Automatic Detection and Response to Environmental Change*

Scott Lenser and Maneula Veloso

Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
{*slenser,mmv*}*@cs.cmu.edu*

Abstract

Robots typically have many sensors which are underutilized. This is usually because no simple mathematical models of the sensors have been developed or the sensors are too noisy to use techniques which require simple noise models. We propose to use these underutilized sensors to determine the state of the environment in which the robot is operating. Being able to identify the state of the environment allows the robot to adapt to current operating conditions and the actions of other agents. Adapting to current operating conditions makes the robot robust to changes in the environment by constantly adapting to the current conditions. This is useful for adapting to different lighting conditions or different flooring conditions amongst many other possible desirable adaptations. The strategy we propose for utilizing these sensors is to group sensor readings into statistical probability distributions and then compare the probability distributions to detect repeated states of the environment.

1 Introduction

It is important that robots be able to identify the state of the environment in which they are operating. Without this ability, the robot is unable to use information that has been acquired previously about the environment (through instruction or learning) to improve its behavior. While the robot may be able to re-adapt to the revisited environmental state, this will almost certainly take longer than identifying that the state has re-occurred. Identifying the current environmental state (including the state of other agents) allows the robot to adapt to the behaviors of other agents by recognizing repetition in their actions.

*This research was sponsored by Grants Nos. DABT63-99-1-0013, F30602-98-2-0135 and F30602-97-2-0250. The information in this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

In the trivial case, the current state of the environment can be determined from a single observation. This case is not very interesting and is handled well by current techniques. In many systems, however, a single observation contains very little information about the current state of the environment. Even all of the observations at a single point in time are likely to be insufficient to determine the current state. This is usually due to each individual sensor reading providing only a small part of the information and/or being subject to large levels of noise. For instance, a single pixel from a camera image (or range from laser range finder/sonar) gives very little information about the state of the world. A complete image (or range scan) provides a treasure trove of information if it can only be extracted. Both of these problems can be overcome by aggregating the individual sensor readings into probability distributions (over time or space). These probability distributions must then be compared to detect similar distributions if repeated states are to be identified. Since the robot should generalize over nearby states when possible, it is not enough to simply use statistical tests to determine similarity. This is due to the nature of statistical tests in merely determining different/same rather than giving an indication of the degree of difference thus preventing the robot from identifying nearby states.

We applied this thinking to the problem of adapting to changing lighting conditions. Many practical color robotic vision systems rely on consistent pixel readings that allow colors to be segmented without reference to the surrounding pixel context. Since actual pixel readings vary with lighting conditions and the separation between colors is often small, these systems are highly dependent on consistent lighting conditions. Changing the lighting conditions involves recalibrating the vision system for the new conditions. We propose a solution to this problem which automatically detects the current lighting conditions and switches to a corresponding human or machine

supplied calibration.

We begin by describing related work in Section 2. In Section 3, we describe the algorithm for segmenting the data into different states. In Section 4, we describe the algorithm we use for converting segmented data into class labels that identify the current state of the environment. We describe how this knowledge is used to improve the performance of robotic vision by adapting to the current environmental state in Section 5. We also describe testing procedures and results in this section. We finish with a discussion of future work and our conclusions in Sections 6 and 7, respectively.

2 Related Work

There is a large body of work along many different lines which is related to our work. This work varies greatly and has various amounts of overlap with the work we have done. We are not aware of any work, however, which combines all of the elements we consider. In particular, we are not aware of any systems that are based on probability distributions, use labelled and unlabelled data, identify the state of a system, are part of a closed loop system, and are on-line.

Much work has been done on various components intended to be used to perform useful tasks as part of an overall system. Classification systems have been used in artificial intelligence for a long time. The problem of identifying the state of the environment can be viewed as a classification problem of identify a class corresponding to the state of the current environment based on labelled examples. More recently, co-training has been used to improve the performance of classification systems by using the structure of unlabelled data to aid in the generalization of the labelled data to the full data set. Co-training in turn is similar to clustering which also has a lengthy history. The problem can be viewed as clustering probability distributions although the problem of assigning labels to the generated clusters is not addressed by clustering systems. A separate line of work has focussed on prediction of time series. The problem of adapting predictions to the system has been investigated, although in this case the adaptation is smooth and not discontinuous. All of these systems share the property that they are not closed loop systems and do not use the resulting classification to do anything useful. These systems also assume the data has already been broken up into probability distributions and only a small number of the usual techniques can use probability distributions as primitive data items in the system. Some of these techniques also assume that there is a one to one function state clusters to labels, whereas we only assume a many to one mapping from state clusters to labels. A final drawback to most of these techniques is that new classes with new labelled examples cannot be added on-line.

