Title: Object Recognition Using Clustering, Classification, and Centroids
Authors: Corey Proscia and Alexander Wang, CSC 242, University of Rochester
Date: May 8, 2006

Abstract:

Intelligent agents can recognize objects they have seen before based on characteristics of those items. We observed ratio measurements of size and color for objects from different angles and distances, then classified objects based on how similar they were to the "clustered" averages of past observations. The current implementation is limited to only the combinations it has been instructed to know; if the Quagent has seen kryptonite and gold individually but not both at the same time, it will not recognize both at the same time. We used the Java Quagent API designed by Aaron Rolett, Jason Freidman, and Chris Tice to interact with the Quagent world.

Keywords:

Quagent, Quake2UR, vision, object recognition, clustering, classification, size ratio, color ratio.

Background, Motivation, and Problem:

Our goal was to create an agent that would be able to recognize an object that it had seen before. We sought to use the clustering and classification method, where observation data on an object is similar over multiple observations. We selected two variables for measuring similarities – the shape ratio and color ratio – that should stay constant independent of lighting and distance in the Quagent world. They are also easy to implement and effective to use in two dimensions.

There are three methods of using clustering and classification – comparing a newly observed object to past observed centroids (average for an entire type of object), nearest neighbors, or n-nearest neighbors. In these systems, the newly observed object is classified as an

object average, single object observation, or n object observations respectively that is closest to the new observation. We chose to pursue the nearest centroid method because we felt the nearest neighbors method would be too prone to flukes. The centroid method does resemble n-nearest neighbors for high values of n, although it is not as biased against objects with a low number of observations, especially those with less than n observations.

Assumptions:

For reasons we can neither explain nor solve, the SocketChannel did not always provide all of the data from the DO LOOK command. The failures seem to increase as resolution size increases, so we tested our work mostly on the 320x240 resolution. Our only solution to this problem is to restart the Quagent and try again. We assume that, since we are using size ratios and color ratios that should be independent of resolution, our object recognition Quagent should have the same effectiveness on any resolution.

We also cropped the image (explained later) so anything in the top 30 rows of pixels or bottom 25 rows were not observed by the Quagent. Objects should be kept within the rest of the viewable screen.

Prior Work:

Teaching assistants Jason Freidman and Aaron Rolett distributed a very extensive package of Java classes that they created along with Chris Tice. This package handles socket connections and manages communications between the client and Quagent. We modified their package to create the Quagent, as explained later, and handle user commands. Our object recognition algorithm, although it exists in the Quagent code, is not based on their package.

Procedure and Discussion:

Our first challenge was to first generate an image from the Quagent. To do so, we used the DO CAMERAON option to change the point-of-view from the client to the Quagent. We then followed it with the DO LOOK command. This command takes a screen shot of the Quagent's point of view and then sends pixel-by-pixel information about the red, green, blue, and alpha values as a response to the Quagent controller. The alpha value indicated transparency levels and we did not use it. All four values were measured on a 0 to 2^8-1 scale and each was stored as eight bits – a single byte. We retrieved these values using the ByteBuffer's get() function. Upon testing this function in a red-wall map, we noticed that the values arrived in reverse order, as virtually all of the first values – which we expected were red – were zero. We stored these three color values as integers in the class Pixel and the entire image as a matrix of Pixels. In order to properly handle the DO LOOK response, we modified the receiveString() function in QuagentSocket. The new, renamed receiveData() function returned a ReceivedData class, which consisted of either an array of Pixels, in the case of a DO LOOK response, or a String, in the case of any other response. We sought to verify that our data was stored properly by creating a method to convert the Pixel matrix into a graphics file. Unfortunately, the conversion required the usage of the javax.imageio library, which was not immediately available on the CSUG network. As an alternate method of verification, we created a method to generate a HTML file that used a single '8' character to represent a pixel. The resulting file did accurately represent the world seen by the Quagent at the time of the DO LOOK command, although the file size of 3.2 MB was large enough to discourage frequent usage of this method.

In order to simplify our vision tasks, we decided to make the background invisible to the Quagent; foreground objects would stand out further in such an environment. We created a simple map consisting of only white tiles using the Pak Explorer and WorldCraft (shareware), both available online from the website quaketerminus.com.

Next, we sought to perform object recognition using the clustering approach. We observed each object from a variety of angles and distances, and measured these observations on two scales. The first scale was color recognition – the average ratio of red and blue to green for the entire object. We used the formula R/(R+G+B) + B/(R+G+B) to calculate this ratio. Using the ratio helps control for brightness – a light gray with RGB value 224-224-224 would be considered equal to a dark gray with RGB value 32-32-32. The second measurement was the area over the perimeter squared. This yields a ratio that corresponds to the two-dimensional shape of the object. A perfect circle would have a ratio of ~.0796 ($1/4\pi$), a perfect square would have a ratio of .0625 (1/16), and an equilateral triangle would have a ratio of ~.0481. We measured area of an object by the number of pixels it encompassed. To calculate perimeter, we counted the number of pixels in the object that were either adjacent or diagonal to a whitespace pixel. This ratio should stay fairly constant as the size of an object increases or decreases, due to game resolution changes or as the Quagent changes its distance to the object. It should also yield similar results as an object rotates. We also noticed during our calculations that some of the area and perimeter counts were higher than we expected. Upon further observation, we realized that Quagent messages at the top of the screen and health indicators at the bottom of the screen were being included in the count. In order to avoid this, we cropped the top 30 rows of pixels and the bottom 25 rows. This could lead to some problems with more complicated object recognition later on, but for our purposes it seemed sufficient.

