times n$ pixels. The probability of synthesizing an appropriate feature vector is minimized by reducing the amount of guessing. This is achieved by limiting interoperability transforms to those from feature vectors with more information onto those with less information.

The possible relations under which comparisons on transformed elements are made are equality, successor, predecessor, min, max. We define a distance functions $H_{dist} : H \times H \to \mathbb{R}$ and $Tdist : T \times T \to \mathbb{R}$ which measure the distance between two histograms and textures respectively. The distance between two histograms is the difference of their mean while the distance between two textures is the difference between their entropy.

We define a relation R_{eq} of arity two which asserts *true* if its operands are equal. This is similar to the exact match in the Zaremski and Wing framework [18]. That is ...

$$fR_{eq}f' \leftrightarrow dist(f,f') \leq \epsilon$$

Where $\epsilon \in \mathbb{R}^+$ is a threshold and f and f' are features of the same type.

We define a relation R_{suc} of arity two which asserts *true* if some feature instance appears after another in an ordering. That is...

$$fR_{suc}f' \leftrightarrow \exists f' \neg(fR_{eq}f') \wedge dist(f,f') \leq dist(f,f").$$

We define a relation R_{pre} of arity two which asserts *true* if some feature appears before another in some ordering. That is ...

$$fR_{pre}f' \leftrightarrow \exists f' \neg(fR_{eq}f') \wedge f'R_{suc}f$$

The relations R_{suc} and R_{pre} are similar to partial ordering matches in the Zeremski and Wing framework [18]. We define a relation R_{max} of arity two which asserts *true* if some feature is of maximum distance some ordering. That is ...

$$fR_{max}f' \leftrightarrow \forall f" dist(f,f") \leq dist(f,f')$$

We define a relation R_{min} of arity two which asserts $false$ if some feature is of minimum distance in some ordering. That is ...

$$fR_{min}f' \leftrightarrow \forall f" dist(f,f") \geq dist(f,f')$$

3.2 Convergent Computing Experiment

The matching framework implementation was a mixed system where the image processing components where implemented in the C++ programming language and the transformations were implemented in the Java programming language. Structural information on feature vectors was implemented using an iterface which returned meta-information about the classes which implemented them. This approach was taken because Java does not allow reflection on package and privatley scoped instance data.

Image processing was performed on grey-level images converted from normalized RGB. Blob data returned from the image processing system is passed to a Java program which constructs feature vector objects for them. It is important to note that only the feature vector information is available as the original blob pixel data is discarded. This experiment involve two cameras (A and B). For camera A returning n blobs $\{r_1^A, \ldots, r_n^A\}$ represented as histogram with k bins and camera B returning m blobs $\{r_1^B, \ldots, r_m^B\}$ represented as texture with j equivalence classes, we have the following closest match semantics without loss of generality for $k > j$ ...

Closest-Match
$\mathbb{U} = \{\{r_1^A, \ldots, r_n^A\}, \{r_1^B, \ldots, r_m^B\}\}$
$M = \{r_1^B, \ldots, r_m^B\}$
$M' = \{r_1^A, \ldots, r_n^A\}$
$T^M = \{T_{T_{class+}}, T_{T_{class-}}, T_{T_{pic}}\}$
$R = \{R_{min}\}$
$T_{T \to H} \doteq Histogram(T_{T_{pic}})$
$T^{M'} = \{T_{H_{bin+}}, T_{H_{bin-}}, T_{H_{pic}}\}$

for each $q \in M$
  $T \leftarrow T_{T_{class-}}$
  for $i = 1$ to $k - j$
    $T \leftarrow T \circ T$
  $T \leftarrow T \circ T_{T \to H}$
  for each $m \in M'$
    if $T(q) R_{min} I(m)$
      then return m

The system implementing Closes-Match semantics is pictured in figure 6. The objects placed in the scene are a clear plastic bag of colored toy blocks placed on top of a tool chest, an inflateable spaceman and swivel chair rolled to the middle of the floor, a beach ball placed on a table, a box containing a pair of toy boxing gloves placed on a table and a plastic container full of metal parts placed on the floor. The environment is a typical cluttered laboratory space. Two cameras were placed at vantage points roughly 45 degrees apart. Camera A was 12 feet from the center of the scene and camera B was 15 feet from the center of the scene. In figure 6 camera A has detected 4 blobs and camera B has detected 5 blobs. This data was generated from a recorded image sequence as the compute hardware used for this experiment was not adequate for on-line visual processing. It is important to note that, the number of blobs detected by each sensor from frame to frame in a particular image sequence is not known a-priori. A sample result produced by this system appears as follows. The ordering of blobs is zero based from left to right. For each blob r_i^B there is a match to some blob r_j^A. A $< r_i^B, r_j^A >$ pair is a match. The matches for a sample result from this experiment appear as follows...

