Automatic Shape Recognition of Hand Gestures Using an Edge-tracing Vision System

Samuel N. Cubero

Abstract—This paper describes a working computer vision system and software algorithms that can automatically recognize different types of hand gestures. The software performs the image capture, analysis and shape identification for three different kinds of hand gestures, namely, the ‘rock’, ‘paper’ and ‘scissors’ hand gestures used in a popular child’s game. The software is comprised of three main elements, namely: (1) A method for collecting data to store normalized or ‘template’ images of known shapes or known hand gestures; (2) An edge detection algorithm; and (3) A method to compare an image with normalized or ‘template’ images. This method of hand gesture recognition could be adapted for use in many practical applications relating to the control of computers (e.g. 3D graphics, controlling games, real-time simulations), robot manipulators, mobile robots and machinery.

Keywords—hand gesture, recognition, vision, image, template

I. INTRODUCTION

This paper describes a simple and reliable method for capturing an image of a hand gesture and correctly identifying the type of gesture by comparing this image against images of known hand gestures (or normalized ‘template’ images). This method of gesture recognition requires edge-tracing of the silhouette of a hand image (or object boundary), encapsulation of this edge data using s-psi (or displacement-angle) coding, normalizing this data to a set number of data points and then comparing this s-psi data to each ‘normalized’ or template image for each known type of hand gesture.

II. RELATED WORK

The motivation for this work began around 2006 when the author wanted to develop a gesture-based control system to manually control the legs of a 6-legged hydraulic walking robot with 4-wheel-drive capability (known as ‘Hydrobug’), which was being developed at Curtin University of Technology, Perth, Western Australia. It was envisaged that the positions of the feet of the walking robot would be able to mimic the positions of the fingertips of the driver, under manual mode. A visiting student from the Punjab Engineering College (Chandigarh, India), Mr. Harvarinder Singh, agreed to work on developing a PC-based hand gesture recognition system as the topic for his final year engineering project, under the supervision of the author. This paper is based mostly on that project but it has been expanded to include many important details that were omitted from the report [11].

Around this time, there were several different computer vision methods that could have been used to recognize hand gestures. For example, McConnell [1] describes a method of hand gesture recognition that employs ‘histograms’ of local orientation. Freeman and Roth [2] expound on this method by using orientation histograms as a ‘feature vector’ for gesture classification and interpolation. Some machine vision researchers have worked on developing software that attempts to read ‘sign language’ and pointing gestures [3], [4]. Unfortunately, these methods required markers to be placed on the fingers and hands so are undesirable and inconvenient.

Several machine vision researchers have been able to create successful hand gesture recognition systems, without the need for markers [5] - [8]. Unfortunately, some of these methods shared disadvantages, such as being highly complex to implement and being computationally inefficient, resulting in low frame-rates for image analysis. Blake and Isard [9] describe a fast contour-based tracker to recognize different hand gestures and lip movements. Blake and Isard’s vision system could trace the edges or curved silhouettes of moving objects (such as hands, lips, legs, vehicles, fruit, etc.) at reasonably good frame-rates. Their method is a synthesis of methods in deformable models, B-splines curve representation and control theory. This method can be executed efficiently enough on standard PCs and workstations so that hands and lips can be used as real-time input devices or game controllers.

From 2001 to 2003, Dunn and Billingsley [10] developed a machine vision system that could trace the edges or curved silhouettes of wild animals (such as sheep, goats and pigs) entering and leaving a natural water source, as part of a project to remotely monitor animal usage of the water source and to automatically block water access to wild pigs. Dunn and Billingsley’s vision software was able to accurately count the many different species of animals walking past a digital camera through a narrow passageway leading to the water hole (with a blue tarpaulin in the background to provide good colour contrast). It could accurately identify the species of a particular animal in view, such as a sheep, goat, pig and even kangaroo, regardless of the size or height of the animal. The lengths of the animals varied from 50 cm to 300 cm, and walking speeds varied from slow to fast (up to 2 or 3 m/s).

