Lab 09

Basic Image Processing
Fall 2018

Last lab's results

Hopefully you have implemented the k-means segmentation algorithm according to the following signature:

```
function [LUT, M] = mykmeans(S,k)
```

Where

`S` is a matrix containing rows as vectors to be clustered
`k` is the number of cluster center points
`LUT` is the look-up-table, it tells you which row of `S` belong to which cluster center
`M` is the matrix containing the cluster center points

Surprise-surprise! The MATLAB's built-in K-means function is called **`kmeans`** and has the exact same parameters.

For the following exercises we are using the MATLAB's implementation of the k-means algorithm.

Repeat: Spaces & dimensions (in MATLAB)

Let us translate the terms of the previous slide into MATLAB.

Consider an `S` space, represented by a *matrix*.

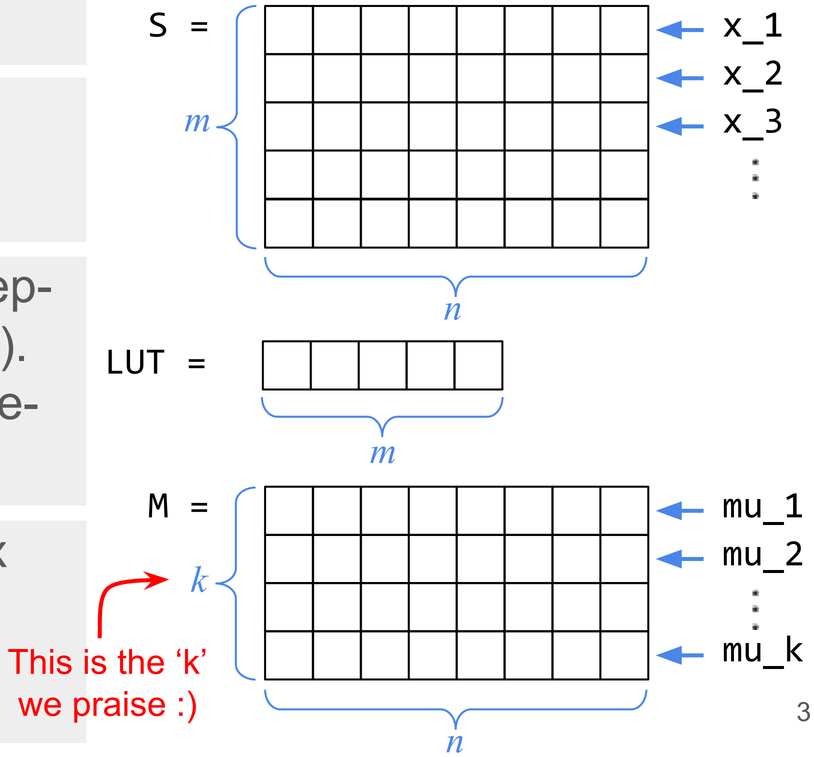
Every *row* of `S` is a vector with n items:

```
S(1,:) = x_1 = [x_11 x_12 … x_1n]
```

The `S_i` subsets of `S` are the clusters. Their representations are stored in a look-up-table (`LUT`). The index represents the index, the value represents the cluster # of a row vector `x_a` of `S`.