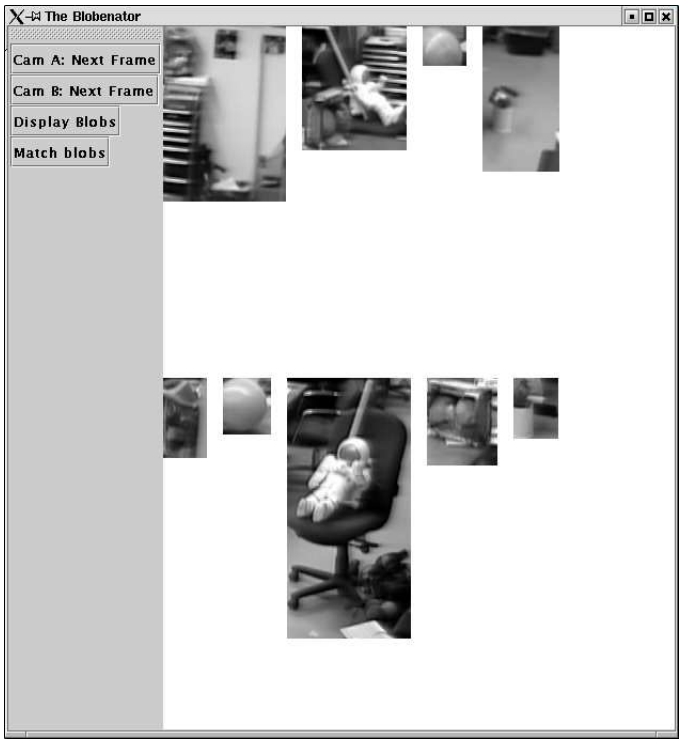

Figure 6: matching blobs: camera A (top) and camera B (bottom) with zero based ordering from left to right

```
TrTextureToHistogram: doTransform: starting...
TrTextureToHistogram: doTransform: ...done
match camB[0] to camA[0] : 49.34060007630613
match camB[0] to camA[1] : 8.47208696203434
match camB[0] to camA[2] : 59.95461688485676
match camB[0] to camA[3] : 34.22528804877324

TrTextureToHistogram: doTransform: starting...
TrTextureToHistogram: doTransform: ...done
match camB[1] to camA[0] : 2.844418500933074
match camB[1] to camA[1] : 43.73057448844396
match camB[1] to camA[2] : 7.721757285324713
match camB[1] to camA[3] : 17.819637815960803

TrTextureToHistogram: doTransform: starting...
TrTextureToHistogram: doTransform: ...done
match camB[2]  to camA[0] : 54.19166292364983
match camB[2]  to camA[1] : 13.441139648238519
match camB[2]  to camA[2] : 64.80991824473445
match camB[2]  to camA[3] : 39.154881801955526

TrTextureToHistogram: doTransform: starting...
TrTextureToHistogram: doTransform: ...done
match camB[3] to camA[0] : 31.59334572072953
match camB[3] to camA[1] : 9.022866199485122
match camB[3] to camA[2] : 42.30964717680223
match camB[3] to camA[3] : 16.704100891018527
```

```
TrTextureToHistogram: doTransform: starting...
TrTextureToHistogram: doTransform: ...done
match camB[4] to camA[0] : 43.158657513539296
match camB[4] to camA[1] : 2.432503059580071
match camB[4] to camA[2] : 53.81102196882243
match camB[4] to camA[3] : 28.08185470587688
```

As can be seen, blob $r_0^B, r_1^B, r_2^B, r_3^B$, and r_4^B are matched with blob $r_1^A, r_0^A, r_1^A, r_1^A$, and r_1^A respectively. This match is incorrect for the $< r_1^B, r_0^A >$ and $< r_4^B, r_1^A >$ pairs. In this experiment, for blob r_0^B we have a special case where the object in the environment (bag of toys) was similar (ground truth) to the r_1^A which contains the toy boxing gloves and plastic spaceman. When viewed from a different pose by camera A, the bag of toys looked significantly different from its ground truth match in camera B due to variation in its surface over a 45 degree rotation. The toy boxing gloves had similar color to many of the toys in the bag of toys. Therefore the match is considered correct. For the incorrect matches $< r_1^B, r_0^A >$ and $< r_4^B, r_1^A >$, if we consider the sucessor to the closest match, we get matches $< r_1^B, r_2^A >$ and $< r_4^B, r_3^A >$. This coincides with ground truth. Thus, we are able to match relatively well across two disparate feature representations.