Hence, edge-tracing of objects and comparing these edges to the edges of known images appeared to be a successful and efficient method for identifying different shapes and objects.
A. **Data format for pixels**

Once a single image is captured by the ‘Eye’ ActiveX control, each of the 8-bit RGB (Red, Green and Blue) colour values for each pixel are stored in an array in memory. Each pixel in the image is comprised of 3 x 8-bit numbers, or consumes 24-bits of memory space, or 3 bytes of data. The colour ‘Black’ would have an RGB value of (0, 0, 0), and the brightest colour for ‘White’ would have an RGB value of (255, 255, 255) = (R, G, B). Pure red would have an RGB colour value of (255, 0, 0). Every other colour in the visible spectrum can be represented by a different combination of R, G and B values. To achieve a good contrast between the hand image and the background, black towels or sheets were used for the background and a bright 100W electric halogen bulb was used to illuminate the hand from the camera position.

B. **Edge-tracing algorithm**

The edge-tracing algorithm is important for converting the boundary of an object into an s-psi (or ‘displacement-angle’) graph, which can be compared against other template graphs.

---

Fig. 2 Search pattern to find a different coloured pixel (Octagon ‘mask’ surrounding the ‘cursor pixel’ as it moves around the edge of an object)
Once a black pixel (or very dark pixel having very low R or G or B values) has been detected for the first time (indicating the background colour has been found), the algorithm marks this position as the ‘start of the boundary’ position, and proceeds to find the next ‘white’ pixel using the search pattern shown in Fig. 2. Figure 3 shows how the ‘cursor pixel’ is marched leftwards until it finds a black pixel (or the dark background colour). Once this is found, the subsequent angle positions for pixels with a colour different to the colour of the current ‘cursor pixel’ are recorded. For example, if the current ‘cursor pixel’ is on white, the edge-detection algorithm must search for the next black pixel. Once the black pixel is found (in the clockwise direction around the shape), the angle from the ‘cursor pixel’ to this black pixel is recorded. This black pixel becomes the new ‘cursor pixel’ position, and the algorithm must now search for the next white pixel. Once the next white pixel is found, the angle from the previous black ‘cursor pixel’ position to this white pixel is calculated and saved, and the entire process repeats as the ‘cursor pixel’ marches its way around the closed area (white) shape for the hand in a clockwise manner. In Fig. 4, each arrow is a vector or angle direction from the previous ‘cursor pixel’ and is recorded for each new ‘cursor pixel’ position. For simplicity and speed of processing, the ‘angle directions’ to each ‘cursor pixel’ position can be stored as numbers from ‘0’ to ‘11’ according to the directions shown in Fig. 2. If the pixel positions in the octagon ‘mask’ of Fig. 2 are scanned for a different colour (either in the clockwise or anti-clockwise direction, depending on the current ‘cursor pixel’ colour), the ‘psi’ angle to the next different coloured pixel can be determined roughly using one of the ‘angle directions’, 0 to 11, relative to the ‘current pixel’ position at the centre.

Once the ‘cursor pixel’ position is very close to the ‘start of the boundary’ position (within an acceptable tolerance distance), the edge-detection algorithm can then terminate.

Dunn and Billingsley [10] use a slight modification to this algorithm, by keeping the background (blue) colour to the left and the bright (animal object) colours to the right of the cursor pixel in the stepping algorithm. Their edge is defined as the boundary between the blue / non-blue pixels, where at each step, the algorithm determines the vector or angle direction stepped from the previous ‘cursor pixel’, so that all ‘cursor pixels’ remain on the bright pixels only (or the object’s area, like the closed shape in Fig. 5). This is achieved by scanning for the next pixel of opposite brightness and, once found, stepping back to the previous arrow (vector), to point at a pixel with the same brightness. Detailed descriptions of common methods for edge detection are found in [12].

