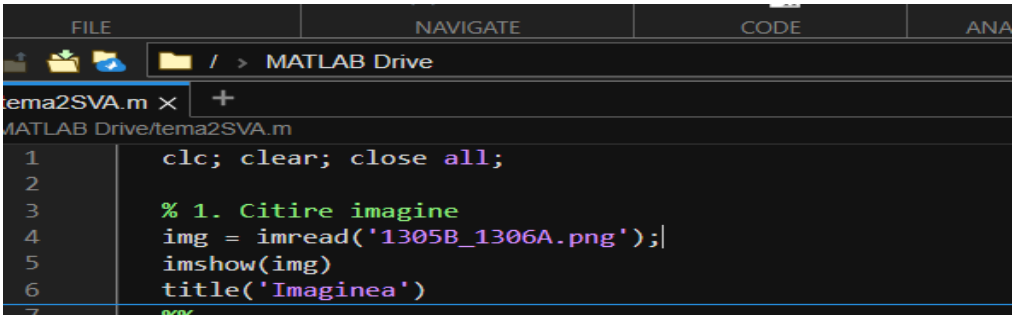
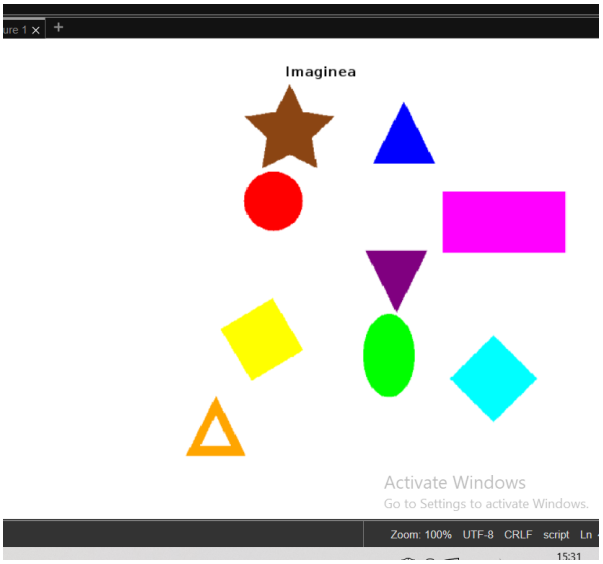
Cojocariu Alexandra-Maria 1305B

Tema 2

Pasul 1

Am descărcat imaginea 1305B_1306A.png și am citit-o în matlab, stocând-o în variabila img.

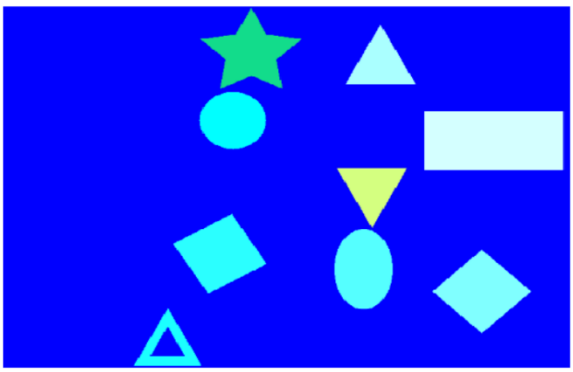




Pasul 2

În continuare vom converti imaginea în format RGB în HSV (Hue, Saturation, Value) pentru o segmentare mai precisă și robustă a culorilor.

```
hsv_img = rgb2hsv(img);
imshow(hsv_img);
```



Cojocariu Alexandra-Maria 1305B

Pasul 3

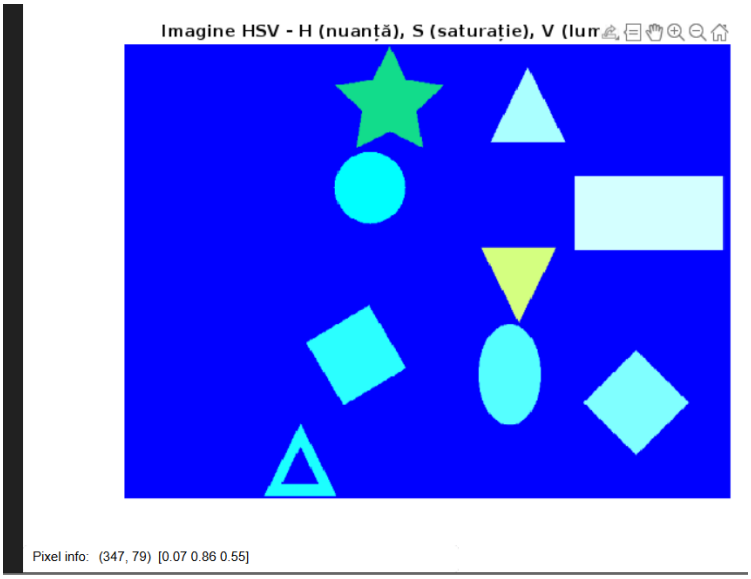
Am definit praguri pentru detectarea culorilor figurilor din imagine.Am folosit celula array colorRange pentru o stoca informatii despre culorile formelor și intervalele componentelor H,S,V pentru fiecare.

