Physical Computing Group 7E: Beatrix Zöllner, Jaspreet Singh, Tsungai Chipato

Lab 4: Classification

We chose the image assignment.

Exercise 1a: *Write a MATLAB script that can recognize objects from an image by using a rule–based classifier. You don't have to restrict yourself to the objects presented in this example; you could recognize other types of objects, such as keys, pens, etc. You can also recognize characters, based on geometrical features such as orientation, eccentricity and Euler number.*
Answer: We have two functions: one function for labeling (`getimage`) the image and one function for classifying objects in the labeled image (`Classifier`). In order to run these functions, a file with image of the objects (in our case: `applekeycoin.bmp`) is needed. The image can be seen in figure 1. We can run the function that labels our image (`getimage`) using the following command:

```
>> getimage('applekeycoin.bmp');
```

This is the MATLAB function:

```
% This function gets the filename ('applekeycoin.bmp').
% We do this by typing the following command in the command window:
% myfunction('applekeycoin.bmp')
% (Because we named the file with omage 'applekeycoin.bmp'.)

function [im] = getimage(filename)
      % We saved the obtained image as 'applekeycoin.bmp'.
      % We use the following line of code to obtain it in MATLAB:
      im = imread(filename);
      % We look at the (original) image:
      imshow(im);
      title('Photo we took');
      % For two seconds:
      pause(2);

      % We cropped the original image.
      im1 = imcrop(im, [0, 0, 635, 435]);
      % We look at the cropped image:
      imshow(im1);
      title('Cropped image');
      % For two seconds
      pause(2);

      % We make a grayscale image of im1:
      im2 = rgb2gray(im1);
      % We look at the grayscale image:
      imshow(im2);
      title('Grayscale image');
      pause(2);

      % We obtain the histogram of the grayscale image.
% We look at the histogram:
imhist(im2);
title('Histogram of the grayscale image');
% For two seconds:
pause(2);

% We use the MATLAB function to obtain the thresholds of the
% grayscale image. We store the value we obtain in the variable
% called level:
level = graythresh(im2);
% We display the threshold:
disp(['Threshold: ', num2str(level)]);
disp([' ']);

% We use the threshold we obtained, of which the value is stored in
% the variable level, to get a black and white image. The threshold
% used to determine what level of gray is considered as black and
% what level of gray is considered as white. We get a black and white
% image:
im3 = im2bw(im2, level);
% We look at the black and white image:
imshow(im3);
title('Black and white image');
% For two seconds:
pause(2);

% We use the MATLAB function, STREL, in order to get disk-shaped
% structuring elements:
im4 = imopen(im3, strel('disk', 13));
% We look at im4:
imshow(im4);
title('STREL of black and white image');
% For 2 seconds:
pause(2);

% We use the MATLAB imcomplement function in order to complement im4:
im5 = imcomplement(im4);
% We look at im5:
imshow(im5);
title('Complemented black and white image');
% For two seconds:
pause(2);

% We use the MATLAB function, STREL, in order to get disk-shaped
% structuring elements:
im6 = imopen(im5, strel('disk', 8));
% We look at im6:
imshow(im6);
% For two seconds:
pause(2);
```

```matlab
% We want use labels to get to know how many there are in the image:
[labels, numlabels] = bwlabel(im6);
% We want to use the variables 'labels' and 'numlabels' also
% when we use the Classifier function to classify objects, so make
% them global variables:
global labels;
global numlabels;

% We use the variable L to label each object. L is an integer and
% starts at 1. We give each found object from the image (im6) a
% label, which is defined by a value (1, ..., n where n = numlabels):
L = bwlabel(im6);

% We use a for-loop to look at each labeled object separately:
for (var = 1:numlabels)
      % We look at an object using the labeling variable L. So, if
      % var = 1, then we look at the object which is labeled with the
      % value L = 1 and the same goes for the rest of the objects:
      imshow(L == var);
      title(['Object ', num2str(var)]);
      % We look at each object for one second:
      pause(1);
% We end the for-loop:
end

% We look at all the labeled objects using the vislabels function:
vislabels(L);
title('All the objects');
% For two seconds:
pause(2);

% We subplot im, ..., im6, each labeled object separately,
% and all labeled objects, using the subplot function:
title('Mosaic');
subplot(3, 3, 1); imshow(im);
subplot(3, 3, 2); imshow(im1);
subplot(3, 3, 3); imshow(im2);
subplot(3, 3, 4); imshow(im3);
subplot(3, 3, 5); imshow(im4);
subplot(3, 3, 6); imshow(im5);
subplot(3, 3, 7); imshow(im6);
subplot(3, 3, 8); imshow(L);
% We use a for-loop to look at each labeled object separately:
for (var = 1:numlabels)
      % We look at an object using the labeling variable L. So, if
      % var = 1, then we look at the object which is labeled with the
      % value L = 1 and the same goes for the rest of the objects:
      imshow(L == var);
      title(['Object ', num2str(var)]);
      % We look at each object for half a second:
      pause(0.5);
```

```
        % We end the for-loop:
        end
        subplot(3, 3, 9); imshow(L);
        % We look at all the labeled objects using the vislabels function:
        vislabels(L);
        % We determine how we want to show the window with the mosaik
        % of subplots:
        set(figure(1), 'Position', [100, 100, 1000, 800]);
% We end the function:
end
```