Once an array of angles (or direction vectors) vs. step positions has been recorded after the edge-tracing algorithm is completed, it would be useful to re-organize this array so that all the ‘index numbers’ or ‘boundary pixel positions’ begin at a common point that matches a known template image point, such as the highest and right-most pixel on the boundary of the image. For example, in this ‘rock, paper, scissors’ case study, the wrist is assumed to be on the left side of the image, while the finger tips are assumed to be on the rightmost side. The ‘first’ step or ‘first cursor point’ on the boundary or edge can be set at the pixel position that has the largest x coordinate (right-most boundary pixel), which usually corresponds to the fingertips or a knuckle of the longest (middle) finger, a good starting point to begin normalized comparisons. The original array of s-psi values can simply be reorganized by copying all the angle values into a new array based on an offset number, so that the first element of the new s-psi array begins at the right-most and highest pixel on the boundary of the image, or the pixel with the largest x-coordinate. If there are two or more pixels that have the same maximum x-coordinate, then start the s-psi array at the pixel that has the maximum x-coordinate and the maximum y-coordinate on the boundary of the shape. Maximum values for x or y coordinates can be found using a simple ‘bubble sort’ algorithm to compare x or y coordinate values. This would make the comparisons between images and template images very robust and reliable because their starting points would be at almost identical locations on the boundary of an image.

C. Normalisation of s-psi (displacement-angle) data

Before comparison of the s-psi data for an image can be compared to the ‘template’ s-psi data of a known object or
hand-gesture, it is essential that both sets of data have the same number of boundary points (i.e. the same number of angles to be compared, or the same number of elements in the arrays to be compared.) This can be achieved in several different ways as will be described briefly in this section.

If the image array has I number of points, and the template array has T number of points, where T > I, then you need to skip points in the template array. For example, if I = 1500 and T = 3000, the ratio of T:I = 2:1, so you only need to compare every 2nd point in the template array to the image array. If T:I = 3:1, then compare every 3rd point, etc. If T/I is not a whole number or integer value, like T/I = 3.2 = 1.5, it is possible to expand both data arrays so that both arrays have the same number of elements. The template array could be doubled in size, simply by duplicating angle values to create a new template array with double the number of elements, and the image array can be tripled in size, so that the T/I ratio becomes 6:6 = 1, allowing both arrays to be compared directly, however, this leads to a lot of wasted computing time. In most situations, you might end up with unpredictable values like T = 2432 and I = 1977. In this case, you can create 2 new data arrays, one for the image array and one for the template array, where each array has only 100 elements. Set up a fixed counter ‘FOR LOOP’ that counts from 1 to 100, then at the 1%, 2%, 3% boundary position values (all the way up to 100%) for both arrays, record the respective ‘psi’ angle into the template and the image arrays, using a ‘0-11’ angle direction vector or a linear interpolation of absolute angles.

For example, the 1% position for the template array would be 0.01 * 2432 = element number or pixel number 24 (rounded down from 24.32) in the original template array. The angle for this 1% position value at pixel 24 should be recorded. The 2% boundary value is 0.02 * 2432 = element number 49 (rounded up from 48.64), hence, the angle for this 2% position value from the start of the boundary should be recorded in the new array of 100 elements. The same should be done for the image array, so that both data arrays will have the same number of angle (or direction vector) elements in the end, where I = T = 100, for easy comparisons of ‘psi’ values. For good accuracy and sharper boundary details, it is recommended that at least I = T = N = 256 angles be used in each array for capturing images of hand gestures, where N = number of angles (or vector directions) or pixel positions in the normalized array. More angles to store and compare will cost more processing time, so it is best to keep N fairly low.

For this ‘rock, paper, scissors’ case study, the normalized images (s-psi arrays) for the three known hand gestures were saved to three different text files on disk and could be loaded up into the software. It is important for the N value (number of normalized boundary pixel points, or displacements on the s-psi graph) to be the same for all ‘template’ images and all hand gesture images being tested. This allows the software to successfully compare captured images and template images even though the original captured images are of different size. Original image size depends on the distance between the hand and the digital camera or the actual physical size of the hand being photographed.

D. Comparison Algorithms

Two different types of simple comparison algorithms were used to check for matches between the captured image and the template images. They are called ‘Method 1’ and ‘Method 2’.

1) Ratio of area to the total number of boundary points: This method of comparison calculates the area (or number of pixels) inside the boundary of the original image and divides this value by the number of boundary points. This method is used to differentiate between the ‘scissor’ type image and the other two gestures (‘rock’ and ‘paper’) because the ratio of area to boundary points is always less than 38, while the ratio for the other two gestures is always greater than 40. Unfortunately, area calculations, especially on high-resolution images, is very time consuming and often takes several seconds for large images. This comparison algorithm is not recommended for real-time or high frame-rate applications.

