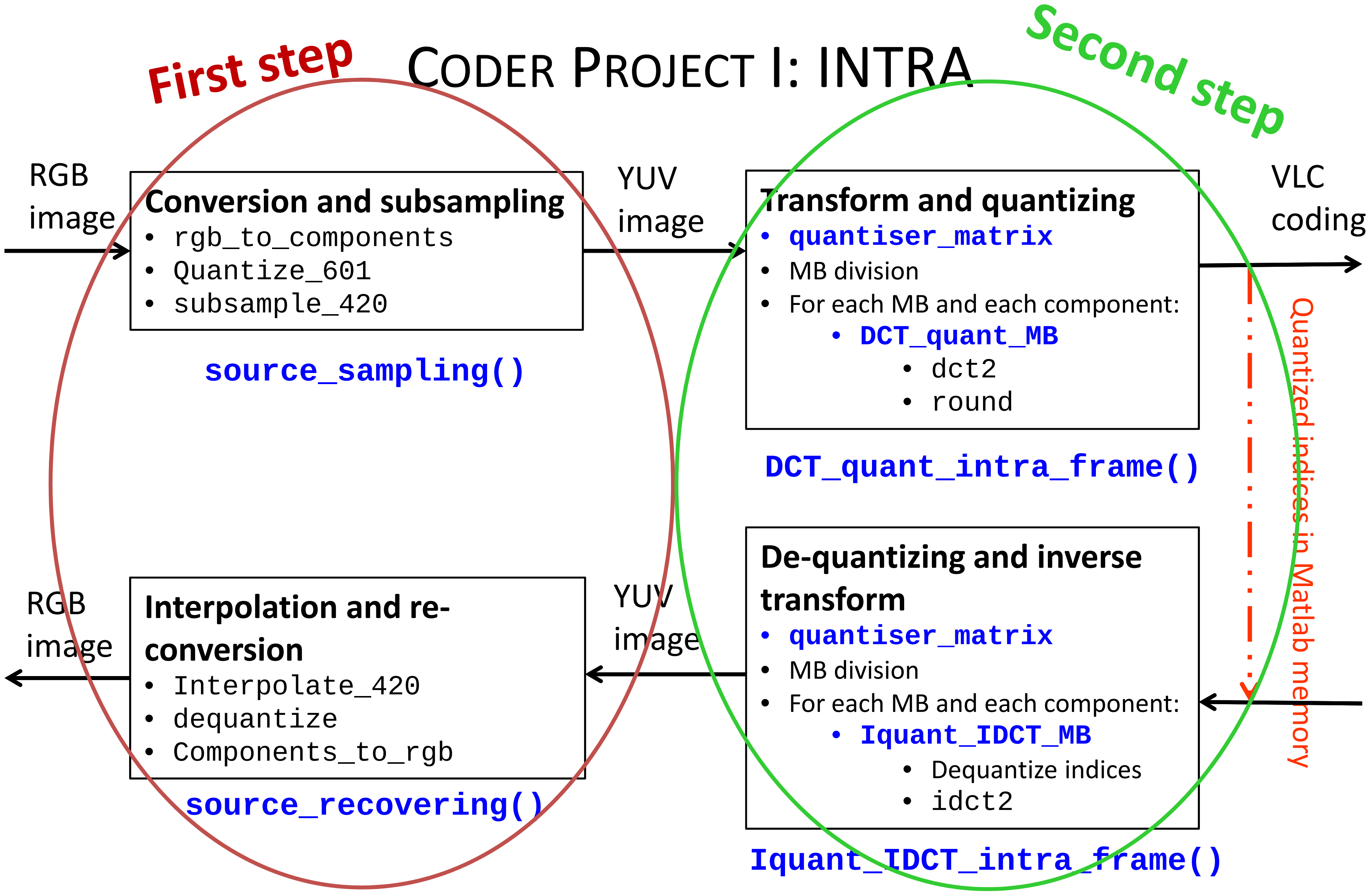
Session 3

CODER PROJECT PART I: INTRA CODING

CODER PROJECT I: INTRA



First step

Second step

RGB image

Conversion and subsampling
- `rgb_to_components`
- `Quantize_601`
- `subsample_420`

`source_sampling()`

YUV image

Transform and quantizing
- **`quantiser_matrix`**
- MB division
- For each MB and each component:
  - **`DCT_quant_MB`**
    - `dct2`
    - `round`

`DCT_quant_intra_frame()`

VLC coding

Quantized indices in Matlab memory

De-quantizing and inverse transform
- **`quantiser_matrix`**
- MB division
- For each MB and each component:
  - **`Iquant_IDCT_MB`**
    - Dequantize indices
    - `idct2`

`Iquant_IDCT_intra_frame()`

YUV image

Interpolation and re-conversion
- `Interpolate_420`
- `dequantize`
- `Components_to_rgb`

`source_recovering()`

RGB image

Coder Project Part I: INTRA

Implement the following funcions:

First step: `source_sampling()` and `source_recovering()`

1. Conversion to components (**done**)
2. Quantizing as 601 standard
3. Sampling 4:2:0
4. Inverse process for decoder

Then testing!!!