4. MATCH ACCURACY

The accuracy of the match algorithm is greatly affected by the interoperability transform since this structure preserving representation is synthesized by sampling. Thus, it is necessary to test how the accuracy of the $T_{T_{pic}}$ transform. This was done in simulation as there was no reasonable way to control the size of the blobs observed by each camera a-priori. In the simulation, we have q square blob of side length varying in 5 pixel increments from 5 to 100. Let X be a random variable which measures a pixel's value. In these experiments, $X \sim Norm(127, 127)$. For each simulated blob, a texture with the number of equivalence classes varying in increments of 4 from 4 to 128 was computed. A $T_{T_{pic}}$ transform was performed on each texture and the resulting pseudo-picture was compared with the original picture. This resulted in 640 different tests. Each experiment was run 50 times to ensure statistically significant results. The testing procedure is as follows....

- generate square blob with pixel values sampled from some distribution
- compute texture
- compute pseudo-picture by applying transform $T_{T_{pic}}$
- match each pixel in pseudo-picture to pixel in square blob within a percentage error

These results appear in figure 7. Since there were so many data points, the confidence intervals were visualized by plotting the 95% and 5% values along with the sample values as points. With this, the graph resembles a series of 3 points stacked up along the z direction. A wider confidence interval corresponds to a taller stack of points while a narrower confidence interval corresponds to a shorter stack of points. The confidence interval decreases with increasing side length. The confidence interval also increases with the number of equivalence classes. This means that match accuraccy increases with the number of pixels and decreases with the number of equivalence classes. Generally, the worst performance was just above a 96% match rate. This excellent performance was somewhat surprizing. Based on the number of errors made by the Closest-Match algorithm on real images (roughly 40% error rate), the Normal assumption for simulated blob pixels must be incorrect. A more correct assumption is a mixed distribution. Estimating this mixture and the parameters of the consituent distributions from image sequence data is an obvious extension.

5. CONCLUSIONS AND FUTURE WORK

This investigation was an examination of how meta-information can be used to achieve integration of disparate feature vectors. It began with an examination of various meta systems and their applications. The one constant across all examples was that access to structural information eased integration because it facilitated the ability to reason about structural equivalence. While structure alone was very useful, some information about semantics was necessary. Inclusion of semantics was done by hand through formal reasoning about the feature types. What is necessary is a run-time facility which gives programmatic access to semantic information. It is not clear to the author how feasible it is to have run-time support that reifies semantics.

Reliable synthesis across two feature is improved if synthesis is performed from a representation with higher information content to one with lower information content. It is my belief that the information content constraint also carries back to programming languages.

Techniques from programming languages are directly applicable in the feature domain by finding a series of transformations on feature vector types which preserve structure without strict regard to content in the same way that an abstract syntax tree (AST) captures the structure of an expression in a programming language. Unlike programming languages, the physical domain is rarely so exact. This was most apparent with closest-match semantics where, in many cases, it was observed that the notion of closest-match was not always the best match. The world is stochastic while programming languages are exact. Because of this, feedback from the environment is neccessary if some notion of best match is to be achieved. For future work, I propose treatment of best match as a sequential descision process. As such, using the framework of Reinforcement Learning [11, 14] we can solve the best match problem. With this treatment, we rely on the client process (figure 4) for definition of state and the reward signal. Since the application domain is tracking, the state is the coordinate of the subject being tracked and some encoding of the feature vectors themselves. The action set is the set of possible matches starting with closest-match semantics on through each of its successors. Once a match and new position of the tracking target have been computed, an adequate reward signal is the change in entropy of the tracking signal due to the new tracking information. With this, the system can learn a policy for matching so as to maintain a low entropy tracking signal. With this reward feedback from the environment, the tracking agent is able to learn under what conditions it