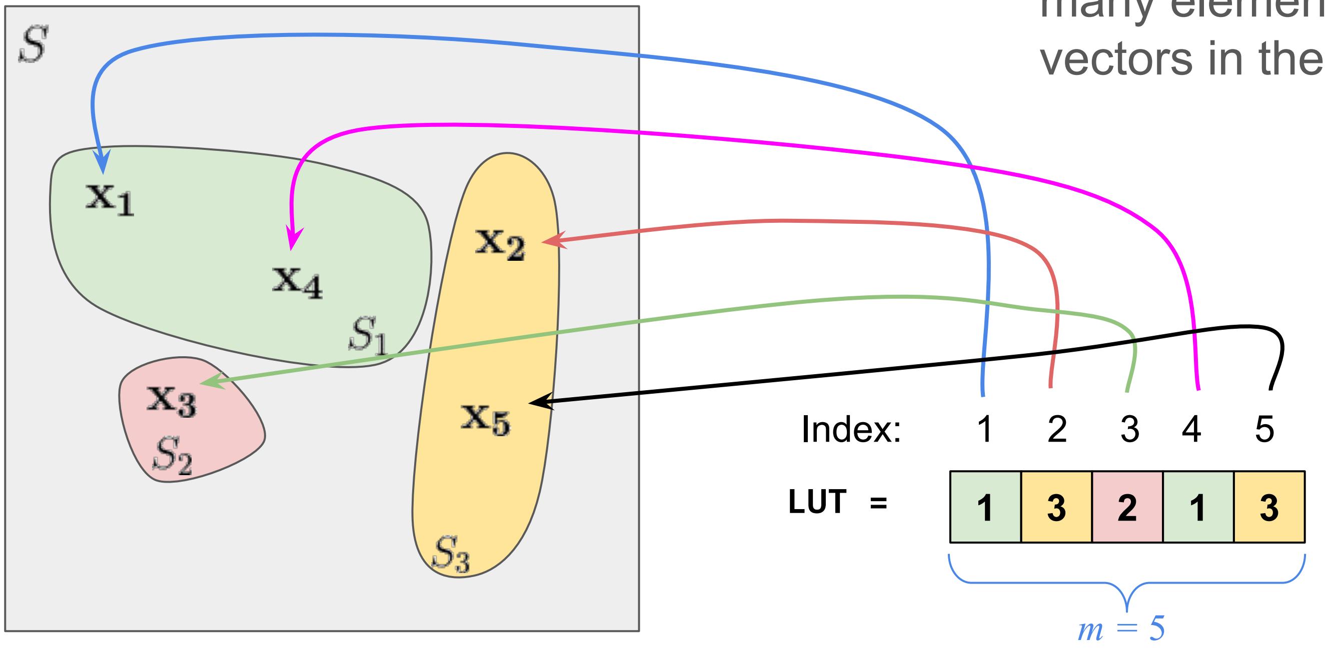
Also, the μ mean vectors are stored in a matrix similar to `S`, denoted by `M`. Its elements are

```
M(j,:) = mu_j = [mu_j1 … mu_jn]
```



<u>Repeat:</u> What is stored in the LUT?

The LUT does the vector-cluster mapping.



The LUT is a vector. It has as many elements as the number of vectors in the space S.

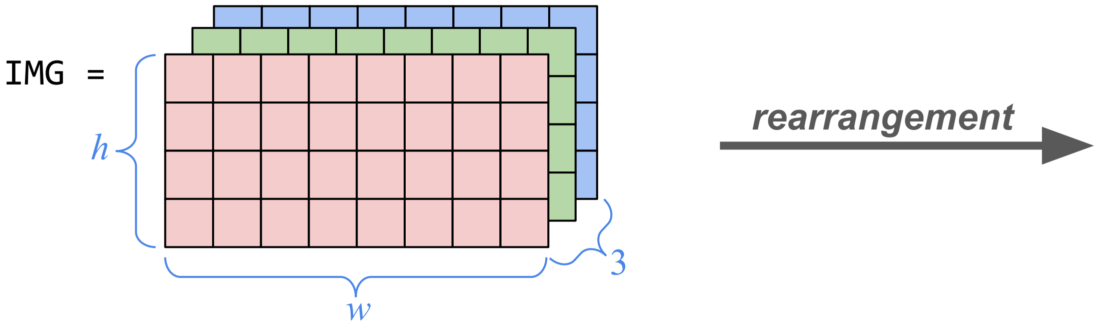
Every element of the LUT has an *index* and a *value*.

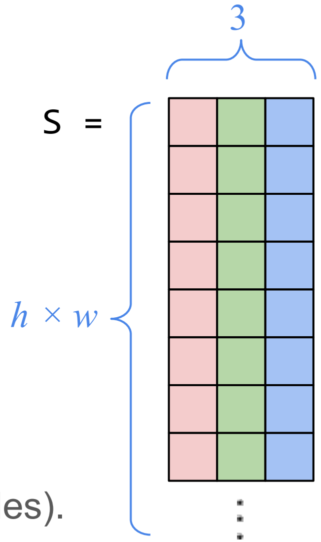
The value at position j tells us which cluster does vector $\mathbf{x}_j$ belong to.

Today we have images as the input

An RGB color image is represented by a 3D matrix. It has h rows, w columns and 3 layers along the 3rd dimension. These layers are the R, G and B color layers.

To be able to cluster an image with the the kmeans function we *somehow* has to transform the 3D matrix into a 2D array of row vectors.





We are using the MATLAB `reshape` function. (See upcoming slides).

