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.

Fig. 1. Robot used in testing.

I. 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.

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. The robot used in this work was built by Sony and is pictured in Figure I. Many practical color robotic vision systems rely on consistent pixel readings that allow colors to be segmented without reference to the surrounding pixel context. We use the free CMVision vision

This research was sponsored by Grant No. DABT63-99-1-0013 and by generous funding by Sony, Inc. The content of this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

system to perform image processing [3], [2]. 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 II. In Section III, we describe the algorithm for segmenting the data into different states. In Section IV, 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 V. We also describe testing procedures and results in this section. We finish with a discussion of future work and our conclusions in Sections VI and VII, respectively.

II. Related Work

There is a large body of work devoted to the analysis of signals. Basseville [1] has created an overview of change detection methods. This work varies in the number of assumptions made in the detection of changes. Most of the work is focussed on the case where strong knowledge about the state of the system before the change is available. This work does not address the problem of grouping the segmented signal into states, detecting repeated states, or adapting to states.

Several techniques based upon prototypes have been proposed for grouping data into states. These techniques all create a set of prototypical sensor readings that new sensor readings are compared against. Linåker and Niklasson [9] propose the adaptive resource allocating vector quantization(ARAVQ) method for segmenting and clustering sensor data. This method learns a segmentation of robotic sonar data on-line into regions of similarity. Updates to prototypes are conditioned on the prototype being a good fit to the data. Prototypes are added when the distance of the sensor data from existing prototypes exceeds a threshold. A moving average of the input vectors is used to help combat noise. Like most techniques, this technique lacks a proper model of the amount of noise in the sensor readings and must resort to an ad-hoc averaging of the input vectors to combat noise. Also like most techniques, it is incapable of noticing differences in the distribution of sensor readings that don't affect the mean. Marsland [10] proposes the Grow When Required(GWR) network for segmenting data. The technique focuses on the problem of novelty detection. This technique is a topology preserving prototype method similar to Kohonen maps. Nodes are habituated when they are chosen as most similar to input patterns. New nodes are created when no existing node is nearby and are not familiar based on habituation. The technique has been used successfully on real robotic data from sonar sensors and a camera with an attentional model. The novelty detector has also been extended to the problem of detecting environments. This is done by training a novelty detector on each environment that should be identified later. Environments are then identified based on which novelty detector is detecting the least novelty. This technique is trained off-line on the environments. Likhachev, Kaess, and Arkin [8] group data into states in a case-based reasoning framework for robot navigation. Cases are created on-line based upon spatial/temporal similarity of the current situation to prototype cases and the performance of the current cases. Case parameters for the matching case/prototype are adjusted by a learning algorithm. The technique is used to choose behaviors from a behavior library to navigate a robot towards a goal. All of these techniques cannot handle general changes such as distributional changes and changes in extremely noisy data.

Techniques that do not rely on prototypes have been developed by other researchers. Penny and Roberts [11] describe the way that Hidden Markov Model(HMM) autoregressive(AR) techniques can be used to segment data. These techniques use EM to learn an HMM of the data with an AR model for each state of the HMM. The techniques are trained off-line and must be seeded with reasonable guesses of the HMM in order for the training algorithm to converge correctly. Another class of techniques which are trained off-line is switching state-space models. See Gharamani [5] for an overview of these techniques. Kohlmorgen and Lemm [6] have developed a technique for automatically segmenting and clustering a time series. The input sequence is projected into a higher dimensional space by including in each sensor reading delayed values of previous sensor readings. A sliding window is then moved over the sensor values resulting in a series of probability distributions. The probability distributions are smoothed using a Gaussian kernel to generate a probability density function(pdf). Distances between pdfs are computed using the L_2-Norm. A cost function is defined which takes into account the ability of prototype distributions to explain the data and the number of switches. Prototype distributions are selected on-line in order to minimize the cost function. The prototype distributions are then clustered using a simple distance threshold. The technique was applied to EEG data. It is unclear how this technique will perform in robotic domains. This technique is non-predictive, has few parameters, and is completely on-line. The problems with this technique are the need to adjust the cost of a transition (effectively encoding the probability of a transition and the significance of differences in probability distributions), the assumption of the effectiveness of Gaussian smoothing the pdfs, and the poor clustering method (which also influences the segmentation).