In order to start the Quagent with a knowledge base, we observed a number of objects at different angles and distances in its "unexperienced" state. We tested five objects – battery (a rotating backpack), gold (a small pot), kryptonite (a rotating container), box (non-rotating), and head (rotating) – and three Quagent types – Mitsu (a standard soldier), CB (a dog), and Randal (a larger soldier). We did not test two objects for technical reasons; tofu is identical to a box, and data is invisible for a few milliseconds of its rotation. We also expected a high variance in shape for data, which would make it more difficult to identify. We tested all the objects thirteen times, except for gold and box, both nearly symmetrical and non-rotating, which we tested eight times. The results of our tests are in Chart 1. We then calculated the average shape ratio and color ratio for each object. We called this average a centroid – or the average.

We sought to have the Quagent determine which item it was seeing by determining the shape ratio and color ratio, then comparing the data for the current item with the stored data. More specifically, we wanted it to determine which centroid it was closest to. In order to balance the shape ratio and color ratio equally – our average shape ratios had a range of .0108 and our average color ratios had a range of .0782 – we normalized each scale from 0 to 1. This meant a difference of .0108 in shape ratio was considered to be equal to a difference of .0782 in color ratio for this specific example. The normalized centroids from this test are plotted in Chart 2. We had the "unexperienced" Quagent receive the average shape ratio and color ratio in a file. It would then normalize these ratios and with this information become "experienced." New objects seen by the "experienced" Quagent would first be normalized, then compared to the centroids. It is important to note that the normalization is relative to the maximum and minimum centroids, so values outside of the 0-1 range are possible for objects.

As a test, we manually fed the original observations back into the Quagent to see what results it would yield. As visible in Chart 5, the Quagent correctly identified kryptonite, box, and Mitsu every time. Overall, the Quagent identified 74 of the 94 original observations correctly (78.7%). The only object that was not identified properly most of the time was the battery, which was only identified properly 5 out of 13 times (38.5%). One unusual pattern we noticed in Chart 1 was that half of our battery observations had very different shape ratios from the other half. This happened with no other object – there were some irregularities with some objects, especially the other Quagents, but never such a distinct and repeated split. We discovered that this was because the backpack-shaped battery would rotate and reveal a transparent area between the straps of the backpack when looked at from the side. This increased area and perimeter by similar amounts, since most of the pixels making up the straps were considered perimeter pixels based on our earlier definition. However, since the perimeter parameter in the shape ratio is squared, we had a large drop in the shape ratio whenever the straps were revealed. We decided that the best way to deal with this problem was to create two centroids, one for each of the two battery clusters. The original single centroid (.7563, .0103) split in half fairly evenly to (.7536, .0071) and (.7515, .0141).

After re-normalizing this data, as in Chart 3, our success rate increased with the original observation identifications. All the batteries were successfully identified, although three objects that were originally correct were now being identified as batteries as well. Our success rate with this run was 79 of 94 correctly identified (84.0%). Overall, the centroids are balanced much better among the clusters with the two-battery-centroid system, as in Chart 4.

We also added a user interface that allows for the user to operate the Quagent with basic movement and turn commands, as well as LOOK.

Conclusion:

The most obvious area for expansion with our project is to increase the sophistication of our object recognition method. Currently, the implementation can only recognize a single type of objects – presumably in any quantity since shape ratio and color ratio should stay constant for multiple objects not obscuring each other. It would be possible yet cumbersome for a person to take a screenshot of multiple objects and store that information as a "battery and data," then have the Quagent able to recognize that specific combination. Our model is limited to only what has been seen, recorded, and averaged before. Based on a system with only single object original observations, we would expect less success for object recognition for multiple types, as well as obscured or partial objects, since the image takes the whole visible screen into account when looking for an object.

Learning is certainly a promising area to connect this project with. The Quagent could see an object, attempt to recognize it, and then ask the user or a knowledgeable program for feedback. Based on that feedback, the Quagent would improve its centroid based on the new observation. This would automate the process that we implemented manually as well as improve the Quagent based on its observations. However, we saw one situation with our experiment that could be troublesome for an automated Quagent, namely the rotating battery, as the Quagent or knowledgeable program would run into the same issues we did in the beginning of the experiment. A Quagent that implemented an n-nearest neighbors method of classification could solve this problem, although we prefer the centroid method of classification since it is not biased against objects with low numbers of observations.

Responsibilities:

Corey was the lead programmer, primarily working with the Java code and the README. Alexander was present for most of the Java coding and the lead author of the writeup.

References:

Anonymous. *Quake Terminus: Quake Tools*. http://www.quaketerminus.com/tools.htm. 2006.

Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach (2nd Ed.)*. New Jersey: Prentice Hall, 2003.

Various. *Quagents: Quake Agents*. http://www.cs.rochester.edu/research/quagents/. 2006.

Appendix: Charts are included in cpaw-5a.pdf.