Exercise 1

RGB feature space

Implement function `step1_A` in which:

- input: `I` - the image matrix
- output: `S` - the RGB feature-space matrix
- convert the input matrix to have type double
- reshape this 3D matrix into a 2D vector of vectors

In MATLAB a 3D matrix (**`MAT`**) has 3 coordinates:

1.) index of the row → (pixel **`y`** coordinate) `1 ≤ y ≤ size(MAT,1)= h`
2.) index of the column → (pixel **`x`** coordinate) `1 ≤ x ≤ size(MAT,2)= w`
3.) index of the layer → (pixel **`c`** coordinate) `1 ≤ c ≤ size(MAT,3)= d`

The goal is to have a matrix that has

- **`(h*w)`** rows (every row is a pixel), and
- **`d`** columns (every column is a *color* layer)

To rearrange a matrix like this you should use the reshape command with the appropriate parameters:

```
S = reshape(MAT, num_of_rows, num_of_cols)
```

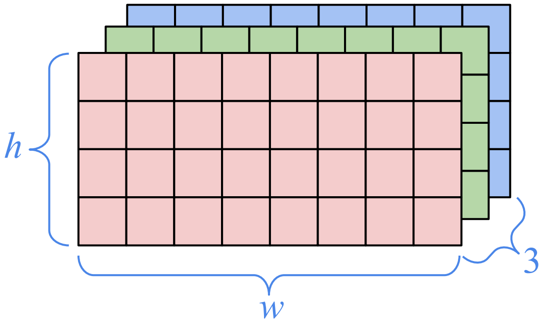
Exercise 2

Moving to RGBXY feature space.

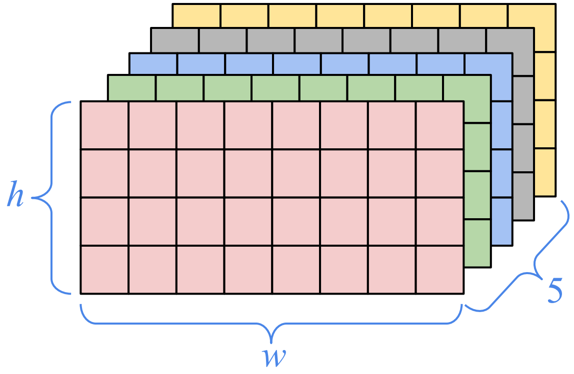
Implement function `step1_B` in which:

- input: `I` - the image matrix
- output: `S` - the RGB feature-space matrix
- convert the input matrix to have type double
- extend this matrix along the 3rd dimension: add two layers where the x and y coordinate of the pixel is stored.
- reshape this 3D matrix into a 2D vector of vectors

RGB

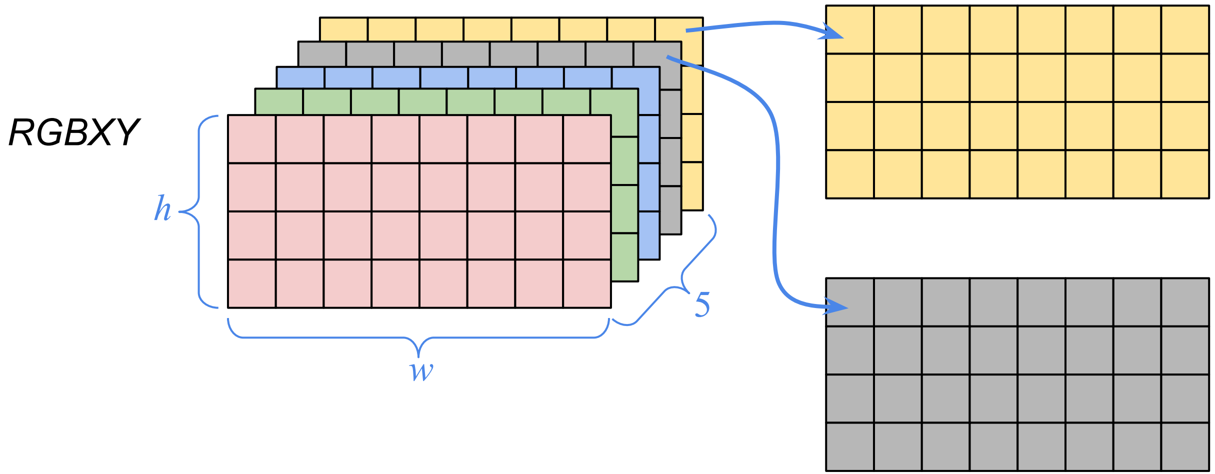


RGBXY



Exercise 2 – continued

Watch out: The X and Y layers are containing the row and column indices:



$$= Y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{bmatrix}$$

$$= X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}$$

One can easily create the matrices X and Y with the help of the `meshgrid` function: `[X, Y] = meshgrid(1:w, 1:h);`

Exercise 3

Moving to H feature space.