In the specific case of a known number of classes with data available off-line, the state identification problem involves a classification problem from regions of sensor data to classes. Some classification techniques, such as k-nearest neighbor, can be applied to the regions of sensor data if an appropriate distance metric can be found. See Section III-C for one possible distance metric. The question of how to segment the data into regions of sensor data must still be addressed in this case.

Deng, Moore, and Nechyba [4] used locally weighted classification techniques to identify data of driver performance into drunk and sober classes. They used an ARMA model to represent the time series data. The parameters of the ARMA model where then used to index into a map from ARMA parameters to classes built using training data and local learning techniques. This system relies on the predictability of the data by an ARMA model which limits its use to data series with dynamics that can be approximately modelled with a linear predictor. The method for segmenting the data is left open in this work and the classified state is not used for adaptation.

III. 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 affects the decisions of the clustering algorithm. The distance metric controls the bias of the learning algorithm. Note that statistical difference tests such as the Kuiper test [7] 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
            dist(p, q) <= dist(i, j) ∀i, j ∈ S.
        Create new internal node r with
            p and q as children.
        S = S − {p, q} + {r}.
```

Fig. 2. Off-line Segmentation of Data

A. 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 algorithm is shown in Figure 2.

B. 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

```
Procedure Insert($T$:tree, $w$:window)
    let $n \leftarrow$ root($T$).
    let $S \leftarrow \{n\}$.
    let $done \leftarrow$ false.
    while($\neg$ $done$)
        let $R \leftarrow \emptyset$.
        foreach $s \in S$
            let $c1$,$c2$ $\leftarrow$ children($s$).
            let $d1$ $\leftarrow$ dist($c1$,$c2$).
            let $d2$ $\leftarrow$ dist($c1$,$w$).
            let $d3$ $\leftarrow$ dist($w$,$c2$).
            if $d2 < d1 \vee d3 < d1$
                if $d2 < d1$
                    $R \leftarrow R + \{c1\}$.
                if $d3 < d1$
                    $R \leftarrow R + \{c2\}$.
            else $R \leftarrow R + \{s\}$.
        $done \leftarrow (R=S)$.
        $S \leftarrow R$.
    Choose $b \in S$ 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 $o$.
```

Fig. 3. On-line Segmentation of Data

find the best place to insert the new window. The pseudo-code for the insertion is shown in Figure 3. This algorithm recurses down the tree like the simpler algorithm. The algorithm behaves differently when the comparison is ambiguous, however. If the new window to insert is more similar to both children than the children are to each other, the algorithm tries both branches. In this case, the comparison at the higher level is not very informative about which branch to take and both classes should be considered.

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.

C. Distance Metric

The distance metric used for measuring distances between probability distributions requires some careful consideration. The distance metric should reflect the degree of similarity of the underlying process generating the data. The distance metric should also make as few assumptions as possible. One possibility is to use a non-parametric statistical test as the distance metric. This makes for a very poor distance metric because it doesn't measure the degree of similarity between the probability distributions. Instead, these tests measure the probability that the distributions differ. This may seem similar to a distance measure but it is not. The problem is that two distributions with low variance separated by a small distance have the same probability of being different as two distributions with low variance separated by a large distance. Similar problems occur in more general cases because these tests do not take into account the magnitude of the change except in how it relates to the variance of the data. Another possible distance metric would be a measure of the difference between the probability density functions. This has the same problem as the statistical tests, namely that a small additive change appears to be a large change in distribution. The distance metric should capture the amount of change required to make the probability distributions the same.

The distance between the probability distributions can be measured as the distance the point samples from the two probability distributions have to be moved to coincide. This distance metric captures the important features of similar probability distributions in that distributions which produce similar samples are considered similar. The usual measure of distance would be to use the sum of the squared distance each point moves. Squaring the distance can lead to some bizarre artifacts, however. If two distributions are similar except for one point on completely opposite sides of the input space, using squared distance will result in a large distance between these distributions even though the probability distributions are similar. The usual motivation for using the distance squared is so that the resulting function can be easily integrated. In this case, however, the probability distributions are made of discrete samples so this criterion is not important and the absolute value of the distance can be used instead. The absolute value distance metric 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. This is the distance metric used in this work. A cumulative probability distribution is formed for both distributions, call them $F(x)$ and $G(x)$. Let $F'(p)$ and $G'(p)$ be 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

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

IV. 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 4). 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.

V. Application

The algorithm was applied to the task of automatically selecting vision thresholds by automatically identifying the