Assuming a division of the input data into probability distributions, some classification systems can be used to generate class labels corresponding to the current state of the environment. We are not aware of attempts to use any of these classification systems to classify probability distributions, however, and it is unclear whether most of them will work for this task in practice. One simple technique for classification would be to use k nearest neighbors with a suitable distance metric between probability distributions (see Section 3.3 for one possibility). Other work has been done with co-training which might also be applicable here, although most algorithms are off-line only. Co-training is the use of unlabelled data along with labelled data to improve the performance of a learner. Dempster, et.al. [6] introduced the well known EM approach to using unlabelled data. See Blum and Mitchell [1] for more recent work on co-training. A more generic starting point is clustering algorithms. Almost all of these are off-line algorithms and hence not applicable here. Particularly fast clustering has the potential to be used on-line, see Cutting [4] for an example. All clustering algorithms leave undetermined the method of generating the class labels from the resulting clusters.

Adapting to changing state has been investigated in the context of time series prediction. One popular approach is to model the time series data using an autoregressive model. The state of the system is then the parameters of the autoregressive model. Adaptation is then done by either allowing the parameters to drift over time by using a Kalman filter or using a HMM to model the state of the system. In the case of the Kalman filter approaches, the state of the system is never identified as being a repeated state which makes adaptation slow. In the case of the HMM approaches, the number of states and a HMM are given which makes the techniques inapplicable in the common case where no model is available.

State grouping has been investigated in both reinforcement learning and neural network learning. Chrisman developed a technique for splitting states in reinforcement learning based on perceptual distinctions [3]. Chrisman focussed on distinguishing states with similar perceptions whereas our focus in on distinguishing different states using noisy perceptions. There has also been work on state grouping in finite state machine induction using neural networks. Das and Mozer [5] used Gaussian clustering of hidden states of recurrent neural networks to identify distinct states. Pawelzik et.al. [10] used annealing to

cluster hidden states of recurrent neural networks. All of these techniques are off-line techniques that slowly converge to the right answer.

The most closely related work is in the area of switching state-space models. Ghahramani and Hinton [7] provide a nice framework and overview of approaches as well as discussing a learning technique for these models. Kohlmorgen and Lemm [8] have developed an on-line algorithm for detecting state changes in sensor data. They focus on learning the model of the state-space whereas we are interested in both learning the model and applying it to robotics problems. The problem of adapting to conditions has also been investigated in the control of non-linear and jump linear plants by Cacciatore and Nowlan [2]. They use a recurrent neural network to learn a gating function which selects among several base neural networks. As with most neural networks, training is slow.

3 Algorithm

In order to understand the on-line algorithm, it helps to start by considering the off-line version. The basic problem is to identify the current lighting conditions from data summarizing what the camera is seeing. It is desirable to use a small amount of information to summarize the overall lighting of the scene captured by the camera to reduce memory and computation requirements. Simply using the average luminance (or brightness) of the scene is sufficient for separation and economical in representation. Of course, the average luminance of a scene depends highly on what is being looked at. Therefore, instead of relying on a single measure of average luminance, a distribution of luminance values over the recent past is considered. The basic problem then becomes to segment the time series of average luminance values into distinct regimes (or regions) which have similar distributions of average luminance measurements.

Having decided to look for similar distributions of measurements, it is now necessary to have a way to determine whether two distributions are in fact the same distribution and how similar they are. Since the shape and form of the distribution is unknown, a non-parametric distance metric is used. The particular distance metric used is not very important as long as it is indicative of the difference between two distributions. Note that statistical difference tests such as the Kuiper test [9] or Kolmogrov-Smirnov are inappropriate as they measure the probability of difference between two distributions but not the distance between them.