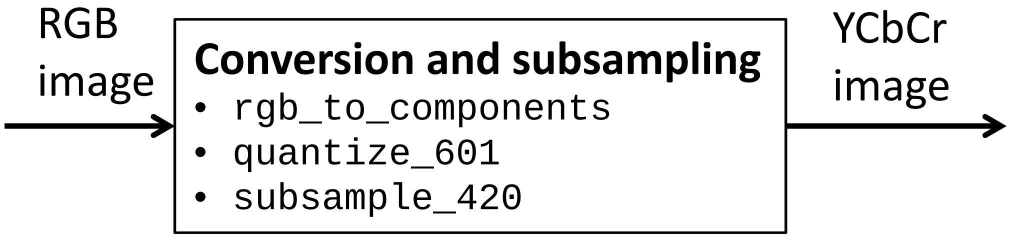
If it works:

Second step: `DCT_Quant_Intra_Frame()`and `Iquant_IDCT_Intra_Frame()`

1. Read MB
2. Scan the image MB to MB
3. Transform and quantizing of each MB (all three components)
4. Inverse process for decoder

First step

CODER PROJECT PART I: INTRA



RGB image

Conversion and subsampling

- `rgb_to_components`
- `quantize_601`
- `subsample_420`

YCbCr image

`source_sampling()`

Input:

- RGB image
- Parameter for filtering option previous to subsampling

Output:

- Y, Cb, Cr in 4:2:0 format (**`double`**)

The function **`source_sampling()`** should do the following:

- Convert the input image to components, call **`rgb_to_components()`**
- Include a call to sub-function **`quantize_601()`**
- Include a call to sub-function **`subsample_420()`**the subsampling process to 4:2:0 format

First step

source_sampling()

```
function [Y420, Cb420, Cr420] =
source_sampling(image_RGB, filter)
% Takes an RGB image and transforms it to a 4:2:0 source
for MPEG coding
```

Input:

- **image_RGB**: RGB image of type **uint8** or **double**
- **filter**: 1: cosited filter previous to color subsampling, neq 1: no filtering

Output: (**double**)

- **Y420**: Luma signal without subsampling
- **Cb420**, **Cr420**: Chroma signals subsampled as 4:2:0

Calls to:

- **rgb_to_components**
- **quantize_601**
- **subsample_420**

First step

quantize_601()

```
function [Yq, Cbq, Crq] = quantize_601(Y,Cb,Cr)
```

Input: (`double`)

- `Y` Luma signal 4:4:4 according to 601
- `Cb`, `Cr`: Chroma signals 4:4:4 according to 601

Output: (`double`)

- `Yq` quantized luma signal in range
- `Cbq`, `Crq`: quantized chroma signal in range

Notes:

- Apply quantizing according to standard 601 for 8 bits as shown in theory
- Remind offset for footroom and headroom

First step

subsample_420()

```
function [Y420, Cb420, Cr420] =
subsampling420(Y,Cb,Cr,filter)
```

Input: (uint8 or double)

- Y Luma signal coded according to standard 601
- Cb,Cr Color differences coded according to standard 601
- filter: 1: **cosited filter** previous to color subsampling, neq 1: no filtering

Output: (double)

- Y420 Luma signal without subsampling
- Cb420, Cr420: Chroma signals subsampled as 4:2:0

Notes:

