Signal and System CA3

Part 1.

1-1) First we create our Mapset with needed characters.

```
characters = ['a':'z', ' ', '.', ',', '!', '"', ';'];
Mapset = cell(2, 32);
Mapset(1, :) = num2cell(characters);
for i = 1 : length(characters)
    Mapset(2, i) = {num2str(dec2bin(i-1, 5))};
end

disp(Mapset);
```

```
  Columns 1 through 12

    {'a'    }    {'b'    }    {'c'    }    {'d'    }    {'e'    }    {'f'    }    {'g'    }    {'h'    }    {'i'
    {'00000'}    {'00001'}    {'00010'}    {'00011'}    {'00100'}    {'00101'}    {'00110'}    {'00111'}    {'01000'

  Columns 13 through 24

    {'m'    }    {'n'    }    {'o'    }    {'p'    }    {'q'    }    {'r'    }    {'s'    }    {'t'    }    {'u'
    {'01100'}    {'01101'}    {'01110'}    {'01111'}    {'10000'}    {'10001'}    {'10010'}    {'10011'}    {'10100'

  Columns 25 through 32

    {'y'    }    {'z'    }    {' '    }    {'.'    }    {','    }    {'!'    }    {'"'    }    {';'    }
    {'11000'}    {'11001'}    {'11010'}    {'11011'}    {'11100'}    {'11101'}    {'11110'}    {'11111'}
```

1-2) In `coding` function first we handle the errors such as not finding the character inside Mapset or the message being longer than the image. When checking for Mapset, we also translate the input message to the corresponding binary string. Then we put each bit of binary string inside the pixels LSB using `bitset` function.

```
path = './data/';
file = 'pic.png';
source = [path, file];
orgPicture = imread(source);
orgPicture = rgb2gray(orgPicture);
codedPicture = coding(orgPicture, 'signal;', Mapset);
```

1-3) Now we use `subplot` to show both original and coded pictures. We cannot observe the difference between them because we put the message in the Least Significant Bit of each pixel so the change is not very significant for the pixel valua and also the message is small in scale of the image.

```
subplot(1, 2, 1);
imshow(orgPicture);
subplot(1, 2, 2);
imshow(codedPicture);
```

1-4) After encoding the image, we use `decoding` function to extract the encoded message. We iterate through pixels and for each pixel we get the LCB, which is our embedded bit, using `getbit` function. Then for each 5 bit which is our encoded character, we find the corresponding character in Mapset and add it to the output message. We stop extracting the bits when we hit 11111 which corresponds to semicolon ";" and shows our message's end.

```
msg = decoding(codedPicture, Mapset);
disp(msg);
```

```
signal;
```

1-5) Since we are encoding the bits inside the Least Significant Bit of each pixel, with even small noises, the value of LCB bit of pixel can change, so the message can be corrupted and we would not be able to decode it.

Functions implementations

```
function enhancedImage = coding(image, message, Mapset)
    binaryMessage = '';
    for i = 1:length(message)
        index = find(strcmp(Mapset(1,:), message(i)));
        if isempty(index)
            error("Character not found inside mapset.");
        end
        binaryMessage = [binaryMessage, Mapset{2, index}];
    end

    if size(image, 3) == 3
        image = myGrayFunc(image);
    end
    enhancedImage = image;
    binaryImage = dec2bin(image, 8);
    [rows, cols] = size(image);
    if length(binaryMessage) > rows*cols
        error('The message is too long to fit in the image.');
    end

    for i = 1:length(binaryMessage)
        row = ceil(i / cols);
```

```
        col = mod(i-1, cols) + 1;

        enhancedImage(row, col) = bitset(image(row, col), 1, str2num(binaryMessage(i)));
    end

    enhancedImage = uint8(enhancedImage);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function message = decoding(codedImage, Mapset)
    codedImage = double(codedImage);
    message = '';
    [rows, cols] = size(codedImage);
    message_bin = zeros(rows*cols, 1);
    for i = 1:rows*cols
        row = ceil(i / cols);
        col = mod(i-1, cols) + 1;
        message_bin(i) = bitget(codedImage(row, col), 1);
        if mod(i, 5) == 0
            chars = '';
            str = message_bin(i-4:i);
            for j = 1:5
                chars = [chars, num2str(str(j))];
            end
            index = find(strcmp(Mapset(2,:), chars));
            message = [message, Mapset{1, index}];
            if all(str == 1)
                break;
            end
        end

    end
end
```

Signal and System CA3

Part 2.

2-1: Synthesis)

First we make our Dataset of all 12 buttons to work with it later.