In order to actually classify the objects from the labeled image (we obtained using the `getimage` function), we have to use the `Classifier` function. We can type in the following command in the command window to run the function:

```
>> Classifier;
```

This is the MATLAB function:

```
% This function can be used to classify an image with
% coins, keys, and apples using our own rule-based classifier.
% Before we use this function, we have to run the getimage function.
% Once we have done that, we can run this function by typing the
% following command in the command window: Classifier

function Classifier
        % We have to use global variables in order get information from the
        % function we used to label our objects:
        global labels;
        global numlabels;

        % We want the area and perimeter of each object, so we can classify
        % the objects by area. If not area, we can use the area and perimeter
        % to calculate the form factor, so that we are still able to classify
        % the objects. We obtain the information we need for classification:
        stats = regionprops(labels, 'all');

        % We initialise the counter for each type of object we want to count
        % to zero. We took an image of an apple, coins, and keys:
        apple = 0;
        coin = 0;
        key = 0;

        % We used a for-loop. We start at 1 and we execute the loop until we
        % have classified all objects. The total number of objects is equal
        % to the value stored in numlabels:
        for (x = 1:numlabels)
                % We calculate the form factor of each object (x is the
                % label of the object, kind of. So, if x = 1, then we calculate
                % the form factor of object 1 which has the label L = x. In
                % this example, L = 1 (because L = x = 1).
                % We calculate the form factor of an object of which the value
                % is stored in the variable f, using the following formula:
```

```
      % f = (4 * pi * A) / P^2 where A is the area of the object and
      % P is its perimeter:
      f = 4 * pi * stats(x).Area / ((stats(x).Perimeter)^2);

      % We display the form factor and area of each object:
      disp(['Object ', num2str(x), ': ']);
      disp(['Form factor of object ' , num2str(x), ': f = ', num2str(f)]);
      disp(['Area of object ', num2str(x), ': ',
      num2str(stats(x).Area)]);

      % If the area of pixels occupied by the objects of the object
      % is bigger than 40000
      if (stats(x).Area > 40000)
            % We display the sentence: 'Object <number of object> is
            % an apple':
            disp(['Object ', num2str(x), ' is an apple.']);
            disp([' ']);
            % We increment the value of variable in which we store the
            % number of counted apples:
            apple = apple + 1;

      % If the form factor of the object is smaller than 0.85:
      elseif (f < 0.85)
            % We display the sentence: 'Object <number of object> is
            % a key':
            disp(['Object ', num2str(x), ' is a key.']);
            disp([' ']);
            % We increment the value of variable in which we store the
            % number of counted keys:
            key = key + 1;

      % We know that if the form factor has an value which is almost
      % equal to 1, then the object is round.
      % If the form factor of the object is bigger than 0.9:
      elseif (f > 0.9)
            % We display the sentence: 'bject <number of object> is
            % a coin':
            disp(['Object ', num2str(x), ' is a coin.']);
            disp([' ']);
            % We increment the value of variable in which we store the
            % number of counted coins:
            coin = coin + 1;
      % We end the if-statement
      end
% We end the for-loop:
end

% We display the total amount of objects found:
disp(['Total number of counted objects: ', num2str(numlabels)]);
disp(['Counted objects:']);
```

```
        % We display the amount of apples:
        % If there is only one apple counted:
        if (apple == 1)
              disp([num2str(apple), ' apple.']);
        % If there is more than one apple counted:
        else
              disp([num2str(apple), ' apples.']);
        % We end the if-statement:
        end

        % We display the amount of coins:
        % If there is only one coin counted:
        if (coin == 1)
              disp([num2str(coin), ' coin.']);
        % If there is more than one coin counted:
        else
              disp ([num2str(coin), ' coins.']);
        % We end the if-statement:
        end

        % We display the amount of keys:
        % If there is only one key counted:
        if (key == 1)
              disp([num2str(key), ' key.']);
        % If there is more than one key detected:
        else
              disp ([num2str(key), ' keys.']);
        % We end the if-statement:
        end
% We end our function:
end
```