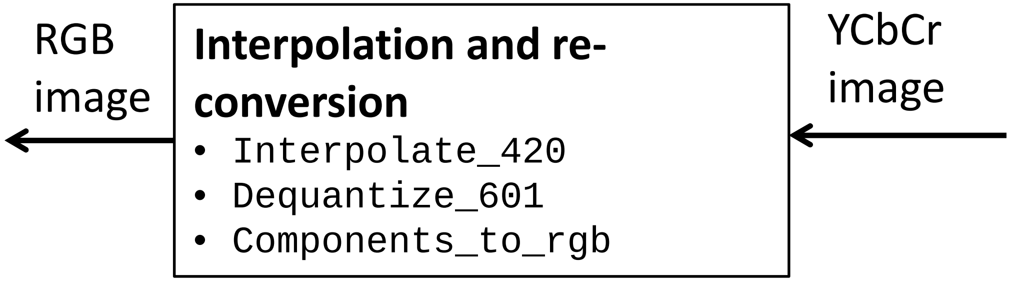
- Use the function downscale()
- Check that the input signals have the expected size, the function should not admit odd sized images because in the recovering step, the size would not be correct.
- Filter: use imfilter() with the option ‘replicate’ and a mask according to the cosite filtering seen in theory.

First step

CODER PROJECT PART I: INTRA

The function **source_recovering()**incorporates the inverse process:

- Interpolation from 4:2:0 to 4:4:4 with replicate or bilinear (integrate in call to function **interpolate_420()**)
- De-quantizing and reconverting from 601 format (integrate in call to function **dequantize_601()**)
- Conversion from components to RGB, call **components_to_rgb()**



RGB image

Interpolation and re-conversion

- `Interpolate_420`
- `Dequantize_601`
- `Components_to_rgb`

YCbCr image

source_recovering()

Input:

- Y, Cb, Cr in 4:2:0 format (**double**)
- Interpolation flag (replicate=0, bilinear=1)

Output:

- RGB image (**uint8**)

First step

source_recovering()

```
function image_rec =
source_recovering(Y420,Cb420,Cr420,order)
% Recovers an RGB image from a 4:2:0 source
```

Input:

- `Y420`: Luma signal without subsampling
- `Cb420`, `Cr420`: Chroma signals subsampled as 4:2:0
- `order`: 0: use replicates, 1: use bilinear interpolation of chroma samples

Output: (`double`)

- `image_rec`: RGB image of type `uint8`

Calls to:

- `interpolate_420()`
- `dequantize_601()`
- `components_to_rgb()`

First step

interpolate_420()

```
function [Y444, Cb444, Cr444] =
interpolate_420(Y,Cb,Cr,order)
```

Input:

- `Y` Luma signal coded according to standard 601
- `Cb,Cr` Color differences coded according to standard 601
- `Order` 0: use replicates, 1: use bilinear interpolation of chroma samples

Output: (`double`)

- `Y444` Recovered luma signal (same as input Y)
- `Cb444`, `Cr444` Chroma signals recovered to original size

Notes:

- Use functions `replicate()`and `bilinear()`

First step

dequantize_601()

```
function [Y,Cb,Cr] = dequantize_601(Yq,Cbq,Crq)
```

Input:

- **Yq** quantized luma signal 4:4:4 according to 601
- **Cbq**, **Crq** quantized chroma signals 4:4:4 according to 601

Output:

- **Y** dequantized luma signal 4:4:4
- **Cb**, **Cr** dequantized chroma signals 4:4:4

First step

For testing

residual()

This function calculates the difference between two input images, called residual or error image.

```
function im_dif = residual(im_orig, im_det)
```

Input: (uint8 or double)

- im_orig Original image
- im_det Deteriorated image

Output: (double)

- im_dif amplified difference image (uint8)

Notes:

- Amplify the difference and convert to uint8 to create a visible output image

First step

TEST PROGRAM – STEP 1

test_sender_step1.m

```
clear; close all; clc;
image = read_image(); % kept in memory
imshow(image), title('original image');
filter = input('Filtering? 0:no, 1:yes');
[Y420, Cb420, Cr420] = source_sampling(image, filter);
% kept in memory
```

Then start the receiver and see if the image is correctly recovered

test_receiver_step1.m

```
im_rec = source_recovering_420(Y420, Cb420, Cr420, 0);
pnsr_rgb = psnr_rgb(im_rec, image)
[psnrY, psnrC] = psnr_ycbcr(im_rec, image)
figure; imshow(im_rec), title('recovered image');
im_error = residual(image, im_rec);
figure; imshow(im_error), title('Error image');
```

EVALUATION

Compress the **required functions** (no report this time) in a zip-file and send per **e-mail** to the professor (martina.aux@gmail.com).

- **source_sampling**
- **rgb_to_components**
- **quantize_601**
- **subsample_420**
- **source_recovering**
- **Components_to_rgb**
- **interpolate_420**
- **dequantize_601**
- **residual**

Until Monday 13th of May

- The zip-file should be carry your named or ID.