Now that the form of the input data has been determined, it is necessary to consider the form of the output and the algorithm to use to produce it. For simplicity, the input data is split into equal size windows of size w. The window size is a parameter of the system and affects the latency and robustness of the resulting detection. Larger window sizes are more robust but have higher latency. The basic idea for representing the output is to avoid making binary decisions until the last possible moment. This is done by representing the division of the input space by a binary tree where each leaf represents a region of the input space. Regions which are similar are stored close to each other in the tree and have common ancestor nodes. Each internal node stores the distance between its two children (as reported by the statistical test). Internal nodes which have internal nodes as children use the union of all the data in the leaves when performing comparisons. The resulting tree can be used to determine the number of modes of the data by applying a threshold split criterion to the tree. Alternatively, if the number of different modes is known, the tree can be used to select a segmentation of the data into the different modes. Thus the tree can be used for determining the location and number of modes of the data and, as will be shown later, determining the current mode of the system and relating it to available calibrations.

Procedure Segment(*input_space*)
  Split *input_space* into n non-overlapping windows
    of size w.
  Create a leaf node for each window.
  Initialize the set S to contain all of the leaf nodes.
  while($|S| > 1$)
    Calculate distance(dist()) between all pairs
      of elements of S.
    Choose p,q such that
      $\text{dist}(p, q) <= \text{dist}(i, j) \forall i, j \in S$.
    Create new internal node r with
      p and q as children.
    $S = S - \{p, q\} + \{r\}$.

Figure 1: *Off-line Segmentation of Data*

3.1 Off-line Segmentation

The tree is easy to construct with a simple but slow off-line algorithm. The algorithm starts by dividing the input space into n non-overlapping windows of size w. Each window is compared to every other window to get a distance value. The two most similar windows are joined together by creating a new internal node with the two windows as children. This new internal node is treated as a mega-window which replaces the two original windows. This leaves $n - 1$ windows. The process is repeated until all of the windows have been joined. The pseudo-code for this

```
Procedure Insert(T:tree, w:window)
    let n ← root(T).
    let S ← {n}.
    let done ← false.
    while(¬ done)
        let R ← ∅.
        foreach s ∈ S
            let c1,c2 ← children(s).
            let d1 ← dist(c1,c2).
            let d2 ← dist(c1,w).
            let d3 ← dist(w,c2).
            if d2 < d1 ∨ d3 < d1
                if d2 < d1
                    R ← R + {c1}.
                if d3 < d1
                    R ← R + {c2}.
            else R ← R + {s}.
        done ← (R=S).
        S ← R.
    Choose b ∈ S that minimizes dist(b,w).
    Create a new node o which has as children b and w.
    Replace b with o in the tree.
    Update the similarity measurements of
        all ancestors of o.
```

Figure 2: *On-line Segmentation of Data*

algorithm is shown in Figure 1.

3.2 On-line Segmentation

The main obstacle to building the tree on-line is to find an efficient way to insert a new window into the growing tree. The first strategy tried was to simply start at the root and follow the branch with which the new window was most similar. This was done recursively until a leaf node was reached or the two children of the node were more similar to each other than to the node being inserted. This algorithm works reasonably but sometimes fails to find the best place in the tree to insert the new window. This usually happens because the top levels of the tree contain mixtures of very different modes and the new window tends to look very different from all of them. This inherently has a lot of noise compared to the distance signal produced by the actual similarity. A more robust algorithm has been developed which improves on the naive algorithm by trying a few different branches of the tree to find the best place to insert the new window. The pseudo-code for the insertion is shown in Figure 2.

In practice this algorithm seems to only have to consider about 10% of the nodes, since the nodes in the tree tend to have very similar children after traversing a very short depth down the tree.

```
Procedure PropagateClassesUp(n:node)
    if leaf(n)
        for c ← 0 to num_classes − 1
            n.ClassCnts[c] ←
                count(n.Examples.hand_label=c).
        n.class ← argmax_c(n.class_cnts[c]).
    else
        foreach child of n.Children
            PropogateClassesUp(child).
        n.ClassCnts ← 0.
        foreach child of n.Children
            n.ClassCnts ← n.ClassCnts +
                child.ClassCnts.
        n.class ← argmax_c(n.class_cnts[c]).

Procedure PropagateClassesDown(n:node,last:class)
    if n.class =UnknownClass
        n.class ← last.
    foreach child of n.Children
        PropogateClassesDown(child,n.class).

Procedure DetermineClasses(T:tree)
    Clear class label of all nodes in T.
    let n ← root(T).
    PropagateClassesUp(n).
    PropagateClassesDown(n,UnknownClass).
```