2) Finding which template is the closest match by summing up angle errors between both normalized s-psi graphs for the image and the template: The absolute angles (or even 0-11 vector direction values) stored in the normalized s-psi arrays for both the template image and captured image can be subtracted from each other to produce a signed angular ‘error’ for each corresponding normalized boundary point (since there are N number of boundary points in both the normalized image and the template arrays). These angle differences, or errors, can be added up for each template image and the template image that has the lowest total error value (below an acceptable error tolerance level) can be considered to be the best match with the captured image. This method proved to be very fast, efficient and reliable, and shows good potential for real-time, high-speed gesture-recognition applications.

Another method that can be used, but is mathematically identical to Method 2, is Comparing the sum of angles: This method involves simply adding up all the signed angles (or 0-11 vector directions) in the normalized image array to get a total ‘sum of angles’ value, and then comparing this to the ‘sum of angles’ value for the normalized template array. If the difference in ‘sum of angles’ for a template is the smallest among all the templates compared, or is below a very small acceptable error tolerance, then a match is confirmed. This method may work for only a few types of hand gestures (like this ‘rock, paper, scissors’ case study), however, if many more hand gestures need to be checked (or more templates are added), there is a higher risk that two or more different hand gestures will have the same or similar ‘sum of angles’ value, therefore, additional comparison checks may be necessary to avoid incorrect matches.

IV. RESULTS AND DISCUSSION

Figures 6, 7 and 8 show screenshots of the machine vision software successfully recognizing three different hand gestures on a Pentium 4, 2.66 MHz Windows PC. Method 2 was the most reliable and fastest comparison algorithm, requiring less than 1 second to process an image (at N=256). Method 1 was slow (often taking 7 or more seconds) [13].

http://www.ijipvc.org/article/IJIPVCV1I301.pdf
Using the comparison method ‘Method 2’ alone, the chances of a false identification, or a false match, increases, and so does the computer processing time, as more template images (or more known hand gestures) are added for comparison with the captured image. A demonstration video of this machine vision software is shown at [13], with both ‘Method 1’ and ‘Method 2’ recognition algorithms activated. The slow recognition speed, as seen in the demonstration video, is because ‘Method 1’ is active (due to the very large number of pixels and boundary points to process in a non-normalized image, where the number of boundary points is typically in the thousands for a high resolution image), however, ‘Method 2’ alone is typically up to 10 or more times faster if N, the number of normalized boundary points, is kept low. (Recognition speeds are typically very fast when N<256)

The ‘Method 2’ comparison method described in the previous section is a very simple one, which simply calculates the difference between the ‘vector angles’ (or 0-11 direction values) at the corresponding ‘cursor pixel’ points for both s-psi graphs. It is still a rare possibility that a hand image could have the same ‘sum of angle differences’ value as a known ‘template image’ for a very different shape. To help reduce the chances of a wrong match, it would be possible to record information on how the subsequent angles change as each ‘cursor pixel’ is being scanned, or store the ‘gradient’ (slope $d\psi/ds$ or even $\psi_n - \psi_{n-1}$ where $n$ is the displacement number or ‘pixel position’), of the s-psi graph at each ‘cursor pixel’ position, and then perform another ‘sum of differences’ comparison for these gradients, as an extra check to see if the actual s-psi graphs are similar. This extra check would significantly reduce or eliminate the chances of a false identification because it would be highly unlikely that both the arrow (or vector) angle and the gradient of the s-psi graph, at the same ‘cursor pixel’ position, would be similar for the s-psi graphs of two completely different hand gestures, let alone for all of the boundary points. The smaller the ‘sum of errors’ for the differences between the magnitudes of the psi angles and their gradients at each pixel position, the better the match is between the captured image and the template image.