Structura fiecărui element (rând) este: {'NumeCuloare', [H_min H_max], [S_min S_max], [V_min V_max]}.

```
% 3. Definim praguri HSV pentru fiecare culoare
colorRanges = {
'Roșu',       [0.95 0.05], [0.5 1], [0.4 1];
'Galben',     [0.13 0.18],   [0.8 1], [0.8 1];
'Albastru',   [0.58 0.70], [0.5 1], [0.4 1];
'Mov',        [0.7 0.85], [0.4 1], [0.1 0.6];
'Verde',      [0.25 0.45], [0.4 1], [0.3 1];
'Cyan',       [0.48 0.55], [0.4 1], [0.4 1];
'Maro',       [0.04 0.09],   [0.5 1], [0.3 0.7];
'Portocaliu', [0.095 0.125], [0.8 1], [0.8 1];
'Roz',        [0.83 0.95], [0.5 1], [0.75 1];
};
```

Urmatoarele linii de cod le-am folosit pentru a afla informații despre pixelii culorilor pentru care pragul a fost mai dificil de aflat.

```
43
44  % 3. Afișează imaginea HSV cu informații despre pixeli
45  figure;
46  imshow(hsv_img);
47  title('Imagine HSV - H (nuanță), S (saturație), V (luminozitate)');
48
49  % 4. Activează afișajul valorilor HSV sub cursor
50  impixelinfo;
```



În această imagine Pixel info contine informații despre culoare maro a formei stea.

Cojocariu Alexandra-Maria 1305B

Pasul 4

În continuare, într-un for care trece prin toate culorile din colorRanges , pentru fiecare culoare vom crea o mască binară pentru a segmenta pixelii din imagine care se încadrează în intervalele HSV definite pentru culoarea curentă. Masca este true (1) pentru pixelii care se află în toate cele trei intervale (Nuanță, Saturație, Valoare) și false (0) în caz contrar.

```
for i = 1:size(colorRanges,1)
name = colorRanges{i,1};
h_range = colorRanges{i,2};
s_range = colorRanges{i,3};
v_range = colorRanges{i,4};

% Masca pentru culoare
if h_range(1) > h_range(2)
    mask = (H >= h_range(1) | H <= h_range(2)) & ...
           (S >= s_range(1) & S <= s_range(2)) & ...
           (V >= v_range(1) & V <= v_range(2));
else
    mask = (H >= h_range(1) & H <= h_range(2)) & ...
           (S >= s_range(1) & S <= s_range(2)) & ...
           (V >= v_range(1) & V <= v_range(2));
end
```

Condiția h_range(1) > h_range(2) este specifică pentru culorile care se întind pe granița ciclului de nuanțe (cum ar fi roșul, care este la aproximativ 0 și 1).

După ce am creat masca binară, am filtrat-o și am umplut masca în locurile unde erau pixeli negri în loc de albi în interiorul conturului.

%mask = bwareaopen(mask, 300); Elimină regiunile conectate (obiectele) din masca binară care au mai puțin de 300 de pixeli. Aceasta ajută la eliminarea zgomotului și a obiectelor mici irelevante.

%imfill(mask, 'holes'); Umple găurile (regiunile de pixeli negri înconjurate de pixeli albi) din interiorul regiunilor albe din mască.

Pasul 5

La pasul acesta am început detecția contururilor obiectelor din masca binară.

[B, L] = bwboundaries(mask, 'noholes');:

B: Este o celulă array în care fiecare celulă conține coordonatele pixelilor de graniță pentru un obiect.

L: Este o matrice etichetată (label matrix) unde pixelii fiecărui obiect conectat au o etichetă unică.