- Each row contains the label of the button (its actual number or symbol), its corresponding wave generated with given values and formula in the project description and its corresponding `fr` and `fc`.

```
fr = [697 770 852 941];
fc = [1209 1336 1477];
fs = 8000;
Ts = 1/fs;
Ton = 0.1;
Toff = 0.1;
t = 0:Ts:Ton;

labels = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '*', '0', '#'};
Dataset = cell(4*3, 4);
counter = 1;

for k = 1:4
    for j = 1:3
        y1 = sin(2*pi*fr(k)*t);
        y2 = sin(2*pi*fc(j)*t);
        output = (y1 + y2) / 2;

        Dataset{counter, 1} = labels{counter};
        Dataset{counter, 2} = output;
        Dataset{counter, 3} = fr(k);
        Dataset{counter, 4} = fc(j);

        counter = counter + 1;
    end
end
```

- Now we iterate through input number and for each digit, add its corresponding wave to `output` by using our Dataset. Between each digit's wave, we also add a `Toff` 0.1 second silence which is just a all-zero wave.
- After playing it, we save it to the `y.wav` file in `exports` directory.

```
input = '43218765';
output = [];
samples_per_toff = round(Toff * fs) + 1;
silence = zeros(1, samples_per_toff);

for i = 1:length(input)
    row = find(strcmp(Dataset(:, 1), input(i)));
```

```
    sound_wave = Dataset{row, 2};

    output = [output sound_wave];
    output = [output silence];
end

sound(output, fs);
audiowrite("exports/y.wav", output, fs);
```

2-2: Analysis)

- First we make the length of the `Ton` and `Toff` segments. Each `Ton + Toff` segment is the digit and the following silence. So we make our steps `Ton + Toff` and in each part and only choose the `Ton` part which is the digit's wave.
- After getting a segment, we corrolate this segment with our Dataset using `xcorr` function to compute the cross-correlation between wave and the Dataset and choose the best match and submit it as the digit.

```
% [y, fs] = audioread('exports/y.wav');
[y, fs] = audioread('data/a.wav');

digits = '';

samples_per_ton = round(Ton * fs) + 1;
samples_per_toff = round(Toff * fs) + 1;

for i = 1:(samples_per_ton+samples_per_toff):length(y)
    y_segment = y(i:min(i+samples_per_ton-1, end));

    max_corr = -Inf;
    best_match = '';

    for j = 1:size(Dataset, 1)
        sound_wave = Dataset{j, 2};
        corr = max(xcorr(y_segment, sound_wave));

        if corr > max_corr
            max_corr = corr;
            best_match = Dataset{j, 1};
        end
    end

    digits = [digits best_match];
end
```

- Now we display the corrolated digits for the `a.wav`.

```
disp(digits);
```

```
810198
```

Signal and System CA3

Part 3.

- First we receive the input images.

```
path = './data/';
file = 'pcb.jpg';
source = [path, file];
PCB = imread(source);
file = 'ic.png';
source = [path, file];
IC = imread(source);
```

- We should make a upside down version of IC imaged using builtin function `imrotate`.

```
rotated_IC = imrotate(IC, 180);
```

- Then convert the images to gray for doing corrolation.

```
gray_IC = rgb2gray(IC);
gray_PCB = rgb2gray(PCB);
gray_rotated_IC = rgb2gray(rotated_IC);
```

- Now we use `corr_matrix` function to calculate corrolation of the IC and PCB.
- This function gets gray format of the IC and PCB and convert the IC to double precision for better calculation.
- Now we have to normalize `gray_IC`. So we calculate the average of `gray_IC` and subtracts this average from `gray_IC` itself.
- Now we can start finding the ICs. We iterate through the 2d matrix of the PCB and in each iteration, we crop a segment, equal to size of the IC. Then we normalize this cropped segment with same technique in the previous part.
- Now we can calculate the corrolation of the cropped part and IC itself using `corr_coef` function. This function implements the formula provided in the project description. Then store this calculated corrolation and store it in `corr_result`.
- We do this for both the IC image and its rotated version.

$$\text{Correlation Coeff}(x, y) = \frac{\sum_{n=1}^{L} x[n]y[n]}{(\sqrt{\sum_{n=1}^{L} x^2[n]}) \times (\sum_{k=1}^{L} y^2[k])}$$