Another noteworthy point is the fact that the actual order of the angle directions (or vector angles between successive ‘cursor pixel points’) for the captured image and template images are not recorded nor compared. Such information could be recorded and compared to gain greater confidence in a ‘positive match’ result. Similar to the ‘gradient comparison’ approach, the ‘increasing’ or ‘decreasing’ trend of the psi angles can be recorded at each boundary pixel position. If the psi angle at the current ‘cursor pixel’ position is greater than the psi angle at the previous ‘cursor pixel’ position, then record an ‘increasing’ (+1 trend value) angle trend. If the psi angle is less, then record a ‘decreasing’ (-1 trend value) angle trend at that current ‘cursor pixel’ position. Hence, ‘increasing’ and ‘decreasing’ angle trend values can be stored in separate arrays for the normalized captured image and each normalized template image, to be used for a comparison check. For example, the trend values can be summed up and compared to check for a match between a captured image and a template image, to gain greater confidence in a ‘positive match’ result.

Method 2 is very simple and more test methods need to be employed to compare certain regions on a captured image against known features on a normalized s-psi graph. For example, Dunn and Billingsley [10] were able to discern the subtle difference between a sheep and a goat by analyzing the s-psi graph near the ‘head region’ of both image and template arrays. Goats have a characteristic longer neck than sheep, hence, only the neck and head regions of the boundary needed.
a comparison check. Similarly, for hand gestures, features like long thin fingertips (where the edges turn sharply like a ‘hairpin bend’ curve on a race-track), thumb joints and other prominent features that are unique to a known hand gesture, could be compared to check for specific image features that match known hand gestures.

Another optional (but not essential) procedure that can be performed before the comparison process is ‘data smoothing’ for all the normalized template s-psi graphs (for known hand gestures) and the normalized image s-psi graph. This involves using a ‘3-point running average’ method, or perhaps, a weighted equation that places greater importance on the psi angle for the current cursor pixel position (n) on the boundary of the object. For example, a new ‘smoother’ s-psi graph (or array of angles) can be created by using a smoothing equation like:

\[
\psi_n = \frac{(\psi_{n+1} + 2 \times \psi_n + \psi_{n-1})}{5} \quad (2)
\]

or for more emphasis on the current cursor pixel’s angle, use

\[
\psi_n = \frac{(\psi_{n+1} + 3 \times \psi_n + \psi_{n-1})}{5} \quad (2)
\]

An obvious advantage of smoothing out the data prior to executing a comparison check would be smaller error magnitudes, however, this comes at the cost of extra computing time.

These kinds of ‘confidence checks’, data smoothing methods and image comparison methods will be tested in future work.

V. CONCLUSIONS

This paper described an efficient and reliable edge-tracing vision system for recognizing three different hand gestures presented to a digital camera in a specific direction or orientation. Every picture ‘image’ is converted to a normalized s-psi graph, which is subsequently compared to the normalized s-psi graphs of known shapes. Normalization of the s-psi graphs allows for simple angle comparison checks and makes the object recognition algorithm insensitive to variations in object image size or distance between the hand and the camera.

These detection algorithms can be modified or extended to deal with variable tilt angles of the hand and their reliability could be further improved by running additional comparison checks.

At present, this hand recognition software could be easily modified and used as a simple controller for a pointing device for a computer, a game controller or a controller to drive mobile robots and other kinds of computer controlled machines. Different normalized s-psi templates can be created and used for each known hand gesture corresponding to each unique command. The hand recognition software can be programmed to continually capture images periodically (say, every 1 second) and respond to successfully identified hand gestures, while ignoring unrecognized hand gestures.

Future work will be extended to the recognition of gestures from two different hands (left and right hands) captured in the same image and to improve software robustness (detection reliability) and speed of processing.

ACKNOWLEDGMENTS

Figures 6, 7 and 8 were reprinted from [11] with the kind permission of Mr. Harvarinder Singh, a former undergraduate engineering student from the Punjab Engineering College, Chandigarh, India. Mr. Singh worked under the guidance and supervision of the author to develop the Windows™ vision system software shown in [13] using Visual Basic™ 6. The Department of Mechanical Engineering at Curtin University of Technology also deserves special thanks for accepting Mr. Singh as an ‘exchange student’ in Perth, Western Australia.

REFERENCES