Implement function `step1_C` in which:

- input: `I` - the image matrix
- output: `S` - the RGB feature-space matrix
- convert the input matrix from rgb to hsv space (`rgb2hsv`)

- remove the S and V layers of the HSV image (set `I = I(:,:,1)`)
- reshape this newly created 2D matrix into a 2D vector of vectors

Test your `step1_A`, `step1_B`, `step1_C` functions
with **script** `script1_to_step1ABC`

Exercise 4

Do the k-means clustering and replace cluster indices with centroid values

Implement function `step2` in which:

- input: `S` - the feature-space matrix,
  `k` - number of clusters,
- output: `A` - the clustered feature-matrix
- calculate the look-up vector (cluster indices) and centroid values with the built-in `kmeans` function, on the previously calculated feature matrix (`S` vector of vectors from the `I` matrix):

```
[LUT, M] = kmeans(S, k)
```

- replace cluster indices with centroid values -- see next slide

Exercise 4 - continued

The result of the clustering is a look-up-table (`LUT`) and a vector of centroid vectors (`M`).

You have to index `M` with `LUT` to create a new array (`A`), that has the same size as `S` and instead of the cluster center indices it contains the cluster center point vectors copied to the appropriate positions. **See next slide!**

MATLAB hint: You can index a vector logically! If the LUT is a vector and you write `LUT == 1` then this expression will return a logical vector: 1 if the element == 1, 0 otherwise.

If `A = [1 2 3 1 1 2 1]` then `A == 1` returns `[1 0 0 1 1 0 1]`

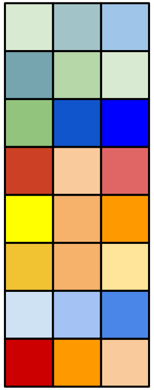
The other trick is that if you index a vector or matrix with a logical vector, the result will be the set of those elements that has the same indices where the logical vector contained 1-s.

If `B = [1 2 3 4 5 6 7 ; 2 0 2 0 1 0 7]` then `B(:,[1 0 0 1 1 0 1])` returns `[1 4 5 7 ; 2 0 1 7]`

Exercise 4 - continued

Before clustering

S =



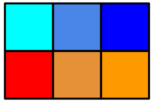
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8

Result of the clustering

k = 2

Index:	1	2	3	4	5	6	7	8
LUT =	1	1	1	2	2	2	1	2

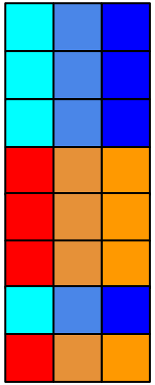
M =



mu_1
mu_2

Replacement result

A =



a_1 = mu_1
a_2 = mu_1
a_3 = mu_1
a_4 = mu_2
a_5 = mu_2
a_6 = mu_2
a_7 = mu_1
a_8 = mu_2

Exercise 5

The previous step gives you an `A` matrix that is 2D and which should be reshaped to a 3D structure that has the same size as the original input (`I`).

The goal is to have a matrix that has

- `h` rows
- `w` columns
- `d` layers

To rearrange a matrix like this you should use the reshape command with the appropriate parameters:

```
SEG = reshape(A, num_of_rows, num_of_cols, num_of_layers)
```

Implement function `step3_A` in which:

- input: `A` - the clustered feature-matrix,
  `I` - original image,
- output: `SEG` - the segmented image itself
- reshape your clustered RGB-feature matrix (`A`) to the size of your original image (`I`),
- convert the datatype back to `uint8`.

Exercise 6

Implement function `step3_B` in which:

- input: `A` - the clustered feature-matrix,
  `I` - original image,
- output: `SEG` - the segmented image itself
- reshape your clustered RGBXY-feature matrix (`A`) to the size of your original image (`I`),
  - remember, your 3rd dimension should contain 5 layers: `reshape` takes care of the order of data, but after reordering, get rid of layers 4&5:
    `SEG = SEG(:, :, 1:3)`
- convert the datatype back to `uint8`.