```
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).
```

Fig. 4. Labelling Classes

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.

A. 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 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.

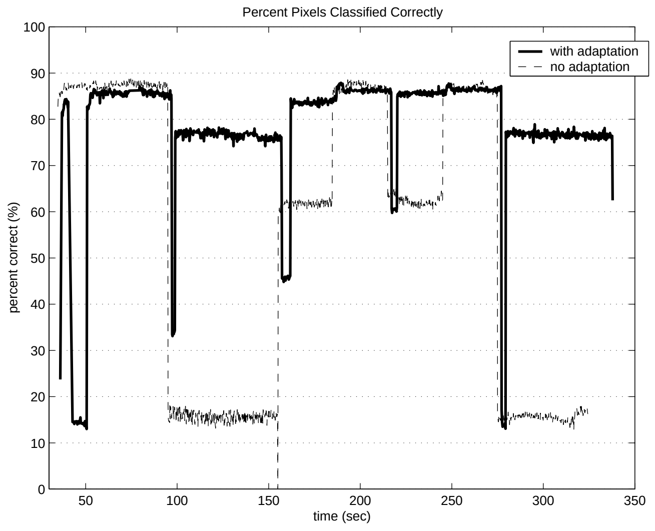

Fig. 5. Image Segmentation Results. The results with adaptation are shown in solid black. The results using only the bright thresholds are shown with dashed lines.

B. Results

The results of testing the robot on its image segmentation performance are shown in Figure 5. 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.

VI. 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 multidimensional 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.

VII. 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] M. Basseville and I. Nikiforov. *Detection of Abrupt Change - Theory and Application*. Prentice–Hall, Englewood Cliffs, N.J., 1993.

[2] J. Bruce, T. Balch, and M. Veloso. CMVision (http://www.coral.cs.cmu.edu/cmvision/).

[3] J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of IROS-2000*, 2000.

[4] K. Deng, A. Moore, and M. Nechyba. Learning to recognize time series: Combining arma models with memory-based learning. In *IEEE Int. Symp. on Computational Intelligence in Robotics and Automation*, volume 1, pages 246–250, 1997.

[5] Z. Ghahramani and G. E. Hinton. Switching state-space models. Technical report, 6 King's College Road, Toronto M5S 3H5, Canada, 1998.

[6] J. Kohlmorgen and S. Lemm. An on-line method for segmentation and identification of non-stationary time series. In *NNSP 2001: Neural Networks for Signal Processing XI*, pages 113–122, 2001.

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

[8] M. Likhachev, M. Kaess, and R. C. Arkin. Learning behavioral parameterization using spatio-temporal case-based reasoning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1282–1289, 2002.

[9] F. Linåker and L. Niklasson. Time series segmentation using an adaptive resource allocating vector quantization network based on change detection. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN 2000)*, pages 323–328, 2000.

[10] S. Marsland. *On-line Novelty Detection Through Self-Organization, with Application to Inspection Robotics*. PhD thesis, University of Manchester, 2001.

[11] W. Penny and S. Roberts. Dynamic models for nonstationary signal segmentation. *Computers and Biomedical Research*, 32(6):483–502, 1999.