Binary vision pipeline

In this assignment, you will implement a full binary vision pipeline capable of recognizing objects of different shapes. For this assignment, you will be using images you take with your own camera. Choose at least three objects of different shapes, preferably with non reflective (shiny) surfaces. Possibilities include paper/cardboard cut-outs, bottle tops, pencils, lego pieces, etc. Take pictures of these objects individually with a solid background, the object should be clearly distinguishable from the background. Your pictures should be close to frontal parallel and far enough away to assume orthographic projection (distance to object is about 10 times the height of the object). See figure 1 for examples.

![Sample images](image)

Before working with your images, crop and/or resize your images so that the largest dimension is around 300 pixels. Also, convert your images from RGB to gray (use Matlab function rgb2gray).

1. Write a function to threshold your images separating the background from the foreground. You may choose the “peakiness” detection algorithm described in class, or this iterative “optimum threshold” algorithm:
   - Choose initial threshold $T$ to be the mean intensity of the image.
   - Let $\mu_a$ and $\mu_b$ be the mean intensity of pixels above and below the threshold, respectively.
   - Choose new threshold $T_{\text{new}} = (\mu_a + \mu_b)/2$.
   - Repeat the previous two steps until $\text{abs}(T - T_{\text{new}}) < d$ where $d$ is some sufficiently small number.

   The output of your function should be a binary image with 0 for all background pixels, and 1 for all foreground pixels. Apply this function to your images and display your results. See figure 2 for results on the example images. (**10 points**)

2. Write a program which implements the connected components algorithm described in class. You may assume either 4 or 8 connectedness, state your assumption. Your program should take as input a binary image (output from previous step) and output a new image of the same size where each connected region is marked with a distinct positive integer. Test your program on the output from the previous step. You don’t need to display results for this step, but you
should verify that the output should equal the input, since there is only one object in each of
the three images. (**15 points**)

3. Write three functions which compute the moments, central moments, and normalized moments
of a marked region. Each function should take as input an image, \(i, j\), and \(d\). The output
should be the \((i, j)^{th}\) moment/central moment/normalized moment of marked region \(d\) in the
input image. The formula for normalized image moments is

\[
\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{(1+i+j)/2}}
\]

which is scale and translation invariant. (**15 points**)

4. Using the function in the previous step, on each of the three images, draw the centroid of each
object. Also compute the eigenvectors of the centralized second moment matrix and draw the
two eigenvectors on the centroid. This should indicate the orientation of each object, see figure
3 for the results on the example images. (**10 points**)

5. Write a function to return a vector of moments you can use to describe each region. This
function should take as input an image and the region \(d\) for which to compute the vector
of moments. The output should be a vector of the scale and translation invariant moments.
Minimally, this vector should include \((\mu_{10}, \mu_{01}, m_{10}, m_{01})^T\), but it may require some exper-
imentation to see which moments are most discriminative. Store a feature vector for each of
your three objects. (**5 points**)

6. Now you will put all the pieces together. Take an additional three pictures with all objects
present in each image. For each image, do the following:

---

Figure 2: Results from thresholding step.

Figure 3: Centroids and eigenvectors of second moment matrix.
- Use your segmentation routine to produce a binary image.
- Use your connected components routine to produce an image of connected regions (should be three). Display your results for this step.
- Extract a feature vector \( f_i \) for each connected region using the function in part 5.
- Label each region by comparing the feature vector to each of the three feature vectors from the first three images. Choose the label for which the euclidean distance between feature vectors is minimized. Specifically, let \( t_j \) be the feature vectors for each of the first three images. The label you will choose is \( l = \arg \min_j \| f_i - t_j \| \). This is essentially what a nearest neighbor classifier does.

Note that your feature vector achieves scale and translation invariance, but not rotation invariance. Therefore, your additional three images should have the objects in different positions, but in the same orientation. At least one image should have the camera either farther or closer to the objects to see if scale invariance was indeed achieved.

Output the result of your algorithm. The results should be in the form of an image with each component labeled by your program. Results for one image can be seen in figure 4. Note that depending on the objects and feature vector used to describe the objects, you may not get perfect results. (20 points)

Figure 4: Final results for one image (a) Connected components result using imagesc and (b) regions labeled using Matlab text function.

7. (Extra Credit). As mentioned, the central and normalized moments only achieve scale and translation invariance. To achieve rotation invariance, Hu moments may be used. Recall the formula for normalized image moments:

\[
\eta_{ij} = \frac{\mu_{ij}}{(1 + \frac{i^2 + j^2}{2})^{3/2}}
\]
Then the seven Hu moments are defined as

\begin{align*}
I_1 &= \eta_{20} + \eta_{02} \\
I_2 &= (\eta_{20} - \eta_{02})^2 + (2\eta_{11})^2 \\
I_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
I_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
I_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + \\
&\quad (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
I_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
I_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + \\
&\quad (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2].
\end{align*}

These moments are scale, rotation, and skew invariant. Write another function similar to the one in part 5 that returns a feature vector using these moments. Now try taking another three pictures with all objects together. This time, you may rotate the objects as well as moving them around. Repeat the steps in part 6 to label each region of the image. As before, show your results in the form of an image with each region labeled according to the output of this step. (10 points)

Turn in code and figures for each step. Please comment your Matlab code appropriately. Also include a paragraph or two describing your results. Did you achieve perfect results? If not, what could be the possible causes? Start early as some of the steps require some experimentation. Some hints:

- For the connected components routine, see Chapter 4 of “Robot Vision” by Horn for alternative implementations.

- To plot the eigenvectors, you may use the quiver function:

\begin{verbatim}
quiver(x, y, vx, vy, 100)
\end{verbatim}

where \(x, y\) are the centroid coordinates, and \(vx, vy\) are the coordinates for one eigenvector.