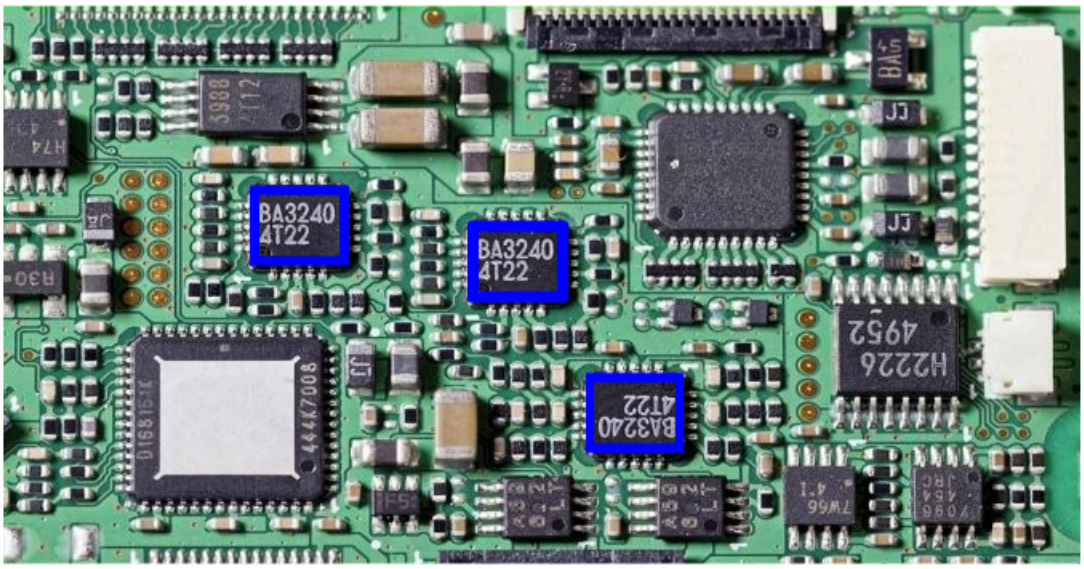
```
normal_res = corr_images(gray_IC , gray_PCB);
rotated_res = corr_images(gray_rotated_IC , gray_PCB);
```

- Now we merge the results in `result` variable for simpler handling of both version's result.

```
result = {normal_res ; rotated_res};
```

- We can draw the blue boxes around each found IC in the image using `draw_boxes` function. This function gets the PCB and IC images and the results of corrolation. This function sets a threshold and only if the calculated corrolation is bigger than this threshold, it is considred as an found IC.
- This threshold is found practical and we didn't use any percise method to calculate it.
- After this, we use the location of the corrolated result to draw the blue rectangles on the PCB image and also return it.

```
final_image = draw_boxes(PCB, gray_IC, result);
```

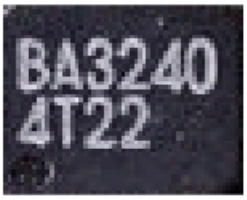


- In the end we show all the images and and the answer.

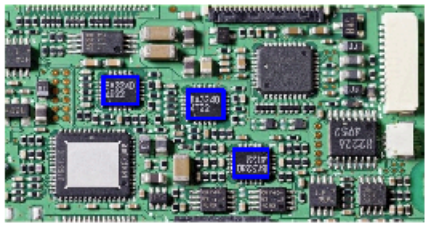
```
figure();
subplot(2, 3, 1), imshow(PCB), title('PCB');
subplot(2, 3, 3), imshow(IC), title('IC');
subplot(2, 3, [4,5,6]), imshow(final_image), title('Found ICs');
```

PCB

IC



Found ICs



```
function corr_coef = corr_2d(x, y)
   corr_coef = sum(x.*y, 'all') / sqrt(sum(x.*x, 'all') * sum(y.*y, 'all'));
end

function output = my_normalize(image, width, height)
    output = image -(sum(image , 'all')/(width * height));
end

function corr_result  = corr_images(gray_IC , gray_PCB)
  [IC_width , IC_height] = size(gray_IC);
  [PCB_width , PCB_height] = size(gray_PCB);
  gray_IC = double(gray_IC);
  corr_result = zeros(PCB_width-IC_width+1 , PCB_height-IC_height+1);
  normal_gray_IC = my_normalize(gray_IC, IC_width, IC_height);
  for i = 1 : PCB_width - IC_width + 1
      for j = 1 : PCB_height- IC_height + 1
         cropped_PCB =  double(gray_PCB(i:i+IC_width-1 , j:j+IC_height-1));
         normal_cropped = my_normalize(cropped_PCB, IC_width, IC_height);
         corr_calc = corr_2d(normal_cropped , normal_gray_IC);
         corr_result(i , j) = corr_calc;
      end
  end
end

function final_image = draw_boxes(PCB, IC, result)
  threshold = 0.65;
  [IC_width, IC_height] = size(IC);
```

```
  figure();
  imshow(PCB);
  hold on;
  for l = 1:length(result)
      [row , col] = find(result{l , 1} > threshold);
       for k = 1 : length(row)
         rectangle('Position', [col(k) row(k) IC_height-1  IC_width-1] , 'Edgecolor' , 'b' , 'L
       end
  end
  final_image = frame2im(getframe(gcf));
end
```