Am calculat într-un o serie de proprietăți ale regiunilor etichetate în matricea L, printre acestea se numără: nuărul total de pixelidin regiune, lungimea conturului regiunii, proprietatea care determină cât de eliptică este regiunea(0 pentru cerc, 1 pentru linie).

```
% Detectare contururi
[B, L] = bwboundaries(mask, 'noholes');
stats = regionprops(L, 'Centroid', 'Area', 'Perimeter', ...
                    'BoundingBox', 'Eccentricity');

for k = 1:length(B)
    boundary = B{k};
    area = stats(k).Area;
    perimeter = stats(k).Perimeter;
    circularity = 4 * pi * area / (perimeter^2);
    centroid = stats(k).Centroid;
    ecc = stats(k).Eccentricity;
    bbox = stats(k).BoundingBox;
    aspect = bbox(3)/bbox(4);
```

Pasul 6

După ce am detectat conturul formei voi detecta numărul de vârfuri , calculând o mască binară doar pentru regiunea curentă.După ce detectez colțurile regiunii, cu ajutorul acestora și al circularității voi clasifica forma regiunii.

Pentru triunghi voi avea condițiile: să aibe trei colțuri și voi calcula și orientarea în sus sau în jos.

Pentru formele cu patru colțuri e mai dificil să precizez din prima ce formă este așa că voi calcula în plus măsura laturilor și unghiurile dintre laturi.

La pătrat: patru colțuri și unghiuri de 90 de grade.

La romb: laturi cu măsuri diferite și unghiuri diferite de 90 de grade.

La dreptunghi: laturi cu măsuri diferite și unghiuri de 90 de grade.

Daca regiunea are mai mult de 4 colțuri este probabil să fie stea.

După ce am pus condiții pentru formele cu colțuti , am luat cazurile în care figura este circulară: dacă are circularitate mare și excentricitatea mai mica de 0.6 este cerc, iar dacă circularitatea este mică și excentricitatea este mai mare sau egala cu 0.6 este elipsă.

Și ultimul caz dacă nu indeplinește nicio condiție atunci forma va rămâne etichetată cu Necunoscut.

```
    % Detectăm colțuri cu corner()
    regionMask = (L == k);
    corners = corner(regionMask, 'QualityLevel', 0.01, 'SensitivityFactor', 0.04);
    numCorners = size(corners,1);

    shape = 'Necunoscut';

    if numCorners == 3
        shape = 'Triunghi';
        [~, idxTop] = min(corners(:,2));
        topPoint = corners(idxTop, :);
        otherPoints = corners(setdiff(1:3, idxTop), :);
        if all(topPoint(2) < otherPoints(:,2))
            orientation = 'sus';
        else
            orientation = 'jos';
        end
        shape = [shape, ' ', orientation];

   elseif numCorners == 4
% Obținem colțurile într-un ordin mai stabil (prin convhull)
kHull = convhull(corners(:,1), corners(:,2));
ordered = corners(kHull(1:4), :);

% Calculează lungimile laturilor
sides = vecnorm(diff([ordered; ordered(1,:)]), 2, 2);

% Calculează unghiurile dintre laturi
```

```
% Calculează unghiurile dintre laturi
angles = zeros(4,1);
for j = 1:4
    v1 = ordered(mod(j-2,4)+1,:) - ordered(j,:);
    v2 = ordered(mod(j,4)+1,:) - ordered(j,:);
    cosAngle = dot(v1,v2) / (norm(v1)*norm(v2));
    angles(j) = acosd(cosAngle);
end

equalSides = std(sides) < 10;
rightAngles = all(abs(angles - 90) < 15);

if equalSides && rightAngles
    shape = 'Pătrat';
elseif ~equalSides && rightAngles
    shape = 'Dreptunghi';
elseif equalSides && ~rightAngles
    shape = 'Romb';
else
    shape = 'Patrulater';
end


    elseif numCorners >= 10 && circularity < 0.6
        shape = 'Stea';

    elseif circularity > 0.85 && ecc < 0.6
        shape = 'Cerc';

    elseif circularity > 0.6 && ecc >= 0.6
        shape = 'Elipsă';
    end
```

Pasul 7

După ce am determinat toate formele, voi desena conturul lor cu o linie gri pe imaginea care afiseaza rezultatul și voi adauga în centrul fiecărui obiect un text care să conțină numele formei si culoarea acesteia.

```
 % Afișare
plot(boundary(:,2), boundary(:,1), 'Color', [0.5 0.5 0.5], 'LineWidth', 2);
text(centroid(1), centroid(2), {shape; name}, 'Color', 'black','FontSize', 12, 'FontWeight', 'bold');
```