Exercise 7

Implement function `step3_C` in which:

- input: `A` - the clustered feature-matrix,
  `I` - original image,
- output: `SEG` - the segmented image itself
- reshape your clustered H-feature matrix (`A`) to the size of your original image (`I`),
  - remember, your 3rd dimension should contain 1 layer only
  - after reordering, extend the shallow matrix with two layers each of them containing 0.7 as value (set `SEG(:,:,2:3)` to `0.7`)
- convert back your array from the HSV-space to the RGB-space (`hsv2rgb`)
  - the content of your data will be in the range [0, 1], so scale it to the range [0, 255] and
  - convert the datatype back to `uint8`.

Test your `step3_A`, `step3_B`, `step3_C` functions
with **script** `script3_to_step3ABC`

Exercise 8

The complete RGB-space segmentation

Implement function `pixel_based_segmentation_with_kmeans_A` in which:

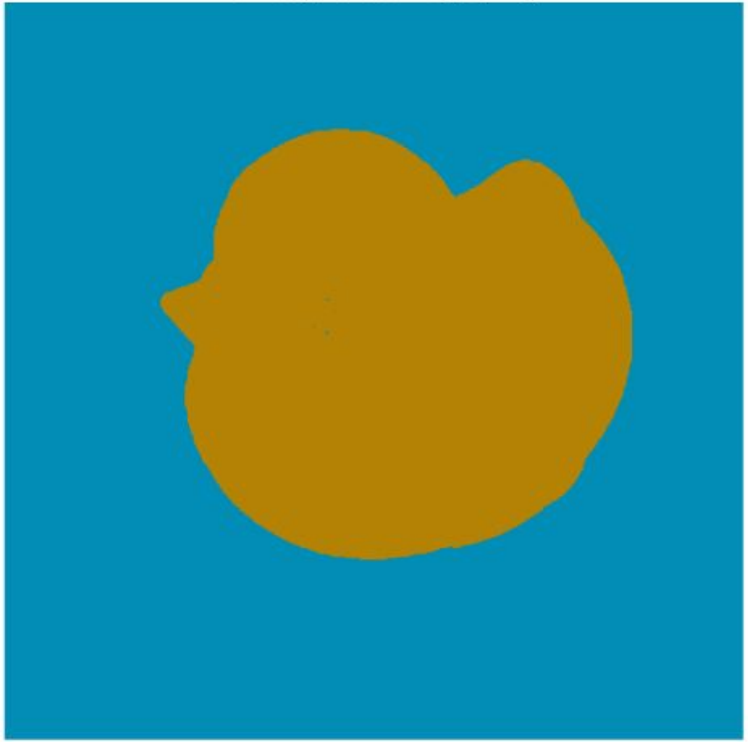
- input: `I` - original image,
  `k` - number of clusters,
- output: `SEG` - the segmented image itself
- with function `step1_A` produce the feature matrix in RGB-space
- with function `step2` convert your feature matrix to a clustered feature matrix
- with function `step3_A` restore the original shape of your image on the clustered feature matrix data

Run it with `script4` to test your RGB-segmentation process on the duck-image (*see next slide*).

Exercise 8 - continued

Input image

RGB segmented image (k=2)