Figure 3: *Labelling Classes*

3.3 Distance Metric

The distance metric used is a very simple measure of the average distance the data points from one distribution would need to be moved in order to match the data points of a second distribution. A cumulative probability distribution is formed for both distributions, call them $f(x)$ and $g(x)$. Let $f'(p)$ and $g'(p)$ by the inverse of $f(x)$ and $g(x)$ respectively. Then the average distance points must be moved to make one distribution match the other is given by

$$\int_{p=0}^{1} |f'(p) - q'(p)| dp$$

4 Labelling Classes

Next, the class of the current window must be determined so that the robot can decide what the current state of the environment is. This is done by labelling the class of every node in the tree from scratch each time a window/node is added to the tree (see Figure 3). The algorithm starts by labelling each leaf with any labelled examples with the most common class label in that leaf. Note that the may be many very different states that correspond to the same label. These labels are then propagated up the tree

assigning each node the most common label found in its subtree. The remaining unlabelled subtrees are labelled with the label of their parent. The class of the most recent window is taken as the class of the current state of the environment.

5 Application

The algorithm was applied to the task of automatically selecting vision thresholds by automatically identifying the current state of the lighting and using the matching thresholds. Robots are usually limited to working in a specific lighting condition for which they are trained. This occurs because thresholds are often used in robotics because they are fast, leaving more processing available for other non-vision tasks. By automatically selecting amongst several pre-trained thresholds, a robot can better adapt to the current lighting conditions of the environment it finds itself in. Rather than have to find a set of thresholds that generalize across all lighting conditions, by applying the state identification technique described above, the robot can have several thresholds each of which generalizes over a much smaller region of the state space of all possible lighting conditions. Since it is possible to find thresholds which generalize over reasonable amounts of the lighting state space, this allows the robot to adapt to a large variety of situations. The resulting more specialized thresholds also give better performance than the more general thresholds at any given lighting level.

We measured the ability of a robot to correctly identify colors in an image under different lighting conditions using both the algorithm described above and simply using one set of thresholds throughout.

5.1 Test Methodology

The robot was placed in front of a set of objects with easy to confuse colors (red, pink, orange, and yellow) which we are interested in segmenting. The robot was started with lighting conditions matching the thresholds used to allow it to auto-center the camera on the objects. The robot recorded the colors it saw every fourth frame to a log file along with a few raw images from the camera. The lighting conditions were changed between three different brightness levels on a schedule timed by a stopwatch. The baseline (no adaptation, always use bright thresholds) and the test case (adaptation via algorithm above) had to be run in two separate trials due to hardware limitations. One of the raw images was selected from the log and hand labelled by a human. The robot's performance was then graded based on the number of pixels that robot classified correctly out of the pixels labelled as a color by the human. The robot was not penalized for the few pixels that were not colored

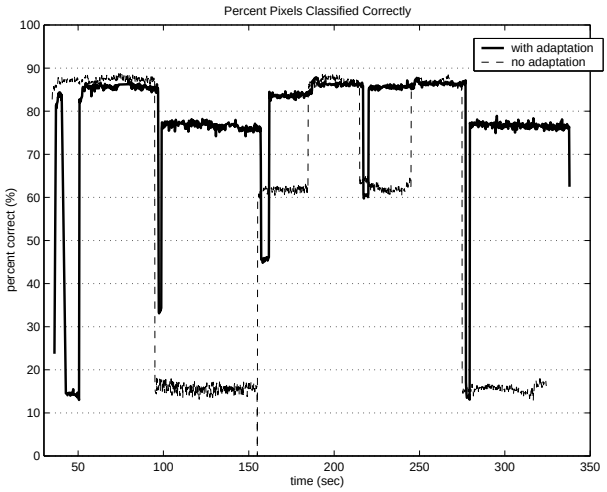

Figure 4: *Image Segmentation Results. The results with adaptation are shown in solid black. The results using only the bright thresholds are shown with dashed lines.*

that the robot thought were one of the colors. This is because most of these misclassifications are easily filtered out and the threshold generating algorithm tends not to produce many of these errors.