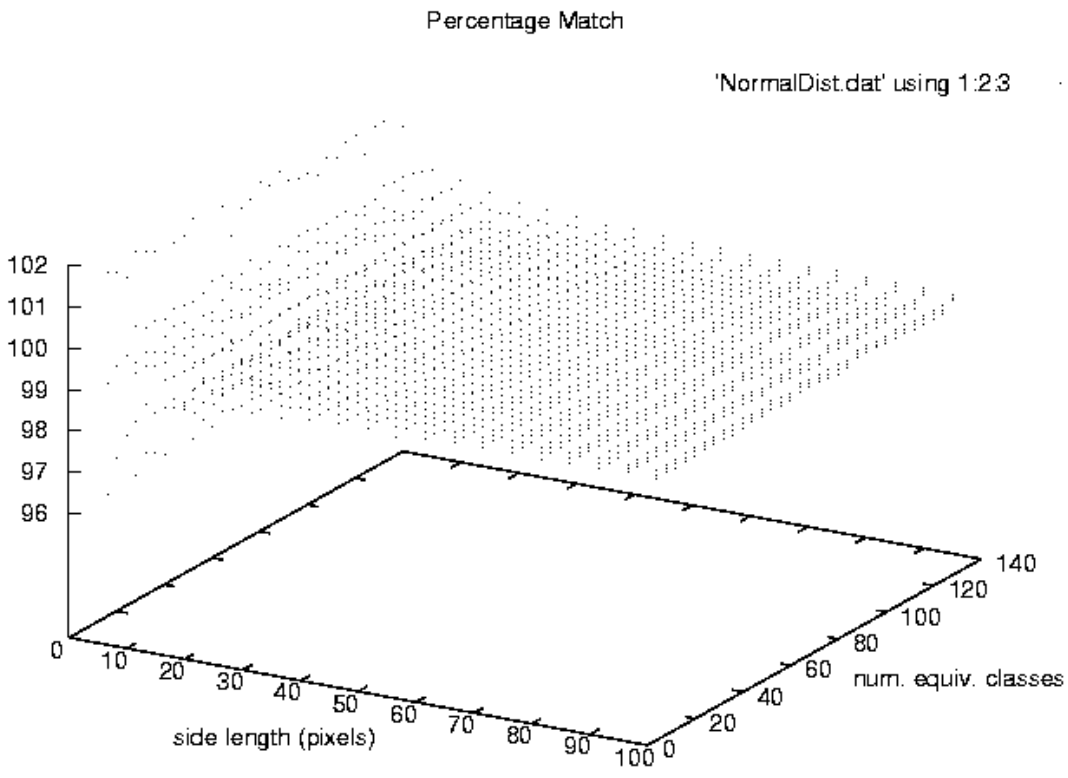

Figure 7: Match performance

is appropriate to choose best-match semantics or one of its sucessors.

6. REFERENCES

[1] Joao Barreto, Jorge Batista, and Helder Araujo. Solutions for visual control of motion: Active tracking applications. In *Proceedings of the 8th IEEE Mediterranean Conference on Control and Automation (MED2000)*, Rio Patras, Greece, 2000. IEEE.

[2] D. Barrett. Polylingual systems: An approach to seamless interoperability, 1998.

[3] Daniel J. Barrett, Alan Kaplan, and Jack C. Wileden. Automated support for seamless interoperability in polylingual software systems. In *Proceedings of the Fourth ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 147–155, New York, 1996. ACM Press.

[4] Gordon S. Blair, G. Coulson, P. Robin, and M. Papathomas. An architecture for next generation middleware. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, London, 1998. Springer-Verlag.

[5] Chris Gaskett, Luke Fletcher, and Alexander Zelinsky. Reinforcement learning for a vision based mobile robot.

[6] Chris Gaskett, Luke Fletcher, and Alexander Zelinsky. Reinforcement learning for a visual servoing of a mobile robot.

[7] J. Gosling, B. Joy, and G. Steele. The java language specification, 1997.

[8] Gregory Hager. Tutorial tt3: A tutorial on visual servo control.

[9] D. Hershberger, R. Burridge, D. Kortenkamp, and R. Simmons. Distributed visual servoing with a roving eye, 2000.

[10] G. Holness, D. Karuppiah, S. Uppala, R. Grupen, and S. Ravela. A service paradigm for reconfigurable agents. In *2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS. ACM.*, Montreal, Canada, May 2001.

[11] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[12] Alan Kaplan, John Ridgway, and Jack Wileden. Why idls are not ideal. In *Proceedings of the 9th IEEE International Workshop on Software Specification and Design*, Ise-Shima, Japan, 1998. IEEE.

[13] D. Karuppiah, P. Deegan, G. Holness, Z. Zhu, B. Lerner, R. Grupen, and E. Riseman. Software mode changes for continuous motion tracking. In *International Workshop on Self Adaptive Software*, Oxford, England, April 2000.

[14] S. Keerthi and B. Ravindran. A tutorial survey of reinforcement learning, 1995.

[15] Dimitri Konstantas. Object-oriented interoperability. In Oscar M. Nierstrasz, editor, *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, volume 707, pages 80–102, Berlin, Heidelberg, New York, Tokyo, 1993. Springer-Verlag.

[16] Bjarne Stroustrup. The c++ programming language.

[17] Michiaki Tatsubori, Shigeru Chiba, Marc-Oliver Killijian, and Kozo Itano. OpenJava: A class-based macro system for Java. In Walter Cazzola, Robert J. Stroud, and Francesco Tisato, editors, *Reflection and Software Engineering*, LNCS 1826, pages 119–135. Springer-Verlag, July 2000.

[18] Amy Moormann Zaremski and Jeannette M. Wing. Signature matching: A key to reuse. In David Notkin, editor, *Proceedings of theFirstACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 182–190. ACM Press, 1993.