Exercise 9

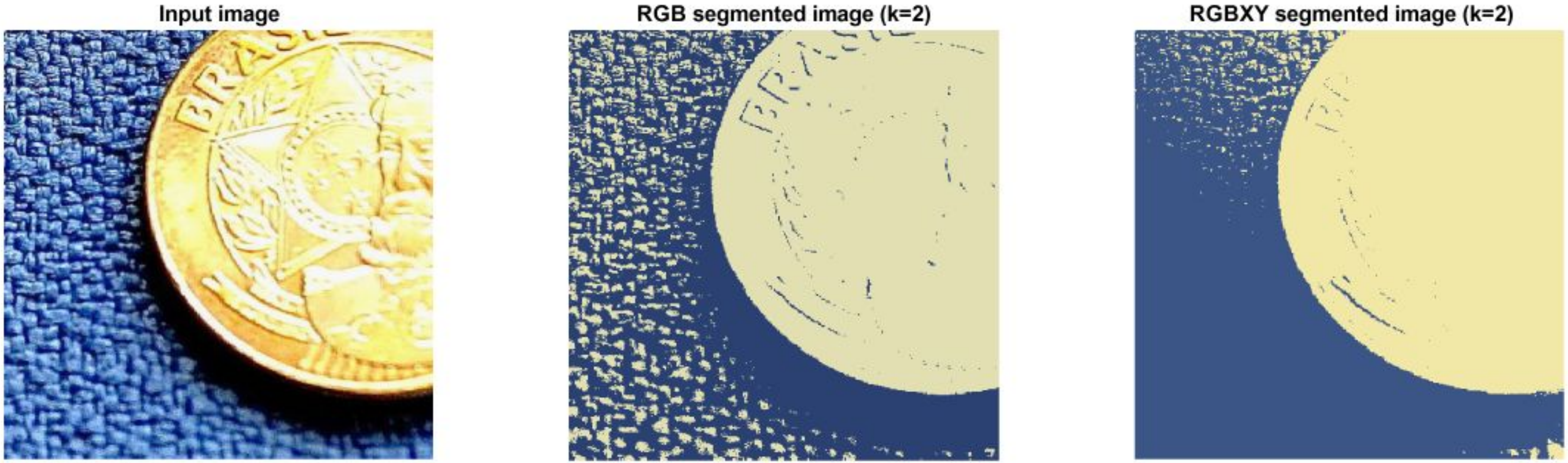
The complete RGBXY-space segmentation

Implement function `pixel_based_segmentation_with_kmeans_B` in which:

- input: `I` - original image,
  `k` - number of clusters,
- output: `SEG` - the segmented image itself
- with function `step1_B` produce the feature matrix in RGBXY-space
- with function `step2` convert your feature matrix to a clustered feature matrix
- with function `step3_B` restore the original shape of your image on the clustered feature matrix data

Run it with `script5` to test your RGBXY-segmentation process on the coin-image (*see next slide*).

Exercise 9 - continued



Exercise 10

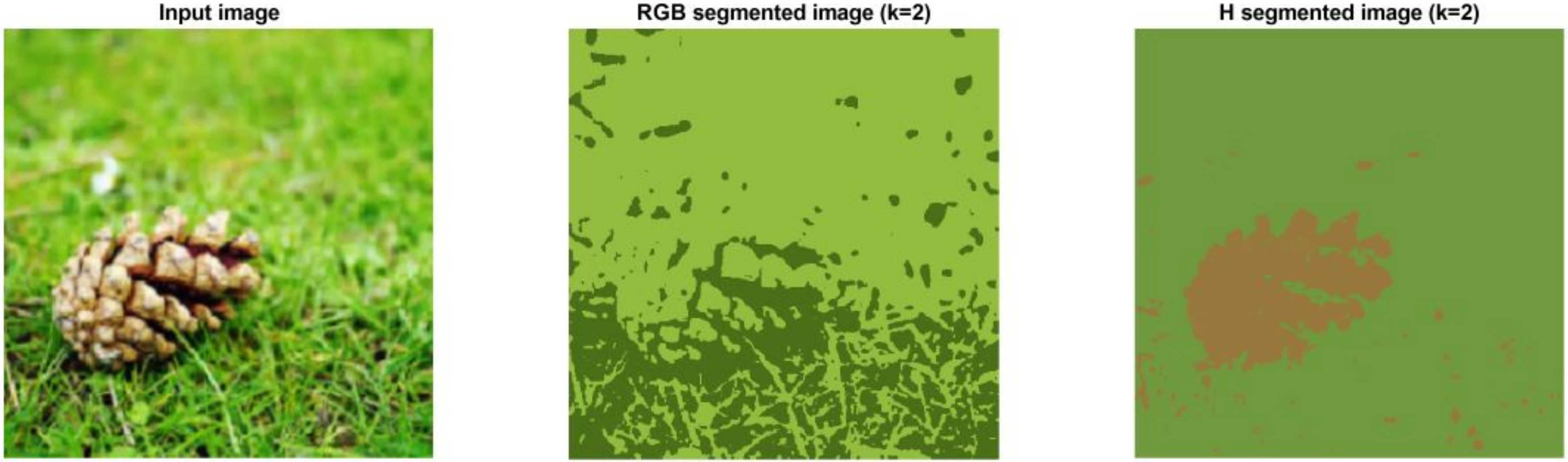
The complete H-space segmentation

Implement function `pixel_based_segmentation_with_kmeans_C` in which:

- input: `I` - original image,
  `k` - number of clusters,
- output: `SEG` - the segmented image itself
- with function `step1_C` produce the feature matrix in H-space
- with function `step2` convert your feature matrix to a clustered feature matrix
- with function `step3_C` restore the original shape of your image on the clustered feature matrix data

Run it with `script6` to test your H-segmentation process on the toboz-image (*see next slide*).

Exercise 10 - continued



THE END