Exercise 1b: *Write a document in which you specify which classes you used, as well as which features and which rules you implemented in your classifier. Include the inputs you tested, and the corresponding classifier's decisions. Also, answer the following question: Do you see any limitations of this method of classification?*

Answer:

The picture below is one we used as an input to test our functions:

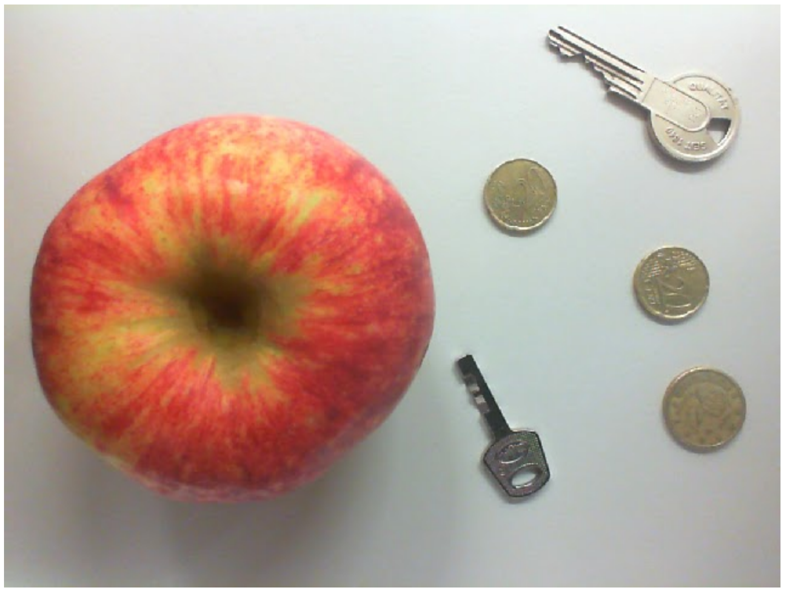


Figure 1   The image we took (also called: applekeycoin)

It is important that the picture is taken approximately 20 to 30 centimeters away from the objects, the objects should not be touching and it should not be too bright. It should be taken straight from above and the background should not have a pattern or other irregularities.

We used three different classes, those are Apple, Key, and Coin. To be able to reliably distinguishes between different classes, we used the form factor and the area of the objects. For the rule-based classifier we used three if conditions to distinguish between the three different classes.

As conditions for the classifier, we chose to distinguish between coins and keys using the form factor. Because the form factor is close to one when it is a round object (thus, in our case: a coin) we set the condition that if the form factor of the current object is bigger than 0,9, it is classified as a coin.
On the other hand, if the form factor is smaller than 0,85, the object gets classified as a key. After that we decided to use the area to classify if an object is an apple, because in relation to a key or a coin this is quite big object. We did not use the form factor for the apple as it looks like a round object in the obtained image.

Our set of conditional if-then rules was the following:

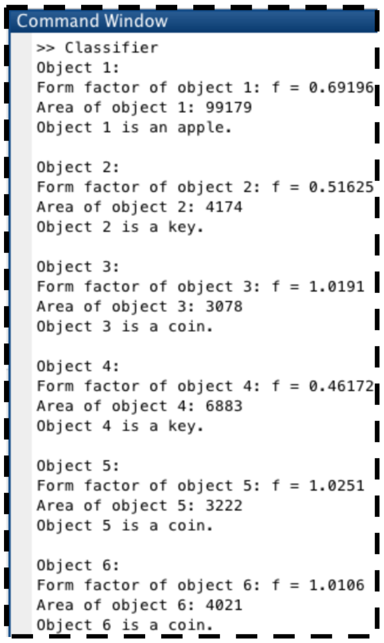
1. If the object had a form factor bigger than 0,9, then we considered the object as a coin.
2. If the form factor was smaller than 0,85, then we considered the object as a key.
3. If the area was bigger than 40000, then we considered the object as an apple.