5.2 Results

The results of testing the robot on its image segmentation performance are shown in Figure 4. In all cases, the algorithm presented chose the correct thresholds after a small delay (to collect enough data). The run with no adaptation used bright thresholds throughout and so did very well in bright conditions, fair in medium conditions, and poorly in dim conditions. The sequence of lighting conditions used can be seen clearly in the performance of the no adaptation case (bright, dim, mid, bright, mid, bright, dim). There is a small amount of registration error between the transitions of the two runs due to starting the stop watch at slightly different times. Notice that when the robot is adapting to conditions, the robot performs poorly for a small period of time before performance approves again. This corresponds to the robot collecting enough information to determine that the lighting conditions have indeed changed (enough time for a window transition plus one full window's worth of data). The strange looking performance at the beginning of the adaptation run is largely an artifact of the test setup. Performance starts high (after the robot is well focused on the objects) and then drops suddenly before improving again. This is due to the training data being provided in the order bright, medium, dim so the robot starts off thinking the lighting conditions were most recently dim. The sudden drop is when the

first radio packet reaches the robot and switches the thresholds to dim. The delay before switching back is due to the robot gathering data which takes extra long since fewer radio packets are being sent since the robot is still starting up.

As can be seen in the figure, the adaptation dramatically improves the performance of the robot in segmenting the image without degrading the performance when the lighting conditions are consistent. This improvement in color segmentation of the image carries over to an improvement in object identification which in turn improves the performance of the robot in almost every task.

6 Future Work

Now that we have made the important step of developing a complete functional system which improves robot performance, we plan on many improvements and extensions to the algorithm. We plan to extend the algorithm to handle multi-dimensional data in both the input space (the space that gets divided) and the output space (the space that gets used for comparisons). Although the algorithm is already quite fast, we also plan to further improve the running speed of the algorithm to ensure that the running time eventually reaches a constant plateau. We would like to use the identified states as input to a Markov Model learning algorithm to give the robot an even better understanding of its world.

We plan on applying the framework to more tasks. In particular, we plan on applying the framework to automatic identification and recover from stuck states (falling over while walking in particular). We also would like to apply the framework to segmentation of images into regions with similar textures.

7 Conclusion

We have demonstrated a proof of concept system showing that sensors can be used to identify the state of the environment and/or system and that this state identification can be used to improve the performance of robots. This is an important first step which will form a base for many future improvements and advances. Naturally, as with any new line of work, there are many improvements that can be made to the algorithm to improve its performance and generality. Yet, despite these possible improvements, the algorithm already performs very useful tasks that are difficult, if not impossible, to do with existing methods. The identification of repeated states is also the first step in generating a Markov (or higher order) model of the world. In particular, we have shown how the algorithm can be used to improve the robustness/performance of the robot in the face of varied lighting conditions. This is a task that sorely needs to be solved to make robotics practical since lighting conditions vary constantly as the robot moves about any reasonably sized environment. The techniques described in this paper demonstrate how the robot can usefully adapt to its environment.

Acknowledgements: We would like to thank James Bruce for many useful discussions during the development of this work.

References

[1] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, 1998.

[2] T. W. Cacciatore and S. J. Nowlan. Mixtures of controllers for jump linear and non-linear plants. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 719–726. Morgan Kaufmann Publishers, Inc., 1994.

[3] L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *National Conference on Artificial Intelligence*, pages 183–188, 1992.

[4] D. R. Cutting, D. Karger, and J. Pedersen. Constant interaction-time scatter/gather browsing of very large document collections. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 126–135, 1993.

[5] S. Das and M. C. Mozer. A unified gradient-descent/clustering architecture for finite state machine induction. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 19–26. Morgan Kaufmann Publishers, Inc., 1994.

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

[7] Z. Ghahramani and G. E. Hinton. Variational learning for switching state-space models. *Neural Computation*, 12(4):831–864, 2000.

[8] J. Kohlmorgen and S. Lemm. A dynamic hmm for on-line segmentation of sequential data. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS 2001: Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[9] N. Kuiper. In *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen, ser. A*, volume 63, pages 38–47, 1962.

[10] K. Pawelzik, J. Kohlmorgen, and K.-R. Muller. Annealed competition of experts for a segmentation and classification of switching dynamics. *Neural Computation*, 8(2):340–356, 1996.