Table with the features by which we could classify the objects:

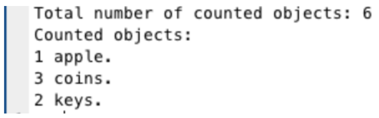
Object	**Area**	**Form factor**

1	99179	0.69196
2	4174	0.51625
3	3078	1.0191
4	6883	0.46172
5	3222	1.0251
6	4021	1.0106

We obtained the information of this table by executing the `Classifier` function. This is the information we got when we executed the function:



```
Command Window
>> Classifier
Object 1:
Form factor of object 1: f = 0.69196
Area of object 1: 99179
Object 1 is an apple.

Object 2:
Form factor of object 2: f = 0.51625
Area of object 2: 4174
Object 2 is a key.

Object 3:
Form factor of object 3: f = 1.0191
Area of object 3: 3078
Object 3 is a coin.

Object 4:
Form factor of object 4: f = 0.46172
Area of object 4: 6883
Object 4 is a key.

Object 5:
Form factor of object 5: f = 1.0251
Area of object 5: 3222
Object 5 is a coin.

Object 6:
Form factor of object 6: f = 1.0106
Area of object 6: 4021
Object 6 is a coin.
```



```
Total number of counted objects: 6
Counted objects:
1 apple.
3 coins.
2 keys.
```

Limitations

If we assume that the picture which was taken is exactly as required, so it contains the following types of objects: Apple, Key, and Coins, then our method works fine.

However, there are limitations for this method of classification, as we are only able to identify the objects based on their form factor and area. This limits us as if we have other objects of the same size or shape, for example a (large) button. Its form factor would be almost equal to one, so it would be classified, using our method, as the coin. Furthermore, as its area is probably almost the same as the coins' area, it would be classified as a coin as well using that feature. But the button is not a coin, so it would not be correctly identified. However, this classification could be achieved with further details about the object, for example its colour, or its brightness.

In addition, if the image that we obtain has been taken incorrectly. i.e. the angle differs or the picture is taken in an environment where it is too bright, causing reflection off of the objects, then there are some risks of misclassification.

The results of this misclassification, in terms of the angle differing can be that coins are misclassified as keys or as an unknown object, as the coins would not retain their round shape, resulting resulting in different values for the form factor (which might not be close to 1 anymore). Thus, the objects would be misclassified.

The results of this misclassification, in terms of the brightness of the environment, can be that, when converting the gray toned image to a black and white picture, some of the pixels are below the threshold that we obtain, which would result in the objects disappearing. This is due to the shiny nature of the objects as light would have been reflected on the objects resulting in them not being detected as objects. Thus, the objects would be misclassified. On the other hand, if the shadows of the objects is prominent, this can alter the shape and size of the object. This results in further misclassification.

Unknown object

We can try to “get rid of” the limitations. To make sure we do not classify an object that does not belong to one of the following classes: Apple, Key, and Coin, we can add another class, which we call “Unknown”. An object belongs to this class if we cannot classify this object according to the set of conditional if-then rules (i.e. if the properties of the object do not match any of the conditional if-then rules).
If we would want to classify objects using the `Classifier` function, we could have used the following segments of codes. However, we did not use these segments in our code. We will explain why we did not use segments of code.

Variable for unknown objects:

```
% We need to initialize a variable in order to store the number of counted
% unknown objects. We use the variable unknown and we initialize it by
% assigning the value zero to it:
unknown = 0;
```

Classify unknown objects:

```
% Else: If none of the definitions for the objects fits an object
% it is classified as unknown:
      else
            % We display the sentence: “Object <number of object> is
            % a unknown”:
            disp(['Object ', num2str(x), ' is an unknown object.']);
            disp([' ']);
            % We increment the value of variable in which we store the
            % number of counted unknown objects:
            unknown  = unknown + 1;
```

Display unknown objects:

```
% We display the amount of unknown objects:
% If there is only one unknown object counted:
if (unknown == 1)
      disp([num2str(unknown), ' unknown object.']);
% If there is more than one unknown object detected:
else
      disp ([num2str(unknown), ' unknown objects.']);
% We end the if-statement:
end
```

Limitation we encounter when we want to use the “Unknown object” class
Unfortunately, we still encounter limitations. As we described earlier, if we have only three classes (“Apple”, “Key”, and “Coin”) and use two features (form factor and area), we might classify certain objects, such as buttons. Buttons, most of the time, are round and therefore, the form factor is close to 1, just like the form factor of coins (which are also round). And that is not all: the area of a button is, we could assume, approximately the same as the are of a coin. As a result, a button would get classified as a coin.

To conclude, even if we use to make sure “unknown objects” we do not get misclassified objects, still we might misclassify certains unknown objects, partly due to the fact that we use too less features.